Freescale Semiconductor, Inc.

3-Phase BLDC Motor Control with Quadrature Encoder using 56F800/E

Design of Motor Control Application Based on Processor Expert

1. Introduction of Application Benefit

This application note describes the design of a 3-phase BLDC (Brushless DC) motor drive based on Motorola's 56800/E hybrid controllers. The software design takes advantage of Processor Expert^TM (PE).

BLDC motors are very popular in a wide variety of applications. The BLDC motor loads a commutator, so it is more reliable than the DC motor. The BLDC motor also has advantages when compared to an AC induction motor. With the ability to generate rotor achieve magnetic flux with rotor magnets, BLDC motors are more efficient and are therefore used in high-end white goods (refrigerators, washing machines, dishwashers, etc.), high-end pumps, fans and other appliances which require high reliability and efficiency.

The heart of the application described is a speed closed-loop BLDC drive using a Quadrature Encoder. It is an example of a BLDC motor control system design using a Motorola hybrid controller with PE software support.

This application note includes the basic motor theory, system design concept, hardware implementation, and software design, including the PC master software visualization tool.

Contents

3-Phase BLDC Motor Control with Quadrature Encoder

MOTOROLA
intelligence everywhere™

digital dna™

2. Motorola Hybrid Controller Advantages and Features

The Motorola 56F800/E families are well suited for digital motor control, combining the DSP's calculation capability with the MCU's controller features on a single chip. These micro-controllers offer many dedicated peripherals, such as Pulse Width Modulation (PWM) module, Analog-to-Digital Converter (ADC), Timers, communication peripherals (SCI, SPI, CAN), on-board Flash and RAM. Generally, all family members are ideal for various motor controls.

A typical member of the 56F800 family, the 56F80x, provides the following peripheral blocks:

- Two Pulse Width Modulator modules (PWMA & PWMB), each with six PWM outputs, three Current Sense inputs, and four Fault inputs, fault tolerant design with dead time insertion, supporting both center-aligned and edge-aligned modes
- Twelve-bit Analog-to-Digital Converters (ADCs), supporting two simultaneous conversions with dual 4-pin multiplexed inputs; the ADC can be synchronized by PWM modules
- Two Quadrature Decoders (Quad Dec0 & Quad Dec1), each with four inputs, or two additional Quad Timers A & B
- Two dedicated General Purpose Quad Timers totaling 6 pins: Timer C with 2 pins and Timer D with 4 pins
- CAN 2.0 A/B Module with 2-pin ports used to transmit and receive
- Two Serial Communication Interfaces (SCI0 & SCI1), each with two pins, or four additional GPIO lines
- Serial Peripheral Interface (SPI), with a configurable 4-pin port, or four additional GPIO lines
- Computer Operating Properly (COP) timer
- Two dedicated external interrupt pins
- Fourteen dedicated General Purpose I/O (GPIO) pins, 18 multiplexed GPIO pins
- External reset pin for hardware reset
- JTAG/On-Chip Emulation (OnCE)
- Software-programmable, Phase Lock Loop-based frequency synthesizer for the core clock

In addition to the fast Analog-to-Digital Converter and the 16-bit Quad Timers, the most interesting peripheral from the BLDC motor control point of view is the Pulse Width Modulation (PWM) module. The PWM module offers a high degree of freedom in its configuration, permitting efficient control of the BLDC motor.

The PWM has the following features:

- Three complementary PWM signal pairs, or six independent PWM signals
- Complementary channel operation
- Dead time insertion
- Separate top and bottom pulse width correction via current status inputs or software
- Separate top and bottom polarity control
- Edge-aligned or center-aligned PWM signals
- 15 bits of resolution
- Half-cycle reload capability
- Integral reload rates from 1 to 16
- Individual software-controlled PWM outputs
- Mask and swap of PWM outputs

- Programmable fault protection
- Polarity control
- 20-mA current sink capability on PWM pins
- Write-protectable registers
- 20-mA current sink capability on PWM pins
- Write-protectable registers

The BLDC motor control utilizes the PWM block set in the complementary PWM mode (for control in complementary mode, see **Section 3.1.2**), permitting the generation of control signals for all switches of the power stage, with inserted dead time. The PWM outputs can be controlled separately by software, where setting the control signal to logical 0 or 1 enables/disables control signal. Another feature very useful in BLDC motor control is the channel swap function, which allows the immediate change of top and bottom transistors in the phase. These functions allow the rotor commutation and speed control to be split into two independent program parts. The state of the control signals can be changed immediately when required by the motor position (phase commutation) without changing the content of the PWM value registers. These changes can be accomplished asynchronously to the PWM duty cycle update.

The Quad Timer is an extremely flexible module, providing all of the required services related to time events. It has the following features:

- Each timer module consists of four 16-bit counters/timers
- Count up/down
- Counters are cascadable
- Programmable count modulo
- Maximum count rate equals the peripheral clock/2 when counting external events
- Maximum count rate equals the peripheral clock when using internal clocks
- Count once or repeatedly
- Counters are preloadable
- Counters can share available input pins
- Each counter has a separate prescaler
- Each counter has capture and compare capability

The BLDC motor application utilizes one channel of the Quad Timer module counting in quadrature mode. It enables rotor position sensing using the Quadrature Encoder. The second channel of the Quad Timer module is set to generate a time base for the speed controller.

The Quadrature Decoder is a module that provides decoding of position signals from a Quadrature Encoder mounted on a motor shaft. It has the following features:

- Includes logic to decode quadrature signals
- Configurable digital filter for inputs
- 32-bit position counter
- 16-bit position difference counter
- Maximum count frequency equals the peripheral clock rate
- Position counter can be initialized by software or external events
- Preloadable 16-bit revolution counter
- Inputs can be connected to a general purpose timer to aid low-speed velocity

The BLDC motor application utilizes the Quadrature Decoder connected to Quad Timer module A. The Decoder's digital input filter filters the Encoder's signals, but does not make use of its decoding functions, freeing the decoder's digital processing capabilities to be used by another application.

The BLDC motor application utilizes the Quadrature Decoder connected to Quad Timer module A. It uses the Decoder's digital input filter to filter the Encoder's signals, but does not make use of its decoding functions, so the decoder's digital processing capabilities are free to be used by another application.

3. Target Motor Theory

A brushless DC (BLDC) motor is a rotating electric machine; the stator is a classic 3-phase stator like that of an induction motor, and the rotor has surface-mounted permanent magnets (see **Figure 3-1**).
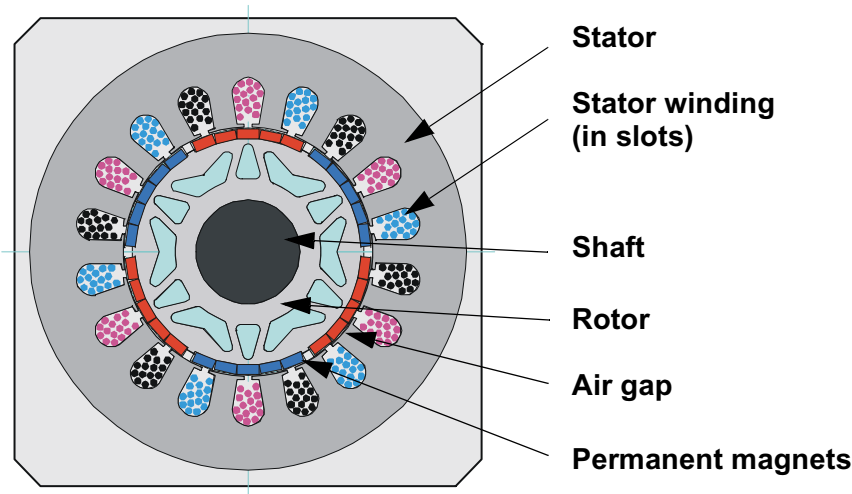

Figure 3-1.   Cross Section of BLDC Motor

In this respect, the BLDC motor is equivalent to an inverted DC commutator motor, in which the magnet rotates while the conductors remain stationary. In the DC commutator motor, the current polarity is reversed by the commutator and brushes. However, in the brushless DC motor, the polarity reversal is performed by power transistors switched in synchronization with the rotor position. Therefore, BLDC motors often incorporate either internal or external position sensors to sense the actual rotor position, or the position can be detected without sensors.

3.1   Digital Control of a BLDC Motor

The BLDC motor is driven by rectangular voltage strokes coupled with the given rotor position; see **Figure 3-2**. The generated stator flux, together with the rotor flux, which is generated by a rotor magnet, defines the torque and thus the speed of the motor. To get the maximum generated torque, the voltage strokes must be applied to the 3-phase winding system, so that the angle between the stator flux and the rotor flux is kept close to 90°. To meet this criteria, the motor requires electronic control for proper operation.
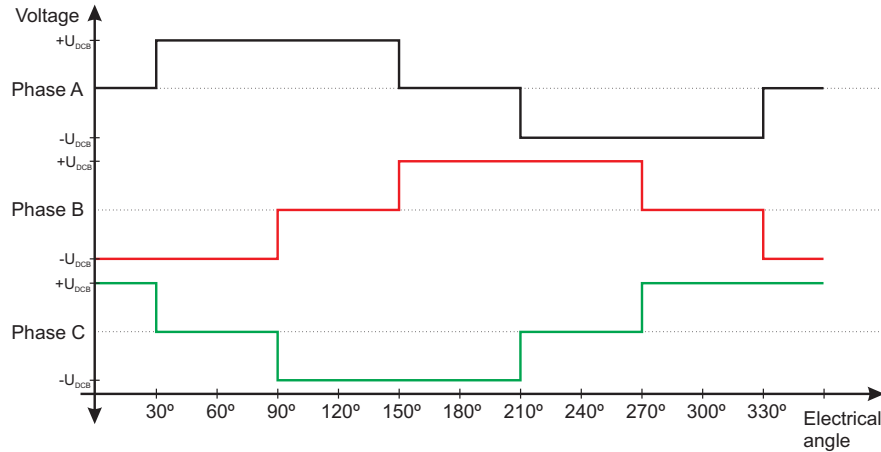
Figure 3-2.   Voltage Strokes Applied onto the 3-ph BLDC Motor

A standard 3-phase power stage is used for the common 3-phase BLDC motor; an example is illustrated in **Figure 3-3**. The power stage utilizes six power transistors with independent switching. The power transistors may be switched to independent or complementary mode.
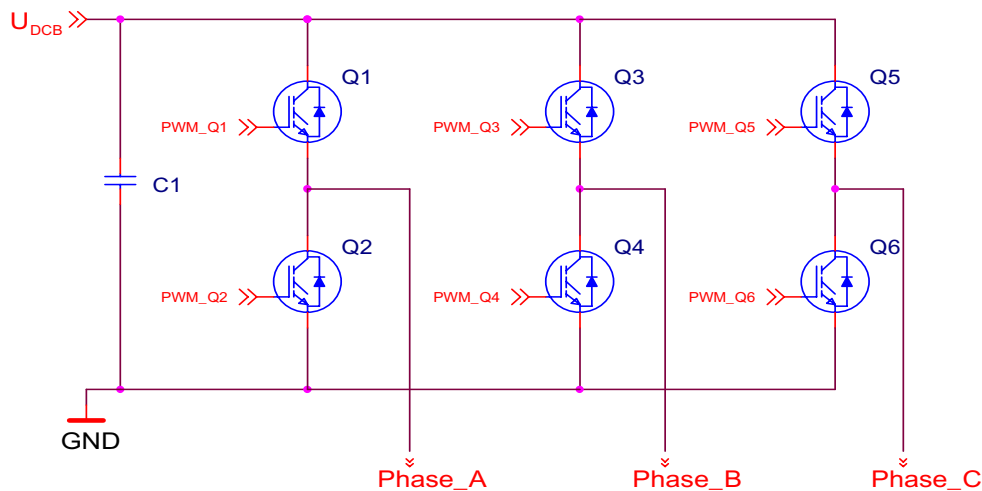
Figure 3-3.   3-phase BLDC Power Stage

The 3-phase power stage supplies two motor phases concurrently in both modes. As shown in **Figure 3-2**, the third phase is not powered, so there are six possible voltage vectors that are applied to the BLDC motor. **Figure 3-2** shows the maximum voltage amplitude applied to the BLDC motor, which is equal to the DCBus voltage. The lower voltage is generated using a PWM technique; see **Figure 3-4** and **Figure 3-5**.

There are two basic types of power transistor switching: independent switching and complementary switching, detailed in the following sections.

3.1.1 Independent Switching of Power Transistors

With independent switching, only two transistors are switched on when the current is conducted from the power supply to the phase of the BLDC motor. In one phase, the top transistor is switched on; in the second phase, the bottom transistor is switched on and the third phase is not powered. During freewheeling, all transistors are switched off; see **Figure 3-4**.
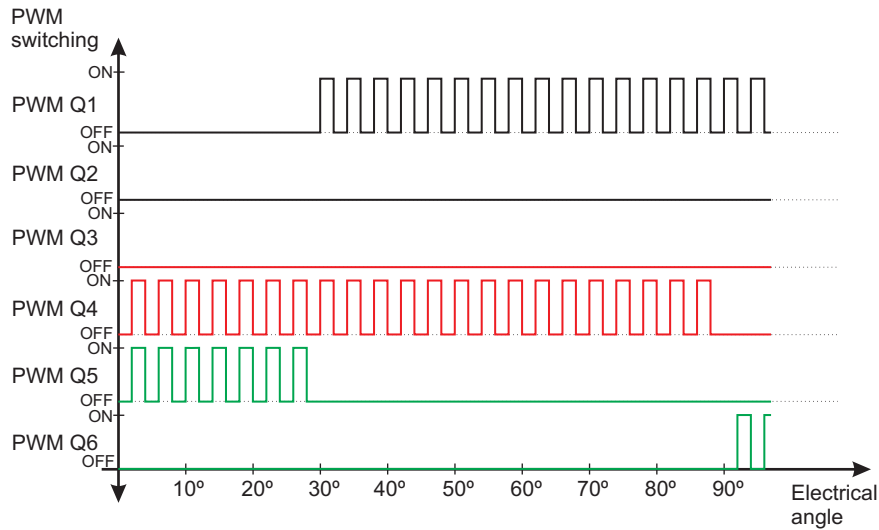
Figure 3-4.   Independent Switching of Power Transistors

3.1.2 Complementary Switching of Power Transistors

With complementary switching, two transistors are switched on when the phase of the BLDC motor is connected to the power supply. But during freewheeling, all the transistors are switched off with independent switching, the current continues to flow in the same direction through freewheeling diodes, and falls to zero. With complementary switching, the opposite occurs: transistors are switched on during freewheeling, so the current is able to flow in the opposite direction. **Figure 3-5** depicts complementary switching.

Note: Both of the switching modes described can work in bipolar or unipolar mode. **Figure 3-4** and **Figure 3-5** illustrate the bipolar switching mode. The application presented utilizes the complementary unipolar PWM mode.
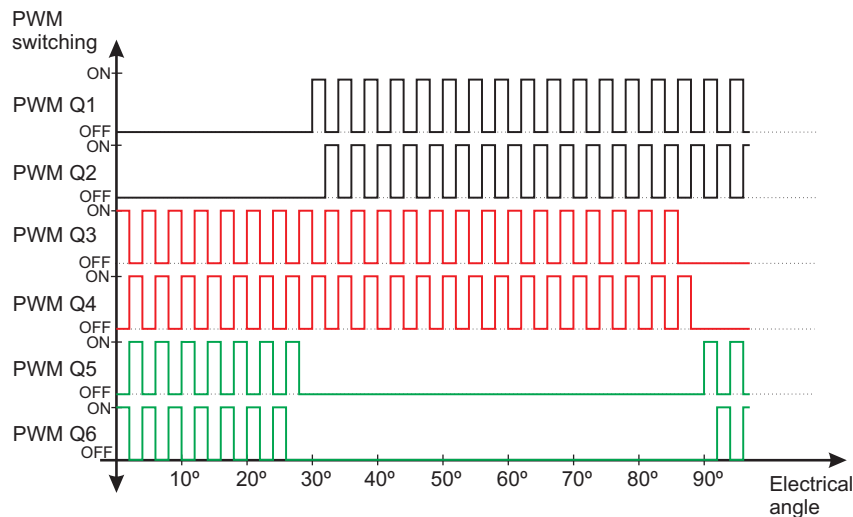

Figure 3-5.   Complementary Switching of Power Transistors

3.1.3 Commutation

Commutation creates a rotation field. As explained previously, proper operation of a BLDC motor requires the angle between stator and rotor flux to remain close to 90°. Six-step control yields a total of six possible stator flux vectors. The stator flux vector must be changed at a certain rotor position.

The rotor position is usually detected by Hall Sensors. The Hall Sensors directly detect the commutation moment. The application presented uses the Quadrature Encoder to sense rotor position. Therefore, the rotor position must be translated to determine the commutation moment.

The electrical revolution can be divided into six sectors. Each sector corresponds to a certain stator flux vector, as illustrated in **Figure 3-6**. The commutation sequence is illustrated in tables **Table 3-1** and **Table 3-2**.
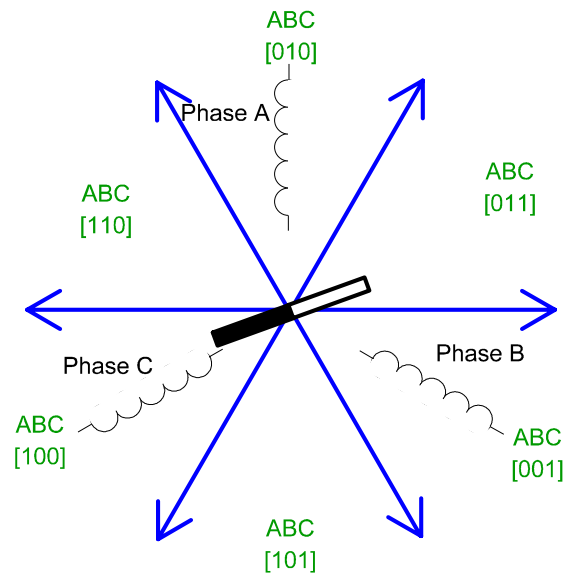
Figure 3-6.   Stator Flux Vectors at Six-Step Control

The next two figures depict the commutation process. The actual rotor position in **Figure 3-7** corresponds to the sector ABC[110]; see **Figure 3-6**. The actual voltage pattern can be derived from **Table 3-2**. Phase A is connected to the positive DCBus voltage by the transistor Q1; Phase C is connected to the ground by transistor Q6; Phase B is unpowered.

As soon as the rotor reaches a certain position (see **Figure 3-7**), the sector is changed from ABC[110] to ABC[100]. A new voltage pattern is selected from **Table 3-2** and applied to the BLDC motor.

As demonstrated, there is no possibility with a six-step control technique of keeping a precise 90° angle between the rotor flux and the stator flux. The actual angle varies from 60° to 120°.

The commutation is repeated every 60 electrical degrees. The commutation event is critical for its angular (time) accuracy; any deviation causes torque ripples and, therefore, speed variation.
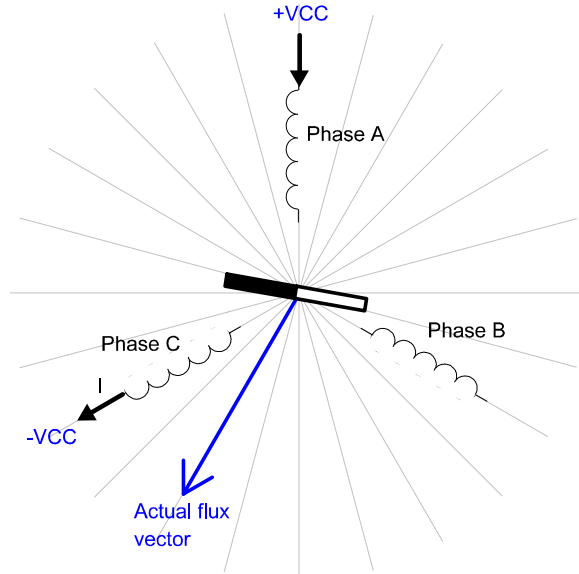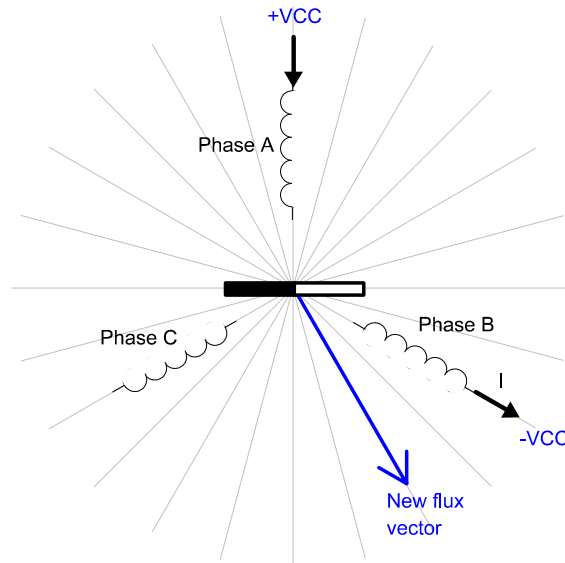
Figure 3-7.   Situation Prior to Commutation

Figure 3-8.   Situation Following Commutation

**Table 3-1.   Commutation Sequence for Clockwise Rotation**

Control word [ABC]			Phase A	Phase B	Phase C
1	0	0	$-V_{DCB}$	$+V_{DCB}$	NC
1	0	1	NC	$+V_{DCB}$	$-V_{DCB}$
0	0	1	$+V_{DCB}$	NC	$-V_{DCB}$
0	1	1	$+V_{DCB}$	$-V_{DCB}$	NC
0	1	0	NC	$-V_{DCB}$	$+V_{DCB}$
1	1	0	$-V_{DCB}$	NC	$+V_{DCB}$

Table 3-2.   Commutation Sequence for Counter Clockwise Rotation

Control word [ABC]			Phase A	Phase B	Phase C
1	0	0	$+V_{DCB}$	$-V_{DCB}$	NC
1	1	0	$+V_{DCB}$	NC	$-V_{DCB}$
0	1	0	NC	$+V_{DCB}$	$-V_{DCB}$
0	1	1	$-V_{DCB}$	$+V_{DCB}$	NC
0	0	1	$-V_{DCB}$	NC	$+V_{DCB}$
1	0	1	NC	$-V_{DCB}$	$+V_{DCB}$

3.1.3.1   Quadrature Encoder versus Hall Sensors

The BLDC motor application uses a Quadrature Encoder for rotor position sensing. The Quadrature Encoder output consists of three signals. Two phases, A and B, represent the rotor position, and an Index pulse defines the zero position. All Quadrature Encoder signals are depicted in **Figure 3-9**. Compared with Hall Sensors, there are some differences which affect the control algorithm. The main differences are that the Quadrature Encoder does not report the commutation moment and absolute position, as the Hall Sensors do.
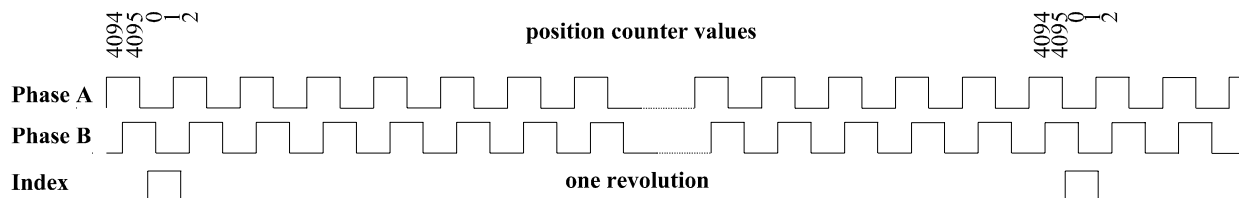
Figure 3-9.   Quadrature Encoder output signals

The differences between the Quadrature Encoder and Hall Sensors are summarized in **Table 3-3**.

**Table 3-3.   Differences between Quadrature Encoder and Hall Sensors**

Quadrature Encoder	Hall Sensors
Three outputs	Three outputs
Does not give absolute position	Gives absolute position
Give precise position	Gives 6 events per electrical revolution

The following sections describe the two control algorithms with the Quadrature Encoder. The first solution is commonly used and periodically scans the Quadrature Encoder. The second solution uses the specific advantages of the controller's peripherals and translates the Quadrature Encoder outputs directly into Hall Sensors signals. These internal signals are used as input for the commutation algorithm, which is implemented in the application presented.

3.1.3.2   Commutation with Periodical Scanning of Quadrature Encoder

The BLDC motor is commutated in the six defined moments. Since the Quadrature Encoder gives the precise position, one electrical revolution is divided into six sectors; see **Figure 3-10**. To recognize the commutation moment, it is necessary to scan the Quadrature Encoder's position very quickly. The frequency of scanning depends on the maximum rotor speed, the number of pole pairs and the required precision of commutation moment detection. The same scan frequency used in the PWM (16KHz) is satisfactory for common applications.

In this case, the rotor position can be scanned in the moment of a PWM reload interrupt. The algorithm translates the actual position into one of the six sectors. If a change of sector is detected, commutation is performed.
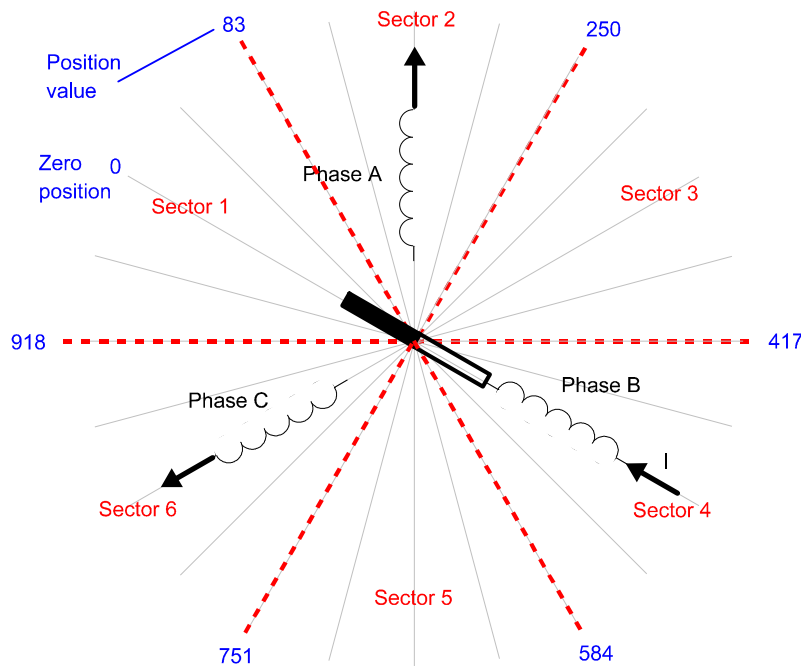

Figure 3-10.   Separation of One Electrical Revolution into Six Commutation Sectors

Note: **Figure 3-10** considers 500 pulses per mechanical revolution; both rising and falling edges counting; two pole pairs (500 x 4 / 2 = 1000 pulses per electrical revolution)

3.1.3.3   Direct Conversion of Quadrature Encoder Signals to a Commutation Sector

The second method is the direct conversion of Encoder output to commutation sectors. The conversion is completely automatic and provided by hardware. The advantage of this method is that an interrupt rate depends on the actual motor speed, while in the previous method, there is a constantly high interrupt rate in order to scan the actual motor position. A description of the direct conversion operation follows:

The electrical revolution is divided into six sectors, shown in **Figure 3-10**. The rotor position is scanned by the quadrature counter, which has its inputs connected to the Quadrature Decoder. After rotor alignment is in a known position, both compare registers are set to values that correspond to sector borders and the counter is set to zero; see **Figure 3-11**. When the motor starts to move, the counter starts to count the pulses of the Quadrature Encoder. Nothing happens until the counter reaches one of the compare values.
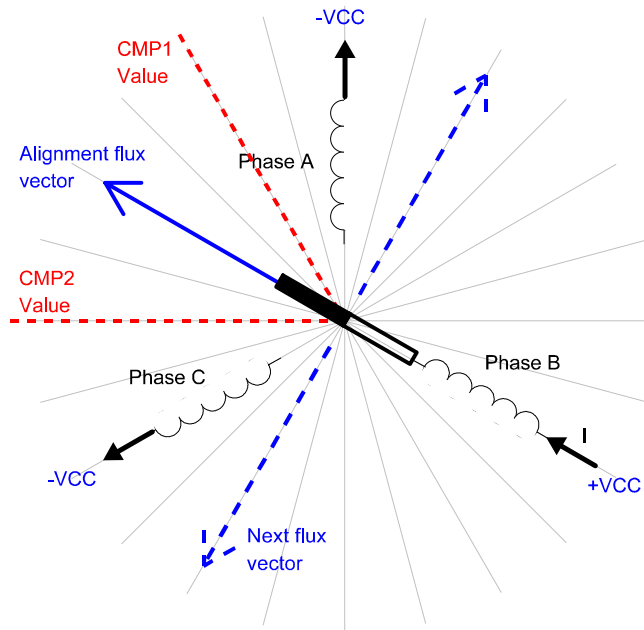
Figure 3-11. Situation Prior to First Compare Following Alignment

The commutation interrupt is called as soon as the counter reaches one of the compare values. The commutation interrupt recognizes the spin direction and sets new values into the compare registers. The new values correspond to the next commutation sector in a clockwise or counter-clockwise direction, according to the actual spin direction. The control word, which saves the actual commutation sector, is then updated and the new voltage pattern is applied to the motor. The corresponding voltage pattern is taken from **Table 3-1** and **Table 3-2**. **Figure 3-12** depicts the new situation after commutation.
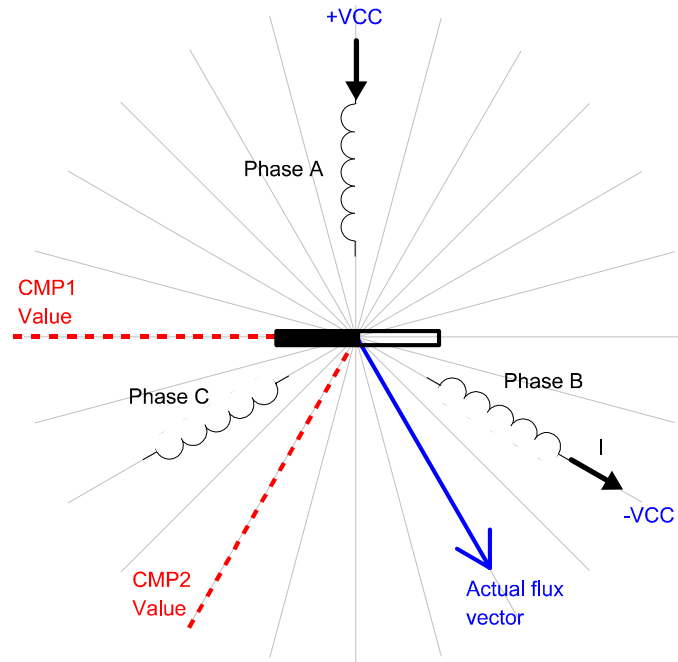
Figure 3-12.   Situation Following Commutation

We can see that the commutation interrupt is called six times per electrical revolution. Note that the commutation interrupt is called in the same moment as when we use Hall Sensors. Thus, we can use the same control routines, which are included in the Processor Expert.

3.1.4 Position Alignment

Since the Quadrature Encoder doesn't give the absolute position, it's important to know the exact rotor position before the motor is started. One possible and very easily implemented method is to align the rotor to a predefined position. The motor is powered by a defined static voltage pattern and the rotor aligns to predefined position. This alignment is done only once, during the initial motor start-up. **Figure 3-13** shows the position of the aligned rotor. After alignment, the compare registers of the position counter are set to +/-30 electric degrees from the alignment position in order to preset the commutation sector; see **Section 3.1.3.3**. The next voltage vector is set to be orthogonal to the alignment position. The resultant direction of the voltage vector is given by polarity of applied voltage.
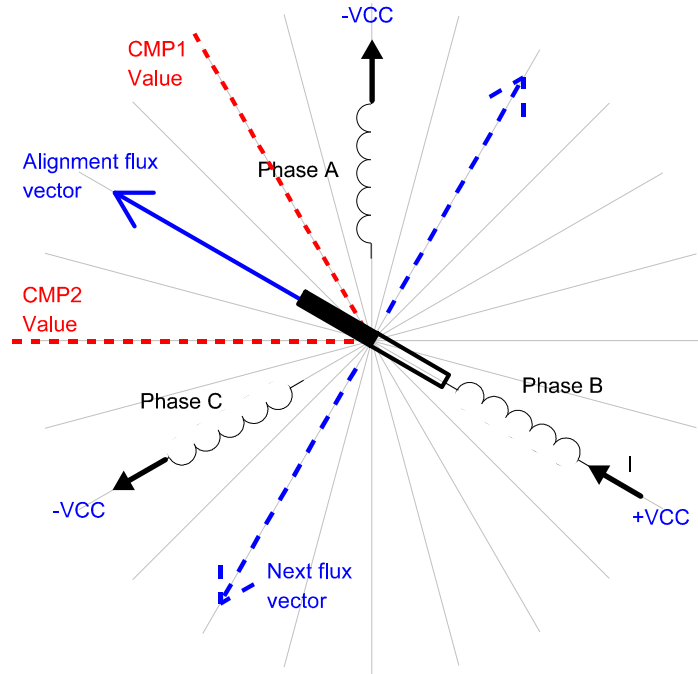
Figure 3-13.   Alignment Rotor Position

3.1.5 Speed Control

Commutation ensures the proper rotor rotation of the BLDC motor, while the motor speed depends only on the amplitude of the applied voltage. This amplitude is changed by the PWM technique. The required speed is controlled by a speed controller, implemented as a conventional PI controller. The PI controller compares the actual speed to the required speed and, using the difference, calculates duty cycle with the voltage amplitude required to correct the discrepancy.
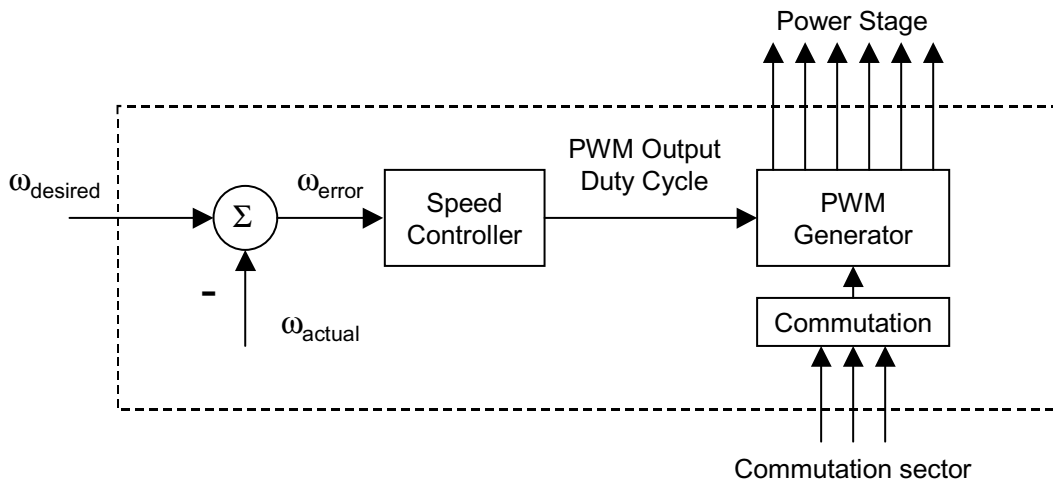
Figure 3-14.   Speed Controller

The speed controller calculates a Proportional-Integral (PI) algorithm according to equations below:

$$u(t) = K_c \left[e(t) + \frac{1}{T_I} \int_0^\tau e(\tau) d\tau \right] \qquad \text{(EQ 3-1.)}$$

Transformation to a discrete time domain with an integral approximation by a Backward Euler method yields the following equations for the numerical PI controller calculation:

$$u(k) = u_P(k) + u_I(k) \qquad \text{(EQ 3-2.)}$$

$$u_P(k) = K_c \cdot e(k) \qquad \text{(EQ 3-3.)}$$

$$u_I(k) = u_I(k-1) + K_c \frac{T}{T_I} \cdot e(k) \qquad \text{(EQ 3-4.)}$$

where:

$e(t), e(\tau)$	=	Input error in time t, τ
$e(k)$	=	Input error in step k
$w(k)$	=	Desired value in step k
$m(k)$	=	Measured value in step k
$u(k)$	=	Controller output in step k
$u_p(k)$	=	Proportional output portion in step k
$u_I(k)$	=	Integral output portion in step k
$u_I(k-1)$	=	Integral output portion in step k-1
T_I	=	Integral time constant
T	=	Sampling time
K_c	=	Controller gain
t, τ	=	Time
p	=	Laplace variable

4. System Concept

4.1 System Outline

The system is designed to drive a 3-phase BLDC motor. The application meets the following performance specifications:

- Voltage control of BLDC motor using Quadrature Encoder
- Targeted for hybrid controller EVM
- Running on 3-phase EVM Motor Board
- Control technique incorporates:
  — Voltage BLDC motor control with speed-closed loop
  — Both directions of rotation
  — Motoring mode
  — Start from any motor position without rotor alignment
  — Minimum speed 50rpm
  — Maximum speed 1000rpm (limited by power supply)
- Manual interface (Start/Stop switch, Up/Down push button control, LED indication)
- PC master software control interface (motor start/stop, speed set-up)

- PC master software monitor
  — PC master software graphical Control Page (required speed, actual motor speed, start/stop status, DCBus voltage level, system status)
  — PC master software Speed Scope (observes actual & desired speeds)
- DCBus under-voltage fault protection

The BLDC drive introduced here is designed to power a low-voltage BLDC motor equipped with a Quadrature Encoder, which is supplied with the EVM Motor Board. The motor's specifications are detailed in **Table 4-1**.

Table 4-1.   Specifications of the 3-phase BLDC Motor

Motor Specifications	Motor Type	3-Phase BLDC Motor 4Poles
	Speed Range	< 5000rpm
	Line Voltage	60V
	Phase Current	2A
Position Sensor Specifications	Sensor 1 Type	3-Phase Hall Sensors
	Sensor 2 Type	Quadrature Encoder 500 Pulses Per Revolution

4.2   Application Description

A standard system concept is chosen for the drive; see **Figure 4-1**. The system incorporates the following hardware boards:

- Power Supply 12V DC, 4Amps
- EVM Motor Board
- BLDC Motor IB23810 with Quadrature Encoder
- Evaluation Board

The system runs the main control algorithm and generates 3-phase PWM output signals for the BLDC inverter according to the user interface and feedback signals.
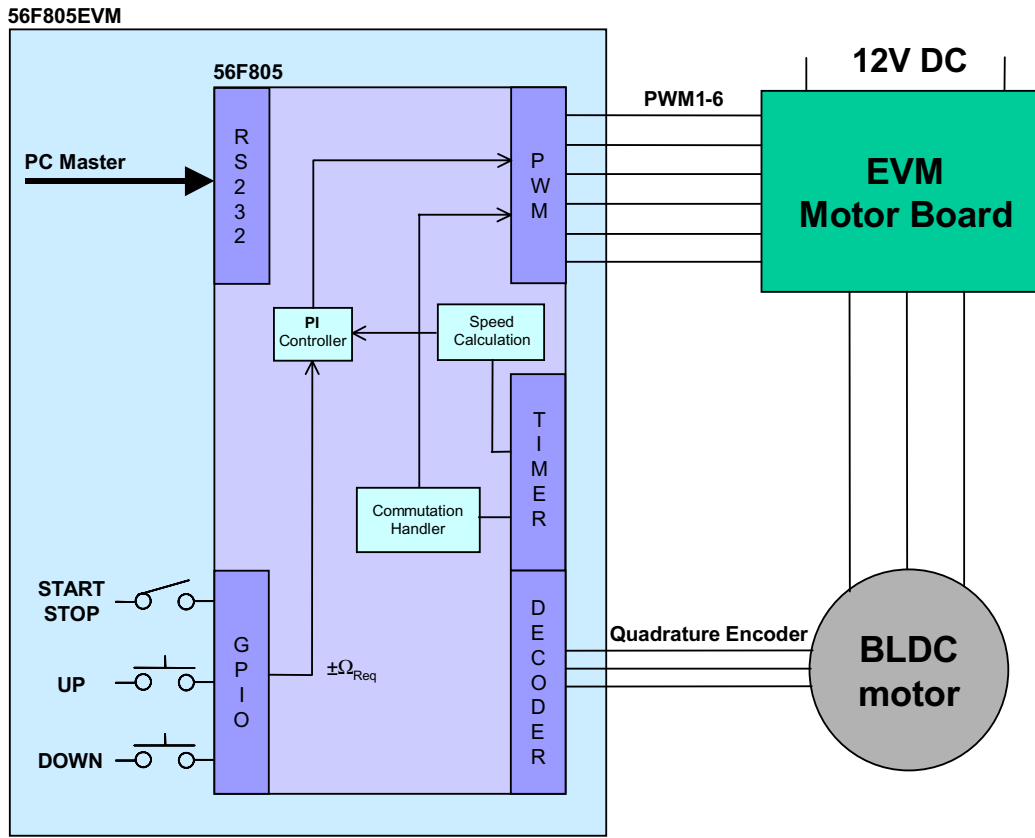
Figure 4-1.   Sample System Concept

The control process follows:

The state of the user interface is periodically scanned, while the speed of the motor is measured on each rising edge from the Quadrature Encoder; only one phase is used for speed measurement. The speed command is calculated according to the state of the control signals (Start/Stop switch, speed up/down buttons). The comparison between the actual speed command and the measured speed generates a speed error. The speed error is brought to the speed PI controller that generates a new corrected duty cycle. Together with the commutation algorithm, the duty cycle value creates the PWM output signals for the BLDC power stage.

The Quadrature Encoder signals are converted to six interrupts per electrical revolution. This interrupt provides the commutation algorithm.

If undervoltage occurs, the PWM outputs are disabled and the fault state is displayed.

4.3   Hardware Implementation

As already stated, the application runs on Motorola hybrid controlers using the EVM Boards and a dedicated 3-phase BLDC platform.

The application can be controlled by the following Motorola hybrid controllers:

- 56F805
- 56F8346

The application can run on an EVM motor board.

The application hardware set-up is shown in **Figure 4-2**. The system hardware set-up for a particular device varies only by the EVM board used. The application software is identical for all devices; the EVM and the chip differences are handled by the Processor Expert off-chip drivers for the particular EVM board.

Detailed application hardware set-up can be found in the device-specific targeting document for BLDC Motor Control Application with Quadrature Encoder.

The boards' User's Manuals describe the individual boards in detail and incorporate a schematic of the board, a description of individual function blocks and a bill of materials. Individual boards can be ordered from Motorola as standard products. The following section illustrates the configuration of an EVM motor board, with references to the documentation. Descriptions of all the mentioned boards and documents can be found at:

http://www.motorola.com/semiconductors

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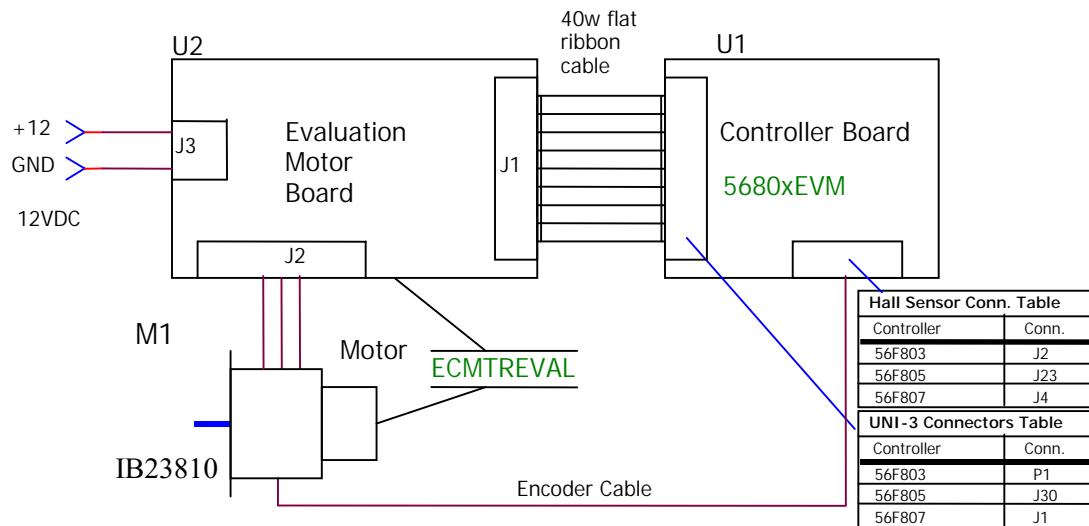
Figure 4-2.   Low-Voltage Evaluation Motor Hardware System Configuration

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

- M1 - IB23810 Motor
    — Supplied in kit ECMTREVAL - Evaluation Motor Board Kit
- U2 EVM Motor Board:
    — Supplied in kit with IB23810 Motor: ECMTREVAL - Evaluation Motor Board Kit
- U1 CONTROLLER BOARD for 56800/E Hybrid Controller:
    — Supplied as: 56F80x or 56F83xx EVM
   — Described in: 56F80x or 56F83xx EVMUM/D Evaluation Module Hardware User's Manual

5. Software Design

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

- Control algorithm data flow
- Software implementation

5.1 Data Flow

The control algorithm of a closed-loop BLDC drive is described in **Figure 5-1**. The individual processes are described in the following sections.
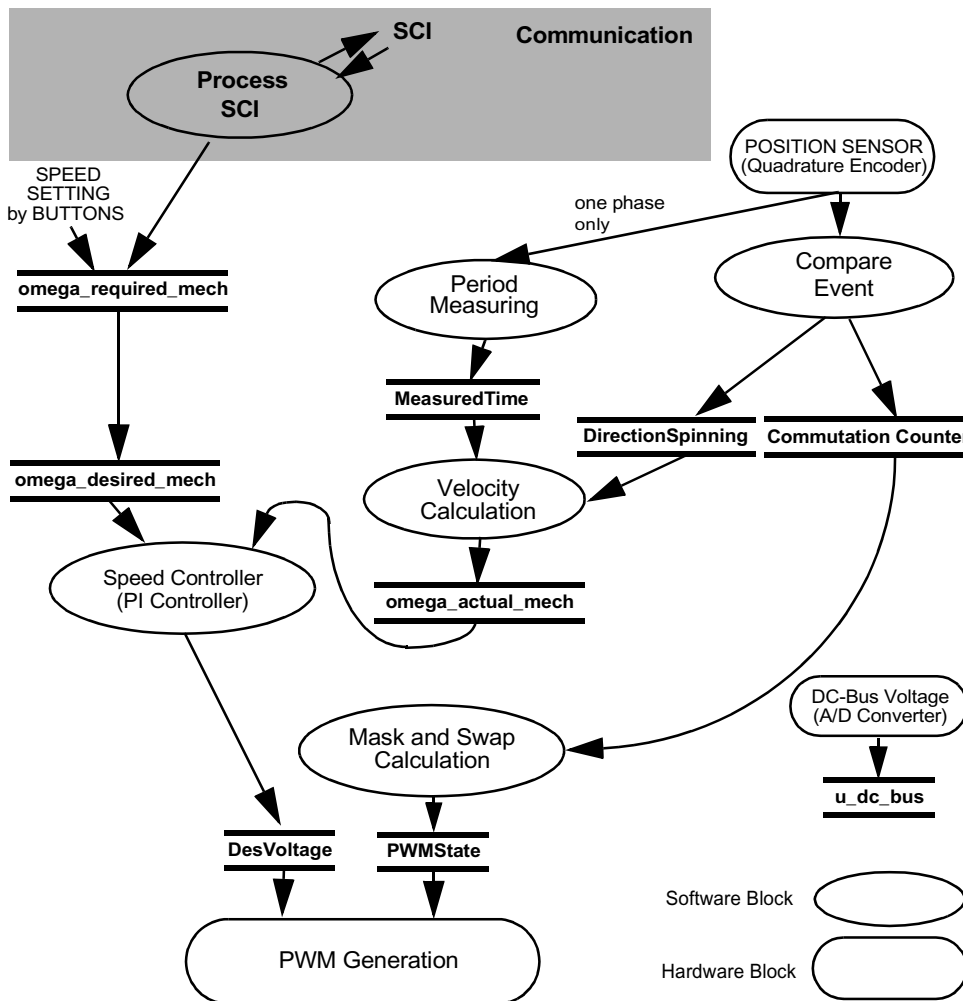
Figure 5-1.   Main Data Flow

The main data flow can be divided into four parts:

- Speed control
- Velocity calculation
- Rotor commutation
- DCBus voltage measurement

3-Phase BLDC Motor Control with Quadrature Encoder **MOTOROLA**

Speed control starts with the required speed *omega_required_mech*. This variable is set by user buttons or remotely by the PC, within allowed limits. The variable *omega_required_mech* is copied to *omega_desired_mech* at a defined moment. This variable is used as a shadow variable to avoid change of the required speed from the PC at any time. The variable *omega_desired_mech* is input to the speed PI controller as a reference value.

The variable *MeasuredTime* incorporates a time period of one phase of Quadrature Encoder. The time period is used for a speed calculation. Calculated speed, *omega_actual_mech*, is input to the speed PI controller as a secondary input. The PI controller output determines the duty cycle of the generated PWM output signals.

Timer A0, set as a quadrature counter, is used for the commutation. As soon as Timer A0 reaches any compare value, a compare interrupt is called. The interrupt routine saves the actual commutation sector to *CommutationCounter*. The *CommutationCounter* variable is input to the mask and swap calculation, which calculates the final shape of the output voltage. The output variable *PWMState* is written directly to the PWM block. The next task, provided by the interrupt routine, is calculation of the spin direction. The result, *DirectionSpinning*, is used for speed calculation. Finally, the routine updates both the compare registers of Timer A0.

The variable *u_dc_bus* contains the actual DCBus voltage. The value is used for undervoltage detection.

5.2   Compare Event

The process Compare Event is executed when Timer A0 reaches any compare value. The process identifies the spin direction and according to the spin direction, updates the compare values to define the limit for the new commutation sector. The number of pulses for one commutation sector is given by the constant NORMAL_COMMUTATION_INTERVAL. Where the number of pulses is not an integer there is a constant, SHORT_COMMUTATION_INTERVAL, which is applied once per electrical revolution. This constant ensures that the number of pulses counted through one electrical revolution is exactly the same as the number of pulses of the Quadrature Encoder. Where the number of pulses for one commutation sector is an integer, the constant SHORT_COMMUTATION_INTERVAL is not implemented.

5.2.1 Period Measuring and Velocity Calculation

The processes Period Measuring and Velocity Calculation read the time between the adjacent edges of one phase of the Quadrature Encoder, and calculate the actual motor speed, *omega_actual_mech*.

5.2.2 Speed Controller

This process compares the required and actual speeds and calculates the duty cycle of the PWM output signals. For detailed information, see **Section 3.1.5**.

5.2.3 Mask and Swap Calculation

This process performs a rotor commutation. As already mentioned, only two phases are powered by a six-step control. The proper PWM output can be generated by changing only the PWM value (duty cycle) registers, but this method has two disadvantages: First, the speed controller, which changes the duty cycle, affects the commutation algorithm (as a result of changing the duty cycle). Second, change in the duty cycle is synchronized with PWM reload, which may cause a delay between a proper commutation moment and the PWM reload. This is especially pronounced at high speeds, when the commutation period is very short.

The 56800/E families have two features dedicated to BLDC motor control: The ability to swap odd and even PWM generator outputs, and the ability to mask (disable) any PWM generator outputs. Thus, the same PWM pulse can be applied on the selected upper and lower switch of the inverter. These two

features allow the creation of a rotational field without changing the contents of the PWM value registers (duty cycle). The influence of masking and swapping on the PWM generator outputs is illustrated in **Figure 5-2**.

The commutation algorithm *bldchsCommHandlerComp* contains the output for *PWMState*, based on the actual control word (*CommutationCounter*). The structure of *PWMState* consists of two parts (variables). The first part, *PWMState.Swap*, defines the swapping of phases, and the second part, *PWMState.Mask*, defines the phase masking. These values are written directly to the PWM generator. The swapping value is written to the PWM Channel Control Register and the masking value to the PWM Output Control Register. The following paragraph describes in detail the commutation transition from the one to the next commutation sector as depicted in **Figure 3-7**, **Figure 3-8** and **Figure 5-2**.

For example, the even PWM value registers are set to 75% duty cycle. The odd PWM value registers are set to complementary value 100% - 75% = 25% duty cycle. The rotor in **Figure 3-7** is situated in sector ABC[110]. The state of masking and swapping is:

- Phase A
  Not masked, not swapped (Phase A is connected to a positive DCBus voltage)
- Phase B
  Masked/disabled
- Phase C
  Not masked, swapped (Phase C is connected to a negative DCBus voltage. The swap of Phase B provides the identical pulses on the upper switch of Phase A and the lower switch of Phase C and, conversely, on the lower switch of Phase A and the upper switch Phase C)
- See **Table 3-2**

As soon as the rotor passes the sector border (**Table 3-8**), the commutation is done in two steps:

- Swap Phase B (unconnected phase)
- Mask/disable Phase C; unmask/enable Phase B

After commutation, the state of masking and swapping is:

- Phase A
  Not masked, not swapped (Phase A is connected to a positive DCBus voltage)
- Phase B
  Not masked, swapped (Phase B is connected to a negative DCBus voltage.)
- Phase C
  Masked/disabled
- See **Table 3-2**

This commutation process is repeated six times per electrical revolution.

Note: In complementary switching mode, it is necessary to use the software control feature for masking. For independent mode, it is possible to use the masking feature in the PWM Channel Control Register. This feature works properly in independent mode only.
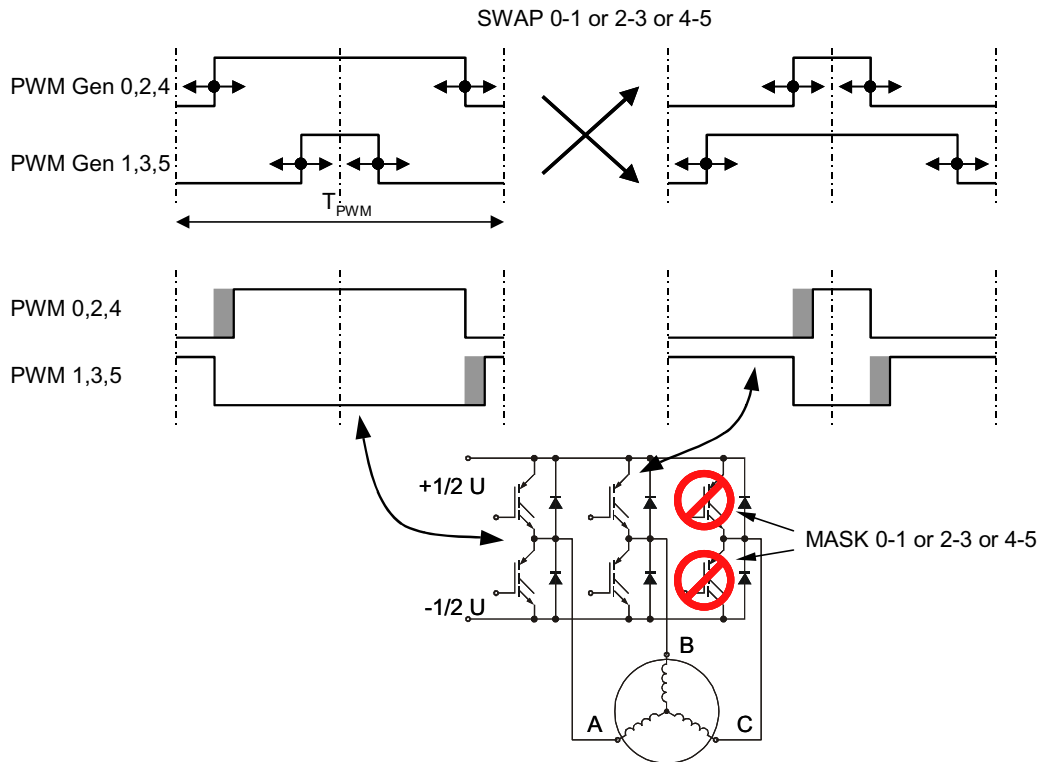
Figure 5-2. PWM Swapping and Masking for BLDC Motor

5.3 Software Implementation

This project is implemented using Processor Expert plug-in and Embedded Beans™ technology in the CodeWarrior Integrated Devolopment Environment (IDE). Processor Expert is designed for rapid application development of embedded applications on many platforms.

5.3.1 Embedded Beans

Embedded Beans are design components which encapsulate functionality of basic elements of embedded systems such as the controller's core; on-chip peripherals; stand-alone peripherals; virtual devices; and pure software algorithms. Embedded Beans allow access to these facilities via simple and uniform interface of properties, methods and events. Additional information can be found in Processor Expert help.

Table 5-1. lists the beans used in implementing the 56F805 application.

5.3.2 Bean Modules

Each peripheral on the hybrid controller chip or on the EVM board is accessible through a bean. Processor Expert generates the source code modules containing the implementation of methods controlling the hardware which provides the bean's functionality.

The following steps are required to generate the code:

- Add the beans to your project (Processor Expert tab of CodeWarrior's project panel)
- Set up the beans according to the hardware configuration
- Generate the source code

- Use beans' methods in your code (generated functions for bean's methods are named *beanName_MethodName*)

You should not modify generated modules; generated code can be found in the *Files* folder of CodeWarriors's project panel.

User modules, which are meant to be modified by the user, are found in the *User Modules* folder, where other user modules could also be added. The *Events* module (*events.c*) contains the handling routines for beans' events, those caused by interrupt, for example. The *Main* module (*projectName.c*) contains the function *main()* and all necessary declarations and variables.

All enabled methods are generated into appropriate bean modules during the code generation process. All Methods of each bean inserted into the project are visible as a subtree of bean in the *Project* panel (Processor Expert tab of CodeWarrior's project panel; see **Figure 5-3**).
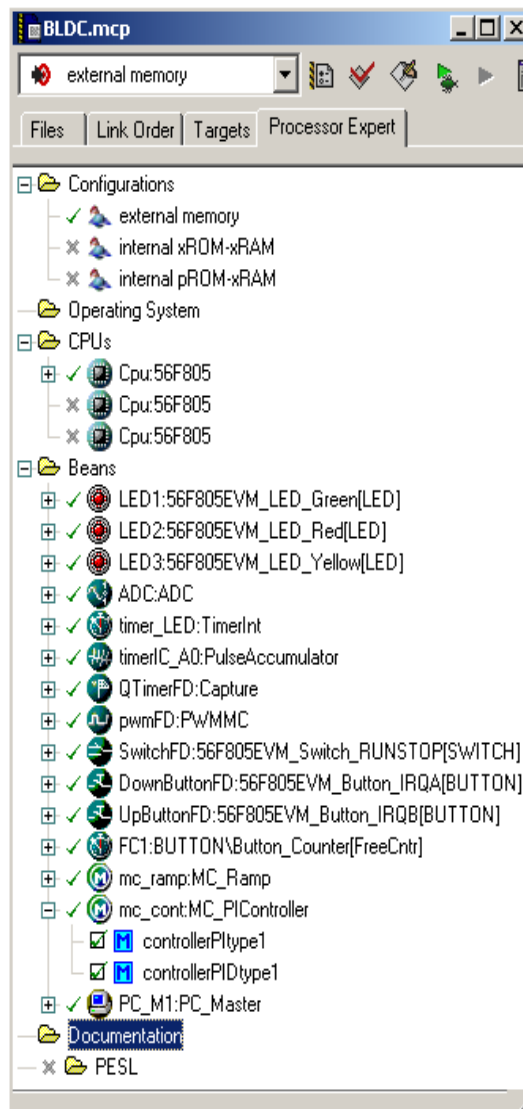
Figure 5-3.   Processor Expert's Project Panel with Beans and Methods

Table 5-1. Processor Expert's Beans

Bean name	Bean type	Description
PC_Master	PC_Master	Provides a serial communication with the PC master software running on PC using the SCI0 on-chip device. Communication speed is 9600bps
UpButtonFD, DownButtonFD	Button	Buttons are used for motor speed settings, with a 50rpm step. Buttons are connected to $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ inputs. Events: • *OnButton* - Invoked by pressing the button
FC1	FreeCounter	This timer bean watches the 50ms minimum delay between presses of the Up and Down buttons. The bean uses the on-chip device TMRD23.

Adc	ADC	Measures the DCBus voltage.
		
		The bean uses channel AD0 of the ADCA on-chip device.
		
		Methods:
		   • *Measure* - Start of the measurement (length of the measurement is 1.7us)
		   • *GetValue* - Reading of the measured value

LED1, LED2, LED3	LED_Green, LED_Red, LED_Yellow	Beans control the LEDs on the EVM connected to pins GPIOB0, GPIOB1 and GPIOB2.
		
		Methods:
		   • *On* - Switches the LED on
		   • *On* - Switches the LED on
		   • *Toggle* - Reverses the state of the LED

SwitchFD	Switch_RUNSTOP	Sets the applcation state to RUN/STOP. The bean uses the GPIOD5 input bit.

Table 5-1.   Processor Expert's Beans

Bean name	Bean type	Description
PwmFD	PWMMC	Algorithm which controls the state of the PWM on-chip device for motor applications. Frequency on the PWM output is 16kHz,dead time is 2.5µs. Pulse width is controlled by the application. Methods: • *SetDuty* - Sets PWM pulse width • *Load* - Updates control registers • *Swap* - Swaps PWM pairs • *OutputPadEnable* - Enables signal output onto CPU pins • *OutputPadDisable* - Disables signal output onto CPU pins
QuadFD	PulseAccumulator	Measures the position of the motor's rotor using a quadrature encoder signal. Measurement is done by timer (TMRA0) in quadrature mode. Methods: • *Enable* - Enables bean • *EnableEvent* - Enables bean's events • *DisableEvent* - Disables bean's events • *SetCounter* - Sets the impulse counter value • *GetCounterValue* - Gets the impulse counter value • *SetCompare1* - Sets compare register 1 value • *GetCompare1* - Gets compare register 1 value • *SetCompare2* - Sets compare register 2 value • *GetCompare2* - Gets compare register 2 value Events: • *OnCompare* - Pre-set number of impulses countdownL

Table 5-1. Processor Expert's Beans

Bean name	Bean type	Description

QtimerFD	Capture	Determines the current commutation sector from Quadrature Encoder input signal.
		
		Value is used for motor speed calculation and for further regulation.
		
		The bean utilizes TMRA1 on-chip device.
		
		Methods:
		  • *Enable* - Enables the bean.
		  • *GetCaptureValue* - Reads the pulse counter value
		
		Events:
		  • *OnCapture* - Occurs when a pulse is detected. The motor position is updated.
		  • *OnOverflow* - Occurs when the speed of the motor is too high and the counter overflows.

TimerFD	TimerInt	Timer base 1ms for the motor speed computation. It uses on-chip timer TMRA2. Events: • *OnInterrupt* - Computes the current speed of the motor
MC1	MC_BldcHallSensor	Software algorithm for motor commutation discovery. Methods: • *bldchsCommHandlerComp* - Gets the value from the motor winding table
MC2	MC_Ramp	Acceleration/decceleration ramp settings Methods: • *rampGetValue* - Gets value of the ramp
MC3	MC_PIController	Computation of motor speed using PI controller. Methods: • *controllerPItype1* - PI controller algorithm

5.3.3 Initialization

The *Main* routine calls *PE_low_level_init()* function generated by Processor Expert. This functions pre-sets the CPU internal peripherals to initial state according to the beans' settings.

Then the main method provides the following application initialization:

- Disables interrupt
- Starts measuring the DCBus voltage
- Sets application state
- Enables PulseAccumulator

- Sets up PI controller parameters
- Enables interrupt

After initialization, the main loop goes into an infinite background loop. This loop contains an application State Machine. The general software diagram shows the Main routine entered from Reset and the interrupt (event) states (see **Figure 5-4**).
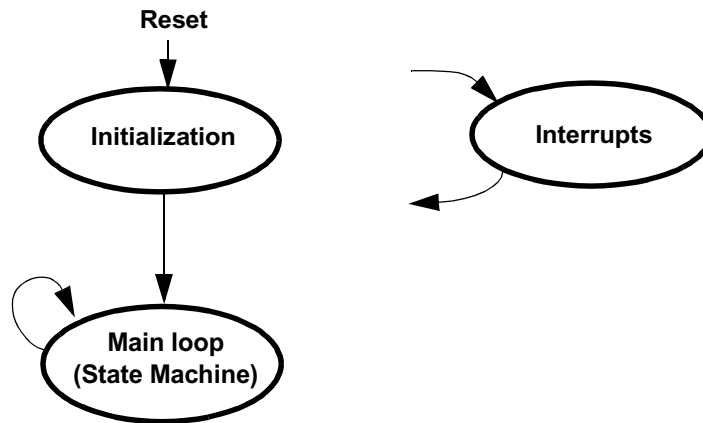
Figure 5-4. State Diagram - General Overview

5.3.4 Drive State Machine

The drive can be in any of the states shown in **Figure 5-5**, which illustrates the transition conditions among the drive states. The user can identify the current state by the green LED diode. In the *init* and *stop* states, the green LED diode blinks with a frequency of 2Hz. In the *fault* state, the green LED diode blinks with a frequency of 8Hz. During the *running* state, the green LED diode is continuously lit.
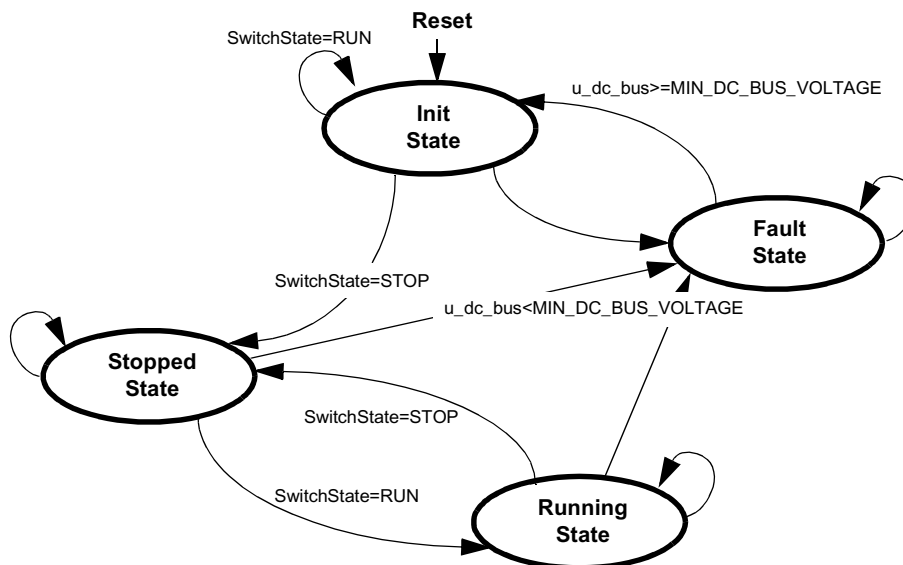
Figure 5-5. Drive State Machine Transitions

6. Implementation Notes

6.1 Scaling of Quantities

The BLDC motor control application uses a fractional representation for all real quantities except time. The N-bit signed fractional format is represented using 1.[N-1] format (1 sign bit, N-1 fractional bits). Signed fractional numbers (SF) lie in the following range:

$$-1.0 \leq SF \leq +1.0 - 2^{-[N-1]}$$ (EQ 6-1.)

For words and long-word signed fractions, the most negative number that can be represented is -1.0, whose internal representation is $8000 and $80000000, respectively. The most positive word is $7FFF or $1.0 - 2^{-15}$, and the most positive long-word is $7FFFFFFF or $1.0 - 2^{-31}$.

The following equation shows the relationship between real and fractional representations:

$$\text{Fractional Value} = \frac{\text{Real Value}}{\text{Real Quantity Range}}$$ (EQ 6-2.)

where:

Fractional Value is a fractional representation of the real value [Frac16]

Real Value is the real value of the quantity [V, A, RPM, etc.]

Real Quantity Range is the maximal range of the quantity, defined in the application [V, A, RPM, etc.]

6.1.1 DCBus Voltage Scaling

The DCBus voltage sense is defined by following equation:

$$u\_dc\_bus = \frac{V_{DC\_BUS}}{V_{MAX}} \cdot 32767$$

Where:

u_dc_bus = Variable of DCBus voltage

V_{DC_BUS} = Measured DCBus voltage

V_{MAX} = Maximum measurable DCBus voltage

V_{MAX} = 16V for the EVM Motor Board

6.1.2 PI Controller Parameters

The constant P was chosen as 0.2 ($26214 * 2^{-17}$), and the constant I was chosen as 0.3 ($31457 * 2^{-20}$) or 0.12 ($31457 * 2^{-18}$). A better response to speed error is achieved when the constant I is changed according to the actual speed. The constant I is equal to 0.3 from 50 to 200rpm. Over 200rpm, the constant I is equal to 0.12. These controller parameters were experimentally tuned.

6.1.3 Velocity Calculation

The constant OMEGA_ACTUAL_MECH_CONST is defined by the following:

Position Difference	=	1/500 rev (given by each rising edge of one phase of the Quadrature Encoder and two pole pairs motor)
Maximum Period Time	=	0.008 s (chosen according to required minimum speed)
v_{min}	=	60*(position difference)/(maximum period time) = 15rpm
v_{max}	=	100*v_{min} = 1500rpm (chosen according to required maximum speed)

OMEGA_ACTUAL_MECH_CONST = 32767*v_{min}/v_{max} = 327

7. PC Master Software

PC master software was designed to provide a debugging, diagnostic and demonstration tool for the development of algorithms and applications. It runs on a PC connected to the EVM via an RS-232 serial cable. A small program resident in the controller communicates with the PC master software to parse commands, return status information to the PC and process control information from the PC. PC master software uses part of Microsoft's Internet Explorer as the user interface. To enable PC master software operation on the hybrid controller target board application, add the PC master software bean to the project and configure it. This automatically includes the SCI driver and installs all necessary services. The SCI communication's default baud rate is 9600bd. It is set automatically by the PC master software driver and can be changed if needed. A detailed description of PC master software is provided in a dedicated User's Manual.

The 3-phase AC Motor V/Hz Speed Open Loop application utilizes PC master software for remote control from the PC. It enables the user to:

- Set the motor speed

PC master software reads and displays these variables to the user:

- Required and actual motor speed
- Application operational mode
- Start/stop status
- DCBus voltage

The PC master software Control Page is illustrated in **Figure 7-1**. Profiles of the required and actual speeds can be seen in the Speed Scope window.
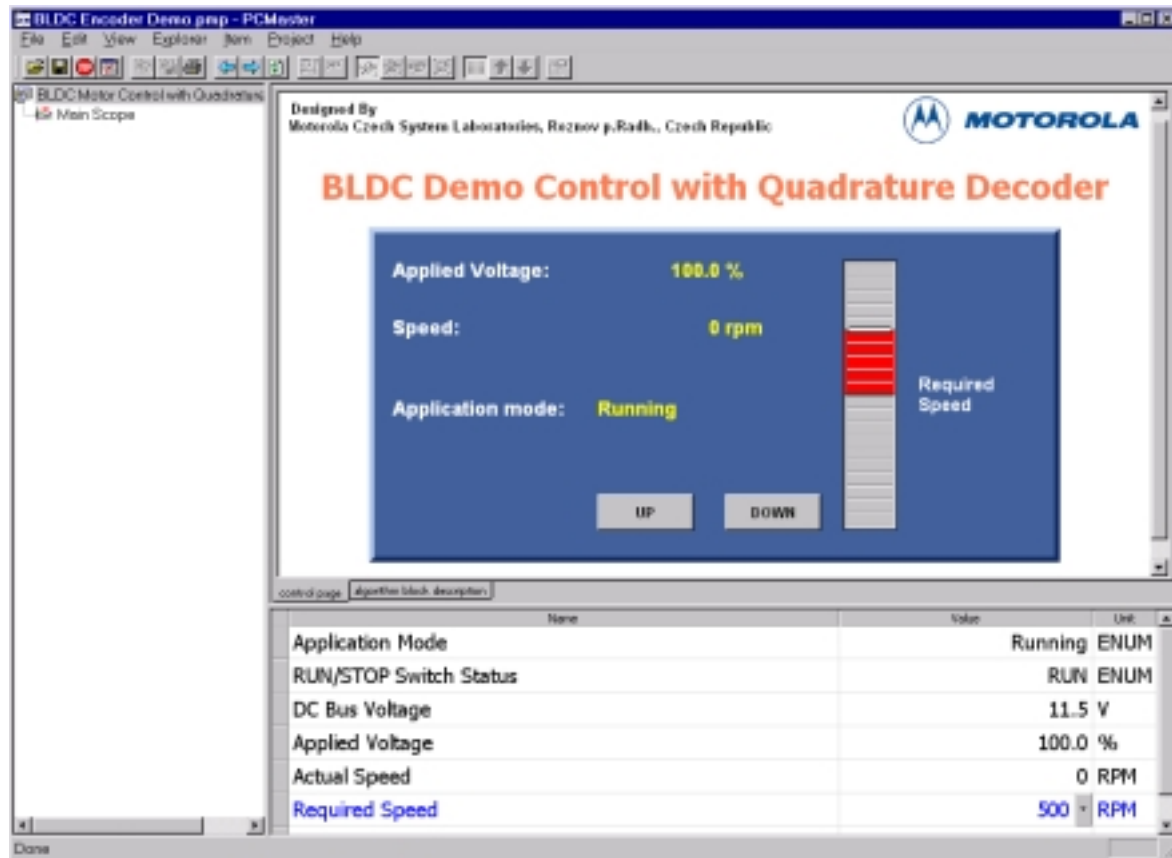
Figure 7-1.   PC Control Window

8.   References

[1] CodeWarrior for Motorola 56800 Embedded Systems, CWDSP56800, Metrowerks

[2] *DSP56800E Reference Manual*, DSP56800ERM/D, Motorola,

[3] *DSP56F800 Family Manual,* DSP56F800FM/D, Motorola,

[4] *DSP56F80x User's Manual*, DSP56F801-7UM/D, Motorola

[5] *MC56F8300 Peripheral User Manual,* MC56F8300UM/D, Motorola

[6] *DSP56F805 Evaluation Module Hardware User's Manual,* DSP56F805EVMUM/D, Motorola

[7] *MC56F8346 Evaluation Module Hardware User's manual,* MC56F8346EVMUM/D

[8] *3-Phase BLDC Motor Control with Hall Sensors Using DSP56F80x*, AN1916/D, Motorola 2001

[9] *Processor Expert Embedded Beans,* Processor Expert Help

[10] *BLDC Motor Control Application with Quadrature Encoder*, 805BLDCQETD/D, Motorola

[11] *BLDC Motor Control Application with Quadrature Encoder*, 8346BLDCQETD/D, Motorola

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

**For More Information On This Product,
Go to: www.freescale.com**