Freescale Semiconductor
Application Note

AN2955
Rev. 0, 06/2005

DC Motor with Speed and Current Closed Loops, 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 a DC motor drive based on 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, eliminating the microprocessor overhead for other duties.

The concept of the application is to create a speed and current closed loop DC driver using an optical, Hall-like position sensor. It serves as an example of a DC motor control system design using a Freescale microprocessor with the eTPU. It also illustrates the usage of dedicated motor control eTPU functions that are included in the DC motor control eTPU function set.

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

Table of Contents

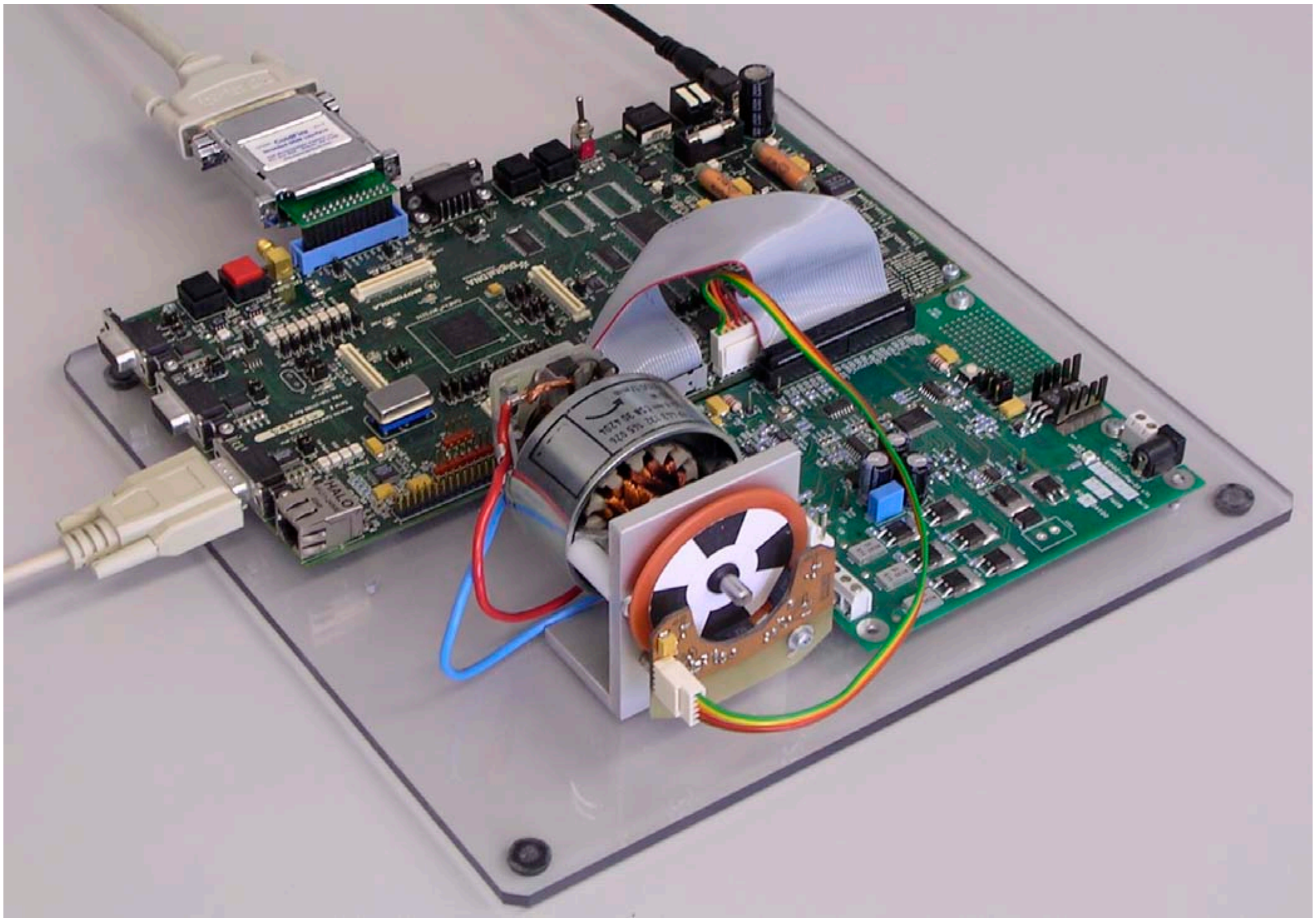


Figure 1. Using M523xEVB, 33395 Evaluation Motor Board, and DC Motor with Optical Sensors

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 5.

2 Target Motor Theory

A DC motor is a rotating electric machine where the stator of a permanent magnet DC motor is composed of two or more permanent magnet pole pieces. The rotor is composed of windings which are connected to a mechanical commutator. The opposite polarities of the energized winding and the stator magnet attract and the rotor will rotate until it is aligned with the stator. Just as the rotor reaches alignment, the brushes move across the commutator contacts and energize the next winding (see Figure 2).

Notice that the commutator is staggered from the rotor poles. If the connections of a DC motor are reversed, the motor will change directions.

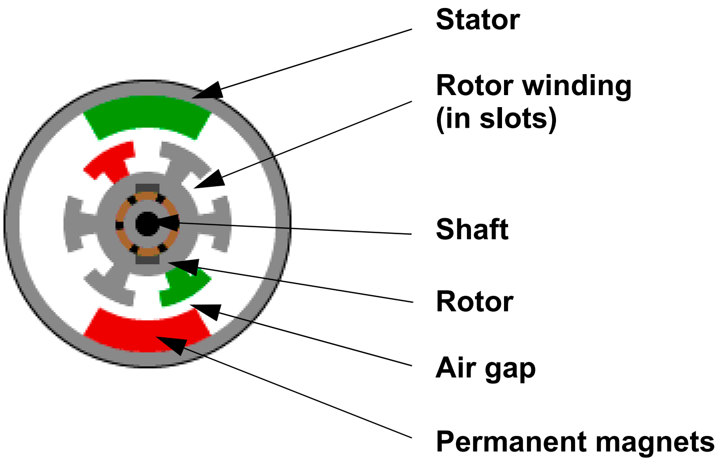


Figure 2. DC Motor - Cross Section

2.1 Digital Control of a DC Motor

For the common DC motor, a 2-phase power stage is used (see Figure 3). The power stage utilizes four power transistors that operate in either an independent or complementary mode.

In both modes, the power stage energizes two motor phases concurrently. The voltage is applied to the DC motor using a pulse width modulation (PWM) technique. 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 14.

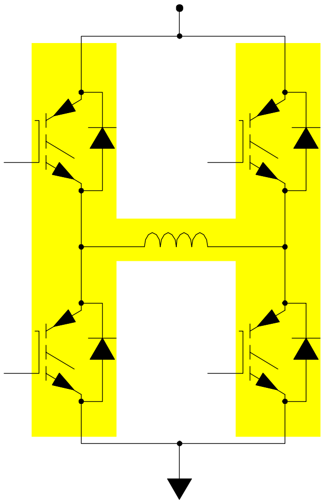


Figure 3. 2-Phase DC Motor Power Stage (H-bridge)

2.1.1 Speed And Current Control

The motor speed 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, controllers the required DC-bus current. The required DC-bus current is controlled by a current controller, which is also implemented as a conventional proportional-integral (PI) controller. The difference between the actual and required DC-bus current 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.

The current controller, which is the inner-loop controller, is updated more frequently, for example every PWM period, compared to the speed controller, which is the outer-loop controller.

The speed controller, as well as the current 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 a DC motor. The application meets the following performance specifications:

- Voltage control of a DC motor
- Targeted at ColdFire MCF523x evaluation board (M523xEVB), 33395 evaluation motor board, and 24V, 3200RPM, DC motor
- Control technique incorporates:
  - Voltage DC motor control with speed and current closed loop
  - Both directions of rotation
  - 4-quadrant operation
  - Start from any motor position without rotor alignment
  - Minimum speed of 200 RPM
  - Maximum speed of 1200 RPM (limited by power supply)
- Manual interface (start/stop switch, up/down push button control, LED indication)
- FreeMASTER control interface (speed set-up)
- FreeMASTER monitor
  - FreeMASTER graphical control page (required speed, actual motor speed, start/stop status, fault status)
  - FreeMASTER speed control scope (observes required, ramp, and actual speeds, required DC-bus current)
  - FreeMASTER current control scope (observes required and actual DC-bus currents, applied voltage)
  - Detail description of all eTPU functions used in the application (monitoring of channel registers and all function parameters in real time)
- DC bus over-current fault protection

3.2 Application Description

A standard system concept is chosen for the motor control function (see Figure 4). The system incorporates the following hardware:

- Evaluation board M523xEVB
- 33395 evaluation motor board
- DC motor with optical, Hall-like sensors
- Power supply 9V DC, 2.7Amps

The eTPU module runs the main control algorithm. The 2-phase PWM output signals for a 2-phase inverter are generated according to feedback signals from optical sensors and the input variable values, provided by the microprocessor CPU.

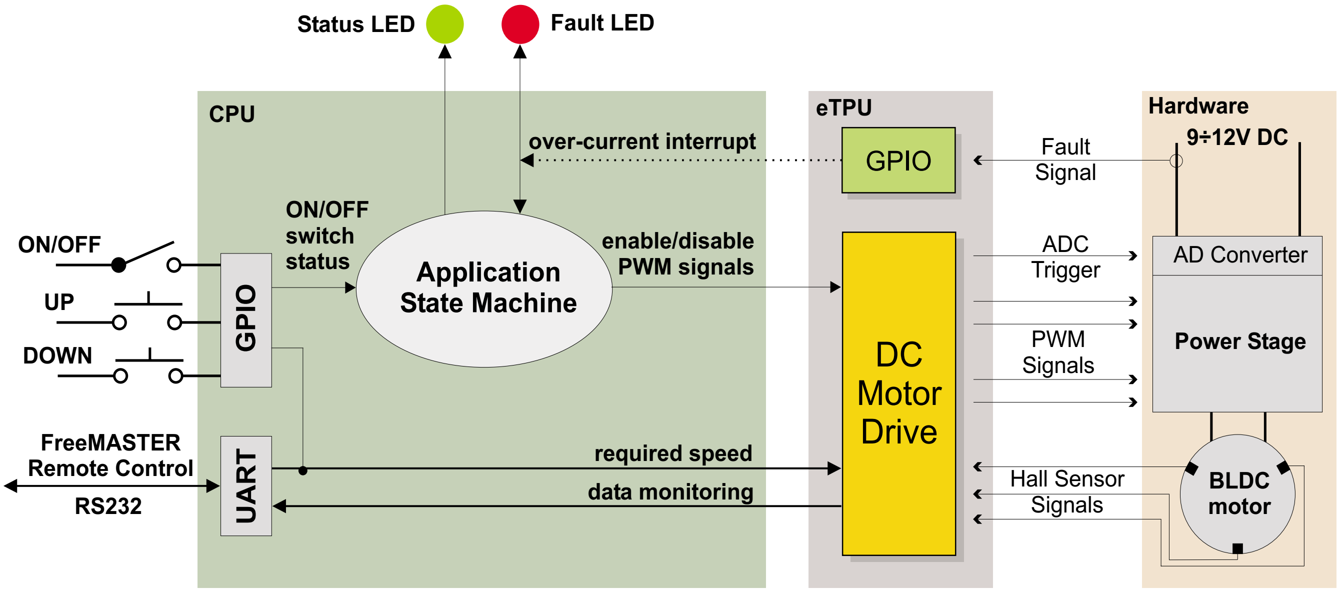


Figure 4. System Concept

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

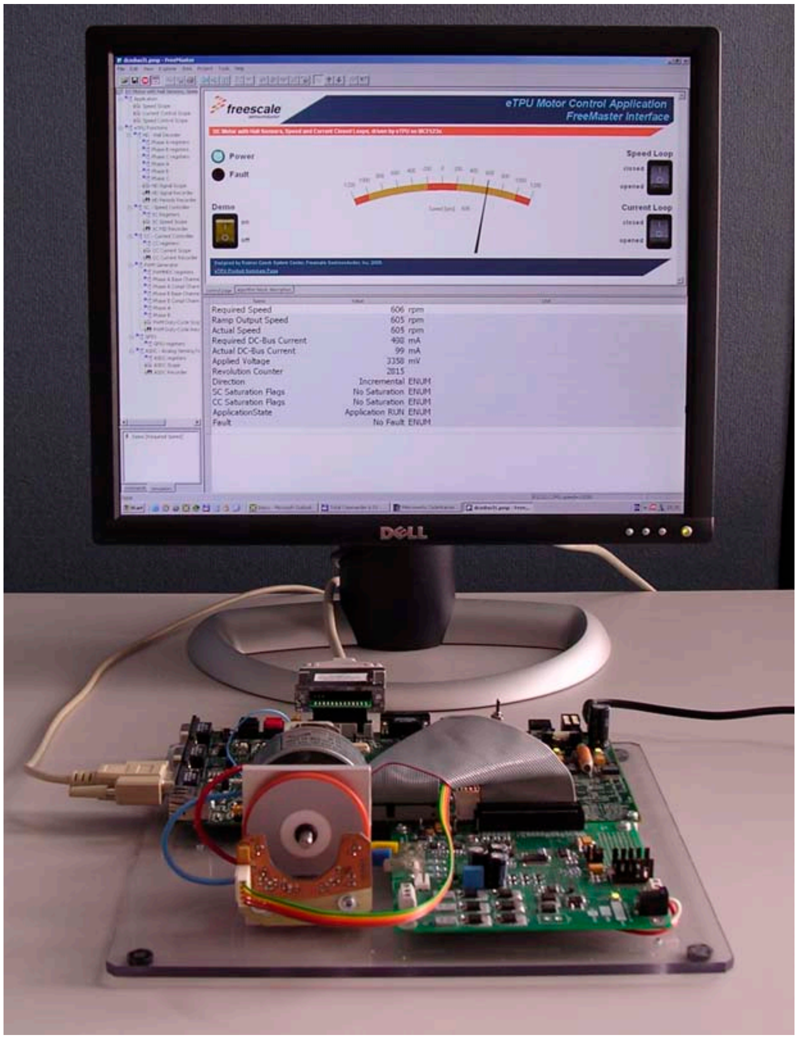
The CPU performs the following tasks:

- Periodically scans the user interface (ON/OFF switch, up and down buttons, FreeMASTER). Based on the user input, it handles the application state machine and calculates the required speeds, which 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:

- Four eTPU channels (PWMF) are used to generate PWM output signals.

- Three eTPU channels (HD) are used to process optical sensor signals. On each incoming edge, a revolution period is calculated.
- One eTPU channel (GPIO) is used to generate an interrupt to the CPU when the over-current fault signal activates.
- eTPU controls a speed closed 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. The speed PI control algorithm processes the error between the required and actual speed. The PI controller output is passed to the current controller as a newly corrected value of the required DC-bus current value.
- eTPU controls a current closed loop. The actual DC-bus current is sampled by an external AD converter, and the converted value is transferred by DMA to the eTPU. The actual value is compared with the required speed, provided by the speed controller. The current PI control algorithm processes the error between the required and actual current. The PI controller output is passed to the PWM generator as a newly corrected value of the applied motor voltage.



Name	Value	Unit
Required Speed	606	rpm
Ramp Output Speed	605	rpm
Actual Speed	605	rpm
Required DC-Bus Current	400	mA
Actual DC-Bus Current	99	mA
Applied Voltage	3358	mV
Revolution Counter	2815	
Direction	Incremental	ENUM
SC Saturation Flags	No Saturation	ENUM
CC Saturation Flags	No Saturation	ENUM
ApplicationState	Application RUN	ENUM
Fault	No Fault	ENUM

Figure 5. The Application and FreeMASTER Screen

3.2.1 User Interface

The application is interfaced by the following:

- ON/OFF switch on M523xEVB.
- Up/down buttons on M523xEVB, or
  FreeMASTER running on a PC connected to the M523xEVB via an RS232 serial cable.

The ON/OFF switch affects the application state and enables and disables the PWM phases. When the switch is in the off-position, no voltage is applied to the motor windings. When the ON/OFF switch is in the on-position, the motor speed can be controlled either by the up and down buttons on the M523xEVB, or by the FreeMASTER on the PC. The FreeMASTER also displays a control page, real-time values of application variables, and their time behavior using scopes.

FreeMASTER 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.

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 4.

3.3 Hardware Implementation and Application Setup

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

- M523xEVB
- 33395 evaluation motor board
- DC motor with optical, Hall-like sensors
- Power supply 9V DC, 2.7Amps

Figure 6 shows the connection of these parts.

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.

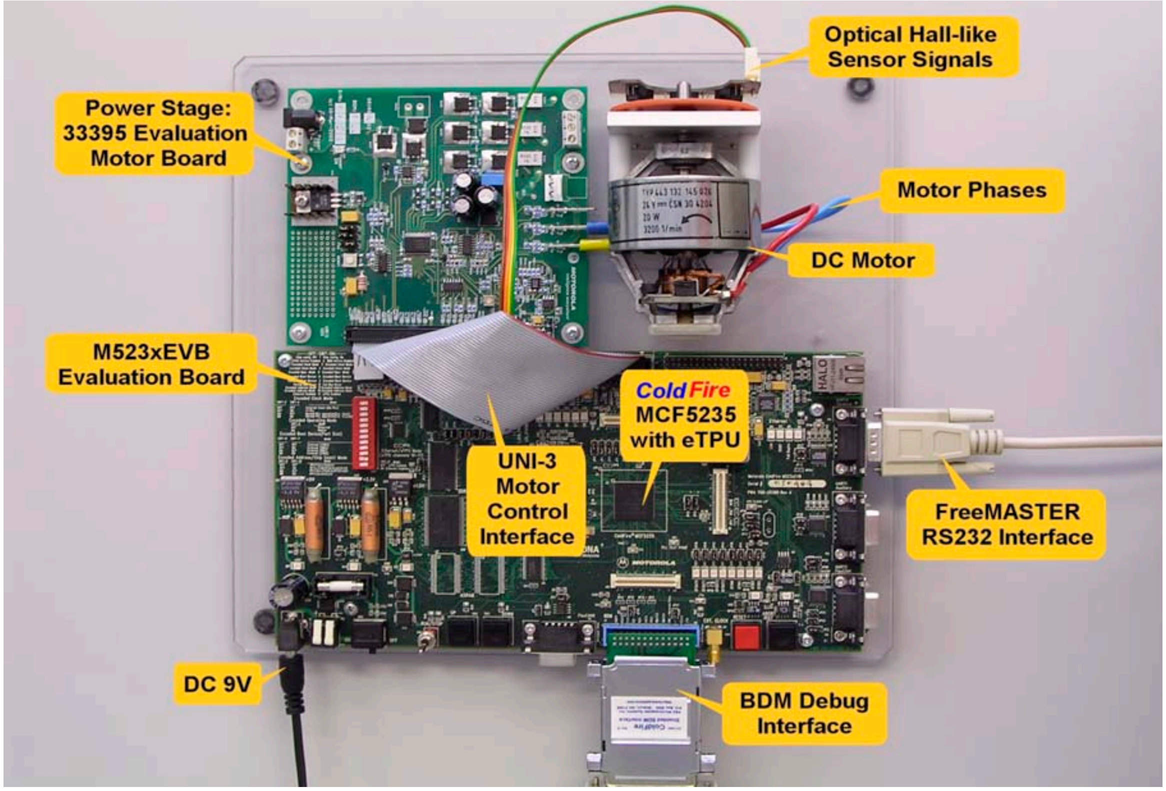

Optical Hall-like Sensor Signals
Power Stage: 33395 Evaluation Motor Board
Motor Phases
DC Motor
M523xEVB Evaluation Board
ColdFire MCF5235 with eTPU
UNI-3 Motor Control Interface
FreeMASTER RS232 Interface
DC 9V
BDM Debug Interface

Figure 6. Connection of Application Parts

Table 1. M523xEVB Jumper Settings

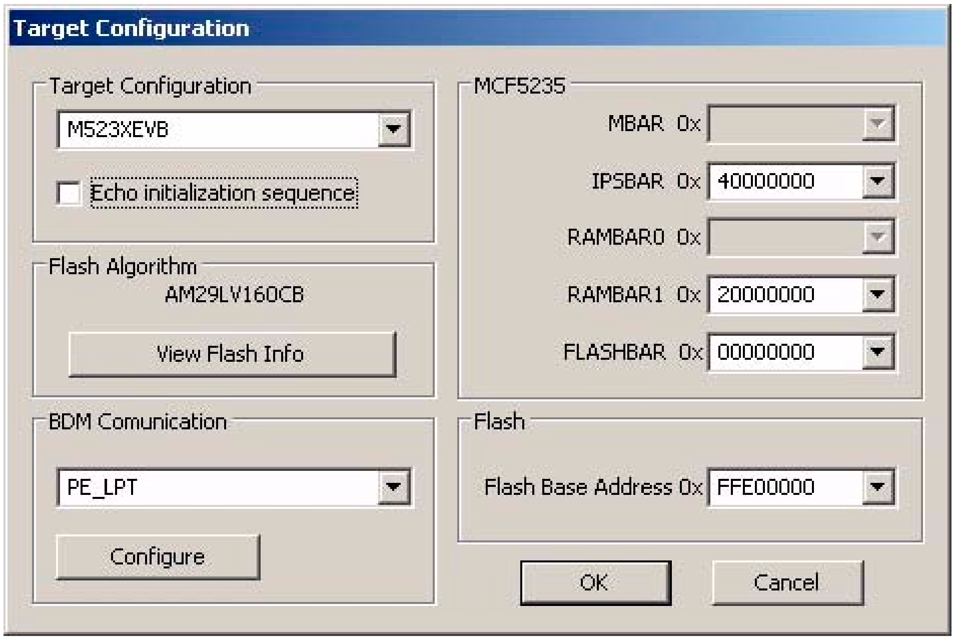
Jumper	Setting
JP1	1 2
JP2	1-2
JP3	1 2
JP4	1-2
JP5	1 2-3
JP6	1-2 3
JP7	1 2-3
JP8	1-2 3
JP9	1 2-3
JP10	1 2-3
JP11	1 2-3
JP12	1 2-3
JP13	1 2-3
JP14	1 2-3
JP15	1 2-3
JP16	1 2-3
JP17	1 2-3
JP18	1 2-3
JP19	1 2-3
JP20	1 2-3
JP21	1 2-3
JP22	1 2-3
JP23	1 2-3
JP24	1 2-3
JP25	1-2 3
JP26	1-2 3
JP27	1-2
JP28	1-2
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
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
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 7). 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 8). 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.



Target Configuration: M523XEVB

□ Echo initialization sequence

Flash Algorithm: AM29LV160CB

BDM Comunication: PE_LPT

MCF5235

MBAR 0x

IPSBAR 0x 40000000

RAMBAR0 0x

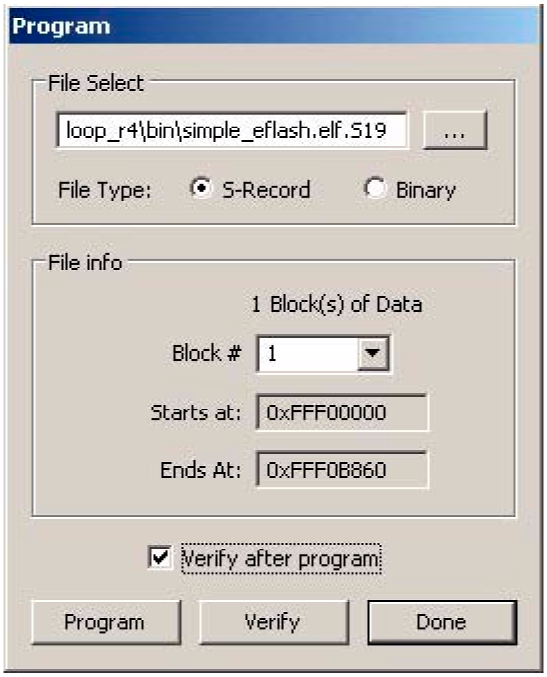
RAMBAR1 0x 20000000

FLASHBAR 0x 00000000

Flash

Flash Base Address 0x FFE00000

Figure 7. CFFlasher Target Configuration Window



File Select

loop_r4\bin\simple_eflash.elf.S19

File Type: ● S-Record ○ Binary

File info

1 Block(s) of Data

Block # 1

Starts at: 0xFFF00000

Ends At: 0xFFF0B860

☑ Verify after program

Figure 8. CFFlasher Program Window

3.3.3 Setting Overcurrent Level

The over-current fault signal is connected to the eTPU output disable pin (LETPUODIS) that handles eTPU hardware faults, along with the proper eTPU configuration. This connection is part of M523xEVB[1]. In order to enable handling of the fault by software, the fault signal, available on the LETPUODIS pin, must be connected to eTPU channel 4, which runs the GPIO function and generates an interrupt request to the CPU in the case of a fault. This connection must be done manually. Connect pin 6 (LETPUODIS) with pin 16 (ETPUCH4) on the eTPU header (see Figure 9).

The over-current level is set by trimmer R41 on M523xEVB (see Figure 10). Reference 3 describes what voltage the trimmer defines for the over-current comparator. Follow the steps below to set up the over-current level properly without measuring the voltage:

1. Connect all system parts according to Figure 6, connect pin 16 with pin 40 on the eTPU header. Now the over-current interrupt is disabled. The over-current fault is handled by hardware only.
2. Download and start the application.
3. Turn the ON/OFF switch ON. Using the up and down buttons, set the required speed to the maximum.
4. Adjust the R41 trimmer. You can find a level from which the red LED starts to light and the motor speed starts to be limited. Set the trimmer level somewhat higher, so that the motor can run at the maximum speed.
5. Turn the ON/OFF switch OFF.

[1] When the eTPU is configured for 32-channels, LTPUODIS is applicable to channels 0-15. When the ethernet is enabled (SW11 on), the function of LTPUODIS then changes to channels 0-7 and UTPUODIS thus controls channels 8-15.
Therefore the UTPUODIS must be tied to LTPUODIS to enable the application to work when ethernet is enabled.

6. Connect pin 16 with pin 6 on the eTPU header. Now the over-current interrupt is enabled. The over-current fault is handled by both hardware and software.
7. Turn the ON/OFF switch ON. Using the up and down buttons, set the required speed to the maximum.
   If the application goes to the fault state during the acceleration, adjust the R41 trimmer level somewhat higher, so that the motor can get to the maximum speed.

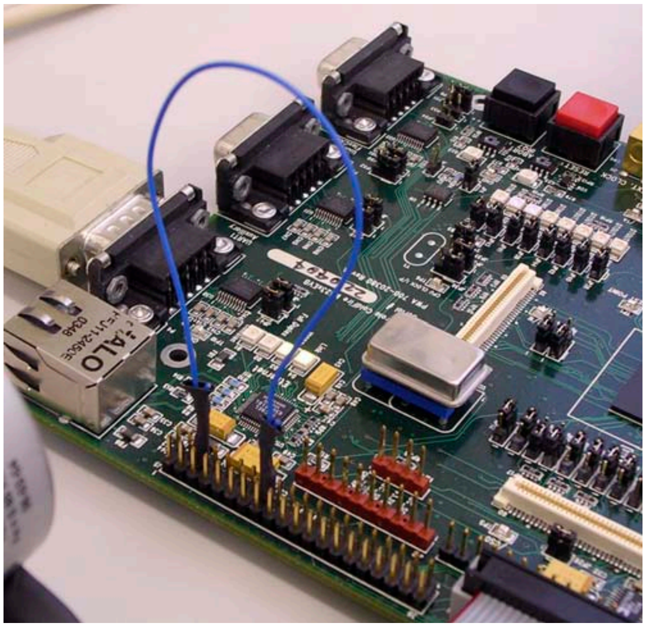


Figure 9. Connection Between LETPUODIS and ETPUCH4 on the eTPU Header

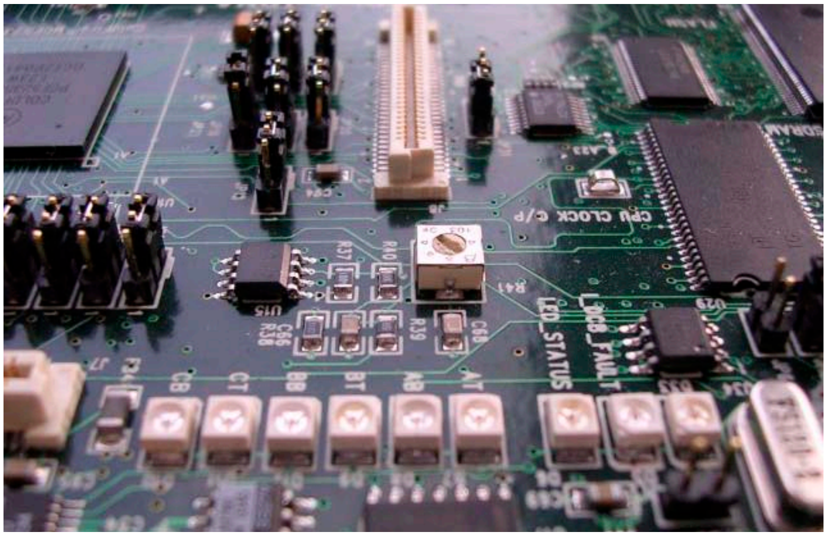


Figure 10. Overcurrent Level Trimmer on M523xEVB (R41)

3.3.4 33395 Evaluation Motor Board

The 33395 evaluation motor board is a 12-volt, 8-amp 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 power stage enables sensing a variety of feedback signals suitable for different motor control techniques. It measures all the three phase currents, reconstructs the DC-bus current from them, the DC-bus voltage, and the back-EMF voltages with zero cross sensing. All the analog signals are adapted to be directly sampled by the A/D converter. This single-board power stage contains an analog bridge gate driver integrated circuitry, sensing and control circuitry, power N-MOSFET transistors, DC-bus break chopper, as well as various interface connectors for the supply and the motor.

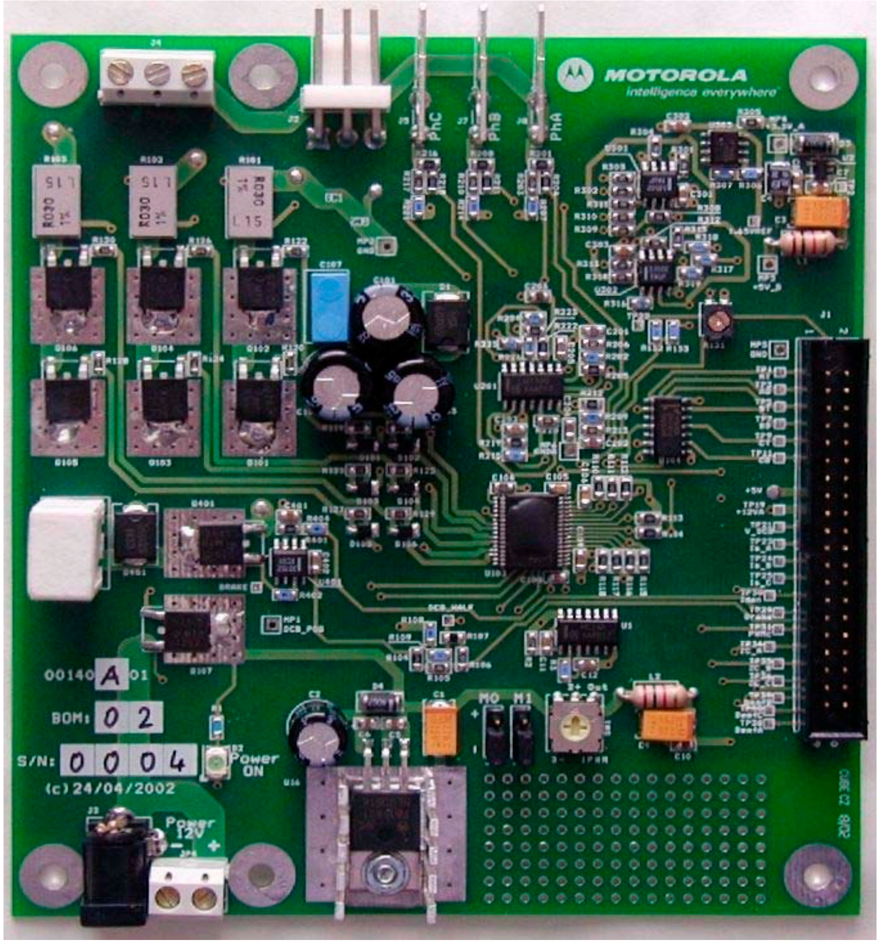


Figure 11. 33395 Evaluation Motor Board

For more information, refer to Reference 3.

3.3.5 DC Motor with Optical, Hall-like Sensors

The enclosed motor is a low-voltage DC motor. The additional equipment which enables to measure motor speed is attached to this motor (see Figure 12). It deals with the following parts:

- A disk with black and white sectors on its surface. The disk is attached to the motor shaft.
- A group of three photomicrosensors EE-SY313 that are placed on the special position sensor board at the distance of 5 mm from the on-shaft disk.

The described equipment replaces Hall sensors which are widely used for motion system position sensing. Each photomicrosensor generates the signal in dependence on the changing color (black or white) of the on-shaft disk. This way three Hall-like sensor signals are produced.

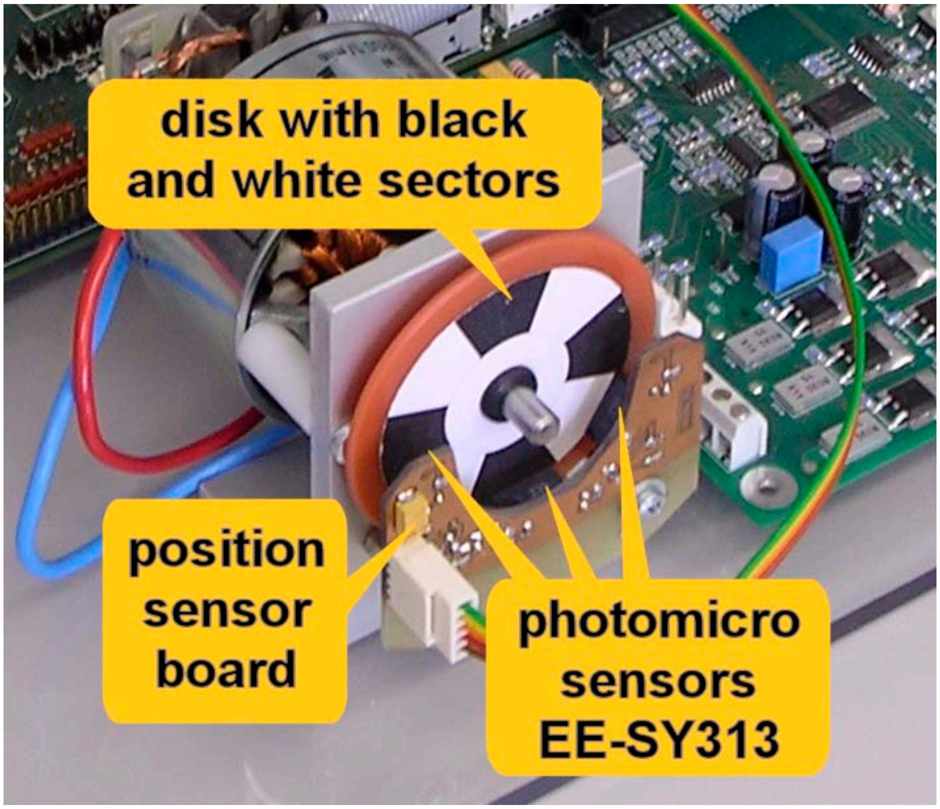

disk with black and white sectors
position sensor board
photomicro sensors EE-SY313

Figure 12. Optical Sensors used for the Motion System Position Sensing

3.3.6 Power Supply

The power supply supplied with the M523xEVB, 9.0V/2.7A, is also used to power the 33395 evaluation motor board. The application is scaled for this 9V power supply. In case a 12V power supply is used instead, the application should be rescaled for the wider voltage and speed range.

4 Software Design

This section describes the software design of the DC 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.

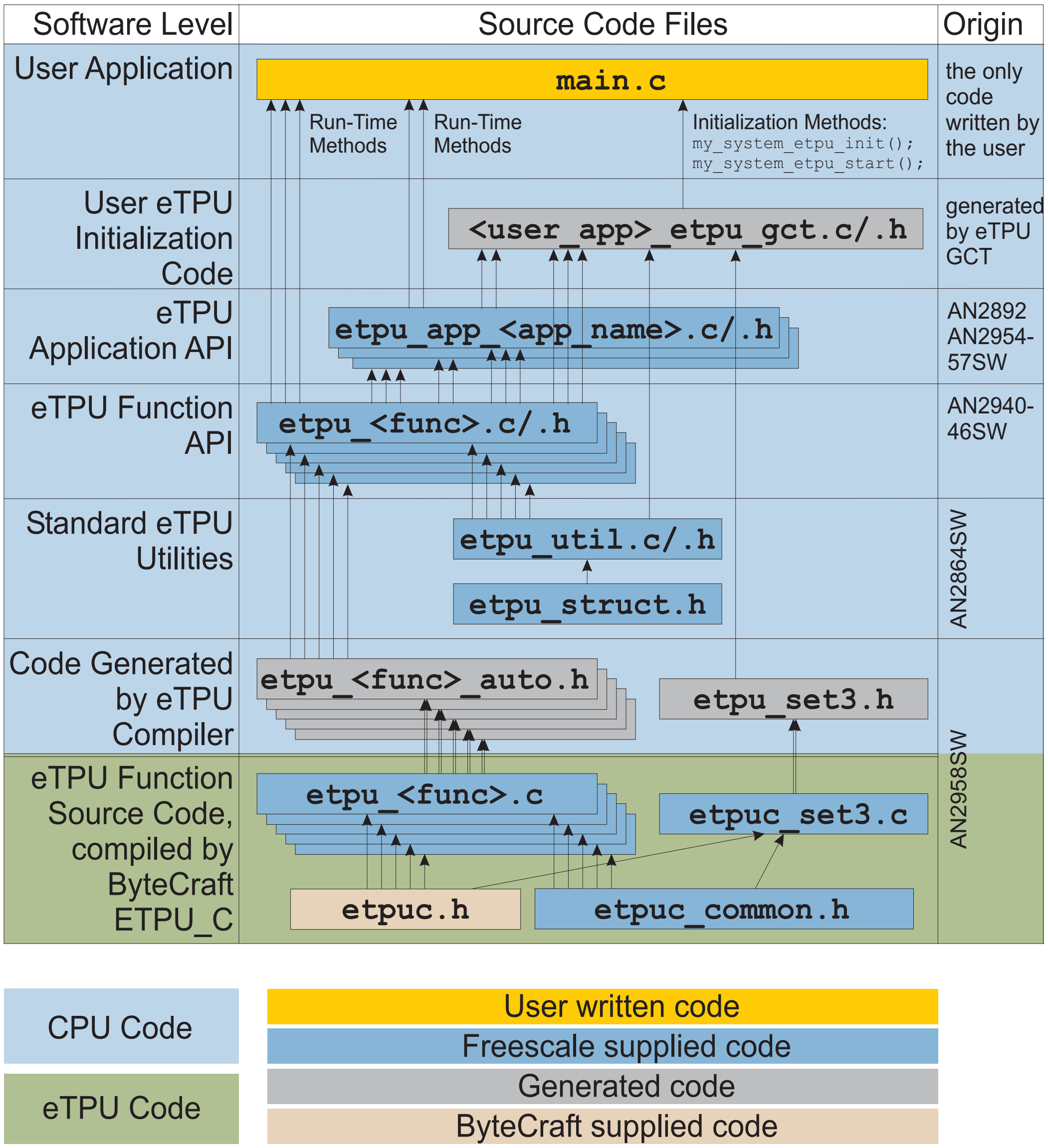


Figure 13. eTPU Project Structure

4.1 CPU Software Flowchart

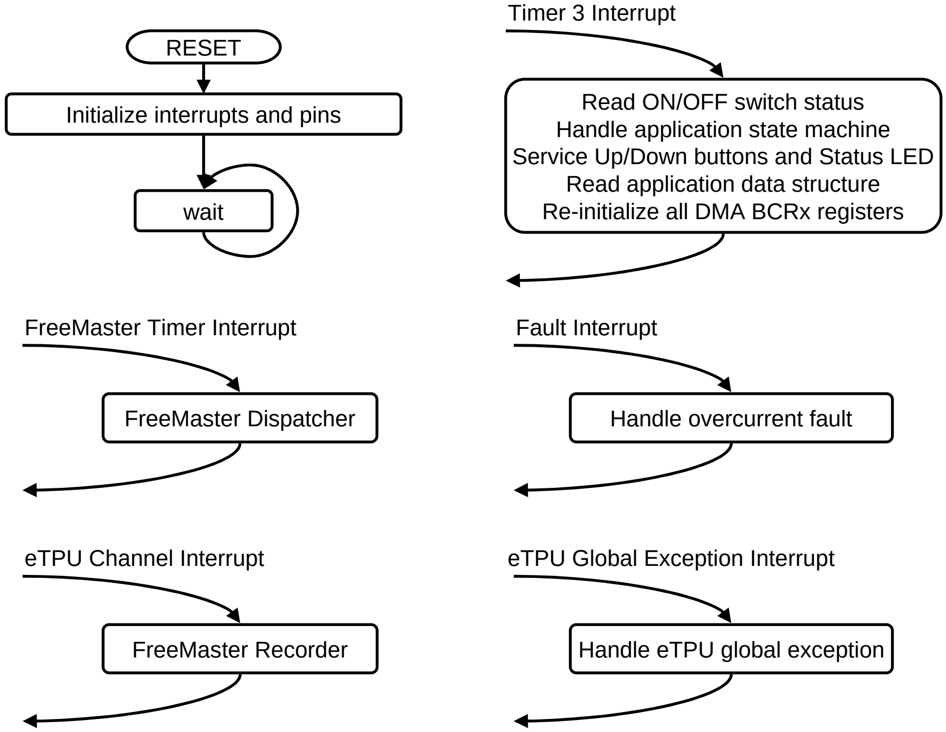

RESET
Initialize interrupts and pins
wait
Timer 3 Interrupt
Read ON/OFF switch status
Handle application state machine
Service Up/Down buttons and Status LED
Read application data structure
Re-initialize all DMA BCRx registers
FreeMaster Timer Interrupt
FreeMaster Dispatcher
Fault Interrupt
Handle overcurrent fault
eTPU Channel Interrupt
FreeMaster Recorder
eTPU Global Exception Interrupt
Handle eTPU global exception

Figure 14. 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 4 and 7 interrupts, and the eTPU global exception interrupt, together with initialization of the GPIO and LETPUODIS 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:

- Read the ON/OFF switch status
- Handle the application state machine

  The application state diagram is described in detail below.
- Service the up and down buttons and the status LED by the ApplicationButtonsAndStatusLed function

- Read the data structure through the eTPU application API routine `fs_etpu_app_dcmhscl1_get_data` (see 4.3).
- Re-initialize all DMA BCRx registers in order to ensure the continuous operation of the DMA channels (see 4.4.5)

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 PWMMDC eTPU function running on eTPU channel 7, is handled by the etpu_ch7_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 4, is handled by the etpu_ch4_isr function. The following actions are performed in order to switch the motor off:

- Reset the required speed
- 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 speed
- 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 15). After reset, the application goes firstly to APP_STATE_INIT. Where the ON/OFF switch is in the OFF position, the APP_STATE_STOP follows; otherwise, the APP_STATE_MOTOR_FAULT is entered and the ON/OFF switch must be turned OFF to get from APP_STATE_MOTOR_FAULT to 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 ON/OFF switch position. 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 15), the APP_STATE_MOTOR_FAULT is entered. This fault is cleared by moving the ON/OFF 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 15), the APP_STATE_GLOBAL_FAULT is entered. The global fault is cleared by moving the ON/OFF switch to the OFF position and thus entering the APP_STATE_INIT.

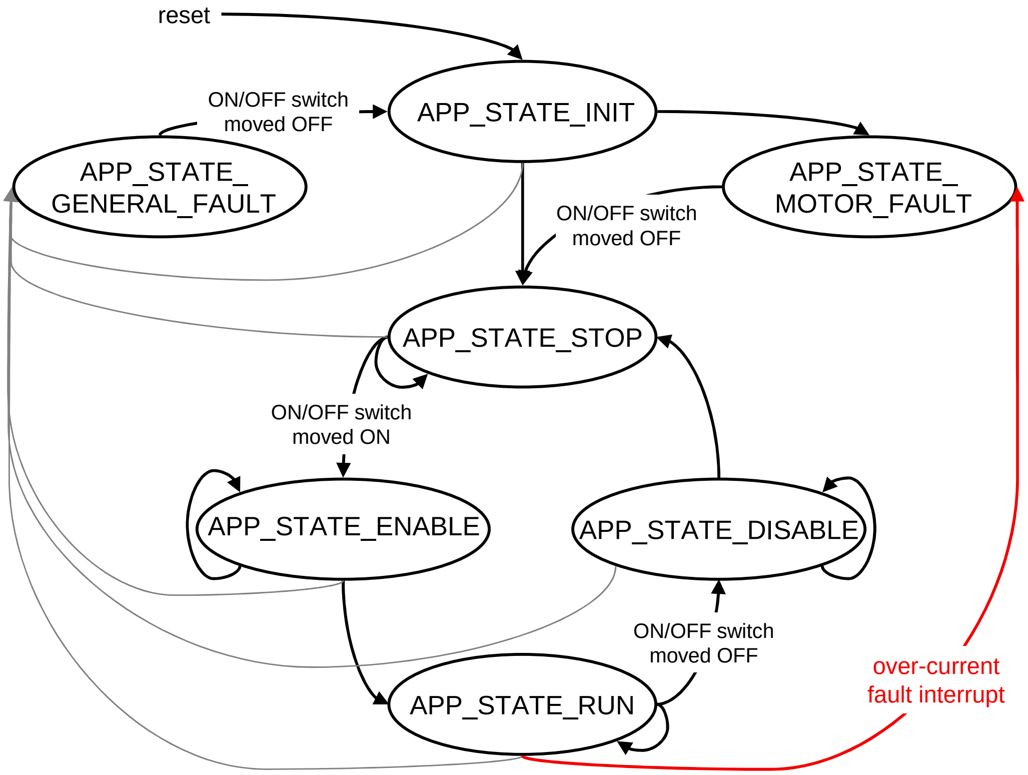


Figure 15. Application State Diagram

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 QSPI & ADC
- Initialize DMA0-3 channels & DMA timer 0
- 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 ON/OFF 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 `dcmhscl1_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 13.

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 optical 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 = 64

  The TCR2 internal eTPU clock is set to a rate of 146.484 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 1 - Hall decoder (HD) - Phase A

  Channel 2 - Hall decoder (HD) - Phase B

Channel 3 - Hall decoder (HD) - Phase C
Channel 5 - speed controller (SC)
Channel 6 - current controller (CC)
Channel 7 - PWM master for DC motors (PWMMDC)
Channel 8 - PWM full range (PWMF) - Phase A - base channel
Channel 10 - PWM full range (PWMF) - Phase B - base channel
Channel 14 - analog sensing for DC motors (ASDC)

These eTPU channels are initialized by the `fs_etpu_app_dcmhscl1_init` eTPU application API function (see 4.3). The application settings are as follows:

— PWM phases-type is full range complementary pairs

— PWM frequency 20kHz

— PWM dead-time 1µs

— Motor speed range 1 400 RPM

— Motor speed minimum 50 RPM

— DC-bus voltage 9V

— Number of motor pole pairs 8

— Motor speed calculated using HD revolution period

— Speed controller update frequency 1250 Hz

— Speed controller PI parameters:
P-gain is 0.256 (0x0020C4 * 2^{-15}), and
I-gain is 0.005127 (0x0000A8 * 2^{-15}).
The controller parameters were experimentally tuned.

— Ramp parameters:
0.3s to ramp up from zero to the maximum speed,
0.3s to ramp down from the maximum speed to zero.

— Current controller PI parameters:
P-gain is 25.6 (0x0CCCCC * 2^{-15}), and
I-gain is 0 (0x000000 * 2^{-15}).
The controller parameters were experimentally tuned.

— DC-bus current measurement range is 14.55A

— ASDC function triggers A/D converter on high-low edge

— DC-bus current measurement time including A/D conversion time and DMA transfer time is 11us

— p_ASDC_result_queue pointer contains the address of the measured sample of DC-bus current

— Measured sample of DC-bus current is shifted left by 10 bits

— DC-bus current sample offset within ASDC_result_queue is 0

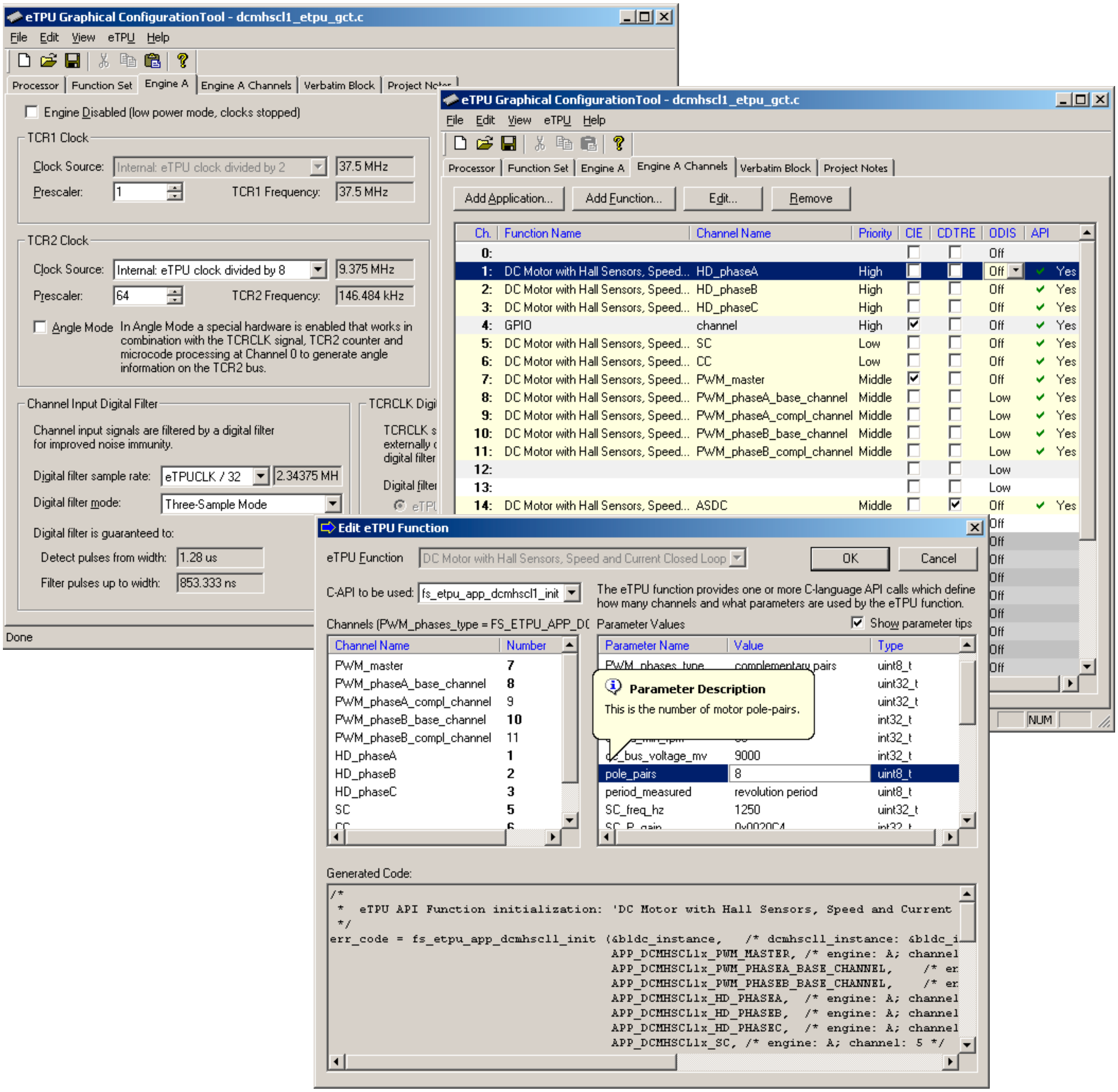
— ASDC EWMA filter time constant is 200 us

- 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.



☐ Engine Disabled (low power mode, clocks stopped)

TCR1 Clock

Clock Source: Internal: eTPU clock divided by 2 — 37.5 MHz

Prescaler: 1 — TCR1 Frequency: 37.5 MHz

TCR2 Clock

Clock Source: Internal: eTPU clock divided by 8 — 9.375 MHz

Prescaler: 64 — TCR2 Frequency: 146.484 kHz

☐ Angle Mode — In Angle Mode a special hardware is enabled that works in combination with the TCRCLK signal, TCR2 counter and microcode processing at Channel 0 to generate angle information on the TCR2 bus.

Channel Input Digital Filter

Channel input signals are filtered by a digital filter for improved noise immunity.

Digital filter sample rate: eTPUCLK / 32 — 2.34375 MH

Digital filter mode: Three-Sample Mode

Digital filter is guaranteed to:

Detect pulses from width: 1.28 us

Filter pulses up to width: 853.333 ns

TCRCLK Digi

TCRCLK s externally c digital filter

Digital filter

○ eTPU

Ch.	Function Name	Channel Name	Priority	CIE	CDTRE	ODIS	API
0:				☐	☐	Off	
1:	DC Motor with Hall Sensors, Speed...	HD_phaseA	High	☐	☐	Off	✓ Yes
2:	DC Motor with Hall Sensors, Speed...	HD_phaseB	High	☐	☐	Off	✓ Yes
3:	DC Motor with Hall Sensors, Speed...	HD_phaseC	High	☐	☐	Off	✓ Yes
4:	GPIO	channel	High	☑	☐	Off	✓ Yes
5:	DC Motor with Hall Sensors, Speed...	SC	Low	☐	☐	Off	✓ Yes
6:	DC Motor with Hall Sensors, Speed...	CC	Low	☐	☐	Off	✓ Yes
7:	DC Motor with Hall Sensors, Speed...	PWM_master	Middle	☑	☐	Off	✓ Yes
8:	DC Motor with Hall Sensors, Speed...	PWM_phaseA_base_channel	Middle	☐	☐	Low	✓ Yes
9:	DC Motor with Hall Sensors, Speed...	PWM_phaseA_compl_channel	Middle	☐	☐	Low	✓ Yes
10:	DC Motor with Hall Sensors, Speed...	PWM_phaseB_base_channel	Middle	☐	☐	Low	✓ Yes
11:	DC Motor with Hall Sensors, Speed...	PWM_phaseB_compl_channel	Middle	☐	☐	Low	✓ Yes
12:				☐	☐	Low	
13:				☐	☐	Low	
14:	DC Motor with Hall Sensors, Speed...	ASDC	Middle	☐	☑	Off	✓ Yes

eTPU Function: DC Motor with Hall Sensors, Speed and Current Closed Loop

C-API to be used: fs_etpu_app_dcmhscl1_init

The eTPU function provides one or more C-language API calls which define how many channels and what parameters are used by the eTPU function.

Channels (PWM_phases_type = FS_ETPU_APP_DC

Channel Name	Number
PWM_master	7
PWM_phaseA_base_channel	8
PWM_phaseA_compl_channel	9
PWM_phaseB_base_channel	10
PWM_phaseB_compl_channel	11
HD_phaseA	1
HD_phaseB	2
HD_phaseC	3
SC	5
CC	6

Parameter Values — ☑ Show parameter tips

Parameter Name	Value	Type
PWM_phases_type	complementary pairs	uint8_t
		uint32_t
		uint32_t
		int32_t
		int32_t
dc_bus_voltage_mv	9000	int32_t
pole_pairs	8	uint8_t
period_measured	revolution period	uint8_t
SC_freq_hz	1250	uint32_t
SC_P_gain	0x0020C4	int32_t

This is the number of motor pole-pairs.

Generated Code:

```
/*
 *  eTPU API Function initialization: 'DC Motor with Hall Sensors, Speed and Current
 */
err_code = fs_etpu_app_dcmhscl1_init (&bldc_instance,   /* dcmhscl1_instance: &bldc_i
                                      APP_DCMHSCL1x_PWM_MASTER, /* engine: A; channel
                                      APP_DCMHSCL1x_PWM_PHASEA_BASE_CHANNEL,    /* en
                                      APP_DCMHSCL1x_PWM_PHASEB_BASE_CHANNEL,    /* en
                                      APP_DCMHSCL1x_HD_PHASEA,  /* engine: A; channel
                                      APP_DCMHSCL1x_HD_PHASEB,  /* engine: A; channel
                                      APP_DCMHSCL1x_HD_PHASEC,  /* engine: A; channel
                                      APP_DCMHSCL1x_SC, /* engine: A; channel: 5 */
```

Figure 16. 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.

4.2.1.3 Initialization of Necessary Peripheral Modules

Several peripheral modules needed for DC-bus current measurement processing must be initialized:

- An external analog to digital converter AD7928BRA, interfaced to the CPU via the QSPI, is used for sampling the DC-bus current analog value. Initialization of the AD Converter and QSPI module is done by the `ADCInit` routine.
- 4 DMA channels and a DMA timer are used to periodically transfer data from QSPI to the eTPU data memory. Initialization of these modules is done by the `DMAInit` routine.

For a detailed description of DC-bus current measurement and data transfer processing, see 4.4.5

4.2.2 APP_STATE_STOP

In this state, the PWM signals are disabled and the motor is off. The motor shaft 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 optical, Hall-like sensor signals in FreeMASTER.

When the ON/OFF 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 speed
- 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 motor is on. The required motor speed can be set using the up and down buttons on the M523xEVB or by using FreeMASTER. The latest value is periodically written to the eTPU.

When the ON/OFF 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 speed
- 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 ON/OFF 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 ON/OFF 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 DCMHSCL1 is an abbreviation for ‘DC Motor with Hall Sensors, Speed and Current Closed Loops’ eTPU motor - control application. As there are several DC motor applications with Hall sensors, speed and current closed loops, the numeral 1 denotes the first such application in order.

The DCMHSCL1 eTPU application API is described in the following paragraphs. There are 5 basic functions added to the DCMHSCL1 application API. The routines can be found in the `etpu_app_dcmhscl1.c/.h` files. All DCMHSCL1 application API routines will be described in order and are listed below:

- Initialization function:

```
int32_t fs_etpu_app_dcmhscl1_init(
  dcmhscl1_instance_t * dcmhscl1_instance,
            uint8_t   PWM_master_channel,
            uint8_t   PWM_phaseA_channel,
            uint8_t   PWM_phaseB_channel,
            uint8_t   HD_phaseA_channel,
            uint8_t   HD_phaseB_channel,
            uint8_t   HD_phaseC_channel,
            uint8_t   SC_channel,
            uint8_t   CC_channel,
            uint8_t   ASDC_channel,
            uint8_t   PWM_phases_type,
            uint32_t  PWM_freq_hz,
            uint32_t  PWM_dead_time_ns,
            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,
            int32_t   SC_P_gain,
            int32_t   SC_I_gain,
            uint32_t  SC_ramp_time_ms,
            int32_t   CC_P_gain,
            int32_t   CC_I_gain,
            int32_t   dc_bus_current_range_ma,
            uint8_t   ASDC_polarity,
            uint24_t  ASDC_measure_time_us,
            uint32_t *ASDC_result_queue,
            uint8_t   ASDC_bit_shift,
            uint8_t   ASDC_queue_offset,
            uint32_t  ASDC_filter_time_constant_us)
```

- Change operation functions:

```
int32_t fs_etpu_app_dcmhscl1_enable(
  dcmhscl1_instance_t * dcmhscl1_instance,
              uint8_t   sc_configuration,
              uint8_t   cc_configuration)

int32_t fs_etpu_app_dcmhscl1_disable(
  dcmhscl1_instance_t * dcmhscl1_instance)

void fs_etpu_app_dcmhscl1_set_speed_required(
  dcmhscl1_instance_t * dcmhscl1_instance,
              int32_t   speed_required_rpm)
```

- Value return functions:

```
void fs_etpu_app_dcmhscl1_get_data(
  dcmhscl1_instance_t * dcmhscl1_instance,
      dcmhscl1_data_t * dcmhscl1_data)
```

4.3.1 int32_t fs_etpu_app_dcmhscl1_init(...)

This routine is used to initialize the eTPU channels for the DC motor with speed and current closed loops application. This function has the following parameters:

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_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 dcmhscl1_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_DCMHSCL1_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_DCMHSCL1_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.

- **SC_channel (uint8_t)** - This is the speed controller channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **CC_channel (uint8_t)** - This is the current controller channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **ASDC_channel (uint8_t)** - This is the analog sensing for DC motors (ASDC) 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_DCMHSCL1_SINGLE_CHANNELS, or FS_ETPU_APP_DCMHSCL1_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.
- **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_DCMHSCL1_REV_PERIOD, or
  FS_ETPU_APP_DCMHSCL1_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.
- **CC_P_gain (fract24_t)** - This is the current 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

- **CC_I_gain (fract24_t)** - This is the current 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
- **dc_bus_current_range_ma (int32_t)** - This is the DC-bus current measurement range in mA.
- **ASDC_polarity (uint8_t)** - This is the polarity to assign to the ASDC function. This parameter should be assigned a value of: FS_ETPU_APP_DCMHSCL1_ASDC_PULSE_HIGH or FS_ETPU_APP_DCMHSCL1_ASDC_PULSE_LOW.
- **ASDC_measure_time_us (uint24_t)** - Time from the first (triggering) edge to the second edge, at which the result queue is supposed to be ready in the DATA_RAM (in us). This value depends on the A/D conversion time and DMA transfer time.
- **ASDC_result_queue (uint32_t *)** - Pointer to the result queue. Result queue is an array of 16-bit words that contains the measured values.
- **ASDC_bit_shift (uint8_t)** - This parameter defines how to align data from the result queue into fract24 (or int24). This parameter should be assigned a values of:
  FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_8,
  FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_10,
  FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_12, or
  FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_16.
- **ASDC_queue_offset (uint8_t)** - Position of the I_DC_BUS sample in the result queue. Offset is defined in bytes.
- **ASDC_filter_time_constant_us (uint32_t)** - This the time constant of the filter which applies when processing the I_DC_BUS samples, in us.

4.3.2 int32_t fs_etpu_app_dcmhscl1_enable(...)

This routine is used to enable the generation of PWM signals and to start speed and current controllers. This function has the following parameters:

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.
- **sc_configuration (uint8_t)** - This is the required configuration of the SC. This parameter should be assigned a value of:
  FS_ETPU_APP_DCMHSCL1_SPEED_LOOP_OPENED, or
  FS_ETPU_APP_DCMHSCL1_SPEED_LOOP_CLOSED.
- **cc_configuration (uint8_t)** - This is the required configuration of the CC. This parameter should be assigned a value of:
  FS_ETPU_APP_DCMHSCL1_CURRENT_LOOP_OPENED, or
  FS_ETPU_APP_DCMHSCL1_CURRENT_LOOP_CLOSED.

4.3.3 int32_t fs_etpu_app_dcmhscl1_disable (dcmhscl1_instance_t * dcmhscl1_instance)

This routine is used to disable the generation of PWM signals and to stop speed and current controllers. This function has the following parameter:

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.

4.3.4 void fs_etpu_app_dcmhscl1_set_speed_required(...)

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

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.
- **speed_required_rpm (int32_t)** - This is the required motor speed in rpm.

4.3.5 void fs_etpu_app_dcmhscl1_get_data(...)

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

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.
- **dcmhscl1_data (dcmhscl1_data_t*)** - This is a pointer to dcmhscl1_data_t structure of application state data, which is updated.

4.4 eTPU Block Diagram

The eTPU functions used to drive the DC motor with speed and current closed loops 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.

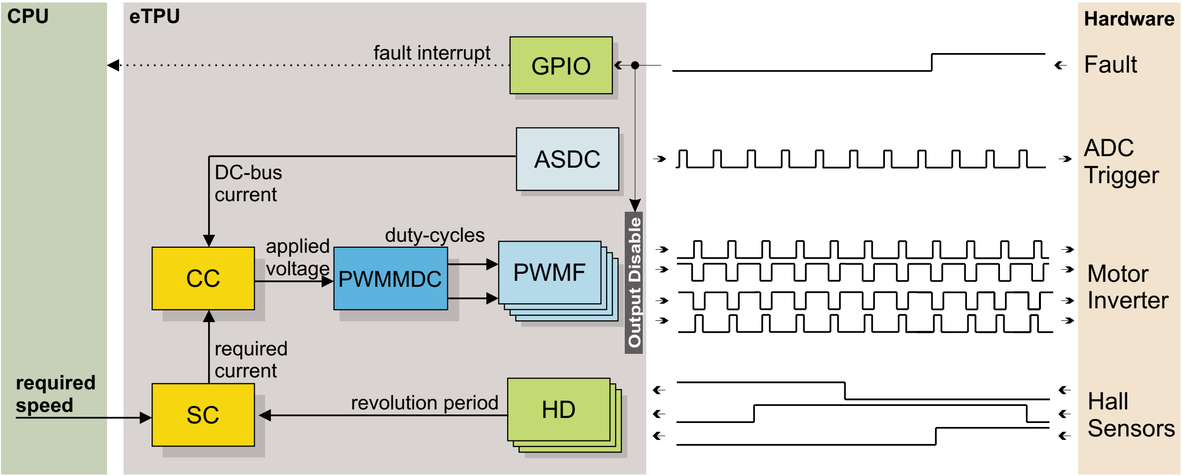


Figure 17. Block Diagram of eTPU Processing

4.4.1 PWM Generator (PWMMDC+PWMF)

The generation of PWM signals for motor-control applications with eTPU is provided by three eTPU functions:

- PWM - master for DC motors (PWMMDC)
- PWM - full range (PWMF)
- PWM - commuted (PWMC)

The PWM master for DC motors (PWMMDC) function calculates a PWM duty cycle and updates the three PWM phases. The phases may be driven either by the PWM full range (PWMF) function, which enables a full (0% to 100%) duty-cycle range, or by the PWM commuted (PWMC) function, which enables switching the phase ON and OFF. The PWMF function is used in the described application.

The PWMF function generates the PWM signals. The PWMMDC function controls two PWMF functions, two PWM phases, and does not generate any drive signal. The PWMMDC can be executed even on an eTPU channel not connected to an output pin (channels 16 to 32).

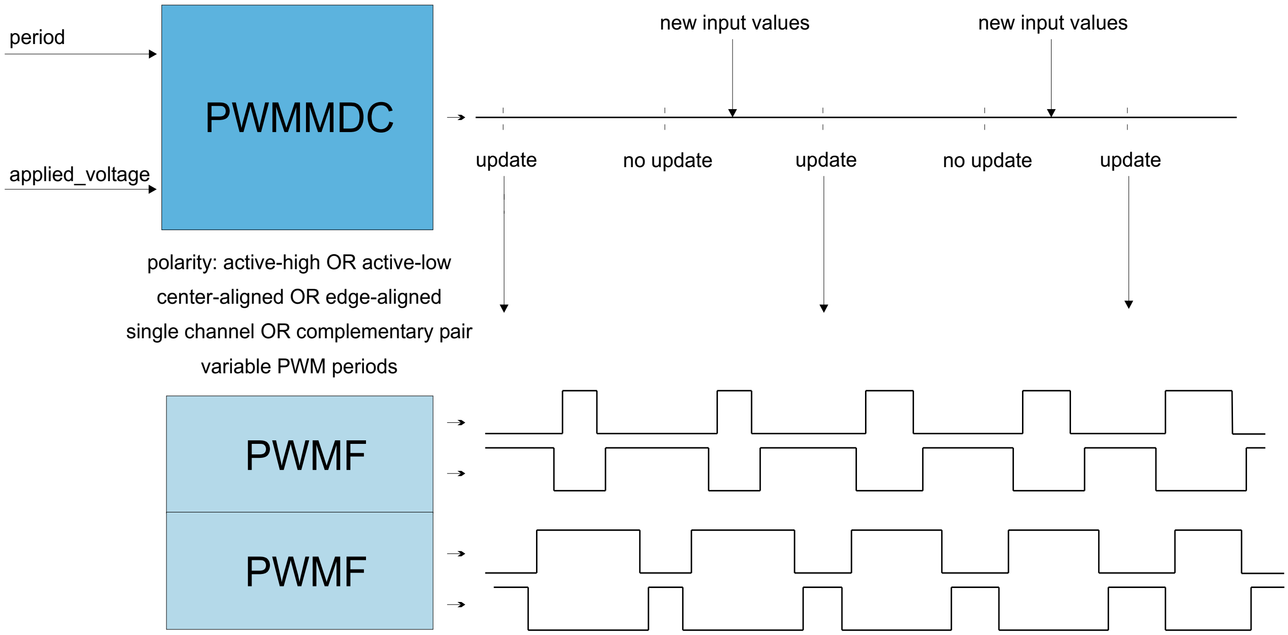


Figure 18. Functionality of PWMMDC+PWMF

For more details about the PWMMDC, PWMF, and PWMC eTPU functions, 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.

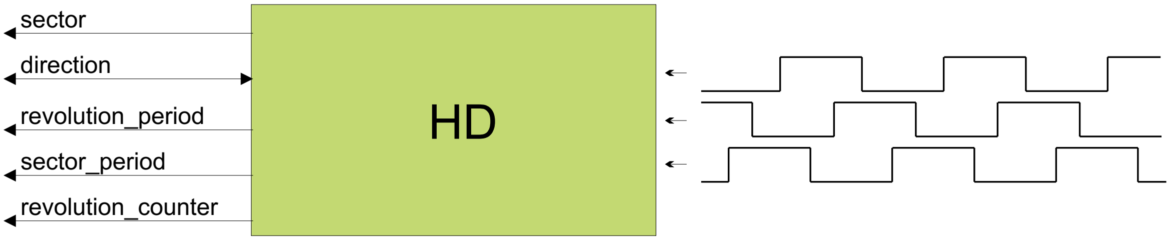


Figure 19. Functionality of HD

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

4.4.3 Speed Controller (SC)

The speed controller eTPU function is not intended to process input or output signals. Its purpose is to control another eTPU function’s input parameter. The SC function can be executed even on an eTPU channel not connected to an output pin. The SC function includes a general PID controller algorithm. The controller calculates its output based on two inputs: a measured value and a required value. The measured value (the actual motor speed) is calculated based on inputs provided by the HD function. The required value is an output of the speed ramp, whose input is a SC function parameter, and can be provided by the CPU or another eTPU function. In the motor-control eTPU function set, this function mostly provides the speed outer-loop.

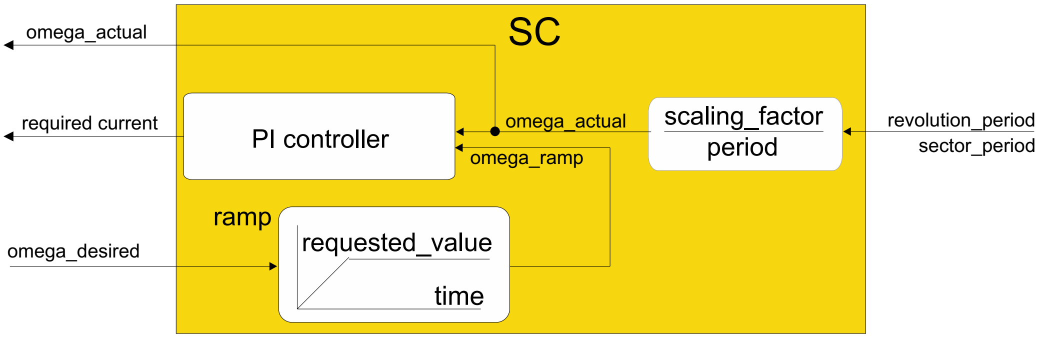


Figure 20. Functionality of SC

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

4.4.4 Current Controller (CC)

The current controller eTPU function is not intended to process input or output signals. Its purpose is to control another eTPU function's input parameter. The CC function can be executed even on an eTPU channel not connected to an output pin. The CC function includes a general PID controller algorithm. The controller calculates its output based on two inputs: a measured value, and a desired value. The measured value (the actual DC-bus current) is usually provided by the analog sensing for DC motors (ASDC) function, that preprocesses the measured analog values. The desired value is a CC function parameter, and can be provided by the CPU or another eTPU function. In the motor-control eTPU function set, this function mostly provides the current closed loop.

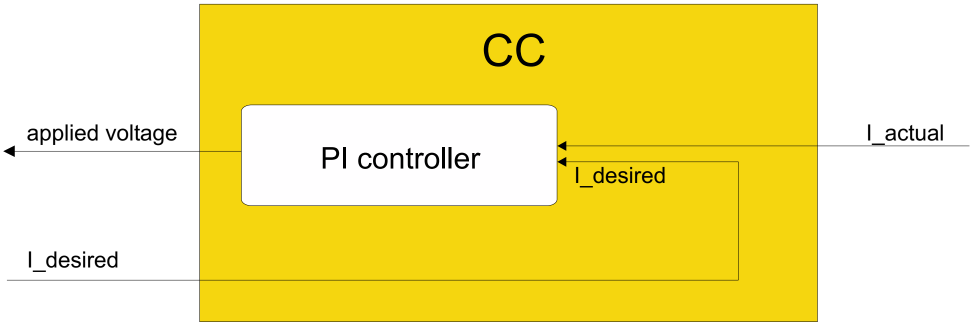


Figure 21. Functionality of CC

For more details about the CC eTPU function, refer to Reference 9.

4.4.5 Analog Sensing for DC Motors (ASDC)

The analog sensing for DC motors eTPU function (ASDC) is useful for preprocessing analog values that are measured by the AD converter and transferred to the eTPU data memory by DMA transfer. The ASDC function is also useful for triggering the AD converter and synchronizing other eTPU functions.

All the above mentioned ASDC features are utilized in the application. The ASDC is initialized to run in PWM synchronized mode, e.g. the first ASDC edge is synchronized with the beginning of the PWM period. Simultaneously, the ASDC manages to synchronize the SC function (outer loop controller) by generating the link to the SC channel every 16th ASDC period and to synchronize the CC function (inter loop controller) by generating the link to the CC channel every ASDC period.

The ASDC function preprocesses the DC-bus current analog value and passes the adjusted value as an input to the CC function. Processing of the DC-bus current analog value includes bit shifting and filtering.

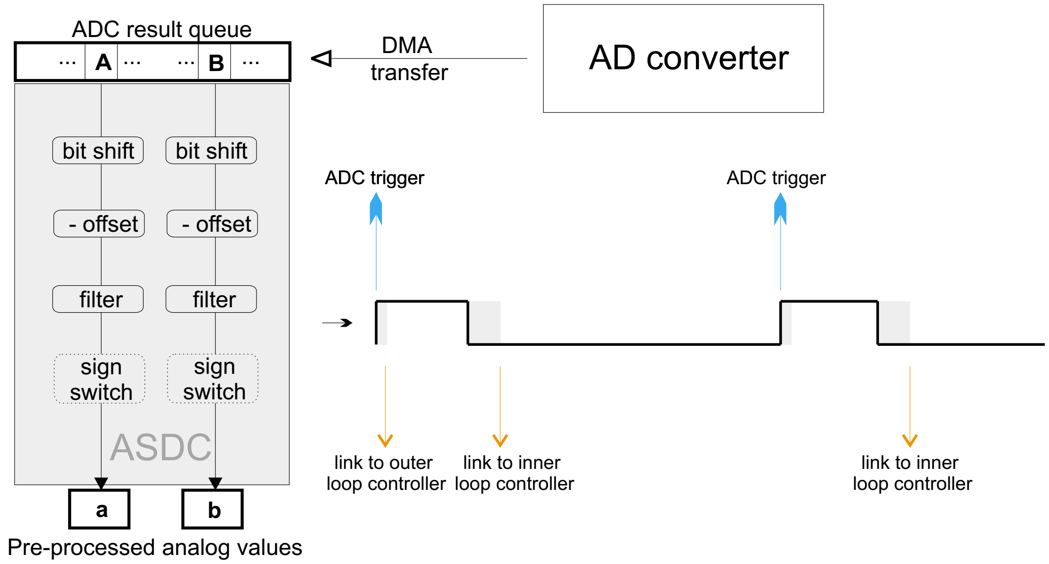


Figure 22. Functionality of ASDC

In order to ensure periodic sampling of the DC-bus current and the quick transfer of the measured data from the AD converter to the ETPU DATA RAM, several peripheral modules are used (see Figure 23):

- An external analog to digital converter (AD7928BRA) is used for sampling of the DC-bus current analog value.
- A queued serial peripheral interface (QSPI) module is used to interface the AD converter. It enables initialization of the AD converter control registers, triggering the AD converter, and receiving measured samples of DC-bus current.
- Direct memory access timer 0 (DMA Timer 0) is used for the DMA request assertion at the time the measured sample is ready in the QSPI receive RAM. DMA timer 0 is configured to run in reference compare mode, asserting the DMA request a defined time after the AD converter triggering. This sampling time is defined in the DTRR0 register of the DMA timer.
- 4 direct memory access (DMA) channels are used as follows:
  - — DMA channel 3 is used for the transfer of one auxiliary byte (0x80) from the ETPU DATA RAM to the QDLYR register of the QSPI module. This operation triggers the QSPI transfer and the consequent AD converter sampling. The DMA channel 3 transfer is initiated by the DMA request generated by the ASDC eTPU function. When the transfer is complete, the link to DMA channel 1 is made causing the DMA request assertion on DMA channel 1.
  - — DMA channel 1 is used for the transfer of one auxiliary byte (0x80) from the ETPU DATA RAM to the DTCN0 register of DMA timer 0 module. This operation clears the DTCN0 register and thus triggers the DMA timer 0 free-running timer counting. The DMA channel 1 transfer is initiated by the link from DMA channel 3.
  - — DMA channel 0 is used for the transfer of the measured sample (2 bytes) from the QDR register of the QSPI module to the ETPU DATA RAM. The DMA channel 0 transfer is initiated by a DMA request generated by DMA timer 0, at the time the counter value reaches the reference value. When the transfer is complete, the link to DMA channel 2 is made,

causing the DMA request assertion on DMA channel 2.

— DMA channel 2 is used for the transfer of one auxiliary byte (0x10) from the ETPU DATA RAM to the QAR register of the QSPI module. This operation sets the pointer in the QSPI to the start of the receive RAM, thus enabling the same sample to be reading in the next cycle. The DMA channel 2 transfer is initiated by the link from DMA channel 0.

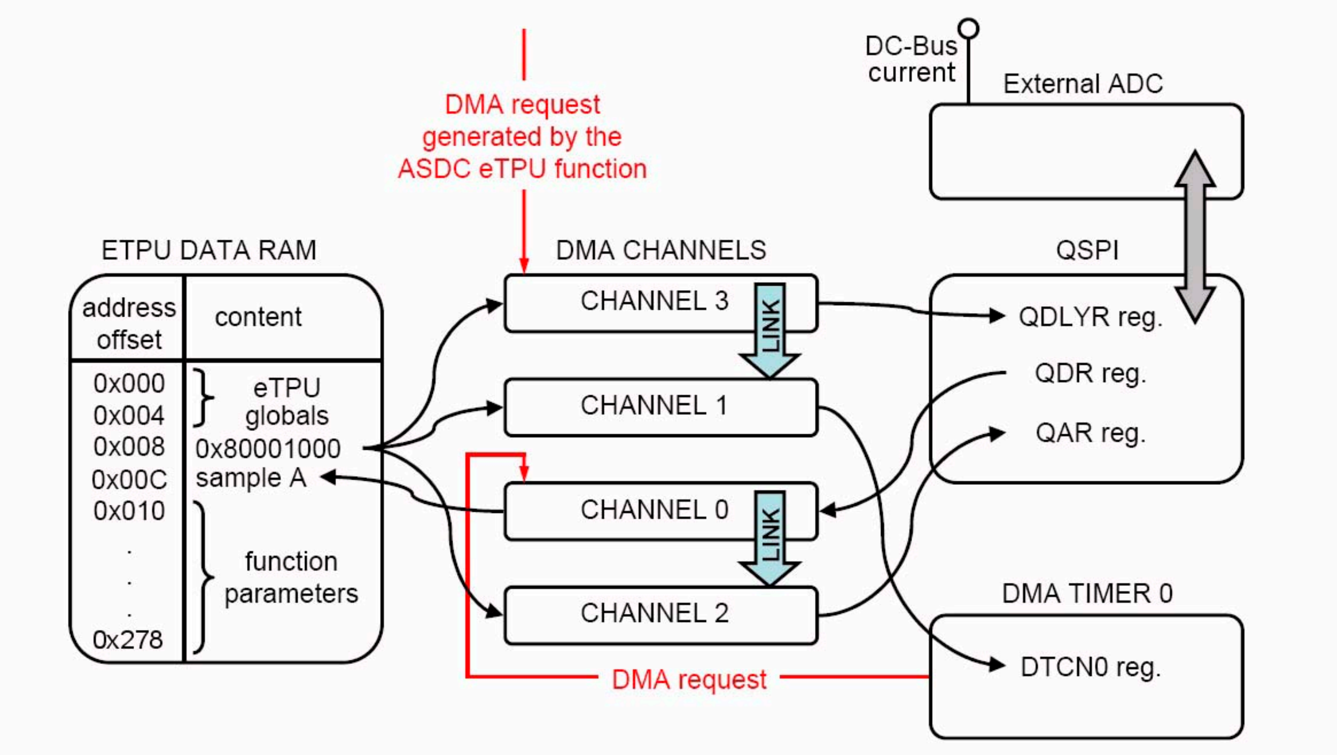


Figure 23. DC-bus current sample transfer to the ETPU DATA RAM

As each of the DMA transfers decrements the DMA BCRx registers by one (DMA channels 1, 2, 3), or by two (DMA channel 0), re-initializing all these registers must be done regularly to enable the continuous operation of the DMA channels in this way (see 4.1.2).

4.4.6 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 11.

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.

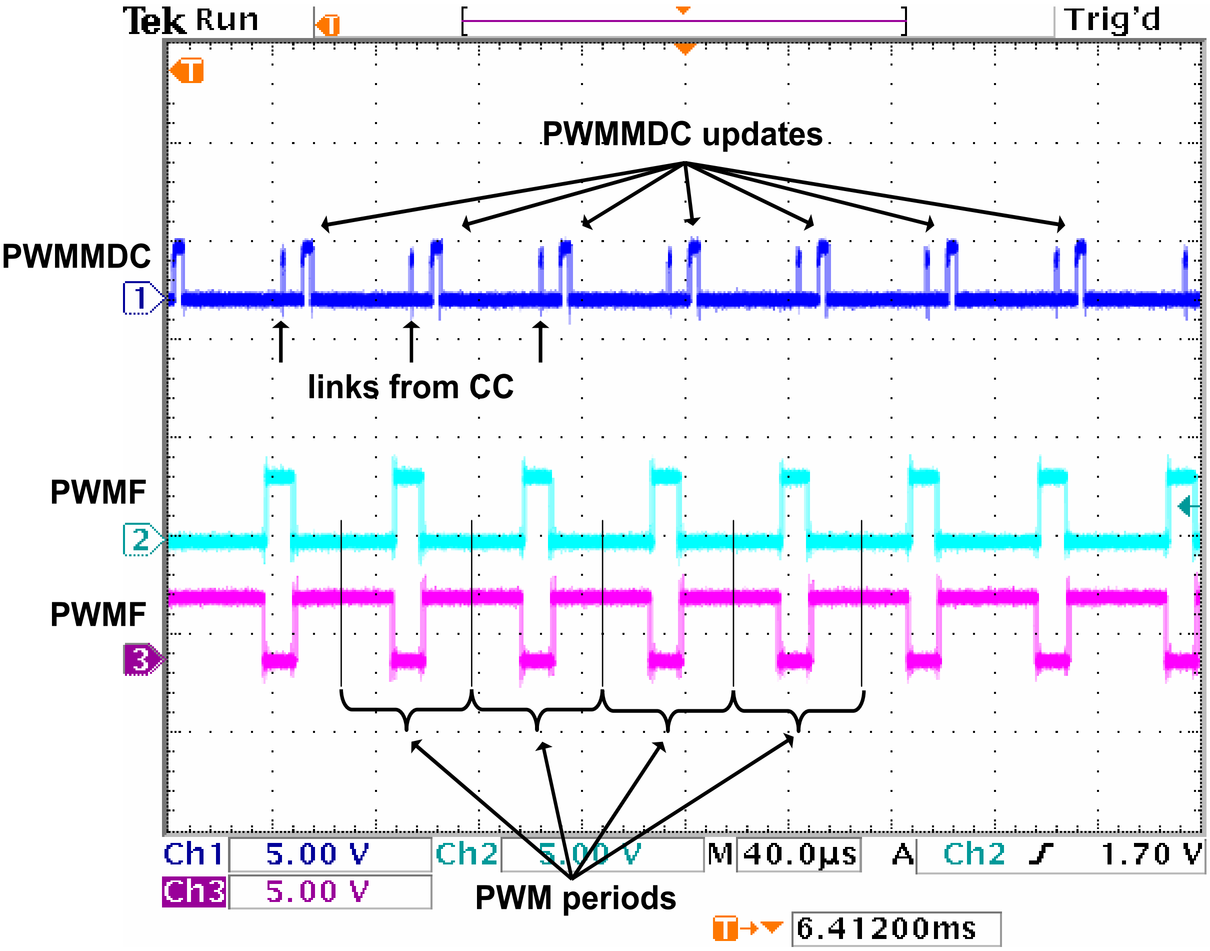


Figure 24. Oscilloscope Screen-shot and Explanation of PWMMDC Timing

Figure 24 and Figure 25 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 SC, CC and PWMMDC activity pulses, which determines the channel service latency ranges.

In Figure 24, signals 2 (cyan) and 3 (pink) are PWM signals of one phase. It is a complementary pair of centre-aligned PWM signals. The base channel (2) is of active-high polarity, while the complementary channel (3) is active-low. The PWM period is 50µs, which corresponds to a PWM frequency of 20kHz.

Signal 2 (cyan) is generated by the PWM master for DC motors (PWMMDC) eTPU function. Its pulses determine the activity of the PWMMDC. The narrow PWMMDC pulses occur after each CC activity and they determine the service time of an CC request to update the new value of applied motor voltage. The wide pulses occur each PWM period and they determine the PWM update. A new value of applied motor voltage is processed during the PWM update.

The `fs_etpu_pwmmdc_init_3ph` function parameter update_time enables the user to adjust the position of the PWM update pulse relative to the PWM period frame. The activity pulse has a scheduled update_time prior to the end of the period frame, so that the update is finished by the end of the period frame, even in the worst case latency. Reference 8 describes how to set the update_time value.

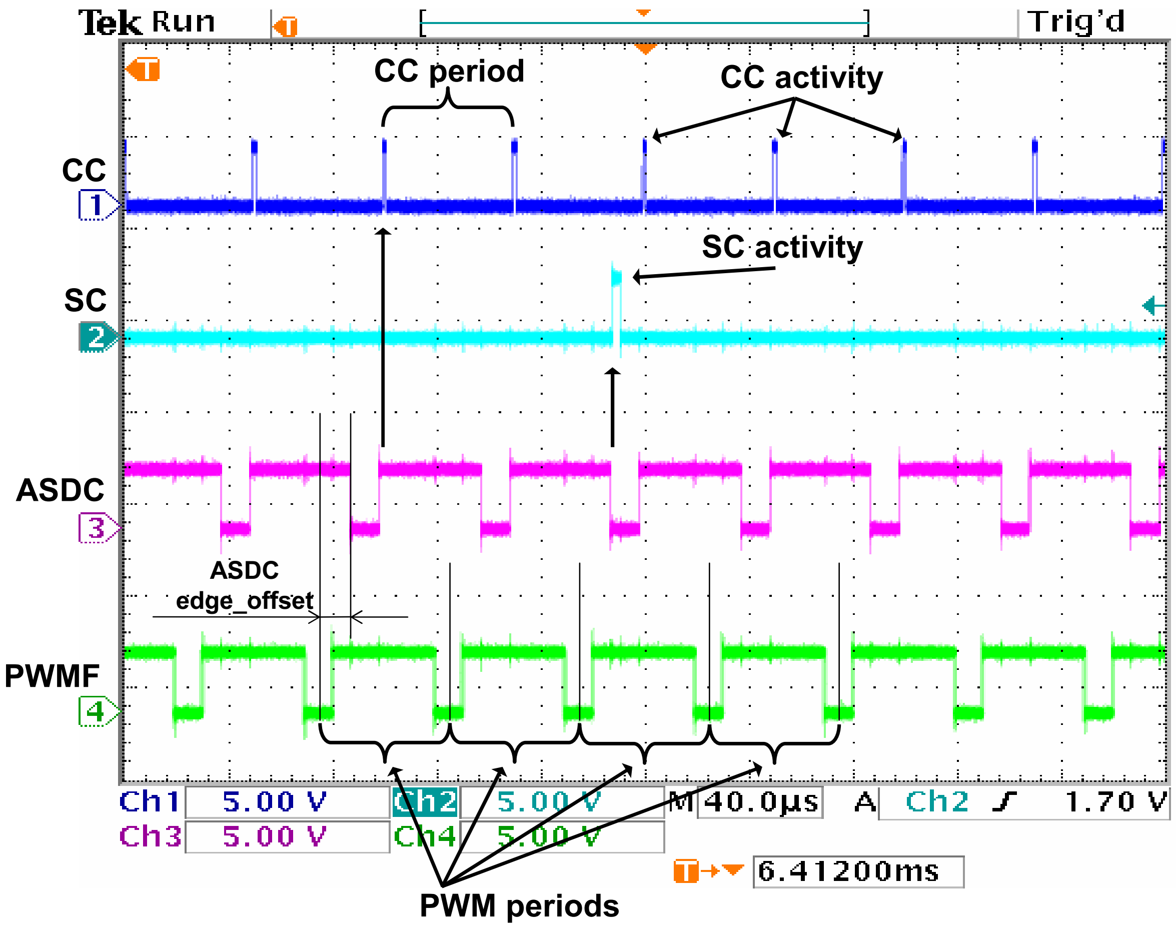


Figure 25. Oscilloscope Screen-shot and Explanation of ASDC, SC, and CC Timing

Figure 25 explains the timing of ASDC, SC, and CC eTPU functions. Signal 4 (green) is the PWM signal of one base channel, as in Figure 24. Signal 3 (pink) is generated by the ASDC eTPU function. The ASDC function triggers the AD converter by generating a DMA request on a high-low edge (active low polarity of ASDC) and simultaneously sending the link to the SC channel every 16th ASDC period (see Figure 25). The position of the ASDC first edge is synchronized with the beginning of the PWM period. The time between the PWM period beginning and the ASDC first edge equals to one-quarter of the PWM period. The ASDC pulse width determines the time necessary to sample the DC-bus current and to transfer this sampled value to the eTPU data memory. ASDC starts measured sample preprocessing at the time of the second edge when a sample is supposed to be ready in the eTPU data memory. Immediately after the DC-bus current sample preprocessing, ASDC function sends the link to the CC channel to start PI controller calculation. This way the CC synchronization is ensured.

Signal 2 (cyan) is generated by the speed controller (SC) eTPU function. Its pulses determine the activity of the SC. The 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 SC-PI controller algorithm. The SC output - the new value of desired current - is passed to the current controller. This calculation is performed periodically at a 1250Hz rate, which is every 16th PWM period.

Signal 1 (blue) is generated by the current controller (CC) eTPU function. Its pulses determine the activity of the CC. The pulse width determines the time necessary to apply the CC-PI controller algorithm. The CC output - the new value of applied motor voltage - is passed to the PWM generator. This calculation is performed periodically at a 20kHz rate, which is every PWM period.

The signals from the optical sensors come asynchronously with the PWM periods. The Hall decoder (HD) eTPU function processes these signals transitions and provides the calculated data to the speed controller eTPU function.

5 Implementation Notes

5.1 Scaling of Quantities

The DC 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

Both the speed controller and the current controller PI 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 speed controller (SC) eTPU function calculates the angular motor speed using a revolution period measured by the Hall decoder (HD) eTPU function. Optionally, the speed controller 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 used DC motor is a 8 pole-pair motor. Hence, the mechanical revolution period is a period of eight 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.

6 Microprocessor Usage

Table 2 shows how much memory is needed to run the application.

Table 2. Memory Usage in Bytes

Memory	Available	Used
FLASH (external)	2M	30 128
RAM	6K	5 828
eTPU code RAM	6K	6 140
eTPU data RAM	1.5K	632

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

- According to Reference 8, the maximum eTPU load produced by PWM generation is 854 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 speed controller calculation takes 232 eTPU cycles. The calculation is performed every 16th PWM period.
- According to Reference 6, the processing of one Hall signal transition takes 56 eTPU cycles. The Hall signal transitions come asynchronously to the PWM periods. Six transitions are processed per one electrical motor revolution.
- According to Reference 10, the ASDC maximum eTPU load takes 42 + 80 eTPU cycles (both the first and then the second edge processing is performed). The ASDC function processing is executed every PWM period.
- According to Reference 9, the current controller calculation takes 138 eTPU cycles. The calculation is performed every PWM period.
- 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, the current controller calculation, ASDC processing, 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 3 shows the eTPU module time load in several typical situations.

Table 3. eTPU Time Load

Situation	Average Time Load [%]	Peak Time Load Within PWM Period [%]
Motor Speed 100 RPM	30.10	37.39
Motor Speed 1000 RPM	30.12	37.39

7 Summary and Conclusions

This application note provides the user with a description of the demo application DC motor with speed and current 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 4. References

1. *MCF5235 Reference Manual*, MCF5235RM
2. *M523xEVB User's Manual*, M5235EVBUM
3. *33395 Evaluation Motor Board Designer Reference Manual* DRM33395/D
4. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER"
5. *Enhanced Time Processing Unit Reference Manual*, ETPURM
6. "Using the Hall Decoder (HD) eTPU Function," AN2841
7. "Using the Speed Controller (SC) eTPU Function," AN2843
8. "Using the DC Motor Control PWM eTPU Functions," AN2480
9. "Using the Current Controller (CC) eTPU Function," AN2844
10. "Using the Analog Sensing for DC Motors (ASDC) eTPU Function," AN2846
11. "Using the General Purpose I/O eTPU functions (GPIO)," AN2850
12. "Using the DC Motor Control eTPU Function Set (set3)," AN2958
13. eTPU Graphical Configuration Tool, http://www.freescale.com, search keyword "ETPUGCT"
14. "DSP56F80x MC PWM Module in Motor Control Applications," AN1927

THIS PAGE INTENTIONALLY LEFT BLANK

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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.