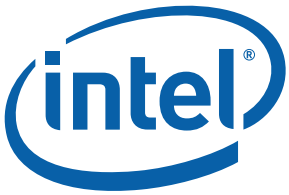


Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor

Developer's Manual

September 2006

Order Number: 252480-006US

Contents

1.0 Introduction 26
- 1.1 About This Document 26
  - 1.1.1 How to Read This Document 26
- 1.2 Other Relevant Documents 26
- 1.3 Terminology and Conventions 26
  - 1.3.1 Number Representation 26
  - 1.3.2 Acronyms and Terminology 27

2.0 Overview of Product Line 30
- 2.1 Intel XScale® Microarchitecture Processor 35
  - 2.1.1 Intel XScale® Processor Overview 36
    - 2.1.1.1 ARM* Compatibility 36
    - 2.1.1.2 Multiply/Accumulate (MAC) 36
    - 2.1.1.3 Memory Management 37
    - 2.1.1.4 Instruction Cache 37
    - 2.1.1.5 Branch Target Buffer 37
    - 2.1.1.6 Data Cache 37
    - 2.1.1.7 Intel XScale® Processor Performance Monitoring 38
- 2.2 Network Processor Engines (NPE) 38
- 2.3 Internal Bus 39
- 2.4 MII Interfaces 39
- 2.5 AHB Queue Manager 39
- 2.6 UTOPIA 2 40
- 2.7 USB v1.1 40
- 2.8 PCI 40
- 2.9 Memory Controller 40
- 2.10 Expansion Bus 41
- 2.11 High-Speed Serial Interfaces 41
- 2.12 Universal Asynchronous Receiver Transceiver 42
- 2.13 GPIO 42
- 2.14 Interrupt Controller 42
- 2.15 Timers 42
- 2.16 JTAG 43

3.0 Intel XScale® Processor 44
- 3.1 Memory Management Unit 44
  - 3.1.1 Memory Attributes 45
    - 3.1.1.1 Page (P) Attribute Bit 45
    - 3.1.1.2 Cacheable (C), Bufferable (B), and eXtension (X) Bits 45
  - 3.1.2 Interaction of the MMU, Instruction Cache, and Data Cache 47
  - 3.1.3 MMU Control 48
    - 3.1.3.1 Invalidate (Flush) Operation 48
    - 3.1.3.2 Enabling/Disabling 48
    - 3.1.3.3 Locking Entries 49
    - 3.1.3.4 Round-Robin Replacement Algorithm 51
- 3.2 Instruction Cache 52
  - 3.2.1 Operation When Instruction Cache is Enabled 52
    - 3.2.1.1 Instruction-Cache 'Miss' 53
    - 3.2.1.2 Instruction-Cache Line-Replacement Algorithm 54
    - 3.2.1.3 Instruction-Cache Coherence 55
- 3.3 Branch Target Buffer 58
  - 3.3.1 Branch Target Buffer (BTB) Operation 58
    - 3.3.1.1 Reset 59

3.4 Data Cache 60
3.4.1 Data Cache Overview 60
3.4.2 Cacheability 63
3.4.3 Reconfiguring the Data Cache as Data RAM 68
3.5 Configuration 73
3.5.1 CP15 Registers 75
3.5.1.1 Register 0: ID and Cache Type Registers 76
3.5.1.2 Register 1: Control and Auxiliary Control Registers 77
3.5.1.3 Register 2: Translation Table Base Register 79
3.5.1.4 Register 3: Domain Access Control Register 80
3.5.1.5 Register 4: Reserved 80
3.5.1.6 Register 5: Fault Status Register 80
3.5.1.7 Register 6: Fault Address Register 81
3.5.1.8 Register 7: Cache Functions 81
3.5.1.9 Register 8: TLB Operations 82
3.5.1.10 Register 9: Cache Lock Down 82
3.5.1.11 Register 10: TLB Lock Down 83
3.5.1.12 Register 11-12: Reserved 84
3.5.1.13 Register 13: Process ID 84
3.5.1.14 The PID Register Affect On Addresses 84
3.5.1.15 Register 14: Breakpoint Registers 85
3.5.1.16 Register 15: Coprocessor Access Register 85
3.5.2 CP14 Registers 86
3.5.2.1 Performance Monitoring Registers 87
3.5.2.2 Clock and Power Management Registers 87
3.5.2.3 Software Debug Registers 88
3.6 Software Debug 88
3.6.1 Definitions 89
3.6.2 Debug Registers 89
3.6.3 Debug Modes 89
3.6.3.1 Halt Mode 90
3.6.3.2 Monitor Mode 90
3.6.4 Debug Control and Status Register (DCSR) 90
3.6.4.1 Global Enable Bit (GE) 91
3.6.4.2 Halt Mode Bit (H) 91
3.6.4.3 Vector Trap Bits (TF,TI,TD,TA,TS,TU,TR) 92
3.6.4.4 Sticky Abort Bit (SA) 92
3.6.4.5 Method of Entry Bits (MOE) 92
3.6.4.6 Trace Buffer Mode Bit (M) 92
3.6.4.7 Trace Buffer Enable Bit (E) 92
3.6.5 Debug Exceptions 92
3.6.5.1 Halt Mode 93
3.6.5.2 Monitor Mode 94
3.6.6 HW Breakpoint Resources 95
3.6.6.1 Instruction Breakpoints 95
3.6.6.2 Data Breakpoints 96
3.6.7 Software Breakpoints 98
3.6.8 Transmit/Receive Control Register 98
3.6.8.1 RX Register Ready Bit (RR) 99
3.6.8.2 Overflow Flag (OV) 100
3.6.8.3 Download Flag (D) 100
3.6.8.4 TX Register Ready Bit (TR) 100
3.6.8.5 Conditional Execution Using TXRXCTRL 101
3.6.9 Transmit Register 101
3.6.10 Receive Register 102
3.6.11 Debug JTAG Access 102
3.6.11.1 SELDCSR JTAG Command 102

3.6.11.2 SELDCSR JTAG Register 103
3.6.11.3 DBGTX JTAG Command 105
3.6.11.4 DBGTX JTAG Register 105
3.6.11.5 DBGRX JTAG Command 106
3.6.11.6 DBGRX JTAG Register 106
3.6.11.7 Debug JTAG Data Register Reset Values 109
3.6.12 Trace Buffer 109
3.6.12.1 Trace Buffer CP Registers 109
3.6.13 Trace Buffer Entries 111
3.6.13.1 Message Byte 111
3.6.13.2 Trace Buffer Usage 114
3.6.14 Downloading Code in ICache 116
3.6.14.1 LDIC JTAG Command 116
3.6.14.2 LDIC JTAG Data Register 117
3.6.14.3 LDIC Cache Functions 118
3.6.14.4 Loading IC During Reset 119
3.6.14.5 Dynamically Loading IC After Reset 123
3.6.14.6 Mini-Instruction Cache Overview 126
3.6.15 Halt Mode Software Protocol 126
3.6.15.1 Starting a Debug Session 126
3.6.15.2 Implementing a Debug Handler 128
3.6.15.3 Ending a Debug Session 131
3.6.16 Software Debug Notes and Errata 132
3.7 Performance Monitoring 133
3.7.1 Overview 133
3.7.2 Register Description 134
3.7.2.1 Clock Counter (CCNT) 134
3.7.2.2 Performance Count Registers 134
3.7.2.3 Performance Monitor Control Register 135
3.7.2.4 Interrupt Enable Register 136
3.7.2.5 Overflow Flag Status Register 136
3.7.2.6 Event Select Register 137
3.7.3 Managing the Performance Monitor 138
3.7.4 Performance Monitoring Events 139
3.7.4.1 Instruction Cache Efficiency Mode 140
3.7.4.2 Data Cache Efficiency Mode 140
3.7.4.3 Instruction Fetch Latency Mode 140
3.7.4.4 Data/Bus Request Buffer Full Mode 141
3.7.4.5 Stall/Write-Back Statistics 141
3.7.4.6 Instruction TLB Efficiency Mode 142
3.7.4.7 Data TLB Efficiency Mode 142
3.7.5 Multiple Performance Monitoring Run Statistics 142
3.7.6 Examples 142
3.8 Programming Model 144
3.8.1 ARM* Architecture Compatibility 144
3.8.2 ARM* Architecture Implementation Options 144
3.8.2.1 Big Endian versus Little Endian 144
3.8.2.2 26-Bit Architecture 145
3.8.2.3 Thumb 145
3.8.2.4 ARM* DSP-Enhanced Instruction Set 145
3.8.2.5 Base Register Update 145
3.8.3 Extensions to ARM* Architecture 146
3.8.3.1 DSP Coprocessor 0 (CP0) 146
3.8.3.2 New Page Attributes 152
3.8.3.3 Additions to CP15 Functionality 153
3.8.3.4 Event Architecture 154
3.9 Performance Considerations 159

3.9.1 Interrupt Latency 159
3.9.2 Branch Prediction 160
3.9.3 Addressing Modes 160
3.9.4 Instruction Latencies 160
3.9.4.1 Performance Terms 160
3.9.4.2 Branch Instruction Timings 162
3.9.4.3 Data Processing Instruction Timings 162
3.9.4.4 Multiply Instruction Timings 163
3.9.4.5 Saturated Arithmetic Instructions 165
3.9.4.6 Status Register Access Instructions 165
3.9.4.7 Load/Store Instructions 165
3.9.4.8 Semaphore Instructions 166
3.9.4.9 Coprocessor Instructions 166
3.9.4.10 Miscellaneous Instruction Timing 167
3.9.4.11 Thumb Instructions 167
3.10 Optimization Guide 167
3.10.1 Introduction 167
3.10.1.1 About This Section 168
3.10.2 Processors' Pipeline 168
3.10.2.1 General Pipeline Characteristics 168
3.10.2.2 Instruction Flow Through the Pipeline 170
3.10.2.3 Main Execution Pipeline 171
3.10.2.4 Memory Pipeline 172
3.10.2.5 Multiply/Multiply Accumulate (MAC) Pipeline 173
3.10.3 Basic Optimizations 173
3.10.3.1 Conditional Instructions 173
3.10.3.2 Bit Field Manipulation 178
3.10.3.3 Optimizing the Use of Immediate Values 178
3.10.3.4 Optimizing Integer Multiply and Divide 178
3.10.3.5 Effective Use of Addressing Modes 179
3.10.4 Cache and Prefetch Optimizations 180
3.10.4.1 Instruction Cache 180
3.10.4.2 Data and Mini Cache 181
3.10.4.3 Cache Considerations 184
3.10.4.4 Prefetch Considerations 185
3.10.5 Instruction Scheduling 191
3.10.5.1 Scheduling Loads 191
3.10.5.2 Scheduling Data Processing Instructions 195
3.10.5.3 Scheduling Multiply Instructions 196
3.10.5.4 Scheduling SWP and SWPB Instructions 197
3.10.5.5 Scheduling the MRA and MAR Instructions (MRRC/MCRR) 197
3.10.5.6 Scheduling the MIA and MIAPH Instructions 198
3.10.5.7 Scheduling MRS and MSR Instructions 198
3.10.5.8 Scheduling CP15 Coprocessor Instructions 199
3.10.6 Optimizing C Libraries 199
3.10.7 Optimizations for Size 199
3.10.7.1 Space/Performance Trade Off 199

4.0 Network Processor Engines (NPE) 202

5.0 Internal Bus 204
5.1 Internal Bus Arbiters 204
5.1.1 Priority Mechanism 205
5.2 Memory Map 205

6.0 PCI Controller 208
6.1 PCI Controller Configured as Host 213
6.1.1 Example: Generating a PCI Configuration Write and Read 216
6.2 PCI Controller Configured as Option 218

6.3 Initializing PCI Controller Configuration and Status Registers for Data Transactions .. 219
6.3.1 Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register ... 220
6.3.2 Example: PCI Memory Base Address Register and South-AHB Translation 222
6.4 Initializing the PCI Controller Configuration Registers ... 222
6.5 PCI Controller South AHB Transactions ... 225
6.6 PCI Controller Functioning as Bus Initiator ... 226
6.6.1 PCI Byte Enables ... 226
6.6.2 Initiated Type-0 Read Transaction ... 227
6.6.3 Initiated Type-0 Write Transaction ... 227
6.6.4 Initiated Type-1 Read Transaction ... 228
6.6.5 Initiated Type-1 Write Transaction ... 229
6.6.6 Initiated Memory Read Transaction ... 229
6.6.7 Initiated Memory Write Transaction ... 230
6.6.8 Initiated I/O Read Transaction ... 231
6.6.9 Initiated I/O Write Transaction ... 231
6.6.10 Initiated Burst Memory Read Transaction ... 232
6.6.11 Initiated Burst Memory Write Transaction ... 233
6.7 PCI Controller Functioning as Bus Target ... 234
6.8 PCI Controller DMA Controller ... 234
6.8.1 AHB to PCI DMA Channel Operation ... 238
6.8.2 PCI to AHB DMA Channel Operation ... 238
6.9 PCI Controller Door Bell Register ... 239
6.10 PCI Controller Interrupts ... 240
6.10.1 PCI Interrupt Generation ... 240
6.10.2 Internal Interrupt Generation ... 240
6.11 PCI Controller Endian Control ... 241
6.12 PCI Controller Clock and Reset Generation ... 248
6.13 PCI RCOMP Circuitry ... 249
6.14 Register Descriptions ... 249
6.14.1 PCI Configuration Registers ... 249
6.14.1.1 Device ID/Vendor ID Register ... 250
6.14.1.2 Status Register/Control Register ... 250
6.14.1.3 Class Code/Revision ID Register ... 252
6.14.1.4 BIST/Header Type/Latency Timer/Cache Line Register ... 252
6.14.1.5 Base Address 0 Register ... 253
6.14.1.6 Base Address 1 Register ... 254
6.14.1.7 Base Address 2 Register ... 254
6.14.1.8 Base Address 3 Register ... 255
6.14.1.9 Base Address 4 Register ... 255
6.14.1.10 Base Address 5 Register ... 256
6.14.1.11 Subsystem ID/Subsystem Vendor ID Register ... 256
6.14.1.12 Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line Register ... 257
6.14.1.13 Retry Timeout/TRDY Timeout Register ... 257
6.14.2 PCI Controller Configuration and Status Registers ... 258
6.14.2.1 PCI Controller Non-pre-fetch Address Register ... 259
6.14.2.2 PCI Controller Non-pre-fetch Command/Byte Enables Register ... 259
6.14.2.3 PCI Controller Non-Pre-fetch Write Data Register ... 260
6.14.2.4 PCI Controller Non-Pre-fetch Read Data Register ... 260
6.14.2.5 PCI Controller Configuration Port Address/Command/ Byte Enables Register ... 260
6.14.2.6 PCI Controller Configuration Port Write Data Register ... 261
6.14.2.7 PCI Controller Configuration Port Read Data Register ... 262
6.14.2.8 PCI Controller Control and Status Register ... 262
6.14.2.9 PCI Controller Interrupt Status Register ... 263
6.14.2.10 PCI Controller Interrupt Enable Register ... 264

6.14.2.11DMA Control Register 265
6.14.2.12AHB Memory Base Address Register 266
6.14.2.13AHB I/O Base Address Register 266
6.14.2.14PCI Memory Base Address Register 267
6.14.2.15AHB Doorbell Register 267
6.14.2.16PCI Doorbell Register 268
6.14.2.17AHB to PCI DMA AHB Address Register 0 268
6.14.2.18AHB to PCI DMA PCI Address Register 0 269
6.14.2.19AHB to PCI DMA Length Register 0 269
6.14.2.20AHB to PCI DMA AHB Address Register 1 270
6.14.2.21AHB to PCI DMA PCI Address Register 1 270
6.14.2.22AHB to PCI DMA Length Register 1 270
6.14.2.23PCI to AHB DMA AHB Address Register 0 271
6.14.2.24PCI to AHB DMA PCI Address Register 0 271
6.14.2.25PCI to AHB DMA Length Register 0 272
6.14.2.26PCI to AHB DMA AHB Address Register 1 272
6.14.2.27PCI to AHB DMA PCI Address Register 1 273
6.14.2.28PCI to AHB DMA Length Register 1 273

7.0 SDRAM Controller 276
7.1 SDRAM Memory Space 279
7.2 Initializing the SDRAM Controller 279
7.2.1 Initializing the SDRAM 283
7.3 SDRAM Memory Accesses 285
7.3.1 Read Transfer 285
7.3.1.1 Read Cycle Timing (CAS Latency of Two Cycles) 285
7.3.1.2 Read Burst Transfer (Interleaved AHB Reads) 286
7.3.2 Write Transfer 286
7.3.2.1 Write Transfer 286
7.4 Register Description 287
7.4.1 Configuration Register 287
7.4.2 Refresh Register 288
7.4.3 Instruction Register 288

8.0 Expansion Bus Controller 292
8.1 Expansion Bus Address Space 293
8.2 Chip Select Address Allocation 294
8.3 Address and Data Byte Steering 295
8.4 Expansion Bus Connections 297
8.5 Expansion Bus Interface Configuration 298
8.6 Using I/O Wait 301
8.7 Special Design Knowledge for Using HPI mode 303
8.8 Expansion Bus Interface Access Timing Diagrams 305
8.8.1 Intel® Multiplexed-Mode Write Access 305
8.8.2 Intel® Multiplexed-Mode Read Access 306
8.8.3 Intel® Simplex-Mode Write Access 307
8.8.4 Intel® Simplex-Mode Read Access 308
8.8.5 Motorola* Multiplexed-Mode Write Access 309
8.8.6 Motorola* Multiplexed-Mode Read Access 310
8.8.7 Motorola* Simplex-Mode Write Access 311
8.8.8 Motorola* Simplex-Mode Read Access 312
8.8.9 TI* HPI-8 Write Access 313
8.8.10 TI* HPI-8 Read Access 314
8.8.11 TI* HPI-16, Multiplexed-Mode Write Access 315
8.8.12 TI* HPI-16, Multiplexed-Mode Read Access 316
8.8.13 TI* HPI-16 Simplex-Mode Write Access 317
8.8.14 TI* HPI-16 Simplex-Mode Read Access 318

8.9 Register Descriptions 319
8.9.1 Timing and Control Registers for Chip Select 0 319
8.9.2 Timing and Control Registers for Chip Select 1 319
8.9.3 Timing and Control Registers for Chip Select 2 320
8.9.4 Timing and Control Registers for Chip Select 3 320
8.9.5 Timing and Control Registers for Chip Select 4 320
8.9.6 Timing and Control Registers for Chip Select 5 321
8.9.7 Timing and Control Registers for Chip Select 6 321
8.9.8 Timing and Control Registers for Chip Select 7 321
8.9.9 Configuration Register 0 322
8.9.9.1 User-Configurable Field 324
8.9.10 Configuration Register 1 324
8.10 Expansion Bus Controller Performance 326

9.0 AHB/APB Bridge 328

10.0 Universal Asynchronous Receiver Transceiver (UART) 332
10.1 High Speed UART 333
10.2 Configuring the UART 335
10.2.1 Setting the Baud Rate 335
10.2.2 Setting Data Bits/Stop Bits/Parity 336
10.2.3 Using the Modem Control Signals 338
10.2.4 UART Interrupts 339
10.3 Transmitting and Receiving UART Data 342
10.4 Register Descriptions 344
10.4.1 Receive Buffer Register 345
10.4.2 Transmit Holding Register 345
10.4.3 Divisor Latch Low Register 346
10.4.4 Divisor Latch High Register 346
10.4.5 Interrupt Enable Register 346
10.4.6 Interrupt Identification Register 347
10.4.7 FIFO Control Register 349
10.4.8 Line Control Register 350
10.4.9 Modem Control Register 352
10.4.10 Line Status Register 353
10.4.11 Modem Status Register 354
10.4.12 Scratch-Pad Register 355
10.4.13 Infrared Selection Register 356
10.5 Console UART 357
10.5.1 Register Description 357
10.5.1.1 Receive Buffer Register 358
10.5.1.2 Transmit Holding Register 358
10.5.1.3 Divisor Latch Low Register 359
10.5.1.4 Divisor Latch High Register 359
10.5.1.5 Interrupt Enable Register 360
10.5.1.6 Interrupt Identification Register 360
10.5.1.7 FIFO Control Register 362
10.5.1.8 Line Control Register 363
10.5.1.9 Modem Control Register 365
10.5.1.10 Line Status Register 366
10.5.1.11 Modem Status Register 367
10.5.1.12 Scratch-Pad Register 368
10.5.1.13 Infrared Selection Register 369

11.0 Internal Bus Performance Monitoring Unit (IBPMU) 372
11.1 Initializing the IBPMU 372
11.2 Using the IBPMU 373

11.2.1 Monitored Events South AHB and North AHB 375
11.2.2 Monitored SDRAM Events 377
11.2.3 Cycle Count 377
11.3 Register Descriptions 378
11.3.1 Event Select Register 378
11.3.2 PMU Status Register (PSR) 381
11.3.3 Programmable Event Counters (PEC1) 381
11.3.4 Programmable Event Counters (PEC2) 382
11.3.5 Programmable Event Counters (PEC3) 382
11.3.6 Programmable Event Counters (PEC4) 382
11.3.7 Programmable Event Counters (PEC5) 383
11.3.8 Programmable Event Counters (PEC6) 383
11.3.9 Programmable Event Counters (PEC7) 384
11.3.10 Previous Master/Slave Register (PSMR) 384

12.0 General Purpose Input/Output (GPIO) 386
12.1 Using GPIO as Inputs/Outputs 386
12.2 Using GPIO as Interrupt Inputs 387
12.3 Using GPIO 14 and GPIO 15 as Clocks 389
12.4 Register Description 391
12.4.1 GPIO Output Register 391
12.4.2 GPIO Output Enable Register 392
12.4.3 GPIO Input Register 392
12.4.4 GPIO Interrupt Status Register 393
12.4.5 GP Interrupt Type Register 1 393
12.4.6 GPIO Interrupt Type Register 2 394
12.4.7 GPIO Clock Register 395

13.0 Interrupt Controller 398
13.1 Interrupt Priority 398
13.2 Assigning FIQ or IRQ Interrupts 399
13.3 Enabling and Disabling Interrupts 399
13.4 Reading Interrupt Status 400
13.5 Interrupt Controller Register Description 401
13.5.1 Interrupt Status Register 402
13.5.2 Interrupt-Enable Register 404
13.5.3 Interrupt Select Register 404
13.5.4 IRQ Status Register 404
13.5.5 FIQ Status Register 404
13.5.6 Interrupt Priority Register 405
13.5.7 IRQ Highest-Priority Register 405
13.5.8 FIQ Highest-Priority Register 406

14.0 Timers 408
14.1 Watch-Dog Timer 408
14.2 Time-Stamp Timer 409
14.3 General-Purpose Timers 409
14.4 Timer Register Definition 411
14.4.1 Time-Stamp Timer 411
14.4.2 General-Purpose Timer 0 411
14.4.3 General-Purpose Timer 0 Reload 412
14.4.4 General-Purpose Timer 1 412
14.4.5 General-Purpose Timer 1 Reload 413
14.4.6 Watch-Dog Timer 413
14.4.7 Watch-Dog Enable Register 414
14.4.8 Watch-Dog Key Register 414

14.4.9 Timer Status 415

15.0 Ethernet MAC A 416

15.1 Ethernet Coprocessor 417
15.1.1 Ethernet Coprocessor APB Interface 418
15.1.2 Ethernet Coprocessor NPE Interface 418
15.1.3 Ethernet Coprocessor MDIO Interface 418
15.1.4 Transmitting Ethernet Frames with MII Interfaces 420
15.1.5 Receiving Ethernet Frames with MII Interfaces 423
15.1.6 General Ethernet Coprocessor Configuration 425

15.2 Register Descriptions 427
15.2.1 Transmit Control 1 428
15.2.2 Transmit Control 2 429
15.2.3 Receive Control 1 429
15.2.4 Receive Control 2 430
15.2.5 Random Seed 430
15.2.6 Threshold For Partially Empty 431
15.2.7 Threshold For Partially Full 431
15.2.8 Buffer Size For Transmit 431
15.2.9 Transmit Deferral Parameters 432
15.2.10 Receive Deferral Parameters 432
15.2.11 Transmit Two Part Deferral Parameters 1 433
15.2.12 Transmit Two Part Deferral Parameters 2 433
15.2.13 Slot Time 433
15.2.14 MDIO Commands Registers 434
15.2.15 MDIO Command 1 434
15.2.16 MDIO Command 2 434
15.2.17 MDIO Command 3 435
15.2.18 MDIO Command 4 435
15.2.19 MDIO Status Registers 435
15.2.20 MDIO Status 1 436
15.2.21 MDIO Status 2 436
15.2.22 MDIO Status 3 436
15.2.23 MDIO Status 4 436
15.2.24 Address Mask Registers 437
15.2.25 Address Mask 1 437
15.2.26 Address Mask 2 438
15.2.27 Address Mask 3 438
15.2.28 Address Mask 4 438
15.2.29 Address Mask 5 438
15.2.30 Address Mask 6 439
15.2.31 Address Registers 439
15.2.32 Address 1 440
15.2.33 Address 2 440
15.2.34 Address 3 440
15.2.35 Address 4 440
15.2.36 Address 5 441
15.2.37 Address 6 441
15.2.38 Threshold for Internal Clock 442
15.2.39 Unicast Address Registers 442
15.2.40 Unicast Address 1 443
15.2.41 Unicast Address 2 443
15.2.42 Unicast Address 3 443
15.2.43 Unicast Address 4 443
15.2.44 Unicast Address 5 444

15.2.45 Unicast Address 6 444
15.2.46 Core Control 444

16.0 Ethernet MAC B 446

17.0 High-Speed Serial Interfaces 448
17.1 High-Speed Serial Interface Receive Operation 448
17.2 High-Speed Serial Interface Transmit Operation 449
17.3 Configuration of the High-Speed Serial Interface 450
17.4 Obtaining High-Speed, Serial Synchronization 453
17.5 HSS Registers and Clock Configuration 454
17.5.1 HSS Clock and Jitter 455
17.5.2 Overview of HSS Clock Configuration 455
17.6 HSS Supported Framing Protocols 457
17.6.1 T1 457
17.6.2 E1 459
17.6.3 MVIP 460
17.6.3.1 MVIP using 2.048Mbps Backplane 461
17.6.3.2 MVIP Using 4.096-Mbps Backplane 463
17.6.3.3 MVIP Using 8.192-Mbps Backplane 464

18.0 Universal Serial Bus (USB) v1.1 Device Controller 468
18.1 USB Overview 468
18.2 Device Configuration 469
18.3 USB Operation 470
18.3.1 Signalling Levels 470
18.3.2 Bit Encoding 471
18.3.3 Field Formats 472
18.3.4 Packet Formats 473
18.3.4.1 Token Packet Type 474
18.3.4.2 Start-of-Frame Packet Type 474
18.3.4.3 Data Packet Type 474
18.3.4.4 Handshake Packet Type 475
18.3.5 Transaction Formats 475
18.3.5.1 Bulk Transaction Type 475
18.3.5.2 Isochronous Transaction Type 476
18.3.5.3 Control Transaction Type 476
18.3.5.4 Interrupt Transaction Type 477
18.3.6 UDC Device Requests 477
18.3.7 UDC Configuration 478
18.4 UDC Hardware Connections 479
18.4.1 Self-Powered Device 479
18.4.2 Bus-Powered Devices 479
18.5 Register Descriptions 479
18.5.1 UDC Control Register (UDCCR) 481
18.5.1.1 UDC Enable 481
18.5.1.2 UDC Active 481
18.5.1.3 UDC Resume (RSM) 481
18.5.1.4 Resume Interrupt Request (RESIR) 481
18.5.1.5 Suspend Interrupt Request (SUSIR) 482
18.5.1.6 Suspend/Resume Interrupt Mask (SRM) 482
18.5.1.7 Reset Interrupt Request (RSTIR) 482
18.5.1.8 Reset Interrupt Mask (REM) 482
18.5.2 UDC Endpoint 0 Control/Status Register (UDCCS0) 483
18.5.2.1 OUT Packet Ready (OPR) 483
18.5.2.2 IN Packet Ready (IPR) 483
18.5.2.3 Flush Tx FIFO (FTF) 484
18.5.2.4 Device Remote Wake-Up Feature (DRWF) 484

18.5.2.5 Sent Stall (SST) 484
18.5.2.6 Force Stall (FST) 484
18.5.2.7 Receive FIFO Not Empty (RNE) 484
18.5.2.8 Setup Active (SA) 484
18.5.3 UDC Endpoint 1 Control/Status Register (UDCCS1) 485
18.5.3.1 Transmit FIFO Service (TFS) 485
18.5.3.2 Transmit Packet Complete (TPC) 486
18.5.3.3 Flush Tx FIFO (FTF) 486
18.5.3.4 Transmit Underrun (TUR) 486
18.5.3.5 Sent STALL (SST) 486
18.5.3.6 Force STALL (FST) 486
18.5.3.7 Bit 6 Reserved 487
18.5.3.8 Transmit Short Packet (TSP) 487
18.5.4 UDC Endpoint 2 Control/Status Register (UDCCS2) 487
18.5.4.1 Receive FIFO Service (RFS) 488
18.5.4.2 Receive Packet Complete (RPC) 488
18.5.4.3 Bit 2 Reserved 488
18.5.4.4 Bit 2 Reserved 488
18.5.4.5 Sent Stall (SST) 488
18.5.4.6 Force Stall (FST) 488
18.5.4.7 Receive FIFO Not Empty (RNE) 488
18.5.4.8 Receive Short Packet (RSP) 489
18.5.5 UDC Endpoint 3 Control/Status Register (UDCCS3) 490
18.5.5.1 Transmit FIFO Service (TFS) 490
18.5.5.2 Transmit Packet Complete (TPC) 490
18.5.5.3 Flush Tx FIFO (FTF) 490
18.5.5.4 Transmit Underrun (TUR) 490
18.5.5.5 Bit 4 Reserved 490
18.5.5.6 Bit 5 Reserved 490
18.5.5.7 Bit 6 Reserved 490
18.5.5.8 Transmit Short Packet (TSP) 491
18.5.6 UDC Endpoint 4 Control/Status Register (UDCCS4) 491
18.5.6.1 Receive FIFO Service (RFS) 491
18.5.6.2 Receive Packet Complete (RPC) 492
18.5.6.3 Receive Overflow (ROF) 492
18.5.6.4 Bit 3 Reserved 492
18.5.6.5 Bit 4 Reserved 492
18.5.6.6 Bit 5 Reserved 492
18.5.6.7 Receive FIFO Not Empty (RNE) 492
18.5.6.8 Receive Short Packet (RSP) 492
18.5.7 UDC Endpoint 5 Control/Status Register (UDCCS5) 493
18.5.7.1 Transmit FIFO Service (TFS) 493
18.5.7.2 Transmit Packet Complete (TPC) 493
18.5.7.3 Flush Tx FIFO (FTF) 494
18.5.7.4 Transmit Underrun (TUR) 494
18.5.7.5 Sent STALL (SST) 494
18.5.7.6 Force STALL (FST) 494
18.5.7.7 Bit 6 Reserved 494
18.5.7.8 Transmit Short Packet (TSP) 495
18.5.8 UDC Endpoint 6 Control/Status Register 495
18.5.8.1 Transmit FIFO Service (TFS) 496
18.5.8.2 Transmit Packet Complete (TPC) 496
18.5.8.3 Flush Tx FIFO (FTF) 496
18.5.8.4 Transmit Underrun (TUR) 496
18.5.8.5 Sent STALL (SST) 496
18.5.8.6 Force STALL (FST) 496
18.5.8.7 Bit 6 Reserved 497
18.5.8.8 Transmit Short Packet (TSP) 497
18.5.9 UDC Endpoint 7 Control/Status Register (UDCCS7) 498

18.5.9.1 Receive FIFO Service (RFS) 498
18.5.9.2 Receive Packet Complete (RPC) 498
18.5.9.3 Bit 2 Reserved 498
18.5.9.4 Bit 3 Reserved 498
18.5.9.5 Sent Stall (SST) 498
18.5.9.6 Force Stall (FST) 498
18.5.9.7 Receive FIFO Not Empty (RNE) 499
18.5.9.8 Receive Short Packet (RSP) 499
18.5.10 UDC Endpoint 8 Control/Status Register (UDCCS8) 500
18.5.10.1 Transmit FIFO Service (TFS) 500
18.5.10.2 Transmit Packet Complete (TPC) 500
18.5.10.3 Flush Tx FIFO (FTF) 500
18.5.10.4 Transmit Underrun (TUR) 500
18.5.10.5 Bit 4 Reserved 500
18.5.10.6 Bit 5 Reserved 501
18.5.10.7 Bit 6 Reserved 501
18.5.10.8 Transmit Short Packet (TSP) 501
18.5.11 UDC Endpoint 9 Control/Status Register (UDCCS9) 502
18.5.11.1 Receive FIFO Service (RFS) 502
18.5.11.2 Receive Packet Complete (RPC) 502
18.5.11.3 Receive Overflow (ROF) 502
18.5.11.4 Bit 3 Reserved 502
18.5.11.5 Bit 4 Reserved 502
18.5.11.6 Bit 5 Reserved 502
18.5.11.7 Receive FIFO Not Empty (RNE) 502
18.5.11.8 Receive Short Packet (RSP) 502
18.5.12 UDC Endpoint 10 Control/Status Register (UDCCS10) 503
18.5.12.1 Transmit FIFO Service (TFS) 503
18.5.12.2 Transmit Packet Complete (TPC) 503
18.5.12.3 Flush Tx FIFO (FTF) 504
18.5.12.4 Transmit Underrun (TUR) 504
18.5.12.5 Sent STALL (SST) 504
18.5.12.6 Force STALL (FST) 504
18.5.12.7 Bit 6 Reserved 504
18.5.12.8 Transmit Short Packet (TSP) 505
18.5.13 UDC End point 11 Control/Status Register (UDCCS11) 505
18.5.13.1 Transmit FIFO Service (TFS) 506
18.5.13.2 Transmit Packet Complete (TPC) 506
18.5.13.3 Flush Tx FIFO (FTF) 506
18.5.13.4 Transmit Underrun (TUR) 506
18.5.13.5 Sent STALL (SST) 506
18.5.13.6 Force STALL (FST) 506
18.5.13.7 Bit 6 Reserved 507
18.5.13.8 Transmit Short Packet (TSP) 507
18.5.14 UDC Endpoint 12 Control/Status Register (UDCCS12) 508
18.5.14.1 Receive FIFO Service (RFS) 508
18.5.14.2 Receive Packet Complete (RPC) 508
18.5.14.3 Bit 2 Reserved 508
18.5.14.4 Bit 3 Reserved 508
18.5.14.5 Sent Stall (SST) 508
18.5.14.6 Force Stall (FST) 509
18.5.14.7 Receive FIFO Not Empty (RNE) 509
18.5.14.8 Receive Short Packet (RSP) 509
18.5.15 UDC Endpoint 13 Control/Status Register (UDCCS13) 510
18.5.15.1 Transmit FIFO Service (TFS) 510
18.5.15.2 Transmit Packet Complete (TPC) 510
18.5.15.3 Flush Tx FIFO (FTF) 510
18.5.15.4 Transmit Underrun (TUR) 511
18.5.15.5 Bit 4 Reserved 511

18.5.15.6 Bit 5 Reserved 511
18.5.15.7 Bit 6 Reserved 511
18.5.15.8 Transmit Short Packet (TSP) 511
18.5.16 UDC Endpoint 14 Control/Status Register (UDCCS14) 512
18.5.16.1 Receive FIFO Service (RFS) 512
18.5.16.2 Receive Packet Complete (RPC) 512
18.5.16.3 Receive Overflow (ROF) 512
18.5.16.4 Bit 3 Reserved 512
18.5.16.5 Bit 4 Reserved 512
18.5.16.6 Bit 5 Reserved 512
18.5.16.7 Receive FIFO Not Empty (RNE) 513
18.5.16.8 Receive Short Packet (RSP) 513
18.5.17 UDC Endpoint 15 Control/Status Register (UDCCS15) 514
18.5.17.1 Transmit FIFO Service (TFS) 514
18.5.17.2 Transmit Packet Complete (TPC) 514
18.5.17.3 Flush Tx FIFO (FTF) 514
18.5.17.4 Transmit Underrun (TUR) 514
18.5.17.5 Sent STALL (SST) 515
18.5.17.6 Force STALL (FST) 515
18.5.17.7 Bit 6 Reserved 515
18.5.17.8 Transmit Short Packet (TSP) 515
18.5.18 UDC Interrupt Control Register 0 (UICR0) 516
18.5.18.1 Interrupt Mask Endpoint x (IMx), Where x is 0 through 7 516
18.5.19 UDC Interrupt Control Register 1 (UICR1) 517
18.5.19.1 Interrupt Mask Endpoint x (IMx), where x is 8 through 15. 517
18.5.20 UDC Status/Interrupt Register 0 (UISR0) 518
18.5.20.1 Endpoint 0 Interrupt Request (IR0) 519
18.5.20.2 Endpoint 1 Interrupt Request (IR1) 519
18.5.20.3 Endpoint 2 Interrupt Request (IR2) 519
18.5.20.4 Endpoint 3 Interrupt Request (IR3) 519
18.5.20.5 Endpoint 4 Interrupt Request (IR4) 519
18.5.20.6 Endpoint 5 Interrupt Request (IR5) 519
18.5.20.7 Endpoint 6 Interrupt Request (IR6) 520
18.5.20.8 Endpoint 7 Interrupt Request (IR7) 520
18.5.21 UDC Status/Interrupt Register 1 (USIR1) 521
18.5.21.1 Endpoint 8 Interrupt Request (IR8) 521
18.5.21.2 Endpoint 9 Interrupt Request (IR9) 521
18.5.21.3 Endpoint 10 Interrupt Request (IR10) 521
18.5.21.4 Endpoint 11 Interrupt Request (IR11) 521
18.5.21.5 Endpoint 12 Interrupt Request (IR12) 521
18.5.21.6 Endpoint 13 Interrupt Request (IR13) 521
18.5.21.7 Endpoint 14 Interrupt Request (IR14) 521
18.5.21.8 Endpoint 15 Interrupt Request (IR15) 522
18.5.22 UDC Frame Number High Register (UFNHR) 522
18.5.22.1 UDC Frame Number MSB (FNMSB) 522
18.5.22.2 Isochronous Packet Error Endpoint 4 (IPE4) 523
18.5.22.3 Isochronous Packet Error Endpoint 9 (IPE9) 523
18.5.22.4 Isochronous Packet Error Endpoint 14 (IPE14) 523
18.5.22.5 Start of Frame Interrupt Mask (SIM) 523
18.5.22.6 Start of Frame Interrupt Request (SIR) 523
18.5.23 UDC Frame Number Low Register (UFNLR) 524
18.5.24 UDC Byte Count Register 2 (UBCR2) 524
18.5.24.1 Endpoint 2 Byte Count (BC[7:0]) 525
18.5.25 UDC Byte Count Register 4 (UBCR4) 525
18.5.25.1 Endpoint 4 Byte Count (BC[7:0]) 525
18.5.26 UDC Byte Count Register 7 (UBCR7) 526
18.5.26.1 Endpoint 7 Byte Count (BC[7:0]) 526
18.5.27 UDC Byte Count Register 9 (UBCR9) 526

18.5.27.1 Endpoint 9 Byte Count (BC[7:0]) 526
18.5.28 UDC Byte Count Register 12 (UBCR12) 527
18.5.28.1 Endpoint 12 Byte Count (BC[7:0]) 527
18.5.29 UDC Byte Count Register 14 (UBCR14) 528
18.5.29.1 Endpoint 14 Byte Count (BC[7:0]) 528
18.5.30 UDC Endpoint 0 Data Register (UDDR0) 528
18.5.31 UDC Data Register 1 (UDDR1) 529
18.5.32 UDC Data Register 2 (UDDR2) 529
18.5.33 UDC Data Register 3 (UDDR3) 530
18.5.34 UDC Data Register 4 (UDDR4) 531
18.5.35 UDC Data Register 5 (UDDR5) 531
18.5.36 UDC Data Register 6 (UDDR6) 532
18.5.37 UDC Data Register 7 (UDDR7) 532
18.5.38 UDC Data Register 8 (UDDR8) 533
18.5.39 UDC Data Register 9 (UDDR9) 533
18.5.40 UDC Data Register 10 (UDDR10) 534
18.5.41 UDC Data Register 11 535
18.5.42 UDC Data Register 12 (UDDR12) 535
18.5.43 UDC Data Register 13 (UDDR13) 536
18.5.44 UDC Data Register 14 (UDDR14) 536
18.5.45 UDC Data Register 15 (UDDR15) 537

19.0 UTOPIA Level-2 538
19.1 UTOPIA Transmit Module 540
19.2 UTOPIA Receive Module 543
19.3 UTOPIA-2 Coprocessor / NPE Coprocessor: Bus Interface 545
19.4 MPHY Polling Routines 546
19.5 UTOPIA Level-2 Clocks 546

20.0 JTAG Interface 548
20.1 TAP Controller 548
20.1.1 Test-Logic-Reset State 549
20.1.2 Run-Test/Idle State 550
20.1.3 Select-DR-Scan State 550
20.1.4 Capture-DR State 550
20.1.5 Shift-DR State 550
20.1.6 Exit1-DR State 551
20.1.7 Pause-DR State 551
20.1.8 Exit2-DR State 551
20.1.9 Update-DR State 551
20.1.10 Select-IR-Scan State 552
20.1.11 Capture-IR State 552
20.1.12 Shift-IR State 552
20.1.13 Exit1-IR State 552
20.1.14 Pause-IR State 552
20.1.15 Exit2-IR State 553
20.1.16 Update-IR State 553
20.2 JTAG Instructions 553
20.3 Data Registers 554
20.3.1 Boundary Scan Register 555
20.3.2 Instruction Register 555
20.3.3 JTAG Device ID Register 555

21.0 AHB Queue Manager (AQM) 556
21.1 Overview 556
21.2 Feature List 556

21.3 Functional Description 557
21.4 AHB Interface 558
21.4.1 Queue Control 559
21.4.2 Queue Status 560
21.4.2.1 Status Update 560
21.4.2.2 Flag Bus 561
21.4.2.3 Status Interrupts 562
21.5 Register Descriptions 562
21.5.1 Queue Access Word Registers 0 - 63 562
21.5.2 Queues 0-31 Status Register 0 - 3 563
21.5.3 Underflow/Overflow Status Register 0 - 1 563
21.5.4 Queues 32-63 Nearly Empty Status Register 564
21.5.5 Queues 32-63 Full Status Register 564
21.5.6 Interrupt 0 Status Flag Source Select Register 0 – 3 565
21.5.7 Queue Interrupt Enable Register 0 – 1 566
21.5.8 Queue Interrupt Register 0 – 1 566
21.5.9 Queue Configuration Words 0 - 63 566

Figures

1 Intel® IXP425 Network Processor Block Diagram 31
2 Intel® IXP423 Network Processor Block Diagram 32
3 Intel® IXP422 Network Processor Block Diagram 33
4 Intel® IXP421 Network Processor Block Diagram 34
5 Intel® IXP420 Network Processor and Intel® IXC1100 Control Plane Processor Block Diagram 35
6 Intel XScale® Technology Architecture Features 36
7 Example of Locked Entries in TLB 52
8 Instruction Cache Organization 53
9 Locked Line Effect on Round-Robin Replacement 57
10 BTB Entry 59
11 Branch History 59
12 Data Cache Organization 61
13 Mini-Data Cache Organization 62
14 Locked Line Effect on Round-Robin Replacement 72
15 SELDCSR Hardware 103
16 SELDCSR Data Register 104
17 DBGTX Hardware 105
18 DBGRX Hardware 106
19 Rx Write Logic 107
20 DBGRX Data Register 108
21 Message Byte Formats 111
22 Indirect Branch Entry Address Byte Organization 114
23 High Level View of Trace Buffer 114
24 LDIC JTAG Data Register Hardware 117
25 Format of LDIC Cache Functions 119
26 Code Download During a Cold Reset For Debug 120
27 Code Download During a Warm Reset For Debug 122
28 Downloading Code in IC During Program Execution 123
29 Processors' RISC Super-Pipeline 169
30 Processors' PCI Bus Configured as a Host 209
31 Processors' PCI Bus Configured as an Option 209
32 Processors' PCI Controller Block Diagram 210
33 Type 0 Configuration Address Phase 214
34 Type 1 Configuration Address Phase 215

35 Initiated PCI TYPE 0 Configuration Read Cycle 227
36 Initiated PCI Type-0 Configuration Write Cycle 228
37 Initiated PCI Type-1 Configuration Read Cycle 228
38 Initiated PCI Type-1 Configuration Write Cycle 229
39 Initiated PCI Memory Read Cycle 230
40 Initiated PCI Memory Write Cycle 231
41 Initiated PCI I/O Read Cycle 231
42 Initiated PCI I/O Write Cycle 232
43 Initiated PCI Burst Memory Read Cycle 233
44 Initiated PCI Burst Memory Write Cycle 234
45 AHB to PCI DMA Transfer Byte Lane Swapping 236
46 PCI to AHB DMA Transfer Byte Lane Swapping 236
47 Byte Lane Routing During PCI Target Accesses of the AHB – AHB Configured as a Big-Endian Bus 243
48 Byte Lane Routing During PCI Target Accesses of the AHB – AHB Configured as a Little-Endian Bus 244
49 Byte Lane Routing During AHB Memory Mapped Accesses of the PCI Bus – AHB Configured as a Big-Endian Bus 245
50 Byte Lane Routing During AHB Memory Mapped Accesses of the PCI Bus – AHB configured as a Little-Endian Bus 246
51 Byte Lane Routing During DMA Transfers 247
52 Byte Lane Routing During Configuration and Status Register Accesses 248
53 8-, 16-, 32-, 64- or 128-Mbyte — One-Bank SDRAM Interface Configuration 277
54 64-, 128- or 256-Mbyte — Two-Bank SDRAM Interface Configuration 278
55 SDRAM Read Example (CAS Latency of 2 Cycles) 285
56 SDRAM Shared South AHB and North AHB Access 286
57 SDRAM Write Example 287
58 Chip Select Address Allocation 295
59 Expansion Bus Memory Sizing 295
60 Expansion Bus Peripheral Connection 297
61 I/O Wait Normal Phase Timing 302
62 I/O Wait Extended Phase Timing 303
63 Expansion-Bus Write (Intel® Multiplexed Mode) 305
64 Expansion-Bus Read (Intel® Multiplexed Mode) 306
65 Expansion-Bus Write (Intel® Simplex Write Mode) 307
66 Expansion-Bus Read (Intel® Simplex Mode) 308
67 Expansion-Bus Write (Motorola* Multiplexed Mode) 309
68 Expansion-Bus Read (Motorola* Multiplexed Mode) 310
69 Expansion-Bus Write (Motorola* Simplex Mode) 311
70 Expansion-Bus Read (Motorola* Simplex Mode) 312
71 Expansion-Bus Write (TI* HPI-8 Mode) 313
72 Expansion-Bus Read (TI* HPI-8 Mode) 314
73 Expansion-Bus Write (TI* HPI-16 Multiplexed Mode) 315
74 Expansion-Bus Read (TI* HPI-16 Multiplexed Mode) 316
75 Expansion-Bus Write (TI* HPI-16 Simplex Mode) 317
76 Expansion-Bus Read (TI* HPI-16 Simplex Mode) 318
77 APB Interface 329
78 UART Timing Diagram 333
79 UART Block Diagram 334
80 Multiple Ethernet PHYS Connected to Processor 407
81 Ethernet Coprocessor Interface 407
82 MDIO Write 410
83 MDIO Read 410
84 Tx Frame-Sync Example (Presuming an Offset of 0) 444
85 Rx Frame-Sync Example (Presuming Zero Offset) 444

86 T1 Transmit Frame 448
87 T1 Receive Frame 448
88 E1 Transmit Frame 449
89 E1 Receive Frame 450
90 MVIP, Interleaved Mapping of a T1 Frame to an E1 Frame 452
91 MVIP, Frame Mapping a T1 Frame to an E1 Frame 453
92 MVIP, Byte Interlacing Two E1 Streams Onto a 4.096-Mbps Backplane 453
93 MVIP, Byte Interleaving Two T1 Streams Onto a 4.096-Mbps Backplane 454
94 MVIP, Byte Interleaving Four E1 Streams on a 8.192-Mbps Backplane Bus 455
95 MVIP, Byte Interleaving Four T1 Streams on a 8.192-Mbps Backplane Bus 455
96 NRZI Bit Encoding Example 462
97 UTOPIA Level-2 Coprocessor 530
98 UTOPIA Level-2 MPHY Transmit Polling 532
99 UTOPIA Level-2 MPHY Receive Polling 535
100 TAP Controller State Diagram 539
101 AHB Queue Manager 547

Tables

1 Acronyms and Terminology 27
2 Network Processor Functions 38
3 Data Cache and Buffer Behavior When X = 0 46
4 Data Cache and Buffer Behavior When X = 1 47
5 Memory Operations that Impose a Fence 47
6 Valid MMU and Data/Mini-Data Cache Combinations 48
7 MRC/MCR Format 74
8 LDC/STC Format when Accessing CP14 75
9 CP15 Registers 75
10 ID Register 76
11 Cache Type Register 77
12 ARM* Control Register 77
13 Auxiliary Control Register 79
14 Translation Table Base Register 79
15 Domain Access Control Register 80
16 Fault Status Register 80
17 Fault Address Register 81
18 Cache Functions 81
19 TLB Functions 82
20 Cache Lock-Down Functions 83
21 Data Cache Lock Register 83
22 TLB Lockdown Functions 83
23 Accessing Process ID 84
24 Process ID Register 84
25 Accessing the Debug Registers 85
26 Coprocessor Access Register 86
27 CP14 Registers 86
28 Accessing the Performance Monitoring Registers 87
29 PWRMODE Register 87
30 Clock and Power Management 88
31 CCLKCFG Register 88
32 Accessing the Debug Registers 88
33 Debug Control and Status Register (DCSR) 90
34 Event Priority 93
35 Instruction Breakpoint Address and Control Register (IBCRx) 96
36 Data Breakpoint Register (DBRx) 96
37 Data Breakpoint Controls Register (DBCON) 97

38 TX RX Control Register (TXRXCTRL) 98
39 Normal RX Handshaking 99
40 High-Speed Download Handshaking States 99
41 TX Handshaking 100
42 TXRXCTRL Mnemonic Extensions 101
43 TX Register 101
44 RX Register 102
45 DEBUG Data Register Reset Values 109
46 CP 14 Trace Buffer Register Summary 110
47 Checkpoint Register (CHKPTx) 110
48 TBREG Format 111
49 Message Byte Formats 112
50 LDIC Cache Functions 118
51 Debug-Handler Code to Implement Synchronization During Dynamic Code Download 125
52 Debug Handler Code: Download Bit and Overflow Flag 131
53 Performance Monitoring Registers 133
54 Clock Count Register (CCNT) 134
55 Performance Monitor Count Register (PMN0 - PMN3) 135
56 Performance Monitor Control Register 135
57 Interrupt Enable Register 136
58 Overflow Flag Status Register 137
59 Event Select Register 138
60 Performance Monitoring Events 139
61 Common Uses of the PMU 139
62 Multiply with Internal Accumulate Format 147
63 MIA{<cond>} acc0, Rm, Rs 147
64 MIAPH{<cond>} acc0, Rm, Rs 148
65 MIAxy{<cond>} acc0, Rm, Rs 148
66 Internal Accumulator Access Format 150
67 MAR{<cond>} acc0, RdLo, RdHi 151
68 MRA{<cond>} RdLo, RdHi, acc0 151
70 Second-Level Descriptors for Coarse Page Table 153
71 Second-Level Descriptors for Fine Page Table 153
69 First-Level Descriptors 153
72 Exception Summary 154
73 Event Priority 155
74 Processors' Encoding of Fault Status for Prefetch Aborts 156
75 Intel XScale® Processor Encoding of Fault Status for Data Aborts 156
76 Branch Latency Penalty 160
77 Latency Example 162
78 Branch Instruction Timings (Those Predicted by the BTB) 162
79 Branch Instruction Timings (Those not Predicted by the BTB) 162
80 Data Processing Instruction Timings 162
81 Multiply Instruction Timings 163
82 Multiply Implicit Accumulate Instruction Timings 165
83 Implicit Accumulator Access Instruction Timings 165
84 Saturated Data Processing Instruction Timings 165
85 Status Register Access Instruction Timings 165
86 Load and Store Instruction Timings 165
87 Load and Store Multiple Instruction Timings 166
88 Semaphore Instruction Timings 166
89 CP15 Register Access Instruction Timings 166
90 CP14 Register Access Instruction Timings 166
91 Exception-Generating Instruction Timings 167
92 Count Leading Zeros Instruction Timings 167

93 Pipelines and Pipe Stages 169
94 Network Processor Functions 202
95 Bus Arbitration Example: Three Requesting Masters 205
96 Memory Map 206
97 PCI Target Interface Supported Commands 211
98 PCI Initiator Interface-Supported Commands 212
99 PCI Memory Map Allocation 221
100 PCI Byte Enables Using CRP Access Method 224
101 PCI Configuration Space 225
102 Command Type for PCI Controller Configuration and Status Register Accesses 225
103 PCI Configuration Register Map 249
104 PCI Controller CSR Address Map 258
105 Supported Configuration of the SDRAM Controller 278
106 Memory Space 279
107 Memory Configurations for Writing the SDRAM Configuration (SDR_CONFIG) Register 280
108 Memory Configurations for Writing the SDRAM Configuration (SDR_CONFIG) Register 280
109 SDRAM Command Description 281
110 SDRAM I/O For Various Commands 282
111 Page Register Allocation 284
112 Data Transfer Sizes of AHB 285
113 SDRAM Register Overview 287
114 SDRAM Configuration Options 289
115 SDRAM Burst Definitions 289
116 SDRAM Commands 290
117 Processors' Trimmed Version of the Memory Map 293
118 Expansion Bus Address and Data Byte Steering 296
119 Expansion Bus Cycle Type Selection 298
120 Multiplexed Output Pins for HPI Operation 303
121 HPI HCNTL Control Signal Decoding 304
122 Expansion Bus Register Overview 319
123 Bit Level Definition for each of the Timing and Control Registers 322
124 Configuration Register 0 Description 323
125 Intel XScale® Processor Speed Expansion Bus Configuration Strappings 324
126 Expansion Bus Configuration Register 1-Bit Definition 325
127 Simulated Expansion Bus Performance 326
128 Address Map for the APB 330
129 Typical Baud Rate Settings 335
130 UART Transmit Parity Operation 337
131 UART Receive Parity Operation 337
132 UART Word-Length Select Configuration 338
133 UART FIFO Trigger Level 343
134 High-Speed UART Registers Overview 344
135 UART IDD Bit Mapping 349
136 Console UART Registers Overview 357
137 Priority Levels of Interrupt Identification Register 361
138 UART Interrupt Identification Bit Level Definition 362
139 Bus Arbitration Example: Three Requesting Masters 373
140 Memory Map 374
141 GPIO Interrupt Selections 378
142 GPIO Clock Frequency Select 380
143 GPIO Duty Cycle Select 380
144 GPIO Registers Overview 381
145 Interrupt Controller Registers 391
146 Timer Registers 401
147 Processors' Devices with Ethernet Interface 406

148 Processors' with Ethernet Interface 436
149 Ethernet MAC B Registers 436
150 Processors with HSS 438
151 HSS Tx/Rx Clock Output 445
152 HSS Tx/Rx Clock Output Frequencies and PPM Error 446
153 HSS Tx/Rx Clock Output Frequencies And Their Associated Jitter Characterization 446
154 HSS Frame Output Characterization 446
155 Jitter Definitions 447
156 Endpoint Configuration: Universal Serial Bus Device Controller 460
157 USB States 461
158 Endpoint Field Addressing 463
159 IN, OUT, and SETUP Token Packet Format 464
160 SOF Token Packet Format 464
161 Data Packet Format 464
162 Handshake Packet Format 465
163 Bulk Transaction Formats 465
164 Isochronous Transaction Formats 466
165 Control Transaction Formats 466
166 Interrupt Transaction Formats 467
167 Host Device Request Summary 468
168 USB-Device Register Descriptions 470
169 Processors' Devices with UTOPIA 528
170 JTAG Instruction Set 543
171 JTAG Device Register Values 545
172 AHB Queue Manager Memory Map 548
173 Queue Status Flags 551

Revision History

Date	Revision	Description
September 2006	006	1. Added the 533MHz IXP423 to Figure 2 2. Updated Table 3.1.1.1, Table 3.8.2.1, and the note for Table 126 3. Updated Section 12.1 4. Added clarifying information regarding the MDI Interface to Section 15.1.3, Section 15.2.46, and Table 153 5. Added additional description information to bit 3 of Table 124 6. Updated Table 96, "Memory Map" Incorporated specification changes, specification clarifications and document changes from the *Intel® IXP4XX Product Line Specification Update* (306428-004 and 306428-005)
March 2005	005	Incorporated specification changes, specification clarifications and document changes from the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Specification Update* (252702-006). 1. Removed Table 1, "Processor Features". Refer to the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Datasheet*. 2. Replaced Figure 1, Figure 2, Figure 3, Figure 4, and Figure 5. Added new product IXP423, Figure 2. 3. Updated Table 2, "Network Processor Functions". 4. Added Section 2.12, "Universal Asynchronous Receiver Transceiver" 5. Updated note on Section 3.1.1.1, "Page (P) Attribute Bit". 6. Updated Table 94, "Network Processor Functions". 7. Added Implication to Section 6.1, "PCI Controller Configured as Host". 8. Added Section 6.6.1, "PCI Byte Enables". 9. Corrected Register table bit 15, Section 6.14.2.8, "PCI Controller Control and Status Register". 10. Corrected Figure 60, "Expansion Bus Peripheral Connection". 11. Replaced all figures in the Expansion Bus Controller chapter (Figure 63–Figure 76). 12. Corrected register table bits 20:17, Section 8.9.9, "Configuration Register 0". 13. Corrected Table 125, "Intel XScale® Processor Speed Expansion Bus Configuration Strappings". 14. Added new Table 127, "Simulated Expansion Bus Performance". 15. Updated Table 151, "Processors' Devices with Ethernet Interface". 16. Updated Figure 80, "Multiple Ethernet PHYS Connected to Processor". 17. Enhanced Deferral Parameter Registers information in Section 15.1.4, "Transmitting Ethernet Frames with MII Interfaces". 18. Enhanced information regarding Dropped Broadcast Frames in Section 15.1.5, "Receiving Ethernet Frames with MII Interfaces". 19. Corrected register description in Section 15.2, "Register Descriptions". 20. Corrected register description in Section 15.2.1, "Transmit Control 1". 21. Corrected register description in Section 15.2.2, "Transmit Control 2". 22. Corrected register description and added notes to Section 15.2.3, "Receive Control 1". 23. Corrected register description in Section 15.2.4, "Receive Control 2". 24. Corrected register description in Section 15.2.9, "Transmit Deferral Parameters". 25. Corrected register description in Section 15.2.10, "Receive Deferral Parameters". 26. Corrected register description in Section 15.2.11, "Transmit Two Part Deferral Parameters 1". 27. Corrected register description in Section 15.2.12, "Transmit Two Part Deferral Parameters 2". 28. Corrected register description in Section 15.2.46, "Core Control". 29. Updated Table 152, "Processors' with Ethernet Interface". 30. Updated Table 154, "Processors with HSS". 31. Updated Table 173, "Processors' Devices with UTOPIA". Change bars indicate areas of change.
June 2004	004	Updated Intel® product branding. Change bars were retained from the previous release of this document (003).

Date	Revision	Description
March 2004	003	Incorporated specification changes, specification clarifications and document changes from the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Specification Update* (252702-003). 1. Added Section 2.5, AHB Queue Manager 2. Updated Ethernet MAC A and High-Speed Serial Interfaces sections. 3. Added footnote to Table 1, Processor Features 4. Updated PCI and Expansion Bus Controller functional overview 5. Updated product number on Table 9, ID Register 6. Updated Section 6.2, PCI Controller in Option Mode 7. Updated Table 104, Support Configuration of the SDRAM Controller, 32 Mbyte 64 Mbit support 8. Updated Section 7.2.1, Initializing the SDRAM, routine 9. Added footnote to Table 117, Expansion Bus Address and Data Byte Steering 10. Updated Section 8.6, Using I/O Wait 11. Updated Section 8.8.14, TI* HPI-16, Simplex-Mode Read Access, Figure 73, Expansion-Bus Read 12. Updated Table 123, Configuration Register 0 Description, bit values 13. Added Section 8.9.9.1, User-Configurable Field 14. Updated Section 8.10, Expansion Bus Controller Performance 15. Updated Table 126, Address Map for the APB, peripheral descriptions 16. Updated Section 12, GPIO, description 17. Updated Table 143, GPIO Interrupt Selections 18. Added Sections 15, Ethernet MAC A 19. Added Sections 17.5-17.6, High Speed Serial Interface
June 2003	002	Incorporated specification changes, specification clarifications and document changes from the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Specification Update*. (252702-001). Incorporated information for the Intel® IXC1100 Control Plane Processor.
February 2003	001	Initial release of this document. Document reissued, without “Confidential” marking.

1.0 Introduction

1.1 About This Document

This document is the main reference for the external architecture of the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor.

1.1.1 How to Read This Document

Familiarity with ARM* Version 5TE Architecture is necessary in order to understand some aspects of this document.

Each chapter in this document focuses on a specific architectural feature of the Intel® IXP42X product line and IXC1100 control plane processors.

Note: This document's special terms and acronyms are listed in "Terminology and Conventions" on page 26.

1.2 Other Relevant Documents

Document Title	Document #
Intel® IXP4XX Product Line Specification Update	306428
Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Datasheet	252479
Intel® IXP400 Software Specification Update	273795
Intel® IXP400 Software Programmer's Guide	252539
ARM Architecture Version 5TE Specification*	ARM DDI 0100E (ISBN 0 201 737191)
PCI Local Bus Specification, Rev. 2.2	N/A
Universal Serial Bus Specification, Revision 1.1	N/A
UTOPIA Level 2 Specification, Revision 1.0	N/A
IEEE 802.3 Specification	N/A
IEEE 1149.1 Specification	N/A

1.3 Terminology and Conventions

1.3.1 Number Representation

All numbers in this document can be assumed to be base 10 unless designated otherwise. In text and pseudo code descriptions, hexadecimal numbers have a prefix of 0x and binary numbers have a prefix of 0b. For example, 107 would be represented as 0x6B in hexadecimal and 0b1101011 in binary.

1.3.2 Acronyms and Terminology

Table 1. Acronyms and Terminology

Acronym / Terminology	Description
AAL	ATM Adaptation Layers
AES	Advanced Encryption Standard
AHB	Advanced High-Performance Bus
APB	Advanced Peripheral Bus
API	Application Program Interface
ARBS	South Arbiter
Assert	The logically active value of a signal or bit.
ATM-TC	Asynchronous Transmission Mode – Transmission Convergence
AQM	AHB Queue Manager
BTB	Branch Target Buffer
Clean	An operation that updates external memory with the contents of the specified line in the data/mini-data cache if any of the dirty bits are set and the line is valid. There are two dirty bits associated with each line in the cache so only the portion that is dirty will get written back to external memory. After this operation, the line is still valid and both dirty bits are deasserted.
Coalescing	Bringing together a new store operation with an existing store operation already resident in the write buffer. The new store is placed in the same write buffer entry as an existing store when the address of the new store falls in the four-word, aligned address of the existing entry. This includes, in PCI terminology, write merging, write collapsing, and write combining.
CRC	Cyclical Redundancy Check
FCS	Frame-Check Sequence
Deassert	The logically inactive value of a signal or bit.
DMA	Direct Memory Access
DSP	Digital Signal Processor
E1	Euro 1 trunk line
FIFO	First In First Out
Flush	An operation that invalidates the location(s) in the cache by de-asserting the valid bit. Individual entries (lines) may be flushed or the entire cache may be flushed with one command. Once an entry is flushed in the cache it can no longer be used by the program.
GCI	General Circuit Interface
GPIO	General-purpose input/output
G.SHDSL	ITU G series specification for Single-Pair HDSL
HDLC	High-level Data Link Control
HDSL	High-Bit-Rate Digital Subscriber Line
HDSL2	High-Bit-Rate Digital Subscriber Line, Version 2
HEC	Head-Error Correction
HPI	(Texas Instrument) Host Port Interfaces
HSS	High-Speed Serial (port)
ISDN	Integrated Services Digital Network
IOM	ISDN Orientated Modular
LFSR	Linear Feedback Shift Register
LSb	Least-Significant bit

Table 1. Acronyms and Terminology (Continued)

Acronym / Terminology	Description
LSB	Least-Significant Byte
LUT	Look-Up Table
MAC	Media Access Controller
MDIO	Management Data Input/Output
MIB	Management Information Base
MII	Media-Independent Interface
MMU	Memory Management Unit
MSb	Most-Significant bit
MSB	Most-Significant Byte
MVIP	Multi-Vendor Integration Protocol
NPE	Network Processor Engine
NRZI	Non-Return To Zero Inverted
PCI	Peripheral Component Interconnect
PEC	Programmable Event Counters
PHY	Physical Layer (Layer 1) Interface
Reserved	A field that may be used by an implementation. Software should not modify reserved fields or depend on any values in reserved fields.
RX	Receive (HSS is receiving from off-chip)
SFD	Start of Frame Delimiter
SRAM	Static Random Access Memory
SDRAM	Synchronous Dynamic Random Access Memory
T1	Type 1 trunk line
TDM	Time Division Multiplex
TLB	Translation Look-Aside Buffer
TX	Transmit (HSS is transmitting off-chip)
UART	Universal Asynchronous Receiver-Transmitter
WAN	Wide Area Network

2.0 Overview of Product Line

The Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor contain an ARM* V5TE-compliant microprocessor referred to as the Intel XScale® Processor. The Intel® IXP42X product line and IXC1100 control plane processors are designed with Intel 0.18-micron production semiconductor process technology. This process technology, along with the compactness of the Intel XScale processor, simultaneous processing of three integrated Network Processing Engines, and numerous dedicated function peripheral interfaces enables the IXP42X product line and IXC1100 control plane processors to operate over a wide range of low-cost networking applications, producing industry-leading performance.

As indicated in Figure 1 through Figure 5, the IXP42X product line and IXC1100 control plane processors combine many features with the Intel XScale processor to create a highly integrated processor applicable to LAN/WAN based networking applications. The IXP42X product line and IXC1100 control plane processors provide two MII interfaces; a UTOPIA Level -2 interface; a USB v1.1 device controller with embedded transceiver; a 32-bit, 33/66-MHz PCI bus; an 16-bit expansion bus; a 32-bit, 133-MHz SDRAM Interface; two UARTs; two High-Speed Serial Interfaces and 16 GPIOs.

Unless otherwise specified, the functional descriptions apply to all of the IXP42X product line and IXC1100 control plane processors. Refer to the table, “Processor Features”, in the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Datasheet* for an overview feature matrix that includes software enables for all supported processors.

Figure 1. Intel® IXP425 Network Processor Block Diagram

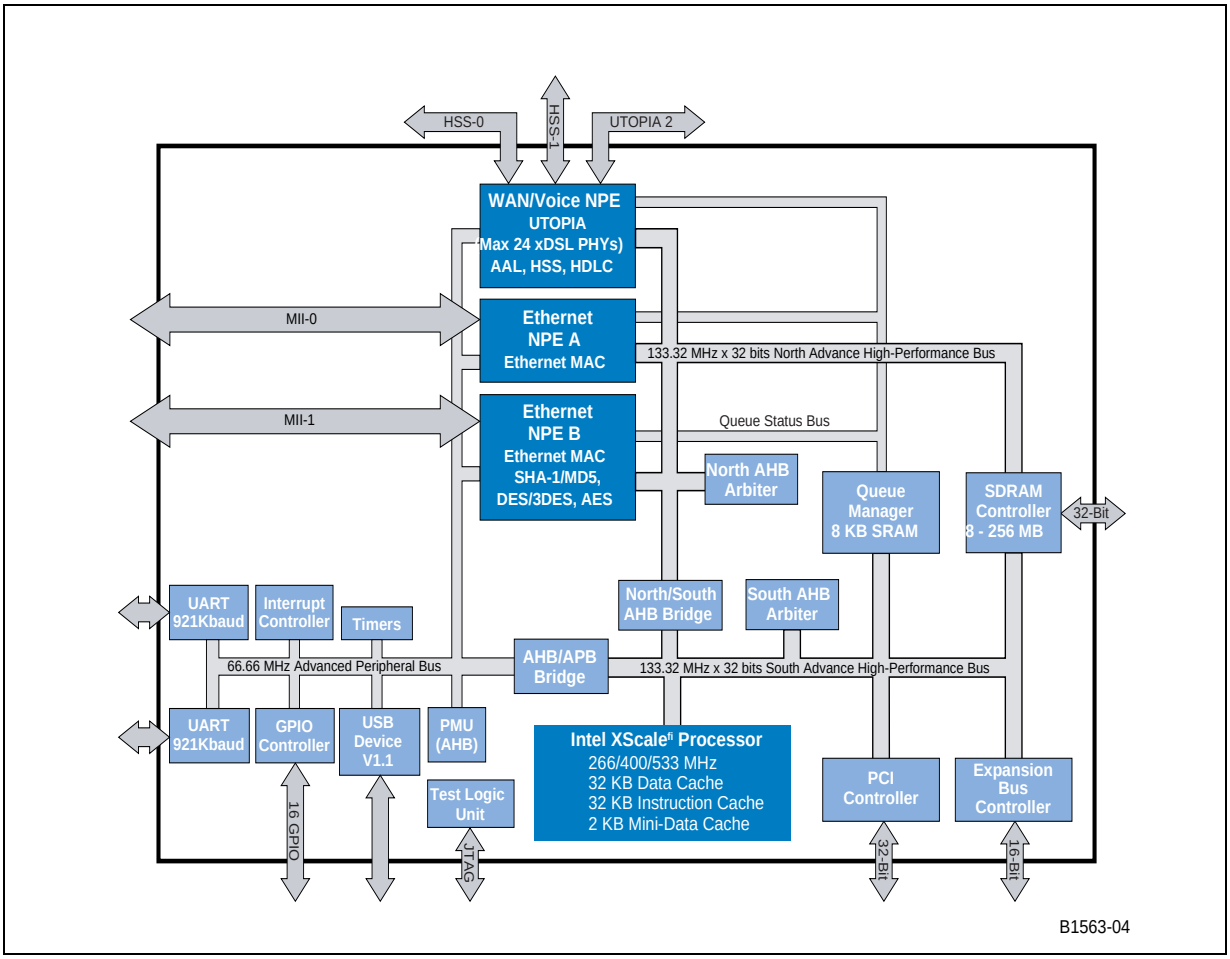


Figure 2. Intel® IXP423 Network Processor Block Diagram

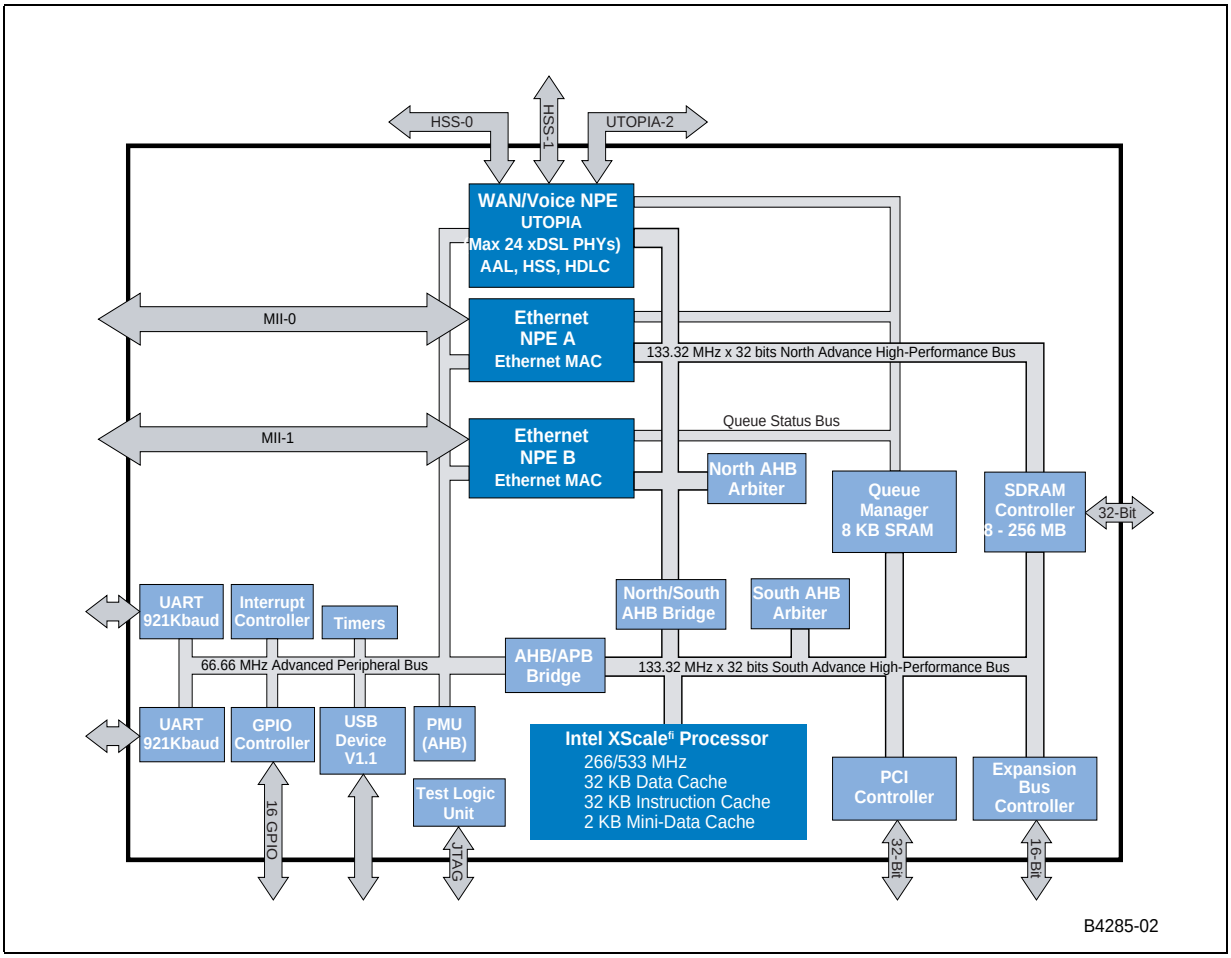


Figure 3. Intel® IXP422 Network Processor Block Diagram

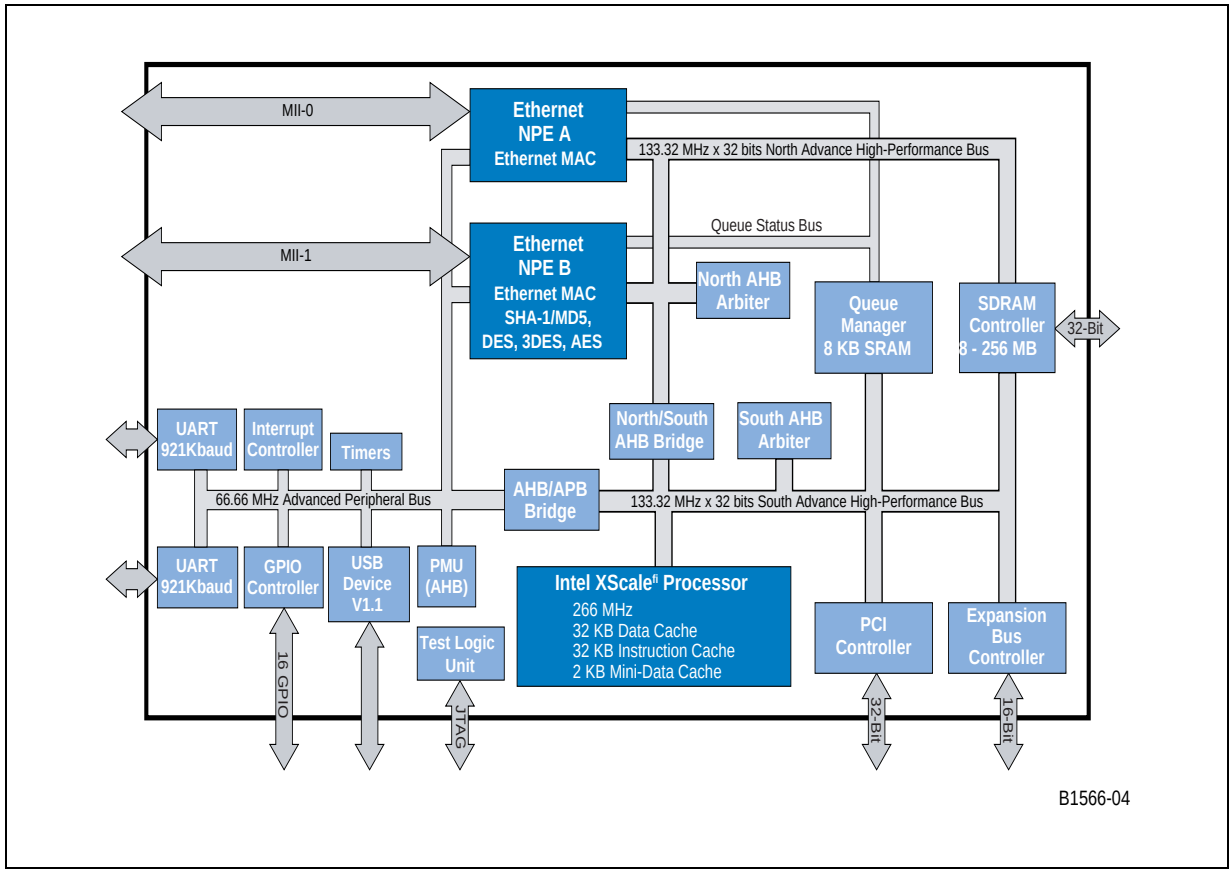


Figure 4. Intel® IXP421 Network Processor Block Diagram

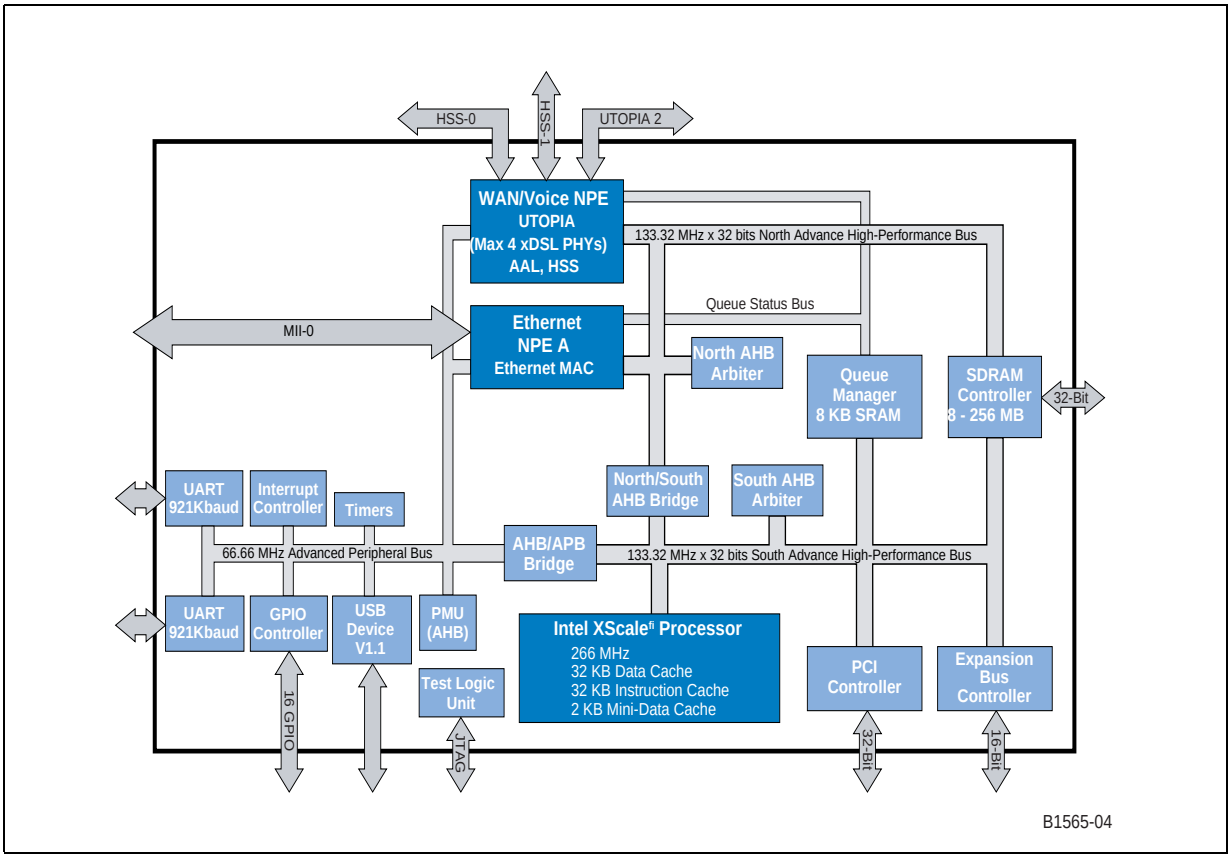
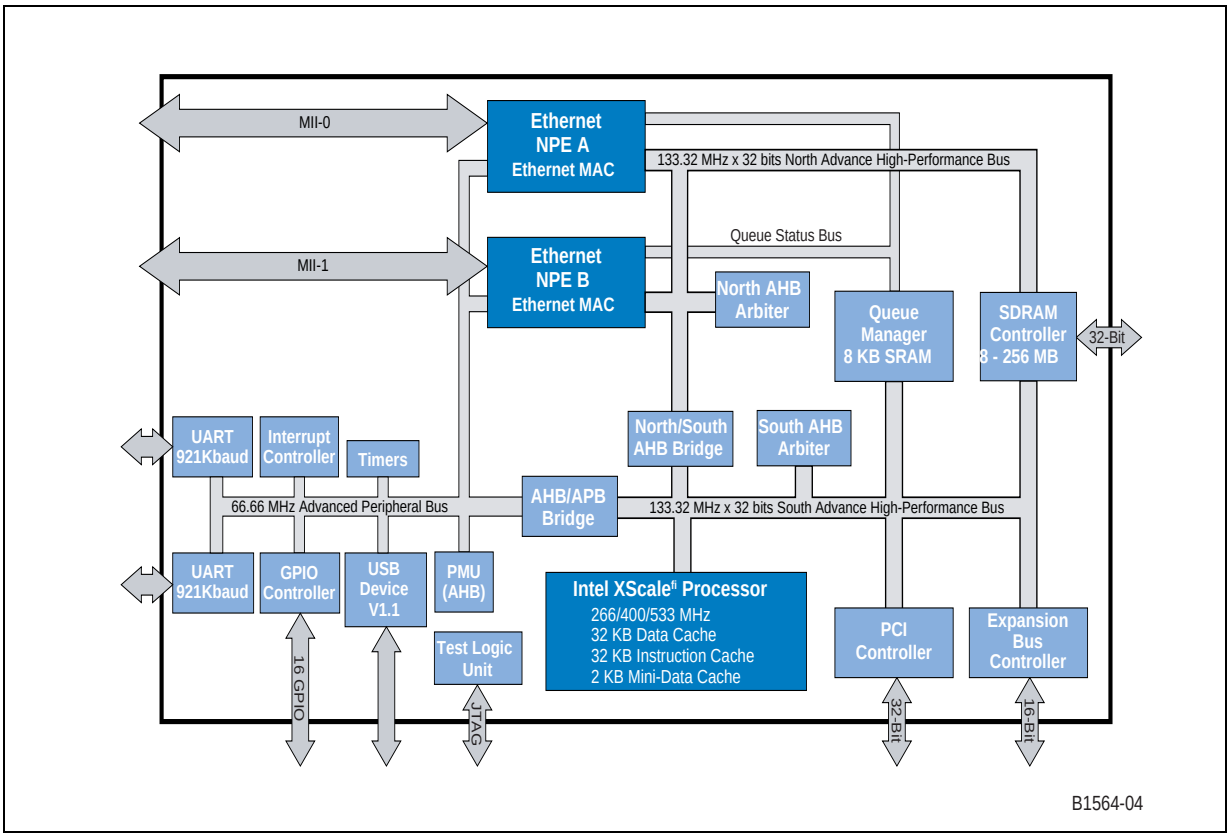


Figure 5. Intel® IXP420 Network Processor and Intel® IXC1100 Control Plane Processor Block Diagram



2.1 Intel XScale® Microarchitecture Processor

The Intel XScale® Processor incorporates an extensive list of architecture features that allows it to achieve high performance. This rich feature set allows programmers to select the appropriate features that obtains the best performance for their application. Many of the architectural features added to Intel XScale processor help hide memory latency which often is a serious impediment to high-performance processors.

Intel XScale® Processor features include:

- The ability to continue instruction execution even while the data cache is retrieving data from external memory
- A write buffer
- Write-back caching
- Various data cache allocation policies that can be configured different for each application
- Cache-locking

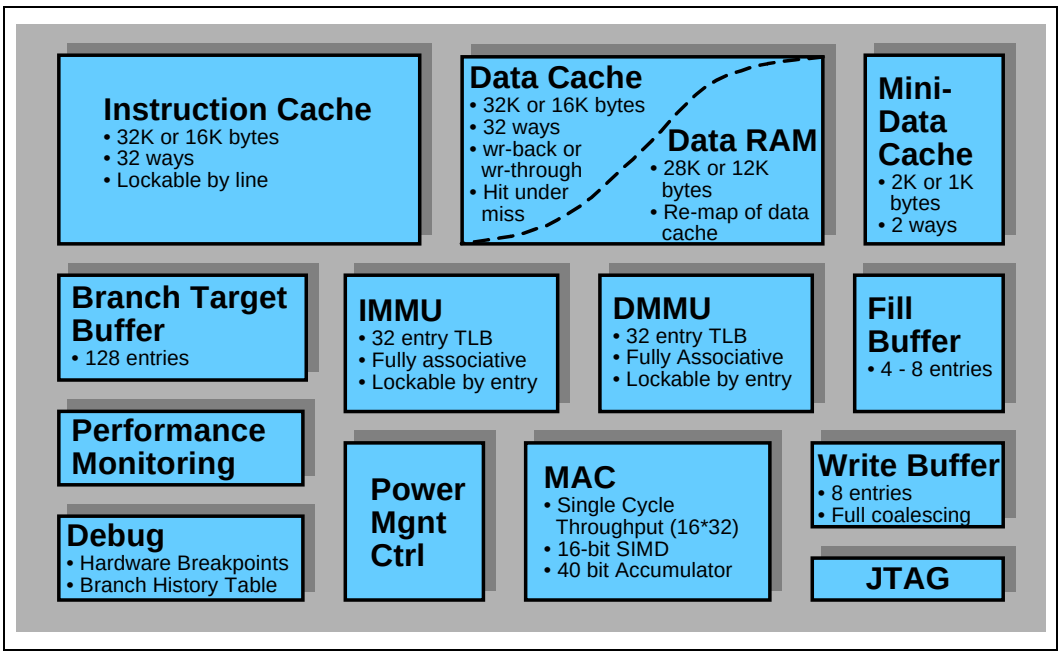
All these features improve the efficiency of the memory bus external to the IXP42X product line and IXC1100 control plane processors.

The IXP42X product line and IXC1100 control plane processors have been equipped to efficiently handle audio processing through the support of 16-bit data types and 16-bit operations. These audio-coding enhancements center around multiply and accumulate operations which accelerate many of the audio filter operations.

2.1.1 Intel XScale® Processor Overview

Figure 6 shows the major functional blocks of the Intel XScale processor. This section gives brief, high-level overviews of these blocks.

Figure 6. Intel XScale® Technology Architecture Features



Note: The Power Management Control feature was not implemented in the IXP42X product line and IXC1100 control plane processors.

2.1.1.1 ARM* Compatibility

ARM* Version 5 Architecture added floating point instructions to ARM Version 4. The Intel XScale processor implements the integer instruction set architecture of ARM V5, but does not provide hardware support of the floating point instructions.

Intel XScale processor provides the Thumb* instruction set (ARM V5T) and the ARM V5E DSP extensions.

Backward compatibility with ARM products is maintained for user-mode applications. Operating systems may require modifications to match the specific hardware features of the IXP42X product line and IXC1100 control plane processors and to take advantage of added performance enhancements.

2.1.1.2 Multiply/Accumulate (MAC)

The MAC unit supports early termination of multiplies/accumulates in two cycles and can sustain a throughput of a MAC operation every cycle. Several architectural enhancements were made to the MAC to support audio coding algorithms, which include a 40-bit accumulator and support for 16-bit packed data.

2.1.1.3 Memory Management

The Intel XScale processor implements the Memory Management Unit (MMU) Architecture specified in the *ARM Architecture Reference Manual*. The MMU provides access protection and virtual-to-physical address translation.

The MMU Architecture also specifies the caching policies for the instruction cache and data cache. These policies are specified as page attributes and include:

- Identifying code as cacheable or non-cacheable
- Selecting between the mini-data cache or data cache
- Write-back or write-through data caching
- Enabling data-write allocation policy
- Enabling the write buffer to coalesce stores to external memory

For more details, see Section 3.1, “Memory Management Unit” on page 44.

2.1.1.4 Instruction Cache

The Intel XScale processor comes with a 32-Kbyte instruction cache. The instruction cache is 32-way set associative and has a line size of 32 bytes. All requests that “miss” the instruction cache generate a 32-byte read request to external memory. A mechanism to lock critical code within the cache also is provided.

For more details, see “Instruction Cache” on page 52.

2.1.1.5 Branch Target Buffer

The Intel XScale processor provides a Branch Target Buffer (BTB) to predict the outcome of branch-type instructions. It provides storage for the target address of branch type instructions and predicts the next address to present to the instruction cache, when the current instruction address is that of a branch.

The BTB holds 128 entries. For more details, see “Branch Target Buffer” on page 58.

2.1.1.6 Data Cache

The Intel XScale processor comes with a 32-Kbyte data cache. Besides the main data cache, a mini-data cache is provided whose size is 1/16th the main data cache. (A 32-Kbyte main data cache has a 2-Kbyte mini-data cache.)

The main data cache is 32-way set associative and the mini-data cache is two-way set associative. Each cache has a line size of 32 bytes and supports write-through or write-back caching.

The data/mini-data cache is controlled by page attributes defined in the MMU Architecture and by coprocessor 15.

For more details, see “Data Cache” on page 60.

The Intel XScale processor allows applications to reconfigure a portion of the data cache as data RAM. Software may place special tables or frequently used variables in this RAM. For more information on this, see “Reconfiguring the Data Cache as Data RAM” on page 68.

2.1.1.7 Intel XScale® Processor Performance Monitoring

Two performance-monitoring counters have been added to the Intel XScale processor that can be configured to monitor various events in the Intel XScale processor. These events allow a software developer to measure cache efficiency, detect system bottlenecks, and reduce the overall latency of programs.

For more details, see "Performance Monitoring" on page 133 and Section 11.0, "Internal Bus Performance Monitoring Unit (IBPMU)" on page 372.

2.2 Network Processor Engines (NPE)

The network processor engines are dedicated function processors integrated into many of the IXP42X product line and IXC1100 control plane processors to off load processing function required by the Intel XScale processor. Table 2 specifies which devices, of the IXP42X product line and IXC1100 control plane processors, have which of these capabilities.

Table 2. Network Processor Functions

Device	UTOPIA	HSS	MII 0	MII 1	AES / DES / 3DES	Multi-Channel HDLC	SHA-1 / MD-5
Intel® IXP425 Network Processor	X	X	X	X	X	8	X
Intel® IXP423 Network Processor	X	X	X	X		8	
Intel® IXP422 Network Processor			X	X	X		X
Intel® IXP421 Network Processor	X	X	X			8	
Intel® IXP420 Network Processor			X	X			
Intel® IXC1100 Control Plane Processor			X	X			

The network processor engines are high-performance, hardware-multi-threaded processors. All instruction code is stored locally with a dedicated instruction-memory bus. These engines support processing of the dedicated peripherals. The peripherals supported using the network processor engines are the following interfaces:

- up to 2 MII
- UTOPIA Level-2
- up to 2 HSS

The combined forces of the hardware multi-threading, local code store, independent instruction memory, and parallel processing allows the Intel XScale processor to be utilized purely for application purposes. This parallel processing of the peripheral interface functions allows unsurpassed performance to be achieved by the application running on the IXP42X product line and IXC1100 control plane processors.

For further information on the network processor engines, see Section 4.0, "Network Processor Engines (NPE)" on page 202.

2.3 Internal Bus

The internal bus architecture of the Intel XScale processor is designed to allow parallel processing to occur and isolate bus utilization based on particular traffic patterns. The bus is segmented into three major buses: the North AHB, the South AHB, and the APB.

The North AHB is a 133.32 MHz, 32-bit bus that can be mastered by the WAN NPE or both of the Ethernet NPEs. The targets of the North AHB can be the SDRAM or the AHB/AHB Bridge.

The AHB/AHB Bridge will allow access by the NPEs to the peripherals and internal targets on the South AHB. Data transfers by the NPEs on the North AHB to the South AHB are targeted predominately to the queue manager. Transfers to the AHB/AHB Bridge may be “posted” when writing or “split” when reading. This allows control of the North AHB to be given to another master on the North AHB and enables the bus to achieve maximum efficiency.

Transfers to the AHB/AHB Bridge are considered to be small and infrequent relative to the traffic passed between the NPEs on the North AHB and the SDRAM.

The South AHB is a 133.32 MHz, 32-bit bus that can be mastered by the Intel XScale processor, PCI Controller DMA engines, AHB/AHB Bridge, and the AHB/APB Bridge. The targets of the South AHB can be the SDRAM, PCI Interface, Queue Manager, or the APB/AHB Bridge. Accessing across the APB/AHB allows interfacing to peripherals attached to the APB.

The APB is a 66.66 MHz, 32-bit bus that can be mastered by the AHB/APB Bridge only. The targets of the APB can be the High-Speed UART Interface, Console UART Interface, USB v1.1 interface, all NPEs, the Performance Monitoring Unit (PMU), Interrupt Controller, General-Purpose Input/Output (GPIO), and timers. The APB interface to the NPEs are used for code download and part configuration.

For more information, see Section 5.0, “Internal Bus” on page 204.

2.4 MII Interfaces

Two industry-standard Media Independent Interfaces (MII) are integrated into the IXP42X product line and IXC1100 control plane processors with separate Media Access Controllers and Network Processing Engines. This enables parallel processing of data traffic on the interfaces and off loading of processing overhead required by the Intel XScale processor.

The IXP42X product line and IXC1100 control plane processors are compliant with the *IEEE,* 802.3 specification.

2.5 AHB Queue Manager

The AHB Queue Manager (AQM) provides queue functionality for various internal blocks. It maintains the queues as circular buffers in an embedded 8KB SRAM. It also implements the status flags and pointers required for each queue.

The AQM manages 64 independent queues. Each queue is configurable for buffer and entry size. Additionally status flags are maintained for each queue.

The AQM interfaces include an Advanced High-performance Bus (AHB) interface to the NPEs and Intel XScale processor (or any other AHB bus master), a Flag Bus interface, an event bus (to the NPE condition select logic) and two interrupts to the Intel XScale processor. The AHB interface is used for configuration of the AQM and provides access to queues, queue status and SRAM. Individual queue status for queues 0-31 is

communicated to the NPEs via the flag bus. Combined queue status for queues 32-63 are communicated to the NPEs via the event bus. The two interrupts, one for queues 0-31 and one for queues 32-63, provide status interrupts to the Intel XScale processor.

For more information on the AHB Queue Manager, see Section 21.0, “AHB Queue Manager (AQM)” on page 556.

2.6 UTOPIA 2

The integrated UTOPIA Level -2 interface has a dedicated network-processing engine. The interface allows a multiple- or single-physical-interface configuration. The network processing engine handles segmentation and reassembly of Asynchronous Transfer Mode (ATM) cells, CRC Checking/Generation, and the transfer of data to and from memory. This enables parallel processing of data traffic on the UTOPIA Level-2 interface, off loading processor overhead required by the Intel XScale processor.

The IXP42X product line and IXC1100 control plane processors are compliant with the *ATM Forum,* UTOPIA Level -2, Revision 1.0 specification.

For more information on the UTOPIA Level-2 interface, see Section 19.0, “UTOPIA Level-2” on page 538.

2.7 USB v1.1

The integrated USB v1.1 interface is a device-only controller. The interface supports full-speed operation and 16 end points and includes an integrated transceiver. The endpoints include:

- Six isochronous endpoints (three input and three output)
- Two control endpoints (one input and one output)
- Two interrupt endpoints (one input and one output)
- Six bulk endpoints (one input and one output)

For more information on the USB v1.1 interface, see Section 18.0, “Universal Serial Bus (USB) v1.1 Device Controller” on page 468.

2.8 PCI

The IXP42X product line and IXC1100 control plane processors’ PCI controller is compatible with the *PCI Local Bus Specification*, Rev. 2.2. The PCI interface is 32-bit compatible bus and capable of operating as either a host or an option (i.e. not the Host)

For more information on the PCI interface, see Section 6.0, “PCI Controller” on page 208.

2.9 Memory Controller

The memory controller manages the interface to external SDRAM memory chips. The interface:

- Operates at 133.32 MHz (which is 4 * OSC_IN input pin.)
- Supports eight open pages simultaneously
- Has two banks to support memory configurations from 8 Mbyte to 256 Mbyte

The memory controller only supports 32-bit memory. If a x16 memory chip is used, a minimum of two memory chips would be required to facilitate the 32-bit interface required by the IXP42X product line and IXC1100 control plane processors. A maximum of four SDRAM memory chips may be attached to the processors.

The memory controller internally interfaces to the North AHB and South AHB with independent peripherals. This architecture allows SDRAM transfers to be interleaved and pipelined to achieve maximum possible efficiency. The maximum burst size supported to the SDRAM Interface is 8-32 bit words. This burst size allows the best efficiency/fairness performance between accesses from the North and South AHB.

For more information on the memory controller, see Section 7.0, “SDRAM Controller” on page 276.

2.10 Expansion Bus

The expansion interface allows easy and — in most cases — glue-less connection to peripheral devices. It also provides input information for device configuration after reset. Some of the peripheral device types are flash, ATM control interfaces, and DSPs used for voice applications. (Some voice configurations can be supported by the HSS interfaces and the Intel XScale® Processor, implementing voice-compression algorithms.)

The expansion bus interface is a 16-bit interface that allows an address range of 512 bytes to 16 Mbytes, using 24 address lines for each of the eight independent chip selects.

Accesses to the expansion bus interface consists of five phases. Each of the five phases can be lengthened or shortened by setting various configuration registers on a per-chip-select basis. This feature allows the IXP42X product line and IXC1100 control plane processors to connect to a wide variety of peripheral devices with varying speeds.

The expansion bus interface supports Intel or Motorola* microprocessor-style bus cycles. The bus cycles can be configured to be multiplexed address/data cycles or separate address/data cycles for each of the eight chip-selects.

Additionally, Chip Selects 4 through 7 can be configured to support Texas Instruments HPI-8 or HPI-16 style accesses for DSPs.

The expansion bus interface is an asynchronous interface to externally connected chips. However, a clock must be supplied to the IXP42X product line and IXC1100 control plane processors’ expansion bus interface for the interface to operate. This clock can be driven from GPIO 15 or an external source. The maximum clock rate that the expansion bus interface can accept is 66.66 MHz.

At the de-assertion of reset, the 24-bit address bus is used to capture configuration information from the levels that are applied to the pins at this time. External pull-up/pull-down resistors are used to tie the signals to particular logic levels

For more information on the Expansion Interface, see Section 8.0, “Expansion Bus Controller” on page 292.

2.11 High-Speed Serial Interfaces

The High-Speed Serial interfaces are a six-signal interface that supports serial transfer speeds from 512 KHz to 8.192 MHz.

For more information on the High-Speed Serial Interfaces, see Section 17.0, “High-Speed Serial Interfaces” on page 448.

2.12 Universal Asynchronous Receiver Transceiver

The UART interfaces are 16550-compliant UARTs with the exception of transmit and receive buffers. Transmit and receive buffers are 64 bytes-deep versus the 16 bytes required by the 16550 UART specification.

The interface can be configured to support speeds from 1,200 baud to 921 Kbaud. The interface support configurations of:

- Five, six, seven, or eight data-bit transfers
- One or two stop bits
- Even, odd, or no parity

The request-to-send (RTS_N) and clear-to-send (CTS_N) modem control signals also are available with the interface for hardware flow control.

For more information on the UART interfaces, see Section 10.0, “Universal Asynchronous Receiver Transceiver (UART)” on page 332.

2.13 GPIO

There are 16 GPIO pins supported by the IXP42X product line and IXC1100 control plane processors. GPIO pins 0 through 13 can be configured to be general-purpose input or general-purpose output. Additionally, GPIO pins 0 through 12 can be configured to be an interrupt input.

GPIO Pin 14 can be configured similar to GPIO pin 13 or as a clock output. The output-clock configuration can be set at various speeds, up to 33.33 MHz, with various duty cycles. GPIO Pin 14 is configured as an input, upon reset.

GPIO Pin 15 can be configured similar to GPIO pin 13 or as a clock output. The output-clock configuration can be set at various speeds, up to 33.33 MHz, with various duty cycles. GPIO Pin 15 is configured as a clock output, upon reset. GPIO Pin 15 can be used to clock the expansion interface, after reset.

For more information on the GPIO pins, see Section 12.0, “General Purpose Input/Output (GPIO)” on page 386.

2.14 Interrupt Controller

The IXP42X product line and IXC1100 control plane processors consist of 32 interrupt sources to allow an extension of the Intel XScale processor’s FIQ and IRQ interrupt sources. These sources can originate from external GPIO pins or internal peripheral interfaces.

The interrupt controller can configure each interrupt source as FIQ, IRQ, or disabled. The interrupt sources tied to Interrupt 0 to 7 can be prioritized. The remaining interrupts are prioritized in ascending order. (For example, 8 has a higher priority than 9.)

For more information on the interrupt controller, see Section 13.0, “Interrupt Controller” on page 398.

2.15 Timers

The IXP42X product line and IXC1100 control plane processors consists of four internal timers operating at 66.66 MHz (which is 2 * OSC_IN input pin.) to allow task scheduling and prevent software lock-ups. The device has four 32-bit counters:

- Watch-Dog Timer
- Timestamp Timer
- Two general-purpose timers

For more information on the timers, see Section 14.0, “Timers” on page 408.

2.16 JTAG

Testability is supported on the IXP42X product line and IXC1100 control plane processors through the Test Access Port (TAP) Controller implementation, which is based on IEEE 1149.1 (JTAG) Standard Test Access Port and Boundary-Scan Architecture. The purpose of the TAP controller is to support test logic internal and external to the IXP42X product line and IXC1100 control plane processors, such as built-in self test and boundary scan.

For more information on JTAG, see Section 20.0, “JTAG Interface” on page 548.

§ §

3.0 Intel XScale® Processor

This chapter provides functional descriptions of the Intel XScale® Processor.

3.1 Memory Management Unit

This section describes the memory management unit implemented in Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor.

The Intel XScale® Processor implements the Memory Management Unit (MMU) Architecture specified in the *ARM* Architecture Reference Manual*. To accelerate virtual-to-physical address translation, Intel XScale processor uses both an instruction Translation Look-Aside Buffer (TLB) and a data TLB to cache the latest translations. Each TLB holds 32 entries and is fully associative.

Not only do the TLBs contain the translated addresses, but also the access rights for memory references.

If an instruction or data TLB miss occurs, a hardware translation-table-walking mechanism is invoked to translate the virtual address to a physical address. Once translated, the physical address is placed in the TLB along with the access rights and attributes of the page or section. These translations can also be locked down in either TLB to guarantee the performance of critical routines.

For more information, refer to "Exceptions" on page 47.

The Intel XScale processor allows system software to associate various attributes with regions of memory:

- Cacheable
- Bufferable
- Line-allocate policy
- Write policy
- I/O
- Mini data cache
- Coalescing

For a description of page attributes, see "Cacheable (C), Bufferable (B), and eXtension (X) Bits" on page 45. For information on where these attributes have been mapped in the MMU descriptors, see "New Page Attributes" on page 152.

Note: The virtual address with which the TLBs are accessed may be remapped by the PID register. For a description of the PID register, see "Register 13: Process ID" on page 84.

ARM MMU Version 5 Architecture introduces the support of tiny pages, which are 1 Kbyte in size. The reserved field in the first-level descriptor (encoding 0b11) is used as the fine page table base address. The exact bit fields and the format of the first and second-level descriptors can be found in "New Page Attributes" on page 152.

The attributes associated with a particular region of memory are configured in the memory management page table and control the behavior of accesses to the instruction cache, data cache, mini-data cache, and the write buffer. These attributes are ignored when the MMU is disabled.

To allow compatibility with older system software, the new Intel XScale processor attributes take advantage of encoding space in the descriptors that was formerly reserved.

3.1.1 Memory Attributes

3.1.1.1 Page (P) Attribute Bit

The selection between address or data coherency is controlled by a software-programmable P-Attribute bit in the Intel XScale processor's Memory Management Unit (MMU) and BYTE_SWAP_EN bit. The BYTE_SWAP_EN bit will be from the Expansion-Bus Controller Configuration Register 1 Table 126, bit 8. When the IXP42X product line and IXC1100 control plane processors is reset, this bit will reset to 0.

The default endian-conversion method for IXP42X product line and IXC1100 control plane processors is address coherency. This was selected for backward compatibility with the IXP425 A0-step device.

The BYTE_SWAP_EN bit is an enable bit that allows data coherency to be performed, based on the P-Attribute bit.

- When the bit is 0, address coherency is always performed.
- When the bit is 1, the type of coherency performed is dependent on the P-Attribute bit.

The P-Attribute bit is associated with each 1-Mbyte page. The P-Attribute bit is output from the Intel XScale processor with any store or load access associated with that page.

Note: When preparing data for processing by the NPE memory (if byte swapping is necessary for the application), the P-attribute bit should be used to byte-swap the entire memory map belonging to the NPE region. For instance, when the Intel XScale processor is operating in little endian mode, all data arriving from the NPE that is to be read by the Intel XScale processor should be configured to swap all bytes of data. When writing this data from the Intel XScale processor to memory (with the intention of the NPE using this data) all bytes should be swapped using the P-attribute. Using the P-attribute bit to byte swap all of the NPE memory region will ensure compatible software code porting to future releases of the Intel XScale processor. Using the P-attribute bit to byte-swap 1-Mbyte regions of the NPE memory may not allow compatible software code porting to a future Intel XScale microarchitecture.

3.1.1.2 Cacheable (C), Bufferable (B), and eXtension (X) Bits

3.1.1.2.1 Instruction Cache

When examining these bits in a descriptor, the Instruction Cache only utilizes the C bit. If the C bit is clear, the Instruction Cache considers a code fetch from that memory to be non-cacheable and will not fill a cache entry. If the C bit is set, then fetches from the associated memory region will be cached.

3.1.1.2.2 Details on Data Cache and Write Buffer Behavior

If the MMU is disabled, all data accesses will be non-cacheable and non-bufferable. This is the same behavior as when the MMU is enabled and a data access uses a descriptor with X, C, and B all set to 0.

The X, C, and B bits determine when the processor should place new data into the Data Cache. The cache places data into the cache in lines (also called blocks). Thus, the basis for making a decision about placing new data into the cache is a called a “Line-Allocation Policy.”

If the Line-Allocation Policy is read-allocate, all load operations that miss the cache, request a 32-byte cache line from external memory and allocate it into either the data cache or mini-data cache. (This statement assumes that the cache is enabled.) Store operations that miss the cache will not cause a line to be allocated.

If read/write-allocate is in effect, load **or** store operations that miss the cache will request a 32-byte cache line from external memory if the cache is enabled.

The other policy determined by the X, C, and B bits is the Write Policy. A write-through policy instructs the data cache to keep external memory coherent by performing stores to both external memory and the cache. A write-back policy only updates external memory when a line in the cache is cleaned or needs to be replaced with a new line. Generally, write-back provides higher performance because it generates less data traffic to external memory. For more details on cache policies, see “Cacheability” on page 63

3.1.1.2.3 Data Cache and Write Buffer

All of these descriptor bits affect the behavior of the Data Cache and the Write Buffer.

If the X bit for a descriptor is zero, the C and B bits operate as mandated by the ARM architecture, refer to the *ARM* Architecture Reference Manual*. This behavior is detailed in Table 3.

If the X bit for a descriptor is one, the C and B bits’ meaning is extended, as detailed in Table 4.

Table 3. Data Cache and Buffer Behavior When X = 0

C B	Cacheable	Bufferable	Write Policy	Line Allocation Policy	Notes
0 0	N	N	-	-	Stall until complete*
0 1	N	Y	-	-	
1 0	Y	Y	Write Through	Read Allocate	
1 1	Y	Y	Write Back	Read Allocate	

Note: Normally, the processor will continue executing after a data access if no dependency on that access is encountered. With this setting, the processor will stall execution until the data access completes. This guarantees to software that the data access has taken effect by the time execution of the data access instruction completes. External data aborts from such accesses will be imprecise (but see “Data Aborts” on page 156 for a method to shield code from this imprecision).

Table 4. Data Cache and Buffer Behavior When X = 1

C B	Cacheable	Bufferable	Write Policy	Line Allocation Policy	Notes
0 0	-	-	-	-	Unpredictable -- do not use
0 1	N	Y	-	-	Writes will not coalesce into buffers[1]
1 0	(Mini Data Cache)	-	-	-	Cache policy is determined by MD field of Auxiliary Control register[2]
1 1	Y	Y	Write Back	Read/Write Allocate	

Notes:
1. Normally, bufferable writes can coalesce with previously buffered data in the same address range.
2. See "Register 1: Control and Auxiliary Control Registers" on page 77 for a description of this register.

3.1.1.2.4 Memory Operation Ordering

A *fence* memory operation (memop) is one that guarantees all memops issued prior to the fence will execute before any memop issued after the fence. Thus software may issue a fence to impose a partial ordering on memory accesses.

Table 5 on page 47 shows the circumstances in which memops act as fences.

Any swap (**SWP** or **SWPB**) to a page that would create a fence on a load or store is a fence.

Table 5. Memory Operations that Impose a Fence

Operation	X	C	B
load	-	0	-
store	1	0	1
load or store	0	0	0

3.1.1.2.5 Exceptions

The MMU may generate prefetch aborts for instruction accesses and data aborts for data memory accesses. The types and priorities of these exceptions are described in "Event Architecture" on page 154.

Data address alignment checking is enabled by setting bit 1 of the Control Register (CP15, register 1). Alignment faults are still reported even if the MMU is disabled. All other MMU exceptions are disabled when the MMU is disabled.

3.1.2 Interaction of the MMU, Instruction Cache, and Data Cache

The MMU, instruction cache, and data/mini-data cache may be enabled/disabled independently. The instruction cache can be enabled with the MMU enabled or disabled. However, the data cache can only be enabled when the MMU is enabled. Therefore only three of the four combinations of the MMU and data/mini-data cache enables are valid. The invalid combination will cause undefined results.

Table 6. Valid MMU and Data / Mini-Data Cache Combinations

MMU	Data / mini-data Cache
Off	Off
On	Off
On	On

3.1.3 MMU Control

3.1.3.1 Invalidate (Flush) Operation

The entire instruction and data TLB can be invalidated at the same time with one command or they can be invalidated separately. An individual entry in the data or instruction TLB can also be invalidated. See Table 19, “TLB Functions” on page 82 for a listing of commands supported by the Intel XScale processor.

Globally invalidating a TLB will not affect locked TLB entries. However, the invalidate-entry operations can invalidate individual locked entries. In this case, the locked contents remain in the TLB, but will never “hit” on an address translation. Effectively, creating a hole is in the TLB. This situation may be rectified by unlocking the TLB.

3.1.3.2 Enabling / Disabling

The MMU is enabled by setting bit 0 in coprocessor 15, register 1 (Control Register).

When the MMU is disabled, accesses to the instruction cache default to cacheable accesses and all accesses to data memory are made non-cacheable.

A recommended code sequence for enabling the MMU is shown in Example 1 on page 49.

Example 1. Enabling the MMU

```
; This routine provides software with a predictable way of enabling the MMU.
; After the CPWAIT, the MMU is guaranteed to be enabled. Be aware
; that the MMU will be enabled sometime after MCR and before the instruction
; that executes after the CPWAIT.
; Programming Note: This code sequence requires a one-to-one virtual to
; physical address mapping on this code since
; the MMU may be enabled part way through. This would allow the instructions
; after MCR to execute properly regardless the state of the MMU.


MRC P15,0,R0,C1,C0,0; Read CP15, register 1
ORR R0, R0, #0x1; Turn on the MMU
MCR P15,0,R0,C1,C0,0; Write to CP15, register 1


; For a description of CPWAIT, see
; "Additions to CP15 Functionality" on page 153
CPWAIT
; The MMU is guaranteed to be enabled at this point; the next instruction or
; data address will be translated.
```

3.1.3.3 Locking Entries

Individual entries can be locked into the instruction and data TLBs. See Table 20, "Cache Lock-Down Functions" on page 83 for the exact commands. If a lock operation finds the virtual address translation already resident in the TLB, the results are unpredictable. An invalidate by entry command before the lock command will ensure proper operation. Software can also accomplish this by invalidating all entries, as shown in Example 2 on page 50.

Locking entries into either the instruction TLB or data TLB reduces the available number of entries (by the number that was locked down) for hardware to cache other virtual to physical address translations.

A procedure for locking entries into the instruction TLB is shown in Example 2 on page 50.

If a MMU abort is generated during an instruction or data TLB lock operation, the Fault Status Register is updated to indicate a Lock Abort (see "Data Aborts" on page 156), and the exception is reported as a data abort.

Example 2. Locking Entries into the Instruction TLB

```
; R1, R2 and R3 contain the virtual addresses to translate and lock into
; the instruction TLB.

; The value in R0 is ignored in the following instruction.
; Hardware guarantees that accesses to CP15 occur in program order

MCR P15,0,R0,C8,C5,0      ; Invalidate the entire instruction TLB

MCR P15,0,R1,C10,C4,0     ; Translate virtual address (R1) and lock into
                          ; instruction TLB
MCR P15,0,R2,C10,C4,0  ; Translate
                          ; virtual address (R2) and lock into instruction TLB
MCR P15,0,R3,C10,C4,0     ; Translate virtual address (R3) and lock into
                          ; instruction TLB

CPWAIT

; The MMU is guaranteed to be updated at this point; the next instruction will
; see the locked instruction TLB entries.
```

Note: If exceptions are allowed to occur in the middle of this routine, the TLB may end up caching a translation that is about to be locked. For example, if R1 is the virtual address of an interrupt service routine and that interrupt occurs immediately after the TLB has been invalidated, the lock operation will be ignored when the interrupt service routine returns back to this code sequence. Software should disable interrupts (FIQ or IRQ) in this case.

As a general rule, software should avoid locking in all other exception types.

The proper procedure for locking entries into the data TLB is shown in Example 3 on page 51.

Example 3. Locking Entries into the Data TLB

```
; R1, and R2 contain the virtual addresses to translate and lock into the data TLB


MCR  P15,0,R1,C8,C6,1      ; Invalidate the data TLB entry specified by the
                           ; virtual address in R1
MCR  P15,0,R1,C10,C8,0     ; Translate virtual address (R1) and lock into
                           ; data TLB

; Repeat sequence for virtual address in R2
MCR  P15,0,R2,C8,C6,1      ; Invalidate the data TLB entry specified by the
                           ; virtual address in R2
MCR  P15,0,R2,C10,C8,0     ; Translate virtual address (R2) and lock into
                           ; data TLB

CPWAIT                     ; wait for locks to complete


; The MMU is guaranteed to be updated at this point; the next instruction will
; see the locked data TLB entries.
```

Note: Care must be exercised here when allowing exceptions to occur during this routine whose handlers may have data that lies in a page that is trying to be locked into the TLB.

3.1.3.4 Round-Robin Replacement Algorithm

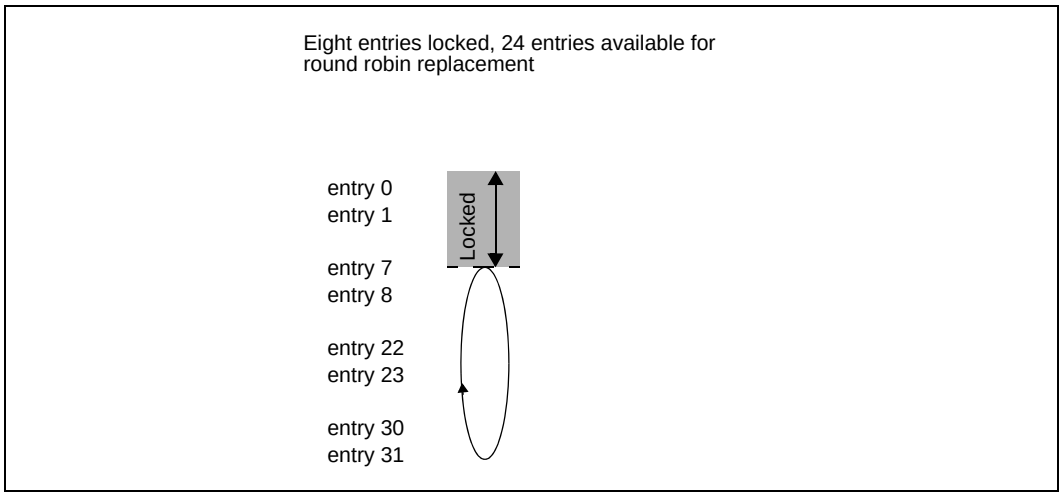
The line replacement algorithm for the TLBs is round-robin; there is a round-robin pointer that keeps track of the next entry to replace. The next entry to replace is the one sequentially after the last entry that was written. For example, if the last virtual to physical address translation was written into entry 5, the next entry to replace is entry 6.

At reset, the round-robin pointer is set to entry 31. Once a translation is written into entry 31, the round-robin pointer gets set to the next available entry, beginning with entry 0 if no entries have been locked down. Subsequent translations move the round-robin pointer to the next sequential entry until entry 31 is reached, where it will wrap back to entry 0 upon the next translation.

A lock pointer is used for locking entries into the TLB and is set to entry 0 at reset. A TLB lock operation places the specified translation at the entry designated by the lock pointer, moves the lock pointer to the next sequential entry, and resets the round-robin pointer to entry 31. Locking entries into either TLB effectively reduces the available entries for updating. For example, if the first three entries were locked down, the round-robin pointer would be entry 3 after it rolled over from entry 31.

Only entries 0 through 30 can be locked in either TLB; entry 31can never be locked. If the lock pointer is at entry 31, a lock operation will update the TLB entry with the translation and ignore the lock. In this case, the round-robin pointer will stay at entry 31.

Figure 7. Example of Locked Entries in TLB



3.2 Instruction Cache

The Intel XScale processor instruction cache enhances performance by reducing the number of instruction fetches from external memory. The cache provides fast execution of cached code. Code can also be locked down when guaranteed or fast access time is required.

Figure 8 shows the cache organization and how the instruction address is used to access the cache.

The instruction cache is available as a 32 K, 32-way set, associative cache**.** Each set is 1,024 bytes in size. Each set contains 32 ways. Each way of a set contains eight 32-bit words and one valid bit, which is referred to as a line. The replacement policy is a round-robin algorithm and the cache also supports the ability to lock code in at a line granularity.

The instruction cache is virtually addressed and virtually tagged.

Note: The virtual address presented to the instruction cache may be remapped by the PID register. For a description of the PID register, see “Register 13: Process ID” on page 84.

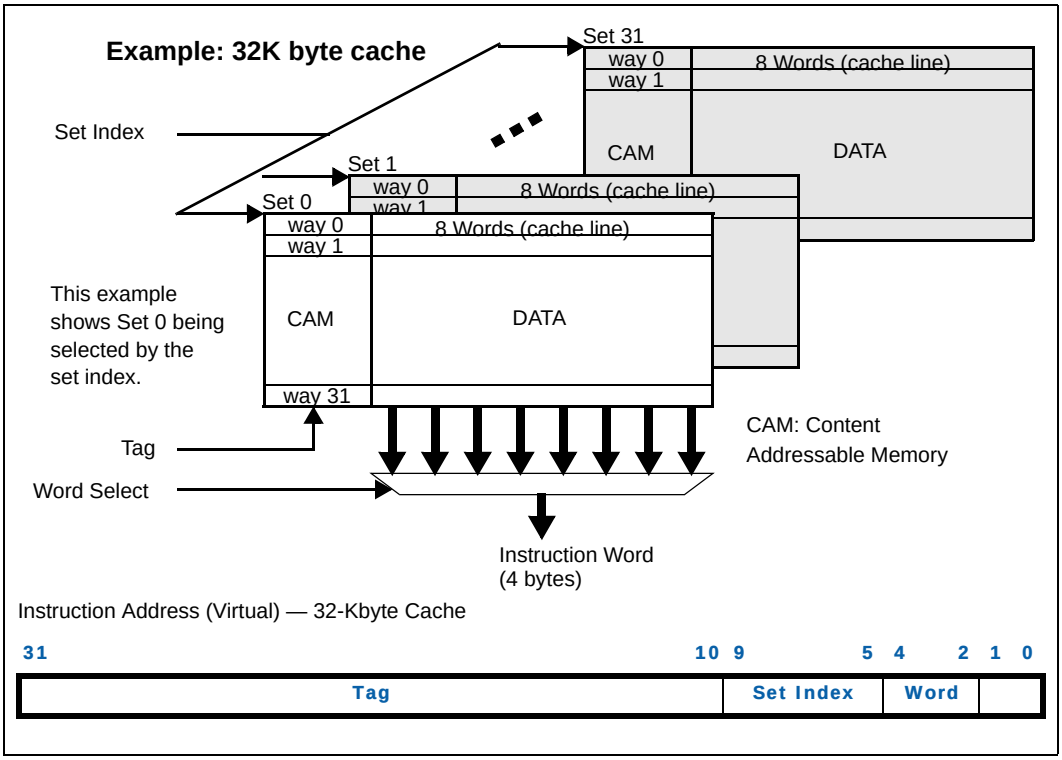
3.2.1 Operation When Instruction Cache is Enabled

When the cache is enabled, it compares every instruction request address against the addresses of instructions that it is currently holding. If the cache contains the requested instruction, the access “hits” the cache, and the cache returns the requested instruction. If the cache does not contain the requested instruction, the access “misses” the cache. The cache requests an eight-word, also known as a line, fetch from external memory that contains the requested instruction using the fetch policy described in “Instruction-Cache ‘Miss’” on page 53. As the fetch returns instructions to the cache, the instructions are placed in one of two fetch buffers and the requested instruction is delivered to the instruction decoder.

A fetched line will be written into the cache if it is cacheable. Code is designated as cacheable when the Memory Management Unit (MMU) is disabled or when the MMU is enable and the cacheable (C) bit is set to 1 in its corresponding page. See "Memory Management Unit" on page 44 for a discussion on page attributes.

Note that an instruction fetch may "miss" the cache but "hit" one of the fetch buffers. When this happens, the requested instruction will be delivered to the instruction decoder in the same manner as a cache "hit."

Figure 8. Instruction Cache Organization



Disabling the cache prevents any lines from being written into the instruction cache. Although the cache is disabled, it is still accessed and may generate a "hit" if the data is already in the cache.

Disabling the instruction cache ***does not*** disable instruction buffering that may occur within the instruction fetch buffers. Two 8-word instruction fetch buffers will always be enabled in the cache disabled mode. So long as instruction fetches continue to "hit" within either buffer (even in the presence of forward and backward branches), no external fetches for instructions are generated. A miss causes one or the other buffer to be filled from external memory using the fill policy described in "Instruction-Cache 'Miss'" on page 53.

3.2.1.1 Instruction-Cache 'Miss'

An instruction-cache "miss" occurs when the requested instruction is not found in the instruction fetch buffers or instruction cache; a fetch request is then made to external memory. The instruction cache can handle up to two "misses." Each external fetch request uses a fetch buffer that holds 32-bytes and eight valid bits, one for each word.

A miss causes the following:

- A fetch buffer is allocated
- The instruction cache sends a fetch request to the external bus. This request is for a 32-byte line.
- Instructions words are returned back from the external bus, at a maximum rate of 1 word per core cycle. The instruction cache can have the eight words of data return in any order, which allows the Intel XScale processor to send the requested instruction first, thus reducing fetch latency. (This is referred to as critical word first.)As each word returns, the corresponding valid bit is set for the word in the fetch buffer.
- As soon as the fetch buffer receives the requested instruction, it forwards the instruction to the instruction decoder for execution.
- When all words have returned, the fetched line will be written into the instruction cache if it's cacheable and if the instruction cache is enabled. The line chosen for update in the cache is controlled by the round-robin replacement algorithm. This update may evict a valid line at that location. For more information on enabling or disabling instruction cache, refer to "Instruction-Cache Coherence" on page 55

1. Once the cache is updated, the eight valid bits of the fetch buffer are invalidated.

3.2.1.2 Instruction-Cache Line-Replacement Algorithm

The line replacement algorithm for the instruction cache is round-robin. Each set in the instruction cache has a round-robin pointer that keeps track of the next line (in that set) to replace. The next line to replace in a set is the one after the last line that was written. For example, if the line for the last external instruction fetch was written into way 5-set 2, the next line to replace for that set would be way 6. None of the other round-robin pointers for the other sets are affected in this case.

After reset, way 31 is pointed to by the round-robin pointer for all the sets. Once a line is written into way 31, the round-robin pointer points to the first available way of a set, beginning with way0 if no lines have been locked into that particular set. Locking lines into the instruction cache effectively reduces the available lines for cache updating. For example, if the first three lines of a set were locked down, the round-robin pointer would point to the line at way 3 after it rolled over from way 31. For more details on cache locking, see "Instruction-Cache Coherence" on page 55.

The instruction cache is protected by parity to ensure data integrity. Each instruction cache word has 1 parity bit. (The instruction cache tag is NOT parity protected.) When a parity error is detected on an instruction cache access, a prefetch abort exception occurs if the Intel XScale processor attempts to execute the instruction. Before servicing the exception, hardware places a notification of the error in the Fault Status Register (Coprocessor 15, register 5).

A software exception handler can recover from an instruction cache parity error. The parity error can be accomplished by invalidating the instruction cache and the branch target buffer and then returning to the instruction that caused the prefetch abort exception. A simplified code example is shown in Example 4 on page 55. A more complex handler might choose to invalidate the specific line that caused the exception and then invalidate the BTB.

Example 4. Recovering from an Instruction Cache Parity Error

```
; Prefetch abort handler

MCR P15,0,R0,C7,C5,0      ; Invalidate the instruction cache and branch target

                          ; buffer



CPWAIT                    ; wait for effect (see "Additions to CP15 Functionality"
on page 153 for a

                        ; description of CPWAIT)



SUBS PC,R14,#4          ; Returns to the instruction that generated the

                          ; parity error



; The Instruction Cache is guaranteed to be invalidated at this point
```

If a parity error occurs on an instruction that is locked in the cache, the software exception handler needs to unlock the instruction cache, invalidate the cache and then re-lock the code in before it returns to the faulting instruction.

The instruction cache does not detect modification to program memory by loads, stores or actions of other bus masters. Several situations may require program memory modification, such as uploading code from disk.

The application program is responsible for synchronizing code modification and invalidating the cache. In general, software must ensure that modified code space is not accessed until modification and invalidating are completed.

3.2.1.3 Instruction-Cache Coherence

To achieve cache coherence, instruction cache contents can be invalidated after code modification in external memory is complete.

If the instruction cache is not enabled, or code is being written to a non-cacheable region, software must still invalidate the instruction cache before using the newly-written code. This precaution ensures that state associated with the new code is not buffered elsewhere in the processor, such as the fetch buffers or the BTB.

Naturally, when writing code as data, care must be taken to force it completely out of the processor into external memory before attempting to execute it. If writing into a non-cacheable region, flushing the write buffers is sufficient precaution (see "Register 7: Cache Functions" on page 81 for a description of this operation). If writing to a cacheable region, then the data cache should be submitted to a Clean/Invalidate operation (see "Cacheability" on page 63) to ensure coherency.

After reset, the instruction cache is always disabled, unlocked, and invalidated (flushed).

The instruction cache is enabled by setting bit 12 in coprocessor 15, register 1 (Control Register). This process is illustrated in Example 5, Enabling the Instruction Cache.

Example 5. Enabling the Instruction Cache

```
; Enable the ICache

MRC P15, 0, R0, C1, C0, 0            ; Get the control register

ORR R0, R0, #0x1000                  ; set bit 12 -- the I bit

MCR P15, 0, R0, C1, C0, 0            ; Set the control register


CPWAIT
```

The entire instruction cache along with the fetch buffers are invalidated by writing to coprocessor 15, register 7. (See Table 18, “Cache Functions” on page 81 for the exact command.) The invalidate command does not unlock any lines that were locked in the instruction cache nor does it invalidate those locked lines. To invalidate the entire cache including locked lines, the unlock instruction cache command needs to be executed before the invalidate command. The unlock command can also be found in Table 20, “Cache Lock-Down Functions” on page 83.

There is an inherent delay from the execution of the instruction cache invalidate command to where the next instruction will see the result of the invalidate. The following routine can be used to guarantee proper synchronization.

Example 6. Invalidating the Instruction Cache

```
MCR P15,0,R1,C7,C5,0      ; Invalidate the instruction cache and branch

                          ; target buffer


CPWAIT


; The instruction cache is guaranteed to be invalidated at this point; the next

; instruction sees the result of the invalidate command.
```

The Intel XScale processor also supports invalidating an individual line from the instruction cache. See Table 18, “Cache Functions” on page 81 for the exact command.

Software has the ability to lock performance critical routines into the instruction cache. Up to 28 lines in each set can be locked; hardware will ignore the lock command if software is trying to lock all the lines in a particular set (i.e., ways 28-31can never be locked). When all ways in a particular set are requested to be locked, the instruction cache line will still be allocated into the cache but the lock will be ignored. The round-robin pointer will stay at way 31 for that set.

Cache lines can be locked into the instruction cache by initiating a write to coprocessor 15. (See Table 20, “Cache Lock-Down Functions” on page 83 for the exact command.) Register *Rd* contains the virtual address of the line to be locked into the cache.

There are several requirements for locking down code:

- The routine used to lock lines down in the cache must be placed in non-cacheable memory, which means the MMU is enabled.

As a result: no fetches of cacheable code should occur while locking instructions into the cache.

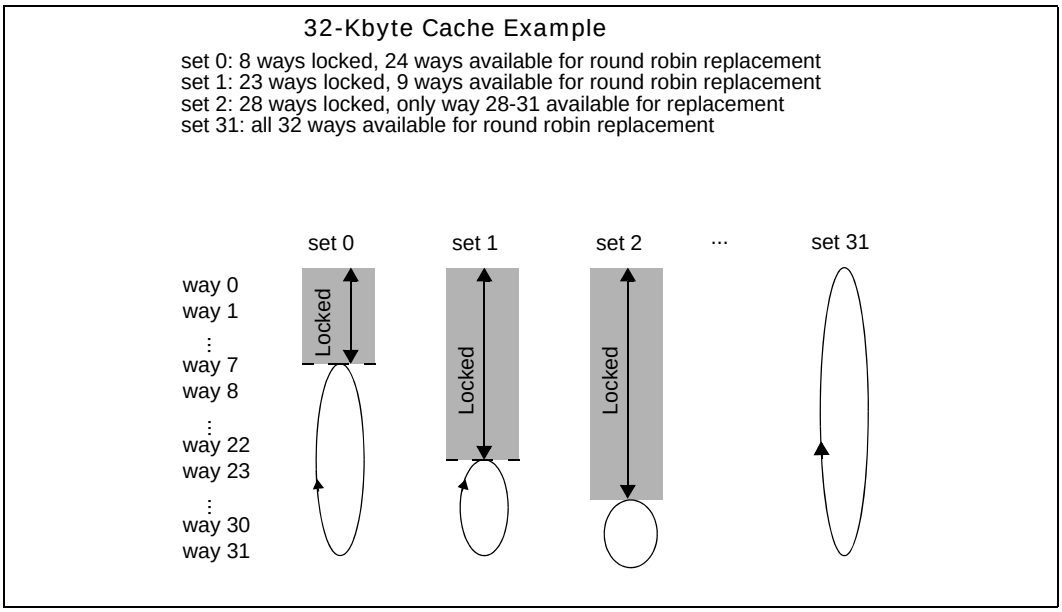
- The code being locked into the cache must be cacheable
- The instruction cache must be enabled and invalidated prior to locking down lines.

Failure to follow these requirements will produce unpredictable results when accessing the instruction cache.

System programmers should ensure that the code to lock instructions into the cache does not reside closer than 128 bytes to a non-cacheable/cacheable page boundary. If the processor fetches ahead into a cacheable page, then the first requirement noted above could be violated.

Lines are locked into a set starting at way 0 and may progress up to way 27; which set a line gets locked into depends on the set index of the virtual address. Figure 9 is an example (32-Kbyte cache) of where lines of code may be locked into the cache along with how the round-robin pointer is affected.

Figure 9. Locked Line Effect on Round-Robin Replacement



Software can lock down several different routines located at different memory locations. This may cause some sets to have more locked lines than others as shown in Figure 9.

Example 7 on page 58 shows how a routine, called “lockMe” in this example, might be locked into the instruction cache. Note that it is possible to receive an exception while locking code (see “Event Architecture” on page 154).

Example 7. Locking Code into the Cache

```
lockMe:                  ; This is the code that will be locked into the cache
       mov r0, #5
       add r5, r1, r2
   . . .
lockMeEnd:
       . . .


codeLock:                ; here is the code to lock the “lockMe” routine
       ldr r0, =(lockMe AND NOT 31); r0 gets a pointer to the first line we
       should lock
       ldr r1, =(lockMeEnd AND NOT 31); r1 contains a pointer to the last line we
       should lock


lockLoop:
       mcr p15, 0, r0, c9, c1, 0; lock next line of code into ICache
       cmp r0, r1        ; are we done yet?
       add r0, r0, #32   ; advance pointer to next line
       bne lockLoop      ; if not done, do the next line
```

The Intel XScale processor provides a global unlock command for the instruction cache. Writing to coprocessor 15, register 9 unlocks all the locked lines in the instruction cache and leaves them valid. These lines then become available for the round-robin replacement algorithm. (See Table 20, “Cache Lock-Down Functions” on page 83 for the exact command.)

3.3 Branch Target Buffer

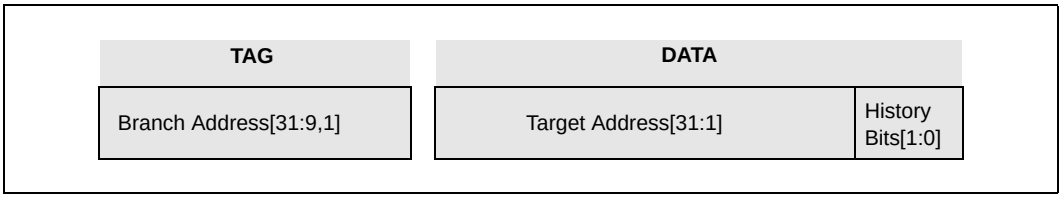
The Intel XScale processor uses dynamic branch prediction to reduce the penalties associated with changing the flow of program execution. The Intel XScale processor features a branch target buffer that provides the instruction cache with the target address of branch type instructions. The branch target buffer is implemented as a 128-entry, direct-mapped cache.

This section is primarily for those optimizing their code for performance. An understanding of the branch target buffer is needed in this case so that code can be scheduled to best utilize the performance benefits of the branch target buffer.

3.3.1 Branch Target Buffer (BTB) Operation

The BTB stores the history of branches that have executed along with their targets. Figure 10 shows an entry in the BTB, where the tag is the instruction address of a previously executed branch and the data contains the target address of the previously executed branch along with two bits of history information.

Figure 10. BTB Entry



The BTB takes the current instruction address and checks to see if this address is a branch that was previously seen. The BTB uses bits [8:2] of the current address to read out the tag and then compares this tag to bits [31:9,1] of the current instruction address. If the current instruction address matches the tag in the cache and the history bits indicate that this branch is usually taken in the past, the BTB uses the data (target address) as the next instruction address to send to the instruction cache.

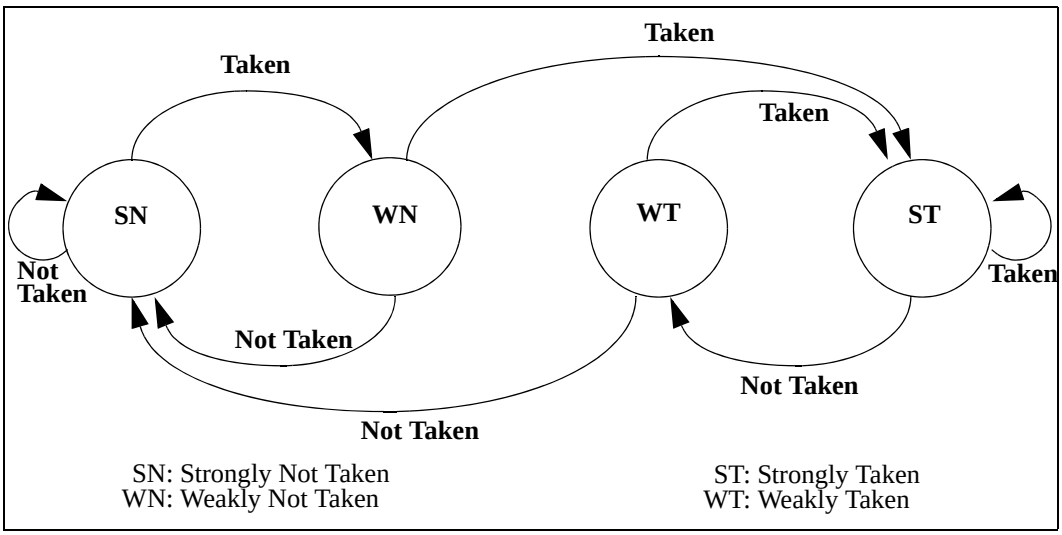
Bit[1] of the instruction address is included in the tag comparison in order to support Thumb execution. This organization means that two consecutive Thumb branch (B) instructions, with instruction address bits[8:2] the same, will contend for the same BTB entry. Thumb also requires 31 bits for the branch target address. In ARM mode, bit[1] is zero.

The history bits represent four possible prediction states for a branch entry in the BTB. Figure 11, "Branch History" on page 59 shows these states along with the possible transitions. The initial state for branches stored in the BTB is Weakly-Taken (WT). Every time a branch that exists in the BTB is executed, the history bits are updated to reflect the latest outcome of the branch, either taken or not-taken.

"Performance Considerations" on page 159 describes which instructions are dynamically predicted by the BTB and the performance penalty for incorrectly predicting a branch.

The BTB does not have to be managed explicitly by software; it is disabled by default after reset and is invalidated when the instruction cache is invalidated.

Figure 11. Branch History



3.3.1.1 Reset

After Processor Reset, the BTB is disabled and all entries are invalidated.

A new entry is stored into the BTB when the following conditions are met:

- The branch instruction has executed
- The branch was taken
- The branch is not currently in the BTB

The entry is then marked valid and the history bits are set to WT. If another valid branch exists at the same entry in the BTB, it will be evicted by the new branch.

Once a branch is stored in the BTB, the history bits are updated upon every execution of the branch as shown in Figure 11.

The BTB is always disabled with Reset. Software can enable the BTB through a bit in a coprocessor register (see “Register 1: Control and Auxiliary Control Registers” on page 77).

Before enabling or disabling the BTB, software must invalidate the BTB (described in the following section). This action will ensure correct operation in case stale data is in the BTB. Software should not place any branch instruction between the code that invalidates the BTB and the code that enables/disables it.

There are four ways the contents of the BTB can be invalidated.

- Reset
- Software can directly invalidate the BTB via a CP15, register 7 function. Refer to “Register 7: Cache Functions” on page 81.
- The BTB is invalidated when the Process ID Register is written.
- The BTB is invalidated when the instruction cache is invalidated via CP15, register 7 functions.

3.4 Data Cache

The Intel XScale processor data cache enhances performance by reducing the number of data accesses to and from external memory. There are two data cache structures in the Intel XScale processor: a 32-Kbyte data cache and a 2-Kbyte mini-data cache. An eight entry write buffer and a four-entry, fill buffer are also implemented to decouple the Intel XScale processor instruction execution from external memory accesses, which increases overall system performance.

3.4.1 Data Cache Overview

The data cache is a 32-Kbyte, 32-way set, associative cache. The 32-Kbyte cache has 32 sets. Each set contains 32 ways. Each way of a set contains 32 bytes (one cache line) and one valid bit. There also exist two dirty bits for every line, one for the lower 16 bytes and the other one for the upper 16 bytes. When a store hits the cache the dirty bit associated with it is set. The replacement policy is a round-robin algorithm and the cache also supports the ability to reconfigure each line as data RAM.

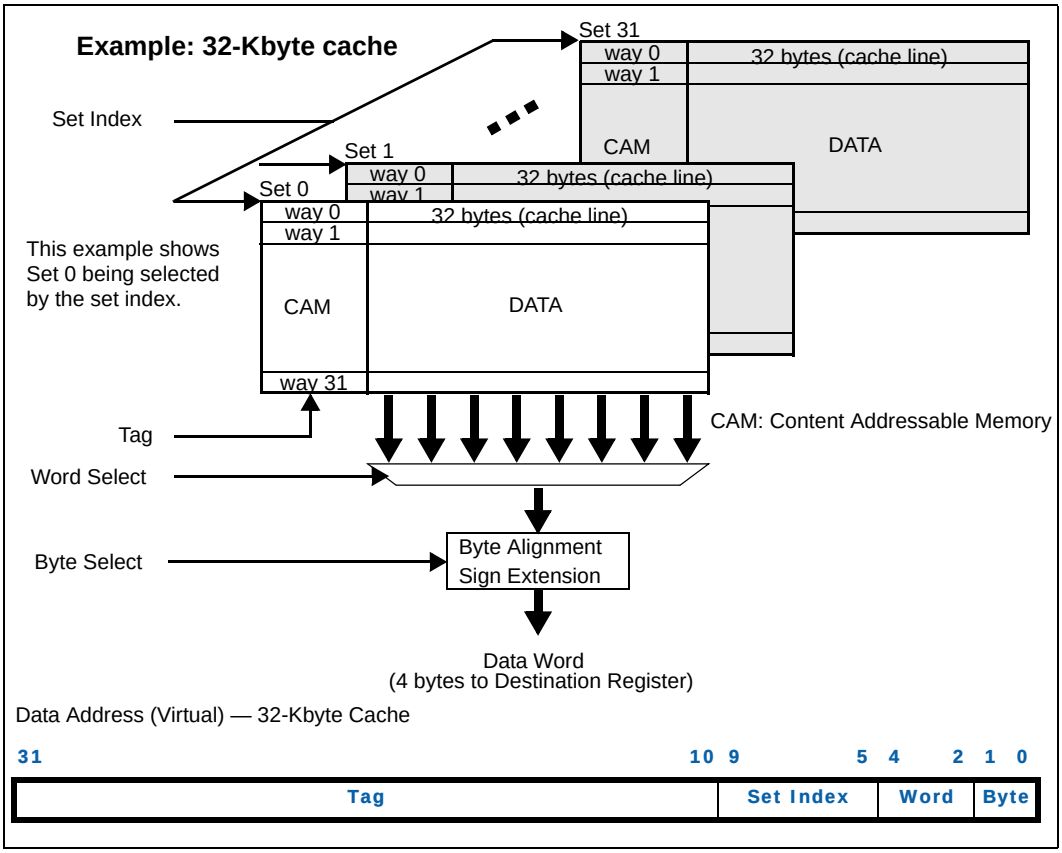
Figure 12, “Data Cache Organization” on page 61 shows the cache organization and how the data address is used to access the cache.

Cache policies may be adjusted for particular regions of memory by altering page attribute bits in the MMU descriptor that controls that memory. See “Memory Attributes” on page 45 for a description of these bits.

The data cache is virtually addressed and virtually tagged. The data cache supports write-back and write-through caching policies. The data cache always allocates a line in the cache when a cacheable read miss occurs and will allocate a line into the cache on

a cacheable write miss when write allocate is specified by its page attribute. Page attribute bits determine whether a line gets allocated into the data cache or mini-data cache.

Figure 12. Data Cache Organization

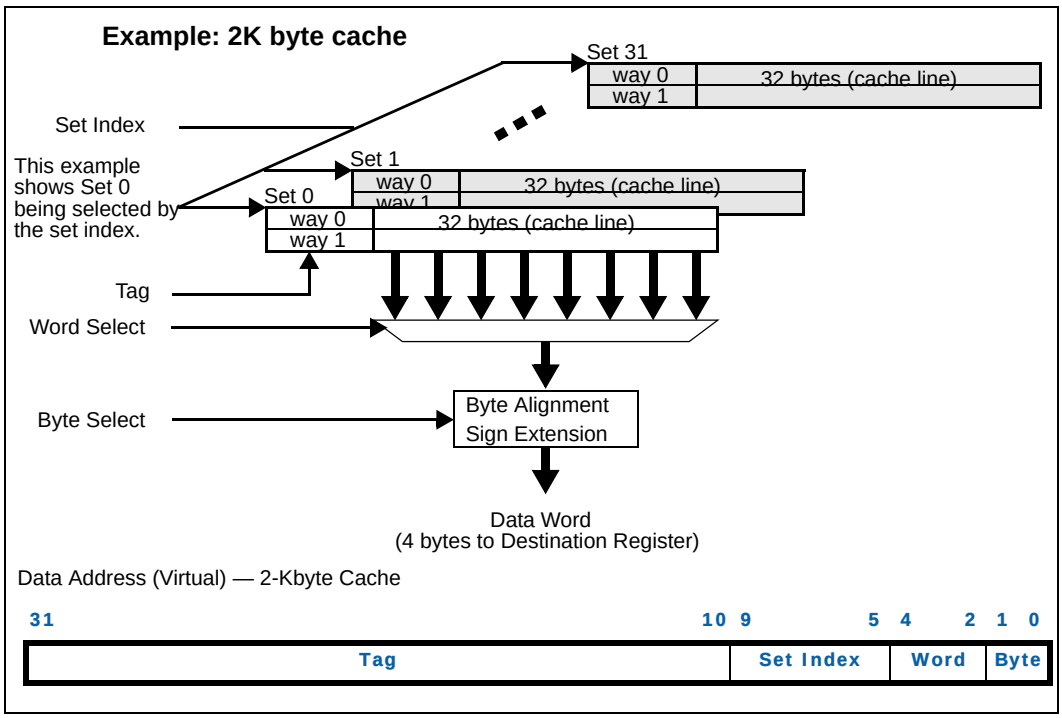


The mini-data cache is 2 Kbytes in size. The 2-Kbyte mini data cache has 32 sets and is two-way set associative. Each way of a set contains 32 bytes (one cache line) and one valid bit. There also exist two dirty bits for every line, one for the lower 16 bytes and the other one for the upper 16 bytes. When a store hits the cache the dirty bit associated with it is set. The replacement policy is a round-robin algorithm.

Figure 13, “Mini-Data Cache Organization” on page 62 shows the cache organization and how the data address is used to access the cache.

The mini-data cache is virtually addressed and virtually tagged and supports the same caching policies as the data cache. However, lines can’t be locked into the mini-data cache.

Figure 13. Mini-Data Cache Organization



The Intel XScale processor employs an eight entry write buffer, each entry containing 16 bytes. Stores to external memory are first placed in the write buffer and subsequently taken out when the bus is available.

The write buffer supports the coalescing of multiple store requests to external memory. An incoming store may coalesce with any of the eight entries.

The fill buffer holds the external memory request information for a data cache or mini-data cache fill or non-cacheable read request. Up to four 32-byte read request operations can be outstanding in the fill buffer before the Intel XScale processor needs to stall.

The fill buffer has been augmented with a four entry pend buffer that captures data memory requests to outstanding fill operations. Each entry in the pend buffer contains enough data storage to hold one 32-bit word, specifically for store operations. Cacheable load or store operations that hit an entry in the fill buffer get placed in the pend buffer and are completed when the associated fill completes. Any entry in the pend buffer can be pended against any of the entries in the fill buffer; multiple entries in the pend buffer can be pended against a single entry in the fill buffer.

Pended operations complete in program order.

The following discussions refer to the data cache and mini-data cache as one cache (data/mini-data) since their behavior is the same when accessed.

When the data/mini-data cache is enabled for an access, the data/mini-data cache compares the address of the request against the addresses of data that it is currently holding. If the line containing the address of the request is resident in the cache, the access “hits” the cache. For a load operation the cache returns the requested data to the destination register and for a store operation the data is stored into the cache. The data associated with the store may also be written to external memory if write-through

caching is specified for that area of memory. If the cache does not contain the requested data, the access 'misses' the cache, and the sequence of events that follows depends on the configuration of the cache, the configuration of the MMU and the page attributes, which are described in "Cacheability" on page 63.

The data/mini-data cache is still accessed even though it is disabled. If a load hits the cache it will return the requested data to the destination register. If a store hits the cache, the data is written into the cache. Any access that misses the cache will not allocate a line in the cache when it's disabled, even if the MMU is enabled and the memory region's cacheability attribute is set.

3.4.2 Cacheability

Data at a specified address is cacheable given the following:

- the MMU is enabled
- the cacheable attribute is set in the descriptor for the accessed address
- and the data/mini-data cache is enabled

The following sequence of events occurs when a cacheable (see "Cacheability" on page 63) load operation misses the cache:

1. The fill buffer is checked to see if an outstanding fill request already exists for that line.

   If so, the current request is placed in the pending buffer and waits until the previously requested fill completes, after which it accesses the cache again, to obtain the request data and returns it to the destination register.

   If there is no outstanding fill request for that line, the current load request is placed in the fill buffer and a 32-byte external memory read request is made. If the pending buffer or fill buffer is full, the Intel XScale processor will stall until an entry is available.
2. A line is allocated in the cache to receive the 32-bytes of fill data. The line selected is determined by the round-robin pointer. (See "Cacheability" on page 63.) The line chosen may contain a valid line previously allocated in the cache. In this case both dirty bits are examined and if set, the four words associated with a dirty bit that's asserted will be written back to external memory as a four word burst operation.
3. When the data requested by the load is returned from external memory, it is immediately sent to the destination register specified by the load. A system that returns the requested data back first, with respect to the other bytes of the line, will obtain the best performance.
4. As data returns from external memory it is written into the cache in the previously allocated line.

A load operation that misses the cache and is NOT cacheable makes a request from external memory for the exact data size of the original load request. For example, **LDRH** requests exactly two bytes from external memory, **LDR** requests 4 bytes from external memory, etc. This request is placed in the fill buffer until, the data is returned from external memory, which is then forwarded back to the destination register(s).

A write operation that misses the cache will request a 32-byte cache line from external memory if the access is cacheable and write allocation is specified in the page. In this case the following sequence of events occur:

1. The fill buffer is checked to see if an outstanding fill request already exists for that line.

   If so, the current request is placed in the pending buffer and waits until the previously requested fill completes, after which it writes its data into the recently allocated cache line.

If there is no outstanding fill request for that line, the current store request is placed in the fill buffer and a 32-byte external memory read request is made. If the pending buffer or fill buffer is full, the Intel XScale processor will stall until an entry is available.

2. The 32-bytes of data can be returned back to the Intel XScale processor in any word order, i.e, the eight words in the line can be returned in any order. Note that it does not matter, for performance reasons, which order the data is returned to the Intel XScale processor since the store operation has to wait until the entire line is written into the cache before it can complete.
3. When the entire 32-byte line has returned from external memory, a line is allocated in the cache, selected by the round-robin pointer. The line to be written into the cache may replace a valid line previously allocated in the cache. In this case both dirty bits are examined and if any are set, the four words associated with a dirty bit that's asserted will be written back to external memory as a 4 word burst operation. This write operation will be placed in the write buffer.
4. The line is written into the cache along with the data associated with the store operation.

If the above condition for requesting a 32-byte cache line is not met, a write miss will cause a write request to external memory for the exact data size specified by the store operation, assuming the write request doesn't coalesce with another write operation in the write buffer.

The Intel XScale processor supports write-back caching or write-through caching, controlled through the MMU page attributes. When write-through caching is specified, all store operations are written to external memory even if the access hits the cache. This feature keeps the external memory coherent with the cache, i.e., no dirty bits are set for this region of memory in the data/mini-data cache. However write through does not guarantee that the data/mini-data cache is coherent with external memory, which is dependent on the system level configuration, specifically if the external memory is shared by another master.

When write-back caching is specified, a store operation that hits the cache will not generate a write to external memory, thus reducing external memory traffic.

The line replacement algorithm for the data cache is round-robin. Each set in the data cache has a round-robin pointer that keeps track of the next line (in that set) to replace. The next line to replace in a set is the next sequential line after the last one that was just filled. For example, if the line for the last fill was written into way 5-set 2, the next line to replace for that set would be way 6. None of the other round-robin pointers for the other sets are affected in this case.

After reset, way 31 is pointed to by the round-robin pointer for all the sets. Once a line is written into way 31, the round-robin pointer points to the first available way of a set, beginning with way 0 if no lines have been re-configured as data RAM in that particular set. Re-configuring lines as data RAM effectively reduces the available lines for cache updating. For example, if the first three lines of a set were re-configured, the round-robin pointer would point to the line at way 3 after it rolled over from way 31. Refer to "Reconfiguring the Data Cache as Data RAM" on page 68 for more details on data RAM.

The mini-data cache follows the same round-robin replacement algorithm as the data cache except that there are only two lines the round-robin pointer can point to such that the round-robin pointer always points to the least recently filled line. A least recently used replacement algorithm is not supported because the purpose of the mini-data cache is to cache data that exhibits low temporal locality, i.e.,data that is placed into the mini-data cache is typically modified once and then written back out to external memory.

The data cache and mini-data cache are protected by parity to ensure data integrity; there is one parity bit per byte of data. (The tags are NOT parity protected.) When a parity error is detected on a data/mini-data cache access, a data abort exception occurs. Before servicing the exception, hardware will set bit 10 of the Fault Status Register register.

A data/mini-data cache parity error is an imprecise data abort, meaning R14_ABORT (+8) may not point to the instruction that caused the parity error. If the parity error occurred during a load, the targeted register may be updated with incorrect data.

A data abort due to a data/mini-data cache parity error may not be recoverable if the data address that caused the abort occurred on a line in the cache that has a write-back caching policy. Prior updates to this line may be lost; in this case the software exception handler should perform a “clean and clear” operation on the data cache, ignoring subsequent parity errors, and restart the offending process.

The **SWP** and **SWPB** instructions generate an atomic load and store operation allowing a memory semaphore to be loaded and altered without interruption. These accesses may hit or miss the data/mini-data cache depending on configuration of the cache, configuration of the MMU, and the page attributes.

After processor reset, both the data cache and mini-data cache are disabled, all valid bits are set to zero (invalid), and the round-robin bit points to way 31. Any lines in the data cache that were configured as data RAM before reset are changed back to cacheable lines after reset, i.e., there are 32 Kbytes of data cache and zero bytes of data RAM.

The data cache and mini-data cache are enabled by setting bit 2 in coprocessor 15, register 1 (Control Register). See “Configuration” on page 73, for a description of this register and others.

Equation 8 shows code that enables the data and mini-data caches. Note that the MMU must be enabled to use the data cache.

Example 8. Enabling the Data Cache

```
enableDCache:


       MCR p15, 0, r0, c7, c10, 4; Drain pending data operations...

                          ; (see Chapter 7.2.8, Register 7: Cache functions)

       MRC p15, 0, r0, c1, c0, 0; Get current control register

       ORR r0, r0, #4      ; Enable DCache by setting ‘C’ (bit 2)

       MCR p15, 0, r0, c1, c0, 0; And update the Control register
```

Individual entries can be invalidated and cleaned in the data cache and mini-data cache via coprocessor 15, register 7. Note that a line locked into the data cache remains locked even after it has been subjected to an invalidate-entry operation. This will leave an unusable line in the cache until a global unlock has occurred. For this reason, do not use these commands on locked lines.

This same register also provides the command to invalidate the entire data cache and mini-data cache. Refer to Table 18, “Cache Functions” on page 81 for a listing of the commands. These global invalidate commands have no effect on lines locked in the data cache. Locked lines must be unlocked before they can be invalidated. This is accomplished by the Unlock Data Cache command found in Table 20, “Cache Lock-Down Functions” on page 83.

A simple software routine is used to globally clean the data cache. It takes advantage of the line-allocate data cache operation, which allocates a line into the data cache. This allocation evicts any cache dirty data back to external memory. Example 9 shows how data cache can be cleaned.

Example 9. Global Clean Operation

```
; Global Clean/Invalidate THE DATA CACHE
; R1 contains the virtual address of a region of cacheable memory reserved for
; this clean operation
; R0 is the loop count; Iterate 1024 times which is the number of lines in the
; data cache


;; Macro ALLOCATE performs the line-allocation cache operation on the
;; address specified in register Rx.
;;

MACRO ALLOCATE Rx

    MCR P15, 0, Rx, C7, C2, 5

ENDM


MOV  R0, #1024

LOOP1:

ALLOCATE R1               ; Allocate a line at the virtual address

                          ; specified by R1.

ADD R1, R1, #32        ; Increment the address in R1 to the next cache line

SUBS R0, R0, #1        ; Decrement loop count

BNE LOOP1

;

;Clean the Mini-data Cache

; Can't use line-allocate command, so cycle 2KB of unused data through.

; R2 contains the virtual address of a region of cacheable memory reserved for
; cleaning the Mini-data Cache

; R0 is the loop count; Iterate 64 times which is the number of lines in the
; Mini-data Cache.


MOV  R0, #64

LOOP2:

LDR R3,[R2],#32 ; Load and increment to next cache line

SUBS R0, R0, #1        ; Decrement loop count

BNE LOOP2

;


; Invalidate the data cache and mini-data cache

MCR P15, 0, R0, C7, C6, 0

;
```

The line-allocate operation does not require physical memory to exist at the virtual address specified by the instruction, since it does not generate a load/fill request to external memory. Also, the line-allocate operation does not set the 32 bytes of data associated with the line to any known value. Reading this data will produce unpredictable results.

The line-allocate command will not operate on the mini Data Cache, so system software must clean this cache by reading 2 Kbytes of contiguous unused data into it. This data must be unused and reserved for this purpose so that it will not already be in the cache. It must reside in a page that is marked as mini Data Cache cacheable (see "New Page Attributes" on page 152).

The time it takes to execute a global clean operation depends on the number of dirty lines in cache.

3.4.3 Reconfiguring the Data Cache as Data RAM

Software has the ability to lock tags associated with 32-byte lines in the data cache, thus creating the appearance of data RAM. Any subsequent access to this line will always hit the cache unless it is invalidated. Once a line is locked into the data cache it is no longer available for cache allocation on a line fill. Up to 28 lines in each set can be reconfigured as data RAM, such that the maximum data RAM size is 28 Kbytes for the 32-Kbyte cache and 12 Kbytes for the 16-Kbyte cache.

Hardware does not support locking lines into the mini-data cache; any attempt to do this will produce unpredictable results.

There are two methods for locking tags into the data cache; the method of choice depends on the application. One method is used to lock data that resides in external memory into the data cache and the other method is used to re-configure lines in the data cache as data RAM. Locking data from external memory into the data cache is useful for lookup tables, constants, and any other data that is frequently accessed. Re-configuring a portion of the data cache as data RAM is useful when an application needs scratch memory (bigger than the register file can provide) for frequently used variables. These variables may be strewn across memory, making it advantageous for software to pack them into data RAM memory.

Code examples for these two applications are shown in Example 10 on page 69 and Example 11 on page 70. The difference between these two routines is that Example 10 on page 69 actually requests the entire line of data from external memory and Example 11 on page 70 uses the line-allocate operation to lock the tag into the cache. No external memory request is made, which means software can map any unallocated area of memory as data RAM. However, the line-allocate operation does validate the target address with the MMU, so system software must ensure that the memory has a valid descriptor in the page table.

Another item to note in Example 11 on page 70 is that the 32 bytes of data located in a newly allocated line in the cache must be initialized by software before it can be read. The line allocate operation does not initialize the 32 bytes and therefore reading from that line will produce unpredictable results.

In both examples, the code drains the pending loads before and after locking data. This step ensures that outstanding loads do not end up in the wrong place -- either unintentionally locked into the cache or mistakenly left out in the proverbial cold. Note also that a drain operation has been placed after the operation that locks the tag into the cache. This drains ensures predictable results if a programmer tries to lock more than 28 lines in a set; the tag will get allocated in this case but not locked into the cache.

Example 10. Locking Data into Data Cache

```
; R1 contains the virtual address of a region of memory to lock,
; configured with C=1 and B=1
; R0 is the number of 32-byte lines to lock into the data cache. In this
; example 16 lines of data are locked into the cache.
; MMU and data cache are enabled prior to this code.

MACRO DRAIN
    MCR P15, 0, R0, C7, C10, 4    ; drain pending loads and stores
ENDM

DRAIN

MOV  R2, #0x1
MCR  P15,0,R2,C9,C2,0         ; Put the data cache in lock mode
CPWAIT
MOV  R0, #16
LOOP1:
MCR  P15,0,R1,C7,C10,1        ; Write back the line if it’s dirty in the cache
MCR  P15,0,R1, C7,C6,1        ; Flush/Invalidate the line from the cache
LDR  R2, [R1], #32            ; Load and lock 32 bytes of data located at [R1]
                              ; into the data cache. Post-increment the address
                              ; in R1 to the next cache line.
SUBS R0, R0, #1; Decrement loop count
BNE LOOP1

          ; Turn off data cache locking
MOV  R2, #0x0
MCR  P15,0,R2,C9,C2,0         ; Take the data cache out of lock mode.
CPWAIT
```

Example 11. Creating Data RAM

```
; R1 contains the virtual address of a region of memory to configure as data RAM,
; which is aligned on a 32-byte boundary.
; MMU is configured so that the memory region is cacheable.
; R0 is the number of 32-byte lines to designate as data RAM. In this example 16
; lines of the data cache are re-configured as data RAM.
; The inner loop is used to initialize the newly allocated lines
; MMU and data cache are enabled prior to this code.
```

Example 11. Creating Data RAM

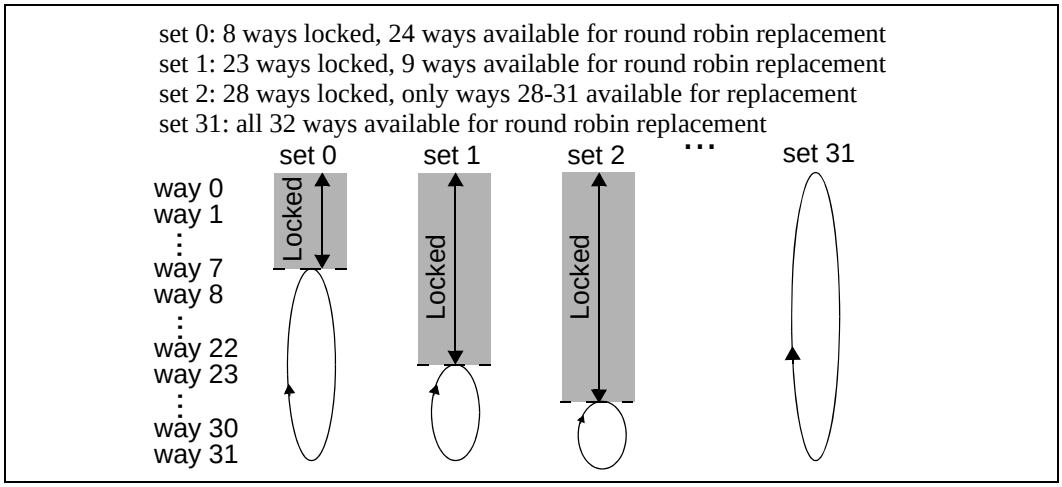
```
MACRO ALLOCATE Rx
    MCR P15, 0, Rx, C7, C2, 5
ENDM

MACRO DRAIN
    MCR P15, 0, R0, C7, C10, 4    ; drain pending loads and stores
ENDM

DRAIN
MOV R4, #0x0
MOV R5, #0x0
MOV R2, #0x1
MCR P15,0,R2,C9,C2,0                    ; Put the data cache in lock mode
CPWAIT

MOV R0, #16
LOOP1:
ALLOCATE R1               ; Allocate and lock a tag into the data cache at
                          ; address [R1].
; initialize 32 bytes of newly allocated line
DRAIN
STRD R4, [R1],#8      ;
STRD R4, [R1],#8   ;
STRD R4, [R1],#8      ;
STRD R4, [R1],#8      ;
SUBS R0, R0, #1           ; Decrement loop count
BNE LOOP1
; Turn off data cache locking
DRAIN                             ; Finish all pending operations
MOV R2, #0x0
MCR P15,0,R2,C9,C2,0; Take the data cache out of lock mode.
CPWAIT
```

Tags can be locked into the data cache by enabling the data cache lock mode bit located in coprocessor 15, register 9. (See Table 20, “Cache Lock-Down Functions” on page 83 for the exact command.) Once enabled, any new lines allocated into the data cache will be locked down.

Note that the **PLD** instruction will not affect the cache contents if it encounters an error while executing. For this reason, system software should ensure the memory address used in the **PLD** is correct. If this cannot be ascertained, replace the **PLD** with a **LDR** instruction that targets a scratch register.

Lines are locked into a set starting at way0 and may progress up to way 27; which set a line gets locked into depends on the set index of the virtual address of the request. Figure 9, “Locked Line Effect on Round-Robin Replacement” on page 57 is an example of where lines of code may be locked into the cache along with how the round-robin pointer is affected.

Figure 14. Locked Line Effect on Round-Robin Replacement



Software can lock down data located at different memory locations. This may cause some sets to have more locked lines than others as shown in Figure 9.

Lines are unlocked in the data cache by performing an unlock operation. See “Register 9: Cache Lock Down” on page 82 for more information about locking and unlocking the data cache.

Before locking, the programmer must ensure that no part of the target data range is already resident in the cache. The Intel XScale processor will not refetch such data, which will result in it not being locked into the cache. If there is any doubt as to the location of the targeted memory data, the cache should be cleaned and invalidated to prevent this scenario. If the cache contains a locked region which the programmer wishes to lock again, then the cache must be unlocked before being cleaned and invalidated.

See “Terminology and Conventions” on page 26 for a definition of coalescing.

The write buffer is always enabled which means stores to external memory will be buffered. The K bit in the Auxiliary Control Register (CP15, register 1) is a global enable/disable for allowing coalescing in the write buffer. When this bit disables coalescing, no coalescing will occur regardless the value of the page attributes. If this bit enables coalescing, the page attributes X, C, and B are examined to see if coalescing is enabled for each region of memory.

All reads and writes to external memory occur in program order when coalescing is disabled in the write buffer. If coalescing is enabled in the write buffer, writes may occur out of program order to external memory. Program correctness is maintained in this case by comparing all store requests with all the valid entries in the fill buffer.

The write buffer and fill buffer support a drain operation, such that before the next instruction executes, all Intel XScale processor data requests to external memory have completed. See Table 18, “Cache Functions” on page 81 for the exact command.

Writes to a region marked non-cacheable/non-bufferable (page attributes C, B, and X all 0) will cause execution to stall until the write completes.

If software is running in a privileged mode, it can explicitly drain all buffered writes. For details on this operation, see the description of Drain Write Buffer in “Register 7: Cache Functions” on page 81.

3.5 Configuration

This section describes the System Control Coprocessor (CP15) and coprocessor 14 (CP14). CP15 configures the MMU, caches, buffers and other system attributes. Where possible, the definition of CP15 follows the definition of the ARM products. CP14 contains the performance monitor registers, clock and power management registers and the debug registers.

CP15 is accessed through **MRC** and **MCR** coprocessor instructions and allowed only in privileged mode. Any access to CP15 in user mode or with **LDC** or **STC** coprocessor instructions will cause an undefined instruction exception.

All CP14 registers can be accessed through **MRC** and **MCR** coprocessor instructions. **LDC** and **STC** coprocessor instructions can only access the clock and power management registers, and the debug registers. The performance monitoring registers can’t be accessed by **LDC** and **STC** because CRm != 0x0. Access to all registers is allowed only in privileged mode. Any access to CP14 in user mode will cause an undefined instruction exception.

Coprocessors, CP15 and CP14, on the Intel XScale® Processor do not support access via **CDP**, **MRRC**, or **MCRR** instructions. An attempt to access these coprocessors with these instructions will result in an Undefined Instruction exception.

Many of the MCR commands available in CP15 modify hardware state sometime after execution. A software sequence is available for those wishing to determine when this update occurs and can be found in “Additions to CP15 Functionality” on page 153.

Like certain other ARM architecture products, the Intel XScale® Processor includes an extra level of virtual address translation in the form of a PID (Process ID) register and associated logic. For a detailed description of this facility, see “Register 13: Process ID” on page 84. Privileged code needs to be aware of this facility because, when interacting with CP15, some addresses are modified by the PID and others are not.

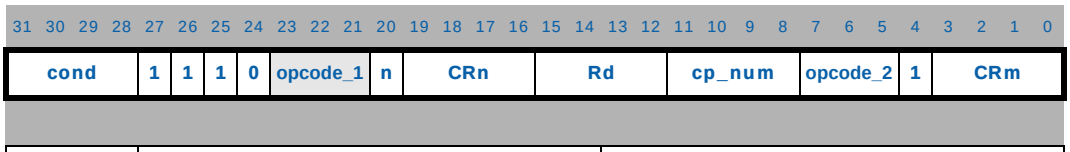
An address that has yet to be modified by the PID (“PIDified”) is known as a *virtual address* (VA). An address that has been through the PID logic, but not translated into a physical address, is a *modified virtual address* (MVA). Non-privileged code always deals with VAs, while privileged code that programs CP15 occasionally needs to use MVAs.

The format of **MRC** and **MCR** is shown in Table 7.

cp_num is defined for CP15, CP14 and CP0 on the Intel XScale processor. CP0 supports instructions specific for DSP and is described in “Programming Model” on page 144

Unless otherwise noted, unused bits in coprocessor registers have unpredictable values when read. For compatibility with future implementations, software should not rely on the values in those bits.

Table 7. MRC/MCR Format



Bits	Description	Notes
31:28	**cond** - ARM condition codes	-
23:21	**opcode_1** - Reserved	Should be programmed to zero for future compatibility
20	**n** - Read or write coprocessor register 0 = MCR 1 = MRC	-
19:16	**CRn** - specifies which coprocessor register	-
15:12	**Rd** - General Purpose Register, R0..R15	-
11:8	**cp_num** - coprocessor number	Intel XScale processor defines three coprocessors: 0b1111 = CP15 0b1110 = CP14 0x0000 = CP0 ***Note:*** Mappings are implementation defined for all coprocessors below CP14 and above CP0. Access to unimplemented coprocessors (as defined by the cpConfig bus) cause exceptions.
7:5	**opcode_2** - Function bits	This field should be programmed to zero for future compatibility unless a value has been specified in the command.
3:0	**CRm** - Function bits	This field should be programmed to zero for future compatibility unless a value has been specified in the command.

The format of **LDC** and **STC** for CP14 is shown in Table 8. **LDC** and **STC** follow the programming notes in the *ARM* Architecture Reference Manual*. Note that access to CP15 with **LDC** and **STC** will cause an undefined exception.

LDC and **STC** transfer a single 32-bit word between a coprocessor register and memory. These instructions do not allow the programmer to specify values for **opcode_1**, **opcode_2**, **or Rm**; those fields implicitly contain zero, which means the performance monitoring registers are not accessible.

Table 8. LDC/STC Format when Accessing CP14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8_bit_word_offset							

Bits	Description	Notes
31:28	**cond** - ARM condition codes	-
24:23,21	**P, U, W** - specifies 1 of 3 addressing modes identified by addressing mode 5 in the *ARM* Architecture Reference Manual*.	-
22	**N** - should be 0 for CP14 coprocessors. Setting this bit to 1 has will have an undefined effect.	
20	**L** - Load or Store 0 = STC 1 = LDC	-
19:16	**Rn** - specifies the base register	-
15:12	**CRd** - specifies the coprocessor register	-
11:8	**cp_num** - coprocessor number	Intel XScale processor defines the following: 0b1111 = Undefined Exception 0b1110 = CP14 ***Note:*** Mappings are implementation defined for all coprocessors below CP13. Access to unimplemented coprocessors (as defined by the cpConfig bus) cause exceptions.
7:0	8-bit word offset	-

3.5.1 CP15 Registers

Table 9 lists the CP15 registers implemented in Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor.

Table 9. CP15 Registers (Sheet 1 of 2)

Register (CRn)	Opcode_2	Access	Description
0	0	Read / Write-Ignored	ID
0	1	Read / Write-Ignored	Cache Type
1	0	Read / Write	Control
1	1	Read / Write	Auxiliary Control
2	0	Read / Write	Translation Table Base
3	0	Read / Write	Domain Access Control
4	-	Unpredictable	Reserved
5	0	Read / Write	Fault Status
6	0	Read / Write	Fault Address
7	0	Read-unpredictable / Write	Cache Operations
8	0	Read-unpredictable / Write	TLB Operations
9	0	Read / Write	Cache Lock Down
10	0	Read-unpredictable / Write	TLB Lock Down

Table 9. CP15 Registers (Sheet 2 of 2)

Register (CRn)	Opcode_2	Access	Description
11 - 12	-	Unpredictable	Reserved
13	0	Read / Write	Process ID (PID)
14	0	Read / Write	Breakpoint Registers
15	0	Read / Write	(CRm = 1) CP Access

3.5.1.1 Register 0: ID and Cache Type Registers

Register 0 houses two read-only register that are used for part identification: an ID register and a cache type register.

The ID Register is selected when *opcode_2*=0. This register returns the code for the Intel® IXP42X product line and IXC1100 control plane processors.

Table 10. ID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	0	0	0	0	0	1	0	1	Core Gen			Core Revision				Product Number								Product Revision

reset value: As Shown

Bits	Access	Description
31:24	Read / Write Ignored	Implementation trademark (0x69 = 'i'= Intel Corporation)
23:16	Read / Write Ignored	Architecture version = ARM Version 5TE (0x05 will be value returned)
15:13	Read / Write Ignored	Core Generation 0b010 = Intel XScale processor This field reflects a specific set of architecture features supported by the core. If new features are added/ deleted/modified, this field will change.
12:10	Read / Write Ignored	Intel XScale processor Revision: This field reflects revisions of core generations. Differences may include errata that dictate different operating conditions, software work-around, etc. Value returned will be 000b
9:4	Read / Write Ignored	Product Number for: IXP42X 533-MHz processor - 011100b IXP42X 400-MHz processor - 011101b IXP42X 266-MHz processor - 011111b
3:0	Read / Write Ignored	Product Revision for:IXP42X product line IXP42X 533-MHz processor - 0001b IXP42X 400-MHz processor - 0001b IXP42X 266-MHz processor - 0001b

The Cache Type Register is selected when *opcode_2*=1 and describes the cache configuration of the Intel XScale processor.

Table 11. Cache Type Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	0	0	Dsize			1	0	1	0	1	0	0	0	0	Isize			1	0	1	0	1	0

reset value: As Shown

Bits	Access	Description
31:29	Read-as-Zero / Write Ignored	Reserved
28:25	Read / Write Ignored	Cache class = 0b0101 The caches support locking, write back and round-robin replacement. They do not support address by index.
24	Read / Write Ignored	Harvard Cache
23:21	Read-as-Zero / Write Ignored	Reserved
20:18	Read / Write Ignored	Data Cache Size (Dsize) 0b110 = 32 KB
17:15	Read / Write Ignored	Data cache associativity = 0b101 = 32-way
14	Read-as-Zero / Write Ignored	Reserved
13:12	Read / Write Ignored	Data cache line length = 0b10 = 8 words/line
11:9	Read-as-Zero / Write Ignored	Reserved
8:6	Read / Write Ignored	Instruction cache size (Isize) 0b110 = 32 KB
5:3	Read / Write Ignored	Instruction cache associativity = 0b101 = 32-way
2	Read-as-Zero / Write Ignored	Reserved
1:0	Read / Write Ignored	Instruction cache line length = 0b10 = 8 words/line

3.5.1.2 Register 1: Control and Auxiliary Control Registers

Register 1 is made up of two registers, one that is compliant with ARM Version 5TE and referred by *opcode_2* = 0x0, and the other which is specific to the Intel XScale processor is referred by *opcode_2* = 0x1. The latter is known as the Auxiliary Control Register.

The Exception Vector Relocation bit (bit 13 of the ARM control register) allows the vectors to be mapped into high memory rather than their default location at address 0. This bit is readable and writable by software. If the MMU is enabled, the exception vectors will be accessed via the usual translation method involving the PID register (see "Register 13: Process ID" on page 84) and the TLBs. To avoid automatic application of the PID to exception vector accesses, software may relocate the exceptions to high memory.

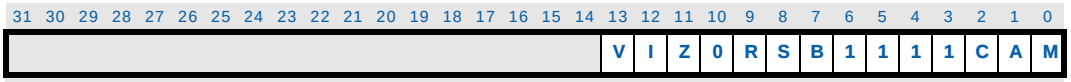
Table 12. ARM* Control Register (Sheet 1 of 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																		V	I	Z	0	R	S	B	1	1	1	1	C	A	M

reset value: writeable bits set to 0

Bits	Access	Description
31:14	Read-Unpredictable / Write-as-Zero	Reserved
13	Read / Write	Exception Vector Relocation (V). 0 = Base address of exception vectors is 0x0000,0000 1 = Base address of exception vectors is 0xFFFF,0000
12	Read / Write	Instruction Cache Enable/Disable (I) 0 = Disabled 1 = Enabled

Table 12. ARM* Control Register (Sheet 2 of 2)



reset value: writeable bits set to 0

Bits	Access	Description
11	Read / Write	Branch Target Buffer Enable (Z) 0 = Disabled 1 = Enabled
10	Read-as-Zero / Write-as-Zero	Reserved
9	Read / Write	**ROM Protection (R)** This selects the access checks performed by the memory management unit. See the *ARM* Architecture Reference Manual* for more information.
8	Read / Write	**System Protection (S)** This selects the access checks performed by the memory management unit. See the *ARM* Architecture Reference Manual* for more information.
7	Read / Write	Big/Little Endian (B) 0 = Little-endian operation 1 = Big-endian operation
6:3	Read-as-One / Write-as-One	= 0b1111
2	Read / Write	Data cache enable/disable (C) 0 = Disabled 1 = Enabled
1	Read / Write	Alignment fault enable/disable (A) 0 = Disabled 1 = Enabled
0	Read / Write	Memory management unit enable/disable (M) 0 = Disabled 1 = Enabled

The mini-data cache attribute bits, in the Auxiliary Control Register, are used to control the allocation policy for the mini-data cache and whether it will use write-back caching or write-through caching.

The configuration of the mini-data cache should be setup before any data access is made that may be cached in the mini-data cache. Once data is cached, software must ensure that the mini-data cache has been cleaned and invalidated before the mini-data cache attributes can be changed.

Table 13. Auxiliary Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																						PTEX				MD		C	B	P	K

reset value: writeable bits set to 0

Bits	Access	Description
31:6	Read-Unpredictable / Write-as-Zero	Reserved
5:4	Read / Write	Mini Data Cache Attributes (MD) All configurations of the Mini-data cache are cacheable, stores are buffered in the write buffer and stores will be coalesced in the write buffer as long as coalescing is globally enable (bit 0 of this register). 0b00 = Write back, Read allocate 0b01 = Write back, Read/Write allocate 0b10 = Write through, Read allocate 0b11 = Unpredictable
3:2	Read-Unpredictable/ Write-as-Zero	(Reserved)
1	Read/Write	Page Table Memory Attribute (P) This is a request to the core memory bus for a hardware page table walk. An ASSP may use this bit to direct the external bus controller to perform some special operation on the memory access.
0	Read / Write	Write Buffer Coalescing Disable (K) This bit globally disables the coalescing of all stores in the write buffer no matter what the value of the Cacheable and Bufferable bits are in the page table descriptors. 0 = Enabled 1 = Disabled

3.5.1.3 Register 2: Translation Table Base Register

Table 14. Translation Table Base Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Translation Table Base																															

reset value: unpredictable

Bits	Access	Description
31:14	Read / Write	**Translation Table Base** - Physical address of the base of the first-level table
13:0	Read-unpredictable / Write-as-Zero	Reserved

3.5.1.4 Register 3: Domain Access Control Register

Table 15. Domain Access Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15				D14				D13				D12				D11				D10				D9				D8			
D7				D6				D5				D4				D3				D2				D1				D0			

reset value: unpredictable

Bits	Access	Description
31:0	Read / Write	**Access permissions for all 16 domains** - The meaning of each field can be found in the *ARM* Architecture Reference Manual*.

3.5.1.5 Register 4: Reserved

Register 4 is reserved. Reading and writing this register yields unpredictable results.

3.5.1.6 Register 5: Fault Status Register

The Fault Status Register (FSR) indicates which fault has occurred, which could be either a prefetch abort or a data abort. Bit 10 extends the encoding of the status field for prefetch aborts and data aborts. The definition of the extended status field is found in "Event Architecture" on page 154. Bit 9 indicates that a debug event occurred and the exact source of the event is found in the debug control and status register (CP14, register 10). When bit 9 is set, the domain and extended status field are undefined.

Upon entry into the prefetch abort or data abort handler, hardware will update this register with the source of the exception. Software is not required to clear these fields.

Table 16. Fault Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																					X	D	0	Domain				Status			

reset value: unpredictable

Bits	Access	Description
31:11	Read-unpredictable / Write-as-Zero	Reserved
10	Read / Write	Status Field Extension (X) This bit is used to extend the encoding of the Status field, when there is a prefetch abort and when there is a data abort. The definition of this field can be found in "Event Architecture" on page 154
9	Read / Write	Debug Event (D) This flag indicates a debug event has occurred and that the cause of the debug event is found in the MOE field of the debug control register (CP14, register 10)
8	Read-as-zero / Write-as-Zero	= 0
7:4	Read / Write	**Domain** - Specifies which of the 16 domains was being accessed when a data abort occurred
3:0	Read / Write	**Status** - Type of data access being attempted

3.5.1.7 Register 6: Fault Address Register

Table 17. Fault Address Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fault Virtual Address																															
reset value: unpredictable																															

Bits	Access	Description
31:0	Read / Write	**Fault Virtual Address** - Contains the MVA of the data access that caused the memory abort

3.5.1.8 Register 7: Cache Functions

All the functions defined in existing ARM products appear here. The Intel XScale processor adds other functions as well. This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

The Drain Write Buffer function not only drains the write buffer but also drains the fill buffer.

The Intel XScale processor does not check permissions on addresses supplied for cache or TLB functions. Due to the fact only privileged software may execute these functions, full accessibility is assumed. Cache functions will not generate any of the following:

- Translation faults
- Domain faults
- Permission faults

The invalidate instruction cache line command does not invalidate the BTB. If software invalidates a line from the instruction cache and modifies the same location in external memory, it needs to invalidate the BTB also. Not invalidating the BTB in this case may cause unpredictable results.

Disabling/enabling a cache has no effect on contents of the cache: valid data stays valid, locked items remain locked. All operations defined in Table 18 work regardless of whether the cache is enabled or disabled.

Since the Clean DCache Line function reads from the data cache, it is capable of generating a parity fault. The other operations will not generate parity faults.

Table 18. Cache Functions

Function	opcode_2	CRm	Data	Instruction
Invalidate I&D cache & BTB	0b000	0b0111	Ignored	MCR p15, 0, Rd, c7, c7, 0
Invalidate I cache & BTB	0b000	0b0101	Ignored	MCR p15, 0, Rd, c7, c5, 0
Invalidate I cache line	0b001	0b0101	MVA	MCR p15, 0, Rd, c7, c5, 1
Invalidate D cache	0b000	0b0110	Ignored	MCR p15, 0, Rd, c7, c6, 0
Invalidate D cache line	0b001	0b0110	MVA	MCR p15, 0, Rd, c7, c6, 1
Clean D cache line	0b001	0b1010	MVA	MCR p15, 0, Rd, c7, c10, 1
Drain Write (& Fill) Buffer	0b100	0b1010	Ignored	MCR p15, 0, Rd, c7, c10, 4
Invalidate Branch Target Buffer	0b110	0b0101	Ignored	MCR p15, 0, Rd, c7, c5, 6
Allocate Line in the Data Cache	0b101	0b0010	MVA	MCR p15, 0, Rd, c7, c2, 5

The line-allocate command allocates a tag into the data cache specified by bits [31:5] of Rd. If a valid dirty line (with a different MVA) already exists at this location it will be evicted. The 32 bytes of data associated with the newly allocated line are not initialized and therefore will generate unpredictable results if read.

Line allocate command may be used for cleaning the entire data cache on a context switch and also when reconfiguring portions of the data cache as data RAM. In both cases, Rd is a virtual address that maps to some non-existent physical memory. When creating data RAM, software must initialize the data RAM before read accesses can occur. Specific uses of these commands can be found in Chapter 3.0, “Data Cache”.

Other items to note about the line-allocate command are:

- It forces all pending memory operations to complete.
- Bits [31:5] of Rd is used to specific the virtual address of the line to be allocated into the data cache.
- If the targeted cache line is already resident, this command has no effect.
- The command cannot be used to allocate a line in the mini Data Cache.
- The newly allocated line is not marked as “dirty” so it will never get evicted. However, if a valid store is made to that line it will be marked as “dirty” and will get written back to external memory if another line is allocated to the same cache location. This eviction will produce unpredictable results.

  To avoid this situation, the line-allocate operation should only be used if one of the following can be guaranteed:

  — The virtual address associated with this command is not one that will be generated during normal program execution. This is the case when line-allocate is used to clean/invalidate the entire cache.

  — The line-allocate operation is used only on a cache region destined to be locked. When the region is unlocked, it must be invalidated before making another data access.

3.5.1.9 Register 8: TLB Operations

Disabling/enabling the MMU has no effect on the contents of either TLB: valid entries stay valid, locked items remain locked. All operations defined in Table 19 work regardless of whether the TLB is enabled or disabled.

This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

Table 19. TLB Functions

Function	opcode_2	CRm	Data	Instruction
Invalidate I&D TLB	0b000	0b0111	Ignored	MCR p15, 0, Rd, c8, c7, 0
Invalidate I TLB	0b000	0b0101	Ignored	MCR p15, 0, Rd, c8, c5, 0
Invalidate I TLB entry	0b001	0b0101	MVA	MCR p15, 0, Rd, c8, c5, 1
Invalidate D TLB	0b000	0b0110	Ignored	MCR p15, 0, Rd, c8, c6, 0
Invalidate D TLB entry	0b001	0b0110	MVA	MCR p15, 0, Rd, c8, c6, 1

3.5.1.10 Register 9: Cache Lock Down

Register 9 is used for locking down entries into the instruction cache and data cache. (The protocol for locking down entries can be found in Chapter 3.0, “Data Cache”.)

Table 20 shows the command for locking down entries in the instruction and data cache. The entry to lock in the instruction cache is specified by the virtual address in Rd. The data cache locking mechanism follows a different procedure than the instruction cache. The data cache is placed in lock down mode such that all subsequent fills to the data cache result in that line being locked in, as controlled by Table 21.

Lock/unlock operations on a disabled cache have an undefined effect.

Read and write access is allowed to the data cache lock register bit[0]. All other accesses to register 9 should be write-only; reads, as with an MRC, have an undefined effect.

Table 20. Cache Lock-Down Functions

Function	opcode_2	CRm	Data	Instruction
Fetch and Lock I cache line	0b000	0b0001	MVA	MCR p15, 0, Rd, c9, c1, 0
Unlock Instruction cache	0b001	0b0001	Ignored	MCR p15, 0, Rd, c9, c1, 1
Read data cache lock register	0b000	0b0010	Read lock mode value	MRC p15, 0, Rd, c9, c2, 0
Write data cache lock register	0b000	0b0010	Set/Clear lock mode	MCR p15, 0, Rd, c9, c2, 0
Unlock Data Cache	0b001	0b0010	Ignored	MCR p15, 0, Rd, c9, c2, 1

Table 21. Data Cache Lock Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															L

reset value: writeable bits set to 0

Bits	Access	Description
31:1	Read-unpredictable / Write-as-Zero	Reserved
0	Read / Write	**Data Cache Lock Mode (L)** 0 = No locking occurs 1 = Any fill into the data cache while this bit is set gets locked in

3.5.1.11 Register 10: TLB Lock Down

Register 10 is used for locking down entries into the instruction TLB, and data TLB. (The protocol for locking down entries can be found in Chapter 3.0, "Memory Management Unit".) Lock/unlock operations on a TLB when the MMU is disabled have an undefined effect.

This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

Table 22 shows the command for locking down entries in the instruction TLB, and data TLB. The entry to lock is specified by the virtual address in Rd.

Table 22. TLB Lockdown Functions

Function	opcode_2	CRm	Data	Instruction
Translate and Lock I TLB entry	0b000	0b0100	MVA	MCR p15, 0, Rd, c10, c4, 0

Table 22. TLB Lockdown Functions

Function	opcode_2	CRm	Data	Instruction
Translate and Lock D TLB entry	0b000	0b1000	MVA	MCR p15, 0, Rd, c10, c8, 0
Unlock I TLB	0b001	0b0100	Ignored	MCR p15, 0, Rd, c10, c4, 1
Unlock D TLB	0b001	0b1000	Ignored	MCR p15, 0, Rd, c10, c8, 1

3.5.1.12 Register 11-12: Reserved

These registers are reserved. Reading and writing them yields unpredictable results.

3.5.1.13 Register 13: Process ID

The Intel XScale processor supports the remapping of virtual addresses through a Process ID (PID) register. This remapping occurs before the instruction cache, instruction TLB, data cache and data TLB are accessed. The PID register controls when virtual addresses are remapped and to what value.

The PID register is a 7-bit value that is ORed with bits 31:25 of the virtual address when they are zero. This action effectively remaps the address to one of 128 “slots” in the 4 Gbytes of address space. If bits 31:25 are not zero, no remapping occurs. This feature is useful for operating system management of processes that may map to the same virtual address space. In those cases, the virtually mapped caches on the Intel XScale processor would not require invalidating on a process switch.

Table 23. Accessing Process ID

Function	opcode_2	CRm	Instruction
Read Process ID Register	0b000	0b0000	MRC p15, 0, Rd, c13, c0, 0
Write Process ID Register	0b000	0b0000	MCR p15, 0, Rd, c13, c0, 0

Table 24. Process ID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Process ID																															

reset value: 0x0000,0000

Bits	Access	Description
31:25	Read / Write	**Process ID** - This field is used for remapping the virtual address when bits 31-25 of the virtual address are zero.
24:0	Read-as-Zero / Write-as-Zero	**Reserved** - Should be programmed to zero for future compatibility

3.5.1.14 The PID Register Affect On Addresses

All addresses generated and used by User Mode code are eligible for being “PIDified” as described in the previous section. Privileged code, however, must be aware of certain special cases in which address generation does not follow the usual flow.

The PID register is not used to remap the virtual address when accessing the Branch Target Buffer (BTB). Any writes to the PID register invalidate the BTB, which prevents any virtual addresses from being double mapped between two processes.

A breakpoint address (see “Register 14: Breakpoint Registers” on page 85) must be expressed as an MVA when written to the breakpoint register. This requirement means the value of the PID must be combined appropriately with the address before it is written to the breakpoint register. All virtual addresses in translation descriptors (see “Memory Management Unit” on page 44) are MVAs.

3.5.1.15 Register 14: Breakpoint Registers

The Intel XScale processor contains two instruction breakpoint address registers (IBCR0 and IBCR1), one data breakpoint address register (DBR0), one configurable data mask/address register (DBR1), and one data breakpoint control register (DBCON). The Intel XScale processor also supports a 256-entry, trace buffer that records program execution information. The registers to control the trace buffer are located in CP14.

Refer to "Software Debug" on page 88 for more information on these features of the Intel XScale processor.

Table 25. Accessing the Debug Registers

Function	opcode_2	CRm	Instruction
Access Instruction Breakpoint Control Register 0 (IBCR0)	0b000	0b1000	MRC p15, 0, Rd, c14, c8, 0 ; read MCR p15, 0, Rd, c14, c8, 0 ; write
Access Instruction Breakpoint Control Register 1(IBCR1)	0b000	0b1001	MRC p15, 0, Rd, c14, c9, 0 ; read MCR p15, 0, Rd, c14, c9, 0 ; write
Access Data Breakpoint Address Register (DBR0)	0b000	0b0000	MRC p15, 0, Rd, c14, c0, 0 ; read MCR p15, 0, Rd, c14, c0, 0 ; write
Access Data Mask/Address Register (DBR1)	0b000	0b0011	MRC p15, 0, Rd, c14, c3, 0 ; read MCR p15, 0, Rd, c14, c3, 0 ; write
Access Data Breakpoint Control Register (DBCON)	0b000	0b0100	MRC p15, 0, Rd, c14, c4, 0 ; read MCR p15, 0, Rd, c14, c4, 0 ; write

3.5.1.16 Register 15: Coprocessor Access Register

This register is selected when *opcode_2* = 0 and *CRm* = 1.

This register controls access rights to all the coprocessors in the system except for CP15 and CP14. Both CP15 and CP14 can only be accessed in privilege mode. This register is accessed with an MCR or MRC with the CRm field set to 1.

This register controls access to CP0, atypical use for this register is for an operating system to control resource sharing among applications. Initially, all applications are denied access to shared resources by clearing the appropriate coprocessor bit in the Coprocessor Access Register. An application may request the use of a shared resource (e.g., the accumulator in CP0) by issuing an access to the resource, which will result in an undefined exception. The operating system may grant access to this coprocessor by setting the appropriate bit in the Coprocessor Access Register and return to the application where the access is retried.

Sharing resources among different applications requires a state saving mechanism. Two possibilities are:

- The operating system, during a context switch, could save the state of the coprocessor if the last executing process had access rights to the coprocessor.
- The operating system, during a request for access, saves off the old coprocessor state and saves it with last process to have access to it.

Under both scenarios, the OS needs to restore state when a request for access is made. This means the OS has to maintain a list of what processes are modifying CP0 and their associated state.

Example 12. Disallowing access to CP0

```
;; The following code clears bit 0 of the CPAR.
;; This will cause the processor to fault if software
;; attempts to access CP0.

        LDR R0, =0x3FFE                 ; bit 0 is clear
        MCR P15, 0, R0, C15, C1, 0      ; move to CPAR
        CPWAIT                          ; wait for effect
```

Table 26. Coprocessor Access Register

31:16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	CP13	CP12	CP11	CP10	CP9	CP8	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0

reset value: 0x0000,0000

Bits	Access	Description
31:16	Read-unpredictable / Write-as-Zero	**Reserved** - Should be programmed to zero for future compatibility
15:14	Read-as-Zero/Write-as-Zero	**Reserved** - Should be programmed to zero for future compatibility
13:1	Read / Write	**Coprocessor Access Rights**- Each bit in this field corresponds to the access rights for each coprocessor.
0	Read / Write	**Coprocessor Access Rights**- This bit corresponds to the access rights for CP0. 0 = Access denied. Any attempt to access the corresponding coprocessor will generate an undefined exception. 1 = Access allowed. Includes read and write accesses.

3.5.2 CP14 Registers

Table 27 lists the CP14 registers implemented in the Intel XScale processor.

Table 27. CP14 Registers

Description	Access	Register# (CRn)	Register# (CRm)
Performance Monitoring	Read / Write	0,1,4,5,8	1
		0-3	2
Clock and Power Management	Read / Write	6-7	0
Software Debug	Read / Write	8-15	0

All other registers are reserved in CP14. Reading and writing them yields unpredictable results.

3.5.2.1 Performance Monitoring Registers

The performance monitoring unit contains a control register (PMNC), a clock counter (CCNT), interrupt enable register (INTEN), overflow flag register (FLAG), event selection register (EVTSEL) and four event counters (PMN0 through PMN3). The format of these registers can be found in "Performance Monitoring" on page 133, along with a description on how to use the performance monitoring facility.

Opcode_2 should be zero on all accesses.

These registers can't be accessed by **LDC** and **STC** coprocessor instructions.

Table 28. Accessing the Performance Monitoring Registers

Description	CRn Register #	CRm Register #	Instruction
(PMNC) Performance Monitor Control Register	0b0000	0b0001	Read: MRC p14, 0, Rd, c0, c1, 0 Write: MCR p14, 0, Rd, c0, c1, 0
(CCNT) Clock Counter Register	0b0001	0b0001	Read: MRC p14, 0, Rd, c1, c1, 0 Write: MCR p14, 0, Rd, c1, c1, 0
(INTEN) Interrupt Enable Register	0b0100	0b0001	Read: MRC p14, 0, Rd, c4, c1, 0 Write: MCR p14, 0, Rd, c4, c1, 0
(FLAG) Overflow Flag Register	0b0101	0b0001	Read: MRC p14, 0, Rd, c5, c1, 0 Write: MCR p14, 0, Rd, c5, c1, 0
(EVTSEL) Event Selection Register	0b1000	0b0001	Read: MRC p14, 0, Rd, c8, c1, 0 Write: MCR p14, 0, Rd, c8, c1, 0
(PMN0) Performance Count Register 0	0b0000	0b0010	Read: MRC p14, 0, Rd, c0, c2, 0 Write: MCR p14, 0, Rd, c0, c2, 0
(PMN1) Performance Count Register 1	0b0001	0b0010	Read: MRC p14, 0, Rd, c1, c2, 0 Write: MCR p14, 0, Rd, c1, c2, 0
(PMN2) Performance Count Register 2	0b0010	0b0010	Read: MRC p14, 0, Rd, c2, c2, 0 Write: MCR p14, 0, Rd, c2, c2, 0
(PMN3) Performance Count Register 3	0b0011	0b0010	Read: MRC p14, 0, Rd, c3, c2, 0 Write: MCR p14, 0, Rd, c3, c2, 0

3.5.2.2 Clock and Power Management Registers

These registers contain functions for managing the core clock and power.

For the IXP42X product line and IXC1100 control plane processors, these registers are not implemented and reserved for future use.

Table 29. PWRMODE Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																														M	

reset value: writeable bits set to 0

Bits	Access	Description
31:0	Read-unpredictable / Write-as-Zero	Reserved
1:0	Read / Write	**Mode (M)** 0 = ACTIVE Never change from 00b

The Intel XScale processor clock frequency cannot be changed by software on the IXP42X product line and IXC1100 control plane processors.

Table 30. Clock and Power Management

Function	Data	Instruction
Read CCLKCFG	ignored	MRC p14, 0, Rd, c6, c0, 0
Write CCLKCFG	CCLKCFG value	MCR p14, 0, Rd, c6, c0, 0

Table 31. CCLKCFG Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															CCLKCFG

reset value: unpredictable

Bits	Access	Description
31:0	Read-unpredictable / Write-as-Zero always	Reserved (write as zero)

3.5.2.3 Software Debug Registers

Software debug is supported by address breakpoint registers (Coprocessor 15, register 14), serial communication over the JTAG interface and a trace buffer. Registers 8 and 9 are used for the serial interface and registers 10 through 13 support a 256 entry trace buffer. Register 14 and 15 are the debug link register and debug SPSR (saved program status register). These registers are explained in more detail in "Software Debug" on page 88.

Opcode_2 and CRm should be zero.

Table 32. Accessing the Debug Registers

Function	CRn (Register #)	Instruction
Access Transmit Debug Register (TX)	0b1000	MRC p14, 0, Rd, c8, c0, 0 MCR p14, 0, Rd, c8, c0, 0
Access Receive Debug Register (RX)	0b1001	MCR p14, 0, Rd, c9, c0, 0 MRC p14, 0, Rd, c9, c0, 0
Access Debug Control and Status Register (DBGCSR)	0b1010	MCR p14, 0, Rd, c10, c0, 0 MRC p14, 0, Rd, c10, c0, 0
Access Trace Buffer Register (TBREG)	0b1011	MCR p14, 0, Rd, c11, c0, 0 MRC p14, 0, Rd, c11, c0, 0
Access Checkpoint 0 Register (CHKPT0)	0b1100	MCR p14, 0, Rd, c12, c0, 0 MRC p14, 0, Rd, c12, c0, 0
Access Checkpoint 1 Register (CHKPT1)	0b1101	MCR p14, 0, Rd, c13, c0, 0 MRC p14, 0, Rd, c13, c0, 0
Access Transmit and Receive Debug Control Register	0b1110	MCR p14, 0, Rd, c14, c0, 0 MRC p14, 0, Rd, c14, c0, 0

3.6 Software Debug

This section describes the software debug and related features implemented in the IXP42X product line and IXC1100 control plane processors, namely:

- Debug modes, registers and exceptions
- A serial debug communication link via the JTAG interface
- A trace buffer
- A mechanism to load the instruction cache through JTAG

- Debug Handler SW requirements and suggestions

3.6.1 Definitions

Debug handler: Debug handler is event handler that runs on IXP42X product line and IXC1100 control plane processors, when a debug event occurs.

Debugger: The debugger is software that runs on a host system outside of IXP42X product line and IXC1100 control plane processors.

3.6.2 Debug Registers

CP15 Registers
CRn = 14; CRm = 8: instruction breakpoint register 0 (IBCR0)
CRn = 14; CRm = 9: instruction breakpoint register 1 (IBCR1)
CRn = 14; CRm = 0: data breakpoint register 0 (DBR0)
CRn = 14; CRm = 3: data breakpoint register 1 (DBR1)
CRn = 14; CRm = 4: data breakpoint control register (DBCON)

CP15 registers are accessible using MRC and MCR. CRn and CRm specify the register to access. The opcode_1 and opcode_2 fields are not used and should be set to 0.

CP14 Registers
CRn = 8; CRm = 0: TX Register (TX)
CRn = 9; CRm = 0: RX Register (RX)
CRn = 10; CRm = 0: Debug Control and Status Register (DCSR)
CRn = 11; CRm = 0: Trace Buffer Register (TBREG)
CRn = 12; CRm = 0: Checkpoint Register 0 (CHKPT0)
CRn = 13; CRm = 0: Checkpoint Register 1 (CHKPT1)
CRn = 14; CRm = 0: TXRX Control Register (TXRXCTRL)

CP14 registers are accessible using MRC, MCR, LDC and STC (CDP to any CP14 registers will cause an undefined instruction trap). The CRn field specifies the number of the register to access. The CRm, opcode_1, and opcode_2 fields are not used and should be set to 0.

Software access to all debug registers must be done in privileged mode. User mode access will generate an undefined instruction exception. Specifying registers which do not exist has unpredictable results.

The TX and RX registers, certain bits in the TXRXCTRL register, and certain bits in the DCSR can be accessed by a debugger through the JTAG interface.

3.6.3 Debug Modes

The IXP42X product line and IXC1100 control plane processors' debug unit, when used with a debugger application, allows software running on an IXP42X product line and IXC1100 control plane processors' target to be debugged. The debug unit allows the debugger to stop program execution and re-direct execution to a debug handling routine. Once program execution has stopped, the debugger can examine or modify processor state, co-processor state, or memory. The debugger can then restart execution of the application.

On IXP42X product line and IXC1100 control plane processors, one of two debug modes can be entered:

- Halt mode
- Monitor mode

3.6.3.1 Halt Mode

When the debug unit is configured for halt mode, the reset vector is overloaded to serve as the debug vector. A new processor mode, DEBUG mode (CPSR[4:0] = 0x15), is added to allow debug exceptions to be handled similarly to other types of ARM exceptions.

When a debug exception occurs, the processor switches to debug mode and redirects execution to a debug handler, via the reset vector. After the debug handler begins execution, the debugger can communicate with the debug handler to examine or alter processor state or memory through the JTAG interface.

The debug handler can be downloaded and locked directly into the instruction cache through JTAG so external memory is not required to contain debug handler code.

3.6.3.2 Monitor Mode

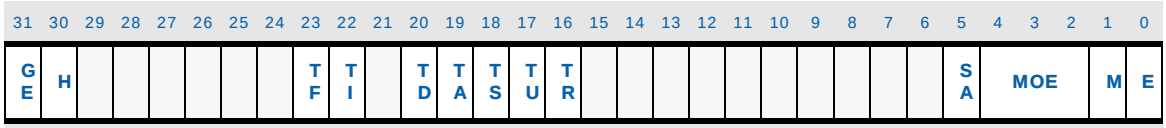
In monitor mode, debug exceptions are handled like ARM prefetch aborts or ARM data aborts, depending on the cause of the exception.

When a debug exception occurs, the processor switches to abort mode and branches to a debug handler using the pre-fetch abort vector or data abort vector. The debugger then communicates with the debug handler to access processor state or memory contents.

3.6.4 Debug Control and Status Register (DCSR)

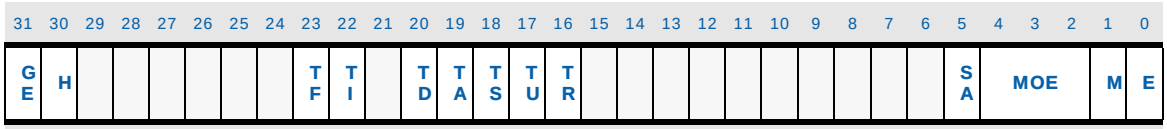
The DCSR register is the main control register for the debug unit. Table 33 shows the format of the register. The DCSR register can be accessed in privileged modes by software running on the processor or by a debugger through the JTAG interface. Refer to “SELDCSR JTAG Register” on page 103 for details about accessing DCSR through JTAG.

Table 33. Debug Control and Status Register (DCSR) (Sheet 1 of 2)



Bits	Access	Description	Reset Value	TRST Value
31	SW Read / Write JTAG Read-Only	Global Enable (GE) 0: disables all debug functionality 1: enables all debug functionality	0	unchanged
30	SW Read Only JTAG Read / Write	Halt Mode (H) 0: Monitor Mode 1: Halt Mode	unchanged	0
29:24	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
23	SW Read Only JTAG Read / Write	Trap FIQ (TF)	unchanged	0
22	SW Read Only JTAG Read / Write	Trap IRQ (TI)	unchanged	0

Table 33. Debug Control and Status Register (DCSR) (Sheet 2 of 2)



Bits	Access	Description	Reset Value	TRST Value
21	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
20	SW Read Only JTAG Read / Write	Trap Data Abort (TD)	unchanged	0
19	SW Read Only JTAG Read / Write	Trap Prefetch Abort (TA)	unchanged	0
18	SW Read Only JTAG Read / Write	Trap Software Interrupt (TS)	unchanged	0
17	SW Read Only JTAG Read / Write	Trap Undefined Instruction (TU)	unchanged	0
16	SW Read Only JTAG Read / Write	Trap Reset (TR)	unchanged	0
15:6	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
5	SW Read / Write JTAG Read-Only	Sticky Abort (SA)	0	unchanged
4:2	SW Read / Write JTAG Read-Only	Method Of Entry (MOE) 000: Processor Reset 001: Instruction Breakpoint Hit 010: Data Breakpoint Hit 011: BKPT Instruction Executed 100: External Debug Event Asserted 101: Vector Trap Occurred 110: Trace Buffer Full Break 111: Reserved	0b000	unchanged
1	SW Read / Write JTAG Read-Only	Trace Buffer Mode (M) 0: Wrap around mode 1: fill-once mode	0	unchanged
0	SW Read / Write JTAG Read-Only	Trace Buffer Enable (E) 0: Disabled 1: Enabled	0	unchanged

3.6.4.1 Global Enable Bit (GE)

The Global Enable bit disables and enables all debug functionality (except the reset vector trap). Following a processor reset, this bit is clear so all debug functionality is disabled. When debug functionality is disabled, the BKPT instruction becomes a loop and external debug breaks, hardware breakpoints, and non-reset vector traps are ignored.

3.6.4.2 Halt Mode Bit (H)

The Halt Mode bit configures the debug unit for either halt mode or monitor mode.

3.6.4.3 Vector Trap Bits (TF,TI,TD,TA,TS,TU,TR)

The Vector Trap bits allow instruction breakpoints to be set on exception vectors without using up any of the breakpoint registers. When a bit is set, it acts as if an instruction breakpoint was set up on the corresponding exception vector. A debug exception is generated before the instruction in the exception vector executes.

Software running on IXP42X product line and IXC1100 control plane processors must set the Global Enable bit and the debugger must set the Halt Mode bit and the appropriate vector trap bit through JTAG to set up a non-reset vector trap.

To set up a reset vector trap, the debugger sets the Halt Mode bit and reset vector trap bit through JTAG. The Global Enable bit does not effect the reset vector trap. A reset vector trap can be set up before or during a processor reset. When processor reset is de-asserted, a debug exception occurs before the instruction in the reset vector executes.

3.6.4.4 Sticky Abort Bit (SA)

The Sticky Abort bit is only valid in Halt mode. It indicates a data abort occurred within the Special Debug State (see "Halt Mode" on page 93). Since Special Debug State disables all exceptions, a data abort exception does not occur. However, the processor sets the Sticky Abort bit to indicate a data abort was detected. The debugger can use this bit to determine if a data abort was detected during the Special Debug State. The sticky abort bit must be cleared by the debug handler before exiting the debug handler.

3.6.4.5 Method of Entry Bits (MOE)

The Method of Entry bits specify the cause of the most recent debug exception. When multiple exceptions occur in parallel, the processor places the highest priority exception (based on the priorities in Table 34) in the MOE field.

3.6.4.6 Trace Buffer Mode Bit (M)

The Trace Buffer Mode bit selects one of two trace buffer modes:

- Wrap-around mode — Trace buffer fills up and wraps around until a debug exception occurs.
- Fill-once mode — The trace buffer automatically generates a debug exception (trace buffer full break) when it becomes full.

3.6.4.7 Trace Buffer Enable Bit (E)

The Trace Buffer Enable bit enables and disables the trace buffer. Both DCSR.e and DCSR.ge must be set to enable the trace buffer. The processor automatically clears this bit to disable the trace buffer when a debug exception occurs. For more details on the trace buffer refer to "Trace Buffer" on page 109.

3.6.5 Debug Exceptions

A debug exception causes the processor to re-direct execution to a debug event handling routine. IXP42X product line and IXC1100 control plane processors' debug architecture defines the following debug exceptions:

- Instruction breakpoint
- Data breakpoint
- Software breakpoint
- External debug break

- Exception vector trap
- Trace-buffer full break

When a debug exception occurs, the processor's actions depend on whether the debug unit is configured for Halt mode or Monitor mode.

Table 34 shows the priority of debug exceptions relative to other processor exceptions.

Table 34. Event Priority

Event	Priority
Reset	1
Vector Trap	2
data abort (precise)	3
data bkpt	4
data abort (imprecise)	5
external debug break, trace-buffer full	6
FIQ	7
IRQ	8
instruction breakpoint	9
pre-fetch abort	10
undef, SWI, software Bkpt	11

3.6.5.1 Halt Mode

The debugger turns on Halt mode through the JTAG interface by scanning in a value that sets the bit in DCSR. The debugger turns off Halt mode through JTAG, either by scanning in a new DCSR value or by a TRST. Processor reset does not effect the value of the Halt mode bit.

When halt mode is active, the processor uses the reset vector as the debug vector. The debug handler and exception vectors can be downloaded directly into the instruction cache, to intercept the default vectors and reset handler, or they can be resident in external memory. Downloading into the instruction cache allows a system with memory problems, or no external memory, to be debugged. Refer top "Downloading Code in ICache" on page 116 for details about downloading code into the instruction cache.

During Halt mode, software running on IXP42X product line and IXC1100 control plane processors cannot access DCSR, or any of hardware breakpoint registers, unless the processor is in Special Debug State (SDS), described below.

When a debug exception occurs during Halt mode, the processor takes the following actions:

- Disables the trace buffer
- Sets DCSR.moe encoding
- Processor enters a Special Debug State (SDS)
- For data breakpoints, trace buffer full break, and external debug break:
  R14_dbg = PC of the next instruction to execute + 4
  for instruction breakpoints and software breakpoints and vector traps:
  R14_dbg = PC of the aborted instruction + 4
- SPSR_dbg = CPSR
- CPSR[4:0] = 0b10101 (DEBUG mode)

- CPSR[5] = 0
- CPSR[6] = 1
- CPSR[7] = 1
- PC = 0x0

Note: When the vector table is relocated (CP15 Control Register[13] = 1), the debug vector is relocated to 0xffff0000.

Following a debug exception, the processor switches to debug mode and enters SDS, which allows the following special functionality:

- All events are disabled. SWI or undefined instructions have unpredictable results. The processor ignores pre-fetch aborts, FIQ and IRQ (SDS disables FIQ and IRQ regardless of the enable values in the CPSR). The processor reports data aborts detected during SDS by setting the Sticky Abort bit in the DCSR, but does not generate an exception (processor also sets up FSR and FAR as it normally would for a data abort).
- Normally, during halt mode, software cannot write the hardware breakpoint registers or the DCSR. However, during the SDS, software has write access to the breakpoint registers (see "HW Breakpoint Resources" on page 95) and the DCSR (see Table 33, "Debug Control and Status Register (DCSR)" on page 90).
- The IMMU is disabled. In halt mode, since the debug handler would typically be downloaded directly into the IC, it would not be appropriate to do TLB accesses or translation walks, since there may not be any external memory or if there is, the translation table or TLB may not contain a valid mapping for the debug handler code. To avoid these problems, the processor internally disables the IMMU during SDS.
- The PID is disabled for instruction fetches. This prevents fetches of the debug handler code from being remapped to a different address than where the code was downloaded.

The SDS remains in effect regardless of the processor mode. This allows the debug handler to switch to other modes, maintaining SDS functionality. Entering user mode may cause unpredictable behavior. The processor exits SDS following a CPSR restore operation.

When exiting, the debug handler should use:

```
subs pc, lr, #4
```

This restores CPSR, turns off all of SDS functionality, and branches to the target instruction.

3.6.5.2 Monitor Mode

In monitor mode, the processor handles debug exceptions like normal ARM exceptions. If debug functionality is enabled (DCSR[31] = 1) and the processor is in Monitor mode, debug exceptions cause either a data abort or a pre-fetch abort.

The following debug exceptions cause data aborts:

- Data breakpoint
- External debug break
- Trace-buffer full break

The following debug exceptions cause pre-fetch aborts:

- Instruction breakpoint
- BKPT instruction

The processor ignores vector traps during monitor mode.

When an exception occurs in monitor mode, the processor takes the following actions:

- Disables the trace buffer
- Sets DCSR.moe encoding
- Sets FSR[9]
- R14_abt = PC of the next instruction to execute + 4 (for Data Aborts)
  R14_abt = PC of the faulting instruction + 4 (for Prefetch Aborts)
- SPSR_abt = CPSR
- CPSR[4:0] = 0b10111 (ABORT mode)
- CPSR[5] = 0
- CPSR[6] = unchanged
- CPSR[7] = 1
- PC = 0xc (for Prefetch Aborts),
  PC = 0x10 (for Data Aborts)

During abort mode, external debug breaks and trace buffer full breaks are internally pended. When the processor exits abort mode, either through a CPSR restore or a write directly to the CPSR, the pended debug breaks will immediately generate a debug exception. Any pending debug breaks are cleared out when any type of debug exception occurs.

When exiting, the debug handler should do a CPSR restore operation that branches to the next instruction to be executed in the program under debug.

3.6.6 HW Breakpoint Resources

IXP42X product line and IXC1100 control plane processors' debug architecture defines two instruction and two data breakpoint registers, denoted IBCR0, IBCR1, DBR0, and DBR1.

The instruction and data address breakpoint registers are 32-bit registers. The instruction breakpoint causes a break before execution of the target instruction. The data breakpoint causes a break after the memory access has been issued.

In this section Modified Virtual Address (MVA) refers to the virtual address ORed with the PID. Refer to "Register 13: Process ID" on page 84 for more details on the PID. The processor does not OR the PID with the specified breakpoint address prior to doing address comparison. This must be done by the programmer and written to the breakpoint register as the MVA. This applies to data and instruction breakpoints.

3.6.6.1 Instruction Breakpoints

The Debug architecture defines two instruction breakpoint registers (IBCR0 and IBCR1). The format of these registers is shown in Table 35., Instruction Breakpoint Address and Control Register (IBCRx). In ARM mode, the upper 30 bits contain a word aligned MVA to break on. In Thumb mode, the upper 31 bits contain a half-word aligned MVA to break on. In both modes, bit 0 enables and disables that instruction breakpoint register. Enabling instruction breakpoints while debug is globally disabled (DCSR.GE=0) may result in unpredictable behavior.

Table 35. Instruction Breakpoint Address and Control Register (IBCRx)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IBCRx																															E

reset value: unpredictable address, disabled

Bits	Access	Description
31:1	Read / Write	Instruction Breakpoint MVA in ARM mode, IBCRx[1] is ignored
0	Read / Write	**IBCRx Enable (E)** - 0 = Breakpoint disabled 1 = Breakpoint enabled

An instruction breakpoint will generate a debug exception before the instruction at the address specified in the ICBR executes. When an instruction breakpoint occurs, the processor sets the DBCR.moe bits to 0b001.

Software must disable the breakpoint before exiting the handler. This allows the break-pointed instruction to execute after the exception is handled.

Single step execution is accomplished using the instruction breakpoint registers and must be completely handled in software (either on the host or by the debug handler).

3.6.6.2 Data Breakpoints

IXP42X product line and IXC1100 control plane processors' debug architecture defines two data breakpoint registers (DBR0, DBR1). The format of the registers is shown in Table 36.

Table 36. Data Breakpoint Register (DBRx)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBRx																															

reset value: unpredictable

Bits	Access	Description
31:0	Read / Write	**DBR0**: Data Breakpoint MVA **DBR1**: Data Address Mask OR Data Breakpoint MVA

DBR0 is a dedicated data address breakpoint register. DBR1 can be programmed for one of two operations:

- Data address mask
- Second data address breakpoint

The DBCON register controls the functionality of DBR1, as well as the enables for both DBRs. DBCON also controls what type of memory access to break on.

Table 37. Data Breakpoint Controls Register (DBCON)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																							M					E1		E0	

reset value: 0x00000000

Bits	Access	Description
31:9	Read-as-Zero / Write-ignored	Reserved
8	Read / Write	**DBR1 Mode (M)** - 0: DBR1 = Data Address Breakpoint 1: DBR1 = Data Address Mask
7:4	Read-as-Zero / Write-ignored	Reserved
3:2	Read / Write	**DBR1 Enable (E1)** - When DBR1 = Data Address Breakpoint 0b00: DBR1 disabled 0b01: DBR1 enabled, Store only 0b10: DBR1 enabled, Any data access, load or store 0b11: DBR1 enabled, Load only When DBR1 = Data Address Mask this field has no effect
1:0	Read / Write	**DBR0 Enable (E0)** - 0b00: DBR0 disabled 0b01: DBR0 enabled, Store only 0b10: DBR0 enabled, Any data access, load or store 0b11: DBR0 enabled, Load only

When DBR1 is programmed as a data address mask, it is used in conjunction with the address in DBR0. The bits set in DBR1 are ignored by the processor when comparing the address of a memory access with the address in DBR0. Using DBR1 as a data address mask allows a range of addresses to generate a data breakpoint. When DBR1 is selected as a data address mask, it is unaffected by the E1 field of DBCON. The mask is used only when DBR0 is enabled.

When DBR1 is programmed as a second data address breakpoint, it functions independently of DBR0. In this case, the DBCON.E1 controls DBR1.

A data breakpoint is triggered if the memory access matches the access type and the address of any byte within the memory access matches the address in DBRx. For example, LDR triggers a breakpoint if DBCON.E0 is 0b10 or 0b11, and the address of any of the 4 bytes accessed by the load matches the address in DBR0.

The processor does not trigger data breakpoints for the PLD instruction or any CP15, register 7, 8, 9, or 10 functions. Any other type of memory access can trigger a data breakpoint. For data breakpoint purposes the SWP and SWPB instructions are treated as stores - they will not cause a data breakpoint if the breakpoint is set up to break on loads only and an address match occurs.

On unaligned memory accesses, breakpoint address comparison is done on a word-aligned address (aligned down to word boundary).

When a memory access triggers a data breakpoint, the breakpoint is reported after the access is issued. The memory access will not be aborted by the processor. The actual timing of when the access completes with respect to the start of the debug handler depends on the memory configuration.

On a data breakpoint, the processor generates a debug exception and re-directs execution to the debug handler before the next instruction executes. The processor reports the data breakpoint by setting the DCSR.MOE to 0b010. The link register of a data breakpoint is always PC (of the next instruction to execute) + 4, regardless of whether the processor is configured for monitor mode or halt mode.

3.6.7 Software Breakpoints

Mnemonics: BKPT (See *ARM* Architecture Reference Manual*, ARMv5T)

Operation: If DCSR[31] = 0, BKPT is a nop;
If DCSR[31] =1, BKPT causes a debug exception

The processor handles the software breakpoint as described in "Debug Exceptions" on page 92.

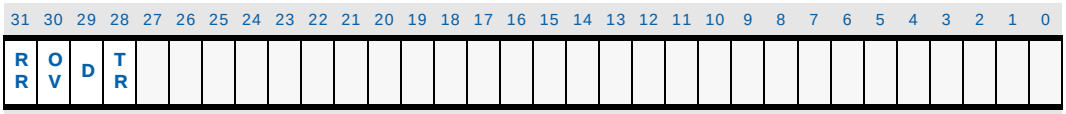
3.6.8 Transmit/Receive Control Register (TXRXCTRL)

Communications between the debug handler and debugger are controlled through handshaking bits that ensures the debugger and debug handler make synchronized accesses to TX and RX. The debugger side of the handshaking is accessed through the DBGTX ("DBGTX JTAG Register" on page 105) and DBGRX ("DBGRX JTAG Register" on page 106) JTAG Data Registers, depending on the direction of the data transfer.The debug handler uses separate handshaking bits in TXRXCTRL register for accessing TX and RX.

The TXRXCTRL register also contains two other bits that support high-speed download. One bit indicates an overflow condition that occurs when the debugger attempts to write the RX register before the debug handler has read the previous data written to RX. The other bit is used by the debug handler as a branch flag during high-speed download.

All of the bits in the TXRXCTRL register are placed such that they can be read directly into the CC flags in the CPSR with an MRC (with Rd = PC). The subsequent instruction can then conditionally execute based on the updated CC value

Table 38. TX RX Control Register (TXRXCTRL)



reset value: 0x00000000

Bits	Access	Description
31	SW Read-only / Write-ignored JTAG Write-only	RR RX Register Ready
30	SW Read / Write	OV RX overflow sticky flag
29	SW Read-only/ Write-ignored JTAG Write-only	D High-speed download flag
28	SW Read-only/ Write-ignored JTAG Write-only	TR TX Register Ready
27:0	Read-as-Zero / Write-ignored	Reserved

3.6.8.1 RX Register Ready Bit (RR)

The debugger and debug handler use the RR bit to synchronize accesses to RX. Normally, the debugger and debug handler use a handshaking scheme that requires both sides to poll the RR bit. To support higher download performance for large amounts of data, a high-speed download handshaking scheme can be used in which only the debug handler polls the RR bit before accessing the RX register, while the debugger continuously downloads data.

Table 39 shows the normal handshaking used to access the RX register.

Table 39. Normal RX Handshaking

Debugger Actions
• Debugger wants to send data to debug handler. • Before writing new data to the RX register, the debugger polls RR through JTAG until the bit is cleared. • After the debugger reads a '0' from the RR bit, it scans data into JTAG to write to the RX register and sets the valid bit. The write to the RX register automatically sets the RR bit.
Debug Handler Actions
• Debug handler is expecting data from the debugger. • The debug handler polls the RR bit until it is set, indicating data in the RX register is valid. • Once the RR bit is set, the debug handler reads the new data from the RX register. The read operation automatically clears the RR bit.

When data is being downloaded by the debugger, part of the normal handshaking can be bypassed to allow the download rate to be increased. Table 40 shows the handshaking used when the debugger is doing a high-speed download. Note that before the high-speed download can start, both the debugger and debug handler must be synchronized, such that the debug handler is executing a routine that supports the high-speed download.

Although it is similar to the normal handshaking, the debugger polling of RR is bypassed with the assumption that the debug handler can read the previous data from RX before the debugger can scan in the new data.

Table 40. High-Speed Download Handshaking States

Debugger Actions
• Debugger wants to transfer code into IXP42X product line and IXC1100 control plane processors' system memory. • Prior to starting download, the debugger must polls RR bit until it is clear. Once the RR bit is clear, indicating the debug handler is ready, the debugger starts the download. • The debugger scans data into JTAG to write to the RX register with the download bit and the valid bit set. Following the write to RX, the RR bit and D bit are automatically set in TXRXCTRL. • Without polling of RR to see whether the debug handler has read the data just scanned in, the debugger continues scanning in new data into JTAG for RX, with the download bit and the valid bit set. • An overflow condition occurs if the debug handler does not read the previous data before the debugger completes scanning in the new data, (See "Overflow Flag (OV)" on page 100 for more details on the overflow condition). • After completing the download, the debugger clears the D bit allowing the debug handler to exit the download loop.
Debug Handler Actions
• Debug is handler in a routine waiting to write data out to memory. The routine loops based on the D bit in TXRXCTRL. • The debug handler polls the RR bit until it is set. It then reads the Rx register, and writes it out to memory. The handler loops, repeating these operations until the debugger clears the D bit.

3.6.8.2 Overflow Flag (OV)

The Overflow flag is a sticky flag that is set when the debugger writes to the RX register while the RR bit is set.

The flag is used during high-speed download to indicate that some data was lost. The assumption during high-speed download is that the time it takes for the debugger to shift in the next data word is greater than the time necessary for the debug handler to process the previous data word. So, before the debugger shifts in the next data word, the handler will be polling for that data.

However, if the handler incurs stalls that are long enough such that the handler is still processing the previous data when the debugger completes shifting in the next data word, an overflow condition occurs and the OV bit is set.

Once set, the overflow flag will remain set, until cleared by a write to TXRXCTRL with an MCR. After the debugger completes the download, it can examine the OV bit to determine if an overflow occurred. The debug handler software is responsible for saving the address of the last valid store before the overflow occurred.

3.6.8.3 Download Flag (D)

The value of the download flag is set by the debugger through JTAG. This flag is used during high-speed download to replace a loop counter.

The download flag becomes especially useful when an overflow occurs. If a loop counter is used, and an overflow occurs, the debug handler cannot determine how many data words overflowed. Therefore the debug handler counter may get out of sync with the debugger — the debugger may finish downloading the data, but the debug handler counter may indicate there is more data to be downloaded - this may result in unpredictable behavior of the debug handler.

Using the download flag, the debug handler loops until the debugger clears the flag. Therefore, when doing a high-speed download, for each data word downloaded, the debugger should set the D bit.

3.6.8.4 TX Register Ready Bit (TR)

The debugger and debug handler use the TR bit to synchronize accesses to the TX register. The debugger and debug handler must poll the TR bit before accessing the TX register. Table 41 shows the handshaking used to access the TX register.

Table 41. TX Handshaking

Debugger Actions
• Debugger is expecting data from the debug handler. • Before reading data from the TX register, the debugger polls the TR bit through JTAG until the bit is set. NOTE: while polling TR, the debugger must scan out the TR bit and the TX register data. • Reading a '1' from the TR bit, indicates that the TX data scanned out is valid • The action of scanning out data when the TR bit is set, automatically clears TR.
Debug Handler Actions
• Debug handler wants to send data to the debugger (in response to a previous request). • The debug handler polls the TR bit to determine when the TX register is empty (any previous data has been read out by the debugger). The handler polls the TR bit until it is clear. • Once the TR bit is clear, the debug handler writes new data to the TX register. The write operation automatically sets the TR bit.

3.6.8.5 Conditional Execution Using TXRXCTRL

All of the bits in TXRXCTRL are placed such that they can be read directly into the CC flags using an MCR instruction. To simplify the debug handler, the TXRXCTRL register should be read using the following instruction:

```
mrc p14, 0, r15, C14, C0, 0
```

This instruction will directly update the condition codes in the CPSR. The debug handler can then conditionally execute based on each C bit. Table 42 shows the mnemonic extension to conditionally execute based on whether the TXRXCTRL bit is set or clear.

Table 42. TXRXCTRL Mnemonic Extensions

TXRXCTRL bit	mnemonic extension to execute if bit set	mnemonic extension to execute if bit clear
31 (to N flag)	MI	PL
30 (to Z flag)	EQ	NE
29 (to C flag)	CS	CC
28 (to V flag)	VS	VC

The following example is a code sequence in which the debug handler polls the TXRXCTRL handshaking bit to determine when the debugger has completed its write to RX and the data is ready for the debug handler to read.

```
loop:   mcr     p14, 0, r15, c14, c0, 0# read the handshaking bit in TXRXCTRL
        mcrmi   p14, 0, r0, c9, c0, 0  # if RX is valid, read it
        bpl     loop                   # if RX is not valid, loop
```

3.6.9 Transmit Register (TX)

The TX register is the debug handler transmit buffer. The debug handler sends data to the debugger through this register.

Table 43. TX Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TX																															
reset value: unpredictable																															

Bits	Access	Description
31:0	SW Read / Write JTAG Read-only	Debug handler writes data to send to debugger

Since the TX register is accessed by the debug handler (using MCR/MRC) and the debugger (through JTAG), handshaking is required to prevent the debug handler from writing new data before the debugger reads the previous data.

The TX register handshaking is described in Table 41, “TX Handshaking” on page 100.

3.6.10 Receive Register (RX)

The RX register is the receive buffer used by the debug handler to get data sent by the debugger through the JTAG interface.

Table 44. RX Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX																															
reset value: unpredictable																															

Bits	Access	Description
31:0	SW Read-only JTAG Write-only	Software reads to receives data/commands from debugger

Since the RX register is accessed by the debug handler (using MRC) and the debugger (through JTAG), handshaking is required to prevent the debugger from writing new data to the register before the debug handler reads the previous data out. The handshaking is described in "RX Register Ready Bit (RR)" on page 99.

3.6.11 Debug JTAG Access

There are four JTAG instructions used by the debugger during software debug: LDIC, SELDCSR, DBGTX and DBGRX. LDIC is described in "Downloading Code in ICache" on page 116. The other three JTAG instructions are described in this section.

SELDCSR, DBGTX and DBGRX use a common 36-bit shift register (DBG_SR). New data is shifted in and captured data out through the DBG_SR. In the UPDATE_DR state, the new data shifted into the appropriate data register.

3.6.11.1 SELDCSR JTAG Command

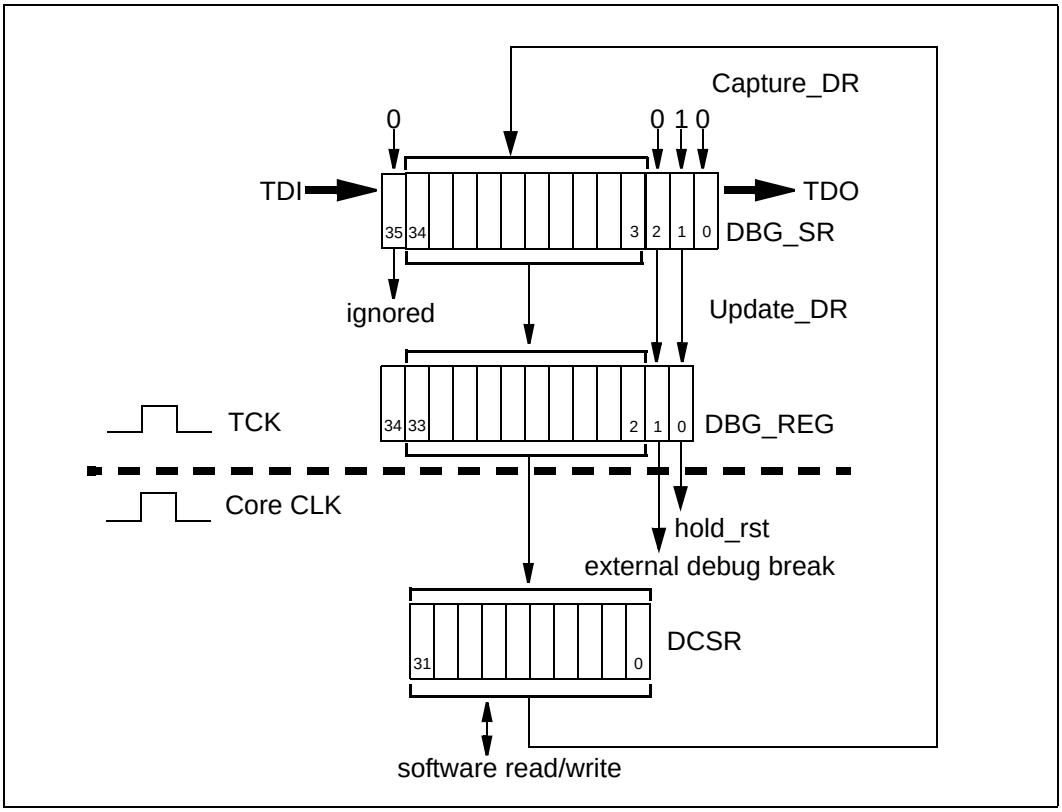
The 'SELDCSR' JTAG instruction selects the DCSR JTAG data register. The JTAG op code is '01001'. When the SELDCSR JTAG instruction is in the JTAG instruction register, the debugger can directly access the Debug Control and Status Register (DCSR). The debugger can only modify certain bits through JTAG, but can read the entire register.

The SELDCSR instruction also allows the debugger to generate an external debug break.

3.6.11.2 SELDCSR JTAG Register

Placing the “SELDCSR” JTAG instruction in the JTAG IR, selects the DCSR JTAG Data register (Figure 15), allowing the debugger to access the DCSR, generate an external debug break, set the hold_rst signal, which is used when loading code into the instruction cache during reset.

Figure 15. SELDCSR Hardware



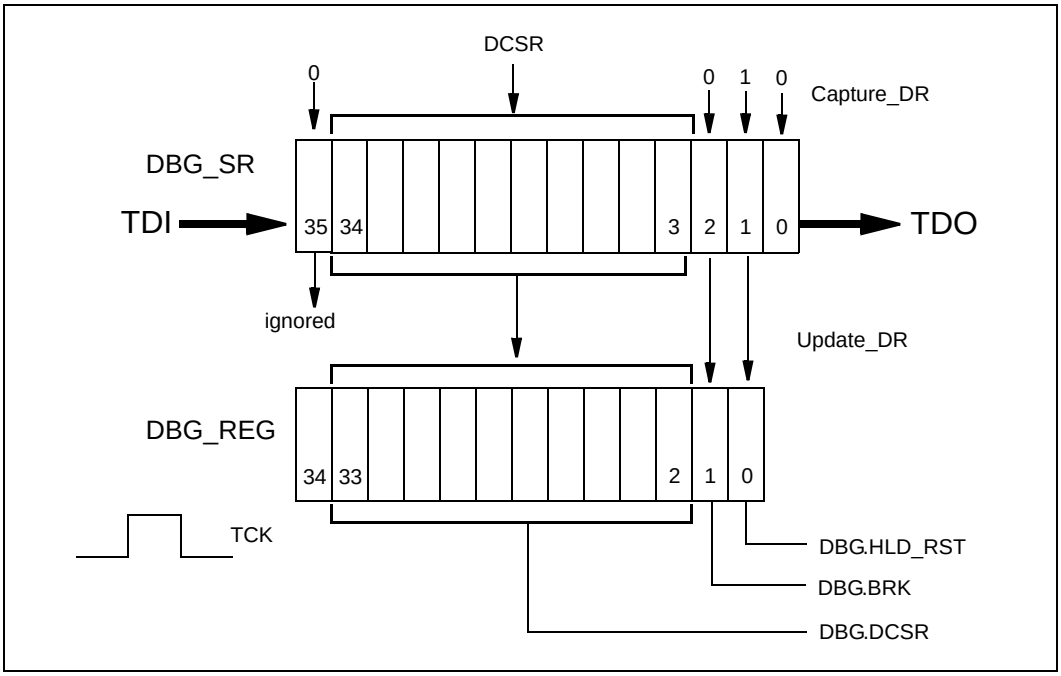
A Capture_DR loads the current DCSR value into DBG_SR[34:3]. The other bits in DBG_SR are loaded as shown in Figure 15.

A new DCSR value can be scanned into DBG_SR, and the previous value out, during the Shift_DR state. When scanning in a new DCSR value into the DBG_SR, care must be taken to also set up DBG_SR[2:1] to prevent undesirable behavior.

Update_DR parallel loads the new DCSR value into DBG_REG[33:2]. This value is then loaded into the actual DCSR register. All bits defined as JTAG writable in Table 33, “Debug Control and Status Register (DCSR)” on page 90 are updated.

An external host and the debug handler running on IXP42X product line and IXC1100 control plane processors must synchronize access the DCSR. If one side writes the DCSR at the same side the other side reads the DCSR, the results are unpredictable.

Figure 16. SELDCSR Data Register



3.6.11.2.1 DBG.HLD_RST

The debugger uses DBG.HLD_RST when loading code into the instruction cache during a processor reset. Details about loading code into the instruction cache are in "Downloading Code in ICache" on page 116.

The debugger must set DBG.HLD_RST before or during assertion of the reset pin. Once DBG.HLD_RST is set, the reset pin can be de-asserted, and the processor will internally remain in reset. The debugger can then load debug handler code into the instruction cache before the processor begins executing any code.

Once the code download is complete, the debugger must clear DBG.HLD_RST. This takes the processor out of reset, and execution begins at the reset vector.

A debugger sets DBG.HLD_RST in one of two ways:

- Either by taking the JTAG state machine into the Capture_DR state, which automatically loads DBG_SR[1] with '1', then the Exit2 state, followed by the Update_Dr state. This will set the DBG.HLD_RST, clear DBG.BRK, and leave the DCSR unchanged (the DCSR bits captured in DBG_SR[34:3] are written back to the DCSR on the Update_DR).
- Alternatively, a '1' can be scanned into DBG_SR[1], with the appropriate value scanned in for the DCSR and DBG.BRK.

DBG.HLD_RST can only be cleared by scanning in a '0' to DBG_SR[1] and scanning in the appropriate values for the DCSR and DBG.BRK.

3.6.11.2.2 DBG.BRK

DBG.BRK allows the debugger to generate an external debug break and asynchronously re-direct execution to a debug handling routine.

A debugger sets an external debug break by scanning data into the DBG_SR with DBG_SR[2] set and the desired value to set the DCSR JTAG writable bits in DBG_SR[34:3].

Once an external debug break is set, it remains set internally until a debug exception occurs. In Monitor mode, external debug breaks detected during abort mode are pended until the processor exits abort mode. In Halt mode, breaks detected during SDS are pended until the processor exits SDS. When an external debug break is detected outside of these two cases, the processor ceases executing instructions as quickly as possible. This minimizes breakpoint skid, by reducing the number of instructions that can execute after the external debug break is requested. However, the processor will continue to process any instructions which may have already begun execution. Debug mode will not be entered until all processor activity has ceased in an orderly fashion.

3.6.11.2.3 DBG.DCSR

The DCSR is updated with the value loaded into DBG.DCSR following an Update_DR. Only bits specified as writable by JTAG in Table 33 are updated.

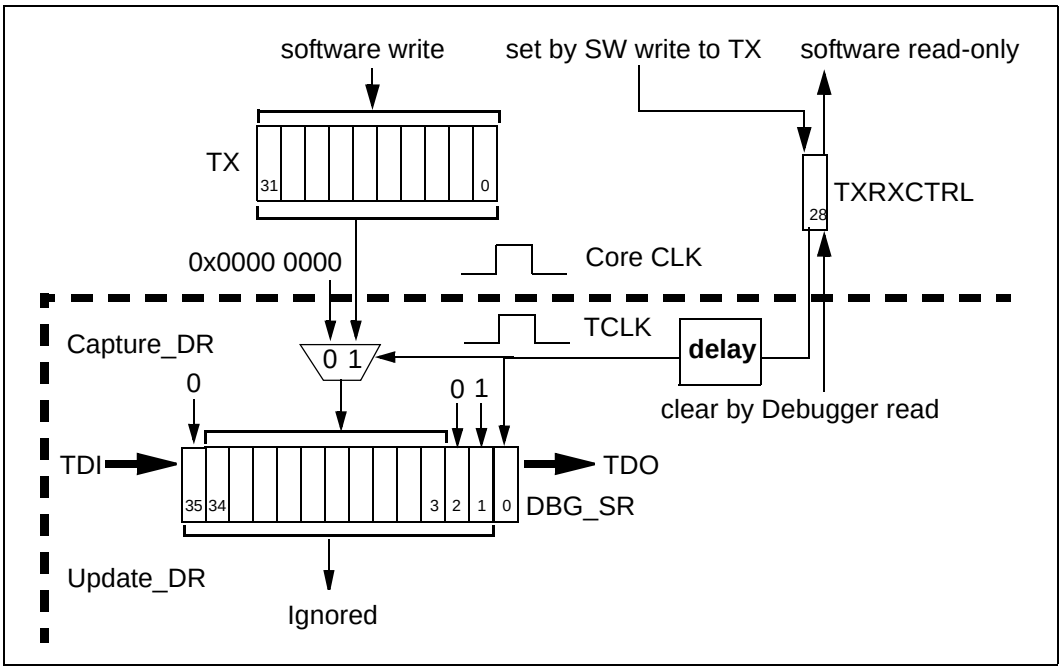
3.6.11.3 DBGTX JTAG Command

The ‘DBGTX’ JTAG instruction selects the DBGTX JTAG data register. The JTAG op code for this instruction is ‘0b10000’. Once the DBGTX data register is selected, the debugger can receive data from the debug handler.

3.6.11.4 DBGTX JTAG Register

The DBGTX JTAG instruction selects the Debug JTAG Data register (Figure 17). The debugger uses the DBGTX data register to poll for breaks (internal and external) to debug mode and once in debug mode, to read data from the debug handler.

Figure 17. DBGTX Hardware



A Capture_DR loads the TX register value into DBG_SR[34:3] and TXRXCTRL[28] into DBG_SR[0]. The other bits in DBG_SR are loaded as shown in Figure 33.

The captured TX value is scanned out during the Shift_DR state.

Data scanned in is ignored on an Update_DR.

A ‘1’ captured in DBG_SR[0] indicates the captured TX data is valid. After doing a Capture_DR, the debugger must place the JTAG state machine in the Shift_DR state to guarantee that a debugger read clears TXRXCTRL[28].

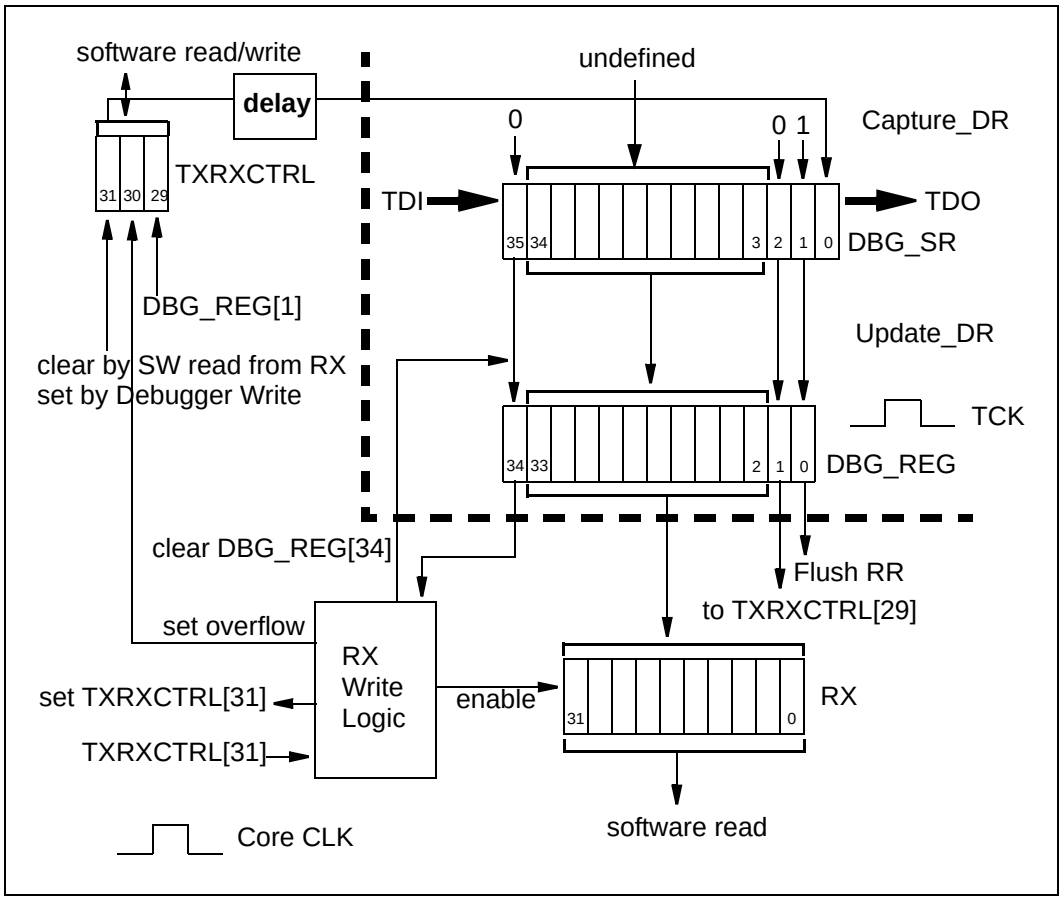
3.6.11.5 DBGRX JTAG Command

The ‘DBGRX’ JTAG instruction selects the DBGRX JTAG data register. The JTAG op code for this instruction is ‘0b00010’. Once the DBGRX data register is selected, the debugger can send data to the debug handler through the RX register.

3.6.11.6 DBGRX JTAG Register

The DBGRX JTAG instruction selects the DBGRX JTAG Data register. The debugger uses the DBGRX data register to send data or commands to the debug handler.

Figure 18. DBGRX Hardware



A Capture_DR loads TXRXCTRL[31] into DBG_SR[0]. The other bits in DBG_SR are loaded as shown in Figure 18.

The captured data is scanned out during the Shift_DR state.

Care must be taken while scanning in data. While polling TXRXCTRL[31], incorrectly setting DBG_SR[35] or DBG_SR[1] may cause unpredictable behavior following an Update_DR.

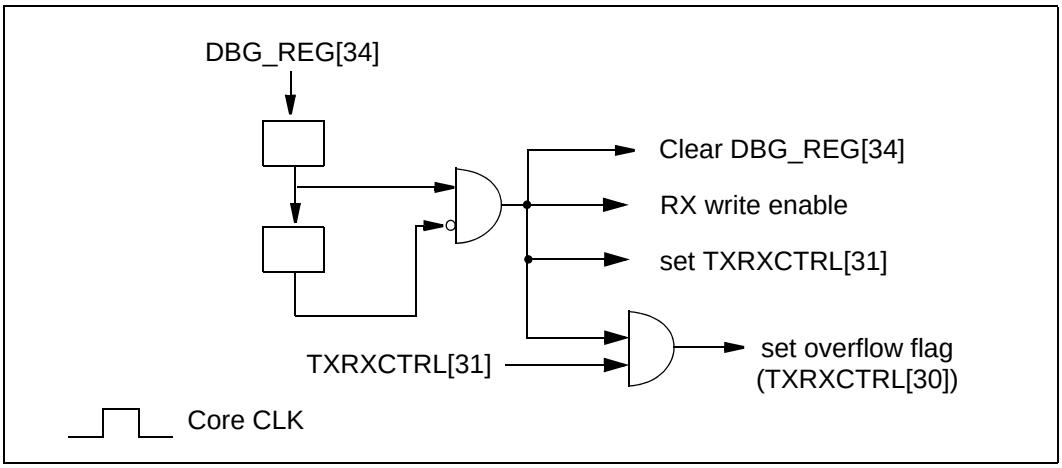
Update_DR parallel loads DBG_SR[35:1] into DBG_REG[34:0]. Whether the new data gets written to the RX register or an overflow condition is detected depends on the inputs to the RX write logic.

3.6.11.6.1 Rx Write Logic

The RX write logic (Figure 20) serves 4 functions:

- Enable the debugger write to RX - the logic ensures only new, valid data from the debugger is written to RX. In particular, when the debugger polls TXRXCTRL[31] to see whether the debug handler has read the previous data from RX. The JTAG state machine must go through Update_DR, which should not modify RX.
- Clear DBG_REG[34] - mainly to support high-speed download. During high-speed download, the debugger continuously scan in a data to send to the debug handler and sets DBG_REG[34] to signal the data is valid. Since DBG_REG[34] is never cleared by the debugger in this case, the '0' to '1' transition used to enable the debugger write to RX would not occur.
- Set TXRXCTRL[31] - When the debugger writes new data to RX, the logic automatically sets TXRXCTRL[31], signalling to the debug handler that the data is valid.
- Set the overflow flag (TXRXCTRL[30] - During high-speed download, the debugger does not poll to see if the handler has read the previous data. If the debug handler stalls long enough, the debugger may overwrite the previous data before the handler can read it. The logic sets the overflow flag when the previous data has not been read yet, and the debugger has just written new data to RX.

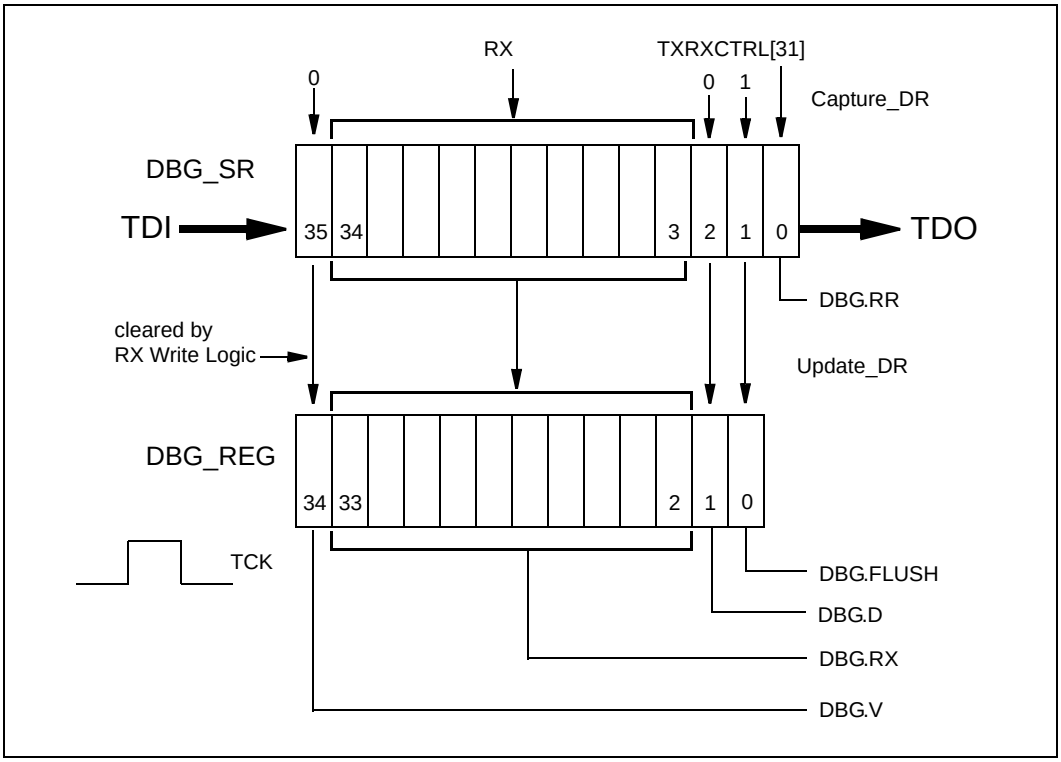
Figure 19. Rx Write Logic



3.6.11.6.2 DBGRX Data Register

The bits in the DBGRX data register (Figure 20) are used by the debugger to send data to the processor. The data register also contains a bit to flush previously written data and a high-speed download flag.

Figure 20. DBGRX Data Register



3.6.11.6.3 DBG.RR

The debugger uses DBG.RR as part of the synchronization that occurs between the debugger and debug handler for accessing RX. This bit contains the value of TXRXCTRL[31] after a Capture_DR. The debug handler automatically sets TXRXCTRL[31] by doing a write to RX.

The debugger polls DBG.RR to determine when the handler has read the previous data from RX.

The debugger sets TXRXCTRL[31] by setting the DBG.V bit.

3.6.11.6.4 DBG.V

The debugger sets this bit to indicate the data scanned into DBG_SR[34:3] is valid data to write to RX. DBG.V is an input to the RX Write Logic and is also cleared by the RX Write Logic.

When this bit is set, the data scanned into the DBG_SR will be written to RX following an Update_DR. If DBG.V is not set and the debugger does an Update_DR, RX will be unchanged.

This bit does not affect the actions of DBG.FLUSH or DBG.D.

3.6.11.6.5 DBG.RX

DBG.RX is written into the RX register based on the output of the RX Write Logic. Any data that needs to be sent from the debugger to the processor must be loaded into DBG.RX with DBG.V set to 1. DBG.RX is loaded from DBG_SR[34:3] when the JTAG enters the Update_DR state.

DBG.RX is written to RX following an Update_DR when the RX Write Logic enables the RX register.

3.6.11.6.6 DBG.D

DBG.D is provided for use during high speed download. This bit is written directly to TXRXCTRL[29]. The debugger sets DBG.D when downloading a block of code or data to IXP42X product line and IXC1100 control plane processors system memory. The debug handler then uses TXRXCTRL[29] as a branch flag to determine the end of the loop.

Using DBG.D as a branch flags eliminates the need for a loop counter in the debug handler code. This avoids the problem were the debugger's loop counter is out of synchronization with the debug handler's counter because of overflow conditions that may have occurred.

3.6.11.6.7 DBG.FLUSH

DBG.FLUSH allows the debugger to flush any previous data written to RX. Setting DBG.FLUSH clears TXRXCTRL[31].

3.6.11.7 Debug JTAG Data Register Reset Values

Upon asserting TRST, the DEBUG data register is reset. Assertion of the reset pin does not affect the DEBUG data register. Table 45 shows the reset and TRST values for the data register. Note: these values apply for DBG_REG for SELDCSR, DBGTX and DBGRX.

Table 45. DEBUG Data Register Reset Values

Bit	TRST	RESET
DBG_REG[0]	0	unchanged
DBG_REG[1]	0	unchanged
DBG_REG[33:2]	unpredictable	unpredictable
DBG_REG[34]	0	unchanged

3.6.12 Trace Buffer

The 256-entry trace buffer provides the ability to capture control flow information to be used for debugging an application. Two modes are supported:

- The buffer fills up completely and generates a debug exception. Then SW empties the buffer.
- The buffer fills up and wraps around until it is disabled. Then SW empties the buffer.

3.6.12.1 Trace Buffer CP Registers

CP14 defines three registers (see Table 46) for use with the trace buffer. These CP14 registers are accessible using MRC, MCR, LDC and STC (CDP to any CP14 registers will cause an undefined instruction trap). The CRn field specifies the number of the register to access. The CRm, opcode_1, and opcode_2 fields are not used and should be set to 0.

Table 46. CP 14 Trace Buffer Register Summary

CP14 Register Number	Register Name
11	Trace Buffer Register (TBREG)
12	Checkpoint 0 Register (CHKPT0)
13	Checkpoint 1 Register (CHKPT1)

Any access to the trace buffer registers in User mode will cause an undefined instruction exception. Specifying registers which do not exist has unpredictable results.

3.6.12.1.1 Checkpoint Registers

When the debugger reconstructs a trace history, it is required to start at the oldest trace buffer entry and construct a trace going forward. In fill-once mode and wrap-around mode when the buffer does not wrap around, the trace can be reconstructed by starting from the point in the code where the trace buffer was first enabled.

The difficulty occurs in wrap-around mode when the trace buffer wraps around at least once. In this case the debugger gets a snapshot of the last N control flow changes in the program, where N <= size of buffer. The debugger does not know the starting address of the oldest entry read from the trace buffer. The checkpoint registers provide reference addresses to help reduce this problem.

Table 47. Checkpoint Register (CHKPTx)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHKPTx																															
reset value: Unpredictable																															

Bits	Access	Description
31:0	Read/Write	**CHKPTx**: target address for corresponding entry in trace buffer

The two checkpoint registers (CHKPT0, CHKPT1) on IXP42X product line and IXC1100 control plane processors provide the debugger with two reference addresses to use for re-constructing the trace history.

When the trace buffer is enabled, reading and writing to either checkpoint register has unpredictable results. When the trace buffer is disabled, writing to a checkpoint register sets the register to the value written. Reading the checkpoint registers returns the value of the register.

In normal usage, the checkpoint registers are used to hold target addresses of specific entries in the trace buffer. Only direct and indirect entries get check-pointed. Exception and roll-over messages are never check-pointed. When an entry is check-pointed, the processor sets bit 6 of the message byte to indicate this (refer to Table 49., Message Byte Formats)

When the trace buffer contains only one check-pointed entry, the corresponding checkpoint register is CHKPT0. When the trace buffer wraps around, two entries will typically be check-pointed, usually about half a buffers length apart. In this case, the first (oldest) check-pointed entry read from the trace buffer corresponds to CHKPT1, the second check-pointed entry corresponds to CHKPT0.

Although the checkpoint registers are provided for wrap-around mode, they are still valid in fill-once mode.

3.6.12.1.2 Trace Buffer Register (TBREG)

The trace buffer is read through TBREG, using MRC and MCR. Software should only read the trace buffer when it is disabled. Reading the trace buffer while it is enabled, may cause unpredictable behavior of the trace buffer. Writes to the trace buffer have unpredictable results.

Reading the trace buffer returns the oldest byte in the trace buffer in the least significant byte of TBREG. The byte is either a message byte or one byte of the 32 bit address associated with an indirect branch message.Table 48 shows the format of the trace buffer register.

Table 48. TBREG Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								Data							

reset value: unpredictable

Bits	Access	Description
31:8	Read-as-Zero/Write-ignored	Reserved
7:0	Read / Write-unpredictable	Message Byte or Address Byte

3.6.13 Trace Buffer Entries

Trace buffer entries consist of either one or five bytes. Most entries are one byte messages indicating the type of control flow change. The target address of the control flow change represented by the message byte is either encoded in the message byte (like for exceptions) or can be determined by looking at the instruction word (like for direct branches). Indirect branches require five bytes per entry. One byte is the message byte identifying it as an indirect branch. The other four bytes make up the target address of the indirect branch. The following sections describe the trace buffer entries in detail.

3.6.13.1 Message Byte

There are two message formats, (exception and non-exception) as shown in Figure 21.

Figure 21. Message Byte Formats

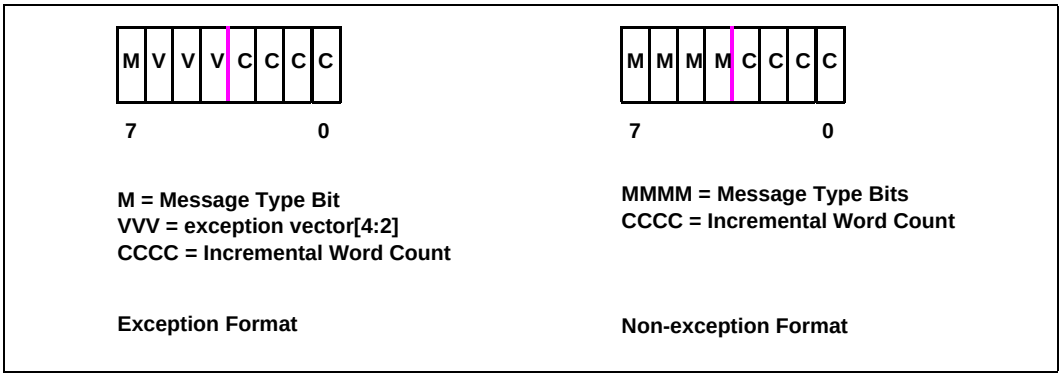


Table 49 shows all of the possible trace messages.

Table 49. Message Byte Formats

Message Name	Message Byte Type	Message Byte Format	# Address Bytes
Exception	exception	0b0VVV CCCC	0
Direct Branch[1]	non-exception	0b1000 CCCC	0
Check-Pointed Direct Branch[1]	non-exception	0b1100 CCCC	0
Indirect Branch[2]	non-exception	0b1001 CCCC	4
Check-Pointed Indirect Branch[2]	non-exception	0b1101 CCCC	4
Roll-over	non-exception	0b1111 1111	0

Notes:
1. Direct branches include ARM and Thumb bl, b.
2. Indirect branches include ARM ldm, ldr, and dproc to PC; ARM and Thumb bx, blx(1) and blx(2); and Thumb pop.

3.6.13.1.1 Exception Message Byte

When any kind of exception occurs, an exception message is placed in the trace buffer. In an exception message byte, the message type bit (M) is always 0.

The vector exception (VVV) field is used to specify bits[4:2] of the vector address (offset from the base of default or relocated vector table). The vector allows the host SW to identify which exception occurred.

The incremental word count (CCCC) is the instruction count since the last control flow change (not including the current instruction for undef, SWI, and pre-fetch abort). The instruction count includes instructions that were executed and conditional instructions that were not executed due to the condition of the instruction not matching the CC flags.

A count value of 0 indicates that 0 instructions executed since the last control flow change and the current exception. For example, if a branch is immediate followed by a SWI, a direct branch exception message (for the branch) is followed by an exception message (for the SWI) in the trace buffer. The count value in the exception message will be 0, meaning that 0 instructions executed after the last control flow change (the branch) and before the current control flow change (the SWI). Instead of the SWI, if an IRQ was handled immediately after the branch (before any other instructions executed), the count would still be 0, since no instructions executed after the branch and before the interrupt was handled.

A count of 0b1111 indicates that 15 instructions executed between the last branch and the exception. In this case, an exception was either caused by the 16th instruction (if it is an undefined instruction exception, pre-fetch abort, or SWI) or handled before the 16th instruction executed (for FIQ, IRQ, or data abort).

3.6.13.1.2 Non-Exception Message Byte

Non-exception message bytes are used for direct branches, indirect branches, and rollovers.

In a non-exception message byte, the four-bit message type field (MMMM) specifies the type of message (refer to Table 49).

The incremental word count (CCCC) is the instruction count since the last control flow change (excluding the current branch). The instruction count includes instructions that were executed and conditional instructions that were not executed due to the condition

of the instruction not matching the CC flags. In the case of back-to-back branches the word count would be 0 indicating that no instructions executed after the last branch and before the current one.

A rollover message is used to keep track of long traces of code that do not have control flow changes. The rollover message means that 16 instructions have executed since the last message byte was written to the trace buffer.

If the incremental counter reaches its maximum value of 15, a rollover message is written to the trace buffer following the next instruction (which will be the 16th instruction to execute). This is shown in Example 13. The count in the rollover message is 0b1111, indicating that 15 instructions have executed after the last branch and before the current non-branch instruction that caused the rollover message.

Example 13. Rollover Messages Examples

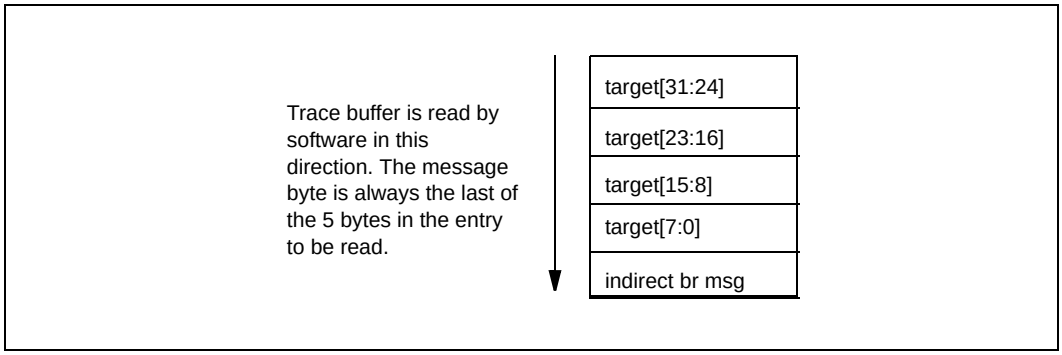
```
count = 5
BL label1          <-- branch message placed in trace buffer after branch executes
count = 0              count = 0b0101
MOV
count = 1
MOV
count = 2
MOV
...

count = 14
MOV
count = 15
MOV                <-- rollover message placed in trace buffer after 16th instruction executes
count = 0              count = 0b1111
```

If the 16th instruction is a branch (direct or indirect), the appropriate branch message is placed in the trace buffer instead of the roll-over message. The incremental counter is still set to 0b1111, meaning 15 instructions executed between the last branch and the current branch.

3.6.13.1.3 Address Bytes

Only indirect branch entries contain address bytes in addition to the message byte. Indirect branch entries always have four address bytes indicating the target of that indirect branch. When reading the trace buffer the MSB of the target address is read out first; the LSB is the fourth byte read out; and the indirect branch message byte is the fifth byte read out. The byte organization of the indirect branch message is shown in Figure 22.

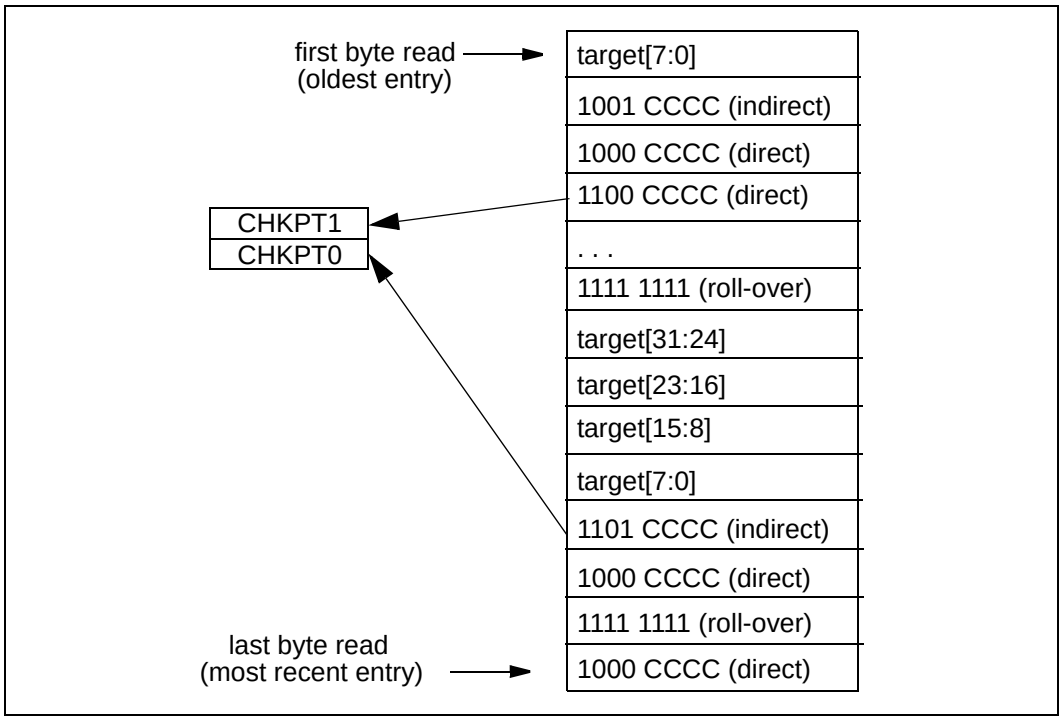
Figure 22. Indirect Branch Entry Address Byte Organization



3.6.13.2 Trace Buffer Usage

IXP42X product line and IXC1100 control plane processors' trace buffer is 256 bytes in length. The first byte read from the buffer represents the oldest trace history information in the buffer. The last (256th) byte read represents the most recent entry in the buffer. The last byte read from the buffer will always be a message byte. This provides the debugger with a starting point for parsing the entries out of the buffer. Because the debugger needs the last byte as a starting point when parsing the buffer, the entire trace buffer must be read (256 bytes on IXP42X product line and IXC1100 control plane processors) before the buffer can be parsed. Figure 23 is a high level view of the trace buffer.

Figure 23. High Level View of Trace Buffer



The trace buffer must be initialized prior to its initial usage, then again prior to each subsequent usage. Initialization is done be reading the entire trace buffer. The process of reading the trace buffer also clears it out (all entries are set to 0b0000 0000), so when the trace buffer has been used to capture a trace, the process of reading the captured trace data also re-initializes the trace buffer for its next usage.

The trace buffer can be used to capture a trace up to a processor reset. A processor reset disables the trace buffer, but the contents are unaffected. The trace buffer captures a trace up to the processor reset.

The trace buffer does not capture reset events or debug exceptions.

Since the trace buffer is cleared out before it is used, all entries are initially 0b0000 0000. In fill-once mode, these 0's can be used to identify the first valid entry in the trace buffer. In wrap around mode, in addition to identifying the first valid entry, these 0 entries can be used to determine whether a wrap around occurred.

As the trace buffer is read, the oldest entries are read first. Reading a series of 5 (or more) consecutive “0b0000 0000” entries in the oldest entries indicates that the trace buffer has not wrapped around and the first valid entry will be the first non-zero entry read out.

Reading 4 or less consecutive “0b0000 0000” entries requires a bit more intelligence in the host SW. The host SW must determine whether these 0s are part of the address of an indirect branch message, or whether they are part of the “0b0000 0000” that the trace buffer was initialized with. If the first non-zero message byte is an indirect branch message, then these 0s are part of the address since the address is always read before the indirect branch message (see “Address Bytes” on page 113). If the first non-zero entry is any other type of message byte, then these 0s indicate that the trace buffer has not wrapped around and that first non-zero entry is the start of the trace.

If the oldest entry from the trace buffer is non-zero, then the trace buffer has either wrapped around or just filled up.

Once the trace buffer has been read and parsed, the host SW should re-create the trace history from oldest trace buffer entry to latest. Trying to re-create the trace going backwards from the latest trace buffer entry may not work in most cases, because once a branch message is encountered, it may not be possible to determine the source of the branch.

In fill-once mode, the return from the debug handler to the application should generate an indirect branch message. The address placed in the trace buffer will be that of the target application instruction. Using this as a starting point, re-creating a trace going forward in time should be straightforward.

In wrap around mode, the host SW should use the checkpoint registers and address bytes from indirect branch entries to re-create the trace going forward. The drawback is that some of the oldest entries in the trace buffer may be untraceable, depending on where the earliest checkpoint (or indirect branch entry) is located. The best case is when the oldest entry in the trace buffer was check-pointed, so the entire trace buffer can be used to re-create the trace. The worst case is when the first checkpoint is in the middle of the trace buffer and no indirect branch messages exist before this checkpoint. In this case, the host SW would have to start at its known address (the first checkpoint) which is half way through the buffer and work forward from there.

3.6.14 Downloading Code in ICache

On IXP42X product line and IXC1100 control plane processors, a 2-K mini instruction cache — physically separate from the 32-K main instruction cache — can be used as an on-chip instruction RAM. An external host can download code directly into either instruction cache through JTAG. In addition to downloading code, several cache functions are supported.

Note: A cache line fill from external memory will never be written into the mini-instruction cache. The only way to load a line into the mini-instruction cache is through JTAG.

The IXP42X product line and IXC1100 control plane processors support loading the instruction cache during reset and during program execution. Loading the instruction cache during normal program execution requires a strict handshaking protocol between software running on the IXP42X product line and IXC1100 control plane processors and the external host.

In the remainder of this section the term ‘instruction cache’ applies to either main or mini instruction cache.

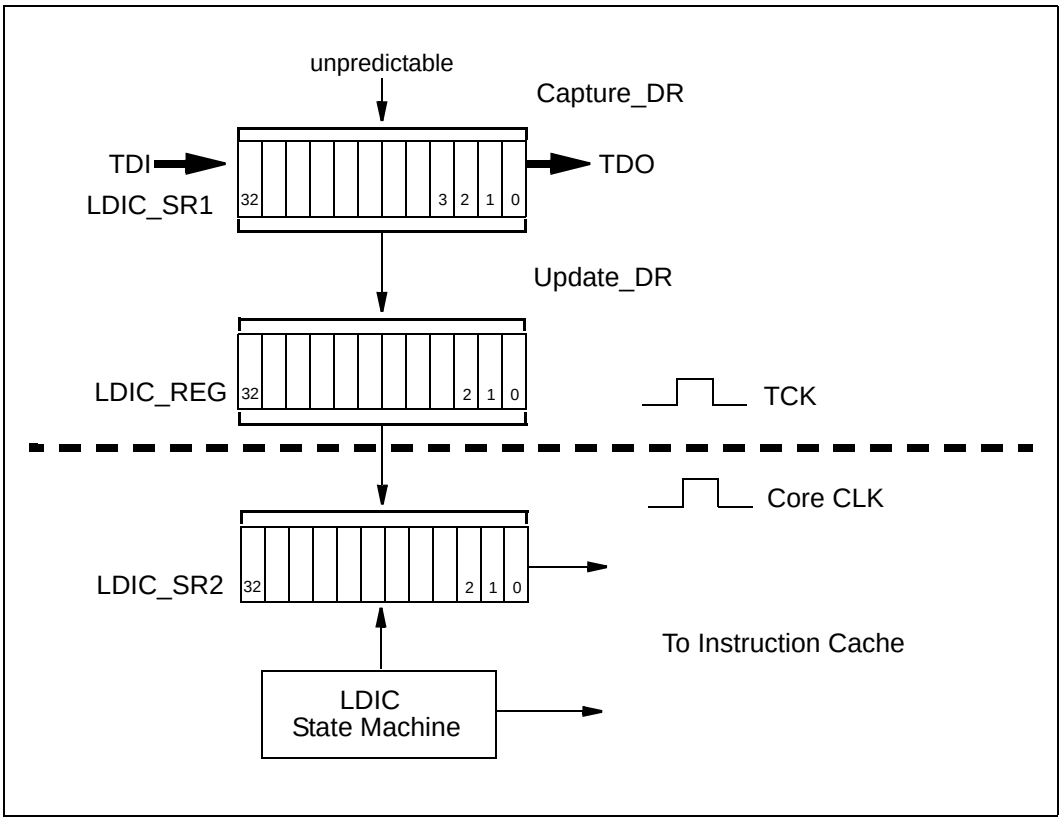
3.6.14.1 LDIC JTAG Command

The LDIC JTAG instruction selects the JTAG data register for loading code into the instruction cache. The JTAG op code for this instruction is ‘00111’. The LDIC instruction must be in the JTAG instruction register in order to load code directly into the instruction cache through JTAG.

3.6.14.2 LDIC JTAG Data Register

The LDIC JTAG Data Register is selected when the LDIC JTAG instruction is in the JTAG IR. An external host can load and invalidate lines in the instruction cache through this data register.

Figure 24. LDIC JTAG Data Register Hardware



The data loaded into LDIC_SR1 during a Capture_DR is unpredictable.

All LDIC functions and data consists of 33-bit packets which are scanned into LDIC_SR1 during the Shift_DR state.

Update_DR parallel loads LDIC_SR1 into LDIC_REG which is then synchronized with the IXP42X product line and IXC1100 control plane processors' clock and loaded into the LDIC_SR2. Once data is loaded into LDIC_SR2, the LDIC State Machine turns on and serially shifts the contents if LDIC_SR2 to the instruction cache.

Note that there is a delay from the time of the Update_DR to the time the entire contents of LDIC_SR2 have been shifted to the instruction cache. Removing the LDIC JTAG instruction from the JTAG IR before the entire contents of LDIC_SR2 are sent to the instruction cache, will result in unpredictable behavior. Therefore, following the Update_DR for the last LDIC packet, the LDIC instruction must remain in the JTAG IR for a minimum of 15 TCKs. This ensures the last packet is correctly sent to the instruction cache.

3.6.14.3 LDIC Cache Functions

The IXP42X product line and IXC1100 control plane processors support four cache functions that can be executed through JTAG. Two functions allow an external host to download code into the main instruction cache or the mini instruction cache through JTAG. Two additional functions are supported to allow lines to be invalidated in the instruction cache. The following table shows the cache functions supported through JTAG.

Table 50. LDIC Cache Functions

Function	Encoding	Arguments	
		Address	# Data Words
Invalidate IC Line	0b000	VA of line to invalidate	0
Invalidate Mini IC	0b001	-	0
Load Main IC	0b010	VA of line to load	8
Load Mini IC	0b011	VA of line to load	8
RESERVED	0b100-0b111	-	-

Invalidate IC line invalidates the line in the instruction cache containing specified virtual address. If the line is not in the cache, the operation has no effect. It does not take any data arguments.

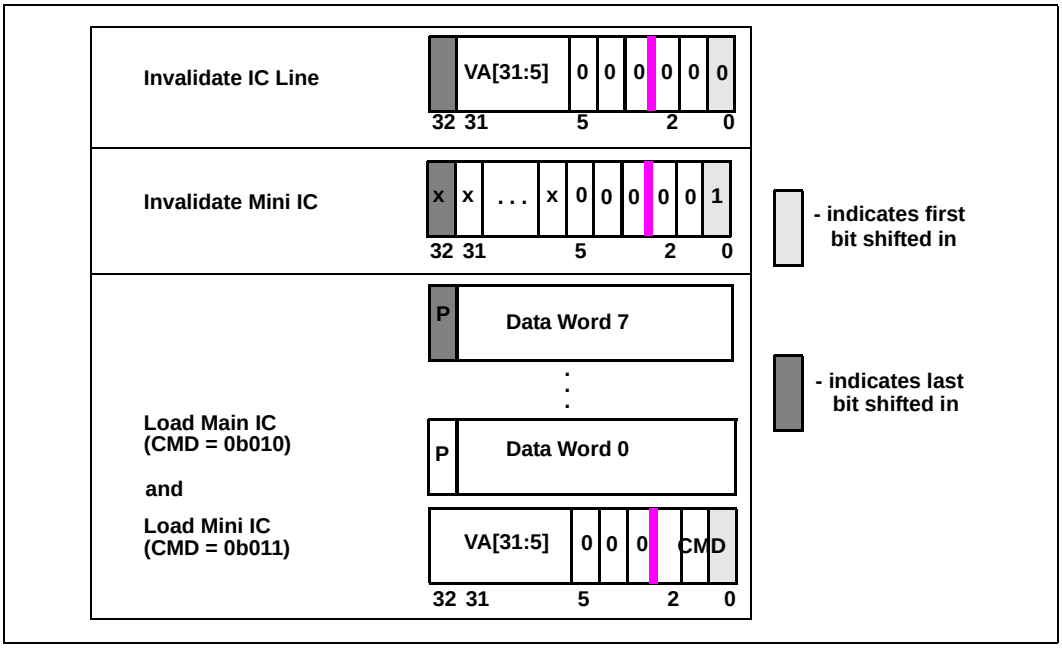
Invalidate Mini IC will invalidate the entire mini instruction cache. It does not effect the main instruction cache. It does not require a virtual address or any data arguments.

Note: The LDIC Invalidate Mini IC function does not invalidate the BTB (like the CP15 Invalidate IC function) so software must do this manually where appropriate.

Load Main IC and Load Mini IC write one line of data (eight ARM instructions) into the specified instruction cache at the specified virtual address.

Each cache function is downloaded through JTAG in 33 bit packets. Figure 25 shows the packet formats for each of the JTAG cache functions. Invalidate IC Line and Invalidate Mini IC each require 1 packet. Load Main IC and Load Mini IC each require 9 packets.

Figure 25. Format of LDIC Cache Functions



All packets are 33 bits in length. Bits [2:0] of the first packet specify the function to execute. For functions that require an address, bits[32:6] of the first packet specify an eight-word aligned address (Packet1[32:6] = VA[31:5]). For Load Main IC and Load Mini IC, eight additional data packets are used to specify eight ARM instructions to be loaded into the target instruction cache. Bits[31:0] of the data packets contain the data to download. Bit[32] of each data packet is the value of the parity for the data in that packet.

As shown in Figure 25, the first bit shifted in TDI is bit 0 of the first packet. After each 33-bit packet, the host must take the JTAG state machine into the Update_DR state. After the host does an Update_DR and returns the JTAG state machine back to the Shift_DR state, the host can immediately begin shifting in the next 33-bit packet.

3.6.14.4 Loading IC During Reset

Code can be downloaded into the instruction cache through JTAG during a processor reset. This feature is used during software debug to download the debug handler prior to starting an application program. The downloaded handler can then intercept the reset vector and do any necessary setup before the application code executes

In general, any code downloaded into the instruction cache through JTAG, must be downloaded to addresses that are not already valid in the instruction cache. Failure to meet this requirement will result in unpredictable behavior by the processor. During a processor reset, the instruction cache is typically invalidated, with the exception of the following modes:

- LDIC mode — Active when LDIC JTAG instruction is loaded in the JTAG IR; prevents the mini instruction cache and the main instruction cache from being invalidated during reset.
- HALT mode — Active when the Halt Mode bit is set in the DCSR; prevents only the mini instruction cache from being invalidated; main instruction cache is invalidated by reset.

During a cold reset (in which both a processor reset and a JTAG reset occurs) it can be guaranteed that the instruction cache will be invalidated since the JTAG reset takes the processor out of any of the modes listed above.

During a warm reset, if a JTAG reset does not occur, the instruction cache is not invalidated by reset when any of the above modes are active. This situation requires special attention if code needs be downloaded during the warm reset.

Note that while Halt Mode is active, reset can invalidate the main instruction cache. Thus debug handler code downloaded during reset can only be loaded into the mini instruction cache. However, code can be dynamically downloaded into the main instruction cache. (refer to "Dynamically Loading IC After Reset" on page 123).

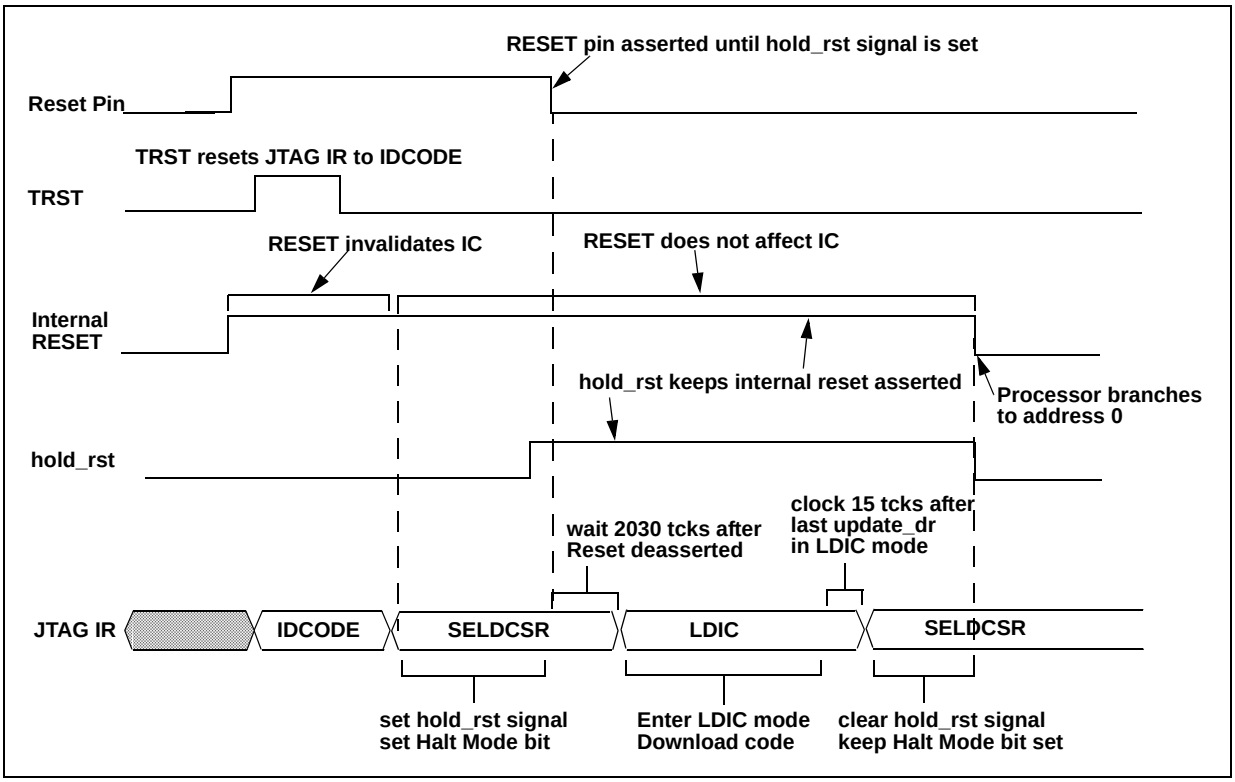
The following sections describe the steps necessary to ensure code is correctly downloaded into the instruction cache.

3.6.14.4.1 Loading IC During Cold Reset for Debug

The Figure 26 shows the actions necessary to download code into the instruction cache during a cold reset for debug.

Note: In the Figure 26 hold_rst is a signal that gets set and cleared through JTAG When the JTAG IR contains the SELDCSR instruction, the hold_rst signal is set to the value scanned into DBG_SR[1].

Figure 26. Code Download During a Cold Reset For Debug



An external host should take the following steps to load code into the instruction cache following a cold reset:

1. Assert the Reset and TRST pins: This resets the JTAG IR to IDCODE and invalidates the instruction cache (main and mini).

2. Load the SELDCSR JTAG instruction into JTAG IR and scan in a value to set the Halt Mode bit in DCSR and to set the hold_rst signal. For details of the SELDCSR, refer to "SELDCSR JTAG Register" on page 103.
3. After hold_rst is set, de-assert the Reset pin. Internally the processor remains held in reset.
4. After Reset is de-asserted, wait 2030 TCKs.
5. Load the LDIC JTAG instruction into JTAG IR.
6. Download code into instruction cache in 33-bit packets as described in "LDIC Cache Functions" on page 118.
7. After code download is complete, clock a minimum of 15 TCKs following the last update_dr in LDIC mode.
8. Place the SELDCSR JTAG instruction into the JTAG IR and scan in a value to clear the hold_rst signal. The Halt Mode bit must remain set to prevent the instruction cache from being invalidated.
9. When hold_rst is cleared, internal reset is de-asserted, and the processor executes the reset vector at address 0.

An additional issue for debug is setting up the reset vector trap. This must be done before the internal reset signal is de-asserted. As described in "Vector Trap Bits (TF,TI,TD,TA,TS,TU,TR)" on page 92, the Halt Mode and the Trap Reset bits in the DCSR must be set prior to de-asserting reset in order to trap the reset vector. There are two possibilities for setting up the reset vector trap:

- The reset vector trap can be set up before the instruction cache is loaded by scanning in a DCSR value that sets the Trap Reset bit in addition to the Halt Mode bit and the hold_rst signal; OR
- The reset vector trap can be set up after the instruction cache is loaded. In this case, the DCSR should be set up to do a reset vector trap, with the Halt Mode bit and the hold_rst signal remaining set.

In either case, when the debugger clears the hold_rst bit to de-assert internal reset, the debugger must set the Halt Mode and Trap Reset bits in the DCSR.

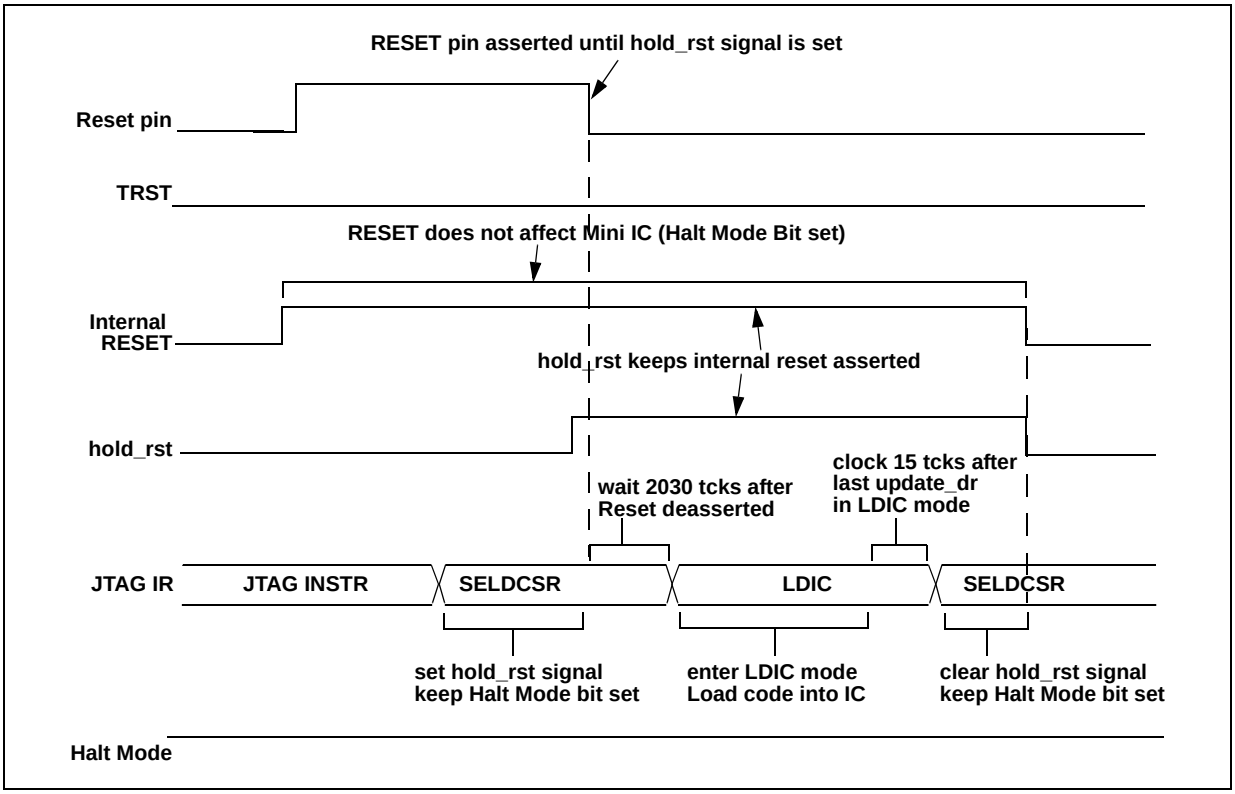
3.6.14.4.2 Loading IC During a Warm Reset for Debug

Loading the instruction cache during a warm reset may be a slightly different situation than during a cold reset. For a warm reset, the main issue is whether the instruction cache gets invalidated by the processor reset or not. There are several possible scenarios:

- While reset is asserted, TRST is also asserted.
  In this case the instruction cache is invalidated, so the actions taken to download code are identical to those described in "Loading IC During Cold Reset for Debug" on page 120
- When reset is asserted, TRST is not asserted, but the processor is not in Halt Mode.
  In this case, the instruction cache is also invalidated, so the actions are the same as described in "Loading IC During Cold Reset for Debug" on page 120, after the LDIC instruction is loaded into the JTAG IR.
- When reset is asserted, TRST is not asserted, and the processor is in Halt Mode.
  In this last scenario, the mini instruction cache does not get invalidated by reset, since the processor is in Halt Mode. This scenario is described in more detail in this section.

In the last scenario described above is shown in Figure 28.

Figure 27. Code Download During a Warm Reset For Debug



As shown in Figure 27, reset does not invalidate the instruction cache because of the processor is in Halt Mode. Since the instruction cache was not invalidated, it may contain valid lines. The host must avoid downloading code to virtual addresses that are already valid in the instruction cache (mini IC or main IC), otherwise the processor may behave unpredictably.

There are several possible solutions that ensure code is not downloaded to a VA that already exists in the instruction cache.

Since the mini instruction cache was not invalidated, any code previously downloaded into the mini IC is valid in the mini IC, so it is not necessary to download the same code again.

If it is necessary to download code into the instruction cache:

1. Assert TRST.
   This clears the Halt Mode bit allowing the instruction cache to be invalidated.
2. Clear the Halt Mode bit through JTAG.
   This allows the instruction cache to be invalidated by reset.
3. Place the LDIC JTAG instruction in the JTAG IR, then proceed with the normal code download, using the Invalidate IC Line function before loading each line.
   This requires 10 packets to be downloaded per cache line instead of the 9 packets described in “LDIC Cache Functions” on page 118

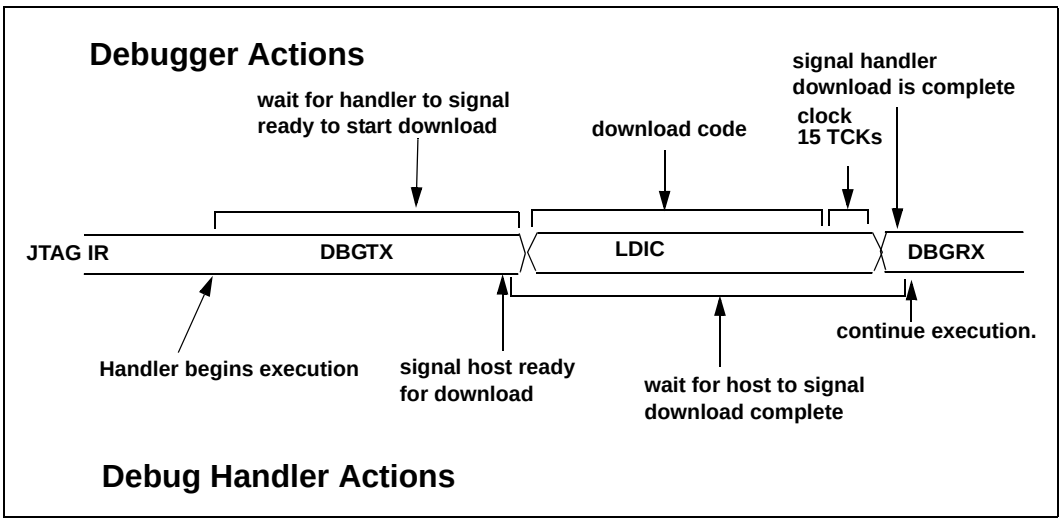
3.6.14.5 Dynamically Loading IC After Reset

An external host can load code into the instruction cache “on the fly” or “dynamically.” This occurs when the host downloads code while the processor is not being reset. However, this requires strict synchronization between the code running on the IXP42X product line and IXC1100 control plane processors and the external host. The guidelines for downloading code during program execution must be followed to ensure proper operation of the processor. The description in this section focuses on using a debug handler running on the IXP42X product line and IXC1100 control plane processors to synchronize with the external host, but the details apply for any application that is running while code is dynamically downloaded.

To dynamically download code during software debug, there must be a minimal debug handler stub, responsible for doing the handshaking with the host, resident in the instruction cache. This debug handler stub should be downloaded into the instruction cache during processor reset using the method described in “Loading IC During Reset” on page 119. “Dynamic Code Download Synchronization” on page 124 describes the details for implementing the handshaking in the debug handler.

Figure 28 shows a high level view of the actions taken by the host and debug handler during dynamic code download.

Figure 28. Downloading Code in IC During Program Execution



The following steps describe the details for downloading code:

- Since the debug handler is responsible for synchronization during the code download, the handler must be executing before the host can begin the download. The debug handler execution starts when the application running on the IXP42X product line and IXC1100 control plane processors generate a debug exception or when the host generates an external debug break.
- While the DBGTX JTAG instruction is in the JTAG IR (see “DBGTX JTAG Command” on page 105), the host polls DBG_SR[0], waiting for the debug handler to set it.
- When the debug handler gets to the point where it is OK to begin the code download, it writes to TX, which automatically sets DBG_SR[0]. This signals the host it is OK to begin the download. The debug handler then begins polling TXRXCTRL[31] waiting for the host to clear it through the DBGRX JTAG register (to indicate the download is complete).
- The host writes LDIC to the JTAG IR, and downloads the code. For each line downloaded, the host must invalidate the target line before downloading code to

that line. Failure to invalidate a line prior to writing it may cause unpredictable operation by the processor.

- When the host completes its download, the host must wait a minimum of 15 TCKs, then switch the JTAG IR to DBGRX, and complete the handshaking (by scanning in a value that sets DBG_SR[35]). This clears TXRXCTL[31] and allows the debug handler code to exit the polling loop. The data scanned into DBG_SR[34:3] is implementation specific.
- After the handler exits the polling loop, it branches to the downloaded code.

Note that this debug handler stub must reside in the instruction cache and execute out of the cache while doing the synchronization. The processor should not be doing any code fetches to external memory while code is being downloaded.

3.6.14.5.1 Dynamic Code Download Synchronization

The following pieces of code are necessary in the debug handler to implement the synchronization used during dynamic code download. The pieces must be ordered in the handler as shown below.

Table 51. Debug-Handler Code to Implement Synchronization During Dynamic Code Download

```
# Before the download can start, all outstanding instruction fetches must
complete.
# The MCR invalidate IC by line function serves as a barrier instruction in
# the core. All outstanding instruction fetches are guaranteed to complete before
# the next instruction executes.

# NOTE1: the actual address specified to invalidate is implementation defined,
but
# must not have any harmful effects.

# NOTE2: The placement of the invalidate code is implementation defined, the only
# requirement is that it must be placed such that by the time the debugger starts
# loading the instruction cache, all outstanding instruction fetches have
completed

    mov r5, address

    mcr p15, 0, r5, c7, c5, 1



# The host waits for the debug handler to signal that it is ready for the
# code download. This can be done using the TX register access handshaking
# protocol. The host polls the TR bit through JTAG until it is set, then begins
# the code download. The following MCR does a write to TX, automatically
# setting the TR bit.

# NOTE: The value written to TX is implementation defined.

    mcr p14, 0, r6, c8, c0, 0



# The debug handler waits until the download is complete before continuing. The
# debugger uses the RX handshaking to signal the debug handler when the download
# is complete. The debug handler polls the RR bit until it is set. A debugger
write
# to RX automatically sets the RR bit, allowing the handler to proceed.

# NOTE: The value written to RX by the debugger is implementation defined - it can
be a bogus value signalling the handler to continue or it can be a target address
for the handler to branch to.

loop:

    mrc     p14, 0, r15, c14, c0, 0       @ handler waits for signal from
debugger

    bpl     loop

    mrc     p14, 0, r0, c8, c0, 0         @ debugger writes target address to RX

    bx      r0
```

In a very simple debug handler stub, the above parts may form the complete handler downloaded during reset (with some handler entry and exit code). When a debug exception occurs, routines can be downloaded as necessary. This basically allows the entire handler to be dynamic.

Another possibility is for a more complete debug handler is downloaded during reset. The debug handler may support some operations, such as read memory, write memory, etc. However, other operations, such as reading or writing a group of CP register, can be downloaded dynamically. This method could be used to dynamically download infrequently used debug handler functions, while the more common operations remain static in the mini-instruction cache.

The Intel Debug Handler is a complete debug handler that implements the more commonly used functions, and allows less frequently used functions to be dynamically downloaded.

3.6.14.6 Mini-Instruction Cache Overview

The mini instruction cache is a smaller version of the main instruction cache. (For more details on the main instruction cache, see "Instruction Cache" on page 52.) It is a 2-Kbyte, two-way set associative cache. There are 32 sets, each containing two ways with each way containing eight words. The cache uses the round-robin replacement policy.

The mini instruction cache is virtually addressed and addresses may be remapped by the PID. However, since the debug handler executes in Special Debug State, address translation and PID remapping are turned off. For application code, accesses to the mini instruction cache use the normal address translation and PID mechanisms.

Normal application code is never cached in the mini instruction cache on an instruction fetch. The only way to get code into the mini instruction cache is through the JTAG LDIC function. Code downloaded into the mini instruction cache is essentially locked - it cannot be overwritten by application code running on the IXP42X product line and IXC1100 control plane processors. However, it is not locked against code downloaded through the JTAG LDIC functions.

Application code can invalidate a line in the mini instruction cache using a CP15 Invalidate IC line function to an address that hits in the mini instruction cache. However, a CP15 global invalidate IC function does not affect the mini instruction cache.

The mini instruction cache can be globally invalidated through JTAG by the LDIC Invalidate IC function or by a processor reset when the processor is not in HALT or LDIC mode. A single line in the mini instruction cache can be invalidated through JTAG by the LDIC Invalidate IC-line function.

3.6.15 Halt Mode Software Protocol

This section describes the overall debug process in Halt Mode. It describes how to start and end a debug session and details for implementing a debug handler. Intel provides a standard Debug Handler that implements some of the techniques in this section. The Intel Debug Handler itself is a a document describing additional handler implementation techniques and requirements.

3.6.15.1 Starting a Debug Session

Prior to starting a debug session in Halt Mode, the debugger must download code into the instruction cache during reset, via JTAG. ("Downloading Code in ICache" on page 116). This downloaded code should consist of:

- A debug handler
- An override default vector table
- An override relocated vector table (if necessary)

While the processor is still in reset, the debugger should set up the DCSR to trap the reset vector. This causes a debug exception to occur immediately when the processor comes out of reset. Execution is redirected to the debug handler allowing the debugger to perform any necessary initialization. The reset vector trap is the only debug exception that can occur with debug globally disabled (DCSR[31]=0). Therefore, the debugger must also enable debug prior to existing the handler to ensure all subsequent debug exceptions correctly break to the debug handler.

3.6.15.1.1 Setting up Override Vector Tables

The override default vector table intercepts the reset vector and branches to the debug handler when a debug exception occurs. If the vector table is relocated, the debug vector is relocated to address 0xffff0000. Thus, an override relocated vector table is required to intercept vector 0xffff0000 and branch to the debug handler.

Both override vector tables also intercept the other debug exceptions, so they must be set up to either branch to a debugger specific handler or go to the application's handlers.

It is possible that the application modifies its vector table in memory, so the debugger may not be able to set up the override vector table to branch to the application's handlers. The Debug Handler may be used to work around this problem by reading memory and branching to the appropriate address. Vector traps can be used to get to the debug handler, or the override vector tables can redirect execution to a debug handler routine that examines memory and branches to the application's handler.

3.6.15.1.2 Placing the Handler in Memory

The debug handler is not required to be placed at a specific pre-defined address. However, there are some limitations on where the handler can be placed due to the override vector tables and the two-way set associative mini instruction cache.

In the override vector table, the reset vector must branch to the debug handler using:

- A direct branch, which limits the start of the handler code to within 32 Mbytes of the reset vector, or
- An indirect branch with a data processing instruction. The data processing instruction creates an address using immediate operands and then branches to the target. An LDR to the PC does not work because the debugger cannot set up data in memory before starting the debug handler

The two-way set associative limitation is due to the fact that when the override default and relocated vector tables are downloaded, they take up both ways of Set 0 (w/ addresses 0x0 and 0xffff0000). Therefore, debug handler code can not be downloaded to an address that maps into Set 0, otherwise it will overwrite one of the vector tables (avoid addresses w/ lower 12 bits=0).

The instruction cache two-way set limitation is not a problem when the reset vector uses a direct branch, since the branch offset can be adjusted accordingly. However, it makes using indirect branches more complicated. Now, the reset vector actually needs multiple data processing instructions to create the target address and branch to it.

One possibility is to set up vector traps on the non-reset exception vectors. These vector locations can then be used to extend the reset vector.

Another solution is to have the reset vector do a direct branch to some intermediate code. This intermediate code can then uses several instructions to create the debug handler start address and branch to it. This would require another line in the mini instruction cache, since the intermediate code must also be downloaded. This method also requires that the layout of the debug handler be well thought out to avoid the intermediate code overwriting a line of debug handler code, or vice versa.

For the indirect branch cases, a temporary scratch register may be necessary to hold intermediate values while computing the final target address. DBG_r13 can be used for this purpose (see “Debug Handler Restrictions” on page 128 for restrictions on DBG_r13 usage).

3.6.15.2 Implementing a Debug Handler

The debugger uses the debug handler to examine or modify processor state by sending commands and reading data through JTAG. The API between the debugger and debug handler is specific to a debugger implementation. Intel provides a standard debug handler and API which can be used by third-party vendors. Issues and details for writing a debug handler are discussed in this section and in the Intel Debug Handler.

3.6.15.2.1 Debug Handler Entry

When the debugger requests an external debug break or is waiting for an internal break, it should poll the TR bit through JTAG to determine when the processor has entered Debug Mode. The debug handler entry code must do a write to TX to signal the debugger that the processor has entered Debug Mode. The write to TX sets the TR bit, signalling the host that a debug exception has occurred and the processor has entered Debug Mode. The value of the data written to TX is implementation defined (debug break message, contents of register to save on host, etc.).

3.6.15.2.2 Debug Handler Restrictions

The Debug Handler executes in Debug Mode which is similar to other privileged processor modes, however, there are some differences. Following are restrictions on Debug Handler code and differences between Debug Mode and other privileged modes.

- The processor is in Special Debug State following a debug exception, and thus has special functionality as described in “Halt Mode” on page 93.
- Although address translation and PID remapping are disabled for instruction accesses (as defined in Special Debug State), data accesses use the normal address translation and PID remapping mechanisms.
- Debug Mode does not have a dedicated stack pointer, DBG_r13. Although DBG_r13 exists, it is not a general purpose register. Its contents are unpredictable and should not be relied upon across any instructions or exceptions. However, DBG_r13 can be used, by data processing (non RRX) and MCR/MRC instructions, as a temporary scratch register.
- The following instructions should not be executed in Debug Mode, they may result in unpredictable behavior:
  - LDM
  - LDR w/ Rd=PC
  - LDR w/ RRX addressing mode
  - SWP
  - LDC
  - STC
- The handler executes in Debug Mode and can be switched to other modes to access banked registers. The handler must not enter User Mode; any User Mode registers that need to be accessed can be accessed in System Mode. Entering User Mode may cause unpredictable behavior.

3.6.15.2.3 Dynamic Debug Handler

On the IXP42X product line and IXC1100 control plane processors, the debug handler and override vector tables reside in the 2-Kbyte, mini instruction cache, separate from the main instruction cache. A “static” Debug Handler is downloaded during reset. This is the base handler code, necessary to do common operations such as handler entry/exit, parse commands from the debugger, read/write ARM registers, read/write memory, etc.

Some functions may require large amounts of code or may not be used very often. As long as there is space in the mini-instruction cache, these functions can be downloaded as part of the static Debug Handler. However, if space is limited, the debug handler also has a dynamic capability that allows a function to be downloaded when it is needed. There are three methods for implementing a dynamic debug handler (using the mini instruction cache, main instruction cache, or external memory). Each method has their limitations and advantages. “Dynamically Loading IC After Reset” on page 123 describes how do dynamically load the mini or main instruction cache.

- Using the Mini IC

  The static debug handler can support a command which can have functionality dynamically mapped to it. This dynamic command does not have any specific functionality associated with it until the debugger downloads a function into the mini instruction cache. When the debugger sends the dynamic command to the handler, new functionality can be downloaded, or the previously downloaded functionality can be used.

  There are also variations in which the debug handler supports multiple dynamic commands, each mapped to a different dynamic function; or a single dynamic command that can branch to one of several downloaded dynamic functions based on a parameter passed by the debugger.

  Debug Handlers that allow code to be dynamically downloaded into the mini instruction cache must be carefully written to avoid inadvertently overwriting a critical piece of debug handler code. Dynamic code is downloaded to the way pointed to by the round-robin pointer. Thus, it is possible for critical debug handler code to be overwritten, if the pointer does not select the expected way.

  To avoid this problem, the debug handler should be written to avoid placing critical code in either way of a set that is intended for dynamic code download. This allows code to be downloaded into either way, and the only code that is overwritten is the previously downloaded dynamic function. This method requires that space within the mini instruction cache be allocated for dynamic download, limiting the space available for the static Debug Handler. Also, the space available may not be suitable for a larger dynamic function.

  Once downloaded, a dynamic function essentially becomes part of the Debug Handler. Since it is in the mini instruction cache, it does not get overwritten by application code. It remains in the cache until it is replaced by another dynamic function or the lines where it is downloaded are invalidated.

- Using the Main IC.

  The steps for downloading dynamic functions into the main instruction cache is similar to downloading into the mini instruction cache. However, using the main instruction cache has its advantages.

  Using the main instruction cache eliminates the problem of inadvertently overwriting static Debug Handler code by writing to the wrong way of a set, since the main and mini instruction caches are separate. The debug handler code does not need to be specially mapped out to avoid this problem. Also, space for dynamic functions does not need to be allocated in the mini instruction cache and dynamic functions are not limited to the size allocated.

  The dynamic function can actually be downloaded anywhere in the address space. The debugger specifies the location of the dynamic function by writing the address to RX when it signals to the handler to continue. The debug handler then does a branch-and-link to that address.

If the dynamic function is already downloaded in the main instruction cache, the debugger immediately downloads the address, signalling the handler to continue.

The static Debug Handler only needs to support one dynamic function command. Multiple dynamic functions can be downloaded to different addresses and the debugger uses the function's address to specify which dynamic function to execute.

Since the dynamic function is being downloaded into the main instruction cache, the downloaded code may overwrite valid application code, and conversely, application code may overwrite the dynamic function. The dynamic function is only guaranteed to be in the cache from the time it is downloaded to the time the debug handler returns to the application (or the debugger overwrites it).

- External memory

  Dynamic functions can also we downloaded to external memory (or they may already exist there). The debugger can download to external memory using the write-memory commands. Then the debugger executes the dynamic command using the address of the function to identify which function to execute. This method has the many of the same advantages as downloading into the main instruction cache.

  Depending on the memory system, this method could be much slower than downloading directly into the instruction cache. Another problem is the application may write to the memory where the function is downloaded. If it can be guaranteed that the application does not modify the downloaded dynamic function, the debug handler can save the time it takes to re-download the code. Otherwise, the ensure the application does not corrupt the dynamic functions, the debugger should re-download any dynamic functions it uses.

For all three methods, the downloaded code executes in the context of the debug handler. The processor will be in Special Debug State, so all of the special functionality applies.

The downloaded functions may also require some common routines from the static debug handler, such as the polling routines for reading RX or writing TX. To simplify the dynamic functions, the debug handler should define a set of registers to contain the addresses of the most commonly used routines. The dynamic functions can then access these routines using indirect branches (BLX). This helps reduce the amount of code in the dynamic function since common routines do not need to be replicated within each dynamic function.

3.6.15.2.4 High-Speed Download

Special debug hardware has been added to support a high-speed download mode to increase the performance of downloads to system memory (vs. writing a block of memory using the standard handshaking).

The basic assumption is that the debug handler can read any data sent by the debugger and write it to memory, before the debugger can send the next data. Thus, in the time it takes for the debugger to scan in the next data word and do an Update_DR, the handler is already in its polling loop, waiting for it. Using this assumption, the debugger does not have to poll RR to see whether the handler has read the previous data - it assumes the previous data has been consumed and immediately starts scanning in the next data word.

The pitfall is when the write to memory stalls long enough that the assumption fails. In this case the download with normal handshaking can be used (or high-speed download can still be used, but a few extra TCKs in the Pause_DR state may be necessary to allow a little more time for the store to complete).

The hardware support for high-speed download includes the Download bit (DCSR[29]) and the Overflow Flag (DCSR[30]).

The download bit acts as a branch flag, signalling to the handler to continue with the download. This removes the need for a counter in the debug handler.

The overflow flag indicates that the debugger attempted to download the next word before the debugger read the previous word.

More details on the Download bit, Overflow flag and high-speed download, in general, can be found in "Transmit/Receive Control Register" on page 98.

Following is example code showing how the Download bit and Overflow flag are used in the debug handler:

Table 52. Debug Handler Code: Download Bit and Overflow Flag

```
hs_write_word_loop:

hs_write_overflow:

    bl      read_RX                @ read data word from host


    @@ read TXRXCTRL into the CCs

    mrc     p14, 0, r15, c14, c0, 0

    bcc     hs_write_done      @ if D bit clear, download complete, exit loop.

    beq     hs_write_overflow  @ if overflow detected, loop until host clears D
bit


    str     r0, [r6], #4       @ store only if there is no overflow.


    b       hs_write_word_loop     @ get next data word


hs_write_done:

    @@ after the loop, if the overflow flag was set, return error message to host

    moveq   r0, #OVERFLOW_RESPONSE

    beq     send_response

    b       write_common_exit
```

3.6.15.3 Ending a Debug Session

Prior to ending a debug session, the debugger should take the following actions:

1. Clear the DCSR (disable debug, exit Halt Mode, clear all vector traps, disable the trace buffer)
2. Turn off all breakpoints.
3. Invalidate the mini instruction cache.
4. Invalidate the main instruction cache.
5. Invalidate the BTB.

These actions ensure that the application program executes correctly after the debugger has been disconnected.

3.6.16 Software Debug Notes and Errata

- Trace buffer message count value on data aborts:
  LDR to non-PC that aborts gets counted in the exception message. But an LDR to the PC that aborts does not get counted on exception message.
- SW Note on data abort generation in Special Debug State.
  - Avoid code that could generate precise data aborts.
  - If this cannot be done, then handler needs to be written such that a memory access is followed by 1 nops. In this case, certain memory operations must be avoided - LDM, STM, STRD, LDC, SWP.
- Data abort on Special Debug State:
  When write-back is on for a memory access that causes a data abort, the base register is updated with the write-back value. This is inconsistent with normal (non-SDS) behavior where the base remains unchanged if write-back is on and a data abort occurs.
- Trace Buffer wraps around and loses data in Halt Mode when configured for fill-once mode:
  It is possible to overflow (and lose) data from the trace buffer in fill-once mode, in Halt Mode. When the trace buffer fills up, it has space for 1 indirect branch message (5 bytes) and 1 exception message (1 Byte).
  If the trace buffer fills up with an indirect branch message and generates a trace buffer full break at the same time as a data abort occurs, the data abort has higher priority, so the processor first goes to the data abort handler. This data abort is placed into the trace buffer without losing any data.
  However, if another imprecise data abort is detected at the start of the data abort handler, it will have higher priority than the trace buffer full break, so the processor will go back to the data abort handler. This 2nd data abort also gets written into the trace buffer. This causes the trace buffer to wrap-around and one trace buffer entry is lost (oldest entry is lost). Additional trace buffer entries can be lost if imprecise data aborts continue to be detected before the processor can handle the trace buffer full break (which will turn off the trace buffer).
  This trace buffer overflow problem can be avoided by enabling vector traps on data aborts.
- TXRXCTRL.RR prevents TX register from being updated (even if TXRXCTRL.TR is clear). This will be fixed on B0-step.
  The problem is that there is incorrect (and unnecessary) interaction between the RX ready (RR) flag and writing the TX register. The debug handler looks at the TX ready bit before writing to the TX register. If this bit is clear, then the handler should be able to write to the TX register. However, in the current implementation even if the TR bit is clear, if the RR bit is set, TX will be unchanged when the handler writes to it. It is OK to prevent a write to TX when the TR bit is set (since the host has not read the previous data in the TX, and we don't want a write to TX to overwrite previous data).
- The TXRXCTRL.OV bit (overflow flag) does not get set during high-speed download when the handler reads the RX register at the same time the debugger writes to it.
  If the debugger writes to RX at the same time the handler reads from RX, the handler read returns the newly written data and the previous data is lost. However, in this specific case, the overflow flag does not get set, so the debugger is unaware that the download was not successful.

3.7 Performance Monitoring

This section describes the performance monitoring facility of the IXP42X product line and IXC1100 control plane processors. The events that are monitored can provide performance information for compiler writers, system application developers and software programmers.

3.7.1 Overview

The IXP42X product line and IXC1100 control plane processors hardware provide four 32-bit performance counters that allow four unique events to be monitored simultaneously. In addition, the IXP42X product line and IXC1100 control plane processors implement a 32-bit clock counter that can be used in conjunction with the performance counters; its main purpose is to count the number of core clock cycles which is useful in measuring total execution time.

The IXP42X product line and IXC1100 control plane processors can monitor either occurrence events or duration events. When counting occurrence events, a counter is incremented each time a specified event takes place and when measuring duration, a counter counts the number of processor clocks that occur while a specified condition is true. If any of the five counters overflow, an interrupt request will occur if it's enabled. Each counter has its own interrupt request enable. The counters continue to monitor events even after an overflow occurs, until disabled by software.

Each of these counters can be programmed to monitor any one of various events.

To further augment performance monitoring, the IXP42X product line and IXC1100 control plane processors clock counter can be used to measure the executing time of an application. This information combined with a duration event can feedback a percentage of time the event occurred with respect to overall execution time.

All of the performance monitoring registers are accessible through Coprocessor 14 (CP14). Refer to Table 27 for more details on accessing these registers with **MRC** and **MCR** coprocessor instructions. Access is allowed in privileged mode only. Note that these registers can't be access with **LDC** or **STC** coprocessor instructions.

Table 53. Performance Monitoring Registers (Sheet 1 of 2)

Description	CRn Register #	CRm Register #	Instruction
(PMNC) Performance Monitor Control Register	0b0000	0b0001	Read: MRC p14, 0, Rd, c0, c1, 0 Write: MCR p14, 0, Rd, c0, c1, 0
(CCNT) Clock Counter Register	0b0001	0b0001	Read: MRC p14, 0, Rd, c1, c1, 0 Write: MCR p14, 0, Rd, c1, c1, 0
(INTEN) Interrupt Enable Register	0b0100	0b0001	Read: MRC p14, 0, Rd, c4, c1, 0 Write: MCR p14, 0, Rd, c4, c1, 0
(FLAG) Overflow Flag Register	0b0101	0b0001	Read: MRC p14, 0, Rd, c5, c1, 0 Write: MCR p14, 0, Rd, c5, c1, 0
(EVTSEL) Event Selection Register	0b1000	0b0001	Read: MRC p14, 0, Rd, c8, c1, 0 Write: MCR p14, 0, Rd, c8, c1, 0
(PMN0) Performance Count Register 0	0b0000	0b0010	Read: MRC p14, 0, Rd, c0, c2, 0 Write: MCR p14, 0, Rd, c0, c2, 0

Table 53. Performance Monitoring Registers (Sheet 2 of 2)

Description	CRn Register #	CRm Register #	Instruction
(PMN1) Performance Count Register 1	0b0001	0b0010	Read: MRC p14, 0, Rd, c1, c2, 0 Write: MCR p14, 0, Rd, c1, c2, 0
(PMN2) Performance Count Register 2	0b0010	0b0010	Read: MRC p14, 0, Rd, c2, c2, 0 Write: MCR p14, 0, Rd, c2, c2, 0
(PMN3) Performance Count Register 3	0b0011	0b0010	Read: MRC p14, 0, Rd, c3, c2, 0 Write: MCR p14, 0, Rd, c3, c2, 0

3.7.2 Register Description

3.7.2.1 Clock Counter (CCNT)

The format of CCNT is shown in Table 54. The clock counter is reset to '0' by setting bit 2 in the Performance Monitor Control Register (PMNC) or can be set to a predetermined value by directly writing to it. It counts core clock cycles. When CCNT reaches its maximum value 0xFFFF,FFFF, the next clock cycle will cause it to roll over to zero and set the overflow flag (bit 0) in FLAG. An interrupt request will occur if it is enabled via bit 0 in INTEN.

Table 54. Clock Count Register (CCNT)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Clock Counter																															
reset value: unpredictable																															

Bits	Access	Description
31:0	Read / Write	**32-bit clock counter** - Reset to '0' by PMNC register. When the clock counter reaches its maximum value 0xFFFF,FFFF, the next cycle will cause it to roll over to zero and generate an interrupt request if enabled.

3.7.2.2 Performance Count Registers (PMN0 - PMN3)

There are four 32-bit event counters; their format is shown in Table 55. The event counters are reset to '0' by setting bit 1 in the PMNC register or can be set to a predetermined value by directly writing to them. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it needs to count will cause it to roll over to zero and set its corresponding overflow flag (bit 1,2,3 or 4) in FLAG. An interrupt request will be generated if its corresponding interrupt enable (bit 1,2,3 or 4) is set in INTEN.

Table 55. Performance Monitor Count Register (PMN0 - PMN3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Event Counter																															

reset value: unpredictable

Bits	Access	Description
31:0	Read / Write	**32-bit event counter** - Reset to '0' by PMNC register. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it needs to count will cause it to roll over to zero and generate an interrupt request if enabled.

3.7.2.3 Performance Monitor Control Register (PMNC)

The performance monitor control register (PMNC) is a coprocessor register that:

- Contains the PMU ID
- Extends CCNT counting by six more bits (cycles between counter rollover = 2^{38})
- Resets all counters to zero
- And enables the entire mechanism

Table 56 shows the format of the PMNC register.

Table 56. Performance Monitor Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID																												D	C	P	E

reset value: E and ID are 0, others unpredictable

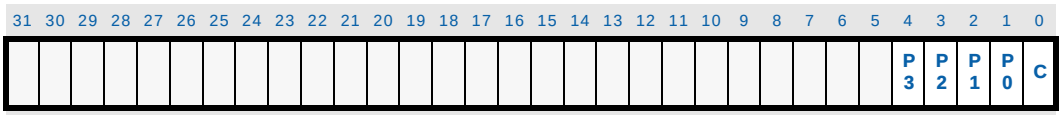
Bits	Access	Description
31:24	Read / Write Ignored	Performance Monitor Identification (ID) - Intel® IXP42X product line and IXC1100 control plane processors = 0x14
23:4	Read-unpredictable / Write-as-0	Reserved
3	Read / Write	**Clock Counter Divider (D)** - 0 = CCNT counts every processor clock cycle 1 = CCNT counts every 64th processor clock cycle
2	Read-unpredictable / Write	**Clock Counter Reset (C)** - 0 = no action 1 = reset the clock counter to '0x0'
1	Read-unpredictable / Write	**Performance Counter Reset (P)** - 0 = no action 1 = reset all performance counters to '0x0'
0	Read / Write	**Enable (E)** - 0 = all counters are disabled 1 = all counters are enabled

3.7.2.4 Interrupt Enable Register

(INTEN)

Each counter can generate an interrupt request when it overflows. INTEN enables interrupt requesting for each counter.

Table 57. Interrupt Enable Register



reset value: [4:0] = 0b00000**, others unpredictable**

Bits	Access	Description
31:5	Read-unpredictable / Write-as-0	Reserved
4	Read / Write	**PMN3 Interrupt Enable (P3)** - 0 = disable interrupt 1 = enable interrupt
3	Read / Write	**PMN2 Interrupt Enable (P2)** - 0 = disable interrupt 1 = enable interrupt
2	Read / Write	**PMN1 Interrupt Enable (P1)** - 0 = disable interrupt 1 = enable interrupt
1	Read / Write	**PMN0 Interrupt Enable (P0)** - 0 = disable interrupt 1 = enable interrupt
0	Read / Write	**CCNT Interrupt Enable (C)** - 0 = disable interrupt 1 = enable interrupt

3.7.2.5 Overflow Flag Status Register

(FLAG)

FLAG identifies which counter has overflowed and also indicates an interrupt has been requested if the overflowing counter's corresponding interrupt enable bit (contained within INTEN) is asserted. An overflow is cleared by writing a '1' to the overflow bit.

Table 58. Overflow Flag Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																											P3	P2	P1	P0	C

reset value: [4:0] = 0b00000, **others unpredictable**

Bits	Access	Description
31:5	Read-unpredictable / Write-as-0	Reserved
4	Read / Write	**PMN3 Overflow Flag (P3)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
3	Read / Write	**PMN2 Overflow Flag (P2)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
2	Read / Write	**PMN1 Overflow Flag (P1)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
1	Read / Write	**PMN0 Overflow Flag (P0)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
0	Read / Write	**CCNT Overflow Flag (C)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit

3.7.2.6 Event Select Register

(EVTSEL)

EVTSEL is used to select events for PMN0, PMN1, PMN2 and PMN3. Refer to Table 60, “Performance Monitoring Events” on page 139 for a list of possible events.

Table 59. Event Select Register

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
evtCount3	evtCount2	evtCount1	evtCount0

reset value: unpredictable

Bits	Access	Description
31:24	Read / Write	**Event Count 3 (evtCount3)** - Identifies the source of events that PMN3 counts. See Table 60 for a description of the values this field may contain.
23:16	Read / Write	**Event Count 2 (evtCount2)** - Identifies the source of events that PMN2 counts. See Table 60 for a description of the values this field may contain.
15:8	Read / Write	**Event Count 1 (evtCount1)** - Identifies the source of events that PMN1 counts. See Table 60 for a description of the values this field may contain.
7:0	Read / Write	**Event Count 0 (evtCount0)** - Identifies the source of events that PMN0 counts. See Table 60 for a description of the values this field may contain.

3.7.3 Managing the Performance Monitor

The following are a few notes about controlling the performance monitoring mechanism:

- An interrupt request will be generated when a counter's overflow flag is set and its associated interrupt enable bit is set in INTEN. The interrupt request will remain asserted until software clears the overflow flag by writing a one to the flag that is set. (Note that the product specific interrupt unit and the CPSR must have enabled the interrupt in order for software to receive it.) The interrupt request can also be deasserted by clearing the corresponding interrupt enable bit. Disabling the facility (PMNC.E) doesn't deassert the interrupt request.
- The counters continue to record events even after they overflow.
- To change an event for a performance counter, first disable the facility (PMNC.E) and then modify EVTSEL. Not doing so will cause unpredictable results.
- To increase the monitoring duration, software can extend the count duration beyond 32 bits by counting the number of overflow interrupts each 32-bit counter generates. This can be done in the interrupt service routine (ISR) where an increment to some memory location every time the interrupt occurs will enable longer durations of performance monitoring. This does intrude upon program execution but is negligible, since the ISR execution time is in the order of tens of cycles compared to the number of cycles it took to generate an overflow interrupt (2^{32}).
- Power can be saved by selecting event 0xFF for any unused event counter. This only applies when other event counters are in use. When the performance monitor is not used at all (PMNC.E = 0x0), hardware ensures minimal power consumption.

3.7.4 Performance Monitoring Events

Table 60 lists events that may be monitored. Each of the Performance Monitor Count Registers (PMN0, PMN1, PMN2, and PMN3) can count any listed event. Software selects which event is counted by each PMNx register by programming the evtCountx fields of EVTSEL.

Table 60. Performance Monitoring Events

Event Number (evtCountx)	Event Definition
0x0	Instruction cache miss requires fetch from external memory.
0x1	Instruction cache cannot deliver an instruction. This could indicate an ICache miss or an ITLB miss. This event will occur every cycle in which the condition is present.
0x2	Stall due to a data dependency. This event will occur every cycle in which the condition is present.
0x3	Instruction TLB miss.
0x4	Data TLB miss.
0x5	Branch instruction executed, branch may or may not have changed program flow. (Counts only B and BL instructions, in both ARM and Thumb mode.)
0x6	Branch incorrectly predicted. (Counts only B and BL instructions, in both ARM and Thumb mode.)
0x7	Instruction executed.
0x8	Stall because the data cache buffers are full. This event will occur every cycle in which the condition is present.
0x9	Stall because the data cache buffers are full. This event will occur once for each contiguous sequence of this type of stall.
0xA	Data cache access, not including Cache Operations (defined in "Register 7: Cache Functions" on page 81)
0xB	Data cache miss, not including Cache Operations (defined in "Register 7: Cache Functions" on page 81)
0xC	Data cache write-back. This event occurs once for each 1/2 line (four words) that are written back from the cache.
0xD	Software changed the PC. All 'b', 'bl', 'blx', 'mov[s] pc, Rm', 'ldm Rn, {Rx, pc}', 'ldr pc, [Rm]', pop {pc} will be counted. An 'mcr p<cp>, 0,pc, ...', will not. The count also does not increment when an event occurs and the PC changes to the event address, e.g., IRQ, FIQ, SWI, etc.
0x10 through 0x17	Reserved.
all others	Reserved, unpredictable results

Some typical combinations of counted events are listed in this section and summarized in Table 61. In this section, we call such an event combination a *mode*.

Table 61. Common Uses of the PMU

Mode	EVTSEL.evtCount0	EVTSEL.evtCount1
Instruction Cache Efficiency	0x7 (instruction count)	0x0 (ICache miss)
Data Cache Efficiency	0xA (Dcache access)	0xB (DCache miss)
Instruction Fetch Latency	0x1 (ICache cannot deliver)	0x0 (ICache miss)
Data/Bus Request Buffer Full	0x8 (DBuffer stall duration)	0x9 (DBuffer stall)
Stall/Writeback Statistics	0x2 (data stall)	0xC (DCache writeback)
Instruction TLB Efficiency	0x7 (instruction count)	0x3 (ITLB miss)
Data TLB Efficiency	0xA (Dcache access)	0x4 (DTLB miss)

Note: PMN0 and PMN1 were used for illustration purposes only. Given there are four event counters, more elaborate combination of events could be constructed. For example, one performance run could select 0xA, 0xB, 0xC, 0x9 events from which data cache

performance statistics could be gathered (like hit rates, number of write-backs per data cache miss, and number of times the data cache buffers fill up per request).

3.7.4.1 Instruction Cache Efficiency Mode

PMN0 totals the number of instructions that were executed, which does not include instructions fetched from the instruction cache that were never executed. This can happen if a branch instruction changes the program flow; the instruction cache may retrieve the next sequential instructions after the branch, before it receives the target address of the branch.

PMN1 counts the number of instruction fetch requests to external memory. Each of these requests loads 32 bytes at a time.

Statistics derived from these two events:

- Instruction cache miss-rate. This is derived by dividing PMN1 by PMN0.
- *The average number of cycles it took to execute an instruction or commonly referred to as cycles-per-instruction (CPI)*. CPI can be derived by dividing CCNT by PMN0, where CCNT was used to measure total execution time.

3.7.4.2 Data Cache Efficiency Mode

PMN0 totals the number of data cache accesses, which includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.

Note that **STM** and **LDM** will each count as several accesses to the data cache depending on the number of registers specified in the register list. **LDRD** will register two accesses.

PMN1 counts the number of data cache and mini-data cache misses. Cache operations do not contribute to this count. See “Register 7: Cache Functions” on page 81 for a description of these operations.

The statistic derived from these two events is:

Data cache miss-rate. This is derived by dividing PMN1 by PMN0.

3.7.4.3 Instruction Fetch Latency Mode

PMN0 accumulates the number of cycles when the instruction-cache is not able to deliver an instruction to the IXP42X product line and IXC1100 control plane processors due to an instruction-cache miss or instruction-TLB miss. This event means that the processor core is stalled.

PMN1 counts the number of instruction fetch requests to external memory. Each of these requests loads 32 bytes at a time. This is the same event as measured in instruction cache efficiency mode.

Statistics derived from these two events:

- *The average number of cycles the processor stalled waiting for an instruction fetch from external memory to return.* This is calculated by dividing PMN0 by PMN1. If the average is high then IXP42X product line and IXC1100 control plane processors may be starved of the bus external to the IXP42X product line and IXC1100 control plane processors.
- *The percentage of total execution cycles the processor stalled waiting on an instruction fetch from external memory to return.* This is calculated by dividing PMN0 by CCNT, which was used to measure total execution time.

3.7.4.4 Data/Bus Request Buffer Full Mode

The Data Cache has buffers available to service cache misses or uncacheable accesses. For every memory request that the Data Cache receives from the processor core a buffer is speculatively allocated in case an external memory request is required or temporary storage is needed for an unaligned access. If no buffers are available, the Data Cache will stall the processor core. How often the Data Cache stalls depends on the performance of the bus external to the IXP42X product line and IXC1100 control plane processors and what the memory access latency is for Data Cache miss requests to external memory. If the IXP42X product line and IXC1100 control plane processors memory access latency is high, possibly due to starvation, these Data Cache buffers will become full. This performance monitoring mode is provided to see if the IXP42X product line and IXC1100 control plane processors are being starved of the bus external to the IXP42X product line and IXC1100 control plane processors, which will effect the performance of the application running on the IXP42X product line and IXC1100 control plane processors.

PMN0 accumulates the number of clock cycles the processor is being stalled due to this condition and PMN1 monitors the number of times this condition occurs.

Statistics derived from these two events:

- *The average number of cycles the processor stalled on a data-cache access that may overflow the data-cache buffers*. This is calculated by dividing PMN0 by PMN1. This statistic lets you know if the duration event cycles are due to many requests or are attributed to just a few requests. If the average is high, the IXP42X product line and IXC1100 control plane processors may be starved of the bus external to the IXP42X product line and IXC1100 control plane processors.
- *The percentage of total execution cycles the processor stalled because a Data Cache request buffer was not available*. This is calculated by dividing PMN0 by CCNT, which was used to measure total execution time.

3.7.4.5 Stall/Write-Back Statistics

When an instruction requires the result of a previous instruction and that result is not yet available, the IXP42X product line and IXC1100 control plane processors stall in order to preserve the correct data dependencies. PMN0 counts the number of stall cycles due to data-dependencies. Not all data-dependencies cause a stall; only the following dependencies cause such a stall penalty:

- Load-use penalty: attempting to use the result of a load before the load completes. To avoid the penalty, software should delay using the result of a load until it's available. This penalty shows the latency effect of data-cache access.
- Multiply/Accumulate-use penalty: attempting to use the result of a multiply or multiply-accumulate operation before the operation completes. Again, to avoid the penalty, software should delay using the result until it's available.
- ALU use penalty: there are a few isolated cases where back to back ALU operations may result in one cycle delay in the execution. These cases are defined in ***Table 3.9, "Performance Considerations" on page 159.***

PMN1 counts the number of write-back operations emitted by the data cache. These write-backs occur when the data cache evicts a dirty line of data to make room for a newly requested line or as the result of clean operation (CP15, register 7).

Statistics derived from these two events:

- *The percentage of total execution cycles the processor stalled because of a data dependency.* This is calculated by dividing PMN0 by CCNT, which was used to measure total execution time. Often a compiler can reschedule code to avoid these penalties when given the right optimization switches.

- Total number of data write-back requests to external memory can be derived solely with PMN1.

3.7.4.6 Instruction TLB Efficiency Mode

PMN0 totals the number of instructions that were executed, which does not include instructions that were translated by the instruction TLB and never executed. This can happen if a branch instruction changes the program flow; the instruction TLB may translate the next sequential instructions after the branch, before it receives the target address of the branch.

PMN1 counts the number of instruction TLB table-walks, which occurs when there is a TLB miss. If the instruction TLB is disabled PMN1 will not increment.

Statistics derived from these two events:

- Instruction TLB miss-rate. This is derived by dividing PMN1 by PMN0.
- *The average number of cycles it took to execute an instruction or commonly referred to as cycles-per-instruction (CPI)*. CPI can be derived by dividing CCNT by PMN0, where CCNT was used to measure total execution time.

3.7.4.7 Data TLB Efficiency Mode

PMN0 totals the number of data cache accesses, which includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.

Note that **STM** and **LDM** will each count as several accesses to the data TLB depending on the number of registers specified in the register list. **LDRD** will register two accesses.

PMN1 counts the number of data TLB table-walks, which occurs when there is a TLB miss. If the data TLB is disabled PMN1 will not increment.

The statistic derived from these two events is:

Data TLB miss-rate. This is derived by dividing PMN1 by PMN0.

3.7.5 Multiple Performance Monitoring Run Statistics

There may be times when more than four events need to be monitored for performance tuning. In this case, multiple performance monitoring runs can be done, capturing different events from each run. For example, the first run could monitor the events associated with instruction cache performance and the second run could monitor the events associated with data cache performance. By combining the results, statistics like total number of memory requests could be derived.

3.7.6 Examples

In this example, the events selected with the Instruction Cache Efficiency mode are monitored and CCNT is used to measure total execution time. Sampling time ends when PMN0 overflows which will generate an IRQ interrupt.

Example 14. Configuring the Performance Monitor

```
; Configure the performance monitor with the following values:
;   EVTSEL.evtCount0 = 7, EVTSEL.evtCount1 = 0 instruction cache efficiency
;   INTEN.inten = 0x7 set all counters to trigger an interrupt on overflow
;   PMNC.C = 1  reset CCNT register
;   PMNC.P = 1  reset PMN0 and PMN1 registers
;   PMNC.E = 1  enable counting
MOV  R0,#0x700
MCR  P14,0,R0,C8,c1,0  ; setup EVTSEL
MOV  R0,#0x7
MCR  P14,0,R0,C4,c1,0  ; setup INTEN
MCR  P14,0,R0,C0,c1,0  ; setup PMNC, reset counters & enable
; Counting begins
```

Counter overflow can be dealt with in the IRQ interrupt service routine as shown below:

Example 15. Interrupt Handling

```
IRQ_INTERRUPT_SERVICE_ROUTINE:
; Assume that performance counting interrupts are the only IRQ in the system
MRC  P14,0,R1,C0,c1,0  ; read the PMNC register
BIC  R2,R1,#1            ; clear the enable bit, preserve other bits in PMNC
MCR  P14,0,R2,C0,c1,0  ; disable counting
MRC  P14,0,R3,C1,c1,0  ; read CCNT register
MRC  P14,0,R4,C0,c2,0  ; read PMN0 register
MRC  P14,0,R5,C1,c2,0  ; read PMN1 register

<process the results>
SUBS PC,R14,#4         ; return from interrupt
```

As an example, assume the following values in CCNT, PMN0, PMN1 and PMNC:

Example 16. Computing the Results

```
; Assume CCNT overflowed


CCNT = 0x0000,0020 ;Overflowed and continued counting

Number of instructions executed = PMN0 = 0x6AAA,AAAA

Number of instruction cache miss requests = PMN1 = 0x0555,5555

Instruction Cache miss-rate = 100 * PMN1/PMN0 = 5%

CPI = (CCNT + 2^32)/Number of instructions executed = 2.4 cycles/instruction
```

In the contrived example above, the instruction cache had a miss-rate of 5% and CPI was 2.4.

3.8 Programming Model

This section describes the programming model of the IXP42X product line and IXC1100 control plane processors, namely the implementation options and extensions to the ARM Version 5TE architecture.

3.8.1 ARM* Architecture Compatibility

The IXP42X product line and IXC1100 control plane processors implement the integer instruction set architecture specified in ARM V5TE. T refers to the thumb instruction set and E refers to the DSP-Enhanced instruction set.

ARM V5TE introduces a few more architecture features over ARM V4, specifically the addition of tiny pages (1 Kbyte), a new instruction that counts the leading zeroes (**CLZ**) in a data value, enhanced ARM-Thumb transfer instructions and a modification of the system control coprocessor, CP15.

3.8.2 ARM* Architecture Implementation Options

3.8.2.1 Big Endian versus Little Endian

The IXP42X product line and IXC1100 control plane processors can operate in big or little endian mode. The B-bit of the Control Register, coprocessor 15, register 1, bit 7 (see Section 3.5.1.2) contained within the IXP42X product line and IXC1100 control plane processors selects the endianess mode of the Intel XScale processor.

Note: This bit takes effect even if the MMU is disabled.

If you choose little endian then you have further options that control whether address and/or data coherency modes. Refer P-Attribute bit in the MMU (see Section 3.1.1.1, “Page (P) Attribute Bit” on page 45) and Expansion Bus Configuration Register 1, Bit 8, BYTE_SWAP_EN (Table 126, “Expansion Bus Configuration Register 1-Bit Definition” on page 325).

Note: The NPEs on the IXP42X product line and IXC1100 control plane processors are **Big-Endian only**; so if you change the endianess of the Intel XScale processor to little endian for your operating system, then this has an impact on how the NPEs and Intel XScale processor exchange data. The Intel® IXP400 Software Release handles this.

3.8.2.2 26-Bit Architecture

The Intel XScale processor does not support 26-bit architecture.

3.8.2.3 Thumb

The Intel XScale processor supports the thumb instruction set.

3.8.2.4 ARM* DSP-Enhanced Instruction Set

The Intel XScale® Processor implements ARM's DSP-enhanced instruction set which is a set of instructions that boost the performance of signal processing applications. There are new multiply instructions that operate on 16-bit data values and new saturation instructions. Saturated instructions are used to ensure accuracy during DSP operations to ensure the signed extension is maintained during an overflow arithmetic operation. Further information on saturated integer arithmetic can be found in the *ARM* Architecture Reference Manual*.

Some of the new instructions are:

- SMLAxy ..... 32<=16x16+32
- SMLAWy ..... 32<=32x16+32
- SMLALxy ..... 64<=16x16+64
- SMULxy ..... 32<=16x16
- SMULWy ..... 32<=32x16
- QADD ..... Adds two registers and saturates the result if an overflow occurred
- QDADD ..... Doubles and saturates one of the input registers then add and saturate
- QSUB ..... Subtracts two registers and saturates the result if an overflow occurred
- QDSUB Doubles and saturates one of the input registers then subtract and saturate

The Intel XScale processor also implements Load Two words (LDRD), Store Two Words (STRD) and cache preload (PLD) instructions with the following implementation notes:

- PLD is interpreted as a read operation by the MMU and is ignored by the data breakpoint unit, i.e., PLD will never generate data breakpoint events.
- PLD to a non-cacheable page performs no action. Also, if the targeted cache line is already resident, this instruction has no affect.
- Both LDRD and STRD instructions will generate an alignment exception when the address bits [2:0] = 0b100.

The transfers of two ARM register values to a coprocessor (MCRR) and the transfer of values from a coprocessor to two ARM registers (MRRC) are only supported on the IXP42X product line and IXC1100 control plane processors when directed to coprocessor 0 and are used to access the internal accumulator. See "Internal Accumulator Access Format" on page 149 for more information. Access to coprocessors 15 and 14 generate an undefined instruction exception.

3.8.2.5 Base Register Update

If a data abort is signalled on a memory instruction that specifies write-back, the contents of the base register will not be updated. This holds for all load and store instructions. This behavior matches that of the first generation ARM processor and is referred to in the ARM V5TE architecture as the *Base Restored Abort Model*.

3.8.3 Extensions to ARM* Architecture

The Intel XScale processor adds a few extensions to the ARM Version 5TE architecture to meet the needs of various markets and design requirements. The following is a list of the extensions which are discussed in the next sections.

- A DSP coprocessor (CP0) has been added that contains a 40-bit accumulator and eight new instructions.
- New page attributes were added to the page table descriptors. The C and B page attribute encoding was extended by one more bit to allow for more encodings: write allocate and mini-data cache.
- Additional functionality has been added to coprocessor 15. Coprocessor 14 was also created.
- Enhancements were made to the Event Architecture, which include instruction cache and data cache parity error exceptions, breakpoint events, and imprecise external data aborts.

3.8.3.1 DSP Coprocessor 0 (CP0)

The Intel XScale processor adds a DSP coprocessor to the architecture for the purpose of increasing the performance and the precision of audio processing algorithms. This coprocessor contains a 40-bit accumulator and eight new instructions.

The 40-bit accumulator is referenced by several new instructions that were added to the architecture; **MIA**, **MIAPH** and **MIAxy** are multiply/accumulate instructions that reference the 40-bit accumulator instead of a register specified accumulator. **MAR** and **MRA** provide the ability to read and write the 40-bit accumulator.

Access to CP0 is always allowed in all processor modes when bit 0 of the Coprocessor Access Register is set. Any access to CP0 when this bit is clear will cause an undefined exception. (See “Register 15: Coprocessor Access Register” on page 85 for more details). Note that only privileged software can set this bit in the Coprocessor Access Register located in CP15.

The 40-bit accumulator will need to be saved on a context switch if multiple processes are using it.

Two new instruction formats were added for coprocessor 0: Multiply with Internal Accumulate Format and Internal Accumulate Access Format. The formats and instructions are described next.

3.8.3.1.1 Multiply With Internal Accumulate Format

A new multiply format has been created to define operations on 40-bit accumulators. Table 7, “MRC/MCR Format” on page 74 shows the layout of the new format. The op code for this format lies within the coprocessor register transfer instruction type. These instructions have their own syntax.

Table 62. Multiply with Internal Accumulate Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	1	0	0	0	1	0	opcode_3				Rs				0	0	0	0	acc			1	Rm			

Bits	Description	Notes
31:28	**cond** - ARM condition codes	-
19:16	**opcode_3** - specifies the type of multiply with internal accumulate	Intel XScale processor defines the following: 0b0000 = **MIA** 0b1000 = **MIAPH** 0b1100 = **MIABB** 0b1101 = **MIABT** 0b1110 = **MIATB** 0b1111 = **MIATT** The effect of all other encodings are unpredictable.
15:12	**Rs** - Multiplier	
7:5	**acc** - select 1 of 8 accumulators	Intel XScale processor only implements acc0; access to any other acc has unpredictable effect.
3:0	**Rm** - Multiplicand	-

Two new fields were created for this format, *acc* and *opcode_3*. The *acc* field specifies one of eight internal accumulators to operate on and *opcode_3* defines the operation for this format. The Intel XScale processor defines a single 40-bit accumulator referred to as *acc0;* future implementations may define multiple internal accumulators. The Intel XScale processor uses opcode_3 to define six instructions, **MIA**, **MIAPH**, **MIABB, MIABT, MIATB and MIATT.**

Table 63. MIA{<cond>} acc0, Rm, Rs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	1	0	0	0	1	0	0	0	0	0	Rs				0	0	0	0	0	0	0	1	Rm			

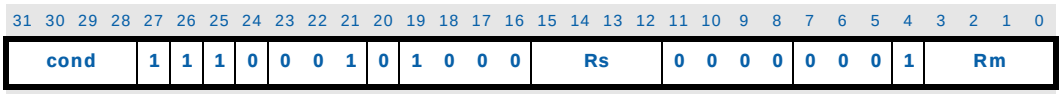
```
Operation:  if ConditionPassed(<cond>) then

                acc0 = (Rm[31:0] * Rs[31:0])[39:0] + acc0[39:0]

Exceptions: none

Qualifiers  Condition Code

            No condition code flags are updated

Notes:      Early termination is supported. Instruction timings can be found

            in “Multiply Instruction Timings” on page 163.

            Specifying R15 for register Rs or Rm has unpredictable results.

            acc0 is defined to be 0b000 on Intel XScale processor.
```

The **MIA** instruction operates similarly to **MLA** except that the 40-bit accumulator is used. **MIA** multiplies the signed value in register Rs (multiplier) by the signed value in register Rm (multiplicand) and then adds the result to the 40-bit accumulator (acc0).

MIA does not support unsigned multiplication; all values in Rs and Rm will be interpreted as signed data values. **MIA** is useful for operating on signed 16-bit data that was loaded into a general purpose register by **LDRSH**.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

Table 64. MIAPH{<cond>} acc0, Rm, Rs




```
Operation:  if ConditionPassed(<cond>) then

                acc0 = sign_extend(Rm[31:16] * Rs[31:16]) +

                       sign_extend(Rm[15:0] * Rs[15:0]) +

                       acc0[39:0]

Exceptions: none

Qualifiers  Condition Code

            S bit is always cleared; no condition code flags are updated

Notes:      Instruction timings can be found
            in "Multiply Instruction Timings" on page 163.

            Specifying R15 for register Rs or Rm has unpredictable results.

            acc0 is defined to be 0b000 on Intel XScale processor
```

The **MIAPH** instruction performs two 16-bit signed multiplies on packed half word data and accumulates these to a single 40-bit accumulator. The first signed multiplication is performed on the lower 16 bits of the value in register Rs with the lower 16 bits of the value in register Rm. The second signed multiplication is performed on the upper 16 bits of the value in register Rs with the upper 16 bits of the value in register Rm. Both signed 32-bit products are sign extended and then added to the value in the 40-bit accumulator (acc0).

The instruction is only executed if the condition specified in the instruction matches the condition code status.

Table 65. MIAxy{<cond>} acc0, Rm, Rs

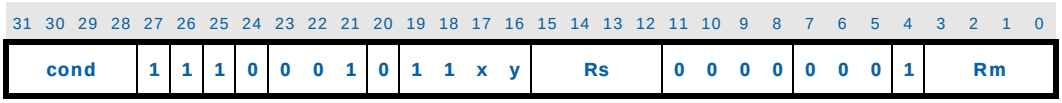


Table 65. MIAxy{<cond>} acc0, Rm, Rs (Continued)

```
Operation:  if ConditionPassed(<cond>) then

                if (bit[17] == 0)

                    <operand1> = Rm[15:0]

                else

                    <operand1> = Rm[31:16]


                if (bit[16] == 0)

                    <operand2> = Rs[15:0]

                else

                    <operand2> = Rs[31:16]


                acc0[39:0] = sign_extend(<operand1> * <operand2>) + acc0[39:0]


Exceptions: none

Qualifiers  Condition Code

            S bit is always cleared; no condition code flags are updated

Notes:      Instruction timings can be found
            in "Multiply Instruction Timings" on page 163.

            Specifying R15 for register Rs or Rm has unpredictable results.

            acc0 is defined to be 0b000 on Intel XScale processor.
```

The **MIAxy** instruction performs one16-bit signed multiply and accumulates these to a single 40-bit accumulator. **x** refers to either the upper half or lower half of register Rm (multiplicand) and **y** refers to the upper or lower half of Rs (multiplier). A value of 0x1 will select bits [31:16] of the register which is specified in the mnemonic as T (for top). A value of 0x0 will select bits [15:0] of the register which is specified in the mnemonic as B (for bottom).

MIAxy does not support unsigned multiplication; all values in Rs and Rm will be interpreted as signed data values.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

3.8.3.1.2 Internal Accumulator Access Format

The Intel XScale processor defines a new instruction format for accessing internal accumulators in CP0. Table 66, "Internal Accumulator Access Format" on page 150 shows that the op code falls into the coprocessor register transfer space.

The *RdHi* and *RdLo* fields allow up to 64 bits of data transfer between ARM registers and an internal accumulator. The *acc* field specifies 1 of 8 internal accumulators to transfer data to/from. The Intel XScale processor implements a single 40-bit accumulator referred to as acc0; future implementations can specify multiple internal accumulators of varying sizes, up to 64 bits.

Access to the internal accumulator is allowed in all processor modes (user and privileged) as long bit 0 of the Coprocessor Access Register is set. (See "Register 15: Coprocessor Access Register" on page 85 for more details).

The IXP42X product line and IXC1100 control plane processors implement two instructions **MAR** and **MRA** that move two ARM registers to acc0 and move acc0 to two ARM registers, respectively.

Table 66. Internal Accumulator Access Format

Bits	Description	Notes
31:28	**cond** - ARM condition codes	-
20	**L** - move to/from internal accumulator 0= move to internal accumulator (MAR) 1= move from internal accumulator (MRA)	-
19:16	**RdHi** - specifies the high order eight (39:32) bits of the internal accumulator.	On a read of the acc, this 8-bit high order field will be sign extended. On a write to the acc, the lower 8 bits of this register will be written to acc[39:32]
15:12	**RdLo** - specifies the low order 32 bits of the internal accumulator	-
7:4	Should be zero	
3	Should be zero	-
2:0	**acc** - specifies 1 of 8 internal accumulators	Intel XScale processor only implements acc0; access to any other acc is unpredictable

Note: **MAR** has the same encoding as **MCRR** (to coprocessor 0) and **MRA** has the same encoding as **MRRC** (to coprocessor 0). These instructions move 64-bits of data to/from ARM registers from/to coprocessor registers. **MCRR** and **MRRC** are defined in ARM's DSP instruction set.

Disassemblers not aware of **MAR** and **MRA** will produce the following syntax:

```
MCRR{<cond>} p0, 0x0, RdLo, RdHi, c0

MRRC{<cond>} p0, 0x0, RdLo, RdHi, c0
```

Table 67. MAR{<cond>} acc0, RdLo, RdHi

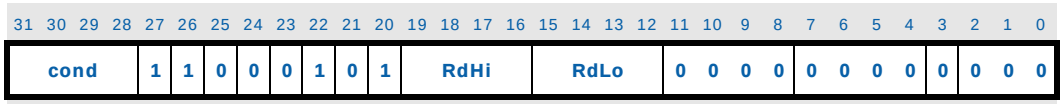

```
Operation:  if ConditionPassed(<cond>) then
                acc0[39:32] = RdHi[7:0]
                acc0[31:0] = RdLo[31:0]

Exceptions: none

Qualifiers  Condition Code
            No condition code flags are updated

Notes:      Instruction timings can be found in

            “Multiply Instruction Timings” on page 163

            Specifying R15 as either RdHi or RdLo has unpredictable results.
```

The MAR instruction moves the value in register RdLo to bits[31:0] of the 40-bit accumulator (acc0) and moves bits[7:0] of the value in register RdHi into bits[39:32] of acc0.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

This instruction executes in any processor mode.

Table 68. MRA{<cond>} RdLo, RdHi, acc0




```
Operation:  if ConditionPassed(<cond>) then
                RdHi[31:0] = sign_extend(acc0[39:32])
                RdLo[31:0] = acc0[31:0]

Exceptions: none

Qualifiers  Condition Code
            No condition code flags are updated

Notes:      Instruction timings can be found in
            “Multiply Instruction Timings” on page 163

            Specifying the same register for RdHi and RdLo has unpredictable
            results.

            Specifying R15 as either RdHi or RdLo has unpredictable results.
```

The MRA instruction moves the 40-bit accumulator value (acc0) into two registers. Bits[31:0] of the value in acc0 are moved into the register RdLo. Bits[39:32] of the value in acc0 are sign extended to 32 bits and moved into the register RdHi.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

This instruction executes in any processor mode.

3.8.3.2 New Page Attributes

The Intel XScale processor extends the page attributes defined by the C and B bits in the page descriptors with an additional X bit. This bit allows four more attributes to be encoded when X=1. These new encodings include allocating data for the mini-data cache and write-allocate caching. A full description of the encodings can be found in “Memory Attributes” on page 45.

The Intel XScale processor retains ARM definitions of the C and B encoding when X = 0, which is different than the ARM products. The memory attribute for the mini-data cache has been moved and replaced with the write-through caching attribute.

When write-allocate is enabled, a store operation that misses the data cache (cacheable data only) will generate a line fill. If disabled, a line fill only occurs when a load operation misses the data cache (cacheable data only).

Write-through caching causes all store operations to be written to memory, whether they are cacheable or not cacheable. This feature is useful for maintaining data cache coherency.

Bit 1 in the Control Register (coprocessor 15, register 1, opcode=1) is used reserved for the P bit memory attribute for memory accesses made during page table walks. The P bit is not implemented on IXP42X product line and IXC1100 control plane processors.

These attributes are programmed in the translation table descriptors, which are highlighted in:

- Table 69, “First-Level Descriptors” on page 153
- Table 70, “Second-Level Descriptors for Coarse Page Table” on page 153
- Table 71, “Second-Level Descriptors for Fine Page Table” on page 153

Two second-level descriptor formats have been defined for the IXP42X product line and IXC1100 control plane processors, one is used for the coarse page table and the other is used for the fine page table.

Table 69. First-Level Descriptors

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ																														0	0
Coarse page table base address																						P	Domain				SBZ			0	1
Section base address								SBZ				TEX				AP		P	Domain				0	C	B	1	0				
Fine page table base address																	SBZ	P	Domain				SBZ							1	1

Table 70. Second-Level Descriptors for Coarse Page Table

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ																														0	0
Large page base address																TEX		AP3		AP2		AP1		AP0			C		B	0	1
Small page base address																		AP3		AP2		AP1		AP0			C		B	1	0
Extended small page base address																		SBZ		TEX				AP			C		B	1	1

Table 71. Second-Level Descriptors for Fine Page Table

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ																														0	0
Large page base address																TEX		AP3		AP2		AP1		AP0			C		B	0	1
Small page base address																		AP3		AP2		AP1		AP0			C		B	1	0
Tiny Page Base Address																				TEX				AP			C		B	1	1

The TEX (Type Extension) field is present in several of the descriptor types. In the Intel XScale processor, only the LSB of this field is defined; this is called the X bit. The remaining bits are reserved for future use and should be programmed as zero (SBZ) on the IXP42X product line and IXC1100 control plane processors.

A Small Page descriptor does not have a TEX field. For these descriptors, TEX is implicitly zero; that is, they operate as if the X bit had a ‘0’ value.

The X bit, when set, modifies the meaning of the C and B bits. Description of page attributes and their encoding can be found in “Memory Management Unit” on page 44.

3.8.3.3 Additions to CP15 Functionality

To accommodate the functionality in the Intel XScale processor, registers in CP15 and CP14 have been added or augmented. See “Configuration” on page 73 for details.

At times it is necessary to be able to guarantee exactly when a CP15 update takes effect. For example, when enabling memory address translation (turning on the MMU), it is vital to know when the MMU is actually guaranteed to be in operation. To address this need, a processor-specific code sequence is defined for the Intel XScale processor. The sequence — called CPWAIT — is shown in Example 12 on page 86.

Example 17. CPWAIT: Canonical Method to Wait for CP15 Update

```
;; The following macro should be used when software needs to be

;; assured that a CP15 update has taken effect.

;; It may only be used while in a privileged mode, because it

;; accesses CP15.


MACRO CPWAIT


        MRC P15, 0, R0, C2, C0, 0            ; arbitrary read of CP15

        MOV R0, R0                           ; wait for it

        SUB PC, PC, #4                       ; branch to next instruction


        ; At this point, any previous CP15 writes are

        ; guaranteed to have taken effect.

ENDM
```

When setting multiple CP15 registers, system software may opt to delay the assurance of their update. This is accomplished by executing CPWAIT only after the sequence of MCR instructions.

The CPWAIT sequence guarantees that CP15 side-effects are complete by the time the CPWAIT is complete. It is possible, however, that the CP15 side-effect will take place before CPWAIT completes or is issued. Programmers should take care that this does not affect the correctness of their code.

3.8.3.4 Event Architecture

3.8.3.4.1 Exception Summary

Table 72 shows all the exceptions that the Intel XScale processor may generate, and the attributes of each. Subsequent sections give details on each exception.

Table 72. Exception Summary (Sheet 1 of 2)

Exception Description	Exception Type[1]	Precise	Updates FAR
Reset	Reset	N	N
FIQ	FIQ	N	N
IRQ	IRQ	N	N
External Instruction	Prefetch	Y	N
Instruction MMU	Prefetch	Y	N
Instruction Cache Parity	Prefetch	Y	N

Notes:
1. Exception types are those described in the *ARM* Architecture Reference Manual*.
2. Refer to "Software Debug" on page 88 for more details.

Table 72. Exception Summary (Sheet 2 of 2)

Exception Description	Exception Type[1]	Precise	Updates FAR
Lock Abort	Data	Y	N
MMU Data	Data	Y	Y
External Data	Data	N	N
Data Cache Parity	Data	N	N
Software Interrupt	Software Interrupt	Y	N
Undefined Instruction	Undefined Instruction	Y	N
Debug Events[2]	varies	varies	N

Notes:
1. Exception types are those described in the *ARM* Architecture Reference Manual*.
2. Refer to "Software Debug" on page 88 for more details.

3.8.3.4.2 Event Priority

The Intel XScale processor follows the exception priority specified in the *ARM* Architecture Reference Manual*. The processor has additional exceptions that might be generated while debugging. For information on these debug exceptions, see "Software Debug" on page 88.

Table 73. Event Priority

Exception	Priority
Reset	1 (Highest)
Data Abort (Precise & Imprecise)	2
FIQ	3
IRQ	4
Prefetch Abort	5
Undefined Instruction, SWI	6 (Lowest)

3.8.3.4.3 Prefetch Aborts

The Intel XScale processor detects three types of prefetch aborts: Instruction MMU abort, external abort on an instruction access, and an instruction cache parity error. These aborts are described in Table 74.

When a prefetch abort occurs, hardware reports the highest priority one in the extended Status field of the Fault Status Register. The value placed in R14_ABORT (the link register in abort mode) is the address of the aborted instruction + 4.

Table 74. Processors' Encoding of Fault Status for Prefetch Aborts

Priority	Sources	FS[10,3:0]*	Domain	FAR
Highest	Instruction MMU Exception Several exceptions can generate this encoding: - translation faults - domain faults, and - permission faults It is up to software to figure out which one occurred.	0b10000	invalid	invalid
	External Instruction Error Exception This exception occurs when the external memory system reports an error on an instruction cache fetch.	0b10110	invalid	invalid
Lowest	Instruction Cache Parity Error Exception	0b11000	invalid	invalid

Note: All other encodings not listed in the table are reserved.

3.8.3.4.4 Data Aborts

Two types of data aborts exist in the Intel XScale processor: precise and imprecise. A precise data abort is defined as one where R14_ABORT always contains the PC (+8) of the instruction that caused the exception. An imprecise abort is one where R14_ABORT contains the PC (+4) of the next instruction to execute and not the address of the instruction that caused the abort. In other words, instruction execution will have advanced beyond the instruction that caused the data abort.

On the Intel XScale processor precise data aborts are recoverable and imprecise data aborts are not recoverable.

Precise Data Aborts

- A lock abort is a precise data abort; the extended Status field of the Fault Status Register is set to 0xb10100. This abort occurs when a lock operation directed to the MMU (instruction or data) or instruction cache causes an exception, due to either a translation fault, access permission fault or external bus fault.

  The Fault Address Register is undefined and R14_ABORT is the address of the aborted instruction + 8.
- A data MMU abort is precise. These are due to an alignment fault, translation fault, domain fault, permission fault or external data abort on an MMU translation. The status field is set to a predetermined ARM definition which is shown in Table 75.

  The Fault Address Register is set to the effective data address of the instruction and R14_ABORT is the address of the aborted instruction + 8.

Table 75. Intel XScale® Processor Encoding of Fault Status for Data Aborts (Sheet 1 of 2)

Priority	Sources		FS[10,3:0]*	Domain	FAR
Highest	Alignment		0b000x1	invalid	valid
	External Abort on Translation	First level Second level	0b01100 0b01110	invalid valid	valid valid
	Translation	Section Page	0b00101 0b00111	invalid valid	valid valid
	Domain	Section Page	0b01001 0b01011	valid valid	valid valid
	Permission	Section Page	0b01101 0b01111	valid valid	valid valid

Note: All other encodings not listed in the table are reserved.

Table 75. Intel XScale® Processor Encoding of Fault Status for Data Aborts (Sheet 2 of 2)

Priority	Sources	FS[10,3:0]*	Domain	FAR
	Lock Abort This data abort occurs on an MMU lock operation (data or instruction TLB) or on an Instruction Cache lock operation.	0b10100	invalid	invalid
	Imprecise External Data Abort	0b10110	invalid	invalid
Lowest	Data Cache Parity Error Exception	0b11000	invalid	invalid

Note: All other encodings not listed in the table are reserved.

Imprecise Data Aborts

- A data cache parity error is imprecise; the extended Status field of the Fault Status Register is set to 0xb11000.
- All external data aborts except for those generated on a data MMU translation are imprecise.

The Fault Address Register for all imprecise data aborts is undefined and R14_ABORT is the address of the next instruction to execute + 4, which is the same for both ARM and Thumb mode.

Although the Intel XScale processor guarantees the *Base Restored Abort Model* for precise aborts, it cannot do so in the case of imprecise aborts. A Data Abort handler may encounter an updated base register if it is invoked because of an imprecise abort.

Imprecise data aborts may create scenarios that are difficult for an abort handler to recover. Both external data aborts and data cache parity errors may result in corrupted data in the targeted registers. Because these faults are imprecise, it is possible that the corrupted data will have been used before the Data Abort fault handler is invoked. Because of this, software should treat imprecise data aborts as unrecoverable.

Note that even memory accesses marked as “stall until complete” (see “Details on Data Cache and Write Buffer Behavior” on page 46) can result in imprecise data aborts. For these types of accesses, the fault is somewhat less imprecise than the general case: it is guaranteed to be raised within three instructions of the instruction that caused it. In other words, if a “stall until complete” LD or ST instruction triggers an imprecise fault, then that fault will be seen by the program within three instructions.

With this knowledge, it is possible to write code that accesses “stall until complete” memory with impunity. Simply place several NOP instructions after such an access. If an imprecise fault occurs, it will do so during the NOPs; the data abort handler will see identical register and memory state as it would with a precise exception, and so should be able to recover. An example of this is shown in Example 18 on page 158.

Example 18. Shielding Code from Potential Imprecise Aborts

```
;; Example of code that maintains architectural state through the

;; window where an imprecise fault might occur.


       LD      R0, [R1]                      ; R1 points to stall-until-complete

                                             ;   region of memory

       NOP

       NOP

       NOP

       ; Code beyond this point is guaranteed not to see any aborts

       ; from the LD.
```

If a system design precludes events that could cause external aborts, then such precautions are not necessary.

Multiple Data Aborts

Multiple data aborts may be detected by hardware but only the highest priority one will be reported. If the reported data abort is precise, software can correct the cause of the abort and re-execute the aborted instruction. If the lower priority abort still exists, it will be reported. Software can handle each abort separately until the instruction successfully executes.

If the reported data abort is imprecise, software needs to check the SPSR to see if the previous context was executing in abort mode. If this is the case, the link back to the current process has been lost and the data abort is unrecoverable.

3.8.3.4.5 Events from Preload Instructions

A **PLD** instruction will never cause the Data MMU to fault for any of the following reasons:

- Domain Fault
- Permission Fault
- Translation Fault

If execution of the **PLD** would cause one of the above faults, then the **PLD** causes no effect.

This feature allows software to issue **PLD**s speculatively. For example, Example 19 on page 159 places a **PLD** instruction early in the loop. This **PLD** is used to fetch data for the next loop iteration. In this example, the list is terminated with a node that has a null pointer. When execution reaches the end of the list, the **PLD** on address 0x0 will not cause a fault. Rather, it will be ignored and the loop will terminate normally.

Example 19. Speculatively issuing PLD

```
;; R0 points to a node in a linked list. A node has the following layout:
;; Offset  Contents
;;---------------------------------
;;      0   data
;;      4   pointer to next node
;; This code computes the sum of all nodes in a list. The sum is placed into R9.
;;
        MOV R9, #0      ; Clear accumulator
sumList:
        LDR R1, [R0, #4]   ; R1 gets pointer to next node
        LDR R3, [R0]       ; R3 gets data from current node
        PLD [R1]           ; Speculatively start load of next node
        ADD R9, R9, R3     ; Add into accumulator
        MOVS R0, R1        ; Advance to next node. At end of list?
        BNE sumList        ; If not then loop
```

3.8.3.4.6 Debug Events

Debug events are covered in "Debug Exceptions" on page 92.

3.9 Performance Considerations

This section describes relevant performance considerations that compiler writers, application programmers and system designers need to be aware of to efficiently use the IXP42X product line and IXC1100 control plane processors. Performance numbers discussed here include interrupt latency, branch prediction, and instruction latencies.

3.9.1 Interrupt Latency

Minimum Interrupt Latency is defined as the minimum number of cycles from the assertion of any interrupt signal (IRQ or FIQ) to the execution of the instruction at the vector for that interrupt. This number assumes best case conditions exist when the interrupt is asserted, e.g., the system isn't waiting on the completion of some other operation.

A sometimes more useful number to work with is the *Maximum Interrupt Latency*. This is typically a complex calculation that depends on what else is going on in the system at the time the interrupt is asserted. Some examples that can adversely affect interrupt latency are:

- The instruction currently executing could be a 16-register LDM
- The processor could fault just when the interrupt arrives
- The processor could be waiting for data from a load, doing a page table walk, etc.
- High core-to-system (bus) clock ratios

Maximum Interrupt Latency can be reduced by:

- Ensuring that the interrupt vector and interrupt service routine are resident in the instruction cache. This can be accomplished by locking them down into the cache.
- Removing or reducing the occurrences of hardware page table walks. This also can be accomplished by locking down the application's page table entries into the TLBs, along with the page table entry for the interrupt service routine.

3.9.2 Branch Prediction

The IXP42X product line and IXC1100 control plane processors implement dynamic branch prediction for the ARM instructions **B** and **BL** and for the thumb instruction **B**. Any instruction that specifies the PC as the destination is predicted as not taken. For example, an **LDR** or a **MOV** that loads or moves directly to the PC will be predicted not taken and incur a branch latency penalty.

These instructions — ARM **B,** ARM **BL** and thumb **B --** enter into the branch target buffer when they are "taken" for the first time. (A "taken" branch refers to when they are evaluated to be true.) Once in the branch target buffer, IXP42X product line and IXC1100 control plane processors dynamically predict the outcome of these instructions based on previous outcomes. Table 76 shows the branch latency penalty when these instructions are correctly predicted and when they are not. A penalty of zero for correct prediction means that the IXP42X product line and IXC1100 control plane processors can execute the next instruction in the program flow in the cycle following the branch.

Table 76. Branch Latency Penalty

Core Clock Cycles		Description
ARM*	Thumb*	
+0	+ 0	***Predicted Correctly***. The instruction is in the branch target cache and is correctly predicted.
+4	+ 5	***Mispredicted***. There are three occurrences of branch misprediction, all of which incur a 4-cycle branch delay penalty. 1. The instruction is in the branch target buffer and is predicted not-taken, but is actually taken. 2. The instruction is not in the branch target buffer and is a taken branch. 3. The instruction is in the branch target buffer and is predicted taken, but is actually not-taken

3.9.3 Addressing Modes

All load and store addressing modes implemented in the IXP42X product line and IXC1100 control plane processors do not add to the instruction latencies numbers.

3.9.4 Instruction Latencies

The latencies for all the instructions are shown in the following sections with respect to their functional groups: branch, data processing, multiply, status register access, load/store, semaphore, and coprocessor.

The following section explains how to read these tables.

3.9.4.1 Performance Terms

- Issue Clock (cycle 0)
  The first cycle when an instruction is decoded ***and*** allowed to proceed to further stages in the execution pipeline (i.e., when the instruction is actually issued).

- Cycle Distance from A to B

  The cycle distance from cycle ***A*** to cycle ***B*** is ***(B-A)*** -- that is, the number of cycles from the start of cycle ***A*** to the start of cycle ***B***. Example: the cycle distance from cycle 3 to cycle 4 is one cycle.

- Issue Latency

  The cycle distance ***from*** the first issue clock of the current instruction ***to*** the issue clock of the next instruction. The actual number of cycles can be influenced by cache-misses, resource-dependency stalls, and resource availability conflicts.

- Result Latency

  The cycle distance ***from*** the first issue clock of the current instruction ***to*** the issue clock of the first instruction that can use the result without incurring a resource dependency stall. The actual number of cycles can be influenced by cache-misses, resource-dependency stalls, and resource availability conflicts.

- Minimum Issue Latency (without Branch Misprediction)

  The minimum cycle distance ***from*** the issue clock of the current instruction ***to*** the first possible issue clock of the next instruction assuming best case conditions (i.e., that the issuing of the next instruction is not stalled due to a resource dependency stall; the next instruction is immediately available from the cache or memory interface; the current instruction does not incur resource dependency stalls during execution that can not be detected at issue time; and if the instruction uses dynamic branch prediction, correct prediction is assumed).

- Minimum Result Latency

  The required minimum cycle distance ***from*** the issue clock of the current instruction ***to*** the issue clock of the first instruction that can use the result without incurring a resource dependency stall assuming best case conditions (i.e., that the issuing of the next instruction is not stalled due to a resource dependency stall; the next instruction is immediately available from the cache or memory interface; and the current instruction does not incur resource dependency stalls during execution that can not be detected at issue time).

- Minimum Issue Latency (with Branch Misprediction)

  The minimum cycle distance ***from*** the issue clock of the current branching instruction ***to*** the first possible issue clock of the next instruction. This definition is identical to *Minimum Issue Latency* except that the branching instruction has been incorrectly predicted. It is calculated by adding *Minimum Issue Latency (without Branch Misprediction)* to the minimum branch latency penalty number from Table 76, which is four cycles.

- Minimum Resource Latency

  The minimum cycle distance from the issue clock of the current multiply instruction to the issue clock of the next multiply instruction assuming the second multiply does not incur a data dependency and is immediately available from the instruction cache or memory interface.

  For the following code fragment, here is an example of computing latencies:

Example 20. Computing Latencies

```
UMLALr6,r8,r0,r1

ADD r9,r10,r11

SUB r2,r8,r9

MOV r0,r1
```

Table 77 shows how to calculate Issue Latency and Result Latency for each instruction. Looking at the issue column, the **UMLAL** instruction starts to issue on cycle 0 and the next instruction, **ADD,** issues on cycle 2, so the Issue Latency for **UMLAL** is two. From

the code fragment, there is a result dependency between the **UMLAL** instruction and the **SUB** instruction. In Table 77, **UMLAL** starts to issue at cycle 0 and the **SUB** issues at cycle 5. thus the Result Latency is five.

Table 77. Latency Example

Cycle	Issue	Executing
0	umlal (1st cycle)	--
1	umlal (2nd cycle)	umlal
2	add	umlal
3	sub (stalled)	umlal & add
4	sub (stalled)	umlal
5	sub	umlal
6	mov	sub
7	--	mov

3.9.4.2 Branch Instruction Timings

Table 78. Branch Instruction Timings (Those Predicted by the BTB)

Mnemonic	Minimum Issue Latency When Correctly Predicted by the BTB	Minimum Issue Latency with Branch Misprediction
B	1	5
BL	1	5

Table 79. Branch Instruction Timings (Those not Predicted by the BTB)

Mnemonic	Minimum Issue Latency When the Branch is not Taken	Minimum Issue Latency When the Branch is Taken
BLX(1)	N/A	5
BLX(2)	1	5
BX	1	5
Data Processing Instruction with PC as the destination	Same as Table 80	4 + numbers in Table 80
LDR PC,< >	2	8
LDM with PC in register list	3 + numreg*	10 + max (0, numreg-3)

Note: numreg is the number of registers in the register list including the PC.

3.9.4.3 Data Processing Instruction Timings

Table 80. Data Processing Instruction Timings (Sheet 1 of 2)

Mnemonic	<shifter operand> is NOT a Shift/Rotate by Register		<shifter operand> is a Shift/Rotate by Register OR <shifter operand> is RRX	
	Minimum Issue Latency	Minimum Result Latency*	Minimum Issue Latency	Minimum Result Latency*
ADC	1	1	2	2
ADD	1	1	2	2
AND	1	1	2	2
BIC	1	1	2	2

Note: If the next instruction needs to use the result of the data processing for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

Table 80. Data Processing Instruction Timings (Sheet 2 of 2)

Mnemonic	<shifter operand> is NOT a Shift/Rotate by Register		<shifter operand> is a Shift/Rotate by Register OR <shifter operand> is RRX	
	Minimum Issue Latency	Minimum Result Latency*	Minimum Issue Latency	Minimum Result Latency*
CMN	1	1	2	2
CMP	1	1	2	2
EOR	1	1	2	2
MOV	1	1	2	2
MVN	1	1	2	2
ORR	1	1	2	2
RSB	1	1	2	2
RSC	1	1	2	2
SBC	1	1	2	2
SUB	1	1	2	2
TEQ	1	1	2	2
TST	1	1	2	2

Note: If the next instruction needs to use the result of the data processing for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

3.9.4.4 Multiply Instruction Timings

Table 81. Multiply Instruction Timings (Sheet 1 of 2)

Mnemonic	Rs Value (Early Termination)	S-Bit Value	Minimum Issue Latency	Minimum Result Latency*	Minimum Resource Latency (Throughput)
MLA	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	2	1
		1	2	2	2
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	3	2
		1	3	3	3
	all others	0	1	4	3
		1	4	4	4
MUL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	2	1
		1	2	2	2
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	3	2
		1	3	3	3
	all others	0	1	4	3
		1	4	4	4

Note: If the next instruction needs to use the result of the multiply for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

Table 81. Multiply Instruction Timings (Sheet 2 of 2)

Mnemonic	Rs Value (Early Termination)	S-Bit Value	Minimum Issue Latency	Minimum Result Latency*	Minimum Resource Latency (Throughput)
SMLAL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	2	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	2	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	2	RdLo = 4; RdHi = 5	4
		1	5	5	5
SMLALxy	N/A	N/A	2	RdLo = 2; RdHi = 3	2
SMLAWy	N/A	N/A	1	3	2
SMLAxy	N/A	N/A	1	2	1
SMULL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	1	RdLo = 4; RdHi = 5	4
		1	5	5	5
SMULWy	N/A	N/A	1	3	2
SMULxy	N/A	N/A	1	2	1
UMLAL	Rs[31:15] = 0x00000	0	2	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00	0	2	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	2	RdLo = 4; RdHi = 5	4
		1	5	5	5
UMULL	Rs[31:15] = 0x00000	0	1	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00	0	1	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	1	RdLo = 4; RdHi = 5	4
		1	5	5	5

Note: If the next instruction needs to use the result of the multiply for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

Table 82. Multiply Implicit Accumulate Instruction Timings

Mnemonic	Rs Value (Early Termination)	Minimum Issue Latency	Minimum Result Latency	Minimum Resource Latency (Throughput)
MIA	Rs[31:15] = 0x0000 or Rs[31:15] = 0xFFFF	1	1	1
	Rs[31:27] = 0x0 or Rs[31:27] = 0xF	1	2	2
	all others	1	3	3
MIAxy	N/A	1	1	1
MIAPH	N/A	1	2	2

Table 83. Implicit Accumulator Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency	Minimum Resource Latency (Throughput)
MAR	2	2	2
MRA	1	(RdLo = 2; RdHi = 3)*	2

Note: If the next instruction needs to use the result of the MRA for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

3.9.4.5 Saturated Arithmetic Instructions

Table 84. Saturated Data Processing Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
QADD	1	2
QSUB	1	2
QDADD	1	2
QDSUB	1	2

3.9.4.6 Status Register Access Instructions

Table 85. Status Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRS	1	2
MSR	2 (6 if updating mode bits)	1

3.9.4.7 Load/Store Instructions

Table 86. Load and Store Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDR	1	3 for load data; 1 for writeback of base
LDRB	1	3 for load data; 1 for writeback of base
LDRBT	1	3 for load data; 1 for writeback of base
LDRD	1 (+1 if Rd is R12)	3 for Rd; 4 for Rd+1; 1 (+1 if Rd is R12) for write-back of base
LDRH	1	3 for load data; 1 for writeback of base
LDRSB	1	3 for load data; 1 for writeback of base

Table 86. Load and Store Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDRSH	1	3 for load data; 1 for writeback of base
LDRT	1	3 for load data; 1 for writeback of base
PLD	1	N/A
STR	1	1 for writeback of base
STRB	1	1 for writeback of base
STRBT	1	1 for writeback of base
STRD	2	2 for write-back of base
STRH	1	1 for writeback of base
STRT	1	1 for writeback of base

Table 87. Load and Store Multiple Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDM[1]	2 + numreg[2]	5-18 for load data (4 + numreg for last register in list; 3 + numreg for 2nd to last register in list; 2 + numreg for all other registers in list); 2+ numreg for write-back of base
STM	2 + numreg	2 + numreg for write-back of base

Notes:
1. See Table 79 on page 162 for LDM timings when R15 is in the register list.
2. numreg is the number of registers in the register list.

3.9.4.8 Semaphore Instructions

Table 88. Semaphore Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
SWP	5	5
SWPB	5	5

3.9.4.9 Coprocessor Instructions

Table 89. CP15 Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRC*	4	4
MCR	2	N/A

Note: MRC to R15 is unpredictable.

Table 90. CP14 Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRC	8	8
MRC to R15	9	9
MCR	8	N/A
LDC	11	N/A
STC	8	N/A

3.9.4.10 Miscellaneous Instruction Timing

Table 91. Exception-Generating Instruction Timings

Mnemonic	Minimum latency to first instruction of exception handler
SWI	6
BKPT	6
UNDEFINED	6

Table 92. Count Leading Zeros Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
CLZ	1	1

3.9.4.11 Thumb Instructions

In general, the timing of Thumb instructions are the same as their equivalent ARM instructions, except for the cases listed below.

- If the equivalent ARM instruction maps to one in Table 78 on page 162, the "Minimum Issue Latency with Branch Misprediction" goes from 5 to 6 cycles. This is due to the branch latency penalty. (See Table 76 on page 160.)
- If the equivalent ARM instruction maps to one in Table 79 on page 162, the "Minimum Issue Latency when the Branch is Taken" increases by 1 cycle. This is due to the branch latency penalty. (See Table 76 on page 160.)
- A Thumb BL instruction when H = 0 will have the same timing as an ARM data processing instruction.

The mapping of Thumb instructions to ARM instructions can be found in the *ARM* Architecture Reference Manual*.

3.10 Optimization Guide

3.10.1 Introduction

This document contains optimization techniques for achieving the highest performance from the IXP42X product line and IXC1100 control plane processors architecture. It is written for developers who are optimizing compilers or performance analysis tools for the devices based on these processors. It can also be used by application developers to obtain the best performance from their assembly language code. The optimizations presented in this section are based on the IXP42X product line and IXC1100 control plane processors, and hence can be applied to all products that are based on it.

The IXP42X product line and IXC1100 control plane processors' architecture includes a super-pipelined RISC architecture with an enhanced memory pipeline. The IXP42X product line and IXC1100 control plane processors instruction set is based on the ARM V5TE architecture; however, the IXP42X product line and IXC1100 control plane processors include new instructions. Code generated for the SA110, SA1100 and SA1110 will execute on the IXP42X product line and IXC1100 control plane processors, however to obtain the maximum performance of your application code, it should be optimized for the IXP42X product line and IXC1100 control plane processors using the techniques presented in this document.

3.10.1.1 About This Section

This guide assumes that you are familiar with the ARM instruction set and the C language. It consists of the following sections:

- "Introduction" on page 167 — Outlines the contents of this guide.
- "Processors' Pipeline" on page 168 — This section provides an overview of IXP42X product line and IXC1100 control plane processors pipeline behavior.
- "Basic Optimizations" on page 173 — This section outlines basic optimizations that can be applied to IXP42X product line and IXC1100 control plane processors.
- "Cache and Prefetch Optimizations" on page 180 — This section contains optimizations for efficient use of caches. Also included are optimizations that take advantage of the prefetch instruction of IXP42X product line and IXC1100 control plane processors.
- "Instruction Scheduling" on page 191 — This section shows how to optimally schedule code for IXP42X product line and IXC1100 control plane processors pipeline.
- "Optimizing C Libraries" on page 199 — This section contains information relating to optimizations for C library routines.
- "Optimizations for Size" on page 199 — This section contains optimizations that reduce the size of the generated code. Thumb optimizations are also included.

3.10.2 Processors' Pipeline

One of the biggest differences between the IXP42X product line and IXC1100 control plane processors and ARM processors is the pipeline. Many of the differences are summarized in Figure 29. This section provides a brief description of the structure and behavior of the IXP42X product line and IXC1100 control plane processors pipeline.

3.10.2.1 General Pipeline Characteristics

While the IXP42X product line and IXC1100 control plane processors pipeline are scalar and single issue, instructions may occupy all three pipelines at once. Out of order completion is possible. The following sections discuss general pipeline characteristics.

3.10.2.1.1 Number of Pipeline Stages

The IXP42X product line and IXC1100 control plane processors have a longer pipeline (seven stages versus five stages) which operates at a much higher frequency than its predecessors do. This allows for greater overall performance. The longer the IXP42X product line and IXC1100 control plane processors pipeline have several negative consequences, however:

- Larger branch misprediction penalty (four cycles in the IXP42X product line and IXC1100 control plane processors instead of one in ARM Architecture). This is mitigated by dynamic branch prediction.
- Larger load use delay (LUD) - LUDs arise from load-use dependencies. A load-use dependency gives rise to a LUD if the result of the load instruction cannot be made available by the pipeline in due time for the subsequent instruction. An optimizing compiler should find independent instructions to fill the slot following the load.
- Certain instructions incur a few extra cycles of delay on the IXP42X product line and IXC1100 control plane processors as compared to ARM processors (**LDM**, **STM**).
- Decode and register file lookups are spread out over two cycles in the IXP42X product line and IXC1100 control plane processors, instead of one cycle in predecessors.

3.10.2.1.2 Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Pipeline Organization

The IXP42X product line and IXC1100 control plane processors single-issue super-pipeline consist of a main execution pipeline, MAC pipeline, and a memory access pipeline. These are shown in Figure 29, with the main execution pipeline shaded.

Figure 29. Processors' RISC Super-Pipeline

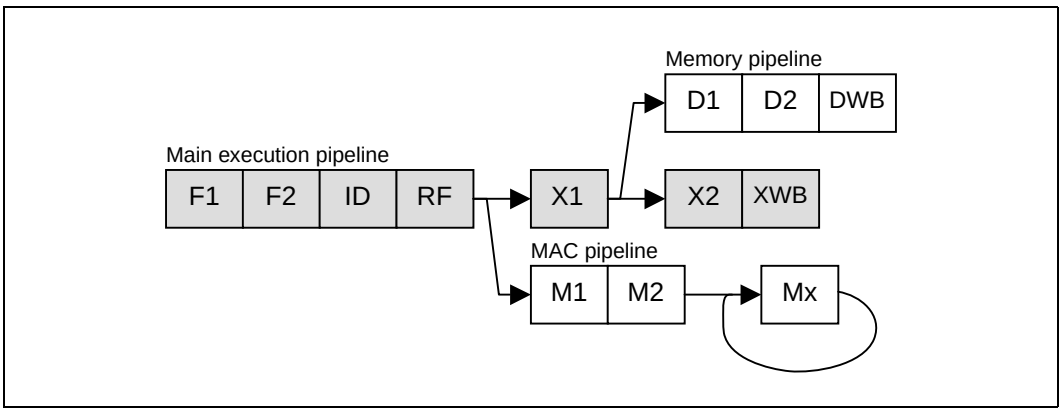


Table 93 gives a brief description of each pipe-stage.

Table 93. Pipelines and Pipe Stages

Pipe / Pipe State	Description	Covered In
Main Execution Pipeline	Handles data processing instructions	"Main Execution Pipeline" on page 171
IF1/IF2	Instruction Fetch	"
ID	Instruction Decode	"
RF	Register File / Operand Shifter	"
X1	ALU Execute	"
X2	State Execute	"
XWB	Write-back	"
Memory Pipeline	Handles load/store instructions	"Memory Pipeline" on page 172
D1/D2	Data Cache Access	"
DWB	Data cache writeback	"
MAC Pipeline	Handles all multiply instructions	"Multiply/Multiply Accumulate (MAC) Pipeline" on page 173
M1-M5	Multiplier stages	"
MWB (not shown)	MAC write-back - may occur during M2-M5	"

3.10.2.1.3 Out-of-Order Completion

Sequential consistency of instruction execution relates to two aspects: first, to the order in which the instructions are completed; and second, to the order in which memory is accessed due to load and store instructions. The IXP42X product line and IXC1100 control plane processors preserve a weak processor consistency because instructions may complete out of order, provided that no data dependencies exist.

While instructions are issued in-order, the main execution pipeline, memory, and MAC pipelines are not lock-stepped, and, therefore, have different execution times. This means that instructions may finish out of program order. Short ‘younger’ instructions may be finished earlier than long ‘older’ ones. (The term ‘to finish’ is used here to indicate that the operation has been completed and the result has been written back to the register file.)

3.10.2.1.4 Register Scoreboarding

In certain situations, the pipeline may need to be stalled because of register dependencies between instructions. A register dependency occurs when a previous MAC or load instruction is about to modify a register value that has not been returned to the register file and the current instruction needs access to the same register. Only the destination of MAC operations and memory loads are scoreboarded. The destinations of ALU instructions are not scoreboarded.

If no register dependencies exist, the pipeline will not be stalled. For example, if a load operation has missed the data cache, subsequent instructions that do not depend on the load may complete independently.

3.10.2.1.5 Use of Bypassing

The IXP42X product line and IXC1100 control plane processors pipeline make extensive use of bypassing to minimize data hazards. Bypassing allows results forwarding from multiple sources, eliminating the need to stall the pipeline.

3.10.2.2 Instruction Flow Through the Pipeline

The IXP42X product line and IXC1100 control plane processors’ pipeline issues a single instruction per clock cycle. Instruction execution begins at the F1 pipe stage and completes at the WB pipe stage.

Although a single instruction may be issued per clock cycle, all three pipelines (MAC, memory, and main execution) may be processing instructions simultaneously. If there are no data hazards, then each instruction may complete independently of the others.

Each pipe stage takes a single clock cycle or machine cycle to perform its subtask with the exception of the MAC unit.

3.10.2.2.1 ARM* V5TE Instruction Execution

Figure 29 on page 169 uses arrows to show the possible flow of instructions in the pipeline. Instruction execution flows from the F1 pipe stage to the RF pipe stage. The RF pipe stage may issue a single instruction to either the X1 pipe stage or the MAC unit (multiply instructions go to the MAC, while all others continue to X1). This means that M1 or X1 will be idle.

All load/store instructions are routed to the memory pipeline after the effective addresses have been calculated in X1.

The ARM V5TE bx (branch and exchange) instruction, which is used to branch between ARM and thumb code, causes the entire pipeline to be flushed (The bx instruction is not dynamically predicted by the BTB). If the processor is in Thumb mode, then the ID pipe stage dynamically expands each Thumb instruction into a normal ARM V5TE RISC instruction and execution resumes as usual.

3.10.2.2.2 Pipeline Stalls

The progress of an instruction can stall anywhere in the pipeline. Several pipe stages may stall for various reasons. It is important to understand when and how hazards occur in the IXP42X product line and IXC1100 control plane processors' pipeline. Performance degradation can be significant if care is not taken to minimize pipeline stalls.

3.10.2.3 Main Execution Pipeline

3.10.2.3.1 F1 / F2 (Instruction Fetch) Pipe Stages

The job of the instruction fetch stages F1 and F2 is to present the next instruction to be executed to the ID stage. Several important functional units reside within the F1 and F2 stages, including:

- Branch Target Buffer (BTB)
- Instruction Fetch Unit (IFU)

An understanding of the BTB (See "Branch Target Buffer" on page 58) and IFU are important for performance considerations. A summary of operation is provided here so that the reader may understand its role in the F1 pipe stage.

- Branch Target Buffer (BTB)
  The BTB predicts the outcome of branch type instructions. Once a branch type instruction reaches the X1 pipe stage, its target address is known. If this address is different from the address that the BTB predicted, the pipeline is flushed, execution starts at the new target address, and the branch's history is updated in the BTB.
- Instruction Fetch Unit (IFU)
  The IFU is responsible for delivering instructions to the *instruction decode* (ID) pipe stage. One instruction word is delivered each cycle (if possible) to the ID. The instruction could come from one of two sources: instruction cache or fetch buffers.

3.10.2.3.2 ID (Instruction Decode) Pipe Stage

The ID pipe stage accepts an instruction word from the IFU and sends register decode information to the RF pipe stage. The ID is able to accept a new instruction word from the IFU on every clock cycle in which there is no stall. The ID pipe stage is responsible for:

- General instruction decoding (extracting the op code, operand addresses, destination addresses and the offset).
- Detecting undefined instructions and generating an exception.
- Dynamic expansion of complex instructions into sequence of simple instructions. Complex instructions are defined as ones that take more than one clock cycle to issue, such as **LDM**, **STM**, and **SWP**.

3.10.2.3.3 RF (Register File / Shifter) Pipe Stage

The main function of the RF pipe stage is to read and write to the *register file unit, or RFU. It provides source data to:*

- EX for ALU operations
- MAC for multiply operations
- Data Cache for memory writes
- Coprocessor interface

The ID unit decodes the instruction and specifies which registers are accessed in the RFU. Based upon this information, the RFU determines if it needs to stall the pipeline due to a register dependency. A register dependency occurs when a previous instruction is about to modify a register value that has not been returned to the RFU and the current instruction needs to access that same register. If no dependencies exist, the RFU will select the appropriate data from the register file and pass it to the next pipe stage. When a register dependency does exist, the RFU will keep track of which register is unavailable and when the result is returned, the RFU will stop stalling the pipe.

The ARM architecture specifies that one of the operands for data processing instructions as the shifter operand, where a 32-bit shift can be performed before it is used as an input to the ALU. This shifter is located in the second half of the RF pipe stage.

3.10.2.3.4 X1 (Execute) Pipe Stages

The X1 pipe stage performs the following functions:

- ALU calculation - the ALU performs arithmetic and logic operations, as required for data processing instructions and load/store index calculations.
- Determine conditional instruction execution - The instruction's condition is compared to the CPSR prior to execution of each instruction. Any instruction with a false condition is cancelled, and will not cause any architectural state changes, including modifications of registers, memory, and PSR.
- Branch target determination - If a branch was incorrectly predicted by the BTB, the X1 pipe stage flushes all of the instructions in the previous pipe stages and sends the branch target address to the BTB, which will restart the pipeline

3.10.2.3.5 X2 (Execute 2) Pipe Stages

The X2 pipe stage contains the *program status registers* (PSRs). This pipe stage selects what is going to be written to the RFU in the WB cycle: PSRs (MRS instruction), ALU output, or other items.

3.10.2.3.6 WB (Write-Back)

When an instruction has reached the write-back stage, it is considered complete. Changes are written to the RFU.

3.10.2.4 Memory Pipeline

The memory pipeline consists of two stages, D1 and D2. The *data cache unit*, or DCU, consists of the data-cache array, mini-data cache, fill buffers, and write buffers. The memory pipeline handles load / store instructions.

3.10.2.4.1 D1 and D2 Pipe Stage

Operation begins in D1 after the X1 pipe stage has calculated the effective address for load/stores. The data cache and mini-data cache returns the destination data in the D2 pipe stage. Before data is returned in the D2 pipe stage, sign extension and byte alignment occurs for byte and half-word loads.

3.10.2.5 Multiply/Multiply Accumulate (MAC) Pipeline

The Multiply-Accumulate (MAC) unit executes the multiply and multiply-accumulate instructions supported by the IXP42X product line and IXC1100 control plane processors. The MAC implements the 40-bit IXP42X product line and IXC1100 control plane processors accumulator register acc0 and handles the instructions, which transfer its value to and from general-purpose ARM registers.

The following are important characteristics about the MAC:

- The MAC is not truly pipelined, as the processing of a single instruction may require use of the same data path resources for several cycles before a new instruction can be accepted. The type of instruction and source arguments determines the number of cycles required.
- No more than two instructions can occupy the MAC pipeline concurrently.
- When the MAC is processing an instruction, another instruction may not enter M1 unless the original instruction completes in the next cycle.
- The MAC unit can operate on 16-bit packed signed data. This reduces register pressure and memory traffic size. Two 16-bit data items can be loaded into a register with one LDR.
- The MAC can achieve throughput of one multiply per cycle when performing a 16-by-32-bit multiply.

3.10.2.5.1 Behavioral Description

The execution of the MAC unit starts at the beginning of the M1 pipe stage, where it receives two 32-bit source operands. Results are completed N cycles later (where N is dependent on the operand size) and returned to the register file. For more information on MAC instruction latencies, refer to “Instruction Latencies” on page 160.

An instruction that occupies the M1 or M2 pipe stages will also occupy the X1 and X2 pipe stage, respectively. Each cycle, a MAC operation progresses for M1 to M5. A MAC operation may complete anywhere from M2-M5. If a MAC operation enters M3-M5, it is considered committed because it will modify architectural state regardless of subsequent events.

3.10.3 Basic Optimizations

This section outlines optimizations specific to ARM architecture. These optimizations have been modified to suit the IXP42X product line and IXC1100 control plane processors where needed.

3.10.3.1 Conditional Instructions

The IXP42X product line and IXC1100 control plane processors’ architecture provides the ability to execute instructions conditionally. This feature combined with the ability of the IXP42X product line and IXC1100 control plane processors instructions to modify the condition codes makes possible a wide array of optimizations.

3.10.3.1.1 Optimizing Condition Checks

The IXP42X product line and IXC1100 control plane processors’ instructions can selectively modify the state of the condition codes. When generating code for if-else and loop conditions it is often beneficial to make use of this feature to set condition codes, thereby eliminating the need for a subsequent compare instruction.

Consider the C code segment:

```
if (a + b)
```

Code generated for the if condition without using an add instruction to set condition codes is:

```
;Assume r0 contains the value a, and r1 contains the value b
    add   r0,r0,r1
    cmp   r0, #0
```

However, code can be optimized as follows making use of add instruction to set condition codes:

```
;Assume r0 contains the value a, and r1 contains the value b
    adds  r0,r0,r1
```

The instructions that increment or decrement the loop counter can also be used to modify the condition codes. This eliminates the need for a subsequent compare instruction. A conditional branch instruction can then be used to exit or continue with the next loop iteration.

Consider the following C code segment:

```
for (i = 10; i != 0; i--)
{
    do something;
}
```

The optimized code generated for the above code segment would look like:

```
L6:
.
.
    subs r3, r3, #1
    bne  .L6
```

It is also beneficial to rewrite loops whenever possible so as to make the loop exit conditions check against the value 0. For example, the code generated for the code segment below will need a compare instruction to check for the loop exit condition.

```
for (i = 0; i < 10; i++)
{
    do something;
}
```

If the loop were rewritten as follows, the code generated avoids using the compare instruction to check for the loop exit condition.

```
for (i = 9; i >= 0; i--)
{
    do something;
}
```

3.10.3.1.2 Optimizing Branches

Branches decrease application performance by indirectly causing pipeline stalls. Branch prediction improves the performance by lessening the delay inherent in fetching a new instruction stream. The number of branches that can accurately be predicted is limited by the size of the branch target buffer. Since the total number of branches executed in a program is relatively large compared to the size of the branch target buffer; it is often beneficial to minimize the number of branches in a program. Consider the following C code segment.

```
int foo(int a)
{
    if (a > 10)
        return 0;
    else
        return 1;
}
```

The code generated for the if-else portion of this code segment using branches is:

```
    cmp   r0, #10
    ble   L1
    mov   r0, #0
    b     L2
L1:
    mov   r0, #1
L2:
```

The code generated above takes three cycles to execute the else part and four cycles for the if-part assuming best case conditions and no branch misprediction penalties. In the case of the IXP42X product line and IXC1100 control plane processors, a branch misprediction incurs a penalty of four cycles. If the branch is incorrectly predicted 50 percent of the time, and if we assume that both the if-part and the else-part are equally likely to be taken, on an average the code above takes 5.5 cycles to execute.

$$\left(\frac{50}{100} \times 4 + \frac{3+4}{2}\right) = 5.5 \qquad cycles$$

If we were to use the IXP42X product line and IXC1100 control plane processors to execute instructions conditionally, the code generated for the above if-else statement is:

```
    cmp   r0, #10
    movgt r0, #0
    movle r0, #1
```

The above code segment would not incur any branch misprediction penalties and would take three cycles to execute assuming best case conditions. As can be seen, using conditional instructions speeds up execution significantly. However, the use of conditional instructions should be carefully considered to ensure that it does improve performance. To decide when to use conditional instructions over branches consider the following hypothetical code segment:

```
if (cond)

    if_stmt

else

    else_stmt
```

Assume that we have the following data:

- $N1_B$..... Number of cycles to execute the if_stmt assuming the use of branch instructions
- $N2_B$.....Number of cycles to execute the else_stmt assuming the use of branch instructions
- P1.....Percentage of times the if_stmt is likely to be executed
- P2..... Percentage of times we are likely to incur a branch misprediction penalty
- $N1_C$... Number of cycles to execute the if-else portion using conditional instructions assuming the if-condition to be true
- $N2_C$... Number of cycles to execute the if-else portion using conditional instructions assuming the if-condition to be false

Once we have the above data, use conditional instructions when:

$$\left(N1_C \times \frac{P1}{100}\right) + \left(N2_C \times \frac{100 - P1}{100}\right) \leq \left(N1_B \times \frac{P1}{100}\right) + \left(N2_B \times \frac{100 - P1}{100}\right) + \left(\frac{P2}{100} \times 4\right)$$

The following example illustrates a situation in which we are better off using branches over conditional instructions. Consider the code sample shown below:

```
    cmp   r0, #0
    bne   L1
    add   r0, r0, #1
    add   r1, r1, #1
    add   r2, r2, #1
    add   r3, r3, #1
    add   r4, r4, #1
    b     L2
L1:
    sub   r0, r0, #1
    sub   r1, r1, #1
    sub   r2, r2, #1
    sub   r3, r3, #1
    sub   r4, r4, #1
L2:
```

In the above code sample, the cmp instruction takes 1 cycle to execute, the if-part takes 7 cycles to execute and the else-part takes 6 cycles to execute. If we were to change the code above so as to eliminate the branch instructions by making use of conditional instructions, the if-else part would always take 10 cycles to complete.

If we make the assumptions that both paths are equally likely to be taken and that branches are mis-predicted 50% of the time, the costs of using conditional execution Vs using branches can be computed as follows:

Cost of using conditional instructions:

$$1 + \left(\frac{50}{100} \times 10\right) + \left(\frac{50}{100} \times 10\right) = 11 \quad cycles$$

Cost of using branches:

$$1 + \left(\frac{50}{100} \times 7\right) + \left(\frac{50}{100} \times 6\right) + \left(\frac{50}{100} \times 4\right) = 9.5 \quad cycles$$

As can be seen, we get better performance by using branch instructions in the above scenario.

3.10.3.1.3 Optimizing Complex Expressions

Conditional instructions should also be used to improve the code generated for complex expressions such as the C shortcut evaluation feature. Consider the following C code segment:

```
int foo(int a, int b)
{
    if (a != 0 && b != 0)
        return 0;
    else
        return 1;
}
```

The optimized code for the if condition is:

```
    cmp   r0, #0
    cmpne r1, #0
```

Similarly, the code generated for the following C segment

```
int foo(int a, int b)
{
    if (a != 0 || b != 0)
        return 0;
    else
        return 1;
}
```

is:

```
    cmp   r0, #0
    cmpeq r1, #0
```

The use of conditional instructions in the above fashion improves performance by minimizing the number of branches, thereby minimizing the penalties caused by branch incorrect predictions. This approach also reduces the utilization of branch prediction resources.

3.10.3.2 Bit Field Manipulation

The IXP42X product line and IXC1100 control plane processors shift and logical operations provide a useful way of manipulating bit fields. Bit field operations can be optimized as follows:

```
;Set the bit number specified by r1 in register r0
    mov   r2, #1
    orr   r0, r0, r2, asl r1
;Clear the bit number specified by r1 in register r0
    mov   r2, #1
    bic   r0, r0, r2, asl r1
;Extract the bit-value of the bit number specified by r1 of the
;value in r0 storing the value in r0
    mov   r1, r0, asr r1
    and   r0, r1, #1
;Extract the higher order 8 bits of the value in r0 storing
;the result in r1
    mov   r1, r0, lsr #24
```

3.10.3.3 Optimizing the Use of Immediate Values

The IXP42X product line and IXC1100 control plane processors' **MOV** or **MVN** instruction should be used when loading an immediate (constant) value into a register. Please refer to the *ARM* Architecture Reference Manual* for the set of immediate values that can be used in a **MOV** or **MVN** instruction. It is also possible to generate a whole set of constant values using a combination of **MOV**, **MVN**, **ORR**, **BIC**, and **ADD** instructions. The **LDR** instruction has the potential of incurring a cache miss in addition to polluting the data and instruction caches. The code samples below illustrate cases when a combination of the above instructions can be used to set a register to a constant value:

```
;Set the value of r0 to 127
    mov   r0, #127
;Set the value of r0 to 0xfffffefb.
    mvn   r0, #260
;Set the value of r0 to 257
    mov   r0, #1
    orr   r0, r0, #256
;Set the value of r0 to 0x51f
    mov   r0, #0x1f
    orr   r0, r0, #0x500
;Set the value of r0 to 0xf100ffff
    mvn   r0, #0xff, 16
    bic   r0, r0, #0xe, 8
; Set the value of r0 to 0x12341234
    mov   r0, #0x8d, 30
    orr   r0, r0, #0x1, 20
    add   r0, r0, r0, LSL #16 ; shifter delay of 1 cycle
```

Note that it is possible to load any 32-bit value into a register using a sequence of four instructions.

3.10.3.4 Optimizing Integer Multiply and Divide

Multiplication by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Multiplication of R0 by 2^n
    mov   r0, r0, LSL #n
;Multiplication of R0 by 2^n+1
    add   r0, r0, r0, LSL #n
```

Multiplication by an integer constant that can be expressed as $(2^n + 1) \cdot (2^m)$ can similarly be optimized as:

```
;Multiplication of r0 by an integer constant that can be
;expressed as (2^n+1)*(2^m)
    add   r0, r0, r0, LSL #n
    mov   r0, r0, LSL #m
```

Please note that the above optimization should only be used in cases where the multiply operation cannot be advanced far enough to prevent pipeline stalls.

Dividing an unsigned integer by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Dividing r0 containing an unsigned value by an integer constant
;that can be represented as 2^n
    mov   r0, r0, LSR #n
```

Dividing a signed integer by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Dividing r0 containing a signed value by an integer constant
;that can be represented as 2^n
    mov   r1, r0, ASR #31
    add   r0, r0, r1, LSR #(32 - n)
    mov   r0, r0, ASR #n
```

The add instruction would stall for 1 cycle. The stall can be prevented by filling in another instruction before add.

3.10.3.5 Effective Use of Addressing Modes

The IXP42X product line and IXC1100 control plane processors provide a variety of addressing modes that make indexing an array of objects highly efficient. For a detailed description of these addressing modes please refer to the *ARM* Architecture Reference Manual*. The following code samples illustrate how various kinds of array operations can be optimized to make use of these addressing modes:

```
;Set the contents of the word pointed to by r0 to the value
;contained in r1 and make r0 point to the next word
    str   r1,[r0], #4
;Increment the contents of r0 to make it point to the next word
;and set the contents of the word pointed to the value contained
;in r1
    str   r1, [r0, #4]!
;Set the contents of the word pointed to by r0 to the value
;contained in r1 and make r0 point to the previous word
    str   r1,[r0], #-4
;Decrement the contents of r0 to make it point to the previous
;word and set the contents of the word pointed to the value
;contained in r1
    str   r1,[r0, #-4]!
```

3.10.4 Cache and Prefetch Optimizations

This section considers how to use the various cache memories in all their modes and then examines when and how to use prefetch to improve execution efficiencies.

3.10.4.1 Instruction Cache

The IXP42X product line and IXC1100 control plane processors have separate instruction and data caches. Only fetched instructions are held in the instruction cache even though both data and instructions may reside within the same memory space with each other. Functionally, the instruction cache is either enabled or disabled. There is no performance benefit in not using the instruction cache. The exception is that code, which locks code into the instruction cache, must itself execute from non-cached memory.

3.10.4.1.1 Cache Miss Cost

The IXP42X product line and IXC1100 control plane processors' performance is highly dependent on reducing the cache miss rate.

Note that this cycle penalty becomes significant when the Intel XScale processor is running much faster than external memory. Executing non-cached instructions severely curtails the processor's performance in this case and it is very important to do everything possible to minimize cache misses.

For the IXP42X product line and IXC1100 control plane processors, care must be taken to optimize code to have a maximum cache hit when accesses have been requested to the Expansion Bus Interface or the PCI Bus Controller. These design recommendations are due to the latency that may be associated with accessing the PCI Bus Controller and Expansion Bus Controller. Retries will be issued to the Intel XScale processor until the requested transaction is completed.

3.10.4.1.2 Round Robin Replacement Cache Policy

Both the data and the instruction caches use a round robin replacement policy to evict a cache line. The simple consequence of this is that at sometime every line will be evicted, assuming a non-trivial program. The less obvious consequence is that predicting when and over which cache lines evictions take place is very difficult to predict. This information must be gained by experimentation using performance profiling.

3.10.4.1.3 Code Placement to Reduce Cache Misses

Code placement can greatly affect cache misses. One way to view the cache is to think of it as 32 sets of 32 bytes, which span an address range of 1,024 bytes. When running, the code maps into 32 blocks modular 1,024 of cache space. Any sets, which are overused, will thrash the cache. The ideal situation is for the software tools to distribute the code on a temporal evenness over this space.

This is very difficult if not impossible for a compiler to do. Most of the input needed to best estimate how to distribute the code will come from profiling followed by compiler based two pass optimizations.

3.10.4.1.4 Locking Code into the Instruction Cache

One very important instruction cache feature is the ability to lock code into the instruction cache. Once locked into the instruction cache, the code is always available for fast execution. Another reason for locking critical code into cache is that with the round robin replacement policy, eventually the code will be evicted, even if it is a very frequently executed function. Key code components to consider for locking are:

- Interrupt handlers
- Real time clock handlers
- OS critical code
- Time critical application code

The disadvantage to locking code into the cache is that it reduces the cache size for the rest of the program. How much code to lock is very application dependent and requires experimentation to optimize.

Code placed into the instruction cache should be aligned on a 1,024-byte boundary and placed sequentially together as tightly as possible so as not to waste precious memory space. Making the code sequential also insures even distribution across all cache ways. Though it is possible to choose randomly located functions for cache locking, this approach runs the risk of landing multiple cache ways in one set and few or none in another set. This distribution unevenness can lead to excessive thrashing of the Data and Mini Caches.

3.10.4.2 Data and Mini Cache

The IXP42X product line and IXC1100 control plane processors allow the user to define memory regions whose cache policies can be set by the user (see “Cacheability” on page 63). Supported policies and configurations are:

- Non Cacheable with no coalescing of memory writes.
- Non Cacheable with coalescing of memory writes.
- Mini-Data cache with write coalescing, read allocate, and write-back caching.
- Mini-Data cache with write coalescing, read allocate, and write-through caching.
- Mini-Data cache with write coalescing, read-write allocate, and write-back caching.
- Data cache with write coalescing, read allocate, and write-back caching.
- Data cache with write coalescing, read allocate, and write-through caching.
- Data cache with write coalescing, read-write allocate, and write-back caching.

To support allocating variables to these various memory regions, the tool chain (compiler, assembler, linker and debugger), must implement named sections.

The performance of your application code depends on what cache policy you are using for data objects. A description of when to use a particular policy is described below.

The IXP42X product line and IXC1100 control plane processors allow dynamic modification of the cache policies at run time, however, the operation is requires considerable processing time and therefore should not be used by applications.

If the application is running under an OS, then the OS may restrict you from using certain cache policies.

3.10.4.2.1 Non-Cacheable Regions

It is recommended that non-cache memory (X=0, C=0, and B=0) be used only if necessary as is often necessary for I/O devices. Accessing non-cacheable memory is likely to cause the processor to stall frequently due to the long latency of memory reads.

3.10.4.2.2 Write-Through and Write-Back Cached Memory Regions

Write through memory regions generate more data traffic on the bus. Therefore is not recommended that the write-through policy be used. The write back policy must be used whenever possible.

However, in a multiprocessor environment it will be necessary to use a write through policy if data is shared across multiple processors. In such a situation all shared memory regions should use write through policy. Memory regions that are private to a particular processor should use the write back policy.

3.10.4.2.3 Read Allocate and Read-Write Allocate Memory Regions

Most of the regular data and the stack for your application should be allocated to a read-write allocate region. It is expected that you will be writing and reading from them often.

Data that is write only (or data that is written to and subsequently not used for a long time) should be placed in a read allocate region. Under the read-allocate policy if a cache write miss occurs a new cache line will not be allocated, and hence will not evict critical data from the Data cache.

3.10.4.2.4 Creating On-Chip RAM

Part of the Data cache can be converted into fast on chip RAM. Access to objects in the on-chip RAM will not incur cache miss penalties, thereby reducing the number of processor stalls. Application performance can be improved by converting a part of the cache into on chip RAM and allocating frequently allocated variables to it. Due to the IXP42X product line and IXC1100 control plane processors' round-robin replacement policy, all data will eventually be evicted. Therefore to prevent critical or frequently used data from being evicted it should be allocated to on-chip RAM.

The following variables are good candidates for allocating to the on-chip RAM:

- Frequently used global data used for storing context for context switching.
- Global variables that are accessed in time critical functions such as interrupt service routines.

The on-chip RAM is created by locking a memory region into the Data cache (see "Reconfiguring the Data Cache as Data RAM" on page 68 for more details).

When creating the on-chip RAM, care must be taken to ensure that all sets in the on-chip RAM area of the Data cache have approximately the same number of ways locked, otherwise some sets will have more ways locked than the others. This uneven allocation will increase the level of thrashing in some sets and leave other sets under utilized.

For example, consider three arrays arr1, arr2 and arr3 of size 64 bytes each that are being allocated to the on-chip RAM and assume that the address of arr1 is 0, address of arr2 is 1024, and the address of arr3 is 2048. All three arrays will be within the same sets, i.e. set0 and set1, as a result three ways in both sets set0 and set1, will be locked, leaving 29 ways for use by other variables.

This can be overcome by allocating on-chip RAM data in sequential order. In the above example allocating arr2 to address 64 and arr3 to address 128, allows the three arrays to use only 1 way in sets 0 through 5.

3.10.4.2.5 Mini-Data Cache

The mini-data cache is best used for data structures, which have short temporal lives, and/or cover vast amounts of data space. Addressing these types of data spaces from the Data cache would corrupt much if not all of the Data cache by evicting valuable data. Eviction of valuable data will reduce performance. Placing this data instead in Mini-data cache memory region would prevent Data cache corruption while providing the benefits of cached accesses.

A prime example of using the mini-data cache would be for caching the procedure call stack. The stack can be allocated to the mini-data cache so that it's use does not trash the main dcache. This would keep local variables from global data.

Following are examples of data that could be assigned to mini-dcache:

- The stack space of a frequently occurring interrupt, the stack is used only during the duration of the interrupt, which is usually very small.
- Video buffers, these are usual large and can occupy the whole cache.

Over use of the Mini-Data cache will thrash the cache. This is easy to do because the Mini-Data cache only has two ways per set. For example, a loop which uses a simple statement such as:

```
for (i=0; I< IMAX; i++)
{
    A[i] = B[i] + C[i];
}
```

Where A, B, and C reside in a mini-data cache memory region and each is array is aligned on a 1-K boundary will quickly thrash the cache.

3.10.4.2.6 Data Alignment

Cache lines begin on 32-byte address boundaries. To maximize cache line use and minimize cache pollution, data structures should be aligned on 32-byte boundaries and sized to multiple cache line sizes. Aligning data structures on cache address boundaries simplifies later addition of prefetch instructions to optimize performance.

Not aligning data on cache lines has the disadvantage of moving the prefetch address correspondingly to the misalignment. Consider the following example:

```
struct {
    long ia;
    long ib;
    long ic;
    long id;
} tdata[IMAX];

for (i=0, i<IMAX; i++)
{
    PREFETCH(tdata[i+1]);
    tdata[i].ia = tdata[i].ib + tdata[i].ic _tdata[i].id];
    ....
    tdata[i].id = 0;
}
```

In this case if tdata[] is not aligned to a cache line, then the prefetch using the address of tdata[i+1].ia may not include element id. If the array was aligned on a cache line + 12 bytes, then the prefetch would halve to be placed on &tdata[i+1].id.

If the structure is not sized to a multiple of the cache line size, then the prefetch address must be advanced appropriately and will require extra prefetch instructions. Consider the following example:

```
struct {
    long ia;
    long ib;
    long ic;
    long id;
    long ie;
} tdata[IMAX];

ADDRESS preadd = tdata

for (i=0, i<IMAX; i++)
{
    PREFETCH(predata+=16);
    tdata[I].ia = tdata[I].ib + tdata[I].ic _tdata[I].id] +
    tdata[I].ie;
    ....
    tdata[I].ie = 0;
}
```

In this case, the prefetch address was advanced by size of half a cache line and every other prefetch instruction is ignored. Further, an additional register is required to track the next prefetch address.

Generally, not aligning and sizing data will add extra computational overhead.

Additional prefetch considerations are discussed in greater detail in following sections.

3.10.4.2.7 Literal Pools

The IXP42X product line and IXC1100 control plane processors do not have a single instruction that can move all literals (a constant or address) to a register. One technique to load registers with literals in the IXP42X product line and IXC1100 control plane processors is by loading the literal from a memory location that has been initialized with the constant or address. These blocks of constants are referred to as literal pools. See “Basic Optimizations” on page 173 for more information on how to do this. It is advantageous to place all the literals together in a pool of memory known a literal pool. These data blocks are located in the text or code address space so that they can be loaded using PC relative addressing. However, references to the literal pool area load the data into the data cache instead of the instruction cache. Therefore it is possible that the literal may be present in both the data and instruction caches, resulting in waste of space.

For maximum efficiency, the compiler should align all literal pools on cache boundaries and size each pool to a multiple of 32 bytes (the size of a cache line). One additional optimization would be group highly used literal pool references into the same cache line. The advantage is that once one of the literals has been loaded, the other seven will be available immediately from the data cache.

3.10.4.3 Cache Considerations

3.10.4.3.1 Cache Conflicts, Pollution, and Pressure

Cache pollution occurs when unused data is loaded in the cache and cache pressure occurs when data that is not temporal to the current process is loaded into the cache.

3.10.4.3.2 Memory Page Thrashing

Memory page thrashing occurs because of the nature of SDRAM. SDRAMs are typically divided into four banks. Each bank can have one selected page where a page address size for current memory components is often defined as 4 k. Memory lookup time or latency time for a selected page address is currently two to three bus clocks. Thrashing occurs when subsequent memory accesses within the same memory bank access different pages. The memory page change adds three to four bus clock cycles to memory latency. This added delay extends the prefetch distance correspondingly making it more difficult to hide memory access latencies. This type of thrashing can be resolved by placing the conflicting data structures into different memory banks or by paralleling the data structures such that the data resides within the same memory page. It is also extremely important to insure that instruction and data sections are in different memory banks, or they will continually trash the memory page selection.

3.10.4.4 Prefetch Considerations

The IXP42X product line and IXC1100 control plane processors have a true prefetch load instruction (PLD). The purpose of this instruction is to preload data into the data and mini-data caches. Data prefetching allows hiding of memory transfer latency while the processor continues to execute instructions. The prefetch is important to compiler and assembly code because judicious use of the prefetch instruction can enormously improve throughput performance of the IXP42X product line and IXC1100 control plane processors. Data prefetch can be applied not only to loops but also to any data references within a block of code. Prefetch also applies to data writing when the memory type is enabled as write-allocate.

The IXP42X product line and IXC1100 control plane processors' prefetch load instruction is a true prefetch instruction because the load destination is the data or mini-data cache and not a register. Compilers for processors which have data caches, but do not support prefetch, sometimes use a load instruction to preload the data cache. This technique has the disadvantages of using a register to load data and requiring additional registers for subsequent preloads and thus increasing register pressure. By contrast, the prefetch can be used to reduce register pressure instead of increasing it.

The prefetch load is a hint instruction and does not guarantee that the data will be loaded. Whenever the load would cause a fault or a table walk, then the processor will ignore the prefetch instruction, the fault or table walk, and continue processing the next instruction. This is particularly advantageous in the case where a linked list or recursive data structure is terminated by a NULL pointer. Prefetching the NULL pointer will not fault program flow.

3.10.4.4.1 Prefetch Loop Limitations

It is not always advantages to add prefetch to a loop. Loop characteristics that limit the use value of prefetch are discussed below.

3.10.4.4.2 Compute versus Data Bus Bound

At the extreme, a loop, which is data bus bound, will not benefit from prefetch because all the system resources to transfer data are quickly allocated and there are no instructions that can profitably be executed. On the other end of the scale, compute bound loops allow complete hiding of all data transfer latencies.

3.10.4.4.3 Low Number of Iterations

Loops with very low iteration counts may have the advantages of prefetch completely mitigated. A loop with a small fixed number of iterations may be faster if the loop is completely unrolled rather than trying to schedule prefetch instructions.

3.10.4.4.4 Bandwidth Limitations

Overuse of prefetches can usurp resources and degrade performance. This happens because once the bus traffic requests exceed the system resource capacity, the processor stalls. The IXP42X product line and IXC1100 control plane processors data transfer resources are:

- Four fill buffers
- Four pending buffers
- Eight half-cache line write buffer

SDRAM resources are typically:

- Four memory banks
- One page buffer per bank referencing a 4K address range
- Four transfer request buffers

Consider how these resources work together. A fill buffer is allocated for each cache read miss. A fill buffer is also allocated each cache write miss if the memory space is write allocate along with a pending buffer. A subsequent read to the same cache line does not require a new fill buffer, but does require a pending buffer and a subsequent write will also require a new pending buffer. A fill buffer is also allocated for each read to a non-cached memory and a write buffer is needed for each memory write to non-cached memory that is non-coalescing. Consequently, a **STM** instruction listing eight registers and referencing non-cached memory will use eight write buffers assuming they don't coalesce and two write buffers if they do coalesce. A cache eviction requires a write buffer for each dirty bit set in the cache line. The prefetch instruction requires a fill buffer for each cache line and 0, 1, or 2 write buffers for an eviction.

When adding prefetch instructions, caution must be asserted to insure that the combination of prefetch and instruction bus requests do not exceed the system resource capacity described above or performance will be degraded instead of improved. The important points are to spread prefetch operations over calculations so as to allow bus traffic to free flow and to minimize the number of necessary prefetches.

3.10.4.4.5 Cache Memory Considerations

Stride, the way data structures are walked through, can affect the temporal quality of the data and reduce or increase cache conflicts. The IXP42X product line and IXC1100 control plane processors data cache and mini-data caches each have 32 sets of 32 bytes. This means that each cache line in a set is on a modular 1-K-address boundary. The caution is to choose data structure sizes and stride requirements that do not overwhelm a given set causing conflicts and increased register pressure. Register pressure can be increased because additional registers are required to track prefetch addresses. The effects can be affected by rearranging data structure components to use more parallel access to search and compare elements. Similarly rearranging sections of data structures so that sections often written fit in the same half cache line, 16 bytes for the IXP42X product line and IXC1100 control plane processors, can reduce cache eviction write-backs. On a global scale, techniques such as array merging can enhance the spatial locality of the data.

As an example of array merging, consider the following code:

```
int a_array[NMAX];
int b_array[NMAX];
int ix;

for (i=0; i<NMAX]; i++)
{
    ix = b[i];
    if (a[i] != 0)
        ix = a[i];
    do_other calculations;
}
```

In the above code, data is read from both arrays a and b, but a and b are not spatially close. Array merging can place a and b specially close.

```
struct {
    int a;
    int b;
} c_arrays;

int ix;

for (i=0; i<NMAX]; i++)
{
    ix = c[i].b;
    if (c[i].a != 0)
        ix = c[i].a;
    do_other_calculations;
}
```

As an example of rearranging often written to sections in a structure, consider the code sample:

```
struct employee {
    struct employee *prev;
    struct employee *next;
    float Year2DatePay;
    float Year2DateTax;
    int ssno;
    int empid;
    float Year2Date401KDed;
    float Year2DateOtherDed;
};
```

In the data structure shown above, the fields Year2DatePay, Year2DateTax, Year2Date401KDed, and Year2DateOtherDed are likely to change with each pay check. The remaining fields however change very rarely. If the fields are laid out as shown above, assuming that the structure is aligned on a 32-byte boundary, modifications to the Year2Date fields is likely to use two write buffers when the data is written out to memory. However, we can restrict the number of write buffers that are commonly used to 1 by rearranging the fields in the above data structure as shown below:

```
struct employee {
    struct employee *prev;
    struct employee *next;
    int ssno;
    int empid;
    float Year2DatePay;
    float Year2DateTax;
    float Year2Date401KDed;
    float Year2DateOtherDed;
};
```

3.10.4.4.6 Cache Blocking

Cache blocking techniques, such as strip-mining, are used to improve temporal locality of the data. Given a large data set that can be reused across multiple passes of a loop, data blocking divides the data into smaller chunks which can be loaded into the cache during the first loop and then be available for processing on subsequence loops thus minimizing cache misses and reducing bus traffic.

As an example of cache blocking consider the following code:

```
for(i=0; i<10000; i++)
    for(j=0; j<10000; j++)
        for(k=0; k<10000; k++)
            C[j][k] += A[i][k] * B[j][i];
```

The variable A[i][k] is completely reused. However, accessing C[j][k] in the j and k loops can displace A[i][j] from the cache. Using blocking the code becomes:

```
for(i=0; i<10000; i++)
    for(j1=0; j<100; j++)
        for(k1=0; k<100; k++)
            for(j2=0; j<100; j++)
                for(k2=0; k<100; k++)
                {
                    j = j1 * 100 + j2;
                    k = k1 * 100 + k2;
                    C[j][k] += A[i][k] * B[j][i];
                }
```

3.10.4.4.7 Prefetch Unrolling

When iterating through a loop, data transfer latency can be hidden by prefetching ahead one or more iterations. The solution incurs an unwanted side affect that the final interactions of a loop loads useless data into the cache, polluting the cache, increasing bus traffic and possibly evicting valuable temporal data. This problem can be resolved by prefetch unrolling. For example consider:

```
for(i=0; i<NMAX; i++)
{
    prefetch(data[i+2]);
    sum += data[i];
}
```

Interactions i-1 and i, will prefetch superfluous data. The problem can be avoid by unrolling the end of the loop.

```
for(i=0; i<NMAX-2; i++)
{
    prefetch(data[i+2]);
    sum += data[i];
}
sum += data[NMAX-2];
sum += data[NMAX-1];
```

Unfortunately, prefetch loop unrolling does not work on loops with indeterminate iterations.

3.10.4.4.8 Pointer Prefetch

Not all looping constructs contain induction variables. However, prefetching techniques can still be applied. Consider the following linked list traversal example:

```
while(p) {
    do_something(p->data);
    p = p->next;
}
```

The pointer variable *p* becomes a pseudo induction variable and the data pointed to by *p->next* can be pre-fetched to reduce data transfer latency for the next iteration of the loop. Linked lists should be converted to arrays as much as possible.

```
while(p) {
    prefetch(p->next);
    do_something(p->data);
    p = p->next;
}
```

Recursive data structure traversal is another construct where prefetching can be applied. This is similar to linked list traversal. Consider the following pre-order traversal of a binary tree:

```
preorder(treeNode *t) {
    if(t) {
        process(t->data);
        preorder(t->left);
        preorder(t->right);
    }
}
```

The pointer variable *t* becomes the pseudo induction variable in a recursive loop. The data structures pointed to by the values *t->left* and *t->right* can be pre-fetched for the next iteration of the loop.

```
preorder(treeNode *t) {
    if(t) {
        prefetch(t->right);
        prefetch(t->left);
        process(t->data);
        preorder(t->left);
        preorder(t->right);
    }
}
```

Note the order reversal of the prefetches in relationship to the usage. If there is a cache conflict and data is evicted from the cache then only the data from the first prefetch is lost.

3.10.4.4.9 Loop Interchange

As mentioned earlier, the sequence in which data is accessed affects cache thrashing. Usually, it is best to access data in a contiguous spatially address range. However, arrays of data may have been laid out such that indexed elements are not physically next to each other. Consider the following C code which places array elements in row major order.

```
for(j=0; j<NMAX; j++)
    for(i=0; i<NMAX; i++)
    {
        prefetch(A[i+1][j]);
        sum += A[i][j];
    }
```

In the above example, A[i][j] and A[i+1][j] are not sequentially next to each other. This situation causes an increase in bus traffic when prefetching loop data. In some cases where the loop mathematics are unaffected, the problem can be resolved by induction variable interchange. The above examples becomes:

```
for(i=0; i<NMAX; i++)
    for(j=0; j<NMAX; j++)
    {
        prefetch(A[i][j+1]);
        sum += A[i][j];
    }
```

3.10.4.4.10 Loop Fusion

Loop fusion is a process of combining multiple loops, which reuse the same data, in to one loop. The advantage of this is that the reused data is immediately accessible from the data cache. Consider the following example:

```
for(i=0; i<NMAX; i++)
{
    prefetch(A[i+1], c[i+1], c[i+1]);
    A[i] = b[i] + c[i];
}
for(i=0; i<NMAX; i++)
{
    prefetch(D[i+1], c[i+1], A[i+1]);
    D[i] = A[i] + c[i];
}
```

The second loop reuses the data elements A[i] and c[i]. Fusing the loops together produces:

```
for(i=0; i<NMAX; i++)
{
    prefetch(D[i+1], A[i+1], c[i+1], b[i+1]);
    ai = b[i] + c[i];
    A[i] = ai;
    D[i] = ai + c[i];
}
```

3.10.4.4.11 Prefetch to Reduce Register Pressure

Pre-fetch can be used to reduce register pressure. When data is needed for an operation, then the load is scheduled far enough in advance to hide the load latency. However, the load ties up the receiving register until the data can be used. For example:

```
    ldr   r2, [r0]
; Process code { not yet cached latency > 60 core clocks }
    add   r1, r1, r2
```

In the above case, r2 is unavailable for processing until the add statement. Prefetching the data load frees the register for use. The example code becomes:

```
    pld   [r0] ;prefetch the data keeping r2 available for use
; Process code
    ldr   r2, [r0]
; Process code { ldr result latency is 3 core clocks }
    add   r1, r1, r2
```

With the added prefetch, register r2 can be used for other operations until almost just before it is needed.

3.10.5 Instruction Scheduling

This section discusses instruction scheduling optimizations. Instruction scheduling refers to the rearrangement of a sequence of instructions for the purpose of minimizing pipeline stalls. Reducing the number of pipeline stalls improves application performance. While making this rearrangement, care should be taken to ensure that the rearranged sequence of instructions has the same effect as the original sequence of instructions.

3.10.5.1 Scheduling Loads

On the IXP42X product line and IXC1100 control plane processors, an **LDR** instruction has a result latency of three cycles assuming the data being loaded is in the data cache. If the instruction after the **LDR** needs to use the result of the load, then it would stall for 2 cycles. If possible, the instructions surrounding the **LDR** instruction should be rearranged.

to avoid this stall. Consider the following example:

```
    add   r1, r2, r3

    ldr   r0, [r5]

    add   r6, r0, r1

    sub   r8, r2, r3

    mul   r9, r2, r3
```

In the code shown above, the **ADD** instruction following the **LDR** would stall for two cycles because it uses the result of the load. The code can be rearranged as follows to prevent the stalls:

```
ldr   r0, [r5]

add   r1, r2, r3

sub   r8, r2, r3

add   r6, r0, r1

mul   r9, r2, r3
```

Note that this rearrangement may not be always possible. Consider the following example:

```
cmp   r1, #0

addne r4, r5, #4

subeq r4, r5, #4

ldr   r0, [r4]

cmp   r0, #10
```

In the example above, the **LDR** instruction cannot be moved before the **ADDNE** or the **SUBEQ** instructions because the **LDR** instruction depends on the result of these instructions. Rewrite the above code to make it run faster at the expense of increasing code size:

```
cmp   r1, #0

ldrne r0, [r5, #4]

ldreq r0, [r5, #-4]

addne r4, r5, #4

subeq r4, r5, #4

cmp   r0, #10
```

The optimized code takes six cycles to execute compared to the seven cycles taken by the unoptimized version.

The result latency for an **LDR** instruction is significantly higher if the data being loaded is not in the data cache. To minimize the number of pipeline stalls in such a situation the **LDR** instruction should be moved as far away as possible from the instruction that uses result of the load. Note that this may at times cause certain register values to be spilled to memory due to the increase in register pressure. In such cases, use a preload instruction or a preload hint to ensure that the data access in the **LDR** instruction hits the cache when it executes. A preload hint should be used in cases where we cannot be sure whether the load instruction would be executed. A preload instruction should be used in cases where we can be sure that the load instruction would be executed. Consider the following code sample:

```
; all other registers are in use
    sub   r1, r6, r7
    mul   r3,r6, r2
    mov   r2, r2, LSL #2
    orr   r9, r9, #0xf
    add   r0,r4, r5
    ldr   r6, [r0]
    add   r8, r6, r8
    add   r8, r8, #4
    orr   r8,r8, #0xf
; The value in register r6 is not used after this
```

In the code sample above, the **ADD** and the **LDR** instruction can be moved before the **MOV** instruction. Note that this would prevent pipeline stalls if the load hits the data cache. However, if the load is likely to miss the data cache, move the **LDR** instruction so that it executes as early as possible - before the **SUB** instruction. However, moving the **LDR** instruction before the **SUB** instruction would change the program semantics. It is possible to move the **ADD** and the **LDR** instructions before the **SUB** instruction if we allow the contents of the register r6 to be spilled and restored from the stack as shown below:

```
; all other registers are in use
    str   r6,[sp, #-4]!
    add   r0,r4,r5
    ldr   r6, [r0]
    mov   r2, r2, LSL #2
    orr   r9, r9, #0xf
    add   r8, r6, r8
    ldr   r6, [sp], #4
    add   r8, r8, #4
    orr   r8,r8, #0xf
    sub   r1, r6, r7
    mul   r3,r6, r2
; The value in register r6 is not used after this
```

As can be seen above, the contents of the register r6 have been spilled to the stack and subsequently loaded back to the register r6 to retain the program semantics. Another way to optimize the code above is with the use of the preload instruction as shown below:

```
; all other registers are in use
    add   r0,r4, r5
    pld   [r0]
    sub   r1, r6, r7
    mul   r3,r6, r2
    mov   r2, r2, LSL #2
    orr   r9, r9, #0xf
    ldr   r6, [r0]
    add   r8, r6, r8
    add   r8, r8, #4
    orr   r8,r8, #0xf
; The value in register r6 is not used after this
```

The IXP42X product line and IXC1100 control plane processors have four fill-buffers that are used to fetch data from external memory when a data-cache miss occurs. The IXP42X product line and IXC1100 control plane processors stall when all fill buffers are in use. This happens when more than 4 loads are outstanding and are being fetched from memory. As a result, the code written should ensure that no more than four loads are outstanding at the same time. For example, the number of loads issued

sequentially should not exceed four. Also note that a preload instruction may cause a fill buffer to be used. As a result, the number of preload instructions outstanding should also be considered to arrive at the number of loads that are outstanding.

Similarly, the number of write buffers also limits the number of successive writes that can be issued before the processor stalls. No more than eight stores can be issued. Also note that if the data caches are using the write-allocate with write-back policy, then a load operation may cause stores to the external memory if the read operation evicts a cache line that is dirty (modified). The number of sequential stores may be limited by this fact.

3.10.5.1.1 Scheduling Load and Store Double (LDRD/STRD)

The IXP42X product line and IXC1100 control plane processors introduce two new double word instructions: **LDRD** and **STRD**. **LDRD** loads 64 bits of data from an effective address into two consecutive registers, conversely, **STRD** stores 64 bits from two consecutive registers to an effective address. There are two important restrictions on how these instructions may be used:

- The effective address must be aligned on an 8-byte boundary
- The specified register must be even (r0, r2, etc.).

If this situation occurs, using **LDRD**/**STRD** instead of **LDM**/**STM** to do the same thing is more efficient because **LDRD**/**STRD** issues in only one/two clock cycle(s), as opposed to **LDM**/**STM** which issues in four clock cycles. Avoid **LDRD**s targeting R12; this incurs an extra cycle of issue latency.

The **LDRD** instruction has a result latency of 3 or 4 cycles depending on the destination register being accessed (assuming the data being loaded is in the data cache).

```
    add   r6, r7, r8
    sub   r5, r6, r9
; The following ldrd instruction would load values
; into registers r0 and r1
    ldrd r0, [r3]
    orr   r8, r1, #0xf
    mul   r7, r0, r7
```

In the code example above, the **ORR** instruction would stall for three cycles because of the four cycle result latency for the second destination register of an **LDRD** instruction. The code shown above can be rearranged to remove the pipeline stalls:

```
; The following ldrd instruction would load values
; into registers r0 and r1
    ldrd  r0, [r3]
    add   r6, r7, r8
    sub   r5, r6, r9
    mul   r7, r0, r7
    orr   r8, r1, #0xf
```

Any memory operation following a **LDRD** instruction (**LDR**, **LDRD**, **STR** and so on) would stall for 1 cycle.

```
; The str instruction below would stall for 1 cycle
    ldrd  r0, [r3]
    str   r4, [r5]
```

3.10.5.1.2 Scheduling Load and Store Multiple (LDM/STM)

LDM and **STM** instructions have an issue latency of 2-20 cycles depending on the number of registers being loaded or stored. The issue latency is typically two cycles plus an additional cycle for each of the registers being loaded or stored assuming a data cache hit. The instruction following an LDM would stall whether or not this instruction depends on the results of the load. A **LDRD** or **STRD** instruction does not suffer from this drawback (except when followed by a memory operation) and should be used where possible. Consider the task of adding two 64-bit integer values. Assume that the addresses of these values are aligned on an 8-byte boundary. This can be achieved using the **LDM** instructions as shown below:

```
; r0 contains the address of the value being copied
; r1 contains the address of the destination location
    ldm   r0, {r2, r3}
    ldm   r1, {r4, r5}
    adds  r0, r2, r4
    adc   r1,r3, r5
```

If the code were written as shown above, assuming all the accesses hit the cache, the code would take 11 cycles to complete. Rewriting the code as shown below using **LDRD** instruction would take only seven cycles to complete. The performance would increase further if we can fill in other instructions after **LDRD** to reduce the stalls due to the result latencies of the **LDRD** instructions.

```
; r0 contains the address of the value being copied
; r1 contains the address of the destination location
    ldrd  r2, [r0]
    ldrd  r4, [r1]
    adds  r0, r2, r4
    adc   r1,r3, r5
```

Similarly, the code sequence shown below takes five cycles to complete.

```
    stm   r0, {r2, r3}
    add   r1, r1, #1
```

The alternative version which is shown below would only take three cycles to complete.

```
    strd  r2, [r0]
    add   r1, r1, #1
```

3.10.5.2 Scheduling Data Processing Instructions

Most IXP42X product line and IXC1100 control plane processors' data processing instructions have a result latency of one cycle. This means that the current instruction is able to use the result from the previous data processing instruction. However, the result latency is two cycles if the current instruction needs to use the result of the previous data processing instruction for a shift by immediate. As a result, the following code segment would incur a one-cycle stall for the MOV instruction:

```
    sub   r6, r7, r8
    add   r1, r2, r3
    mov   r4, r1, LSL #2
```

The code above can be rearranged as follows to remove the one-cycle stall:

```
add   r1, r2, r3
sub   r6, r7, r8
mov   r4, r1, LSL #2
```

All data processing instructions incur a two cycle issue penalty and a two-cycle result penalty when the shifter operand is a shift/rotate by a register or shifter operand is RRX. Since the next instruction would always incur a 2 cycle issue penalty, there is no way to avoid such a stall except by re-writing the assembler instruction. Consider the following segment of code:

```
mov   r3, #10
mul   r4, r2, r3
add   r5, r6, r2, LSL r3
sub   r7, r8, r2
```

The subtract instruction would incur a one-cycle stall due to the issue latency of the add instruction as the shifter operand is shift by a register. The issue latency can be avoided by changing the code as follows:

```
mov   r3, #10
mul   r4, r2, r3
add   r5, r6, r2, LSL #10
sub   r7, r8, r2
```

3.10.5.3 Scheduling Multiply Instructions

Multiply instructions can cause pipeline stalls due to either resource conflicts or result latencies. The following code segment would incur a stall of zero to three cycles depending on the values in registers r1, r2, r4 and r5 due to resource conflicts.

```
mul   r0, r1, r2
mul   r3, r4, r5
```

The following code segment would incur a stall of one to three cycles, depending on the values in registers r1 and r2 due to result latency.

```
mul   r0, r1, r2
mov   r4, r0
```

Note that a multiply instruction that sets the condition codes blocks the whole pipeline. A four-cycle multiply operation that sets the condition codes behaves the same as a 4 cycle issue operation. Consider the following code segment:

```
muls  r0, r1, r2
add   r3, r3, #1
sub   r4, r4, #1
sub   r5, r5, #1
```

The add operation above would stall for three cycles if the multiply takes four cycles to complete. It is better to replace the code segment above with the following sequence:

```
mul   r0, r1, r2
add   r3, r3, #1
sub   r4, r4, #1
sub   r5, r5, #1
cmp   r0, #0
```

Please refer to "Instruction Latencies" on page 160 to get the instruction latencies for various multiply instructions. The multiply instructions should be scheduled taking into consideration these instruction latencies.

3.10.5.4 Scheduling SWP and SWPB Instructions

The **SWP** and **SWPB** instructions have a five-cycle issue latency. As a result of this latency, the instruction following the **SWP**/**SWPB** instruction would stall for 4 cycles. **SWP** and **SWPB** instructions should, therefore, be used only where absolutely needed.

For example, the following code may be used to swap the contents of two memory locations:

```
; Swap the contents of memory locations pointed to by r0 and r1
    ldr   r2, [r0]
    swp   r2, [r1]
    str   r2, [r1]
```

The code above takes nine cycles to complete. The rewritten code below, takes six cycles to execute:

```
; Swap the contents of memory locations pointed to by r0 and r1
    ldr   r2, [r0]
    ldr   r3, [r1]
    str   r2, [r1]
    str   r3, [r0]
```

3.10.5.5 Scheduling the MRA and MAR Instructions (MRRC/MCRR)

The **MRA** (**MRRC**) instruction has an issue latency of one cycle, a result latency of two or three cycles depending on the destination register value being accessed and a resource latency of two cycles.

Consider the code sample:

```
    mra   r6, r7, acc0
    mra   r8, r9, acc0
    add   r1, r1, #1
```

The code shown above would incur a one-cycle stall due to the two-cycle resource latency of an **MRA** instruction. The code can be rearranged as shown below to prevent this stall.

```
    mra   r6, r7, acc0
    add   r1, r1, #1
    mra   r8, r9, acc0
```

Similarly, the code shown below would incur a two-cycle penalty due to the three-cycle result latency for the second destination register.

```
    mra   r6, r7, acc0
    mov   r1, r7
    mov   r0, r6
    add   r2, r2, #1
```

The stalls incurred by the code shown above can be prevented by rearranging the code:

```
mra   r6, r7, acc0
add   r2, r2, #1
mov   r0, r6
mov   r1, r7
```

The **MAR** (**MCRR**) instruction has an issue latency, a result latency, and a resource latency of two cycles. Due to the two-cycle issue latency, the pipeline would always stall for one cycle following a **MAR** instruction. The use of the **MAR** instruction should, therefore, be used only where absolutely necessary.

3.10.5.6 Scheduling the MIA and MIAPH Instructions

The **MIA** instruction has an issue latency of one cycle. The result and resource latency can vary from one to three cycles depending on the values in the source register.

Consider the following code sample:

```
mia   acc0, r2, r3
mia   acc0, r4, r5
```

The second **MIA** instruction above can stall from zero to two cycles depending on the values in the registers r2 and r3 due to the one-to-three-cycle resource latency.

Similarly, consider the following code sample:

```
mia   acc0, r2, r3
mra   r4, r5, acc0
```

The **MRA** instruction above can stall from zero to two cycles depending on the values in the registers r2 and r3 due to the one-to-three-cycle result latency.

The **MIAPH** instruction has an issue latency of one cycle, result latency of two cycles and a resource latency of two cycles.

Consider the code sample shown below:

```
add   r1, r2, r3
miaph acc0, r3, r4
miaph acc0, r5, r6
mra   r6, r7, acc0
sub   r8, r3, r4
```

The second **MIAPH** instruction would stall for one-cycle due to a two-cycle resource latency. The **MRA** instruction would stall for one-cycle due to a two-cycle result latency. These stalls can be avoided by rearranging the code as follows:

```
miaph acc0, r3, r4
add   r1, r2, r3
miaph acc0, r5, r6
sub   r8, r3, r4
mra   r6, r7, acc0
```

3.10.5.7 Scheduling MRS and MSR Instructions

The **MRS** instruction has an issue latency of one cycle and a result latency of two cycles. The **MSR** instruction has an issue latency of 2 cycles (6 if updating the mode bits) and a result latency of one cycle.

Consider the code sample:

```
mrs   r0, cpsr
orr   r0, r0, #1
add   r1, r2, r3
```

The **ORR** instruction above would incur a one cycle stall due to the two-cycle result latency of the **MRS** instruction. In the code example above, the **ADD** instruction can be moved before the **ORR** instruction to prevent this stall.

3.10.5.8 Scheduling CP15 Coprocessor Instructions

The **MRC** instruction has an issue latency of one cycle and a result latency of three cycles. The **MCR** instruction has an issue latency of one cycle.

Consider the code sample:

```
add   r1, r2, r3
mrc   p15, 0, r7, C1, C0, 0
mov   r0, r7
add   r1, r1, #1
```

The **MOV** instruction above would incur a two-cycle latency due to the three-cycle result latency of the mrc instruction. The code shown above can be rearranged as follows to avoid these stalls:

```
mrc   p15, 0, r7, C1, C0, 0
add   r1, r2, r3
add   r1, r1, #1
mov   r0, r7
```

3.10.6 Optimizing C Libraries

Many of the standard C library routines can benefit greatly by being optimized for the IXP42X product line and IXC1100 control plane processors architecture. The following string and memory manipulation routines should be tuned to obtain the best performance from the IXP42X product line and IXC1100 control plane processors' architecture (instruction selection, cache usage and data prefetch):

strcat, strchr, strcmp, strcoll, strcpy, strcspn, strlen, strncat, strncmp, strpbrk, strrchr, strspn, strstr, strtok, strxfrm, memchr, memcmp, memcpy, memmove, memset

3.10.7 Optimizations for Size

For applications such as cell phone software it is necessary to optimize the code for improved performance while minimizing code size. Optimizing for smaller code size will, in general, lower the performance of your application. This section contains techniques for optimizing for code size using the IXP42X product line and IXC1100 control plane processors instruction set.

3.10.7.1 Space/Performance Trade Off

Many optimizations mentioned in the previous sections improve the performance of ARM code. However, using these instructions will result in increased code size. Use the following optimizations to reduce the space requirements of the application code.

3.10.7.1.1 Multiple Word Load and Store

The **LDM**/**STM** instructions are one word long and let you load or store multiple registers at once. Use the **LDM**/**STM** instructions instead of a sequence of loads/stores to consecutive addresses in memory whenever possible.

3.10.7.1.2 Use of Conditional Instructions

Using conditional instructions to expand if-then-else statements as described in "Conditional Instructions" on page 173 will result in increasing the size of the generated code. Therefore, do not use conditional instructions if application code space requirements are an issue.

3.10.7.1.3 Use of PLD Instructions

The preload instruction **PLD** is only a hint, it does not change the architectural state of the processor. Using or not using them will not change the behavior of your code, therefore, you should avoid using these instructions when optimizing for space.

§ §

4.0 Network Processor Engines (NPE)

The Network Processor Engines (NPE) are dedicated function processors containing hardware co-processors that are integrated into the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor. The NPEs are used to off load processing functions required by the Intel XScale® Processor.

The Network Processor Engines are high-performance, hardware multi-threaded processors with additional local hardware assist functionality used to off load processor intensive functions such as MII (MAC), CRC checking/generation, AAL2, DES, 3DES, AES, SHA-1, MD-5, etc.

Note: Certain NPEs are not available — depending on which of the IXP42X product line and IXC1100 control plane processors is used. Table 94 shows which network-processor models have these NPEs available.

Table 94. Network Processor Functions

Device	UTOPIA	HSS	MII 0	MII 1	AES / DES / 3DES	Multi-Channel HDLC	SHA-1 / MD-5
IXP425	X	X	X	X	X	8	X
IXP423	X	X	X	X		8	
IXP422			X	X	X		X
IXP421	X	X	X			8	
IXP420			X	X			
IXC1100			X	X			

All instruction code for the NPEs is stored locally with a dedicated instruction memory bus and dedicated data memory bus. These engines support processing of dedicated peripherals interfaces on the IXP42X product line and IXC1100 control plane processors. The peripherals supported by the use of the NPEs are the 2-MII interfaces, UTOPIA-2 interface, and two high-speed serial interfaces.

The NPE core is a hardware multi-threaded processor engine that is used to accelerate functions that are difficult to achieve high performance in a standard RISC processor. Each NPE core is a 133-MHz processor core that contains self-contained instruction memory and self-contained data memory that operate in parallel.

In addition to having separate instruction/data memory and local code store, the NPE core supports hardware multi-threading with support for multiple contexts. The support of hardware multi-threading allows an efficient processor engine with minimal processor stalls due to the ability of the processor core to switch context to a new context in a single clock cycle based upon a prioritized/preemptive basis.

The prioritized/preemptive nature of the context switching allows time critical applications to be implemented in a low-latency fashion, which is required when processing multi-media applications. The NPE core also connects several hardware-based co-processors. The co-processors are used to implement several functions that

are difficult for a processor to implement. The type of functions implemented by the co-processors are serialization/de-serialization, CRC checking/generation, DES/3DES, AES, SHA-1, MD-5, and HDLC bit-stuffing/de-stuffing. These coprocessors are implemented in hardware, therefore allowing the coprocessors and the Network Processor Engine core to operate in parallel.

The combined forces of the hardware multi-threading, local code store, independent instruction memory, independent data memory, and parallel processing allows the Intel XScale processor to be utilized for application purposes. The multi-processing capability of the peripheral interface functions allows unparalleled performance to be achieved by the application running on the Intel XScale processor.

§ §

5.0 Internal Bus

The internal bus architecture of the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor are designed to allow parallel processing to occur and isolate bus utilization based upon particular traffic patterns. The bus is segmented into three major buses, the North AHB, the South AHB, and the APB.

The North AHB is a 133-MHz, 32-bit bus that can be mastered by the WAN/Voice Network Processor Engine (NPE), Ethernet NPE A, or Ethernet NPE B. The targets of the North AHB can be the SDRAM or the AHB/AHB Bridge. The AHB/AHB Bridge will allow access by the NPEs to the peripherals and internal targets on the South AHB.

Data transfers by the NPEs from the North AHB to the South AHB are targeted predominately to the queue manager. Transfers to the AHB/AHB Bridge may be “posted” when writing or “split” when reading — allowing control of the North AHB to be given to another master on the North AHB and allowing the bus to achieve maximum efficiency.

Transfers to the AHB/AHB Bridge are considered to be small and infrequent, relative to the traffic passed between the NPEs on the North AHB and the SDRAM.

The South AHB is a 133-MHz, 32-bit bus that can be mastered by the Intel XScale® Processor, PCI Controller, and the AHB/AHB Bridge. The targets of the South AHB can be the SDRAM, PCI Controller, Queue Manager, Expansion Bus Controller, or the AHB/APB Bridge. Accessing across the AHB/APB Bridge allows interfacing to peripherals attached to the APB Bus.

The APB is a 66.66 MHz (which is 2 * OSC_IN input pin.) (32-bit bus that can be mastered by the AHB/APB Bridge only. The targets of the APB can be the High-Speed UART Interface, Console UART Interface, USB v 1.1 interface, all NPEs, the Internal Bus Performance Monitoring Unit (PMU), Interrupt Controller, GPIO, and Timers. The APB interface to the NPEs is used for NPE code download, part configuration, and status collection.

The maximum length that any AHB master can hold the AHB is for eight 32-bit words. This feature allows for fairness among all masters on the AHBs.

5.1 Internal Bus Arbiters

The Intel® IXP42X product line and IXC1100 control plane processors contain two internal bus arbiters, one arbiter for North AHB transactions and one arbiter for South AHB transactions. The arbiters are used to ensure that at any particular time only one AHB master has access to a given AHB. The arbiters perform this function by observing all of the AHB master requests to the given AHB segment and deciding which AHB master will be the next owner of the AHB.

The arbiters have a standard interface to all bus masters and split-capable slaves in the system. Any AHB master can request an AHB at any cycle. The arbiters sample the AHB requests. If the particular AHB master is requesting the AHB and is next in the round robin list, the arbiter will grant the Master the AHB.

The arbiters also have the capability to handle split transfers. A split transfer is when:

- An AHB master request a read from a split capable AHB target
- The split capable AHB target issues a split transfer indication to the arbiter
- The arbiter allows other transactions to take place on the AHB while the AHB master that issued the request that resulted in the split transfer waits on the read data to be returned from the split capable AHB target
- The split capable AHB target completes the read transaction and notifies the arbiter
- The arbiter will grant the AHB master that requested the split transfer the bus in the normal round robin progression
- The read data will be transferred from the split capable AHB target to the AHB master that issued the request that resulted in the split transfer

All split capable AHB targets split a single AHB master read request at any given instance. If the split capable AHB target receives another read request while servicing a split transaction, the split capable AHB target will issue a retry. The only split capable targets on the South AHB is the Expansion Bus. The only split capable target on the North AHB is the AHB/AHB Bridge.

The arbiters send event information to the Internal Bus Performance Monitoring Unit (IBPMU) so that the North AHB and South AHB bus performance can be observed. The events that may be monitored are provided in section 3.11 Internal Bus Performance Monitoring Unit (IBPMU).

The North AHB Arbiter is identical to the South AHB Arbiter in all respects except for the bus masters and targets in which they are connected.

5.1.1 Priority Mechanism

The arbiters allow the bus initiators access to the AHBs using a round-robin scheme. Table 95 illustrates a generic arbitration example for three AHB masters requesting the AHB. The functionality of the independent arbiters is identical.

Each of the bus initiators (X, Y, and Z) is constantly requesting the bus. The bottom row of Table 95 lists the current bus initiator/winner of the initiators. For example, when all three masters are requesting access, X will be the winner, and then Y and Z will be requesting. Next, Y wins the AHB and X returns with a new request. So ZX are still valid with Z being the oldest. Next, Z wins the bus, etc.

Table 95. Bus Arbitration Example: Three Requesting Masters

	Initial	+1	+2	+3	+4	+5	+6	+7	+8	+9
Requesting Masters	XYZ	YZ	ZX	XY	YZ	ZX	XY	YZ	ZX	XY
Winning Bus Initiator	-	X	Y	Z	X	Y	Z	X	Y	Z

5.2 Memory Map

Table 96 shows the memory map of peripherals connected to the AHB.

Table 96. Memory Map

Start Address	End Address	Size	Use
0000_0000	0FFF_FFFF	256 MB	Expansion Bus Data (Mirrored)/SDRAM Data†
1000_0000	2FFF_FFFF	768 MB	SDRAM Data (Aliased)†
3000_0000	3FFF_FFFF	256 MB	(Reserved)
4000_0000	47FF_FFFF	128 MB	(Reserved)
4800_0000	4FFF_FFFF	128 MB	PCI Data
5000_0000	5FFF_FFFF	256 MB	Expansion Bus Data
6000_0000	63FF_FFFF	64 MB	Queue manager
6400_0000	BFFF_FFFF		(Reserved)
C000_0000	C3FF_FFFF	64 MB	PCI Controller Configuration and Status Registers
C400_0000	C7FF_FFFF	64 MB	Expansion Bus Configuration Registers
C800_0000	C800_0FFF	1 KB	High-Speed UART
C800_1000	C800_1FFF	1 KB	Console UART
C800_2000	C800_2FFF	1 KB	Internal Bus Performance Monitoring Unit
C800_3000	C800_3FFF	1 KB	Interrupt Controller
C800_4000	C800_4FFF	1 KB	GPIO Controller
C800_5000	C800_5FFF	1 KB	Timers
C800_6000	C800_6FFF	1 KB	WAN/Voice NPE = NPE-A (IXP400 software Definition)– Not User Programmable
C800_7000	C800_7FFF	1 KB	Ethernet NPE A = NPE-B (IXP400 software Definition) – Not User Programmable
C800_8000	C800_8FFF	1 KB	Ethernet NPE B = NPE-C (IXP400 software Definition) – Not User Programmable
C800_9000	C800_9FFF	1 KB	Ethernet MAC A
C800_A000	C800_AFFF	1 KB	Ethernet MAC B
C800_B000	C800_BFFF	1 KB	USB Controller
C800_C000	C800_FFFF		(Reserved)
C801_0000	CBFF_FFFF		(Reserved)
CC00_0000	CC00_00FF	256 Byte	SDRAM Configuration Registers
CC00_0100	FFFF_FFFF		(Reserved)

† The lowest 256 MB of address space is configurable based on the value of a configuration register located in the Expansion Bus Controller.

- When bit 31 (MEM_MAP) of configuration register #0 (EXP_CNFG0) is set to logic 1, the Expansion Bus occupies the lowest 256 MB of address space.
- When bit 31 (MEM_MAP) of configuration register #0 (EXP_CNFG0) is set to logic 0 the Expansion Bus occupies 256 MB of address space starting at 5000_0000 while the SDRAM occupies the lowest 256 MB of address address space

  In both cases, regardless of the value of MEM_MAP, the SDRAM occupies the 768 MB (1000_0000 to 2FFF_FFFF) immediately following the lowest 256 MB and the Expansion Bus can be accessed starting at address 5000_0000.

  The largest SDRAM memory size supported by the Intel® IXP42X product line and IXC1100 control plane processors is 256 MB. The actual memory implemented in any given configuration will be aliased (repeated) to fill the 1 GB SDRAM address space. Due to aliasing, all of the SDRAM will be accessible even when the Expansion Bus occupies the lowest 256 MB of address space. On reset, bit 31 (MEM_MAP) of configuration register #0 (EXP_CNFG0) in the Expansion Bus will be set to logic 1. This setting is required because the dedicated boot memory is flash memory located on the Expansion Bus.

§ §

6.0 PCI Controller

The Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor contains a 32-bit, 66-MHz PCI interface compatible with PCI Version 2.2. The PCI interface is capable of operating as either a host or an option (i.e., not the host). The PCI Controller supports these modes of operation by enabling access to the Intel® IXP42X product line and IXC1100 control plane processors' configuration register space from either the Intel XScale® Processor when operating as a host or from an external PCI Device using the PCI Bus when configured as an option. Initiator or target operations are supported by the PCI interface irrelevant of the IXP42X product line and IXC1100 control plane processors' configurations (could be configured as a PCI Host or PCI Option Function).

When the PCI Controller is configured as a host, an internal PCI arbiter may be utilized to allow up to four devices to be connected to the IXP42X product line and IXC1100 control plane processors without the need for an external arbiter. However, even though the internal PCI arbiter exists, the internal PCI arbiter is not required to be used when the PCI Controller is configured in host or for that matter option mode of operation.The arbiter functionality is completely independent from the PCI mode of operation. An example connection of this configuration is contained in Figure 30. The PCI arbiter function will allow access to the PCI bus in a round-robin fashion. The PCI Controller operating as an initiator can generate Memory, I/O, or Configuration PCI bus cycles.

Operating as a target, the PCI Controller can accept Memory, I/O, or Configuration PCI bus cycles. When the PCI Controller is configured as an option, the internal PCI arbiter can be disabled and REQ0/GNT0 are used to connect to an external arbiter on the PCI bus.

An example of the IXP42X product line and IXC1100 control plane processors connected in this configuration is shown in Figure 31. The PCI arbiter can be enabled/ disabled independently from the PCI host/option configuration. The PCI Controller also contains two DMA engines to allow data movement between the PCI bus and the SDRAM without the aid of the Intel XScale processor.

Figure 30. Processors' PCI Bus Configured as a Host

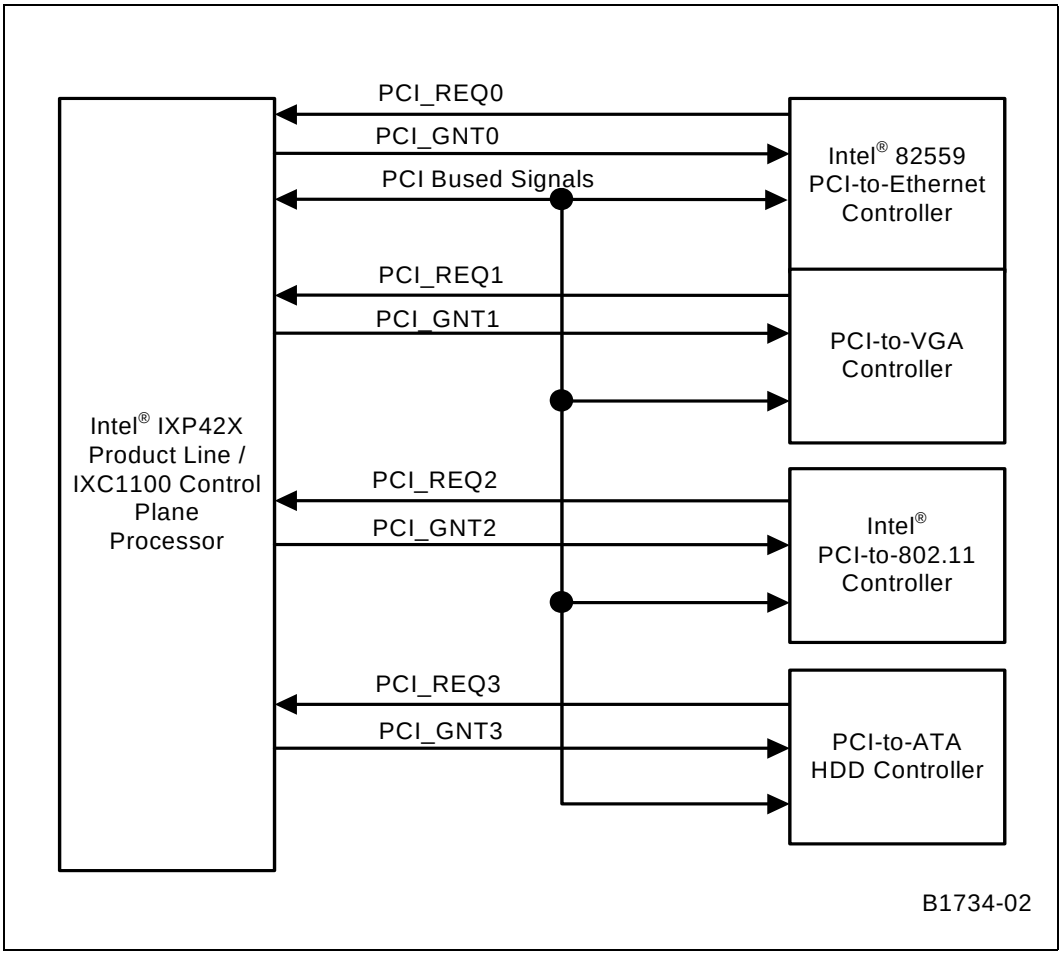
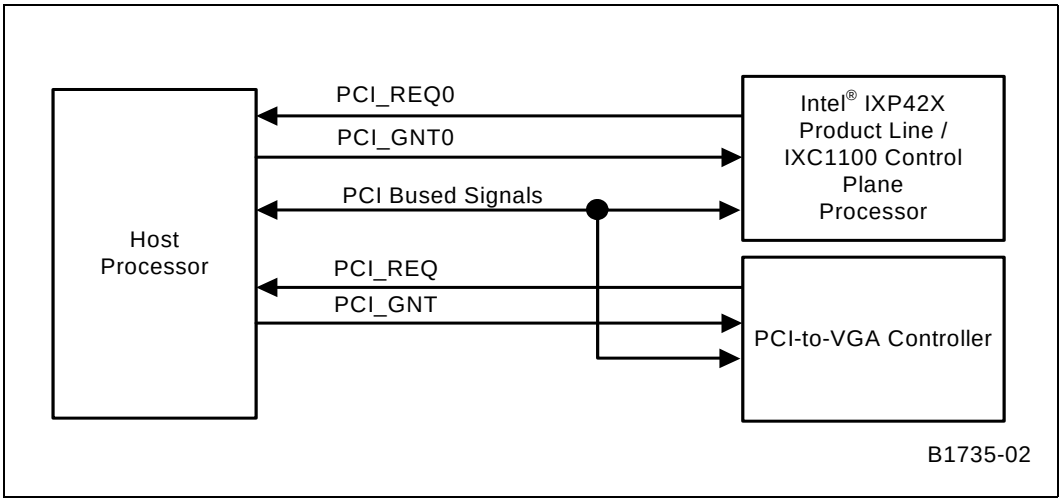
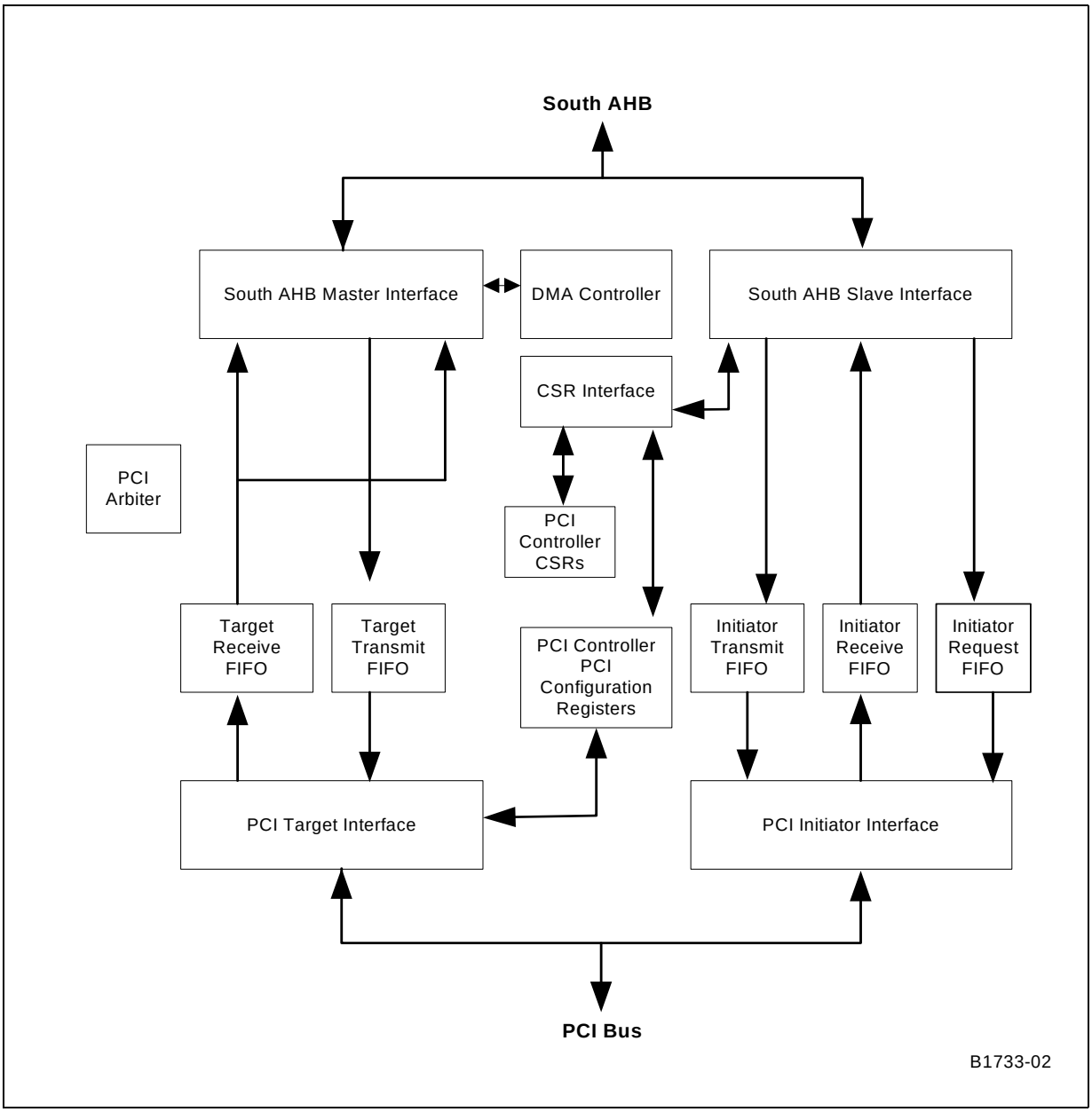


Figure 31. Processors' PCI Bus Configured as an Option



The IXP42X product line and IXC1100 control plane processors PCI Controller block diagram is given in Figure 32.

Figure 32. Processors' PCI Controller Block Diagram



The PCI Controller has two main interfaces that allow interconnection between an external PCI bus and the South AHB. The external PCI bus has both a target and an initiator controller that interfaces to the PCI bus.

When an external PCI device want to use the IXP42X product line and IXC1100 control plane processors as the target of a PCI transfer, the PCI Controller Target interface will interpret the data and forward the appropriate information (data/address/control) to the Target interface FIFOs. The Target Interface FIFOs are 8 words deep. The PCI-

target interface — in conjunction with the target interface FIFOs — will use the South AHB Master interface of the PCI Controller to provided read and write access to AHB agents, PCI Controller PCI Configuration registers (through Configuration cycles). The PCI Controller Configuration and Status Registers accessible through Target transactions will be accessed directly from the PCI-Target Interface.

Table 97 lists the supported command types, when the IXP42X product line and IXC1100 control plane processors are used as a target of a PCI transaction. The PCI Target interface does not support the following features:

- Lock cycles
- VGA palette snoop
- Dual address cycles
- Cache-line wrap mode addressing (Disconnected after first data phase of transaction)
- Type 1 Configuration space

Table 97. PCI Target Interface Supported Commands

PCI Command/Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	Not Supported
0x1	Special Cycle	Not Supported
0x2	I/O Read	Supported
0x3	I/O Write	Supported
0x4	(Reserved)	
0x5	(Reserved)	
0x6	Memory Read	Supported
0x7	Memory Write	Supported
0x8	(Reserved)	
0x9	(Reserved)	
0xA	Configuration Read	Supported
0xB	Configuration Write	Supported
0xC	Memory Read Multiple	Converted To Memory Read
0xD	Dual Address Cycle	Not Supported
0xE	Memory Read Line	Converted To Memory Read
0xF	Memory Write and Invalidate	Converted To Memory Write

When the IXP42X product line and IXC1100 control plane processors want to use an external PCI device as the target of a PCI transfer, the PCI Controller Initiator interface will be used to generate the appropriate PCI bus cycles and forward the information to the PCI bus. There are three ways in which PCI bus cycles may be initiated:

— The DMA channels generate PCI Memory cycles. Refer to "PCI Controller DMA Controller" on page 234 for additional details.

— AHB masters generate PCI Memory cycles using memory-mapped direct access on the AHB bus. Refer to "Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register" on page 220 for additional details.

— Interrupt Acknowledge, Special Cycle, I/O, Configuration, and single-data-phase Memory cycles are generated indirectly by AHB masters using a Non-

pre-fetch CSR mechanism. Refer to "PCI Controller Configured as Host" on page 213 for additional details.

For PCI bus memory cycles, the PCI Initiator interface will receive requests for PCI transfer from the PCI Controller DMA channels. For PCI bus I/O cycles and configuration cycles, the PCI Initiator interface will receive requests for PCI transfer from an AHB master (a single word PCI Bus Memory Cycle can be produced using this method).

Requests for PCI transfers using the PCI Controller Initiator interface are buffered in the Initiator Request FIFO and handled by the PCI Controller Initiator interface when appropriate. The Initiator interface will receive the appropriate transfer information from the Initiator Request FIFO; which is the address, word count, byte enables, and PCI command type.

The Initiator Request FIFO is a four-entry FIFO allowing up to four requests to be buffered. If a request is issued that generates an initiator transaction and the Initiator Request FIFO is already full, a retry will be issued to the AHB master that initiated the request.

After gathering the appropriate information, the PCI Initiator interface performs the specified transaction on the PCI bus, handling all bus protocol and any retry/disconnect situations. The data will be moved from the South AHB to the PCI bus using the Initiator Data FIFOs. The Initiator Data FIFOs are eight words deep. Table 98 lists the supported PCI transaction types produced by the IXP42X product line and IXC1100 control plane processors' PCI Controller Initiator Interface.

Table 98. PCI Initiator Interface-Supported Commands

PCI Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	Supported
0x1	Special Cycle	Supported
0x2	I/O Read	Supported
0x3	I/O Write	Supported
0x4	(Reserved)	
0x5	(Reserved)	
0x6	Memory Read	Supported
0x7	Memory Write	Supported
0x8	(Reserved)	
0x9	(Reserved)	
0xA	Configuration Read	Supported
0xB	Configuration Write	Supported
0xC	Memory Read Multiple	Not supported
0xD	Dual Address Cycle	Not supported
0xE	Memory Read Line	Not supported
0xF	Memory Write and Invalidate	Not supported

It is important to note that the target interface — and the DMA channels used for supporting the initiator interface — can contend for the use of the South AHB Master Controller. When this contention occurs, arbitration for control of South AHB Master Controller is carried out on two levels.

On the first level, the PCI Target Interface requests and the DMA requests alternate for priority access. On the first transaction the PCI Target interface would gain access to the South AHB Master Controller's services, followed by one of the DMA channels

gaining access to the South AHB Master Controller's services, then the PCI Target interface would gain access to the South AHB Master Controller's services again, followed by the second DMA channel gaining access to the South AHB Master Controller's services, etc.

On the second level of arbitration for the South AHB Master Controller's services, the two DMA channels will alternate for priority access. DMA channel 0 would gain access to the South AHB Master Controller's services first, followed by DMA channel 1, and then DMA channel 0, etc. This arbitration scheme balances the high-bandwidth DMA traffic with what should be lower bandwidth PCI Target Interface traffic and is only used in cases where contention exists. For instance, if there are only PCI Target Interface requests being received by the South AHB Master Controller. The PCI Target would continually get access to the South AHB Master Controller until a DMA request is detected.

The PCI Controller also contains two configuration spaces. The PCI Controller Control and Status Register (CSR) configuration space is used to configure the PCI Controller, initiate single cycle PCI transactions using the non-pre-fetch registers, operate the DMA channels, report PCI Controller status, and allow access to the PCI Controller PCI Configuration Registers. The PCI Configuration Space is a 64-byte, PCI type-0 configuration space that supports a single function.

The PCI Configuration Space can be written or read using registers defined in the Control and Status Registers when the IXP42X product line and IXC1100 control plane processors are configured as a PCI Host. An external PCI Master using PCI Configuration Cycles can write or read the PCI Configuration Space when the IXP42X product line and IXC1100 control plane processors are configured as an Option. The PCI Configuration Space may be accessed by the Intel XScale processor or the PCI bus but never by both at the same time.

6.1 PCI Controller Configured as Host

The IXP42X product line and IXC1100 control plane processors can be configured as a host function on the PCI bus. Configuring the IXP42X product line and IXC1100 control plane processors as a host does not require the internal PCI arbiter function in the PCI Controller to be enabled.

The first step to using the PCI interface in any mode of operation is to determine the mode of operation and then configure the interface. The PCI bus mode of operation can be obtained by reading bit 0 of the PCI Controller Control and Status Register (PCI_CSR). If bit 0 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 0, the IXP42X product line and IXC1100 control plane processors is required to function as an Option on the PCI bus. If bit 0 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 1, the IXP42X product line and IXC1100 control plane processors is required to function as the Host on the PCI bus.

Bit 0 of the PCI Controller Control and Status Register (PCI_CSR) will be set by the logic level contained on Expansion Bus Address Bus bit 1 at the de-assertion of the reset signal supplied to the IXP42X product line and IXC1100 control plane processors. The internal arbiter will be enabled/disabled based on the logic level contained on Expansion Bus Address Bus bit 2 at the de-assertion of the reset signal supplied to the IXP42X product line and IXC1100 control plane processors. The PCI Controller Control and Status Register (PCI_CSR) bit 1 captures the logic level contained on Expansion Bus Address Bus bit 2 at the de-assertion of reset.

If bit 1 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 1, the IXP42X product line and IXC1100 control plane processors's internal arbiter is enabled. If bit 1 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 0, the IXP42X product line and IXC1100 control plane processors' Internal Arbiter is disabled.

Once the PCI controller has determined that the mode of operation is to be host, the IXP42X product line and IXC1100 control plane processors are required to configure the rest of the PCI bus. However, before the IXP42X product line and IXC1100 control plane processors can configure the rest of the PCI bus, the PCI Controller must be configured.

The Configuration and Status Registers must be initialized and the PCI Controller Configuration and Status Registers must be initialized. (For more detail on initializing the Configuration and Status Registers, see "Initializing PCI Controller Configuration and Status Registers for Data Transactions" on page 219. For more detail on initializing the PCI Controller Configuration and Status Registers, see "Initializing the PCI Controller Configuration Registers" on page 222.)

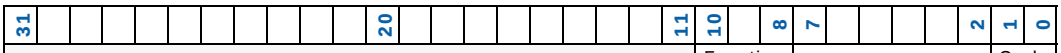
After the local Configuration and Status Register and PCI Controller Configuration and Status Registers have been initialized, the remainder of the PCI bus is ready to be configured by the hosting IXP42X product line and IXC1100 control plane processors. The IXP42X product line and IXC1100 control plane processors will now begin to initiate configuration cycles to all of the potential devices on the PCI bus.

The order and nature in which the devices are learned is up to the individual application. However, one bit that must be configured prior to initiating PCI Configuration Cycles with the IXP42X product line and IXC1100 control plane processors. Bit 2 of the PCI Control Register/Status (PCI_SRCR) Register must be set to logic 1 using the methods described in "Initializing the PCI Controller Configuration Registers" on page 222. The setting of bit 2 to logic 1 enables PCI bus-mastering capability.

Two types of PCI Configuration Cycles can be generated using the IXP42X product line and IXC1100 control plane processors: Type 0 and Type 1 Configuration Cycles. Type 0 Configuration Cycles are use to communicate to a PCI device which is contained on the same local segment that the generator of the Configuration Cycles. Type 1 Configuration Cycles are use to communicate to a PCI device which is contained on another segment of the PCI bus other than the PCI bus segment that is generating the Configuration Cycles, i.e., a segment on the other side of a PCI bridge.

A PCI bus can have up to 32 devices (logically but there are loading restrictions that limit this number) per segment and up to 256 segments. Figure 33 shows the address makeup for Type 0 PCI bus configuration cycles and Figure 34 shows the address makeup Type 1 PCI bus configuration cycles.

Figure 33. Type 0 Configuration Address Phase



Cycle Type=00 for Type 0 Configuration Cycles

Register Number=Defines one of 64 PCI defined 32-bit registers

Function Number=Decodes 1 of 8 possible functions per PCI device (only Function 0 supported for the IXP42X product line and IXC1100 control plane processors)

Figure 34. Type 1 Configuration Address Phase

31 … 24	23 … 16	15 … 11	10 … 8	7 … 2	1 … 0
(Reserved)	Bus Segment Number	Device Number	Function Number	Register Number	Cycle Type

Cycle Type =01 for Type 1 Configuration Cycles

Register Number =Defines one of 64 PCI defined 32-bit registers

Function Number=Decodes one of eight possible functions per PCI device (only Function 0 supported for the IXP42X product line and IXC1100 control plane processors)

Device Number =Decodes one of 32 possible devices per PCI bus segment (may be limited by loading restrictions, refer to the *PCI Local Bus Specification*, Rev. 2.2)

Bus Segment Number = Decodes one of 256 possible bus segments per PCI Bus (refer to the *PCI Local Bus Specification*, Rev. 2.2)

Configuration cycles will be produced by the IXP42X product line and IXC1100 control plane processors using four 32-bit Configuration and Status Registers referred to as the Non-Pre-fetch Registers. These registers are

- PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register
- PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register
- PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register
- PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register

The Non-Pre-fetch Register accesses can also be used to produce memory and I/O PCI bus cycle. However, these cycles can only be single cycle accesses.

Non-Pre-fetch Read Cycles will be implemented by placing a 32-bit PCI address in the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register and then placing the PCI Command Type and Byte Enables for the desired read cycle in the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register. The PCI Controller then will initiate the proper transaction on the PCI bus to read the requested data. Then the returned data is placed in the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register.

To avoid incorrect data from being read by the initiator of this transaction, retries will be issued to any AHB master that attempts to read the PCI Controller Configuration and Status Registers prior to the Non-Pre-fetch PCI read data being placed into the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register. This action assures that the next read of the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register does not contain stale data.

Non-Pre-fetch Write Cycles will be implemented by:

- Placing a 32-bit PCI address in the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register
- Placing the PCI Command Type and Byte Enables for the desired write cycle in the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register
- Writing the data that is to be placed onto the PCI bus into the PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register

The PCI Controller then will initiate the proper transaction on the PCI bus to place the requested write data onto the PCI bus. To avoid the write data from being corrupted by new request from an AHB master, retries will be issued to any AHB master that attempts to write the PCI Controller Configuration and Status Registers prior to the completion of the requested PCI transaction.

It is also noteworthy to mention that the PCI Controller does not interpret or manipulate the contents of the Non-Pre-fetch Registers. The address, command, byte enables, and write data are passed to the PCI bus as-is. For example, I/O read and I/O write requests must be set-up such that the byte-enables are consistent with the 2 LSBs of the address in accordance with the PCI local-bus specification.

Implication: If an external PCI device has non-prefetch memory and requires either a 16-bit or 8-bit read, there is a possibility that the device will not respond correctly to the IXP42X product line and IXC1100 control plane processors' memory reads. This is because the IXP42X product line and IXC1100 control plane processors always perform a 32-bit read to the non-prefetch memory region specified in register PCI_NP_AD.

The 8-bit or 16-bit external device should respond with a "target abort," as per the PCI 2.2 specification, if a 32-bit read is performed to its non-prefetch memory and it requires a 16-bit or 8-bit read.

The IXP42X product line and IXC1100 control plane processors will drive all the byte enables asserted during all memory cycle reads of the external PCI device, no matter what the PCI_NP_CBE register contains in the byte enable bits.

To read non-prefetch memory sub-DWORDS (8-bit or 16-bit), use I/O reads. If it is necessary to use memory cycle reads of sub-DWORDS, a hardware resolution may be required. Contact your Intel field application engineer if you require a hardware resolution.

6.1.1 Example: Generating a PCI Configuration Write and Read

This example examines the initializing of the Base Address Register.

1. Assume a PCI device has been located and now the Base Address Register configuration of this PCI device is going to be initialized. The first step is to write all logic 1s to the PCI Base Address Registers.

   Base Address Register 0 will be located at hexadecimal offset of 0x10 when the ID_SEL of this device is active and the access is a PCI Bus Configuration Cycle. The intent of this exercise is to initialize this Base Address Register.

2. Write a hexadecimal value of 0x00010010 to the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register.

   This value will allow a write to a Type 0 PCI configuration space address location 0x10. Notice also that address bit 16 is set to logic 1. This bit is set, assuming that ID_SEL for a given device on the local segment is selected using address bit 16. This value chosen for PCI_NP_AD follows the convention outlined in Figure 33, "Type 0 Configuration Address Phase" on page 214.

3. Write a hexadecimal value of 0x0000000B to the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register.

   Bits 7:4 of this register specify the byte enables for the data transfer. The selection of all bits to logic 0 signifies that all bytes are to be written. Bits 3:0 of this register specify the PCI Command Type to be used for the data transfer. A logic value of 1011b signifies that a Configuration Write Cycle is being requested.

4. Write a hexadecimal value of 0xFFFFFFFF to the PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register.

   This write to the Configuration and Status Registers will cause a PCI Configuration Write Cycle with all byte-enables active to be initiated on the PCI bus.

5. Base Address Register 0 has been written with all logic 1s. However, only some of these bits will be set to logic 1.

   Logic 1s will only be written to the bits corresponding to a given address space defined for the PCI device. For instance, assume that the PCI device being configured requires a 64-Mbyte address space for Base Address Register 0 used for

memory transactions with no adverse side effects to reads. Only bits (31:26) would be written.

Now, the IXP42X product line and IXC1100 control plane processors must read Base Address Register 0 to determine the Address Space, Address space type (memory or I/O), and any limitations to reading this address space.

6. Write a hexadecimal value of 0x00010010 to the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register.

   This value will allow a read from a Type 0 PCI configuration space address location 0x10. Notice also that address bit 16 is set to logic 1.

   This bit is set assuming that ID_SEL for a given device on the local segment is selected, using address bit 16. This device is the one that is attempting to be accessed.

7. Write a hexadecimal value of 0x0000000A to the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register.

   Bits 7:4 of this register specify the byte-enables for the data transfer. The selection of all bits to logic 0 signifies that all bytes are to be read.

   Bits 3:0 of this register specify the PCI Command Type to be used for the data transfer. A logic value of 1010b signifies that a Configuration Read Cycle is being requested. This action causes the PCI Controller to initiate the read transaction.

8. The data returned will be placed in the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register.

   The returned value looks like hexadecimal 0xFC000008. This value signifies that an address space of 64 Mbyte is being requested by Base Address Register 0 of the PCI device to be mapped anywhere into the PCI address map, the address space is a Memory Space, and there are no special read conditions that apply to this address space. (See the *PCI Local Bus Specification*, Rev. 2.2 for more details.)

9. Now the IXP42X product line and IXC1100 control plane processors must specify the PCI address space that this Base Address Register is going to occupy. This is done by executing a Configuration Write to bits 31:26 of Base Address Register 0 — with the logical value where the address is going to reside.

   Assume we want the address to reside at PCI location 0xA0000000. A Configuration Write of 0xA0000000 will be written to Base Address Register 0 of the external PCI device. No other PCI assignment can be placed between PCI addresses 0xA0000000 and 0xA3FFFFFF.

When the IXP42X product line and IXC1100 control plane processors are functioning in Host mode of operation, all other PCI Configuration Registers contained on external PCI devices will be configured or used to configure the PCI Bus using PCI Configuration Read/Write Cycles produced from the IXP42X product line and IXC1100 control plane processors for each device on the PCI bus. Some examples of these parameters are Device Identifications, Vendor Identifications, Base Address Register, and Grant Latencies. (For more details on exact settings/usage of these parameters for a given application, see the *PCI Local Bus Specification*, Rev. 2.2.)

The IXP42X product line and IXC1100 control plane processors have now been successfully configured as a PCI host and successfully configured the PCI bus. PCI memory and PCI I/O transaction can now take place.

For more detail on generating PCI Memory and PCI I/O transactions using the IXP42X product line and IXC1100 control plane processors, see "PCI Controller Functioning as Bus Initiator" on page 226. For more detail on accepting PCI Memory and PCI I/O transactions using the IXP42X product line and IXC1100 control plane processors, see "PCI Controller Functioning as Bus Target" on page 234.

6.2 PCI Controller Configured as Option

The IXP42X product line and IXC1100 control plane processors can be configured as an option function on the PCI bus. As with configuring the PCI Controller as a host functions, the IXP42X product line and IXC1100 control plane processors, functioning as an option does not require the Internal Arbiter function in the PCI Controller to be enabled. Therefore, the Internal Arbiter can be enabled independently and the host/option configuration can be selected independently. The option function is selected similarly to the manner in which the host function is selected. (For more details, see "PCI Controller Configured as Host" on page 213.)

If the IXP42X product line and IXC1100 control plane processors are configured as an option, an external PCI Host will want to access the PCI Configuration Space of the IXP42X product line and IXC1100 control plane processors. The PCI Host will complete these accesses using PCI Configuration Cycles. However, if the IXP42X product line and IXC1100 control plane processors receive Configuration Cycles prior to being initialized, improper PCI bus configuration may occur.

To prevent this event from occurring, the IXP42X product line and IXC1100 control plane processors can refuse to accept configuration cycles from an external source by programming bit 15 of the PCI Controller Control and Status (PCI_CSR) Register. Bit 15 of the PCI Controller Control and Status (PCI_CSR) Register is the Initialization Complete bit. When bit 15 is set to logic 0, the PCI Controller Target Interface will issue retries to PCI Configuration cycles. When bit 15 is set to logic 1, PCI Configuration Cycles will be accepted.

In Option mode, the PCI bus initialization and bus enumeration will be performed by an external host processor, not the IXP42X product line and IXC1100 control plane processors. However, the processor still has a boot sequence and there are several PCI Configuration registers that must be "initialized" by the Intel XScale processor before the external host starts the PCI initialization. For example, the pci_sidsvid (Subsystem ID and Vendor ID). This register is read-only from the external host processor but is read-write from the Intel XScale processor because its contents are application dependent. So when the IXP42X product line and IXC1100 control plane processors comes out of reset in HOST mode, the PCI Configuration registers are accessible from the Intel XScale processor and inaccessible from the PCI bus. (Note: by inaccessible from the PCI bus, that the IXP42X product line and IXC1100 control plane processors will respond to any bus cycle with a RETRY). This is necessary because the external host cannot be allowed to read any of the PCI Configuration registers before they have been initialized to valid values. Once the registers (like pci_sidsvid) are initialized, S/W writes a 1 to the IC (Init Complete) bit of pci_csr which makes the PCI Configuration Registers accessible from PCI but inaccessible from the Intel XScale processor. So the complete sequence is:

1. Exit reset in Option mode (pci_csr.IC = 0, PCI cycles Retried, Intel XScale processor has access to PCI Configuration registers)
2. The Intel XScale processor initializes PCI Configuration registers as appropriate.
3. Intel XScale processor writes a 1 to pci_csr.IC (PCI cycles now accepted, PCI Configuration registers not accessible from Intel XScale processor)
4. External host initialization of IXP42X product line and IXC1100 control plane processors PCI Configuration Registers can proceed.

The Initialization Complete bit allows time for the IXP42X product line and IXC1100 control plane processors to configure the chip prior to accepting cycles from an external PCI device. If initialization is not completed in the first 2^{25} PCI clocks after the PCI reset signal is de-asserted, the possibility exists for the external PCI Host to assume that no PCI device is resident or active at this particular IDSEL.

An access to the IXP42X product line and IXC1100 control plane processors' PCI Controller PCI Configuration Registers occurs when the PCI_IDSEL input is asserted, the PCI command field as represented by the PCI Command/Byte enable signals is a configuration read or write, PCI_AD[1:0] = 00 indicating a type 0 configuration cycle, and the PCI Controller Target Interface is allowed to accept Type 0 Configuration Cycles by asserting the Initialization Complete bit. The PCI Configuration Register accessed is determined by the value contained on the PCI_AD[7:2] pins during the address phase of the PCI Configuration Transaction. Accesses to the PCI Configuration Register can be a single-word only. The PCI Controller Target Interface will disconnect any burst longer than 1 word.

During reads of the PCI Configuration Registers, byte-enables are ignored and the full 32-bit register value is always returned. Read accesses to unimplemented registers complete normally on the bus and return all zeroes.

During PCI Configuration Register writes, the PCI byte-enables determine the byte(s) that are written within the addressed register. Write accesses to unimplemented PCI Configuration Registers complete normally on the bus but the data is discarded. The PCI Configuration Space supported by the IXP42X product line and IXC1100 control plane processors are a single-function, Type 0 configuration space. (For more information on the PCI Configuration Space and additional configuration details, see "PCI Configuration Registers" on page 249 and the *PCI Local Bus Specification*, Rev. 2.2.)

6.3 Initializing PCI Controller Configuration and Status Registers for Data Transactions

In order to use the PCI Controller for transactions other than single word initiator transaction implemented by Non-Pre-fetch transactions, various registers must be set in the PCI Controller Configuration and Status Registers. The registers that must be initialized are:

- AHB Memory Base Address Register (PCI_AHBMEMBASE)
- AHB I/O Base Address Register (PCI_AHBIOBASE)
- PCI Memory Base Address Register (PCI_PCIMEMBASE).

The AHB Memory Base Address Register (PCI_AHBMEMBASE) is used to map the address of a PCI Memory Cycle Target transfers from the address of the PCI Bus to the address of the South AHB. The AHB I/O Base Address Register (PCI_AHBIOBASE) is used to map the address of a PCI I/O Cycle Target transfers from the address of the PCI Bus to the address of the South AHB. The PCI Memory Base Address Register (PCI_PCIMEMBASE) is used to map the address of direct access PCI memory-mapped transfers from the address of the South AHB to the address of the PCI Bus.

When the IXP42X product line and IXC1100 control plane processors are the target of a PCI bus transaction, the values written or read by external PCI Bus Initiators using the Base Address Registers contained within the IXP42X product line and IXC1100 control plane processors must be translated to an address location within the IXP42X product line and IXC1100 control plane processors. The configuration of the internal memory allocation is implemented differently for each of the Base Address Registers (BAR). The following paragraphs describe the implementation for each of the Base Address Registers.

For Base Address Registers 0 through 3 — which are used to complete PCI Bus Memory Cycles Target transactions — the AHB Memory Base Address (PCI_AHBMEMBASE) register is used to translate PCI Memory Cycle accesses to their appropriate AHB locations. The AHB Memory Base Address (PCI_AHBMEMBASE) register is used to determine the upper 8 AHB address bits when an external Initiator on the PCI bus accesses the memory spaces of the IXP42X product line and IXC1100 control plane

processors. The IXP42X product line and IXC1100 control plane processors PCI Controller can be configured to support four 16-Mbyte locations for PCI Target Memory Cycle transactions using the AHB Memory Base Address (PCI_AHBMEMBASE) register and the PCI Base Address Registers.

The AHB Memory Base Address (PCI_AHBMEMBASE) register consists of four 8-bit fields. Each of these fields corresponds to a PCI Base Address Register

- Bits 31:24 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 0 and the first 16-Mbyte AHB memory location (AHB base 0)
- Bits 23:16 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 1 and the second 16-Mbyte AHB memory location (AHB base 1)
- Bits 15:8 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 2 and the third 16-Mbyte AHB memory location (AHB base 2)
- Bits 7:0 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 3 and the fourth 16-Mbyte AHB memory location (AHB base 3).

Base Address Register 4 is used to complete accesses to internal PCI Controller Configuration and Status registers. (These registers are not the PCI Controller PCI Configuration Registers.) PCI Base Address Register 4 is used to decode that an access has been made to the Configuration and Status Register Space. There are no AHB cycles produced for this type of an access, as all accesses to this Base Address Register will be internal to the PCI controller. Therefore, an address translation register is not required.

For Base Address Register 5 — which is used to complete PCI bus I/O cycles — the AHB I/O Base Address (PCI_AHBIOBASE) register is used to translate I/O PCI accesses to their appropriate AHB locations. The IXP42X product line and IXC1100 control plane processors PCI Controller can be configured to support a single 256-Byte location for PCI target I/O cycle transactions, using the AHB I/O Base Address (PCI_AHBIOBASE) register and PCI Base Address Register 5.

The AHB I/O Base Address (PCI_AHBIOBASE) register consists of a single 24-bit field. The AHB I/O Base Address (PCI_AHBIOBASE) register is used to determine the upper 24 AHB address bits, when an external initiator on the PCI bus accesses the I/O space of the IXP42X product line and IXC1100 control plane processors.

6.3.1 Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register

The following example can be used to understand the operation of the AHB Memory Base Address Register (PCI_AHBMEMBASE), AHB I/O Base Address Register (PCI_AHBIOBASE), and PCI Memory Base Address Register (PCI_PCIMEMBASE).

1. Assume that PCI_AHBMEMBASE = 0x04010506 and PCI_AHBIOBASE = 0x000A1200.
2. Assume that the PCI Bus has gone through configuration and the Base Address Registers (BAR0 – BAR5) are set as follows:
   - BAR0 = 0xA0000000
   - BAR1 = 0xA1000000
   - BAR2 = 0xA2000000
   - BAR3 = 0xA3000000

— BAR4 = 0xA4000000

— BAR5 = 0xA5123400

3. An external PCI device initiates a PCI bus transfer to the IXP42X product line and IXC1100 control plane processors' BAR1. The PCI address looks like the following PCI Address = 0xA100402C. The address placed on the South AHB is 0100402C.

   Notice that the third byte from the right, of the PCI_AHBMEMBASE = 0x04010506, is substituted for the A1 located in the fourth byte from the right of the PCI Address = 0xA100402C.

4. Next, an external PCI device initiates a PCI bus transfer to the IXP42X product line and IXC1100 control plane processors' BAR3. The PCI address looks like the following: PCI Address = 0xA3004014. The address placed on the South AHB is 06004014.

   Notice that the first byte from the right of the PCI_AHBMEMBASE = 0x04010506 is substituted for the A3 located in the fourth byte from the right of the PCI Address = 0xA3004014.

5. PCI I/O space example is an external PCI device initiates a PCI bus transfer to the IXP42X product line and IXC1100 control plane processors' BAR5. The PCI address looks like the following PCI Address = 0xA5123418. The address placed on the South AHB is 0x0A120018.

   Notice that the first three bytes from the right of the PCI_AHBIOBASE = 0x000A1200 is substituted for the A51234 located in the PCI Address = 0xA5123418.

6. The final example is an external PCI device initiates a PCI bus transfer to the IXP42X product line and IXC1100 control plane processors' BAR4. This allows access to the PCI Controller Configuration and Status Register. The PCI address looks like the following PCI Address = 0xA4000038. There is no address placed on the South AHB. This causes an access of the PCI Doorbell Register on the IXP42X product line and IXC1100 control plane processors.

   The PCI Doorbell Register can be used to generate an interrupt to the Intel XScale processor.

When the IXP42X product line and IXC1100 control plane processors are the initiator of a PCI Bus transaction and desires the transaction to produce PCI Memory Transactions, the values may be written or read by providing a transfer to the PCI Memory Cycle Address Space defined for the IXP42X product line and IXC1100 control plane processors. The 64-Mbyte address space defined for the PCI Memory Cycle Address Space is from AHB address location 0x48000000 to 0x4BFFFFFF.

Only four 16-Mbyte windows can be enabled. The four 16-Mbyte windows are divided among the addresses as shown in Table 99.

Table 99. PCI Memory Map Allocation

Description	Starting Address	Ending Address
First 16-Mbyte window	0x48000000	0x48FFFFFF
Second 16-Mbyte window	0x49000000	0x49FFFFFF
Third 16-Mbyte window	0x4A000000	0x4AFFFFFF
Fourth 16-Mbyte window	0x4B000000	0x4BFFFFFF

The four 16-Mbyte windows translate their South AHB addresses to the PCI Bus addresses using the PCI Memory Base Address Register (PCI_PCIMEMBASE).

The PCI Memory Base Address Register (PCI_PCIMEMBASE) register consists of four 8-bit fields. Each of these fields corresponds to a given 16-Mbyte window:

- Bits 31:24 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the first 16-Mbyte window from South AHB address 0x48000000 to 0x48FFFFFF
- Bits 23:16 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the second 16-Mbyte window from South AHB address 0x49000000 to 0x49FFFFFF
- Bits 15:8 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the third 16-Mbyte window from South AHB address 0x4A000000 to 0x4AFFFFFF
- Bits 7:0 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the fourth 16-Mbyte window from South AHB address 0x4B000000 to 0x4BFFFFFF.

The PCI Memory Base Address Register (PCI_PCIMEMBASE) register is used to determine the upper eight PCI address bits when the IXP42X product line and IXC1100 control plane processors access the memory spaces of external Targets on the PCI bus.

6.3.2 Example: PCI Memory Base Address Register and South-AHB Translation

The following example discusses the operation of the PCI Memory Base Address Register (PCI_PCIMEMBASE) and the South AHB translation.

1. Assume that PCI_PCIMEMBASE = 0xC3A24169.
2. The next example shows an access to the first 16-Mbyte window.
   The South AHB address is for the access is 0x48123450. The address presented on the PCI bus is 0xC3123450.
3. The next example shows an access to the second 16-Mbyte window.
   The South AHB address is for the access is 0x49123450. The address presented on the PCI bus is 0xA2123450.
4. The next example shows an access to the third 16-Mbyte window.
   The South AHB address is for the access is 0x4A123450. The address presented on the PCI bus is 0x41123450.
5. The next example shows an access to the fourth 16-Mbyte window.
   The South AHB address is for the access is 0x4B123450. The address presented on the PCI bus is 0x69123450.

6.4 Initializing the PCI Controller Configuration Registers

The PCI Base Address Registers along with any other pertinent PCI Configuration Registers, located in the PCI Controller PCI Configuration Register space, must be initialized by the Intel XScale processor when the IXP42X product line and IXC1100 control plane processors are configured as the PCI host. The PCI Base Address Registers must be initialized by an external PCI device when the IXP42X product line and IXC1100 control plane processors are configured as a PCI option.

The PCI Base Address Registers — along with any other registers in the PCI Configuration Space — will be accessed by the Intel XScale processor using three Configuration and Status Registers:

- PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register
- PCI Configuration Port Write Data (PCI_CRP_WDATA) Register
- PCI Configuration Port Read Data (PCI_CRP_RDATA) Register.

The IXP42X product line and IXC1100 control plane processors are a single-function, Type 0 Configuration space when functioning as a PCI option. For detailed information on the values to program the PCI Controller Configuration and Status Registers, see the *PCI Local Bus Specification*, Rev. 2.2.

The PCI Configuration Port Write Data (PCI_CRP_WDATA) Register is a 32-bit register that is used to place the data that is to be written into the PCI Configuration Space. The PCI Configuration Port Read Data (PCI_CRP_RDATA) Register is a 32-bit register that is used to capture the data that is returned from the PCI Configuration Space. The PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register is a register that provides the address, byte enables, and control for the read and write access to the PCI Configuration Space from the internal side of the IXP42X product line and IXC1100 control plane processors.

- Bits 23:20 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the byte enables for the access to the PCI Configuration Space

  These bits directly correspond to the four - byte field associated with the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register. Table 100 on page 224 shows the mapping of the byte enables of the PCI Configuration Port Address/ Command/Byte Enables (PCI_CRP_AD_CBE) Register to the byte lane fields of the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register.

- Bits 7:2 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the address for the register access within the 64 32-bit Word PCI Configuration Space.

  The 64 32-bit Word PCI Configuration Space is shown in Table 101 on page 225.

- Bits 19:16 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the command to execute on the PCI Configuration Space. The only two commands currently defined are read and write.

  Table 102 on page 225 shows valid command codes for accessing the PCI Configuration Space. When a read command is written into the command field of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register along with the appropriate address of the PCI Configuration register to be accessed, the data from the address requested will be returned to the PCI Configuration Port Read Data (PCI_CRP_RDATA) Register.

  A master on the AHB bus can then read the PCI Configuration Port Read Data (PCI_CRP_RDATA) Register. For example:

1. PCI_CRP_AD_CBE is written with hexadecimal 0x00300004, which causes the contents of the PCI Control Register/Status Register (PCI_SRCR) to be written into the PCI_CRP_RDATA register.

   Note that bits 23:20 are set to hexadecimal 3. For read accesses, byte-enables are ignored. Bits 19:16 are set to hexadecimal 0, which denotes a read command. Bits 7:0 are set to hexadecimal 04.

2. PCI_CRP_RDATA is read by the AHB master that requested the PCI_SRCR to be returned to the PCI_CRP_RDATA register.

When a write to the PCI Configuration Space is desired, the AHB master requesting the write must update the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register with the data that is to be written to the PCI Configuration Register. Once the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register has been updated, a write command is written into the command field of the PCI Configuration Port Address/ Command/Byte Enables (PCI_CRP_AD_CBE) Register along with the appropriate byte enables and address of the PCI Configuration register to be accesses.

The data contained in the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register will be written to the PCI Configuration Register specified by the address and byte enables contained in the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register. For Example:

1. An AHB master that wants to write a particular PCI Configuration Register writes PCI_CRP_AD_CBE register first. Assume that the AHB master wants to write a hexadecimal value of 0x85008086 to the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register. The PCI_CRP_AD_CBE register is written with a hexadecimal 0x00010040.

   Note that bits 23:20 are set to hexadecimal 0. For write accesses byte enables are active low. Bits 19:16 are set to hexadecimal 1, which denotes a write command. Bits 7:0 are set to hexadecimal 40, which addresses the PCI_RTOTTO register.

2. Next, the hexadecimal value of 0x85008086 is written to the PCI Configuration Register PCI_CRP_WDATA register, which causes the contents of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register to be written with a hexadecimal value of 0x85008086.

One more example will demonstrate the effects of the byte-enables on write accesses to the PCI Configuration Space:

1. Assume that the objective is to update the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8) without updating the TRDY terminal count value of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 7:0). The Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register is located at hexadecimal address 0x40. Also assume the value currently contained in the Retry Timeout/ TRDY Timeout (PCI_RTOTTO) Register is a hexadecimal 0x00008080.

2. The PCI_CRP_AD_CBE is written with hexadecimal 0x00D10040.

   Note that bits 23:20 are set to hexadecimal D. For write accesses this allows only byte 1 to be written (bits 15:8). Bits 19:16 are set to hexadecimal 1, which denotes a write command. Bits 7:0 are set to hexadecimal 40, which addresses the PCI_RTOTTO register.

3. Assume that the AHB master wants to write a hexadecimal value of 0x0000AB00 to the second byte of the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8). The PCI_CRP_WDATA will be loaded with a value of 0x0000AB00, which causes the contents of the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8) to be written with a hexadecimal value of 0x0000AB00. The value that is now contained within the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register is 0x0000AB80. Notice that only one byte of data was manipulated.

Table 100 shows the PCI Byte Enables Byte Lane Mapping (accesses to the PCI Configuration Space from within the IXP42X product line and IXC1100 control plane processors) when using the CRP access mechanism.

Table 100. PCI Byte Enables Using CRP Access Method

PCI_CRP_AD_CBE(23:20)	PCI_CRP_WDATA (31:24)	PCI_CRP_WDATA (24:16)	PCI_CRP_WDATA (15:8)	PCI_CRP_WDATA (7:0)
0000	X	X	X	X
0001	X	X	X	
0010	X	X		X
0011	X	X		
0100	X		X	X
0101	X		X	
0110	X			X
0111	X			
1000		X	X	X
1001		X	X	

Table 100. PCI Byte Enables Using CRP Access Method

PCI_CRP_AD_CBE(23:20)	PCI_CRP_WDATA (31:24)	PCI_CRP_WDATA (24:16)	PCI_CRP_WDATA (15:8)	PCI_CRP_WDATA (7:0)
1010		X		X
1011		X		
1100			X	X
1101			X	
1110				X
1111				

Table 101. PCI Configuration Space

Offset	Register Name	Description
0x00	PCI_DIDVID	Device ID/Vendor ID
0x04	PCI_SRCR	Status Register/Control Register
0x08	PCI_CCRID	Class Code/Revision ID
0x0C	PCI_BHLC	BIST/Header Type/Latency Timer/Cache Line
0x10	PCI_BAR0	Base Address 0
0x14	PCI_BAR1	Base Address 1
0x18	PCI_BAR2	Base Address 2
0x1C	PCI_BAR3	Base Address 3
0x20	PCI_BAR4	Base Address 4
0x24	PCI_BAR5	Base Address 5
0x28	RESERVED	(Reserved)
0x2c	PCI_SIDSVID	Subsystem ID/Subsystem Vendor ID
0x30-38	RESERVED	(Reserved)
0x3C	PCI_LATINT	Defines Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line
0x40	PCI_RTOTTO	Defines retry timeout and trdy timeout parameters

Table 102. Command Type for PCI Controller Configuration and Status Register Accesses

Command Value pci_crp_ad_cbe [19:16]	Command Type	Description
0x0	Read	Initiates a read of the PCI Controller Configuration and Status Register Accesses
0x1	Write	Initiates a write to the PCI Controller Configuration and Status Register Accesses
0x2 – 0xF	(Reserved)	Reserved for future use. Use of these values produce unpredictable results.

6.5 PCI Controller South AHB Transactions

The PCI Controller provides access to internal functionality within the IXP42X product line and IXC1100 control plane processors. The PCI Controller provides access to the South AHB through the AHB Target Interface and the AHB Master Interface. The AHB

Target Interface is used to accept transaction request from other AHB Masters. The AHB Master Interface is used to initiate transaction requests to other AHB Targets. The two DMA channels as well as the PCI Target Interface use the AHB Master Interface.

The AHB Target Interface can accept 8-bit (1 Byte) transactions, 16-bit transactions, and 32-bit transactions. Due to the South AHB not using byte enables, all 16-bit transactions to the PCI Controller AHB Target Interface must be implemented as consecutive-byte addresses. Inability to do this will result in multiple byte wide transactions.

The AHB Master interface will initiate 8-bit (1 Byte) transactions and 32-bit (word) transactions only. The DMA engines will initiate only 32-bit transactions. PCI Target Interface initiated transactions will be 32-bit transactions. Sub 32-bit transactions — initiated by the PCI Target Interface — will be implemented as multiple 8-bit transactions initiated by the PCI Controller AHB Master on the AHB. For information on prioritization of the three functional blocks that use the PCI Controller AHB Master Interface, see “PCI Controller Functioning as Bus Initiator” on page 226.

6.6 PCI Controller Functioning as Bus Initiator

The IXP42X product line and IXC1100 control plane processors can be used to initiate PCI transactions in one of three ways:

- Using the Non-Pre-fetch Registers — as described in section “PCI Controller Configured as Host” on page 213

  The Non-Pre-fetch Registers allow various single 32-bit word PCI Cycles to be produced. The Non-Pre-fetch Registers can be used to initiate Type 0 Configuration Cycles, Type 1 Configuration Cycles, Memory Cycles, I/O Cycles, and Special Cycles.
- Writing to the PCI Memory Cycle Address Space located between AHB address 0x48000000 and 0x4BFFFFFF as described in section “Initializing PCI Controller Configuration and Status Registers for Data Transactions” on page 219
- Using the PCI Controller DMA channels — as described in “PCI Controller DMA Controller” on page 234

The remainder of the section shows example of each cycle type that may be initiated.

The details in this section are provided to understand some functional aspects of the PCI Controller on the IXP42X product line and IXC1100 control plane processors. For complete details please refer to the *PCI Local Bus Specification*, Rev. 2.2.

6.6.1 PCI Byte Enables

I/O reads and memory-cycle writes drive individual byte enables. However, it is important to note that the PCI controller drives *all* byte enables low (asserted) during a memory cycle read of non-prefetch memory.

If an external PCI device has non-prefetch memory and requires either a 16-bit or 8-bit read, there is a possibility that the device will not respond correctly to IXP42X product line and IXC1100 control plane processors memory reads. This is because the IXP42X product line and IXC1100 control plane processors always perform a 32-bit read to the non-prefetch memory region specified in register PCI_NP_AD.

The 8-bit or 16-bit external device should respond with a “target abort,” as per the PCI 2.2 specification, if a 32-bit read is performed to its non-prefetch memory and it requires a 16-bit or 8-bit read.

The IXP42X product line and IXC1100 control plane processors will drive all the byte enables asserted during all memory cycle reads of the external PCI device, no matter what the PCI_NP_CBE register contains in the byte enable bits.

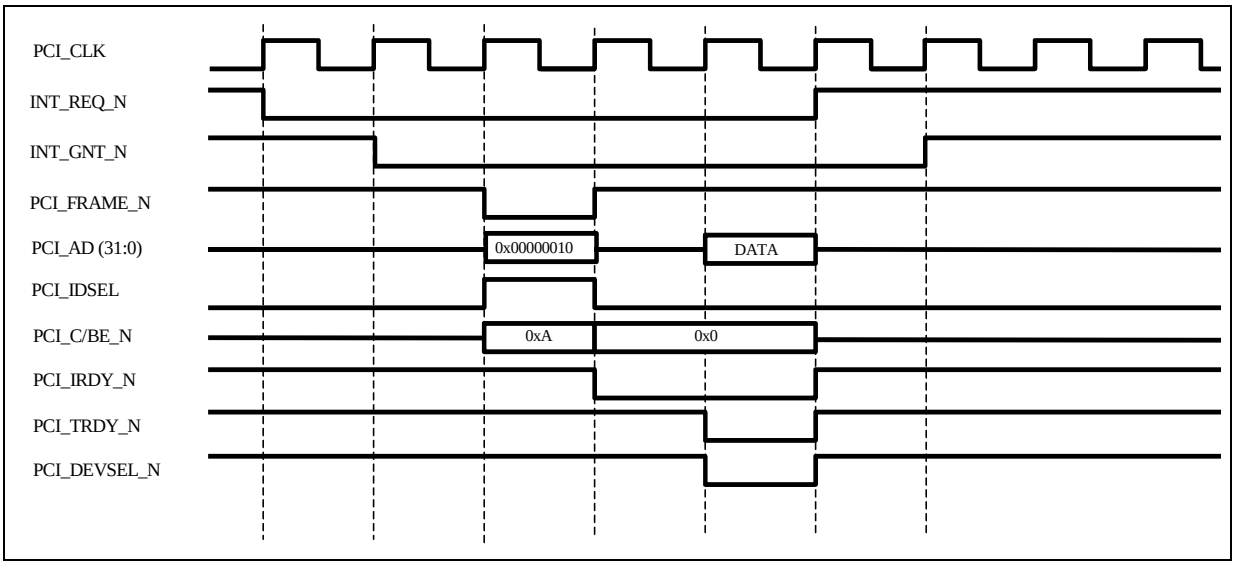
To read non-prefetch memory sub-DWORDS (8-bit or 16-bit), use I/O reads. If it is necessary to use memory cycle reads of sub-DWORDS, a hardware resolution may be required. Contact your Intel field application engineer if you require a hardware resolution.

6.6.2 Initiated Type-0 Read Transaction

The following transaction is a PCI Configuration Read Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The Configuration transaction is initiated to the local PCI bus segment, Device number (chosen by IDSEL), Function 0, and Base Address Register 0.

The IDSEL signal is left up to the user to determine how to drive this signal. It may be driven from one of the upper address signals on the PCI_AD bus. A hexadecimal value of 0xA, written on the PCI_C/BE_N bus during the PCI Bus address phase, signifies that this is a PCI Bus Configuration Read Cycle.

Figure 35. Initiated PCI TYPE 0 Configuration Read Cycle

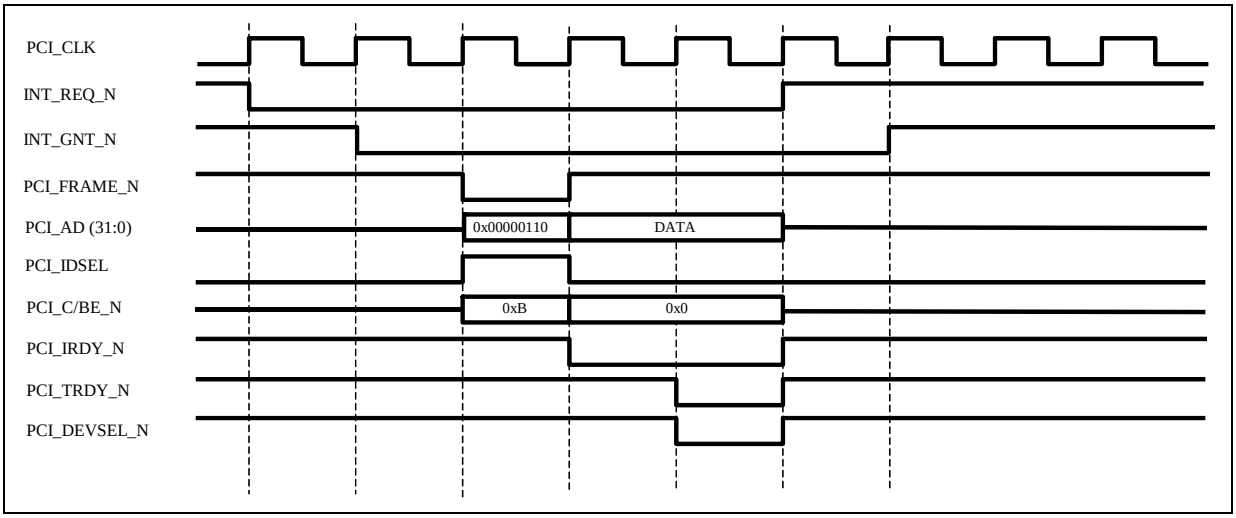


6.6.3 Initiated Type-0 Write Transaction

The following transaction is a PCI Configuration Write Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to the local PCI bus segment, Device number (chosen by IDSEL), Function number 2, and Base Address Register 0.

The IDSEL signal is left up to the user to determine how to drive this signal. It may be driven from one of the upper address signals on the PCI_AD bus. A hexadecimal value of 0xB written on the PCI_C/BE_N bus — during the address phase — signifies that this is a PCI Bus Configuration Write Cycle.

Figure 36. Initiated PCI Type-0 Configuration Write Cycle



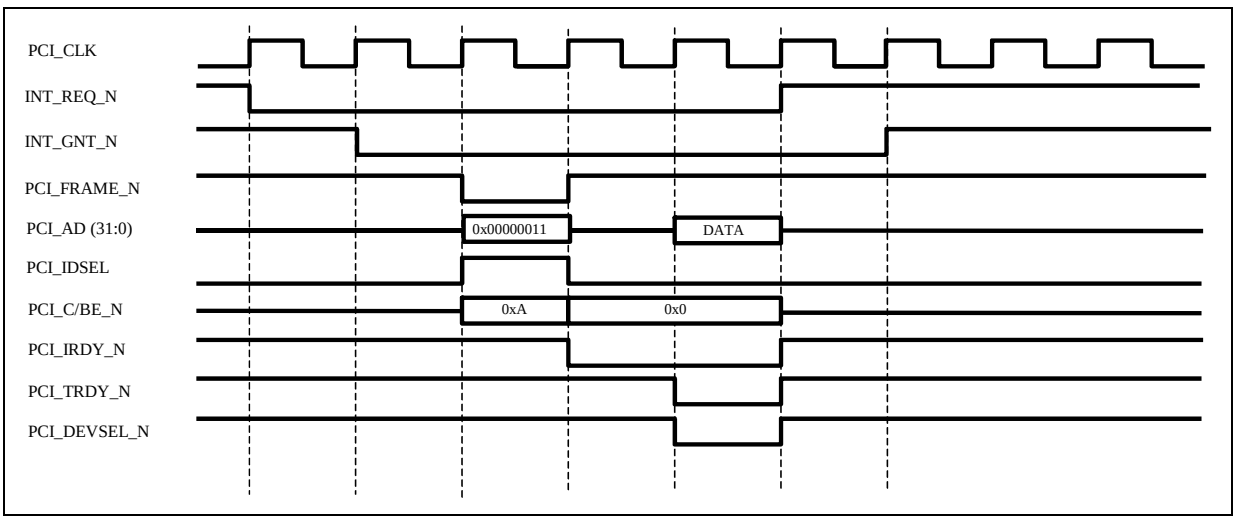
6.6.4 Initiated Type-1 Read Transaction

The following transaction is a PCI Configuration Read Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to PCI bus segment 0, Device number 0, Function 0, and Base Address Register 0.

This configuration cycle is a Type 1 configuration cycle and is intended for another PCI bus segment. Binary 01 being located in bits 1:0 of the PCI_AD bus during the address phase denotes a Type 1 PCI Configuration cycle.

A hexadecimal value of 0xA — written on the PCI_C/BE_N bus during the address phase — signifies that this is a PCI Bus Configuration Read Cycle. Due to the fact that the access is on another PCI Bus Segment, the PCI_TRDY_N signal may take longer to respond and therefore may be extended by several clocks and is not shown here.

Figure 37. Initiated PCI Type-1 Configuration Read Cycle



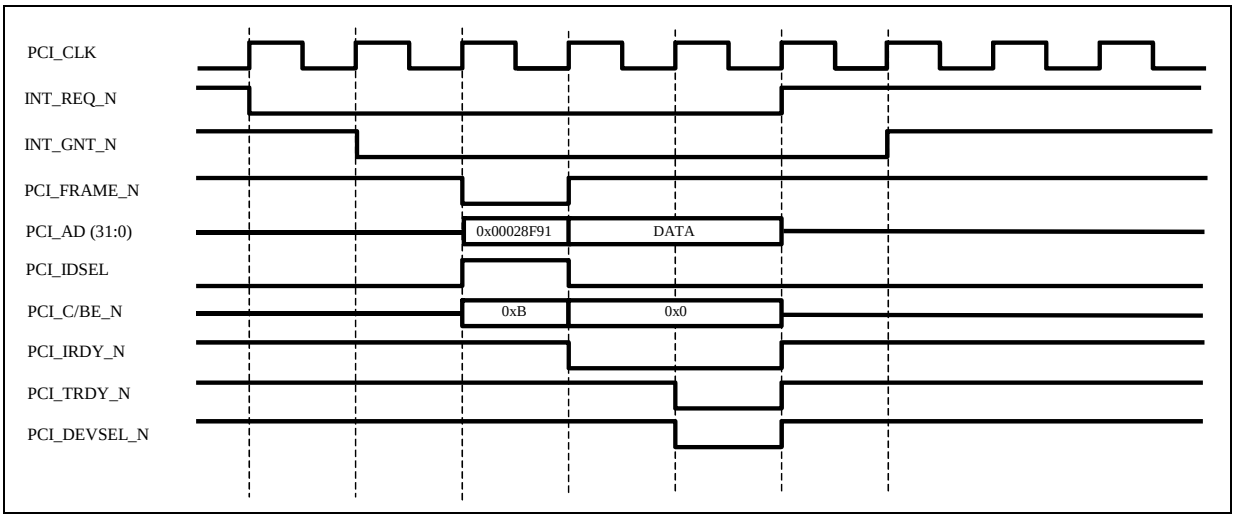
6.6.5 Initiated Type-1 Write Transaction

The following transaction is a PCI Configuration Write working-site Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to PCI bus segment 5, Device number 3, Function number 7, and Base Address Register 0.

This configuration cycle is a Type 1 configuration cycle and is intended for another PCI bus segment. Binary 01 — being located in bits 1:0 of the PCI_AD bus during the address phase — denotes a Type 1 PCI Configuration cycle. A hexadecimal value of 0xB — written on the PCI_C/BE_N bus during the address phase — signifies that this is a PCI Bus Configuration Write Cycle.

Due to the fact that the access is on another PCI Bus Segment, the PCI_TRDY_N signal may take longer to respond and therefore may be extended by several clocks and is not shown here.

Figure 38. Initiated PCI Type-1 Configuration Write Cycle

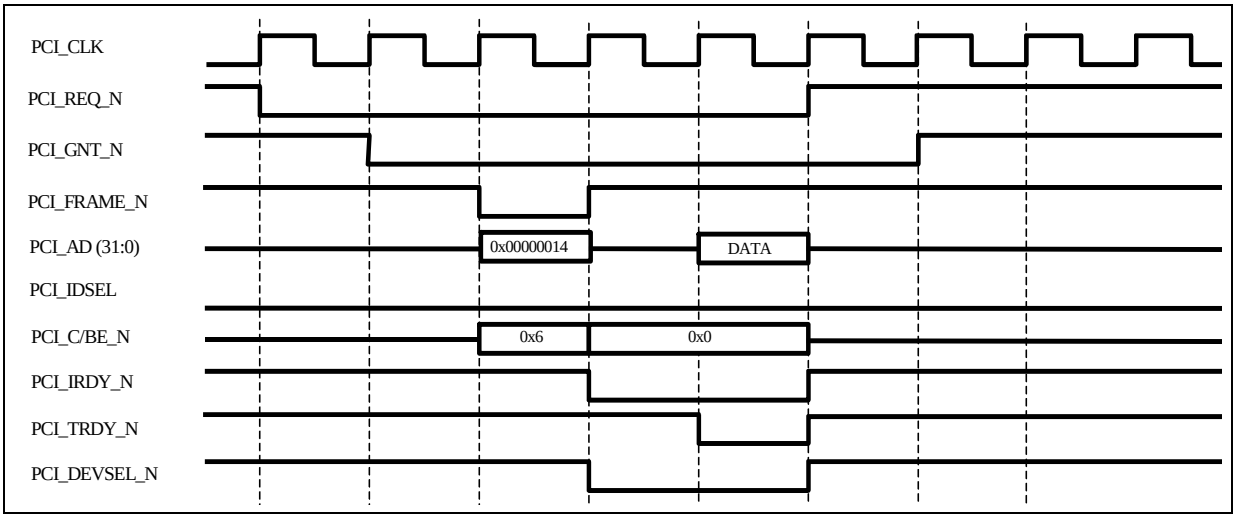


6.6.6 Initiated Memory Read Transaction

The following transaction is a PCI Memory Read Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear increment transfer type.

A hexadecimal value of 0x6 — written on the PCI_C/BE_N bus during the address phase — signifies that this is a PCI Bus Memory Read Cycle. All byte enables are asserted for the transaction.

Figure 39. Initiated PCI Memory Read Cycle

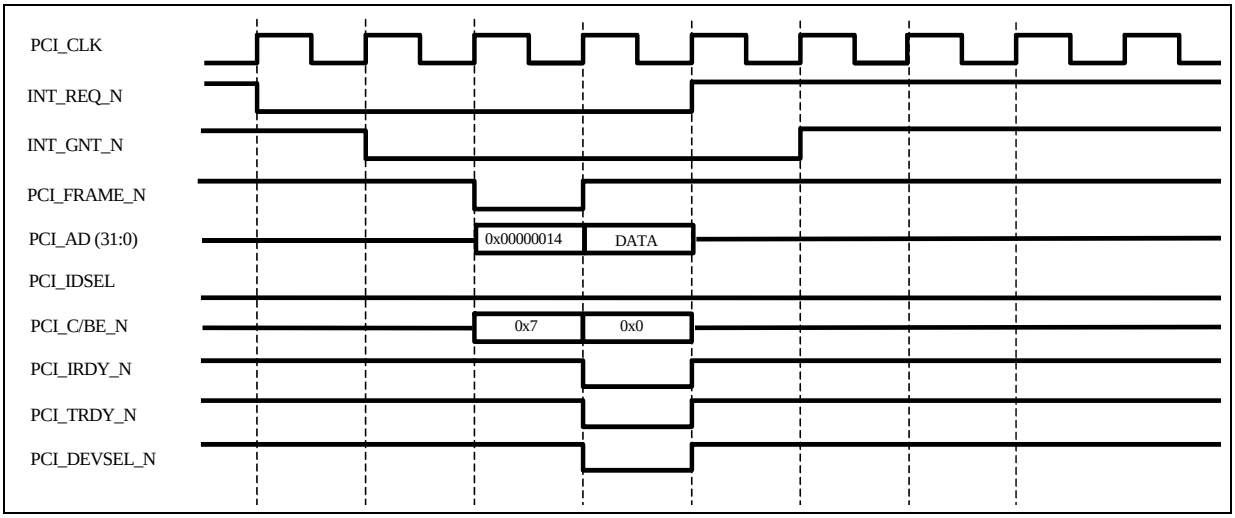


6.6.7 Initiated Memory Write Transaction

The following transaction is a PCI Memory Write Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear-increment transfer type. A hexadecimal value of 0x7 — written on the PCI_C/BE_N bus during the address phase — signifies that this is a PCI Bus Memory Read Cycle. All byte enables are asserted for the transaction.

Notice that on this transaction the PCI_DEVSEL_N signal timing is different. This signal-timing differential is due to the fact that the DEVSEL_N signal must become active within the first three clocks after the FRAME_N becoming active. This requirement could be different for every device that is on the PCI Bus. There is also no relationship to when TRDY_N becomes active other than the TRDY_N signal must not become active prior to the DEVSEL_N signal becoming active.

Figure 40. Initiated PCI Memory Write Cycle

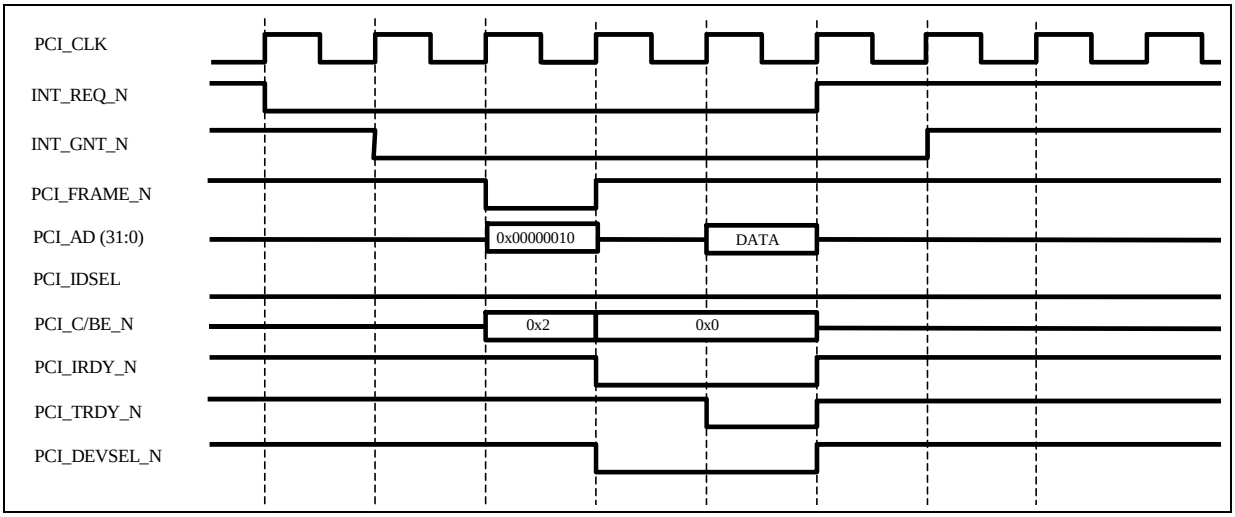


6.6.8 Initiated I/O Read Transaction

The following transaction is a PCI I/O Read Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to address location hexadecimal 0x00000010.

A hexadecimal value of 0x2 — written on the PCI_C/BE_N bus during the address phase — signifies that this is a PCI Bus I/O Read Cycle.

Figure 41. Initiated PCI I/O Read Cycle



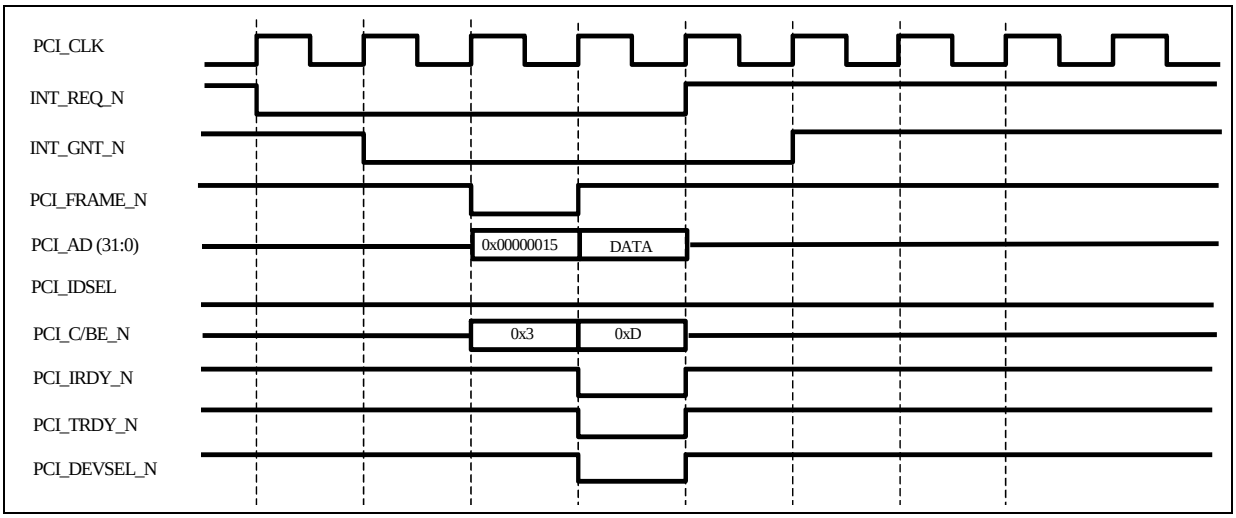
6.6.9 Initiated I/O Write Transaction

The following transaction is a PCI I/O Write Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller

implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to address location hexadecimal 0x00000015. The value of binary 01 in PCI_AD (1:0) indicates that the transfer is a valid byte address of the first byte of 32-bit word address 0x00000014 (0x00000014 + 0x00000001 = 0x00000015).

The byte-enables being 0xD — during the data transfer — signify that the transfer is a byte transfer to the above-mentioned address. A hexadecimal value of 0x3 written on the PCI_C/BE_N bus during the address phase signifies that this is a PCI Bus I/O Write Cycle.

Figure 42. Initiated PCI I/O Write Cycle



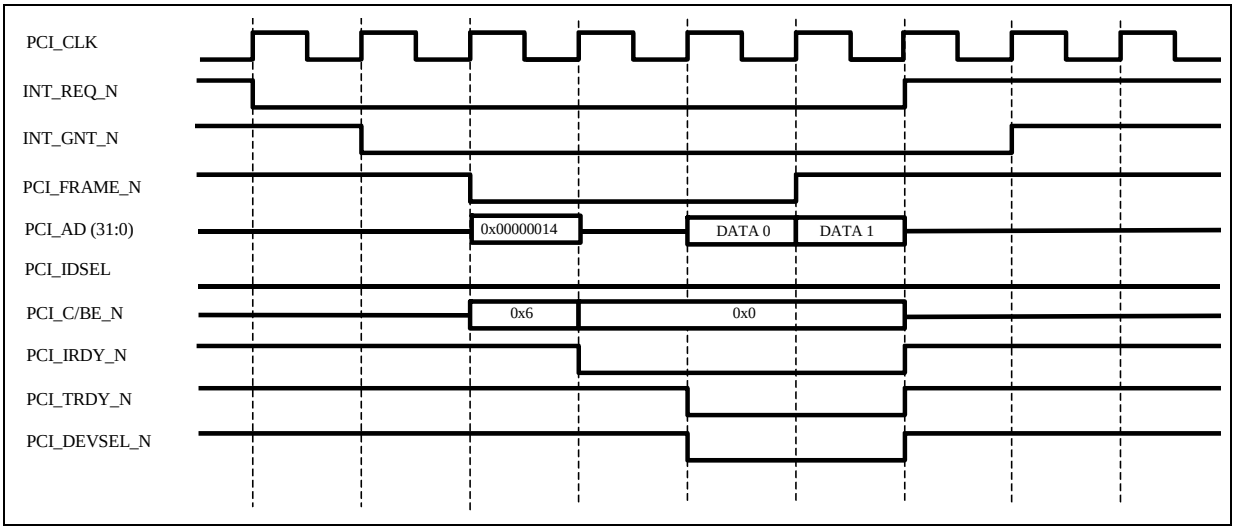
6.6.10 Initiated Burst Memory Read Transaction

The following transaction is a two word bursting PCI Memory Read Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to initial address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear increment transfer type. The second data word transferred will be from address hexadecimal 0x00000018.

A hexadecimal value of 0x6 — written on the PCI_C/BE_N bus during the address phase — signifies that this is a PCI Bus Memory Read Cycle. All byte-enables are active for the transaction.

A maximum burst length of eight 32-bit words is supported for initiated Memory Cycle transactions from the IXP42X product line and IXC1100 control plane processors.

Figure 43. Initiated PCI Burst Memory Read Cycle



6.6.11 Initiated Burst Memory Write Transaction

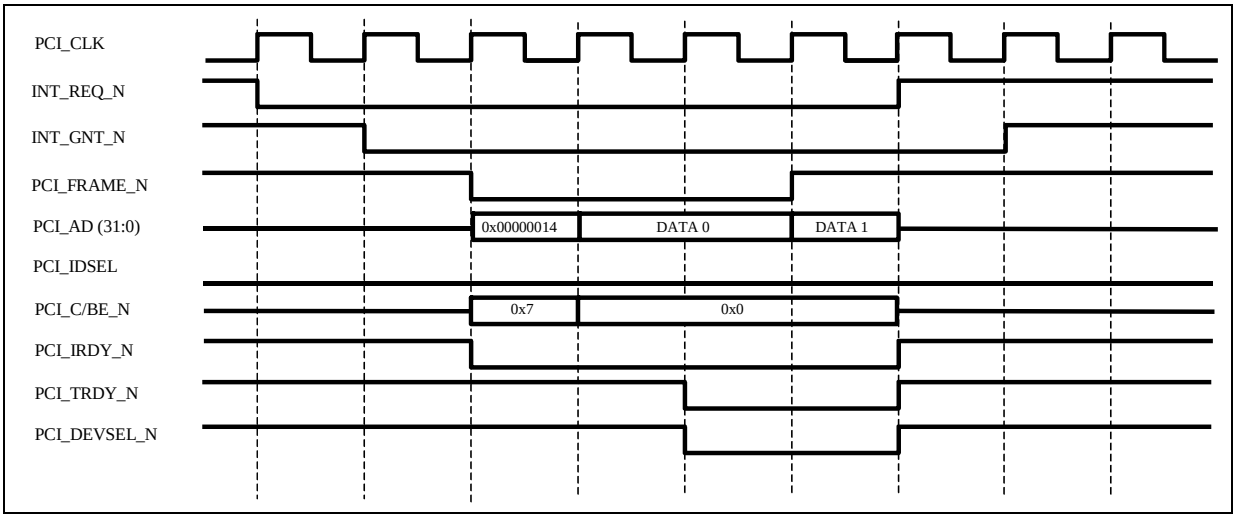
The following transaction is a two word bursting PCI Memory Write Cycle initiated from the IXP42X product line and IXC1100 control plane processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP42X product line and IXC1100 control plane processors. The transaction is initiated to initial address location hexadecimal 0x00000014.

The value of binary 00 in PCI_AD (1:0) indicates that this is an linear increment transfer type. The second data word transferred will be from address hexadecimal 0x00000018.

A hexadecimal value of 0x7 — written on the PCI_C/BE_N bus during the address phase — signifies that this is a PCI Bus Memory Write Cycle. All byte-enables are active for the transaction.

A maximum burst length of eight 32-bit words is supported for initiated Memory Cycle transactions from the IXP42X product line and IXC1100 control plane processors.

Figure 44. Initiated PCI Burst Memory Write Cycle



6.7 PCI Controller Functioning as Bus Target

The IXP42X product line and IXC1100 control plane processors can be the target of PCI transactions. Operating as a PCI target, the PCI bus can accept Memory Cycles, I/O Cycles, or Configuration Cycles. Target transactions can take place independent of the Host/Option configuration of the IXP42X product line and IXC1100 control plane processors. Please refer to Table 97, “PCI Target Interface Supported Commands” on page 211 for additional information on supported commands.

Only Type 0 Configuration Cycles are supported.

Timing diagrams are not shown for the target transactions because they are similar to initiated transactions. The only differences are the PCI devices that source/sink the various PCI signals.

For target-read transactions, a retry will be issued upon the IXP42X product line and IXC1100 control plane processors receiving a request to transfer data. Between the time that the retry occurs and the access to the IXP42X product line and IXC1100 control plane processors reoccurs, the PCI Controller on the IXP42X product line and IXC1100 control plane processors retrieve the data from the previously requested location.

For additional details, see the *PCI Local Bus Specification*, Rev. 2.2.

6.8 PCI Controller DMA Controller

The IXP42X product line and IXC1100 control plane processors contain two channels that can be used for DMA (Direct Memory Accesses) to/from the PCI bus and the AHB. The DMA Controller function provides two channels of DMA capability to off load, from the Intel XScale processor, large data transfers between the PCI bus and AHB.

The DMA channels are unidirectional: one DMA channel is used for PCI-to-AHB transfers and one DMA channel is used for AHB-to-PCI transfers. The DMA transfers are implemented using three of the PCI Controller Configuration and Status Registers to specify the PCI address, the AHB address, and the transfer length. Each DMA channel has two sets of three registers to provide buffering for consecutive transfers.

For each direction, when a DMA channel is executing one transfer using the active DMA register set, the other DMA register set can be set-up by the Intel XScale processor to specify the next transfer. Both DMA channels can run concurrently so that individual PCI-to-AHB transfers and AHB-to-PCI transfers that make up the DMA transfers are interleaved on the AHB and PCI bus.

Individual DMA-complete and DMA-error status indication is provided for each channel using the DMA Control Register (PCI_DMACTRL) with an interrupt that may be optionally generated in each case.

The register sets associated with the DMA channels are as follows:

1. PCI to AHB Transfers
   a. Register Set 0
   — PCI to AHB DMA AHB Address Register 0 (PCI_PTADMA0_AHBADDR)
   — PCI to AHB DMA PCI Address Register 0 (PCI_PTADMA0_PCIADDR)
   — PCI to AHB DMA Length Register 0 (PCI_PTADMA0_LENGTH)
   b. Register Set 1
   — PCI to AHB DMA AHB Address Register 1 (PCI_PTADMA1_AHBADDR)
   — PCI to AHB DMA PCI Address Register 1 (PCI_PTADMA1_PCIADDR)
   — PCI to AHB DMA Length Register 1 (PCI_PTADMA1_LENGTH)
2. AHB to PCI Transfers
   a. Register Set 0
   — AHB to PCI DMA AHB Address Register 0 (PCI_ATPDMA0_AHBADDR)
   — AHB to PCI DMA PCI Address Register 0 (PCI_ATPDMA0_PCIADDR)
   — AHB to PCI DMA Length Register 0 (PCI_ATPDMA0_LENGTH)
   b. Register Set 1
   — AHB to PCI DMA AHB Address Register 1 (PCI_ATPDMA1_AHBADDR)
   — AHB to PCI DMA PCI Address Register 1 (PCI_ATPDMA1_PCIADDR)
   — AHB to PCI DMA Length Register 1 (PCI_ATPDMA1_LENGTH)

The PCI Address Registers described above are used to specify the beginning 32-bit word address for the PCI side of the DMA transfers. The AHB Address Registers, described above, are used to specify the beginning 32-bit word address for the AHB side of the DMA transfers. The least significant two bits of both addresses are hard-wired to logic 0. Thus, all transfers are word-aligned.

The Length Registers are used for three purposes:

- Sixteen bits to define a word count
  Bits 15:0 of the Length Registers define the word count.
- One bit to enable the DMA transfer
  Bit 31 of the Length Registers enables the DMA transfer to execute. When bit 31 of the Length Registers is set to logic 1, the DMA transfer executes until a word count of zero is reached. When the word count reaches zero and bit 31 of the Length Registers is set to logic 1, bit 31 of the Length Register is cleared to logic 0. When bit 31 of the Length Register is set to logic 0, the register set associated with the DMA channel is disabled. The second register set may be active and using the DMA channel when the first DMA has finished.
- One bit to define the byte order of the data transferred.

Bit 28 of the Length Register is used to provide a byte swap on the DMA data as data is transferred from the AHB to the PCI bus or from the PCI Bus to AHB, depending upon the direction of the DMA transfer. When bit 28 is set to logic 1, a byte swap will occur on the DMA data. Figure 45 and Figure 46 demonstrates the DMA transfer byte lane swapping.

Figure 45. AHB to PCI DMA Transfer Byte Lane Swapping

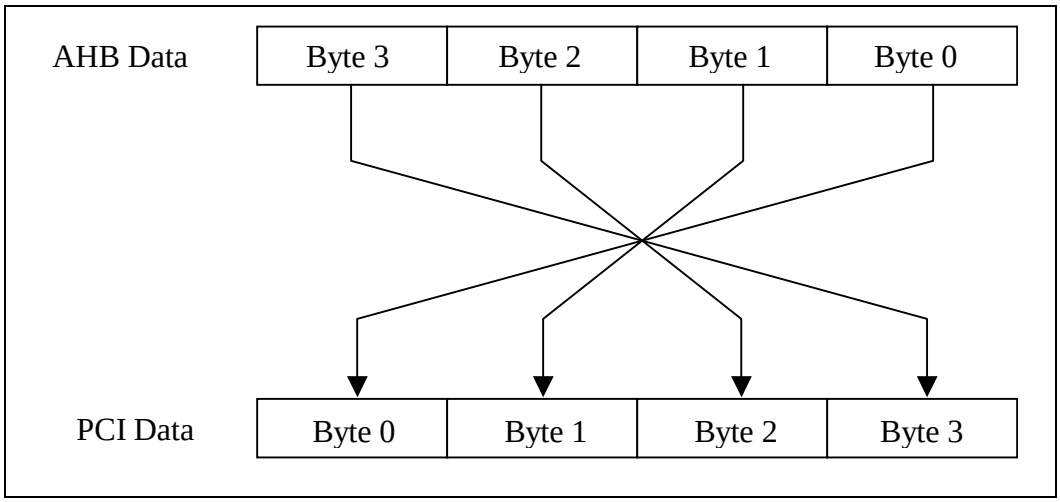
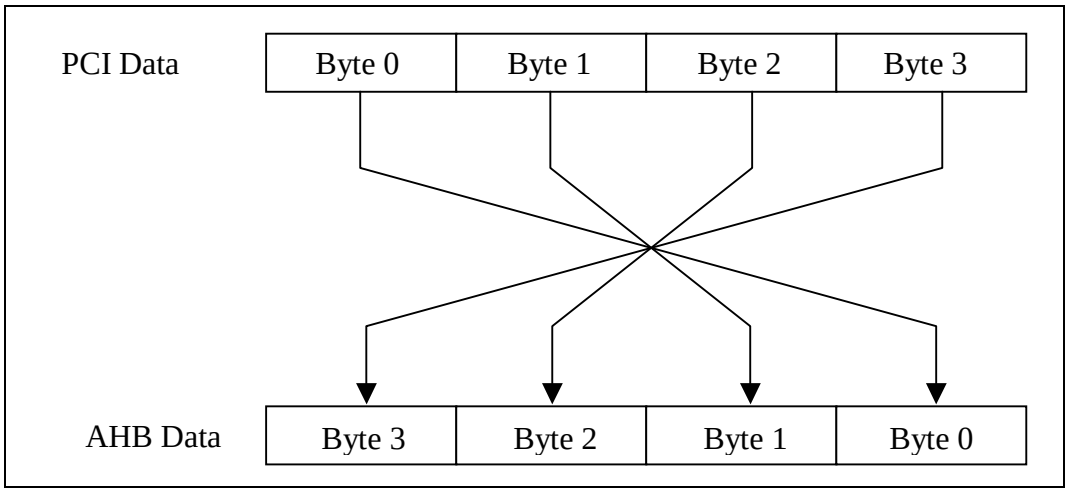


Figure 46. PCI to AHB DMA Transfer Byte Lane Swapping



The DMA channels share resources with the AHB Master and Target interfaces and therefore must arbitrate for these resources. AHB-to-PCI DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Transmit FIFO. PCI-to-AHB DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Receive FIFO. Use of the AHB Master Interface will revolve between the two DMA channels and PCI requests that appear in the Target Receive FIFO.

While a particular channel is accessing the Initiator Request FIFO, accesses to the PCI bus coming in to the AHB Target Interface from the AHB will be retried. The access will be flagged by the hardware to signal the DMA Controller channels that a PCI access is pending (AHB masters must attempt retried transfers until complete). This enables the DMA channels to permit the AHB initiated PCI access to go through to the PCI bus.

Additionally, while the AHB Master Interface is in use by a DMA channel, PCI requests that appear in the Target Receive FIFO are flagged to allow these received requests to gain access of the AHB bus.

Access to the PCI Controller Control and Status Registers from the AHB is unrestricted while the DMA channels are operating. The ability to access the PCI Controller Control and Status Registers is provided to allow the Intel XScale processor to set up the off-line DMA Register set while the on-line DMA Register set is operating.

When the current transfer is complete, the DMA complete bit is set in the DMA Control Register (PCI_DMACTRL) and the channel-enable bit is cleared in the Length Register. If the channel is enabled in the second DMA Length Register, DMA execution starts using the second set of DMA Registers. Transfers continue in this fashion until the channel enable bits in both sets of DMA length registers for each channel are set to logic 0.

Whenever possible, the DMA channels use eight-word burst accesses for the PCI-to-AHB and AHB-to-PCI transfers. Eight-word bursts are the maximum sustained length that the AHB — and the PCI bus — can transfer. Every eight words, the PCI bus will disconnect and reconnect later.

This implementation allows fairness among all devices on the PCI bus. In the general case, a transfer will issue a beginning burst transfer from one to eight words to align the AHB word address of the DMA to an eight-word boundary.

The subsequent transfers will be issued as eight-word bursts until the words remaining to be transferred are eight words or less. The final transfer will complete the DMA with a burst of one to eight words.

The example below demonstrates how to use the DMA channels.

1. The goal is to:
   - Write a 16-word burst to the PCI Bus with no byte-swapping, using the AHB-to-PCI DMA channel
   - Initialize a 16-word burst read from the PCI Bus, using the PCI-to-AHB DMA channel
   - Initialize a six-word burst write to the PCI Bus using the AHB to PCI DMA channel.

     The AHB-to-PCI DMA channel is used to complete PCI Memory Cycle write accesses and the PCI-to-AHB DMA channel is used to complete PCI Memory Cycle read accesses always.
2. Update the AHB to PCI DMA AHB Address Register 0 (PCI_ATPDMA0_AHBADDR) with PCI_ATPDMA0_AHBADDR = 0x00004000 and the AHB to PCI DMA PCI Address Register 0 (PCI_ATPDMA0_PCIADDR) with PCI_ATPDMA0_PCIADDR = 0xFC000004.
3. Update the AHB to PCI DMA AHB Address Register 1 (PCI_ATPDMA1_AHBADDR) with PCI_ATPDMA1_AHBADDR = 0x00004F00 and the AHB to PCI DMA PCI Address Register 1 (PCI_ATPDMA1_PCIADDR) with PCI_ATPDMA1_PCIADDR = 0xA2000004.
4. Update the PCI to AHB DMA AHB Address Register 0 (PCI_PTADMA0_AHBADDR) with PCI_PTADMA0_AHBADDR = 0x00004A00 and the PCI to AHB DMA PCI Address Register 0 (PCI_PTADMA0_PCIADDR) with PCI_PTADMA0_PCIADDR = 0x10000004.
5. Update the AHB to PCI DMA Length Register 0 (PCI_ATPDMA0_LENGTH) with PCI_ATPDMA0_LENGTH = 0x80000010.

   The DMA write transfer to the PCI bus begins.
6. Update the PCI to AHB DMA Length Register 0 (PCI_PTADMA0_LENGTH) with PCI_PTADMA0_LENGTH = 0x80000010.

Assume that this DMA channel is enabled prior to the end of the first eight-word burst of the first write DMA transfer ending. The DMA read transfer to the PCI bus becomes interleaved with the first write transfer. So the first eight words of the read starts towards completion.

7. Update the AHB to PCI DMA Length Register 1 (PCI_ATPDMA1_LENGTH) with PCI_ATPDMA1_LENGTH = 0x90000006.

   Assume this is set while the above read DMA transaction is occurring.

8. The next PCI transfer is completing the last eight words of the initial 16-word write DMA transfer. That is followed by the last eight words of the 16-word read DMA transfer and the execution of the six-word write transfer with the data byte lanes swapped.

6.8.1 AHB to PCI DMA Channel Operation

The AHB-to-PCI (ATP) channel uses the PCI Core Initiator Request and Initiator Transmit FIFOs. The channel reads data from the AHB bus and writes it to a PCI target on word-aligned boundaries.

A DMA transfer from AHB to PCI is processed as follows:

1. An AHB master writes the PCI starting address, AHB starting address, and word count to the PCI_ATPDMA0/1_PCIADDR, PCI_ATPDMA0/1_AHBADDR, PCI_ATPDMA0/1_LENGTH registers respectively. If the channel enable bit is set in the PCI_ATPDMA0/1_LENGTH register, the DMA transfer commences.
2. The DMA Controller signals the AHB Slave Interface to retry all access attempts from the AHB bus and waits for any AHB accesses of the PCI Bus to complete.
3. The DMA Controller signals the AHB Master Interface to stop servicing requests from the PCI bus, and waits for any pending accesses from PCI to complete.
4. When access is obtained, data is read from AHB and loaded into the Initiator Transmit FIFO. A PCI write request is loaded into the Initiator Request FIFO.
5. When the transfer completes on the PCI bus, the DMA address and length registers are updated.
6. Steps 4-6 are repeated until all data is transfered. Once the DMA Controller gets control of the hardware, the DMA channel reads from AHB and writes to PCI 8 words at a time until the transfer is done or 1) an AHB or PCI access is attempted or 2) the other DMA channel has a transfer enabled. In the case of 1) or 2), the DMA channel will release the resources, then go to step 2 where it re-requests access to these resources to continue the transfer. When done, the channel enable bit in the PCI_ATPDMA0/1_LENGTH register is cleared, the DMA complete status bit is set and — if enabled — an interrupt is asserted on PCC_ATPDMA_INT AND/OR PCC_INT. PCC_PTADMA_INT and PCC_INT are signal internal to the IXP42X product line and IXC1100 control plane processors and are routed to the Interrupt Controller.
7. In response to the interrupt, an AHB agent may read the DMA Control register (PCI_DMACTRL) to determine the status of the transfer.

6.8.2 PCI to AHB DMA Channel Operation

The PCI to AHB (PTA) channel uses the PCI Core Initiator Request and Initiator Receive FIFOs. The channel reads data from the PCI bus and writes it to an AHB slave on word-aligned boundaries.

A DMA transfer from PCI to AHB is processed as follows:

1. An AHB master writes the PCI starting address, AHB starting address, and word count to the PCI_PTADMA0/1_PCIADDR, PCI_PTADMA0/1_AHBADDR,

PCI_PTADMA0/1_LENGTH registers respectively. If the channel enable bit is set in the PCI_PTADMA0/1_LENGTH register, the DMA transfer commences.

2. The DMA Controller signals the AHB Slave Interface to retry all access attempts from the AHB bus and waits for any pending AHB accesses of the PCI Bus to complete.
3. The DMA Controller signals the AHB Master Interface to stop servicing requests from the PCI bus, and waits for any pending accesses from PCI to complete
4. The DMA Controller issues a read request to the PCI Core via the Request FIFO.
5. When the PCI data arrives in the Initiator Receive FIFO, the data is written to the AHB Bus.
6. When the transfer completes on the AHB bus, the DMA address and length registers are updated.
7. Steps 4-6 are repeated. Once the DMA Controller gets control of the hardware, the DMA channel reads from the PCI Bus and write to the AHB Bus 8 words at a time until the transfer is done or 1) an AHB or PCI access is attempted or 2) the other DMA channel has a transfer enabled. In the case of 1) or 2) above, the currently active DMA channel will release the resources, then go to step 2 where it re-requests access to these resources to continue the transfer. When done, the channel enable bit in the PCI_PTADMA0/1_LENGTH register is cleared, the DMA complete status bit is set and, if enabled, an interrupt is asserted on PCC_PTADMA_INT and/or PCC_INT. PCC_PTADMA_INT and PCC_INT are signal internal to the IXP42X product line and IXC1100 control plane processors and are routed to the Interrupt Controller.
8. In response to the interrupt, an AHB agent may read the DMA Control register (PCI_DMACTRL) to determine the status of the transfer.

6.9 PCI Controller Door Bell Register

The PCI Controller has two registers that make up the Door Bell register logic on the IXP42X product line and IXC1100 control plane processors. These two registers are the AHB Door Bell (PCI_AHBDOORBELL) register and the PCI Door Bell Register (PCI_PCIDOORBELL).

An external PCI device writes the AHB Door Bell (PCI_AHBDOORBELL) register to generate an interrupt signal to the Intel XScale processor. If the AHB doorbell interrupt is enabled (PCI_INTEN.ADBEN = 1) in the PCI interrupt registers, any bit set to logic 1 in the AHB Door Bell (PCI_AHBDOORBELL) will force the interrupt signal to occur.

The AHB Door Bell (PCI_AHBDOORBELL) register is set from the PCI bus only by writing logic 1 to the register. Writing logic 1 to the set bits from the SOUTH AHB clears bits that are set in the AHB Door Bell (PCI_AHBDOORBELL).

An example of using the AHB Door Bell (PCI_AHBDOORBELL) is as follows:

1. An external PCI device writes logic 1 to a bit or pattern of bits to generate an interrupt to the Intel XScale processor.
2. The AHB agent reads the AHB Door Bell (PCI_AHBDOORBELL) register and writes logic 1(s) to all set bits clear the set bit(s). This in turn de-assert the interrupt generated to the Intel XScale processor.

Using the South AHB, the Intel XScale processor writes the PCI Door Bell Register (PCI_PCIDOORBELL) to generate an interrupt to an external PCI device over the PCI_INTA_N signal. Any bit set to logic 1 in the PCI Door Bell Register (PCI_PCIDOORBELL) will generate the PCI interrupt if the PCI doorbell interrupt is enabled the PCI Interrupt Enable Register (PCI_INTEN).

The PCI Door Bell Register (PCI_PCIDOORBELL) register can only be written by the AHB. The external PCI device must write logic 1 to all set bits in the PCI Door Bell Register (PCI_PCIDOORBELL) in order to clear the bits set by the Intel XScale processor.

An example of using the PCI Door Bell (PCI_PCIDOORBELL) is as follows:

1. The Intel XScale processor writes logic 1 to a bit or pattern of bits in the PCI Door Bell Register (PCI_PCIDOORBELL) to generate an interrupt on the PCI bus using PCI_INTA_N.
2. An external PCI device reads the PCI Door Bell Register (PCI_PCIDOORBELL) and writes logic 1(s) to all set bits to clear the set bit(s). This causes the interrupt that is asserted to de-assert.

6.10 PCI Controller Interrupts

The PCI Controller supports generation of a single PCI interrupt and interrupts to the Intel XScale processor. Complete control of the interrupt sources and enabling is provided using two registers: the PCI Interrupt Status Register (PCI_ISR) and PCI Interrupt Enable Register (PCI_INTEN).

6.10.1 PCI Interrupt Generation

The PCI Door Bell Register (PCI_PCIDOORBELL) is used to generate an interrupt on the PCI Bus using the PCI_INTA_N signal. For more information on the interrupt generation, see “PCI Controller Door Bell Register” on page 239.

The PDB bit — Bit 7 of the PCI Interrupt Enable Register (PCI_INTEN) — is used to enable the external PCI Interrupt. When bit 7 is set to logic 1, the external PCI Interrupt logic is enabled. When bit 7 is set to logic 0 the external PCI Interrupt logic is disabled.

Bit 7 of the PCI Interrupt Status Register (PCI_ISR) displays the status of the external PCI Interrupt. This bit will be set to logic 1 when any of the PCI_PCIDOORBELL bits are set to logic 1.

6.10.2 Internal Interrupt Generation

The PCI Controller employs three signals internal to the IXP42X product line and IXC1100 control plane processors that are sent to the Interrupt Controller to signal the Intel XScale processor at the occurrence of various events:

- PCI Controller Interrupt signal (PCC_INT) — A general-purpose interrupt
- PCI Controller AHB to PCI DMA Interrupt signal (PCC_ATPDMA_INT) — A DMA interrupt
- PCI Controller PCI to AHB DMA Interrupt signal (PCC_PTADMA_INT) — A DMA interrupts.

All interrupts are active high and remain asserted until the Intel XScale processor clears the interrupting source by writing the appropriate Configuration and Status Register bits in the PCI Controller.

The PCI Controller Interrupt signal (PCC_INT) can be asserted when:

- A PCI error occurs
- An AHB error occurs
- A AHB-to-PCI DMA transfer is complete or terminates due to an error
- A PCI-to-AHB DMA transfer is complete or terminates due to an error

- A Doorbell is “pushed” by an external PCI device

The PCI Interrupt Status Register (PCI_ISR) indicates the source(s) of the PCI Controller Interrupt signal (PCC_INT). The PCI Controller Interrupt Enable (PCC_INTEN) Register provides an enable for each of the sources located in the PCI Interrupt Status Register (PCI_ISR).

When a bit is set in PCI Interrupt Status Register (PCI_ISR) and the corresponding bit is enabled in the PCI Interrupt Enable (PCI_INTEN) register, then the PCI Controller Interrupt (PCC_INT) signal will be asserted to the Interrupt Controller and then the Intel XScale processor. The interrupt remains asserted until either the source of the interrupt in the PCI Interrupt Status (PCI_ISR) register is cleared or the enable in the PCI Interrupt Enable (PCI_INEN) Register is cleared.

Clearing an interrupt source may involve clearing bits in other Configuration and Status registers.

The PCI_AHBDOORBELL register is used to generate the doorbell interrupt to an AHB agent. This register is read/write-1-to-set from the PCI bus, and read/write-1-to-clear from the AHB bus. All bits are ORed together to generate the interrupt. The sequence is:

1. An external PCI agent writes a pattern of ones to the PCI_AHBDOORBELL register, setting the corresponding bits in the register and asserting the interrupt to the AHB agent.
2. The AHB agent reads the bit pattern in the doorbell register and writes the same pattern back to clear the bits and de-assert the interrupt.

6.11 PCI Controller Endian Control

The *PCI Local Bus Specification*, Rev. 2.2 defines the byte-addressing convention on the PCI Bus as little endian. Since the byte-addressing convention on the PCI bus is defined as little endian and the convention used on the IXP42X product line and IXC1100 control plane processors’ AHB bus is defined as big-endian, the data passing from the PCI bus and the South AHB may be translated from one endianness to the other endianness.

The endian translation can be done by software implemented on the Intel XScale processor. However, several endianness hardware assist functions have been added to the PCI Controller to help remove the endian-swapping burden from the Intel XScale processor, when 32-bit word transactions are occurring on the PCI bus.

There is a hardware-assist function that provides a 32-bit-word-wide, byte-lane reversal process when the IXP42X product line and IXC1100 control plane processors is being used as a PCI target, when the IXP42X product line and IXC1100 control plane processors are initiating PCI Bus transactions using the PCI Memory Cycle Address Space (AHB address 0x48000000 to 0x4BFFFFFF), and the DMA channels are being used as described previously.

There are three bits of the PCI Controller Control and Status Register (PCI_CSR) that control the steering of addresses between the South AHB and PCI Bus when the IXP42X product line and IXC1100 control plane processors are being used as a PCI target and when the IXP42X product line and IXC1100 control plane processors are initiating PCI Bus transactions using the PCI Memory Cycle Address Space (AHB address 0x48000000 to 0x4BFFFFFF). These three bits are:

- Bit 4 (AHB Big-endian Addressing Mode)
- Bit 3 (PCI Target Transfer Byte Swap)
- Bit 2 (AHB Memory Mapped Byte Swap).

Bit 4 (AHB Big-endian Addressing Mode) defines to the PCI Controller how the data being sent to and from the AHB master and target interfaces are addressed. Figure 47 through Figure 52 shows the various configurations and the values returned from the PCI Controller when the AHB Master and Target Interfaces are configured in both big endian and little endian mode of operation. It is stressed that the PCI Controller will not function properly if bit 4 is set to little-endian address mode (logic 0). Big-endian mode of operation must always be used and this bit should be the first bit set in the PCI Controller initialization process.

As shown in Figure 47, when an external PCI device accesses an AHB address using the PCI Controller Target Interface with the South AHB configured in big-endian mode of operation (Bit 4 (AHB Big-endian Addressing Mode) of the PCI Controller Control and Status Register set to logic 1) and the PCI Target Transfer Byte Swap (Bit 3 of the PCI Controller Control and Status Register set to logic 1), the data passed between the PCI Bus and the AHB will be swapped.

When an external PCI device accesses an AHB address using the PCI Controller Target Interface with the South AHB configured in big-endian mode of operation 4 (Bit 4 (AHB Big-endian Addressing Mode) of the PCI Controller Control and Status Register set to logic 1) and the PCI Target Transfer Byte Swap (Bit 3 of the PCI Controller Control and Status Register) set to logic 0, the data passed between the PCI Bus and the AHB will not be swapped. In the following figures, the PCI bytes are numbered according to the corresponding PCI byte enables and AHB bytes are numbered according to the corresponding byte address on the AHB bus.

For example, a PCI write with byte enable 2 asserted, the PCI Target Transfer Byte Swap (Bit 3 of the PCI Controller Control and Status Register) set to logic 1, and the AHB Big-endian Addressing Mode set to logic 1 will produce an AHB byte write with AHB address bits [1:0] = 10b. When the PCI Target Transfer Byte Swap (Bit 3 of the PCI Controller Control and Status Register) set to logic 0, the PCI write with PCI byte enable 2 asserted will produce an AHB byte write with AHB address bits [1:0] = 01b.

Figure 47. Byte Lane Routing During PCI Target Accesses of the AHB – AHB Configured as a Big-Endian Bus

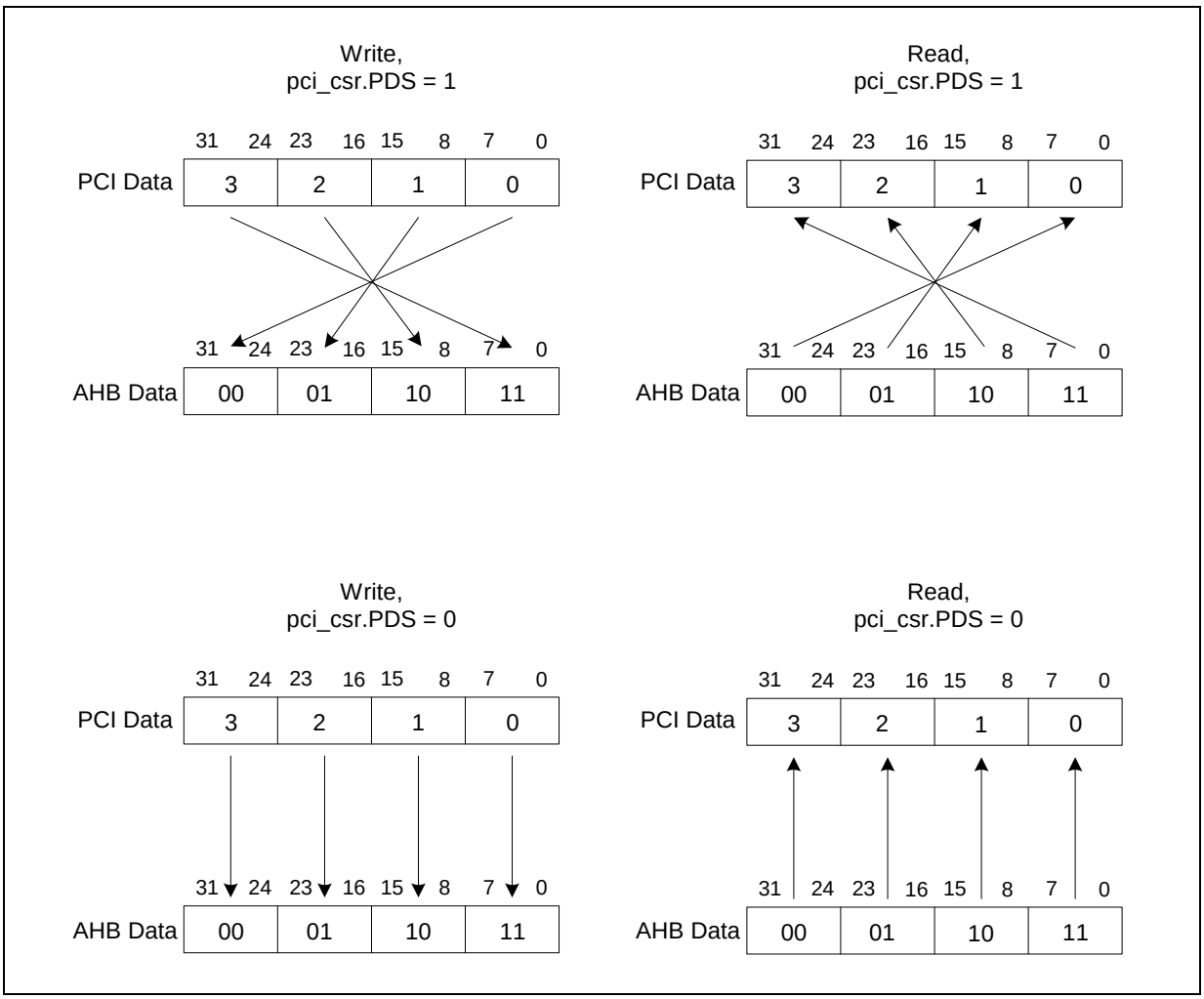
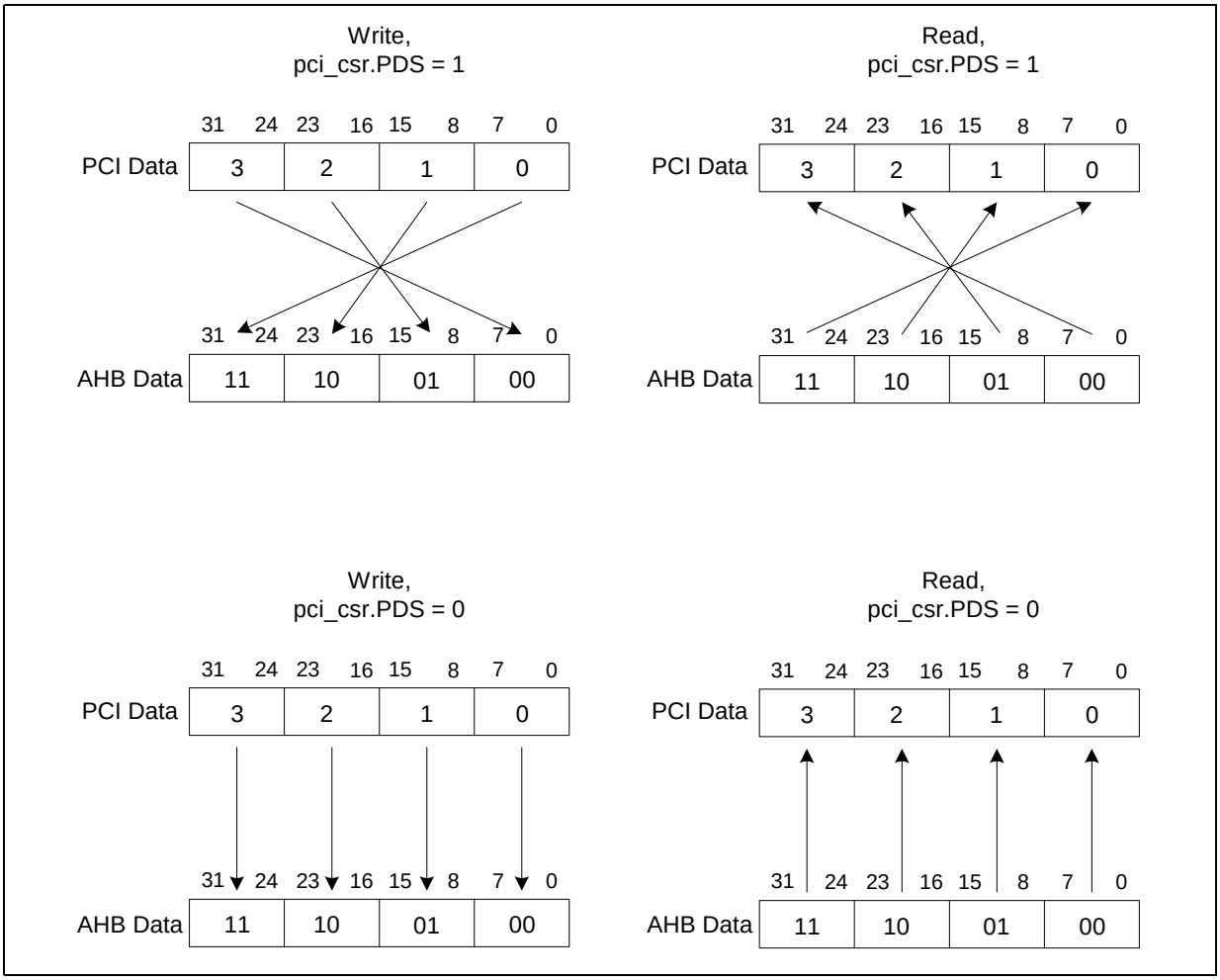


Figure 48. Byte Lane Routing During PCI Target Accesses of the AHB – AHB Configured as a Little-Endian Bus



In a similar fashion — as described for PCI Target accesses directed towards the IXP42X product line and IXC1100 control plane processors — bit 2 (AHB Memory Mapped Byte Swap) of the PCI Controller Control and Status Register (PCI_CSR) controls byte-lane routing when an AHB agent accesses an external PCI device using memory-mapped accesses (0x48000000 to 0x4BFFFFFF). Figure 49 and Figure 50 illustrate the data routing when the AHB is configured as big endian and little-endian modes.

Figure 49. Byte Lane Routing During AHB Memory Mapped Accesses of the PCI Bus – AHB Configured as a Big-Endian Bus

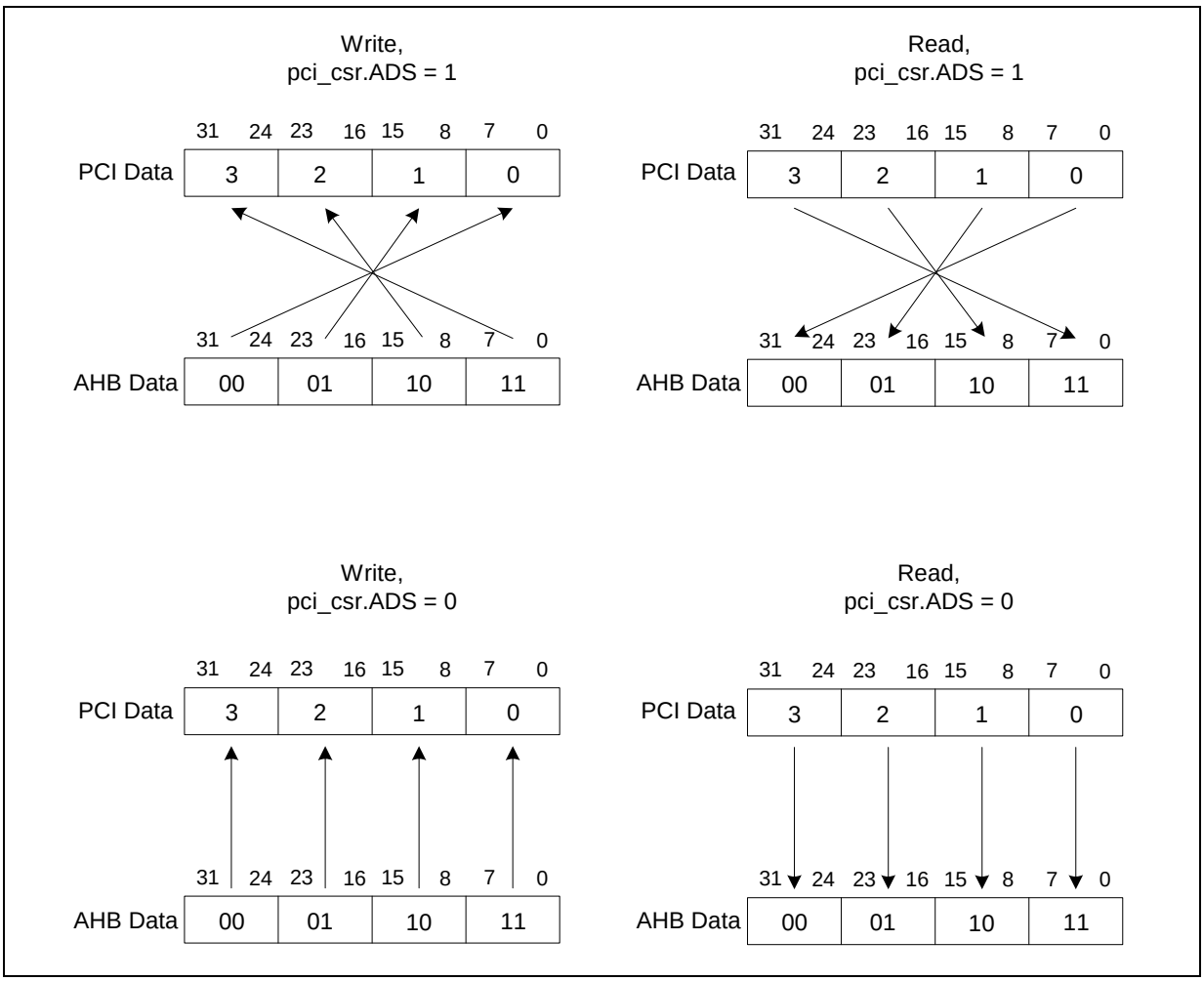
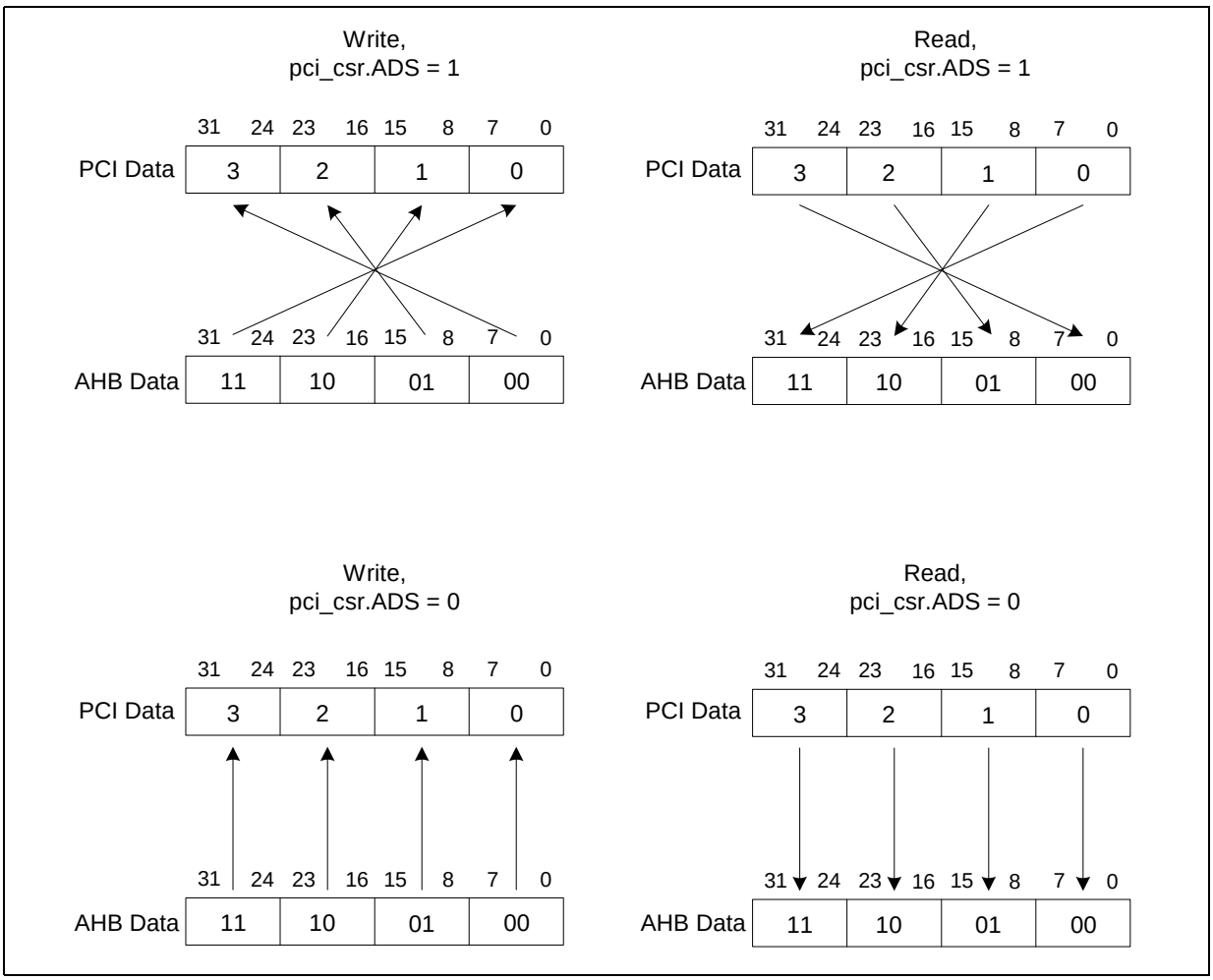
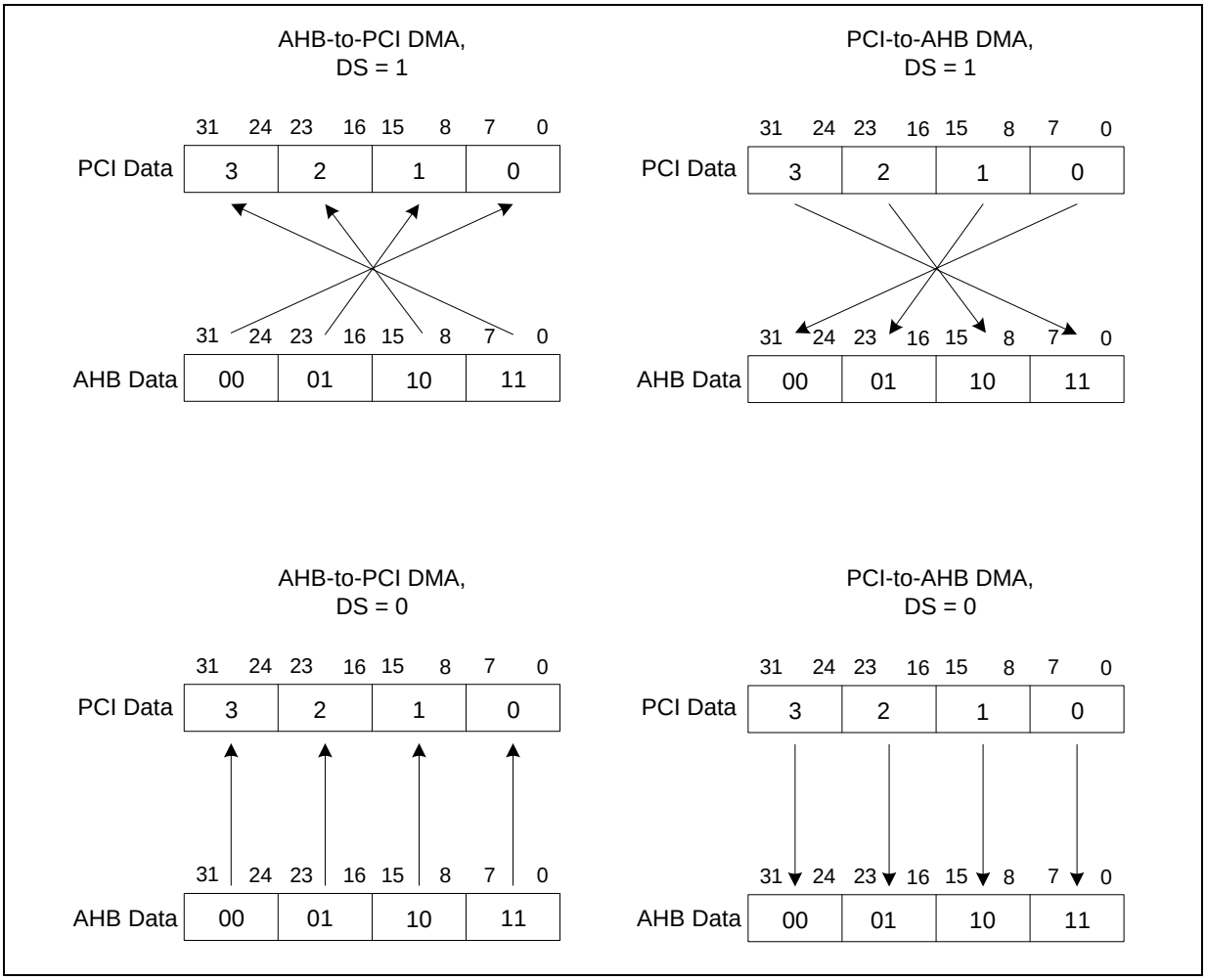


Figure 50. Byte Lane Routing During AHB Memory Mapped Accesses of the PCI Bus – AHB configured as a Little-Endian Bus



As described previously, during DMA transfers the DS bit in the DMA length registers controls byte-lane routing. Figure 51 shows the byte lane routing between the PCI bus and the South AHB during DMA transfers. The DMA channels are word-only accesses.

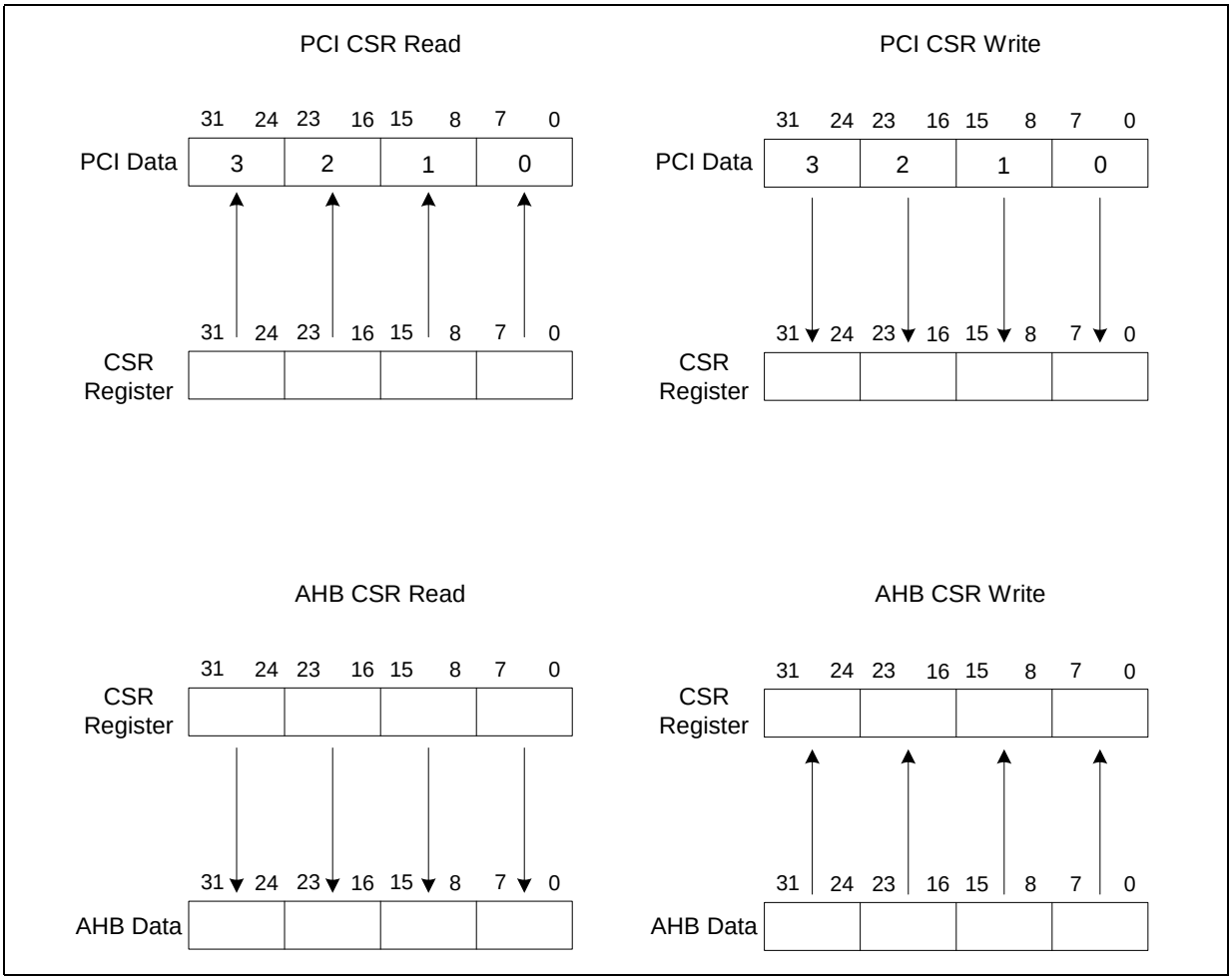
Figure 51. Byte Lane Routing During DMA Transfers



There is no byte-lane reversal process for accesses to PCI Controller Configuration and Status Registers or PCI Configuration Registers. Figure 52 shows the byte-lane routing for these types of accesses.

For example, a PCI Configuration Register write from an external PCI Device, with byte enable 2 asserted, will write to bits 23:16 of the addressed register. An AHB write with Address bits [1:0] = 10b — while being configured in big-endian mode (pci_csr.ABE = 1) of operation — will write bits 15:8 of an addressed PCI Controller Configuration and Status Register. When the AHB is configured in little-endian mode (pci_csr.ABE = 0) of operation, bits 23:16 of the same addressed PCI Controller Configuration and Status Register will be written.

Figure 52. Byte Lane Routing During Configuration and Status Register Accesses



6.12 PCI Controller Clock and Reset Generation

The PCI Reset and PCI clock signals can be provided using general-purpose input/output (GPIO) outputs or from an external source. GPIO 14 can be used to source up to a 33-MHz clock output that can be used as the PCI Clock input to the chip and other PCI Devices.

One of GPIO 13:0 can be used to generate the PCI Reset. Both signals can be sourced from an external device as well. The Intel XScale processor can generate the PCI reset and PCI clock outputs to satisfy the reset timing requirements of the PCI bus.

A PCI startup sequence could be as follows:

1. Power-on reset occurs to the IXP42X product line and IXC1100 control plane processors, the Intel XScale processor starts execution (internal PLL assumed locked and internal clocks stable).
2. Software configures PCI reset and PCI clock GPIOs as outputs driving 0. A pull-down on the GPIO pin chosen to drive the PCI reset signal is required. This pull-down is required because the GPIO are at a tristated value until the device comes completely out of reset and the PCI reset needs to be low from the start.

3. Wait 1ms to satisfy minimum reset assertion time of the PCI specification.
4. Configure the PCI clock GPIO for the proper PCI bus frequency (defined in the section GPIO).
5. Enable the PCI clock GPIO to drive the PCI clock
6. Wait 100 µs to satisfy the “minimum reset assertion time from clock stable” requirement of the PCI specification.
7. Set the PCI reset GPIO output to drive a 1.
   This releases the PCI bus.

Note: The PCI reset can be asserted and de-asserted asynchronously with respect to the PCI clock. It is also important to note the PCI reset signal can not be the same signal as the RESET_IN_N signal going to the IXP42X product line and IXC1100 control plane processors due to PCI reset timing and PCI initialization requirements.

6.13 PCI RCOMP Circuitry

The PCI RCOMP circuitry dynamically compensates for variations in operating conditions due to process, temperature and voltage. These variations are measured through a resistive mechanism in a special I/O Pad and evaluated in the associated compensation circuitry. Adjustments are made to the drive strength of the buffers for the PCI interface ensuring error free operation over the entire range of operating conditions.

The RCOMP circuitry requires an external reference resistor that models the load the PCI pads will see in the board environment. For specific RCOMP pin requirements, see the *Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor Datasheet* (252479). The circuitry adjusts the PCI pads' current sourcing strength by comparing the voltage of the output buffer driven through the external reference resistor with an internally generated 60% threshold voltage. The circuitry adjusts the PCI pads' current sinking strength by comparing the output buffer voltage with an internally generated 40% threshold voltage. Once drive strengths are determined for the 60% and 40% thresholds a multiplier is applied to the drive strengths to provide for a margin above and below the 60% and 40% thresholds, respectively.

6.14 Register Descriptions

6.14.1 PCI Configuration Registers

Table 103 lists the registers comprising the configuration registers as defined in the *PCI Local Bus Specification*, Rev. 2.2. They are accessible from the PCI bus using configuration read and write transactions and from the Intel XScale processor by accessing the PCI Controller CSR-based PCI Configuration register port. Access to these registers are described in “Initializing the PCI Controller Configuration Registers” on page 222

Table 103. PCI Configuration Register Map (Sheet 1 of 2)

Offset	Register Name	Description
0x00	PCI_DIDVID	Device ID/Vendor ID
0x04	PCI_SRCR	Status Register/Control Register
0x08	PCI_CCRID	Class Code/Revision ID
0x0C	PCI_BHLC	BIST/Header Type/Latency Timer/Cache Line
0x10	PCI_BAR0	Base Address 0

Table 103. PCI Configuration Register Map (Sheet 2 of 2)

Offset	Register Name	Description
0x14	PCI_BAR1	Base Address 1
0x18	PCI_BAR2	Base Address 2
0x1C	PCI_BAR3	Base Address 3
0x20	PCI_BAR4	Base Address 4
0x24	PCI_BAR5	Base Address 5
0x28	RESERVED	(Reserved)
0x2C	PCI_SIDSVID	Subsystem ID/Subsystem Vendor ID
0x30-38	RESERVED	(Reserved)
0x3C	PCI_LATINT	Defines Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line
0x40	PCI_RTOTTO	Defines retry timeout and trdy timeout parameters

6.14.1.1 Device ID/Vendor ID Register

(PCI_DIDVID)

Register Name:	PCI_DIDVID				
Hex Offset Address:	0x00		Reset Hex Value:	0x85008086	
Register Description:	Provides Device ID and Vendor ID values as specified in the PCI 2.2 Local Bus Specification				
Access: See below.					

31															16	15														0
Device ID																Vendor ID													

Register	PCI_DIDVID				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	DeviceID	Unique device identifier assigned by Intel	0x8500	RO	RO
15:0	VendorID	Unique vendor identifier assigned to Intel by PCISIG	0x8086	RO	RO

6.14.1.2 Status Register/Control Register

(PCI_SRCR)

Register Name:	PCI_SRCR				
Hex Offset Address:	0x04		Reset Hex Value:	0x02A00000	
Register Description:	Contains the Command and Status registers as specified in the PCI 2.2 Local Bus Specification				
Access: See below.					

31															16	15														0
Status Register																Command Register													

Register	PCI_SRCR (Sheet 1 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31	DPE	Detected Parity Error. Set when this device detects a parity error on the bus even when parity handling is disabled. Writing a 1 to this bit clears it.	0	RW1C	RW1C
30	SSE	Signaled System Error. Set when this device generates a System Error SERR#. Writing a 1 to this bit clears it.	0	RW1C	RW1C
29	RMA	Received Master Abort. Set by this device as a Master when its transaction terminates due to a master abort (except for special cycles). Writing a 1 to this bit clears it.	0	RW1C	RW1C
28	RTA	Received Target Abort. Set by this device as a Master when its transaction is terminated due to a target abort. Writing a 1 to this bit clears it.	0	RW1C	RW1C
27	STA	Signaled Target Abort. Set by this device as a Target when it terminates a transaction with a target abort. Writing a 1 to this bit clears it.	0	RW1C	RW1C
26:25	DEVSEL	Defines the DEVSEL speed for this device. Set to medium.	01	RO	RO
24	MDPE	Master Data Parity Error. Set by this device as a Master if PER (bit 6) is set and this device either asserted the PERR# signal or saw PERR# asserted for one of its data phases.	0	RW1C	RW1C
23	FBBC	Fast Back-to-Back Capable.	1	RO	RO
22	UDF	User Definable Features supported. 0 = not supported	0	RO	RO
21	66MHZ	66MHz Capable. Indicates if this device is capable of 66-MHz operation. 1 = 66MHz capable.	1	RO	RW
20	CLI	Capabilities List Indicator, Not supported	0	RO	RO
19:10	-	(Reserved)	00	RO	RO
9	FBBE	Fast Back-to-Back Enable. When set to a 1 enables the device to generate fast back-to-back cycles to different targets as a Master.	0	RW	RW
8	SER	System Error Enable. When set to a 1, enables the SERR# output driver. 0 disables the driver.	0	RW	RW
7	SC	Stepping Control. When set to a 1, enables address stepping on the bus. This feature not supported.	0	RO	RO
6	PER	Parity Error Response. When set to a 1, enables reporting of parity errors on PERR#. When set to 0, parity errors not reported on PERR# but the DPE bit (bit 31) is still set.	0	RW	RW
5	PSE	Palette Snoop Enable. When set to a 1, enables VGA palette snooping. This feature not supported.	0	RO	RO
4	MWIE	Memory Write and Invalidate Enable. When set to a one, enables this device to generate the memory write and invalidate command.	0	RW	RW
3	SCE	Special Cycle Enable. When set, enables this device to monitor for Special Cycles. This feature not supported.	0	RO	RO

Register	PCI_SRCR (Sheet 2 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
2	BME	Bus Master Enable. When set, enables this device to act as a bus Master.	0	RW	RW
1	MAE	Memory Access Enable. When set to a 1, enables memory accesses as a target.	0	RW	RW
0	IOAE	I/O Access Enable. When set to a 1, enables I/O accesses as a target.	0	RW	RW

6.14.1.3 Class Code/Revision ID Register

(PCI_CCRID)

Register Name:	PCI_CCRID				
Hex Offset Address:	0x08		Reset Hex Value:	0x0B400000	
Register Description:	Provides Class Code and Revision ID values as specified in the PCI 2.2 Local Bus Specification.				
Access: See below.					

31							24	23						16	15							8	7							0
Class Code								Sub-Class Code							Interface								RevisionID							

Register	PCI_CCRID				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	Class Code	Class/Sub-Class identifier for the device as defined in the PCI specification. 0x0b = processor	0x0b	RO	RW
23:16	Sub-Class	Sub-Class identifier for Class Code 0x0b. 0x40 = co-processor	0x40	RO	RW
15:8	Interface	Programming Interface code. Always 0x00 for this class.	0x00	RO	RW
7:0	RevisionID	Silicon revision for the device.	0x01	RO	RW

6.14.1.4 BIST/Header Type/Latency Timer/Cache Line Register

(PCI_BHLC)

Register Name:	PCI_BHLC				
Hex Offset Address:	0x0C		Reset Hex Value:	0x00000000	
Register Description:	Provides BIST, Header Type, Latency Timer, and Cache Line Size registers as specified in the PCI 2.2 Local Bus Specification.				
Access: See below.					

31							24	23						16	15							8	7							0
BIST								Header Type							LatencyTimer								CacheLine							

Register	PCI_BHLC				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	BIST	BIST control register, not supported	0x00	RO	RO
23	Header Type[7]	Single Function/Multi-Function Device. Set to 0 to identify this device as a single-function PCI device.	0	RO	RO
22:16	Header Type[6:0]	Configuration Header Type for this device. Set to 00	0x00	RO	RO
15:10	Latency Timer[7:2]	Latency Timer value in units of four PCI bus clocks.	0x00	RW	RW
9:8	Latency 1 Timer[1:0]	Hard-wired low order Latency Timer bits	0x0	RO	RO
7:0	Cache Line	Cache Line Size in units of 32-bit words.	0x00	RW	RW

6.14.1.5 Base Address 0 Register (PCI_BAR0)

Register Name:	PCI_BAR0				
Hex Offset Address:	0x10		Reset Hex Value:	0x00000008	
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification				
Access: See below.					

31							24	23																4	3	2	1	0
RWBase								FixedBase																	PREF	Type		MSI

Register	PCI_BAR0				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16Mbyte address range for this BAR.	0x000	RO	RO
3	PREF	Pre-fetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

6.14.1.6 Base Address 1 Register

(PCI_BAR1)

Register Name:	PCI_BAR1				
Hex Offset Address:	0x14		Reset Hex Value:	0x00000008	
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.				
Access: See below.					

31							24	23																	4	3	2	1	0
RWBase								FixedBase																		PREF	Type		MSI

Register	PCI_BAR1				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16Mbyte address range for this BAR.	0x000	RO	RO
3	PREF	Pre-fetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

6.14.1.7 Base Address 2 Register

(PCI_BAR2)

Register Name:	PCI_BAR2				
Hex Offset Address:	0x18		Reset Hex Value:	0x00000008	
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.				
Access: See below.					

31							24	23																	4	3	2	1	0
RWBase								FixedBase																		PREF	Type		MSI

Register	PCI_BAR2				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16Mbyte address range for this BAR.	0x000	RO	RO
3	PREF	Pre-fetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

6.14.1.8 Base Address 3 Register

(PCI_BAR3)

Register Name:	PCI_BAR3				
Hex Offset Address:	0x1C		Reset Hex Value:	0x00000008	
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification				
Access: See below.					

31							24	23																	4	3	2	1	0
RWBase								FixedBase																		PREF	Type		MSI

Register	PCI_BAR3				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16Mbyte address range for this BAR.	0x000	RO	RO
3	PREF	Pre-fetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

6.14.1.9 Base Address 4 Register

(PCI_BAR4)

Register Name:	PCI_BAR4				
Hex Offset Address:	0x20		Reset Hex Value:	0x00000008	
Register Description:	PCI Base Address register for PCI Controller PCI accessible CSRs. Format as specified in the PCI 2.2 Local Bus Specification.				
Access: See below.					

31							24	23																	4	3	2	1	0
RWBase								FixedBase																		PREF	Type		MSI

Register	PCI_BAR4				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16Mbyte address range for this BAR.	0x000	RO	RO
3	PREF	Pre-fetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

6.14.1.10 Base Address 5 Register

(PCI_BAR5)

Register Name:	PCI_BAR5				
Hex Offset Address:	0x24		Reset Hex Value:	0x00000001	
Register Description:	PCI Base Address register for AHB I/O space access. Format as specified in the PCI 2.2 Local Bus Specification.				
Access: See below.					

31																					8	7					2	1	0
RWBase																						FixedBase						(Rsvd)	IOSI

Register	PCI_BAR5				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	RWBase	Read/Write bits of Base Address register.	0x000000	RW	RW
7:2	FixedBase	Read-only bits of Base Address register. Specifies fixed 256 byte address range for this BAR.	0x00	RO	RO
1		(Reserved)	0	RO	RO
0	MSI	I/O space indicator. Hard-wire to 1 for I/O space.	1	RO	RO

6.14.1.11 Subsystem ID/Subsystem Vendor ID Register

(PCI_SIDSVID)

Register Name:	PCI_SIDSVID				
Hex Offset Address:	0x2C		Reset Hex Value:	0x00000000	
Register Description:	Provides Subsystem Device ID and Subsystem Vendor ID values as specified in the PCI 2.2 Local Bus Specification				
Access: See below.					

31															16	15														0
SdeviceID																SvendorID														

Register	PCI_SIDSVID				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	SDeviceID	Subsystem Device ID	0x0000	RO	RW
15:0	SVendorID	Subsystem Vendor ID	0x0000	RO	RW

6.14.1.12 Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line Register (PCI_LATINT)

Register Name:	PCI_LATENT				
Hex Offset Address:	0x3C		Reset Hex Value:	0x00000100	
Register Description:	Miscellaneous register provides Max Latency, Min Grant, Interrupt Pin and Interrupt Line parameters as specified in the PCI 2.2 Local Bus Specification				
Access: See below.					

31	24	23	16	15	8	7	0
MaxLat		MinGnt		IntPin		IntLine	

Register	PCI_LATENT				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	MaxLat	Indicates how often this device needs access to the bus, in units of 0.25us. Used by configuration software to set the value of the Latency Timer.	0x00	RO	RW
23:16	MinGnt	Indicates the time interval required for a burst operation, in units of 0.25us. Used by configuration software to set the value of the Latency Timer.	0x00	RO	RW
15:8	IntPin	Indicates which interrupt pin this device connects to. Set to connect to INT_A#.	0x01	RO	RW
7:0	IntLine	Indicates interrupt line routing information.	0x00	RW	RW

6.14.1.13 Retry Timeout/TRDY Timeout Register (PCI_RTOTTO)

Register Name:	PCI_RTOTTO				
Hex Offset Address:	0x40		Reset Hex Value:	0x00008080	
Register Description:	Specifies values for the Retry and TRDY timeout timers.				
Access: See below.					

31	16	15	8	7	0
(Reserved)		RetryTO		TRDYTO	

Register	PCI_RTOTTO				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:16		(Reserved)	0x00	RO	RO
15:8	RetryTO	Specifies value for the Retry timer. Specifies the maximum number of retries the Master Interface will accept before terminating the transaction.	0x80	RW	RW
7:0	TRDYTO	Specifies value for the TRDY timer. Specifies the number of PCI clocks the Master Interface will wait before terminating a transfer with Master Abort when a target accepts a transaction by asserting DEVSEL# but does not assert TRDY# or STOP#.	0x80	RW	RW

6.14.2 PCI Controller Configuration and Status Registers

These registers are accessible from the AHB and are memory mapped in the AHB address space. Table 104 shows the address map for the Control and Status Register. The PCI offset is relative to the base address in PCI_BAR4 for accesses from the PCI bus.

Table 104. PCI Controller CSR Address Map

AHB Address	PCI Offset	Register Name	Description
0 x C0000000	0x00	PCI_NP_AD	PCI non-pre-fetch address register
0 x C0000004	0x04	PCI_NP_CBE	PCI non-pre-fetch command/byte enables register
0 x C0000008	0x08	PCI_NP_WDATA	PCI non-pre-fetch write data register
0 x C000000C	0x0C	PCI_NP_RDATA	PCI non-pre-fetch read data register
0 x C0000010	0x10	PCI_CRP_AD_CBE	PCI configuration port address/cmd/byte enables register
0 x C0000014	0x14	PCI_CRP_WDATA	PCI configuration port write data register
0 x C0000018	0x18	PCI_CRP_RDATA	PCI configuration port read data register
0 x C000001C	0x1C	PCI_CSR	PCI Controller Control and Status register
0 x C0000020	0x20	PCI_ISR	PCI Controller Interrupt Status register
0 x C0000024	0x24	PCI_INTEN	PCI Controller Interrupt Enable register
0 x C0000028	0x28	PCI_DMACTRL	DMA control register
0 x C000002C	0x2C	PCI_AHBMEMBASE	AHB Memory Base Address Register
0 x C0000030	0x30	PCI_AHBIOBASE	AHB I/O Base Address Register
0 x C0000034	0x34	PCI_MEMBASE	PCI Memory Base Address Register
0 x C0000038	0x38	PCI_AHBDOORBELL	AHB Doorbell Register
0 x C000003C	0x3C	PCI_PCIDOORBELL	PCI Doorbell Register
0 x C0000040	0x40	PCI_ATPDMA0_AHBADDR	AHB to PCI DMA AHB Address Register 0
0 x C0000044	0x44	PCI_ATPDMA0_PCIADDR	AHB to PCI DMA PCI Address Register 0
0 x C0000048	0x48	PCI_ATPDMA0_LENGTH	AHB to PCI DMA Length Register 0
0 x C000004C	0x4C	PCI_ATPDMA1_AHBADDR	AHB to PCI DMA AHB Address Register 1
0 x C0000050	0x50	PCI_ATPDMA1_PCIADDR	AHB to PCI DMA PCI Address Register 1
0 x C0000054	0x54	PCI_ATPDMA1_LENGTH	AHB to PCI DMA Length Register 1
0 x C0000058	0x58	PCI_PTADMA0_AHBADDR	PCI to AHB DMA AHB Address Register 0
0 x C000005C	0x5C	PCI_PTADM0_PCIADDR	PCI to AHB DMA PCI Address Register 0
0 x C0000060	0x60	PCI_PTADM0_LENGTH	PCI to AHB DMA Length Register 0
0 x C0000064	0x64	PCI_PTADM1_AHBADDR	PCI to AHB DMA AHB Address Register 1
0 x C0000068	0x68	PCI_PTADM1_PCIADDR	PCI to AHB DMA PCI Address Register 1
0 x C000006C	0x6C	PCI_PTADM1_LENGTH	PCI to AHB DMA Length Register 1

6.14.2.1 PCI Controller Non-pre-fetch Address Register (PCI_NP_AD)

Register Name:	PCI_NP_AD				
Hex Offset Address:	0xC0000000		Reset Hex Value:	0x00000000	
Register Description:	PCI non-pre-fetch access address register. Provides address for CSR-initiated non-pre-fetch PCI accesses.				
Access: See below.					

31															16	15														0
NP_ADDRESS																NP_ADDRESS													

Register	PCI_NP_AD				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_address	Address of the non-pre-fetch PCI cycle. The format of the address depends on the PCI command type used.	0x00000000	RO	RW

6.14.2.2 PCI Controller Non-pre-fetch Command/Byte Enables Register (PCI_NP_CBE)

Register Name:	PCI_NP_CBE				
Hex Offset Address:	0xC0000004		Reset Hex Value:	0x00000000	
Register Description:	PCI non-pre-fetch access command/byte enables register. Provides PCI command and data byte enables for CSR-initiated non-pre-fetch PCI accesses				
Access: See below.					

31																						8	7			4	3			0
(Reserved)																							NP_BE				NP_CMD			

Register	PCI_NP_CBE				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8		(Reserved) – Read as 0	0x000000	RO	RW
7:4	NP_BE	Byte enables driven onto the C/BE[3:0]# lines of the PCI bus during the data phase of the non-pre-fetch PCI access.	0x0	RO	RW
3:0	NP_CMD	PCI command driven onto the C/BE[3:0]# lines of the PCI bus during the address phase of the non-pre-fetch PCI access.	0x0	RO	RW

6.14.2.3 PCI Controller Non-Pre-fetch Write Data Register (PCI_NP_WDATA)

Register Name:	PCI_NP_WDATA				
Hex Offset Address:	0xC0000008		Reset Hex Value:	0x00000000	
Register Description:	PCI non-pre-fetch access write data register. Provides write data for CSR-initiated non-pre-fetch PCI write access.				
Access: See below.					

31																												0
NP_WDATA																												

Register	PCI_NP_WDATA				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_wdata	Write data for the non-pre-fetch PCI write cycle.	0x00000000	RO	RW

6.14.2.4 PCI Controller Non-Pre-fetch Read Data Register (PCI_NP_RDATA)

Register Name:	PCI_NP_RDATA				
Hex Offset Address:	0xC000000C		Reset Hex Value:	0x00000000	
Register Description:	PCI non-pre-fetch access read data register. Holds read data from the CSR-initiated non-pre-fetch PCI read access.				
Access: See below.					

31																												0
NP_RDATAH																												

Register	PCI_NP_RDATA				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_rdata	Read data from the non-pre-fetch PCI read cycle.	0x00000000	RO	RO

6.14.2.5 PCI Controller Configuration Port Address/Command/ Byte Enables Register (PCI_CRP_AD_CBE)

Register Name:	PCI_CRP_AD_CBE				
Hex Offset Address:	0xC0000010		Reset Hex Value:	0x00000000	
Register Description:	PCI configuration port address/command/byte enables register. Provides address, command, and data byte enables for CSR-initiated accesses of the PCI Controller PCI configuration registers in the PCI Core. A write to this register that sets the CRP_CMD[16] bit to a 0 (read) will initiate a read of the PCI Controller PCI configuration register addressed by CRP_AD[7:2]. The resultant read data will be written to the pci_crp_rdata register.				
Access: See below.					

31							24	23			20	19			16	15				11	10								0
(Reserved)								CRP_BE				CRP_CMD				(Reserved)					CRP_AD								

Register	PCI_CRP_AD_CBE				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24		(Reserved) – Read as 0	0x00	RO	RO
23:20	CRP_BE	Active-low byte enables for a PCI configuration port write access. This field corresponds to byte lanes in the pci_crp_wdata register and addressed PCI configuration register as follows: CRP_BE[0] à bits 7:0 CRP_BE[1] à bits 15:8 CRP_BE[2] à bits 23:16 CRP_BE[3] à bits 31:24	0x0	RO	RW
19:16	CRP_CMD	Command for the PCI configuration port access. xxx0 = read, xxx1 = write	0x0	RO	RW
15:11		(Reserved) – Read as 0	00000	RO	RO
10:0	CRP_AD	Byte address for the PCI configuration port access. PCI configuration registers are word-addressed so bits 1 and 0 should always be 0. Bits 10:8 specify the function number and must always be 000.	0x000	RO	RW

6.14.2.6 PCI Controller Configuration Port Write Data Register (PCI_CRP_WDATA)

Register Name:	PCI_CRP_WDATA				
Hex Offset Address:	0xC0000014		**Reset Hex Value:**	0x00000000	
Register Description:	PCI configuration port write data register. Provides write data for CSR-initiated accesses of the PCI Controller PCI configuration registers in the PCI Core. If the pci_crp_ad_cmd_be.CRP_CMD[16] bit is 1 (write), a write to pci_crp_wdata will initiate a write to the PCI Controller PCI configuration register addressed by pci_crp_ad_cmd_be.CRP_AD[7:2]. The pci_crp_ad_cmd_be.CRP_BE field determines which bytes are affected.				
Access: See below.					

31																													**0**
CRP_WDATA																													

Register	PCI_CRP_WDATA				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:0	CRP_WDATA	Write data for the configuration port write access.	0x00000000	RO	RW

6.14.2.7 PCI Controller Configuration Port Read Data Register (PCI_CRP_RDATA)

Register Name:	PCI_CRP_RDATA				
Hex Offset Address:	0xC0000018		Reset Hex Value:	0x00000000	
Register Description:	PCI configuration port read data register. Provides read data for CSR-initiated read accesses of the PCI Controller PCI configuration registers in the PCI Core.				
Access: See below.					

31																												0
CRP_RDATA																												

Register	PCI_CRP_RDATA				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	CRP_RDATA	Read data for the configuration port read access.	0x00000000	RO	RO

6.14.2.8 PCI Controller Control and Status Register (PCI_CSR)

Register Name:	PCI_CSR				
Hex Offset Address:	0xC000001C		Reset Hex Value:	0x0000000x	
Register Description:	Control and status for the PCI Controller.				
Access: See below.					

31														17	16	15	14					9	8	7	6	5	4	3	2	1	0
(Reserved)															PRST	IC	(Reserved)						ASE	(Rsvd)		DBT	ABE	PBS	ABS	ARBEN	HOST

Register	PCI_CSR (Sheet 1 of 2)				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:17		(Reserved) – Read as 0	0x0000	RO	RO
16	PRST	PCI Reset, When set to a 1, initializes the PCI controller flip flops clocked by the PCI clock.	0x0	RO	RW
15	IC	Initialization Complete. When at a logic 0 state, forces the PCI Controller Target Interface to retry PCI cycles. When set to a 1, PCI cycles will be accepted.	0	RO	RW
14:9		(Reserved) – Read as 0	0x00	RO	RO
8	ASE	Assert System Error. When set to a 1, the PCI SERR# output (PCI_SERR_N) will be asserted for 1 PCI clock cycle if the pci_srcr.SER bit is set.	0	RO	RW
7:6		(Reserved) – Read as 0	00	RO	RO
5	DBT	Doorbell Test mode enable. When set to a 1, the doorbell registers pci_ahbdoorbell, pci_pcidoorbell become normal read/write registers from the AHB bus.	0	RO	RW

Register	PCI_CSR (Sheet 2 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
4	ABE	AHB big-endian addressing. When 0, little-endian addressing is employed on both AHB master and slave interfaces. When 1, big-endian addressing is implemented.	0	RO	RW
3	PDS	PCI byte swap. Controls byte lane data routing between PCI and AHB buses during PCI Target accesses of the AHB bus. When 1, byte lane swapping is performed. When 0, no swapping is done.	0	RO	RW
2	ADS	AHB byte swap. Controls byte lane data routing between PCI and AHB buses during AHB Slave accesses of the PCI bus. When 1, byte lane swapping is performed. When 0, no swapping is done.	0	RO	RW
1	ARBEN	Arbiter enable status. Indicates the state of the Expansion bus address input bit corresponding to the PCI arbiter setting at the de-assertion of reset_n.	0 or 1	RO	RO
0	HOST	Host status. Indicates the state of the Expansion bus address input bit corresponding to the PCI host/option setting at the deassertion of reset_n	0 or 1	RO	RO

6.14.2.9 PCI Controller Interrupt Status Register (PCI_ISR)

Register Name:	PCI_ISR				
Hex Offset Address:	0xC0000020		**Reset Hex Value:**	0x00000000	
Register Description:	Indicates the interrupt source(s) for the pcc_int interrupt output and the PCI interrupt PCI_INTA_N. With the exception of PDB (PCI Doorbell), when any bit is a 1 and the corresponding bit in the pci_inten register is set, pcc_int will be asserted. If the PDB bit is a 1 and the pci_inten.PDB bit is set, the PCI_INTA_N output will be asserted to generate an interrupt on the PCI bus.				
Access: See below.					

31	…	8	7	6	5	4	3	2	1	0
(Reserved)			PDB	ADB	PADC	APDC	AHBE	PPE	PFE	PSE

Register	PCI_ISR (Sheet 1 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:8		(Reserved) – Read as 0	0x000000	RO	RO
7	PDB	PCI Doorbell interrupt. Asserted high when any one of the bits in the pci_pcidoorbell register is set. This bit does not cause an interrupt to be asserted on pcc_int. When the pci_inten.PDB bit is set, the PCI_INTA_N PCI interrupt output is asserted.	0	RO	RO
6	ADB	AHB Doorbell interrupt. Asserted high when any one of the bits in the pci_ahbdoorbell register is set.	0	RO	RO
5	PADC	PCI to AHB DMA Complete. Asserted high when a PCI to AHB DMA transfer is complete.	0	RO	RO
4	APDC	AHB to PCI DMA Complete. Asserted high when a AHB to PCI DMA transfer is complete.	0	RO	RO
3	AHBE	AHB Error indication. Set to a 1 when the AHB Master Interface receives an ERROR response.	0	RO	RW1C

Register	PCI_ISR (Sheet 2 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
2	PPE	PCI Parity Error. Set to a 1 when a parity error occurs on the PCI bus: Parity error detected during Master Interface read cycle. pad_perr_n (PERR#) asserted by an external target during a Master write cycle.	0	RO	RW1C
1	PFE	PCI Fatal Error. Set to a 1 when one of the following errors occurs on the PCI bus: master abort (target did not respond) target abort TRDY timeout (external target asserts DEVSEL# but never asserts TRDY#) retry timeout (external target issued more retries than specified by the RetryTO field in the pci_rtotto register)	0	RO	RW1C
0	PSE	PCI System Error. Set to a 1 when the PCI Controller detects that the PCI SERR# signal has been asserted.	0	RO	RW1C

6.14.2.10 PCI Controller Interrupt Enable Register (PCI_INTEN)

Register Name:	PCI_INTEN				
Hex Offset Address:	0xC0000024		Reset Hex Value:	0x00000000	
Register Description:	Interrupt enables for the interrupt status bits in the pci_isr register. Set to a 1 to enable the particular interrupt. With the exception of PDB (PCI Doorbell), when an interrupt is enabled and the corresponding bit in the pci_isr register is set, pcc_int will be asserted. If the PDB interrupt is enabled and the pci_isr.PDB bit is asserted, the PCI_INTA_N output will be asserted to generate an interrupt on the PCI bus.				

Access: See below.

31																						8	7	6	5	4	3	2	1	0
(Reserved)																							PDB	ADB	PADC	APDC	AHBE	PPE	PFE	PSE

Register	PCI_INTEN				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:8		(Reserved) – Read as 0	0x000000	RO	RO
7	PDB	PCI Doorbell interrupt enable.	0	RO	RW
6	ADB	AHB Doorbell interrupt enable.	0	RO	RW
5	PADC	PCI to AHB DMA Complete interrupt enable.	0	RO	RW
4	APDC	AHB to PCI DMA Complete interrupt enable.	0	RO	RW
3	AHBE	AHB Error indication interrupt enable.	0	RO	RW
2	PPE	PCI Parity Error interrupt enable.	0	RO	RW
1	PFE	PCI Fatal Error interrupt enable.	0	RO	RW
0	PSE	PCI System Error interrupt enable.	0	RO	RW

6.14.2.11 DMA Control Register

(PCI_DMACTRL)

Register Name:	**PCI_DMACTRL**				
Hex Offset Address:	0xC0000028		**Reset Hex Value:**	0x00000000	
Register Description:	Control and status for the DMA Controller channels.				
Access: See below.					

31															**16**	**15**	**14**	**13**	**12**	**11**		**9**	**8**	**7**	**6**	**5**	**4**	**3**		**1**	**0**
(Reserved)																PADE1	PADC1	PADE0	PADC0	(Rsvd)			PADCEN	APDE1	APDC1	APDE0	APDC0	(Rsvd)			APDCEN

Register	**PCI_DMACTRL** (Sheet 1 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:16		(Reserved) – Read as 0	0x0000	RO	RO
15	PADE1	PCI to AHB DMA error for buffer 1. Set to a 1 when the DMA transfer specified by the pci_ptadma1_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the PADC1 bit.	0	RO	RO
14	PADC1	PCI to AHB DMA complete for buffer 1. Set to a 1 when the DMA transfer specified by the pci_ptadma1_xxx registers is complete or terminated due to an error. If the PADCEN bit is a 1, the pcc_ptadma_int output is asserted.	0	RO	RW1C
13	PADE0	PCI to AHB DMA error for buffer 0. Set to a 1 when the DMA transfer specified by the pci_ptadma0_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the PADC0 bit.	0	RO	RO
12	PADC0	PCI to AHB DMA complete for buffer 0. Set to a 1 when the DMA transfer specified by the pci_ptadma0_xxx registers is complete or terminated due to an error. If the PADCEN bit is a 1, the pcc_ptadma_int output is asserted.	0	RO	RW1C
11:9		(Reserved) – Read as 0	000	RO	RO
8	PADCEN	PCI to AHB DMA Complete interrupt enable. If this bit is set and either PADC0 or PADC1 are 1, the pcc_ptadma_int output is asserted.	0	RO	RW
7	APDE1	AHB to PCI DMA error for buffer 1. Set to a 1 when the DMA transfer specified by the pci_atpdma1_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the APDC1 bit.	0	RO	RO
6	APDC1	AHB to PCI DMA complete for buffer 1. Set to a 1 when the DMA transfer specified by the pci_atpdma1_xxx registers is complete or terminated due to an error. If the APDCEN bit is a 1, the pcc_atpdma_int output is asserted.	0	RO	RW1C
5	APDE0	AHB to PCI DMA error for buffer 0. Set to a 1 when the DMA transfer specified by the pci_atpdma0_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the APDC0 bit.	0	RO	RO

Register	PCI_DMACTRL (Sheet 2 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
4	APDC0	AHB to PCI DMA complete for buffer 0. Set to a 1 when the DMA transfer specified by the pci_atpdma0_xxx registers is complete or terminated due to an error. If the APDCEN bit is a 1, the pcc_atpdma_int output is asserted high.	0	RO	RW1C
3:1		(Reserved) – Read as 0	000	RO	RO
0	APDCEN	AHB to PCI DMA Complete interrupt enable. If this bit is set and either APDC0 or APDC1 are 1, the pcc_atpdma_int output is asserted active high.	0	RO	RW

6.14.2.12 AHB Memory Base Address Register

(PCI_AHBMEMBASE)

Register Name:	PCI_AHBMEMBASE				
Hex Offset Address:	0xC000002C		Reset Hex Value:	0xc0000000	
Register Description:	Provides upper 8 AHB address bits for PCI accesses of AHB bus. Lower 24 bits of AHB address provided directly from PCI bus. Four AHBbase fields correspond to accesses from the PCI bus that target addresses in PCI configuration base address registers pci_bar0/1/2/3.				
Access: See below.					

31							24	23						16	15						8	7						0
AHBbase0								AHBbase1							AHBbase2							AHBbase3						

Register	PCI_AHBMEMBASE				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	AHBbase0	Upper 8 AHB address bits for PCI accesses that target pci_bar0. By default this register maps to an upper region of the AHB memory map.	0xc0	RO	RW
23:16	AHBbase1	Upper 8 AHB address bits for PCI accesses that target pci_bar1.	0x00	RO	RW
15:8	AHBbase2	Upper 8 AHB address bits for PCI accesses that target pci_bar2.	0x00	RO	RW
7:0	AHBbase3	Upper 8 AHB address bits for PCI accesses that target pci_bar3.	0x00	RO	RW

6.14.2.13 AHB I/O Base Address Register

(PCI_AHBIOBASE)

Register Name:	PCI_AHBIOBASE				
Hex Offset Address:	0xC0000030		Reset Hex Value:	0x00000000	
Register Description:	Provides upper 24 AHB address bits for PCI accesses of AHB I/O space. Lower 8 bits of AHB address provided directly from PCI bus.				
Access: See below.					

31							24	23						16	15													0
(Reserved)								IObase																				

Register	PCI_AHBIOBASE				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24		(Reserved) – Read as 0	0x00	RO	RO
23:0	Iobase	Upper 24 AHB address bits for PCI accesses that target pci_bar5.	0x000000	RO	RW

6.14.2.14 PCI Memory Base Address Register

(PCI_PCIMEMBASE)

Register Name:	PCI_PCIMEMBASE				
Hex Offset Address:	0xC0000034		Reset Hex Value:	0x00000000	
Register Description:	Provides upper 8 PCI address bits for AHB accesses of PCI memory space. Lower 24 bits of PCI address provided directly from the AHB bus. Four Membase fields correspond to accesses from the AHB bus that target specific AHB address ranges.				
Access: See below.					

31							24	23						16	15						8	7						0
Membase0								Membase1							Membase2							Membase3						

Register	PCI_PCIMEMBASE				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	PCIbase0	Upper 8 PCI address bits for AHB accesses that target the first 16Mbyte PCI memory partition.	0x00	RO	RW
23:16	PCIbase1	Upper 8 PCI address bits for AHB accesses that target the second 16Mbyte PCI memory partition.	0x00	RO	RW
15:8	PCIbase2	Upper 8 PCI address bits for AHB accesses that target the third 16Mbyte PCI memory partition.	0x00	RO	RW
7:0	PCIbase3	Upper 8 PCI address bits for AHB accesses that target the fourth 16Mbyte PCI memory partition.	0x00	RO	RW

6.14.2.15 AHB Doorbell Register

(PCI_AHBDOORBELL)

Register Name:	PCI_AHBDOORBELL				
Hex Offset Address:	0xC0000038		Reset Hex Value:	0x00000000	
Register Description:	An external PCI device writes this register to assert the pcc_int signal to interrupt the Intel XScale processor. Any bit set to a 1 will assert pcc_int if the AHB doorbell interrupt is enabled (pci_inten.ADBEN = 1). This register is write-1-to-set from PCI and write-1-to-clear from AHB. The PCI device writes a 1 to a bit or pattern of bits to generate the interrupt. The AHB agent reads the register and writes 1(s) to clear the bit(s) and de-assert the interrupt. If the DBT (Doorbell Test) bit is set in the pci_csr register, all bits become read/write from the AHB bus.				
Access: See below.					

31																											0
ADB																											

Register	PCI_AHBDOORBELL				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:0	ADB	PCI generated doorbell interrupt to an AHB agent. Normally read/write-1-to-set from PCI and read/write-1-to-clear from AHB. Read/write from the AHB side if Doorbell Test mode is enabled by setting pci_csr.DBT to a 1.	0x00000000	RW1S	RW1C (RW if pci_csr.DBT =1)

6.14.2.16 PCI Doorbell Register

(PCI_PCIDOORBELL)

Register Name:	PCI_PCIDOORBELL				
Hex Offset Address:	0xC000003C		Reset Hex Value:	0x00000000	
Register Description:	The Intel XScale processor writes this register to generate an interrupt to an external PCI device on PCI_INTA_N (INTA# on PCI). Any bit set to a 1 will generate the PCI interrupt if the PCI doorbell interrupt is enabled (pci_inten.PDBEN = 1). This register is write-1-to-set from AHB and write-1-to-clear from PCI. The Intel XScale processor writes a 1 to a bit or pattern of bits to generate the interrupt. The external PCI device reads the register and writes 1(s) to clear the bit(s) and de-assert the interrupt. If the DBT (Doorbell Test) bit is set in the pci_csr register, all bits become read/write from the AHB bus.				
Access: See below.					

31																													0
PDB																													

Register	PCI_PCIDOORBELL				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:0	PDB	AHB generated doorbell interrupt to PCI. Normally read/write-1-to-set from AHB and read/write-1-to-clear from PCI. Read/write from the AHB side if Doorbell Test mode is enabled by setting pci_csr.DBT to a 1.	0x00000000	RW1C	RW1S (RW if pci_csr.DBT =1)

6.14.2.17 AHB to PCI DMA AHB Address Register 0

(PCI_ATPDMA0_AHBADDR)

Register Name:	PCI_ATPDMA0_AHBADDR				
Hex Offset Address:	0xC0000040		Reset Hex Value:	0x00000000	
Register Description:	Source address on the AHB bus for AHB to PCI DMA transfers. Paired with pci_atpdma1_ahbaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																												2	1	0
address																													0	0

Register	PCI_ATPDMA0_AHBADDR				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	address	AHB word address	0x00000000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

6.14.2.18 AHB to PCI DMA PCI Address Register 0

(PCI_ATPDMA0_PCIADDR)

Register Name:	PCI_ATPDMA0_PCIADDR				
Hex Offset Address:	0xC0000044		Reset Hex Value:	0x00000000	
Register Description:	Destination address on the PCI bus for AHB to PCI DMA transfers. Paired with pci_atpdma1_pciaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																											2	1	0
address																												0	0

Register	PCI_ATPDMA0_PCIADDR				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	PCI word address	0x00000000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

6.14.2.19 AHB to PCI DMA Length Register 0

(PCI_ATPDMA0_LENGTH)

Register Name:	PCI_ATPDMA0_LENGTH				
Hex Offset Address:	0xC0000048		Reset Hex Value:	0x00000000	
Register Description:	Provides word count and control for AHB to PCI DMA transfers. Paired with pci_atpdma1_length to allow buffering of DMA transfer requests.				
Access: See below.					

31	30	29	28	27										16	15														0
EN	(Rsvd)		DS	(Reserved)											Wordcount														

Register	PCI_ATPDMA0_LENGTH				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29		(Reserved). Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the AHB bus is byte swapped before being sent to the PCI bus. When 0, no swapping is done.	0	RO	RW
27:16		(Reserved). Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

6.14.2.20 AHB to PCI DMA AHB Address Register 1

(PCI_ATPDMA1_AHBADDR)

Register Name:	PCI_ATPDMA1_AHBADDR				
Hex Offset Address:	0xC000004C		Reset Hex Value:	0x00000000	
Register Description:	Source address on the AHB bus for AHB to PCI DMA transfers. Paired with pci_atpdma0_ahbaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																											2	1	0
address																												0	0

Register	PCI_ATPDMA1_AHBADDR				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	AHB word address	0x00000000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

6.14.2.21 AHB to PCI DMA PCI Address Register 1

(PCI_ATPDMA1_PCIADDR)

Register Name:	PCI_ATPDMA1_PCIADDR				
Hex Offset Address:	0xC0000050		Reset Hex Value:	0x00000000	
Register Description:	Destination address on the PCI bus for AHB to PCI DMA transfers. Paired with pci_atpdma0_pciaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																											2	1	0
address																												0	0

Register	PCI_ATPDMA1_PCIADDR				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	PCI word address	0x00000000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

6.14.2.22 AHB to PCI DMA Length Register 1

(PCI_ATPDMA1_LENGTH)

Register Name:	PCI_ATPDMA1_LENGTH				
Hex Offset Address:	0xC0000054		Reset Hex Value:	0x00000000	
Register Description:	Provides word count and control for AHB to PCI DMA transfers. Paired with pci_atpdma0_length to allow buffering of DMA transfer requests.				
Access: See below.					

31	30	29	28	27										16	15															0
EN	(Rsvd)		BE	(Reserved)											wordcount															

Register		PCI_ATPDMA1_LENGTH			
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29		(Reserved). Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the AHB bus is byte swapped before being sent to the PCI bus. When 0, no swapping is done.	0	RO	RW
27:16		(Reserved). Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

6.14.2.23 PCI to AHB DMA AHB Address Register 0

(PCI_PTADMA0_AHBADDR)

Register Name:	PCI_PTADMA0_AHBADDR				
Hex Offset Address:	0xC0000058		**Reset Hex Value:**	0x00000000	
Register Description:	Destination address on the AHB bus for PCI to AHB DMA transfers. Paired with pci_ptadma1_ahbaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																											2	1	0
address																												0	0

Register		PCI_PTADMA0_AHBADDR			
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	address	AHB word address	0x00000000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

6.14.2.24 PCI to AHB DMA PCI Address Register 0

(PCI_PTADMA0_PCIADDR)

Register Name:	PCI_PTADMA0_PCIADDR				
Hex Offset Address:	0xC000005c		**Reset Hex Value:**	0x00000000	
Register Description:	Source address on the PCI bus for PCI to AHB DMA transfers. Paired with pci_ptadma1_pciaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																											2	1	0
address																												0	0

Register	PCI_PTADMA0_PCIADDR				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	address	PCI word address	0x00000000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

6.14.2.25 PCI to AHB DMA Length Register 0

(PCI_PTADMA0_LENGTH)

Register Name:	PCI_PTADMA0_LENGTH				
Hex Offset Address:	0xC0000060		Reset Hex Value:	0x00000000	
Register Description:	Provides word count and control for PCI to AHB DMA transfers. Paired with pci_ptadma1_length to allow buffering of DMA transfer requests.				
Access: See below.					

31	30	29	28	27										16	15													0
EN	(Rsvd)		BE	(Reserved)											wordcount													

Register	PCI_PTADMA0_LENGTH				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29		(Reserved). Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the PCI bus is byte swapped before being sent to the AHB bus. When 0, no swapping is done.	0	RO	RW
27:16		(Reserved). Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

6.14.2.26 PCI to AHB DMA AHB Address Register 1

(PCI_PTADMA1_AHBADDR)

Register Name:	PCI_PTADMA1_AHBADDR				
Hex Offset Address:	0xC0000064		Reset Hex Value:	0x00000000	
Register Description:	Destination address on the AHB bus for PCI to AHB DMA transfers. Paired with pci_ptadma0_ahbaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																											2	1	0
address																												0	0

Register	PCI_PTADMA1_AHBADDR				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	Address	AHB word address	0x00000000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

6.14.2.27 PCI to AHB DMA PCI Address Register 1 (PCI_PTADMA1_PCIADDR)

Register Name:	PCI_PTADMA1_PCIADDR				
Hex Offset Address:	0xC0000068		Reset Hex Value:	0x00000000	
Register Description:	Source address on the PCI bus for PCI to AHB DMA transfers. Paired with pci_ptadma0_pciaddr to allow buffering of DMA transfer requests.				
Access: See below.					

31																										2	1	0
address																											0	0

Register	PCI_PTADMA1_PCIADDR				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	Address	PCI word address	0x00000000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

6.14.2.28 PCI to AHB DMA Length Register 1 (PCI_PTADMA1_LENGTH)

Register Name:	PCI_PTADMA1_LENGTH				
Hex Offset Address:	0xC000006C		Reset Hex Value:	0x00000000	
Register Description:	Provides word count and control for PCI to AHB DMA transfers. Paired with pci_ptadma0_length to allow buffering of DMA transfer requests.				
Access: See below.					

31	30	29	28	27										16	15															0
EN	(Rsvd)		BE	(Reserved)											wordcount															

Register	PCI_PTADMA1_LENGTH (Sheet 1 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29		(Reserved). Read as 0.	00	RO	RO

Register		PCI_PTADMA1_LENGTH (Sheet 2 of 2)			
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
28	DS	Data Swap indicator. When set to a 1, data from the PCI bus is byte swapped before being sent to the AHB bus. When 0, no swapping is done.	0	RO	RW
27:16		(Reserved). Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

§ §

7.0 SDRAM Controller

The SDRAM Controller performs data movement between the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor and an attached SDRAM. The SDRAM Controller is a target only function on both AHB interfaces and supports a maximum of 256 Mbyte of addressable space.

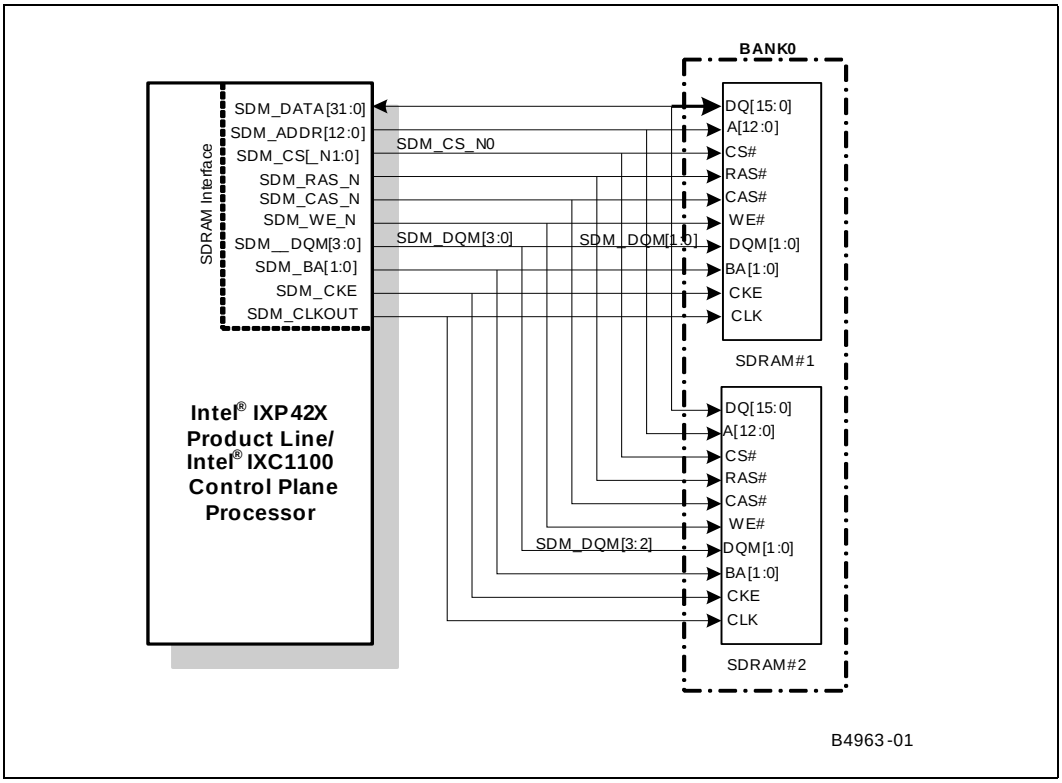
Table 105 shows the supported memory configuration. Support is included for two memory banks of SDRAM devices. The SDRAM Controller supports a maximum burst length of eight words. The eight-word burst length was derived from the Intel XScale® Processor cache line size. This choice of the eight-word burst size optimizes the performance of the Intel XScale processor at the same time ensuring fairness among all resources trying to obtain access to the SDRAM.

The SDRAM Controller can be configured from the South AHB bus only. The SDRAM controller provides separate interfaces to the South AHB and North AHB to allow for maximum efficiency of the SDRAM accesses. The SDRAM Controller supports a RAS-to-CAS delay of three clocks. The SDRAM Controller can be programmed to support a CAS-to-data delay of two or three clocks. The SDRAM Controller can maintain up to eight simultaneously open pages.

Two independent chip-selects are provided which allows the SDRAM Controller to support a total of two physical banks of memory. A minimum of 8Mbyte (using 64Mbit density chips) to maximum of 256Mbyte (using 512Mbit density chips) memory configurations are supported.

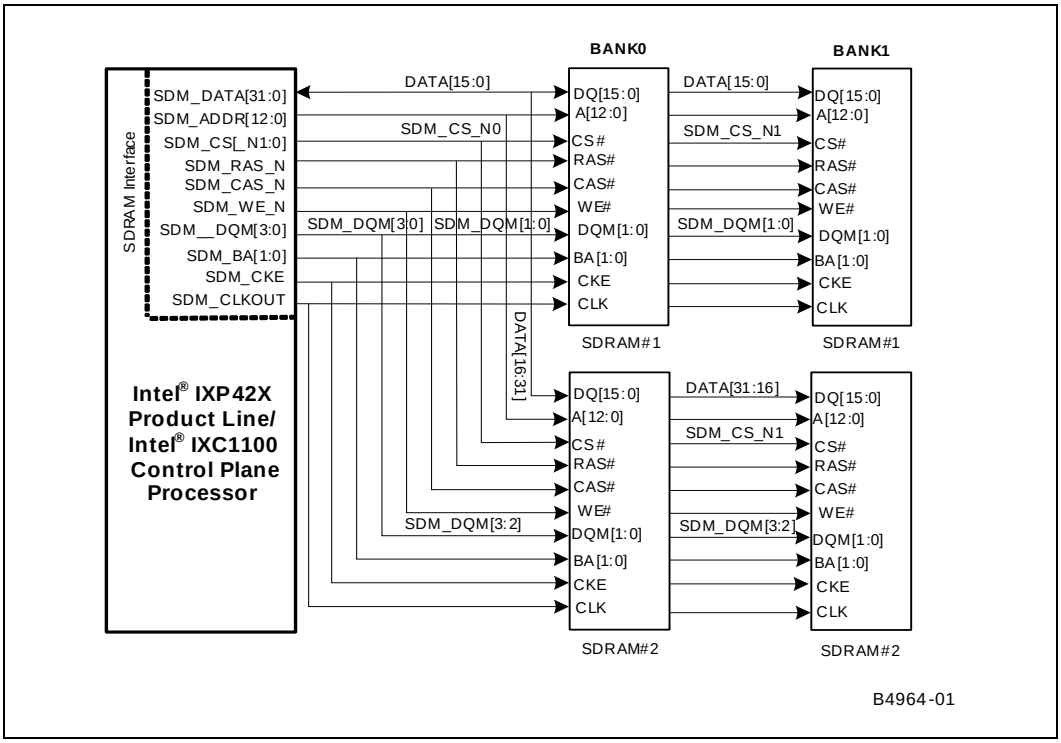
Figure 53 shows a configuration of SDRAM using only Bank0 of the Intel® IXP42X product line and IXC1100 control plane processors. Bank 0 consists of two SDRAM devices that are of type by 16 (x16) or 16-bit.

Figure 53. 8-, 16-, 32-, 64- or 128-Mbyte — One-Bank SDRAM Interface Configuration



As stated previously, the supported memory types for each bank of SDRAM must be 64Mbit, 128-Mbit, 256-Mbit, or 512-Mbit memory device types. The IXP42X product line and IXC1100 control plane processors memory controller supports only two memory devices per each bank. Figure 54 illustrates how to interface to two banks of SDRAM memory. Each memory device in the below example is a 16-bit device (x16).

Figure 54. 64-, 128- or 256-Mbyte — Two-Bank SDRAM Interface Configuration



To remove the need for SDRAM refreshes to be implemented by the Intel XScale processor, the SDRAM controller can be configured to perform automatic refreshes by utilizing an internal refresh counter.

Table 105. Supported Configuration of the SDRAM Controller

Total Memory	64 Mbit	128 Mbit	256 Mbit	512 Mbit
8 Mbyte	One Chip 2M x 32			
16 Mbyte	Two chips 4M x 16 or two chips 2M x 32			
32 Mbyte	Four chips 4M x 16	Two chips 8 M x 16		
64 Mbyte		Four chips 8 M x 16	Two chips 16 M x 16	
128 Mbyte			Four chips 16 M x16	Two chips 32 M x 16
256 Mbyte				Four chips 32 M x 16
Note: 64-Mbit, 128-Mbit, 256-Mbit, and 512-Mbit columns refer to memory technologies.				

The SDRAM Controller is a target-only device residing on the AHB. A master residing on the North AHB and the South AHB initiates every transaction on the SDRAM interface via the SDRAM Controller. Whenever an AHB Master initiates a data transfer to an address location mapped to the SDRAM memory access location, the SDRAM controller responds by decoding the address and the AHB transfer type.

After decoding is complete, the SDRAM Controller completes the read or write transaction to the SDRAM.

Byte and half-word transfers are implemented by controlling the DQM pins of the SDRAM. The SDRAM Controller performs byte lane steering for write operations to the SDRAM. Read operations performed by the SDRAM Controller to the SDRAM do not support byte-lane steering. Read operations performed by the SDRAM Controller to the SDRAM must be 32-bits. If sub-word accesses are requested a full 32-bits will be returned.

7.1 SDRAM Memory Space

The SDRAM memory space is defined with the base address beginning at hexadecimal 0x00000000 and ending at hexadecimal 0x3FFFFFFF.

The SDRAM memory space overlaps the expansion bus memory space during the boot sequence. Once the boot sequence has started, a configuration register — located in the expansion bus configuration space — must be written to remove the expansion bus mapping from the SDRAM space. Therefore enabling the SDRAM memory space to start in hexadecimal location 0x000000000.

The mapping of the SDRAM and Expansion Bus is described in detail in Section 8.9.9, “Configuration Register 0” on page 322.

Table 106. Memory Space

SDRAM Tech.	SDRAM Type	# Chips	Address Size		Total Mem. Size
			Row	Column	
64 Mbit	2 M x32	1	11	8	8 Mbyte
		2	11	8	16 Mbyte
64 Mbit	4 M x16	2	12	8	16 Mbyte
		4	12	8	32 Mbyte
128 Mbit	8 M x16	2	12	9	32 Mbyte
		4	12	9	64 Mbyte
256 Mbit	16 M x16	2	13	9	64 Mbyte
		4	13	9	128 Mbyte
512 Mbit	32 M x16	2	13	10	128 Mbyte
		4	13	10	256 Mbyte

During a read or write access, only one chip-select pin — SDR_CS_N (SDRAM Chip Select) — will be active at a time.

7.2 Initializing the SDRAM Controller

In order to use the SDRAM interface, the device must be configured properly. There are three configuration registers used to initialize the SDRAM Controller. Before using the SDRAM Controller, the SDRAM Configuration (SDR_CONFIG) Register must be initialized.

The SDRAM Configuration (SDR_CONFIG) Register will provide the CAS to data delay parameter and the external Memory Configuration. Bit 3 of the SDRAM Configuration (SDR_CONFIG) Register is used to specify the CAS to data delay. After reset, the CAS-

to-data delay will be initialized to two clocks. The initial value in bit 3 will be logic 0. If a CAS to data delay of three clocks is required, bit 3 of the SDRAM Configuration (SDR_CONFIG) Register must be set to logic 1.

Bits 2:0 of the SDRAM Configuration (SDR_CONFIG) Register are used to configure the SDRAM Controller to operate with a given physical memory configuration. Table 108 shows the values that are necessary to be programmed into these register bits for proper operation.

Table 107. Memory Configurations for Writing the SDRAM Configuration (SDR_CONFIG) Register

SDR_CONFIG [2:0]	Total memory	64 Mbit	64 Mbit	
000	8 Mbyte	One chip 2M x32		
001	16 Mbyte	Two chips 2M x32		
010	16 Mbyte		Two chips 4M x16	
011	32 Mbyte		Four chips 4M x16	
100	(Reserved)			
101	(Reserved)			
110	(Reserved)			
111	(Reserved)			

Notes:
1. Bit 5 of the sdr_config register is set to “1”
2. It is possible to have different values sent to the SDRAM status mode register from what is written to this register (CAS/RAS latencies). This may result in an undefined operation of the controller.
3. For more detail on the SDRAM Configuration register, see “Configuration Register” on page 287.

Table 108. Memory Configurations for Writing the SDRAM Configuration (SDR_CONFIG) Register

SDR_CONFIG [2:0]	Total memory	128 Mbit	256 Mbit	512 Mbit
000	32 Mbyte	Two chips 8 M x16		
001	64 Mbyte	Four chips 8 M x 16		
010	64 Mbyte		Two chips 16 M x 16	
011	128 Mbyte		Four chips 16 M x16	
100	128 Mbyte			Two chips 32 M x 16
101	256 Mbyte			Four chips 32 M x 16
110	Reserved			
111	Reserved			

Notes:
1. Bit 5 of the sdr_config register is set to “0”
2. It is possible to have different values sent to the SDRAM status mode register from what is written to this register (CAS/RAS latencies). This may result in an undefined operation of the controller.

Additionally, it is physically possible to have different memory configurations and types attached to the controller than what is written to these registers. Two banks would be attached when a one-bank configuration is written to SDR_CONFIG and one bank would be attached when a two-bank configuration is written.

The results are undefined in this case.

An example of configuring the SDRAM Configuration (SDR_CONFIG) Register is shown below:

1. Assume that the application being configured is a 256-Mbyte configuration using four chips (32 Mbyte x 16) and a CAS to data delay of three clocks. See Figure 54 for an SDRAM Connection Example of a similar configuration.
2. A hexadecimal value of 0x0000000Dis written to 0xCC000000.

The SDRAM Refresh (SDR_REFRESH) Register is used to determine the number of cycles before a mandatory refresh command is issued. The SDRAM Refresh (SDR_REFRESH) Register is a 16-bit register that is used to determine the termination count of the automatic refresh command. The counter used to implement the automatic refresh timer is 16 bits operating at 133 MHz. The counter size allows a maximum of 492 microseconds between mandatory refreshes.

When a hexadecimal value of 0x00000000 is programmed to this register, the automatic refresh capability of the SDRAM Controller will be disabled. When an automatic refresh is performed, the valid bits of the open page registers will be “invalid.” In addition, when the refresh timer expires and the SDRAM Controller issues an auto-refresh command, all pages are closed.

If the SDRAM controller is idle when the auto-refresh timer expires, the SDRAM Controller initiates the refresh operation the next clock after detecting a refresh request. If the SDRAM controller is busy processing a read or write transaction, the SDRAM Controller does not initiate the refresh operation until it completes the previous transaction. The default value of the SDRAM Refresh (SDR_REFRESH) Register is hexadecimal 0x0384.

An example of how to use the SDRAM Refresh (SDR_REFRESH) Register is shown below:

1. Assume that the application being configured requires an automatic refresh time of hexadecimal 0x00001AB5.
2. A hexadecimal value of 0x00001AB5 is written to 0xCC000004. Therefore a refresh would occur about every 51.4 microseconds.

The SDRAM Instruction (SDR_IR) Register is used to provide specific commands to the SDRAM that can be useful in configuration of the SDRAM initialization and operation. The SDRAM Instruction (SDR_IR) Register is a 3-bit register that contains a command decode. Table 109 shows the commands that can be used to produce specific operation over the SDRAM interface.

Table 109. SDRAM Command Description (Sheet 1 of 2)

Command Name	SDR_IR[2:0]	Description
Mode-Register-Set	000	Produces SDRAM cycles that set the SDRAM mode register with a CAS to data delay of 2, a write burst mode set to single location access (meaning reads will be use the burst length value and writes will be single location access), an operating mode set to standard operation, a burst type set to sequential, and a burst length set to a value of 8.
Mode-Register-Set	001	Produces SDRAM cycles that set the SDRAM mode register with a CAS to data delay of 3, a write burst mode set to single location access (meaning reads will be use the burst length value and writes will be single location access), an operating mode set to standard operation, a burst type set to sequential, and a burst length set to a value of 8.
Precharge-All	010	Used to pre-charge all banks of memory. A pre-charge is used to close all open banks of memory.
NOP	011	A command used to produce a null command to the SDRAM. This command is used during initialization and can be used to filter any spurious commands during idle or wait states.

Table 109. SDRAM Command Description (Sheet 2 of 2)

Command Name	SDR_IR[2:0]	Description
Auto-Refresh	100	Used to produce a refresh command to the SDRAM to avoid loss of data. The times between successive refresh commands is a function of the SDRAM that is chosen.
Burst Terminate	101	A command issued to the SDRAM to terminate a current fixed length burst.
(Reserved)	110	Writing to this location will cause undetermined results.
(Reserved)	111	Writing to this location will cause undetermined results.

Writing hexadecimal value 0x00000000 to address location 0xCC000008 will cause a Mode Register Set command to be initiated. See the chosen SDRAM Memory vendor's datasheet for Mode-Register-Set Timing with CAS to data delay of 2 to view the signaling associated with this command.

Writing hexadecimal value 0x00000001 to address location 0xCC000008 will cause a Mode Register Set command to be initiated. See the chosen SDRAM Memory vendor's datasheet for Mode-Register-Set Timing with CAS to data delay of 3 to view the signaling associated with this command.

Writing hexadecimal value 0x00000002 to address location 0xCC000008 will cause a Precharge All command to be initiated. See the chosen SDRAM Memory vendor's datasheet for Precharge-All Timing, to view the signaling associated with this command.

Writing hexadecimal value 0x00000003 to address location 0xCC000008 will cause a NOP command to be initiated. See the chosen SDRAM Memory vendor's datasheet for NOP Timing to view the signaling associated with this command.

Writing hexadecimal value 0x00000004 to address location 0xCC000008 will cause a Auto-Refresh command to be initiated. See the chosen SDRAM Memory vendor's datasheet for Auto-Refresh Timing to view the signaling associated with this command.

Writing hexadecimal value 0x00000005 to address location 0xCC000008 will cause a Burst Terminate command to be initiated. See the chosen SDRAM Memory vendor's datasheet for Burst Terminate Timing to view the signaling associated with this command.

Table 110 shows the values contained on the control signals for each command that is issued using the SDRAM Instruction Register (SDR_IR).

Table 110. SDRAM I/O For Various Commands

Command	SDR_IR[2:0]	SDR_CS_N	SDR_RAS_N	SDR_CAS_N	SDR_WE_N	SDR_DQM	SDR_ADDR
MODE-REGISTER-SET	000	0	0	0	0	X	CODE
MODE-REGISTER-SET	001						
PRECHARGE-ALL	010	0	0	1	0	X	CODE
NOP	011	0	1	1	1	X	X
AUTO-REFRESH	100	0	0	0	1	X	X
BURST TERMINATE	101	0	1	1	0	X	X

7.2.1 Initializing the SDRAM

Once the Intel XScale processor configures the SDRAM Configuration (SDR_CONFIG) Register and the SDRAM Refresh (SDR_REFRESH) Register, the following sequence of commands — using the SDRAM Instruction (SDR_IR) Register — must be performed to initialize the SDRAM. (This routine can change depending on the SDRAM part that is connected to the SDRAM interface.)

This routine is included for reference to demonstrate the initialization operation of an SDRAM:

- The memory controller applies the clock pin (SDM_CKE) during power up and must stabilize the clock signal within 100 µs after power stabilizes.
- The memory controller holds all the control pins to the memory inactive (SDM_RAS_N, SDM_CAS_N, SDM_WE_N, SDM_CS_N[1:0]=1) for a minimum of 1 millisecond after supply voltage reaches the desired level.
- SDM_CKE is driven to VCC all the time. The IXP42X product line and IXC1100 control plane processors never de-assert SDM_CKE.
- Software disables the refresh counter by setting SDR_REFRESH to zero.
- Software issues one NOP cycle after the 1milliseconds SDRAM device deselect. A NOP is accomplished by setting SDR_IR to 011. The memory controller asserts SDM_CKE with the NOP.
- Software pauses 200 µs after the NOP.
- Software re-enables the refresh counter by setting the SDR_REFRESH to the required value.
- Software issues a precharge-all command to the SDRAM interface by setting SDR_IR to 010.
- Software provides eight auto-refresh cycles. An auto-refresh cycle is accomplished by setting SDR_IR to 100. Software must ensure at least T_{rc} cycles between each auto-refresh command. T_{rc} (active-to-active command period) is determined by the SDRAM being used.
- Software issues a mode-register-select command by writing to SDR_IR to program the SDRAM parameters. Setting SDR_IR to 000 programs the SDRAM Controller for CAS Latency of two while setting the SDR_IR to 001 programs the memory controller and SDRAM for CAS Latency of three.
- The SDRAM Controller may issue a row activate command three clocks after the mode register set command.

Please refer to the chosen SDRAM Memory vendor's datasheet for SDRAM Initialization Access to view the operation on the SDRAM signals during the initialization sequence.

In addition to the above features, there is a set of eight registers used by the SDRAM Controller to manage up to eight open pages. These registers are called the SDRAM Page (SDR_PG) Registers.

The Intel XScale processor has the ability to read only the status of these registers. Signals associated with these registers are routed to the Internal Bus Performance Monitoring Unit (IBPMU). The Internal Bus Performance Monitoring Unit (IBPMU) can then be used directly to monitor the SDRAM Page Hit/Miss characteristics and Intel XScale processor code can be optimized to achieve the highest level of performance.

The memory controller may have eight memory pages open simultaneously (one per leaf). The SDRAM Controller supports devices containing four internal banks. These internal banks are defined as a leaf to help avoid confusion with a memory bank.

A page hit is valid if the memory location falls within the location as specified by the open page register:

Table 111. Page Register Allocation

SDM_CS_N[1:0] (Physical Bank)	SDM_BA[1:0] (Internal Bank)	Valid bit	Page Registers (13 bits)
Bank 0	Leaf 0		RAS address 0
Bank 0	Leaf 1		RAS address 1
Bank 0	Leaf 2		RAS address 2
Bank 0	Leaf 3		RAS address 3
Bank 1	Leaf 0		RAS address 4
Bank 1	Leaf 1		RAS address 5
Bank 1	Leaf 2		RAS address 6
Bank 1	Leaf 3		RAS address 7

If the RAS address for the current SDRAM access matches that stored within a valid page registers, then there is a page hit. When the current transaction hits an open page, then the page is already active. The read or write command may be issued without a row-activate command. It is important to note that when the refresh timer expires, the SDRAM Controller will issue a precharge command, which closes all pages, followed by the issue of an auto-refresh command.

When the current transaction misses, the open page (maintained by the page registers) is selected then the SDRAM controller closes the open page pointed to by issuing a precharge command. The SDRAM controller then opens the correct page with a row-activate command and the SDRAM Controller completes the requested transaction with a read or a write command. When the SDRAM Controller opens the new page, the RAS address is stored in the page address register. This new value stored in the page register may be used to compare for future transaction page hit/ misses.

The SDRAM controller interfaces to the AHB as a non-splitting bus slave. In so doing this, the SDRAM Controller follows a certain set of rules during any access. These rules can be helpful in understanding the performance and capability of the overall chip performance.

- The South AHB that host the Intel XScale processor will be the only master capable of writing to the configuration register of the SDRAM.
- AHBs will access the SDRAM in a pipe-lined fashion (bus accesses will be pipe-lined together whenever possible).
- The SDRAM controller will insert wait states back to an AHB Master if the SDRAM Controller is currently involved in an SDRAM transaction from the other AHB.
- If both AHB interfaces try to access the SDRAM at the same time (two transactions occur at the same time on both buses to access the SDRAM controller), the South AHB bus that host the Intel XScale processor will have priority.
- Transfers on both AHBs have a maximum of eight words and will be word aligned.
- Data transfers on both AHBs support data size listed in the Table 112 but will be word aligned. Therefore, the DQM(3:0) signals going to the SDRAM will always be 0x0, 0x1, 0x2, 0x3, 0x4, 0x6, 0x8, or 0xC. Any other values on DQM(3:0) will be unachievable.

Table 112. Data Transfer Sizes of AHB

Size	Description
8 bits	Byte
16 bits	Half word
32 bits	Word

7.3 SDRAM Memory Accesses

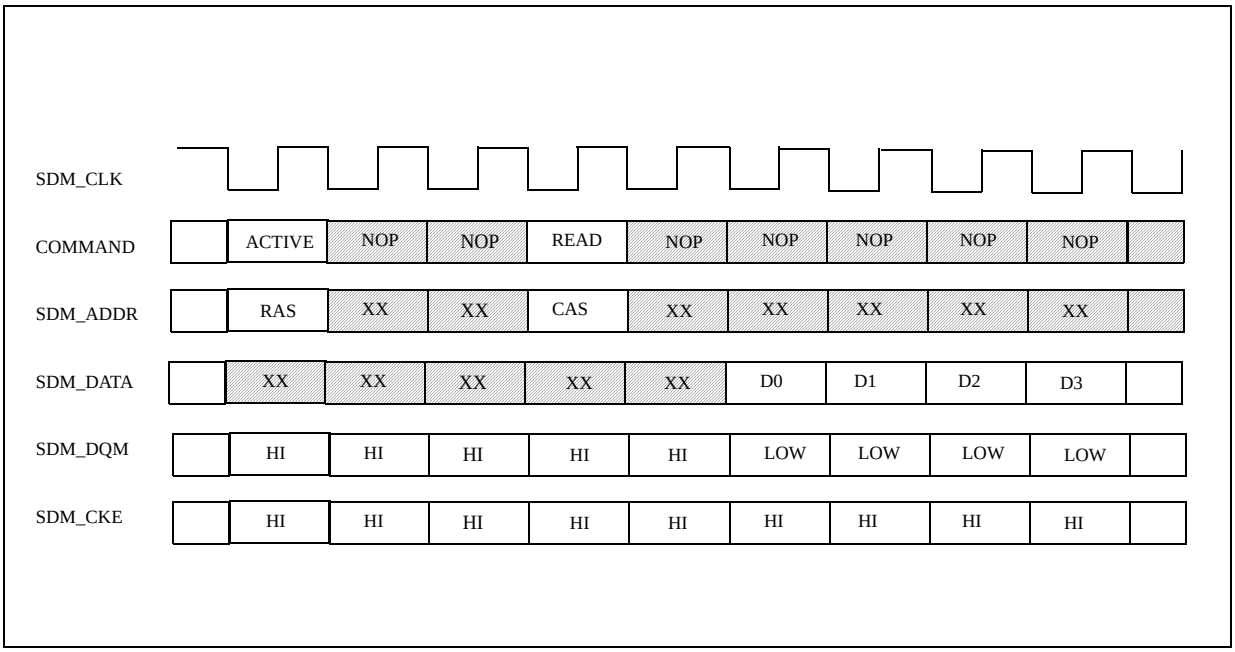
7.3.1 Read Transfer

When the AHBs generate a read transaction with an address located in the SDRAM space, a read from SDRAM is initiated. The SDRAM detects the read initiation request from the AHB. The SDRAM Interface control signals perform an SDRAM active cycle with the appropriate row address followed by a RAS-to-CAS delay. After a RAS-to-CAS delay of a three clocks, the SDRAM Interface control signal generates a read command and presents the column address. The control signal then waits for a number of clocks (CAS-to-data delay) before registering the data that is returned from the SDRAM. The SDRAM controller performs these reads until either the end of the transfer on the AHB or until the column address increments to hit a page-crossover condition. Upon such a crossover condition, the SDRAM Controller terminates the transaction by performing a Burst Terminate followed by pre-charge cycle and resumes the read transfer from the incremented address.

7.3.1.1 Read Cycle Timing (CAS Latency of Two Cycles)

Figure 55 shows the timing cycles on the SDRAM for a read cycle with a CAS Latency of 2 Cycles.

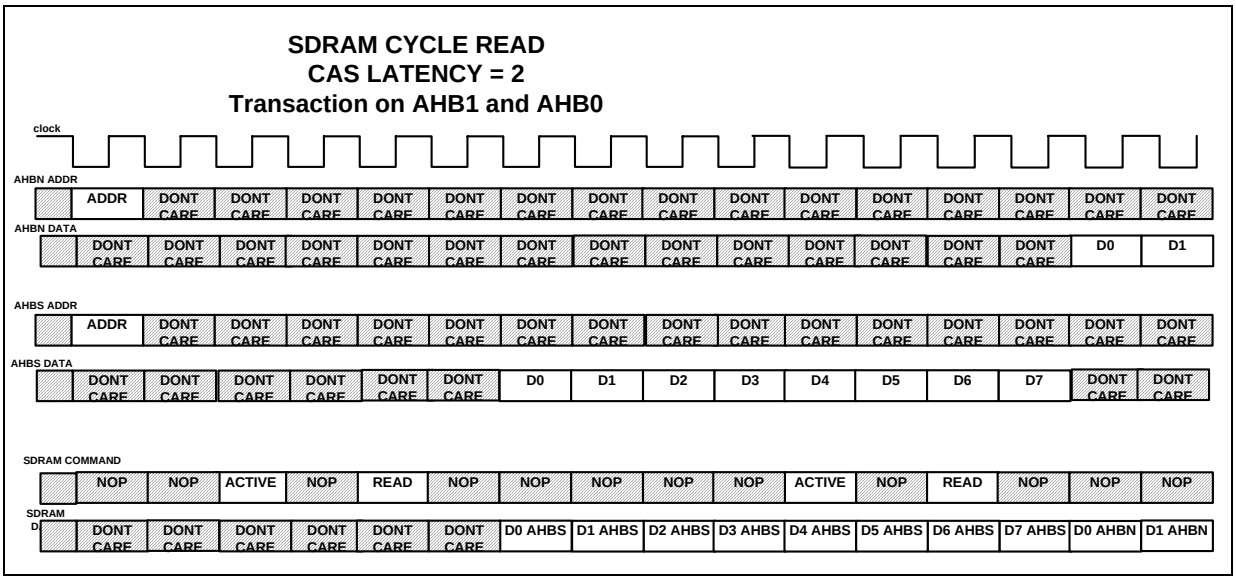
Figure 55. SDRAM Read Example (CAS Latency of 2 Cycles)



7.3.1.2 Read Burst Transfer (Interleaved AHB Reads)

The timing diagram in Figure 56 shows read requests from an NPE on the North AHB and the Intel XScale processor on the South. Both masters access different memory banks.

Figure 56. SDRAM Shared South AHB and North AHB Access



7.3.2 Write Transfer

The AHBs need to generate a write transaction with the address located in the SDRAM space in order to initiate a write from SDRAM. One of the AHB Masters addresses the SDRAM memory space. Upon detection of the write initiation, the control signal performs an active cycle with the appropriate row address, followed with a RAS-to-CAS delay.

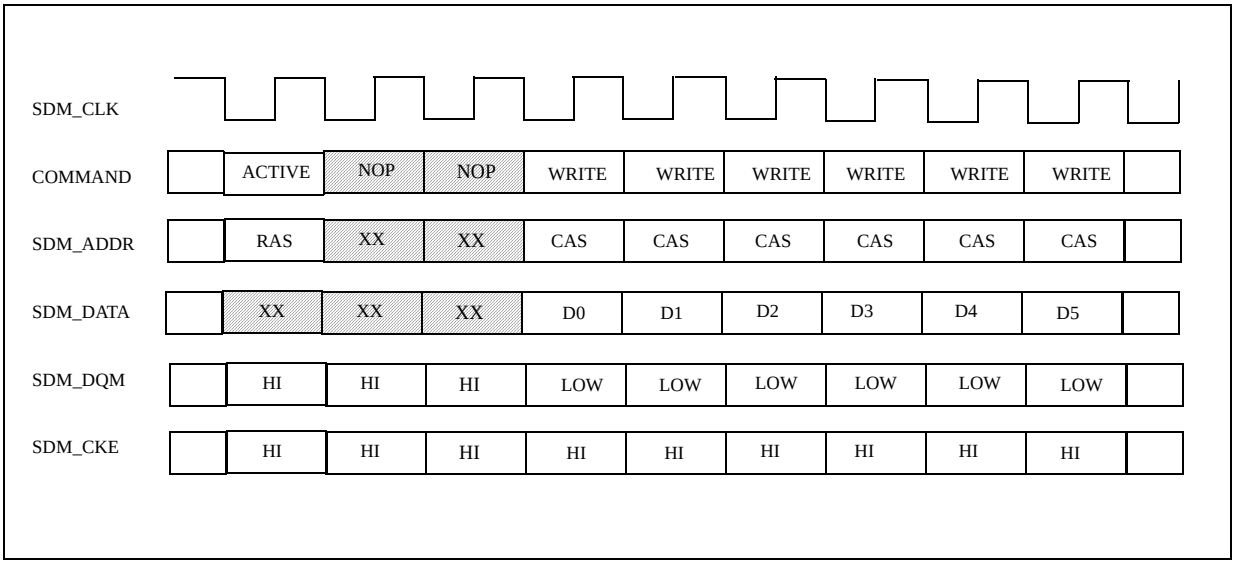
After a RAS-to-CAS delay of a finite number of clocks (determined by the latency setting – which has a default of three), the control signal generates a write command and presents the column address. The SDRAM controller performs such writes until either the end of the transfer on the AHB or until the column address increments to hit a page crossover condition.

Upon such a crossover condition, the SDRAM Controller terminates the transaction by performing a Burst Terminate followed by pre-charge cycle and resumes the read transfer from the incremented address.

7.3.2.1 Write Transfer

The timing diagram in Figure 57 shows cycles on the SDRAM for a write cycle.

Figure 57. SDRAM Write Example



7.4 Register Description

The IXP42X product line and IXC1100 control plane processors' SDRAM interface is programmed through a set of configuration registers that are described in the following sections. Many timing parameters are encoded as a number of SDM_CKE clock cycles.

The registers of the SDRAM controllers are 32-bit each. Each register access is 32-bits only (no byte or half-word access). Therefore if software running on the Intel XScale processor wishes to change one bit, it must read the entire contents of the register and write them back with that one bit changed. These registers are accessible via the South AHB interface. The table below shows the overview of the SDRAM controllers' addresses.

Table 113. SDRAM Register Overview

Register Name	R/W	Reset Hex Value	Hex Address	Description
sdr_config	R/W	0x00000010	0xCC000000	SDRAM Configuration Register
sdr_refresh	R/W	0x00000384	0xCC000004	SDRAM Refresh Register
sdr_ir	R/W	0x00000000	0xCC000008	SDRAM Instruction Register

7.4.1 Configuration Register

The configuration register (SDR_CONFIG) is a read/write register that contains control bits for configuring the SDRAM. The two physical SDRAM banks must be implemented with the same type of SDRAM devices.

Register Name:	SDR_CONFIG				
Hex Offset Address:	0xCC000000		Reset Hex Value:	0x00000010	
Register Description:	Configuration of the memory/memory controller.				
Access: Read/Write					

31	...	5	4	3	2		0
(Reserved)		64M en	RAS Lat	CAS Lat	Mem config		

Register		SDR_CONFIG
Bits	Name	Description
31:21	(Reserved)	Reserved
5	Enable 64Mbit	1 = 64Mbit, 0 = 128/256/512 Mbit chips
4	RAS Latency	1 = Three-cycle latency. This is hard-coded.
3	CAS Latency	1 = Three-cycle latency. Default is two-cycle latency.
2:0	Memory Config	Denotes memory size and type.
Note:	It is possible to have different values sent to the SDRAM status mode register from what is written to this register (cas/ras latencies). This may result in an undefined operation of the controller.	

7.4.2 Refresh Register

The refresh register (SDR_REFRESH) is a read/write register and contains control bits for refresh of the SDRAM banks. It holds the number of cycles before the Intel XScale processor issues a mandatory refresh command. The SDRAM refresh interval field applies to all types of SDRAM (asynchronous and synchronous).

Register Name:	SDR_REFRESH				
Hex Offset Address:	0xCC000004		Reset Hex Value:	0x00000384	
Register Description:	Refresh register that holds Number of cycles before a mandatory refresh command is issued.				
Access: Read/Write					

31	...	16	15	...	0
(Reserved)			Refresh Time		

Register		SDR_REFRESH
Bits	Name	Description
31:16		(Reserved)
15:0	Refresh time	Number of cycles before a mandatory refresh command is issued. Defaults to 900 cycles.

7.4.3 Instruction Register

The instruction register is a read/write register that holds commands that are used to determine the operation mode of the SDRAM controller and the mode register of the SRDAM devices.

Register Name:	SDR_IR				
Hex Offset Address:	0xCC000008		Reset Hex Value:	0x00000000	
Register Description:	Instruction register, holds commands that determine operation mode of the SDRAM controller and mode register for SDRAMs.				
Access: Read/Write					

31														16	15										3	2		0
(Reserved)																										Instruction (command)		

Register		SDR_IR
Bits	Name	Description
31:3		(Reserved)
2:0	Instruction Register	Commands to be sent out to the SDRAM. 000 - **Mode-Register-Set Command** where CAS# Latency 2. 001 - **Mode-Register-Set Command** where CAS# Latency 3. 010 - **Precharge-All Command**: The MCU issues one **precharge-all** command to the SDRAM devices. 011 - **NOP Command**: The MCU issues one **NOP** command to the SDRAM devices. 100 - **Auto-Refresh Command**: The MCU issues one **auto-refresh** command to the SDRAM devices. 101 - **Burst Terminate Command**: The MCU issues one Burst Terminate command to the SDRAM devices. This is intended to be used after reset only. 11x - **(Reserved)**

A “set mode register” command would write the following to the SDRAM. This is a standard definition of a mode register from an SDRAM and not a register within the SDRAM controller:

Table 114. SDRAM Configuration Options

[15:11]	Don’t Care	Don’t Care
10	Write Burst Mode	0 = Programmed burst length 1 = Single-location access
9:7	Operating Mode	00 = Standard operation All other values reserved.
6:4	CAS Latency	010 = For two cycles 011 = For three cycles All other values are reserved.
3	Burst Type	0 = Sequential 1 = Interleaved ***Note:*** This value is hard-coded to 0.
2:0	Burst Length	To set this value, refer to Table 115.

Table 115. SDRAM Burst Definitions

Burst Length	Starting Column Address	Order of Accesses within a Burst	
		Type = Sequential	Type = Interleaved
2	A0		
	0	0-1	0-1
	1	1-0	1-0

Table 115. SDRAM Burst Definitions

Burst Length	Starting Column Address			Order of Accesses within a Burst	
				Type = Sequential	Type = Interleaved
4		A1	A0		
		0	0	0-1-2-3	0-1-2-3
		0	1	1-2-3-0	1-0-3-2
		1	0	2-3-0-1	2-3-0-1
		1	1	3-0-1-2	3-2-1-0
8	A2	A1	A0		
	0	0	0	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7
	0	0	1	1-2-3-4-5-6-7-0	1-0-3-2-5-4-7-6
	0	1	0	2-3-4-5-6-7-0-1	2-3-0-1-7-6-5-4
	0	1	1	3-4-5-6-7-0-1-2	3-2-1-0-7-6-5-4
	1	0	0	4-5-6-7-0-1-2-3	4-5-6-7-0-1-2-3
	1	0	1	5-6-7-0-1-2-3-4	5-4-7-6-1-0-3-2
	1	1	0	6-7-0-1-2-3-4-5	6-7-5-4-2-3-0-1
	1	1	1	7-0-1-2-3-4-5-6	7-6-5-4-3-2-1-0

These are the commands issued by the memory controller to the SDRAM and are not accessible through the AHB (i.e. a master cannot issue these commands on the internal bus to the memory controller unit).

Table 116. SDRAM Commands

Command	CS#	RAS#	CAS#	WE#	DQM	ADDR	DATA
COMMAND INHIBIT	H	X	X	X	X	X	
NO OPERATION	L	H	H	H	X	X	
ACTIVE	L	L	H	H	X	Bank/Row*	
READ	L	H	L	H	X	Bank/Col	
WRITE	L	H	L	L	X	Bank/Col	VALID
BURST TERMINATE	L	H	H	L	X	X	ACTIVE
PRECHARGE	L	L	H	L	X	CODE	
AUTO REFRESH OR SELF REFRESH	L	L	L	H	X	X	
LOAD MODE REGISTER	L	L	L	L	X	BA = 00 ADDR = opcode	
WRITE ENABLE/ OUTPUT ENABLE	-	-	-	-	L	-	ACTIVE
WRITE INHIBIT/ OUTPUT HIGH-Z	-	-	-	-	H	-	HIGH-Z

Note: Bank is used in the “SDRAM” sense, synonymous to LEAF in the rest of the documentation.

§ §

8.0 Expansion Bus Controller

The Expansion Bus Controller provides an interface from internal South AHB to external flash, Host-Port Interfaces (HPI), SRAM and other devices such as ATM control interfaces, and DSPs used for voice applications.

The Expansion Bus includes a 24-bit address bus and a 16-bit-wide data path and maps transfers between the South AHB and external devices. Intel and Motorola*, multiplexed and non-multiplexed, micro-controller-style address/data bus accesses are both supported using the expansion interface. Applications having less than 16-bit external data paths may connect to an 8-bit interface.

The Expansion Bus Controller occupies 256 Mbytes of address space in the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor' memory map. Eight chip selects are supported to allow up to eight independent external devices to be connected. The address space for each chip select is up to 16 Mbytes.

A clock input is required to operate the expansion interface. The maximum clock frequency supported by the Expansion Bus Controller is 66.66 MHz. The clock input is provided to allow a wide variety of different peripherals to be connected to the expansion interface.

GPIO 15 provides a clock output after reset. The clock produced by GPIO 15 is programmable at speeds up to 33.33 MHz and can be used to provide the clock input to the expansion bus interface. GPIO 15 must be externally routed on the board to connect to EX_CLK.

This implementation gives the designer the option to choose between a lower part count and the speed of the interface operations.

To provide a glueless interface to a wide variety of devices, the Expansion Bus Controller supplies eight chips selects to a 16-bit-wide external bus, which can be configured as Intel, Motorola, or HPI-style controls. The signaling characteristics and timing for each chip select is individually programmable. After chip reset, chip-select 0 defaults to conservative timing values for controlling a flash device and the size of the flash is determined by the value of Expansion Bus Address bit 0 at the de-assertion of RESET_IN_N signal. The Expansion Bus address bit all have internal pull-up resistors. Each bit may be pulled low by placing a pull-down resistor on the address signal. The remaining chip selects are un-programmed.

The Expansion Bus Controller contains configuration registers beyond what is required for its own configuration. There are several bits of configuration signals provided as output from the Expansion Bus Controller to the rest of the Intel® IXP42X product line and IXC1100 control plane processors. These signals provide the AHB with function like the software interrupt capabilities, location of Expansion Bus Controller in IXP42X product line and IXC1100 control plane processors' memory map, PCI Host/Arbiter information, and configuration information on devices connected to the Expansion Bus and SDRAM Controller.

One of these general-purpose configuration registers is used to capture the value on the address pins immediately after reset. In the Expansion Bus Interface, 24 address lines are used to capture this configuration information at the release of reset. When power up is complete and reset is asserted, the 24 address lines are configured as inputs. When reset is released (RESET_IN_N), the configuration registers capture the values contained on the 24 address lines and the 24 address lines become configured as outputs. The Expansion Bus address signals have internal pull-up resistors of about 50 KΩ. Pull-down resistors may be placed on some of the 24 address lines in order to select the configuration values at reset. Please refer to Section 8.9.9, “Configuration Register 0” on page 322 for additional details on particular configuration options.

In normal mode of operation, the 24 bit address bus is used to present the 24 bits of the address [23:0] used for the requested transaction accompanied by an address latch enable output signal, (EX_ALE). The address phase normally last one clock cycle in non-multiplexed mode and two clock cycles in multiplexed mode but may be extended by one to three clock cycles using the T1 parameter in the Timing and Configuration Register for the particular Chip Select.

The remaining control signals used by the Expansion Bus Interface can be configured to operate as Intel, Motorola, or HPI-style control signals. As an example, the write strobe for Intel mode (EX_WR_N) would be translated to the data strobe (DS_N) when placed into Motorola mode. Similarly, the Intel-mode read signal (EX_RD_N) would become read-not-write (R/W_N) when placed in Motorola* mode.

The EX_IOWAIT_N signal is available to be shared by the devices attached to chip 0 through 7, when the chip selects are configured in Intel or Motorola mode of operation. The EX_IOWAIT_N signal allows an external device to hold off completion of the read phase of a transaction until the external device can supply the data requested.

Similarly, EX_RDY[3:0] are provided for chip selects 7 through 4, respectively. The EX_RDY[3:0] signals are used to hold off data transfers when chip selects 7 through 4 are configured in HPI mode. For example when chip select 5 is configured in HPI mode of operation. Chip select 5 will no longer respond to the EX_IOWAIT_N signal and will only respond to the EX_RDY_N[1]. All other chip selects will respond to the EX_IOWAIT_N signal. Chip selects 7 through 4 are the only chip selects that can be configured in HPI mode of operation.

8.1 Expansion Bus Address Space

Table 117. Processors’ Trimmed Version of the Memory Map

Start Address	End Address	Size	Use
0000_0000	0FFF_FFFF	256 Mbyte	Expansion bus
0000_0000	3FFF_FFFF	1 Gbyte	SDRAM
4000_0000	47FF_FFFF	128 Mbyte	(Reserved)
4800_0000	4FFF_FFFF	128 Mbyte	PCI
5000_0000	5FFF_FFFF	256 Mbyte	Expansion bus
6000_0000	63FF_FFFF	64 Mbyte	Queue manager

As seen in Table 117, on the South AHB, the lowest 256 Mbytes of address space (0x00000000 to 0x0FFFFFFF) is overlapped with the SDRAM address space (0x00000000 to 0x3FFFFFFF). The actual interface that is accessed when the overlapped region is addressed is configurable based on the value of a configuration register bit located in the Expansion Bus Controller.

When bit 31 of the Configuration Register 0 (EXP_CNFG0) is set to logic 1, the Expansion Bus accesses occupy the lowest 256-Mbytes of address space. When bit 31 of the Configuration Register 0 (EXP_CNFG0) is set to logic 0, the SDRAM occupies the lowest 256 Mbytes of address.

On reset, bit 31 of the Configuration Register 0 (EXP_CNFG0) is set to logic 1. This setting is required to allow the boot memory to be accessed which is located at hexadecimal address 0x00000000 in non-volatile storage on the Expansion Bus.

The first instruction execution of the Intel XScale® Processor is located at address 0x00000000. Once the boot sequence starts, the Intel XScale processor will switch bit 31 of the Configuration Register 0 (EXP_CNFG0) from logic 1 to logic 0, at an appropriate time.

The information transfer from the flash to the SDRAM can be completed in one of two ways:

- The configuration bit can be swapped to allow the SDRAM to have access at address 0x00000000 and the remainder of the flash information can be retrieved from the expansion-bus address location 0x50000000 to 0x5FFFFFFF
- The SDRAM can be written by writing to the aliased sections of the SDRAM address space.

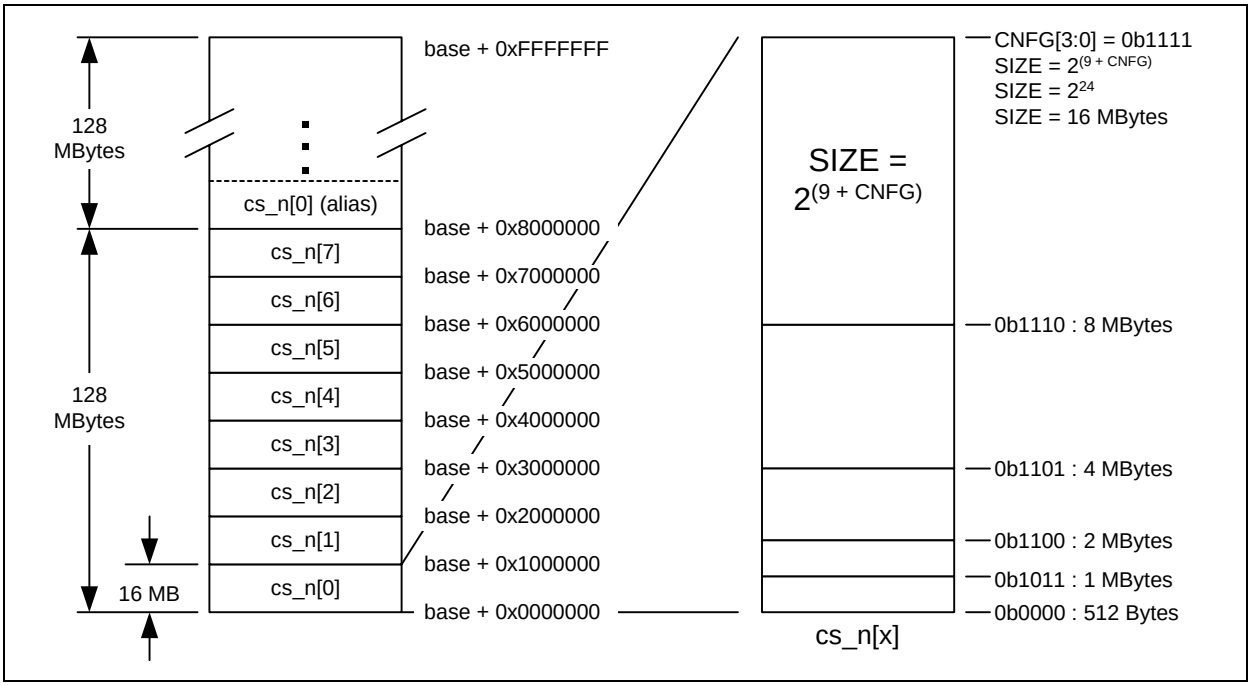
The SDRAM only supports a maximum of 256 Mbytes of addressable memory space. The remaining three memory locations (0x10000000 to 0x1FFFFFFF, 0x20000000 to 0x2FFFFFFF, 0x30000000 to 0x3FFFFFFF) — in the 1-Gbyte address space defined by the IXP42X product line and IXC1100 control plane processors' Memory Map — are re-mapped to the address space located at 0x00000000 to 0x0FFFFFFF.

8.2 Chip Select Address Allocation

The Expansion Bus Controller occupies 256 Mbytes of address space in the IXP42X product line and IXC1100 control plane processors memory map. The Expansion Bus Controller uses bits 27:0, from the South AHB, to determine how to translate the South AHB address to the Expansion Bus Address. The lower 24 bits of the South AHB address are translated to the lower 24 bits of the Expansion Bus address, EX_ADDR [23:0].

Bits 26:24 of the South AHB are used to decode one of eight chip-select regions implemented by the expansion bus, each region being 16 Mbytes. Address bit 27 is not used and will currently alias each chip select region as shown on the left side of Figure 58.

Figure 58. Chip Select Address Allocation



The right side of Figure 58 shows the implementation of bit 13:10 of the each Timing and Control (EXP_TIMING_CS) Register. A Timing and Control (EXP_TIMING_CS) Register is implemented for each of the eight chip selects. Each chip select defines a base region size of 512 bytes with the actual size of the region given by the formula shown in Figure 59.

Figure 59. Expansion Bus Memory Sizing

Region Size = $2^{(9+CNFG[3:0])}$

For Examples of how to use this feature:

If bits 13:10 of Timing and Control (EXP_TIMING_CS0) Register 0 = “0000” an address space of 2^9 = 512 Bytes is defined for chip select 0 (EX_CS0_N).

If bits 13:10 of Timing and Control (EXP_TIMING_CS1) Register 1 = “1000” an address space of 2^{17} = 128KBytes is defined for chip select 1 (EX_CS1_N).

If bits 13:10 of Timing and Control (EXP_TIMING_CS7) Register 7 = “1111” an address space of 2^{24} = 16Mbytes is defined for chip select 7 (EX_CS7_N).

8.3 Address and Data Byte Steering

Table 118 shows the address and data mapping from the South AHB to the Expansion Bus. This table applies to Intel- and Motorola-defined cycles only. Note that — for 32-bit operations — only read cycles are permissible. Also, for 32-bit read operations to a byte wide interface. Multiple bytes are collected and then transferred as a complete 32-bit word. This pattern occurs as shown below for any allowable sub-length read access.

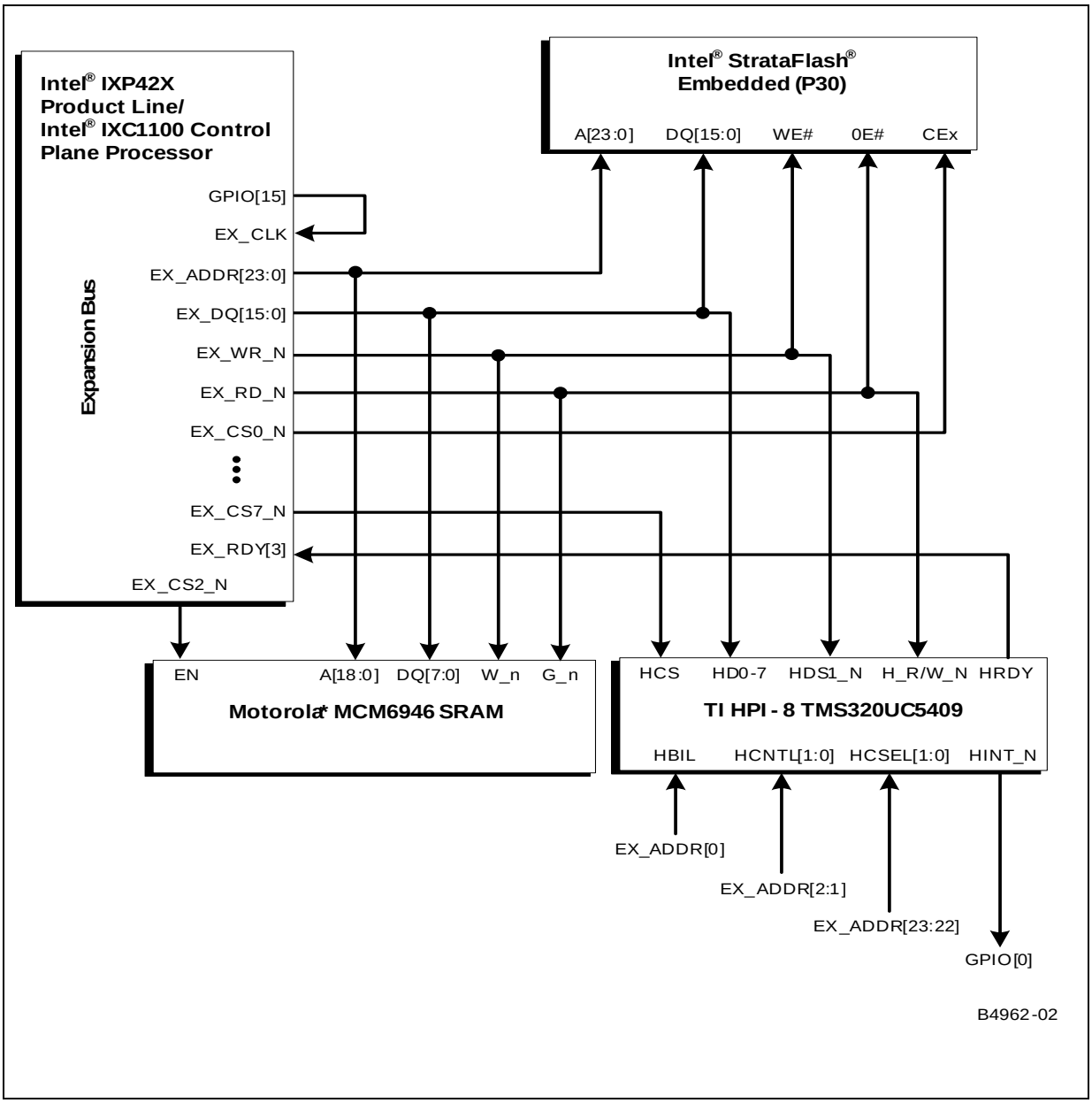
Table 118. Expansion Bus Address and Data Byte Steering

South AHB Bus Cycle	Device Connected to Expansion Bus (8-bit or 16-bit)	South AHB Address Value (SAHB_ADDR[1:0])	Expansion Bus Address Value (EX_ADDR[1:0])	Data Location Translation Between Expansion Data Bus and South AHB Data Bus
32-bit write	8-bit			Not allowed.
32-bit write	16-bit			Not allowed.
32-bit read	8-bit	00	00	AHB data bus [31:24] ← Expansion data bus [7:0]
			01	AHB data bus [23:16] ← Expansion data bus [7:0]
			10	AHB data bus [15:8] ← Expansion data bus [7:0]
			11	AHB data bus [7:0] ← Expansion data bus [7:0]
32-bit read	16-bit	00	0x	AHB data bus [31:16] ← Expansion data bus [15:0]
			1x	AHB data bus [15:0] ← Expansion data bus [15:0]
16-bit read	8-bit			Not allowed.
16-bit read	16-bit	0x	0x	AHB data bus [31:16] ← Expansion data bus [15:0]
	16-bit	1x	1x	AHB data bus [15:0] ← Expansion data bus [15:0]
16-bit write	8-bit			Not allowed.
16-bit write	16-bit	0x	0x	AHB data bus [31:16] → Expansion data bus [15:0]
	16-bit	1x	1x	AHB data bus [15:0] → Expansion data bus [15:0]
8-bit read	8-bit	00	00	AHB data bus [31:24] ← Expansion data bus [7:0]
	8-bit	01	01	AHB data bus [23:16] ← Expansion data bus [7:0]
	8-bit	10	10	AHB data bus [15:8] ← Expansion data bus [7:0]
	8-bit	11	11	AHB data bus [7:0] ← Expansion data bus [7:0]
8-bit read	16-bit			BYTE_RD16 dependant
8-bit write	8-bit	00	00	AHB data bus [31:24] → Expansion data bus [7:0]
	8-bit	01	01	AHB data bus [23:16] → Expansion data bus [7:0]
	8-bit	10	10	AHB data bus [15:8] → Expansion data bus [7:0]
	8-bit	11	11	AHB data bus [7:0] → Expansion data bus [7:0]
8-bit write	16-bit			Not allowed.
This will cause an AHB error that will result in a data-abort error.				

8.4 Expansion Bus Connections

Figure 60 shows a typical connection for various devices connected on the expansion bus. Note that GPIO(0) is used as an example, and that your design can use any GPIO port. Also note that EX_CS2_N and EX_CS7_N are shown as examples, and that your design can use any EX_CS_N port.

Figure 60. Expansion Bus Peripheral Connection



8.5 Expansion Bus Interface Configuration

There are eight registers — called the Timing and Control (EXP_TIMING_CS) Registers — that define the operating mode for each chip select. When designing with the Expansion Bus Interface, placing the devices on the correct chip selects is required.

Chip Select 0 through 7 can be configured to operate with devices that require an Intel or Motorola Micro-Processor style bus accesses. These chip selects can be configured to operate in a multiplexed or a simplex mode of operation — for either Intel- or Motorola-style bus accesses. Additionally, Chip Select 4 through 7 can be configured to generate Texas Instruments* HPI-style bus accesses. The mode of operation (Intel, Motorola, or TI HPI) is set by bits 15:14 of each Timing and Control (EXP_TIMING_CS) Register. Table 119 shows the possible settings for the Cycle Type selection using bits 15:14 of the Timing and Control (EXP_TIMING_CS) Register.

Table 119. Expansion Bus Cycle Type Selection

Bit[15:14]	CYC_TYPE
00	Configures the Expansion Bus Chip Select (x) for Intel cycles
01	Configures the Expansion Bus Chip Select (x) for Motorola* cycles.
10	Configures the Expansion Bus Chip Select (x) for TI* HPI cycles (Only valid for Chip Select 4 through 7)
11	(Reserved)
Note: (x) Can be 0 through 7.	

Once the cycle type has been determined, the mode of operation must be set. There are two configurable modes of operation for each chip select, multiplexed and non-multiplexed. Bit 4 of the Timing and Control (EXP_TIMING_CS) Registers is used to select this mode. If bit 4 of the Timing and Control (EXP_TIMING_CS) Register is set to logic 1, the access mode for that Chip Select is multiplexed. Likewise, if bit 4 of the Timing and Control (EXP_TIMING_CS) Register is set to logic 0, the access mode for that Chip Select is non-multiplexed.

Multiplexed and non-multiplexed can imply different operations depending upon the Cycle Type that is selected. For more information refer to section “Expansion Bus Interface Access Timing Diagrams” on page 305.

The size of the data bus for each device connected to the expansion bus must be configured. The data bus size is selected on a per-chip-select basis, allowing the most flexibility when connecting devices to the expansion bus.

There are two valid selections that can be configured for each data bus size, 8-bit or 16-bit. Bit 0 of each Timing and Control (EXP_TIMING_CS) Register is used to select the data bus size on a per-chip-select basis. When bit 0 is set to logic 0, the data bus width for the given chip select will be set to 16-bits. When bit 0 is set to logic 1, the data bus width for the given chip select will be set to 8-bits.

One special case for the data bus width selection is for chip select 0. Chip select 0 (data bus width) is selected by the value contained on Expansion Bus Address bit 0 at the de-assertion of reset. At the de-assertion of reset, Expansion Bus Address bit 0 will be captured into Timing and Control (EXP_TIMING_CS) Register for Chip Select 0. This feature allows either an 8-bit or 16-bit flash device to be connected to the Expansion Bus Interface for a boot device.

Each chip select can be independently enabled or disabled by setting a value in bit 31 of each Timing and Control (EXP_TIMING_CS) Register. Setting bit 31 — of the Timing and Control (EXP_TIMING_CS) Register — to logic 0 disables the corresponding chip select. Setting bit 31 — of the Timing and Control (EXP_TIMING_CS) Register — to logic 1 enables the corresponding chip select.

Split transfers are supported for all transfer types and controlled by setting bit 3 of the Timing and Control (EXP_TIMING_CS) Register. Multi-word read transfers requested by the South AHB might be split. Only one access at a time may be split.

These transfers require that the read data from the expansion bus be stored in an eight-word FIFO — until all expansion-bus transfers are complete — before that data is forwarded on the South AHB. When split transfers are initiated, the Expansion Bus Controller acknowledges the read request. The South AHB will be relinquished until all the data is acquired from the expansion bus and stored in the eight-word FIFO contained in the Expansion Bus Controller.

After all of the data has been acquired by the Expansion Bus Controller, the requesting master on the South AHB will be signaled that the read data is in the FIFO and the read transfer will complete — uninterrupted in its normal rotation in the arbitration scheme. This feature allows for slow devices — connected to the expansion bus — not to impede the performance of data flow from high-speed peripherals (like PCI) on the South AHB.

Retries also are supported and used predominately when expansion-bus requests are issued while a split transfer is in progress. Setting bit 3 — of each Timing and Control (EXP_TIMING_CS) Register — to logic 1 enables split transfers for accesses to the corresponding chip select. Setting bit 3 — of each Timing and Control (EXP_TIMING_CS) Register — to logic 0 disables split transfers for accesses to the corresponding chip select.

For Chip Select 0, split transfers are disabled after reset. This feature allows the boot device to provide uninterrupted 32-bit data words to the Intel XScale processor.

Each chip select region has the ability to be write-protected by setting bit 1 of each Timing and Control (EXP_TIMING_CS) Register. When bit 1 of Timing and Control (EXP_TIMING_CS) Register is set to logic 0, writes to a specified chip select region are ignored. When bit 1 of Timing and Control (EXP_TIMING_CS) Register is set to logic 1, writes are allowed to a specified chip select region. Chip select 0 will be write-protected after reset.

For chip selects 4 through 7 configured in HPI mode of operation, there is an associated ready bit (EX_RDY [0:3]). The ready bit is only used when the mode of operation is set to Texas Instruments HPI mode. The ready bits are used to hold off the Intel XScale processor when the given DSP is not ready to complete the transfer.

However, the polarity of this ready bit can vary based upon the DSP that is selected. Bit 5 of each Timing and Control (EXP_TIMING_CS) Register allows the polarity used by each ready bit to be independently set. When bit 5 — of the Timing and Control (EXP_TIMING_CS) Register — is set to logic 0, the ready bit is set to respond to an active low signal (logic 0). When bit 5 — of the Timing and Control (EXP_TIMING_CS) Register — is set to logic 1, the ready bit is set to respond to an active high signal (logic 1).

One final set of parameters that may be set prior to using Expansion Bus Interface Chip Select 1 through Chip Select 8. After boot up, these parameters may be adjusted for Chip Select 0 as well. These five parameters are the timing extension parameters for each phase of an Expansion Bus access.

There are five phases to every Expansion Bus access:

- T1 – Address Timing

- T2 – Setup/Chip Select Timing
- T3 – Strobe Timing
- T4 – Hold Timing
- T5 – Recovery Phase

The expansion-bus address is used to present the 24 bits of the address [23:0] used for the expansion bus access accompanied by an address latch enable output signal, EX_ALE. The address phase normally last one clock cycle, in non-multiplexed mode, and two clock cycles, in multiplexed mode. The address phase may be extended by one to three clock cycles using the T1 – Address Timing parameter, bits 29:28 in the Timing and Control (EXP_TIMING_CS) Register for the particular Chip Select. When the address phase T1 is extended, the ALE pulse is extended and always de-asserts one cycle prior to the end of the T1 phase.

In multiplexed mode only, the EX_ALE signal is asserted with the address at the beginning of the address phase and de-asserted one clock cycle later to provide plenty of address setup time to an external device or latch. The address is placed onto the 16-bit data bus — along with EX_ADDR [0:7] signals during the first cycle of the address phase — when using the ALE signal. The lower 16 bits of address are placed on the data bus and the upper bits of address are placed on the address bus signals EX_ADDR [23:16]. The ALE is used to capture the address signals.

During the second cycle of the address phase, the data bus now will output data — when attempting to complete a write — or tristate — when attempting to complete a read. The address signals will retain their state.

Due to the fact that, in HPI mode of operation, it is possible to begin an access to a busy device (EX_RDY is false), special consideration must be taken with programming the T1 – Address Timing parameter when using the chip select in HPI mode. The T1 – Address Timing parameter must be set to a minimum of two additional cycles (T1 must equal to 0x2). Programming the T1 – Address Timing parameter to this value ensures that the asynchronous EX_RDY input is sampled and available to the controlling hardware logic before beginning the new HPI access over the expansion bus.

The chip-select signal is presented for one expansion bus phase before the Strobe Phase. The chip select will be presented for the remainder of the expansion bus cycles (setup, strobe, and hold phases).

The Setup/Chip Select Timing phase may also be extended by one to three clock cycles, using bits 27:26 of the Timing and Control (EXP_TIMING_CS) Register, T2 – Setup/Chip Select Timing parameter. In HPI mode of operation, T2 is defined as the time required by the external DSP device to drive EX_RDY false for the current access plus the time required by the Expansion Bus Controller to sample and synchronize the EX_RDY signal. The T2 – Setup/Chip Select Timing parameter must have a minimum value of two additional cycles (T2 must equal 0x2). Programming the T2 – Setup/Chip Select Timing parameter to be three clock cycles in length ensures that when the Strobe Phase, T3, begins, the Strobe Phase will be able to sample the EX_RDY signal and exit the Strobe Phase at the proper time.

The Strobe Phase of an expansion-bus access is when the read or write strobe is applied. The 24 Expansion Bus Interface Address bits are maintained in non-multiplexed mode or the Expansion Bus Interface Data bus is switched from address to data when configured in multiplexed mode during the Strobe Phase.

The Strobe Phase may be extended from one to 15 clock cycles, as defined by programming bits 25:22 of the Timing and Control (EXP_TIMING_CS) Register, T3 – Strobe Timing parameter. In HPI mode of operation, the T3 – Strobe Timing parameter must have a minimum value of one additional cycle (T3 must equal 0x1). Programming

the T3 – Strobe Timing parameter to be two clock cycles in length ensures that any data sent to the DSP is captured regardless of when the EX_RDY signal is asserted by the DSP.

The Hold Phase of an expansion-bus access is provided to allow a hold time for data to remain valid after the data strobe has transitioned to an invalid state. During a write access, the Hold Phase provides hold time for data written to an external device on the expansion bus, after the strobe pulse has completed.

During a read access, the Hold Phase allows an external device time to release the bus after driving data back to the controller. The Hold Phase may be extended one to three clock cycles, using bits 21:20 of the Timing and Control (EXP_TIMING_CS) Register, T4 – Hold Timing parameter. In HPI mode of operation, the Hold Phase is defined the same as described for the Intel and Motorola* modes of operation.

After the address and chip select is de-asserted, the Expansion Bus Controller can be programmed to wait a number of clocks before starting the next Expansion Bus access. This action is referred to as the Recovery Phase. The Recovery Phase is may be extended one to 15 clock cycles using bits 19:16 of the Timing and Control (EXP_TIMING_CS) Register, T5 – Recovery Timing parameter. In HPI mode of operation, the Recovery Phase is defined the same as described for the Intel and Motorola modes of operation.

8.6 Using I/O Wait

The EX_IOWAIT_N signal is available to be shared by devices attached to chip selects 0 through 7, when configured in Intel or Motorola modes of operation. The main purpose of this signal is to properly communicate with slower devices requiring more time to respond during data access. During idle cycles, the board is responsible for ensuring that EX_IOWAIT_N is pulled-up. The Expansion bus controller will always ignore EX_IOWAIT_N for synchronous Intel mode writes.

As shown in Figure 61, a normal phase transaction is initiated during the T1 (Address Timing) period, in which the processor drives the address lines with an address that is decoded by the peripheral being accessed.

The next segment of the transaction is the T2 (Chip Select Timing) period, in which the processor asserts Chip Select and the Address signals have reached a stable state. EX_IOWAIT_N must be asserted during the T2 period. If not asserted at this time, the processor ignores EX_IOWAIT_N and treats it as it never occurred. If EX_IOWAIT_N is asserted during T2, the processor expects the signal to be deasserted during the T3 (Strobe Timing) period.

T3 can be programmed from 0 to F, where the value indicates the number of cycles that the processor waits for EX_IOWAIT_N to be deasserted. The counter starts at the rising edge of the clock when EX_RD_N or EX_WR_N is asserted. The following rules describe processor operation during T3:

- If EX_IOWAIT_N is deasserted at least two clock cycles before the T3 counter expires, then the processor deasserts EX_RD_N/EX_WR_N at the end of the number of cycles programmed in T3.
- If EX_IOWAIT_N is deasserted after T3 counter has expired, it will take 2 more clock cycles before the processor deasserts the EX_RD_N/EX_WR_N signal.
- If EX_IOWAIT_N is not deasserted before the T3 counter expires, then EX_RD_N/EX_WR_N will continue to be asserted for as long as EX_IOWAIT_N continues to be asserted, plus 2 more clock cycles.

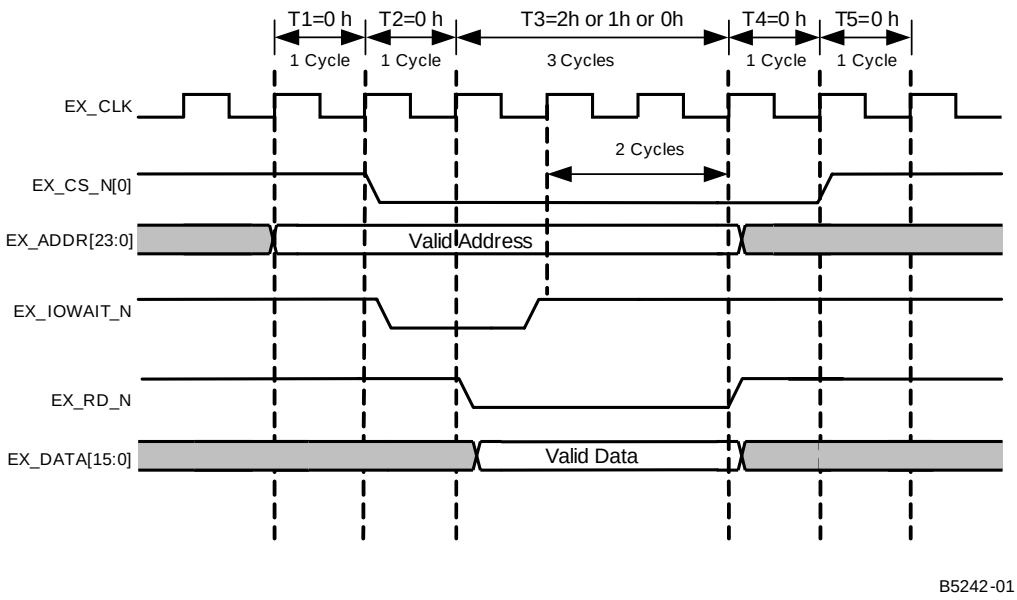
The T4 (Hold Timing) period is the time interval in which Chip Select will be held after READ is deasserted. T4 prevents bus contention while Chip Select is asserted, in case the peripheral driving the bus continues to send data out after READ has been deasserted. During T4 no other transaction can start, since the current transaction will not finish until Chip Select is deasserted by the processor.

T5 is the recovery time required before the next transaction can start.

The EX_IOWAIT_N signal only affects the interface during T2 when it is asserted and during T3 when it is deasserted. If Chip Selects 4 through 7 are configured in HPI mode of operation, each chip select will have a corresponding HRDY signal called EX_RDY. The polarity of the ready signal is programmable. Chip Select 4 corresponds to EX_RDY signal 0 and Chip Select 7 corresponds to EX_RDY signal 3.

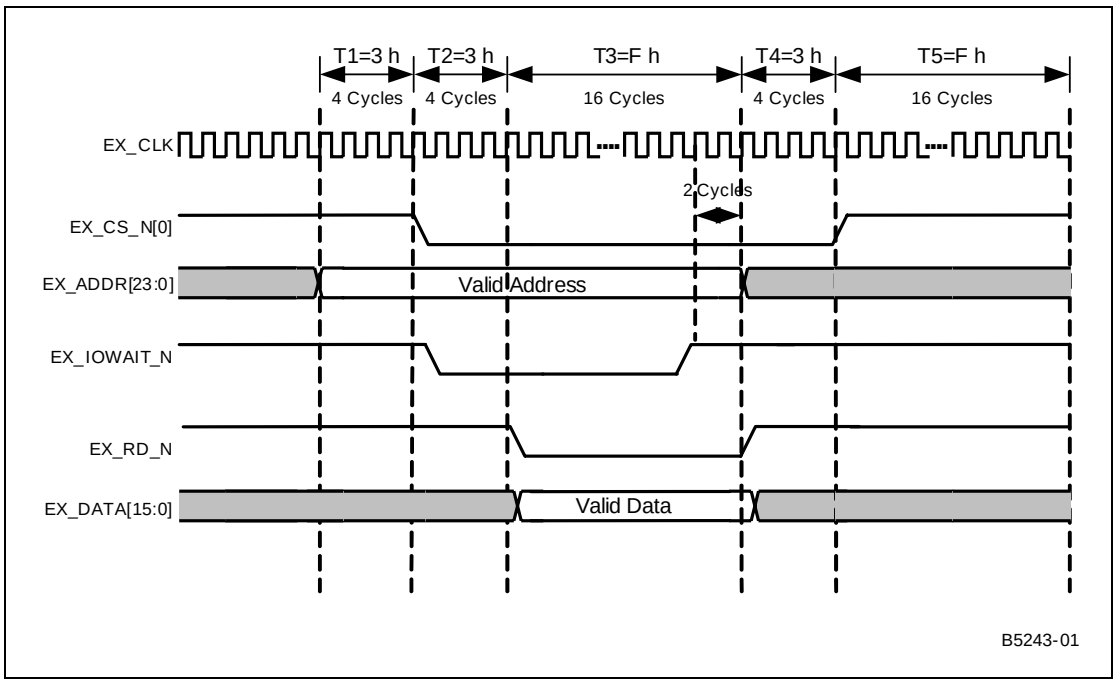
In the case of extended phase timing, EX_IOWAIT_N is used in the same way as the normal phase, however, the T1, T2, T4 and T5 periods take place over 4 cycles. T3 is still programmable but each value is a multiple of 4 cycles. See Figure 62 for details.

Figure 61. I/O Wait Normal Phase Timing



Note: Notice that the access is an Intel-style simplex read access. The data strobe phase is set to a value to last three clock cycles. The data is returned from the peripheral device prior to the three clocks and the peripheral device de-asserts EX_IOWAIT_N. The data strobe phase terminates after two clocks even though the strobe phase was configured to pulse for three clocks.

Figure 62. I/O Wait Extended Phase Timing



8.7 Special Design Knowledge for Using HPI mode

The Expansion Bus Controller supports a number of the 8-bit and 16-bit versions of the Texas Instruments Host Port Interface (HPI) standards. This flexibility allows the TMS320C54xx family of Digital Signals Processors (DSP) to seamlessly interface to the IXP42X product line and IXC1100 control plane processors Expansion Bus.

If the Expansion Bus CS (Chip-Select) is configured to operate to operate in HPI-8 Mode, then a STRH (16-bit write) Intel XScale processor instruction must be used for writing to the HPI-8 device, even though it is in an 8-bit device. If a STRB (8-bit write) instruction is used instead, then the Intel XScale processor's data abort handler will be initiated, causing the assigned HPI-8 CS signal to deassert.

However, there are some special things to note when using the Expansion Bus in HPI mode of operation. These features are shown in the following tables.

The expansion-bus address-pins bits 0, 1, 2, 22, and 23 are multiplexed with special function signal pins for HPI as shown in Table 120.

Table 120. Multiplexed Output Pins for HPI Operation

HPI Control Signal	Output Signal Pin
EX_HBIL	EX_ADDR [0]
EX_HCNTL [1:0]	EX_ADDR [2:1]
EX_HCSEL [1:0]	EX_ADDR [23:22]

The byte identification signal, EX_HBIL, is used to determine the byte transfer order. (EX_HBIL is driven low for the first byte of the transfer and driven high for the second byte.)

The byte order bit (BOB) in the HPIC register (contained in the DSP) — within the HPI device — is used to determine the placement for the two bytes of the transfer. Please consult the datasheet of the specific DSP being connected to determine the order of the transferred bytes.

When operating in HPI mode, bits 13:10 in the Timing and Control (EXP_TIMING_CS) Registers are ignored.

When operating in HPI-16, non-multiplexed mode, the expansion bus address bus provides direct accesses to the DSP memory space. The data associated with this address will be read or written from the location specified by the value contained on the Expansion Bus address bits.

The signals EX_HCNTL [1:0] are multiplexed onto the EX_ADDR [2:1] pins. When communicating to a multiplexed HPI interface, the EX_HCNTL [1:0] signals are used to select one of four internal registers used for interfacing to the DSP. The EX_HCNTL [1:0] mapping is described in the Table 121 below:

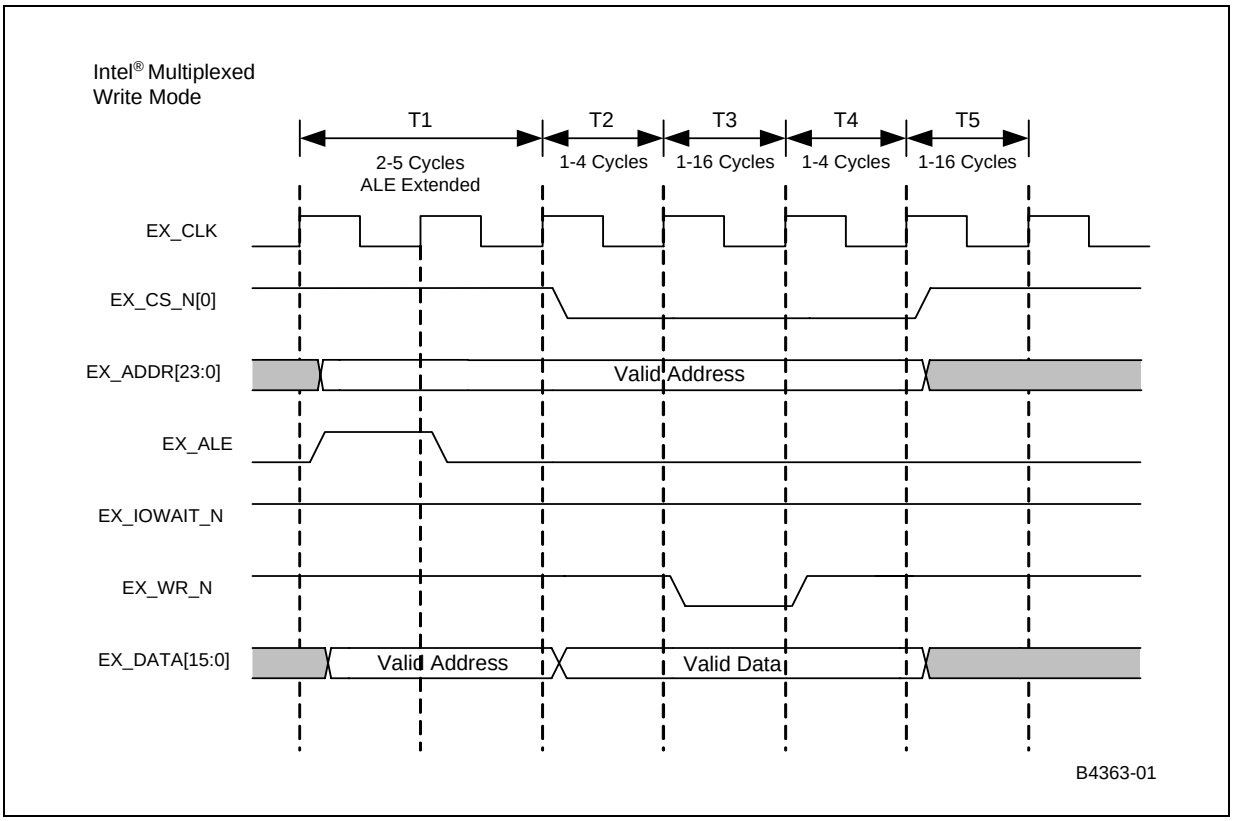
Table 121. HPI HCNTL Control Signal Decoding

hcntl[1:0]	Required Access
00	Read / write control register (HPIC)
01	Read / write data register (HPID) • HPI-8: — Post-increment HPIA on reads, pre-increment on writes. • HPI-16: — Post-increment HPIA on reads and writes
10	Read / write address register (HPIA)
11	Read / write data register (HPID)

8.8 Expansion Bus Interface Access Timing Diagrams

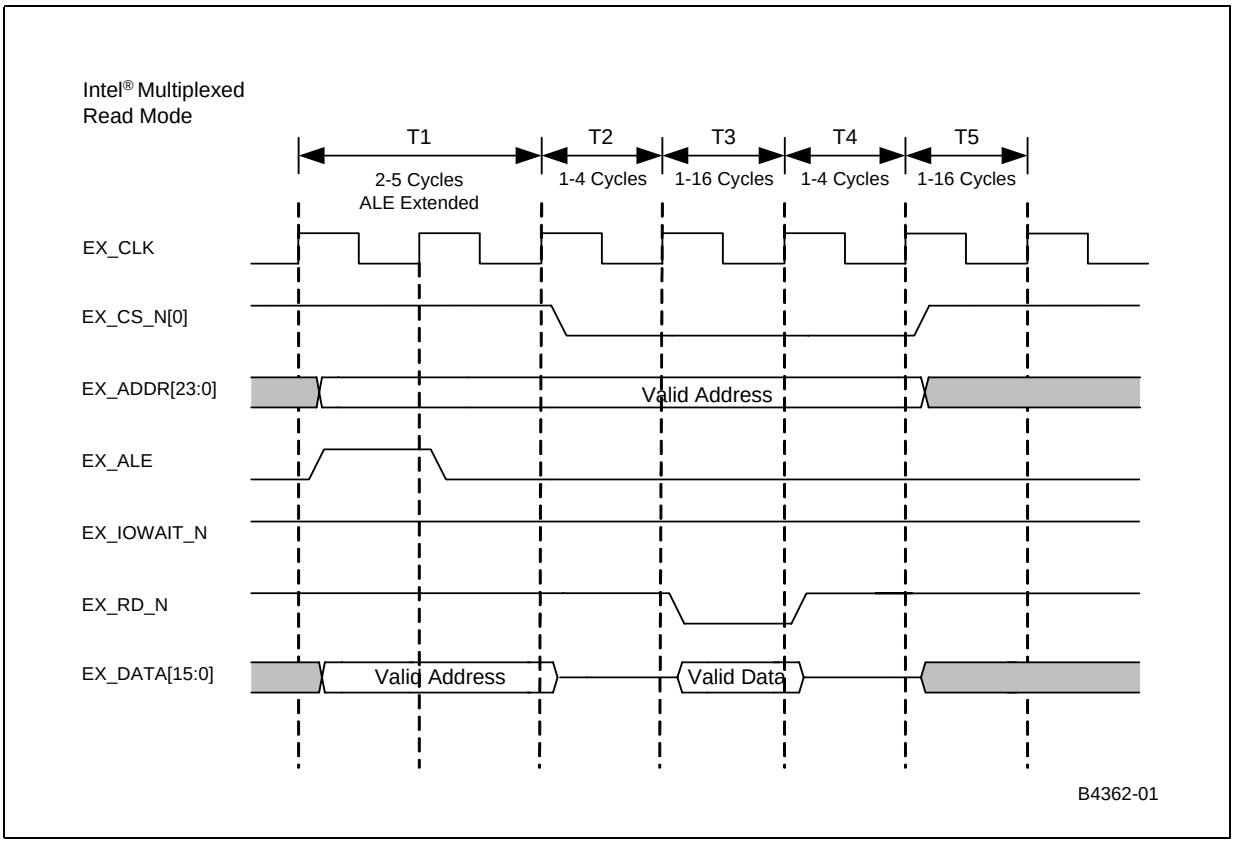
8.8.1 Intel® Multiplexed-Mode Write Access

Figure 63. Expansion-Bus Write (Intel® Multiplexed Mode)



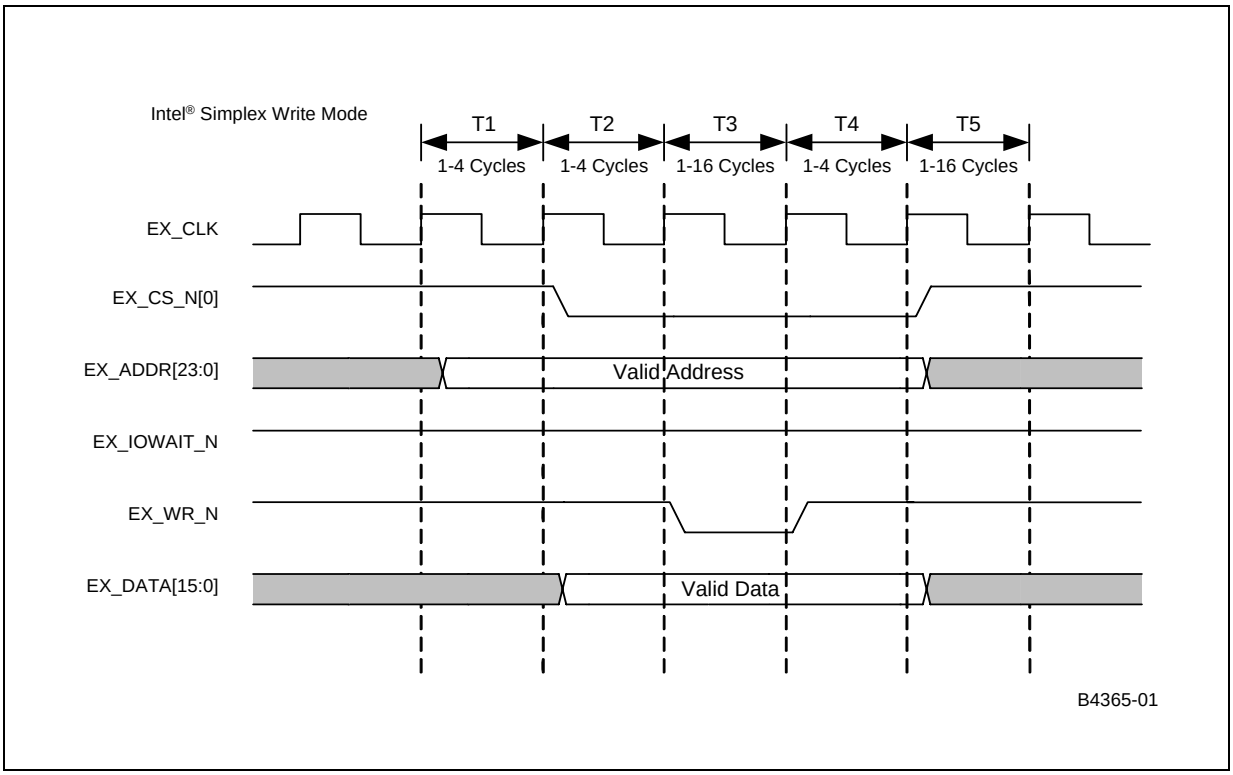
8.8.2 Intel® Multiplexed-Mode Read Access

Figure 64. Expansion-Bus Read (Intel® Multiplexed Mode)



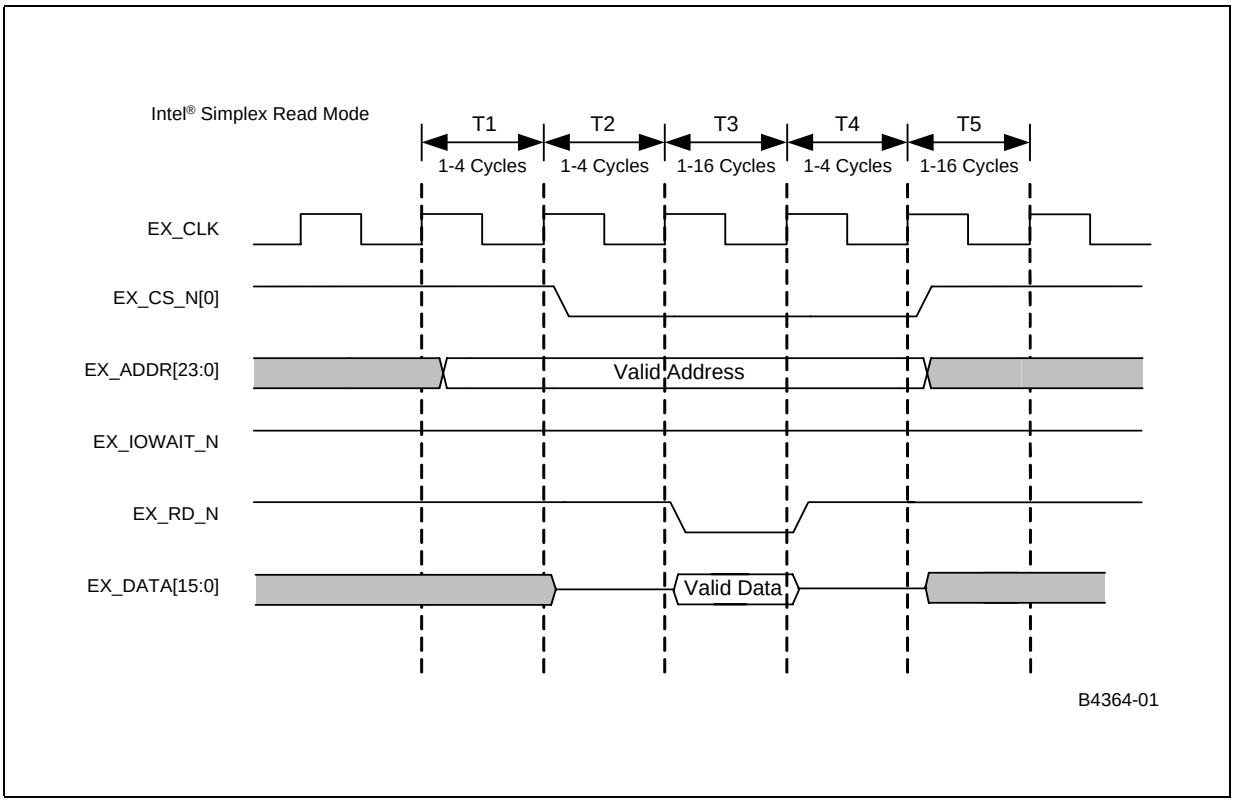
8.8.3 Intel® Simplex-Mode Write Access

Figure 65. Expansion-Bus Write (Intel® Simplex Write Mode)



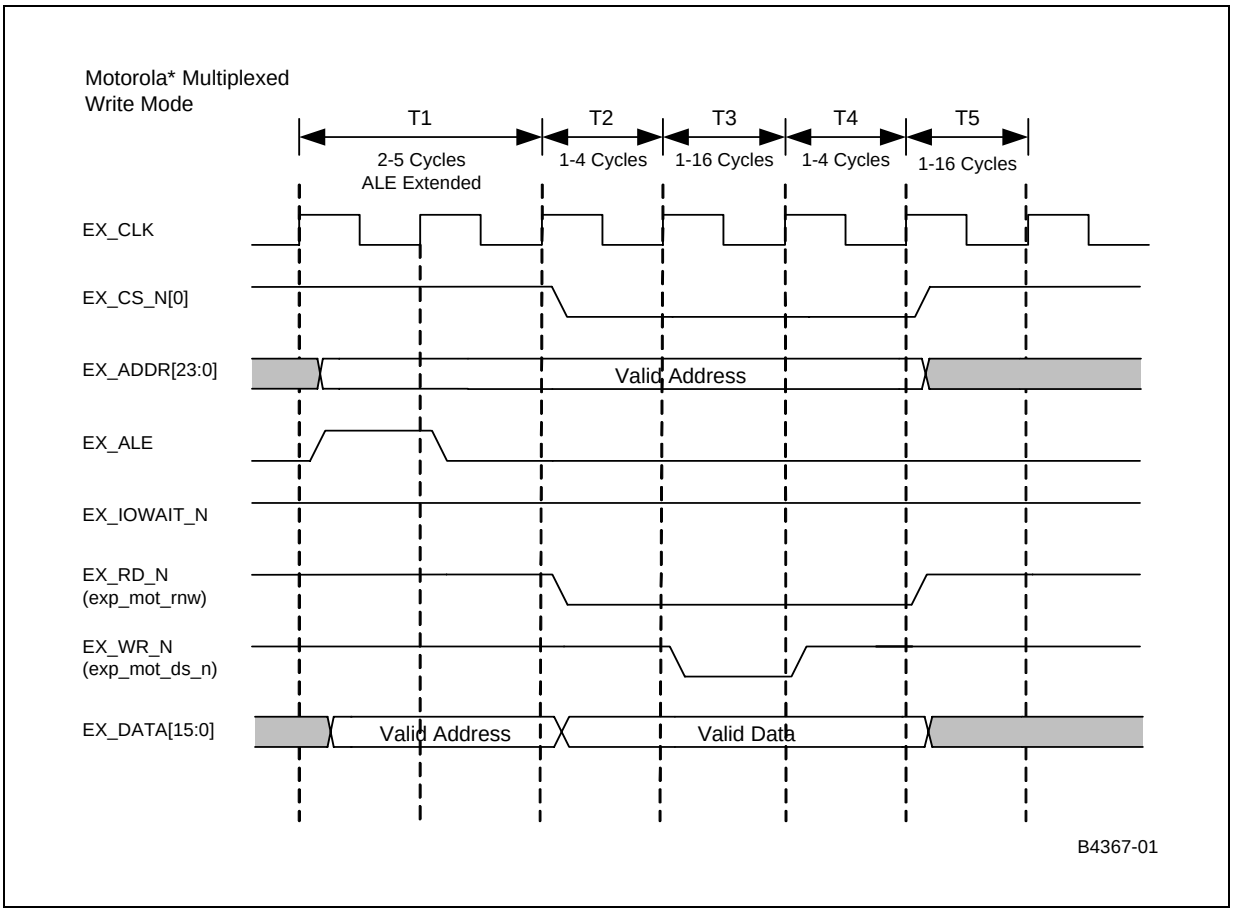
8.8.4 Intel® Simplex-Mode Read Access

Figure 66. Expansion-Bus Read (Intel® Simplex Mode)



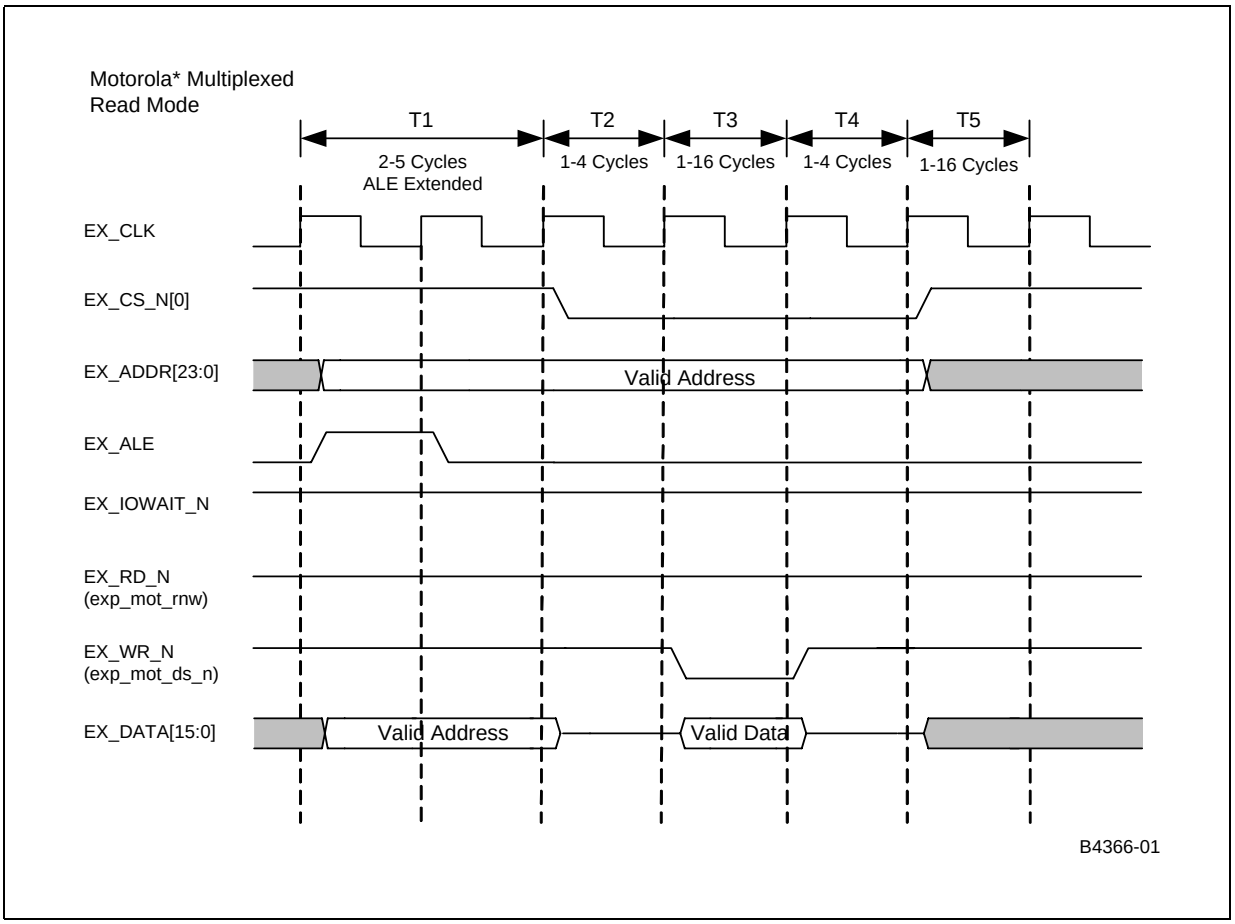
8.8.5 Motorola* Multiplexed-Mode Write Access

Figure 67. Expansion-Bus Write (Motorola* Multiplexed Mode)



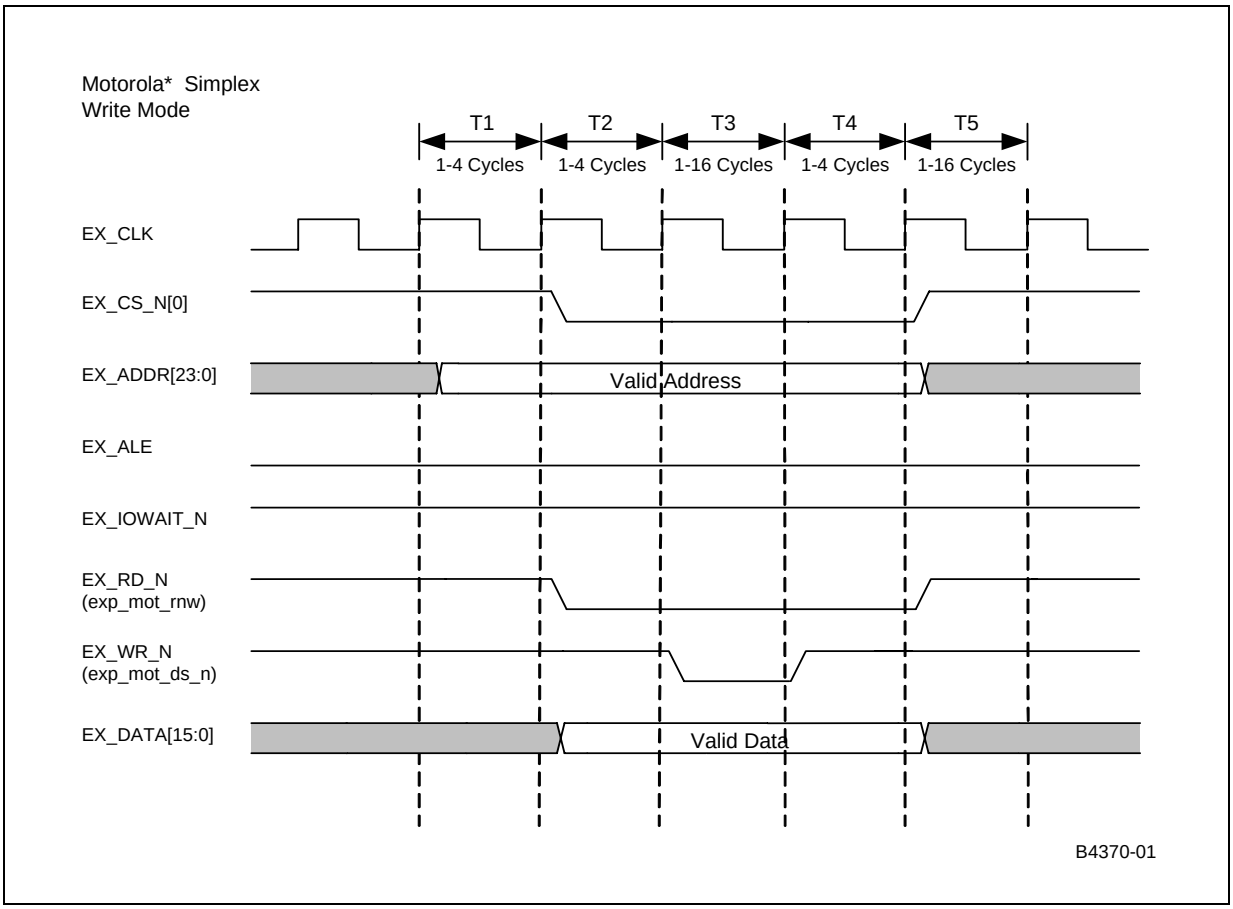
8.8.6 Motorola* Multiplexed-Mode Read Access

Figure 68. Expansion-Bus Read (Motorola* Multiplexed Mode)



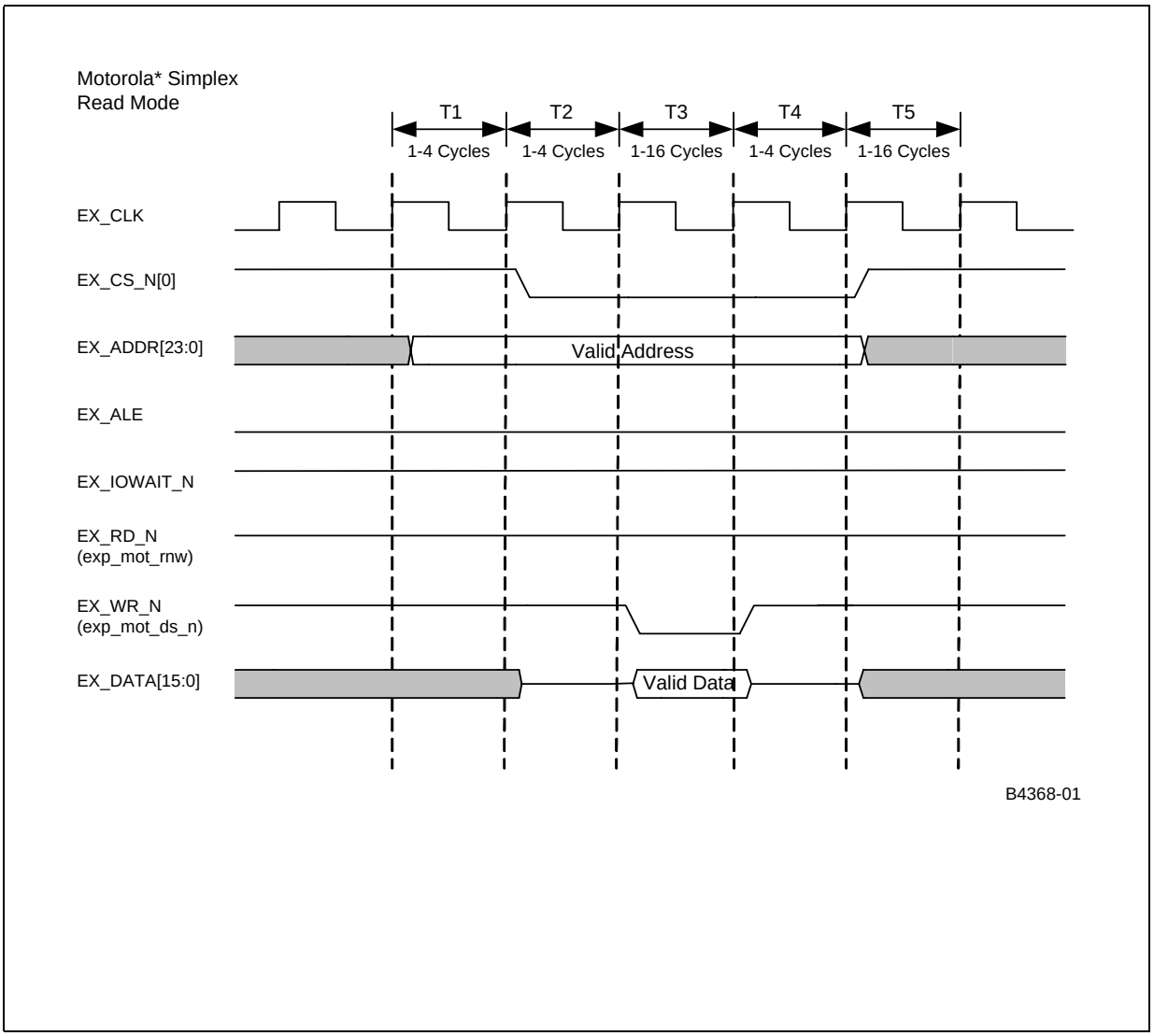
8.8.7 Motorola* Simplex-Mode Write Access

Figure 69. Expansion-Bus Write (Motorola* Simplex Mode)



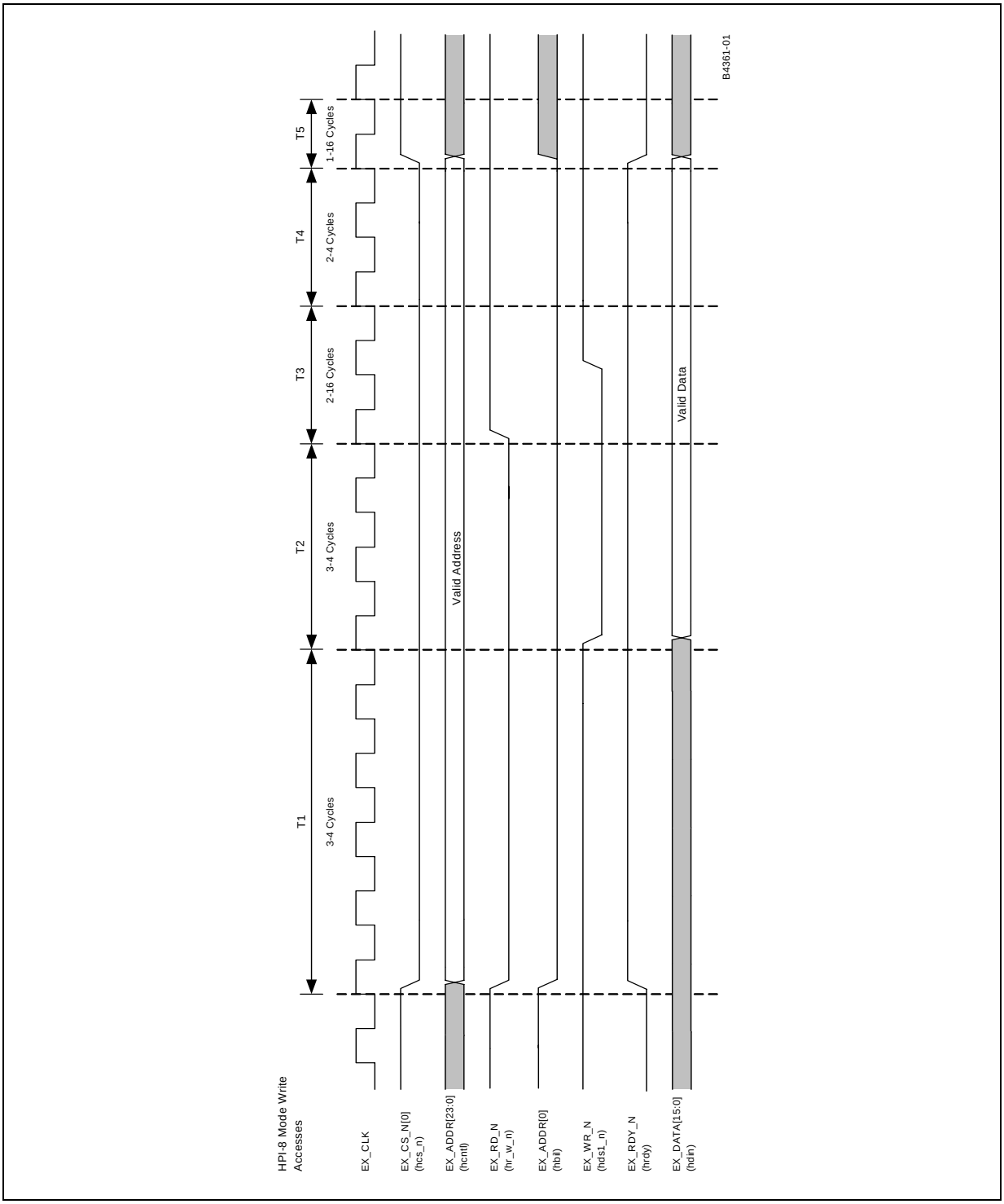
8.8.8 Motorola* Simplex-Mode Read Access

Figure 70. Expansion-Bus Read (Motorola* Simplex Mode)



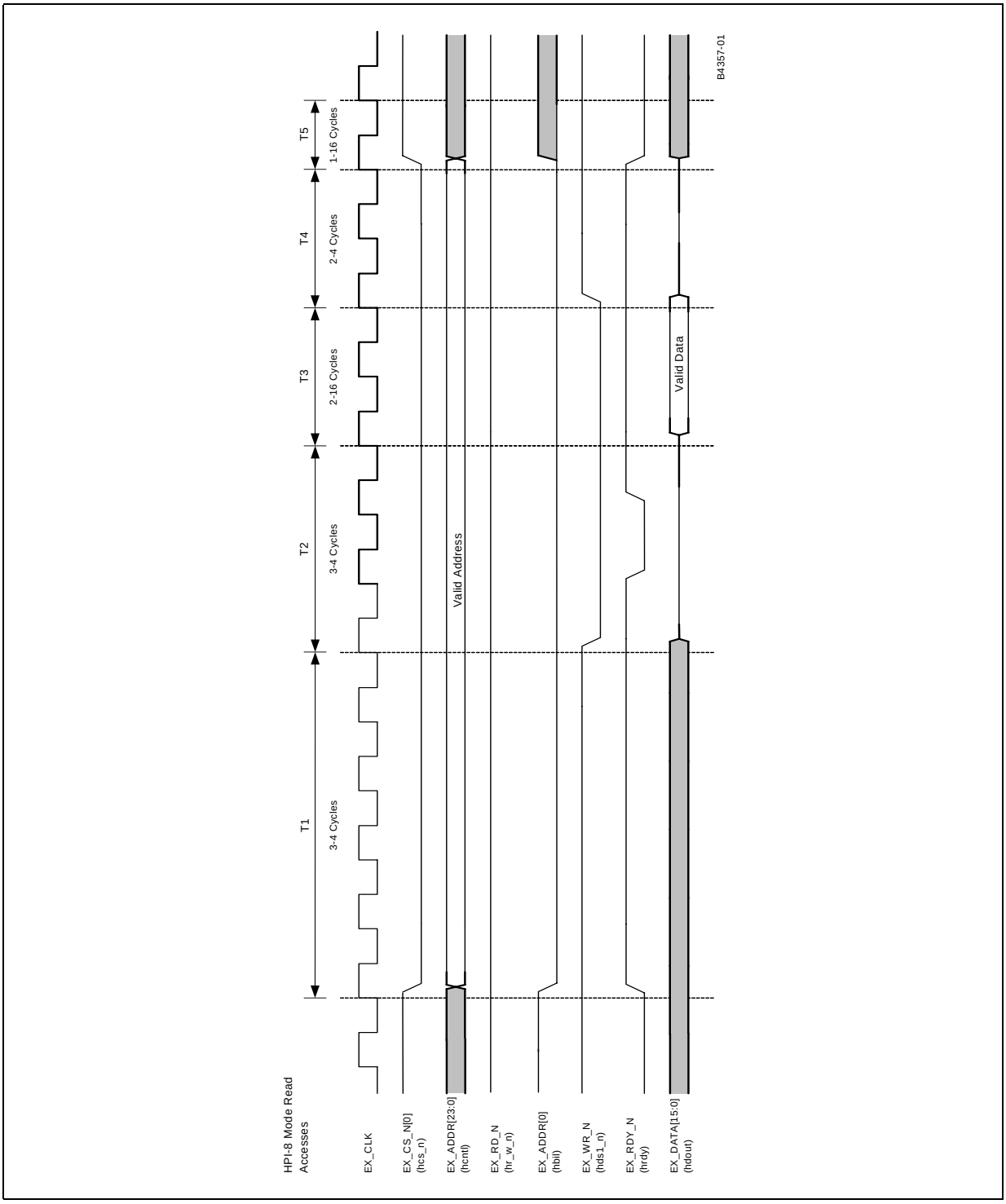
8.8.9 TI* HPI-8 Write Access

Figure 71. Expansion-Bus Write (TI* HPI-8 Mode)



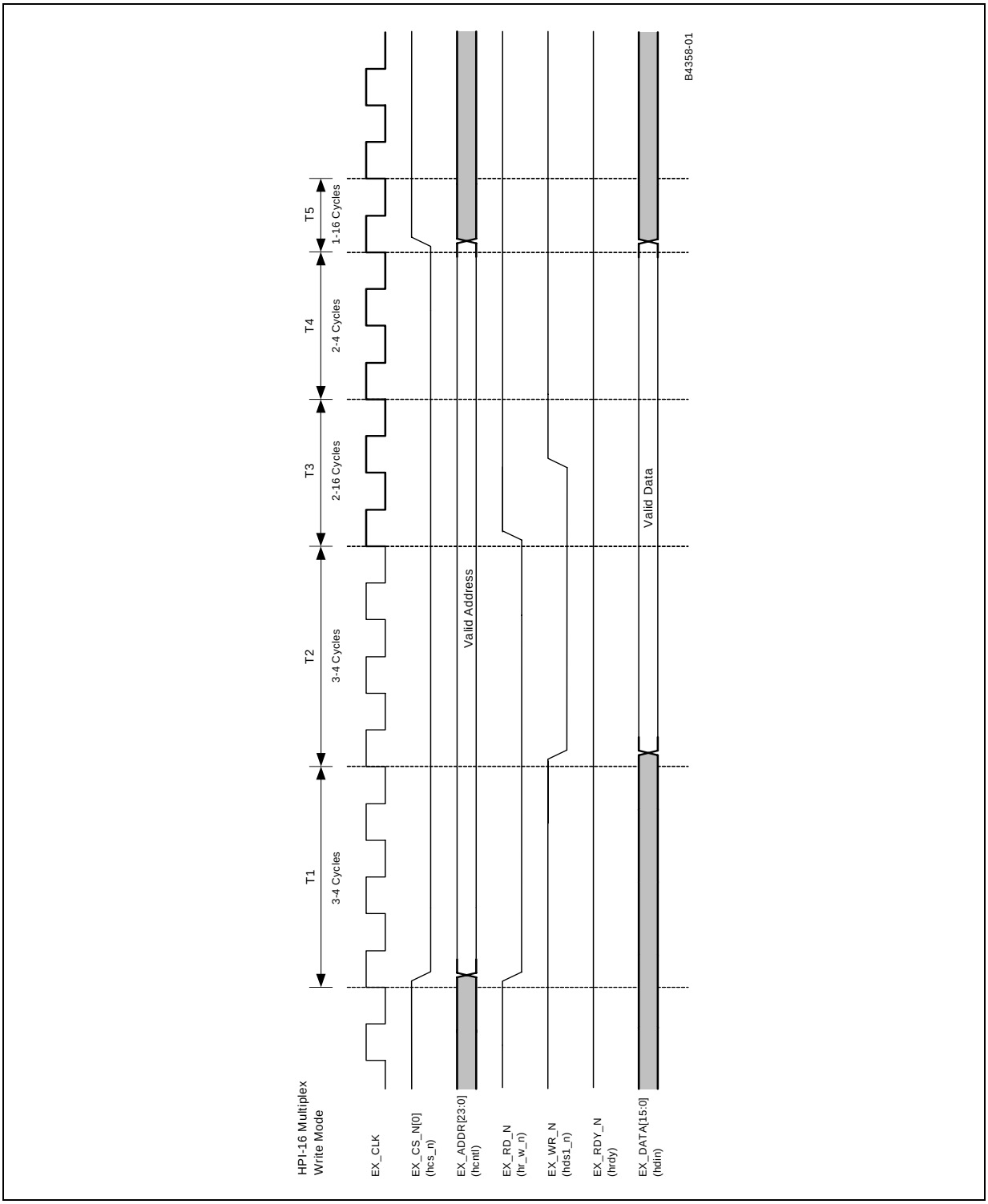
8.8.10 TI* HPI-8 Read Access

Figure 72. Expansion-Bus Read (TI* HPI-8 Mode)



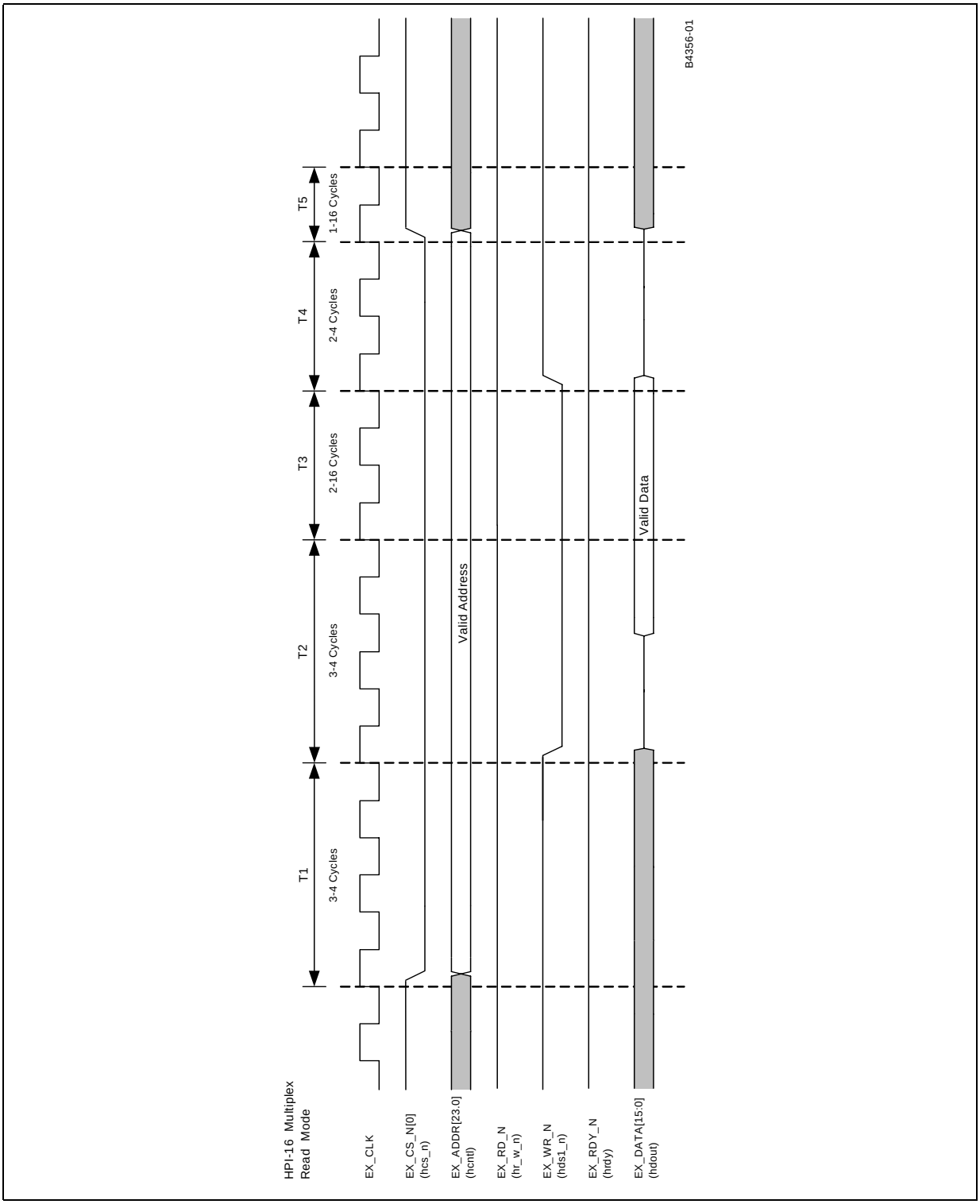
8.8.11 TI* HPI-16, Multiplexed-Mode Write Access

Figure 73. Expansion-Bus Write (TI* HPI-16 Multiplexed Mode)



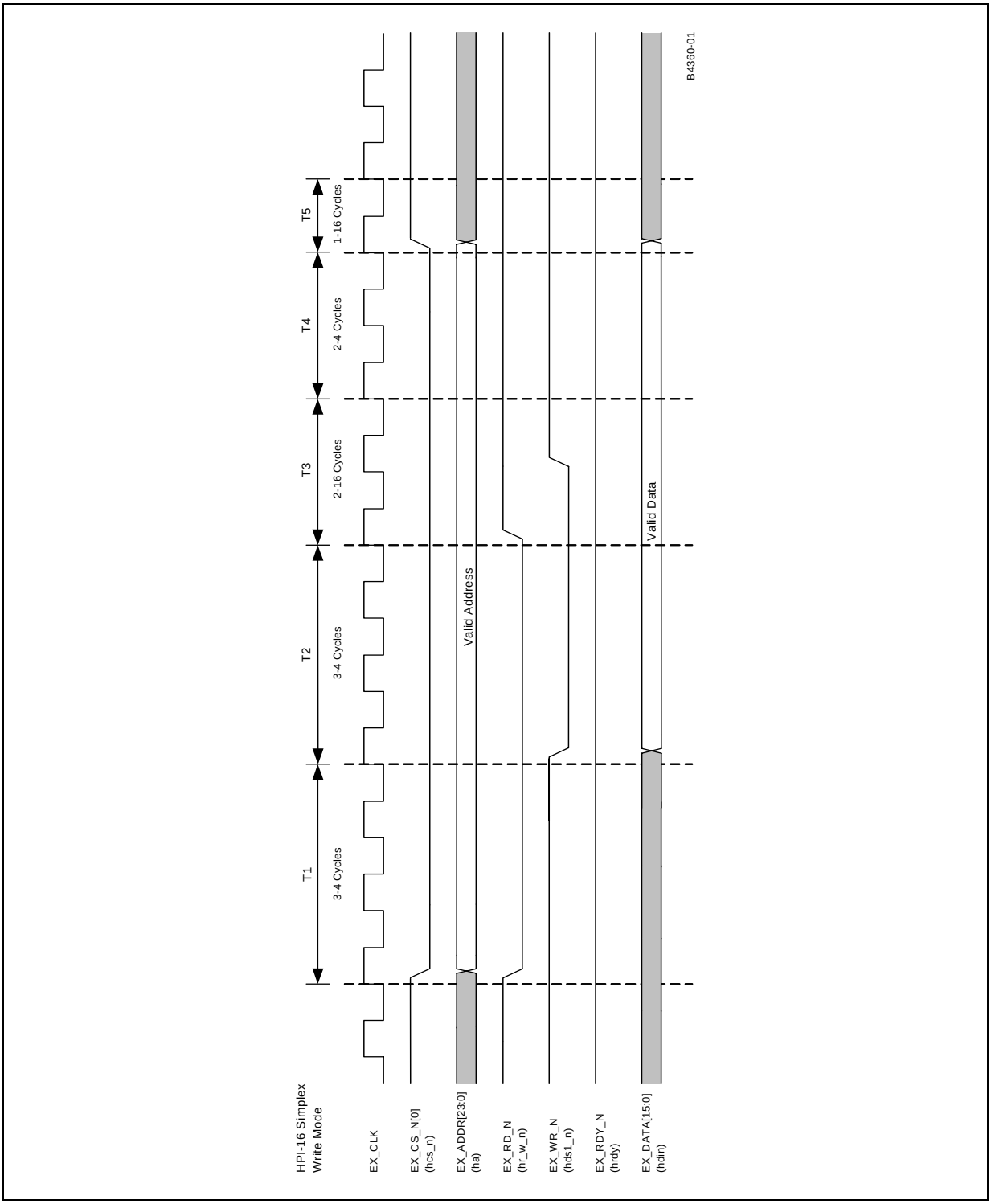
8.8.12 TI* HPI-16, Multiplexed-Mode Read Access

Figure 74. Expansion-Bus Read (TI* HPI-16 Multiplexed Mode)



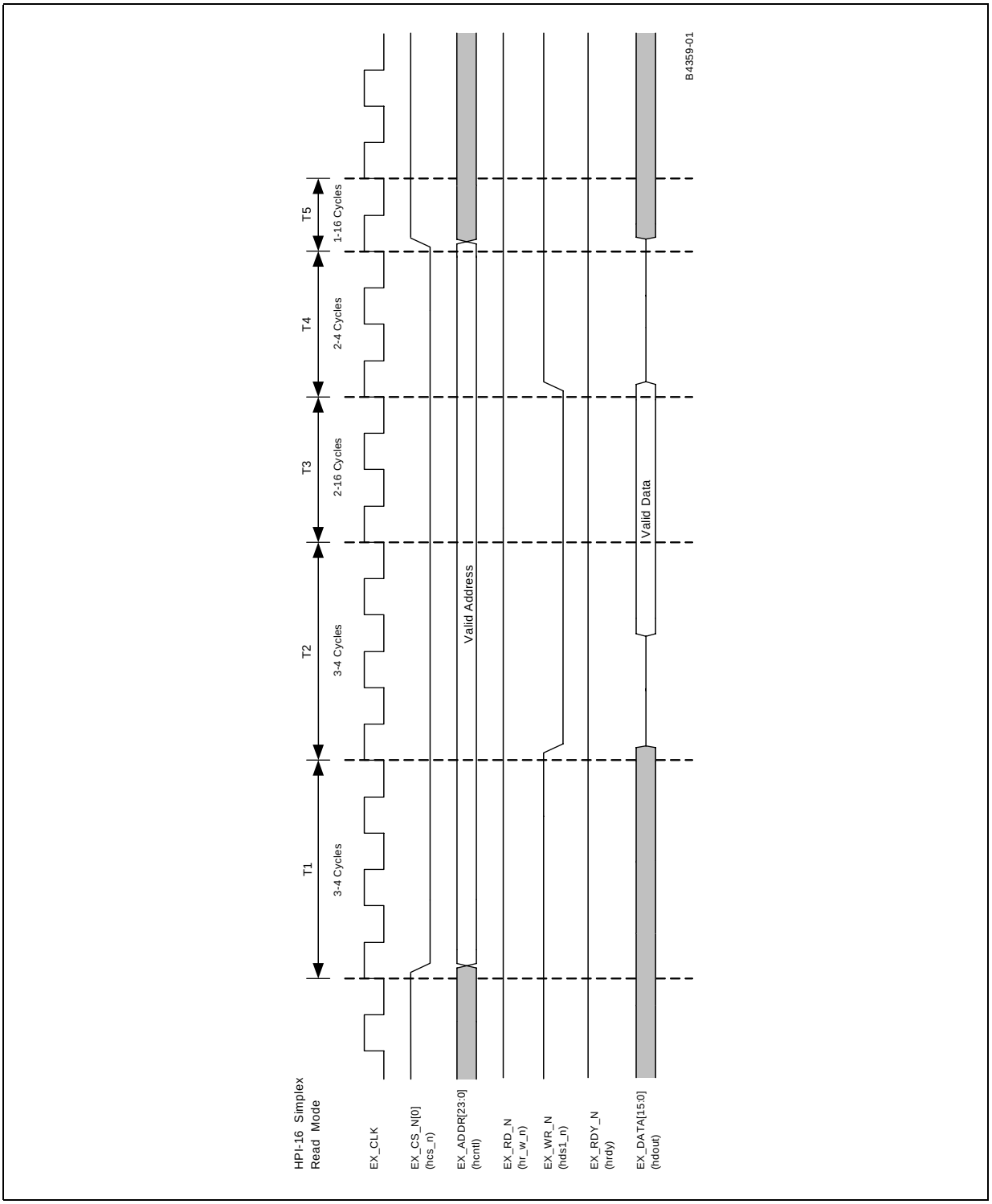
8.8.13 TI* HPI-16 Simplex-Mode Write Access

Figure 75. Expansion-Bus Write (TI* HPI-16 Simplex Mode)



8.8.14 TI* HPI-16 Simplex-Mode Read Access

Figure 76. Expansion-Bus Read (TI* HPI-16 Simplex Mode)



8.9 Register Descriptions

Table 122. Expansion Bus Register Overview

Address	R/W	Name	Description
0xC4000000	R/W	EXP_TIMING_CS0	Timing and Control Register for Chip Select 0
0xC4000004	R/W	EXP_TIMING_CS1	Timing and Control Register for Chip Select 1
0xC4000008	R/W	EXP_TIMING_CS2	Timing and Control Register for Chip Select 2
0xC400000C	R/W	EXP_TIMING_CS3	Timing and Control Register for Chip Select 3
0xC4000010	R/W	EXP_TIMING_CS4	Timing and Control Register for Chip Select 4
0xC4000014	R/W	EXP_TIMING_CS5	Timing and Control Register for Chip Select 5
0xC4000018	R/W	EXP_TIMING_CS6	Timing and Control Register for Chip Select 6
0xC400001C	R/W	EXP_TIMING_CS7	Timing and Control Register for Chip Select 7
0xC4000020	R/W	EXP_CNFG0	General Purpose Configuration Register 0
0xC4000024	R/W	EXP_CNFG1	General Purpose Configuration Register 1
0xC4000028	-	-	Reserved

8.9.1 Timing and Control Registers for Chip Select 0

Register Name:	EXP_TIMING_CS0			
Hex Offset Address:	0XC4000000		Reset Hex Value:	0xBFFF3C4x
Register Description:	Timing and Control Registers			
Access: Read/Write				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3				T4		T5				CYCLE_TYPE		CNFG[3:0]				(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

Note: The undefined (X) in the reset value is dependent upon values supplied to the chip on the Expansion Bus address at the de-assertion of RESET_IN_N. Please refer to Section 8.9.9, “Configuration Register 0” on page 322 for additional details.

8.9.2 Timing and Control Registers for Chip Select 1

Register Name:	EXP_TIMING_CS1			
Hex Offset Address:	0XC4000004		Reset Hex Value:	0x00000000
Register Description:	Timing and Control Registers			
Access: Read/Write				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3				T4		T5				CYCLE_TYPE		CNFG[3:0]				(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

8.9.3 Timing and Control Registers for Chip Select 2

Register Name:	EXP_TIMING_CS2			
Hex Offset Address:	0XC4000008		Reset Hex Value:	0x00000000
Register Description:	Timing and Control Registers			
Access: Read/Write.				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3			T4			T5			CYCLE_TYPE			CNFG[3:0]			(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

8.9.4 Timing and Control Registers for Chip Select 3

Register Name:	EXP_TIMING_CS3			
Hex Offset Address:	0XC400000C		Reset Hex Value:	0x00000000
Register Description:	Timing and Control Registers			
Access: Read/Write				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3			T4			T5			CYCLE_TYPE			CNFG[3:0]			(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

8.9.5 Timing and Control Registers for Chip Select 4

Register Name:	EXP_TIMING_CS4			
Hex Offset Address:	0XC4000010		Reset Hex Value:	0x00000000
Register Description:	Timing and Control Registers			
Access: Read/Write.				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3			T4			T5			CYCLE_TYPE			CNFG[3:0]			(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

8.9.6 Timing and Control Registers for Chip Select 5

Register Name:	EXP_TIMING_CS5			
Hex Offset Address:	0XC4000014		Reset Hex Value:	0x00000000
Register Description:	Timing and Control Registers			
Access: Read/Write.				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3				T4		T5				CYCLE_ TYPE		CNFG[3:0]				(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

8.9.7 Timing and Control Registers for Chip Select 6

Register Name:	EXP_TIMING_CS6			
Hex Offset Address:	0XC4000018		Reset Hex Value:	0x00000000
Register Description:	Timing and Control Registers			
Access: Read/Write.				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3				T4		T5				CYCLE_ TYPE		CNFG[3:0]				(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

8.9.8 Timing and Control Registers for Chip Select 7

Register Name:	EXP_TIMING_CS7			
Hex Offset Address:	0XC400001C		Reset Hex Value:	0x00000000
Register Description:	Timing and Control Registers			
Access: Read/Write.				

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13			10	9			6	5	4	3	2	1	0
CSx_EN	(Rsvd)	T1		T2		T3				T4		T5				CYCLE_ TYPE		CNFG[3:0]				(Rsvd)			BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	(Rsvd)	WR_EN	BYTE_EN

Table 123. Bit Level Definition for each of the Timing and Control Registers

Bits	Name	Description
31	CSx_EN	0 = Chip Select x disabled 1 = Chip Select x enabled
30		(Reserved)
29:28	T1 – Address timing	00 = Generate normal address phase timing 01 - 11 = Extend address phase by 1 - 3 clocks
27:26	T2 – Setup / Chip Select Timing	00 = Generate normal setup phase timing 01 - 11 = Extend setup phase by 1 - 3 clocks
25:22	T3 – Strobe Timing	0000 = Generate normal strobe phase timing 0001-1111 = Extend strobe phase by 1 - 15 clocks
21:20	T4 – Hold Timing	00 = Generate normal hold phase timing 01 - 11 = Extend hold phase by 1 - 3 clocks
19:16	T5 – Recovery Timing	0000 = Generate normal recovery phase timing 0001-1111 = Extend recovery phase by 1 - 15 clocks
15:14	CYC_TYPE	00 = Configures the expansion bus for Intel cycles. 01 = Configures the expansion bus for Motorola* cycles. 10 = Configures the expansion bus for HPI cycles. (HPI reserved for chip selects [7:4] only) 11 = Reserved
13:10	CNFG[3:0]	Device Configuration Size. Calculated using the formula: SIZE OF ADDR SPACE = $2^{(9+CNFG[3:0])}$ For Example: 0000 = Address space of 2^9 = 512 Bytes ... 1000 = Address space of 2^{17} = 128 Kbytes ... 1111 = Address space of 2^{24} = 16 Mbytes
9:7		(Reserved)
6	BYTE_RD16	Byte read access to Half Word device 0 = Byte access disabled. 1 = Byte access enabled.
5	HRDY_POL	HPI HRDY polarity (reserved for exp_cs_n[7:4] only) 0 = Polarity low true. 1 = Polarity high true.
4	MUX_EN	0 = Separate address and data buses. 1 = Multiplexed address / data on data bus.
3	SPLT_EN	0 = AHB split transfers disabled. 1 = AHB split transfers enabled.
2		(Reserved)
1	WR_EN	0 = Writes to CS region are disabled. 1 = Writes to CS region are enabled.
0	BYTE_EN	0 = Expansion bus uses 16-bit-wide data bus. 1 = Expansion bus uses only 8-bit data bus.

8.9.9 Configuration Register 0

At power up or whenever a reset is asserted, the expansion-bus address outputs are switched to inputs and the states of the bits are captured and stored in Configuration Register 0, bits 23 through 0. This occurs on the first cycle after the synchronous de-assertion of the reset signal.

These configuration bits are made available to the system as outputs from the Expansion Bus Controller block. With the exception of bits 23, 22 and 21, which are read only, all other bits may be written and read from the South AHB.

Register Name:	**EXP_CNFG0**				
Hex Offset Address:	0XC4000020		**Reset Hex Value:**	0x8XXXXXXX	
Register Description:	Configuration Register #0				
Access: Read/Write.					

31	**30**						**24**	**23**	**22**	**21**	**20**			**17**	**16**									**5**	**4**	**3**	**2**	**1**	**0**
MEM_MAP	(Reserved)							CLK bit 2	CLK Bit 1	CLK Bit 0	User-configurable				(Reserved)										PCI_ CLK	RES	PCI_ARB	PCI_ HOST	8/16

Table 124. Configuration Register 0 Description

Bit	Name	Description
31	MEM_MAP	Location of EXPBus in memory map space: 0 = Located at “50000000” (normal mode) 1 = Located at “00000000” (boot mode)
30:24		(Reserved)
23:21	Intel XScale processor Clock Set[2:0]	Allow a slower Intel XScale processor clock speed to override device fuse settings. However cannot be used to over clock core speed. Refer to Section 125, “Intel XScale® Processor Speed Expansion Bus Configuration Strappings” on page 324 for additional details.
20:17	User-configurable	User-configurable. See Section 8.9.9.1 for additional comments.
16:5		(Reserved)
4	PCI_CLK	Sets the clock speed of the PCI Interface 0 = 33 MHz 1 = 66 MHz
3		(Reserved). EX_ADDR[3] must not be pulled down during address strapping. This bit must be written to ‘1’ if performing a write to this register.
2	PCI_ARB	Enables the PCI Controller Arbiter 0 = PCI arbiter disabled 1 = PCI arbiter enabled
1	PCI_HOST	Configures the PCI Controller as PCI Bus Host 0 = PCI as non-host 1 = PCI as host
0	8/16 FLASH	Specifies the data bus width of the FLASH memory device 0 = 16-bit data bus 1 = 8-bit data bus
Note: The bits that are (Reserved) should remain reserved because they may be assigned to future steppings of the IXP42X product line and IXC1100 control plane processors**.**		

The chip-level memory map used is determined by the state of bit 31. At system reset this bit is a ‘1’ and the memory map places the Expansion Bus at address 0x00000000 through 0x0FFFFFFF. This allows boot code stored in flash to be retrieved and executed as required.

Once the boot sequence completes this bit is written to a '0,' switching the default system memory map to place the SDRAM controller at address 0x00000000 to 0x0FFFFFFF. The Expansion Bus Controller now resides at address 0x50000000 to 0x5FFFFFFF. Weak pull-up resistors are placed on each expansion-bus address pin.

Table 125. Intel XScale® Processor Speed Expansion Bus Configuration Strappings

Intel XScale® Processor Speed (Factory Part Speed)	CFG_EN_N EX_ADDR(23)	CFG1 EX_ADDR(22)	CFGO EX_ADDR(21)	Actual Core Speed (MHz)
533 MHz	1	0	0	533 MHz
533 MHz	0	0	0	533 MHz
533 MHz	0	0	1	400 MHz
533 MHz	0	1	1	266 MHz
400 MHz	1	0	0	400 MHz
400 MHz	0	0	0	400 MHz
400 MHz	0	0	1	400 MHz
400 MHz	0	1	1	266 MHz
266 MHz	1	0	0	266 MHz
266 MHz	0	0	0	266 MHz
266 MHz	0	0	1	266 MHz
266 MHz	0	1	1	266 MHz

Note that the Intel XScale processor can operate at slower speeds than the factory programmed speed setting. This is done by placing a value on Expansion bus address bits 23,22,21 at the de-assertion of RESET_IN_N and knowing the speed grade of the part from the factory. Column 1 above denotes the speed grade of the part from the factory. Column 2, 3, and 4 denotes the values captured on the Expansion Bus address bits at the de-assertion of reset. Column 5 represents the speed at which the Intel XScale processor speed will now be operating at.

8.9.9.1 User-Configurable Field

On the IXP42X product line and IXC1100 control plane processors, the expansion bus address lines for the user-configurable bit-field are internally pulled up. Users may then change the values by adding weak pull-down resistors (~10KΩ). Switches can also be used so that changeable values are available for the user configuration bits.

The user-defined bit-field can be used in many ways. For example, this field could be used for board-revision identification; a series of board revisions may be made over the course of development. To indicate a particular board revision, one of the 16 possible values can be encoded using hardware configuration as stated above. Another potential use for this field would be to predefine a set of values to indicate a particular board configuration — for example, one with a different set of devices and memory map. Many other creative options, not identified in this document, are possible.

8.9.10 Configuration Register 1

One additional configuration register is defined within the Expansion Bus Controller for use by the IXP42X product line and IXC1100 control plane processors.

Register Name:	EXP_CNFG1				
Hex Offset Address:	0XC4000024		Reset Hex Value:	0x00000000	
Register Description:	Configuration Register #1				
Access: Read/Write.					

31	9	8	7	2	1	0
(Reserved)		BYTE_SWAP_EN	(Reserved)		SW_ INT1	SW_ INT0

Table 126. Expansion Bus Configuration Register 1-Bit Definition

Bit	Name	Description
31:16		(Reserved)
8	BYTE_SWAP_EN	Sets byte swapping at the Intel XScale processor 1 = byte swapping enabled 0 = byte swapping disabled ***Note:*** See note, below.
7:2		(Reserved)
1	SW_INT1	1 = Generate interrupt 0 = Disable interrupt
0	SW_INT0	1 = Generate interrupt 0 = Disable interrupt

Note: The selection between address or data coherency is controlled by a software-programmable, P-attribute bit in the Memory Management Unit (MMU) and the BYTE_SWAP_EN bit of the IXP42X product line and IXC1100 control plane processors. The BYTE_SWAP_EN bit will be from Expansion Bus Controller Configuration Register 1, Bit 8. When the IXP42X product line and IXC1100 control plane processors is reset, this bit will reset to 0.
The default endian conversion method for IXP42X product line and IXC1100 control plane processors is address coherency. This was selected to enable backward compatible with the Intel® IXP425 Network Processor A-Step processor.
The BYTE_SWAP_EN bit is an enable bit that enables data coherency to be performed, based on the P-attribute bit.
When the bit is 0, address coherency is always performed.
When the bit is 1, the type of coherency depends on the P-attribute bit.
The P-attribute bit is associated with each 1-Mbyte page. The P-attribute bit is output, from the Intel XScale processor, with any store or load access associated with that page.

Only two bits in Configuration Register 1 are currently defined. Under software control, they allow interrupts to be generated to the Interrupt Controller.

8.10 Expansion Bus Controller Performance

Table 127 shows simulated expansion bus throughput.

Table 127. Simulated Expansion Bus Performance

Intel XScale® Processor Command	Expansion Bus Width (bits)	Expansion Bus Frequency (MHz)	Maximum Sustainable Throughput (MBytes/s)
ldrb	8	33	2.01
ldrb	8	66	3.21
ldrb	16	33	2.01
ldrb	16	66	3.21
strb	8	33	1.98
strb	8	66	3.29
ldrh	16	33	4.01
ldrh	16	66	6.43
strh	16	33	3.95
strh	16	66	6.59
ldr	8	33	3.81
ldr	8	66	6.85
ldr	16	33	5.96
ldr	16	66	10.11
ldr (cacheable region)	8	33	5.18
ldr (cacheable region)	8	66	10.20
ldr (cacheable region)	16	33	10.12
ldr (cacheable region)	16	66	19.66

Note: Maximum sustainable throughput numbers were obtained by simulations. Throughput may vary depending on what is attached to the expansion bus.

§ §

9.0 AHB/APB Bridge

The APB Bridge is used to connect the high-speed AHB to the lower-speed peripherals connected to the APB. The AHB-APB Bridge provides a 32-bit wide data path between the South AHB and the APB.

Address bits 31:16 of the AHB bus are not sent to the bridge, but are instead decoded by the AHB arbiter and a select signal is used to indicate that the address references a location downstream of the bridge. Address bits 15:12 are decoded by the bridge and used to determine which of the APB peripherals is selected. Address bits 1:0 are not sent to the APB peripheral. All access must be word-aligned irrelevant of the bus size of the peripheral. There are 1,024 32-bit addresses supplied for each peripheral. Figure 77 shows the peripherals attached to the APB.

Figure 77. APB Interface

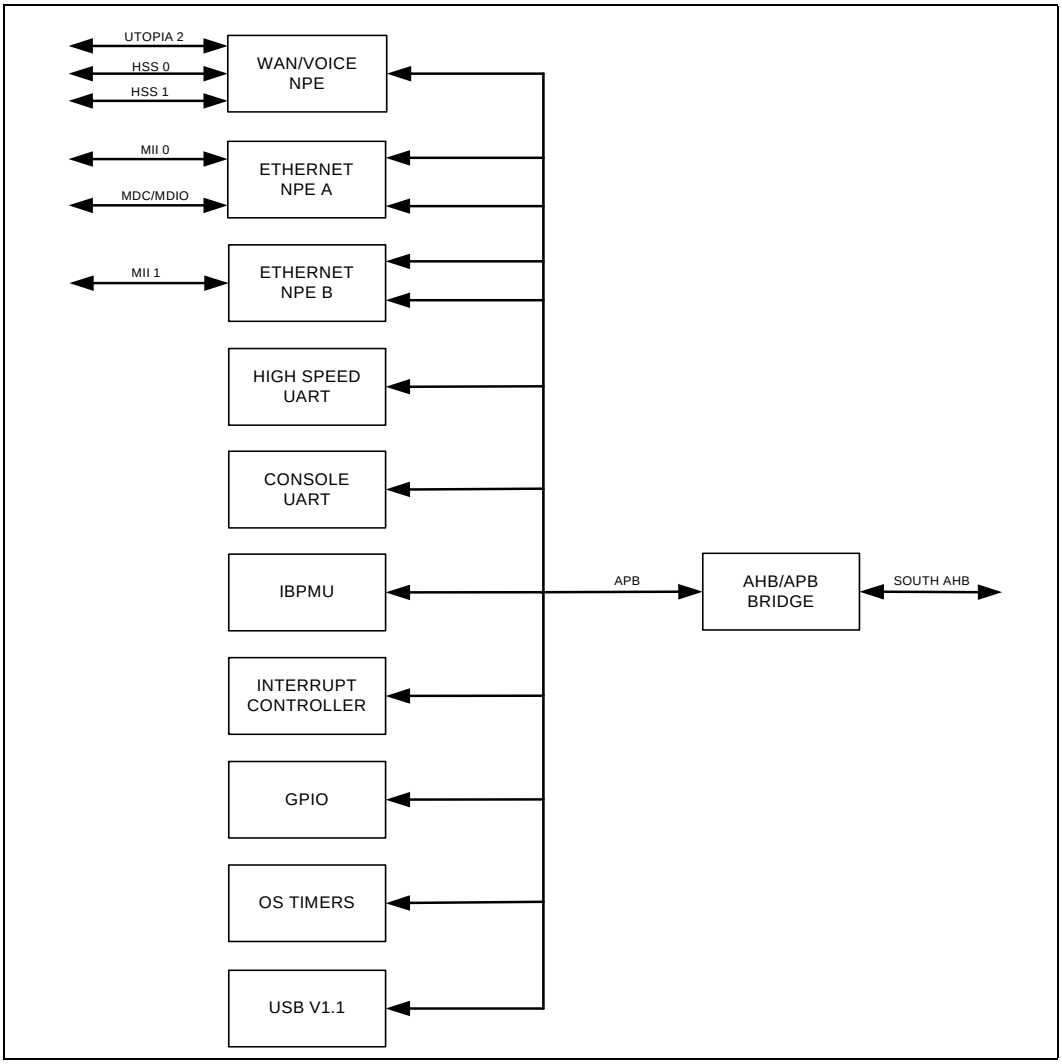


Table 128. Address Map for the APB

Offset Address	Peripheral
C8000000 – C8000FFF	UART 1
C8001000 – C8001FFF	UART 2
C8002000 – C8002FFF	IBPMU
C8003000 – C8003FFF	Interrupt Controller
C8004000 – C8004FFF	GPIO Controller
C8005000 – C8005FFF	OS Timer
C8006000 – C8006FFF	WAN/Voice NPE = NPE-A (IXP400 software Definition)– Not User Programmable
C8007000 – C8007FFF	Ethernet NPE A = NPE-B (IXP400 software Definition) – Not User Programmable
C8008000 – C8008FFF	Ethernet NPE B = NPE-C (IXP400 software Definition) – Not User Programmable
C8009000 – C8009FFF	Ethernet NPE A
C800A000 – C800AFFF	Ethernet NPE B
C800B000 – C800BFFF	USB Device Controller
C800C000 – C800EFFF	(Reserved)

§ §

10.0 Universal Asynchronous Receiver Transceiver (UART)

The Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor supports two Universal Asynchronous Receiver-Transmitter (UART) interfaces. One UART is intended primarily to support debug and control functions. The other UART is primarily intended to support connection to high-speed serial devices, such as Bluetooth*.

Each UART supports four pins:

- Transmit Data
- Receive Data
- Request to Send
- Clear to Send

10.1 High Speed UART

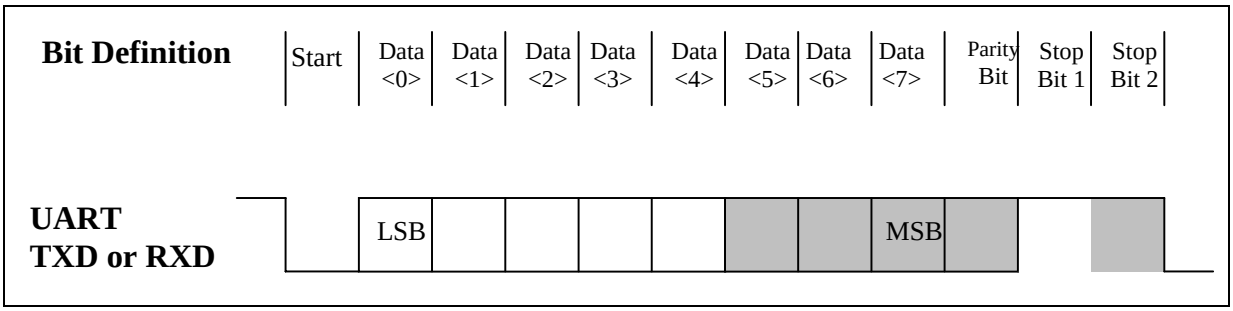
The UARTs performs serial-to-parallel conversion — on data characters received from a peripheral device or a modem — and parallel-to-serial conversion — on data characters received from the Intel XScale® Processor.

The Intel XScale processor, within the Intel® IXP42X product line and IXC1100 control plane processors, can read the complete status of the UART at any time during functional operation. Available status information includes the type and condition of the transfer operations being performed by the UART, as well as any detected error conditions (such as parity, overrun, framing, or break interrupt).

The UART is compatible with the 16550 UART specification, with enhancements in place to support higher speeds than defined by the 16550 UART specification. The UARTs are capable of supporting data transfers containing five, six, seven, or eight data bits. The data transfers may be configured to have one or two stop bits and supports even, odd, or no parity.

Figure 78 shows a functional waveform of the data that could be contained on the UART transmit and receive lines. Notice that Data bits 5 through 7, the Parity Bit, and Stop Bit 2 are shaded. The Data bits 5 through 7, Parity Bit, and Stop Bit 2 are all programmable and optional as previously described.

Figure 78. UART Timing Diagram



The serial port can operate in either FIFO or non-FIFO mode. In FIFO mode, a 64-byte transmit FIFO holds data coming from the Intel XScale processor to be transmitted on the serial link, and the 64-byte Receive FIFO, buffers data received from the serial link until the data is read by the Intel XScale processor.

The UARTs include a programmable baud rate generator capable of dividing the 14.7456-MHz, UART input clock by divisors of 1 to (2^{16} –1) and produces a 16X clock to drive the internal transmitter and receiver logic. The 14.7456-MHz, UART input clock is generated internally to the IXP42X product line and IXC1100 control plane processors.

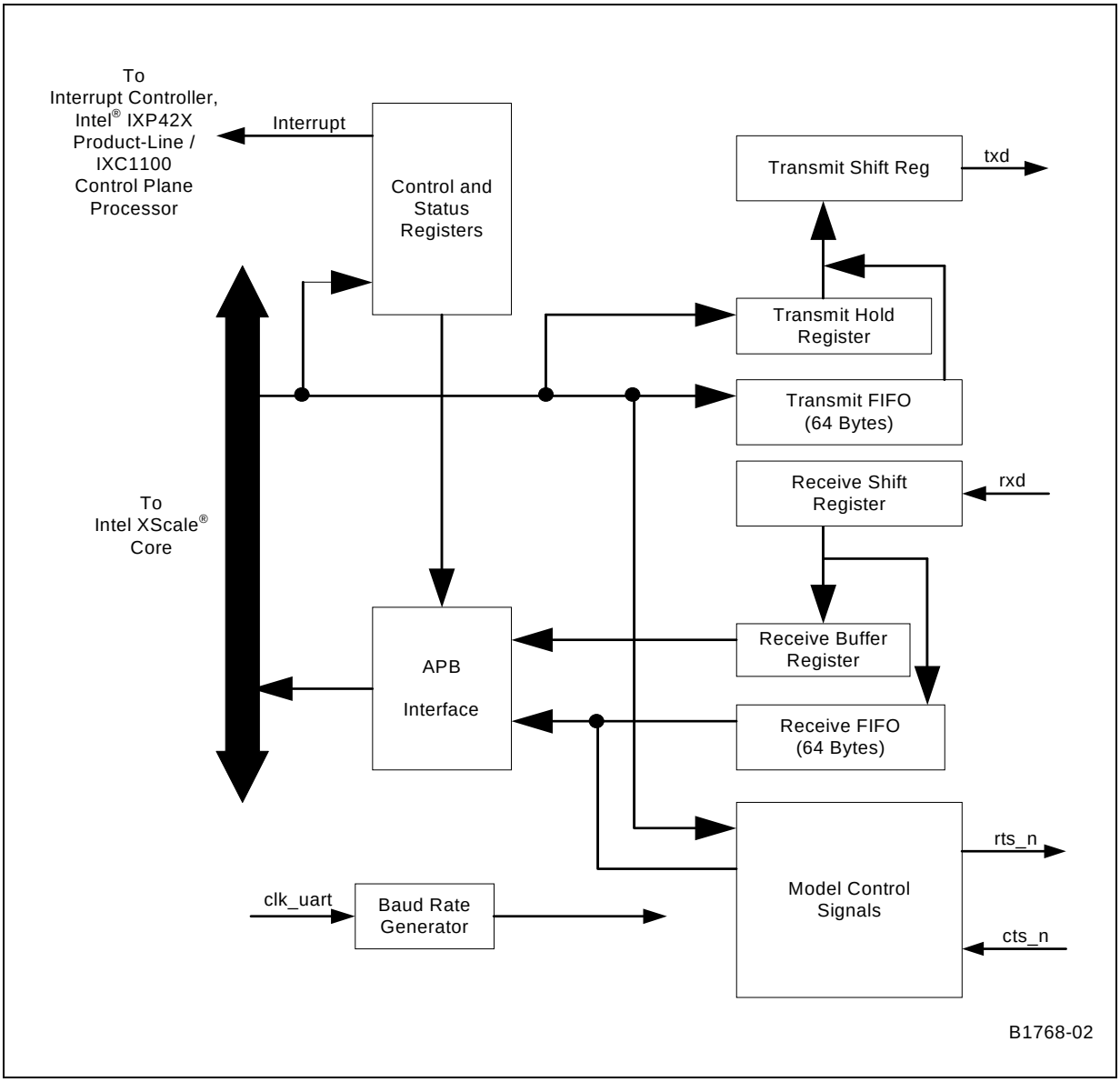
Interrupts can be programmed to the user's requirements, minimizing the computing required to handle the communications link. Each UART can be operated in a polled or an interrupt driven environment as selected by software.

The maximum baud rate supported by the High-Speed UART and Console UART is 921.6 Kbps. The divisors programmed in divisor latch registers should be equal to or greater than 1 for proper operation.

The device UARTs may be initialized by setting 13 configuration registers.

Figure 79 shows a functional block diagram of the UART interface.

Figure 79. UART Block Diagram



10.2 Configuring the UART

The following sections provide a detailed description of configuring the UART interfaces for operation.

10.2.1 Setting the Baud Rate

Each UART contains a programmable baud-rate generator that is capable of taking the 14.7456 MHz, input clock and dividing it by any divisor ranging from 1 to (2^{16} –1). The output frequency of the baud-rate generator is 16 times the baud rate. So, if a 1,200 Baud rate was required, the output frequency of the baud-rate generator would be 1,200 KHz *16 = 19,200 KHz.

Two 8-bit registers store the divisor in a 16-bit binary format: the Divisor Latch Low Register (DLL) and the Divisor Latch High Register (DLH). The Divisor Latch Low register makes up the lower eight bits of the 16-bit divisor and the Divisor Latch High register makes up the upper eight bits of the 16-bit divisor.

The two Divisor Latch registers must be loaded during initialization to ensure proper operation of the baud-rate generator. If both Divisor Latches are loaded with 0, the 16X output clock is stopped. A Divisor value of 0 in the Divisor Latch Low Register is not allowed.

The reset value of the divisor is hexadecimal 0x0002. The value of hexadecimal 0x0002 implies a value of hexadecimal 0x00 in the Divisor Latch High Register and a value of hexadecimal 0x02 in the Divisor Latch Low Register. The Divisor Latch High Register and Divisor Latch Low Register can only be written after the DLAB bit (bit 7 of the Serial Line Control Register) is set to logic 1.

The baud rate of the UART transmit and receive data is given by:

Baud Rate = 14.7456 MHz/(16xDivisor)

Table 129 shows some commonly used baud rates.

Table 129. Typical Baud Rate Settings

Divisor Latch High Register	Divisor Latch Low Register	Divisor		Baud Rate Generator Clock Output	Baud Rate
		Hexadecimal	Decimal		
0x00	0x01	0x0001	1	14.7456 MHz	921,600
0x00	0x02	0x0002	2	7.3728 MHz	460,800
0x00	0x04	0x0004	4	3.6864 MHz	230,400
0x00	0x08	0x0008	8	1.8432 MHz	115,200
0x00	0x10	0x0010	16	921.6 KHz	57,600
0x00	0x20	0x0020	32	460.8 KHz	28,800
0x00	0x30	0x0030	48	307.2 KHz	19,200
0x00	0x40	0x0040	64	230.4 KHz	14,400
0x00	0x60	0x0060	96	115.2 KHz	9,600
0x00	0xC0	0x00C0	192	76.8 KHz	4,800
0x01	0x80	0x0180	384	38.4 KHz	2,400
0x03	0x00	0x0300	768	19.2 KHz	1,200

10.2.2 Setting Data Bits / Stop Bits / Parity

The Line Control Register (LCR) is an 8-bit register that enables the system programmer to specify the format of the asynchronous data communications exchange. The serial data format consists of a start bit (logic 0), five to eight data bits, an optional parity bit, and one or two stop bits (logic 1). The Line Control Register also contains a bits used for accessing the Divisor Latch Registers and causing a UART break condition. The programmer also has the ability to read the contents of the Line Control Register.

The 8-bit Line Control Register is broken up into seven smaller registers

- Divisor-Latch Access Bit
- Set-Break Bit
- Sticky-Parity Bit
- Even-Parity Select Bit
- Parity-Enable Bit
- Stop-Bits Bit
- Word-Length Select Bits

The Divisor-Latch Access Bit must be set to logic 1 to access the Divisor Latch Registers of the baud-rate generator. When the Divisor-Latch Access Bit is set to logic 0, accessing the same addresses as defined for accessing the Divisor Latch Registers will allow access to the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register.

The Set-Break Control Bit causes a UART break condition to be transmitted to the receiving UART. When Set-Break Control Bit is set to logic 1, the serial output data (TXD) is forced to the spacing or logic 0 state and remains in the spacing state until Set-Break Control Bit is set to logic 0. When the Set-Break Control Bit is set to logic 0, the serial output data is transmitted normally from the transmit-holding register. The Set-Break feature enables the processor to alert a terminal in a computer communications system.

The Sticky-Parity Bit (STKYP) is used to transmit the complement of the Even-Parity Select when parity is enabled. When the Parity-Enable Bit and Sticky-Parity Bit are logic 1, the bit that is transmitted — in the parity bit location of the serial output data stream — is the complement of the Even-Parity Select Bit.

For example, the Even-Parity Select Bit is logic 0 and the parity-bit location of the serial output data stream will be transmitted as logic 1. When the Sticky-Parity Bit and Parity-Enable bits are logic 1, the receiver logic compares the parity bit that is received — in the parity bit location of the serial data input stream — with the complement of the Even-Parity Select Bit.

If the values being compared are not equal, the receiver sets the Parity-Error Bit in Line-Status Register and causes an interrupt error if Line-Status Interrupts are enabled.

If the Even-Parity Select Bit is logic 0, the receiver expects the bit received at the parity-bit location of the serial data input stream to be logic 1. If the parity bit received is logic 0, the parity-error bit is set to logic 1. By forcing the bit value at the parity bit location, rather than calculating a parity value, a system with a master transmitter and multiple receivers can identify some transmitted characters as receiver addresses and the rest of the characters as UART data.

If Parity enable is set to logic 0, Sticky Parity will be ignored.

The Even-Parity Select (EPS) Bit is used to determine the parity type to transmit or check on receive data when the Parity-Enable (PEN) Bit in the Line-Control Register enables parity. When the Parity-Enable bit is logic 1 and the Even-Parity Select bit is logic 0, the parity generator will transmit odd parity and the parity checker will validate against odd parity on the received data.

When the Parity-Enable Bit is logic 1 and the Even-Parity Select Bit is logic 1, the parity generator will transmit even parity and the parity checker will validate against even parity on the received data.

Table 130 shows the serial output data configurations for transmission of the parity bit.

Table 130. UART Transmit Parity Operation

PEN	EPS	Data to be Transmitted (Even or Odd Count of 1s to be Transmitted)	Value of Parity Bit to be Transmitted
1	0	10101010	1
1	0	10101011	0
1	1	10101010	0
1	1	10101011	1
0	X	XXXXXXXX	No Parity Bit Sent

Table 131 shows the serial input data configurations for reception of data containing a parity bit:

Table 131. UART Receive Parity Operation

PEN	EPS	Data being Received + Parity Bit (Even or Odd Count of 1s to be Transmitted)	Value of Parity Bit Should Be
1	0	10101010**1**	1
1	0	10101011**0**	0
1	1	10101010**0**	0
1	1	10101011**1**	1
0	X	XXXXXXXX	Parity Checking Disabled

The Stop-Bits Bit (STB) configures the number of stop bits to be transmitted or received in each serial character. When the Stop-Bits Bit is logic 0, one stop bit is generated in the transmitted data. When the Stop-Bits Bit is logic 1, and a 5-bit word length is selected — via the Word Length select bits in the Line Control Register — 1.5 stop bits will be transmitted.

When the Stop-Bits Bit is logic 1 and the word length is selected as a 6, 7, or 8-bit word, 2 stop bits are transmitted. The UART receiver logic checks the first stop bit only, regardless of the number of stop bits configured by the Stop Bits bit.

The Word-Length Select (WLS) Bits specify the number of data bits contained in each transmitted or received serial character. The Word-Length Select Bits configuration is shown in Table 132.

Table 132. UART Word-Length Select Configuration

WLS		Number of Data Bits Contained in Each Transmitted Or Received Character
Bit 1	Bit 0	
0	0	5-bit character (default)
0	1	6-bit character
1	0	7-bit character
1	1	8-bit character

The Line-Control Register is initialized to hexadecimal 0x00 after reset. The Line-Status Register is initialized to hexadecimal 0x60 after reset.

10.2.3 Using the Modem Control Signals

The IXP42X product line and IXC1100 control plane processors provide two modem control signals, the Clear-to-Send input (CTS_N) and the Ready to Send output (RTS_N).

The Clear-to-Send input signal is sampled by reading the Modem Status Register and the Ready-to-Send output is controlled by the Modem Control Register. The Modem Control Register is a 5-bit register that provides control for four modem control signals:

- OUT1
- OUT2
- Data Terminal Ready
- Ready-to-Send
- Loop-back test control bit

Only one of the four modem-control signals — Ready-to-Send — is replicated to the external pins of the IXP42X product line and IXC1100 control plane processors. The three other modem-control signals are only utilized when the UART is being used in loop-back diagnostic mode. Bit 4 of the Modem-Control Register is the Loop Back Test Mode bit (LOOP). The Loop Back Test Mode bit provides a local loop-back feature for diagnostic testing of the UART.

When the Loop Back Test Mode bit is set to logic 1, the following will event will occur:

- The transmitter serial output is set to logic 1 state.
- The receiver serial input is disconnected from the pin.
- The output of the Transmitter Shift Register is “looped back” into the Receiver Shift Register input.
- The Clear-to-Send input signal is disconnected from the pin
- The Ready-to-Send output signal is forced to logic 1.
- The lower four bits of the Modem Control Register are connected to the upper four bits of the Modem Status Register

  Status register bit mapping, while in loop back mode:

  — DTR = 1 forces DSR to a 1

  — RTS = 1 forces CTS to a 1

  — OUT1 = 1 forces RI to a 1

  — OUT2= 1 forces DCD to a 1

Leaving loop-back mode and returning to normal mode may result in unpredictable activation of the Modem Status Register (MSR). It is recommended that the Modem Status Register be read once to clear the Modem Status Bits in the Modem Status Register.

In the loop-back diagnostic mode, data that is transmitted will be immediately received. The ability to loop back the transmit data path to the receive path allows the Intel XScale processor to verify the transmit data path and receive data path of the UART. The transmit interrupts, receive interrupts, and modem-control interrupts are operational, when placed in the loop-back diagnostic mode. The Modem-Control Register Bits — instead of the Modem Control inputs — will activate the modem-control interrupts. A break signal can also be transferred — from the transmitter section to the receiver section — while operating in loop-back diagnostic mode.

Bit 1 of the Modem-Control Register is the Request-to-Send Bit. The Request-to-Send Control Bit is used to program the RTS_N output pin. When the Request-to-Send control bit is programmed to logic 0, the RTS_N signal will output logic 1. When the Request-to-Send Control Bit is programmed to logic 1, the RTS_N signal will output logic 0.

The Modem-Status Register is used to monitor the status of an external modem or data set. The Modem Status Register is an 8-bit register that is used to detect when a modem is capable of accepting new data. The status of the modem to accept incoming data is monitored by reading the Clear-to-Send Bit of the Modem Status Register and the Delta Clear-to-Send Bit of the Modem Status Register. The other six bits of the Modem-Status Register can be used for UART debug purposed only.

Modem Status Register Bit 4 is the Clear-to-Send (CTS). The Clear-to-Send Bit will be the complement of the Clear-to-Send (CTS_N) input signal. The Clear-to-Send signal will be connected to the Ready-to-Send bit of the Modem Control Register when the LOOP bit — in the Modem Control Register — is set to logic 1.

- CTS = logic 0 = CTS_N pin is 1
- CTS = logic 1 = CTS_N pin is 0

Modem Status Register Bit 0 is the Delta Clear-to-Send (DCTS). The Delta Clear-to-Send bit will inform the IXP42X product line and IXC1100 control plane processors that nothing has happened to the Clear-to-Send Status since the last time that the Modem-Status Register was read. The Delta Clear-to-Send bit will be set to logic 0 after a read of the Modem-Status Register.

- DCTS = logic 0 = No change in CTS_N pin since last read of the Modem-Status Register
- DCTS = logic 1 = CTS_N pin has changed state since the last read of the Modem-Status Register

The Modem-Control Register is initialized to hexadecimal 0x00 after reset. The Modem-Status Register is initialized to hexadecimal 0x00 after reset.

10.2.4 UART Interrupts

The UART Interrupt Enable Register (IER) is an 8-bit register that enables five types of UART based interrupts and enables the UART functionality and other control functionality not used by the IXP42X product line and IXC1100 control plane processors.

UART Interrupt Enable Register bit 6 is the UART Unit Enable (UUE) bit. When the UART Unit Enable bit is set to logic 0, the UART will be completely non-functional. Likewise, when the UART Unit Enable bit is set to logic 1, the UART will be enabled.

UART Interrupt Enable Register bits 4 through 0 represent five different interrupt types that can be individually enabled/disabled:

- Receiver Time Out Interrupt Enable (RTOIE)
- Modem Interrupt Enable (MIE)
- Receiver Line Status Interrupt Enable (RLSE)
- Transmit Data Request Interrupt Enable (TIE)
- Receiver Data Available Interrupt Enable (RAVIE)

The Receiver Line Status Interrupt Enable allows interrupts to be generated to the IXP42X product line and IXC1100 control plane processors Interrupt Controller and captured in the UART Interrupt Identification Register (IIR) when a receive error is detected. Such Receiver Line Status Conditions that would cause the interrupt to occur are:

- Overrun
- Parity
- Framing
- Break
- FIFO error

In FIFO mode, each received character carries the line status along with the character. When in FIFO Mode, the Receive Line Status Interrupt will be active on the character located in the bottom of the Receive FIFO. If a Line Status error condition is detected on the character in the bottom of the Receive FIFO, a Receive Line Status Interrupt is generated. Reading the Line Status Register clears the Receive Line Status Interrupt.

When in Non-FIFO Mode, the Receive Line Status Interrupt will be active on the character located in the bottom of the Receive Buffer Register. If a line-status error condition is detected on the character in the Receive Buffer Register, a Receive Line Status Interrupt is generated.

The Receiver Data Available Interrupt Enable allows interrupts to be generated to the IXP42X product line and IXC1100 control plane processors' Interrupt Controller and captured in the UART Interrupt Identification Register (IIR) when UART data is available to be read by the Intel XScale processor.

When the UART is in FIFO mode, the Receive Data Available interrupt will be encoded in the Interrupt Identification Register, after the FIFO trigger level defined in the FIFO Control Register (FCR) is reached. When the UART is in non-FIFO mode, the Receive Data Available interrupt will be encoded in the Interrupt Identification Register after data is in the Receive Buffer Register (RBR).

When operating in FIFO Mode, the Receive Data Available Interrupt is cleared, when the FIFO drops below the programmed trigger level. When operating in Non-FIFO Mode, the Receive Data Available Interrupt is cleared when the received character is read by the IXP42X product line and IXC1100 control plane processors from the Receive Holding Register.

The Receiver Interrupt Time Out Enable can be used only in FIFO Mode and allows interrupts to be generated to the IXP42X product line and IXC1100 control plane processors' Interrupt Controller and captured in the UART Interrupt Identification Register (IIR). This happens when:

- At least one character is available in the receive FIFO
- The last character received by the UART receive interface occurred more than four continuous character times ago
- The most-recent read of the receive FIFO, by the IXP42X product line and IXC1100 control plane processors, was more than four continuous character times ago

For example, the maximum time between a received character and a Receive Character Time-Out Interrupt is 160 ms at 1,200 baud with a 12-bit receive character (i.e., 1 start, 8 data, 1 parity, and 2 stop bits).

```
(1/ ((1200characters/second)/12characters))) * 4
characters =

= 0.040 seconds
 = 40 ms
```

The time-out interrupt is cleared by the IXP42X product line and IXC1100 control plane processors reading the Receive FIFO or setting the RESETRF bit to logic 1, in the FIFO Control Register.

If a Receive Character Time-Out Interrupt is active, the interrupt-time-out counter will be reset only after the IXP42X product line and IXC1100 control plane processors reads a character from the Receive FIFO. If a Receive Character Time-Out Interrupt is non-active, the interrupt time-out counter will be reset after the IXP42X product line and IXC1100 control plane processors reads a character from the Receive FIFO or when a new character is placed into the Receive FIFO by the receive interface.

The Transmit Data Request Interrupt Enable can be used in Non-FIFO mode or FIFO Mode.

In Non-FIFO Mode, the Transmit Data Request Interrupt Enable allows interrupts to be generated to the IXP42X product line and IXC1100 control plane processors' Interrupt Controller and captured in the UART Interrupt Identification Register (IIR), when the Transmit Holding Register is empty. Reading the Interrupt Identification Register or writing a new character into the Transmit Holding Register clears the Transmit Data Request Interrupt.

In FIFO Mode, the Transmit Data Request Interrupt Enable allows interrupts to be generated to the IXP42X product line and IXC1100 control plane processors' Interrupt Controller and captured in the UART Interrupt Identification Register (IIR), when the Transmit FIFO is half empty or less.

The Transmit FIFO size is 64 characters. When 32 or fewer characters are remaining in the Transmit FIFO to be transmitted, the Transmit Data Request Interrupt will be generated. Reading the Interrupt Identification Register — or writing a new data into the Transmit FIFO — clears the Transmit Data Request Interrupt.

The Modem Status Interrupt Enable allows interrupts to be generated to the IXP42X product line and IXC1100 control plane processors Interrupt Controller — and captured in the UART Interrupt Identification Register (IIR) — when the Clear-to-Send, Data-Set-Ready, Ring-Indicator, or Received-Line Signal Detect bits in the Modem Status Register are set to logic 1. Reading the Modem Status Register clears the Modem Status Interrupt.

Clearing the appropriate bit of the Interrupt-Enable Register may disable each of the interrupt types previously described. Similarly, by setting the appropriate bits, selected interrupts can be enabled on a per-interrupt basis.

When the UART interrupts are disabled, the UART is placed into polled mode of operation. Since the UART receiver and the UART transmitter are controlled separately, either one or both interfaces can be placed in the polled mode of operation.

In the polled mode of operation, software routines running on the Intel XScale processor checks receiver and transmitter status via the Line Status Register. Line Status Register bit 0 will be logic 1, when a character is available to be read from the

Receive Interface. Lines Status Register bits 1 through 4 specify which error(s) has occurred — for the character at the bottom of the FIFO or in the Receive Buffer Register.

In FIFO mode, Line Status Register bit 1 through 3 is stored with each received character in the Receive FIFO. The Line Status Register shows the status bits of the character at the bottom of the Receive FIFO. When the character at the bottom of the FIFO has errors, the Line Status error bits are set and are not cleared until the Intel XScale processor reads the Line Status Register. Even if the character in the FIFO is read — and a new character is now at the bottom of the FIFO — the interrupts will not be cleared until the Line-Status Register is read.

Character-error status is handled in the same way as when the UART is operating in interrupt mode of operation. Setting Line-Status Register bit 5 to logic 1 indicates that the Transmit FIFO or the Transmit Holding Register is requesting data. Line Status Register bit 6 identifies that both the Transmit FIFO and the Transmit Shift Register have no data. Line Status Register bit 7 indicates the status of any errors in the Receive FIFO.

In non-FIFO mode, three of the LSR register bits — parity error, framing error, and break interrupt — show the error status of the character that has just been received.

The Receive Time-Out Interrupt is separated from the Receive-Data-Available Interrupt to prevent an Interrupt Controller routine and a Data Service controller routing from servicing the receive FIFO at the same time. Bit 7 of the Interrupt-Enable Register is used as the enable bit of Data Service requests. Bit 5 of the Interrupt Enable Register is used as the enable bit of NRZ-coding enable.

Bits 7 and 5 are not implemented by the IXP42X product line and IXC1100 control plane processors. The use of bit 7 through bit 4, of the Interrupt-Enable Register, is defined differently from the register definition of standard 16550 UART.

The Interrupt-Enable Register is initialized to all zeros after receiving a reset. The Interrupt Identification Register is a hexadecimal 0x01. Bit 5 and Bit 4 of the Interrupt Identification Register will always be logic 0.

10.3 Transmitting and Receiving UART Data

Transmitting and receiving data — using the IXP42X product line and IXC1100 control plane processors' UARTs — is achievable in two modes: Non-FIFO Mode and FIFO Mode.

In Non-FIFO mode, data will be transmitted and received using two registers — the Transmit-Holding Register (THR) and the Receive-Buffer Register (RBR) — along with the UART control, status, and interrupt registers.

In FIFO mode, data will be transmitted and received using two 64-entry FIFOs, the Transmit FIFO, and the Receive FIFO — along with the UART Control, Status, and Interrupt registers.

The Transmit FIFO is 64 entries deep by 8 bits wide. The Transmit FIFO sizing allows a complete 8-bit data character to be stored in each entry. When characters smaller than 8 bits are transmitted, they are right-justified.

If a 5-bit character is to be transmitted, the character is represented by a binary 10110. The value located in the FIFO entry will be hexadecimal 0x16.

The Receive FIFO is 64 entries deep and 11 bits wide. The Receive FIFO sizing allows for an 8-bit character to be received along with the over-run flag, parity error flag, and framing error flag for each received character. Smaller characters will be right-justified, as described for the transmit FIFO.

The error flags position will remain constant, independent of the character size. The mode of operation and FIFO control parameters will be programmed using the FIFO Control Register (FCR).

The FIFO Control Register is an 8-bit register that configures the UARTs' mode of operation. The Transmit and Receive FIFO Enable Bit (TRFIFOE) — Bit 0 of the FIFO Control Register — determines the UARTs' mode of operation: FIFO Mode or Non-FIFO Mode. When set to logic 0, the UART will function in Non-FIFO Mode. When set to logic 1, the UART will function in FIFO Mode.

Two bits of the FIFO Control Register are used to reset the Transmit and Receive FIFO: the Reset Transmit FIFO bit (bit 2 of FCR) and the Reset Receive FIFO (bit 1 of FCR). Writing logic 0 to these bits has no effect. Writing logic 1 to the Reset Transmit FIFO bit will cause the Transmit FIFO counter to be reset to 0 and the transmit-data request bit to be set in the Line-Status Register.

Writing logic 1 to the Reset Receive FIFO bit will cause the Receive FIFO counter to be reset to 0 and the data ready bit in the Line-Status Register to be cleared. The Overrun Error Flag, Parity Error Flag, Framing Error Flag, and Break Interrupt Flag in the Line Status Register will remain unaltered. The Reset Transmit FIFO and Reset Receive FIFO will be cleared autonomously when the reset has been completed.

The Receive FIFO interrupt trigger level also is set in the FIFO Control Register. The Receive FIFO interrupt trigger level is used to generate an interrupt when the number of characters in the Receive FIFO is greater than to or equal to the trigger-level value. The Interrupt Trigger Level is defined as bit 6 and bit 7 of the FIFO Control Register. The bit definitions are shown in Table 133.

Table 133. UART FIFO Trigger Level

Interrupt Trigger Level [7:6]	Description
00	1 byte or more in the FIFO causes an interrupt
01	8 bytes or more in the FIFO causes an interrupt
10	16 bytes or more in the FIFO causes an interrupt
11	32 bytes or more in the FIFO causes an interrupt

The UART must be configured prior to transmitting and receiving data to and from the UART. The Transmit Holding Register (THR) is used to transmit characters over the UART interface. The Receive Buffer Register is used to receive characters from the UART interface.

Transmitting UART data can be implemented using FIFO Mode or Non-FIFO mode. In FIFO mode, writing a character to the Transmit Holding Register will put data on the top of the transmit FIFO. In Non-FIFO mode, writing a character to the Transmit Holding Register will put data in the Transmit Holding Register. The next character transmitted will be the character contained in the Transmit Holding Register.

If characters less than 8 bits are sent, the characters will need to be right-justified. For example, if a 5-bit data character is to be transmitted with a binary value of 01011. The data written to the Transmit Holding Register will need to be written as hexadecimal 0x0B.

Receiving UART data can be implemented using FIFO Mode or Non-FIFO mode. In FIFO mode, reading a character from the Receive Buffer Register will read a character from the bottom of the receive FIFO. In Non-FIFO mode, reading a character from the Receive Buffer Register will read the data contained in the Receive Buffer Register. The next character received will be the character contained in the Receive Buffer Register.

If characters less than 8 bits are received, the characters will need to be right-justified. For example, if a 5-bit data character is received having a binary value of 00101. The data read from the Receive Buffer Register will be a hexadecimal 0x05. (Notice that the three most-significant bits of the byte are filled with zeros.)

The UART transmit data pin is logic 1 and the transmit FIFO and receive FIFO pointers are initialized to the empty value after reset.

10.4 Register Descriptions

There are 13 registers to monitor and control the UART. The registers are all 32 bits in size, but only the lower 8 bits have valid data. The 13 UART registers share nine address locations in the
I/O address space.

Note that the state of the Divisor Latch Bit (DLAB) — the **most**-significant bit of the Serial Line Control Register — affects the selection of certain of the UART registers. The DLAB bit must be set high by the system software to access the Baud Rate Generator Divisor Latches.

Table 134. High-Speed UART Registers Overview

Address	DLAB	R/W	Name	Description
0xC8000000	0	R	RBR	Receive Buffer Register
	0	W	THR	Transmit Holding Register
	1	R/W	DLL	Divisor Latch Low Register
0x C8000004	0	R/W	IER	Interrupt Enable Register
	1	R/W	DLH	Divisor Latch High Register
0x C8000008	0/1	R	IIR	Interrupt Identification Register
	0/1	W	FCR	FIFO Control Register
0x C800000C	0/1	R/W	LCR	Line Control Register
0x C8000010	0/1	R/W	MCR	Modem Control Register
0x C8000014	0/1	R	LSR	Line Status Register
0x C8000018	0/1	R	MSR	Modem Status Register
0x C800001C	0/1	R/W	SPR	Scratch Pad Register
0x C8000020	0	R/W	ISR	Slow Infrared Select Register

10.4.1 Receive Buffer Register

Register Name:	RBR				
Hex Offset Address:	0xC800 0000		Reset Hex Value:	0x00000000	
Register Description:	Receive Buffer Register				
Access: Read Only.					

31																					8	7							0
(Reserved)																						RBR							

Register		RBR
Bits	**Name**	**Description**
31:8		(Reserved)
7:0	RBR	In non-FIFO mode, this register holds the character received by the UART's Receive Shift Register. If fewer than 8 bits are received, the bits are right-justified and the leading bits are zeroed. In FIFO mode, this register latches the value of the data byte at the bottom of the Receive FIFO. The DLAB bit in the Line Control Register must be set to logic 0 to access this register.

10.4.2 Transmit Holding Register

Register Name:	THR				
Hex Offset Address:	0xC800 0000		**Reset Hex Value:**	0x00000000	
Register Description:	Transmit Holding Register				
Access: Write Only.					

31																					**8**	**7**							**0**
(Reserved)																						THR							

Register		THR
Bits	**Name**	**Description**
31:8		(Reserved)
7:0	THR	In Non-FIFO mode, this register holds the next data byte to be transmitted. When the Transmit Shift Register becomes empty, the contents of the Transmit Holding Register are loaded into the shift register and the transmit data request (TDRQ) bit in the Line Status Register is set to 1. In FIFO mode, writing to THR puts data to the top of the Transmit FIFO. The data at the bottom of the Transmit FIFO is loaded to the shift register when the shift register becomes empty. The DLAB bit in the Line Control Register must be set to logic 0 to access this register.

10.4.3 Divisor Latch Low Register

Register Name:	DLL				
Hex Offset Address:	0xC800 0000		**Reset Hex Value:**	0x00000002	
Register Description:	Divisor Latch Low Register				
Access: Read/Write.					

31																					**8**	**7**							**0**
(Reserved)																						DLL							

Register		DLL
Bits	**Name**	**Description**
31:8		(Reserved)
7:0	DLL	Lower byte of compare-value used by the baud rate generator. The DLAB bit in the Line Control Register must be set to logic 1 to access this register.

10.4.4 Divisor Latch High Register

Register Name:	DLH				
Hex Offset Address:	0xC800 0004		**Reset Hex Value:**	0x00000000	
Register Description:	Divisor Latch High Register				
Access: Read/Write.					

31																					**8**	**7**							**0**
(Reserved)																						DLH							

Register		DLH
Bits	**Name**	**Description**
31:8		(Reserved)
7:0	DLH	Upper byte of compare-value used by the baud-rate generator. The DLAB bit in the Line Control Register must be set to logic 1 to access this register.

10.4.5 Interrupt Enable Register

The DLAB bit in the Line-Control Register must be set to logic 0 to access this register.

Note: All interrupts will be masked, if OUT2 = 0 of the Modem Control Register (see "Modem Control Register" on page 352). If Interrupts need to be enabled, set OUT2 = 1 of the Modem Control Register.

Register Name:	IER				
Hex Offset Address:	0xC800 0004		**Reset Hex Value:**	0x00000000	
Register Description:	Interrupt Enable Register				
Access: See below.					

31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																						DMAE	UUE	NRZE	RTOIE	RIE	RLSE	TIE	RAVIE

Register		IER
Bits	**Name**	**Description**
31:8		(Reserved)
7	DMAE	**DMA Requests Enable**: 0 = DMA requests are disabled 1 = DMA requests are enabled Not Used On the Intel® IXP42X product line and IXC1100 control plane processors
6	UUE	**UART Unit Enable**: 0 = the unit is disabled 1 = the unit is enabled
5	NRZE	**NRZ coding Enable**: 0 = NRZ coding disabled 1 = NRZ coding enabled Not Used On the Intel® IXP42X product line and IXC1100 control plane processors
4	RTOIE	Receiver Time Out Interrupt Enable: 0 = Receiver data Time out interrupt disabled 1 = Receiver data Time out interrupt enabled
3	RIE	**Modem Interrupt Enable**: 0 = Modem Status interrupt disabled 1 = Modem Status interrupt enabled
2	RLSE	**Receiver Line Status Interrupt Enable**: 0 = Receiver Line Status interrupt disabled 1 = Receiver Line Status interrupt enabled The DLAB bit in the Line Control Register must be set to logic 0 to access this register.
1	TIE	**Transmit Data request Interrupt Enable**: 0 = Transmit FIFO Data Request interrupt disabled 1 = Transmit FIFO Data Request interrupt enabled
0	RAVIE	**Receiver Data Available Interrupt Enable**: 0 = Receiver Data Available (Trigger level reached) interrupt disabled 1 = Receiver Data Available (Trigger level reached) interrupt enabled

10.4.6 Interrupt Identification Register

In order to minimize software overhead during data character transfers, the UART prioritizes interrupts into four levels and records these in the Interrupt-Identification Register. The Interrupt-Identification Register (IIR) stores information indicating that a prioritized interrupt is pending and the source of that interrupt.

Priority Level	Interrupt origin
1 (highest)	**Receiver Line Status:** One or more error bits were set.
2	**Received Data is available:** In FIFO mode, trigger level was reached. In non-FIFO mode, RBR has data.

Priority Level	Interrupt origin
2	**Receiver Time out occurred:** It happens in FIFO mode only, when there is data in the receive FIFO but no activity for a time period.
3	**Transmitter requests data:** In FIFO mode, the transmit FIFO is half or more than half empty. In non-FIFO mode, THR is read already.
4 (lowest)	**Modem Status:** One or more of the modem input signals has changed state.

Register Name:	IIR				
Hex Offset Address:	0xC800 0008		Reset Hex Value:	0x00000001	
Register Description:	Interrupt Identification Register				
Access: See below.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			FIFOES		(Rsvd)		TOD	IID		IP_N

Register		IIR
Bits	Name	Description
31:8		(Reserved)
7:6	FIFOES	FIFO Mode Enable Status: 00 = Non-FIFO mode is selected 01 = (Reserved) 10 = (Reserved) 11 = FIFO mode is selected (TRFIFOE = 1)
5:4		(Reserved)
3	TOD	Time Out Detected: 0 = No time out interrupt is pending 1 = Time out interrupt is pending. (FIFO mode only)
2:1	IID	Interrupt Source Encoded: 00 = Modem Status (CTS, DSR, RI, DCD modem signals changed state) 01 = Transmit FIFO requests data 10 = Received Data Available 11 = Receive error (Overrun, parity, framing, break, FIFO error)
0	IP_N	Interrupt Pending: 0 = Interrupt is pending. (Active low) 1 = No interrupt is pending

Table 135. UART IDD Bit Mapping

Interrupt ID Bits				Interrupt SET/RESET Function			
3	2	1	0	Priority	Type	Source	RESET Control
0	0	0	1	-	None	No Interrupt is pending	-
0	1	1	0	Highest	Receiver Line Status	Overrun Error, Parity Error, Framing Error, Break Interrupt	Reading the Line Status Register.
0	1	0	0	Second-Highest	Received Data Available.	Non-FIFO mode: Receive Buffer is full. FIFO mode: Trigger level was reached.	Non-FIFO mode: Reading the Receiver Buffer Register. FIFO mode: Reading bytes until Receiver FIFO drops below trigger level or setting RESETRF bit in FCR register.
1	1	0	0	Second-Highest	Character Time-out indication.	FIFO Mode only: At least 1 character is in receiver FIFO and there was no activity for a time period.	Reading the Receiver FIFO or setting RESETRF bit in FCR register.
0	0	1	0	Third-Highest	Transmit FIFO Data Request	Non-FIFO mode: Transmit Holding Register Empty FIFO mode: Transmit FIFO has half or less than half data.	Reading the IIR Register (if the source of the interrupt) or writing into the Transmit Holding Register. Reading the IIR Register (if the source of the interrupt) or writing to the Transmitter FIFO.
0	0	0	0	Fourth-Highest	Modem Status	Clear-to-Send, Data Set Ready, Ring Indicator, Received Line Signal Detect	Reading the Modem Status Register.

10.4.7 FIFO Control Register

FCR is a write-only register that is located at the same address as the IIR. (IIR is a read-only register.) FCR enables/disables the transmitter/receiver FIFOs, clears the transmitter/receiver FIFOs, and sets the receiver FIFO trigger level.

Register Name:	FCR				
Hex Offset Address:	0xC800 0008		Reset Hex Value:	0x00000000	
Register Description:	FIFO Control Register				
Access: Write Only.					

31	...	8	7	6	5	...	3	2	1	0
(Reserved)			ITL		(Rsvd)			RESETTF	RESETRF	TRFIFOE

Register		FCR
Bits	**Name**	**Description**
31:8		(Reserved)
7:6	ITL	**Interrupt Trigger Level:** When the number of entries in the receive FIFO equals the interrupt trigger level programmed into this field and the Received Data Available Interrupt is enabled (via IER), an interrupt is generated and appropriate bits are set in the IIR. 00 = 1 byte or more in FIFO causes interrupt 01 = 8 bytes or more in FIFO causes interrupt 10 = 16 bytes or more in FIFO causes interrupt 11 = 32 bytes or more in FIFO causes interrupt
5:3		(Reserved)
2	RESETTF	**Reset Transmitter FIFO:** When RESETTF is logic 1, the transmitted FIFO count logic is set to 0, effectively clearing all the entries in the transmit FIFO. The TDRQ bit in LSR are set and IIR shows a transmitter requests data interrupt if the TIE bit in the IER register is set. The transmitter shift register is not cleared; it completes the current transmission. After the FIFO is cleared, RESETTF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The transmitter FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0.
1	RESETRF	**Reset Receiver FIFO:** When RESETRF is set to 1, the receive FIFO counter is reset to 0, effectively clearing all the entries in the receive FIFO. The DR bit in the LSR is reset to 0. All the error bits in the FIFO and the FIFOE bit in LSR are cleared. Any error bits, OE, PE, FE or BI that had been set in LSR are still set. The receiver shift register is not cleared. If IIR had been set to Received Data Available, it is cleared. After the receive FIFO is cleared, RESETRF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The receiver FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
0	TRFIFOE	**Transmit and Receive FIFO Enable:** TRFIFOE enables/disables the transmitter and receiver FIFO s. When TRFIFOE = 1, both FIFOs are enabled (FIFO Mode). When TRFIFOE = 0, the FIFO s are both disabled (non-FIFO Mode). Writing a 0 to this bit clears all bytes in both FIFO s. 0 = FIFOs are disabled 1 = FIFOs are enabled

10.4.8 Line Control Register

Register Name:	LCR																												
Hex Offset Address:	0xC800 000C													**Reset Hex Value:**								0x00000000							
Register Description:	Line Control Register																												
Access: Read/Write.																													
31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																						DLAB	SB	STKYP	EPS	PEN	STB	WLS	

Register		LCR
Bits	**Name**	**Description**
31:8		(Reserved)
7	DLAB	**Divisor Latch Access Bit:** This bit must be set to logic 1 to access the Divisor Latches of the Baud Rate Generator during a READ or WRITE operation. It must be set low (logic 0) to access the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register.
6	SB	**Set break:** This bit causes a break condition to be transmitted to the receiving UART. When SB is set to logic 1, the serial output (TXD) is forced to the spacing (logic 0) state and remains there until SB is set to logic 0. This bit acts only on the TXD pin. 0 = No effect on TXD output 1 = Forces TXD output to 0 (space)
5	STKYP	**Sticky Parity:** This bit is the “sticky parity” bit, which can be used in multiprocessor communications. When PEN and STKYP are logic 1, the bit that is transmitted in the parity bit location (the bit just before the stop bit) is the complement of the EPS bit. If EPS is 0, then the bit at the parity bit location will be transmitted as a 1. In the receiver, if STKYP and PEN are logic 1, the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. If the values being compared are not equal, the receiver sets the Parity Error bit in LSR and causes an error interrupt if line status interrupts were enabled. For example, if EPS is 0, the receiver expects the bit received at the parity bit location to be 1. If it is not, then the parity error bit is set. By forcing the bit value at the parity bit location — rather than calculating a parity value — a system with a master transmitter and multiple receivers can identify some transmitted characters as receiver addresses and the rest of the characters as data. If PEN = 0, STKYP is ignored. 0 = No effect on parity bit 1 = Forces parity bit to be opposite of EPS bit value
4	EPS	Even-Parity Select: 0 = Sends or checks for odd parity 1 = Sends or checks for even parity
3	PEN	**Parity enable:** This is the parity enable bit. When PEN is logic 1, a parity character is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. 0 = No parity function 1 = Allows parity generation and checking
2	STB	**Stop bits:** This bit specifies the number of stop bits transmitted and received in each serial character. If STB is a logic 0, one stop bit is generated in the transmitted data. If STB is a logic 1 when a 5-bit word length is selected via bits 0 and 1, then 1 and one half stop bits are generated. If STB is a logic 1 when either a 6, 7, or 8-bit word is selected, then two stop bits are generated. The receiver checks the first stop bit only, regardless of the number of stop bits selected. 0 = One stop bit 1 = Two stop bits, except for 5-bit character, then 1.5 bits
1:0	WLS	**Word-Length Select:** Specify the number of data bits in each transmitted or received serial character. 00 = 5-bit character (default) 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character

10.4.9 Modem Control Register

Register Name:	MCR				
Hex Offset Address:	0xC800 0010		Reset Hex Value:	0x00000000	
Register Description:	Modem Control Register				
Access: Read/Write.					

31	...	5	4	3	2	1	0
(Reserved)			LOOP	OUT2	OUT1	RTS	DTR

Register		MCR
Bits	**Name**	**Description**
31:5		(Reserved)
4	LOOP	**Loop back test mode:** This bit provides a local Loop back feature for diagnostic testing of the UART. When LOOP is set to logic 1, the following will occur: • The transmitter serial output is set to a logic 1 state. • The receiver serial input is disconnected from the pin. • The output of the Transmitter Shift register is “looped back” into the receiver shift register input. • The four modem-control inputs (CTS#, DSR#, DCD#, and RI#) are disconnected from the pins and the modem control output pin RTS_N is forced to its inactive state. ***Note:*** Coming out of the loop back test mode may result in unpredictable activation of the delta bits (bits 3:0) in the Modem Status Register (MSR). It is recommended that MSR is read once to clear the delta bits in the MSR. The lower four bits of the Modem Control register are connected to the upper four Modem Status register bits: • DTR = 1 forces DSR to a 1 • RTS = 1 forces CTS to a 1 • OUT1 = 1 forces RI to a 1 • OUT2= 1 forces DCD to a 1 — 0 = Normal UART operation — 1 = Test mode UART operation
3	OUT2	Interrupt Mask: The OUT2 bit is used to mask the UARTs’ interrupt output to the Interrupt Controller unit. OUT2 = 0 UART interrupt masked OUT2 = 1 UART interrupt not masked When LOOP=1, this bit is not used as interrupt mask. The interrupt goes to the processor without mask in loop-back mode.
2	OUT1	**Test bit:** This bit is used only in loop-back test mode. See LOOP row, above.
1	RTS	**Request to Send:** This bit controls the Request to Send (RTS_N) output pin. 0 = RTS_N pin is 1 1 = RTS_N pin is 0
0	DTR	Data Terminal Ready: 0 = DTR# pin is 1 1 = DTR# pin is 0

10.4.10 Line Status Register

Register Name:	LSR				
Hex Offset Address:	0xC800 0014		Reset Hex Value:	0x00000060	
Register Description:	Line Status Register				
Access: Read Only.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			FIFOE	TEMT	TDRQ	BI	FE	PE	OE	DR

Register		LSR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		(Reserved)
7	FIFOE	**FIFO Error Status:** In non-FIFO mode, this bit is 0. In FIFO Mode, FIFOE is set to 1 when there is at least a parity error, framing error, or break indication for any of the characters in the FIFO. Note that a processor read to the Line Status register does not reset this bit. FIFOE is reset when all error bytes have been read from the FIFO. 0 = Non-FIFO mode or no errors in receiver FIFO 1 = At least one character in receiver FIFO has errors
6	TEMT	**Transmitter Empty:** TEMT is set to a logic 1 when the Transmit Holding register and the Transmitter Shift register are both empty. It is reset to logic 0 when either the Transmit Holding register or the transmitter shift register contains a data character. In FIFO mode, TEMT is set to 1 when the transmitter FIFO and the Transmit Shift register are both empty.
5	TDRQ	**Transmit Data Request:** TDRQ indicates that the UART is ready to accept a new character for transmission. In Non-FIFO mode, The TDRQ bit is set to logic 1 when a character is transferred from the Transmit Holding register into the Transmit Shift register. The bit is reset to logic 0 concurrently with the loading of the Transmit Holding register by the processor. In FIFO mode, TDRQ is set to 1 when half of the characters in the FIFO have been loaded into the Shift register or the RESETTF bit in FCR has been set to 1. It is cleared when the FIFO has more than half data. If more than 64 characters are loaded into the FIFO, the excess characters are lost.
4	BI	**Break Interrupt:** BI is set to a logic 1 when the received data input is held in the spacing (logic 0) state for longer than a full word transmission time (that is, the total time of Start bit + data bits + parity bit + stop bits). The Break indicator is reset when the processor reads the Line Status Register. In FIFO mode, only one character (equal to 00H) is loaded into the FIFO regardless of the length of the break condition. BI shows the break condition for the character at the bottom of the FIFO, not the most recently received character.
3	FE	**Framing Error:** FE indicates that the received character did not have a valid stop bit. FE is set to a logic 1 when the bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). The FE indicator is reset when the processor reads the Line Status Register. The UART will resynchronize after a framing error. To do this, it assumes that the framing error was due to the next start bit, so it samples this “start” bit twice and then takes in the “data”. In FIFO mode, FE shows a framing error for the character at the bottom of the FIFO, not for the most recently received character.

Register		LSR (Sheet 2 of 2)
Bits	**Name**	**Description**
2	PE	**Parity Error:** PE indicates that the received data character does not have the correct even or odd parity, as selected by the even parity select bit. The PE is set to logic 1 upon detection of a parity error and is reset to logic 0 when the processor reads the Line Status register. In FIFO mode, PE shows a parity error for the character at the bottom of the FIFO, not the most recently received character.
1	OE	**Overrun Error:** In non-FIFO mode, OE indicates that the processor did not read data the receiver buffer register before the next character was received. The new character is lost. In FIFO mode, OE indicates that all 64 bytes of the FIFO are full and the most recently received byte has been discarded. The OE indicator is set to logic 1 upon detection of an overrun condition and reset when the processor reads the Line Status register.
0	DR	**Data Ready:** Bit 0 is set to logic 1 when a complete incoming character has been received and transferred into the receiver buffer register or the FIFO. In non-FIFO mode, DR is reset to 0 when the receive buffer is read. In FIFO mode, DR is reset to a logic 0 if the FIFO is empty (last character has been read from RBR) or the RESETRF bit is set in the FCR. 0 = No data has been received 1 = Data is available in RBR or the FIFO

10.4.11 Modem Status Register

This register provides the current state of the control lines from the modem or data set (or a peripheral device emulating a modem) to the processor. In addition to this current state information, four bits of the Modem Status register provide change information. The 3:0 bits are set to a logic 1 when a control input from the modem changes state. The bits are reset to a logic 0 when the processor writes ones to the bits of the Modem Status Register.

Note: When bits 0, 1, 2, or 3 are set to logic 1, a Modem Status Interrupt is generated, if bit 3 of the Interrupt Enable Register is set.

Register Name:	**MSR**																										
Hex Offset Address:	0xC800 0018													**Reset Hex Value:**								0x00000000					
Register Description:	Modem Status Register																										
Access: Read Only.																											
31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																						DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS

Register		MSR
Bits	**Name**	**Description**
31:8		(Reserved)
7	DCD	**Data Carrier Detect:** This bit is the complement of the Data Carrier Detect (DCD#) input. This bit is equivalent to bit OUT2 of the Modem Control register if LOOP in the MCR is set to 1. 0 = DCD# pin is 1 1 = DCD# pin is 0
6	RI	**Ring Indicator:** This bit is the complement of the ring Indicator (RI#) input. This bit is equivalent to bit OUT1 of the Modem Control register if LOOP in the MCR is set to 1. 0 = RI# pin is 1 1 = RI# pin is 0
5	DSR	**Data Set Ready:** This bit is the complement of the Data Set Ready (DSR#) input. This bit is equivalent to bit DTR of the Modem Control register if LOOP in the MCR is set to 1. 0 = DSR# pin is 1 1 = DSR# pin is 0
4	CTS	**Clear to Send:** This bit is the complement of the Clear to Send (CTS#) input. This bit is equivalent to bit RTS of the Modem Control register if LOOP in the MCR is set to 1. 0 = CTS# pin is 1 1 = CTS# pin is 0
3	DDCD	Delta Data Carrier Detect: 0 = No change in DCD# pin since last read of MSR 1 = DCD# pin has changed state
2	TERI	Trailing Edge Ring Indicator: 0 = RI# pin has not changed from 0 to 1 since last read of MSR 1 = RI# pin has changed from 0 to 1
1	DDSR	Delta Data Set Ready: 0 = No change in DSR# pin since last read of MSR 1 = DSR# pin has changed state
0	DCTS	Delta Clear To Send: 0 = No change in CTS# pin since last read of MSR 1 = CTS# pin has changed state

10.4.12 Scratch-Pad Register

This read/write register has no effect on the UART. The register is intended as a scratch-pad register for use by the programmer.

Register Name:	SPR				
Hex Offset Address:	0xC800 001C		**Reset Hex Value:**	0x00000000	
Register Description:	Scratch Pad Register				
Access: Read/Write.					

31																					8	7							0
(Reserved)																						SCR							

Register		SPR
Bits	**Name**	**Description**
31:8		(Reserved)
7	SCR	No effect on UART functionality

10.4.13 Infrared Selection Register

The Slow Infrared (SIR) Interface can be used in conjunction with a standard UART to support two-way, wireless communications using infrared radiation. It provides a transmit encoder and receive decoder to support a physical link that conforms to the Infrared Data Association Serial Infrared Specification.

In the IXP42X product line and IXC1100 control plane processors, infrared mode is not supported. The reason for including this register in the address map and register description section is so software can ensure that this mode is never enabled.

Register Name:	ISR				
Hex Offset Address:	0xC800 0020		**Reset Hex Value:**	0x00000000	
Register Description:	Infrared Selection Register				
Access: Read/Write.					

31																					8	7	6	5	4	3	2	1	0
(Reserved)																						DMAE	UUE	NRZE	RTOIE	RIE	RLSE	TIE	RAVIE

Register		ISR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:5		(Reserved)
4	RXPL	Receive Data Polarity: 0 = SIR decoder takes positive pulses as zeros 1 = SIR decoder takes negative pulses as zeros
3	TXPL	Transmit Data Polarity: 0 = SIR encoder generates a positive pulse for a data bit of zero 1 = SIR encoder generates a negative pulse for a data bit of zero

Register		ISR (Sheet 2 of 2)
Bits	**Name**	**Description**
2	XMODE	**Transmit Pulse Width Select:** When XMODE is set to 0, clocking of the IRDA transmit and receive logic is done by the UART clock, which must be operating in the 16X mode. When XMODE is set to a 1, the operation of the receive decoder does not change. The transmit encoder, however, generates pulses 1.6 µs wide (which is three clock periods at frequency 1.8432 MHz) — instead of the normal 3/16th of a bit time wide. The shorter infrared pulse — generated with XMODE set to 1 — reduces the power consumed by the LEDs. At 2,400 bps, the LED would normally be activated for 78 µs for each 0 bit transmitted. With XMODE set, the LED would be activated for only 1.6 µs. This would reduce the power consumed by the LED by a factor of almost 48. 0 = Transmit pulse width is $3/16^{th}$ of a bit time wide 1 = Transmit pulse with is 1.6us
1	RCVEIR	**Receiver SIR Enable:** When the RCVEIR bit is set to logic 1, the IRDA decoder processes the signal from the rxd pin. If RCVEIR is cleared, all clocking to the IRDA decoder is blocked and the device's rxd pin is fed directly to the UART. 0 = Receiver input is in normal UART mode 1 = Receiver input is in infrared mode
0	XMITIR	**Transmitter SIR Enable:** When the XMITIR bit is set to logic 1, the IRDA encoder processes the normal txd output from the UART before it is fed to the device pin. If XMITIR is cleared, all clocking to the IRDA encoder is blocked and the UART's txd signal is connected directly to the device pin. 0 = Transmit output is in normal UART mode 1 = Transmit output is in infrared mode

10.5 Console UART

The Console Universal Asynchronous Receiver-Transmitter (UART) behaves exactly like the High-Speed UART. Speed will be programmable between 1,200 Baud to 231 KBaud.

10.5.1 Register Description

There are 13 registers to monitor and control the UART. The registers are all 32 bits in size, but only lower 8 bits have valid data. The 13 UART registers share nine address locations in the I/O address space.

Note that the state of the Divisor Latch Bit (DLAB) — the **most**-significant bit of the Serial Line Control Register — affects the selection of certain of the UART registers. The DLAB bit must be set high by the system software to access the Baud Rate Generator Divisor Latches.

Table 136. Console UART Registers Overview

Address	DLAB	R/W	Name	Description
0xC8001000	0	R	RBR	Receive Buffer Register
	0	W	THR	Transmit Holding Register
	1	R/W	DLL	Divisor Latch Low Register
0x C8001004	0	R/W	IER	Interrupt Enable Register
	1	R/W	DLH	Divisor Latch High Register
0x C8001008	0/1	R	IIR	Interrupt Identification Register
	0/1	W	FCR	FIFO Control Register
0x C800100C	0/1	R/W	LCR	Line Control Register
0x C8001010	0/1	R/W	MCR	Modem Control Register

Table 136. Console UART Registers Overview

Address	DLAB	R/W	Name	Description
0x C8001014	0/1	R	LSR	Line Status Register
0x C8001018	0/1	R	MSR	Modem Status Register
0x C800101C	0/1	R/W	SPR	Scratch Pad Register
0x C8001020	0	R/W	ISR	Slow Infrared Select Register

10.5.1.1 Receive Buffer Register

Register Name:	RBR				
Hex Offset Address:	0xC800 1000		Reset Hex Value:	0x00000000	
Register Description:	Receive Buffer Register				
Access: Read Only.					

31																						8	7							0
(Reserved)																							RBR							

Register		RBR
Bits	Name	Description
31:8		(Reserved)
7:0	RBR	In non-FIFO mode, this register holds the character received by the UART's Receive Shift Register. If fewer than 8 bits are received, the bits are right-justified and the leading bits are zeroed. In FIFO mode, this register latches the value of the data byte at the bottom of the Receive FIFO. The DLAB bit in the Line Control Register must be set to logic 0 to access this register.

10.5.1.2 Transmit Holding Register

Register Name:	THR				
Hex Offset Address:	0xC800 1000		Reset Hex Value:	0x00000000	
Register Description:	Transmit Holding Register				
Access: Write Only.					

31																						8	7							0
(Reserved)																							THR							

Register		THR
Bits	**Name**	**Description**
31:8		(Reserved)
7:0	THR	In Non-FIFO mode, This register holds the next data byte to be transmitted. When the Transmit Shift Register becomes empty, the contents of the Transmit Holding Register are loaded into the shift register and the transmit data request (TDRQ) bit in the Line Status Register is set to 1. In FIFO mode, writing to THR puts data to the top of the Transmit FIFO. The data at the bottom of the Transmit FIFO is loaded to the shift register when the shift register becomes empty. The DLAB bit in the Line Control Register must be set to logic 0 to access this register.

10.5.1.3 Divisor Latch Low Register

Register Name:	DLL				
Hex Offset Address:	0xC800 1000		**Reset Hex Value:**	0x00000002	
Register Description:	Divisor Latch Low Register				
Access: Read/Write.					

31																					**8**	**7**							**0**
(Reserved)																						DLL							

Register		DLL
Bits	**Name**	**Description**
31:8		(Reserved)
7:0	DLL	Lower byte of compare-value used by the baud-rate generator. The DLAB bit in the Line Control Register must be set to logic 1 to access this register.

10.5.1.4 Divisor Latch High Register

Register Name:	DLH				
Hex Offset Address:	0xC800 1004		**Reset Hex Value:**	0x00000000	
Register Description:	Divisor Latch High Register				
Access: Read/Write.					

31																					**8**	**7**							**0**
(Reserved)																						DLH							

Register		DLH
Bits	**Name**	**Description**
31:8		(Reserved)
7:0	DLH	Upper byte of compare-value used by the baud rate generator. The DLAB bit in the Line Control Register must be set to logic 1 to access this register.

10.5.1.5 Interrupt Enable Register

The DLAB bit in the Line Control Register must be set to logic 0 to access this register.

Register Name:	IER				
Hex Offset Address:	0xC800 1004		Reset Hex Value:	0x00000000	
Register Description:	Interrupt Enable Register				
Access: Read/Write.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			DMAE	UUE	NRZE	RTOIE	RIE	RLSE	TIE	RAVIE

Register		IER
Bits	**Name**	**Description**
31:8		(Reserved)
7	DMAE	**DMA Requests Enable**: 0 = DMA requests are disabled 1 = DMA requests are enabled Not Used On the Intel® IXP42X product line and IXC1100 control plane processors.
6	UUE	**UART Unit Enable**: 0 = the unit is disabled 1 = the unit is enabled
5	NRZE	**NRZ coding Enable**: 0 = NRZ coding disabled 1 = NRZ coding enabled Not Used On the Intel® IXP42X product line and IXC1100 control plane processors.
4	RTOIE	Receiver Time Out Interrupt Enable: 0 = Receiver data Time out interrupt disabled 1 = Receiver data Time out interrupt enabled
3	RIE	**Modem Interrupt Enable**: 0 = Modem Status interrupt disabled 1 = Modem Status interrupt enabled
2	RLSE	**Receiver Line Status Interrupt Enable**: 0 = Receiver Line Status interrupt disabled 1 = Receiver Line Status interrupt enabled The DLAB bit in the Line Control Register must be set to logic 0 to access this register.
1	TIE	**Transmit Data request Interrupt Enable**: 0 = Transmit FIFO Data Request interrupt disabled 1 = Transmit FIFO Data Request interrupt enabled
0	RAVIE	**Receive Data Available Interrupt Enable**: 0 = RAVIE interrupt disabled 1 = RAVIE interrupt enabled

10.5.1.6 Interrupt Identification Register

In order to minimize software overhead during data character transfers, the UART prioritizes interrupts into four levels and records these in the Interrupt Identification register. The Interrupt Identification Register (IIR) stores information indicating that a prioritized interrupt is pending and the source of that interrupt.

Table 137. Priority Levels of Interrupt Identification Register

Priority Level	Interrupt origin
1 (Highest)	**Receiver Line Status:** One or more error bits were set.
2	**Received Data is Available:** In FIFO mode, trigger level was reached. In non-FIFO mode, RBR has data.
2	**Receiver Time Out Occurred:** It happens in FIFO mode only, when there is data in the receive FIFO but no activity for a time period.
3	**Transmitter Requests Data:** In FIFO mode, the transmit FIFO is half or more than half empty. In non-FIFO mode, THR is read already.
4 (Lowest)	**Modem Status:** One or more of the modem input signals has changed state.

Register Name:	IIR				
Hex Offset Address:	0xC800 1008		Reset Hex Value:	0x00000001	
Register Description:	Interrupt Identification Register				
Access: Read Only.					

31																				8	7	6	5	4	3	2	1	0
(Reserved)																					FIFOES		(Rsvd)		TOD	IID		IP_N

Register		IIR
Bits	**Name**	**Description**
31:8		(Reserved)
7:6	FIFOES	**FIFO Mode Enable Status:** 00 = Non-FIFO mode is selected 01 = (Reserved) 10 = (Reserved) 11 = FIFO mode is selected (TRFIFOE = 1)
5:4		(Reserved)
3	TOD	**Time Out Detected:** 0 = No time out interrupt is pending 1 = Time out interrupt is pending. (FIFO mode only)
2:1	IID	**Interrupt Source Encoded:** 00 = Modem Status (CTS, DSR, RI, DCD modem signals changed state) 01 = Transmit FIFO requests data 10 = Received Data Available 11 = Receive error (Overrun, parity, framing, break, FIFO error)
0	IP_N	**Interrupt Pending:** 0 = Interrupt is pending. (Active low) 1 = No interrupt is pending

Table 138. UART Interrupt Identification Bit Level Definition

Interrupt ID Bits				Interrupt SET/RESET Function			
3	2	1	0	Priority	Type	Source	Reset Control
0	0	0	1	-	None	No Interrupt is pending	-
0	1	1	0	Highest	Receiver Line Status	Overrun Error, Parity Error, Framing Error, Break Interrupt.	Reading the Line Status Register.
0	1	0	0	Second Highest	Received Data Available.	Non-FIFO mode: Receive Buffer is full. FIFO mode: Trigger level was reached.	Non-FIFO mode: Reading the Receiver Buffer Register. FIFO mode: Reading bytes until Receiver FIFO drops below trigger level or setting RESETRF bit in FCR register.
1	1	0	0	Second Highest	Character Time-out indication.	FIFO Mode only: At least 1 character is in receiver FIFO and there was no activity for a time period.	Reading the Receiver FIFO or setting RESETRF Bit in FCR Register.
0	0	1	0	Third Highest	Transmit FIFO Data Request	Non-FIFO mode: Transmit Holding Register Empty FIFO mode: Transmit FIFO has half or less than half data.	Reading the IIR Register (if the source of the interrupt) or writing into the Transmit Holding Register. Reading the IIR Register (if the source of the interrupt) or writing to the Transmitter FIFO.
0	0	0	0	Fourth Highest	Modem Status	Clear to Send, Data Set Ready, Ring Indicator, Received Line Signal Detect	Reading the Modem Status Register.

10.5.1.7 FIFO Control Register

FCR is a write-only register that is located at the same address as the IIR. (IIR is a read-only register.) FCR enables/disables the transmitter/receiver FIFOs, clears the transmitter/receiver FIFOs, and sets the receiver FIFO trigger level.

Register Name:	FCR				
Hex Offset Address:	0xC800 1008		Reset Hex Value:	0x00000000	
Register Description:	FIFO Control Register				
Access: Write Only.					

31	...	8	7	6	5	...	3	2	1	0
(Reserved)			ITL		(Rsvd)			RESETTF	RESETRF	TRFIFOE

Register		FCR
Bits	**Name**	**Description**
31:8		(Reserved)
7:6	ITL	**Interrupt Trigger Level:** When the number of entries in the receive FIFO equals the interrupt trigger level programmed into this field and the Received Data Available Interrupt is enabled (via IER), an interrupt is generated and appropriate bits are set in the IIR. 00 = 1 byte or more in FIFO causes interrupt 01 = 8 bytes or more in FIFO causes interrupt 10 = 16 bytes or more in FIFO causes interrupt 11 = 32 bytes or more in FIFO causes interrupt
5:3	—	Reserved
2	RESETTF	**Reset Transmitter FIFO:** When RESETTF is logic 1, the transmitted FIFO count logic is set to 0, effectively clearing all the entries in the transmit FIFO. The TDRQ bit in LSR are set and IIR shows a transmitter requests data interrupt if the TIE bit in the IER register is set. The transmitter shift register is not cleared; it completes the current transmission. After the FIFO is cleared, RESETTF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The transmitter FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
1	RESETRF	**Reset Receiver FIFO:** When RESETRF is set to 1, the receive FIFO counter is reset to 0, effectively clearing all the entries in the receive FIFO. The DR bit in the LSR is reset to 0. All the error bits in the FIFO and the FIFOE bit in LSR are cleared. Any error bits, OE, PE, FE or BI that had been set in LSR are still set. The receiver shift register is not cleared. If IIR had been set to Received Data Available, it is cleared. After the receive FIFO is cleared, RESETRF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The receiver FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
0	TRFIFOE	**Transmit and Receive FIFO Enable:** TRFIFOE enables/disables the transmitter and receiver FIFO s. When TRFIFOE = 1, both FIFOs are enabled (FIFO Mode). When TRFIFOE = 0, the FIFO s are both disabled (non-FIFO Mode). Writing a 0 to this bit clears all bytes in both FIFO s. 0 = FIFOs are disabled 1 = FIFOs are enabled

10.5.1.8 Line Control Register

Register Name:	LCR																											
Hex Offset Address:	0xC800 100C			**Reset Hex Value:**	0x00000000																							
Register Description:	Line Control Register																											
Access: Read-Write.																												
31																						**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																							DLAB	SB	STKYP	EPS	PEN	STB	WLS	

Register		LCR
Bits	**Name**	**Description**
31:8		(Reserved)
7	DLAB	**Divisor Latch Access Bit:** This bit must be set to logic 1 to access the Divisor Latches of the Baud Rate Generator during a READ or WRITE operation. It must be set low (logic 0) to access the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register.
6	SB	**Set break:** This bit causes a break condition to be transmitted to the receiving UART. When SB is set to logic 1, the serial output (TXD) is forced to the spacing (logic 0) state and remains there until SB is set to logic 0. This bit acts only on the TXD pin. 0 = No effect on TXD output 1 = Forces TXD output to 0 (space)
5	STKYP	**Sticky Parity:** This bit is the “sticky-parity” bit, which can be used in multiprocessor communications. When PEN and STKYP are logic 1, the bit that is transmitted in the parity bit location (the bit just before the stop bit) is the complement of the EPS bit. If EPS is 0, then the bit at the parity bit location will be transmitted as a 1. In the receiver, if STKYP and PEN are logic 1, the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. If the values being compared are not equal, the receiver sets the Parity Error Bit in LSR and causes an error interrupt if line status interrupts were enabled. If EPS is 0, the receiver expects the bit received at the parity bit location to be 1. If it is not, then the parity error bit is set. By forcing the bit value at the parity bit location — rather than calculating a parity value — a system with a master transmitter and multiple receivers can identify some transmitted characters as receiver addresses and the rest of the characters as data. If PEN = 0, STKYP is ignored. 0 = No effect on parity bit 1 = Forces parity bit to be opposite of EPS bit value
4	EPS	Even parity Select: 0 = Sends or checks for odd parity 1 = Sends or checks for even parity
3	PEN	**Parity Enable:** This is the parity enable bit. When PEN is logic 1, a parity character is generated (transmit data) or checked (receive data) between the last data word bit and stop bit of the serial data. 0 = No parity function 1 = Allows parity generation and checking
2	STB	**Stop Bits:** This bit specifies the number of stop bits transmitted and received in each serial character. If STB is a logic 0, one stop bit is generated in the transmitted data. If STB is a logic 1 when a 5-bit word length is selected via bits 0 and 1, then 1 and one half stop bits are generated. If STB is a logic 1 when either a 6, 7, or 8-bit word is selected, then two stop bits are generated. The receiver checks the first stop bit only, regardless of the number of stop bits selected. 0 = One stop bit 1 = Two stop bits, except for 5-bit character, then 1.5 bits
1:0	WLS	**Word-Length Select:** The Word-Length Select bits specify the number of data bits in each transmitted or received serial character. 00 = 5-bit character (default) 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character

10.5.1.9 Modem Control Register

Register Name:	MCR				
Hex Offset Address:	0xC800 1010		Reset Hex Value:	0x00000000	
Register Description:	Modem Control Register				
Access: See below.					

31	...	5	4	3	2	1	0
(Reserved)			LOOP	OUT2	OUT1	RTS	DTR

Register		MCR
Bits	**Name**	**Description**
31:5		(Reserved)
4	LOOP	**Loop back test mode:** This bit provides a local Loop back feature for diagnostic testing of the UART. When LOOP is set to logic 1, the following will occur: The transmitter serial output is set to a logic 1 state. The receiver serial input is disconnected from the pin. The output of the Transmitter Shift register is “looped back” into the receiver shift register input. The four modem-control inputs (CTS#, DSR#, DCD#, and RI#) are disconnected from the pins and the modem control output pin RTS_N is forced to its inactive state. ***Note:*** Coming out of the loop-back test mode may result in unpredictable activation of the delta bits (bits 3:0) in the Modem Status Register (MSR). It is recommended that MSR is read once to clear the delta bits in the MSR. The lower four bits of the Modem Control Register are connected to the upper four Modem Status Register bits: • DTR = 1 forces DSR to a 1 • RTS = 1 forces CTS to a 1 • OUT1 = 1 forces RI to a 1 • OUT2= 1 forces DCD to a 1 0 = Normal UART operation 1 = Test mode UART operation
3	OUT2	Interrupt Mask: The OUT2 bit is used to mask the UART's interrupt output to the Interrupt Controller unit. OUT2 = 0 UART interrupt masked OUT2 = 1 UART interrupt not masked When LOOP=1, this bit is not used as interrupt mask. The interrupt goes to the processor without mask in loop-back mode.
2	OUT1	**Test bit:** This bit is used only in loop-back test mode. See (LOOP) Above.
1	RTS	**Request to Send:** This bit controls the Request to Send (RTS_N) output pin. 0 = RTS_N pin is 1 1 = RTS_N pin is 0
0	DTR	Data Terminal Ready: 0 = DTR# pin is 1 1 = DTR# pin is 0

10.5.1.10 Line Status Register

Register Name:	LSR				
Hex Offset Address:	0xC800 1014		Reset Hex Value:	0x00000060	
Register Description:	Line Status Register				
Access: Read Only.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			FIFOE	TEMT	TDRQ	BI	FE	PE	OE	DR

Register		LSR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		(Reserved)
7	FIFOE	**FIFO Error Status:** In non-FIFO mode, this bit is 0. In FIFO Mode, FIFOE is set to 1 when there is at least a parity error, framing error, or break indication for any of the characters in the FIFO. Note that a processor read to the Line Status Register does not reset this bit. FIFOE is reset when all error bytes have been read from the FIFO. 0 = Non-FIFO mode or no errors in receiver FIFO 1 = At least one character in receiver FIFO has errors
6	TEMT	**Transmitter Empty:** TEMT is set to a logic 1 when the Transmit-Holding Register and the Transmitter Shift Register are both empty. It is reset to logic 0 when either the Transmit Holding register or the transmitter shift register contains a data character. In FIFO mode, TEMT is set to 1 when the transmitter FIFO and the Transmit Shift register are both empty.
5	TDRQ	**Transmit Data Request:** TDRQ indicates that the UART is ready to accept a new character for transmission. In Non-FIFO mode, The TDRQ bit is set to logic 1 when a character is transferred from the Transmit Holding register into the Transmit Shift register. The bit is reset to logic 0 concurrently with the loading of the Transmit Holding register by the processor. In FIFO mode, TDRQ is set to 1 when half of the characters in the FIFO have been loaded into the Shift register or the RESETTF bit in FCR has been set to 1. It is cleared when the FIFO has more than half data. If more than 64 characters are loaded into the FIFO, the excess characters are lost.
4	BI	**Break Interrupt:** BI is set to a logic 1 when the received data input is held in the spacing (logic 0) state for longer than a full word transmission time (that is, the total time of Start bit + data bits + parity bit + stop bits). The Break indicator is reset when the processor reads the Line Status Register. In FIFO mode, only one character (equal to 00H) is loaded into the FIFO regardless of the length of the break condition. BI shows the break condition for the character at the bottom of the FIFO, not the most recently received character.
3	FE	**Framing Error:** FE indicates that the received character did not have a valid stop bit. FE is set to a logic 1 when the bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). The FE indicator is reset when the processor reads the Line Status Register. The UART will resynchronize after a framing error. To do this, it assumes that the framing error was due to the next start bit, so it samples this “start” bit twice and then takes in the “data.” In FIFO mode, FE shows a framing error for the character at the bottom of the FIFO, not for the most recently received character.

Register		LSR (Sheet 2 of 2)
Bits	**Name**	**Description**
2	PE	**Parity Error:** PE indicates that the received data character does not have the correct even or odd parity, as selected by the even parity select bit. The PE is set to logic 1 upon detection of a parity error and is reset to logic 0 when the processor reads the Line Status register. In FIFO mode, PE shows a parity error for the character at the bottom of the FIFO, not the most recently received character.
1	OE	**Overrun Error:** In non-FIFO mode, OE indicates that the processor did not read data the receiver buffer register before the next character was received. The new character is lost. In FIFO mode, OE indicates that all 64 bytes of the FIFO are full and the most recently received byte has been discarded. The OE indicator is set to logic 1 upon detection of an overrun condition and reset when the processor reads the Line Status register.
0	DR	**Data Ready:** Bit 0 is set to logic 1 when a complete incoming character has been received and transferred into the receiver buffer register or the FIFO. In non-FIFO mode, DR is reset to 0 when the receive buffer is read. In FIFO mode, DR is reset to a logic 0 if the FIFO is empty (last character has been read from RBR) or the RESETRF bit is set in the FCR. 0 = No data has been received 1 = Data is available in RBR or the FIFO

10.5.1.11 Modem Status Register

This register provides the processor with the current state of the control lines from the modem or data set (or a peripheral device emulating a modem). In addition to this current state information, four bits of the Modem Status Register provide change information.

These bits — 3:0 — are set to a logic 1 when a control input from the modem changes state. They are reset to a logic 0 when the processor writes ones to the bits of the Modem Status Register.

Note: When bits 0, 1, 2, or 3 are set to logic 1, a Modem Status Interrupt is generated — if bit 3 of the Interrupt Enable Register is set.

Register Name:	MSR			
Hex Offset Address:	0xC800 1018	**Reset Hex Value:**	0x00000000	
Register Description:	Modem Status Register			
Access: Read Only.				

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																								DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS

Register		MSR
Bits	**Name**	**Description**
31:8		(Reserved)
7	DCD	**Data Carrier Detect:** This bit is the complement of the Data Carrier Detect (DCD#) input. This bit is equivalent to bit OUT2 of the Modem Control Register if LOOP in the MCR is set to 1. 0 = DCD# pin is 1 1 = DCD# pin is 0
6	RI	**Ring Indicator:** This bit is the complement of the ring Indicator (RI#) input. This bit is equivalent to bit OUT1 of the Modem Control register if LOOP in the MCR is set to 1. 0 = RI# pin is 1 1 = RI# pin is 0
5	DSR	**Data Set Ready:** This bit is the complement of the Data Set Ready (DSR#) input. This bit is equivalent to bit DTR of the Modem Control Register if LOOP in the MCR is set to 1. 0 = DSR# pin is 1 1 = DSR# pin is 0
4	CTS	**Clear-to-Send:** This bit is the complement of the Clear-to-Send (CTS#) input. This bit is equivalent to bit RTS of the Modem Control Register if LOOP in the MCR is set to 1. 0 = CTS# pin is 1 1 = CTS# pin is 0
3	DDCD	Delta Data Carrier Detect: 0 = No change in DCD# pin since last read of MSR 1 = DCD# pin has changed state
2	TERI	Trailing-Edge Ring Indicator: 0 = RI# pin has not changed from 0 to 1 since last read of MSR 1 = RI# pin has changed from 0 to 1
1	DDSR	Delta Data Set Ready: 0 = No change in DSR# pin since last read of MSR 1 = DSR# pin has changed state
0	DCTS	Delta Clear-to-Send: 0 = No change in CTS# pin since last read of MSR 1 = CTS# pin has changed state

10.5.1.12 Scratch-Pad Register

This read/write register has no effect on the UART, it is intended as a scratch-pad register for use by the programmer.

Register Name:	SPR				
Hex Offset Address:	0xC800 101C		**Reset Hex Value:**	0x00000000	
Register Description:	Scratch Pad Register				
Access: Read/Write.					

31																					**8**	**7**							**0**
(Reserved)																						SCR							

Register		SPR
Bits	**Name**	**Description**
31:8		(Reserved)
7	SCR	No effect on UART functionality

10.5.1.13 Infrared Selection Register

The Slow Infrared (SIR) Interface can be used — in conjunction with a standard UART — to support two-way, wireless communications using infrared radiation. The register provides a transmit encoder and receive decoder to support a physical link which conforms to the Infrared Data Association Serial Infrared Specification.

For this application, however, the infrared mode will not be used. This register is included in the address map and register description section so software can ensure that this mode is never enabled.

Register Name:	ISR				
Hex Offset Address:	0xC800 1020		**Reset Hex Value:**	0x00000000	
Register Description:	Infrared Selection Register				
Access: Read/Write.					

31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																									RXPL	TXPL	XMODE	RCVEIR	XMITIR

Register		ISR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:5		(Reserved)
4	RXPL	Receive Data Polarity: 0 = SIR decoder takes positive pulses as zeros 1 = SIR decoder takes negative pulses as zeros
3	TXPL	Transmit Data Polarity: 0 = SIR encoder generates a positive pulse for a data bit of zero 1 = SIR encoder generates a negative pulse for a data bit of zero

Register		ISR (Sheet 2 of 2)
Bits	**Name**	**Description**
2	XMODE	**Transmit Pulse Width Select:** When XMODE is set to 0, clocking of the IRDA transmit and receive logic is done by the UART clock, which must be operating in the 16X mode. When XMODE is set to a 1, the operation of the receive decoder does not change, but the transmit encoder generates pulses that are 1.6 µs wide (which is three clock periods at the frequency of 1.8432 MHz) — rather than the normal pulses of 3/16th of a bit time wide. The shorter infrared pulse generated with XMODE set to 1, reduces the power consumed by the LEDs. At 2,400 bps, the LED would normally be activated for 78 µs for each 0 bit transmitted. With XMODE set, the LED would be activated for only 1.6 µs. This would reduce the power consumed by the LED by a factor of almost 48. 0 = Transmit pulse width is 3/16th of a bit time wide 1 = Transmit pulse with is 1.6 µs
1	RCVEIR	**Receiver SIR Enable:** When the RCVEIR bit is set to logic 1, the IRDA decoder processes the signal from the rxd pin. If RCVEIR is cleared, then all clocking to the IRDA decoder is blocked and the device's rxd pin is fed directly to the UART. 0 = Receiver input is in normal UART mode 1 = Receiver input is in infrared mode
0	XMITIR	**Transmitter SIR Enable:** When the XMITIR bit is set to logic 1, the IRDA encoder processes the normal txd output from the UART before it is fed to the device pin. If XMITIR is cleared, all clocking to the IRDA encoder is blocked and the UART's txd signal is connected directly to the device pin. 0 = Transmit output is in normal UART mode 1 = Transmit output is in infrared mode

§ §

11.0 Internal Bus Performance Monitoring Unit (IBPMU)

The Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor' Internal Bus Performance Monitoring Unit (IBPMU) may be used to gather statistics about transactions occurring on North AHB, South AHB, and the SDRAM Controller. These statistics can be used to optimize the software running on the Intel XScale® Processor which, in turn, will improve overall system performance.

There are two types of events that can be measured: occurrence and duration.

An occurrence event is an action that occurs periodically and is measured by counting the number of times that the event occurred. An example of an occurrence event is counting the number of times the Intel XScale processor requests the AHB bus.

A duration event is an action that measures the amount of time it takes to complete a requested action. For example, a duration event could be keeping track of the amount of time the Intel XScale® Processor spends requesting the South AHB before it is granted mastership to the South AHB.

The occurrence event and the duration event, used in the examples above, could be used to determine the average amount of delay the Intel XScale processor is experiencing in acquiring the South AHB.

The IBPMU statistics are gathered in seven 27-bit, programmable event counters (PEC). The counter size was chosen to allow the measurement of a duration event to be approximately 1 second for 133-MHz accesses. The 133-MHz accesses originate from the speed of the North AHB, South AHB, and the SDRAM Controller.

Therefore, when a duration event is monitored and the programmable event counter rolls over, an interrupt is generated. The programmable event counter will have measured 1 second in time. Whenever a programmable event counter rolls over, the counter will set a bit in a local interrupt status register and start counting from zero. All the bits from the status register will be combined to send a single interrupt signal that is sent to the Interrupt Controller.

11.1 Initializing the IBPMU

The seven 27-bit programmable event counters can monitor the events — shown in Table 140 and Table 141 — by setting the values in the 32-bit Event Select Register (ESR). The Event Select Register is broken up into seven 3-bit registers that indicate the event type to monitor, occurrence or duration, and the exact event.

The three bits associated with Programmable Event Counter 1 are located in bits 20 through 22 of the Event Select Register. The three bits associated with Programmable Event Counter 7 are located in bits 2 through 4 of the Event Select Register. All other programmable event counter configurations are in 3-bit increments, going down from bit 19 down to bit 2. (For example, PEC2 is configured by bits 19 through 17 of the event select register and PEC3 is configured by bits 16 through 14 of the event select register.)

The most-significant bit of each three-bit programmable event counter configuration register will select if the programmable event counter is to read an occurrence event or a duration event.

A value of logic 1 — in the most-significant bit of the programmable event counter's configuration register (for example, bit 22 for PEC1) — will indicate that the programmable event counter is to capture a predefined duration event when in South AHB or North AHB monitoring mode or a page miss when in SDRAM monitoring mode. A value of logic 0 — in the most-significant bit of the programmable event counter configuration register — will indicate that the programmable event counter is to capture a predefined occurrence event when in South AHB or North AHB monitoring mode or a page hit when in SDRAM monitoring mode.

The least-significant two bits of each of the programmable event counter's event select register will indicate the specific event to capture. For a list of possible captured events, see Section 11.3.1, "Event Select Register" on page 378.

In addition to the seven 3-bit registers contained within the event select register, there is a 2-bit register — that is the least significant two bits of the Event Select Register (bits 1 and 0) — that contains the mode select of the programmable event counters. The mode select will be used to select the mode of operation that the IBPMU is executing in.

The mode that is selected will be valid across all of the programmable event counters. That means the South AHB, North AHB, and SDRAM interfaces must be measured independently. The four modes of operation are:

- Halt
- North AHB Monitoring
- South AHB Monitoring
- SDRAM Monitoring

The mode selection is shown in Table 139:

Table 139. IBPMU Mode Selection Operation

Mode Bits (Event Select Register Bits 1 and 0)	Mode of Operation
00	Halt
01	South AHB
10	North AHB
11	SDRAM

The programmable event counters will be set to 0 after exiting from the halt mode and going into any other mode. Entering halt mode from any other mode causes all programmable event counters to halt at the same time, allowing a snapshot in time to be read across all of the programmable event counters. All programmable event counters will be reset and started whenever a new value is written to the ESR and the mode that is written is not HALT. The Event Select register will be set to all 0s, after a reset signal is received.

11.2 Using the IBPMU

The IBPMU is designed to allow efficient, real-time collection of statistical information regarding internal bus and SDRAM acquisitions. This enables a designer to tune the execution of the software running on the Intel XScale processor to achieve maximum performance.

The programmable event counters (PEC) and the previous master and slave register (PMSR) can be used for easy collection of the gathered statistical information.

The programmable event counters can be read periodically or when an interrupt occurs to the Intel® IXP42X product line and IXC1100 control plane processors' Interrupt Controller. An interrupt will occur when any of the programmable event counters roll over their maximum count value.

Writing logic 1 to the PMU status register bit — associated with the overflow condition that caused the interrupt to occur — clears the interrupt. The information obtained by reading the programmable event counters is entirely dependent on how the IBPMU is initialized in the event select register.

The IBPMU can be configured to monitor occurrence events. An occurrence event causes the programmable event counter to increase by one, each time the event occurs. Table 140 lists the various occurrence events that can be monitored by the IXP42X product line and IXC1100 control plane processors.

Table 140. Occurrence Events

Observed Interface	Monitored Event
AHB_South	Number of grants to the Intel XScale processor, AHB_Bridge, APB bridge and PCI_Controller.
	Number of transactions initiated by the Intel XScale processor, AHB_Bridge, APB bridge and PCI_Controller
	Number of retries Signaled by the PCI_Controller and Expansion_bus to the initiators.
	Number of split transfers claimed by the PCI_Controller and Expansion_Bus to the initiators
AHB_North	Number of grants to NPE A, NPE B and NPE C
	Number of transactions initiated by NPE A, NPE B and NPE C.
	Number of retries signaled by the AHB_Bridge to the initiators.
	Number of split transfers claimed by the AHB_Bridge to the initiators
SDRAM	Number of SDR0 misses
	Number of SDR1 misses
	Number of SDR2 misses
	Number of SDR3 misses
	Number of SDR4 misses
	Number of SDR5 misses
	Number of SDR6 misses
	Number of SDR7 misses
	Number of SDR0 hits
	Number of SDR1 hits
	Number of SDR2 hits
	Number of SDR3 hits
	Number of SDR4 hits
	Number of SDR5 hits
	Number of SDR6 hits
	Number of SDR7 hits

For a duration event, the programmable event counters will count the number of clocks during which a particular condition or set of conditions is valid. Duration measurements examples include:

- Bus Acquisition Latency — The elapsed time between when an AHB bus initiator issues a request for the AHB bus, to when the AHB bus initiator is granted the AHB bus.
- Split Transfer Latency — The elapsed time between when the bus initiator that issued a split request receives a split response from the target that completed the split transfer and the first DWORD of data read is received.
- Bus Ownership Time — The elapsed time that an initiator is in control of the bus.
- Initiator Bus Data Transfer Time — The elapsed time that an initiator takes to transfers data.
- Split Data Transfer Time — The elapsed time during which a split transfer target transfers data (data read) to a split transfer initiator. (Note that this metric is associated with the split transfer initiator and not the split transfer target.)

Table 141 lists the various duration events that may be monitored by the IXP42X product line and IXC1100 control plane processors.

Table 141. Duration Events

Observed interface	Monitored Event
AHB_South	Number of clocks the AHB bus is doing Data Writes
	Number of clocks the AHB bus is doing Data Reads
	Number of clocks the AHB bus is Idle
	Bus acquisition latency for AHB_South_Agents (Intel XScale processor, AHB_Bridge, APB bridge, PCI_Bus)
	Bus ownership metrics for AHB_South_Agents (Intel XScale processor, AHB_Bridge, APB bridge, PCI_Bus)
	Initiator data transfer time for AHB_Agents (Intel XScale processor, AHB_Bridge, APB bridge, PCI_Bus)
AHB_North	Number of clocks the AHB bus is doing Data Writes
	Number of clocks the AHB bus is doing Data Reads
	Number of clocks the AHB bus is Idle
	Bus acquisition latency for AHB_North_Agents (NPE A, NPE B, NPE C)
	Bus ownership metrics for AHB_North_Agents (NPE A, NPE B, NPE C)
	Initiator data transfer time for AHB_North_Agents (NPE A, NPE B, NPE C)
	Number of clocks the AHB_Bus is doing Data Reads during a split transfer claimed by an AHB_North_Bus slave: ABB

In addition to the programmable event counters, the IXP42X product line and IXC1100 control plane processors has the ability to determine the last initiator and target that controlled the North AHB and South AHB. The information can be obtained by reading the previous master and slave register (PMSR).

11.2.1 Monitored Events South AHB and North AHB

Table 142 describes the events that can be monitored on the South AHB and North AHB.

Table 142. North and South Modes Event Descriptions (Sheet 1 of 2)

Event	Bus Components	Type	Description
Bus_idle	North AHB South AHB	Duration	Increments the counter every AHB Bus idle cycle. An idle cycle occurs when there is no activity on the bus due to data being transferred and the bus is not in an overhead cycle. An overhead cycle is a cycle when an initiator owns the bus, however the initiator is unable to send data or the target is unable to receive data - hence no data is transferred.
Bus_Write	North AHB South AHB	Duration	Increments the counter on every AHB Bus Write data cycle. This enables calculation of data utilization of the Write data bus.
Bus_Read	North AHB South AHB	Duration	Increments the counter on every AHB Bus Read data cycle. This enables calculation of data utilization of the Write data bus.
MstrX_Req	North: • WAN/Voice NPE • Ethernet NPE A • Ethernet NPE B South: • Intel XScale processor • AHB/AHB Bridge • PCI Bus • AHB/APB Bridge	Duration	Counts the number of clocks spent by a master acquiring the AHB bus. The counter increments on every clock cycle after the master has requested use of the bus but has not actively driven the bus as an initiator. This is an event primitive, used in conjunction with another event primitive (number of grants granted) to calculate the average acquisition latency.
MstrX_Own	North: • WAN/Voice NPE • Ethernet NPE A • Ethernet NPE B South: • Intel XScale processor • AHB/AHB Bridge • PCI Bus • AHB/APB Bridge	Duration	Counts the duration for which the master is the initiator on the AHB bus. The counter increments on every clock cycle during which the master is the bus initiator.
MstrX_Write	North: • WAN/Voice NPE • Ethernet NPE A • Ethernet NPE B South: • Intel XScale processor • AHB/AHB Bridge • PCI Bus • AHB/APB Bridge	Duration	Increments counter every AHB bus write data cycle where the master is the initiator.
MstrX_Read	North: • WAN/Voice NPE • Ethernet NPE A • Ethernet NPE B South: • Intel XScale processor • AHB/AHB Bridge • PCI Bus • AHB/APB Bridge	Duration	Increments counter every AHB bus read data cycle where the master is the initiator.

Table 142. North and South Modes Event Descriptions (Sheet 2 of 2)

Event	Bus Components	Type	Description
MstrX_Grant	North: • WAN/Voice NPE • Ethernet NPE A • Ethernet NPE B South: • Intel XScale processor • AHB/AHB Bridge • PCI Bus • AHB/APB Bridge	Occur	Monitors the number of times the master is granted the bus. It increments the counter when the master is the bus initiator. The counter is incremented once for every new transaction. For multi-cycle transactions, the counter increments once on the first cycle.
MstrX_Xfer	North: • WAN/Voice NPE • Ethernet NPE A • Ethernet NPE B South: • Intel XScale processor • AHB/AHB Bridge • PCI Bus • AHB/APB Bridge	Occur	Increments the counter every time that the master initiates a transaction on the bus. This event primitive can be used in conjunction with another event primitive, MstrX_Retry, to calculate the effectiveness of transactions initiated on the AHB Bus by the master (i.e., percentage transactions initiated by the master that are not retried.).
MstrX_Retry	North: • WAN/Voice NPE • Ethernet NPE A • Ethernet NPE B South: • Intel XScale processor • AHB/AHB Bridge • PCI Bus • AHB/APB Bridge	Occur	Increments the counter every time that a transaction initiated by the master is responded to with a retry by the target.
SlaveX_Split	North: • AHB/AHB Bridge South: • PCI Bus • Expansion Bus	Occur	Counts the number of times that SlaveX claims a Split Transaction.

11.2.2 Monitored SDRAM Events

Selecting SDRAM mode enables the programmable event counters to monitor SDRAM hits and misses. All programmable event counters are clocked at 133 MHz frequency.

11.2.3 Cycle Count

In all but halt mode, Programmable Event Counter 7 can be configured to count 133-MHz bus-clock cycles. The ability to count the 133-MHz bus-clock cycles is referred to as “cycle count.”

The cycle count allows an accurate measurement of the duration of the test to be made. The cycle count is reset and started each time a value is written to the ESR and the mode is not halt. When HALT mode is selected the counter stops, preserving its last value. If the count overflows the PEC7, the overflow interrupt is set.

11.3 Register Descriptions

Table 143. Internal Bus PMU Register Overview

Address	R/W	Name	Description
0xC8002000	R/W	ESR	Event Select Register
0xC8002004	R/COW	PSR	PMU Status Register
0xC8002008	R	PEC1	Programmable Event Counter 1
0xC800200C	R	PEC2	Programmable Event Counter 2
0xC8002010	R	PEC3	Programmable Event Counter 3
0xC8002014	R	PEC4	Programmable Event Counter 4
0xC8002018	R	PEC5	Programmable Event Counter 5
0xC800201C	R	PEC6	Programmable Event Counter 6
0xC8002020	R	PEC7	Programmable Event Counter 7
0xC8002024	R	PMSR	Previous Master/Slave Register

11.3.1 Event Select Register

The Event Select Register (ESR) controls the specific mode of operation of performance monitoring. There are four general modes supported:

- Halt
- North
- South
- SDRAM

Each PEC counter also has three control bits. The most-significant bit indicates whether a duration or occurrence event is to be monitored. The remaining two bits select from four possible inputs for this configuration.

To change the monitored mode, it is necessary to write the entire ESR. The Programmable Event Counters are reset and started when a new value is written to the ESR and the mode is not HALT. Performance monitoring is disabled in HALT (the default) mode. HALT mode stops all counters and the PMSR from updating. Thus providing a stable result for reading by the processor.

Register Name:	ESR																															
Hex Offset Address:	0xC800 2000											Reset Hex Value:	0x00000000																			
Register Description:	Event Select Register																															
Access: Read/Write.																																
31								23	22		20	19		17	16		14	13		11	10		8	7			4	3		1	0		
(Reserved)									PEC1 ctrl			PEC2 ctrl			PEC3 ctrl			PEC4 ctrl			PEC5 ctrl			PEC6 ctrl				PEC7 ctrl			Mode

Register		ESR
Bits	**Name**	**Description**
31:23		(Reserved). Always zero
22:20	PEC1 ctrl	Selects Enable conditions for counter PEC1. 0xx = Occur / Hit 1xx = Duration / Miss ***Note:*** The lower two bits select one of four event or duration inputs depending on the mode and which type of event is selected.
19:17	PEC2 ctrl	Selects Enable conditions for counter PEC2. 0xx = Occur / Hit 1xx = Duration / Miss ***Note:*** The lower two bits select one of four event or duration inputs depending on the mode and which type of event is selected.
16:14	PEC3 ctrl	Selects Enable conditions for counter PEC3. 0xx = Occur / Hit 1xx = Duration / Miss ***Note:*** The lower two bits select one of four event or duration inputs depending on the mode and which type of event is selected.
13:11	PEC4 ctrl	Selects Enable conditions for counter PEC4. 0xx = Occur / Hit 1xx = Duration / Miss ***Note:*** The lower two bits select one of four event or duration inputs depending on the mode and which type of event is selected.
10:8	PEC5 ctrl	Selects Enable conditions for counter PEC5. 0xx = Occur / Hit 1xx = Duration / Miss ***Note:*** The lower two bits select one of four event or duration inputs depending on the Mode and which type of event is selected.
7:5	PEC6 ctrl	Selects Enable conditions for counter PEC6. 0xx = Occur / Hit 1xx = Duration / Miss ***Note:*** The lower two bits select one of four event or duration inputs depending on the mode and which type of event is selected.
4:2	PEC7 ctrl	Selects Enable conditions for counter PEC7. 0xx = Occur / Hit 1xx = Duration / Miss ***Note:*** The lower two bits select one of four event or duration inputs depending on the mode and which type of event is selected.
1:0	Mode	Value in this bit field determines the monitored interface on the Intel® IXP42X product line and IXC1100 control plane processors. 00 = Mode HaltPerformance Monitoring Disabled 01 = Mode SouthAHB South Bus events 10 = Mode NorthAHB North Bus events 11 = Mode SDRAMSDRAM transactions

Table 144. Possible Event Settings

Mode [1:0]	Occur / Duration	EventXSel [1:0]	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
South	Occur	00	XScale Grant	XScale Transfer	XScale Retry	PCI Split	ABB Grant	PCI Grant	APB Retry
	Occur	01	ABB Grant	ABB Transfer	ABB Retry	EXP Split	ABB Xfer	PCI Xfer	Cycle Count
	Occur	10	PCI Grant	PCI Transfer	PCI Retry	APB Grant	ABB Retry	PCI Retry	Cycle Count
	Occur	11	APB Grant	APB Transfer	APB Retry	APB Transfer	EXP Split	PCI Split	Cycle Count
	Duration	00	XScale Request	XScale Own	XScale Write	XScale Read	ABB Request	PCI Request	APB Request
	Duration	01	ABB Request	ABB Own	ABB Write	ABB Read	ABB Own	PCI Own	APB Own
	Duration	10	PCI Request	PCI Own	PCI Write	PCI Read	Bus Idle	Bus Write	Bus Read
	Duration	11	APB Request	APB Own	APB Write	APB Read			Cycle Count
North	Occur	00	NPE A Grant	NPE A Transfer	NPE A Retry	ABB Split	NPE B Grant	NPE C Grant	Cycle Count
	Occur	01	NPE B Grant	NPE B Transfer	NPE B Retry		NPE B Transfer	NPE C Transfer	Cycle Count
	Occur	10	NPE C Grant	NPE C Transfer	NPE C Retry		NPE B Retry	NPE C Retry	Cycle Count
	Occur	11							
	Duration	00	Bus Idle	Bus Write	Bus Read	NPE A Request	NPE B Request	NPE C Request	Cycle Count
	Duration	01	NPE A Request	NPE A Own	NPE A Write	NPE A Read	NPE B Own	NPE C Own	Cycle Count
	Duration	10	NPE B Request	NPE B Own	NPE B Write	NPE B Read	NPE B Write	NPE B Write	Cycle Count
	Duration	11	NPE C Request	NPE C Own	NPE C Write	NPE C Read	NPE B Read	NPE C Read	Cycle Count
SDRAM	Occur	00	SDR0 Hit	SDR1 Hit	SDR2 Hit	SDR3 Hit	SDR4 Hit	SDR5 Hit	Cycle Count
	Occur	01	SDR1 Hit	SDR2 Hit	SDR3 Hit	SDR4 Hit	SDR5 Hit	SDR6 Hit	Cycle Count
	Occur	10	SDR2 Hit	SDR3 Hit	SDR4 Hit	SDR5 Hit	SDR6 Hit	SDR7 Hit	Cycle Count
	Occur	11	SDR3 Hit	SDR4 Hit	SDR5 Hit	SDR6 Hit	SDR7 Hit	SDR0 Hit	Cycle Count
	Duration	00	SDR4 Miss	SDR5 Miss	SDR6 Miss	SDR7 Miss	SDR0 Miss	SDR1 Miss	Cycle Count
	Duration	01	SDR5 Miss	SDR6 Miss	SDR7 Miss	SDR0 Miss	SDR1 Miss	SDR2 Miss	Cycle Count
	Duration	10	SDR6 Miss	SDR7 Miss	SDR0 Miss	SDR1 Miss	SDR2 Miss	SDR3 Miss	Cycle Count
	Duration	11	SDR7 Miss	SDR0 Miss	SDR1 Miss	SDR2 Miss	SDR3 Miss	SDR4 Miss	Cycle Count

11.3.2 PMU Status Register (PSR)

The PSR allows access to the over flow flags from the PEC counters. These flags remain set until cleared by writing a 1 to the bit.

Register Name:	PSR				
Hex Offset Address:	0xC800 2004		Reset Hex Value:	0x00000000	
Register Description:	PMU Status Register				
Access: Read, Clear on write.					

31	...	7	6	5	4	3	2	1	0
(Reserved)			OFL7		OFL6	OFL5	OFL4	OFL3	OFL2

Register		PSR
Bits	Name	Description
31:6		(Reserved). Always zero
6	OFL7	1 = PEC7 has overflowed
5	OFL6	1 = PEC6 has overflowed
4	OFL5	1 = PEC5 has overflowed
3	OFL4	1 = PEC4 has overflowed
2	OFL3	1 = PEC3 has overflowed
1	OFL2	1 = PEC2 has overflowed
0	OFL1	1 = PEC1 has overflowed

11.3.3 Programmable Event Counters (PEC1)

There are seven programmable event counters (PEC1 – PEC7). These counters are 27 bits wide and are read-only.

The value in any register is incremented based on the current programmed ESR value. If a new value is written to the ESR, all PEC counter registers are reset to 0. The counters are halted when HALT mode is selected.

Register Name:	PEC1				
Hex Offset Address:	0xC800 2008		Reset Hex Value:	0x00000000	
Register Description:	Programmable Event Counter				
Access: Read.					

31	...	27	26	...	0
(Reserved)			PEC1		

Register		PEC1
Bits	Name	Description
31:27		(Reserved). Always 0
26:0	PEC1	This is a 27-bit, read-only counter register.

11.3.4 Programmable Event Counters (PEC2)

Register Name:	PEC2				
Hex Offset Address:	0xC800 200C		Reset Hex Value:	0x00000000	
Register Description:	Programmable Event Counter				
Access: Read.					

31	27	26	0
(Reserved)		PEC2	

Register		PEC2
Bits	Name	Description
31:27		(Reserved). Always 0
26:0	PEC2	This is a 27-bit, read-only counter register.

11.3.5 Programmable Event Counters (PEC3)

Register Name:	PEC3				
Hex Offset Address:	0xC800 2010		Reset Hex Value:	0x00000000	
Register Description:	Programmable Event Counter				
Access: Read.					

31	27	26	0
(Reserved)		PEC3	

Register		PEC3
Bits	Name	Description
31:27		(Reserved). Always 0
26:0	PEC3	This is a 27-bit, read-only counter register.

11.3.6 Programmable Event Counters (PEC4)

Register Name:	PEC4				
Hex Offset Address:	0xC800 2014		Reset Hex Value:	0x00000000	
Register Description:	Programmable Event Counter				
Access: Read.					

31	27	26	0
(Reserved)		PEC4	

Register		PEC4
Bits	**Name**	**Description**
31:27		(Reserved). Always 0
26:0	PEC4	This is a 27-bit, read-only counter register.

11.3.7 Programmable Event Counters (PEC5)

Register Name:	PEC5				
Hex Offset Address:	0xC800 2018		**Reset Hex Value:**	0x00000000	
Register Description:	Programmable Event Counter				
Access: Read.					

31				**27**	**26**																							**0**
(Reserved)					PEC5																							

Register		PEC5
Bits	**Name**	**Description**
31:27		(Reserved). Always 0
26:0	PEC5	This is a 27-bit, read-only counter register.

11.3.8 Programmable Event Counters (PEC6)

Register Name:	PEC6				
Hex Offset Address:	0xC800 201C		**Reset Hex Value:**	0x00000000	
Register Description:	Programmable Event Counter				
Access: Read.					

31				**27**	**26**																							**0**
(Reserved)					PEC6																							

Register		PEC6
Bits	**Name**	**Description**
31:27		(Reserved). Always 0.
26:0	PEC6	This is a 27-bit, read-only counter register.

11.3.9 Programmable Event Counters (PEC7)

Register Name:	PEC7				
Hex Offset Address:	0xC800 2020		Reset Hex Value:	0x00000000	
Register Description:	Programmable Event Counter				
Access: Read.					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)					PEC7																										

Register		PEC7
Bits	Name	Description
31:27		(Reserved). Always 0.
26:0	PEC7	This is a 27-bit, read-only counter register.

11.3.10 Previous Master/Slave Register (PSMR)

Register Name:	PSMR				
Hex Offset Address:	0xC800 2024		Reset Hex Value:	0x00000000	
Register Description:	Previous Master/Slave Register				
Access: Read.					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)														PSS						PSN				PMS				PMN			

Register		PSMR (Sheet 1 of 2)
Bits	Name	Description
31:18		(Reserved)
17:12	PSS	Indicates which of the Slaves on South AHB was previously accessed by South AHB Masters: 000001 = Expansion Bus 000010 = SDRAM controller 000100 = PCI 001000 = Queue Manager 010000 = AHB-APB Bridge 100000 = (Reserved)

Register		PSMR (Sheet 2 of 2)
Bits	**Name**	**Description**
11:8	PSN	Indicates which of the Slaves on North AHB was previously accessed North AHB Masters: 0001 = SDRAM controller 0010 = AHB-AHB Bridge 0100 = (Reserved) 1000 = (Reserved)
7:4	PMS	Indicates which of the Masters on South AHB was previously accessing the South AHB: 0001 = Intel XScale processor 0010 = AHB-AHB Bridge 0100 = PCI 1000 = APB
3:0	PMN	Indicates which of the Master on North AHB was previously accessing the North AHB: 0001 = WAN/Voice NPE = NPE-A (IXP400 software Definition) 0010 = Ethernet NPE A = NPE-B (IXP400 software Definition) 0100 = Ethernet NPE B = NPE-C (IXP400 software Definition) 1000 = (Reserved)

12.0 General Purpose Input/Output (GPIO)

The Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor provide 16 general-purpose input/output (GPIO) pins for use in capturing and generating application specific input and output signals. Each GPIO can be programmed as either an input or an output.

GPIO Pin 14 can be configured similar to GPIO pin 13 or as a clock output. The output-clock configuration can be set at various speeds, up to 33.33 MHz, with various duty cycles. GPIO Pin 14 is configured as an input, upon reset.

GPIO Pin 15 can be configured similar to GPIO pin 13 or as a clock output. The output-clock configuration can be set at various speeds, up to 33.33 MHz, with various duty cycles. GPIO Pin 15 is configured as a clock output, upon reset. GPIO Pin 15 can be used to clock the expansion interface, after reset. Each GPIO pin will be capable of sinking or sourcing 16 mA of current that may be used to drive external LEDs.

When used as an interrupt source, each pin can detect interrupts as active high, active low, rising edge, falling edge, or transitional. There are eight distinct register functions used in the GPIO module.

The GPIO is controlled through six registers and two other registers provide GPIO status. Each register can be read through the APB interface and all registers except general-purpose input register (GPINR) can be written through the APB interface.

12.1 Using GPIO as Inputs/Outputs

The GPIO on the Intel® IXP42X product line and IXC1100 control plane processors can be configured to be used as general-purpose inputs or general-purpose outputs. Three 16-bit registers are used in order to configure, initialize, and use the general purpose I/O. These registers are:

- General-Purpose Data Output Register (GPOUTR)
- General-Purpose Output Enable Register (GPOER)
- General-Purpose Input Status Register (GPINR)

The General-Purpose Output Enable Register is used to configure the GPIO pins as an input or an output. There is a one-for-one relationship between the register bit mapping and the GPIO. For example, bit 0 of the general-purpose Output Enable Register corresponds to GPIO 0 and bit 1 of the General-Purpose Output Enable Register corresponds to GPIO 1.

When a bit of the General-Purpose Enable Register contains logic 0, the corresponding GPIO will be configured as an output. A logic 1 in the same bit of the General-Purpose Enable Register will cause the corresponding GPIO to be configured as an input.

For example, the General-Purpose Output Enable Register contains a hexadecimal value of 0x00000500. GPIO 8 and GPIO 10 will be configured as inputs and all other GPIO will be configured as outputs. The GPIO that are configured as outputs — by the values contained in the General-Purpose Enable Register — will be driven by the values contained in the General-Purpose Data Output Register.

The General-Purpose Data Output Register is a 16-bit register with a one-for-one correspondence between the 16 bits of the General-Purpose Data Output Register and the 16-bit GPIO. When logic 1 is written to a bit in the General-Purpose Data Output Register — and the corresponding bit in the General-Purpose Enable Register is set to logic 0 — logic 1 will be replicated to the corresponding GPIO. When logic 0 is written to the same bit in the General-Purpose Data Output Register — and the corresponding bit in the General-Purpose Enable Register is still set to a logic 0 — logic 0 will be replicated to the corresponding GPIO.

For example, the General-Purpose Enable Register is set to hexadecimal 0xFFFFFAFF and the General-Purpose Data Output Register is set to hexadecimal 0x00000401. GPIO8 and GPIO 10 will be configured as outputs, GPIO8 will drive logic 0, and GPIO10 will drive logic 1.

Notice that bit 0 of the General-Purpose Data Output Register is set to logic 1. However, GPIO0 is configured as an input so the logic 1 in the General-Purpose Data Output Register is not replicated to GPIO0.

Reading of the current status of the GPIO can be obtained by reading the General-Purpose Input Status Register. The General-Purpose Input Status Register is a 16-bit register with a one-for-one correspondence between the 16 bits of the General-Purpose Input Status Register and the 16-bit GPIO.

When reading the General-Purpose Input Status Register, the current logic value of the GPIO will be reflected in the corresponding bits of the General-Purpose Input Status Register. The values will be replicated regardless of the configuration of the GPIO as defined by the General-Purpose Enable Register.

If a GPIO is configured as an output, the value driven to the output will be read when reading the corresponding bits of the General-Purpose Input Status Register. For example, the General-Purpose Enable Register is set to hexadecimal 0xFFFFFAFF and the General-Purpose Data Output Register is set to hexadecimal 0x00000401 and the GPIO pins have the following signals being supplied as inputs or driven as outputs, hexadecimal 0xAE37 with GPIO[15:12] = A and GPIO[3:0] = 7. GPIO8 and GPIO10 are configured as outputs as defined by the General-Purpose Enable Register. All other GPIO pins are configured as inputs. When the General-Purpose Input Status Register is read the value of hexadecimal 0x0000AE37 will be returned.

12.2 Using GPIO as Interrupt Inputs

The IXP42X product line and IXC1100 control plane processors GPIO may be programmed to accept interrupts from external sources. Some of these sources may be Ethernet or ADSL Physical interfaces. The type of interrupt can be one of five types:

- Active High – GPIO is high for more than five 66-MHz clock pulses
- Active Low – GPIO is low for more than five 66-MHz clock pulses
- Rising Edge – GPIO goes from an active low to an active high for more than five 66-MHz clock pulses
- Falling Edge – GPIO goes from an active high to an active low for more than five 66-MHz clock pulses
- Transitional (rising edge or falling edge) – Active high to active low or active low to active high for more than five 66-MHz clock pulses

The GPIO interrupt can be detected by setting the two general-purpose interrupt type registers (GPIT1R and GPIT2R). The general-purpose interrupt type registers are two 24-bit registers. These 24-bit interrupt type registers are broken up into 3-bit registers.

Each 3-bit register represents one GPIO interrupt configuration:

- GPIT1R represents GPIO 0 through GPIO 7
- GPIT2R represents GPIO 8 through GPIO 15
- General-Purpose Interrupt Type Register 1 bits 0 through 2 represent GPIO0
- General-Purpose Interrupt Type Register 1 bits 3 through 5 represent GPIO2, ...
- General-Purpose Interrupt Type Register 1 bits 21 through 23 represent GPIO7
- General-Purpose Interrupt Type Register 2 bits 0 through 2 represent GPIO8
- General-Purpose Interrupt Type Register 2 bits 3 through 5 represent GPIO9, ...
- General-Purpose Interrupt Type Register 2 bits 21 through 23 represent GPIO15.

The possible settings for each of the 3-bit registers are shown in Table 145.

Table 145. GPIO Interrupt Selections

Register Name	GPIO	Register Bit Definition			Interrupt Description	Interrupt Type
GPIT2R	GPIO15	Bit 23	Bit 22	Bit 21	Not used.	
	GPIO14	Bit 20	Bit 19	Bit 18	Not used.	
	GPIO13	Bit 17	Bit 16	Bit 15	Not used.	
	GPIO12	Bit 14	Bit 13	Bit 12	Reset to Active High	
	GPIO11	Bit 11	Bit 10	Bit 9	Reset to Active High	
	GPIO10	Bit 8	Bit 7	Bit 6	Reset to Active High	
	GPIO9	Bit 5	Bit 4	Bit 3	Reset to Active High	
	GPIO8	Bit 2	Bit 1	Bit 0	Reset to Active High	
GPIT1R	GPIO7	Bit 23	Bit 22	Bit 21	Reset to Active High	
	GPIO6	Bit 20	Bit 19	Bit 18	Reset to Active High	
	GPIO5	Bit 17	Bit 16	Bit 15	Reset to Active High	
	GPIO4	Bit 14	Bit 13	Bit 12	Reset to Active High	
	GPIO3	Bit 11	Bit 10	Bit 9	Reset to Active High	
	GPIO2	Bit 8	Bit 7	Bit 6	Reset to Active High	
	GPIO1	Bit 5	Bit 4	Bit 3	Reset to Active High	
	GPIO0	Bit 2	Bit 1	Bit 0	Reset to Active High	
		0	0	0	Interrupt detected after GPIO is logic 1 for five 66-MHz clocks	Active High
		0	0	1	Interrupt detected after GPIO is logic 0 for five 66-MHz clocks	Active Low
		0	1	0	Interrupt detected after GPIO goes from an logic 0 to logic 1	Rising Edge
		0	1	1	Interrupt detected after GPIO goes from an logic 1 to logic 0	Falling Edge
		1	X	X	Interrupt detected after GPIO goes from an logic 1 to logic 0 or GPIO goes from logic 0 to logic 1	Transitional

The interrupt detection logic uses a pulse width of five 66-MHz clock cycles before an interrupt will occur when a GPIO input is configured as an active high or active low interrupt type. The length of four 66-MHz clocks was chosen to discriminate against small pulses and ensures that glitches are not detected in the active high and active low interrupts when configured in these two interrupt modes.

Once an appropriate interrupt condition is reached, the corresponding bits are set in the General-Purpose Interrupt Status Register (GPISR). The General-Purpose Interrupt Status Register can be read in a polled mode or programmed to generate an interrupt.

If a bit is set in the General-Purpose Interrupt Status Register is set, writing logic 1 to the corresponding set bit in the General-Purpose Interrupt Status Register can clear the bit. For instance, a value of hexadecimal 0x00002001 is read from the General-Purpose Interrupt Status Register. The value that is read from the General-Purpose Interrupt Status Register specifies that GPIO 0 and GPIO 13 both have detected interrupts.

The interrupts can be cleared one-by-one by writing a value hexadecimal value of 0x00002000 to the General-Purpose Interrupt Status Register, followed by writing hexadecimal 0x00000001. Both interrupts may be cleared simultaneously by writing a hexadecimal value of 0x00002001 to the General-Purpose Interrupt Status Register. The General-Purpose Interrupt Status Register will be set to a hexadecimal value of 0x00000000 after receiving a reset. The General-Purpose Interrupt Type Registers will be set to a hexadecimal value of 0x00000000 after receiving a reset.

It is important to note that level interrupts can only be cleared when the interrupt condition that caused the interrupt disappears (i.e., input signal level has changed) whereas edge interrupts can be cleared anytime after the interrupt has occurred.

12.3 Using GPIO 14 and GPIO 15 as Clocks

The GPIO also provides the capability to output two programmable clocks on GPIO14 and GPIO15. The frequency of these two clocks can be up to 33.33 MHz. Writing the General-Purpose Clock Control Register (GPCLKR) can configure these clocks. The General-Purpose Clock Control Register is broken up into two major sections, bits 24 through bit 16 for GPIO 15 and bits 8 through bit 0 for GPIO 14. The nine bits of each section are broken down into three registers, the output control multiplexer register (MUX15 and MUX14), the clock Frequency Terminal Count Register (CLK1TC and CLK0TC), and the duty cycle terminal count (CLK1DC and CLK0DC).

The output control multiplexer register is a single-bit register within the General-Purpose Clock Control Register that determines the location that drives the GPIO. The two places that the GPIO 14 and GPIO 15 can be driven are the General-Purpose Data Output Register or the general-purpose clock. When the General-Purpose Output Enable Register configures GPIO 14 and GPIO15 as outputs, the output control multiplexer register for GPIO 14 and GPIO 15 are used to determine which location the GPIO is driven from. When the output control multiplexer is set as logic 0, the GPIO will be driven from the General-Purpose Data Output Register. When the output control multiplexer is set as logic 1, the GPIO will be driven from the general-purpose clocks.

The general-purpose clock frequency and duty cycle are programmed by configuring the clock Frequency Terminal Count Register and the duty cycle terminal count register which are part of the General-Purpose Clock Control Register. The Frequency Terminal Count Register and the Duty Cycle Terminal Count Register exist and are separate for both GPIO 14 and GPIO 15.

Programming a value into the Frequency Terminal Count Register sets the clock period. The valid frequency values are shown in Table 146.

Table 146. GPIO Clock Frequency Select

Frequency	GPIO15 – Frequency Terminal Count				GPIO14 – Frequency Terminal Count			
	Bit 23	Bit 22	Bit 21	Bit 20	Bit 7	Bit 6	Bit 5	Bit 4
logic 1	0	0	0	0	0	0	0	0
33.33 MHz	0	0	0	1	0	0	0	1
22 MHz	0	0	1	0	0	0	1	0
16.5 MHz	0	0	1	1	0	0	1	1
13.2 MHz	0	1	0	0	0	1	0	0
11 MHz	0	1	0	1	0	1	0	1
9.43 MHz	0	1	1	0	0	1	1	0
8.25 MHz	0	1	1	1	0	1	1	1
7.33 MHz	1	0	0	0	1	0	0	0
6.6 MHz	1	0	0	1	1	0	0	1
6 MHz	1	0	1	0	1	0	1	0
5.5 MHz	1	0	1	1	1	0	1	1
5.08 MHz	1	1	0	0	1	1	0	0
4.71 MHz	1	1	0	1	1	1	0	1
4.4 MHz	1	1	1	0	1	1	1	0
4.125 MHz†	1	1	1	1	1	1	1	1

† When the value of the GPIO frequency terminal count is set to 0xF and the value of the GPIO duty cycle terminal count is set to 0xF, the associated GPIO (GPIO 14 or GPIO 15) will output a clock of 33.33 MHz with a 50% duty cycle.

Table 147. GPIO Duty Cycle Select (Sheet 1 of 2)

Duty Cycle (Low Time)	GPIO15 – Duty Cycle Terminal Count				GPIO14 – Duty Cycle Terminal Count			
	Bit 19	Bit 18	Bit 17	Bit 16	Bit 3	Bit 2	Bit 1	Bit 0
15 ns	0	0	0	0	0	0	0	0
30 ns	0	0	0	1	0	0	0	1
45 ns	0	0	1	0	0	0	1	0
60 ns	0	0	1	1	0	0	1	1
75 ns	0	1	0	0	0	1	0	0
90 ns	0	1	0	1	0	1	0	1
105 ns	0	1	1	0	0	1	1	0
120 ns	0	1	1	1	0	1	1	1
135 ns	1	0	0	0	1	0	0	0
150 ns	1	0	0	1	1	0	0	1
165 ns	1	0	1	0	1	0	1	0
180 ns	1	0	1	1	1	0	1	1
190 ns	1	1	0	0	1	1	0	0

Table 147. GPIO Duty Cycle Select (Sheet 2 of 2)

Duty Cycle (Low Time)	GPIO15 – Duty Cycle Terminal Count				GPIO14 – Duty Cycle Terminal Count			
	Bit 19	Bit 18	Bit 17	Bit 16	Bit 3	Bit 2	Bit 1	Bit 0
210 ns	1	1	0	1	1	1	0	1
225 ns	1	1	1	0	1	1	1	0
240 ns†	1	1	1	1	1	1	1	1

† When the value of the GPIO frequency terminal count is set to 0xF and the value of the GPIO duty cycle terminal count is set to 0xF, the associated GPIO (GPIO 14 or GPIO 15) will output a clock of 33 MHz with a 50% duty cycle.

There is one special condition defined for the output clock on GPIO 14 and GPIO 15. When the GPIO Frequency Terminal Count = F and the GPIO Duty Cycle Terminal Count = F, the output of the clock generator block will be 33.33 MHz with a 50% duty cycle. If the GPIO Duty Cycle Terminal Count is programmed to be greater than or equal to the GPIO Frequency Terminal Count (TC), the clock generator will output logic 1. The General-Purpose Clock Control Register is set to a hexadecimal value of 0x01100000 after receiving a reset.

12.4 Register Description

Table 148. GPIO Registers Overview

Address	R/W	Name	Description
0xC8004000	R/W	GPOUTR	GPIO pin data output register.
0xC8004004	R/W	GPOER	GPIO pin out enable register.
0xC8004008	R	GPINR	GPIO pin status register.
0xC800400C	R/W	GPISR	GPIO interrupt status register.
0xC8004010	R/W	GPIT1R	GPIO interrupt type register, inputs 7:0
0xC8004014	R/W	GPIT2R	GPIO pin interrupt type register, inputs 15:8
0xC8004018	R/W	GPCLKR	GPIO Clock Control Register
0xC800401C	RO	Reserved	Reserved

12.4.1 GPIO Output Register

(GPOUTR)

Each pin's output data is controlled by programming this register. Each of the 16 bits in the register represents the data to be put on the output through a tristate buffer, depending upon the status of the GPOER.

Register Name:	GPOUTR																													
Hex Offset Address:	0xC800 4000																Reset Hex Value:	0x00000000											
Register Description:	I/O Output register. Controls output value of GPIO pins, depending on tristate control from GPOER.																													
Access: Read/Write.																														
31																16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																	DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Register		GPOUTR
Bits	**Name**	**Description**
31:16		(Reserved). Reads back 0.
15	DO15	1 = Output a 1 on output pin, depends on GPCLKR24, GPOER15 0 = Output a 0 on output pin, depends on GPCLKR24, GPOER15 Reset: 0
14	DO14	1 = Output a 1 on output pin, depends on GPCLKR8, GPOER14 0 = Output a 0 on output pin, depends on GPCLKR8, GPOER14 Reset: 0
13:9	DO13:DO9	1 = Output a 1 on output pin, depends on GPOER[13:9] 0 = Output a 0 on output pin, depends on GPOER[13:9] Reset: 0
8:0	DO8:DO0	1 = Output a 1 on output pin GPOER[8:0] 0 = Output a 0 on output pin GPOER[8:0] Reset: 0

12.4.2 GPIO Output Enable Register

(GPOER)

Each pin's output tristate buffer is controlled by programming this register. Each of the 16 bits in the register represents the control to the tristate buffer of the output pin.

Register Name:	GPOER																											
Hex Offset Address:	0xC800 4004															**Reset Hex Value:**	0x00007FFF										
Register Description:	I/O Output Enable register. Turns on output driver.																											
Access: Read/Write.																												
31															**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																OE15	OE14	OE13	OE12	OE11	OE10	OE9	OE8	OE7	OE6	OE5	OE4	OE3	OE2	OE1	OE0

Register		GPOER
Bits	**Name**	**Description**
31:16		(Reserved). Reads back 0.
15	OE15	1 = Output pin is tristated or input 0 = Output pin is driven Reset: 0 (as clock is driven out on this pin during reset).
14:0	OE14:OE0	1 = Output pin is tristated or input 0 = Output pin is driven Reset: 1

12.4.3 GPIO Input Register

(GPINR)

This is a read-only register. This register contains the level of the I/O pin, either a 1 or a 0.

Register Name:	GPINR				
Hex Offset Address:	0xC800 4008		Reset Hex Value:	0x00000000	
Register Description:	This register is used to monitor input pins.				
Access: Read.					

31														16	15														0
(Reserved)															IN_LEV													

Register		GPINR
Bits	Name	Description
31:16	-	Not used. Ignored on writes and driven logic 0 on reads.
15:0	IN_LEV	Level of general purpose inputs 15-0 1 = 1 on GPIO 0 = 0 on GPIO Reset = 0x0

12.4.4 GPIO Interrupt Status Register

(GPISR)

This register is used to store status of a GP input interpreted as an interrupt. GP input interrupts can be configured as active high, active low, rising edge, falling edge, or transitional depending on the configuration of the GPIT[1:0]R register. A 1 read from this register indicates a pending interrupt. Writing 1 back to this register will clear the interrupt provided the interrupting condition no longer exists. The interrupts are all masked in the Interrupt Controller block.

Register Name:	GPISR				
Hex Offset Address:	0xC800 400C		Reset Hex Value:	0x00000000	
Register Description:	This register is used to store status of interrupts received on GP input pins.				
Access: Read/Write.					

31														16	15		13	12											0
(Reserved)															Not Used			INT_STAT											

Register		GPISR
Bits	Name	Description
31:16		Not used. Ignored on writes and driven logic '0' on reads.
15:13		Not used.
12:0	INT_STAT	1 = Interrupt pending. 0 = No interrupt pending.

12.4.5 GP Interrupt Type Register 1

(GPIT1R)

This register describes how to interpret GPIO [7:0] as interrupts — either level or edge — along with high, low, rising, falling, and transitional. Three bits describe each GPIO pin, as described in the following table.

Register Name:	GPIT1R				
Hex Offset Address:	0xC800 4010		Reset Hex Value:	0x00000000	
Register Description:	This register is used to control interrupt type for GPIO 7:0.				
Access: See below.					

31							24	23		21	20		18	17		15	14		12	11		9	8		6	5		3	2		0
(Reserved)								GPIO7			GPIO6			GPIO5			GPIO4			GPIO3			GPIO2			GPIO1			GPIO0		

Register		GPIT1R
Bits	Name	Description
31:24		Not used. Ignored on writes and driven logic ‘0’ on reads.
23:21	GPIO7	000 = Active High 001 = Active Low 010 = Rising Edge 011 = Falling Edge 1xx = Transitional Resets to 000 – Active High
20:18	GPIO6	As per GPIO7
17:15	GPIO5	As per GPIO7
14:12	GPIO4	As per GPIO7
11:9	GPIO3	As per GPIO7
8:6	GPIO2	As per GPIO7
5:3	GPIO1	As per GPIO7
2:0	GPIO0	As per GPIO7

12.4.6 GPIO Interrupt Type Register 2

(GPIT2R)

This register describes how to interpret GPIO[15:8] as interrupts — either level or edge — along with high, low, rising, falling, and transitional. Three bits describe each GPIO pin, as described in the following table.

Register Name:	GPIT2R				
Hex Offset Address:	0xC800 4014		Reset Hex Value:	0x00000000	
Register Description:	This register is used to control interrupt type for GPIO 12:8.				
Access: Read/Write.					

31							24	23		21	20		18	17		15	14		12	11		9	8		6	5		3	2		0
(Reserved)								Not used			Not used			Not used			GPIO12			GPIO11			GPIO10			GPIO9			GPIO8		

Register		GPIT2R (Sheet 1 of 2)
Bits	Name	Description
31:24		Not used. Ignored on writes and driven logic ‘0’ on reads.
23:21	GPIO15	Not used.
20:18	GPIO14	Not used.

Register		GPIT2R (Sheet 2 of 2)
Bits	**Name**	**Description**
17:15	GPIO13	Not used.
14:12	GPIO12	000 = Active High 001 = Active Low 010 = Rising Edge 011 = Falling Edge 1xx = Transitional Resets to 000 – Active High
11:9	GPIO11	As per GPIO12
8:6	GPIO10	As per GPIO12
5:3	GPIO9	As per GPIO12
2:0	GPIO8	As per GPIO12

12.4.7 GPIO Clock Register

(GPCLKR)

This register controls the use of GPIO15 and GPIO14 as clock outputs. GPOER defines the output enable for the driver; this register controls both the clock dividers and a MUX between the clock data and the data defined in GPOUTR.

Register Name:	GPCLKR				
Hex Offset Address:	0xC800 4018		**Reset Hex Value:**	0x01100000	
Register Description:	This register controls the use of GPIO 15 and GPIO14 as clock sources.				
Access: See below.					

31						25	24	23			20	19			16	15						9	8	7				4	3			0
(Reserved)							MX15	CLK1TC				CLK1DC				(Reserved)							MX14	CLK0TC				CLK0DC			

Register		GPCLKR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:25		Not used. Ignored on writes and driven logic 0 on reads.
24	MUX15	0 = Control from GPOUTR Register 1 = Clock output Reset: 1
23:20	CLK1TC	Terminal count for a 4 bit up counter @ PCLK. An F in this field and the CLK1DC field is a special case to provide PCLK/2. Reset: 0x1
19:16	CLK1DC	Represents the number of counts for which clock output should be low. Reset: 0x0
15:9	-	Not used. Ignored on writes and driven logic 0 on reads.

Register		GPCLKR (Sheet 2 of 2)
Bits	**Name**	**Description**
8	MUX14	0 = Control from GPOUTR Register 1 = Clock Output Reset: 0
7:4	CLK0TC	Terminal count for a 4-bit up counter @ PCLK. An F in this field and the CLK0DC field is a special case to provide PCLK/2. Reset: 0x0
3:0	CLK0DC	Represents the number of counts for which clock output should be low. Reset: 0x0

§ §

13.0 Interrupt Controller

The Interrupt Controller takes as inputs 32 individual interrupts. These 32 individual interrupts originate either from the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor internal blocks or from 14 dedicated GPIO pins.

The highest priority interrupt is bit [0] assigned to the WAN/Voice NPE and the lowest priority interrupt is bit [31] assigned to a Software Interrupt. The Intel® IXP42X product line and IXC1100 control plane processors' Interrupt Controller is implemented to expand the interrupt capabilities of the Intel XScale® Processor.

The Intel XScale processor only has a fast interrupt (FIQ) and a maskable interrupt (IRQ). The interrupts collected by the Interrupt Controller are combined and configured to be an FIQ interrupt or an IRQ. The FIQ signal going to the Intel XScale processor will be set when any of the interrupts assigned to be an FIQ become set. The IRQ signal going to the Intel XScale processor will be set when any of the interrupts assigned to be an IRQ are set.

The Interrupt Controller consists of:

- A 32-bit interrupt status register
- A 32-bit interrupt select register
- A 32-bit FIQ status register
- A 6-bit IRQ highest priority register
- A 32-bit interrupt enable register
- A 32-bit IRQ status register
- A 32-bit interrupt priority register
- A 6-bit FIQ highest priority register

The Interrupt Controller has no concept of setting or clearing any interrupts. The intent of the Interrupt Controller is to collect and prioritize the received interrupts from other sources. In order to set an interrupt, the device connected to the assigned interrupt line must assert the interrupt.

Clearing of the interrupt must be made at the device that caused the interrupt. If the interrupt is not cleared at the device that asserted the interrupt, the interrupt will be service again.

13.1 Interrupt Priority

Selecting the priority of interrupts is done through the assignment of the interrupt priority register and natural interrupt priority assigned by interrupt number. As described, the interrupts follow a natural priority. The interrupt connected to interrupt 0 (WAN/Voice NPE) has the highest priority and the interrupt connected to interrupt 31 (Software Interrupt 1) has the lowest priority.

In addition to the natural priority the lowest eight interrupts, interrupt 0 to interrupt 7, can be assigned a priority value by writing the interrupt priority register (INTR_PRTY). The interrupt priority register is broken up into eight 3-bit registers.

Bits 0 through 2 of the interrupt priority register assign a priority value to interrupt 0 (WAN/Voice NPE). Bits 3 through 5 of the interrupt priority register assign a priority to interrupt 1 (Ethernet A NPE). The interrupt priority values are assigned in a similar

pattern to the assignments above for the first eight interrupts with the last interrupt priority assignment being bits 21 through 23 of the interrupt priority register assigning a priority value to interrupt 7.

The 3-bit interrupt priorities for each of the first eight interrupts can take on a value from 0 to 7. A value of 0 — located the 3-bit interrupt priority register for each of the first eight interrupts — signifies the highest priority interrupt. A value of 7 — located the 3-bit interrupt priority register for each of the first eight interrupts — signifies the lowest priority interrupt. In the case of two interrupts being assigned the same priority, the interrupts natural priority will assign who has the highest priority.

For example, interrupt number 1 and interrupt number 3 both have a value of 0 — written as their 3-bit interrupt priorities. Interrupt number 1 would take priority over interrupt number 3 due to their individual natural priorities.

The priorities assigned to each of the 3-bit interrupt priorities for the first eight interrupts will be set to a value of the corresponding interrupt number (interrupt number 0 gets assigned a value of 0 in the associated 3-bit interrupt priority register, interrupt number 1 gets assigned a value of 1, etc.) when receiving a reset. Therefore, allowing natural priorities to be the default after a reset.

13.2 Assigning FIQ or IRQ Interrupts

The IXP42X product line and IXC1100 control plane processors Interrupt Controller provide the capability to assign each interrupt as an FIQ or an IRQ interrupt. As discussed earlier, the Intel XScale processor only receives a single FIQ interrupt signal and a single IRQ interrupt signal.

The Interrupt Controller allows multiple interrupts to be sent to the Intel XScale processor as either FIQ interrupts or IRQ interrupts. Each interrupt may be assigned as an FIQ interrupt or an IRQ interrupt but never both. The interrupts are assigned as an FIQ interrupt or an IRQ interrupt by writing bits in the interrupt select register (INTR_SEL).

The Interrupt Select Register is a 32-bit register that assigns each of the 32 interrupts to as an FIQ interrupt or an IRQ interrupt. Bit 0 of the Interrupt Select Register corresponds to interrupt number 0 (WAN/Voice NPE). Bit 31 of the Interrupt Select Register corresponds to interrupt number 31 (Software Interrupt 1).

Logic 1 written to a bit in the Interrupt Select Register will assign the corresponding interrupt number as an FIQ interrupt. Writing logic 0 to the same bit in the Interrupt Select Register will assign the corresponding interrupt number as an IRQ interrupt.

For example, the Interrupt Select Register is written with a hexadecimal value of 0x00000005. The result of this write would set interrupt number 0 (WAN/Voice NPE) as an FIQ and interrupt number 2 (Ethernet NPE B) as an FIQ.

All other interrupt numbers would be assigned as IRQ interrupts. All interrupts are assigned as IRQ interrupts upon receiving a reset.

13.3 Enabling and Disabling Interrupts

The interrupts on the IXP42X product line and IXC1100 control plane processors can be individually enabled or disabled by writing to the Interrupt-Enable Register (INTR_EN). By disabling an interrupt, the Intel XScale processor will not be interrupted by either the FIQ interrupt signal or the IRQ interrupt signal when an interrupt occurs on the corresponding disabled interrupt number.

For instance, interrupt number 0 is disabled and an interrupt occurs on interrupt number 0. The interrupt generated by interrupt number 0 will not be seen by the Intel XScale processor.

The Interrupt-Enable Register is a 32-bit register that can individually enable or disable each of the 32 interrupts. Bit 0 of the Interrupt-Enable Register corresponds to interrupt number 0 (WAN/Voice NPE). Bit 31 of the Interrupt-Enable Register corresponds to interrupt number 31 (Software Interrupt 1).

Logic 1 written to a bit in the Interrupt-Enable Register will enable the corresponding interrupt number. Writing logic 0 to the same bit in the Interrupt-Enable Register will disable the corresponding interrupt number.

For example, the Interrupt-Enable Register is written with a hexadecimal value of 0x0000000A. The result of this write would enable interrupt number 1 (Ethernet NPE A) and interrupt number 3 (Queue Manager Queues 1-32). All other interrupt numbers would be disabled. All interrupts are disabled upon receiving a reset.

13.4 Reading Interrupt Status

The IXP42X product line and IXC1100 control plane processors provide several mechanisms in which interrupt status can be obtained from the Interrupt Controller. One method of obtaining interrupt status is to read the interrupt status register directly (INTR_ST).

The Interrupt Status Register is a 32-bit register that has a one-for-one relationship with the interrupt number. Interrupt number 0 (WAN/Voice NPE) will be the status represented on bit 0 of the Interrupt Status Register and interrupt number 31 will be the status represented on bit 31 of the Interrupt Status Register.

Reading Logic 1 from a bit in the Interrupt Status Register represent that the device connected to that particular interrupt number has asserted an interrupt to the Interrupt Controller. For example, a read is performed on the Interrupt Status Register and the result returned is a hexadecimal 0x00000002. The Interrupt Status Register is telling the Intel XScale processor that the interrupt number 1 (Ethernet NPE A) has caused an interrupt.

The Intel XScale processor will service the interrupt and clear the interrupt by updating the register that caused the interrupt condition in the Ethernet NPE A. Using the Interrupt Status Register allows an interrupt service routine to assign relative priorities to the interrupts and removes all relationship from the priority algorithms or the interrupt types assigned by the IXP42X product line and IXC1100 control plane processors Interrupt Controller. The Interrupt Status Register is set to all zeros upon reset.

All other methods of reading interrupt status will involve the use of the register sets provided by the IXP42X product line and IXC1100 control plane processors to aid in the determination of which interrupts should be serviced first.

The IXP42X product line and IXC1100 control plane processors provide the capability of reading the interrupt status of the interrupt numbers that have been assigned as FIQ interrupts or reading the interrupt status of the interrupt numbers that have been assigned as IRQ interrupts. The status of the interrupt numbers — that have been assigned as FIQ interrupts — can be read by reading the FIQ status register (INTR_FIQ_ST). The status of the interrupt numbers that have been assigned as IRQ interrupts can be read by reading the IRQ status register (INTR_IRQ_ST).

The FIQ Status Register and the IRQ Status Register are 32-bit registers that have a one-for-one relationship with the interrupt number. Interrupt number 0 (WAN/Voice NPE) will be the status represented on bit 0 of both the FIQ Status Register and the IRQ Status Registers. Interrupt number 31 will be the status represented on bit 31 of both the FIQ Status Register and the IRQ Status Registers.

Reading logic 1 from a bit in either the FIQ Status Register or the IRQ Status Registers represent that the device connected to that particular interrupt number has asserted an interrupt to the Interrupt Controller. For example, a read is performed on the FIQ Status Register and the result returned is a hexadecimal 0x00000001. The Interrupt Status Register is telling the Intel XScale processor that the interrupt number 0 (WAN/Voice NPE) has caused an interrupt and the interrupt is assigned as an FIQ interrupt.

The Intel XScale processor will service the interrupt and clear the interrupt by updating the register that caused the interrupt condition in the WAN/Voice NPE. The same action will be applied to an interrupt that would be caused by an IRQ interrupt. Allowing the capability to separate the FIQ and IRQ interrupts allows separate interrupt service routines to be built based on the type of interrupt received. This allows greater control in application that may be developed by the IXP42X product line and IXC1100 control plane processors.

The FIQ Status Register and IRQ Status Register are set to all zeros upon reset.

The IXP42X product line and IXC1100 control plane processors also allows the capability to read the highest-priority IRQ interrupt or the highest-priority FIQ interrupt as determined by the priority algorithm described in “Interrupt Priority” on page 398.

The highest-priority IRQ interrupt can be read by reading the IRQ Highest-Priority Register (INTR_IRQ_ENC_ST). The highest-priority FIQ interrupt can be read by reading the FIQ Highest-Priority Register (INTR_FIQ_ENC_ST). The IRQ Highest-Priority Register and the FIQ Highest-Priority Registers are 6-bit registers that will return the highest-priority interrupt of each the IRQ interrupts and the FIQ interrupts.

The value obtained by reading the IRQ Highest-Priority Register will be the interrupt number of the highest-priority IRQ interrupt, incremented by one and the sum of the add left shifted by two bits. Therefore, the six bits — that contain the IRQ highest priority — are actually located in bits 2 through 7 of the IRQ Highest-Priority Register.

For example, interrupt number 1 (Ethernet NPE A) is the highest-priority IRQ interrupt, the value obtained when reading the IRQ Highest-Priority Interrupt Register would be hexadecimal 0x00000008. A value of 0 — returned when reading the IRQ Highest-Priority Register — signifies that no IRQ interrupts are pending.

The IRQ Highest-Priority Register will be reset to a value of 0. The FIQ Highest-Priority Register will behave in an identical fashion to the IRQ Highest-Priority Register.

13.5 Interrupt Controller Register Description

Table 149. Interrupt Controller Registers

Address	R/W	Name	Description
0xC8003000	R	INTR_ST	Interrupt Status Register
0xC8003004	R/W	INTR_EN	Interrupt Enable Register
0xC8003008	R/W	INTR_SEL	Interrupt Select Register
0xC800300C	R	INTR_IRQ_ST	IRQ Status register
0xC8003010	R	INTR_FIQ_ST	FIQ status Register

Table 149. Interrupt Controller Registers

Address	R/W	Name	Description
0xC8003014	R/W	INTR_PRTY	Interrupt Priority Register
0xC8003018	R	INTR_IRQ_ENC_ST	IRQ Highest Priority Register
0xC800301C	R	INTR_FIQ_ENC_ST	FIQ Highest Priority Register

13.5.1 Interrupt Status Register

Register Name:	INTR_ST				
Hex Offset Address:	0xC800 3000		Reset Hex Value:	0x00000000	
Register Description:	This register indicates the state of each incoming interrupt.				
Access: Read.					
31					0
Incoming Interrupt Status					

Register			INTR_ST (Sheet 1 of 2)
Interrupt Bit	Default Priority [1]	Source	Description
Int0	0	WAN/Voice NPE	Debug/Execution/MBox
Int1	1	Ethernet NPE A	Debug/Execution/MBox
Int2	2	Ethernet NPE B	Debug/Execution/MBox
Int3	3	Queue Manager	Queue[1-32]
Int4	4	Queue Manager	Queue[33-64]
Int5	5	Timers	General-Purpose Timer 0
Int6	6	GPIO	GPIO[0]
Int7	7	GPIO	GPIO[1]
Int8	8	PCI	PCI Interrupt
Int9	9	PCI	PCI DMA Channel 1
Int10	10	PCI	PCI DMA Channel 2
Int11	11	Timers	General-Purpose Timer 1
Int12	12	USB	USB
Int13	13	Console UART	Console UART
Int14	14	Timers	Timestamp Timer
Int15	15	High-Speed UART	High-Speed UART
Int16	16	Timers	Watchdog Timer
Int17	17	Performance Monitoring Unit	Performance Monitoring Unit counter rollover
Int18	18	Intel XScale® Processor Performance Monitoring Unit	Intel XScale processor PMU counter rollover
Int19	19	GPIO	GPIO[2]
Int20	20	GPIO	GPIO[3]

Register			INTR_ST (Sheet 2 of 2)
Interrupt Bit	**Default Priority [1]**	**Source**	**Description**
Int21	21	GPIO	GPIO[4]
Int22	22	GPIO	GPIO[5]
Int23	23	GPIO	GPIO[6]
Int24	24	GPIO	GPIO[7]
Int25	25	GPIO	GPIO[8]
Int26	26	GPIO	GPIO[9]
Int27	27	GPIO	GPIO[10]
Int28	28	GPIO	GPIO[11]
Int29	29	GPIO	GPIO[12]
Int30	30	Expansion Bus	SW Interrupt 0
Int31	31	Expansion Bus	SW Interrupt 1

1. The priorities of Interrupts 0 through 7 are programmable; the priorities of Interrupts 8 through 31 are fixed; priority 0 is the highest priority and 31 is the lowest.

13.5.2 Interrupt-Enable Register

Register Name:	INTR_EN				
Hex Offset Address:	0xC800 3004		Reset Hex Value:	0x00000000	
Register Description:	Provides enables for the interrupts. This register allows the Intel XScale processor to disable interrupts from selected blocks. To enable an interrupt, a 1 is written into corresponding bit, to disable it, a 0 is written.				
Access: Read/Write.					

31																												0
Interrupt Enables																												

13.5.3 Interrupt Select Register

Register Name:	INTR_SEL				
Hex Offset Address:	0xC800 3008		Reset Hex Value:	0x00000000	
Register Description:	This register decides if an interrupt is to be presented to the Intel XScale processor as an FIQ or an IRQ. If a bit corresponding to an interrupt is set (to 1), that interrupt is presented as a FIQ. If the bit is reset to 0, the interrupt is presented as an IRQ.				
Access: Read/Write.					

31																												0
Interrupt Selects																												

13.5.4 IRQ Status Register

Register Name:	INTR_ IRQ_ST				
Hex Offset Address:	0xC800 300C		Reset Hex Value:	0x00000000	
Register Description:	This register is an “AND” of the incoming status with the INTR_EN and the inverted version of the INTR_SEL. The IRQ_ST indicates which of the incoming interrupts are enabled as an IRQ. An Interrupt is enabled if the corresponding bit is set, else it is disabled.				
Access: Read.					

31																												0
IRQ Status Information																												

13.5.5 FIQ Status Register

Register Name:	INTR_FIQ_ST				
Hex Offset Address:	0xC800 3010		Reset Hex Value:	0x00000000	
Register Description:	This register is an “AND” of the incoming status with the INTR_EN and the INTR_SEL. The IRQ_ST indicates which of the incoming interrupts are enabled as a FIQ. An Interrupt is enabled if the corresponding bit is set. Otherwise, it is disabled.				
Access: Read.					

31																												0
FIQ Status Info																												

13.5.6 Interrupt Priority Register

Register Name:	INTR_PRTY				
Hex Offset Address:	0xC800 3014		Reset Hex Value:	0x00FAC688	
Register Description:	The highest eight priority interrupts can be programmed via this register, each of the 3-bit sets can be programmed to any priority from 0(000) through 7(111). This register applies to both IRQ and FIQ interrupts.				
Access: Read/Write.					

31							24	23																				0
Undefined/Zero								Interrupt Priority selects																				

Register		INTR_PRTY
Bits	Name	Description
31	Zero	Read as undefined, write as 0
30	Zero	Read as undefined, write as 0
29	Zero	Read as undefined, write as 0
28	Zero	Read as undefined, write as 0
27	Zero	Read as undefined, write as 0
26	Zero	Read as undefined, write as 0
25	Zero	Read as undefined, write as 0
24	Zero	Read as undefined, write as 0
23:21	Prior_Intbus7 [2:0]	Set the priority of the Intr_bus [7]; default is 7
20:18	Prior_Intbus6 [2:0]	Set the priority of the Intr_bus [6]; default is 6
17:15	Prior_Intbus5 [2:0]	Set the priority of the Intr_bus [5]; default is 5
14:12	Prior_Intbus4 [2:0]	Set the priority of the Intr_bus [4]; default is 4
11:9	Prior_Intbus3 [2:0]	Set the priority of the Intr_bus [3]; default is 3
8:6	Prior_Intbus2 [2:0]	Set the priority of the Intr_bus [2]; default is 2
5:3	Prior_Intbus1 [2:0]	Set the priority of the Intr_bus [1]; default is 1
2:0	Prior_Intbus0 [2:0]	Set the priority of the Intr_bus [0]; default is 0

13.5.7 IRQ Highest-Priority Register

Register Name:	INTR_IRQ_ENC_ST				
Hex Offset Address:	0xC800 3018		Reset Hex Value:	0x00000000	
Register Description:	This register returns the “incremented number” of the highest-priority interrupt that is pending for the IRQ. For example, if interrupt 0 is the highest IRQ pending, the register returns 1. If the register returns 0, it means that there is no interrupt pending or a spurious interrupt. Note that the encoded number is shifted left by two bits, a software requirement for the value to be multiplied by 4 before being read.				
Access: Read.					

31																					8	7					2	1	0
(Undefined)																						IRQ_ENC_ST						RES	

Register		INTR_IRQ_ENC_ST
Bits	**Name**	**Description**
31:8	Zero	Read as undefined
7:2	IRQ_ENC_ST	Indicates the highest priority pending Maskable interrupt (the interrupt “number” incremented by 1)
1:0	Zero	Zero

13.5.8 FIQ Highest-Priority Register

Register Name:	INTR_FIQ_ENC_ST				
Hex Offset Address:	0xC800 301C		Reset Hex Value:	0x00000000	
Register Description:	This register returns the “incremented number” of the highest-priority interrupt that is pending for the FIQ. For example, if interrupt ‘0’ is the highest FIQ pending, the register returns 1. Note that the encoded number is shifted left by two bits, a software requirement for the value to be multiplied by 4 before being read.				
Access: Read.					

31																					8	7					2	1	0
(Undefined)																						FIQ_ENC_ST						RES	

Register		INTR_FIQ_ENC_ST
Bits	**Name**	**Description**
31:8	Zero	Read as undefined
7:2	FIQ_ENC_ST	Indicates the highest-priority, pending FIQ interrupt (the interrupt “number” incremented by 1)
1:0	Zero	Zero

§ §

14.0 Timers

The Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor contain four 32-bit internal timers that increment on the rising edge of a 66.66 MHz (which is 2* OSC_IN input pin.). The watch-dog timer is a 32-bit, down counter that may be used by software applications to monitor inactivity. The time-stamp counter is a 32-bit, free-running, up counter that may be used by software applications to maintain a real time count or apply time stamps to events.

The two general-purpose counters are 32-bit counters that may be used by software applications to generate periodic interrupts to the Intel® IXP42X product line and IXC1100 control plane processors' Interrupt Controller. The general-purpose timer interrupts sent to the Interrupt Controller will allow software application, running on the Intel XScale® processor, to aid in the implementation of system-level traffic-shaping algorithms. Alternately, they can be used as an operating-system timer.

All read and write accesses to the timers must be made as 32-bit accesses. Bits that correspond to reserved register bits are ignored on writes. These same reserved register bits will be returned as zeros on read accesses.

14.1 Watch-Dog Timer

The watch-dog timer is composed of a 32-bit writeable down counter, a 3-bit enable register, a 16-bit key register, and a 5-bit status register. The watch-dog timer can only be written or read by utilizing the APB bus. The Watch-Dog Timer Key Register (ost_wdog_key) can be written at any time over the APB bus. However, the Watch-Dog Timer Down Counter (ost_wdog) and the Watch-Dog Enable Register (ost_wdog_enab) can only be written when the Watch-Dog Timer Key Register contains the value 0x482E.

Note: The value 0x482E will be referred to as the "key-value" from here on.

A write to the Watch-Dog Timer Down Counter and the Watch-Dog Timer Enable Register when the Watch-Dog Timer Key Register does not equal the key-value will have no effect on the Watch-Dog Timer Down Counter or the Watch-Dog Timer Enable Register. The Watch-Dog Timer Key Register is provided to prevent accidental writes to the Watch-Dog Timer Down Counter Register and the Watch-Dog Timer Enable Register. Typical operation of the Watch-Dog Counter would be for the software application to write the key-value into the Watch-Dog Timer Key Register, write to the Watch-Dog Timer Down Counter or the Watch-Dog Timer Enable Register, then write a value other than key-value into the Watch-Dog Timer Key Register. The software application will periodically write the key-value repeat the above process to prevent the Watch-Dog Timer Down Counter from reaching zero.

The watch-dog enable register contains 3-bits. The 3-bits are the watch-dog counter enable bit, the watch-dog interrupt enable bit, and the watch-dog reset enable bit. The watch-dog counter enable bit enables and disables the Watch-Dog Timer Down Counter register. When the watch-dog counter enable bit is set to logic 1, the Watch-Dog Timer Down Counter will decrement. When the watch-dog counter enable bit is set to logic 0 the watch-dog counter down counter will halt.

The watch-dog interrupt enable bit enables and disables the interrupt that may be generated to the IXP42X product line and IXC1100 control plane processors Interrupt controller when the Watch-Dog Timer Down Counter reaches zero. When the watch-dog interrupt enable bit is set to logic 1, the Watch-Dog Timer Down Counter will cause an interrupt to be generated to the Interrupt Controller when the down counter reaches a value of zero. Writing logic 1 to the associated bit in the watch-dog status register will clear the interrupt. When the watch-dog interrupt enable bit is set to logic 0, the Watch-Dog Timer Down Counter will not cause an interrupt to be generated to the Interrupt Controller when the down counter reaches a value of zero.

The watch-dog reset enable bit enables and disables the watch-dog timer chip reset capability. The watch-dog timer can be configured to reset the chip after the Watch-Dog Timer Down Counter reaches zero. If the watch-dog reset enable bit is set to logic 1 and the watch-dog counter down counter reaches a value of zero, a reset signal will be asserted to the chip causing all register to be set to their associated default values and the watch-dog timer warm reset bit will be set to a logic 1. The warm reset bit in the watch-dog status register will signify to the software application that the watch-dog timer reaching a count of zero caused the last reset event to occur. The warm reset bit in the watch-dog status register can be cleared by writing logic 1 to the warm reset bit in the watch-dog status register or the system level reset being asserted.

The watch-dog timer enable bit, watch-dog timer interrupt enable bit, and the watch-dog timer reset enable bit will be disabled after reset. Therefore, disabling the Watch-Dog Timer Down Counter. A reset will cause the Watch-Dog Timer Down Counter to assume a value of all ones.

14.2 Time-Stamp Timer

The time-stamp timer is a readable 32-bit, free-running counter. When reset occurs, the time-stamp timer is set to all zeros and starts counting up as soon as reset is released. When the time-stamp timer reaches the maximum value the counter rolls over to zero and continues to count.

The time-stamp counter will also generate an interrupt signal to the IXP42X product line and IXC1100 control plane processors' Interrupt Controller. The time-stamp timer-interrupt signal contained in the Timer-Status Register can be cleared by a writing a 1 to the associated time-stamp timer interrupt bit in the Timer-Status Register.

14.3 General-Purpose Timers

The two general-purpose timers are composed of a 32-bit, down counter, a one-shot control bit, a count-enable bit, a 30-bit reload register, and an interrupt-status bit in the Timer Status Register. The 30-bit, reload register will be used to load the most-significant 30 bits of the 32-bit, general-purpose timer down counter. The timer will be reloaded immediately on setting the timer reload register. The least-significant two bits of the 32-bit, down counter will be loaded with zeros.

The 32-bit, general-purpose down counter will only decrement when the general-purpose timer control enable bit is set to logic 1. Logic 0 — written to the general-purpose timer control enable bit — will halt the counting of the 32-bit, general-purpose down counters.

The general-purpose-timer, one-shot control bit will be utilized to determine the event that takes place with the general-purpose timer down counter after the counter has reached a value of 0. The general-purpose, timer down counter can load the 30-bit, reload value back into the down counter after reaching 0 or the general-purpose, timer down counter can stop after reaching 0.

The general-purpose-timer, one-shot control bit will be used to select which of the preceding events take place after the general-purpose, timer-down counter reaches a value of 0. When the general-purpose-timer, one-shot control bit is set to logic 0:

- The general-purpose, timer-down counter will be reloaded with the general-purpose timer, 30-bit reload register
- An interrupt will be sent to the IXP42X product line and IXC1100 control plane processors' Interrupt Controller
- The interrupt will be captured by the timer status block
- The 32-bit, general-purpose timer counter will continue to decrement with the counters newly loaded values

Each general-purpose timer will have independent interrupts supplied to the IXP42X product line and IXC1100 control plane processors Interrupt Controller and independent interrupt status bit contained in the Timer Status Register. The interrupts contained in the timer-status registers can be cleared by writing logic 1 to the associated interrupt bit in the Timer Status Register.

The general-purpose timers behave differently when the general-purpose timer, one-shot control bit is set to logic 1. When the general-purpose timer, one-shot control bit is set to logic 1:

- The general-purpose, timer-down counter will down count to 0
- The general-purpose, timer-down counter will be reloaded with the general-purpose timer, 30-bit reload register
- An interrupt will be sent to the IXP42X product line and IXC1100 control plane processors' Interrupt Controller
- The interrupt will be captured by the timer-status block
- The 32-bit general-purpose timer counter will halt, by setting the general-purpose, timer-count enable bit to a logic 0.

The general-purpose timer count enable bit will need to be set to a logic 1, to resume the counting by the 32-bit, general-purpose, timer-down counter.

The General-Purpose Timers Reload Register and down counter will be reset to 0 when receiving a reset. The general-purpose timers will be disabled, when receiving a reset.

14.4 Timer Register Definition

Table 150. Timer Registers

Address	R/W	Name	Description
0xC8005000	R/W	ost_ts	Time-Stamp Timer
0xC8005004	R	ost_tim0	General-Purpose Timer 0
0xC8005008	R/W	ost_tim0_rl	General-Purpose Timer 0 Reload
0xC800500C	R	ost_tim1	General-Purpose Timer 1
0xC8005010	R/W	ost_tim1_rl	General-Purpose Timer 1 Reload
0xC8005014	R/W	ost_wdog*	Watch-Dog Timer
0xC8005018	R/W	ost_wdog_enab*	Watch-Dog Enable Register
0xC800501C	R/W	ost_wdog_key	Watch-Dog Key Register
0xC8005020	R	ost_sts	Timer Status
Note: Only writeable through watch-dog protection mechanism.			

14.4.1 Time-Stamp Timer

Register Name:	**OST_TS**				
Hex Offset Address:	0x C800 5000		**Reset Hex Value:**	0x00000000	
Register Description:	Time-stamp timer				
Access: Read/Write.					

31																												**0**
timer_val																												

Register		**OST_TS**
Bits	**Name**	**Description**
31:0	timer_val	Current value of the time-stamp timer.

14.4.2 General-Purpose Timer 0

Register Name:	**OST_TIM0**				
Hex Offset Address:	0x C800 5004		**Reset Hex Value:**	0x00000000	
Register Description:	General-Purpose Timer 0				
Access: Read.					

31																												**0**
timer_val																												

Register		**OST_TIM0**
Bits	**Name**	**Description**
31:0	timer_val	Current value of the general-purpose timer.

14.4.3 General-Purpose Timer 0 Reload

Register Name:	OST_TIM0_RL				
Hex Offset Address:	0x C800 5008		Reset Hex Value:	0x00000000	
Register Description:	General-Purpose Timer 0 Reload				
Access: Read/Write.					

31																										2	1	0
reload_val																											timer control	

Register		OST_TIM0_RL
Bits	Name	Description
31:2	reload_val	Value loaded into ost_tim0. bits 1 and 0 will be 00. ***Note:*** As the counter counts to zero writing a value of X to these bits will generate interrupts at intervals of (X*4)+1 cycles.
1	tim1_one_shot	The one shot control bit specifies what action should be taken when the general-purpose down counter reaches 0. If the one-shot control bit is set to 0, the counter will load the reload value and continue to count down after reaching a value of 0. If the one-shot control bit is set to one the counter will load the “reload value register” and stop counting.
0	tim0_enable	The Cnt_Enable Register controls whether the counter is enabled for counting. When set to one the counter is enabled.

14.4.4 General-Purpose Timer 1

Register Name:	OST_TIM1				
Hex Offset Address:	0xC800 500C		Reset Hex Value:	0x00000000	
Register Description:	General-Purpose Timer 1				
Access: Read.					

31																												0
timer_val																												

Register		OST_TIM1
Bits	Name	Description
31:0	timer_val	Current value of the general-purpose timer

14.4.5 General-Purpose Timer 1 Reload

Register Name:	OST_TIM1_RL				
Hex Offset Address:	0x C800 5010		Reset Hex Value:	0x00000000	
Register Description:	General-Purpose Timer 1 Reload				
Access: Read/Write.					

31																										2	1	0
reload_val																											timer control	

Register		OST_TIM1_RL
Bits	Name	Description
31:2	reload_val	Value loaded into ost_tim1. bits 1 and 0 will be 00. ***Note:*** As the counter counts to zero writing a value of X to these bits will generate interrupts at intervals of (X*4)+1 cycles.
1	tim1_one_shot	The one shot control bit specifies what action should be taken when the general-purpose down counter reaches zero. If the one-shot control bit is set to zero, the counter will load the reload value and continue to count down after reaching a value of zero. If the one-shot control bit is set to one the counter will load the “reload value register” and stop counting.
0	tim1_enable	The cnt_enable register controls whether the counter is enabled for counting. When set to one the counter is enabled.

14.4.6 Watch-Dog Timer

Register Name:	OST_WDOG				
Hex Offset Address:	0x C800 5014		Reset Hex Value:	0xFFFFFFFF	
Register Description:	Watch-Dog Counter				
Access: Read/Write.					

31																												0
tim_val																												

Register		OST_WDOG
Bits	Name	Description
31:0	tim_val	Watch-dog down counter value. A Write to this register has no effect unless ost_wdog_key=0x482E.

14.4.7 Watch-Dog Enable Register

Register Name:	OST_WDOG_ENAB				
Hex Offset Address:	0x C800 5018		Reset Hex Value:	0x00000000	
Register Description:	Watch-Dog Enable Register				
Access: Read/Write.					

31	...	16	15	...	3	2		0
(Reserved)						watch-dog enable bits		

Register		OST_WDOG_ENAB
Bits	Name	Description
31:3		(Reserved). Returns 0 if read
2	wdog_cnt_ena	Watch-dog count enable bit. Logic 1 enables Watch-Dog Timer Down Counter to decrement. A write to this register has no effect unless ost_wdog_key=0x482E
1	wdog_int_ena	Watch-dog Interrupt Enable. Logic 1 enables the watch-dog interrupt signal to be generated when the Watch-Dog Timer Down Counter reaches a value of zero. A write to this register has no effect unless ost_wdog_key=0x482E
0	wdog_rst_ena	Watch-dog Reset Enable. Logic 1 enables the watch-dog timer reset to be generated to the chip when the Watch-Dog Timer Down Counter reaches a value of zero. A write to this register has no effect unless ost_wdog_key=0x482E

14.4.8 Watch-Dog Key Register

Register Name:	OST_WDOG_KEY				
Hex Offset Address:	0x C800 501C		Reset Hex Value:	0x00000000	
Register Description:	Watch-Dog Key Register				
Access: Read/Write.					

31	...	16	15	...	0
(Reserved)			key_value		

Register		OST_WDOG_KEY
Bits	Name	Description
31:16		(Reserved). Returns 0, if read.
15:0	key_value	Reset Key Value. Value of the reset key in which access is allowed to ost_wdog_enab and ost_wdog. The key value is 0x482E.

14.4.9 Timer Status

Register Name:	OST_STATUS				
Hex Offset Address:	0x C800 5020		Reset Hex Value:	0x00000000	
Register Description:	Timer Status Register				
Access: Read/Bit Clear.					

31																							5	4				0
(Reserved)																								ost status bits				

Register		OST_STATUS
Bits	**Name**	**Description**
31:5		(Reserved). Returns 0 if read
4	warm_reset	Logic 1 if a warm reset has occurred. A warm reset is when the watch-dog timer caused the reset to occur. Writing logic 1 to this bit will clear the bit if the bit is set.
3	Ost_wdog_int_val	Logic 1 when a watch-dog timer's interrupt has occurred and the watch-dog timer interrupt enable bit is set. Writing logic 1 to this bit will clear the bit when the condition that caused the bit to be set is no longer present.
2	Ost_ts_int_val	Logic 1 when the time-stamp timer reaches the maximum count value. Writing logic 1 to this bit will clear the bit when the condition that caused the bit to be set is no longer present.
1	Ost_tim1_int_val	Logic 1 when the general-purpose timer's down counter has reached a value of zero. Writing logic 1 to this bit will clear the bit when the condition that caused the bit to be set is no longer present.
0	Ost_tim0_int_val	Logic 1 when the general-purpose timer's down counter has reached a value of zero. Writing logic 1 to this bit will clear the bit when the condition that caused the bit to be set is no longer present.

§ §

15.0 Ethernet MAC A

The functionality supported by the MII Interfaces is tightly coupled with the code written on the Network Processor Engine (NPE) core. This chapter details the full hardware capabilities of the MII Interface contained within the Ethernet coprocessor of the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described below.

Not all of the Intel® IXP42X product line and IXC1100 control plane processors have this functionality. See Table 151.

Table 151. Processors' Devices with Ethernet Interface

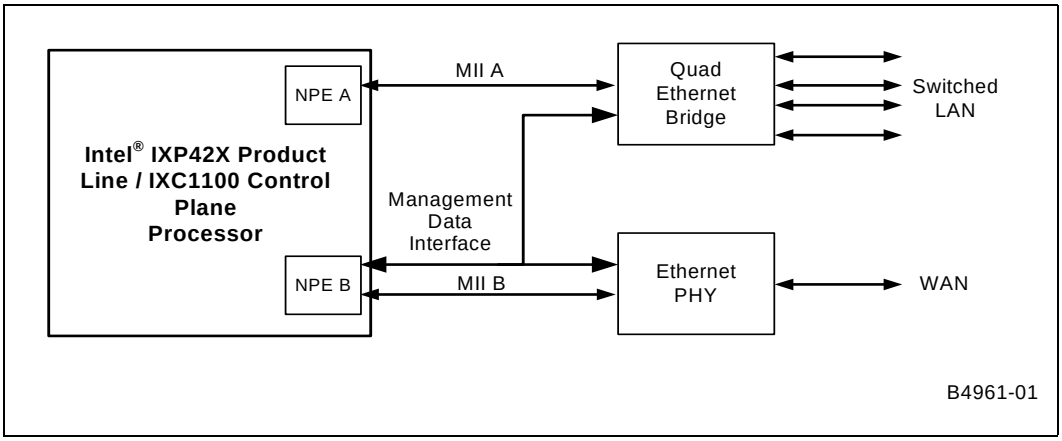
Device	UTOPIA	HSS	MII 0	MII 1	AES / DES / 3DES	Multi-Channel HDLC	SHA-1 / MD-5
IXP425	X	X	X	X	X	8	X
IXP423	X	X	X	X		8	
IXP422			X	X	X		X
IXP421	X	X	X			8	
IXP420			X	X			
IXC1100			X	X			

The IXP42X product line and IXC1100 control plane processors contains three NPEs. Two of the three NPEs are used to process Ethernet traffic utilizing the MII interfaces. Each NPE core used for Ethernet traffic connects to a Transmit FIFO and Receive FIFO through the NPE Coprocessor interface. These Transmit and Receive FIFOs are used to buffer data between the Ethernet Media Access Controller (MAC) and the NPE core.

The Transmit FIFO, Receive FIFO, and MAC are contained in an NPE coprocessor unit called the Ethernet Coprocessor. The MAC contained in the Ethernet coprocessor is compliant to the IEEE 802.3 specification as well as handling flow control for the IEEE 802.3Q VLAN specification.

One Management Data Interface is shared between the two MII interfaces. The single Management Data Interface is used to configure the physical devices attached to each of the MII interfaces. Figure 80 shows a typical application that may be used in connecting to the MII interface.

Figure 80. Multiple Ethernet PHYS Connected to Processor

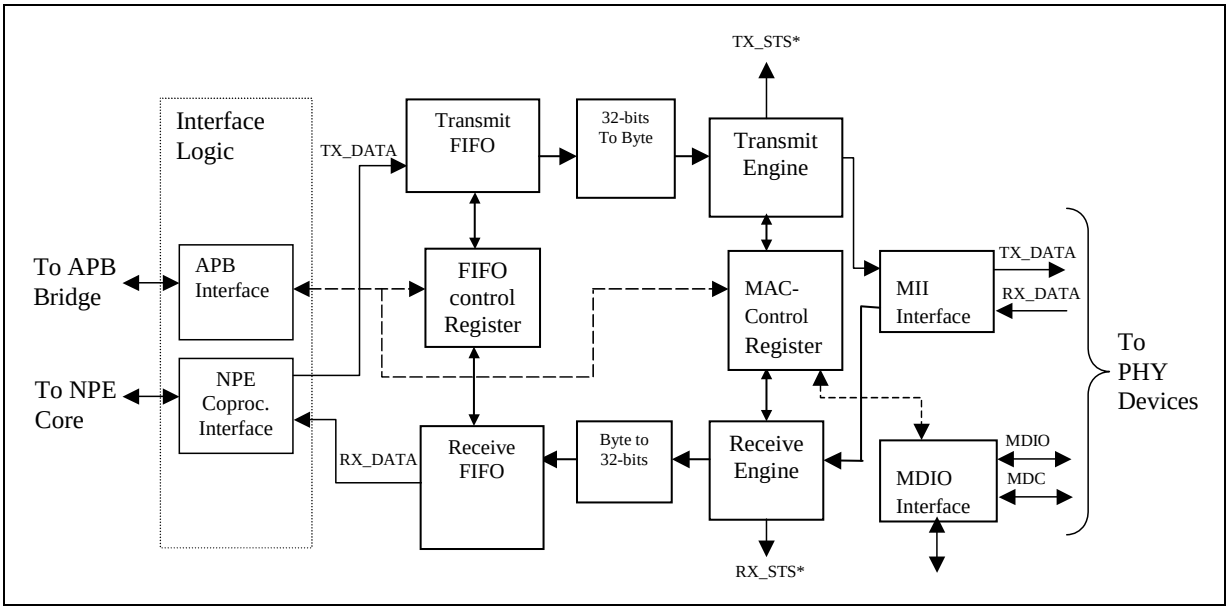


15.1 Ethernet Coprocessor

The Ethernet coprocessor contains a Media Access Controller, a transmit data FIFO, and a receive data FIFO.

Figure 81 displays a block diagram of the Ethernet coprocessor.

Figure 81. Ethernet Coprocessor Interface



The Ethernet Coprocessor communicates to peripheral devices and the remainder of the IXP42X product line and IXC1100 control plane processors over four interfaces:

- APB Interface
- NPE Coprocessor Interface

- MII Interface
- Management Data Interface

15.1.1 Ethernet Coprocessor APB Interface

The APB interface is used to allow the Intel XScale® Processor to communicate directly to configuration and control registers utilized by the Media Access Controller. The Ethernet coprocessor's APB interface will be used to configure the Ethernet MAC, monitor Ethernet status, and configure the physical devices connected via the MII interfaces. The physical devices connected to the MII interface will be configured using the shared Management Data Interface.

15.1.2 Ethernet Coprocessor NPE Interface

The NPE Coprocessor Interface is used to communicate between the Ethernet coprocessor and the NPE core. The NPE coprocessor interface will be used to transfer incoming and outgoing data traffic to and from the NPE core. The NPE core — along with other coprocessor — will take the Ethernet data and perform data manipulation, forward the data to the SDRAM, and update the Queue Manager.

15.1.3 Ethernet Coprocessor MDIO Interface

The Management Data Interface is a two-wire interface that resides in NPE B and supports both MII Interfaces. The Management Data Interface consists of the Management Data Input/Output (MDIO) signal and the Management Data Clock (MDC). The Management Data Input/Output signal is a bi-directional signal that is used to transfer control, configuration, and status information between the IXP42X product line and IXC1100 control plane processors and any peripheral devices connected to the MII interfaces.

The Ethernet Coprocessor is initiated twice to support two Ethernet PHYs outside the device. However, only the MDC and MDIO of one of the coprocessors were brought out and they are used to program both Ethernet PHYs.

The Management Data Clock can be configured as an input or an output, enabling the IXP42X product line and IXC1100 control plane processors to source the Management Data Clock or enable an external device to source the clock. The Management Date Clock is used to clock the data sent on the Management Data Input/Output Signal.

Data transfers will be initiated over the MDIO using the MDIO Command Register (MDIOCMD). The MDIO Command Register is broken into four 8-bit registers that make up a full 32-bit command word.

If data is to be sent to the PHY over the MDIO interface, the Intel XScale processor will write a value to each of the four command words in sequential order:

- MDIO Command 1 (MDIOCMD1) Register and MDIO Command 2 (MDIOCMD2) Register will contain the 16 bits of data that the destination PHY will receive.
- MDIO Command 3 (MDIOCMD3) Register and MDIO Command 4 (MDIOCMD4) Register will determine which PHY number is to be addressed, the internal register of the addressed PHY, the direction of the access (read/write), and when to begin the access.

There can be a limit of 32 physical ports and a limit of 32 registers per physical port that may be addressed.

MDIOCMD3 makes up bits (23:16) of MDIOCMD and MDIOCMD4 make up bits (31:24):

- Bits (25:21) of the MDIO Command (MDIOCMD) Register are used to select the physical interface that is to accept the transmitted data or return the requested data.
- Bits (20:16) of the MDIO Command (MDIOCMD) Register are used to select the register within the physical interface that is to accept the transmitted data or return the requested data.
- Bit 26 of the MDIO Command (MDIOCMD) Register is used to determine if the requested command is a read or a write.
  - — Writing logic 1 to bit 26 of the MDIO Command (MDIOCMD) Register will cause the transaction to be a write.
  - — Writing logic 0 to bit 26 of the MDIO Command (MDIOCMD) Register will cause the transaction to be a read.

Setting Bit 31 of the MDIO Command (MDIOCMD) Register to logic 1 will initiate the transfer. Bit 31 of the MDIO Command (MDIOCMD) will remain at logic 1 until the transaction is complete.

Figure 82 shows an example of the data being written from the MII Management Master (IXP42X product line and IXC1100 control plane processors) to a physical interface (PHY) using the MDIO interface.

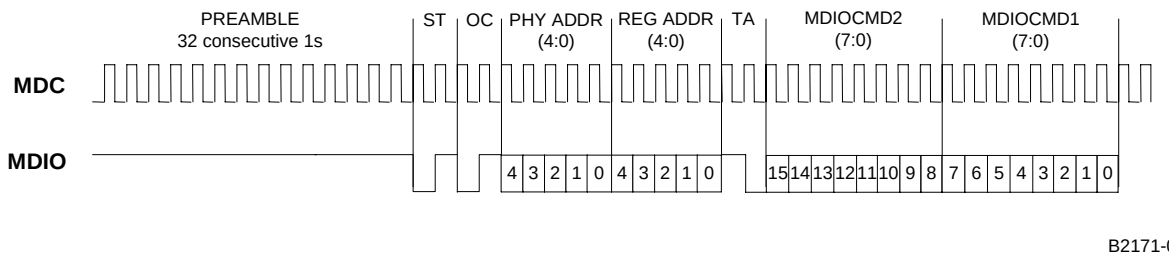
As stated previously, when bit 26 of the MDIO Command (MDIOCMD) Register is set to logic 0 the Intel XScale processor is requesting a read from a physical interface device using the MDIO interface. The data that the physical interface returns from the MDIO signal will be captured in the MDIO STATUS (MDIOSTS) Register.

The MDIO Status Register is broken into four 8-bit registers. The data returned from the physical interface will be captured in MDIO Status 0 (MDIOSTS0) Register and MDIO Status 1 (MDIOSTS1) Register:

- MDIO Status 0 (MDIOSTS0) Register corresponds to bits (7:0) of the MDIO Status (MDIOSTS) Register.
- MDIO Status 1 (MDIOSTS1) Register corresponds to bits (15:8) of the MDIO Status (MDIOSTS) Register.
- Bits (30:16) of the MDIO Status (MDIOSTS) Register are reserved and will return zeros when read.
- Bit 31 of the MDIO Status (MDIOSTS) Register will indicate the condition of the read.
  - — If logic 1 is read from bit 31 of the MDIO Status (MDIOSTS) Register after a read transaction from the physical interface is complete, the read contained an error and should be disregarded.
  - — If logic 0 is read from bit 31 of the MDIO Status (MDIOSTS) Register after a read transaction from the physical interface is complete, the read was valid and error free.

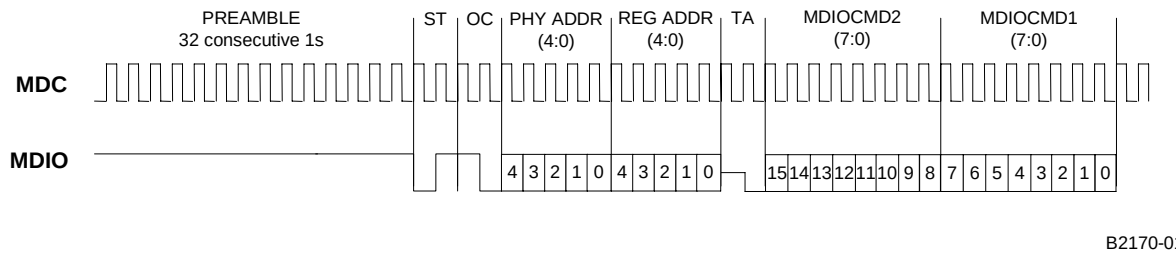
Figure 83 shows an example of the data being read from the physical interface (PHY) by the MII Management Master (IXP42X product line and IXC1100 control plane processors) using the MDIO interface. Manipulation of these registers directly may result in unpredictable behavior. These registers should be manipulated using Intel-supplied APIs.

Figure 82. MDIO Write



Notes:
1. ST (Start Bits) is a signal that is logic 0 followed by logic 1 after a PREAMBLE stage.
2. OC (Op Code) is a two-bit signal that informs the destination PHYs if the current requested transaction is a read or a write. Logic0 followed by logic 1 indicates a write transaction is requested. Logic 1 followed by logic 0 indicates a read transaction.
3. TA (Turn Around) is a two-bit, turn-around time used to allow the control of the MDIO to change directions. For write operations, the TA bits will be logic 1 followed by logic 0. For read transactions, the TA bits will be high-impedance (Z) followed by the selected PHY driving the MDIO signal with logic 0.
4. For write operations, the Management Interface Master will drive the MDIO signal for the duration of the access.
5. For read operation, the Management Interface Master will drive the MDIO signal until the turn around cycle. The PHY will drive the MDIO signal for the second bit of the turn around and the remaining sixteen data bits.

Figure 83. MDIO Read



Notes:
1. ST (Start Bits) is a signal that is logic 0 followed by logic 1 — after a PREAMBLE stage.
2. OC (Op Code) is a two-bit signal that informs the destination PHYs, if the current requested transaction is a read or a write. Logic 0 followed by logic 1 indicates a write transaction is requested. Logic 1 followed by logic 0 indicates a read transaction.
3. TA (Turn Around) is a two-bit, turn-around time used to allow the control of the MDIO to change directions. For write operations, the TA bits will be logic 1 followed by logic 0. For read transactions, the TA bits will be high-impedance (Z) followed by the selected PHY driving the MDIO signal with logic 0.
4. For write operations, the Management Interface Master will drive the MDIO signal for the duration of the access.
5. For read operation, the Management Interface Master will drive the MDIO signal until the turn around cycle. The PHY will drive the MDIO signal for the second bit of the turn around and the remaining sixteen data bits.

15.1.4 Transmitting Ethernet Frames with MII Interfaces

Using IXP42X product line and IXC1100 control plane processors API calls, the Intel XScale processor can request that packets be transmitted by the MII Interface. The Intel XScale processor prepares an Ethernet packet to be transmitted. When the preparation is complete, the Intel XScale processor uses the Intel-supplied API calls to inform the Ethernet NPE that a packet is ready to be transmitted. The Ethernet NPE fetches the packet from the SDRAM attached to the IXP42X product line and IXC1100 control plane processors and forwards the data over the NPE coprocessor interface to the 256-byte Transmit FIFO contained in the Ethernet coprocessor.

Once the data has reached a predefined trigger point — known as the Buffer Size for Transmit Register (TXBUFFSIZE), in the Transmit FIFO — or the End-of-Frame signal is received, a data packet will begin to be transmitted over the MII interface. The Buffer Size for Transmit Register (TXBUFFSIZE), for each frame transmitted, holds the minimum number of bytes that must be contained in the transmit FIFO before the frames transmission may start. If the total size of the frame is less than the Buffer Size for Transmit Register, the frame will be transmitted when the end of frame signal is received.

When entries leave the bottom of the Transmit FIFO, the entries are 32-bits. Setting of the Buffer Size for Transmit Register (TXBUFFSIZE), the Threshold for Partially Full (THRESHPF), and the Threshold for Partially Empty (THRESHPE) are tightly coupled with the code written for the NPE core. Manipulation of the values will result in unpredictable behavior.

The Threshold for Partially Full (THRESHPF) is a parameter that — when the number of entries in the Transmit or Receive FIFOs is larger than the value programmed in this register — a status flag, going to the NPE core, will be set. The Threshold for Partially Full (THRESHPE) is a parameter that — when the number of entries in the Transmit or Receive FIFOs is smaller than the value programmed in this register — a status flag, going to the NPE core, will be set.

After the data begins leaving the FIFO, the data is sent through a converter function that is used to convert the bits from 32-bits to byte-wide entries to supply to the Transmit Engine. The Transmit Engine will take the bytes supplied from the converter, manipulate the data as defined by MAC control registers and forward the data over the MII interface as 4-bit nibbles.

The Transmit Engine can be configured using IXP42X product line and IXC1100 control plane processors' API calls to:

- Append a Frame Check Sequence to the end of a transmitted frame
- Autonomously append bytes to frames that are smaller than the minimum frame size (64 bytes)
- Enable/Disable transmit retries
- Set the number of times a frame can be retried due to collision conditions before being dropped
- Select half- or full-duplex mode of operation
- Select a one- or two-part deferral to be used
- Enable/Disable the Transmit Engine

A Frame Check Sequence (FCS) can be autonomously generated and appended to the end of each Ethernet frame. The Frame Check Sequence can be used to ensure proper delivery of data between two Ethernet devices. The Frame Check Sequence consists of a 4-byte Cyclic Redundancy Generator that adheres to the polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The Frame Check Sequence will be computed over all fields beginning after the Start-of-Frame Delimiter (SFD) and up to the Frame Check Sequence (FCS) value. Autonomous insertion of the Frame-Check Sequence into the transmitted frame can be enabled/disabled by setting bit 4 of Transmit Control Register 1 (TXCTRL1). Setting bit 4 of Transmit Control Register 1 (TXCTRL1) to logic 1 will cause a CRC value to be generated and inserted into the Frame-Check Sequence field of the transmitted frame. Setting bit 4 of Transmit Control Register 1 (TXCTRL1) to logic 0 will cause a transmitted frame to be sent without a Frame Check Sequence attached.

Transmit Control Register 1 can be accessed directly. Intel, however, recommends that the Transmit Control Register 1 values be manipulated through Intel supplied APIs. Failure to use the Intel-supplied APIs will result in unpredictable results.

The Transmit Engine also can be configured to append additional bytes to frames that are smaller than the 64-byte frame minimum. When the transmit engine observes that the length field is smaller than 64 bytes — and bit 3 of the Transmit Control Register 1 (TXCTRL1) is set to logic 1 — the transmit engine will append additional bytes to equal the 64-byte minimum size. A CRC value will be calculated over the appended bytes. The appended bytes will be all zeros.

Transmit Control Register 1 can be accessed directly, but Intel recommends that the Transmit Control Register 1 values be manipulated through Intel-supplied APIs. Failure to use the Intel-supplied APIs will result in unpredictable results.

The IXP42X product line and IXC1100 control plane processors also provide the capability to enable or disable transmit retries. When bit 2 of the Transmit Control Register 1 (TXCTRL1) is set to logic 1 and collisions occur, the transmit engine will attempt to retry sending the packet up to the maximum number of transmit retries specified in bits (3:0) of Transmit Control Register 2 (TXCTRL2). A maximum of 16 retries can be attempted.

Note: Setting bit 1 of the Transmit Control Register 1 (TXCTRL1) to logic 1 configures a device in half-duplex mode of operation. Setting the bit to logic 0 configures the device to full-duplex mode.

If a packet is being transmitted and a collision is detected prior to 64 bytes of the packet (minimum packet size) are transmitted, the transmit engine will rewind the transmit FIFO pointer to the beginning of the frame that is being transmitted and attempt to send the frame after a calculated back-off time, based upon the Slot Time (SLOTTIME) Register.

If a packet is being transmitted and a collision is detected after 64 bytes of a packet (minimum packet size) are transmitted, the transmit engine will forward the transmit FIFO pointer to the beginning of the next packet that is to be transmitted and discard the current packet that is being transmitted.

In a properly configured network, a collision should not occur after the first 64 bytes of a packet is sent. The back-off time algorithm adheres to the IEEE 802.3 specifications “truncated binary exponential algorithm.”

If the MII interface is configured in half-duplex mode of operation, the transmit interface must listen to the activity on the line for a defined wait period called a transmit deferral period. If there is any activity — transmit or receive — on the medium, the transmit interface will defer the transmission of the packet that the transmit engine has been requested to send.

When the transmit engine detects that the medium has gone silent, the transmit engine will continue to defer for a period of time equal to the inter-frame spacing. The deferral period will be assigned by the transmit deferral parameters. The transmit deferral time can be specified as a one-part or an optional two-part deferral by manipulating bit 5 of Transmit Control Register 1 (TXCTRL1).

Setting bits 5 of the Transmit Control Register 1 (TXCTRL1) to logic 1 enables the optional, two-part transmit deferral. Setting bits 5 of the Transmit Control Register 1 (TXCTRL1) to logic 0 enables the one-part transmit deferral.

If the transmit engine has a frame ready to transmit and one-part transmit deferral is selected, the transmit engine will wait for the medium to go silent and then wait for the time period specified in the Transmit Deferral Register (TXDEFPARS). The deferral

period will be the number of transmit clock cycles specified by the 8-bit Transmit Deferral Register minus three transmit clock cycles. The Single Transmit Deferral parameter specifies the Inter Frame Gap.

In the two-part deferral process the deferral period is defined using the Transmit Two Part Deferral Parameters 1 Register (TX2PARTDEFPARS1) and Transmit Two Part Deferral Parameters 2 Register (TX2PARTDEFPARS2). To ensure fair access to the medium, the values specified in these two registers, when added together, should not be less than the minimum Inter Frame Gap.

The Transmit Two Part Deferral Parameters 1 Register is typically set to two thirds of the Inter Frame Gap, and the Transmit Two Part Deferral Parameters 2 Register is typically set to the remaining one third.

If the transmit engine has a frame ready to transmit and two-part transmit deferral is selected, the transmit engine will wait for the medium to go silent and then wait for the time period specified in the Transmit Two Part Deferral Parameters 1 Register. If the medium is not silent during this first part of the deferral then the deferral counter is reset. If the medium is silent during the first part of the deferral then the transmit engine continues to wait for the time period specified in the Transmit Two Part Deferral Parameters 2 Register.

When the MII interface of the IXP42X product line and IXC1100 control plane processors is configured in full-duplex mode of operation, the two-part transmit deferral parameters and the back-off times will not be utilized for data transmission.

For more information on Deference, refer to IEEE 802.3, Section 4.2.3.2.1.

15.1.5 Receiving Ethernet Frames with MII Interfaces

When data is received using the MII interface, the Receive engine will be used to convert the data from 4-bit nibbles into 8-bit bytes. The receive interface will then send the data from a byte to 32-bit word converter. The 32-bit word will be written into the 256-byte receive FIFO.

A flag will be generated to inform the NPE that the receive FIFO has new data to be removed after the Threshold for Partial Full/Empty value has been reached. The Receive Engine implements the following functions:

- Enable/Disable the receive engine
- Implements uni-cast/multi-cast/broadcast, single-address filtering
- Checks for runt frames
- Checks for valid length/type fields
- Removes padded bits added to frames
- Implements the Frame-Check Sequence Algorithm

When new receive data is detected, the Receive Engine will look to see if the address filtering checks are enabled. If the address filtering checks are enable by setting bit 5 of the Receive Control Register 1 (RXCTRL1) to logic 1, the Receive Engine will obtain the destination address from the received frame, check the destination address against the address filtering parameters, and pass the frame to the 8-bit to 32-bit framing engine, if the test passed.

The Receive Engine is capable of filtering broadcast frames, multi-cast frames, and uni-cast frames. The frame filtering for multi-cast frames and uni-cast frames is controlled by registers Address Mask Register (ADDRMASK), the Address Register (ADDR), and the Uni-Cast Register (UNIADDR).

Broadcast frames can be dropped and prevented from being sent to the NPE. To accomplish this, the following three conditions must be met:

1. Broadcast Disable bit "**b7=1**" of the Received Control Register RXCTRL1.
2. Address Mask Registers 1 to 6 are **NOT 00 00 00 00 00 00**.
3. Address Filter Enable bit "**b5=1**" of the Received Control Register RXCTRL1.

Multi-cast frame filtering and uni-cast frame filtering can be enabled/disabled by setting bit 5 of Receive Control Register 1 (RXCTRL1). Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 0 allows all non-broadcast frames to be sent to the NPE. The NPE can then operate in promiscuous mode and implement a more comprehensive filtering algorithm if required.

Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 1 enables filtering for uni-cast frames that are received. When uni-cast frames are detected, the destination address of the received frame must match exactly the value contained in the Uni-Cast Address Register (UNIADDR). Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 0 allows all uni-cast frames to be passed to the NPE.

For example, if the Uni-Cast Address Register — which is broken into six bytes — contains the value of 0x00ABCDEF1234 and uni-cast address filtering is enabled: Any uni-cast frames that are received (determined by bit 47 being set to logic 0) — with anything other than 0x00ABCDEF1234 — will be discarded.

Multi-cast is when frames are sent to multiple destinations from a single source with a single-frame transmission. The Address Mask Register (ADDRMASK) and the Address Register (ADDR) Multi-Cast frames can be used to filter multi-cast frames or frames with common addresses. The address mask is used to tell what each destination address has in common.

For example, bytes 3 and bytes 4 of every address in this group contain the same value. The Address Mask Register (ADDRMASK) would contain a hexadecimal value of 0x00FFFF000000. The Address Mask Register will be a logical "AND" with the Address Register and then a logical "AND" with the destination address. The result of the logical "AND" values will be compared to see if they match. If they match, the frame will be passed to the NPE via the remaining logic.

The values seen were as follows:

- Address Mask Register (ADDRMASK) = 00-FF-FF-00-00-00
- Address Register (ADDR) = 00-C1-D2-38-72-00
- Destination Address = A1-C1-D2-47-63-21

Notice that the underlined values are all that matters for the comparison because of the address mask. The frame in this example would be forwarded to the next phase of the receive logic. An example of a frame that would be dropped due to address filtering follows:

- Address Mask Register (ADDRMASK) = 00-FF-FF-00-00-00
- Address Register (ADDR) = 00-C1-D2-38-72-00
- Destination Address = A1-C1-D3-47-63-21

To disable the multi-cast address filtering feature set the Address Mask Register to all logic 0s.

After the received frame has passed all address filtering checks, the Receive Engine will capture the length/type field. If the length/type is determined to be a length value field and the length is less than 64 bytes, the Receive Engine will remove any padded bytes

(assuming bit 1 of Receive Control Register 1 is set to logic 1) and capture the remaining data. Padded bytes will not be removed from the received packet when bit 1 of Receive Control Register is set to logic 0.

If the packet is less than 64 bytes and there are no padded bytes, the packet is determined to be a runt frame. The Receive Engine has the capability to discard these frames or forward the frames to the NPE via the remaining receive logic.

Setting bit 6 of Receive Control Register 1 (RXCTRL1) to logic 1 informs the Receive Engine to allow the packet to be sent to the NPE. Setting bit 6 of Receive Control Register 1 (RXCTRL1) to logic 0 informs the Receive Engine to terminate the reception of the runt packet and purge the runt packet from the rest of the receive logic.

If the length/type is determined to be a type field, the field is looked at for validity. Invalid frames are purged from the receive logic.

Once the received frame has passed the frame validity checks, the received frame will be checked for integrity using the Frame-Check Sequence Algorithm called out in the transmit-frame section. If the frame passes the Frame Check Sequence, the frame will be forwarded on to the NPE via the remaining receive logic. If the frame fails the Frame Check Sequence, the frame will be discarded along with purging all of the remaining receive logic of the frames contents.

Padded bytes will be included in the calculation of the Receive Engine's Frame Check Sequence for frames that were transmitted and were smaller than the 64-byte minimum frame size. A status flag will be sent to the NPE to inform the NPE what to do with the received frame.

In addition to the above features, the MII receive interface allows some features to be used for test and debug. The transmit interface can be looped back to the receive interface by setting bit 4 of Receive Control Register 1 (RXCTRL1) to logic 1. In loop-back mode, the Receive Engine will receive all of the data sent by the Transmit Engine.

The loop-back feature allows software developers to develop their application code and test the code before they start dealing with physical interface problems. Setting bit 0 of Receive Control Register 1(RXCTRL1) to logic 1 enables the Receive Engine. This feature allows all initialization of the product to occur prior to bring the receive interface online. Bit 0 of Receive Control Register 2 (RXCTRL2) enables deferral checking on the receive side. The IXP42X product line and IXC1100 control plane processors do not use the receive side deferral checking feature.

Bit 0 of Receive Control Register 2 (RXCTRL2) must be set to logic 0 for proper operation. Failure to do so will result in unpredictable behavior.

Intel recommends that the register values described in this section be manipulated through Intel-supplied APIs. Failure to use the Intel supplied APIs will result in unpredictable results.

15.1.6 General Ethernet Coprocessor Configuration

The Ethernet coprocessor contains various other registers that are used to configure the interface. Some of these registers are included due to the generic nature of the Ethernet coprocessor. Other registers are added to allow greater flexibility in configuration of the Ethernet coprocessor.

The Threshold for Internal Clock (THRESH_INTCLK) Register is used to determine the frequency relationship between the MII interface and the host processor that is used to control the MII interface. The value in the Threshold for Internal Clock (THRESH_INTCLK) Register will be manipulated based upon the ratio of PHY_CLK_SPEED/HOST_CLK_SPEED.

The physical interface clock speed will be divided by the host-side clock speed and then rounded to the nearest whole number. The value from this calculation will be written to the Threshold for Internal Clock (THRESH_INTCLK) Register. For the IXP42X product line and IXC1100 control plane processors, the value contained in the Threshold for Internal Clock (THRESH_INTCLK) Register must always be set to hexadecimal 0x01. Failure to do so will result in unpredictable behavior. The value of the Threshold for Internal Clock (THRESH_INTCLK) Register will be pre-programmed by the Intel-supplied APIs to the proper value. Manipulation of this value will result in unpredictable behavior.

The Core Control (CORE_CONTROL) Register was added to allow the Intel XScale processor to have some control over the Ethernet coprocessor. Configuring bit 4 of the Core Control (CORE_CONTROL) Register can configure as an input or as an output the MDC clock.

Setting bit 4 of the Core Control (CORE_CONTROL) Register to logic 1 enables the IXP42X product line and IXC1100 control plane processors to drive the MDC clock. Setting bit 4 of the Core Control (CORE_CONTROL) Register to logic 0 enables an external source to drive the MDC clock. The IXP42X product line and IXC1100 control plane processors becomes a recipient of that MDC clock.

Configuring bit 3 of the Core Control (CORE_CONTROL) Register can configure the Transmit Engine to send a JAM sequence if a new packet starts to be received. Setting bit 3 of the Core Control (CORE_CONTROL) Register to logic 1 enables the IXP42X product line and IXC1100 control plane processors to send a JAM sequence if a new packet is receive.

Setting bit 3 of the Core Control (CORE_CONTROL) Register to logic 0 allows the Transmit Engine to function in normal mode of operation. Configuring bit 2 of the Core Control (CORE_CONTROL) Register causes the Transmit FIFO to be flushed. Any packets that are currently in the Transmit FIFO are discarded.

Setting bit 2 of the Core Control (CORE_CONTROL) Register to logic 1 clears the MII Interface Transmit FIFO. Setting bit 2 of the Core Control (CORE_CONTROL) Register back to logic 0 allows the Transmit FIFO to resume normal mode of operation.

Configuring bit 1 of the Core Control (CORE_CONTROL) Register causes the Receive FIFO to be flushed. Any packets that are currently in the Receive FIFO are discarded. Setting bit 1 of the Core Control (CORE_CONTROL) Register to logic 1 clears the MII Interface Receive FIFO. Setting bit 1 of the Core Control (CORE_CONTROL) Register back to logic 0 allows the Receive FIFO to resume normal mode of operation. Bit 0 of the Core Control (CORE_CONTROL) Register controls the reset state of the Media Access Controller (MAC) contained within the Ethernet coprocessor.

Setting bit 0 of the Core Control (CORE_CONTROL) Register back to logic 1 causes the Media Access Controller to be reset. De-assertion (setting bit 0 to logic 0) allows the Media Access Controller to resume operation in a fully reset state.

This register needs to be manipulated using Intel-supplied APIs. Failure to manipulate this register with Intel-supplied APIs will result in unpredictable behavior.

The Random-Seed Register is an 8-bit register used to support PHYs that support Auto MDI/MDI-X detection. The Random-Seed Register value is the value that is used to feed the Linear-Feedback Shift Register that is used to select which configuration to start initialization.

The Random-Seed Register needs to be manipulated using Intel-supplied APIs. Failure to manipulate this could result in unpredictable behavior.

15.2 Register Descriptions

The internal registers shown below are accessible via the APB bus interface. Unspecified addresses are reserved and should not be written; if read, a zero value will be returned.

All of the Ethernet internal configuration and control registers are directly readable and writeable by the Intel XScale processor via APB bus.

Address	Description
0xC800 9000	Transmit Control 1
0xC800 9004	Transmit Control 2
0xC800 9010	Receive Control 1
0xC800 9014	Receive Control 2
0xC800 9020	Random Seed
0xC800 9030	Threshold For Partial Empty
0xC800 9038	Threshold For Partial Full
0xC800 9040	Buffer Size For Transmit
0xC800 9050	Transmit Single Deferral Parameters
0xC800 9054	Receive Deferral Parameters
0xC800 9060	Transmit Two Part Deferral Parameters 1
0xC800 9064	Transmit Two Part Deferral Parameters 2
0xC800 9070	Slot Time
0xC800 9080	Reserved
0xC800 9084	Reserved
0xC800 9088	Reserved
0xC800 908C	Reserved
0xC800 9090	Reserved
0xC800 9094	Reserved
0xC800 9098	Reserved
0xC800 909C	Reserved
0xC800 90A0	Address Mask 1
0xC800 90A4	Address Mask 2
0xC800 90A8	Address Mask 3
0xC800 90AC	Address Mask 4
0xC800 90B0	Address Mask 5
0xC800 90B4	Address Mask 6
0xC800 90C0	Address 1
0xC800 90C4	Address 2
0xC800 90C8	Address 3
0xC800 90CC	Address 4
0xC800 90D0	Address 5
0xC800 90D4	Address 6
0xC800 90E0	Threshold For Internal Clock
0xC800 90F0	Unicast Address 1

Address	Description
0xC800 90F4	Unicast Address 2
0xC800 90F8	Unicast Address 3
0xC800 90FC	Unicast Address 4
0xC800 9100	Unicast Address 5
0xC800 9104	Unicast Address 6
0xC800 91FC	Core Control

15.2.1 Transmit Control 1

Register Name:	txctrl1				
Hex Offset Address:	0xC8009000		Reset Hex Value:	0x00000000	
Register Description:	Transmit Control Register One				
Access: **Read / Write**.					

31	...	7	6	5	4	3	2	1	0
(Reserved)			MII CFG	2PRT DEF	APP FCS	PAD EN	RET EN	HALFDUP	TX EN

Register		txctrl1
Bits	Name	Description
31:7	(Reserved)	
6	MII config	0 = Configures the PHY interface as a MII.
5	Two-part deferral	1 = Causes the optional two part deferral to be used.
4	Append FCS	1 = Causes FCS to be computed and appended to transmit frames before they are sent to the PHY.
3	Pad enable	1 = Causes transmit frames less than to minimum frame size to be padded before they are sent to the PHY.
2	Retry enable	1 = Causes transmit frames to be retried until the maximum retry limit shown in the Transmit Control 2 Register is reached, when collisions occur.
1	Half duplex	1 = Half-duplex operation 0 = Full-duplex
0	Transmit enable	1 = Causes transmission to be enabled.

15.2.2 Transmit Control 2

Register Name:	txcrtl2				
Hex Offset Address:	0xC8009004		Reset Hex Value:	0x00000000	
Register Description:	Transmit Control Register Two				
Access: Read/Write.					

31	...	4	3			0
(Reserved)			Maximum Retries			

Register		txcrtl2
Bits	**Name**	**Description**
4:31	(Reserved)	
3:0	Maximum retries	Maximum number of retries for a packet when collisions occur.

15.2.3 Receive Control 1

Register Name:	rxctrl1				
Hex Offset Address:	0xC8009010		Reset Hex Value:	0x00000000	
Register Description:	Receive Control Register One				
Access: Read/Write.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			BCDIS	RX RP	ADD FILT	LOOP EN	PSE EN	CRC	PAD STRP	RX EN

Register		rxcrtl1 (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8	(Reserved)	
7	Broadcast disable	1 = Prevents broadcast packets from being passed to the application logic.
6	Receive runt packet	1 = Causes runt packets to be passed to the application logic. 0 = Runt packets are dropped.
5	Address filter enable	1 = Causes address filtering to take place. Non-broadcast packets are only passed to the application logic if they pass the address filter.
4	Loopback enable	1 = Causes loop-back operation. ***Note:*** In order for the loop-back operation to operate correctly, the Ethernet coprocessor requires synchronous and in-phase clocks to be provided on the rx_clk and tx_clk pins.
3	Pause enable	1 = Enables detection of Pause frames. Upon detecting a pause frame, data transmission is halted based on the data in the pause frame. The 2-byte data of the received pause frame indicates the time, as a number of 512 bit times, to halt the transmission.
2	Send CRC	1 = Causes the CRC data to be sent to the application logic.

Register		rxcrtl1 (Sheet 2 of 2)
Bits	**Name**	**Description**
1	Pad strip	1 = Causes the pad bytes to be stripped from receive data.
0	Receive enable	1 = Causes reception to be enabled.

Notes: Broadcast packets will only be dropped if the following three conditions are met.
1. "Broadcast Disable" bit,"**b7=1**".
2. The Address M ask register is **NOT 00 00 00 00 00 00**.
3. "Address Filter Enable" bit, "**b5=1**".

Setting the address mask register to all 00 (i.e., don't care about the address) and setting the Broadcast Disable bit to "**b7=1**" will not prevent packets from being accepted. Broadcast packets will continue to be received. All three conditions shown above must be satisfied to stop receiving packets.

15.2.4 Receive Control 2

Register Name:	rxctrl2				
Hex Offset Address:	0xC8009014		Reset Hex Value:	0x00000000	
Register Description:	Receive Control Register Two				
Access: Read/Write.					

31																												1	0
(Reserved)																													RX DEF EN

Register		rxctrl2
Bits	**Name**	**Description**
31:1	(Reserved)	
0	Receive deferral enable	1 = Enables receive deferral checking.

15.2.5 Random Seed

Register Name:	rndmseed				
Hex Offset Address:	0xC8009020		Reset Hex Value:	0x00000000	
Register Description:	Random Seed Register				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							Random Seed							

Register		rndmseed
Bits	**Name**	**Description**
31:8	(Reserved)	
7:0	Random Seed	Random seed used for LFSR initialization in the back-off block.

15.2.6 Threshold For Partially Empty

Register Name:	threshpe				
Hex Offset Address:	0xC8009030		Reset Hex Value:	0x00000000	
Register Description:	FIFO Partially Full/Empty Threshold Register. The threshold is the number of 32-bit words in the FIFO.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							Partial Empty							

Register		threshpe
Bits	Name	Description
31:8	(Reserved)	
7:0	Partial Empty	Marks the partial empty thresholds of the Transmit FIFO and Receive FIFO. When the number of entries in the Transmit FIFO is less than or equal to the contents of this register, tx_fifo_p_empty is asserted. When the number of entries in the Receive FIFO is less than or equal to the contents of this register, rx_fifo_p_empty is asserted.

15.2.7 Threshold For Partially Full

Register Name:	threshpf				
Hex Offset Address:	0xC8009038		Reset Hex Value:	0x00000000	
Register Description:	FIFO Partially Full/Empty Threshold Register. The threshold is the number of 32-bit words in the FIFO.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							Partial Full							

Register		threshpf
Bits	Name	Description
31:8	(Reserved)	
7:0	Partial Full	Marks the partial full thresholds of the Transmit FIFO and Receive FIFO. When the number of entries in the Transmit FIFO is greater than or equal to the contents of this register, tx_fifo_p_full is asserted. When the number of entries in the Receive FIFO is greater than or equal to the contents of this register, rx_fifo_p_full is asserted.

15.2.8 Buffer Size For Transmit

Register Name:	txbuffsize				
Hex Offset Address:	0xC8009040		Reset Hex Value:	0x00000000	
Register Description:	Transmit Buffer Size Register				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							Tx Buffer Size							

Register		txbuffsize
Bits	Name	Description
31:8	(Reserved)	
7:0	Tx Buffer size	Holds minimum number of bytes of each frame that must be in the Transmit FIFO for that frame's transmission to start. If a complete frame is less than this minimum, it is always transmitted.

15.2.9 Transmit Deferral Parameters

Register Name:	txdefpars				
Hex Offset Address:	0xC8009050		Reset Hex Value:	0x00000000	
Register Description:	Transmit Deferral Parameters Register				
Access: Read/Write.					

31																					8	7							0
(Reserved)																						Receive Deferral							

Register		txdefpars
Bits	Name	Description
31:8	(Reserved)	
7:0	Transmit Deferral	Number of transmit clock cycles (tx_clk) in the transmit deferral period minus three, when single deferral is used for transmission (Transmit Control[15] = 0).

15.2.10 Receive Deferral Parameters

Register Name:	rxdefpars				
Hex Offset Address:	0xC8009054		Reset Hex Value:	0x00000000	
Register Description:	Receive Deferral Parameters Register				
Access: Read/Write.					

31																					8	7							0
(Reserved)																						Receive Deferral							

Register		rxdefpars
Bits	Name	Description
31:8	(Reserved)	
7:0	Receive Deferral	Number of receive clock cycles (rx_clk) in the receive deferral period minus three, when checking the Inter Frame Gap for packets received (Receive Control 2[0] = 0).

15.2.11 Transmit Two Part Deferral Parameters 1

Register Name:	tx2partdefpars1				
Hex Offset Address:	0xC8009060		Reset Hex Value:	0x00000000	
Register Description:	Transmit Two Part Deferral Parameters Register 1.				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					First Deferral Period							

Register		tx2partdefpars1
Bits	Name	Description
31:8	(Reserved)	
7:0	First Deferral Period	Number of transmit clock cycles (tx_clk) in the first deferral period minus three, when two-part deferral is used for transmission (Transmit Control 1[5] = 1) and half-duplex mode.

15.2.12 Transmit Two Part Deferral Parameters 2

Register Name:	tx2partdefpars2				
Hex Offset Address:	0xC8009064		Reset Hex Value:	0x00000000	
Register Description:	Transmit Two Part Deferral Parameters Register 2.				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					Second Deferral Period							

Register		tx2partdefpars2
Bits	Name	Description
31:8	(Reserved)	
7:0	Second Deferral Period	Number of transmit clock cycles (tx_clk) in the second deferral period minus three, when two-part deferral is used for transmission (Transmit Control 1[5] = 1) and half-duplex mode.

15.2.13 Slot Time

Register Name:	slottime				
Hex Offset Address:	0xC8009070		Reset Hex Value:	0x00000000	
Register Description:	Slot Time Register				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					Slot Time							

Register		slottime
Bits	**Name**	**Description**
31:8	(Reserved)	
7:0	Slot Time	Slot time for back-off algorithm Expressed in number of tx_clk cycles. 128 in MII mode.

15.2.14 MDIO Commands Registers

Four registers make up the 32-bit MDIO Command that services both MII interfaces:

- MDIO Command[31:24] — MDIO Command 4
- MDIO Command[23:16] — MDIO Command 3
- MDIO Command[15:8] — MDIO Command 2
- MDIO Command[7:0] — MDIO Command 1

The detailed bit descriptions follow the four commands' bit maps.

15.2.15 MDIO Command 1

Register Name:	mdiocmd1				
Hex Offset Address:	0xC8009080		Reset Hex Value:	0x00000000	
Register Description:	MDIO Command 1 (8 Bits of 32-Bit Register).				
Access: Read/Write.					

7	6	5	4	3	2	1	0
MDIO Command [7:0]							

15.2.16 MDIO Command 2

Register Name:	mdiocmd2				
Hex Offset Address:	0x C8009084		Reset Hex Value:	0x00000000	
Register Description:	MDIO Command Register				
Access: Read/Write.					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																								MDIO_COMMAND [15:8]							

15.2.17 MDIO Command 3

Register Name:	mdiocm3				
Hex Offset Address:	0x C8009088		Reset Hex Value:	0x00000000	
Register Description:	MDIO Command Register				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					MDIO_COMMAND [23:16]							

15.2.18 MDIO Command 4

Register Name:	mdiocm4				
Hex Offset Address:	0x C800908C		Reset Hex Value:	0x00000000	
Register Description:	MDIO Command Register				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					MDIO_COMMAND[31:24]							

Register	MDIO Command	
Bits	**Name**	**Description**
31	Go	Application logic sets this to 1 to start the MDIO access. This bit remains 1 during the access. When the access is finished, the Ethernet core resets this bit to 0.
30:27	(Reserved)	
26	MDIO Write	1 = MDIO write access 0 = MDIO read access.
25:21	PHY address	Physical address of the PHY to be accessed.
20:16	PHY Register	Register number of the PHY Register to be accessed.
15:0	Write data	Write data on MDIO write accesses.

15.2.19 MDIO Status Registers

Four registers make up the 32-bit MDIO status:

- MDIO Status[31:24] — MDIO Status 4
- MDIO Status[23:16] — MDIO Status 3
- MDIO Status[15:8] — MDIO Status 2
- MDIO Status[7:0] — MDIO Status 1

The detailed bit descriptions follow the four registers' bit maps.

15.2.20 MDIO Status 1

Register Name:	mdiosts1				
Hex Offset Address:	0x C8009090		Reset Hex Value:	0x00000000	
Register Description:	MDIO Status Register				
Access: Read Only.					

31	...	8	7	...	0
(Reserved)			MDIO_STATUS[7:0]		

15.2.21 MDIO Status 2

Register Name:	mdiosts2				
Hex Offset Address:	0x C8009094		Reset Hex Value:	0x00000000	
Register Description:	MDIO Status Register				
Access: Read Only.					

31	...	8	7	...	0
(Reserved)			MDIO_STATUS[15:8]		

15.2.22 MDIO Status 3

Register Name:	mdiosts3				
Hex Offset Address:	0x C8009098		Reset Hex Value:	0x00000000	
Register Description:	MDIO Status Register				
Access: Read Only.					

31	...	8	7	...	0
(Reserved)			MDIO_STATUS[23:16]		

15.2.23 MDIO Status 4

Register Name:	mdiosts4				
Hex Offset Address:	0x C800909C		Reset Hex Value:	0x00000000	
Register Description:	MDIO Status Register				
Access: Read Only.					

31	...	8	7	...	0
(Reserved)			MDIO_STATUS[31:24]		

Register		MDIO Status
Bits	**Name**	**Description**
31	Successful read	0 = A successful read 1= A read error. Read only.
30:16	(Reserved)	Read only.
15:0	Read data	Read only.

15.2.24 Address Mask Registers

Six registers make up the 48-bit Address Mask:

- Address Mask[47:40] — Address Mask 1
- Address Mask[39:32] — Address Mask 2
- Address Mask[31:24] — Address Mask 3
- Address Mask[23:16] — Address Mask 4
- Address Mask[15:8] — Address Mask 5
- Address Mask[7:0] — Address Mask 6

Example: Address Mask is 00-A0-24-D1-7F-02

- Address Mask 1 = 0x00
- Address Mask 2 = 0x00
- Address Mask 3 = 0x00
- Address Mask 4 = 0xFF
- Address Mask 5 = 0xFF
- Address Mask 6 = 0x00

The detailed bit descriptions follow the six registers' bit maps.

15.2.25 Address Mask 1

Register Name:	**addrmask1**																															
Hex Offset Address:	0x C80090A0													**Reset Hex Value:**	0x00000000															
Register Description:	Address Mask Register #1. First register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted																															
Access: Read/Write.																																
31																						**8**	**7**								**0**	
(Reserved)																							ADDRESS MASK[7:0]									

15.2.26 Address Mask 2

Register Name:	addrmask2				
Hex Offset Address:	0x C80090A4		Reset Hex Value:	0x00000000	
Register Description:	Address Mask Register #1. Second register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS MASK[15:8]							

15.2.27 Address Mask 3

Register Name:	addrmask3				
Hex Offset Address:	0x C80090A8		Reset Hex Value:	0x00000000	
Register Description:	Address Mask Register #1. Third register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS MASK [23:16]							

15.2.28 Address Mask 4

Register Name:	addrmask4				
Hex Offset Address:	0x C80090AC		Reset Hex Value:	0x00000000	
Register Description:	Address Mask Register #1. Forth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS MASK[31:24]							

15.2.29 Address Mask 5

Register Name:	addrmask5				
Hex Offset Address:	0x C80090B0		Reset Hex Value:	0x00000000	
Register Description:	Address Mask Register #1. Fifth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS MASK [39:32]							

15.2.30 Address Mask 6

Register Name:	addrmask6				
Hex Offset Address:	0x C80090B4		Reset Hex Value:	0x00000000	
Register Description:	Address Mask Register #1. Sixth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					ADDRESS MASK [47:40]							

Register		Address Mask
Bits	Name	Description
47:0	Address Mask1-6	Address Mask 1-6 is used with Address 1-6 for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.

15.2.31 Address Registers

Six registers that make up the 48 bit Address Mask are:

- Address Mask[47:40] — Address Mask 1
- Address Mask[39:32] — Address Mask 2
- Address Mask[31:24] — Address Mask 3
- Address Mask[23:16] — Address Mask 4
- Address Mask[15:8] — Address Mask 5
- Address Mask[7:0] — Address Mask 6

Example: Address Mask is 00-A0-24-D1-7F-02

- Address Mask 1 = 0x00
- Address Mask 2 = 0x00
- Address Mask 3 = 0x00
- Address Mask 4 = 0xFF
- Address Mask 5 = 0xFF
- Address Mask 6 = 0x00

The detailed bit descriptions follow the six registers' bit maps.

15.2.32 Address 1

Register Name:	addr1				
Hex Offset Address:	0x C80090C0		Reset Hex Value:	0x00000000	
Register Description:	Address Register #1. First register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS[7:0]								

15.2.33 Address 2

Register Name:	addr2				
Hex Offset Address:	0x C80090C4		Reset Hex Value:	0x00000000	
Register Description:	Address Register #1. Second register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS[15:8]								

15.2.34 Address 3

Register Name:	addr3				
Hex Offset Address:	0x C80090C8		Reset Hex Value:	0x00000000	
Register Description:	Address Register #1. Third register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS[23:16]								

15.2.35 Address 4

Register Name:	addr4				
Hex Offset Address:	0x C80090CC		Reset Hex Value:	0x00000000	
Register Description:	Address Register #1. Forth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																						8	7							0
(Reserved)																							ADDRESS[31:24]								

15.2.36 Address 5

Register Name:	addr5				
Hex Offset Address:	0x C80090D0		Reset Hex Value:	0x00000000	
Register Description:	Address Register #1. Fifth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					ADDRESS[39:32]							

15.2.37 Address 6

Register Name:	addr6				
Hex Offset Address:	0x C80090D4		Reset Hex Value:	0x00000000	
Register Description:	Address Register #1. Sixth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					ADDRESS[47:40]							

Six registers that make up the 48 bit Address are:

- Address[47:40] — Address 1
- Address[39:32] — Address 2
- Address[31:24] — Address 3
- Address[23:16] — Address 4
- Address[15:8] — Address 5
- Address[7:0] — Address 6

Example: Address is 00-A0-24-D1-7F-02

- Address 1 = 0x00
- Address 2 = 0xA0
- Address 3 = 0x24
- Address 4 = 0xD1
- Address 5 = 0x7F
- Address 6 = 0x02

Register		
Bits	Name	Description
47:0	Address	Address Mask 1-6 is used with Address 1-6 for multi-cast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.

15.2.38 Threshold for Internal Clock

Register Name:	thresh_intclk				
Hex Offset Address:	0xC80090E0		Reset Hex Value:	0x00000000	
Register Description:	Threshold for Internal Clock Register				
Access: Read/Write.					

31																					8	7							0
(Reserved)																						CLOCK RATIO							

Register		thresh_intclk
Bits	Name	Description
31:8	(Reserved)	
7:0	Clock ratio	Holds ratio of PHY side clock (tx_clk or rx_clk) to sys_clk. PHY side clock frequency/application clock frequency. Round up for value to be written to this register. Example: 25 MHz PHY clock, 133 MHz application clock — Sets this register to 1. Any application, clock frequency greater than the tx_clk or rx_clk frequency will have a clock ratio of 1. **Always set to 1 for the** Intel® IXP42X product line and IXC1100 control plane processors**.**

15.2.39 Unicast Address Registers

Six registers that make up the 48 bit Unicast Address are:

- Unicast Address[47:40] — Unicast Address 1
- Unicast Address[39:32] — Unicast Address 2
- Unicast Address[31:24] — Unicast Address 3
- Unicast Address[23:16] — Unicast Address 4
- Unicast Address[15:8] — Unicast Address 5
- Unicast Address[7:0] — Unicast Address 6

Example: Unicast Address is 00-A0-24-D1-7F-02

- Unicast Address 1 = 0x00
- Unicast Address 2 = 0xA0
- Unicast Address 3 = 0x24
- Unicast Address 4 = 0xD1
- Unicast Address 5 = 0x7F
- Unicast Address 6 = 0x02

The detailed bit descriptions follow the six registers' bit maps.

15.2.40 Unicast Address 1

Register Name:	uniaddr1				
Hex Offset Address:	0x C80090F0		Reset Hex Value:	0x00000000	
Register Description:	Unicast Address Register #1. First register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.				
Access: Read/Write.					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																								UNICAST ADDRESS[7:0]							

15.2.41 Unicast Address 2

Register Name:	uniaddr2				
Hex Offset Address:	0x C80090F4		Reset Hex Value:	0x00000000	
Register Description:	Unicast Address Register #1. Second register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.				
Access: Read/Write.					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																								UNICAST ADDRESS[15:8]							

15.2.42 Unicast Address 3

Register Name:	uniaddr3				
Hex Offset Address:	0x C80090F8		Reset Hex Value:	0x00000000	
Register Description:	Unicast Address Register #1. Third register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.				
Access: Read/Write.					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																								UNICAST ADDRESS[23:16]							

15.2.43 Unicast Address 4

Register Name:	uniaddr4				
Hex Offset Address:	0x C80090FC		Reset Hex Value:	0x00000000	
Register Description:	Unicast Address Register #1. Forth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.				
Access: Read/Write.					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																								UNICAST ADDRESS[31:24]							

15.2.44 Unicast Address 5

Register Name:	uniaddr5				
Hex Offset Address:	0x C8009100		Reset Hex Value:	0x00000000	
Register Description:	Unicast Address Register #1. Fifth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.				
Access: Read/Write.					

31	…	8	7	…	0
(Reserved)			UNICAST ADDRESS[39:32]		

15.2.45 Unicast Address 6

Register Name:	uniaddr6				
Hex Offset Address:	0x C8009104		Reset Hex Value:	0x00000000	
Register Description:	Unicast Address Register #1. Sixth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE				
Access: Read/Write.					

31	…	8	7	…	0
(Reserved)			UNICAST ADDRESS[47:40]		

Register		Unicast Address
Bits	Name	Description
47:0		Unicast Address 1-6 are matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.

15.2.46 Core Control

Register Name:	core_control				
Hex Offset Address:	0xC80091FC		Reset Hex Value:	0x00000000	
Register Description:	Controls key functions of the core				
Access: Read/Write.					

31	…	5	4	3	2	1	0
(Reserved)			mdc en	send jam	clr tx err	clr rx err	rst_mac

Register		core_control
Bits	**Name**	**Description**
31:5	(Reserved)	
4	Mdc_en	1 = Configures the MDC as an output clock. **Set to 1 for the** Intel® IXP42X product line and IXC1100 control plane processors **MAC A.** This bit is reserved on **MAC B.**
3	Send_jam	1 = Causes a jam sequence to be sent if reception of a packet begins.
2	clr_tx_err	Assertion (“1”) causes the Transmit FIFO to be flushed. Data in the Transmit FIFO is discarded.
1	clr_rx_err	Assertion (“1”) causes the Receive FIFO to be flushed. Data in the Receive FIFO is discarded.
0	rst_mac	Assertion (“1”) causes the MAC to be reset.

§ §

16.0 Ethernet MAC B

Not all of the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor have this functionality. See Table 152.

Table 152. Processors' with Ethernet Interface

Device	UTOPIA	HSS	MII 0	MII 1	AES / DES / 3DES	Multi-Channel HDLC	SHA-1 / MD-5
IXP425	X	X	X	X	X	8	X
IXP423	X	X	X	X		8	
IXP422			X	X	X		X
IXP421	X	X	X			8	
IXP420			X	X			
IXC1100			X	X			

Table 153. Ethernet MAC B Registers (Sheet 1 of 2)

Address	Description
0xC800 A000	Transmit Control 1
0xC800 A004	Transmit Control 2
0xC800 A010	Receive Control 1
0xC800 A014	Receive Control 2
0xC800 A020	Random Seed
0xC800 A030	Threshold For Partial Empty
0xC800 A038	Threshold For Partial Full
0xC800 A040	Buffer Size For Transmit
0xC800 A050	Transmit Single Deferral Parameters
0xC800 A054	Receive Deferral Parameters
0xC800 A060	Transmit Two Part Deferral Parameters 1
0xC800 A064	Transmit Two Part Deferral Parameters 2
0xC800 A070	Slot Time
0xC800 C080	MDIO Command 1†
0xC800 C084	MDIO Command 2†
0xC800 C088	MDIO Command 3†

Table 153. Ethernet MAC B Registers (Sheet 2 of 2)

Address	Description
0xC800 C08C	MDIO Command 4†
0xC800 C090	MDIO Status 1†
0xC800 C094	MDIO Status 2†
0xC800 C098	MDIO Status 3†
0xC800 C09C	MDIO Status 4†
0xC800 A0A0	Address Mask 1
0xC800 A0A4	Address Mask 2
0xC800 A0A8	Address Mask 3
0xC800 A0AC	Address Mask 4
0xC800 A0B0	Address Mask 5
0xC800 A0B4	Address Mask 6
0xC800 A0C0	Address 1
0xC800 A0C4	Address 2
0xC800 A0C8	Address 3
0xC800 A0CC	Address 4
0xC800 A0D0	Address 5
0xC800 A0D4	Address 6
0xC800 A0E0	Threshold For Internal Clock
0xC800 A0F0	Unicast Address 1
0xC800 A0F4	Unicast Address 2
0xC800 A0F8	Unicast Address 3
0xC800 A0FC	Unicast Address 4
0xC800 A100	Unicast Address 5
0xC800 A104	Unicast Address 6
0xC800 C1FC	Core Control
† The MDI interface on Ethernet MAC B (NPE-C) is inactive. All external PHYs are configured via Ethernet MAC A (NPE-B).	

§ §

17.0 High-Speed Serial Interfaces

The functionality supported by the High-Speed Serial (HSS) interfaces are tightly coupled with the code written on the Network Processor Engine (NPE) core. This chapter details the full hardware capabilities of the HSS interfaces contained within the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described below.

Not all of the Intel® IXP42X product line and IXC1100 control plane processors have this functionality. See Table 154.

Table 154. Processors with HSS

HSS	Intel® IXP425 Network Processor B Step	Intel® IXP423 Network Processor	Intel® IXP422 Network Processor	Intel® IXP421 Network Processor	Intel® IXP420 Network Processor	Intel® IXC1100 Control Plane Processor
HSS 0	X	X		X		
HSS 1	X	X		X		

The HSS coprocessor enables the IXP42X product line and IXC1100 control plane processors to communicate in a Time Divisible Multiplexed (TDM) bit serial fashion with external chips. The HSS interfaces are six-wire, serial interfaces that can operate at speeds from 512 KHz to 8.192 MHz.

The NPEs core controls each HSS interface. By programming certain parameters to the HSS interfaces — such as frame length/offset, frame signal polarity, and data endianness — the interfaces can be configured to support a variety of bit serial protocols.

For a list of supported protocols, see the *Intel® IXP400 Software Programmer's Guide*.

17.1 High-Speed Serial Interface Receive Operation

Each High-Speed Serial Interface contains five receive FIFOs and four receive FIFOs intended for facilitating the use of HDLC. Each is two 32-bit words in length. The four receive FIFOs are further divided into two buffers, each one 32-bit word in length. These FIFOs are hardware support added to facilitate the implementation of four HDLC channels and do not preclude the implementation of more HDLC channels using the Network Processor Engine core and the HDLC Co-processor connected.

The HSS interface will be filling one buffer while the NPE empties the other buffer. The fifth receive FIFO is intended for voice-processing support and is four 32-bit words in length. This receive FIFO is split into two buffers, each buffer two 32-bit words in length.

These buffers also behave in a ping-pong fashion, so the NPE will read two 32-bit words at a time for processing. The location that each received byte is placed into these FIFOs is a function of a user programmable look-up table (LUT) and the protocol that is being implemented.

The look-up table will characterize each received byte as one of four types:

- Unassigned
- HDLC
- Voice
- 56-K mode

This characterization will be assigned on a time-slot basis using Intel-supplied APIs. For example, time slot 0 may be defined as a voice cell, time slot 1 as an HDLC wrapped packet, time slot 2 as an undefined time slot, and time slot 3 defined as an 56-K mode cell.

When the HSS receive interface processes the first byte (time slot 0), the look-up table will indicate that this received byte is a voice cell and needs to be placed into the Voice FIFO. Likewise, when the high-speed serial receive interface processes the second byte (time slot 1), the look-up table will indicate that this received byte is an HDLC cell and needs to be placed into one of the HDLC FIFOs. The actual FIFO the byte is placed in is dependent on the protocol implemented and the FIFO arrangement.

For more details, see the *Intel® IXP400 Software Programmer's Guide*.

When the high-speed serial receive interface processes the third byte (time slot 2), the look-up table will indicate that this received byte is an unassigned cell and needs to be discarded. When the high-speed serial receive interface processes the fourth byte (time slot 3), the look-up table will indicate that this received byte is a 56-K mode cell and will also be placed into the Voice FIFO.

17.2 High-Speed Serial Interface Transmit Operation

For transmission using the High-Speed Serial Interface, each High-Speed Serial Interface contains five transmit FIFOs, organized exactly as the receive FIFOs. (For additional details on the FIFO organization, see Section 17.1, "High-Speed Serial Interface Receive Operation" on page 448.)

Each transmitted byte is placed into these FIFOs. The data is transmitted using the High-Speed Serial Interface as a function of a user programmable look-up table (LUT) and the protocol that is being implemented.

The look-up table will characterize each byte to be transmitted as one of four types

- Unassigned
- HDLC
- Voice
- 56-K mode

This characterization will be assigned on a time-slot basis using Intel-supplied APIs.

Assume the same example as before, time slot 0 is be defined as a voice cell, time slot 1 is be defined as an HDLC wrapped packet, time slot 2 is be defined as an undefined time slot, and time slot 3 is be defined as an 56-K mode cell.

When the HSS transmit interface is ready to process the first byte (time slot 0), the look-up table will indicate that the byte to be transmitted is a voice cell and needs to be extracted from the Voice FIFO and placed onto the HSS interface. Likewise, when the HSS transmit interface is ready to process the second byte (time slot 1), the look-up table will indicate that the byte to be transmitted is an HDLC cell and needs to be extracted from one of the HDLC FIFOs and placed onto the HSS interface.

The actual FIFO the byte is extracted from is dependent upon the protocol implemented and the FIFO arrangement. For more details, see the *Intel® IXP400 Software Programmer's Guide*.

When the HSS transmit interface processes the third byte (time slot 2), the look-up table will indicate that the byte to be transmitted is an unassigned cell. Using Intel-supplied APIs, the Intel XScale® Processor can program the HSS interface to transmit one of three values in an unassigned time slot:

- All zeros
- All ones
- High-impedance

When the HSS transmit interface processes the fourth byte (time slot 3), the look-up table will indicate that the byte to be transmitted is a 56-K mode cell and is located in the Voice FIFO. When the transmit interface detects from the transmit look-up table that the slot to be transmitted is a 56-K mode byte, only seven of the eight bits in a time slot will be valid. The most-significant bit or the least-significant bit will be invalid.

Using Intel-supplied APIs, the Intel XScale processor can program the invalid bit location as well as the value to be placed into the invalid bits location when data is transmitted. The data inserted into the invalid bit location can be programmed to be logic 0, logic 1, or tristate. For more details, see the *Intel® IXP400 Software Programmer's Guide*.

17.3 Configuration of the High-Speed Serial Interface

As shown in the previous sections, a wide variety of interface flexibility can be implemented over the High-Speed Serial Interface using various configuration parameters and developing new code on the Network Processor Engine (NPE). For details on current HSS interface configurations supported, see the *Intel® IXP400 Software Programmer's Guide*.

The remainder of this section outlines the HSS interface configuration parameters that can be programmed using Intel-supplied APIs.

There are various programmable functions of the HSS interfaces that have already been described, such as

- Ability to support 56-K time slots
- Ability to select the location of the invalid bit and —
  - — The value to be transmitted when transmitting the invalid bit location
  - — The a value to be sent when transmitting an unassigned time slot
- Ability to classify each time slot using the transmit and receive look-up tables

There are, however, many other programmable features of the HSS interface. They include the ability to:

- Set the frame length
- Set the sampling edge of the frame-sync and the data independently
- Generate/receive a framing bit
- Source/receive the frame-sync signal
- Source/receive the transmit and receive clocks
- Manipulate the definition of a valid frame-sync signal
- Set an offset for the frame-sync signal
- Source various output clock rates

- Loop back the transmit to receive interface internally.

The following discussion briefly describes these features. For more detail on manipulating these settings, see the *Intel® IXP400 Software Programmer's Guide*.

The frame length can be set as a variable function with the maximum frame length being 1,024 bits. The size of the frame length is tightly coupled with the protocol that is implemented by the Network Processor Engine. For the maximum/minimum frame length value and the protocols that are implemented, see the *Intel® IXP400 Software Programmer's Guide*.

The High-Speed Serial Interfaces provides the capability to sample data on the rising or falling edge of the receive clock. The same ability exists for the sampling of the frame-sync signal when the frame-sync signal is configured as an input. The HSS interfaces provides the capability to source data and the frame-sync signal on the rising and falling edge of the transmit clock assuming that the frame-sync signal is being sourced by the IXP42X product line and IXC1100 control plane processors.

If the transmit frame-sync signal is being generated from an external source and the IXP42X product line and IXC1100 control plane processors is sampling this signal, the HSS interface has the capability to sample this data on the rising or falling edge. The data and frame-sync edge generation/detection logic can be set independently for both receive and transmit directions.

For example:

- The receive data may be sampled on the rising (or falling) edge of the receive clock
- The receive frame-sync signal may be sampled on the rising (or falling) edge of the receive clock
- The transmit data may be sourced on the rising (or falling) edge of the transmit clock
- The transmit frame sync may be generated on the falling (or rising) edge of the transmit clock when the IXP42X product line and IXC1100 control plane processors sources the frame-sync signal
- The transmit frame sync may be sampled on the falling (or rising) edge of the transmit clock when the IXP42X product line and IXC1100 control plane processors received the frame-sync signal from an external source

The ability to select the clock edge for sourcing or receiving these signals allows for the maximum flexibility when attempting to meet setup/hold timing between the devices connected. Another point to note is that the receive frame-sync pulse can be programmed to be driven or received by the IXP42X product line and IXC1100 control plane processors. The programmable selection of the data and frame-sync signals are implemented for independently for both HSS interfaces.

To allow the maximum flexibility in a users design, the IXP42X product line and IXC1100 control plane processors provide the ability for the clocks to be generated by the HSS interface or the clocks to be sampled from an external source by the HSS interface. The direction selected for the clocks can be on an individual basis.

The transmit clock can be defined as an input and the receive clock can be configured as an output. The settings are available for each HSS interface. When the clocks are configured as an output, the clock speeds that can be configured are:

- 512 KHz
- 1.536 MHz
- 1.544 MHz
- 2.048 MHz
- 4.096 MHz
- 8.192 MHz

The frame-sync signal is used to allow the HSS interface to synchronize to external devices. The synchronization is obtained by the activity produced on the frame-sync signals. There is a separate frame-sync signal for both transmit and receive for each HSS interface. Using Intel supplied APIs, the HSS frame-sync signals can be programmed to be inputs or outputs.

The APIs also allow the user the ability to select the detection mechanism by which the devices will synchronize. The frame-sync signals can be configured to detect/produce a frame-sync signal in one of four ways:

- Active-low detection/production
- Active-high detection/production
- Falling-edge detection/ production
- Rising-edge detection/production

If the frame-sync signal is configured as an output and is active-low production, the frame-sync signal will output logic l and then assert the frame-sync signal to logic 0 for one clock as defined by the frame-sync offset value. If the frame-sync signal is configured as an input and is active-low detection, the frame-sync signal will signal a detected frame-sync signal when the received frame-sync signal goes from logic l to logic 0 for one clock in the appropriate frame-sync offset location.

If the frame-sync signal is configured as an output and is active-high production, the frame-sync signal will output logic 0 and then assert the frame-sync signal to logic 1 for one clock as defined by the frame-sync offset value. If the frame-sync signal is configured as an input and is active-high detection, the frame-sync signal will signal a detected frame-sync signal when the received frame-sync signal goes from logic 0 to logic 1 for one clock in the appropriate frame-sync offset location.

If the frame-sync signal is configured as an output and as falling-edge production, the frame-sync signal will output logic l and then assert the frame-sync signal to logic 0 for one clock as defined by the frame-sync offset value on the falling edge of the clock. If the frame-sync signal is configured as an input and as falling-edge detection, the frame-sync signal will signal a detected frame-sync primitive when the received frame-sync signal goes from logic l to logic 0 at the appropriate frame-sync offset location.

If the frame-sync signal is configured as an output and as rising-edge production, the frame-sync signal will output logic 0 and then assert the frame-sync signal to logic 1 for one clock as defined by the frame-sync offset value on the rising edge of the clock. If the frame-sync signal is configured as an input and as rising-edge detection, the frame-sync signal will signal a detected frame-sync primitive when the received frame-sync signal goes from logic 0 to logic 1 at the appropriate frame-sync offset location.

The frame-sync offset defines the relation of the frame-sync pulse to the start of the data. The frame-sync pulse can be programmed as a value between 0 and 1,023 bits, using Intel-supplied APIs running on the Intel XScale processor.

If an offset is programmed, the data transmitted will be transmitted the offset number of clock cycles prior to the production/detection of the transmission frame-sync signal. For received data, the first data time slot will be received an offset number of clock cycles prior to the production/detection of the receive frame-sync signal.

Loop-back is a debug function that can be used to deduce and debug problems observed when using the HSS interface. Loop back can help isolate a problem by eliminating parts externally connected to the HSS interface as a source of the problem. If the system works correctly in loop-back mode, an indication is given that any problem is most-likely within the devices externally connected to the HSS interface.

All transmit and receive FIFOs should be empty before loop back is entered. The HSS interface must synchronize to the frame-sync pulse before any operations commence, as in normal mode of operation.

Either an internal or external frame-sync pulse or clock can still be utilized. When using the internal frame-sync pulse/clock, the transmit frame-sync pulse/clock is internally looped to the receive side of the HSS interface. If an external frame pulse/clock is used, the clock/frame-sync pulse must be supplied on the transmit frame-sync pulse/clock pins. The value contained on the transmit frame-sync pulse/clock pins is then replicated to the receive logic. No data is required to be supplied on any of the receive signals. This requirement also means that no data is sent or received to or from the external devices while in loop back mode. The Intel XScale processor — using Intel-supplied APIs — can be used to activate the loop-back mode.

All of the configuration parameters are predicated on the implementation in the Network Processor Engine (NPE). For details on exact supported configurations and protocols using the HSS interface, see the *Intel® IXP400 Software Programmer's Guide*.

17.4 Obtaining High-Speed, Serial Synchronization

Before the High-Speed Serial Interface indicates to the Network Processor Engine that it has receive data, the HSS interface must synchronize to the frame-sync signal. Receive synchronization must be obtained before any data is placed into the receive FIFOs.

As stated previously, the Intel APIs and Intel XScale processor can program an offset between the frame pulse and the start of the frame. This offset will influence the synchronization process. When there is no offset defined, the HSS interface will be required to receive two valid frame-sync pulses before synchronization is achieved. When there is an offset defined, the HSS interface will be required to receive three valid frame-sync pulses before synchronization is achieved.

If no offset is defined, the first time slot of the second frame will be considered a valid frame. If an offset is defined, the first time slot of the third frame will be considered a valid frame. In either case, due to the synchronization protocol the first data the NPE receives will begin at the start of a frame, thus eliminating the need for time slot counters.

Conversely, the HSS interface will not request for data from the NPE until the HSS interface has synced to the transmit frame-sync pulse. The synchronization process is defined the same as for the receive side. Once transmit synchronization has been achieved, data will be sent from the NPE to the FIFOs and transmitted over the HSS interface at the start of the next frame.

If an offset is programmed, the data transmitted over the HSS interface will be transmitted the offset number of clock cycles before the frame-pulse detection/generation.

If the HSS Core has not synchronized to the frame-sync pulse, the HSS interface will remain in an idle state. Data received before synchronization is acquired will be dropped. When synchronization is not achieved, the transmit data output is set to high impedance.

If FIFO under-run/over-run occurs, frame synchronization is maintained if the frame-sync pulses are still generated.

The HSS interface supports “gapped” frame pulses meaning that only an out position frame pulse will cause a loss of synchronization. A missing frame pulse will not necessarily cause loss of synchronization to occur.

The behavior of the HSS interface is indifferent to the source of the frame-sync pulse. The frame-sync pulse can be sourced externally or generated internally. Figure 84 and Figure 85 shows an example of the HSS interfaces obtaining synchronization with no offset defined.

Figure 84. Tx Frame-Sync Example (Presuming an Offset of 0)

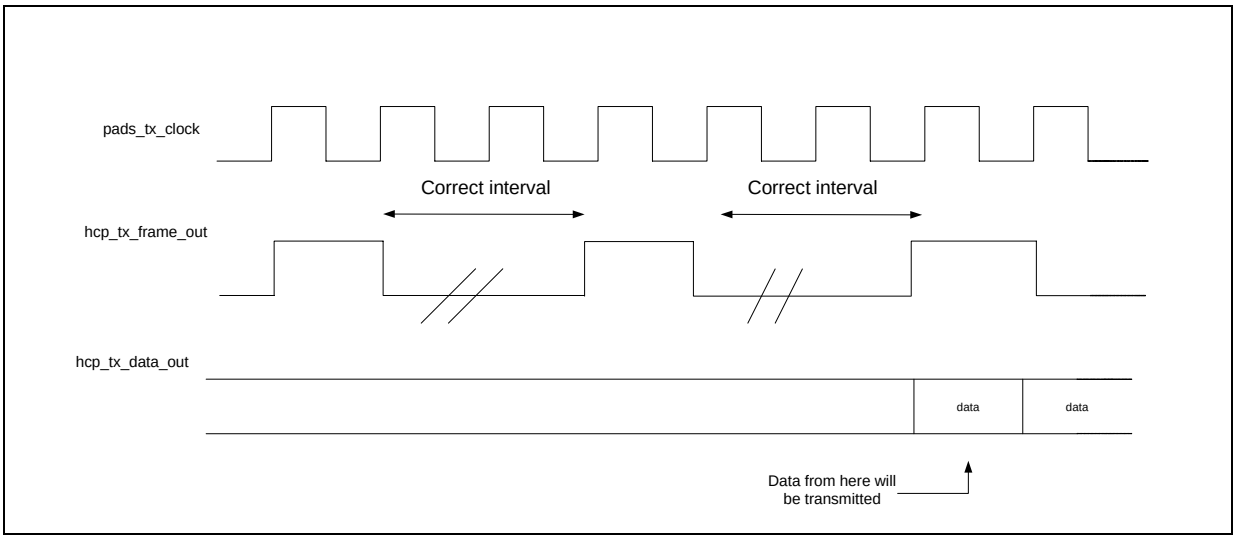
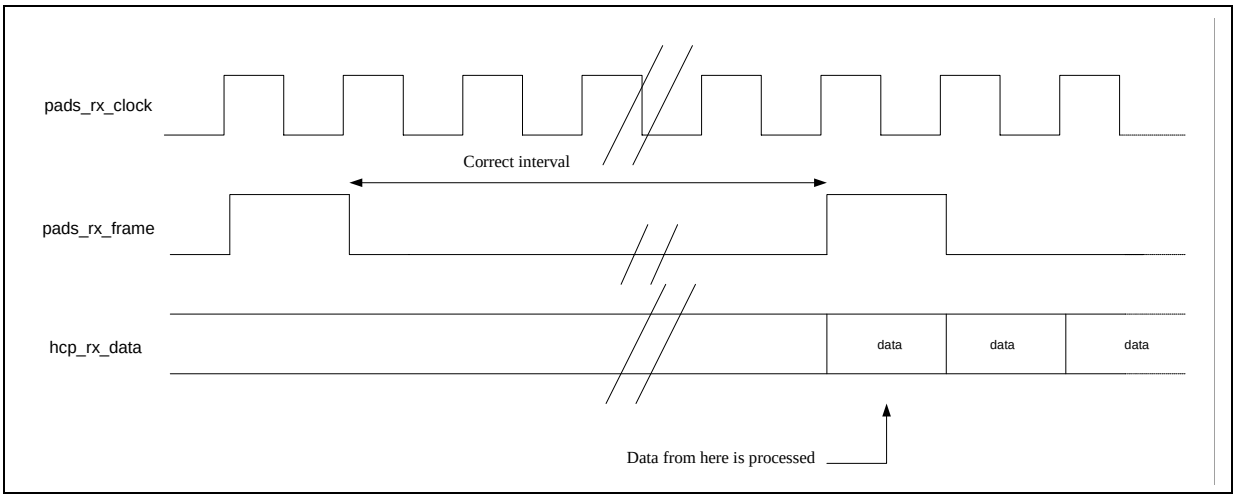


Figure 85. Rx Frame-Sync Example (Presuming Zero Offset)



17.5 HSS Registers and Clock Configuration

There are numerous Control and Status Registers (CSRs) contained within the IXP42X product line and IXC1100 control plane processors' NPE/HSS interface that are used to configure the many unique HSS settings discussed in this manual. The functional details of these registers are not exposed within any IXP42X product line and IXC1100 control plane processors collateral and are reserved for use by NPE firmware only. The IxHssAcc API, however, provides indirect access to the these registers to allow the complete control and configuration of all HSS features enabled by a particular Intel® IXP400 Software Release. The *Intel® IXP400 Software Programmer's Guide* should be referenced for specific information regarding use of the IxHssAcc API.

There is one register titled the HSS Clock Divider Register that provides a means to generate a unique data clock for each of the two IXP42X product line and IXC1100 control plane processors' HSS interfaces. This may help reduce BOM cost by eliminating the need for a clock oscillator in cases where the system can tolerate an imperfect data clock with a certain amount of jitter. The IxHssAcc API will configure the HSS clock divider register with the appropriate values depending on which clock frequencies is selected, being 512 KHz, 1.536 MHz, 1.544 MHz, 2.048 MHz, 4.096 MHz, and 8.192 MHz. This is discussed further in the next section:

17.5.1 HSS Clock and Jitter

Each of the two High-Speed Serial (HSS) interfaces — on the IXP425, IXP423 and IXP421 network processors — can be configured to generate an output clock on the HSS_TXCLK and HSS_RXCLK pins. This output clock, however, is not as accurate and stable as using an external oscillator. That is because the HSS clock is based on the internal, 133.32-MHz AHB bus. This frequency does not divide down easily. Subsequently, jitter and error are introduced into the resultant HSS output clock.

If developers are clocking a framer, DAA, or other device with a sensitive input PLL, they should use an external clock.

To provide developers with additional data, this chapter contains five tables:

1. Table 155, "HSS Tx/Rx Clock Output" on page 455
2. Table 156, "HSS Tx/Rx Clock Output Frequencies and PPM Error" on page 456
3. Table 157, "HSS Tx/Rx Clock Output Frequencies And Their Associated Jitter Characterization" on page 456
4. Table 158, "HSS Frame Output Characterization" on page 456
5. Table 159, "Jitter Definitions" on page 457

The jitter and error calculations, in the following tables, are valid only for IXP400 software Release 1.4 and later.

17.5.2 Overview of HSS Clock Configuration

The HSS clock is generated by the WAN/Voice network processor engine (NPE). This clock can be configured by using the HSS API in IXP400 software releases 1.4 and later. The clock speed can be set by using the HSS API.

It is important to be aware of the jitter and frequency error on the output clock of the HSS. The six clock values in Table 155 have been pre-defined by IXP400 software release 1.4 to minimize jitter and PPM error. The values shown in the following table are those currently supported in software.

Table 155. HSS Tx/Rx Clock Output

HSS Clock Output Frequency	Protocol	Frame Size (Bits)	Notes
512 KHz	GCI	32	1
1.536 MHz	GCI	96	1
1.544 MHz	1 T1	193	
2.048 MHz	1 T1/E1	256	2

Table 155. HSS Tx/Rx Clock Output

4.096 MHz	2 T1/E1	512	2
8.192 MHz	4 T1/E1	1,024	2

Notes:
1. These clock speeds are supported using the HSS API. However, the GCI protocol is not supported by the IXP400 software.
2. When the frequencies of 2.048, 4.096, or 8.192 MHz are used for T1, the data rate for each T1 remains at 1.544 MHz by making certain time slots within a frame unassigned and with no data. The IXP425, IXP423 and IXP421 network processors' HSS can be configured to discard unassigned time slots.

Table 156. HSS Tx/Rx Clock Output Frequencies and PPM Error

HSS Tx/Rx Frequency	Min. Frequency (MHz)	Avg. Frequency (MHz)	Max. Frequency (MHz)	Avg. Frequency Error (PPM)	Notes
512 KHz	0.508855	0.512031	0.512769	-60.0096	1, 2, 3
1.536 MHz	1.515	1.536	1.55023	-60.0096	1, 2, 3
1.544 MHz	1.515	1.5439	1.55023	+60.0024	1, 2, 3
2.048 MHz	2.01998	2.0481	2.08313	-60.0096	1, 2, 3
4.096 MHz	3.92118	4.0962	4.16625	-60.0096	1, 2, 3
8.192 MHz	7.4066667	8.1925	8.3325	-60.0096	1, 2, 3

Notes:
1. These HSS Tx/Rx clock output frequencies are based on the set-up parameters in Table 155.
2. Characterization data of the HSS Tx/Rx clock output frequency data was determined by silicon simulation.
3. The parts-per-million (PPM) error rate is calculated using the average output frequency versus the ideal frequency.

Table 157. HSS Tx/Rx Clock Output Frequencies And Their Associated Jitter Characterization

HSS Tx/Rx Clock Output Frequency	Pj Max. (ns)	Cj Max. (ns)	Aj Max. (ns)
512 KHz	12.189	15	18.283
1.536 MHz	9.063	15	86.102
1.544 MHz	12.359	15	210.099
2.048 MHz	-8.204	15	118.957
4.096 MHz	10.9	15	190.742
8.192 MHz	12.951	15	226.634

Note: For jitter definitions, see Table 159.

Table 158. HSS Frame Output Characterization

HSS Tx/Rx Frequency	Frame Size (Bits)	Actual Frame Length (µS)	Frame Length Error (PPM)
512 KHz	32	62.496249	-60.0096
1.536 MHz	96	62.496249	60.016
1.544 MHz	193	125.007499	60.0024
2.048 MHz	256	124.9925	-60.0096

Table 158. HSS Frame Output Characterization

4.096 MHz	512	62.496	-60.0096
8.192 MHz	1024	62.49624	-60.0096
Note: PPM frame length error is calculated from ideal frame frequency.			

Table 159. Jitter Definitions

Jitter Type	Jitter Definition
Period Jitter (Pj):	$Pj_{(i)} = Period_{(i)} - Period_{average}$
Cycle-to-Cycle Jitter (Cj):	$Cj_{(i)} = Pj_{(i+1)} - Pj_{(i)}$
Wander-or-Accumulated Jitter (Aj):	$Aj_{(i)} = \sum_{i} Pj$

17.6 HSS Supported Framing Protocols

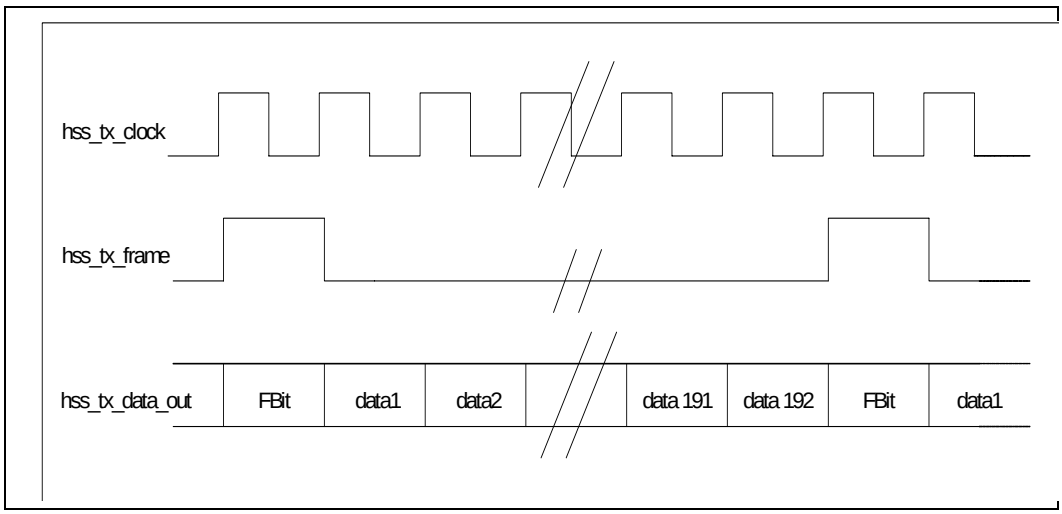
The following sections provide an overview of some Framing Protocols supported by each of the two IXP421, IXP423 and IXP425 network processors' HSS interfaces in addition to the recommended HSS interface setting for each protocol that are configured using the IxHssAcc API defined in the *Intel® IXP400 Software Programmer's Guide*.

17.6.1 T1

T1 is a serial data stream operating at 1.544 MHz. The stream is composed of frames of which there are 8000 a second. Each frame consists of 1 frame bit and 24 timeslots, each timeslot is a byte in size. As there are 24 timeslots per frame, T1 can carry 24 channels.

Figure 86 and Figure 87 illustrate a typical T1 frame with an active high frame sync (level) and a positive edge clock for generating or sampling data. The HSS clock and frame pulse can be programmed to be either HSS outputs or HSS inputs. An offset can be programmed indicating when the TX frame is to be transmitted. The Polarity of the received data and the level of the frame can also be programmed using the IxHssAcc API.

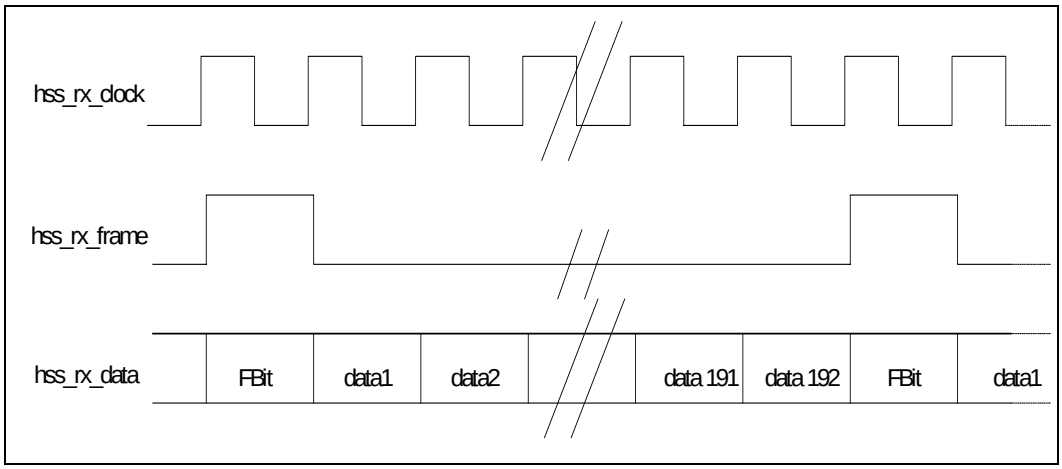
Figure 86. T1 Transmit Frame



In Figure 86, the FBit to be transmitted is stored in the HSS Transmit FIFO. The HSS knows which timeslot in the FIFO is holding the FBit, as it knows from the timeslot counter and frame offset when the FBit should be transmitted.

In Figure 87, the FBit to be received is padded with 7 other bits (zeros) and placed into the HSS Receive FIFO.

Figure 87. T1 Receive Frame



By using the IxHssAcc API, the following settings should be considered when configuring HSS interface for T1 operation:

- Frame size 193 bits (for T1).
- Frame sync simultaneous with FBit - TX frame offset and RX frame offset should be set due to HSS logic, different values due to external device can be accommodated.
- Select use of input/output TX/RX frame syncs.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.

- Frame sync active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to use FBit (not data for T1) at frame start.
- Select value for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Configure lookup tables.

17.6.2 E1

E1 is a High-Speed Serial stream operating at 2.048 MHz. The stream is composed of frames of which there are 8000 a second. Each frame consists of 32 slots, and each slot is a byte in size. As there are 32 slots per frame, E1 can carry 32 channels. There are no frame bits in this protocol.

Figure 88 and Figure 89 illustrate a typical E1 frame with an active high frame sync (level) and a positive edge clock for generating or sampling data. The HSS clock and frame pulse can be programmed to be either HSS outputs or HSS inputs. An offset can be programmed indicating when the TX frame is to be transmitted. The Polarity of the received data and the level of the frame can also be programmed using the IxHssAcc API.

Figure 88. E1 Transmit Frame

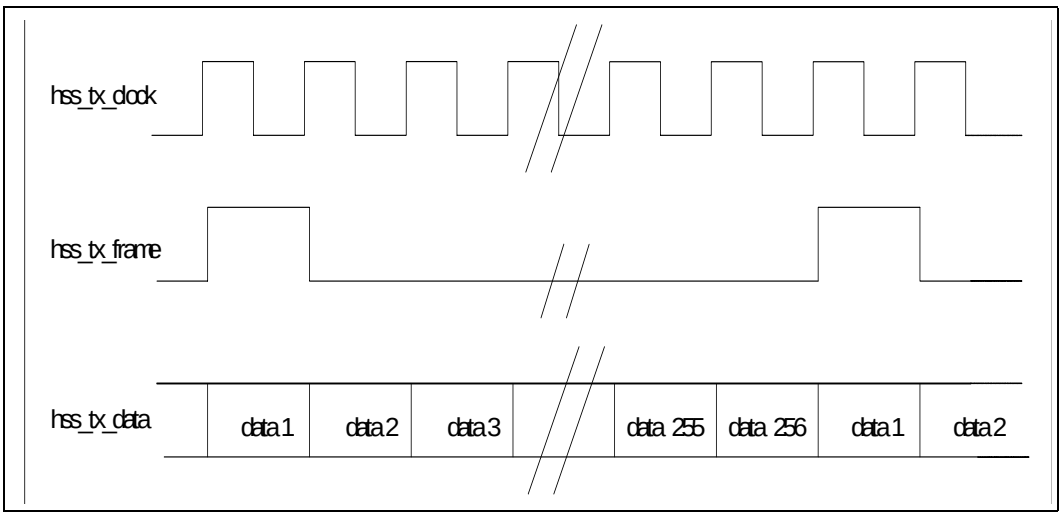
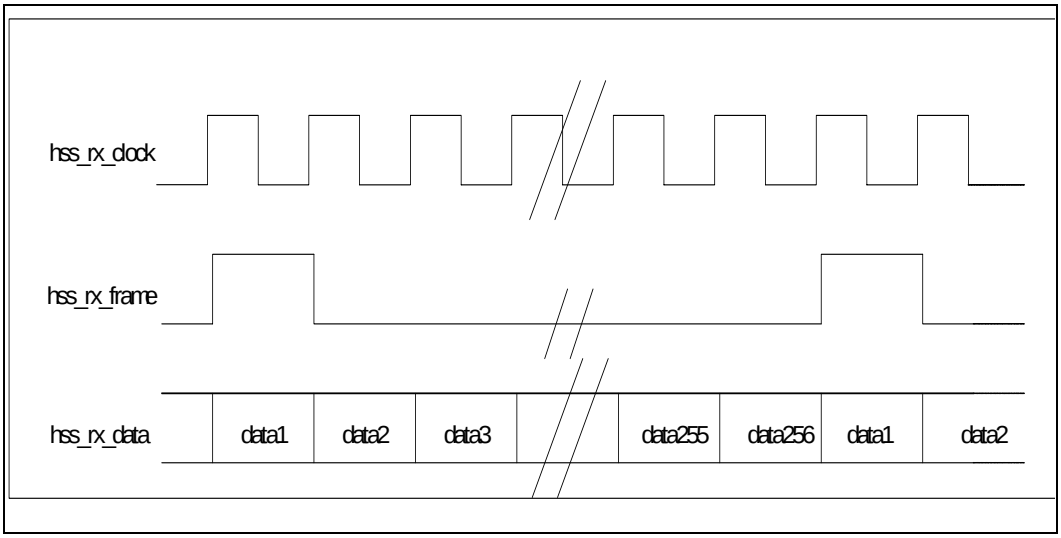


Figure 89. E1 Receive Frame



By using the IxHssAcc API, the following settings should be considered when configuring HSS interface for E1 operation:

- Frame size 256 bits (for E1).
- Frame sync simultaneous with first bit in first timeslot - TX frame offset and RX frame offset should be set due to HSS logic, different values due to external device can be accommodated.
- Select use of input/output TX/RX frame syncs.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame sync active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to not use FBit at frame start.
- Select value for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Configure lookup tables.

17.6.3 MVIP

MVIP provides a method of interlacing multiple E1 streams onto a single E1 line, and multiple T1 streams onto a single T1 line. Increasing the clock speed and alternating various timeslots helps provide this functionality. A single T1 line can also be mapped into an E1 line.

By using the IxHssAcc API, the following settings should be considered when configuring the HSS interface for MVIP:

- Set frame size for MVIP.

- Frame sync simultaneous with first data nibble - set TX frame offset and RX frame offset due to HSS logic, different values due to external device can be accommodated.
- Select use of input/output TX/RX frame syncs.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame sync active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select not to use FBit in the frame.
- Select level for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Set interlace mode (byte/frame)
- Set lookup tables.

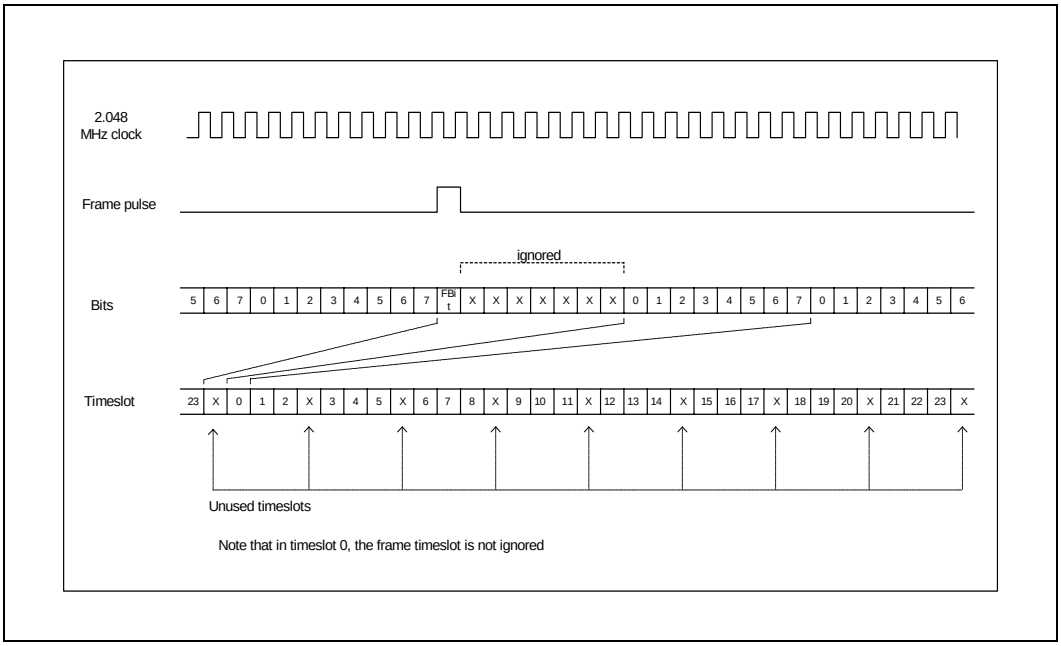
The HSS clocks are capable of running at 1.544MHz, 2,048MHz, 4.096MHz and 8.192 MHz for MVIP mode.

The following three sections describe the three ways in which the MVIP protocol can operate. The TX and RX are not explicitly described in these sections, as the RX side of the protocol is identical to the TX side of the protocol.

17.6.3.1 MVIP using 2.048Mbps Backplane

The clock rate for this form of T1 is 2.048Mbps as opposed to the usual T1 rate of 1.544 Mbps. The data rate remains at 1.544Mbps, meaning that certain timeslots within the frame carry no data and thus are discarded by the HSS interface, as defined by unassigned timeslots configured by the look up tables. This is illustrated in Figure 90

Figure 90. MVIP, Interleaved Mapping of a T1 Frame to an E1 Frame



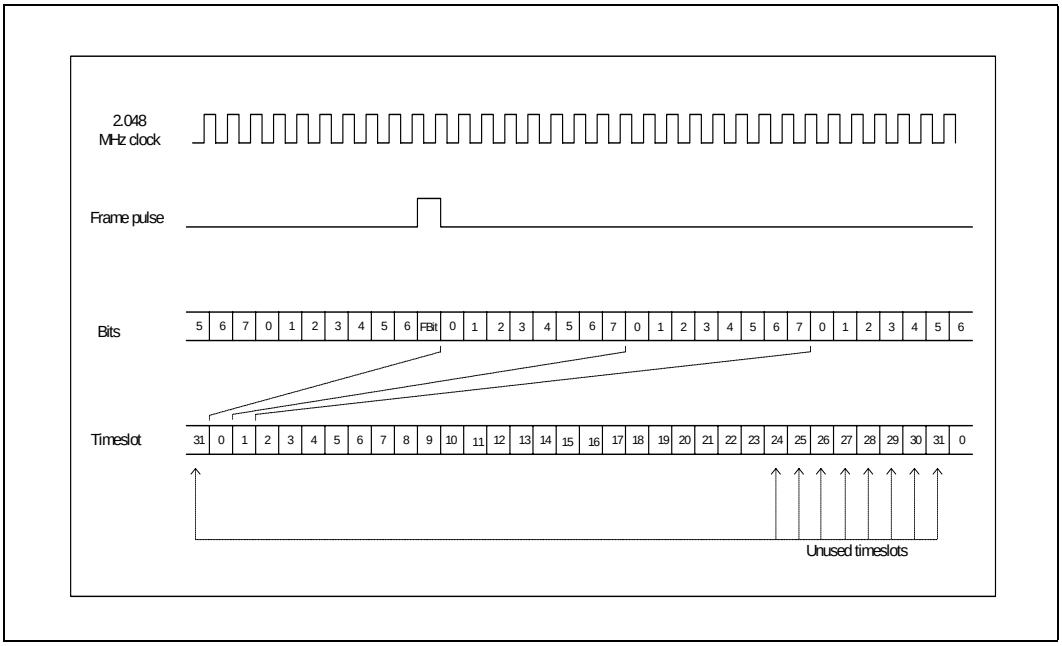
Every fourth timeslot received by the HSS is discarded, meaning it is not loaded into the FIFO and is therefore not sent to the NPE.

The HSS Interface can transmit all zeros/ones (HSS programmable) for the duration of the unassigned timeslots.

Received unassigned timeslots are not sent to the NPE as they are discarded by the HSS.

Another method of sending a T1 line over an E1 line called frame mapping is illustrated in Figure 91. This method groups the 24 timeslots together and places unassigned timeslots towards the end of the frame. The FBit is located at the last bit of the 32nd timeslot, the HSS will not treat this timeslot any differently to other timeslots, it is up the software to detect the FBit. The NPE can select which method it desires to use.

Figure 91. MVIP, Frame Mapping a T1 Frame to an E1 Frame



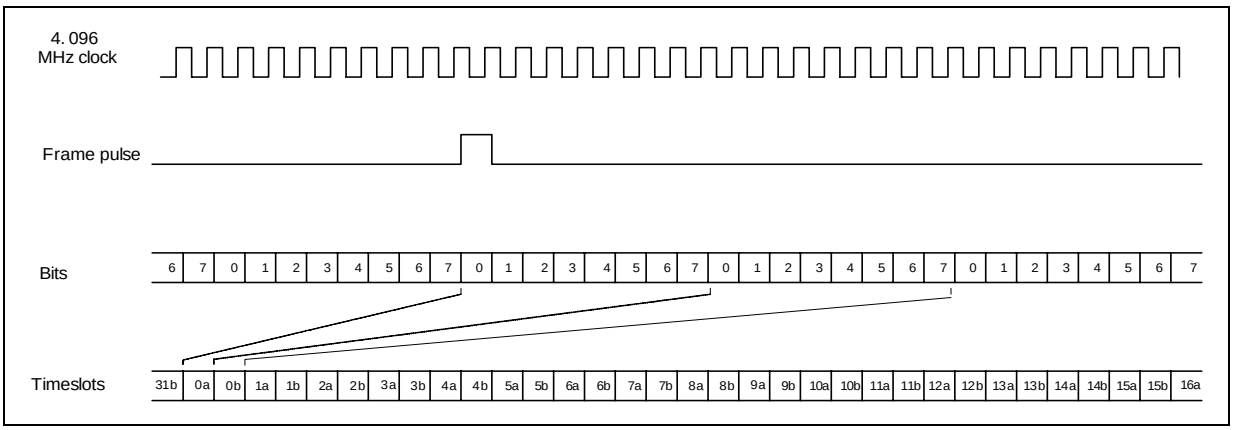
The HSS interface can be programmed to automatically ignore (lookup table assigned) the last eight timeslots meaning the NPE will not receive the contents of the last eight timeslots. When timeslot 23 is transmitted, the next data from the NPE will not be transmitted until timeslot zero occurs. The HSS will transmit all zeros/ones for the duration of the empty timeslots (NPE programmable).

The NPE must program the HSS to indicate which method of T1 mapping is used (if any).

17.6.3.2 MVIP Using 4.096-Mbps Backplane

This backplane is used to transport 2 E1s or 2 T1s on a single line utilizing a clock rate of 4.096 MHz. Two complete E1 frames will fill this frame, therefore unassigned timeslots are not compulsory. When transporting T1 frames, unassigned timeslots are used for padding the frame due to the shorter length of the T1 frame.

Figure 92. MVIP, Byte Interlacing Two E1 Streams Onto a 4.096-Mbps Backplane



In Figure 92, the 'a' denotes the first E1 stream, the 'b' denotes the second E1 stream, the 2 streams are interlaced byte wise.

Another method of placing E1 stream on this backplane is to process the first entire E1 stream followed by the second complete E2 stream (frame interleaving).

Figure 93. MVIP, Byte Interleaving Two T1 Streams Onto a 4.096-Mbps Backplane

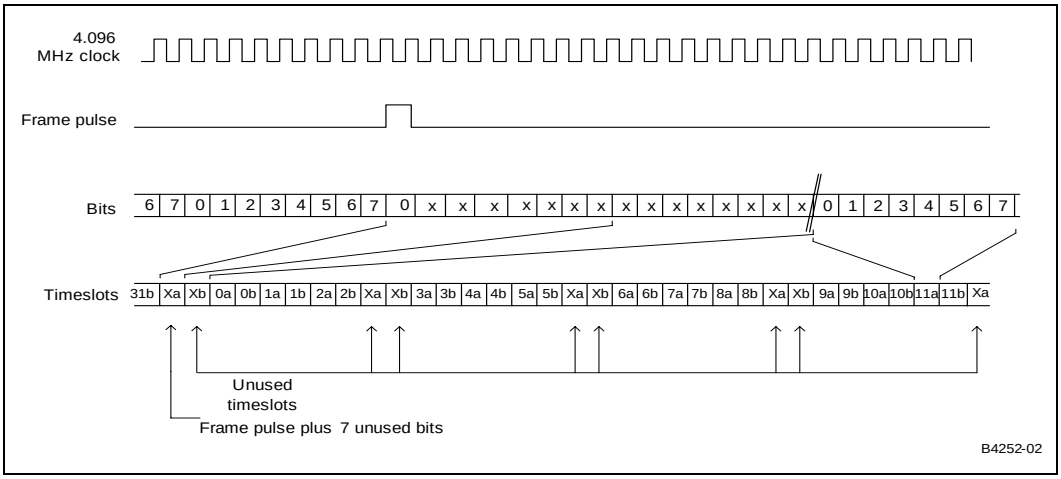


Figure 93 illustrates a method in which 2 T1 frames are placed on a 4.096Mbps backplane. It can be seen that the first 2 timeslots are unassigned with the exception of the frame pulse. The first three timeslots of each T1 stream are then placed (interlaced) in succession on the bus, then 1 unassigned timeslot per T1 stream present are placed on the bus. Unassigned RX bytes do not pass through the HSS FIFO (lookup tables give unassigned timeslots).

The HSS can also transmit unassigned timeslots, the value of which is programmable. The NPE need only supply the contents of the T1 frames, it does not need to transmit unused timeslots to the HSS. The location of these unassigned timeslots are defined by the lookup table.

The backplane can contain the 2 T1 streams byte interlaced as shown in Figure 93 or the T1 stream can be placed in its entirety first followed by 8 unassigned timeslots (frame pulse at the last bit of the 32nd timeslot. The second T1 stream then commences followed by 8 unassigned timeslots. The frame pulse is coincidental with the last bit in the 32nd timeslot. The second timeslot follows the format of the first timeslot and together take up the 64 timeslots available. Once again the HSS can be programmed by the NPE to ignore the 8 unassigned timeslots while taking the frame bit into account.

17.6.3.3 MVIP Using 8.192-Mbps Backplane

This backplane supports 4 E1 or 4 T1 streams on a single line which require that it operates at 8.192MHz.

Figure 94. MVIP, Byte Interleaving Four E1 Streams on a 8.192-Mbps Backplane Bus

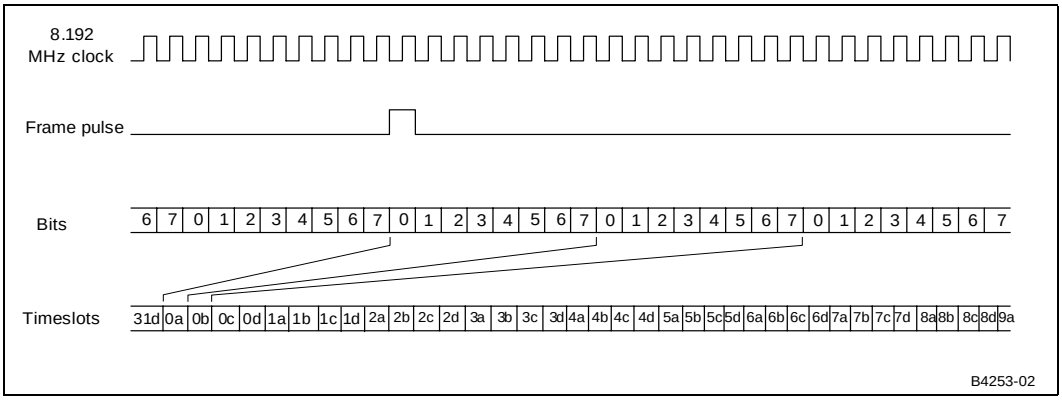
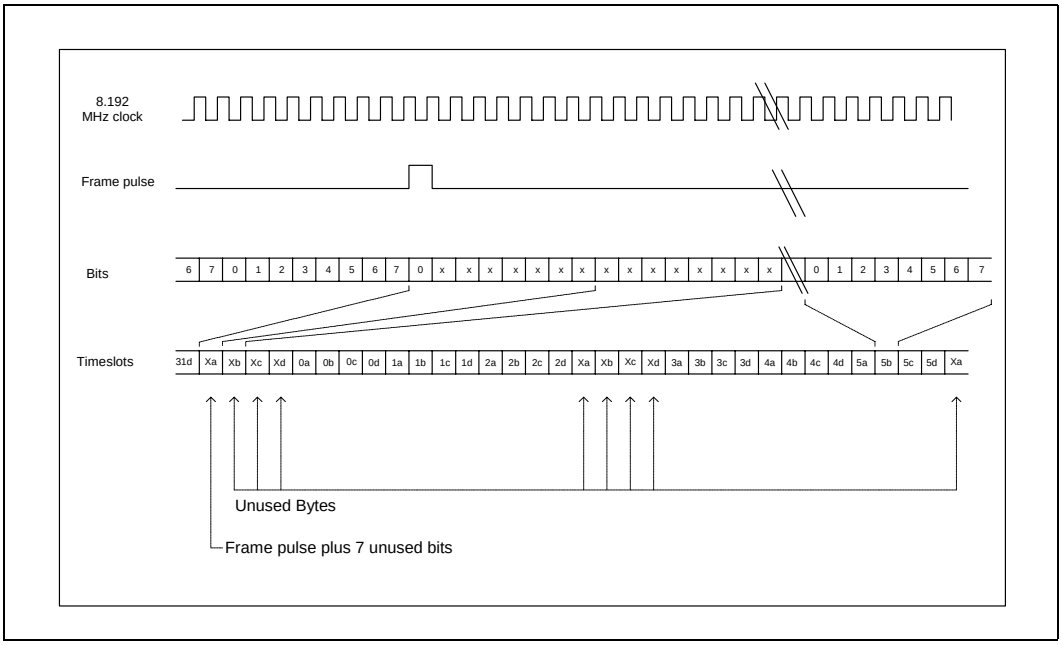


Figure 94 illustrates that 4 E1 streams can be byte interleaved. The frame pulse occurs at the first byte of the first E1 stream. No unassigned timeslots are necessary here as the E1 frames can completely fill all the timeslots available.

When T1 frames are placed on this backplane bus, then unassigned timeslots are required as a T1 frame is 24 timeslots wide unlike E1, which is 32 timeslots wide.

Byte interleaving involves disregarding four in every sixteen timeslots (as shown in Figure 95). The first 4 timeslots are unassigned except for the framing pulse, the following 4 bytes are byte 0 of each T1 frame, the next 4 are byte 1 of each T1 frame and the next 4 bytes are byte 2 of each T1 frame, the following 4 bytes are again unassigned. Unassigned timeslots are dictated by the look up tables.

Figure 95. MVIP, Byte Interleaving Four T1 Streams on a 8.192-Mbps Backplane Bus



Frame interleaving T1 frames onto this backplane bus would be to process the first T1 frame in its entirety starting with the first timeslot and finishing on the 24th timeslot, the frame bit/pulse is located on the last bit of the 32nd timeslot.

The second T1 frame is then processed and so on until all 4 frames are processed, this fills the entire 128 timeslots available. This mode is also programmable by the NPE.

§ §

18.0 Universal Serial Bus (USB) v1.1 Device Controller

This chapter describes the Universal Serial Bus (USB) protocol and its implementation-specific options for device controllers. These options include:

- Endpoint number, type, and function
- Interrupts to the Intel XScale® Processor
- A transmit/receive FIFO interface

A working knowledge of the USB standard is vital to effective use of this chapter. The Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor's Universal Serial Bus Device Controller (UDC) is USB-compliant and supports all standard device requests issued by the host. UDC operation summaries and quick reference tables are provided.

Refer to the *Universal Serial Bus Specification*, revision 1.1, for a full description of the USB protocol. The *Universal Serial Bus Specification* is available at http://www.usb.org.

18.1 USB Overview

The UDC supports 16 endpoints and can operate half-duplex at a rate of 12 Mbps (as a slave only, not as a host or hub controller). The UDC supports four device configurations. Configurations 1, 2, and 3 each support two interfaces. This allows the host to accommodate dynamic changes in the physical bus topology. A configuration is a specific combination of USB resources available on the device. An interface is a related set of endpoints that present a device feature or function to the host.

The UDC transmits serial information that contains layers of communication protocols. Fields are the most basic protocol. UDC fields include:

- Sync
- Packet identifier (PID)
- Address
- Endpoint
- Frame number
- Data
- Cyclic Redundancy Check (CRC)

Fields are combined to produce packets.

A packet's function determines the combination and number of fields that make up the packet. Packet types include:

- Token
- Start of frame
- Data
- Handshake

Packets are assembled into groups to produce transactions. Transactions fall into four groups:

- Bulk
- Control
- Interrupt
- Isochronous

Endpoint 0 is used only to communicate the control transactions that configure the UDC. Endpoint 0s responsibilities include:

- Connection
- Address assignment
- Endpoint configuration
- Bus enumeration
- Disconnection

The UDC uses a dual-port memory to support FIFO operations. Each Bulk and Isochronous Endpoint FIFO structure is double-buffered to enable the endpoint to process one packet as it assembles another. The Intel XScale® processor can fill and empty the FIFOs. An interrupt is generated when a packet has been received. Interrupts are also generated when the FIFO encounters a short packet or zero-length packet. Endpoint 0 has a 16-entry-long, 8-bit-wide FIFO that can only be read or written by the Intel XScale® processor.

For endpoints 1-15, the UDC uses its dual-ported memory to hold data for a Bulk OUT transaction while the transaction is checked for errors. If the Bulk OUT transaction data is invalid, the UDC sends a NAK handshake to request the host to resend the data. The software is not notified that the OUT data is invalid until the Bulk OUT data is received and verified. If the host sends a NAK handshake in response to a Bulk IN data transmission, the UDC resends the data. Because the FIFO maintains a copy of the data, the software does not have to reload the data.

The external pins dedicated to the UDC interface are UDC+ and UDC-. The USB protocol uses differential signalling between the two pins for half-duplex data transmission. A 1.5-KΩ, pull-up resistor must be connected to the USB cable's D+ signal to pull the UDC+ pin high when it is not driven. Pulling the UDC+ pin high when it is not driven allows the UDC to be a high-speed, 12-Mbps device and provides the correct polarity for data transmission.

The serial bus uses differential signalling to transmit multiple states simultaneously. These states are combined to produce transmit data and various bus conditions, including:

- Idle
- Resume
- Start of Packet
- End of Packet
- Disconnect
- Connect
- Reset

18.2 Device Configuration

The configuration of the Universal Serial Bus V 1.1 Device Controller is shown in Table 160.

Table 160. Endpoint Configuration: Universal Serial Bus Device Controller

Endpoint Number	Type	Function	FIFO Size (bytes) X Number of FIFOs
0	Control	IN/OUT	16
1	Bulk	IN	64x2
2	Bulk	OUT	64x2
3	Isochronous	IN	256x2
4	Isochronous	OUT	256x2
5	Interrupt	IN	8
6	Bulk	IN	64x2
7	Bulk	OUT	64x2
8	Isochronous	IN	256x2
9	Isochronous	OUT	256x2
10	Interrupt	IN	8
11	Bulk	IN	64x2
12	Bulk	OUT	64x2
13	Isochronous	IN	256x2
14	Isochronous	OUT	256x2
15	Interrupt	IN	8

Data flow is relative to the USB host. IN packets represent data flow from the UDC to the USB host. OUT packets represent data flow from the USB host to the UDC.

The FIFOs for the bulk and isochronous endpoints are double-buffered so that one packet can be processed while the next is being assembled. While the UDC is busy transmitting an IN packet from a particular endpoint, the Intel XScale® processor can be loading the same endpoint for the next frame transmission. Likewise, while the Intel XScale® processor is unloading an OUT endpoint, the UDC can continue to process the next incoming packet to that endpoint.

18.3 USB Operation

After an Intel XScale® processor reset — or when the USB host issues a USB reset — the UDC configures all endpoints and is forced to use the USB default address, 0. After the UDC configures the endpoints, the USB host assigns the UDC a unique address. At this point, the UDC is under the USB host's control and responds to commands that use control transactions to transmit to endpoint 0.

The following sections provide details of the USB protocol in a bottom-up fashion, starting with signalling levels.

18.3.1 Signalling Levels

USB uses differential signalling to encode data and to indicate various bus conditions. The USB specification refers to the J and K data states to differentiate between high- and low-speed transmission. Because the UDC supports only 12-Mbps transmission, references are made only to actual data state 0 and actual data state 1.

By decoding the polarity of the UDC+ and UDC- pins and using differential data, four distinct states are represented. Two of the four states are used to represent data. A 1 indicates that UDC+ is high and UDC- is low. A 0 indicates that UDC+ is low and UDC- is high. The two remaining states and pairings of the four encodings are further decoded to represent the current state of the USB.

Table 161 shows how differential signalling represents eight different bus states.

Table 161. USB States

Bus State	UDC+ / UDC- Pin Levels
Idle	UDC+ high, UDC- low (same as a 1).
Suspend	Idle state for more than 3 ms.
Resume	UDC+ low, UDC- high (same as a 0).
Start of Packet	Transition from idle to resume.
End of Packet	UDC+ AND UDC- low for 2-bit times followed by an idle for 1-bit time.
Disconnect	UDC+ AND UDC- below single-ended low threshold for more than 2.5 µs. (Disconnect is the static bus condition that results when no device is plugged into a hub port.)
Connect	UDC+ OR UDC- high for more than 2.5 µs.
Reset	UDC+ AND UDC- low for more than 2.5 µs. (Reset is driven by the USB host controller and sensed by a device controller.)

USB Hosts and USB hubs have pull-down resistors on both the D+ and D- lines. When a device is not attached to the cable, the pull-down resistors cause D+ and D- to be pulled down below the single-ended low threshold of the USB host or USB hub. This creates a state called single-ended zero (SE0). A disconnect is detected by the USB host when an SE0 persists for more than 2.5 µs (30-bit times). When the UDC is connected to the USB cable, the pull-up resistor on the UDC+ pin causes D+ to be pulled above the single-ended high threshold level. After 2.5 µs elapse, the USB host detects a connect.

After the USB Host detects a Connect, the bus is in the idle state because UDC+ is high and UDC- is low. The bus transitions from the Idle state to the Resume state (a 1-to-0 transition) to signal the Start of Packet (SOP). Each USB packet begins with a Sync field that starts with the 1-to-0 transition.

After the packet data is transferred, the bus signals the End of Packet (EOP) state by pulling both UDC+ and UDC- low for 2 bit times, followed by an Idle state for 1 bit time. If the idle persists for more than 3 ms, the UDC enters Suspend state. The USB Host can awaken the UDC from the Suspend state by signalling a reset or by switching the bus to the resume state via normal bus activity. Under normal operating conditions, the USB host periodically signals an Start of Frame (SOF) to ensure that devices do not enter the suspend state.

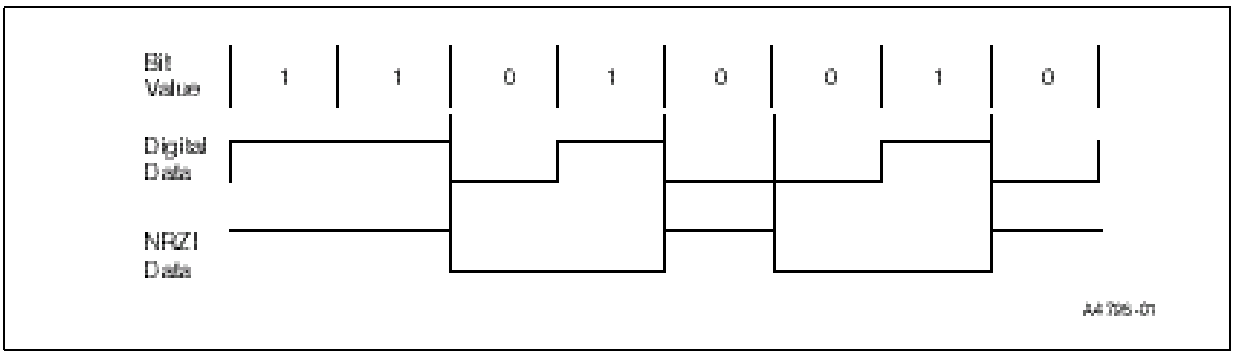
18.3.2 Bit Encoding

USB uses non-return to zero inverted (NRZI) to encode individual bits. Both the clock and the data are encoded and transmitted within the same signal. Instead of representing data by controlling the state of the signal, transitions are used. A 0 is represented by a transition, and a 1 is represented by no transition (this produces the data).

Each time a 0 occurs, the receiver logic synchronizes the baud clock to the incoming data (thus producing the clock). To ensure the receiver is periodically synchronized, any time six consecutive 1s are detected in the serial bit stream, a 0 is automatically inserted by the transmitter. This procedure is known as “bit stuffing.” The receiver logic, in turn, automatically detects stuffed bits and removes them from the incoming data.

Bit stuffing causes a transition on the incoming signal at least once every 7 bit-times to guarantee baud-clock lock. Bit stuffing is enabled for an entire packet beginning when the start of packet is detected and maintained until the end of packet is detected (enabled during the sync field all the way through the CRC field). Figure 96 shows the NRZI encoding of the data byte 0b1101 0010.

Figure 96. NRZI Bit Encoding Example



18.3.3 Field Formats

Individual bits are assembled into groups called fields. Fields are used to construct packets and packets are used to construct frames or transactions. There are seven USB field types:

- Sync
- PID
- Address
- Endpoint
- Frame Number
- Data
- CRC

A Sync is preceded by the Idle state and is the first field of every packet. The first bit of a Sync field signals the SOP to the UDC or host. A Sync is 8 bits wide and consists of seven 0s, followed by a 1 (0x80). Bits are transmitted to the bus least-significant-bit-first in every field, except the CRC field.

The PID is 1 byte wide and always follows the sync field. The first four bits contain an encoded value that represents packet type (Token, Data, Handshake, and Special), packet format, and type of error detection. The last four bits contain a check field that ensures the PID is transmitted without errors.

The check field is generated by performing a 1s complement of the PID. The UDC XORs the PID and CRC fields and takes the action prescribed in the USB standard if the result does not contain all ones, which indicates an error has occurred in transmission.

The Address and Endpoint fields are used to access the UDC’s 16 endpoints. The Address field contains seven bits and permits 127 unique devices to be placed on the USB. After the USB host signals a reset, the UDC and all other devices are assigned the default address, 0.

The host is then responsible for assigning a unique address to each device on the bus. Addresses are assigned in the enumeration process, one device at a time. After the host assigns the an address to the UDC, the UDC only responds to transactions directed to that address. The Address field follows the PID in every packet transmitted.

When the UDC detects a packet that is addressed to it, it uses the Endpoint field to determine which of the UDCs endpoints is being addressed. The Endpoint field contains four bits. Encodings for endpoints 0 (0000b) through 15 (1111b) are allowed. The Endpoint field follows the Address field. Table 162 shows the valid values for the endpoint field.

Table 162. Endpoint Field Addressing

Endpoint Field Value	UDC Endpoint Selected
0000	Endpoint 0
0001	Endpoint 1
0010	Endpoint 2
0011	Endpoint 3
0100	Endpoint 4
0101	Endpoint 5
0110	Endpoint 6
0111	Endpoint 7
1000	Endpoint 8
1001	Endpoint 9
1010	Endpoint 10
1011	Endpoint 11
1100	Endpoint 12
1101	Endpoint 13
1110	Endpoint 14
1111	Endpoint 15

The Frame Number is an 11-bit field incremented by the host each time a frame is transmitted. When it reaches its maximum value (2,047 [0x7FF]) its value rolls over. Frame Number is transmitted in the SOF packet, which the host outputs in 1-ms intervals.

Device controllers use the Frame Number field to control isochronous transfers. Data fields are used to transmit the packet data between the host and the UDC. A data field consists of 0 to 1,023 bytes. Each byte is transmitted least-significant-bit-first. The UDC generates an interrupt to indicate that a Start of Frame event has occurred.

CRC fields are used to detect errors introduced during token and data-packet transmission and are applied to all the fields in the packet except the PID field. The PID contains its own 4-bit 1s complement check field for error detection. Token packets use a 5-bit CRC (x^5+x^2+1) called CRC5 and Data packets use a 16-bit CRC ($x^{16}+x^{15}+x^2+1$) called CRC16. For both CRCs, the checker resets to all 1s at the start of each packet.

18.3.4 Packet Formats

USB supports four packet types:

- Token
- Handshake
- Data
- Special

A PRE (Preamble) PID precedes a low-speed (1.5 Mbps) USB transmission. The UDC supports high-speed (12 Mbps) USB transfers only. PRE packets that signify low-speed devices and the low-speed data transfer that follows such PRE packets are ignored.

18.3.4.1 Token Packet Type

A token packet is placed at the beginning of a frame and is used to identify OUT, IN, SOF, and SETUP transactions. OUT and IN frames are used to transfer data, SOF packets are used to time isochronous transactions, and SETUP packets are used for control transfers to configure endpoints.

A Token packet consists of a sync, a PID, an address, an endpoint, and a CRC5 field (see Table 163). For OUT and SETUP transactions, the address and endpoint fields are used to select the UDC endpoint that receives the data. For an IN transaction, the address and endpoint fields are used to select the UDC endpoint that transmits data.

Table 163. IN, OUT, and SETUP Token Packet Format

8 bits	8 bits	7 bits	4 bits	5 bits
Sync	PID	Address	Endpoint	CRC5

18.3.4.2 Start-of-Frame Packet Type

A Start-of-Frame (SOF) packet is a special type of token packet that is issued by the USB host at a nominal interval of once every 1 ms +/- 0.0005 ms.

SOF packets consist of a sync, a PID, a frame number (which is incremented after each frame is transmitted), and a CRC5 field, as shown in Table 164. The presence of SOF packets every 1 ms prevents the UDC from going into suspend mode.

Table 164. SOF Token Packet Format

8 bits	8 bits	11 bits	5 bits
Sync	PID	Frame Number	CRC5

18.3.4.3 Data Packet Type

Data packets follow token packets and are used to transmit data between the USB host and UDC. There are two types of data packets as specified by the PID: DATA0 and DATA1. These two types of Data Packets are used to provide a mechanism to guarantee data sequence synchronization between the transmitter and receiver across multiple transactions.

During the handshake phase, the transmitter and receiver determine which data token type to transmit first. For each subsequent packet transmitted, the data packet type is toggled (DATA0, DATA1, DATA0, and so on).

A data packet consists of a sync; a PID; from 0 to 1,023 bytes of data; and a CRC16 field, as shown in Table 165. Note that the UDC supports a maximum of 8 bytes of data for an Interrupt IN data payload, a maximum of 64 bytes of data for a bulk data payload and a maximum of 256 bytes of data for an isochronous data payload.

Table 165. Data Packet Format

8 bits	8 bits	0 - 1,023 bytes	16 bits
Sync	PID	Data	CRC16

18.3.4.4 Handshake Packet Type

Handshake packets consist of a Sync and a PID. Handshake packets do not contain a CRC because the PID contains its own check field. Handshake packets are used to report data transaction status, including confirmation that data was successfully received, flow control, and stall conditions. Only transactions that support flow control can return handshakes.

The three types of handshake packets are: ACK, NAK, and STALL.

ACK indicates that a data packet was received without bit stuffing, CRC, or PID check errors. NAK indicates that the UDC was unable to accept data from the host or has no data to transmit. STALL indicates that the UDC was unable to transmit or receive data, and requires host intervention to clear the stall condition.

The receiving unit signals Bit stuffing, CRC, and PID errors by omitting a handshake packet. Table 166 shows the format of a handshake packet.

Table 166. Handshake Packet Format

8 bits	8 bits
Sync	PID

18.3.5 Transaction Formats

Packets are assembled into groups to form transactions. The USB protocol uses four different transaction formats. Each transaction format is specific to a particular type of endpoint: Bulk, Control, Interrupt, or Isochronous.

Endpoint 0, by default, is a control endpoint and receives only control transactions. All USB transactions are initiated by the host controller and transmitted in one direction at a time (known as half-duplex) between the host and UDC.

18.3.5.1 Bulk Transaction Type

Bulk transactions guarantee error-free data transmission between the host and UDC by using packet-error detection and retry. The host schedules bulk packets when the bus has available time.

Bulk transactions are made up of three packet types: Token, Data, and Handshake. The eight types of bulk transactions are based on data direction, error, and stall conditions.

The types of bulk transactions are shown in Table 167. Packets sent from the UDC to the host are highlighted in boldface type and packets sent from the host to the UDC are not.

Table 167. Bulk Transaction Formats

Action	Token Packet	Data Packet†	Handshake Packet†
Host successfully received data from UDC	IN	DATA0/DATA1	ACK
UDC temporarily unable to transmit data	IN	None	NAK
UDC endpoint needs host intervention	IN	None	STALL
Host detected PID, CRC, or bit stuff error	IN	DATA0/DATA1	None
UDC successfully received data from host	OUT	DATA0/DATA1	ACK
UDC temporarily unable to receive data	OUT	DATA0/DATA1	NAK
UDC endpoint needs host intervention	OUT	DATA0/DATA1	STALL
UDC detected PID, CRC, or bit stuff error	OUT	DATA0/DATA1	None

† Packets from UDC to host are **boldfaced**.

18.3.5.2 Isochronous Transaction Type

Isochronous transactions ensure constant-rate, error-tolerant transmission of data between the host and UDC. The host schedules isochronous packets during every frame.

The USB protocol allows isochronous transfers to take up to 90% of the USB bandwidth. Unlike bulk transactions, if corrupted data is received, the UDC will continue to process the corrupted data that corresponds to the current start of frame indicator.

Isochronous transactions do not support a handshake phase or retry capability. Two packet types are used to construct isochronous transactions: token and data. The types of isochronous transactions based on data direction are shown in Table 168.

Table 168. Isochronous Transaction Formats

Action	Token Packet†	Data Packet†
Host received data from UDC	IN	DATA0
UDC received data from host	OUT	DATA0

† Packets from UDC to host are **boldfaced**.

18.3.5.3 Control Transaction Type

The host uses control transactions to configure endpoints and query their status. Like bulk transactions, control transactions begin with a setup packet, followed by an optional data packet, then a handshake packet. Control transactions, by default, use DATA0 type transfers.

Table 169 shows the four types of control transactions.

Table 169. Control Transaction Formats

Action	Token Packet	Data Packet†	Handshake Packet†
UDC successfully received control from host	SETUP	DATA0	ACK
UDC temporarily unable to receive data	SETUP	DATA0	NAK
UDC endpoint needs host intervention	SETUP	DATA0	STALL
UDC detected PID, CRC, or bit stuff error	SETUP	DATA0	None

† Packets from UDC to host are **boldfaced**.

To assemble control transfers, the host sends a control transaction to tell the UDC what type of control transfer is taking place (control read or control write), followed by one or more data transactions. The setup is the first stage of the control transfer. The device must respond with an ACK or no handshake (if the data is corrupted).

The control transaction, by default, uses a DATA0 transfer and each subsequent data transaction toggles between DATA1 and DATA0 transfers. A control write to an endpoint uses OUT transactions.

Control reads use IN transactions. The transfer direction is the opposite of the last data transaction. The transfer direction is used to report status and functions as a handshake.

For a control write, the last transaction is an IN from the UDC to the host. For a control read, the last transaction is an OUT from the host to the UDC. The last data transaction always uses a DATA1 transfer, even if the previous transaction used DATA1.

18.3.5.4 Interrupt Transaction Type

The host uses interrupt transactions to query the status of the device. Like bulk transactions, interrupt transactions begin with a setup packet, followed by an optional data packet, then a handshake packet. Interrupt transactions, by default, use DATA0 type transfers.

Table 170 shows the four types of interrupt transactions.

Table 170. Interrupt Transaction Formats

Action	Token Packet	Data Packet†	Handshake Packet†
Host successfully received data from UDC	IN	DATA0	ACK
UDC temporarily unable to transmit data	IN	None	NAK
UDC endpoint needs host intervention	IN	None	STALL
Host detected PID, CRC, or bit stuff error	IN	DATA0	None

† Packets from UDC to host are **boldfaced**.

18.3.6 UDC Device Requests

The UDC uses its control, status, and data registers to control and monitor the transmit and receive FIFOs for endpoints 1 - 15. The host controls all other UDC configuration and status reporting using device requests that are sent as control transactions to endpoint 0 via the USB.

Each setup packet to Endpoint 0 is 8 bytes long and specifies:

- Data transfer direction: host to device, device to host
- Data transfer type:
  - Standard
  - Class
  - Vendor
- Data recipient:
  - Device
  - Interface
  - Endpoint
  - Other

- Number of bytes to transfer
- Index or offset
- Value: used to pass a variable-sized data parameter
- Device request

Table 171 shows a summary of all device requests. For a full description of host device requests, see the *Universal Serial Bus Specification Revision 1.1*.

Table 171. Host Device Request Summary

Request	Name
SET_FEATURE	Enables a specific feature such as device remote wake-up or endpoint stalls.
CLEAR_FEATURE	Clears or disables a specific feature.
SET_CONFIGURATION	Configures the UDC for operation. Used after a reset of the Intel XScale® processor or after a reset has been signalled via the USB.
GET_CONFIGURATION	Returns the current UDC configuration to the host.
SET_DESCRIPTOR	Sets existing descriptors or add new descriptors. Existing descriptors include: • Device•Configuration • String•Interface • Endpoint
GET_DESCRIPTOR	Returns the specified descriptor, if it exists.
SET_INTERFACE	Selects an alternate setting for the UDC's interface.
GET_INTERFACE	Returns the selected alternate setting for the specified interface.
GET_STATUS	Returns the UDC's status including: remote wake-up, self-powered, data direction, endpoint number, and stall status.
SET_ADDRESS	Sets the UDC's 7-bit address value for all future device accesses.
SYNCH_FRAME	Sets then reports an endpoint's synchronization frame.

The UDC decodes most standard device commands with no intervention required by the user. The following commands are not passed to the user are:

- Set Address
- Get Configuration
- Sync Frame
- Set Feature
- Get Status
- Clear Feature
- Get Interface

The Set Configuration and Set Interface commands are passed to the user to indicate that the host set the specified configuration or interface and the software must take any necessary actions. The Get Descriptor and Set Descriptor commands are passed to the user to be decoded.

Because the Set Feature and Clear Feature commands are not passed on, the user is not able decode the device remote wake-up feature commands. To solve this problem, the status bit UDCCS0:DRWF indicates whether or not the device remote wake-up feature is enabled. UDCCS0: DRWF is a read-only bit. When the bit is set to 1, the device remote wake-up feature is enabled. When the bit is set to 0, the feature is not enabled.

18.3.7 UDC Configuration

In response to the GET_DESCRIPTOR command, the user device sends back a description of the UDC configuration. The UDC can physically support more data channel bandwidth than the USB specification allows.

When the device responds to the host, it must specify a legal USB configuration. For example, if the device specifies a configuration of six isochronous endpoints of 256 bytes each, the host is not able to schedule the proper bandwidth and does not take the UDC out of Configuration 0. The user device determines which endpoints to report to the host. If an endpoint is not reported, it is not used.

Another option, attractive — for use with isochronous endpoints — is to describe a configuration of a packet with a maximum size less than 256 bytes to the host. For example, if software responds to the GET_DESCRIPTOR command that Endpoint 3 only supports 64 bytes maximum packet isochronous IN data, the user device must set the UDCCS3[TSP] bit after it loads 64 bytes for transmission. Similarly, if Endpoint 4 is described as supporting 128 bytes maximum packet Isochronous OUT data, the UDC recognizes the end of the packet, sets UDCCS4[RPC], and an interrupt is generated.

The direction of the endpoints is fixed. Physically, the UDC only supports interrupt endpoints with a maximum packet size of 8 bytes or less, bulk endpoints with a maximum packet size of 64 bytes or less, and isochronous endpoints with a maximum packet size of 256 bytes or less.

To make the Intel® IXP42X product line and IXC1100 control plane processors more adaptable, the UDC supports a total of four configurations. Each of these configurations are identical in the UDC, software can make three distinct configurations, each with two interfaces. Configuration 0 is a default configuration of Endpoint 0 and can not be used in any other configuration.

After the host completes a SET_CONFIGURATION or SET_INTERFACE command, the software must decode the command to empty the OUT endpoint FIFOs and allow the Intel XScale® processor to set up the proper power/peripheral configurations.

18.4 UDC Hardware Connections

This section explains how to connect the USB interface for a variety of devices.

18.4.1 Self-Powered Device

The IXP42X product line and IXC1100 control plane processors do not implement any low-power modes of operation. Therefore, the UDC can be connected to act as a full-speed device in a permanent configuration by pulling the USB_D+ signal to a logic high.

The USB_D+ signals should be pulled high by the 5-V signal received over the USB connector. The ground received from the USB connector should be tied to the ground of the board.

18.4.2 Bus-Powered Devices

The IXP42X product line and IXC1100 control plane processors do not support bus-powered devices because it is required to consume less that 500 µA when the USB host issues a suspend. (See Section 7.2.3 of the *USB Specification*, version 1.1.) The IXP42X product line and IXC1100 control plane processors cannot limit the amount of current it consumes to 500 µA.

18.5 Register Descriptions

All configuration, request/service, and status reporting is controlled by the USB host controller and is communicated to the UDC via the USB. The UDC has registers that control the interface between the UDC and the software.

A control register enables the UDC and masks the interrupt sources in the UDC. A status register indicates the state of the interrupt sources. Each of the 16 endpoints (control, OUT, and IN) have a control or status register. Endpoint 0 (control) has an address for the 16-by-8 data FIFO that can be used to transmit and receive data. Endpoint 0 also has a write count register that is used to determine the number of bytes the USB host controller has sent to Endpoint 0.

Table 172. USB-Device Register Descriptions (Sheet 1 of 2)

Address	Name	Description
0 x C800 B000	UDCCR	UDC Control Register
0 x C800 B004	Reserved	Reserved for Future Use
0 x C800 B008	Reserved	Reserved for Future Use
0 x C800 B00C	Reserved	Reserved for Future Use
0 x C800 B010	UDCCS0	UDC Endpoint 0 Control/Status Register
0 x C800 B014	UDCCS1	UDC Endpoint 1 (IN) Control/Status Register
0 x C800 B018	UDCCS2	UDC Endpoint 2 (OUT) Control/Status Register
0 x C800 B01C	UDCCS3	UDC Endpoint 3 (IN) Control/Status Register
0 x C800 B020	UDCCS4	UDC Endpoint 4 (OUT) Control/Status Register
0 x C800 B024	UDCCS5	UDC Endpoint 5 (Interrupt) Control/Status Register
0 x C800 B028	UDCCS6	UDC Endpoint 6 (IN) Control/Status Register
0 x C800 B02C	UDCCS7	UDC Endpoint 7 (OUT) Control/Status Register
0 x C800 B030	UDCCS8	UDC Endpoint 8 (IN) Control/Status Register
0 x C800 B034	UDCCS9	UDC Endpoint 9 (OUT) Control/Status Register
0 x C800 B038	UDCCS10	UDC Endpoint 10 (Interrupt) Control/Status Register
0 x C800 B03C	UDCCS11	UDC Endpoint 11 (IN) Control/Status Register
0 x C800 B040	UDCCS12	UDC Endpoint 12 (OUT) Control/Status Register
0 x C800 B044	UDCCS13	UDC Endpoint 13 (IN) Control/Status Register
0 x C800 B048	UDCCS14	UDC Endpoint 14 (OUT) Control/Status Register
0 x C800 B04C	UDCCS15	UDC Endpoint 15 (Interrupt) Control/Status Register
0 x C800 B050	UICR0	UDC Interrupt Control Register 0
0 x C800 B054	UICR1	UDC Interrupt Control Register 1
0 x C800 B058	UISR0	UDC Status Interrupt Register 0
0 x C800 B05C	UISR1	UDC Status Interrupt Register 1
0 x C800 B060	UFNHR	UDC Frame Number Register High
0 x C800 B064	UFNLR	UDC Frame Number Register Low
0 x C800 B068	UBC2	UDC Byte Count Register 2
0 x C800 B06C	UBC4	UDC Byte Count Register 4
0 x C800 B070	UBC7	UDC Byte Count Register 7
0 x C800 B074	UBC9	UDC Byte Count Register 9
0 x C800 B078	UBC12	UDC Byte Count Register 12
0 x C800 B07C	UBC14	UDC Byte Count Register 14
0 x C800 B080	UDDR0	UDC Endpoint 0 Data Register
0 x C800 B100	UDDR1	UDC Endpoint 1 Data Register
0 x C800 B180	UDDR2	UDC Endpoint 2 Data Register

Table 172. USB-Device Register Descriptions (Sheet 2 of 2)

Address	Name	Description
0 x C800 B200	UDDR3	UDC Endpoint 3 Data Register
0 x C800 B400	UDDR4	UDC Endpoint 4 Data Register
0 x C800 B0A0	UDDR5	UDC Endpoint 5 Data Register
0 x C800 B600	UDDR6	UDC Endpoint 6 Data Register
0 x C800 B680	UDDR7	UDC Endpoint 7 Data Register
0 x C800 B700	UDDR8	UDC Endpoint 8 Data Register
0 x C800 B900	UDDR9	UDC Endpoint 9 Data Register
0 x C800 B0C0	UDDR10	UDC Endpoint 10 Data Register
0 x C800 BB00	UDDR11	UDC Endpoint 11 Data Register
0 x C800 BB80	UDDR12	UDC Endpoint 12 Data Register
0 x C800 BC00	UDDR13	UDC Endpoint 13 Data Register
0 x C800 BE00	UDDR14	UDC Endpoint 14 Data Register
0 x C800 B0E0	UDDR15	UDC Endpoint 15 Data Register

18.5.1 UDC Control Register (UDCCR)

The UDC control register (UDCCR) contains seven control bits: one to enable the UDC, one to show activity, and five to show status and associated control functions.

18.5.1.1 UDC Enable

The UDC Enable (UDE) bit enables the UDC. When UDE is set to a 1, the UDC is enabled for USB serial transmission or reception. When UDE is set to a 0, the UDC is disabled and the UDC+ and UDC- pins are tristated. This means that the UDC ignores all activity on the USB bus.

If UDE is set to a 0 the entire UDC design is reset. If the reset occurs while the UDC is actively transmitting or receiving data, it stops immediately and the remaining bits in the transmit or receive serial shifter are reset.

All entries in the transmit and receive FIFO are also reset.

18.5.1.2 UDC Active

The read-only UDC Active (UDA) bit can be read to determine if the UDC is currently active or in a USB reset. This bit is only valid when the UDC is enabled. A 0 indicates that the UDC is currently receiving a USB reset from the USB Host. A 1 indicates that the UDC is currently involved in a transaction.

18.5.1.3 UDC Resume (RSM)

When the UDC is in a suspend state, this bit can be written to force the UDC into a non-idle state (K state) for 3 ms to perform a remote wake-up operation. If the host PC does not start a wake-up sequence in 3 ms, the UDC returns to the suspend mode.

This bit is a trigger bit for the UDC and is automatically cleared.

18.5.1.4 Resume Interrupt Request (RESIR)

The resume interrupt request bit is set if the SRM bit in the UDC control register is cleared, the UDC is currently in the suspended state, and the USB is driven with resume signalling.

18.5.1.5 Suspend Interrupt Request (SUSIR)

The suspend interrupt request register is set when the USB remains idle for more than 6 ms. The SUSIR bit retains state so software can determine that the USB is idle. If SRM is 0, SUSIR being set will not generate an interrupt but status continues to be updated.

18.5.1.6 Suspend/Resume Interrupt Mask (SRM)

The suspend/resume interrupt mask (SRM) masks or enables the suspend interrupt request to the interrupt controller. When SRM is 1, the interrupt is masked and the setting of SUSIR will not generate an interrupt. When SRM is 0, the setting of SUSIR generates an interrupt when the USB is idle for more than 6 ms.

Programming SRM does not affect the state of SUSIR.

18.5.1.7 Reset Interrupt Request (RSTIR)

The reset interrupt request register is set when the host issues a reset. When the host issues a reset, the entire UDC is reset. The RSTIR bit retains its state so software can determine that the design was reset. If REM is 0, RSTIR being set does not generate an interrupt but status continues to be updated.

18.5.1.8 Reset Interrupt Mask (REM)

The reset interrupt mask (REM) masks or enables the reset interrupt request to the interrupt controller. When REM is 1, the interrupt is masked and the setting of RSTIR does not generate an interrupt. When REM is 0, the RSTIR setting generates an interrupt when the USB host controller issues an UDC reset.

Programming REM does not affect the state of RSTIR.

The UDE bit is cleared to 0, which disables the UDC following a Intel XScale® processor reset. Writes to reserved bits are ignored and reads return zeros.

Register Name:	UDCCR						
Hex Offset Address:	0XC800B000		Reset Hex Value:	0x000000A0			
Register Description:	Universal Serial Bus Device Controller Control Register						
Access: Read/Write and Read-Only							

Bits																															
31																							8	7	6	5	4	3	2	1	0
(Reserved)																								REM	RSTIR	SRM	SUSIR	RESIR	RSM	UDA	UDE
X																								1	0	1	0	0	0	0	0
Resets (Above)																															

Register		UDCCR
Bits	**Name**	**Description**
31:8		Reserved for future use
7	REM	Reset interrupt mask.(read/write) 0 = Reset interrupt enabled. 1 = Reset interrupt disabled.
6	RSTIR	Reset interrupt request (read/write 1 to clear). 1 = UDC was reset by the host.
5	SRM	Suspend/resume interrupt mask (read/write). 0 = Suspend/resume interrupt enabled. 1 = Suspend/resume interrupt disabled.
4	SUSIR	Suspend interrupt request (read/write 1 to clear). 1 = UDC received suspend signalling from the host.
3	RESIR	Resume interrupt request (read/write 1 to clear). 1 = UDC received resume signalling from the host.
2	RSM	Device Resume (read/write 1 to set) 0 = Maintain UDC suspend state 1= Force UDC out of suspend
1	UDA	UDC active (read-only). 0 = UDC currently inactive. 1 = UDC currently active.
0	UDE	UDD enable.(read/write) 0 = UDD disable 1 = UDD enabled, UDC+ and UDC- used for USB serial transmission/reception.

18.5.2 UDC Endpoint 0 Control/Status Register (UDCCS0)

The UDC endpoint 0 control/status register contains seven bits that are used to operate endpoint 0 (control endpoint).

18.5.2.1 OUT Packet Ready (OPR)

The OUT packet ready bit is set by the UDC when it receives a valid OUT packet to endpoint 0. When this bit is set, the USIR0[IR0] bit will be set in the UDC status/ interrupt register if endpoint zero interrupts are enabled. This bit is cleared by writing a 1. The UDC is not allowed to enter the data phase of a transaction until this bit is cleared.

18.5.2.2 IN Packet Ready (IPR)

The IN packet ready bit is set by the Intel XScale® processor if less than max_packet bytes (16) have been written to the endpoint 0 FIFO to be transmitted. The Intel XScale® processor must not set this bit if a max_packet is to be transmitted. The UDC clears this bit when the packet has been successfully transmitted, the UDCCS0[FTF] bit has been set, or a control OUT is received. When this bit is cleared due to a successful IN transmission or the reception of a control OUT, the USIR0[IR0] bit in the UDC interrupt register is set if the endpoint 0 interrupt is enabled via UICR0[IM0]. The Intel XScale® processor is not able to clear UDCCS0[IPR] and always reads back a 0.

When software enables the status stage for Vendor/Class commands and control data commands such as GET_DESCRIPTOR, GET_CONFIGURATION, GET_INTERFACE, GET_STATUS, and SET_DECSCRIPTOR, software must also set IPR. The data in the Transmit FIFO must be transmitted and the interrupt must be processed before the IPR is set for the status stage.

The status stage for all other USB Standard Commands that do not have a data stage, such as SET_ADDRESS, SET_CONFIGURATION, SET_INTERFACE, SET_FEATURE, and CLEAR_FEATURE, is handled by the UDC and the software must not set IPR.

18.5.2.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers the reset of the endpoint 0 transmit FIFO. It is set when software writing a 1 or when the UDC receives an OUT packet from the host on endpoint 0. This bit always reads back a 0 value.

18.5.2.4 Device Remote Wake-Up Feature (DRWF)

The host indicates the state of the device remote wake-up feature by sending a Set Feature command or a Clear Function command. The UDC decodes the command sent by the host and sets this bit to a 1 if the feature is enabled and a 0 if the feature is disabled.

This bit is read-only.

18.5.2.5 Sent Stall (SST)

The sent stall bit is set by the UDC when FST successfully forces a software-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host when a STALL handshake is returned automatically. In this event, there is no intervention by the Intel XScale® processor and the UDC clears the STALL status before the host sends the next SETUP command.

When the UDC sets this bit, the transmit FIFO is flushed. The Intel XScale® processor writes a 1 to this bit to clear it.

18.5.2.6 Force Stall (FST)

The force stall bit can be set by the Intel XScale® processor to force the UDC to issue a STALL handshake. The UDC issues a STALL handshake for the current setup control transfer and the bit is cleared by the UDC because endpoint 0 can not remain in a stalled condition.

18.5.2.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that the receive FIFO contains unread data. To determine if the FIFO has data in it, this bit must be read when the UDCCS0[OPR] bit is set. The receive FIFO must continue to be read until this bit clears or the data will be lost.

If UDCCS0[RNE] is not set when an interrupt generated by UDCCS0[OPR] is initially serviced, it indicates that a zero-length OUT packet was received.

18.5.2.8 Setup Active (SA)

The Setup Active bit indicates that the current packet in the FIFO is part of a USB setup command. This bit generates an interrupt and becomes active at the same time as UDCCS0[OPR]. Software must clear this bit by writing a 1 to it. Both UDCS0[OPR] and UDCCS0[SA] must be cleared.

Register Name:	UDCCS0				
Hex Offset Address:	0 x C800B010		Reset Hex Value:	0 x 00000000	
Register Description:	Universal Serial Bus Device Controller Endpoint 0 Control and Status Register				
Access: Read/Write					

Bits																															
31																						8	7	6	5	4	3	2	1	0
(Reserved)																							SA	RNE	FST	SST	DRWF	FTF	IPR	OPR
X																							0	0	0	0	0	0	0	0
Resets (Above)																															

Register		UDCCS0
Bits	Name	Description
31:8		Reserved for future use
7	SA	Setup Active (read/write 1 to clear) 1 = Setup command is active on the USB
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5	FST	Force stall (read/write 1 to set). 1 = Force stall handshake
4	SST	Sent stall (read/write 1 to clear). 1 = UDC sent stall handshake
3	DRWF	Device remote wake-up feature (read-only) 0 = Device Remote Wake-Up Feature is disabled. 1 = Device Remote Wake-Up Feature is enabled.
2	FTF	Flush Tx FIFO (always read 0/write 1 to set) 1 = Flush the contents of Tx FIFO.
1	IPR	IN packet ready (always read 0/write 1 to set). 1 = IN packet ready.
0	OPR	OUT packet ready (read/write 1 to clear) 1 = OUT packet ready.

18.5.3 UDC Endpoint 1 Control/Status Register (UDCCS1)

The UDC Endpoint 1control status register contains 6 bits that are used to operate endpoint 1, a Bulk IN endpoint.

18.5.3.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is active if one or fewer data packets remain in the transmit FIFO. TFS is cleared when two complete packets of data remain in the FIFO. A complete packet of data is signified by loading 64 bytes of data or by setting UDCCS1[TSP].

18.5.3.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR1 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled. This bit can be used to validate the other status/error bits in the endpoint 1control/status register.

The UDCCS1[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR1 bit in the appropriate UDC status/interrupt register, but the IR1 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS1[TSP].

18.5.3.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

18.5.3.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS1[TUR] does not generate an interrupt and is for status only. UDCCS1[TUR] is cleared by writing a 1 to it.

18.5.3.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command. The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS1[SST] bit is set.

To allow the software to continue to send the STALL condition on the USB bus, the UDCCS1[FST] bit must be set again. The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.3.6 Force STALL (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS1[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS1[FST] bit is set. The UDCCS1[FST] bit is automatically cleared when the UDCCS1[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS1[FTF] bit.

18.5.3.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.3.8 Transmit Short Packet (TSP)

The software uses the transmit short packet bit to indicate that the last byte of a data transfer to the FIFO has occurred. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a 64-byte packet is to be transmitted. When the data packet is successful transmitted, the UDC clears this bit.

Register Name:	UDCCS1																															
Hex Offset Address:	0 x C800B014				Reset Hex Value:	0 x 00000001																										
Register Description:	Universal Serial Bus Device Controller Endpoint 1 Control and Status Register																															
Access: Read/Write																																
Bits																																
31																								8	7	6	5	4	3	2	1	0
(Reserved)																									TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS
X																									0	0	0	0	0	0	0	1
Resets (Above)																																

Register		UDCCS1
Bits	Name	Description
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear) 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data 1 = Transmit FIFO has room for at least 1 complete data packet

18.5.4 UDC Endpoint 2 Control/Status Register (UDCCS2)

The UDC Endpoint 2 Control/Status Register contains seven bits that are used to operate endpoint 2, a Bulk OUT endpoint.

18.5.4.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 64 bytes, a short packet, or a zero packet. This bit is not cleared until all data has been read from both buffers.

18.5.4.2 Receive Packet Complete (RPC)

The receive packet complete bit is set by the UDC when an OUT packet is received. When this bit is set, the IR2 bit in the appropriate UDC status/interrupt register is set, if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 2 Control/ Status Register. The UDCCS2[RPC] bit is cleared by writing a 1 to it. The UDC issues NAK handshakes to all OUT tokens while this bit is set and both buffers have unread data.

18.5.4.3 Bit 2 Reserved

Bit 2 is reserved for future use.

18.5.4.4 Bit 2 Reserved

Bit 3 is reserved for future use.

18.5.4.5 Sent Stall (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

Any valid data in the FIFO remains valid and the software must unload it. The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS2[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS2[FST] bit must be set again.

The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.4.6 Force Stall (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all OUT tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS2[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS2[FST] bit is set. The UDCCS2[FST] bit is automatically cleared when the UDCCS2[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS2[FTF] bit.

18.5.4.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that unread data remains in the receive FIFO.

This bit must be polled when the UDCCS2[RPC] bit is set to determine if there is any data in the FIFO that the Intel XScale® processor did not read. The receive FIFO must continue to be read until this bit clears or data will be lost.

18.5.4.8 Receive Short Packet (RSP)

The UDC uses the receive short packet bit to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS2[RSP] is a 1 and UDCCS2[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale® processor must not read the data register. UDCCS2[RSP] is cleared when the next OUT packet is received.

Register Name:	**UDCCS2**																															
Hex Offset Address:	0 x C800 B018																**Reset Hex Value:**		0 x 00000000												
Register Description:	Universal Serial Bus Device Controller Endpoint 2 Control and Status Register																															
Access: Read/Write																																
Bits																																
31																															**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																															RSP	RNE	FST	SST	DME	(Rsvd)	RPC	RFS
X																															0	0	0	0	0	0	0	0
Resets (Above)																																

Register		**UDCCS2**
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5	FST	Force stall (read/write). 1 = Issue STALL handshakes to OUT tokens.
4	SST	Sent stall (read/write 1 to clear). 1 = STALL handshake was sent.
3		(Reserved)
2		(Reserved). Always reads zero.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than one data packet. 1 = Receive FIFO has 1 or more data packets.

18.5.5 UDC Endpoint 3 Control/Status Register (UDCCS3)

The UDC endpoint 3control status register contains four bits that are used to operate endpoint 3, an Isochronous IN endpoint.

18.5.5.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is be set if one or fewer data packets remain in the transmit FIFO. UDCCS3[TFS] is cleared when two complete data packets are in the FIFO.

A complete packet of data is signified by loading 256 bytes or by setting UDCCS3[TSP].

18.5.5.2 Transmit Packet Complete (TPC)

The UDC sets transmit packet complete bit when an entire packet is sent to the host. When this bit is set, the IR3 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the endpoint 3control/ status register. The UDCCS3[TPC] bit gets cleared by writing a 1 to it. This clears the interrupt source for the IR3 bit in the appropriate UDC status/interrupt register, but the IR3 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS3[TSP].

18.5.5.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

18.5.5.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, UDCCS3[TUR] generates an interrupt.

UDCCS3[TUR] is cleared by writing a 1 to it.

18.5.5.5 Bit 4 Reserved

Bit 4 is reserved for future use.

18.5.5.6 Bit 5 Reserved

Bit 5 is reserved for future use.

18.5.5.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.5.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a packet of 256 bytes is to be transmitted. When the data packet is successfully transmitted, this bit is cleared by the UDC.

Register Name:	UDCCS3																													
Hex Offset Address:	0 x C800 B01C															Reset Hex Value:	0 x 00000001												
Register Description:	Register Description: Universal Serial Bus Device Controller Endpoint 3 Control and Status Register																													
Access: Read/Write																														
Bits																														
31																							8	7	6	5	4	3	2	1	0
(Reserved)																								TSP	(Rsvd)	(Rsvd)	(Rsvd)	TUR	FTF	TPC	TFS
																								0	0	0	0	0	0	0	1
Resets (Above)																														

Register		UDCCS3
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least 1 complete data packet.

18.5.6 UDC Endpoint 4 Control/Status Register (UDCCS4)

The UDC endpoint 4 control/status register contains six bits that are used to operate endpoint 4, an Isochronous OUT endpoint.

18.5.6.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 256 bytes, a short packet, or a zero packet.

UDCCS4[RFS] is not cleared until all data is read from both buffers.

18.5.6.2 Receive Packet Complete (RPC)

The receive packet complete bit gets set by the UDC when an OUT packet is received. When this bit is set, the IR4 bit in the appropriate UDC status/interrupt register is set if receive interrupts are enabled. This bit can be used to validate the other status/error bits in the endpoint 4 control/status register.

The UDCCS4[RPC] bit is cleared by writing a 1 to it.

18.5.6.3 Receive Overflow (ROF)

The receive overflow bit generates an interrupt on IR4 in the appropriate UDC status/interrupt register to alert the software that Isochronous data packets are being dropped because neither FIFO buffer has room for them.

This bit is cleared by writing a 1 to it.

18.5.6.4 Bit 3 Reserved

Bit 3 is reserved for future use.

18.5.6.5 Bit 4 Reserved

Bit 4 is reserved for future use.

18.5.6.6 Bit 5 Reserved

Bit 5 is reserved for future use.

18.5.6.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that the receive FIFO has unread data in it. When the UDCCS4[RPC] bit is set, this bit must be read to determine if there is any data in the FIFO that Intel XScale® processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

18.5.6.8 Receive Short Packet (RSP)

The receive short packet bit is used by the UDC to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet.

This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer. If UDCCS4[RSP] is a 1 and UDCCS4[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale® processor must not read the data register.

UDCCS4[RSP] clears when the next OUT packet is received.

Register Name:	UDCCS4																															
Hex Offset Address:	0x C800B020															Reset Hex Value:	0x00000000															
Register Description:	Universal Serial Bus Device Controller Endpoint 4 Control and Status Register																															
Access: Read/Write																																
Bits																																
31																								8	7	6	5	4	3	2	1	0
(Reserved)																									RSP	RNE	(Rsvd)	(Rsvd)	(Rsvd)	ROF	RPC	RFS
X																									0	0	0	0	0	0	0	0
Resets (Above)																																

Register		UDCCS4
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3		(Reserved)
2	ROF	Receive overflow (read/write 1 to clear). 1 = Isochronous data packets are being dropped from the host because the receiver is full.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than 1 data packet. 1 = Receive FIFO has 1 or more data packets.

18.5.7 UDC Endpoint 5 Control/Status Register (UDCCS5)

The UDC Endpoint 5 Control Status Register contains six bits that are used to operate endpoint 5, an Interrupt IN endpoint.

18.5.7.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if the FIFO does not contain any data bytes and UDCCS5[TSP] is not set.

18.5.7.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is be set by the UDC when an entire packet is sent to the host. When this bit is set, the IR5 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 5 Control/ Status Register.

The UDCCS5[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR5 bit in the appropriate UDC status/interrupt register, but the IR5 bit must also be cleared.

The UDC issues NAK handshakes to all IN tokens if this bit is set and the buffer is not triggered by writing 8 bytes or setting UDCCS5[TSP].

18.5.7.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO.

The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE. The bit's read value is 0.

18.5.7.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host.

UDCCS5[TUR] does not generate an interrupt and is for status only. UDCCS5[TUR] is cleared by writing a 1 to it.

18.5.7.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS5[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS5[FST] bit must be set again.

The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.7.6 Force STALL (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS5[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS5[FST] bit is set. The UDCCS5[FST] bit is automatically cleared when the UDCCS5[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS5[FTF] bit.

18.5.7.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.7.8 Transmit Short Packet (TSP)

Software uses the transmit short to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a packet of 8 bytes is to be transmitted. When the data packet is successfully transmitted, the UDC clears this bit.

Register Name:	UDCCS5																											
Hex Offset Address:	0x C800B024				Reset Hex Value:	0x00000001																						
Register Description:	Universal Serial Bus Device Controller Endpoint 5 Control and Status Register																											
Access: Read/Write																												
Bits																												
31																						8	7	6	5	4	3	2	1	0
(Reserved)																							TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS
X																							0	0	0	0	0	0	0	1
Resets (Above)																														

Register		UDCCS5
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for 1 complete data packet.

18.5.8 UDC Endpoint 6 Control/Status Register

(UDCCS6)

The UDC Endpoint 6 Control Status Register contains six bits that are used to operate endpoint 6, a Bulk IN endpoint.

18.5.8.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is active if one or fewer data packets remain in the transmit FIFO. TFS is cleared when two complete packets of data remain in the FIFO.

A complete packet of data is signified by loading 64 bytes of data or by setting UDCCS6[TSP].

18.5.8.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR6 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled. This bit can be used to validate the other status/error bits in the Endpoint 6 Control/Status Register.

The UDCCS6[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR6 bit in the appropriate UDC status/interrupt register, but the IR6 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS6[TSP].

18.5.8.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO.

The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE. The bit's read value is 0.

18.5.8.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS6[TUR] does not generate an interrupt and is for status only.

UDCCS6[TUR] is cleared by writing a 1 to it.

18.5.8.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS6[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS6[FST] bit must be set again. The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.8.6 Force STALL (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS6[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS6[FST] bit is set. The UDCCS6[FST] bit is automatically cleared when the UDCCS6[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS6[FTF] bit.

18.5.8.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.8.8 Transmit Short Packet (TSP)

The software uses the transmit short packet bit to indicate that the last byte of a data transfer to the FIFO has occurred. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a 64-byte packet is to be transmitted. When the data packet is successful transmitted, the UDC clears this bit.

Register Name:	**UDCCS6**																													
Hex Offset Address:	0 x C800B028													**Reset Hex Value:**								0 x 00000001								
Register Description:	Universal Serial Bus Device Controller Endpoint 6Control and Status Register																													
Access: Read/Write																														
Bits																														
31																						**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																							TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS
X																							0	0	0	0	0	0	0	1
Resets (Above)																														

Register		**UDCCS6**
Bits	**Name**	**Description**
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least 1 complete data packet.

18.5.9 UDC Endpoint 7 Control/Status Register (UDCCS7)

The UDC Endpoint 7 Control/Status Register contains seven bits that are used to operate endpoint 7, a Bulk OUT endpoint.

18.5.9.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 64 bytes, a short packet, or a zero packet.

This bit is not cleared until all data has been read from both buffers.

18.5.9.2 Receive Packet Complete (RPC)

The receive packet complete bit is set by the UDC when an OUT packet is received. When this bit is set, the IR7 bit in the appropriate UDC status/interrupt register is set — if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 7 Control/ Status Register. The UDCCS7 [RPC] bit is cleared by writing a 1 to it. The UDC issues NAK handshakes to all OUT tokens while this bit is set and both buffers have unread data.

18.5.9.3 Bit 2 Reserved

Bit 2 is reserved for future use.

18.5.9.4 Bit 3 Reserved

Bit 3 is reserved for future use.

18.5.9.5 Sent Stall (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

Any valid data in the FIFO remains valid and the software must unload it. The end-point operation continues normally and does not send another STALL condition, even if the UDCCS7[SST] bit is set.

To allow the software to continue to send the STALL condition on the USB bus, the UDCCS7[FST] bit must be set again.

The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.9.6 Force Stall (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all OUT tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS7[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS7[FST] bit is set. The UDCCS7[FST] bit is automatically cleared when the UDCCS7[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS7[FTF] bit.

18.5.9.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that unread data remains in the receive FIFO. This bit must be polled when the UDCCS7[RPC] bit is set to determine if there is any data in the FIFO that the Intel XScale® processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

18.5.9.8 Receive Short Packet (RSP)

The UDC uses the receive-short-packet bit to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS7[RSP] is a 1 and UDCCS7[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale® processor must not read the data register.

UDCCS7[RSP] is cleared when the next OUT packet is received.

Register Name:	**UDCCS7**																								
Hex Offset Address:	0 x C800 B02C													**Reset Hex Value:**				0 x 00000000							
Register Description:	Universal Serial Bus Device Controller Endpoint 7Control and Status Register																								
Access: Read/Write																									
Bits																									
31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																						RSP	RNE	FST	SST	(Rsvd)	(Rsvd)	RPC	RFS
X																						0	0	0	0	0	0	0	0
Resets (Above)																									

Register		**UDCCS7** (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5	FST	Force stall (read/write). 1 = Issue STALL handshakes to OUT tokens.
4	SST	Sent stall (read/write 1 to clear). 1 = STALL handshake was sent.
3		(Reserved)

Register		UDCCS7 (Sheet 2 of 2)
Bits	Name	Description
2		(Reserved). Always reads zero.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than 1 data packet. 1 = Receive FIFO has 1 or more data packets.

18.5.10 UDC Endpoint 8 Control/Status Register (UDCCS8)

The UDC Endpoint 8 Control Status Register contains four bits that are used to operate endpoint 8, an isochronous IN endpoint.

18.5.10.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is be set if one or fewer data packets remain in the transmit FIFO. UDCCS8[TFS] is cleared when two complete data packets are in the FIFO. A complete packet of data is signified by loading 256 bytes or by setting UDCCS8[TSP].

18.5.10.2 Transmit Packet Complete (TPC)

The the UDC sets transmit packet complete bit when an entire packet is sent to the host. When this bit is set, the IR8 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 8 Control/ Status Register. The UDCCS8[TPC] bit gets cleared by writing a 1 to it. This clears the interrupt source for the IR8 bit in the appropriate UDC status/interrupt register, but the IR8 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS8[TSP].

18.5.10.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

18.5.10.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, UDCCS8[TUR] generates an interrupt.

UDCCS8[TUR] is cleared by writing a 1 to it.

18.5.10.5 Bit 4 Reserved

Bit 4 is reserved for future use.

18.5.10.6 Bit 5 Reserved

Bit 5 is reserved for future use.

18.5.10.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.10.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a packet of 256 bytes is to be transmitted. When the data packet is successfully transmitted, this bit is cleared by the UDC.

Register Name:	**UDCCS8**																													
Hex Offset Address:	0 x C800 B030															**Reset Hex Value:**	0 x 00000001											
Register Description:	Register Description: Universal Serial Bus Device Controller Endpoint 8Control and Status Register																													
Access: Read/Write																														
Bits																														
31																							**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																								TSP	(Rsvd)	(Rsvd)	(Rsvd)	TUR	FTF	TPC	TFS
																								0	0	0	0	0	0	0	1
Resets (Above)																														

Register		**UDCCS8**
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1= Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least 1 complete data packet.

18.5.11 UDC Endpoint 9 Control/Status Register (UDCCS9)

The UDC Endpoint 9 Control/Status Register contains six bits that are used to operate endpoint 9, an isochronous OUT endpoint.

18.5.11.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 256 bytes, a short packet, or a zero packet. UDCCS9[RFS] is not cleared until all data is read from both buffers.

18.5.11.2 Receive Packet Complete (RPC)

The receive packet complete bit gets set by the UDC when an OUT packet is received. When this bit is set, the IR9 bit in the appropriate UDC status/interrupt register is set if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 9 Control/ Status Register. The UDCCS9[RPC] bit is cleared by writing a 1 to it.

18.5.11.3 Receive Overflow (ROF)

The receive overflow bit generates an interrupt on IR9 in the appropriate UDC status/ interrupt register to alert the software that Isochronous data packets are being dropped because neither FIFO buffer has room for them. This bit is cleared by writing a 1 to it.

18.5.11.4 Bit 3 Reserved

Bit 3 is reserved for future use.

18.5.11.5 Bit 4 Reserved

Bit 4 is reserved for future use.

18.5.11.6 Bit 5 Reserved

Bit 5 is reserved for future use.

18.5.11.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that the receive FIFO has unread data in it. When the UDCCS9[RPC] bit is set, this bit must be read to determine if there is any data in the FIFO that Intel XScale® processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

18.5.11.8 Receive Short Packet (RSP)

The receive short packet bit is used by the UDC to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS9[RSP] is a one and UDCCS9[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale® processor must not read the data register.

UDCCS9[RSP] clears when the next OUT packet is received.

Register Name:	UDCCS9				
Hex Offset Address:	0x C800B034		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Controller Endpoint 9Control and Status Register				
Access: Read/Write					

Bits																															
31																						8	7	6	5	4	3	2	1	0
(Reserved)																							RSP	RNE	(Rsvd)	(Rsvd)	(Rsvd)	ROF	RPC	RFS
X																							0	0	0	0	0	0	0	0
Resets (Above)																															

Register	UDCCS9	
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3		(Reserved)
2	ROF	Receive overflow (read/write 1 to clear). 1 = Isochronous data packets are being dropped from the host because the receiver is full.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than one data packet. 1 = Receive FIFO has one or more data packets.

18.5.12 UDC Endpoint 10 Control/Status Register (UDCCS10)

The UDC Endpoint 10 Control Status Register contains six bits that are used to operate endpoint 10, an Interrupt IN endpoint.

18.5.12.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if the FIFO does not contain any data bytes and UDCCS10[TSP] is not set.

18.5.12.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is be set by the UDC when an entire packet is sent to the host. When this bit is set, the IR10 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 10 Control/ Status Register. The UDCCS10[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR10 bit in the appropriate UDC status/interrupt register, but the IR10 bit must also be cleared.

The UDC issues NAK handshakes to all IN tokens if this bit is set and the buffer is not triggered by writing 8 bytes or setting UDCCS10[TSP].

18.5.12.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

18.5.12.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS10[TUR] does not generate an interrupt and is for status only. UDCCS10[TUR] is cleared by writing a 1 to it.

18.5.12.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS10[SST] bit is set.

To allow the software to continue to send the STALL condition on the USB bus, the UDCCS10[FST] bit must be set again. The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.12.6 Force STALL (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS10[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS10[FST] bit is set. The UDCCS10[FST] bit is automatically cleared when the UDCCS10[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS10[FTF] bit.

18.5.12.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.12.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a packet of 8 bytes is to be transmitted. When the data packet is successfully transmitted, the UDC clears this bit.

Register Name:	UDCCS10				
Hex Offset Address:	0x C800B038		Reset Hex Value:	0x00000001	
Register Description:	Universal Serial Bus Device Controller Endpoint 10Control and Status Register				
Access: Read/Write					

Bits																															
31																						8	7	6	5	4	3	2	1	0
(Reserved)																							TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS
X																							0	0	0	0	0	0	0	1
Resets (Above)																															

Register		UDCCS10
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for 1 complete data packet.

18.5.13 UDC Endpoint 11 Control/Status Register (UDCCS11)

The UDC Endpoint 11 Control Status Register contains six bits that are used to operate endpoint 11, a Bulk IN endpoint.

18.5.13.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is active if one or fewer data packets remain in the transmit FIFO. TFS is cleared when two complete packets of data remain in the FIFO. A complete packet of data is signified by loading 64 bytes of data or by setting UDCCS11[TSP].

18.5.13.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR11 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 11 Control/ Status Register. The UDCCS11[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR11 bit in the appropriate UDC status/interrupt register, but the IR11 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS11[TSP].

18.5.13.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

18.5.13.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS11[TUR] does not generate an interrupt and is for status only.

UDCCS11[TUR] is cleared by writing a 1 to it.

18.5.13.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS11[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS11[FST] bit must be set again.

The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.13.6 Force STALL (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS11[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS11[FST] bit is set. The UDCCS11[FST] bit is automatically cleared when the UDCCS11[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS11[FTF] bit.

18.5.13.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.13.8 Transmit Short Packet (TSP)

The software uses the transmit short packet bit to indicate that the last byte of a data transfer to the FIFO has occurred. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a 64-byte packet is to be transmitted. When the data packet is successful transmitted, the UDC clears this bit.

Register Name:	**UDCCS11**																															
Hex Offset Address:	0 x C800B03C																					**Reset Hex Value:**								0 x 00000001		
Register Description:	Universal Serial Bus Device Controller Endpoint 11 Control and Status Register																															
Access: Read/Write																																
Bits																																
31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**			
(Reserved)																						TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS			
X																						0	0	0	0	0	0	0	1			
Resets (Above)																																

Register		**UDCCS11** (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.

Register		UDCCS11 (Sheet 2 of 2)
Bits	Name	Description
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least 1 complete data packet.

18.5.14 UDC Endpoint 12 Control/Status Register (UDCCS12)

The UDC Endpoint 12 Control/Status Register contains seven bits that are used to operate endpoint 12, a Bulk OUT endpoint.

18.5.14.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 64 bytes, a short packet, or a zero packet.

This bit is not cleared until all data has been read from both buffers.

18.5.14.2 Receive Packet Complete (RPC)

The receive packet complete bit is set by the UDC when an OUT packet is received. When this bit is set, the IR12 bit in the appropriate UDC status/interrupt register is set, if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the endpoint 12 control/status register. The UDCCS12[RPC] bit is cleared by writing a 1 to it. The UDC issues NAK handshakes to all OUT tokens while this bit is set and both buffers have unread data.

18.5.14.3 Bit 2 Reserved

Bit 2 is reserved for future use.

18.5.14.4 Bit 3 Reserved

Bit 3 is reserved for future use.

18.5.14.5 Sent Stall (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

Any valid data in the FIFO remains valid and the software must unload it. The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS12[SST] bit is set.

To allow the software to continue to send the STALL condition on the USB bus, the UDCCS12[FST] bit must be set again. The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.14.6 Force Stall (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all OUT tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS12[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS12[FST] bit is set. The UDCCS12[FST] bit is automatically cleared when the UDCCS12[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS12[FTF] bit.

18.5.14.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that unread data remains in the receive FIFO. This bit must be polled when the UDCCS12[RPC] bit is set to determine if there is any data in the FIFO that the Intel XScale® processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

18.5.14.8 Receive Short Packet (RSP)

The UDC uses the receive short packet bit to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS12[RSP] is a 1 and UDCCS12[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale® processor must not read the data register. UDCCS12[RSP] is cleared when the next OUT packet is received.

Register Name:	**UDCCS12**																															
Hex Offset Address:	0 x C800 B040																					**Reset Hex Value:**	0 x 00000000							
Register Description:	Universal Serial Bus Device Controller Endpoint 12 Control and Status Register																															
Access: Read/Write																																
Bits																																
31																						**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
(Reserved)																							RSP	RNE	FST	SST	(Rsvd)	(Rsvd)	RPC	RFS		
X																							0	0	0	0	0	0	0	0		
Resets (Above)																																

Register		UDCCS12
Bits	Name	Description
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5	FST	Force stall (read/write). 1 = Issue STALL handshakes to OUT tokens.
4	SST	Sent stall (read/write 1 to clear). 1 = STALL handshake was sent.
3		(Reserved)
2		(Reserved). Always reads zero.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than 1 data packet. 1 = Receive FIFO has 1 or more data packets.

18.5.15 UDC Endpoint 13 Control/Status Register (UDCCS13)

The UDC Endpoint 13 Control Status Register contains four bits that are used to operate endpoint 13, an Isochronous IN endpoint.

18.5.15.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is be set if one or fewer data packets remain in the transmit FIFO. UDCCS13[TFS] is cleared when two complete data packets are in the FIFO. A complete packet of data is signified by loading 256 bytes or by setting UDCCS13[TSP].

18.5.15.2 Transmit Packet Complete (TPC)

The the UDC sets transmit packet complete bit when an entire packet is sent to the host. When this bit is set, the IR13 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the endpoint 13 Control/ Status Register. The UDCCS13[TPC] bit gets cleared by writing a 1 to it. This clears the interrupt source for the IR13 bit in the appropriate UDC status/interrupt register, but the IR13 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS13[TSP].

18.5.15.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is zero.

18.5.15.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, UDCCS13[TUR] generates an interrupt.

UDCCS13[TUR] is cleared by writing a 1 to it.

18.5.15.5 Bit 4 Reserved

Bit 4 is reserved for future use.

18.5.15.6 Bit 5 Reserved

Bit 5 is reserved for future use.

18.5.15.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.15.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit, if a packet of 256 bytes is to be transmitted. When the data packet is successfully transmitted, this bit is cleared by the UDC.

Register Name:	**UDCCS13**																															
Hex Offset Address:	0 x C800 B044																					**Reset Hex Value:**								0 x 00000001		
Register Description:	Register Description: Universal Serial Bus Device Controller Endpoint 13 Control and Status Register																															
Access: Read/Write																																
Bits																																
31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**			
(Reserved)																						TSP	(Rsvd)	(Rsvd)	(Rsvd)	TUR	FTF	TPC	TFS			
																						0	0	0	0	0	0	0	1			
Resets (Above)																																

Register		**UDCCS13** (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.

Register		UDCCS13 (Sheet 2 of 2)
Bits	**Name**	**Description**
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least one complete data packet.

18.5.16 UDC Endpoint 14 Control/Status Register (UDCCS14)

The UDC Endpoint 14 Control/Status Register contains six bits that are used to operate endpoint 14, an Isochronous OUT endpoint.

18.5.16.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 256 bytes, a short packet, or a zero packet.

UDCCS14[RFS] is not cleared until all data is read from both buffers.

18.5.16.2 Receive Packet Complete (RPC)

The receive packet complete bit gets set by the UDC when an OUT packet is received. When this bit is set, the IR14 bit in the appropriate UDC status/interrupt register is set if receive interrupts are enabled. This bit can be used to validate the other status/error bits in the endpoint 14 control/status register.

The UDCCS14[RPC] bit is cleared by writing a 1 to it.

18.5.16.3 Receive Overflow (ROF)

The receive overflow bit generates an interrupt on IR14 in the appropriate UDC status/interrupt register to alert the software that Isochronous data packets are being dropped because neither FIFO buffer has room for them.

This bit is cleared by writing a 1 to it.

18.5.16.4 Bit 3 Reserved

Bit 3 is reserved for future use.

18.5.16.5 Bit 4 Reserved

Bit 4 is reserved for future use.

18.5.16.6 Bit 5 Reserved

Bit 5 is reserved for future use.

18.5.16.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that the receive FIFO has unread data in it. When the UDCCS14[RPC] bit is set, this bit must be read to determine if there is any data in the FIFO that Intel XScale® processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

18.5.16.8 Receive Short Packet (RSP)

The receive short packet bit is used by the UDC to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS14[RSP] is a 1 and UDCCS14[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale® processor must not read the data register.

UDCCS14[RSP] clears when the next OUT packet is received.

Register Name:	**UDCCS14**																															
Hex Offset Address:	0x C800B048																	**Reset Hex Value:**										0x00000000				
Register Description:	Universal Serial Bus Device Controller Endpoint 14 Control and Status Register																															
Access: Read/Write																																
Bits																																
31																							**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																								RSP	RNE	(Rsvd)	(Rsvd)	(Rsvd)	ROF	RPC	RFS
X																								0	0	0	0	0	0	0	0
Resets (Above)																																

Register		**UDCCS14** (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3		(Reserved)

Register		UDCCS14 (Sheet 2 of 2)
Bits	**Name**	**Description**
2	ROF	Receive overflow (read/write 1 to clear). 1 = Isochronous data packets are being dropped from the host because the receiver is full.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1= Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than one data packet. 1 = Receive FIFO has one or more data packets.

18.5.17 UDC Endpoint 15 Control/Status Register (UDCCS15)

The UDC Endpoint 15 Control Status Register contains six bits that are used to operate endpoint 15, an Interrupt IN endpoint.

18.5.17.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if the FIFO does not contain any data bytes and UDCCS15[TSP] is not set.

18.5.17.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is be set by the UDC when an entire packet is sent to the host. When this bit is set, the IR15 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 15 Control/ Status Register.

The UDCCS15[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR15 bit in the appropriate UDC status/interrupt register, but the IR15 bit must also be cleared.

The UDC issues NAK handshakes to all IN tokens if this bit is set and the buffer is not triggered by writing 8 bytes or setting UDCCS15[TSP].

18.5.17.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

18.5.17.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS15[TUR] does not generate an interrupt and is for status only.

UDCCS15[TUR] is cleared by writing a 1 to it.

18.5.17.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale® processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS15[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS15[FST] bit must be set again.

The Intel XScale® processor writes a 1 to the sent stall bit to clear it.

18.5.17.6 Force STALL (FST)

The Intel XScale® processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale® processor clears this bit by sending a Clear Feature command.

The UDCCS15[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS15[FST] bit is set. The UDCCS15[FST] bit is automatically cleared when the UDCCS15[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS15[FTF] bit.

18.5.17.7 Bit 6 Reserved

Bit 6 is reserved for future use.

18.5.17.8 Transmit Short Packet (TSP)

Software uses the transmit short to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software must not set this bit if a packet of 8 bytes is to be transmitted.

When the data packet is successfully transmitted, the UDC clears this bit.

Register Name:	**UDCCS15**																													
Hex Offset Address:	0x C800B04C													**Reset Hex Value:**								0x00000001								
Register Description:	Universal Serial Bus Device Controller Endpoint 15 Control and Status Register																													
Access: Read/Write																														
Bits																														
31																						**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																							TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS
X																							0	0	0	0	0	0	0	1
Resets (Above)																														

Register		UDCCS15
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for 1 complete data packet.

18.5.18 UDC Interrupt Control Register 0 (UICR0)

The UICR0 contains eight control bits to enable/disable interrupt service requests from data endpoints 0 - 7. All of the UICR0 bits are reset to a 1 so interrupts are not generated on initial system reset.

18.5.18.1 Interrupt Mask Endpoint x (IMx), Where x is 0 through 7

The UICR0[IMx] bit is used to mask or enable the corresponding endpoint interrupt request, USIR0[IRx]. When the mask bit is set, the interrupt is masked and the corresponding bit in the USIR0 register is not allowed to be set.

When the mask bit is cleared and an interruptible condition occurs in the endpoint, the appropriate interrupt bit is set. Programming the mask bit to a 1 does not affect the current state of the interrupt bit. It only blocks future 0-to-1 transitions of the interrupt bit.

Register Name:	**UICR0**																													
Hex Offset Address:	0 x C800B050													**Reset Hex Value:**								0x000000FF								
Register Description:	Universal Serial Bus Device Controller Interrupt Control Register 0																													
Access: Read/Write and Read-Only																														
Bits																														
31																						**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																							IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
X																							1	1	1	1	1	1	1	1
Resets (Above)																														

Register		UICR0
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	IM7	Interrupt Mask for Endpoint 7. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
6	IM6	Interrupt Mask for Endpoint 6. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
5	IM5	Interrupt mask for Endpoint 5. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
4	IM4	Interrupt mask for Endpoint 4. 0 = Receive Interrupt enabled. 1 = Receive Interrupt disabled.
3	IM3	Interrupt mask for Endpoint 3. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
2	IM2	Interrupt Mask for Endpoint 2. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
1	IM1	Interrupt Mask for Endpoint 1. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
0	IM0	Interrupt mask for endpoint 0. 0 = Endpoint zero interrupt enabled. 1 = Endpoint zero interrupt disabled.

18.5.19 UDC Interrupt Control Register 1 (UICR1)

The UICR1 contains eight control bits to enable/disable interrupt service requests from endpoints 8 through 15. The UICR1 bits are reset to 1 so interrupts are not generated on initial system reset.

18.5.19.1 Interrupt Mask Endpoint x (IMx), where x is 8 through 15.

The UICR1[IMx] bit is used to mask or enable the corresponding endpoint interrupt request, USIR1[IRx]. When the mask bit is set, the interrupt is masked and the corresponding bit in the USIR1 register is not allowed to be set.

When the mask bit is cleared and an interruptible condition occurs in the endpoint, the appropriate interrupt bit is set. Programming the mask bit to a 1 does not affect the current state of the interrupt bit. It only blocks future 0-to-1 transitions of the interrupt bit.

Register Name:	UICR1				
Hex Offset Address:	0 x C800B054		Reset Hex Value:	0x000000FF	
Register Description:	Universal Serial Bus Device Controller Interrupt Control Register 1				
Access: Read/Write					

Bits																															
31																							8	7	6	5	4	3	2	1	0
(Reserved)																								IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8
																								1	1	1	1	1	1	1	1
Resets (Above)																															

Register		UICR1
Bits	Name	Description
31:8		Reserved for future use.
7	IM15	Interrupt mask for Endpoint 15. 0 = Transmit interrupt enabled. 1= Transmit interrupt disabled.
6	IM14	Interrupt mask for Endpoint 14. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
5	IM13	Interrupt mask for Endpoint 13. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
4	IM12	Interrupt mask for Endpoint 12. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
3	IM11	Interrupt mask for Endpoint 11. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
2	IM10	Interrupt mask for Endpoint 10. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
1	IM9	Interrupt mask for Endpoint 9. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
0	IM8	Interrupt Mask for Endpoint 8. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.

18.5.20 UDC Status/ Interrupt Register 0 (UISR0)

The UDC status/interrupt registers (USIR0 and USIR1) contain bits that are used to generate the UDC's interrupt request. Each bit in the UDC status/interrupt registers is logically ORed together to produce one interrupt request.

When the ISR for the UDC is executed, it must read the UDC status/interrupt register to determine why the interrupt occurred.

The bits in USIR0 and USIR1 are controlled by a mask bit in the UDC Interrupt Control Register (UICR0/1). The mask bits, when set, prevent a status bit in the USIRx from being set. If the mask bit for a particular status bit is cleared and an interruptible condition occurs, the status bit is set.

To clear status bits, the Intel XScale® processor must write a 1 to the position to be cleared. The interrupt request for the UDC remains active as long as the value of the USIRx is non-zero.

18.5.20.1 Endpoint 0 Interrupt Request (IR0)

The endpoint 0 interrupt request is set if the IM0 bit in the UDC control register is cleared and, in the UDC endpoint 0 control/status register, the OUT packet ready bit is set, the IN packet ready bit is cleared, or the sent STALL bit is set. The IR0 bit is cleared by writing a 1 to it.

18.5.20.2 Endpoint 1 Interrupt Request (IR1)

The interrupt request bit is set if the IM1 bit in the UDC interrupt control register is cleared and the IN packet complete (TPC) in UDC endpoint 1 control/status register is set.

The IR1 bit is cleared by writing a 1 to it.

18.5.20.3 Endpoint 2 Interrupt Request (IR2)

The interrupt request bit is set if the IM2 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready bit (RPC) in the UDC Endpoint 2 Control/Status Register is set.

The IR2 bit is cleared by writing a 1 to it.

18.5.20.4 Endpoint 3 Interrupt Request (IR3)

The interrupt request bit is set if the IM3 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or Transmit Underrun (TUR) in UDC Endpoint 3 Control/Status Register is set.

The IR3 bit is cleared by writing a 1 to it

18.5.20.5 Endpoint 4 Interrupt Request (IR4)

The interrupt request bit is set if the IM4 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready (RPC) or receiver overflow (ROF) in the UDC Endpoint 4 Control/Status Register or the Isochronous Error Endpoint 4 (IPE4) in the UFNHR are set.

The IR4 bit is cleared by writing a 1 to it.

18.5.20.6 Endpoint 5 Interrupt Request (IR5)

The interrupt request bit is set if the IM5 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) in UDC Endpoint 5 Control/Status Register is set.

The IR5 bit is cleared by writing a 1 to it.

18.5.20.7 Endpoint 6 Interrupt Request (IR6)

The interrupt request bit gets set if the IM6 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) in UDC endpoint 6 control/status register gets set.

The IR6 bit is cleared by writing a 1 to it.

18.5.20.8 Endpoint 7 Interrupt Request (IR7)

The interrupt request bit is set if the IM7 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready bit (RPC) in the UDC Endpoint 7 Control/Status Register is set.

The IR7 bit is cleared by writing a 1 to it.

Register Name:	USIR0				
Hex Offset Address:	0 x C800B058		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Controller Interrupt Status Register 0				
Access: Read/Write and Read-Only					

Bits																										
31																						8	7	6	5	4	3	2	1	0
(Reserved)																							IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
X																							0	0	0	0	0	0	0	0
Resets (Above)																														

Register		USIR0
Bits	Name	Description
31:8		Reserved for future use.
7	IR7	Interrupt Request Endpoint 7 (read/write 1 to clear). 1 = Endpoint 7 needs service.
6	IR6	Interrupt Request Endpoint 6 (read/write 1 to clear). 1 = Endpoint 6 needs service.
5	IR5	Interrupt Request Endpoint 5 (read/write 1 to clear). 1 = Endpoint 5 needs service.
4	IR4	Interrupt Request Endpoint 4 (read/write 1 to clear). 1 = Endpoint 4 needs service.
3	IR3	Interrupt Request Endpoint 3 (read/write 1 to clear). 1 = Endpoint 3 needs service.
2	IR2	Interrupt Request Endpoint 2 (read/write 1 to clear). 1 = Endpoint 2 needs service.
1	IR1	Interrupt Request Endpoint 1 (read/write 1 to clear). 1 = Endpoint 1needs service.
0	IR0	Interrupt Request Endpoint 0 (read/write 1 to clear). 1 = Endpoint 0 needs service.

18.5.21 UDC Status/Interrupt Register 1 (USIR1)

18.5.21.1 Endpoint 8 Interrupt Request (IR8)

The interrupt request bit is set if the IM8 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or Transmit Underrun (TUR) in UDC Endpoint 8 Control/Status Register is set.

The IR8 bit is cleared by writing a 1 to it.

18.5.21.2 Endpoint 9 Interrupt Request (IR9)

The interrupt request bit is set if the IM9 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready (RPC) or receiver overflow (ROF) in the UDC Endpoint 9 Control/Status Register or the Isochronous Error Endpoint 9 (IPE9) in the UFNHR are set.

The IR9 bit is cleared by writing a 1 to it.

18.5.21.3 Endpoint 10 Interrupt Request (IR10)

The interrupt request bit is set if the IM10 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or in UDC endpoint 10 control/status register is set.

The IR10 bit is cleared by writing a 1 to it.

18.5.21.4 Endpoint 11 Interrupt Request (IR11)

The interrupt request bit is set if the IM11 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) in UDC Endpoint 11 Control/Status Register is set.

The IR11 bit is cleared by writing a 1 to it.

18.5.21.5 Endpoint 12 Interrupt Request (IR12)

The interrupt request bit is set if the IM12 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready bit (RPC) in the UDC endpoint 12 control/status register is set.

The IR12 bit is cleared by writing a 1 to it.

18.5.21.6 Endpoint 13 Interrupt Request (IR13)

The interrupt request bit is set if the IM13 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or Transmit Underrun (TUR) in UDC Endpoint 13 Control/Status Register is set.

The IR13 bit is cleared by writing a 1 to it.

18.5.21.7 Endpoint 14 Interrupt Request (IR14)

The interrupt request bit is set if the IM14 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready (RPC) or receiver overflow (ROF) in the UDC Endpoint 14 Control/Status Register or the Isochronous Error Endpoint 14 (IPE14) in the UFNHR are set.

The IR14 bit is cleared by writing a 1 to it.

18.5.21.8 Endpoint 15 Interrupt Request (IR15)

The interrupt request bit is set if the IM15 bit in the UDC interrupt control is set.

The IR15 bit is cleared by writing a 1 to it.

Register Name:	USIR1																															
Hex Offset Address:	0 x C800B05C													Reset Hex Value:	0x00000000																	
Register Description:	Universal Serial Bus Device Controller Interrupt Status Register 1																															
Access: Read/Write and Read-Only																																
Bits																																
31																								8	7	6	5	4	3	2	1	0
(Reserved)																									IR15	IR14	IR13	IR12	IR11	IR10	IR9	IR8
X																									0	0	0	0	0	0	0	0
Resets (Above)																																

Register		USIR1
Bits	Name	Description
31:8		Reserved for future use.
7	IR15	Interrupt Request Endpoint 15 (read/write 1 to clear). 1 = Endpoint 15 needs service.
6	IR14	Interrupt Request Endpoint 14 (read/write 1 to clear). 1 = Endpoint 14 needs service.
5	IR13	Interrupt Request Endpoint 13 (read/write 1 to clear). 1= Endpoint 13 needs service.
4	IR12	Interrupt Request Endpoint 12 (read/write 1 to clear). 1 = Endpoint 12 needs service.
3	IR11	Interrupt Request Endpoint 11 (read/write 1 to clear). 1 = Endpoint 11 needs service.
2	IR10	Interrupt Request Endpoint 10 (read/write 1 to clear). 1 = Endpoint 10 needs service.
1	IR9	Interrupt Request Endpoint 9 (read/write 1 to clear). 1 = Endpoint 9 needs service.
0	IR8	Interrupt Request Endpoint 8 (read/write 1 to clear). 1 = Endpoint 8 needs service.

18.5.22 UDC Frame Number High Register (UFNHR)

The UDC Frame Number High Register holds the three most-significant bits of the frame number contained in the last received SOF packet, the isochronous OUT endpoint error status, and the SOF interrupt status/interrupt mask bit.

18.5.22.1 UDC Frame Number MSB (FNMSB)

The UFNHR[FNMSB] is the three most-significant bits of the 11-bit frame number contained in the last received SOF packet. The remaining bits are located in the UFNLR. This information is used for isochronous transfers.

These bits are updated every SOF.

18.5.22.2 Isochronous Packet Error Endpoint 4 (IPE4)

The isochronous packet error for Endpoint 4 is set if Endpoint 4 is loaded with a data packet that is corrupted. This status bit is used in the interrupt generation of endpoint 4.

To maintain synchronization, the software must monitor this bit when it services an SOF interrupt and reads the frame number. This bit is not set if the token packet is corrupted or if the sync or PID fields of the data packet are corrupted.

18.5.22.3 Isochronous Packet Error Endpoint 9 (IPE9)

The isochronous packet error for Endpoint 9 is set if Endpoint 9 is loaded with a data packet that is corrupted. This status bit is used in the interrupt generation of endpoint 9.

To maintain synchronization, software must monitor this bit when it services the SOF interrupt and reads the frame number. This bit is not set if the token packet is corrupted or if the sync or PID fields of the data packet are corrupted.

18.5.22.4 Isochronous Packet Error Endpoint 14 (IPE14)

The isochronous packet error for Endpoint 14 is set if Endpoint 14 is loaded with a data packet that is corrupted. This status bit is used in the interrupt generation of endpoint 14.

To maintain synchronization, software must monitor this bit when it services the SOF interrupt and reads the frame number. This bit is not set if the token packet is corrupted or if the sync or PID fields of the data packet are corrupted.

18.5.22.5 Start of Frame Interrupt Mask (SIM)

The UFNHR[SIM] bit is used to mask or enable the SOF interrupt request. When UFNHR[SIM]=1, the interrupt is masked and the SIR bit is not allowed to be set. When UFNHR[SIM]=0, the interrupt is enabled and when an interruptible condition occurs in the receiver, the UFNHR[SIR] bit is set.

Setting UFNHR[SIM] to a 1 does not affect the current state of UFNHR[SIR]. It only blocks future zero to one transitions of UFNHR[SIR].

18.5.22.6 Start of Frame Interrupt Request (SIR)

The interrupt request bit is set if the UFNHR[SIM] bit is cleared and an SOF packet is received.

The UFNHR[SIR] bit is cleared by writing a 1 to it.

Register Name:	**UFNHR**																															
Hex Offset Address:	0 x C800B060																					**Reset Hex Value:**	0x00000040								
Register Description:	Universal Serial Bus Device Frame Number High Register																															
Access: Read-Only																																
Bits																																
31																					**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**			
(Reserved)																						SIR	SIM	IPE14	IPE9	IPE4	3-Bit Frame Number MSB					
X																						0	1	0	0	0	0	0	0			
Resets (Above)																																

Register		UFNHR
Bits	**Name**	**Description**
31:8		Reserved for future use.
7	SIR	SOF Interrupt Request (read/write 1 to clear). 1 = SOF has been received.
6	SIM	SOF interrupt mask. 0 = SOF interrupt enabled. 1 = SOF interrupt disabled.
5	IPE14	Isochronous Packet Error Endpoint 14 (read/write 1 to clear). 1 = Status indicator that data in the endpoint fifo is corrupted.
4	IPE9	Isochronous Packet Error Endpoint 9 (read/write 1 to clear). 1 = Status indicator that data in the endpoint fifo is corrupted.
3	IPE4	Isochronous Packet Error Endpoint 4 (read/write 1 to clear). 1 = Status indicator that data in the endpoint fifo is corrupted.
2:0	FNMSB	Frame Number MSB. Most-significant three bits of 11-bit frame number associated with last receive SOF. Reset to all zeros.

18.5.23 UDC Frame Number Low Register (UFNLR)

The UDC frame number low register is the eight least-significant bits of the 11-bit frame number contained in the last received SOF packet. The three remaining bits are located in the UFNHR. This information is used for isochronous transfers.

These bits are updated every SOF.

Register Name:	UFNLR																															
Hex Offset Address:	0 x C800B064															**Reset Hex Value:**	0x00000000														
Register Description:	Universal Serial Bus Device Frame Number Low Register																															
Access: Read-Only																																
Bits																																
31																							**8**	**7**								**0**
(Reserved)																								8-Bit Frame Number LSB								
X																								0	0	0	0	0	0	0	0	
Resets (Above)																																

Register		UFNLR
Bits	**Name**	**Description**
31:8		Reserved for future use.
7.0	FNLSB	Frame number LSB. Least significant eight bits of frame number associated with last received SOF. Reset to all zeros.

18.5.24 UDC Byte Count Register 2 (UBCR2)

The Byte-Count Register maintains the remaining byte count in the active buffer of OUT endpoint2.

18.5.24.1 Endpoint 2 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read. The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:	UBCR2																													
Hex Offset Address:	0 x C800B068															Reset Hex Value:	0x00000000													
Register Description:	Universal Serial Bus Device Endpoint 2 Byte Count																													
Access: Read-Only																														
Bits																														
31																							8	7						0
(Reserved)																								BC[7:0]						
X																								0	0	0	0	0	0	0	0
Resets (Above)																														

Register		UBCR2
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1)

18.5.25 UDC Byte Count Register 4 (UBCR4)

The Byte-Count Register maintains the remaining byte count in the active buffer of out endpoint 4.

18.5.25.1 Endpoint 4 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:	UBCR4																													
Hex Offset Address:	0 x C800B06C															Reset Hex Value:	0x00000000													
Register Description:	Universal Serial Bus Device Endpoint 4 Byte Count																													
Access: Read-Only																														
Bits																														
31																							8	7						0
(Reserved)																								BC[7:0]						
X																								0	0	0	0	0	0	0	0
Resets (Above)																														

Register		UBCR4
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

18.5.26 UDC Byte Count Register 7 (UBCR7)

The Byte-Count Register maintains the remaining byte count in the active buffer of out endpoint 7.

18.5.26.1 Endpoint 7 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:	UBCR7				
Hex Offset Address:	0 x C800B070		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 7 Byte Count				
Access: Read-Only					

Bits																															
31																						8	7							0
(Reserved)																							BC[7:0]							
X																							0	0	0	0	0	0	0	0
Resets (Above)																															

Register		UBCR7
Bits	Name	Description
31:0		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

18.5.27 UDC Byte Count Register 9 (UBCR9)

The Byte-Count Register maintains the remaining byte count in the active buffer of out endpoint 9.

18.5.27.1 Endpoint 9 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:	UBCR9				
Hex Offset Address:	0 x C800B074		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 9 Byte Count				
Access: Read-Only					

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	(Reserved)																								BC[7:0]							
	X																								0	0	0	0	0	0	0	0
Resets (Above)																																

Register		UBCR9
Bits	Name	Description
31:0		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

18.5.28 UDC Byte Count Register 12 (UBCR12)

The byte count register maintains the remaining byte count in the active buffer of out endpoint 12.

18.5.28.1 Endpoint 12 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:	UBCR12				
Hex Offset Address:	0 x C800B078		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 12 Byte Count				
Access: Read-Only					

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	(Reserved)																								BC[7:0]							
	X																								0	0	0	0	0	0	0	0
Resets (Above)																																

Register		UBCR12
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

18.5.29 UDC Byte Count Register 14 (UBCR14)

The Byte-Count Register maintains the remaining byte count in the active buffer of out endpoint 14.

18.5.29.1 Endpoint 14 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:	UBCR14				
Hex Offset Address:	0 x C800B07C		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 14 Byte Count				
Access: Read-Only					

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	(Reserved)																								BC[7:0]							
Resets (Above)	X																								0	0	0	0	0	0	0	0

Register		UBCR14
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

18.5.30 UDC Endpoint 0 Data Register (UDDR0)

The UDC Endpoint 0 Data Register is an 16-entry by 8-bit bidirectional FIFO. When the host transmits data to the UDC Endpoint 0, the Intel XScale® processor reads the UDC Endpoint 0 Register to access the data.

When the UDC sends data to the host, the Intel XScale® processor writes the data to be sent in the UDC Endpoint 0 Register. The Intel XScale® processor can only read and write the FIFO at specific points in a control sequence.

The direction that the FIFO flows is controlled by the UDC. Normally, the UDC is in an idle state, waiting for the host to send commands. When the host sends a command, the UDC fills the FIFO with the command from the host and the Intel XScale® processor reads the command from the FIFO when it arrives. The only time the Intel XScale® processor may write the endpoint 0 FIFO is after a valid command from the host is received and it requires a transmission in response.

Register Name:	UDDR0				
Hex Offset Address:	0 x C800B080		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 0 Data Register				
Access: Read/Write					

Bits																															
31																					8								0		
(Reserved)																					(Data)										
X																															
Resets (Above)																															

Register		UDDR0		
Bits	Name	Description	Read Access	Write Access
31:8		Reserved for future use.		
7:0	DATA	Top/bottom of endpoint 0 FIFO data. Read Bottom of endpoint 0 FIFO data. Write Top of endpoint 0 FIFO data.	Bottom of Endpoint 0 FIFO	Top of Endpoint 0 FIFO

18.5.31 UDC Data Register 1 (UDDR1)

Endpoint 1 is a double-buffered bulk IN endpoint that is 64 bytes deep. Data can be loaded via direct Intel XScale® processor writes. Because it is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:	UDDR1				
Hex Offset Address:	0 x C800B100		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 1 Data Register				
Access: Write					

Bits																															
31																					8	7	6	5	4	3	2	1	0		
(Reserved)																					(8-Bit Data)										
X																					0	0	0	0	0	0	0	0			
Resets (Above)																															

Register		UDDR1
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.32 UDC Data Register 2 (UDDR2)

Endpoint 2 is a double-buffered bulk OUT endpoint that is 64 bytes deep. The UDC will generate an interrupt as soon as the EOP is received.

Since it is double-buffered, up to two packets of data may be ready. Via direct read from the Intel XScale® processor, the data can be removed from the UDC. If one packet is being removed and the packet behind it has already been received, the UDC will issue a NAK to the host the next time it sends an OUT packet to endpoint 2.

This NAK condition will remain in place until a full packet space is available in the UDC at Endpoint 2.

Register Name:	UDDR2				
Hex Offset Address:	0 x C800B180		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 2 Data Register				
Access: Read					

Bits	31																				8	7							0
	(Reserved)																					(8-Bit Data)							
	X																					0	0	0	0	0	0	0	0
Resets (Above)																													

Register	UDDR2	
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.

18.5.33 UDC Data Register 3 (UDDR3)

Endpoint 3 is a double-buffered isochronous IN endpoint that is 256 bytes deep. Data can be loaded via direct Intel XScale® processor writes.

Because it-is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:	UDDR3				
Hex Offset Address:	0 x C800B200		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 3 Data Register				
Access: Write					

Bits	31														16	15						8	7							0
	(Reserved)																						(8-Bit Data)							
	X																						0	0	0	0	0	0	0	0
Resets (Above)																														

Register	UDDR3	
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.34 UDC Data Register 4 (UDDR4)

Endpoint 4 is a double-buffered, isochronous OUT endpoint that is 256 bytes deep. The UDC generates an interrupt when the EOP is received.

Because it is double-buffered, up to two packets of data may be ready. The data can be removed from the UDC via a direct read from the Intel XScale® processor. If one packet is being removed and the packet behind it has already been received, the UDC issues a NAK to the host the next time it sends an OUT packet to Endpoint 4.

This NAK condition remains in place until a full packet space is available in the UDC at Endpoint 4.

Register Name:	**UDDR4**																															
Hex Offset Address:	0 x C800B400																**Reset Hex Value:**				0x00000000										
Register Description:	Universal Serial Bus Device Endpoint 4 Data Register																															
Access: Read																																
Bits																																
31																						**8**	**7**								**0**	
(Reserved)																							(8-Bit Data)									
X																							0	0	0	0	0	0	0	0		
Resets (Above)																																

Register		**UDDR4**
Bits	**Name**	**Description**
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.

18.5.35 UDC Data Register 5 (UDDR5)

Endpoint 5 is an interrupt IN endpoint that is 8 bytes deep. Data must be loaded via direct Intel XScale® processor writes.

Because the USB system is a host-initiator model, the host must poll Endpoint 5 to determine interrupt conditions. The UDC can not initiate the transaction.

Register Name:	**UDDR5**																															
Hex Offset Address:	0 x C800B008																**Reset Hex Value:**				0x00000000										
Register Description:	Universal Serial Bus Device Endpoint 5 Data Register																															
Access: Write.																																
Bits																																
31																						**8**	**7**								**0**	
(Reserved)																							(8-Bit Data)									
X																							0	0	0	0	0	0	0	0		
Resets (Above)																																

Register		UDDR5
Bits	**Name**	**Description**
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.36 UDC Data Register 6 (UDDR6)

Endpoint 6 is a double-buffered, bulk IN endpoint that is 64 bytes deep. Data can be loaded via direct Intel XScale® processor writes.

Because it is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:	UDDR6				
Hex Offset Address:	0 x C800B600		**Reset Hex Value:**	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 6 Data Register				
Access: Write					

Bits																															
31																						**8**	**7**								**0**
(Reserved)																							(8-Bit Data)								
X																							0	0	0	0	0	0	0	0	
Resets (Above)																															

Register		UDDR6
Bits	**Name**	**Description**
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.37 UDC Data Register 7 (UDDR7)

Endpoint 7 is a double-buffered, bulk OUT endpoint that is 64 bytes deep. The UDC will generate an interrupt request as soon as the EOP is received.

Since it is double-buffered, up to two packets of data may be ready. Via direct read from the Intel XScale® processor, the data can be removed from the UDC. If one packet is being removed and the packet behind it has already been received, the UDC will issue a NAK to the host the next time it sends an OUT packet to endpoint 7.

This NAK condition will remain in place until a full packet space is available in the UDC at Endpoint 7.

Register Name:	UDDR7				
Hex Offset Address:	0 x C800B680		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 7 Data Register				
Access: Read					

Bits	31																				8	7							0
	(Reserved)																					(8-Bit Data)							
Resets (Above)	X																					0	0	0	0	0	0	0	0

Register		UDDR7
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.

18.5.38 UDC Data Register 8 (UDDR8)

Endpoint 8 is a double-buffered, isochronous IN endpoint that is 256 bytes deep. Data can be loaded via direct Intel XScale® processor writes.

Because it-is double buffered, up to two packets of data may be loaded for transmission.

Register Name:	UDDR8				
Hex Offset Address:	0 x C800B700		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 8 Data Register				
Access: Write					

Bits	31																				8	7							0
	(Reserved)																					(8-Bit Data)							
Resets (Above)	X																					0	0	0	0	0	0	0	0

Register		UDDR8
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.39 UDC Data Register 9 (UDDR9)

Endpoint 9 is a double-buffered, isochronous OUT endpoint that is 256 bytes deep. The UDC generates an interrupt request when the EOP is received.

Because it is double-buffered, up to two packets of data may be ready. The data can be removed from the UDC via a direct read from the Intel XScale® processor. If one packet is being removed and the packet behind it has already been received, the UDC issues a NAK to the host the next time it sends an OUT packet to Endpoint 9.

This NAK condition remains in place until a full packet space is available in the UDC at Endpoint 9.

Register Name:	UDDR9				
Hex Offset Address:	0 x C800B900		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 9 Data Register				
Access: Read					

Bits	31																				8	7							0
	(Reserved)																					(8-Bit Data)							
	X																					0	0	0	0	0	0	0	0
Resets (Above)																													

Register	UDDR9	
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.

18.5.40 UDC Data Register 10 (UDDR10)

Endpoint 10 is an interrupt IN endpoint that is 8 bytes deep. Data must be loaded via direct Intel XScale® processor writes.

Because the USB system is a host-initiator model, the host must poll Endpoint 10 to determine interrupt conditions. The UDC can not initiate the transaction.

Register Name:	UDDR10				
Hex Offset Address:	0 x C800B0C0		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 10 Data Register				
Access: Write					

Bits	31																				8	7							0
	(Reserved)																					(8-Bit Data)							
	X																					0	0	0	0	0	0	0	0
Resets (Above)																													

Register	UDDR10	
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.41 UDC Data Register 11 (UDDR11)

Endpoint 11 is a double-buffered, bulk IN endpoint that is 64 bytes deep. Data can be loaded via direct Intel XScale® processor writes.

Because it is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:	UDDR11				
Hex Offset Address:	0 x C800BB00		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 11 Data Register				
Access: Write					

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	(Reserved)																								(8-Bit Data)							
Resets (Above)	X																								0	0	0	0	0	0	0	0

Register	UDDR11	
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.42 UDC Data Register 12 (UDDR12)

Endpoint 12 is a double-buffered, bulk OUT endpoint that is 64 bytes deep. The UDC will generate an interrupt request as soon as the EOP is received.

Since it is double-buffered, up to two packets of data may be ready. Via direct read from the Intel XScale® processor, the data can be removed from the UDC. If one packet is being removed and the packet behind it has already been received, the UDC will issue a NAK to the host the next time it sends an OUT packet to endpoint 12.

This NAK condition will remain in place until a full packet space is available in the UDC at Endpoint 12.

Register Name:	UDDR12				
Hex Offset Address:	0 x C800BB80		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 12 Data Register				
Access: Read					

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	(Reserved)																								(8-Bit Data)							
Resets (Above)	X																								0	0	0	0	0	0	0	0

Register		UDDR11
Bits	**Name**	**Description**
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.43 UDC Data Register 13 (UDDR13)

Endpoint 13 is a double-buffered, isochronous IN endpoint that is 256 bytes deep. Data can be loaded via direct Intel XScale® processor writes.

Because it-is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:	UDDR13				
Hex Offset Address:	0 x C800BC00		**Reset Hex Value:**	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 13 Data Register				
Access: Write					

Bits																															
31																						**8**	**7**								**0**
(Reserved)																							(8-Bit Data)								
X																							0	0	0	0	0	0	0	0	
Resets (Above)																															

Register		UDDR13
Bits	**Name**	**Description**
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.44 UDC Data Register 14 (UDDR14)

Endpoint 14 is a double-buffered, isochronous OUT endpoint that is 256 bytes deep. The UDC generates an interrupt request when the EOP is received.

Because it is double-buffered, up to two packets of data may be ready. The data can be removed from the UDC via a direct read from the Intel XScale® processor. If one packet is being removed and the packet behind it has already been received, the UDC issues a NAK to the host the next time it sends an OUT packet to Endpoint 14.

This NAK condition remains in place until a full packet space is available in the UDC at Endpoint 14.

Register Name:	UDDR14				
Hex Offset Address:	0 x C800BE00		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 14 Data Register				
Access: Read					

Bits																										
31																					8	7							0
(Reserved)																						(8-Bit Data)							
X																						0	0	0	0	0	0	0	0
Resets (Above)																													

Register		UDDR14
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

18.5.45 UDC Data Register 15 (UDDR15)

Endpoint 15 is an interrupt IN endpoint that is 8 bytes deep. Data must be loaded via direct Intel XScale® processor writes.

Because the USB system is a host-initiator model, the host must poll Endpoint 15 to determine interrupt conditions. The UDC can not initiate the transaction.

Register Name:	UDDR15				
Hex Offset Address:	0 x C800B0E0		Reset Hex Value:	0x00000000	
Register Description:	Universal Serial Bus Device Endpoint 15 Data Register				
Access: Write					

Bits																										
31																					8	7							0
(Reserved)																						(8-Bit Data)							
X																						0	0	0	0	0	0	0	0
Resets (Above)																													

Register		UDDR15
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

§ §

19.0 UTOPIA Level-2

The functionality supported by the UTOPIA Level-2 interface is tightly coupled with the code written on the Network Processor Engine (NPE). This chapter details the full hardware capabilities of the UTOPIA-2 interface contained within the UTOPIA Level-2 coprocessor of the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described in this chapter.

Not all of the Intel® IXP42X product line and IXC1100 control plane processors have this functionality. See Table 173.

Table 173. Processors' Devices with UTOPIA

	Intel® IXP425 Network Processor B Step	Intel® IXP423 Network Processor	Intel® IXP422 Network Processor	Intel® IXP421 Network Processor	Intel® IXP420 Network Processor	Intel® IXC1100 Control Plane Processor
UTOPIA Level-2	X	X		X		

The UTOPIA Level-2 is an industry-standard interface that is used to provide connection between Asynchronous Transmission Mode (ATM) and physical layer (PHY) of an ATM network. The UTOPIA-2 coprocessor is the entity, within the IXP42X product line and IXC1100 control plane processors, that provides the UTOPIA Level-2 interface.

The UTOPIA Level-2 coprocessor, implemented on the IXP42X product line and IXC1100 control plane processors, provides an 8-bit, UTOPIA Level-2 interface operating at speeds of up to 33 MHz. The UTOPIA Level-2 interface can be configured to operate in a single-PHY (SPHY) or a multiple-PHY (MPHY) environment.

The interface contains five transmit and five receive address lines for multi-PHY selection. The UTOPIA Level-2 coprocessor is comprised of three functional modules:

- UTOPIA Transmit Module
- UTOPIA Receive Module
- Network Processor Engine (NPE) Core Interface Module

Two 128-byte-deep FIFOs are contained in each direction of data flow — one FIFO for transmit and one FIFO for receive. Each FIFO is organized into two cell buffers, each being 64 bytes deep. This FIFO arrangement allows the receive module to be processing a cell and storing it away at the same time the Network Processor Engine core is processing a previously received cell.

In the transmit direction, the Network Processor Engine core can be placing a cell into one transmit buffer while the Transmit Module is removing the cell from the other transmit buffer.

When operating in single-PHY (SPHY) mode, the UTOPIA Level-2 interface will support octet- or cell-level handshaking as defined by the UTOPIA Level-2 specification. When configured in multiple-PHY (MPHY) mode, only cell-level handshaking is supported.

The hardware interface allows connection of up to 31 physical interface devices, as defined in the UTOPIA Level-2 specification. However, the ATM Adaptation Layers (AAL) supports only the following number of physical devices:

- Intel® IXP425 Network Processor — 24 physical devices
- Intel® IXP423 Network Processor — 4 physical devices
- Intel® IXP421 Network Processor — 4 physical devices

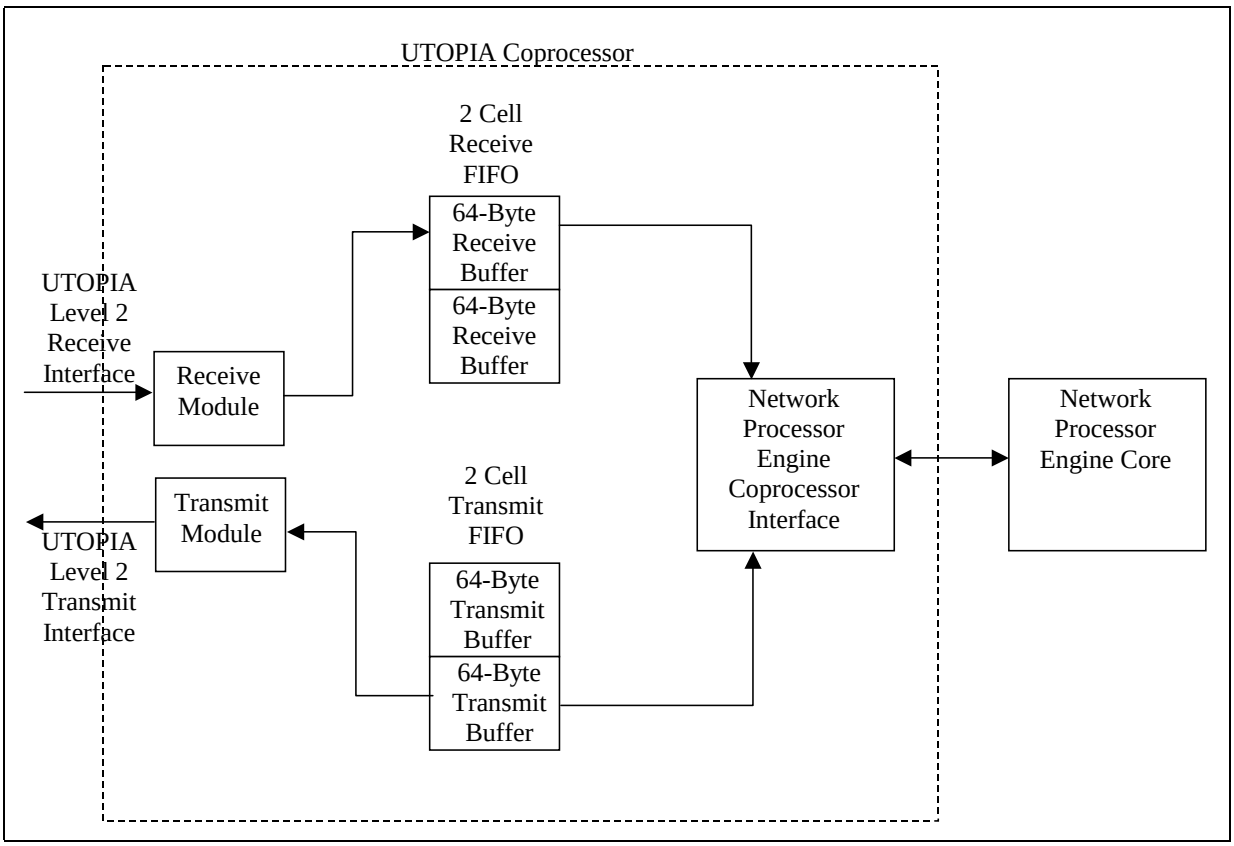
Additionally, the two-cycle polling routine defined by the UTOPIA Level-2 specification limits the number of physical devices to 26.

The ATM Adaptation Layers (AAL) implemented by the Network Processor Engine may have some further limitations on the number of physical interfaces supported. For more details on the number of physical interfaces supported, see the *Intel® IXP400 Software Programmer's Guide*.

On the Network Processor Engine side of the UTOPIA-2 coprocessor, the UTOPIA-2 coprocessor interfaces to the Network Processor Engine Core via the Network Processor Engine Coprocessor Bus Interface. The Network Processor Engine Coprocessor Bus Interface is used to transfer data to and from the Network Processor Engine core. The Network Processor Engine Coprocessor Bus Interface also is used to access status and configuration information for the UTOPIA-2 coprocessor.

Figure 97 shows the various modules within the UTOPIA-2 coprocessor.

Figure 97. UTOPIA Level-2 Coprocessor



19.1 UTOPIA Transmit Module

The functionality supported by the Transmit Module is tightly coupled with the code written on the Network Processor Engine. This section details the full hardware capabilities of the Transmit Module contained within the UTOPIA Level-2 coprocessor of the IXP42X product line and IXC1100 control plane processors. The module's user-accessible features are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described in this section.

The UTOPIA Level-2 Transmit interface transfers ATM cells to one or more UTOPIA-compliant physical devices. In multiple-PHY (MPHY) mode, the UTOPIA Level-2 transmit interface uses a round-robin polling routine to poll the various physical interfaces using the five transmit address lines (UTP_TX_ADDR) to determine which physical interfaces are ready to accept data transfers. The result of the polling is provided as status to the Network Processor Engine.

The Transmit Module is the entity, within the UTOPIA coprocessor, that implements this functionality. The Transmit Module will poll a programmable number of physical interfaces as defined by the Transmit Address Range (TXADDRRANGE) register.

If five physical interfaces are connected to the UTOPIA Level-2 interface, a value of four can be programmed into the Transmit Address Range (TXADDRRANGE) register by the Network Processor Engine Core. The polling will always begin at logic address 0 and poll sequentially to the value contained in the Transmit Address Range (TXADDRRANGE) register.

To allow the most flexibility, a logical address to physical address table is provided. The look-up table makes it possible for the five addresses — that were called out, above — not to be in sequential order. For example, the following logical to physical address map could be used for the above example of five physical interfaces.

- Logical Address 0 => Physical Address 3 => UTP_TX_ADDR lines = 00011
- Logical Address 1 => Physical Address 5 => UTP_TX_ADDR lines = 00101
- Logical Address 2 => Physical Address 7 => UTP_TX_ADDR lines = 00111
- Logical Address 3 => Physical Address 9 => UTP_TX_ADDR lines = 01001
- Logical Address 4 => Physical Address 22 => UTP_TX_ADDR lines = 10110

Once the physical address is driven to all physical interfaces, using the UTP_TX_ADDR signals, the physical interface is ready to accept a cell. The physical interface is configured to the address signals that match the values contained on the UTP_TX_ADDR signals and responds to the UTOPIA Level-2 interface on the IXP42X product line and IXC1100 control plane processors by driving their UTP_OP_FCI (a.k.a TX_FULL_N/TX_CLAV) signal to inform the UTOPIA Level-2 Interface that the physical interface is ready to receive a cell.

The Transmit Port Status (TXPORTSTAT**)** register — contained within the Transmit Module — stores the polling result for each of the physical interfaces. The Network Processor Engine core uses the values stored in the Transmit Port Status (TXPORTSTAT**)** Register to select a physical interface that is ready to complete a transfer and loads the Transmit FIFO.

The Transmit FIFO informs the Transmit Module that a cell is ready to be transmitted to a specific physical interface. The Transmit Module will then remove the cell information from the Transmit FIFO and begin transmitting the data to the specified physical interface. It is important to note that the NPE code will send to the Transmit Module the logical port address of the physical interface to be selected along with the cell data. This feature allows the NPE to have full control over transmitted data based upon the polling status returned from the hardware.

While transmitting the data, an optional head-error correction (HEC) value can be calculated from the header and inserted into the data stream. The HEC generation unit takes the header data and uses the data with an internal, 8-bit HEC cyclical redundancy check (CRC) residue register to produce the value for the next HEC CRC residue. The HEC is generated new for every cell transmitted and has no dependencies on previous cells transmitted.

The HEC residue may be inserted directly into the data stream being transmitted over the UTOPIA Level-2 interface or, optionally, the HEC residue may be exclusive-ORed with hexadecimal 0x55 — to generate a COSET value, before being inserted into the data stream.

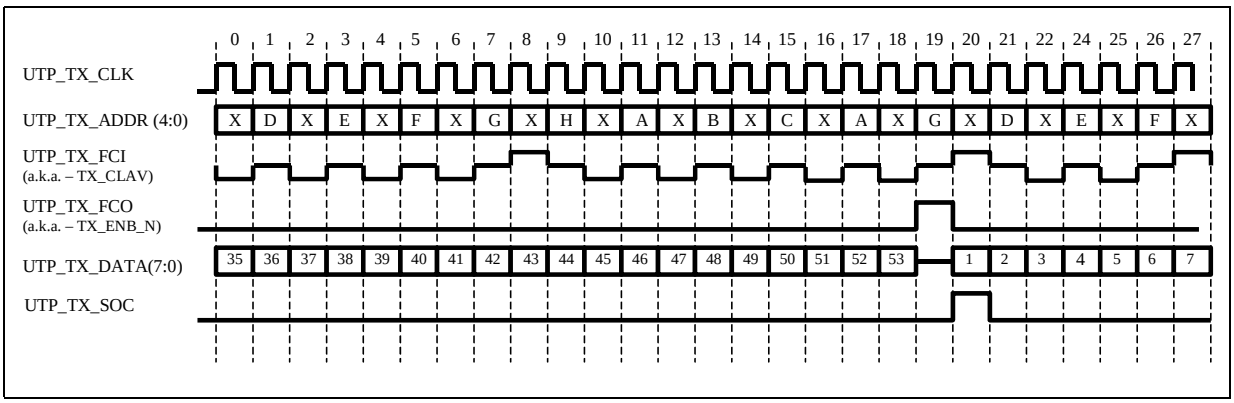
The HEC value is available one clock period after the last byte of the header information is transmitted. Therefore, a complete stream of cell data (H0, H1, H2, H3, HEC, D0, D1, ...) can be transmitted in successive clock cycles without interruption to the data stream.

When the Transmit HEC (TxHEC**)** configuration bit is enabled, the UTOPIA transmit interface will always insert an extra byte (valid HEC) into the cell being transmitted. In normal operation (TxHEC is enabled), the UTOPIA Level-2 Coprocessor Transmit Module will expect 52 bytes from the Transmit FIFO and the Transmit Module will insert a valid HEC field into the data stream.

Figure 98 shows the transmission of a cell in multiple-PHY (MPHY) mode. The following assumptions are made for the figure:

- There are eight active physical interfaces connected, named A through H, that map to logical address 0 through 7.
- Physical Interface A is the currently selected physical interface for clock cycles 0 through18.
- Notice on clock 8 that the result from Physical Interface G is that Physical Interface G is ready to receive a cell. The UTP_TX_FCI signal flags that a full cell can be sent to Physical Interface G by the Physical Interface asserting the UTP_TX_FCI to logic 1 one clock after Physical Interface G has been polled.
- On clock 17, the final Physical Interface polled is the Physical Interface that is currently selected. This polling is irrelevant to the Physical Interface that was polled previously prior to this location.
- Notice on clock cycles 19 and 20 that Physical Interface G is selected as the next Physical Interface that the IXP42X product line and IXC1100 control plane processors will transmit data to.

Figure 98. UTOPIA Level-2 MPHY Transmit Polling



In cell-level single-PHY (SPHY) mode the physical interface indicates that it can accept a cell by asserting the UTP_OP_FCI (also known as TX_FULL_N/TX_CLAV) signal. The UTOPIA Level-2 Interface subsequently transmits a cell to the PHY at the same time asserting UTP_OP_FCO (a.k.a.TX_ENB_N).

For more timing diagrams and more details on operation in single-PHY (SPHY) mode of operation, see the *UTOPIA Level-2, Revision 1.0 Specification*.

When using the UTOPIA Level-2 Interface in single-PHY (SPHY) mode the UTP_TX_ADDR is driven with an address of all logic 1s and the UTP_RX_ADDR must be driven with all logic 1s.

In octet-level single-PHY (SPHY) mode, the physical interface indicates to the UTOPIA Level-2 Interface on the IXP42X product line and IXC1100 control plane processors that the physical interface can accept data by de-asserting UTP_OP_FCI (also known as TX_FULL_N/TX_CLAV) signal. The UTOPIA Level-2 Interface subsequently transmits data to the PHY at the same time, asserting UTP_OP_FCO (also known as TX_ENB_N).

When the physical interface de-asserts UTP_OP_FCI (also known as TX_FULL_N/ TX_CLAV), it indicates to the UTOPIA Level-2 Interface that the physical interface will only accept four more bytes of data.

For more timing diagrams and details on the single-PHY (SPHY) mode of operation, see the *UTOPIA Level-2, Revision 1.0 Specification*.

In addition to supporting data transmission and HEC generation, the Transmit Module maintains some statistical values. The statistics that can be maintained are on a single physical port address on a specified VPI/VCI address value. The 32-bit counters will maintain the following counts:

- The number of cells transmitted
- The number of idle cells transmitted

The counters are not cleared when read by the Network Processor Engine core. The Network Processor Engine core must perform an explicit write to the specified register to clear the counter values. There is an overflow bit for each counter to indicate that the count has “rolled-over.” A mask-able interrupt mechanism is used to enable the UTOPIA Level-2 coprocessor to flag to the Network Processor Engine core that the “roll over” has occurred.

19.2 UTOPIA Receive Module

The functionality supported by the Receive Module is tightly coupled with the code written on the Network Processor Engine core. This section details the full hardware capabilities of the Receive Module contained within the UTOPIA Level-2 Coprocessor of the IXP42X product line and IXC1100 control plane processors. The module’s user-accessible features are described in the *Intel® IXP400 Software Programmer’s Guide* and may be a subset of the features described in this section.

The UTOPIA Level-2 Receive interface receives ATM cells from one or more UTOPIA-compliant physical devices.

In multiple-PHY (MPHY) mode, the UTOPIA Level-2 receive interface uses a round-robin polling routine to poll the various physical interfaces using the five receive address lines (UTP_RX_ADDR) to determine which physical interfaces are ready to send data. The result of the polling is provided as status to the Network Processor Engine core. The Receive Module is the entity within the UTOPIA coprocessor that implements this functionality.

The Receive Module will poll a programmable number of physical interfaces, as defined by the Receive Address Range (RXADDRRANGE) register. If three physical interfaces are connected to the UTOPIA Level-2 interface, a value of two can be programmed into the Receive Address Range (RXADDRRANGE) register by the Network Processor Engine core. The polling will always begin at address 0 and poll sequentially to the value contained in the Receive Address Range (RXADDRRANGE) register. If, for example, a two was programmed into the Receive Address Range (RXADDRRANGE) register, the external physical interfaces would have to be configured to respond to the first three physical addresses produced by the UTOPIA Level-2 UTP_RX_ADDR signals.

To allow the most flexibility a logical address to physical address table is provided. The look-up table makes it possible for the three addresses that were called out above not to be in sequential order.

For example, the following logical to physical address map could be used for the above example of three physical interfaces.

- Logical Address 0 => Physical Address 3 => UTP_RX_ADDR lines = “00011”
- Logical Address 1 => Physical Address 5 => UTP_RX_ADDR lines = “00101”
- Logical Address 2 => Physical Address 7 => UTP_RX_ADDR lines = “00111”

Once the physical address is driven to all physical interfaces using the UTP_RX_ADDR signals. The physical interface that is prepared to send a cell — and configured to the address signals that match the values contained on the UTP_RX_ADDR signals — responds to the UTOPIA Level-2 Interface on the IXP42X product line and IXC1100

control plane processors by driving their UTP_IP_FCI (also known as RX_EMPTY_N/ RX_CLAV) signal, to inform the UTOPIA Level-2 Interface that the physical interface is ready to send a cell.

The Receive Port Status (RXPORTSTAT**)** register, contained within the Receive Module, stores the polling result for each of the physical interfaces. The UTOPIA Level-2 hardware uses the values — stored in the Receive Port Status (RXPORTSTAT**)** Register — to determine the physical interface the received cell originated from. The Receive Module will store the received cell along with the physical interface address that the cell was received from into the Receive FIFO with some basic filtering capability if desired. The Network Processor Engine will then read the address from which the cell originated and then the cell.

The Receive Module performs an optional, cell-level filtering that may cause a cell to be discarded prior to being placed into the Receive FIFO. Some of these features can be enabled or disabled include:

- Received cells that are too short are discarded
- Excess bytes of received cells that are too long are discard
- Detection of HEC errors in the cell header causes the cell to be discarded (Can be enabled/disabled)
- Detection of idle cells will be discarded (Can be enabled/disabled. The definition of an idle cell is programmable by setting the appropriate values in the Receive Define Idle [RxDefineIdle] registers)

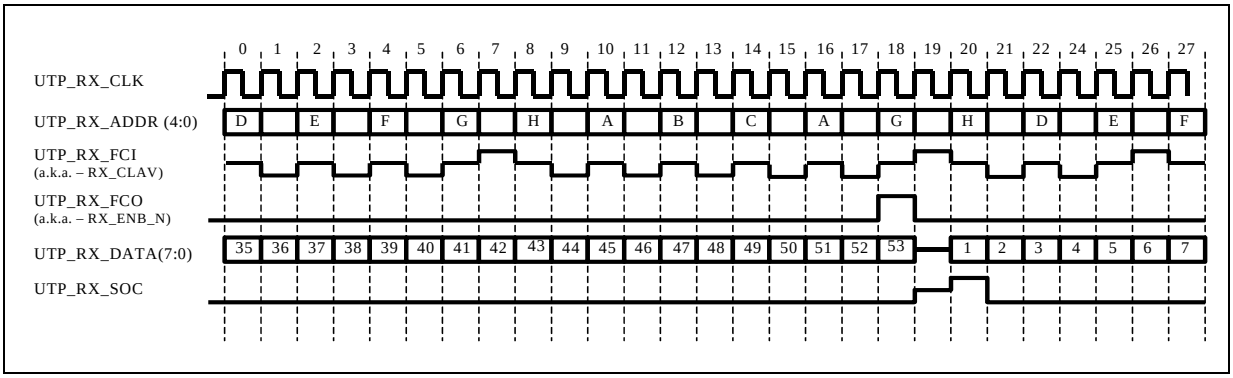
As the Receive Module is placing data into the Receive FIFO, the header information is being passed to the Receive Pre-Hash Unit.

The UTOPIA Coprocessor Receive Pre-Hash Module provides a mechanism for allowing incoming UTOPIA cells to have the ATM header looked up in a hash table to achieve faster address recognition. The hash unit takes the incoming ATM header, combines it with the arriving port information, and produces a header that can be read by the Network Processor Engine core. The Receive Pre-Hash Function can be enabled or disabled and can be used in single-PHY or multiple-PHY modes of operation. When used in single-PHY mode of operation, the port address will always be zero.

Figure 99 shows the reception of a cell in multiple-PHY (MPHY) mode. The following assumptions are made for the figure:

- There are eight active physical interfaces connected named A through H which map to logical address 0 through 7.
- Physical Interface A is the currently selected physical interface for clock cycles 0 through18.
- Notice on clock 7 that the result from Physical Interface G is that Physical Interface G has a full cell ready for the IXP42X product line and IXC1100 control plane processors. The UTP_RX_FCI signal flags that a full cell is ready to be sent by Physical Interface G to the IXP42X product line and IXC1100 control plane processors, asserting the UTP_RX_FCI to logic 1 one clock after Physical Interface G has been polled.
- On clock 16, the final Physical Interface polled is the Physical Interface that is currently selected. This polling is irrelevant to the Physical Interface that was polled previously prior to this location.
- Notice on clock cycles 18 and 19 that Physical Interface G is selected as the next Physical Interface that the IXP42X product line and IXC1100 control plane processors will receive data from that Physical Interface.

Figure 99. UTOPIA Level-2 MPHY Receive Polling



In cell-level single-PHY (SPHY) mode, the physical interface indicates that a cell is ready to be sent by asserting the UTP_IP_FCI (a.k.a. RX_EMPTY_N/RX_CLAV) signal. The UTOPIA Level-2 Interface on the IXP42X product line and IXC1100 control plane processors subsequently initiates the transfer of a cell from the physical interface by asserting UTP_IP_FCO (a.k.a. RX_ENB_N).

In octet-level single-PHY (SPHY) mode, the UTOPIA Level-2 Interface on the IXP42X product line and IXC1100 control plane processors indicate to the physical interface that the UTOPIA Receive interface is ready to receive bytes by asserting UTP_IP_FCO (a.k.a. RX_ENB_N) signal. The physical interface indicates a valid byte is on the UTOPIA data bus by de-asserting UTP_IP_FCI (also known as RX_EMPTY_N/RX_CLAV) signal.

The Receive Module maintains various statistical counters. The statistics that can be maintained are on a single physical port address on a specified VPI/VCI address value. The 32-bit counters will maintain the following counts:

- The number of cells received
- The number of cells with an incorrect cell size
- The number of cells containing HEC errors.
- The number of idle cells received

The counters are not cleared when read by the Network Processor Engine core. The Network Processor Engine core must perform an explicit write to the specified register to clear the counter values.

There is an overflow bit per counter to indicate that the count has “rolled over.” A mask-able interrupt mechanism is used to allow the UTOPIA Level-2 Coprocessor to flag to the Network Processor Engine Core that a “roll over” has occurred.

19.3 UTOPIA-2 Coprocessor / NPE Coprocessor: Bus Interface

The Network Processor Engine Coprocessor Interface Module provides the necessary interface logic required for configuration, monitoring, control, and test of the UTOPIA Level-2 coprocessor. All of the UTOPIA Level-2 coprocessor’s internal configuration and control registers, instruction registers, and FIFOs are directly accessible by the Network Processor Engine core.

19.4 MPHY Polling Routines

The UTOPIA Level-2 coprocessor implements a round-robin polling algorithm. The Receive and Transmit modules use a logical-to-physical address-translation table to determine the actual physical interface that is to be polled. This feature allows the designer complete control over the physical address polling sequence.

The multiple-PHY (MPHY) address translation is used by the UTOPIA Level-2 Interface, on the IXP42X product line and IXC1100 control plane processors, to poll physical addresses that are not contiguous or do not start at 0.

There are two translation tables implemented. One translation table is used for receive interface polling and the other translation table is used for transmit interface polling. Each translation table is implemented as 31 5-bit registers. Each register is addressed from 0 to 30, corresponding to one of 31 logical addresses.

The five bits of each register are used to designate a physical interface number. Therefore if a binary value of 00101 is written to address location 0 (logical port 0) of the transmit translation table, the polling sequence would actually assert a five on the transmit (UTP_TX_ADDR) address lines of the UTOPIA Level-2 interface during logical port 0's turn in the polling algorithm.

For example, make the following assumptions:

- A design requires eight physical interfaces to be connected, which are configured to respond to addresses 0 through 7.
- The polling order of the physical interfaces is required to be 1, 3, 5, 7, 0, 2, 4, 6 for both transmit and receive.

To accomplish this polling sequence:

1. The Network Processor Engine core will set the TXADDRRANGE and the RXADDRRANGE to a hexadecimal value of 0x7.

   This will identify that there are eight physical interfaces attached and involved in the polling sequence.
2. Define the values in both the transmit and receive translation tables as follows:
   - Address 0 (logical address 0) = A binary value of 00001
   - Address 1 (logical address 1) = A binary value of 00011
   - Address 2 (logical address 2) = Aa binary value of 00101
   - Address 3 (logical address 3) = Aa binary value of 00111
   - Address 4 (logical address 4) = A binary value of 00000
   - Address 5 (logical address 5) = A binary value of 00010
   - Address 6 (logical address 6) = A binary value of 00100
   - Address 7 (logical address 7) = A binary value of 00110

The polling sequence at the physical interface will rotate from logical address 0 through 7. This event will cause the physical address polling values on the UTOPIA Level-2 physical interface to be 1, 3, 5, 7, 0, 2, 4, 6.

19.5 UTOPIA Level-2 Clocks

The UTOPIA Level-2 interface on the IXP42X product line and IXC1100 control plane processors characterizes the interface for clock speeds of 25 MHz and 33 MHz.

The UTOPIA Level-2 interface requires both transmit and receive clock inputs to be supplied from an external source. The transmit module and receive module of the UTOPIA Level-2 interface can have independent clocks running at separate clock speeds.

§ §

20.0 JTAG Interface

The JTAG signals JTG_TCK, JTG_TRST_N, and JTAG_TDI will be routed to both the Test Logic Unit (TLU) and the Intel XScale® Processor. The Test Logic Unit is a unit that is provided to implement JTAG functions that are specific the Intel® IXP42X Product Line of Network Processors and IXC1100 Control Plane Processor. The Intel XScale processor JTAG functionality is consistent among the Intel XScale processor family of processors.

For additional details on the JTAG functionality supported by the Intel XScale processor, Section 3.6, “Software Debug” on page 88.

The Test Logic Unit receives the current state of the TAP controller located in the Intel XScale processor and the current instruction loaded in Intel XScale processor JTAG Instruction Register. The Test Logic Unit determines if the instruction is for the Intel XScale processor or the Intel® IXP42X product line and IXC1100 control plane processors Test Logic Unit by decoding the JTAG instruction located in the Intel XScale processor.

All of the IXP42X product line and IXC1100 control plane processors-specific instructions — except for HIGH_Z — are intended for one of three data registers in the TLU (Test Register, Key/Fuse Register, and the Boundary Scan Register) and, consequently, connect the data register between TDI and TDO.

The Boundary Scan Register is the only register that is accessible to users.

20.1 TAP Controller

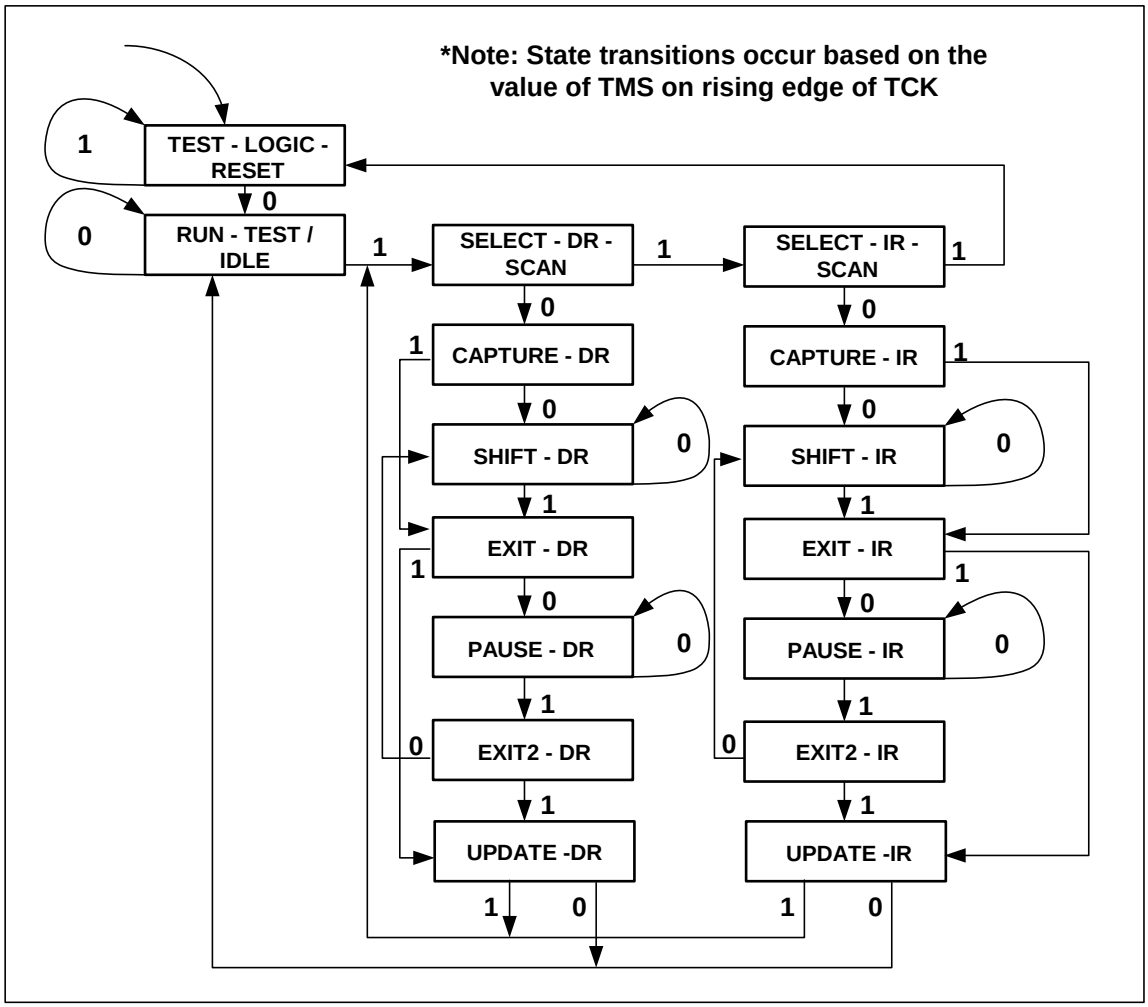
A JTAG TAP controller is implemented in the Intel XScale processor. The current state of the TAP controller is directed to the Test Logic Unit through direct signalling.

The TAP controller is a 16-state, synchronous, finite state machine that changes state on the rising edge of JTG_TCK. The controller’s next state is controlled by the signal present on the JTG_TMS input (which only goes to Intel XScale processor, since the IXP42X product line and IXC1100 control plane processors don’t implement their own TAP controller).

The TAP controller generates control signals, that — together with JTG_TCK and control signals decoded from the instruction active in the Intel XScale processor instruction register — determine the operation of the test circuitry.

For greater detail on the state machine, see *IEEE 1149.1a Standard Test Access Port and Boundary-Scan Architecture*.

Figure 100. TAP Controller State Diagram



Note: Numbers associated with arrows indicate the value on the JTG_TMS signal.

20.1.1 Test-Logic-Reset State

In Test-Logic-Reset State, test logic is disabled to allow normal operation of the chip. Loading the IDCODE register disables test logic.

No matter what the state of the controller, it enters Test-Logic-Reset state when JTG_TMS input is held high for at for five rising edges of JTG_TCK. The controller remains in Test-Logic-Reset state while JTG_TMS is high. The TAP controller is also forced to enter this state by enabling JTG_TRST_N.

If the controller exits the Test-Logic-Reset controller state as a result of an erroneous low signal on the JTG_TMS line — at the time of a rising edge on JTG_TCK (for example, a glitch due to external interference) — the state machine returns to the Test-Logic-Reset state following three rising edges of JTG_TCK with the JTG_TMS line held at logic 1.

Test logic operation is designed such that no disturbance is caused to on-chip system logic operation as the result of such an error.

20.1.2 Run-Test/Idle State

The TAP controller enters the Run-Test/Idle state between scan operations. The controller remains in this state as long as the JTG_TMS is held at logic 0. For the IXP42X product line and IXC1100 control plane processors, the Run-Test/Idle state is strictly an idle state.

When the JTG_TMS is logic 1 on the rising edge of TCK, the controller moves to the Select-DR-Scan state.

20.1.3 Select-DR-Scan State

The Select-DR-Scan state is a temporary controller state. The test data registers selected by the current instruction retain their previous state.

If the JTG_TMS signal is held to logic 0 on the rising edge of JTG_TCK when the controller is in the Select-DR-Scan state, the controller moves into the Capture-DR state and a scan sequence — for the selected test data register — is initiated.

If JTG_TMS is held to logic 1 on the rising edge of JTG_TCK, the controller moves into the Select-IR-Scan state. The instruction does not change while the TAP controller is in this state.

20.1.4 Capture-DR State

When the controller is in this state and the current instruction is BS_SAMPLE/PRELOAD, the Boundary-Scan Register captures input pin data on the rising edge of JTG_TCK. Test data registers that do not have parallel inputs are not changed.

If the BS_SAMPLE/PRELOAD instruction is not selected, while in the Capture-DR state, the Boundary-Scan registers retain their previous state. In addition, any other data register under test will place the current value of the selected register into the shift register connected between JTG_TDI and JTG_TDO.

The instruction does not change while the TAP controller is in the Capture-DR state.

If JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller enters the Exit1-DR state. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller enters the Shift-DR state.

20.1.5 Shift-DR State

In the Shift-DR state, the test data register — which is connected between JTG_TDI and JTG_TDO as a result of the current instruction — shifts data one bit position nearer to its serial output on each rising edge of JTG_TCK. At the same time that data is shifted out, new data for updating the currently selected test register connected between JTG_TDI and JTG_TDO can be loaded with new data. Passing through the Update-DR state causes this newly loaded value to be captured by the currently selected test data register. Test data registers that the current instruction select but do not place in the serial path, retain their previous value during this state.

The instruction does not change while the TAP controller is in this state.

If JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller enters the Exit1-DR state. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller remains in the Shift-DR state.

20.1.6 Exit1-DR State

The Exit1-DR state is a temporary controller state. When the TAP controller is in the Exit1-DR state and JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller enters the Update-DR state, which terminates the scanning process. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller enters the Pause-DR state.

The instruction does not change while the TAP controller is in this state. All test data registers selected by the current instruction retain their previous value during this state.

20.1.7 Pause-DR State

The Pause-DR state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between JTG_TDI and JTG_TDO. The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as JTG_TMS is logic 0. When JTG_TMS changes to logic 1 on the rising edge of JTG_TCK, the controller moves to the Exit2-DR state.

20.1.8 Exit2-DR State

The Exit2-DR state is a temporary state. If JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller enters the Update-DR state, which terminates the scanning process. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller enters the Shift- DR state.

The instruction does not change while the TAP controller is in this state. All test-data registers selected by the current instruction retain their previous value during this state.

20.1.9 Update-DR State

The Boundary-Scan Register is provided with a latched, parallel output. This output prevents changes at the parallel output while data is shifted in response to the BS_EXTEST and BS_SAMPLE/PRELOAD instructions.

When the Boundary-Scan Register is selected, while the TAP controller is in the Update-DR state, data is latched onto the Boundary-Scan Register's parallel output from the shift-register path on the falling edge of JTG_TCK. The data held at the latched, parallel output does not change unless the controller is in this state.

Any other test-data register with parallel output registers will be updated with the value contained in the shift register — for the currently selected test and data register — when the Update-DR state is entered.

While the TAP controller is in this state, all of the test data register's shift-register bit positions selected by the current instruction retain their previous values.

The instruction does not change while the TAP controller is in this state.

When the TAP controller is in the Update-DR state and JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller enters the Select-DR-Scan state. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller enters the Run-Test/Idle state.

20.1.10 Select-IR-Scan State

The Select-IR Scan state is a temporary controller state. The test data registers selected by the current instruction retain their previous state.

In this state, if JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

20.1.11 Capture-IR State

When the controller is in the Capture-IR state, the shift register — implemented for the instruction register — loads the fixed value 0000001 on the rising edge of JTG_TCK as defined by the IEEE-1149 specification.

The test data register, selected by the current instruction, retains its previous value during this state. The instruction does not change in this state.

While in this state, the holding of JTG_TMS at logic 1 on the rising edge of JTG_TCK causes the controller to enter the Exit1-IR state. If JTG_TMS is held at logic 0 on the rising edge of JTG_TCK, the controller enters the Shift-IR state.

20.1.12 Shift-IR State

When the controller is in the Shift-IR state, the shift register, contained in the instruction register, is connected between JTG_TDI and JTG_TDO and shifts data one bit position nearer to its serial output on each rising edge of JTG_TCK. The shifting also allows the loading of the next instruction that is to be loaded into the TAP controller. The value of binary 0000001 loaded during the Capture-IR state will be shifted towards the JTG_TDO output.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change.

If JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller enters the Exit1-IR state. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller remains in the Shift-IR state.

20.1.13 Exit1-IR State

The Exit1-IR state is a temporary state. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller enters the Update-IR state, which terminates the scanning process. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state.

The instruction does not change and the instruction register retains its state.

20.1.14 Pause-IR State

The Pause-IR state allows the test controller to temporarily halt the shifting of data through the instruction register. The test data registers selected by the current instruction retain their previous values during this state.

The instruction does not change and the instruction register retains its state.

The controller remains in this state as long as JTG_TMS is logic 0. When JTG_TMS changes to logic 1 on the rising edges of JTG_TCK, the controller moves to the Exit2-IR state.

20.1.15 Exit2-IR State

The Exit2-IR state is a temporary state. If JTG_TMS is held high on the rising edge of JTG_TCK, the controller enters the Update-IR state, which terminates the scanning process. If JTG_TMS is held low on the rising edge of JTG_TCK, the controller enters the Shift- IR state.

This test data register selected by the current instruction retains its previous value during this state. The instruction does not change and the instruction register retains its state.

20.1.16 Update-IR State

The instruction shifted into the instruction register during the Shift-IR state is latched onto the parallel output from the shift-register path on the falling edge of JTG_TCK. Once latched, the new instruction becomes the current instruction. Test data registers selected by the current instruction retain their previous values.

If JTG_TMS is logic 1 on the rising edge of JTG_TCK, the controller enters the Select-DR-Scan state. If JTG_TMS is logic 0 on the rising edge of JTG_TCK, the controller enters the Run-Test/Idle state.

20.2 JTAG Instructions

Table 174 defines the instructions that either the Intel XScale processor or the IXP42X product line and IXC1100 control plane processors will implement.

Table 174. JTAG Instruction Set (Sheet 1 of 2)

Op Code	Intel XScale® Processor or IXP425 Op Code (X= XScale, I = IXP425, R = Reserved)	Instruction Name	Description
0000000	I	BS_EXTEST	Extest initiates testing of external circuitry, typically board-level interconnects and off-chip circuitry. Extest connects the Boundary Scan register between TDI and TDO in the Shift-DR state. When Extest is selected, all output signal pin values are driven by values shifted into the Boundary Scan Register and may change only on the falling edge of TCK in the Update-DR state. Also, when Extest is selected the input pin states must be loaded into the Boundary Scan Register on the rising edge of TCK in the Capture-DR state. Values shifted into input latches in the Boundary Scan Register are never used by the processors internal logic for the extest instruction.
0000001	I	BS_SAMPLE/PRELOAD	When the TAP controller is in the capture_dr state, the BS_Sample instruction provides a snapshot of data flowing from the system pins to the on-chip system logic, or vice versa (without interfering with normal operation). The snapshot is taken on the rising edge of TCK. When the TAP controller is in the Update_DR state, the BS_SAMPLE instruction causes the transfer of data held in the Boundary Scan cells to the latched, parallel output register cells. Typically, the latched output data is then applied to the system outputs by means of the BS_EXTEST instruction.
0000010	X	DBGRX	Used for accessing the RX Debug Register within the Intel XScale processor.

Table 174. JTAG Instruction Set (Sheet 2 of 2)

Op Code	Intel XScale® Processor or IXP425 Op Code (X= XScale, I = IXP425, R = Reserved)	Instruction Name	Description
0000011	R	Reserved	Reserved
0000111	X	LDIC	Used for loading the Intel XScale processor instruction cache.
0001001	X	DCSR	Used for accessing the Debug Control/Status Register.
0001010	X	BIST	Used for Intel XScale processor BIST operation.
0001100	X	SNAPDAT	Used for putting Intel XScale processor into snapshot mode.
0010000	X	DBGTX	Used for accessing the TX Debug Register within the Intel XScale processor.
1000000	R	Reserved	Reserved
1000001	R	Reserved	Reserved
1000010	R	Reserved	Reserved
1000011	R	Reserved	Reserved
1000100	R	Reserved	Reserved
1000101	R	Reserved	Reserved
1000110	R	Reserved	Reserved
1001001	I	BS_CLAMP	This instruction allows “guarding” values to be placed on the output pins of a device while connecting the Bypass Register between TDI and TDO in the shift_dr state. While the BS_CLAMP instruction is selected, the state of all signals driven from system output pins is defined by the data held in the Boundary Scan Register.
1001010	I	HIGH_Z	The Highz instruction places the device in a state such that all output and bidirectional pins are placed in an inactive drive state. The Highz instruction connects the Bypass Register between TDI and TDO in the shift_dr state.
1111110	X	IDCODE	Connects the ID register in Intel XScale processor between TDI and TDO during the TAP controller’s shift_dr state. The IDCODE Register will be a 32-bit register and will contain the following values when read: 533 MHz speed grade = 0x09274013 400 MHz speed grade = 0x09275013 266 MHz speed grade = 0x09277013
Any Others	X	BYPASS	All other instructions will connect a single bit Bypass Register between TDI and TDO.

20.3 Data Registers

The Data Registers are essentially a shift register and a read/write register juxtaposed. In other words, the shift register is used to shift data in and out and the read/write register gets parallel loaded by the shift register during the Update-DR state and writes back to the shift register (again, parallel-wise) during the Capture-DR state.

Typically, the data contained in the read/write registers are configuration bits, status bits, or ID bits. The manner in which each data register is read and written, follows the IEEE 1149.1 protocol using JTG_TDI, JTG_TDO, JTG_TMS, JTG_TRST_N, and JTG_TCK signals. All the data registers (with the exception of the Instruction Register) can be serially loaded with the corresponding data-register_write or data-register_read instruction loaded in the JTAG Instruction Register along with the TAP Controller being in the Shift-DR state.

Data is received, from JTG_TDI, through a shift register and exits through JTG_TDO one bit at a time on the rising edge of JTG_TCK. The data can then be copied (parallel loaded) into the associated read/write register when the data-register_write command is loaded in the JTAG Instruction Register and the TAP controller enters the Update-DR state.

Finally, when the TAP controller enters the Capture-DR state and the JTAG Instruction Register has either data-register_write or data-register_read in it, the read/write register data is copied into the associated shift register and will be out to TDO when the TAP controller enters the Shift-DR state.

20.3.1 Boundary Scan Register

The Boundary Scan Register is a shift register, comprised of boundary-scan cells in the pads. Power, ground, and JTAG pins are not equipped with boundary-scan cells. The Boundary Scan Register is fully IEEE 1149.1-compliant. This N-bit register is connected between TDI and TDO when BS_EXTEST**,** BS_SAMPLE/PRELOAD, or the BS_CLAMP instructions are selected. The Boundary Scan Register is used for capturing signal-pin data on the input pins, forcing fixed values on the output signal pins and selecting the direction and drive characteristics (a logic value or high impedance) of the bidirectional and three-state signal pins.

Also, if BS_SAMPLE/PRELOAD or BS_EXTEST are loaded into the JTAG Instruction Register, the TAP controller states SHIFT-DR, CAPTURE-DR, and UPDATE-DR are used to conduct the boundary-scan testing. SHIFT-DR will now control the shift-select multiplexer in the boundary scan logic of the pads, CAPTURE-DR will clock the boundary scan shift register, and UPDATE-DR will drive the boundary scan data to either the internal logic of the IXP42X product line and IXC1100 control plane processors or to a pin depending on the direction of the pin (input or output).

20.3.2 Instruction Register

The JTAG Instruction Register, that resides in the Intel XScale processor JTAG unit, behaves similar to the Data Registers. No instruction is required to load the JTAG Instruction Register and the pertinent, relative TAP controller states are Shift-IR, Capture-IR, and Update-IR.

The Intel XScale processor provides the Test Logic Unit with the instruction status so that the Test Logic Unit can decode the current instruction and act accordingly.

20.3.3 JTAG Device ID Register

Table 175. JTAG Device Register Values

JTAG ID (32-Bit Value)	Speed (MHz)
0x19274013	533
0x19275013	400
0x19277013	266

§ §

21.0 AHB Queue Manager (AQM)

The purpose of this chapter is to outline the functionality of the AHB Queue Manager (AQM) which helps users to better understand the software and hardware architecture. The Intel® IXP400 Software manages these queues.

21.1 Overview

The AHB Queue Manager (AQM) provides queue functionality for various internal blocks. It maintains the queues as circular buffers in an embedded 8KB SRAM. It also implements the status flags and pointers required for each queue.

The AQM manages 64 independent queues. Each queue is configurable for buffer and entry size. Additionally status flags are maintained for each queue.

The AQM interfaces include an AHB interface to the NPEs and Intel XScale® Processor (or any other AHB bus master), a Flag Bus interface, an event bus (to the NPE condition select logic) and two interrupts to the Intel XScale processor. The AHB interface is used for configuration of the AQM and provides access to queues, queue status and SRAM. Individual queue status for queues 0-31 is communicated to the NPEs via the flag bus. Combined queue status for queues 32-63 are communicated to the NPEs via the event bus. The two interrupts, one for queues 0-31 and one for queues 32-63, provide status interrupts to the Intel XScale processor.

Read or write entries to a queue, will be accomplished by performing AHB read/write accesses to any of the corresponding Queue Access Register addresses. The AQM will intercept these accesses, since no physical data resides at these addresses, and lookup the appropriate queue pointer to perform the requested read or write. Upon a read or write access to a queue, the pointers and status for the queue are updated as needed. Further detail is given in the following sections.

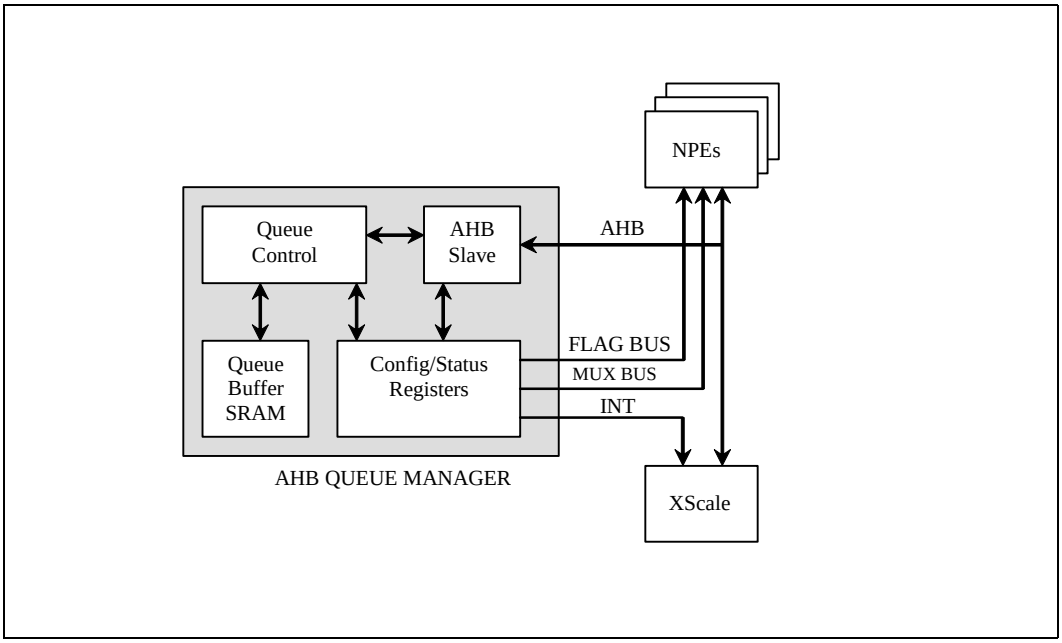
21.2 Feature List

- Provides queue functionality for NPEs and Intel XScale processor
- Manages 64 independent queues
- Implements queues as FIFOs with circular buffer rotation in SRAM
- Implements read/write pointers for each queue in SRAM
- Programmable queue entry size supported (i.e., queues may be configured for 1, 2, or 4 word entries)
- Programmable queue size supported (i.e., queues may be configured for 16, 32, 64 or 128 word buffer)
- Maintains empty (E), nearly empty (NE), nearly full (NF), and full (F) status flags on each of the queues 0-31
- Programmable queue watermarks for assessing NE and NF queue status flags
- Provides status flag information, E, NE, F and NF, for queues 0-31 to the NPEs via a common Flag Bus

- Provides Underflow and Overflow Status Flags for each of the queues 0-31
- Two Intel XScale processor interrupts, one for queues 0-31 and one for queues 32-63
- Individual interrupt enables for each queue
- Programmable interrupt source for each of the queues 0-31 as the assertion or de-assertion of 1 of 4 status flags, E, NE, NF or F
- NE status flag used as the interrupt source for each of the queues 32-63
- Provides read/write access to all queues, queue pointers, status flags, configuration registers, interrupt registers and SRAM via the AHB

21.3 Functional Description

Figure 101. AHB Queue Manager



A block diagram of the AHB Queue Manager is shown in Figure 101. The AQM provides 64 independent queues. It maintains these queues as circular buffers in an internal 8KB SRAM. Status flags are implemented for each queue to indicate relative fullness of each queue. The status flags for queues 0-31 are mapped and transmitted to the NPEs via the Flag Bus. External agents, specific to each of the cores, latch queue information relevant to them from the Flag Bus. Two interrupts, one for queues 0-31 and one for queues 32-63, are used as queue status interrupts. The AHB interface provides for complete queue configuration, queue access, queue status access, interrupt configuration and SRAM access.

The AQM provides for autonomous access to the queues. Via the AHB interface, any master on the AHB can request a queue read or a queue write operation. The AQM has no knowledge of either the contents of a queue entry or to whom these queues are assigned, or what are the contents or semantic meaning of the queue. The AQM will respond to a queue request by fetching the read or write pointer to the requested queue and then perform the requested operation. For a queue read request, the data is

returned via the AHB and for a queue write request, the data from the AHB is written into the queue. Following a queue access, the read or write pointer is incremented and the status for the accessed queue is updated as needed

21.4 AHB Interface

The AHB interface provides read/write access to all AQM configuration/status registers, queues and SRAM. The AQM is a slave with a 32 bit data bus configuration on the AHB. The address map for the AQM is shown in Table 176. Support is provided for the AHB slave requirements outlined in the AMBA specification, Rev. 2.0. Unsupported exceptions to the AHB slave requirements include accesses with a data transfer size of byte or half-word, wrapping burst accesses and 16 beat incrementing burst accesses. These accesses will result in an Error response. In addition, early burst termination support, master busy support, Retry responses and Split responses will not be implemented in the AQM. Accesses to any unused locations within the AQM address space will result in an OKAY response on the AHB. Read accesses to the unused address locations will result in zeroes returned on the AHB. The AQM will not perform any internal operations on write accesses to any unused locations. Wait state performance for the AHB interface is given in Section 21.4. Data formats for all registers accessible via the AHB is given in Section 21.5.

Table 176. AHB Queue Manager Memory Map

Address	AQM Function
0x03FFF 0x02100	64 Queue Buffer Space - SRAM 1984 x 4 Bytes
0x020FF 0x02000	64 Queue Configuration Words - SRAM 64 x 4 Bytes
0x01FFF 0x00440	Reserved
0x0043F 0x00438	Queue 0 to 63 Interrupt Register 2 x 4 Bytes
0x00437 0x00430	Queue 0 to 63 Interrupt Enable Register 2 x 4 Bytes
0x0042F 0x00420	Queue 0 to 31Interrupt Status Flag Source Select Register 4x 4 Bytes
0x0041C	Queue 32 to 63 Full Status Register
0x00418	Queue 32 to 63 Nearly Empty Status Register

Table 176. AHB Queue Manager Memory Map

0x00417 0x00410	Queue 0 to 31 Underflow/Overflow Status Register 2 x 4 Bytes
0x0040F 0x00400	Queue 0 to 31 Interrupt Status Register 4 x 4 bytes
0x003FF 0x00000	Queue 0 to 63 Read/Write Access 64 x 16 bytes

21.4.1 Queue Control

The queues are implemented as circular buffers where adding an entry is performed by a write to a queue and removing an entry is performed by a read from a queue. Entries are read from a queue in the same order in which they were written to the queue. The read/write pointers track the removal/addition of entries from/to a queue. The queue control performs the autonomous access of the queues. External agents wanting to access a queue, will perform an AHB read or write to the Queue Access Register locations. As a result of the access to these locations, the AQM will perform the requested access to the queue in SRAM. Support is provided for 64 queues. Upon receiving a queue read or queue write from the AHB interface, queue control fetches the selected queue configuration from SRAM. The queues or circular buffers will reside in internal SRAM. Configuration for each queue will consists of:

- A Base Address – this is the address where the queue starts and is configurable for placing the queue buffer on any 16 word boundary within the SRAM address range of 000H to 7C0H (word address).
- A Write Pointer – this is a pointer to the next queue location to be written and is maintained by queue control.
- A Read Pointer – this is a pointer to the next location to be read and is maintained by queue control.
- Queue Entry Size – this indicates size of each queue entry and is configurable for 1, 2, or 4 words.
- Queue Size – this indicates the number of words allocated to the queue and is configurable for 16, 32, 64 or 128 words.
- NE watermark – this field indicates the maximum number of occupied entries for which a queue is considered to be nearly empty. It can be set to 0, 1, 2, 4, 8, 16, 32, or 64 entries.
- NF watermark – this field indicates the maximum number of empty entries for which a queue is considered to be nearly full. It can be set to 0, 1, 2, 4, 8, 16, 32, or 64 entries.

To access any given queue, after the selected queue configuration is read from SRAM, the queue base address is summed with the read or write pointer to form the queue address. If the queue isn't empty, when the request is a read, or full, when the request is a write, queue control will perform the requested queue access at the calculated queue address. If the request is a read, the queue data read from SRAM at the calculated queue address is passed to the AHB interface for the corresponding data acknowledge to the AHB read request. If the request is a write, the data from the AHB

interface is written into SRAM at the calculated queue address. When the read and write pointers are equal, the queue is either full or empty as determined by the full or empty status flags.

When a read request of an empty queue buffer is performed, queue control will return zeroes in the data field to the AHB interface and will set the Underflow Status Flag. When a write request of a full queue is performed, queue control will set the Overflow Status Flag. Underflow and Overflow Status Flags are maintained on queues 0-31 only. Otherwise for a read request of an empty queue or write request of a full queue, queue control will not perform any action upon the queue buffer or queue configuration word.

Following the queue access, the appropriate read or write pointer, as indicated by the type of queue access, is incremented and the queue configuration word, with all other fields maintained, is written back into SRAM. The queue size will be used in determining the number of active bits within the allocated 7 bit field for the read and write pointers. Configurable queue sizes of 16, 32, 64 and 128 directly correspond to read and write active bit widths of 4, 5, 6, and 7. The unused bits of the read and write pointers will be zeroed prior to writing the queue configuration word back into SRAM.

One to four Queue Access Register addresses, 0, 4, 8, and 0xC, are allocated per queue as determined by the queue's programmed entry size. Thus a queue, with an entry size set to one word, will support accesses to the first location, 0, and a queue with an entry size set to two, will support accesses to the first and second locations, 0 and 4. A queue with an entry size set to four words will support accesses to all four locations. Accesses to the non-supported locations will not be performed and queue read/write pointers will be unchanged. A queue, with a programmed entry size of two or four words, requires the two or four accesses of each queue entry to be performed sequentially beginning with address 0. Accesses performed out of order will not be performed. Incomplete accesses (e.g. reading only the first two words of a four word entry) will not update the pointers. The sequential accesses can be performed via multiple single word accesses or via a burst access on the AHB. If a Queue burst access of more than four words is attempted, the AQM will perform the accesses to the first two or four locations, as determined by the queue's entry size, and the remainder of the accesses of the burst will not be performed. The AQM will respond with the OKAY response on the AHB when the queue accesses aren't performed. Queue read accesses which are not performed, will return zeroes in the data field on the AHB.

21.4.2 Queue Status

Status information for the 64 queues is provided in the status registers. Six status flags will be maintained for each of the queues 0-31, empty, nearly empty, full, nearly full, underflow, and overflow. Only the four Empty, Nearly Empty, Nearly Full, and Full Status Flags are provided for queues 32-63. The status flags will be read/write accessible via the AHB, although the arrangement of flags differs between queues 0-31 and 32-63. The Flag Bus will communicate status for queues 0-31 to the NPEs, while two interrupts will provide status interrupting capability for the Intel XScale processor. The following sections outline the queue status requirements.

21.4.2.1 Status Update

Following any updates for a queue access, the read and write pointers will be used in determination of status flag settings for the accessed queue. When the queue entry size is set to 1, status flags are updated following every queue access. If the queues are set for multi-word entry sizes, 2 or 4, the status flags will be updated following the queue access, that completely fills or empties a queue entry. If the read and write pointers are equal and the last access to the queue was a read, then the queue is empty. If the read and write pointers are equal and the last access was a write, then the queue is full. The nearly empty and nearly full configurable watermarks are used in determining the settings for the NE and NF Status flags. These watermarks can be set to 0, 1, 2, 4, 8, 16, 32, or 64 entries. If the number of completely empty entries is less

than or equal to the full watermark, the queue is considered nearly full. If the number of completely full entries is less than or equal to the empty watermark, the queue is considered nearly empty.

Of course, status may be read at any time by an AHB read of the appropriate status register. The status read will reflect the status of the queue at the time of the read, respecting all previous AHB operations. By example, if two closely spaced AHB reads are performed, one to a queue and another to that queue's status register, the status register read will reflect the completion of the queue read. This will require that the second read have additional wait states inserted (by hardware) into the bus operation in order to insure that the first operation has been completed.

The operation of the Nearly Full, Nearly Empty, Full and Empty Status Flags with the Nearly Empty & Nearly Full watermarks set to various levels is demonstrated in Table 177. For this example, the buffer size is set to 64 and the entry size is set to 1. When the watermarks are set to zero, the Nearly Empty and Empty Flags will be identical and the Nearly Full and Full Flags will be identical.

Please note in the following table the "# Entries in the Queue" represents the cardinal number of entries in the queue for each watermark value.

Table 177. Queue Status Flags

Nearly Empty Watermark	Nearly Full Watermark	# Entries in the Queue	E	NE	NF	F
0 (000)	0 (000)	0	1	1	0	0
		1 – 62	0	0	0	0
		63	0	0	1	1
1 (001)	1 (001)	0	1	1	0	0
		1	0	1	0	0
		2 – 61	0	0	0	0
		62	0	0	1	0
		63	0	0	1	1
2 (010)	2 (010)	0	1	1	0	0
		1 – 2	0	1	0	0
		3 - 60	0	0	0	0
		61 – 62	0	0	1	0
		63	0	0	1	1
4 (011)	4 (011)	0	1	1	0	0
		1 – 4	0	1	0	0
		5 – 58	0	0	0	0
		59- 62	0	0	1	0
		63	0	0	1	1
8 (100)	8 (100)	0	1	1	0	0
		1 - 8	0	1	0	0
		9 – 54	0	0	0	0
		55 - 62	0	0	1	0
		63	0	0	1	1

21.4.2.2 Flag Bus

Status, for queues 0-31 only, will be directly communicated to the NPE via a dedicated interface, referred to as the Flag Bus. The Flag Bus is composed of 4 independent strobe signals, aqm_flag_strb[3:0], a queue identification bus, aqm_flag_que_id[4:0], and a queue status bus, aqm_flag_que_stat[3:0]. The queue status bus will provide the empty, nearly empty, nearly full and full status flags respectively on bits 0-3 of the

bus. Following each queue access where queue status is updated, status will be transmitted on the Flag Bus, and only then. The Flag Bus Strobe(s) will be asserted high for one clock cycle to indicate the presence of status on the Flag Bus.

21.4.2.3 Status Interrupts

Two processor interrupts will be provided, one for queues 0-31, aqm_int[0], and one for queues 32-63, aqm_int[1]. Each of the interrupt signals is computed as a masked 32-way logical-OR of one edge-sensitive status bit per queue. In other words, each queue contributes a single edge-sensitive input into one of the 32-way logical-OR combinations. For queues 0-31, this input is independently configurable. It may be a positive or negative edge-sensitive version of any one of the E, NE, NF or F status flag bits. The selected status flag may be different for each queue. For queues 32-63, the input is always the NE status flag bit with a positive edge-sensitive version only. The set of selected 32 condition signals is masked by the corresponding interrupt enable register prior to computing the logical-OR.

These interrupts are generated for active high, level triggered usage. On occurrence of the selected transition of one or more of the status flag sources, an active high interrupt level is registered. Via the AHB, the processor can read a 32-bit Interrupt register to determine the source or sources for each interrupt. Selective interrupt reset capability will be provided for each of the queue sources via writing a one to the appropriate queue bit(s) within the interrupt register. Upon clearing (i.e. writing a ‘1’ to) the appropriate bit(s) in the Interrupt Register, the interrupt cannot be generated again by the same source, until the active status flag condition is removed and then are asserted again.

There is a bit in INT0SRCSELREG0 which will modify the reset operation of the interrupts. If this bit is set to 0, then the interrupts will reset as described in the above paragraph. If this bit is set to 1, then the interrupts will only reset if the interrupting condition has also been cleared when the write the QUEINTREG occurs. In other words, this bit determines if the interrupt is globally rising edge sensitive (INT0SCRSELREG0 is ‘0’) or is level sensitive (INT0SCRSELREG0 is ‘1’).

21.5 Register Descriptions

21.5.1 Queue Access Word Registers 0 - 63

External agents wanting to access a queue, will perform an AHB read or write to the Queue Access Register locations. As a result of the access to these locations, the AQM will perform the requested access to the queue in SRAM. See Section 21.4.1 for clarification on AHB queue accesses to the AQM. As described above, these queue access registers are defined in a block of four 32-bit words, where only the first 32-bit word is defined for a word size of one, only the first two 32-bit words are defined for a word size of two and all four 32-bit words are defined for a word size of four.

Register Name:	QUEACC (0 <= n <=63)		
Physical Address:	Queue #n 0x(0000 + 16n + 4x)	Reset Hex Value:	Not Applicable
Register Description:	Queue #n access register. There are 1-4 addresses (0 <= x <=3), as determined by the programmed entry size, for requesting read/write accesses to individual queues. No physical data resides at these addresses.		

Access: Read/Write

31							24	23						16	15						8	7						0
Queue Read/Write Data																												

21.5.2 Queues 0-31 Status Register 0 - 3

The access to these status registers is read/write, however except for initialization, diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually *change* the status, it only writes the shadow register which contains the status.

Register Name:	QUELOWSTAT (0 <= n <=3)		
Physical Address:	Reg #n 0x(0400 + 4n)	Reset Hex Value:	0x33333333
Register Description:	Queue status register for the queues 0-31. F/NF/NE/E: '1' – active flag		

Access: Read/Write

31														16	15						8	7						0
Queue(8n+7)				Queue(8n+6)				Queue(8n+5)				Queue(8n+4)			Queue(8n+3)				Queue(8n+2)			Queue(8n+1)				Queue(8n)			

21.5.3 Underflow/Overflow Status Register 0 - 1

The access to these status registers is read/write, however except for initialization, diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:	QUEUOSTAT (0 <= n <=1)		
Physical Address:	Queue #n 0x(0410 + 4n)	Reset Hex Value:	0x00000000
Register Description:	Queue underflow/overflow status register for the queues 0-31. OF/UF: '1' – Overflow/Underflow has occurred		

Access: Read / Write

31							24	23					16	15						8	7						0
Queue (16n + 15)		Queue (16n + 14)		Queue (16n + 13)		Queue (16n + 12)		Queue (16n + 11)		Queue (16n + 10)		Queue (16n + 9)		Queue (16n + 8)		Queue (16n + 7)		Queue (16n + 6)		Queue (16n + 5)		Queue (16n + 4)		Queue (16n + 3)		Queue (16n + 2)		Queue (16n + 1)		Queue (16n)

21.5.4 Queues 32-63 Nearly Empty Status Register

The access to these status registers is read/write, however except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:	QUEUPPSTATNE		
Physical Address:	0x0418	Reset Hex Value:	0xFFFFFFFF
Register Description:	Queue status register for queues 32-63. NE: '1' – flag set		

Access: Read / Write

31															16	15							8	7							0
Q63 NE	Q62 NE	Q61 NE	Q60 NE	Q59 NE	Q58 NE	Q57 NE	Q56 NE	Q55 NE	Q54 NE	Q53 NE	Q52 NE	Q51 NE	Q50 NE	Q49 NE	Q48 NE	Q47 NE	Q46 NE	Q45 NE	Q44 NE	Q43 NE	Q42 NE	Q41 NE	Q40 NE	Q39 NE	Q38 NE	Q37 NE	Q36 NE	Q35 NE	Q34 NE	Q33 NE	Q32 NE

21.5.5 Queues 32-63 Full Status Register

The access to these status registers is read/write, however except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:	QUEUPPSTATF		
Physical Address:	0x041C	Reset Hex Value:	0x00000000
Register Description:	Queue status register for queues 32-63. F: '1' – flag set		
Access: Read / Write			

31							24	23							16	15							8	7							0
Q63 F	Q62 F	Q61 F	Q60 F	Q59 F	Q58 F	Q57 F	Q56 F	Q55 F	Q54 F	Q53 F	Q52 F	Q51 F	Q50 F	Q49 F	Q48 F	Q47 F	Q46 F	Q45 F	Q44 F	Q43 F	Q42 F	Q41 F	Q40 F	Q39 F	Q38 F	Q37 F	Q36 F	Q35 F	Q34 F	Q33 F	Q32 F

21.5.6 Interrupt 0 Status Flag Source Select Register 0 – 3

The interrupt source for each queue is selectable as the positive or negative (NOT) edge-sensitive version of any one of the E, NE, NF or F status flag bits on interrupt 0, aqm_int[0]. The selection is configurable for interrupt 0 only, while interrupt 1, aqm_int[1], is hard wired to the NE Status Flag bit.

Register Name:	INT0SRCSELREG (0 <= n <=3)		
Physical Address:	Reg #n 0x(0420 + 4n)	Reset Hex Value:	0x00000000
Register Description:	Status Flag selection for interrupt 0 source on queues 0-31.		
Access: Read / Write			

31							24	23							16	15							8	7							0
Reserved	Queue (8n + 7) Stat Src Sel			Reserved	Queue (8n + 6) Stat Src Sel			Reserved	Queue (8n + 5) Stat Src Sel			Reserved	Queue (8n + 4) Stat Src Sel			Reserved	Queue (8n + 3) Stat Src Sel			Reserved	Queue (8n + 2) Stat Src Sel			Reserved	Queue (8n + 1) Stat Src Sel			Spec for 0	Queue (8n) Stat Src Sel			

21.5.7 Queue Interrupt Enable Register 0 – 1

Register Name:	QUEIEREG(0 <= n <=1)		
Physical Address:	Reg #n 0x(0430 + 4n)	Reset Hex Value:	0x00000000
Register Description:	Interrupt enables for the queues 0-63. IE: '1' – Enable		

Access: Read / Write

31							24	23							16	15						8	7							0
Q(32n + 31) IE	Q(32n + 30) IE	Q(32n + 29) IE	Q(32n + 28) IE	Q(32n + 27) IE	Q(32n + 26) IE	Q(32n + 25) IE	Q(32n + 24) IE	Q(32n + 23) IE	Q(32n + 22) IE	Q(32n + 21) IE	Q(32n + 20) IE	Q(32n + 19) IE	Q(32n + 18) IE	Q(32n + 17) IE	Q(32n + 16) IE	Q(32n + 15) IE	Q(32n + 14) IE	Q(32n + 13) IE	Q(32n + 12) IE	Q(32n + 11) IE	Q(32n + 10) IE	Q(32n + 9) IE	Q(32n + 8) IE	Q(32n + 7) IE	Q(32n + 6) IE	Q(32n + 5) IE	Q(32n + 4) IE	Q(32n + 3) IE	Q(32n + 2) IE	Q(32n + 1) IE	Q(32n) IE

21.5.8 Queue Interrupt Register 0 – 1

There are two interrupt registers corresponding respectively to the two AQM interrupts, aqm_int[0] and aqm_int[1]. Queue Interrupt register 0 represents queues 0-31, while register 1 represents queues 32-63. Following an interrupt, the appropriate register can be read to determine which queue or queues caused the interrupt. Any bit in the interrupt register can be cleared via writing a one to the appropriate bit position. Writing a'1' to a bit in the Interrupt register will provide a reset only operation for that bit. Clearing all set bits (by writing '1's in those locations) in the Interrupt Register will remove the interrupt (de-assert). The interrupt cannot be generated again by the same source, until the active status flag condition is removed and then reasserted again. The bit INT0SRCSELREG0 in the INT0SRCSELREG0 register changes the operation of the interrupt from a rising edge sensitive operation to a level sensitive one.

Register Name:	QUEINTREG(0 <= n <=1)		
Physical Address:	Reg #n 0x(0438 + 4n)	Reset Hex Value:	0x00000000
Register Description:	Interrupt Register for the 64 queues. INT: '1' – interrupt occurred		

Access: Read / Write to Reset

31							24	23							16	15						8	7							0
Q(32n + 31) INT	Q(32n + 30) INT	Q(32n + 29) INT	Q(32n + 28) INT	Q(32n + 27) INT	Q(32n + 26) INT	Q(32n + 25) INT	Q(32n + 24) INT	Q(32n + 23) INT	Q(32n + 22) INT	Q(32n + 21) INT	Q(32n + 20) INT	Q(32n + 19) INT	Q(32n + 18) INT	Q(32n + 17) INT	Q(32n + 16) INT	Q(32n + 15) INT	Q(32n + 14) INT	Q(32n + 13) INT	Q(32n + 12) INT	Q(32n + 11) INT	Q(32n + 10) INT	Q(32n + 9) INT	Q(32n + 8) INT	Q(32n + 7) INT	Q(32n + 6) INT	Q(32n + 5) INT	Q(32n + 4) INT	Q(32n + 3) INT	Q(32n + 2) INT	Q(32n + 1) INT	Q(32n) INT

21.5.9 Queue Configuration Words 0 - 63

The 64 queue configuration words are located in internal SRAM and require initialization prior to AQM usage. The read and write pointers need to be cleared on initialization, since this reflects an empty queue. A system reset sets the status registers to reflect

empty queues but until the queue configuration words have been set, this state is somewhat inconsistent. Write accesses to any of the Queue Configuration Words 0-31 cause the corresponding queue status to be communicated on the Flag Bus. Once the AQM is fully configured, these configuration words should be used for read only purposes to monitor queue pointers. If a queue needs to be reset, a pair of operations are required. First the appropriate queue status flags must be configured then the Queue Configuration Word must be set. The data format for the Queue Configuration Word is shown in the QUECONFIG register.

Reading and/or writing to a queue where the queue configuration word has not been initialized will produce undefined operations and may cause spurious parity errors and/or data corruption of another queue.

Register Name:	QUECONFIG (0 <= n <=63)		
Physical Address:	Queue #n 0x(2000 + 4n)	Reset Hex Value:	0xUUUUUUUU
Register Description:	Queue #n configuration word located in SRAM.		
Access:			

31							24	23						16	15						8	7							0
Queue (n) NF Watermark			Queue (n) NE Watermark			Q (n) Buffer Size		Q (n) Entry Size		Queue (n) Base Address					Res.		Queue (n) Read Pointer						Queue (n) Write Pointer						

Register	QUECONFIG (0 <= n <=63)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:29	Nearly Full Watermark	The Nearly Full Watermark can be configured via these bits. The Nearly Full Watermark can be set in the range 0 (“000”), 1 (“001”), 2 (“010”), 04(“011”), 8 (“100”), 16 (“101”), 32 (“110”), or 64 (“111”) entries from the top of the queue. The usable range for the Nearly Full Watermark selection is limited to a value less than the buffer size.	U	R/W
28:26	Nearly Empty Watermark	The Nearly Empty Watermark can be configured via these bits. The Nearly Empty Watermark can be set in the range 0 (“000”), 1 (“001”), 2 (“010”), 04(“011”), 8 (“100”), 16 (“101”), 32 (“110”), or 64 (“111”) entries from the bottom of the queue. The usable range for the Nearly Empty Watermark selection is limited to a value less than the buffer size.	U	R/W
25:24	Buffer Size	These bits are used to configure the queue buffer size. The buffer size can be configured for 16 (“00”), 32 (“01”), 64 (“10”) or 128 (“11”) words	U	R/W
23:22	Entry Size	These bits are used to configure the queue entry size of the queue. The entry size can be set at 1 (“00”) or 2 (“01”) or 4 (“10”) words. An input of “11” results in the entry size being set at 1.	U	R/W
21:14	Base Address	This field is used to configure the starting base address of the queue. The read and write pointer are offset addresses from the base address. This base address is a SRAM 16 word address. Base addresses, 00 to 03, are reserved for the Queue Configuration Words. The most significant bit of the base address will be reserved for growth to a 16 KB SRAM, thus providing 8 KB of additional queue buffer space.	U	R/W
13:7	Read Pointer	This is a pointer to the next entry to be read from the queue. The pointer is the AQM’s internal SRAM word address. In general, this pointer should initialized to zero, and not be written except for diagnostic or test purposes.	U	R/W
6:0	Write Pointer	This is a pointer to the next entry to be written to the queue. The pointer is the AQM’s internal SRAM word address. In general, this pointer should initialized to zero, and not be written except for diagnostic or test purposes.	U	R/W

§ §