# AN14429

## Dual 3-phase 48V PMSM Motor Control Kit with S32K344 using RTD low-level API

Rev. 2.0 — 5 June 2025 Application note

**Document information**

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

# 1 Introduction

This application note describes the design of a dual 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), are 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 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 sensor-less algorithm or encoder or resolver sensor
- Application control user interface using FreeMASTER debugging tool
- Motor Control Application Tuning (MCAT) tool

# 2 System concept

The system is designed to drive two 3-phase PMSMs independently. The application meets the following specifications:

- 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) a NXP’s complimentary integrated development environment (IDE) (see section [2], [3])
- Control technique incorporating:
  – Field Oriented Control for two 3-phase PM synchronous motors with and without position sensor
  – Closed-loop speed control with action period of 1ms
  – Closed-loop current control with action period of 100μs
  – 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 or resolver sensor
  – Robust open-loop start up with two stage rotor alignment
  – Measurement of three-phase motor currents from three shunt resistors
  – FOC state variables sampled with 100μs 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 SW 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, sensor-less 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, $i_s$ is stator current vector and $\psi_s$ 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.

Section 3.1 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:

- Measure the motor quantities (DC link voltage and currents, rotor position/speed).
- 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.
- The stator current torque ($i_{sq}$) and flux ($i_{sd}$) producing components are separately controlled in d, q rotating frame.

- 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.

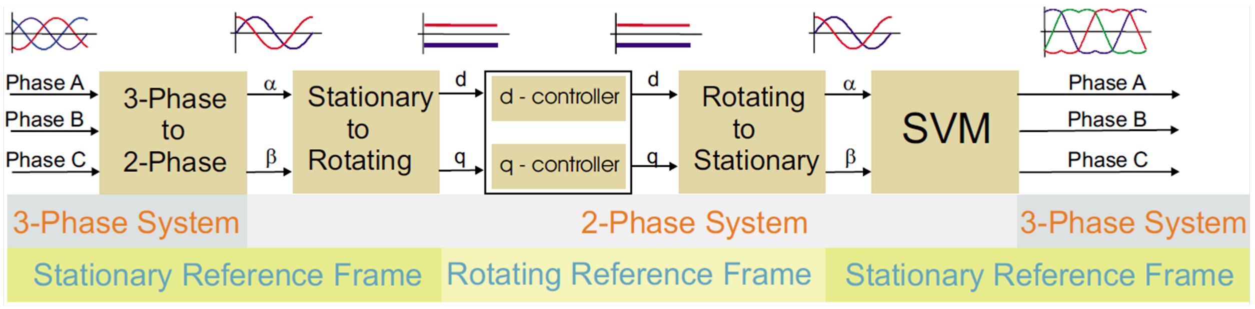


**Figure 1. Field oriented control transformations**

## 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 and $L_{ab}$=$L_{ba}$, $L_{bc}$=$L_{cb}$, $L_{ca}$=$L_{ac}$ are mutual inductances between respective stator phases. The term $\Psi_{PM}$ represents the magnetic flux generated by the rotor permanent magnets, and $\theta_e$ is electrical rotor angle.

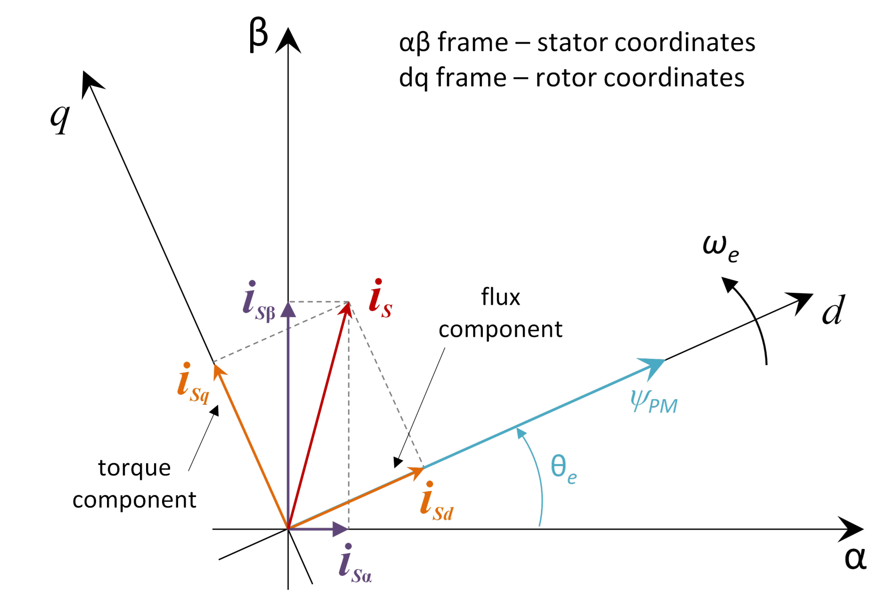


**Figure 2. Orientation of stator (stationary) and rotor (rotational) reference frames, with current components transformed into both frames**

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 Section 3.2. Such transformation, after some mathematical corrections, yields the following set of equations:

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

where $\omega_e$ 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:

$$u_d = R_s i_d + L_d \frac{di_d}{dt} \tag{5}$$

$$u_q = R_s i_q + L_q \frac{di_q}{dt} \tag{6}$$

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:

$$K_p = 2\xi\omega_0 L - R \tag{7}$$

$$K_i = \omega_0^2 L \tag{8}$$

where $\omega_0$ represents the system *natural frequency* [rad/sec] and $\xi$ is the Damping factor [-] of the current control loop.

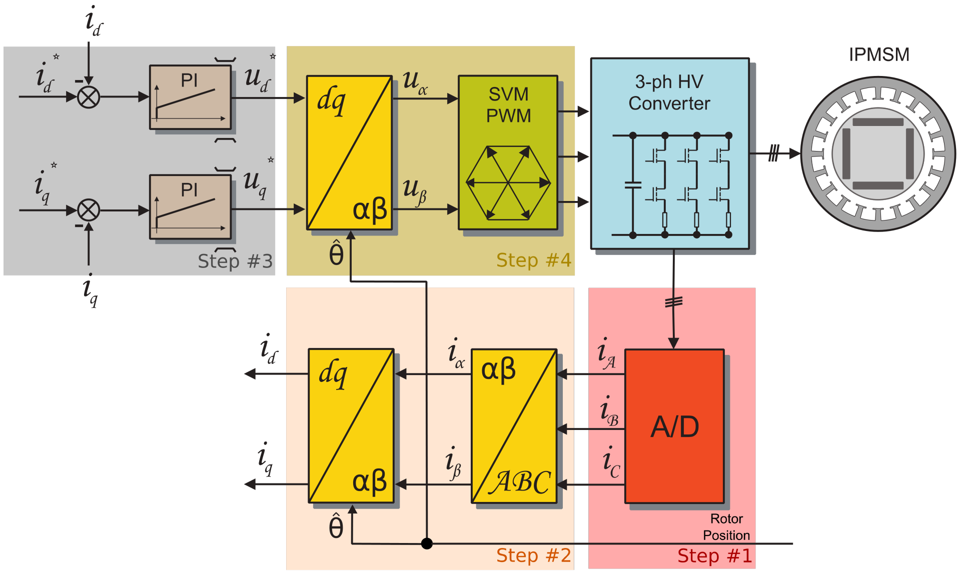


Figure 3. FOC Control Structure

## 3.3 Phase current measurement and output voltage actuation

One leg of 3-phase voltage source inverter, shown in Figure 4, 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.

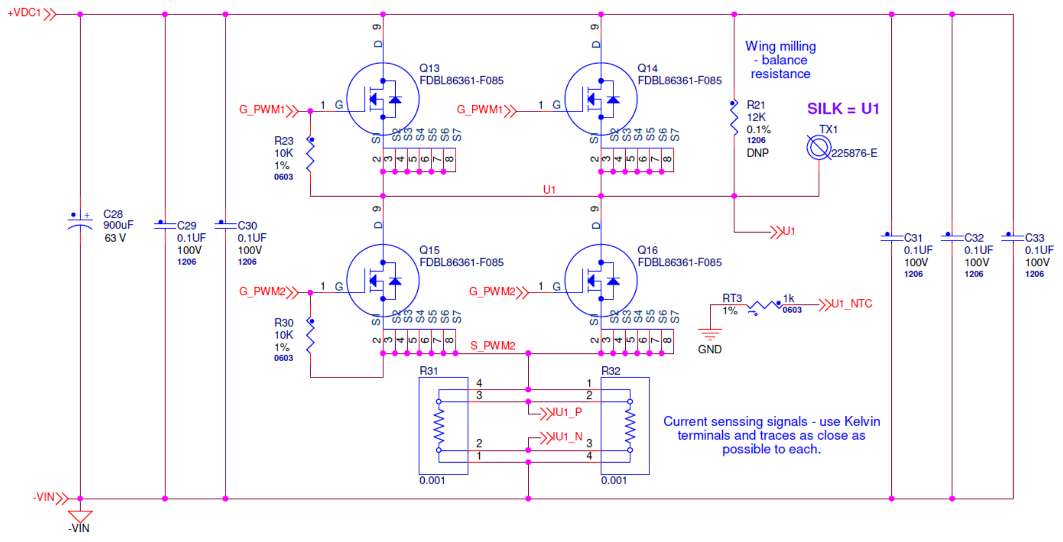


Figure 4. 3-phase DC/AC inverter with shunt resistors for current measurement

Figure 5 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].

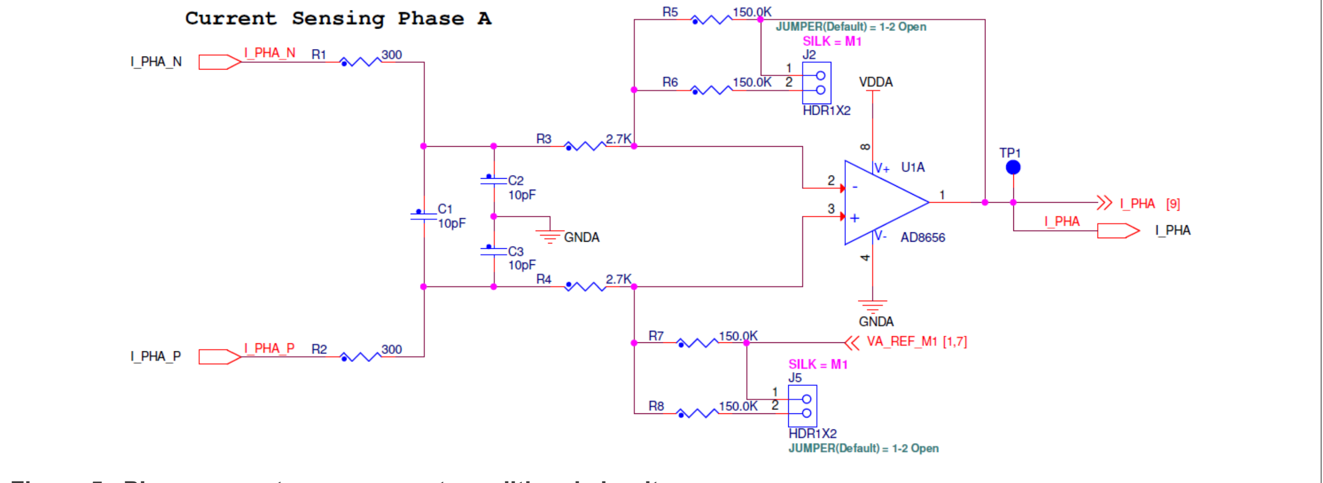


Figure 5. Phase current measurement conditional circuitry

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 the following figure. 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.

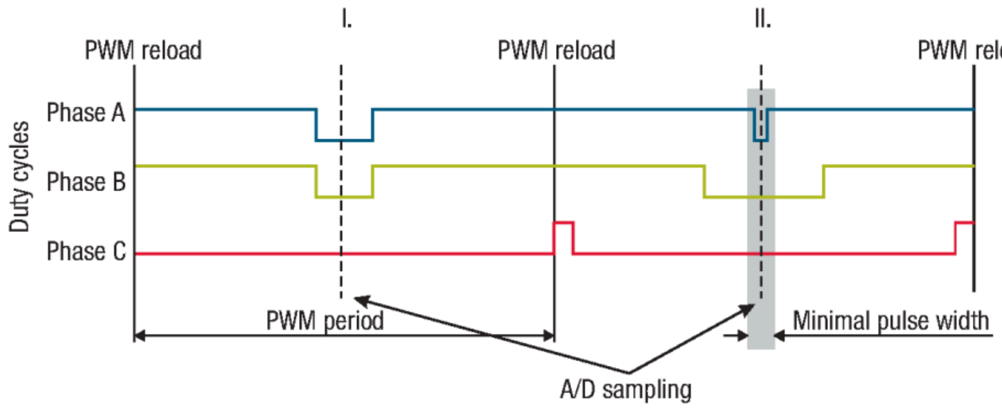


Figure 6. Generated phase duty cycles in different PWM periods

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 and measurement of current is not precise enough. 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]*

## 3.4 Rotor position/speed estimation

In this application, rotor position and speed are either estimated sensor-less by eBEMF observer or obtained by encoder or resolver sensor. eBEMF observer as well as incremental encoder sensor 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 4.3.4.5.

Application in Sensor-less mode must start with open loop start-up sequence to move 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 resolver can start from zero speed because speed and position are provided by sensor.

Structure and implementation details are discussed in section 4.3.5.

***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.

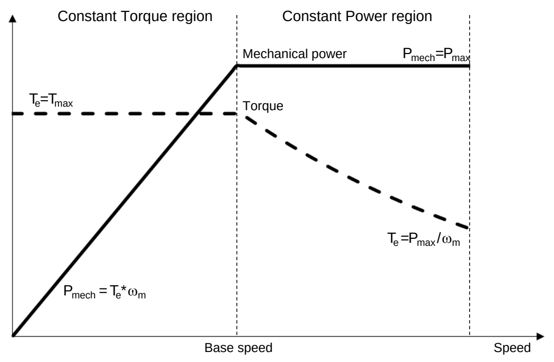


**Figure 7. Constant torque/power operating regions**

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.

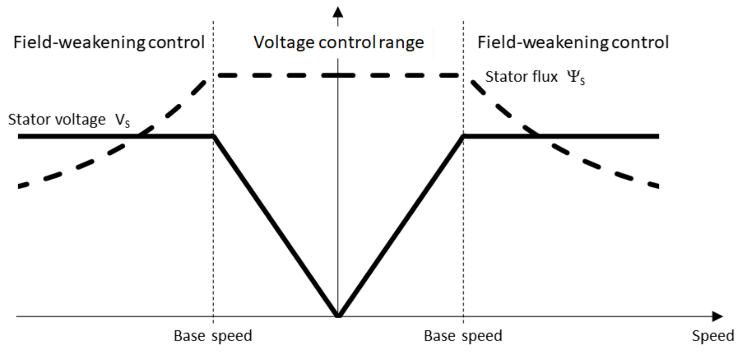


**Figure 8. Constant flux/voltage operational regions**

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 $\Psi_S$. Reduced stator flux $\Psi_S$ 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.

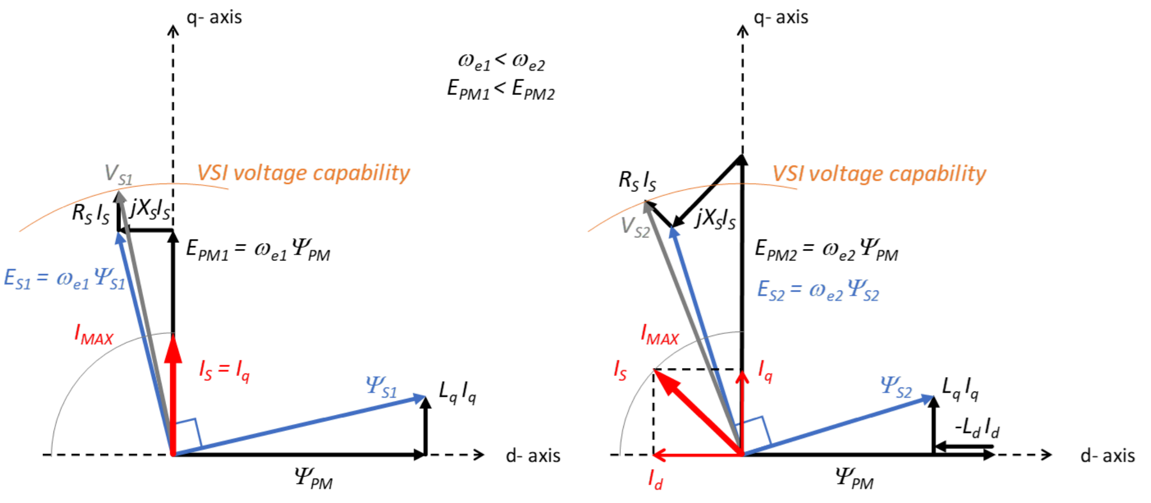


**Figure 9. Steady-state phasor diagram of PMSM operation up to base speed (left) and above speed (right)**

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 $\Psi_{S1}$ is composed of rotor magnetic flux linkage $\Psi_{PM}$, 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 $\Psi_{PM}$ and stator magnetic flux linkage $\Psi_{S1}$ produce BEMF voltage $E_{PM1}=\omega_{e1}\Psi_{PM}$ perpendicularly oriented to rotor magnetic flux $\Psi_{PM}$ in q-axis and BEMF voltage $E_{S1}=\omega_{e1}\Psi_{S1}$ perpendicularly oriented to stator magnetic flux $\Psi_{S1}$, respectively (see Figure 9 left). Both voltages are directly proportional to the rotor speed $\omega_{e1}$. 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 $\Psi_{S2}$ 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 $\Psi_S$ consists of three components now: rotor magnetic flux linkage $\Psi_{PM}$, 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 $\Psi_{PM}$.

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

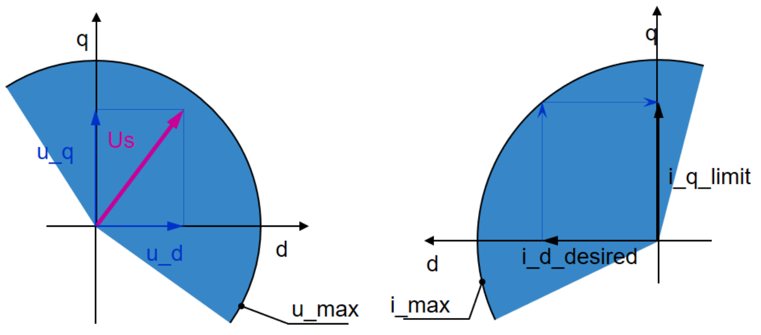


**Figure 10. Voltage (left) and current (right) limits for PMSM drive operation**

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 4.3.5.

## 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 and Figure 12 show a simplified module interconnection for a dual PMSM FOC application working in sensor-less or sensor-based mode using triple-shunt current sensing and either encoder or resolver as 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 3-phase half bridges in order to drive one or two 3-phase AC motors or one 6-phase AC motor. In this case, first configuration of 3-phase half bridge is used to drive first 3-phase PMSM motor (Motor#1), and second configuration of 3-phase half bridge is used to drive second 3-phase PMSM motor (Motor#2). Control of the motors is realized independently.

#### 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 trigger for analog data capturing in the precise moment and counting edges of encoder signal. 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.

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 CH1-CH3 create 3-phase center aligned PWM signals for Motor#1 and share PWM time base CH0, while the eMIOS1 channels CH9-CH11 create 3-

phase center aligned PWM signals for Motor#2 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. The LCU0 OUT0-OUT5 create complementary PWM pairs to control particular MOSFET transistors of the Motor#1, while the LCU0 OUT6-OUT11 create complementary PWM pairs to control particular MOSFET transistors of the Motor#2. The LCU uses Look Up Table, output polarity control and configurable digital filters to generate control signals for transistors with inserted deadtime. The eMIOS1 CH4 and CH5 are dedicated for trigger functionality of the Motor#1, while the eMIOS1 CH12, CH13 and CH14 are dedicated for trigger functionality of the Motor#2. Same as in case of PWM signals OPWMB mode is used for triggers. All trigger channels are linked with trigger time base CH23. Time bases CH0, CH8 and CH23 are synchronized, what offers possibility of an independent configuration of sampling and PWM frequency for both motors.

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, four 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 for both PM motors. 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 of Motor#1 and PHD and PHE of Motor#2 into digital signals, which carry captured edges per particular rotor direction. The eMIOS0 module works as a counter and holds captured edges for clockwise CH3 and counterclockwise CH4 direction of the Motor#1 and also holds captured edges for clockwise CH5 and counterclockwise CH6 direction of the Motor#2. Absolute position of dedicated PM motor is obtained by subtracting counters values of corresponding eMIOS channels.

Resolver functionality is achieved by cooperation of eMIOS, ADC and BCTU modules. The eMIOS1 CH20 provides reference square wave signal for the Motor#1 and CH21 provides reference square wave signal for the Motor#2. Dedicated excitation signal is subsequently filtered by third order Sallen Key low pass filter, which is duplicated to be standalone circuit for each motor. This sine-wave output signal is used as an excitation for resolver. The eMIOS1 CH5 is dedicated for resolver triggering functionality of the Motor#1 and eMIOS1 CH13 is dedicated for resolver triggering functionality of the Motor#2. 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.

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

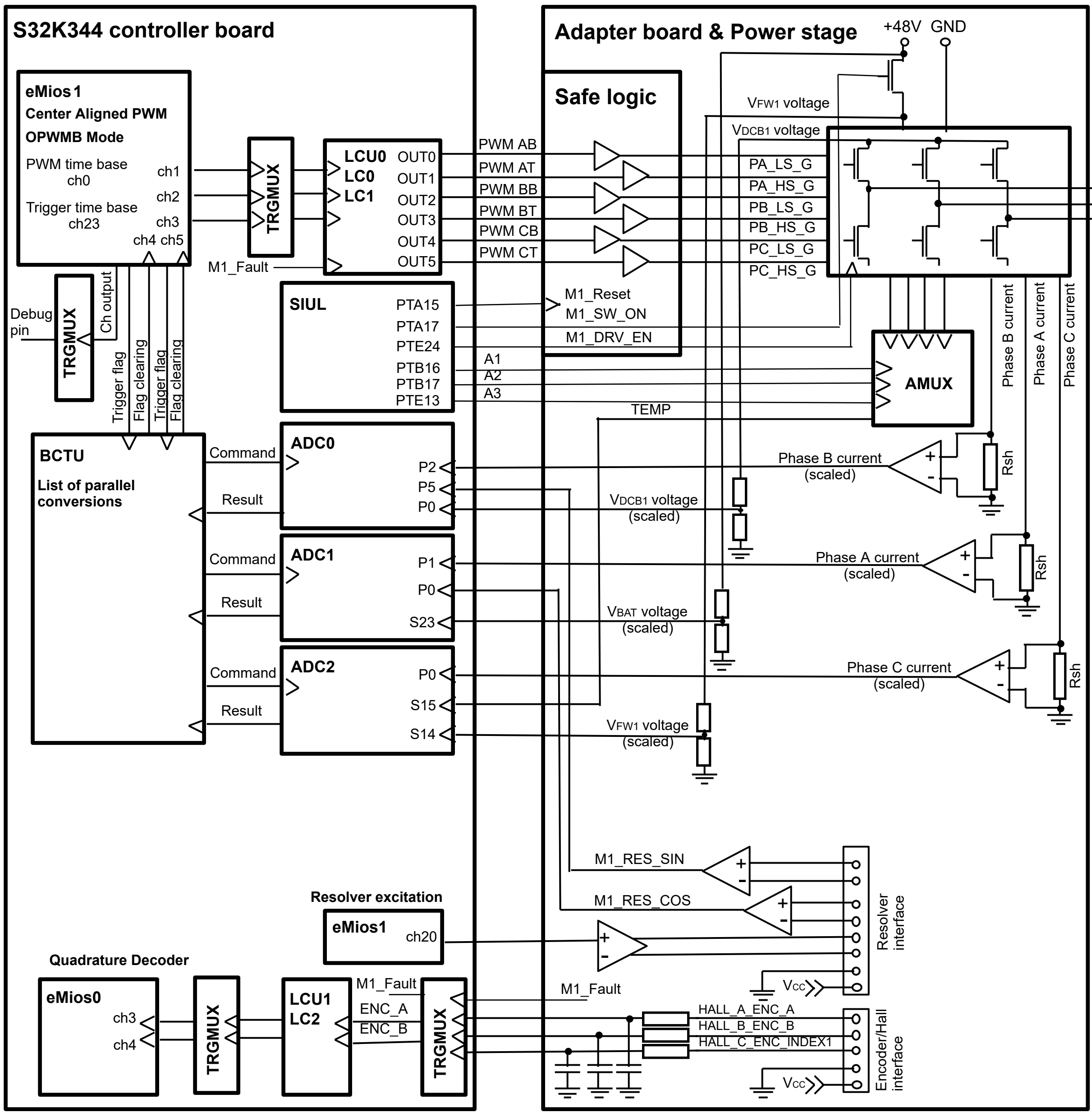


Figure 11.  S32K344 module interconnection for Motor#1

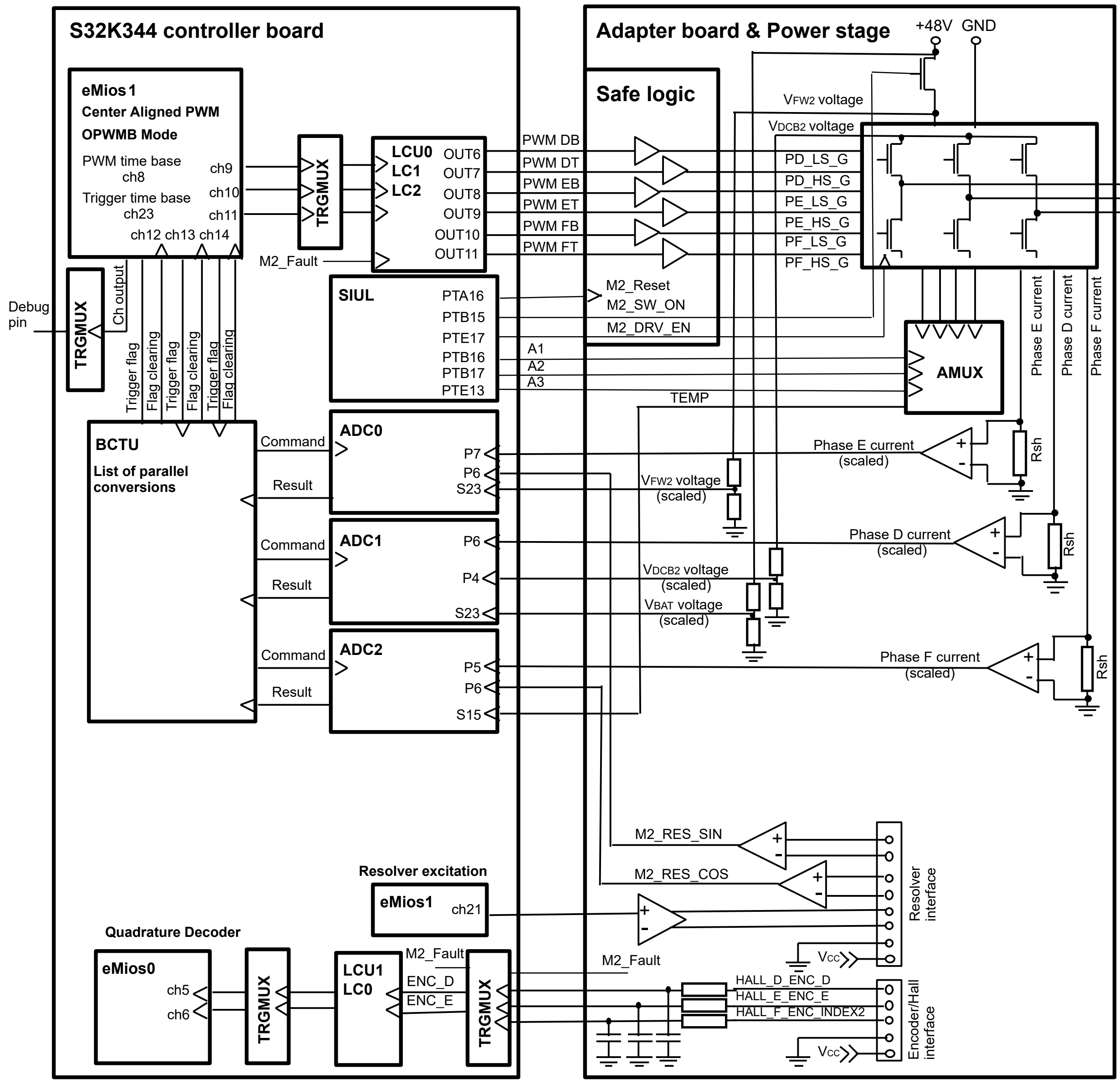


Figure 12. S32K344 module interconnection for Motor#2

## 4.1.2 Module involvement in digital PMSM FOC control loop

This section discusses timing and modules synchronization to accomplish dual PMSM FOC on the S32K344 and the internal hardware features. The time diagram of the automatic synchronization between PWM and ADC in the dual PMSM application is shown in Figure 13.

The dual 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 CH0 creates the PWM time base for channels

CH1, CH2 and CH3 which are responsible for PWM signal generation of the Motor#1 while CH8 creates the PWM time base for channels CH9, CH10 and CH11 which are responsible for PWM signal generation of the Motor#2. The CH23 creates a TRIGGER time base for CH4, CH5, CH12, CH13 and CH14 which are responsible for triggering BCTU. Resolver excitation time base for both PM motors is created by CH23 as well as TRIGGER time base. 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 pre-scaler. Offset between time bases is given by time base channel initial counter value. In this example, local time bases CH0 for the Motor#1 and CH8 for the Motor#2 are shifted to each other and the offset between them is 2us. This offset creates a time shift of generated PWM signals, so PWM signals of the Motor#2 are postponed to PWM signals of the Motor#1. Therefore, measurement of feedback quantities, especially motor phase currents is not overlapped, and it is performed consequently.

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 of the Motor#1 is generated by output of the CH1 while PWM signal for phase A of the Motor#2 is generated by output of the CH9. Center aligned PWM is achieved by proper setting of register A, which defines position of rising edge and register B, which defines position of falling edge. 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 13.

Trigger signals CH4 and CH5 for the Motor#1 and CH12, CH13 and CH14 for the Motor#2 are formed in the same way as PWM signals. An important point here is that the connection between BCTU and eMIOS1 CH4, CH5, CH12, CH13 and CH14 is through the flag of CH4, CH5, CH12, CH13 and CH14 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 CH4, CH5, CH12, CH13 and CH14 output signals. The CH4 and CH12 output signals are routed using the TRGMUX to microcontroller pin for trigger debugging.

When a flag of eMIOS1 CH4 is set, the BCTU starts first list of conversion controlling ADC0, ADC1 and ADC2 and also clears back the CH4 flag. $I_{PHA}$, $I_{PHB}$ and $I_{PHC}$ stator currents of the Motor#1 are measured simultaneously, however measurement is delayed behind 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 inverter legs of first motor configuration are turned on, and currents flow through shunt resistors. Based on SVM state, only two phase currents are taking into account (selection is according SVM state) and third is calculated based on Equation 9.

When a flag of eMIOS1 CH12 is set, the BCTU starts second list of conversion controlling ADC0, ADC1 and ADC2 and also clears back the CH12 flag. $I_{PHD}$, $I_{PHE}$ and $I_{PHF}$ stator currents of the Motor#2 are measured simultaneously, however measurement is delayed behind 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 inverter legs of second motor configuration are turned on, and currents flow through shunt resistors. Based on SVM state, only two phase currents are taking into account (selection is according SVM state) and third is calculated based on Equation 9. Beginning of PWM cycle of Motor#2 is delayed behind the beginning of PWM cycle of Motor#1.

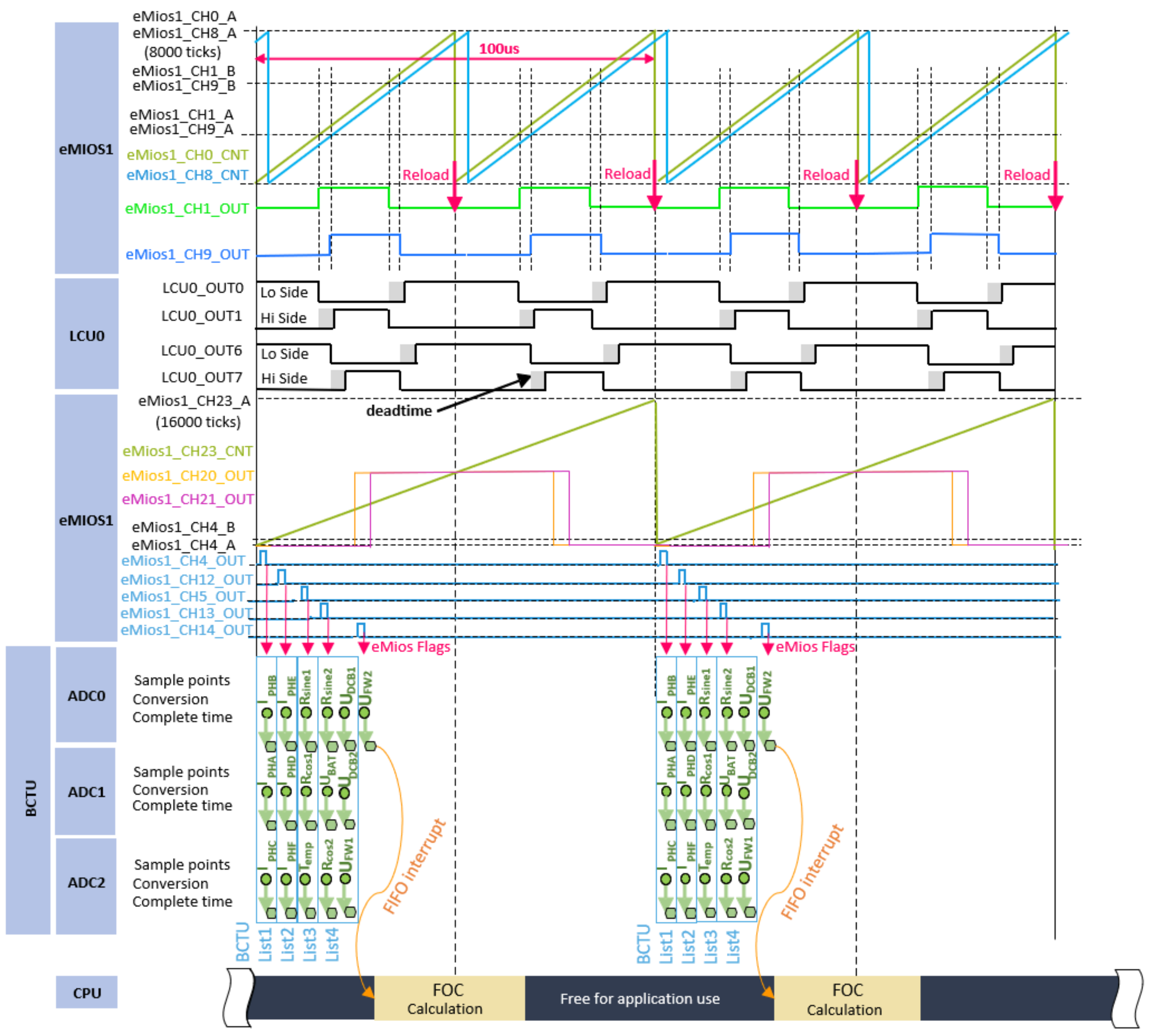


Figure 13. Time Diagram of PWM and ADC Synchronization

When flag of eMIOS1 CH5 is set, the BCTU starts third list of conversions controlling ADC0, ADC1 and ADC2 and also clears back the CH5 flag. Sine and Cosine signals of resolver of the Motor#1 and temperature are measured simultaneously, however measurement of this list is delayed behind the previous measurement, the phase current measurement of the Motor#2. The time of this delay is set in order to respect resolver excitation signal of the Motor#1, where envelope of sine and cosine signals is extracted.

When flag of eMIOS1 CH13 is set, the BCTU starts fourth list of conversions controlling ADC0, ADC1 and ADC2 and also clears back the CH13 flag. Sine and Cosine signals of resolver of the Motor#2 and battery voltage are measured simultaneously, however measurement of this list is delayed behind the previous measurement, resolver sine and cosine signals measurement of the Motor#1. The time of this delay is set in order to respect resolver excitation signal of the Motor#2, where envelope of sine and cosine signals is extracted.

DC-bus voltage $U_{DCB1}$ of the Motor#1, DC-bus voltage $U_{DCB2}$ of the Motor#2 and Forward voltage $U_{FW1}$ of the Motor#1 are measured shortly after measurement of sine and cosine signals of resolver of the Motor#2 and battery voltage. Last measurement is triggered by CH14, where Forward voltage $U_{FW2}$ of the Motor#2 is measured. This measurement is triggered after certain time where all previous measurements are done, and no collision might happen. 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 individually for each motor based on measured quantities. Update of eMIOS1 CH1, CH2, CH3 for the Motor#1 and eMIOS1 CH9, CH10, CH11 for the Motor#2 is done subsequently. 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 14.

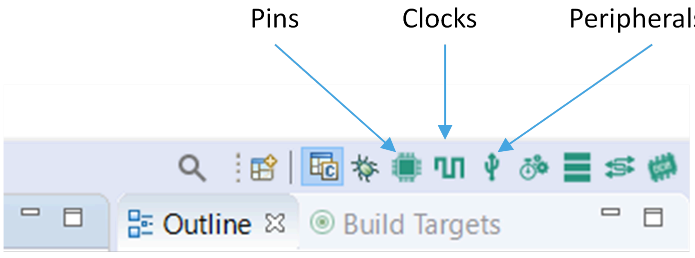


**Figure 14. Config tools**

Figure 15 describes the example project structure in the S32 Design Studio. Current settings of Config tools are stored in MCDXTM4CK344_PMSM_FOC_3Sh_ll.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.

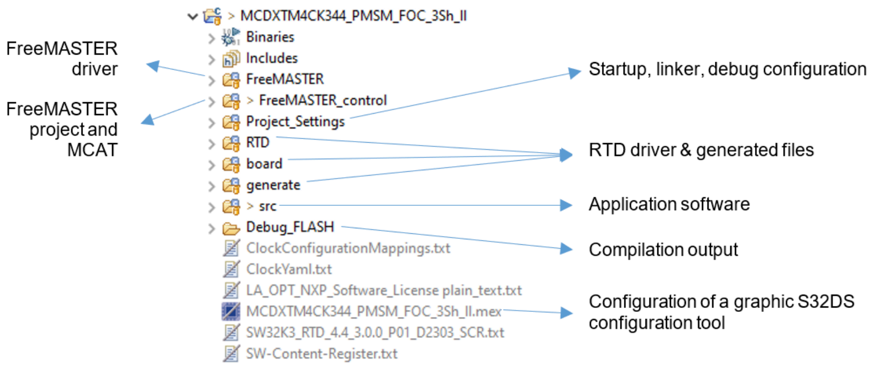

FreeMASTER driver
FreeMASTER project and MCAT
MCDXTM4CK344_PMSM_FOC_3Sh_ll
Binaries
Includes
FreeMASTER
FreeMASTER_control
Project_Settings
RTD
board
generate
src
Debug_FLASH
ClockConfigurationMappings.txt
ClockYaml.txt
LA_OPT_NXP_Software_License plain_text.txt
MCDXTM4CK344_PMSM_FOC_3Sh_ll.mex
SW32K3_RTD_4.4_3.0.0_P01_D2303_SCR.txt
SW-Content-Register.txt
Startup, linker, debug configuration
RTD driver & generated files
Application software
Compilation output
Configuration of a graphic S32DS configuration tool

**Figure 15. Example project structure**

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. 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\S32DS.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 Sensor-less/based FOC control**

| Module | Motor | Signal name | Pin name / Functionality | Description |
|---|---|---|---|---|
| **LCU0** | 1st | PWMA_HS | PTD27 / LCU0_OUT1 | PWM signal for high-side of phase A |
| | | PWMA_LS | PTD26 / LCU0_OUT0 | PWM signal for low-side phase A |
| | | PWMB_HS | PTA2 / LCU0_OUT3 | PWM signal for high-side phase B |
| | | PWMB_LS | PTA3 / LCU0_OUT2 | PWM signal for low-side phase B |
| | | PWMC_HS | PTA1 / LCU0_OUT5 | PWM signal for high-side phase C |
| | | PWMC_LS | PTA0 / LCU0_OUT4 | PWM signal for low-side phase C |
| | 2nd | PWMA_HS | PTB14 / LCU0_OUT7 | PWM signal for high-side of phase A |
| | | PWMA_LS | PTD4 / LCU0_OUT6 | PWM signal for low-side phase A |
| | | PWMB_HS | PTB10 / LCU0_OUT9 | PWM signal for high-side phase B |
| | | PWMB_LS | PTB11 / LCU0_OUT8 | PWM signal for low-side phase B |
| | | PWMC_HS | PTB8 / LCU0_OUT11 | PWM signal for high-side phase C |
| | | PWMC_LS | PTB9 / LCU0_OUT10 | PWM signal for low-side phase C |
| **ADC0** | 1st | M1_DCBV | PTD1 / ADC0_P0 | DC bus voltage measurement |
| | | PHB_I_BEMF_B | PTA8 / ADC0_P2 | Phase B stator current measurement |
| | | M1_RES_SIN | PTE10 / ADC0_P5 | Resolver sine signal measurement |
| | 2nd | M2_V_FW | PTD21 / ADC0_S23 | Forward voltage measurement |
| | | PHE(B)_I_BEMF_E(B) | PTA9 / ADC0_P7 | Phase B stator current measurement |
| | | M2_RES_SIN | PTE11 / ADC0_P6 | Resolver sine signal measurement |
| **ADC1** | 1st | PHA_I_BEMF_A | PTA13 / ADC1_P1 | Phase A stator current measurement |
| | | M1_RES_COS | PTA12 / ADC1_P0 | Resolver cosine signal measurement |
| | 2nd | PHD(A)_I_BEMF_D(A) | PTE6 / ADC1_P6 | Phase A stator current measurement |
| | | M2_DCBV | PTA14 / ADC1_P4 | DC bus voltage measurement |

Table 1. Pins assignment for S32K344 PMSM Sensor-less/based FOC control*...continued*

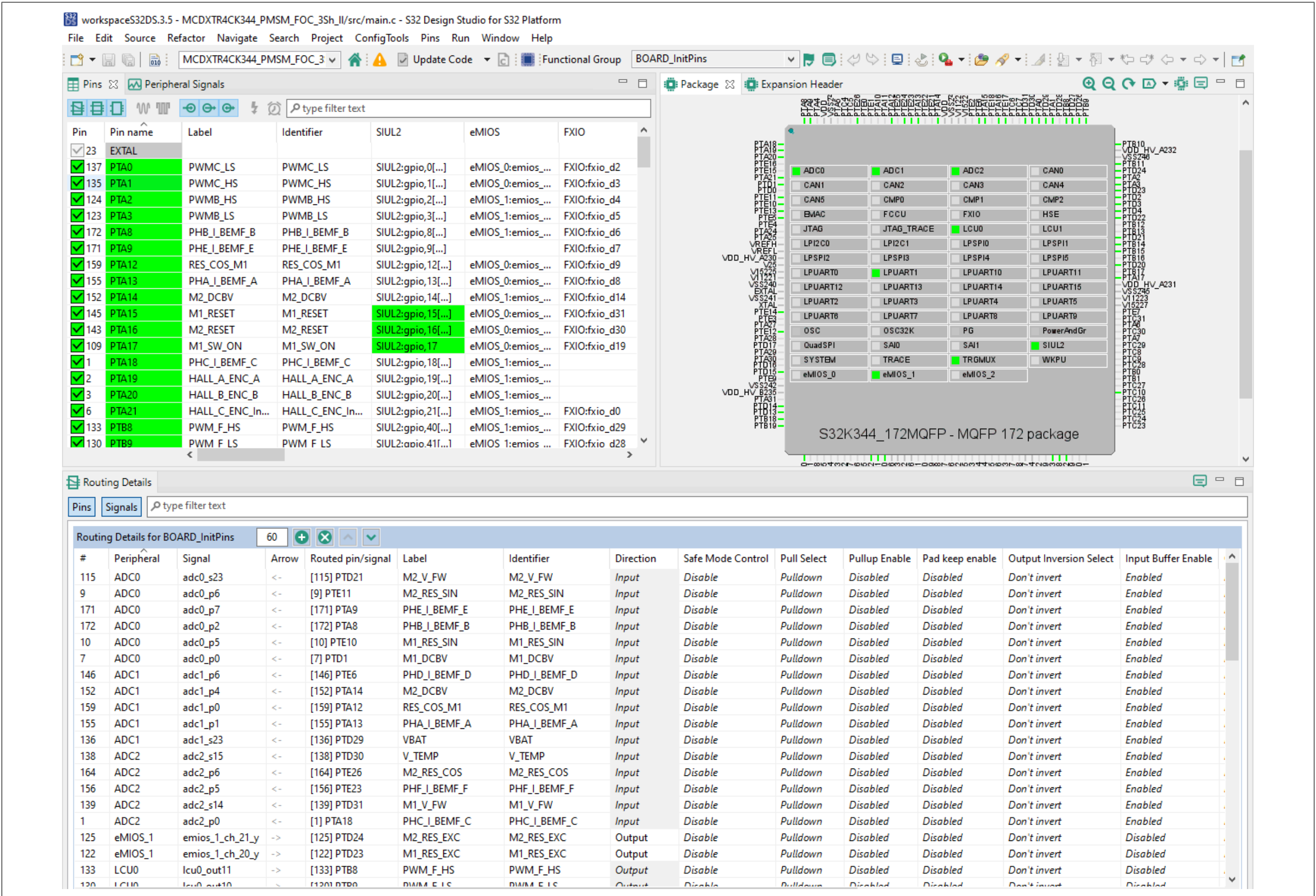
| Module | Motor | Signal name | Pin name / Functionality | Description |
|---|---|---|---|---|
| | - | VBAT | PTD29 / ADC1_S23 | Battery voltage |
| ADC2 | 1st | M1_V_FW | PTD31 / ADC2_S14 | Forward voltage measurement |
| | | PHC_I_BEMF_C | PTA18 / ADC2_P0 | Phase C stator current measurement |
| | 2nd | M2_RES_COS | PTE26 / ADC2_P6 | Resolver cosine signal measurement |
| | | PHF(C)_I_BEMF_F(C) | PTE23 / ADC2_P5 | Phase C stator current measurement |
| | - | V_TEMP | PTD30 / ADC2_S15 | Temperature measurement |
| LPUART1 | | OPEN_SDA_RX | PTD13 / LPUART1_RX | UART transmit data (FreeMASTER) |
| | | OPEN_SDA_TX | PTD14 / LPUART1_TX | UART receive data (FreeMASTER) |
| eMIOS_1 | 1st | M1_RES_EXC | PTD23 / emios_1_ch_20_y | Resolver1 excitation signal |
| | 2nd | M2_RES_EXC | PTD24 / emios_1_ch_21_y | Resolver2 excitation signal |
| TRGMUX | 1st | HALL_A_ENC_A | PTA19 / TRGMUX_IN13 | Phase A signal of the Encoder1 |
| | | HALL_B_ENC_B | PTA20 / TRGMUX_IN14 | Phase B signal of the Encoder1 |
| | | HALL_C_ENC_Index1 | PTA21 / TRGMUX_IN15 | Index signal of the Encoder1 |
| | 2nd | HALL_D_ENC_D | PTD2 / TRGMUX_IN5 | Phase D signal of the Encoder2 |
| | | HALL_E_ENC_E | PTD3 / TRGMUX_IN4 | Phase E signal of the Encoder2 |
| | | HALL_F_ENC_Index2 | PTC10 / TRGMUX_IN11 | Index signal of the Encoder2 |
| TRGMUX | | TRG_OUT10 | PTB19 / TRGMUX_OUT10 | Pin for debugging microcontroller internal signals |
| | | TRG_OUT11 | PTB20 / TRGMUX_OUT11 | Pin for debugging microcontroller internal signals |
| SIUL2 | 1st | M1_SW_UP | PTC13 / GPIO | Application control via board button SW2 |
| | | M1_SW_RUN | PTC12 / GPIO | Application control via board button SW1 |
| | | M1_SW_DOWN | PTC18 / GPIO | Application control via board button SW3 |
| | | M1_SW_ON | PTA17 / GPIO | Motor#1 DC-link soft charge enable |
| | | M1_DRV_EN | PTE17 / GPIO | Motor#1 Gate drivers enable |
| | | M1_RESET | PTA15 / GPIO | Motor#1 Reset of latched faults |
| | 2nd | 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 | Motor#2 DC-link soft charge enable |
| | | M2_DRV_EN | PTE24 / GPIO | Motor#2 Gate drivers enable |
| | | M2_RESET | PTA16 / GPIO | Motor#2 Reset of latched faults |
| SIUL2 | | MCU_RUN | PTE5 / GPIO | MCU control of external fans |
| | | MCU_LED1 | PTE12 / GPIO | LED signalization |
| | | MCU_LED2 | PTE14 / GPIO | LED signalization |
| | | MCU_LED3 | PTD15 / GPIO | LED signalization |

**Table 1. Pins assignment for S32K344 PMSM Sensor-less/based FOC control***...continued*

| Module | Motor | Signal name | Pin name / Functionality | Description |
|---|---|---|---|---|
| | | A1 | PTB16 / GPIO | Control of analog mux for temperature measurement |
| | | A2 | PTB17 / GPIO | Control of analog mux for temperature measurement |
| | | A3 | PTE13 / GPIO | Control of analog mux for temperature measurement |
| | | 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


workspaceS32DS.3.5 - MCDXTR4CK344_PMSM_FOC_3Sh_II/src/main.c - S32 Design Studio for S32 Platform
File Edit Source Refactor Navigate Search Project ConfigTools Pins Run Window Help
MCDXTR4CK344_PMSM_FOC_3
Update Code
Functional Group
BOARD_InitPins
Pins
Peripheral Signals
type filter text
Package
Expansion Header
ADC0 ADC1 ADC2 CAN0
CAN1 CAN2 CAN3 CAN4
CAN5 CMP0 CMP1 CMP2
EMAC FCCU FXIO HSE
JTAG JTAG_TRACE LCU0 LCU1
LPI2C0 LPI2C1 LPSPI0 LPSPI1
LPSPI2 LPSPI3 LPSPI4 LPSPI5
LPUART0 LPUART1 LPUART10 LPUART11
LPUART12 LPUART13 LPUART14 LPUART15
LPUART2 LPUART3 LPUART4 LPUART5
LPUART6 LPUART7 LPUART8 LPUART9
OSC OSC32K PG PowerAndGr
QuadSPI SAI0 SAI1 SIUL2
SYSTEM TRACE TRGMUX WKPU
eMIOS_0 eMIOS_1 eMIOS_2
S32K344_172MQFP - MQFP 172 package
Routing Details
Pins
Signals
type filter text
Routing Details for BOARD_InitPins
60

| Pin | Pin name | Label | Identifier | SIUL2 | eMIOS | FXIO |
|---|---|---|---|---|---|---|
| 23 | EXTAL | | | | | |
| ✓ 137 | PTA0 | PWMC_LS | PWMC_LS | SIUL2:gpio,0[...] | eMIOS_0:emios_... | FXIO:fxio_d2 |
| ✓ 135 | PTA1 | PWMC_HS | PWMC_HS | SIUL2:gpio,1[...] | eMIOS_0:emios_... | FXIO:fxio_d3 |
| ✓ 124 | PTA2 | PWMB_HS | PWMB_HS | SIUL2:gpio,2[...] | eMIOS_1:emios_... | FXIO:fxio_d4 |
| ✓ 123 | PTA3 | PWMB_LS | PWMB_LS | SIUL2:gpio,3[...] | eMIOS_1:emios_... | FXIO:fxio_d5 |
| ✓ 172 | PTA8 | PHB_I_BEMF_B | PHB_I_BEMF_B | SIUL2:gpio,8[...] | eMIOS_1:emios_... | FXIO:fxio_d6 |
| ✓ 171 | PTA9 | PHE_I_BEMF_E | PHE_I_BEMF_E | SIUL2:gpio,9[...] | | FXIO:fxio_d7 |
| ✓ 159 | PTA12 | RES_COS_M1 | RES_COS_M1 | SIUL2:gpio,12[...] | eMIOS_0:emios_... | FXIO:fxio_d9 |
| ✓ 155 | PTA13 | PHA_I_BEMF_A | PHA_I_BEMF_A | SIUL2:gpio,13[...] | eMIOS_0:emios_... | FXIO:fxio_d8 |
| ✓ 152 | PTA14 | M2_DCBV | M2_DCBV | SIUL2:gpio,14[...] | eMIOS_1:emios_... | FXIO:fxio_d14 |
| ✓ 145 | PTA15 | M1_RESET | M1_RESET | SIUL2:gpio,15[...] | eMIOS_0:emios_... | FXIO:fxio_d31 |
| ✓ 143 | PTA16 | M2_RESET | M2_RESET | SIUL2:gpio,16[...] | eMIOS_0:emios_... | FXIO:fxio_d30 |
| ✓ 109 | PTA17 | M1_SW_ON | M1_SW_ON | SIUL2:gpio,17 | eMIOS_0:emios_... | FXIO:fxio_d19 |
| ✓ 1 | PTA18 | PHC_I_BEMF_C | PHC_I_BEMF_C | SIUL2:gpio,18[...] | eMIOS_1:emios_... | |
| ✓ 2 | PTA19 | HALL_A_ENC_A | HALL_A_ENC_A | SIUL2:gpio,19[...] | eMIOS_1:emios_... | |
| ✓ 3 | PTA20 | HALL_B_ENC_B | HALL_B_ENC_B | SIUL2:gpio,20[...] | eMIOS_1:emios_... | |
| ✓ 6 | PTA21 | HALL_C_ENC_In... | HALL_C_ENC_In... | SIUL2:gpio,21[...] | eMIOS_1:emios_... | FXIO:fxio_d0 |
| ✓ 133 | PTB8 | PWM_F_HS | PWM_F_HS | SIUL2:gpio,40[...] | eMIOS_1:emios_... | FXIO:fxio_d29 |
| ✓ 130 | PTB9 | PWM_F_LS | PWM_F_LS | SIUL2:gpio,41[...] | eMIOS_1:emios_... | FXIO:fxio_d28 |

| # | Peripheral | Signal | Arrow | Routed pin/signal | Label | Identifier | Direction | Safe Mode Control | Pull Select | Pullup Enable | Pad keep enable | Output Inversion Select | Input Buffer Enable |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 115 | ADC0 | adc0_s23 | <- | [115] PTD21 | M2_V_FW | M2_V_FW | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 9 | ADC0 | adc0_p6 | <- | [9] PTE11 | M2_RES_SIN | M2_RES_SIN | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 171 | ADC0 | adc0_p7 | <- | [171] PTA9 | PHE_I_BEMF_E | PHE_I_BEMF_E | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 172 | ADC0 | adc0_p2 | <- | [172] PTA8 | PHB_I_BEMF_B | PHB_I_BEMF_B | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 10 | ADC0 | adc0_p5 | <- | [10] PTE10 | M1_RES_SIN | M1_RES_SIN | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 7 | ADC0 | adc0_p0 | <- | [7] PTD1 | M1_DCBV | M1_DCBV | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 146 | ADC1 | adc1_p6 | <- | [146] PTE6 | PHD_I_BEMF_D | PHD_I_BEMF_D | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 152 | ADC1 | adc1_p4 | <- | [152] PTA14 | M2_DCBV | M2_DCBV | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 159 | ADC1 | adc1_p0 | <- | [159] PTA12 | RES_COS_M1 | RES_COS_M1 | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 155 | ADC1 | adc1_p1 | <- | [155] PTA13 | PHA_I_BEMF_A | PHA_I_BEMF_A | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 136 | ADC1 | adc1_s23 | <- | [136] PTD29 | VBAT | VBAT | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 138 | ADC2 | adc2_s15 | <- | [138] PTD30 | V_TEMP | V_TEMP | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 164 | ADC2 | adc2_p6 | <- | [164] PTE26 | M2_RES_COS | M2_RES_COS | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 156 | ADC2 | adc2_p5 | <- | [156] PTE23 | PHF_I_BEMF_F | PHF_I_BEMF_F | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 139 | ADC2 | adc2_s14 | <- | [139] PTD31 | M1_V_FW | M1_V_FW | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 1 | ADC2 | adc2_p0 | <- | [1] PTA18 | PHC_I_BEMF_C | PHC_I_BEMF_C | Input | Disable | Pulldown | Disabled | Disabled | Don't invert | Enabled |
| 125 | eMIOS_1 | emios_1_ch_21_y | -> | [125] PTD24 | M2_RES_EXC | M2_RES_EXC | Output | Disable | Pulldown | Disabled | Disabled | Don't invert | Disabled |
| 122 | eMIOS_1 | emios_1_ch_20_y | -> | [122] PTD23 | M1_RES_EXC | M1_RES_EXC | Output | Disable | Pulldown | Disabled | Disabled | Don't invert | Disabled |
| 133 | LCU0 | lcu0_out11 | -> | [133] PTB8 | PWM_F_HS | PWM_F_HS | Output | Disable | Pulldown | Disabled | Disabled | Don't invert | Disabled |
| 130 | LCU0 | lcu0_out10 | -> | [130] PTB9 | PWM_F_LS | PWM_F_LS | Output | Disable | Pulldown | Disabled | Disabled | Don't invert | Disabled |

**Figure 16. Pins**

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. The pins assignment can be carried out by means of Pins tool.

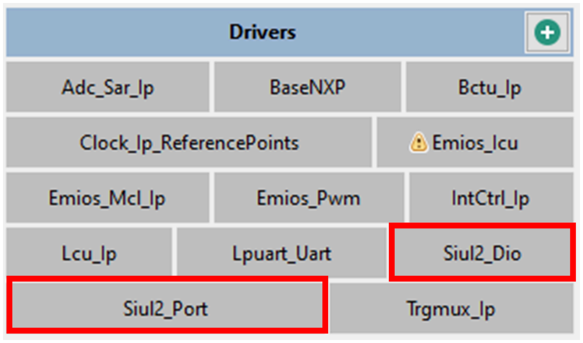


**Figure 17. Pins SW drivers**

The pin assignment of the example is shown in Figure 16. 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 Example 1.

**Example 1. 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 */
    drvFOC1.cntrState.usrControl.btSpeedUp = !
Siul2_Dio_Ip_ReadPin(M1_SW_UP_PORT, M1_SW_UP_PIN);
    drvFOC1.cntrState.usrControl.btSpeedDown = !
Siul2_Dio_Ip_ReadPin(M1_SW_DOWN_PORT, M1_SW_DOWN_PIN);
    drvFOC1.cntrState.usrControl.btFlipFlop = !
Siul2_Dio_Ip_ReadPin(M1_SW_RUN_PORT, M1_SW_RUN_PIN);
...
    drvFOC2.cntrState.usrControl.btSpeedUp = !
Siul2_Dio_Ip_ReadPin(M2_SW_UP_PORT, M2_SW_UP_PIN);
    drvFOC2.cntrState.usrControl.btSpeedDown = !
Siul2_Dio_Ip_ReadPin(M2_SW_DOWN_PORT, M2_SW_DOWN_PIN);
    drvFOC2.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.

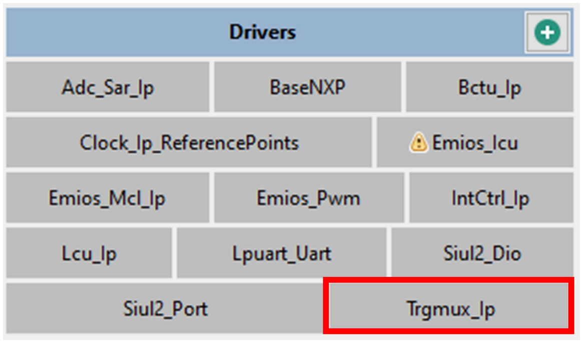
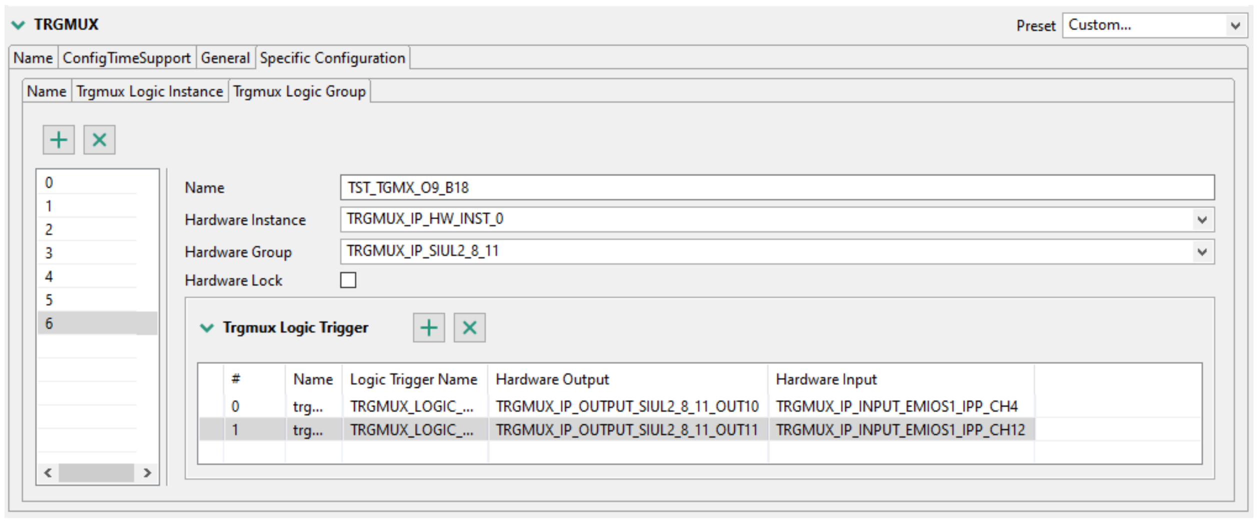


**Figure 18. TRGMUX SW driver**

The connection is made by selecting specific hardware output and input. PTB19 visualizes output of eMIOS1 CH4, which is a trigger signal for current measurement of Motor#1 and PTB20 visualizes eMIOS1 CH12, which is a trigger signal for current measurement of Motor#2. Other signals like reload can be visualized by changing the "Hardware Input" configuration. Setting is applied by calling Trgmux_Ip_Init function.

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


TRGMUX Preset Custom...

Name | ConfigTimeSupport | General | Specific Configuration

Name | Trgmux Logic Instance | Trgmux Logic Group

0
1
2
3
4
5
6

Name: TST_TGMX_O9_B18
Hardware Instance: TRGMUX_IP_HW_INST_0
Hardware Group: TRGMUX_IP_SIUL2_8_11
Hardware Lock: ☐

Trgmux Logic Trigger

| # | Name | Logic Trigger Name | Hardware Output | Hardware Input |
|---|---|---|---|---|
| 0 | trg... | TRGMUX_LOGIC_... | TRGMUX_IP_OUTPUT_SIUL2_8_11_OUT10 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH4 |
| 1 | trg... | TRGMUX_LOGIC_... | TRGMUX_IP_OUTPUT_SIUL2_8_11_OUT11 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH12 |

**Figure 19. TRGMUX groups for debugging purpose**

### 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.

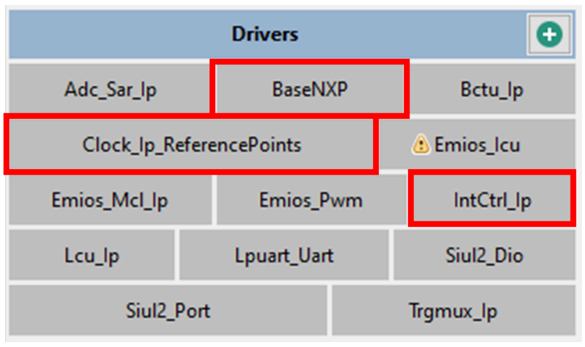


**Figure 20. Clock, Os Interface and interrupts**

### 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).

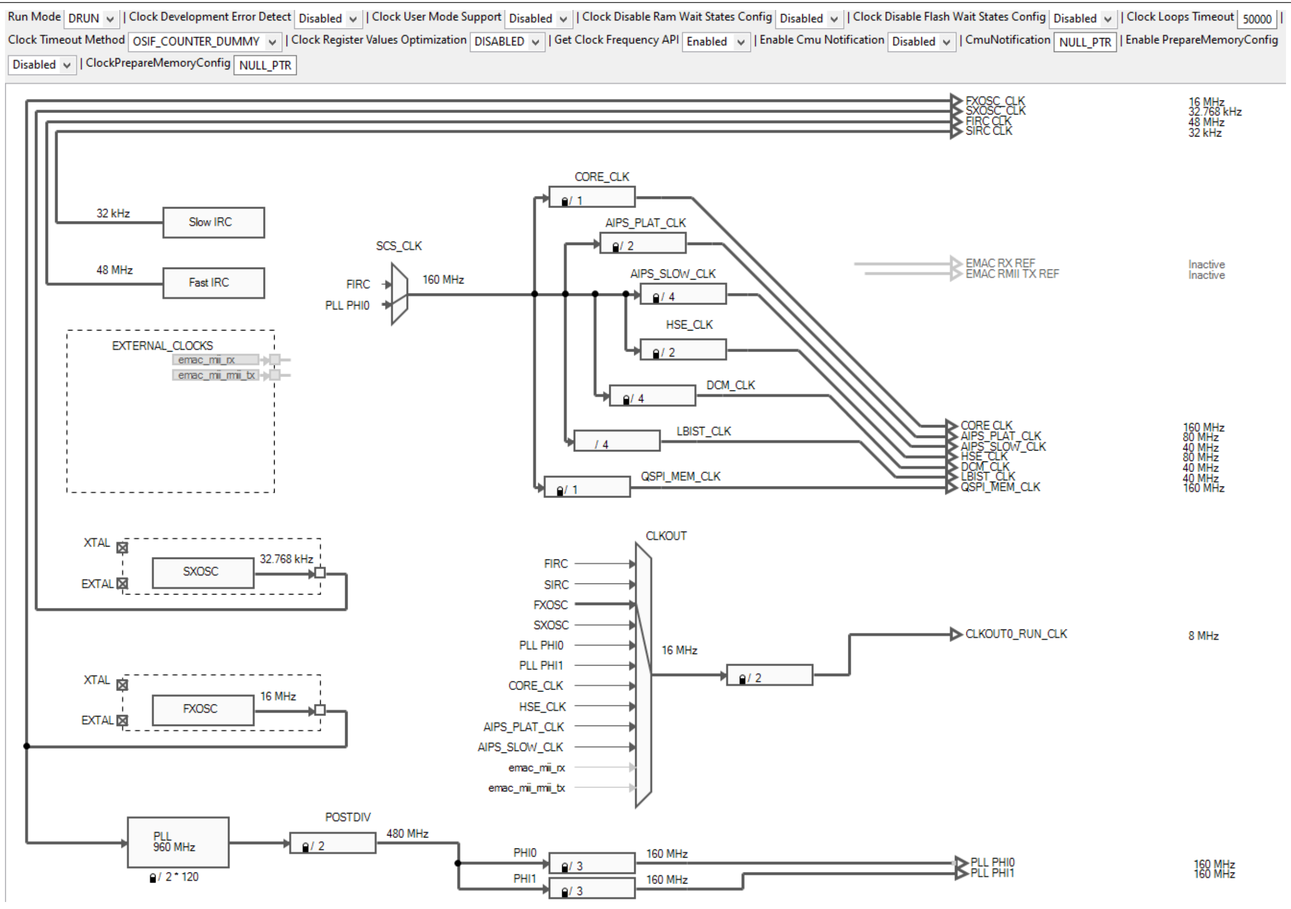

Run Mode DRUN | Clock Development Error Detect Disabled | Clock User Mode Support Disabled | Clock Disable Ram Wait States Config Disabled | Clock Disable Flash Wait States Config Disabled | Clock Loops Timeout 50000 | Clock Timeout Method OSIF_COUNTER_DUMMY | Clock Register Values Optimization DISABLED | Get Clock Frequency API Enabled | Enable Cmu Notification Disabled | CmuNotification NULL_PTR | Enable PrepareMemoryConfig Disabled | ClockPrepareMemoryConfig NULL_PTR

**Figure 21. Clocks tool**

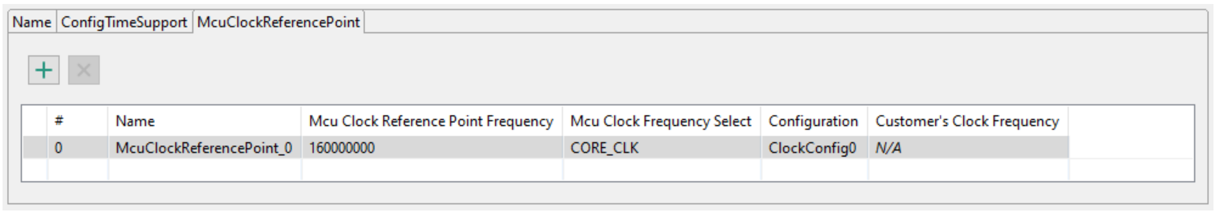
To run the core of the S32K344 at maximum frequency 160MHz, S32K344 is supplied externally by 16 MHz crystal. This clock source supplies Phase-lock-loop (PLL), and its output is adjusted to 160 MHz frequency. PLL output PHI0 is then used to supply the core CORE_CLK. All real-time control peripherals 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

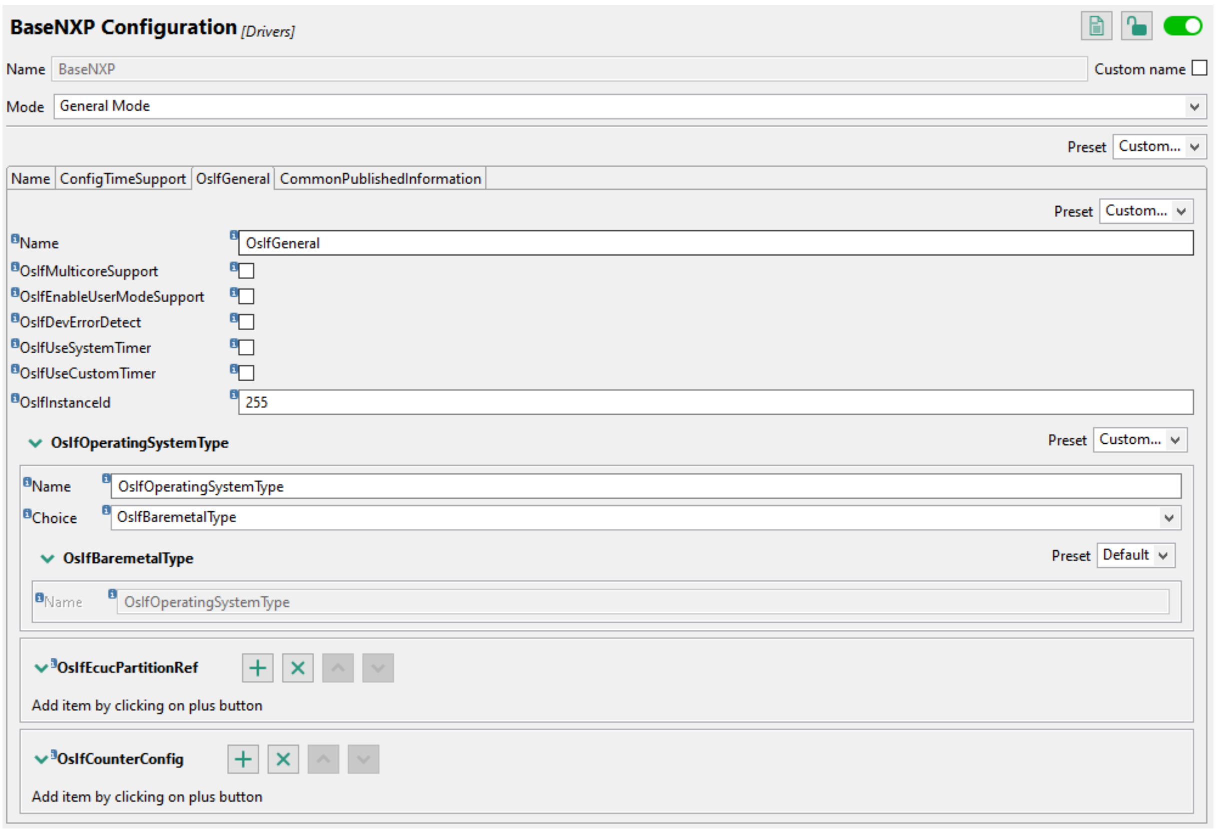
| Clock | Frequency | MCU Peripherals |
|---|---|---|
| CORE_CLK | 160 MHz | BCTU, LCU0-1, eMIOS0-2 |
| CORE_CLK/2 | 80 MHz | ADC0-2 |
| AIPS_SLOW_CLK | 40 MHz | LPUART1, TRGMUX |

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 a bare-metal software without operating system, but other drivers can use OsIF for timeouts detection. OSIF settings are applied by calling OsIf_Init() function.



| # | Name | Mcu Clock Reference Point Frequency | Mcu Clock Frequency Select | Configuration | Customer's Clock Frequency |
|---|---|---|---|---|---|
| 0 | McuClockReferencePoint_0 | 160000000 | CORE_CLK | ClockConfig0 | N/A |

Figure 22. Clock configuration



BaseNXP Configuration *[Drivers]*

Name: BaseNXP  Custom name ☐

Mode: General Mode

Preset: Custom...

Tabs: Name | ConfigTimeSupport | OsIfGeneral | CommonPublishedInformation

Preset: Custom...

| | |
|---|---|
| Name | OsIfGeneral |
| OsIfMulticoreSupport | ☐ |
| OsIfEnableUserModeSupport | ☐ |
| OsIfDevErrorDetect | ☐ |
| OsIfUseSystemTimer | ☐ |
| OsIfUseCustomTimer | ☐ |
| OsIfInstanceId | 255 |

OsIfOperatingSystemType  Preset: Custom...

| | |
|---|---|
| Name | OsIfOperatingSystemType |
| Choice | OsIfBaremetalType |

OsIfBaremetalType  Preset: Default

| | |
|---|---|
| Name | OsIfOperatingSystemType |

OsIfEcucPartitionRef

Add item by clicking on plus button

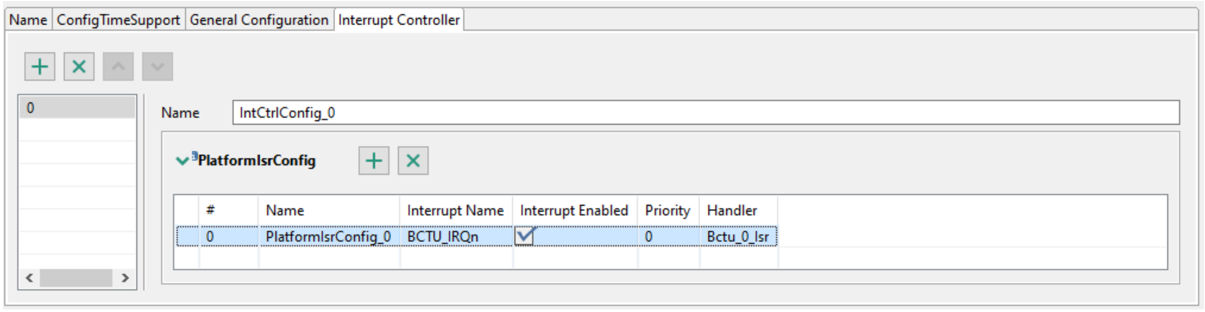
OsIfCounterConfig

Add item by clicking on plus button

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. This example is using only interrupt from BCTU. There are 3 options to set in “*Interrupt controller”* column “*Handler*”. If a 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 handles its interrupt and call notification function on specific event defined by Bctu_Ip component settings in peripheral tool (BctuFifoNotif).



Tabs: Name | ConfigTimeSupport | General Configuration | Interrupt Controller

0

Name: IntCtrlConfig_0

PlatformIsrConfig

| # | Name | Interrupt Name | Interrupt Enabled | Priority | Handler |
|---|---|---|---|---|---|
| 0 | PlatformIsrConfig_0 | BCTU_IRQn | ✓ | 0 | Bctu_0_Isr |

Figure 24. Interrupt controller

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

**Example 2. Interrupt controller API**

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

### 4.2.3 Center-aligned PWM

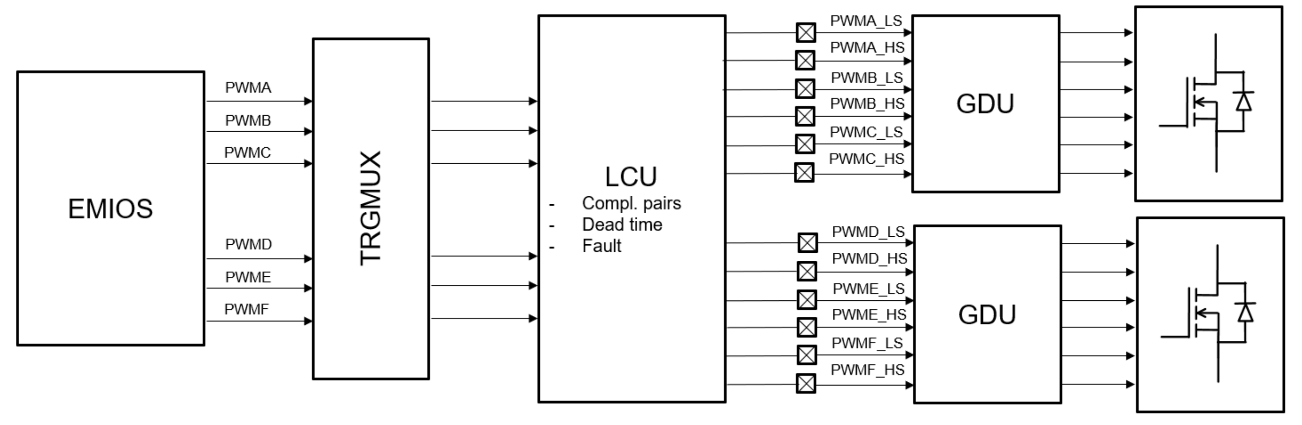


**Figure 25. Block diagram of key modules for PWM generation**

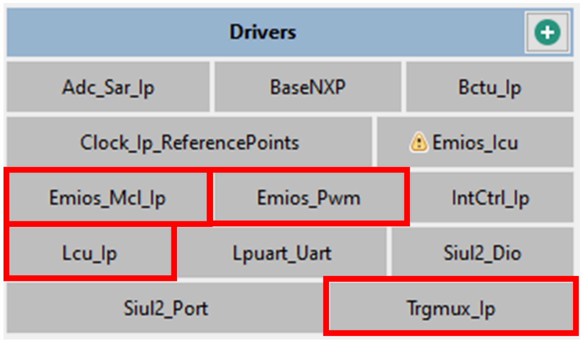
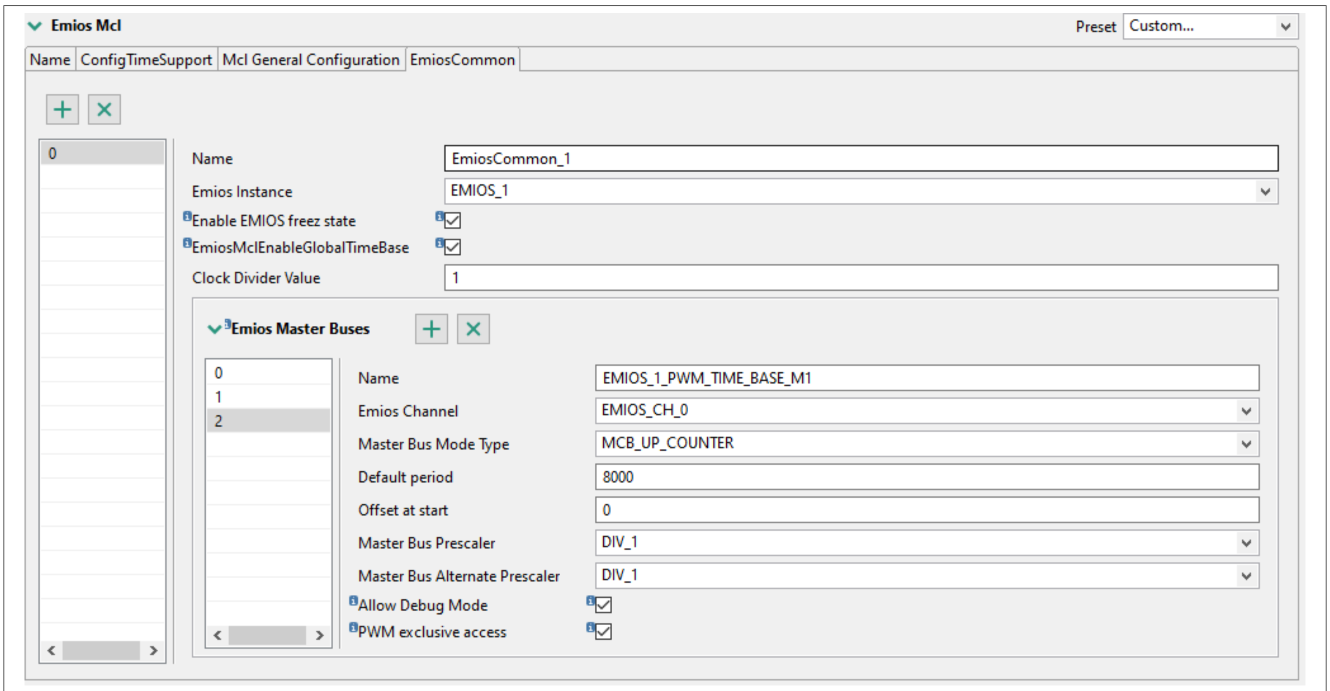


**Figure 26. Drivers for PWM generation**

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

The eMIOS1 CH0 and eMIOS1 CH8 are configured as a time bases for PWM signals. The Motor#1 configuration uses eMIOS1 CH0 while the Motor#2 uses eMIOS1 CH8. The eMIOS1 CH0 channel can create local time base for CH 1-7 while the eMIOS1 CH8 channel can create local time base for CH 9-15. Both channels operate in a Modulus Counter Buffered (MCB) up-counting mode only.



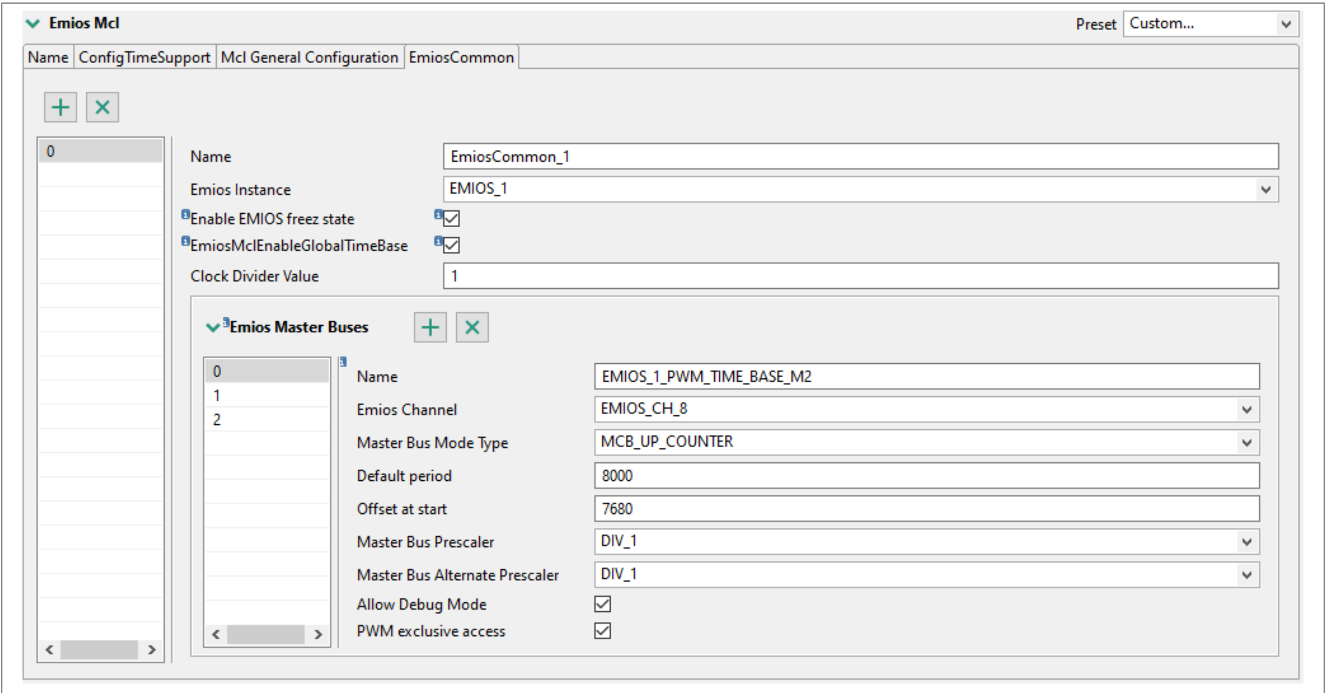
| | |
|---|---|
| Name | EmiosCommon_1 |
| Emios Instance | EMIOS_1 |
| Enable EMIOS freez state | ✓ |
| EmiosMclEnableGlobalTimeBase | ✓ |
| Clock Divider Value | 1 |

Emios Master Buses: 0, 1, 2

| | |
|---|---|
| Name | EMIOS_1_PWM_TIME_BASE_M1 |
| Emios Channel | EMIOS_CH_0 |
| Master Bus Mode Type | MCB_UP_COUNTER |
| Default period | 8000 |
| Offset at start | 0 |
| Master Bus Prescaler | DIV_1 |
| Master Bus Alternate Prescaler | DIV_1 |
| Allow Debug Mode | ✓ |
| PWM exclusive access | ✓ |

**Figure 27. PWM time-base configuration for Motor#1**

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. In this example, eMIOS1 CH8 channel starts counting with offset while eMIOS1 CH0 channel starts counting from zero, so generation of PWM of the Motor#2 is delayed behind PWM signals of the Motor#1.



Emios Mcl

Preset: Custom...

Name | ConfigTimeSupport | Mcl General Configuration | EmiosCommon

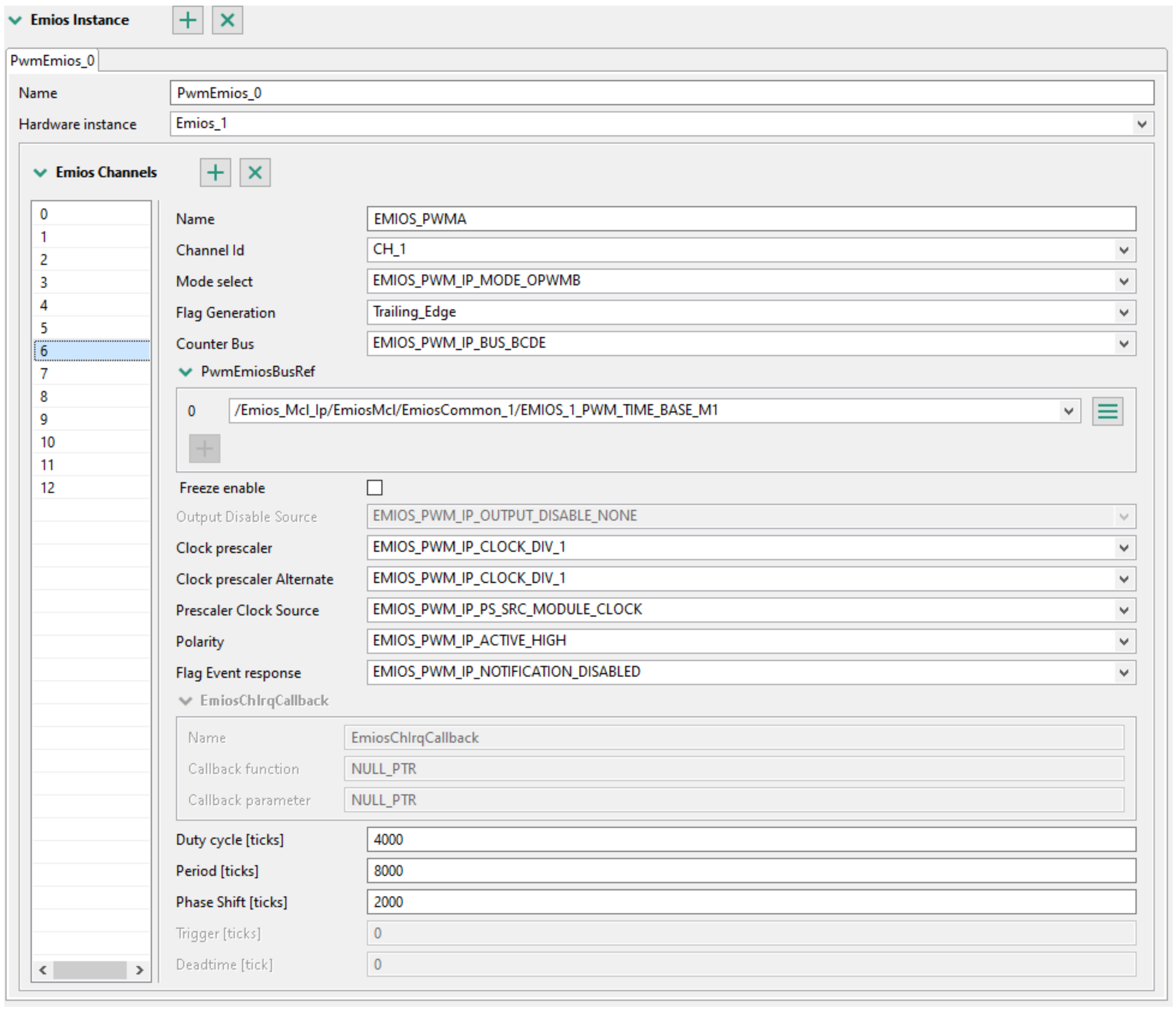
| | |
|---|---|
| Name | EmiosCommon_1 |
| Emios Instance | EMIOS_1 |
| Enable EMIOS freez state | ☑ |
| EmiosMclEnableGlobalTimeBase | ☑ |
| Clock Divider Value | 1 |

Emios Master Buses (0, 1, 2)

| | |
|---|---|
| Name | EMIOS_1_PWM_TIME_BASE_M2 |
| Emios Channel | EMIOS_CH_8 |
| Master Bus Mode Type | MCB_UP_COUNTER |
| Default period | 8000 |
| Offset at start | 7680 |
| Master Bus Prescaler | DIV_1 |
| Master Bus Alternate Prescaler | DIV_1 |
| Allow Debug Mode | ☑ |
| PWM exclusive access | ☑ |

Figure 28. PWM time-base configuration for Motor#2

eMIOS1 CH 1-3 are configured to generate the PWM signal for the Motor#1 phases PHA-C and eMIOS1 CH 9-11 are configured to generate the PWM signal for the Motor#2 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. Channels are able to see time base counter value through the BCDE bus and compare it with its registers A and B. The eMIOS1 CH 1-3 are linked with eMIOS1 CH0 channel time base while the eMIOS1 CH 9-11 are linked with eMIOS1 CH8 time base channel. "*Polarity*" defines output state on specific compare. Complete timing diagram can be found in Figure 13. 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 the 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.



Name: PwmEmios_0
Hardware instance: Emios_1

| | |
|---|---|
| Name | EMIOS_PWMA |
| Channel Id | CH_1 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_BCDE |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_PWM_TIME_BASE_M1 |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_HIGH |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| EmiosChIrqCallback: Name | EmiosChIrqCallback |
| Callback function | NULL_PTR |
| Callback parameter | NULL_PTR |
| Duty cycle [ticks] | 4000 |
| Period [ticks] | 8000 |
| Phase Shift [ticks] | 2000 |
| Trigger [ticks] | 0 |
| Deadtime [tick] | 0 |

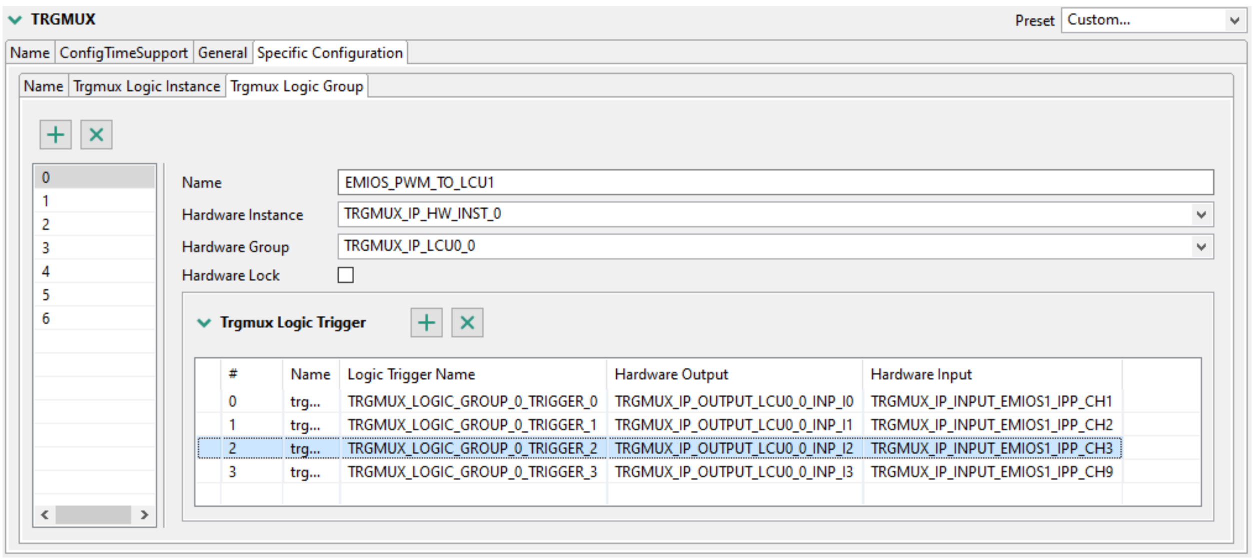
Figure 29. PWM channel configuration

### Example 3.eMIOS API for PWM

```
void main (void)
{
 /*********************************************************************
 * eMios Driver
 *********************************************************************/
 Emios_Mcl_Ip_Init(1U, &Emios_Mcl_Ip_1_Config_BOARD_INITPERIPHERALS);

#if Motor1
 Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch1);    /* PWM A    */
 Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch2);    /* PWM B    */
 Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch3);    /* PWM C    */
#endif
#if Motor2
 Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch9);    /* PWM A    */
 Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch10);   /* PWM B    */
 Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch11);   /* PWM C    */
#endif
...
 /*Enable eMIOS clock at last to ensure the correct trigger order*/
 Emios_Mcl_Ip_ConfigureGlobalTimebase(1U, TRUE);
...
}
tBool ACTUATE_SetDutycycle(pmsmDrive_t *ptr)
{
 Emios_Mcl_Ip_ComparatorTransferDisable((ptr->pwm_cfg.PWM_Inst),(ptr-
>pwm_cfg.PWM_Mask));
...
 Emios_Pwm_Ip_UpdateUCRegA((ptr->pwm_cfg.PWM_A.eMIOS_Inst), (ptr-
>pwm_cfg.PWM_A.eMIOS_Chan), pwmShiftA_A);
 Emios_Pwm_Ip_UpdateUCRegB((ptr->pwm_cfg.PWM_A.eMIOS_Inst), (ptr-
>pwm_cfg.PWM_A.eMIOS_Chan), pwmShiftA_B);
 Emios_Pwm_Ip_UpdateUCRegA((ptr->pwm_cfg.PWM_B.eMIOS_Inst), (ptr-
>pwm_cfg.PWM_B.eMIOS_Chan), pwmShiftB_A);
 Emios_Pwm_Ip_UpdateUCRegB((ptr->pwm_cfg.PWM_B.eMIOS_Inst), (ptr-
>pwm_cfg.PWM_B.eMIOS_Chan), pwmShiftB_B);
 Emios_Pwm_Ip_UpdateUCRegA((ptr->pwm_cfg.PWM_C.eMIOS_Inst), (ptr-
>pwm_cfg.PWM_C.eMIOS_Chan), pwmShiftC_A);
 Emios_Pwm_Ip_UpdateUCRegB((ptr->pwm_cfg.PWM_C.eMIOS_Inst), (ptr-
>pwm_cfg.PWM_C.eMIOS_Chan), pwmShiftC_B);
...
 Emios_Mcl_Ip_ComparatorTransferEnable((ptr->pwm_cfg.PWM_Inst),(ptr-
>pwm_cfg.PWM_Mask));
}
```

### 4.2.3.2 TRIGGER MUX

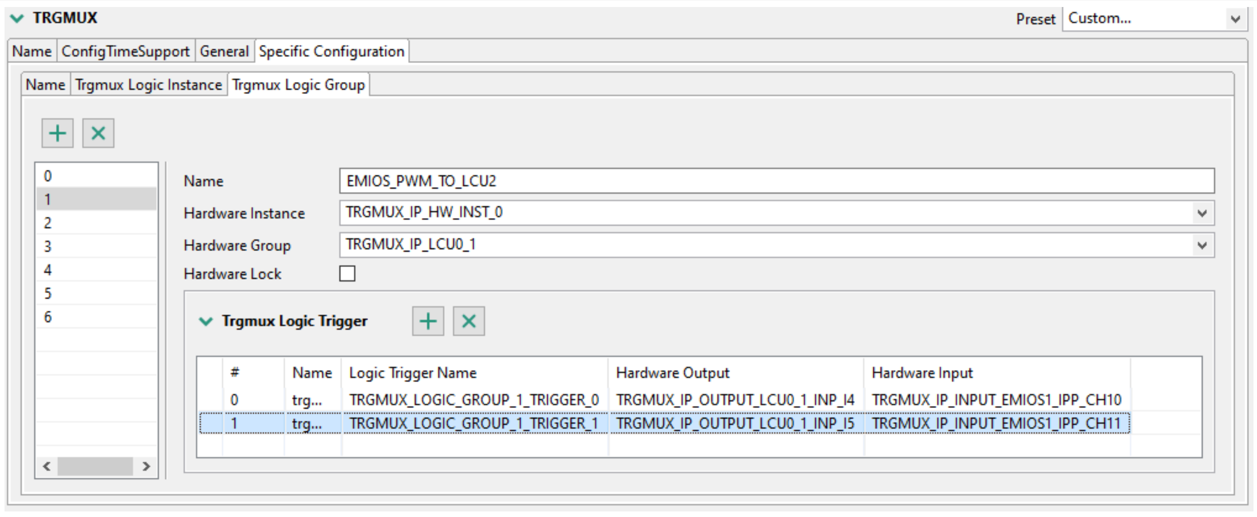
TRGMUX ensures a connection between eMIOS and LCU. Settings within the "*Hardware group"*



Name: EMIOS_PWM_TO_LCU1
Hardware Instance: TRGMUX_IP_HW_INST_0
Hardware Group: TRGMUX_IP_LCU0_0
Hardware Lock: ☐

| # | Name | Logic Trigger Name | Hardware Output | Hardware Input |
|---|---|---|---|---|
| 0 | trg... | TRGMUX_LOGIC_GROUP_0_TRIGGER_0 | TRGMUX_IP_OUTPUT_LCU0_0_INP_I0 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH1 |
| 1 | trg... | TRGMUX_LOGIC_GROUP_0_TRIGGER_1 | TRGMUX_IP_OUTPUT_LCU0_0_INP_I1 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH2 |
| 2 | trg... | TRGMUX_LOGIC_GROUP_0_TRIGGER_2 | TRGMUX_IP_OUTPUT_LCU0_0_INP_I2 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH3 |
| 3 | trg... | TRGMUX_LOGIC_GROUP_0_TRIGGER_3 | TRGMUX_IP_OUTPUT_LCU0_0_INP_I3 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH9 |

**Figure 30. TRGMUX settings for PWM signals of Motor#1 and phase A of Motor#2**



| | |
|---|---|
| Name | EMIOS_PWM_TO_LCU2 |
| Hardware Instance | TRGMUX_IP_HW_INST_0 |
| Hardware Group | TRGMUX_IP_LCU0_1 |
| Hardware Lock | ☐ |

| # | Name | Logic Trigger Name | Hardware Output | Hardware Input |
|---|---|---|---|---|
| 0 | trg... | TRGMUX_LOGIC_GROUP_1_TRIGGER_0 | TRGMUX_IP_OUTPUT_LCU0_1_INP_I4 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH10 |
| 1 | trg... | TRGMUX_LOGIC_GROUP_1_TRIGGER_1 | TRGMUX_IP_OUTPUT_LCU0_1_INP_I5 | TRGMUX_IP_INPUT_EMIOS1_IPP_CH11 |

**Figure 31. TRGMUX settings for PWM signals of phases B and C of Motor#2**

TRGMUX_IP_LCU0_0 and TRGMUX_IP_LCU0_1 connects outputs of eMIOS1 channels 1-3 and channels 9-11 to LCU0 inputs 0-5. Setting is applied by calling Trgmux_Ip_Init() function.

### 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 following functionality is needed: Input multiplexing, Look Up Table (LUT), Digital filters, output polarity settings as is shown in Figure 32. Full featured LCU diagram can be found in S32K3xx Reference Manual [7].

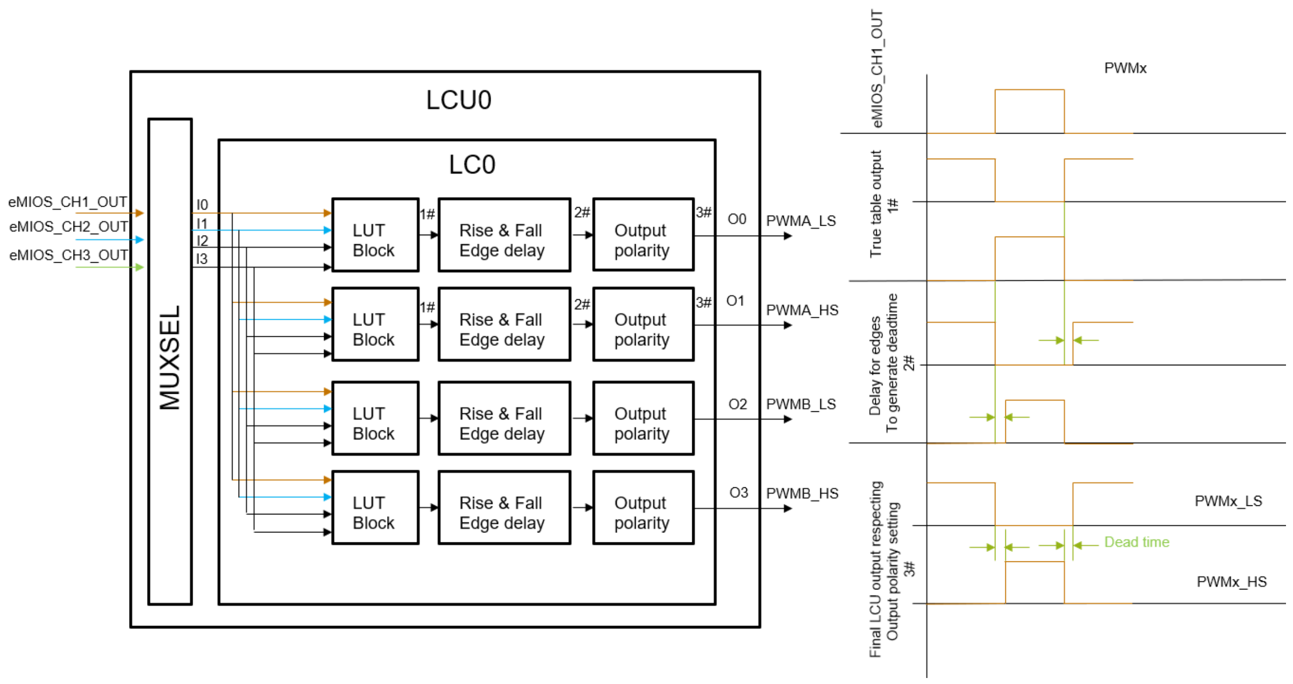


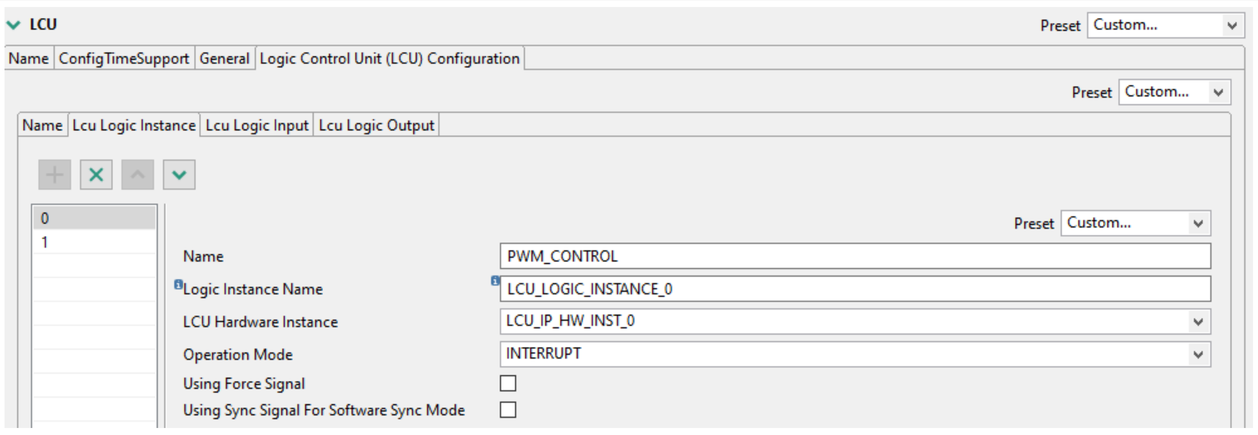
**Figure 32. Simplified LCU features block diagram for PWM signals**

Lcu_Ip driver is used to configure and to control LCU. In this example LCU0 instance is selected to generate PWM complementary pairs for both motors. LC0 generates signals for phases A and phase B of the Motor#1, LC1 generates signals for phases C of the Motor#1 and phase A (phase D) of the Motor#2 and LC2 generates signals for phase B and C (phase E and F) of the Motor#2. First configuration relates to inputs multiplexing. Configurations 0-5 in *“Lcu Logic Input”* section create a connection between LCU instance inputs and LC inputs. Multiplexor inputs, *“Mux Select”,* 0, 1 (eMIOS1 CH1, 2) are connected to LC0 input 0-1, *“Hardware input”,* multiplexor inputs 2, 3 (eMIOS1 CH3, 9) are connected to LC1 inputs 0 and 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 for the Motor#1 and configurations 6-11 for the Motor#2. 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.

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

| LC0_I3 | LC0_I2 | LC0_I1 | LC0_I0 | | LC0_O0 | LC0_O1 | LC0_O2 | LC0_O3 |
|---|---|---|---|---|---|---|---|---|
| x | x | PWM_PHB | PWM_PHA | | PWMA_LS | PWMA_HS | PWMB_LS | PWMB_HS |
| 0 | 0 | 0 | 0 | | 1 | 0 | 1 | 0 |
| 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** |



LCU

Preset Custom...

Name | ConfigTimeSupport | General | Logic Control Unit (LCU) Configuration

Preset Custom...

Name | Lcu Logic Instance | Lcu Logic Output | Lcu Logic Output

0
1

Preset Custom...

Name: PWM_CONTROL

Logic Instance Name: LCU_LOGIC_INSTANCE_0

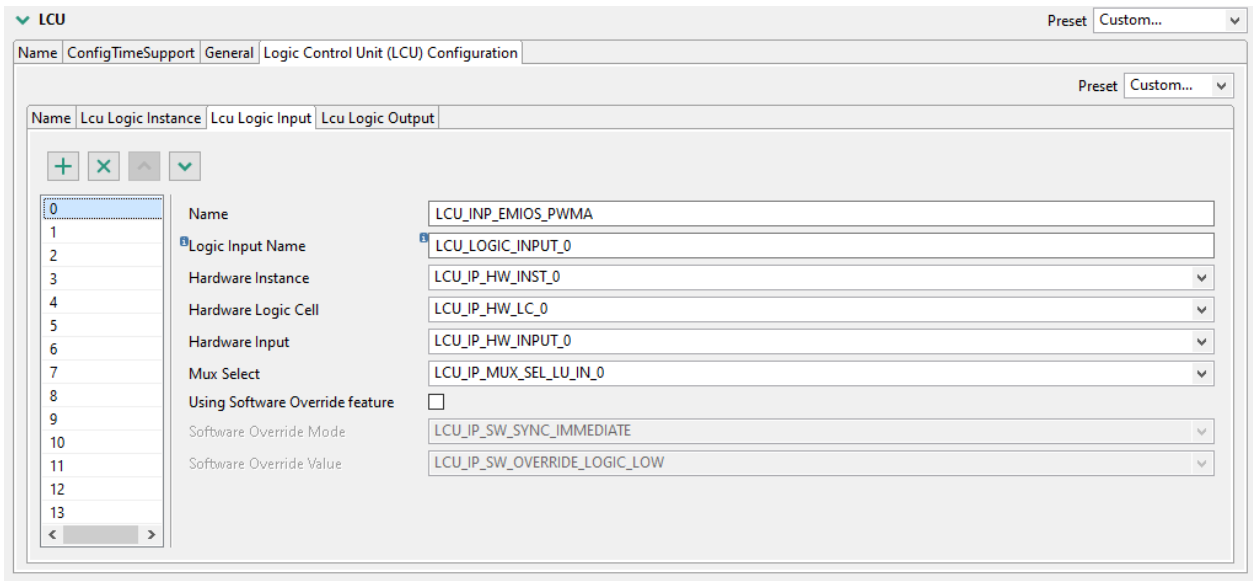
LCU Hardware Instance: LCU_IP_HW_INST_0

Operation Mode: INTERRUPT

Using Force Signal: ☐

Using Sync Signal For Software Sync Mode: ☐

Figure 33. LCU instance configuration for PWM



LCU

Preset Custom...

Name | ConfigTimeSupport | General | Logic Control Unit (LCU) Configuration

Preset Custom...

Name | Lcu Logic Instance | Lcu Logic Input | Lcu Logic Output

0
1
2
3
4
5
6
7
8
9
10
11
12
13

Name: LCU_INP_EMIOS_PWMA

Logic Input Name: LCU_LOGIC_INPUT_0

Hardware Instance: LCU_IP_HW_INST_0

Hardware Logic Cell: LCU_IP_HW_LC_0

Hardware Input: LCU_IP_HW_INPUT_0

Mux Select: LCU_IP_MUX_SEL_LU_IN_0

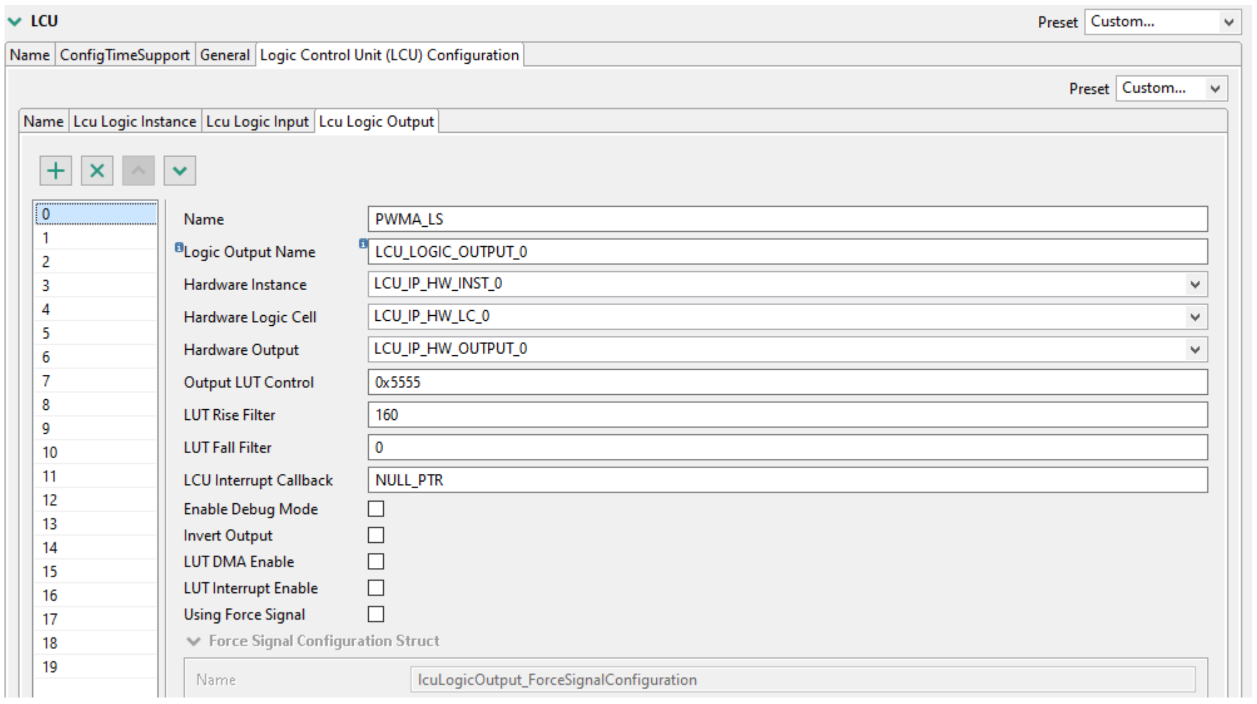
Using Software Override feature: ☐

Software Override Mode: LCU_IP_SW_SYNC_IMMEDIATE

Software Override Value: LCU_IP_SW_OVERRIDE_LOGIC_LOW

Figure 34. LC inputs configuration for PWM

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 32. 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().


LCU
Preset: Custom...
Name | ConfigTimeSupport | General | Logic Control Unit (LCU) Configuration
Preset: Custom...
Name | Lcu Logic Instance | Lcu Logic Input | Lcu Logic Output

| | |
|---|---|
| Name | PWMA_LS |
| Logic Output Name | LCU_LOGIC_OUTPUT_0 |
| Hardware Instance | LCU_IP_HW_INST_0 |
| Hardware Logic Cell | LCU_IP_HW_LC_0 |
| Hardware Output | LCU_IP_HW_OUTPUT_0 |
| Output LUT Control | 0x5555 |
| LUT Rise Filter | 160 |
| LUT Fall Filter | 0 |
| LCU Interrupt Callback | NULL_PTR |
| Enable Debug Mode | ☐ |
| Invert Output | ☐ |
| LUT DMA Enable | ☐ |
| LUT Interrupt Enable | ☐ |
| Using Force Signal | ☐ |

Force Signal Configuration Struct
Name: lcuLogicOutput_ForceSignalConfiguration

Figure 35. LC outputs for PWM

## 4.2.4 Analog data capturing

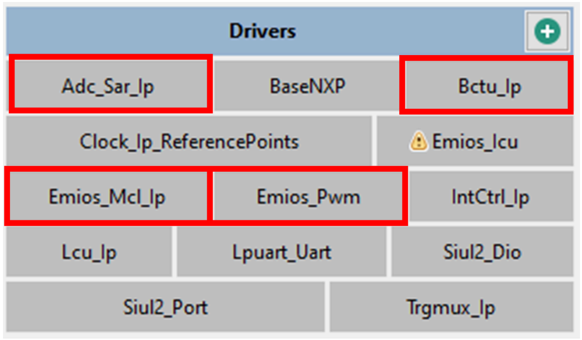


Figure 36. Drivers for analog feedback capturing

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

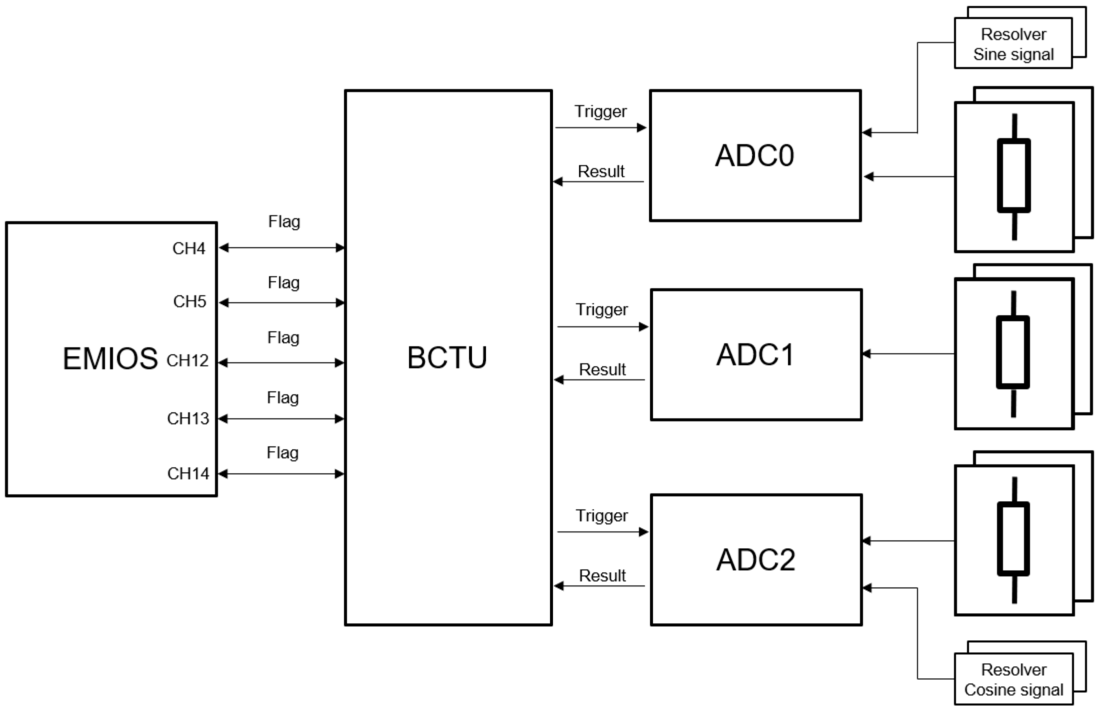


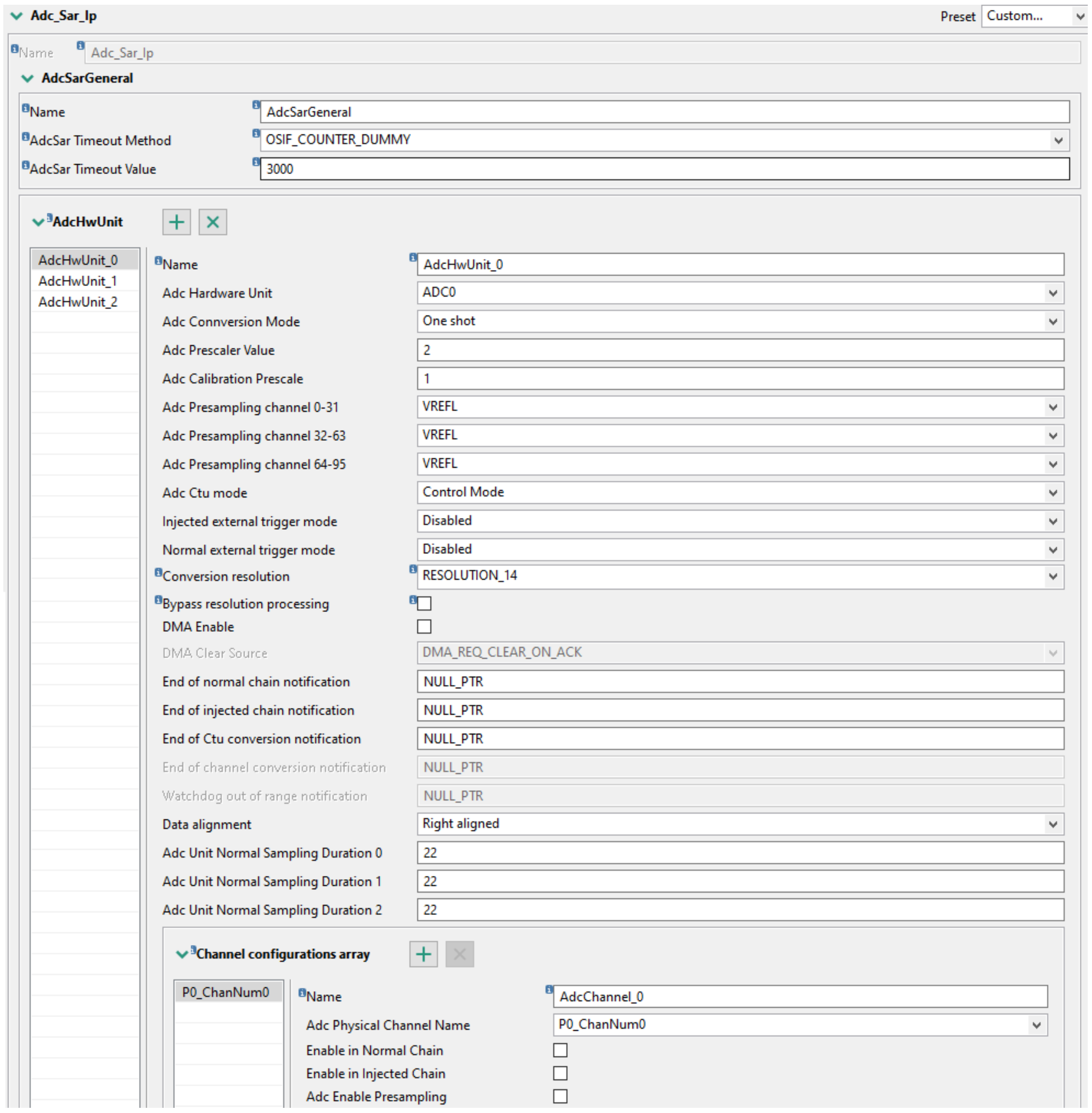
**Figure 37. MC analog feedback capturing**

### 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().

**Example 4.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);...
}
```



Adc_Sar_Ip — Preset: Custom...

Name: Adc_Sar_Ip

AdcSarGeneral

| | |
|---|---|
| Name | AdcSarGeneral |
| AdcSar Timeout Method | OSIF_COUNTER_DUMMY |
| AdcSar Timeout Value | 3000 |

AdcHwUnit: AdcHwUnit_0, AdcHwUnit_1, AdcHwUnit_2

| | |
|---|---|
| Name | AdcHwUnit_0 |
| Adc Hardware Unit | ADC0 |
| Adc Connversion Mode | One shot |
| Adc Prescaler Value | 2 |
| Adc Calibration Prescale | 1 |
| Adc Presampling channel 0-31 | VREFL |
| Adc Presampling channel 32-63 | VREFL |
| Adc Presampling channel 64-95 | VREFL |
| Adc Ctu mode | Control Mode |
| Injected external trigger mode | Disabled |
| Normal external trigger mode | Disabled |
| Conversion resolution | RESOLUTION_14 |
| Bypass resolution processing | ☐ |
| DMA Enable | ☐ |
| DMA Clear Source | DMA_REQ_CLEAR_ON_ACK |
| End of normal chain notification | NULL_PTR |
| End of injected chain notification | NULL_PTR |
| End of Ctu conversion notification | NULL_PTR |
| End of channel conversion notification | NULL_PTR |
| Watchdog out of range notification | NULL_PTR |
| Data alignment | Right aligned |
| Adc Unit Normal Sampling Duration 0 | 22 |
| Adc Unit Normal Sampling Duration 1 | 22 |
| Adc Unit Normal Sampling Duration 2 | 22 |

Channel configurations array: P0_ChanNum0

| | |
|---|---|
| Name | AdcChannel_0 |
| Adc Physical Channel Name | P0_ChanNum0 |
| Enable in Normal Chain | ☐ |
| Enable in Injected Chain | ☐ |
| Adc Enable Presampling | ☐ |

Figure 38. ADC configuration for MC measurements

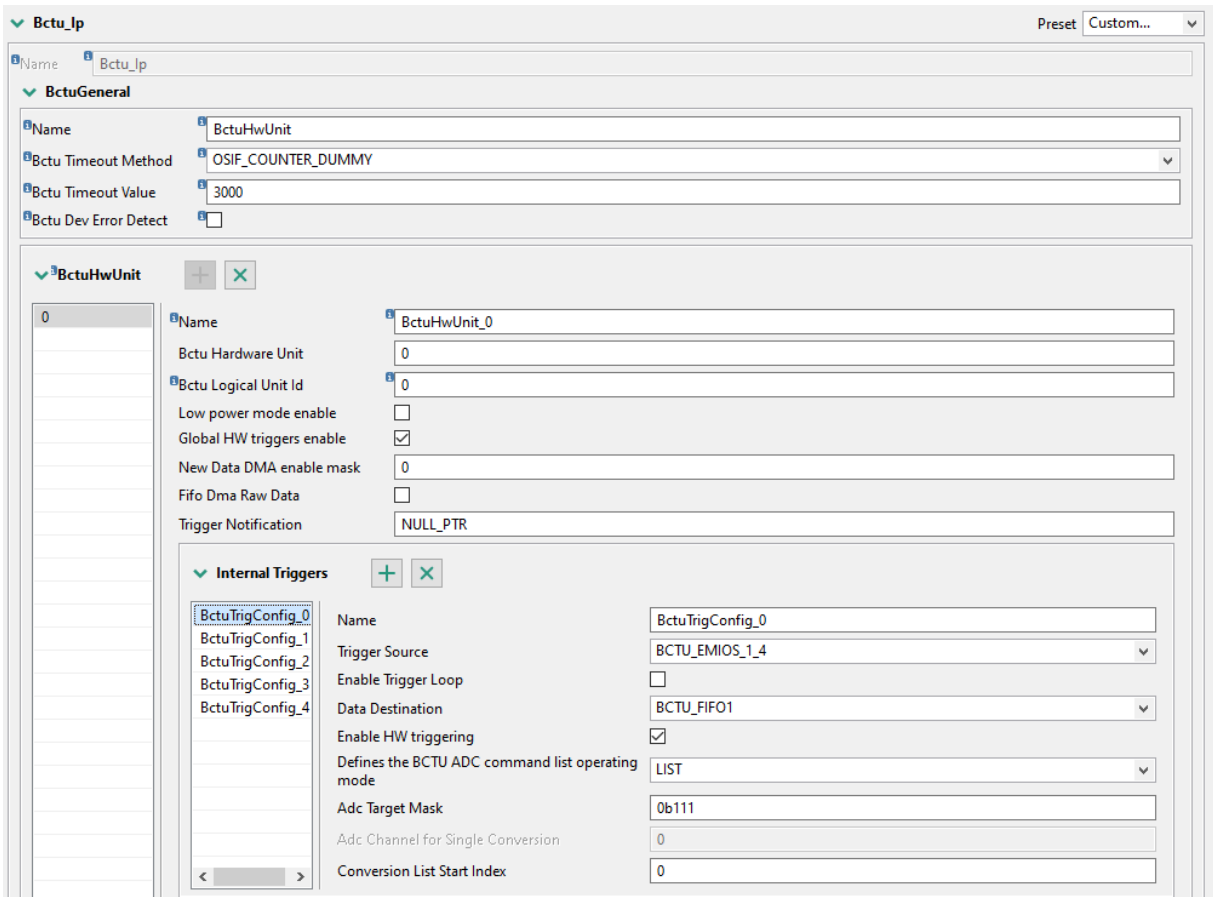
#### 4.2.4.2 BCTU

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 CH4 is selected as a trigger for phase current measurement of the Motor#1, eMIOS1 CH12 is selected as a trigger for phase current measurement of the Motor#2, eMIOS1 CH5 is selected as a trigger for resolver sine and cosine signals measurement of the Motor#1, eMIOS1 CH13 is selected as a trigger for resolver sine and cosine signals measurement of the Motor#2 and eMIOS1 CH14 is selected as a trigger for single conversion of the forward voltage of the Motor#2. FIFO1 is selected for *“Data Destination”* for *all mentioned triggers*. First four triggers are configured as a list of parallel conversions ADC0, ADC1 and ADC2 in *“Adc Target Mask”*, but last trigger is configured as a single conversion. Lists of ADC channels are 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 15 and *“Interrupt Notification”* is enabled.

When first trigger comes (eMIOS1 CH4), parallel conversion of the first list of three items starts (phase currents of the Motor#1). Then, measurement process is waiting until next trigger comes. Second parallel conversion (the second list) of the three items (phase currents of the Motor#2) starts when second trigger comes (eMIOS1 CH12). Then, measurement process is waiting again, until next trigger comes. Third parallel conversion (the third list) of the three items (resolver sine, cosine signals of the Motor#1 and temperature) starts when third trigger comes (eMIOS1 CH5). The measurement process is waiting again until next trigger comes. Fourth parallel conversion (the fourth list) of the three items (resolver sine, cosine signals of the Motor#2 and battery voltage) starts when fourth trigger comes (eMIOS1 CH13). Execution of first twelve measurements is purely driven by eMIOS triggers and timing of measurement of individual list items is set according to various conditions i.e. time shift between eMIOS1 CH0 and CH8 time-bases channels with propagation delay or envelopes extraction of sine and cosine signals of resolver sensors.

Conversion of following three items (DC bus voltages of the Motor#1 and Motor#2 and forward voltage of the Motor#1) is executed shortly after measurement of last three items. In this case, measurement is not waiting on next trigger. Last measurement (forward voltage of the Motor#2) is only single conversion driven by trigger (eMIOS1 CH14). Once all results have been stored into the 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.



Bctu_Ip

Preset: Custom...

Name: Bctu_Ip

BctuGeneral

| | |
|---|---|
| Name | BctuHwUnit |
| Bctu Timeout Method | OSIF_COUNTER_DUMMY |
| Bctu Timeout Value | 3000 |
| Bctu Dev Error Detect | ☐ |

BctuHwUnit

0

| | |
|---|---|
| Name | BctuHwUnit_0 |
| Bctu Hardware Unit | 0 |
| Bctu Logical Unit Id | 0 |
| Low power mode enable | ☐ |
| Global HW triggers enable | ✓ |
| New Data DMA enable mask | 0 |
| Fifo Dma Raw Data | ☐ |
| Trigger Notification | NULL_PTR |

Internal Triggers

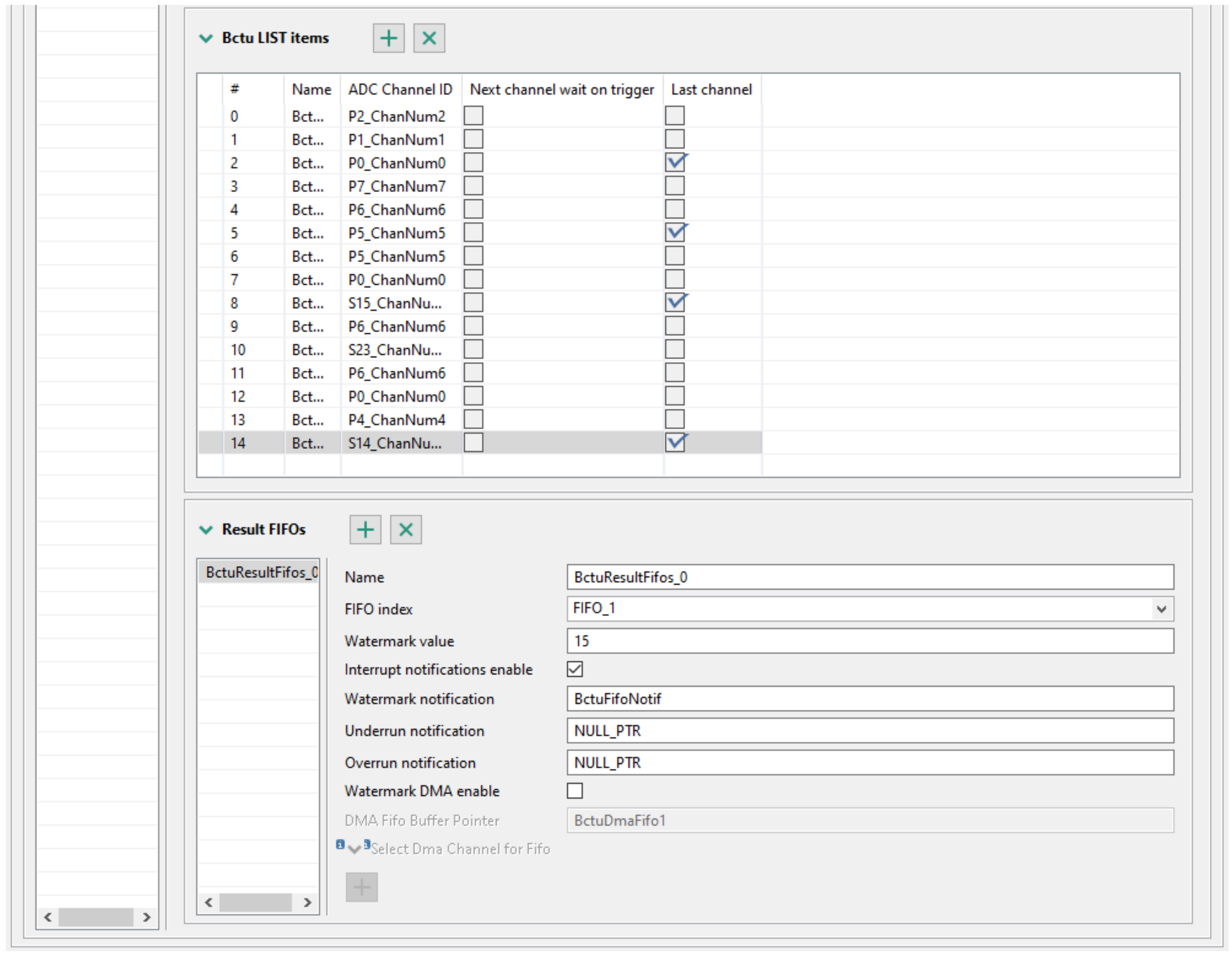
BctuTrigConfig_0
BctuTrigConfig_1
BctuTrigConfig_2
BctuTrigConfig_3
BctuTrigConfig_4

| | |
|---|---|
| Name | BctuTrigConfig_0 |
| Trigger Source | BCTU_EMIOS_1_4 |
| Enable Trigger Loop | ☐ |
| Data Destination | BCTU_FIFO1 |
| Enable HW triggering | ✓ |
| Defines the BCTU ADC command list operating mode | LIST |
| Adc Target Mask | 0b111 |
| Adc Channel for Single Conversion | 0 |
| Conversion List Start Index | 0 |

Figure 39. 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 |



| # | Name | ADC Channel ID | Next channel wait on trigger | Last channel |
|---|---|---|---|---|
| 0 | Bct... | P2_ChanNum2 | ☐ | ☐ |
| 1 | Bct... | P1_ChanNum1 | ☐ | ☐ |
| 2 | Bct... | P0_ChanNum0 | ☐ | ✓ |
| 3 | Bct... | P7_ChanNum7 | ☐ | ☐ |
| 4 | Bct... | P6_ChanNum6 | ☐ | ☐ |
| 5 | Bct... | P5_ChanNum5 | ☐ | ✓ |
| 6 | Bct... | P5_ChanNum5 | ☐ | ☐ |
| 7 | Bct... | P0_ChanNum0 | ☐ | ☐ |
| 8 | Bct... | S15_ChanNu... | ☐ | ✓ |
| 9 | Bct... | P6_ChanNum6 | ☐ | ☐ |
| 10 | Bct... | S23_ChanNu... | ☐ | ☐ |
| 11 | Bct... | P6_ChanNum6 | ☐ | ☐ |
| 12 | Bct... | P0_ChanNum0 | ☐ | ☐ |
| 13 | Bct... | P4_ChanNum4 | ☐ | ☐ |
| 14 | Bct... | S14_ChanNu... | ☐ | ✓ |

| | |
|---|---|
| Name | BctuResultFifos_0 |
| FIFO index | FIFO_1 |
| Watermark value | 15 |
| Interrupt notifications enable | ✓ |
| Watermark notification | BctuFifoNotif |
| Underrun notification | NULL_PTR |
| Overrun notification | NULL_PTR |
| Watermark DMA enable | ☐ |
| DMA Fifo Buffer Pointer | BctuDmaFifo1 |

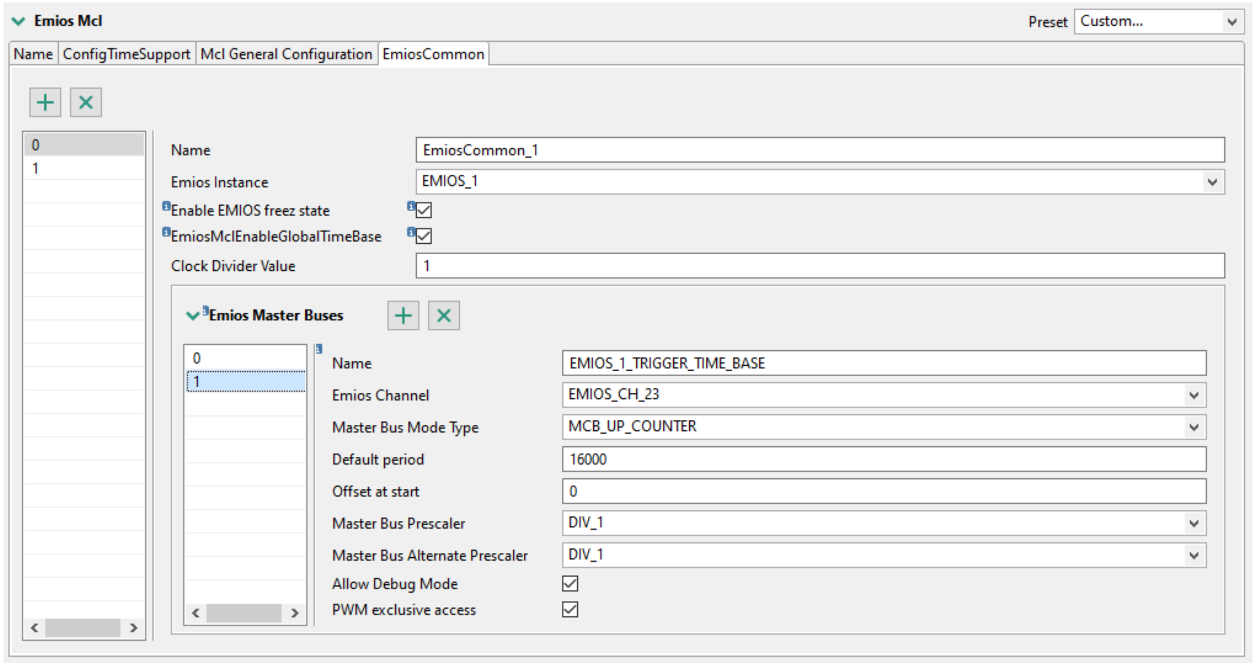
Figure 40. BCTU list and FIFO configuration

**Example 5.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);
...
}
  voidBctu_FIFO1_WatermarkNotification (void)
{
...
 mCount = 0;
 while (Bctu_Ip_GetFifoCount(0U, 0U))
 {
 Bctu_Ip_GetFifoResult(0U, 0U, &measuredValues[mCount++]);
 }
...
}
```

### 4.2.4.3 eMIOS

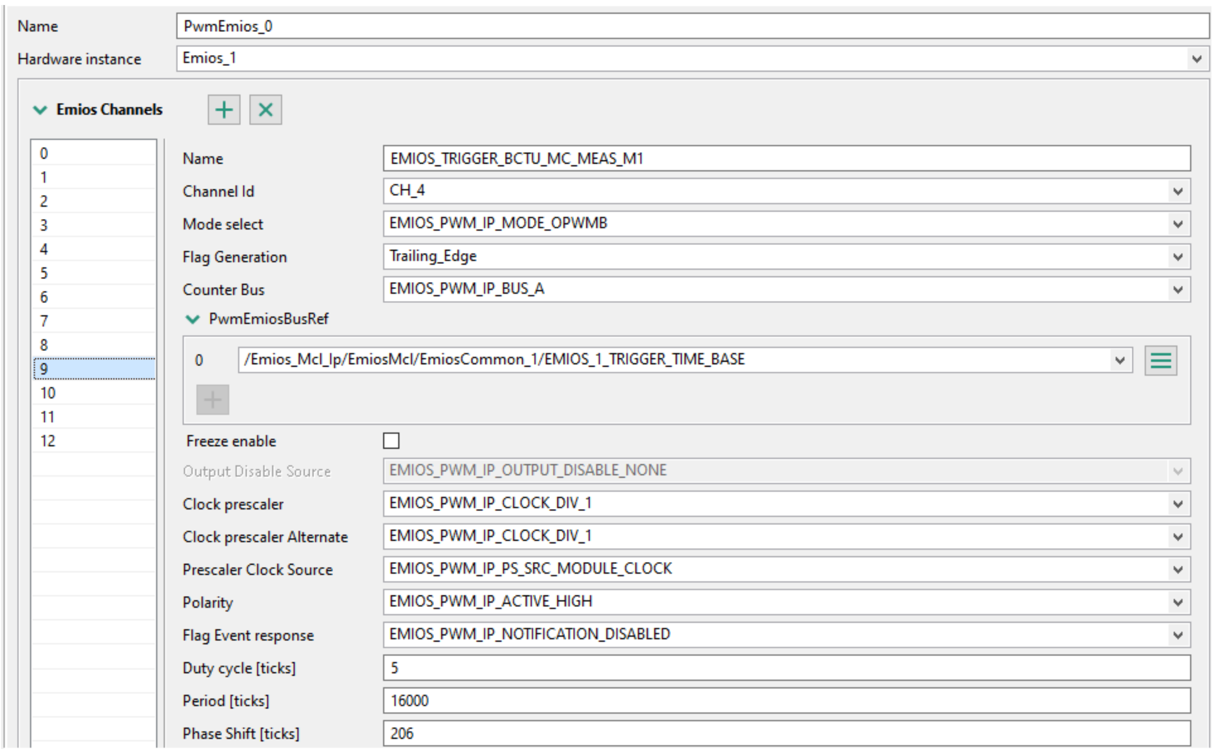
In this example, several eMIOS channels are engaged in triggering of BCTU. Namely, eMIOS1 CH4, CH5 for the Motor#1 and eMIOS1 CH12, CH13 and CH14 for the Motor#2 are configured to generate the triggers for BCTU in precise moment. Same modes and drivers are used as in the use case of PWM generation in chapter 4.2.3.1. Trigger channels use a global time base A (eMIOS1 CH23). This trigger time base is synchronized with both PWM time bases, however only Motor#1 time base is synchronized with trigger 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 all eMIOS channels for triggering purpose 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 all triggering eMIOS1 channel outputs, which can be visualized on the pin using TRGMUX. First trigger (eMIOS1 CH4) is generated 206 cycles after trigger time base reload due to delay of HW circuitry (gate drivers, safe logic, etc.). PWM time base reload of the Motor#1 and trigger time base reload are overlapping since there is no delay. Second trigger (eMIOS1 CH12) is generated 526 cycles after trigger time base reload, which reflects delay of PWM time base reload for the Motor#2 together with delay of HW circuitry of Motor#2. Third trigger (eMIOS1 CH5) is generated 1280 cycles after trigger time base reload, which is an exact moment of resolver sine and cosine envelop signal measurement of the Motor#1 and fourth trigger (eMIOS1 CH13) is generated 1920 cycles after trigger time base reload, which is an exact moment of resolver sine and cosine envelop signal measurement as well of the Motor#2. Last trigger (eMIOS1 CH14) is generated 2560 cycles after trigger time base reload, so enough time elapsed between last execution and this one can start. 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().



| | |
|---|---|
| Name | EmiosCommon_1 |
| Emios Instance | EMIOS_1 |
| Enable EMIOS freez state | ✓ |
| EmiosMclEnableGlobalTimeBase | ✓ |
| Clock Divider Value | 1 |

| | |
|---|---|
| Name | EMIOS_1_TRIGGER_TIME_BASE |
| Emios Channel | EMIOS_CH_23 |
| Master Bus Mode Type | MCB_UP_COUNTER |
| Default period | 16000 |
| Offset at start | 0 |
| Master Bus Prescaler | DIV_1 |
| Master Bus Alternate Prescaler | DIV_1 |
| Allow Debug Mode | ✓ |
| PWM exclusive access | ✓ |

**Figure 41. eMIOS trigger time-base configuration**

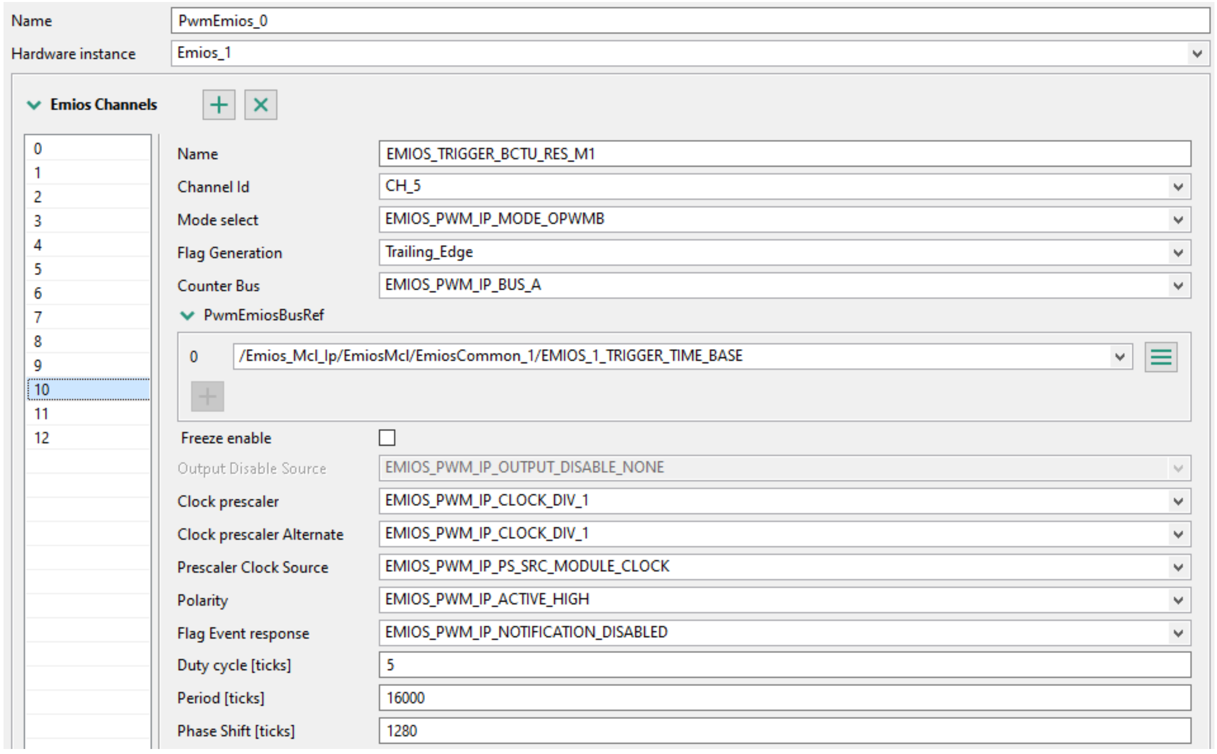


| | |
|---|---|
| Name | PwmEmios_0 |
| Hardware instance | Emios_1 |

Emios Channels: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

| | |
|---|---|
| Name | EMIOS_TRIGGER_BCTU_MC_MEAS_M1 |
| Channel Id | CH_4 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_A |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_TRIGGER_TIME_BASE |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_HIGH |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| Duty cycle [ticks] | 5 |
| Period [ticks] | 16000 |
| Phase Shift [ticks] | 206 |

Figure 42. eMIOS phase current trigger configuration for the Motor#1

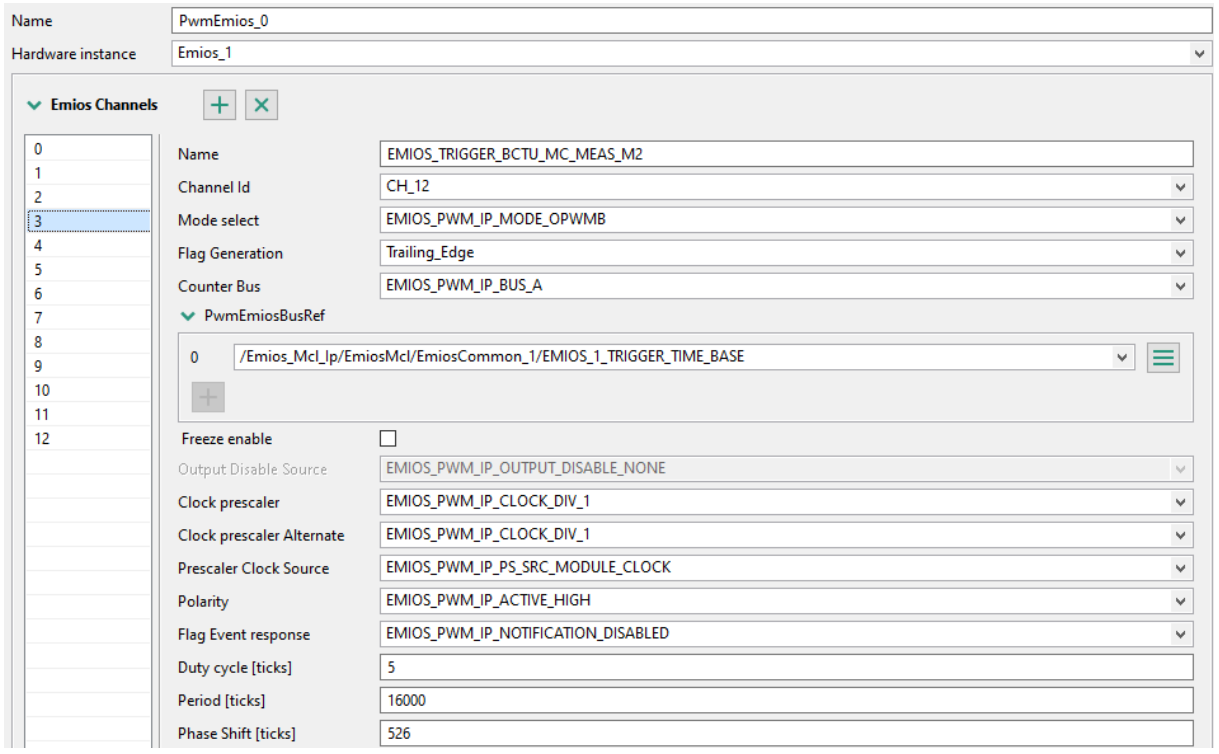


| | |
|---|---|
| Name | PwmEmios_0 |
| Hardware instance | Emios_1 |

Emios Channels: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

| | |
|---|---|
| Name | EMIOS_TRIGGER_BCTU_RES_M1 |
| Channel Id | CH_5 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_A |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_TRIGGER_TIME_BASE |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_HIGH |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| Duty cycle [ticks] | 5 |
| Period [ticks] | 16000 |
| Phase Shift [ticks] | 1280 |

Figure 43. eMIOS resolver trigger configuration for the Motor#1

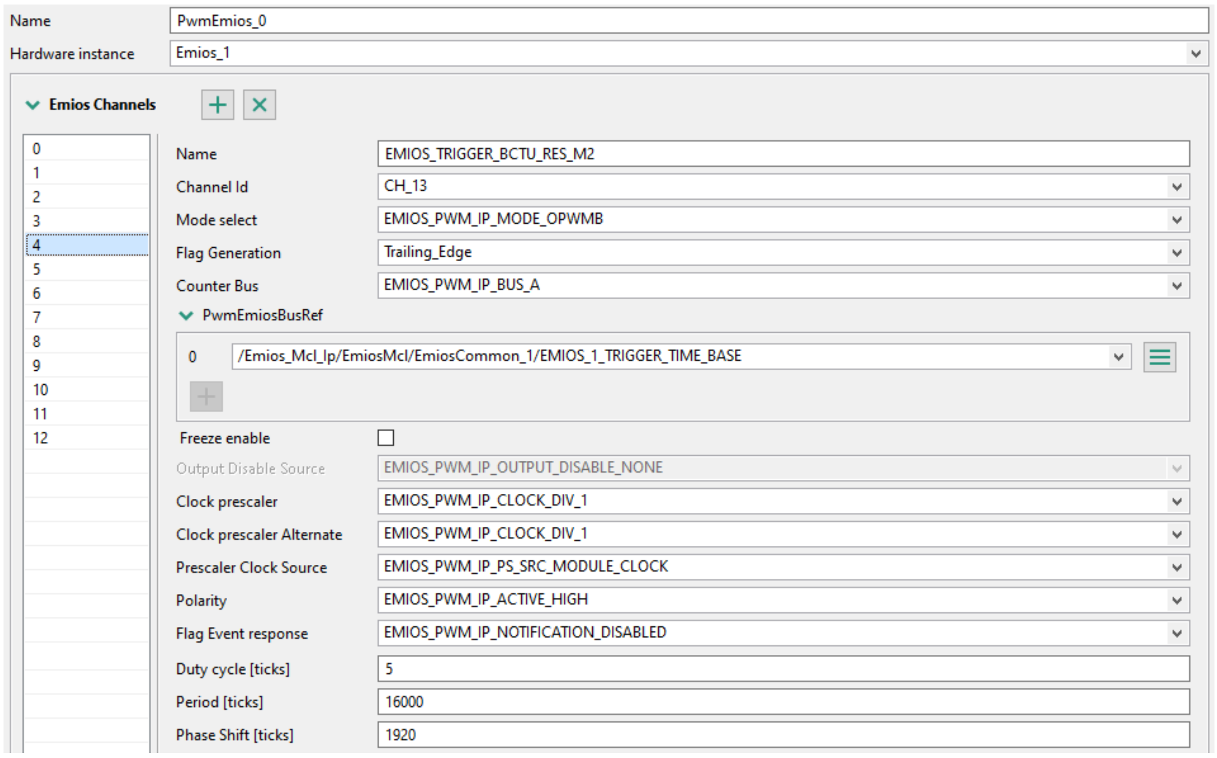


| | |
|---|---|
| Name | PwmEmios_0 |
| Hardware instance | Emios_1 |

Emios Channels: 0, 1, 2, 3 (selected), 4, 5, 6, 7, 8, 9, 10, 11, 12

| | |
|---|---|
| Name | EMIOS_TRIGGER_BCTU_MC_MEAS_M2 |
| Channel Id | CH_12 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_A |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_TRIGGER_TIME_BASE |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_HIGH |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| Duty cycle [ticks] | 5 |
| Period [ticks] | 16000 |
| Phase Shift [ticks] | 526 |

Figure 44. eMIOS phase current trigger configuration for the Motor#2

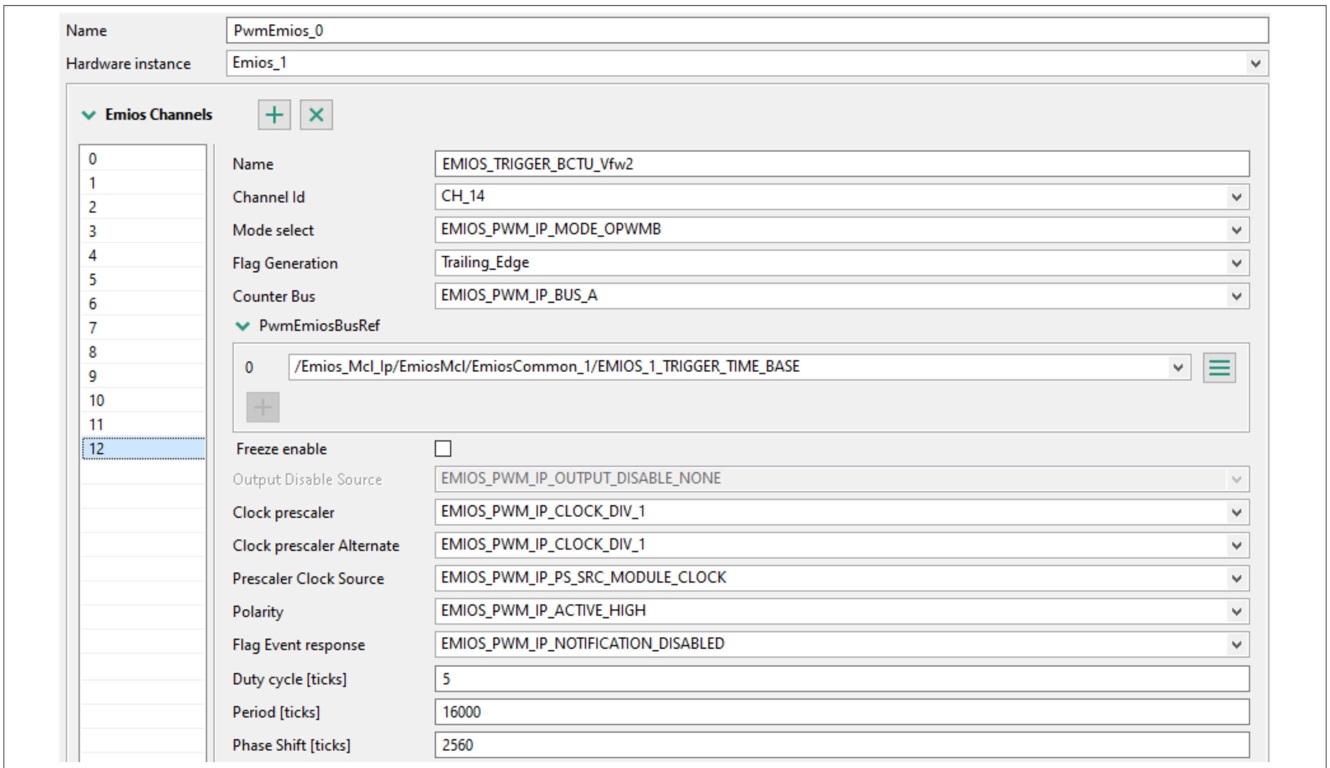


| | |
|---|---|
| Name | PwmEmios_0 |
| Hardware instance | Emios_1 |

Emios Channels: 0, 1, 2, 3, 4 (selected), 5, 6, 7, 8, 9, 10, 11, 12

| | |
|---|---|
| Name | EMIOS_TRIGGER_BCTU_RES_M2 |
| Channel Id | CH_13 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_A |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_TRIGGER_TIME_BASE |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_HIGH |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| Duty cycle [ticks] | 5 |
| Period [ticks] | 16000 |
| Phase Shift [ticks] | 1920 |

Figure 45. eMIOS resolver trigger configuration for the Motor#2



Name: PwmEmios_0
Hardware instance: Emios_1

Emios Channels: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

| | |
|---|---|
| Name | EMIOS_TRIGGER_BCTU_Vfw2 |
| Channel Id | CH_14 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_A |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_TRIGGER_TIME_BASE |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_HIGH |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| Duty cycle [ticks] | 5 |
| Period [ticks] | 16000 |
| Phase Shift [ticks] | 2560 |

**Figure 46. eMIOS trigger configuration for single conversion of forward voltage of the Motor#2**

### Example 6.eMIOS API for PWM

```
void main (void)
{
    /*****************************************************************
    * eMios Driver
    ******************************************************************/
    Emios_Mcl_Ip_Init(1U, &Emios_Mcl_Ip_1_Config_BOARD_INITPERIPHERALS);
    /* Trigger channels of eMIOS */
    Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch4);  /* TRG_I_M1  */
    Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch5);  /* TRG_Res_M1 */

    Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch12);  /* TRG_I_M2  */
    Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch13);  /* TRG_Res_M2 */

    Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch14);  /* TRG_Ufw  */
    /*Enable eMIOS clock at last to ensure the correct trigger order*
    Emios_Mcl_Ip_ConfigureGlobalTimebase(1U, TRUE);
...
}
```

### 4.2.5 Resolver sensor

This PMSM motor application uses the resolver for rotor position sensing for both motors. 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].

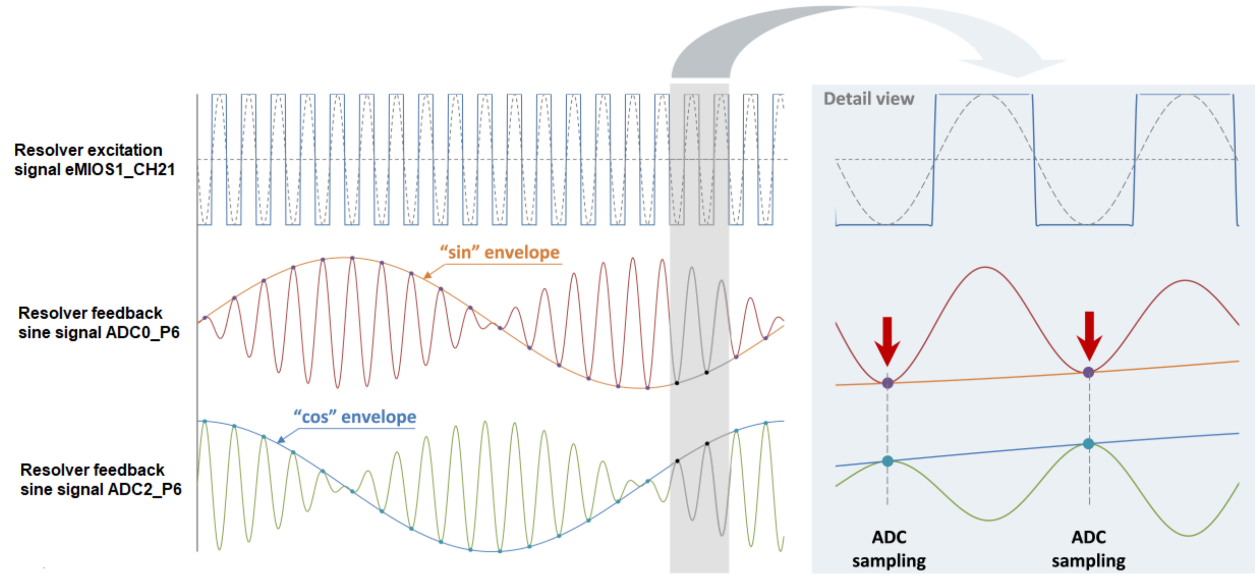


**Figure 47. An envelope extractor of resolver feedback signals**

Resolver systems utilize an Angle Tracking Observer (ATO), see Figure 48 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].

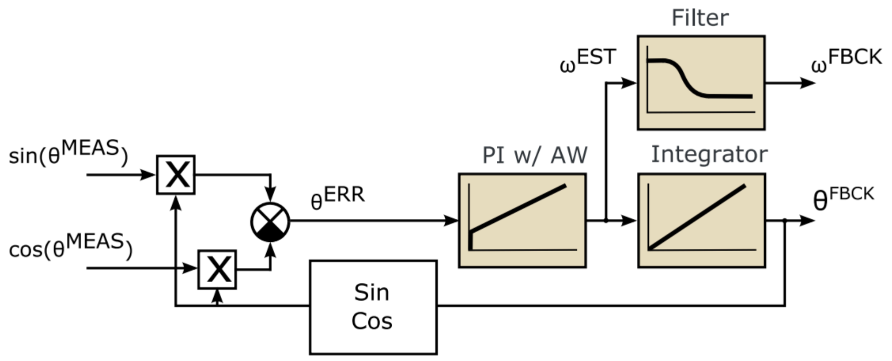


**Figure 48. Angle tracking observer for resolver sensor**

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

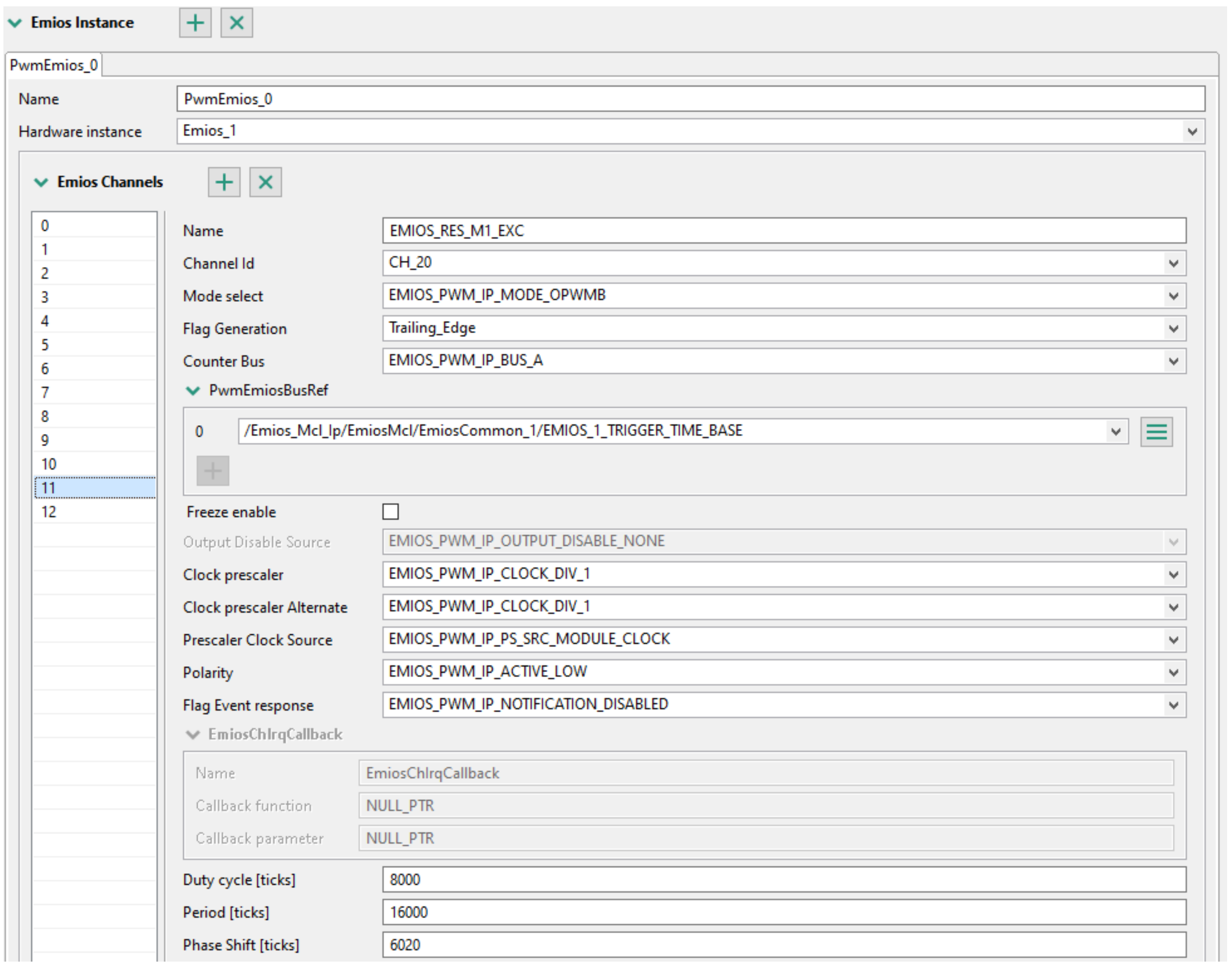
$$\sin(\theta_{MEAS})\cos(\theta_{FBCK}) - \sin(\theta_{FBCK})\cos(\theta_{MEAS}) = \sin(\theta_{MEAS} - \theta_{FBCK}) \qquad (10)$$

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 resolver. The other one must be disabled. To enable one of them, a dedicated macro for particular motor (ENCODER 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 for both motors. 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 for the Motor#1 is depicted in Figure 41.

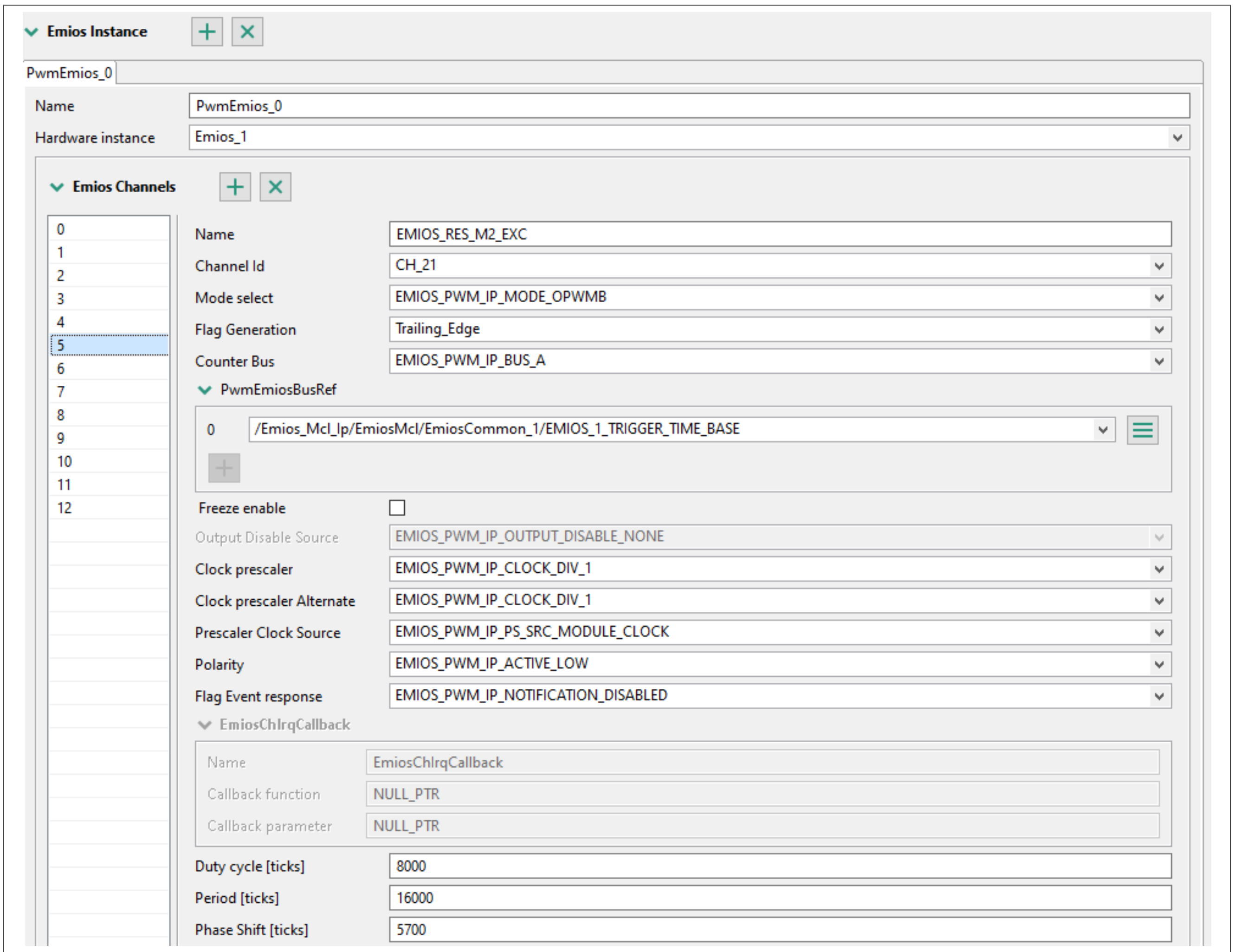


|  |  |
|---|---|
| Name | PwmEmios_0 |
| Hardware instance | Emios_1 |

|  |  |
|---|---|
| Name | EMIOS_RES_M1_EXC |
| Channel Id | CH_20 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_A |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_TRIGGER_TIME_BASE |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_LOW |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| EmiosChIrqCallback Name | EmiosChIrqCallback |
| Callback function | NULL_PTR |
| Callback parameter | NULL_PTR |
| Duty cycle [ticks] | 8000 |
| Period [ticks] | 16000 |
| Phase Shift [ticks] | 6020 |

**Figure 49. Resolver excitation channel configuration for Motor#1**

eMIOS1 CH20 is configured to generate the PWM signal for resolver excitation of the Motor#1. Motor#2 uses eMIOS1 CH21 as an excitation signal for resolver sensor. Channels operate in Output PWM Buffered (OPWMB) mode similarly to PWM signals. Both channels select global time base A as a counter bus and time base settings are also referenced through "*PwmEmiosBusRef*" field. Channels are 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. Both resolver excitation signals operate with 50% duty cycle and they are shifted in order to match sampling of sine cosine envelope signals with excitation signals. The eMIOS1 CH5 and eMIOS1 CH13 are configured to generate the triggers for BCTU in precise moment, where individual envelops sine and cosine signals are measured. 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 13. Settings are applied by calling Emios_Pwm_Ip_InitChannel(), Emios_Mcl_Ip_Init() and Emios_Mcl_Ip_ConfigureGlobal Timebase(), where after calling the Emios_Mcl_Ip_ConfigureGlobalTimebase() time-base counting is started. PWM signal is static and is not changing during the runtime.


Emios Instance

PwmEmios_0

Name: PwmEmios_0

Hardware instance: Emios_1

Emios Channels: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

| | |
|---|---|
| Name | EMIOS_RES_M2_EXC |
| Channel Id | CH_21 |
| Mode select | EMIOS_PWM_IP_MODE_OPWMB |
| Flag Generation | Trailing_Edge |
| Counter Bus | EMIOS_PWM_IP_BUS_A |
| PwmEmiosBusRef 0 | /Emios_Mcl_Ip/EmiosMcl/EmiosCommon_1/EMIOS_1_TRIGGER_TIME_BASE |
| Freeze enable | ☐ |
| Output Disable Source | EMIOS_PWM_IP_OUTPUT_DISABLE_NONE |
| Clock prescaler | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Clock prescaler Alternate | EMIOS_PWM_IP_CLOCK_DIV_1 |
| Prescaler Clock Source | EMIOS_PWM_IP_PS_SRC_MODULE_CLOCK |
| Polarity | EMIOS_PWM_IP_ACTIVE_LOW |
| Flag Event response | EMIOS_PWM_IP_NOTIFICATION_DISABLED |
| EmiosChIrqCallback: Name | EmiosChIrqCallback |
| Callback function | NULL_PTR |
| Callback parameter | NULL_PTR |
| Duty cycle [ticks] | 8000 |
| Period [ticks] | 16000 |
| Phase Shift [ticks] | 5700 |

Figure 50. Resolver excitation channel configuration for Motor#2

**Example 7.eMIOS API for PWM**

```
void main (void)
  {
  ...
  /***************************************************************
  * eMios Driver
  ***************************************************************/
  Emios_Mcl_Ip_Init(1U, &Emios_Mcl_Ip_1_Config_BOARD_INITPERIPHERALS);
  #if RESOLVER1 && Motor1
  Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch20); /* EXC M1 */
   #endif

  #if RESOLVER2 && Motor2
  Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch21); /* EXC M2 */
```

```
    #endif
    Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch5); /* TRG_Res_M1 */
    Emios_Pwm_Ip_InitChannel(1U, &Emios_Pwm_Ip_I1_Ch13); /* TRG_Res_M2 */
    ...
    /*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 chapter 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 chapter 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 51. 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.

In order to get the rotor position, encoder signals HALL_A_ENC_A and HALL_B_ENC_B of the Motor#1 and HALL_D_ENC_D and HALL_E_ENC_E of the Motor#2 are brought to the LCU. LCU preprocess them individually per motor 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.

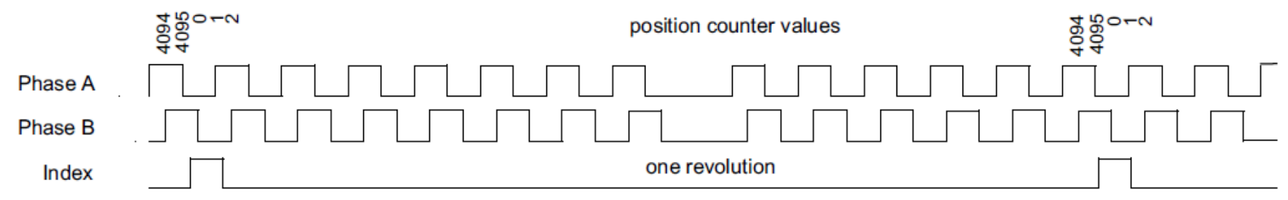


**Figure 51. Output signals of the 1024 pulses Encoder**

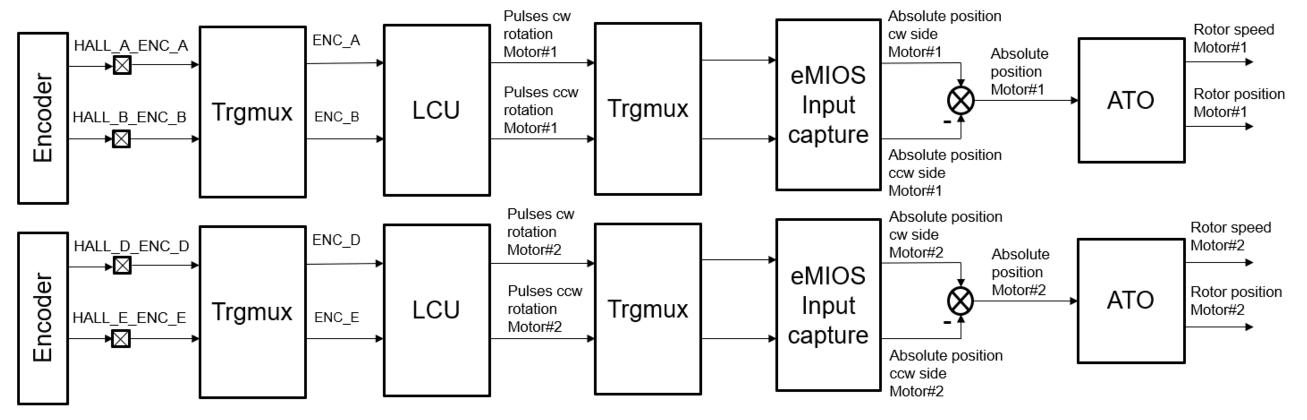

Encoder
HALL_A_ENC_A
HALL_B_ENC_B
Trgmux
ENC_A
ENC_B
LCU
Pulses cw rotation Motor#1
Pulses ccw rotation Motor#1
Trgmux
eMIOS Input capture
Absolute position cw side Motor#1
Absolute position ccw side Motor#1
Absolute position Motor#1
ATO
Rotor speed Motor#1
Rotor position Motor#1
Encoder
HALL_D_ENC_D
HALL_E_ENC_E
Trgmux
ENC_D
ENC_E
LCU
Pulses cw rotation Motor#2
Pulses ccw rotation Motor#2
Trgmux
eMIOS Input capture
Absolute position cw side Motor#2
Absolute position ccw side Motor#2
Absolute position Motor#2
ATO
Rotor speed Motor#2
Rotor position Motor#2

**Figure 52. Peripherals interconnection for quadrature encoder**

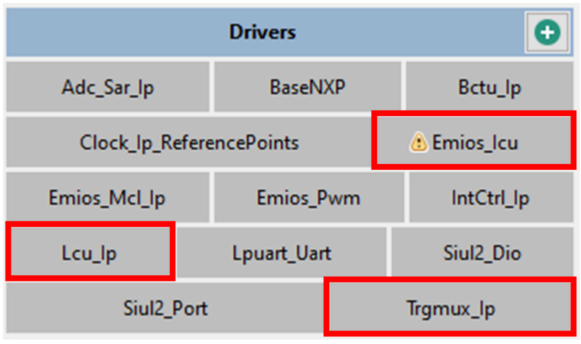


**Figure 53. Drivers for quadrature decoder feature**

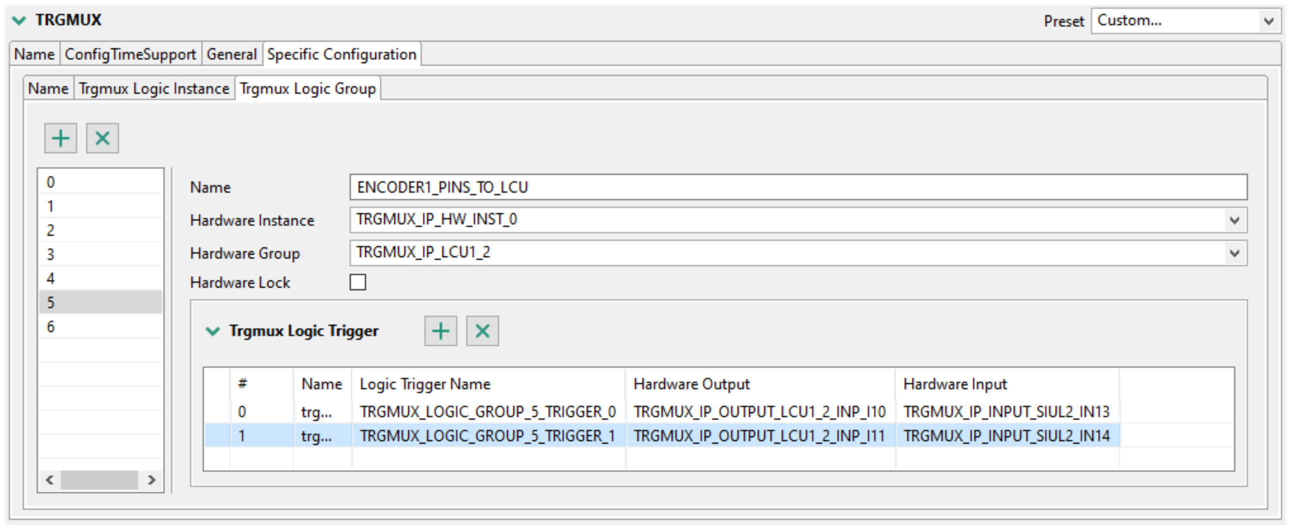
***Note:*** *Only one sensor routine can be enabled at the same time, either encoder or resolver. The other one must be disabled. To enable one of them, a dedicated macro for particular motor (ENCODER or RESOLVER) has to be set true.*

### 4.2.6.1 TRIGGER MUX

TRGMUX ensures a connection between Input pins and LCU and between eMIOS and LCU.

Settings within the *“Hardware Group”* ENCODER1_PINS_TO_LCU connects PTA19(TRGMUX_IN13) and PTA20(TRGMUX_IN14) to LCU1, LC2 inputs 0-1. This setting is intended to be used for the Motor#1. Settings within the *“Hardware Group”* ENCODER1_LCU_TO_EMIOS connects LCU1 LC2 outputs 2-3 to eMIOS0 inputs of channels 3-4. The setting is applied by calling Trgmux_Ip_Init() function. This setting is intended to be used for the Motor#1 as well.

Settings within the *“Hardware Group”* ENCODER2_PINS_TO_LCU connects PTD2(TRGMUX_IN5) and PTD3(TRGMUX_IN4) to LCU1, LC0 inputs 0-1. This setting is intended to be used for the Motor#2. Settings within the *“Hardware Group”* ENCODER2_LCU_TO_EMIOS connects LCU1 LC0 outputs 2-3 to eMIOS0 inputs of channels 5-6. All settings are applied by calling Trgmux_Ip_Init() function.



**TRGMUX** Preset: Custom...

Name | ConfigTimeSupport | General | Specific Configuration

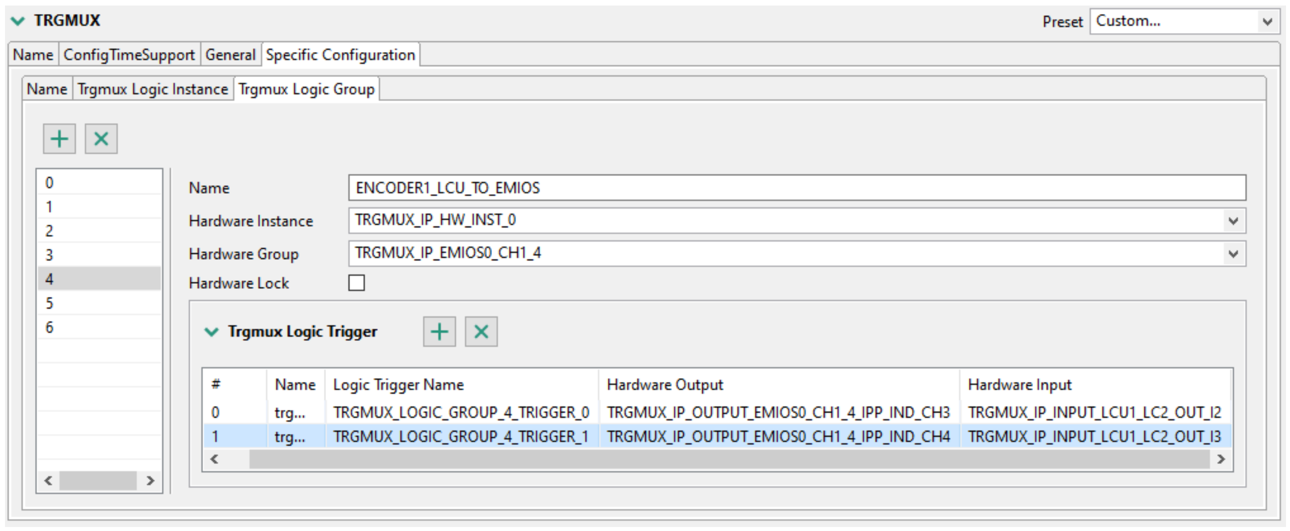
Name | Trgmux Logic Instance | Trgmux Logic Group

0, 1, 2, 3, 4, 5, 6

| | |
|---|---|
| Name | ENCODER1_PINS_TO_LCU |
| Hardware Instance | TRGMUX_IP_HW_INST_0 |
| Hardware Group | TRGMUX_IP_LCU1_2 |
| Hardware Lock | ☐ |

**Trgmux Logic Trigger**

| # | Name | Logic Trigger Name | Hardware Output | Hardware Input |
|---|---|---|---|---|
| 0 | trg... | TRGMUX_LOGIC_GROUP_5_TRIGGER_0 | TRGMUX_IP_OUTPUT_LCU1_2_INP_I10 | TRGMUX_IP_INPUT_SIUL2_IN13 |
| 1 | trg... | TRGMUX_LOGIC_GROUP_5_TRIGGER_1 | TRGMUX_IP_OUTPUT_LCU1_2_INP_I11 | TRGMUX_IP_INPUT_SIUL2_IN14 |



**TRGMUX** Preset: Custom...

Name | ConfigTimeSupport | General | Specific Configuration

Name | Trgmux Logic Instance | Trgmux Logic Group

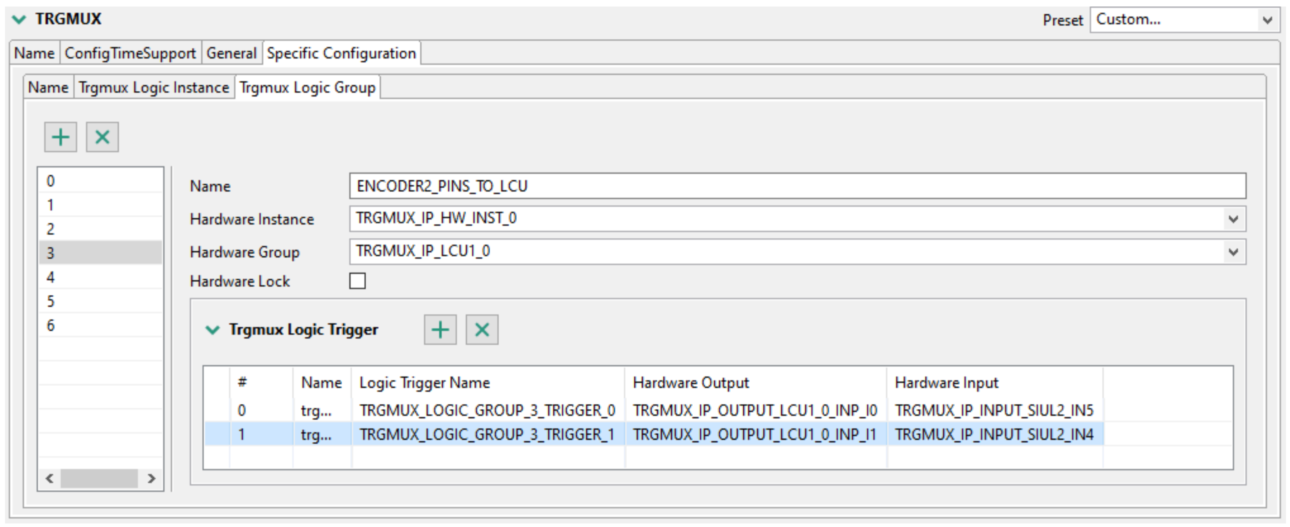
0, 1, 2, 3, 4, 5, 6

| | |
|---|---|
| Name | ENCODER1_LCU_TO_EMIOS |
| Hardware Instance | TRGMUX_IP_HW_INST_0 |
| Hardware Group | TRGMUX_IP_EMIOS0_CH1_4 |
| Hardware Lock | ☐ |

**Trgmux Logic Trigger**

| # | Name | Logic Trigger Name | Hardware Output | Hardware Input |
|---|---|---|---|---|
| 0 | trg... | TRGMUX_LOGIC_GROUP_4_TRIGGER_0 | TRGMUX_IP_OUTPUT_EMIOS0_CH1_4_IPP_IND_CH3 | TRGMUX_IP_INPUT_LCU1_LC2_OUT_I2 |
| 1 | trg... | TRGMUX_LOGIC_GROUP_4_TRIGGER_1 | TRGMUX_IP_OUTPUT_EMIOS0_CH1_4_IPP_IND_CH4 | TRGMUX_IP_INPUT_LCU1_LC2_OUT_I3 |

**Figure 54. TRGMUX settings for quadrature decoder of the Motor#1**



**TRGMUX** Preset: Custom...

Name | ConfigTimeSupport | General | Specific Configuration

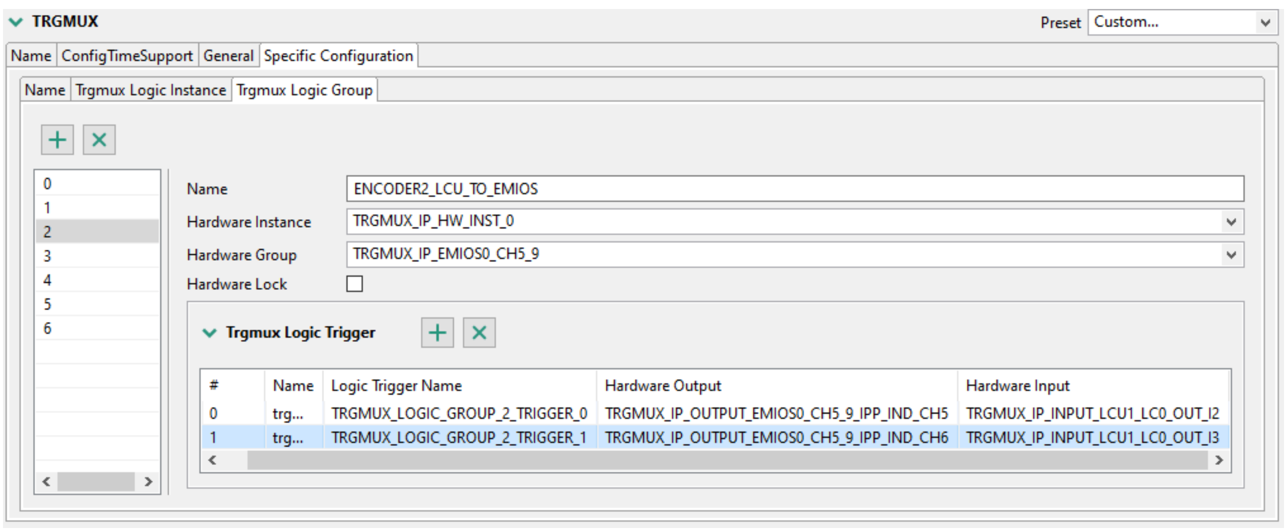
Name | Trgmux Logic Instance | Trgmux Logic Group

0, 1, 2, 3, 4, 5, 6

| | |
|---|---|
| Name | ENCODER2_PINS_TO_LCU |
| Hardware Instance | TRGMUX_IP_HW_INST_0 |
| Hardware Group | TRGMUX_IP_LCU1_0 |
| Hardware Lock | ☐ |

**Trgmux Logic Trigger**

| # | Name | Logic Trigger Name | Hardware Output | Hardware Input |
|---|---|---|---|---|
| 0 | trg... | TRGMUX_LOGIC_GROUP_3_TRIGGER_0 | TRGMUX_IP_OUTPUT_LCU1_0_INP_I0 | TRGMUX_IP_INPUT_SIUL2_IN5 |
| 1 | trg... | TRGMUX_LOGIC_GROUP_3_TRIGGER_1 | TRGMUX_IP_OUTPUT_LCU1_0_INP_I1 | TRGMUX_IP_INPUT_SIUL2_IN4 |



TRGMUX

Preset: Custom...

Name | ConfigTimeSupport | General | Specific Configuration

Name | Trgmux Logic Instance | Trgmux Logic Group

0, 1, 2, 3, 4, 5, 6

Name: ENCODER2_LCU_TO_EMIOS

Hardware Instance: TRGMUX_IP_HW_INST_0

Hardware Group: TRGMUX_IP_EMIOS0_CH5_9

Hardware Lock: ☐

Trgmux Logic Trigger

| # | Name | Logic Trigger Name | Hardware Output | Hardware Input |
|---|---|---|---|---|
| 0 | trg... | TRGMUX_LOGIC_GROUP_2_TRIGGER_0 | TRGMUX_IP_OUTPUT_EMIOS0_CH5_9_IPP_IND_CH5 | TRGMUX_IP_INPUT_LCU1_LC0_OUT_I2 |
| 1 | trg... | TRGMUX_LOGIC_GROUP_2_TRIGGER_1 | TRGMUX_IP_OUTPUT_EMIOS0_CH5_9_IPP_IND_CH6 | TRGMUX_IP_INPUT_LCU1_LC0_OUT_I3 |

**Figure 55. TRGMUX settings for quadrature decoder of the Motor#2**

### 4.2.6.2 LCU

In this example LCU1 instance is selected for preprocessing encoder signals phase A and phase B of the Motor#1 and phase D and phase E of the Motor#2. Same LCU features are used as in chapter 4.2.3.3 but, whole preprocessing is carried out in LC2 for the Motor#1 and in LC0 for the Motor#2. Configurations 10-13 In *“Lcu Logic Input”* section create a connection between LCU instance inputs and LC inputs for the Motor#1 and configurations 6-9 In *“Lcu Logic Input”* section create a connection between LCU instance inputs and LC inputs for the Motor#2. Motor#1 uses multiplexor inputs 10, 11(pins PTA19, PTA20), which are connected to LC2 input 0,1 and multiplexor feedback inputs 8, 9 (LC2 out 0,1), which are connected to LC2 input 2,3. Motor#2 uses multiplexor inputs 0,1(pins PTD2, PTD3), which are connected to LC0 input 0,1 and multiplexor feedback inputs 0,1 (LC0 out 0,1), which are connected to LC0 input 2,3.

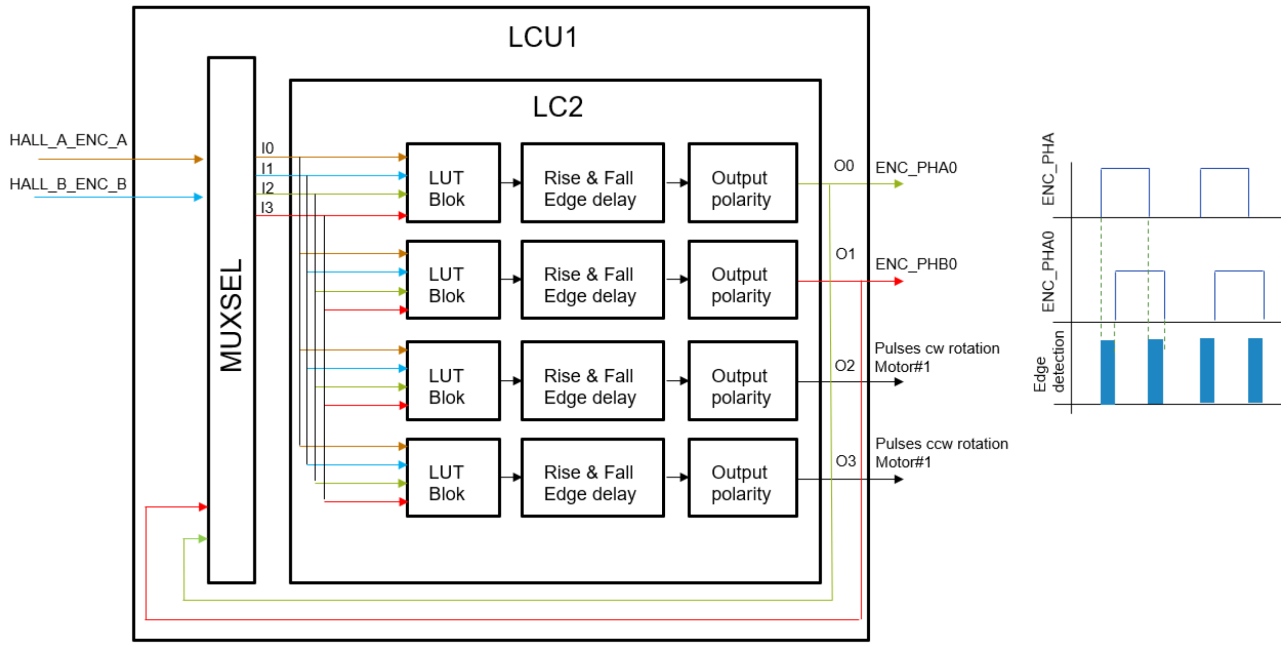
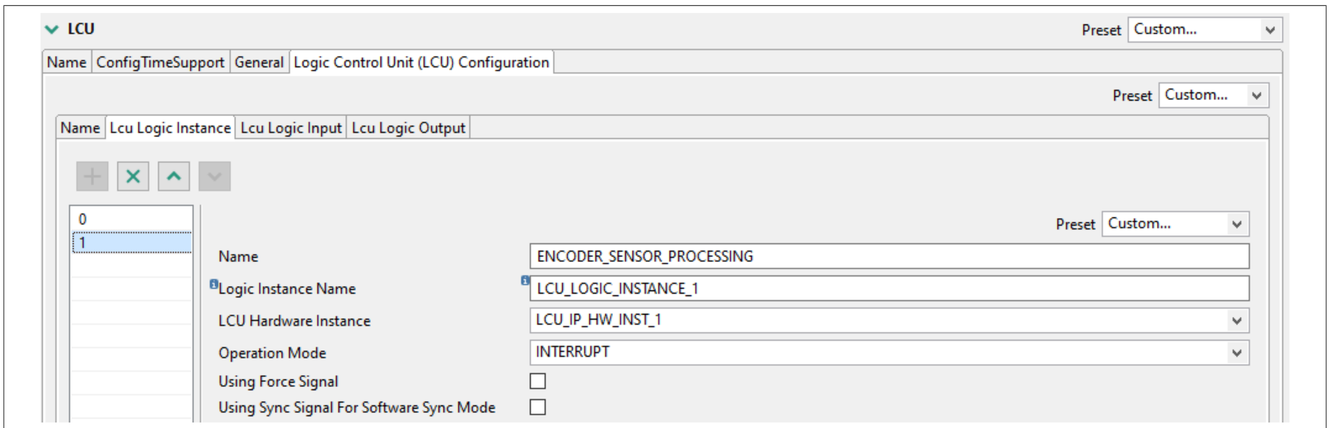


**Figure 56. Simplified LCU features block diagram of quadrature decoder mode for Motor#1**


LCU — Preset: Custom...

Tabs: Name | ConfigTimeSupport | General | Logic Control Unit (LCU) Configuration — Preset: Custom...

Tabs: Name | Lcu Logic Instance | Lcu Logic Input | Lcu Logic Output

Instances: 0, 1 — Preset: Custom...

| | |
|---|---|
| Name | ENCODER_SENSOR_PROCESSING |
| Logic Instance Name | LCU_LOGIC_INSTANCE_1 |
| LCU Hardware Instance | LCU_IP_HW_INST_1 |
| Operation Mode | INTERRUPT |
| Using Force Signal | ☐ |
| Using Sync Signal For Software Sync Mode | ☐ |

Figure 57. LCU instance configuration for quadrature decoder for both motor configuration

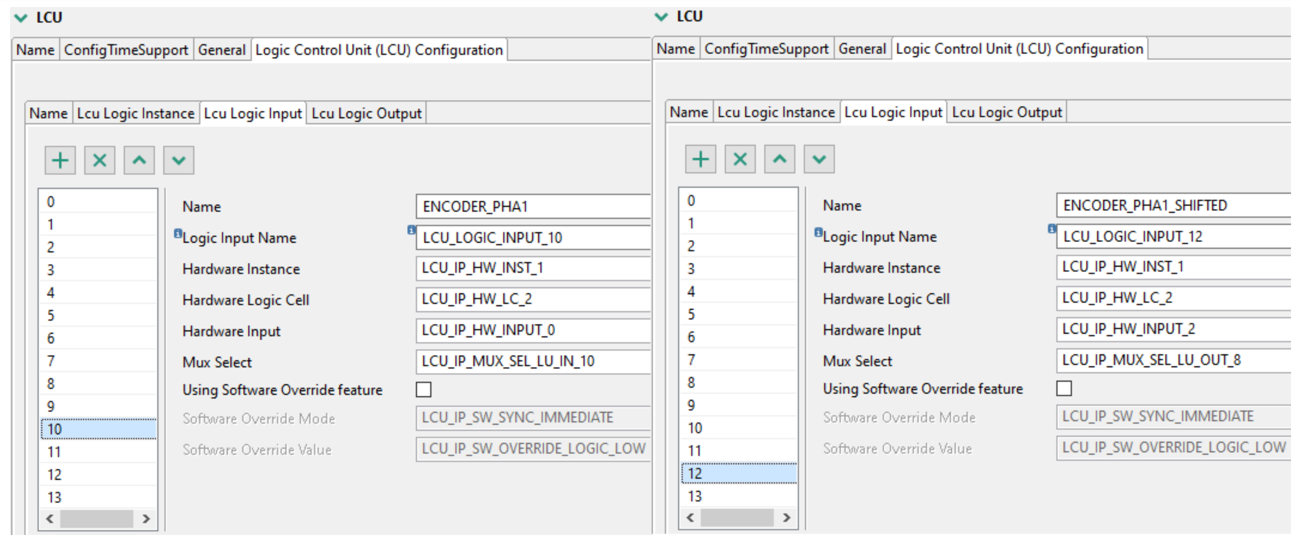
Outputs configurations are in section *"Lcu Logic Output"* configurations 16-19 is dedicated for the Motor#1 and configurations 12-15 is dedicated for the Motor#2. Look Up Table inputs 0,1 are mirrored on the outputs 0,1 of particular LC and rising and falling edges are delayed by digital filters. Look Up Table of outputs 2,3 of particular LC use information of all associated inputs. They detect edges of the encoder phases ENC_PHA and ENC_PHB using auxiliary signals ENC_PHA0 and ENC_PHB0 of the Motor#1 and ENC_PHD and ENC_PHE using auxiliary signals ENC_PHD0 and ENC_PHE0 of the Motor#2 as is depicted in waveform Figure 56 for only Motor#1. 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_PHA and ENC_PHB or 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 a protection against a noise on input pins, HALL_A_ENC_A, HALL_B_ENC_B pins of the Motor#1 and HALL_D_ENC_D and HALL_E_ENC_E pins of the Motor#2. All settings are applied by calling Lcu_Ip_Init() function.

Table 5. LUT configurations for LCU1 LC0 and LC2 for both motor configurations

| LC2_I3 | LC2_I2 | LC2_I1 | LC2_I0 | | LC2_O0 | LC2_O1 | LC2_O2 | LC2_O3 |
|---|---|---|---|---|---|---|---|---|
| ENC_PHA0 | ENC_PHB0 | ENC_PHA | ENC_PHB | | ENC_PHA0 | ENC_PHB0 | 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 |

**Table 5. LUT configurations for LCU1 LC0 and LC2 for both motor configurations***...continued*

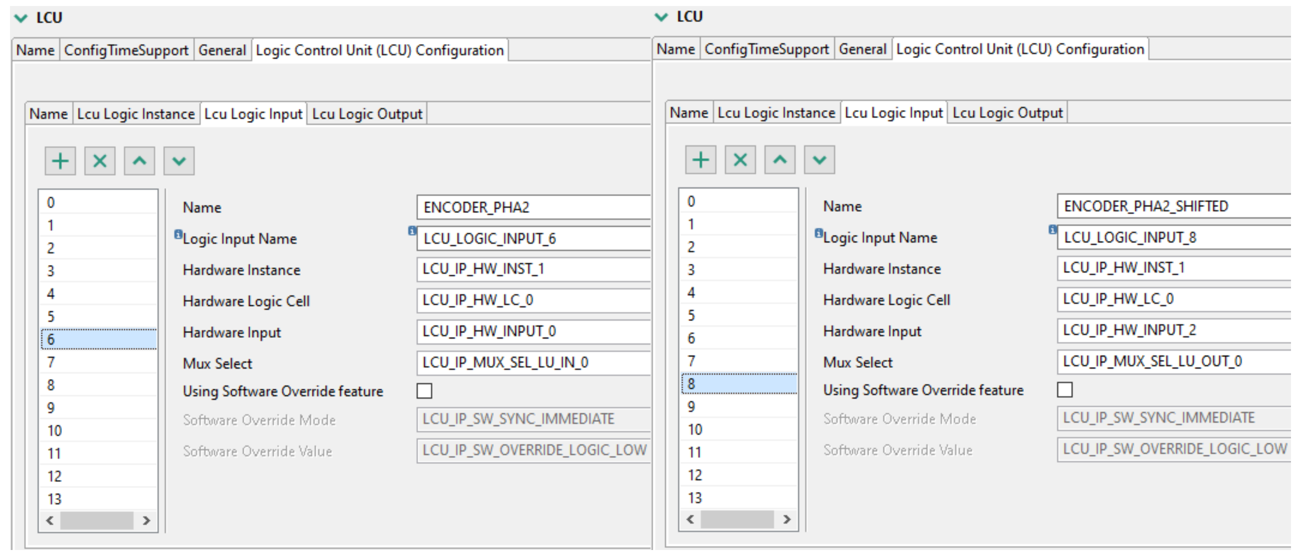
| LC2_I3 | LC2_I2 | LC2_I1 | LC2_I0 | | LC2_O0 | LC2_O1 | LC2_O2 | LC2_O3 |
|---|---|---|---|---|---|---|---|---|
| ENC_PHA0 | ENC_PHB0 | ENC_PHA | ENC_PHB | | ENC_PHA0 | ENC_PHB0 | Pulses cw | Pulses ccw |
| 1 | 1 | 1 | 0 | | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | | 1 | 1 | 0 | 0 |
| LUT Motor#1 | | | | | 0xAAAA | 0xCCCC | 0x4182 | 0x2814 |
| 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 Motor#2 | | | | | 0xAAAA | 0xCCCC | 0x4182 | 0x2814 |



| | |
|---|---|
| Name | ENCODER_PHA1 |
| Logic Input Name | LCU_LOGIC_INPUT_10 |
| Hardware Instance | LCU_IP_HW_INST_1 |
| Hardware Logic Cell | LCU_IP_HW_LC_2 |
| Hardware Input | LCU_IP_HW_INPUT_0 |
| Mux Select | LCU_IP_MUX_SEL_LU_IN_10 |
| Using Software Override feature | ☐ |
| Software Override Mode | LCU_IP_SW_SYNC_IMMEDIATE |
| Software Override Value | LCU_IP_SW_OVERRIDE_LOGIC_LOW |

| | |
|---|---|
| Name | ENCODER_PHA1_SHIFTED |
| Logic Input Name | LCU_LOGIC_INPUT_12 |
| Hardware Instance | LCU_IP_HW_INST_1 |
| Hardware Logic Cell | LCU_IP_HW_LC_2 |
| Hardware Input | LCU_IP_HW_INPUT_2 |
| Mux Select | LCU_IP_MUX_SEL_LU_OUT_8 |
| Using Software Override feature | ☐ |
| Software Override Mode | LCU_IP_SW_SYNC_IMMEDIATE |
| Software Override Value | LCU_IP_SW_OVERRIDE_LOGIC_LOW |

**Figure 58. LCU inputs configuration for quadrature decoder mode of the Motor#1**



LCU — Name | ConfigTimeSupport | General | Logic Control Unit (LCU) Configuration

Name | Lcu Logic Instance | Lcu Logic Input | Lcu Logic Output

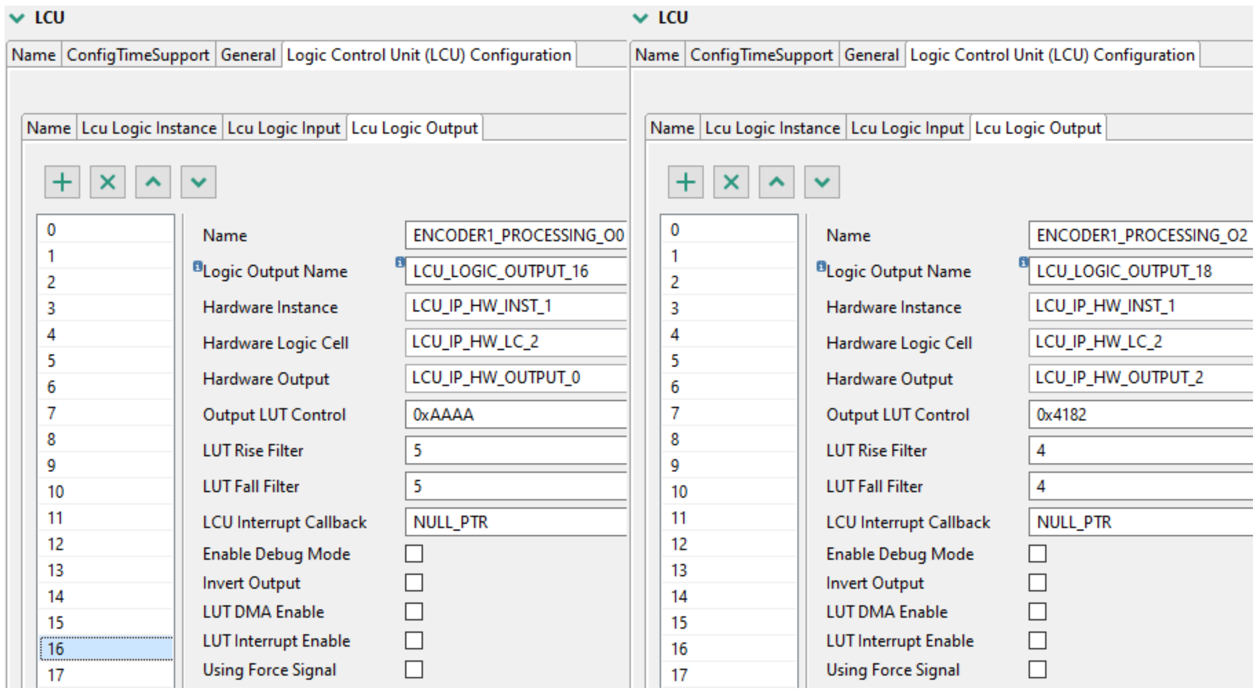
Left panel (selected item 6):

| | |
|---|---|
| Name | ENCODER_PHA2 |
| Logic Input Name | LCU_LOGIC_INPUT_6 |
| Hardware Instance | LCU_IP_HW_INST_1 |
| Hardware Logic Cell | LCU_IP_HW_LC_0 |
| Hardware Input | LCU_IP_HW_INPUT_0 |
| Mux Select | LCU_IP_MUX_SEL_LU_IN_0 |
| Using Software Override feature | ☐ |
| Software Override Mode | LCU_IP_SW_SYNC_IMMEDIATE |
| Software Override Value | LCU_IP_SW_OVERRIDE_LOGIC_LOW |

Right panel (selected item 8):

| | |
|---|---|
| Name | ENCODER_PHA2_SHIFTED |
| Logic Input Name | LCU_LOGIC_INPUT_8 |
| Hardware Instance | LCU_IP_HW_INST_1 |
| Hardware Logic Cell | LCU_IP_HW_LC_0 |
| Hardware Input | LCU_IP_HW_INPUT_2 |
| Mux Select | LCU_IP_MUX_SEL_LU_OUT_0 |
| Using Software Override feature | ☐ |
| Software Override Mode | LCU_IP_SW_SYNC_IMMEDIATE |
| Software Override Value | LCU_IP_SW_OVERRIDE_LOGIC_LOW |

**Figure 59. LCU inputs configuration for quadrature decoder mode of the Motor#2**



LCU — Name | ConfigTimeSupport | General | Logic Control Unit (LCU) Configuration

Name | Lcu Logic Instance | Lcu Logic Input | Lcu Logic Output

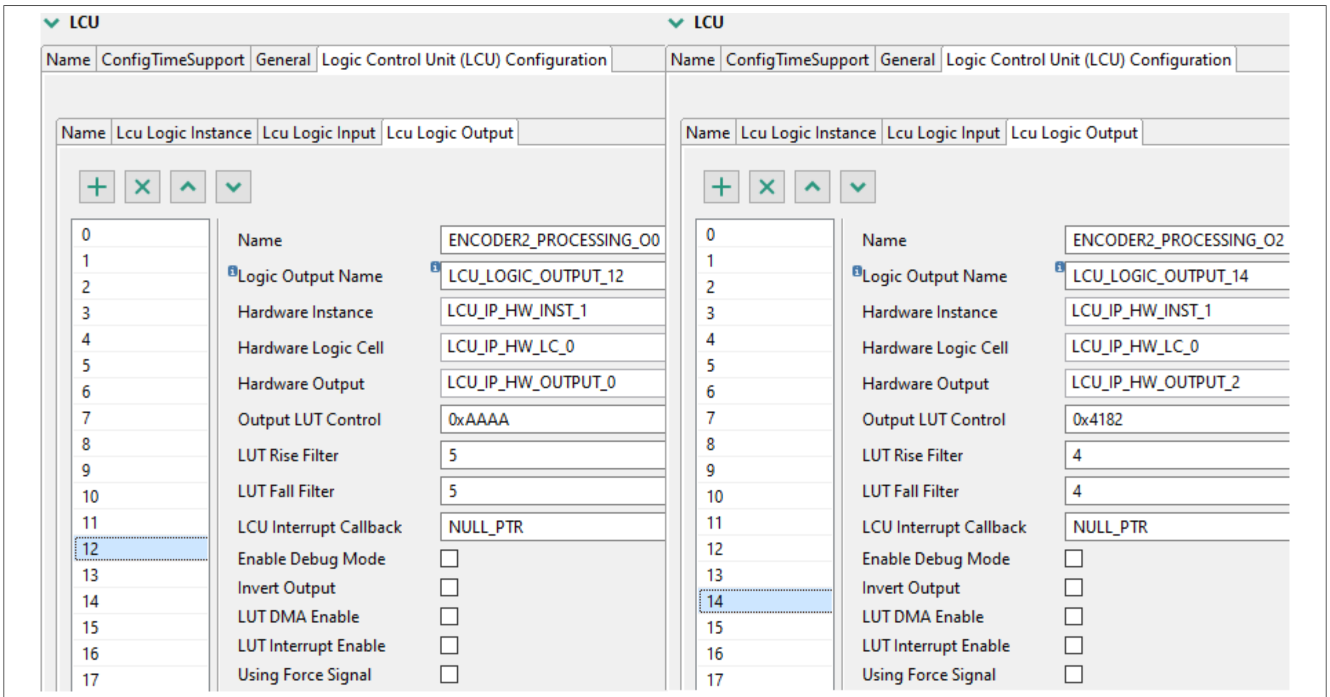
Left panel (selected item 16):

| | |
|---|---|
| Name | ENCODER1_PROCESSING_O0 |
| Logic Output Name | LCU_LOGIC_OUTPUT_16 |
| Hardware Instance | LCU_IP_HW_INST_1 |
| Hardware Logic Cell | LCU_IP_HW_LC_2 |
| Hardware Output | LCU_IP_HW_OUTPUT_0 |
| Output LUT Control | 0xAAAA |
| LUT Rise Filter | 5 |
| LUT Fall Filter | 5 |
| LCU Interrupt Callback | NULL_PTR |
| Enable Debug Mode | ☐ |
| Invert Output | ☐ |
| LUT DMA Enable | ☐ |
| LUT Interrupt Enable | ☐ |
| Using Force Signal | ☐ |

Right panel:

| | |
|---|---|
| Name | ENCODER1_PROCESSING_O2 |
| Logic Output Name | LCU_LOGIC_OUTPUT_18 |
| Hardware Instance | LCU_IP_HW_INST_1 |
| Hardware Logic Cell | LCU_IP_HW_LC_2 |
| Hardware Output | LCU_IP_HW_OUTPUT_2 |
| Output LUT Control | 0x4182 |
| LUT Rise Filter | 4 |
| LUT Fall Filter | 4 |
| LCU Interrupt Callback | NULL_PTR |
| Enable Debug Mode | ☐ |
| Invert Output | ☐ |
| LUT DMA Enable | ☐ |
| LUT Interrupt Enable | ☐ |
| Using Force Signal | ☐ |

**Figure 60. LCU outputs for quadrature decoder mode of the Motor#1**



LCU — Name | ConfigTimeSupport | General | Logic Control Unit (LCU) Configuration

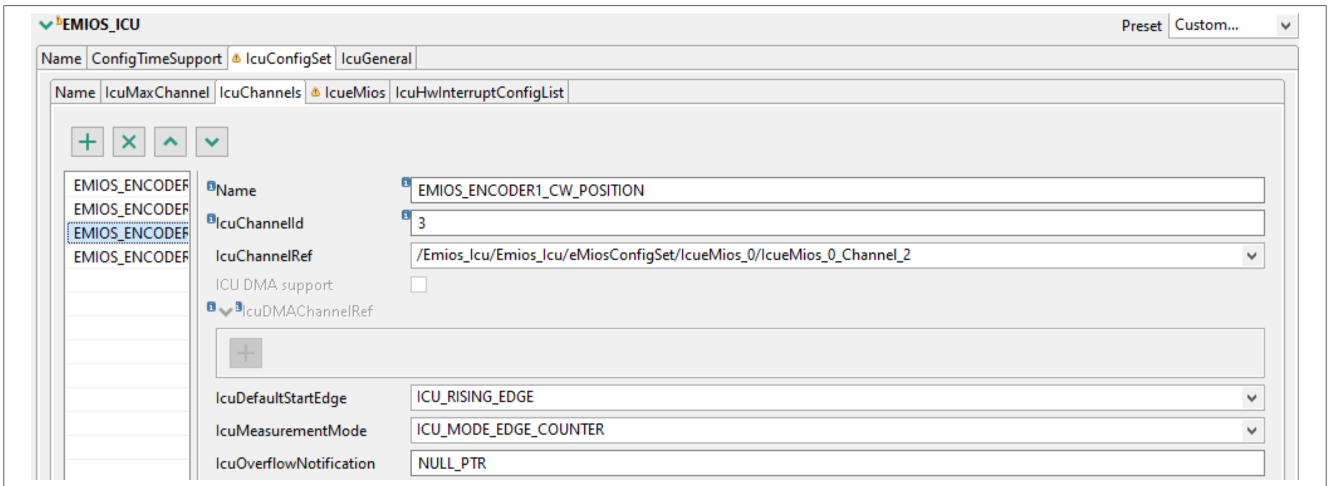
Name | Lcu Logic Instance | Lcu Logic Input | Lcu Logic Output

| | Left (selected 12) | Right (selected 14) |
|---|---|---|
| Name | ENCODER2_PROCESSING_O0 | ENCODER2_PROCESSING_O2 |
| Logic Output Name | LCU_LOGIC_OUTPUT_12 | LCU_LOGIC_OUTPUT_14 |
| Hardware Instance | LCU_IP_HW_INST_1 | LCU_IP_HW_INST_1 |
| Hardware Logic Cell | LCU_IP_HW_LC_0 | LCU_IP_HW_LC_0 |
| Hardware Output | LCU_IP_HW_OUTPUT_0 | LCU_IP_HW_OUTPUT_2 |
| Output LUT Control | 0xAAAA | 0x4182 |
| LUT Rise Filter | 5 | 4 |
| LUT Fall Filter | 5 | 4 |
| LCU Interrupt Callback | NULL_PTR | NULL_PTR |
| Enable Debug Mode | ☐ | ☐ |
| Invert Output | ☐ | ☐ |
| LUT DMA Enable | ☐ | ☐ |
| LUT Interrupt Enable | ☐ | ☐ |
| Using Force Signal | ☐ | ☐ |

Figure 61. LCU outputs for quadrature decoder of the Motor#2

### 4.2.6.3 eMIOS

eMIOS0 CH3, CH4, 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, eMIOS0 CH3 and CH4 are used for the Motor#1 and eMIOS0 CH5 and CH6 are used for the Motor#2. 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_A_ENC_A and HALL_B_ENC_B of encoder sensor of the Motor#1 and HALL_D_ENC_D and HALL_E_ENC_E of encoder sensor of the Motor#2. 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.



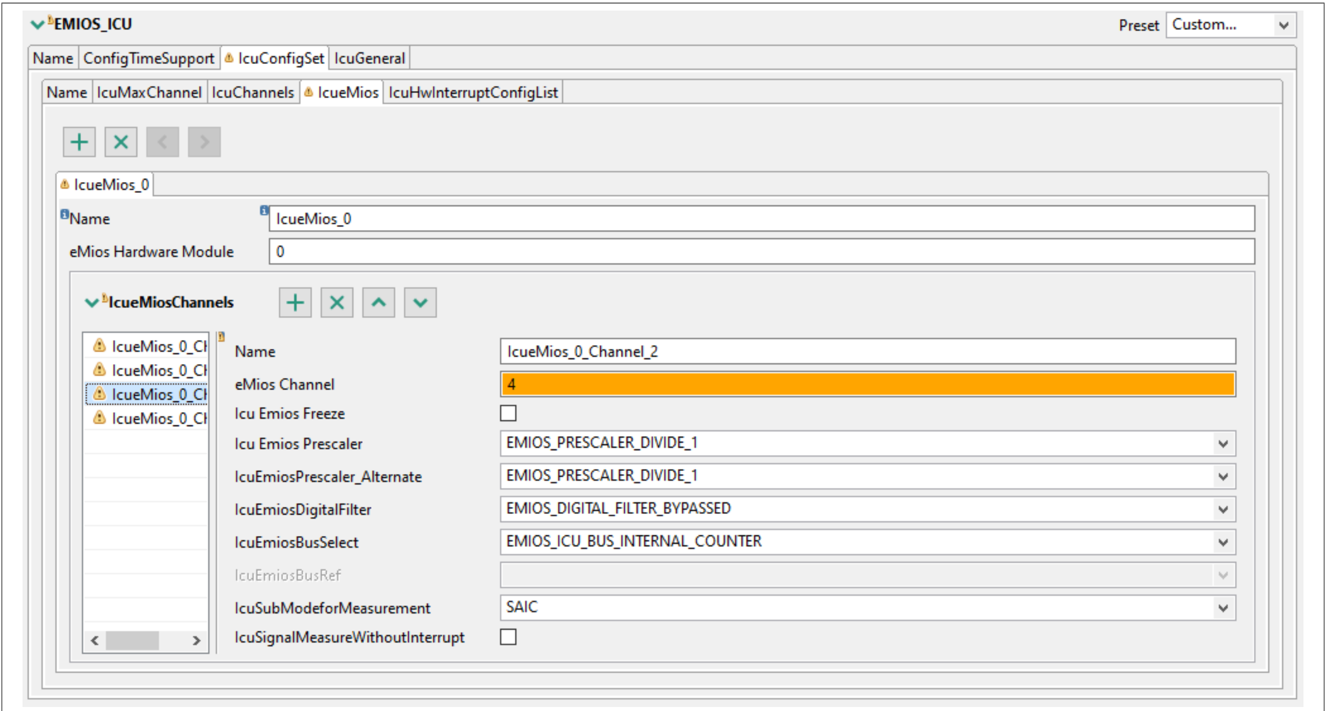
EMIOS_ICU — Preset: Custom...

Name | ConfigTimeSupport | IcuConfigSet | IcuGeneral

Name | IcuMaxChannel | IcuChannels | IcueMios | IcuHwInterruptConfigList

| | |
|---|---|
| Name | EMIOS_ENCODER1_CW_POSITION |
| IcuChannelId | 3 |
| IcuChannelRef | /Emios_Icu/Emios_Icu/eMiosConfigSet/IcueMios_0/IcueMios_0_Channel_2 |
| ICU DMA support | ☐ |
| IcuDMAChannelRef | |
| IcuDefaultStartEdge | ICU_RISING_EDGE |
| IcuMeasurementMode | ICU_MODE_EDGE_COUNTER |
| IcuOverflowNotification | NULL_PTR |

Figure 62. General ICU configuration effective for both motors



Name: IcueMios_0
eMios Hardware Module: 0

Name: IcueMios_0_Channel_2
eMios Channel: 4
Icu Emios Freeze: ☐
Icu Emios Prescaler: EMIOS_PRESCALER_DIVIDE_1
IcuEmiosPrescaler_Alternate: EMIOS_PRESCALER_DIVIDE_1
IcuEmiosDigitalFilter: EMIOS_DIGITAL_FILTER_BYPASSED
IcuEmiosBusSelect: EMIOS_ICU_BUS_INTERNAL_COUNTER
IcuEmiosBusRef:
IcuSubModeforMeasurement: SAIC
IcuSignalMeasureWithoutInterrupt: ☐

Figure 63. eMIOS channels for input capture effective for both motors

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

**Example 8.eMIOS API for quadrature decoder**

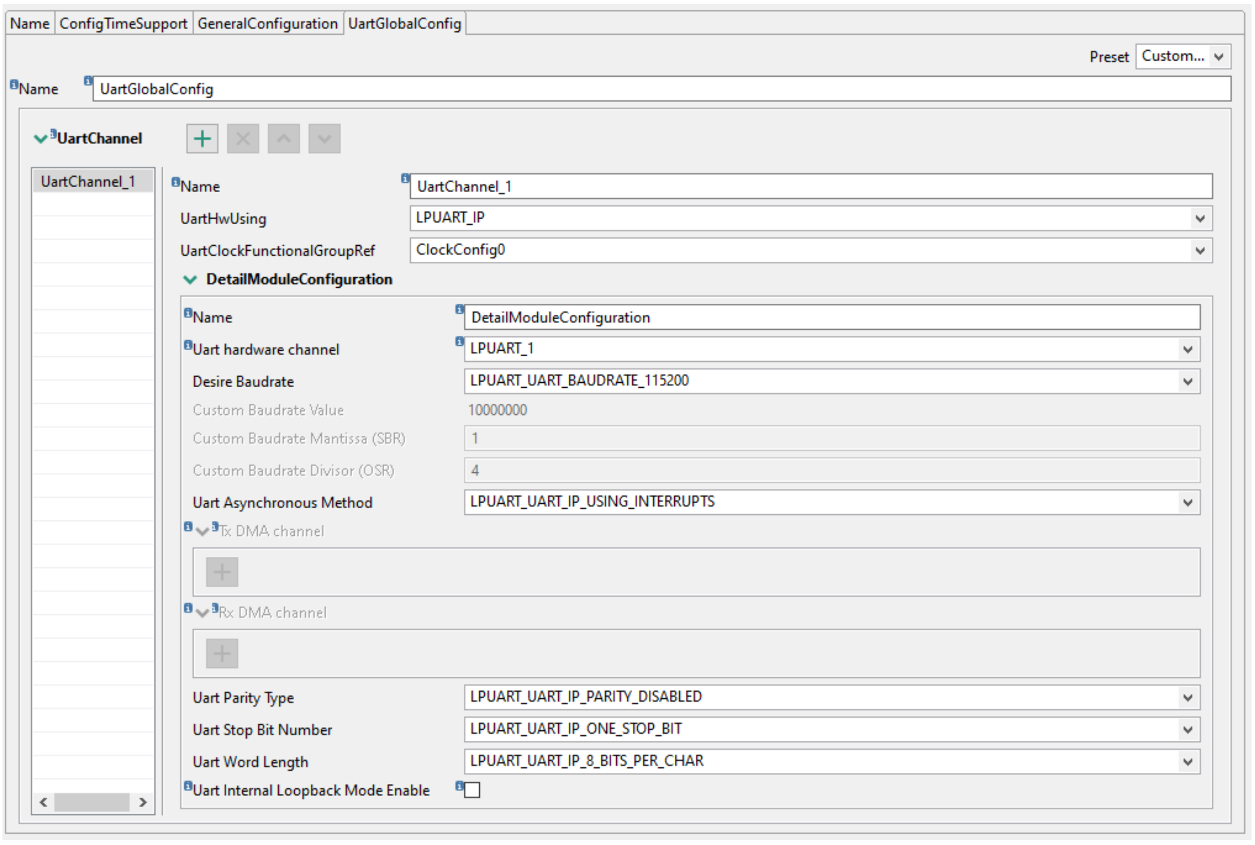
```
void main (void)
{
    /**********************************************
    * eMios Driver
    **********************************************/
#if ENCODER1 && Motor1
    Emios_Icu_Ip_Init(0U, &eMios_Icu_Ip_0_Config_PB);

    Emios_Icu_Ip_SetInitialCounterValue(0U, 3U, (uint32_t)0x1U);
    Emios_Icu_Ip_SetInitialCounterValue(0U, 4U, (uint32_t)0x1U);

    Emios_Icu_Ip_SetMaxCounterValue(0U, 3U, (4*M1_ENC_PULSES));
    Emios_Icu_Ip_SetMaxCounterValue(0U, 4U, (4*M1_ENC_PULSES));

    Emios_Icu_Ip_EnableEdgeCount(0u, 3U);
    Emios_Icu_Ip_EnableEdgeCount(0u, 4U);
#endif

#if ENCODER2 && Motor2
    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, (4*M2_ENC_PULSES));
    Emios_Icu_Ip_SetMaxCounterValue(0U, 6U, (4*M2_ENC_PULSES));

    Emios_Icu_Ip_EnableEdgeCount(0u, 5U);
    Emios_Icu_Ip_EnableEdgeCount(0u, 6U);
#endif
...
}
tBool POSPE_GetPospeElEnc (pmsmDrive_t *ptr)
{
...
    /* read encoder edges to get mechanical position */
    /* CW  counter */
    counterCW = (uint16_t)((Emios_Icu_Ip_GetEdgeNumbers(ptr-
>pospePeriphCfg.encCfg.CW_dir.eMIOS_Inst,
  ptr->pospePeriphCfg.encCfg.CW_dir.eMIOS_Chan)) - ptr-
>encoderPospe.counterCwOffset);
    /* CCW counter */
    counterCCW = (uint16_t)((Emios_Icu_Ip_GetEdgeNumbers(ptr-
>pospePeriphCfg.encCfg.CCW_dir.eMIOS_Inst,
  ptr->pospePeriphCfg.encCfg.CCW_dir.eMIOS_Chan)) - ptr-
>encoderPospe.counterCcwOffset);
...
}
```

***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].


Name | ConfigTimeSupport | GeneralConfiguration | UartGlobalConfig

Preset: Custom...

Name: UartGlobalConfig

UartChannel: UartChannel_1

| | |
|---|---|
| Name | UartChannel_1 |
| UartHwUsing | LPUART_IP |
| UartClockFunctionalGroupRef | ClockConfig0 |

DetailModuleConfiguration

| | |
|---|---|
| Name | DetailModuleConfiguration |
| Uart hardware channel | LPUART_1 |
| Desire Baudrate | LPUART_UART_BAUDRATE_115200 |
| Custom Baudrate Value | 10000000 |
| Custom Baudrate Mantissa (SBR) | 1 |
| Custom Baudrate Divisor (OSR) | 4 |
| Uart Asynchronous Method | LPUART_UART_IP_USING_INTERRUPTS |
| Tx DMA channel | |
| Rx DMA channel | |
| Uart Parity Type | LPUART_UART_IP_PARITY_DISABLED |
| Uart Stop Bit Number | LPUART_UART_IP_ONE_STOP_BIT |
| Uart Word Length | LPUART_UART_IP_8_BITS_PER_CHAR |
| Uart Internal Loopback Mode Enable | ☐ |

**Figure 64. LPUART configuration**

## 4.3 Software architecture

### 4.3.1 Introduction

This section describes the software design of the Sensorless PMSM Field Oriented Control framework application for dual 3-phase motor control. The application overview and description of software implementation are provided. The aim of this chapter is to help in understanding of the designed software.

### 4.3.2 Application software design

To make the software easily applicable for one or multi-motor application, all parameters and variables are grouped in a specific data structure. In this application, data structure *pmsmDrive_t* represents a predefined format that is used for organizing and storing data for the purpose of working on it with various algorithms, routines, and functions. The proposed software structure includes all variables related to cascade structure for PMSM FOC, state variables related to the application, and data for peripherals configuration such as base address or channel offset.

The declaration of structures used for dual motor control application is as follows:

**Example 9.Declaration of structures for dual motor control**

```
/* Application and FOC control */
#if Motor1
pmsmDrive_t drvFOC1;  /* Field Oriented Control Variables M1 */
#endif
```

```
#if Motor2
pmsmDrive_t drvFOC2;  /* Field Oriented Control Variables M2 */
#endif
________________________________________________________________
```

The names of all variables are composed in a prefix form given by the name of structure drvFOC1 or drvFOC2 and the name of the variables/structures which are members of the main structure *pmsmDrive_t*. This example also contains #if, #else directives, which control compilation of portion of code. In this case, those directives are used for compilation of Motor#1 or Motor#2 parameters with their position sensors (Encoder and Resolver) and functions. All functions in the software example including state machine utilize pointer arithmetic in order to avoid duplication of function declaration for particular motor and achieve higher performance.

### 4.3.3 Application data flow overview

The application software is an interrupt driven running in real time. There is one periodic interrupt service routines associated with the ADC conversion complete interrupt, executing all motor control tasks.

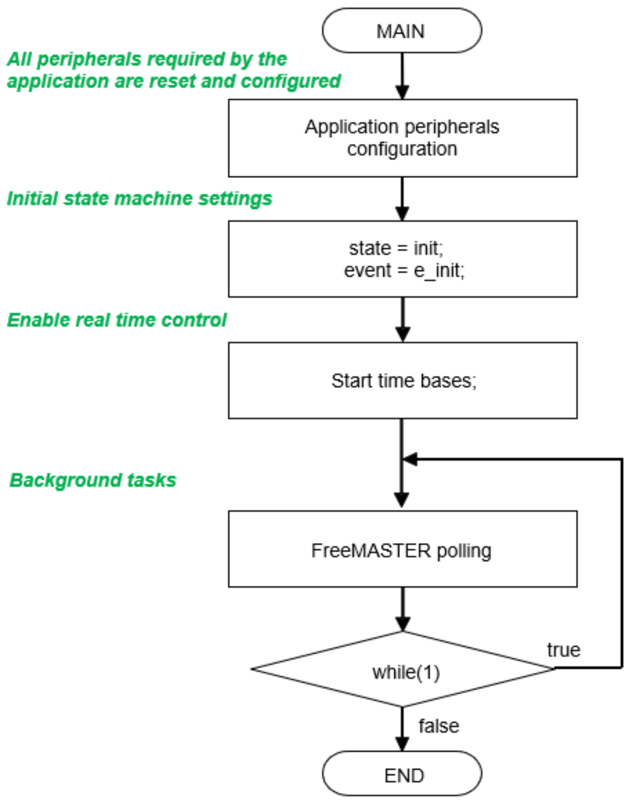

MAIN
All peripherals required by the application are reset and configured
Application peripherals configuration
Initial state machine settings
state = init;
event = e_init;
Enable real time control
Start time bases;
Background tasks
FreeMASTER polling
while(1)
true
false
END

**Figure 65. Flow chart diagram of main function with background loop.**

This interrupt routine is common for both motors to acquire feedback variables, however state machine is handled for each motor individually. This means that all motor-control related functions of Motor#1 are executed independently from Motor#2. All tasks are performed in an order described by the application state machine shown in Figure 67, and application flowcharts shown in Figure 65 and Figure 66.

To achieve precise and deterministic sampling of analog quantities and to execute all necessary motor control calculations, state machine functions of both motors are called within a periodic notification function. Hence, in order to actually call particular 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 particular state machine functions are called from the BCTU notification function. The background loop handles non-critical timing tasks, such as the FreeMASTER communication polling.

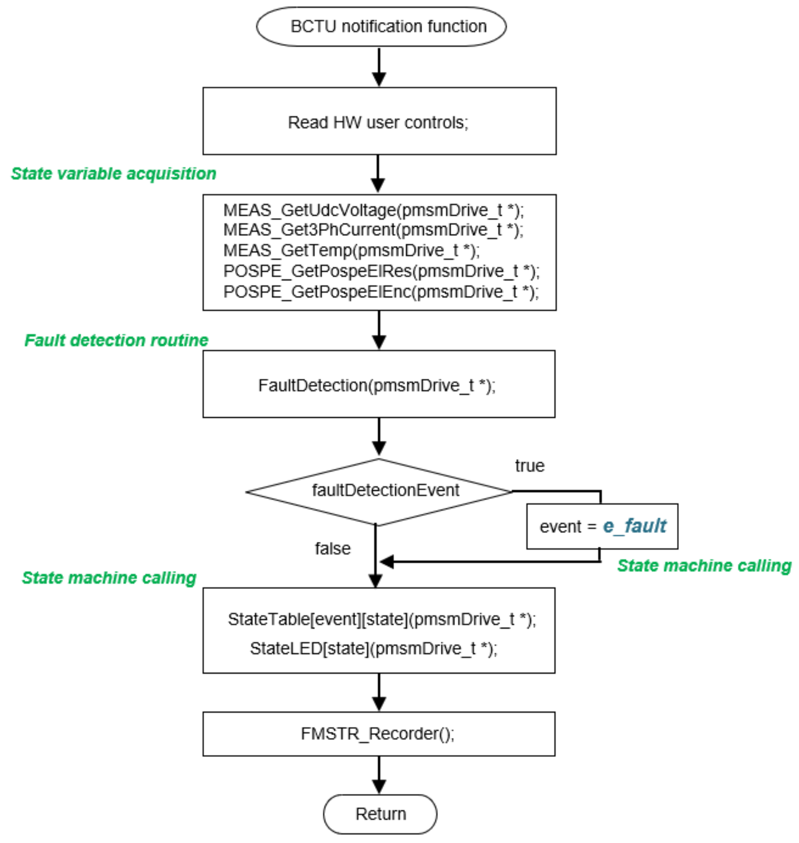

BCTU notification function
Read HW user controls;
State variable acquisition
MEAS_GetUdcVoltage(pmsmDrive_t *);
MEAS_Get3PhCurrent(pmsmDrive_t *);
MEAS_GetTemp(pmsmDrive_t *);
POSPE_GetPospeElRes(pmsmDrive_t *);
POSPE_GetPospeElEnc(pmsmDrive_t *);
Fault detection routine
FaultDetection(pmsmDrive_t *);
faultDetectionEvent
true
event = e_fault
false
State machine calling
State machine calling
StateTable[event][state](pmsmDrive_t *);
StateLED[state](pmsmDrive_t *);
FMSTR_Recorder();
Return

**Figure 66. Flow chart diagram of periodic interrupts notification functions.**

## 4.3.4 State machine

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

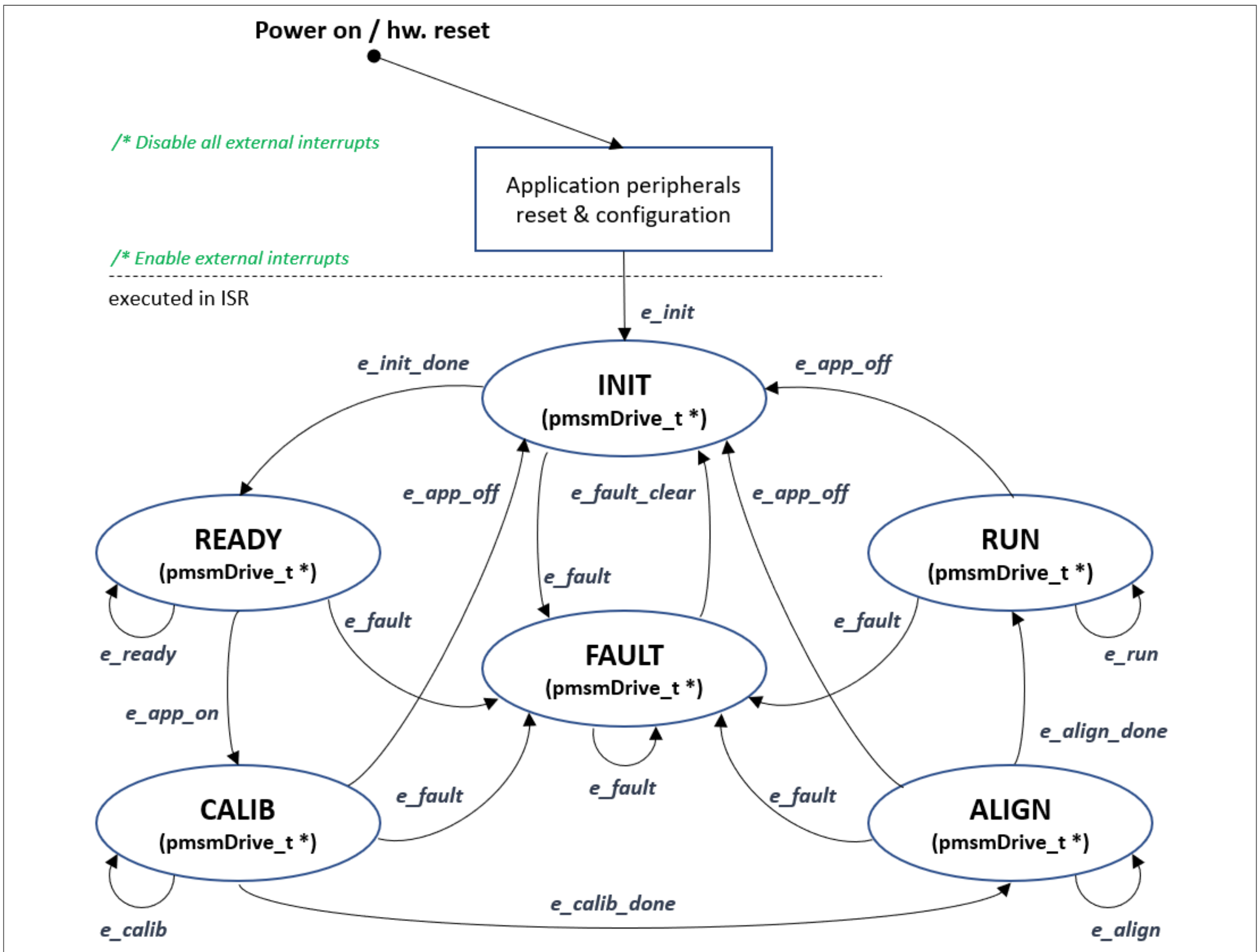


Figure 67. Application state machine

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

### 4.3.4.1 State – FAULT

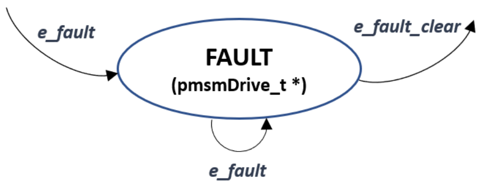


**Figure 68. FAULT state with transitions**

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 (SW2 and SW3 on adapter board) in case of the Motor#1 and (SW4 and SW5 on adapter board) in case of the Motor#2 on the adapter 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, Example 10.

**Example 10.Fault clearing sequence**

```
void StateFault(pmsmDrive_t *ptr)
{
    if (ptr->cntrState.usrControl.switchFaultClear)
   {
        // Clear permanent and temporary SW faults
        ptr->permFaults.mcu.R           = 0;       // Clear mcu faults
        ptr->permFaults.motor.R  = 0;     // Clear motor faults
        ptr->permFaults.stateMachine.R  = 0;     // Clear state machine faults
        // When all Faults cleared prepare for transition to next state.
        ptr->cntrState.usrControl.readFault             = true;
        ptr->cntrState.usrControl.switchFaultClear      = false;
        ptr->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 66, function *faultDetection(pmsmDrive_t *ptr)* is called before state machine function *state_table[event][state](pmsmDrive_t *ptr)*. 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(pmsmDrive_t *ptr)* 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](pmsmDrive_t *ptr)* 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/ D/ E/ F-over current

The thresholds for fault detection can be modified in MCAT tool 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)

### 4.3.4.2 State – INIT

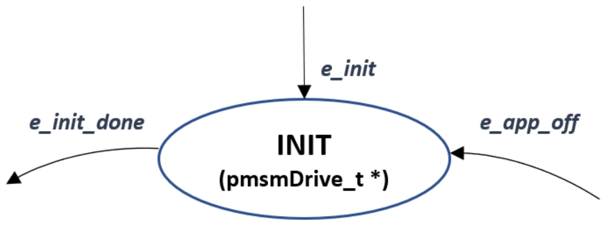


**Figure 69. INIT state with transitions**

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.

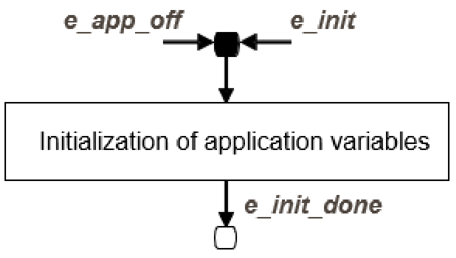


**Figure 70. Flow chart of state INIT**

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.

### 4.3.4.3 State – READY

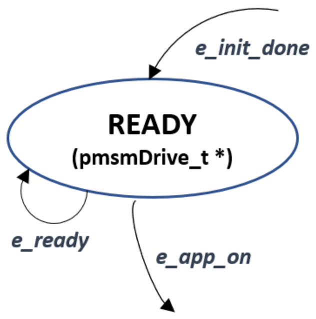


**Figure 71. READY state with transitions**

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 SW3 or SW2 or on falling edge of the SW1 in case of Motor#1 or SW4 or SW5 or on falling edge of the SW6 or on falling edge of the SW6 in case of Motor#2 or by FreeMASTER interface setting the variable *switchAppOnOff* = *true* (see flow chart in Figure 72).

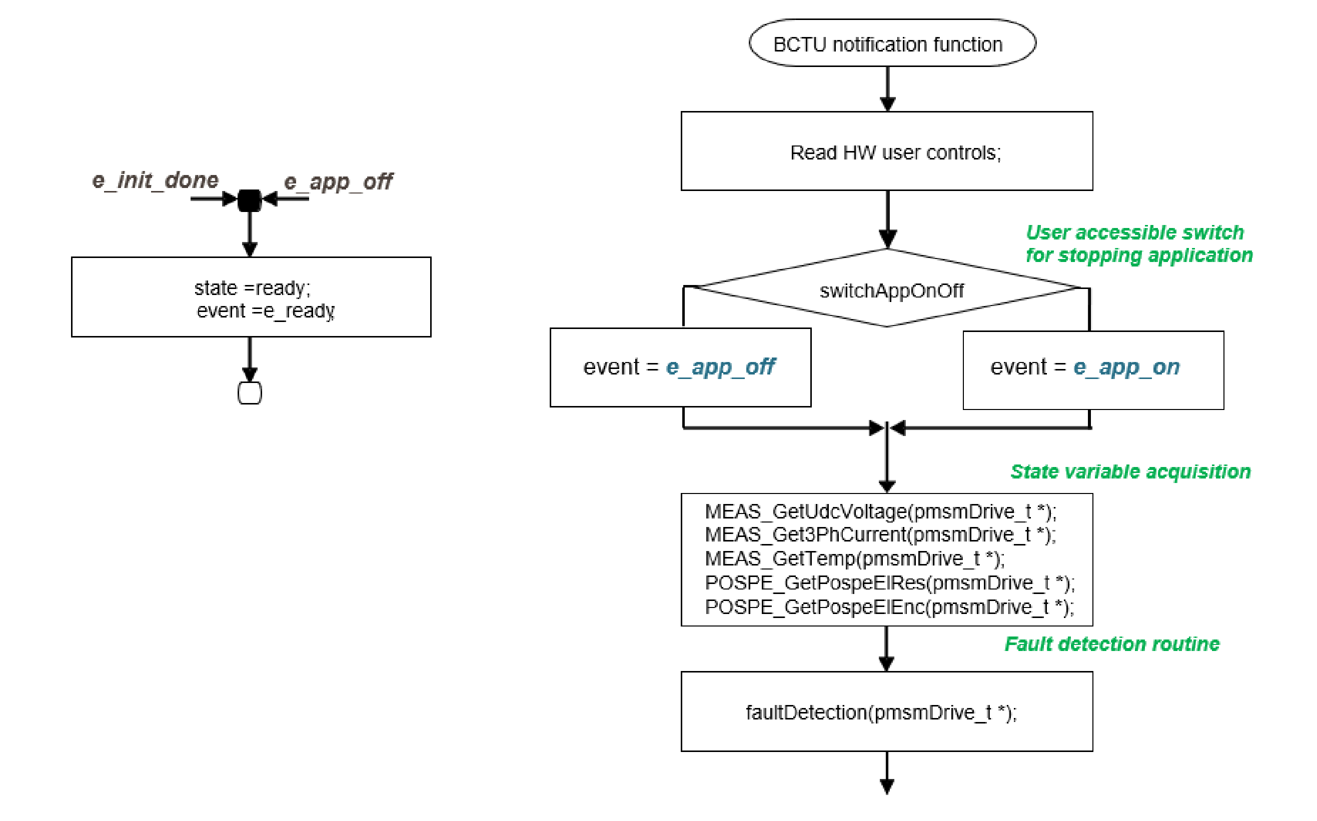

e_init_done
e_app_off
state =ready;
event =e_ready
BCTU notification function
Read HW user controls;
User accessible switch for stopping application
switchAppOnOff
event = e_app_off
event = e_app_on
State variable acquisition
MEAS_GetUdcVoltage(pmsmDrive_t *);
MEAS_Get3PhCurrent(pmsmDrive_t *);
MEAS_GetTemp(pmsmDrive_t *);
POSPE_GetPospeElRes(pmsmDrive_t *);
POSPE_GetPospeElEnc(pmsmDrive_t *);
Fault detection routine
faultDetection(pmsmDrive_t *);

**Figure 72. Flow chart of state READY**

### 4.3.4.4 State – CALIB

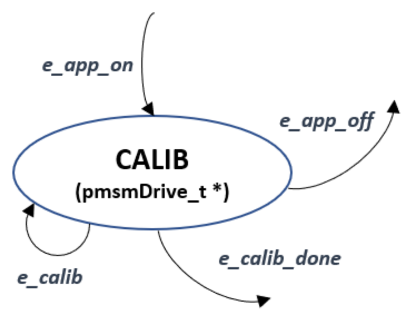


**Figure 73. CALIB state with transitions**

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 74). 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 74).

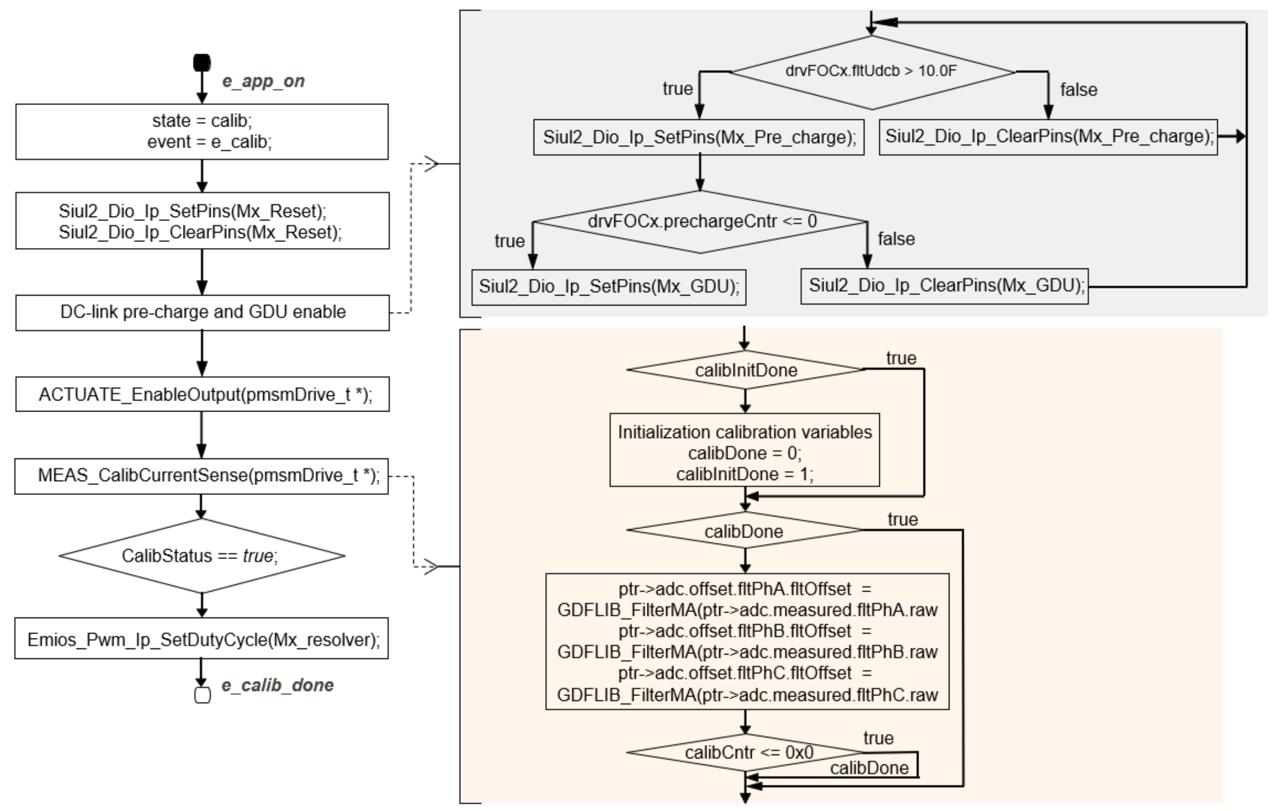

e_app_on
state = calib;
event = e_calib;
Siul2_Dio_Ip_SetPins(Mx_Reset);
Siul2_Dio_Ip_ClearPins(Mx_Reset);
DC-link pre-charge and GDU enable
ACTUATE_EnableOutput(pmsmDrive_t *);
MEAS_CalibCurrentSense(pmsmDrive_t *);
CalibStatus == true;
Emios_Pwm_Ip_SetDutyCycle(Mx_resolver);
e_calib_done
drvFOCx.fltUdcb > 10.0F
true
false
Siul2_Dio_Ip_SetPins(Mx_Pre_charge);
Siul2_Dio_Ip_ClearPins(Mx_Pre_charge);
drvFOCx.prechargeCntr <= 0
true
false
Siul2_Dio_Ip_SetPins(Mx_GDU);
Siul2_Dio_Ip_ClearPins(Mx_GDU);
calibInitDone
true
Initialization calibration variables
calibDone = 0;
calibInitDone = 1;
calibDone
true
ptr->adc.offset.fltPhA.fltOffset =
GDFLIB_FilterMA(ptr->adc.measured.fltPhA.raw
ptr->adc.offset.fltPhB.fltOffset =
GDFLIB_FilterMA(ptr->adc.measured.fltPhB.raw
ptr->adc.offset.fltPhC.fltOffset =
GDFLIB_FilterMA(ptr->adc.measured.fltPhC.raw
calibCntr <= 0x0
true
calibDone

**Figure 74. 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.

### 4.3.4.5 State – ALIGN

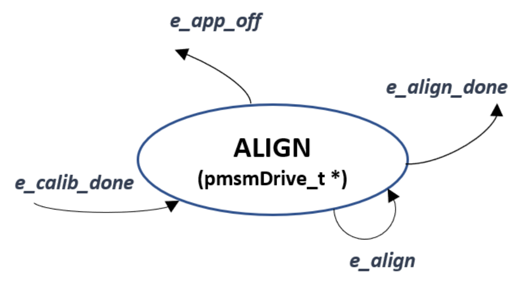


Figure 75. ALIGN state with transitions

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.

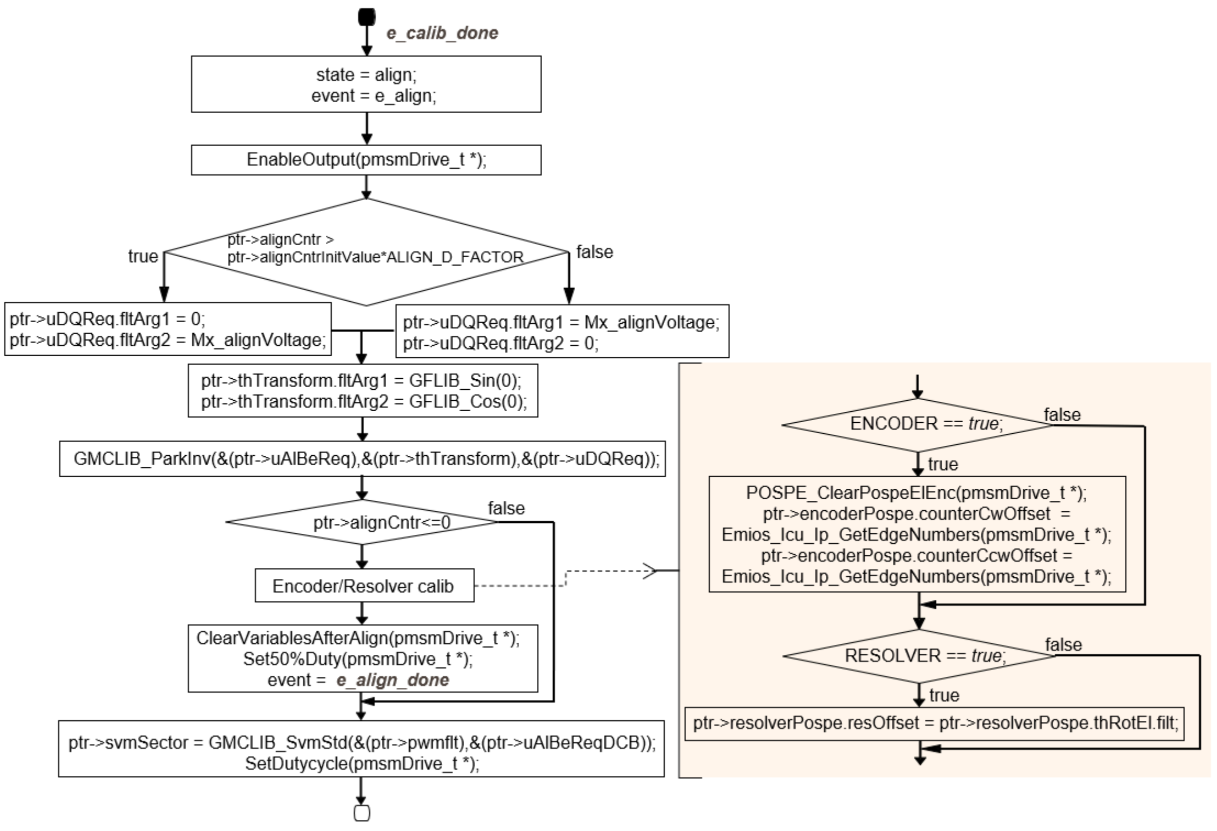

e_calib_done
state = align;
event = e_align;
EnableOutput(pmsmDrive_t *);
ptr->alignCntr >
ptr->alignCntrInitValue*ALIGN_D_FACTOR
true
false
ptr->uDQReq.fltArg1 = 0;
ptr->uDQReq.fltArg2 = Mx_alignVoltage;
ptr->uDQReq.fltArg1 = Mx_alignVoltage;
ptr->uDQReq.fltArg2 = 0;
ptr->thTransform.fltArg1 = GFLIB_Sin(0);
ptr->thTransform.fltArg2 = GFLIB_Cos(0);
GMCLIB_ParkInv(&(ptr->uAlBeReq),&(ptr->thTransform),&(ptr->uDQReq));
ptr->alignCntr<=0
false
Encoder/Resolver calib
ClearVariablesAfterAlign(pmsmDrive_t *);
Set50%Duty(pmsmDrive_t *);
event = e_align_done
ptr->svmSector = GMCLIB_SvmStd(&(ptr->pwmflt),&(ptr->uAlBeReqDCB));
SetDutycycle(pmsmDrive_t *);
ENCODER == true;
false
true
POSPE_ClearPospeElEnc(pmsmDrive_t *);
ptr->encoderPospe.counterCwOffset =
Emios_Icu_Ip_GetEdgeNumbers(pmsmDrive_t *);
ptr->encoderPospe.counterCcwOffset =
Emios_Icu_Ip_GetEdgeNumbers(pmsmDrive_t *);
RESOLVER == true;
false
true
ptr->resolverPospe.resOffset = ptr->resolverPospe.thRotEl.filt;

Figure 76. Flow chart of state ALIGN

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.

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. 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 76. 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 SW1 in case of the Motor#1 or on rising edge of the SW6 in case of the Motor#2 on adapter board (blue board) of NXP 48V development platform.

#### 4.3.4.6 State – RUN

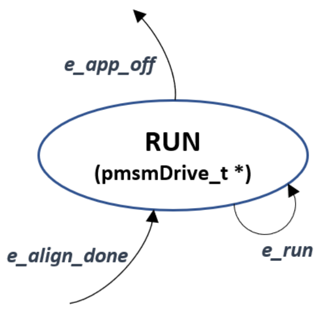


**Figure 77. RUN state with transitions**

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 five 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.*

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. If *switchSensor* = *Encoder,* the application uses Encoder mode only. If *switchSensor* = *Resolver,* the application uses Resolver mode only. The *switchSensor* is automatically set to *Sensorless*, if any of the sensors are not present (ENCODER=*false,* RESOLVER=*false*).

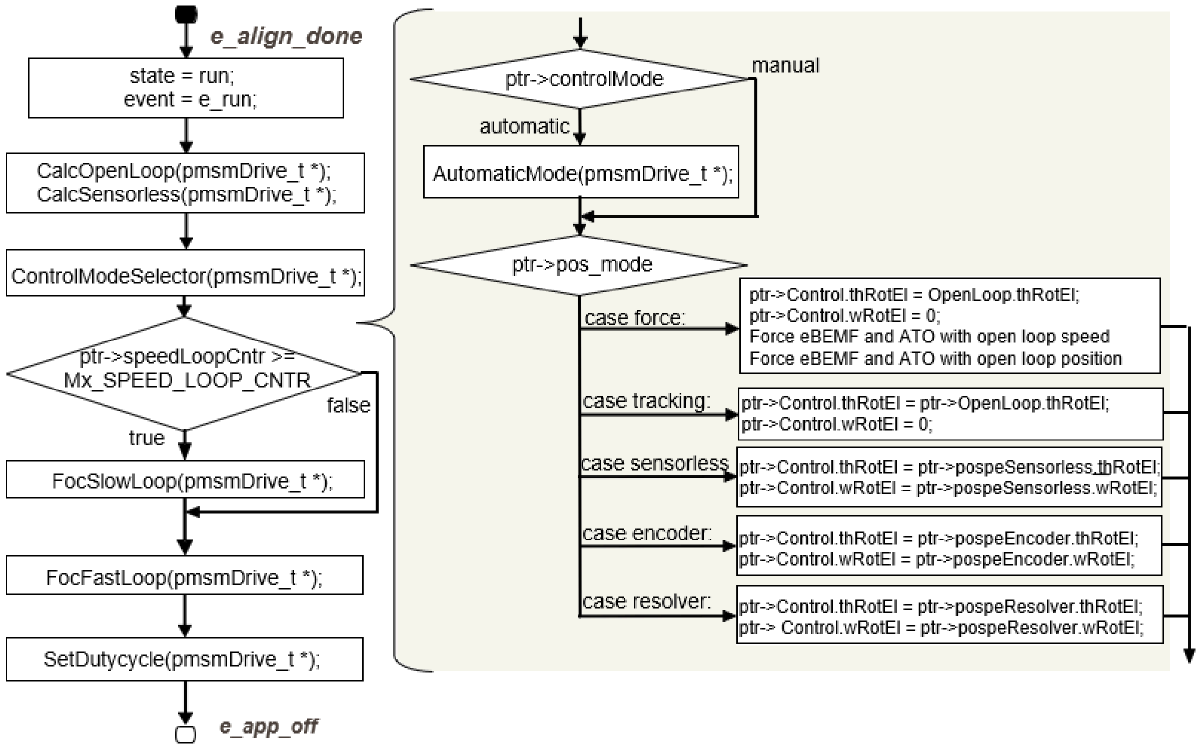

e_align_done
state = run;
event = e_run;
CalcOpenLoop(pmsmDrive_t *);
CalcSensorless(pmsmDrive_t *);
ControlModeSelector(pmsmDrive_t *);
ptr->speedLoopCntr >= Mx_SPEED_LOOP_CNTR
false
true
FocSlowLoop(pmsmDrive_t *);
FocFastLoop(pmsmDrive_t *);
SetDutycycle(pmsmDrive_t *);
e_app_off
ptr->controlMode
manual
automatic
AutomaticMode(pmsmDrive_t *);
ptr->pos_mode
case force:
ptr->Control.thRotEl = OpenLoop.thRotEl;
ptr->Control.wRotEl = 0;
Force eBEMF and ATO with open loop speed
Force eBEMF and ATO with open loop position
case tracking:
ptr->Control.thRotEl = ptr->OpenLoop.thRotEl;
ptr->Control.wRotEl = 0;
case sensorless
ptr->Control.thRotEl = ptr->pospeSensorless.thRotEl;
ptr->Control.wRotEl = ptr->pospeSensorless.wRotEl;
case encoder:
ptr->Control.thRotEl = ptr->pospeEncoder.thRotEl;
ptr->Control.wRotEl = ptr->pospeEncoder.wRotEl;
case resolver:
ptr->Control.thRotEl = ptr->pospeResolver.thRotEl;
ptr-> Control.wRotEl = ptr->pospeResolver.wRotEl;

**Figure 78. 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 78).

Figure 79 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*. Application allows to use only one sensor at the same time.

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 SW1 in case of the Motor#1 or on rising edge of the SW6 in case of the Motor#2 on adapter board.

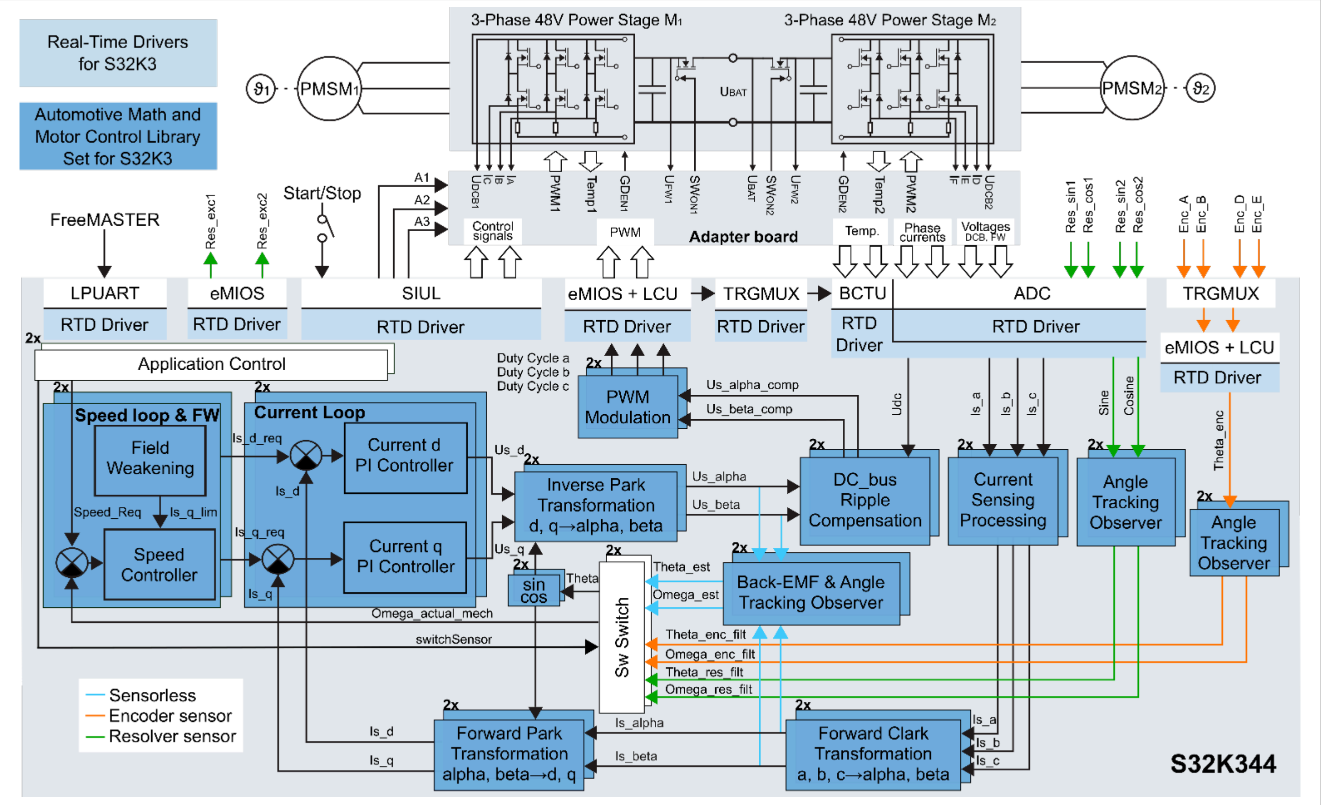


Figure 79. Sensorless and Sensor-based FOC with FW implementation on S32K344

### 4.3.5 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 79. 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 79. 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 80.

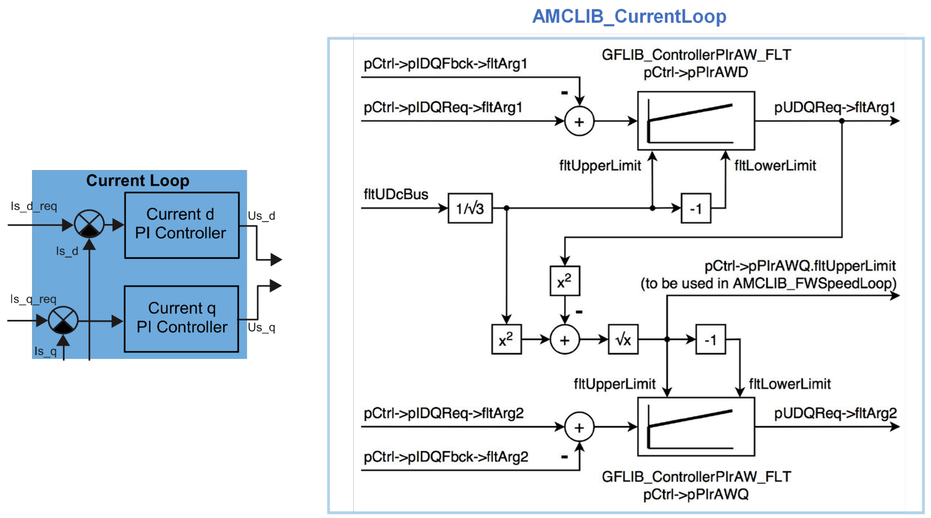


Figure 80. Functions and data structures in AMCLIB_CurrentLoop

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 79. 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 81. 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 81.

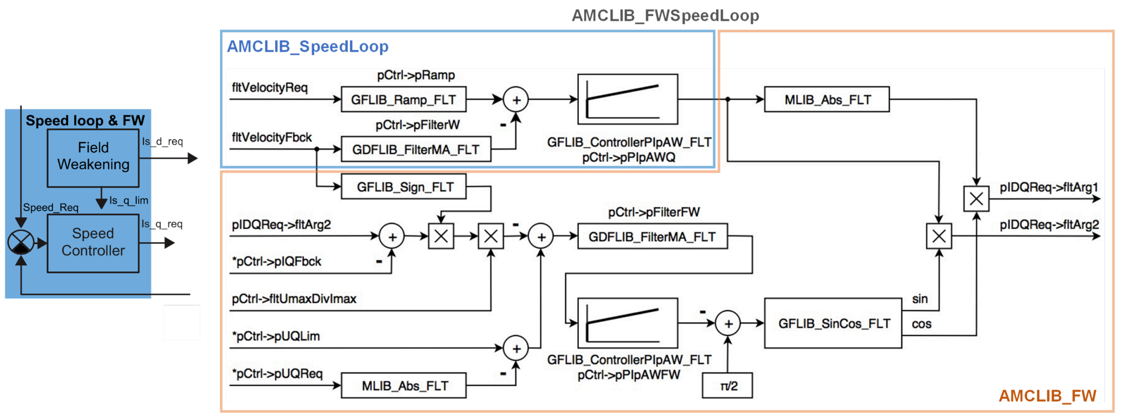


Figure 81. Functions and data structures in AMCLIB_FWSpeedLoop

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 79. They estimate rotor position and speed based on the inputs, namely stator voltages $u_{\alpha\beta}$ and currents $i_{\alpha\beta}$, Figure 82. 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 $\theta_{err}$. 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 $\theta_{err}$, which is provided as an output of the BEMF observer.

To align both frames and provide accurate estimates, this phase error $\theta_{err}$ 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 82. 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.

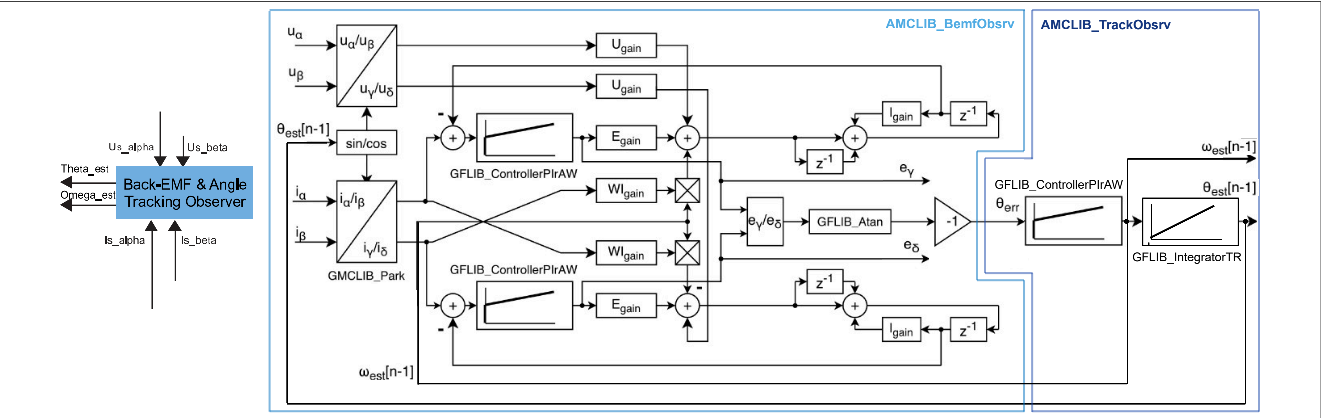


Figure 82. Structure of the AMCLIB_BemfObsrv and AMCLIB_TrackObsrv

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].

## 4.3.6 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.

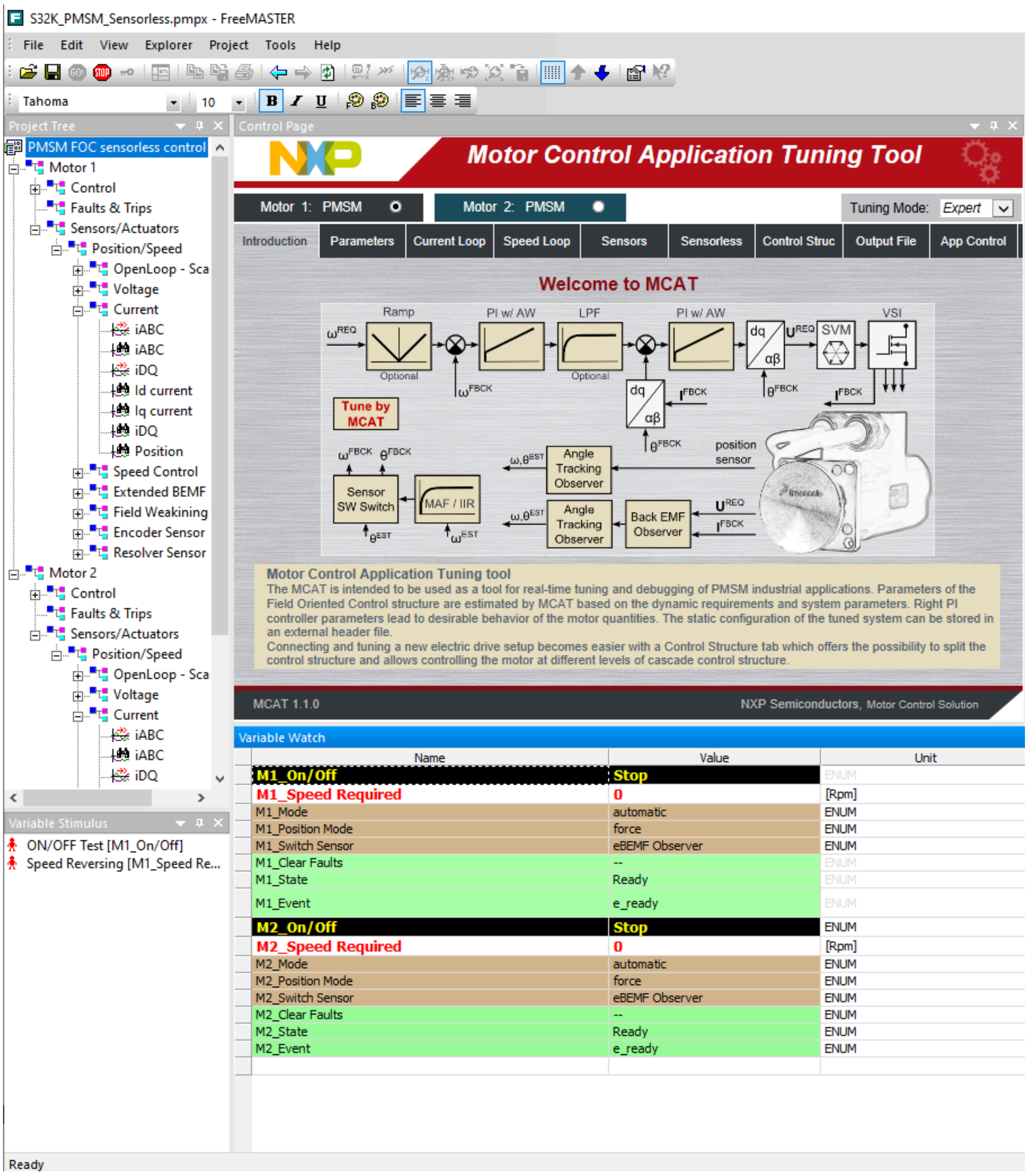
The MCAT tool runs under FreeMASTER online monitor, which allows the real-time tuning of the motor control application. Respecting the parameters of the controlled drive, the correct values of control structure parameters are calculated, which can be directly updated to the application or stored in an 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.



**Motor Control Application Tuning tool**

The MCAT is intended to be used as a tool for real-time tuning and debugging of PMSM industrial applications. Parameters of the Field Oriented Control structure are estimated by MCAT based on the dynamic requirements and system parameters. Right PI controller parameters lead to desirable behavior of the motor quantities. The static configuration of the tuned system can be stored in an external header file.

Connecting and tuning a new electric drive setup becomes easier with a Control Structure tab which offers the possibility to split the control structure and allows controlling the motor at different levels of cascade control structure.

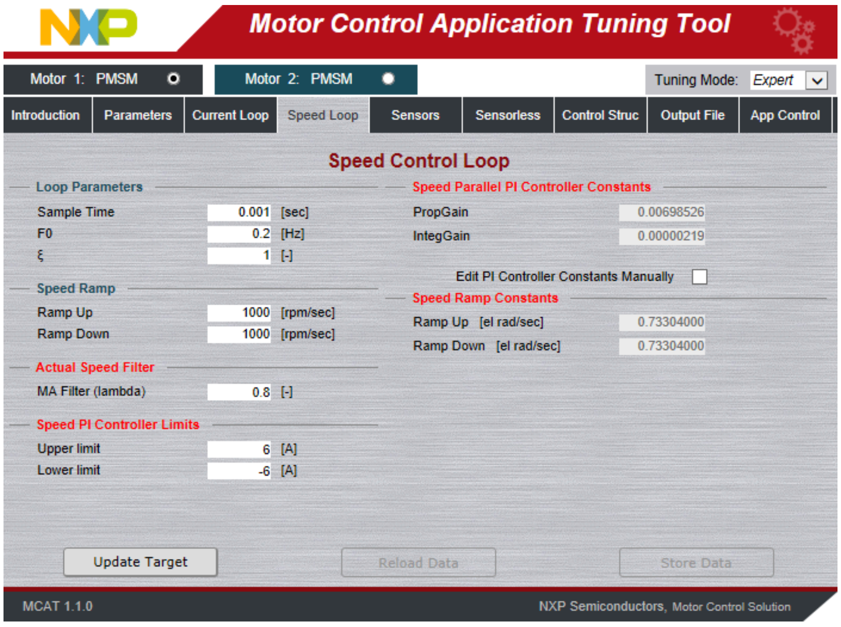
| Name | Value | Unit |
|---|---|---|
| M1_On/Off | Stop | ENUM |
| M1_Speed Required | 0 | [Rpm] |
| M1_Mode | automatic | ENUM |
| M1_Position Mode | force | ENUM |
| M1_Switch Sensor | eBEMF Observer | ENUM |
| M1_Clear Faults | -- | ENUM |
| M1_State | Ready | ENUM |
| M1_Event | e_ready | ENUM |
| M2_On/Off | Stop | ENUM |
| M2_Speed Required | 0 | [Rpm] |
| M2_Mode | automatic | ENUM |
| M2_Position Mode | force | ENUM |
| M2_Switch Sensor | eBEMF Observer | ENUM |
| M2_Clear Faults | -- | ENUM |
| M2_State | Ready | ENUM |
| M2_Event | e_ready | ENUM |

**Figure 83. 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 83) enables online application tuning and control. This tunning and control process is individual for each motor, which allows to do independent dual motor control. There are two tabs (Figure 83) separating Motor#1 and Motor#2 tuning and control processes. The MCAT tuning shall be used before the very first run of the drive to generate the configuration header file (M1_PMSM_appconfig.h for Motor#1 and M2_PMSM_appconfig.h for Motor#2). 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 84). 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 85).



Speed Control Loop

Loop Parameters

| | | |
|---|---|---|
| Sample Time | 0.001 | [sec] |
| F0 | 0.2 | [Hz] |
| ξ | 1 | [-] |

Speed Ramp

| | | |
|---|---|---|
| Ramp Up | 1000 | [rpm/sec] |
| Ramp Down | 1000 | [rpm/sec] |

Actual Speed Filter

| | | |
|---|---|---|
| MA Filter (lambda) | 0.8 | [-] |

Speed PI Controller Limits

| | | |
|---|---|---|
| Upper limit | 6 | [A] |
| Lower limit | -6 | [A] |

Speed Parallel PI Controller Constants

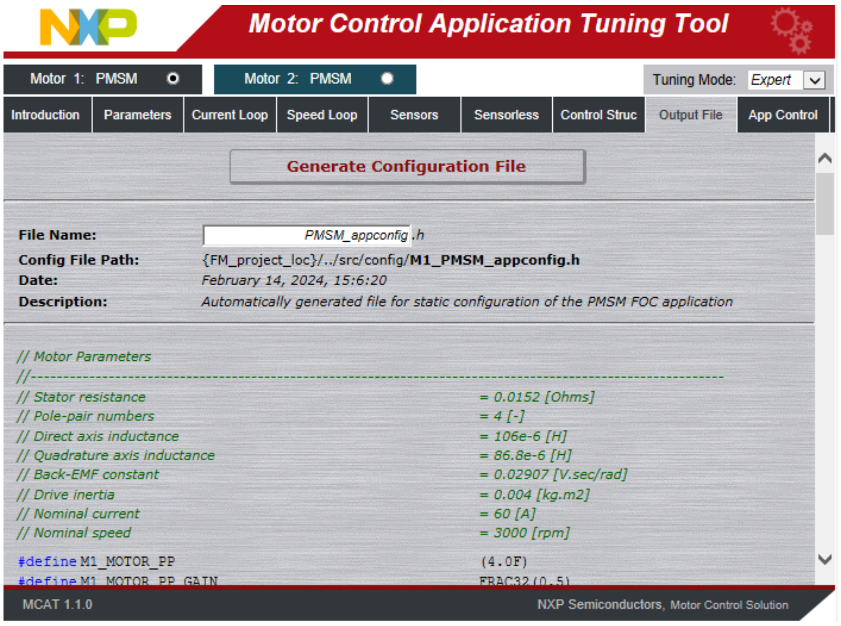
| | |
|---|---|
| PropGain | 0.00698526 |
| IntegGain | 0.00000219 |

Edit PI Controller Constants Manually ☐

Speed Ramp Constants

| | |
|---|---|
| Ramp Up [el rad/sec] | 0.73304000 |
| Ramp Down [el rad/sec] | 0.73304000 |

Figure 84. Parameters for online tunning



| | |
|---|---|
| **File Name:** | PMSM_appconfig .h |
| **Config File Path:** | {FM_project_loc}/../src/config/**M1_PMSM_appconfig.h** |
| **Date:** | *February 14, 2024, 15:6:20* |
| **Description:** | *Automatically generated file for static configuration of the PMSM FOC application* |

```
// Motor Parameters
//----------------------------------------------------------------------------------------------------
// Stator resistance                                  = 0.0152 [Ohms]
// Pole-pair numbers                                  = 4 [-]
// Direct axis inductance                             = 106e-6 [H]
// Quadrature axis inductance                         = 86.8e-6 [H]
// Back-EMF constant                                  = 0.02907 [V.sec/rad]
// Drive inertia                                      = 0.004 [kg.m2]
// Nominal current                                    = 60 [A]
// Nominal speed                                      = 3000 [rpm]

#define M1_MOTOR_PP                                   (4.0F)
#define M1_MOTOR_PP_GAIN                              FRAC32(0.5)
```

Figure 85. Output File panel and “Generate Configuration File” button

Parameters runtime update is done using the “Update Target” button (see Figure 86). 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 Mx_PMSM_appconfig.h header file. File holding the configuration is “{Project Location}\FreeMASTER_control\ MCAT\param_files\M1_params.txt” for Motor#1 and “{Project

Location}\FreeMASTER_control\ MCAT\param_files\M2_params.txt" for Motor#2. 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 83), 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.*

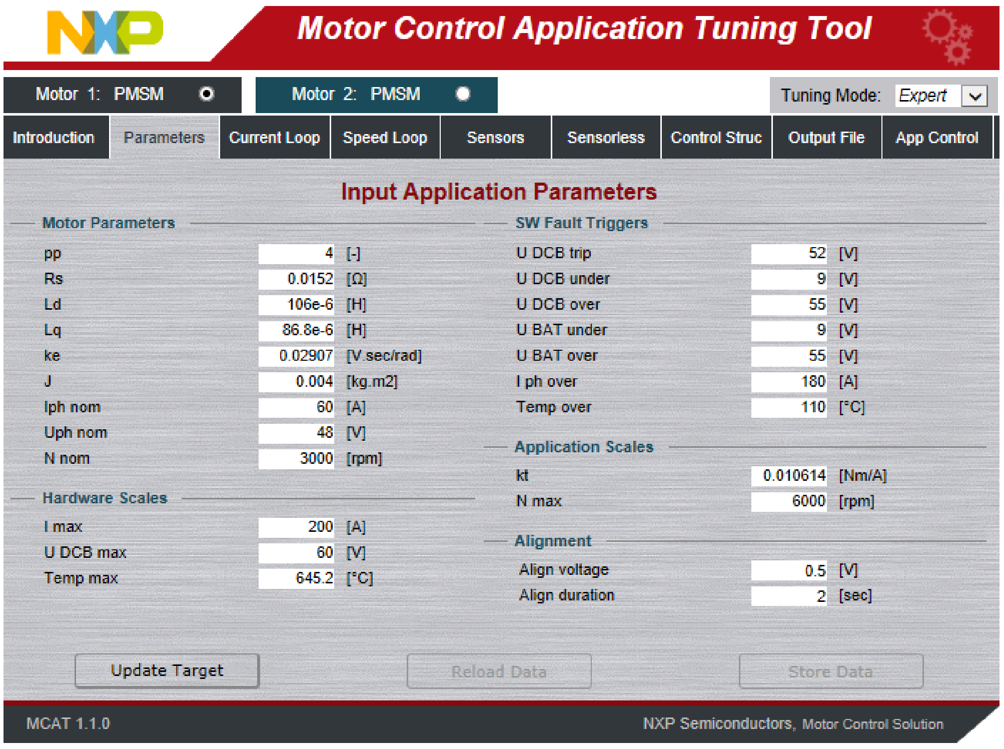
The application tuning is provided by a set of MCAT pages dedicated to every part of the control structure. An example of the Application Parameters Tuning page is in Figure 86. Following list of settings pages is based on the PMSM sensorless application.

**Table 6. Motor Control Parameter Table**

| Category | Parameters |
|---|---|
| Parameters | Motor Parameters, Hardware Scales, SW Fault Triggers, Application Scales, Alignment |
| Current Loop | Loop Parameters, D axis PI Controller, Q axis PI Controller, Current PI Controller Limits, DC-bus voltage MA filter settings |
| Speed Loop | Loop Parameters, Speed PI Controller Constants, Speed Ramp, Speed Ramp Constants, Actual Speed Filter, Speed 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 87), 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)



Input Application Parameters

Motor Parameters

| | | |
|---|---|---|
| pp | 4 | [-] |
| Rs | 0.0152 | [Ω] |
| Ld | 106e-6 | [H] |
| Lq | 86.8e-6 | [H] |
| ke | 0.02907 | [V.sec/rad] |
| J | 0.004 | [kg.m2] |
| Iph nom | 60 | [A] |
| Uph nom | 48 | [V] |
| N nom | 3000 | [rpm] |

Hardware Scales

| | | |
|---|---|---|
| I max | 200 | [A] |
| U DCB max | 60 | [V] |
| Temp max | 645.2 | [°C] |

SW Fault Triggers

| | | |
|---|---|---|
| U DCB trip | 52 | [V] |
| U DCB under | 9 | [V] |
| U DCB over | 55 | [V] |
| U BAT under | 9 | [V] |
| U BAT over | 55 | [V] |
| I ph over | 180 | [A] |
| Temp over | 110 | [°C] |

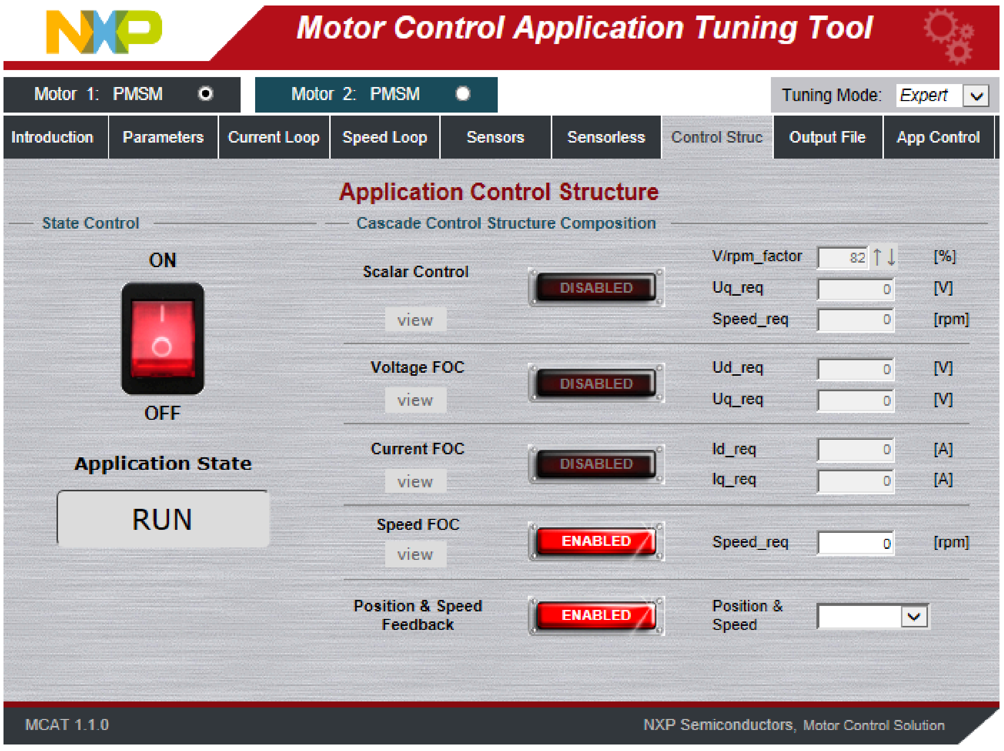
Application Scales

| | | |
|---|---|---|
| kt | 0.010614 | [Nm/A] |
| N max | 6000 | [rpm] |

Alignment

| | | |
|---|---|---|
| Align voltage | 0.5 | [V] |
| Align duration | 2 | [sec] |

Figure 86. MCAT input application parameters page



Application Control Structure

State Control

Application State: RUN

Cascade Control Structure Composition

| | | | | |
|---|---|---|---|---|
| Scalar Control | DISABLED | V/rpm_factor | 82 | [%] |
| | | Uq_req | 0 | [V] |
| | | Speed_req | 0 | [rpm] |
| Voltage FOC | DISABLED | Ud_req | 0 | [V] |
| | | Uq_req | 0 | [V] |
| Current FOC | DISABLED | Id_req | 0 | [A] |
| | | Iq_req | 0 | [A] |
| Speed FOC | ENABLED | Speed_req | 0 | [rpm] |
| Position & Speed Feedback | ENABLED | Position & Speed | | |

Figure 87. 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 88. 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 88, 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”.

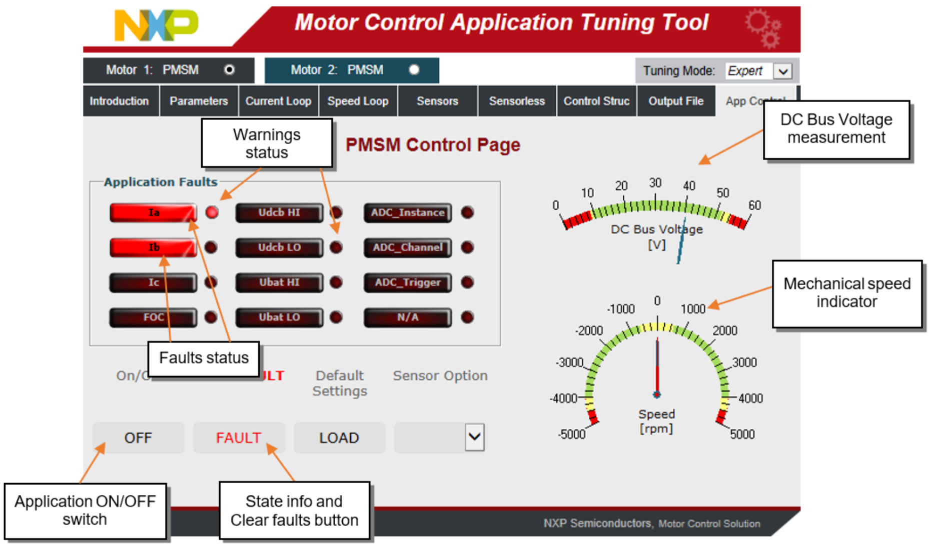


Figure 88. FreeMASTER MCAT Control Page for controlling the application

## 6 Conclusion

Design, described in this application note shows the simplicity and efficiency in using the S32K344 microcontroller for Dual Sensorless as well as sensor-based PMSM motor control and introduces it as an appropriate candidate for various applications in the automotive area. Moreover, support of encoder or resolver 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

This boiler plate is not ready for publication until the steps found in the Freescale Trademark Attribution Worksheet have been completed.

Note also that the ARM attribution must be updated from “ARMnnn” to reflect the correct product name.

## 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:

## 9 Revision history

**Table 7. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| AN14429 v.2.0 | 05 June 2025 | Updated the pinout information in the "S32K344 module interconnection for Motor#1" and "S32K344 module interconnection for Motor#2" images to match the information in the "Port control & pin configuration" topic |

Table 7. Revision history*...continued*

| Document ID | Release date | Description |
|---|---|---|
| AN14429 v.1.0 | 19 March 2025 | Initial release |

## Legal information

## Tables

## Figures

## Contents