NXP Semiconductors
Application Note

Document Number: AN12017
Rev. 1, August, 2018

3-Phase PMSM Development Kit with MPC5744P

Featuring Motor Control Application Tuning (MCAT) Tool

by: NXP Semiconductors

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 and resolver position sensing. The design is targeted for automotive motor control (MC) applications.

The proposed design exhibits the suitability and advantages of the MPC5744P microcontroller for motor control applications. It serves as an example of a PMSM control design using the NXP 32-bit MCU built on the Power Architecture™ technology. It meets the highest functional safety standards for automotive and industrial functional safety applications.

System features:

- Modular software concept of a 3-phase PMSM speed Field Oriented Control (FOC) approach.
- Motor phase currents sensing with three shunts
- Rotor position and speed measurement using resolver transducer.
- Application control user interface through FreeMASTER and MCAT tool.

2 System concept

The proposed system, Figure 1. on page 2, is designed to drive the 3-phase PMSM with a resolver sensor. The concept meets the following specifications:

- Targets the MPC5744P microcontroller [1, 2]
- Runs on the MPC5744P Motor Controller Board (CB) with a PCI-Express MC interface [3]
- Runs on the 3-phase Low Voltage Power Stage board with a PCI-Express MC interface [4]
- Incorporates a control technique with:
  - — Vector control of a 3-phase PMSM with resolver sensor
  - — Cascade control structure with current and speed closed loop control
  - — Bi-directional speed operation
  - — Start-up with a rotor alignment procedure

— 100 ms sampling period of all analog quantities (currents, DC-bus voltage, resolver)

— 20 kHz switching frequency of PWM

- FreeMASTER project for real-time debugging and data visualization [5]

— Graphical control page to control the motor (motor start/stop, speed setup)

— Application faults monitoring and visualization

- MCAT tool, AN4642 [6, 7]

— Debugging and tuning the MC applications

— Export the static configuration of MC application to a header file

- DC-Bus over-voltage and under-voltage, over-current, overload, and start-up fail protection
- Included software provides the option for single or dual motor control

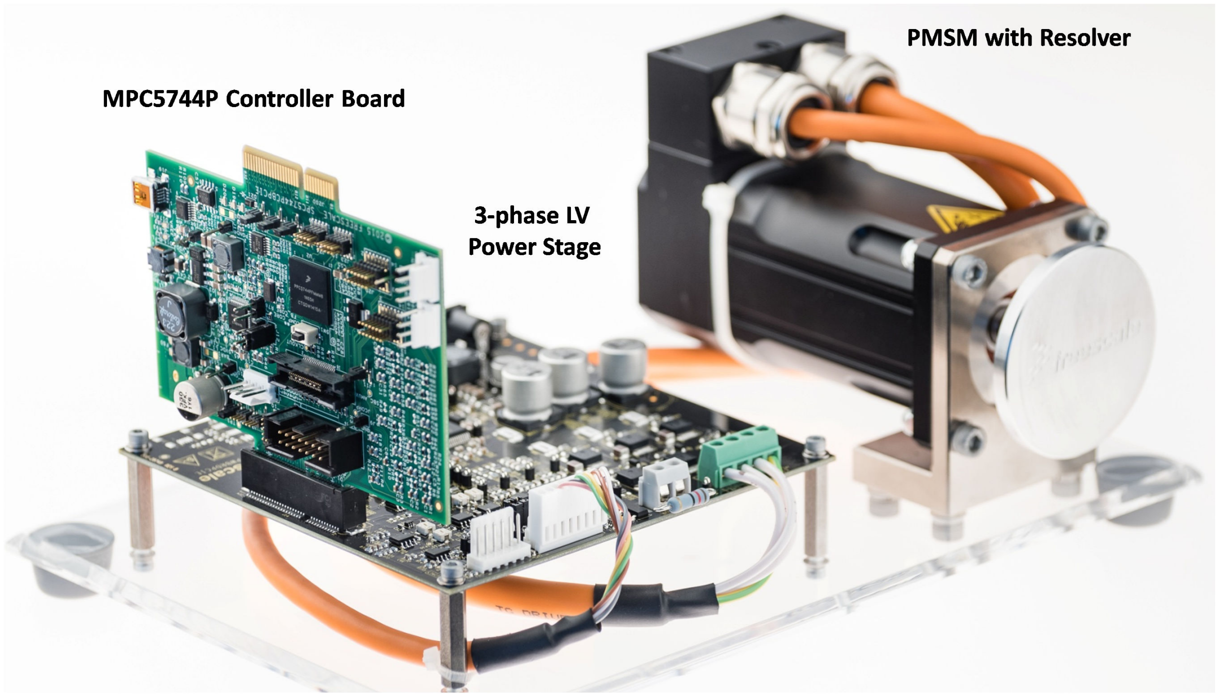


Figure 1. 3-phase PMSM Development Kit with MPC5744P

3 PMSM field-oriented control

This section covers the following sub-sections:

- Fundamental principle of PMSM FOC
- PMSM model in quadrature phase synchronous reference frame
- Phase current measurement
- Resolver signal processing

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 PMSMs. The vector control techniques are usually also referred to as field-oriented control. The FOC concept is based on an efficient torque control requirement, which is essential for achieving high control dynamics. Analogous to standard DC machines, AC machines develop maximal torque when the armature current vector is perpendicular to the flux linkage vector. Therefore, 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:

Equation 1

$$T_e = \frac{3}{2} pp \, \bar{\psi}_S \times \bar{i}_S$$

where *pp* is the number of motor pole-pairs,

$$\bar{i}_S$$

is the stator current vector and

$$\bar{\psi}_S$$

represents the vector of the stator flux. Constant 3/2 indicates a non-power invariant form of transformation used.

In the case of DC machines, the requirement of having 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 machines, the functionality of the commutator has to be substituted electrically, by enhanced current control. This therefore implies orientation of the stator current vector in such a way so as to isolate the component of stator current magnetizing the machine (flux component) 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, which rotates synchronously with the rotor. It has become standard to position the dq reference frame to have the d-axis aligned with the position of the rotor flux vector, so that current in the d-axis alters the amplitude of the rotor flux linkage vector. That requires the reference frame position to be updated such that the d-axis is always aligned with the rotor flux axis. Because the rotor flux axis is locked to the rotor position, for PMSM machines, a mechanical position transducer can be used to measure the rotor position and hence position of the rotor flux axis. Having the reference frame phase set so that the d-axis is aligned with the rotor flux axis, the current in the q-axis represents solely the torque producing current component. Setting the reference frame speed to be synchronous with the rotor flux axis speed results in both d and q axis current components having DC values. This implies utilization of simple current controllers to control the demanded torque and magnetizing flux of the machine, simplifying the control structure design.

Figure 3. on page 4 shows the basic structure of the vector control algorithm for the PMSM. To perform vector control, perform these steps:

- Measure and obtain the motor states, variables and quantities. For example: phase voltages, currents, rotor speed, and position.
- Transform quantities into the two-phase system (αβ) using a Clarke transformation.
- Transform stator currents (αβ) into the dq reference frame using a Park transformation.

Also keep in mind:

- The stator current torque (isq) and flux (isd) 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 3-phase voltage is generated using a space vector modulation.

To be able to decompose currents into torque and flux producing components (isq , isd), 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.

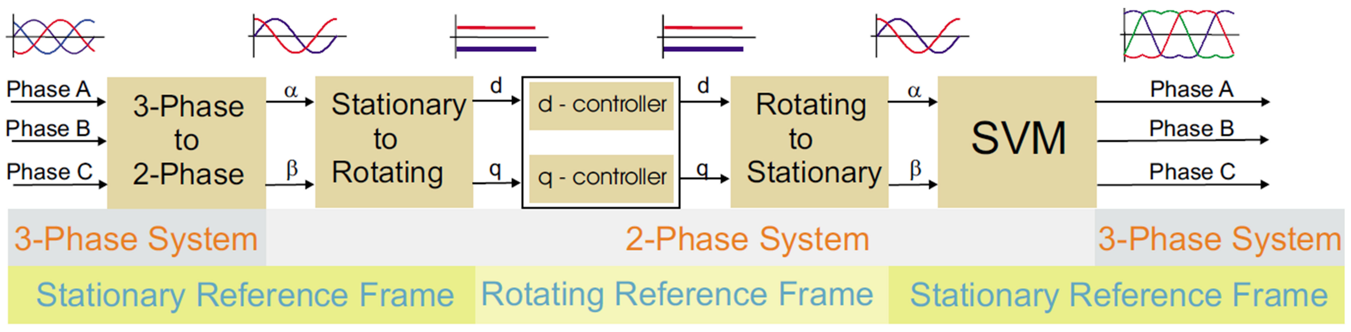


Figure 3. FOC motor phase currents transformations

3.2 PMSM model in quadrature phase synchronous reference frame

Quadrature phase model in synchronous reference frame is very popular for FOC structures because both controllable quantities, current and voltage, are DC values. This allows the use of 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 3-phase windings of a permanent magnet synchronous machine can be written in matrix form as follows:

Equation 2

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

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

Equation 3

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

where:

- L_{aa}, L_{bb}, and L_{cc} are stator phase self inductances
- $L_{ab} = L_{ba}$, $L_{bc} = L_{cb}$, and $L_{ca} = L_{ac}$ are mutual inductance between respective stator phases
- the term

  $$\psi_{PM}$$

  represents the magnetic flux generated by the rotor permanent magnets
- $$\theta_e$$

  is the electrical rotor angle

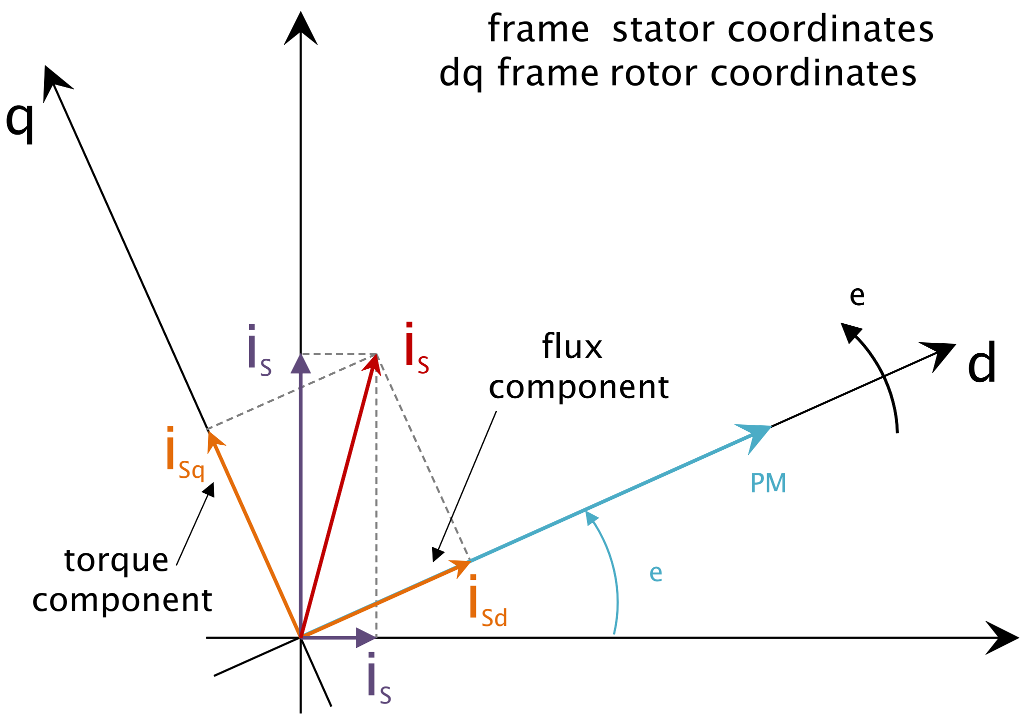


Figure 4. Orientation of stator (stationary) and rotor (rotational) reference frames, with current components transformed into both frames

The voltage equation of the quadrature phase synchronous reference frame model can be obtained by transforming the 3-phase voltage equations (Equation 2, Equation 3) into a 2-phase rotational frame which is aligned and rotates synchronously with the rotor as shown in Figure 4. on page 5 above. Such transformation, after some mathematical corrections, yields the following set of equations:

Equation 4

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

Above equation 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 is employed, both cross-coupling terms shall be compensated 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 after compensation:

Equation 5

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_S \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix}$$

which describes the model of the plant for d and q current loop. Both equations 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 either equation, using standard PI controllers, then the controller proportional and integral gains can be derived, using a pole-placement method, as follows:

Equation 6

$$K_P = 2\zeta\omega_0 L - R$$
$$K_I = \omega_0^2 L$$

where

$$\omega_0$$

represents *Natural Frequency* [rad s^{-1}] and

$$\xi$$

is the *Damping Factor* [-] of the current control loop. To calculate the parameters of current PI controllers an MCAT tool con be used.

3.3 Phase current measurement

The following figure shows how the 3-phase voltage source inverter uses three shunt resistors (R38, R39, and R40) placed in each of the inverter legs as phase current sensors. Stator phase current flows through the shunt resistor, producing a voltage drop which is interfaced to the AD converter of the microcontroller.

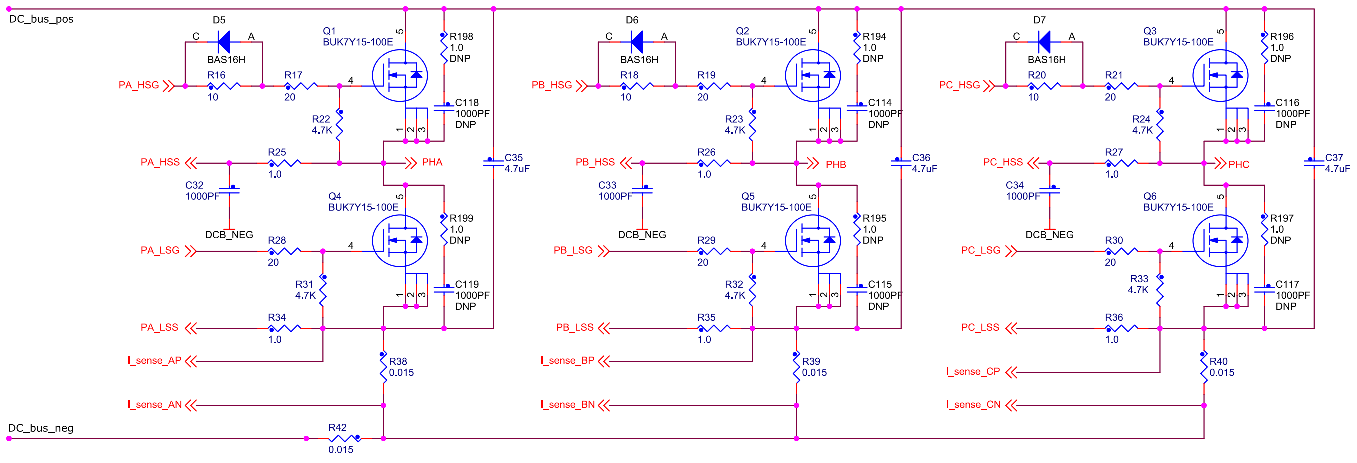


Figure 5. 3-phase low-voltage inverter with shunt resistors for current measurement

An operational amplifier and input signal filtering circuit provide the conditional circuitry and adjust the shunt voltage drops to fit into the ADC input voltage range.

The phase current sampling technique is a critical issue for detection of phase current differences and for acquiring full 3-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 the current flows through the shunt resistor only when the bottom transistor of the respective inverter leg is switched on. Therefore, considering the diagram depicted in Figure 5. on page 6, phase A current is measured using the R38 shunt resistor and can be sampled only when the transistor Q4 is switched on. Correspondingly, the current in phase B can be measured only if the transistor Q5 is switched on, and the current in phase C can be measured only if the transistor Q6 is switched on. To get an actual instant of current sensing, voltage waveform analysis has to be performed. Generated duty cycles phase A, phase B, and phase C of two different PWM periods are depicted in Figure 6. on page 7. These phase voltage waveforms correspond to a center-aligned PWM with sine-wave modulation. As seen in first PWM period, 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. The second PWM period 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, dependent on the hardware design, the current cannot be correctly measured.

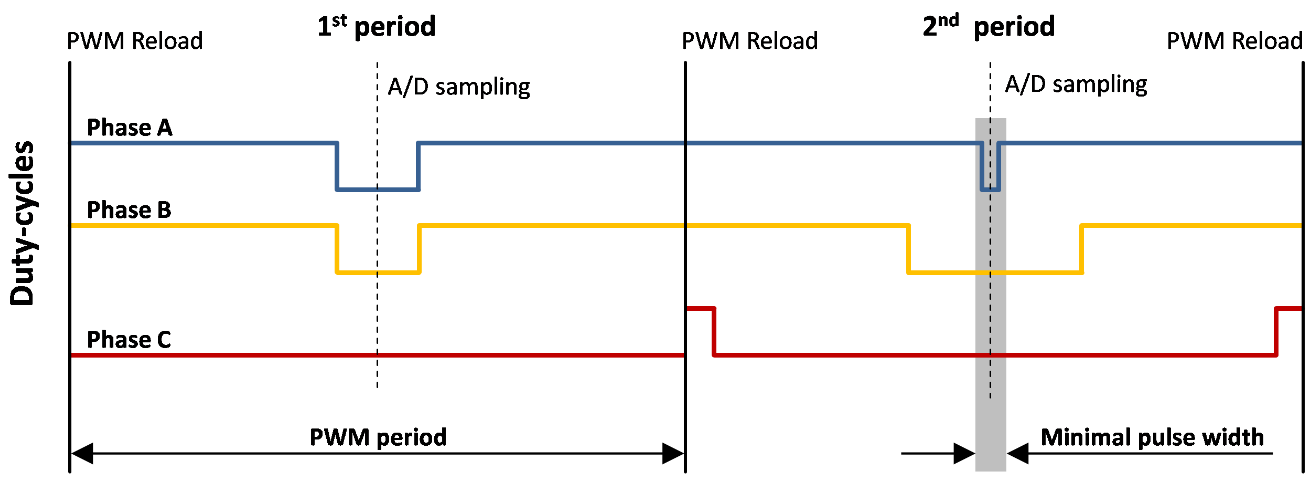


Figure 6. Generated phase duty-cycles in different PWM periods

In 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 Space Vector Modulation (SVM) technique increases, there is an instant when one of the bottom transistors is switched on for a very short time instance. Therefore, only two currents are measured and the third one is calculated from equation:

Equation 7

$$i_A + i_B + i_C = 0$$

Therefore, a minimum on time of the low-side switch is required for 3-phase current reconstruction.

3.4 Resolver signal processing

Different sensor type might require different approach to evaluate the speed and position of the motor. The NXP approach for resolver systems utilizes an Angle Tracking Observer (ATO), Figure 7. on page 8, which is based on the Phase Lock Loop technique. The ATO input is a position error between the position given by the sensor and estimated ATO position. The PI controller in the ATO loop minimizes the input error by adjustment of a control variable, in this case the control variable is equivalent to a motor speed. Integration of the speed leads to the estimated position.

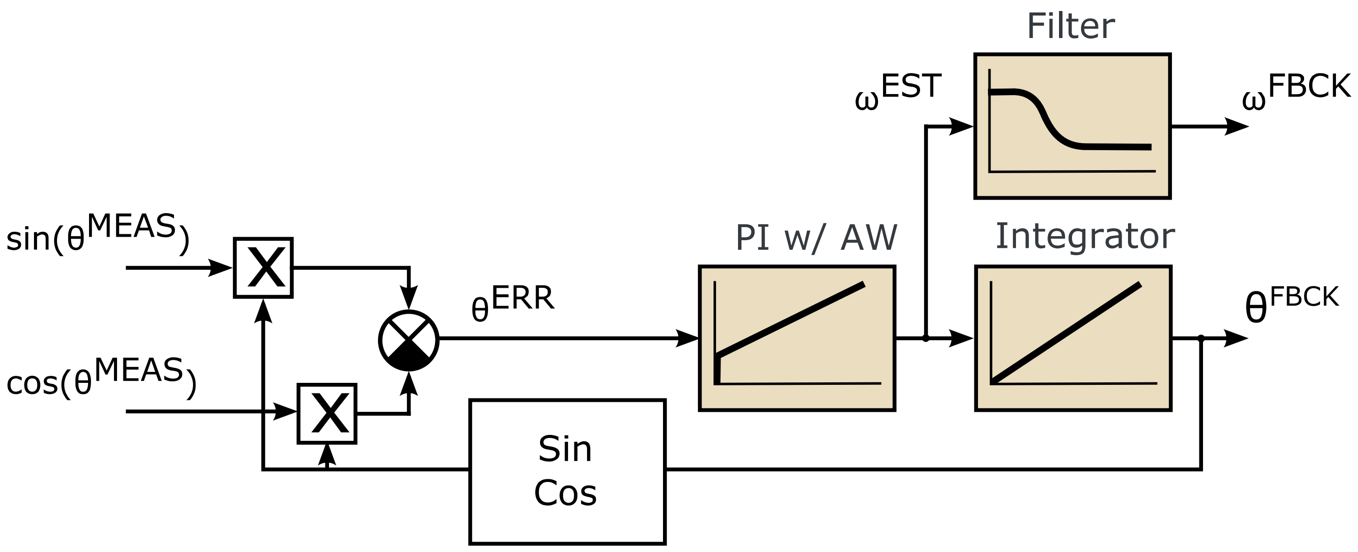


Figure 7. ATO for resolver systems

The ATO for resolver system is characterized by the position error calculation. The observer error corresponds to the following formula:

Equation 8

$$\sin(\theta_r)\cos(\theta_{est}) - \sin(\theta_{est})\cos(\theta_r) = \sin(\theta_r - \theta_{est})$$

The coefficients of ATO PI controller, Integrator and filter can be tuned by MCAT tool. The ATO function is a member of the motor control SW library [8] and is available as AMCLIB_TrackObsrv function.

4 MPC5744P – development kit configuration

MPC5744P development kit is designed to control the PMSM motor equipped with the resolver position sensor. Development kit consists of two hardware boards, the first one is the controller board and the second one is the low voltage 3-phase power stage.

NOTE

The power stage cannot be powered without the controller board when Brake Resistor is populated. The absence of controller board makes the BRAKE_GATE signal high, and a large current flows through the BRAKE_RESISTOR. This creates a considerable burn hazard, as the resistor will dissipate enough heat to harm on contact.

4.1 Controller board

The MPC5744P controller board is intended to control two PMSM motors and therefore it has two PCI-Express 4-edge connectors as the motor control interface. Single PMSM development kit, Figure 1. on page 2, is configured to utilize the PCIe connector J1.

NOTE

The controller board does not contain any of motor control peripheries; all of them are located on the power stage board. The only communication and debug interfaces are part of the controller board.

Following table shows the jumper configuration linked with the PCI-Express motor control interface.

Table 1. MPC5744P controller board jumper configuration

Jumper	Interface	Function	Jumper settings
J2 J3 J4	MCRGM_ABS1 MCRGM_ABS2 MCRGM_FAB	Single Chip Mode configuration	open open 1-2 close
J12	SBC	SBC configured in debug mode	close
J26	SINE-WAVE	Resolver excitation from SW_GEN (*not used*) PCIe connector J1 PCIe connector J200	 1-2 close 2-3 close

For description and explanation of all jumpers on the controller board look for the MPC5744P Motor Controller Board User Manual [3].

4.2 3-phase power stage board

It is a part of NXP motor control development tool series that can be used to supply a wide range of 3-phase AC motors such as PMSM, BLDC or ACIM motors. Low voltage power stage operates up to 50 V in the DC-Bus, being able to deliver and measure current up to 20 A (default setting is 10 A).

The power stage key parts:

- 3-phase MOSFET H-bridge with shunt resistors for current measurement and intelligent pre-driver MC33937 for 3-phase MC applications
- Operational amplifiers and signal filtering to adjust measured voltage signals into the ADC input voltage range
  - — 3-phase current sensing
  - — DC-bus current sensing
  - — DC-bus voltage sensing
  - — Back-EMF voltage sensing
- Motor control peripheries
  - — J8 – Resolver connector with resolver excitation and signal processing circuitry
  - — JP1 – Encoder/Hall interface
- Motor Control interface
  - — J14 – PCI-Express connector
- PWM Braking Interface
  - — J2 – BRAKE_RESISTOR jumper, ensure resistor is removed if not using PWM braking to avoid burn hazard

Table 2 shows the Power Stage board jumper configuration to meet the PMSM FOC application requirements.

Table 2. 3-phase LV power stage board jumper configuration

Jumper	Interface	Function	Setting
J2	BRAKE_RESISTOR	PWM BRAKING. Controlled by BRAKE_GATE signal	Remove resistor if not using PWM braking in application
J5	Resolver Sin-Cos	Resolver S4 output enters operational amplifier (*default*) DC offset compare value (not implemented)	1-2 closed 2-3
J6	Resolver Sin-Cos	Resolver S3 output enters operational amplifier (*default*) DC offset compare value (not implemented)	1-2 closed 2-3
J7	Resolver	Resolver excitation – square signal (*default*) Resolver excitation – SWG source	2-3 closed 1-2
J9	DC-bus Current measurement	By an external operational amplifier By a MC33937	1-2 closed 2-3
J10	Overcurrent threshold reference	+5V DC V_ref	1-2 2-3 closed
J11	Over Current fault	External comparator (*default*) MC33937 output	1-2 closed 2-3
J16	Zero-Cross	Zero-cross detection Zero-cross signal from MC33937 Encoder / Hall sensors - PhA	2-3 1-2
J17	Zero-Cross	Zero-cross detection Zero-cross signal from MC33937 Encoder / Hall sensors - PhB	2-3 1-2
J18	Zero-Cross	Zero-cross detection Zero-cross signal from MC33937 Encoder / Hall sensors	2-3 1-2

NOTE

Even if the example development kit does not contain the PMSM motor with an encoder sensor, the example software offers the possibility to add the encoder software routine for position and speed processing and evaluation.

5 MPC5744P – MCU configuration

The PMSM FOC framework application software and hardware configured for single or dual PMSM application. It is designed to meet the following hardware configuration and technical specifications:

- Speed FOC control with resolver sensor
- Current control loop runs at 10 kHz frequency
- Speed control loop runs at 1 kHz frequency
- Data acquisition sampling period 100 μs
- PWM output switching frequency is 20 kHz

For each motor, the PMSM FOC framework application software requires periodic measurements of a set of process values. These process values are acquired through processor peripheral modules allocated for each motor as follows:

Table 3. Peripheral Table for Motor 1 (J1 PCI-e connector)

Process Variable	**Peripheral**
Phase A Current Phase B Current Phase C Current	ADC0/1 channel 11 ADC0/1 channel 12 ADC0/1 channel 13
DC-bus Voltage	ADC0/1 channel 14
DC-bus Current	ADC0/1 channel 1
Resolver Sine Resolver Cosine	ADC0 channel 0 ADC1 channel 1
Encoder Phase A Encoder Phase B	eTimer0 channel 0 eTimer0 channel 1

Table 4. Peripheral Table for Motor 2 (J200 PCI-e connector)

[JG1] Process Variable	**Peripheral**
Phase A Current Phase B Current Phase C Current	ADC2/3 channel 0 ADC2/3 channel 1 ADC1/3 channel 4
DC-bus Voltage	ADC1/3 channel 5
DC-bus Current	ADC1/3 channel 7
Resolver Sine Resolver Cosine	ADC2 channel 2 ADC3 channel 3
Encoder Phase A Encoder Phase B	eTimer1 channel 0 eTimer1 channel 1

In this application, the FOC framework utilizes a resolver to evaluate the speed and position of the motor. For each motor, a resolver excitation signal must be generated by the controller board to excite the resolver windings and allow for shaft angle encoding. These signals are generated for each motor by the following peripherals:

Table 5. Resolver Excitation Signal Peripheral Table

Motor	Peripheral
Motor 1 (J1 PCI-e connector)	eTimer0 channel 5
Motor 2 (J200 PCI-e connector)	eTimer1 channel 5

The motor driver peripherals used to provide the 3-phase PWM outputs to each motor are the following:

Table 6. 3-Phase PWM output for Motor 2

Process Variable	Peripheral
Phase A	FlexPWM_1 submodule 0 A FlexPWM_1 submodule 0 B
Phase B	FlexPWM_1 submodule 1 A FlexPWM_1 submodule 1 B
Phase C	FlexPWM_1 submodule 2 A FlexPWM_1 submodule 2 B

Table 7. 3-Phase PWM output for Motor 1

Process Variable	Peripheral
Phase A	FlexPWM_0 submodule 0 A FlexPWM_0 submodule 0 B
Phase B	FlexPWM_0 submodule 1 A FlexPWM_0 submodule 1 B
Phase C	FlexPWM_0 submodule 2 A FlexPWM_0 submodule 2 B

The FOC application framework requires that the processor synchronize the sampling of process variables with the resolver excitation signal and the PWM Master Reload Signal. The processor module in charge of this is the Scheduler Unit (SU) of the processor's Cross Triggering Unit module (CTU). A CTU is assigned to each motor as follows:

Table 8. CTU Module Allocation

Motor	CTU Module
Motor 1	CTU_0
Motor 2	CTU_1

5.1 MPC5744P configuration file

The static configuration of the MPC5744P device is stored in an external header file *MPC5744P_appconfig.h*. The configuration file includes defines to configure registers of following MCU modules and peripheries:

- Interrupts; ISR source, name and its priority
- System Integration Unit Lite2 (SIUL2)

- Pulse Width Modulator (FlexPWM)
- Cross-Triggering Unit (CTU)
- Analog-to-Digital Converter (ADC)
- Enhanced Motor Control Timer (eTimer)
- Deserial Serial Peripheral Interface (DSPI)
- Periodic Interrupt Timer (PIT)

5.2 Pulse width modulator module (FlexPWM)

The MPC5744P Clock Generation Module (MC_CGM) is configured to generate a clock signal of 160 MHz on the MOTC_CLK bus. The Pulse Width Modulator module (FlexPWM) is clocked from the MOTC_CLK, therefore it is placed behind the IPS Bus Clock Sync Bridge.

The MPC5744P device contains four PWM channels, each of which is configured to control a single half-bridge power stage. Two modules are included on 257 MAPBGA package, while only one module is present on the 144 LQFP package. Targeted MPC5744P Control Drive board is based on 257 MAPBGA device and therefore each of the FlexPWM modules can be used to control 3-phase PMSM. FlexPWM_0 is routed to motor control PCI-Express interface J1 and FlexPWM_1 is routed to motor control PCI-Express interface J200. Both FlexPWM modules can be synchronized using the Master Reload Signal (MRS), as shown in Figure 12. on page 23.

For 3-phase motor control application submodules #0, #1 and #2 are configured.

5.2.1 FlexPWM general settings

Output Enable Register (FlexPWM0_OUTEN)

PWMA / PWMB Output Enable for submodules #0, #1 and #2

- `#define FlexPWM0_OUTEN 0x0770`
- `#define FLexPWM1_OUTEN 0x0770`

Mask Register (FlexPWM0_MASK)

PWMA / PWMB Output Mask Enable for submodules #0, #1 and #2

- `#define FlexPWM0_MASK 0x0770`
- `#define FlexPWM1_MASK 0x0770`

Master Control Register (FlexPWM0_MCTRL)

PWM Generator Clock Enabled for submodules #0, #1 and #2,

PWM23 selected to generate complementary PWM pair

- `#define FlexPWM0_MCTRL 0x0700`
- `#define FlexPWM1_MCTRL 0x0700`

Fault Control Register (FlexPWMn_FCTRL)

Two FlexPWM0 FAULT pins are used to monitor the over-current and over-voltage. These faults are read and evaluated in the software periodic routine thus there is no need the CPU to generate the interrupt by the FAULTx pins.

- Level indicating Fault: logic 0 for submodules #0 and #1
- Fault clearing: Manual
- Fault mode: Safe mode
- Fault interrupt request: Disabled

- `#define FlexPWM0_FCTRL 0xC0F0`
- `#define FlexPWM1_FCTRL 0xC0F0`

The defines to configure the general settings of FlexPWM module are stored in the *MPC5744P_appconfig.h* file.

5.2.2 FlexPWM SUBmodule settings

For Motor 1, the FlexPWM_0 submodule #0 is configured to run as a master and to generate MRS and counter synchronization signal (master sync) for other submodules in FlexPWM module 0. Similarly, the FlexPWM_1 submodule #0 is configured to run as a master and to generate MRS and counter synchronization signal (master sync) for other submodules in FlexPWM_1. The MRS signal for both FlexPWM modules is generated every second opportunity of submodule #0, VAL1 compare, that is, full cycle reload. All double buffered registers, including compare registers VAL2, VAL3, VAL4, and VAL5 are updated on the occurrence of MRS, therefore new PWM duty cycles are updated every two PWM periods.

FlexPWM modulo counting, for generation of center-aligned PWM signals, is achieved by setting VAL0 register to zero and INIT register to negative value of VAL1. When PWM clock of 160MHz, required PWM output 20 kHz and PWM reload period 100μs, then INIT, VAL0, and VAL1 registers of submodules 0, 1, and 2 are set as follows:

- INIT = -160000000/20000/2 = - 4000 DEC = 0xF060
- VAL0 = 0 DEC
- VAL1 = - INIT - 1 = 3999 DEC = 0x F9F

Reload frequency of FlexPWM_0 and FlexPWM_1 submodules 0, 1, and 2 is set to “Every two opportunities“ and “Full-cycle reload” is enabled. Because submodule #0 is a master that generates MRS signal, reload of double buffered registers of submodule #0 is done on Local Reload. Submodules 1 and 2 are slaves, therefore reload of their double buffered registers is done on Master Reload, broadcast from submodule #0. PWM clock frequency of 160 MHz is achieved by setting the prescaler to zero.

- `#define FlexPWM0_SUBn_CTRL1 0x1400 (where n=0,1,2)`
- `#define FlexPWM1_SUBn_CTRL1 0x1400 (where n=0,1,2)`

Similarly, submodules #0 counter is initialized on Local Sync event, while submodules 1 and 2 on Master Sync event again are broadcast from submodule #0. Because some registers are double buffered on occurrence of FORCE OUT signal, all submodules of FlexPWM_0 and FlexPWM_1 have as force source selected Local Reload event. All PWM channels are used to drive a 3-phase DC/AC inverter, therefore each PWM pair is driven in complementary mode, with dead-time automatically added on each rising edge of the respective PWM signal.

- `#define FlexPWM0_SUB0_CTRL2 0xC0A0`
- `#define FlexPWM0_SUBn_CTRL2 0xC2AE (where n=1,2)`
- `#define FlexPWM1_SUB0_CTRL2 0xC0A0`
- `#define FlexPWM1_SUBn_CTRL2 0xC2AE (where n=1,2)`

MC33937 FET pre-driver inverts the polarity of PWM signals for top transistors (active low logic) so PWM A output polarity in all submodules is set as Inverted. Therefore, the output of PWM A of each submodule is set to logic one during the fault state as well.

- `#define FlexPWM0_SUBn_OCTRL 0x0410 (where n=0,1,2)`
- `#define FlexPWM1_SUBn_OCTRL 0x0410 (where n=0,1,2)`

FlexPWM modules include 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 or zero (depends on the Fault Level settings in FCTRL) on any of the FAULTx pins. To enable mapping of all fault pins, fault disable mapping registers (DISMAP) of all submodules must be enabled.

5.3 Cross-Triggering Unit (CTU)

The MPC5744P CTU is clocked from the MOTC_CLK clock signal of 160MHz on the bus. Two modules of CTU are included on 257 MAPBGA package, while the only one module is present on the 144 LQFP package. Targeted MPC5744P Control Drive board is based on 257 MAPBGA device and therefore each of the CTU modules can be used to schedule the state variables acquisition from both MC PCI-Express interfaces with respect to PWM cycle.

For dual PMSM control, two CTU modules are used on the 257 MAPBGA package:

- CTU_0
- CTU_1.

The MRS signal generated from the FlexPWM_0 0 and FlexPWM_1 module is internally routed to the CTU_0 and CTU_1 module respectively. The MRS signal is selected using the input selection register Trigger Generator Subunit Input Selection Register (TGSISR) as the source of master reload signal for CTU.

- `#define CTU0_TGSISR 0x00000001`
- `#define CTU1_TGSISR 0x00000001`

This signal is used to reload trigger compare registers (TnCR) and to reload the TGS counter with the value stored in the TGS counter reload register. Because the MRS signal is generated every two PWM periods, the CTU counter can count up to value of 16000DEC considering the initial value is set to zero.

- `#define CTU0_TGSCCR 0x3E80`
- `#define CTU1_TGSCCR 0x3E80`

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. If Toggle Mode for external trigger enabled, then:

- `#define CTU0_TGSCR 0x0100`
- `#define CTU1_TGSCR 0x0100`

Following TnCR trigger compare registers are used in the application for trigger events:

- `#define CTU0_T0CR 0x00A0`
- `#define CTU0_T1CR 0x1D4C`
- `#define CTU0_T2CR 0x3C8C`
- `#define CTU1_T0CR 0x00A0`
- `#define CTU1_T1CR 0x1D4C`
- `#define CTU1_T2CR 0x3C8C`

Until the TGS is responsible to generate triggers, the CTU Scheduler subunit (SU) is responsible for generation of ADC commands or triggers to on chip logic such as timers.

Any of trigger events from TGS can be associated with each of the SU outputs. This is implemented by the Trigger Handler block. The group of first four trigger outputs is configured as:

- `#define CTU0_THCR1 0x00424261`
- `#define CTU1_THCR1 0x00505061`

NOTE

Since eTimer_0 is used for Motor 1 and eTimer_1 is used for Motor 2, it is important to specify the correct triggers in the Trigger Handler block for each of the CTU modules.

Trigger event outputs are generated by the CTU SU according to the occurred trigger event. The list of trigger events outputs for a given FOC application is as follows:

- For Motor 1:
  - — CTU0_T0CR – generates the ADC command event output synchronized with respect to PWM period. It is used for motor phase currents, DC-bus voltage and resolver outputs measurement.
  - — CTU0_T1CR – is associated with eTimer_0 event output to generate the rising edge of the square signal for resolver excitation.
  - — CTU0_T2CR – is associated with eTimer_0 event output to generate the falling edge of the square signal for resolver excitation.
- For Motor 2:
  - — CTU1_T0CR – generates the ADC command event output synchronized with respect to PWM period. It is used for motor phase currents, DC-bus voltage and resolver outputs measurement.
  - — CTU1_T1CR – is associated with eTimer_1 event output to generate the rising edge of the square signal for resolver excitation.
  - — CTU1_T2CR – is associated with eTimer_1 event output to generate the falling edge of the square signal for resolver excitation.

A 10 kHz square signal is filtered to a sine-wave form and processed by conditional resolver circuit to feed the resolver input excitation winding. Triggers T1 and T2 are therefore set to toggle the eTimer_0 output at 10 kHz frequency respecting the formula: MOTC_CLK/(2(T2CR-T1CR)) = 10 kHz. The process to align the T1 and T2 with respect to PWM period is shown in CTU triggers, ADC conversion and interrupt timing on page 19.

The SU uses a Commands List, 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 the 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 a small number (A0 HEX), it generates the ADC start of conversion request almost 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.

For Motor 1 - At each T0CR trigger event, four 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 that the DC Bus Voltage be sampled, as well as the third phase current. The third command specifies the resolver sine and cosine feedback signals to be sampled, and the fourth command specifies that the DC Bus current be sampled.

For Motor 2 - At each T0CR trigger event, five 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 ADC1/ADC3 and ADC2/3). The second command specifies that the DC Bus Voltage be sampled. The third command specifies that the third phase current be sampled. The fourth command specifies that the resolver sine and cosine feedback signals be sampled, and the fifth command specifies that the DC Bus Current be sampled.

The index pointer to the ADC command list T0_INDEX is updated according to the sector in which the actual output voltage vector resides, 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 is calculated. Which phases are measured and which are calculated changes according to the voltage vector angle, that is, the phases with the largest PWM on-pulse on the bottom transistors are selected to get the best current information.

ADC Command list:

- For Motor 1

Table 9. ADC command list for Motor 1

ADC cmd.	First cmd.	Interrupt cmd.	Conversion Type	ADC A CH	ADC B CH	ADC A Signal	ADC B Signal	CLRx	#define
0	*x*		*dual*	*11*	*12*	*I_phA*	*I_phB*	*0*	*0x618B*
1			*dual*	*13*	*14*	*I_phC*	*Udc_bus*	*1*	*0x21CD*
2			*dual*	*0*	*0*	*R_sin*	*R_cos*	*2*	*0x2000*
3		*x*	*dual*	*15*	*1*	*-*	*Idc_bus*	*3*	*0xA02F*

- For Motor 2

Table 10. ADC command list for Motor 2

ADC cmd.	First cmd.	Interrupt cmd.	Conversion Type	ADC A CH	ADC B CH	ADC A Signal	ADC B Signal	CLRx	#define
0	*x*		*dual*	*0*	*1*	*I_phA*	*I_phB*	*0*	*0x6020*
1			*dual*	*6*	*5*	*-*	*Udc_bus*	*1*	*0x20A6*
2			*dual*	*9*	*4*	*-*	*I_phC*	*2*	*0x2089*
3			*dual*	*2*	*3*	*R_sin*	*R_cos*	*3*	*0x2062*
4		*x*	*dual*	*8*	*7*	*-*	*Idc_bus*	*4*	*0xA0E8*

NOTE

Motor 2 requires an extra CTU-ADC command because CTU_1 can only interface with ADC2 and ADC3, but some process variables are tied to shared channels between ADC1 and ADC3. Thus, instead of sampling ADC1 and ADC3 simultaneously, CTU_1 must sample ADC3 twice. This is why Udc_bus and I_phC are on separate commands for Motor 2 unlike Motor 1 where they are sampled simultaneously.

5.4 Enhanced Motor Control Timer (eTimer)

The eTimer modules are used for resolver excitation square signal generation and can be used for decoding two square-wave signals from the encoder sensor.

5.4.1 Resolver excitation

As discussed in section Cross-Triggering Unit (CTU) on page 15, the sequence of the trigger events T1 and T2 generated by TGS is associated with eTimer0 and eTimer1 output events generated by the Scheduler Subunit (SU).

The eTimer0 and eTimer1 modules are then used as generators of the square wave signal for the resolver excitation. By enabling the output bit (OEN) the OFLAG output signal is driven on the external pin. For Motor 1, the controller board design utilizes channel 5 of the eTimer0 module. For Motor 2, channel 5 of eTimer1 is used. eTimer_0 and eTimer_1 (using channel 5) are then configured as follows:

- Counting mode - Edge of secondary source triggers primary count till compare
- Count direction - Count Up

- Primary source - IP Bus clock divide by 1 prescaler
- Count stop mode - Count Repeatedly
- Count length - Count Until Compare then Reinitialize
- Secondary source - AUX #0, which is an output signal from CTU ETIMER 0_TRG event output
  - `#define eTimer0_CTRL1_CH5 0xD848`
  - `#define eTimer1_CTRL1_CH5 0xD848`
- Polarity of secondary source - True
- 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
  - `#define eTimer0_CTRL2_CH5 0x8403`
  - `#define eTimer1_CTRL2_CH5 0x8403`

5.4.2 Encoder decoding

This section describes how to configure the eTimer for decoding the square signals from encoder sensor. Even if the PMSM motor in the MPC5744P Kit is not equipped with this type of sensor, the encoder connector is available on the power stage board and therefore the setting and decoding routines are available in the example software.

The goal is to decode two ninety-degree shifted square-wave signals. The eTimer counter value then represents the information about the actual rotor position. The quadrature count mode increases the precision four time by counting up or down all rising and falling edges of the encoder square signals.

The hardware design of the controller board utilizes the eTimer_0 and eTimer_1 modules with encoder Phase A and Phase B connected to channel 0 and 1 respectively.

The eTimer_0 module and eTimer_1 module, channel 0 is then configured as follows:

- Counting mode - Quadrature count mode, uses primary and secondary sources
- Count direction - Count Up
- Primary source - Counter #0 input
- Secondary source - Counter #1 input
- Count stop mode - Count Repeatedly
- Count length - Count Until Compare then Reinitialize
  - `#define eTimer0_CTRL1_CH0 0x8041`
  - `#define eTimer1_CTRL1_CH0 0x8041`
- 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
  - `#define eTimer0_CCCTRL_CH0 0xDE00`
  - `#define eTimer1_CCCTRL_CH0 0xDE00`

The compare registers of eTimer0 channel #0 are set according to the number of encoder pulses per one mechanical revolution. As an example, the encoder sensor with 1024 pulses is used. In quadrature mode, it means the encoder capability of position recognition with a precision that is four times higher than the number of pulses in the application, the maximum numbers of edges is 4096.

The compare registers and comparator load registers are calculated as follows:

- COMP1 = 4096/2 - 1 = 2047 DEC (0x07FF HEX)
- COMP2 = -4096/2 = -2048 DEC (0xF800 HEX)
    - `#define eTimer0_COMP1_CH0 0x07FF`
    - `#define eTimer0_COMP2_CH0 0xF800`
    - `#define eTimer0_CMPLD1_CH0 0x07FF`
    - `#define eTimer0_CMPLD2_CH0 0XF800`
    - `#define eTimer1_COMP1_CH0 0x07FF`
    - `#define eTimer1_COMP2_CH0 0xF800`
    - `#define eTimer1_CMPLD1_CH0 0x07FF`
    - `#define eTimer1_CMPLD2_CH0 0XF800`

5.5 CTU triggers, ADC conversion and interrupt timing

Configuration of FlexPWM0/FlexPWM1, CTU0/CTU1, and eTimer0/eTimer1 peripheral modules, as described in the above sections, results in a sequence of triggers/events that are explained in the following timing diagrams.

The timing of the CTU triggers linked with the resolver signal generation and processing is shown in the following figure.

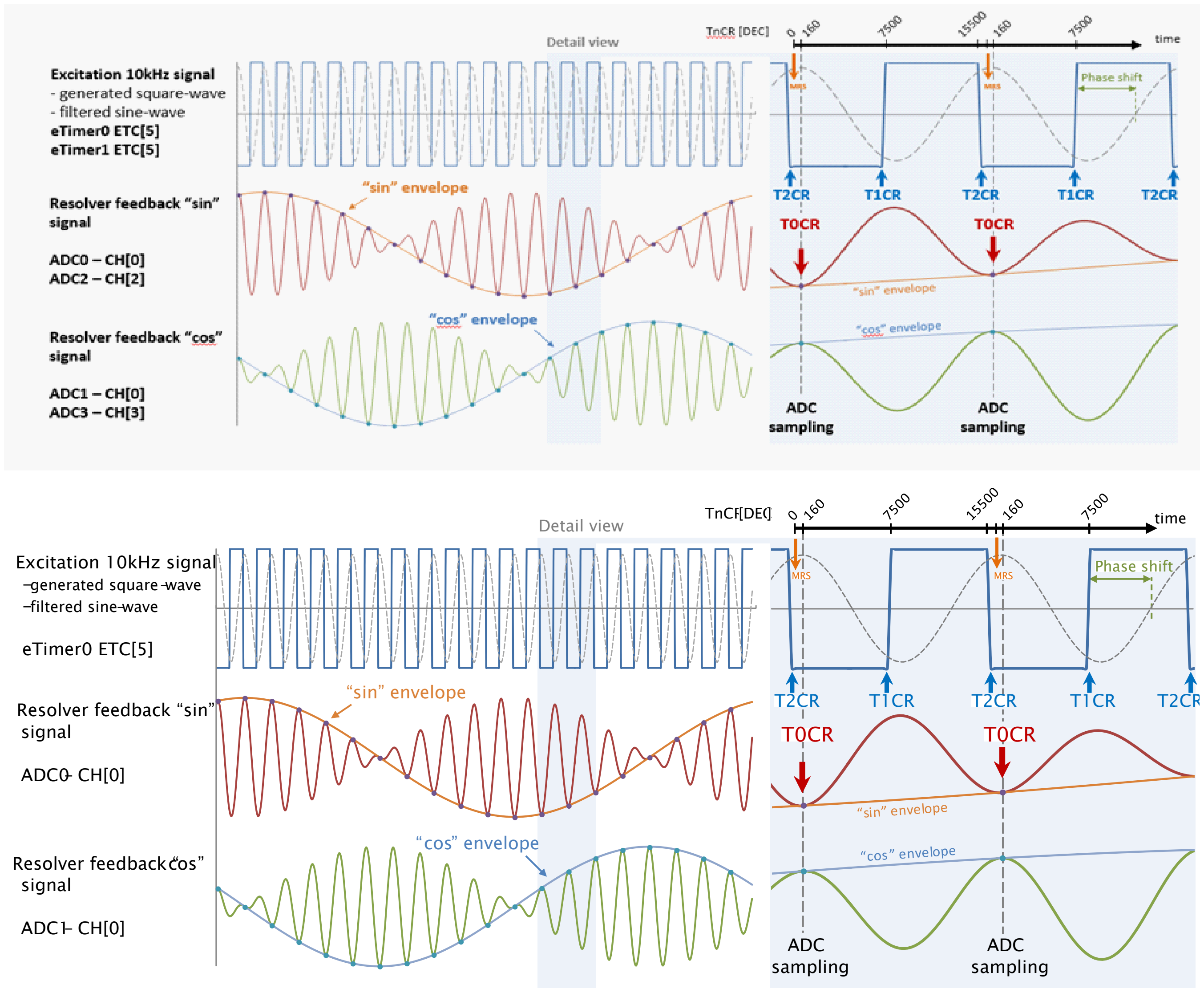


Figure 9. An envelope extractor of resolver feedback signals

As described earlier, triggers T1 and T2 are used to generate the square-wave excitation signal. This signal is then processed by a 3rd order Sallen-Key low-pass filter. The cut-off frequency of the filter is 10 kHz (precision may vary according to the tolerance of the RC components in the filter). Proposed solution results to a sine-wave signal that is directly used for feeding the resolver excitation winding. However, the low-pass filter makes the sine-wave signal phase-shifted to the origin square-wave signal. This phenomenon has to be taken into the consideration when CTU trigger compare registers T1CR and T2CR are set.

Both triggers T1 and T2 are set with respect to the T0 trigger which handles the sequence of the phase current, DC-bus voltage, and resolver feedback measurement. The goal is to have the positive peak of phase-shifted sine signal aligned with T0 instance and therefore the T1 and T2 are manually set according to the detailed view of Figure 9. on page 20.

The application state machine functions are called from an interrupt service routine (ISR), which is associated with CTU-ADC command interrupt request. As shown in the ADC command list, the ADC command interrupt is linked with CTU trigger T0CR, that is, when measurement of the phase currents, DC bus voltage and resolver feedbacks is finished.

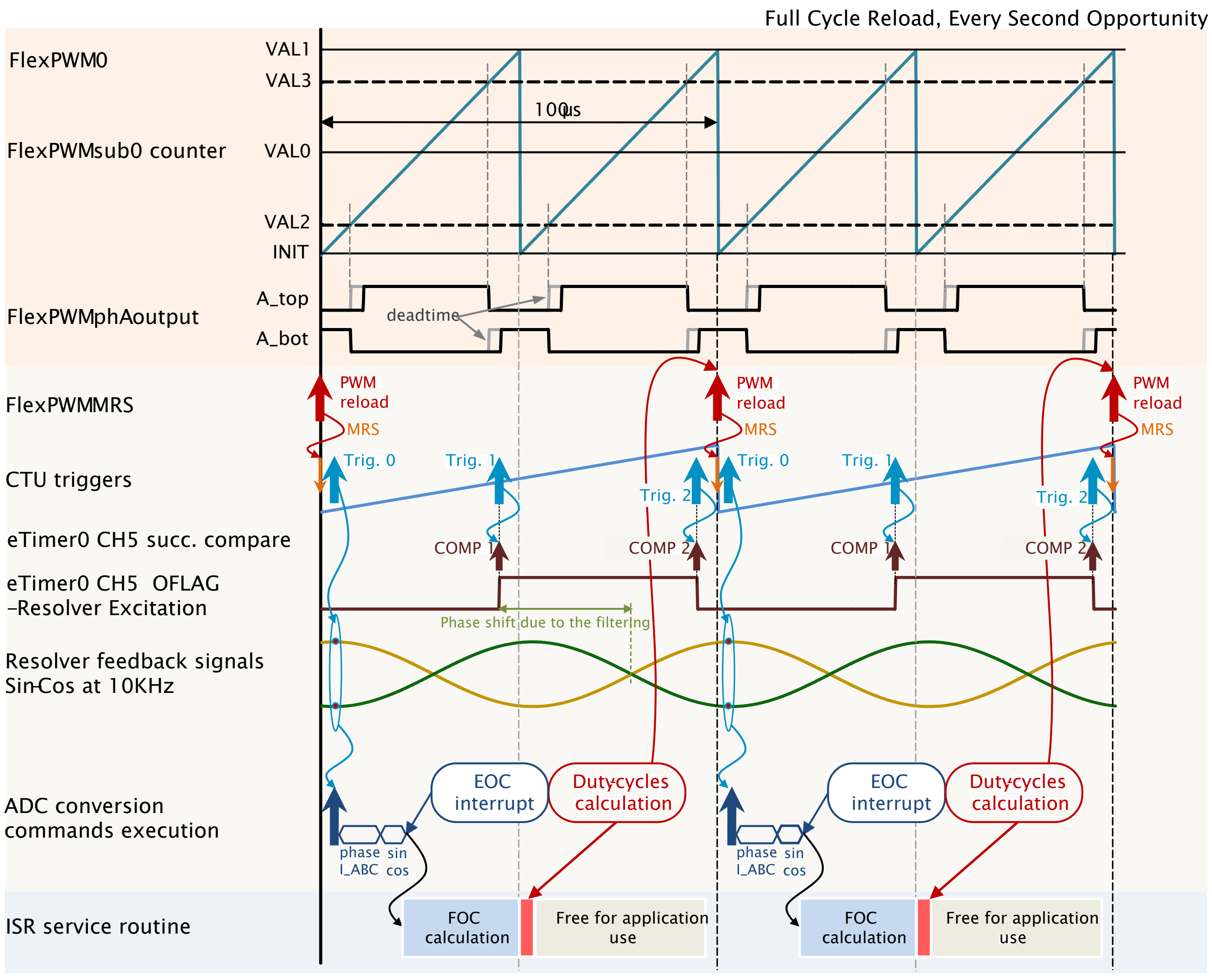


Figure 10. FlexPWM-CTU-ADC conversion timing for single motor control

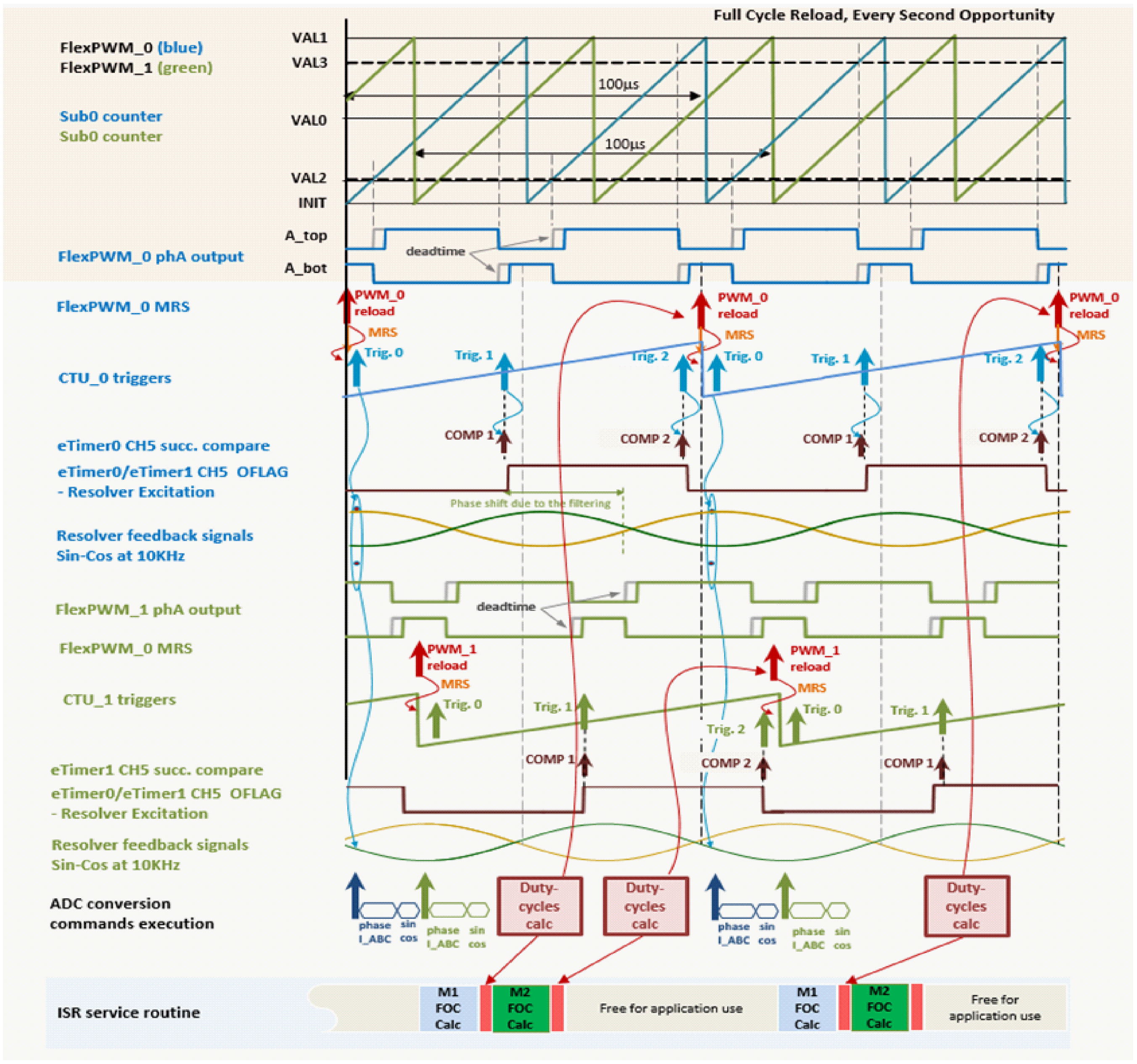


Figure 11. FlexPWM-CTU-ADC conversion timing for dual motor control

To ensure correct calculation of the FOC algorithm, the measurement of the resolver outputs and phase current must be executed at the same time. Because there are only two independent sample and hold circuitries on MPC5744P, parallel sampling of all signals is impossible. Two phase currents can be read at the same time as well as two resolver signals can be read at the same time. Two measurements of these four signals must, however, be sequenced one after another. Sequencing of FOC quantities measurement is handled by ADC Command List.

NOTE

The CTU external trigger in toggle mode can be used to debug the application timing processes.

5.6 On-chip motor control peripherals interconnection

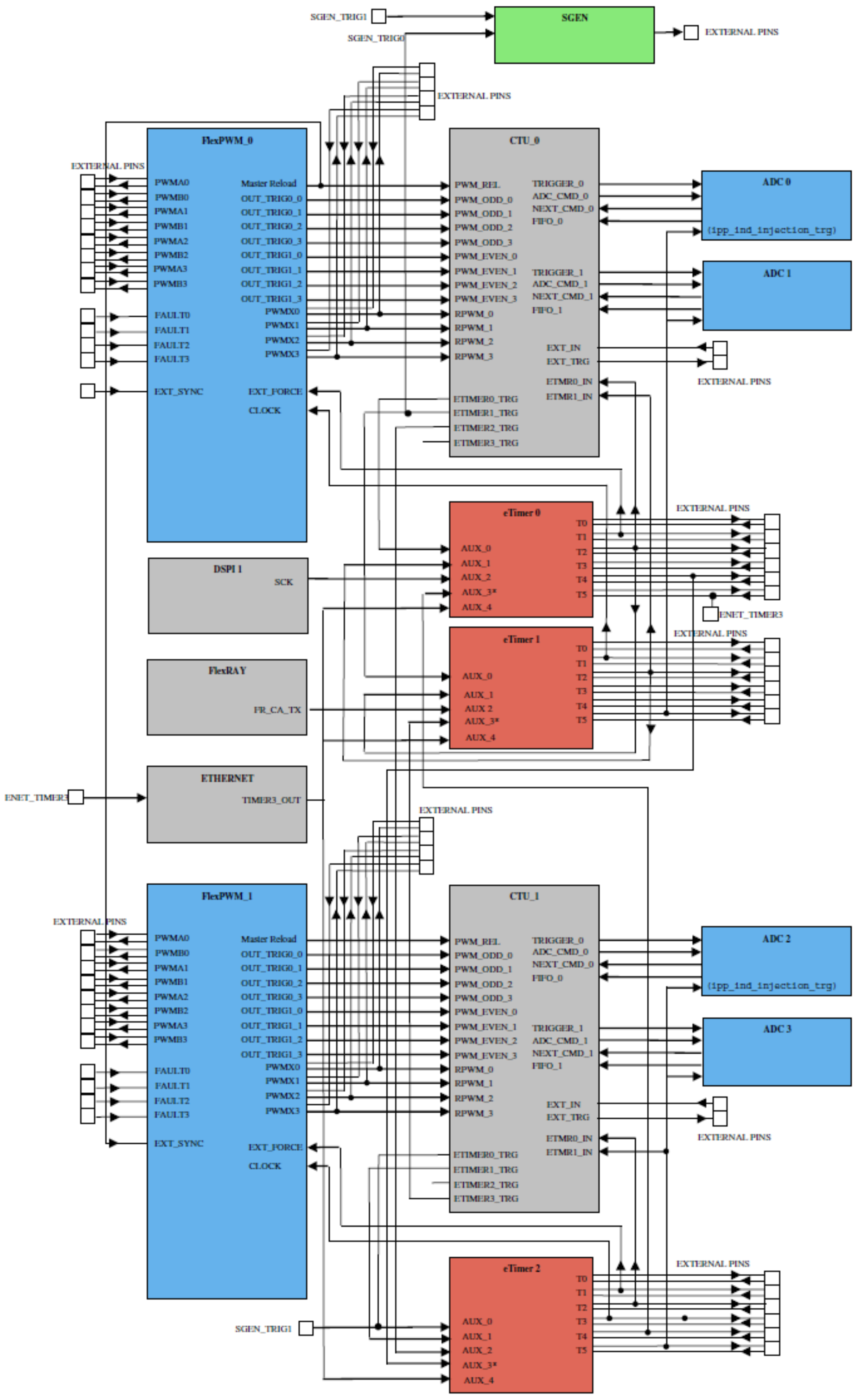


Figure 12. MPC5744P motor control peripherals connection

6 Software design

6.1 Introduction

This section describes the embedded software framework of the PMSM FOC application and its implementation in MPC5744P. The aim of this section is to develop a clear understanding of the designed software focusing on key parts such as software framework based on the application state machine, loop control and interrupt routines.

6.2 Application software design

The idea of proposed concept is to split the software into two main layers. The first one is linked with the device and hardware configuration, and is called as a hardware dependent layer. While the second one is hardware independent layer and contains software modules for signal processing.

The application defines a new type named pmsmFOC_t available in PMSM_struct.h file.

```
typedefstruct  {
        AppFaultStatus  faultID;      // Application actual fault
        AppFaultStatus  faultIDp;     // Application fault flagst
        U32             svmSector;    // Space Vector Modulation sector
        SWLIBS_2Syst    iDQFbck;      // dq axis feedback currents
        SWLIBS_2Syst    iDQReq;       // dq axis required currents, given by speed PI
        SWLIBS_2Syst    iDQReqZC;     // dq axis currents after zero cancellation
        SWLIBS_2Syst    iDQErr;       // Error between reference and feedback signal
        SWLIBS_2Syst    uDQReq;       // dq axis required voltages, given by current PIs
        SWLIBS_2Syst    thTransform;  // transformation angle - for Park transformation
        SWLIBS_2Syst    iAlBeFbck;    // alpha/beta axis feedback currents
        SWLIBS_2Syst    uAlBeReq;     // alpha/beta required voltages
        SWLIBS_2Syst    uAlBeReqDCB;  // alpha/beta voltages, DC Bus ripple elimination

        GMCLIB_ELIMDCBUSRIP_T       elimDcbRip;  // DC-bus voltage ripple elimination
        alignment_t                 align;       // Alignment procedure
        GDFLIB_FILTER_IIR1_T        dAxisZC;     // d-axis current zero cancellation
        GDFLIB_FILTER_IIR1_T        qAxisZC;     // q-axis current zero cancellation
        GFLIB_CONTROLLER_PIAW_P_T   dAxisPIp;    // d-axis current PI controller
        GFLIB_CONTROLLER_PIAW_P_T   qAxisPIp;    // q-axis current PI controller
        GFLIB_CONTROLLER_PIAW_P_T   speedPIp;    // Speed Loop PI controller

        scalarControl_t       scalarControl;     // Scalar Control states variables
        driveStates_t         cntrState;         // Control states variables
        controlLoop_t         controlLoop;       // Position/Speed variables
        resolver_data_t       Resolver;          // SENSOR - resolver s/w data type
        ATO_observer_t        Resolver_SW;       // SENSOR - Tracking Observer s/w data type
        flexPWM_sw_t          pwm;               // ACTUATOR - FlexPWM s/w data type
        MC33937_T             MC33937;           // ACTUATOR - 3-ph. pre-driver s/w data type
        mpc5744P_DevKit_t     MPC5744P_HW;       // HW initialization of Development Kit
modules

        ph_current_meas_data_t      iAbcFbck;  // SENSOR - three phases current feedback
        dcb_voltage_meas_data_t     uDcbFbck;  // SENSOR - raw/filtered value of Udc
        volatile tU8      btSpeedUp;           //button on power stage to speed up motor
        volatile tU8      btSpeedDown;        //button on power stage to slow down motor
        tU8motorID;                          //unique motor identifier
} pmsmFOC_t;
```

The proposed structure includes all variables related to cascade structure for PMSM FOC, state variables related to the application and development kit, and data for peripherals configuration such as base address or channel offset.

Table 11. MotorID and FOC Structure Table

motorID	pmsmFOC_t
0 – (Motor 1)	FOC_one
1 – (Motor 2)	FOC_two

6.3 Application data flow overview

The application software is interrupt driven running in real time. There are two periodic interrupt service routines (FOC_one_Fast_ISR() and FOC_two_Fast_ISR()) associated with CTU-ADC command interrupt requests. These ISRs execute all motor control tasks including both fast current and slow speed loop control. All tasks are performed in an order described by the application state machine shown on Figure 15. on page 27 and application flowcharts shown on Figure 14. on page 26 and Figure 13. on page 25.

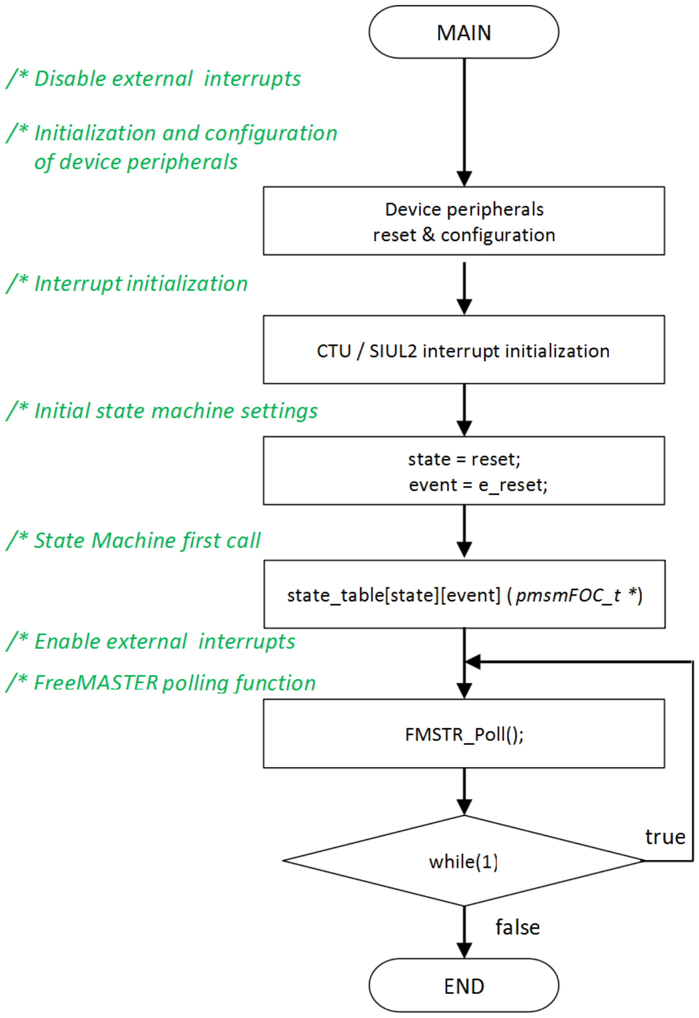

MAIN
/* Disable external interrupts
/* Initialization and configuration of device peripherals
Device peripherals reset & configuration
/* Interrupt initialization
CTU / SIUL2 interrupt initialization
/* Initial state machine settings
state = reset; event = e_reset;
/* State Machine first call
state_table[state][event] (pmsmFOC_t *)
/* Enable external interrupts
/* FreeMASTER polling function
FMSTR_Poll();
while(1)
true
false
END

Figure 13. Flow chart diagram of a main function

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 ISR. Therefore, to call state machine functions, the periphery causing this periodic interrupt must be properly configured and the interrupt enabled. As described later, all peripherals are initially configured and all interrupts are enabled before calling the state machine, see Figure 15. on page 27.

As soon as interrupts are enabled and all peripheries are correctly configured as described in MPC5744P – MCU configuration on page 10, the state machine functions are called from the CTU-ADC interrupt service routine. The background loop handles noncritical timing tasks, such as the FreeMASTER communication polling.

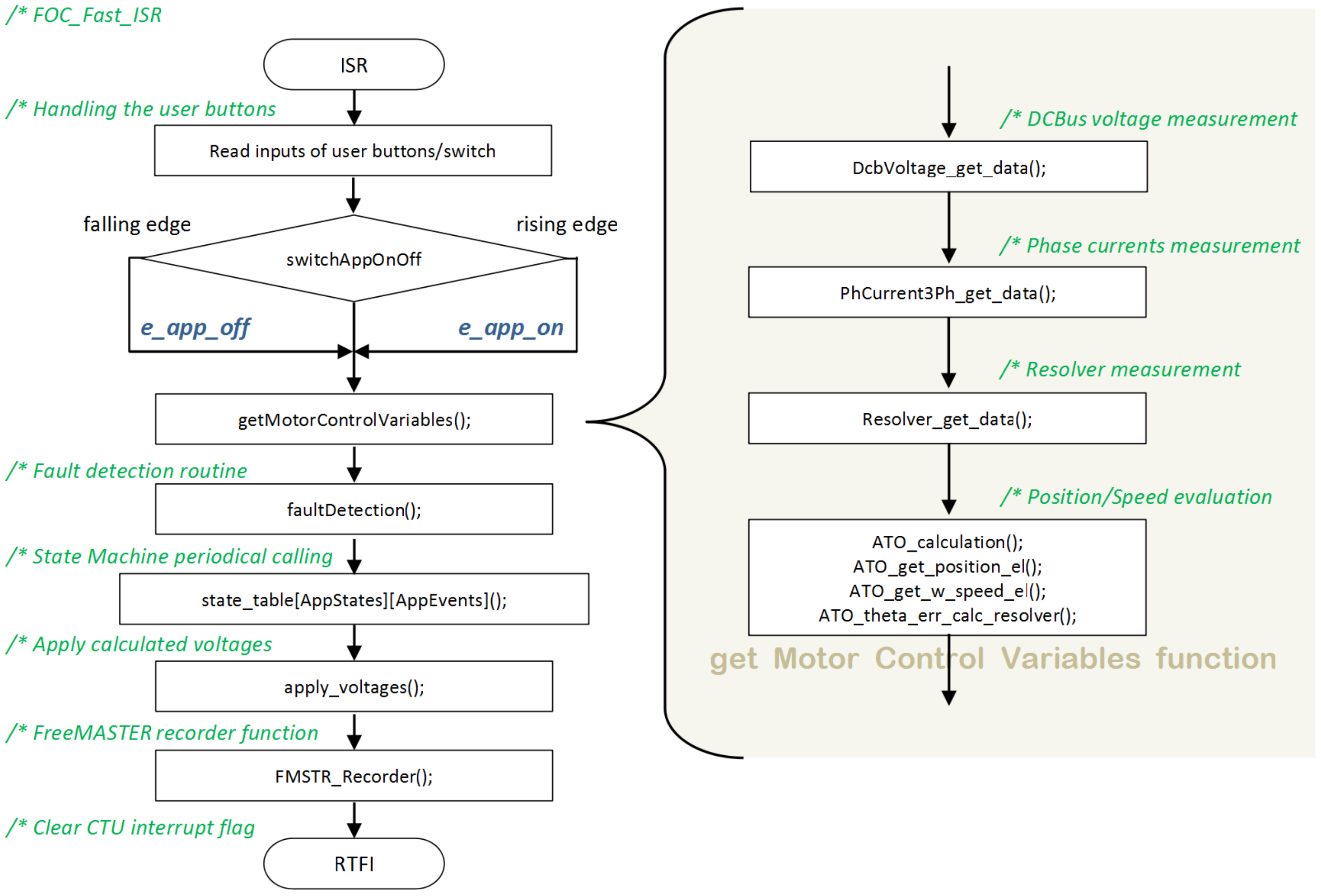

/* FOC_Fast_ISR
ISR
/* Handling the user buttons
Read inputs of user buttons/switch
falling edge
rising edge
switchAppOnOff
e_app_off
e_app_on
getMotorControlVariables();
/* Fault detection routine
faultDetection();
/* State Machine periodical calling
state_table[AppStates][AppEvents]();
/* Apply calculated voltages
apply_voltages();
/* FreeMASTER recorder function
FMSTR_Recorder();
/* Clear CTU interrupt flag
RTFI
/* DCBus voltage measurement
DcbVoltage_get_data();
/* Phase currents measurement
PhCurrent3Ph_get_data();
/* Resolver measurement
Resolver_get_data();
/* Position/Speed evaluation
ATO_calculation();
ATO_get_position_el();
ATO_get_w_speed_el();
ATO_theta_err_calc_resolver();
get Motor Control Variables function

Figure 14. Flow chart diagram of periodic interrupt service routine

6.4 State machine

The application state machine is implemented using a two-dimensional array of pointers to functions called state_table_one[][]() and state_table_two[][]() for Motor 1 and Motor 2 respectively. 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_one[][]() or state_table_two[][]() is called.

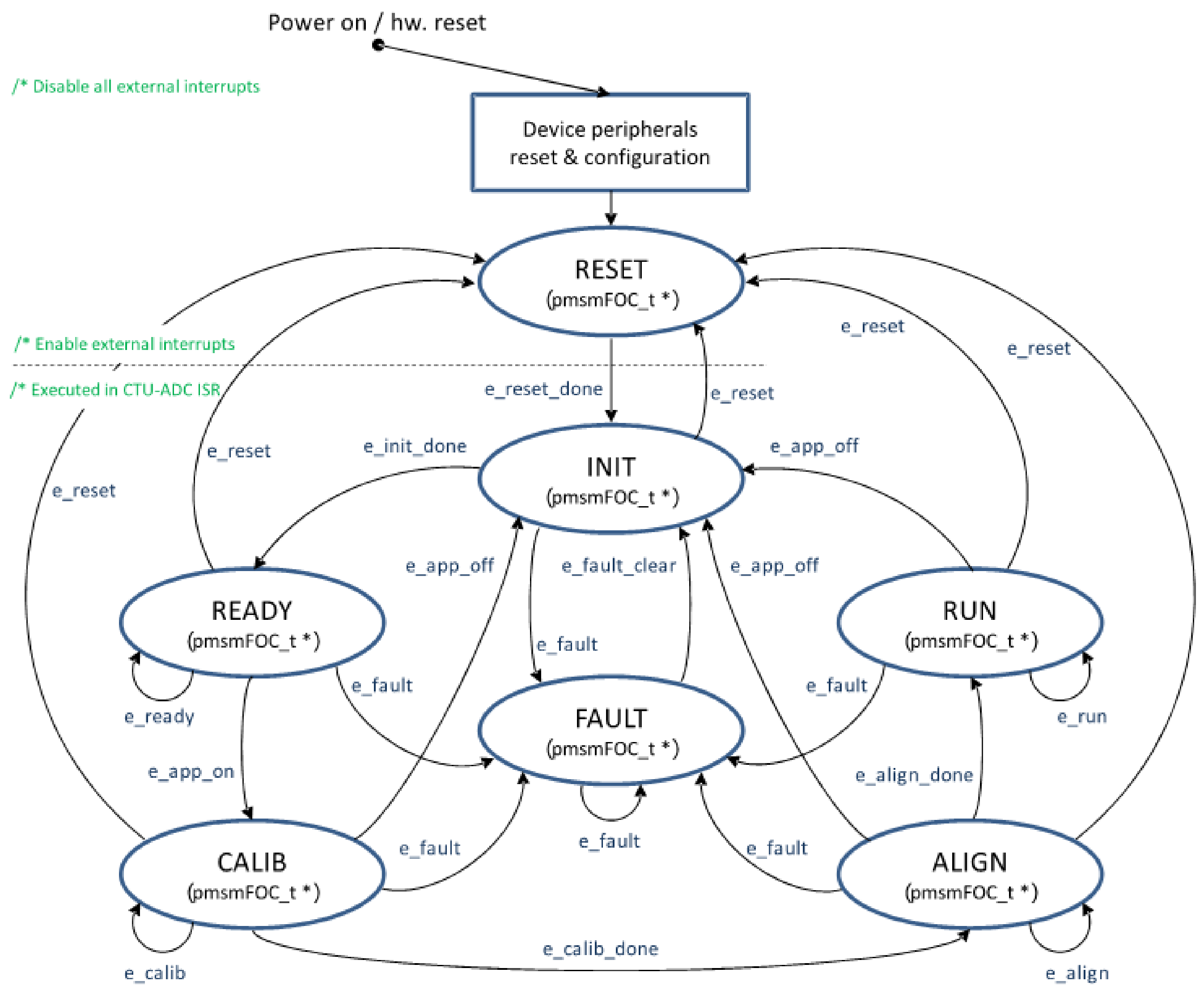


Figure 15. Application state machine

The application state machine considers seven states represented by a variable state defined as AppStates :

```
typedef enum {
        reset               = 0,
        init                = 1,
        fault               = 2,
        ready               = 3,
        calib               = 4,
        align               = 5,
        run                 = 6
    }AppStates;             /* Application state user type*/
```

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

```
typedef enum {
    e_reset          = 0,
    e_reset_done     = 1,
    e_fault          = 2,
    e_fault_clear    = 3,
    e_init_done      = 4,
    e_ready          = 5,
    e_app_on         = 6,
    e_calib          = 7,
    e_calib_done     = 8,
    e_align          = 9,
    e_align_done     = 10,
    e_run            = 11,
    e_app_off        = 12
    }AppEvents;          /* Application event user type */
```

6.4.1 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 MPC5744P – MCU configuration on page 10). Before configuring peripheral modules, all interrupts are disabled. Interrupts are enabled after the RESET state pass. 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 is from within the CTU-ADC interrupt service routine.

According to the data flow diagram of the CTU-ADC interrupt service routine, Figure 14. on page 26, the routine for three phase current measurement `PhCurrent3Ph_get_data()` 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 is 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 `e_ reset_done`
- interrupts enable

6.4.2 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.

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

6.4.3 State – FAULT

The application goes immediately to this state when a fault is detected. The system allows all state machine states to pass into the FAULT state by setting `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 applicationevent
```

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.

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

Table 12. Application fault status user type

x	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	FLTx	FLTx	FLTx	RESERVED				FLTx	FLTx	FLTx	FLTx	RESERVED				
x	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RESERVED					FLTx	FLTx	FLTx	FLTx	FLTx	FLTx	FLTx	FLTx	FLTx	FLTx	FLTx

- AppFaultStatus field description

Table 13. AppFaultStatus field description

Field	Description
FLT_0	OverDCBusVoltage (Over-voltage on DC bus)
FLT_1	UnderDCBusVoltage (Under-voltage on DC bus)
FLT_2	OverDCBusCurrent (Over-current on DC bus)
FLT_3	OverLoad (Overload Flag)
FLT_4	MainsFault (Mains out of range)
FLT_5	WrongHardware (Wrong hardware fault flag)
FLT_6	OverHeating (Overheating fault flag)
FLT_7	OverPhaseACurrent (Over-current on phase A)
FLT_8	OverPhaseBCurrent (Over-current on phase B)
FLT_9	OverPhaseCCurrent (Over-current on phase C)

Table continues on the next page...

Table 13. AppFaultStatus field description (continued)

FLT_10	OffCancError (Offset cancellation error)
FLT_21	FOCError (error in FOC calculation function)
FLT_22	AlignError (error during alignment)
FLT_23	CalibError (error during ADC calibration)
FLT_24	InitError (error during app initialization)
FLT_29	FLEXPWM_Error (error in FlexPWM hw initialization)
FLT_30	ADC_Error (error in ADC hw initialization)
FLT_31	CTU_Error (error in CTU hw initialization)

There are two approaches to detect the fault state. Over-currents and over-voltage detection is performed by external comparators with output brought to the Flex_PWM fault pin. These pins are red in the faultDetect() funtion within FOC_Fast_ISR() routine, Figure 14. on page 26. To detect fault FLT_5, a SIUL2 external interrupt is used to detect the INT pin on MC33937. The INT pin becomes high as soon as a fault appears and is detected by MC33937, [9].

6.4.4 State – READY

The state READY is a state that allows transition back to itself, provided no faults are present and the user does not request the start of the application. In other words, t his is a wait state where the application waits for the user to initiate a transition to a run state.

Transition to the RUN state is conditioned by the states CALIB and ALIGN. Transition to CALIB state is performed by switching the application on (setting event to `e_app_on`) , which is done automatically on the rising edge of `switchAppOnOff=true` in FreeMASTER or on the falling edge of the SW1 flop/flop switch on power stage board.

Transition to the FAULT state is performed automatically when a fault occurs.

6.4.5 State – CALIB

In this state, ADC DC offset calibration is performed. There are three ADC channels for 3-phase currents measurement as well as two ADC channels for resolver sine and cosine signals measurement.

When the state machine enters the CALIB state, all PWM outputs are enabled and resolver sensor excitation is disabled.

Calibration of the phase current DC offsets is achieved by generating 50% duty-cycle on the PWM outputs and applying a moving average filter to all configured current channels. The output of the filter represents an averaged DC offset value. The DC offset is stored for each current channel and is subtracted from the measured value when in normal operation. In this way, the half-range DC offset, caused by voltage shift of 1.65V in conditional circuitry, is removed in all three phases.

Calibration of the resolver output DC offsets requires disabling the resolver excitation signal which ensures zero voltage value on both sine and cosine signals. Then the same procedure as described in phase current DC offset calibration is performed. Acquired DC offset caused by voltage shift of 1.65V in conditional circuitry, is removed in both resolver signals when in normal operation. After successful calibration, the resolver excitation is enabled.

The state CALIB is a state that allows transition back to itself, provided no faults are present, the user does not requested stop of the application, and the calibration process has not finished. The calibration window is set by default to 1ms. When calibration counter reached 1ms and the averaged values are successfully saved, the application event is automatically set to `event=e_calib_done` and state machine can proceed to the state ALIGN.

Transition to the FAULT state is performed automatically when a fault occurs.

6.4.6 State – ALIGN

Alignment of the rotor and stator flux vectors is used to mark zero electrical position. This position can be used to calibrate position sensors.

When using a relative position sensor such as encoder, the initial position is not known, therefore the eTimer counter counting the encoder edges has to be correctly reset at the start of operation in the ALIGN state. When using an absolute position sensor such as resolver, the initial position is known but it might be loaded by an error due to the light misalignment of the sensor and rotor. Resolver position DC offset represents a non-zero value of measured electrical rotor position when rotor aligned to zero electrical position in the ALIGN state. The position offset value is stored and is subtracted from the measured value when in normal operation.

In the ALIGN state a DC voltage is applied in phase A for a certain period. This forces the rotor to turn to “align” position, that is, the stator and rotor fluxes are aligned. The rotor position in which the rotor stabilizes after applying this DC voltage is set as zero position.

To wait for the rotor to stabilize in the aligned position, the DC voltage is constantly applied during the period defined by an alignment duration. The alignment time duration and amplitude of the DC voltage is specified in the RESET state. Timing is implemented using a software counter that counts from a predefined 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 a predefined value and the event is automatically set to `event=e_align_done`. This enables transition to the RUN state.

Transition to the FAULT state is performed automatically when a fault occurs.

6.4.7 State – RUN

In this state, all calculations for the FOC algorithm as described in the section PMSM field-oriented control on page 2 are performed. Calculation of the fast current loop is executed every CTU-ADC interrupt when in the RUN state, while calculation of the slow speed loop is executed every N^{th} CTU-ADC interrupt. Arbitration is done using a counter that counts from zero up to value N. When value N is reached, the counter is reset back to zero and the slow speed loop calculation is performed. This way, only one interrupt is needed for both loops and timing of both loops is synchronized, that is, slow loop calculations are finished before entering fast loop calculations.

Transition to the INIT state is performed by switching the application off, in other words by setting event to `event=e_app_off`, which is done automatically on the falling edge of `switchAppOnOff=false` in FreeMASTER or on the rising edge of the SW1 flop/flop switch on power stage board.

Transition to the FAULT state is performed automatically when a fault occurs.

The following figure shows the implementation of the FOC algorithm with an overview of used functions and variable nomenclatures. The position and speed evaluation routines is implemented for the resolver sensor. FOC motor control functions are supported by Automotive Math and Motor Control Library Set for MPC574xP [8].

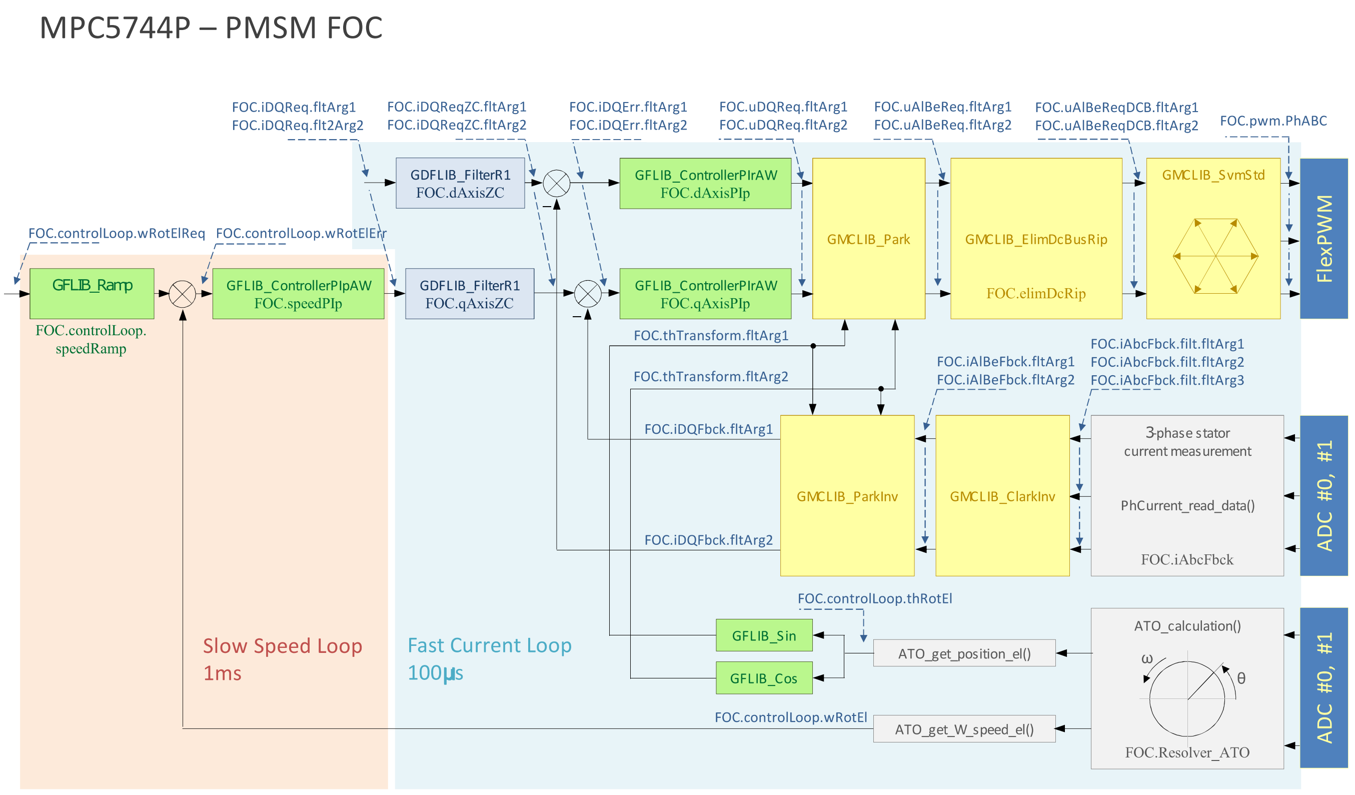


Figure 16. Variables/function name convention of implemented FOC algorithm

6.5 Current, speed, and position measurement

This section describes the routines for measurement of the FOC feedback variables such as phase currents, motor speed and position. The FOC feedback variables are obtained by calling the *getMotorControlVariables()* function periodically in CTU-ADC interrupt service routine *FOC_one_Fast_ISR()* and *FOC_two_Fast_ISR()*.

6.5.1 Current measurement

Due to several available hardware configurations (PCIe interface J1 and J200) the current measurement hardware initialization is done by the `PhCurrent_meas_hw_init()` function called in the RESET state.

Three phase currents can be obtained by calling either the `PhCurrent2Ph_get_data()` or `PhCurrent3Ph_get_data()` function. The `PhCurrent2Ph_get_data()` function provides the measurement only of two currents at any time, and third one is calculated according to Phase current measurement on page 6. The currents to be measured and current to be calculated depend on a sector in which the actual output voltage vector lies. The sector is calculated by `GMCLIB_SvmStd()` function, which generates three phase duty-cycles for the inverter by employing Space Vector Modulation technique.

The function `PhCurrent3Ph_get_data()` provides the measurement of all three currents at any time. This approach requires a minimal PWM pulse width for successful current sampling, hence it cannot be used up to full PWM duty-cycle. This technique is used in the project and minimal pulse width is achieved by limitation of the required voltages to 90% of full duty-cycle.

6.5.2 Resolver position and speed measurement

Section Resolver signal processing on page 7 describes the ATO approach to extract the information about the speed and position from the resolver output signals.

Due to several available hardware configurations (PCIe interface J1 and J200) the resolver hardware initialization is done by the `Resolver_hw_init()` function called in the RESET state. Resolver sine and cosine signals are acquired by calling the `Resolver_get_data()` function. The speed and position are then obtained by calling the following functions: `ATO_calculation()`, `ATO_get_position_el()`, and `ATO_get_w_speed_el()`. The position error for the ATO algorithm is calculated in separate function `ATO_theta_err_calc_resolver()` because different sensor requires different approach to calculate the position error.

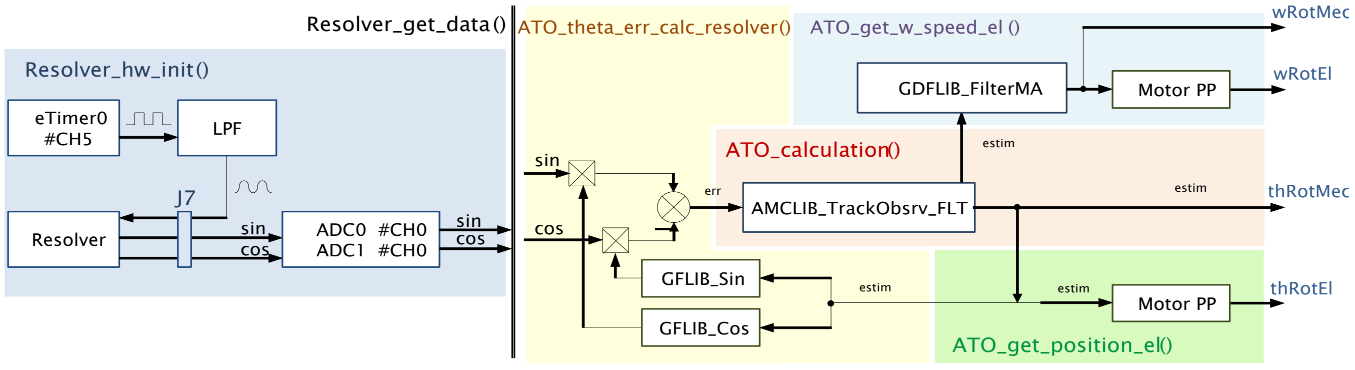


Figure 17. ATO approach for resolver speed and position measurement

The resolver is an absolute position sensor, that is, there is no need for a mechanical alignment. However, the resolver must either:

- be mounted precisely on the rotor shaft, 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 has to be known a priory.

This offset is then always subtracted from the measured position. Initialization of the resolver offset is done at the end of the align procedure (ALIGN state), by writing the value measured from the resolver into the offset.

7 Application control user interface

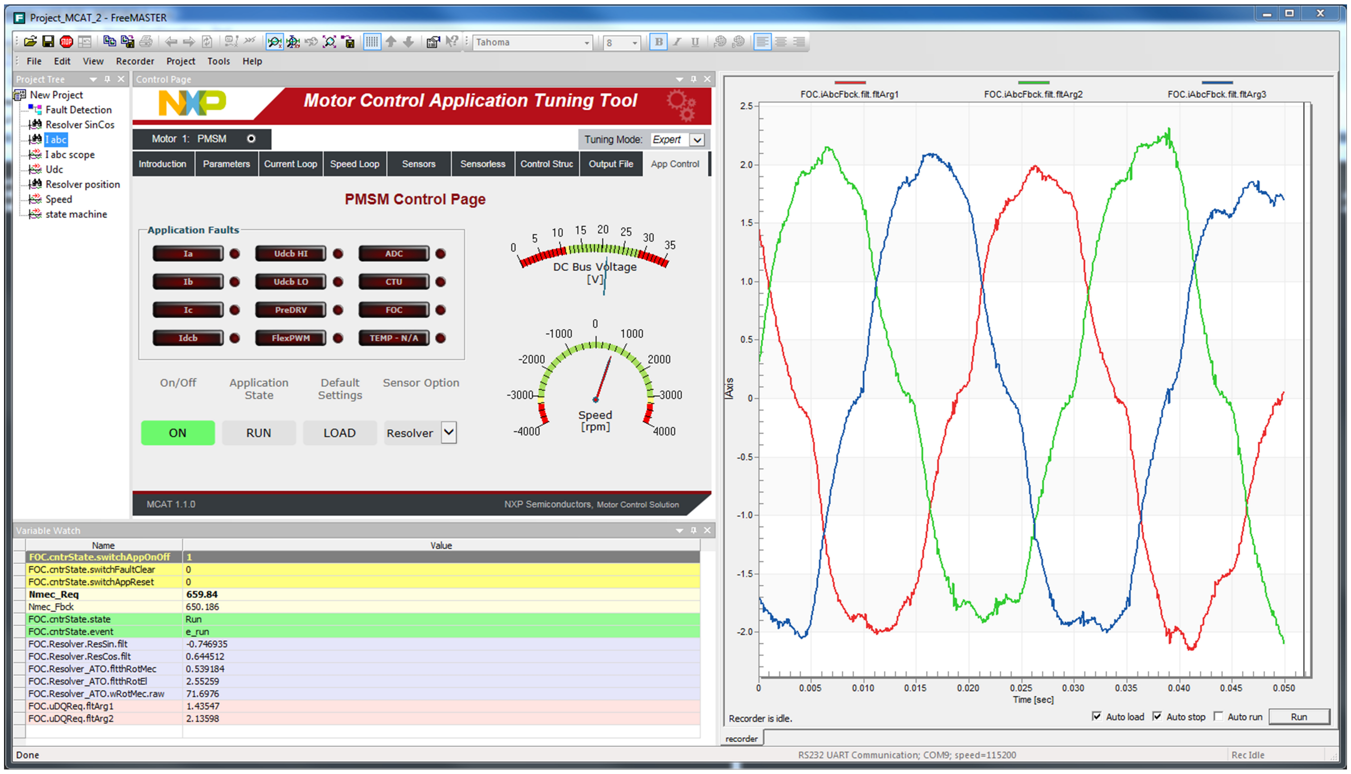
To control the application and monitor the variables in runtime, use the NXP real-time debug monitor and data visualization tool FreeMASTER [5]. The example software package of the MPC5744P Development Kit contains a preconfigured FreeMASTER example project. An integral part of the FreeMASTER project is the Motor Control Application Tuning (MCAT) tool [6] to help the user tune the FOC application.

7.1 FreeMASTER

Communication with the host PC is made via USB. Because FreeMASTER supports RS232 communication, there must be a driver installed on the host PC that creates a virtual COM port from the USB. Use this COM port for the FreeMASTER communication.

The application configures the LINFlex module of the MPC5744P for communication speed of 115 200 bps. Set FreeMASTER to the same speed by navigating to FreeMASTER menu \Project>Options> and selecting the Comm tag.

- Watch on-board variables behavior in various formats using FreeMASTER
- Monitor the real-time behavior of the phase currents and other motor quantities using FreeMASTER recorder.
- Control the application using MCAT Control Page, Figure 18. on page 34.
- Tune the motor application using MCAT



Variable Watch

Name	Value
FOC.cntrState.switchAppOnOff	1
FOC.cntrState.switchFaultClear	0
FOC.cntrState.switchAppReset	0
Nmec_Req	**659.84**
Nmec_Fbck	650.186
FOC.cntrState.state	Run
FOC.cntrState.event	e_run
FOC.Resolver.ResSin.filt	-0.746935
FOC.Resolver.ResCos.filt	0.644512
FOC.Resolver_ATO.fltthRotMec	0.539184
FOC.Resolver_ATO.fltthRotEl	2.55259
FOC.Resolver_ATO.wRotMec.raw	71.6976
FOC.uDQReq.fltArg1	1.43547
FOC.uDQReq.fltArg2	2.13598

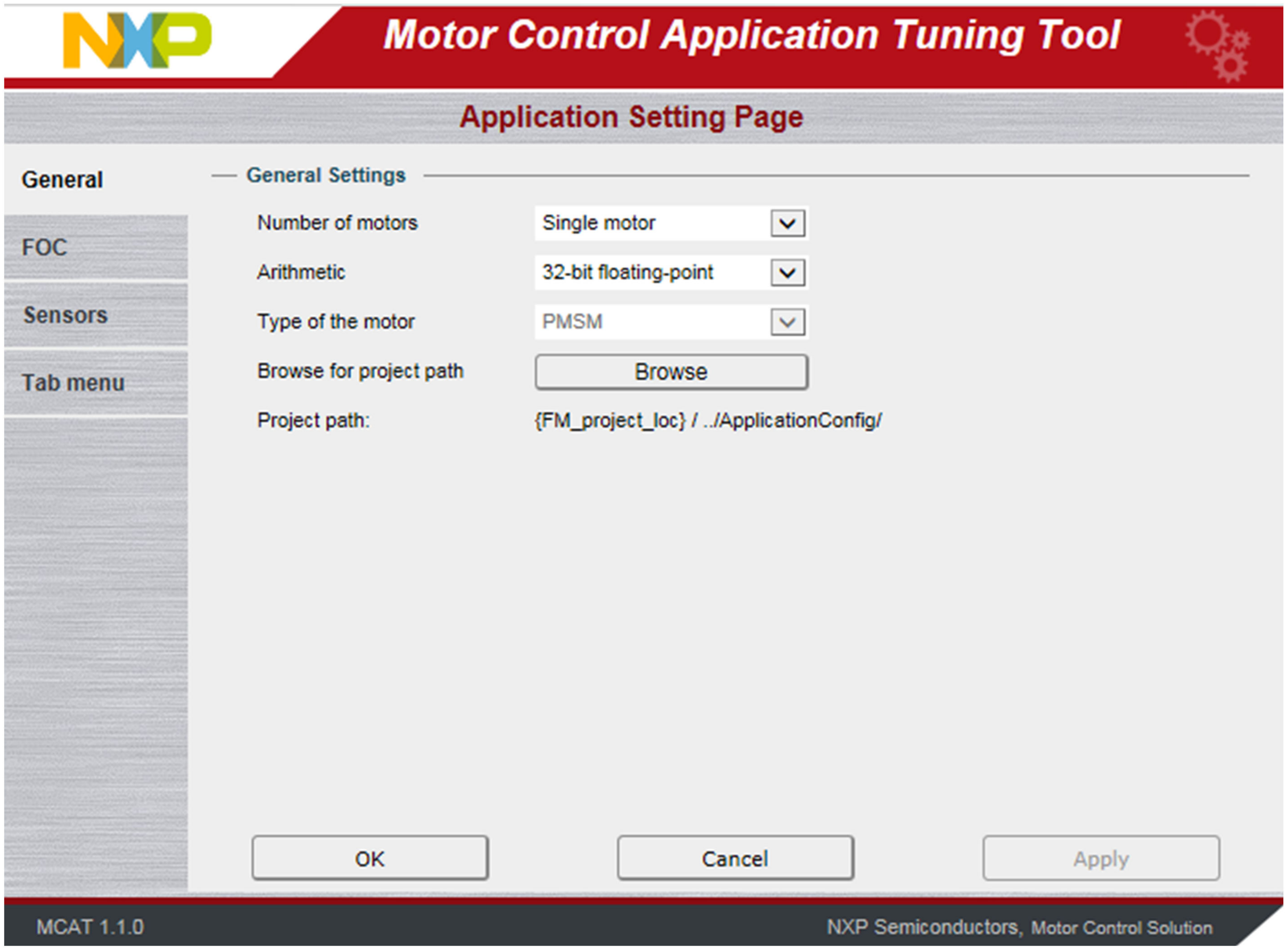
Done

Figure 18. FreeMASTER real-time monitor with MCAT control page

7.2 Motor Control Application Tuning (MCAT) tool

The MCAT tool is a user-friendly graphical plug-in tool for FreeMASTER, which allowstuning and controlling the 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 static configuration header file. For more details about MCAT see [6, 7] .

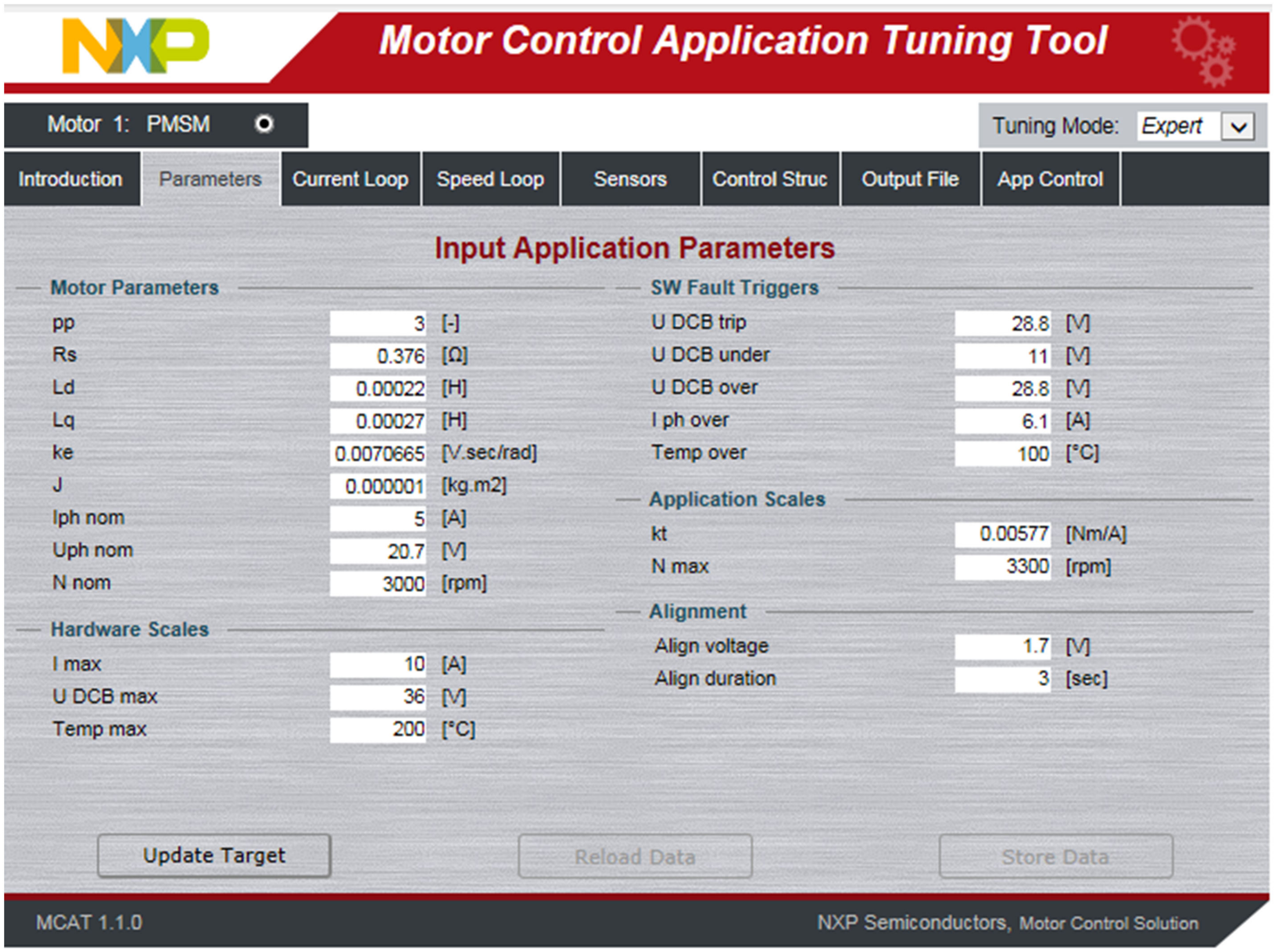
The MCAT should be first configured to be aligned with the application, Figure 19. on page 35. Click the Setting icon in the right upper corner to open the setting window and perform the general settings, FOC structure settings, chose the application mechanical sensors, and organize the panels in the MCAT tab menu. All changes are stored clicking the OK button.



Number of motors: Single motor
Arithmetic: 32-bit floating-point
Type of the motor: PMSM
Browse for project path: Browse
Project path: {FM_project_loc} / ../ApplicationConfig/

Figure 19. MCAT Setting page

The next mandatory step is to fill the parameters of the application. For that purpose, enter the Parameters tab. Store Data button stores all changes in the panel, Reload Data loads the default values, and Update Target button updates the Alignment parameters to the MCU.



Input Application Parameters

Motor Parameters

pp	3	[-]
Rs	0.376	[Ω]
Ld	0.00022	[H]
Lq	0.00027	[H]
ke	0.0070665	[V.sec/rad]
J	0.000001	[kg.m2]
Iph nom	5	[A]
Uph nom	20.7	[V]
N nom	3000	[rpm]

Hardware Scales

I max	10	[A]
U DCB max	36	[V]
Temp max	200	[°C]

SW Fault Triggers

U DCB trip	28.8	[V]
U DCB under	11	[V]
U DCB over	28.8	[V]
I ph over	6.1	[A]
Temp over	100	[°C]

Application Scales

kt	0.00577	[Nm/A]
N max	3300	[rpm]

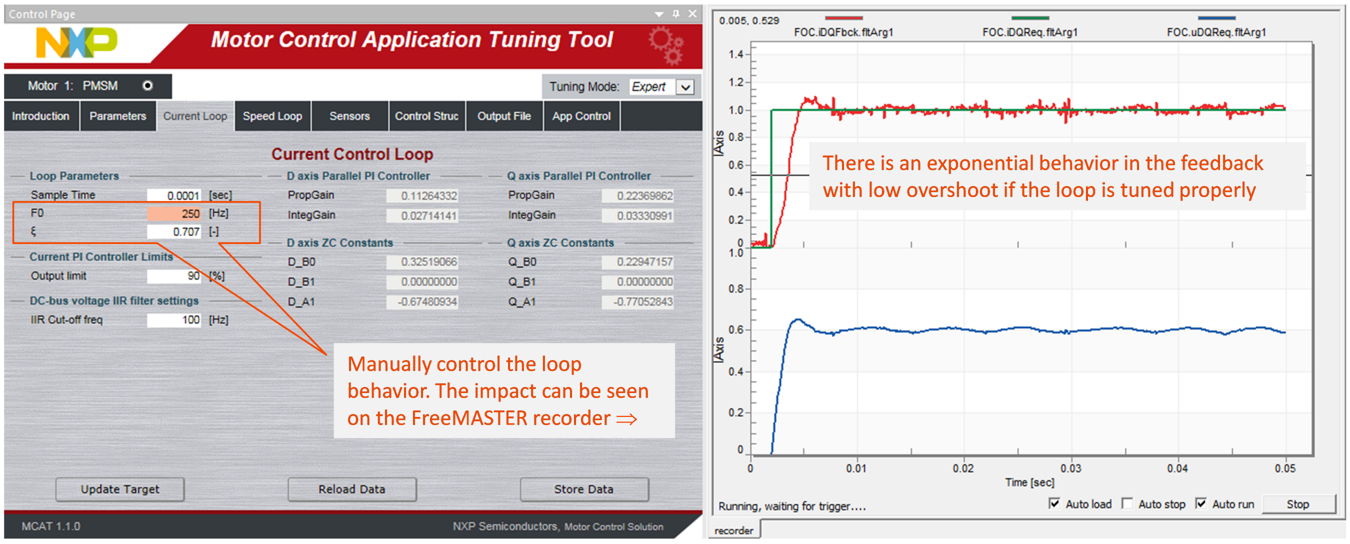
Alignment

Align voltage	1.7	[V]
Align duration	3	[sec]

Figure 20. MCAT Input Application Parameters page

Once the motor and application parameters are entered, the tuning process can start. MCAT is fully compatible with speed FOC structure with current and speed loop. Speed and position signal processing routines can be tuned by MCAT as well. There is an option to select either resolver sensor, encoder sensor, or sensorless model-based approach in the MCAT setting panel.

As an example, the current loop tuning can be seen in figure below. FreeMASTER recorder can trigger required current and display the current response as well as the acting voltage value.


Control Page

Motor Control Application Tuning Tool

Motor 1: PMSM

Tuning Mode: Expert

Introduction | Parameters | Current Loop | Speed Loop | Sensors | Control Struc | Output File | App Control

Current Control Loop

Loop Parameters		
Sample Time	0.0001	[sec]
F0	250	[Hz]
ξ	0.707	[-]

Current PI Controller Limits		
Output limit	90	[%]

DC-bus voltage IIR filter settings		
IIR Cut-off freq	100	[Hz]

D axis Parallel PI Controller		Q axis Parallel PI Controller	
PropGain	0.11264332	PropGain	0.22369862
IntegGain	0.02714141	IntegGain	0.03330991

D axis ZC Constants		Q axis ZC Constants	
D_B0	0.32519066	Q_B0	0.22947157
D_B1	0.00000000	Q_B1	0.00000000
D_A1	-0.67480934	Q_A1	-0.77052843

Manually control the loop behavior. The impact can be seen on the FreeMASTER recorder ⇒

Update Target | Reload Data | Store Data

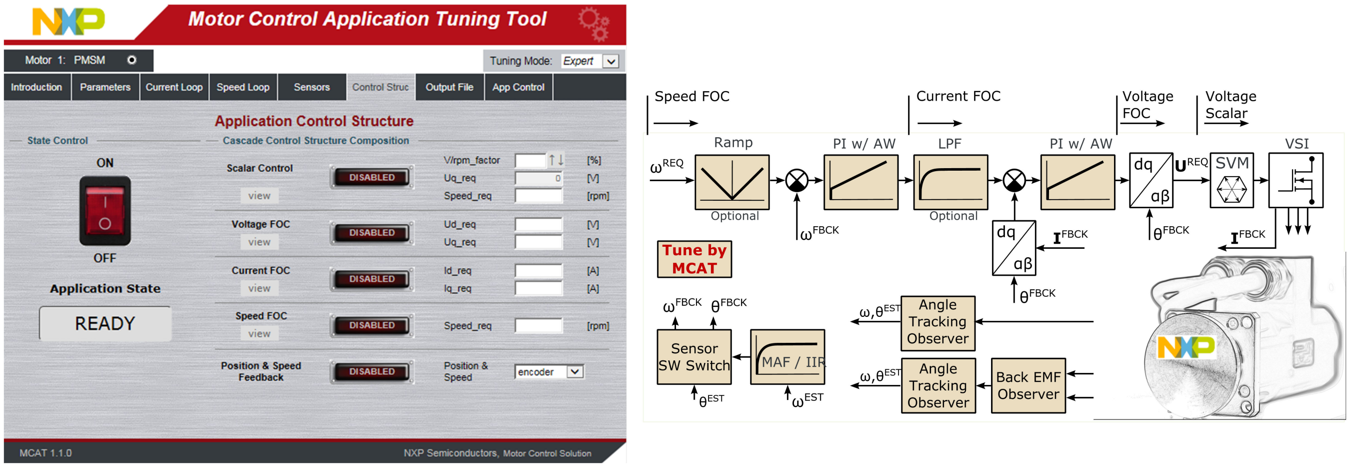
MCAT 1.1.0 — NXP Semiconductors, Motor Control Solution

There is an exponential behavior in the feedback with low overshoot if the loop is tuned properly

Running, waiting for trigger....

Figure 21. MCAT tuning process – current loop

An Application Control Structure page brings an advantage of MCAT to control the motor at different level of cascade control structure, figure below. Scalar control offers a possibility to control the motor in open loop manner without any feedback request. Next level is a voltage FOC control where only position information is required. Current FOC incorporates first PI controllers in the loop and requires the current information. The full control is Speed FOC where all control loops are closed.


Motor Control Application Tuning Tool

Motor 1: PMSM

Tuning Mode: Expert

Introduction | Parameters | Current Loop | Speed Loop | Sensors | Control Struc | Output File | App Control

Application Control Structure

State Control: ON / OFF

Application State: READY

Cascade Control Structure Composition

Scalar Control	DISABLED	V/rpm_factor		[%]
		Uq_req	0	[V]
		Speed_req		[rpm]
Voltage FOC	DISABLED	Ud_req		[V]
		Uq_req		[V]
Current FOC	DISABLED	Id_req		[A]
		Iq_req		[A]
Speed FOC	DISABLED	Speed_req		[rpm]
Position & Speed Feedback	DISABLED	Position & Speed	encoder	

MCAT 1.1.0 — NXP Semiconductors, Motor Control Solution

Figure 22. MCAT Application Control Structure page

The application configuration file is generated in the Output File tab. Clicking the button a header file is generated to the configured path which is selected in the Setting page.

7.3 MCAT control page

MCAT control page simplifies the control task using the speed gauge and buttons to control the motor, and displays the application actual state and faults, figure below.

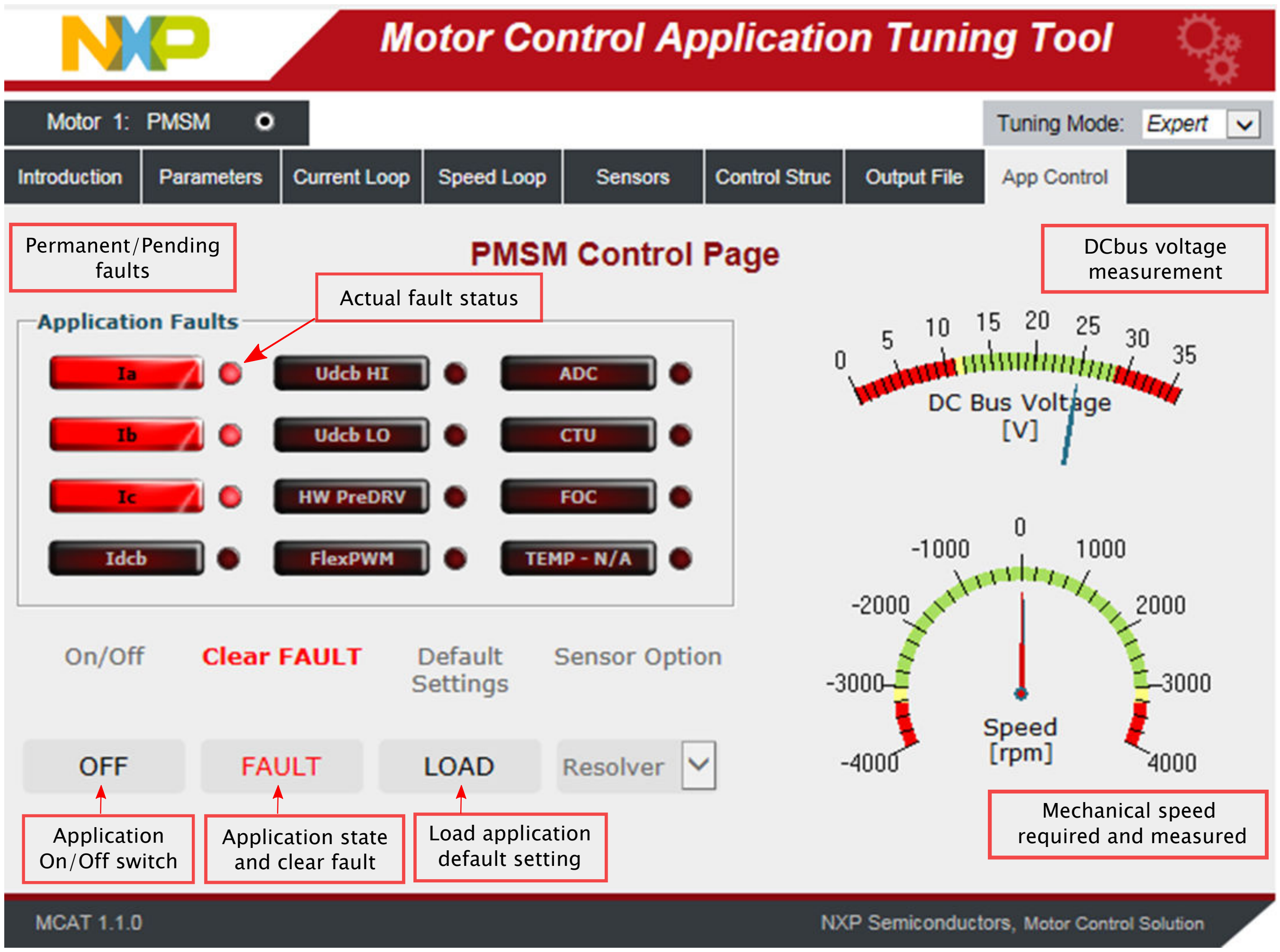


Figure 23. MCAT Control page

The Application State field displays the actual state machine status. In case of the Fault state:

- The permanent/pending faults are indicated by a highlighted red bar with the name of the fault source.
- The actual faults are indicated by the circle-shaped LED-like indicator, which is placed next to the bar with the name of the fault source.
- The actual presence of a fault is indicated by highlighting the respective LED-like indicator.
- The RUN button changes to FAULT and is highlighted by red color.

If a fault state occurs for any reason, the application switches off automatically, and displays the FAULT application state, indicated by a pink frame indicator.

When all actual fault sources are removed (which is indicated by none of the LED-like fault indicators being lit), clear the pending faults by pressing the FAULT CLEAR button that appears instead of the RUN button. This clears all pending faults and enables the transition of the state machine into the INIT state and then to the READY state.

When the application faults are cleared and the application pass to READY state, all state variables of the control algorithm a re set to zero , but all control algorithm parameters do not change their value.

Load the default setting and the algorithm parameters pressing Load button. Load button enables the transition to the RESET (then – INIT – READY) state where the application is set to its default configuration defined by `PMSM_appconfig.h`.

Start the application by clicking the "On/Off" button.

In the RUN state, all control loops of the FOC algorithm are active. That means that the three-phase currents are measured and used to close the current loop, and the actual rotor speed is measured to close the speed loop. Select the desired speed of the rotor in the variable watch window. The variable for controlling the speed is called Nreq; it is the required speed recalculated to the mechanical speed in revolutions per minute.

Switch the application off by clicking the On/Off button, and it proceeds to the INIT state and consequently to the READY state. Because of the motor used, select the required speed in the range from -3000 rpm to 3000 rpm.

The field-weakening algorithm is not implemented, and the required value of the d-axis current is set to zero.

8 References

1. MPC574xP MCU, product page
2. MPC5744P Reference Manual
3. MPC5744P Controller Board user manual
4. 3-phase Low Voltage Power Stage User Manual
5. FreeMASTER Run-Time Debugging Tool
6. Motor Control Application Tuning (MCAT) Tool
7. AN4642, Motor Control Application Tuning (MCAT) Tool for 3-phase PMSM
8. Automotive Math and Motor Control Library Set for MPC5744P
9. Three Phase Field Effect Transistor Pre-driver MC33937

9 Revision history

This section documents the changes done in this document.

Revision number	Date	Substantive changes
0	08/2017	Initial release
1	08/2018	Added documentation for dual motor control including the software configuration, peripheral descriptions for two motors, and an updated timing diagram

Document Number: AN12017
Rev. 1, August, 2018