MOTOROLA
intelligence everywhere™

digital dna™

High-voltage BLDC Drive for Domestic Appliances using the MC68HC908MR8

Designer Reference Manual

M68HC08
Microcontrollers

DRM048
Rev 0, 12/2003

MOTOROLA.COM/SEMICONDUCTORS

High-voltage BLDC Drive for Domestic Appliances using the MC68HC908MR8

Designer Reference Manual — Rev 0

by: Petr Uhlir and 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)
December 2003	0	Initial release	N/A

WARNING: ***This application operates in an environment that includes dangerous voltages and rotating machinery. The application power stage and optoisolation board are not electrically isolated from the mains voltage. They are live, and there is a risk of electric shock if they are touched.***

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

Section 1. Introduction

1.1 Application Outline

This reference design describes the design of a low-cost compressor drive using sensorless 3-phase brushless direct current (BLDC) motor control with back-EMF (electromotive force) zero-crossing sensing. It is based on Motorola's MC68HC908MR8 microcontroller, which is intended for appliance applications.

The system is designed as a motor drive system for low-power 3-phase BLDC motors and is aimed at applications in automotive, industrial and appliance fields (e.g. in fridge compressors, air conditioning units, pumps and simple industrial drives).

This reference design focuses on the motor control part of the refrigeration system. The cooling control algorithms are intellectual property of the fridge manufacturers and can be resolved using different approaches. These algorithms can be programmed into the system according to the requirements of the application.

The reference design incorporates both hardware and software parts of the system including hardware schematics.

1.2 Benefits of Variable Speed Compressor Drives

As stated above, one area where the advantages of BLDC motors can be used is refrigeration. Let us look at the advantages offered by variable-speed compressors over current solutions.

Most current refrigerators use conventional compressors. These compressors use uncontrolled single-phase induction motors with auxiliary start-up windings. The advantage of this solution is that it offers cheap and simple construction with no need for electronic control. The

main disadvantage is the poor efficiency of the system, caused by on/off control, and inefficient use of the parameters of the induction motor.

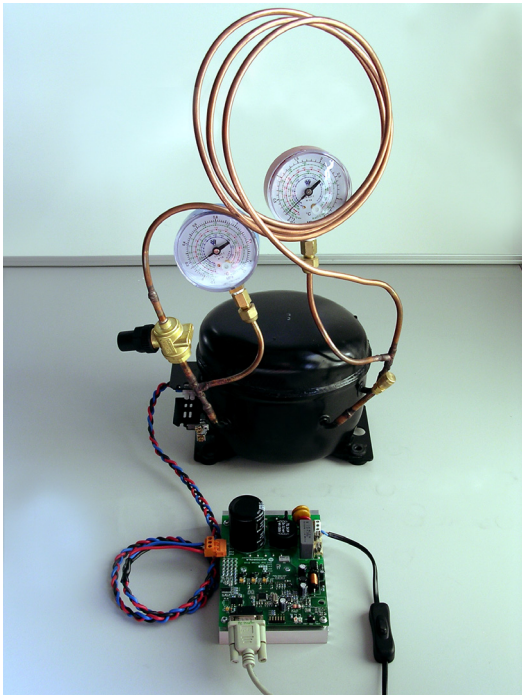


Figure 1-1. Fridge Compressor Drive

Further energy savings cannot be achieved with standard compressors. New compressors with variable-speed-controlled drives, which allow the output power to be adjusted to suit the requirements of the refrigeration systems, must be used. The benefits of using variable-speed compressors instead of conventional compressors are as follows.

- Energy savings up to 30%
- Low noise due to the ability to operate at low speeds
- Faster cooling due to operation at higher speeds
- Many other advantages offered by the microcontroller

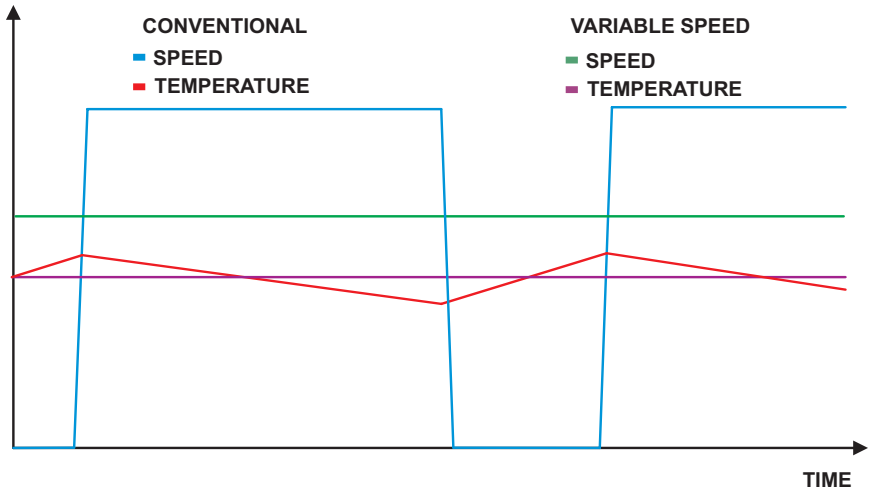


Figure 1-2. Conventional Versus Variable Speed Compressor

The difference in temperature control between conventional and variable-speed compressors is shown in **Figure 1-2**.

The fridge with a conventional compressor controls the inside temperature using a bimetallic switch. When the inside temperature exceeds the upper threshold, the switch switches on the motor, causing the temperature to decrease until the temperature reaches the lower threshold when the motor is stopped. This ‘bang-bang’ control causes the temperature to fluctuate by about 4 degrees.

The electronically controlled variable-speed compressor uses the temperature read from the temperature sensor to calculate the motor speed required to keep the inside temperature constant.

The most suitable motor used in variable-speed compressors is the BLDC motor. The advantages of this motor for fractional horsepower applications and its sensorless control are described in detail in this reference design.

1.3 Benefits of the BLDC Motors in Compressors

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

Today more and more variable-speed drives are put in appliance or automotive products to increase the overall system efficiency and the product performance. To satisfy the requirements for high efficiency, high performance and low system costs, control systems based on semiconductor components and MCUs must be used.

Using semiconductor devices, it is possible to replace classical universal and DC motors with maintenance-free electrically commutated BLDC motors. This allows the manufacturer to benefit from the advantages of BLDC motors, without affecting overall system costs.

Advantages of BLDC motors over universal and DC motors include the following:

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

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

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

The advantages of this type of control are:

- Saving in costs of position sensors and wiring

- It can be used where it is not possible or it is too expensive to make additional connections between position sensors and the control unit; this is typically the case for compressors
- Low-cost system (medium demand for control MCU power) required for high volume applications

Introduction

Section 2. System Description

2.1 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 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 between 0 V and 15 V. Zero-crossing detection is done by low-voltage comparators that compares the back-EMF signal from unpowered motor phase with half of the DC-bus. The outputs of the comparators are connected directly to the GPIO (general purpose input/output) pins of the MCU.

Zero-crossing detection is synchronized with the center of the PWM signals by the software, to filter high-voltage spikes produced by switching the IGBTs (insulated gate bipolar transistors).

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 (0–5 V with 2.5 V offset). The internal MCU analog-to-digital converter (ADC) and zero-crossing detection are synchronized with the PWM signal. This synchronization avoids spikes when the IGBTs are switched, and simplifies the electric circuit.

System Description

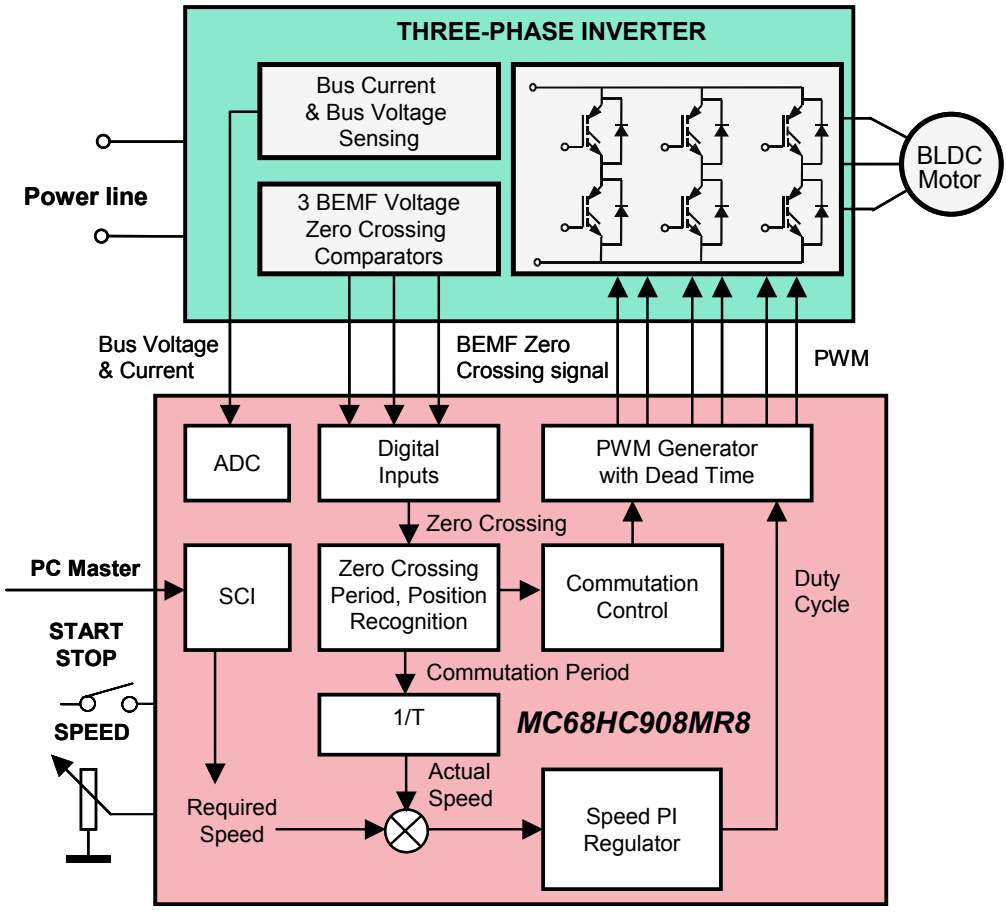


Figure 2-1System Concept

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

The ADC is also used to sense the DC-bus 5.0 V signal level by a resistor network.

The power inverter is designed using the PS21353-N intelligent power module from Mitsubishi, which is targeted at refrigerators, washing machines and other low-power applications. The PWM technique is used to control motor phase voltage.

2.2 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 is an example of a sensorless BLDC motor control system using Motorola's MC68HC908MR8 MCU. It also illustrates the use of on-chip peripherals dedicated to motor control.

The system for BLDC motor control consists of hardware and software. The hardware in designed for variable line voltages of 115–230 VAC and medium power (phase current < 10 A)

2.2.1 Software Specification

The software is written in C. The software 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 software that allows reading and setting of all the system variables, and which 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 tuning the software parameters 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	MC68HC908MR8
Language	C with some arithmetical functions in assembler
Compiler	Metrowerks ANSI-C/C++ Compiler for HC08
Application Control	PC master software (remote) interface (via RS232 using PC)
MCU Oscillator Frequency	4 MHz (with default software setting)
MCU Bus Frequency	8 MHz (with default software setting)
Minimum BLDC Motor Commutation Period (Without PC Master Software Communication)	333 µs (with default software setting and COEF_HLFCMT = 0.450)
Minimum 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 the BLDC high-voltage drive
Software Configuration and Parameter 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.h with predefined parameter settings for running on targeted hardware.
	PWM frequency 15.626 kHz with default software setting, possibly changeable in const.h file

2.2.2 Hardware and Drive Specifications

The other system specifications are determined by the targeted hardware and motor characteristics. The board and its connections are shown in **4 Hardware Design** and **6.2.1 Hardware Configuration**.

2.2.2.1 High-voltage BLDC Drive Specification

This hardware set is designed for medium power (phase current < 10 A) and mains voltage. The specifications for a high-voltage hardware and motor set are listed in **Table 2-2**. The hardware operates on 230 VAC and 115 VAC mains.

Table 2-2. Hardware Specifications

Hardware Boards Characteristics	Input voltage:	230 VAC or 115 VAC
	Maximum DC-bus voltage:	401 V
	Maximum output current:	10 A
Motor Characteristics	Motor type:	3 phase, star connected BLDC compressor motor
Drive Characteristics	Speed range:	< 2500 rpm (determined by motor used)
	Maximum DC-bus voltage:	380 V
	Protection:	Over-current, over-voltage, and under-voltage fault protection
Load Characteristic	Type:	Compressor

System Description

Section 3. BLDC Motor Control

3.1 Brushless DC Motor Control Theory

3.1.1 BLDC Motor Targeted by this Application

The 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 , 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.

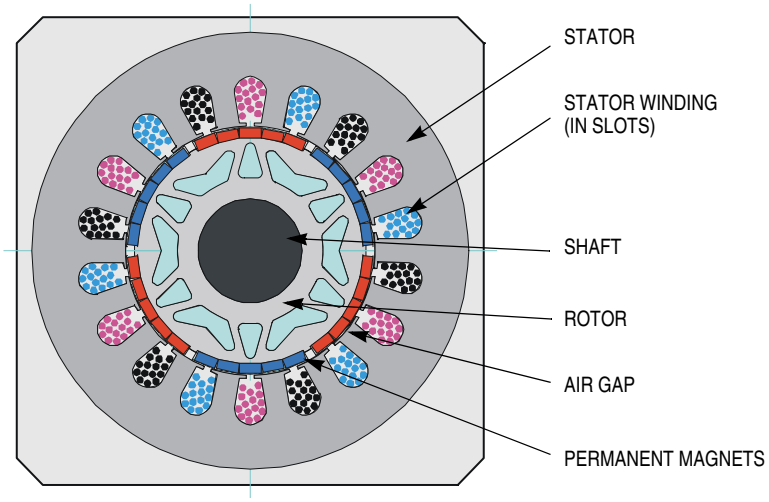


Figure 3-1. BLDC Motor Cross-section

The motor can have more than 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 for each mechanical revolution.

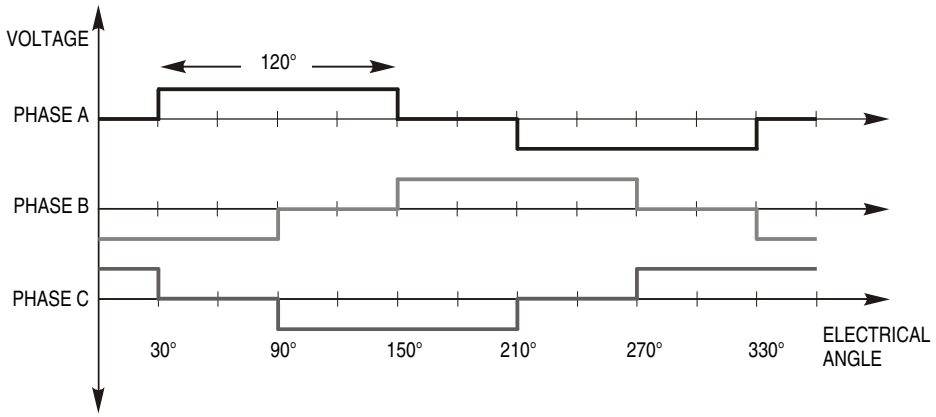


Figure 3-2. 3-phase Voltage System

The easy-to-create rectangular shape of applied voltage ensures the simplicity of control and drive. However, the rotor position must be known at certain angles to be able to align the applied voltage with the back-EMF (the voltage induced by 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.

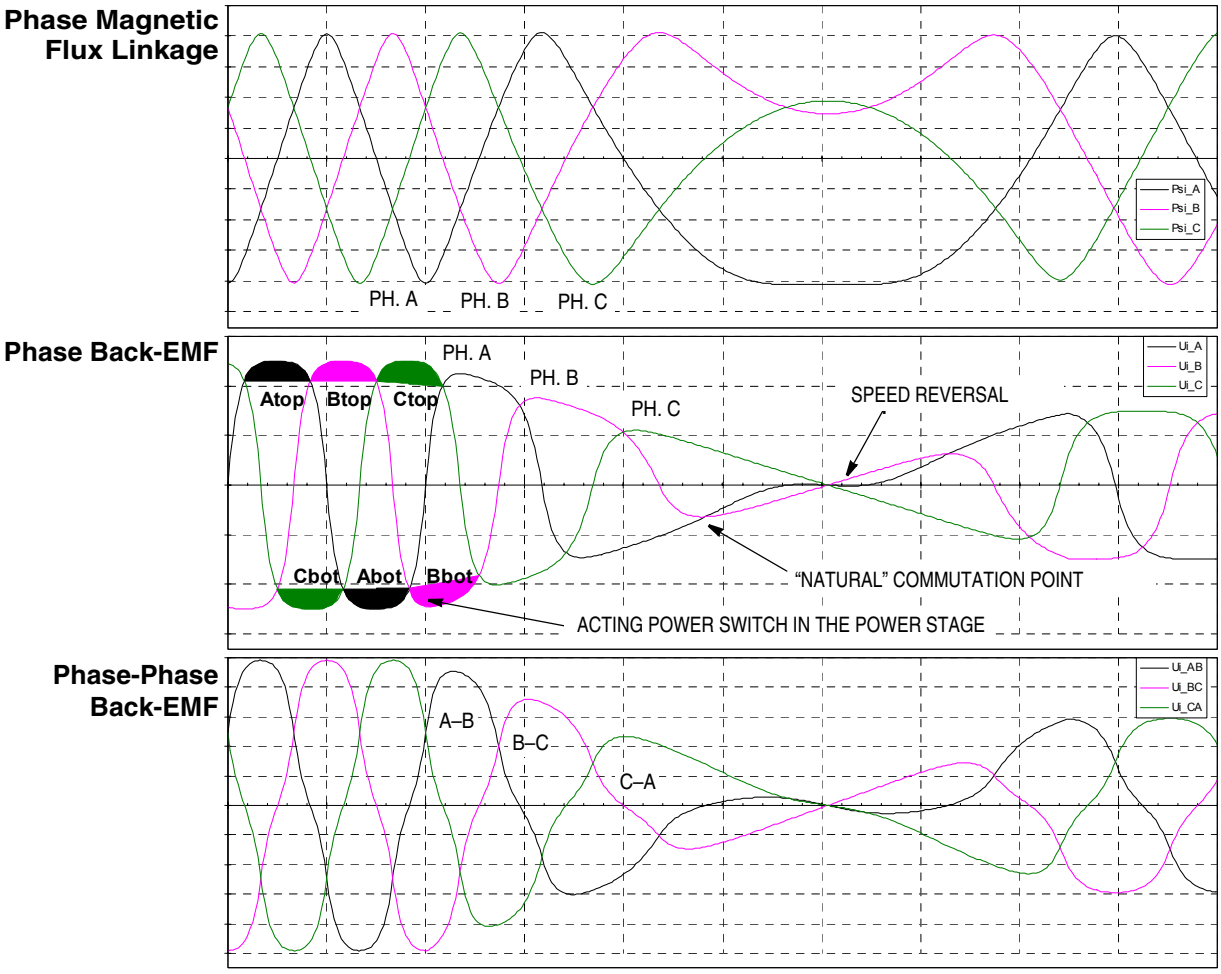


Figure 3-3. BLDC Motor Back-EMF and Magnetic Flux

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 6-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 to control the drive above the pulse-width modulator (PWM) voltage control.

3.1.2 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.

3.1.3 Why Sensorless Control?

As explained in the previous section, rotor position must be known tobe able 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 BLDC Motors in Compressors**).

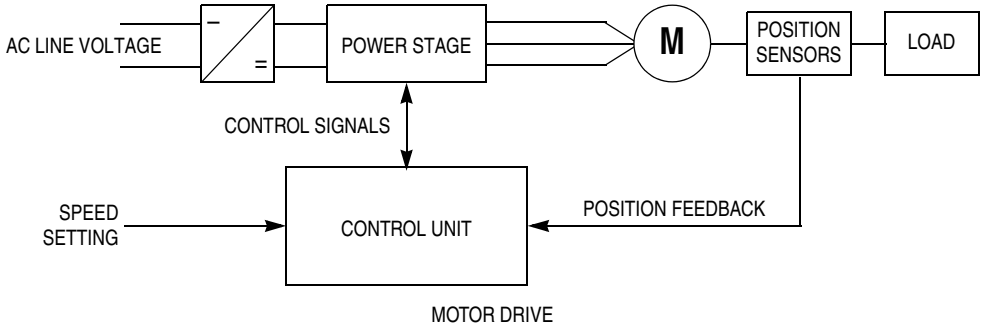


Figure 3-4. Classical System

3.1.4 Power Stage: Motor System Model

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**.

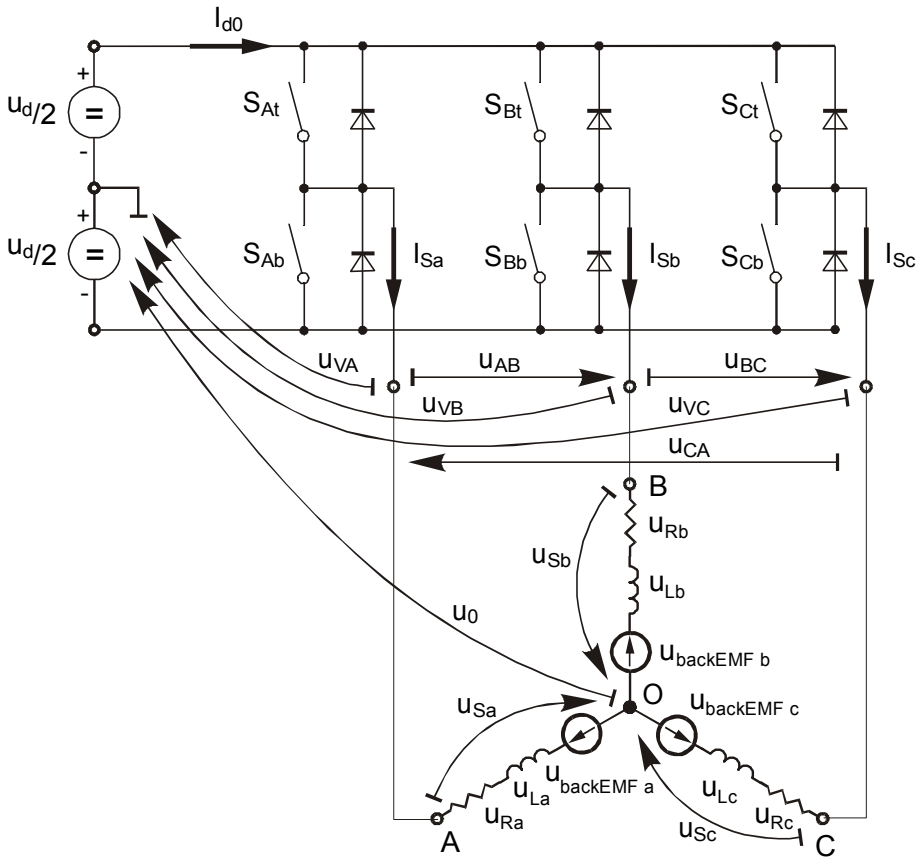


Figure 3-5. Power Stage: Motor Topology

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.

3.1.4.1 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.

3.1.4.2 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 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.

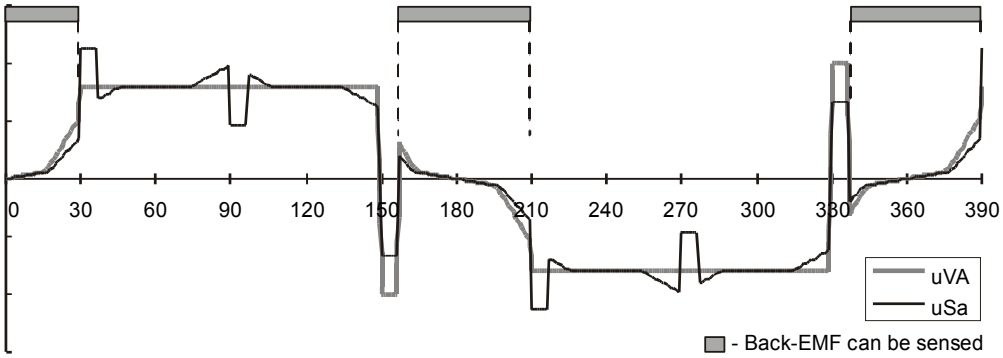


Figure 3-6. Phase Voltage Waveform

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

3.1.4.3 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.

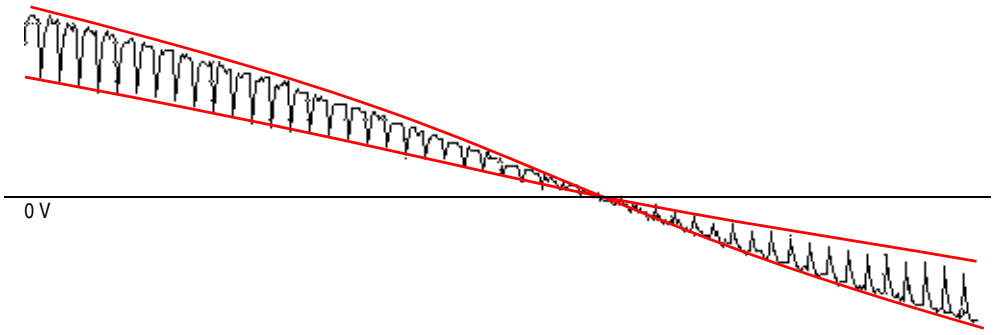


Figure 3-7. Mutual Inductance Effect

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.*

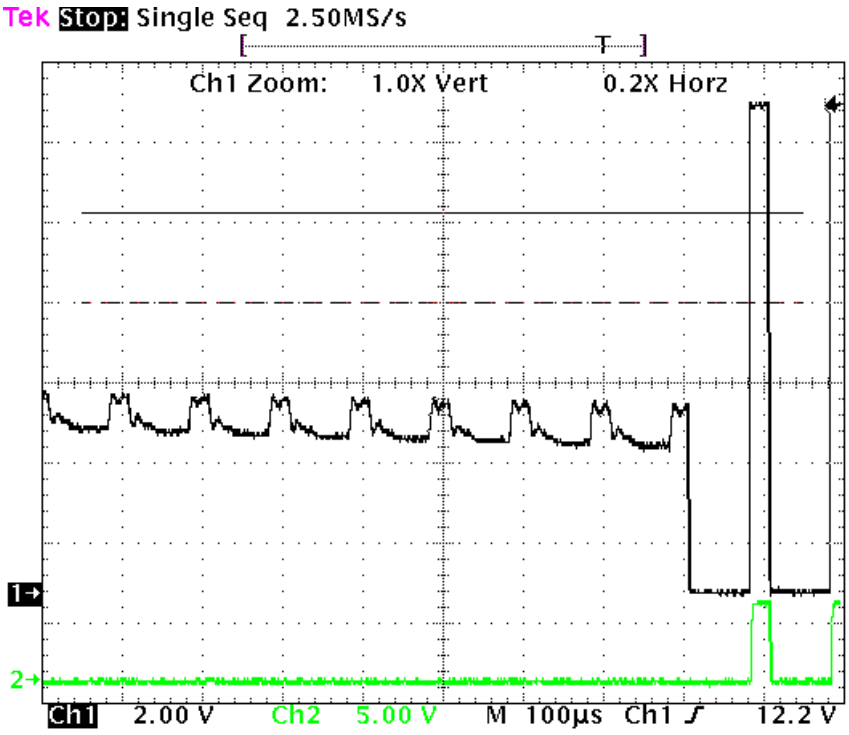


Figure 3-8. Detail of Mutual Inductance Effect

3.1.4.4 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**.

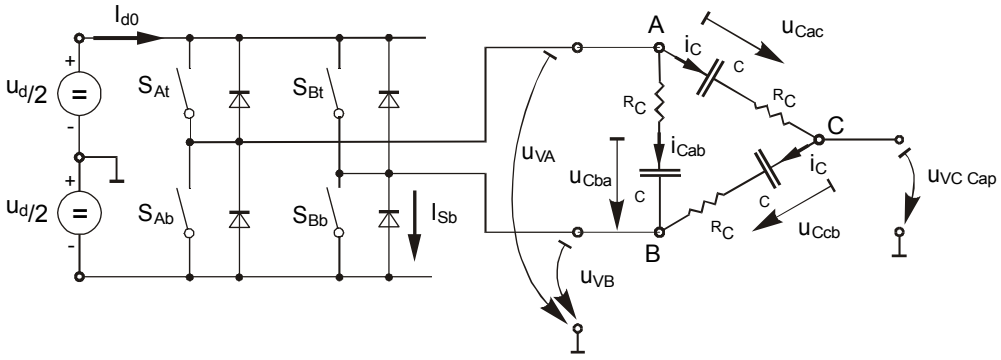


Figure 3-9. Mutual Capacitance Model

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 imbalance 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**.*

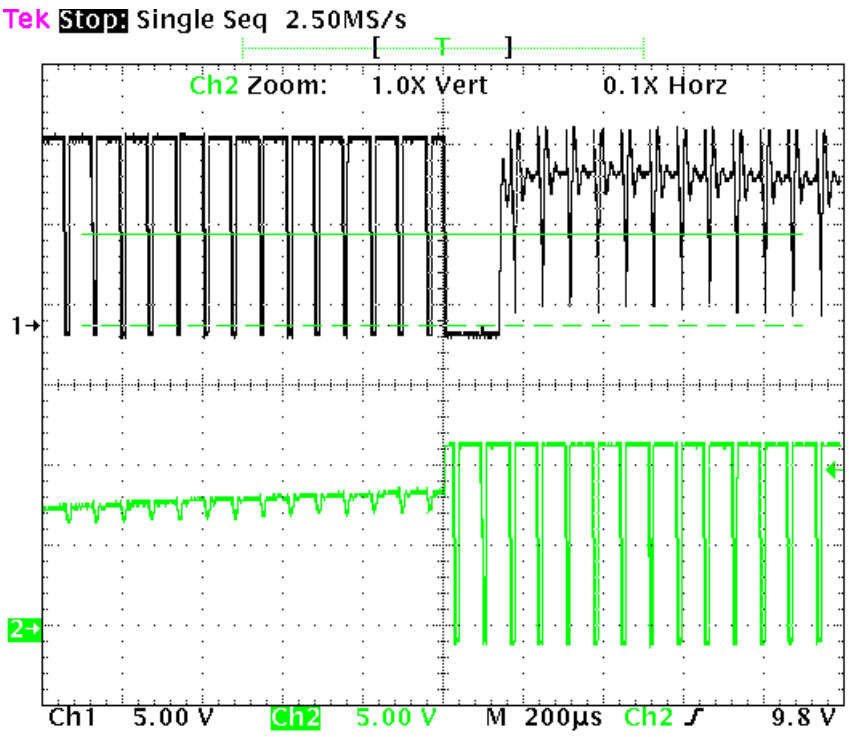


Figure 3-10. Distributed Back-EMF by Unbalanced Capacity Coupling

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 imbalance was made purposely by adding a small capacitor on the motor terminals, to demonstrate the effect more clearly. After the imbalance was removed the “branch” voltage is clean, without any spikes.

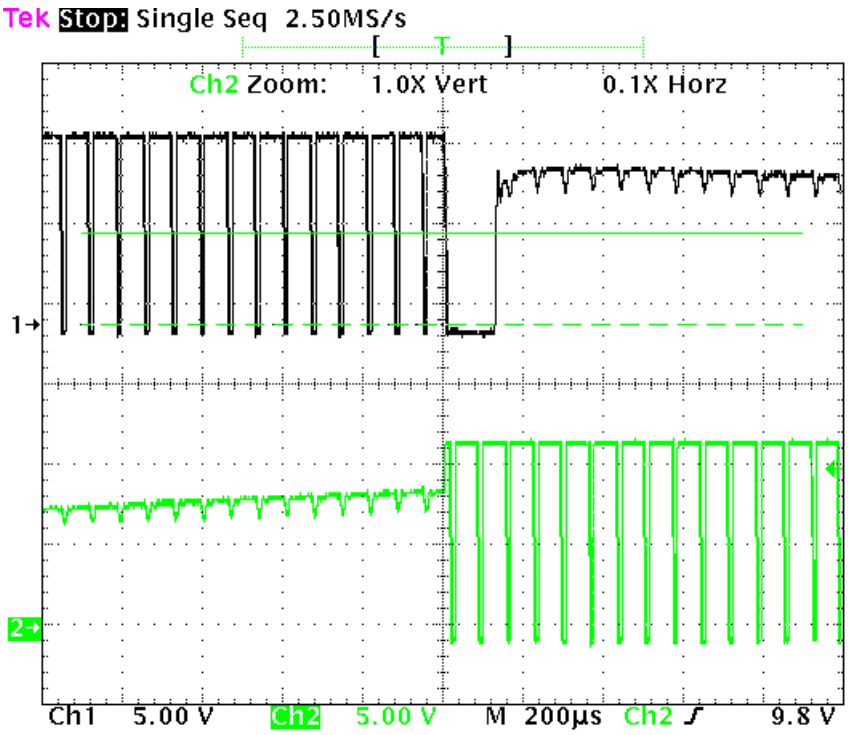


Figure 3-11. Balanced Capacity Coupling

NOTE: *The configuration of the phase windings end-turns has significant impact; therefore, it must 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.*

3.1.5 Back-EMF Sensing Circuit

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

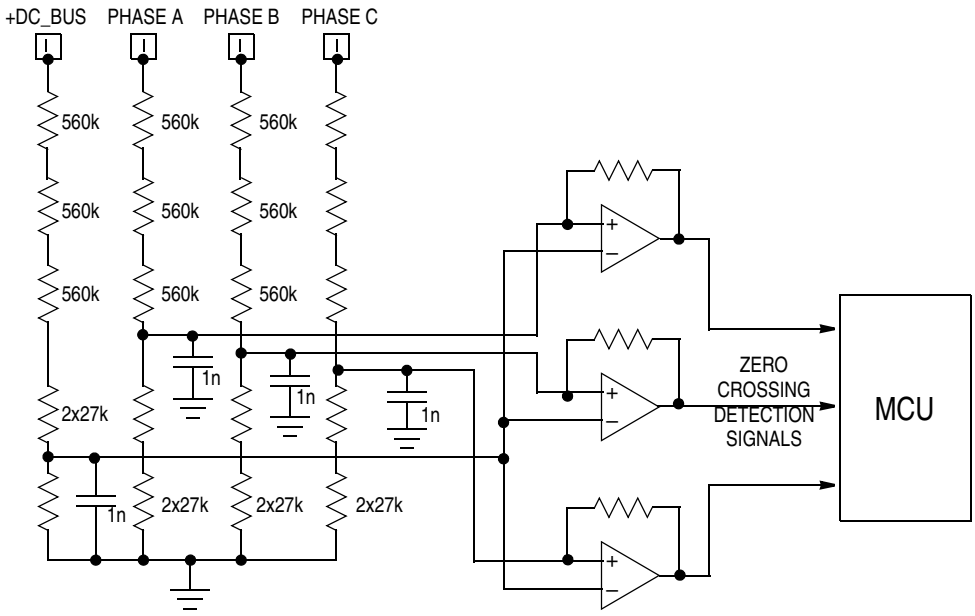


Figure 3-12. Back-EMF Sensing Circuit Diagram

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 comparator outputs are directly connected to the GPIO input pins of the MCU.

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

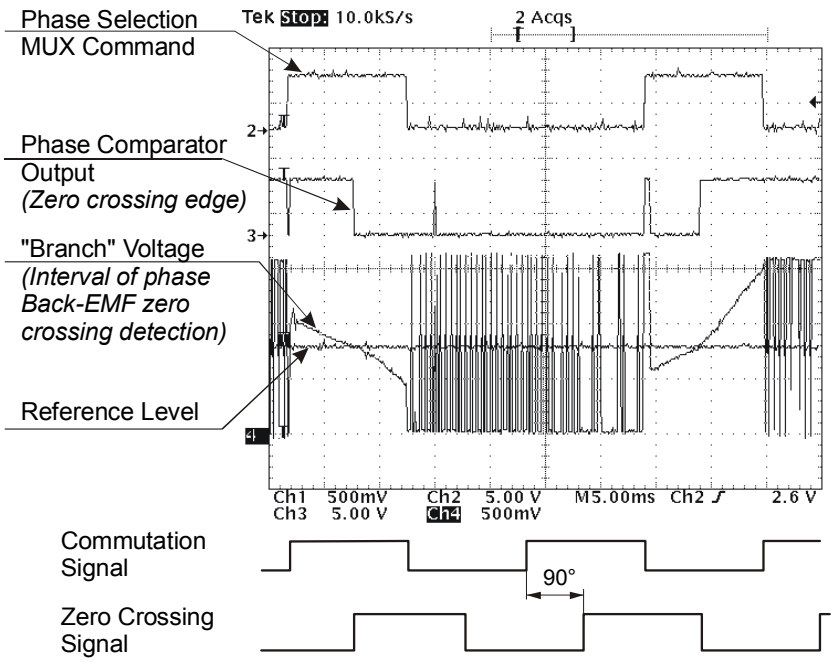


Figure 3-13. The Zero-crossing Detection

3.2 Control Technique Used

3.2.1 Sensorless Commutation Control

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

To start and run the BLDC motor, the control algorithm must 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.

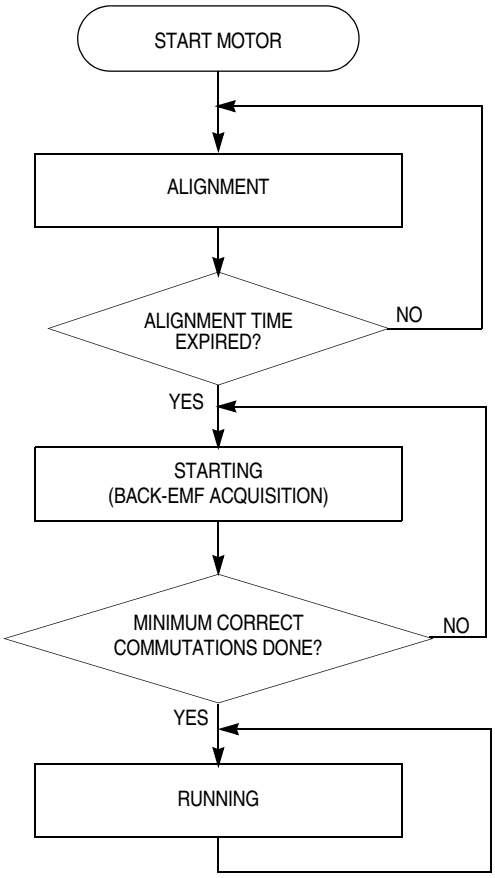


Figure 3-14. Commutation Control Stages

3.2.1.1 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 to create a high start-up torque. When the preset timeout 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, 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**.

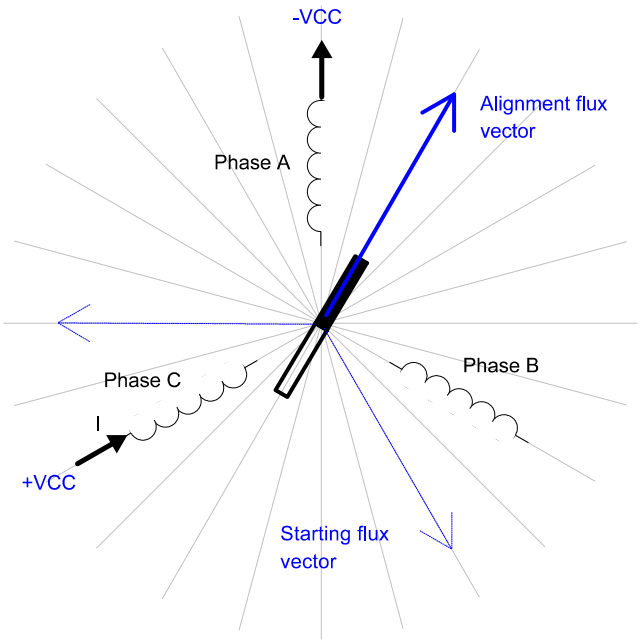


Figure 3-15. Alignment

3.2.1.2 Running

The commutation process is a series of states that ensure:

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

The following processes must be provided:

- BLDC motor commutation service

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

From the diagrams, an overview of how the commutation works can be obtained. After commuting the motor phases, there is a time interval (Per_Toff[n]) when the shape of the back-EMF must be 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 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**.

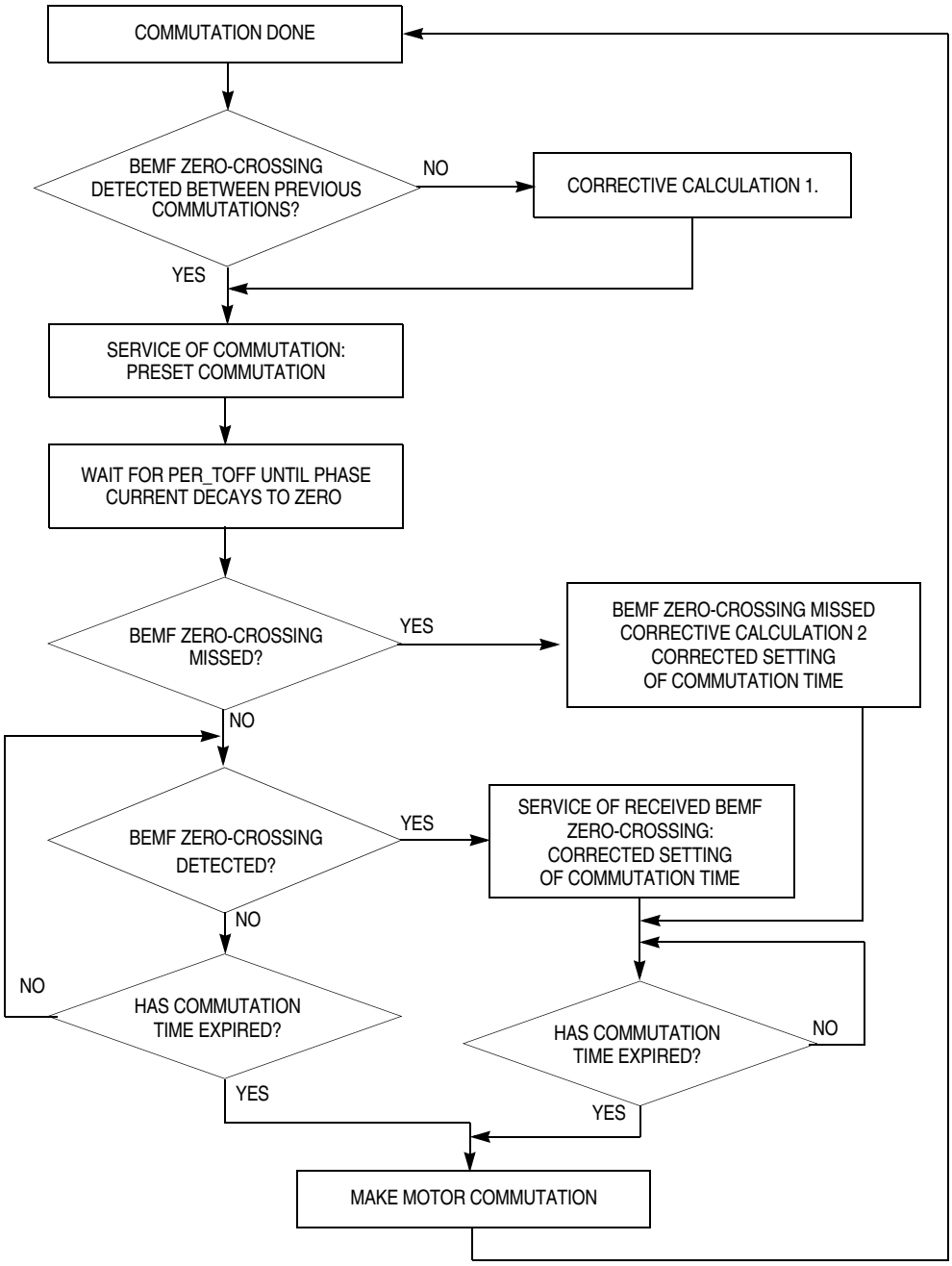


Figure 3-16. BLDC Commutation with Back-EMF Zero-crossing Sensing Flowchart

3.2.1.3 Running: Commutation Time Calculation

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

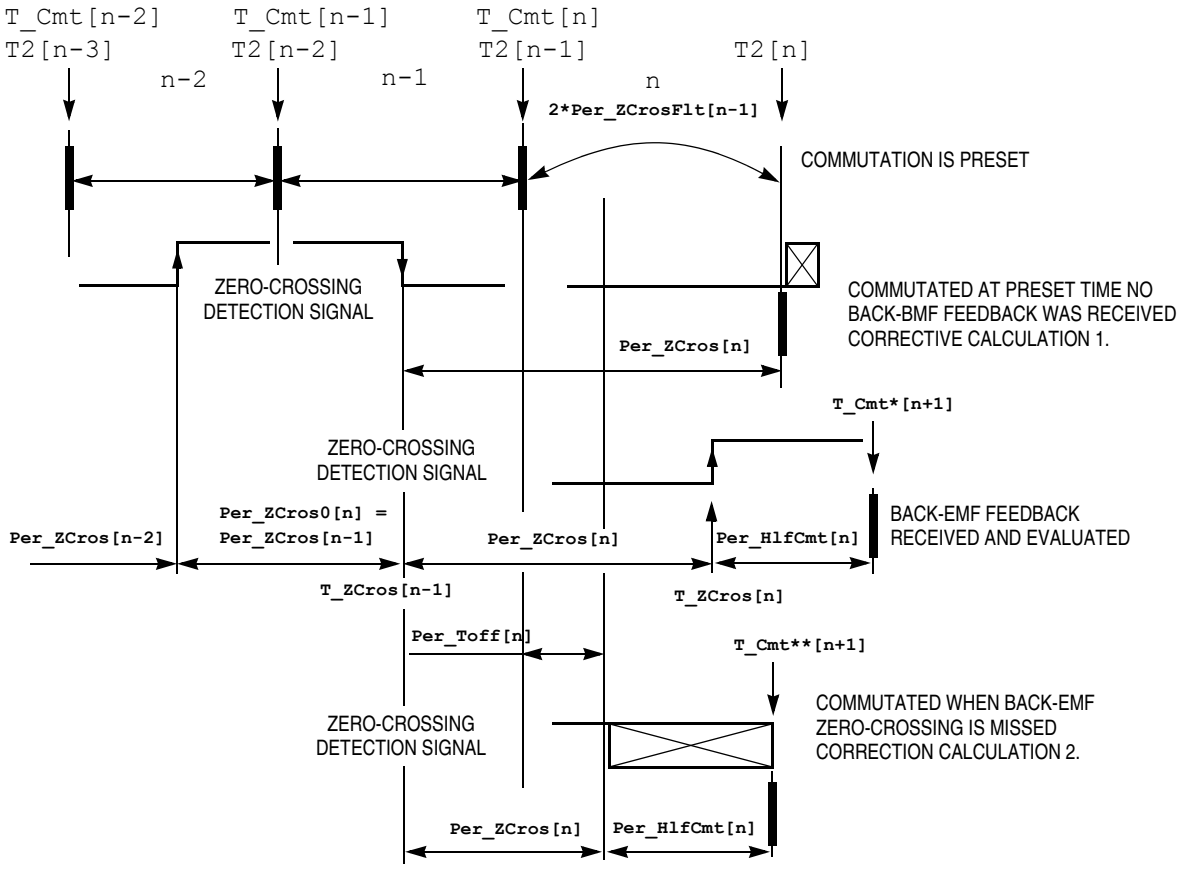


Figure 3-17. BLDC Commutation Time with Zero-crossing Sensing

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**.

3.2.1.4 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.

To start the BLDC motor, 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.

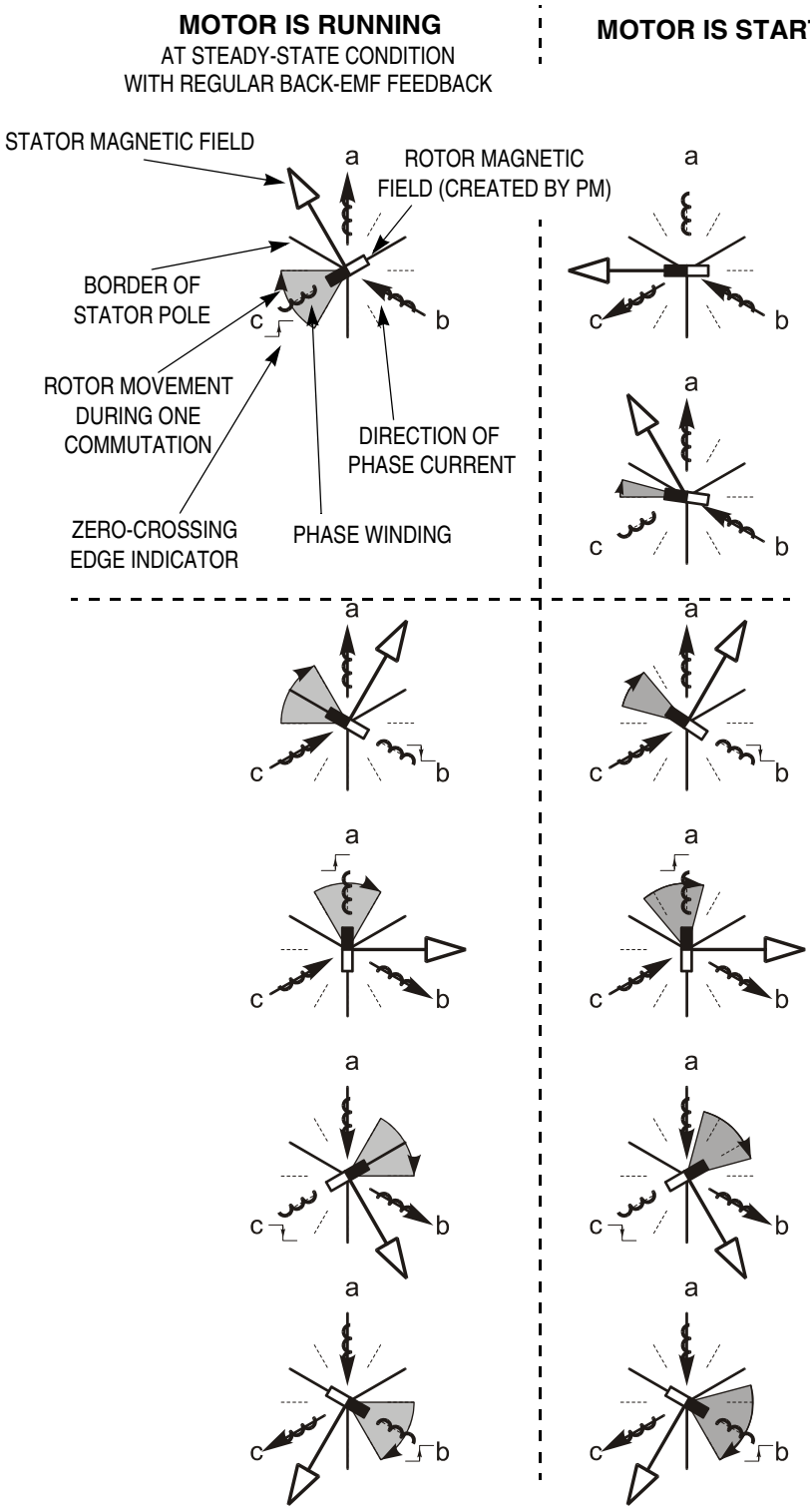
The second condition is more difficult to fulfill without any position feedback information. After the alignment state, the stator and the rotor magnetic fields are aligned (0° angle). Therefore, two fast commutations (faster than the rotor can follow) must be applied to create an angular difference 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 minimum 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 **3.2.1.2 Running**) ensures 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.



ALIGNMENT STATE
The rotor position is stabilized by applying PWM signals to only two motor phases

STARTING (BACK-EMF ACQUISITION)
The two fast (faster than the rotor can move) commutations are applied to create an angular difference of the stator magnetic field and rotor magnetic field.

The back-EMF feedback is tested. When the back-EMF zero-crossing is recognized the time of new commutation is evaluated. Until at least two successive back-EMF zero-crossings are received the exact commutation time can not be calculated. Therefore, the commutation is done in advance to ensure that successive back-EMF zero-crossing events would not be missed.

RUNNING
After several back-EMF zero-crossing events the exact commutation time is calculated. The commutation process is adjusted. The motor is running with regular back-EMF feedback.

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.

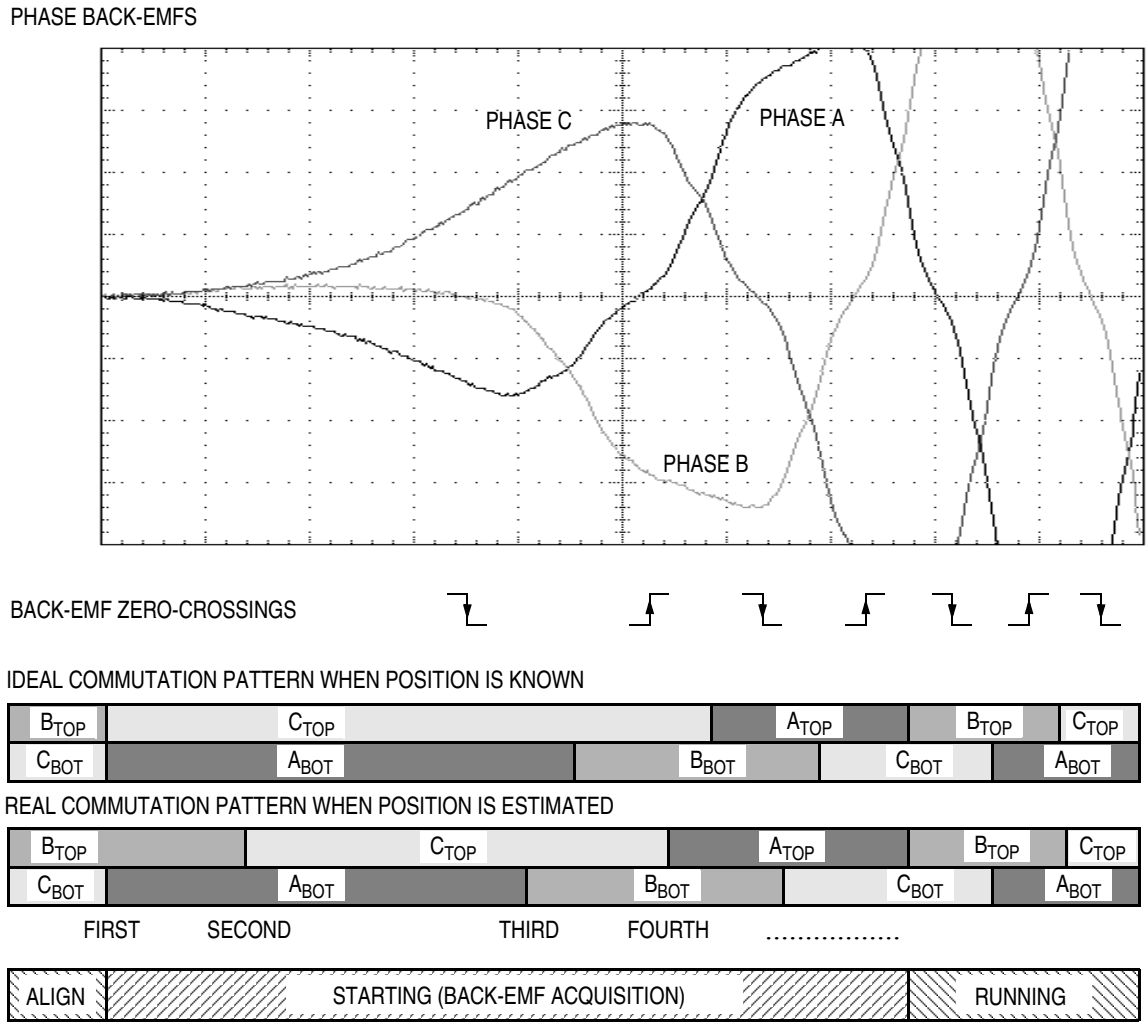


Figure 3-19. Back-EMF at Start Up

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.

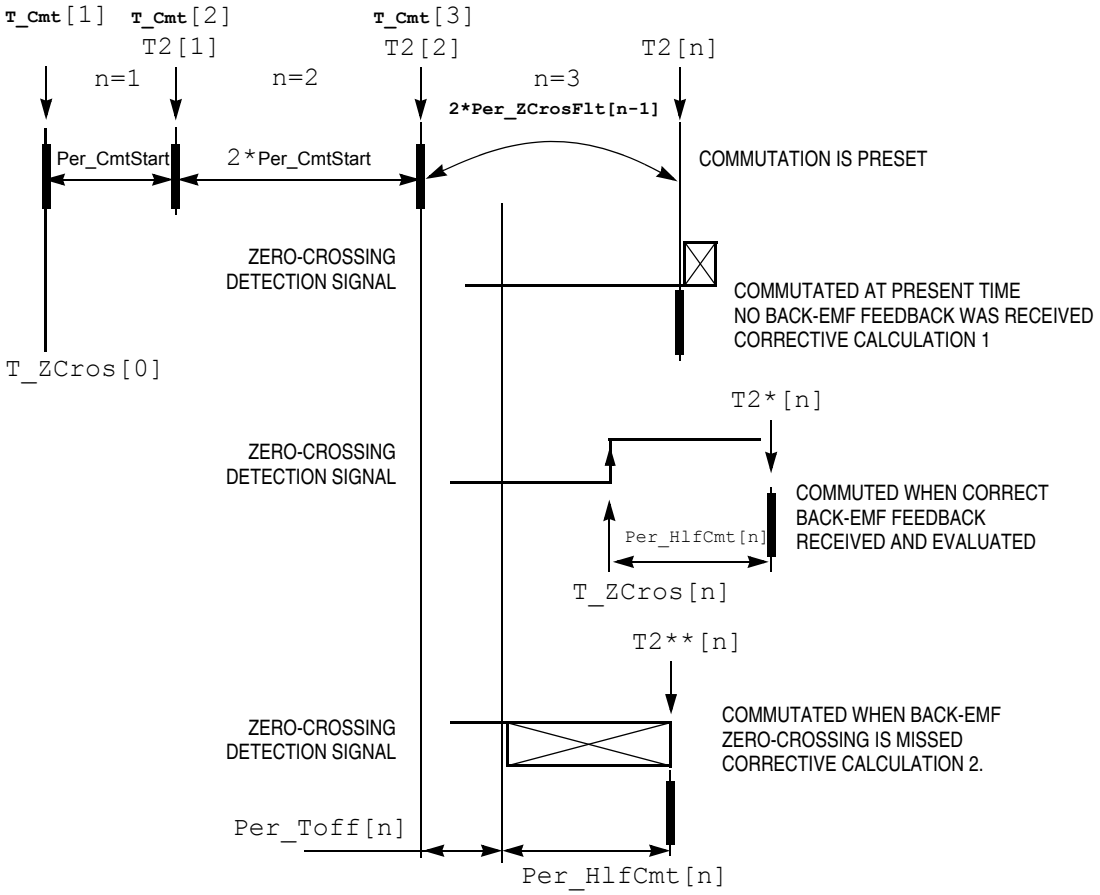


Figure 3-20. Calculation of Commutation Times During the Starting (Back-EMF Acquisition) State

3.2.1.5 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 **3.2.1.3 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
```

3.2.2 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 **5.2.7 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 **5.2.8 Process Speed Control**.

3.3 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.

3.3.1 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 Internet Explorer as a user interface on the PC.

3.3.1.1 Communication with PC Master Software Specifications

SCI communication protocol with a default of 9600 baud, 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

After reset, the BLDC control MCU software is in manual mode. 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.

3.3.1.2 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	Response Byte	Response Description
Set PC master software mode	01	None	Setting of PC master software mode	00 55	OK Failed
Set manual mode	02	None	Setting of manual mode	00 55	OK Failed

3.3.1.3 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]	Maximum speed limit
Speed_Min_RPM	U16	O	< 0; 65535> [rpm]	Minimum 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 16-bit, U16 = unsigned 16-bit

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 System Configuration and Documentation

The 3-phase high-voltage BLDC sensorless drive is a 400 watt single board drive controlled with MC68HC908MR8 microcontroller. The drive is shown in **Figure 4-1**.

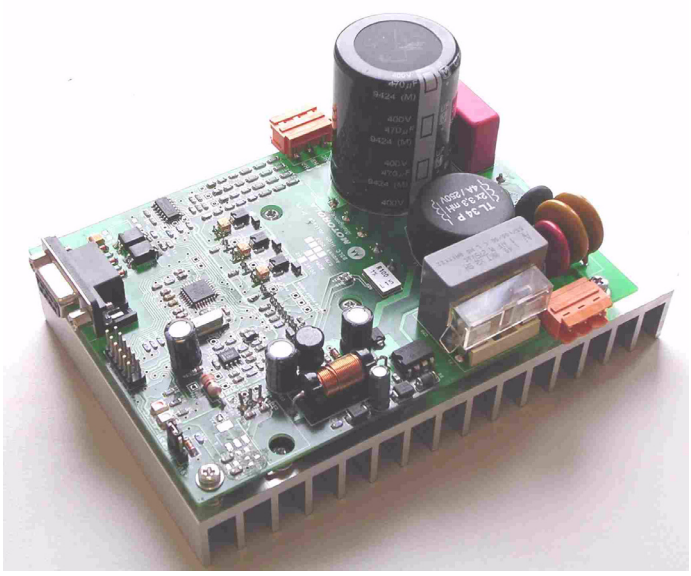


Figure 4-1. The High-voltage BLDC Drive

The drive provides a ready-to-use reference design as well as a ready-made software development platform for fractional horsepower BLDC motors. The rotor position needed for controlling the motor commutations is measured using the back-EMF signals. This control technique omits the position sensors, allowing reduced system cost and increased reliability to be achieved.

The high-voltage BLDC drive in comprises several system blocks (see **Figure 4-2**).

- Controller block
- Serial interface
- Back-EMF zero-crossing comparators
- Power supply energizing the inverter
- Power inverter
- Switched mode power supply

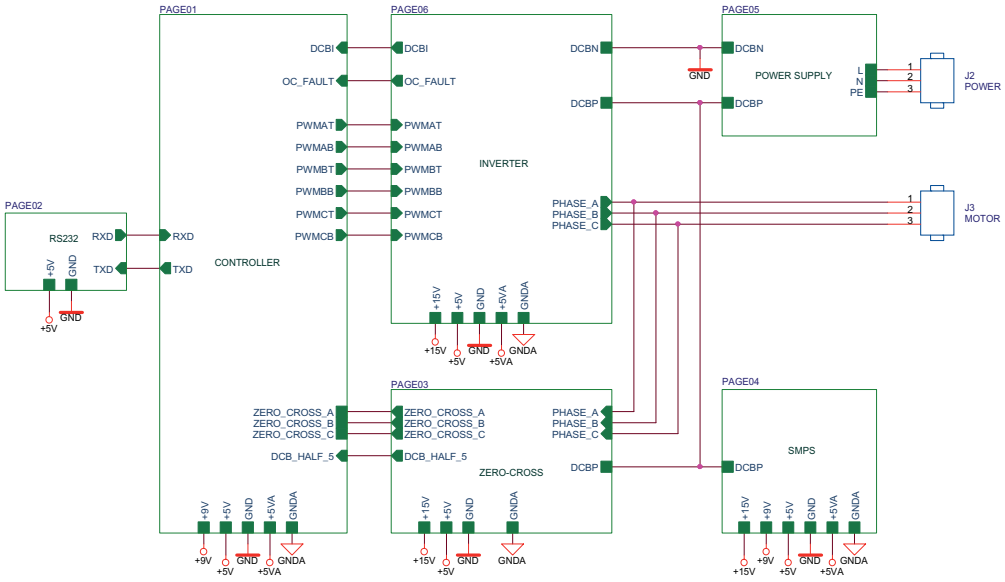


Figure 4-2. BLDC High-voltage Drive Blocks

Table 4-1 provides the electrical characteristics of the high-voltage BLDC drive, at 25°C with a 165 VDC power supply voltage.

Table 4-1. Electrical Characteristics of Drive

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
Analog output range	V_{Out}	0	—	5.0	V
Bus current sense voltage	I_{Sense}	—	610	—	mV/A
Bus voltage sense voltage	V_{Bus}	—	2.49	—	mV/V
Peak output current	I_{PK}	—	—	4.1	A
Total power dissipation	P_{diss}	—	—	100	W

4.2 System Modules

4.2.1 The Controller block

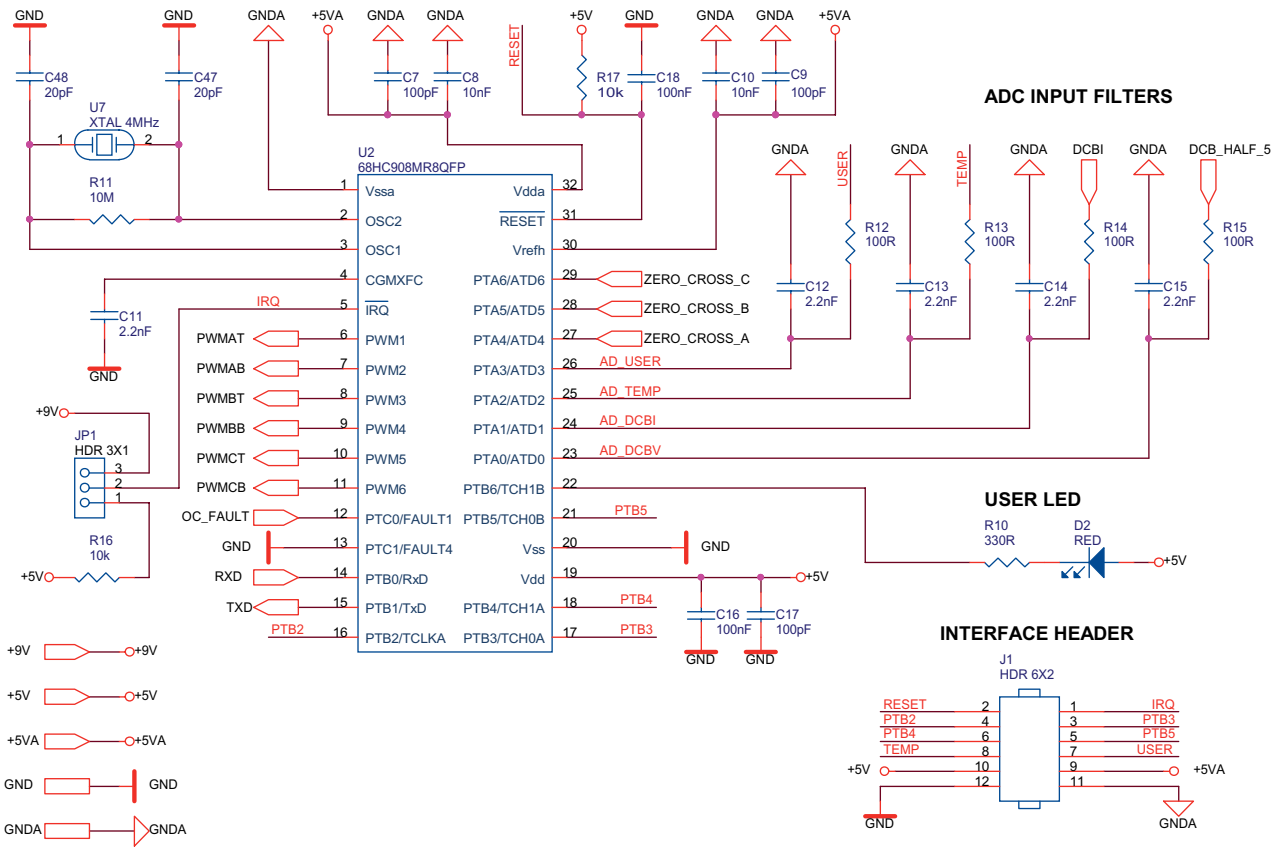


Figure 4-3. The Controller Block

The heart of the drive is the controller. It takes advantage of the features of Motorola's MC68HC908MR8 microcontroller, which has been designed specially for motor control applications.

The back-EMF sensorless control approach requires the sensing of the back-EMF voltage signal measured on the unpowered motor phase. The zero-crossing signals generated by the low-cost comparators give the information needed for BLDC motor commutation control. The signals

are connected to PTA4-PTA6 (GPIO inputs) and read by the control software.

The other feedback signal needed to perform the application control is the DC-bus voltage, DC-bus current fed to ATD0 and ATD1.

After the control algorithms are performed, the calculated duty cycles are sent to the PWM module, which generates the control signals for the inverter module.

Communication with other systems, such as PCs or master controllers, are done using the SCI interface. Because the control circuits are powered from unisolated SMPSs, the isolation of the SCI interface is provided by optocouplers.

Protection against drive faults is done using the FAULT1 input, which is connected to the power module fault output.

The state of the drive is indicateded by the LED on PTB6.

Spare pins that are not used for the motor control are terminated on the board expansion header J1, and can be used to connect a user interface (see **Table 4-2**).

Table 4-2. J1 Expansion Header

Pin number	Signal
1	IRQ
2	RESET
3	PTB3
4	PTB2
5	PTB5
6	PTB4
7	ATD3
8	ATD2

To enter monitor mode, the VDD voltage connected to the IRQ pin can be selected by a jumper on the JP1 header (**Table 4-3**).

Table 4-3. JP1 Header

Jumper position	IRQ voltage
1-2	+9V
2-3	+5V

4.2.2 RS232 Interface

RS-232 serial communication is provided by the circuit in **Figure 4-4**. It is optically isolated for safety and is suitable for communication rates up to 9600 baud.

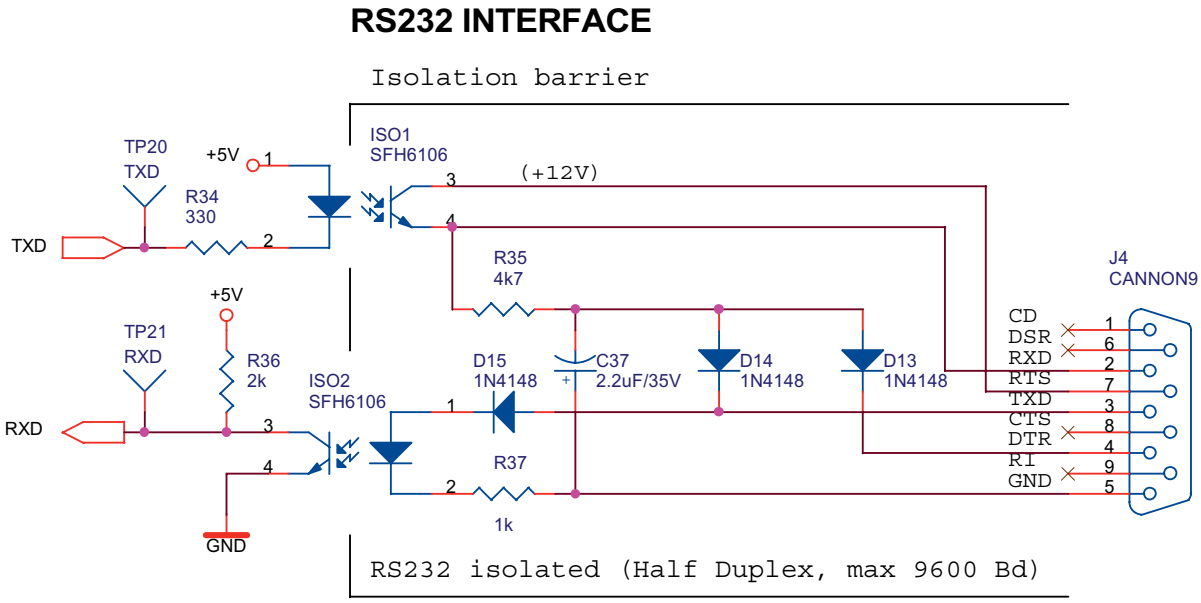


Figure 4-4. RS232 Interface

The EIA RS-232 specification states that signal levels can range from ±3 volts to ±25 volts. A mark is defined as a signal that ranges from -3 volts to -25 volts. A space is defined as a signal that ranges from +3 volts to +25 volts. Therefore, to meet the RS-232 specification, signals to and from a terminal must pass through 0 volts as they change from a mark

to a space. Breaking the isolated RS-232 circuit into input and output sections makes explanation of the circuit simpler.

The input interface is through optocoupler ISO2. To send data from a PC through ISO2, it is necessary to satisfy the SCI input on the MR8. In the idle condition, the SCI input must be at a logic 1. To accomplish this, the transistor in ISO2 must be off. The idle state of the transmit data line (TXD) on the PC serial port is a mark (-3 V to -25 V). Therefore, the diode in ISO2 is off and the transistor in ISO2 is off, providing a logic 1 to the SCI input. When the start bit is sent to the SCI from the PC's serial port, the PC's TXD changes from a mark to a space (+3 V to +25 V), forward biasing the diode in ISO2. Forward biasing the diode in ISO2 turns on the transistor in ISO2, providing a logic 0 to the input of the SCI. Simply stated, the input half of the circuit provides input isolation, signal inversion, and level shifting from the PC to the MR8's SCI port. An RS-232 line receiver, such as an MC1489, serves the same purpose without the optoisolation function.

To send data from the MR8 control board to a PC serial port input, it is necessary to satisfy the PC's receive data (RXD) input requirements. In an idle condition, the RXD input to the PC must be at mark (-3 V to -25 V). The data terminal ready output (DTR) on the PC outputs a mark when the port is initialized. The request to send RTS output is set to a space (+3 V to +25 V) when the PC's serial port is initialized. Because the interface is half-duplex, the PC's TXD output is also at a mark, as it is idle. The idle state of the transmit data line (TXD) on the MR8's SCI is a logic 1. The logic 1 out of the SCI's output port forces the diode in ISO1 to be turned off. With the diode in ISO1 turned off, the transistor in ISO1 is also turned off. The junction of D14 and D15 are at a mark (–3 V to –25 V). With the transistor in ISO1 turned off, the input is pulled to a mark through current limiting resistor R35, satisfying the PC's serial input in an idle condition. When a start bit is sent from the MR8's SCI port to the output of the MR8's SCI, output transitions to a logic 0. That logic 0 turns on the diode in ISO1, thus turning on the transistor in ISO1. The conducting transistor in ISO1 passes the voltage output from the PC's RTS output, that is now at a space (+3 V to +25 V), to the PC's receive data (RXD) input. Capacitor C37 is a bypass capacitor used to "stiffen" the mark signal. The output half of the circuit. provides output isolation, signal inversion, and level shifting from the MR32's SCI output port to the

PC's serial port. Again, an RS-232 line driver, such as an MC1488, serves the same purpose without the optoisolation function.

4.2.3 Zero-crossing interface

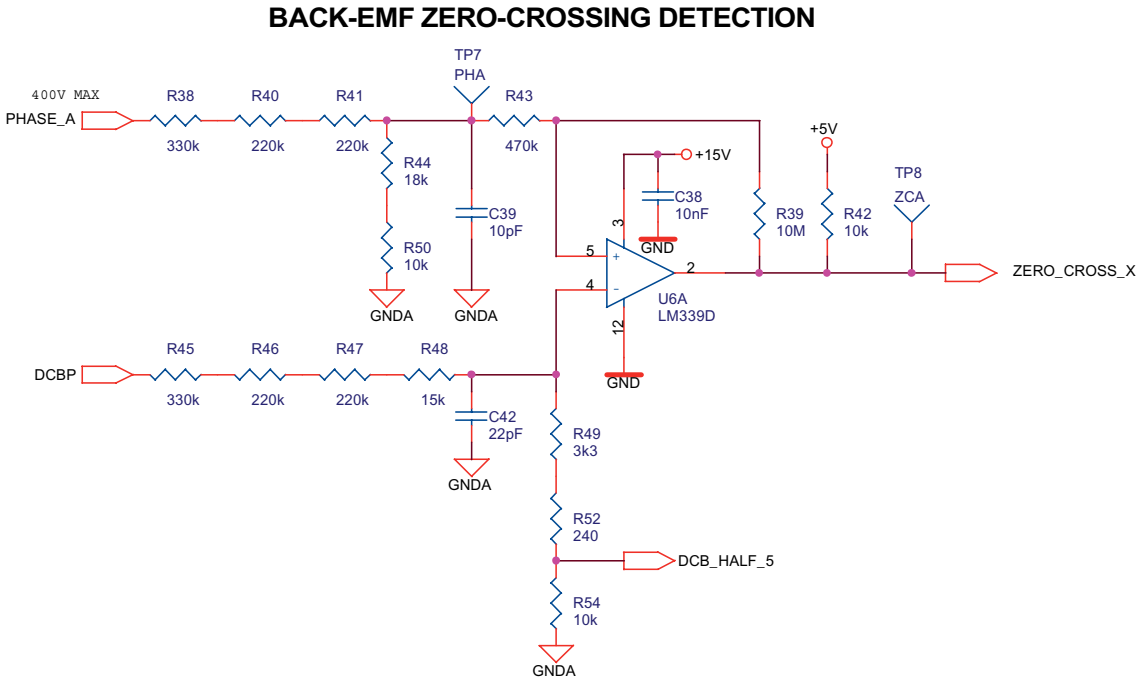


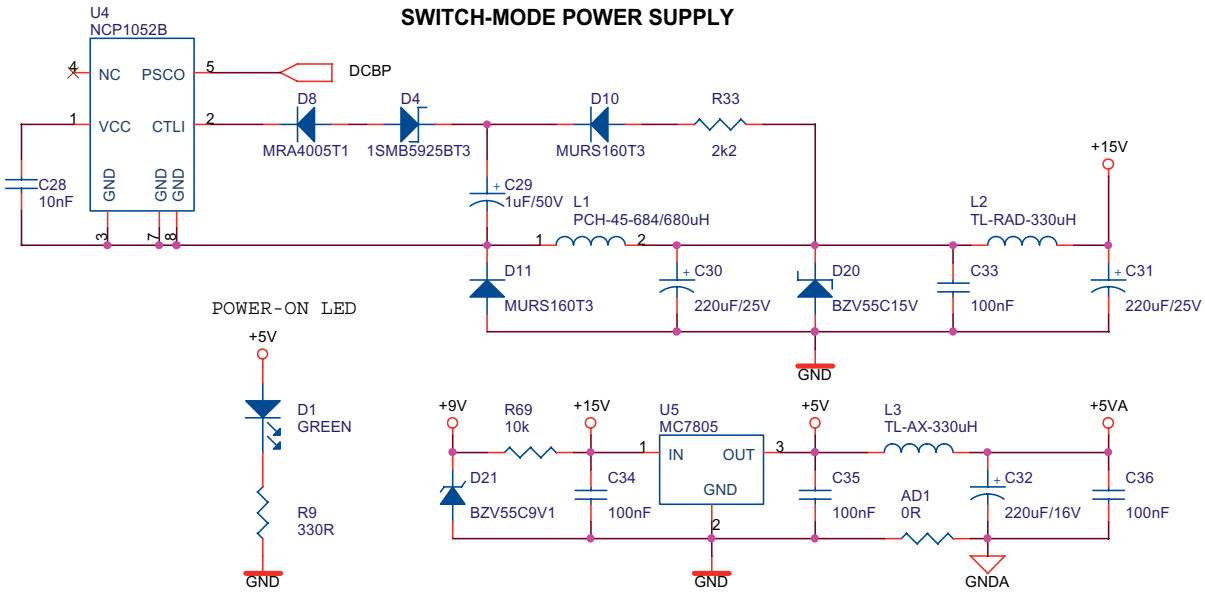
Figure 4-5. Back-EMF Zero-crossing Detection Circuit

Back-EMF and zero-crossing signals are included to support sensorless algorithms for BLDC motors. Referring to **Figure 4-5**, which shows circuitry for phase A, the raw phase voltage is scaled down by a voltage divider consisting of R38, R40, R41, R44 and R50. A zero-crossing signal is obtained by comparing the motor phase voltage with one-half of the motor bus voltage. Comparator U6A preforms this function, producing zero-crossing signal ZERO_CROSS_X.

The output of the low-voltage LM339D is connected to the GPIO input of the controller module.

Except for the zero-crossing signals, additional information about the DC-bus voltage level is provided.

4.2.4 Switched Mode Power Supply



Analog and digital ground should be connected in one suitable place on PCB. In BLDC High Voltage board it is connected through 0 Ohm resistor (AD1) on bottom side of the PCB close to VSSA

Figure 4-6. Switched Mode Power Supply

The low-cost switched mode power supply has been designed to cover the power consumption of the active components on the board. It uses the monolithic high-voltage regulator NCP1052-B from ON Semiconductor. It can supply 15 V @ 250 mA.

Table 4-4. Supplied Voltages

Voltage	Supply
+15V	IPM (intelligent power module) drivers
+9V	Programming voltage for MR8
+5V	Digital power supply
+5VA	Analog power supply

4.2.5 DC-bus Power Supply

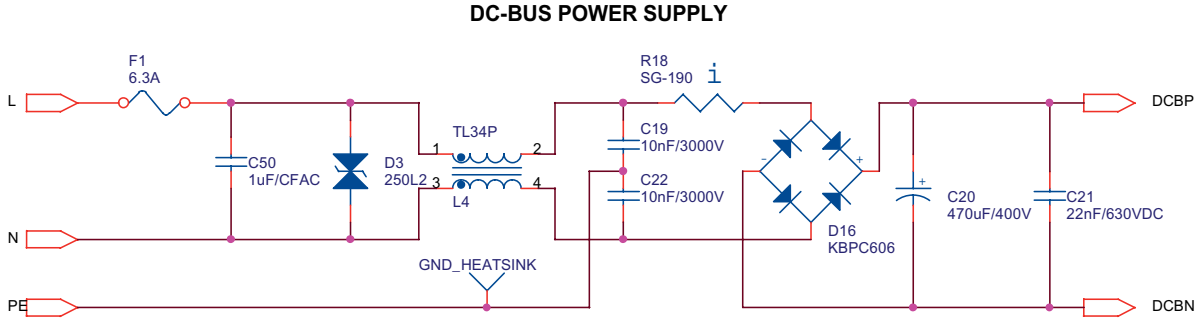


Figure 4-7. DC-bus Power Supply

The 3-phase inverter PS21323 is supplied from the DC voltage. This voltage is rectified from the AC line using the standard bridge rectifier D16 and bulk DC-bus capacitor C20. The small polypropylene capacitor C21 must be added to minimize the ESR (equivalent series resistance) of the electrolytic capacitor C20. The current surge caused by initial charging of the DC-bus capacitor C20 during device power-on is minimized by the thermistor R18 (SG-190). The line over-voltage surge protection is provided by the D3 varistor. The input electromagnetic interference filter is provided by capacitor C50, C19, C22 and common mode choke TL34P. The fast-blowing fuse provides the device with protection against short-circuits.

4.2.6 Current Sensing and Over-current Protection

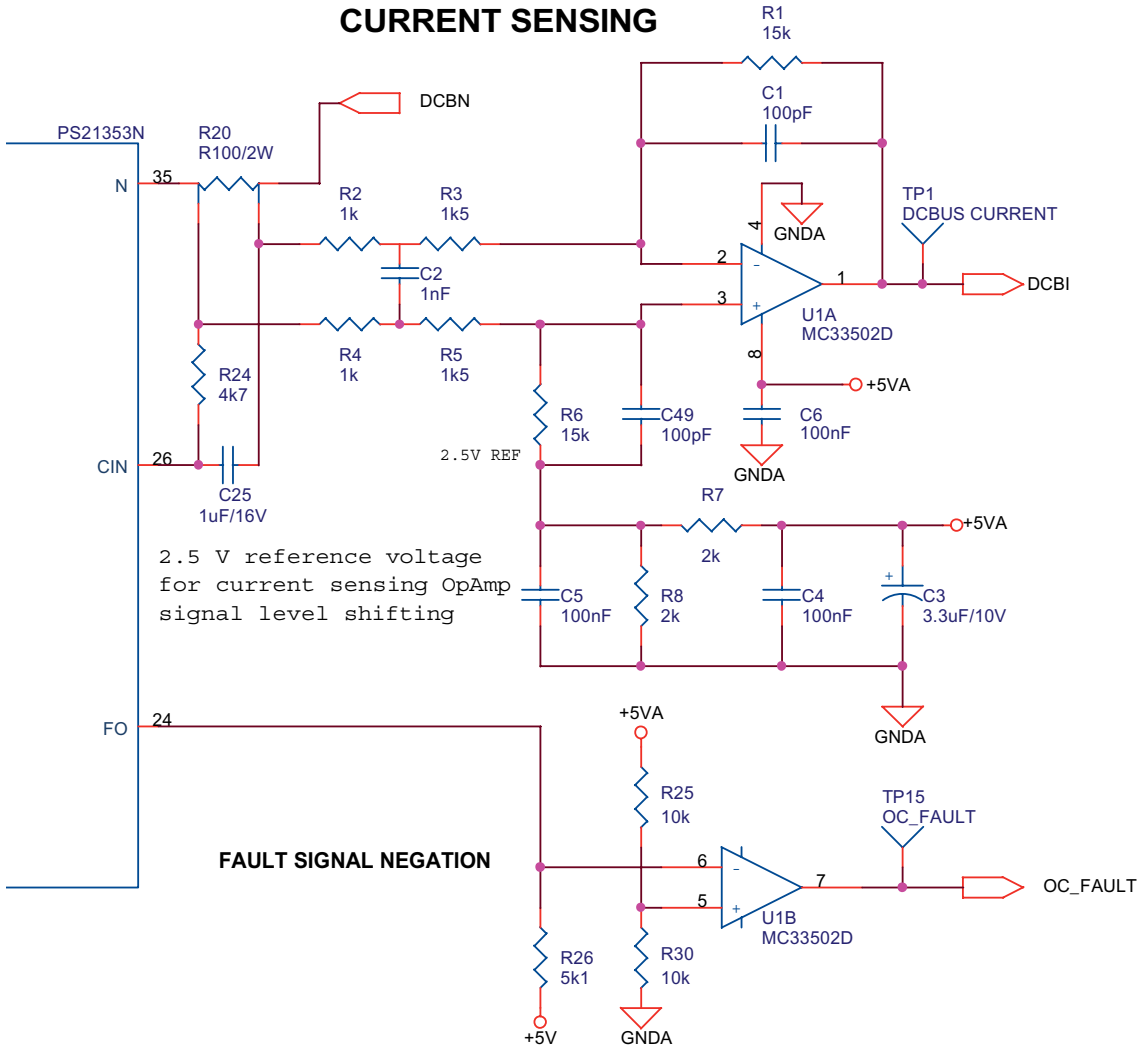


Figure 4-8. Current Sensing

Bus current is sampled by resistor R20 and amplified by the operating amplifier U1A. This circuit provides a voltage output suitable for sampling with ATD (analog-to-digital) inputs. An MC33502D is used for the differential amplifier. With R1=R6, R2=R4 and R3=R5, the gain is given by:

A = R1/(R2+R3)

The output voltage is shifted up by 2.5V to accommodate both positive and negative current swings. A +/- 410 mV voltage drop across the sense resistor corresponds to a measured current range 0 V – 5 V

The DIP-IPM short-circuit protection accepts the voltage value across the external current sensing resistance into the control IC as SC trip (reference voltage) and operates by internally interrupting the output. The scheme for setting the value of external current sensing resistance is shown below:

Selecting shunt resistance:

Current sensing resistance value:

$R = V_{SC(ref)}/SC$

where $V_{SC(ref)}$ is the SC reference voltage (trip level) of the control IC. The SC trip level maximum value must be set below the minimum value of the IGBT saturation current, which is 1.7 times the rating current. The rating current of the PS21353-N is 10 Amps. The minimum $V_{SC(ref)}$ short-circuit trip level voltage is 0.42 V.

The resultant value of the current sensing resistor for PS21353-N is then 25 mΩ. The PS21353-N IPM has been used in the reference design because of better availability and price conditions in lower volumes than the PS21352-N or PS20351-N. In this reference design, the 100 mΩ resistor has been chosen.

Filter Circuit Setting for Short-circuit operation:

The RC filter created by R24 and C25 prevents the SC (short-circuit) protection from malfunction caused by noise on the shunt resistor. The RC filter circuit immediately interrupts short-circuit currents due to its characteristic. To set the time constant, the IGBT ability should be considered. The RC time constant is set to 2 µs. (Refer to the detailed description in **References 11**, **12**).

If an over current fault or other power module fault occurs, the 1.8 ms fault pulse is generated on the FO pin of the module. Because the logic is different than on the MR8 the signal inversion is performed on a spare gate U1B of the operating amplifier.

4.2.7 Power Inverter

The power inverter has been designed using the Mitsubishi Mini-DIP IPM PS21353-N. This module is targeted at low-power inverters. Documentation for this module and recommendations for its use can be found in **References 11** and **12**.

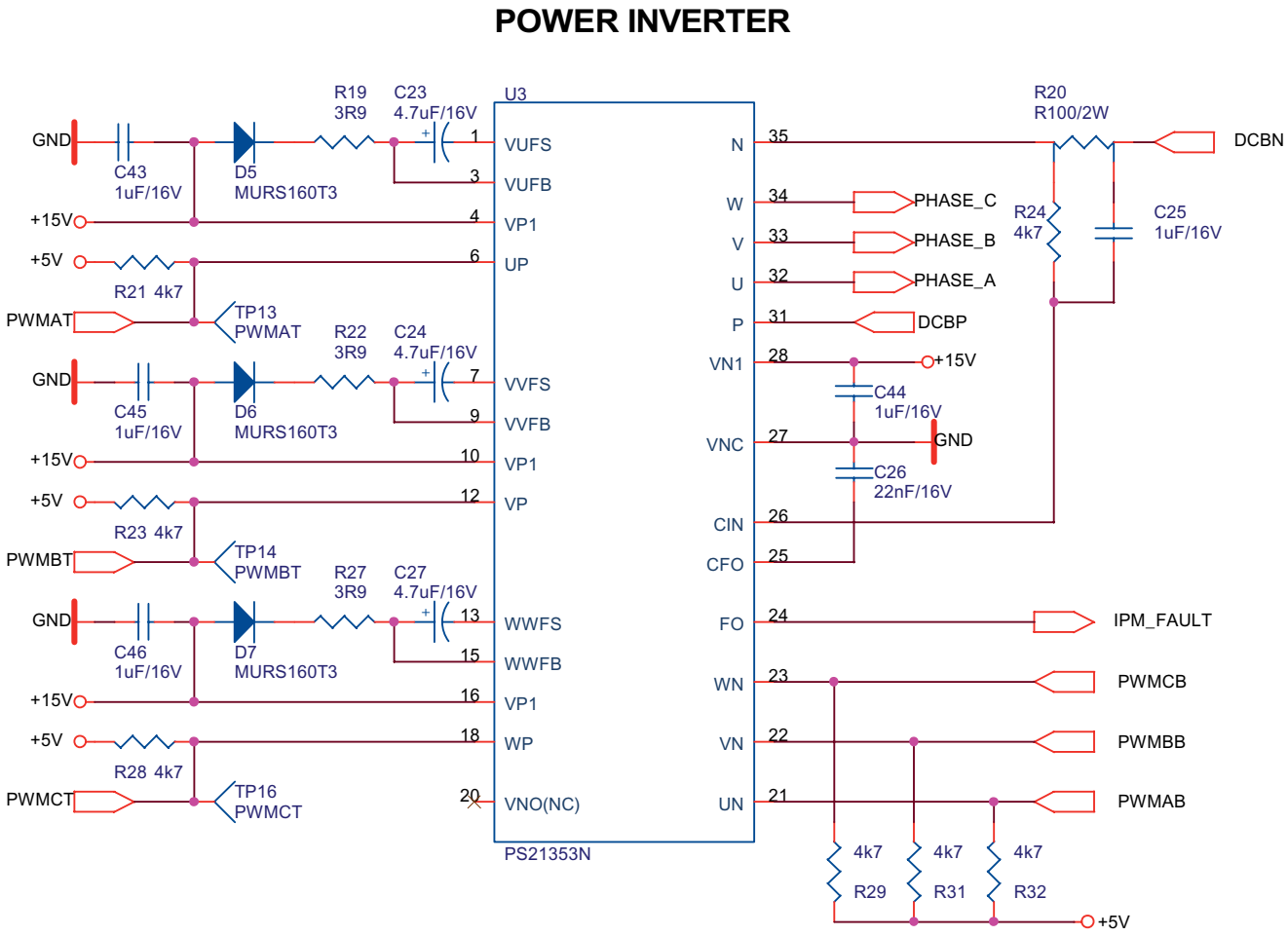


Figure 4-9. Power Inverter with Mini-DIP IPM

4.3 The BLDC High-voltage Motor

Although the system can drive the compressor with a BLDC motor, for SW evaluation the BLDC Motor-Brake system (SM40V + SG40N) has been chosen. It is supplied as ECMTRHIVBLDC and can be ordered from the Motorola web site. The motor is shown in **Figure 4-10** and its parameters are given in **Table 4-5**.

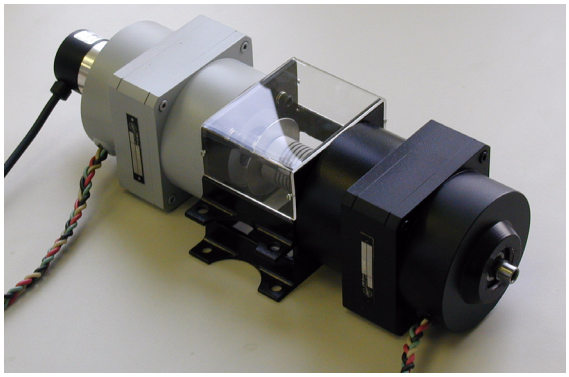


Figure 4-10. The BLDC High-voltage Motor SM40

Table 4-5. Motor Specifications

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

Section 5. Software Design

5.1 Introduction

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

- Data flow
- Main software flowchart
- State diagram

For more information on the control technique used, see 3.2 Control Technique Used.

5.2 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**.

5.2.1 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	8 flags	System variable #1
Speed_Min_U8	U8	< 0; Speed_Range_Max_RPM)	Minimum speed (system units)
Sp_Input	U8	< 0; 255>	Speed input variable used for required speed calculation
Coef_Speed_Inp	U8		Coefficient 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]	Minimum 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

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

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

5.2.2 Process Measurement

The process provides measurement of analog values using analog to digital conversion. 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 5.4 State Diagram.

5.2.3 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 **5.4.3 Stand-by** and **5.4.8 Fault State**.

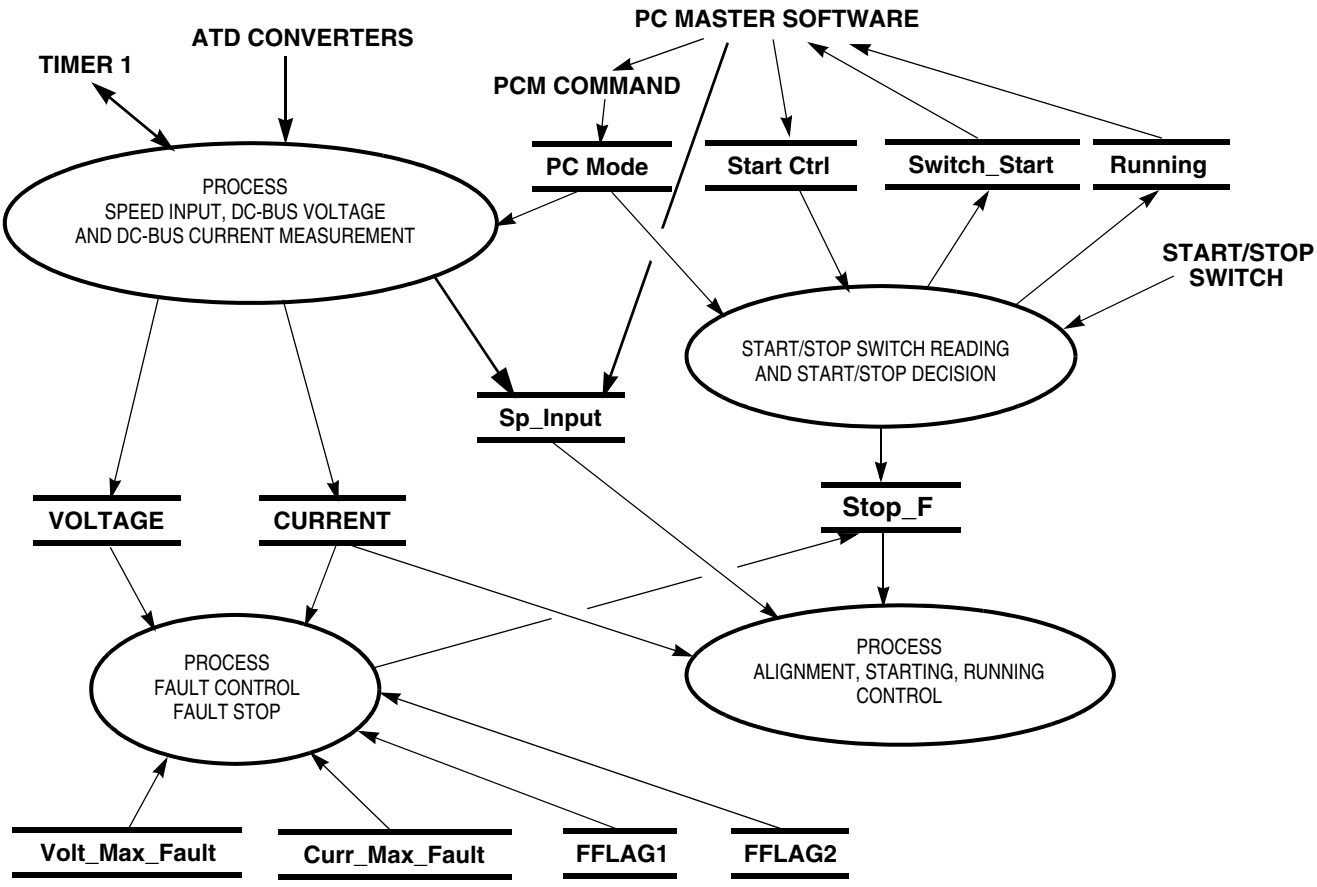


Figure 5-1. Main Data Flow, Part1

5.2.4 Process Fault Control Fault Stop

The process provides fault control and fault stop as described in **5.4.8 Fault State**, **5.4.3 Stand-by**, **5.4.4 Align State**, **5.4.5 Back-EMF Acquisition State**, and **5.4.6 Running State**.

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

5.2.5 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 **5.2 Data Flow** and **Figure 5-6**.

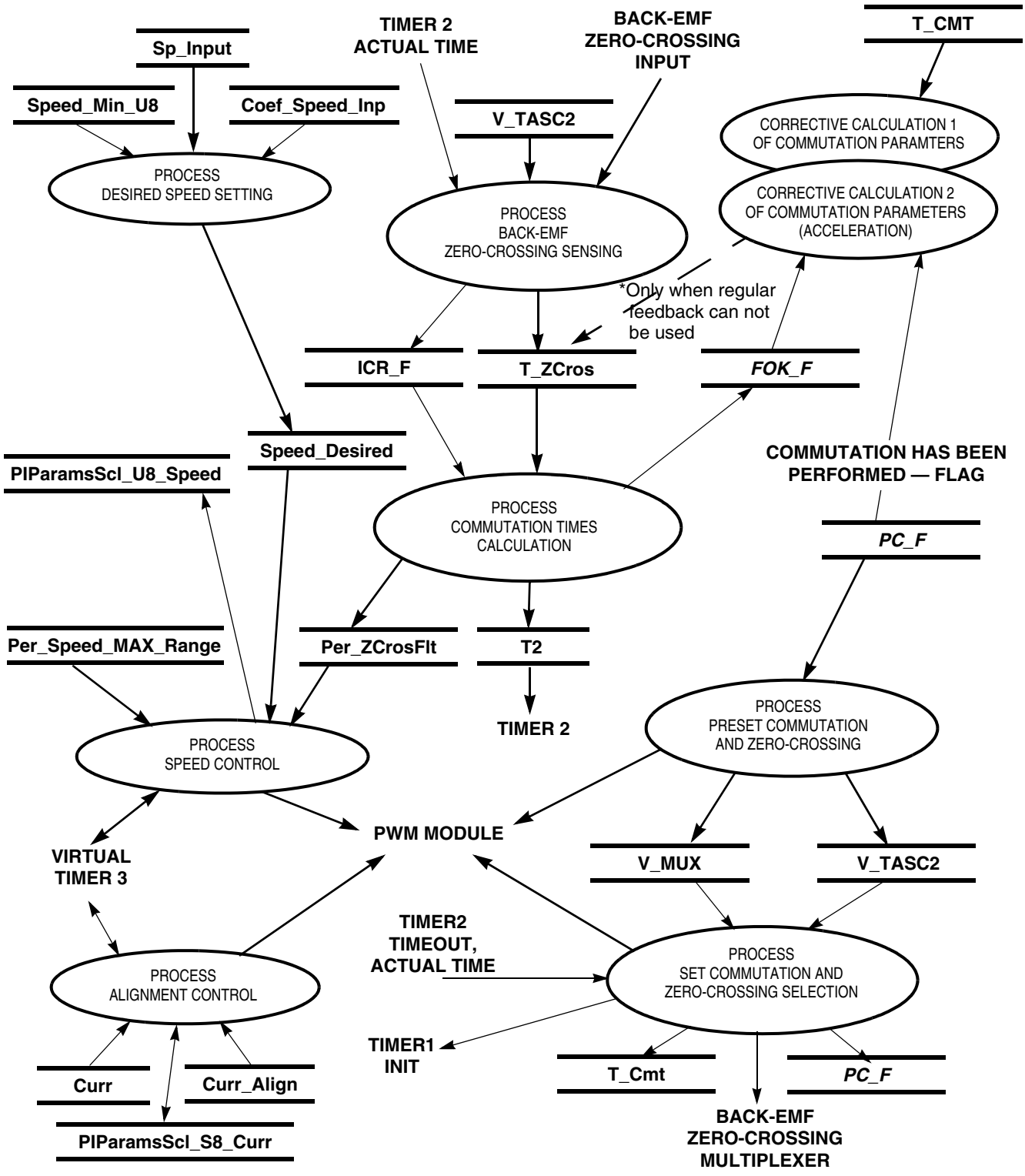


Figure 5-2. Main Data Flow, Part 2: Alignment, Starting, Running Control

5.2.6 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 set Timer 2 with variable **T2**. These calculations are described in 3.2.1.5 Starting: Commutation Time Calculation and **3.2.1.3 Running: Commutation Time Calculation**.

5.2.7 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

5.2.8 Process Speed Control

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

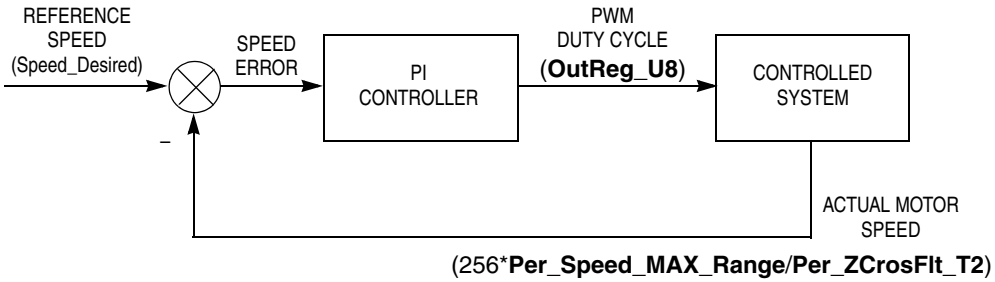


Figure 5-3. Closed Loop Control System

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

The actual motor speed is calculated from the 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 to compensate for the error. The speed regulator parameters, 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 the regulator output, **OutReg_U8**. The maximum duty cycle is at **OutReg_U8** = 255. The implementation is described in 5.5.3 BLDC Speed Control and Calculation.

5.2.9 Process Alignment Control

The process alignment control controls the current, Curr, using the PI regulator during alignment state (see **5.4 State Diagram**). The DC-bus current is regulated to required value Curr_Align. The current regulator parameters, 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 timer 1, with the constant **PER_CS_T1_US**.

5.2.10 Processes Commutation and Zero-crossing Preset and Set

The processes commutation and zero-crossing preset and set provide the BLDC commutation and zero-crossing selection. Here the BLDC commutation means generation of the 6-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 3.1.4.2 Indirect Back-EMF Sensing

and **3.1.5 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 5.5.2 BLDC Commutation and Zero-crossing Selection.

5.3 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, to be able to measure negative and positive current values. The status registers are initialized and the PWM generator is started. Also, timer 1 is started at the correct moment to be synchronized with the PWM generator. This way, the current measurement is executed at the defined moment of the PWM signal.

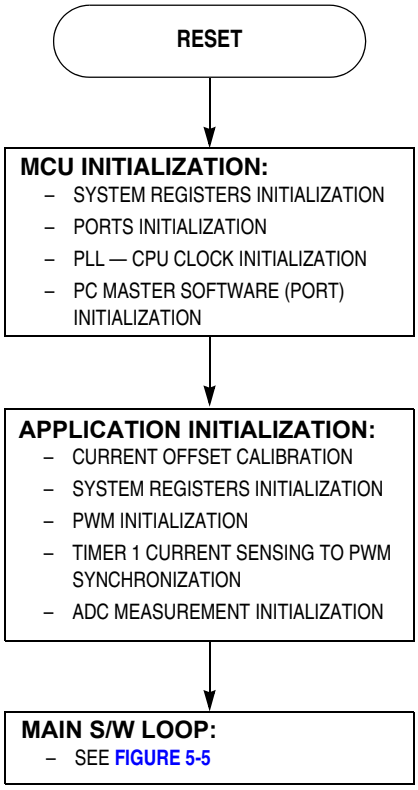


Figure 5-4. Main Software Flowchart

1. 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.

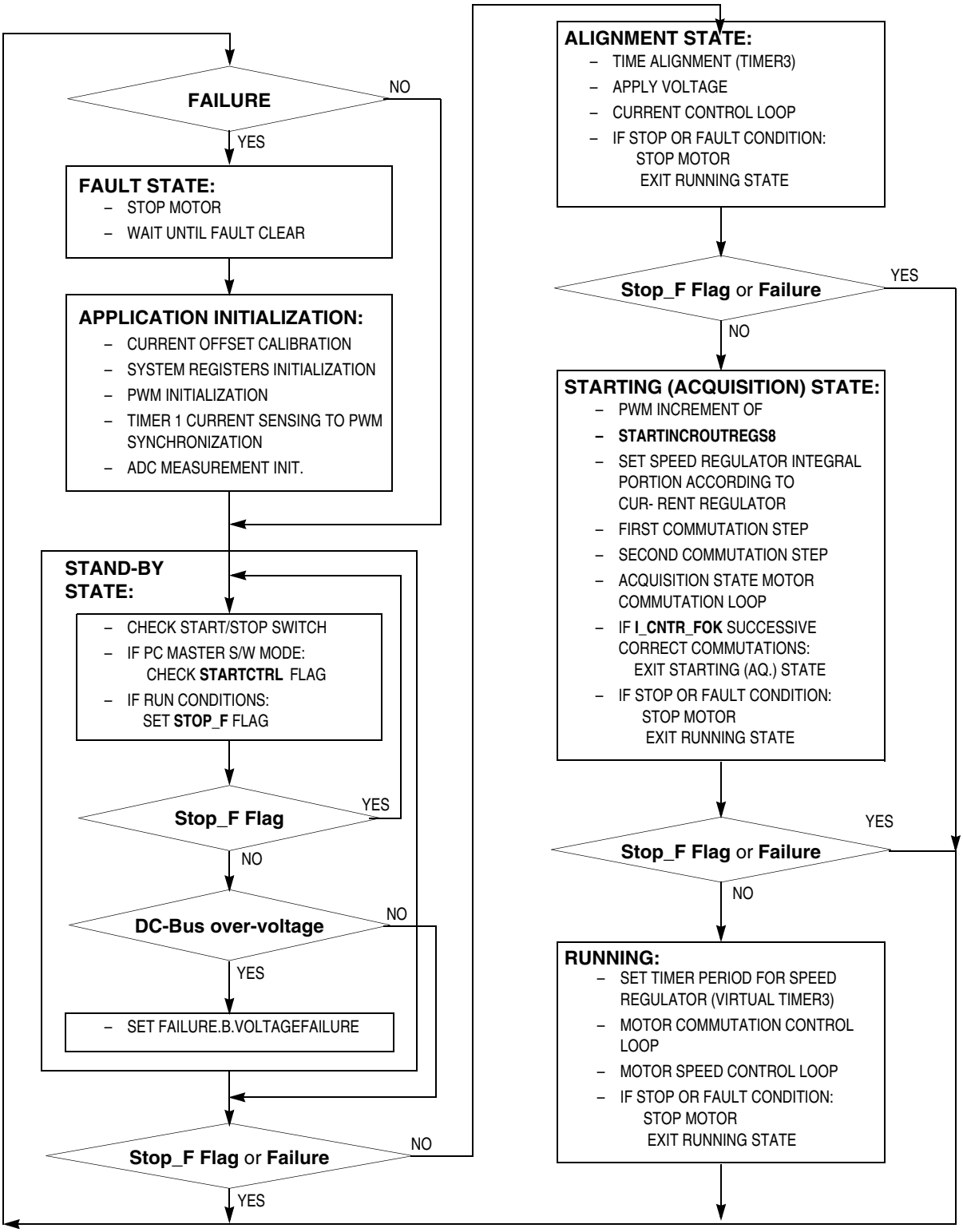


Figure 5-5. Main Software Flowchart: Main Software Loop

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 **3.2.1 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.

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 left only 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 the 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**.

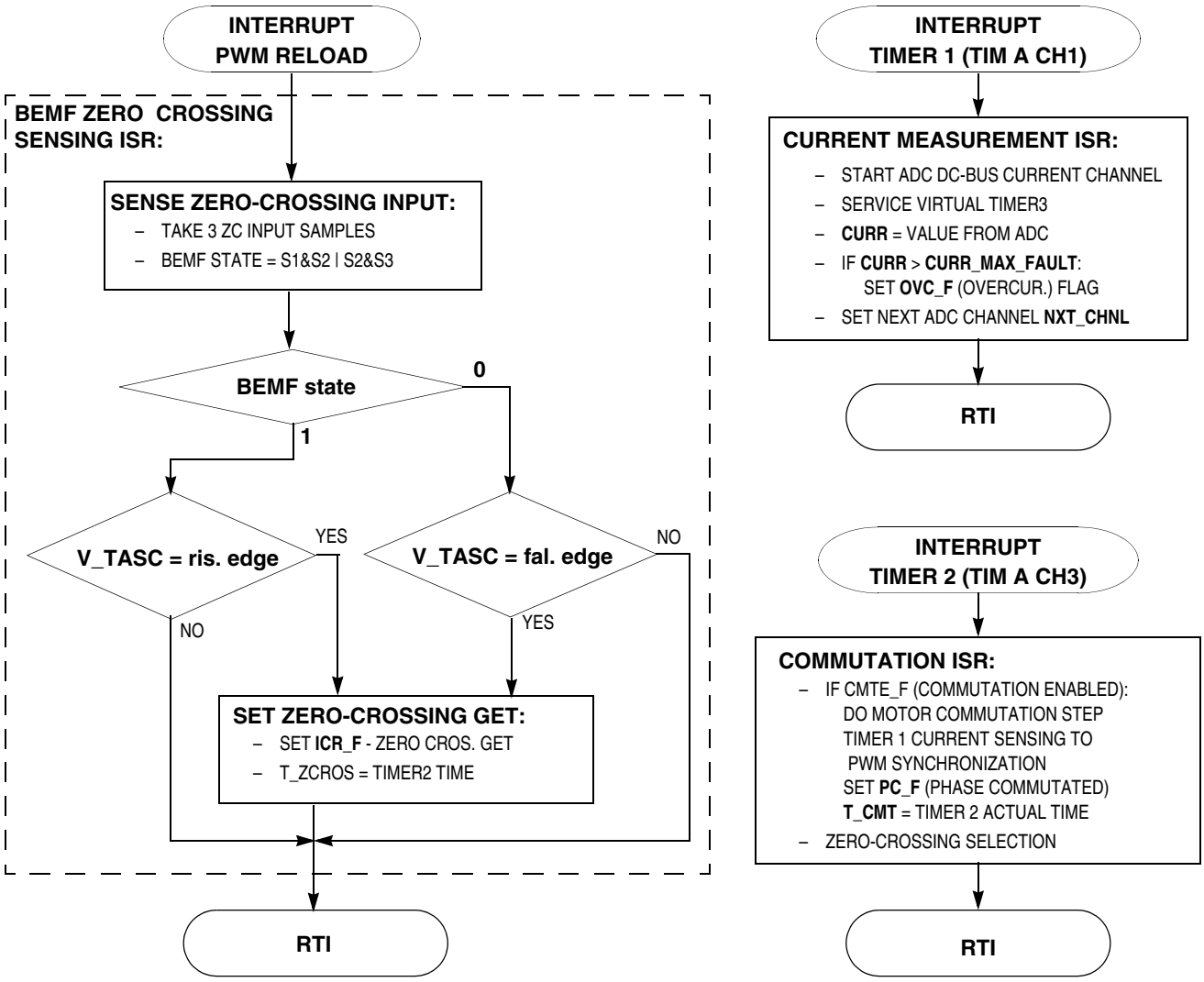


Figure 5-6. Software Flowchart: Interrupts

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**.

5.4 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.

5.4.1 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.

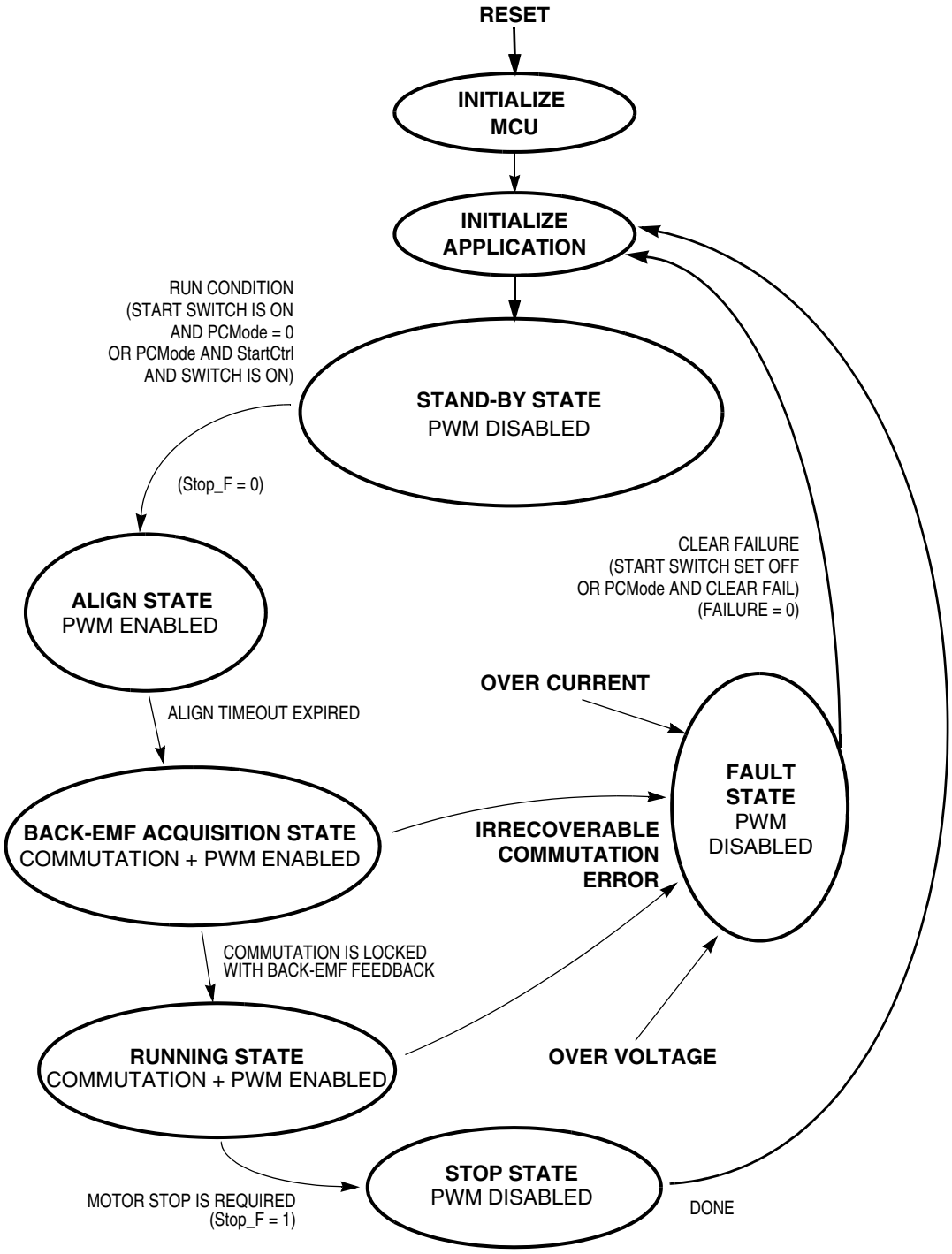


Figure 5-7. Application State Transitions

5.4.2 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, output compare (OC), pulse width modulation (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 are initialized
- Ready LED is turned ON

and the state is exited.

5.4.3 Stand-by

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

5.4.3.1 Current Measurement and Calibration When PWM is OFF

Before testing the start switch, the 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 2.5 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 the fault state:

5.4.3.2 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 the start condition is valid, then Strt_F is set, Stop_F is cleared, and the next state is entered.

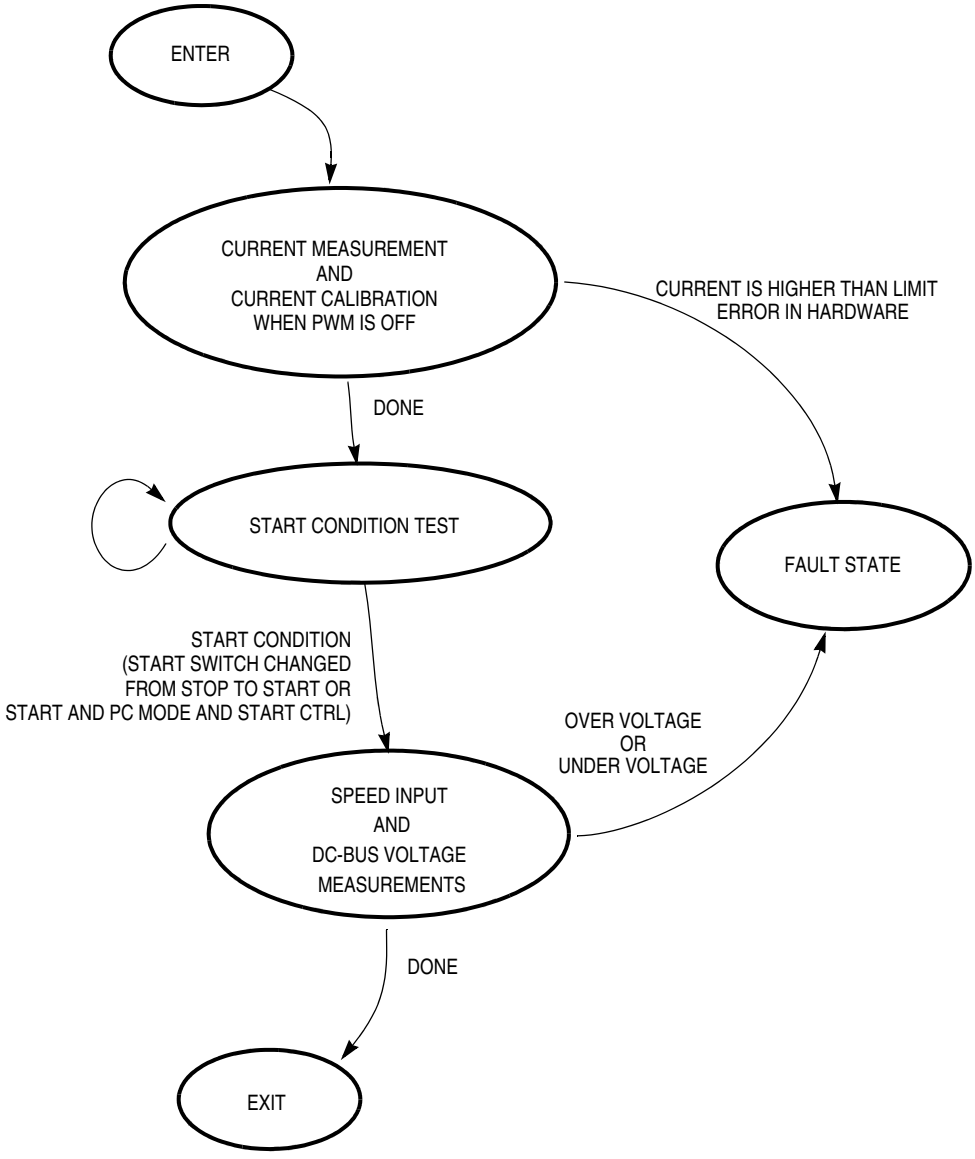


Figure 5-8. Stand-by State

5.4.3.3 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 turning on. Where the DC-bus voltage is not within the limits, the control flow enters the fault state.

5.4.4 Align State

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

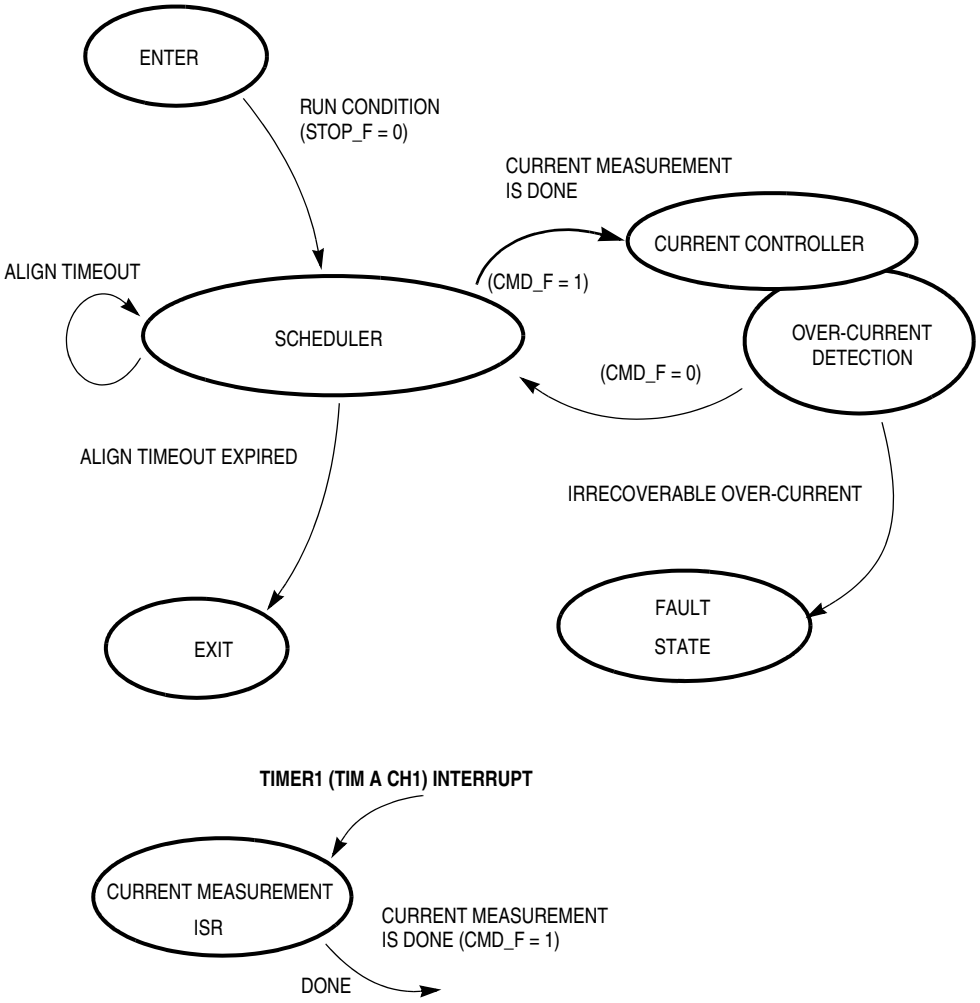


Figure 5-9. Align State

5.4.4.1 Scheduler

The scheduler handles the state transitions in the align state. The DC-bus current measurement is done in the OC interrupt service routine, to keep synchronization with the PWM cycle. After the 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 timeout (software timer 3) of this state is defined in the software by the constants: **PER_T_ALIGN** and **PER_BASE_T3_ALIGN**.

5.4.4.2 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.

5.4.4.3 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.

5.4.4.4 Over-current Detection

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

5.4.5 Back-EMF Acquisition State

The back-EMF acquisition state provides the functionality described in **3.2.1.4 Starting (Back-EMF Acquisition)** and **3.2.1.5 Starting: Commutation Time Calculation**. **Figure 5-10** shows the state transitions for the state.

5.4.5.1 First Commutation

After the align state timeout 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 align state.

The calculation of the commutation time is explained in 3.2.1.5 Starting: Commutation Time Calculation.

5.4.5.2 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 prepared for when the commutation interrupt performs the next commutation.

The calculation of the commutation time is explained in 3.2.1.5 Starting: Commutation Time Calculation.

5.4.5.3 Measurements Handler

As explained in **5.4.4.1 Scheduler** and **Over-current Detection**, the DC-bus current is scanned and the 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 also. When this state is finished, the values of DC-bus current, DC-bus voltage, and speed command are updated.

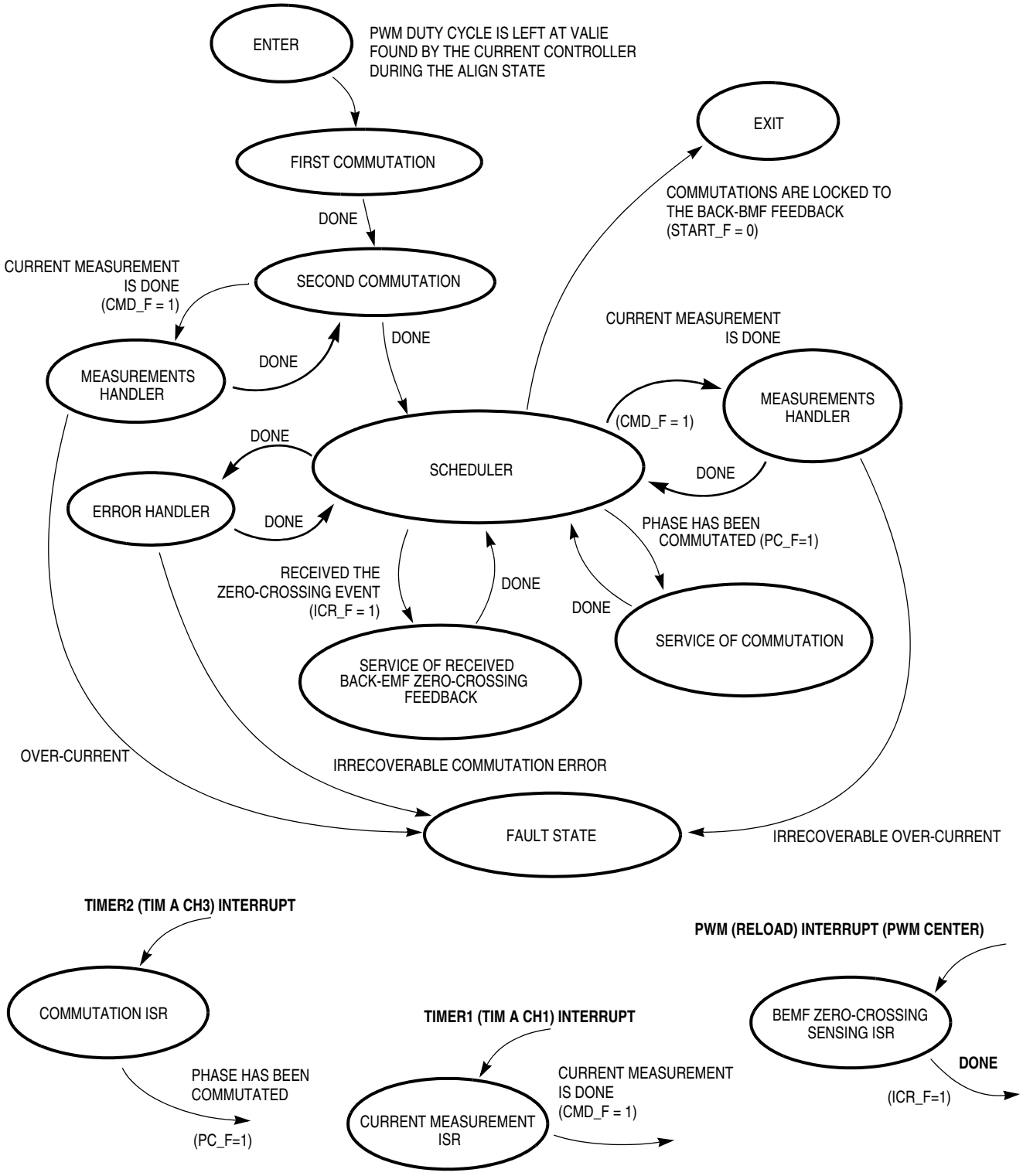


Figure 5-10. Back-EMF Acquisition

5.4.5.4 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 **5.2.10 Processes Commutation and Zero-crossing Preset and Set**.

5.4.5.5 Service of Received Back-EMF Zero-crossing

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

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 **3.2.1.5 Starting: Commutation Time Calculation**.

5.4.5.6 BEMF Zero-crossing Sensing Interrupt Service Routine

This ISR (interrupt service routine) is used to evaluate the back-EMF zero-crossing. Back-EMF is evaluated here 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, 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.

5.4.5.7 Current Measurement Interrupt Service Routine

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

5.4.5.8 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 assessed as an irrecoverable commutation error, and the fault state is entered.

5.4.5.9 Measurement Handler

The measurement handler ensures 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 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 the fault state.

5.4.6 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 the PWM duty cycle. **Figure 5-11** shows the state transitions.

5.4.6.1 Measurements Handler

Explained in **5.4.5 Back-EMF Acquisition State**.

5.4.6.2 Service of Commutation

Explained in **5.4.5 Back-EMF Acquisition State**.

5.4.6.3 Service of Received Back-EMF Zero-crossing

Explained in **5.4.5 Back-EMF Acquisition State**. The difference is that the commutation parameters are recalculated with different constants (see **3.2.1.3 Running: Commutation Time Calculation**)

5.4.6.4 BEMF Zero-crossing Interrupt Service Routine

Explained in **5.4.5 Back-EMF Acquisition State**.

5.4.6.5 Current Measurement Interrupt Service Routine

Explained in **5.4.5 Back-EMF Acquisition State**.

5.4.6.6 Error Handler

Explained in **5.4.5 Back-EMF Acquisition State**.

5.4.6.7 Over-current

Explained in **5.4.5 Back-EMF Acquisition State**.

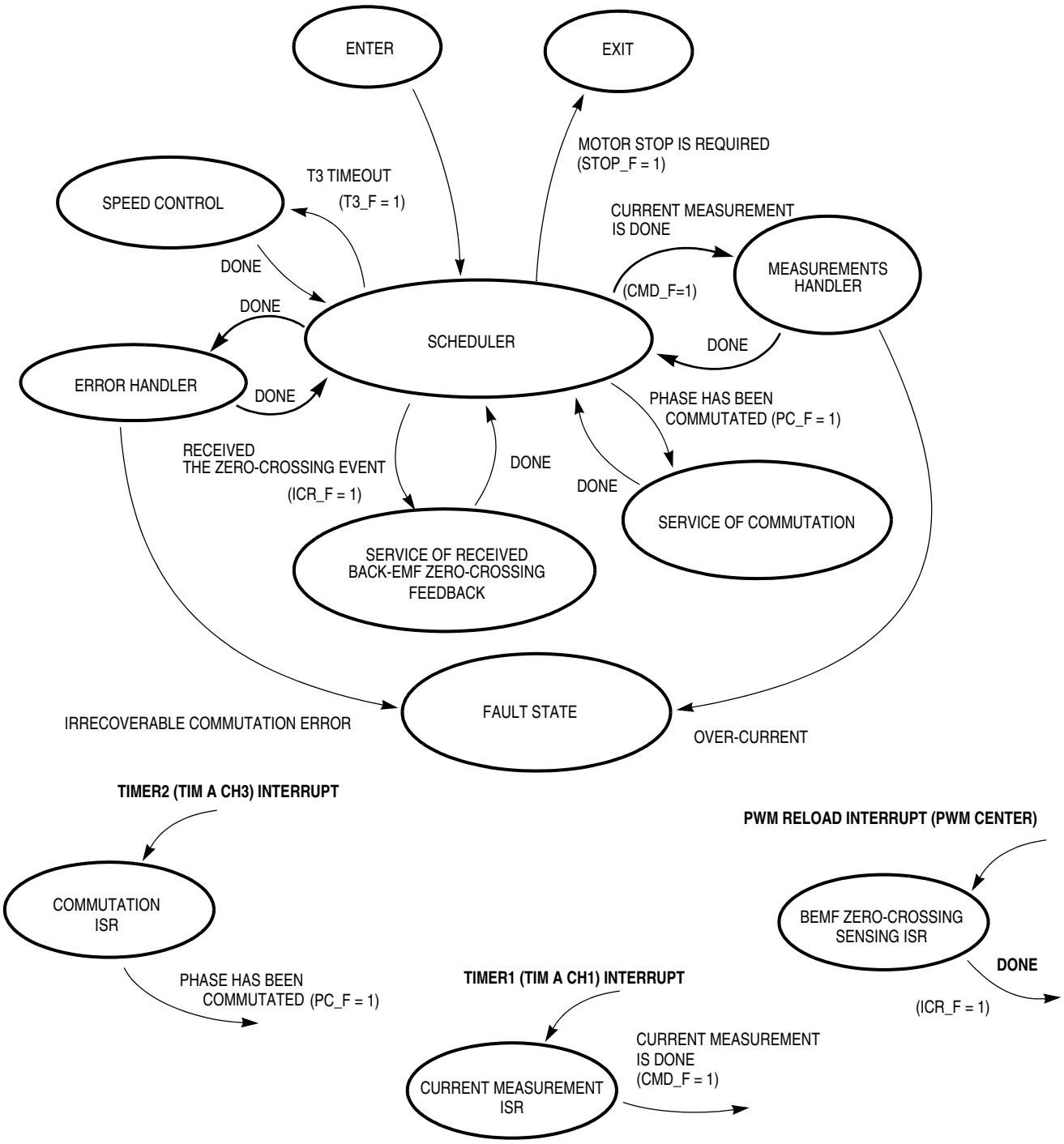


Figure 5-11. Running State

5.4.7 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**.

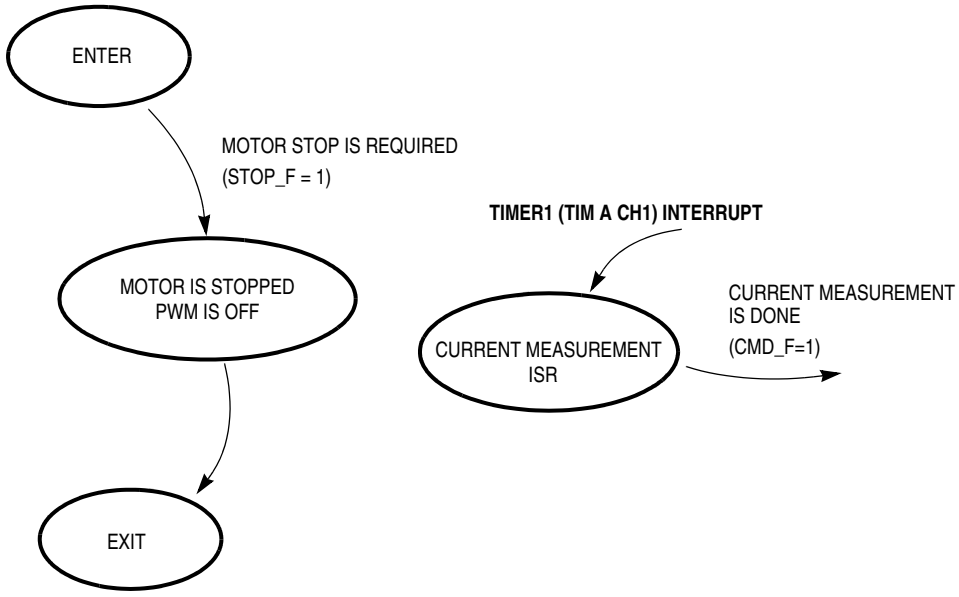


Figure 5-12. STOP State

5.4.8 Fault State

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

5.4.8.1 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**.

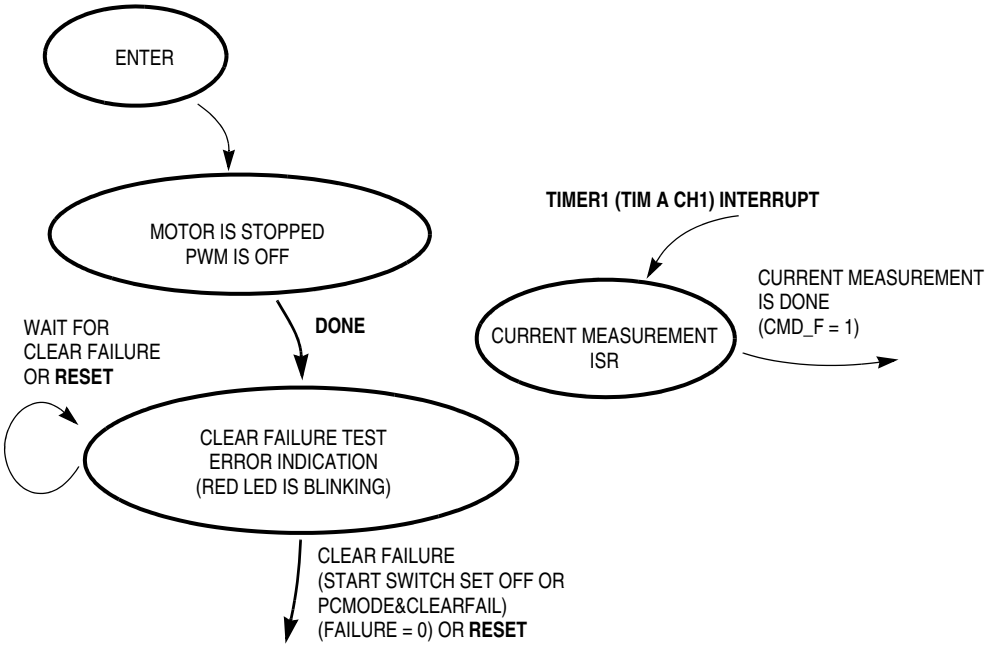


Figure 5-13. Fault State

5.5 Implementation Notes

5.5.1 Software Files

The software files and structure are described in **6.2.3 Software Setup**.

5.5.2 BLDC Commutation and Zero-crossing Selection

The required BLDC motor voltage system commutation is provided using the MC68HC908MR8 PWM block.

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

As shown in **5.2 Data Flow**, the commutation and back-EMF zero-crossing selection process comprises two steps:

1. Preset BLDC commutation and BEMF zero-crossing selection

2. Set BLDC commutation and BEMF zero-crossing selection

5.5.2.1 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 at logic 0. The commutation preset is accomplished by setting the buffered registers PVAL1H, PVAL1L through PVAL6H, PVAL6L. To preset the active PWM channel, the most significant bits of the dedicated PVALxH registers are set to 0. To preset the PWM channel to the logical 0, the most significant bits of the dedicated PVALxH registers are set to 1. This is due to the signature bit functionality as described in *68HC908MRxx Advance Information* (Motorola document order number MC68HC908MR8/D). This commutation preset is provided by the function **Commut()**. In **Commut()**, the function field **Set_PWM(P_Step_Cmt]()** calls one of the **Shft_PWM0..5()** functions according to the **P_Step_Cmt** value, so the PVALxH registers are set as required. This will not effect the PWM signals until the **LDOK** bit is set. Also, the back-EMF zero-crossing selection preset is provided in this function, by setting the **V_MUX** variable according to the **P_Step_Cmt** value.

5.5.2.2 Set BLDC Commutation and BEMF Zero-crossing Selection

The commutation set is provided in the timer 2 interrupt function **TIMACh3_Int()** in the **code_isr.c** file. The **PWMEN** is toggled and the **LDOK** bit set, to restart immediately 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 port F according to **V_MUX** variable.

5.5.3 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 must be taken into consideration.

5.5.3.1 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 optimum code using the MUL instruction.

5.5.3.2 PWM Duty Cycle

PWM duty cycle is set for all six PWM channels according to regulators output **OutReg_U8**. The maximum 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**.

5.5.4 Timers

Timer 1 and timer 2 are implemented using MC68HC08MR8 timers. Timer 3 is a software virtual timer using the time base of timer 1.

5.5.4.1 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 to set 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.

5.5.4.2 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 to set 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. 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

WARNING: ***This application operates in an environment that includes dangerous voltages and rotating machinery. The application power stage and optoisolation board are not electrically isolated from the mains voltage. They are live, and there is a risk of electric shock if they are touched.***

An isolating transformer should be used when operating from an AC power line. If an isolating transformer is not used, power stage grounds and oscilloscope grounds will be at different potentials, unless the oscilloscope is floating. Note that probe grounds, such as in the case of a floating oscilloscope, are subject to dangerous voltages. Take note of the following points and recommendations

- ***Switch off the high-voltage supply before moving scope probes or making connections.***
- ***Avoid inadvertently touching live parts, and use plastic covers where possible.***
- ***When the high voltage is applied, using only one hand to operate the test setup will reduce the possibility of electrical shock.***
- ***Avoid using the application in laboratory environments that have grounded tables or chairs.***
- ***Wear safety glasses, avoid ties and jewelry, use shields, and make use of personnel trained in high-voltage laboratory techniques.***
- ***The power module heatsink 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.***

6.1 Suitability Guide for Customer Application and Motor

6.1.1 Minimum Application Speed

It is well known that the back-EMF voltage is proportionally dependent on motor speed. As the sensorless back-EMF zero-crossing sensing technique is based on back-EMF voltage, it has some minimum speed limitations. Motor start-up is solved by the starting (back-EMF acquisition) state, but minimum operating speed is limited.

The minimum speed depends on many factors of the motor and hardware design, and differs for each application. This is because the back-EMF zero-crossing is disturbed and effected by the zero-crossing comparator threshold, as explained below and in **6.1.4.2 Effect of Mutual Inductance** and **6.1.4.1 Effect of Mutual Phase Capacitance**.

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

6.1.2 Maximum Application Speed

The maximum motor speed is limited by the minimum commutation period:

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

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

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

6.1.3 Voltage Close Loop

As shown in **6.2 Application Hardware and Software Configuration**, speed control is based on voltage close loop control. This should be sufficient for most applications.

6.1.4 Motor Suitability

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

6.1.4.1 Effect of Mutual Phase Capacitance

The effect of the mutual phase capacitances can play an important part in back-EMF sensing. Usually the mutual capacitance is very small. Its influence is significant only during PWM switching, when the system experiences very high *du/dt*. The effect of mutual capacitance is described in **3.1.4.4 Effect of Mutual Phase Capacitance**.

NOTE: *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, which are usually hand-wound.*

NOTE: *Failing to maintain balance of the mutual capacitance can easily disqualify such motors from using sensorless techniques based on the back-EMF sensing. Usually, BLDC motors with windings wound on*

separate poles show minor presence of mutual capacitance. Thus, the disturbance is insignificant.

6.1.4.2 Effect of Mutual Inductance

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

The effect of mutual inductance is described in **3.1.4.3 Effect of Mutual Inductance**.

NOTE: *A 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.*

NOTE: *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.*

6.2 Application Hardware and Software Configuration

6.2.1 Hardware Configuration

As mentioned before, the software can be configured to run with any suitable motor platform.

1. MMDS Evaluation Board (KITMMDSMR8)
   Using a real-time debugger (supplied with the Metrowerks compiler) the evaluation board is connected to the board via an emulator connector. This solution is recommended for software evaluation.
2. Programmed MCU with serial bootloader soldered on the

high-voltage BLDC drive board with MC68HC908MR8. This solution is recommended for final tests.

6.2.2 Hardware Setup

WARNING: ***Do not touch any part of the board when the power supply is applied to INPUT LINE connector and green power-on LED D1 is on. There is high risk of electric shock caused by high voltage, which may cause serious injury or death.***

1. Connect the motor to MOTOR terminal J3.
2. Connect the board to PC using serial cable (not null modem). The program is controlled only from the PC using the optoisolated serial interface. A manual interface has not been provided on the board, to minimize the risk of electric shock.
3. Connect the power supply cable with ON/OFF switch to INPUT LINE terminal J2.

CAUTION: *Do not mix the MOTOR and INPUT LINE connector. If the power supply voltage is applied to the motor connector, the power module will be damaged.*

6.2.3 Software Setup

6.2.3.1 Application Software Files

The application HC08 software files are:

- **...\CBMR8_SM40\CBMR8.mcp**, application project file
- **...\CBMR8_SM40\sources\const_cust.h**, definitions for software customizing for high-voltage (230/115 Vac) power board
- **...\CBMR8_SM40\sources\code_fun.c**, program C language functions
- **...\CBMR8_SM40\sources\code_fun.h**, program C language functions header
- **...\CBMR8_SM40\sources\const.h**, main program definitions

- **...\CBMR8_SM40\sources\mr8io.h**, MC68HC908MR8 registers definitions file
- **...\CBMR8_SM40\sources\mr8_bit.h**, MC68HC908MR8 register bits definitions file
- **...\CBMR8_SM40\sources\bldc08.c**, main program
- **...\CBMR8_SM40\sources\code_start.c**, motor alignment and starting (back-EMF acquisition) state functions
- **...\CBMR8_SM40\sources\code_start.h**, motor alignment and starting (back-EMF acquisition) state function header
- **...\CBMR8_SM40\sources\code_run.c**, motor running state function
- **...\CBMR8_SM40\sources\code_run.h**, motor running state function header
- **...\CBMR8_SM40\sources\code_isr.c**, program interrupt functions
- **...\CBMR8_SM40\sources\code_isr.h**, program interrupt functions header
- **...\CBMR8_SM40\sources\ram.c**, general RAM definitions
- **...\CBMR8_SM40\sources\ram.h**, general RAM declarations header
- **...\CBMR8_SM40\sources\ram_bit.h**, general RAM bits definitions header
- **...\CBMR8_SM40\sources\ram_cust_param.c**, RAM variables for software customizing definitions
- **...\CBMR8_SM40\sources\ram_cust_param.h**, RAM variables for software customizing header declarations
- **...\CBMR8_SM40\sources\tab_cust.c**, constants/tables definitions
- **...\CBMR8_SM40\sources\tab_cust.h**, constants/tables definitions header

- **...\CBMR8_SM40\sources\sci.c**,
  PC master software communication subroutines
- **...\CBMR8_SM40\sources\sci.h**,
  PC master software communication subroutines header
- **...\CBMR8_SM40\sources\code_asm.asm**,
  program assembler functions
- **...\CBMR8_SM40\sources\code_asm.h**,
  program assembler functions header
- **...\CBMR8_SM40\prms\default.prm**,
  linker command file

6.2.3.2 Application PC Master Software Control Files

The application PC master software control files are:

- **...\CBMR8_SM40\pcmaster\CDHC08.pmp**,
  PC master software project file
- **...\CBMR8_SM40\pcmaster\source**,
  directory with PC master software control page files

6.2.3.3 Application Build

To build the sensorless BLDC with back-EMF zero-crossing application, open the **CBMR8.mcp** project file and execute the *Make* command, as shown in **Figure 6-1**. This will build and link the application and all required Metrowerks libraries.

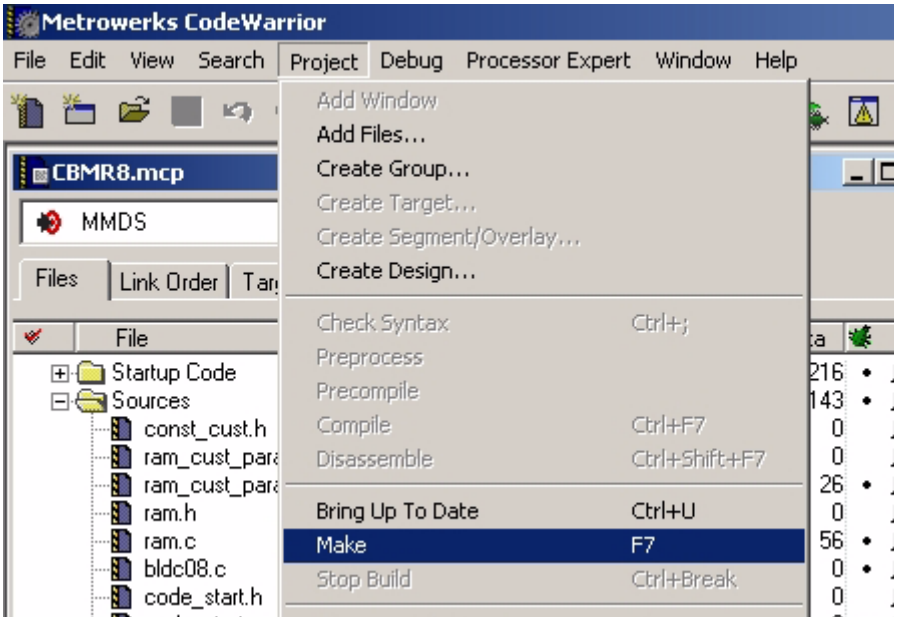


Figure 6-1. Execute Make Command

6.2.3.4 Programming the MCU

For easy changes of the software configuration as well as easy microcontroller reprogramming during the software customizing, the serial bootloader has been chosen. The bootloader must be programmed into the MCU before the MCU is soldered onto the board. You can use the ICS08MR board, or another device used for programming the HC08 microcontrollers, to get the bootloader into the MR's FLASH.

The HC08 programming with bootloader works in a way similar to monitor mode programming. The advantage of this solution is that no special hardware is required on the board. For a detailed description of the bootloader functionality, refer to the application note in **References 13**

If the SM40 motor is used, the software image located in the bin directory of the project can be used to program the device. For other motor types, the software must be customized. A detailed description of the software customization is in **6 User Guide**.

To program the board with a new software image, use the following steps.

1. Switch off the board supply and wait until the green power-on LED (D1) is off.
2. Make sure that any application (PC Master) does not occupy the serial port COM1. If other port is used for programming, change the settings in the “flash.bat” batch file.
3. Run the “flash.bat” batch file in the CBMR8_SM40 directory. This batch file calls the serial bootloader “hc08sprg.exe 1 bin/CBMR8.sx”
4. When you receive the bootloader message “Waiting the HC08 reset ACK”, switch on the power supply. This will cause the MCU power-on reset.
5. The bootloader will connect to the device and display the messages sent from the MCU.
6. If you confirm the part programming, the device will be programmed with the software image.
7. After the device programming has been finished the bootloader resets the MR8 and the execution of the new program will start.

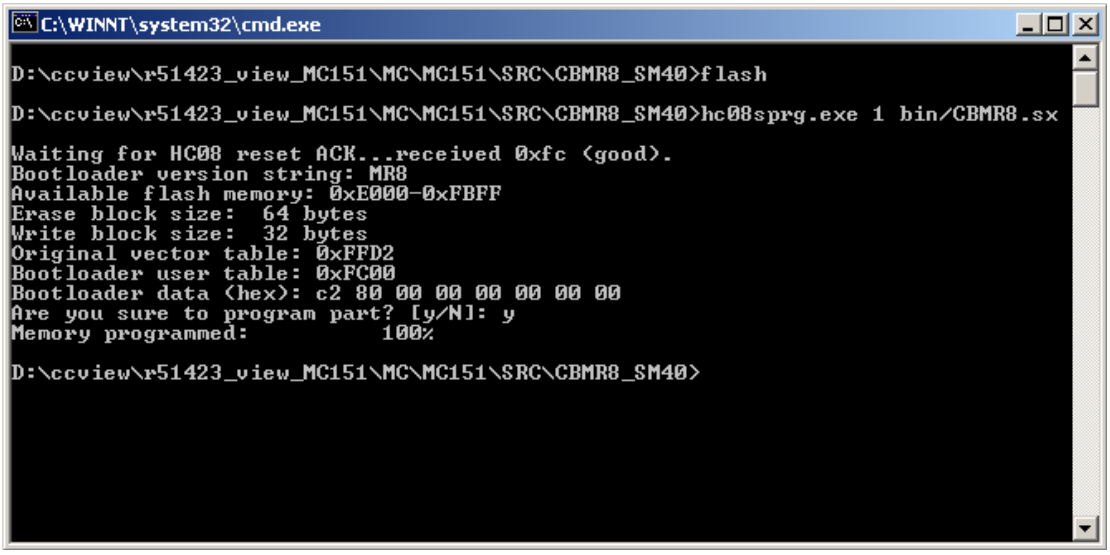


Figure 6-2. Bootloader Messages During MR8 Programming

6.2.4 Application Control

As mentioned before, for safety reasons the BLDC sensorless motor control application can operate only remotely from PC Master control.

6.2.4.1 Controlling the motor using PC master software

When the application is programmed into the MCU and the program is running correctly, run the PC master project file CDHC08.pmp located in pcmaster directory. (The PC Master application must be installed before it can be used.) The latest version PC Master software can be downloaded from the Motorola web pages. The application description page **Figure 6-3** should be displayed after PC Master start-up. It contains a brief description of the application functionality and an overview of the MC68HC908MR8 features.

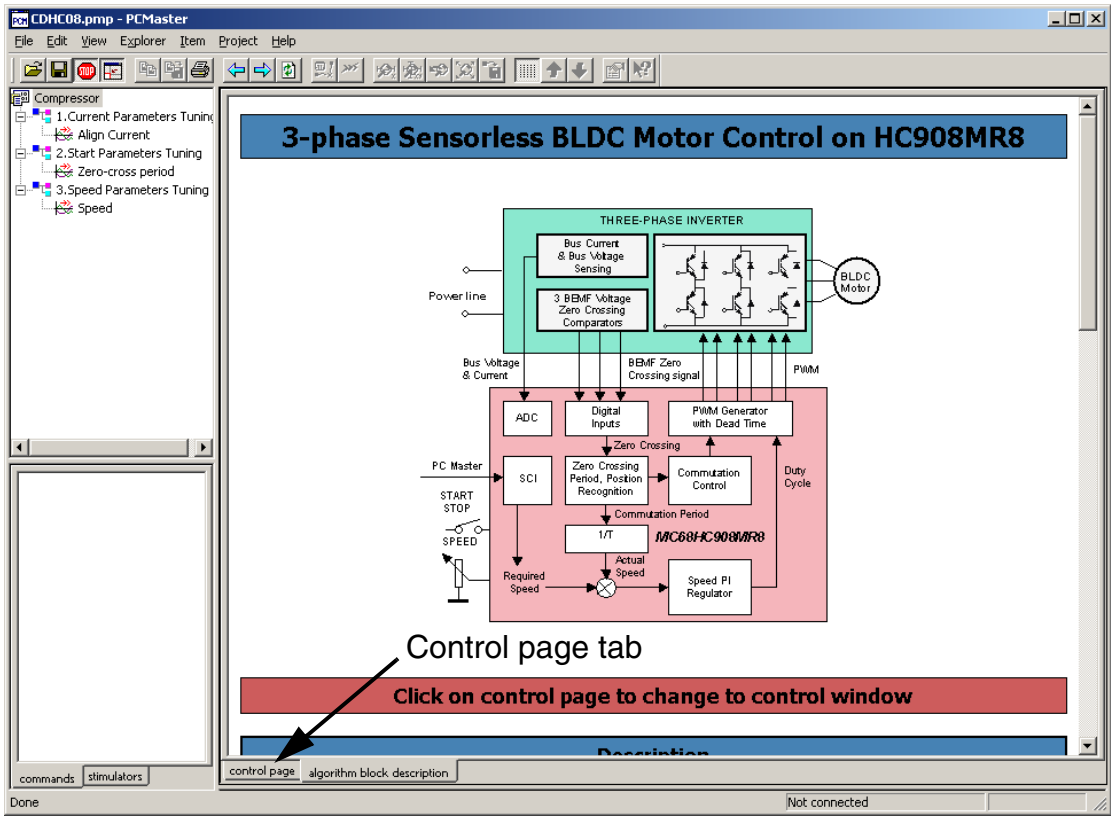


Figure 6-3. Application description page screenshot

The control page displayed in **Figure 6-4** should be seen when the application is running and communication with board has been successfully established.

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 file (..\bin\CBMR8.map) 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.*

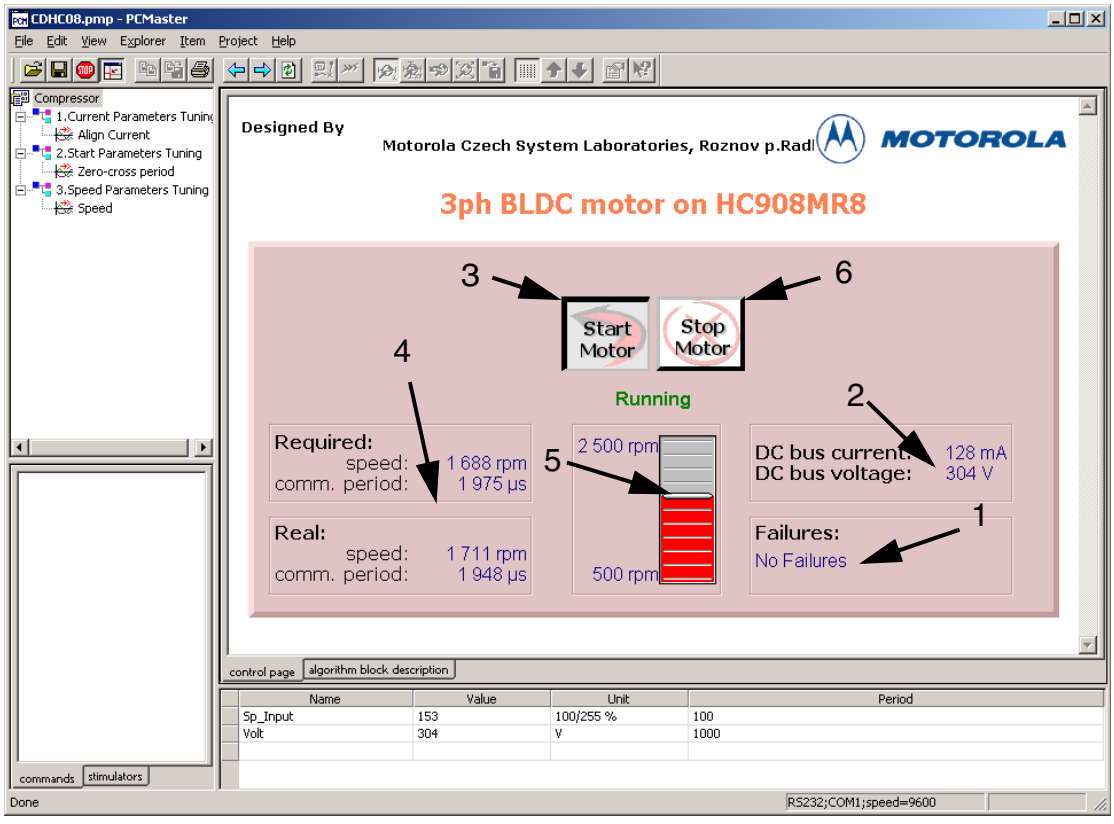


Figure 6-4. PC Master control page screenshot

If there are no failures (1), the application enters the stop state. The information about the DC-bus voltage and current are displayed in (2).

To run the motor, click on the “Start Motor” button (3). The motor will start running at minimum speed; about 500 rpm. You can see the information about the actual speed at (4) and (5).

The speed of the motor can be controlled by clicking on the “Speed Slider” (5) from its minimum speed up to maximum speed. The indication of the running state is provided by the ON state of the red user LED (D2).

In a case of fault the motor will stop. The information about the fault type is displayed in (1) and the red user LED (D2) will be flashing at 1s period.

To stop the motor, click on the “Stop Motor” button (6)

To run the sensorless BLDC application, the following software is needed:

- Metrowerks compiler for HC08 — installed on your PC
- Sensorless BLDC application HC08 software files (located in the CBMR8_SM40 directory)

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

- PC master software for PC — installed on your PC
- Sensorless BLDC application PC master software control files (located in CBMR8_SM40\pcmaster directory)

The HC08 and PC master software control files for the sensorless BLDC application are delivered together in the **CBMR8_SM40** directory. It consists of the files listed in **6.2.3 Software Setup**.

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 the fault before starting again.*

6.3 Parameter Setting and Tuning for Customer Motor

This section describes how to modify the software parameters for any BLDC motor and provides some hardware adaptations. The software parameters can be evaluated from a PC 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 6-5**.
- 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 6.1 Suitability Guide for Customer Application and Motor.
- The **6.3.3 Parameter File Selection** must be made according to **6.2.1 Hardware Configuration** used.
- If a modified hardware power stage is used, the appropriate constants in const_cust.h file must be set as described in **6.3.3.1 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 done as explained in:
  - 6.3.4 Software Customizing to Motor: Voltage and Current Settings
  - **6.3.4.3 Alignment Current and Current Regulator Setting**
  - 6.3.5 Software Customizing to Motor: Commutation and Start-up Control Setting
  - 6.3.6 Software Customizing to Motor: Speed Control Setting

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

to the current regulator sampling period as explained in 6.3.7 PWM Frequency and Current Sampling Period Setting.

If there is still a problem running the motor, check that the motor is suitable for sensorless control with back-EMF zero-crossing (see **Figure 6-7**) as described in **6.1.4 Motor Suitability**. Then, again check the application suitability.

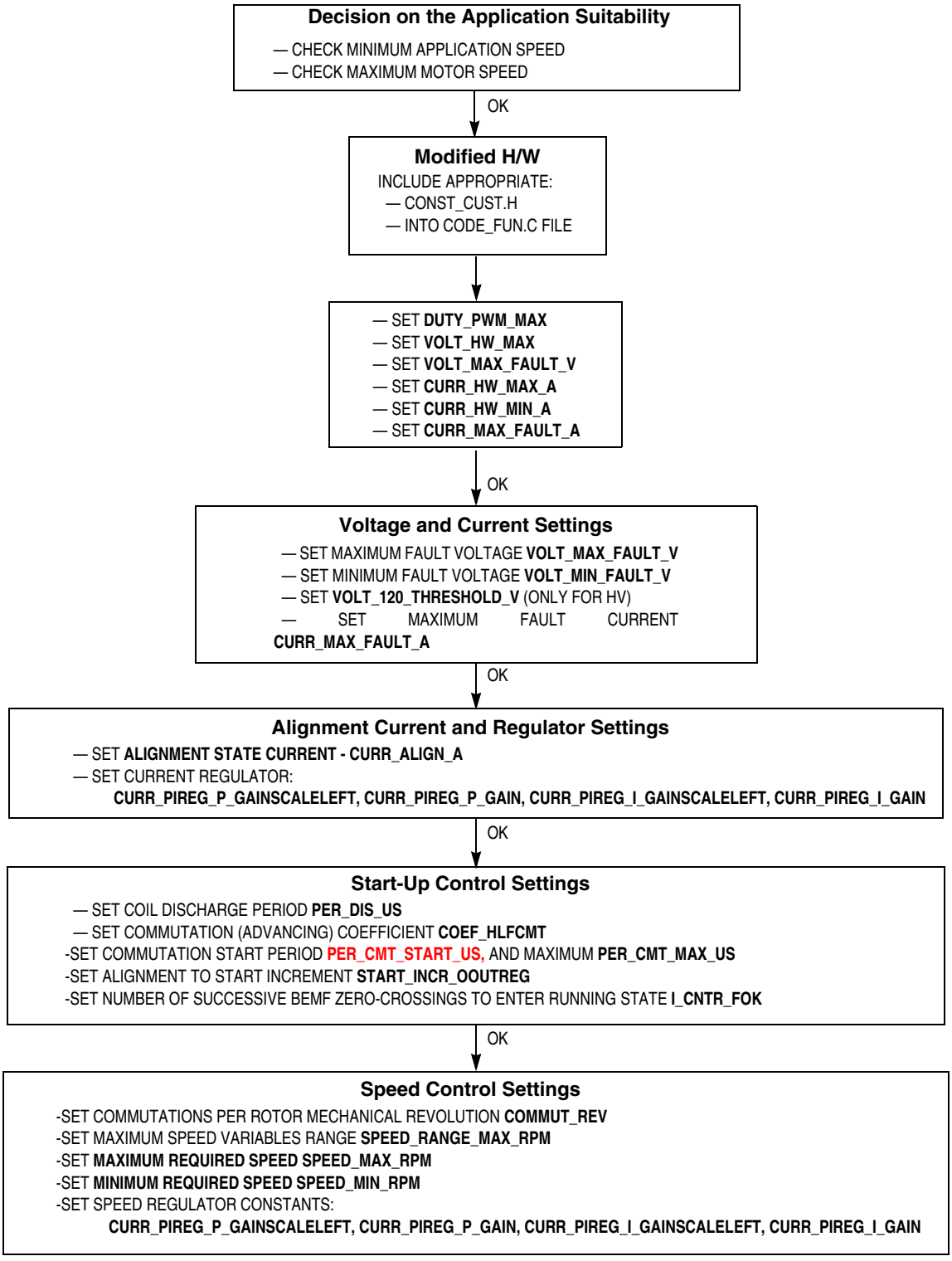


Figure 6-5. Follow-up for Software Customizing to Customer Motor

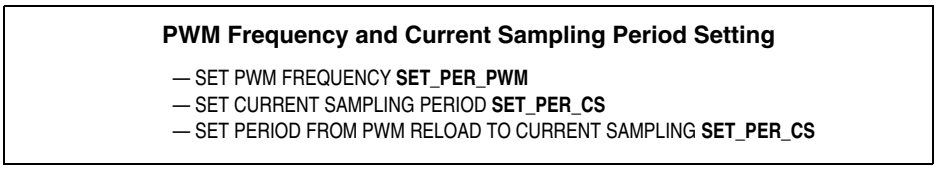


Figure 6-6. Follow-up for Advanced Software Customizing

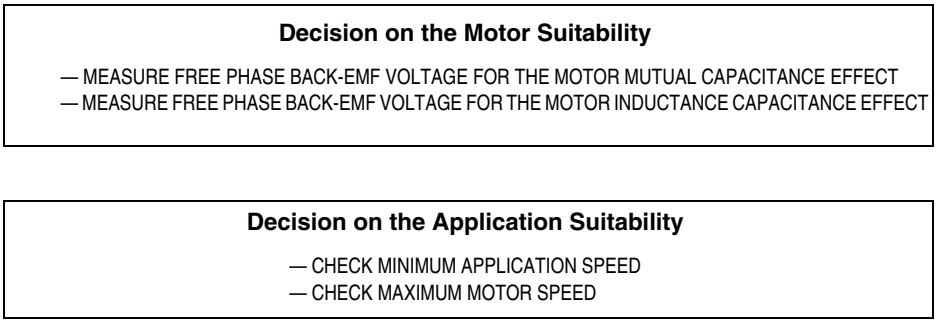


Figure 6-7. Follow-up for Software Customizing Trouble Shouting

6.3.1 Software Parameter Tuning with PC Master Software Project File

Sensorless BLDC software is provided with a PC master software project file for on-line software parameter tuning (see **Figure 6-8**). This file supports:

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

The remote application control uses the same control page as described in **6.2.4.1 Controlling the motor using PC master software**. 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, you must have PC master software installed on your PC.*

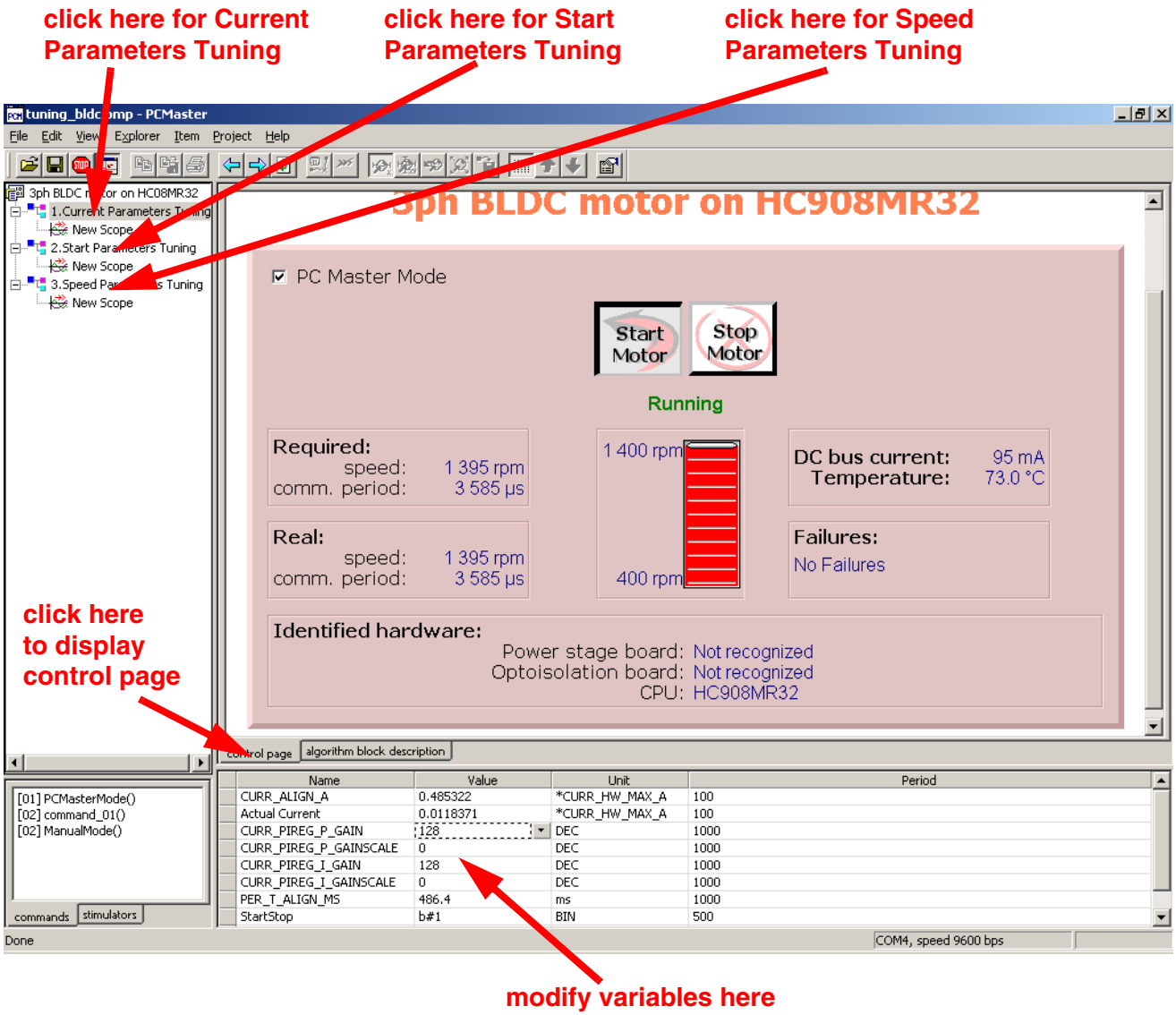


Figure 6-8. PC Master Software Parameters Tuning Control Window

Start the PC master software parameter tuning application:

...\CBMR8_SM40\pcmaster\CDHC08.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 6-8**). Then you can press “control page” to make the control window visible (and provide control in the same way as in **6.2.4.1 Controlling the motor using PC master software**). Or, you can display the oscilloscope window (see **Figure 6-9**). You can then modify the variable values in the variable window (**Figure 6-9**), which is visible for both control page or oscilloscope page turned on. The variables can be modified according to their defined limits.

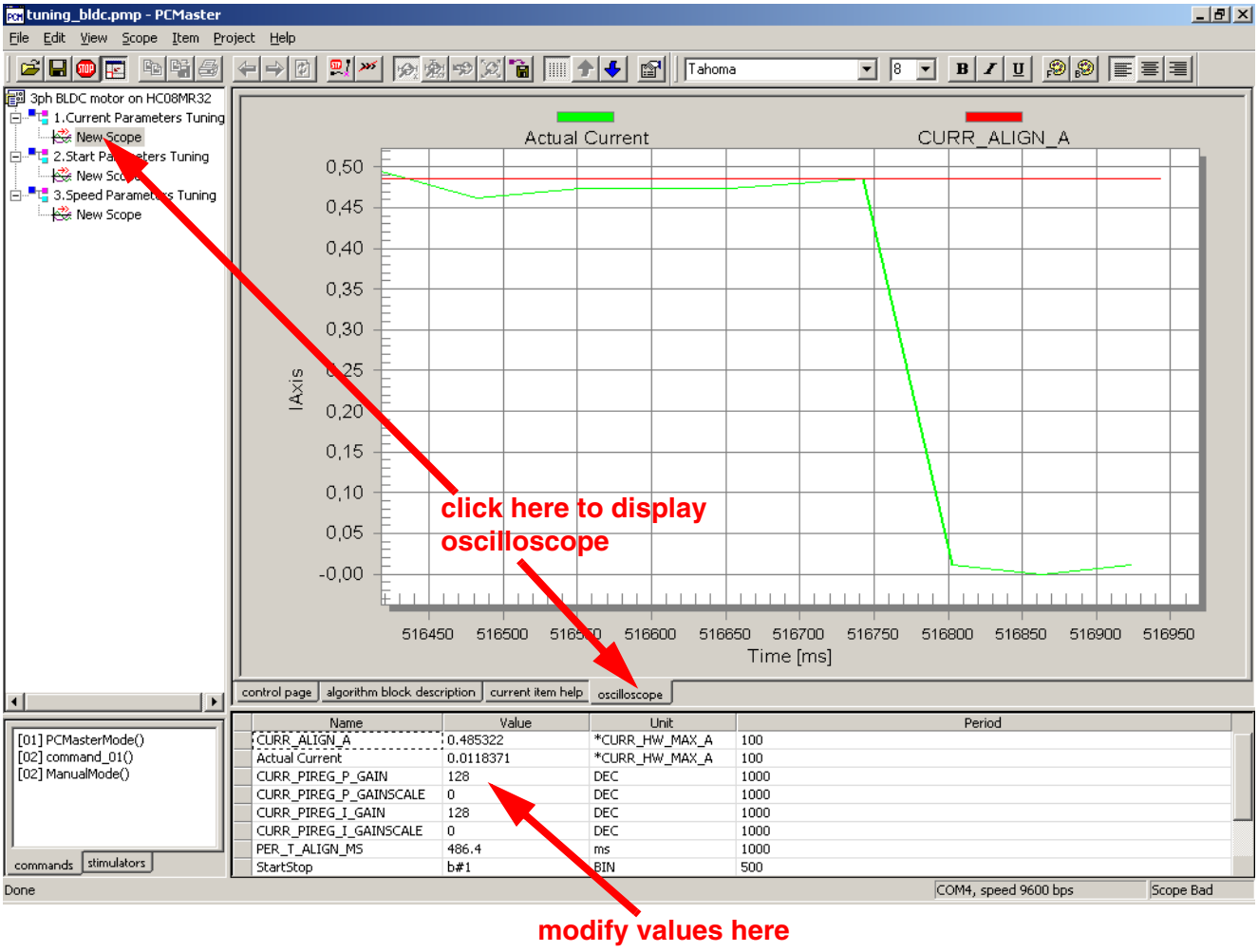


Figure 6-9. PC Master Software Parameters Tuning Control Window

NOTE: *The software parameters can be temporarily modified and evaluated using the PC master software tuning file. However, the parameter settings are not stored in non-volatile memory (after reset the software loads parameters from const_cust.h file). Otherwise, you will get default setting after the MCU reset.*

6.3.2 Software Parameter Setting Follow-up

The software is provided for the BLDC motor SM40, but can be configured for other 3-phase trapezoidal BLDC motors (or other board configurations).

Usually, the motor control drive must be set or tuned for:

- Current or voltage parameters
- Dynamic parameters

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

6.3.2.1 Labels in the File const_cust.h

Most of the software parameter settings are provided in const_cust.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

6.3.2.2 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

Follow the next sections, or the labels in the source code, to customize the software.

6.3.3 Parameter File Selection

As explained in **6.2.3 Software Setup**, one of the following files is used for most of the software parameter configuration:

...\CBMR8_SM40\sources\const_cust.h, definitions for software customizing, for high-voltage (230/115 Vac) power board

...\CBMR8_SM40\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_cust.h** in the following sections.*

6.3.3.1 Software Customizing to Power Stage

The hardware board parameter customizing is provided in the **const_cust.h** file.

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

An example of software customizing to power stage is shown in **6.3.3.5 Example of Software Customizing to Hardware**.

6.3.3.2 Maximum PWM Duty Cycle

Maximum PWM duty cycle [-]:

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

Range: < 0,1 >

The proportional value of the maximum PWM duty cycle is determined by the power stage boards used.

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

6.3.3.3 Voltage Setting Hardware Customizing

Maximum 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 the voltage sensing range is different from the 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 maximum voltage when the drive fault state should be entered. So the constant **VOLT_MAX_FAULT_V** must be set according to the maximum voltage limit of the power stage or the motor, using the lower value. Therefore, setting this constant is also mentioned in **6.3.4.1 Maximum and Minimum Voltage Limits Setting** under a different label (**CAN_CHANGE_1**).

6.3.3.4 Current Setting Hardware Customizing

The maximum measurable current determined by hardware current sensing [A]:

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

Range: <0,infinity)

The minimum 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.

Maximum 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 maximum 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 **6.3.4.2 Maximum and Minimum Current Limits Setting**)

6.3.3.5 Example of Software Customizing to Hardware

For BLDC high-voltage drive system, the following parameters in **const_cust.h** must be modified

1. Maximum PWM duty cycle remains the same:

```
#define DUTY_PWM_MAX 0.99
```

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

```
#define VOLT_HW_MAX 401.0
```

3. Maximum limit of DC-bus voltage should be set according to motor or application requirements, but VOLT_MAX_FAULT_V > 380V

```
#define VOLT_MAX_FAULT_V 380.0
```

4. Board has maximum measurable current unchanged

```
#define CURR_HW_MAX_A (+4.10)
#define CURR_HW_MIN_A (-4.10)
```

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

```
#define CURR_MAX_FAULT_A 1.50
```

When the software parameters are set for the hardware, you should follow the settings in **6.3.4 Software Customizing to Motor: Voltage and Current Settings**.

6.3.4 Software Customizing to Motor: Voltage and Current Settings

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

NOTE: *First, voltage and current settings must be done. For settings which must be done, follow the labels* ***MUST_CHANGE_nn*** *and* ***MUST_CHANGE_EXPER_nn*** *in file* ***const_cust.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.h***

Detailed description starts here.

6.3.4.1 Maximum and Minimum Voltage Limits Setting

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

Maximum 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 maximum voltage when the drive fault state should be entered. So, the constants **VOLT_MAX_FAULT_V** must be set according to maximum voltage limit of the motor or the power stage, using the lower value. Therefore, the setting of these constants is also mentioned in **6.3.3.3 Voltage Setting Hardware Customizing** under a different label **MUST_IF_HW_CHANGE_3**. It should be changed when there are problems with over-voltage.

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

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

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

Range: <0,VOLT_RANGE_MAX>

The 120 V voltage threshold setting should be used for high-voltage hardware only. 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.

6.3.4.2 Maximum and Minimum Current Limits Setting

Most of the current limit settings do not have 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 + minimum allowed offset error (EQ 6-3.)

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

Maximum 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 maximum 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.

6.3.4.3 Alignment Current and Current Regulator Setting

The current during the 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 than nominal motor current.

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

It may 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:

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

```
/* 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 have to be changed (if change is required see **6.3.7 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.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 **6.2.3 Software Setup**, **6.2.3.3 Application Build**, and **6.2.4 Application Control**).
4. Start the PC master software tuning project.
5. Select Current Parameters Tuning subproject (see **6.3.1 Software Parameter Tuning with PC Master Software Project File**) 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 a real oscilloscope.

7. Set PC master software control mode, and start the motor (see **6.2.4.1 Controlling the motor using PC master software**).

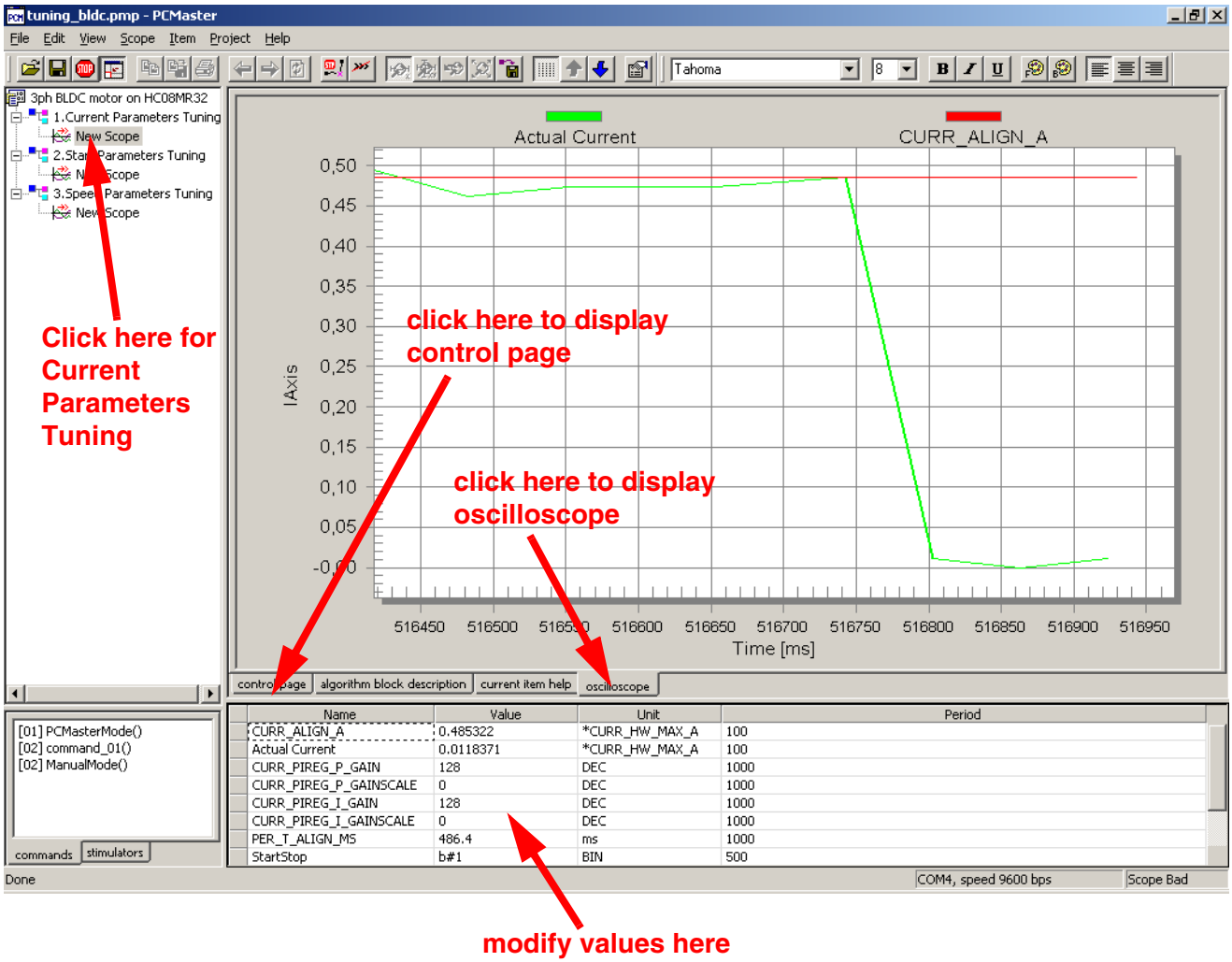


Figure 6-10. PC Master Software Current Parameters Tuning Window

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 evaluate further the setting of the regulator parameters to get a smoother current waveform, or until the regulation seems to be performing well.
13. Open **const_cust.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 **6.2.3.3 Application Build**).
16. You can also tune regulator dynamic characteristics of current transients (steps **17** to **26**) or finish the regulator tuning.
17. Run the code (see **6.2.3 Software Setup** and **6.2.4 Application Control**).
18. Start the PC master software tuning project.
19. Select Current Parameters Tuning subproject (see 6.3.1 Software Parameter Tuning with PC Master Software Project File) 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 a real oscilloscope.
21. Set PC master software control mode and start motor (see **6.2.4.1**

Controlling the motor using PC master software).

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.h** and modify the regulator parameters with the final variable values evaluated with PC master software.
26. Build the code (see **6.2.3.3 Application 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 the Alignment state.

6.3.5 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 must be set.

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

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

Detailed description starts here.

6.3.5.1 Commutation Parameters

Commutation time period to discharge coil current [µs]

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

Range: <0,minimum commutation period*COEF_TOFF>

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

Must be:

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

where: **COEF_TOFF** is commutation Toff period coefficient from const.h file explained in **3.2 Control Technique Used**.

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°.

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

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 **3.2.1.3 Running: Commutation Time Calculation**.

6.3.5.2 Start-up Constants and Maximum Commutation Period

Constants defining start-up must 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, decrease or increase the **PER_CMT_START_US** value. Table 6-1 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 must be set with a specific motor.*

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

Table 6-1. 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

Maximum 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

6.3.4.3 Alignment Current and Current Regulator Setting in **const_cust.h**

2. Ensure that **PER_DIS_US** and **COEF_HLFCMT** are properly set in **const_cust.h**
3. Set #define **PER_CMT_START_US** in **const_cust.h** according to **Table 6-1**.
4. Ensure **PER_CMT_START_US**<=**PER_CMT_MAX_US**/2
5. Set #define **START_INCR_OOUTREG 20.0** in **const_cust.h**
6. To disable the 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.h** file
9. Build and run the code (see **6.2.3 Software Setup**, **6.2.3.3 Application Build**, and **6.2.4 Application Control**)
10. Start the PC master software tuning project
11. Select Start Parameters Tuning subproject (see 6.3.1 Software Parameter Tuning with PC Master Software Project File) 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

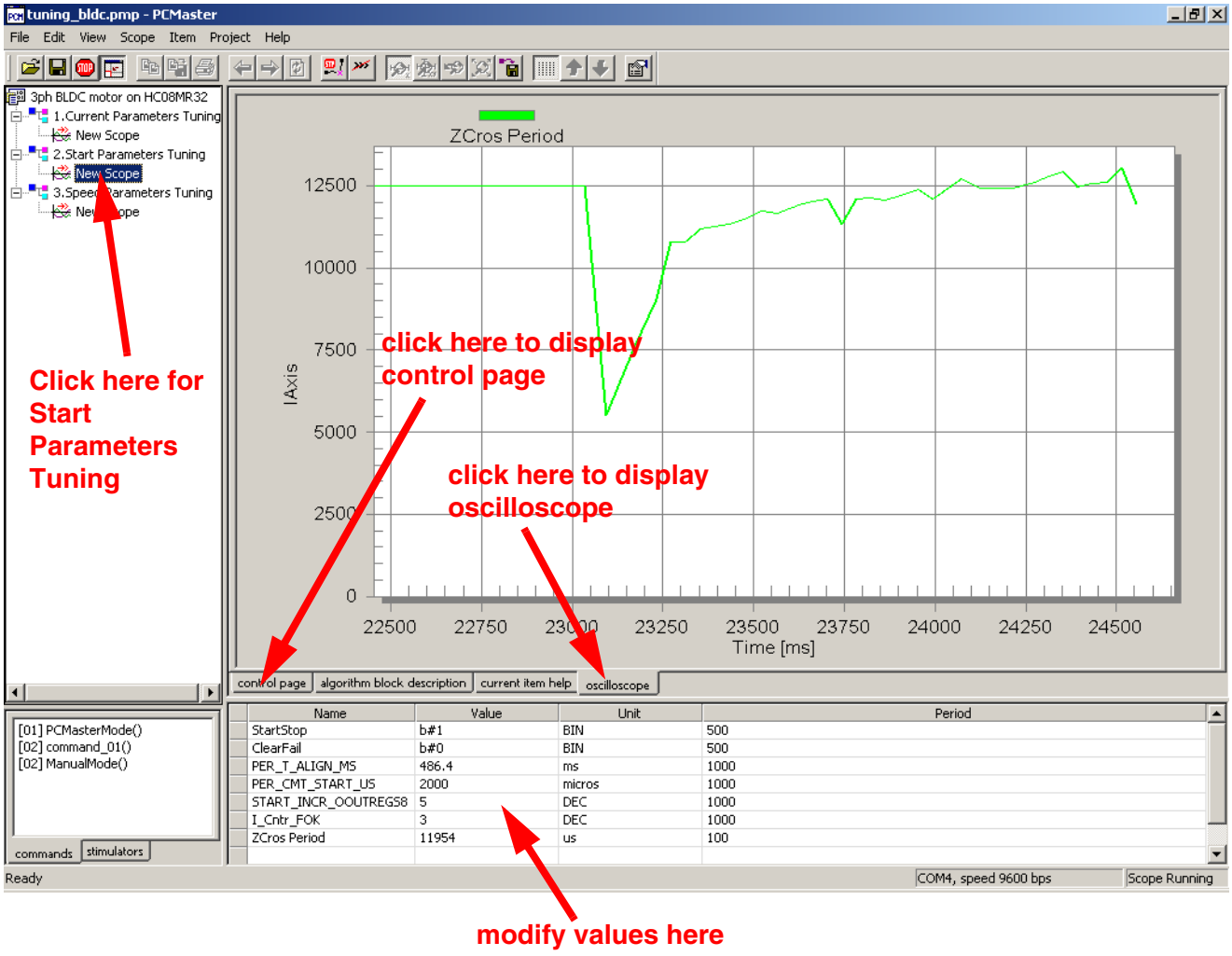


Figure 6-11. PC Master Software Start Parameters Tuning Window

13. Set PC master software control mode (see **6.2.4.1 Controlling the motor using PC master software**)
14. Start motor (see **6.2.4.1 Controlling the motor using PC master software**)
15. If the software signals 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 this is 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.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.h** file to enable speed regulation
22. Build the code (see **6.2.3.3 Application Build**)

6.3.6 Software Customizing to Motor: Speed Control Setting

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

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

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

Maximum 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 maximum actual motor speed (even during speed transient).

Maximum 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 maximum 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:

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

Minimum 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 minimum desired speed. It must be changed for any motor accommodation. The minimum 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 minimum speed SPEED_MIN_RPM must be evaluated experimentally.

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

Minimum PWM Duty cycle limit [-]:

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

Range: <0,1>

DUTY_PWM_MIN determines minimum PWM duty cycle limit, and in this way it restricts minimum 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 (6.3.5.2 Start-up Constants and Maximum Commutation Period in **const_cust.h**
2. Set **const_cust.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.h**
4. Set **SPEED_MIN_RPM** as required (should be **SPEED_MIN_RPM** > **SPEED_MAX_RPM/5** for reliable commutation at low speed)

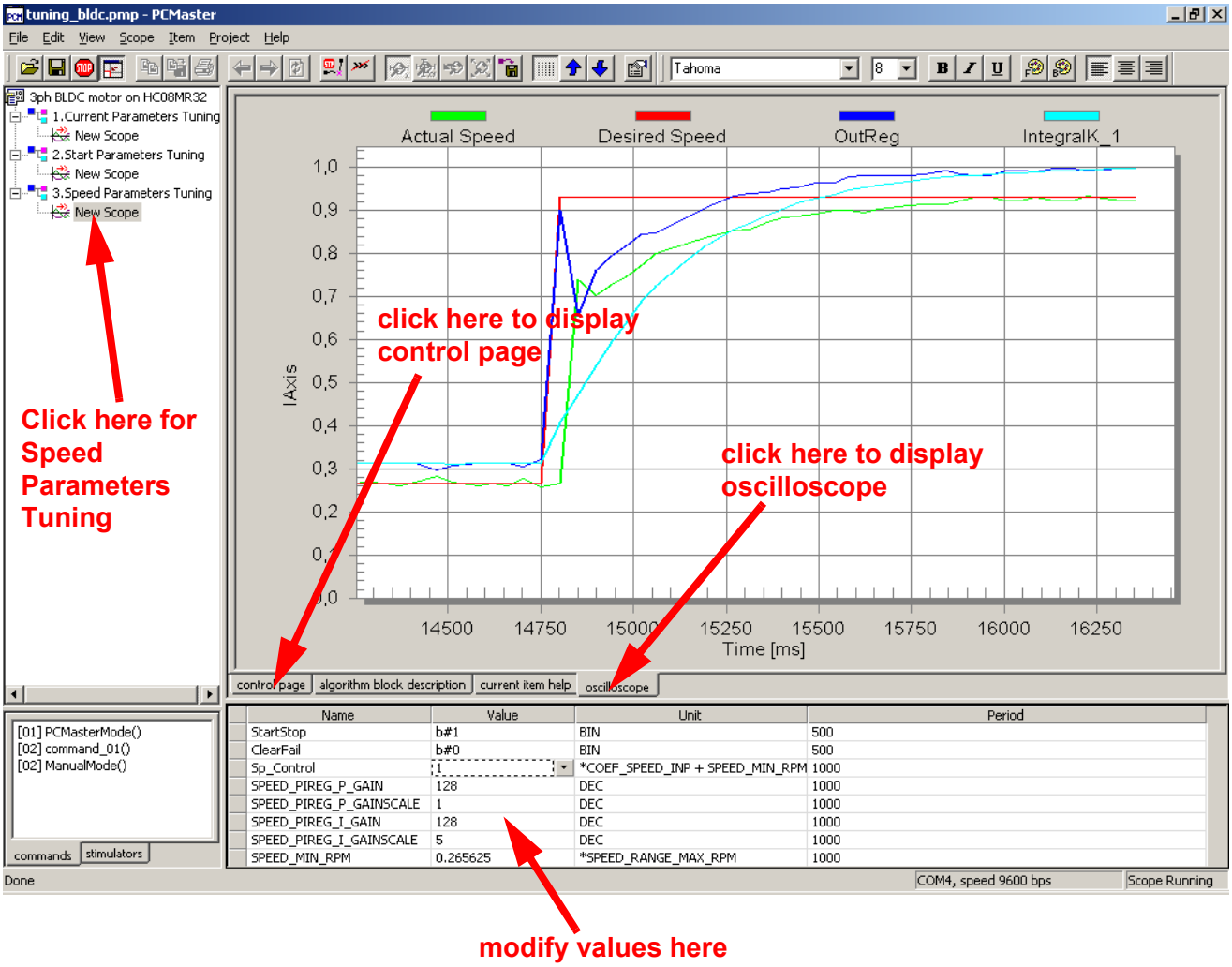


Figure 6-12. PC Master Software Speed Parameters Tuning Window

5. Build and run the code (see **6.2.3 Software Setup**, **6.2.3.3 Application Build**, and **6.2.4 Application Control**)
6. Start the PC master software tuning project
7. Select Speed Parameters Tuning subproject (see 6.3.1 Software Parameter Tuning with PC Master Software Project File) to be able to modify the current regulator
8. You can see the actual speed (and the 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 **6.2.4.1 Controlling the motor using PC master software**)
10. Set the speed to the middle of minimum and maximum 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 evaluate further the setting of the regulator parameters to get a smoother current waveform until the regulation seems to be performing well
16. Open **const_cust.h** and modify the regulator parameters with the final variable values evaluated with PC master software
17. Start motor (see **6.2.4.1 Controlling the motor using PC master software**)
18. Set minimum desired speed
19. If the displayed real speed is not able to go down to the desired minimum speed, it is necessary to decrease minimum PWM duty cycle DUTY_PWM_MIN in **const_cust.h**.
20. Then, you can tune dynamic characteristics of speed regulators (steps **22** to **29**) or finish tuning the regulators
21. Start motor (see **6.2.4.1 Controlling the motor using PC master software**)
22. Set minimum speed

23. Set maximum speed and observe the speed transient
24. Set minimum 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.h** and modify the regulator parameters with the final variable values evaluated with PC master software
29. Build the code (see **6.2.3.3 Application Build**)

Most important software settings are described in previous sections, but for some applications, PWM frequency must be modified. It is described in **6.3.7 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.h**. Then, it is possible to program the FLASH memory of the MC68HC908MR32 device.

6.3.7 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.h**.

6.3.7.1 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 6-2**.

Table 6-2. 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.*

6.3.7.2 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]

6.3.7.3 Current Sampling Instant

Time period from a PWM reload event (middle of central aligned PWM) to current sampling (time shift of ATD 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:

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

6.3.8 Conclusion: Software Parameter Setting and Tuning

If all the points in **6.3 Parameter Setting and 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 **6.1 Suitability Guide for Customer Application and Motor**, 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. Schematics and Parts List

A.1 Schematics

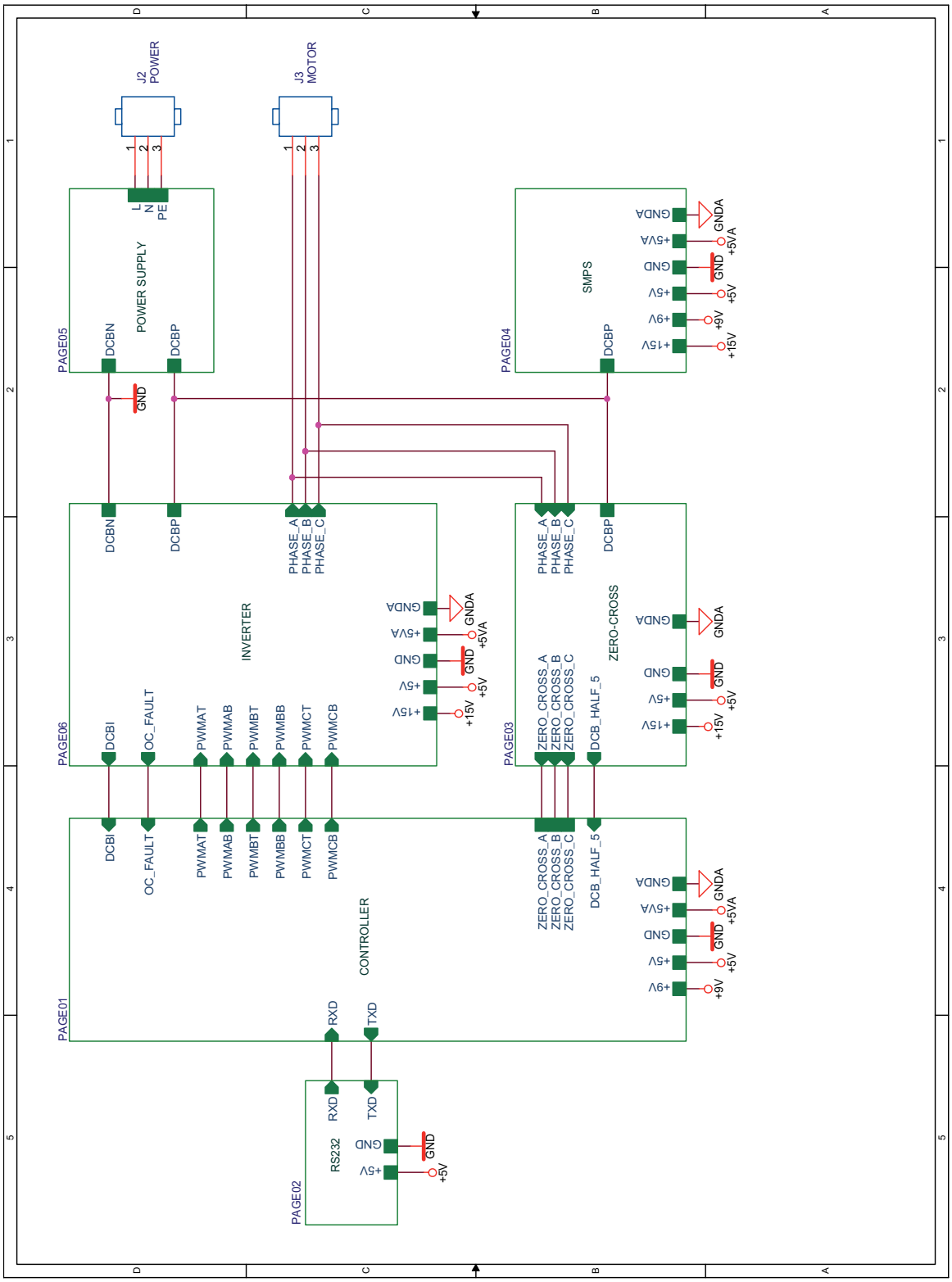


Figure A-1. Drive Overview

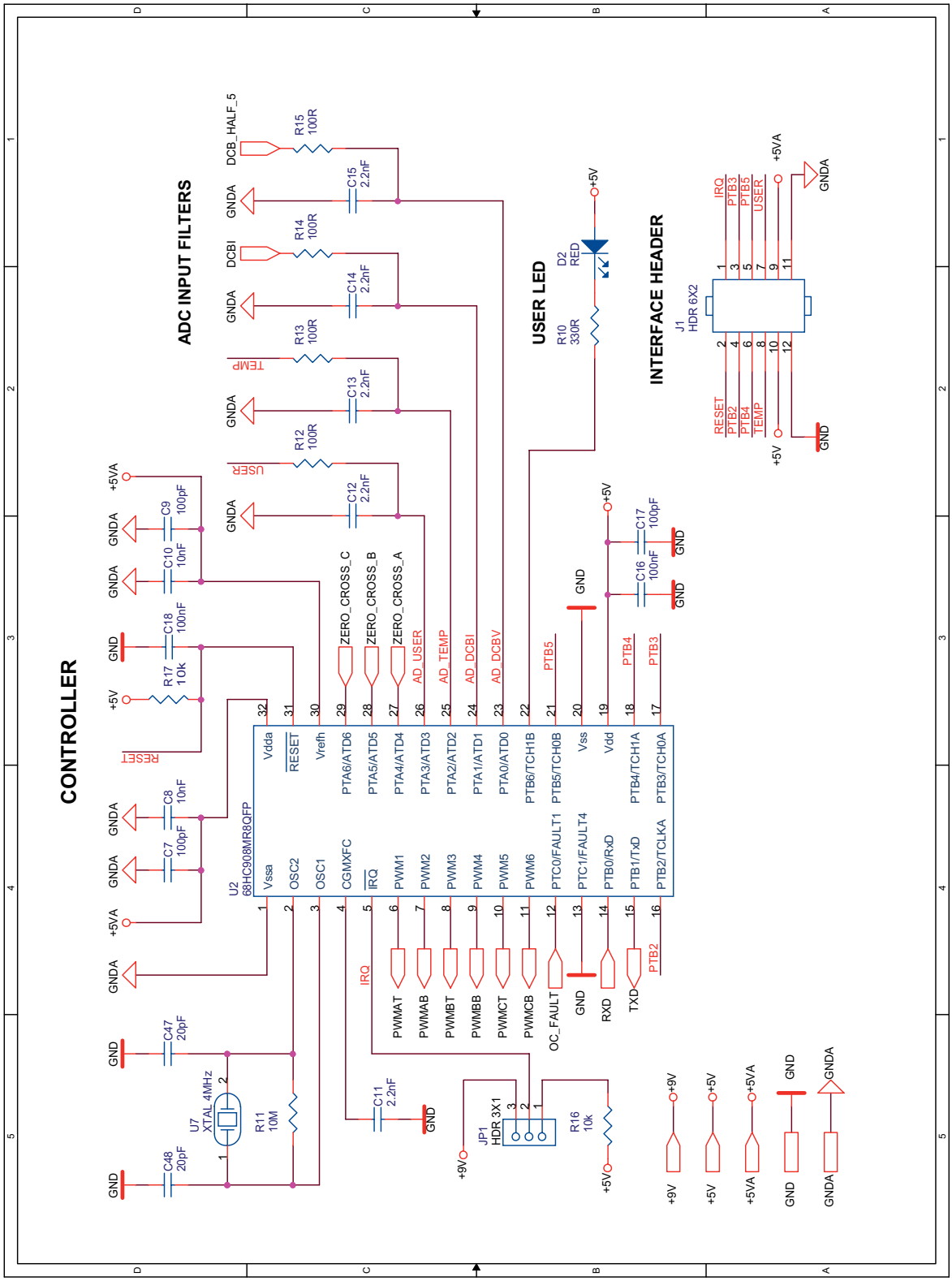


Figure A-2. Controller with MC68HC908MR8

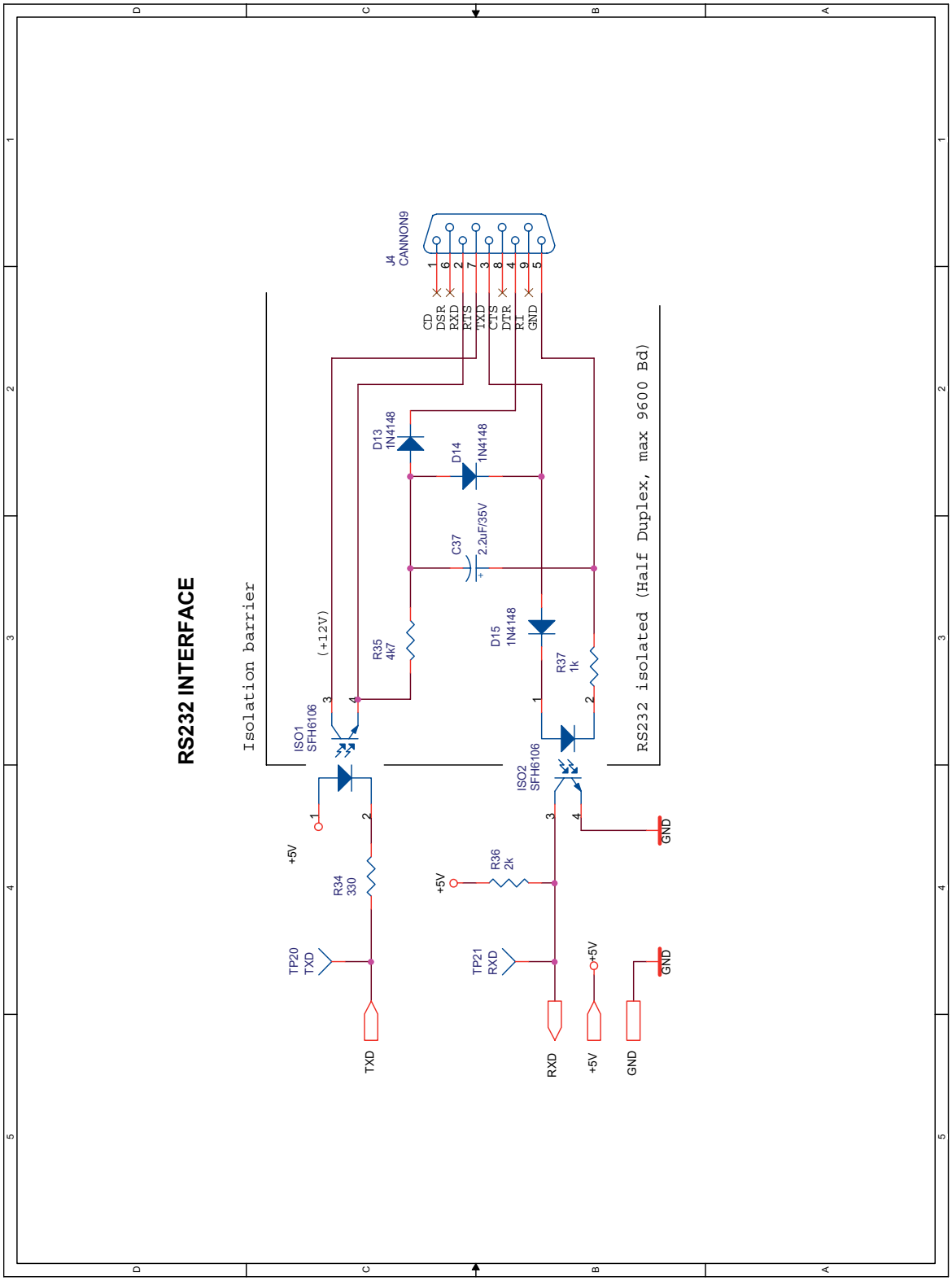


Figure A-3. Serial Communication Interface

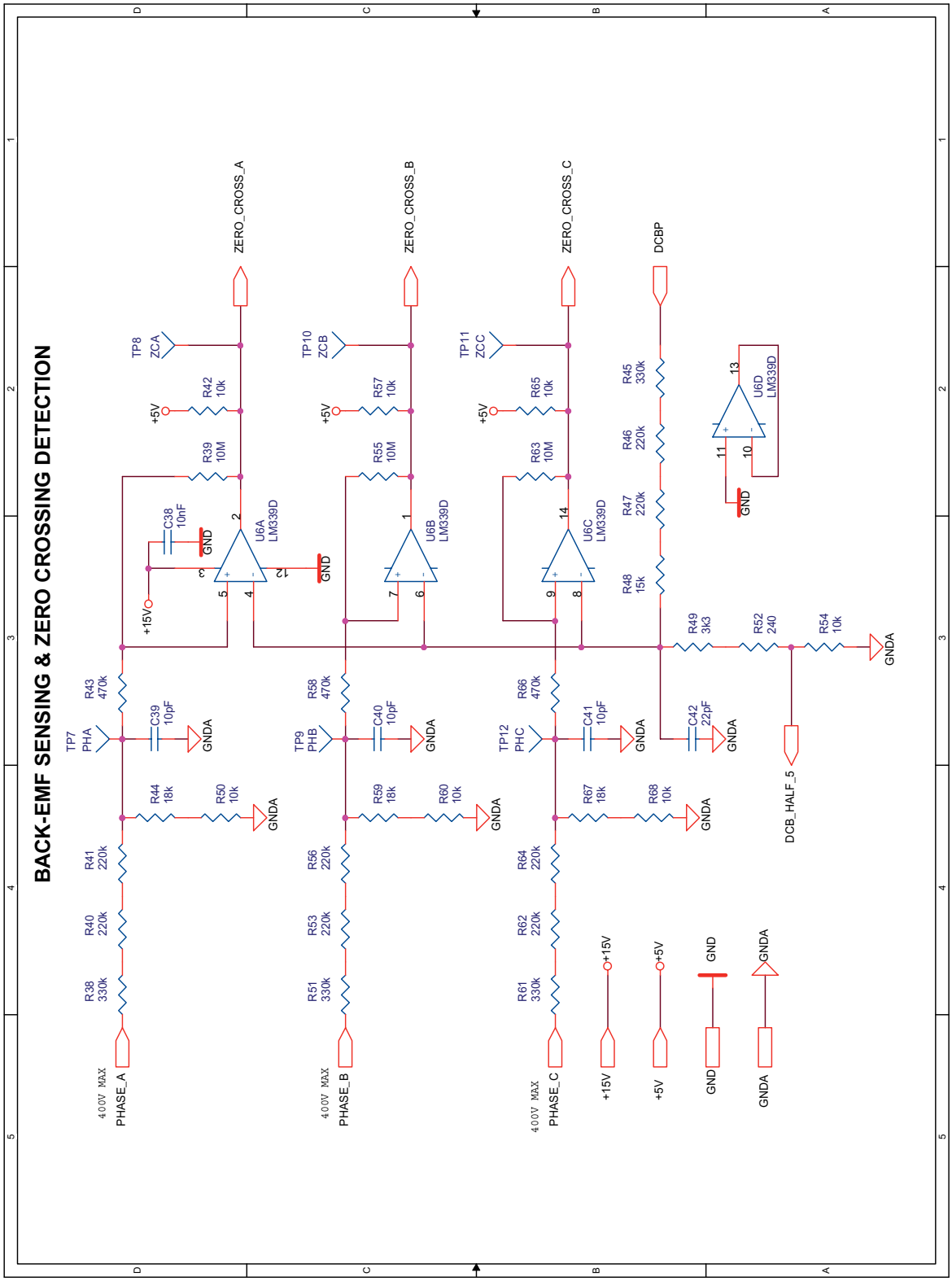


Figure A-4. Back-EMF Zero-crossing Detection

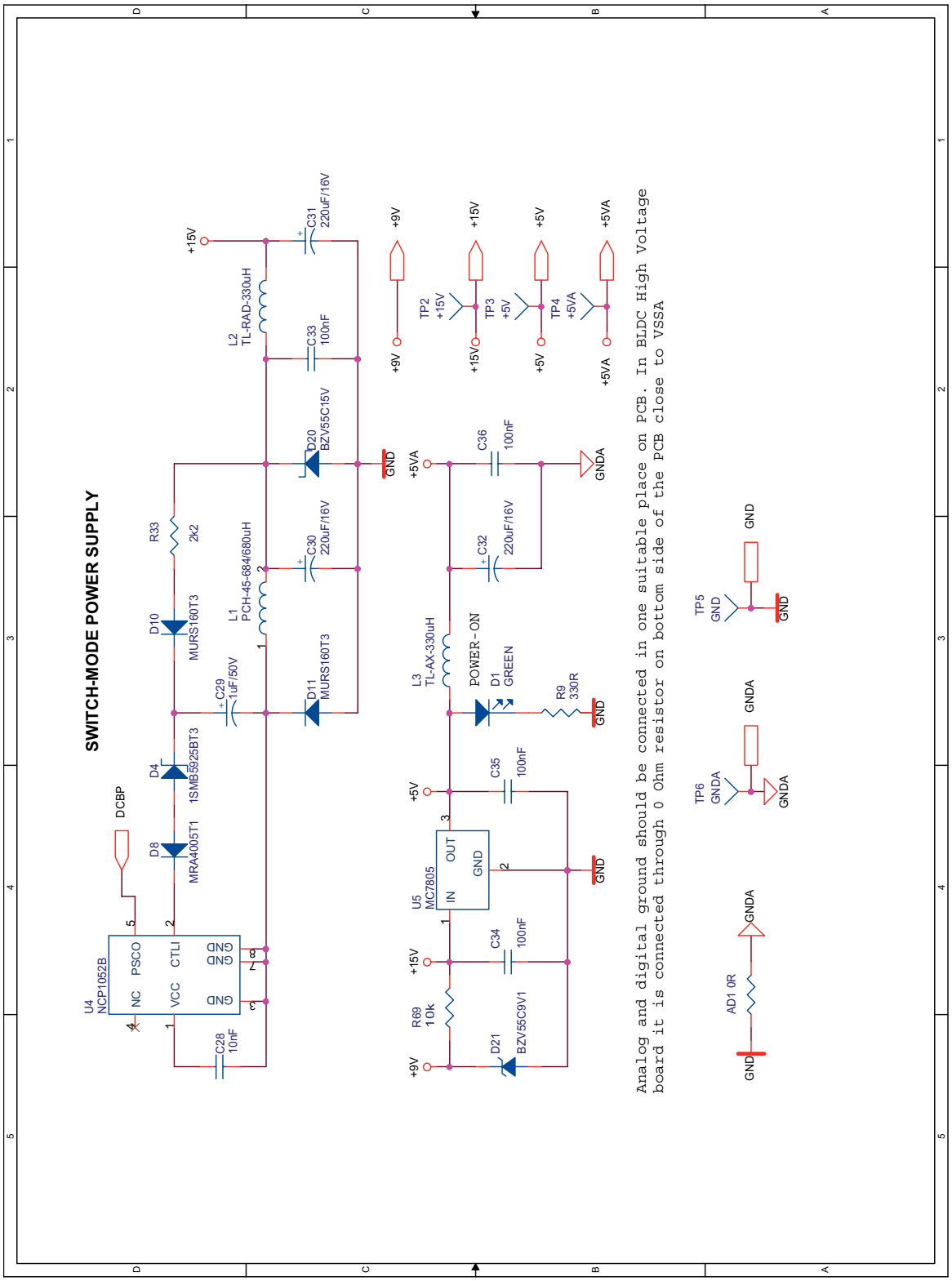


Figure A-5. Switch Mode Power Supply

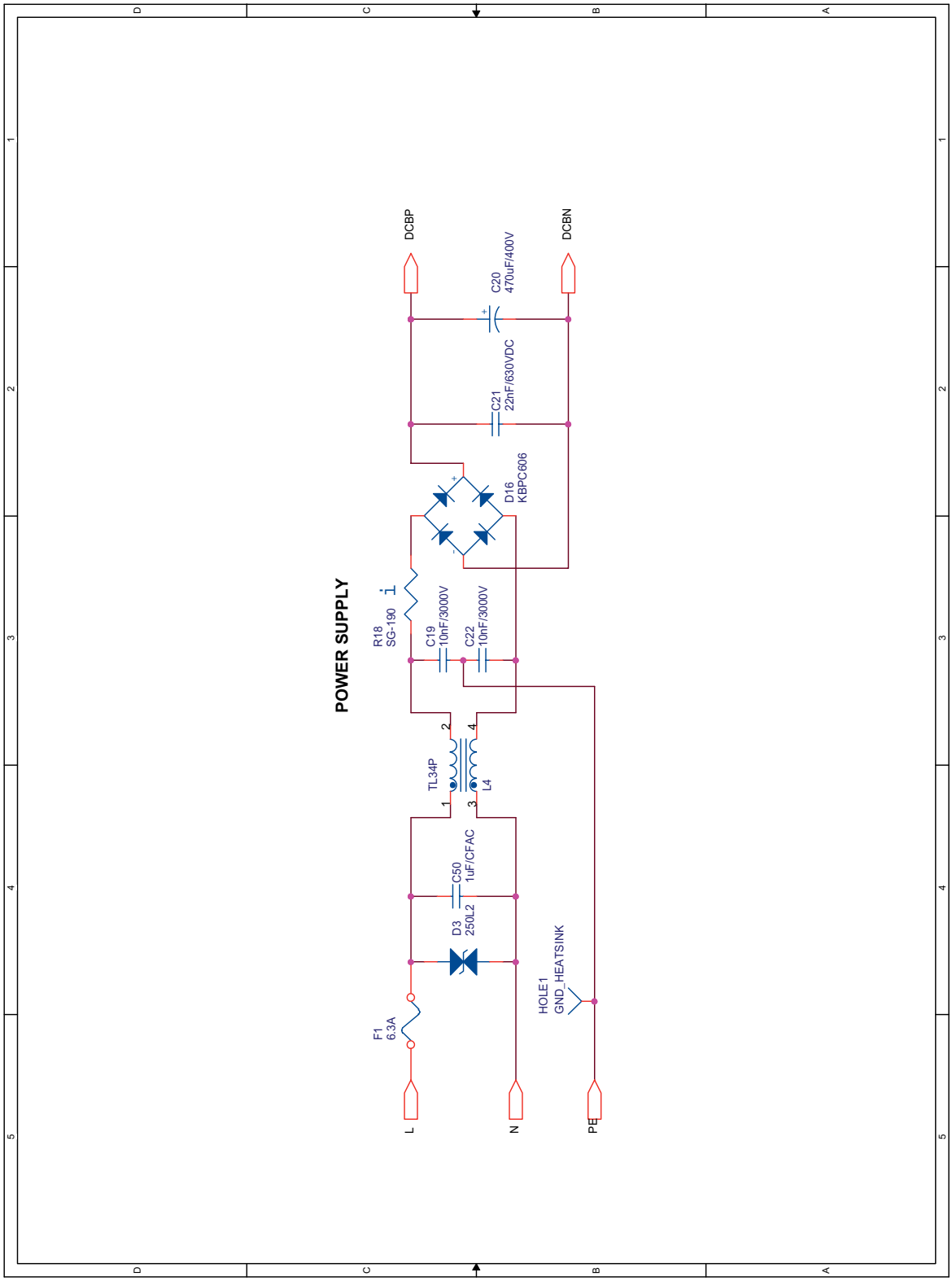


Figure A-6. DC-bus Link Power Supply

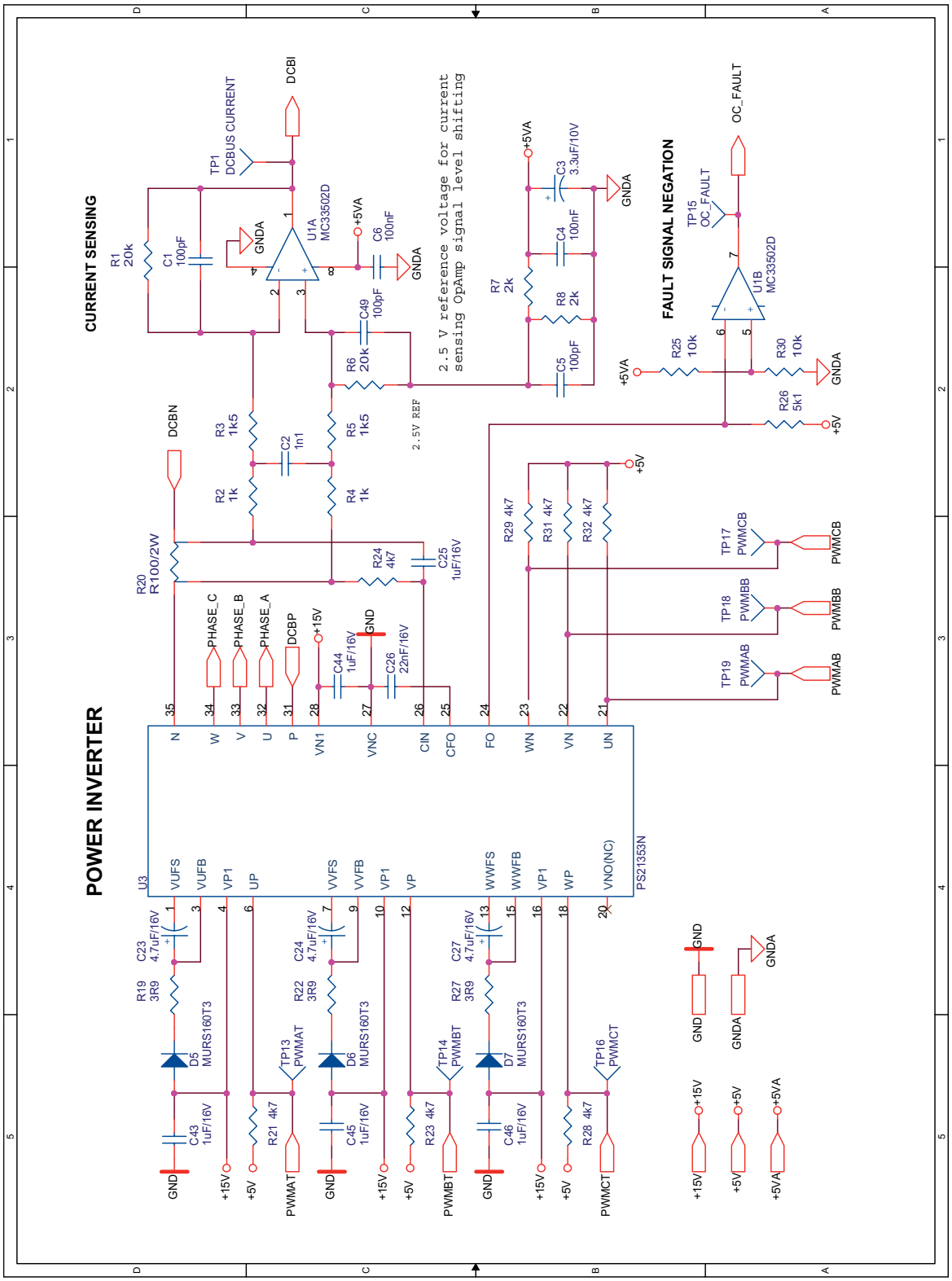


Figure A-7. Power Inverter with PS21353-N

A.2 Parts List

Table A-1. Printed Circuit Board Parts List

Designators	Qty	Description	Manufacturer	Part Number
AD1	1	0Ω / 0805	Any acceptable	
C1,C5,C7,C9,C17,C49	5	100pF ceramic 0805	Any acceptable	
C2	1	1nF ceramic 0805	Any acceptable	
C3	1	3.3µF / 10V tantalum size SMA	Any acceptable	
C4,C6,C16,C18, C33,C34,C35,C36	8	100nF ceramic 0805	Any acceptable	
C8,C10,C28,C38	4	10nF ceramic 0805	Any acceptable	
C11,C12,C13,C14,C15	5	2.2nF ceramic 0805	Any acceptable	
C19,C22	2	10nF / 3000V	Thomson	5ST410MCMCA
C20	1	470µF / 400 V	Philips Components	15746471
C21	1	22nF / 630V	WIMA	MKP10
C23,C24,C27	3	4.7µF / 20V size SMB	Vishay	293D475X_020B2_
C25,C43,C44,C45,C46	5	1µF / 20V size SMA	Vishay	293D105X_020A2_
C26	1	22nF / 20V ceramic 0805	Any acceptable	
C29	1	1µF / 50V electrolytic	Panasonic	ECA1HM010 Farnell code .320-1648
C30,C31,C32	3	220µF / 25V electrolytic	Farnell	order .286-382
C37	1	2.2µF / 35V size SMB	Vishay	293D225X_035B2_
C39,C40,C41	3	10pF / 25V ceramic 0805	Any acceptable	
C42,C47,C48	3	22pF / 25V ceramic 0805	Any acceptable	
C50	1	1µF / X2	EZK	CFAC 1M
D1	1	Green Display LED	Kingbright	KA-3828SGT
D2	1	Red Display LED	Kingbright	KA-3828EC
D3	1	V14E250	EPCOS	SOIV-S-10K250
D4	1s	Zener	ON Semiconductor	1SMB5925BT3
D5,D6,D7,D10,D11	5	1A / 600V Ultfafast	ON Semiconductor	MURS160T3
D8	1	1A 600V Standard Recovery Rectifier	ON Semiconductor	MRA4005T1
D13,D14,D15	3	1N4148	Fairchild	1N4148LL-34
D16	1	6A / 600V Rectifier	International Rectifier	KBPC606
D20	1	15V / 0.5W Zener	Philips	BZV55C15V
D21	1	9.1V / 0.5W Zener	Philips	BZV55C9V1
F1	1	Fuse holder	MULTICOMP	MCHTE15M
ISO1,ISO2	2	5.3 kV Optocoupler	Vishay	SFH6106
J1	1	HEADER 6X2	Any acceptable	
JP1	1	HEADER 3x1	Any acceptable	
J2, J4	2	PCB Terminal Block 3-pin, single level, 90° orientation	Weidmuller	LP 5.08/90

Table A-1. Printed Circuit Board Parts List (Continued)

Designators	Qty	Description	Manufacturer	Part Number
L1	1	680μH Axial Lead Power Choke	Coilcraft	PCH-45-684 / 680μH
L2	1	330μH Radial Bead	Fastron	09P/F-331K
L3	1	330μH Axial Bead	Fastron	HBCC-330K
L4	1	2x3.3mH 4A/250V Current Compensated Toroid Choke	Tesla Blatna	TL 34P
R1,R6	2	20kΩ resistor 1/10W 1% 0805	Any acceptable	
R2,R4,R37	3	1kΩ resistor 1/10W 1% 0805	Any acceptable	
R3,R5	2	1.5kΩ resistor 1/10W 1% 0805	Any acceptable	
R7,R8,R36	3	2kΩ resistor 1/10W 1% 0805	Any acceptable	
R10,R9, R34	2	330Ω resistor 1/10W 5% 0805	Any acceptable	
R11,R39,R55,R63	4	10MΩ resistor 1/10W 5% 0805	Any acceptable	
R12,R13,R14,R15	4	100Ω resistor 1/10W 5% 0805	Any acceptable	
R16,R17,R25,R30,R42, R57,R65,R69	8	10kΩ resistor 1/10W 5% 0805	Any acceptable	
R50,R54,R60, R68	4	10kΩ resistor 1/10W 1% 0805	Any acceptable	
R18	1	Inrush limiter	Rhopoint Components	SG190
R19,R22,R27	3	3.9Ω resistor 1/10W 1% 0805	Any acceptable	
R20	1	0.100Ω / 3W 1%	Isabellenhutte	PMA-A-R100-1
R21,R23,R24,R28, R29,R31,R32,R35	8	4.7kΩ resistor 1/10W 5% 0805	Any acceptable	
R26	1	5.1kΩ resistor 1/10W 5% 0805	Any acceptable	
R33	1	2.2kΩ resistor 1/10W 5% 0805	Any acceptable	
R38,R45,R51,R61	4	330kΩ resistor 1/10W 1% 1206	Any acceptable	
R40,R41,R46,R47, R53,R56, R62,R64	8	220kΩ resistor 1/10W 1% 1206	Any acceptable	
R43,R58,R66	3	470kΩ resistor 1/10W 1% 0805	Any acceptable	
R44,R59,R67	3	18kΩ resistor 1/10W 1% 0805	Any acceptable	
R48	1	15kΩ resistor 1/10W 1% 0805	Any acceptable	
R49	1	3.3kΩ resistor 1/10W 1% 0805	Any acceptable	
R52	1	240Ω resistor 1/10W 1% 0805	Any acceptable	
U1	1	Dual operating amplifier	ON Semiconductor	MC33502D
U2	1	Microcontroller	Motorola	68HC908MR8QFP
U3	1	Mini-DIP Intelligent power module	Mitsubishi	PS21353-N
U4	1	SMPS controller	ON Semiconductor	NCP1052B
U5	1	5V regulator	ON Semiconductor	MC78L05ACD
U6	1	Dual comparator	ON Semiconductor	LM339D
U7	1	4-MHz surface mount crystal	Farnell	HC49/4HSMX-A120E order. code .639-564

Schematics and Parts List

Appendix B. 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. *68HC908MR8, Advance Information* (document order number MC68HC908MR8/D), Motorola
10. *Low Cost High Efficiency Sensorless Drive for Brushless DC*

References

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

11. *Mini-DIP Intelligent Power Module PS21353-N* from Mitsubishi Semiconductor
12. *Mini-DIP IPM Application Note* - Mitsubishi Semiconductor
13. *Developer's Serial Bootloader for M68HC08* - AN2295/D Motorola application note describing how to use the serial bootloader

Appendix C. Glossary

AC — alternating current

ACIM — AC induction motor

ADC — analog-to-digital converter

ATD — analog-to-digital

BEMF — back-EMF

BLDC — brushless direct current (motor)

CW — CodeWarrior; compilers produced by Metrowerks

DC — direct current

DT — dead time; 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 quoted as a percentage.

ECLOVACBLDC — 3-phase AC/BLDC Low-voltage Power Stage

ECMTRLOVBLDC — 3-phase BLDC Low-voltage Motor-Brake SM40N + SG40N

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

ECOPTHIVACBLDC — 3-phase AC/BLDC High-voltage Power Stage + Optoisolation Board

ECMTRHIVBLDC — 3-phase BLDC High-voltage Motor-Brake SM40V + SG40N

Glossary

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

IC — integrated circuit

IDE — integrated development environment

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.

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

KITMMDS08MR32 — MMDS 68HC908MR32/16 development 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

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

PC — personal computer

PCM — PC master software for communication between PC and system

PLL (phase-locked loop) — 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 communications 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

SMPS — switched mode power supply

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

DRM048

Freescale Semiconductor, Inc.