AN14454

3-phase Sensorless PMSM Motor Control with S32M244

Rev. 1.0 — 19 October 2024 **Application note**

Document information

Information	Content
Keywords	PMSM, S32M244, Motor Control Application Tuning (MCAT) tool
Abstract	This application note describes the design of a 3-phase Permanent Magnet Synchronous Motor (PMSM) vector control (Field Oriented Control - FOC) drive with single shunt current sensing with and without position sensor.

1 Introduction

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

This design serves as an example of motor control design using the NXP family of automotive motor control MCUs based on a 32-bit ARM® Cortex®-M4 with floating point unit, optimized for a full range of automotive applications.

Following are the supported features:

- 3-phase PMSM speed Field Oriented Control
- Current sensing with a single shunt resistor
- Shaft position and speed estimated by a sensorless algorithm or, optionally, by an encoder position sensor
- Application control user interface using FreeMASTER debugging tool
- Motor Control Application Tuning (MCAT) tool

2 System concept

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

- Targeted at the S32M24x PMSM/BLDC motor control evaluation boards (S32M24xEVB-C064 or S32M24xEVB-L064) used together with BLDC low voltage motor control accessory kit (see section References for more information).
- S32 Configuration Tools (S32CT) used as the S32M244 device configuration and control tool being a part of the S32 Design Studio for S32 Platform IDE (see section References).
- Control technique incorporating:
  - Field Oriented Control of 3-phase PM synchronous motor with/without position sensor.
  - Flux and torque independent control.
  - Bi-directional rotation.
  - Field weakening control extending the speed range of the PMSM beyond the base speed.
  - Open-loop start up with 2-step rotor alignment.
  - Position and speed are estimated by an extended BEMF observer or obtained by Encoder sensor.
  - Reconstruction of three-phase motor currents from a single shunt resistor
  - FOC state variables sampled with 100 µs period.
  - Closed-loop speed control with action period 1 ms.
  - Closed-loop current control with action period 100µs.
- Automotive Math and Motor Control Library (AMMCLib) - FOC algorithm built on blocks of precompiled SW library (see section References).
- FreeMASTER software control interface (motor start/stop, speed setup).
- FreeMASTER software monitor.
- FreeMASTER embedded Motor Control Application Tuning (MCAT) tool (motor parameters, current loop, sensorless parameters, speed loop) (see section References).
- 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 (reconstructed motor currents, vector control algorithm quantities).

- DC-Bus over-voltage and under-voltage, over-current, overload, and start-up fail protection.

3 PMSM field oriented control

3.1 Fundamental principle of PMSM FOC

The description of a fundamental PMSM FOC principles can be found in NXP application note AN12235: *3-phase Sensorless PMSM Motor Control Kit with S32K144* (see section References).

3.2 Output voltage actuation and phase current measurement

The 3-phase voltage source inverter shown in Figure 1 uses single DC Bus shunt resistor (R134 and R135 in parallel), refer to the S32M24xEVB schematic - see section References.

DC Bus current flowing through the shunt resistor produces a voltage drop, which is interfaced to the AD converter of the microcontroller via the integrated Digital Programmable Gain Amplifier (DPGA) integrated inside S32M244.

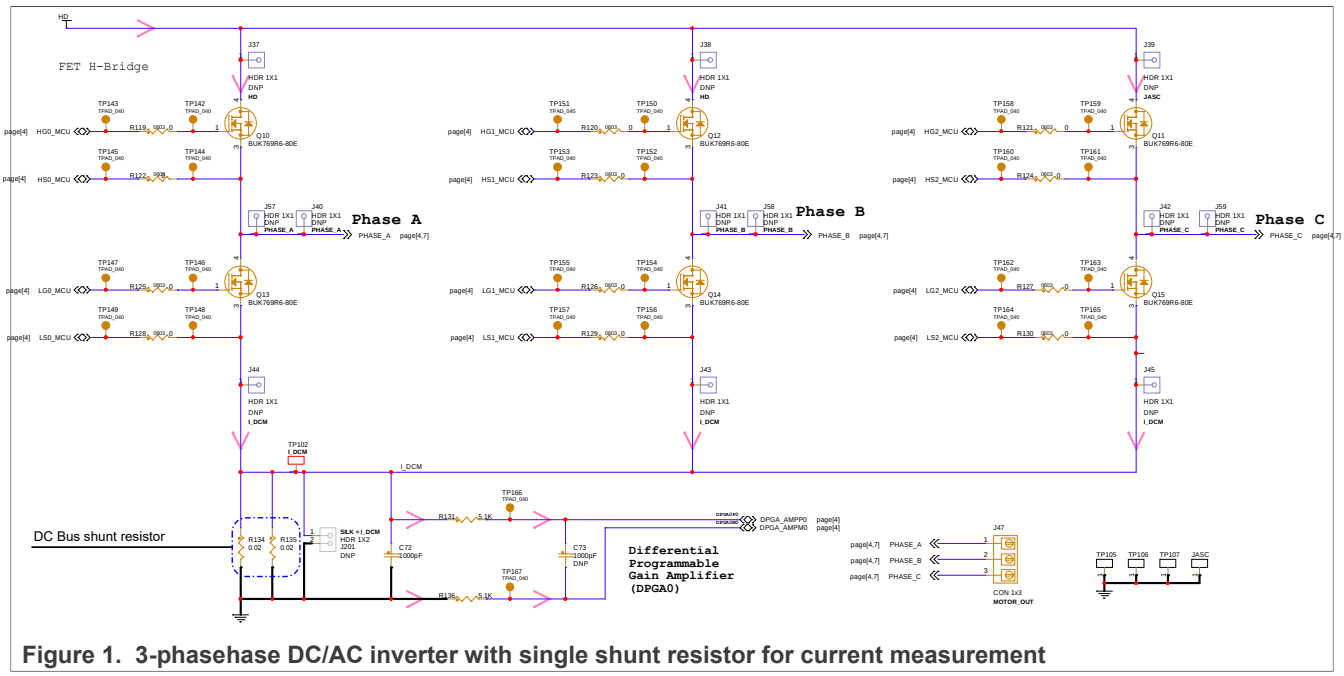

Figure 1. 3-phasehase DC/AC inverter with single shunt resistor for current measurement

Since there is only the DC Bus current measured, the phase motor currents have to be reconstructed by software. Various approaches of phase current reconstruction can be found in NXP application note AN14164: *Current Sensing Techniques in Motor Control Applications* (see section References).

This software example employs an adaptive double switching approach for current sensing and reconstruction described in the aforementioned AN14164.

As described in AN14164, the method of single shunt current measurement relies upon the fact that while there is a switching combination in the inverter forming an active vector, the DC Bus current is equal to one of the motor phase currents or its inverted value, meaning that during the period in which an active vector is present, one phase current can be measured. The desired output voltage vector is created by a vector combination of two neighboring active vectors (and zero vectors), therefore in each PWM period two phase currents can be measured (and the third phase current is calculated). Unfortunately, either in case of low demanded voltage amplitudes and/or in the vicinity of SVM sector borders, the duration of the active vector (current sampling window) is not long enough (or there is only one active vector present) to measure reliably the respective phase

current. See Figure 2, where the current, which we are able to measure and reconstruct as phase motor current is denoted as $i_{RDCsense}$.

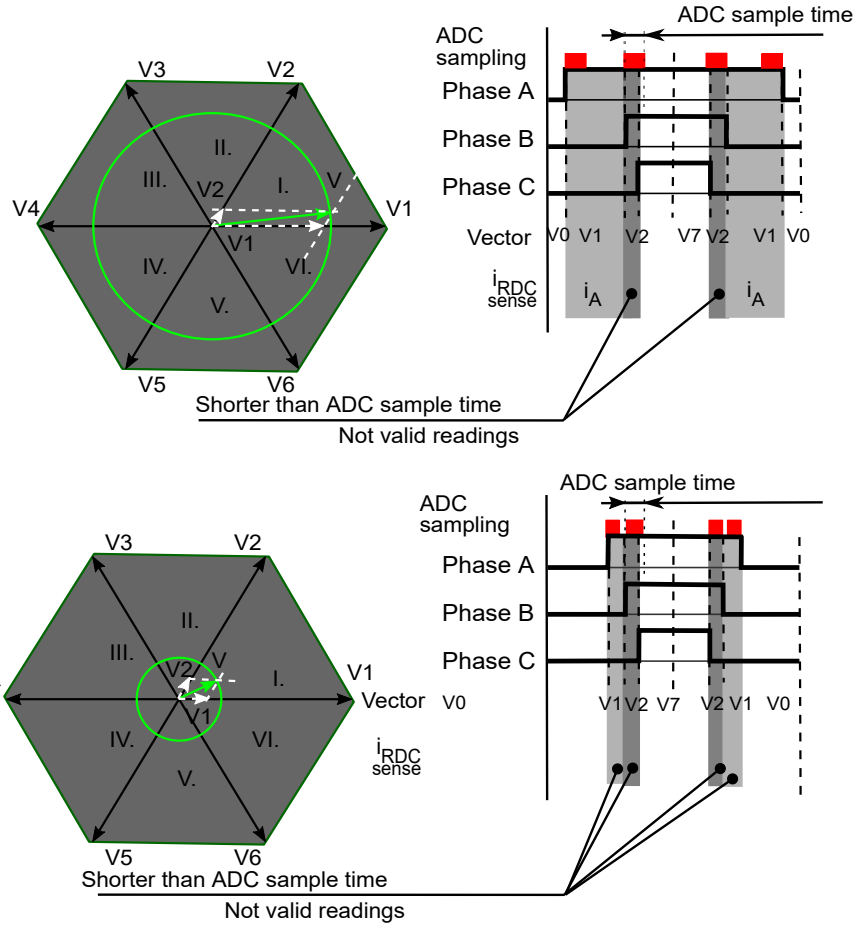

Figure 2. Operation in the vicinity of sector border (top) and operation with low voltage amplitude (bottom)

Since the FOC needs to have the information of the phase currents always, one of the single shunt techniques have to be used to always obtain information on phase currents.

In this example, adaptive double switching is used. The principle of the double switching method is the creation of an appropriate sampling window in two stages:

1. Insertion of zero pulse to the middle of the PWM patterns. This divides the PWM period into two halves and creates two symmetrical half-pulses per phase per PWM period. The sum of the two half-pulse lengths needs to be the same as the length of the original pulse. The width of the zero pulse needs to be stipulated so that the inverter transistors are reliably switched on and off.
2. Shifting the halves of one of the phase patterns (which form an insufficient sampling window) to the sides, therefore enlarging the sampling window to be able to measure the current reliably. The phase with the duty cycle of the shortest length is kept as is and either the halves of the phase with the longest (see the example shown in Figure 3) or mid-length duty cycle are shifted to the sides (in case of low voltage amplitude both longest and mid-length pulses are shifted). As shown in Figure 3, the double switching algorithm not only extends the window for current measurement (V1 extension) but also introduces new vectors (V4) in the way the vector sum gives the same resulting voltage vector as without double switching. The voltage vectors present during double switching are shown in Figure 3 in blue.

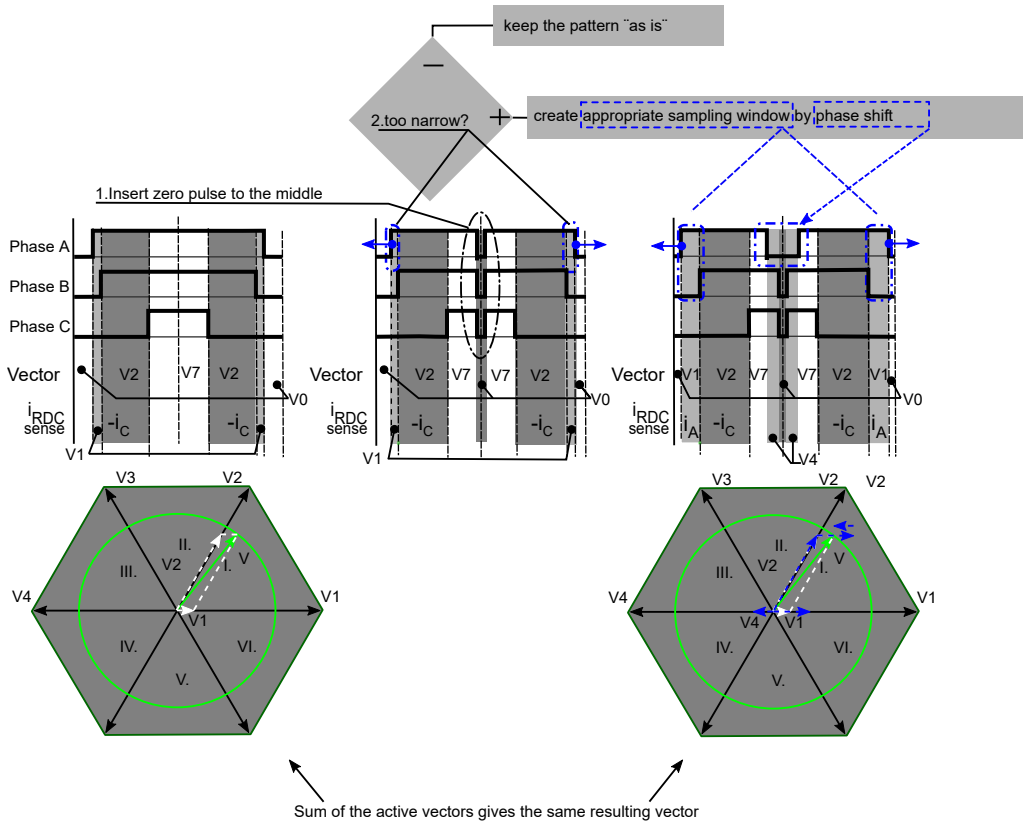

Figure 3. Double switching – shifting the phase with the longest duty cycle duration to the sides. Original PWM (left), inserted zero pulse (middle), final double-switched PWM (right)

The algorithm of double switching can be visualized as per Figure 4.

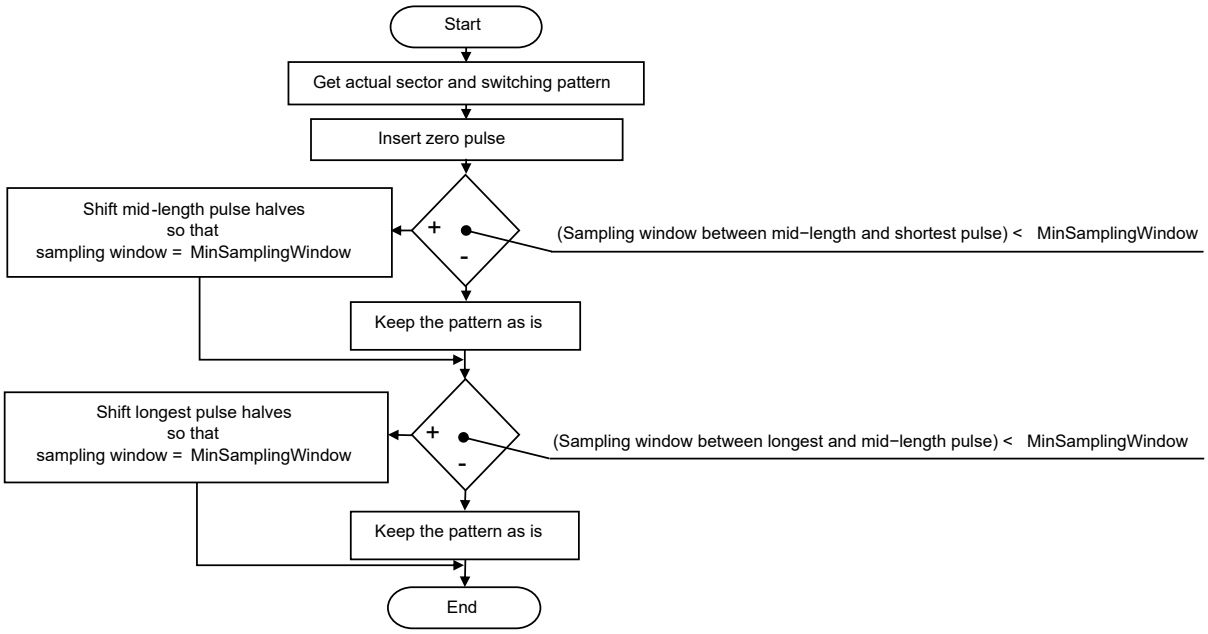

Figure 4. Double switching flowchart

On top of the double switching algorithm, the adaptive double switching algorithm has been introduced, which only applies the double switching algorithm when the window for current measurement is not sufficient, otherwise the standard PWM (without the zero pulse) is used.

The calculation of the double switching algorithm is performed in *src/actuate_s32m.c* in *ACTUATE_SetDutyCycle()* function. The function calculates the edges of the PWM pattern for each particular phase for each SVM sector according to the double switching algorithm (Figure 4). It also calculates the triggers for ADC current measurements, as per Figure 5.

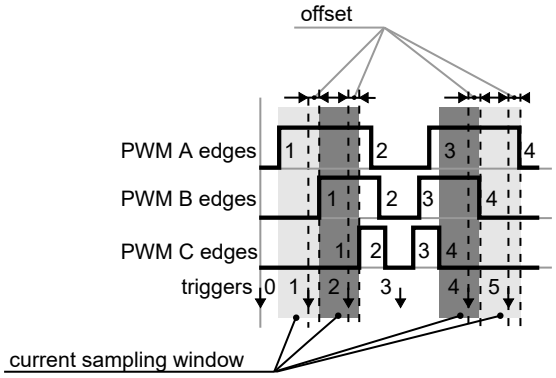

Figure 5. Phase PWM edges and triggers calculation

In S32M244, the double switching PWM pattern needs to be created by two consecutive PWM timer cycles, as described in 4.2.2.1. Thus the function *ACTUATE_SetDutyCycle()* calculates two sets of PWM edges (1, 2 and 3,4) which are loaded to the PWM timer in either odd or even PWM cycle. This happens during FTM3 reload ISR, which occurs as per Figure 7. The newly calculated two sets of PWM edges are refreshed using the function *ACTUATE_PwmUpdateBuffer()*, which is called in PDB0 ISR, see the timing diagram Figure 7.

The triggers for current measurement (triggers 1, 2, 4, 5) are calculated for each current sampling window in the way there is an offset inserted from the end of the particular current sampling window. As per the example shown in Figure 5, the respective window ends are defined by PWM B edge 1, PWM C edge 1, PWM A edge 4 and PWM B edge 4.

3.2.1 Double switching configuration

The SW example, as mentioned before, uses an adaptive double switching approach for current reconstruction. The adaptive double switching (employing the double switching pattern only when necessary) is activated by macro DOUBLE_SW_ADAPTIVE. If the macro is set to zero, then the double switching pattern is active permanently. Even though the double switching has been tuned for S32M24xEVB in cooperation with the BLDC PMSM low voltage motor control accessory kit, for a different hardware or motor it may be needed to tune the parameters. These can be found in *src/actuate_s32m.c*:

- *minZeroPulseCnt* – this is half of the zero pulse inserted in the middle of a double-switching PWM pattern. It needs to be set so the MOSFETs are reliably able to insert the zero pulse in the middle of the PWM switching pattern, i.e.: not too short.
- *minSamplingPulseCnt* - this is the minimal length of the current sampling window. It needs to be set so the ringing from MOSFET switching does not affect the phase current reconstruction. I.e.: set the parameter higher if you are seeing ringing from switching affecting the reconstructed motor phase currents.
- *minSumPulseCnt* – is the sum of *minZeroPulseCnt* and *minSamplingPulseCnt*.
- *pdbTriggerOffset* - is the sampling time of the ADC, not including the conversion time, this parameter defines the offset mentioned in Figure 5.

3.3 Rotor position/speed estimation

In this application, rotor position and speed are either estimated by back-EMF observer or obtained by an Encoder sensor. Back-EMF observer and incremental Encoder sensor provide only relative position. To get an absolute position, the initial position must be known. This application uses mechanical rotor alignment when the rotor is moved from an unknown to a known position applying DC align voltage.

The alignment algorithm applies DC voltage to d-axis resulting full DC voltage applied to phase A and negative half of the DC voltage applied to phase B, C for a certain period. This causes the rotor to move to the “align” position, where stator and rotor fluxes are aligned. The rotor position in which the rotor stabilizes after applying DC voltage is set as zero position. Motor is ready to produce full startup torque once the rotor is properly aligned. The detailed alignment explanation can be found in the chapter State – ALIGN.

The application, while in Sensorless mode, must start with an open loop start-up sequence to move the motor up to a speed value where the observer provides sufficiently accurate speed and position estimations. When the observer provides appropriate estimates, the application transits to closed-loop mode, when the rotor speed and position calculation are based on the estimation of a BEMF in the stationary reference frame using a Luenberger type of observer. BEMF observer is as a part of the NXP’s Automotive Math and Motor Control Library. Structure and implementation details are discussed in section 4.3.4.

3.4 Field weakening

The description of field weakening principles can be found in NXP application note AN12335: *3-phase Sensorless PMSM Motor Control Kit with S32K144* (see section References).

4 Software implementation on the S32M244

4.1 S32M244 – key modules for PMSM FOC control

The S32M244 is an integrated solution, which comprises Digital part (in this AN referred to as MCU) and Analog Extension part (in this AN referred to as AE). The Analog Extension part includes modules such as Gate Driver Unit (GDU) and Digital Programmable Gain Amplifier (DPGA) which allow for using S32M244 in motor control applications with a minimum of external components. The MCU part includes modules such as the FlexTimer Module (FTM), Trigger MUX Control (TRGMUX), Programmable Delay Block (PDB) and Analogue-to-Digital Converter (ADC) suitable for motor control applications. These modules are directly interconnected and can be configured to meet various motor control application requirements. Figure 6 shows module interconnection for a typical PMSM FOC application working in Sensorless or Sensor based mode using single shunt current sensing. The modules are described below and a detailed description can be found in the S32M24x Reference Manual (see section References).

4.1.1 Module interconnection

As mentioned earlier, S32M24x consists of two parts: MCU and AE, which are connected together via die-to-die (D2D) connections.

This includes an SPI interface, which serves for parameter settings and status monitoring of AE and connections used for motor control loop such as connections between FTM3 and GDU, DPGA, and ADC, AE fault monitoring via GPIO (PTD3), see Figure 6.

The AE comprises modules vital for motor control. The GDU drives the power MOSFETs. The GDU is interconnected with the outputs of FTM via D2D connections.

The shunt resistor voltage drop signal (proportional to DC Bus current) is conditioned by the integrated DPGA which output is connected to the ADC via D2D connection.

DC Bus voltage is conditioned via an integrated voltage divider which output is connected to the ADC via D2D connection.

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

Each control loop cycle can be initiated either by FTM initialization trigger *init_trig* or by FTM external trigger *ext_trig*. While *init_trig* signal is generated at the beginning of the PWM cycle, *ext_trig* can be generated anytime within the PWM period based on the value defined in the corresponding FTM Channel Value register CnV.

FTM trigger signal is routed to the hardware trigger input of the PDB module through the flexible TRGMUX unit.

PDB pre-triggers *ch0pretrigx* are used as a precondition for ADC module. They are directly connected to ADHWTS ports to select ADC channels and order of the channels by configurable pre-triggers delays. When ADC receives the rising edge of the trigger, ADC will start conversion according to the order defined by pre-triggers *ch0pretrigx*.

PDB pre-trigger delays must be properly set to allow reliable operation between PDB and the corresponding ADC module. When the first pre-trigger is asserted, the associated lock of the pre-trigger becomes active until the corresponding conversion is not completed. This associated lock is released by corresponding ADC conversion complete flag ADC_SC1[COCOx]. This means that the next pre-trigger can be generated only if the ongoing conversion is completed.

The second FTM module can work in Quadrature Decoder mode, counting the rising/falling edges of the Phase A and Phase B encoder signals to determine the rotor position and speed independently from the control loop (see section 4.2.2.2).

A detailed description can be found in the S32M24x Reference Manual (see section References).

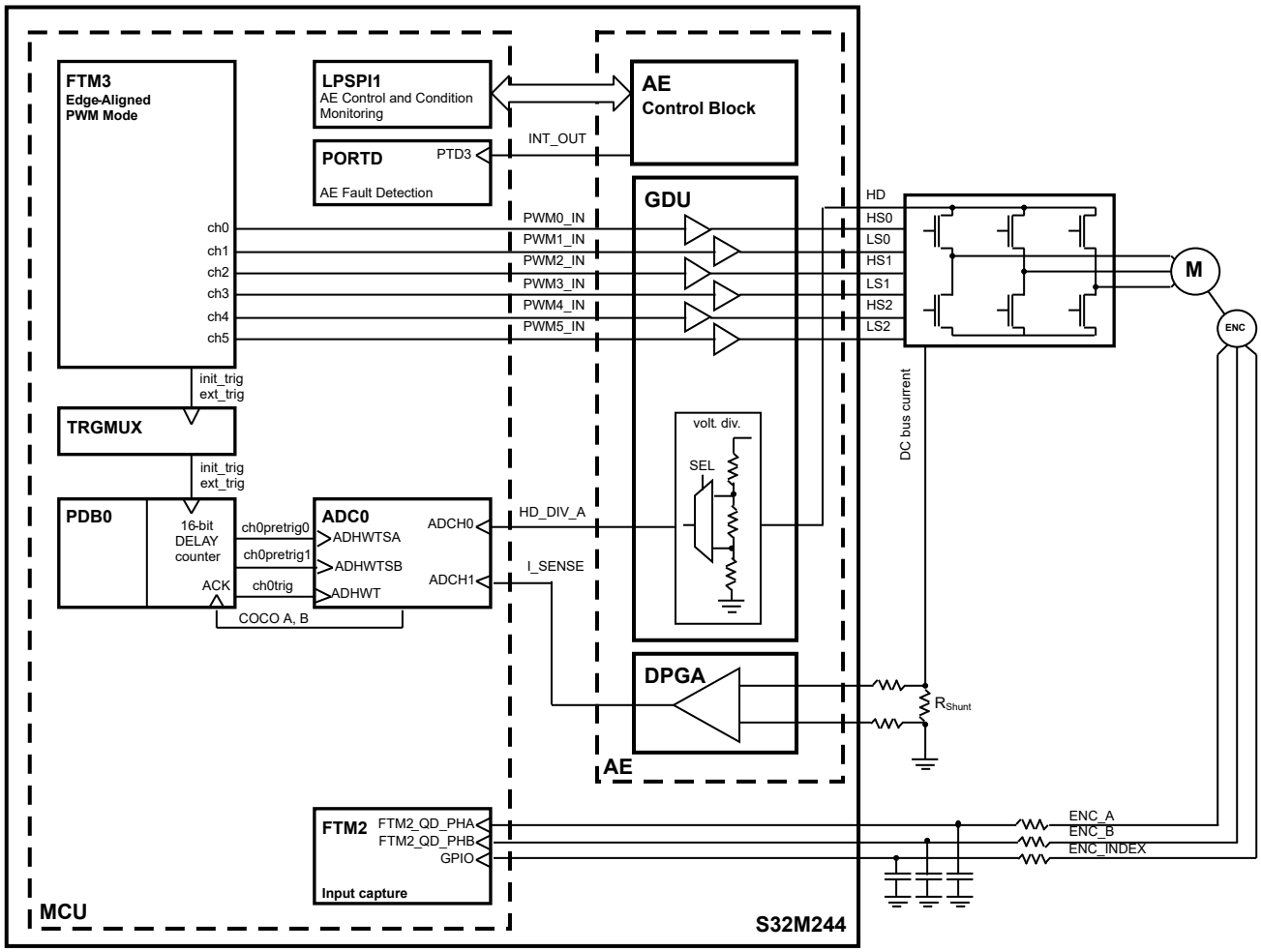

Figure 6. S32M244 module interconnection

4.1.2 Module involvement in digital PMSM sensorless control loop

This section discusses timing and modules synchronization to accomplish PMSM Sensorless FOC on the S32M244 and the internal hardware features.

The timing diagram of the automatic synchronization between PWM and ADC in the PMSM application is shown in Figure 7.

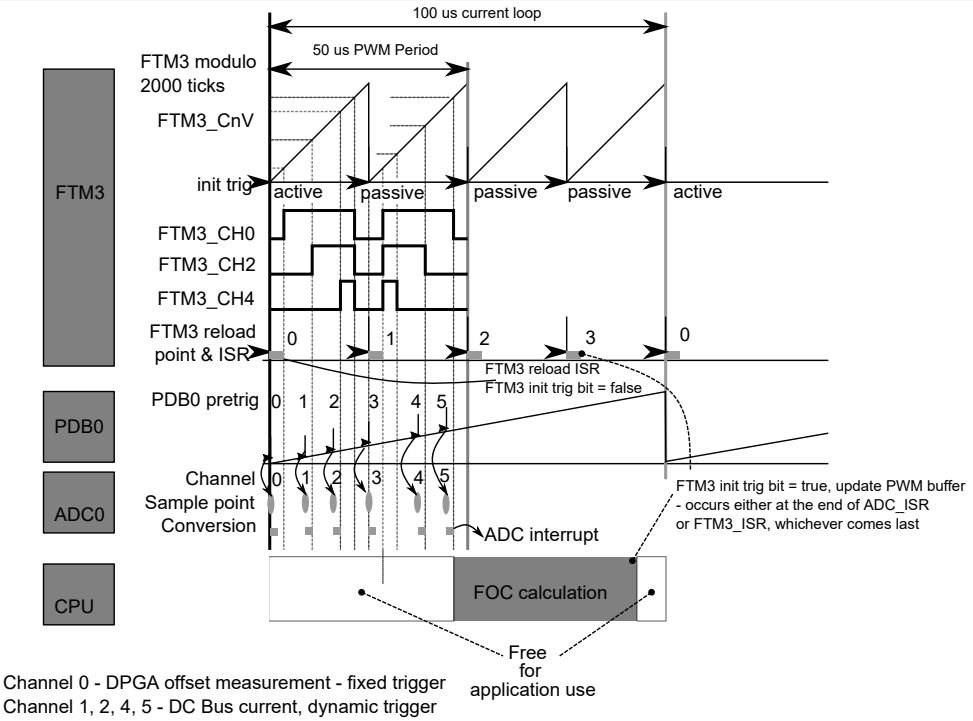

Figure 7. Time diagram of PWM and ADC synchronization

The PMSM Sensorless FOC control with single shunt current measurement with double switching approach is based on dynamic timing; meaning the trigger point instances of the ADC conversions are varying from one FOC control cycle to the next cycle, depending on the three phase PWM edges timing.

There are, however, two fixed trigger points: in the beginning of the PWM period for DPGA offset measurement (pretrig 0) and in the middle of the PWM cycle for DC Bus voltage measurement (*pretrig 3*).

Pretrig0 is disabled when values of *Pretrig1* are too low in order to avoid PDB0 errors, see 4.3.3.4.1.

Each control cycle starts with FTM3 initialization trigger *init_trig*, which is generated at the beginning of the PWM cycle as shown in Figure 7. Initialization trigger restarts the PDB0 module and updates its double buffered registers. ADC0 channels are triggered based on the PDB0 pre-trigger delays. When the PDB counter reaches the first pre-trigger delay value, PDB initiates the first ADC channel measurement.

In the beginning of the PWM cycle, the measurement of DPGA offset is triggered (*pretrig 0*). DC Bus current measurement is triggered by PDB0 at the first sampling window (*pretrig 1*) in which Phase A current is visible. DC Bus current measurement is triggered again by PDB0 at the second sampling window (*pretrig 2*) in which Phase C current is visible. Then, the DC Bus voltage measurement is triggered at the middle point of the PWM (*pretrig 3*). The fifth trigger point (*pretrig 4*) is for DC Bus current sampling when Phase C current is visible. And the sixth trigger point (*pretrig 5*) is for DC Bus current sampling when Phase A current is visible. The ADC conversion results are automatically stored into a predefined queue in memory. The ADC results measured at *pretrig 1* and *pretrig 5* are averaged, similarly to the ADC results measured at *pretrig 2* and *pretrig 4* since in both of the cases the current of the same phase is visible.

The CPU is triggered by the ADC0 conversion complete interrupt service routine. Based on the stored ADC0 values, the current PI controllers calculate new PWM duty cycles. These are then sent as a new reference for PWM module (FTM3) and become effective in the next PWM cycle.

The FTM3 initialization trigger is disabled in the FTM3 reload interrupt service routine. As a consequence, PDB0 is not triggered in the next PWM period due to the missing init_trig signal. FTM3 initialization trigger is reenabled again in either 3rd FTM3 ISR or in the end of ADC ISR - whichever comes last, since the timing is dynamic and

the beginning of ADC ISR is not fixed, at low duty cycles the ADC ISR may be finished earlier than 3rd FTM3 ISR. This strategy ensures that ADC0 sampling occurs in every 100 µs cycle as depicted in Figure 7.

It needs to be noted that double switching is not able to operate with a 100% duty cycle, since there has to be room left for performing the pulse shifting as a part of the double switching algorithm. The limit has to be set with respect to the minimal length of the current sampling window and if set too high the reconstructed currents will be distorted/not accurate at high duty cycles. The limit is set in *src/main.c* by writing into the variable *drvFOC.AlBeReqDCBLim.fltLimit*. The limit is set to 0.9 in the SW example. The maximum theoretical duty cycle limit is 93%.

4.2 S32M244 device initialization

To simplify and accelerate application development, an embedded part of the PMSM Sensorless motor control application has been created using S32K1_S32M24x - Real-Time Drivers for Cortex-M (RTD). S32M244 can be configured either by means of the S32 Configuration Tool (S32CT) or programmed directly using RTD drivers. Peripherals are initialized at the beginning of the *main()* function. For each S32M244 module, there is a specific configuration function that uses S32M244 RTD APIs and configuration structures generated by S32CT to configure the MCU:

- *McuClockConfig()* – MCU clock configuration
- *McuCacheConfig()* – MCU cache configuration
- *McuPowerConfig()* – MCU power management configuration
- *McuIntConfig()* – MCU interrupt management configuration
- *McuTrigmuxConfig()* – TRGMUX module configuration
- *McuPinsConfig()* – PINs and PORT modules configuration
- *McuLpuartConfig()* – LPUART module configuration
- *McuAdcConfig()* – ADC modules configuration
- *McuPdbConfig()* – PDB modules configuration
- *McuFtmConfig()* – FTM modules configuration
- *McuSPIConfig()* – SPI module configuration
- *AECConfig()* – AE module configuration
- *AEC_DPGAConfig()* – DPGA module configuration
- *AEC_GDUConfig()* – GDU module configuration
- *AEC_HVMConfig()* – HV module configuration

Detailed RTD documentation can be found in the folder created with the S32 Design Studio installation (see section References). It is recommended to keep calling of the initialization functions in the order in which they appear in the example code.

4.2.1 Clock configuration and power management

S32M244 features a complex clocking sourcing, distribution, and power management. To run a core of the S32M244 and some MCU peripherals at maximum frequency 80 MHz in normal RUN mode, the S32M244 is supplied externally by 16 MHz crystal. This clock source supplies a Phase-lock-loop (PLL), which circuit multiplies frequency by 20 and divides by 2 resulting 160 MHz frequency on output. PLL output is then divided by 2 to supply core and system (80 MHz), further divided by two and four to supply bus clock (40 MHz) and flash clock (20 MHz), respectively. This clock configuration belongs to one of the typical and recommended. It is summarized in Table 1.

Table 1. S32M244 clock configuration in RUN mode

Clock	Frequency
CORE_CLOCK	80 MHz

Table 1. S32M244 clock configuration in RUN mode...*continued*

Clock	Frequency
SYS_CLK	80 MHz
BUS_CLK	40 MHz
FLASH_CLK	20 MHz (max frequency in RUN mode)

The clock configuration can be set up by the S32M244 RTD clocks graphical tool.

Once the clock configuration is set, RTD generates a static configuration structure *Clock_Ip_aClockConfig[0]*, the respective RTD API is then called in *MCUClockConfig()*.

Example 1. S3M244 clock configuration controlled by RTD

```
/**********************************************************************************
 *
 * Function:  void McuClockConfig(void)
 *
 * Description:  This function installs the pre-defined clock configuration
 table
 *     to the clock manager. For more details see configuration
 *     in Config Tools.
 *
 **********************************************************************************/
void McuClockConfig(void)
{
Clock_Ip_Init(&Clock_Ip_aClockConfig[0]);
}
```

As discussed at the beginning of this chapter, power management of the S32M244 is configured for normal RUN mode. This power mode can be set in the RTD peripherals graphical tool, Figure 8.

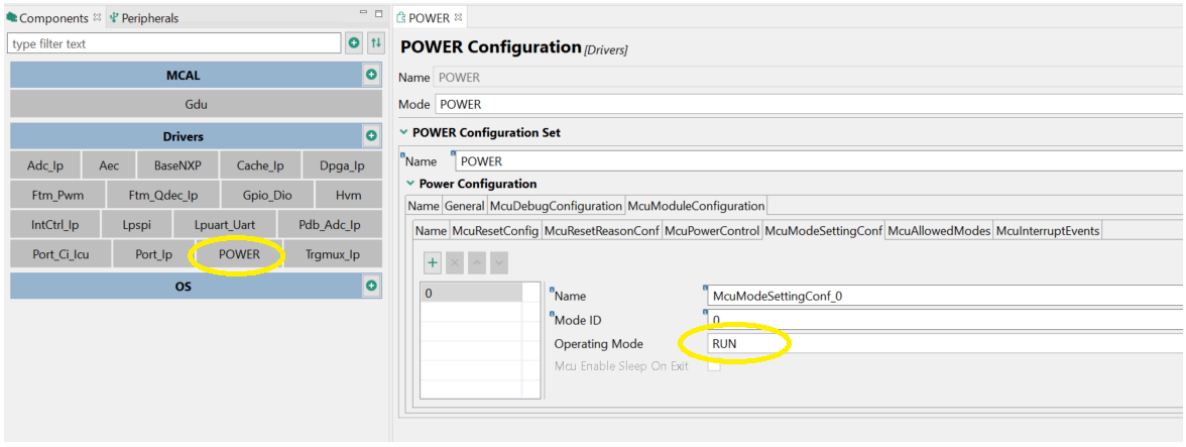

Figure 8. S32M244 power management configuration in S32CT peripherals tool

Static configuration generated by RTD is called by the respective RTD API and this is encapsulated in the *MCUPowerConfig()* function.

Example 2. S32M244 power management controlled by RTD

```
/**********************************************************************************
 *
 * Function:  void McuPowerConfig(void)
 *
```

```
 * Description:  This function configures the Power manager for operation.
 *     For more details see configuration in Config Tools.
 *
 *********************************************************************************/
void McuPowerConfig(void)
{
/* Power mode configuration for RUN mode */
Power_Ip_SetMode(&Power_Ip_aModeConfigPB[0]);
}
```

The same mechanism for peripherals configuration by RTD works for all S32M244 peripherals, which are discussed below.

4.2.2 FlexTimer Module (FTM)

The FlexTimer module (FTM) is built upon a timer with a 16-bit counter. It contains an extended set of features that meet the demands of motor control, including the signed up-counter, dead time insertion hardware, fault control inputs, enhanced triggering functionality, and initialization and polarity control.

4.2.2.1 Edge-aligned PWM mode

FTM3 instance outputs are routed to GDU inputs, therefore FTM3 is used in PMSM sensorless motor control application to generate center-aligned double switching PWM by six, complementary oriented channels to control power MOSFETs.

However, since a double switching approach for single shunt current measurement is used, the advanced (center-aligned) PWM pattern is generated by edge aligned mode of FTM3. One PWM cycle of the double switching (50us) is created by two consecutive FTM3 cycles (25us) as depicted in Figure 7.

As depicted in Figure 7, up counting mode is selected as a dedicated counting mode for FTM3 edge-aligned PWM. The 40 kHz FTM3 PWM frequency is adjusted by FTM3 Modulo register (FTM3_MOD = 2000) taking 80 MHz clock source frequency into account. To protect power MOSFETs against short circuit, dead time 0.5µs is inserted for each complementary channels pair in the number of clock ticks 40 with default dead time prescaler *Divide_by_1*. This FTM3 configuration can be carried out by using RTD, FTM_Pwm module, Figure 9. Three complementary output channel pairs are configured in the tab *PwmFtmCh*, as depicted in Figure 9. For detailed settings, please see all the settings tabs in the *Ftm_Pwm module* in S32CT peripherals tool.

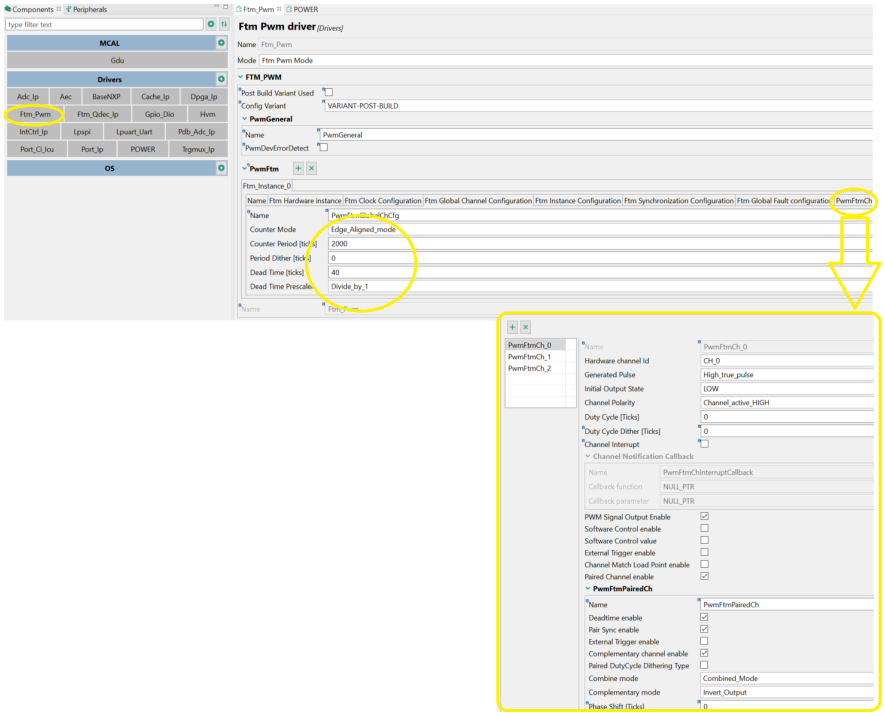

Figure 9. S32M244 FTM3 configuration in S32CT

As discussed in chapter 4.1.2, to initiate the control loop every second PWM cycle at the beginning of the PWM period, the initialization trigger is enabled. To be able to synchronize PWM and update FTM double buffered registers at certain synchronization point simultaneously, *Software_sync_trigger* and *Trigger_on_Reload_Point* are enabled in *FTM instance configuration* and *FTM synchronization* configuration tab, Figure 9. It should be noticed that the max loading point is the time instant, when FTM3 counter equals modulo register value (FTM3_MOD = 2000).

Once the FTM3 setting is completed, RTD generates configuration structure *Ftm_Pwm_Ip_BOARD_INITPERIPHERALS_UserCfg3* which, using the respective RTD API *Ftm_Pwm_Ip_Init* configures FTM3, this is encapsulated in function *MCUFTMConfig()*, which can be found in the SW example in *src/Peripherals/peripherals_config.c*.

4.2.2.2 Quadrature decoder mode

The FTM module offers a quadrature decoder mode to decode the quadrature signals generated by rotary sensors used in the motor control domain. This mode is used to process encoder signals and determine rotor position and speed.

There are three output signals generated by the incremental encoder as shown in Figure 10. 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 may be used to check the pulse-counting consistency, however, it is not used in the example.

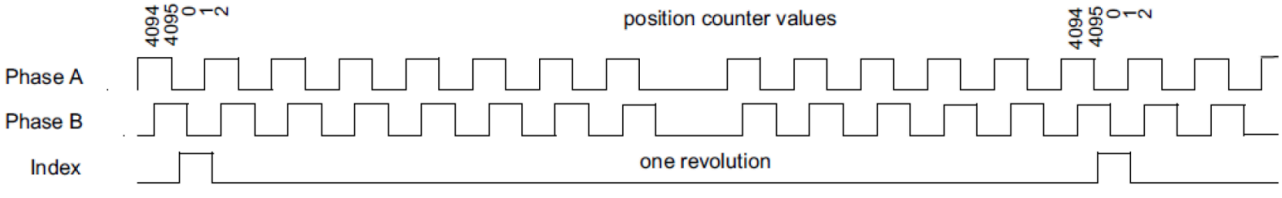

Figure 10. Output signals of the 1024 pulses encoder

To process the Phase A and Phase B signals from the Encoder sensor, quadrature decoder mode with phase encode mode have to be enabled in S32CT, Figure 11. Note that so-called X4 encoding is used, meaning both rising and falling edges of phase A and B are counted, which effectively quadruples the number of encoder pulses. In addition, *Max counter value* has to be set according to the number of the encoder edges. The value has to be set to (2**Number of encoder pulses* -1), for example: in 1024-pulse encoder the max counter value has to be set to 2047. The Min Counter value which for correct angle wrapping needs to be equal to (-2**Number of encoder pulses*) is set in the SW in *src/main.c/AlignState()* since RTD does not allow for setting a negative number. This is to ensure correct motor angle wrapping in the range of <-180; 180> degrees. for example: In 1024-pulse encoder the value The *min counter value* is set to 0xF800 what effectively means -2048 pulses. In quadrature decoder mode, the phase A and phase B signals indicate the counting direction and the counting rate. If the phase B signal lags the phase A signal, the FTM2 counter increments after every detected rising/falling edge of both signals. If the phase B signal leads the phase A signal, the FTM2 counter decrements after every detected rising/falling edge of both signals and the QUADIR bit in the FTM_QDCTRL register indicates the counting direction. To allow for filtering of possible glitches in the encoder signals, the value of the filter can be set in *A/B PhFilterVal* as per Figure 11.

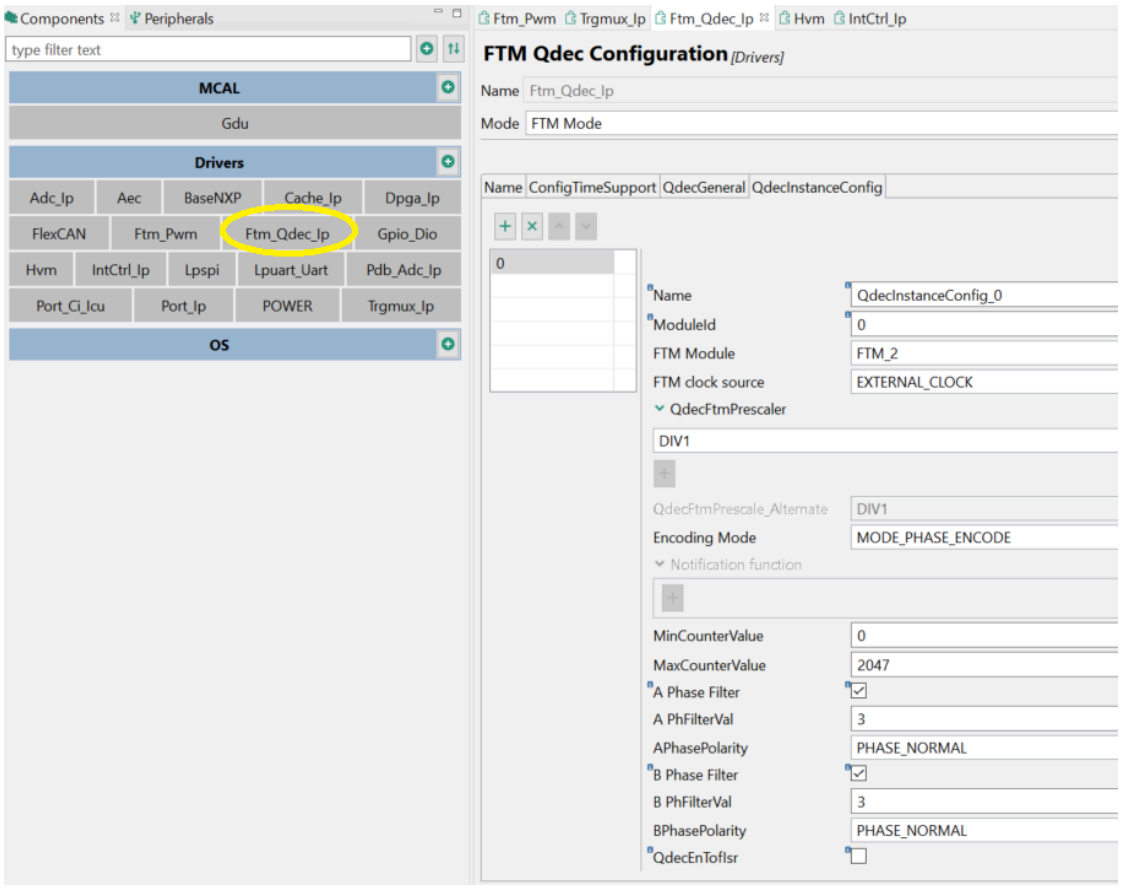

Figure 11. S32M244 FTM2 configuration in S32CT

The configuration structure of the quadrature decoder mode generated by S32CT is called *Ftm_Qdec_Ip_InstanceConfig_BOARD_INITPERIPHERALS[0]*. This structure is used with RTD API *Ftm_Qdec_Ip_Init*, which is called in *StateAlign()* in *src/main.c*.

Note: *The S32M24xEVB board is designed to process encoder signals through the FTM2 module. Software example contains a routine for encoder signal processing. This routine is disabled by default since the motor of the BLDC low voltage motor control accessory kit is not equipped with an Encoder sensor. To enable the encoder signal processing routine, set ENCODER macro to true.*

4.2.3 Trigger MUX control (TRGMUX)

The TRGMUX provides an extremely flexible mechanism for connecting various trigger sources to multiple pins/peripherals. With the TRGMUX, each peripheral that accepts external triggers usually has one specific 32-bit trigger control register. Each control register supports up to four triggers, and each trigger can be selected from the available input triggers.

To trigger the PDB0 module by FTM3 initialization trigger signal *init_trig*, TRGMUX needs to be set appropriately.

S32CT allows to generate configuration structure *Trgmux_Ip_xTrgmuxInitPB* that sets all TRGMUX registers to assign trigger inputs with trigger outputs as demanded, Figure 12.

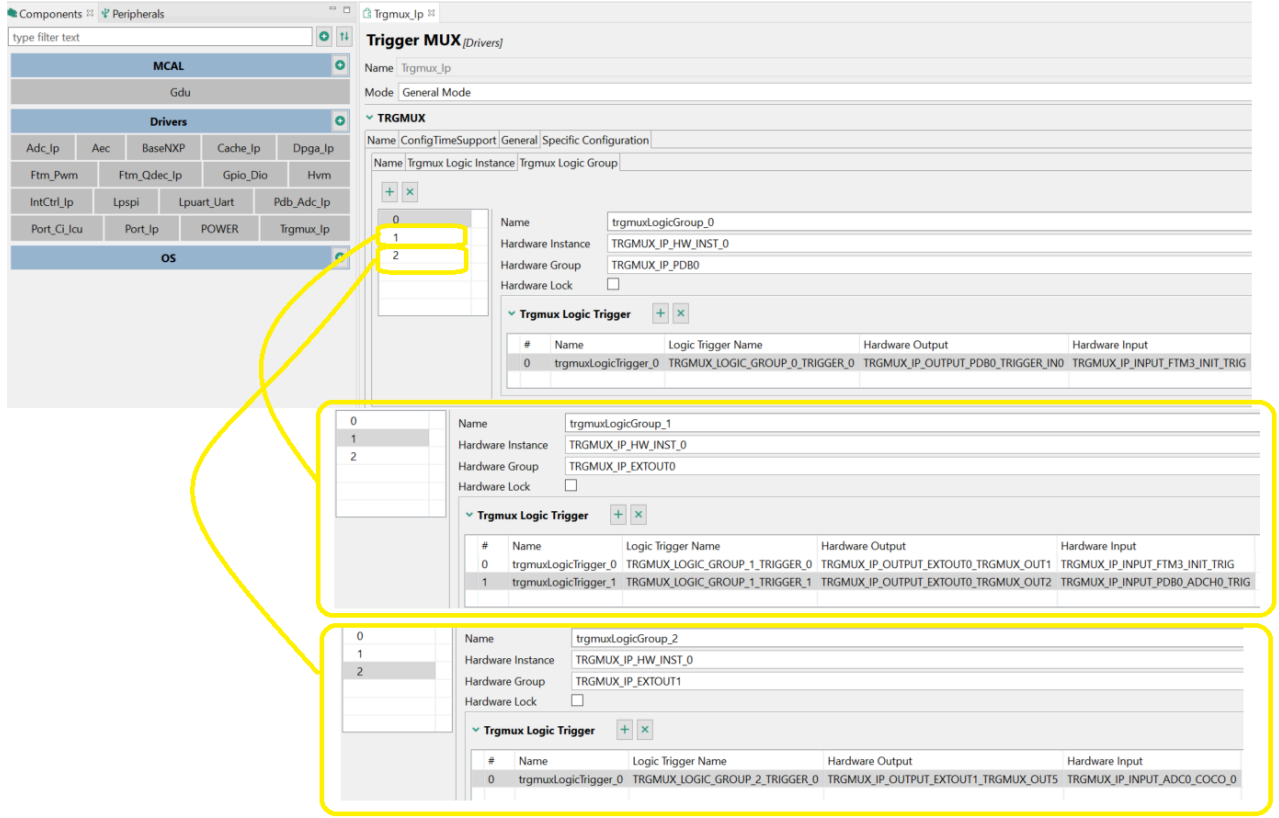

Figure 12. S32M244 TRGMUX configuration in S32CT

In particular, the FTM3 initialization trigger signal as a source is assigned to two targets namely: PDB0 and TRGMUX output 1. PDB0 channel 0 trigger is routed to TRGMUX output 2 and ADC0 conversion complete flag COCO is assigned to TRGMUX output 6. TRGMUX outputs are directly assigned to chip pins, so that the triggering scheme between FTM3, PDB0, and ADC0 can be visualized with oscilloscope as depicted in Figure 7.

4.2.4 Programmable Delay Block (PDB)

The Programmable Delay Block (PDB) is intended to completely avoid CPU involvement in the timed acquisition of state variables during the control cycle. The PDB module contains a 16-bit programmable delay counter that delays FTM3 initialization trigger and schedules ADC channels sampling through PDB pre-triggers delays. When FTM3 initialization trigger is detected on the PDB0 and PDB1 trigger input, PDB0 and PDB1 generate

hardware signal to trigger ADC0 and ADC1 channels in order defined by pre-trigger delays, Figure 13. In this example only PDB0 and ADC0 are used, and PDB1 and ADC1 are free for user specific requirements.

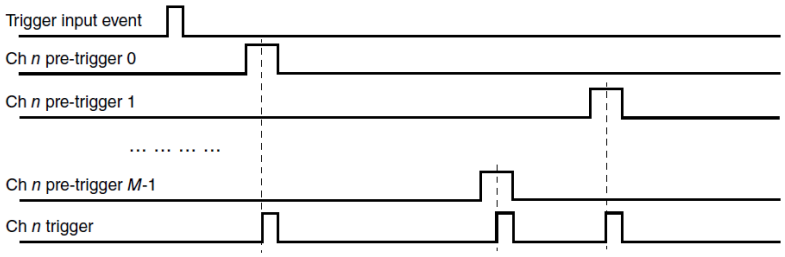

Figure 13. PDB pre-triggers and trigger output

PDB pre-trigger delays can be set independently using CHnDLYm registers. Since the PDB0 and FTM3 modules are synchronized and share the same source frequency 80 MHz, values of the CHnDLYm registers are set using the same time base as for PWM. Table 2 shows all PDB0 pre-triggers used in PMSM sensorless FOC motor control application with single shunt current measurement.

Table 2. PDB0 pre-triggers

FOC state variable	PDB pre-trigger	CHnDLYm value [ticks]	Relation to PWM
DPGA offset	pdb0_ch0_pretrig0	0	The beginning of the PWM. Used for run-time DPGA offset compensation, see 4.3.3.4.1.
Stator current 1, sample 1	pdb0_ch0_pretrig1	dynamic, set according to PWM pattern	1st sampling window for phase current
Stator current 2, sample 1	pdb0_ch0_pretrig2	dynamic, set according to PWM pattern	2nd sampling window for phase current
DC Bus voltage	pdb0_ch0_pretrig3	2000	In ½ of the PWM
Stator current 2, sample 2	pdb0_ch0_pretrig4	dynamic, set according to PWM pattern	3rd sampling window for phase current
Stator current 1, sample 2	pdb0_ch0_pretrig5	dynamic, set according to PWM pattern	4th sampling window for phase current

PDB sequence error interrupt is activated as redundancy to protect the triggering mechanism once blocked due to the wrong PDB pre-trigger delay timing. Pre-triggers delays must respect ADC conversion time that typically takes ~1.25µs considering short ADC sample time and 40 MHz ADC input frequency. This time can be converted to a PDB pre-trigger delay format defined in the number of ticks 100.

Pre-triggers delays are dynamic values, which are changing every 100 µs control cycle according to corresponding PWM edges values.

It should be also noticed that MOD, IDLY, and CHnDLYx are double buffered registers, meaning values are loaded from their buffers based on the selected updating method.

General settings of the PDB module such as clock pre-scaler, input trigger source, loading mechanism for double buffered registers and operation mode for pre-triggers can be configured with S32CT as shown in Figure 14.

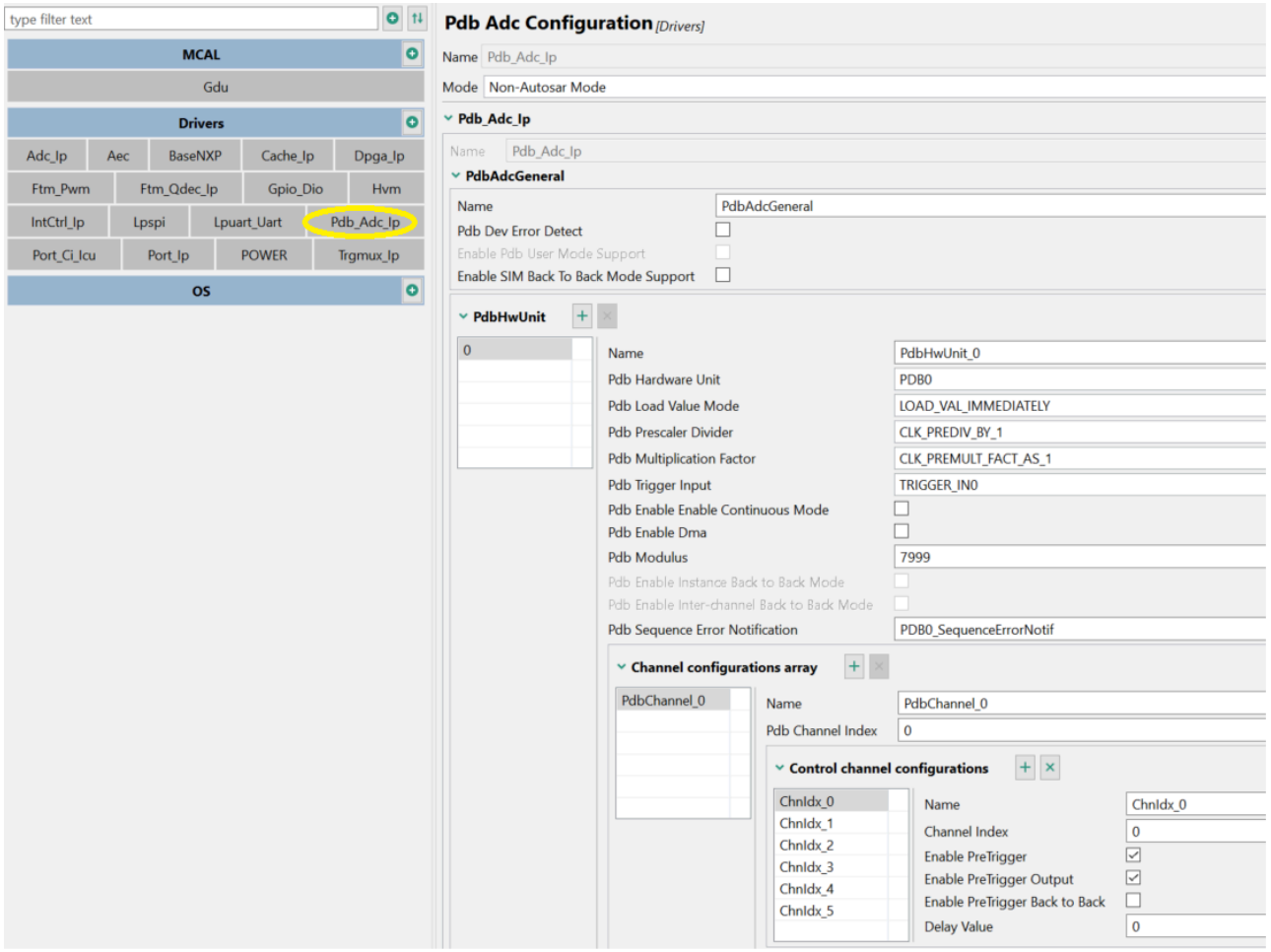

Figure 14. S32M244 PDB0 module and pre-triggers configuration in S32CT

S32CT generates configuration structures *PdbHwUnit_0_BOARD_INITPERIPHERALS* for appropriate PDB registers. This configuration is loaded calling *src/peripherals/peripherals_config.c/McuPdbConfig()*.

4.2.5 Analog-to-Digital Converter (ADC)

The S32M244 device has two 12-bit Analog-to-Digital Converters (ADCs). These are 32-channel multiplexed input successive approximation ADCs with 16 result registers.

ADC channels are sampled in the order defined by PDB pre-triggers. When the first pre-trigger is asserted, the associated lock of the pre-trigger becomes active waiting for the conversion complete flag COCO generated by the corresponding ADC channel. This sequence is repeated for each PDB pre-trigger and ADC channel couple.

The clock source of the ADC module is derived from the system clock frequency, further divided by 2 resulting 40 MHz supply frequency. To combine high conversion resolution and short conversion time, 12-bit resolution mode with sample time 12 clock cycles are set in S32CT, Figure 15.

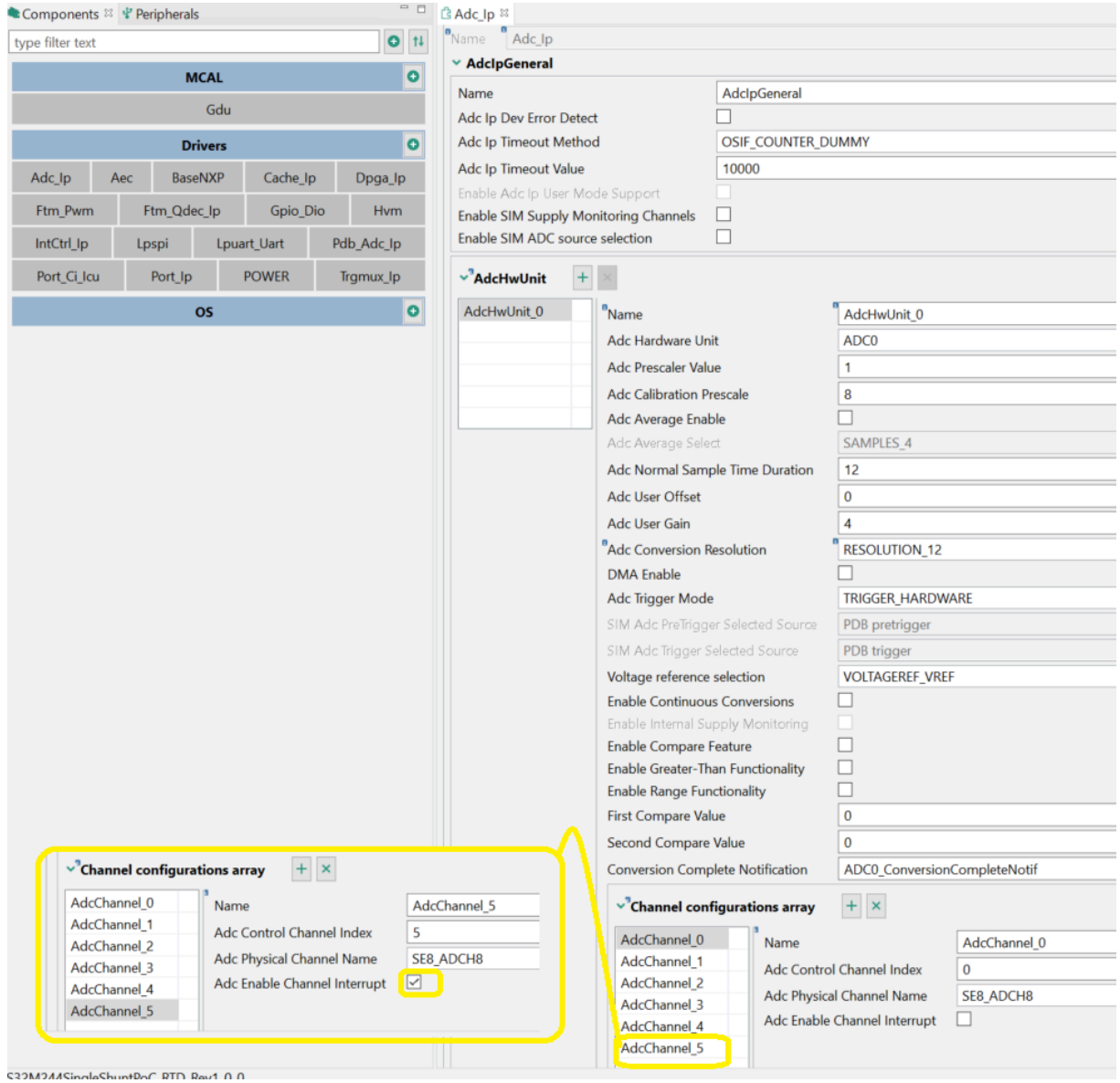

Figure 15. S32M244 ADC0 module and channels configuration in S32CT

Channel 0 measures DC Bus current in zero vector, which serves for run-time DPGA offset measurement, channels 1,2,4,5 measure DC Bus current (signal at DPGA output). Channel 3 measures DC Bus voltage. Channel 5, being the last channel in the list, has also interrupt enabled.

S32CT generates the module configuration structure *AdcHwUnit_0_BOARD_INITPERIPHERALS*, which takes effect calling the respective RTD API in the function *src/peripherals/peripherals_config/McuAdcConfig()*.

4.2.6 Low Power Serial Peripheral Interface (LPSPI)

LPSPI is used as communication interface between S3M244 MCU and AE.

Configuration of LPSPI1 by means of S32CT can be seen in Figure 16. The details on the AE communication protocol are described in S32M244 reference manual (see section References).

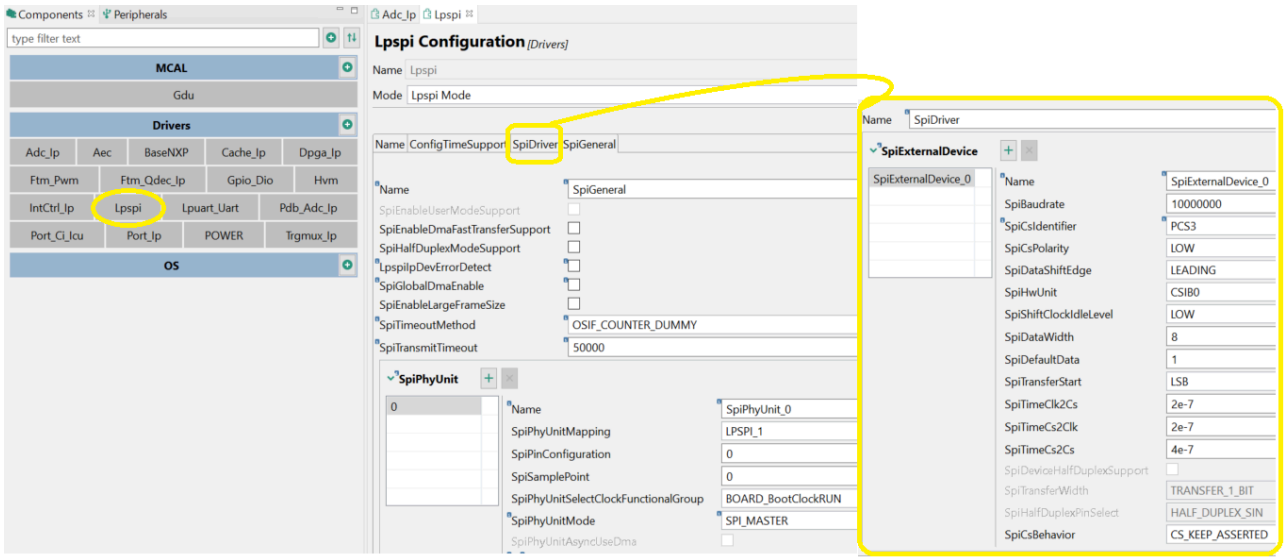

Figure 16. S32M244 LPSPI module configuration in S32CT

LPSPI is then initialized using the S32CT-generated configuration structure in *src/peripherals/peripherals_config/McuSPIConfig()*.

4.2.7 Low Power Universal Asynchronous Receiver/Transmitter (LPUART)

LPUART0 is used as a communication interface between the S32M244 processor and FreeMASTER run-time debugging and visualization tool. Function *src/Peripherals/peripherals_config/McuLpuartConfig()* initializes LPUART0 module with baud rate 115200 bps, 8 bits per channel, no parity, and 1 stop bit. This configuration is carried out by RTD's LPUART driver.

Configuration structure *Lpuart_Uart_Ip_xHwConfigPB_0_BOARD_INITPERIPHERALS* is configured by means of S32CT as shown in Figure 17.

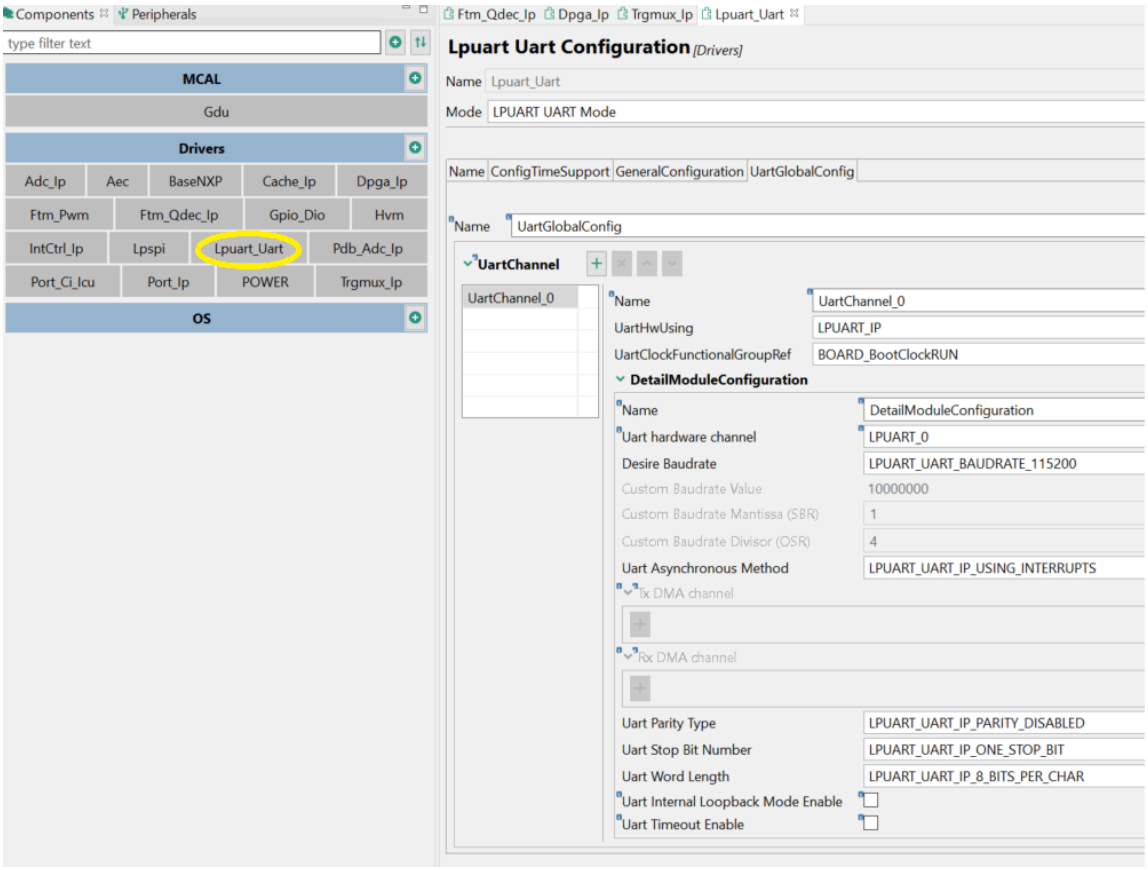

Figure 17. S32M244 LPUART0 module configuration in S32CT

4.2.8 Port control and pin multiplexing

PMSM FOC sensorless motor control application requires following on chip pins assignment, Table 3.

Table 3. Pins assignment for S32M244 PMSM sensorless FOC control

Connection	Peripheral	Signal	Pin	Description
D2D	ADC0	DCBI	PTB13 / ADC0_SE8	DC Bus current
		DCBV	PTA6 / ADC0_SE2	DC Bus voltage
	FTM3	PWMA_HS	PTA2 / FTM3_CH0	PWM A high-side driver
		PWMA_LS	PTA3 / FTM3_CH1	PWM A low-side driver
		PWMB_HS	PTB10 / FTM3_CH2	PWM B high-side driver
		PWMB_LS	PTB11 / FTM3_CH3	PWM B low-side driver
		PWMC_HS	PTC10 / FTM3_CH4	PWM C high-side driver
		PWMC_LS	PTC11 / FTM3_CH5	PWM C low-side driver
	LPSPI1	SPI_SCLK	PTB14 / LPSPI1_SCK	SPI clock
		SPI_SIN	PTB15 / LPSPI1_SIN	SPI data in
		SPI_SOUT	PTB16 / LPSPI1_ SOUT	SPI data out

Table 3. Pins assignment for S32M244 PMSM sensorless FOC control ***...continued***

Connection	Peripheral	Signal	Pin	Description
		SPI_CS	PTB17 / LPSPI1_ PCS3	SPI chip select
	GPIO	AE_Fault	PTD3	Application Extension fault (active low)
S32M244 package pins	LPUART0	LPUART_In	PTC2 / lpuart_rx	Receive data from Free MASTER
		LPUART_Out	PTC3 / lpuart_tx	Send data to FreeMASTER
	TRGMUX	TRGMUX_InitTrg	PTD0 / trgmux_out1	FTM3 Init trigger visualization
		TRGMUX_AdcTrg	PTD1 / trgmux_out2	PDB/ADC trigger visualization
		TRGMUX_CoCo	PTE11 / trgmux_out5	ADC0 CH1 conversion complete visualization
	GPIO	PTE16	PTE16	Toggled pin for FOC execution time measurement
		LED	PTE15	LED for indication: steady lit – ready, slow blinking – run, fast blinking - fault
		SW0	PTB4	Application control via board button
		SW1	PTB5	
	FTM2	ENC_A	PTE5	Phase A signal of the Encoder sensor
		ENC_B	PTE4	Phase B signal of the Encoder sensor

This pins assignment can be carried out by means of the S32CT opening pins tool. Pin assignment of the FTM3 module is shown in Figure 18 as an example.

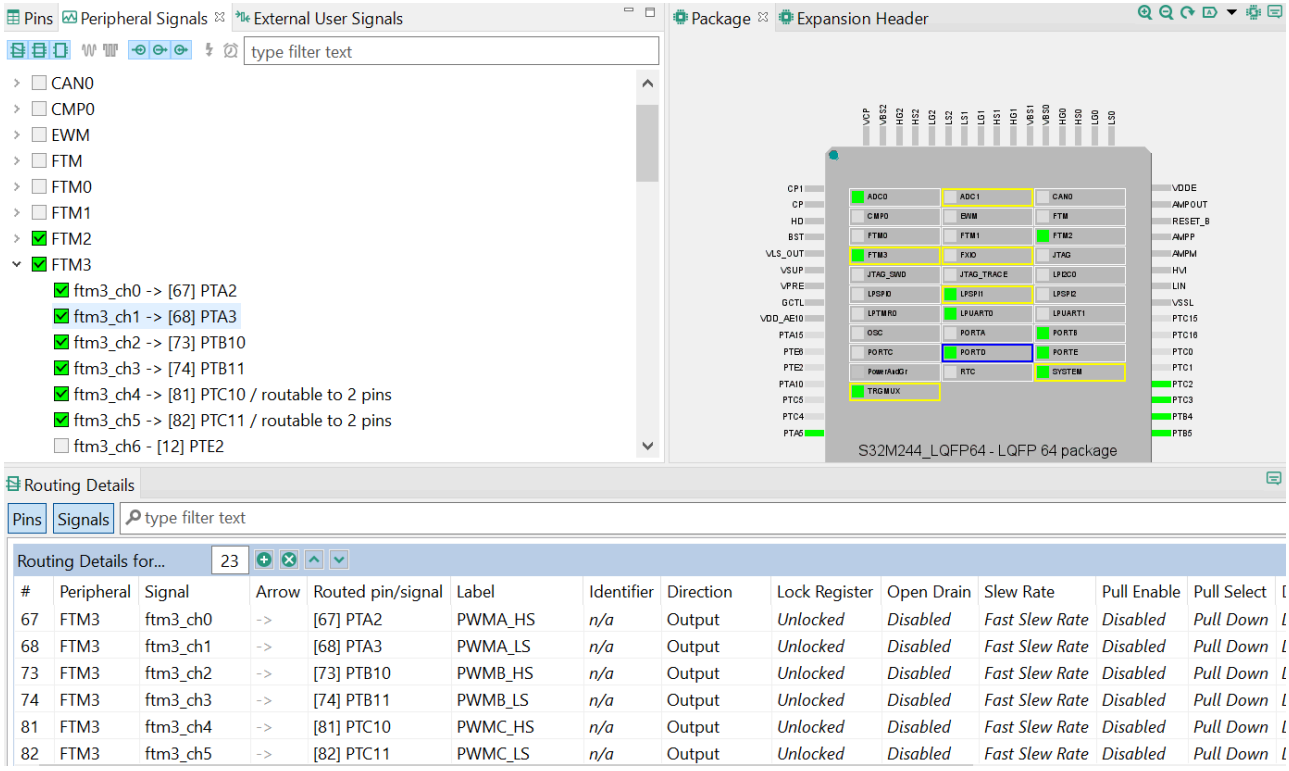

Figure 18. S32M244 pins assignment for FTM3 in S32CT

The S32CT -generated configuration structure *g_pin_mux_InitConfigArr_BOARD_InitPins* and the respective RTD function is then called in *src/peripherals/peripherals_config.c/MCUPinsConfig()*.

Note: *At the time of release of example SW version 1.0, ADC0 interleave on pin PTB13 was not available in the S32CT Pins tool, therefore the interleave is set in src/Peripherals/peripherals_config.c/MCUPinsConfig() by calling Port_Ci_Port_Ip_SetMuxModeSel(IP_PORTB, 13, PORT_MUX_ADC_INTERLEAVE).*

4.2.9 Interrupt configuration

The usage of the respective interrupts takes place in the S32CT peripherals tool. The interrupts are configured as per Table 4.

Table 4. S32M244 interrupt usage

Name	Functionality
ADC0_IRQn	reading of the ADC results, FOC calculation, calculation of PWM edges and triggers
PDB0_IRQn	handling of PDB sequence error
PORTD_IRQn	handling of AE fault
FTM3_Ovf_Reload_IRQn	loading of PWM edges into FRM3 for odd and even cycles

The configuration of the interrupts in the S32CT peripherals tool is shown in Figure 19.

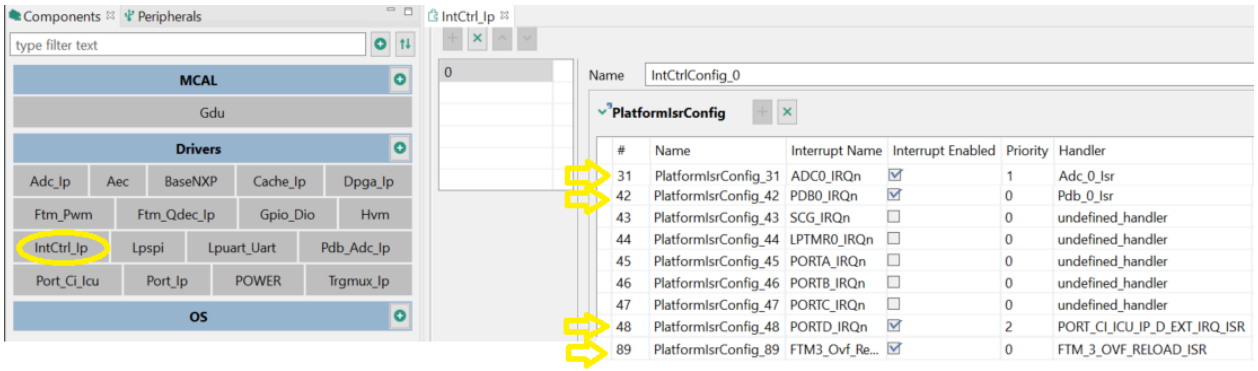

Figure 19. S32M244 interrupt configuration in RTD

The MCU can detect AE fault through falling edge on PTD3. This event can be handled by PTD3 interrupt, this needs to be set in S32CT peripherals tool as well, a part of the setting is shown in Figure 20. See the detailed setting in the S32CT peripherals tool for the module Port_Ci_Icu.

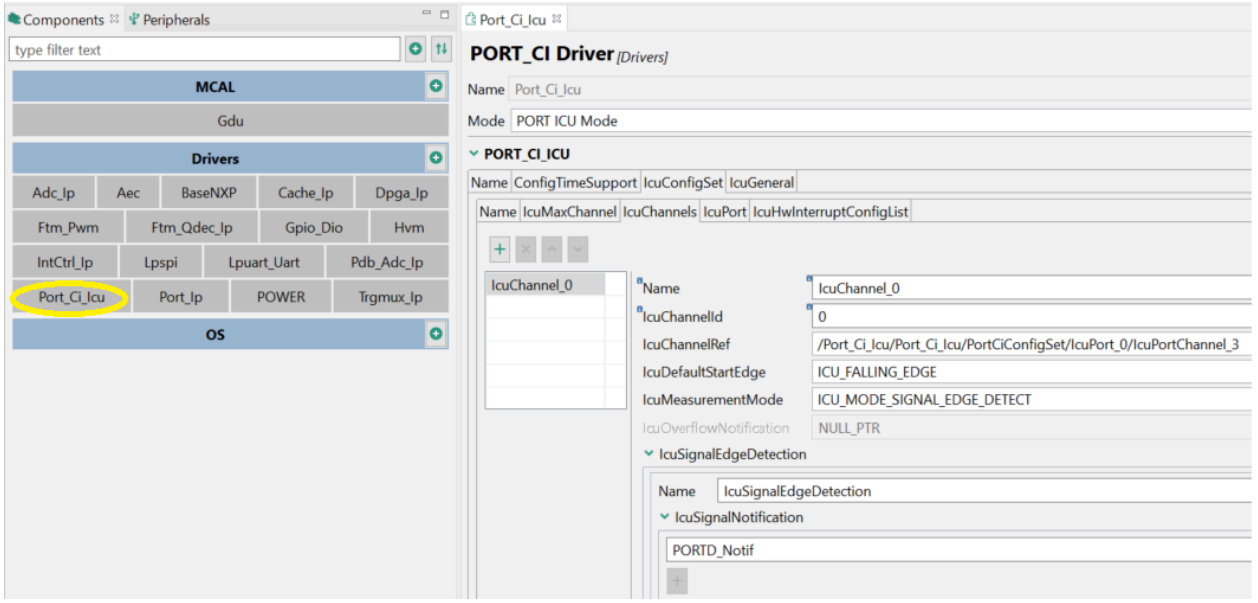

Figure 20. S32M244 enabling edge detection

4.2.10 Application Extension (AE) configuration

As discussed earlier, S32M244 comprises an MCU part and an Application Extension (AE). AE needs also to be configured and the configuration is carried out by sending SPI commands to the AE. The device-specific SPI protocol is implemented in RTD.

4.2.10.1 AE Power Management Controller (PMC) configuration

The configuration of AE PMC is done using S32CT peripherals tool as per Figure 21. For the SW example, the only relevant setting is selection of VDD voltage level. Another selection of voltage detectors and enabling of CANBUS or LINPHY supplies is also possible according to specific customer needs.

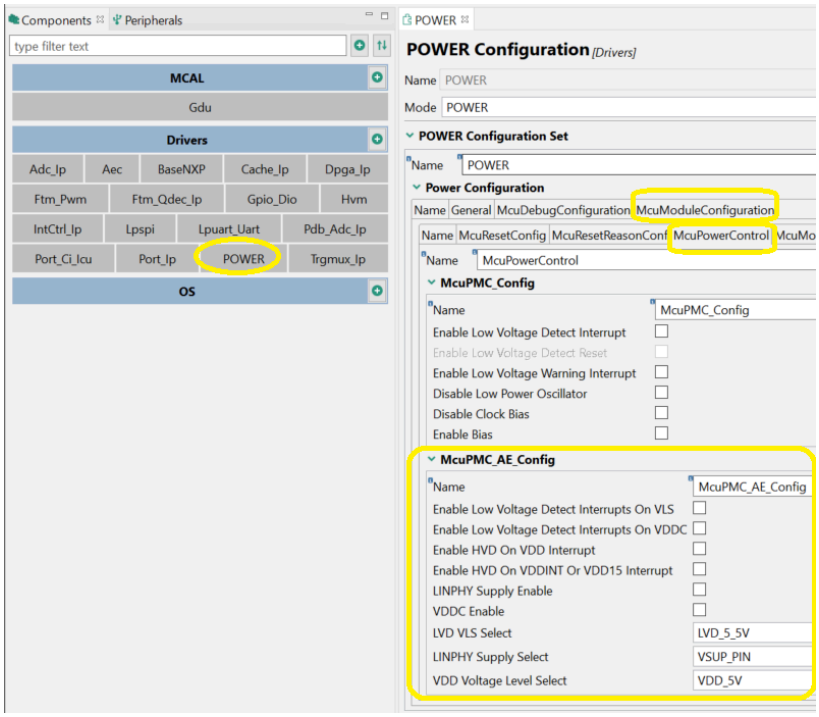

Figure 21. S32M244 PMC AE configuration in S32CT

Note: *There was a limitation in the RTD at the time of release of example SW version 1.0: to generate the AE PMC configuration structure, the respective tick box, as per Figure 22, had to be disabled, otherwise the AE PMC configuration structure was not generated.*

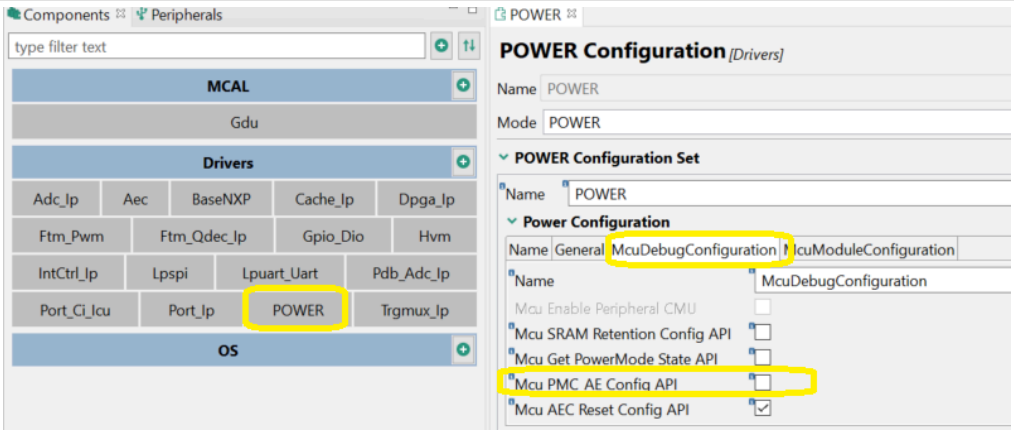

Figure 22. S32M244 PMC AE necessary setting to enable configuration structure generation

The generated configuration structure is then sent over SPI to AE. The details can be seen in *src/Peripherals/peripherals_config.c/AEC_PMCConfig()* where the user can also select whether the internal or external VPRE ballast transistor is used.

4.2.10.2 AE Reset generator configuration

AE contains several modules, which are enabled by writing to RSTGEN_CFG register of AE over SPI. The generation of RSTGEN_CFG configuration structure (and enablement of the respective AE modules) is done via S32CT according to Figure 23. for example: this application uses DPGA and GDU AE modules, therefore the two are selected in RSTGEN_CFG via *McuResetGeneratorConfiguration*.

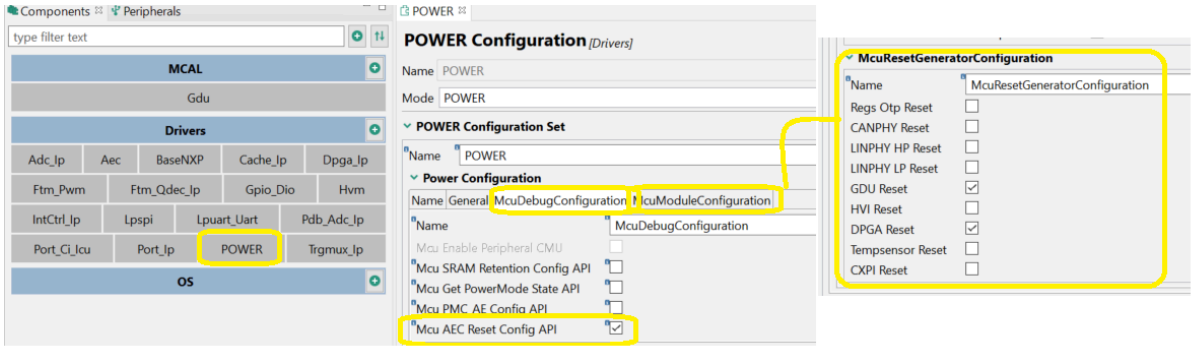

Figure 23. S32M244 AE RSTGEN_CFG configuration using S32CT

The generated configuration structure is then sent over SPI to AE. The details can be seen in *src/Peripherals/peripherals_config.c/AEC_ResetConfig()*.

4.2.10.3 AE Digital Programmable Gain Amplifier (DPGA) configuration

The DPGA takes care of the differential measurement of voltage drop across DC Bus shunt resistor, Figure 1. For detailed DPGA configuration, please refer to the respective chapter describing the DPGA in the S32M24x Reference Manual.

The SW example DPGA key settings are depicted in Figure 24.

For the SW example, the important settings are:

- Amplifier Gain, which needs to be selected considering shunt resistance, DPGA output range, nominal motor current and DPGA temperature offset drift. The maximum theoretical amplitude of the measured current can be calculated as:

$$I_{max} = \frac{\frac{V_{ref}}{2}}{R_{shunt} DPGA_{gain}} \quad (1)$$

Where V_{ref} is the ADC reference voltage (5 V or 3.3 V as per VDD settings), R_{shunt} is the shunt resistance and $DPGA_{gain}$ is the gain of the DPGA.

- Output common mode voltage, which is set to half of the DPGA range, since in motor control application both directions of DC Bus currents can be expected in certain applications. This parameter sets the "artificial zero" at DPGA output, enabling for bidirectional DC Bus current measurement.
- Input common mode coarse sets the amount of level shifting current flowing through level shifting resistors connected between the shunt and the DPGA. Please refer to the S32M24x Reference Manual for details.

The parameters for blanking time allow for setting the blanking time duration and blanking trigger events. The blanking feature helps overcome a possible saturation of DPGA inputs by disconnecting the DPGA inputs from the measured circuitry (shunt resistor) during blanking time. Please refer to the S32M24x Reference Manual for more details (see section References). The rest of the parameters serve for output and input DPGA offset. These parameters are for static DPGA offset compensation. To achieve better accuracy, a run-time DPGA offset compensation has also been introduced, see 4.3.3.4.1. Please refer to the S32M24x Reference Manual for more details on DPGA and DPGA settings (see section References).

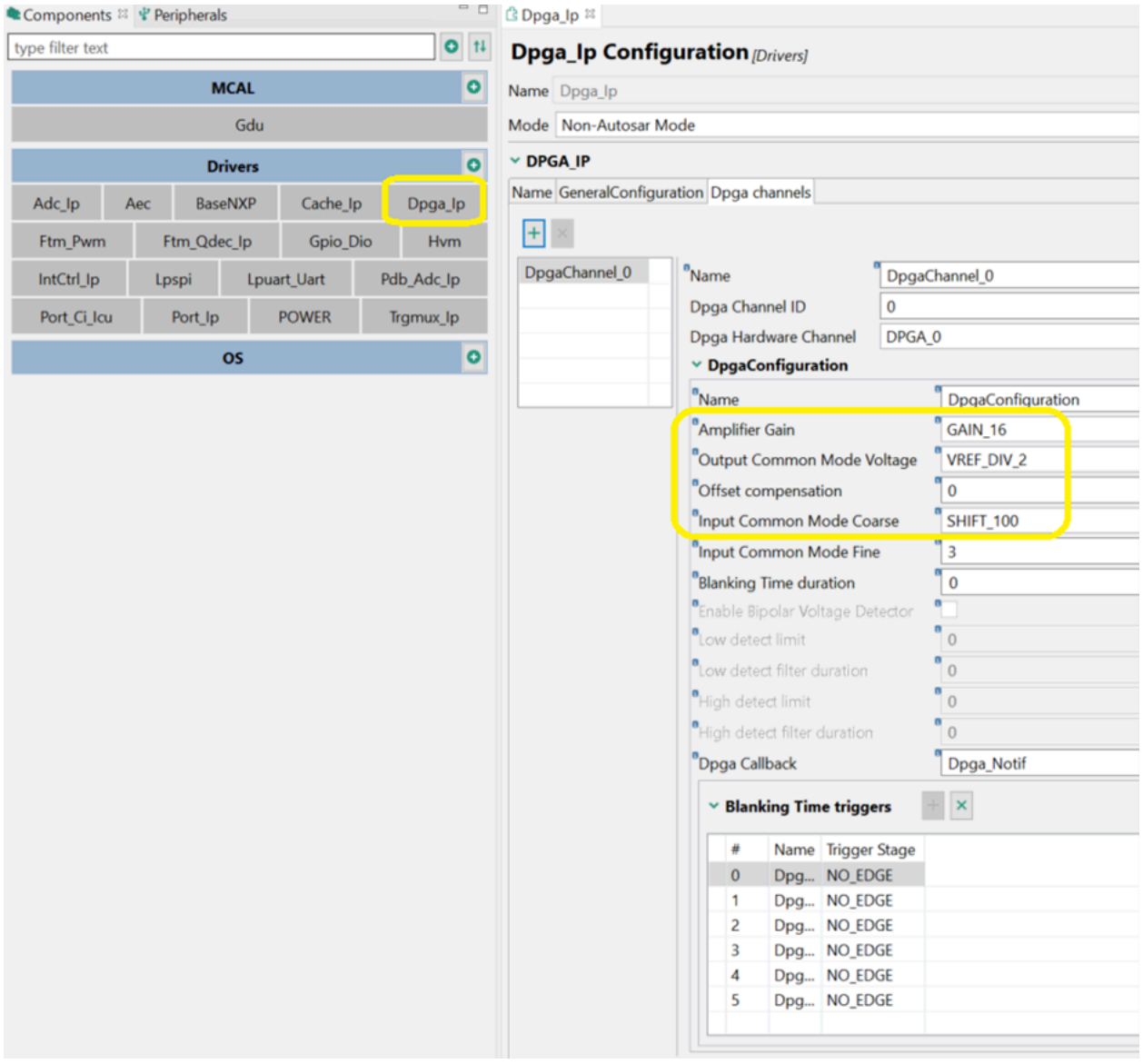

Figure 24. S32M244 AE DPGA configuration using S32CT

The S32CT-generated configuration structure is then sent over SPI to AE. The details can be seen in *src/peripherals/peripherals_config.c/AEC_DPGAConfig()*.

This function also allows the user to set the DPGA voltage detector as its setup is currently not supported in the RTD. The voltage detector is set via *AEC_DPGAConfig()* function parameters:

- *BIVDEN* - bidirectional voltage detector enable
- *VDEN* - voltage detector enable
- *HDFDUR* - filter duration for high limit
- *HDLIM* - high limit
- *LDFDUR* - filter duration for low limit
- *LDLIM* - low limit

The voltage detector monitors the voltage drops across the shunt resistor. When set, it generates a fault if the voltage drop across the shunt resistor is outside the limits set by the respective parameters. Please refer to the S32M24x Reference Manual for more details on voltage detector and voltage detector limits calculation.

4.2.10.4 AE Gate Driver Unit (GDU) configuration

The AE comprises an advanced GDU, which offers, for example: advanced slew rate control and desaturation protection.

The GDU settings are displayed in Figure 25. The SW example uses the desaturation protection and slew rate settings out of the features offered by the GDU. The other possibilities such as overvoltage detection or boost may be enabled here according to customer specific needs. The boost feature allows operating the MOSFETs also in case the HD voltage is below 7 V. See S32M2xx data sheet and S32M24x Reference Manual for details (see section References). Please note that the boost feature does not boost the voltage for the connected motor. It boosts the voltage for S32M2xx (including the GDU) on VSUP pin so the MOSFETs can be controlled safely.

The parameter HD and High-side divider sets the division ratio of the voltage dividers on HD and HS pins, please refer to S32M2xx data sheet for the division ratio details (see section References).

It is important to set all the slew rates (only switch-on slew rates for the high side are shown in the picture) according to customer specific hardware and slew rates demands. This needs to be done for switching on and off and for high and low side drivers. Please refer to the S32M24x Reference Manual and S32M2xx data sheet for slew rate setting scheme details.

The slew rate settings principle is displayed in Figure 26. The switching on (left) and switching off (right) process can be divided into three intervals and for each of the intervals the MOSFET gate current (i.e., slew) can be adjusted. Note that Figure 26 is for illustration only and the response of an actual MOSFET gate voltage (V_{GS}) to the gate current (I_G) depends on the actual MOSFET type and inverter hardware design.

There are separate slew rate settings parameters for switching on and off process for high and low side. The summary of the parameters is listed in Table 5. The user can set the currents injected into the MOSFET gate for all three intervals and the length of interval 1 and 2 is adjustable as well.

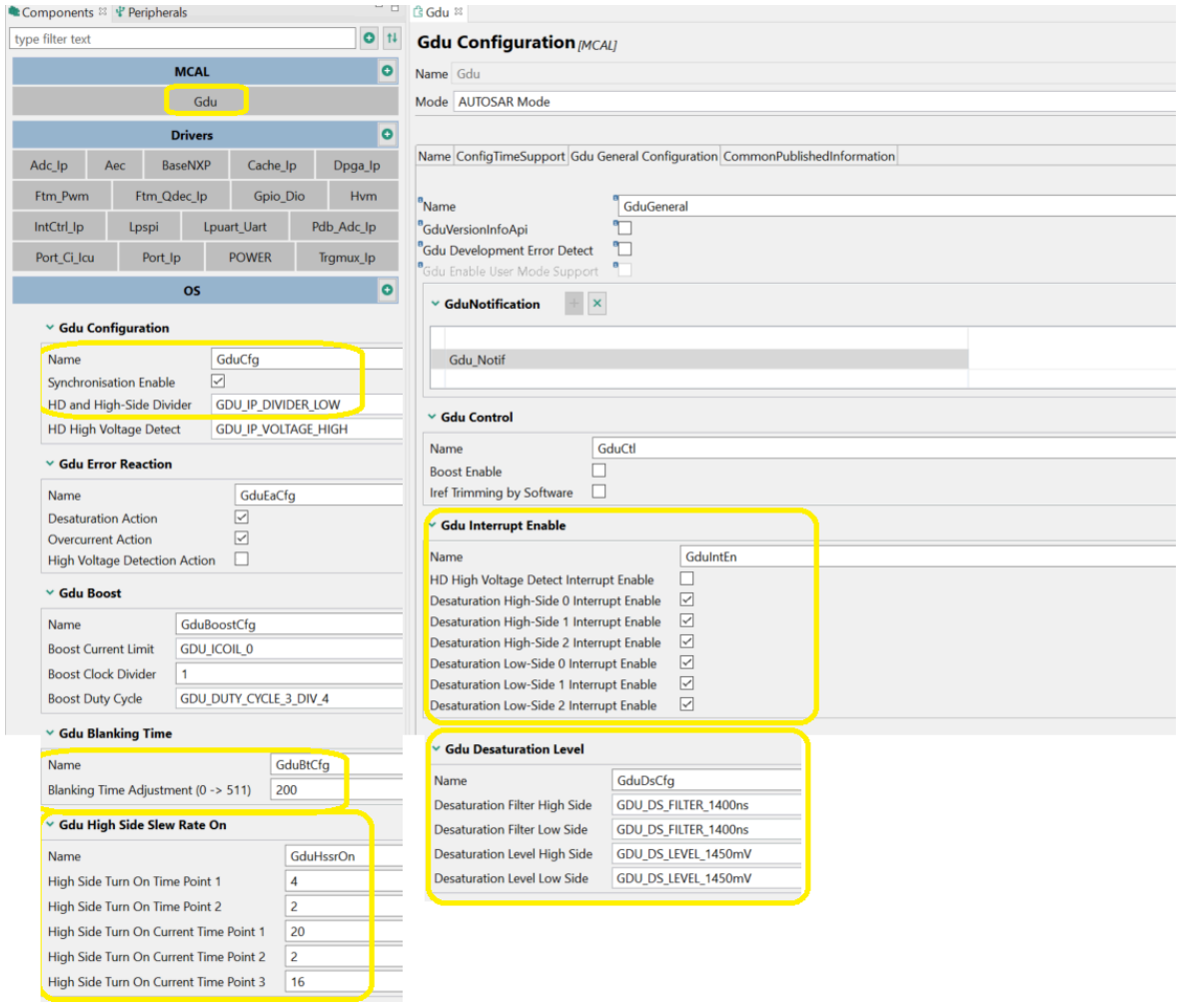

Figure 25. S32M244 AE GDU configuration using S32CT

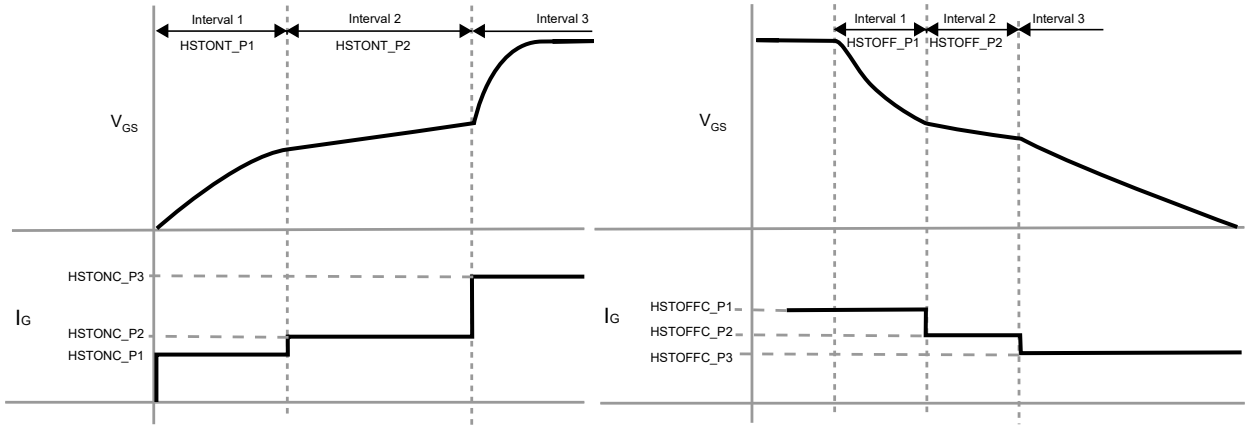

Figure 26. S32M244 GDU slew rate settings principle

Table 5. GDU slew rate settings

	Process	1st interval		2nd interval		3rd interval
		duration	gate current	duration	gate current	gate current
High side	Switch on	HSTONT_P1	HSTONC_P1	HSTONT_P2	HSTONC_P2	HSTONC_P3
	Switch off	HSTOFFT_P1	HSTOFFC_P1	HSTOFFT_P2	HSTOFFC_P2	HSTOFFC_P3
Low side	Switch on	LSTONT_P1	LSTONC_P1	LSTONT_P2	LSTONC_P2	LSTONC_P3
	Switch off	LSTOFFT_P1	LSTOFFC_P1	LSTOFFT_P2	LSTOFFC_P2	LSTOFFC_P3

The duration of the intervals is set in AE clock ticks, while the AE clock runs at 42 MHz.

By being able to adjust the gate current in three stages and to control the length of the stages, the user is able to shape the slew rates and MOSFET response according to application needs with regard to switching losses, switching performance and EMC. The resolution of gate current adjustment (step size) differs for lower, mid, and higher level gate currents:

- fine step size – for lower-level gate currents.
- more coarse step size – for mid-level gate currents.
- coarse step size – for higher-level gate currents.

The step size is described in the S32M2xx data sheet. For more details on the slew rate parameters and how the parameter settings of gate current correspond to actual gate current values, please see S32M24x Reference Manual and S32M2xx data sheet (References).

The other important parameters are GDU Desaturation levels and blanking time of desaturation protection and GDU blanking time adjustment (see Figure 25) which are also hardware specific and need to be set according to customer specific project needs.

The S32CT-generated configuration structure is then sent over SPI to AE. The details can be seen in *src/peripherals/peripherals_config.c/AEC_GDUConfig()*.

4.2.10.5 Other functions for AE control used in SW example

Various other functions have been prepared for the SW example and can be found under *src/Peripherals/peripherals_config.c/*. These are mainly for reset of fault and event flags in the respective AE modules, which are to be cleared as Write 1 To Clear (W1C). The W1C functions are:

- *AEC_PMCResetW1C()* for clearing AE PMC W1C faults and events
- *AEC_DPGAResetW1C()* for clearing AE DPGA W1C faults and events
- *AEC_GDUResetW1C()* for clearing AE GDU W1C faults and events
- *AEC_HVMResetW1C()* for clearing AE HVM (High Voltage Module) W1C faults and events
- *AEC_AEResetW1C()* for clearing AE controller W1C faults and events

Apart from these fault and event clearing functions also functions *AEC_VDDE_Enable()* and *AEC_VDDE_Disable()* have been prepared for enabling/disabling the VDDE supply for external sensors.

4.3 Software architecture

4.3.1 Introduction

This section describes the software design of the sensorless PMSM field oriented control framework application. The application overview and description of software implementation are provided. The aim of this chapter is to help in understanding of the designed software.

4.3.2 Application data flow overview

The application software is interrupt driven running in real time. There is one periodic interrupt service routine associated with the ADC conversion complete interrupt, executing all motor control tasks. This includes both fast current and slow speed loop control. All tasks are performed in an order described by the application state machine shown in Figure 29, and application flowcharts shown in Figure 27 and Figure 28.

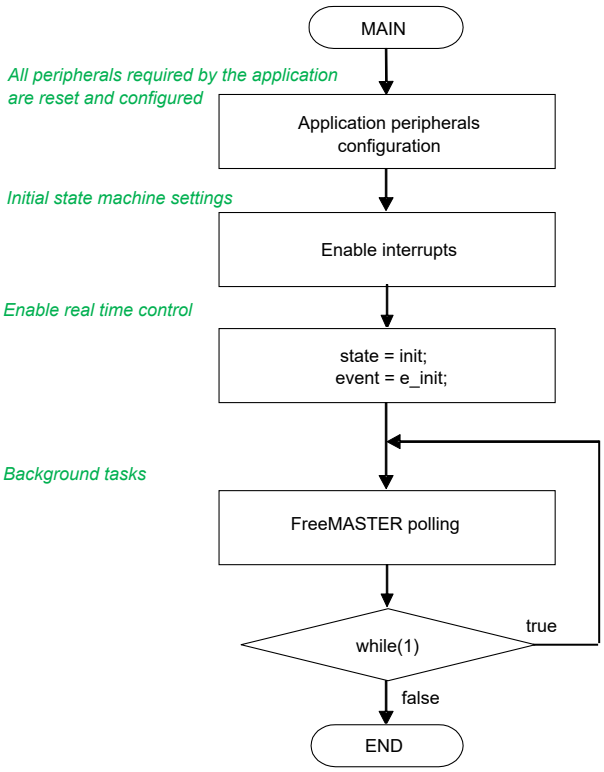

Figure 27. Flowchart diagram of main function with background loop.

To achieve precise and deterministic sampling of analog quantities and to execute all necessary motor control calculations, the state machine functions are called within a periodic interrupt service routine. Therefore, to actually call state machine functions, the peripheral causing this periodic interrupt must be properly configured and the interrupt enabled. As described in this section 4.2, all peripherals are initially configured and all interrupts are enabled after a reset of the device. When interrupts are enabled and all S32M244 peripherals are correctly configured, the state machine functions are called from the ADC interrupt service routine (ISR) which is pictured in Figure 28. The background loop handles non-critical timing tasks, such as the FreeMASTER communication polling, this is illustrated in Figure 27.

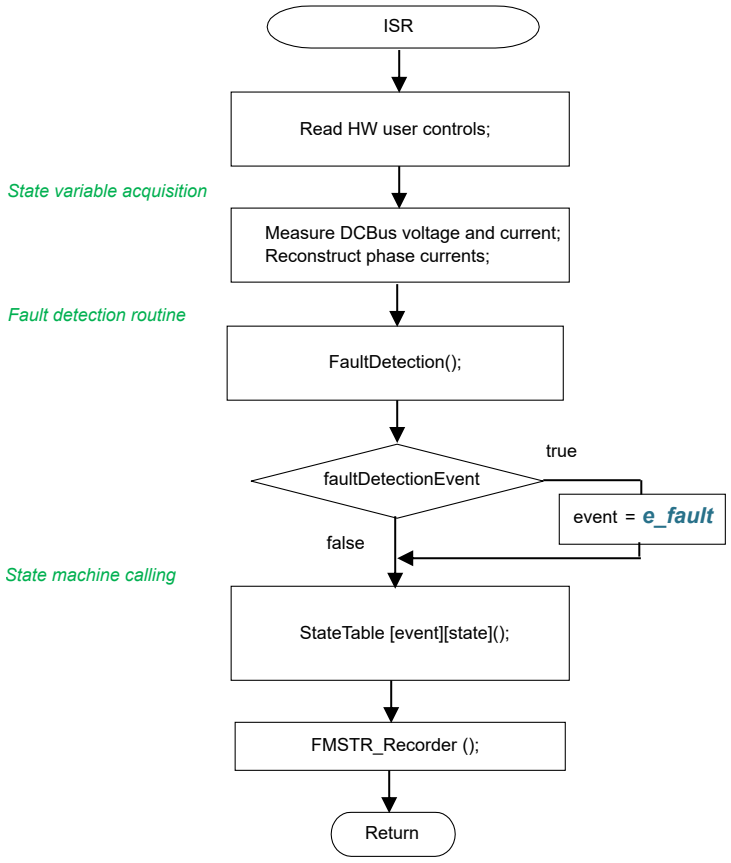

Figure 28. Flowchart diagram of periodic interrupt service routine.

4.3.3 State machine

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

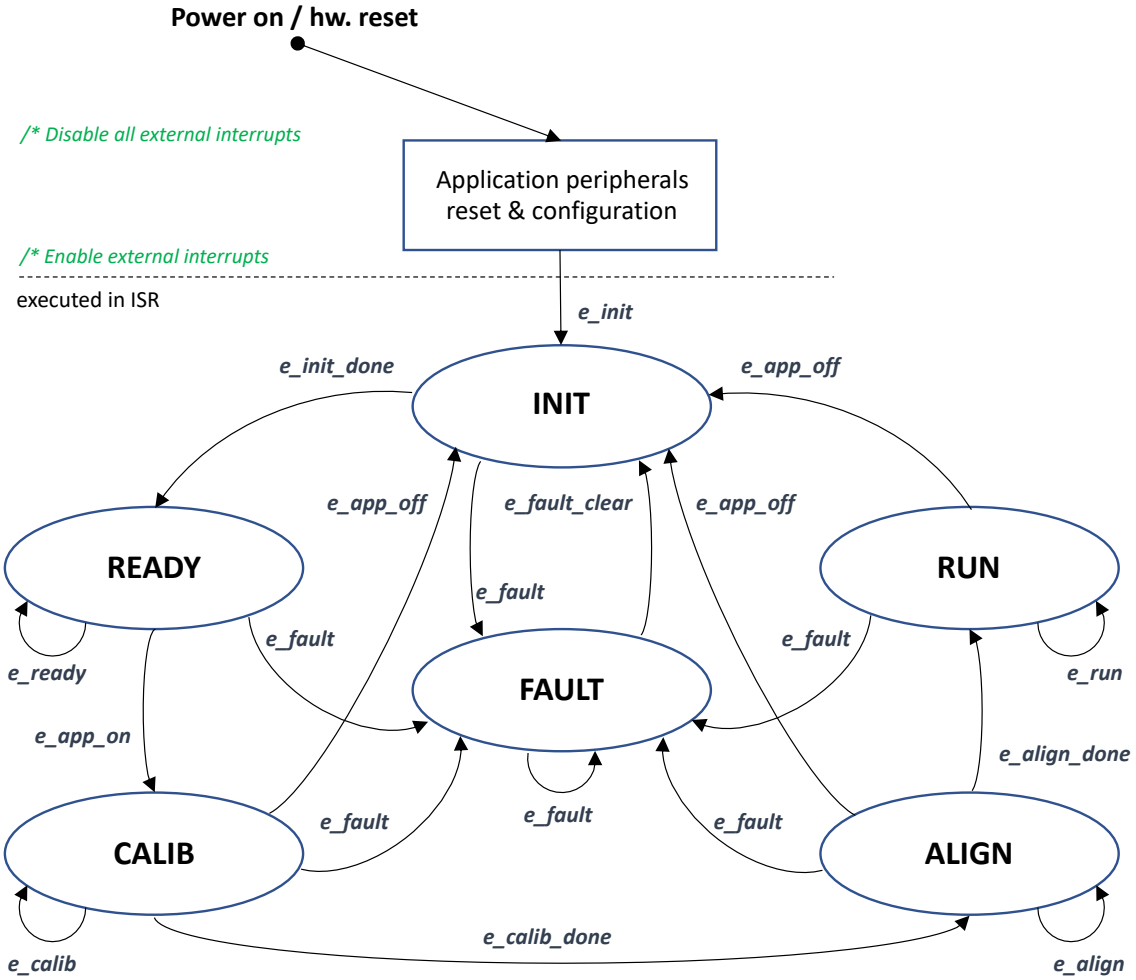

Figure 29. Application state machine

The application state machine consists of the following six states, which are selected using the 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 the 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_app_off - event = 11
- e_calib - event = 6
- e_calib_done - event = 7
- e_align - event = 8
- e_align_done - event = 9
- e_run - event = 10

4.3.3.1 State – FAULT

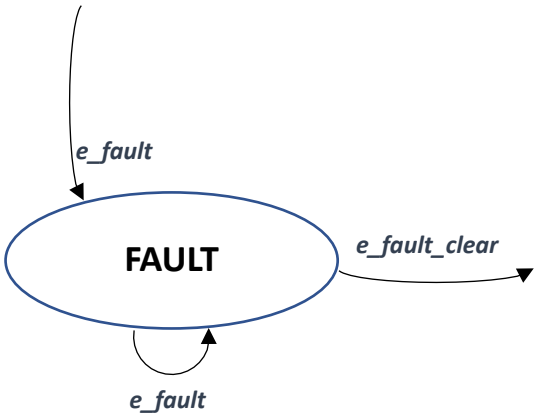

Figure 30. FAULT state with transitions

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 the 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 (SW0 and SW1) on the S32M24xEVB evaluation board. 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*; a fault-clearing sequence is executed.

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 28, function *FaultDetection()* is called before state machine function state_table[event][state] (). Therefore, all faults will be checked again and if there is any fault condition remaining in the system, the respective bits in *permFaults* and *tempFaults* variables will be set. As a consequence of permFaults not equal to zero, the function *FaultDetection()* will modify the application event from *e_fault_clear* back to *e_fault*, which means jump to fall state when state machine function *state_table[event][state]()* is called. Therefore, 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 the application variable *switchFaultClear* is manually set to true.

The application is scanning for the following system warnings and errors:

- DC Bus over voltage

- DC Bus under voltage
- Phase over current

The thresholds for fault detection can be modified in the INIT state. Please see References for further information on how to set these thresholds using the MCAT. In addition, a fault state is entered if following errors are detected:

- PDB errors (PDB Sequence error)
- Application Extension errors (as per the events and faults enabled by the user for Application Extension)
- FOC error (irrelevant event call in state machine or Back-EMF failure)

4.3.3.2 State – INIT

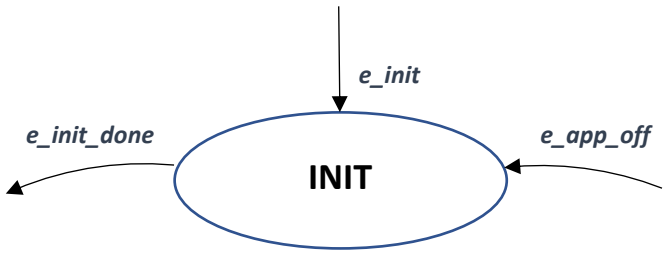

Figure 31. INIT state with transitions

State INIT is a “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.

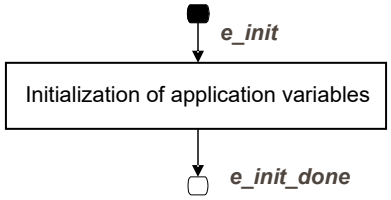

Figure 32. Flowchart of state INIT

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

4.3.3.3 State – READY

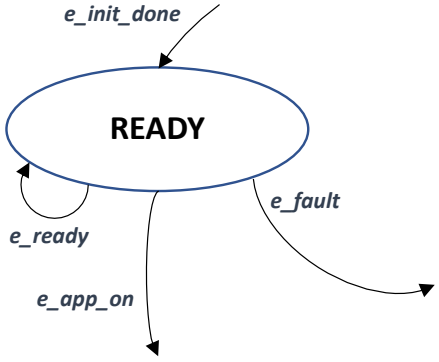

Figure 33. READY state with transitions

In the READY state, the application is waiting for a user command to start the motor. The application is released from waiting mode by pressing the on board button SW0 or SW1 or by FreeMASTER interface setting the

variable *switchAppOnOff = true* (see flowchart in Figure 34). DC Bus voltage and currents are monitored during READY state for fault detection purposes.

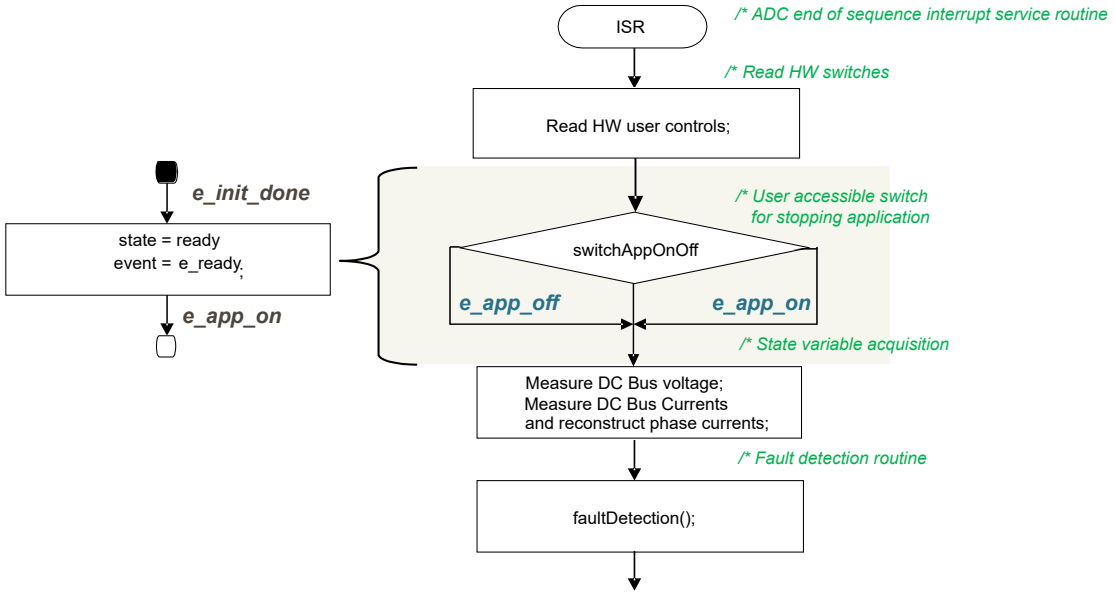

Figure 34. Flowchart of state READY

4.3.3.4 State – CALIB

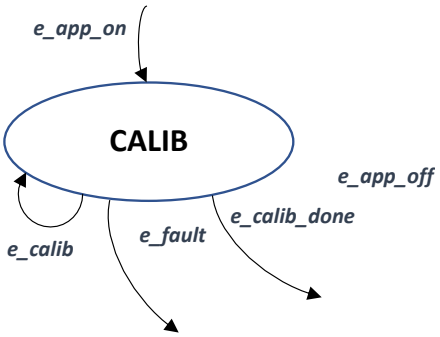

Figure 35. CALIB state with transitions

In this state, ADC DC offset calibration is performed. Once the state machine enters CALIB state, all PWM outputs are enabled. Calibration of the DC offset is achieved by generating 50% duty-cycle on the PWM outputs and taking several measurements of the ADC0 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 when in normal operation. This way the half range DC offset, caused by voltage shift of 2.5 V (according to DPGA settings), is removed in the measured phase. 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 default to 2^10 = 1024, and can be modified in the state INIT. After all 1024 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 flowchart in Figure 36).

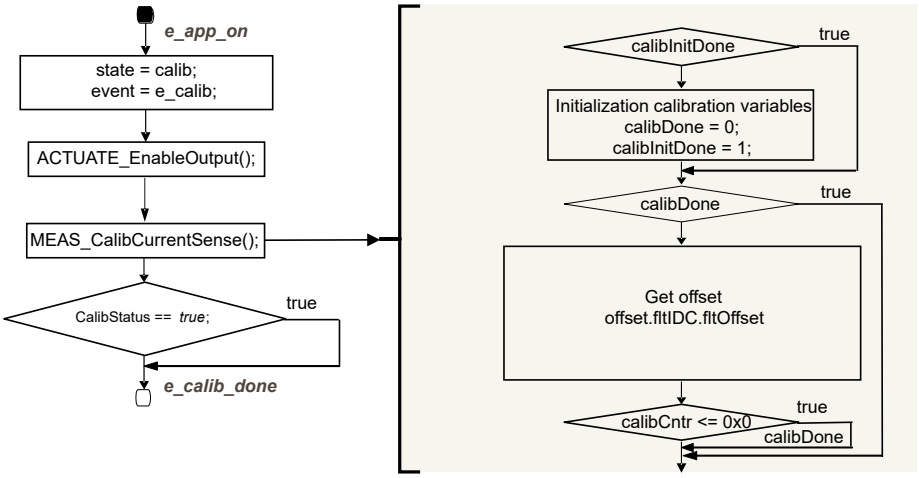

Figure 36. Flowchart of state CALIB

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 the falling edge of switchAppOnOff=false using FreeMASTER.

4.3.3.4.1 Run-time DPGA offset compensation

For better accuracy, also an alternative approach of current measurement offset compensation has been implemented. DC Bus current is measured in zero vector (as depicted in Figure 7) in the beginning of the PWM cycle.

The DC Bus current in zero vector is expected to be zero. Since a DC offset has been introduced to measure bidirectional DC Bus current, a non-zero current (or an "artificial zero") is expected during zero vector. The measurement of the DPGA offset is done during the calibration sequence and during motor run-time. This allows for DPGA offset temperature drift compensation while the motor is spinning. The run-time DPGA offset compensation can be enabled by setting the macro *DPGA_OFFSET_COMP*. Otherwise, the conventional "static" DPGA offset calibration (with offset measured only during CALIB state) is used.

To avoid PDB errors or measurement of transients in case the zero vector is too short, the length of the zero vector is continuously observed and the DPGA offset in the zero vector is only measured when the zero vector is sufficiently long. The zero vector minimal length (in PDB clock ticks) is set by the value of the variable *minZeroVectorDPGAOffset*.

4.3.3.5 State – ALIGN

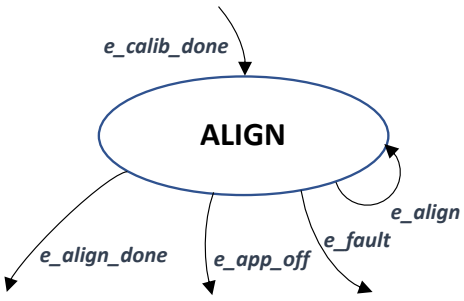

Figure 37. ALIGN state with transitions

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 two state alignment is used to avoid sticking at 180 deg. A DC voltage is applied to a q-axis voltage for a certain period and after that to d-axis voltage for the rest of the alignment time. The ratio between d and q axis alignment time is given by macro ALIGN_D_FACTOR. This causes 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 the rotor stabilized at aligned position, a certain time is selected for the alignment process. This time and the amplitude of the DC voltage used for alignment can be modified by an MCAT tool. Timing is implemented using a software counter that counts from a pre-defined value down to zero. During this time, the event remains set to *cntrState.event=e_align*. When the counter reaches zero, the counter is reset back to the pre-defined value, and the event is automatically set to *cntrState.event=e_align_done*. This enables a transition to RUN state see the flow chart in Figure 38.

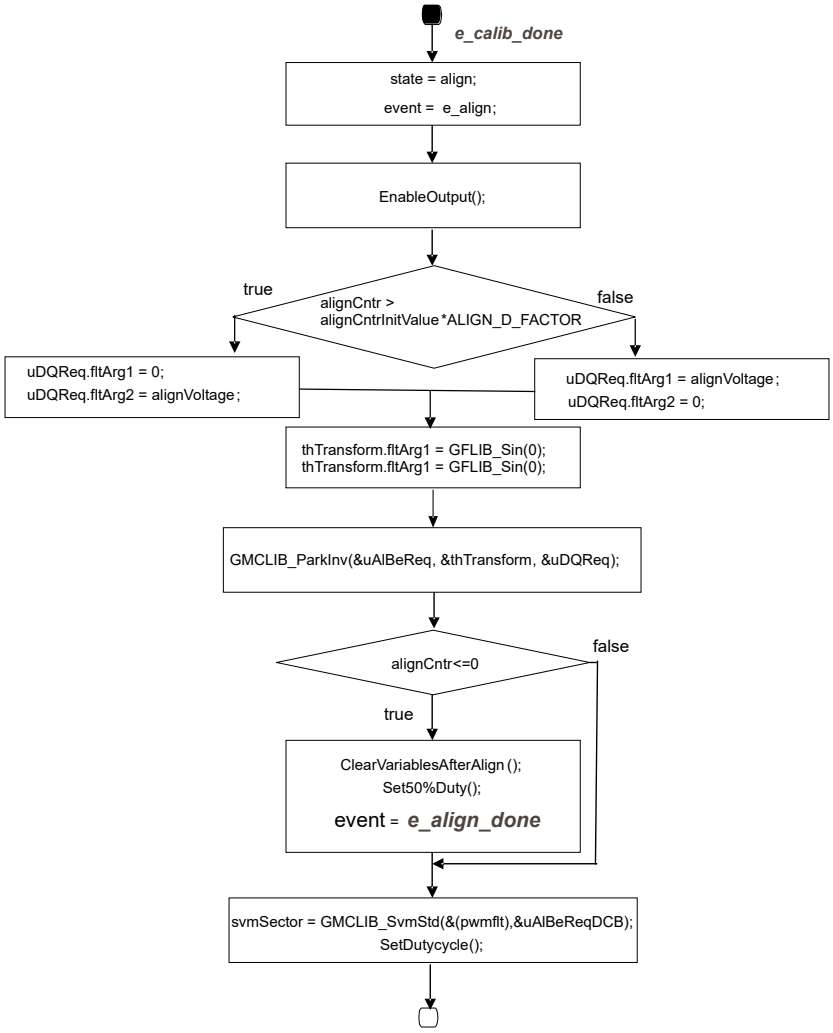

Figure 38. Flowchart of state ALIGN

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 the falling edge of *switchAppOnOff=false* using FreeMASTER or using the user buttons SW0 and SW1.

4.3.3.6 State – RUN

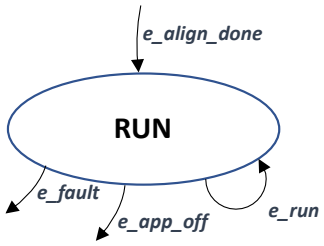

Figure 39. RUN state with transitions

In this state, the FOC algorithm is calculated, as described in NXP application note AN12235: *3-phase Sensorless PMSM Motor Control Kit with S32K144* (see section References).

The control is designed such that the drive might be operated in four 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 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: Both FOC control and eBEMF observer using estimated position.
4. Encoder mode: FOC control uses position and speed obtained from an Encoder sensor. This mode is available only if *ENCODER* macro is set to *true*.

Position mode can be controlled by the *pos_mode* variable in the 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*, the 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 3.3). Variable *switchSensor* defines whether position/speed feedback comes from a back-EMF Observer or Encoder sensor. It is automatically set to *Sensorless* if the Encoder sensor is not present (*ENCODER=false*).

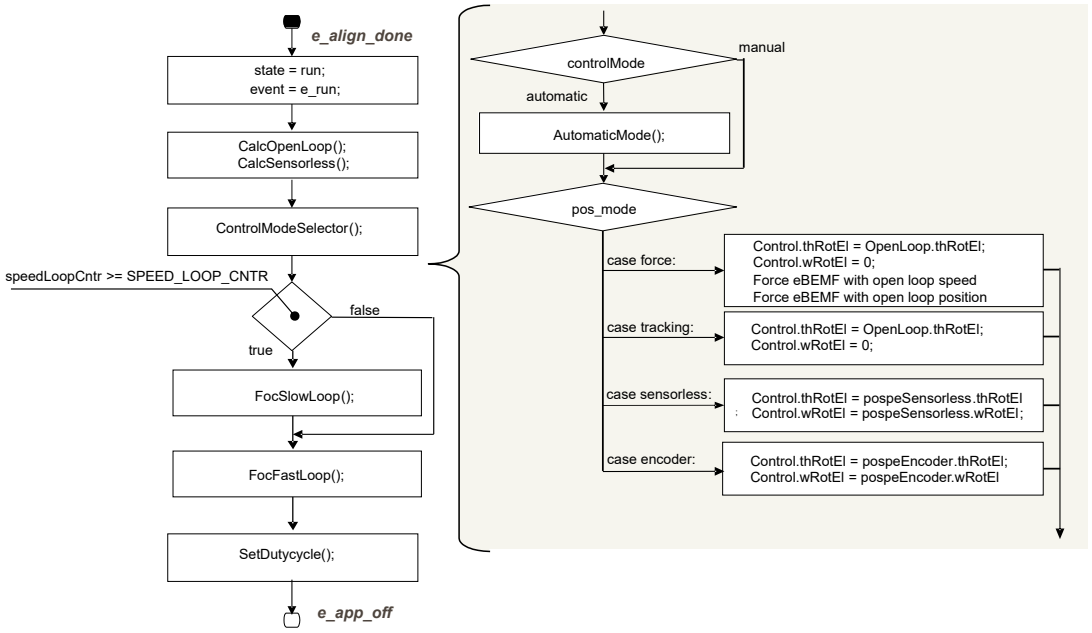

Figure 40. Flowchart of state RUN

Calculation of fast current loop is executed at every ADC end of sequence interrupt, while calculation of slow speed loop is executed every Nth ADC end of sequence interrupt. Arbitration is done using a counter that counts from the value N down to zero. When zero is reached, the counter is reset back to N and a slow speed loop calculation is performed. This way, only one interrupt is needed for both loops and the timing of both loops is synchronized. Slow loop calculations are finished before entering fast loop calculations (see flowchart in Figure 39). The value of N depends on the ratio of speed and current loop sample time, which are set in MCAT, please see the section 5.

Figure 40 shows implementation of FOC algorithm and 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 test encoder based FOC, *ENCODER* macro must be set to *true* and PM motor provided with this motor control kit replaced by PM motor of the comparable power and equipped with Encoder sensor. As mentioned previously, encoder based FOC can be activated/deactivated by setting *switchSensor* variable to *encoder/sensorless*.

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 keeping user buttons SW0 and SW1 pressed.

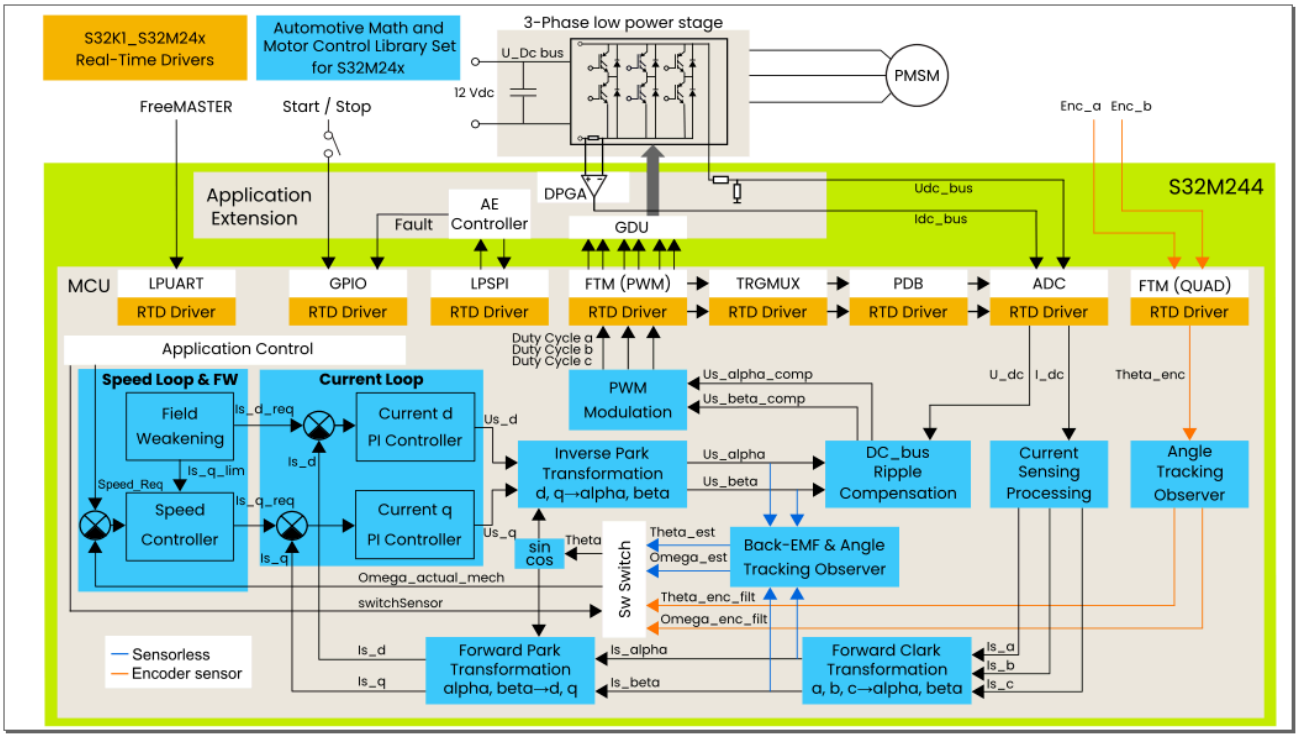

Figure 41. Sensorless and sensor based FOC with FW implementation on S32M244

4.3.4 AMMCLib Integration

The integration of AMMCLib is done in the manner identical to the one described in NXP application note AN12235: *3-phase sensorless PMSM Motor Control Kit with S32K144* (see References).

5 FreeMASTER and MCAT user interface

The FreeMASTER debugging tool is used to control the application and monitor variables during runtime . FreeMASTER and MCAT interface enables online application tuning and control.

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 the proposed approach is automatic calculation and real-time tuning of selected control structure parameters. Connecting and tuning a 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 the cascade control structure.

FreeMASTER and MCAT user interface are described in application note AN12235: *3-phase Sensorless PMSM Motor Control Kit with S32K144* (see References).

For an in-depth description of motor control application tuning using MCAT, please see NXP application note AN4642: *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (see References).

6 Conclusion

Design, described in this application note shows the simplicity and efficiency in using the S32M244 microcontroller for sensorless PMSM motor control and introduces it as an appropriate candidate for various low-cost applications in the automotive area. MCAT tool provides an interactive online tool, which makes the

PMSM drive application tuning friendly and intuitive. For other motor control use cases of S32M2, please see NXP community page (References).

7 References

- S32 Design Studio IDE for S32 Platform
- Real-Time Drivers (RTD)
- FreeMASTER Run-Time Debugging Tool
- Automotive Math and Motor Control Library
- S32M24x Reference Manual
- S32M2xx Data Sheet
- S32M24x PMSM/BLDC Motor Control Evaluation Boards
- BLDC PMSM low voltage motor control accessory kit
- 3-Phase Sensorless PMSM Motor Control Kit with S32K144
- Current Sensing Techniques in Motor Control Applications
- Rashid, M. H. Power Electronics Handbook, 2nd Edition. Academic Press
- Motor Control Application Tuning (MCAT) Tool
- Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
- NXP community - S32M2xx motor control use cases
- PMSM Electrical Parameters Measurement

8 Revision history

Table 6. Revision history

Document ID	Release date	Description
AN14454 v.1.0	19 October 2014	Initial release

9 Note about the source code in the document

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

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

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

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

ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Legal information

Definitions

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

Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Trademarks

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

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

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Contents

1 Introduction 2
2 System concept 2
3 PMSM field oriented control 3
3.1 Fundamental principle of PMSM FOC 3
3.2 Output voltage actuation and phase current measurement 3
3.2.1 Double switching configuration 6
3.3 Rotor position/speed estimation 7
3.4 Field weakening 7
4 Software implementation on the S32M244 7
4.1 S32M244 – key modules for PMSM FOC control 7
4.1.1 Module interconnection 7
4.1.2 Module involvement in digital PMSM sensorless control loop 9
4.2 S32M244 device initialization 11
4.2.1 Clock configuration and power management 11
4.2.2 FlexTimer Module (FTM) 13
4.2.2.1 Edge-aligned PWM mode 13
4.2.2.2 Quadrature decoder mode 14
4.2.3 Trigger MUX control (TRGMUX) 16
4.2.4 Programmable Delay Block (PDB) 16
4.2.5 Analog-to-Digital Converter (ADC) 18
4.2.6 Low Power Serial Peripheral Interface (LPSPI) 19
4.2.7 Low Power Universal Asynchronous Receiver/Transmitter (LPUART) 20
4.2.8 Port control and pin multiplexing 21
4.2.9 Interrupt configuration 23
4.2.10 Application Extension (AE) configuration 24
4.2.10.1 AE Power Management Controller (PMC) configuration 24
4.2.10.2 AE Reset generator configuration 25
4.2.10.3 AE Digital Programmable Gain Amplifier (DPGA) configuration 26
4.2.10.4 AE Gate Driver Unit (GDU) configuration 28
4.2.10.5 Other functions for AE control used in SW example 30
4.3 Software architecture 30
4.3.1 Introduction 30
4.3.2 Application data flow overview 31
4.3.3 State machine 32
4.3.3.1 State – FAULT 34
4.3.3.2 State – INIT 35
4.3.3.3 State – READY 35
4.3.3.4 State – CALIB 36
4.3.3.5 State – ALIGN 37
4.3.3.6 State – RUN 39
4.3.4 AMMCLib Integration 41
5 FreeMASTER and MCAT user interface 41
6 Conclusion 41
7 References 42
8 Revision history 42
9 Note about the source code in the document 42
Legal information 44