Freescale Semiconductor
Application Note

AN2948
Rev. 2, 09/2005

Three 3-Phase BDLC Motors with Speed Closed Loop, Driven by eTPU on MCF523x

Covers MCF523x and all eTPU-equipped Devices

by: Milan Brejl & Michal Princ
System Application Engineers
Roznov Czech System Center

This application note describes the design of multiple 3-phase brushless DC (BLDC) motors driven by Freescale's ColdFire MCF523x microprocessor. The application design takes advantage of the enhanced time processing unit (eTPU) module, which is used as a motor control co-processor. The eTPU completely handles the motor control processing required to drive three motors simultaneously, eliminating the microprocessor overhead for other duties.

BLDC motors are very popular in a wide array of applications. Compared to a DC motor, the BLDC motor uses an electric commutator (replacing the mechanical commutator), making it more reliable than the DC motor. In BLDC motors, rotor magnets generate the rotor's magnetic flux, making BLDC motors more efficient and well-suited for high-end white goods (refrigerators, washing machines, dishwashers, etc.), high-end pumps, fans, and other appliances that require high reliability and efficiency.

The concept of the application is to create a speed-closed loop BLDC driver using a Hall position sensor. It serves as an example of a BLDC motor control system design

Table of Contents

freescale semiconductor

using a Freescale microprocessor with the eTPU. It also illustrates the usage of dedicated motor control eTPU functions that are included in the motor control eTPU function set.

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

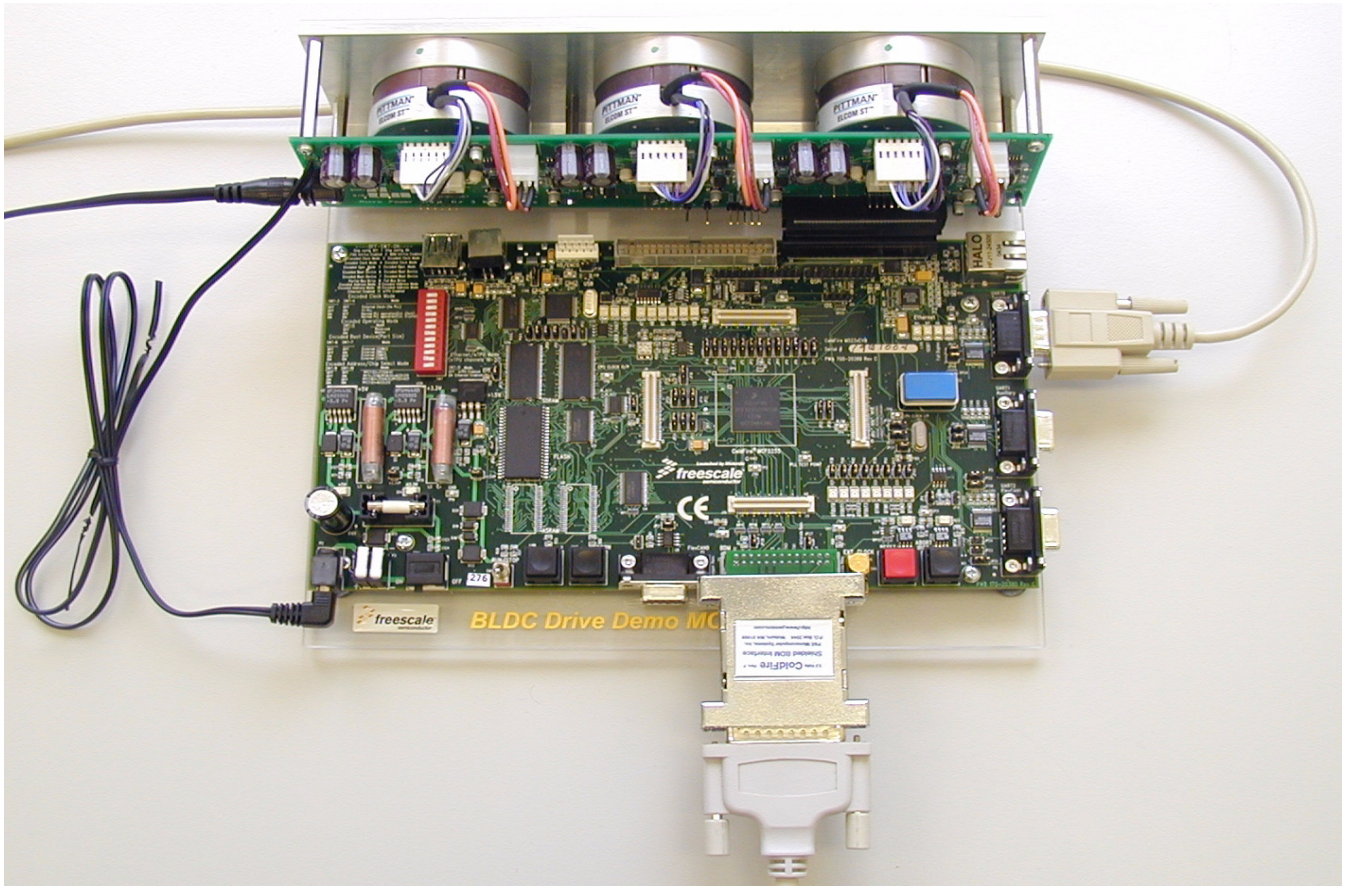

Figure 1. M523xEVB, Micro Power Stage for 3 BLDC Motors and 3 Pittman BLDC Motors

1 ColdFire MCF523x and eTPU Advantages and Features

1.1 ColdFire MCF523x Microprocessor

The MCF523x is a family of highly-integrated, 32-bit microprocessors based on the V2 ColdFire core. It features a 16- or 32-channel eTPU, 64 Kbytes of internal SRAM, a 2-bank SDRAM controller, four 32-bit timers with DMA request capability, a 4-channel DMA controller, up to two CAN modules, three UARTs, and a queued SPI. The MCF523x family has been designed for general purpose industrial control applications. It is also a high-performance upgrade for users of the MC68332.

This 32-bit device is based on the version 2 ColdFire reduced instruction set computer (RISC) core, operating at a core frequency of up to 150 MHz and a bus frequency of up to 75 MHz. On-chip modules include the following:

- V2 ColdFire core with an enhanced multiply-accumulate unit (EMAC) providing 144 Dhrystone 2.1MIPS @ 150 MHz
- eTPU with 16 or 32 channels, 6 Kbytes of code memory, and 1.5 Kbytes of data memory with eTPU debug support
- 64 Kbytes of internal SRAM
- External bus speed of half the CPU operating frequency (75 MHz bus @ 150 MHz core)
- 10/100 Mbps bus-mastering Ethernet controller
- 8 Kbytes of configurable instruction/data cache
- Three universal asynchronous receiver/transmitters (UARTs) with DMA support
- Controller area network 2.0B (FlexCAN module)
  - Optional second FlexCAN module multiplexed with the third UART
- Inter-integrated circuit (I2C) bus controller
- Queued serial peripheral interface (QSPI) module
- Hardware cryptography accelerator (optional)
  - Random number generator
  - DES/3DES/AES block cipher engine
  - MD5/SHA-1/HMAC accelerator
- 4-channel, 32-bit direct memory access (DMA) controller
- 4-channel, 32-bit input capture/output compare timers with optional DMA support
- 4-channel, 16-bit periodic interrupt timers (PITs)
- Programmable software watchdog timer
- Interrupt controller capable of handling up to 126 interrupt sources
- Clock module with phase locked loop (PLL)
- External bus interface module including a 2-bank synchronous DRAM controller
- 32-bit, non-multiplexed bus with up to 8 chip select signals that support page-mode FLASH memories

For more information, refer to Reference 1.

1.2 eTPU Module

The eTPU is an intelligent, semi-autonomous co-processor designed for timing control, I/O handling, serial communications, motor control, and engine control applications. It operates in parallel with the host CPU. The eTPU processes instructions and real-time input events, performs output waveform generation, and accesses shared data without the host CPU's intervention. Consequently, the host CPU setup and service times for each timer event are minimized or eliminated.

The eTPU has up to 32 timer channels, in addition to having 6 Kbytes of code memory and 1.5 Kbytes of data memory that store software modules downloaded at boot time, and can be mixed and matched as needed for any application.

The eTPU provides more specialized timer processing than the host CPU can achieve. This is partially due to the eTPU implementation, which includes specific instructions for handling and processing time events. In addition, channel conditions are available for use by the eTPU processor, thus eliminating many branches. The eTPU creates no host CPU overhead for servicing timing events.

For more information, refer to Reference 6.

2 Target Motor Theory

A brushless DC (BLDC) motor is a rotating electric machine where the stator is a classic three-phase stator, like that of an induction motor, and the rotor has surface-mounted permanent magnets (see Figure 2).

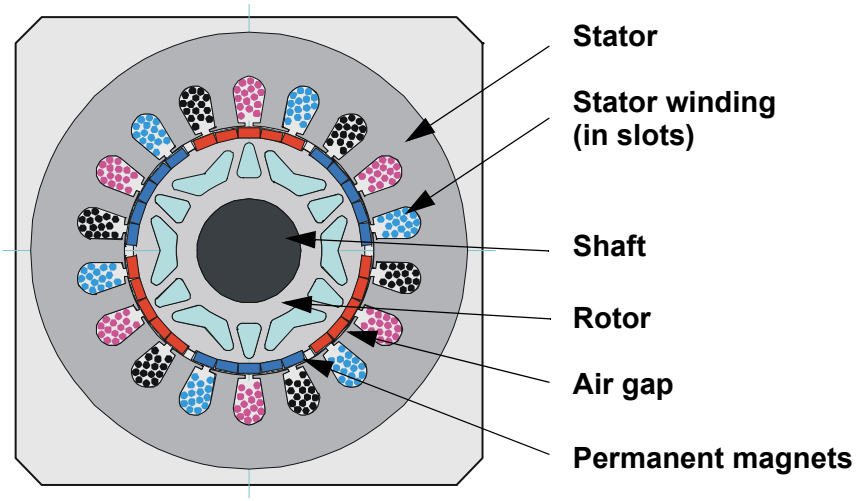

Figure 2. BLDC Motor - Cross Section

In this respect, the BLDC motor is equivalent to a reversed DC commutator motor, in which the magnet rotates while the conductors remain stationary. In the DC commutator motor, the current polarity is altered by the commutator and brushes. Unlike the brushless DC motor, the polarity reversal is performed by power transistors switching 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.

2.1 Digital Control of a BLDC Motor

The BLDC motor is driven by rectangular voltage strokes coupled with the given rotor position (see Figure 3). The generated stator flux interacts with the rotor flux, which is generated by a rotor magnet and defines the torque and thus the speed of the motor. The voltage strokes must be properly applied to two phases of the three-phase winding system so that the angle between the stator flux and the rotor flux is kept as close to 90° as possible, to get the maximum generated torque. Therefore, the motor requires electronic control for proper operation.

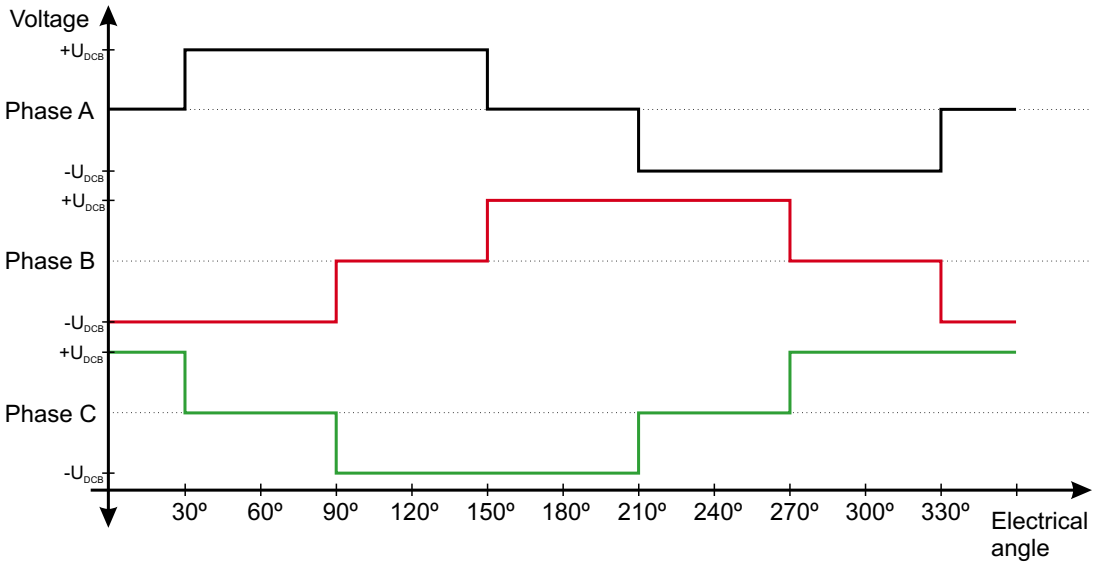

Figure 3. Voltage Strokes Applied to the 3-Phase BLDC Motor

For the common 3-phase BLDC motor, a standard 3-phase power stage is used (see Figure 4). The power stage utilizes six power transistors that operate in either an independent or complementary mode.

In both modes, the 3-phase power stage energizes two motor phases concurrently. The third phase is unpowered (see Figure 3). Thus, we get six possible voltage vectors that are applied to the BLDC motor using a pulse width modulation (PWM) technique (see Figure 5). There are two basic types of power transistor switching schemes: independent and complementary. Both switching modes are able to work in bipolar or unipolar mode. The presented application utilizes the complementary bipolar PWM mode.

For more information about PWM techniques, refer to Reference 13.

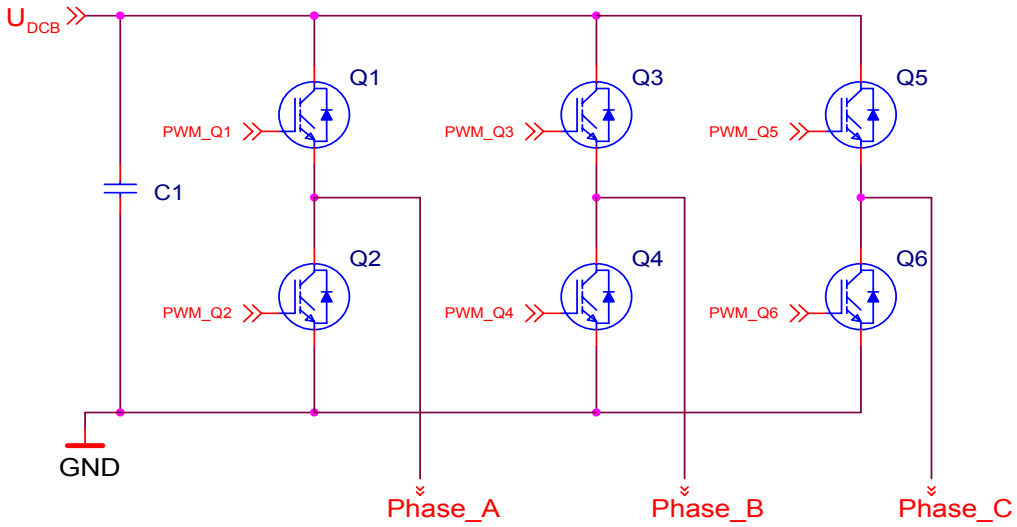

Figure 4. 3-Phase BLDC Power Stage

2.1.1 Commutation

Commutation provides the creation of a rotational field. As mentioned earlier, for proper operation of a BLDC motor, it is necessary to keep the angle between the stator and rotor flux as close to 90° as possible. We get a total of six possible stator flux vectors with a six-step control. The stator flux vector must be changed at specific rotor positions, which are usually sensed by the Hall sensors. The Hall sensors generate three signals that also consist of six states. Each of the Hall sensors' states correspond to a certain stator flux vector. All of the Hall sensors states, with corresponding stator flux vectors, are illustrated in Figure 5.

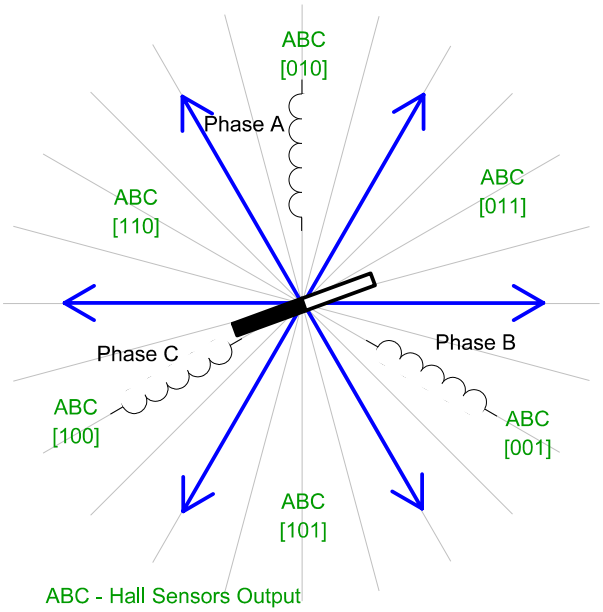

Figure 5. Stator Flux Vectors at Six-Step Control

The next two figures depict the commutation process. The actual rotor position in Figure 6 corresponds to the Hall sensors state ABC[110] (see Figure 5). Phase A is connected to the positive DC bus voltage by the transistor Q1; phase C is connected to the ground by transistor Q6, and phase B is unpowered.

As soon as the rotor reaches a certain position (see Figure 7), the Hall sensors state changes its value from ABC[110] to ABC[100]. A new voltage pattern is selected and applied to the BLDC motor.

As shown below, when using the six-step control technique, it is difficult to keep the angle between the rotor flux and the stator flux precisely at 90° in a six-step control technique. The actual angle varies from 60° to 120°.

The commutation process is repeated per each 60 electrical degrees and is critical to maintain its angular (time) accuracy. Any deviation causes torque ripples, resulting in speed variation.

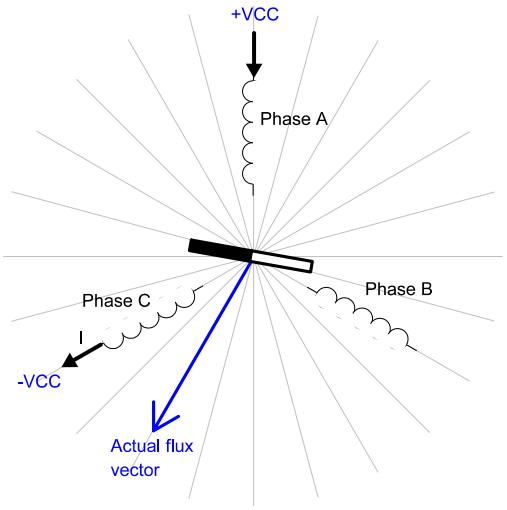

Figure 6. Situation Right Before Commutation (Counter-Clockwise Motion)

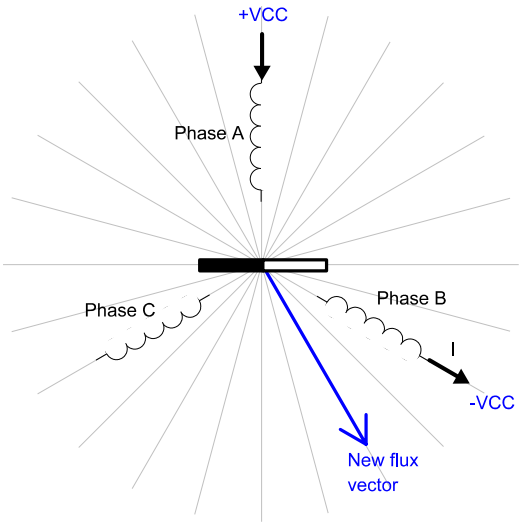

Figure 7. Situation Right After Commutation

2.1.2 Speed Control

Commutation ensures the proper rotor rotation of the BLDC motor, while the motor speed only depends on the amplitude of the applied voltage. The amplitude of the applied voltage is adjusted using the PWM technique. The required speed is controlled by a speed controller, which is implemented as a conventional proportional-integral (PI) controller. The difference between the actual and required speeds is input to the PI controller which then, based on this difference, controls the duty cycle of the PWM pulses which correspond to the voltage amplitude required to maintain the desired speed.

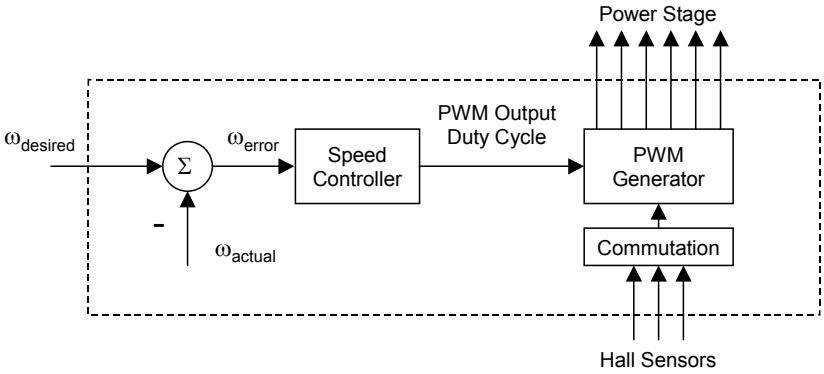

Figure 8. Speed Controller

The speed controller calculates the PI algorithm given in the equation below:

$$u(t) = K_c\left[e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau\right]$$

After transforming the equation into a discrete time domain using an integral approximation with the Backward Euler method, we get the following equations for the numerical PI controller calculation:

$$u(k) = u_P(k) + u_I(k)$$

$$u_P(k) = K_c \cdot e(k)$$

$$u_I(k) = u_I(k-1) + K_c\frac{T}{T_I} \cdot e(k)$$

where:

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

3 System Concept

3.1 System Outline

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

- Voltage control of a BLDC motor using Hall sensors
- Targeted at ColdFire MCF523x evaluation board (M523xEVB), micro power stage for 3 BLDC motors, and three Pittman BLDC motors (N2311)

- Control technique incorporates:
  - Voltage BLDC motor control with speed-closed loop
  - Both directions of rotation
  - 4-quadrant operation
  - Start from any motor position without rotor alignment
  - Minimum speed of 500 RPM
  - Maximum speed of 10000 RPM (limited by power supply)
- FreeMaster control interface (power switch, speeds set-up, indicators)
- FreeMASTER monitor
  - FreeMASTER graphical control page (required speeds, actual motor speeds, start/stop status, fault status)
  - FreeMASTER speed scope (observes required, ramp, and actual speeds)
  - FreeMASTER Hall sensors scope (observes actual Hall sensors states, motor directions, and revolution counters)
- DC bus over-current fault protection

3.2 Application Description

A standard system concept is chosen for the motor control function (see Figure 9). The system incorporates the following hardware:

- Evaluation board M523xEVB
- Micro power stage for 3 BLDC motors
- Three Pittman BLDC motors (N2311) with Hall sensors
- Power supply 12V DC, 5A

The eTPU module runs the main control algorithm. The 3-phase PWM output signals for a 3-phase inverter are generated according to feedback signals from Hall sensors and the input variable values, provided by the microprocessor CPU.

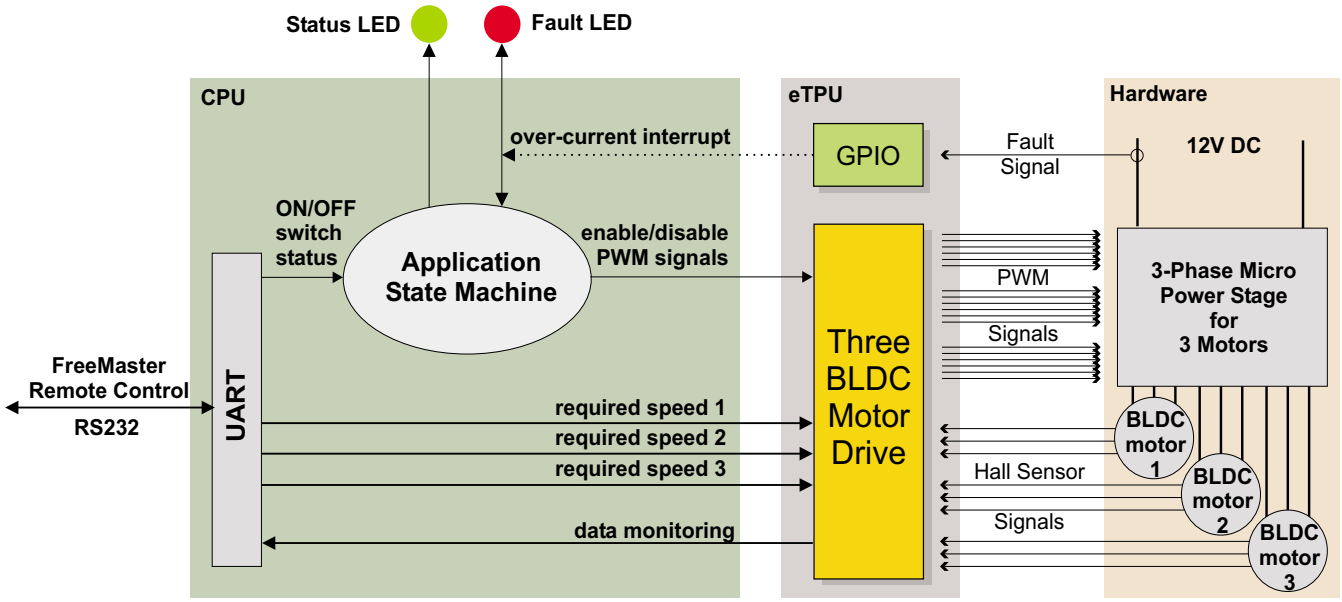

Figure 9. System Concept

The system processing is distributed between the CPU and the eTPU, which both run in parallel.

The CPU performs the following tasks:

- Communicates with the FreeMASTER user interface. Based on the user input, it handles the application state machine and calculates the required speeds that is passed to the eTPU.
- Periodically reads application data from ETPU DATA RAM in order to monitor application variables.
- In the event of an overcurrent fault, the PWM outputs are immediately temporarily disabled by the eTPU hardware. Then, after an interrupt latency, the CPU disables the PWM outputs permanently and displays the fault state.

The eTPU performs the following tasks:

- 3 x 6 eTPU channels (PWMC) are used to generate PWM output signals.
- 3 x 3 eTPU channels (HD) are used to process Hall sensor signals. On each incoming edge, a revolution period is calculated and the PWM output signals are commuted.
- One eTPU channel (GPIO) is used to generate an interrupt to the CPU when the over-current fault signal activates.
- 3 x 1 eTPU channel (PWMMDCSC) is internally used to close the speed loop. The actual motor speed is calculated based on the revolution period and compared with the required speed, provided by the CPU and passed through a ramp. A speed PI control algorithm processes the error between the required and the actual speed. The PI controller output is passed to the PWM generator as a new corrected value of applied motor voltage.

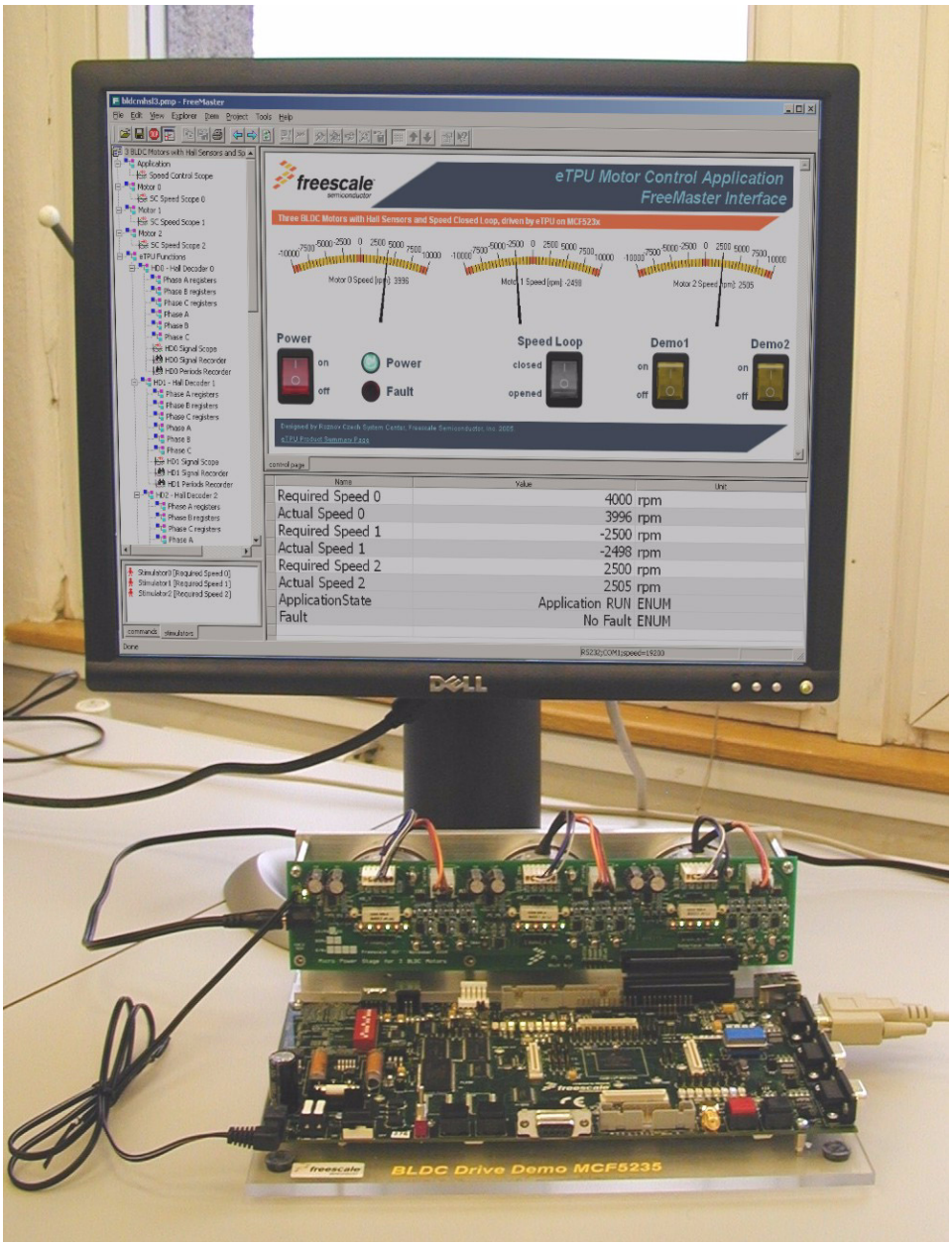

Figure 10. The Application and FreeMASTER Screen

3.2.1 User Interface

FreeMASTER software serves as the application interface. This software was designed to provide an application-debugging, diagnostic, and demonstration tool for the development of algorithms and applications. It runs on a PC connected to the M523xEVB via an RS232 serial cable. A small program resident in the microprocessor communicates with the FreeMASTER software to return status information to the PC and process control information from the PC. FreeMASTER software, executing on a PC, uses part of Microsoft Internet Explorer as the user interface.

The power switch on the FreeMASTER control page affects the application state by enabling and disabling the PWM phases. When the switch is in the off-position, no voltage is applied to the motor windings; it's in the on-position, the motor speed can be controlled by clicking on the gauge, dragging the gauge needle to the requested speed value, or setting the requested values in the watch section at the bottom of the screen. In addition to that, FreeMASTER also displays real-time values of application variables and their time behavior using scopes.

Note, that FreeMASTER version 1.2.31.1 or higher is required. The FreeMASTER application can be downloaded from http://www.freescale.com. For more information about FreeMASTER, refer to Reference 5.

3.3 Hardware Implementation and Application Setup

As previously stated, the application runs on the MCF523x family of ColdFire microprocessors using the following:

- M523xEVB
- Micro power stage for 3 BLDC motors
- Three 3-phase pittman BLDC motor (N2311)
- Power supply, 12V DC, minimum 4.5 A

Figure 11 shows the connection of these parts. All system parts are supplied by Freescale and documented according to references.

3.3.1 ColdFire MCF523x Evaluation Board (M523xEVB)

The EVB is intended to provide a mechanism for customers to easily evaluate the MCF523x family of ColdFire microprocessors. The heart of the evaluation board is the MCF5235; all other M523x family members have a subset of the MCF5235 features and can therefore be fully emulated using the MCF5235 device.

The M523xEVB is fitted with a single 512K x 16 page-mode FLASH memory (U19), giving a total memory space of 2 Mbytes. Alternatively, a footprint is available for upgrading flash to a 512K x 32 page-mode FLASH memory (U35), doubling the memory size to 4 Mbytes.

For more information, refer to Reference 2.

Table 1 lists all M523xEVB jumper settings used in the application.

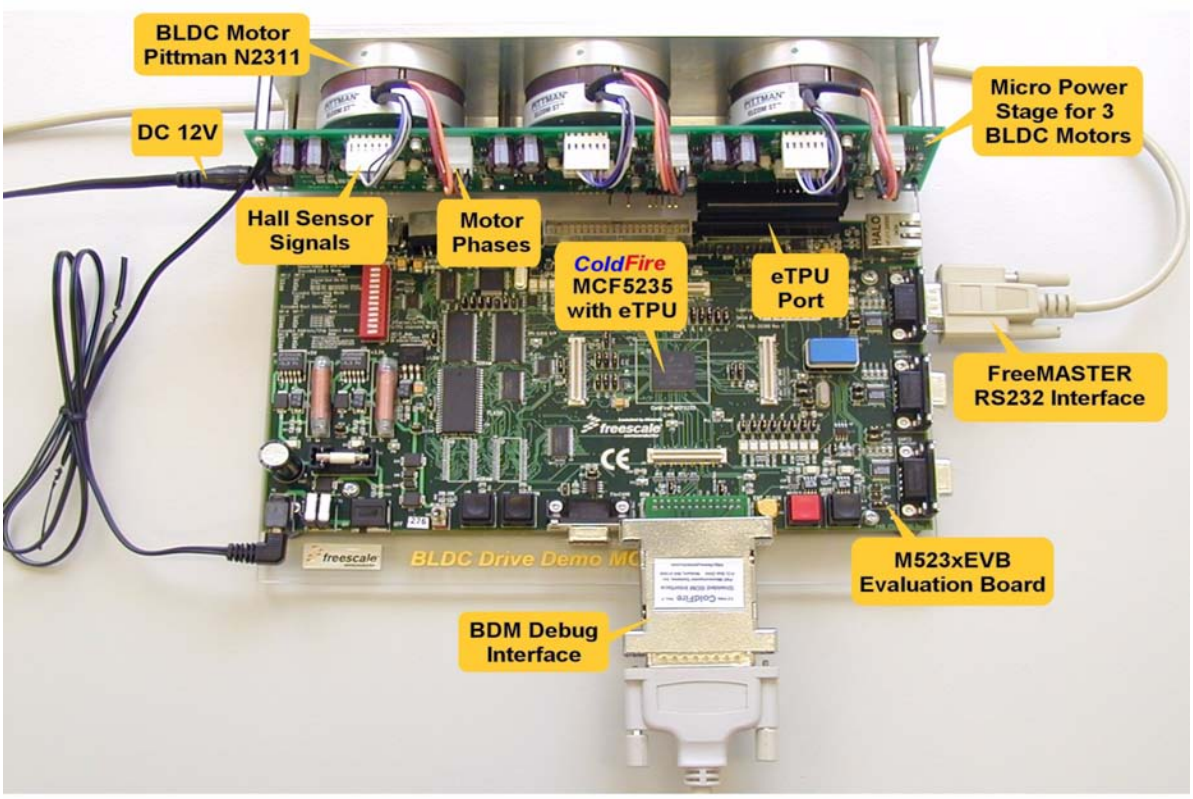

Figure 11. Connection of Application Parts

Table 1. M523xEVB Jumper Settings

Jumper	Setting
JP1	1 2
JP2	1-2
JP3	1 2
JP4	1-2
JP5	1-2 3
JP6	1-2 3
JP7	1 2-3
JP8	1-2 3
JP9	1-2 3
JP10	1-2 3
JP11	1-2 3
JP12	1 2-3
JP13	1-2 3
JP14	1-2 3
JP15	1-2 3
JP16	1-2 3
JP17	1-2 3
JP18	1-2 3
JP19	1-2 3

Jumper	Setting
JP20	1-2 3
JP21	1-2 3
JP22	1-2 3
JP23	1-2 3
JP24	1-2 3
JP25	1-2 3
JP26	1-2 3
JP27	1-2
JP28	1-2
JP29	1-2
JP30	1-2
JP31	1 2-3
JP32	1-2 3
JP33	1-2
JP34	1-2
JP35	1-2 3
JP36	1-2 3
JP37	1-2
JP38	1-2
JP39	1-2

Jumper	Setting
JP40	1-2
JP41	1-2
JP42	1-2
JP43	1-2
JP44	1-2
JP45	1-2
JP46	1-2
JP47	1-2
JP48	1-2
JP49	1-2
JP50	1-2 3
JP51	1-2 3
JP52	1-2 3
JP53	1 2
JP54	1 2
JP55	1 2
JP56	1-2 3
JP57	1-2
JP58	1-2
JP59	1-2

Jumper	Setting
JP60	1-2
JP61	1-2
JP62	1 2
JP63	1 2
JP64	1 2-3
DIP1	ON
DIP2	ON
DIP3	ON
DIP4	ON
DIP5	ON
DIP6	ON
DIP7	OFF
DIP8	ON
DIP9	ON
DIP10	ON
DIP11	OFF
DIP12	ON

3.3.2 Flashing the M523xEVB

The CFFlasher utility can be used for programming code into the FLASH memory on the MCF523xEVB. Check for correct setting of switches and jumpers: SW7-6 on, SW7-7 off, JP64 2-3, (JP31 2-3). The flashing procedure is as follows:

1. Run Metrowerks CodeWarrior for ColdFire and open the project. Choose the simple_eflash target and compile the application. A file simple_eflash.elf.S19, which will be loaded into FLASH memory, is created in the project directory bin.
2. Run the CFFlasher application, click on the "Target Config" button. In the Target Configuration window select the type of board as M523xEVB and the BDM Communication as PE_LPT (see Figure 12). Click OK to close the window.
3. Go to the Program section by clicking the "Program" button. Select the simple_eflash.elf.S19 file and check the "Verify after program" option (see Figure 13). Finally, press the "Program" button at the bottom of the window to start loading the code into the FLASH memory.
4. If the code has been programmed correctly, remove the BDM interface and push the RESET button on the M523xEVB. The application should now run from the FLASH.

The CFFlasher application can be downloaded from http://www.freescale.com/coldfire.

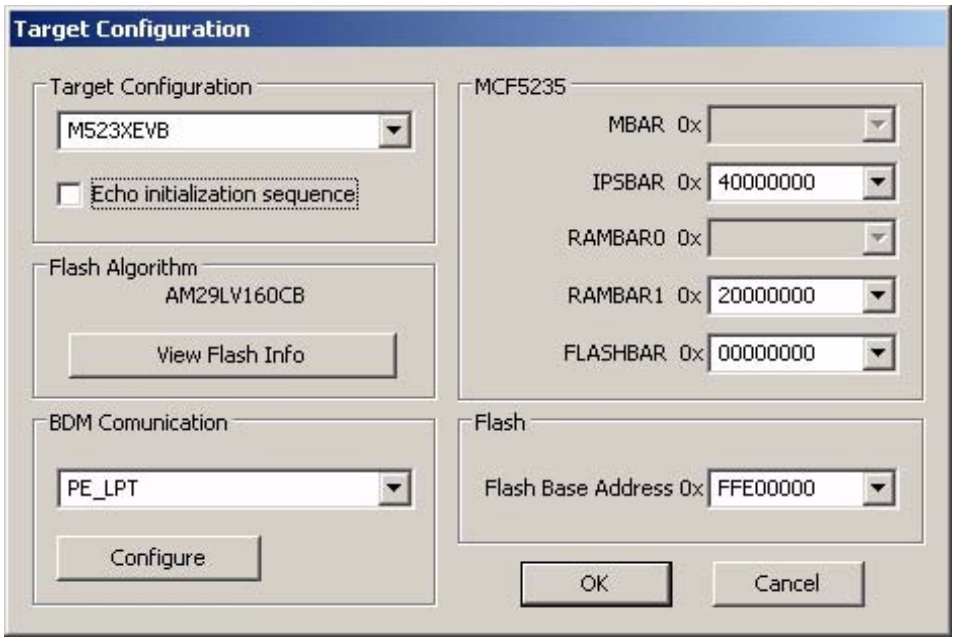

Figure 12. CFFlasher Target Configuration Window

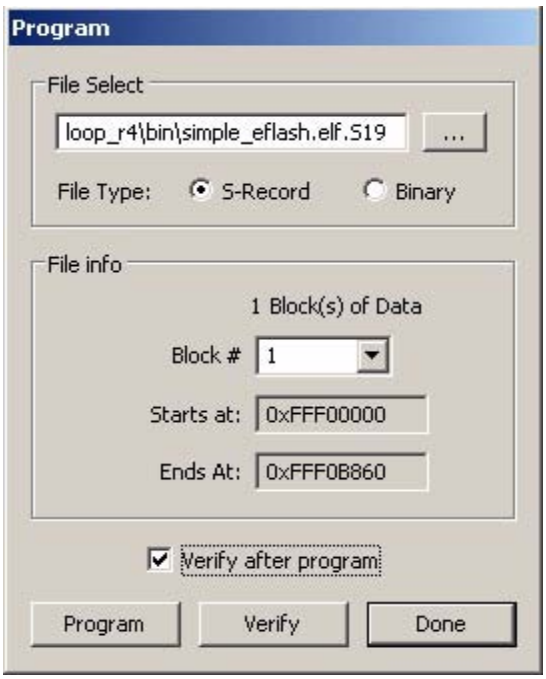

Figure 13. CFFlasher Program Window

3.3.3 Setting Overcurrent Level

The over-current fault signal is connected to the eTPU output disable pins (LTPUODIS, UTPUODIS) that handle the eTPU hardware faults, along with a proper eTPU configuration. This connection is part of the micro power stage for 3 BLDC motors. In order to enable handling of the fault by software, the fault signal, available on the LTPUODIS and the UTPUODIS pins, are connected to eTPU channel 9 which runs the GPIO function and generates interrupt request to the CPU in the case of a fault. The over-current level is set by trimmer R2 on micro power stage for 3 BLDC motors (see Figure 14).

3.3.4 Micro Power Stage for 3 BLDC Motors

The micro power stage for the 3 BLDC motors is a 12V/5A surface-mount power stage, which is supplied with a 40-pin ribbon cable. In combination with the M523xEVB, it provides an out-of-the-box software development platform for small brushless DC motors.

The board is equipped with the over-current protection. The over-current level is set by the trimmer R2 that is located in the middle of the power stage. A fault is detected when the red LED diode (D2) is lit.

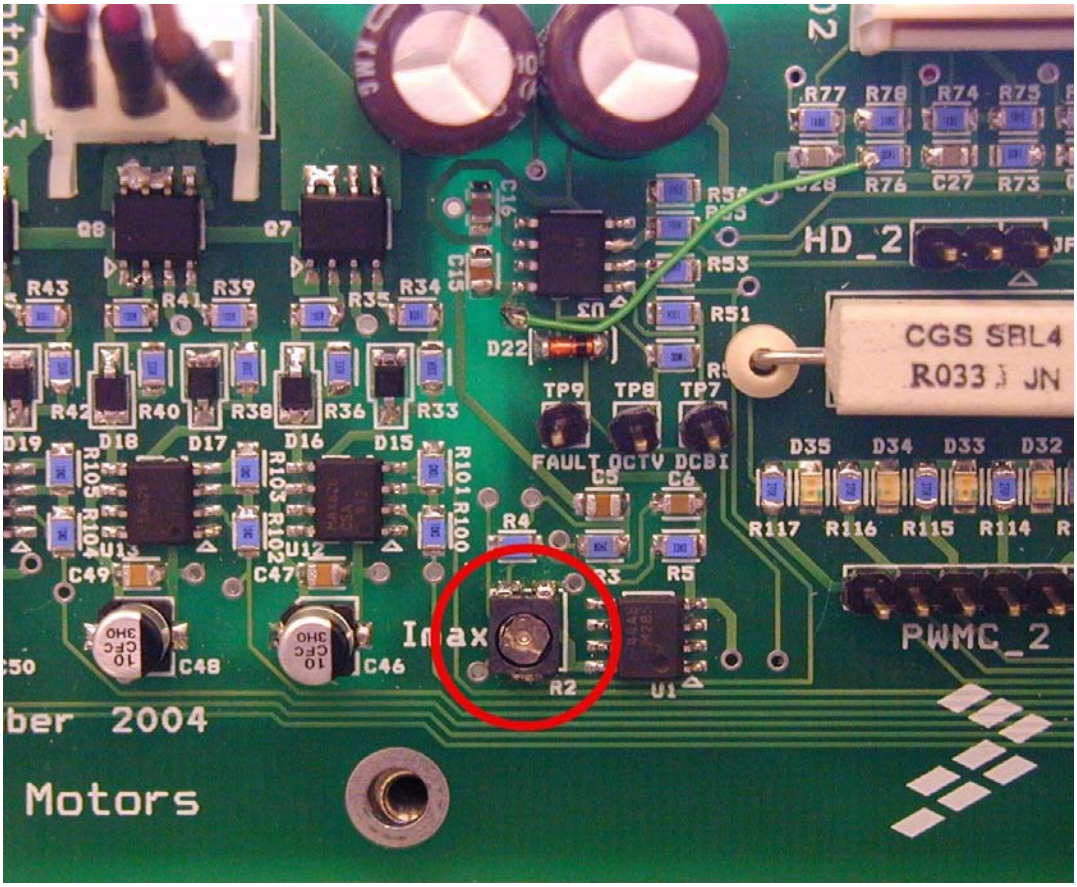

Figure 14. Overcurrent Trimmer R2 on Micro Power Stage for 3 BLDC Motors

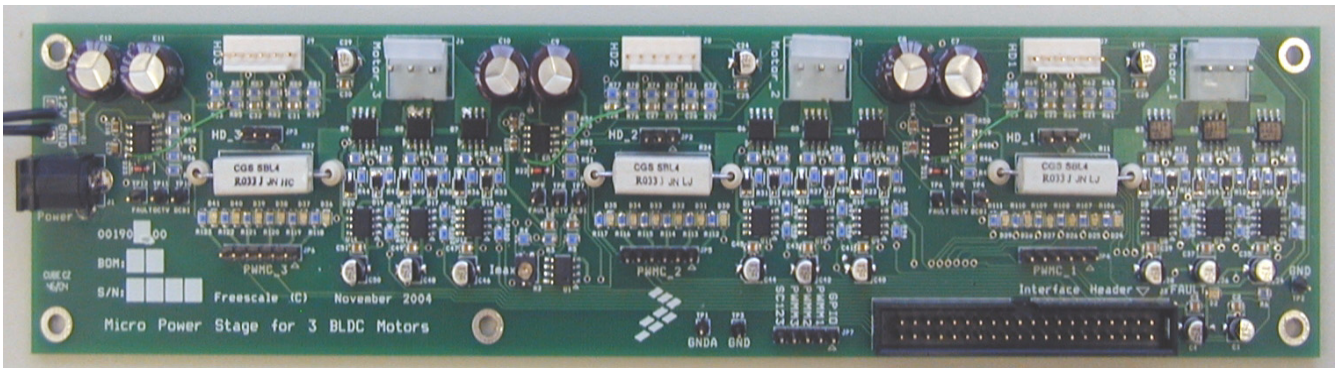

Figure 15. Micro Power Stage for 3 BLDC Motors

For more information, refer to Reference 3.

3.3.5 BLDC Motor with Hall Sensors

The enclosed motors are a low-voltage Pittman BLDC motors (N2311). The motors is capable of being controlled by Hall sensor techniques. The motor characteristics in Table 2 apply to operation at 25°C.

Figure 25 depicts the motor timing. For more motor specifications, refer to Reference 4.

Table 2. Pittman BLDC Motor (N2311) Motor Characteristics

Characteristic	Symbol	Min	Type	Max	Units
Reference Winding Voltage	V_t	—	—	9.6	V
Speed @ V_t		—	—	12000	RPM
Torque Constant	K_t	—	0.007	—	Nm/A
	K_t	—	1.082	—	oz-in/A
Voltage Constant	K_e	—	0.8	—	V/kRPM
Terminal Resistance	R_t	0.13	—	0.18	W
Winding Inductance	L	—		—	mH
Continuous Current	I_{cs}	—	—	9.96	A
No Load Current @ V_t	I_{ps}	—	1.20	—	A
Number of Poles	J_m	—	8	—	—
Temperature Rating		-10	—	80	
		14	—	176	

3.3.6 Power Supply

The power supply (12V / 5A, 60W) is used to power the micro power stage for 3 BLDC motors. The MCF523xEVB is powered by the micro power stage through the 2.1mm power connector. It is also possible to power the MCF523xEVB with an independent power supply (9-12V, 1.8A). The application is scaled for this 12V power supply. If a different power supply is used, the application should be rescaled for the different voltage and speed range.

4 Software Design

This section describes the software design of the BLDC motor drive application. The system processing is distributed between the CPU and the eTPU, which run in parallel. The CPU and eTPU tasks are described in terms of the following:

- CPU
  - Software flowchart
  - Application state diagram
  - eTPU application API
- eTPU
  - eTPU block diagram
  - eTPU timing

The CPU software uses several ready-to-use Freescale software drivers. The communication between the microprocessor and the FreeMASTER on a PC is handled by software included in `fmaster.c/.h` files. The eTPU module uses the general eTPU utilities, eTPU function interface routines (eTPU function API), and eTPU application interface routines (eTPU application API). The general utilities, included in the etpu_util.c/.h files, are used for initialization of global eTPU module and engine settings. The eTPU function API routines are used for initialization of the eTPU channels and interfacing each eTPU function during run-time. An eTPU application API encapsulates several eTPU function APIs. The use of an eTPU application API eliminates the need to initialize each eTPU function separately and to handle all eTPU function initialization settings, and so ensures the correct cooperation of eTPU functions.

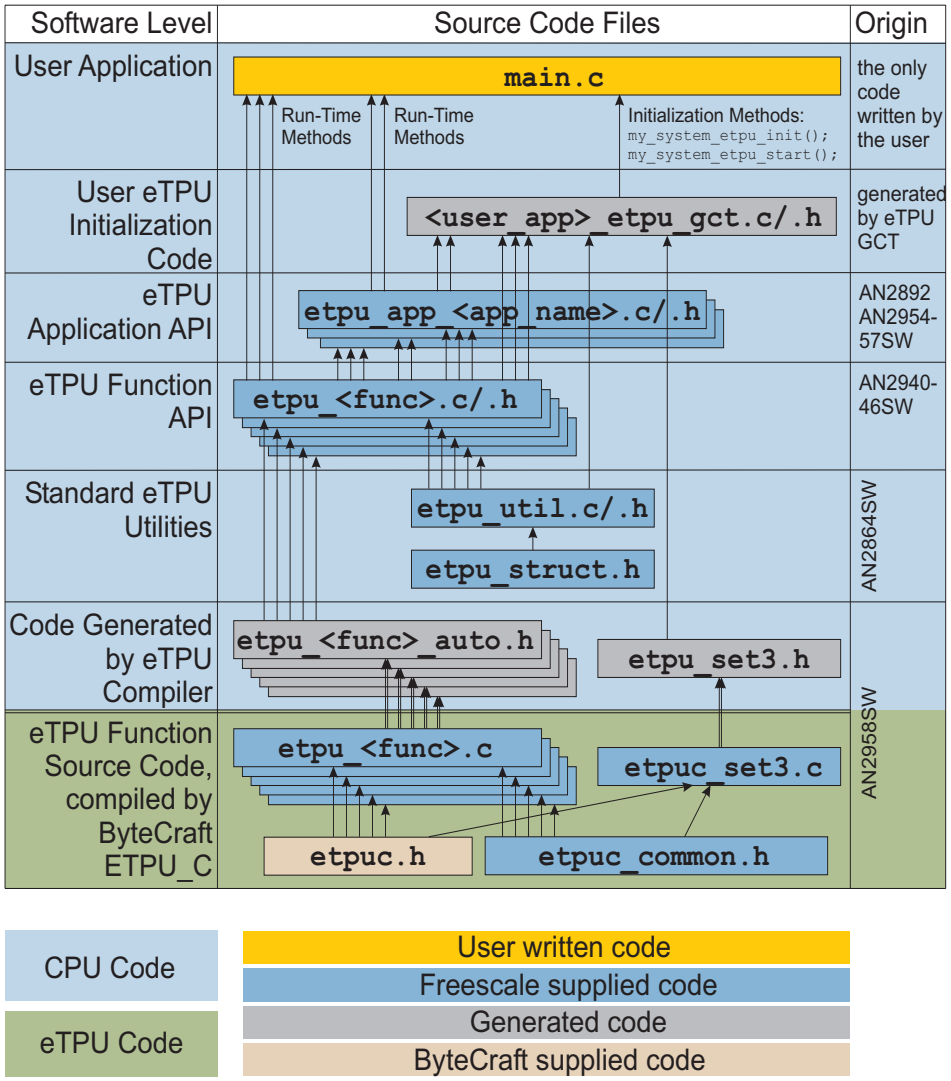

Figure 16. eTPU Project Structure

4.1 CPU Software Flowchart

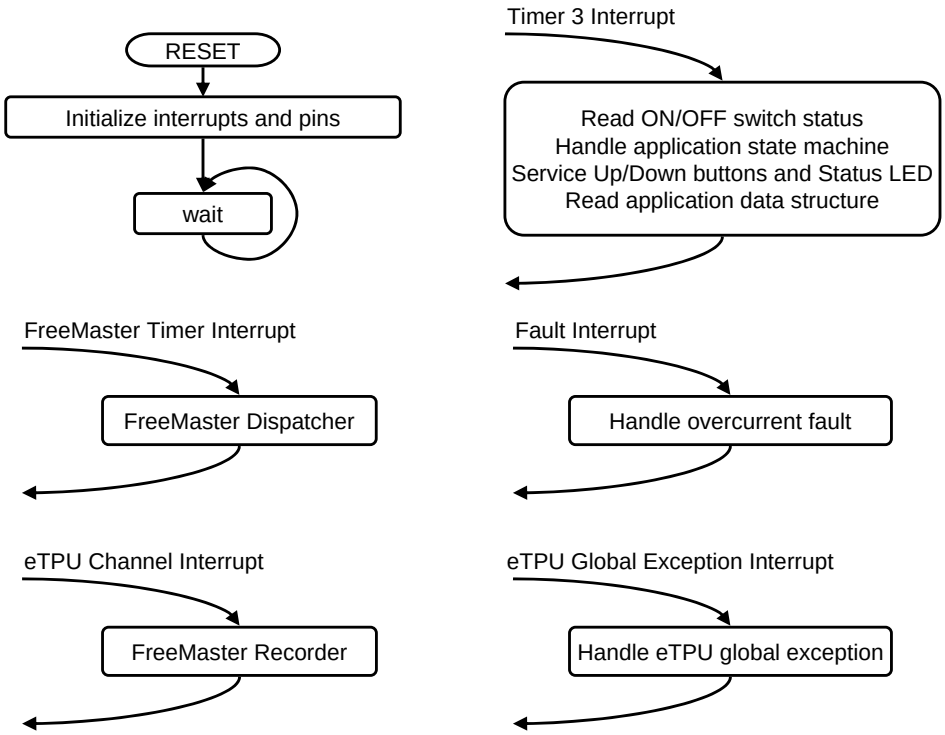

Figure 17. CPU Software Flowchart

After reset, the CPU software initializes peripherals, interrupts, and pins, and starts the eTPU module. At this point, the CPU and the eTPU run in parallel. The following CPU processing is incorporated in two periodic timer interrupts and one fault interrupt.

4.1.1 Initialization of Interrupts and Pins

The initialization of Timer 3, eTPU channel 9 and 28 interrupts, and the eTPU global exception interrupt, together with initialization of the GPIO, LTPUODIS, and UTPUODIS pins, is done by the InitInterruptsAndPins function.

4.1.2 Timer 3 Interrupt Service Routine

The timer 3 interrupt is handled by the timer3_isr function. The following actions are performed periodically, in timer3_isr:

- Handle the application state machine

  The application state diagram is described in detail below.
- Read the data structure through the eTPU application API routine `fs_etpu_app_bldcmhsl3_get_data` (see 4.3).

4.1.3 FreeMASTER Interrupt Service Routine

The FreeMASTER interrupt service routine is called fmasterDispatcher. This function is implemented in fmaster.c.

4.1.4 eTPU Channel Interrupt Service Routine

This interrupt, which is raised every PWM period by the PWMMDCSC0 eTPU function running on eTPU channel 28, is handled by the etpu_ch28_isr function. This function calls fmasterRecorder, implemented in fmaster.c, enabling the configuration of application variable time courses with a PWM-period time resolution.

4.1.5 Fault Interrupt Service Routine

The over-current fault interrupt, which is raised by the GPIO eTPU function running on eTPU channel 9, is handled by the etpu_ch9_isr function. The following actions are performed in order to switch the motor off:

- Reset the required speeds
- Disable the generation of PWM signals
- Switch the Fault LED on
- Enter APP_STATE_MOTOR_FAULT
- Set FAULT_OVERCURRENT

4.1.6 eTPU Global Exception Interrupt Service Routine

The global exception interrupt is handled by the etpu_globalexception_isr function. The following situations can cause this interrupt assertion:

- Microcode global exception is asserted
- Illegal instruction flag is asserted
- SCM MISC flag is asserted

The following actions are performed in order to switch the motor off:

- Reset the required speeds
- Disable the generation of PWM signals
- Enter APP_STATE_GLOBAL_FAULT
- Based on the eTPU global exception source, set FAULT_MICROCODE_GE, FAULT_ILLEGAL_INSTR, or FAULT_MISC.

4.2 Application State Diagram

The application state diagram consists of seven states (see Figure 18). After reset, the application goes firstly to the APP_STATE_INIT and then to the APP_STATE_STOP. Then the cycle between

APP_STATE_STOP, APP_STATE_ENABLE, APP_STATE_RUN, and APP_STATE_DISABLE can be repeated, depending on the power switch position on the FreeMASTER control page. APP_STATE_ENABLE and APP_STATE_DISABLE states are introduced in order to ensure the safe transitions between the APP_STATE_STOP and APP_STATE_RUN states. Where the over-current fault interrupt is raised (see red line on Figure 18), the APP_STATE_MOTOR_FAULT is entered. This fault is cleared by moving the power switch to the OFF position and thus entering the APP_STATE_STOP. Where the eTPU global exception interrupt is raised (see gray line on Figure 18), the APP_STATE_GLOBAL_FAULT is entered. The global fault is cleared by moving the power switch to the OFF position and thus entering the APP_STATE_INIT.

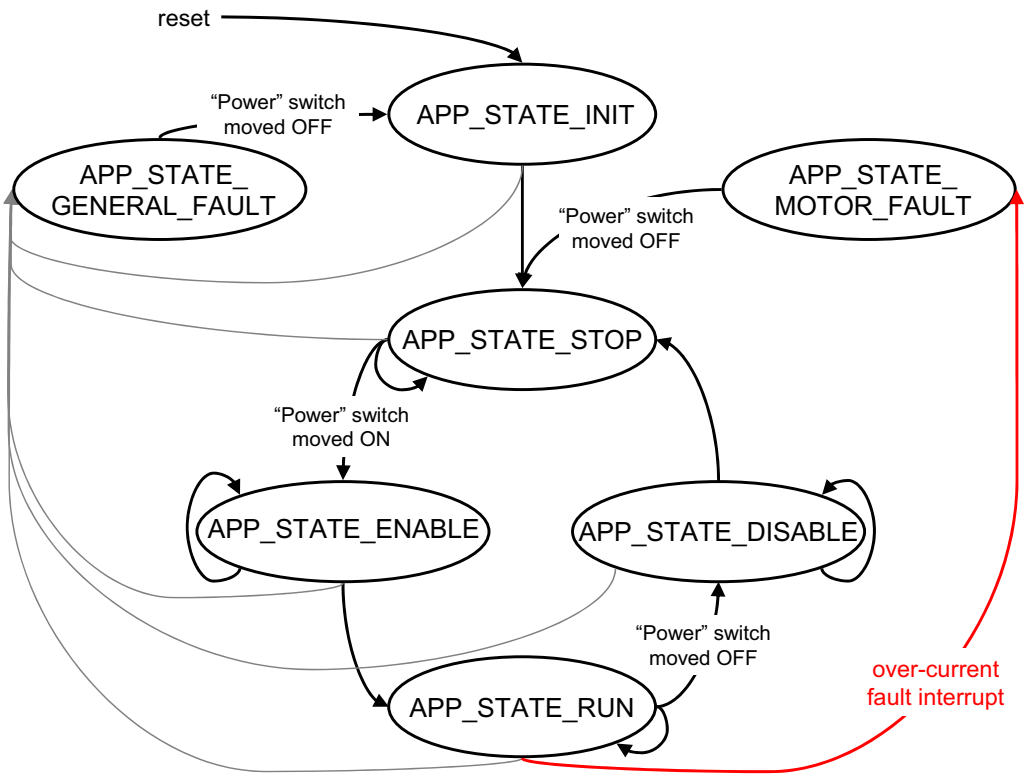

Figure 18. Application State Diagram

The following paragraphs describe the processing in each of the application states.

4.2.1 APP_STATE_INIT

This state is passed through only. It is entered either after a reset, or after the APP_STATE_GLOBAL_FAULT. The following actions are performed in order to initialize (re-initialize) the application:

- Call `my_system_etpu_init` routine for eTPU module initialization
- Get eTPU functions DATA RAM addresses for FreeMASTER
- Get the addresses of channel configuration registers for FreeMASTER
- Initialize the UART for FreeMASTER

- Initialize FreeMASTER
- Call `my_system_etpu_start` routine for eTPU start. At this point, the CPU and the eTPU run in parallel.
- Depending on the power switch position, enter APP_STATE_STOP or APP_STATE_MOTOR_FAULT

4.2.1.1 Initialization and Start of eTPU Module

The eTPU module is initialized using the `my_system_etpu_init` function. Later, after initialization of all other peripherals, the eTPU is started by `my_system_etpu_start`. These functions use the general eTPU utilities and eTPU function API routines. Both the `my_system_etpu_init` and `my_system_etpu_start` functions, included in `bldcmhsl3_etpu_gct.c` file, are generated by the eTPU graphical configuration tool. The eTPU graphical configuration tool can be downloaded from http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=eTPU. For more information, refer to Reference 12.

The `my_system_etpu_init` function first configures the eTPU module and motor settings. Some of these settings include the following:

- Channel filter mode = three-sample mode
- Channel filter clock = etpuclk div 32

  The input signals (from Hall sensors) are filtered by channel filters. The filter settings guarantee filtering all noise pulses up to a width of 853ns and pass pulses from a width of 1.28µs (at 150 MHz system clock).
- TCR1 source = etpuclk div 2
- TCR1 prescaler = 1

  The TCR1 internal eTPU clock is set to its maximum rate of 37.5 MHz (at 150 MHz system clock), corresponding to the 27ns resolution of generated PWM signals.
- TCR2 source = etpuclk div 8
- TCR2 prescaler = 12

  The TCR2 internal eTPU clock is set to a rate of 781.25 kHz (at 150 MHz system clock). The TCR2 clock settings are optimized for motor speed calculation precision.

After configuring the module and engine settings, the `my_system_etpu_init` function initializes the eTPU channels.

Channel 0 - Hall decoder (HD) of the first motor - phase A
Channel 1 - Hall decoder (HD) of the first motor - phase B
Channel 2 - Hall decoder (HD) of the first motor - phase C
Channel 3 - Hall decoder (HD) of the second motor - phase A
Channel 4 - Hall decoder (HD) of the second motor - phase B
Channel 5 - Hall decoder (HD) of the second motor - phase C
Channel 6 - Hall decoder (HD) of the third motor - phase A
Channel 7 - Hall decoder (HD) of the third motor - phase B

Channel 8 - Hall decoder (HD) of the third motor - phase C
Channel 10 - PWM commuted (PWMC) - phase A of the first motor - base channel
Channel 11 - PWM commuted (PWMC) - phase A of the first motor - complementary channel
Channel 12 - PWM commuted (PWMC) - phase B of the first motor - base channel
Channel 13 - PWM commuted (PWMC) - phase B of the first motor - complementary channel
Channel 14 - PWM commuted (PWMC) - phase C of the first motor - base channel
Channel 15 - PWM commuted (PWMC) - phase C of the first motor - complementary channel
Channel 16 - PWM commuted (PWMC) - phase A of the second motor - base channel
Channel 17 - PWM commuted (PWMC) - phase A of the second motor - complementary channel
Channel 18 - PWM commuted (PWMC) - phase B of the second motor - base channel
Channel 19 - PWM commuted (PWMC) - phase B of the second motor - complementary channel
Channel 20 - PWM commuted (PWMC) - phase C of the second motor - base channel
Channel 21 - PWM commuted (PWMC) - phase C of the second motor - complementary channel
Channel 22 - PWM commuted (PWMC) - phase A of the third motor - base channel
Channel 23 - PWM commuted (PWMC) - phase A of the third motor - complementary channel
Channel 24 - PWM commuted (PWMC) - phase B of the third motor - base channel
Channel 25 - PWM commuted (PWMC) - phase B of the third motor - complementary channel
Channel 26 - PWM commuted (PWMC) - phase C of the third motor - base channel
Channel 27 - PWM commuted (PWMC) - phase C of the third motor - complementary channel
Channel 28 - PWM master for DC motors with speed controller (PWMMDCSC) of the first motor
Channel 29 - PWM master for DC motors with speed controller (PWMMDCSC) of the second motor
Channel 30 - PWM master for DC motors with speed controller (PWMMDCSC) of the third motor

These eTPU channels are initialized by the `fs_etpu_app_bldcmhsl3_init` eTPU application API function (see 4.3). This function is called three-times to initialize all three motors. The application settings are as follows:

— PWM phases-type is commuted complementary pairs
— PWM frequency 20kHz
— PWM dead-time 1µs
— Motor speed range 14 000 RPM
— Motor speed minimum 300 RPM
— DC-bus voltage 12V
— Number of motor pole pairs 4
— Motor speed calculated using HD revolution period
— Speed controller update frequency 10kHz
— PI controller parameters:
P-gain is 0.5 ($0x004000 * 2^{-15}$), and
I-gain is 0.0078125 ($0x000100 * 2^{-15}$).

The controller parameters were experimentally tuned.

— Ramp parameters:
4s to ramp up from zero to the maximum speed,
4s to ramp down from the maximum speed to zero.

- Channel 4 - general purpose I/O (GPIO)

  This eTPU channel is initialized by the `fs_etpu_gpio_init` API function. The setting is:

  — Channel priority: high

The `my_system_etpu_start` function first applies the settings for the channel interrupt enable and channel output disable options, then enables the eTPU timers, thus starting the eTPU.

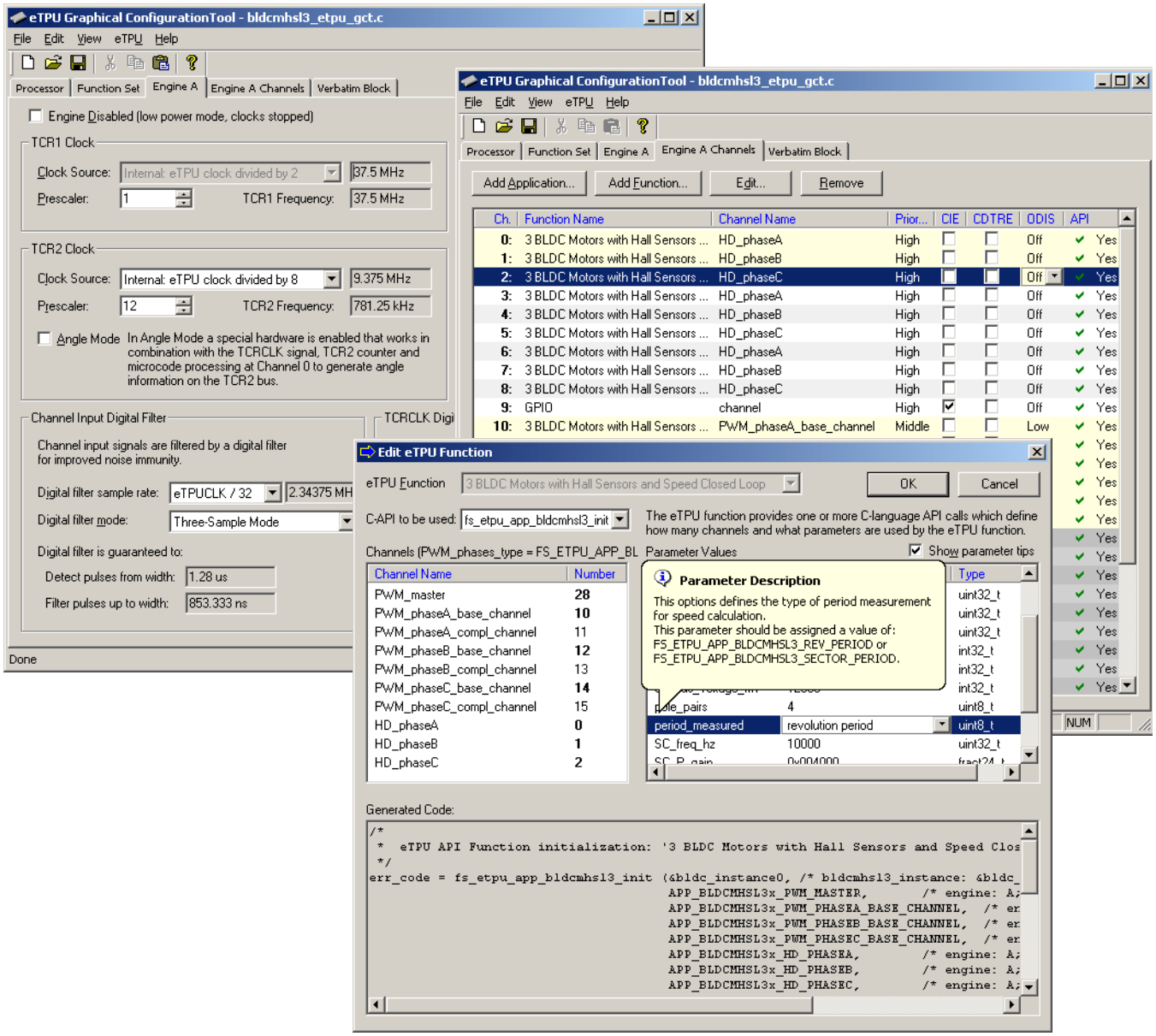

Figure 19. eTPU Configuration Using the eTPU Graphical Configuration Tool

4.2.1.2 Initialization of FreeMASTER Communication

Prior to the FreeMASTER initialization, it is necessary to set pointers to the eTPU functions DATA RAM bases and configuration register bases. Based on these pointers, which are read by FreeMASTER during the initialization, the locations of all eTPU function parameters and configuration registers are defined. This is essential for correct FreeMASTER operation!

FreeMASTER consists of software running on a PC and on the microprocessor, connected via an RS-232 serial port. A small program resident in the microprocessor communicates with the FreeMASTER on the PC in order to return status information to the PC, and processes control information from the PC. The microprocessor part of the FreeMASTER is initialized by two functions: iniFmasterUart and fmasterInit. Both functions are included in fmaster.c, which automatically initializes the UART driver and installs all necessary services.

4.2.2 APP_STATE_STOP

In this state, the PWM signals are disabled and the motors are off. The motor shafts can be rotated by hand, which enables the user to explore the functionality of the Hall decoder (HD) eTPU function, to watch variables produced by the HD, and to see Hall sensor signals in FreeMASTER.

When the power switch is turned on, the application goes through APP_STATE_ENABLE to APP_STATE_RUN.

4.2.3 APP_STATE_ENABLE

This state is passed through only. The following actions are performed in order to switch the motor drive on:

- Reset the required speeds
- Enable the generation of PWM signals

If the PWM phases were successfully enabled, the GPIO eTPU function is configured as input, interrupt on rising edge, and APP_STATE_RUN is entered. Where the PWM phases were not successfully enabled, the application state does not change.

4.2.4 APP_STATE_RUN

In this state, the PWM signals are enabled and the motors are on. The required motor speeds can be set using FreeMASTER. The latest value is periodically written to the eTPU.

When the power switch is turned off, the application goes through APP_STATE_DISABLE to APP_STATE_STOP.

4.2.5 APP_STATE_DISABLE

This state is passed through only. The following actions are performed in order to switch the motor drive off:

- Reset the required speeds
- Disable the generation of PWM signals

If PWM phases were successfully disabled, APP_STATE_STOP is entered. Where PWM phases were not successfully disabled, the application state remains the same.

4.2.6 APP_STATE_MOTOR_FAULT

This state is entered after the over-current fault interrupt service routine. The application waits until the power switch is turned off. This clears the fault and the application enters the APP_STATE_STOP.

4.2.7 APP_STATE_GLOBAL_FAULT

This state is entered after the eTPU global exception interrupt service routine. The application waits until the power switch is turned off. This clears the fault and the application enters the APP_STATE_INIT.

4.3 eTPU Application API

The eTPU application API encapsulates several eTPU function APIs. The eTPU application API includes CPU methods which enable initialization, control, and monitoring of an eTPU application. The use of eTPU application API functions eliminates the need to initialize and set each eTPU function separately, and ensures correct cooperation of the eTPU functions. The eTPU application API is device independent and handles only the eTPU tasks.

In order to shorten the eTPU application names, abbreviated application names are introduced. The abbreviations include:

- Motor type (DCM = DC motor, BLDCM = brushless DC motor, PMSM = permanent magnet synchronous motor, ACIM = AC induction motor, SRM = switched reluctance motor, SM = stepper motor)
- Sensor type (H = Hall sensors, E = shaft encoder, R = resolver, S = sincos, X = sensorless)
- Control type (OL = open loop, PL = position loop, SL = speed loop, CL = current loop, SVC = speed vector control, TVC = torque vector control)

Based on these definitions, the BLDCMHSL3 is an abbreviation for “BLDC motor with Hall sensors and speed closed loop” eTPU motor-control application. As there are several BLDC motor applications with hall sensors and speed closed loop, the number 3 denotes the third such application in order. This third application, demonstrates the eTPU usage to drive three BLDC motors.

The BLDCMHSL3 eTPU application API is described in the following paragraphs. There are 5 basic functions added to the BLDCMHSL3 application API. The routines can be found in the `etpu_app_bldcmhsl3.c/.h` files. All BLDCMHSL3 application API routines will be described in order and are listed below:

- Initialization function:

```
int32_t fs_etpu_app_bldcmhsl3_init(
bldcmhsl3_instance_t * bldcmhsl3_instance,
             uint8_t   PWM_master_channel,
             uint8_t   PWM_phaseA_channel,
             uint8_t   PWM_phaseB_channel,
             uint8_t   PWM_phaseC_channel,
             uint8_t   HD_phaseA_channel,
             uint8_t   HD_phaseB_channel,
             uint8_t   HD_phaseC_channel,
             uint8_t   PWM_phases_type,
             uint32_t  PWM_freq_hz,
             uint32_t  PWM_dead_time_ns,
             uint32_t  PWM_min_pw_ns,
             uint32_t  start_offset,
             int32_t   speed_range_rpm,
             int32_t   speed_min_rpm,
             int32_t   dc_bus_voltage_mv,
             uint8_t   pole_pairs,
             uint8_t   period_measured,
             uint32_t  SC_freq_hz,
             fract24_t SC_P_gain,
             fract24_t SC_I_gain,
             uint32_t  SC_ramp_time_ms)
```

- Change operation functions:

```
int32_t fs_etpu_app_bldcmhsl3_enable(
   bldcmhsl3_instance_t * bldcmhsl3_instance,
                uint8_t   configuration)

int32_t fs_etpu_app_bldcmhsl3_disable(
   bldcmhsl3_instance_t * bldcmhsl3_instance)

void fs_etpu_app_bldcmhsl3_set_speed_required(
   bldcmhsl3_instance_t * bldcmhsl3_instance,
                int32_t   speed_required_rpm)
```

- Value return functions:

```
void fs_etpu_app_bldcmhsl3_get_data(
   bldcmhsl3_instance_t * bldcmhsl3_instance,
       bldcmhsl3_data_t * bldcmhsl3_data)
```

4.3.1 int32_t fs_etpu_app_bldcmhsl3_init(...)

This routine is used to initialize the eTPU channels for the “3 BLDC motors with Hall decoder and speed closed loop” application. This function has the following parameters:

- **bldcmhsl3_instance (bldcmhsl3_instance_t*)** - This is a pointer to bldcmhsl3_instance_t structure, which is filled by `fs_etpu_app_bldcmhsl3_init`. This structure must be declared in the user application. Where there are more instances of the application running simultaneously, there must be a separate bldcmhsl3_instance_t structure for each one.
- **PWM_master_channel (uint8_t)** - This is the PWM master channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **PWM_phaseA_channel (uint8_t)** - This is the PWM phase A channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B. In the case of complementary signal generation (PWM_phases_type==FS_ETPU_APP_BLDCMHSL3_COMPL_PAIRS), the complementary channel is one channel higher.
- **PWM_phaseB_channel (uint8_t)** - This is the PWM phase B channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B. In the case of complementary signal generation (PWM_phases_type==FS_ETPU_APP_BLDCMHSL3_COMPL_PAIRS), the complementary channel is one channel higher.
- **PWM_phaseC_channel (uint8_t)** - This is the PWM phase C channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B. In the case of complementary signal generation (PWM_phases_type==FS_ETPU_APP_BLDCMHSL3_COMPL_PAIRS), the complementary channel is one channel higher.
- **HD_phaseA_channel (uint8_t)** - This is the Hall decoder phase A channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **HD_phaseB_channel (uint8_t)** - This is the Hall decoder phase B channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **HD_phaseC_channel (uint8_t)** - This is the Hall decoder phase C channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **PWM_phases_type (uint8_t)** - This parameter determines the type of all PWM phases. This parameter should be assigned a value of:
  FS_ETPU_APP_BLDCMHSL3_SINGLE_CHANNELS, or
  FS_ETPU_APP_BLDCMHSL3_COMPL_PAIRS.
- **PWM_freq_hz (uint32_t)** - This is the PWM frequency in Hz.
- **PWM_dead_time_ns (uint32_t)** - This is the PWM dead-time in ns.
- **PWM_min_pw_ns (uint32_t)** - This is the PWM minimum pulse width in ns.
- **start_offset (uint32_t)** - This parameter is used to synchronize several PWMMDCSC functions running on one eTPU engine. The first PWMMDCSC update starts the start_offset TCR1 clocks after initialization.
- **speed_range_rpm (int32_t)** - This is the maximum motor speed in rpm.
- **speed_min_rpm (int32_t)** - This is the minimum (measurable) motor speed in rpm.
- **dc_bus_voltage_mv (int32_t)** - This is the DC-bus voltage in mV.

- **pole_pairs (uint8_t)** - This is the number of motor pole-pairs.
- **period_measured (uint8_t)** - This option defines the type of period measurement for speed calculation. This parameter should be assigned a value of:
  FS_ETPU_APP_BLDCMHSL3_REV_PERIOD, or
  FS_ETPU_APP_BLDCMHSL3_SECTOR_PERIOD.
- **SC_freq_hz (uint32_t)** - This is the speed controller update frequency in Hz. The assigned value must be equal to the PWM_freq_hz divided by 1, 2, 3, 4, 5, ...
- **SC_P_gain (fract24_t)** - This is the speed controller P-gain in 24-bit signed fractional format (9.15).
  0x008000 corresponds to 1.0
  0x000001 corresponds to 0.0000305 ($30.5*10^{-6}$)
  0x7FFFFF corresponds to 255.9999695
- **SC_I_gain (fract24_t)** - This is the speed controller I-gain in 24-bit signed fractional format (9.15).
  0x008000 corresponds to 1.0
  0x000001 corresponds to 0.0000305 ($30.5*10^{-6}$)
  0x7FFFFF corresponds to 255.9999695
- **SC_ramp_time_ms (uint32_t)** - This parameter defines the required speed ramp time in ms. A step change of the required speed from 0 to speed_range_rpm is slowed down by the ramp to take the defined time.

4.3.2 int32_t fs_etpu_app_bldcmhsl3_enable(...)

This routine is used to enable the generation of PWM signals, commutations on Hall signal transitions, and to start the speed controller. This function has the following parameters:

- **bldcmhsl3_instance (bldcmhsl3_instance_t*)** - This is a pointer to bldcmhsl3_instance_t structure, which is filled by `fs_etpu_app_bldcmhsl3_init`.
- **configuration (uint8_t)** - This is the required configuration of the SC. This parameter should be assigned a value of:
  FS_ETPU_APP_BLDCMHSL3_SPEED_LOOP_OPENED, or
  FS_ETPU_APP_BLDCMHSL3_SPEED_LOOP_CLOSED.

4.3.3 int32_t fs_etpu_app_bldcmhsl3_disable (bldcmhsl3_instance_t * bldcmhsl3_instance)

This routine is used to disable the generation of PWM signals, commutation on Hall signal transitions, and to stop the speed controller. This function has the following parameter:

- **bldcmhsl3_instance (bldcmhsl3_instance_t*)** - This is a pointer to bldcmhsl3_instance_t structure, which is filled by `fs_etpu_app_bldcmhsl3_init`.

4.3.4 void fs_etpu_app_bldcmhsl3_set_speed_required(...)

This routine is used to set the required motor speed. This function has the following parameters:

- **bldcmhsl3_instance (bldcmhsl3_instance_t*)** - This is a pointer to bldcmhsl3_instance_t structure, which is filled by `fs_etpu_app_bldcmhsl3_init`.
- **speed_required_rpm (int32_t)** - This is the required motor speed in rpm.

4.3.5 void fs_etpu_app_bldcmhsl3_get_data(...)

This routine is used to get the application state data. This function has the following parameters:

- **bldcmhsl3_instance (bldcmhsl3_instance_t*)** - This is a pointer to bldcmhsl3_instance_t structure, which is filled by `fs_etpu_app_bldcmhsl3_init`.
- **bldcmhsl3_data (bldcmhsl3_data_t*)** - This is a pointer to bldcmhsl3_data_t structure of application state data, which is updated.

4.4 eTPU Block Diagram

The eTPU functions used to drive the BLDC motors with speed closed loop are located in the motor-control set of eTPU functions (set3 - DC motors). The eTPU functions within the set serve as building blocks for various motor-control applications. The following paragraphs describe the functionality of each block.

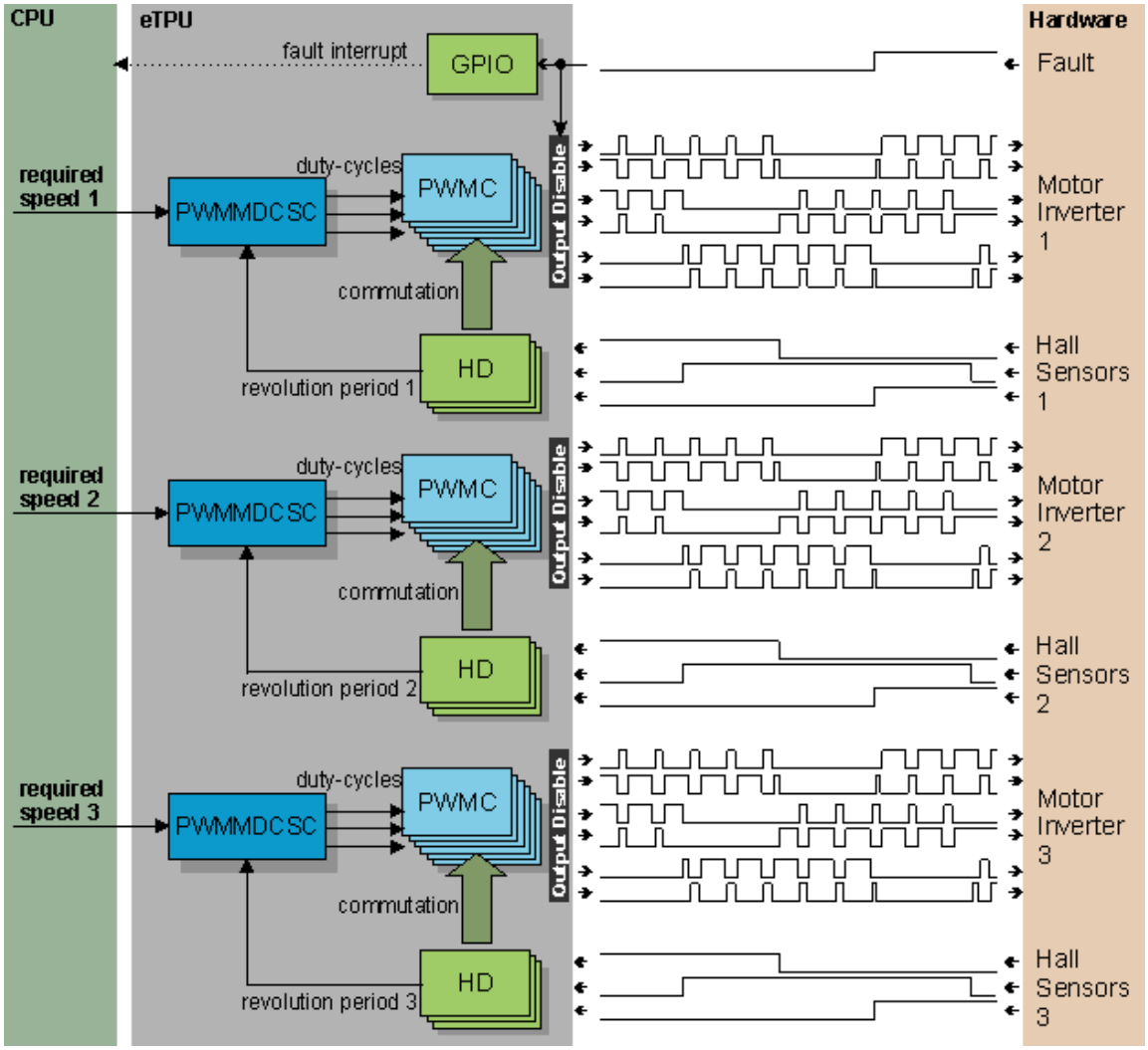

Figure 20. Block Diagram of eTPU Processing

4.4.1 PWM Generator (PWMMDCSC+PWMC)

The PWM master for DC motors with speed controller (PWMMDCSC) function and PWM commutated (PWMC) function enable the eTPU to generate PWM signals for driving a motor. The PWMMDCSC function was specially designed for this application. Actually, it includes two motor-control eTPU functions: speed controller (SC) and PWM master for DC motors (PWMMDC).

PWMMDC function synchronizes and updates up to three PWM phases. The phases are driven by the PWM commutated (PWMC) function that enables switching the phase ON and OFF. The PWMC function generates the PWM signals. The PWMMDC function controls the PWMC functions, does not generate any drive signal, and can be executed even on an eTPU channel not connected to an output pin. If connected to an output pin, the PWMMDC function turns the pin high and low, so that the high-time identifies the period of time in which the PWMMDC execution is active. In this way, the PWMMDC function, as with many of the motor-control eTPU functions, supports checking eTPU latencies using an oscilloscope.

The SC function is not intended to process an input or output signal. The purpose of the SC function is to control another eTPU function's input parameter. The SC function includes a controller algorithm. The controller calculates its output based on two inputs: a measured value and a desired value. The measured value, the actual motor speed, is calculated based on inputs provided by the quadrature decoder (QD) or Hall decoder (HD) function. The desired value is an output of a speed ramp, whose input is a SC function parameter, and can be provided by the CPU or another eTPU function. The controller algorithm is a general proportional-integral-derivative (PID) algorithm. It can be configured as PI or PID controller.

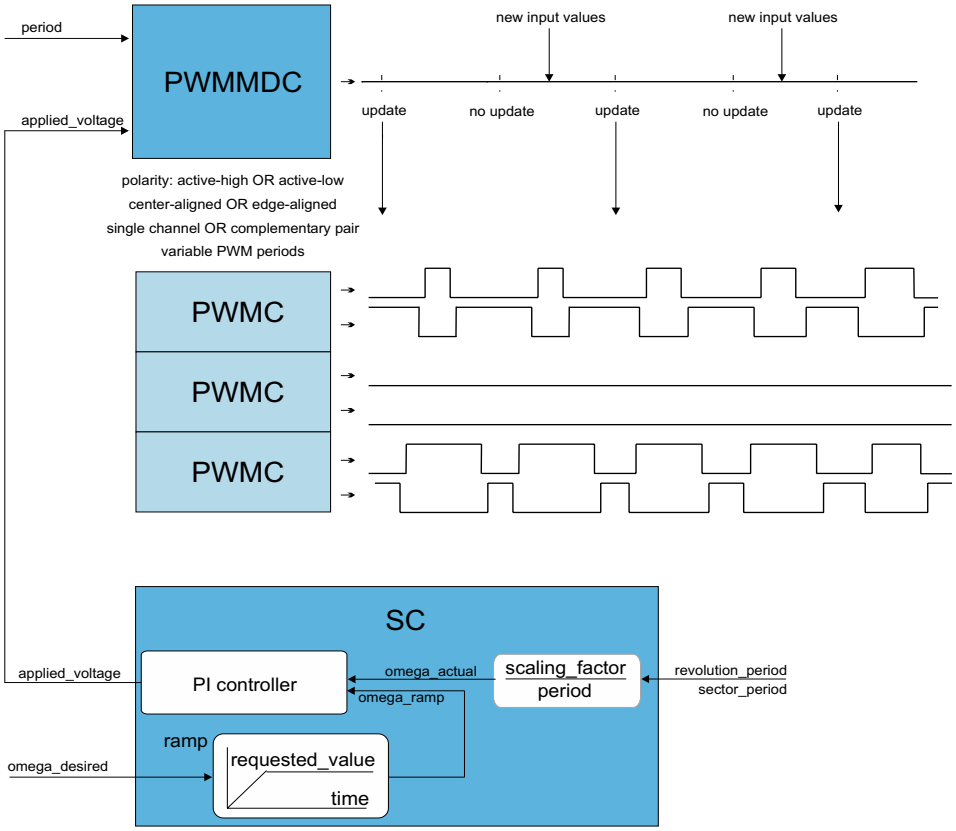

Figure 21. Functionality of SC+PWMMDC+PWMC

For more details about the PWMMDC and PWMC eTPU functions, refer to Reference 9.

For more details about the SC eTPU function, refer to Reference 8.

4.4.2 Hall Decoder (HD)

The Hall decoder eTPU function is intended to process signals generated by Hall sensors in motion control systems. The HD function uses three adjacent eTPU channels configured as inputs. The HD function calculates the following parameters for the CPU:

- Sector - determines the position of the motion system in one of the sectors.
- Direction - determines the direction of the motion system. A direction value 0 means a positive (incremental) direction, other values mean a negative (decremental) direction.

- Revolution counter - determines the number of motion system electrical revolutions. The revolution counter is incremented or decremented on each revolution, based on the current direction.
- Revolution period - determines the TCR time of the last revolution. The parameter value is updated each time the sector is changed. The revolution period is measured from the last edge of a similar type (low-high / high-low), on the same channel, to the current edge.
- Sector period - determines the TCR time between the last two changes of the sector. The parameter value is updated each time the sector is changed. The sector period is measured from the last edge to the current edge.
- Last edge time - stores the TCR time of the last incoming edge.

The HD function also performs commutations of PWMC phases.

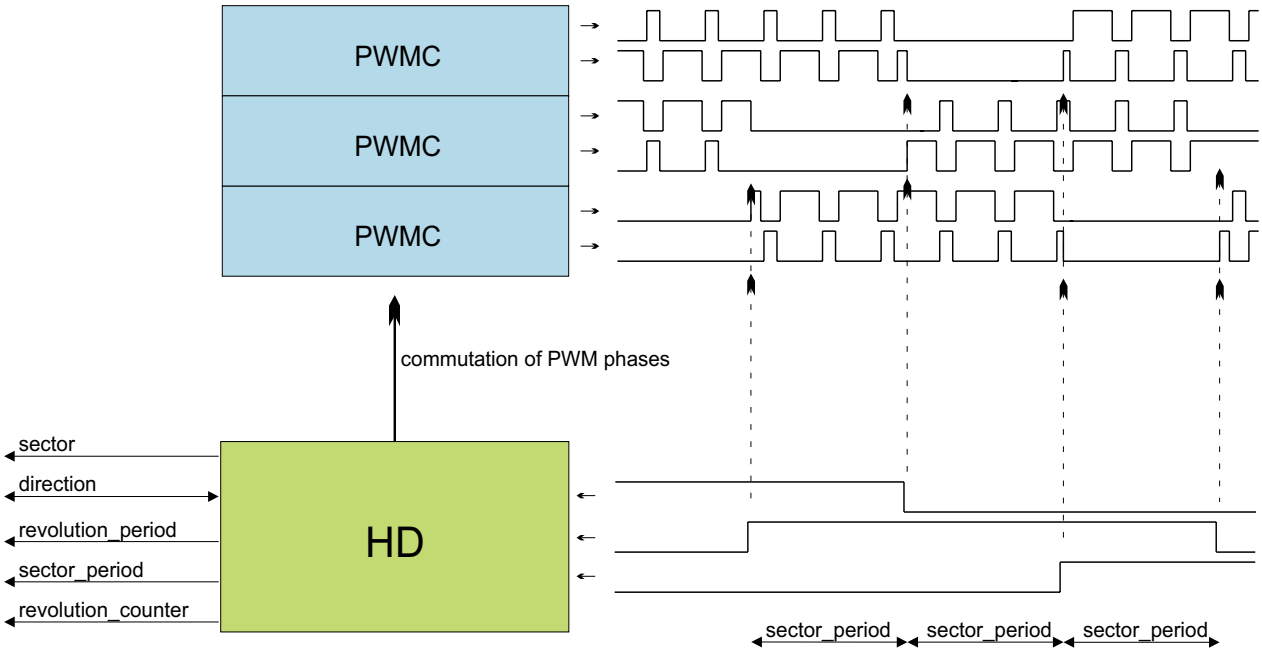

Figure 22. Functionality of HD

For more details about the HD eTPU function, refer to Reference 7.

4.4.3 General Purpose I/O (GPIO)

This function originates from the general eTPU function set (set1) and is included in the DC motor control eTPU function set as well. It allows the user to configure an eTPU channel as a general purpose input or output. There are 7 function modes supported:

- Input mode - update periodically
- Input mode - update on transition - either edge
- Input mode - update on transition - falling edge
- Input mode - update on transition - rising edge
- Input mode - update on request/disable transition and match updates

- Output mode - output low
- Output mode - output high

GPIO function is configured to generate an interrupt to the CPU in case of an overcurrent fault. The fault signal, which is usually connected to the eTPU output disable input, can be also connected to the eTPU channel assigned a GPIO function. When the fault signal turns active, selected output channels are immediately disabled by the eTPU hardware. At the same time, the GPIO function generates an interrupt, in order to notify the CPU about the fault occurrence.

For more details about the GPIO eTPU function, refer to Reference 10.

4.5 eTPU Timing

eTPU processing is event-driven. Once an event service begins, its execution cannot be interrupted by another event service. The other event services have to wait, which causes a service request latency. The maximum service request latency, or worst case latency (WCL), differs for each eTPU channel. The WCL is affected by the channel priority and activity on other channels. The WCL of each channel must be kept below a required limit. For example, the WCL of the PWMC channels must be lower than the PWM period.

A theoretical calculation of WCLs, for a given eTPU configuration, is not a trivial task. The motor control eTPU functions introduce a debugging feature that enables the user to check channel latencies using an oscilloscope, and eliminates the necessity of theoretical WCL calculations.

As mentioned earlier, some eTPU functions are not intended to process any input or output signals for driving the motor. These functions turn the output pin high and low, so that the high-time identifies the period of time in which the function execution is active. An oscilloscope can be used to determine how much the channel activity pulse varies in time, which indicates the channel service latency range. For example, when the oscilloscope time base is synchronized with the PWM periods, the behavior of a tested channel activity pulse can be described by one of the following cases:

- The pulse is asynchronous with the PWM periods. This means that the tested channel activity is not synchronized with the PWM periods.
- The pulse is synchronous with the PWM periods and stable. This means that the tested channel activity is synchronous with the PWM periods and is not delayed by any service latency.
- The pulse is synchronous with the PWM periods but its position varies in time. This means that the tested channel activity is synchronous with the PWM periods and the service latency varies in this time range.

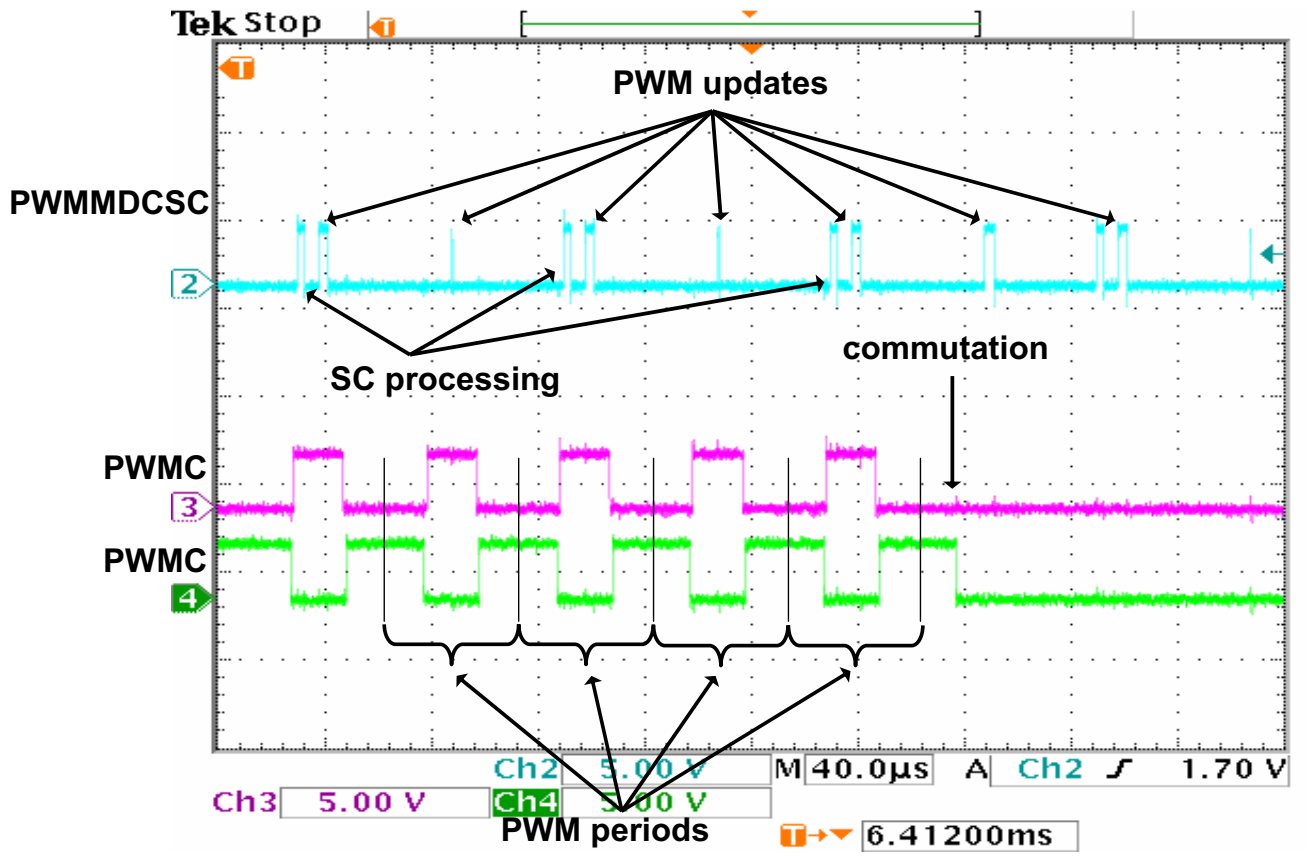

Figure 23. Oscilloscope Screen-shot and Explanation of eTPU Timing

Figure 23 and Figure 24 explain the application eTPU timing. The oscilloscope screen-shots depict a typical situation described below. A live view on the oscilloscope screen enables the user to see the variation of PWMMDCSC activity pulses, which determines the channel service latency ranges.

In Figure 23, signals 3 (pink) and 4 (green) are PWM signals of one phase. It is a complementary pair of centre-aligned PWM signals. The base channel (3) is of active-high polarity, while the complementary channel (4) is active-low. The PWM phase commutation is recognizable on the screen. The PWM period is 50µs, which corresponds to a PWM frequency of 20kHz.

Signal 2 (cyan) is generated by the PWMMDCSC eTPU function. Its pulses determine the speed controller (SC) processing time and PWM updates. SC processing pulse width determines the time necessary to calculate the motor speed from a revolution period measured by the Hall decoder (HD), calculate the required speed ramp, and apply the PI controller algorithm. This calculation is performed periodically at a 10kHz rate, which is every second PWM period.

The PWM update activity pulse is sometimes narrow and sometimes wide. The pulse is wide when a new value of applied motor voltage has been processed; it is narrow when no new value has been processed and the PWM duty-cycles are not updated.

The live view on the oscilloscope screen shows that the range of the PWMMDCSC channel service latencies are very low. The only noticeable latency is caused by Hall decoder (HD) activity. The HD eTPU

function processes the Hall signals transitions and commutes the PWM phases. The Hall signals transitions come asynchronously with the PWM periods.

The synchronization of the 3 PWMMDCSC functions is obvious from the Figure 25.

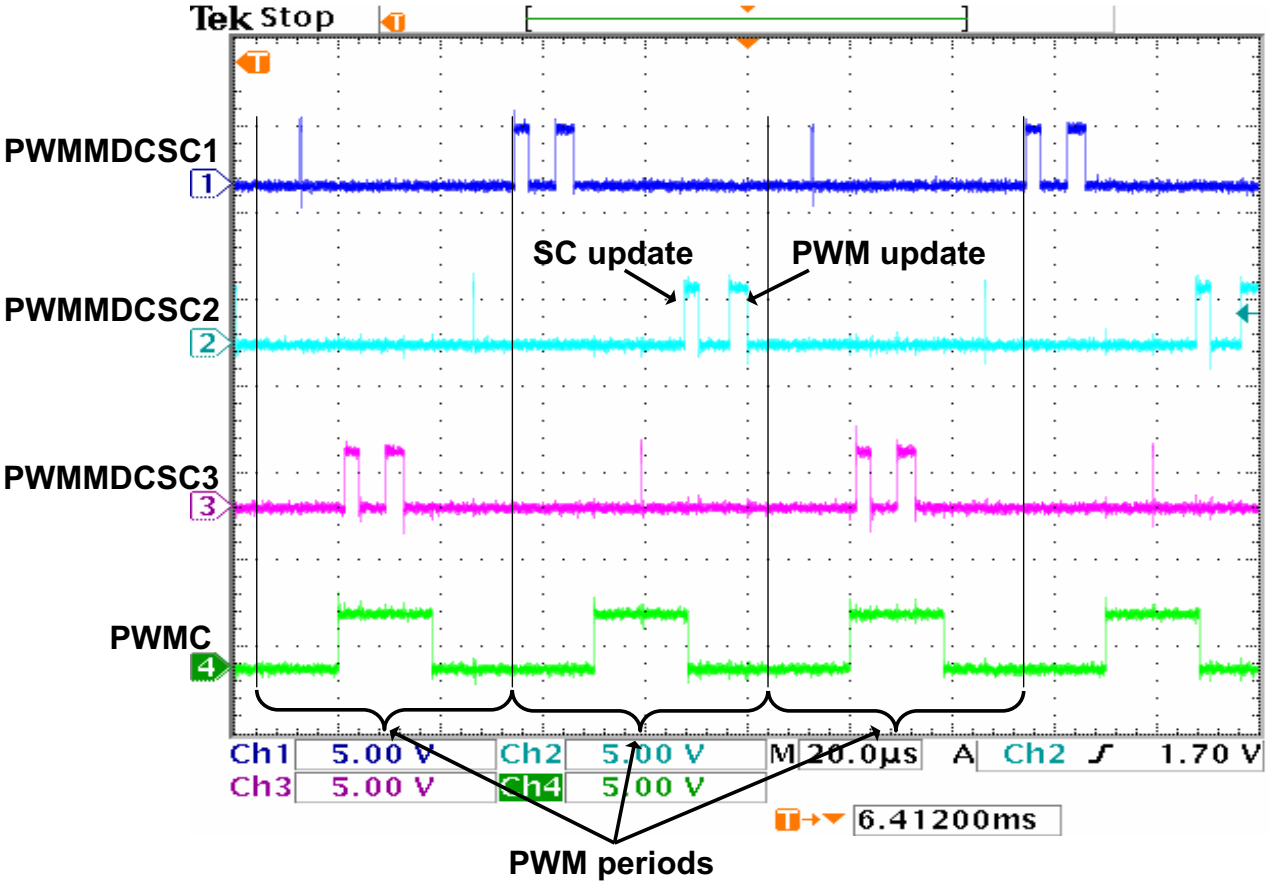

Figure 24. Oscilloscope Screen-shot and Explanation of 3 PWMMDCSC Functions Synchronization

5 Implementation Notes

5.1 Scaling of Quantities

The BLDC motor control algorithm running on eTPU uses a 24-bit fractional representation for all real quantities except time. The 24-bit signed fractional format is represented using 1.23 format (1 sign bit, 23 fractional bits). The most negative number that can be represented is -1.0, whose internal representation is 0x800000. The most positive number is 0x7FFFFF or 1.0 - 2^{-23}.

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

$$\text{Fractional Value} = \frac{\text{Real Value}}{\text{Real Quatity Range}}$$

where:

Fractional Value is a fractional representation of the real value [fract24]
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, RPM, etc.]

5.1.1 PI Controller Parameters

The PI controller parameters are set in a 32-bit extended fractional format 9.23. This format enables the user to set values in the range of -256.0 to 256.0 - 2^{-23}. Internally, the parameter value is transformed into one of two 24-bit formats, either 9.15, or 1.23, based on the value.

5.2 Speed Calculation

The PWMMDCSC eTPU function calculates the angular motor speed using a revolution period measured by the Hall decoder (HD) eTPU function. Optionally, the PWMMDCSC can use the sector period instead of the revolution period. The sector period is the time between two consecutive Hall signal transitions. A sum of six sector periods equals one revolution period. At a constant speed, each of the six sector periods may have a slightly different value, caused by an angular error in the Hall sensor positions. This error affects the PI controller behavior in a negative way. The revolution period is not affected by this error because the period is measured from a particular Hall signal transition to the same transition one revolution later. The revolution period is updated on each transition - six times per period.

The revolution period measured by the HD is the period of one electrical revolution. The electrical revolution is related to the mechanical revolution via the number of motor pole-pairs. The Pittman BLDC motor (N2311) is a 4 pole-pair motor. Hence, the mechanical revolution period is a period of four electrical revolutions.

The speed controller calculates the angular motor speed using the following equation:

$$\text{omega\_actual} = \frac{1}{\text{revolution\_period}} \cdot \text{scaling\_factor}$$

where:

omega_actual [fract24] is the actual angular speed as a fraction of the maximum speed range
1 is expressed as fractional value 0x7FFFFF
revolution_period [number of TCR ticks] is the period of one electrical revolution
scaling_factor is pre-calculated using the following equation

$$\text{scaling\_factor} = \frac{60 \cdot \text{etpu\_tcr\_freq}}{\text{omega\_max} \cdot \text{pole\_pairs}}$$

where:

etpu_tcr_freq [Hz] is a frequency of the internal eTPU timer (TCR1 or TCR2) used
omega_max [RPM] is a maximal speed range
pole_pairs is a number of motor pole-pairs

The internal eTPU timer (TCR1 or TCR2) frequency must be set so that the calculation of omega_actual both fits into the 24-bits arithmetic and its resolution is sufficient.

5.3 Definition of Commutation Tables

The PWM phases are commuted on each of the Hall signal transitions. This is internally done by applying two commutation commands that are associated with the particular Hall signal transition. The first command turns a phase off, and the second turns another phase on. Such pairs of commutation commands must be defined for each Hall signal transition, low-high and high-low, on each phase, and for both motor directions. These definitions are located in `etpu_app_bldcmhsl3.c` file. Each commutation command is a 32-bit word that consists of the following 8-bit parts.

- Channel number of the PWM phase base channel
- New base channel commutation state. It can be:
  - ON_ACTIVE_HIGH
  - ON_ACTIVE_LOW
  - OFF_LOW
  - OFF_HIGH
- New complementary channel commutation state. It can be:
  - ON_ACTIVE_HIGH
  - ON_ACTIVE_LOW
  - OFF_LOW
  - OFF_HIGH
- New phase options:
  - DUTY_POS
  - DUTY_NEG.

For a full description of all commutation command options, refer to Reference 9.

Figure 25 depicts the Pittman BLDC motor (N2311) motor timing diagram. The following example describes how to define several of the commutation commands based on this timing diagram:

A Hall signal phase A low-high transition comes at 300 electrical degrees (blue dotted line). The PWM phase B is turned off, with the pin in low state, and phase A on, with active-high polarity on the base channel and active-low polarity on the complementary channel, on this transition. Furthermore, the PWM phase B option is set to not change the calculated duty-cycle value, and the phase A option is set to negate the signed duty-cycle value, which generates a negative voltage. On phase A, this applies immediately, while, on phase B, this will apply 60 degrees later, when the phase is commuted on again. So, the commutation commands associated with the phase A low-high transition in an incremental motor direction are defined, using predefined macros, as follows:

```
phaseA_commut_cmds.lh_i_0 = PWM_phaseB_channel+
                            (FS_ETPU_PWMMDCSC_OFF_LOW  <<  8)+
```

```
                        (FS_ETPU_PWMMDCSC_OFF_LOW  << 16)+
                        (FS_ETPU_PWMMDCSC_DUTY_POS << 24)
phaseA_commut_cmds.lh_i_1 = PWM_phaseA_channel+
                        (FS_ETPU_PWMMDCSC_ON_ACTIVE_HIGH <<  8)+
                        (FS_ETPU_PWMMDCSC_ON_ACTIVE_LOW  << 16)+
                        (FS_ETPU_PWMMDCSC_DUTY_NEG       << 24)
```

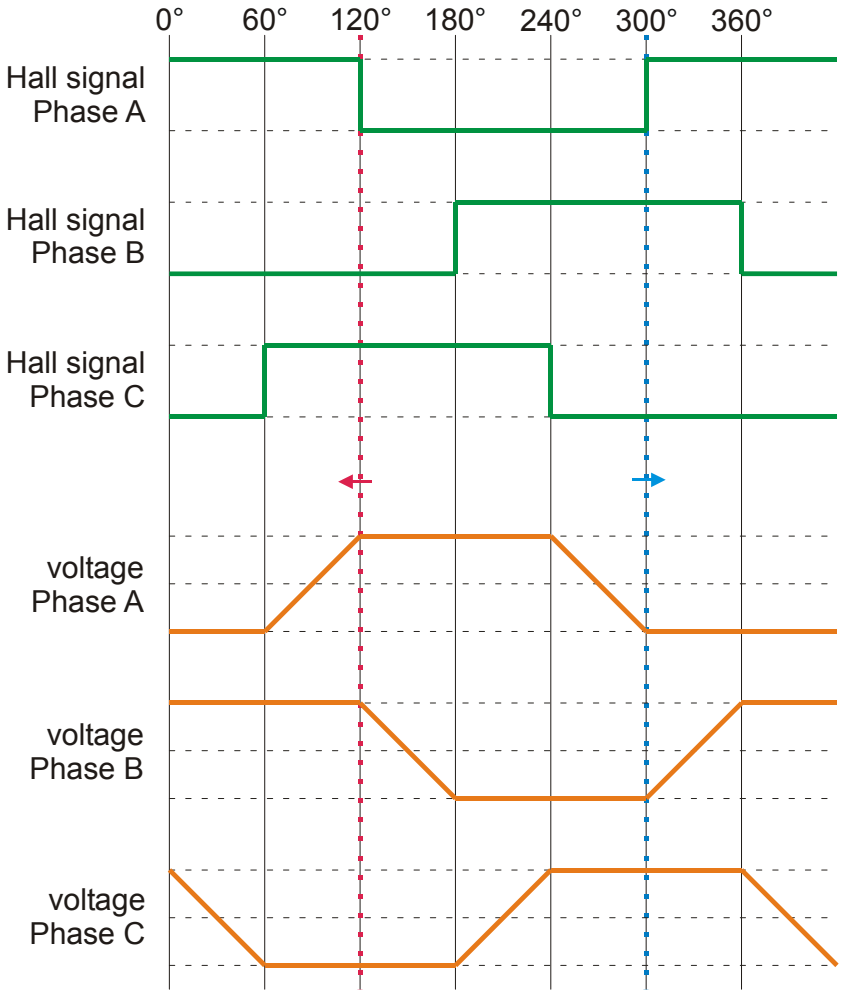

Figure 25. Pittman BLDC Motor (N2311) Timing Diagram

For decremental direction, read the motor timing diagram from right to left. The Hall signal phase A low-high transition comes at 120 electrical degrees (red dotted line). The PWM phase A is turned off and phase B on during this transition. The PWM phase A option is set to negate the duty-cycle in order to generate negative phase voltage, and the phase B option not to negate in order to generate positive voltage. So, the commutation commands associated with the phase A low-high transition in a decremental motor direction are as follows:

```
phaseA_commut_cmds.lh_d_0 = PWM_phaseA_channel+
```

```
                                (FS_ETPU_PWMMDCSC_OFF_LOW  <<  8)+
                                (FS_ETPU_PWMMDCSC_OFF_LOW  << 16)+
                                (FS_ETPU_PWMMDCSC_DUTY_NEG << 24)
phaseA_commut_cmds.lh_d_1 = PWM_phaseB_channel+
                                (FS_ETPU_PWMMDCSC_ON_ACTIVE_HIGH <<  8)+
                                (FS_ETPU_PWMMDCSC_ON_ACTIVE_LOW  << 16)+
                                (FS_ETPU_PWMMDCSC_DUTY_POS       << 24)
```

This way all commutation commands can be defined.

6 Microprocessor Usage

Table 3 shows how much memory is needed to run the application.

Table 3. Memory Usage in Bytes

Memory	Available	Used
FLASH (external)	2M	29 440
RAM	6K	5 999
eTPU code RAM	6K	5 428
eTPU data RAM	1.5K	1 304

The eTPU module usage in terms of time load can be easily determined based on the following facts:

- The maximum eTPU load produced by PWMMDCSC generation is 1316 eTPU cycles per one PWM period. The PWM frequency is set to 20kHz, thus the PWM period is 3750 eTPU cycles (eTPU module clock is 75 MHz, half of the 150 MHz CPU clock).
- According to Reference 7, the processing of one Hall signal transition, including the commutation, takes 308 eTPU cycles. The Hall signal transitions come asynchronously to the PWM periods. Six transitions are processed per one electrical motor revolution.
- The GPIO function processing does not affect the eTPU time load.

The values of eTPU load by each of the functions are influenced by compiler efficiency. The above numbers are given for guidance only and are subject to change. For up to date information, refer to the information provided in the latest release available from Freescale.

The peak of the eTPU time load occurs when the speed controller calculation and a Hall signal transition are processed within one PWM period. This peak value must be kept below 100%, which ensures that all processing fits into the PWM period, no service latency is longer than the PWM period, and thus the generated PWM signals are not affected.

Table 4 shows the eTPU module time load in several typical situations.

Table 4. eTPU Time Load

Situation	Average Time Load [%]	Peak Time Load Within PWM Period [%]
Motor Speed 300 RPM (120 commutations per second)	52.8	65
Motor Speed 10000 RPM (4000 commutations per second)	57.6	65

7 Summary and Conclusions

This application note provides the user with a description of the demo application 3 BLDC motors with speed closed loop. The application also demonstrates usage of the eTPU module on the ColdFire MCF523x, which results in a CPU independent motor drive. Lastly, the demo application is targeted at the MCF523x family of devices, but it could be easily reused with any device that has an eTPU.

8 References

Table 5. References

1. *MCF5235 Reference Manual*, MCF5235RM
2. *M523xEVB User's Manual*, M5235EVBUM
3. *Micro Power Stage for three BLDC Motors User's Manual*
4. Pittman's Motors web: http://www.pittmannet.com
5. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER"
6. *Enhanced Time Processing Unit Reference Manual*, ETPURM
7. "Using the Hall Decoder (HD) eTPU Function," AN2841
8. "Using the Speed Controller (SC) eTPU Function," AN2843
9. "Using the DC Motor Control PWM eTPU Functions," AN2480
10. "Using the General Purpose I/O eTPU functions (GPIO)," AN2850
11. "Using the DC Motor Control eTPU Function Set (set3)," AN2958
12. eTPU Graphical Configuration Tool, http://www.freescale.com, search keyword "ETPUGCT"
13. "DSP56F80x MC PWM Module in Motor Control Applications," AN1927

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2948
Rev. 2
09/2005