AN14313

Single 3-phase 48V PMSM Motor Control Kit with S32K344 using RTD low-level API (Featuring Motor Control Application Tuning (MCAT) tool)

Rev. 2.0 — 5 June 2025 Application note

Document information

Information	Content
Keywords	48V, S32K3, Motor control, Field oriented control (FOC), Sensorless, Encoder sensor, Hall sensor, Resolver sensor
Abstract	This application note describes the design of a single 3-phase Permanent Magnet Synchronous Motor Field-Oriented Control 48 V drive with 3-shunt current sensing with and without position sensor.

Single 3-phase 48V PMSM Motor Control Kit with S32K344 using RTD low-level API (Featuring Motor Control Application Tuning (MCAT) tool)

1 Introduction

This application note describes the design of a single 3-phase Permanent Magnet Synchronous Motor (PMSM) Field Oriented Control (FOC) 48V drive with 3-shunt current sensing with and without position sensor.

This document serves as an example of motor control design using NXP S32K3 automotive family with MCUs based on a 32-bit Arm®Cortex®-M7 core with IEEE-754 compliant single-precision floating point unit optimized for a full range of automotive applications. An innovative drivers set, Real-Time Drivers (RTD), is used to configure and control the MCU. It complies with Automotive-SPICE, ISO 26262, ISO 9001, and IATF 16949. Low-level drivers of RTD and S32 Design Studio Configuration Tools (S32CT) are used to demonstrate the non-AUTOSAR approach.

Following are the supported features:

- 3-phase PMSM speed field oriented control with 3-shunt resistors.
- Shaft position and speed estimated either by sensorless algorithm or encoder or resolver or hall position sensor.
- Application control user interface using FreeMASTER debugging tool.
- Motor Control Application Tuning (MCAT) tool.

2 System concept

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

- It is based on the NXP 48V MC development platform (see [1]for more information).
- Real-Time Drivers (RTD) and S32CT (non-AUTOSAR) used as S32K344 device configuration and control tool being a part of the S32 Design Studio for S32 Platform (S32DS), NXP's complimentary integrated development environment (IDE) (see section [2], [3]).
- Control technique incorporating:
  - Field oriented control of single 3-phase PM synchronous motor with and without position sensor.
  - Closed-loop speed control with an action period of 1 ms.
  - Closed-loop current control with action period of 100 µs.
  - Bi-directional rotation.
  - Flux and torque independent control.
  - Field weakening control extending speed range of the PMSM beyond the base speed.
  - Position and speed are estimated by either Extended Back Electromotive Force (eBEMF) observer or obtained by encoder, resolver, or hall sensor.
  - Robust open-loop start up with two-stage alignment.
  - Measurement of three-phase motor currents from three shunt resistors.
  - FOC state variables sampled with 100 µs period.
  - Soft-charge of DC-link capacitors.
  - Temperature measurement by NTC thermistors on power stage board [1].
- Automotive Math and Motor Control Library (AMMCLib) - FOC algorithm built on blocks of precompiled software library (see [5]).
- FreeMASTER software control interface (motor start/stop, speed setup) (see [4]).
- FreeMASTER software monitor (monitoring/visualization of application variables).
- FreeMASTER embedded Motor Control Application Tuning (MCAT) tool (motor parameters, current loop, sensorless parameters, speed loop) (see [9]).
- FreeMASTER software MCAT 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 (motor currents, vector control algorithm quantities).
- DC-Link and Battery over/undervoltage, overcurrent, overload and start-up fail protection.

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, Field Oriented Control is used for PM synchronous motors.

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 magnetomotive force is considered, the torque T_e developed by an AC machine, in vector notation, is given by the following equation:

$$T_e = \frac{3}{2} \cdot pp \cdot \bar{\psi}_s \times \bar{i}_s \tag{1}$$

where:

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

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 implies that stator current vector should be oriented in such a way that the component necessary for magnetizing of the machine (flux component) shall be 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 a 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 should 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 or position observer 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.

Setting the reference frame speed synchronously with the rotor flux axis further results into d and q axis current components appearing as 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 the following four steps:

1. Measure the motor quantities (DC link voltage and currents, rotor position/speed).

2. Transform measured currents into the two-phase orthogonal system (α, β) using a Clarke transformation. After that transform the currents in α, β coordinates into the d, q reference frame using a Park transformation.
3. The stator current torque (i_{sq}) and flux (i_{sd}) producing components are separately controlled in d, q rotating frame.
4. The output of the control is stator voltage space vector and it is transformed by an inverse Park transformation back from the d, q reference frame into the two-phase orthogonal system fixed with the stator. The output three-phase voltage is generated using a space vector modulation.

Clarke/Park transformations discussed above are part of the Automotive Math and Motor Control Library set (see [5]).

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 knowledge of accurate rotor position as being strictly fixed with magnetic flux. This application note deals with the FOC control where the position and velocity are obtained by either a position/velocity estimator or incremental encoder sensor or resolver sensor or hall sensors.

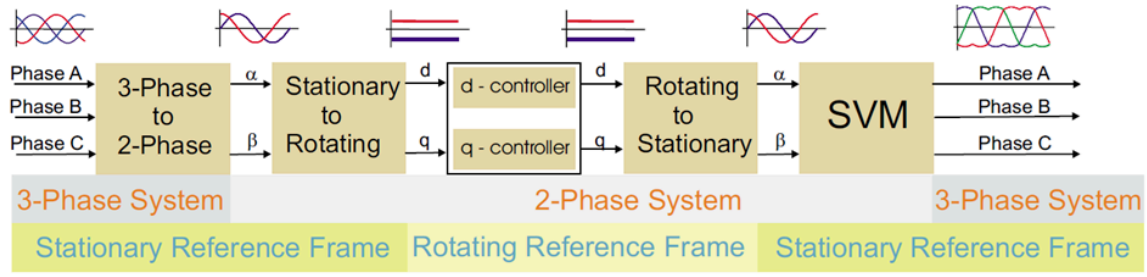


Figure 1. Field oriented control transformations

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:

$$\begin{bmatrix} u_a \\ u_b \\ u_c \end{bmatrix} = R_s \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_a \\ \psi_b \\ \psi_c \end{bmatrix} \tag{2}$$

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

$$\begin{bmatrix} \psi_a \\ \psi_b \\ \psi_c \end{bmatrix} = \begin{bmatrix} L_{aa} & L_{ab} & L_{ac} \\ L_{ba} & L_{bb} & L_{bc} \\ L_{ca} & L_{cb} & L_{cc} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \Psi_{PM} \begin{bmatrix} \cos(\theta_e) \\ \cos\left(\theta_e - \frac{2\pi}{3}\right) \\ \cos\left(\theta_e + \frac{2\pi}{3}\right) \end{bmatrix} \tag{3}$$

where:

- L_{aa}, L_{bb}, L_{cc} are stator phase self-inductances.
- L_{ab}=L_{ba}, L_{bc}=L_{cb}, L_{ca}=L_{ac} are mutual inductances 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.

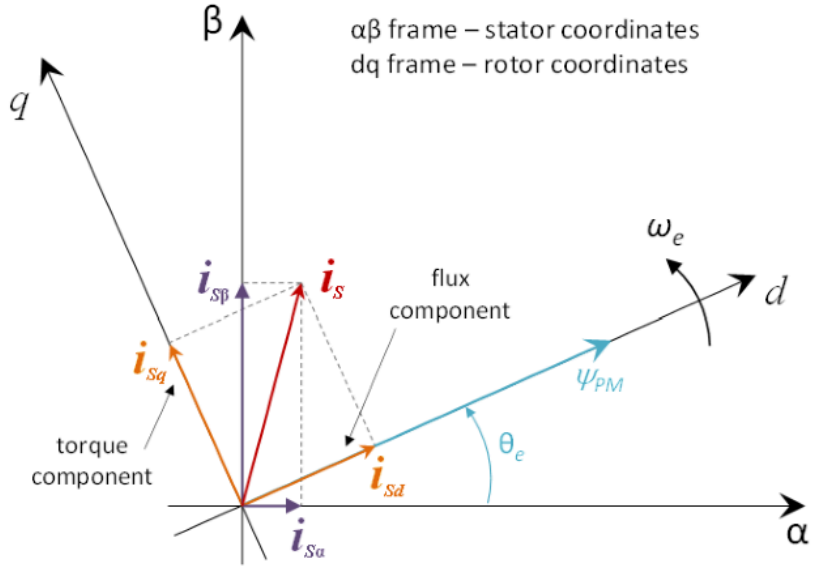


Figure 2. Orientation of stator (stationary) and rotor (rotational) reference frames, with current components transformed into both frames

The voltage equation of the quadrature phase synchronous reference frame model can be obtained by transforming the three phase voltage equations Equation 2 and flux equations Equation 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:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} 0 & -L_q \\ L_d & 0 \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \Psi_{PM} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (4)$$

where ω_e is electrical rotor speed

It can be seen that Equation 4 represents a non-linear cross dependent system, with cross-coupling terms in both d and q axis and BEMF 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 after compensation:

$$u_d = R_s i_d + L_d \frac{di_d}{dt} \quad (5)$$

$$u_q = R_s i_q + L_q \frac{di_q}{dt} \quad (6)$$

This equation 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:

$$K_p = 2\xi\omega_0 L - R \quad (7)$$

$$K_i = \omega_0^2 L \quad (8)$$

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

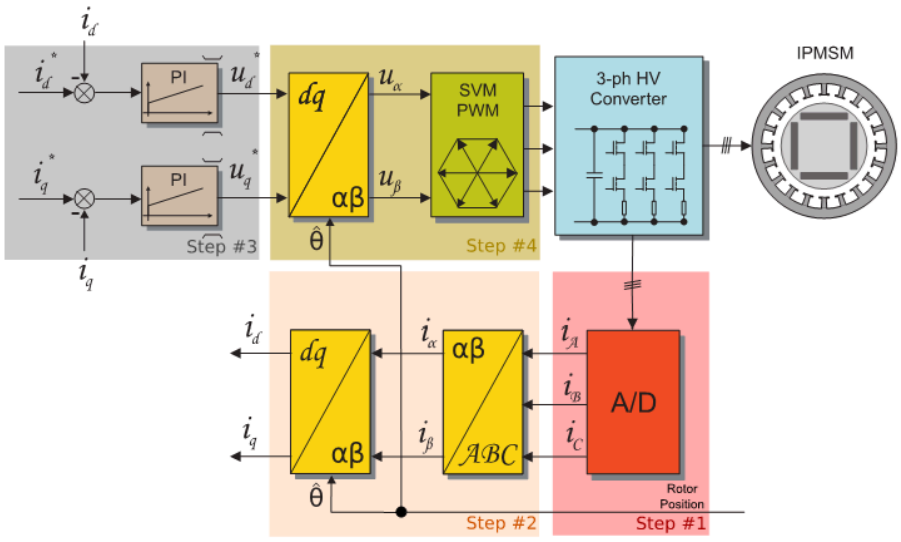


Figure 3. FOC Control Structure

3.3 Phase current measurement and output voltage actuation

One leg of 3-phase voltage source inverter, shown in Section 3.3, uses parallel connection of two shunt resistors (R31 and R32 for phase A) placed in low side of the inverter as phase current sensors. Stator phase current which flows through the shunt resistor produces a voltage drop which is interfaced to the Analog-to-Digital Converter (ADC) of microcontroller through conditional circuitry. Each leg of inverter uses parallel connection of two shunt resistors namely, R45, R46 for phase B, R58 and R59 for phase C, R74, R75 for phase D, R87 and R88 for phase E and R100 and R101 for phase F.

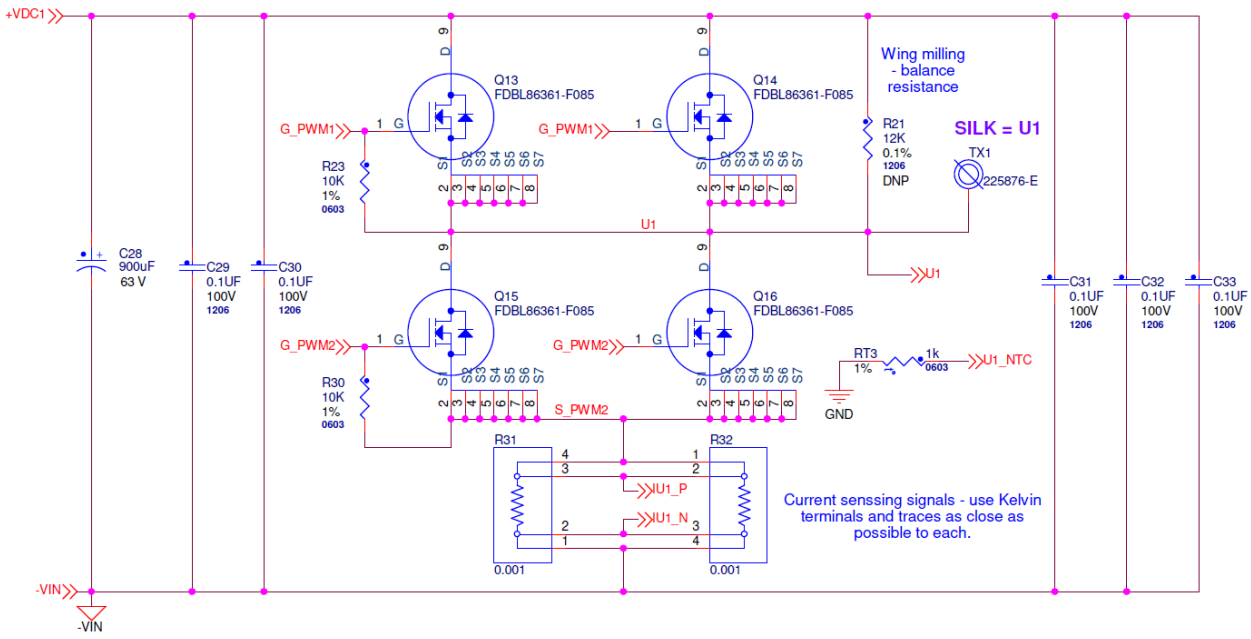


Figure 4. 3-phase DC/AC inverter with shunt resistors for current measurement

Section 3.3 shows a gain setup and input signal filtering circuit for operational amplifier which provides the conditional circuitry and adjusts voltages to fit into the ADC input voltage range. For more details about power circuit arrangement and current sensing conditional circuitry, please refer to [1].

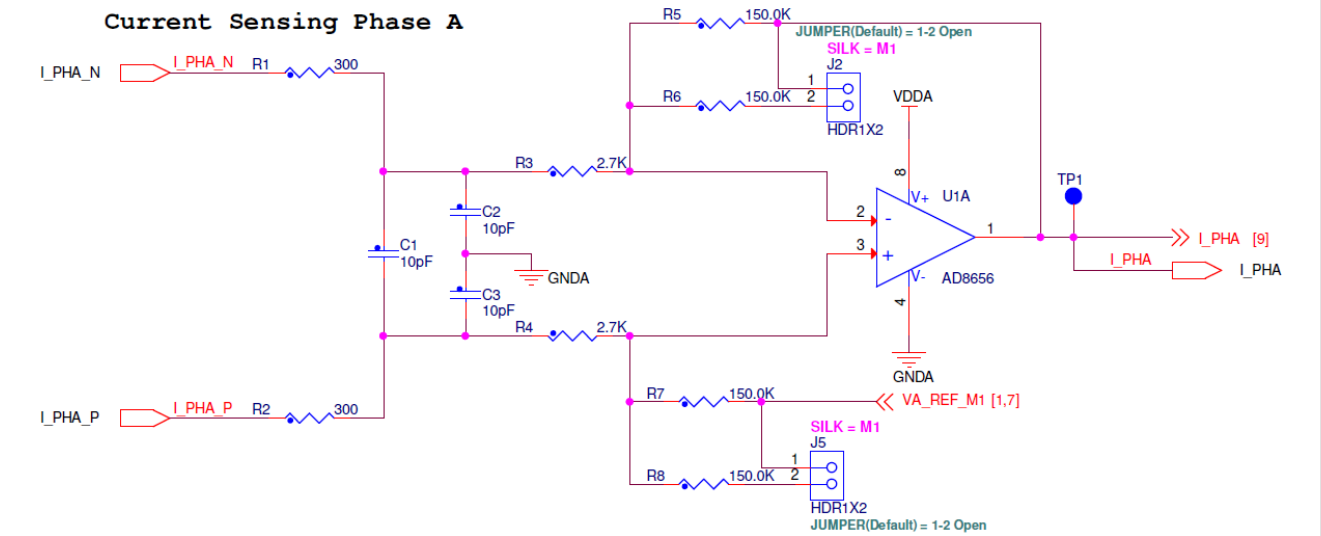


Figure 5. Phase current measurement conditional circuitry

The phase current sampling technique is a challenging task for detection of phase current differences and for acquiring full three phase information of stator current by its reconstruction. Phase currents flowing through shunt resistors produce a voltage drop which needs to be appropriately sampled by the ADC 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 only flows through the shunt resistor when the bottom transistor of the respective inverter leg is switched on. Therefore, phase A current is measured using the R31 and R32 shunt resistors and can only be sampled when the low side transistors Q15 and Q16 are switched on. Correspondingly, the current in phase B can only be measured if the low side transistors Q19 and Q20 are switched on, and the current in phase C can only be measured if the low side transistors Q23 and Q24 are switched on. Same approach applies for phases D, E, F. To get an actual instant of current sensing, transistor switching combination needs to be known. For more details about schematic, please refer to [1].

Generated duty cycles (phase A, phase B, phase C) of two different PWM periods are shown in Figure 6. These phase voltage waveforms correspond to a center-aligned PWM with sine-wave 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 the 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 hardware design), the current cannot be correctly measured.

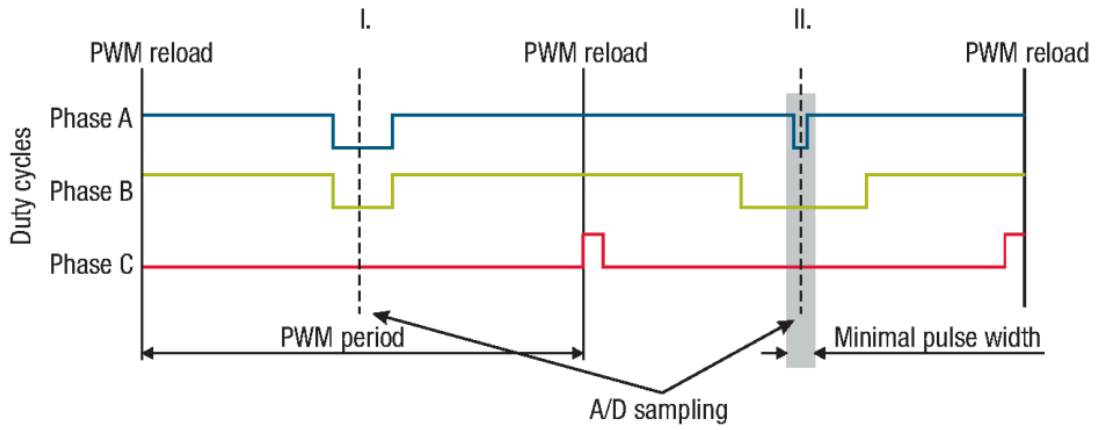


Figure 6. Generated phase duty cycles in different PWM periods

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 duty cycle goes to 100%, 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:

$$i_A + i_B + i_C = 0 \tag{9}$$

Note: *Although, there are three shunt resistors available on the power stage board for single 3-phase Voltage Supply Inverter (VSI) and S32K344 has three AD converters, only two currents are measured simultaneously in this application in order to demonstrate ADC Single-shot mode and BCTU control mode in parallel. Third stator current is calculated based on Equation 9. To measure two stator currents in two inverter legs correctly, selection of appropriate measurement combination of two shunts depends on Space Vector Modulation (SVM) state. See [15] for more information.*

3.4 Rotor position/speed estimation

In this application, rotor position and speed are either estimated sensorless by eBEMF observer or obtained by encoder or resolver or hall sensor. eBEMF observer as well as incremental encoder or hall sensors provide only relative position. To get absolute position, initial position must be known. This application uses electrical rotor alignment when the rotor is moved from unknown to known position. The two stage alignment process is described in detail in the section Section 4.3.3.5.

Application in sensorless mode must start with open loop start-up sequence to accelerate the motor up to a speed value where the observer provides sufficiently accurate speed and position estimations. As soon as the observer provides appropriate estimates, application transits to closed-loop mode, when the rotor speed and position calculation is based on the estimation of an eBEMF in the stationary reference frame using a Luenberger type of observer. eBEMF observer is a part of the NXP's Automotive Math and Motor Control library [5].

As soon as the rotor is aligned, application with encoder or hall or resolver can start from zero speed because speed and position are provided by sensor. FOC application using Hall sensors as a speed and position feedback is described in section Section 4.2.8.

Structure and implementation details are discussed in section Section 4.3.4.

Note: *Although, there is used resolver as an absolute position sensor, mechanical two stage alignment is applied. The reason is that mechanical offset between rotor position of motor and resolver initial position varies from one motor manufacturer to another. This brings a certain complexity and challenges of motor control application tuning in order to achieve maximum torque from zero speed.*

3.5 Field weakening

Field weakening is an advanced control approach that extends standard FOC to allow electric motor operation beyond the base speed. The back electromotive force (BEMF) is proportional to the rotor speed and counteracts the motor supply voltage. If a given speed is to be reached, the terminal voltage must be increased to match the increased stator BEMF. A sufficient voltage is available from the inverter in the operation up to the base speed. Beyond the base speed, motor voltages u_d and u_q are limited and cannot be increased because of the ceiling voltage given by inverter. Base speed defines the rotor speed at which the BEMF reaches maximal value and motor still produces the maximal torque.

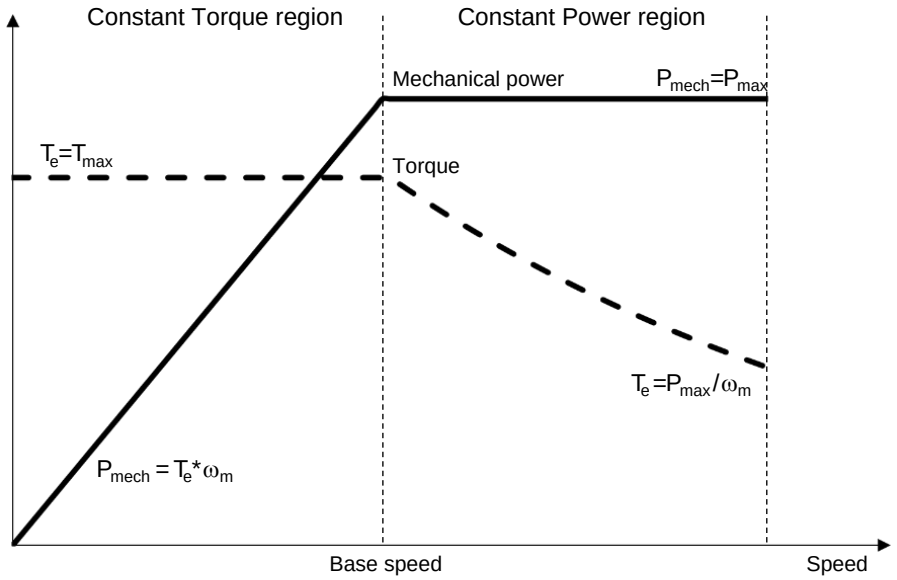


Figure 7. Constant torque/power operating regions

As the difference between the induced BEMF and the supply voltage decreases, the phase current flow is limited, hence the currents i_d and i_q cannot be controlled sufficiently. Further increase of speed would eventually result in BEMF voltage equal to the limited stator voltage, which means a complete loss of current control. The only way to retain the current control even beyond the base speed is to lower the generated BEMF by weakening the flux that links the stator winding. Base speed splits the whole speed motor operation into two regions: constant torque and constant power, see Figure 7.

Operation in constant torque region means that maximal torque can be constantly developed while the output power increases with the rotor speed. The phase voltage increases linearly with the speed and the current is controlled to its reference. The operation in constant power region is characterized by a rapid decrease in developed torque while the output power remains constant. The phase voltage is at its limit while the stator flux decreases proportionally with the rotor speed, see Figure 8.

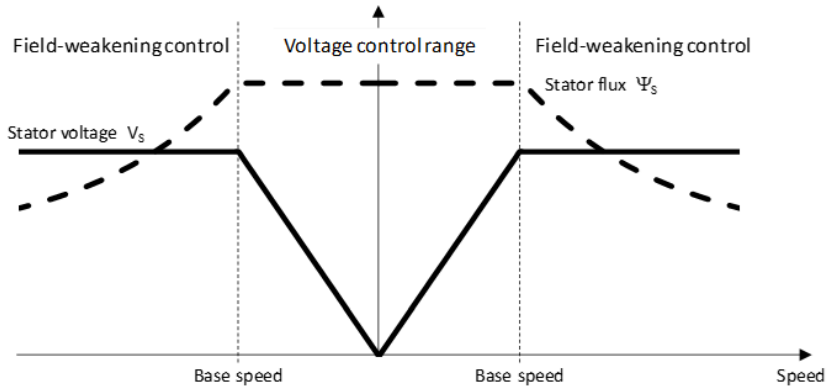


Figure 8. Constant flux/voltage operational regions

FOC splits phase currents into the q-axis torque component and d-axis flux component. The flux current component I_d is used to weaken the stator magnetic flux linkage Ψ_S. Reduced stator flux Ψ_S yields to lower BEMF and condition of Field Weakening is met. More details can be seen from the following phasor diagrams of the PMSM motor operated exposing FOC control without (left) and with FW (right), Figure 9.

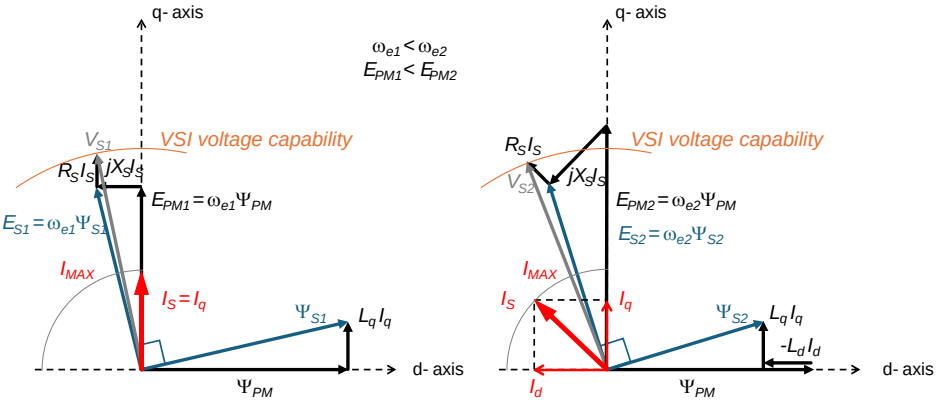


Figure 9. Steady-state phasor diagram of PMSM operation up to base speed (left) and above speed (right)

FOC without FW is operated demanding d-axis current component to be zero (I_d=0) to excite electric machine just by permanent magnets mounted on the rotor. This is an operation within constant torque region (see Figure 7), since whole amount of the stator current consists of the torque producing component I_q only (see Figure 9 left). Stator magnetic flux linkage Ψ_{S1} is composed of rotor magnetic flux linkage Ψ_{PM}, which represents the major contribution and small amount of the magnetic flux linkage in q-axis $L_q I_q$ produced by q-axis current component I_q. Based on the Faraday's law, rotor magnetic flux linkage Ψ_{PM} and stator magnetic flux linkage Ψ_{S1} produce BEMF voltage $E_{PM1}=\omega_{e1}\Psi_{PM}$ perpendicularly oriented to rotor magnetic flux Ψ_{PM} in q-axis and BEMF voltage $E_{S1}=\omega_{e1}\Psi_{S1}$ perpendicularly oriented to stator magnetic flux Ψ_{S1}, respectively (see Figure 9 left). Both voltages are directly proportional to the rotor speed ω_{e1}. If the rotor speed exceeds the base speed, the BEMF voltage $E_{S1}=\omega_{e1}\Psi_{S1}$ approaches the limit given by VSI and I_q current cannot be controlled. Hence, field weakening has to take place.

In FW operation, I_d current is controlled to negative values to "weaken" stator flux linkage Ψ_{S2} by $-L_d I_d$ component as shown in Figure 9 right. Thanks to this field weakening approach, BEMF voltage induced in the stator windings E_{S2} is reduced below the VSI voltage capability even though E_{PM2} exceeds it. I_q current can be controlled again to develop torque as demanded. Unlike the previous case, this is an operation within constant power region (see Figure 7), where I_q current is limited due to I_s current vector size limitation (see Figure 9 right). In FW operation, stator magnetic flux linkage Ψ_S consists of three components now: rotor magnetic flux linkage Ψ_{PM}, magnetic flux linkage in q-axis $\Psi_q = L_q I_q$ produced by q-axis current component I_q and magnetic flux linkage in d-axis $\Psi_d = -L_d I_d$ produced by negative d-axis I_d current component that counteracts to Ψ_{PM}.

There are some limiting factors that must be taken into account when operating FOC control with field weakening:

- Voltage amplitude *Us* is limited to *u_max* defined by actual DC bus voltage as shown in Figure 10 left.
- Phase current amplitude *i_max* is limited by capabilities of power devices and motor thermal design as shown in Figure 10 right.
- Flux linkage in d-axis is limited to prevent demagnetization of the permanent magnets.

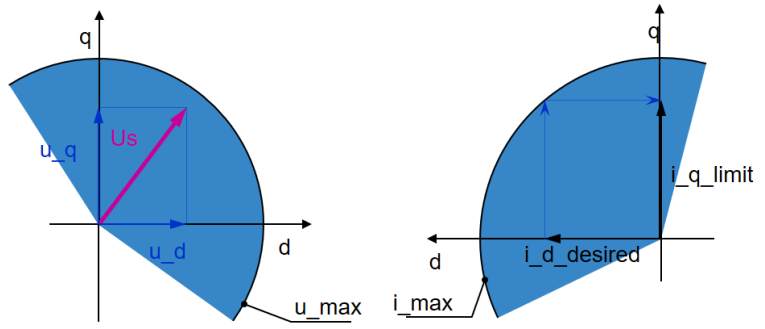


Figure 10. Voltage (left) and current (right) limits for PMSM drive operation

NXP's Automotive Math and Motor Control library offers a software solution for the FOC with field weakening respecting all limitations discussed above. This library-based function is discussed in section Section 4.3.4.

4 Software implementation on the S32K344

4.1 S32K344 – key modules for PMSM FOC control

The S32K344 device includes modules such as the Enhanced Modular IO Subsystem (eMIOS), Logic Control Unit (LCU), Trigger MUX (TRGMUX), Body Cross-triggering Unit (BCTU) and Analog-to-Digital Converter (ADC) suitable for real-time control applications, in particular, motor control applications. These modules are directly interconnected and can be configured to meet various motor control application requirements. Figure 11 shows a simplified module interconnection for a typical PMSM FOC application working in sensorless or sensor-based mode using triple shunt current sensing and either encoder, resolver or hall position sensor. Modules interconnection is described below and detailed description of each module can be found in the S32K3xx Reference Manual (see [7]). The NXP 48V development platform is equipped with two standalone 3-phase half bridges, which creates first and second motor configuration together with dedicated circuits. Therefore, this platform is capable to drive one 3-phase AC motor, two 3-phase AC motors or one 6-phase AC motor. In our case, the second motor configuration is used to drive single 3-phase PMSM motor. The SW configuration is linked with D-/E-/F-phases and second motor arrangement of encoder, resolver or hall sensor.

4.1.1 Module interconnection

The modules involved in output actuation, data acquisition and synchronization of actuation and acquisition, form the so-called Control Loop. This control loop consists of the eMIOS, LCU, TRGMUX, BCTU and ADC modules. The control loop is a modular concept and is very flexible in operation and can support static, dynamic or asynchronous timing.

eMIOS plays a role of the real time timer/counter. Within the control loop, it is responsible for generation of PWM signal (period, duty cycle), generation of the triggers for analog data capturing in the precise moment, counting edges of encoder signal, or capturing time stamp for FOC hall sensor application. LCU enriches this modular concept with advance features. In PWM generation it is responsible for creation of PWM complementary pairs, dead time insertion, disabling/enabling PWM outputs or it preprocess signals from an

encoder sensor to get quadrature decoder functionality. Moreover, it also preprocesses hall signals to get X-OR signal for speed and rotor position calculation.

BCTU and ADC modules are responsible for analog data capturing. BCTU answers question “what is going to be measured?” by a predefined list of ADC channels. The ADC answers question “How it is going to be measured?” by setting a conversion resolution, sampling duration etc.

eMIOS and LCU are connected through TRGMUX unit which is responsible for a configurable signal interconnection within the microcontroller. The eMIOS1 channels CH9-CH11 create 3-phase center aligned PWM signals and share PWM time base CH8. The center aligned PWM is formed using flexible Output Pulse Width Modulation Buffered (OPWMB) eMIOS mode where each channel uses 2 compare registers (A, B) to control rising and falling edge independently. LCU0 OUT0-OUT5 create complementary PWM pairs to control particular MOSFET transistors. The LCU uses Look Up Table, output polarity control and configurable digital filters to generate control signals for transistors with inserted deadtime. The eMIOS1 CH12 and CH13 are dedicated for trigger functionality. Same as in case of PWM signals, OPWMB mode is also used for trigger. Both trigger channels are linked with trigger time base CH23. Time bases CH8 and CH23 are synchronized, what offers possibility of an independent configuration of sampling and PWM frequency.

BCTU is linked with eMIOS channels through the channel flag. When the flag is set, BCTU starts to execute conversions according to the list of conversions and clears the flag back. BCTU is capable of controlling all three ADCs so list of single or parallel conversions can be invoked. In this example, two lists of parallel conversions of ADC0, ADC1 and ADC2 are used to obtain phase currents, Battery, DC-bus voltages and resolver sine and cosine signals. Conversion results are stored to BCTU FIFO.

Quadrature decoder functionality is achieved by cooperation of eMIOS, LCU and TRGMUX. LCU decodes encoder signals PHA and PHB into digital signals, which carry captured edges per particular rotor direction. The eMIOS0 module works as a counter and holds number of captured edges for clockwise CH5 and counterclockwise CH6 direction. Absolute position is obtained by subtracting counters values.

Resolver functionality is achieved by cooperation of eMIOS, ADC and BCTU modules. eMIOS1 CH21 provides reference square wave signal, which is subsequently filtered by third order Sallen Key low pass filter. This sine-wave output is used as an excitation for resolver. eMIOS1 CH13 is dedicated for resolver triggering functionality. Sine and Cosine signals of resolver are processed and filtered by HW active filters, where single ended types of signals are measured by ADC module in cooperation with BCTU.

Hall sensor functionality for FOC application is achieved by cooperation of eMIOS, LCU and TRGMUX modules. Three hall sensor signals are connected over TRGMUX inputs to the LCU, where X-OR signal is obtained from all three hall signals. This X-OR signal is subsequently connected to eMIOS0 CH7 over TRGMUX, where speed of rotor is measured. Rotor position is obtained by integration of speed signal.

Detailed description of real-time control modules can be found in S32K3xx Reference Manual (see [7]).

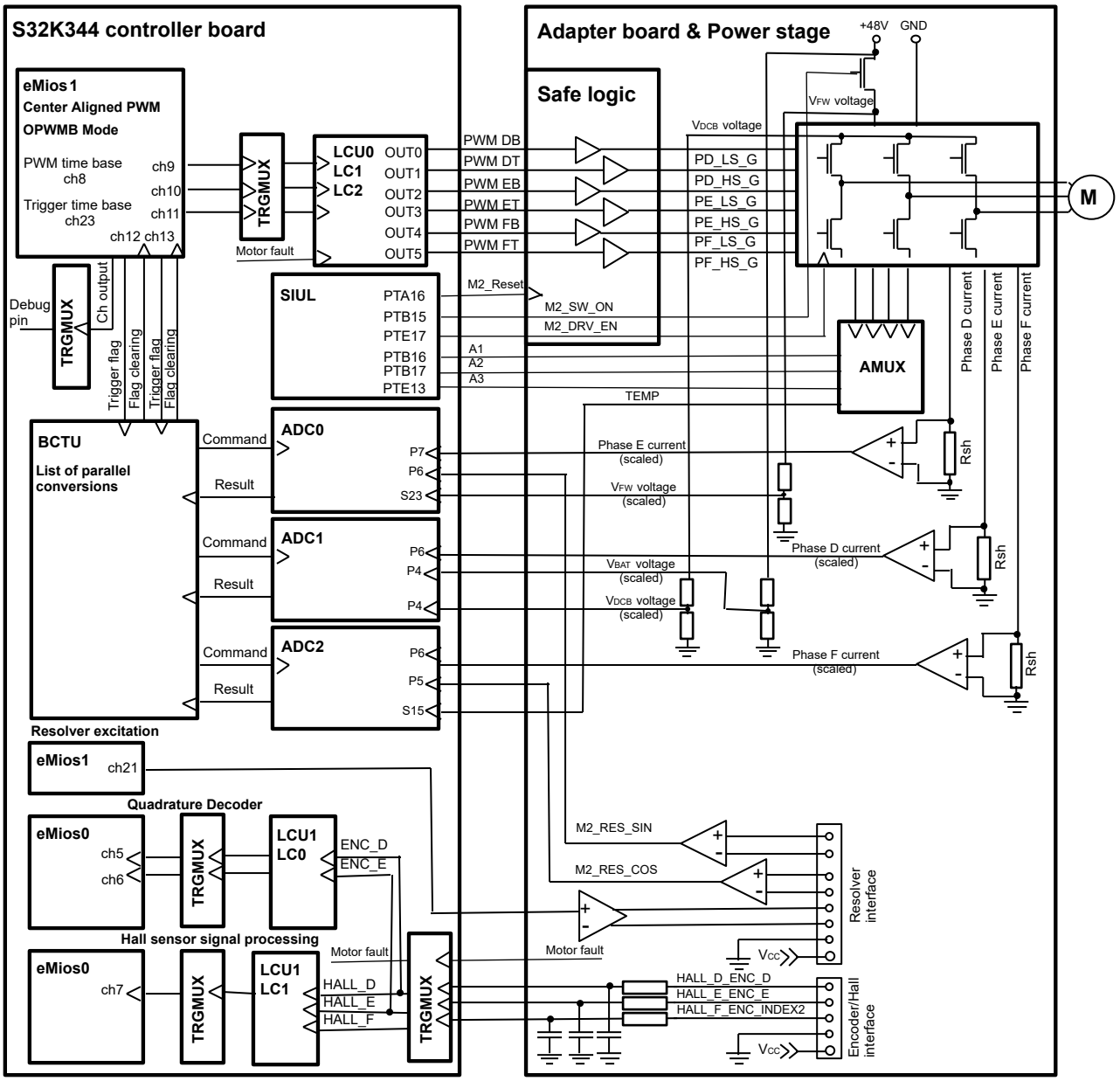


Figure 11. S32K344 module interconnection

4.1.2 Module involvement in digital PMSM FOC control loop

This section discusses timing and modules synchronization to accomplish PMSM FOC on the S32K344 and the internal hardware features. The time diagram of the automatic synchronization between PWM and ADC in the PMSM application is shown in Figure 12.

The PMSM FOC control with triple-shunt current measurement is based on static timing; meaning the trigger point of the ADC conversions is located at fixed place within every control loop cycle. This trigger point is also configurable during runtime.

eMIOS timer uses the concept of time bases for signal synchronization. There are 5 channels (CH0, CH8, CH16, CH22 and CH23) which can act as the time base what means that other channels can see a value of their counter through the bus. CH0, CH8, CH16 can create local time bases for 7 channels and CH22 and CH23

can create a global time bases for any channel. In the example CH8 creates the PWM time base for channels CH9, CH10 and CH11 which are responsible for PWM signal generation. The CH23 creates a TRIGGER time base for CH12 and CH13 which are responsible for triggering BCTU and resolver excitation time base for CH21. Both time bases operate in Modulus Counter Buffered (MCB) up counting mode, where period is set by register A. It is possible to start time bases synchronously by enabling eMIOS global prescaler. Offset between time bases is given by time base channel initial counter value. In this example, time bases are synchronous with no offset.

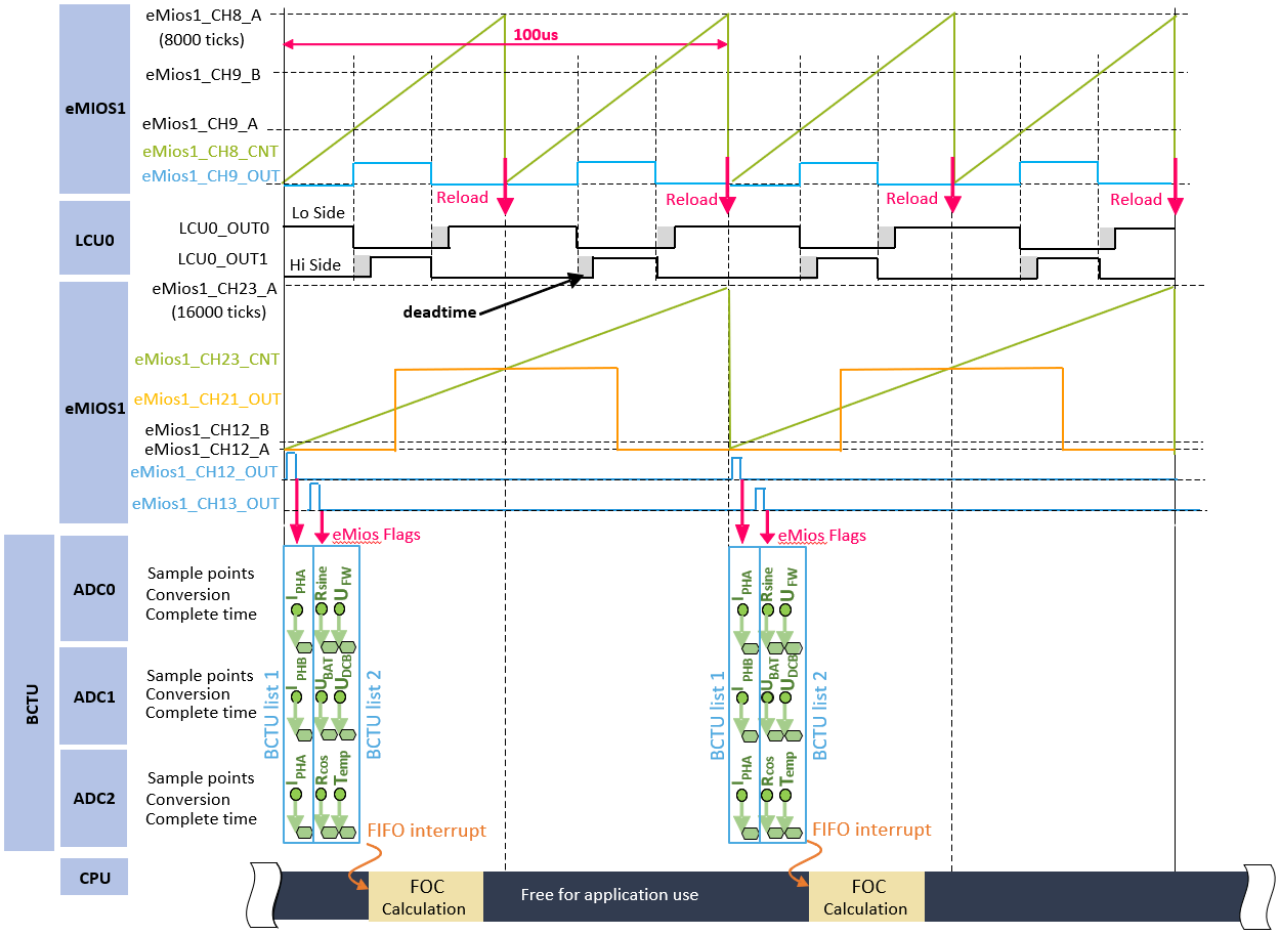


Figure 12. Time Diagram of PWM and ADC Synchronization

PWM frequency is 20kHz and ADC sampling frequency is 10kHz. PWM channels and trigger channels operates in OPWMB mode. PWM channel output signal is formed by comparing channel registers A and B with time base counter. For example, PWM signal for phase A is generated by output of the CH9. Center aligned PWM is achieved by proper setting of registers A and B. PWM A signal is routed to LCU where complementary signals for particular MOSFETs are created (LCU0 OUT0 and OUT1) respecting pre-driver input polarity and the dead time is inserted (see Figure 12).

Trigger signals CH12 and CH13 are formed in the same way as PWM signals. An important point here is that the connection between BCTU and CH12, CH13 is through the flag of CH12 or CH13 and not through their output. Flag can be generated on both compare events or on compare with register B only. In this example, the flags are set on register B only it means on falling edges of CH12 and CH13 output signals. CH12 and CH13 output signals can be routed using the TRGMUX to microcontroller pin for trigger debugging.

When flag of eMIOS1 CH12 is set, the BCTU starts list of conversion controlling ADC0, ADC1 and ADC2 and also clears back the CH12 flag. I_{PHD}, I_{PHE} and I_{PHF} stator currents are measured simultaneously shortly after

the beginning of PWM cycle, due to propagation delay in forward path of PWM. The beginning of PWM cycle is in the middle of non-active vector, where bottom MOSFETs of both inverter legs are turned on, and currents flow through shunt resistors. Based on SVM state, only two phase currents are taking into account (selection is according to the SVM state) and third is calculated based on Equation 9. See [15] for more information.

When flag of eMIOS1 CH13 is set, the BCTU starts list of conversions controlling ADC0, ADC1 and ADC2 and also clears back the CH13 flag. Sine and cosine signals of resolver and battery voltage are measured simultaneously, however measurement of this list is delayed after phase current measurement. This delay is set in order to respect resolver excitation signal, where envelope of sine and cosine signals is extracted. DC-bus voltage U_{DCB}, Forward voltage U_{FW} and temperature are measured shortly after measurement of sine and cosine signals of resolver and battery voltage. The ADC results are stored into BCTU FIFO result registers and interrupt is raised on watermark event. FOC control algorithm calculates new duty-cycle values based on measured quantities. Update of eMIOS1 CH9, CH10, CH11 is done simultaneously. Register A and B are double-buffered so change will be coherently propagated on channels time base reload.

4.2 S32K344 device initialization

To simplify and accelerate an application development, embedded part of the PMSM FOC motor control application has been created using S32 Design studio, RTD drivers (low level part) and S32K344 is configured using S32 configuration tools Figure 13.

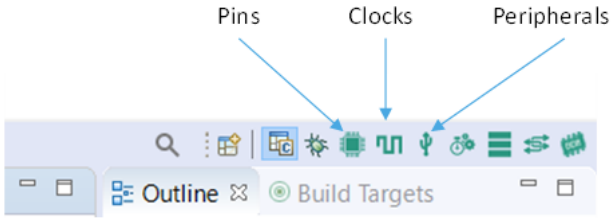


Figure 13. Config tools

Figure 14 describes the example project structure in the S32 Design Studio. Current settings of Config tools are stored in MCSXTM4CK344_PMSM_FOC_3Sh_II.mex file and generated files by config tools (all configuration structures) can be found in folders *board* and *generate*. When a component is added using the config tool, its sw driver is copied into folder RTD so only used drivers are part of the project. Peripherals are initialized at beginning of the main() function. For each S32K344 module, there is a specific initialization function, that uses configuration structures generated by Config tools to configure the MCU.

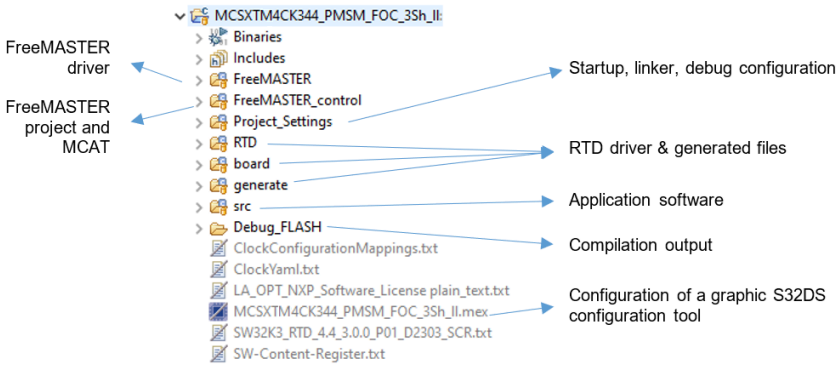


Figure 14. Example project structure

XXX_Init functions must be called before any other Application Programming Interface (API) from the module. It is important to initialize Clock and OsIf at first. OsIf initializes systic timer which can be used for timeout

measurements in other modules. The last function to call during the initialization process is Emios_Mcl_Ip_Init. It initializes time bases and enables their counters what initiate control cycle.

List of the initialization APIs:

- Clock_Ip_Init() - Initializes MCU clock configuration
- OsIf_Init() - Initializes the OS interface (basic timing/Os services for drivers)
- IntCtrl_Ip_Init() - Initializes the configured interrupts
- Siul2_Port_Ip_Init() - Initializes PINs and PORT configuration
- Lpuart_Uart_Ip_Init() - Initializes LPUART module configuration
- Trgmux_Ip_Init() - Initializes TRGMUX module configuration
- Adc_Sar_Ip_Init() - Initializes ADC modules configuration
- Lcu_Ip_Init() - Initializes LCU module configuration
- Emios_Mcl_Ip_Init() - Initializes eMios time-bases configuration
- Emios_Pwm_Ip_InitChannel() - Initializes emios PWM and Trigger channels configuration
- Emios_Icu_Ip_Init() - Initializes eMios input capture configuration
- Bctu_Ip_Init() - Initializes BCTU module configuration

RTD documentation can be found in the folder created in the S32 Design Studio installation path: “c:\NXP\S32 DS.3.5\S32DS\software\PlatformSDK_S32K3\RTD”.

Note: *The SW example of single 3-phase PMSM FOC is built on S32K3_RTD_4_0_0_HF01 version, and S32 Design Studio version 3.5.*

4.2.1 Port control & pin configuration

PMSM FOC sensorless/based motor control application requires following on chip pins assignment:

Table 1. Pins assignment for S32K344 PMSM Sensorless/based FOC control

Module	Signal name	Pin name / functionality	Description
LCU0	PWMD_HS	PTB14 / LCU0_OUT7	PWM signal for phase D high-side driver
	PWMD_LS	PTD4 / LCU0_OUT6	PWM signal for phase D low-side driver
	PWME_HS	PTB10 / LCU0_OUT9	PWM signal for phase E high-side driver
	PWME_LS	PTB11 / LCU0_OUT8	PWM signal for phase E low-side driver
	PWMF_HS	PTB8 / LCU0_OUT11	PWM signal for phase F high-side driver
	PWMF_LS	PTB9 / LCU0_OUT10	PWM signal for phase F low-side driver
ADC0	M2_V_FW	PTD21 / ADC0_S23	Forward voltage measurement
	PHE_I_BEMF_E	PTA9 / ADC0_P7	Phase E stator current measurement
	M2_RES_SIN	PTE11 / ADC0_P6	Resolver sine signal measurement
ADC1	PHD_I_BEMF_D	PTE6 / ADC1_P6	Phase D stator current measurement
	VBAT	PTD29 / ADC1_S23	Battery voltage
	M2_DCBV	PTA14 / ADC1_P4	DC bus voltage measurement
ADC2	V_TEMP	PTD30 / ADC2_S15	Temperature measurement
	M2_RES_COS	PTE26 / ADC2_P5	Resolver cosine signal measurement
	PHF_I_BEMF_F	PTE23 / ADC2_P6	Phase F stator current measurement
LPUART1	OPEN_SDA_RX	PTD13 / LPUART1_RX	UART transmit data (FreeMASTER)
	OPEN_SDA_TX	PTD14 / LPUART1_TX	UART receive data (FreeMASTER)

Table 1. Pins assignment for S32K344 PMSM Sensorless/based FOC control*...continued*

Module	Signal name	Pin name / functionality	Description
eMIOS_1	M2_RES_EXC	PTD24 / emios_1_ch_21_y	Resolver excitation signal
TRGMUX	TRG_OUT10	PTB19 / TRGMUX_OUT10	Pin for debugging microcontroller internal signals
	TRG_OUT11	PTB20 / TRGMUX_OUT11	Pin for debugging microcontroller internal signals
	HALL_D_ENC_D	PTD2 / TRGMUX_IN5	Phase D signal of the encoder/hall sensor
	HALL_E_ENC_E	PTD3 / TRGMUX_IN4	Phase E signal of the encoder/hall sensor
	HALL_F_ENC_Index2	PTC10 / TRGMUX_IN11	Phase F/Index signal of the encoder/hall sensor
SIUL2	MCU_LED1	PTE12 / GPIO	LED signalization
	MCU_LED2	PTE14 / GPIO	LED signalization
	MCU_LED3	PTD15 / GPIO	LED signalization
	MCU_RUN	PTE5 / GPIO	MCU control of external fans
	A1	PTB16 / GPIO	Control of analog mux for temp. measurement
	A2	PTB17 / GPIO	Control of analog mux for temp. measurement
	A3	PTE13 / GPIO	Control of analog mux for temp. measurement
	M2_SW_UP	PTD11 / GPIO	Application control via board button SW4
	M2_SW_RUN	PTD10 / GPIO	Application control via board button SW6
	M2_SW_DOWN	PTD12 / GPIO	Application control via board button SW5
	M2_SW_ON	PTB15 / GPIO	DC-link soft charge enable
	M2_DRV_EN	PTE24 / GPIO	Gate drivers enable
	M2_RESET	PTA16 / GPIO	Reset of latched faults
	DBG1	PTB18 / GPIO	Debug pin for SW execution evaluation

Pins tool and peripherals tool simplify configuration and particular RTD drivers offers an API to control the ports during the runtime.

4.2.1.1 SIUL2

System Integration Unit Lite2 (SIUL2) is a peripheral which provides control over all electrical pin controls and ports. It enables selection of the functions and electrical characteristics that appear on external chip pins.

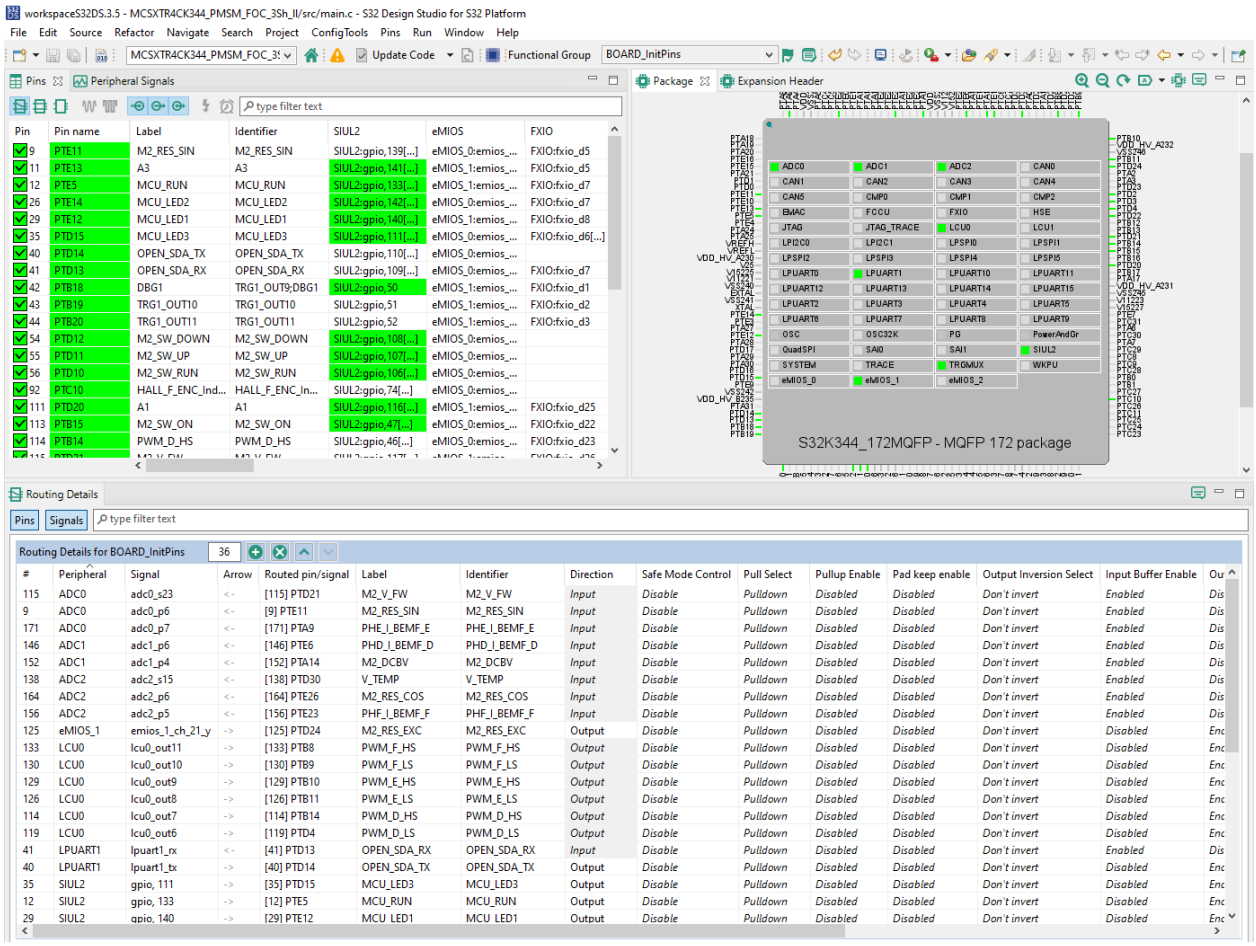


Figure 15. Pins

The pins assignment can be carried out by means of Pins tool. The pin assignment of the example is shown in Figure 15.

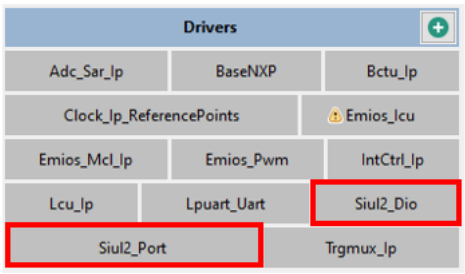


Figure 16. Pins SW drivers

Electrical characteristics as well as functionality are set in "*Routing Details*" section. Tool also offers visualization of the pinout placement in the selected package. In order to control SIUL2, following drivers are used and configured using Peripherals tool.

Siul2_Dio and Siul_Port drivers use configuration generated by pins tool. Siul_Port initializes all pins and Siul2_Dio is used to control GPIO functionality as is shown in Figure 17.

Figure 17. Example: Pin control API

```
void main (void)
{
...
Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS_PortContainer_0_BOARD_InitPeripherals,
g_pin_mux_InitConfigArr_PortContainer_0_BOARD_InitPeripherals);
...
}
void BctuFifoNotif  (void)
{
   /* Board buttons to control the application from board */
   cntrState.usrControl.btSpeedUp =  !Siul2_Dio_Ip_ReadPin(M2_SW_UP_PORT, M2_SW_UP_PIN);
   cntrState.usrControl.btSpeedDown =  !Siul2_Dio_Ip_ReadPin(M2_SW_DOWN_PORT, M2_SW_DOWN_PIN);
   cntrState.usrControl.btFlipFlop =  !Siul2_Dio_Ip_ReadPin(M2_SW_RUN_PORT, M2_SW_RUN_PIN);
...
   Siul2_Dio_Ip_SetPins(MCU_RUN_PORT, (1 << MCU_RUN_PIN));
...
   Siul2_Dio_Ip_ClearPins(MCU_RUN_PORT, (1 << MCU_RUN_PIN));
...
}
```

4.2.1.2 TRIGGER MUX

The TRGMUX peripheral provides an extremely flexible mechanism for interconnection various trigger sources to multiple pins/peripherals which is also very useful feature for debugging. This is configured using Trgmux_Ip driver. TRGMUX implements configurable connection between peripherals, which offers flexible triggering scheme in S32K3 device. This device has 16 pads (SIUL2) mapped to TRGMUX inputs and TRGMUX outputs, so internal signals can be visualized to output pin. In the example, pins PTB19 (TRGMUX OUT 10) and PTB20 (TRGMUX_OUT 11) are selected as pins for internal signal monitoring. Connection is created within TRGMUX hardware group. For example, hardware group TRGMUX_IP_SIUL_8_11 gathers TRGMUX SIUL outputs 8-11. The connection is made by selecting specific hardware output and input. PTB19 visualizes output of eMIOS1 CH12, which is a trigger signal for current measurement and PTB20 visualizes eMIOS1 CH13, which is a trigger signal for resolver sine and cosine measurement. Other signals like reload can be visualized by changing the "*Hardware Input*" configuration. Setting is applied by calling Trgmux_Ip_Init function.

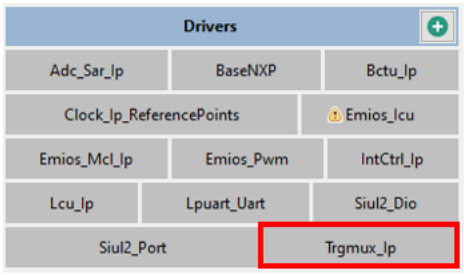


Figure 18. TRGMUX SW driver

Full list of all possible interconnections can be found in S32K3XX_TRGMUX_connectivity.xls attached to S32K3xx Reference Manual [7].

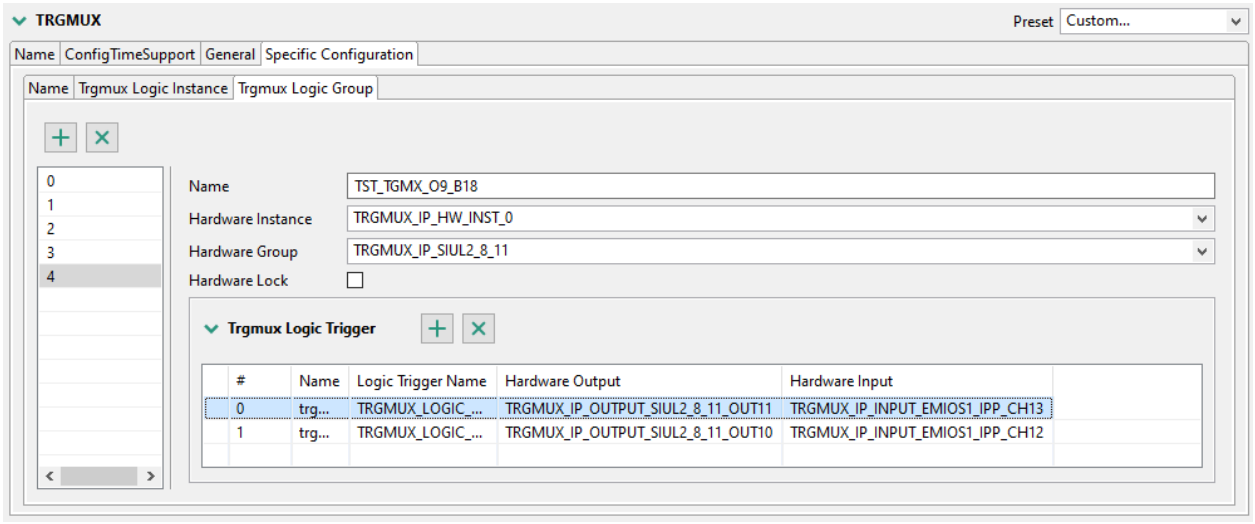


Figure 19. TRGMUX groups for debugging purposes

4.2.2 Clock & interrupt configuration

In order to configure S32K3 clocks and interrupts RTD offers Clocks Configuration tool companioned by Clock_Ip driver and Peripherals tool for OsIF and IntCrlt_Ip driver configuration. Configuration of OsIF is a part of BaseNXP driver.

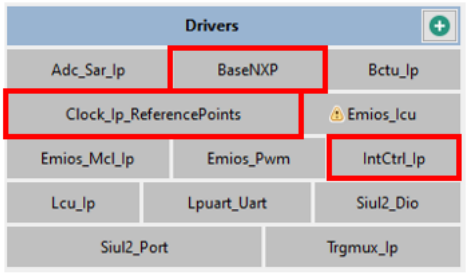


Figure 20. Clock, Os Interface and interrupts

4.2.2.1 Clocking

S32K344 features a complex clocking sourcing by Fast internal RC oscillator (FIRC), Slow internal RC oscillator (SIRC), Fast external crystal oscillator (FXOSC), Slow external crystal oscillator (SXOSC), Phase-locked loop (PLL), Clock Generation Module (MC_CGM), Mode Entry module's (MC_ME).

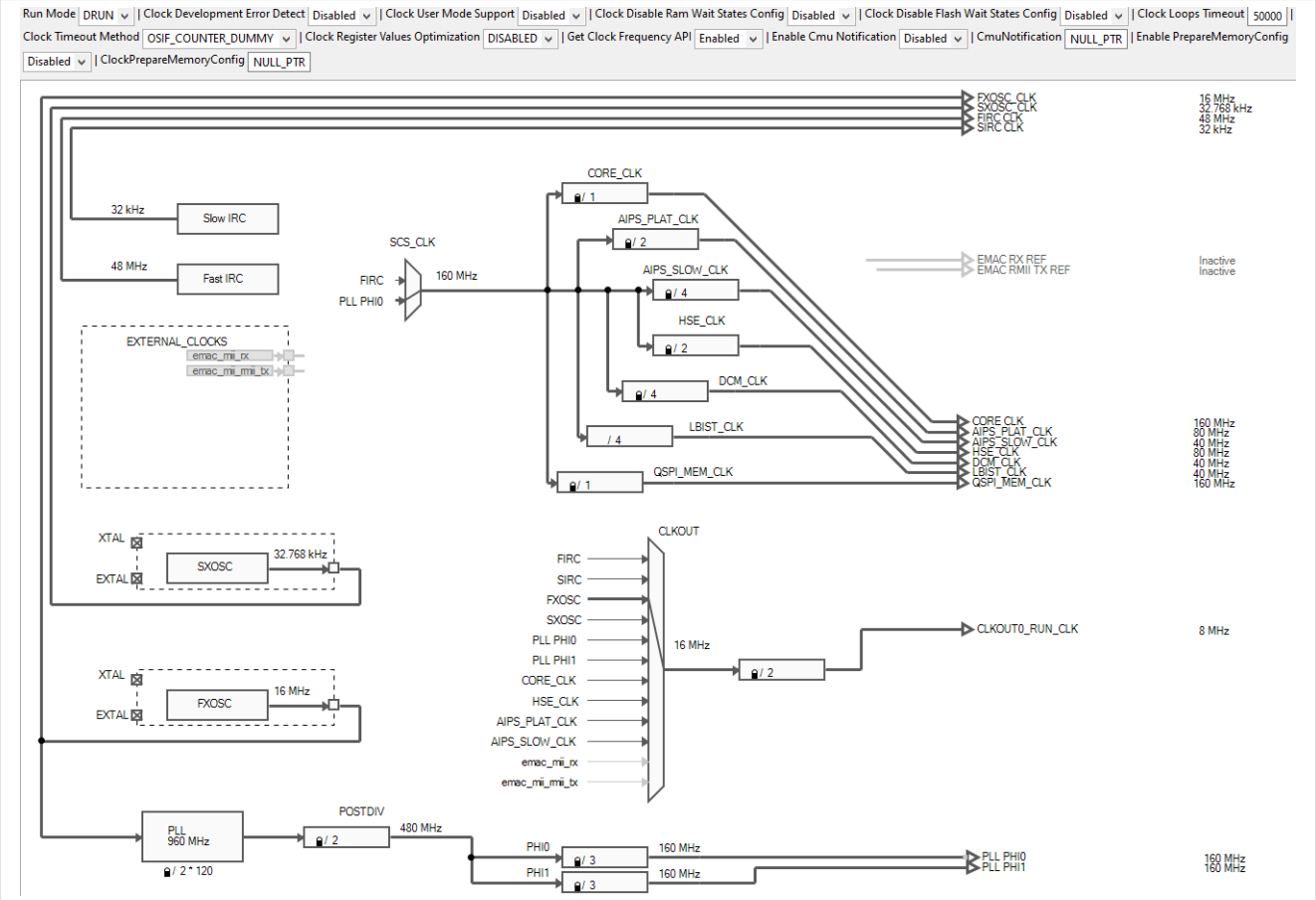


Figure 21. Clocks tool

To run the core of the S32K344 at maximum frequency 160MHz, S32K344 is supplied externally by 16 MHz crystal. This clock source supplies Phase-lock-loop (PLL), and its output is adjusted to 160 MHz frequency. PLL output PHI0 is then used to supply the core CORE_CLK. All real-time control peripherals are supplied by CORE_CLK, what eliminates unwanted wait states on the bus when peripherals are controlled by core during runtime. This clock configuration can be setup by S32 Clock Configuration tool which offers visual graphical user interface (GUI) to change the settings. Clock settings are applied by calling Clock_Ip_Init() function, where generated configuration by Clocks tool is an argument. Additional configuration of clock is also available in Clock_Ip_ReferencePoints driver.

Clock setting is summarized in:

Table 2. S32K344 clock configuration

Clock	Frequency	MCU Peripheral
CORE_CLK	160 MHz	BCTU, LCU0-1, eMIOS0-2
CORE_CLK/2	80 MHz	ADC0-2
AIPS_SLOW_CLK	40 MHz	LPUART1, TRGMUX

Operating System Interface (OsIF) configuration is available in BaseNXP driver, which provides basic timing/OS services for drivers, allowing for OS independent implementations. This example is bare-metal software without operating system, but other drivers can use OSIF for timeouts detection. OsIF settings are applied by calling OsIf_Init() function.

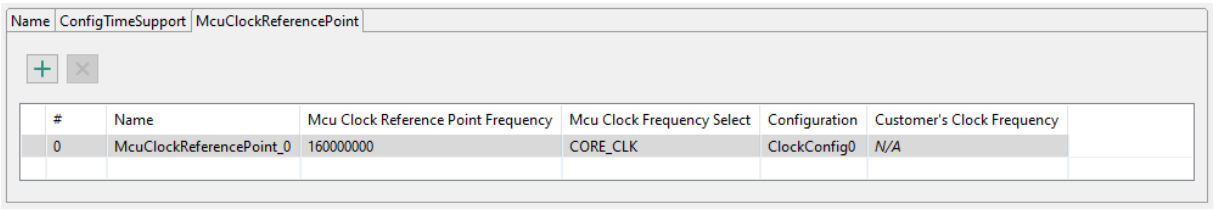


Figure 22. Clock configuration

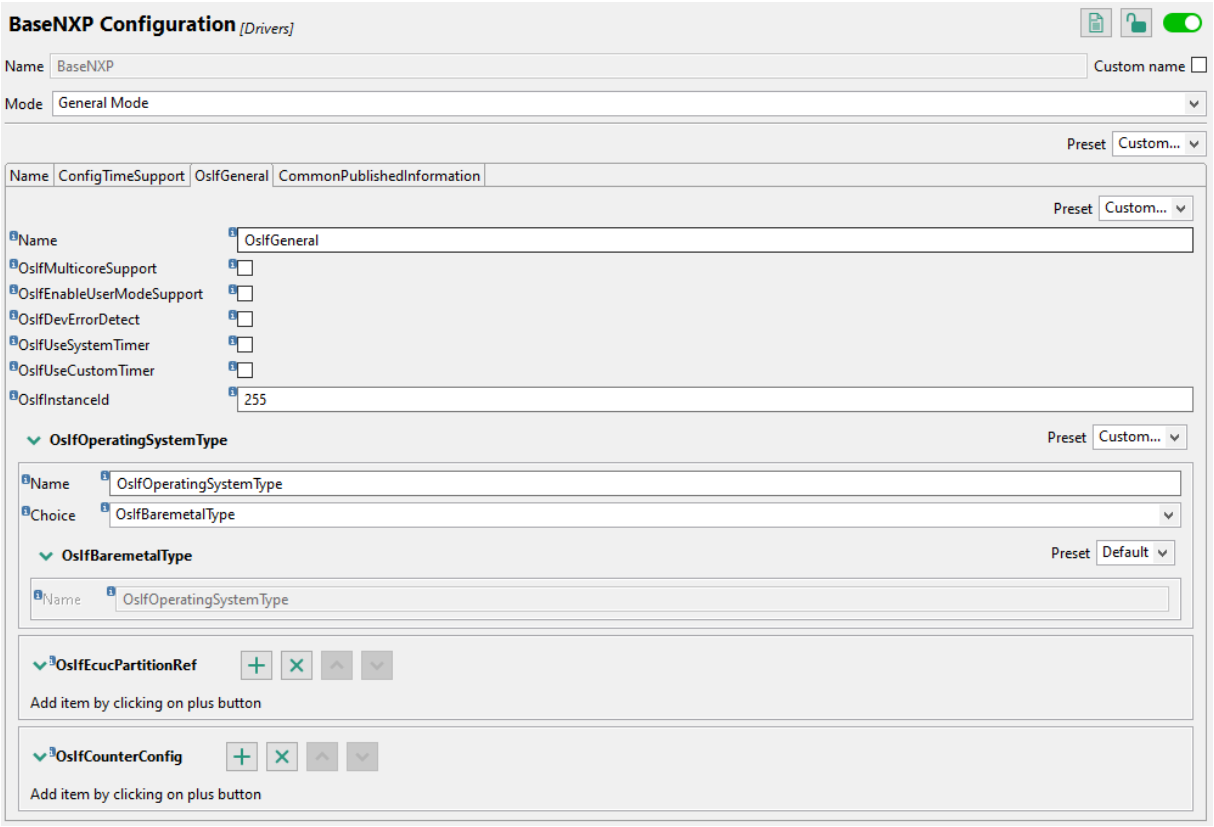


Figure 23. BaseNXP configuration

4.2.2.2 Interrupts

IntCrtl_Ip driver is responsible for an interrupt configuration on S32K3 platform. Settings impact Miscellaneous System Control Module (MSCM), Nested vectored interrupt controller (NVIC), and interrupt vector table. In the example, two interrupts are used: Interrupt generated by eMIOS0 CH4 and Interrupt from BCTU. There are 3 options to set in “Interrupt controller” column “Handler”. If new interrupt routine is added, undefined handler is set by default. Another option of handler configuration is managed by user, where own custom handler can be set (but this interrupt service routine must be defined and implemented in custom code) or interrupt service routine from RTD driver. Naming of RTD interrupt service routines can be found in integration manual of particular RTD driver. Bctu_0_Isr and EMIOS0_4_IRQ handle their interrupts and call notification functions on specific event defined by Emios_Icu and Bctu_Ip component settings in peripheral tool (eMIOS0IcuNotify, BctuFifoNotif).

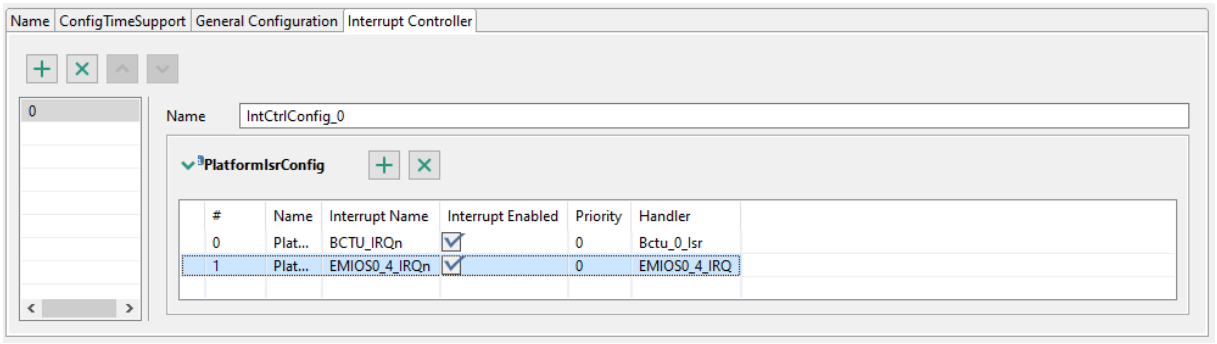


Figure 24. Interrupt controller

Interrupt setting and handlers installation to vector table are realized by calling IntCtrl_Ip_Init()

Figure 25. Example: Interrupt controller API

```
void main (void)
{
...
   /*************************************************************************************************
    *Configure and enable interrupts
    *************************************************************************************************/
    IntCtrl_Ip_Init(&IntCtrlConfig_0);
...
}
```

4.2.3 Center-aligned PWM

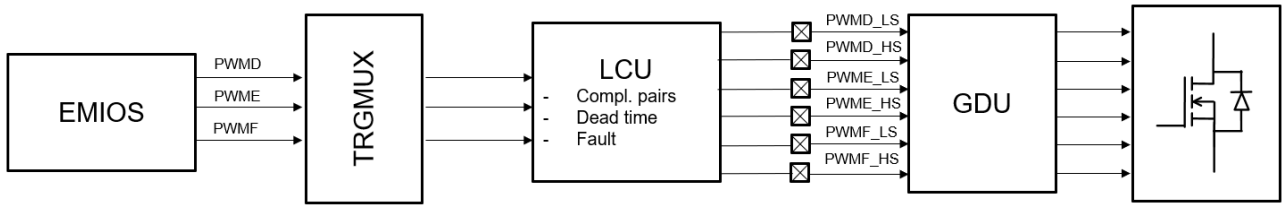


Figure 26. Block diagram of key modules for PWM generation

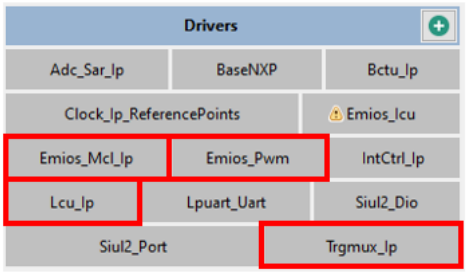


Figure 27. Drivers for PWM generation

Generation of the center aligned PWM functionality is realized by modules eMIOS, TRGMUX and LCU. In order to configure and control those peripherals following RTD drivers are used: Emios_Mcl_Ip to configure eMIOS timebase, Emios_Pwm to configure and control eMIOS PWM channels, Lcu_Ip to configure and control LCU and Trgmux_Ip to interconnect eMIOS and LCU.

4.2.3.1 eMIOS

eMIOS1 CH8 is configured as a time base for PWM signals. This channel can create local time base for CH 9-15. Chanel operates in a Modulus Counter Buffered (MCB) mode with up counting mode only. When the internal counter matches a value defined by field period (channel register A of the eMIOS channel) and a clock tick occurs, the internal counter is reset to 1 and reload is generated. Considering 160MHz and Clock Divider Value 1 and Master Bus Prescaler DIV_1, the "*Default period*" 8000 ticks means 50µs/20kHz. "*Offset at start*" gives the opportunity to initialize counter value before the counting is started what allows to configure delay between multiple synchronized time bases.

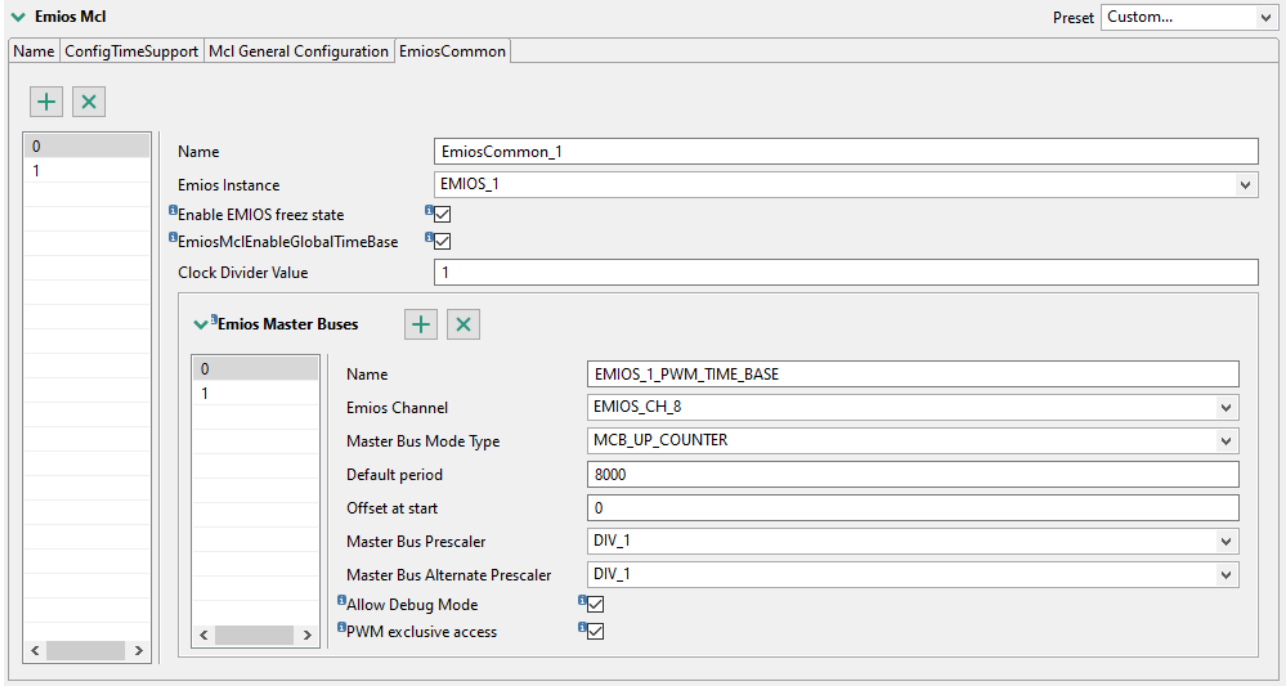


Figure 28. PWM time base configuration

eMIOS1 CH 9-11 are configured to generate the PWM signals for motor phases PHD-F. Channels operates in Output PWM Buffered (OPWMB) mode. This is the most flexible eMIOS PWM mode, which offers independent setting of both PWM signal edges (by channels register A and B) and can form the most common types of PWM signal. Channels select local time base BCDE as a counter bus and time base settings are also referenced through "*PwmEmiosBusRef*" field. Channel is able to see time base counter value through the BCDE bus and compare it with its registers A and B. "*Polarity*" defines output state on specific compare. Complete timing diagram can be found in Figure 12. Driver offers an abstraction where "*duty cycle*" is an active pulse (space between compare A and B) and "*Phase shift*" defines placement of this active pulse within the PWM period. Proper values for register A and B are calculated during runtime by special API, Emios_Pwm_Ip_UpdateUCRegA(). The phase shift value for each phase is one of the API parameters and is calculated based on demanded duty cycle and half of the PWM period. Init values of the *"Phase shift"* and *"duty cycle"* are set in Peripherals tool. Settings are applied by calling Emios_Pwm_Ip_InitChannel(), Emios_Mcl_Ip_Init() and Emios_Mcl_Ip_ConfigureGlobalTimebase(). After calling Emios_Mcl_Ip_Configure GlobalTimebase() time base counting is started. PWM signal is modified during the runtime by disabling PWM update, updating the duty cycle and the phase shift and enabling the update. Registers A and B are double buffered in OPWMB mode, so new values of registers A and B are propagated on nearest reload generated by time base.

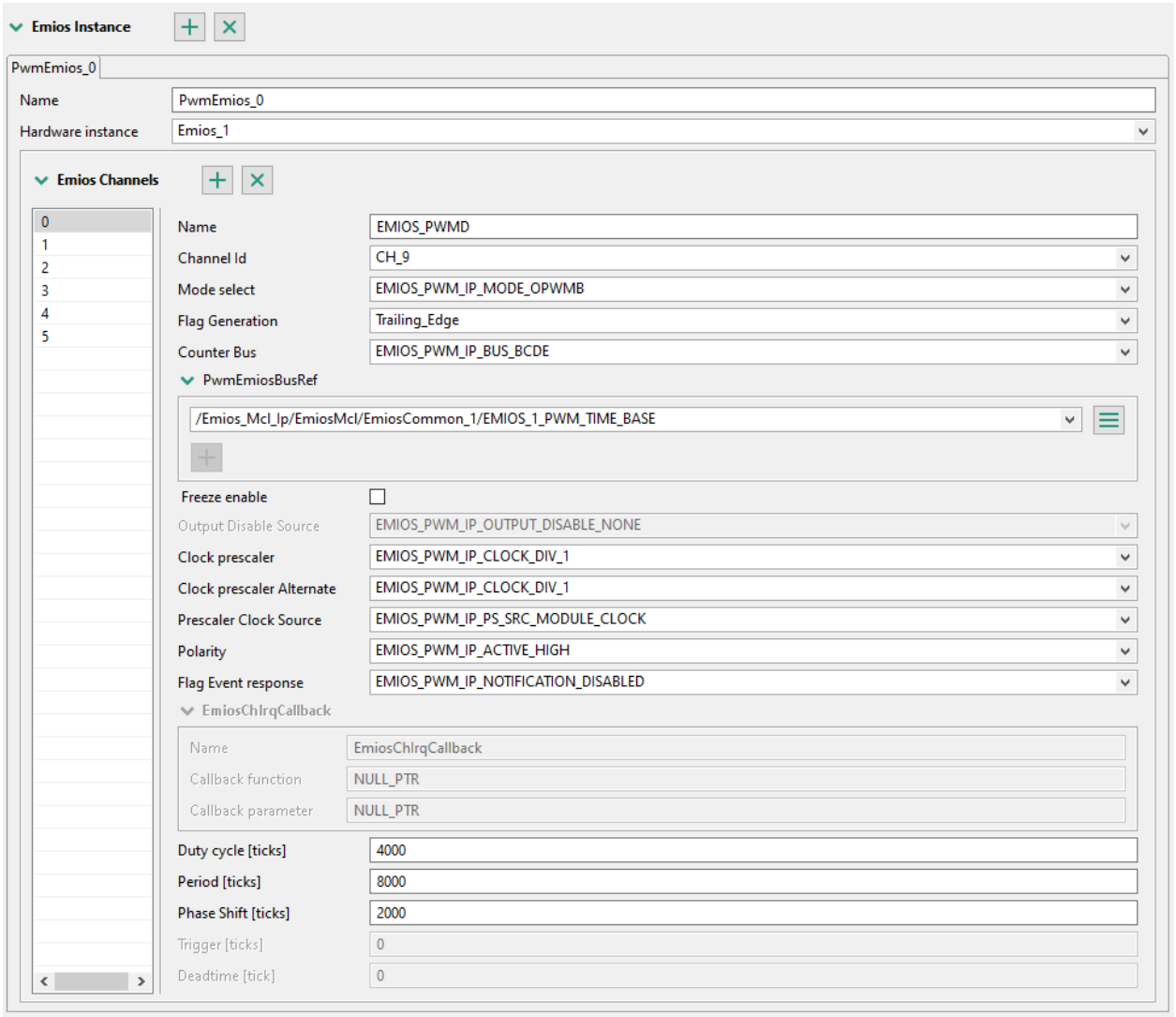


Figure 29. PWM channel configuration

Figure 30. Example: eMIOS API for PWM

```
void main (void)
{
...
   /*******************************************************************************************************
    * eMios Driver
    *******************************************************************************************************/
   Emios_Mcl_Ip_Init(1U, &Emios_Mcl_Ip_1_Config_BOARD_INITPERIPHERALS);
   Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch9);
   Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch10);
   Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch11);
...
   /*Enable eMIOS clock at last to ensure the correct trigger order*/
   Emios_Mcl_Ip_ConfigureGlobalTimebase(1U, TRUE);
...
}
tBool ACTUATE_SetDutycycle(SWLIBS_3Syst_FLT *fltpwm)
{
...
   Emios_Mcl_Ip_ComparatorTransferDisable(1U,(uint32_t)0xE00U);
```

```
...
    Emios_Pwm_Ip_UpdateUCRegA(1U, 9U, pwmShiftD_A);
    Emios_Pwm_Ip_UpdateUCRegB(1U, 9U, pwmShiftD_B);
    Emios_Pwm_Ip_UpdateUCRegA(1U, 10U, pwmShiftE_A);
    Emios_Pwm_Ip_UpdateUCRegB(1U, 10U, pwmShiftE_B);
    Emios_Pwm_Ip_UpdateUCRegA(1U, 11U, pwmShiftF_A);
    Emios_Pwm_Ip_UpdateUCRegB(1U, 11U, pwmShiftF_B);
    Emios_Mcl_Ip_ComparatorTransferEnable(1U,(uint32_t)0xE00U);
}
```

4.2.3.2 Trigger MUX

TRGMUX ensures a connection between eMIOS and LCU. Settings within the "*Hardware group*" TRGMUX_IP_LCU0_0 and TRGMUX_IP_LCU0_1 connects outputs of eMIOS1 channels 9-11 to LCU0 inputs 3-5. Setting is applied by calling Trgmux_Ip_Init() function.

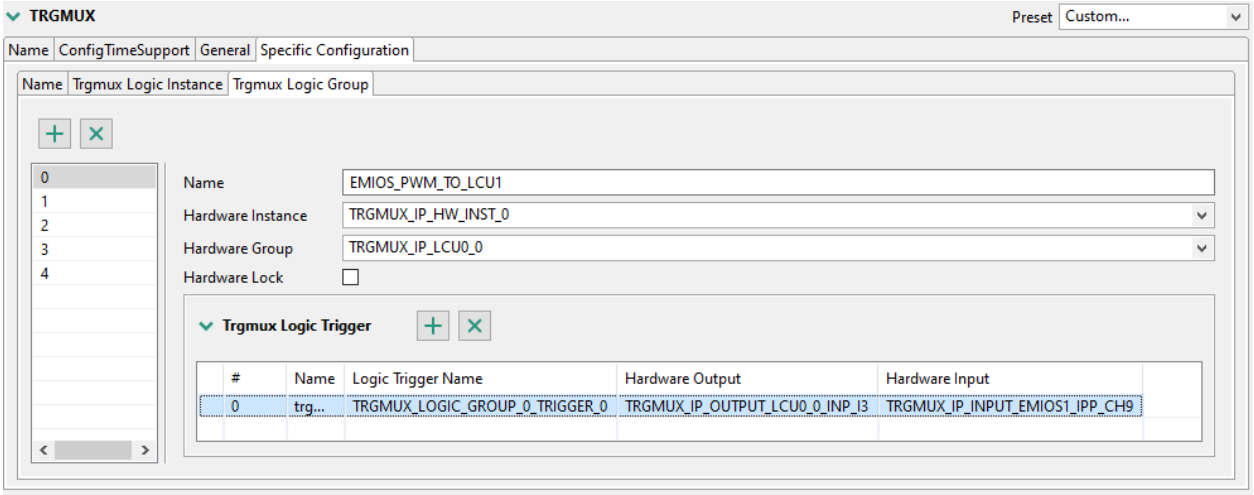


Figure 31. TRGMUX settings for PWM signal of phase D

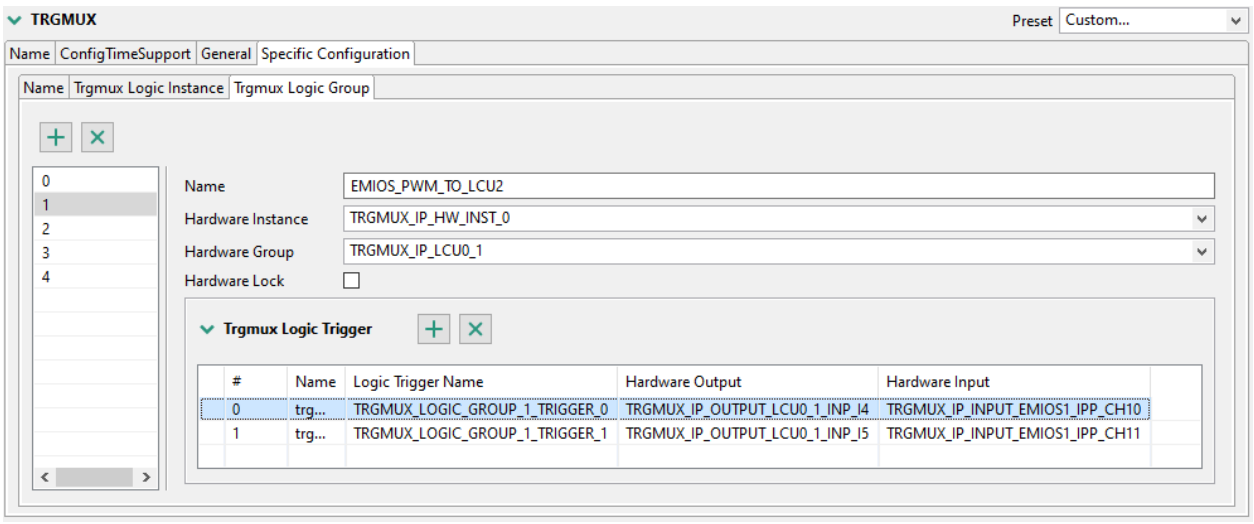


Figure 32. TRGMUX settings for PWM signals of phases E and F

4.2.3.3 LCU

Logic control unit (LCU) is a peripheral for a real time control, which offers a programmable logic function to create output waveforms or to process digital signals. LCU contains 3 Logic cells (LC) embedded each with 4 inputs and outputs with configurable Look Up Table for each output and more other features like digital filters, force inputs, sync inputs, SW override logic. In order to generate the PWM complementary signal, the following functionality is needed: Input multiplexing, Look Up Table (LUT), Digital filters, output polarity settings as is shown in Figure 33. Full featured LCU diagram can be found in S32K3xx Reference Manual [7].

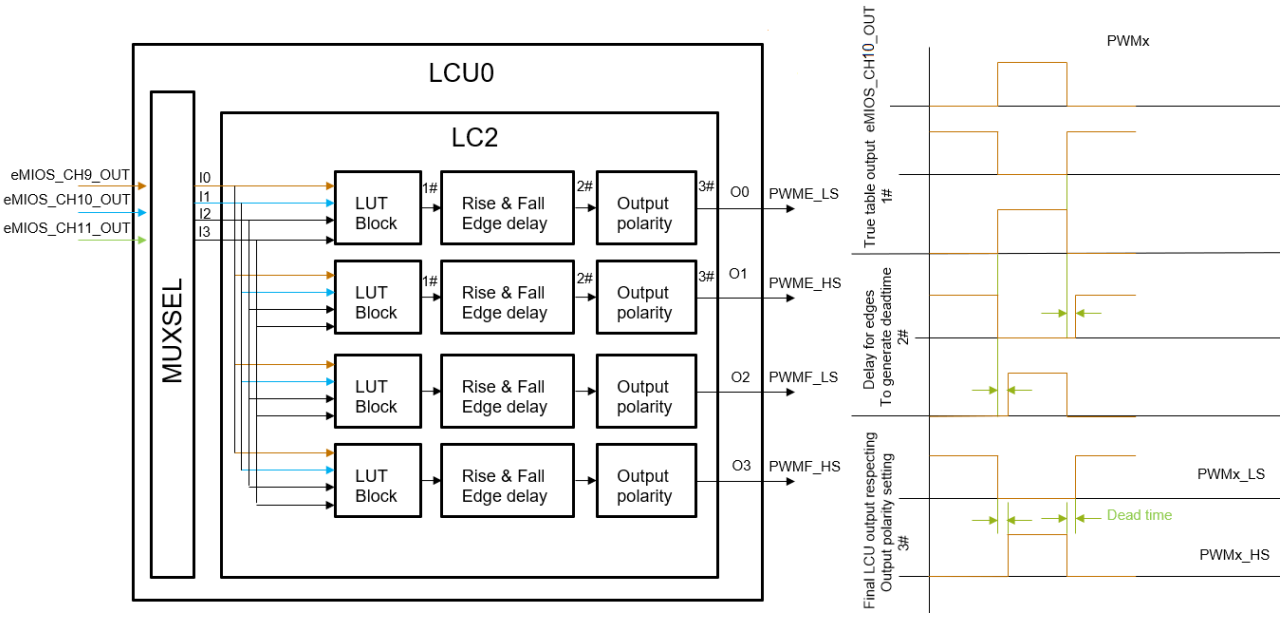


Figure 33. Simplified LCU features block diagram for PWM signals

Lcu_Ip driver is used to configure and to control LCU. In this example LCU0 instance is selected to generate PWM complementary pairs. LC1 generate signals for phases D and LC2 generates signals for Phase E and F. First configuration relates to inputs multiplexing. Configurations 0-2 in *"Lcu Logic Input"* section create a connection between LCU instance inputs and LC inputs. Multiplexor input 3 (eMIOS1 CH 9) is connected to LC1 input 1 and multiplexor inputs 4, 5 (eMIOS1 CH10, 11) are connected to LC2 inputs 0 and 1. Output configuration for complementary pairs is in section *"Lcu Logic Output"* configurations 0-5. The first important thing to configure is an output polarity. NXP 48V development platform uses simple gate drivers in half bridge configuration, so change of output polarity for high side or low side is not needed. Next setting is Look-up Table (LUT) for every output. LUT defines output state of the LUT Block for every combination of four inputs (combination 0000 is least significant bit of the LUT register). For example, I0 is negated by LCU to O0 and I0 is mirrored by LCU to O1 (as complementary channel) as is shown in Table 3. Last thing to configure is a dead time. It is generated using digital filters where rising edges of the LUT block output are delayed. Complete waveform composition of complementary channels can be seen in the Figure 33. In case of simpler drivers or external fault logic, LCU offers asynchronous Force logic which can automatically disable LCU outputs on external pin event. This functionality is available in NXP 48V development platform, however it is not implemented in SW example. For more details about this feature see S32K3xx Reference Manual [7]. In order to configure and control LCU, a Lcu_Ip RTD driver is used. Settings are applied by calling Lcu_Ip_Init() function and outputs can be enabled/disabled by calling Lcu_Ip_SetSyncOutputEnable().

Table 3. LUT configurations for LCU0 LC2

LC2_I3	LC2_I2	LC2_I1	LC2_I0		LC2_O0	LC2_O1	LC2_O2	LC2_O3
x	x	PWM_PHF	PWM_PHE		PWME_LS	PWME_HS	PWMF_LS	PWMF_HS
0	0	0	0		1	0	1	0

Table 3. LUT configurations for LCU0 LC2...*continued*

LC2_I3	LC2_I2	LC2_I1	LC2_I0		LC2_O0	LC2_O1	LC2_O2	LC2_O3
x	x	PWM_PHF	PWM_PHE		PWME_LS	PWME_HS	PWMF_LS	PWMF_HS
0	0	0	1		0	1	1	0
0	0	1	0		1	0	0	1
0	0	1	1		0	1	0	1
0	1	0	0		1	0	1	0
0	1	0	1		0	1	1	0
0	1	1	0		1	0	0	1
0	1	1	1		0	1	0	1
1	0	0	0		1	0	1	0
1	0	0	1		0	1	1	0
1	0	1	0		1	0	0	1
1	0	1	1		0	1	0	1
1	1	0	0		1	0	1	0
1	1	0	1		0	1	1	0
1	1	1	0		1	0	0	1
1	1	1	1		0	1	0	1
LUT					**0x5555**	**0xAAAA**	**0x3333**	**0xCCCC**

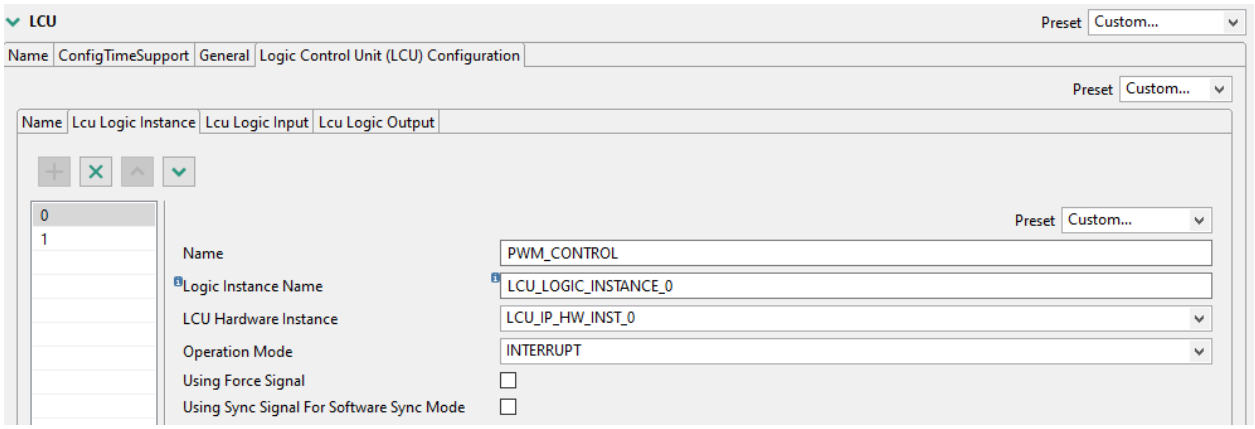


Figure 34. LCU instance configuration for PWM

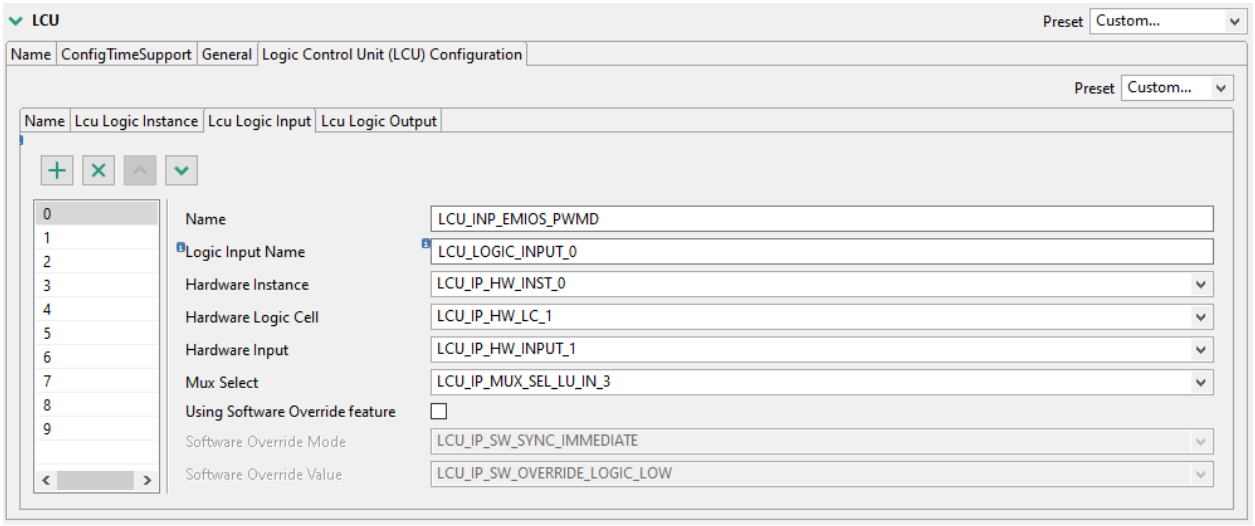


Figure 35. LC inputs configuration for PWM

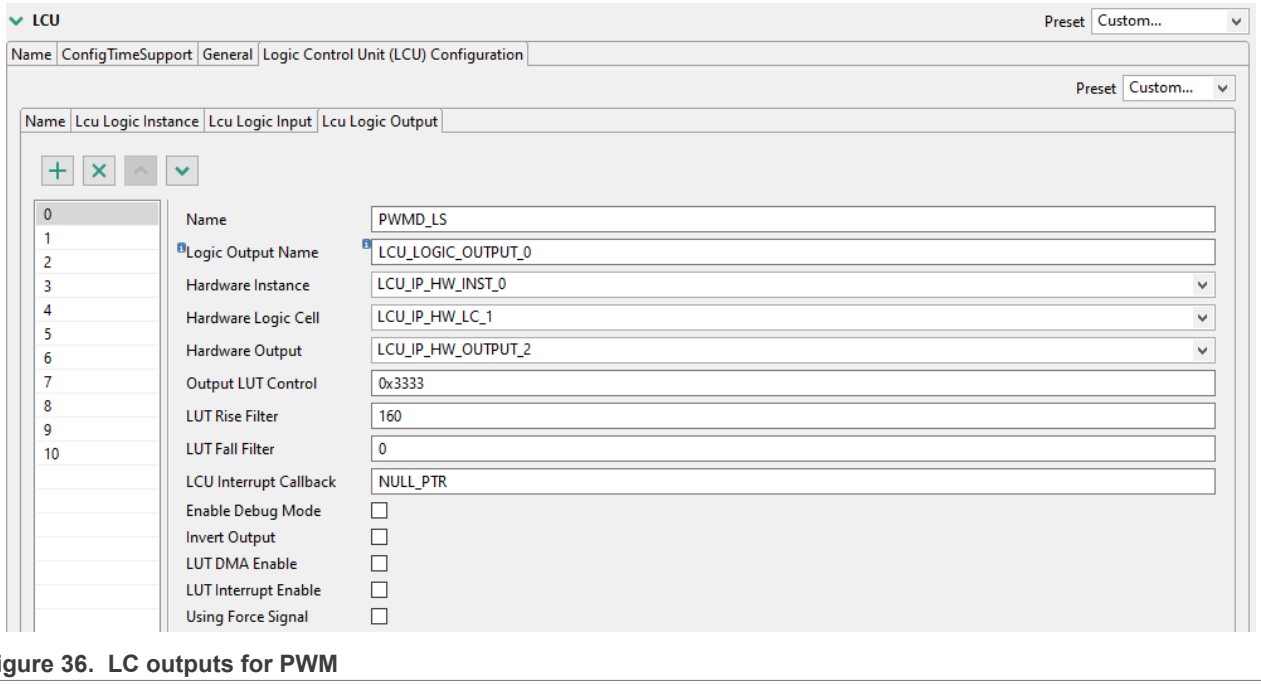


Figure 36. LC outputs for PWM

4.2.4 Analog data capturing

Motor control analog feedback capturing is realized by ADC0, ADC1, ADC2, BCTU and eMIOS peripherals. BCTU controls parallel conversion of ADC0, ADC1 and ADC2.

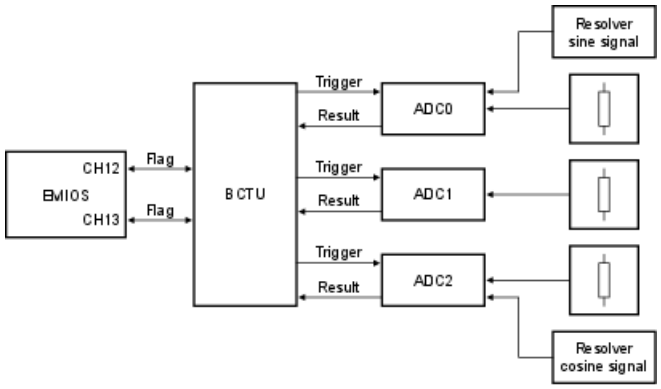


Figure 37. MC analog feedback capturing

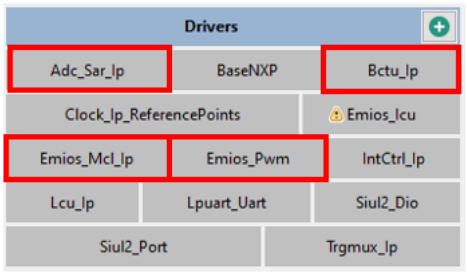


Figure 38. Drivers for analog feedback capturing

eMIOS defines the trigger point when the conversion should start. In order to configure and to control those peripherals, following RTD software drivers are used: Adc_Sar_Ip, Bctu_Ip, Emios_Mcl_Ip, Emios_Pwm.

4.2.4.1 ADC

The S32K344 device has three Analog-to-Digital Converters (ADCs) with the SAR algorithm. The ADC channels are divided into 3 groups - Precision, Standard and External (each allows independent configuration settings and different accuracy/performance level). Each channel has selectable resolution (8-, 10-, 12-, 14-bit). Conversion can be started by Normal conversion trigger, Injected conversion trigger or BCTU conversion trigger. There is also special mode, BCTU control mode, where it is explicitly set that only the BCTU can start a conversion of ADC instance. All other trigger sources are ignored. This mode is used for MC measurement and utilizes all ADC instances. The most important setting can be seen in the Peripherals tools settings. Settings are applied calling Adc_Sar_Ip_Init() function and after the configuration ADCs are calibrated by Adc_Sar_Ip_DoCalibration().

Figure 39. Example: ADC API

```
void main (void)
{
...
   /*****************************************************************************************************
    * ADC Driver
    *****************************************************************************************************/
   do {
        status = (StatusType)Adc_Sar_Ip_Init(0U, &AdcHwUnit_0);
   } while (status != E_OK);
   do {
        status = (StatusType)Adc_Sar_Ip_Init(1U, &AdcHwUnit_1);
   } while (status != E_OK);
   do {
```

```
        status = (StatusType)Adc_Sar_Ip_Init(2U, &AdcHwUnit_2);
    } while (status != E_OK);
    do {
        status = (StatusType)Adc_Sar_Ip_DoCalibration(0U);
    } while (status != E_OK);
    do {
        status = (StatusType)Adc_Sar_Ip_DoCalibration(1U);
    } while (status != E_OK);
    do {
         status = (StatusType)Adc_Sar_Ip_DoCalibration(2U);
     } while (status != E_OK);...
}
```

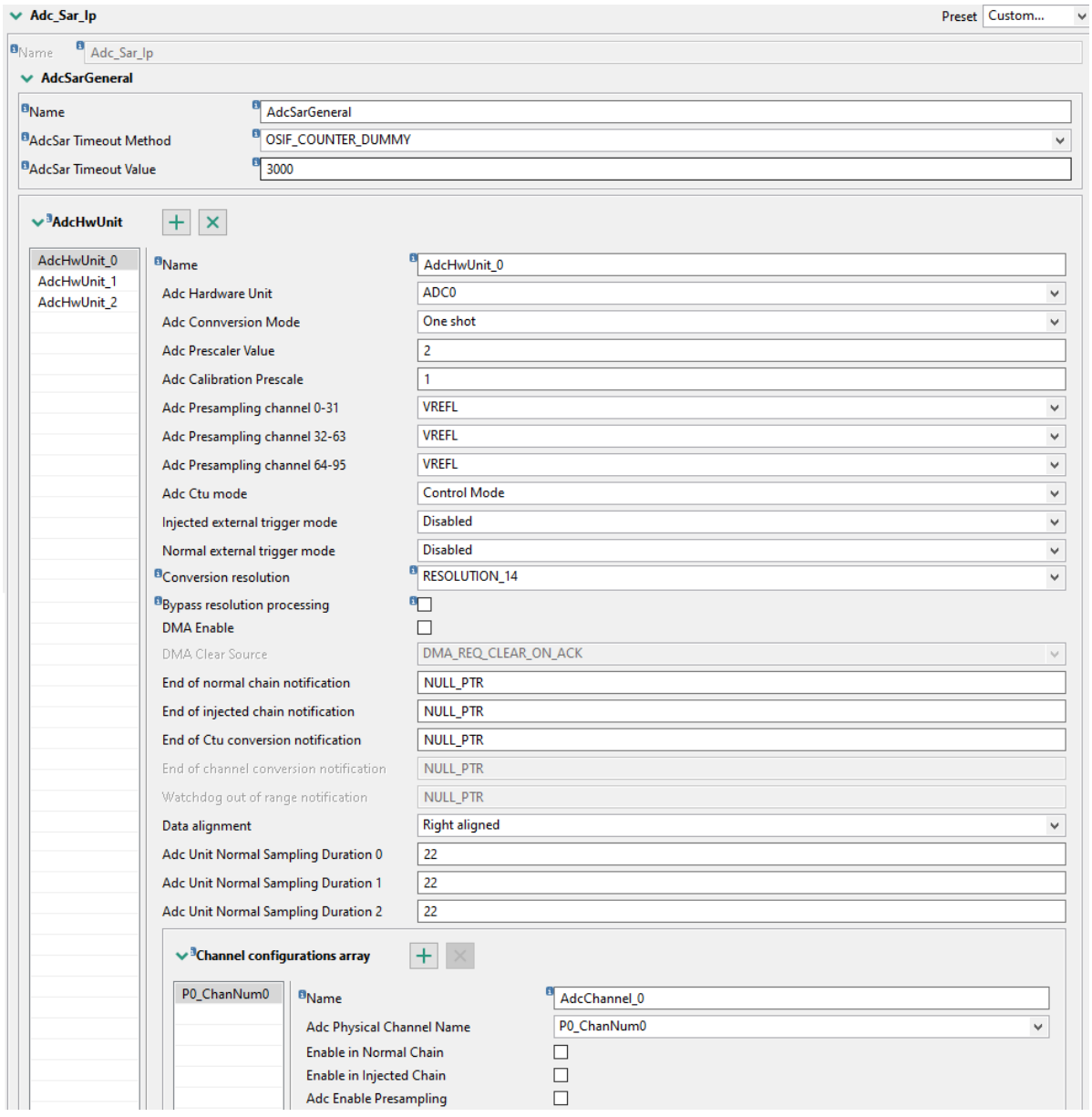


Figure 40. ADC configuration for MC measurements

4.2.4.2 BCTU

S32K344 has single instance of a BCTU. The BCTU accepts ADC conversion-request trigger inputs and executes ADC conversions configured for this trigger. There are 72 trigger inputs. 69 inputs are coming from eMIOS channels (connection is realized through channels flag) and 3 from TRGMUX (TRGMUX output is routed to BCTU). All triggers can be also invoked by a software instead of the HW source. Every trigger can be configured to invoke single conversion or predefined list of conversions. Conversion result can be stored into BCTU data register (there is one register per ADC instance), one of the BCTU FIFOs or into a memory buffer by DMA transfer. Conversion results remain also in the result register of ADC channel.

In this example eMIOS1 CH12 is selected as a trigger for phase current measurement and eMIOS1 CH13 is selected as a trigger for resolver sine and cosine signals measurement. FIFO1 is selected for "*Data Destination*". The trigger is configured as a list of parallel conversions ADC0, ADC1 and ADC2 in "*Adc Target Mask*". List of ADC channels is defined in "*BCTU List Items*" while order is given by the "*Adc Target Mask*": BctuListItems_0 is ADC0, BctuListItems_1 is ADC1 etc. Watermark of the FIFO1 "*Watermark Value*" is set on 8 and "*Interrupt Notification*" is enabled. When first trigger comes (eMIOS1 CH12), parallel conversion of the first list of three items starts (phase currents). Conversion of following three items (the second list) is waiting until second trigger (eMIOS1 CH13) comes. Once this following conversion has been completed, next channels as a part of second list take a place (DC bus voltage, Forward voltage and temperature). Once all results have been stored into FIFO1, an interrupt is raised and handled by BCTU RTD interrupt handler and custom notification function BctuFifoNotif is called. Conversion result (Data and additional information about conversion like trigger number, ADC channel and ADC instance) are read from the FIFO using Bctu_Ip_GetFifoResult() function. Settings are applied by calling Bctu_Ip_Init() function. After enabling the notification function and BCTU global trigger, BCTU is active.

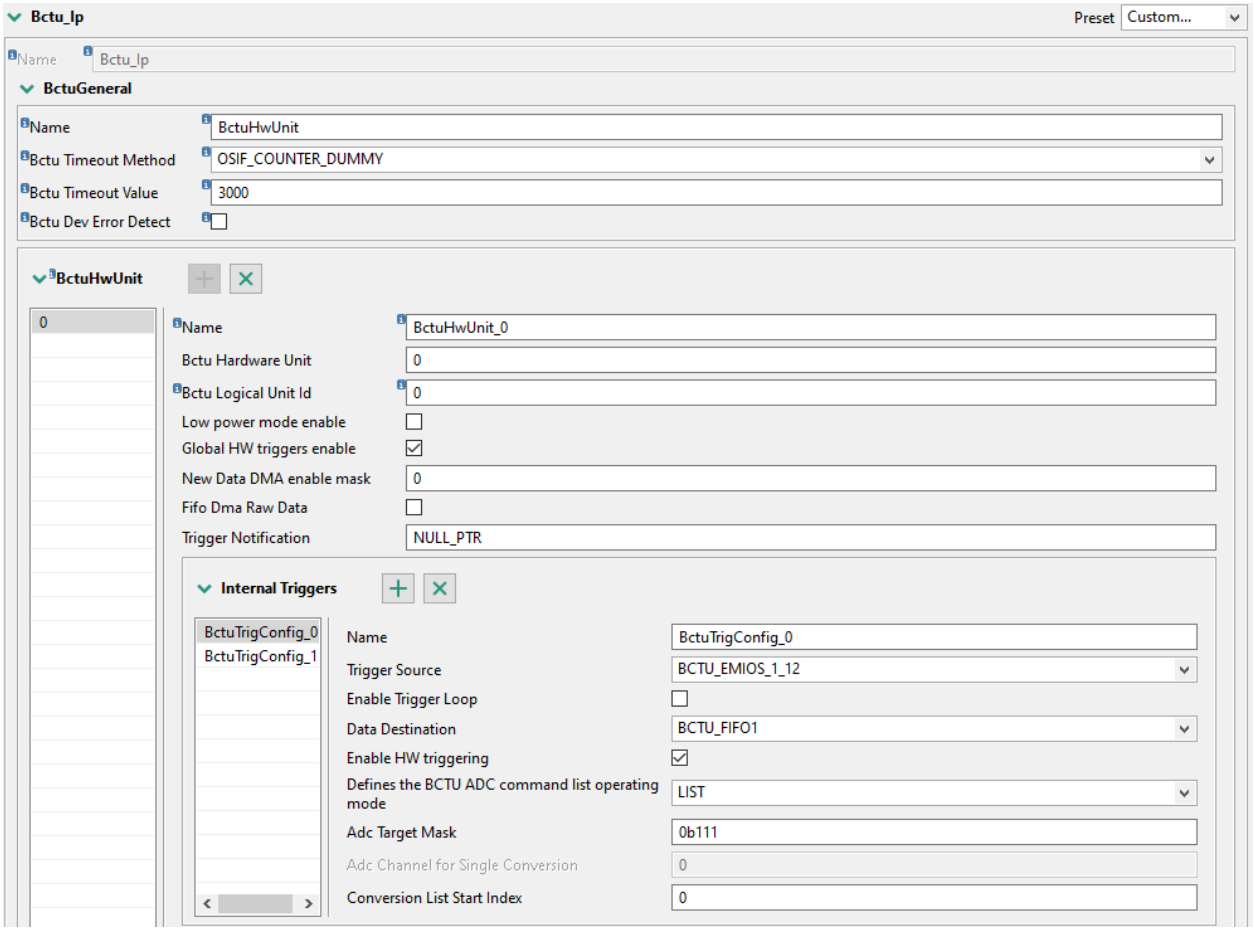


Figure 41. BCTU Trigger configuration

Table 4. Possible variations of ADC target mask

ADC target mask			Defines the BCTU ADC command list operating mode	
			LIST	SINGLE
0	0	1	of single conversions ADC0	Conversion ADC0
0	1	0	of single conversions ADC1	Conversion ADC1
1	0	0	of single conversions ADC2	Conversion ADC2
0	1	1	of parallel conversions ADC0, ADC1	X
1	1	0	of parallel conversions ADC1, ADC2	X
1	0	1	of parallel conversions ADC0, ADC2	X
1	1	1	of parallel conversions ADC0, ADC1, ADC2	X

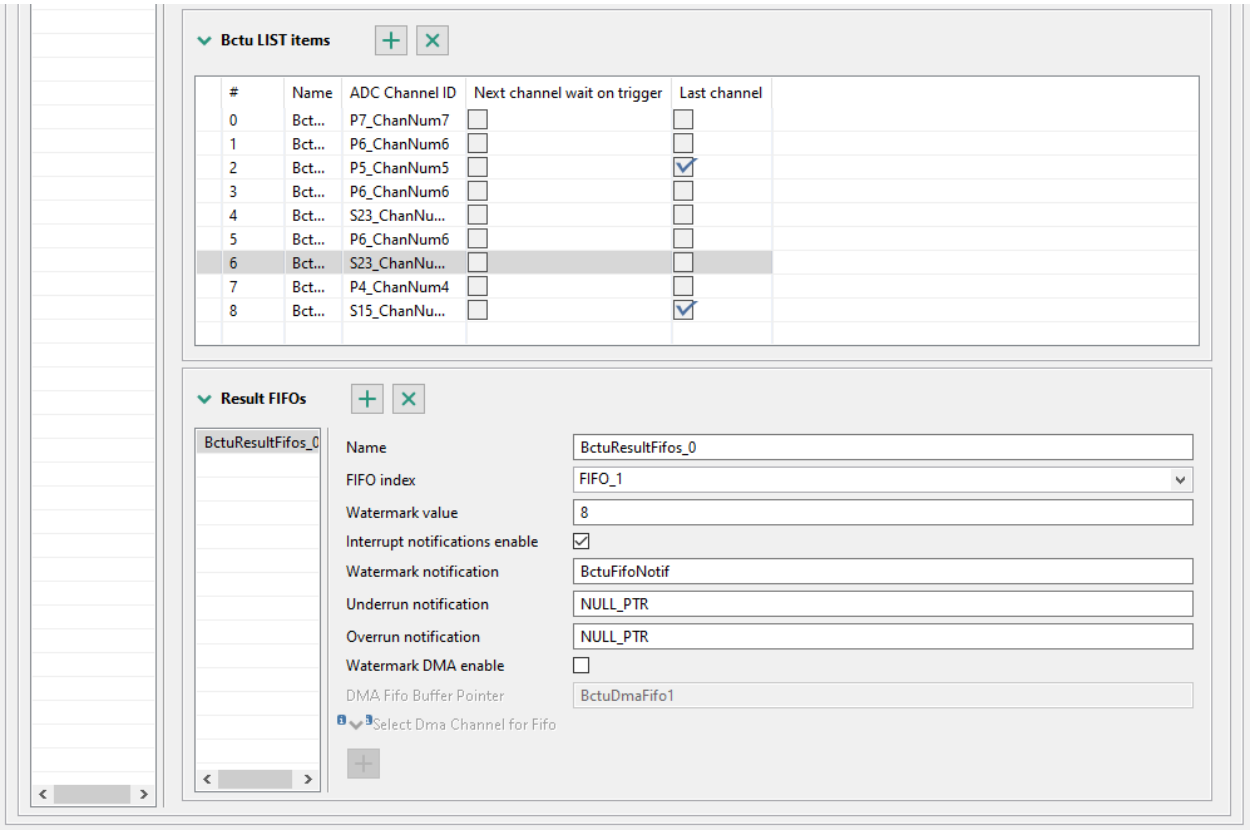


Figure 42. BCTU list and FIFO configuration

Figure 43. Example: BCTU API

```
void main (void)
{
...
   /*************************************************************************************************
    * BCTU Driver
    *************************************************************************************************/
   Bctu_Ip_Init(0U, &BctuHwUnit_0);
   Bctu_Ip_EnableNotifications(0U, BCTU_IP_NOTIF_LIST);
   Bctu_Ip_SetGlobalTriggerEn(0U, TRUE);
...
}
void Bctu_FIFO1_WatermarkNotification (void)
{
...
    mCount = 0;
    while (Bctu_Ip_GetFifoCount(0U, 0U))
    {
        Bctu_Ip_GetFifoResult(0U, 0U, &measuredValues[mCount++]);
    }
...
}
```

4.2.4.3 eMIOS

eMIOS1 CH12 and CH13 are configured to generate the triggers for BCTU in precise moment. Same modes and drivers are used as in the use case of PWM generation in section 4.2.3.1 .Trigger channels use a global time base A (eMIOS1 CH 23). This trigger time base is synchronized with PWM time base with no delay. Considering 160MHz, and Clock Divider Value 1 and Master Bus Prescaler DIV_1, a period 16000 tick means

100µs, so the sampling frequency of motor quantities is 10kHz. An important setting for CH12 and CH13 is "*Flag generation*". Trialing_Edge means generating flag (what is the event when BCTU is triggered) on compare with register B. With a used "*Polarity*" it is falling edge of the CH12 and CH13 outputs, which can be visualized on the pin using TRGMUX. First trigger (eMIOS1 CH12) is generated 206 cycles after reload due to delay of HW circuitry (gate drivers, safe logic, etc.). PWM time base reload and Trigger time base reload are overlapping since there is no delay. Second trigger (eMIOS1 CH13) is generated 320 cycles after reload, which is an exact moment of resolver sine and cosine envelop signal measurement. In this example, the trigger moments are not changing during the runtime, but it is possible to change trigger moment in the same way as update of PWM channels. Settings are applied by calling Emios_Mcl_Ip_Init functions(), Emios_Pwm_Ip_InitChannel(), and Emios_Mcl_Ip_ConfigureGlobalTimebase().

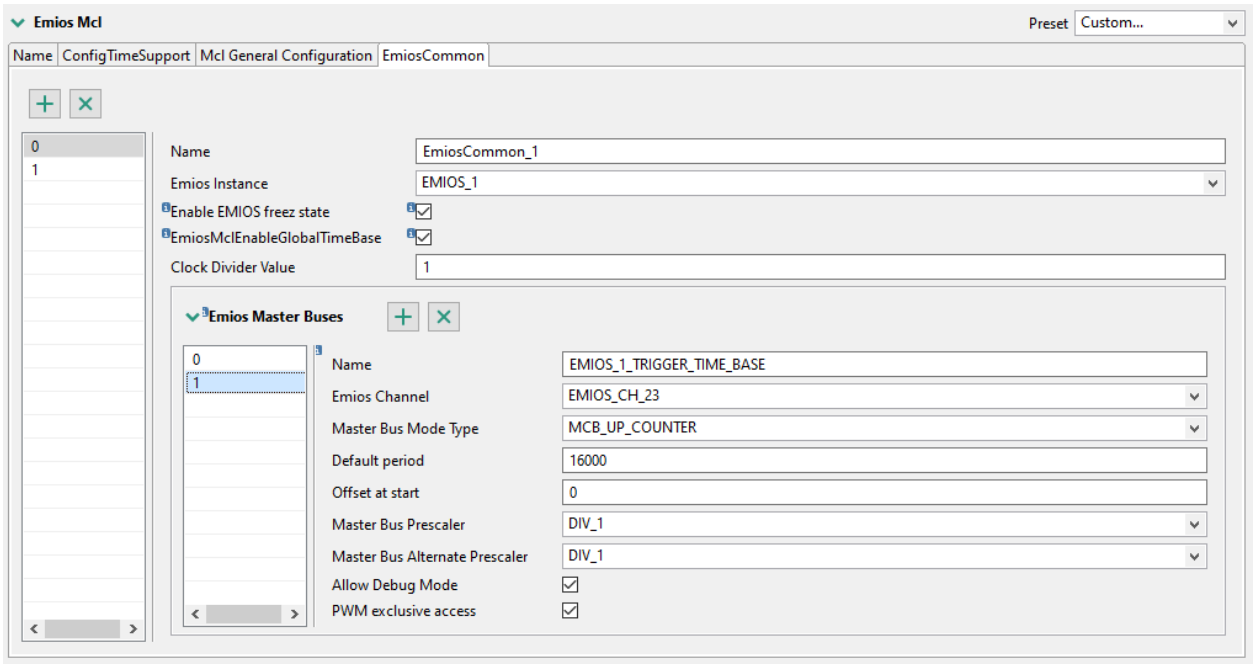


Figure 44. eMIOS trigger time base configuration

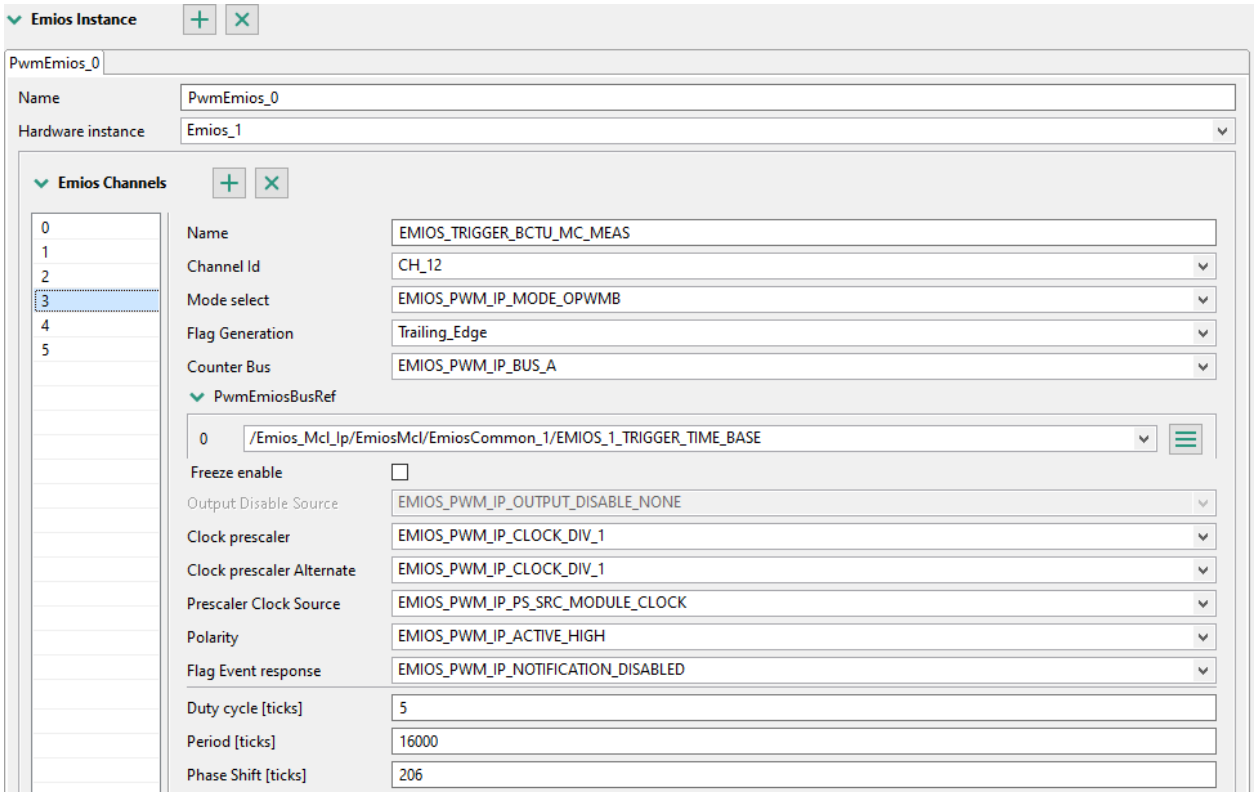


Figure 45. eMIOS phase current trigger configuration

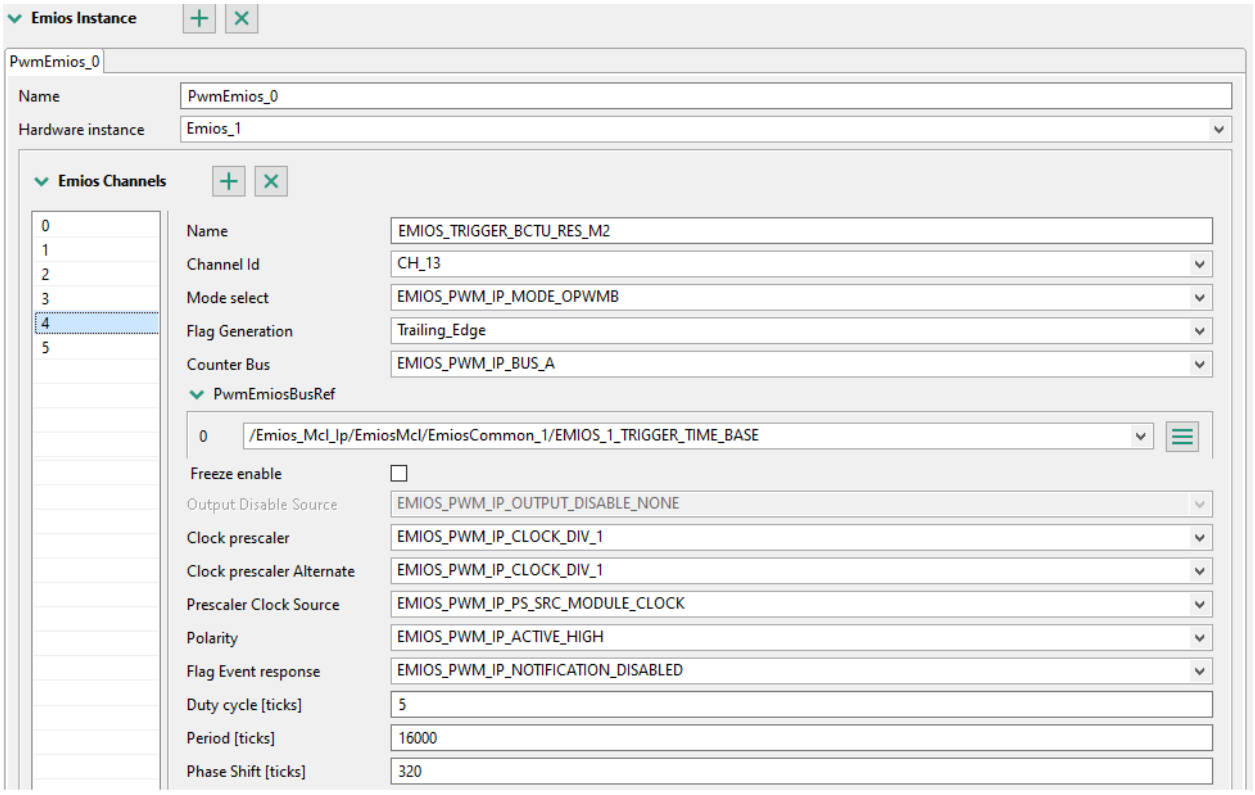


Figure 46. eMIOS resolver trigger configuration

Figure 47. Example: eMIOS API for PWM

```
void main (void)
{
...

       /
*******************************************************************************************
       * eMios Driver

 *******************************************************************************************/
...
   Emios_Mcl_Ip_Init(1U, &Emios_Mcl_Ip_1_Config_BOARD_INITPERIPHERALS);
   Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch12);
   Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch13);
...
   /*Enable eMIOS clock at last to ensure the correct trigger order*
   Emios_Mcl_Ip_ConfigureGlobalTimebase(1U, TRUE);
...
}
```

4.2.5 Resolver sensor

The PMSM motor application uses the resolver for rotor position sensing. Resolver hardware can be viewed as two inductive position sensors, which, upon an excited sinusoidal-shaped signal on input, generate two sinusoidal signals on output. The output signals' amplitudes depend on the position of the shaft. The amplitude of one signal amplitude is proportional to the sine, and the amplitude of the other to the cosine, of the shaft angle position [12].

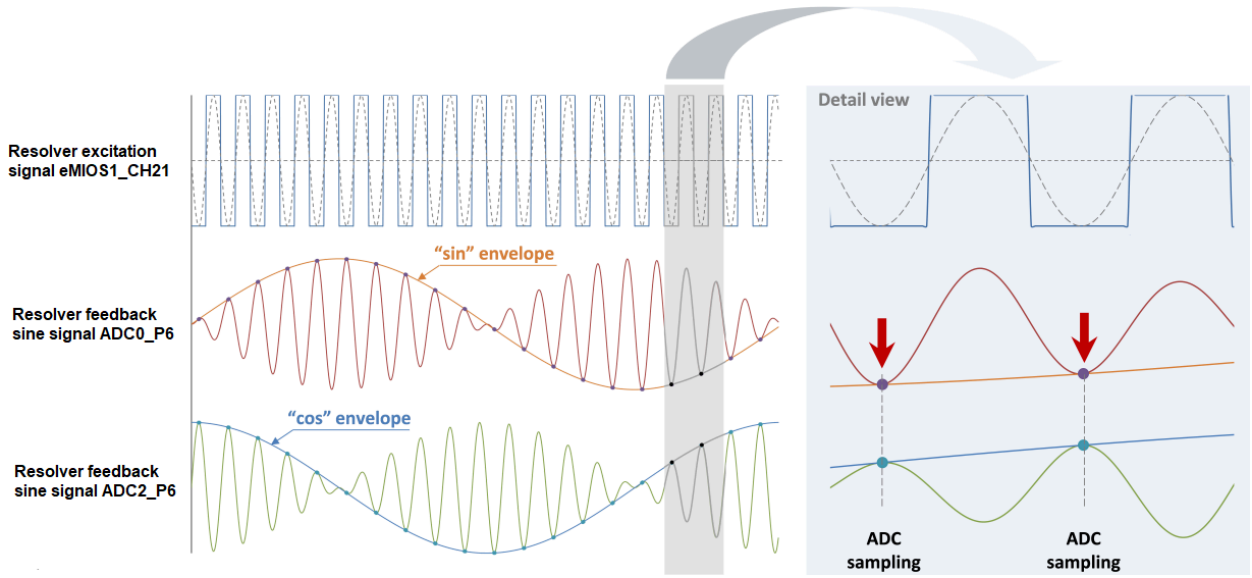


Figure 48. An envelope extractor of resolver feedback signals

Resolver systems utilize an Angle Tracking Observer (ATO), see Figure 49 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 [13], [14].

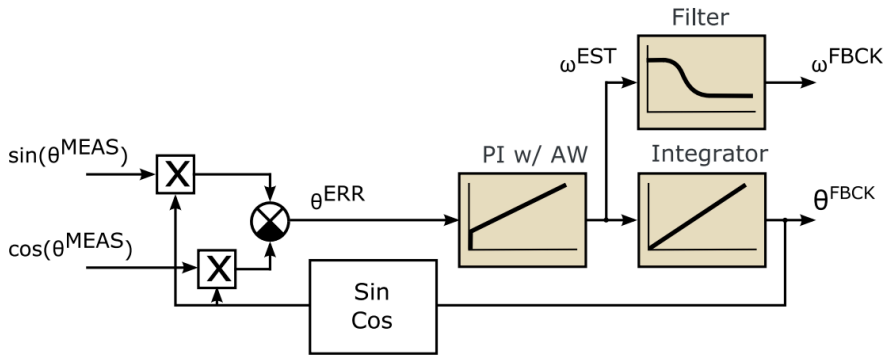


Figure 49. Angle tracking observer for resolver sensor

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

$$sin(\theta_{MEAS})cos(\theta_{FBCK}) - sin(\theta_{FBCK})cos(\theta_{MEAS}) = sin(\theta_{MEAS} - \theta_{FBCK}) \quad (10)$$

The coefficients of ATO PI controller, Integrator and filter can be tuned by MCAT tool. The ATO function is a part of the automotive math and motor control library [5]. NXP 48V development platform uses additional HW circuitry which process PWM excitation signal generated by MCU in order to get pure sinusoidal excitation signal for resolver coil. Moreover, signal processing of differential type of feedback sine and cosine signals of resolver by means of additional HW circuitry is also available on NXP 48V development platform [13], [14]. For more information about signal processing, please refer to [1].

Note: *Only one sensor routine can be enabled at the same time, either encoder or hall or resolver. The others must be disabled. To enable one of them, a dedicated macro (ENCODER, HALL or RESOLVER) has to be set true.*

4.2.5.1 eMIOS

eMIOS1 CH23 is configured as a time base for trigger signals and resolver excitation signal. This channel can create global time base for all eMIOS1 channels. Chanel operates in a Modulus Counter Buffered (MCB) mode where there is just up counting. When the internal counter matches a value defined by field period (channel register A of the eMIOS channel) and a clock tick occurs, the internal counter is reset to 1 and reload is generated. Considering 160MHz and Clock Divider Value 1 and Master Bus Prescaler DIV_1, the "*Default period*" 16000 ticks means 100µs/10kHz. Configuration of eMIOS driver is depicted in Figure 44.

eMIOS1 CH21 is configured to generate the PWM signal for resolver excitation. Channel operates in Output PWM Buffered (OPWMB) mode similarly to PWM signals. Channel select global time base A as a counter bus and time base settings are also referenced through "*PwmEmiosBusRef*" field. Channel is able to see time base counter value through the A bus and compare it with its registers A and B. "*Polarity*" defines output state on specific compare, which is active low. Driver offers an abstraction where "*duty cycle*" is an active pulse (space between compare A and B) and "*Phase shift*" defines placement of this active pulse within the PWM period. Proper values for register A and B are calculated by driver. Init values of the "*Phase shift*" and "*duty cycle*" are set in Peripherals tool. Resolver excitation signal operates with 50% duty cycle and it is shifted in order to match sampling of sine cosine envelope signals with excitation signal. eMIOS1 CH13 is configured to generate the trigger for BCTU in precise moment, where envelop sine and cosine signals are measured.

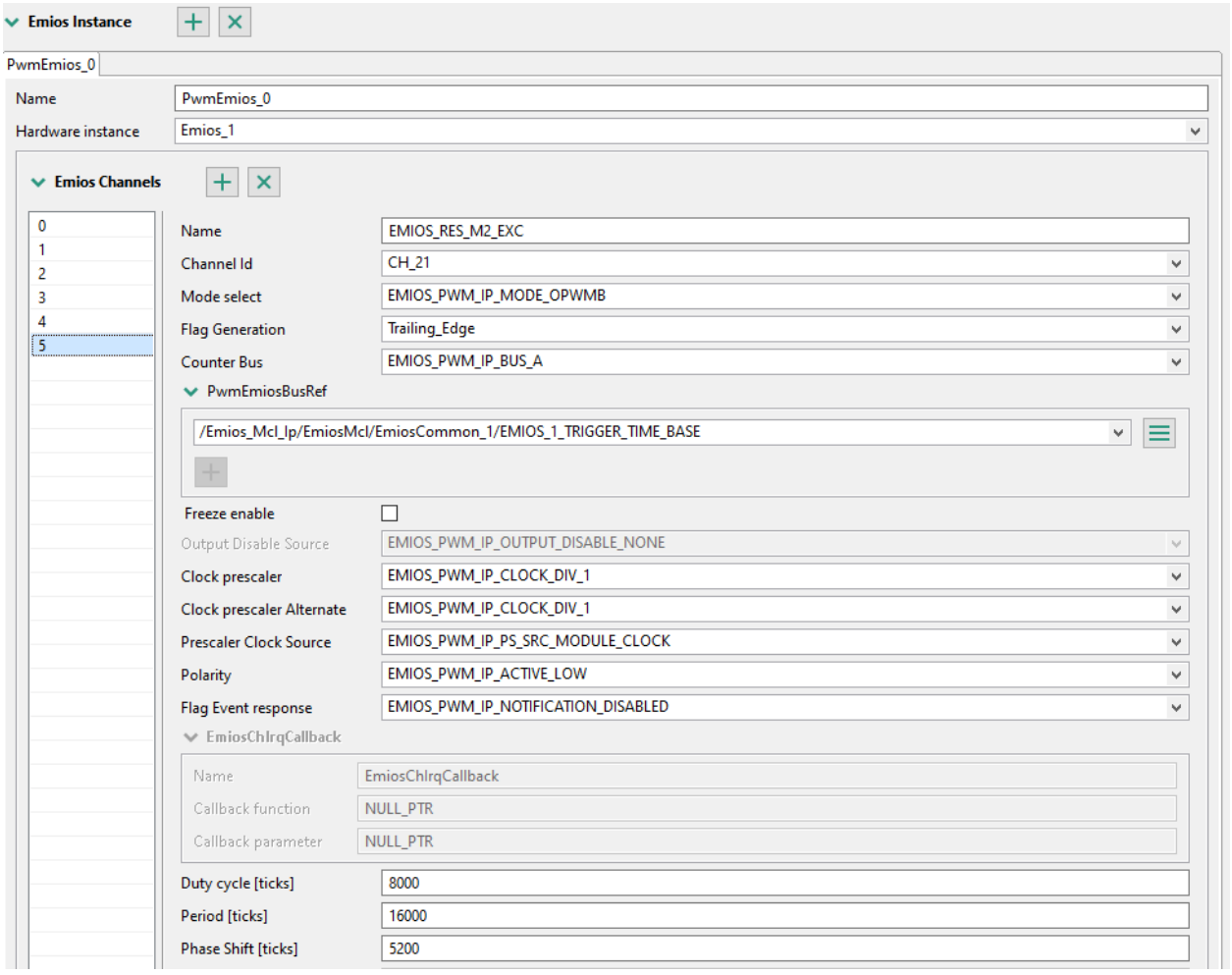


Figure 50. Resolver excitation channel configuration

For more details about configuration of eMIOS trigger signal for resolver measurement, please refer to 4.2.4.3. Timing diagram of resolver signal processing is depicted in Figure 12. Settings are applied by calling Emios_Pwm_Ip_InitChannel(), Emios_Mcl_Ip_Init() and Emios_Mcl_Ip_ConfigureGlobalTimebase(), where after calling the Emios_Mcl_Ip_ConfigureGlobalTimebase() time base counting is started. PWM signal is static and is not changing during the runtime.

Figure 51. Example: eMIOS API for PWM

```
void main (void)
{
...
/*******************************************************************************************
 * eMios Driver
 *******************************************************************************************/
...
     Emios_Mcl_Ip_Init(1U, &Emios_Mcl_Ip_1_Config_BOARD_INITPERIPHERALS);
     Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch13);
     Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch21);
...
     /*Enable eMIOS clock at last to ensure the correct trigger order*/
     Emios_Mcl_Ip_ConfigureGlobalTimebase(1U, TRUE);
...
}
```

4.2.5.2 ADC

Resolver sensor requires involvement of ADC module due to measurement of analog feedback signals of sine and cosine. All important details about ADC module of S32K344 and configuration of it also for resolver signal processing purposes are already mentioned in section 4.2.4.1.

4.2.5.3 BCTU

Resolver sensor requires also involvement of BCTU module due to measurement of analog feedback signals of sine and cosine. All important details about BCTU module of S32K344 and configuration of it also for resolver signal processing purposes are already mentioned in section 4.2.4.2.

4.2.6 Quadrature decoder

Quadrature decoder feature is achieved by cooperation of eMIOS, TRGMUX and LCU modules. This feature is used to decode the quadrature signals generated by rotary sensors used in motor control domain. This mode is used to process encoder signals and determine rotor position and speed.

There are three output signals generated by incremental encoder as shown in Figure 52. Phase A and Phase B signals consist of a series of pulses which are phase-shifted by 90° (therefore the term “quadrature” is used). The third signal (called “Index”) provides the absolute position information. In the motion control, it is used to check the pulse-counting consistency. Index signal is not used in this example hence position offset is calibrated during the rotor alignment.

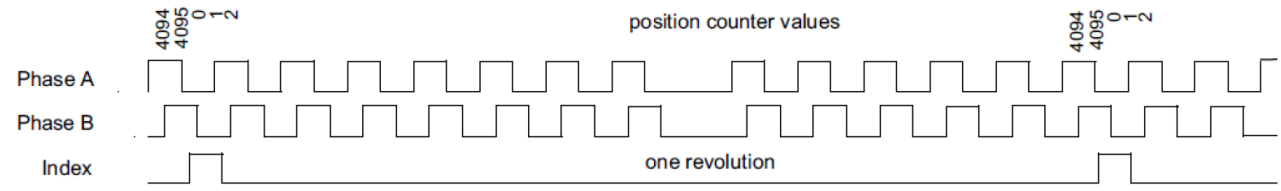


Figure 52. Output signals of the 1024 pulses encoder

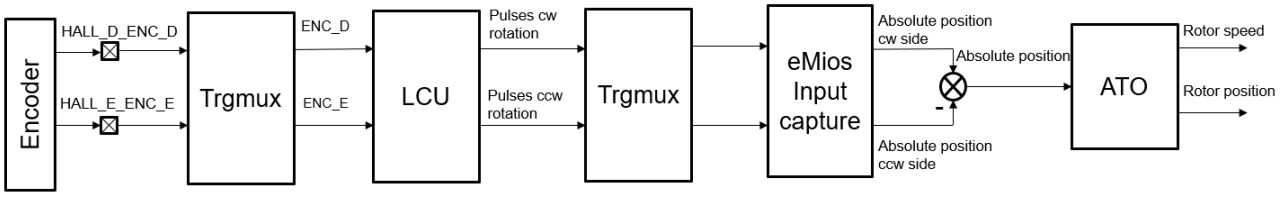


Figure 53. Peripherals interconnection for quadrature encoder

In order to get the rotor position, encoder signals HALL_D_ENC_D and HALL_E_ENC_E are brought to the LCU. LCU preprocess them and generates pulses based on rotor speed direction. eMIOS acts as a counter to get the rotor absolute position. An angle tracking observer (ATO) is used to calculate the final rotor speed and position. Emios_Icu, Lcu_Ip and Trgmux_Ip drivers are used to control and to configure peripherals for this use case.

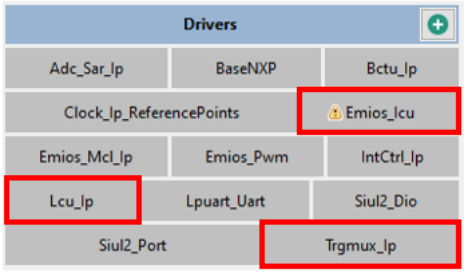


Figure 54. Drivers for quadrature decoder feature

Note: *Only one sensor routine can be enabled at the same time, either encoder or hall or resolver. The others must be disabled. To enable one of them, a dedicated macro (ENCODER, HALL or RESOLVER) has to be set true.*

4.2.6.1 TRIGGER MUX

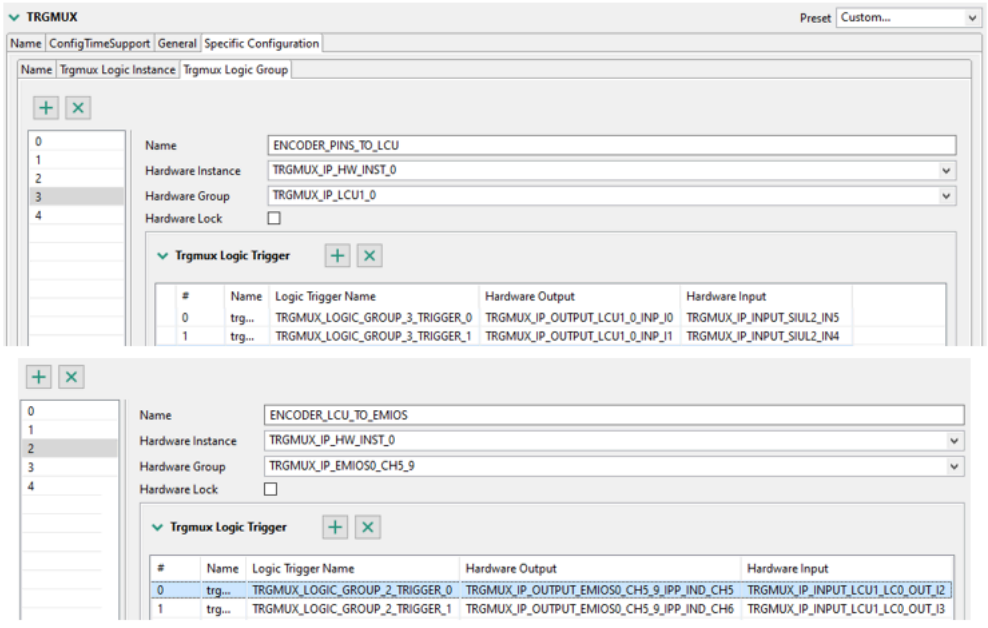


Figure 55. TRGMUX settings for quadrature decoder

TRGMUX ensures a connection between Input pins and LCU and between eMIOS and LCU. Settings within the “*Hardware Group*” ENCODER_PINS_TO_LCU connects PTD2 (TRGMUX_IN5) and PTD3 (TRGMUX_IN4) to LCU1 inputs 0-1. Settings within the “*Hardware Group*” ENCODER_LCU_TO_EMIOS connects LCU1 LC0 outputs 2-3 to eMIOS0 inputs of channels 5-6. The setting is applied by calling Trgmux_Ip_Init() function.

4.2.6.2 LCU

In this example LCU1 instance is selected for preprocessing encoder signals phase D and phase E. Same LCU features are used as in section 4.2.3.3 but, whole preprocessing is realized in LC0. Configurations 3-6 in the "*Lcu Logic Input*" section create a connection between LCU instance inputs and LC inputs. Multiplexor inputs 0,1(pins PTD2, PTD3) are connected to LC0 input 0,1 and multiplexor feedback inputs 0,1 (LC0 out 0,1) are connected to LC0 input 2,3. Outputs configurations are in the section "*Lcu Logic Output*" configurations 6-9. Look Up Table inputs 0,1 are mirrored to outputs 0,1 and rising and falling edges are delayed by digital filters. Look Up Table of outputs 2,3 use information of all associated inputs. They detect edges of the encoder phases ENC_PHD and ENC_PHE using auxiliary signals ENC_PHD0 and ENC_PHE0 as is depicted in waveform Figure 56 Detected edge is represented by short pulse (in this example 5 ticks filters settings of outputs 0,1). Based on actual value of signals ENC_PHD and ENC_PHE, logic function in LUT distinguishes the direction of rotation and a detected edge is placed on proper output (clockwise or counterclockwise). Look Up Table of all outputs (given by LUT) can be found in Table 5. Filters of outputs 2,3 work as a glitch filters. If generated pulse is shorter than 4 ticks it will not appear on the output. It is protection against a noise on HALL_D_ENC_D and HALL_E_ENC_E pins. All settings are applied by calling Lcu_Ip_Init() function.

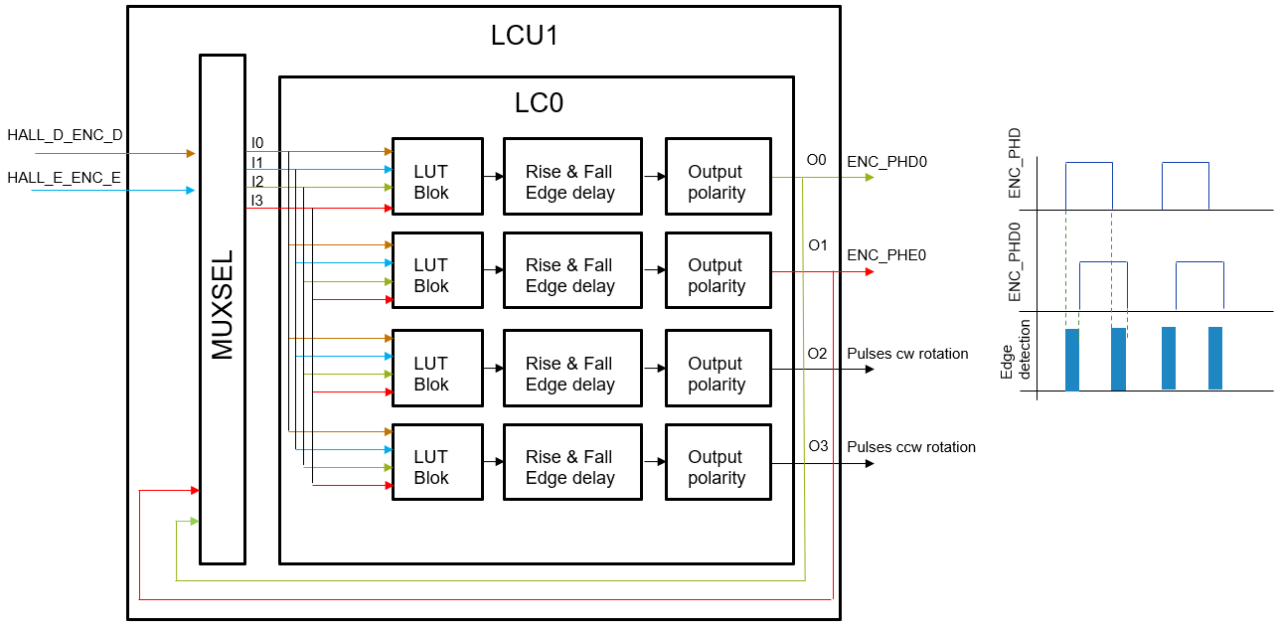


Figure 56. Simplified LCU features block diagram for quadrature decoder

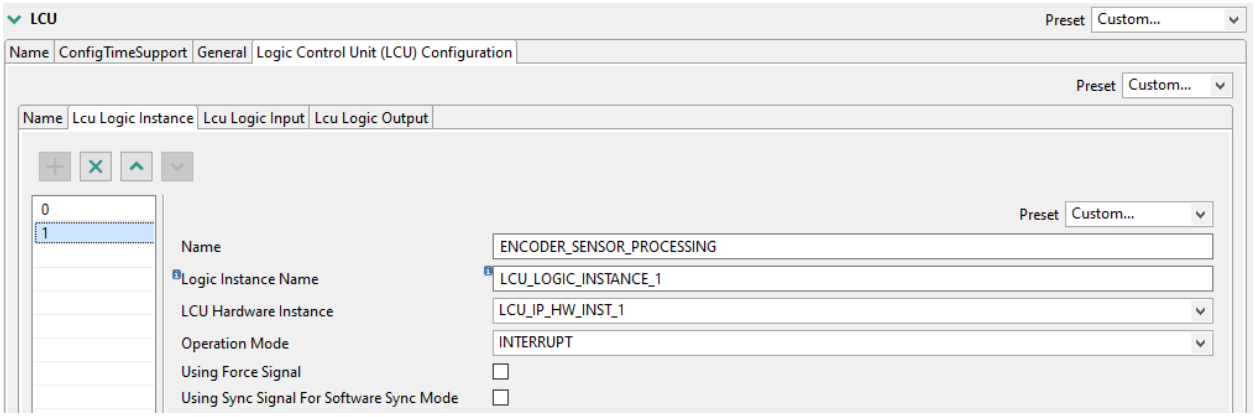


Figure 57. LCU instance configuration for quadrature decoder

Table 5. LUT configurations for LCU1 LC0

LC0_I3	LC0_I2	LC0_I1	LC0_I0		LC0_O0	LC0_O1	LC0_O2	LC0_O3
ENC_PHE0	ENC_PHD0	ENC_PHE	ENC_PHD		ENC_PHD0	ENC_PHE0	Pulses cw	Pulses ccw
0	0	0	0		0	0	0	0
0	0	0	1		1	0	1	0
0	0	1	0		0	1	0	1
0	0	1	1		1	1	0	0
0	1	0	0		0	0	0	1
0	1	0	1		1	0	0	0
0	1	1	0		0	1	0	0
0	1	1	1		1	1	1	0
1	0	0	0		0	0	1	0
1	0	0	1		1	0	0	0
1	0	1	0		0	1	0	0
1	0	1	1		1	1	0	1
1	1	0	0		0	0	0	0
1	1	0	1		1	0	0	1
1	1	1	0		0	1	1	0
1	1	1	1		1	1	0	0
LUT					**0xAAAA**	**0xCCCC**	**0x4182**	**0x2814**

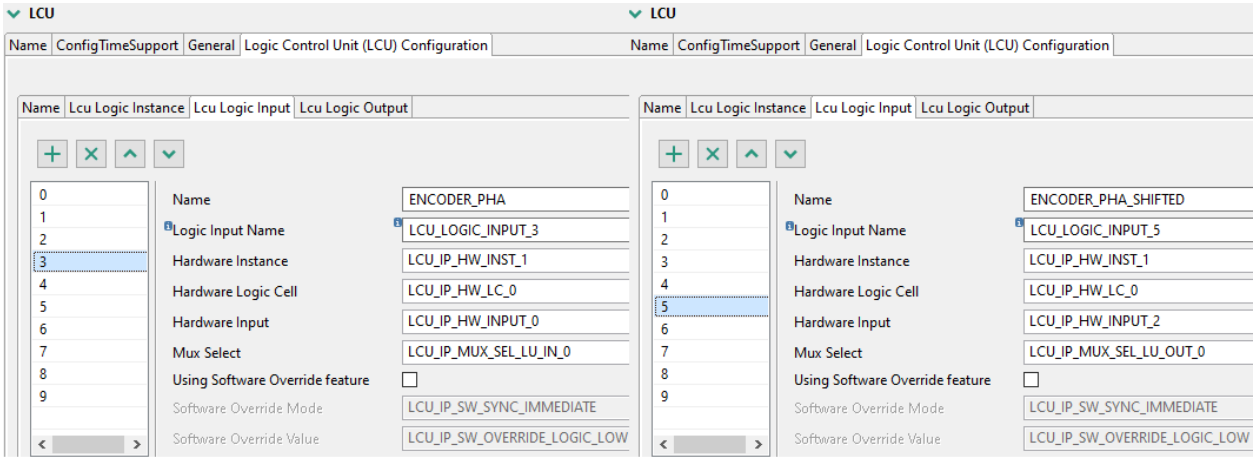


Figure 58. LCU inputs configuration for quadrature decoder

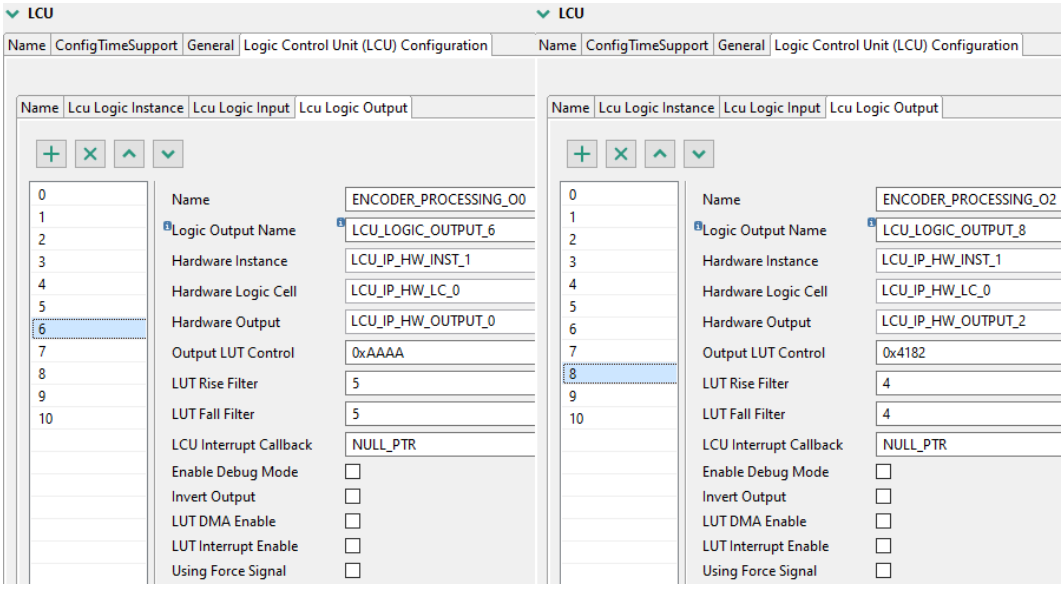


Figure 59. LCU outputs for quadrature decoder

4.2.6.3 eMIOS

eMIOS0 CH5 and CH6 are channels of type G so they contain their own counter and are able to count edges of the channel input signal. In the example, those channels operate in modulus counter buffered (MCB) mode and count rising edges of signals coming from LCU which represents detected edges of signals HALL_D_ENC_D and HALL_E_ENC_E of the encoder sensor. For more details about eMIOS channel types see S32K3xx Reference Manual [7]. All settings are applied by calling Emios_Icu_Ip_Init(), Emios_Icu_Ip_SetInitialCounterValue(), Emios_Icu_Ip_SetMaxCounterValue() functions and by enabling edge counting using Emios_Icu_Ip_EnableEdgeCount() functions. Actual counter value is obtained by calling Icu_GetEdgeNumbers() for particular channel.

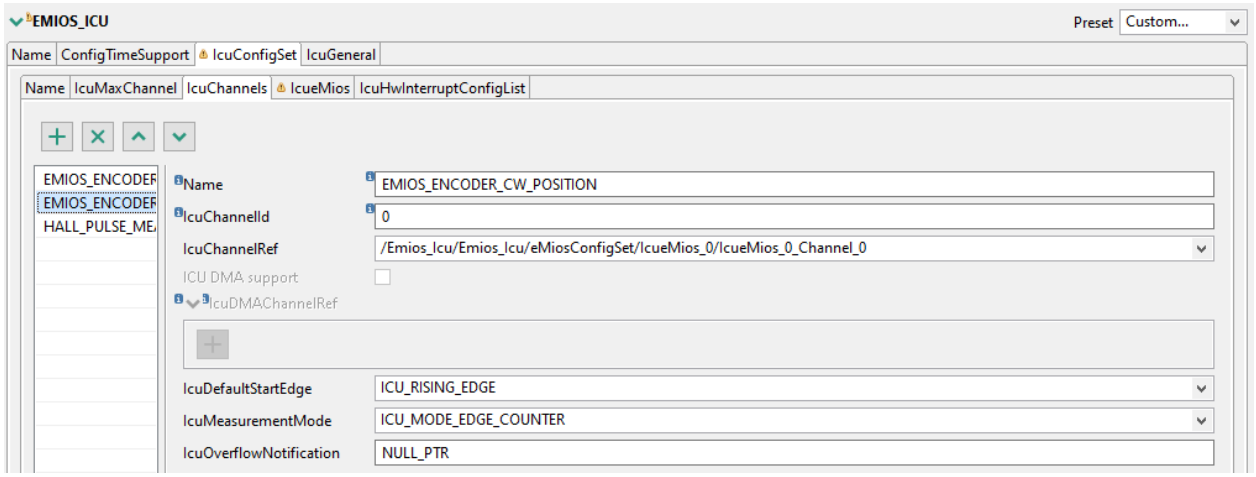


Figure 60. General ICU configuration

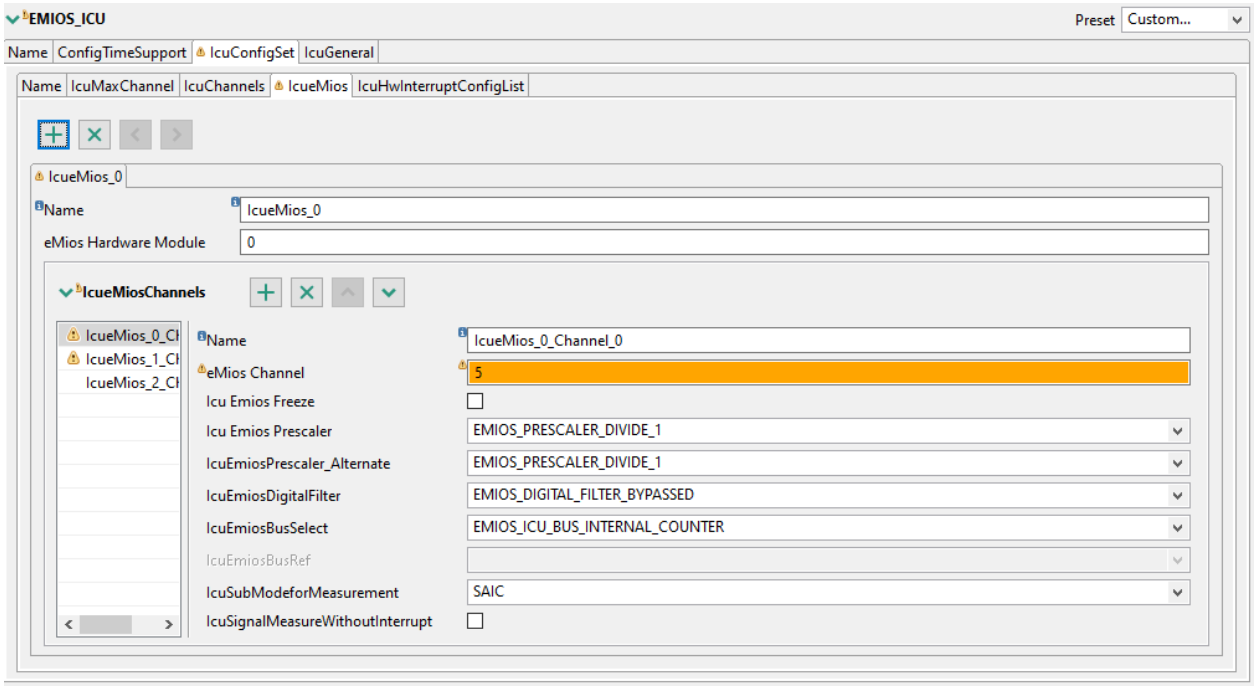


Figure 61. eMIOS channels for input capture

Note: *Property IcuSubModeforMeasurement is not applicable for ICU_MODE_EDGE_COUNTER. Channels are set into MCB mode by calling Emios_Icu_Ip_EnableEdgeCount function.*

Figure 62. Example: eMIOS API for quadrature decoder

```
void main (void)
{
...
   /*************************************************************************************************
    * eMios Driver
    *************************************************************************************************/
   Emios_Icu_Ip_Init(0U, &eMios_Icu_Ip_0_Config_PB);
   Emios_Icu_Ip_SetInitialCounterValue(0U, 5U, (uint32_t)0x1U);
   Emios_Icu_Ip_SetInitialCounterValue(0U, 6U, (uint32_t)0x1U);
   Emios_Icu_Ip_SetMaxCounterValue(0U, 5U, (uint32_t)(4*ENC_PULSES));
```

```
    Emios_Icu_Ip_SetMaxCounterValue(0U, 6U, (uint32_t)(4*ENC_PULSES));
    Emios_Icu_Ip_EnableEdgeCount(0u, 5U);
    Emios_Icu_Ip_EnableEdgeCount(0u, 6U);
...
}
tBool POSPE_GetPospeElEnc (encoderPospe_t *ptr)
{
...
    counterCW = (uint16_t)((Emios_Icu_Ip_GetEdgeNumbers(0U, 5U))- ptr->counterCwOffset); /* CW
 counter */
    counterCCW = (uint16_t)((Emios_Icu_Ip_GetEdgeNumbers(0U, 6U))- ptr->counterCcwOffset); /* CCW
 counter */
...
}
```

Note: *Various input pins or TRGMUX output can be selected for eMIOS input. This selection is realized in SIUL2 IMCR register.*

4.2.7 Communication

4.2.7.1 UART

LPUART1 is used as a communication interface between S32K344 MCU and FreeMASTER run-time debugging and visualization tool. Lpuart_Uart RTD driver is used to configure LPUART. Configuration is applied by calling Lpuart_Uart_Ip_Init(). LPUART must be configure before any API of FreeMASTER embedded driver is called (functions: FMSTR_Init(), FMSTR_Poll(), FMSTR_Recorder()). For more details about FreeMASTER see [4].

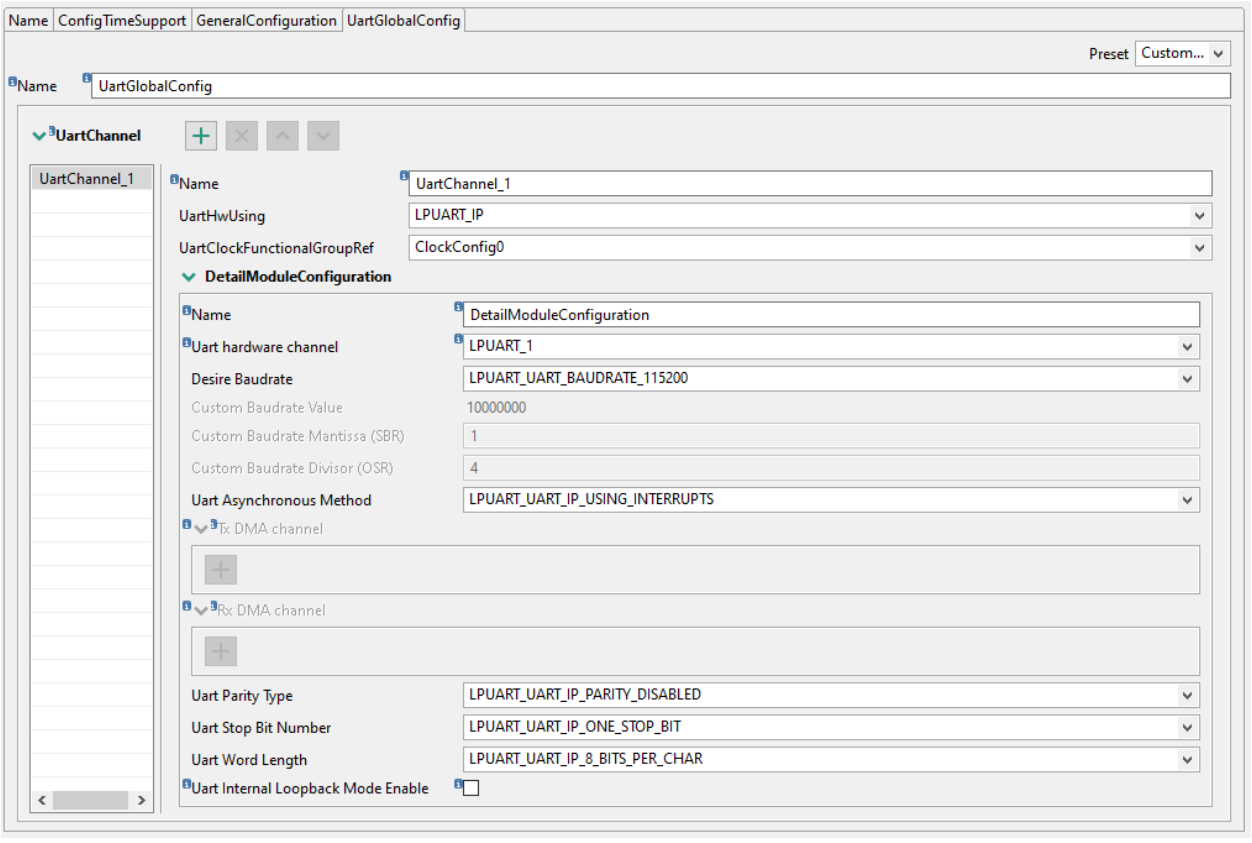


Figure 63. LPUART configuration

4.2.8 Hall sensor FOC

Electrically commutated motor with hall sensors is usually used for block commutation control techniques. FOC algorithm requires to know position of the rotor precisely in order to achieve high performance and efficiency of the drive. Due to cost reason, utilization of hall sensors and BLDC type of motors with FOC control technique is becoming more popular despite of lower FOC performance.

The hall sensors are sensing elements, which provide logic level output based on detected magnetic field. The hall sensors are usually inserted into the stator. If hall sensor, placed in stator, is closer to north pole of the permanent magnet, placed on rotor, then its output signal provides log. 1, otherwise log. 0. Standard 3-phase BLDC machines have three hall sensors (one per phase), which means a combination of 8 status from 000 to 111. However, due to the hardware constraint, signals 000 and 111 don't exist. The rest of 6 status can divide one electrical revolution (360°) into six areas, where our initial position is set in sector 0 after mechanical alignment. Waveform of hall sensors with sector displacement is depicted in Figure 64.

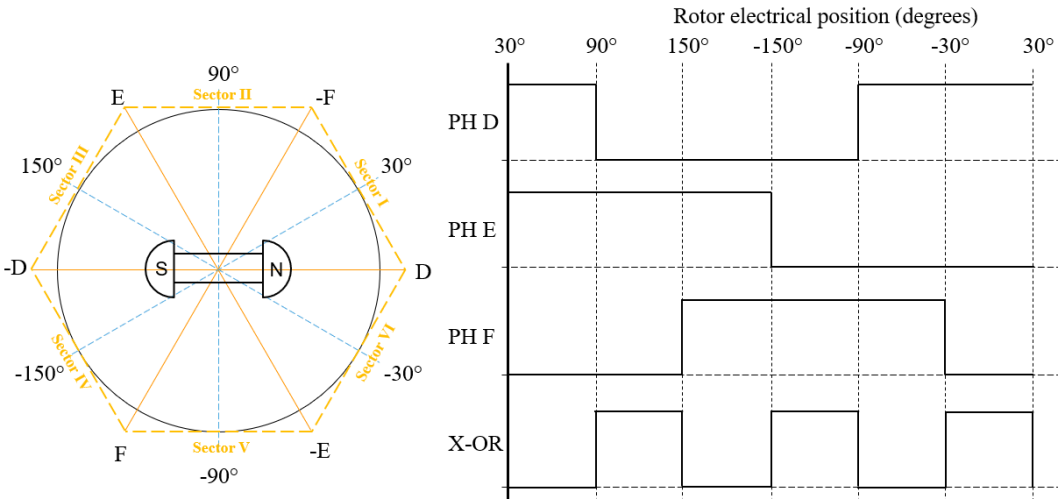


Figure 64. Hall signals waveform

In order to process signals from hall sensors for FOC application, various MCU modules and SW routines need to be implemented. NXP 48V development platform uses same interface for hall sensors and encoder sensor. Block diagram of MCU peripherals with SW routines is depicted in Figure 65.

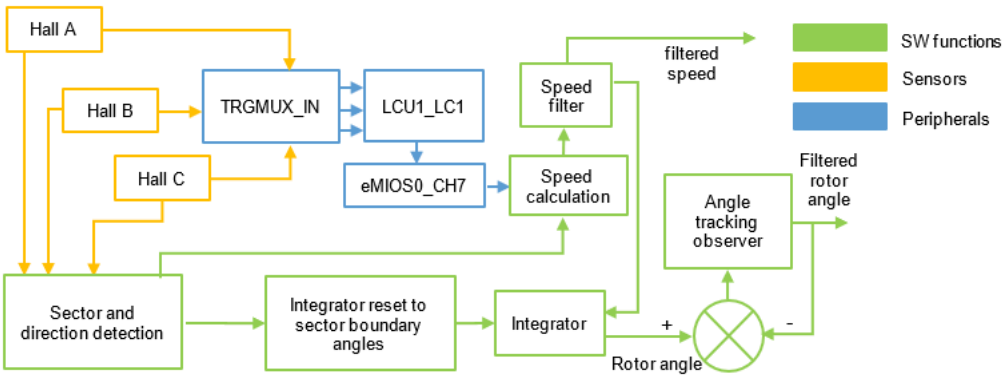


Figure 65. FOC hall signal processing block diagram

Rotor angle is calculated by integration of speed acquired from hall sensors. Speed integrator is reset upon detection of a new hall state. There are 6 different hall states per electrical revolution which are in ideal case 60 degrees apart. The exact electrical angle of hall sensor is known, therefore speed integrator can be reset to

exact rotor position 6 times per electrical revolution. In real world, the hall sensors are not exactly 60 degrees apart, thus a calibration routine and displacement compensation is needed for more accurate motor control. This calibration routine is executed only at once. Sector and direction of rotation is detected by means of reading hall sensor signal level and ATO is used in order to filter position of rotor. Speed is calculated according to formula below:

$$speed = (A_2 - A_1)\frac{f_{eMIOS}}{P} \quad (11)$$

A_2, A_1 are current and previous angle, f_{eMIOS} is an eMIOS frequency and P is number of pulses counted by eMIOS between two consecutive X-OR output pulses.

Note: *Only one sensor routine can be enabled at the same time, either encoder or hall or resolver. The others must be disabled. To enable one of them, a dedicated macro (ENCODER, HALL or RESOLVER) has to be set true.*

4.2.8.1 TRIGER MUX

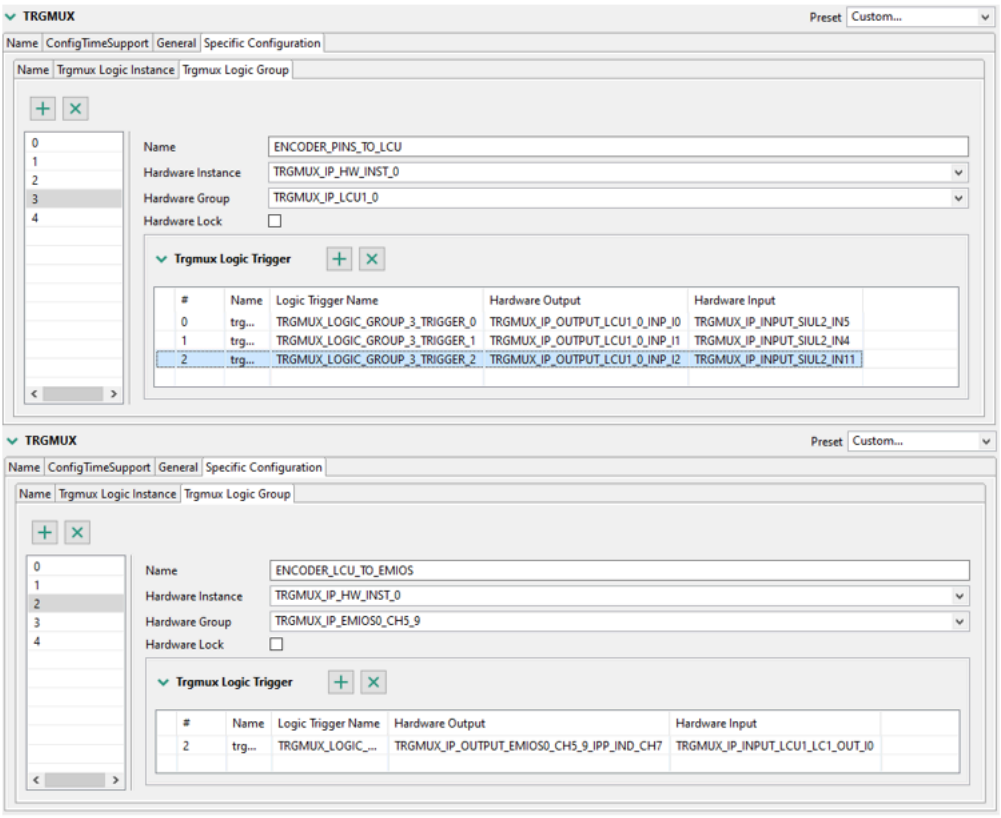


Figure 66. TRGMUX settings for quadrature decoder

TRGMUX ensures a connection between Input pins and LCU and between eMIOS and LCU. Settings within the "*Hardware Group*" ENCODER_PINS_TO_LCU connects PTD2(TRGMUX_IN5), PTD3(TRGMUX_IN4) and PTC10 (TRGMUX_IN11) to LCU1 inputs 0-2. Settings within the "*Hardware Group*" ENCODER_LCU_TO_EMIOS connects LCU1 LC1 outputs 0 to eMIOS0 inputs of channels 7. The setting is applied by calling Trgmux_Ip_Init() function.

Note: *Interconnection of hall sensors with LCU inputs is an identical with encoder sensor due to utilization of same HW interface for encoder and hall on NXP 48V development platform.*

4.2.8.2 LCU

In this example LCU1 instance is selected for preprocessing hall signals phase D, phase E and phase F. Same LCU features are used as in section 4.2.3.3 but, whole preprocessing is realized in LC1. Configurations 7-9 In "*Lcu Logic Input*" section create a connection between LCU instance inputs and LC inputs. Multiplexor inputs 0-2 (pins PTD2, PTD3 and PTC10 routed through TRGMUX) are connected to LC1 input 0,1,2. Outputs configuration is in a section "*Lcu Logic Output*" configuration 10. True table of output 0, depicted in Table 6, creates X-OR type of signal from input hall signals. Utilization of LCU with waveforms of input hall signals and output X-OR signal is depicted in Figure 67. All settings are applied by calling Lcu_Ip_Init() function.

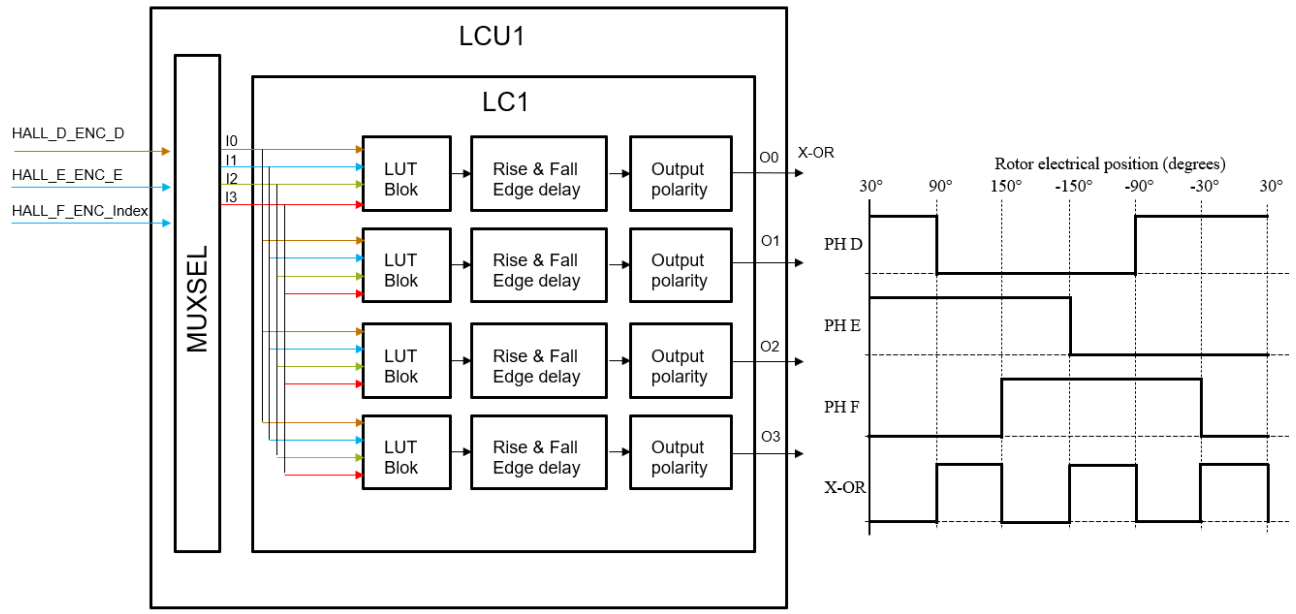


Figure 67. Simplified LCU features block diagram for FOC hall sensor signal processing

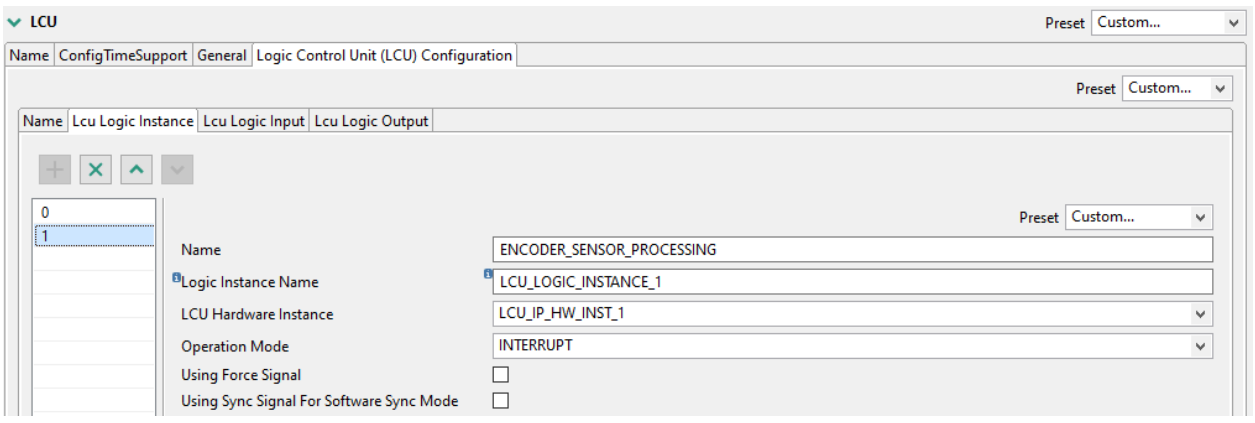


Figure 68. LCU instance configuration for FOC hall sensor signal processing

Table 6. LUT configurations for LCU1 LC1

LC1_I3	LC1_I2	LC1_I1	LC1_I0		LC1_O0	LC1_O1	LC1_O2	LC1_O3
x	HALL_F	HALL_E	HALL_D		X-OR	x	x	x
0	0	0	0		0	0	0	0

Table 6. LUT configurations for LCU1 LC1*...continued*

LC1_I3	LC1_I2	LC1_I1	LC1_I0		LC1_O0	LC1_O1	LC1_O2	LC1_O3
x	HALL_F	HALL_E	HALL_D		X-OR	x	x	x
0	0	0	1		1	0	0	0
0	0	1	0		1	0	0	0
0	0	1	1		0	0	0	0
0	1	0	0		1	0	0	0
0	1	0	1		0	0	0	0
0	1	1	0		0	0	0	0
0	1	1	1		1	0	0	0
1	0	0	0		0	0	0	0
1	0	0	1		1	0	0	0
1	0	1	0		1	0	0	0
1	0	1	1		0	0	0	0
1	1	0	0		1	0	0	0
1	1	0	1		0	0	0	0
1	1	1	0		0	0	0	0
1	1	1	1		1	0	0	0
LUT					**0x9696**			

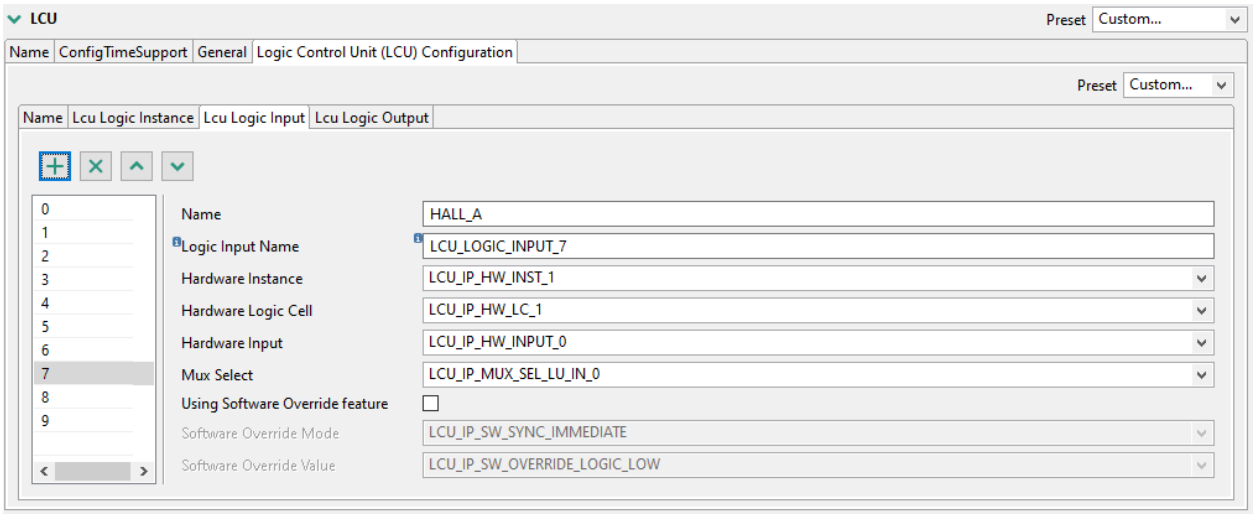


Figure 69. LCU inputs configuration for FOC hall sensor signal processing

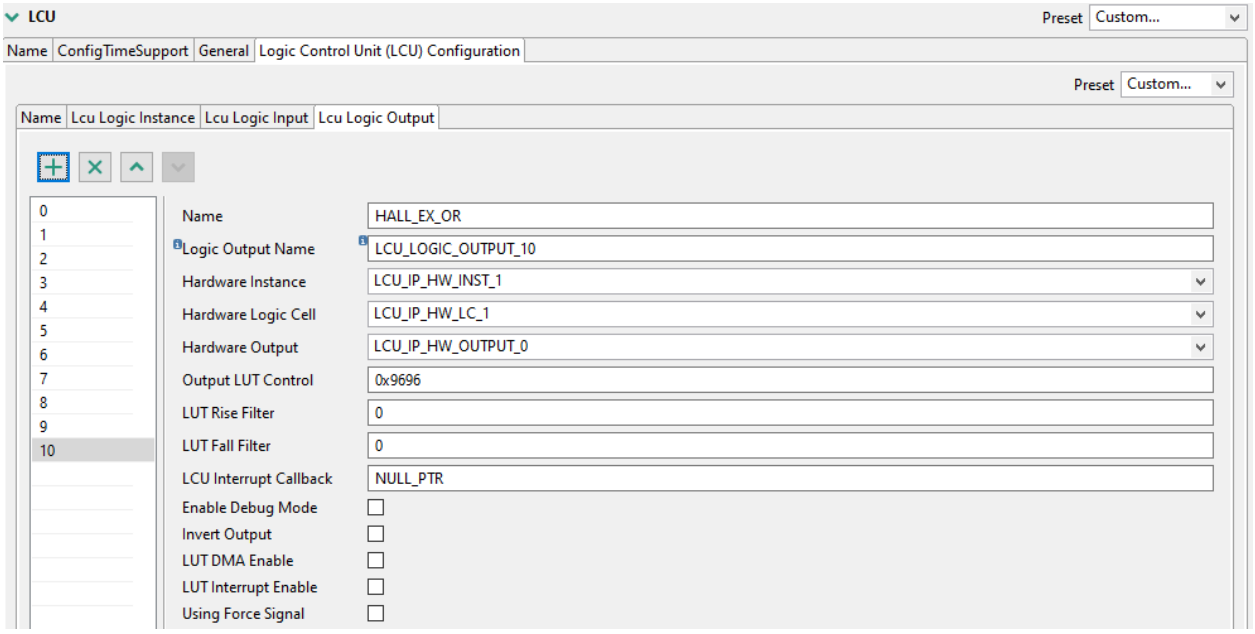


Figure 70. LCU outputs for FOC hall sensor signal processing

4.2.8.3 eMIOS

eMIOS0 CH23 is configured as a time base for pulse width measurement of X-OR signal coming from LCU. This channel represents global time base so all channels of eMIOS0 can see time base counter value. Chanel operates in a Modulus Counter Buffered (MCB) mode where there is just up-counting. Considering 160MHz, Clock Divider 256 and Master Bus Prescaler DIV_1 the "*Default period*" 65535 ticks means 1.6µs/625kHz. LCU X-OR output signal is interconnected with eMIOS0 CH7 channel. This channel has a possibility to see time base counter value of eMIOS0 CH23. When raising or falling edge of X-OR signal comes, value of eMIOS0 CH23 counter is captured. Once counter of eMIOS0 CH23 overflows, an interrupt is raised and handled by eMIOS RTD interrupt handler. In addition, custom notification function eMIOS0IcuNotify is called. In this routine, captured value are processed and rotor speed and position are calculated. eMIOS0 CH7 can operate in single action input capture (SAIC) mode. In this example, detection of both edges is set by "*IcuDefaultStartEdge*" property, timestamp mode of operation is configured by "*IcuMeasurementMode*" property and circular buffer is used in "*IcuTimestampMeasurementProperty*" property. For more details about eMIOS module and configuration see S32K3xx Reference Manual [7].

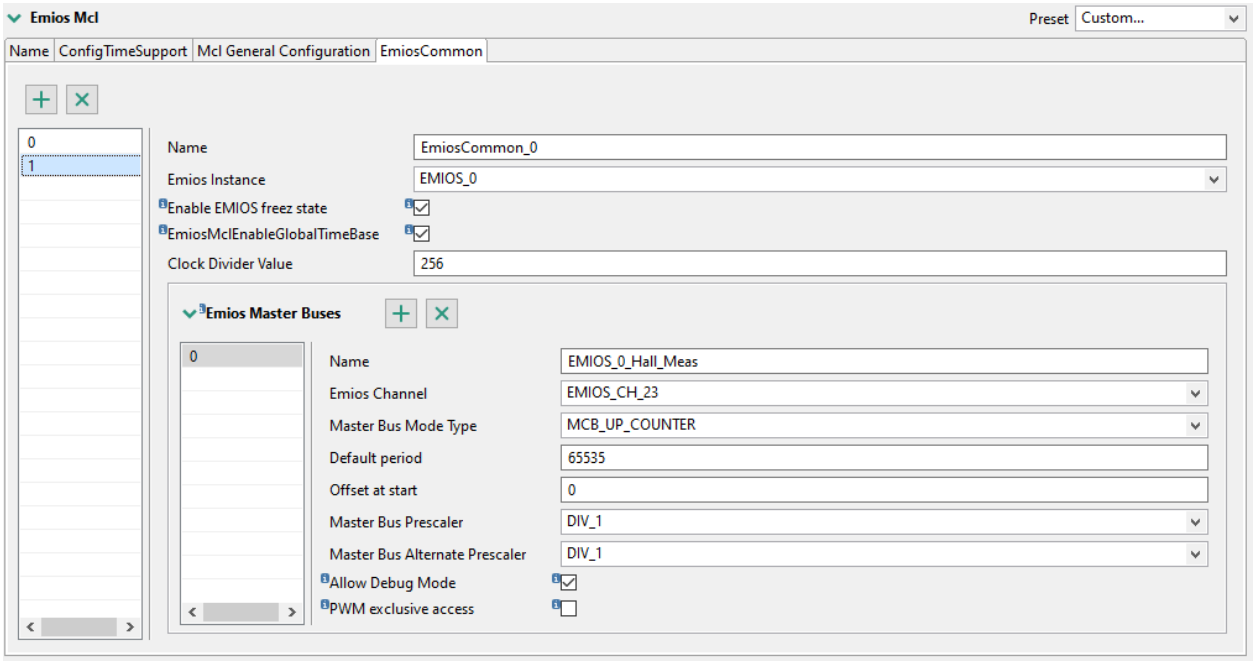


Figure 71. General ICU configuration

All settings are applied by calling Emios_Icu_Ip_Init() and Emios_Icu_Ip_StartTimestamp(). Enabling of pulse width measurement is handled by Emios_Icu_Ip_EnableNotification() function.

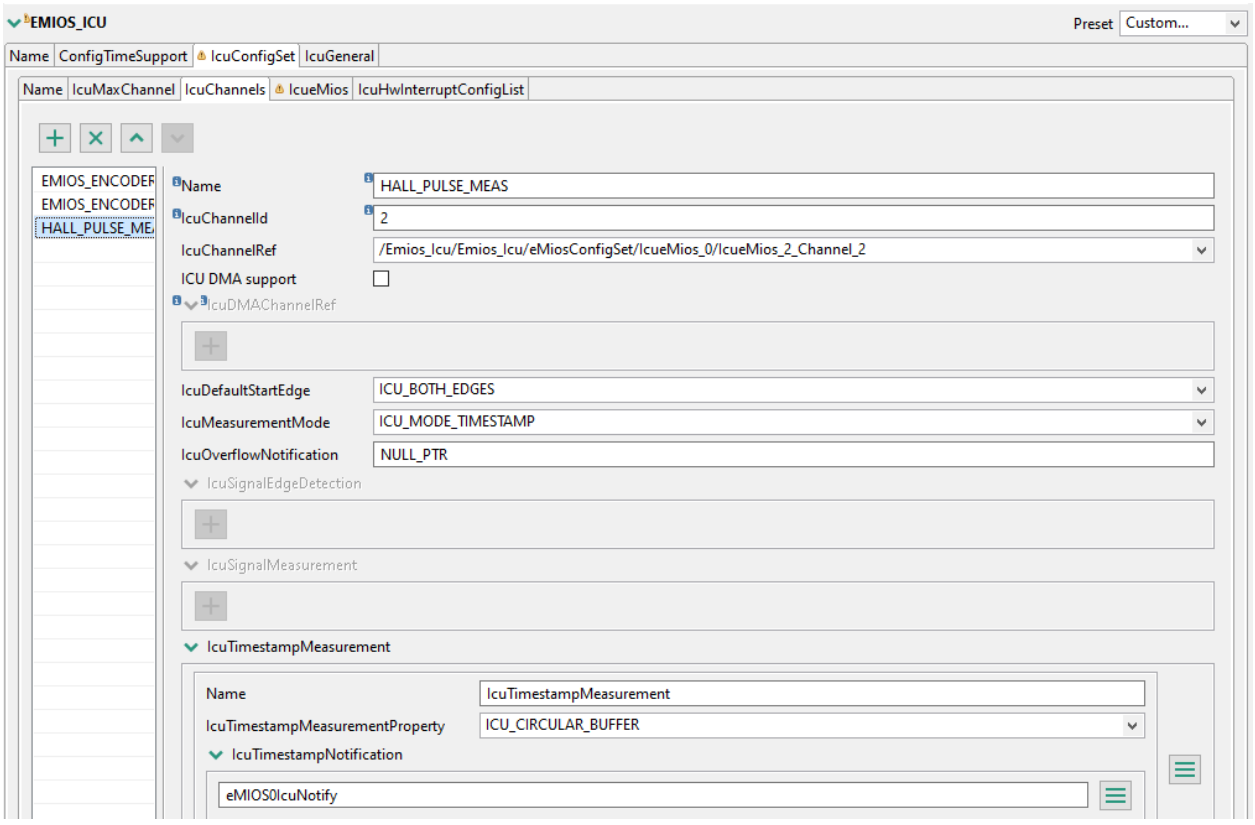


Figure 72. General ICU configuration

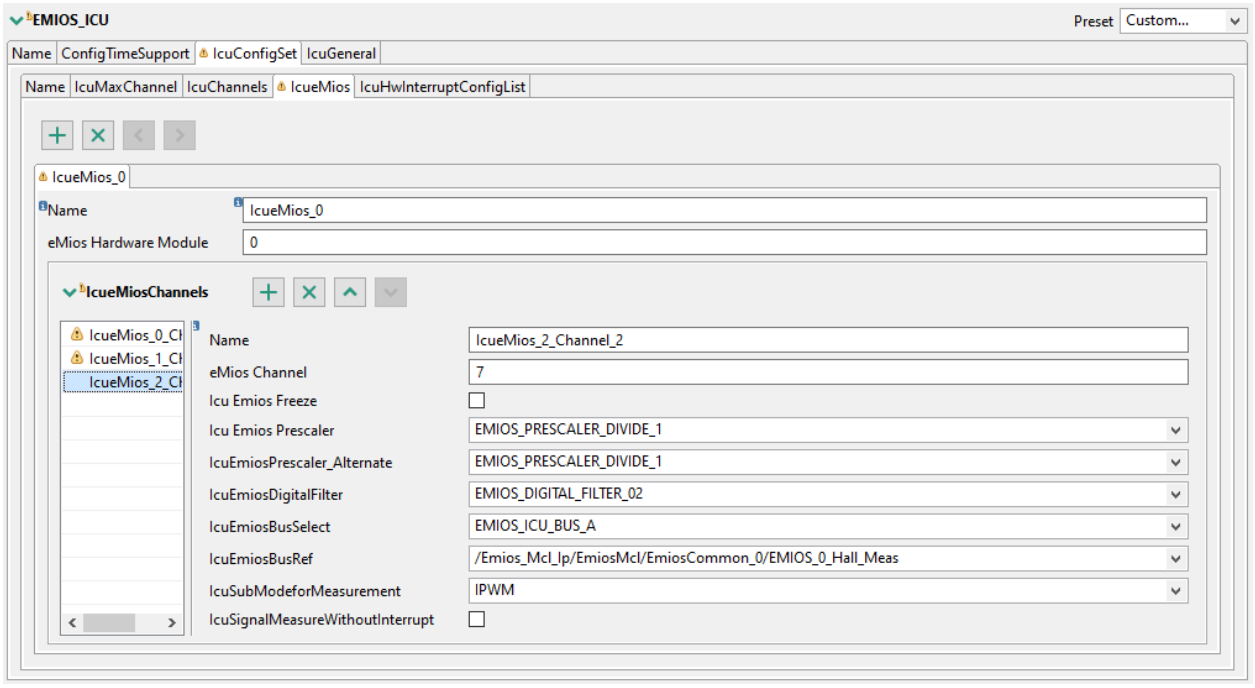


Figure 73. eMIOS channels for input capture

Figure 74. Example: eMIOS API for FOC hall sensor signal processing

```
void main (void)
{
...

    /*******************************************************************************************************
    * eMios Driver
    *******************************************************************************************************/
    Emios_Icu_Ip_Init(0U, &eMios_Icu_Ip_0_Config_PB);
    Emios_Icu_Ip_StartTimestamp(0U, 7U, &IcuTimeStampBuffer, 1U, 1U);
    Emios_Icu_Ip_EnableNotification(0U, 7U);
...
}
void eMIOS0IcuNotify(void)
{
...
   if(IcuTimeStampBuffer > old_IcuTimeStampBuffer)
   {
   SensorHall.Period = IcuTimeStampBuffer - old_IcuTimeStampBuffer;
   }
   else
   {
      SensorHall.Period = 0xFFFF - old_IcuTimeStampBuffer + IcuTimeStampBuffer;
   }
   old_IcuTimeStampBuffer = IcuTimeStampBuffer;
...
}
```

Note: *Various input pins or TRGMUX output can be selected for eMIOS input. This selection is realized in SIUL2 IMCR register.*

4.3 Software architecture

4.3.1 Introduction

This section describes the software design of the Sensorless PMSM Field Oriented Control framework application. The application overview and description of software implementation are provided. The aim of this section is to help in understanding of the designed software.

4.3.2 Application data flow overview

The application software is an interrupt driven running in real time. There are periodic interrupt service routines, which one of them is periodic and is associated with the ADC conversion complete interrupt. This interrupt service routine is executing all motor control tasks. Then, there is asynchronous interrupt service routine associated with the eMIOS input capture functionality, which processes FOC hall sensor task where speed and position are calculated. The most important interrupt routine is solely ADC conversion interrupt. This includes both fast current and slow speed loop control. All tasks are performed in an order described by the application state machine shown in Figure 77, and application flowcharts shown in Figure 75 and Figure 76.

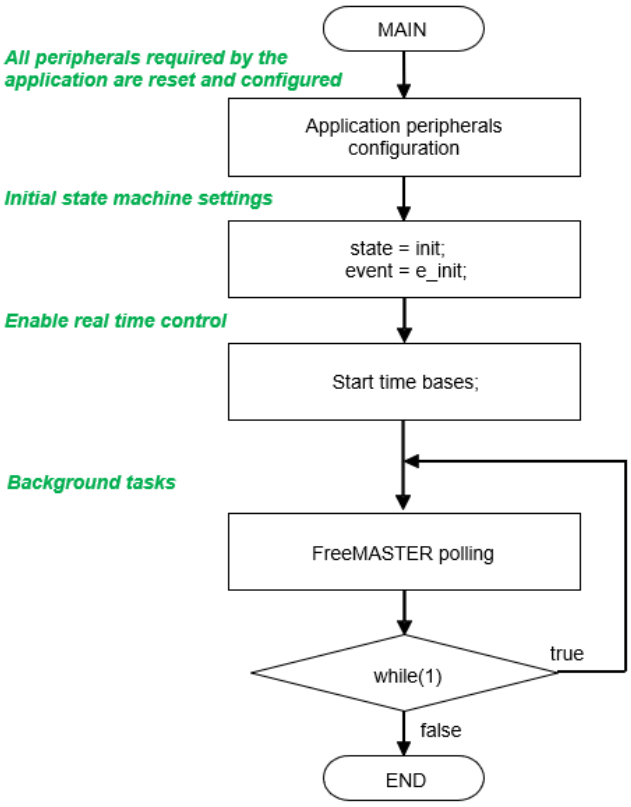


Figure 75. Flow chart diagram of main function with background loop.

To achieve precise and deterministic sampling of analog quantities and to execute all necessary motor control calculations, state machine functions are called within a periodic notification function. Hence, in order to actually call state machine functions, the peripheral causing this periodic interrupt must be properly configured and the interrupt enabled. As described in section 4.2 all peripherals are initially configured and all interrupts are enabled after a reset of the device. As soon as all S32K344 peripherals are correctly configured, the state machine functions are called from the BCTU notification function. The background loop handles non-critical timing tasks, such as the FreeMASTER communication polling.

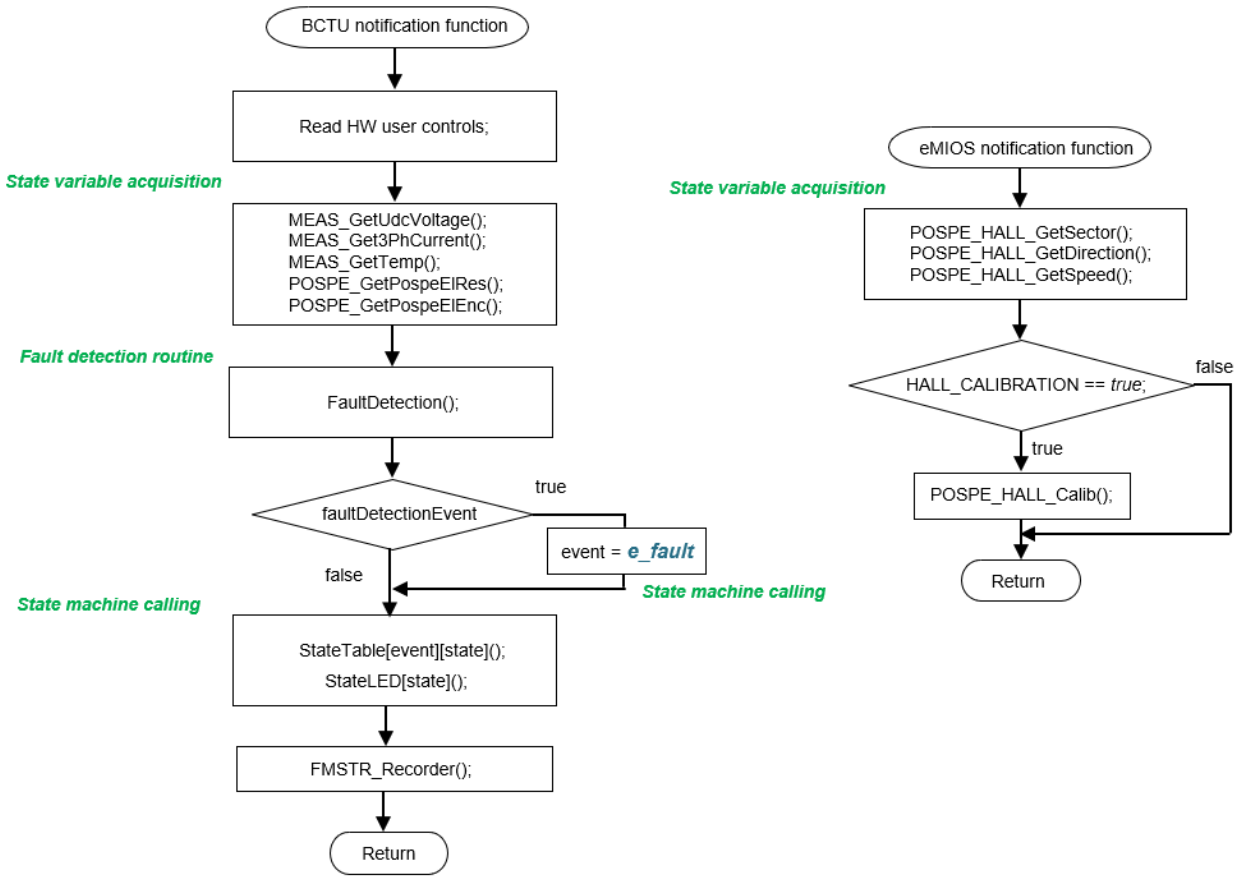


Figure 76. Flow chart diagram of periodic interrupts notification functions.

4.3.3 State machine

The application state machine is implemented using a two-dimensional array of pointers to the functions using variable called *StateTable[][]*. 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 invokes a function call whenever *StateTable[][]()* is called.

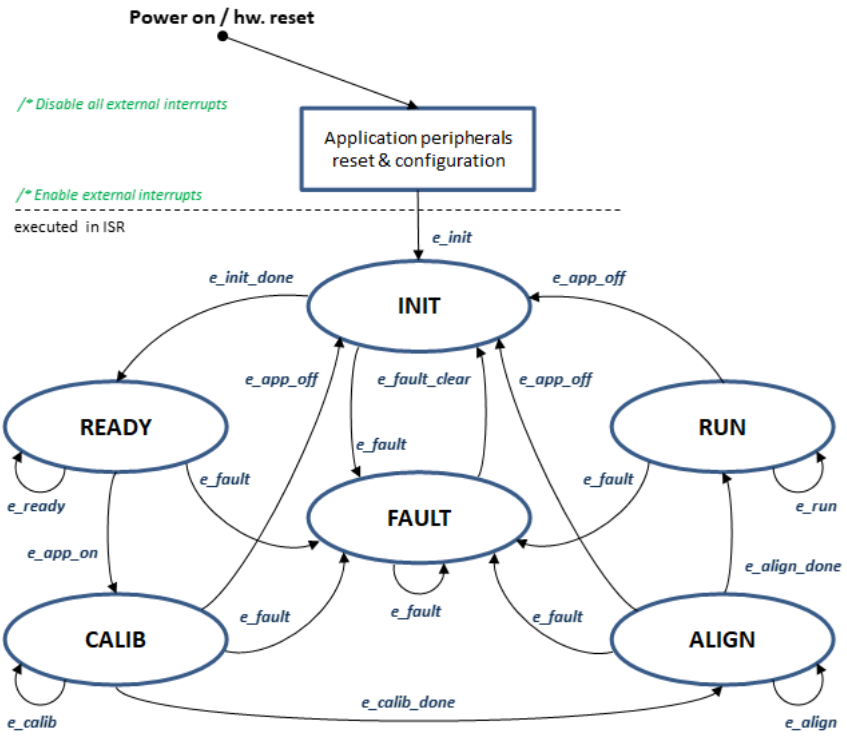


Figure 77. Application state machine

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

AppStates:

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

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

AppEvents:

- e_fault - event = 0
- e_fault_clear - event = 1
- e_init - event = 2
- e_init_done - event = 3
- e_ready - event = 4
- e_app_on - event = 5
- e_calib - event = 6
- e_calib_done - event = 7
- e_align - event = 8
- e_align_done - event = 9
- e_run - event = 10

- e_app_off - event = 11

4.3.3.1 State – FAULT

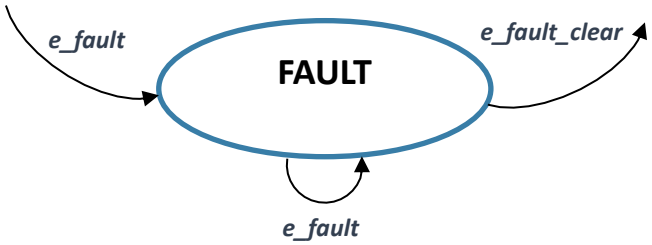


Figure 78. FAULT state with transitions

The application goes immediately to this state when a fault is detected. The system allows all states to pass into the FAULT state by setting *cntrState.event* = *e_fault*. State FAULT is a state that transitions back to itself if the fault is still 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 warning register *tempFaults* represents the current state of the fault pin/variable to warn the user that the system is getting close to its critical operation. And the fault register *permFaults* represents a fault flag, which is set and put the application immediately to fault state. Even if fault source disappears, the 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) or by simultaneously pressing the user buttons (SW4 and SW5) on the adapter board (blue board) of the NXP 48V development platform. 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 (see the following example).

Figure 79. Example: Fault clearing sequence

```
void StateFault(void)
{
...
    if (cntrState.usrControl.switchFaultClear)
       {
           // Clear permanent and temporary SW faults
           permFaults.mcu.R = 0; // Clear mcu faults
           permFaults.motor2.R = 0; // Clear motor faults
           permFaults.stateMachine.R = 0; // Clear state machine faults

           Hall_Seq_OK = 1;

           // When all Faults cleared prepare for transition to next state.
           cntrState.usrControl.readFault = true;
           cntrState.usrControl.switchFaultClear = false;
           cntrState.event = e_fault_clear;
        }
}
```

Setting event to *cntrState.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 76, 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 *permFaults* and *tempFaults* variables will be set. As a consequence of *permFaults* not equal to zero, function *faultDetection()* will modify the application event from *e_fault_clear back* to *e_fault*, which means jump to fault 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 (*permFaults* = 0x0) and application variable *switchFaultClear* is manually set to true.

The application is scanning for following system warnings and errors:

- Battery over voltage
- Battery under voltage
- DC bus over voltage
- DC bus under voltage
- Forward over voltage
- Forward under voltage
- Phase A/B /C over-current

The thresholds for fault detection can be modified in MCAT tool of FreeMASTER and changes are propagated on embedded side in PMSM_appconfig.h file generated by MCAT. Please see [9] for further information on how to set these thresholds using the MCAT. In addition to previous thresholds, fault state is entered if following errors are detected:

- BCTU trigger faults.
- FOC Error (irrelevant event call in state machine or eBEMF observer failure).

4.3.3.2 State – INIT

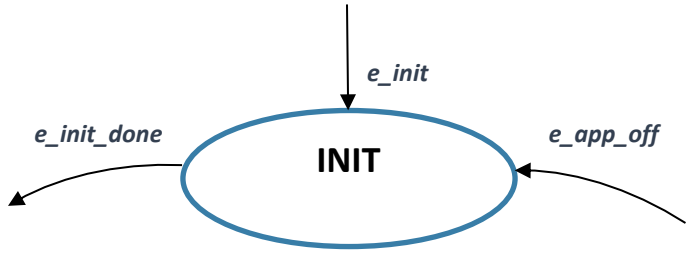


Figure 80. INIT state with transitions

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

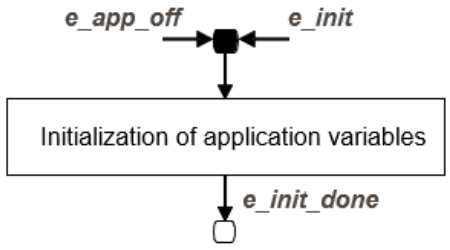


Figure 81. Flow chart of state INIT

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

4.3.3.3 State – READY

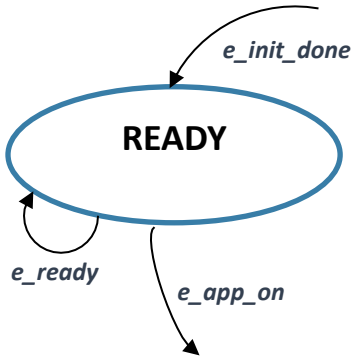


Figure 82. READY state with transitions

In READY state, application is waiting for user command to start the motor. The application is released from waiting mode by pressing the on board button SW4, SW5 or on falling edge of the SW6 or by FreeMASTER interface setting the variable *switchAppOnOff* = *true* (see flow chart in Figure 83).

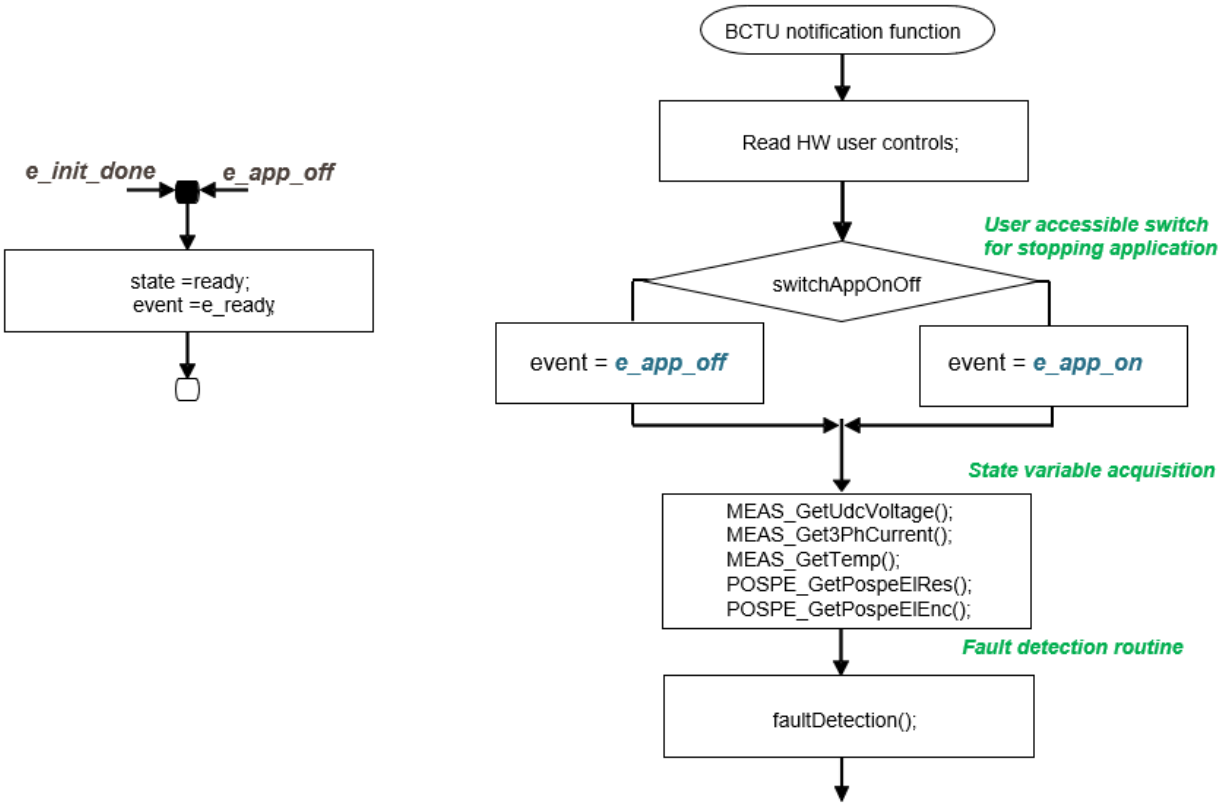


Figure 83. Flow chart of state READY

4.3.3.4 State – CALIB

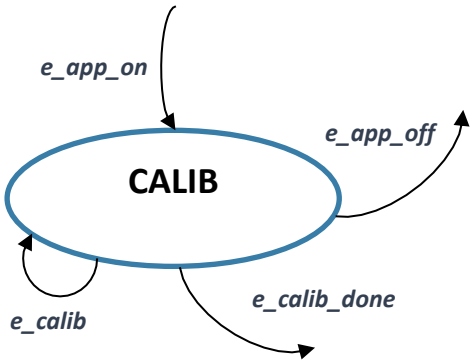


Figure 84. CALIB state with transitions

In this state, ADC DC offset calibration, DC-link capacitors pre-charging, gate drivers enabling and resolver calibration are performed. Once the state machine enters CALIB state, HW reset signals are cleared and DC-link voltage level is checked if pre-charge of DC-link can be performed. Enabling of gate driver is performed once auxiliary timeout is elapsed, which means DC-link is finally pre-charged (see flow chart in Figure 85). Then all PWM output are enabled. Calibration of the ADC DC offset is performed by generating 50% duty-cycle on the PWM outputs, and taking several measurements of the ADC0, ADC1 and ADC2 channels connected to the current sensors. These measurements are then averaged, and the average value for the channel is stored. This value will be subtracted from the measured value in RUN state. This calibration procedure removes DC offset 2.5V from the measured signal caused by conditional circuity and allows bidirectional current measurement. Resolver sensor calibration is performed in similar way as the current measurement calibration. Certain offset value on ADC channels, represented sine and cosine signals, are captured and subtracted from the measured values in RUN state. In case of successful calibration of resolver sensor, excitation of resolver sensor can be performed. State CALIB is a state that allows transition back to itself, provided no faults are present, the user does not request stop of the application (by *switchAppOnOff=true*), and the calibration process has not finished. The number of samples for averaging is set by macro FILTER_SAMPLE_NO_MEAS where actual number of samples is *u16CalibSamples * 5*. After all samples have been taken and the averaged values successfully saved, the application event is automatically set to *cntrState.event=e_calib_done* and state machine can proceed to state ALIGN (see flow chart in Figure 85).

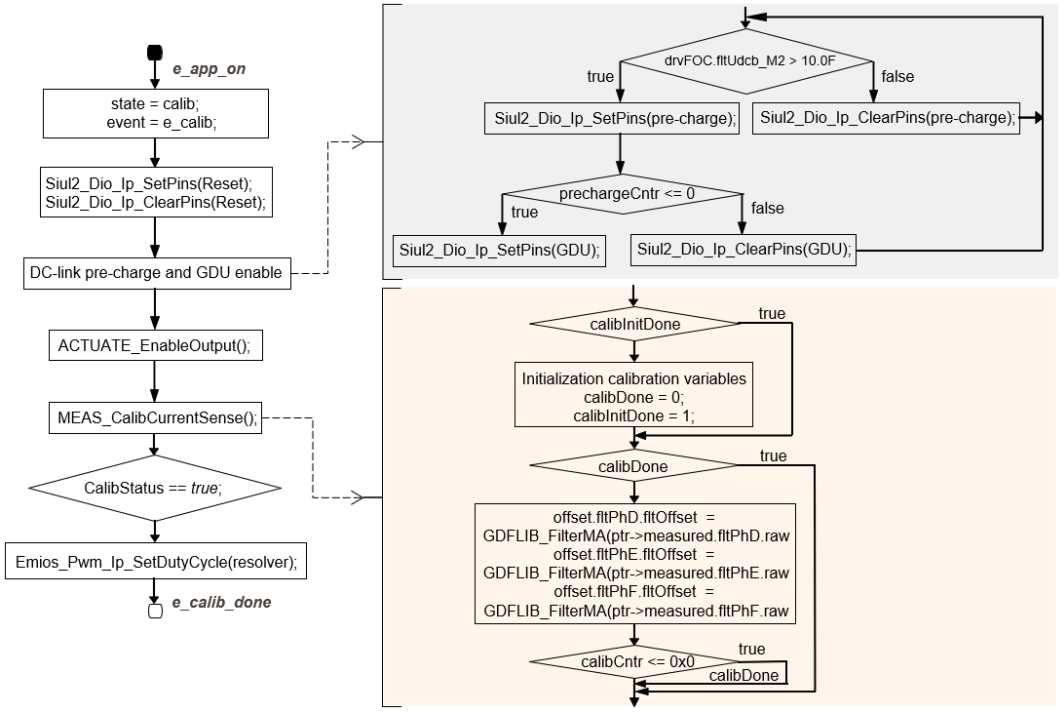


Figure 85. Flow chart of state CALIB

A transition to FAULT state is performed automatically when a fault occurs. A transition to INIT state is performed by setting the event to *cntrState.event=e_app_off*, which is done automatically on falling edge of *switchAppOnOff=false* using FreeMASTER.

4.3.3.5 State – ALIGN

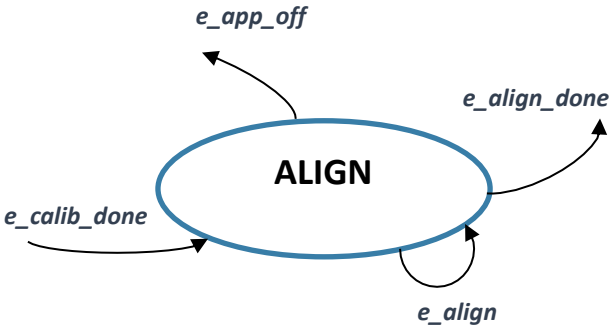


Figure 86. ALIGN state with transitions

This state manages alignment of the rotor and stator flux vectors to mark zero position. When using a model based approach for position estimation, the zero position is not known. The zero position is obtained at ALIGN state, where 2 state alignment is used to avoid rotor to be stuck at 180deg. A DC voltage is applied to q-axis voltage for a certain period and after that to d-axis voltage for the rest of the alignment time. Ratio between d and q axis alignment time is given by macro ALIGN_D_FACTOR. 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. To get rotor stabilized at aligned position, a certain time is selected for alignment process. This time and the amplitude of DC voltage used for alignment can be modified by MCAT tool. Timing is implemented using a software counter that counts from a pre-defined value down to zero.

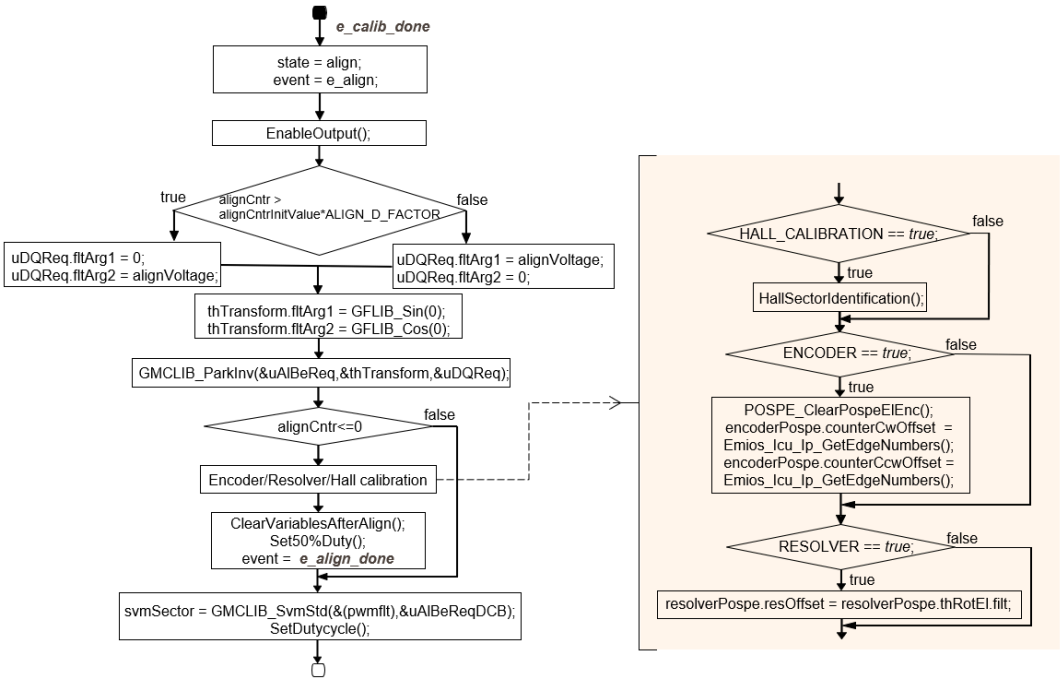


Figure 87. Flow chart of state ALIGN

If encoder sensor is used in application (ENCODER=*true*), calibration routine is executed. Offset value of rotor position is captured during this calibration and subtracted in RUN state. If hall sensor case is used and enabled together with calibration routine (HALL_CALIBRATION macro is set to *true*) identification of the physical order of hall sensors is performed. During this time, the event remains set to *cntrState.event=e_align*. When the counter reaches zero, the counter is reset back to the pre-defined value, and event is automatically set to *cntrState.event=e_align_done*. This enables a transition to RUN state see flow chart in Figure 87. A transition to FAULT state is performed automatically when a fault occurs. Transition to INIT state is performed by setting the event to *cntrState.event=e_app_off*, which is done automatically on falling edge of *switchAppOnOff=false* using FreeMASTER or on rising edge of the SW6 on adapter board (blue board) of NXP 48V development platform.

4.3.3.6 State – RUN

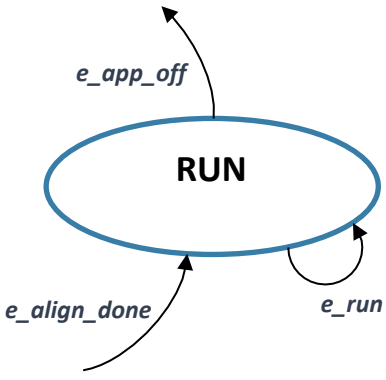


Figure 88. RUN state with transitions

In this state, the FOC algorithm is calculated, as described in section 3.

The control is designed such that the drive might be operated in six position modes depending on the source of the position information:

1. **Force mode**: The FOC control is based on the generated position (so called open loop position), also this position is supplied to eBEMF observer in order to initialize its state.
2. **Tracking mode**: The FOC control is still using the open loop position, however, the eBEMF observer is left on its own, meaning that the observer is using its own estimated position and speed one calculation step delayed.
3. **Sensorless mode**: FOC control use estimated position and speed from eBEMF observer.
4. **Resolver mode**: FOC control uses position and speed obtained from resolver sensor. This mode is available only if RESOLVER macro is set to *true*.
5. **Encoder mode**: FOC control uses position and speed obtained from encoder sensor. This mode is available only if ENCODER macro is set to *true*.
6. **Hall mode**: FOC control uses position and speed obtained from hall sensors. This mode is available only if HALL macro is set to *true* and HALL_CALIBRATION macro is set to *false*.

Position mode can be controlled by pos_mode variable in FreeMASTER interface. It might be modified manually or automatically depending on the state of the variable *cntrState.usrControl.controlMode*. If *cntrState.usrControl.controlMode* = *automatic* and *switchSensor* = *Sensorless*, application automatically transits from Force mode (open loop mode) to Sensorless mode (closed loop mode) through Tracking mode based on the actual rotor speed and speed limits defined for each position mode (see section 3.4). Variable *switchSensor* defines whether position/speed feedback comes from eBEMF Observer or encoder sensor or resolver sensor or hall sensor. If *switchSensor* = *Encoder*, the application uses encoder mode only. If *switchSensor* = *Resolver*, the application uses resolver mode only. If *switchSensor* = *Hall*, the application uses hall mode only. The switchSensor is automatically set to Sensorless, if any of the sensors are not present (ENCODER=*false*, RESOLVER=*false*, HALL=*false*).

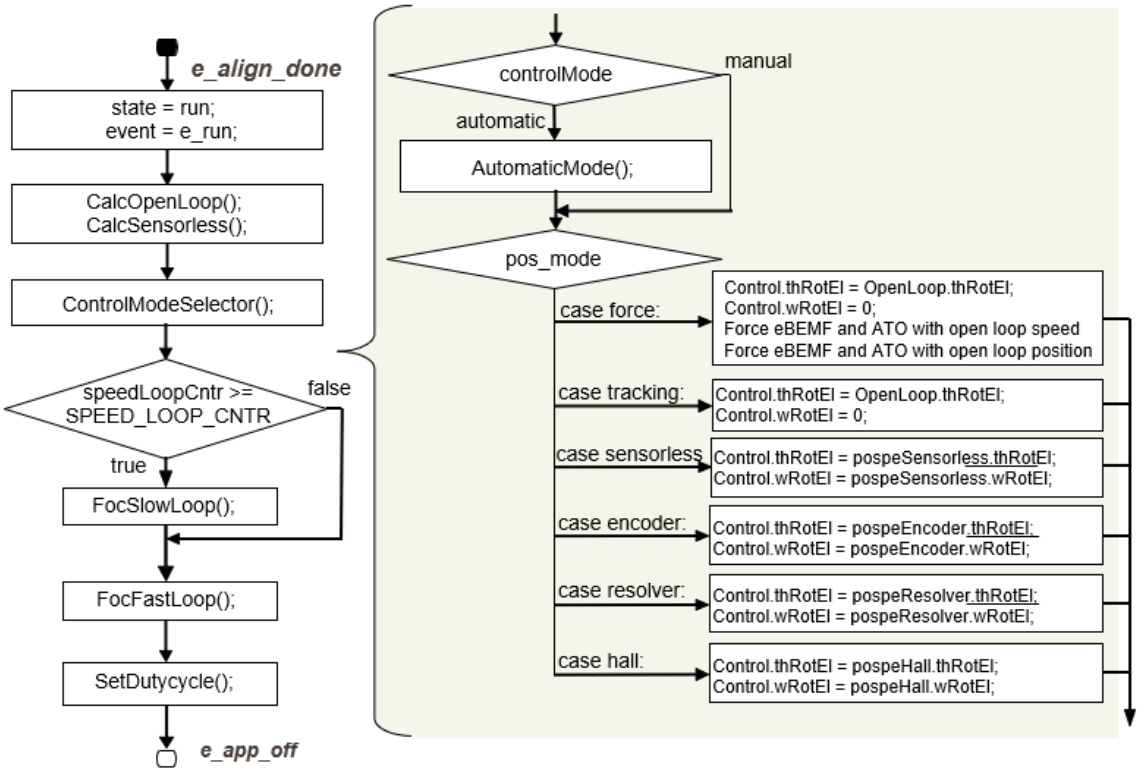


Figure 89. Flow chart of state RUN

Calculation of fast current loop is executed every BCTU interrupt, while calculation of slow speed loop is executed every Nth BCTU 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. N value

(macro SPEED_LOOP_CNTR) is automatically calculated by MCAT from current loop sample time and speed loop sample time parameters. 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 (see flow chart in Figure 89).

Figure 90 shows implementation of FOC algorithm, used functions and variables. As can be seen from the diagram, rotor position and speed are estimated by eBEMF observer. This is a default rotor position and speed feedback for FOC. To run encoder based FOC, ENCODER macro must be set to *true* and PM motor must be equipped with encoder sensor. As mentioned previously, encoder based FOC can be activated/deactivated by setting *switchSensor* variable to *encoder/sensorless*. To run resolver based FOC, RESOLVER macro must be set to *true* and PM motor must be equipped with resolver sensor. As mentioned previously, resolver based FOC can be activated/deactivated by setting *switchSensor* variable to *resolver/sensorless*. To run hall based FOC, HALL macro must be set to *true*, HALL_CALIBRATION macro must be set to *false* and PM motor must be equipped with hall sensors. As mentioned previously, hall based FOC can be activated/deactivated by setting *switchSensor* variable to *hall/sensorless*. Application allows to use only one sensor at the same time.

If HALL_CALIBRATION macro is set to *true* hall sensor calibration routine continues execution also in RUN state. Identification of precise physical hall sensors position is performed by means of sensorless operation of the motor. Therefore, tuning of sensorless algorithm is crucial for hall sensor calibration routine. PM motor with hall sensors are spined almost at nominal speed at certain time. Algorithm is calculating precise physical position on hall sensors by comparing position of sensorless operation and initial position of sensors from sector identification routine. Calibration function is called during custom notification function *eMIOS0IcuNotify*. This procedure must be performed for both directions of motor rotation. Whole calibration routine is performed almost automatically and once it is done, the macro HALL_CALIBRATION must be set to *false*.

A transition from RUN state to FAULT state is performed automatically when a fault occurs. A transition to INIT state is performed by setting the event to *cntrState.event=e_app_off*, which is done automatically on falling edge of *switchAppOnOff=false* using FreeMASTER or on rising edge of the SW6 on adapter board (blue board) pressed.

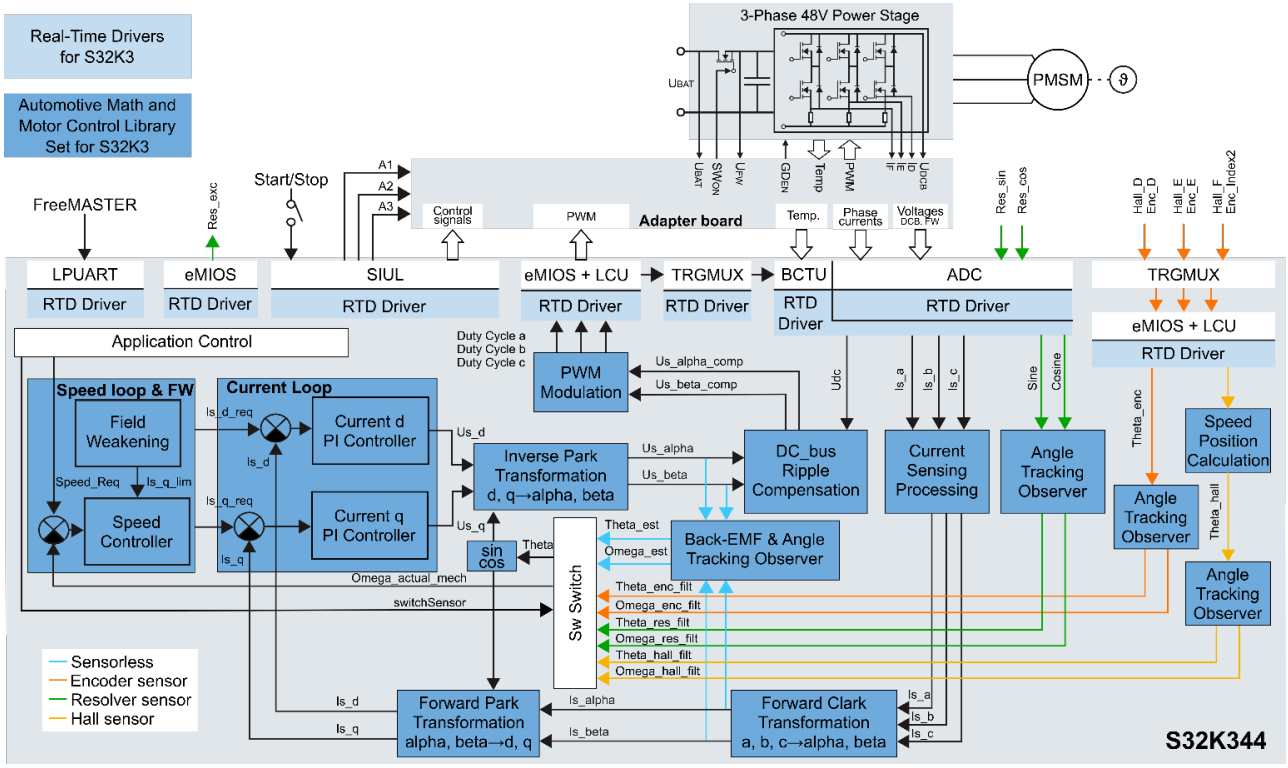


Figure 90. Sensorless and sensor-based FOC with FW implementation on S32K344

4.3.4 AMMCLib integration

Application software of the FOC sensorless control with field weakening is built using NXP's Automotive Math and Motor Control Library set (AMMCLib), a precompiled, highly speed-optimized off-the-shelf software library designed for motor control applications. The most essential blocks of the FOC structure are presented in Figure 90. AMMCLib supports all available data type implementations: 32-bit fixed-point, 16-bit fixed-point and single precision floating-point. In order to achieve high performance of the S32K344 core, floating point arithmetic is used as a reference for this motor control application.

Current Loop function AMCLIB_CurrentLoop unites and optimizes most inner loop of the FOC cascade structure Figure 90. It consists of two PI controllers and basic mathematical operations which calculate errors between required and feedback currents and limits for PI controllers based on the actual value of the DC bus voltage. All functions and data structures are presented in Figure 91.

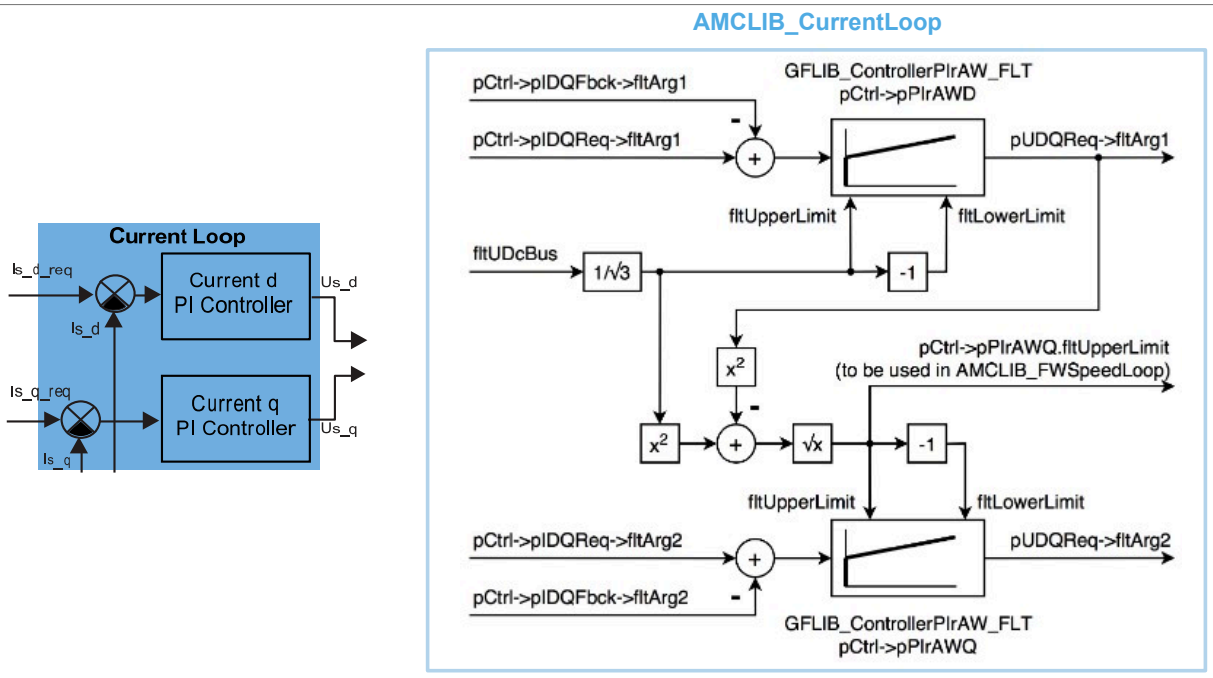


Figure 91. Functions and data structures in AMCLIB_CurrentLoop

Required d- and q-axis stator currents can be either manually modified or generated by outer loop of the cascade structure consisting of: Speed Loop and Field Weakening (FW) as shown in Figure 90. To achieve highly optimized level, AMCLIB_FWSpeedLoop merges two functions of the AMMCLIB, namely speed control loop AMCLIB_SpeedLoop and field weakening control AMCLIB_FW, Figure 92. AMCLIB_SpeedLoop consists of speed PI controller GFLIB_ControllerPIpAW, speed ramp GFLIB_Ramp placed in feedforward path and exponential moving average filter GFLIB_FilterMA placed in the speed feedback. AMCLIB_FW function is NXP's patented algorithm (US Patent No. US 2011/0050152 A1) that extends the speed range of PMSM beyond the base speed by reducing the stator magnetic flux linkage as discussed in section 3.5. All functions and data structures used in AMCLIB_FW function are shown in Figure 92.

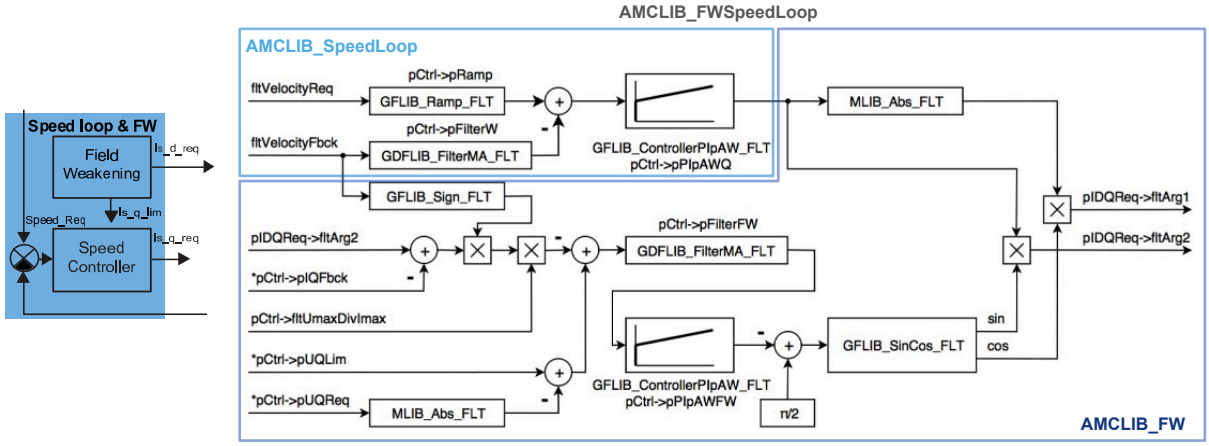


Figure 92. Functions and data structures in AMCLIB_FWSpeedLoop

AMCLIB_FW key advantages:

- Fully utilize the drive capabilities (speed range, load torque).
- Reduces stator linkage flux only when necessary.
- Supports four quadrant operations.
- The algorithm is very robust - as a result, the PMSM behaves as a separately excited wound field synchronous motor drive.
- Allows maximum torque optimal control.

eBEMF observer AMCLIB_BemfObsrv and Angle tracking observer AMCLIB_TrackObsrv constitute important blocks in sensorless mode, Figure 90. They estimate rotor position and speed based on the inputs, namely stator voltages $u_{\alpha\beta}$ and currents $i_{\alpha\beta}$, Figure 93. AMCLIB_BemfObsrv transforms inputs quantities from stationary reference frame α/β to quasi-synchronous reference frame γ/δ that follows the real synchronous rotor flux frame d/q with an error θ_{err}. AMCLIB_BemfObsrv algorithm is based on the mathematical model of the PMSM motor with excluded BEMF terms $e_{\gamma\delta}$. BEMF terms are estimated as disturbances in this model, generated by PI controllers. The estimated BEMF values are used for calculating the phase error θ_{err}, which is provided as an output of the BEMF observer.

To align both frames and provide accurate estimates, this phase error θ_{err} must be driven to zero. This is a main role of the Angle tracking observer AMCLIB_TrackObsrv which is attached to function of the eBEMF observer AMCLIB_BemfObsrv, Figure 93. AMCLIB_TrackObsrv is an adopted phase-locked-loop algorithm that estimates rotor speed and position keeping $\theta_{err} = 0$. This is ensured by a loop compensator that is PI controller. While PI controller generates estimated rotor speed, integrator used in this phase-locked-loop algorithm serves estimated rotor position.

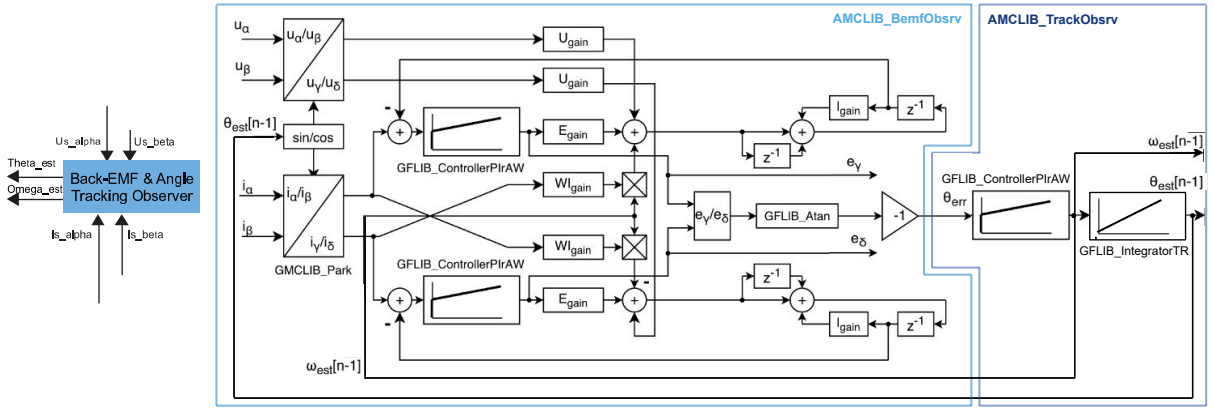


Figure 93. Structure of the AMCLIB_BemfObsrv and AMCLIB_TrackObsrv

More details related to AMMCLib FOC functions can be found in S32K34x AMMCLib User's Guide on standard installation path C:\NXP\AMMCLIB\S32K3xx_AMMCLIB_vX.Y.Z\doc\S32K3XXMCLUG.pdf. Parameters of the PI controllers placed in the speed control loop, current control loop, eBEMF and Angle tracking observer can be tuned by using NXP's Motor Control Application Tuning tool (MCAT). Detailed instructions on how to tune parameters of the FOC structure by MCAT are presented in [10], [11].

4.3.5 MCAT integration

MCAT (Motor Control Application Tuning) is a graphical tool dedicated to motor control developers and the operators of modern electrical drives. The main feature of proposed approach is automatic calculation and real-time tuning of selected control structure parameters. Connecting and tuning new electric drive setup becomes easier because the MCAT tool offers a possibility to split the control structure and consequently to control the motor at various levels of cascade control structure.

The MCAT tool runs under FreeMASTER online monitor, which allows the real-time tuning of the motor control application. Respecting the parameters of the controlled drive, the correct values of control structure parameters are calculated, which can be directly updated to the application or stored in an application static configuration file. The electrical subsystems are modeled using physical laws and parameters of the PI controllers are determined using Pole-placement method. FreeMASTER MCAT control and tuning is described in the section *5*.

The MCAT tool generates a set of constants to the dedicated header file (for example "{Project Location}\src \config\PMSM_appconfig.h"). The names of the constants can be redefined within the MCAT configuration file "Header_file_constant_list.xml" ("{Project Location}\FreeMASTER_control\ MCAT\src\xml_files\"). The PMSM_appconfig.h contains application scales, fault triggers, control loops parameters, speed sensor and/or observer settings and FreeMASTER scales. The PMSM_appconfig.h should be linked to the project and the constants should be used for the variables initialization.

The FreeMASTER enables an online tuning of the control variables using MCAT control and tuning view. However, the FreeMASTER must be aware of the used control-loop variables. A set of the names is stored in "FM_params_list.xml" ("{Project Location}\FreeMASTER_control\MCAT\src\xml_files\").

5 FreeMASTER and MCAT user interface

The FreeMASTER debugging tool is used to control the application and monitor variables during run time. Communication with the host PC passes via USB. However, because FreeMASTER supports serial port communication, there must be a driver for the physical USB interface, OpenSDA, installed on the host PC that creates a virtual COM port from the USB. The driver shall be installed automatically plugging S32K344 controller board to USB port. The application configures the LPUART module of the S32K344 for a

communication speed of 115200bps. Therefore, the FreeMASTER user interface also needs to be configured respectively.

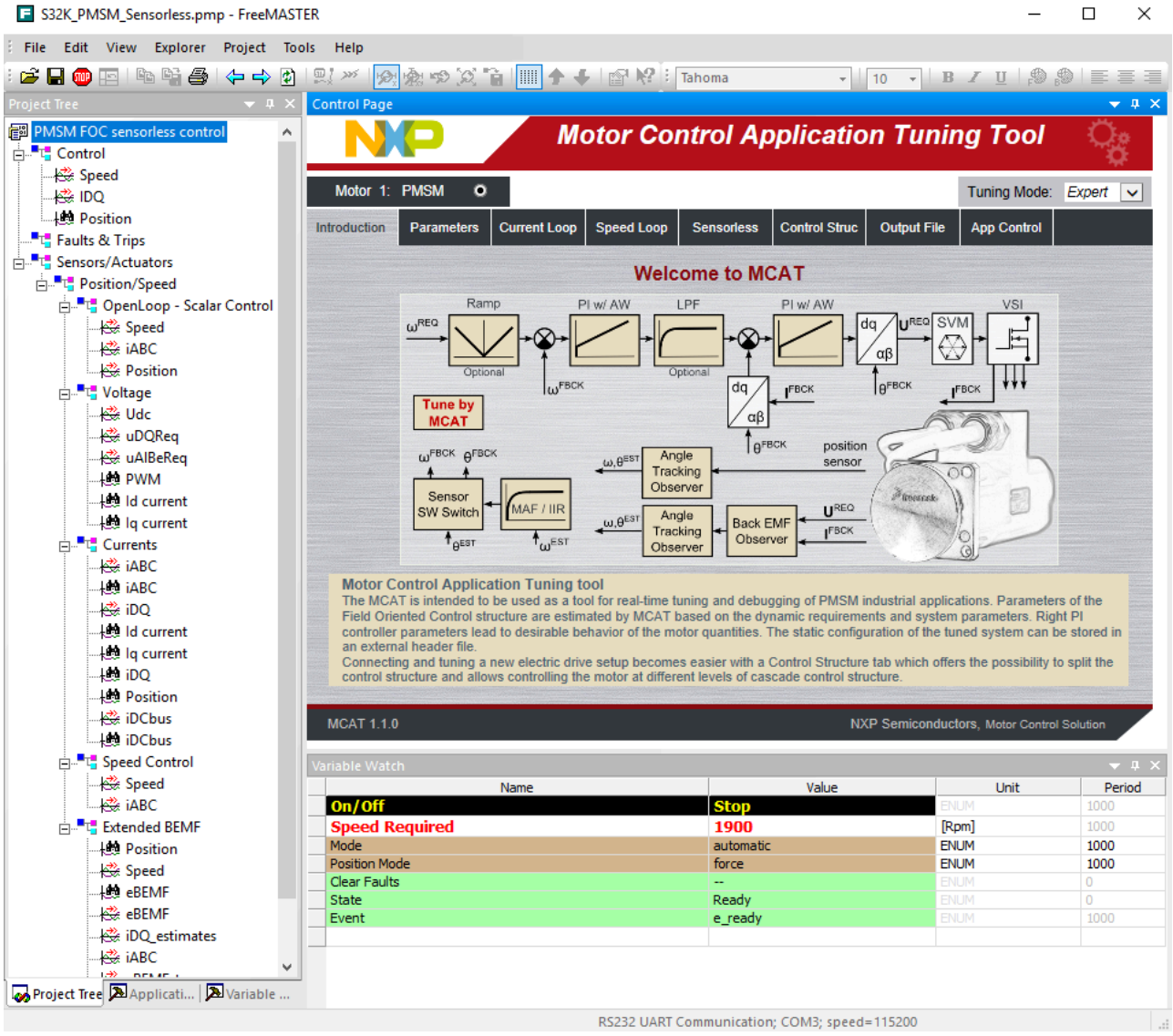


Figure 94. FreeMASTER and Motor Control Application Tunning Tool

5.1 MCAT settings and tuning

5.1.1 Application configuration and tuning

FreeMASTER and MCAT interface (Figure 94) enables online application tuning and control. The MCAT tuning shall be used before the very first run of the drive to generate the configuration header file (PMSM_appconfig.h). Most of the variables are accessible via MCAT online tuning (thus can be updated anytime). They are highlighted when mouse pointer is over the button “Update Target” (see Figure 95). Some parameters (especially the fault limit thresholds) must be set using the configuration header file generation, which can be done on the “Output File” panel by clicking the “Generate Configuration File” (see Figure 96).

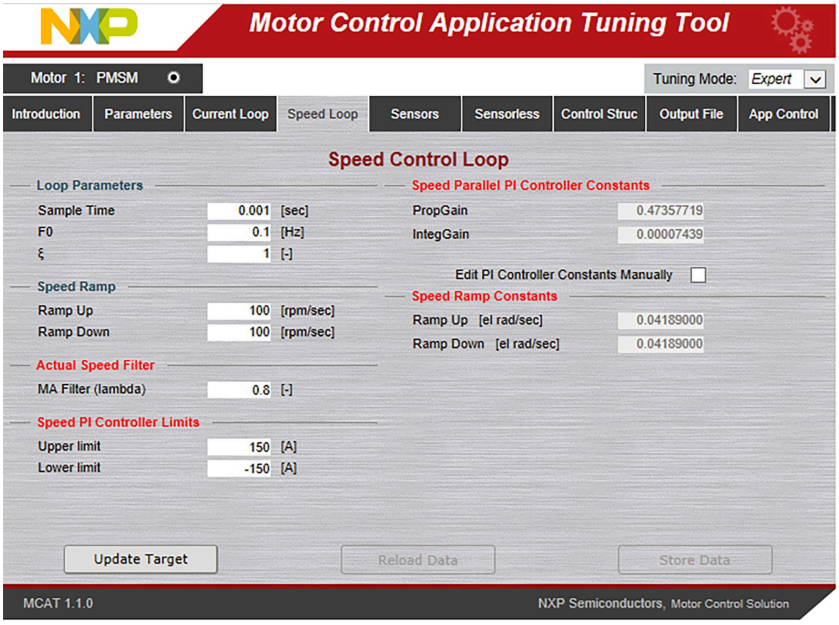


Figure 95. Parameters for online tunning

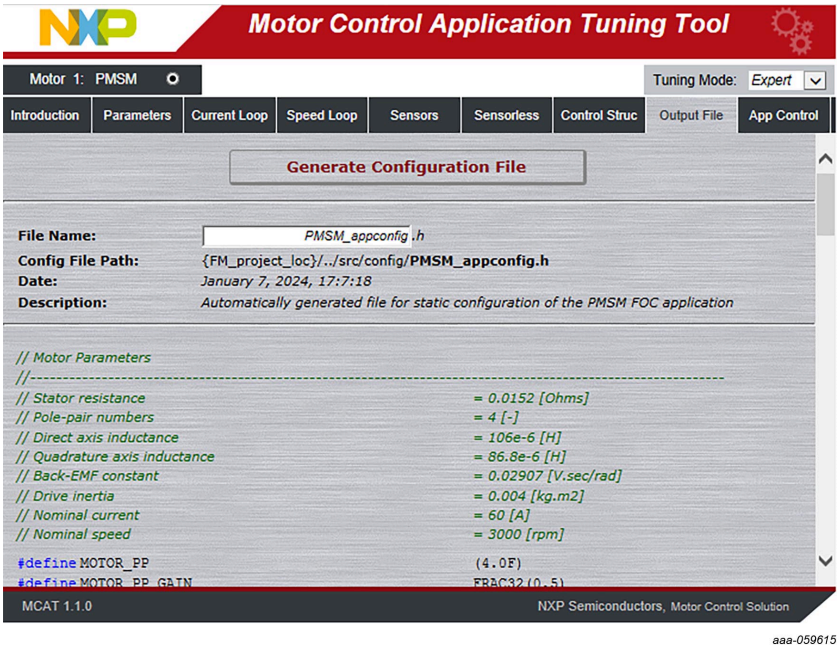


Figure 96. Output File panel and “Generate Configuration File” button

Parameters runtime update is done using the “Update Target” button (see Figure 97). Changes can be also saved using “Store Data” button, or reloaded to previously saved configuration using “Reload Data” button. Only stored configuration can be generated to PMSM_appconfig.h header file. File holding the configuration is “{Project Location}\FreeMASTER_control\ MCAT\param_files\M1_params.txt”. Settings for various motors, scenarios can be backed up and selected setting can be loaded by replacing the content of M1_params.txt.

Any change of parameters highlights the cells that have not been saved using “Store data”. Changes can be reverted using “Reload Data” to previously saved configuration. This button is disabled if no change has been made.

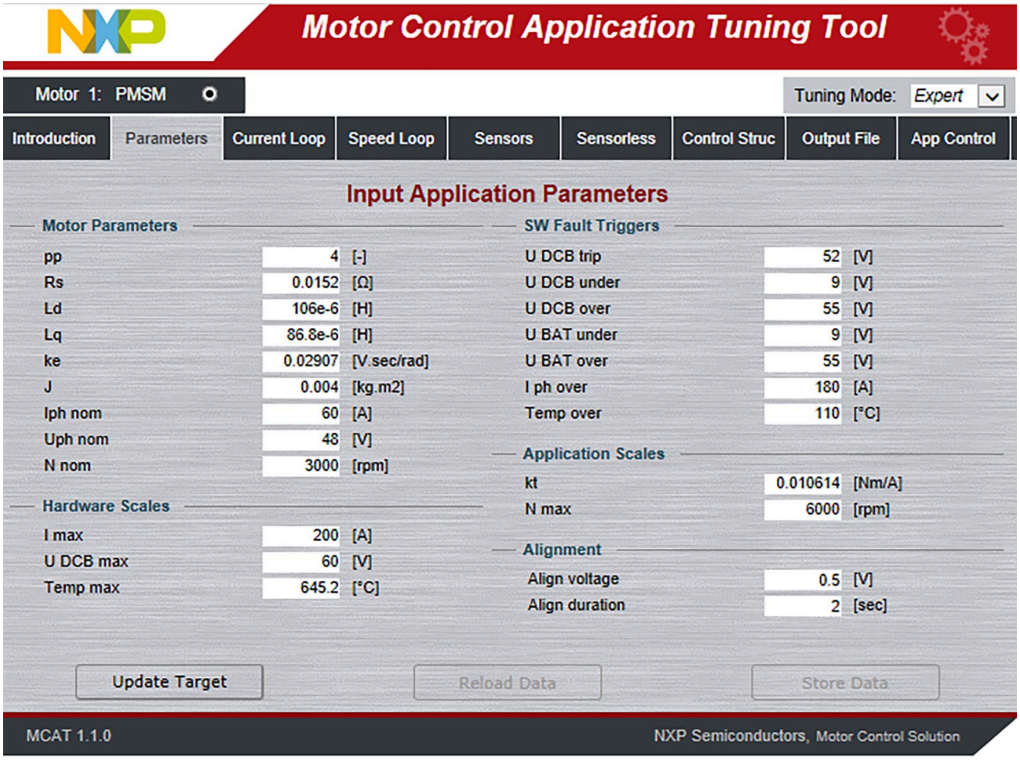
Note: *MCAT tool can be configured using hidden mouse-over "Settings" button (see Figure 94), where a set of advanced settings, for example PI controller types, speed sensors and other blocks of the control structure can be changed. However, it is not recommended to change these settings since it will force the MCAT to look for a different variables names and to generate different set of constants than the application is designed for. See MCAT tool documentation available at nxp.com.*

The application tuning is provided by a set of MCAT pages dedicated to every part of the control structure. An example of the Application Parameters Tuning page is in Figure 97. Following list of settings pages is based on the PMSM sensorless application.

• Parameters – Motor Parameters – Hardware Scales – SW Fault Triggers – Application Scales	• Speed Loop – Loop Parameters – Speed PI Controller Constants – Speed Ramp – Speed Ramp Constants – Actual Speed Filter – Speed PI Controller Limits
• Current loop – Loop Parameters – D axis PI Controller – Q axis PI Controller – Current PI Controller Limits	• Sensorless – BEMF Observer Parameters – BEMF DQ Observer Coefficients – Tracking Observer PI Constants – Tracking Observer Integrator – Open Loop Start-up Parameters – BEMF DQ Observer PI Controller Constants

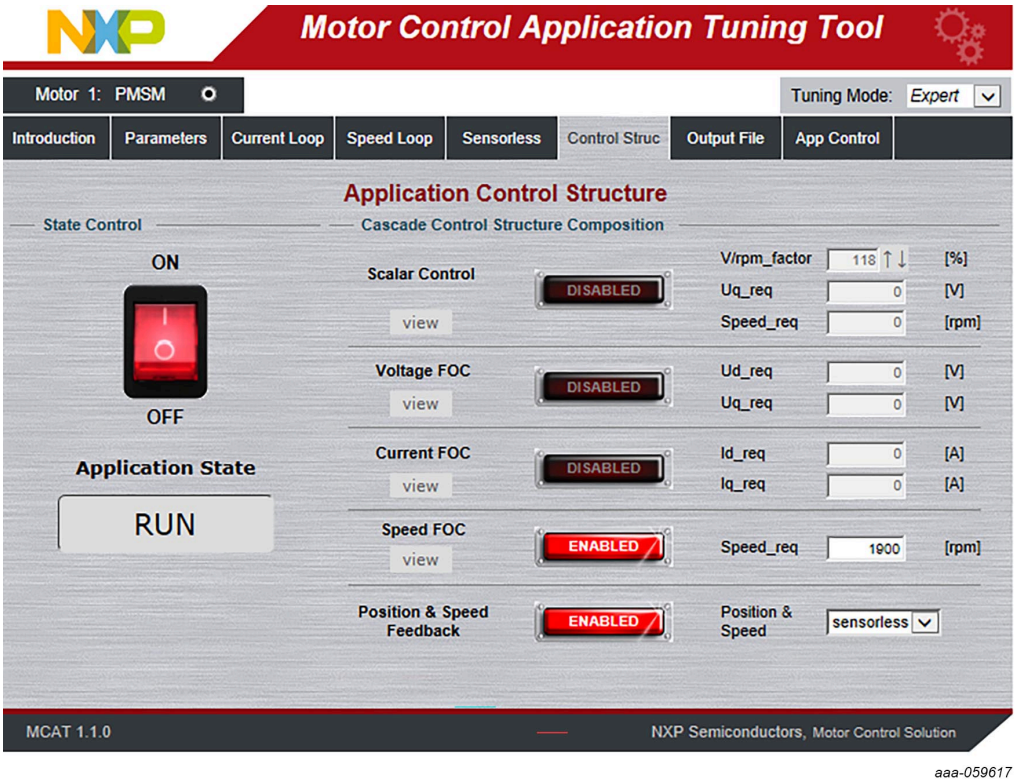
Changes can be tested using MCAT "Control Struc" page (Figure 98), where the following control structures can be enabled:

- Scalar Control
- Voltage FOC (Position & Speed Feedback is enabled automatically)
- Current FOC (Position & Speed Feedback is enabled automatically)
- Speed FOC (Position & Speed Feedback is enabled automatically)



aaa-059616

Figure 97. MCAT input application parameters page



aaa-059617

Figure 98. MCAT application control structure page

5.2 MCAT application control

All application state machine variables can be seen on the FreeMASTER MCAT App control page as shown in Figure 99. Warnings and faults are signaled by a highlighted red color bar with name of the fault source. The warnings are signaled by a round LED-like indicator, which is placed next to the bar with the name of the fault source. The status of any fault is signaled by highlighting respective indicators. In Figure 99, for example, there are two pending fault flags ("Ia and Ib" – overcurrent state) and one warning indicated ("Ia" – overcurrent state). That means that the measured phase current exceeds the limit set in the MCAT_Init function. The warning indicator is still on if the phase current is higher than the warning limit set in INIT state. In this case, the application state FAULT is selected, which is shown by a frame indicator hovering above FAULT state. After all actual fault sources have been removed, no warning indicators are highlighted, but the fault indicators will remain highlighted. The pending faults can now be cleared by pressing the "FAULT" button. This will clear all pending faults and will enable transition of the state machine into INIT and then READY state. 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 application On/Off switch. Successful selection is indicated by highlighting the On/Off button in green. Required speed can be set by clicking on speed gauge or by modifying FreeMASTER variable" Speed Required".

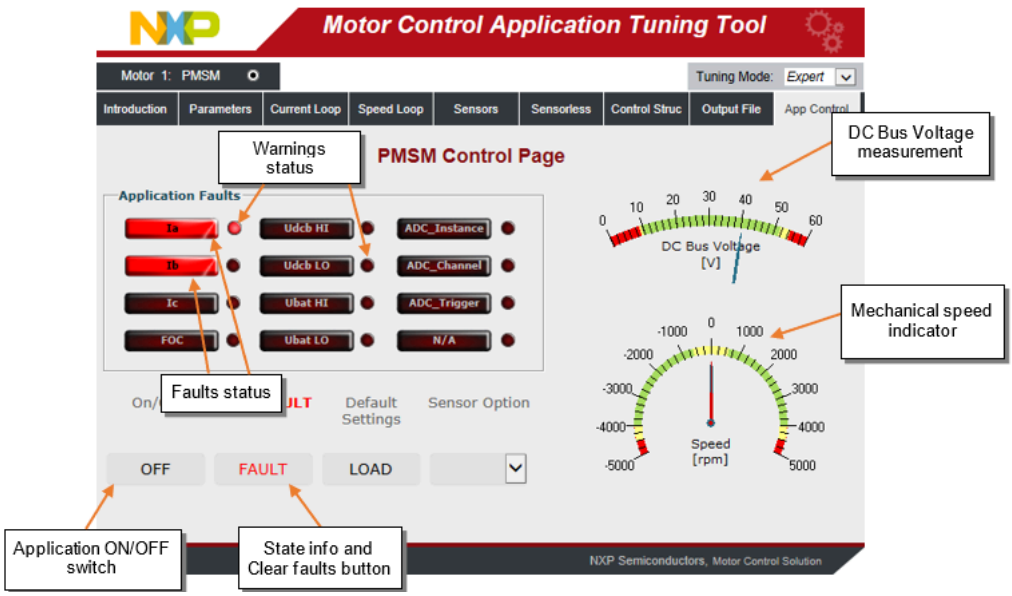


Figure 99. FreeMASTER MCAT Control Page for controlling the application

6 Conclusion

Design, described in this application note shows the simplicity and efficiency in using the S32K344 microcontroller for sensorless PMSM motor control and introduces it as an appropriate candidate for various applications in the automotive area. Moreover, support of encoder, resolver or hall position sensors allows to extend application usage among the use cases which are becoming more and more popular like Light Electric Vehicles (LEV). MCAT tool provides interactive online tool which makes the PMSM drive application tuning friendly and intuitive.

7 References

1. MCPUG: NXP 48 V Motor Control Platform
2. S32 Design Studio IDE for S32 Platform

3. Real-Time Drivers (RTD)
4. FreeMASTER Run-Time Debugging Tool
5. Automotive Math and Motor Control Library Set
6. S32K3xx MCU Family
7. S32K3xx MCU Family - Reference Manual
8. Rashid, M. H. Power Electronics Handbook, 2nd Edition. Academic Press
9. Motor Control Application Tuning (MCAT) Tool
10. AN4912 Tuning 3-Phase PMSM Sensorless Control Application Using MCAT Tool
11. AN4642 Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
12. AN4480 Permanent Magnet Synchronous Motor with Resolver, Vector Control, Driven by eTPU on MPC5500
13. AN4518 Dual 3-Phase PMSM Development Kit with MPC5643L
14. AN12017 3-Phase PMSM Development Kit with MPC5744P
15. AN14164 Current Sensing Techniques in Motor Control Applications

8 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9 Revision history

Table 7. Revision history

Document ID	Release date	Description
AN14313 v.2.0	05 June 2025	Updated the pinout information in the "S32K344 module interconnection" image to match the information in the "Pins assignment for S32K344 PMSM Sensorless/based FOC control" table
AN14313 v.1.0	19 March 2025	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Figures

Single 3-phase 48V PMSM Motor Control Kit with S32K344 using RTD low-level API (Featuring Motor Control Application Tuning (MCAT) tool)

Contents