MOTOROLA
intelligence everywhere™

digital dna™

Sensorless BLDC Motor Control Using the MC68HC908MR32

Designer Reference Manual

M68HC08
Microcontrollers

DRM028/D
Rev. 0, 03/2003

MOTOROLA.COM/SEMICONDUCTORS

Sensorless BLDC Motor Control Using the MC68HC908MR32

Designer Reference Manual — Rev 0

by: Libor Prokop
Motorola Czech System Laboratories
Roznov pod Radhostem, Czech Republic

Revision history

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.motorola.com/semiconductors

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision history

Date	Revision Level	Description	Page Number(s)
February, 2003	1	Initial release	N/A

List of Sections

List of Sections

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

Section 1. Introduction

1.1 Contents

1.2 Application Functionality

This Reference Design describes the design of a low-cost sensorless 3-phase brushless dc (BLDC) motor control with back-EMF (electromotive force) zero-crossing sensing. It is based on Motorola's MC68HC908MR32 microcontroller which is dedicated for motor control applications. The system is designed as a motor drive system for medium power three phase BLDC motors and is targeted for applications in automotive, industrial and appliance fields (e.g. compressors, air conditioning units, pumps or simple industrial drives). The reference design incorporates both hardware and software parts of the system including hardware schematics.

1.3 Benefits of the Solution

The design of very low cost variable speed BLDC motor control drives has become a prime focus point for the appliance designers and semiconductor suppliers.

Today more and more variable speed drives are put in appliance or automotive products to increase the whole system efficiency and the product performance. Using of the control systems based on semiconductor components and MCUs is mandatory to satisfy requirements for high efficiency, performance and cost of the system.

Introduction

Once using the semiconductor components, it is opened to replace classical universal and DC-motors with maintenance-free electrically commutated BLDC motors. This brings many advantages of BLDC motors when the system costs could be maintained equivalent.

The advantages of BLDC motor versus universal and DC-motors are:

- high efficiency
- reliability (no brushes)
- low noise
- easy to drive features

To control the BLDC motor, the rotor position must be known at certain angles in order to align the applied voltage with the back-EMF, which is induced in the stator winding due to the movement of the permanent magnets on the rotor.

Although some BLDC drives uses sensors for position sensing, there is a trend to use sensorless control. The position is then evaluated from voltage or current going to the motor. One of the sensorless technique is sensorless BLDC control with back-EMF (electromotive force) zero-crossing sensing.

The advantages of this control are:

- Save cost of the position sensors & wiring
- Can be used where there is impossibility or expansive to make additional connections between position sensors and the control unit
- Low cost system (medium demand for control MCU power)

Section 2. System Description

2.1 Contents

2.2 System Concept

The application block diagram is shown in **Figure 2-1**. The sensorless rotor position technique detects the zero crossing points of back-EMF induced in the motor windings. The phase back-EMF zero crossing points are sensed while one of the three phase windings is not powered. The information obtained is processed in order to commutate the energized phase pair and control the phase voltage, using pulse width modulation.

The back-EMF zero crossing detection enables position recognition. The resistor network is used to step down sensed voltages to a 0–3.3 V level. Zero crossing detection is synchronized with the middle of center aligned PWM signals by the software, in order to filter high voltage spikes produced by switching the IGBTs (MOSFETs). The software selects by MUX command the phase comparator output that corresponds to the current commutation step. The multiplexer (MUX) circuit selects this signal, which is then transferred to the MCU input.

The voltage drop resistor is used to measure the dc-bus current which is chopped by the pulse-width modulator (PWM). The signal obtained is rectified and amplified (0–3.3 V with 1.65 V offset). The internal MCU analog-to-digital (A/D) converter and zero crossing detection are synchronized with the PWM signal. This synchronization avoids spikes when the IGBTs (or MOSFETs) are switched and simplifies the electric circuit.

System Description

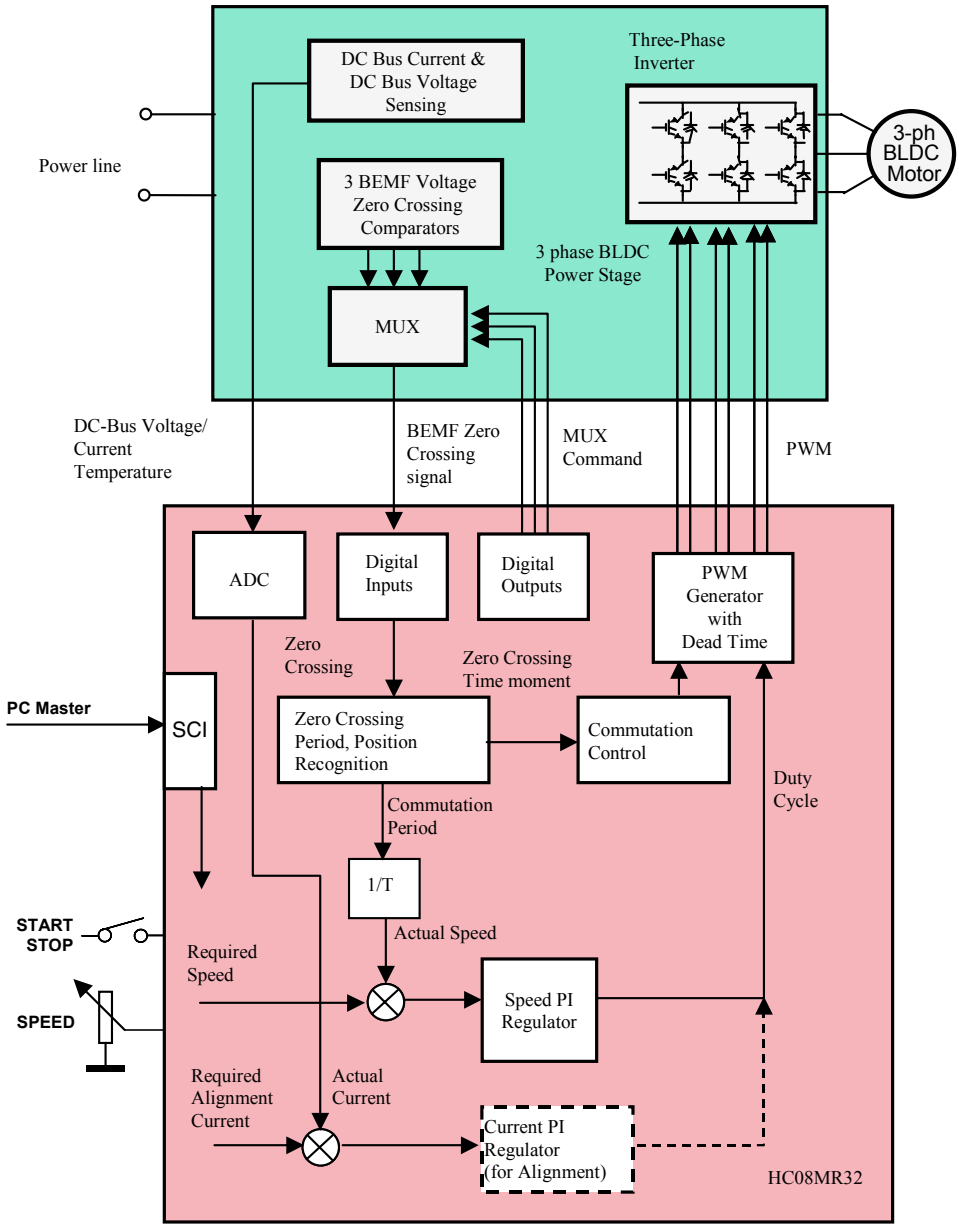

Figure 2-1System Concept

During the rotor alignment state, the dc-bus current is controlled by the current PI regulator. In the other states (motor running), the phase voltage (PWM duty cycle) is controlled by the speed PI regulator.

The A/D converter is also used to sense the dc-bus voltage and the drive temperature. The dc-bus voltage is stepped down to a 3.3-V signal level by a resistor network.

The six IGBTs (copack with built-in fly back diode), or MOSFETs, and gate drivers create a compact power stage. The drivers provide the level shifting that is required to drive the high side switch. The PWM technique is used to control motor phase voltage.

2.3 System Specification

The concept of the application is that of a speed-closed loop drive using back-EMF zero crossing technique for position detection. It serves as an example of a sensorless BLDC motor control system using Motorola's MC68HC908MR32 MCU. It also illustrates the usage of dedicated motor control on-chip peripherals.

The system for BLDC motor control consists of hardware and software. The application uses universal modular motion control development hardware boards, which are provided by Motorola for customer development support. For a description of these hardware boards refer to **Appendix A. References 3.**,**4.**,**5.**,**6.**,**7.**, and the World Wide Web at:

http://www.motorola.com

There are three board and motor hardware sets for the application:

1. High-Voltage Hardware Set — For variable line voltage 115–230 Vac and medium power (phase current < 2.93 A)
2. Low-Voltage Evaluation Motor Hardware Set — For automotive voltage (12 V) and very low power (phase current < 4 A)
3. Low-Voltage Hardware Set — For automotive voltage (12 V or possibly 42 V) and medium power (phase current < 50 A)

2.3.1 Software Specification

The application software is practically the same for all three hardware platforms. The only modification needed is to include one of three constants that customize the hardware and motor parameter settings.

The software (written in C language) specifications are listed in Table 2-1. A useful feature of the software is serial communication with PC master software protocol via RS232. The PC master software is PC computer software which allows reading and setting of all the system variables, and can also run html script pages to control the application from the PC. Another feature of the BLDC control software, is on-line parameter modification with PC master software, which can be used for software parameter tuning to a customer motor.

Table 2-1. Software Specifications

Control Algorithm	3-phase trapezoidal BLDC motor control star or delta! connected
	Sensorless, with back-EMF zero crossing commutation control
	Speed closed loop control
	Motoring mode
Target Processor	MC68HC908MR32
Language	C-language with some arithmetical functions in assembler
Compiler	Metrowerks ANSI-C/cC++ Compiler for HC08
Application Control	Manual interface (start/stop switch, speed potentiometer control, LED indication)
	PC master software (remote) interface (via RS232 using PC computer)
MCU Oscillator Frequency	4 MHz (with default software setting)
MCU Bus Frequency	8 MHz (with default software setting)
Minimal BLDC Motor Commutation Period (Without PC Master Software Communication)	333 µs (with default software setting and COEF_HLFCMT = 0.450)

Table 2-1. Software Specifications

Minimal BLDC Motor Commutation Period (with PC Master Software Control)	520 µs (with default software setting and COEF_HLFCMT = 0.450)
Targeted Hardware	Software is prepared to run on three optional board and motor hardware sets: • High-voltage hardware set at variable line voltage 115–230 Vac (software customizing file const_cust_hv.h) • Low-voltage evaluation motor hardware set (software customizing file const_cust_evmm.h) • Low-voltage hardware set (software customizing file const_cust_lv.h)
Software Configuration and Parameters Setting	Configuration to one of the three required hardware sets is provided by inclusion of dedicated software customizing files. The software pack contains the files const_cust_hv.h, const_cust_lv.h, and const_cust_evm.h with predefined parameter settings for running on one of the optional board and motor hardware sets. The required hardware must be selected in code_fun.c file by one of these files #include.
	Where software is configuration for different customer motors, the software configuration for any motor is provided in the dedicated customizing file, according to the hardware board used.
	PWM frequency 15.626 kHz with default software setting, possibly changeable in const.h file

2.3.2 Hardware and Drive Specifications

The other system specifications are determined by hardware boards and motor characteristics. The boards and their connections are shown in **Section 4. Hardware Design**. and **Section 6. User Guide**, **6.4.1 Hardware Configuration**. The hardware set specifications are discussed in the following subsections.

2.3.2.1 High-Voltage Hardware Set Specification

This hardware set is dedicated for medium power (phase current < 2.93 A) and main voltage. The specifications for a high-voltage hardware

and motor set are listed in **Table 2-2**. The hardware operates on both 230 Vac and 115 Vac mains.

Table 2-2. High Voltage Hardware Set Specifications

Hardware Boards Characteristics	Input voltage:	230 Vac or 115 Vac
	Maximum dc-bus voltage:	407 V
	Maximal output current:	2.93A
Motor -Brake Set	Manufactured	EM Brno, Czech Republic
Motor Characteristics	Motor type:	EM Brno SM40V 3 phase, star connected BLDC motor,
	Pole-Number:	6
	Speed range:	2500 rpm (at 310 V)
	Maximum electrical power:	150 W
	Phase voltage:	3*220 V
	Phase current:	0.55 A
Brake Characteristics	Brake Type:	SG40N 3-Phase BLDC Motor
	Nominal Voltage:	3 x 27 V
	Nominal Current:	2.6 A
	Pole-Number:	6
	Nominal Speed:	1500 rpm
Drive Characteristics	Speed range:	< 2500 rpm (determined by motor used)
	Maximum dc-bus voltage:	380 V
	Optoisolation:	Required
	Protection:	Over-current, over-voltage, and under-voltage fault protection
Load Characteristic	Type:	Varying

2.3.2.2 Low-Voltage Evaluation Hardware Set Specification

This hardware set is dedicated for 12 V voltage and very low power (phase current < 4 A). The specifications for a low-voltage evaluation hardware and motor set are listed in **Table 2-3**. It is targeted first of all to software evaluation with small motors.

Table 2-3. Low Voltage Evaluation Hardware Set Specifications

Hardware Boards Characteristics	Input voltage:	12 Vdc
	Maximum dc-bus voltage:	16.0 V
	Maximal output current:	4.0 A
Motor Characteristics	Motor type:	4 poles, three phase, star connected, BLDC motor
	Speed range:	< 5000 rpm (at 60 V)
	Maximal line voltage:	60 V
	Phase current:	2 A
	Output torque:	0.140 Nm (at 2 A)
Drive Characteristics	Speed range:	< 1400 rpm
	Input voltage:	12 Vdc
	Maximum dc-bus voltage:	15.8 V
	Protection:	Over-current, over-voltage, and under-voltage fault protection
Load Characteristic	Type:	Varying

2.3.2.3 Low-Voltage Hardware Set Specification

This hardware set is dedicated for medium power (phase current < 50 A) and automotive voltage. The specifications for a low-voltage hardware and motor set are listed in **Table 2-4**. The hardware power stage board is dedicated for 12 V, but can be simply configured to a 42 V supply (described in documentation for the ECLOVACBLDC board). The supplied motor is targeted for 12 V.

Table 2-4. Low Voltage Hardware Set Specifications

Hardware Boards Characteristics	Input voltage:	12 Vdc or 42 V
	Maximum dc-bus voltage:	16.0 V or 55.0 V
	Maximal output current:	50.0 A
Motor -Brake Set	Manufactured	EM Brno, Czech Republic
Motor Characteristics	Motor type:	EM Brno SM40N 3 phase, star connected BLDC motor,
	Pole-Number:	6
	Speed range:	3000 rpm (at 12 V)
	Maximum electrical power:	150 W
	Phase voltage:	3*6.5 V
	Phase current:	17 A
Brake Characteristics	Brake Type:	SG40N 3-Phase BLDC Motor
	Nominal Voltage:	3 x 27 V
	Nominal Current:	2.6 A
	Pole-Number:	6
	Nominal Speed:	1500 rpm
Drive Characteristics	Speed range:	< 2500 rpm
	Input voltage:	12 Vdc
	Maximum dc-bus voltage:	15.8 V
	Protection:	Over-current, over-voltage, and under-voltage fault protection
Load Characteristic	Type:	Varying

Section 3. BLDC Motor Control

3.1 Contents

3.2 Brushless DC Motor Control Theory

3.2.1 BLDC Motor Targeted by This Application

The brushless dc motor (BLDC motor) is also referred to as an electronically commutated motor. There are no brushes on the rotor, and commutation is performed electronically at certain rotor positions. The stator magnetic circuit is usually made from magnetic steel sheets. Stator phase windings are inserted in the slots (distributed winding) as shown in **Figure 3-1**, or it can be wound as one coil on the magnetic pole. Magnetization of the permanent magnets and their displacement on the rotor are chosen in such a way that the back-EMF (the voltage induced into the stator winding due to rotor movement) shape is trapezoidal. This allows a rectangular shaped 3-phase voltage system (see **Figure 3-2**) to be used to create a rotational field with low torque ripples.

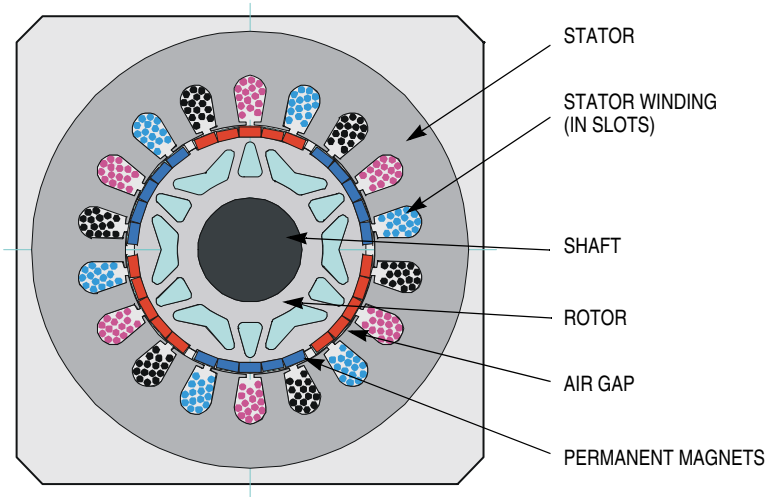

Figure 3-1. BLDC Motor Cross Section

The motor can have more than just one pole-pair per phase. This defines the ratio between the electrical revolution and the mechanical revolution. The BLDC motor shown has three pole-pairs per phase, which represent three electrical revolutions per one mechanical revolution.

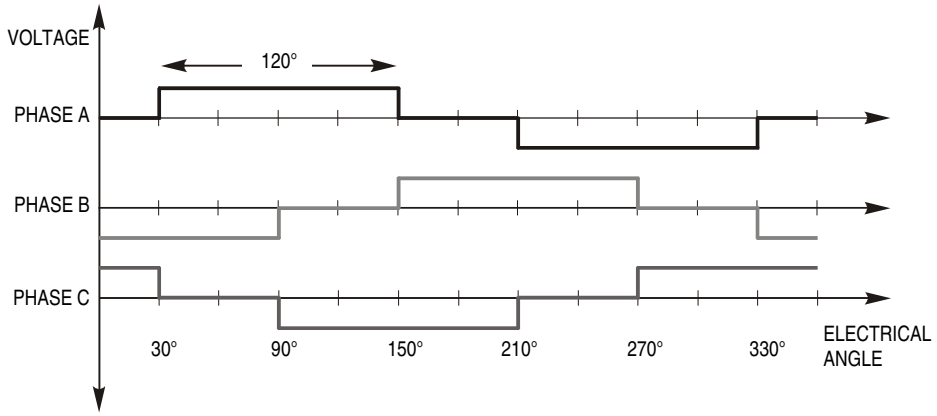

Figure 3-2. 3-Phase Voltage System

The easy to create rectangular shape of applied voltage ensures the simplicity of control and drive. But, the rotor position must be known at certain angles in order to align the applied voltage with the back-EMF (voltage induced due to movement of the PM). The alignment between

back-EMF and commutation events is very important. In this condition, the motor behaves as a dc motor and runs at the best working point. Thus, simplicity of control and good performance make this motor a natural choice for low-cost and high-efficiency applications.

Figure 3-3 shows a number of waveforms:

- Magnetic flux linkage
- Phase back-EMF voltage
- Phase-to-phase back-EMF voltage

Magnetic flux linkage can be measured. However, in this case it was calculated by integrating the phase back-EMF voltage (which was measured on the non-fed motor terminals of the BLDC motor). As can be seen, the shape of the back-EMF is approximately trapezoidal and the amplitude is a function of the actual speed. During speed reversal, the amplitude changes its sign and the phase sequence changes.

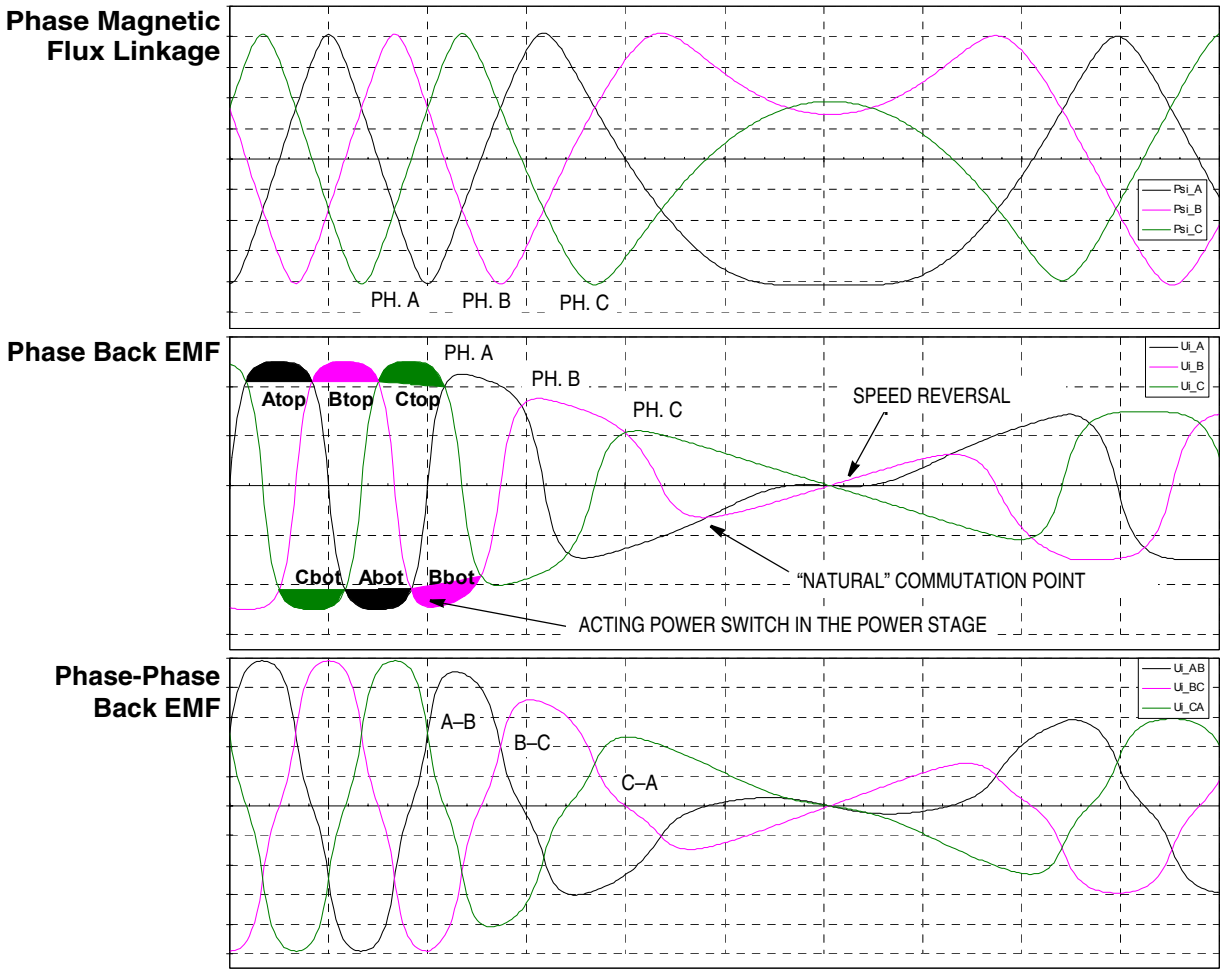

Figure 3-3. BLDC Motor Back EMF and Magnetic Flux

The filled areas in the tops of the phase back-EMF voltage waveforms indicate the intervals where the particular phase power stage commutations occur. The power switches are cyclically commutated through the six steps; therefore, this technique is sometimes called six step commutation control. The crossing points of the phase back-EMF voltages represent the natural commutation points. In normal operation the commutation is performed here. Some control techniques advance the commutation by a defined angle in order to control the drive above the pulse-width modulator (PWM) voltage control.

3.2.2 3-Phase BLDC Power Stage

The voltage for 3-phase BLDC motor is provided by a 3-phase power stage controlled by digital signals. Its topology is the one as for the AC induction motor (refer to **Figure 3-5**). The power stage is usually controlled by a dedicated microcontroller with on-chip PWM module.

3.2.3 Why Sensorless Control?

As explained in the previous section, rotor position must be known in order to drive a brushless dc motor. If any sensors are used to detect rotor position, sensed information must be transferred to a control unit (see **Figure 3-4**). Therefore, additional connections to the motor are necessary. This may not be acceptable for some applications (see **1.3 Benefits of the Solution**).

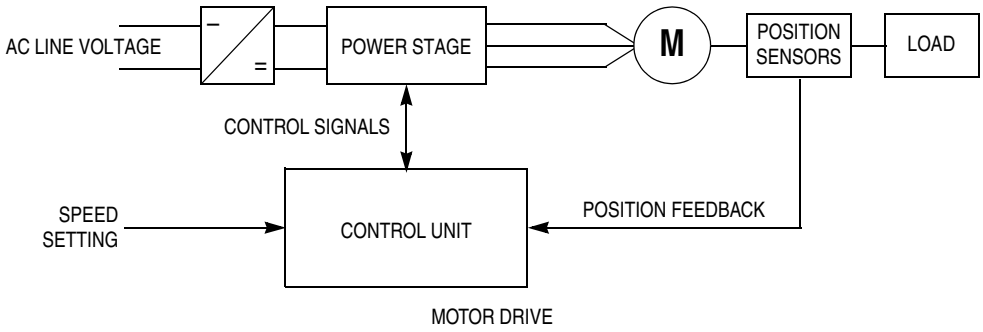

Figure 3-4. Classical System

3.2.4 Power Stage — Motor System Model

In order to explain and simulate the idea of back-EMF sensing techniques a simplified mathematical model based on the basic circuit topology has been created. See **Figure 3-5**.

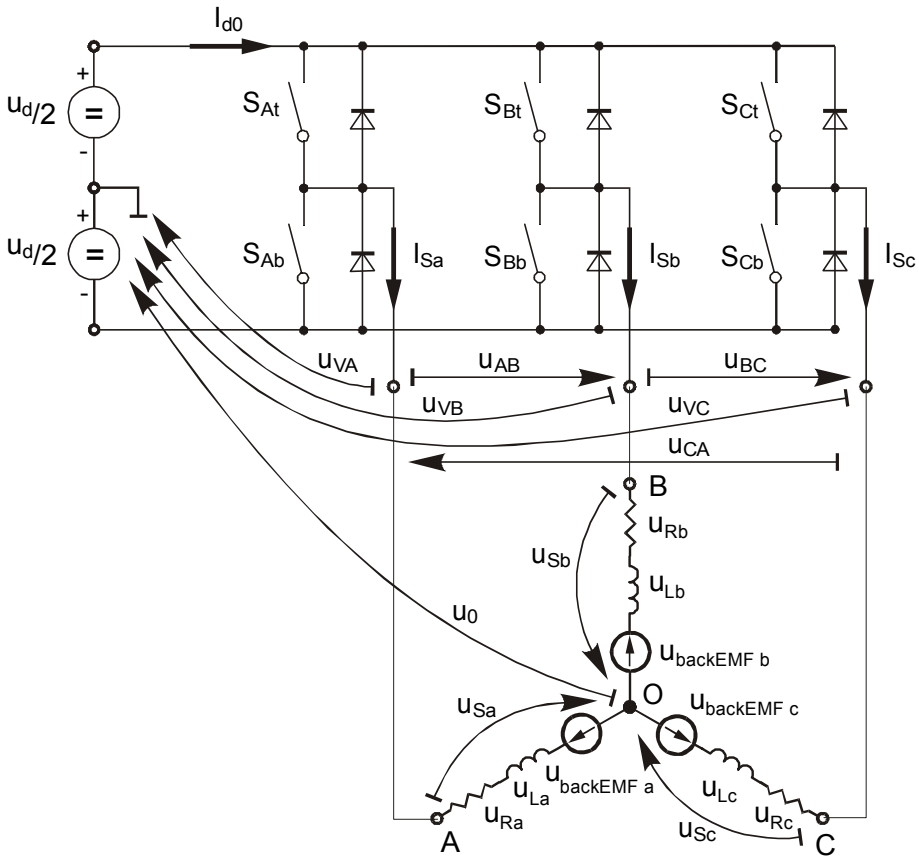

Figure 3-5. Power Stage — Motor Topology

The second goal of the model is to find how the motor characteristics depend on the switching angle. The **switching angle** is the angular difference between a real switching event and an ideal one (at the point where the **phase-to-phase** back-EMF crosses zero).

The motor-drive model consists of a normal 3-phase power stage plus a brushless dc motor. Power for the system is provided by a voltage source (U_d). Six semiconductor switches ($S_{A/B/C\ t/b}$), controlled elsewhere, allow the rectangular voltage waveforms (see **Figure 3-2**) to be applied. The semiconductor switches and diodes are simulated as ideal devices. The natural voltage level of the whole model is put at one half of the dc-bus voltage. This simplifies the mathematical expressions.

3.2.4.1 Stator Winding Equations

The BLDC motor is usually very symmetrical. All phase resistances, phase and mutual inductances, flux-linkages can be thought of as equal to, or as a function of the position θ with a 120° displacement.

The electrical BLDC motor model then consists of a set of the following stator voltage equations **(EQ 3-1.)**.

$$\begin{bmatrix} u_{Sa} \\ u_{Sb} \\ u_{Sc} \end{bmatrix} = R_S \begin{bmatrix} i_{Sa} \\ i_{Sb} \\ i_{Sc} \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \Psi_{Sa} \\ \Psi_{Sb} \\ \Psi_{Sc} \end{bmatrix} \quad \textbf{(EQ 3-1.)}$$

The task of this section is to explain the background of the back-EMF sensing and to demonstrate how the zero crossing events can be detected. Parasitic effects that negatively influence the back-EMF detection are discussed and their nature analyzed.

3.2.4.2 Indirect Back EMF Sensing

Let us assume a usual situation, where the BLDC motor is driven in six-step commutation mode using PWM technique, where both top and bottom switches in the diagonal are controlled using the same signal (so called “hard switching PWM” technique). The motor phases A and B are powered, and phase C is free, having no current. So the phase C can be used to sense the back-EMF voltage. This is described by the following conditions:

$$\begin{gathered} S_{Ab}, S_{Bt} \leftarrow \mathrm{PWM} \\ u_{VA} = \mp\frac{1}{2}u_d, u_{VB} = \pm\frac{1}{2}u_d \\ i_{Sa} = -i_{Sb} = i, di_{Sa} = -di_{Sb} = di \\ i_{Sc} = 0, di_{Sc} = 0 \\ u_{backEMF\ a} + u_{backEMF\ b} + u_{backEMF\ c} = 0 \end{gathered} \quad \textbf{(EQ 3-2.)}$$

The branch voltage u_{Vc} can be calculated using the above conditions,

$$u_{VC} = u_{Sc} - \frac{1}{3}\left[\sum_{x=a}^{c} u_{backEMF\ x} + (L_{ac} - L_{bc})\frac{di}{dt} - u_{VC}\right] \quad \textbf{(EQ 3-3.)}$$

After evaluation the expression of the branch voltage u_{Vc} is as follows:

$$u_{VC} = \frac{3}{2}u_{backEMF\ c} - \frac{1}{2}(L_{ac} - L_{bc})\frac{di}{dt} \quad \textbf{(EQ 3-4.)}$$

The same expressions can also be found for phase A and B:

$$u_{VA} = \frac{3}{2}u_{backEMF\ a} - \frac{1}{2}(L_{ba} - L_{ca})\frac{di}{dt} \quad \textbf{(EQ 3-5.)}$$

$$u_{VB} = \frac{3}{2}u_{backEMF\ b} - \frac{1}{2}(L_{cb} - L_{ab})\frac{di}{dt} \quad \textbf{(EQ 3-6.)}$$

The first member in the equation **(EQ 3-6.)** demonstrates the possibility to indirectly sense the back-EMF between the free (not powered) phase terminal and the zero point, defined at half of the dc-bus voltage (see Figure 3-5.). Simple comparison of these two levels can provide the required zero crossing detection.

As shown in **Figure 3-5**, the branch voltage of phase B can be sensed between the power stage output B and the zero voltage level. Thus, back-EMF voltage is obtained and the zero crossing can be recognized.

When $L_{cb} = L_{ab}$, this general expressions can also be found:

$$u_{Vx} = \frac{3}{2}u_{backEMFx} \text{where } x = A,B,C \quad \textbf{(EQ 3-7.)}$$

There are two necessary conditions which must be met:

- Top and bottom switches (in diagonal) have to be driven with the same PWM signal
- No current goes through the non-fed phase that is used to sense the back-EMF

Figure 3-6 shows branch and motor phase winding voltages during a 0–360° electrical interval. Shaded rectangles designate the validity of the equation **(EQ 3-7.)**. In other words, the back-EMF voltage can be sensed during designated intervals.

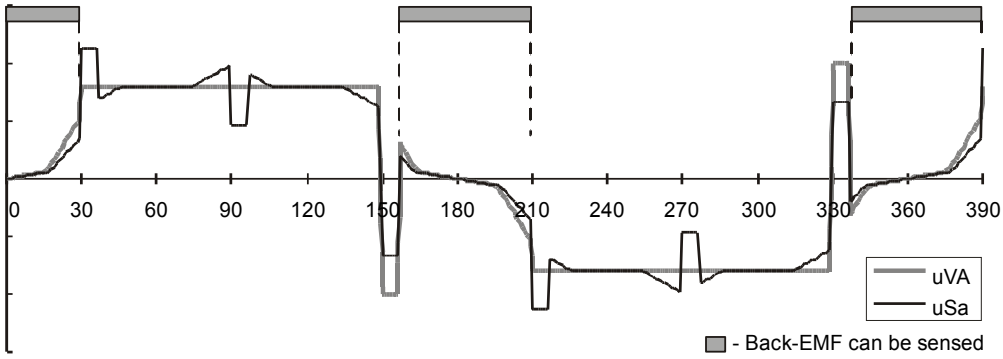

Figure 3-6. Phase Voltage Waveform

However simple this solution looks, in reality it is more difficult, because the sensed “branch” voltage also contains some ripples.

3.2.4.3 Effect of Mutual Inductance

As shown in previous equations **(EQ 3-4.)** through **(EQ 3-6.)**, the mutual inductances play an important role here. The difference of the mutual inductances between the coils which carry the phase current, and the coil used for back-EMF sensing, causes the PWM pulses to be superimposed onto the detected back-EMF voltage. In fact, it is produced by the high rate of change of phase current, transferred to the free phase through the coupling of the mutual inductance.

Figure 3-7 shows the real measured “branch” voltage. The red curves highlight the effect of the difference in the mutual inductances. This difference is not constant.

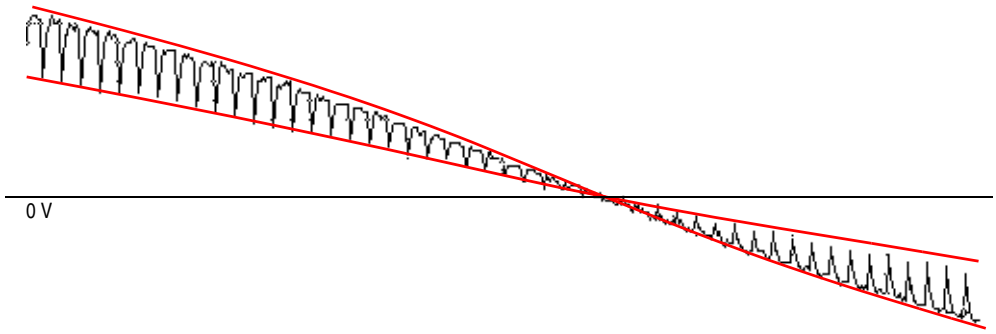

Figure 3-7. Mutual Inductance Effect

Due to the construction of the BLDC motor, both mutual inductances vary. They are equal at the position that corresponds to the back-EMF zero crossing detection.

The branch waveform detail is shown in **Figure 3-8**. Channel 1 in **Figure 3-8** shows the disturbed “branch” voltage. The superimposed ripples clearly match the width of the PWM pulses, and thus prove the conclusions from the theoretical analysis.

The effect of the mutual inductance corresponds well in observations carried out on the five different BLDC motors. These observations were made during the development of the sensorless technique.

NOTE: *The BLDC motor with stator windings distributed in the slots has technically higher mutual inductances than other types. Therefore, this effect is more significant. On the other hand the BLDC motor with windings wounded on separate poles, shows minor presence of the effect of mutual inductance.*

CAUTION: *However noticeable this effect, it does not degrade the back-EMF zero crossing detection because it is cancelled at the zero crossing point. Simple additional filtering helps to reduce ripples further.*

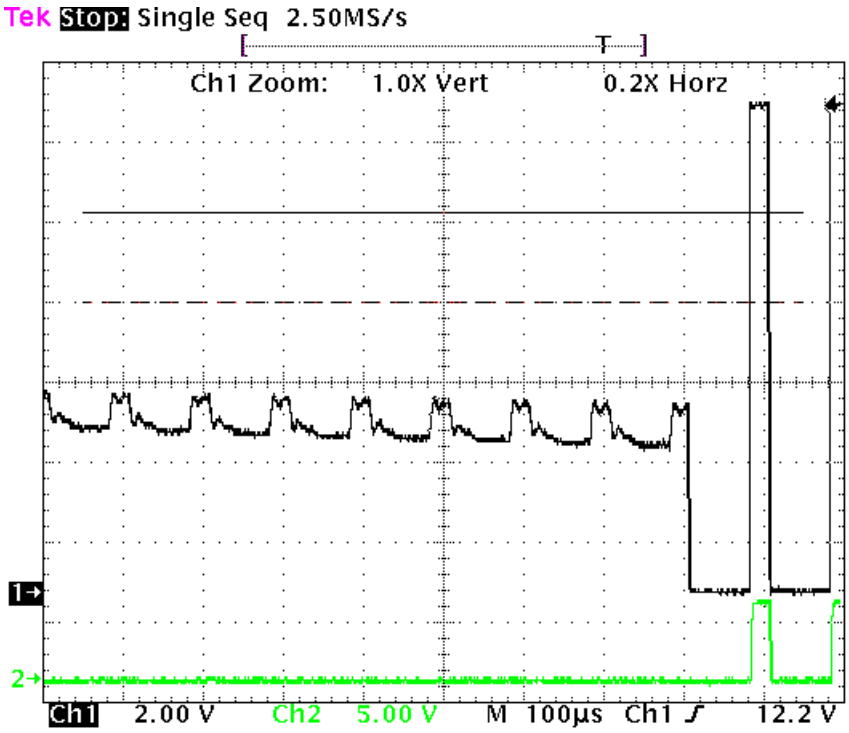

Figure 3-8. Detail of Mutual Inductance Effect

3.2.4.4 Effect of Mutual Phase Capacitance

The negative effect of mutual inductance is not the only one to disturb the back-EMF sensing. So far, the mutual capacitance of the motor phase windings was neglected in the motor model, since it affects neither the phase currents nor the generated torque. Usually the mutual capacitance is very small. Its influence is only significant during PWM switching, when the system experiences very high du/dt.

The effect of the mutual capacitance can be studied using the model shown in **Figure 3-9**.

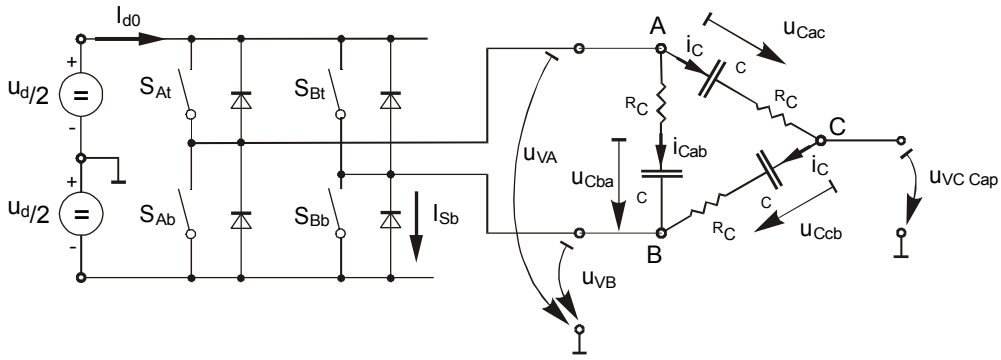

Figure 3-9. Mutual Capacitance Model

Let us focus on the situation when the motor phase A is switched from negative dc-bus rail to positive, and the phase B is switched from positive to negative. This is described by these conditions **(EQ 3-8.)**:

$$S_{Ab}, S_{Bt} \leftarrow PWM$$
$$u_{VA} = -\frac{1}{2}u_d \rightarrow \frac{1}{2}u_d, u_{VB} = \frac{1}{2}u_d \rightarrow -\frac{1}{2}u_d \qquad \textbf{(EQ 3-8.)}$$
$$i_{Cac} = i_{Ccb} = i_C$$

The voltage that disturbs the back-EMF sensing, utilizing the free (not powered) motor phase C, can be calculated based the equation:

$$u_{VC\ Cap} = \frac{1}{2}(u_{Ccb} + u_{Cac} + 2R_C) - (u_{Ccb} + R_C) = \frac{1}{2}(u_{Cac} - u_{Ccb}) \qquad \textbf{(EQ 3-9.)}$$

The final expression for disturbing voltage can be found as follows:

$$u_{VC\ Cap} = \frac{1}{2}\left(\frac{1}{C_{ac}} - \frac{1}{C_{cb}}\right)\int i_C\,dt = \frac{1}{2}\left(\frac{C_{cb} - C_{ac}}{C_{cb} \cdot C_{ac}}\right)\int i_C\,dt \qquad \textbf{(EQ 3-10.)}$$

NOTE: ***(EQ 3-10.)*** *expresses the fact that only the unbalance of the mutual capacitance (not the capacitance itself) disturbs the back-EMF sensing. When both capacities are equal (they are balanced), the disturbances disappear. This is demonstrated in* ***Figure 3-10*** *and* ***Figure 3-11****.*

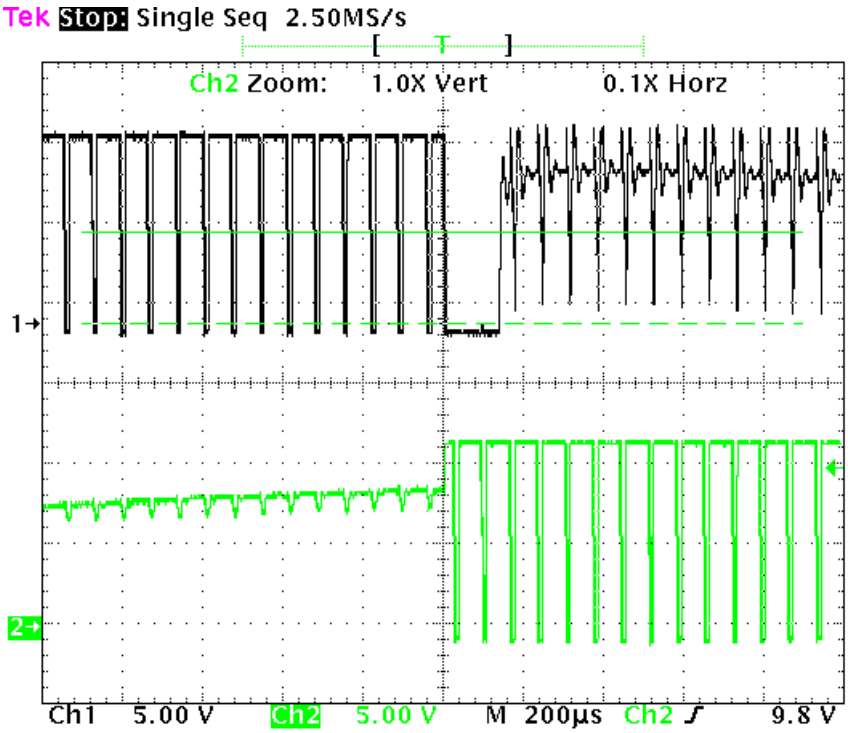

Figure 3-10. Distributed Back-EMF by Unbalanced Capacity Coupling

Channel 1 in **Figure 3-11** shows the disturbed “branch” voltage, while the other phase (channel 2) is not affected because it faces balanced mutual capacitance. The unbalance was purposely made by adding a small capacitor on the motor terminals, in order to better demonstrate the effect. After the unbalance was removed the “branch” voltage is clean, without any spikes.

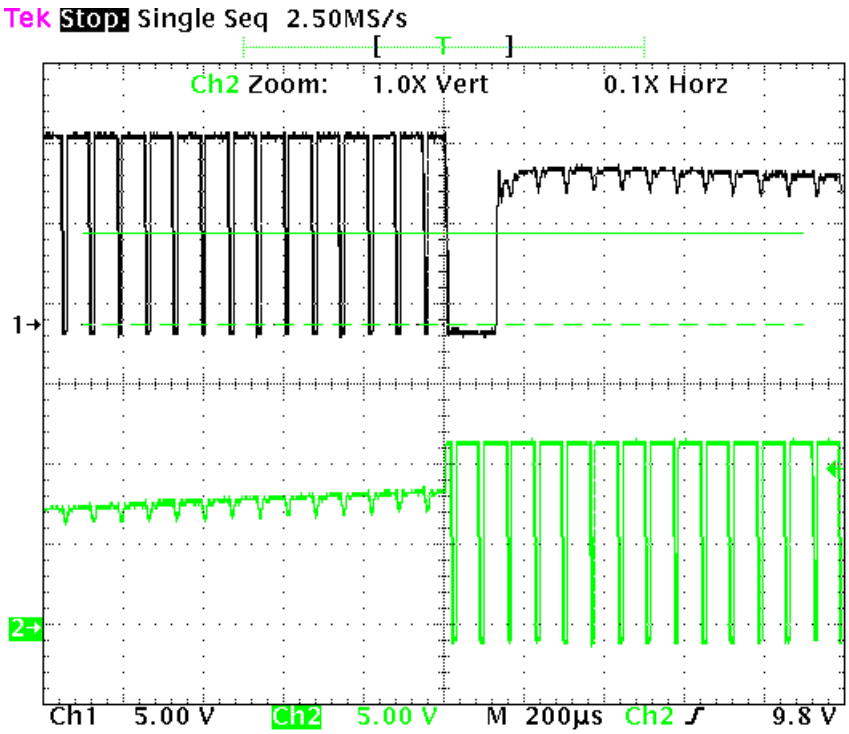

Figure 3-11. Balanced Capacity Coupling

NOTE: *The configuration of the phase windings end-turns has significant impact; therefore, it needs to be properly managed to preserve the balance in the mutual capacity. This is important, especially for prototype motors that are usually hand-wound.*

CAUTION: *Failing to maintain balance in the mutual capacitance can easily disqualify such a motor from using sensorless techniques based on the back-EMF sensing. Usually, the BLDC motors with windings wound on separate poles show minor presence of the mutual capacitance. Thus, the disturbance is also insignificant.*

3.2.5 Back-EMF Sensing Circuit

An example of the possible implementation of the back-EMF sensing circuit is shown in **Figure 3-12**.

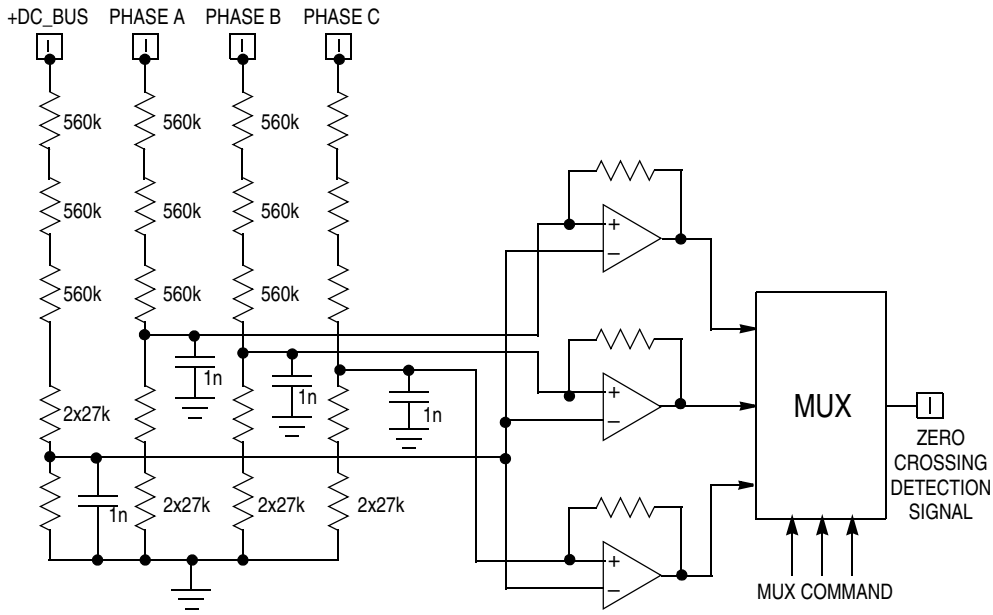

Figure 3-12. Back-EMF Sensing Circuit Diagram

As explained in the theoretical part of this designer reference manual, the phase zero crossing event can be detected at the moment when the branch voltage (of a free phase) crosses the half dc-bus voltage level. The resistor network is used to step down sensed voltages down to a 0–15 V voltage level. The comparators sense the zero voltage difference in the input signal. The multiple resistors reduce the voltage across each resistor component to an acceptable level. A simple RC filter prevents the comparators from being disturbed by high voltage spikes produced by IGBT switching. The multiplexer (MUX) selects the phase comparator output, which corresponds to the current commutation stage. This zero crossing detection signal is transferred to the timer input pin.

The comparator control and zero crossing signals plus the voltage waveforms are shown in **Figure 3-13**.

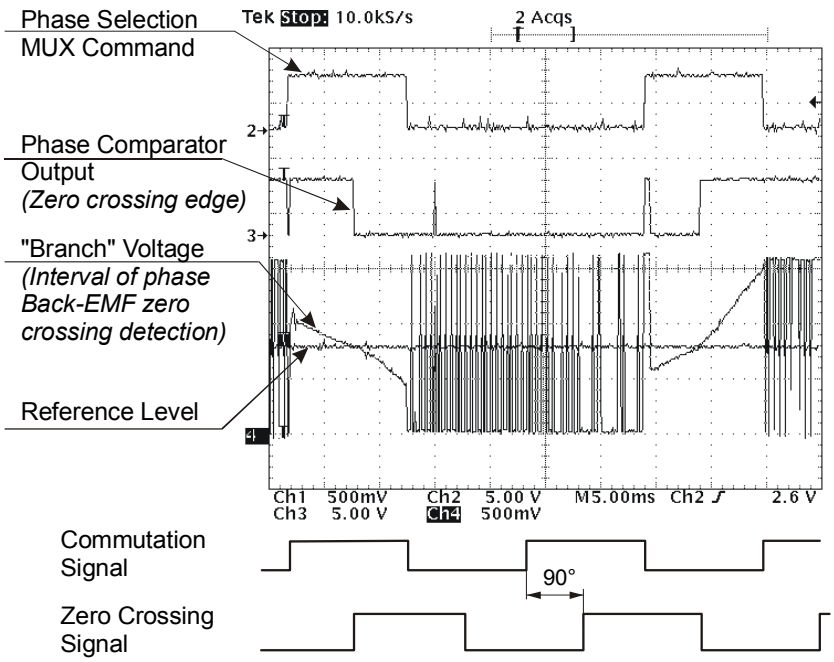

Figure 3-13. The Zero Crossing Detection

3.3 Used Control Technique

3.3.1 Sensorless Commutation Control

This section concentrates on sensorless BLDC motor commutation with back-EMF zero crossing technique.

In order to start and run the BLDC motor, the control algorithm has to go through the following states:

- **Alignment**
- **Starting (Back-EMF Acquisition)**
- **Running**

Figure 3-14 shows the transitions between the states. First the rotor is aligned to a known position; then the rotation is started without the position feedback. When the rotor moves, back-EMF is acquired so the

position is known, and can be used to calculate the speed and processing of the commutation in the running state.

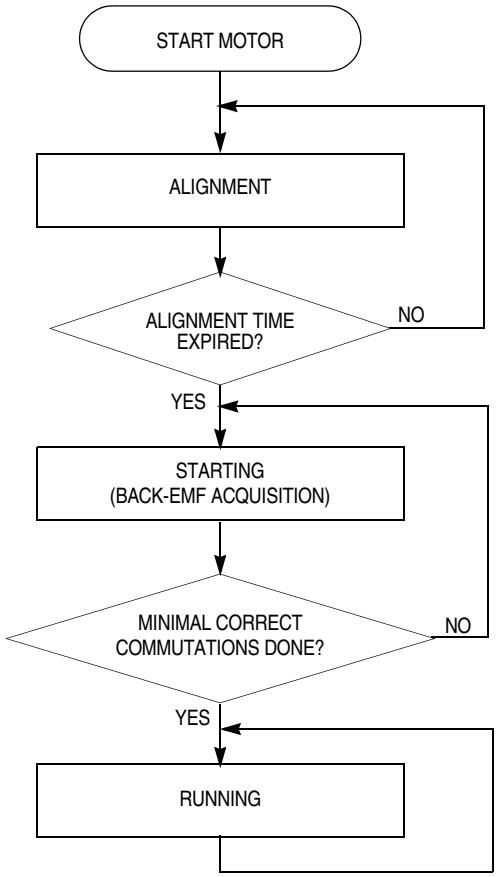

Figure 3-14. Commutation Control Stages

3.3.1.1 Alignment

Before the motor starts, there is a short time (depending on the motor's electrical time constant) when the rotor position is stabilized by applying PWM signals to only two motor phases (no commutation). The current controller keeps current within predefined limits. This state is necessary in order to create a high start-up torque. When the preset time-out expires then this state is finished.

The current controller subroutine, with PI regulator, is called to control dc-bus current. It sets the correct PWM ratio for the required current. The current PI controller works with constant execution (sampling) period. This period should be a multiple of the PWM period, in order to synchronize the current measurement with PWM:

Current controller period = n/PWM frequency

The BLDC motor rotor position with flux vectors during alignment is shown in **Figure 3-15**.

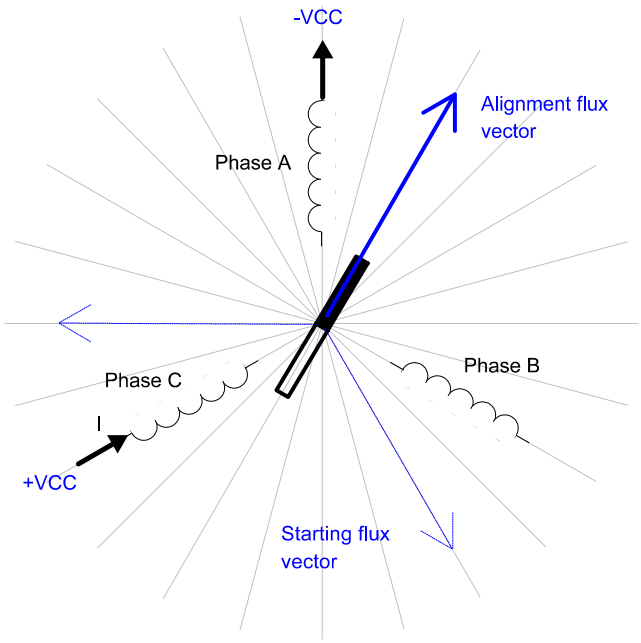

Figure 3-15. Alignment

3.3.1.2 Running

The commutation process is a series of states which assure:

- The back-EMF zero crossing is successfully captured
- The new commutation time is calculated
- The commutation is performed

The following processes need to be provided:

- BLDC motor commutation service
- Back-EMF zero crossing moment capture service
- Calculation of commutation time
- Interactions between these commutation processes

From diagrams an overview of how the commutation works can be understood. After commuting the motor phases, there is a time interval (Per_Toff[n]) when the shape of back-EMF must stabilized (after the commutation the fly-back diodes are conducting the decaying phase current; therefore, sensing of the back-EMF is not possible). Then the new commutation time (T2[n]) is preset. The new commutation will be performed at this time if the back-EMF zero crossing is not captured. If the back-EMF zero crossing is captured before the preset commutation time expires, then the exact calculation of the commutation time (T2*[n]) is made, based on the captured zero crossing time (T_ZCros[n]). The new commutation is performed at this new time.

If for any reason the back-EMF feedback is lost within one commutation period, corrective actions are taken in order to return to the regular states.

The flowchart explaining the principle of BLDC commutation control with back-EMF zero crossing sensing is shown in **Figure 3-16**.

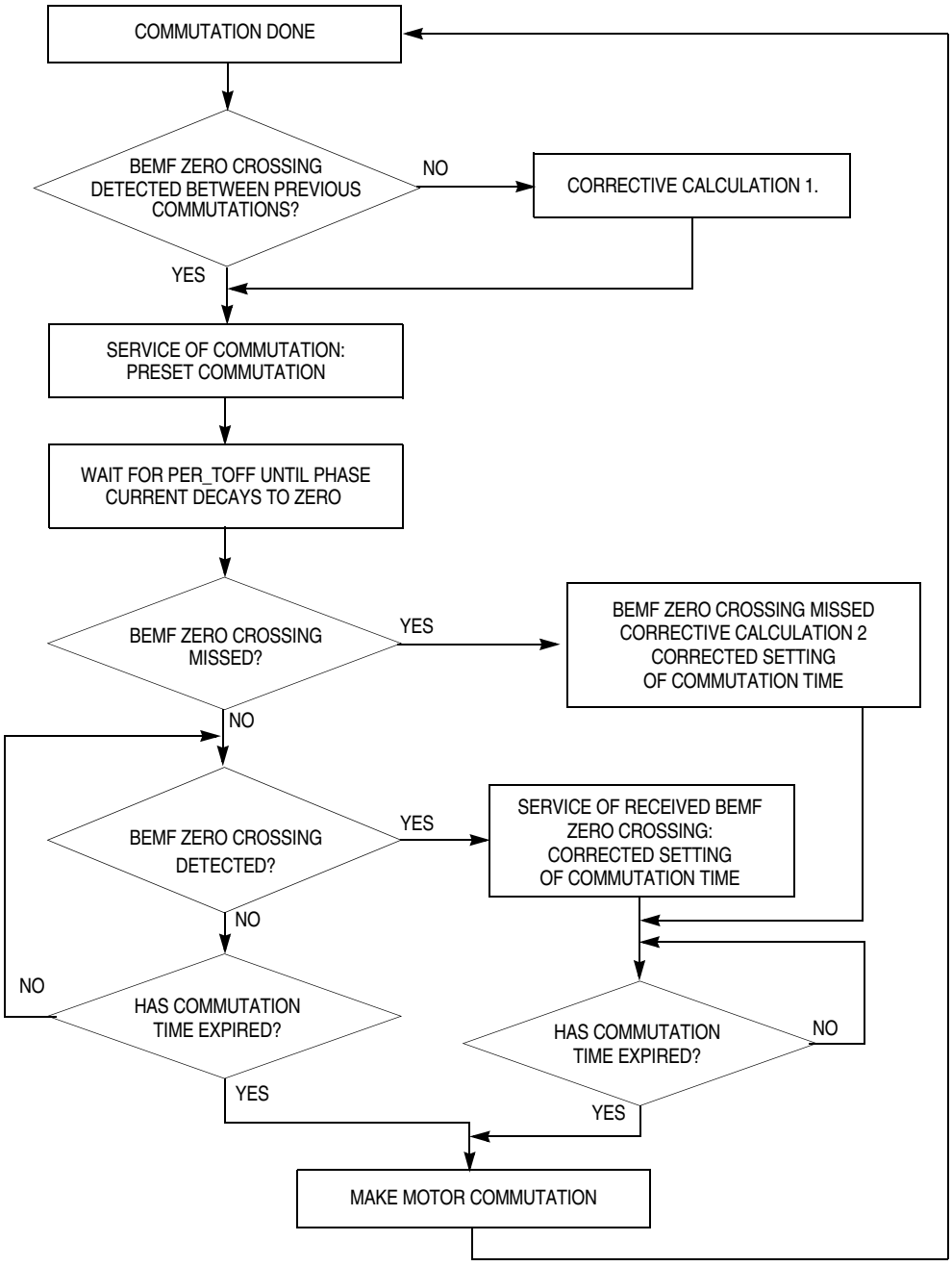

Figure 3-16. BLDC Commutation with Back-EMF Zero Crossing Sensing Flowchart

3.3.1.3 Running — Commutation Time Calculation

Commutation time calculation is shown in **Figure 3-17**.

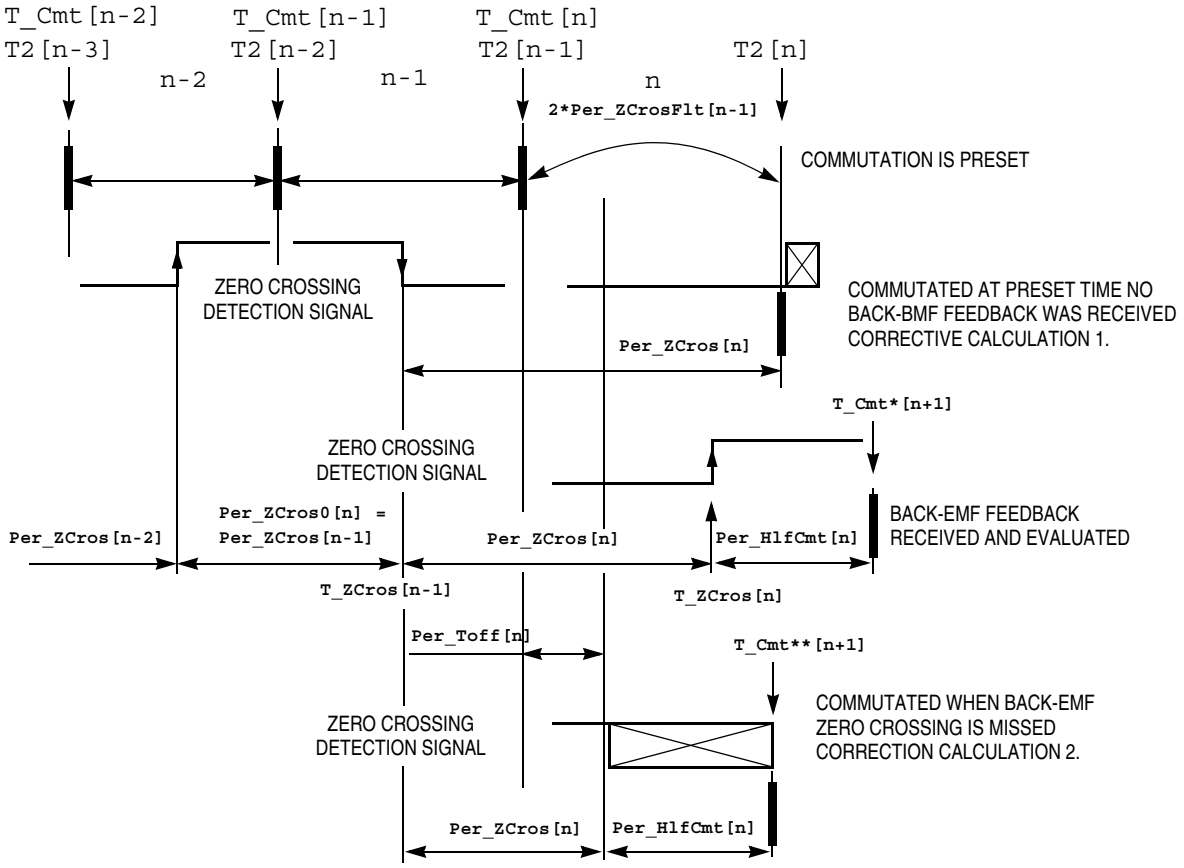

Figure 3-17. BLDC Commutation Time with Zero Crossing Sensing

The following calculations are made to calculate the commutation time (`T2[n]`) during the **Running state**:

- **Service of commutation** — The commutation time (`T2`[n]) is predicted:

```
T2[n] = T_Cmt[n] + 2*Per_ZCrosFlt[n-1]
If 2*Per_ZCrosFlt>Per_Cmt_Max
then result is limited at Per_Cmt_Max
```

- **Service of received back-EMF zero crossing** — The commutation time (`T2`*[n]) is evaluated from the captured back-EMF zero crossing time (T_ZCros[n]):

```
Per_ZCros[n] = T_ZCros[n] - T_ZCros[n-1] = T_ZCros[n] - T_ZCros0
Per_ZCrosFlt[n] = (1/2*Per_ZCros[n]+1/2*Per_ZCros0)
HlfCmt[n] = 1/2*Per_ZCrosFlt[n]- Advance_angle =
          = 1/2*Per_ZCrosFlt[n]- C_CMT_ADVANCE*Per_ZCrosFlt[n]=
            Coef_HlfCmt*Per_ZCrosFlt[n]
```

```
        The best commutation was get with Advance_angle:
        60Deg*1/8 = 7.5Deg
        which means Coef_HlfCmt = 0.375 at Running state
        with default s/w setting
Per_Toff[n+1] = Per_ZCrosFlt*Coef_Toff and Per_Dis minimum
        Coef_Toff = 0.375 at Running state, Per_Dis = 150
        with default s/w setting
Per_ZCros0 <-- Per_ZCros[n]
T_ZCros0 <-- T_ZCros[n]
T2*[n] = T_ZCros[n] + HlfCmt[n]
```

- If no back-EMF zero crossing was captured during preset commutation period (T2P[n] then **Corrective Calculation 1.** is made:

```
T_ZCros[n] <-- CmtT[n+1]
Per_ZCros[n] = T_ZCros[n] - T_ZCros[n-1] = T_ZCros[n] - T_ZCros0
Per_ZCrosFlt[n] = (1/2*Per_ZCros[n]+1/2*Per_ZCros0)
HlfCmt[n] = 1/2*Per_ZCrosFlt[n]-Advance_angle =
Coef_HlfCmt*Per_ZCrosFlt[n]
        The best commutation was get with Advance_angle:
        60Deg*1/8 = 7.5Deg
        which means Coef_HlfCmt = 0.375 at Running state!
Per_Toff[n+1] = Per_ZCrosFlt*Coef_Toff and Per_Dis minimum
Per_ZCros0 <-- Per_ZCros[n]
T_ZCros0 <-- T_ZCros[n]
```

- If back-EMF zero crossing is missed then **Corrective Calculation 2.** is made:

```
T_ZCros[n] <-- CmtT[n]+Toff[n]
Per_ZCros[n] = T_ZCros[n] - T_ZCros[n-1] = T_ZCros[n] - T_ZCros0
Per_ZCrosFlt[n] = (1/2*T_ZCros[n]+1/2*T_ZCros0)
HlfCmt[n] = 1/2*Per_ZCrosFlt[n]-Advance_angle =
Coef_HlfCmt*Per_ZCrosFlt[n]
        The best commutation was get with Advance_angle:
        60Deg*1/8 = 7.5Deg
        which means Coef_HlfCmt = 0.375 at Running state!
Per_ZCros0 <-- Per_ZCros[n]
T_ZCros0 <-- T_ZCros[n]
```

- Where:

```
T_Cmt = time of the last commutation
T2 = Time of the Timer 2 event (for Timer Setting)
T_ZCros = Time of the last zero crossing
T_ZCros0 = Time of the previous zero crossing
Per_Toff = Period of the zero crossing off
Per_ZCros = Period between zero crossings (estimates required
        commutation period)
Per_ZCros0 = Pervious period between zero crossings
Per_ZCrosFlt = Estimated period of commutation filtered
```

```
Per_HlfCmt = Period from zero crossing to commutation (half
commutation)
```

The required commutation timing is provided by setting commutation constants **Coef_HlfCmt, COEF_TOFF**.

3.3.1.4 Starting (Back-EMF Acquisition)

The back-EMF sensing technique enables a sensorless detection of the rotor position; however, the drive must be first started without this feedback. This is due to the fact that the amplitude of the induced voltage is proportional to the motor speed. Hence, the back-EMF cannot be sensed at a very low speed and a special start-up algorithm must be performed.

In order to start the BLDC motor, the adequate torque must be generated. The motor torque is proportional to the multiplication of the stator magnetic flux, the rotor magnetic flux, and the sine of the angle between these magnetic fluxes.

It implies (for BLDC motors) the following:

1. The level of phase current must be high enough.
2. The angle between the stator and rotor magnetic fields must be 90°±30°.

The first condition is satisfied during the alignment state by maintaining dc-bus current at a level sufficient to start the motor. In the starting (back-EMF acquisition) state, the same value of PWM duty cycle is used as the one which has stabilized the dc-bus current during the align state.

The second condition is more difficult to fulfill without any position feedback information. After the alignment state, the stator and the rotor magnetic fields are aligned (0° angle). Therefore, two fast commutations (faster than the rotor can follow) must be applied to create an angular difference in the magnetic fields (see **Figure 3-18**).

The commutation time is defined by the start commutation period (**Per_CmtStart**) . This allows starting the motor such that minimal speed (defined by state when back-EMF can be sensed) and is achieved

during several commutations, while producing the required torque. Until the back-EMF feedback is locked, the commutation process (explained in **Running**) assures that commutations are done in advance, so that successive back-EMF zero crossing events are not missed.

After several successive back-EMF zero crossings:

- Exact commutation time can be calculated
- Commutation process is adjusted
- Control flow continues to the Running state

The BLDC motor is then running with regular feedback and the speed controller can be used to control the motor speed by changing the PWM duty cycle value.

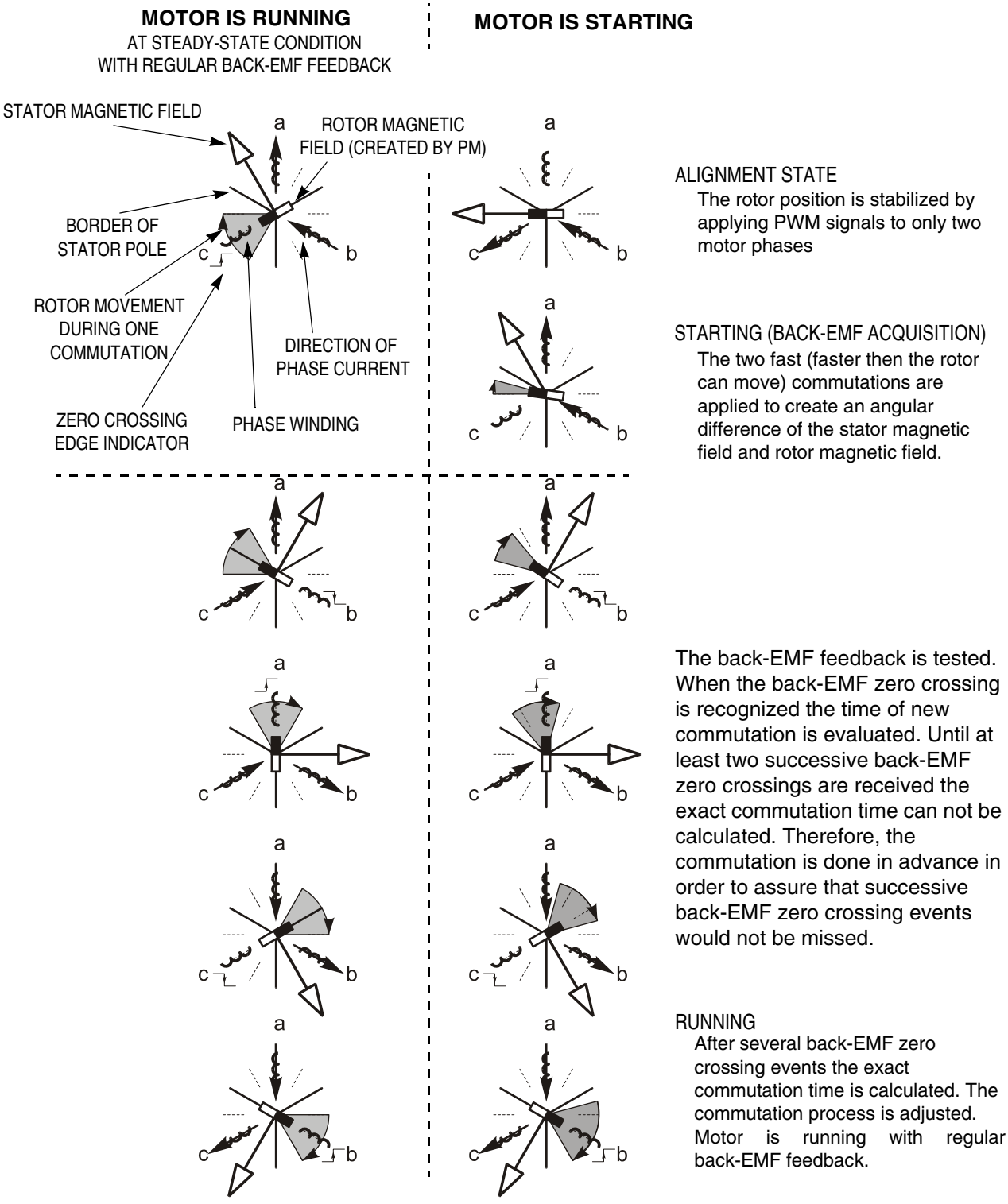

Figure 3-18. Vectors of Magnetic Fields

Figure 3-19 demonstrates the back-EMF during the start up. The amplitude of the back-EMF varies according to the rotor speed. During the starting (back-EMF acquisition) state the commutation is done in advance. In the running state the commutation is done at the right moments.

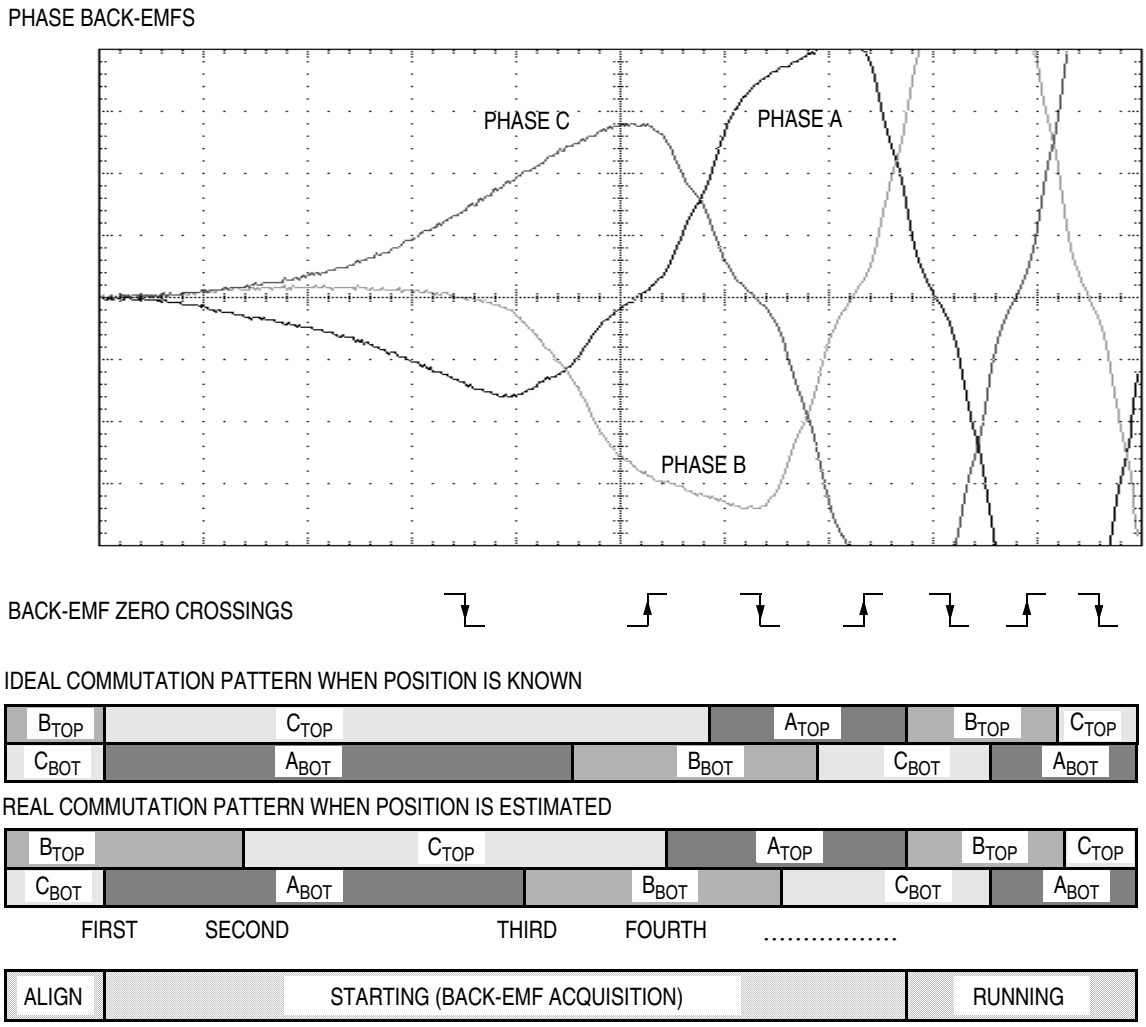

Figure 3-19. Back-EMF at Start Up

Figure 3-20 illustrates the sequence of the commutations during the starting (back-EMF acquisition) state. The commutation times T2[1] and T2[2] are calculated without any influence of back-EMF feedback. The commutation time calculations are explained in the following section.

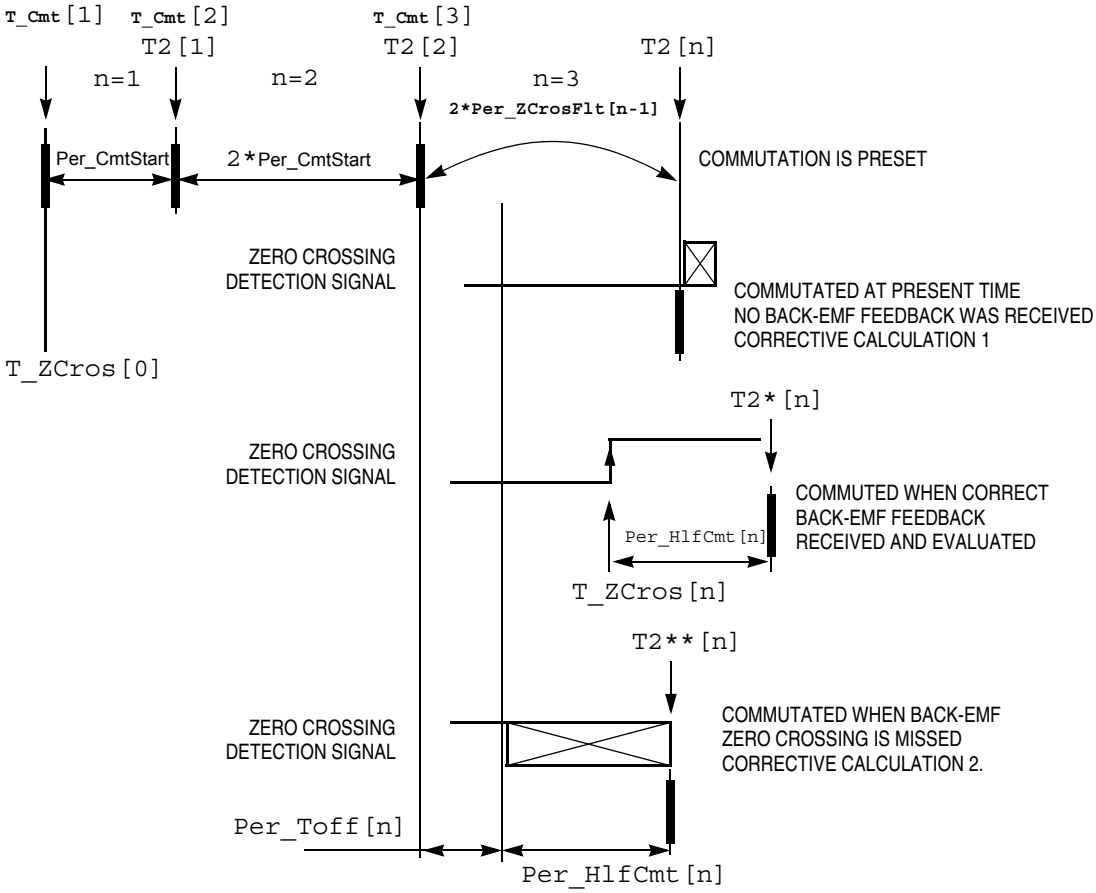

Figure 3-20. Calculation of the Commutation Times During the Starting (Back-EMF Acquisition) State

3.3.1.5 Starting — Commutation Time Calculation

Even the sub-states of the commutation process in the starting (back-EMF acquisition) state remain the same as in the running state. The required commutation timing depends on application state (starting state, running state). So the commutation time calculation is the same as that described in

Running — Commutation Time Calculation, but the following computation coefficients are different:

```
coefficient Coef_HlfCmt = 0.125 with advanced angle
Advance_angle: 60Deg*3/8 = 22.5Deg
       at Starting state!
```

Coef_Toff = 0.5 at Running state, **Per_Dis** = 150 with default s/w setting

3.3.2 Speed Control

The speed close loop control is provided by a well known PI regulator. The required speed is calculated from speed input variable, as explained in **Process Desired Speed Setting**. The actual speed is calculated from the average of two back-EMF zero crossing periods (time intervals), received from the sensorless commutation control block. The speed regulator output is a PWM duty cycle.

The speed controller works with the constant execution (sampling) period **PER_T3_RUN_US**. A detailed explanation is provided in **Processs Speed Control**.

3.4 Application Control

The application can be controlled in two basic modes:

- Manual mode
- PC master software mode

In manual mode, it is controlled by an on-board start/stop switch and speed potentiometer. In PC master mode, it is controlled from a computer using PC master software. In both modes, the individual variables can be observed using the PC master software.

3.4.1 PC Master Software

PC master software was designed to provide the debugging, diagnostic, and demonstration tools for developing algorithms and applications. It consists of components running on PCs and parts running on the target MCU, connected by an RS232 serial port. A small program is resident in the MCU that communicates with the PC master software to parse commands, return status information, and process control information from the PC. The PC master software uses Microsoft[1] Internet Explorer as a user interface on the PC.

3.4.1.1 Communication with PC Master Software Specifications

SCI communication protocol with a default of 9.6 Kbaud, is used for communication as described in *User's Manual for PC Master Software*, Motorola 2000, found on the World Wide Web at:
http://e-www.motorola.com

PC master software controls and senses the status of the application with:

- PC master software — BLDC demonstration suitcase communication commands
- PC master software — BLDC demonstration suitcase communication bytes

After reset, the BLDC control MCU software is in manual mode. In order to control the system from PC master software, it is necessary to set PC master software mode, and then to provide the MCU software control from PC master software via application interface variables.

3.4.1.2 PC Master Software, BLDC Control MCU Software API, Communication Commands

Commands defined for the BLDC control MCU software are listed in **Table 3-1**. The commands are very simple. If the software executes the command, it responds with OK byte 00. If it is unable to execute the command, it responds with failed code 55. The commands "Set PC master software mode", "Set manual mode" can only be executed when the START/STOP switch on the demonstration suitcase is set to STOP and the motor is stopped. Otherwise, a failed response is sent.

Table 3-1. PC Master Software Communication Commands

Command	Command Code	Data Bytes	Demo Suitcase Action	Respons Byte	Response Description
Set PC master software mode	01	None	Setting of PC master software mode	00 55	OK Failed

1. Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries.

Table 3-1. PC Master Software Communication Commands

Command	Command Code	Data Bytes	Demo Suitcase Action	Respons Byte	Response Description
Set manual mode	02	None	Setting of manual mode	00 55	OK Failed

3.4.1.3 PC Master Software, BLDC Control MCU Software API, Communication Variables

The application interface, data variables used for the exchange between the BLDC control MCU software and PC master software, are shown in Table 3-2. These variables are used for status sensing and control. PC master software accesses these bytes directly from their physical memory addresses.

Table 3-2. PC Master Software API Variables

Name	Type	I/O	Representing Range	Description
Sys3	Sys3_Def	I/O	8flags	System variable #3
Motor_Ctrl	Motor_Ctrl_Def	I	8flags	Motor control variable
Motor_Status	Motor_Status_Def	O	8flags	Motor status variable
Failure	Failure_Def	O	8flags	Failure variable
Sp_Input	U8	I	< 0; 255>	Speed input variable used for required speed calculation
Speed_Range_Max_RPM	U16	O	< 0; 65535> [rpm]	Speed range maximum
Speed_Max_RPM	U16	O	< 0; 65535> [rpm]	Maximal speed limit
Speed_Min_RPM	U16	O	< 0; 65535> [rpm]	Minimal speed limit
Commut_Rev	U8	O	< 0; 255>	Commutations per motor revolution
Curr	S8	O	<-Curr_Range_Max_cA; Curr_Range_Max_cA)	dc-bus current
Curr_Range_Max_cA	S16	O	<-32768;32767> [A*10^-2]	Current range maximum [A*10^-2]

Type: *S8- signed 8 bit, U8- unsigned 8 bit,S16- signed 16bit, U16- unsigned 16bit*

The system registers **Sys3, Motor_Ctrl, Motor_Status, Failure** flags are described by definitions of **Sys3_Def, Motor_Ctrl_Def, Motor_Status_Def, Failure_Def:**

```
typedef union
{
  struct
    {
     unsigned int HV   : 1;  /* BIT0 High Voltage board Flag */
     unsigned int LV   : 1;  /* BIT1 Low Voltage board */
     unsigned int EVMm : 1;  /* BIT2 EVMm board */
     unsigned int BIT3 : 1;  /* BIT3 RESERVED */
    unsigned int PCMode : 1;/* BIT4 PCMaster/manual mode Flag */
     unsigned int BIT5 : 1;  /* BIT5 RESERVED */
     unsigned int BIT6 : 1;  /* BIT6 RESERVED */
     unsigned int Alignment : 1; /* BIT7 Alignment state
Proceeding */
    } B;
/* |Alignment|***|***|PCMode|***|EVMm|LV||HV| */
     char R;
} Sys3_Def;
/* System register #3 Definition */

typedef union
{
  struct
    {
     unsigned int StartCtrl : 1; /* Switch Start set to START
Flag */
     unsigned int BIT1      : 1; /* BIT1 RESERVED */
     unsigned int BIT2      : 1; /* BIT2 RESERVED */
     unsigned int BIT3      : 1; /* BIT5 RESERVED */
     unsigned int BIT4      : 1; /* BIT4 RESERVED */
     unsigned int BIT5      : 1; /* BIT6 RESERVED */
     unsigned int BIT6      : 1; /* BIT6 RESERVED */
    unsigned int ClearFail : 1; /* BIT7 Clear failure Status  */
    } B;
/* |ClearFail|***|***|***|***|***|***|StartCtrl| */
     char R;
} Motor_Ctrl_Def;
/* PC master software Motor Control Flags Definition */

typedef union
{
  struct
    {
```

```
     unsigned int Switch_Start : 1; /* BIT0 Switch START/STOP
                              set to START Flag */
    unsigned int Running : 1;/* BIT1 Motor is running (Alignment,
Start
                                                              or
Running state) */
     unsigned int BIT2 : 1;  /* BIT2 RESERVED */
     unsigned int V120 : 1;  /* BIT3 120 V DC-Bus detected
                         (only for HV DC-Bus) */
     unsigned int BIT4 : 1;  /* BIT4 RESERVED */
     unsigned int BIT5 : 1;  /* BIT5 RESERVED */
     unsigned int BIT6 : 1;  /* BIT6 RESERVED */
     unsigned int BIT7 : 1;  /* BIT7 RESERVED */
    } B;
/* |***|***|***|***|V120|***|Running|Switch_Start| */
     char R;
} Motor_Status_Def;
/* PC master software Motor Status Flags register Definition */

typedef union
{
  struct
    {
     unsigned int OverCurrent : 1;     /* BIT0 Over-Current
Failure */
     unsigned int OverHeating : 1;     /* BIT1 Over-Heating */
     unsigned int VoltageFailure : 1;  /* BIT2 Over-Voltage */
     unsigned int BIT3        : 1;     /* BIT5 RESERVED */
     unsigned int BIT4        : 1;     /* BIT4 RESERVED */
     unsigned int BIT5        : 1;     /* BIT6 RESERVED */
    unsigned int BoardIdFail : 1;     /* BIT6 pcb Identification
                             Failure */
    unsigned int ErrCmt  : 1;        /* BIT7 error Commutation */
    } B;
 /*
|ErrCmt|***|***|***|***|VoltageFailure|OverHeating|OverCurrent|
*/
     char R;
} Failure_Def; /* Failure Flags register Definition */
```

Table 3-2 declares if the variable is used as an output or input from the BLDC control MCU software side. The variable is described and the unit defined.

When PC master software mode is set, the system start and stop is controlled by **StartCtrl** flag in **Motor_Ctrl** variable. When the application enters the fault state, the variable **Failure** displays the fault reason. Setting the **ClearFail** flag in **Motor_Ctrl** will exit the fault state.

The **Sp_Input** variable is used for speed control. In PC master software mode, it can be modified from PC master software (otherwise, it is set according to speed potentiometer value).

Desired speed [rpm] =
Sp_Input/255*(Speed_Max_RPM-Speed_Min_RPM) + Speed_Min_RPM

So, the required motor commutation period is determined by the **Speed_Max_RPM** and **Speed_Min_RPM** variables. These are chosen according to which optional board and motor set by the BLDC control MCU software.

The variable **Speed_Range_Max_RPM** determines scaling of the speed variables.

The actual speed of the motor can be calculated from **Per_Speed_MAX_Range** and zero crossing period **Per_ZCrosFlt_T2**:

Actual speed [rpm] =
Speed_Range_Max_RPM*Per_Speed_MAX_Range/Per_ZCrosFlt_T2

The variable **Commut_Rev** can be used for calculation of the BLDC motor commutation period:

Commutation Period [s] = 60 / Actual Speed [rpm] / **Commut_Rev**

The variable **Curr_Range_Max_cA** determines scaling of the current variables. So, the actual dc-bus current is:

dc-bus current [A] = **Curr** / 256*__Curr_Range_Max_cA__ / 100

Section 4. Hardware Design

4.1 Contents

4.2 System Configuration and Documentation

The application is designed to drive the 3-phase BLDC motor. The HW is a modular system composed from board and motor. There are three possible hardware options:

- **High-Voltage Hardware Set Configuration**
- **Low-Voltage Evaluation Motor Hardware Set Configuration**
- **Low-Voltage Hardware Set Configuration**

The following subsection shows the system configurations.

They systems consists of the following modules (see also **Figure 4-1**, **Figure 4-2**, **Figure 4-3**):

For all hardware options:

- **MC68HC908MR32 Control Board**

For High-Voltage Hardware Set cofiguration:

- **3-Phase AC/BLDC High Voltage Power Stage**
- **Optoisolation Board**
- **3-phase BLDC High Voltage Motor with Motor Brake**

For Low-Voltage Evaluation Motor Hardware Set configuration:

- **EVM Motor Board**
- **3-phase Low Voltage EVM BLDC Motor**

Low-Voltage Hardware Set configuration:

- **3-Ph AC/BLDC Low Voltage Power Stage**
- **3-phase BLDC Low Voltage Motor with Motor Brake**

The supplied controller boards for MC68HC908MR32 (ECCTRMR32) allows two possibilities for software execution:

1. MMDS Evaluation Board (KITMMDSMR32)
   Using a real-time debugger (supplied with the Metrowerks compiler) the evaluation board is connected to the controller board (ECCTRMR32) via an emulator connector. This solution is recommended for software evaluation.
2. Programmed MCU (MC68HC908MR32)

   Where a daughter board module, with MC68HC908MR32 processor, is plugged into the controller board (ECCTRMR32) instead of the emulator cable, the processor must be programmed in an external programmer. This solution is recommended for final tests.

Figure 4-1, **Figure 4-2** and **Figure 4-3** show the configuration with MMDS evaluation board.

The sections **4.3 All HW Sets Components**, **4.4 High-Voltage Hardware Set Components**, **4.5 Low-Voltage Evaluation Motor Hardware Set Components** and **4.6 Low-Voltage Hardware Set Components** will describe the individual boards.

4.2.1 High-Voltage Hardware Set Configuration

The system configuration for a high-voltage hardware set is shown in Figure 4-1.

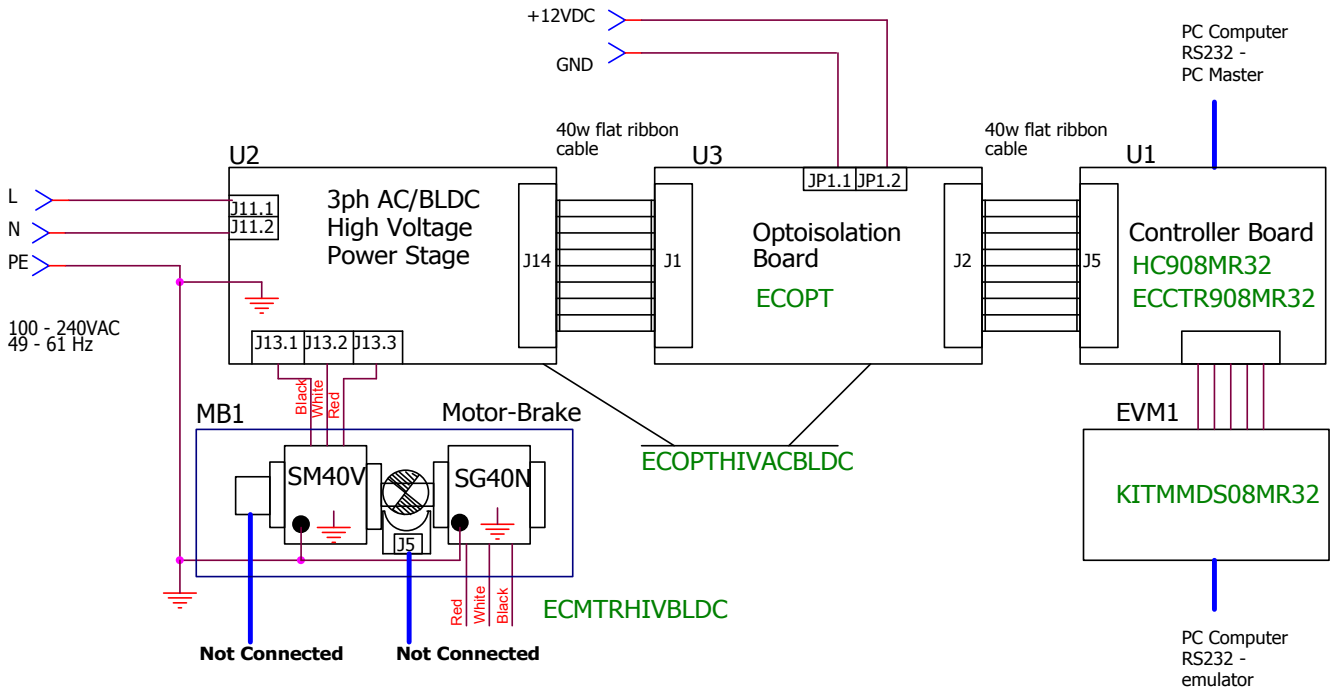

Figure 4-1. High-Voltage Hardware System Configuration

All the system parts are supplied and documented according to the following references:

- EVM1 — Modular Development System with EM08MR32 Daughter Board:
  - Supplied as: KITMMDS08MR32
  - Described in: Manual supplied with kit
- U1 — Controller Board for MC68HC908MR32:
  - Supplied as: ECCTR908MR32

  - Described in: *MC68HC908MR32 Control Board — User's Manual* (Motorola document order number MEMCMR32CBUM/D), see **References 3**

- U2 — 3-Phase AC/BLDC High Voltage Power Stage:
  - Supplied in kit with optoisolation board as: ECOPTHIVACBLDC
  - Described in: *3-Phase AC Brushless DC High Voltage Power Stage User's Manual* (Motorola document order number MEMC3PBLDCPSUM/D), see **References 4.**
- U3 — Optoisolation Board
  - Supplied with 3-phase AC/BLDC high voltage power stage as: ECOPTHIVACBLDC
  - Or, supplied alone as: ECOPT–ECOPT optoisolation board
  - Described in: *Optoisolation Board User's Manual* (Motorola document order number MEMCOBUM/D), see **References 5**

NOTE: *It is strongly recommended to use opto-isolation (optocouplers and optoisolation amplifiers) during development time to avoid any damage to the development equipment.*

- MB1 — Motor-Brake SM40V + SG40N
  - Supplied as: ECMTRHIVBLDC

The individual modules are described in some sections below. More detailed descriptions of the boards can be found in comprehensive User's Manuals belonging to each board (**References 3**, **4**, **5**). These manuals are available on on the World Wide Web at:

http://www.motorola.com

The User's Manual incorporates the schematic of the board, description of individual function blocks and a bill of materials. An individual board can be ordered from Motorola as a standard product.

4.2.2 Low-Voltage Evaluation Motor Hardware Set Configuration

The system configuration for a low-voltage evaluation motor hardware set is shown in **Figure 4-2**.

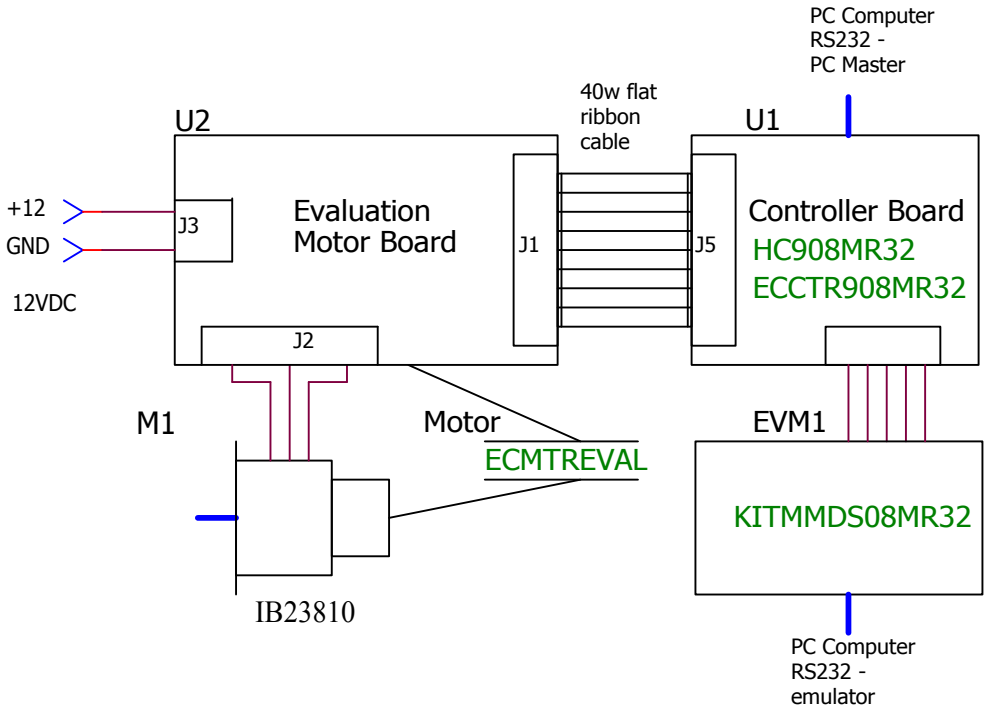

Figure 4-2. Low-Voltage Evaluation Motor Hardware System Configuration

All the system parts are supplied and documented according to the following references:

- EVM1 — Modular Development System with EM08MR32 Daughter Board:
  - Supplied as: KITMMDS08MR32
  - Described in: Manual supplied with kit
- U1 — Controller Board for MC68HC908MR32:
  - Supplied as: ECCTR908MR32

  - Described in: *MC68HC908MR32 Control Board — User's Manual* (Motorola document order number MEMCMR32CBUM/D), see **References 3**

- M1 — IB23810 Motor
  - Supplied in kit with IB23810 Motor as: ECMTREVAL — Evaluation Motor Board Kit
- U2 — 3-Phase AC/BLDC Low Voltage Power Stage:
  - Supplied in kit with IB23810 Motor as: ECMTREVAL — Evaluation Motor Board Kit
  - Described in: *Motorola Embedded Motion Control Evaluation Motor Board User's Manual* (Motorola document order number MEMCEVMBUM/D) see **References 6**

The individual modules are described in some sections below. More detailed descriptions of the boards can be found in comprehensive User's Manuals belonging to each board (**References 3**, **6**). These manuals are available on on the World Wide Web at:

http://www.motorola.com

The User's Manual incorporates the schematic of the board, description of individual function blocks and a bill of materials. An individual board can be ordered from Motorola as a standard product.

4.2.3 Low-Voltage Hardware Set Configuration

The system configuration for low-voltage hardware set is shown in **Figure 4-3**.

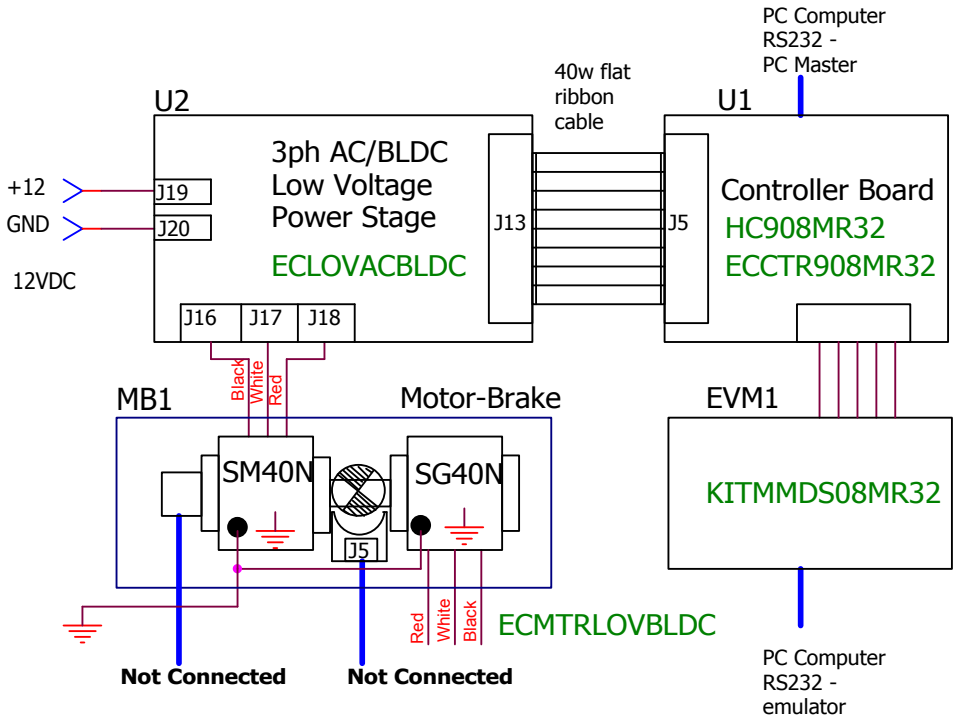

Figure 4-3. Low-Voltage Hardware System Configuration

All the system parts are supplied and documented according to the following references:

- EVM1 — Modular Development System with EM08MR32 Daughter Board:
  - Supplied as: KITMMDS08MR32
  - Described in: Manual supplied with kit
- U1 — Controller Board for MC68HC908MR32:
  - Supplied as: ECCTR908MR32
  - Described in: *MC68HC908MR32 Control Board — User's Manual* (Motorola document order number MEMCMR32CBUM/D), see **References 3**
- U2 — 3-Phase AC/BLDC Low Voltage Power Stage
  - Supplied as: ECLOVACBLDC

  - Described in: *Motorola Embedded Motion Control 3-Phase BLDC Low-Voltage Power Stage User's Manual* (Motorola document order number MEMC3PBLDCLVUM/D3), see **References 7**

- MB1 — Motor-Brake SM40N + SG40N
  - Supplied as: ECMTRLOVBLDC

The individual modules are described in some sections below. More detailed descriptions of the boards can be found in comprehensive User's Manuals belonging to each board (**References 3**, **7**). These manuals are available on on the World Wide Web at:

http://www.motorola.com

The User's Manual incorporates the schematic of the board, description of individual function blocks and a bill of materials. An individual board can be ordered from Motorola as a standard product.

4.3 All HW Sets Components

4.3.1 MC68HC908MR32 Control Board

Motorola's embedded motion control series MR32 motor control board is designed to provide control signals for 3-phase ac induction, 3-phase brushless dc (BLDC), and 3-phase switched reluctance (SR) motors. In combination with one of the embedded motion control series power stages, and an optoisolation board, it provides a software development platform that allows algorithms to be written and tested without the need to design and build hardware. With software supplied on the CD-ROM, the control board supports a wide variety of algorithms for ac induction, SR, and BLDC motors. User control inputs are accepted from START/STOP, FWD/REV switches, and a SPEED potentiometer located on the control board. Alternately, motor commands can be entered via a PC and transmitted over a serial cable to DB-9 connector. Output connections and power stage feedback signals are grouped together on 40-pin ribbon cable connector. Motor feedback signals can be connected to Hall sensor/encoder connector. Power is supplied

through the 40-pin ribbon cable from the optoisolation board or low-voltage power stage.

The control board is designed to run in two configurations. It can be connected to an M68EM08MR32 emulator via an M68CBL08A impedance matched ribbon cable, or it can operate using the daughter board. The M68EM08MR32 emulator board may be used in either an MMDS05/08 or MMEVS05/08 emulation system.

Figure 4-4 shows a block diagram of the board's circuitry.

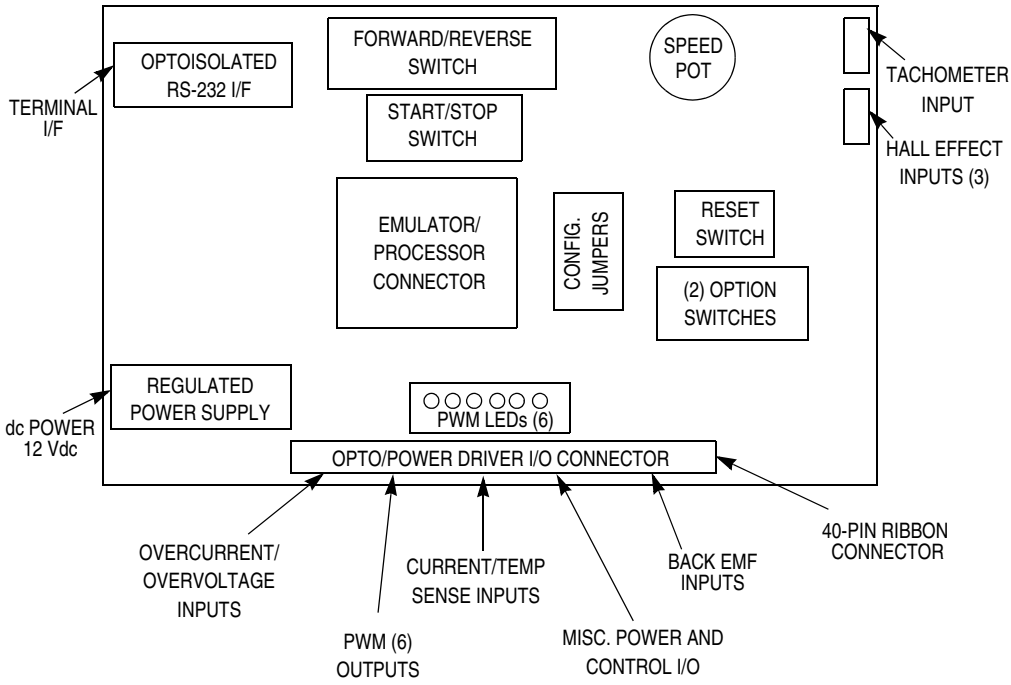

Figure 4-4. MC68HC908MR32 Control Board

4.3.1.1 Electrical Characteristics of the MC68HC908MR32 Control Board

The electrical characteristics in **Table 4-1** apply to operation at 25°C.

Table 4-1. Electrical Characteristics of Control Board

Characteristic	Symbol	Min	Typ	Max	Units
dc power supply voltage(1)	Vdc	10.8	12	16.5	V
Quiescent current	I_{CC}	—	80	—	mA
Min logic 1 input voltage (MR32)	V_{IH}	2.0	—	—	V
Max logic 0 input voltage (MR32)	V_{IL}	—	—	0.8	V
Propagation delay (Hall sensor/encoder input)	t_{dly}	—	—	500	ns
Analog input range	V_{In}	0	—	5.0	V
RS-232 connection speed		—	—	9600	Baud
PWM sink current	I_{PK}	—	—	20	mA

1. When operated and powered separately from other Embedded Motion Control tool set products

4.4 High-Voltage Hardware Set Components

4.4.1 3-Phase AC/BLDC High Voltage Power Stage

Motorola's embedded motion control series high-voltage (HV) ac power stage is a 180-watt (one-fourth horsepower), 3-phase power stage that will operate off of dc input voltages from 140 to 230 volts and ac line voltages from 100 to 240 volts. In combination with one of the embedded motion control series control boards and an optoisolation board, it provides a software development platform that allows algorithms to be written and tested without the need to design and build a power stage. It supports a wide variety of algorithms for both ac induction and brushless dc (BLDC) motors.

Input connections are made via 40-pin ribbon cable connector J14. Power connections to the motor are made on output connector J13. Phase A, phase B, and phase C are labeled PH_A, Ph_B, and Ph_C on the board. Power requirements are met with a single external 140- to

230-volt dc power supply or an ac line voltage. Either input is supplied through connector J11. Current measuring circuitry is set up for 2.93 amps full scale. Both bus and phase leg currents are measured. A cycle-by-cycle over-current trip point is set at 2.69 amps.

The high-voltage ac power stage has both a printed circuit board and a power substrate. The printed circuit board contains IGBT gate drive circuits, analog signal conditioning, low-voltage power supplies, power factor control circuitry, and some of the large, passive, power components. All of the power electronics which need to dissipate heat are mounted on the power substrate. This substrate includes the power IGBTs, brake resistors, current sensing resistors, a power factor correction MOSFET, and temperature sensing diodes. **Figure 4-5** shows a block diagram.

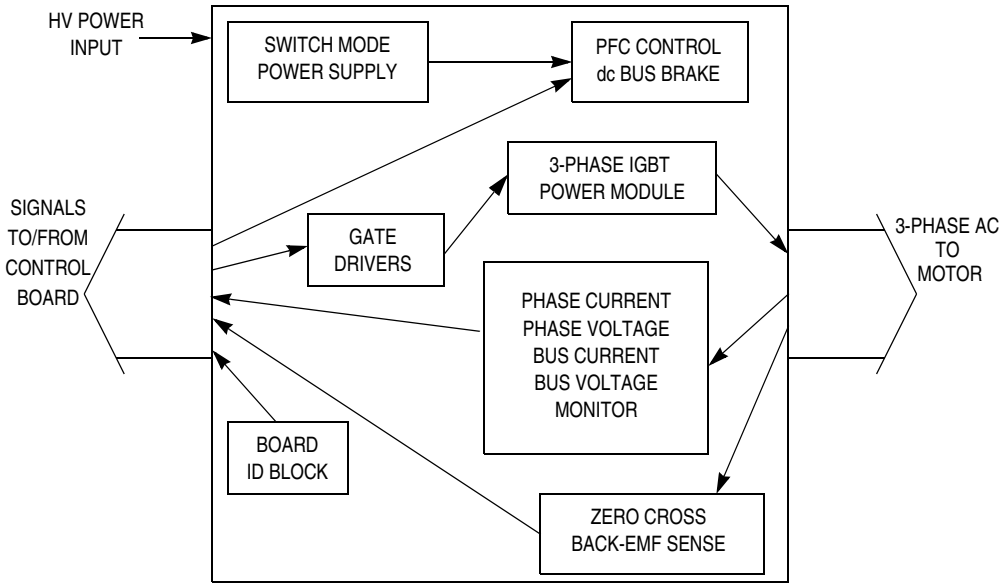

Figure 4-5. 3-Phase AC High Voltage Power Stage

4.4.1.1 Electrical Characteristics of the 3-Phase AC/BLDC High Voltage Power Stage

The electrical characteristics in **Table 4-2** apply to operation at 25°C with a 160-Vdc power supply voltage.

Table 4-2. Electrical Characteristics of Power Stage

Characteristic	Symbol	Min	Typ	Max	Units
dc input voltage	Vdc	140	160	230	V
ac input voltage	Vac	100	208	240	V
Quiescent current	I_{CC}	—	70	—	mA
Min logic 1 input voltage	V_{IH}	2.0	—	—	V
Max logic 0 input voltage	V_{IL}	—	—	0.8	V
Input resistance	R_{In}	—	10 kΩ	—	
Analog output range	V_{Out}	0	—	3.3	V
Bus current sense voltage	I_{Sense}	—	563	—	mV/A
Bus voltage sense voltage	V_{Bus}	—	8.09	—	mV/V
Peak output current	I_{PK}	—	—	2.8	A
Brake resistor dissipation (continuous)	P_{BK}	—	—	50	W
Brake resistor dissipation (15 sec pk)	$P_{BK(Pk)}$	—	—	100	W
Total power dissipation	P_{diss}	—	—	85	W

4.4.2 Optoisolation Board

Motorola's embedded motion control series optoisolation board links signals from a controller to a high-voltage power stage. The board isolates the controller, and peripherals that may be attached to the controller, from dangerous voltages that are present on the power stage. The optoisolation board's galvanic isolation barrier also isolates control signals from high noise in the power stage and provides a noise-robust systems architecture.

Signal translation is virtually one-for-one. Gate drive signals are passed from controller to power stage via high-speed, high dv/dt, digital optocouplers. Analog feedback signals are passed back through HCNR201 high-linearity analog optocouplers. Delay times are typically

250 ns for digital signals, and 2 µs for analog signals. Grounds are separated by the optocouplers' galvanic isolation barrier.

Both input and output connections are made via 40-pin ribbon cable connectors. The pin assignments for both connectors are the same. For example, signal PWM_AT appears on pin 1 of the input connector and also on pin 1 of the output connector. In addition to the usual motor control signals, an MC68HC705JJ7CDW serves as a serial link, which allows controller software to identify the power board.

Power requirements for controller side circuitry are met with a single external 12-Vdc power supply. Power for power stage side circuitry is supplied from the power stage through the 40-pin output connector.

4.4.2.1 Electrical Characteristics of the Optoisolation Board

The electrical characteristics in **Table 4-3** apply to operation at 25°C, and a 12-Vdc power supply voltage.

Table 4-3. Electrical Characteristics

Characteristic	Symbol	Min	Typ	Max	Units	Notes
Power Supply Voltage	Vdc	10	12	30	V	
Quiescent Current	I_{CC}	70(1)	200(2)	500(3)	mA	dc/dc converter
Min Logic 1 Input Voltage	V_{IH}	2.0	—	—	V	HCT logic
Max Logic 0 Input Voltage	V_{IL}	—	—	0.8	V	HCT logic
Analog Input Range	V_{In}	0	—	3.3	V	
Input Resistance	R_{In}	—	10	—	kΩ	
Analog Output Range	V_{Out}	0	—	3.3	V	
Digital Delay Time	t_{DDLY}	—	0.25	—	µs	
Analog Delay Time	t_{ADLY}	—	2	—	µs	

1. Power supply powers optoisolation board only.
2. Current consumption of optoisolation board plus DSP EMV board (powered from this power supply)
3. Maximum current handled by dc/dc converters

4.4.3 3-phase BLDC High Voltage Motor with Motor Brake

The High Voltage BLDC motor-brake set incorporates a 3-phase High Voltage BLDC motor and attached BLDC motor brake. The BLDCmotor has six poles. The incremental position encoder is coupled to the motor shaft, and position Hall sensors are mounted between motor and brake. They allow sensing of the position if required by the control algorithm. Detailed motor-brake specifications are listed in **Table 2-2**, **Section 2.**

4.5 Low-Voltage Evaluation Motor Hardware Set Components

4.5.1 EVM Motor Board

Motorola's embedded motion control series EVM motor board is a 12-volt, 4-amp, surface-mount power stage that is shipped with an MCG IB23810-H1 brushless dc motor. In combination with one of the embedded motion control series control boards, it provides a software development platform that allows algorithms to be written and tested without the need to design and build a power stage. It supports algorithms that use Hall sensors, encoder feedback, and back EMF (electromotive force) signals for sensorless control.

The EVM motor board does not have overcurrent protection that is independent of the control board, so some care in its setup and use is required if a lower impedance motor is used. With the motor that is supplied in the kit, the power output stage will withstand a full-stall condition without the need for overcurrent protection. Current measuring circuitry is set up for 4 amps full scale. In a 25^{o}C ambient operation at up to 6 amps continuous RMS output current is within the board's thermal limits.

Input connections are made via 40-pin ribbon cable connector J1. Power connections to the motor are made on output connector J2. Phase A, phase B, and phase C are labeled on the board. Power requirements are met with a single external 12-Vdc, 4-amp power supply. Two connectors, labeled J3 and J4, are provided for the 12-volt power supply. J3 and J4 are located on the front edge of the board. Power is supplied to one or the other, but not both.

4.5.1.1 Electrical Characteristics of the EVM Motor Board

The electrical characteristics in **Table 4-4** apply to operation at 25°C and a 12-Vdc power supply voltage.

Table 4-4. Electrical Characteristics of the EVM Motor Board

Characteristic	Symbol	Min	Typ	Max	Units
Power Supply Voltage	Vdc	10	12	16	V
Quiescent Current	I_{CC}	—	50	—	mA
Min Logic 1 Input Voltage	V_{IH}	2.4	—	—	V
Max Logic 0 Input Voltage	V_{IL}	—	—	0.8	V
Input Resistance	R_{In}	—	10	—	kΩ
Analog Output Range	V_{Out}	0	—	3.3	V
Bus Current Sense Voltage	I_{Sense}	—	412	—	mV/A
Bus Voltage Sense Voltage	V_{Bus}	—	206	—	mV/V
Power MOSFET On Resistance	$R_{DS(On)}$	—	32	40	MΩ
RMS Output Current	I_M	—	—	6	A
Total Power Dissipation	P_{diss}	—	—	5	W

4.5.2 3-phase Low Voltage EVM BLDC Motor

The EVM Motor Board is shipped with an MCG IB23810-H1 brushless dc motor. Motor-brake specifications are listed in **Table 2-3**, **Section 2.** Other detailed motor characteristics are in **Table 4-5** this section. They apply to operation at 25°C.

Table 4-5. Characteristics of the BLDC motor

Characteristic	Symbol	Min	Typ	Max	Units
Terminal Voltage	V_t	—	—	60	V
Speed @ V_t		—	5000	—	RPM
Torque Constant	K_t	—	0.08	—	Nm/A

Table 4-5. Characteristics of the BLDC motor

Voltage Constant	K_e	—	8.4	—	V/kRPM
Winding Resistance	R_t	—	2.8	—	Ω
Winding Inductance	L	—	8.6	—	mH
Continuous Current	I_{cs}	—	—	2	A
Peak Current	I_{ps}	—	—	5.9	A
Inertia	J_m	—	0.075	—	$kgcm^2$
Thermal Resistance		—	—	3.6	°C/W

4.6 Low-Voltage Hardware Set Components

4.6.1 3-Ph AC/BLDC Low Voltage Power Stage

Motorola's embedded motion control series low-voltage (LV) brushless DC (BLDC) power stage is designed to run 3-ph. BLDC and PM Synchronous motors. It operates from a nominal 12-volt motor supply, and delivers up to 30 amps of rms motor current from a dc bus that can deliver peak currents up to 46 amps. In combination with one of Motorola's embedded motion control series control boards, it provides a software development platform that allows algorithms to be written and tested, without the need to design and build a power stage. It supports a wide variety of algorithms for controlling BLDC motors and PM Synchronous motors.

Input connections are made via 40-pin ribbon cable connector J13. Power connections to the motor are made with fast-on connectors J16, J17, and J18. They are located along the back edge of the board, and are labeled Phase A, Phase B, and Phase C. Power requirements are met with a 12-volt power supply that has a 10- to 16-volt tolerance. Fast-on connectors J19 and J20 are used for the power supply. J19 is labeled +12V and is located on the back edge of the board. J20 is labeled 0V and is located along the front edge. Current measuring circuitry is set up for 50 amps full scale. Both bus and phase leg currents are measured. A cycle by cycle overcurrent trip point is set at 46 amps.

The LV BLDC power stage has both a printed circuit board and a power substrate. The printed circuit board contains MOSFET gate drive circuits, analog signal conditioning, low-voltage power supplies, and some of the large passive power components. This board also has a 68HC705JJ7 microcontroller used for board configuration and identification. All of the power electronics that need to dissipate heat are mounted on the power substrate. This substrate includes the power MOSFETs, brake resistors, current-sensing resistors, bus capacitors, and temperature sensing diodes. **Figure 4-5** shows a block diagram.

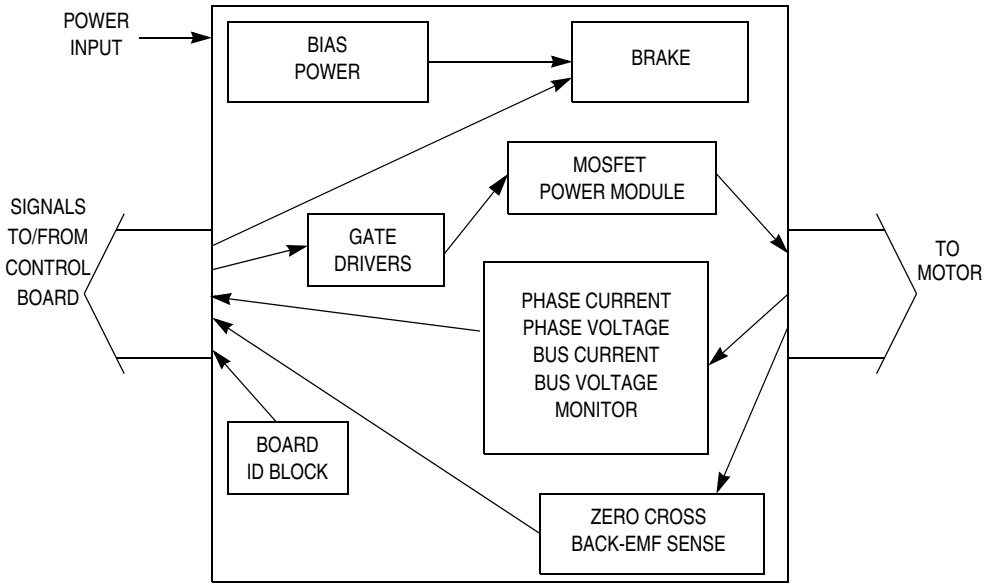

Figure 4-6. Block Diagram

4.6.1.1 Electrical Characteristics of the 3-Ph BLDC Low Voltage Power Stage

The electrical characteristics in **Table 4-6** apply to operation at 25°C with a 12-Vdc supply voltage.

Table 4-6. Electrical Chatacteristics of the 3-Ph BLDC Low Voltage Power Stage

Characteristic	Symbol	Min	Typ	Max	Units
Motor Supply Voltage	Vac	10	12	16	V
Quiescent current	I_{CC}	—	175	—	mA
Min logic 1 input voltage	V_{IH}	2.0	—	—	V
Max logic 0 input voltage	V_{IL}	—	—	0.8	V
Analog output range	V_{Out}	0	—	3.3	V
Bus current sense voltage	I_{Sense}	—	33	—	mV/A
Bus voltage sense voltage	V_{Bus}	—	60	—	mV/V
Peak output current (300 ms)	I_{PK}	—	—	46	A
Continuous output current	I_{RMS}	—	—	30	A
Brake resistor dissipation (continuous)	P_{BK}	—	—	50	W
Brake resistor dissipation (15 sec pk)	$P_{BK(Pk)}$	—	—	100	W
Total power dissipation	P_{diss}	—	—	85	W

4.6.2 3-phase BLDC Low Voltage Motor with Motor Brake

The Low Voltage BLDC motor-brake set incorporates a 3-phase Low Voltage BLDC motor EM Brno SM40N and attached BLDC motor brake SG40N. The BLDCmotor has six poles. The incremental position encoder is coupled to the motor shaft, and position Hall sensors are mounted between motor and brake. They allow sensing of the position if required by the control algorithm, which is not required in this sensorless application. Detailed motor-brake specifications are listed in **Table 2-4**, **Section 2.**

Section 5. Software Design

5.1 Contents

5.2 Introduction

This section describes the design of the software blocks of the drive. The software will be described in terms of:

- **Data Flow**
- **Main Software Flowchart**
- **State Diagram**

For more information on the control technique used see **3.3 Used Control Technique**.

5.3 Data Flow

The control algorithm obtains values from the user interface and sensors, processes them and generates 3-phase PWM signals for motor control, as can be seen on the data flow analysis shown in **Figure 5-1** and **Figure 5-2**.

5.3.1 Software Variables and Defined Constants

Important system variables are listed in Table 5-1.

Table 5-1. Software Variables

Name	Type	Representing Range	Description
Sys1	Sys1_Def	8flags	System variable #1
Speed_Min_U8	U8	< 0; Speed_Range_Max_RPM)	Minimal speed [system units]
Sp_Input	U8	< 0; 255>	Speed input variable used for required speed calculation
Coef_Speed_Inp	U8		Coeficient Sp_Inp to Speed_Desired calculation
Speed_Desired	U8	< 0; Speed_Range_Max_RPM)	Desired speed
PIParamsScl_U8_Speed	Structure		Speed PI regulator parameters
Per_Speed_MAX_Range	U16	[UNIT_PERIOD_T2_US]	Minimal commutation period of the speed range (at Speed_Range_Max_RPM)
Per_ZCrosFlt	U16	[UNIT_PERIOD_T2_US]	Zero crossing period — filtered
T2	U16(union)	[UNIT_PERIOD_T2_US]	Timer 2 variable
T_ZCros	U16	[UNIT_PERIOD_T2_US]	Zero crossing time [n]
T_ZCros0	U16(union)	[UNIT_PERIOD_T2_US]	Zero crossing time [n-1]
T_Cmt	U16	[UNIT_PERIOD_T2_US]	Commutation time
Curr	S8	<-Curr_Range_Max_cA; Curr_Range_Max_cA)	dc-bus current
Curr_Align	S8	<-Curr_Range_Max_cA; Curr_Range_Max_cA)	Required current during alignment state
PIParamsScl_S8_Currr	Structure		Current PI regulator parameters
Volt	U8	<-VOLT_RANGE_MAX; VOLT_RANGE_MAX)	dc-bus voltage
V_TASC2	U8		Back-EMF zero crossing expecting edge
V_MUX	U8		Preset value of back-EMF zero crossing phase multiplexer

Type: S8- signed 8 bit, U8- unsigned 8 bit, S16- signed 16bit, U16- unsigned 16bit, (union)- 16 bits access or 2*8bit access

The system registers **Sys1** flags are described by definitions of **Sys3_Def:**

```
typedef union
{
  struct
    {
    unsigned int PC_F    : 1;  /* BIT0 Phase Commutation Flag */
     unsigned int Off_F    : 1;  /* BIT1 Offset timeout flag
                                  - Offset timeout can be measured */
     unsigned int ICR_F    : 1;  /* BIT2 Input Capture
                                    was succesfuly Received - Flag */
     unsigned int Rmp_F    : 1;  /* BIT3 Speed Ramp Flag - motor
ramping */
     unsigned int Stop_F   : 1;  /* BIT4 Motor is going or is
stopped */
     unsigned int Strt_F   : 1;  /* BIT5 Start Phase Flag */
     unsigned int Run_F    : 1;  /* BIT6 Motor Running with
                                          back-EMF feedback Flag */
    unsigned int FOK_F    : 1;  /* BIT7 Feedback within the righ
time Flag */
    } B;
                /*
|FOK_F|Run_F|Strt_F|Stop_F|Rmp_F|ICR_F|Off_F|PC_F| */
  char R;
} Sys1_Def; /* System register #1 Definition */
```

Main data flow is displayed in **Figure 5-1**. The processes are described in the following subsections.

5.3.2 Process Measurement

The process provides measurement of analog values using ADC. The measured inputs are: dc-bus current, dc-bus voltage, and speed input. The measurement is provided by the measurement handler. The state diagram is explained in **State Diagram**.

5.3.3 Start/Stop Switch Reading and Start/Stop Decision

The process reads the start stop switch and provides start condition and clear failure decisions, as explained in **Stand-By** and **Fault State**.

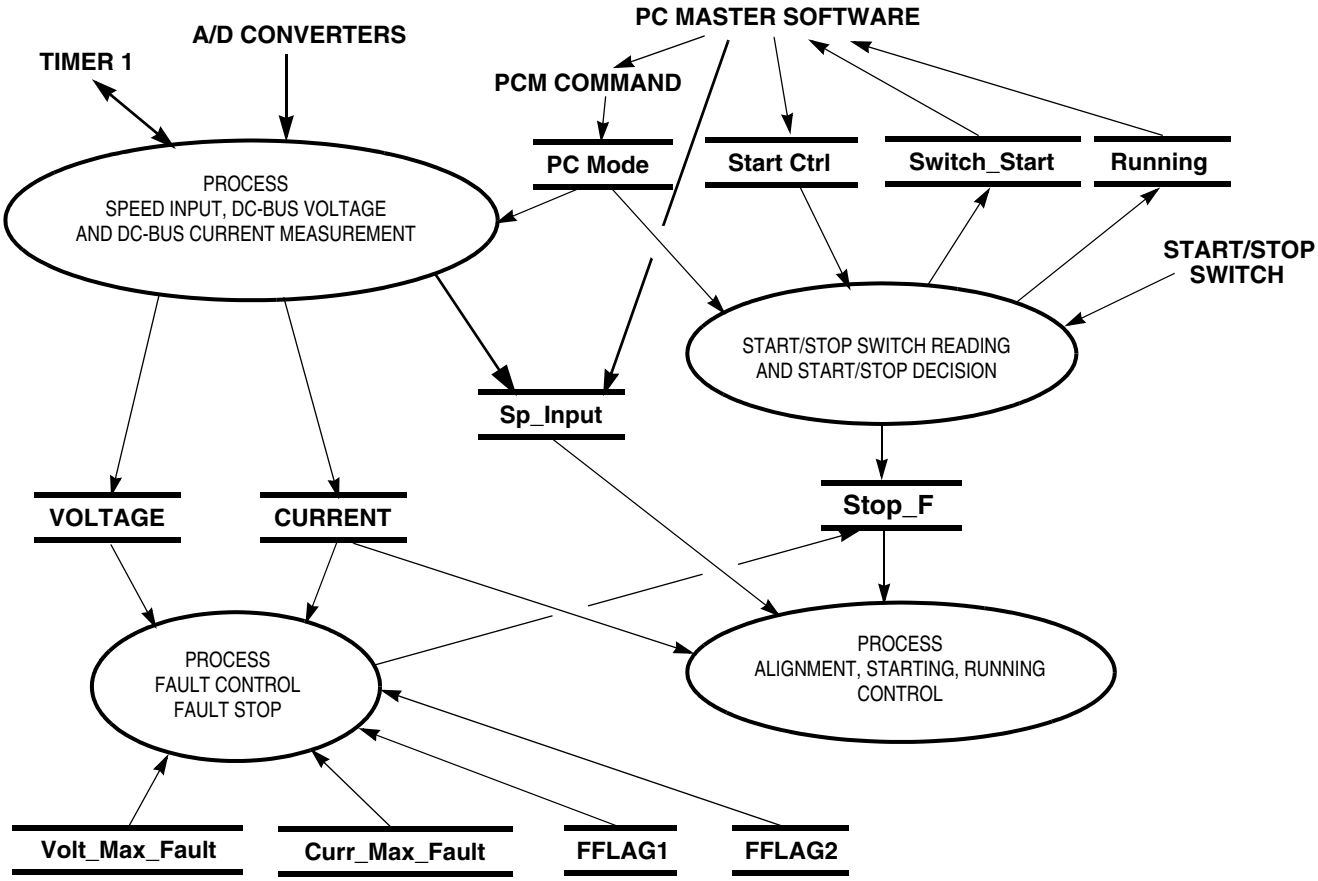

Figure 5-1. Main Data Flow — Part1

5.3.4 Process Fault Control Fault Stop

The process provides fault control and fault stop as described in **Fault State**, **Stand-By**, **Align State**, **5.5.5 Back-EMF Acquisition State**, and **Running State**.

The processes alignment, starting, and running control are displayed in **Figure 5-2**. The processes are described in the following subsections.

5.3.5 Process Back, EMF Zero Crossing Sensing

Back-EMF zero crossing process provides:

- Back-EMF zero crossing sampling in synchronization with PWM,

- Evaluates the zero crossing
- Records its time in T_ZCros

Further explanation is provided in **Data Flow** and **Figure 5-6**.

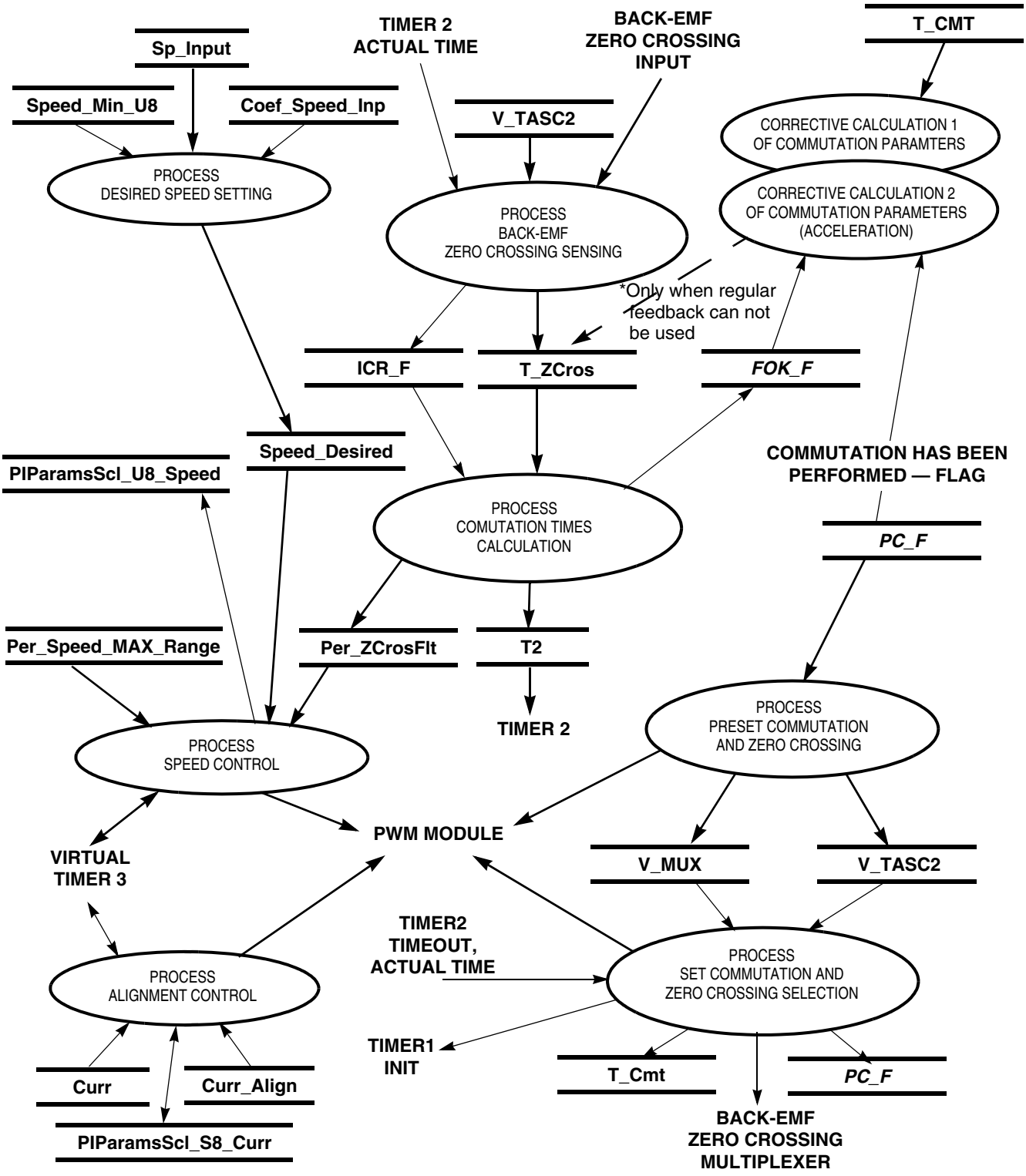

Figure 5-2. Main Data Flow — Part 2: Alignment, Starting, Running Control

5.3.6 Process Commutation Time Calculation, Corrective Calculation 1, Corrective Calculation 2

These processes provide calculations of commutation time intervals (periods) (**Per_ZCros**, **Per_ZCrosFlt**), from captured time (**T_Cmt**, **T_ZCros**, **T_ZCros0**), and sets Timer 2 with variable **T2**. These calculations are described in **3.3.1.5 Starting — Commutation Time Calculation** and **3.3.1.3 Running — Commutation Time Calculation**.

5.3.7 Process Desired Speed Setting

The desired speed, held in register **Speed_Desired**, is calculated from the following formula:

Speed_Desired = **Sp_Input***Coef_Speed_Inp/255 + **Speed_Min_U8**

5.3.8 Processs Speed Control

The general principle of the speed PI control loop is illustrated in **Figure 5-3**.

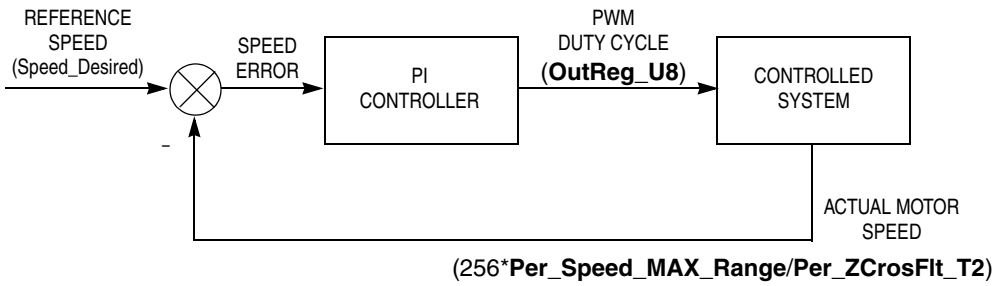

Figure 5-3. Closed Loop Control System

The speed closed loop control is characterized by the feedback of the actual motor speed.

The actual motor speed is calculated from zero crossing period:

Actual motor speed = 256***Per_Speed_MAX_Range/Per_ZCrosFlt_T2**

This information is compared with the reference set point and the error signal is generated. The magnitude and polarity of the error signal corresponds to the difference between the actual and desired speeds. Based on the speed error, the PI controller generates the corrected motor voltage in order to compensate for the error. The speed regulator parameters (gain...), internal, and input/output variables are located in the structure **PIParamsScl_U8_Speed**.

The speed controller works with a constant execution (sampling) period. The period is timed by timer 3, with the constant **PER_T3_RUN_US**.

PWM duty cycle is set for all six PWM channels according to regulator output, **OutReg_U8**. The maximum duty cycle is at **OutReg_U8** = 255. The implementation is described in **Implementation Notes** — **5.6.3 BLDC Speed Control and Calculation**.

5.3.9 Process Alignment Control

The process alignment control controls the current, Curr, using the PI regulator during alignment state (see **5.5 State Diagram**). The dc-bus current is regulated to required value Curr_Align. The current regulator parameters (gain...), internal, and input/output variables are located in the structure **PIParamsScl_S8_Curr**.

The current controller works with a constant execution (sampling) period. The period is timed by timer1, with the constant **PER_CS_T1_US**.

5.3.10 Processes Commutation and Zero Crossing Preset and Set

The processes commutation and zero crossing preset and set provides the BLDC commutation and zero crossing selection. Here the BLDC commutation means generation of the six step commutation which creates the voltage system shown in **Figure 3-2**. The required BLDC motor voltage system and commutation is provided using the MC68HC08MR32 PWM block.

The zero crossing selection means the selection of the required zero crossing phase as described in **3.2.4.2 Indirect Back EMF Sensing** and

3.2.5 Back-EMF Sensing Circuit.The zero crossing selection is provided by the multiplexer setting.

As shown in **Figure 5-2**, the commutation and back-EMF zero crossing selection process is split into two actions:

- Preset commutation and zero crossing selection
- The preset means setting the buffered registers and RAM variables for commutation
- Set commutation and zero crossing selection
- The setting means loading the registers with buffered variables

The implementation is described in **Implementation Notes - 5.6.2 BLDC Commutation and Zero Crossing Selection**.

5.4 Main Software Flowchart

The main software flowchart incorporates the main routine entered from the reset and interrupt states. The main routine includes initializing the MCU and the main loop. The flowcharts are shown in **Figure 5-4**, **Figure 5-5**, and **Figure 5-6**.

MCU Initialization is entered only after system reset. It provides initialization of system registers, ports, and CPU clock. The **MCU Initialization** is provided in **MCUInit()** function.

After **MCU Initialization** the **Application Initialization** is executed as **AppInit()** function, which performs the following actions. First the zero current offset of the dc-bus current measurement path is calibrated. This offset on the ADC input should be 1.65 V at zero current. This is implemented in the hardware design, in order to be able to measure negative and positive current values. The status registers are initialized and PWM generator is started. Also, timer 1 is started at the right moment to be synchronized with the PWM generator. This way the current measurement is executed at the defined moment of the PWM signal.

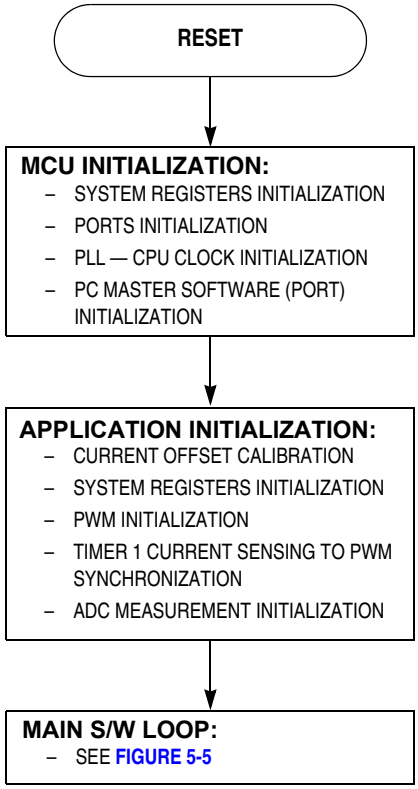

Figure 5-4. Main Software Flowchart

In the **Stand-By** state function, the start/stop switch is checked using **StSWReadStart ()** function. The **DecideStaSto ()** function is called to decide if the application should start. The start condition differs if manual or PC master software mode is set. When in manual mode (**PCMode** = 0), the start condition is the switch in the start position. When PC master software mode (**PCMode** = 1), the start condition is a start request from PC master software (**StartCtrl** = 1). In both modes, **Stop_F** is cleared when the software evaluates the start condition. When **Stop_F** is cleared, the software checks the over-voltage condition and the application starts.

The system **Alignment** and **Starting (Back-EMF Acquisition)** states are provided by **Alignment()** and **Start ()** functions in the **code_start.c** file, both are called from **main()**. The functionality during the start and running state is described in **3.3.1 Sensorless Commutation Control**. During the starting (back-EMF acquisition) state, the commutation time

preset calculations are prepared in the **StrtCmtPreset()** function, and commutation time set calculations are provided by the **StrtCmtSet()** function.

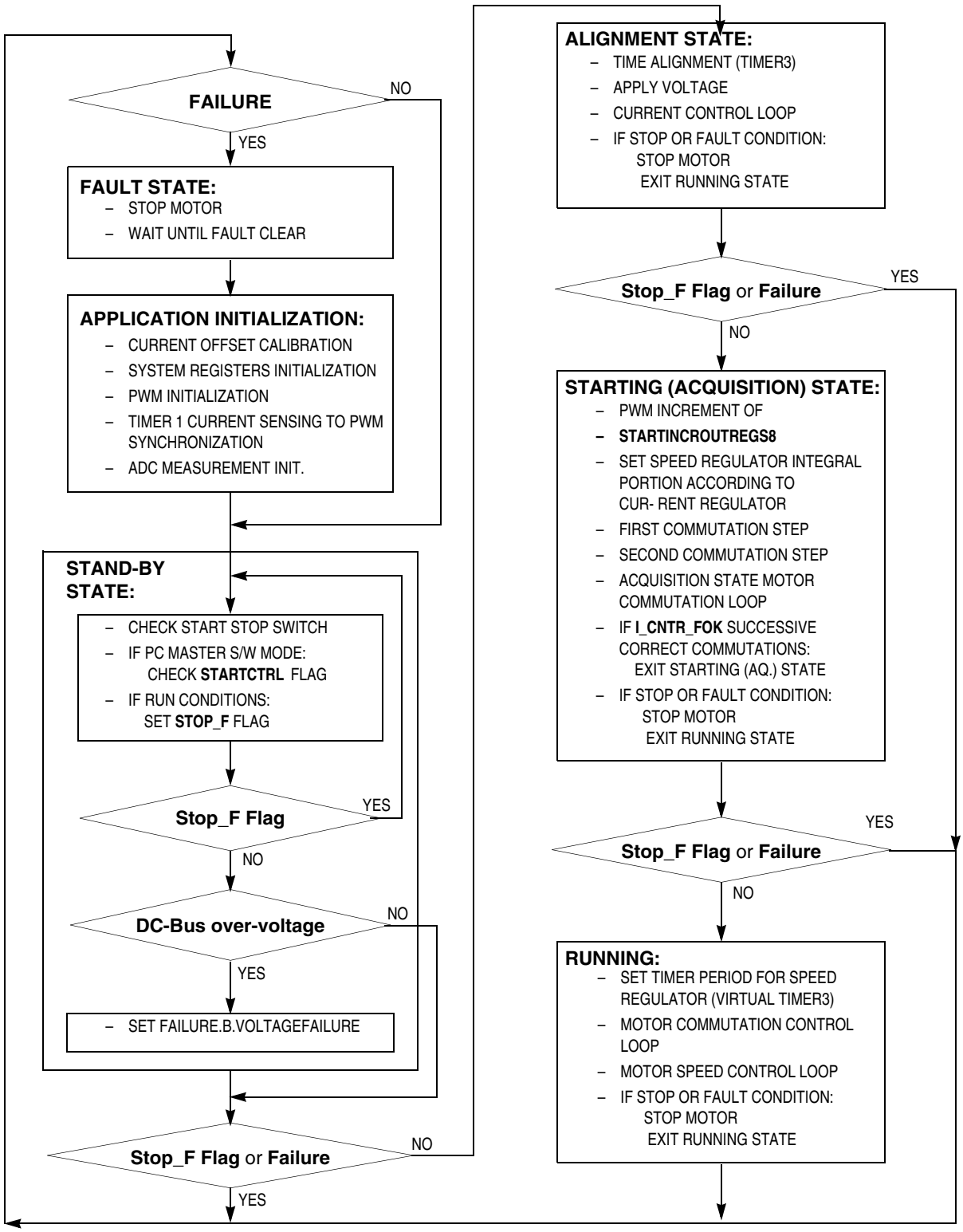

Figure 5-5. Main Software Flowchart — Main Software Loop

When the start is successfully completed, the **Running ()** function is called from **main()**. During the **Running** state, the commutation time preset calculations are provided by the **CmtPreset()** function, and commutation time set calculations are provided by the **CmtSet()** function.

During the **Running**, **Start** or **Alignment** states, the **DecideStop** () function is called to check drive stop conditions and can set the **Stop_F** flag. When the stop flag is set the **MotorStop ()** function is called to stop the motor, and running, start, or alignment state is left. The software enters stand-by state.

Also, the commutation error (**Cntr_Err**>= **MAX_ZC_ERR**) and over-current (**OverCurrent** flag = 1) fault are checked in **ERRHndl ()** and **OVCurrent ()** functions during the **Running**, **Start** or **Alignment** states. If any error is detected, the function **MotorStop ()** function is called. Then the software enters **Fault** state through the **Fault()** function. This is only left when the failures are cleared (variable **Failure** = 0). This decision is provided **DecideCleSto ()** function, called from **ErrorStop ()**. In manual control, the failures are cleared by setting the Start/Stop switch to Stop. In case of PC master software control, the failures are cleared by the **ClearFail** flag from the software. When the failures are cleared, the software enters **Application Initialization**.

Besides the main loop, there are three interrupts: timer 1, timer 2, and PWM reload interrupts. They are described in **Figure 5-6**.

Interrupt Timer 1 is periodically called with period **PER_CS_T1**. The interrupt function provides dc-bus current measurement and virtual timer 3 service, where timer 1 is providing the timer 3 time base. When over-current is discovered, the **OVC_F** flag is set. Finally, the ADC is set according to the **Nxt_Chnl** variable to prepare speed potentiometer or temperature sensing. This interrupt is provided by the **TIMACh1_Int()** function.

Interrupt Timer 2 sets commutation actions. If commutation is enabled (**CmtE_F** flag is set), the following actions are done:

- PWM commutation step
- Synchronization of timer 1, for current measurement with PWM

- Phase commutated flag **PC_F** is set
- Actual time (from timer counter register) = commutation time is recorded in **T_Cmt**.

This interrupt is provided by the **TIMACh3_Int()** function.

Interrupt PWM Reload provides back-EMF zero crossing sensing. The zero crossing input is sampled 2 or 3 times. The back-EMF state value is compared with expecting (rising/falling) edge. If the value corresponds with expecting edge, the zero crossing get flag **ICR_F** is set, and the actual time (from timer counter register) = zero crossing time is recorded in **T_ZCros**. This interrupt is provided by **PWMMC_Int()** function in **code_isr.c**.

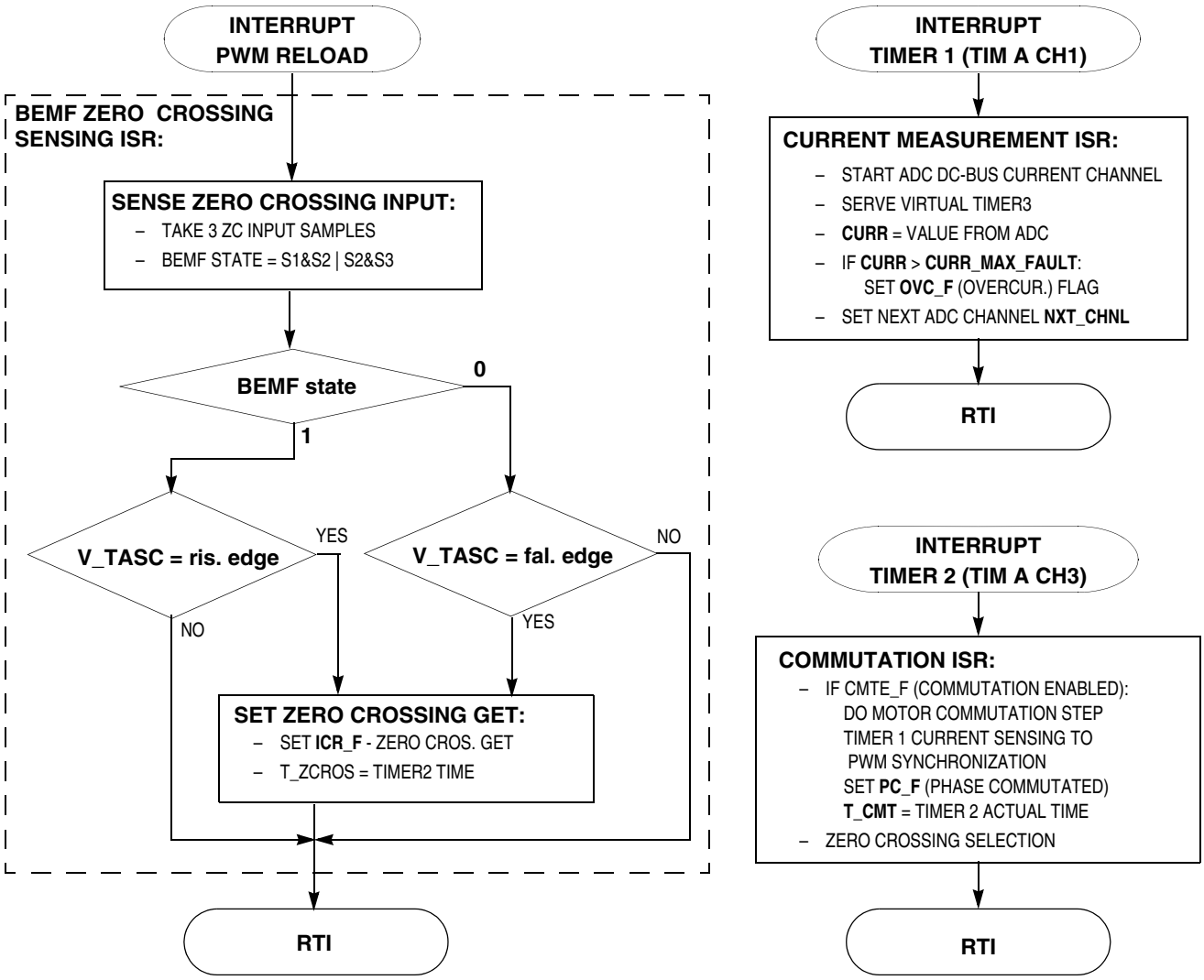

Figure 5-6. Software Flowchart — Interrupts

5.5 State Diagram

The motor control application can be in one of the eight states shown in Figure 5-7. Each of these states is described in the subsections following the figure.

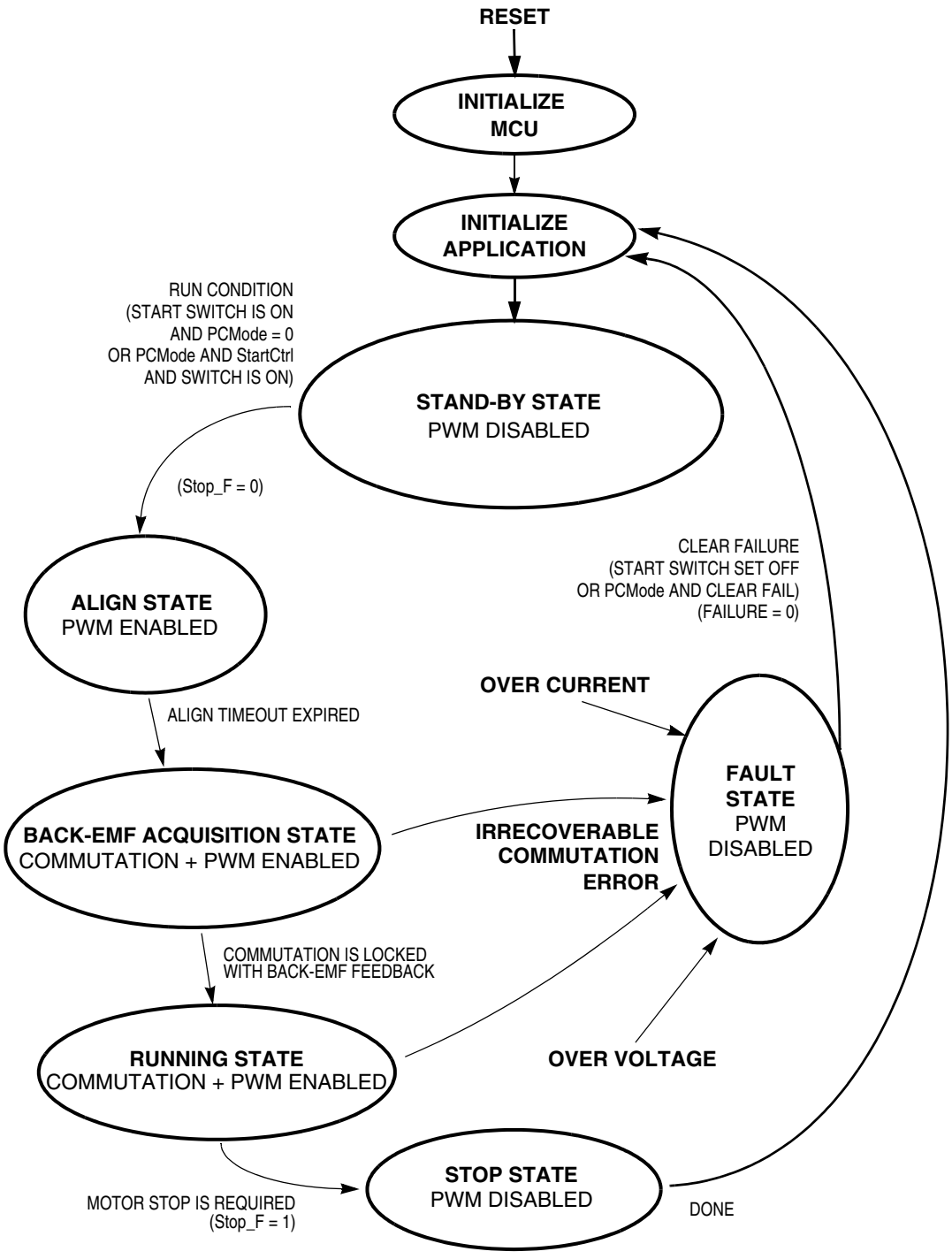

Figure 5-7. Application State Transitions

5.5.1 Initialize MCU

This state is entered after the MCU is reset, and performs the following actions:

- MCU ports are configured for the application
- Some application (system) variables are initialized
- MCU clock PLL is locked
- Hardware boards used are identified, and parameters initialized
- PC master communication software is initialized with SCI port
- ADC is initialized

and the state is exited.

5.5.2 Initialize Application

This state is used as an application reset, called following a return from fault or stop states.

In this state the following actions are done:

- Current measurement path calibrated and checked for error
- Some application (system) variables initialized
- Some MCU peripherals are configured (timer, OC function, PWM)
- PWM outputs for motor control are turned OFF
- Timer 1 (Tim A Ch1) is synchronized with the PWM cycle
- Speed input, dc-bus voltage and temperature measurement is initialized
- Ready LED is turned ON

and the state is exited.

5.5.3 Stand-By

State diagram for this software is shown in **Figure 5-8**.

Current measurement and current calibration when PWM is OFF

Before testing of the start switch, dc-bus current is measured when PWM is OFF. This way the dc-bus current measurement path is calibrated. The calibrated value **Offset0_Curr** is used for the final current calculation.

This offset on the ADC input should ideally be 1.65 V at 0 dc-bus current.

If the sensed value exceeds the limit (Offset_Max_Curr) when PWM is OFF, this indicates some hardware error, and the control flow enters into the fault state:

Start condition test

The start condition is tested. If in manual mode (PCMode = 0), the start condition is a movement in the switch from stop to start. In PC master software mode (PCMode = 1), the start condition in switch start position and StartCtrl = 1. If start condition valid then Strt_F is set, Stop_F is cleared, and the next state entered.

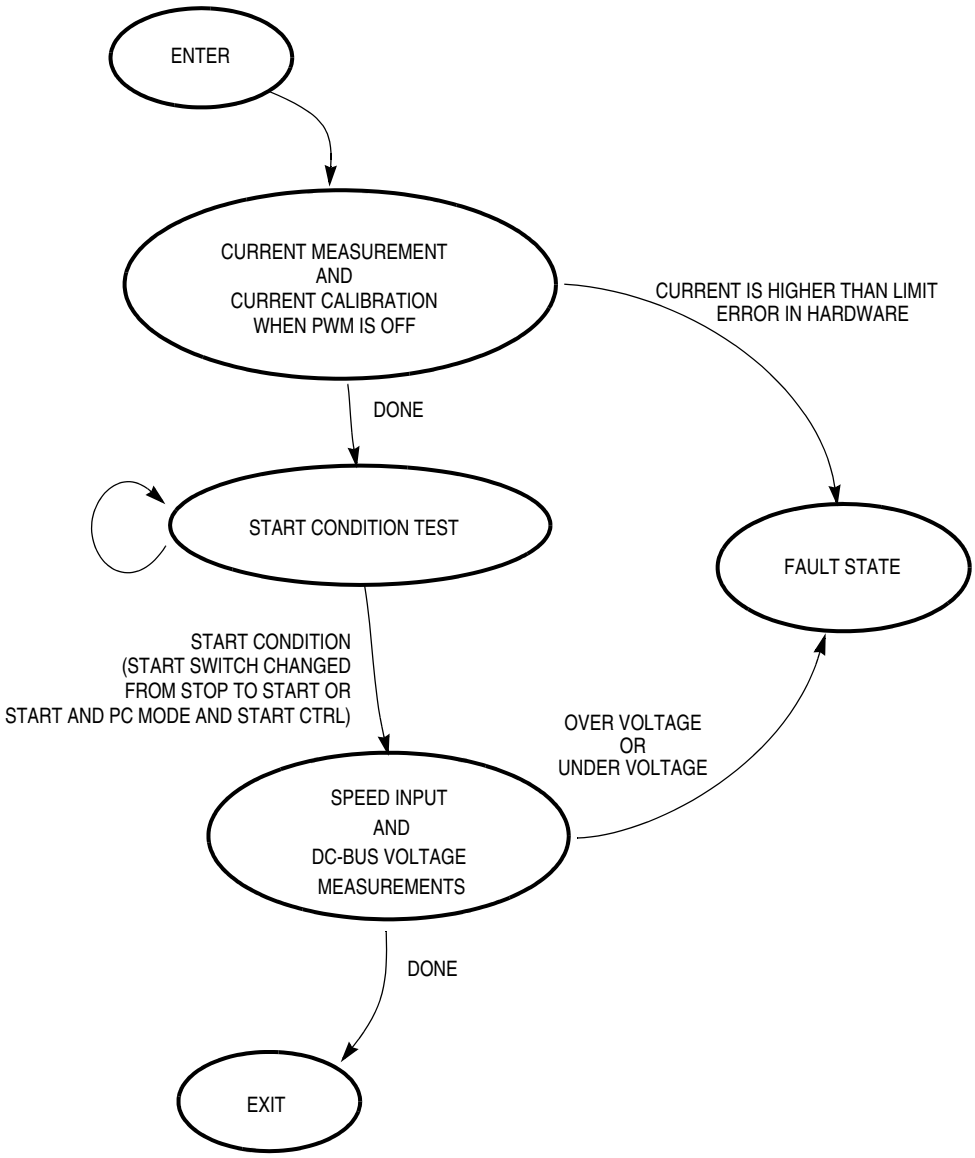

Figure 5-8. Stand-by State

Speed input and dc-bus voltage measurements

The dc-bus voltage is measured after the start switch is turned ON. This prevents the measurement being disturbed by the power turn ON. Where that dc-bus voltage is not within the limits, the control flow enters into the fault state.

5.5.4 Align State

In the align state the rotor position is stabilized by applying PWM signals to only two motor phases (no commutation). When preset time-out expires, then this state is finished. See **Figure 5-9**.

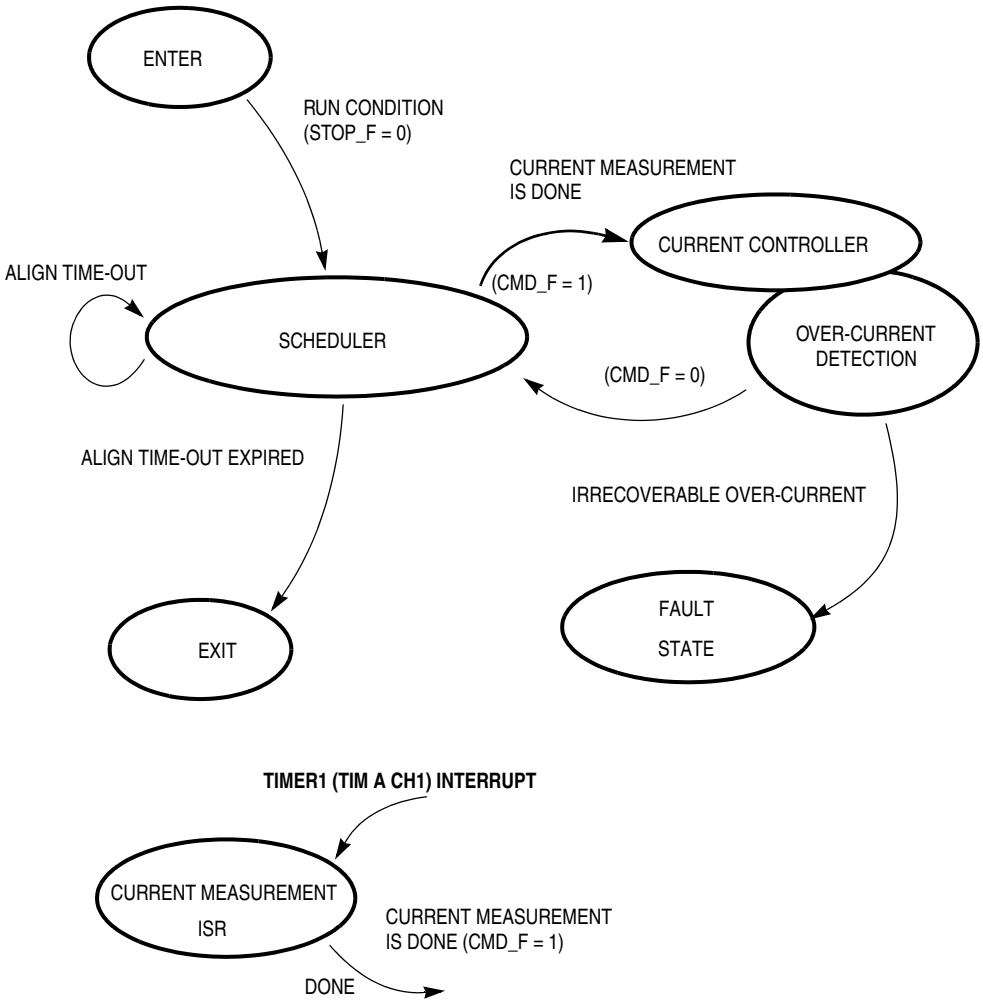

Figure 5-9. Align State

Scheduler

The scheduler handles the state transitions in the align state. The dc-bus current measurement is done in OC interrupt service routine, in order to keep synchronization with PWM cycle. After measurement is made, the scheduler allows calculation by the current controller and

the over-current detection. The CMD_F (current measurement done) flag indicates that the new value of dc-bus current is ready to be processed by the current controller.

The time-out (software timer 3) of this state is defined in the software by the constants: **PER_T_ALIGN** and **PER_BASE_T3_ALIGN**.

Current Controller

The current controller subroutine is called every **PER_CS_T1_US** µs (128 µs with default software setting) after a new value of the dc-bus current has been obtained (**CMD_F**=1). It sets all six PVALx register pairs to get the right PWM ratio for the required current.

Timer 1 Interrupt

Once the synchronization of OC function with the PWM cycle has been achieved, it must be maintained because the current measurement is initiated here.

Over-Current Detection

The dc-bus current is periodically sensed and the over-current condition is evaluated. After a defined number successive over-current events **I_CNTR_OVC**, the control flow enters into the fault state.

5.5.5 Back-EMF Acquisition State

The back-EMF acquisition state provides the functionality described in **3.3.1.4 Starting (Back-EMF Acquisition)** and **3.3.1.5 Starting — Commutation Time Calculation**. **Figure 5-10** shows the state transitions for the state.

First Commutation

After the align state time out expires, voltage is applied to another phase pair. The first commutation is made and the PWM duty cycle is constant. This value has been defined by the current controller during the Aalign state.

The calculation of the commutation time is explained in **3.3.1.5 Starting — Commutation Time Calculation**.

Second Commutation

The commutation time (T2) is calculated from the previous commutation time and the start commutation period (**Per_CmtStart**). This time is set to timer 2. Then, the new PWM multiplexer logic data is readied for when the commutation interrupt performs the next commutation.

The calculation of the commutation time is explained in **3.3.1.5 Starting — Commutation Time Calculation**.

Measurements Handler

As explained in **Scheduler** and **Over-Current Detection**, the dc-bus current is scanned and over-current condition is evaluated. Where there is an over-current, the control enters the fault state (see **Fault State**). The voltage and the speed commands are scanned too. When this state is finished, the values of dc-bus current, dc-bus voltage, and speed command are updated.

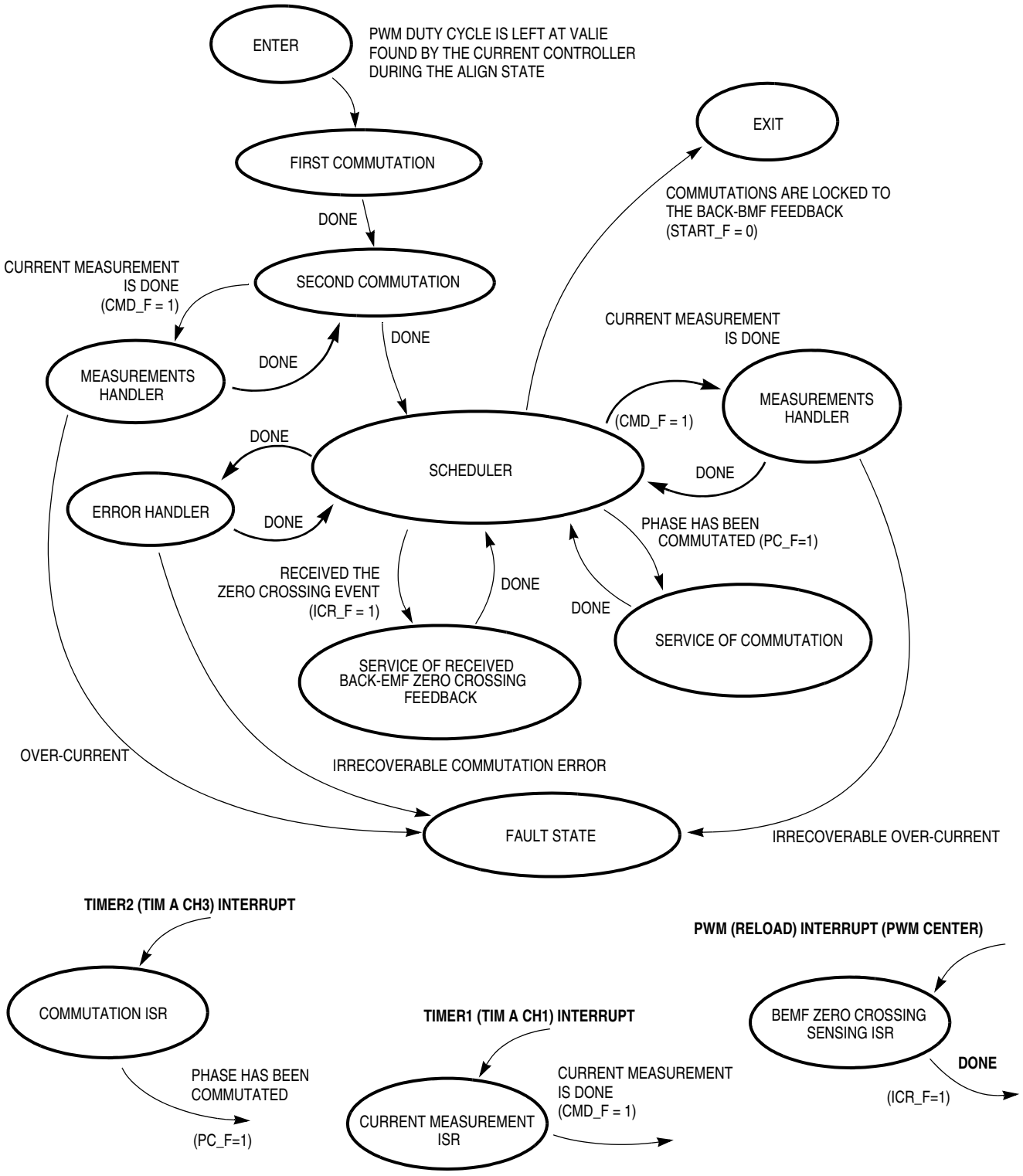

Figure 5-10. Back-EMF Acquisition

Service of Commutation

As already explained, the motor phase commutation is performed in the OC interrupt service routine. The phase commutated flag (**PC_F**=1) indicates this action to the scheduler, which allows the performed commutation to be serviced. Detailed explanation of this state is in **Processes Commutation and Zero Crossing Preset and Set**.

Service of Received Back-EMF Zero Crossing

The back-EMF zero crossing is detected by PWM middle function block. Then the appropriate flag (captured received the zero crossing event - ICR_F) is set by PWM centre interrupt service routine. This indicates to the scheduler that the zero crossing event must be served.

The following actions are taken:

1. Commutation parameters are recalculated more precisely based on the received feedback
2. Commutation time is preset to the output compare register of timer 2

For a better understanding of how the commutation process works, see **3.3.1 Sensorless Commutation Control**, (see **3.3.1.5 Starting — Commutation Time Calculation**).

BEMF Zero Crossing Sensing Interrupt Service Routine

This ISR is used to evaluate the back-EMF zero crossing. Back-EMF is evaluated here in order to synchronize zero crossing capture with the middle of central-aligned PWM. This technique rejects the noise caused by PWM from the back-EMF signal. When this ISR is initiated, then three samples of the zero crossing input (**BEMF_IN**) are taken and the state is evaluated. Based on the expected edge (**V_TASC2,** ELS2A_ELS2B) and the evaluated state of the **BEMF_IN** pin, the zero crossing event is verified. If it is accepted, then the captured time is stored in variable (**T_ZCros**) and the PWM ISR is finished. The appropriate flag (captured received the zero crossing event — **ICR_F**) is set.

Current Measurement Interrupt Service Routine

The output compare function is used to synchronize initiating the dc-bus current sampling with the PWM cycle, and also for the commutation timing.

Error Handler

If the BLDC motor is controlled properly, commutation events must be locked to the back-EMF zero crossing feedback. When that feedback is lost, commutation time is derived from previous commutation events. If feedback does not recover during a defined number of commutations (constant — C_MaxErr), then the situation is evaluated as irrecoverable commutation error, and the fault state is entered.

Measurement Handler

The measurement handler assures that the measurement process is done in the right order. The dc-bus voltage, speed command, and temperature are scanned sequentially. After the state has run three times, all the values for dc-bus current, dc-bus voltage, speed input, and temperature are updated. dc-bus current is scanned with a constant time period in the current measurement ISR, but the over-current condition is evaluated in the main software loop. After a defined number (I_OVC_Cnt) of successive over-current events, the control flow enters into the fault state.

5.5.6 Running State

The BLDC motor is run with regular feedback. The speed controller is used to control the motor speed by changing the value of PWM duty cycle. **Figure 5-11** shows the state transitions.

Measurements Handler

Explained in **5.5.5 Back-EMF Acquisition State**.

Service of Commutation

Explained in **5.5.5 Back-EMF Acquisition State**.

Service of Received Back-EMF Zero Crossing

Explained in **5.5.5 Back-EMF Acquisition State**. The difference is that the commutation parameters are recalculated with different constants (see **3.3.1.3 Running — Commutation Time Calculation**)

BEMF Zero Crossing Interrupt Service Routine

Explained in **5.5.5 Back-EMF Acquisition State**.

Current Measurement Interrupt Service Routine

Explained in **5.5.5 Back-EMF Acquisition State**.

Error Handler

Explained in **5.5.5 Back-EMF Acquisition State**.

Over Current

Explained in **5.5.5 Back-EMF Acquisition State**.

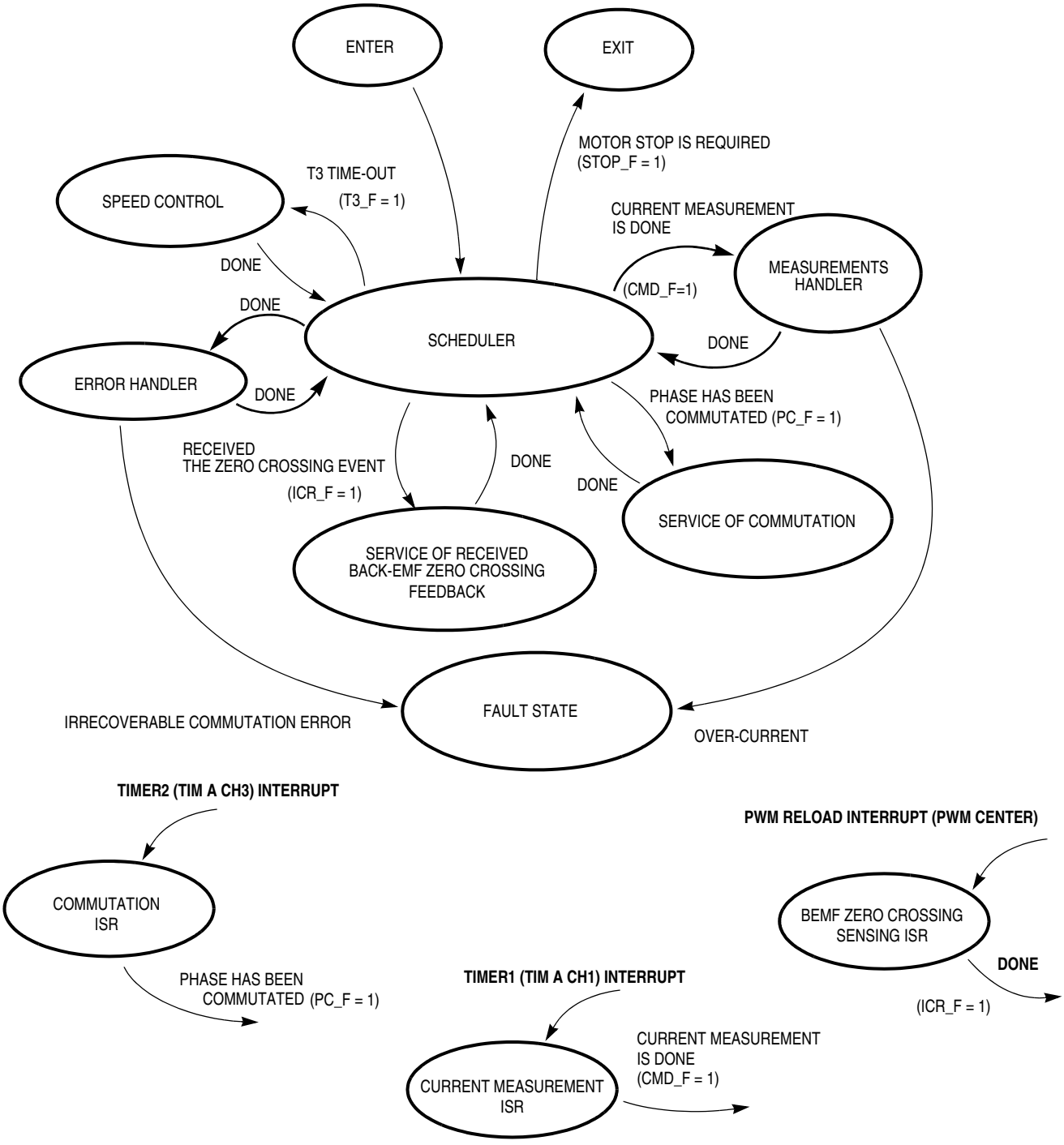

Figure 5-11. Running State

5.5.7 Stop State

When motor stop is required, the PWM signals are disabled and the power switches are switched off. The state diagram for this state is shown in
Figure 5-12.

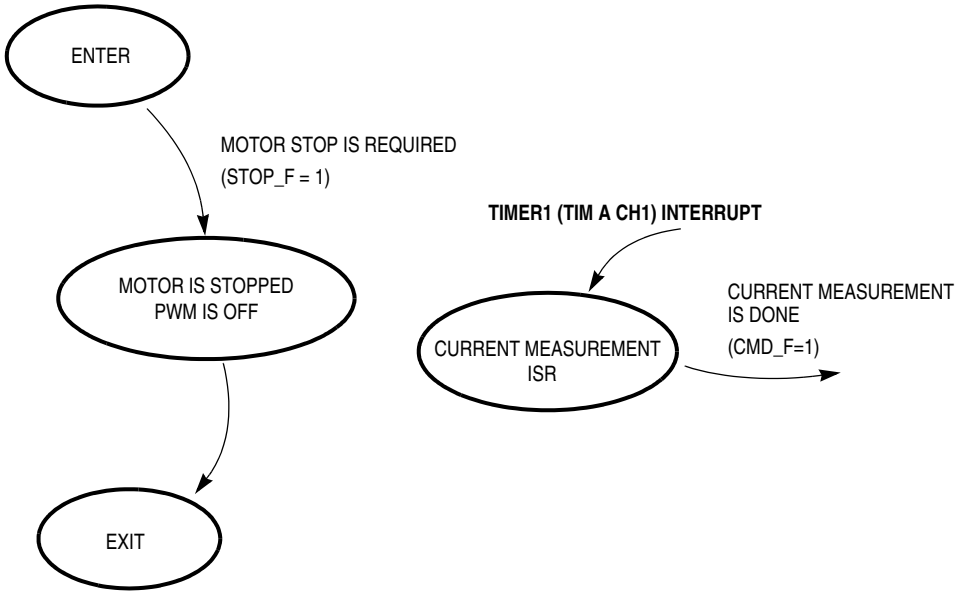

Figure 5-12. STOP State

5.5.8 Fault State

If over-voltage, over-current, or commutation fault occurs, the motor control PWM's are disabled and control enters the fault state, where it remains until RESET or clear failure (start switch set OFF or PCMode&ClearFail). The fault state is indicated by the Red LED diode blinking.

Clear Failure Test

The failure (Failure = 0) is tested. In manual mode (PCMode = 0), the switch in the stop position clears the failure. In PC master software mode (PCMode = 1), the failure is cleared by ClearFail flag or the switch in the stop position. If start condition is valid Strt_F is set, Stop_F is cleared, and the next state entered.

See **Figure 5-13**.

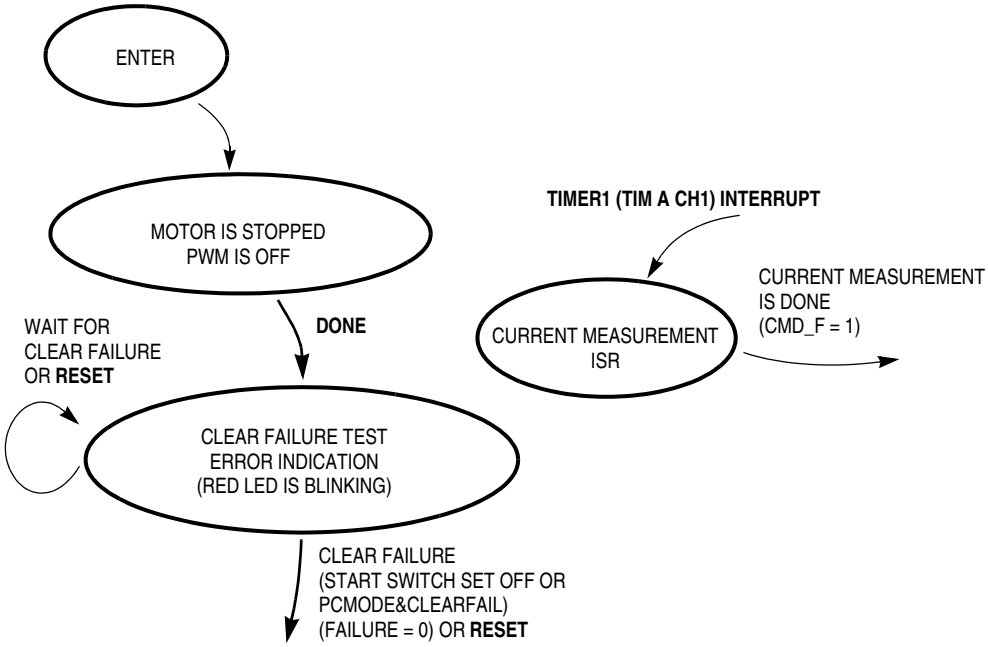

Figure 5-13. Fault State

5.6 Implementation Notes

5.6.1 Software Files

The software files and structure are described in section ***6.4.2.1 Application HC08 Software Files***.

5.6.2 BLDC Commutation and Zero Crossing Selection

The required BLDC motor voltage system commutation is provided using the MC68HC08MR32 PWM block.

The zero crossing selection is provided by setting port F pins PTF1–PTF3 connected to the multiplexer.

As shown in **Data Flow**, the commutation and back-EMF zero-crossing selection process is split into two actions:

1. Preset BLDC commutation and BEMF zero-crossing selection
2. Set BLDC commutation and BEMF zero-crossing selection

5.6.2.1 Preset BLDC Commutation and BEMF Zero Crossing Selection

In each phase of the 6-step commutation two PWM channels (bottom and top switch) are active and the other four PWM channels are logical 0. The commutation preset is accomplished by setting the buffered registers PVAL1H, PVAL1L through PVAL6H, PVAL6L. To preset the active PWM channel, the MSB bits of the dedicated PVALxH registers are set to 0.To preset the PWM channel to the logical 0, the MSB bits of the dedicated PVALxH registers are set to 1. This is due to the signature bit functionality as described in *68HC908MR32, 68HC9908MR16 Advance Information* (Motorola document order number MC68HC908MR32/D) .This commutation preset is provided by the function **Commut ()**. In **Commut()**, the function field **Set_PWM(P_Step_Cmt] ()** calls one of **Shft_PWM0..5()** functions according to **P_Step_Cmt** value, so the PVALxH registers are set as required. This will not effect the PWM signals until **LDOK** bit is set. Also, the back-EMF zero crossing selection preset is provided in this function, by setting **V_MUX** variable according to **P_Step_Cmt** value.

5.6.2.2 Set BLDC Commutation and BEMF Zero Crossing Selection

The commutation set is provided in timer 2 interrupt function **TIMACh3_Int()** in the **code_isr.c** file. The **PWMEN** is toggled and **LDOK** bit set, in order to immediately restart the PWM generator and reload the PWM with the buffered PVAL1H, PVAL1L through PVAL6H,

PVAL6L registers. Then the back-EMF zero crossing selection set is provided by setting PORTF according to **V_MUX** variable.

5.6.3 BLDC Speed Control and Calculation

Desired speed calculation and PWM duty cycle setting is quite simple, but there are some C language syntax tricks. Also, the scaling aspect needs to be taken into consideration.

5.6.3.1 Desired Speed Calculation

The 8 bit value **Speed_Desired** is calculated using 8-bit multiplication of two 8-bit variables, **Sp_Input** and **Coef_Speed_Inp**. The syntax is:

(unsigned char)((**Sp_Input***Coef_Speed_Inp**)>>8)

This syntax is used to generate optimal code using the MUL instruction.

5.6.3.2 PWM Duty Cycle

PWM duty cycle is set for all six PWM channels according to regulators output **OutReg_U8**. The maximal duty cycle is at **OutReg_U8** = 255. The registers **PVAL1H, PVAL1L** through PVAL6H, PVAL6L are set proportionally to the PWM modulus register **PMOD** = **MCPWM_MODULUS** constant (100% duty cycle is when **PVALx** = **PMOD**).

The **PWM_Val_Max** variable is:

PWM_Val_Max = **DUTY_PWM_MAX***MCPWM_MODULUS**

This variable is used for scaling of the regulator output **OutReg_U8**. The registers PVAL1H, PVAL1L through PVAL6H, PVAL6L are loaded with **PWM_Val** calculated from **OutReg_U8**:

PWM_Val = **PWM_Val_Max***OutReg_U8/256**

This calculation is provided with macro **umul_16x8_macro**.

5.6.4 Timers

Timer 1 and timer 2 are implemented using MC68HC08MR32 timers. Timer 3 is a software virtual timer using time base of timer 1.

5.6.4.1 Timer 1

Timer 1 is provided by timer A channel 1 set in output compare mode. In this mode the registers TACH1H and TACH1L are used for setting the output compare value, T1.

- TACNTH and TACNTL form a 16-bit timer A counter with an infinite counting 16-bit roll over.
- The timer A channel 1 interrupt is called whenever TACH1H = TACNTH and TACH1L = TACNTL. With each interrupt, the registers TACH1H and TACH1L are loaded with the new value, **T1 = T1+PER_CS_T1**, where **T1+PER_CS_T1** is a 16-bit addition with no saturation. So, the constant interrupt period **PER_CS_T1** of the timer T1 interrupts is generated.
- The timer unit **UNIT_PERIOD_T1_US** of timer A is determined by the MCU bus frequency (8 MHz with a 4-MHz oscillator and default software setting) and timer prescaler set in TASC. So, the default software value is 2 μs. The timer 1 interrupt function is provided by the **TIMACh1_Int ()** function.

5.6.4.2 Timer 2

Timer 2 is provided by timer A channel 3 set in output compare mode. In this mode, the registers TACH3H and TACH3L are used for setting the output compare value, T2.

- TACNTH and TACNTL form a 16-bit timer A counter, with infinite counting 16-bit roll over.
- The actual time is sensed from TACNTH and TACNTL base (e.g., time of the commutation, **T_Cmt**).
- The timer A channel 3 (timer 2) interrupt is called whenever TACH3H = TACNTH and TACH3L = TACNTL. So, the registers TACH3H and TACH3L are loaded with the T2 variable value.

- The value T2 (e.g., **T2 = T_Cmt + Per_HlfCmt**) is calculated using a 16-bit addition with no saturation. So, the time duration up to 65,536 **UNIT_PERIOD_T2_US** from actual time (TACNTH, TACNTL) can be timed at any TACNTH, TACNTL timer A counter value.
- The timer unit **UNIT_PERIOD_T2_US** of timer A is determined by the MCU bus frequency (8 MHz with 4-MHz oscillator and default software setting) and timer prescaler set in TASC. So, the default software value is 2 μs. The timer 2 interrupt function is provided by function **TIMACh3_Int()**.

Software Design

Section 6. User Guide

6.1 Contents

6.2 Application Suitability Guide

This application suitability guide deals with issues which may be encountered when tailoring application using customer motor.

6.2.1 Minimal Application Speed

As it is known, the back-EMF voltage is proportionally dependent on motor speed. Since the sensorless back-EMF zero crossing sensing technique is based on back-EMF voltage, it has some minimal speed limitations! The motor start-up is solved by starting (back-EMF acquisition) state, but minimal operation speed is limited.

The minimal speed depends on many factors of the motor and hardware design, and differs for any application. This is because the back-EMF zero crossing is disturbed and effected by the zero crossing comparator threshold as explained below and in the sections **6.2.4.2 Effect of Mutual Inductance** and **6.2.4.1 Effect of Mutual Phase Capacitance** .

NOTE: *Usually, the minimal speed for reliable operation is from 7% to 20% of the motor's nominal speed.*

6.2.2 Maximal Application Speed

The maximal motor speed is limited by the minimal commutation period:

$$\text{maximal speed[rpm]} = \frac{60(10^6)}{\text{min. commutation period [us]*COMMUT\_REV}} \quad \textbf{(EQ 6-1.)}$$

COMMUT_REV — commutations per motor revolution, must be set according to rotor poles:

$$\text{COMMUT\_REV} = \frac{6p}{2} \quad \textbf{(EQ 6-2.)}$$

where: p = rotor poles

The minimal commutation period is determined by execution time of the software. With default software settings and COEF_HLFCMT = 0.450 it is 333 µs, as shown in **Table 2-1**. So for example, the 4 pole (3-phase) motor can be controlled up to the maximal speed of 15,015 rpm.

NOTE: *Using PC master software in the application increases the minimal commutation time. This is due to the execution of PC master software routine. In this case, the minimal execution time is 520 ms. The minimal commutation period could be decreased using pure assembler code instead of C coding.*

6.2.3 Voltage Closed Loop

As shown in **Application Hardware and Software Configuration**, the speed control is based on voltage closed loop control. This should be sufficient for most applications.

6.2.4 Motor Suitability

Back-EMF zero crossing sensing is achievable for most of BLDC motors with a trapezoidal back-EMF. However, for some BLDC motors the back-EMF zero crossing sensing can be problematic since it is affected by unbalanced mutual phase capacitance and inductance. It can disqualify some motors from using sensorless techniques based on the back-EMF sensing.

6.2.4.1 Effect of Mutual Phase Capacitance

The effect of the mutual phase capacitances can play an important role in the back-EMF sensing. Usually the mutual capacitance is very small. Its influence is only significant during the PWM switching when the system experiences very high *du/dt*.The effect of mutual capacitance is described in section **3.2.4.4 Effect of Mutual Phase Capacitance**.

NOTE: *Note that the configuration of the end-turns of the phase windings has a significant impact. Therefore, it must be properly managed to preserve the balance of the mutual capacity. This is especially important for prototype motors that are usually hand-wound.*

CAUTION: *Failing to maintain balance of the mutual capacitance can easily disqualify such motors from using sensorless techniques based on the back-EMF sensing. Usually the BLDC motors with windings wound on separate poles show minor presence of the mutual capacitance. Thus, the disturbance is insignificant.*

6.2.4.2 Effect of Mutual Inductance

The negative effect on back-EMF sensing of mutual inductance, is not to such a degree as unbalanced mutual capacitance. However, it can be noticed on the sensed phase. The difference of the mutual inductances between the coils which carry the phase current and the coil used for back-EMF sensing, causes the PWM pulses to be superimposed onto the detected back-EMF voltage.

The effect of mutual inductance is described in section **3.2.4.3 Effect of Mutual Inductance**.

NOTE: *The BLDC motor with stator windings distributed in the slots has technically higher mutual inductances than other types. Therefore, this effect is more significant. On the other hand, the BLDC motor with windings wound on separate poles, shows minor presence of the effect of mutual inductance.*

CAUTION: *However noticeable this effect, it does not degrade the back-EMF zero crossing detection, because it is cancelled at the zero crossing point. Additional simple filtering helps to reduce ripples further.*

6.3 Warning

This application operates in an environment that includes dangerous voltages and rotating machinery.

Be aware that the application power stage and optoisolation board are not electrically isolated from the mains voltage - they are live with risk of electric shock when touched.

An isolation transformer should be used when operating off an ac power line. If an isolation transformer is not used, power stage grounds and oscilloscope grounds are at different potentials, unless the oscilloscope is floating. Note that probe grounds and, therefore, the case of a floated oscilloscope are subjected to dangerous voltages.

The user should be aware that:

- Before moving scope probes, making connections, etc., it is generally advisable to power down the high-voltage supply.
- To avoid inadvertently touching live parts, use plastic covers.
- When high voltage is applied, using only one hand for operating the test setup minimizes the possibility of electrical shock.
- Operation in lab setups that have grounded tables and/or chairs should be avoided.
- Wearing safety glasses, avoiding ties and jewelry, using shields, and operation by personnel trained in high-voltage lab techniques are also advisable.
- Power transistors, the PFC coil, and the motor can reach temperatures hot enough to cause burns.

When powering down; due to storage in the bus capacitors, dangerous voltages are present until the power-on LED is off.

6.4 Application Hardware and Software Configuration

6.4.1 Hardware Configuration

As mentioned, the software can be configured to run on one of the three hardware and motor platforms:

- **High-Voltage Hardware Set Configuration**
- **Low-Voltage Evaluation Motor Hardware Set Configuration**
- **Low-Voltage Hardware Set Configuration**

The hardware setups are shown in **Figure 6-1**, **Figure 6-2**, and **Figure 6-3**. These setups are described in following subsections (see **Hardware and Drive Specifications** for each platform characteristics).

The supplied controller boards for MC68HC908MR32 (ECCTRMR32) allows two possibilities for software execution:

1. MMDS Evaluation Board (KITMMDSMR32)
   Using a real-time debugger (supplied with the Metrowerks compiler) the evaluation board is connected to the controller board (ECCTRMR32) via an emulator connector. This solution is recommended for software evaluation.
2. Programmed MCU (MC68HC908MR32)

   Where a daughter board module, with MC68HC908MR32 processor, is plugged into the controller board (ECCTRMR32) instead of the emulator cable, the processor must be programmed in an external programmer. This solution is recommended for final tests.

Figure 6-1, **Figure 6-2**, and **Figure 6-3** show the configuration with MMDS evaluation board.

NOTE: *All the system parts are described in **Section 4. Hardware Design**. They can be ordered as a standard products. For the the supply and in detail document references see **Section 4. Hardware Design**.*

6.4.1.1 High-Voltage Hardware Set Configuration

The system configuration for a high-voltage hardware set is shown in **Figure 6-1**

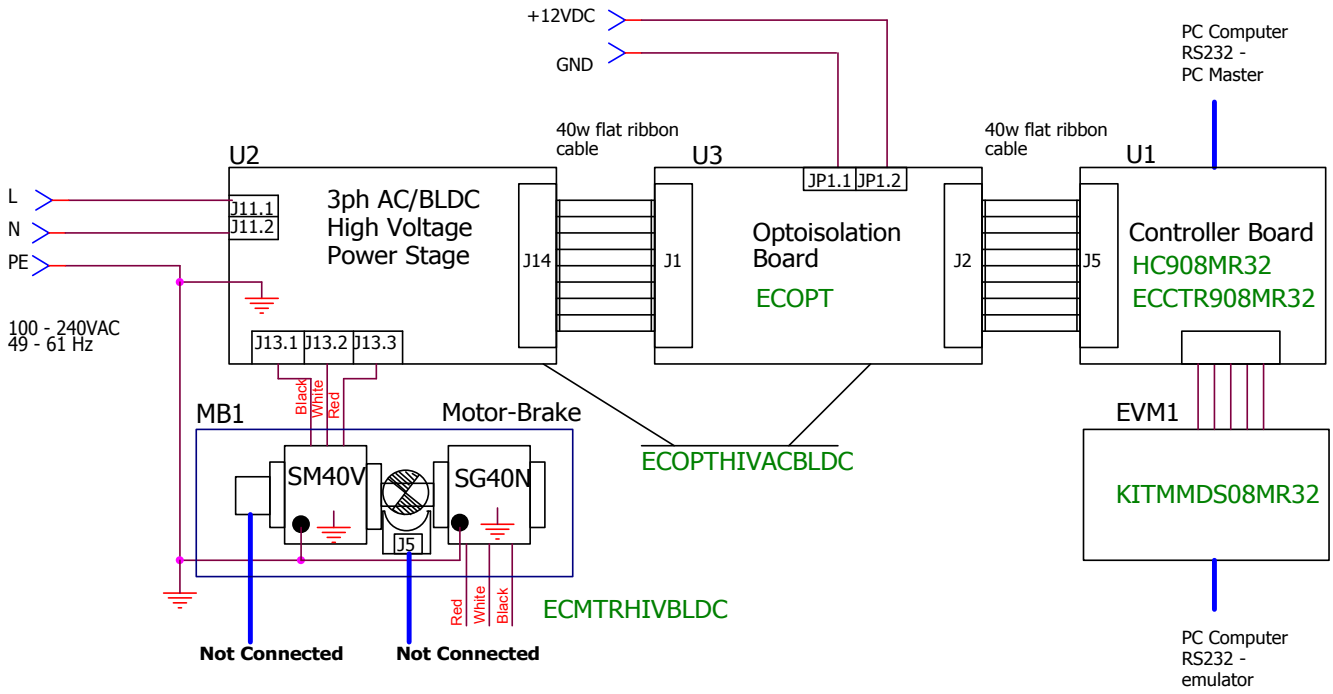

Figure 6-1. High-Voltage Hardware System Configuration

NOTE: *It is strongly recommended to use opto-isolation (optocouplers and optoisolation amplifiers) during development time to avoid any damage to the development equipment.*

NOTE: *All the system parts are described in **Section 4. Hardware Design**. They can be ordered as a standard products. For the the supply and in detail document references shown see section **4.2.1 High-Voltage Hardware Set Configuration**.*

6.4.1.2 Low-Voltage Evaluation Motor Hardware Set Configuration

The system configuration for a low-voltage evaluation motor hardware set is shown in **Figure 6-2**.

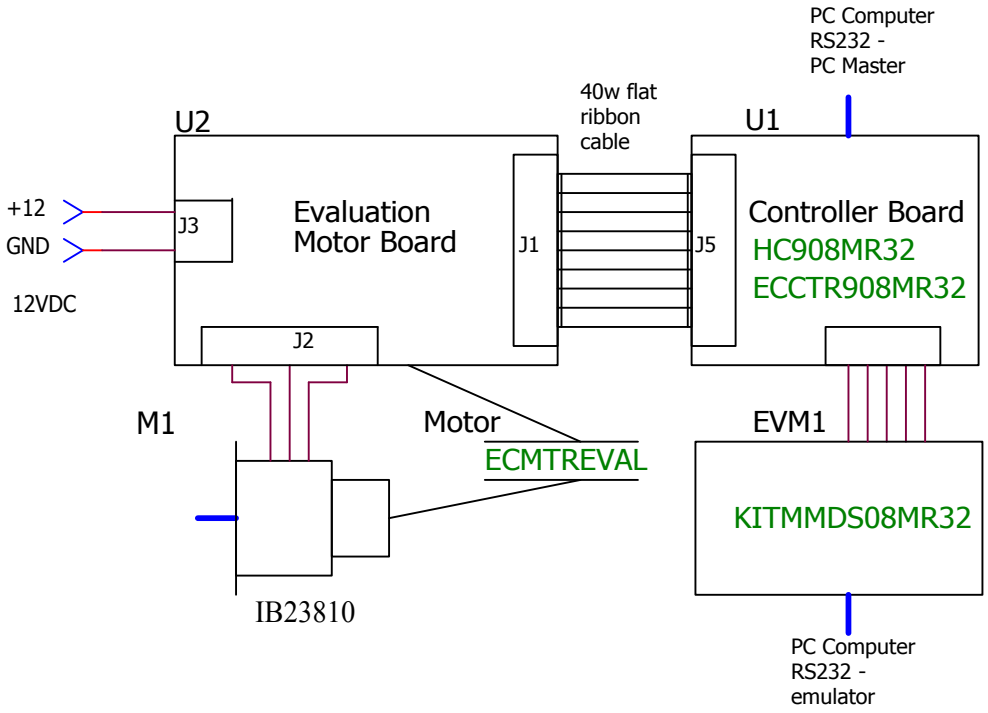

Figure 6-2. Low-Voltage Evaluation Motor Hardware System Configuration

NOTE: *All the system parts are described in section **Section 4. Hardware Design**. They can be ordered as a standard products. For the the supply and in detail document references shown see section **4.2.2 Low-Voltage Evaluation Motor Hardware Set Configuration**.*

6.4.1.3 Low-Voltage Hardware Set Configuration

The system configuration for low-voltage hardware set is shown in **Figure 6-3**.

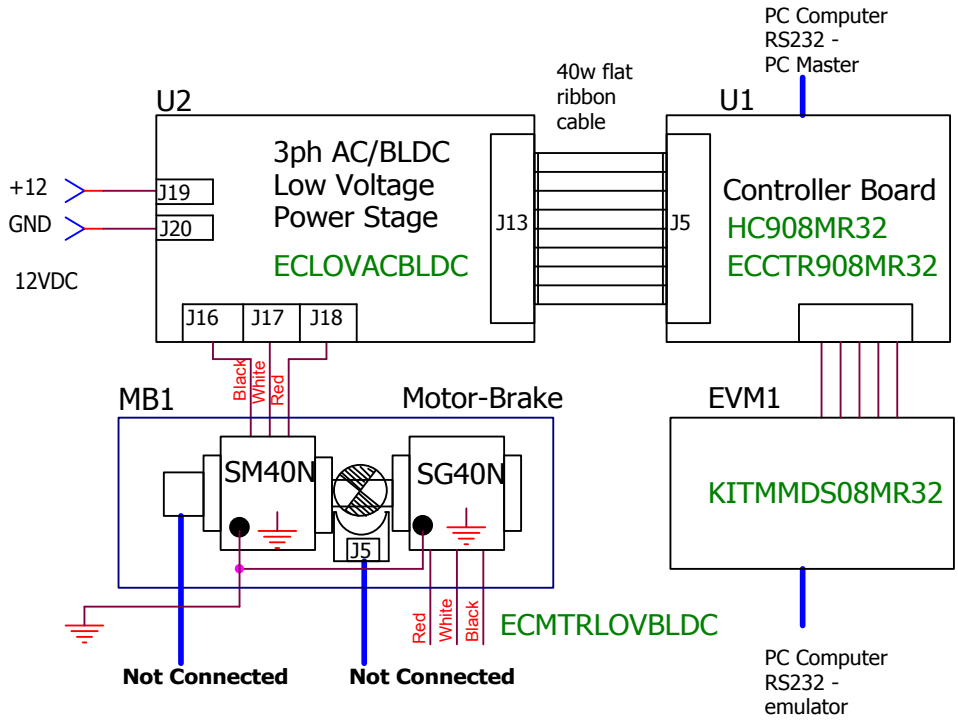

Figure 6-3. Low-Voltage Hardware System Configuration

NOTE: *All the system parts are described in section **Section 4. Hardware Design**. They can be ordered as a standard products. For the the supply and in detail document references shown see section **4.2.3 Low-Voltage Hardware Set Configuration**.*

6.4.1.4 Controller Board Settings

Controller board settings are the same for all hardware platforms.

Jumpers JP3 and JP7 must be connected with the other jumpers disconnected. See **Figure 6-4**.

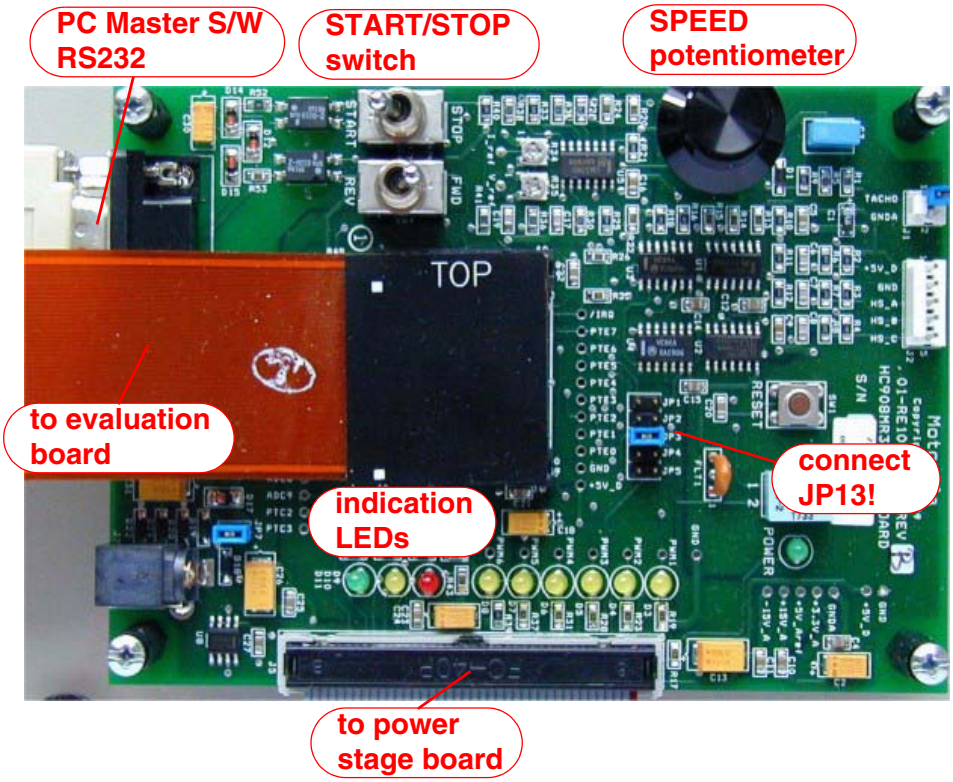

Figure 6-4. Controller Board

6.4.1.5 EVM Board Settings

EVM board settings are the same for all hardware platforms.

CAUTION: *Remember, the MMDS MCU clock must be set to 4 MHz. Change the crystal oscillator, or set oscillator W1 to MMDS, and set for 4 MHz in the debugger!*

6.4.2 Software Setup

In order to run the sensorless BLDC application the following software is needed:

- Metrowerks compiler for HC08 — installed on your PC computer
- Sensorless BLDC application HC08 software files (located in bldc_zerocros08MR32 directory)

For application PC master software (remote) control, the following software is needed:

- PC master software for PC — installed on your PC computer
- Sensorless BLDC application PC master software control files (located in bldc_zerocros08MR32\pc_master directory)

Both the HC08 and PC master software control files for the sensorless BLDC application are delivered together in the **bldc_zerocros08MR32** directory. It consists of files listed in **Application HC08 Software Files**.

6.4.2.1 Application HC08 Software Files

The application HC08 software files are:

- **...\bldc_zerocros08MR32\bldc_zerocross.mcp**, application project file
- **...\bldc_zerocros08MR32\sources\const_cust_hv.h**, definitions for software customizing for high voltage (230/115 Vac) power board
- **...\bldc_zerocros08MR32\sources\const_cust_evm.h**, definitions for software customizing for EVM motor board (12 V low power)
- **...\bldc_zerocros08MR32\sources\const_cust_lv.h**, definitions for software customizing for low voltage (12 Vdc) power board

NOTE: *Change the code_fun.c file to include (#include) one of const_cust_hv.h, const_cust_evmm.h, or const_cust_lv.h files according to the hardware platform used! See **Table 6-1**.*

Table 6-1. Required Software Configuration for Dedicated Hardware Platform

Hardware Platform	Dedicated Customizing File	Required Software Configuration
High-voltage hardware	const_cust_hv.h	#include const_cust_hv.h into code.fun.c (done with default software setting)

Table 6-1. Required Software Configuration for Dedicated Hardware Platform

Hardware Platform	Dedicated Customizing File	Required Software Configuration
Low-voltage evaluation motor hardware	const_cust_evmm.h	#include const_cust_evmm.h into code.fun.c
Low-voltage hardware	const_cust_lv.h	#include const_cust_lv.h into code.fun.c

- **...\bldc_zerocros08MR32\sources\code_fun.c**, program C language functions
- **...\bldc_zerocros08MR32\sources\code_fun.h**, program C language functions header
- **...\bldc_zerocros08MR32\sources\const.h**, main program definitions
- **...\bldc_zerocros08MR32\sources\mr32io.h**, MC68HC908MR32 registers definitions file
- **...\bldc_zerocros08MR32\sources\mr32_bit.h**, MCHC908MR32 register bits definitions file
- **...\bldc_zerocros08MR32\sources\bldc08.c**, main program
- **...\bldc_zerocros08MR32\sources\code_start.c**, motor alignment and starting (back-EMF acquisition) state functions
- **...\bldc_zerocros08MR32\sources\code_start.h**, motor alignment and starting (back-EMF acquisition) state function header
- **...\bldc_zerocros08MR32\sources\code_run.c**, motor running state function
- **...\bldc_zerocros08MR32\sources\code_run.h**, motor running state function header
- **...\bldc_zerocros08MR32\sources\code_isr.c**, program interrupt functions
- **...\bldc_zerocros08MR32\sources\code_isr.h**, program interrupt functions header

- **...\bldc_zerocros08MR32\sources\ram.c**, general RAM definitions
- **...\bldc_zerocros08MR32\sources\ram.h**, general RAM declarations header
- **...\bldc_zerocros08MR32\sources\ram_bit.h**, general RAM bits definitions header
- **...\bldc_zerocros08MR32\sources\ram_cust_param.c**, RAM variables for software customizing definitions
- **...\bldc_zerocros08MR32\sources\ram_cust_param.h**, RAM variables for software customizing header declarations
- **...\bldc_zerocros08MR32\sources\tab_cust.c**, constants/tables definitions
- **...\bldc_zerocros08MR32\sources\tab_cust.h**, constants/tables definitions header
- **...\bldc_zerocros08MR32\sources\pcmaster.c**, PC master software communication subroutines
- **...\bldc_zerocros08MR32\sources\pcmaster.h**, PC master software communication subroutines header
- **...\bldc_zerocros08MR32\sources\code_asm.asm**, program assembler functions
- **...\bldc_zerocros08MR32\sources\code_asm.h**, program assembler functions header
- **...\bldc_zerocros08MR32\prms\default.prm**, linker command file

6.4.2.2 Application PC Master Software Control Files

The application PC master software control files are:

- **...\bldc_zerocros08MR32\pc_master\BLDC.pmp**, PC master software project file
- **...\bldc_zerocros08MR32\pc_master\source**, directory with PC master software control page files

6.4.3 Software Execution

6.4.3.1 Build

To build the BLDC sensorless with the back-EMF zero crossing application, open the **bldc_zerocross.mcp** project file and execute the *Make* command, as shown in **Figure 6-5**. This will build and link the application and all needed Metrowerks libraries.

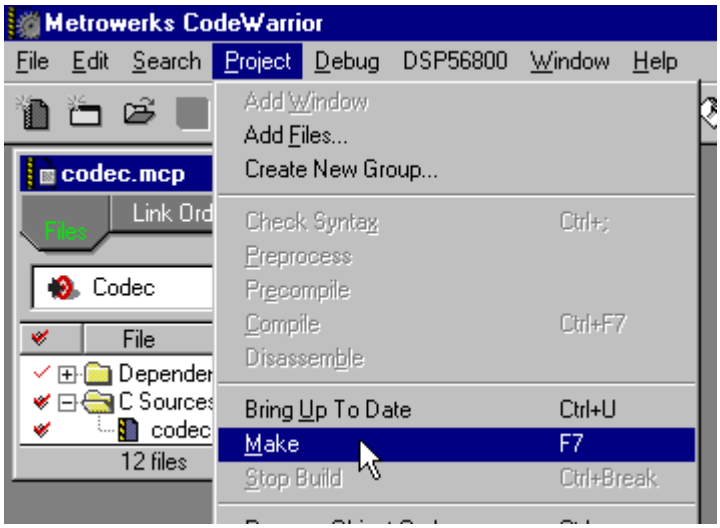

Figure 6-5. Execute Make Command

6.4.3.2 Execute from Evaluation Board

To execute the application from MMDS evaluation board (KITMMDSMR32), choose the *Project/Debug* command in the Code Warrior IDE. This will start real-time debugger, load firmware, and application software to evaluation board MMDS.

The application should then be started from the real-time debugger IDE by the *Run/Start* command. For more help with these commands, refer to the CodeWarrior tutorial documentation located in the CodeWarrior installation directory.

NOTE: *Remember, the MMDS MCU clock must be set to 4 MHz. Change the crystal oscillator, or set oscillator W1 to MMDS, and set 4 MHz in the real-time debugger — MMDS0508/target signals/4 MHz!*

Once the application is running, move the RUN/STOP switch to the RUN position and set the required speed with the SPEED potentiometer. If successful, the BLDC motor will be spinning.

NOTE: *If the RUN/STOP switch is set to the RUN position when the application starts, toggle the RUN/STOP switch between the STOP and RUN positions to enable motor spinning. This is a protection feature that prevents the motor from starting when the application is executed from CodeWarrior.*

6.4.3.3 Execute from Pre-programmed MCU

When the software is built, the S-record file **bldc_zerocros08mr32_MMDS.sx** is generated in:

...\bldc_zerocros08MR32\bin\bldc_zerocros08mr32_MMDS.sx

NOTE: *The software must be built (see **Build**) to generate by linker the **bldc_zerocros08mr32_MMDS.sx** file (last update)*

This S-record file can be used for programming of MC68HC908MR32 MCU devices. An external programmer (e.g., Motorola M68HC08 serial programmer) must be used to program the device!

The programmed MCU, with MC68HC908MR32 daughter board module, can then be plugged into the controller board (ECCTRMR32) instead of the emulator cable.

6.4.4 Application Control

This BLDC sensorless motor control application can operate in two modes:

1. **Manual Operating Mode**
2. **PC Master Software (Remote) Operating Mode**

6.4.4.1 Manual Operating Mode

In the manual operating mode, the drive is controlled by the RUN/STOP switch and the required speed is set by the SPEED potentiometer. The RUN/STOP switch enables/disables motor spinning. The yellow LED will light whenever the application software correctly executes (so, it will also light when motor spinning is disabled or at a fault state).

- When motor spinning is enabled and starts spinning (alignment or starting/back-EMF acquisition state), only the yellow LED lights.
- When motor rotation is enabled and the motor runs with speed closed loop (running state), the green LED lights (yellow LED also lights).
- If an over-current or over-voltage fault occurs, the internal fault logic is asserted and the application enters a fault state indicated by a red LED blinking (yellow LED lights).This state can be exited only by an application RESET or setting RUN/STOP switch to STOP.

NOTE: *It is strongly recommended that you inspect the entire application to locate the source of the fault, before starting again.*

6.4.4.2 PC Master Software (Remote) Operating Mode

In the PC master software (remote) operating mode, the drive is controlled remotely from a PC through the serial communication interface (SCI) channel of the MCU device via an RS-232 physical interface. The drive is enabled by the RUN/STOP switch, which can be used to safely stop the application at any time.

For the PC master software (remote) control it is necessary to have PC master software installed on your PC computer!

Start the PC master software application:

...\bldc_zerocros08MR32\pc_master\BLDC.pmp

After you start the PC master software, press “control page” to make the control window visible! **Figure 6-6** illustrates the PC master software Control Window.

NOTE: *After you start the PC master software, the algorithm block description window appears instead of the PC master control window; therefore, press "control page". If the PC master software project (**..pmp** file) is unable to control the application, it is possible that the wrong load map (**..\bin\bldc_zerocros08mr32_MMDS.map** file) has been selected. PC master software uses the load map to determine addresses for global variables being monitored. Once the PC master project has been launched, this option may be selected in the PC master window under Project/Select Other Map FileReload.*

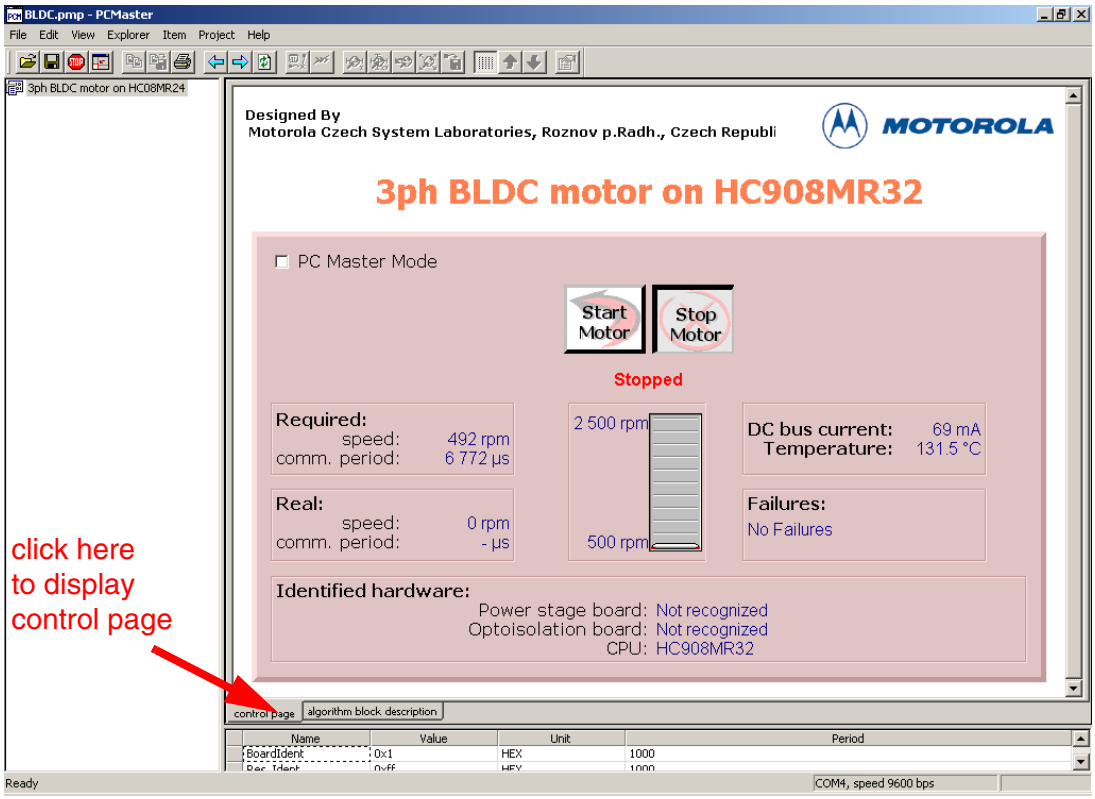

Figure 6-6. PC Master Software Control Window

The following control actions are supported:

- Setting PC master software/manual control mode (PC Master Mode Radio button)

NOTE: *Application control from PC master software requires that PC master software control mode must be set. Before changing PC master software/manual control mode (by PC Master Mode Radio button) the controller board START/STOP switch must be set to STOP. This is a protection feature that prevents the motor from unexpected starts!*

- Start the motor (Start Motor button)

NOTE: *To start the motor with PC master software control mode, two conditions must be fulfilled. START/STOP switch on the controller board must be set to START and Start Motor button on PC master software control page must be "pressed down".*

- Stop the motor (Stop Motor button)
- Set the required speed of the motor (via bar graph)
- Clear failures (Clear Failures button)

PC master software displays the following information:

- Required speed of the motor
- Actual speed of the motor
- dc-bus current
- Temperature of the power stage
- Fault status (no fault, over-voltage, under-voltage, or over-current in dc-bus)
- Motor status — running/stopped

NOTE: *Hardware board identification is not implemented in the software. Therefore, the PC master software control window displays Power Stage board, Optoisolation board: Not Recognized.*

If the fault status is different from the no faults (when over-current, over-voltage, or under-voltage fault), the red LED blinks and the motor is stopped. This state can be exited by application RESET or Clear Failures button on the PC master software control page.

NOTE: *It is strongly recommended that you inspect the entire application to locate the source of fault before starting it again.*

6.5 Tuning for Customer Motor

This section describes how to modify the software parameters for any BLDC motor and some hardware adaptations. The software parameters can be evaluated from a PC computer using PC master software, so the first subsection describes tuning the PC master software project file.

- A follow-up for software customizing to a customer motor is shown in **Figure 6-7**.
- Before starting the software modification for a customer motor and application, it is recommended that you check the application and motor suitability. This is explained in the section **6.2 Application Suitability Guide**.
- The **Parameters File Selection** must be made according to **Hardware Configuration** used.
- If a modified hardware power stage is used, the appropriate constants in const_cust_x.h file must be set as described in the section **6.5.3.1 Software Customizing to Power Stage**.
- If a low-voltage board with a modification for 42 V is used, the constants **VOLT_HW_MAX** and **VOLT_MAX_FAULT_V** must be changed.
- If one of the three standard power stages is used, the software customizing to power stage is not needed.
- For software customizing to customer motor and application, a setting must be made as explained in:
  - **6.5.4 Software Customizing to Motor — Voltage and Current Settings**
  - **6.5.4.3 Alignment Current and Current Regulator Setting**
  - **6.5.5 Software Customizing to Motor — Commutation and Start-up Control Setting**
  - **6.5.6 Software Customizing to Motor — Speed Control Setting**.

Accomplishing the above steps should be sufficient for most applications. However, in some cases there may be a need for advanced software customizing (see Figure 6-8.) with changes to motor PWM

frequency or to the current regulator sampling period as explained in the section **6.5.7 PWM Frequency and Current Sampling Period Setting**.

If there's still a problem running the motor, check if the motor is suitable for sensorless control with back-EMF zero crossing (see Figure 6-9.) as described in the section **6.2.4 Motor Suitability**. Then, again check the application suitability.

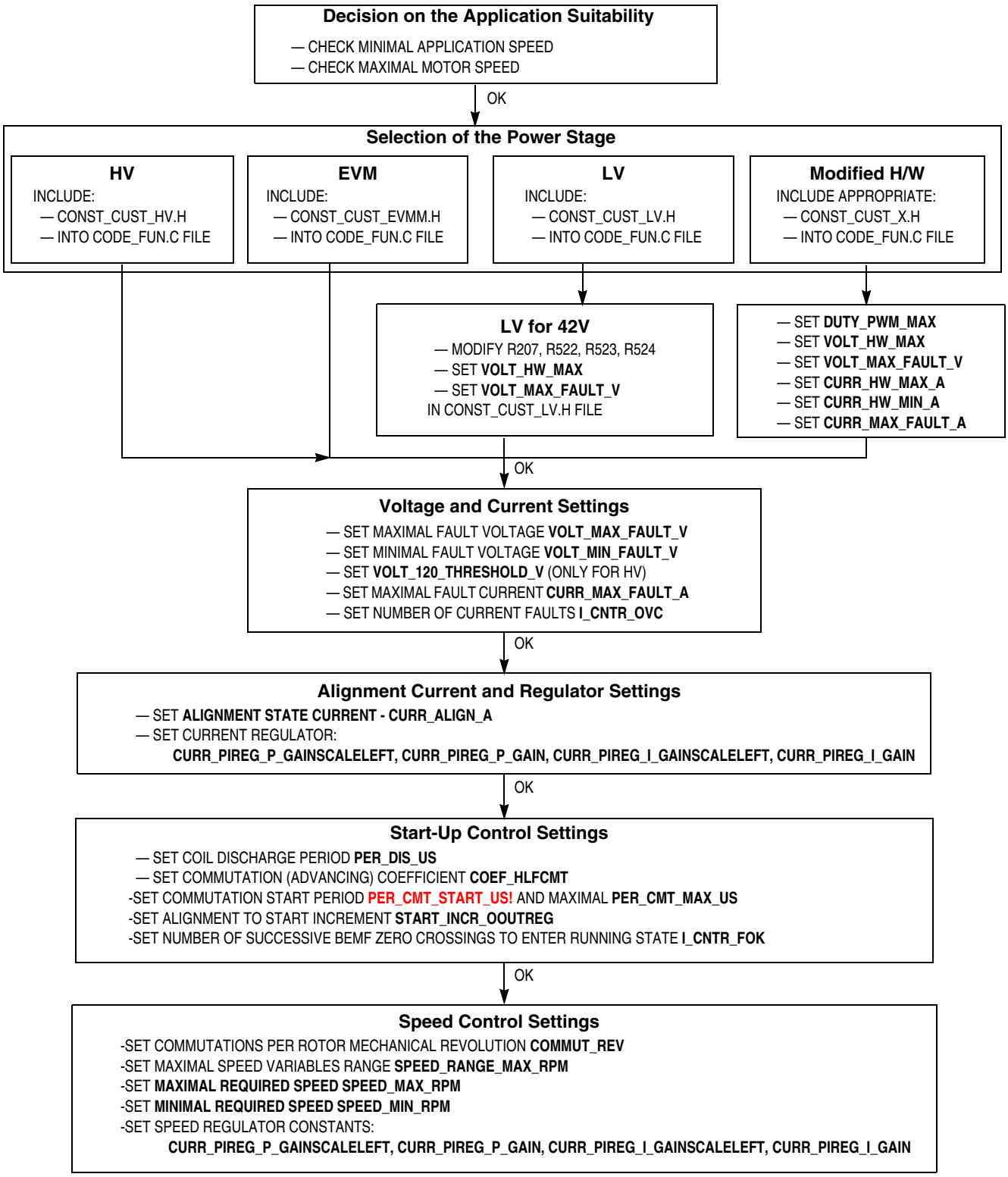

Figure 6-7. Follow-up for Software Customizing to Customer Motor

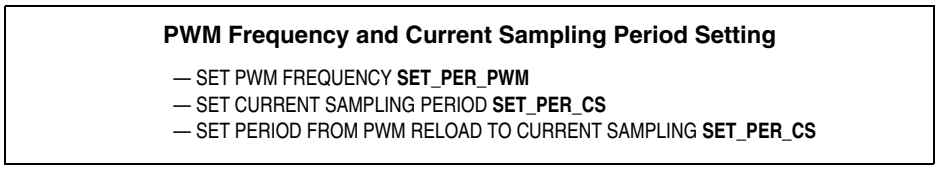

Figure 6-8. Follow-up for Advanced Software Customizing

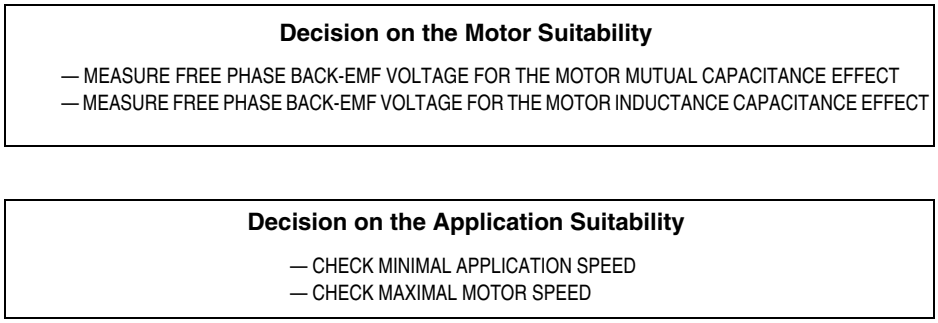

Figure 6-9. Follow-up for Software Customizing Trouble Shouting

6.5.1 Software Parameters Tuning with PC Master Software Project File

Sensorless BLDC software is provided with a PC master software project file for on-line software parameters tuning (see **Figure 6-10**). This file supports:

- Remote application control
- Key software parameters modification for:
  - Current parameters tuning
  - Start-up parameters tuning
  - Speed parameters tuning
- PC master software “oscilloscope” windows with required variables

The remote application control uses the same control page as described in **PC Master Software (Remote) Operating Mode**. Moreover, the tuning file incorporates subprojects for a dedicated system variables

setting, and PC master software “oscilloscope“ windows for watching dedicated parameters (variables).

NOTE: *For software parameter tuning with PC master software, it is necessary to have PC master software installed on your PC computer!*

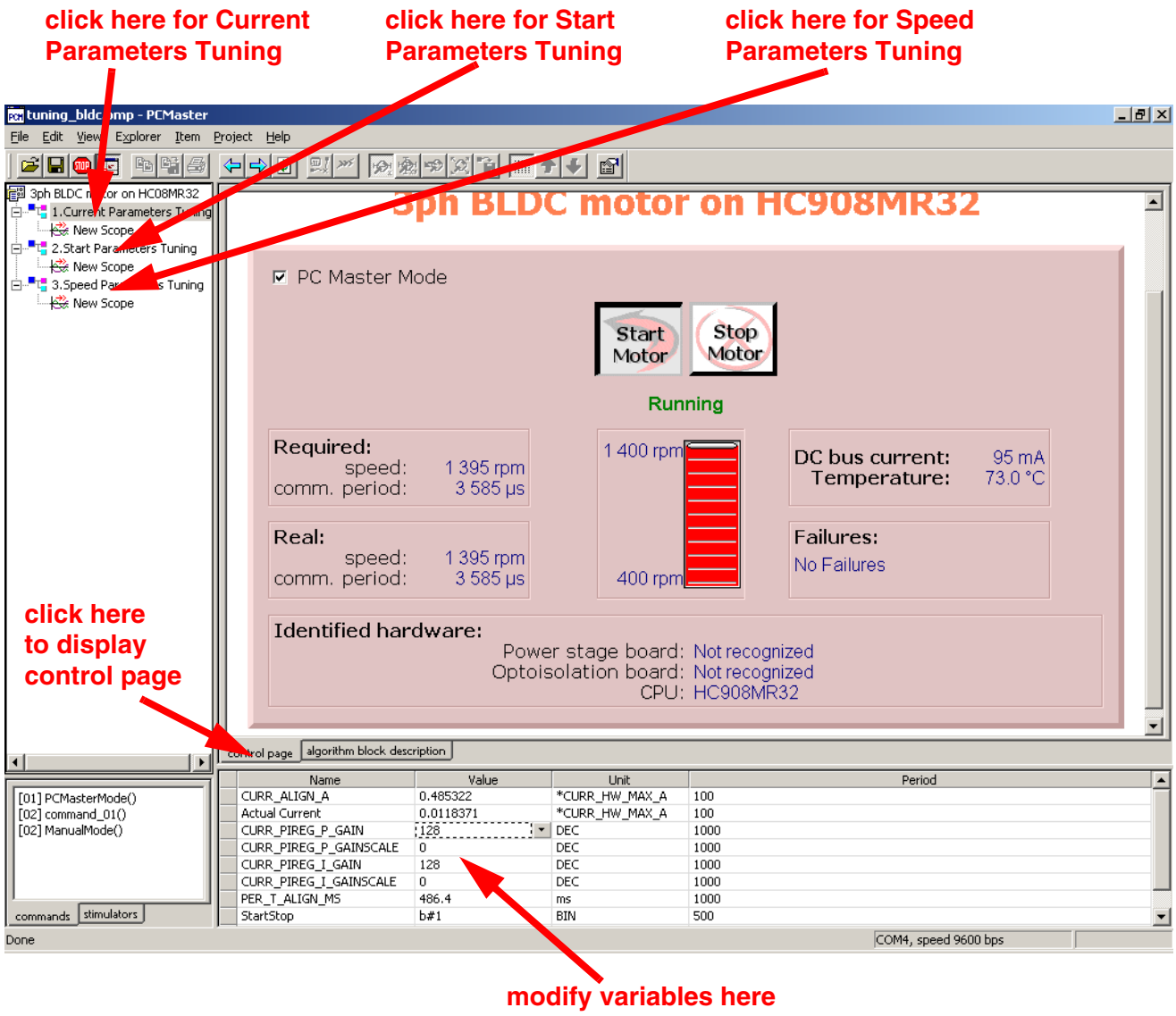

Figure 6-10. PC Master Software Parameters Tuning Control Window

Start the PC master software parameters tuning application:

...\bldc_zerocros08MR32\pc_master\tuning_bldc.pmp

After you start the PC master software, you can choose which parameters you are going to tune (current, start-up, speed parameters — see **Figure 6-10**). Then you can press “control page” to make the control window visible (and provide control in the same way as in **PC Master Software (Remote) Operating Mode**). Or, you can display the oscilloscope window (see **Figure 6-11**). You can then modify the variable values in the variable window (**Figure 6-11**), which is visible for both control page or oscilloscope page turned on. The variables can be modified according to their defined limits.

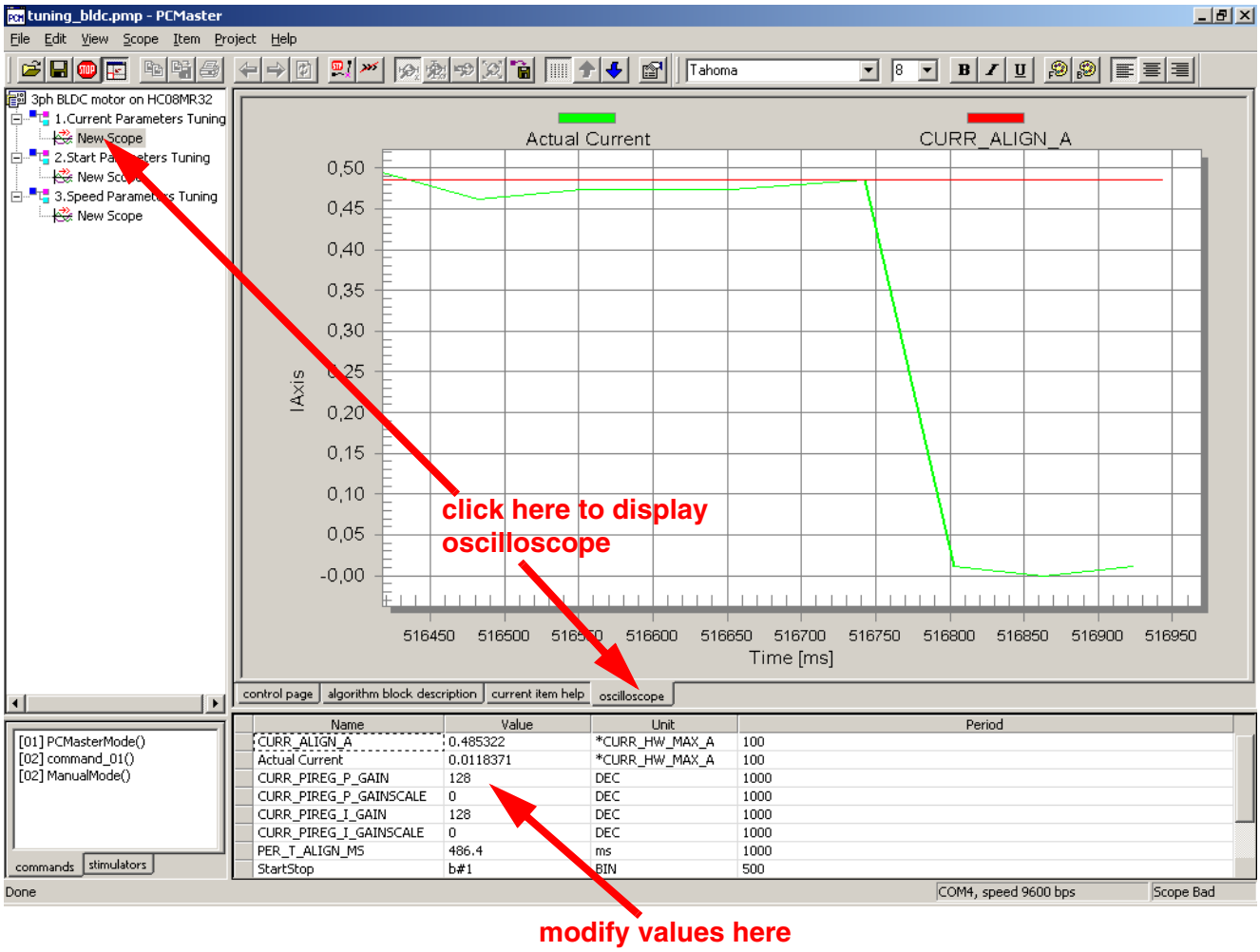

Figure 6-11. PC Master Software Parameters Tuning Control Window

NOTE: *The software parameters can be temporarily modified and evaluated using the PC master software tuning file. But, the parameter settings are not stored in the non-volatile memory (after reset the software loads parameters from const_cust_x.h file). When you finish the software parameters evaluation, you must open one of the* ***const_cust_hv.h****,* ***const_cust_evmm.h****, and* ***const_cust_lv.h*** *files and manually modify the parameters according to the final variable values evaluated (otherwise, you will get default setting after the MCU reset!).*

6.5.2 Software Parameters Setting Follow-up

The software is provided with three parameters sets (files const_cust_hv.h, const_cust_evmm.h, and const_cust_lv.h) configured for three hardware and motor kits (HV, LV, and EVM), as was described before. But, the software can be configured for other 3-phase trapezoidal BLDC motors (or possibly other hardware board parameters).

The motor control drive usually needs setting/tuning for:

- Current/voltage parameters
- Dynamic parameters

The parameter configurations must be set in source code before compilation. However, some parameters can also be temporarily changed using PC master software (experimental setting). Finally, when an appropriate parameter value is found, it can be set in the source code. The software parameters settings are described in the subsections below and in the software code by comments.

You should proceed with some steps to customize the software according to your motor (or hardware) characteristics. The source code is commented with descriptive labels to simplify the process.

6.5.2.1 Labels in the Files const_cust_hv.h, const_cust_evm.h, const_cust_lv.h

Most of the software parameter settings are provided in one of these files: const_cust_hv.h, const_cust_evmm.h or const_cust_lv.h. The required steps are marked:

/* MUST_CHANGE_nn: */

Label for changes which must be set (changed) when adapting software for a motor

/* MUST_CHANGE_nn_EXPER: */

Label for changes which must be set (changed) when adapting software for a motor — the setting can be done experimentally

/* MUST_IF_HW_CHANGE_nn */

Label for changes which must be set (changed) when a power stage board different from high voltage power board is used

/* CAN_CHANGE_nn */

Label for changes which can be set (changed) when adapting software for a motor, but usually the setting is not needed

/* CAN_CHANGE_nn_EXPER */

Label for changes which can be set (changed) when adapting software for a motor, but usually the setting is not needed — the setting can be done experimentally

6.5.2.2 Labels in the File const.h

The other parameters, like motor PWM frequency and current sampling period can be set in the file const.h. The required steps are marked:

/* CAN_CHANGE_FPWM_n */

Label for definitions which should be modified, when changing PWM frequency

/* CAN_CHANGE_PERCURSAMP_n */

lLabel for definitions which should be corrected, when changing current sampling period

Let's follow the next sections, or the labels in the source code to customize the software.

6.5.3 Parameters File Selection

As explained before (see **Software Setup**) one of the following files is used for most of software parameters configuration:

...\bldc_zerocros08MR32\sources\const_cust_hv.h, definitions for software customizing, for high-voltage (230/115 Vac) power board

...\bldc_zerocros08MR32\sources\const_cust_evm.h, definitions for software customizing for EVM motor board (12 V low power)

...\bldc_zerocros08MR32\sources\const_cust_lv.h, definitions for software customizing for low-voltage (12 Vdc) power board

According to hardware used, the designated file (see **Table 6-1**) must be selected by including:

...\bldc_zerocros08MR32\sources\code_fun.c, program c language functions

NOTE: *The following parameter settings will be provided in the selected file. Therefore, it will be referred to as* ***const_cost_x.h*** *in the following sections.*

6.5.3.1 Software Customizing to Power Stage

The hardware boards parameters customizing is provided in the **const_cust_x.h** file.

NOTE: *Skip this section, when standard modular motion control development hardware boards are used without any changes. When low-voltage power stage* ECLOVACBLDC *is configured for 42 V, some* ***const_cust_x.h*** *changes are needed.*

For setting, follow the labels **MUST_IF_HW_CHANGE_nn** in file ***const_cust_x.h*** from nn = 1. Detailed description starts here.

An example of software customizing to power stage is shown in **Example of Software Customizing to Hardware**.

6.5.3.2 Maximal PWM Duty Cycle

Maximal PWM duty cycle [-]:

```
/* MUST_IF_HW_CHANGE_1: */
#define DUTY_PWM_MAX 0.96
```

Range: <0,1>

Proportional value of maximal PWM duty cycle is determined by power stage boards used.

DUTY_PWM_MAX must be set for any hardware customizing. Some hardware boards need maximal duty cycle<1, in order to charge high side drivers for power inverters.

6.5.3.3 Voltage Setting Hardware Customizing

Maximal measurable voltage determined by hardware voltage sensing [V]:

```
/* MUST_IF_HW_CHANGE_2: */
#   define VOLT_HW_MAX  407.0
```

Range: <0,infinity)

VOLT_HW_MAX must be changed when voltage sensing range is different from default hardware.

Maximum limit of dc-bus voltage allowable for the hardware [V]:

```
/* MUST_IF_HW_CHANGE_3 */
#define VOLT_MAX_FAULT_V 380.0
```

Range: <0,VOLT_RANGE_MAX>

VOLT_MAX_FAULT_V determines the maximal voltage when the drive fault state should be entered. So the constant **VOLT_MAX_FAULT_V** must be set according to maximal voltage limit of the power stage or the motor, using the lower value. Therefore, setting this constant is also mentioned in **Maximal and Minimal Voltage Limits Setting** under a different label (**CAN_CHANGE_1**).

6.5.3.4 Current Setting Hardware Customizing

Maximal measurable current determined by hardware current sensing [A]:

```
/* MUST_IF_HW_CHANGE_4: */
#define CURR_HW_MAX_A  2.93
```

Range: <0,infinity)

Minimal measurable current determined by hardware current sensing [A]:

```
/* MUST_IF_HW_CHANGE_5: */
#define CURR_HW_MIN_A (-2.93)
```

Range: (-infinity,0>

CURR_HW_MAX_A and **CURR_HW_MIN_A** must be changed when current sensing range is different from default hardware.

Maximal limit of dc-bus current allowable for the hardware [A]:

```
/* MUST_IF_HW_CHANGE_6: */
#define CURR_MAX_FAULT_A 1.5
```

Range: <0,CURRENT_RANGE_MAX_A>

CURR_MAX_FAULT_A determines the maximal current when the drive fault state should be entered. So, it must be set to the maximum current allowed for the power stage or the motor (see also **Maximal and Minimal Current Limits Setting**)

6.5.3.5 Example of Software Customizing to Hardware

Let's have low-voltage power stage ECLOVACBLDC modified to 42 V (from 12 V) as described in its documentation. So, software must be customized for hardware changes. Because of low-voltage set, the **const_cust_lv.h** must be modified

1. Maximal PWM duty cycle remains the same:

   ```
   #define DUTY_PWM_MAX 0.942
   ```

2. Modified maximal measurable voltage is 55 V, so set:

   ```
   #define VOLT_HW_MAX 55.0
   ```

3. Maximum limit of dc-bus voltage should be set according to motor or application requirements, but VOLT_MAX_FAULT_V > 42V

   ```
   #define VOLT_MAX_FAULT_V 63.0
   ```

4. Board modified to 42 V has maximal measurable current unchanged

   ```
   #define CURR_HW_MAX_A  2.93
   #define CURR_HW_MIN_A (-2.93)
   ```

5. Maximum limit of dc-bus current should remain unchanged or set according to motor or application requirements:

   ```
   #define CURR_MAX_FAULT_A 45.0
   ```

When the software parameters are set for the hardware, you should follow the settings in **Software Customizing to Motor — Voltage and Current Settings**.

6.5.4 Software Customizing to Motor — Voltage and Current Settings

The software parameter settings according to customer motor are described in this section.

NOTE: *First of all, voltage and current settings need to be done. For settings which must be done, follow the labels **MUST_CHANGE_nn** and **MUST_CHANGE_EXPER_nn** in file **const_cust_x.h** where nn = 1.*

*For changes which can be done (but usually are not necessary), follow the labels **CAN_CHANGE_nn** and **CAN_CHANGE_EXPER_nn** in file **const_cust_x.h***

Detailed description starts here.

6.5.4.1 Maximal and Minimal Voltage Limits Setting

Most of voltage limit settings do not necessarily need to be done:

Maximal limit of dc-bus voltage [V]:

```
/* CAN_CHANGE_1: */
#define VOLT_MAX_FAULT_V 380.0
```

Range: <0,VOLT_RANGE_MAX>

VOLT_MAX_FAULT_V determines the maximal voltage when the drive fault state should be entered. So, the constants **VOLT_MAX_FAULT_V** must be set according to maximal voltage limit of the motor or the power stage, using the lower value. Therefore, the setting of this constants is also mentioned in **Voltage Setting Hardware Customizing** under a different label **MUST_IF_HW_CHANGE_3**. It should be changed when there are problems with over-voltage.

Minimal limit of dc-bus voltage [V]:

```
/* CAN_CHANGE_2_EXPER: */
#define VOLT_MIN_FAULT_V 100.0
```

Range: <0,VOLT_RANGE_MAX>

VOLT_MIN_FAULT_V determines the minimal voltage when the drive fault state should be entered. So the constants **VOLT_MIN_FAULT_V** must be set according to minimal voltage limits of the motor application. It should be changed when there are problems with under-voltage.

Dc-bus voltage threshold mains 120V/230V [V]:

```
/* CAN_CHANGE_10: */
#define VOLT_120_THRESHOLD_V 150
```

Range: <0,VOLT_RANGE_MAX>

120 V voltage threshold setting should only be used for high-voltage hardware. It determines if 120 or 230 V mains voltage will be detected by software. But the VOLT_120_THRESHOLD_V detection has no importance for the software functionality! For low-voltage hardware the VOLT_120_THRESHOLD_V should be set to 0.

6.5.4.2 Maximal and Minimal Current Limits Setting

Most of current limits settings do not necessarily need to be done.

Current offset limit for fault during calibration (initialization) [V]:

```
/* CAN_CHANGE_4: */
#define OFFSET_MAX_CURR_V (1.65+0.225)
```

Range: <0,5>

When PWM is off, the default hardware determined offset should be 1.65 V.The actual offset is checked during current calibration. The fault offset limit should be:

OFFSET_MAX_CURR_V = Default h/w offset + minimal allowed offset error **(EQ 6-3.)**

OFFSET_MAX_CURR_V should only be changed if there are over-current problems during current offset calibration (at MCU initialization).

Maximal limit of dc-bus current [A]:

```
/* CAN_CHANGE_3: */
#define CURR_MAX_FAULT_A 1.5
```

Range: <0,CURRENT_RANGE_MAX_A>

CURR_MAX_FAULT_A should be changed for a motor with maximal allowable current lower than the board.

Initial value for OVer-Current Counter [-]:

```
/* CAN_CHANGE_5: */
#define I_CNTR_OVC   0x04
```

Range: <0,255>

I_CNTR_OVC determines the number of current samples with current value > **CURR_MAX_FAULT_A** needed before entering the drive fault state. The current sampling period is **PER_CS_T1_US** = 128 µs at default software and PWM frequency setting. **I_CNTR_OVC** should normally not be changed. Lower value of **I_CNTR_OVC** secures a fast, safer over-current switch-off. High value of **I_CNTR_OVC** secures an unexpected over-current switch-off.

6.5.4.3 Alignment Current and Current Regulator Setting

The current during alignment state (before motor starts) [A]:

```
/* MUST_CHANGE_1_EXPER: */
#define CURR_ALIGN_A 0.55
```

Range: <0,CURRENT_RANGE_MAX_A>

It is recommended that nominal motor current value be set. Sometimes when power source is not able to deliver the required current, it is necessary to set a lower value then nominal motor current.

NOTE: *CURR_ALIGN_A can be evaluated with PC master software tuning file tuning_bldc.pmp.*

It might also be necessary to set the current PI regulator constants:

```
/* MUST_CHANGE_2_EXPER: */
#define CURR_PIREG_P_GAINSCALELEFT 0
```

Range: <0,8>

```
/* MUST_CHANGE_3_EXPER: */
#define CURR_PIREG_P_GAIN  128
```

Range: <0,255>

where the current regulator proportional gain is:

$$KP = CUR\_PIREG\_P\_GAIN * 2^{CURR\_OUREG\_P\_GAINSCALELEFT} \quad \text{(EQ 6-4.)}$$

```
/* MUST_CHANGE_4_EXPER: */
```

```
#define CURR_PIREG_I_GAINSCALELEFT 0
```

Range: <0,8>

```
/* MUST_CHANGE_5_EXPER: */
#define CURR_PIREG_I_GAIN  64
```

Range: <0,255>

where the current regulator integral gain is:

$$KP = CUR\_PIREG\_I\_GAIN * 2^{CURR\_PIREG\_I\_GAINSCALELEFT} \quad \textbf{(EQ 6-5.)}$$

These constants can be calculated according to regulators theory. The current sampling (regulator execution) period is **PER_CS_T1_US** = 128 µs, at the default software setting. Normally it does not need to be changed (if change is required see **PWM Frequency and Current Sampling Period Setting**). Another recommended solution is an experimental setting.

NOTE: *CURR_PIREG_P_GAINSCALELEFT, CURR_PIREG_P_GAIN, CURR_PIREG_I_GAINSCALELEFT, CURR_PIREG_I_GAIN can be evaluated with PC master software tuning file tuning_bldc.pmp.*

We suggest using PC master software with tuning file tuning_bldc.pmp for regulator parameters evaluation. You can use this procedure:

1. Set **const_cust_x.h**:

   CURR_PIREG_P_GAINSCALELEFT 0
   CURR_PIREG_P_GAIN 0
   CURR_PIREG_I_GAINSCALELEFT 0
   CURR_PIREG_I_GAIN 0
2. Temporarily change the software: in **code_start.c** file, label TUNING_1 enable goto Align (it will cause infinite time for alignment state, where the current is tuned)
3. Build and run the code (see **Software Execution**, **Build**, **Execute from Evaluation Board**)
4. Start the PC master software tuning project
5. Select Current Parameters Tuning subproject (see **Software Parameters Tuning with PC Master Software Project File**) in order to be able to modify the current regulator
6. You can see the actual current (and required alignment current)

on the Current Parameters Tuning\New Scope, or measure the powered motor coil current on real oscilloscope

7. Set PC master software control mode, and start the motor (see **Application Control** and **PC Master Software (Remote) Operating Mode**)

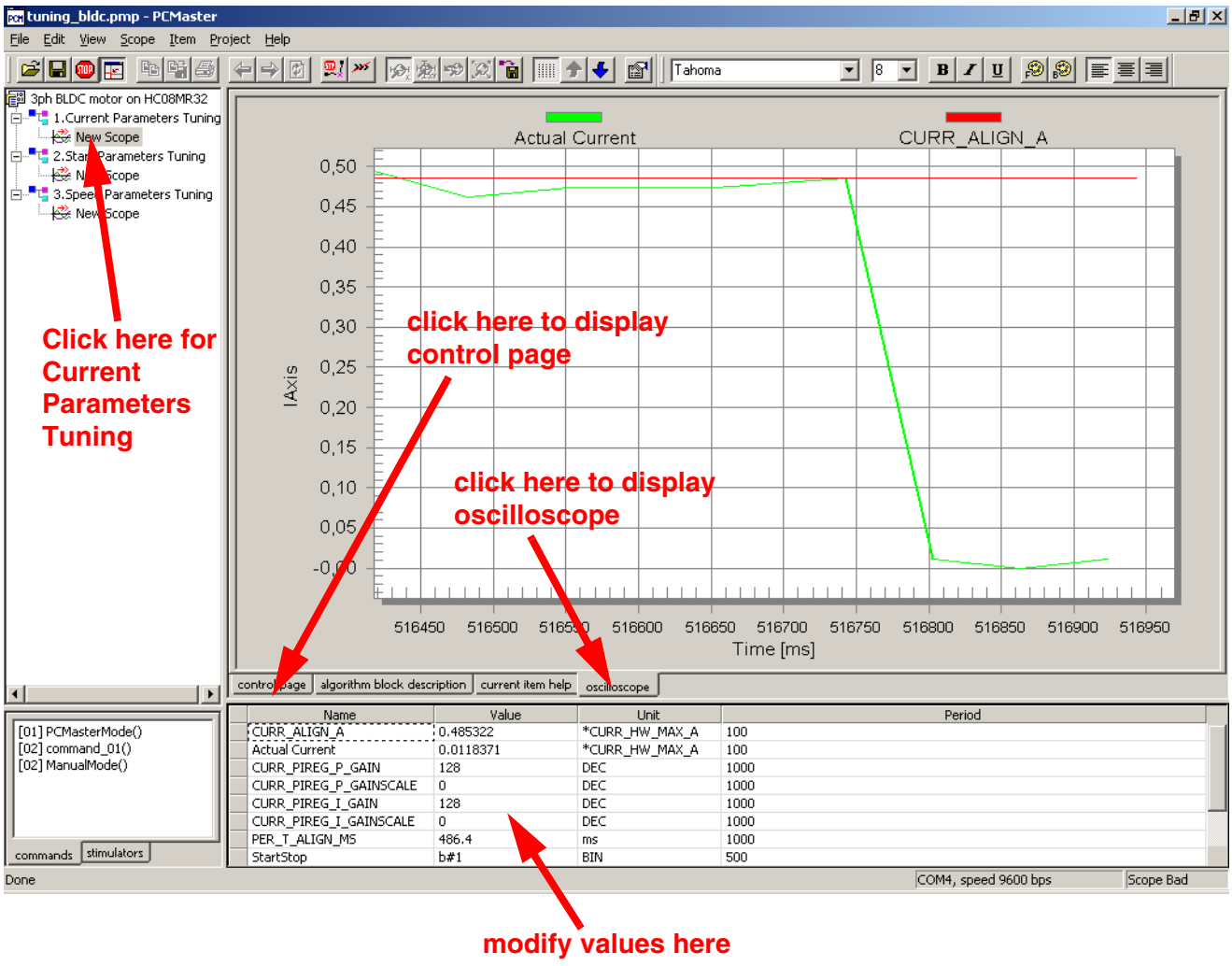

Figure 6-12. PC Master Software Current Parameters Tuning Window

8. Increase, step by step, the proportional gain CURR_PIREG_P_GAIN with PC master software, until current noise or oscillation appears, or up to 128

9. If CURR_PIREG_P_GAIN is set to 128, do further proportional gain increase CURR_PIREG_P_GAINSCALELEFT with PC master software, steps 0, 1, 2...8, otherwise leave CURR_PIREG_P_GAINSCALELEFT as 0
10. Increase, step by step, the integral gain CURR_PIREG_I_GAIN with PC master software, up to current oscillation or noise, or up to 128
11. If CURR_PIREG_I_GAIN is set to 128, do further integral gain increases to CURR_PIREG_I_GAINSCALELEFT with PC master software, steps 0, 1, 2...8; otherwise, leave CURR_PIREG_I_GAINSCALELEFT as 0
12. You can further evaluate the setting of the regulator parameters in order to get a smoother current waveform, or until the regulation seems to be performing well
13. Open **const_cust_x.h** and modify the regulator parameters with the final variable values evaluated with PC master software
14. Change the software back to normal: in code_start.c file, label TUNING_1 remove goto (modify as comment): /* goto Align */ (it will allow finishing Alignment state when alignment period ends)
15. Build the code (see **Software Execution**, **Build**)
16. You can also tune regulator dynamic characteristics of current transients (steps **17.** to **26.**) or finish the regulators tuning
17. Run the code (see **Software Execution** and **Execute from Evaluation Board**)
18. Start the PC master software tuning project
19. Select Current Parameters Tuning subproject (see **Software Parameters Tuning with PC Master Software Project File**) in order to be able to modify the current regulator
20. You can see the actual current (and required alignment current) on the Current Parameters Tuning\New Scope, or measure the powered motor coil current on real oscilloscope
21. Set PC master software control mode and start motor (see **Application Control** and **PC Master Software (Remote) Operating Mode**)

22. Observe the current transient at Alignment start, then stop motor (or reset software)
23. Then modify the regulator parameters with PC master software as in steps **8.**, **9.**, **10.**, and **11.**
24. Repeat steps **21.** to **23.** until regulation is improved
25. Open **const_cust_x.h** and modify the regulator parameters with the final variable values evaluated with PC master software
26. Build the code (see **Software Execution** and **Build**)

The last Alignment setting constant is Alignment Time period [ms]:

```
/* MUST_CHANGE_6_EXPER: */
#define PER_T_ALIGN_MS  1000.0
```

Range: <0,PER_BASE_T3_ALIGN_US/1000/255>

This period can be set experimentally. This constant can also be evaluated using PC master software tuning file. This period must be high enough to let the rotor stabilize during Alignment state. It is recommended that you begin with large values such as 20,000 ms, then it can be lowered. The period should be set to ensure that the rotor (and, therefore, also the current) is stabilized at the end of Alignment state.

6.5.5 Software Customizing to Motor — Commutation and Start-up Control Setting

When all voltage and current settings are done, the motor commutation and start-up parameters need to be set.

For settings which must be done, follow the labels **MUST_CHANGE_nn**, **MUST_CHANGE_EXPER_nn** in file ***const_cust_x.h***.

For changes, which can be done (but usually are not necessary), follow the labels **CAN_CHANGE_nn**, **CAN_CHANGE_EXPER_nn** in file ***const_cust_x.h***

NOTE: *Thanks to the Motorola patented start-up technique, the start parameters setting is quite simple and reliable. However, in order to reliably start the motor, the commutation control constants must be properly set.*

Detailed description starts here.

6.5.5.1 Commutation Parameters

Commutation time period to discharge coil current [µs]

```
/* MUST_CHANGE_7: */
#define PER_DIS_US 300.0
```

Range: <0,minimal commutation period*COEF_TOFF>

It is the maximal allowed current decay period, determined by motor winding and maximal current.

Must be:

PER_DIS_US < minimal motor commutation period[µs] * **COEF_TOFF**

where: **COEF_TOFF** is commutation Toff period coefficient from const.h file explained in the section **3.3 Used Control Technique**.

NOTE: *If **PER_DIS_US** is too high, it can cause commutation errors at high speed*

Half Commutation (advancing) Coefficient [-]:

```
/* CAN_CHANGE_9: */
#define COEF_HLFCMT  0.375
```

Range: <0,1>

COEF_HLFCMT, multiplied by commutation period, determines the time from back-EMF zero crossing to motor commutation. So, it sets the electrical angle from back-EMF zero crossing to motor commutation step. The software controls BLDC motor with a 6-step commutation (six commutations per one electrical rotation), which means 60° between commutations. For ideal commutation with no advancing (no field weakening), the back-EMF zero crossing should be just in the middle between commutations, which means that the electrical angle (ZC-Cmt angle), between back-EMF zero crossing and commutation, is 30°.

$$\text{COEF\_HLFCMT} = \frac{\text{ZC-Cmt angle}}{60} \qquad \textbf{(EQ 6-6.)}$$

ZC-Cmt angle = 15° for COEF_HLFCMT = 0.25

ZC-Cmt angle = 22.5° for COEF_HLFCMT = 0.375

ZC-Cmt angle = 30° for COEF_HLFCMT = 0.5

In the real system, the ZC-Cmt angle is a little bit greater than the theoretical calculation. This is due to the response time of the hardware back-EMF zero crossing sensing. Therefore, the default software setting is **COEF_HLFCMT** = 0.375

Normally, **COEF_HLFCMT** should only be changed if you need a different commutation angle (time from back-EMF zero crossing to commutation). For example, for motor field weakening.

The relation between **COEF_HLFCMT** and the commutation can also be defined by Advance_angle, which is the electrical angle shift from ideal commutation.

$$Advance\_angle = 30\ Deg - ZC\text{-}Cmt\ angle \quad \textbf{(EQ 6-7.)}$$

$$COEF\_HLFCMT = \frac{1}{2} - \frac{Advance\_angle}{60} \quad \textbf{(EQ 6-8.)}$$

Advance_angle = 15° for COEF_HLFCMT = 0.25

Advance_angle = 7.5° for COEF_HLFCMT = 0.375

Advance_angle = 0° for COEF_HLFCMT = 0.5

The relation between back-EMF zero crossing and the commutation is explained in the section **3.3.1.3 Running — Commutation Time Calculation**.

6.5.5.2 Start-up Constants and Maximal Commutation Period

Constants defining start-up need to be changed according to the drive dynamics.

Start Commutation Period [µs]:

```
/* MUST_CHANGE_8_EXPER: */
#define PER_CMT_START_US 4000.0
```

Range: <0,PER_CMT_MAX_US/2>

PER_CMT_START_US is the period used to calculate the first (start) commutation period.

PER_CMT_START_US period must be changed for any motor accommodation. It can be set experimentally. If the motor displays errors during Starting (Back-EMF Acquisition) state, beginning

Running state, or has a low start-up torque, DO decrease or increase the **PER_CMT_START_US** value. **Table 6-2** shows typical setting examples.

Must be:

PER_CMT_START_US<=PER_CMT_MAX_US/2

NOTE: *Setting this constant is an empirical process. It is difficult to use a precise formula, because there are many factors involved which are difficult to obtain in the case of a real drive (motor and load mechanical inertia, motor electromechanical constants, and sometimes also the motor load). So they need to be set with a specific motor.*

***PER_CMT_START_US** can be evaluated with PC master software tuning file tuning_bldc.pmp.*

Table 6-2. Start-up Period

Motor Size	Typical PER_CMT_START_US	First-to-Second Commutation Step Period	Second-to-Third Commutation Step Period
Slow motor/high load and motor mechanical inertia	8000.0 µs	8 ms	8–16 ms
Fast motor/high load and motor mechanical inertia	2000.0 µs	2 ms	2–4 ms

Maximal commutation period limit [µs]:

```
/* CAN_CHANGE_6_EXPER: */
#define PER_CMT_MAX_US 65536.0
```

Range: <0,65535*UNIT_PERIOD_T2_US>

Usually it is not recommended to change **PER_CMT_MAX_US**. The change is only necessary if the commutation period at start-up is to long.

Alignment to Start Increment of the regulators output [-]:

```
/* CAN_CHANGE_7_EXPER: */
#define START_INCR_OOUTREG 20.0
```

Range: <-128,127>

START_INCR_OOUTREG should not necessarily be changed for a motor accommodation. It can be set experimentally. If the motor has a low torque, increase the value. If the motor starts with a high speed, then slows down by regulator, decrease the value.

NOTE: ***START_INCR_OOUTREG*** *can be evaluated with PC master software tuning file tuning_bldc.pmp.*

Number of successive feedbacks necessary to enter the Running state [-]:

```
/* CAN_CHANGE_8_EXPER: */
#define I_CNTR_FOK 0x03
```

Range: <0,255>

The motor starts spinning with Starting (Back-EMF Acquisition) state. The software enters regular Running state with speed regulation after **I_CNTR_FOK** back-EMF successive commutation steps are done.

Usually it is not recommended to change **I_CNTR_FOK**, but it can be evaluated when there are problems with motor start up.

NOTE: ***I_CNTR_FOK*** *can be evaluated with PC master software tuning file uning_bldc.pmp.*

We suggest using PC master software with tuning file tuning_bldc.pmp for start-up parameters evaluation. You can use this procedure:

1. Ensure that the Alignment current and regulator were properly set (**Alignment Current and Current Regulator Setting**)**6.5.4.3** in **const_cust_x.h**
2. Ensure that **PER_DIS_US** and **COEF_HLFCMT** are properly set in **const_cust_x.h**
3. Set #define **PER_CMT_START_US** in **const_cust_x.h** according to **Table 6-2**.
4. Ensure **PER_CMT_START_US**<=**PER_CMT_MAX_US**/2
5. Set #define **START_INCR_OOUTREG 20.0** in **const_cust_x.h**
6. In order to disable speed regulator, temporarily change the software by clearing speed regulator parameters:
7. #define SPEED_PIREG_P_GAIN 0 /* 64 */

#define SPEED_PIREG_I_GAIN 0

8. in **const_cust_x.h** file
9. Build and run the code (see **Software Execution**, **Build**, and **Execute from Evaluation Board**)
10. Start the PC master software tuning project
11. Select Start Parameters Tuning subproject (see **Software Parameters Tuning with PC Master Software Project File**) in order to be able to modify the start parameters
12. You can see the actual zero-crossing (commutation) period on the Start Parameters Tuning\New Scope, or measure the phase a, b, and c voltages on a real oscilloscope

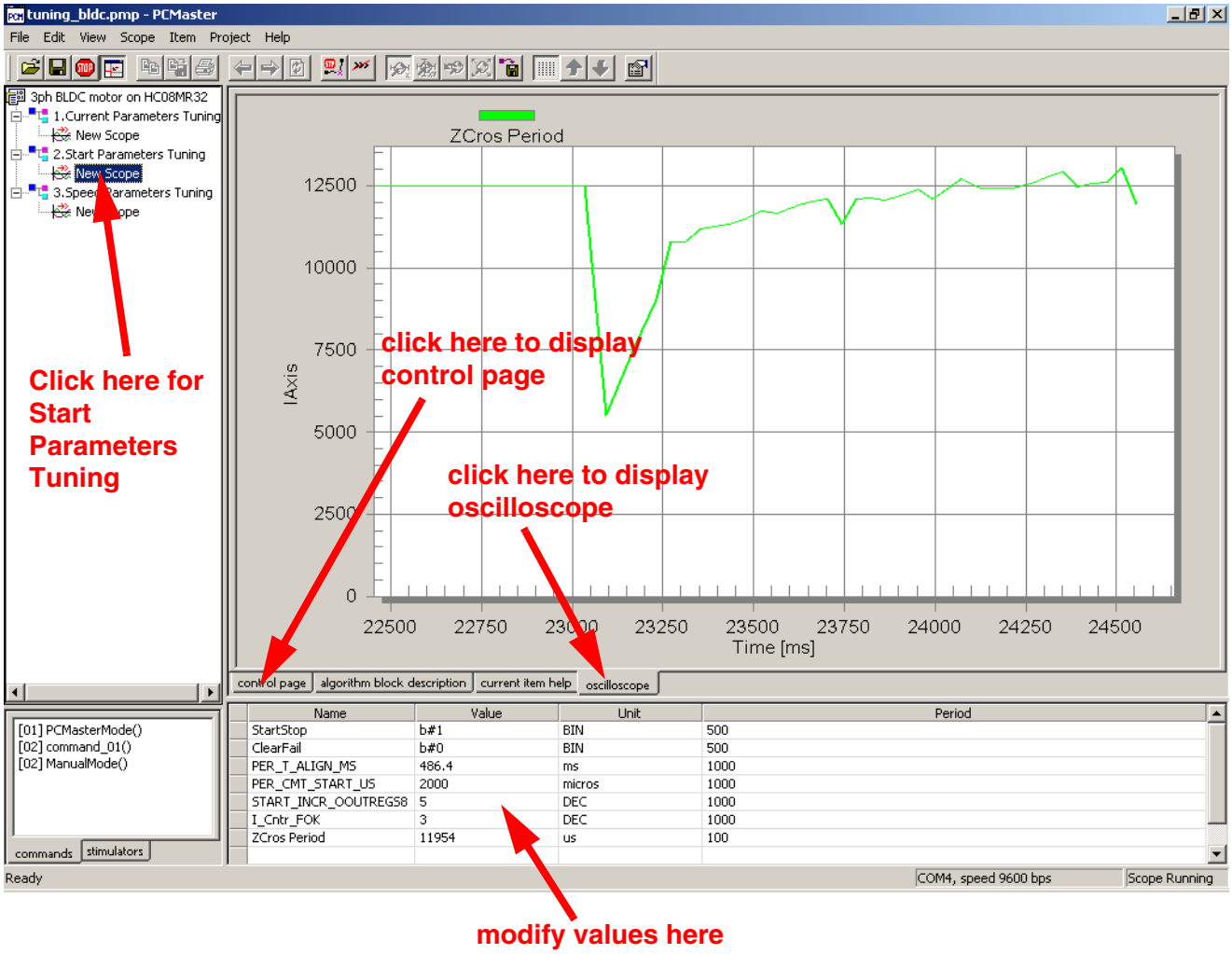

Figure 6-13. PC Master Software Start Parameters Tuning Window

13. Set PC master software control mode (see **Application Control** and **PC Master Software (Remote) Operating Mode**)
14. Start motor (see **Application Control** and **PC Master Software (Remote) Operating Mode**)
15. If the software signalizes errors (usually commutation error), clear the errors, stop the motor, and change **PER_CMT_START_US** (increase or decrease!) by PC master software
16. Repeat step **15.** until the motor starts well. If the motor starts against a high start-up torque, or if Alignment state current is low,

it is recommended to change **START_INCR_OOUTREG** by PC master software. (If it is a problem to start the motor, then **I_CNTR_FOK** can also be changed from default 0x03, but not recommended!)

17. If the motor starts and continues running, after you repeatedly start/stop, the start-up parameters are set properly
18. Open **const_cust_x.h** and modify parameters with the final variable values **PER_CMT_START_US, START_INCR_OOUTREG**, evaluated with PC master software.
19. Change the software back to normal, set speed regulator parameters to:
20. #define SPEED_PIREG_P_GAIN 64
    #define SPEED_PIREG_I_GAIN 0
21. in **const_cust_x.h** file to enable speed regulation
22. Build the code (see **Software Execution** and **Build**)

6.5.6 Software Customizing to Motor — Speed Control Setting

When the motor commutation setting is done, the speed control parameters need to be set.

For settings which must be done, follow the labels **MUST_CHANGE_nn**, **MUST_CHANGE_EXPER_nn** in file ***const_cust_x.h***.

For changes which can be done (but usually are not necessary), follow the labels **CAN_CHANGE_nn**, **CAN_CHANGE_EXPER_nn** in file ***const_cust_x.h***

Number of commutations per motor revolution:

```
/* MUST_CHANGE_9: */
#define COMMUT_REV   18.0
```

Range: <0,255>

COMMUT_REV period must be changed for any motor accommodation. Set the number of commutations according to the number of rotor poles (there are six commutations for one electrical angle revolution). Therefore:

$$COMMUT\_REV = \frac{6*motor\ poles}{2} \quad \textbf{(EQ 6-9.)}$$

Maximal speed range [rpm]:

```
/* MUST_CHANGE_10: */
#define SPEED_RANGE_MAX_RPM 3000.0
```

Range: <0,infinity>

Determines scaling of speed variables. **SPEED_RANGE_MAX_RPM** must be changed for any motor accommodation as the software calculates the internal speed variables using this constant. For proper speed control it is important to set **SPEED_RANGE_MAX_RPM** higher then maximal actual motor speed (even during speed transient).

Maximal speed of the drive [rpm]:

```
/* MUST_CHANGE_11: */
#define SPEED_MAX_RPM 2500.0
```

Range: <0,SPEED_RANGE_MAX_RPM>

SPEED_MAX_RPM determines the maximal desired speed. It must be changed for any motor accommodation.

The software calculates the internal speed variables using **SPEED_RANGE_MAX_RPM** constant. For proper speed control it is important that **SPEED_MAX_RPM** and **SPEED_RANGE_MAX_RPM** constants relation must be set according to the following equation:

$$SPEED\_MAX\_RPM < SPEED\_RANGE\_MAX\_RPM \quad \textbf{(EQ 6-10.)}$$

Minimal speed of the drive [rpm]:

```
/* MUST_CHANGE_12_EXPER: */
#define SPEED_MIN_RPM 500.0
```

Range: <0,SPEED_RANGE_MAX_RPM>

SPEED_MIN_RPM determines the minimal desired speed. It must be changed for any motor accommodation. The minimal speed is also determined by the back-EMF zero crossing technique. Usually:

$$SPEED\_MIN\_RPM = (0.07 \text{ to } 0.5)SPEED\_MAX\_RPM \quad \textbf{(EQ 6-11.)}$$

Therefore, for low speed requirements minimal speed SPEED_MIN_RPM must be evaluated experimentally.

NOTE: ***SPEED_MIN_RPM*** *can be evaluated with PC master software tuning file tuning_bldc.pmp.*

Minimal PWM Duty cycle limit [-]:

```
/* CAN_CHANGE_11: */
#define DUTY_PWM_MIN 0.250
```

Range: <0,1>

DUTY_PWM_MIN determines minimal PWM duty cycle limit, and in this way it restricts minimal voltage on the motor.

Therefore, **DUTY_PWM_MIN** must be changed if its default setting creates a higher voltage than is physically necessary to run with a speed close to **SPEED_MIN_RPM**.

CAUTION: *If the motor is unable to run down to the speed set in* **SPEED_MIN_RPM***, then decrease* **DUTY_PWM_MIN** *constant.*

Speed PI regulator constants:

```
/* MUST_CHANGE_13_EXPER: */
#define SPEED_PIREG_P_GAINSCALELEFT 0
```

Range: <0,8>

```
/* MUST_CHANGE_14_EXPER: */
#define SPEED_PIREG_P_GAIN  128
```

Range: <0,255>

where the current regulator proportional gain is:

$$KP = SPEED\_PIREG\_P\_GAIN * 2^{SPEED\_PIREG\_P\_GAINSCALELEFT} \quad \textbf{(EQ 6-12.)}$$

```
/* MUST_CHANGE_15_EXPER: */
#define SPEED_PIREG_I_GAINSCALERIGHT 0
```

Range: <0,8>

```
/* MUST_CHANGE_16_EXPER: */
#define SPEED_PIREG_I_GAIN  64
```

Range: <0,255>

where the current regulator integral gain is:

$$KI = SPEED\_PIREG\_I\_GAIN * 2^{(-SPEED\_PIREG\_I\_GAINSCALERIGHT)}$$

(EQ 6-13.)

These constants can be calculated according to regulators theory. The speed sampling (regulator execution) period is **PER_T3_RUN_US** = 2.560 ms at default software setting. Another recommended solution is experimental setting.

NOTE: *SPEED_PIREG_P_GAINSCALELEFT, SPEED_PIREG_P_GAIN, SPEED_PIREG_I_GAINSCALERIGHT, SPEED_PIREG_I_GAIN can be evaluated with PC master software tuning file tuning_bldc.pmp.*

We suggest using PC master software with tuning file tuning_bldc.pmp for regulator parameters evaluation. You can use this procedure:

1. Ensure that the start-up and commutation parameters were set properly (**Start-up Constants and Maximal Commutation Period**)**6.5.4.3** in **const_cust_x.h**
2. Set **const_cust_x.h**:

   SPEED_PIREG_P_GAINSCALELEFT 0
   SPEED_PIREG_P_GAIN 0
   SPEED_PIREG_I_GAINSCALERIGHT 7
   SPEED_PIREG_I_GAIN 0
3. Ensure that **COMMUT_REV**, **SPEED_RANGE_MAX_RPM**, **SPEED_MAX_RPM** are set properly in **const_cust_x.h**
4. Set **SPEED_MIN_RPM** as required (should be **SPEED_MIN_RPM** > **SPEED_MAX_RPM/5** for reliable commutation at low speed)

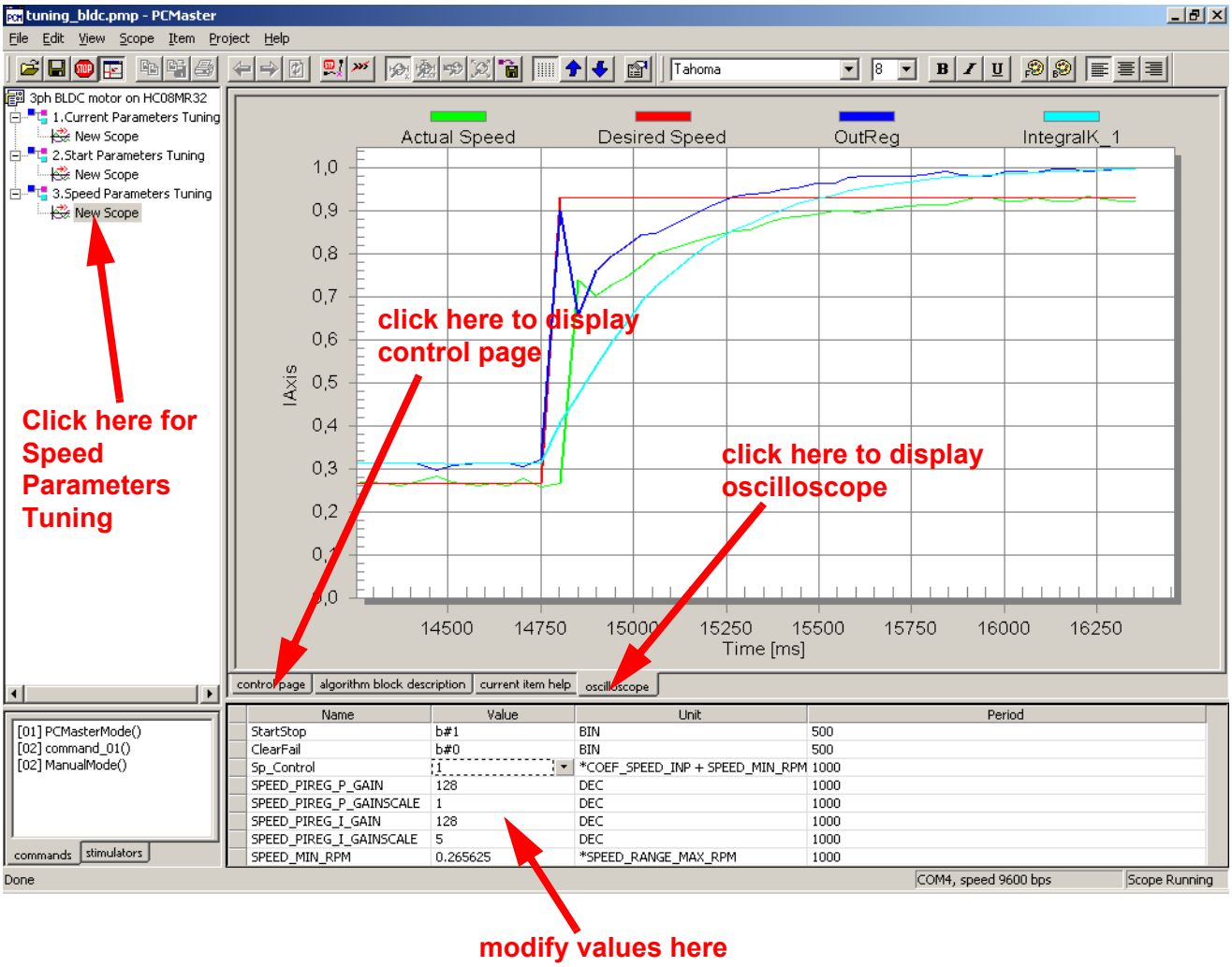

Figure 6-14. PC Master Software Speed Parameters Tuning Window

5. Build and run the code (see **Software Execution**, **Build**, and **Execute from Evaluation Board**)
6. Start the PC master software tuning project
7. Select Speed Parameters Tuning subproject (see **Software Parameters Tuning with PC Master Software Project File**) in order to be able to modify the current regulator
8. You can see the actual speed (and desired speed) on the Speed Parameters Tuning\New Scope, or measure the phase voltage period on real oscilloscope

9. Set PC master software control mode and start motor (see **Application Control** and **PC Master Software (Remote) Operating Mode**)
10. Set the speed to the middle of minimal and maximal speed
11. Increase, step by step, the proportional gain SPEED_PIREG_P_GAIN with PC master software, until speed noise or oscillation appears, or up to 128
12. If SPEED_PIREG_P_GAIN is set to 128, increase proportional gains in SPEED_PIREG_P_GAINSCALELEFT further with PC master software, steps 0, 1, 2 ... 8 otherwise leave SPEED_PIREG_P_GAINSCALELEFT as 0
13. Increase, step by step, the integral gain SPEED_PIREG_I_GAIN with PC master software, up to current oscillation or noise, or up to 128
14. If SPEED_PIREG_I_GAIN is set to 128, do further integral gain increases to SPEED_PIREG_I_GAINSCALELEFT with PC master software, steps 6, 5, 4...0; otherwise, leave SPEED_PIREG_I_GAINSCALELEFT as 7
15. You can further evaluate the setting of the regulator parameters in order to get a smoother current waveform until the regulation seems to be performing well
16. Open **const_cust_x.h** and modify the regulator parameters with the final variable values evaluated with PC master software
17. Start motor (see **Application Control** and **PC Master Software (Remote) Operating Mode**)
18. Set minimal desired speed
19. If the displayed real speed is not able to go down to the desired minimal speed, it is necessary to decrease minimal PWM duty cycle DUTY_PWM_MIN in **const_cust_x.h**!
20. Then, you can tune dynamic characteristics of speed regulators (steps **22.** to **29.**) or finish tuning the regulators
21. Start motor (see **Application Control** and **PC Master Software (Remote) Operating Mode**)

22. Set minimal speed
23. Set maximal speed and observe the speed transient
24. Set minimal speed and observe the speed transient
25. Then, modify the regulator parameters with PC master software as in steps **11.** to **14.**
26. Change **SPEED_MIN_RPM** if problems occur at low speed
27. Repeat steps **21.** to **25.** until regulation is improved
28. Open **const_cust_x.h** and modify the regulator parameters with the final variable values evaluated with PC master software
29. Build the code (see **Software Execution** and **Build**)

Most important software settings are described in previous sections, but for some applications, PWM frequency must be modified. It is described in **PWM Frequency and Current Sampling Period Setting**.

Once you set the speed control and the motor is running in all start, speed up, and slow down conditions, the software parameters are set for the motor. Remember that all parameters are set in **const_cust_x.h**. Then, it is possible to program the FLASH memory of the MC68HC908MR32 device.

6.5.7 PWM Frequency and Current Sampling Period Setting

PWM frequency and current sampling period settings are not usually needed. The PWM frequency also affects the current sampling period. Consequently, the current regulation setting should be done, while understanding their mutual dependency. Therefore, the PWM frequency setting is provided in the file **const.h**, instead of **const_cust_x.h**.

6.5.7.1 PWM Frequency

For the PWM frequency setting, follow the label **CAN_CHANGE_FPWM_n** in **const.h** file.

The PWM frequency setting is provided by:

```
/* CAN_CHANGE_FPWM_1: */
#define SET_PER_PWM     32.0
```

Range: <1,255>

The final PWM period is defined by setting **SET_PER_PWM**.

The PWM period [µs] is:

PWM period = PERIOD_PWM_US = SET_PER_PWM*2 (EQ 6-14.)

With default software setting (oscillator clock, etc.).

The final PWM frequency [Hz] is:

$$PWMfrequency = \frac{10^6}{2SET\_PER\_PWM} \quad \textbf{(EQ 6-15.)}$$

With default software setting.

Settings for some important PWM frequencies are listed in **Table 6-3**.

Table 6-3. PWM Frequency Setting

SET_PER_PWM	PWM Frequency (FREQUENCY_PWM)	PWM Period (PERIOD_PWM_US)
16.0	31.250 Hz	32 µs
25.0	20.000 Hz	50 µs
32.0 (default)	15.625 Hz	64 µs
128.0	3.90625 Hz	256 µs

CAUTION: *Current measurement sampling period is synchronized with PWM. Therefore, changing PWM frequency automatically changes the current sampling period. This, apart from other things, effects the current regulator. Therefore, after changing PWM frequency, changing (or checking) current sampling period is strongly recommended.*

6.5.7.2 Current Sampling Period

Current sampling period should usually be changed in two cases:

1. When PWM frequency is changed
2. Motors with externally low electrical constant

If the motor electrical constant is lower than default current sampling period of 128 µs, the current regulator may not work properly.

For current sampling period setting follow the label **CAN_CHANGE_PERCURSAMP_n** in **const.h** file.

Current sampling period setting is provided by:

```
/* CAN_CHANGE_FPWM_n: */
/* CAN_CHANGE_PERCURSAMP_n: */
#define SET_PER_CS      2.0
```

Range: <1,->

The final current sampling period [µs] is:

$$\text{Current sampling period} = \text{PWM period} * \text{SET\_PER\_CS} \quad \textbf{(EQ 6-16.)}$$

current sampling period = PWM period * **SET_PER_CS** [µs]

PER_CS_T1_US = **PERIOD_PWM_US** * **SET_PER_CS** [µs]

6.5.7.3 Current Sampling Instant

Time period from a PWM reload event (middle of central aligned PWM) to current sampling (time shift of A/D conversion with PWM) [µs]:

```
/* CAN_CHANGE_PERCURSAMP_n: */
#define PER_PWM_CS_US   5.0
```

Range: <-PERIOD_PWM_US/2,PERIOD_PWM_US/2>

Usually it is not recommended to change **PER_PWM_CS_US**, but it can be evaluated when there are problems with back-EMF zero crossing noise.

It is necessary to set **SET_PER_CS** according to the following equation:

$$-\frac{\text{PERIOD\_PWM\_US}}{2} < \text{PER\_PWM\_CS\_US} < \frac{\text{PERIOD\_PWM\_US}}{2} \quad \textbf{(EQ 6-17.)}$$

6.5.8 Conclusion Software Parameters Setting and Tuning

If all the points in **Tuning for Customer Motor** are done, the software should be customized to customer motor.

If the software customizing of your motor was not successful, it is recommended that you read **Application Suitability Guide**, since the

software may not be suitable for some applications. Some important recommendations can also be found under the ***Caution*** and ***Note*** labels in this designer reference manual.

Appendix A. References

1. *Sensorless BLDC Motor Control on MC68HC908MR32 - Software Porting to Customer Motor* (document order number AN2356/D), Motorola 2002
2. Motion Control Development Tools found on the World Wide Web at:
   http://e-www.motorola.com
3. *Motorola Embedded Motion Control MC68HC908MR32 Control Board User's Manual*, (document order number MEMCMR32CBUM/D), Motorola 2000
4. *Motorola Embedded Motion Control 3-Phase AC BLDC High-Voltage Power Stage User's Manual* (document order number MEMC3PBLDCPSUM/D), Motorola 2000
5. *Motorola Embedded Motion Control Optoisolation Board* (document order number MEMCOBUM/D), Motorola 2000
6. *Motorola Embedded Motion Control Evaluation Motor Board User's Manual* (document order number MEMCEVMBUM/D), Motorola 2000
7. *Motorola Embedded Motion Control 3-Phase BLDC Low-Voltage Power Stage User's Manual* (document order number MEMC3PBLDCLVUM/D), Motorola 2000
8. User's Manual for PC Master Software, Motorola 2000, found on the World Wide Web at:
   http://e-www.motorola.com
9. *68HC908MR32, 68HC908MR16 Advance Information* (document order number MC68HC908MR32/D), Motorola
10. *Low Cost High Efficiency Sensorless Drive for Brushless DC*

References

Motor using MC68HC(7)05MC4 (document order number AN1627), Motorola

Appendix B. Glossary

AC — Alternative Current.

ACIM — AC Induction Motor.

A/D converter— analog to digital converter.

ADC — analog to digital converter - see “A/D converter”

back-EMF — back Electro-Motive Force (in this document it means the voltage inducted into motor winding due to rotor movement)

BLDC — Brushless DC motor.

CW — CodeWarrior - compillers produced by Metrowerks

DC — Direct Current.

dc-bus — part of power converter with direct current

DC-motor — Direct Current motor, if not mentioned differently, it means the motor with brushes.

DT — see “Dead Time (DT)”

Dead Time (DT) — short time that must be inserted between the turning off of one transistor in the inverter half bridge and turning on of the complementary transistor due to the limited switching speed of the transistors.

duty cycle — A ratio of the amount of time the signal is on versus the time it is off. Duty cycle is usually represented by a percentage.

ECLOVACBLDC — 3-ph AC/BLDC Low Voltage Power Stage

ECMTRLOVBLDC — 3-ph BLDC Low Voltage Motor-Brake SM40N + SG40N

Glossary

ECMTREVAL — Evaluation Motor Board Kit (supplied in kit with trapezoidal BLDC IB23810)

ECOPTHIVACBLDC — 3-ph AC/BLDC High Voltage Power Stage + Optoisolation Board

ECMTRHIVBLDC — 3-ph BLDC High Voltage Motor-Brake SM40V + SG40N

ECOPTINL — Optoisolation between host computer and MCU board evaluation or customer target cards (optoisolation board)

ECOPT — Optoisolation between power stage and processor evaluation or controller cards (in line optoisolator)

ECCTR908MR32 — Motor Control board for HC908MR32

IDE — Integrated Developement Environment

interrupt — A temporary break in the sequential execution of a program to respond to signals from peripheral devices by executing a subroutine.

input/output (I/O) — Input/output interfaces between a computer system and the external world. A CPU reads an input to sense the level of an external signal and writes to an output to change the level on an external signal.

KITMMDS08MR32 — MMDS 68HC908MR32/16 developement kit

logic 1 — A voltage level approximately equal to the input power voltage (V_{DD}).

logic 0 — A voltage level approximately equal to the ground voltage (V_{SS}).

M68HC8 — A Motorola family of 8-bit MCUs.

MC — Motor Control

MCU — Microcontroller Unit. A complete computer system, including a CPU, memory, a clock oscillator, and input/output (I/O) on a single integrated circuit.

MW — Metrowerks Corporation

PCM — PC master software for communication between PC computer and system

phase-locked loop (PLL) — A clock generator circuit in which a voltage controlled oscillator produces an oscillation which is synchronized to a reference signal.

PMP — PC master software project file

PVAL — PWM value register of motor control PWM module of MC68HC908MR32 microcontroller. It defines the duty cycle of generated PWM signal.

PWM — Pulse Width Modulation

reset — To force a device to a known condition.

SCI — See "serial communication interface module (SCI)."

serial communications interface module (SCI) — A module that supports asynchronous communication.

serial peripheral interface module (SPI) — A module that supports synchronous communication.

software — Instructions and data that control the operation of a microcontroller.

software interrupt (SWI) — An instruction that causes an interrupt and its associated vector fetch.

SPI — See "serial peripheral interface module (SPI)."

SR — switched reluctance motor.

timer — A module used to relate events in a system to a point in time.

Glossary

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

DRM028/D