Freescale Semiconductor
Application Note

AN3206
Rev. 1, 03/2012

Permanent Magnet Synchronous Motor Vector Control, Driven by eTPU on MPC5500

Covers MPC5500, MPC5600, 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 3-phase Permanent Magnet Synchronous Motor (PMSM) speed and torque vector control drive based on Freescale's Power Architecture MPC5500 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 handles the motor control processing, eliminating the microprocessor overhead for other duties.

PMSMs are very popular in a wide array of applications. Compared with a DC motor, the PMSM misses a commutator and so it is more reliable than the DC motor. The PMSM also has advantages when compared to an AC induction motor. PMSM generate the rotor magnetic flux with rotor magnets, so achieve higher efficiency. Therefore PMSM are used in electric and hybrid vehicles, high-end white goods (refrigerators, washing machines, dishwashers, etc.), high-end pumps, fans and in other appliances, which require high reliability and efficiency.

The concept of the application is to create a vector control PMSM driver with optional speed closed-loop, using a quadrature encoder. It serves as an example of a

Table of Contents

freescale semiconductor

PMSM vector 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 Freescale AC 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.

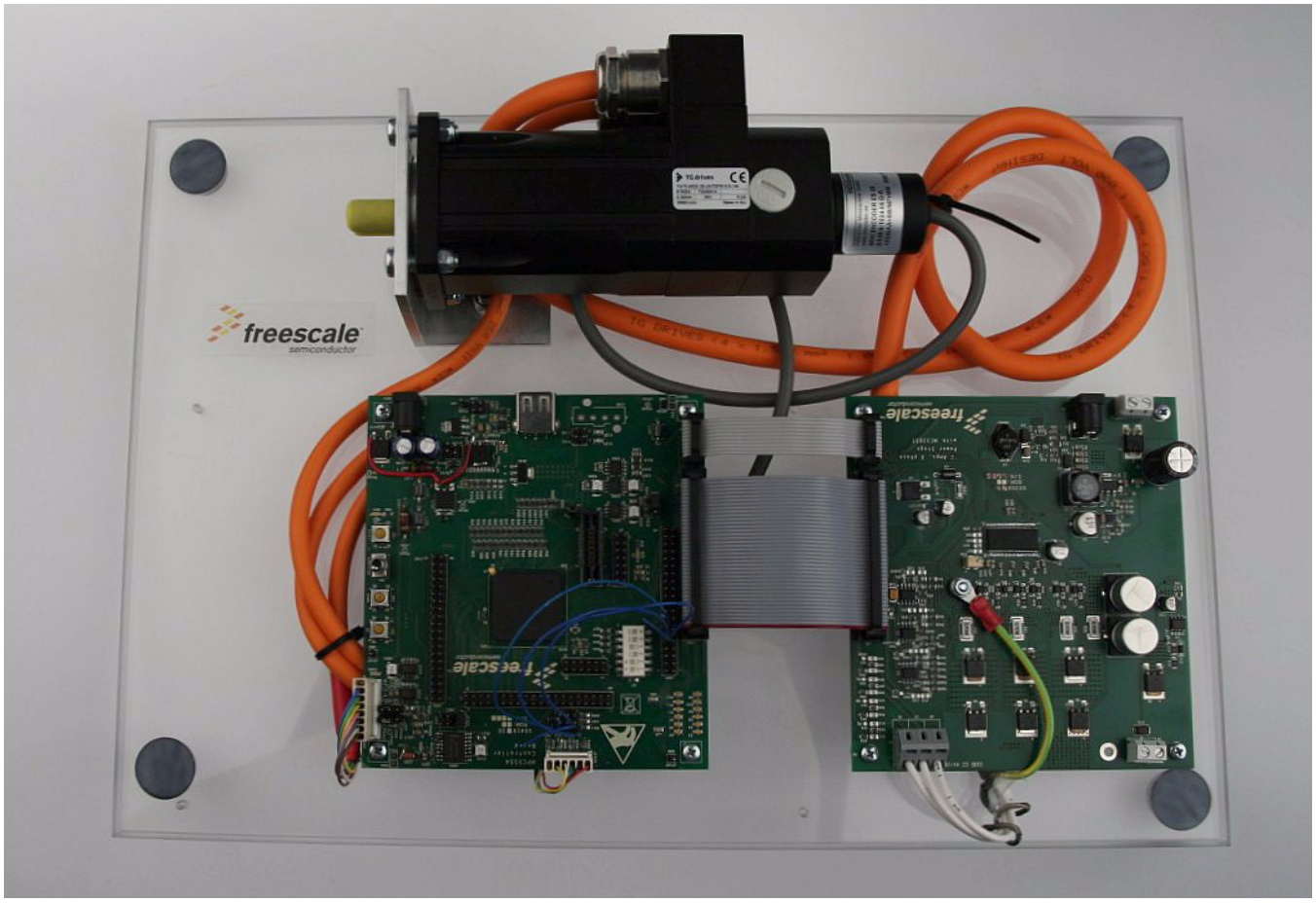


Figure 1. MPC5554 Controller Board, Power Stage with MC33937, and TGT2-0032-30-24 PMSM

1 Power Architecture MPC5554 and eTPU Advantages and Features

1.1 Power Architecture MPC5554 Microprocessor

The MPC5554 microcontroller, which is used in this application, belongs to a family of powertrain microcontrollers based on the Power Architecture Book E architecture. The application can be easily ported to any other MPC5500 or MPC5600 device featuring the eTPU module. The eTPU handles all real-time motor control tasks, minimizing CPU usage. Besides the eTPU, the following peripheral modules are utilized:

- Enhanced Direct Memory Access controller (eDMA)
- Enhanced Queued Analog-to-Digital Converter (eQADC)
- Deserial Serial Peripheral Interface (DSPI) modules
- Enhanced Serial Communication Interface (eSCI) modules

For more information about MPC5554, 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 on the MPC5554 microcontroller has two engines with up to 32 timer channels for each. In addition it has 16 Kbytes of code memory and 3 Kbytes of data memory that stores software modules downloaded at boot time and that can be mixed and matched as required for any specific 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 7.

2 Target Motor Theory

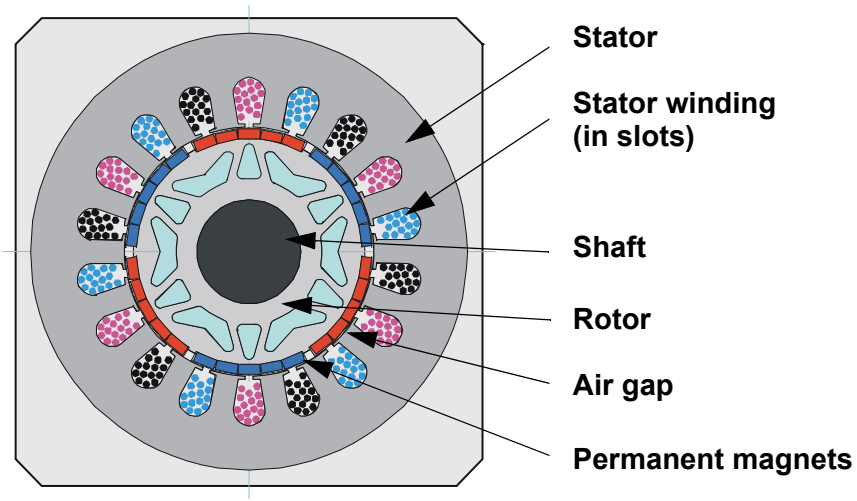


Figure 2. Permanent Magnet AC Machine - Cross Section

Permanent magnet ac (PMAC) machines provide automotive actuator designers with a unique set of features and capabilities. There are two principal classes of permanent magnet ac machines, the first type

are sinusoidally excited, so called permanent magnet synchronous motors (PMSM), and the second type are trapezoidaly excited machines, so called brushless DC (BLDC) motors. The conceived construction differences are that while stator windings of trapezoidal PMAC machines are concentrated into narrow-phase pole, the windings of a sinusoidal machine are typically distributed over multiple slots in order to approximate a sinusoidal distribution. These differences in construction are reflected in their corresponding motion characteristics as well. This implies the consequence that the first type of PMAC provides sinusoidal back-electromotive force (back-EMF) generation, and the second type provides trapezoidal back-EMF.

The PMSM machines enjoy unique advantages of unsurpassed efficiency and power density characteristics which are primarily responsible for their wide appeal. On the other hand, PMSM machines are synchronous which certainly requires accompanying power electronics, but it also provides the basis for achieving high-quality actuator control. The torque ripple associated with sinusoidal PMAC (PMSM) machines is generally less than that developed in trapezoidal machines, providing one the reasons that sinusoidal motors are preferred in high performance motion control application such as electro-mechanical braking. This application note targets the PMSM only.

2.1 Digital Control of PMSM

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

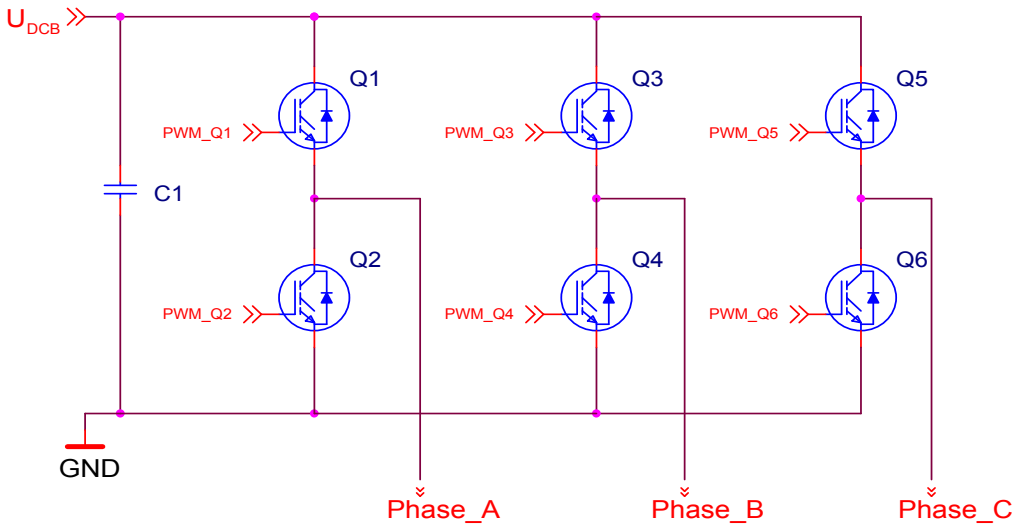


Figure 3. 3-Phase Power Stage

The inverter consists of three half-bridge units where the upper and lower switches are controlled complementarily, meaning when the upper one is turned on, the lower one must be turned off, and vice versa. Because the power device's turn-off time is longer than its turn-on time, some dead time must be inserted between turning off one transistor of the half-bridge, and turning on its complementary device. The output voltage is mostly created by a Pulse Width Modulation (PWM) technique. The 3-phase voltage waves are shifted 120^{o} to one another, thus a 3-phase motor can be supplied.

2.2 Vector Control of PMSM

Vector Control is an elegant method of controlling the permanent magnet synchronous motor (PMSM), where field oriented theory is used to control space vectors of magnetic flux, current, and voltage. It is possible to set up the co-ordinate system to decompose the vectors into an electro-magnetic field generating part and a torque generating part. Then the structure of the motor controller (Vector Control controller) is almost the same as for a separately excited DC motor, which simplifies the control of PMSM. This Vector Control technique was developed in the past especially to achieve similar excellent dynamic performance of PMSM.

As explained in Figure 4, the choice has been made of a widely used current control with an inner position closed loop. In this method, the decomposition of the field generating part and the torque generating part of the stator current allows separate control of the magnetic flux and the torque. In order to do so, we need to set up the rotary co-ordinate system connected to the rotor magnetic field. This co-ordinate system is generally called “d-q reference co-ordinate system”. For more information, refer to Reference 12.

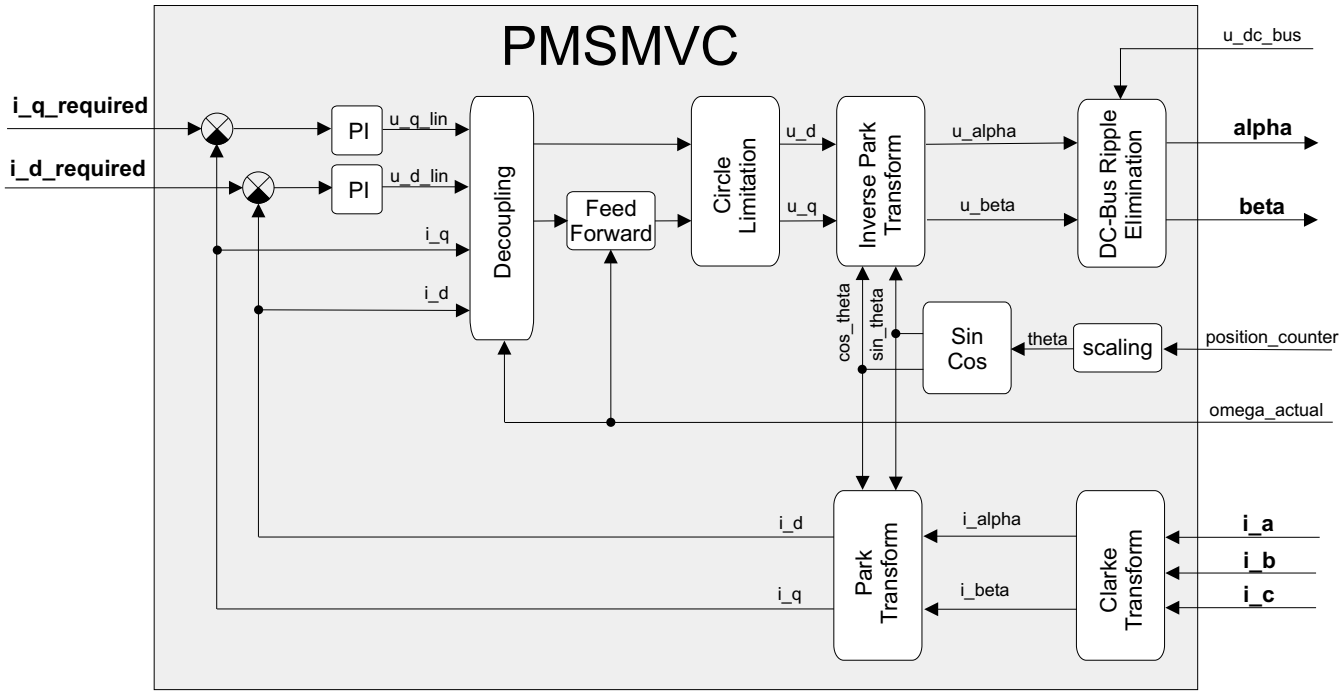


Figure 4. PMSM Vector Control Current Loop Block Diagram

2.3 Quadrature Encoder

The PMSM motor application uses the quadrature encoder for rotor position sensing. The quadrature encoder output consists of three signals. Two phases, A and B, represent the rotor position, and an index pulse defines the zero position. All quadrature encoder signals are depicted in Figure 5.

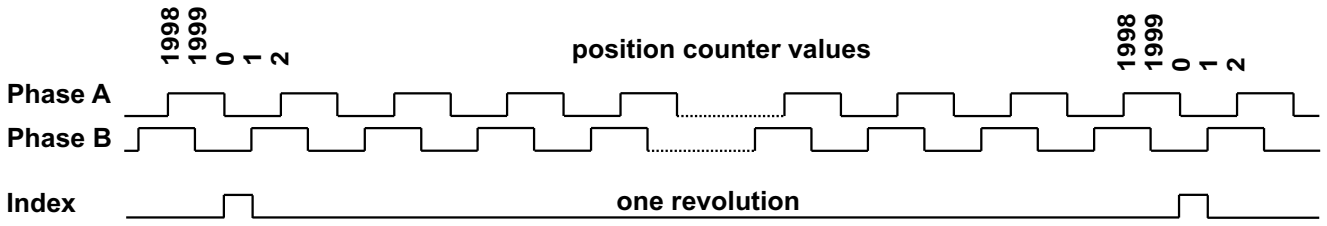


Figure 5. Quadrature Encoder Output Signals

2.3.1 Position Alignment

Since the quadrature encoder doesn't give the absolute position, we need to know the exact rotor position before the motor is started. One possible and very easily implemented method is the rotor alignment to a predefined position. The motor is powered by a defined static voltage pattern and the rotor aligns to the predefined position. This alignment is done during the motor start up.

3 System Concept

3.1 System Outline

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

- Voltage control of a PMSM using quadrature encoder ES28-6-1024-05-D-R
- Targeted at MPC5554 Controller Board (MPC5554CB), 10 A, 3-Phase Power Stage with MC33937, and TGT2-0032-30-24 Permanent Magnet Synchronous Motor (PMSM)
- Control technique incorporates:
  - PMSM vector control with optional speed-closed loop
  - Both directions of rotation
  - 4-quadrant operation
  - Rotor alignment to the start position
  - Minimum speed of 50 RPM
  - Maximum speed of 3000 RPM
- Manual interface (start/stop switch, up/down push button control)
- FreeMASTER control interface (speed set-up, speed control/torque control choice)
- FreeMASTER monitor
  - FreeMASTER graphical control page (required speed, required torque, actual motor speed, actual torque, start/stop status, fault status)
  - FreeMASTER control scope (observes required and actual speeds and torques, 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
- Error handling

3.2 Application Description

A standard system concept is chosen for the motor control function (see Figure 6). The system incorporates the following hardware:

- MPC5554 controller board (MPC5554CB)
- 10 A, 3-phase power stage with MC33937
- TGT2-0032-30-24 PMSM
- Quadrature encoder Mini Encoder ES28 (ES28-6-1024-05-D-R)
- Power supply 18 V DC, 4 A

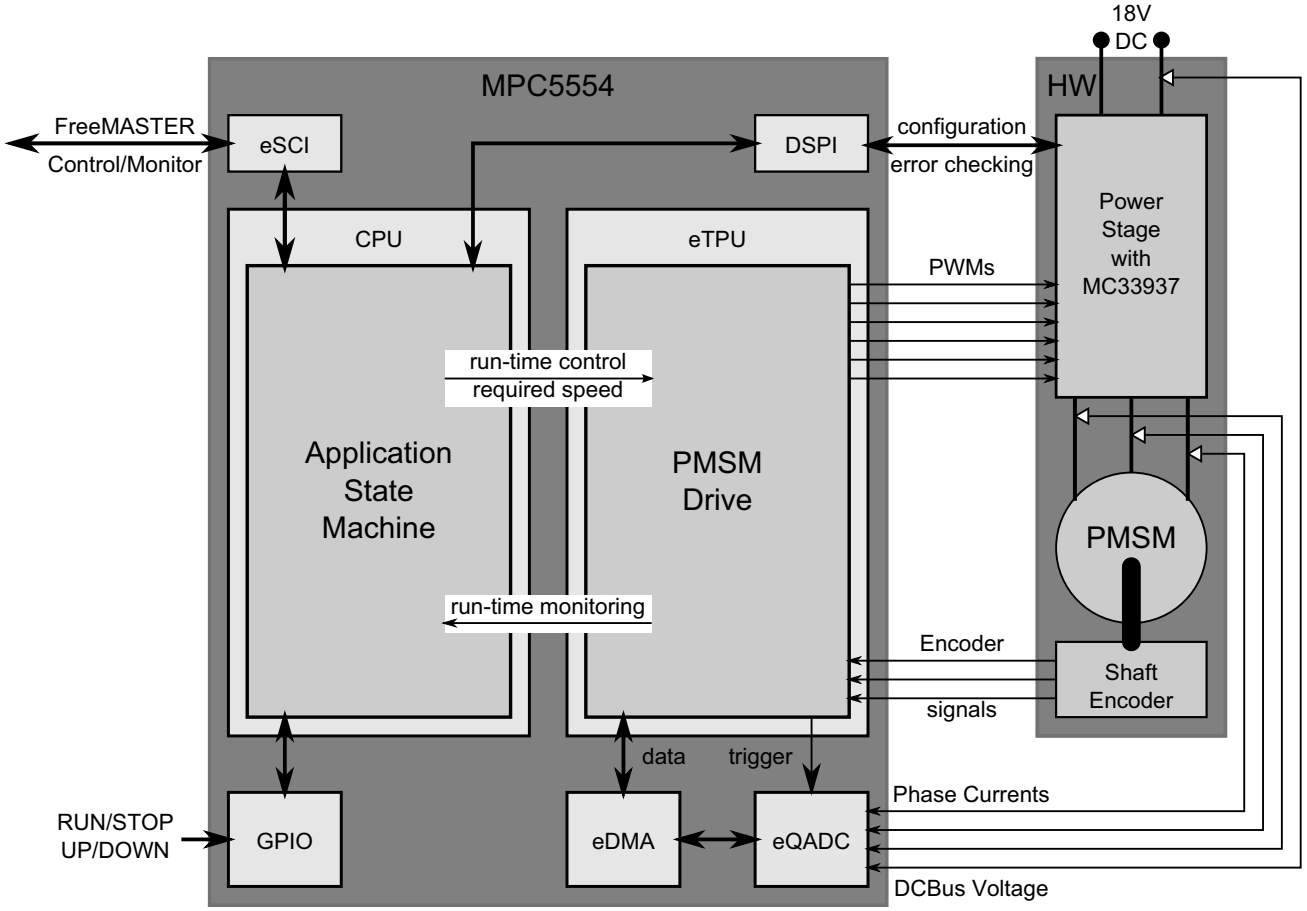


Figure 6. System Concept

The eTPU module runs the main control algorithm. The 3-phase PWM output signals for a 3-phase inverter are generated, using the vector control algorithm, according to feedback signals from the quadrature

encoder, the actual values of phase currents, and the input variable values provided by the microprocessor CPU. The DC-bus voltage and phase currents are sampled by an on-chip analog-to-digital converter (eQADC module). The transfer of measured samples from eQADC to eTPU data RAM is provided by the eDMA module.

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

The CPU performs the following tasks:

- Handles the application state machine and sets the required speeds to the eTPU.
- Monitors the eTPU operation, and enables the FreeMASTER recorder to sample eTPU internal variables with a sampling period of 50 µs.

The eTPU performs the following tasks:

- Six eTPU channels (PWMF) are used to generate PWM output signals.
- Three eTPU channels (QD) are used to process quadrature encoder signals.
- One eTPU channel (BC) is used for controlling the DC-bus break.
- One eTPU channel (ASAC) is used to trigger the on-chip AD convertor, and preprocess the sampled values. ASAC uses the eDMA to transfer eQADC conversion commands from eTPU data RAM into eQADC and to transfer the results of conversion rom eQADC into eTPU data RAM.
- One eTPU channel (PWMMAC) is internally used to synchronize the PWM outputs and calculate space vector modulation, based on applied motor voltage vector Alpha-Beta coordinates.
- One eTPU channel (SC) is internally used to calculate the actual motor speed, and control a speed closed loop in case of speed vector control. The actual motor speed is calculated based on the QD position counter and QD last edge time. The required speed is 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 PMSMVC eTPU function as a newly corrected value of the required motor torque.
- One eTPU channel (PMSMVC) is internally used to calculate the current vector control closed loop. It takes the values of actual phase currents from ASAC, the actual rotor position from QD, and the actual motor speed from SC. The phase currents are transferred to Alpha-Beta and D-Q coordinate system, using Clark and Park transformations. One PI control algorithm processes the error between the required and actual D-current, and another one the error between the required and the actual Q-current. The required value of the D-current, which is the flux controlling current, is set to zero. The required value of the Q-current, which is the torque controlling current, is either set directly by the CPU (torque vector control), or provided by the SC output (speed vector control). The PI controller outputs create the motor applied voltage vector, in D-Q coordinate system. It is transformed back to Alpha-Beta coordinate system and passed to PWM generator.

3.2.1 User Interface

The application is interfaced by the following:

- Start/stop switch and up/down buttons on the MPC5554CB.

- FreeMASTER running on a PC connected to the MPC5554CB via USB cable. The MPC5554CB includes USB-to-serial gate.

The start/stop switch, either on the board or on teh FreeMASTER Control Page, affects the application state and enables and disables the PWM phases. When the switch is on, the motor speed can be controlled either by the Up and Down buttons, or by the FreeMASTER graphical needle. The FreeMASTER also displays real-time values of application variables and their time behavior using scopes and recorders.

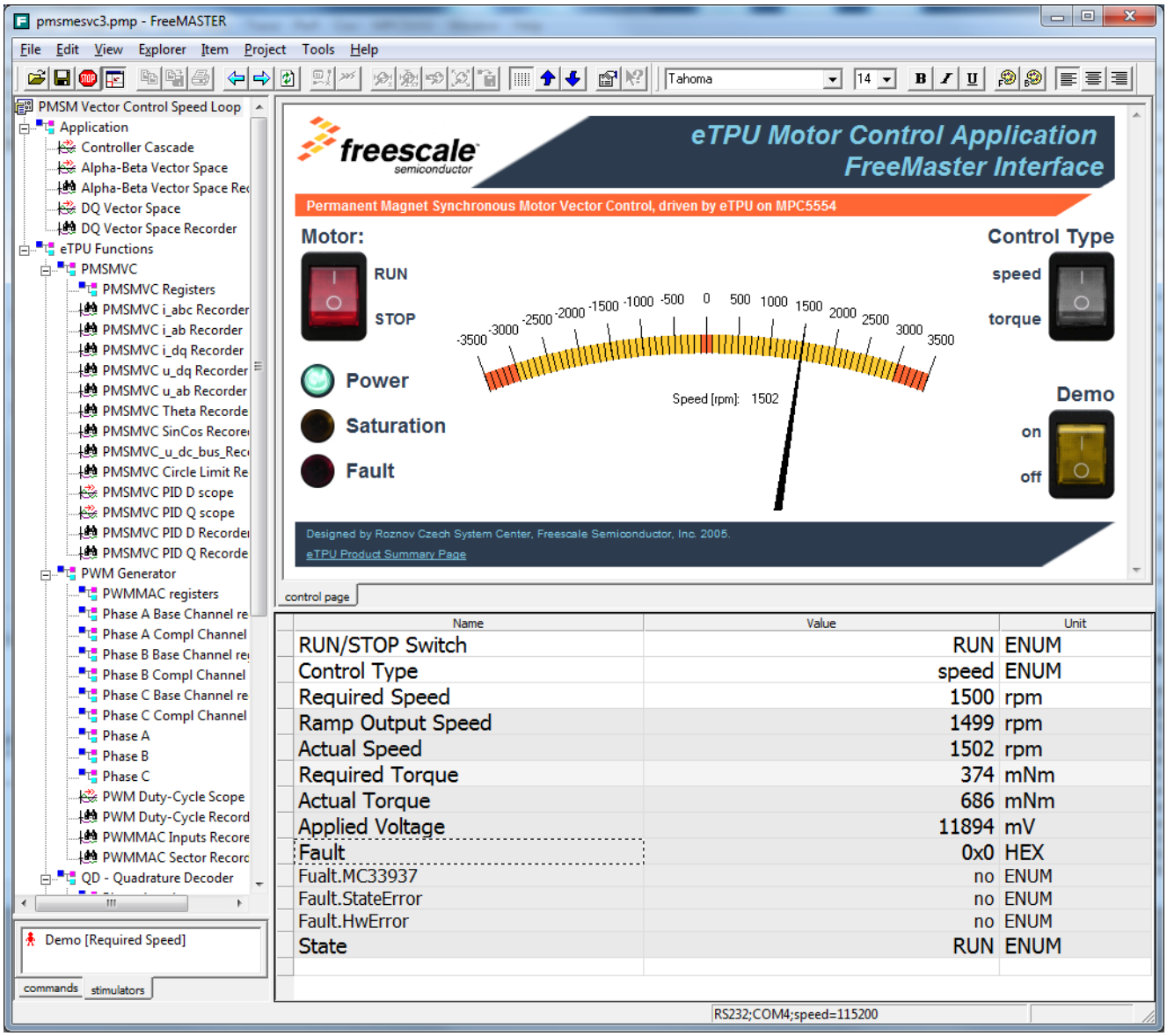


Figure 7. FreeMASTER Control Page

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 MPC5554CB via a USB 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.

The FreeMASTER application can be downloaded from http://www.freescale.com. For more information about FreeMASTER, refer to Reference 6.

3.3 Hardware Implementation and Application Setup

As previously stated, the application runs on the Power Architecture MPC5554 microprocessor using the following:

- MPC5554 Controller Board (MPC5554CB)
- 10 A, 3-Phase Power Stage with MC33937
- TGT2-0032-30-24 PMSM
- Quadrature encoder Mini Encoder ES28 (ES28-6-1024-05-D-R)
- Power Supply 18 V DC, 4 A

Figure 8 shows the connection of these parts.

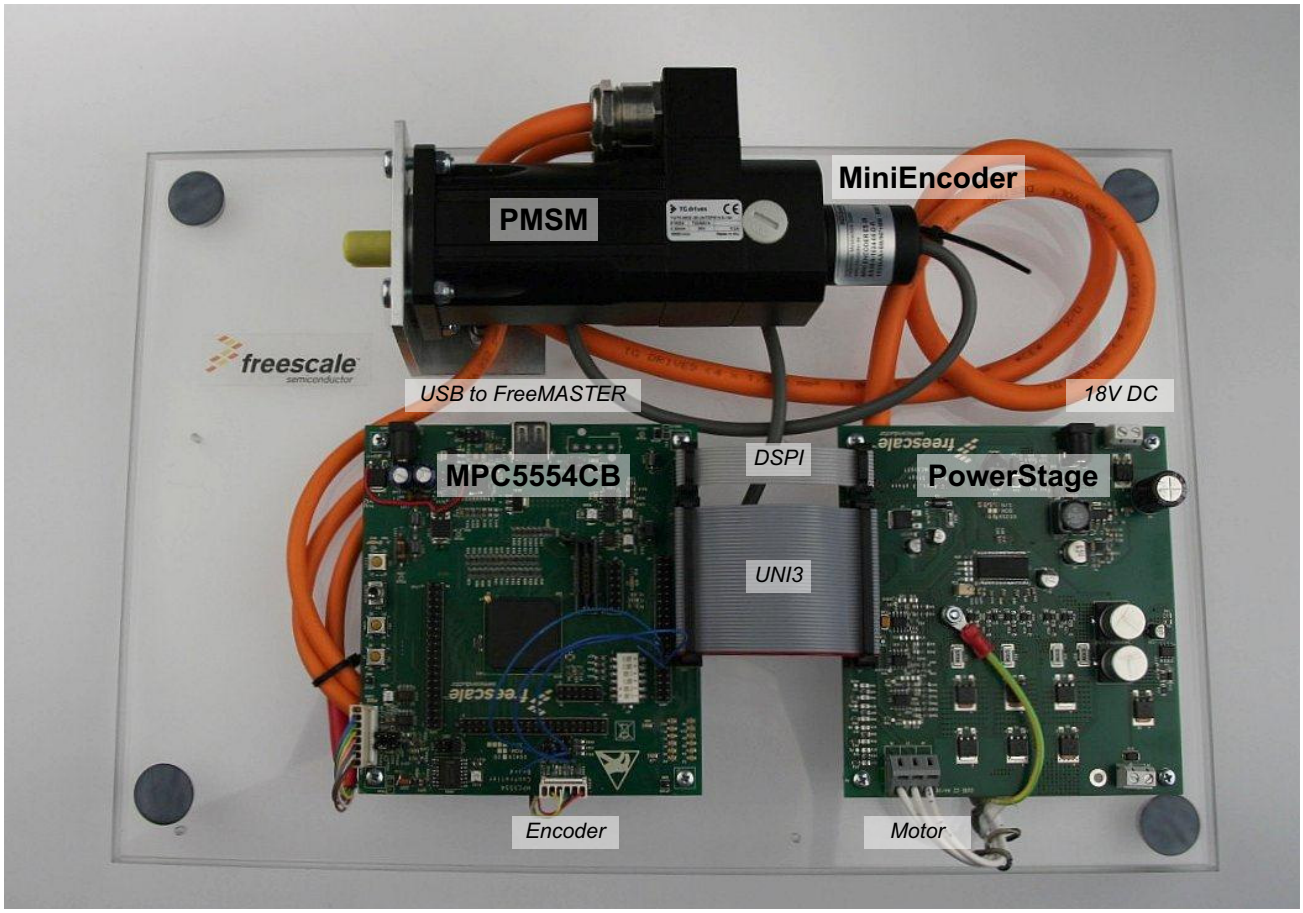


Figure 8. Connection of Application Parts

3.3.1 Power Architecture MPC5554 Controller Board (MPC5554CB)

The MPC5554CB features the MPC5554 microprocessor and circuitry for motor control applications:

- Start/stop switch and up/down buttons
- UNI3 motor control interface
- DSPI interface for MC33937
- Quadrature encoder connection
- Resolver connection
- Serial-to-USB and USB connection for FreeMASTER

For schematic, refer to Reference 2.

3.3.2 10 A 3-Phase Power Stage with MC33937

The 3-phase PMSM/BLDC power stage is based on MC33937 pre-driver integrated circuits. It operates on a wide range of input voltages from 8 V to 50 V, and is capable of driving currents of up to 10 A. The 3-phase power stage is an integral part of Freescale's embedded motion control series of development tools and can be used to drive 3-phase BLDC, PMSM, and ACIM motors.
For more information and schematic, refer to Reference 3.

3.3.3 PMSM with Quadrature Encoder

The motor used is a TG Drives low-voltage PMSM (TGT2-0032-30-24). Its characteristics are shown in Table 1.

Table 1. PMSM Motor Characteristics

Characteristic	Units	Value
Nominal DC voltage	V	18
Nominal torque	Nm	0.30
Nominal speed	rpm	3000
Nominal current	Amp	5.20
Stall torque	Nm	0.32
Stall current	Amp	4.948
Torque constant	Nm/Amp	0.065
Voltage constant	V/1000rpm	3.91
Resistance 2 ph.	Ohm	0.583
Inductance 2 ph.	mH	0.43

Table 1. PMSM Motor Characteristics (continued)

Characteristic	Units	Value
Pole-pairs	-	6

For more motor specifications, refer to Reference 4.

Quadrature encoder Mini Encoder ES28 (ES28-6-1024-05-D-R) is attached to the motor in order to scan and encode shaft movement. The basic encoder features are as follows:

- Three channel quadrature output with index pulse
- Resolution 1024 counts per revolution
- External mounting
- TTL compatible
- Single 5V supply

For more quadrature encoder specifications, refer to Reference 5.

3.3.4 Power Supply

The power stage board is powered by an 18 V / 4 A power supply and provides a 12 V supply for the MPC5554CB board.

4 Software Design

This section describes the software design of the PMSM vector control 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

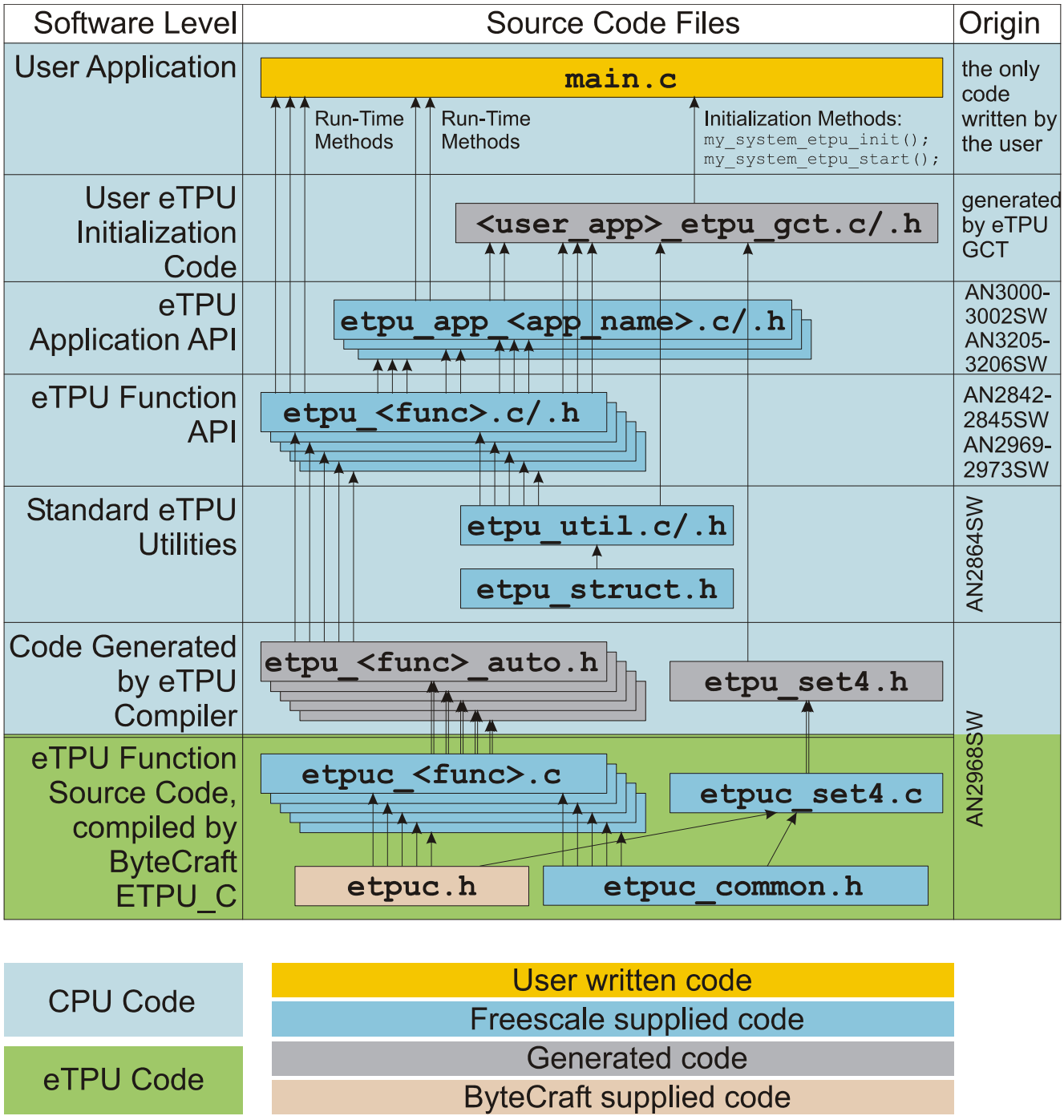


Figure 9. eTPU Project Structure

The CPU software uses several ready-to-use Freescale software drivers. Communications between the microprocessor and FreeMASTER on the PC is handled by software included in freemaster_protocol.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 for interfacing each eTPU

function during runtime. 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 coordination of eTPU functions.

4.1 CPU Software Flowchart

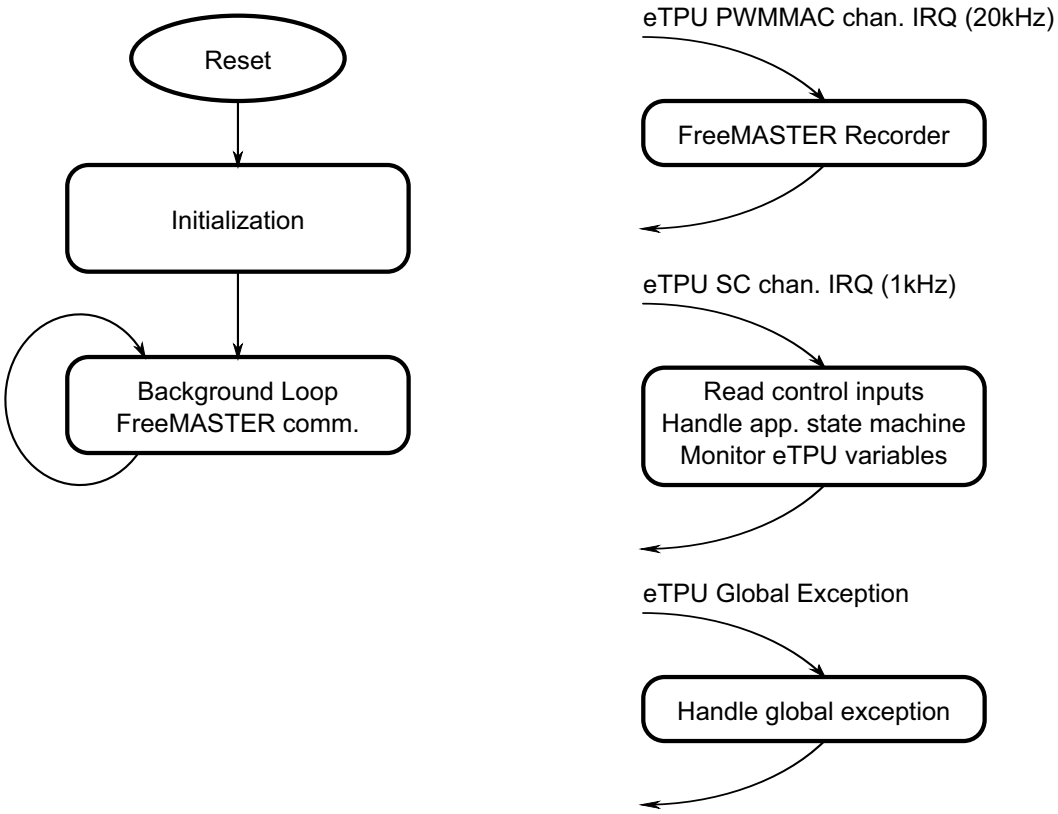


Figure 10. CPU Software Flowchart

After reset, the CPU software initializes pins, clock, interrupts, and peripheral modules. The ensuing CPU processing is incorporated in two periodic eTPU channel interrupts and one fault interrupt. FreeMASTER communication with the PC is managed in the background.

4.1.1 eTPU PWMMAC Channel Interrupt Service Routine

The eTPU runs a PWM Master (PWMMAC) eTPU function configured for updates at a rate of 20 kHz. The PWMMAC generates a channel interrupt on update. Propagation of this interrupt into the interrupt controller is enabled and the 20 kHz periodic interrupt is used to sample eTPU or CPU variables by FreeMASTER recorders. This enables exploration of the motor drive operation with each update.

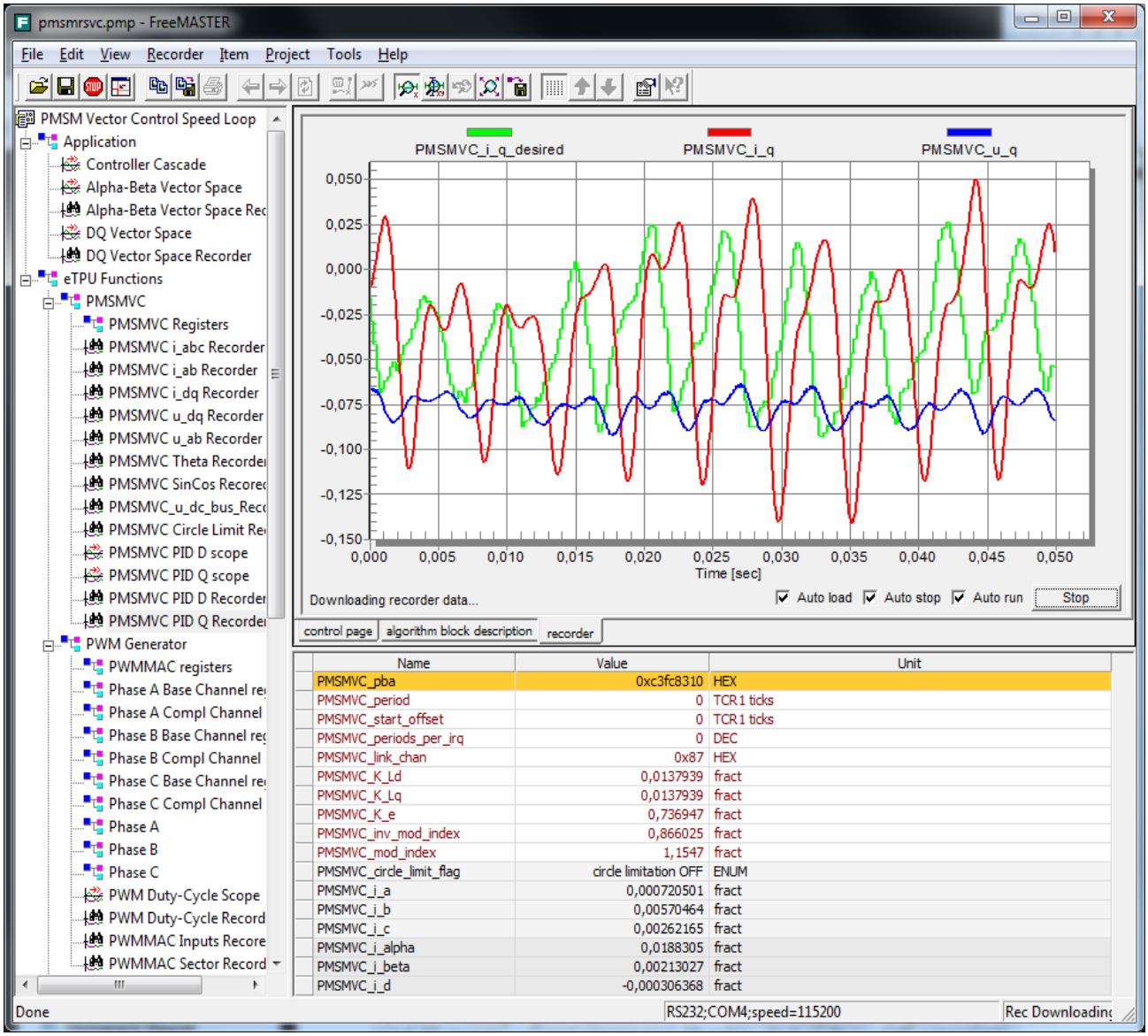


Figure 11. FreeMASTER Recorder of Q-portion PID Controller

4.1.2 eTPU SC Channel Interrupt Service Routine

The eTPU runs a Speed Controller (SC) eTPU function configured for an update every fourth PWM period. The SC generates a channel interrupt every fifth update, which results in a 1 kHz interrupt rate. Propagation of this interrupt into the interrupt controller is enabled and the 1 kHz periodic interrupt is used to handle the application state machine.

The following actions are performed:

- Read the status of the start/stop switch and of the up/down button.
- Read MC33937 faults.
- Manage the application state machine.

The application state diagram is described in detail below.

- Read the eTPU internal data.

4.1.3 eTPU Global Exception Interrupt Service Routine

The following situations can cause this eTPU global exception assertion:

- Microcode Global Exception
- Illegal Instruction Flag
- SCM MISC Flag
- Other flags specific only for eTPU2 on MPC5600 family

The type of eTPU global exception is recorded and a fault event is generated to the application state machine. Consequently, the motor drive is stopped and reinitialized.

4.2 Application State Diagram

The application state diagram consists of seven states and twelve events (see Figure 12).

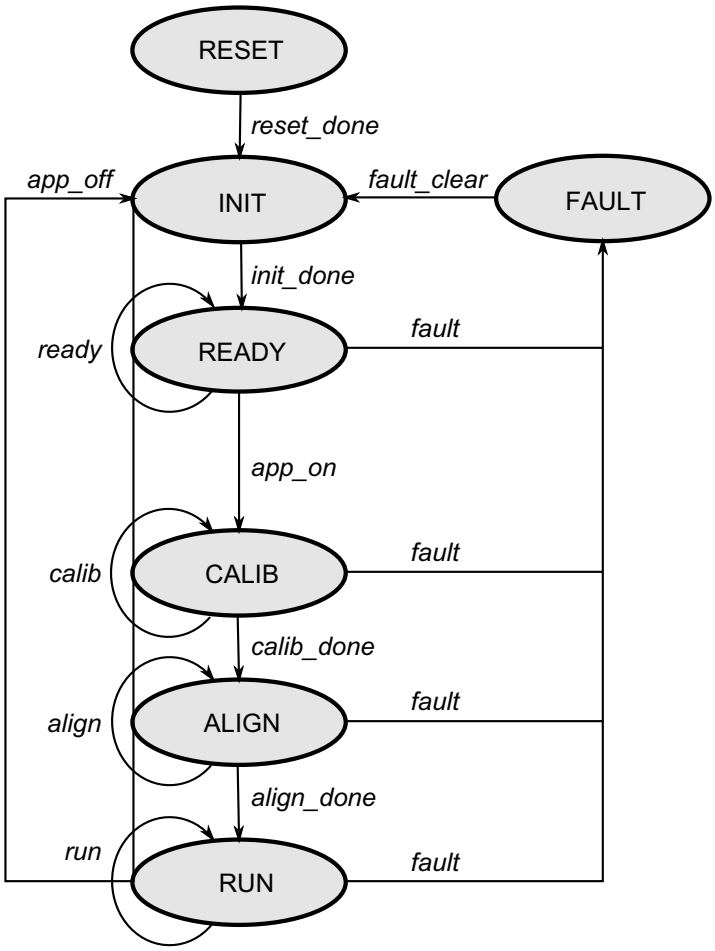


Figure 12. Application State Diagram

The next sections describe the processing in each of the application states.

4.2.1 RESET

This state is passed through only once. It is entered after a reset. The following actions are performed to configure the device:

- Initialize pins and clock.
- Initialize DSPI for communication with MC33937 predriver and initialize the predriver.
- Initialize eTPU and enable FreeMASTER to access eTPU function variables.
- Initialize eSCI for communication with FreeMASTER and initialize the FreeMASTER.
- Initialize eQADC and eDMA.
- Install interrupt handlers and enable interrupts.
- Start eQADC, eDMA and eTPU.

4.2.2 INIT

This state is passed through only. The following actions are performed in order to initialize (re-initialize) the application:

- Disable (or keep disabled) PWM outputs.
- Reset software controls and required speed.

4.2.3 READY

In this state the application waits until the user turns the motor on.

4.2.4 CALIB

In this state the application calibrates the analog input DC offsets. The application remains in this state for a defined number of cycles, so that the analog inputs and input filter outputs stabilize before reading the DC offsets.

When this state is entered, the *fs_etpu_app_pmsmesvc3_calib_start(...)* routine is called. After the defined delay, *fs_etpu_app_pmsmesvc3_calib_finish(...)* is called and the CALIB state is exited.

4.2.5 ALIGN

In this state the application aligns the motor shaft to a known position (0 electrical degrees) and resets the quadrature decoder position counter. The application remains in this state for a defined number of cycles, so that the shaft movement is finished before reading the zero-degree electrical position.

When this state is entered, the *fs_etpu_app_pmsmesvc3_align_start(...)* routine is called. After the defined delay, *fs_etpu_app_pmsmesvc3_align_finish(...)* is called the ALIGN state is exited.

4.2.6 RUN

The application remains in this state until the user turns the motor off. The required motor speed, set by FreeMASTER or read from the up/down buttons, is set to the eTPU motor drive using *fs_etpu_app_pmsmesvc3_set_speed_required(...)*.

4.2.7 FAULT

This state is entered on the fault event, which may occur in any state. The state in which the fault occurred is recorded. The following actions are performed:

- Disable PWM outputs.
- Reset software controls and required speed.

The FAULT state is left when the hardware start/stop switch is turned off.

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 PMSMESVC3 is an abbreviation for ‘PMSM with quadrature encoder and speed vector control’ eTPU motor-control application. As there can be several applications like this, the number 3 denotes the third such application in order.

The PMSMESVC3 eTPU application API is described in the next sections. There are 8 basic functions added to the PMSMESVC3 application API. The routines can be found in the etpu_app_pmsmesvc3.c/.h files. All PMSMESVC3 application API routines will be described in order and are listed below:

- Initialization function:

```
int32_t fs_etpu_app_pmsmesvc3_init(
  pmsmesvc3_instance_t * pmsmesvc3_instance,
                uint8_t   PWM_master_channel,
                uint8_t   PWM_phaseA_channel,
                uint8_t   PWM_phaseB_channel,
                uint8_t   PWM_phaseC_channel,
                uint8_t   QD_phaseA_channel,
                uint8_t   QD_index_channel,
                uint8_t   SC_channel,
                uint8_t   BC_channel,
                uint8_t   PMSMVC_channel,
                uint8_t   ASAC_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_range_mv,
                uint8_t   pole_pairs,
```

```
                    uint32_t  SC_freq_hz,
                    int32_t   SC_PID_gain_permil,
                    int32_t   SC_I_time_const_us,
                    uint32_t  SC_ramp_time_ms,
                    int32_t   PMSMVC_D_PID_gain_permil,
                    int32_t   PMSMVC_D_I_time_const_us,
                    int32_t   PMSMVC_Q_PID_gain_permil,
                    int32_t   PMSMVC_Q_I_time_const_us,
                    int32_t   PMSM_Ke_mv_per_krpm,
                    int32_t   PMSM_L_uH,
                    uint32_t  QD_pc_per_rev,
                    int32_t   phase_current_range_ma,
                    uint8_t   BC_mode,
                    uint8_t   BC_polarity,
                    uint8_t   BC_u_dc_bus_ON_perc,
                    uint8_t   BC_u_dc_bus_OFF_perc,
                    uint8_t   ASAC_polarity,
                    uint32_t  ASAC_measure_time_us,
                    uint32_t *ASAC_result_queue,
                    uint8_t   ASAC_bit_shift,
                    uint8_t   ASAC_ia_queue_offset,
                    uint8_t   ASAC_ib_queue_offset,
                    uint8_t   ASAC_ic_queue_offset,
                    uint8_t   ASAC_u_dcbus_queue_offset,
                    uint32_t  ASAC_filter_time_constant_i_us,
                    uint32_t  ASAC_filter_time_constant_u_us);
```

- Change operation functions:

```
int32_t fs_etpu_app_pmsmesvc3_calib_start(
   pmsmesvc3_instance_t * pmsmesvc3_instance)
int32_t fs_etpu_app_pmsmesvc3_calib_finish(
   pmsmesvc3_instance_t * pmsmesvc3_instance)

int32_t fs_etpu_app_pmsmesvc3_align_start(
   pmsmesvc3_instance_t * pmsmesvc3_instance,
                uint24_t  alignment_u_d)
int32_t fs_etpu_app_pmsmesvc3_align_finish(
   pmsmesvc3_instance_t * pmsmesvc3_instance,
                uint8_t   sc_configuration)
int32_t fs_etpu_app_pmsmesvc3_disable(
   pmsmesvc3_instance_t * pmsmesvc3_instance)
```

```
void fs_etpu_app_pmsmesvc3_set_speed_required(
   pmsmesvc3_instance_t * pmsmesvc3_instance,
                int32_t   speed_required_rpm)
```

- Value return function:

```
void fs_etpu_app_pmsmesvc3_get_data(
   pmsmesvc3_instance_t * pmsmesvc3_instance,
       pmsmesvc3_data_t * pmsmesvc3_data)
```

4.3.1 int32_t fs_etpu_app_pmsmesvc3_init(...)

This routine is used to initialize the eTPU channels for the PMSM with quadrature encoder and speed vector control application. This function has the following parameters:

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`. This structure must be declared in the user application. When there are more instances of the application running simultaneously, there must be a separate pmsmesvc3_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_PMSMESVC3_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_PMSMESVC3_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_PMSMESVC3_COMPL_PAIRS), the complementary channel is one channel higher.
- **QD_phaseA_channel (uint8_t)** - This is the quadrature decoder phase A channel number. 0-30 for ETPU_A, and 64-94 for ETPU_B. The quadrature decoder phase A channel is one channel higher.
- **QD_index_channel (uint8_t)** - This is the quadrature decoder index 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.
- **BC_channel (uint8_t)** - This is the break controller channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **PMSMVC_channel (uint8_t)** - This is the PMSM vector control function channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.

- **ASAC_channel (uint8_t)** - This is the analog sensing for AC motors (ASAC) 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_PMSMESVC3_SINGLE_CHANNELS, or FS_ETPU_APP_PMSMESVC3_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_range_mv (int32_t)** - This is the maximum measurable DC-bus voltage in mV.
- **pole_pairs (uint8_t)** - This is the number of motor pole-pairs.
- **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_PID_gain_permil (int32_t)** - This is the speed PI controller gain in millesimals.
- **SC_I_time_constant_us (int32_t)** - This is the speed PI controller integral time constant in µs.
- **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.
- **PMSMVC_D_PID_gain_permil (int32_t)** - This is the D-current (flux controlling current) PI controller gain in millesimals.
- **PMSMVC_D_I_time_constant_us (int32_t)** - This is the D-current (flux controlling current) PI controller integral time constant in µs.
- **PMSMVC_Q_PID_gain_permil (int32_t)** - This is the Q-current (torque controlling current) PI controller gain in millesimals.
- **PMSMVC_Q_I_time_constant_us (int32_t)** - This is the Q-current (torque controlling current) PI controller integral time constant in µs.
- **PMSM_Ke_mv_per_krpm (int32_t)** - This is the motor electrical constant in mV/1000RPM.
- **PMSM_L_uH (int32_t)** - This is the motor induction in µH.
- **QD_qd_pc_per_rev (uint32_t)** - This is the number of QD position counter increments per one revolution.
- **phase current_range_ma (int32_t)** - This is the maximum measurable phase current in mA.
- **BC_mode (uint8_t)** - This is the BC function mode. This parameter should be assigned a value of: FS_ETPU_APP_PMSMESVC3_BC_MODE_ON_OFF, or FS_ETPU_APP_PMSMESVC3_BC_MODE_PWM.
- **BC_polarity (uint8_t)** - This is the BC output polarity. This parameter should be assigned a value of: FS_ETPU_APP_PMSMESVC3_BC_ON_HIGH, or FS_ETPU_APP_PMSMESVC3_BC_ON_LOW.

- **BC_u_dc_bus_ON_perc (uint8_t)** - This is the proportion between U_DC_BUS, above which the BC output is ON, and the nominal U_DC_BUS, expressed in percentage (usually about 130%).
- **BC_u_dc_bus_OFF_perc (uint8_t)** - This is the proportion between U_DC_BUS, below which the BC output is OFF, and the nominal U_DC_BUS, expressed in percentage (usually about 110%).
- **ASAC_polarity (uint8_t)** - This is the polarity to assign to the ASAC function. This parameter should be assigned a value of: FS_ETPU_APP_PMSMESVC3_ASAC_PULSE_HIGH or FS_ETPU_APP_PMSMESVC3_ASAC_PULSE_LOW.
- **ASAC_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.
- **ASAC_result_queue (uint32_t *)** - Pointer to the result queue in eTPU DATA RAM. Result queue is an array of 16-bit words that contains the measured values.
- **ASAC_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_PMSMESVC3_ASAC_SHIFT_LEFT_BY_8, FS_ETPU_APP_PMSMESVC3_ASAC_SHIFT_LEFT_BY_10, FS_ETPU_APP_PMSMESVC3_ASAC_SHIFT_LEFT_BY_12, or FS_ETPU_APP_PMSMESVC3_ASAC_SHIFT_LEFT_BY_16.
- **ASAC_ia_queue_offset (uint8_t)** - Position of the phase A current sample in the result queue. Offset is defined in bytes.
- **ASAC_ib_queue_offset (uint8_t)** - Position of the phase B current sample in the result queue. Offset is defined in bytes.
- **ASAC_ic_queue_offset (uint8_t)** - Position of the phase C current sample in the result queue. Offset is defined in bytes.
- **ASAC_u_dcbus_queue_offset (uint8_t)** - Position of the DC-bus voltage sample in the result queue. Offset is defined in bytes.
- **ASAC_filter_time_constant_i_us (uint32_t)** - This is the time constant of an Exponentially-Weighted Moving Average (EWMA) filter which applies when processing the phase current samples, in us.
- **ASAC_filter_time_constant_u_us (uint32_t)** - This is the time constant of an EWMA filter which applies when processing the DC-bus voltage samples, in us.

Note that besides these parameters, there are additional configuration items hardcoded in the initialization code, such as PWM output polarity, ADC commands that define analog input channels, etc. The example code configures the motor drive for specific hardware and contains helpful comments.

4.3.2 int32_t fs_etpu_app_pmsmesvc3_calib_start(...)

This routine turns the PWM top outputs off and bottom outputs on, to enable measurement of phase current DC offsets. This function has the following parameter:

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`.

4.3.3 int32_t fs_etpu_app_pmsmesvc3_calib_finish(...)

This routine uses the filtered phase currents (output of Analog Sensing (ASAC) eTPU functions) to set the DC offsets. Additionally, the Break Controller (BC) absolute threshold values are set using the filtered DC bus voltage and percentage thresholds. This function has the following parameter:

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`.

4.3.4 int32_t fs_etpu_app_pmsmesvc3_align_start(...)

This routine sets a fixed D-voltage in the DQ coordinates and enables the generation of PWM signals. The motor will start to move the position of 0 electrical degrees. This function has the following parameters:

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`.
- **alignment_u_d (int24_t)** - This is the D-voltage used for motor alignment as a 24-bit signed fractional value

4.3.5 int32_t fs_etpu_app_pmsmesvc3_align_finish(...)

This routine resets the Quadrature Decoder (QD) position and enables the motor drive with Speed Controller (SC) control loop either closed for speed control, or open for torque control. This function has the following parameters:

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`.
- **sc_configuration (uint8_t)** - This is the required configuration of the SC. This parameter should be assigned a value of:
  FS_ETPU_APP_PMSMESVC3_SPEED_LOOP_OPENED, or
  FS_ETPU_APP_PMSMESVC3_SPEED_LOOP_CLOSED.

4.3.6 int32_t fs_etpu_app_pmsmesvc3_disable (pmsmesvc3_instance_t * pmsmesvc3_instance)

This routine is used to disable the generation of PWM signals and to stop the vector control loop and the speed controller. This function has the following parameter:

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`.

4.3.7 void fs_etpu_app_pmsmesvc3_set_speed_required(...)

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

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`.
- **speed_required_rpm (int32_t)** - This is the required motor speed in rpm.
  In case the speed loop is opened (sc_configuration has been set to FS_ETPU_APP_PMSMESVC3_SPEED_LOOP_OPENED in `fs_etpu_pmsmesvc3_align_finish(...)`), the required speed value is passed directly to the speed controller output, which is the required motor torque. In this case, the required motor speed in RPM, as a fraction of the speed range in RPM, corresponds to the required motor torque, as a fraction of the maximum motor torque.

4.3.8 void fs_etpu_app_pmsmesvc3_get_data(...)

This routine is used to get the application state data, including

- actual motor speed
- required motor torque
- actual motor torque
- applied motor voltage
- revolution counter
- motor direction
- SC, D and Q PID controllers' saturation flags

. This function has the following parameters:

- **pmsmesvc3_instance (pmsmesvc3_instance_t*)** - This is a pointer to pmsmesvc3_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc3_init`.
- **pmsmesvc3_data (pmsmesvc3_data_t*)** - This is a pointer to pmsmesvc3_data_t structure of application state data, which is updated.

4.4 eTPU Block Diagram

The eTPU functions used in PMSM vector control drive are located in the AC motor-control set of eTPU functions (set4 - AC motors). The eTPU functions within the set serve as building blocks for various AC motor-control applications. The following paragraphs describe the functionality of each block.

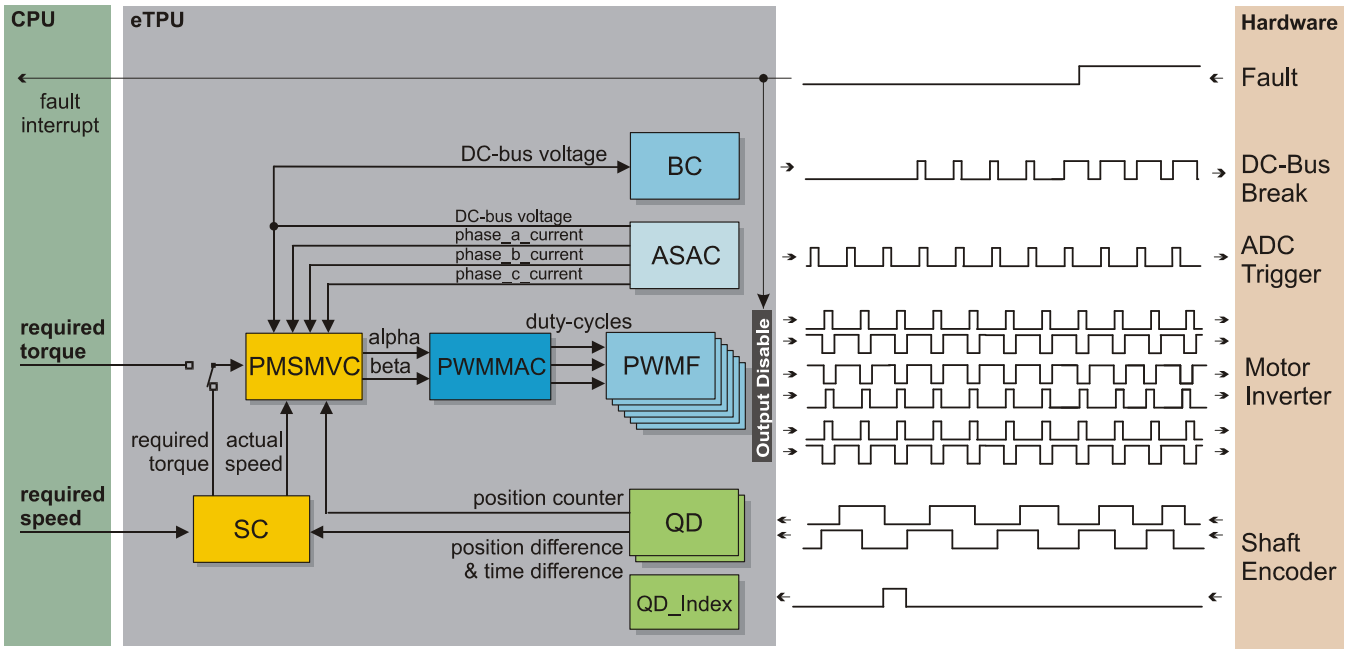


Figure 13. Block Diagram of eTPU Processing

4.4.1 PWM Generator (PWMMAC+PWMF)

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

PWM - master for AC motors (PWMMAC)

PWM - full range (PWMF)

The PWM master for AC motors (PWMMAC) function calculates sine wave or space vector modulation resulting in PWM duty cycles, and updates the three PWM phases. The phases are driven by the PWM full range (PWMF) function, which enables a full (0% to 100%) duty-cycle range.

The PWMF function generates the PWM signals. The PWMMAC function controls three PWMF functions, three PWM phases, and does not generate any drive signal. The PWMMAC can be executed even on an eTPU channel not connected to an output pin.

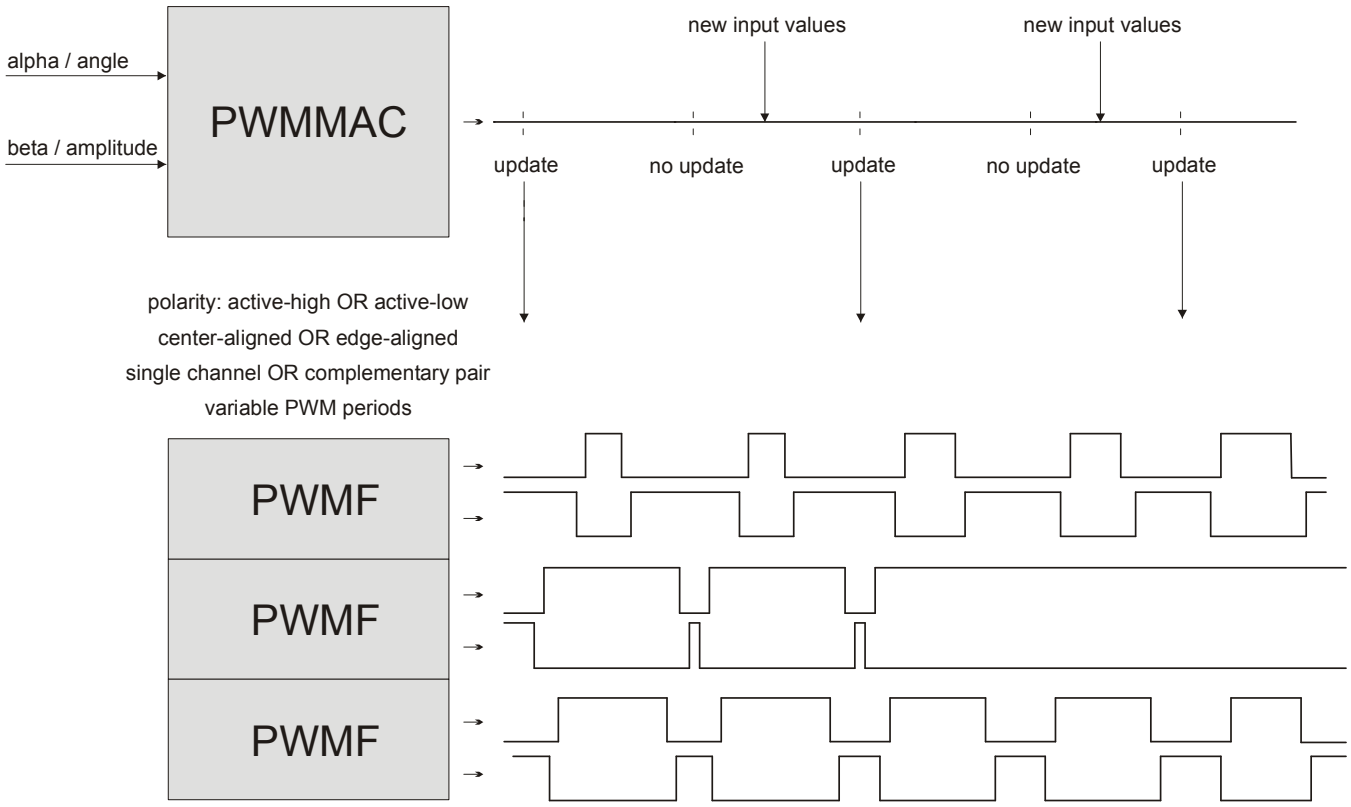
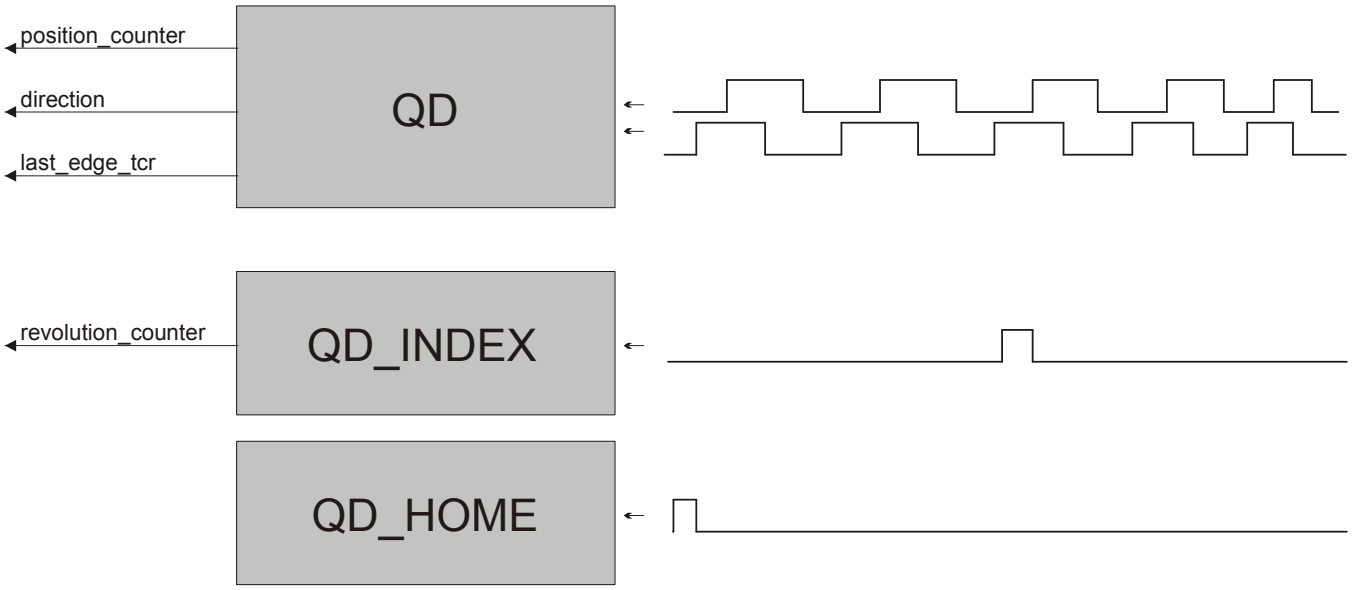


Figure 14. Functionality of PWMMAC+PWMF

For more details about the PWMMAC and PWMF eTPU functions, refer to Reference 10.

4.4.2 Quadrature Decoder (QD)

The quadrature decoder eTPU function set is intended to process signals generated by a shaft encoder in a motion control systems. It uses two channels to decode a pair of out-of-phase encoder signals and to produce a 24-bit bi-directional position counter, together with direction information, for the CPU. An additional input channels can also be processed. The index channel receives a pulse on each revolution. Based on the actual direction, a revolution counter is incremented or decremented on the index pulse. A further additional input channel can indicate a home position, but it is not used in this application.



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

4.4.3 Analog Sensing for AC Motors (ASAC)

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

All the above mentioned ASAC features are utilized in the application. The ASAC is initialized to run in PWM synchronized mode, e.g. the first ASAC edge is synchronized with the beginning of the PWM period. Simultaneously, the ASAC manages to synchronize the SC function by generating the link to the SC channel every 4^{th} ASAC period and to synchronize the PMSMVC function by generating the link to the PMSMVC channel each ASAC period.

The ASAC function preprocesses the phase currents and DC-bus voltage analog values and passes the adjusted values as an input to the PMSMVC and BC functions. Processing of the DC-bus voltage sample includes bit shifting, dc-offset removing, and filtering. Processing of phase current samples includes also computation of the third phase current from the other two, based on actual motor position in one of 6 sectors, and computation of dead-time compensation parameters.

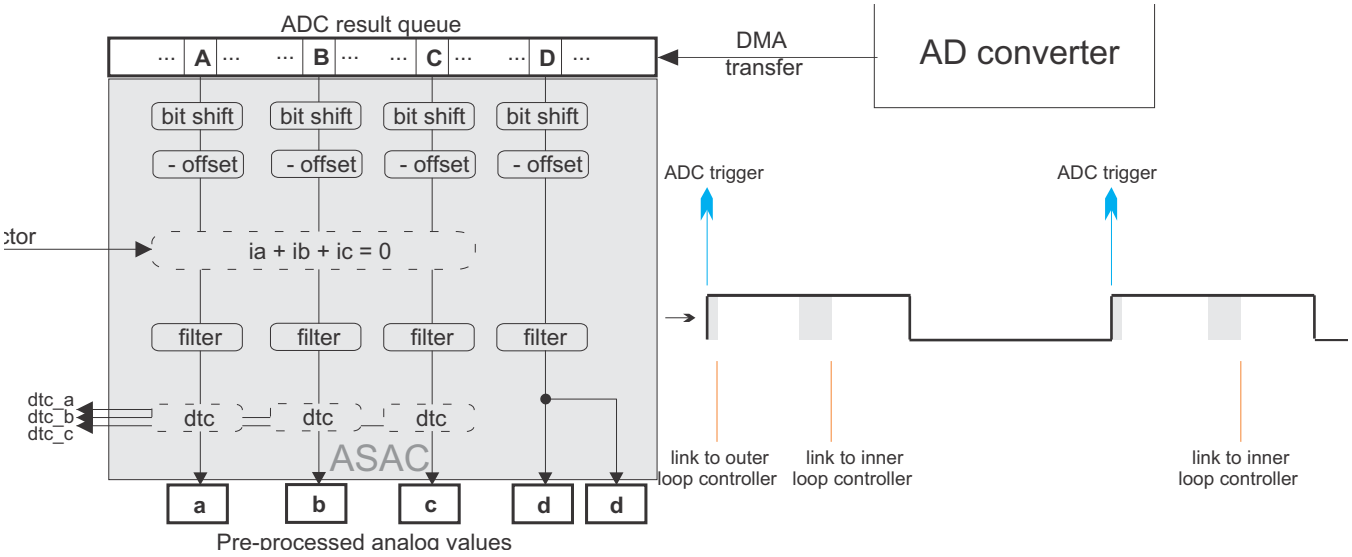


Figure 15. Functionality of ASAC

For more details about the ASAC eTPU function, refer to Reference 11.

In order to ensure periodic sampling and the quick transfer of the measured data from the AD converter to the eTPU DATA RAM, several peripheral modules are used, see Figure 16:

- On-chip analog to digital converter (eQADC) is used for sampling of the analog values. Sampling of analog values is triggered by an eTRIG signal generated internally by the ASAC eTPU function running on eTPU channel 29.
- 4 direct memory access (eDMA) channels are used as follows:
  - eDMA channel 47 is used for the transfer of four 32-bits eQADC conversion commands from the eTPU DATA RAM (p_ASAC_command_queue) to the EQADC_CFPR2 (CFIFO Push Register 2) of the eQADC module. The eDMA channel 47 transfer is initiated by a DMA request generated by ASAC eTPU function. ASAC changes the command queue based on actual motor position in one of 6 sectors in order to ensure the defined order of analog values sampling.
  - eDMA channel 1 is used for the transfer of the 16-bits phaseA current sampling result from the EQADC_RFPR0 (result FIFO pop register 0) to the eTPU DATA RAM. The DMA channel 1 transfer is initiated by a DMA request generated by eQADC module after the conversion is finished.
  - eDMA channel 3 is used for the transfer of the 16-bits phaseB current sampling result from the EQADC_RFPR1 (result FIFO pop register 1) to the eTPU DATA RAM. The DMA channel 3 transfer is initiated by a DMA request generated by eQADC module after the conversion is finished.
  - eDMA channel 5 is used for the transfer of two 16-bits sampling results (phaseC current and DC BUS voltage) from the EQADC_RFPR3 (result FIFO pop register 3) to the eTPU DATA RAM. The DMA channel 5 transfer is initiated by a DMA request generated by eQADC module after the conversion is finished.

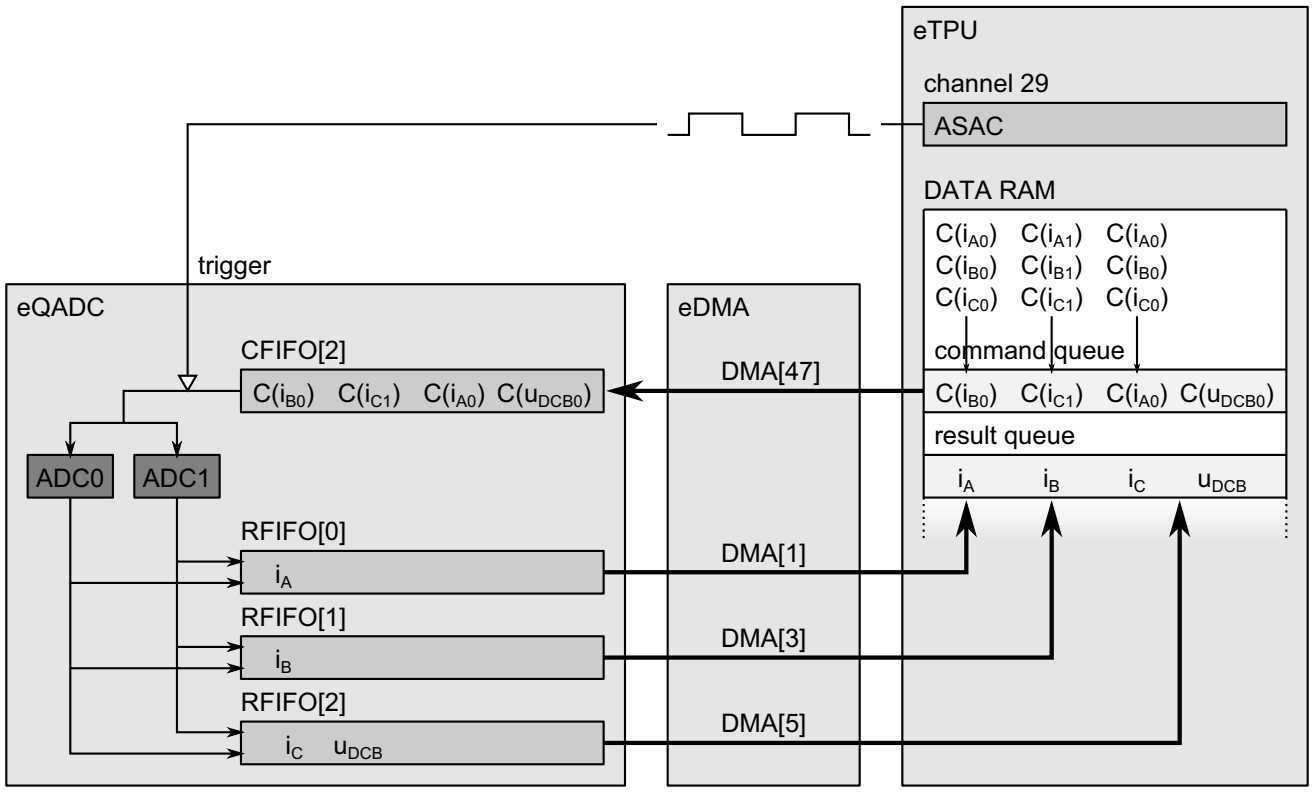


Figure 16. Cooperation of eTPU, eDMA and eQADC modules

4.4.4 PMSM Vector Control (PMSMVC)

The PMSM Vector Control eTPU function is not intended to process input or output signals. Its purpose is to control another eTPU function's input parameter. The PMSMVC function can be executed even on an eTPU channel not connected to an output pin.

The purpose of the PMSMVC function is to perform the current control loop of a field-oriented (vector control) drive of a permanent magnet synchronous motor (PMSM). The sequence of PMSMVC calculations consists of the following steps:

- Forward Clarke transformation
- Forward Park transformation (establishing the DQ coordinate system)
- D&Q current controllers calculation
- Decoupling and back-EMF feed forward
- Circle limitation
- Inverse Park transformation
- DC-bus ripple elimination

The PMSMVC calculates applied voltage vector components alpha & beta based on measured phase currents and required values of phase currents in 2-phase orthogonal rotating reference frame (D-Q). The

PMSMVC function optionally enables to perform the limitation of calculated D and Q components of the stator voltages into the circle.

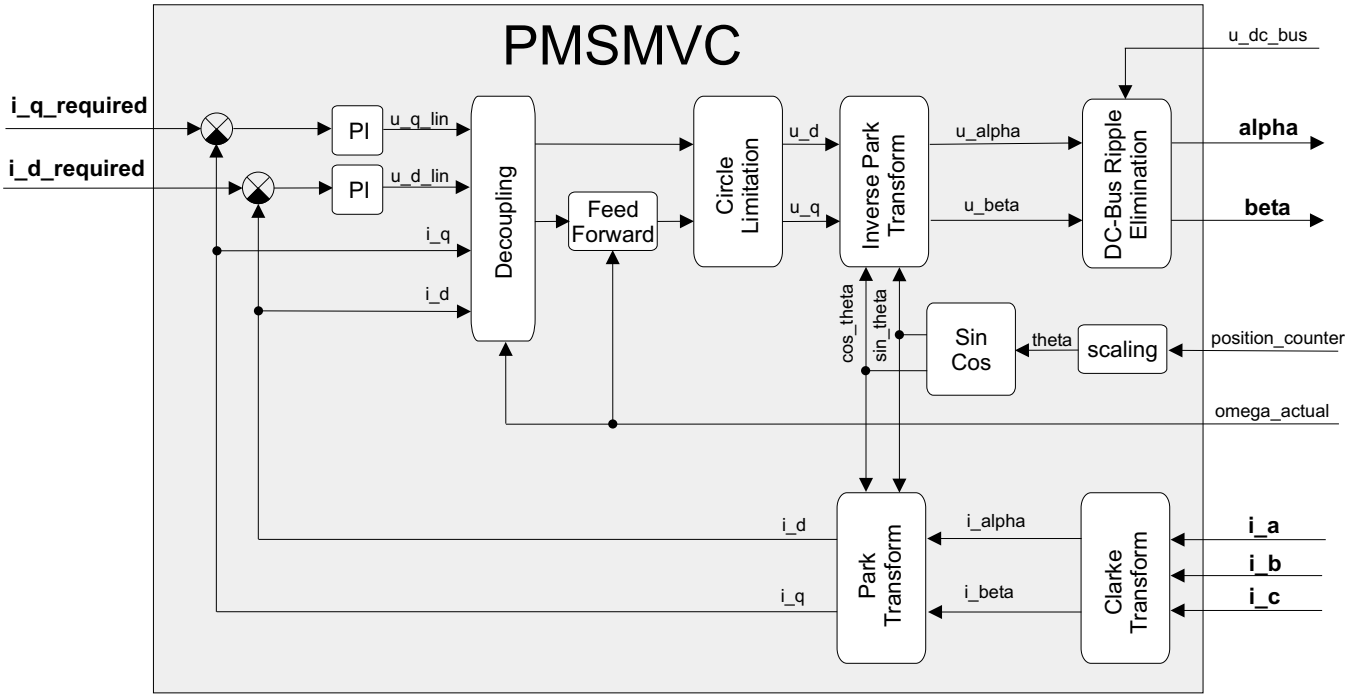


Figure 17. Functionality of PMSMVC

For more details about the PMSMVC eTPU function, refer to Reference 12.

4.4.5 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 QD 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.

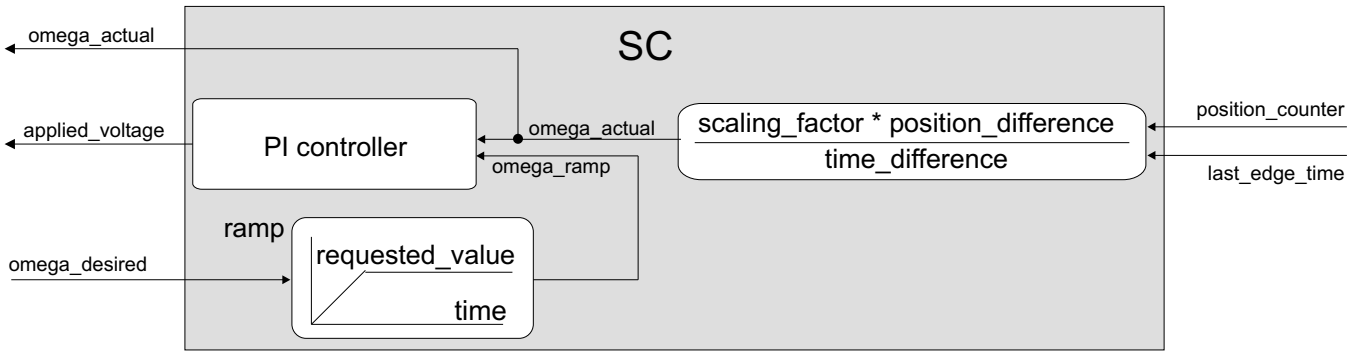


Figure 18. Functionality of SC

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

4.4.6 Break Controller (BC)

The purpose of the break controller (BC) eTPU function is to eliminate DC-bus overvoltage when a motor is driven in the generating mode. The BC function generates the DC-bus break control signal (see Figure 19) based on the actual DC-bus voltage.

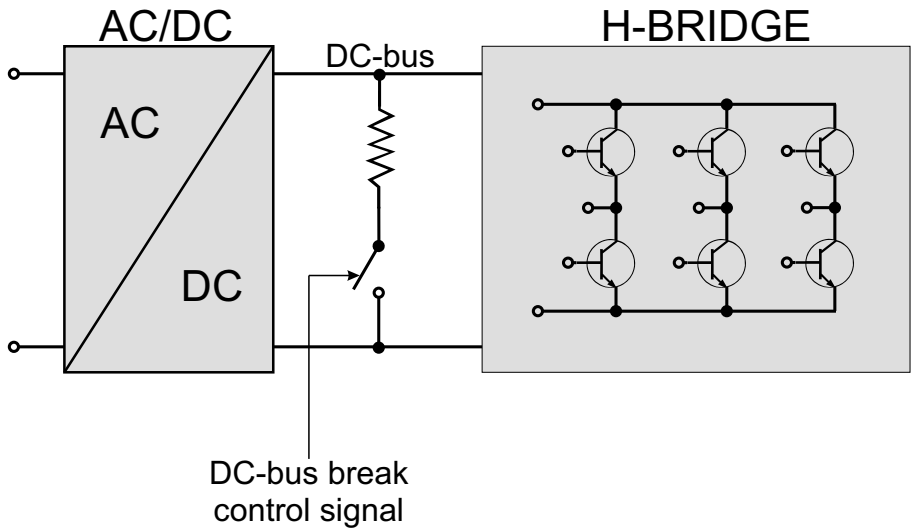


Figure 19. Functionality of BC

The described application uses the PWM mode of the BC function. In this mode, the BC function switches softly using a PWM signal. The *u_dc_bus_ON* and *u_dc_bus_OFF* thresholds define a ramp (see Figure 20). When the DC-bus voltage is lower than *u_dc_bus_OFF*, the control signal is turned off. Between the *u_dc_bus_OFF* and *u_dc_bus_ON* thresholds, a PWM signal with a duty-cycle linearly increasing from 0% to 100% is generated. Above the *u_dc_bus_ON* threshold, the control signal is turned on.

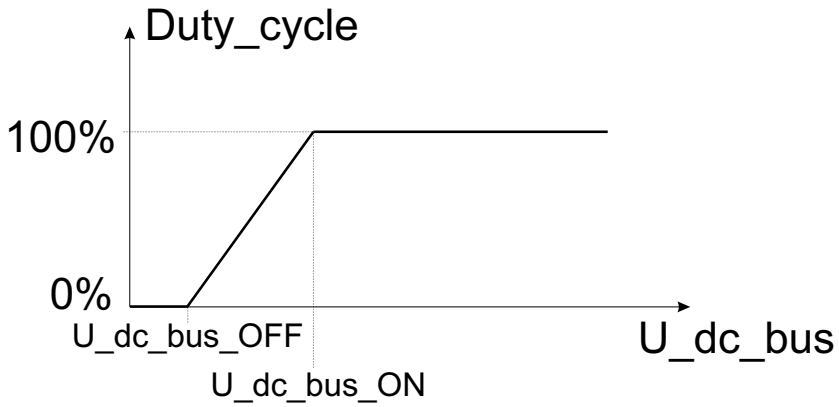


Figure 20. PWM Mode of the Break Controller Function

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. For correct operation, the WCL of each channel must be kept below a required limit. For example, the WCL of the PWMF 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 ASAC output (ADC trigger), the behavior of a channel activity pulse can be described by one of the following cases:

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

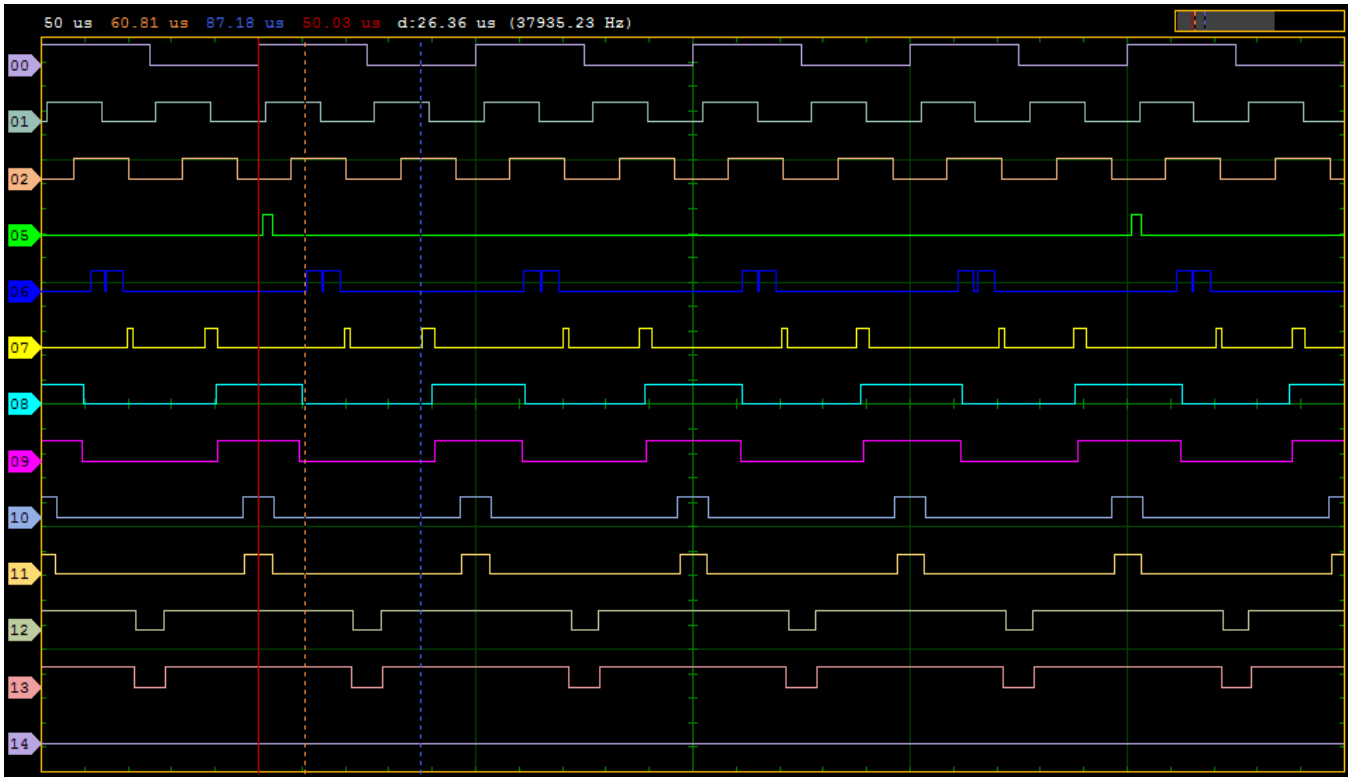


Figure 21. Logic Analyzer Screen-shot of eTPU Timing

Figure 21 explains the application eTPU timing. The logic analyzer screen-shot depicts a typical situation described below:

The Figure 21 shows all input and output eTPU signals. The signals are numbered according to eTPU channels. The whole picture shows a time section of 6 PWM periods.

Signal 0 is generated by the Analog Sensing for AC motors (ASAC) eTPU function. This signal triggers the AD converter on the low-high edge, which is also the beginning of a PWM period frame. The high-low edge is the center of the PWM period.

Signals 1 and 2 are coming from the shaft encoder and are processed by quadrature decoder (QD) eTPU function. In the moment, the motor speed is 3000 RPM.

Signal 5 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 the QD position counter and QD last edge time, calculate the required speed ramp, and apply the PI controller algorithm. This calculation is performed periodically at a 5kHz rate, which is every 4^{th} PWM period.

Signal 6 is generated by the PMSM vector control (PMSMVC) eTPU function. The pairs of pulses show that the update of the PMSMVC is divided in two consecutive threads. The PMSM update is performed every PWM period.

Signal 7 is generated by the PWM master for AC motors (PWMMAC) eTPU function. Its pulses determine the activity of the PWMMAC. Immediately after PMSMVC update is finished, a narrow PWMMAC pulse occurs. These pulses determine the service time of an PMSMVC request to update the applied motor

voltage vector. This service includes also the calculation of space vector modulation. Apart from these pulses, a wider pulses signal the actual update for the next PWM period.

Signals 8 to 13 are three phases of PWM signals - three center-aligned complementary pairs. The base channels (8, 10, 12) are of negated polarity. The PWM period is 50µs, which corresponds to a PWM frequency of 20kHz.

Signal 14 is generated by the break controller (BC) eTPU function. When the break controller channel receives the link, the actual value of DC-bus voltage is compared with the defined over-voltage thresholds, and according to this comparison it generates the PWM-based break control signal.

The first ASAC thread executed on the ASAC low-high edge requests SC update every 4^{th} period, by sending a link to the SC channel. Consequently, the SC requests BC update. Both SC and BC updates are finished prior to the ASAC second thread. This thread preprocess the analog inputs and requests PMSMVC update every period. This happens after approximately a quarter of the PWM period frame — see the orange cursor on Figure 21. The PMSMVC update is separated into two threads. After the second one is finished, the PMSMVC requests PWMMAC to update the applied motor voltage vector. All of these ASAC, SC, BC, and PMSMVC activities have to be finished prior to the start of PWMMAC update. If they are not finished the PWMMAC update is postponed to the next PWM period. The update starts at 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 case. Reference 10 describes how to set the update_time value.

5 Implementation Notes

5.1 Scaling of Quantities

The PMSM vector control algorithm running on eTPU uses a 24-bit fractional representation for all real quantities except time. The 24-bit signed fractional format is mostly 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.]

Some quantities are represented using 3.21 format (3 signed integer bits, 21 fractional bits) in order to eliminate the need of saturation to range (-1, 1). The most negative number that can be represented is -4.0, whose internal representation is 0x800000. The most positive number is 0x7FFFFF or $4.0 - 2^{-21}$. In this format the components of applied motor voltage vector are passed from PMSMVC to PWMMAC.

5.2 Speed Calculation

The speed controller (SC) eTPU function calculates the angular motor speed using pc_sc and last_edge parameters of the QD eTPU function. The following equation applies:

$$\text{omega\_actual} = \frac{\text{position\_difference}}{\text{time\_difference}} \cdot \text{scaling\_factor}$$

where:

omega_actual [fract24] is the actual angular speed as a fraction of the maximum speed range

position_difference [int24] is the difference between the updated value of QD position counter and the previous value, which was captured by SC in the previous SC period. In fact the position_difference is readable from pc_sc parameter of the QD function. After SC reads the new updated value it resets this pc_sc parameters which ensures that the position_difference is available in the pc_sc parameter next time SC reads it.

time_difference [int24] is the difference between the updated value of QD last_edge and the previous value, which was captured by SC in the previous SC period

scaling_factor is pre-calculated using the following equation:

$$\text{scaling\_factor} = \frac{30 \cdot 256 \cdot \text{etpu\_tcr\_freq}}{\text{omega\_max} \cdot \text{pc\_per\_rev}}$$

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

pc_per_rev is a number of QD position counter increments per one revolution

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	2M	49K
RAM	64K	4K
eTPU code RAM	16K	12K
eTPU data RAM	3K	1K

7 Summary and Conclusions

This application note provides the user with a description of the demo application driving a PMSM motor using quadrature encoder and vector control techniques. The application also demonstrates use of the eTPU module, which results in a CPU-independent motor drive. Lastly, the demo application is targeted at the MPC5554 microprocessor, but could be easily reused with any device of the MPC5500 and MPC5600 family that has an eTPU.

8 References

Table 3. References

1. MPC5554 Reference Manual, MPC5554RM
2. MPC5554 Controller Board Schematic, MPC5554CB_SCH included in AN3206SW/doc
3. 3-Phase Power Stage with MC33937 User's Manual, 3phLVPMSMPSUM included in AN3206SW/doc
4. TG drives web: http://www.tgdrives.com
5. Quadrature Encoder Mini Encoder ES28: http://www.inducoder.de/pdf/es28.pdf.
6. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER"
7. Enhanced Time Processing Unit Reference Manual, ETPURM
8. "Using the Quadrature Decoder (QD) eTPU Function," AN2842
9. "Using the Speed Controller (SC) eTPU Function," AN2843
10. "Using the AC Motor Control PWM eTPU Functions," AN2969
11. "Using the Analog Sensing for AC Motors (ASAC) eTPU Function," AN2970
12. "Using the PMSM Vector Control (PMSMVC) eTPU Function," AN2972
13. "Using the Break Controller (BC) eTPU Function," AN2845
14. "Using the AC Motor Control eTPU Function Set (set4)," AN2968

9 Revision history

Table 4. Revision history

Revision number	Revision date	Description of changes
1	02 May 2012	Updated for support of motor drives with resolver position sensor.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN3206
Rev. 1
03/2012

freescale™