Freescale Semiconductor
Application Note

Document Number: AN4561
Rev. 1, 11/2015

3-Phase PMSM Motor Control Kit with the MPC5604P

by: Roman Filka and Marek Stulrajter

Contents

1 Introduction

This application note describes the design of a 3-phase Permanent Magnet Synchronous Motor (PMSM) vector control drive with 3-shunt current sensing with a position sensor. The design is targeted for automotive applications. This cost-effective solution benefits from a MPC5604P device dedicated for motor control.

The system is designed to drive a 3-phase PM synchronous motor. Application features:

- 3-phase PMSM speed Field Oriented Control.
- Current sensing with three shunt resistors.
- Support for encoder and resolver position transducers.
- Application control user interface using FreeMASTER debugging tool.
- Motor Control Application Tuning (MCAT) tool

2 System concept

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

- Targeted at the MPC5604P Controller (refer to dedicated user manual for MPC5604P found at www.freescale.com), see References for more information

- Running on the MPC5604P Control Drive board (refer to dedicated user manual for MPC5604P Controller Board), see References for more information
- Control technique incorporating:
  - Vector control of 3-phase PM synchronous motor with position sensor
  - Closed-loop speed control
  - Bi-directional rotation
  - Both motor and generator modes
  - Close-loop current control
  - Flux and torque independent control
  - Start up with alignment
  - Reconstruction of three-phase motor currents from two shunt resistors
  - 100 µs sampling period with FreeMASTER recorder
- FreeMASTER software control interface (motor start/stop, speed setup), see References for more information
- FreeMASTER software monitor
  - FreeMASTER software graphical control page (required speed, actual motor speed, start/stop status, DC-Bus voltage level, motor current, system status)
  - FreeMASTER software speed scope (observes actual and desired speeds, DC-Bus voltage and motor current)
  - FreeMASTER software high-speed recorder (reconstructed motor currents, vector control algorithm quantities)
- DC-Bus over-voltage and under-voltage, over-current, overload and start-up fail protection
- Motor Control Application Tuning (MCAT) tool (see References)
  - The MCAT tool is a user-friendly graphical FreeMASTER's plug-in tool for debugging and tuning of motor-control applications

3 PMSM field oriented control

3.1 Fundamental principle of PMSM FOC

High-performance motor control is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/deceleration. To achieve such control, vector control techniques are used for PM synchronous motors. The vector control techniques are usually also referred to as field-oriented control (FOC).

The FOC concept is based on an efficient torque control requirement, which is essential for achieving a high control dynamic. Analogous to standard DC machines, AC machines develop maximal torque when the armature current vector is perpendicular to the flux linkage vector. Thus, if only the fundamental harmonic of stator-mmf is considered, the torque T_e developed by an AC machine, in vector notation, is given by:

$$T_e = \frac{3}{2} \cdot \rho\rho \cdot \bar{\psi}_s \times \bar{i}_s$$

Equation 1. Electric motor torque

where pp is the number of motor pole-pairs, $\bar{i}_s$ is stator current vector and $\overline{\Psi}_s$ represents vector of the stator flux. Constant 3/2 indicates a non-power invariant form of transformation used.

In instances of DC machines, the requirement to have the rotor flux vector perpendicular to the stator current vector is satisfied by the mechanical commutator. Because there is no such mechanical commutator in AC Permanent Magnet Synchronous Machines (PMSM), the functionality of the commutator has to be substituted electrically, by enhanced current control. This suggests orientating the stator current vector in so that the component of stator current magnetizing the machine (flux component) is isolated from the torque producing component.

This can be accomplished by decomposing the current vector into two components projected in the reference frame, often called the dq frame, that rotates synchronously with the rotor. It has become standard to position the dq reference frame such that the d-axis is aligned with the position of the rotor flux vector, so that the current in the d-axis will alter the amplitude of the rotor flux linkage vector. The reference frame position must be updated so that the d-axis will be always aligned with the rotor flux axis.

Because the rotor flux axis is locked to the rotor position, when using PMSM machines, a mechanical position transducer can be utilized to measure the rotor position and the position of the rotor flux axis. When the reference frame phase is set such that the d-axis is aligned with the rotor flux axis, the current in the q-axis represents solely the torque producing current component.

What further results from setting the reference frame speed to be synchronous with the rotor flux axis speed is that both d and q axis current components are DC values. This implies utilization of simple current controllers to control the demanded torque and magnetizing flux of the machine, thus simplifying the control structure design.

Figure 1 shows the basic structure of the vector control algorithm for the PM synchronous motor. To perform vector control, it is necessary to perform these steps:

- Measure the motor quantities (phase voltages and currents, rotor speed and position).
- Transform them into the two-phase system (α, β) using a Clarke transformation.
- Transform stator currents into the d, q reference frame using a Park transformation.

Also keep these points in mind:

- The stator current torque (i_{sq}) and flux (i_{sd}) producing components are separately controlled.
- The output stator voltage space vector is calculated using the decoupling block.
- The stator voltage space vector is transformed by an inverse Park transformation back from the d, q reference frame into the two-phase system fixed with the stator.
- The output three-phase voltage is generated using a space vector modulation.

To be able to decompose currents into torque and flux producing components (i_{sd}, i_{sq}), position of the motor-magnetizing flux has to be known. This requires accurate sensing of the rotor position and velocity. Incremental encoders or resolvers attached to the rotor are naturally used as position transducers for vector control drives.

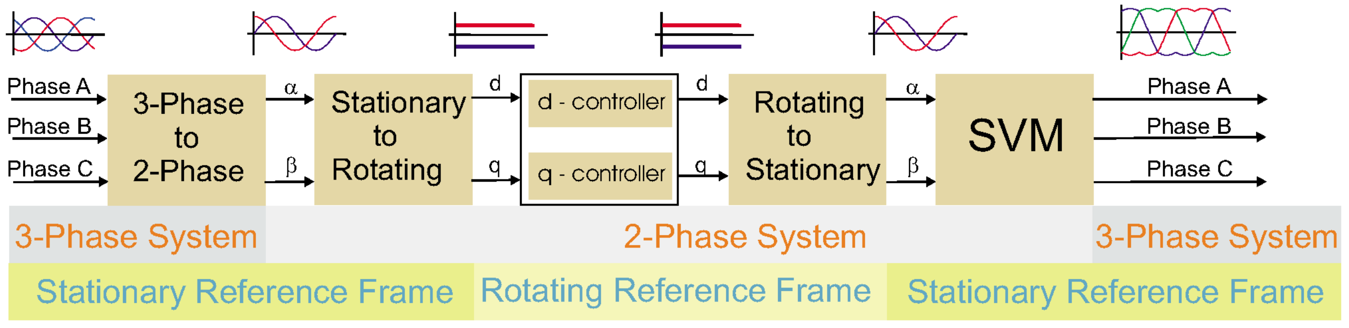


Figure 1. Field oriented control transformations

3.2 PMSM model in quadrature phase synchronous reference frame

Quadrature phase model in synchronous reference frame is very popular for field oriented control structures, because both controllable quantities, current and voltage, are DC values. This allows to employ only simple controllers to force the machine currents into the defined states. Furthermore full decoupling of the machine flux and torque can be achieved, which allows dynamic torque, speed and position control.

The equations describing voltages in the three phase windings of a permanent magnet synchronous machine can be written in matrix form as follows:

$$\begin{bmatrix} u_A \\ u_B \\ u_C \end{bmatrix} = R_s \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix}$$

Equation 2. Three phase voltage model of AC motor

where the total linkage flux in each phase is given as:

$$\begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix} = \begin{bmatrix} L_{aa} & L_{ab} & L_{ac} \\ L_{ba} & L_{bb} & L_{bc} \\ L_{ca} & L_{cb} & L_{cc} \end{bmatrix} \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \psi_{PM} \begin{bmatrix} \cos(\theta_e) \\ \cos(\theta_e - (2\pi)/3) \\ \cos(\theta_e + (2\pi)/3) \end{bmatrix}$$

Equation 3. PMSM total linkage flux matrix

where L_{aa}, L_{bb}, L_{cc}, are stator phase self inductances and L_{ab}=L_{ba}, L_{bc}=L_{cb}, L_{ca}=L_{ac} are mutual inductance between respective stator phases. The term ψ_{PM} represents the magnetic flux generated by the rotor permanent magnets, and θ_e is electrical rotor angle.

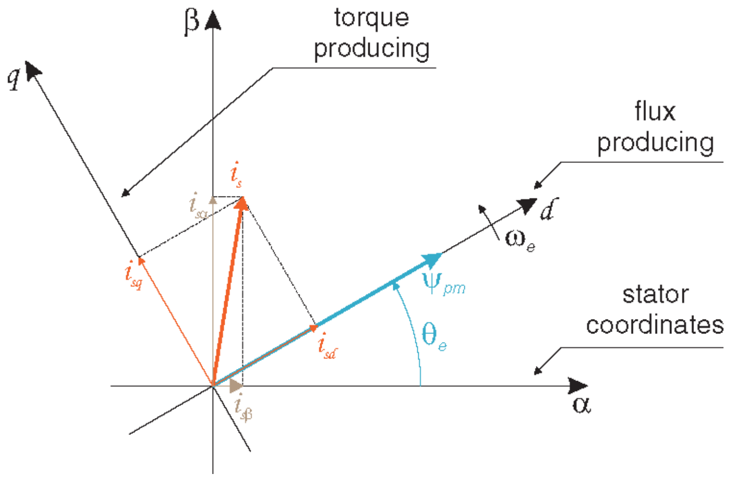


Figure 2. Orientation of stator(stationary) and rotor(rotational) reference frames, with current components transformed into both frames.

The voltage equation of the quadrature phase synchronous reference frame model can be obtained by transforming the three phase voltage equations (Equation 1 on page 2, Equation 2 on page 3) into a two phase rotational frame which is aligned and rotates synchronously with the rotor as shown in Figure 2. Such transformation, after some mathematical corrections, yields the following set of equations:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} + \omega_e \begin{bmatrix} & -L_q \\ L_d & \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \psi_{PM} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation 4. DQ voltage model of PMSM

It can be seen that Equation 4 on page 4 represents a non-linear cross dependent system, with cross-coupling terms in both d and q axis and back-EMF voltage component in the q-axis. When FOC concept is employed, both cross-coupling terms shall be compensated in order to allow independent control of current d and q components. Design of the controllers is then governed by following pair of equations, derived from Equation 4 on page 4 after compensation:

$$u_d = R_s i_d + L_d \frac{di_d}{dt}$$

Equation 5. D axis voltage

$$u_q = R_s i_q + L_q \frac{\mathrm{d}i_q}{\mathrm{d}t}$$

Equation 6. Q axis voltage

which describes the model of the plant for d and q current loop. It is obvious that both Equation 5 on page 4 and Equation 6 on page 4 are structurally identical, therefore the same approach of controller design can be adopted for both d and q controllers. The only difference is in values of d and q axis inductances, which results in different gains of the controllers. Considering closed loop feedback control of a plant model as in Equation 5 on page 4 or Equation 6 on page 4, using standard PI controllers, then the controller proportional and integral gains can be derived, using a pole-placement method, as follows:

$$K_p = 2\varsigma\,\omega_0 L - R$$

Equation 7. Proportional gain equation

$$K_I = \omega_0^2 L$$

Equation 8. Integral gain equation

where ω_0 represents the system natural frequency [rad/sec] and ζ is the Damping factor [-] of the current control loop.

3.3 Phase current measurement

The 3-phase voltage source inverter depicted in Figure 3 uses three shunt resistors (R32, R31 and R32) placed in each of the inverter leg as phase current sensors. Stator phase current which flows through the shunt resistor produces a voltage drop which is interfaced to the AD converter of microcontroller through conditional circuitry (refer to “3-Phase BLDC/PMSM Low Voltage Power Stage - User Manual”).

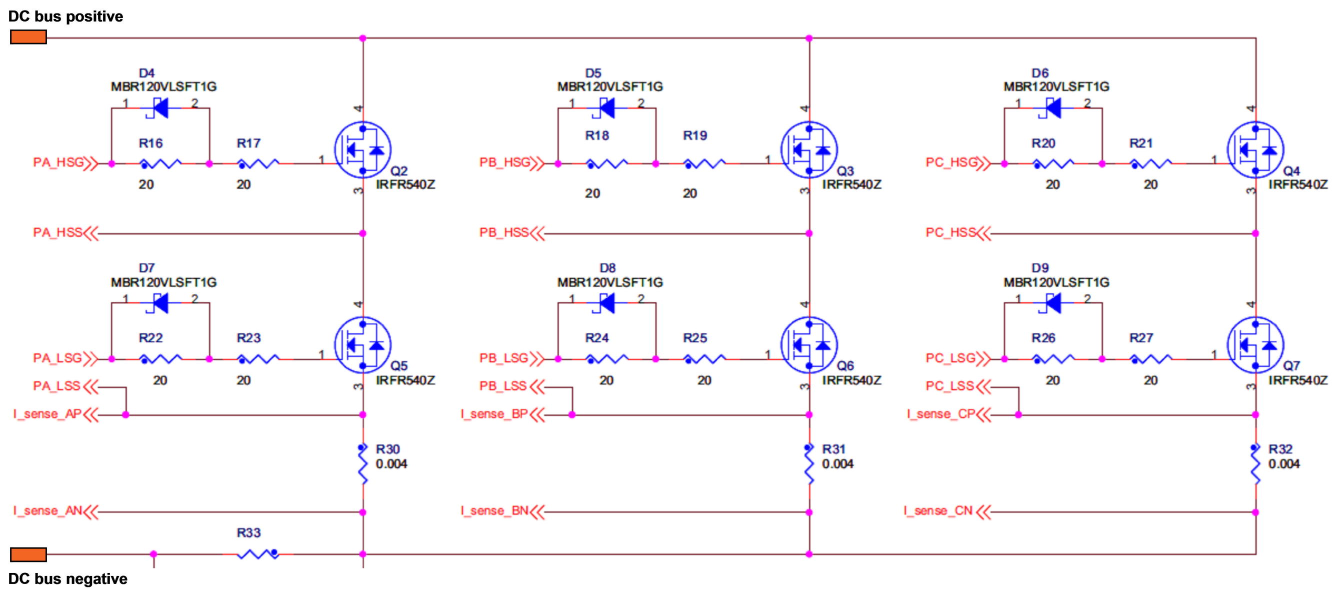


Figure 3. 3-phase DC/AC inverter with shunt resistors for current measurement

Figure 4 shows an operational amplifier and input signal filtering circuit which provides the conditional circuitry and adjusts voltages to fit into the ADC input voltage range.

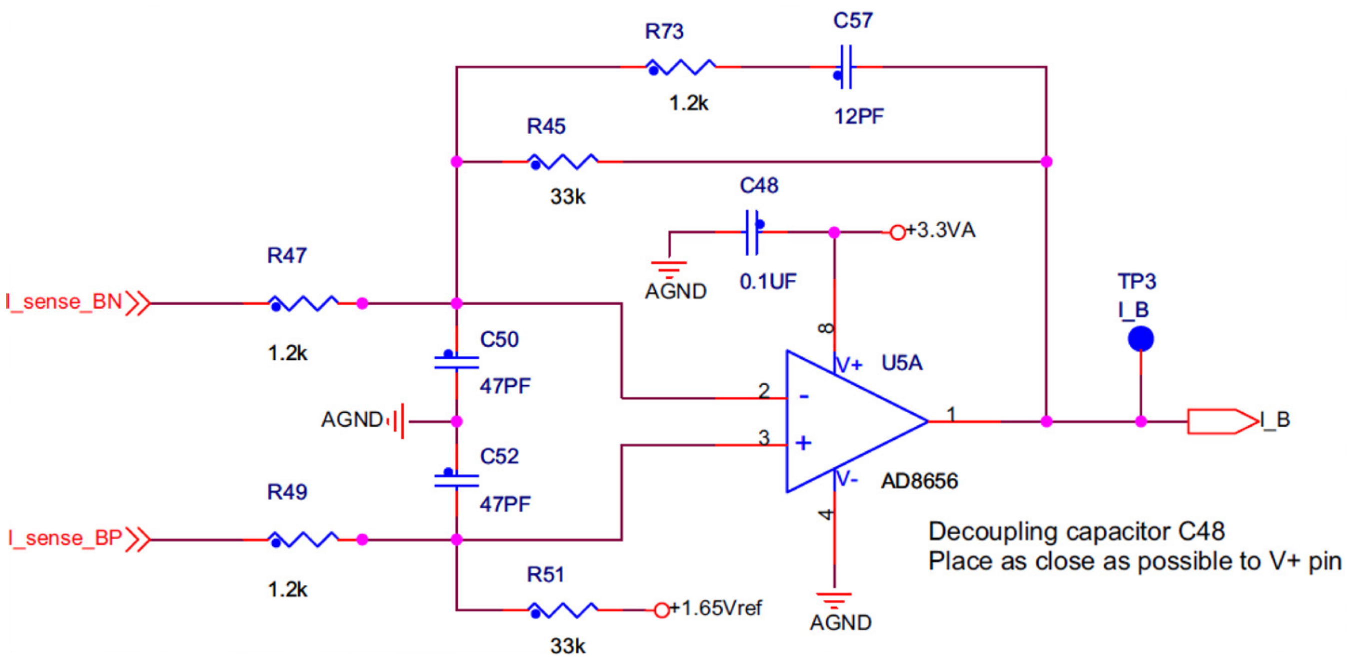


Figure 4. Phase current measurement conditional circuitry

The phase current sampling technique is a critical issue for detection of phase current differences and for acquiring full three phase information of stator current by its reconstruction. Phase current flowing through shunt resistors produces a voltage drop which needs to be appropriately sampled by the AD converter when low-side transistors are switched on. The current cannot be measured by the current shunt resistors at an arbitrary moment. This is because that the current only flows through the shunt resistor when the bottom transistor of the respective inverter leg is switched on. Therefore considering the diagram depicted in Figure 3, phase A current is measured using the R30 shunt resistor and can only be sampled when the transistor Q5 is switched on. Correspondingly, the current in phase B can only be measured if the transistor Q6 is switched on, and the current in phase C can only be measured if the transistor Q7 is switched on. In order to get an actual instant of current sensing, voltage waveform analysis has to be performed.

Generated duty cycles (phase A, phase B, phase C) of two different PWM periods are depicted in Figure 5. These phase voltage waveforms correspond to a center-aligned PWM with sinewave modulation. As shown in the following figure, (PWM period I), the best sampling instant of phase current is in the middle of the PWM period, where all bottom transistors are switched on. However, not all three currents can be measured at an arbitrary voltage shape. PWM period II in the following figure shows a case when the bottom transistor of phase A is on for a very short time. If the on time is shorter than a certain critical time (depends on h/w design), the current cannot be correctly measured.

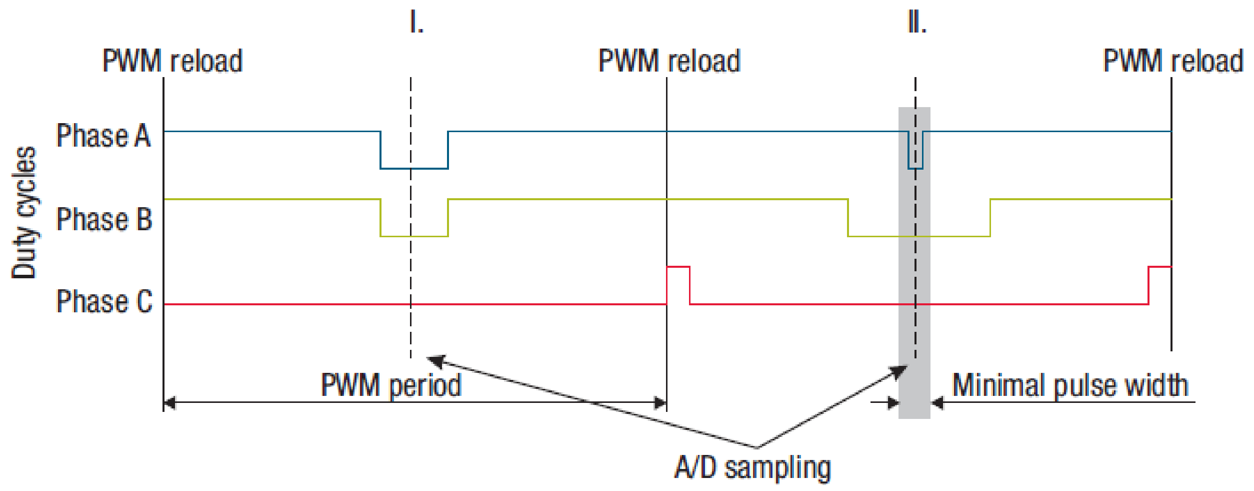


Figure 5. Generated phase duty cycles in different PWM periods.

In case of standard motor operation where the supplied voltage is generated using the space vector modulation, the sampling instant of phase current takes place in the middle of the PWM period in which all bottom transistors are switched on. If the modulation index of applied SVM technique increases there is an instant when one of the bottom transistors is switched on for a very short time period. Therefore, only two currents are measured and the third one is calculated from equation:

$$i_A + i_B + i_C = 0$$

Equation 9. Currents in three phase balanced system

Therefore, a minimum on time of the low-side switch is required for three phase current reconstruction.

4 MPC5604P- CB configuration

The PMSM Speed Field Oriented Control framework application software and hardware is designed to meet the following technical specifications:

- MPC5604P Controller Board is used (refer to dedicated user manual for MPC5604P Controller Board)
- 3-phase low voltage power stage with MC33937 pre-driver is used
  - refer to dedicated user manual for 3-phase low voltage power-stage with MC33937 pre-driver. See References for more information.
  - refer to dedicated user manual for MC33937 pre-driver on www.freescale.com
  - refer to dedicated document describing communication with and configuration of MC33937 pre-driver using DSPI communication bus
- PWM output frequency = 20 kHz
- current loop sampling period = 100 µs
- speed loop sampling period = 2 ms
- 3-phase current measurement using three shunt resistors on bottom side of each inverter leg. Phase current measurement feedback is routed to ADC0 and ADC1 as follows:
  - phase A current: ADC0/1 - CH12
  - phase B current: ADC0/1 - CH13
  - phase C current: ADC0/1 - CH14
- DC bus voltage measurement routed to ADC1 as follows:
  - DC bus voltage: ADC1 - CH0
- encoder position feedback routed to eTimer1 module as follows:
  - phase A: eTimer1 - CH1 input
  - phase B: eTimer1 - CH2 input

- resolver excitation signal routed from eTimer0 - CH5
- resolver position feedback routed to ADC0 and ADC1 as follows:
  - resolver sin: ADC0 - CH1
  - resolver cos: ADC1 - CH1

4.1 FlexPWM

The MPC5604P Clock Generation Module is configured to generate a clock signal of 120 MHz on the MC_PLL_CLK bus. The FlexPWM module is clocked from the MC_PLL_CLK, therefore it is placed behind the IPS Bus Clock Sync Bridge.

The FlexPWM sub-module #0 is configured to run as a master and to generate Master Reload Signal (MRS) and counter synchronization signal (master sync) for other sub-modules. The MRS signal is generated every second opportunity of sub-module #0, VAL1 compare, i.e. full cycle reload. All double buffered registers, including compare registers VAL2, VAL3, VAL4, VAL5 are updated on occurrence of MRS, therefore update of new PWM duty cycles is done every two PWM periods.

FlexPWM modulo counting, for generation of centrr-aligned PWM signals, is achieved by setting VAL0 register to zero and INIT register to negative value of VAL1. Considering PWM clock of 120 MHz, required PWM output 20 kHz and PWM reload period 100 µs, then INIT, VAL0 and VAL1 registers of sub-modules 0,1,2 are set as follows:

- INIT = -120000000/20000/2 = -3000 DEC = 0xF448
- VAL0 = 0 DEC
- VAL1 = - INIT = 3000 DEC = 0x0BB8

Reload frequency of sub-modules 0,1,2 is set to "Every two opportunities" and "Full cycle reload" is enabled in all sub-modules. Because sub-module #0 is a master that generates MRS signal, reload of double buffered registers of sub-module #0 is done on "Local Reload". Sub-modules 1 and 2 are slaves, so reload of their double buffered registers is done on "Master Reload", broadcast from sub-module #0. Similarly, the sub-module #0 counter is initialized on "Local Sync" event, while sub-modules 1 and 2 on "Master Sync" event, which is also broadcast from sub-module #0.

Because some registers are double buffered on occurrence of FORCE OUT signal, all sub-modules have "Local Reload" event selected as force source.

All PWM channels are used to drive a 3-phase DC/AC inverter, so each PWM pair is driven in complementary mode, with dead-time automatically added on each rising edge of respective PWM signal. Used power stage with MC33937 pre-driver inverts the polarity of PWM signals for top transistors (active low logic), so PWM A output polarity in all sub-modules is set as "Inverted". Therefore, during fault state, the output of PWM A of each sub-module is set to logic one.

The FlexPWM module includes a Fault Protection logic, which can control any combination of PWM output pins and automatically disable PWM outputs during a fault state. Faults are generated by a logic one on any of the FAULTx pins. In order to enable mapping of all fault pins, fault disable mapping registers (DISMAP) of all sub-modules must be enabled (logic one).

4.2 CTU

The MRS signal generated from the FlexPWM module is internally routed to the CTU module, where it is selected using the input selection register Trigger Generator Subunit Input Selection Register (TGSISR) as source of master reload signal for CTU. This signal is used for reload trigger compare registers, and reloads the TGS counter with the value stored in TGS counter reload register. The TGS counter register is used to compare the current counter value with the values of trigger compare registers, when the two values are the same an event is generated. TGS is configured in triggered mode.

Because the MRS signal is generated every two PWM periods, the CTU counter can count up to value of 12000DEC when the initial value is set to zero.

The following TGS trigger compare registers are used for trigger events:

- T0CR = 0x0 (=0 DEC)

- [1] T1CR = 0x099C (=2460 DEC)
- [1]T2CR = 0x210C (= 8460 DEC)
- [1]T3CR = 0x2DF0 (=11760)

The CTU Scheduler subUnit (SU) generates the trigger event output according to the trigger event that occurred. The following trigger event outputs are generated:

- ADC command output: T0CR generates ADC command event output, with command offset initially set to one. It is used as synchronization signal to ADC (ADC commands #1 - #12 for phase current and DC bus voltage measurement).
- [2] eTimer0 output: T1CR generates eTimer0 event output, which toggles its output to generate rising edge of resolver exciting signal.T1CR is phase shifted to account for delay caused by the MPC5604P resolver hardware circuitry and to allow ADC sampling of resolver signals just before PWM reload.
- [2]eTimer0 output: T2CR generates eTimer0 event output, which toggles its output to generate falling edge of the resolver exciting signal; frequency of resolver exciting signal is: resolverSignalFreq = MC_PLL_CLK/(2*(T2CR-T1CR)) = 120000000/(2*(8460-2460))=10kHz
- [2]ADC command output: T3CR generates ADC command event output, with command offset set to zero. It is used as synchronization signal to ADC (ADC command #0 for resolver signals sampling)

The SU uses a Commands List in order to select the command to send to the ADC when a trigger event occurs. Each ADC command sent by the CTU into the ADC specifies:

- whether the actual command is a first command of a new stream of consecutive commands or not
- whether an End Of Conversion (EOC) interrupt is issued when conversion specified by the command is finished
- which channels are to be converted for both ADC modules
- the target FIFO register for storing the conversion results

Because the trigger compare register for trigger T0CR is set to zero, it generates the ADC start of conversion request at the beginning of each PWM reload cycle. When a T0CR trigger event occurs, the ADC command selected by the index value T0_INDEX in command list control registers CLCR1 is sent to the ADC.

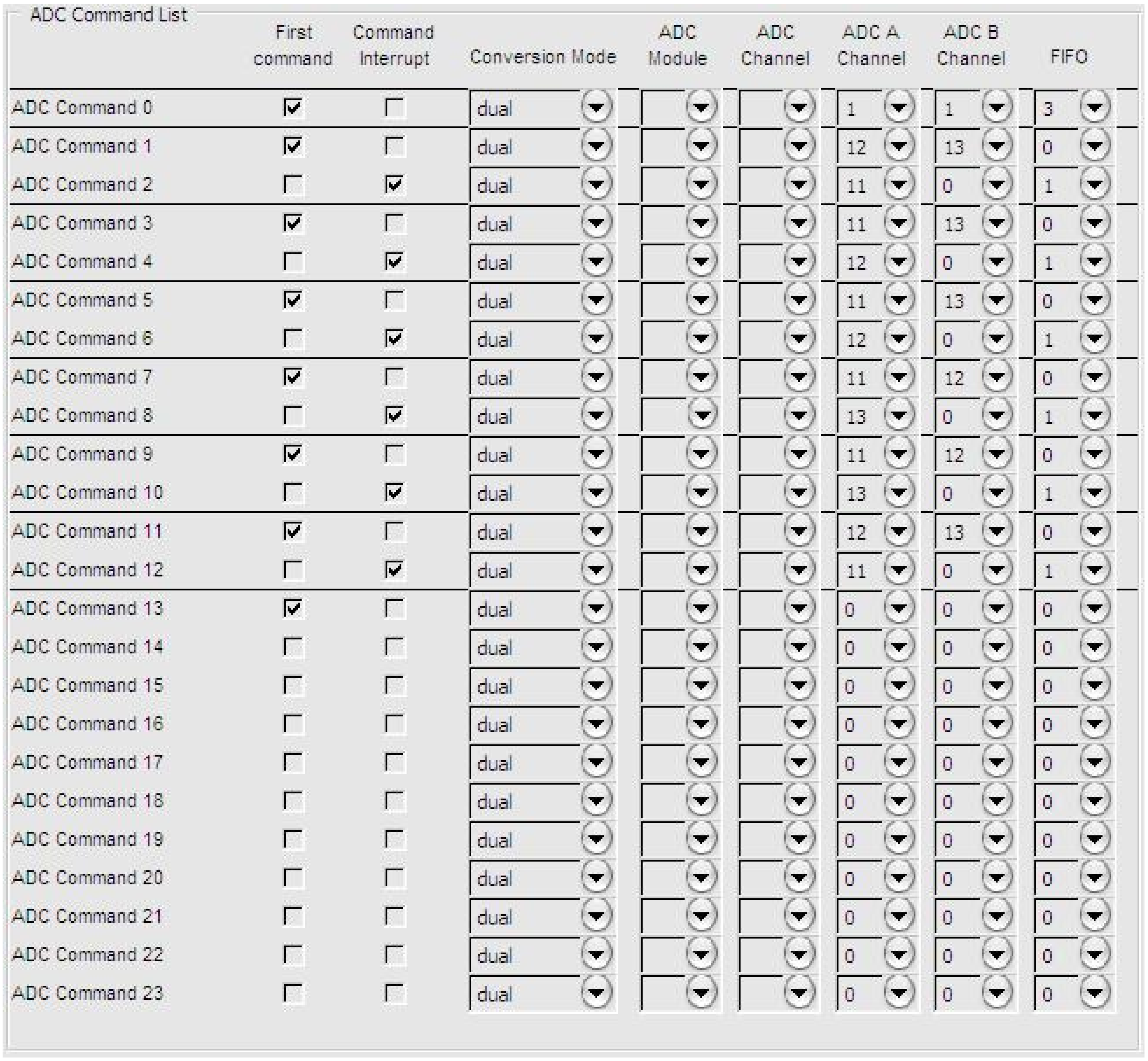
At each T0CR trigger event, two ADC commands are executed in a stream. The first command in a stream specifies two phase currents to be sampled simultaneously (all phase current signals are routed to pins shared between both ADC modules). The second command specifies the third phase current and DC bus voltage to be sampled.

The index pointer to the ADC command list T0_INDEX is updated according to the sector in which resides the actual output voltage vector, calculated by the space vector modulation of the FOC algorithm. There are six sectors within the output voltage hexagon of the inverter, therefore six different ADC command sequences are selected for one full revolution of the voltage vector. This technique is necessary when the phase current measurement is done using three shunt resistors placed in the bottom side of each inverter leg.

Because the shunt resistor is placed at the bottom side of the inverter leg, the phase current can be measured only when bottom transistor is switched on. Because the sum of the three currents in the motor windings is zero, only two currents are measured and the third one is calculated. Which phases are measured and which are calculated changes according to the voltage vector angle, i.e. the phases with the largest PWM on-pulse on the bottom transistors are selected to get the best current information.

Configuration of CTU ADC commands is shown in Figure 6.

1. These triggers are only necessary when using resolver position sensor.
2. These trigger event outputs are only necessary when using resolver position sensor.



ADC Command List

	First command	Command Interrupt	Conversion Mode	ADC Module	ADC Channel	ADC A Channel	ADC B Channel	FIFO
ADC Command 0	✓	☐	dual			1	1	3
ADC Command 1	✓	☐	dual			12	13	0
ADC Command 2	☐	✓	dual			11	0	1
ADC Command 3	✓	☐	dual			11	13	0
ADC Command 4	☐	✓	dual			12	0	1
ADC Command 5	✓	☐	dual			11	13	0
ADC Command 6	☐	✓	dual			12	0	1
ADC Command 7	✓	☐	dual			11	12	0
ADC Command 8	☐	✓	dual			13	0	1
ADC Command 9	✓	☐	dual			11	12	0
ADC Command 10	☐	✓	dual			13	0	1
ADC Command 11	✓	☐	dual			12	13	0
ADC Command 12	☐	✓	dual			11	0	1
ADC Command 13	✓	☐	dual			0	0	0
ADC Command 14	☐	☐	dual			0	0	0
ADC Command 15	☐	☐	dual			0	0	0
ADC Command 16	☐	☐	dual			0	0	0
ADC Command 17	☐	☐	dual			0	0	0
ADC Command 18	☐	☐	dual			0	0	0
ADC Command 19	☐	☐	dual			0	0	0
ADC Command 20	☐	☐	dual			0	0	0
ADC Command 21	☐	☐	dual			0	0	0
ADC Command 22	☐	☐	dual			0	0	0
ADC Command 23	☐	☐	dual			0	0	0

Figure 6. Configuration of CTU ADC commands for the PMSM FOC application

4.3 eTimer0

This eTimer module is only used for generation of an excitation signal for resolver. Because of hardware design, channel #5 of this eTimer module is selected. The purpose is to generate a square wave signal with a frequency of 10 kHz, which is then processed by a hardware low pass filter designed on used controller board. Because of the low pass filter, the resulting harmonic signal is phase shifted.

In order to generate a square wave signal synchronized with the MRS signal, timer channel #5 of module eTimer0 (eTimer1-ETC[5]) is configured as follows:

- counting mode - "Edge of secondary source triggers primary count until compare"
- count direction - "Count up"
- primary source - "IPBus clock"
- secondary source - "AUX #0", which is an output signal from CTU ETIMER #0 trigger event output

- count stop mode - "Count repeatedly"
- count length - "Count until compare then reinitialize"
- compare mode - "Use COMP1 when counting up and COMP2 when counting up"
- output mode - "Toggle OFLAG on successful compare with COMP1 and/or COMP2"
- COMP1 = 0x0
- COMP2 = 0x0
- LOAD = 0x0
- direction of the channel pin - "Output - OFLAG"

4.4 eTimer1

This eTimer module is only used for decoding two square-wave signals from an encoder. Because of hardware design, channel #0 of this eTimer module is selected. The purpose is to decode the two 90°-shifted square wave signals and count up or down all rising/falling edges based on their sequences. The software routine then reads the associated counter value to get the information about rotor position. Reading of the counter value is performed from within `POSPE_GetPositionElEnc()` function, periodically within CTU-ADC interrupt service routine. See the data flow diagram shown in Figure 10.

In order to decode the encoder signals, timer channel #0 of module eTimer1 (eTimer1-ETC[0]) is configured as follows:

- counting mode - "Quadrature count mode, uses primary and secondary sources"
- count direction - "Count up"
- primary source - "Counter #1 input"
- secondary source - "Counter #2 input"
- count stop mode - "Count repeatedly"
- count length - "Count until compare then reinitialize"
- preload control for CNTR
  - "Load CNTR with CMPLD1 upon successful compare with COMP2"
  - "Load CNTR with CMPLD2 upon successful compare with COMP1"
- compare mode - "Use COMP1 when counting up and COMP2 when counting down"
- output mode - "Toggle OFLAG on successful compare with COMP1 and/or COMP2"
- COMP1 = 0x07FF (2047 DEC)
- COMP2 = 0xF800 (-2048 DEC)
- CMPLD1 = 0x07FF (2047 DEC)
- CMPLD2 = 0xF800 (-2048 DEC)
- LOAD = 0x0, this value is updated by a software routine during ALIGN phase

The compare registers of eTimer0 channel #0 are set according to the number of encoder pulses per one mechanical revolution. In this case, an encoder sensor with 1024 pulses is used. Considering quadrature mode, the encoder has a capability of position recognition with a precision that is four times higher than the number of pulses in the application, for a maximum number of edges of 4096. The compare registers are calculated as follows:

- COMP1 = 4096/2 - 1 = 0x07FF (2047 DEC)
- COMP2 = -4096/2 = 0xF800 (-2048 DEC)

4.5 On-chip motor control peripherals interconnection

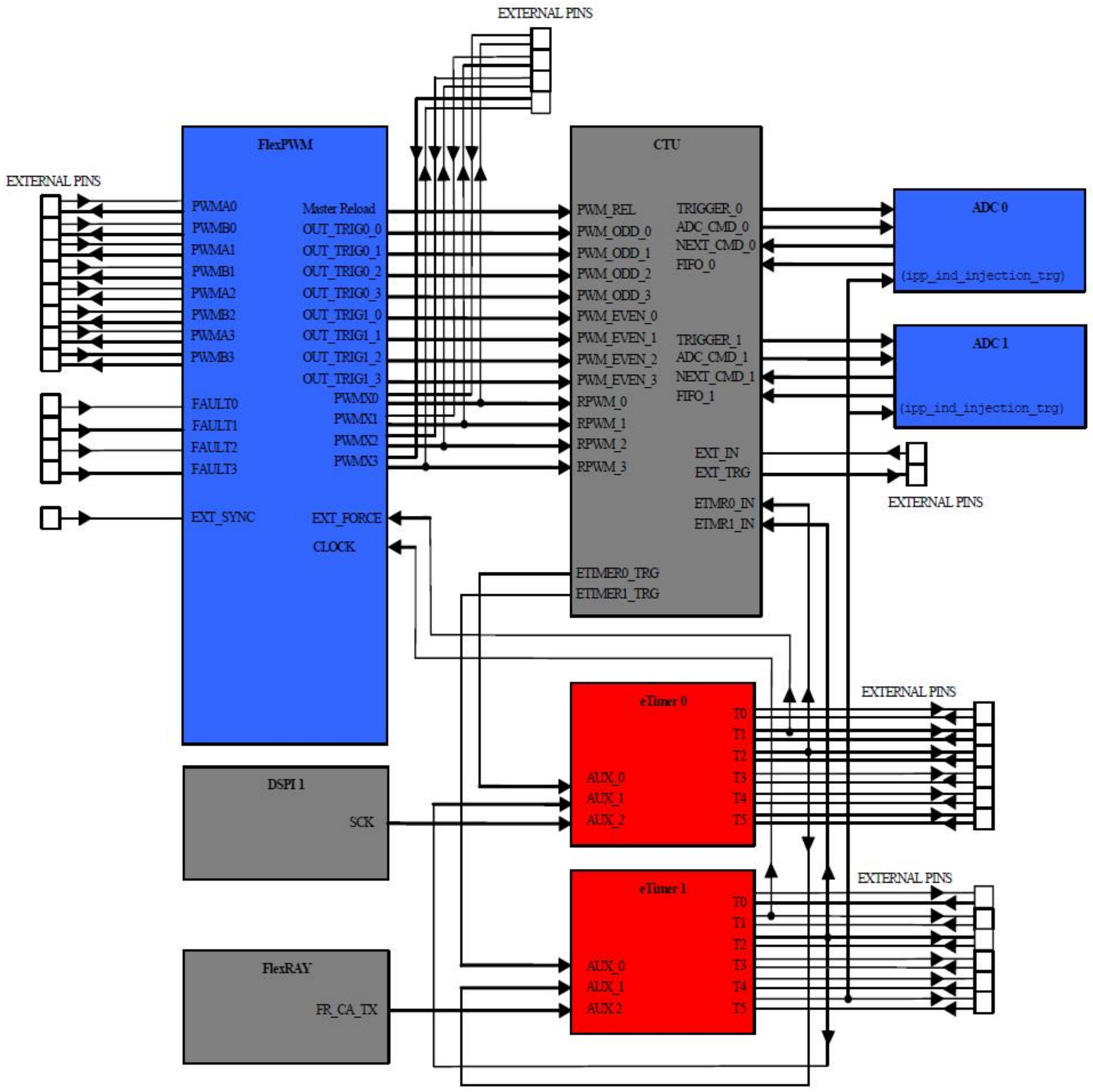


Figure 7. MPC5604P motor control peripherals connection

4.6 ADC conversion and interrupt timing

Configuration of FlexPWM, CTU and eTimer0 peripheral modules, as described in FlexPWM, CTU, and eTimer0, results in a sequence of trigger/events that are shown in the following figure. The application state machine functions are called from an interrupt service routine, which is associated with CTU-ADC command interrupt request. As can be seen from ADC command list configuration, shown in Figure 6, the ADC command interrupts are linked with CTU trigger T0CR, in other words, when measurement of phase currents and DC bus voltage is finished.

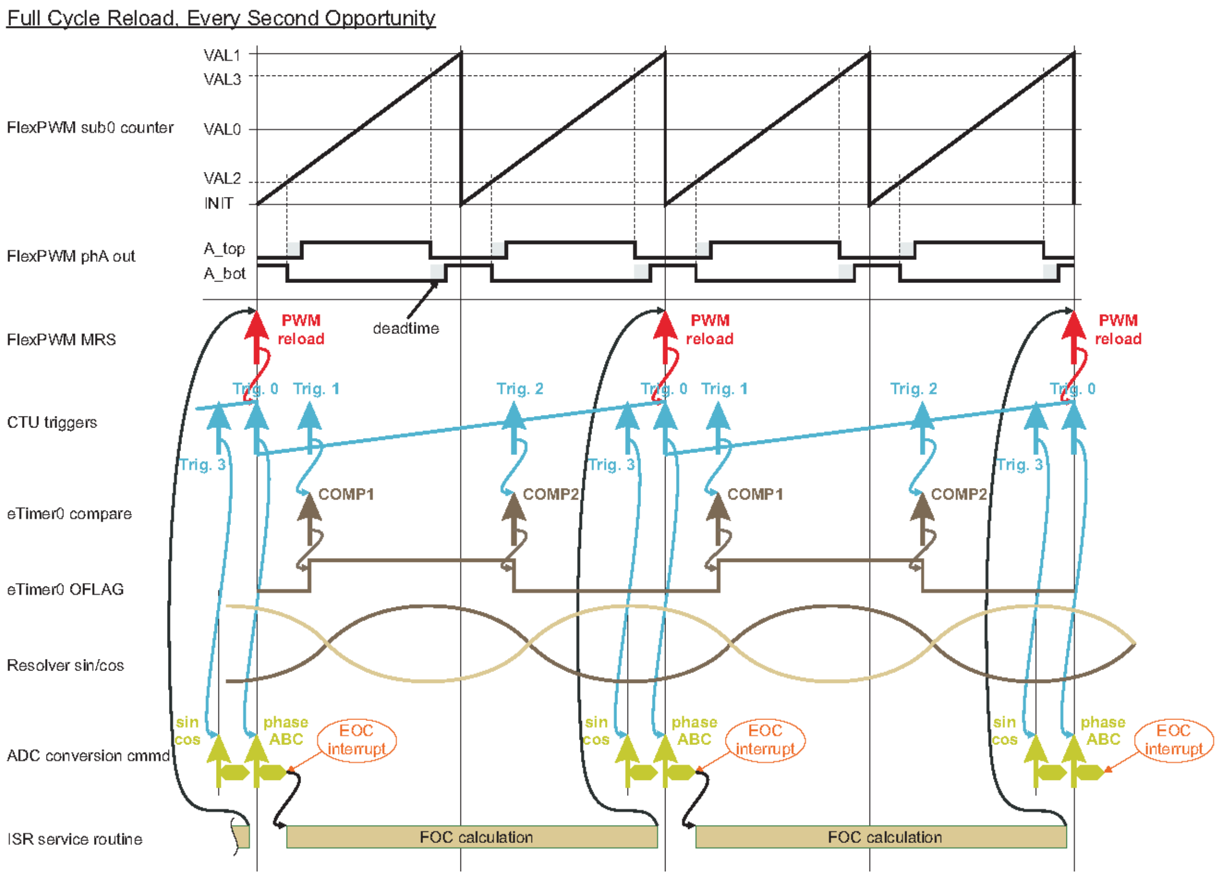


Figure 8. FlexPWM-CTU-ADC conversion timing

5 Software design

5.1 Introduction

This section describes the software design of the PMSM Field Oriented Control framework application. First, the application overview and description of software implementation are provided. Then the numerical scaling in fixed-point fractional arithmetic of the controller is discussed, followed by a detailed description of speed and current sensing. The aim of this chapter is to help in understanding of the designed software.

5.2 Application data flow overview

The application software is interrupt driven running in real time. There is one periodic interrupt service routine associated with the CTU-ADC command interrupt request, executing all motor control tasks. These include both fast, current, and slow speed loop control. All tasks are performed in an order described by the application state machine shown in Figure 11, and application flowcharts shown in Figure 9 and Figure 10.

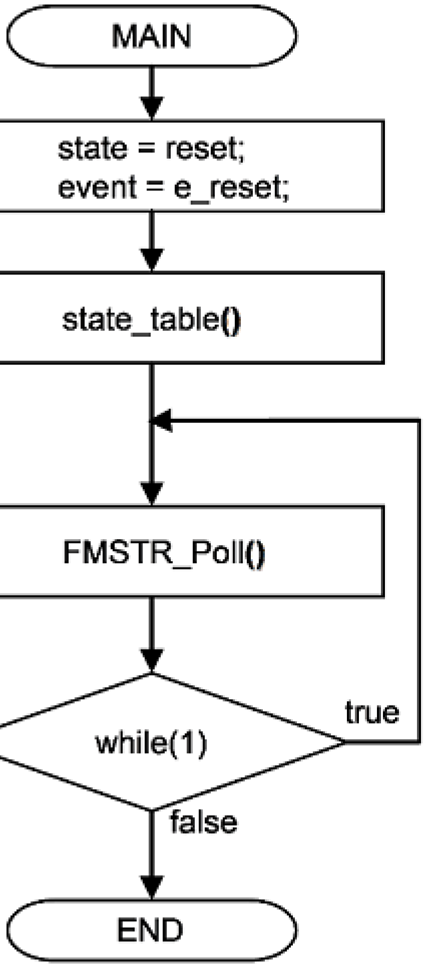


Figure 9. Flow chart diagram of main function with background loop.

To achieve precise and deterministic sampling of analog quantities and to execute all necessary motor control calculations, the state machine functions are called within a periodic interrupt service routine. Hence in order to actually call state machine functions the periphery causing this periodic interrupt must be properly configured and the interrupt enabled. As is described later, all peripherals are initially configured and all interrupts are enabled within a state RESET of the state machine. Therefore state machine is called once in main function before the background loop to enter RESET state and to enable interrupts. As soon as interrupts are enabled and all peripheries are correctly configured (see MPC5604P- CB configuration for configuration of peripherals), the state machine functions are called from the CTU-ADC interrupt service routine. The background loop handles non-critical timing tasks, such as the FreeMASTER communication polling.

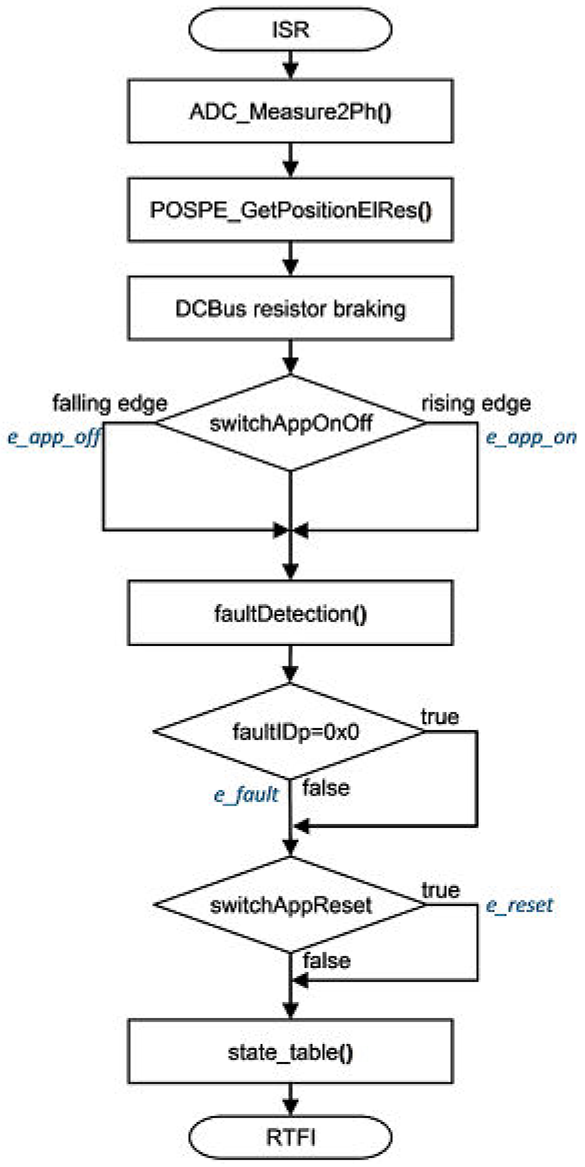


Figure 10. Flow chart diagram of periodic interrupt service routine.

5.3 State machine

The application state machine is implemented using a two-dimensional array of pointers to the functions variable called `state_table[][]()`. The first parameter describes the current application event, and the second parameter describes the actual application state. These two parameters select a particular pointer to state machine function, which causes a function call whenever `state_table[][]()` is called.

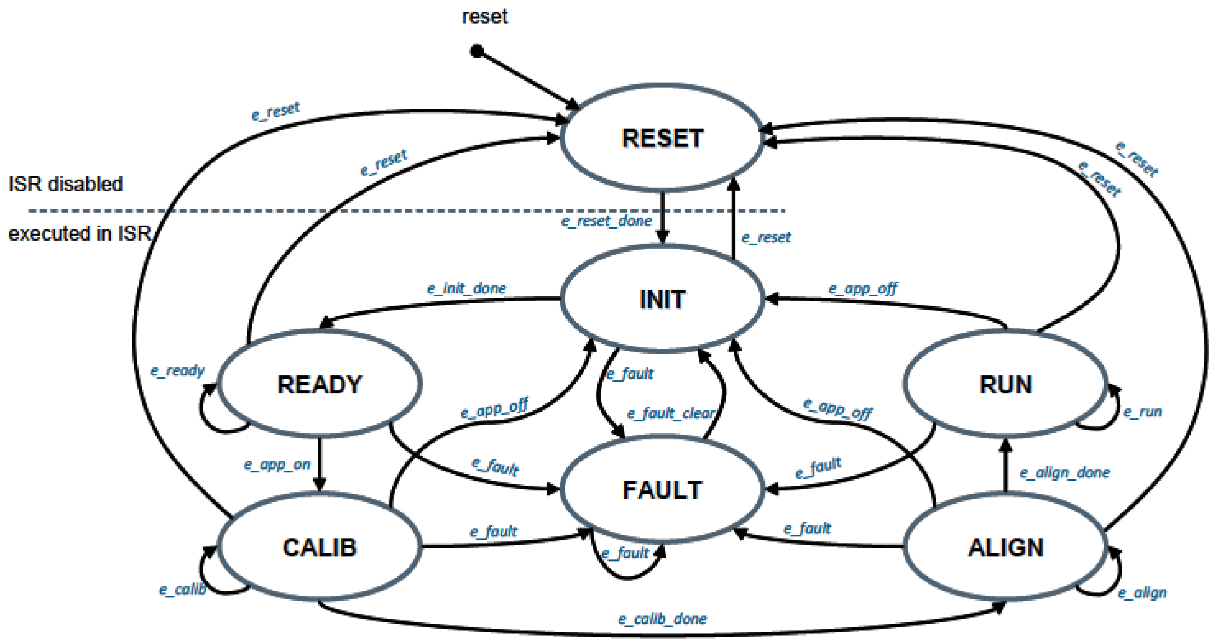


Figure 11. Application state machine

The application state machine consists of following seven states, which are selected using variable `state` defined as `AppStates`:

- RESET - `state = 0`
- INIT - `state = 1`
- FAULT - `state = 2`
- READY - `state = 3`
- CALIB - `state = 4`
- ALIGN - `state = 5`
- RUN - `state = 6`

To signalize/initiate a change of state, thirteen events are defined, and are selected using variable `event` defined as `AppEvents`:

- e_reset - `event = 0`
- e_reset_done - `event = 1`
- e_fault - `event = 2`
- e_fault_clear - `event = 3`
- e_init_done - `event = 4`
- e_ready - `event = 5`
- e_app_on - `event = 6`
- e_app_off - `event = 12`
- e_calib - `event = 7`
- e_calib_done - `event = 8`
- e_align - `event = 9`
- e_align_done - `event = 10`
- e_run - `event = 11`

5.4 State – RESET

State RESET is the first state the state machine enters after power on or reset, in other words, when the application enters main() function. In RESET state, all used peripherals are reset and configured as required by the application (see MPC5604P-CB configuration). Before configuring peripheral modules, all interrupts are disabled. Interrupts are enabled at the end of RESET state. Therefore, the periodic CTU-ADC interrupt is not requested, and the state machine functions cannot be executed until all interrupts are enabled and all peripherals set.

State RESET is a "one pass" function/state. It is entered and executed only once, and the next state is called after RESET is finished. If there is no error/fault during RESET execution, the application event is set to `event=e_reset_done` and all interrupts are enabled at the end of the function. From this point, the CTU-ADC interrupts are enabled, and if the peripherals are correctly configured, the next call of state machine function will be from within the CTU-ADC interrupt service routine.

According to the data flow diagram of the CTU-ADC interrupt service routine, shown in Figure 10, the routine for three phase current reconstruction `ADC_Measure2Ph()` is executed first, followed by the rotor position measurement routine. The fault detection function is always called before the state machine function call, ensuring correct transition to FAULT state in case a fault is detected. If there is no fault detected, the application event remains set to `event=e_reset_done`, hence INIT state will be selected as the next state to execute.

The user can initiate a jump to RESET state from any state of the state machine by setting the event to `event=e_reset`. This is done by setting `switchAppReset` variable to true using FreeMASTER. The entire RESET procedure as described above is then repeated. The following sequence is performed in this order:

- interrupts disable
- all peripherals reset and configure
- user control and fault variables reset

```
switchAppOnOff          = false;
switchAppOnOffState     = false;
switchFaultClear        = false;
switchAppReset          = false;
faultID.R               = 0x0;
faultIDp.R              = 0x0;
```

- event set to `event=e_reset_done`
- interrupts enable

5.5 State – FAULT

The application goes immediately to this state when a fault is detected. The system allows all states to pass into the FAULT state by seting `event = e_fault`. State FAULT is a state that allows transition back to itself if a fault is present in the system and the user does not request clearing of fault flags.

There are two different variables to signal fault occurrence in the application. The actual fault register `faultID` represents the current state of the fault pin/variable etc., and the pending fault register `faultIDp` represents a fault flag, which is set once actual fault is/was true. Even if the actual fault is reset (fault source disappears), the pending fault remains set until manually cleared by the user. Such mechanisms allow for stopping the application and analyzing the cause of failure, even if the fault was caused by a short glitch on monitored pins/variables.

State FAULT can only be left when application variable `switchFaultClear` is manually set to true (using FreeMASTER). That is, the user has acknowledged that the fault source has been removed and the application can be restarted. When the user sets `switchFaultClear = true;` the following sequence is automatically executed:

```
faultID.R           = 0x0;              // Clear Fault register
faultIDp.R          = 0x0;              // Clear Pending Fault register
switchFaultClear    = false;            // Reset fault clearing switch
event               = e_fault_clear;    // new application event
```

Seting event to `event = e_fault_clear` while in FAULT state represents a new request to proceed to INIT state. This request is purely user action and does not depend on actual fault status. In other words, it is up to the user to decide when to set `switchFaultClear` true). However, according to the interrupt data flow diagram shown in Figure 10, function `faultDetection()` is called before state machine function `state_table[event][state]()`. Therefore, all faults will be checked again and if there is any fault condition remaining in the system, the respective bits in `faultID` and `faultIDp` variables will be set. As a consequence of `faultID` and `faultIDp` not equal to zero, function `faultDetection()` will modify the application event from `e_fault_clear` back to `e_fault`, which means jump to fall state when state machine function `state_table[event][state]()` is called. Hence, INIT state will not be entered even though the user tried to clear the fault flags using `switchFaultClear`.

When the next state (INIT) is entered, all fault bits are cleared, which means no fault is detected (faultIDp.R = 0x0) and application variable `switchFaultClear` is manually set to true.

5.5.1 Application faults

Both faultID and `faultIDp` are defined as `AppFaultStatus`, which is a 32 bit long data type. Application faults are bit mapped in `AppFaultStatus` type as follows:

Table 1. AppFaultStatus type

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FLT 31	FLT 30	FLT 29	RESERVED				FLT 24	FLT 23	FLT 22	FLT 21	RESERVED		FLT 18	FLT 17	FLT 16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLT 15	FLT 14	FLT 13	FLT 12	FLT 11	FLT 10	FLT9	FLT8	FLT7	FLT6	FLT5	FLT4	FLT3	FLT2	FLT1	FLT0

- FLT0 - OverDCBusVoltage (Over-voltage on DC bus)
- FLT1 - UnderDCBusVoltage (Under-voltage on DC bus)
- FLT2 - OverDCBusCurrent (Over-current on DC bus)
- FLT3 - OverLoad (Overload Flag)
- FLT4 - MainsFault (Mains out of range)
- FLT5 - WrongHardware (Wrong hardware fault flag)
- FLT6 - OverHeating (Overheating fault flag)
- FLT7 - OverPhaseACurrent (Over-current on phase A)
- FLT8 - OverPhaseBCurrent (Over-current on phase B)
- FLT9 - OverPhaseCCurrent (Over-current on phase C)
- FLT10 - OffCancError (Offset cancellation error)
- FLT11 - MC33937_TLIM (over temperature) - not used in current software version
- FLT12 - MC33937_DESAT (desaturation detected) - not used in current software version
- FLT13 - MC33937_VLS (low VLS detected) - not used in current software version
- FLT14 - MC33937_OC (over current detected) - not used in current software version
- FLT15 - MC33937_PhaseE (phase error) - not used in current software version
- FLT16 - MC33937_FrameE (SPI communication frame error) - not used in current software version
- FLT17 - MC33937_WriteE (SPI communication write error) - not used in current software version
- FLT18 - MC33937_RST (reset event) - not used in current software version
- FLT21 - FOCError (error in FOC calculation function)
- FLT22 - AlignError (error during alignment)
- FLT23 - CalibError (error during ADC calibration)
- FLT24 - InitError (error during app initialization)
- FLT29 - FLEXPWM_Error (error in FlexPWM hardware initialization)

- FLT30 - ADC_Error (error in ADC hardware initialization)
- FLT31 - CTU_Error (error in CTU hardware initialization)

5.6 State – INIT

State INIT is similar to state RESET "one pass" state/function, and can be entered from all states except for READY state, provided there are no faults detected. All application variables and parameters are initialized in state INIT.

After the execution of INIT state, the application event is automatically set to `event=e_init_done`, and state READY is selected as the next state to enter.

5.7 State – CALIB

In this state, ADC DC offset calibration is performed. Once the state machine enters CALIB state, all PWM output are enabled.

Calibration of the DC offset is achieved by generating 50% duty-cycle on the PWM outputs, and taking several measurements of all configured ADC channels. These measurements are then averaged, and the average value for each channel is stored. This value will be subtracted from the measured value when in normal operation. This way the half range DC offset, caused by voltage shift of 1.65V in conditional circuitry (see Figure 4), is removed in all three phases.

State CALIB is a state that allows transition back to itself, provided no faults are present, the user does not request RESET (by switchAppReset=true) or stop of the application (by switchAppOnOff=true), and the calibration process has not finished.

The number of samples for averaging is set by default to 2^8=256, and can be modified in the state INIT.

After all 256 samples have been taken and the averaged values successfully saved, the application event is automatically set to `event=e_calib_done` and state machine can proceed to state ALIGN.

A transition to RESET state is performed by setting the event to `event=e_reset`, which is done automatically when the user sets switchAppReset to true using FreeMASTER.

A transition to FAULT state is performed automatically when a fault occurs.

A transition to INIT state is performed by setting the event to `event=e_app_off`, which is done automatically on falling edge of `switchAppOnOff=false` using FreeMASTER.

5.8 State – ALIGN

This state shows alignment of the rotor and stator flux vectors to mark zero position (only necessary for relative position sensors such as encoders, etc.). When using a relative position sensor such as an encoder, the zero position is not known and therefore, counting of encoder edges has to be correctly reset at the start of operation. This is done in ALIGN state, where a DC voltage is applied in phase A for a certain period. This will cause the rotor to rotate to "align" position, where stator and rotor fluxes are aligned. The rotor position in which the rotor stabilizes after applying this DC voltage is set as zero position, therefore the timer eTimer1-CH0 is reset to zero.

In order to wait for rotor to stabilize in an aligned position, a certain time period is selected during which the DC voltage is constantly applied. The period of time and the amplitude of DC voltage can be modified in INIT state. Timing is implemented using a software counter that counts from a pre-defined value down to zero. During this time, the event remains set to `event=e_align`. When the counter reaches zero, the counter is reset back to the pre-defined value, timer eTimer1-CH0 is reset, and event is automatically set to `event=e_align_done`. This enables a transition to RUN state.

A transition to RESET state is performed by setting the event to `event=e_reset`, which is done automatically when the user sets `switchAppReset` to true using FreeMASTER.

A transition to FAULT state is performed automatically when a fault occurs.

Transition to INIT state is performed by setting the event to `event=e_app_off`, which is done automatically on falling edge of `switchAppOnOff=false` using FreeMASTER.

5.9 State – RUN

In this state, all calculations for FOC algorithm as described in PMSM field oriented control are performed. Calculation of fast current loop is executed every CTU-ADC interrupt when in RUN state while calculation of slow speed loop is executed every Nth CTU-ADC interrupt.

Arbitration is done using a counter that counts from value N down to zero. When zero is reached, the counter is reset back to N and slow speed loop calculation is performed. This way, only one interrupt is needed for both loops and timing of both loops is synchronized. Slow loop calculations are finished before entering fast loop calculations.

Figure 12 shows implementation of FOC algorithm and the functions and variables used. As can be seen from the diagram, position/speed measurement is prepared for Encoder as well as for Resolver sensor. Encoder sensor feedback is selected as default.

A transition to RESET state is performed by setting the event to `event=e_reset`, which is done automatically when the user sets `switchAppReset` to true using FreeMASTER.

A transition to FAULT state is performed automatically when a fault occurs.

A transition to INIT state is performed by setting the event to `event=e_app_off`, which is done automatically on falling edge of `switchAppOnOff=false` using FreeMASTER.

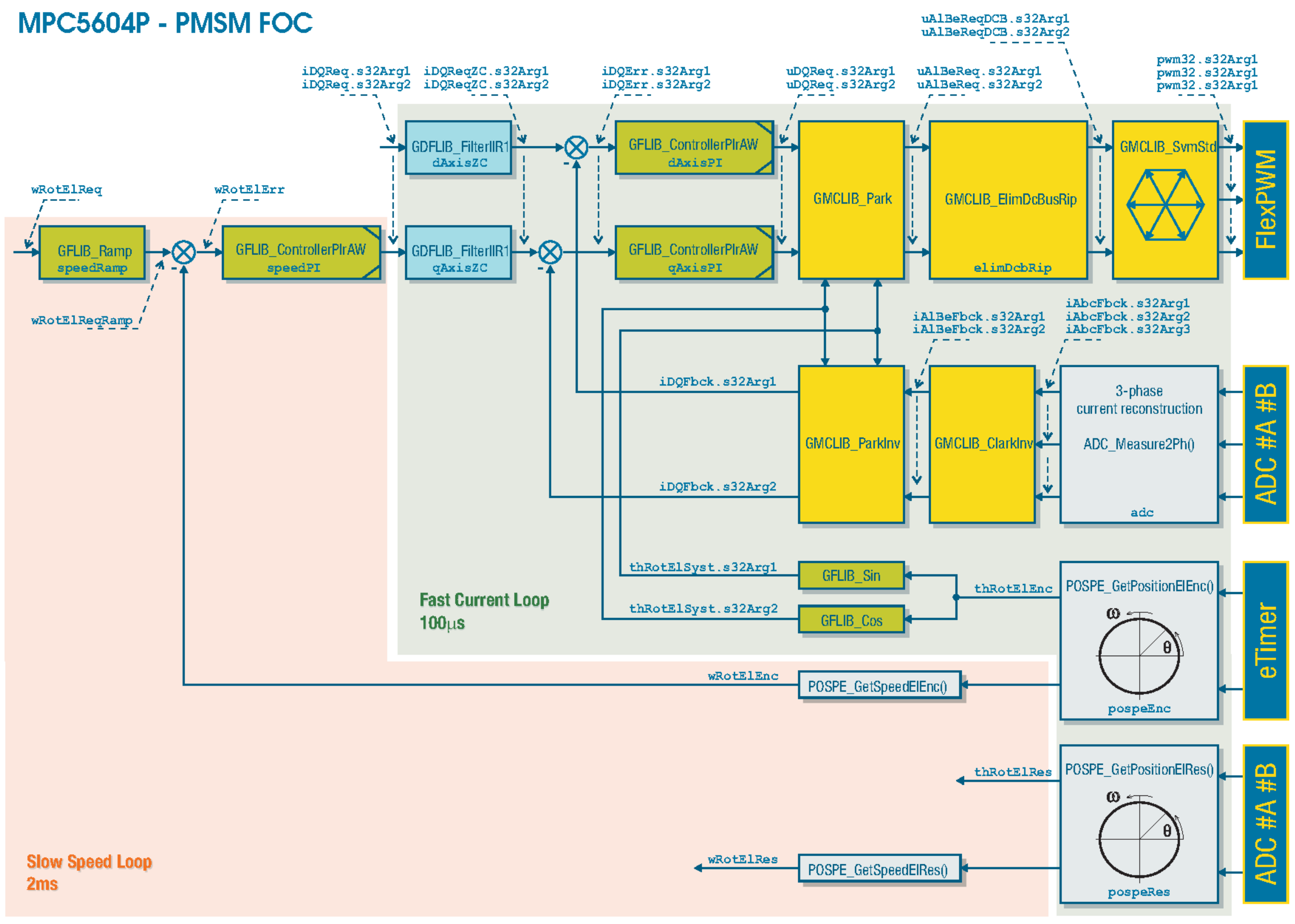


Figure 12. Variables and functions as implemented in FOC calculation

5.9.1 Current measurement

Three phase currents are obtained by calling `ADC_Measure2Ph()` function. As described in Phase current measurement, only two currents are measured at a time, and third current is calculated. Which currents are measured and which calculated depends on a sector in which lies the actual output voltage vector. The sector is calculated by `GMCLIB_SvmStd()` function, which generates three phase duty-cycles for the inverter by employing Space Vector Modulation technique.

The 3-phase inverter can switch six active voltage vectors and two zero vectors. These are given by combinations of the corresponding power switches. Plotting all six active vectors in a complex plane results in a hexagon with six sectors.

5.9.2 Position/speed measurement

Information about rotor position is obtained by calling function `POSPE_GetPosElEnc()` if an encoder is used, or `POSPE_GetPosElRes()` if a resolver is used as position sensor. Both functions calculate the angle tracking observer, which is implemented using PI controller and an integrator.

Output of the integrator represents the estimated (tracked) position, which is subtracted from the measured position (from the sensor), and the resulting difference is used as an error signal for the PI controller. Because the integrator is connected to the PI controller output, the PI controller output represents the estimated angular velocity of the rotor.

To obtain information about the rotor position/speed, the parameters of the tracking observer—such as Kp and Ki gains of PI controller, and input and output scales of the integrator—must be properly configured.

5.9.2.1 Encoder sensor

If an encoder is used as a position sensor, the information about rotor position is obtained by reading the value of the timer eTimer1-CH0 (see eTimer1 for configuration of eTimer1 module).

Because the encoder is a relative position sensor, the counter/timer used for counting the encoder edges has to be correctly reset at the start of operation. This is done in ALIGN state, where a constant current is applied in the d-axis with position manually set to zero. The current amplitude must be chosen large enough to cause rotor movement but must not damage the stator winding. Usually a current amplitude of 10-20% is sufficient. This current is applied for a fixed period of time, in order to allow motor to settle in an "align" position, where stator and rotor fluxes are aligned. This position is then set as zero position, and counter eTimer1-CH0 is set to zero.

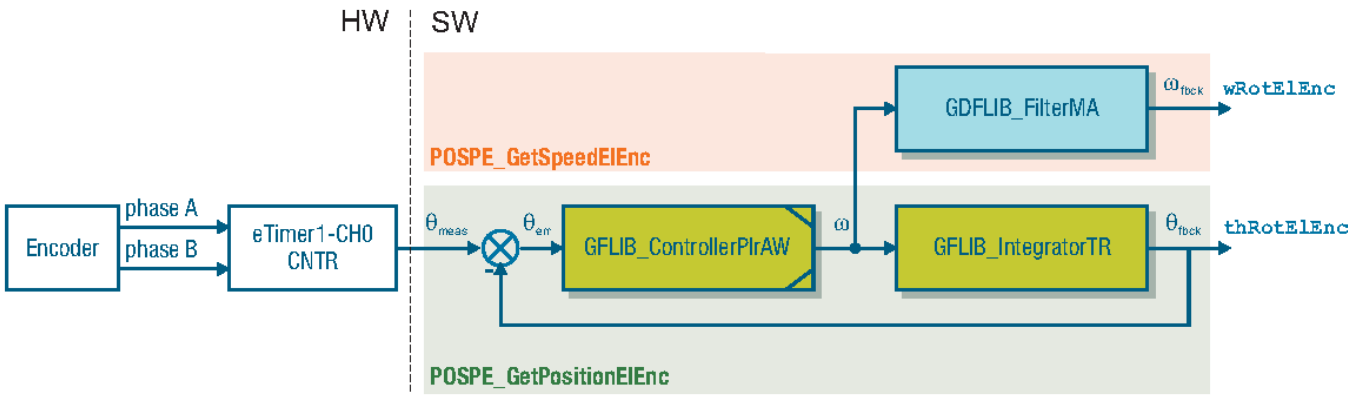


Figure 13. Angle tracking observer used for position/speed estimation using encoder.

5.9.2.2 Resolver sensor

If a resolver is used as a position sensor, the information about rotor position is obtained by reading values of the ADC #A channel 1 and ADC #B channel 1. The timer eTimer0-CH5 is used for generation of an exciting signal for the resolver (see eTimer0 for configuration of eTimer0 module).

The resolver is an absolute position sensor, and there is no need for a mechanical alignment. However, the resolver either has to be mounted precisely on the rotor shaft where the aligned position of rotor and stator fluxes result in resolver sin/cos signals representing zero, or an offset between real zero position and zero position indicated by the resolver sin/cos signals must be a known a priory. This offset is then always subtracted from the measured position.

Initialization of the resolver offset is done at the end of an align procedure (ALIGN state), by writing the value measured from a resolver during alignment into the offset.

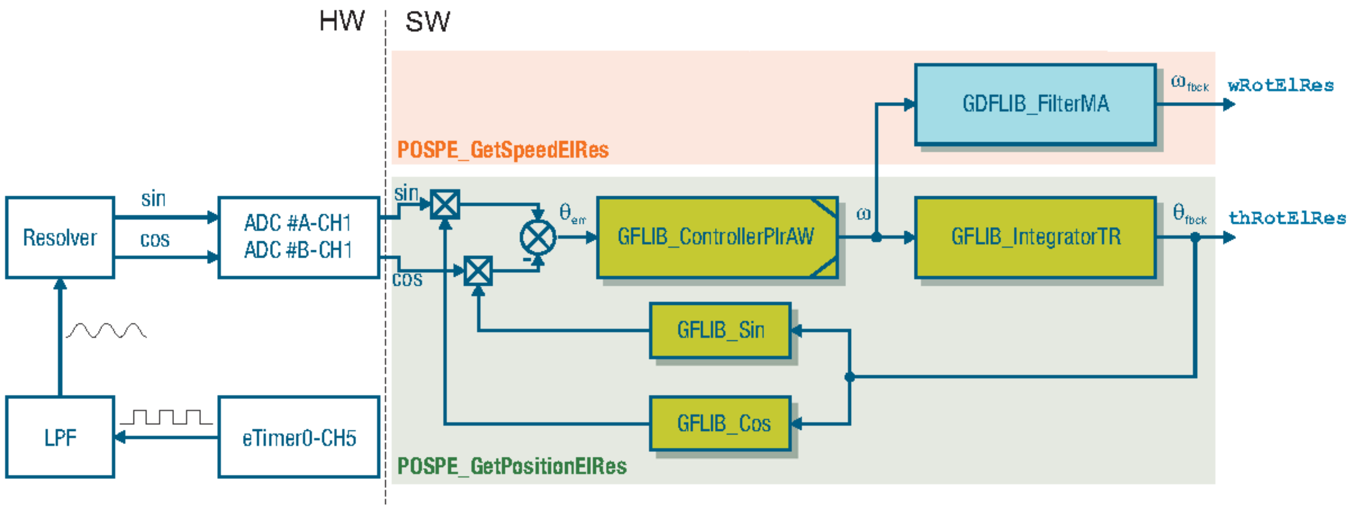


Figure 14. Angle tracking observer used for position/speed estimation using resolver.

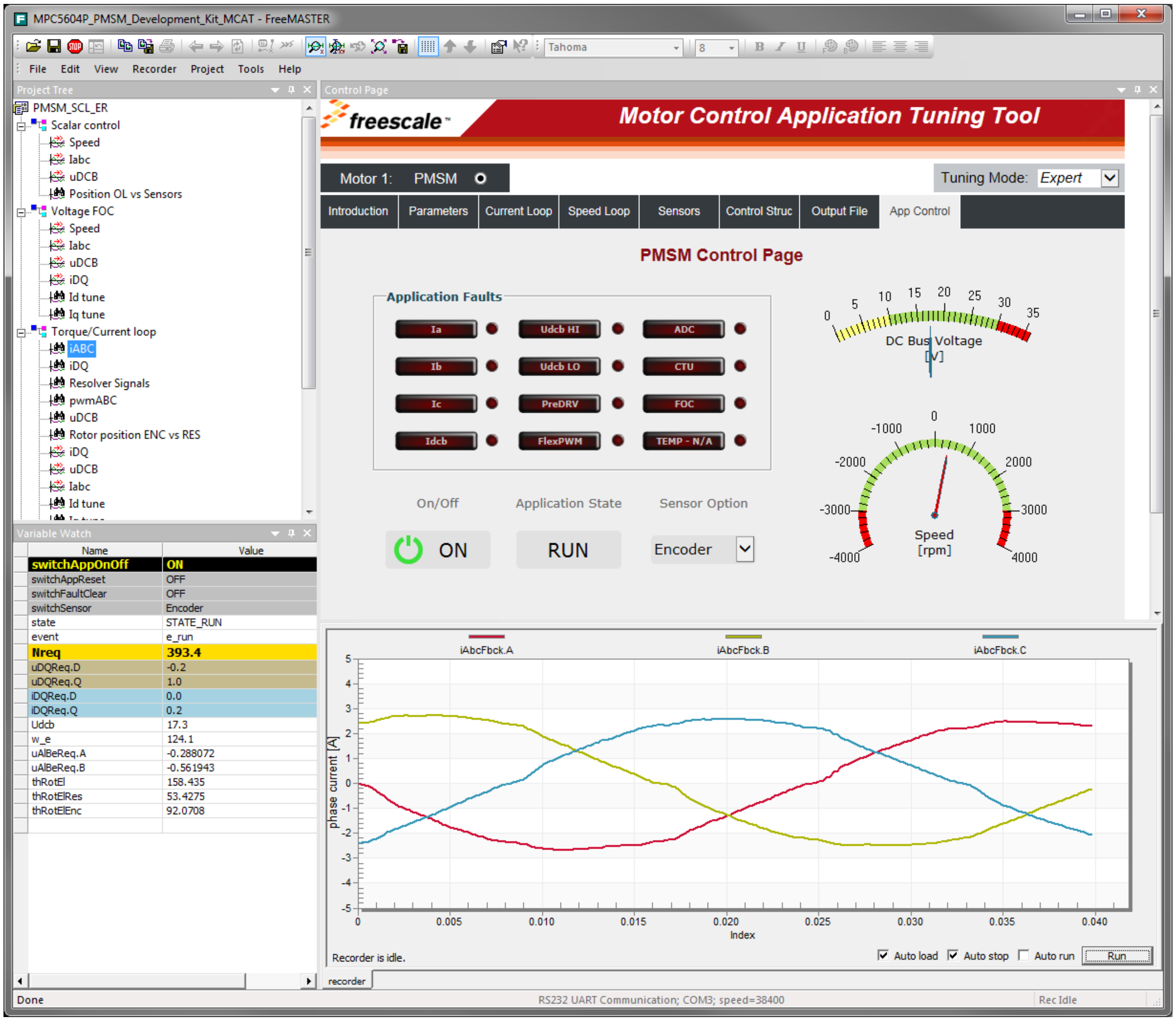
6 Application control user interface

To control the application and monitor variables during run time, the Freescale run-time debugging tool "FreeMASTER" is used (see References).

An example software package of the MPC5604P Development Kit contains a related FreeMASTER project. An integral part of the FreeMASTER project is also the Motor Control Application Tuning (MCAT) tool (see References).

Communication with the host PC is via USB. However, because FreeMASTER supports RS232 communication, there must be a driver installed on the host PC that creates a virtual COM port from the USB. This COM port can then be used for FreeMASTER communication.

The application configures the LINFlex module of MPC5604P for communication speed 38400 bps. Therefore, FreeMASTER must also be set for this speed. This can be done in FreeMASTER menu \Project>Options> by selecting tag Comm.



Variable Watch

Name	Value
switchAppOnOff	ON
switchAppReset	OFF
switchFaultClear	OFF
switchSensor	Encoder
state	STATE_RUN
event	e_run
Nreq	393.4
uDQReq.D	-0.2
uDQReq.Q	1.0
iDQReq.D	0.0
iDQReq.Q	0.2
Udcb	17.3
w_e	124.1
uAlBeReq.A	-0.288072
uAlBeReq.B	-0.561943
thRotEl	158.435
thRotElRes	53.4275
thRotElEnc	92.0708

Figure 15. FreeMASTER Control Page for controlling the application

View all application state machine variables on the FreeMASTER control page (as shown in Figure 15), which is a part of the MCAT tool. The MCAT tool is a user-friendly graphical plug-in tool for FreeMASTER, which enables you to easily tune and control motor-control applications. It supports up to three PMSM motors and is fully compliant with the FOC cascade control structure. The added value of MCAT is the capability to calculate the parameters of the PI controller in the control structure. All application parameters are stored and can be exported as a static configuration header file. For More details about MCAT see AN4642 (see References). Permanent/pending faults are signaled by a highlighted red color bar with name of the fault source. Actual faults are signaled by a round LED-like indicator, which is placed next to the bar with the name of the fault source.

The actual presence of any fault is signaled by highlighting respective indicators. In the previous figure, for example, there are five pending faults and one actual fault ("Udcb LO" - DC bus under-voltage). That means that low voltage on the DC bus is still present in the system.

The other pending faults highlighted in the figure above indicate there was an error latched, but it is no longer present in the system. In this case, the application state FAULT is selected, which is shown by a frame indicator hovering above FAULT state in the middle of the control page.

After all actual fault sources have been removed, no fault indicators are highlighted. The pending faults can now be cleared by pressing the "FAULT CLEAR" button. This will clear all pending faults and will enable transition of the state machine into INIT and then READY state.

Because INIT is a one-pass state, transitions to FAULT-INIT-READY happen faster than the control page can display, which may seem as if the state machine went from FAULT to READY directly. This is not an error, and is caused by slow communication via RS232 and/or slow refresh rate of the control page.

After the application faults have been cleared and the application is in READY state, all variables should be set to their default values. The application can be started by clicking the On/Off button. A successful selection is indicated by highlighting the On/Off button in green.

If there is no fault detected in the system, after starting the application by clicking on the On/Off button, the application should proceed to CALIB state for DC offset calibration, then to ALIGN state for marking the zero position, and finally state RUN is entered.

When in RUN state, all control loops of FOC algorithm are active. That means, 3-phase currents are measured and used to close the current loop, and actual rotor speed is measured for closing the speed loop. The user can now select the desired speed of the rotor in the variable watch window, or by selecting the required speed in "Rotor speed control [rpm]" application gauge on the FreeMaster control page. Click on the speed scale of the gauge to make the selection. The variable for controlling the speed is called Nreq, and it is the required speed recalculated to mechanical speed in revolutions per minute.

Because of the used motor, the required speed can be selected from -4000[rpm] to 4000[rpm].

Field weakening algorithm is not implemented, therefore the required value of d-axis current is set to zero.

6.1 Application quick start

1. Install USB driver to create a virtual COM port for emulation of RS232 communication (for example, "CP210x USB to UART Bridge VCP Drivers" available from https://www.silabs.com/Support%20Documents/Software/CP210x_VCP_Win2K_XP_S2K3.zip)
2. Connect USB cable to MPC5604P controller board and to host PC
3. Connect power supply to the power-stage. Controller board power supply is taken from the power stage. The PMSM motor used is designed for phase voltage = 18V.
4. Start FreeMASTER project located in MPC5604P_PMSM_Development_Kit\FreeMASTER_control \MPC5604P_PMSM_Development_Kit_MCAT.pmp
5. Enable communication by pressing "STOP" button in the toolbar in FreeMASTER, or by pressing "CTRL+K"
6. Successful communication is signaled in the status bar (see Figure 15 for example).
7. GPIOA13 is turned ON at the beginning of the calculation step, and turned OFF at the end. The period of calculation is 100 µs, so the LED D18 on MPC5604P controller board will flash with a period of 10 kHz if the application runs correctly.
8. Functionality of GPIO12 (LED D11 on MPC5604P controller board) is as follows:
   - OFF if the application is in READY, INIT states
   - ON if the application is in RUN, CALIB, ALIGN states
   - flashing if the application is in FAULT state (flashing with period 1 - 2 Hz so that it is clearly visible with the naked eye)
9. If no actual faults are present in the system, all indicators on the FreeMaster control page are dark red. If there is a fault present, identify the source of the fault and remove it. Successful removal is signaled when the respective indicator on the FreeMaster control page turns off.
10. Fault condition is also signaled by blinking LED diode D11 on the MPC5604P controller board. Press UP + DOWN buttons (SW2/SW3 on MPC5604P controller board) simultaneously to clear fault status register once in a FAULT state. The application can be restarted by positioning RUN/STOP switch (SW4 on MPC5604P controller board) to RUN position (transition from STOP to RUN in case the switch was in RUN state when the fault event occurred).
11. If all indicators on the FreeMaster control page are off, clear pending faults by pressing the "FAULT CLEAR" button on the FreeMaster control page, or by pressing UP+DOWN buttons (SW2/SW3 on MPC5604P controller board) simultaneously. The RUN/STOP switch (SW4 on MPC5604P controller board) must be in position STOP.

12. Start the application by clicking the On/Off button on the FreeMaster control page, or by positioning RUN/STOP switch (SW4 on MPC5604P controller board) to RUN position (transition from STOP to RUN in case the switch was in RUN state when the fault event occurred).
13. Enter required speed by assigning this value to "Nreq" variable in the variables watch window, or use the speed gauge in the PMSM Control Page that responds to a mouse click. Value is in revolutions per minute. Alternatively, rotor speed can be increased/decreased by pressing UP/DOWN switches on MPC5604P controller board. RUN/STOP switch (SW4 on MPC5604P controller board) must be in START position.
14. Stop the application by clicking the On/Off button on the FreeMaster control page, or by positioning RUN/STOP switch (SW4 on MPC5604P controller board) to STOP position.
15. RESET the application at any time by entering the value "1" to the switchAppReset variable in the Variable Watch window.

7 References

1. *MPC560xP Controller Data sheet* (document MPC5604P)
2. *MPC560xP Controller Board User's Guide* (document MPC5604PMCBUG)
3. FreeMASTER Run-Time Debugging Tool (www.freescale.com/FREEMASTER)
4. Automotive Math and Motor Control Library Set (www.freescale.com/AutoMCLib)
5. *MC33937 Three Phase Pre-driver Data Sheet* (document MC33937)
6. *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642)
7. Motor Control Application Tuning (MCAT) Tool (www.freescale.com/MCAT)

8 Revision history

This section documents the changes done to this document.

Table 2. Revision history

Revision	Date	Substantive changes
0	10/2012	Initial release.
1	11/2015	Updated Figure 15. Updated Sections 1, 2, 6, and 7.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4561
Revision 1, 11/2015