AN14504

EV Traction Inverter Control Reference Design Gen 3 SW application note

Rev. 2.0 — 27 October 2026 **Application note**

Document information

Information	Content
Keywords	Traction Inverter Reference Design, Dynamic Gate Strength, Software, System FuSa, Efficiency, S32K39 MCU, FS26 SBC, GD3162 HV Gate Driver
Abstract	The purpose of the software application note document is to provide detailed information related to software for the EV Traction Inverter Reference Design Gen 3.

IMPORTANT NOTICE

For engineering development or evaluation purposes only



NXP provides this evaluation product under the following conditions:

Evaluation kits or reference designs are intended solely for technically qualified professionals, specifically for use in research and development environments to facilitate evaluation purposes. This evaluation kit or reference design is not a finished product, nor is it intended to be a part of a finished product. Any software or software tools provided with an evaluation product are subject to the applicable terms that accompany such software or software tool.

The evaluation kit or reference design is provided as a sample IC pre-soldered to a printed circuit board to make it easier to access inputs, outputs, and supply terminals. This evaluation kit or reference design may be used with any development system or other source of I/O signals by connecting it to the host MCU or computer board via off-the-shelf cables. Final device in an application will be heavily dependent on proper printed circuit board layout and heat sinking design as well as attention to supply filtering, transient suppression, and I/O signal quality. This evaluation kit or reference design provided may not be complete in terms of required design, marketing, and or manufacturing related protective considerations, including product safety measures typically found in the end device incorporating the evaluation product. Due to the open construction of the evaluation product, it is the responsibility of the user to take all appropriate precautions for electric discharge. To minimize risks associated with the customers' applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards. For any safety concerns, contact NXP sales and technical support services.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

1 Introduction

The purpose of the SW application note document is to provide detailed information related to SW for the EV Traction Inverter Control Reference Design Gen 3. It outlines the configuration of a 3-phase permanent magnet synchronous motor (PMSM) vector control drive equipped with three Hall effect current sensors and resolver position detection. The design is targeted for automotive motor control applications.

This design serves as an example of motor control SW using the NXP family of automotive motor control S32K39 MCUs based on Arm Cortex-M7 technology.

The following are the supported features:

- 3-phase PMSM speed and current field oriented control (FOC)
- Current sensing with Hall effect sensors
- Integration of a user interface for application control using the FreeMASTER debugging tool
- Motor control application tuning (MCAT) tool
- Rotor position and speed measurement using a resolver transducer

2 System concept

The system is designed to drive a 3-phase PMSM, meeting specific performance criteria as outlined below:

- Uses the S32 Design Studio ref.[1]
- Incorporates the FS26 safety SBC, designed to support the S32K3 series
- Incorporates GD3162 isolated HV gate drivers, offering a comprehensive set of functions and condition monitoring capabilities
- Control methodology includes:
  - FOC for 3-phase PMSM with resolver position sensor
  - Closed-loop speed control with an adjustable sampling period
  - Closed-loop current control with an adjustable sampling period
  - Bidirectional rotation support
  - Independent control of flux and torque
  - Position and speed computations performed by the enhanced time processing unit (eTPU)
  - Sensing of 3-phase motor currents
  - Sampling of FOC state variables over a 100 µs period
  - Dynamic gate strength control
  - DC link discharge control
- Utilization of the Automotive Math and Motor Control Library (AMMCLib), featuring FOC algorithms built on precompiled SW library blocks ref.[2]
- Integration of the eTPU motor control function set to offload CPU
- Incorporates the FreeMASTER software for control interface (motor parameters, current loop, speed loop)
- Features a graphical control page MCAT within FreeMASTER software displaying required speed, actual motor speed, start/stop status, DC bus voltage level, motor current, and system status
- Provides scope functionality within FreeMASTER software for observing various motor control variables
- Offers a high-speed recorder with FreeMASTER software for capturing reconstructed motor currents and vector control algorithm quantities
- Implements protection mechanisms for DC bus overvoltage and undervoltage, overcurrent, and startup failure

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

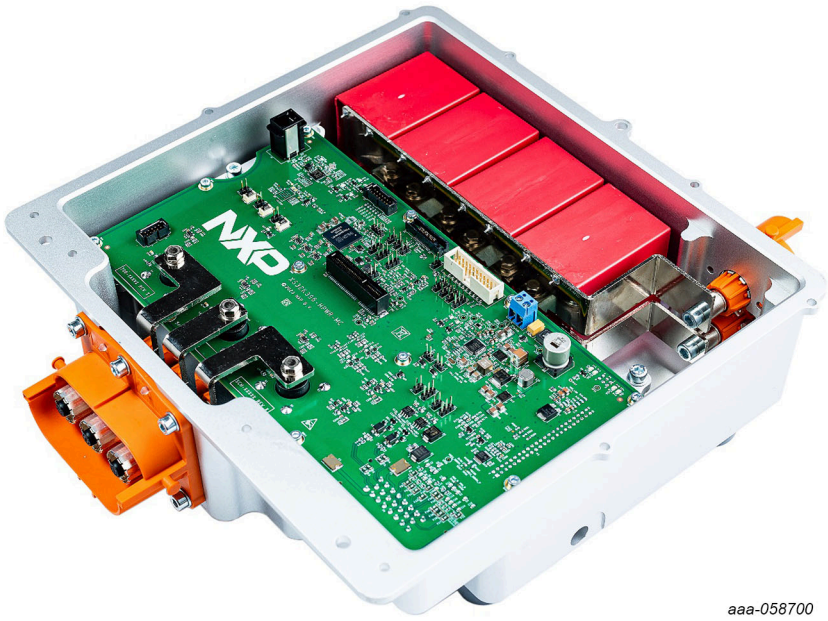


Figure 1. EV Traction Inverter Control Reference Design Gen 3

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, maintaining full torque control even at zero speed, and rapid acceleration/deceleration. This level of control is realized through the utilization of FOC for PMSM.

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 featuring surface-mounted permanent magnets develop maximum torque when the armature current vector aligns perpendicularly to the flux linkage vector. Therefore, when only considering the fundamental harmonic of stator magnetomotive force, the torque T_e developed by an AC machine, expressed in vector rotation, is given by the following equation:

$$T_e = \frac{3}{2} p \, \bar{\psi}_S \times \bar{i}_S \tag{1}$$

where p is the number of motor pole pairs, i_s is the stator current vector and ψ_s represents the vector of the stator flux. Constant 3/2 indicates a nonpower invariant transformation form.

The requirement for the rotor flux vector to be perpendicular to the stator current vector is handled electrically through enhanced current control. This highlights the requirement for the stator current vector to be positioned in such a manner that the flux component, crucial for magnetizing the machine, is separated from the component responsible for generating torque.

This positioning can be accomplished by decomposing the current vector into two components projected in the reference frame, commonly referred to as the *dq* frame, which rotates synchronously with the rotor. It has become a standard practice to orient the *dq* reference frame in such a way that the *d*-axis aligns with the position of the rotor flux vector, enabling the current in the *d*-axis to modulate the amplitude of the rotor flux linkage vector. The reference frame position must be updated so that the *d*-axis is always aligned with the rotor flux axis.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Given that the rotor flux axis is fixed to the rotor position in PMSMs, a mechanical position transducer or position observer can be employed to measure both the rotor position and the position of the rotor flux axis. When the reference frame phase is set such that the *d*-axis aligns with the rotor flux axis, the current in the *q*-axis represents solely the torque-producing current component.

Furthermore, aligning the reference frame speed to be synchronous with the rotor flux axis speed results in both the *d*- and *q*-axis current components becoming DC values. This alignment implies the utilization of simple current controllers to control the demanded torque and magnetizing flux of the machine, therefore simplifying the control structure design.

- Measure the motor quantities (DC bus voltage and currents, rotor position/speed)
- Transform measured currents into the 2-phase orthogonal system *αβ* using a Clarke transformation, after that transform the currents in *αβ* coordinates into the *dq* reference frame using a Park transformation
- The stator current torque- and flux-producing components are separately controlled in the *dq* rotating frame
- The output of the control is the stator voltage space vector and it is transformed by an inverse Park transformation back from the *dq* reference frame into the 2-phase orthogonal system *αβ* fixed with the stator, the output 3-phase voltage is generated using a space vector modulation

The Clarke/Park transformations discussed above are part of the AMMCLib set ref.[2].

In order to decompose currents into torque- and flux-producing components (i_{sd}, i_{sq}), the position of the motor-magnetizing flux has to be known. This requires knowledge of the accurate rotor position as being strictly fixed with magnetic flux. This application note deals with the sensor-based FOC control, where the position and velocity are obtained by the position/velocity estimator executed by eTPU. Position and speed are processed by the eTPU coprocessor that runs independently on the system core.

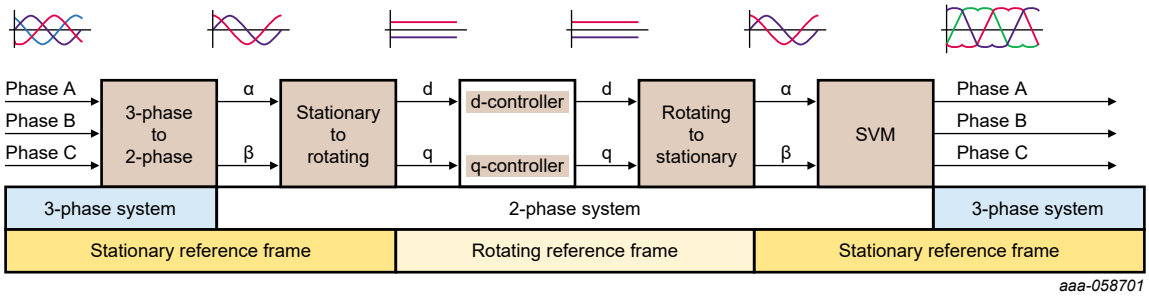


Figure 2. FOC transformations

3.2 PMSM model in rotational reference frame *dq*

The PMSM model in the *dq* reference frame is popular for FOC structures, because both controllable quantities, current and voltage, are DC values. Being DC values allows employing only simple controllers to force the machine currents into the defined states. Furthermore, full decoupling of the machine flux and torque can be achieved, which allows dynamic torque, speed, and position control. The equations describing voltages in the 3-phase windings of a PMSM can be written in matrix form as follows:

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

where R_s is the stator resistance, and the total linkage flux in each phase is given as:

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

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

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 Ψ_{PM} represents the magnetic flux generated by the rotor permanent magnets, and θ_e is the electrical rotor position.

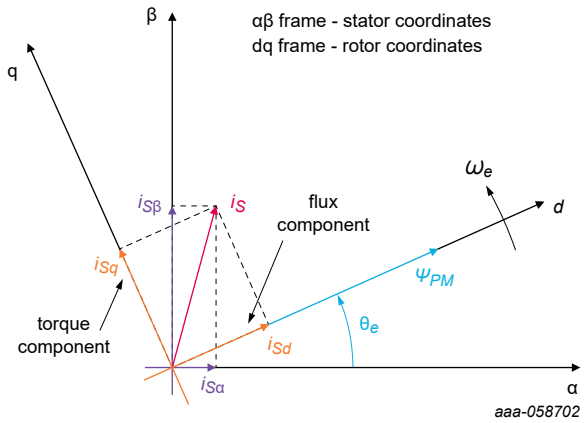


Figure 3. Orientation of stationary and rotational reference frames, with current components transformed into both frames

The voltage equations of the *dq* reference frame model are obtained by transforming the 3-phase voltage equations (Equation 2) and flux equations (Equation 3) into a 2-phase rotational frame aligned and rotating synchronously with the rotor, as shown in Figure 3. Following this transformation, the resulting equations are derived as:

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

where L_d, L_q are the *dq* inductances, and ω_e is the electrical angular velocity. It can be seen that Equation 4 represents a nonlinear cross-dependent system with cross-coupling terms in both the *d*- and *q*-axis and back-EMF voltage component in the *q*-axis.

3.2.1 Current control

When the FOC concept is employed, both cross-coupling terms are compensated to allow independent control of current *dq* components. The design of the controllers is then governed by the following pair of equations, derived from Equation 4 after compensation:

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

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

The previous equations describe the model of the plant for the *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. The closed current loop can then be designed with the following transfer functions representing the RL circuit and PI controller:

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

$$F_{RL}(s) = \frac{1}{Ls+R} \tag{7}$$

$$F_{PI}(s) = \frac{K_P s+K_I}{s} \tag{8}$$

The transfer function of a simplified closed current loop can be derived based on Equation 7 and Equation 8:

$$F_i(s) = \frac{F_{PI}(s)F_{RL}(s)}{1+F_{PI}(s)F_{RL}(s)} = \frac{\frac{K_P}{L}s+\frac{K_I}{L}}{s^2+\left(\frac{K_P+R}{L}\right)s+\frac{K_I}{L}} \tag{9}$$

The characteristic equation (Equation 9) reveals a second order system. However, the numerator of the transfer function introduces zero due to the PI controller. It increases the system overshoot, lowering the potential closed loop bandwidth. The NXP's motor control solution compensates the zero by a zero cancellation block in the feedforward. This addition can be performed in the MCAT tool. The zero cancellation block is defined as a first order filter with the following transfer function:

$$F_{ZC}(s) = \frac{1}{\frac{K_P}{K_I}s+1} \tag{10}$$

Introducing Equation 10 to Equation 9 the closed current loop transfer function is rearranged as:

$$F_i(s) = F_{ZC}(s)\frac{F_{PI}(s)F_{RL}(s)}{1+F_{PI}(s)F_{RL}(s)} = \frac{\frac{K_I}{L}}{s^2+\left(\frac{K_P+R}{L}\right)s+\frac{K_I}{L}} \tag{11}$$

The proportional and integral gains of the current controller can be designed by comparing the closed loop characteristic polynomial with the ideal second order system:

$$s^2 + s\left(\frac{K_P+R}{L}\right) + \frac{K_I}{L} = s^2 + 2\xi\omega_0 s + {\omega_0}^2 \tag{12}$$

where ω_0 represents the system's natural frequency and ξ is the damping factor. The proportional and integral gains of the current controller in the continuous domain can then be derived from Equation 12 as follows:

$$K_P = 2\xi\omega_0 L - R \tag{13}$$

$$K_I = {\omega_0}^2 L \tag{14}$$

The system's natural frequency is defined as:

$$\omega_0 = 2\pi f_0 \tag{15}$$

where f_0 is the frequency parameter, which can be modified in the MCAT tool by the application user. However, in real applications the control loop is updated in discrete steps. Therefore, a discrete representation of the current controller by bilinear discretization method from *s* to *z* domain is used:

$$K_P(z) = K_P \tag{16}$$

$$K_I(z) = K_I\frac{T_S}{2} \tag{17}$$

where T_S is the sampling period of the current loop, which is adjustable in NXP's EV-INVERTERGEN3 application. In the NXP's EV-INVERTERGEN3 solution, the recurrent PI controllers are set as a default in the MCAT tool regarding the FOC current loop. The difference equation derived by the bilinear discretization method for the recurrent form of the PI controller can be determined by inserting Equation 18 to the transfer function of PI controller (Equation 8).

$$s = \frac{2}{T_S}\frac{z-1}{z+1} \tag{18}$$

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Performing multiple mathematical operations, the following difference equation is derived:

$$u(k) = u(k-1) + \left(K_P + K_I\frac{T_S}{2}\right)e(k) + \left(-K_P + K_I\frac{T_S}{2}\right)e(k-1) \tag{19}$$

The terms with the proportional and integral gains in Equation 19 derived by the bilinear discretization method are defined as recurrent PI controller's coefficients CC1 and CC2:

$$CC1 = K_P + K_I\frac{T_S}{2} \tag{20}$$

$$CC2 = -K_P + K_I\frac{T_S}{2} \tag{21}$$

The block diagram of the entire NXP's solution of FOC current loop structure implemented in S32K396 is shown in Figure 4:

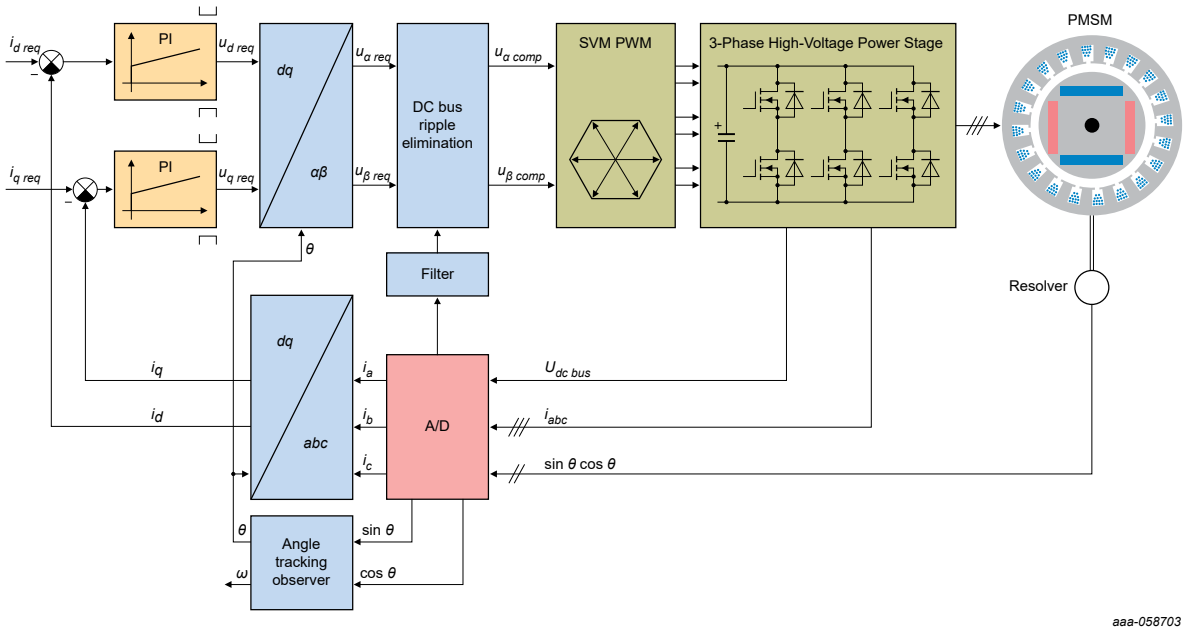


Figure 4. FOC current loop structure

Note: *The information about rotor position is needed in the FOC current loop to transform the measured 3-phase currents to the dq reference frame. The angle tracking observer (ATO) is used in the NXP's application to process the signals from the resolver to the electrical rotor position. The position is needed for Park transformation as illustrated in Figure 4.*

3.2.2 Speed control

The speed loop is the outer loop of the FOC structure with one PI controller. The output from the speed controller creates a reference for the *q*-axis current controller. Using a PI controller with an anti-windup function is advantageous to avoid current overloading of the motor or inverter. The parameters of the speed controller can be defined based on the transfer function of the mechanical system:

$$F_{MEC}(s) = \frac{k_t}{Js+B} \tag{22}$$

where k_t is the torque constant, J is the moment of inertia, and B is the viscous friction coefficient.

Note: *The viscous friction coefficient B is commonly neglected in many motor control applications due to its negligible value. Currently, the NXP's motor control solution is not considering coefficient B in the controllers' tuning.*

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

If the time constant of the motor electrical part is much smaller than the time constant of the motor mechanical part, the current loop can be neglected entirely in the design of the speed loop. The closed speed loop transfer function can then be derived based on the transfer functions of the speed controller and the mechanical system:

$$F_\omega(s) = \frac{F_{PI}(s)F_{MEC}(s)}{1+F_{PI}(s)F_{MEC}(s)} = \frac{\frac{k_t K_P}{J}s+\frac{k_t K_I}{J}}{s^2+\left(\frac{k_t K_P}{J}\right)s+\frac{k_t K_I}{J}} \quad (23)$$

Similar to the design of the current loop, zero is created in the transfer function numerator due to the influence of the PI controller. The zero in the transfer function can be compensated in the same way as in the case of the current loop, using a first order filter. This addition can be performed in the MCAT tool.

$$F_\omega(s) = F_{ZC}(s)\frac{F_{PI}(s)F_{MEC}(s)}{1+F_{PI}(s)F_{MEC}(s)} = \frac{\frac{k_t K_I}{J}}{s^2+\left(\frac{k_t K_P}{J}\right)s+\frac{k_t K_I}{J}} \quad (24)$$

Note: *MCAT tool also allows to choose the incremental ramp instead of zero cancellation block in the feedforward path.*

The characteristic polynomial from Equation 24 is then compared with the ideal polynomial of the second order system as follows:

$$s^2+\left(\frac{k_t K_P}{J}\right)s+\frac{k_t K_I}{J} = s^2+2\xi\omega_0 s+{\omega_0}^2 \quad (25)$$

The proportional and integral gains of the speed controller in the continuous domain can be then derived from Equation 25 as follows:

$$K_P = \frac{2\xi\omega_0 J}{k_t} \quad (26)$$

$$K_I = \frac{J}{k_t}\omega_0^2 \quad (27)$$

The discrete form of the controllers for the real application can be derived with the bilinear discretization method from *s* to *z* domain with the following equations:

$$K_P(z) = K_P \quad (28)$$

$$K_I(z) = K_I\frac{T_s}{2} \quad (29)$$

where T_s is the sampling period of the speed loop, which is adjustable in NXP's EV-INVERTERGEN3 application. In the NXP's EV-INVERTERGEN3 with S32K396, the parallel PI controller is set as a default in the MCAT tool regarding FOC speed loop. However, the MCAT tool allows to make a change from parallel to recurrent PI speed loop controller type. The block diagram of the entire NXP's solution of the FOC speed loop structure is shown in Figure 5.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

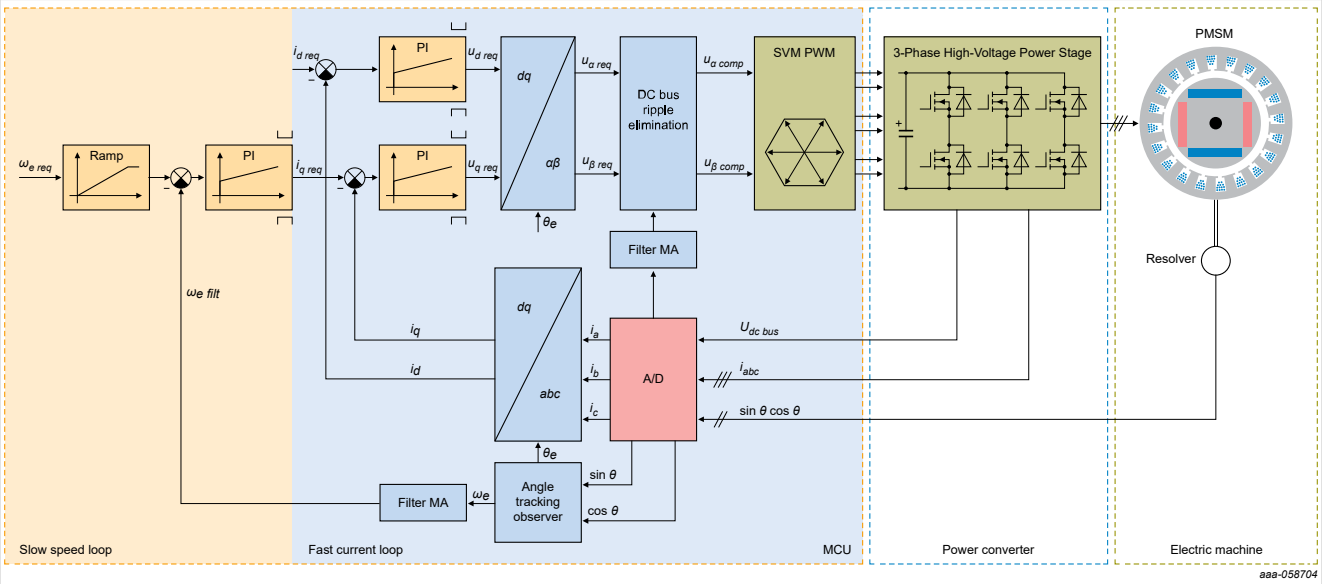
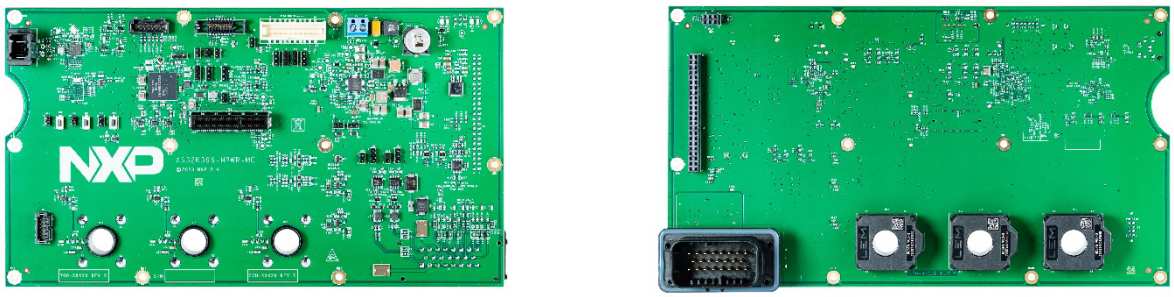


Figure 5. FOC speed loop structure

3.3 Current, voltage, and position sensing

The EV-INVERTERGEN3 uses three Hall effect sensors placed in output phases. DC, AC, or pulsed stator phase current, which flows through the sensor, produces a voltage drop based on the Hall effect. This effect is interfaced with the AD converter of the microcontroller through conditional circuitry schematic diagram sheets Figure 7 and Figure 8. The schematic diagram Figure 8 shows a gain setup and input signal filtering circuit for an operational amplifier that provides the conditional circuitry and adjusts voltages to fit into the ADC input voltage range.

Figure 6. Board layout



top bottom

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

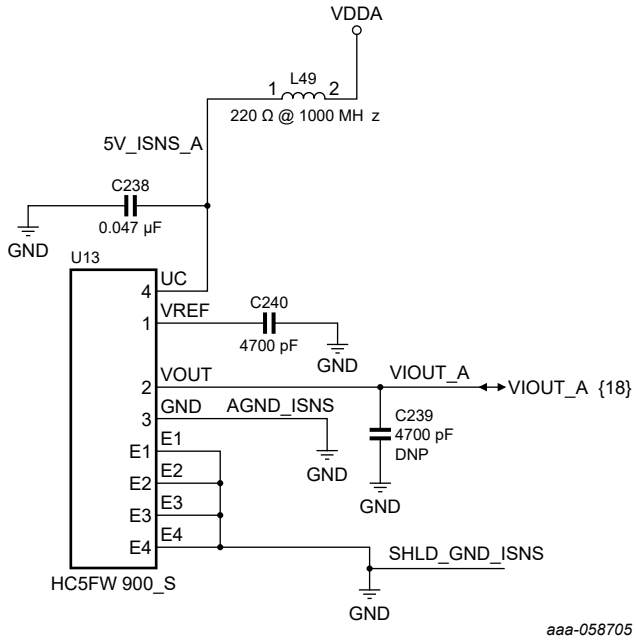


Figure 7. Signal conditioning circuit for Hall effect sensor HC5FW 900-S

The phase current is sampled during the PWM period. In standard motor operation, where the supplied voltage is generated using the space vector modulation, the sampling instant of phase current occurs during the PWM period. The total number of ten samples is used for the average value calculation. Calculated value is used for FOC algorithm utilization. The typical bandwidth for open loop transducers HC5FW is 40 kHz.

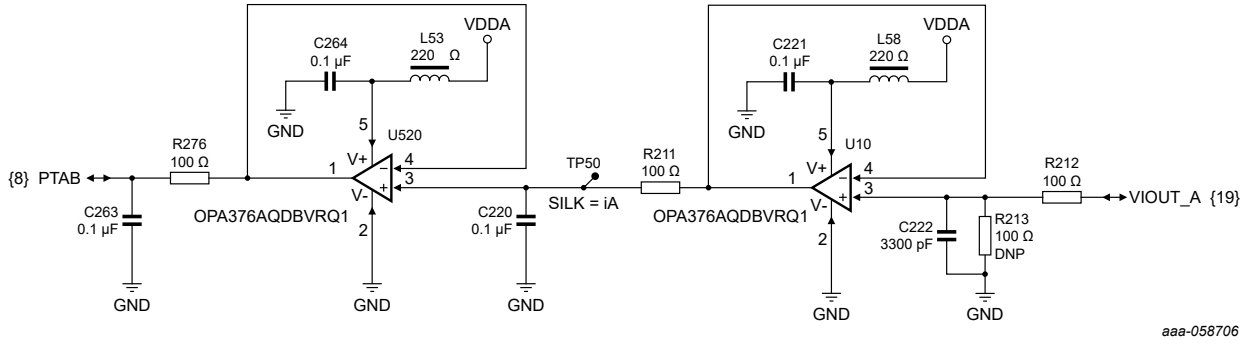


Figure 8. Signal conditioning circuit for phase A current sensing

In a typical voltage sensing implementation, a resistive voltage divider is used to scale the DC bus voltage to suit the input range of the voltage sensor U11. A differential output voltage that is proportional to the input voltage is created on the other side of the optical isolation barrier. A typical 100 kHz wide bandwidth provides enough information for the effective motor control use case.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

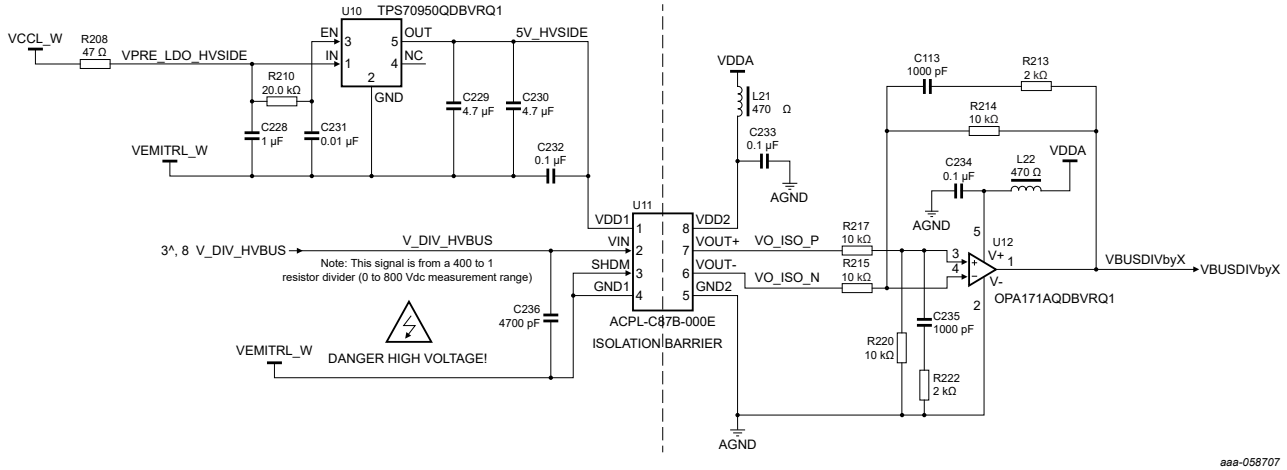


Figure 9. Isolated voltage sensing circuit

The excitation signal for the resolver is generated by a series of operational amplifiers. Excitation circuits consist of a Sallen-Key third-order filter and output amplifier. The Sallen-Key filter produces a sine wave for resolver excitation at 10 kHz. The output amplifier modulates excitation amplitude at the specific level necessary for the resolver.

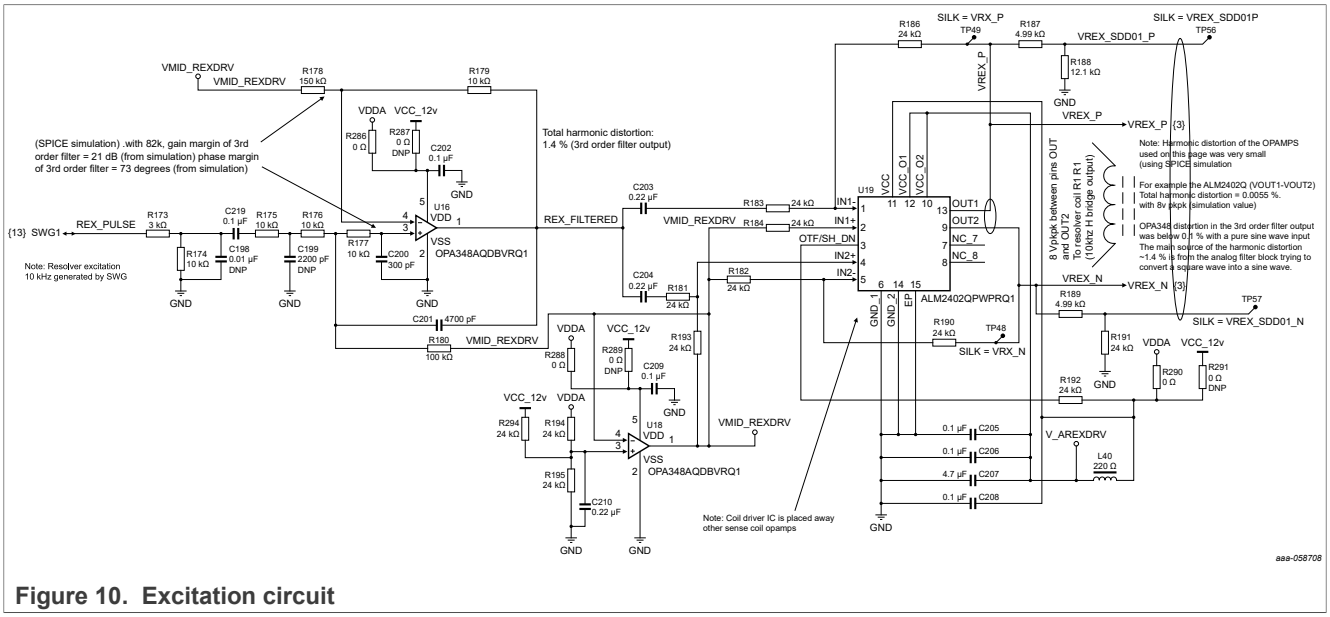


Figure 10. Excitation circuit

Output signal from resolver sin and cos signals are connected directly to conditioning circuits. After conditioning, signals are processed in SDADC running in differential mode.

3.4 Field weakening

Field weakening is an advanced control approach that extends standard FOC to allow electric motor operation beyond a base speed. The back electromotive force (EMF) 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 back-EMF. 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 the inverter. Base speed defines the rotor speed at which the back-EMF reaches the maximal value, and the motor still produces the maximal torque.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

As the difference between the induced back-EMF and the supply voltage decreases, the phase current flow is limited, therefore the currents i_d and i_q cannot be controlled sufficiently. Further increase in speed eventually results in a back-EMF 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 back-EMF 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 11.

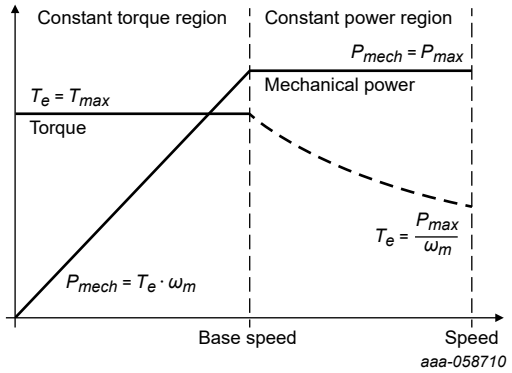


Figure 11. Constant torque/power operating regions

Operation in the 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 toward its reference. The operation in the 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 12.

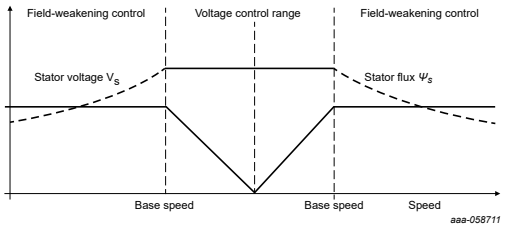


Figure 12. Constant flux/voltage operational regions

FOC splits phase currents into the *q*-axis torque component and the *d*-axis flux component. The flux current component i_d is used to weaken the stator magnetic flux linkage Ψ_s. Reduced stator flux Ψ_s yields lower back-EMF and the condition of field weakening is met. More details can be seen from the following phasor diagrams of the PMSM operated exposing FOC control without (left) and with field weakening (right) Figure 13.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

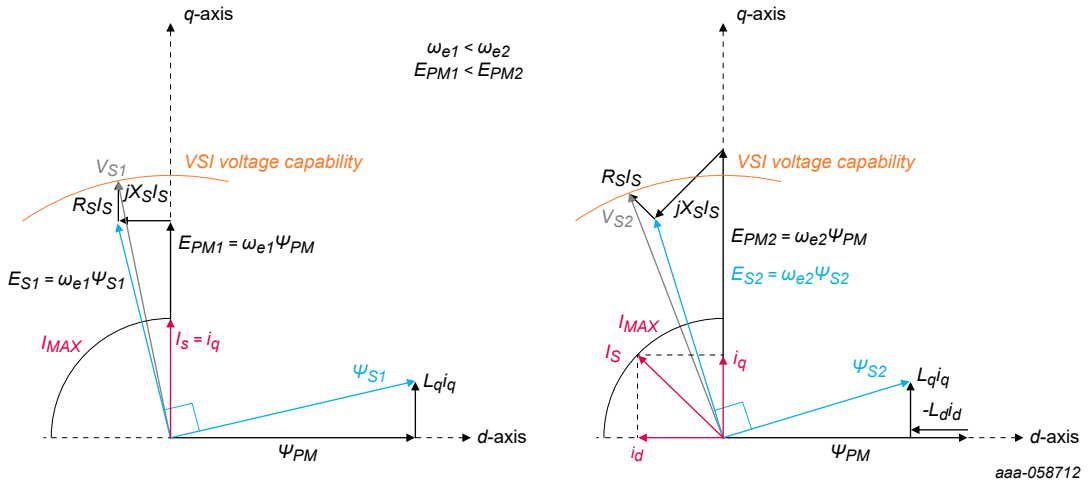


Figure 13. Steady-state phasor diagram of PMSM operation up to base speed (left) and above speed (right)

FOC without field weakening is operated by demanding the *d*-axis current component to be zero (i_d = 0) to excite the electric machine just by permanent magnets mounted on the rotor. This is an operation within the constant torque region (see Figure 11) because the whole amount of the stator current consists of the torque-producing component i_q only (see Figure 13, left). If the rotor speed exceeds the base speed, the back-EMF voltage $E_{s1} = \omega_{e1}\Psi_{s1}$ approaches the limit given by VSI, and the 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 the stator flux linkage Ψ_{s2} by $-L_d I_d$ component as shown in Figure 13 (right). Thanks to this approach, back-EMF voltage induced in the stator windings E_{s2} is reduced below the VSI voltage capability. The *q*-axis current i_q current can be controlled again to develop torque as demanded. Unlike the previous case, this is an operation within the constant power region (see Figure 11) where i_q current is limited due to the stator current I_s limitation as evident in Figure 13 (right).

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

- Voltage amplitude U_s is limited to u_{max} defined by actual DC bus voltage as shown in Figure 14 (left)
- Phase current amplitude i_{max} is limited by the capabilities of power devices and motor thermal design as shown in Figure 14 (right)
- Flux linkage in the *d*-axis is limited to prevent demagnetization of the permanent magnets

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

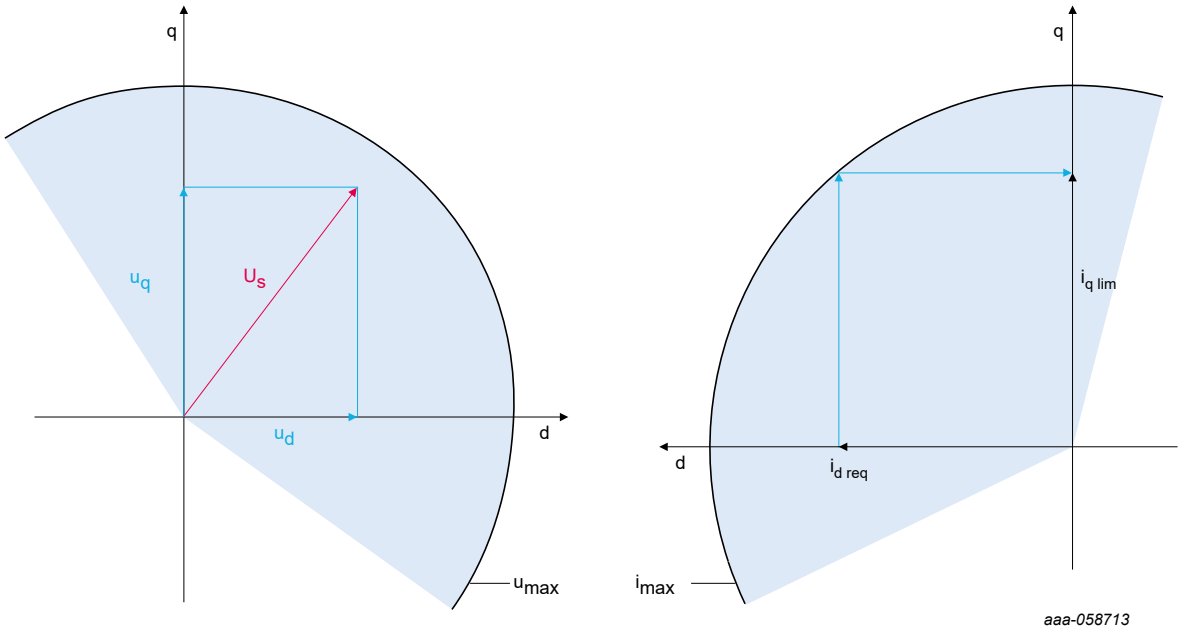


Figure 14. Voltage (left) and current (right) limits for PMSM drive operation

NXP's AMMCLib offers a software solution for the FOC with field weakening respecting all limitations discussed above. This library-based function is discussed in Section 4.4.4 "AMMCLib integration".

Note: *Field weakening functionality is implemented through the AMCLIB_FWSpeedLoop_FLT function. Operation in the field weakening region was not evaluated using the software version and dynamometer test setup described in system application note AN14505.*

4 Software implementation on the S32K396

4.1 eTPU

The eTPU is a programmable I/O controller with its own core and memory system, allowing it to perform complex timing and I/O management independently of the CPU. The eTPU is used as a coprocessor, specialized for advanced timing functions, such as handling complex motor control and communication tasks independently of the CPU.

A new complex library of eTPU functions enabling the eTPU to drive motor control applications was developed. This library represents a step forward compared to its predecessor, the motor control function sets (set3 and set4). The new motor control eTPU library benefits from NXP eTPU development tools from CodeWarrior. The eTPU(2) development suite is based on the Eclipse IDE and includes the C and assembly compiler, simulator, and debugger.

4.1.1 eTPU based resolver to digital converter (RDC)

The resolver digital interface uses an eTPU-based trigger to initiate sine wave generator (SGEN). SGEN generates a single harmonic sinusoidal frequency needed to resolver coil excitation. A sinusoidal SGEN output signal is passed through an external low-pass filter and used as a resolver excitation signal. In the resolver position sensor, this excitation signal is modulated by the sine and cosine of the actual motor angle. The feedback sine and cosine signals are sampled by an on-chip ADC and the conversion results can be transferred to eTPU DATA RAM by direct memory access (DMA). Then, the eTPU can process the digital samples of resolver output signals. Motor angular position, angular speed, revolution counter, and diagnostics are results of the sine and cosine feedback signal processing. See Figure 15.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

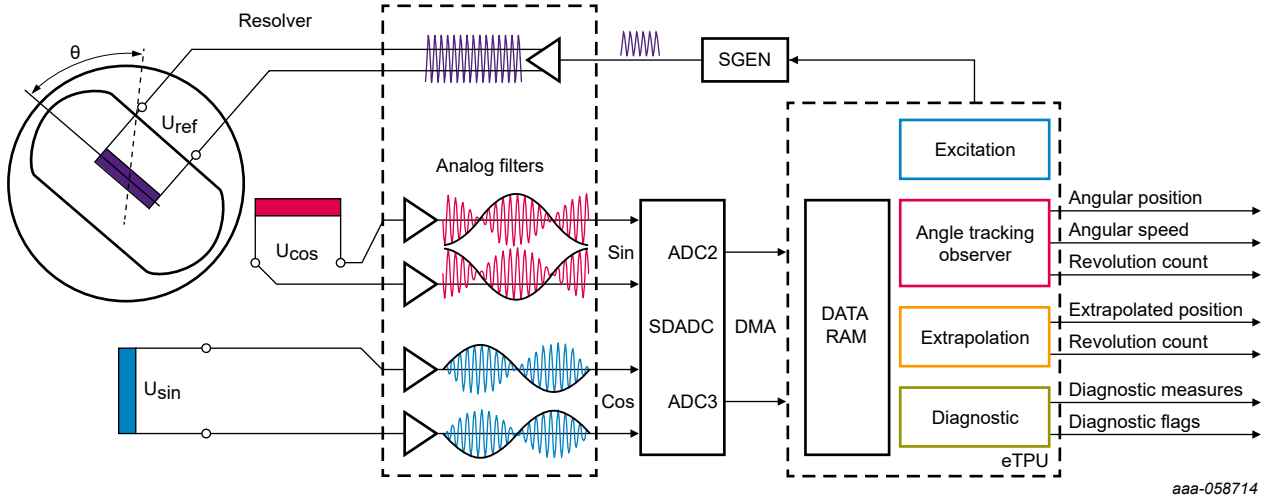


Figure 15. eTPU resolver digital interface block diagram

Processing of the feedback signals is executed on a separate channel. Another channel is used to perform linear extrapolation of the last updated position from ATO to any other time. The use of another channel is an important feature because the ATO updates come with a certain period (~50 µs), which most likely is not aligned with the control loop frequency.

Optionally, another eTPU channel can be used to process diagnostics either on the same eTPU engine after the feedback signal processing is finished or on the other eTPU engine in parallel to the motor angle and speed calculation. This enables the CPU application to read the new motor angle and, at the same time, check the diagnostic results to ensure that the motor angle is correct.

Two SDADC modules are used to continuously sample the sine and cosine signals in parallel, as shown in Figure 16. These modules are configured to obtain 32 samples of each signal per period instead of 1 sample at the presumed peak, as it is implemented in most of the SW resolver applications. This oversampling method, together with demodulation and filtration, brings more robustness toward the induced noise. Furthermore, the position is evaluated twice per resolver excitation period.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

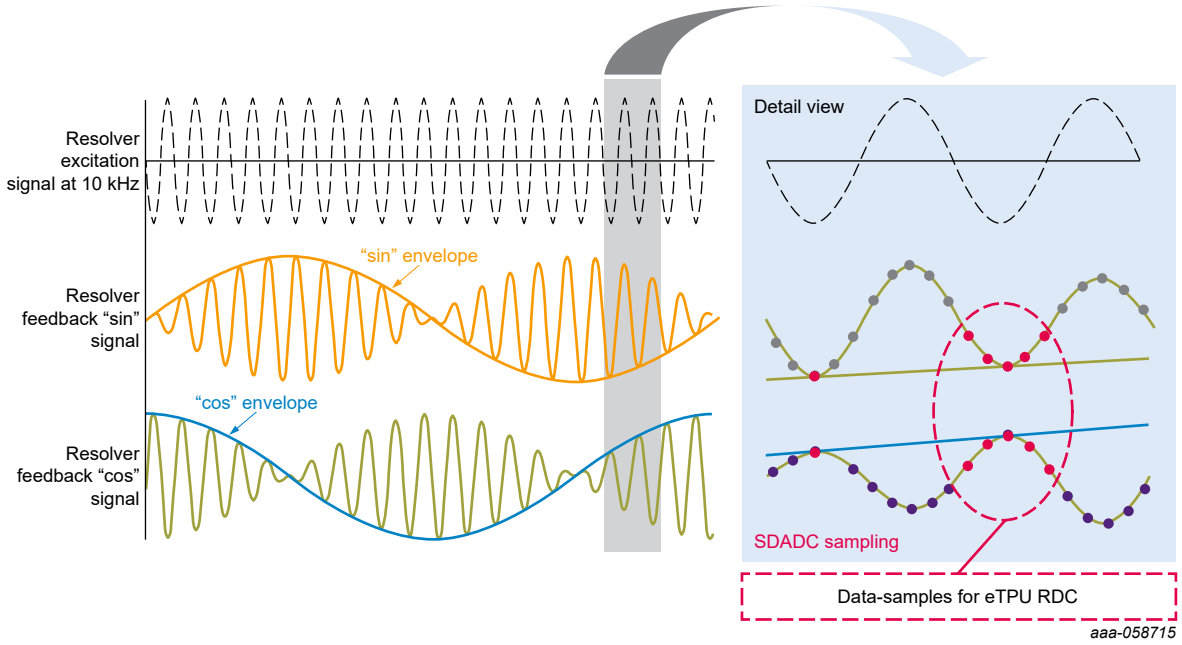


Figure 16. Oversampling and demodulation of resolver feedback signals

4.2 S32K396 - key modules for PMSM FOC control

The key modules for motor control on the S32K396 device are introduced in this part of the application note. Using the eTPU motor control library function set, this programmable timer coprocessor is capable of resolver feedback signal processing. To enable full eTPU resolver functionality, it is crucial to use SDADCs together with a DMA module. For phase current sensing, the SARADC is used.

Figure 17 shows module interconnection for a typical PMSM FOC application working in sensor-based mode using current sensing with eTPU motor control library functions involvement.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

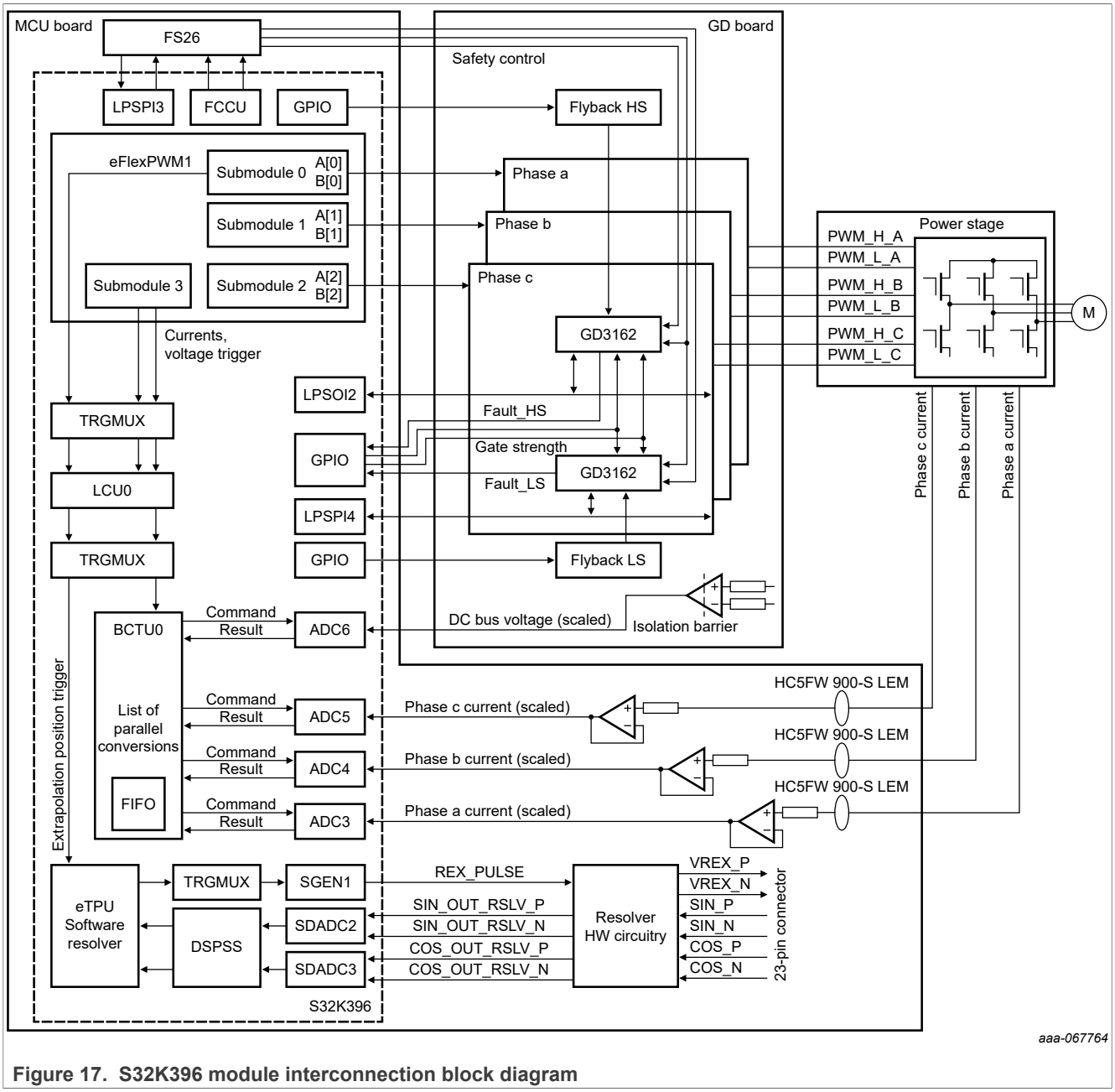


Figure 17. S32K396 module interconnection block diagram

4.2.1 Module involvement in PMSM FOC control

This section discusses timing and module synchronization to accomplish PMSM FOC on the S32K396 and the internal hardware features. The timing diagram of the automatic synchronization between eFlexPWM, eTPU, and ADC in the EV-INVERTERGEN3 PMSM FOC application is shown in Figure 18.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

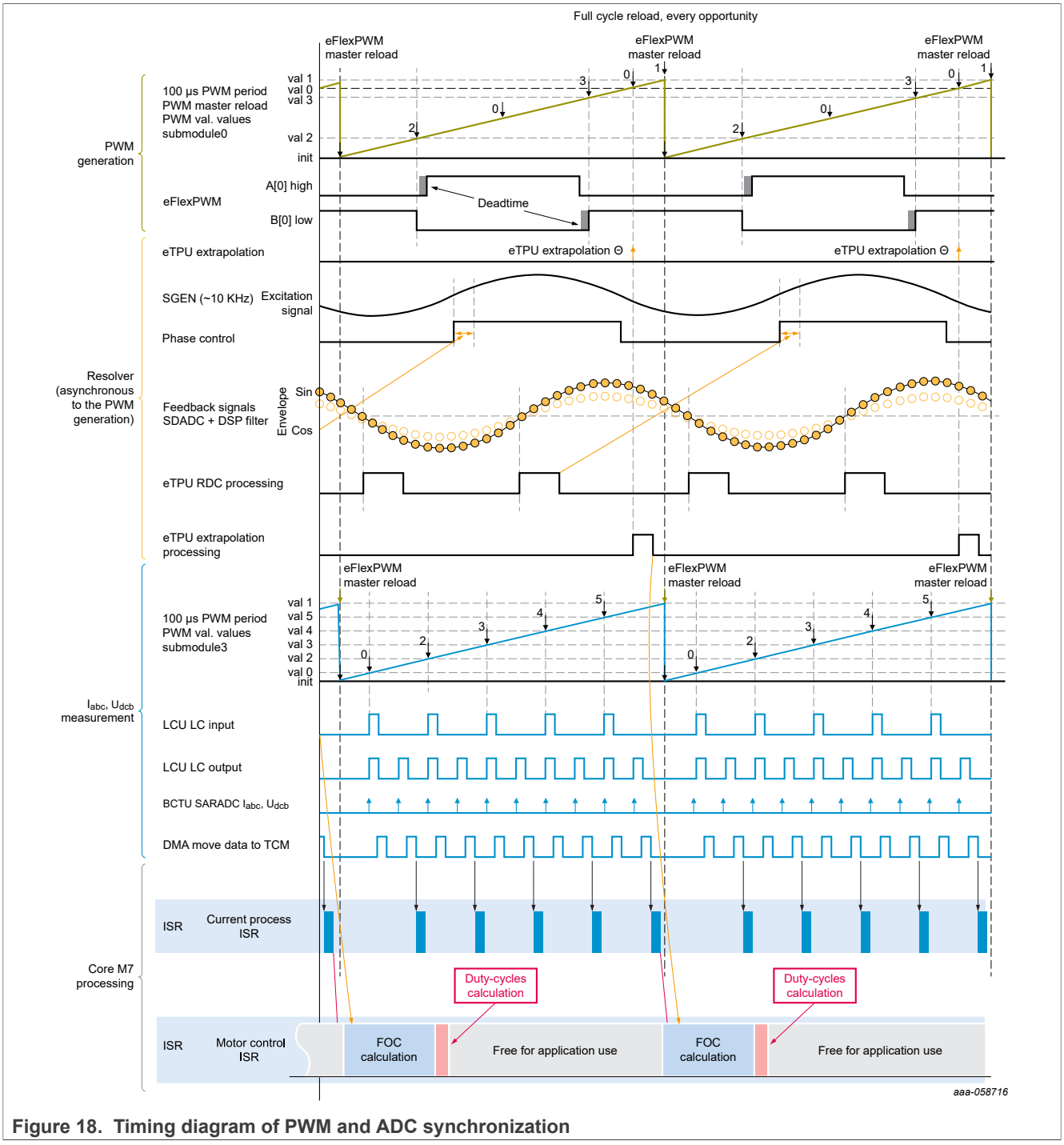


Figure 18. Timing diagram of PWM and ADC synchronization

The eFlexPWM module generates a main triggering signal for the whole application. The LCU creates the triggers for the 3-phase currents and DC bus voltage measurement. It performs an oversampling of these variables ten times per PWM period. When conversions of 3-phase currents and DC bus voltage measurement are done, data are stored in BCTU FIFO. Data are transferred by DMA to Core TCM. The interrupt service routine (ISR) is triggered after the DMA transfer. At the beginning of ISR, position is read from eTPU memory and FOC is calculated with PWM update values. eTPU is triggering SGEN and providing shifting of sampling signal automatically by an independent regulator.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.3 S32K396 device initialization

To simplify and accelerate application development, an embedded part of the PMSM FOC motor control application has been created. Peripherals are initialized in the *main()* function. For each S32K396 module, there is a specific configuration function:

- Clock_Ip_Init(); - Clock configuration
- OsIf_Init(); - Operation system timer configuration
- IntCtrl_Ip_Init(); - Configuration of custom interrupt
- Power_Ip_SetMode(); - Mode entry initialization
- Siul2_Port_Ip_Init(); - Pin configuration
- Siul2_Port_Ip_SetInputBuffer(); - Assign eTPU input from TRGMUX
- Siul2_Icu_Ip_Init(); Siul2_Icu_Ip_EnableInterrupt(); Siul2_Icu_Ip_EnableNotification(); - Configure and enable GD3162 interrupts
- Trgmux_Ip_Init(); - TRGMUX configuration
- Igf_Init(); - Input Glitch Filter configuration
- Lcu_Ip_Init(); Lcu_EnableOutput(); - LCU configuration
- Lpspi_Ip_Init(); - LPSPI initialization
- Sbc_fs26_Init(); Sbc_fs26_WriteRegister(); - FS26 initialization
- Gd3162_EnableFlybackSupplies(); - Enable flybacks for GD3162 HS and LS
- CDD_Gd3162_InitDriver(); CDD_Gd3162_RequestReset(); CDD_Gd3162_ConfigDevice(); - GD3162 initialization
- Gd3162_EnableGateStrengthGpioControl(); - GD3162 gate strength GPIO control configuration
- Adc_Sar_Ip_Init(); - SARADC configuration
- Bctu_Ip_Init(); - BCTU configuration
- Sgen_Init(); - SGEN configuration
- Etpu_Ip_Init(); Etpu_Resolver_Ip_Init(); - Resolver on the eTPU initialization
- Etpu_Ip_Start(); - Start eTPU functionality
- Sdadc_Ip_Init(); Sdadc_Ip_EnableHwTrigger(); Sdadc_Ip_SetHwTrigger(); - SDADC configuration
- Sdadc_Ip_SwTriggerConversion(); - SDADC synchronous start
- Dspss_Start(); - DSPSS start
- Dma_Ip_Init(); Dma_Configure(); - DMA configuration
- DmaMux_Configure(); - DMA MUX configuration
- FlexCan_Ip_Init(); - FlexCAN initialization
- FMSTR_Init(); - FreeMASTER initialization
- Gd3162_SetGateStrengthByGpio(); - GD3162 gate strength control through GPIO
- FlexPwm_Ip_Init(); FlexPwm_ConfigureTriggers(); - FlexPWM configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.3.1 Clock configuration

The S32K396 platform supports up to 320 MHz. The eTPU bus interface, memories, and functional module run at M7_CORE_CLK, which is up to 320 MHz. The new DSPSS module is connected to CORE_CLK. However, the RAM clock of the DSPSS is derived by M7_CORE_CLK. Table 1 shows the clock configuration in this application.

Table 1. S32K396 clocking configuration

Clock	Frequency
CORE_CLK	160
M7_CORE_CLK	320
AIPS_SLOW_CLK	40
FlexCAN	40
eTPU	M7_CORE_CLK
SDADC	80
SGEN	20
DSP	160
SARADC	80
BCTU	CORE_CLK
DMA	CORE_CLK
eFlexPWM	CORE_CLK
LCU	CORE_CLK
TRGMUX	AIPS_SLOW_CLK
LPSPI	AIPS_SLOW_CLK

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

The configuration of the overall system clock is performed within a function called *Clock_Ip_Init()*. See the code example below.

```
Clock_Ip_StatusType Clock_Ip_Init(Clock_Ip_ClockConfigType const * Config)
{
    Clock_Ip_StatusType ClockStatus = CLOCK_IP_ERROR;
    Clock_Ip_PllStatusType PllStatus;
    (void)Clock_Ip_au8XoscCallbackIndex;
    (void)Clock_Ip_axExtOscCallbacks;
    (void)Clock_Ip_au8IrcoscCallbackIndex;
    (void)Clock_Ip_axIntOscCallbacks;
    (void)Clock_Ip_au8PcfsCallbackIndex;
    (void)Clock_Ip_axPcfsCallbacks;
    (void)Clock_Ip_au8DividerCallbackIndex;
    (void)Clock_Ip_axDividerCallbacks;

    (void)CLOCK_IP_MODULE_INSTANCE;
    (void)CLOCK_IP_CALLBACK;
    (void)CLOCK_IP_EXTENSION_INDEX;
    (void)CLOCK_IP_POWER_MODE_INDEX;
    (void)CLOCK_IP_SELECTOR_INDEX;
    (void)CLOCK_IP_DIVIDER_INDEX;
    (void)CLOCK_IP_GATE_INDEX;
    (void)CLOCK_IP_PCFS_INDEX;
    (void)CLOCK_IP_CMU_INDEX;

#if (defined(CLOCK_IP_ENABLE_USER_MODE_SUPPORT))
    #if (STD_ON == CLOCK_IP_ENABLE_USER_MODE_SUPPORT)
        /* Set user access allowed for Clock */
        Clock_Ip_Command(Config, CLOCK_IP_SET_USER_ACCESS_ALLOWED_COMMAND);
    #endif
#endif
    CLOCK_IP_DEV_ASSERT(NULL_PTR != Config);
    Clock_Ip_InitClock(Config);

    if (DriverContext.ClockTreeIsConsumingPll)
    {
        PllStatus = Clock_Ip_GetPllStatus();
        if(CLOCK_IP_PLL_LOCKED == PllStatus)
        {
            Clock_Ip_DistributePll();
            ClockStatus = CLOCK_IP_SUCCESS;
        }
    }
    else
    {
        ClockStatus = CLOCK_IP_SUCCESS;
    }
return ClockStatus;
}
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.3.2 Port control and pin multiplexing

The EV-INVERTERGEN3 application has the following pins configuration:

Table 2. Port control and pin multiplexing

Module	Signal name	Pin name/Functionality	Description
eFlexPWM1	PWM_GD_AH	PTC31/PWM_1_A[0]	Complementary PWM output for phase A HS
	PWM_GD_AL	PTA6/PWM_1_B[0]	Complementary PWM output for phase A LS
	PWM_GD_BH	PTC30/PWM_1_A[1]	Complementary PWM output for phase B HS
	PWM_GD_BL	PTA7/PWM_1_B[1]	Complementary PWM output for phase B LS
	PWM_GD_CH	PTC29/PWM_1_A[2]	Complementary PWM output for phase C HS
	PWM_GD_CL	PTC8/PWM_1_B[2]	Complementary PWM output for phase C LS
LPSPI2	GD3162_HS_CS	PTF3/LPSPI2_PCS0	GD3162 HS LPSPI chain chip select
	GD3162_HS_MOSI	PTF2/LPSPI2_SOUT	GD3162 HS LPSPI chain data output
	GD3162_HS_MISO	PTF1/LPSPI2_SIN	GD3162 HS LPSPI chain data input
	GD3162_HS_SCK	PTF0/LPSPI2_SCK	GD3162 HS LPSPI chain clock
LPSPI4	GD3162_LS_CS	PTE23/LPSPI4_PCS0	GD3162 LS LPSPI chain chip select
	GD3162_LS_MOSI	PTC11/LPSPI4_SOUT	GD3162 LS LPSPI chain data output
	GD3162_LS_MISO	PTE21/LPSPI4_SIN	GD3162 LS LPSPI chain data input
	GD3162_LS_SCK	PTC27/LPSPI4_SCK	GD3162 LS LPSPI chain clock
SIUL	GD3162_EN_FLYBK_HS	PTD30/GPIO126	GD3162 enable the HS flybacks power supply
	GD3162_EN_FLYBK_LS	PTD31/GPIO127	GD3162 enable the LS flybacks power supply
	GD3162_FSS_LS	PTC28/GPIO92	GD3162 fail save state for the LS
	GD3162_FSS_HS	PTF4/GPIO164	GD3162 fail save state for the HS
	GD3162_FSENB	PTG0/GPIO192	GD3162 fail save output enabled
	GD3162_INTB_LS	PTC25/EIRQ20	GD3162 LS failure interrupt
	GD3162_INTB_HS	PTC26/EIRQ21	GD3162 HS failure interrupt
	GD3162_GS_LA	PTB1/GPIO33	GD3162 phase A falling edge GS control
	GD3162_GS_HA	PTF21/GPIO181	GD3162 phase A rising edge GS control
	GD3162_GS_LB	PTC20/GPIO84	GD3162 phase B falling edge GS control
	GD3162_GS_HB	PTC24/GPIO88	GD3162 phase B rising edge GS control
	GD3162_GS_LC	PTC9/GPIO73	GD3162 phase C falling edge GS control
	GD3162_GS_HC	PTF18/GPIO178	GD3162 phase C rising edge GS control
LPSPI3	FS26_LS_CS	PTF16/LPSPI3_PCS0	FS26 LPSPI chain chip select
	FS26_LS_MOSI	PTA17/LPSPI3_SOUT	FS26 LPSPI chain data output
	FS26_LS_MISO	PTD20/LPSPI3_SIN	FS26 LPSPI chain data input
	FS26_LS_SCK	PTE7/LPSPI3_SCK	FS26 LPSPI chain clock
CAN0	CAN0_RX	PTC23/CAN0_RX	FreeMASTER CAN communication interface
	CAN0_TX	PTC21/CAN0_TX	
SWG1	—	Pin A8	Resolver excitation signal

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 2. Port control and pin multiplexing*...continued*

Module	Signal name	Pin name/Functionality	Description
ADC3	ADC3 channel P1	PTA8/ADC3_P1	Phase A current measurement
ADC4	ADC4 channel P5	PTB8/ADC4_P5	Phase B current measurement
ADC5	ADC5 channel P2	PTD27/ADC5_P2	Phase C current measurement
ADC6	ADC6 channel P4	PTA0/ADC6_P4	DC bus voltage measurement
ADC0/ADC1	MTRTD1_SIG	PTD1/ADC1_S13/ADC0_P0	Measurement of the external temperature sensor
	MTRTD2_SIG	PTE26/ADC1_P0/ADC0_S12	
SDADC2	SIN_OUT_RSLV_P	PTB9/SDADC2_AN[0]	Resolver differential positive sin feedback signal
	SIN_OUT_RSLV_N	PTD26/SDADC2_AN[1]	Resolver differential negative sin feedback signal
SDADC3	COS_OUT_RSLV_P	PTD28/SDADC3_AN[0]	Resolver differential positive cos feedback signal
	COS_OUT_RSLV_N	PTA1/SDADC3_AN[1]	Resolver differential negative cos feedback signal

4.3.3 FlexCAN configuration

In this application, the FlexCAN is used for communication between the S32K396 and FreeMASTER Run-Time Debugging Tool. The S32K396 contains up to six instances of FlexCAN. The instance FlexCAN_0 is used in this application. The FlexCAN RTD configuration is shown in Figure 19.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

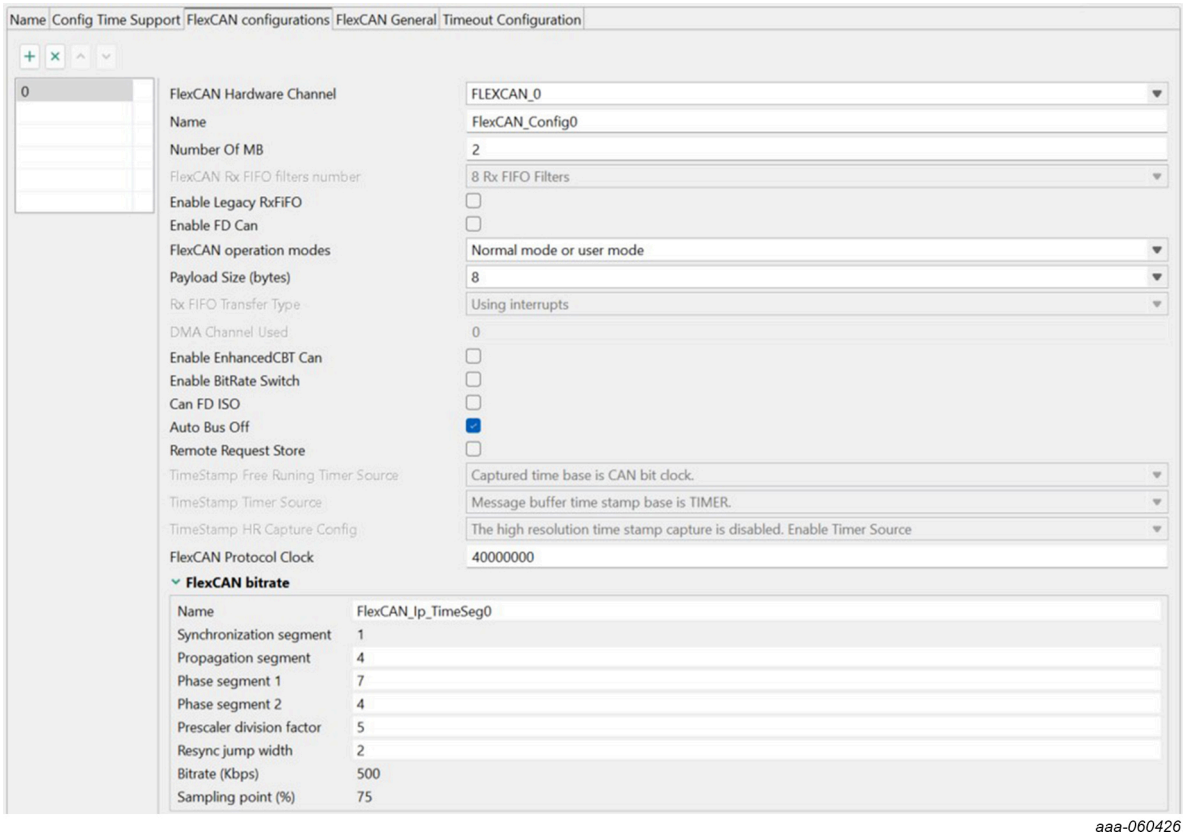


Figure 19. FlexCAN configuration

4.3.4 eTPU Resolver configuration

In this application, the eTPU is used to implement the resolver to digital converter. This project uses the eTPU RTD to implement the software resolver, shown in Figure 20. The resolver function is designed so that it cooperates closely with SDADC modules. SDADCs are configured to provide 32 samples per period of resolver feedback signal. Considering the resolver excitation frequency to be 10 kHz means a 320 kHz output data rate is required for the SDADC. With the given limitations of SDADC clocking and oversampling rate configurations, the closest possible output data rate that can be configured is 320.512 kHz. See Table 3. This determines the resolver excitation frequency to be 10.016 kHz, 99.84 µs period respectively. Resolver bandwidth configuration can be changed by modifying ATO P and I gains. The current configuration of ATO is for 1 kHz bandwidth. Excitation P and I gains are tuned for a balanced phase-shift reaction. Therefore, it is recommended to use this configuration. For more information about eTPU resolver implementation and its configuration, refer to the eTPU RDC and RDC Checker User Guide ref.[3].

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

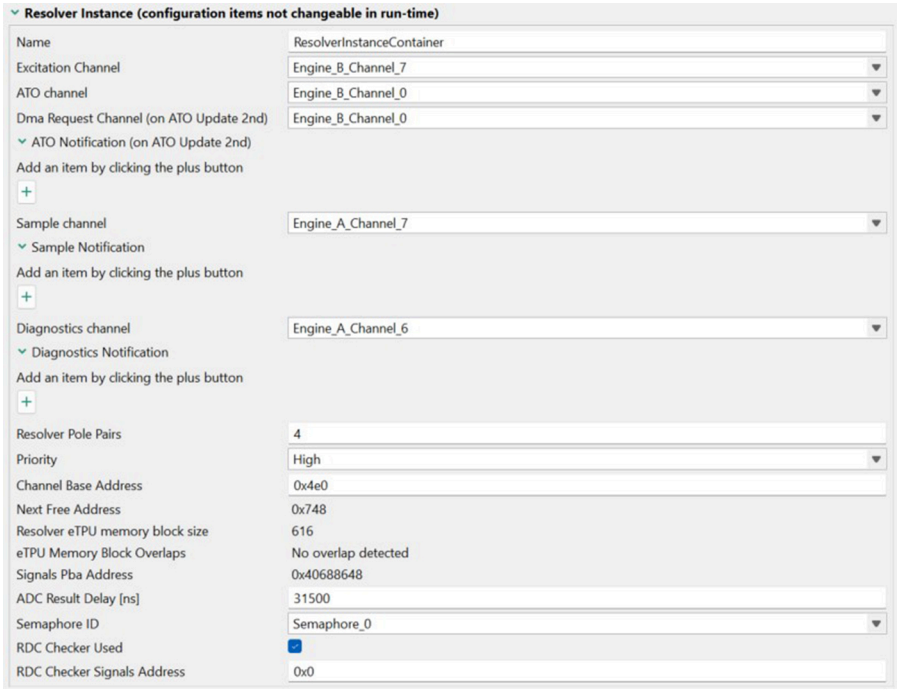


Figure 20. eTPU Resolver configuration

```
Etpu_Resolver_Ip_ConfigType Etpu_Resolver_Ip_Config_BOARD_INITPERIPHERALS_I0 =
{
    /* .SmprId */                ETPU_RESOLVER_SEMAPHORE_0,
    /* .Options */               (uint8)(ETPU_RESOLVER_OPTIONS_CALCULATION_ON
                                        |
 ETPU_RESOLVER_OPTIONS_EXC_ADAPTATION_ON
                                        |
 ETPU_RESOLVER_OPTIONS_EXC_GENERATION_ON
                                        | ETPU_RESOLVER_OPTIONS_RDC_CHECKER_ON),
    /* .ExcitationPeriod */      32000U,
    /* .AtoPGain */              593075,
    /* .AtoIGain */              20972,
    /* .ExcPGain */              0,
    /* .ExcIGain */              1007,
    /* .QEwmaSpeed */            8304721
};
```

4.3.5 SDADC configuration

The sine and cosine analog feedback signals need to be converted to a digital representation and transferred to eTPU DATA RAM for resolver function processing. This should be done independently of the CPU using an on-chip ADC and DMA. Although many of the ADC modules can be used, the described configuration adapts the SDADC.

Two SDADC modules are used to continuously sample the sine and cosine signals in parallel. They are configured to obtain 32 samples of each signal per period. The following table details the configuration for a 10 kHz excitation signal and a 320 kHz sampling frequency.

All information provided in this document is subject to legal disclaimers.

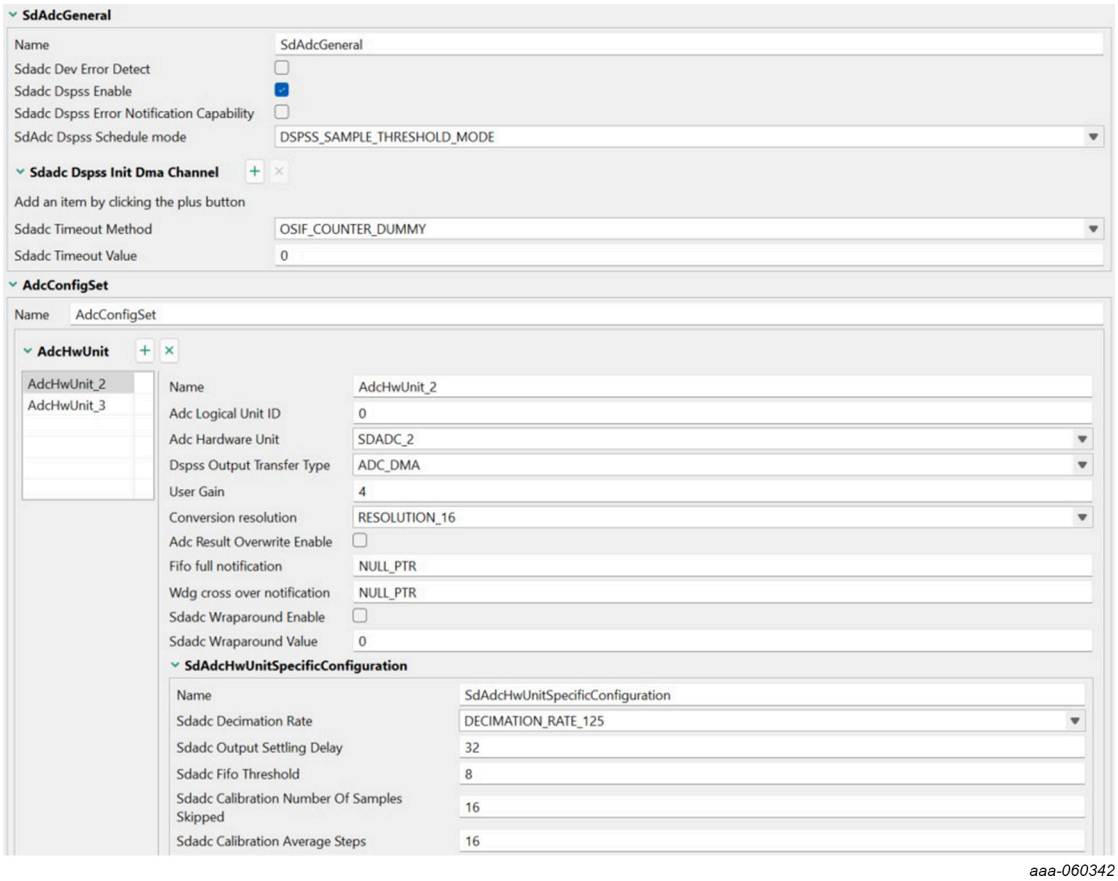
© 2026 NXP B.V. All rights reserved.

Table 3. SDADC configuration for resolver

Configuration item	Value
SDADC clock	80 MHz
SDADC sampling frequency	40 MHz
ADC decimation rate	125
Result output data rate	40 MHz / 125 = 320 kHz
Input mode	Differential
FIFO size	16 words
FIFO threshold	8
DMA request on FIFO full	selected and enabled

Note: *In this application, differential mode is configured for SDADC to sense resolver feedback signals (sine and cosine). Differential configuration for SDADC brings the advantage of noise rejection.*

The SDADC in S32K396 platform removes the internal high-pass filter component, and DSP SUBSYSTEM (DSPSS) is added for signal processing. SDADC_2 and SDADC_3 instances are used for sine and cosine signals respectively. The configuration parameters of SDADC_2 instance are shown in Figure 21. The SDADC_3 instance has similar configuration parameters.



aaa-060342

Figure 21. SDADC General configuration

The following initialization process is needed by SDADC:

Copyright 2026 NXP. NXP Confidential. This software is owned or controlled by

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
NXP and may only be used strictly in accordance with the applicable license
terms found at https://www.nxp.com/LA_OPT_NXP_SW. Only the "internal use
license" in Section 2.2 in the NXP SOFTWARE LICENSE AGREEMENT is granted for this software.

 /* SDADC configuration */
 Sdadc_Ip_Init(2, &SdadcIpConfig_0_BOARD_InitPeripherals);
 Sdadc_Ip_Init(3, &SdadcIpConfig_1_BOARD_InitPeripherals);
 Sdadc_Ip_EnableHwTrigger(2);
 Sdadc_Ip_EnableHwTrigger(3);
 Sdadc_Ip_SetHwTrigger(2, SDADC_IP_TRIGGER_RISING_EDGE, 2);
 Sdadc_Ip_SetHwTrigger(3, SDADC_IP_TRIGGER_RISING_EDGE, 2);
```

Note: *When DSPSS is enabled, the SDADC converted data will be transferred to the data memory (XMEM) of DSPSS aligned to 16-bit. The data for the eTPU resolver should be obtained from DSPSS RAM.*

4.3.6 SGEN configuration

S32K396 has two SGEN modules, which can supply a high-quality sinusoidal voltage signal. It can be programmed with the desired oscillation frequency and amplitude voltage. A wide frequency range (1 kHz to 50 kHz) is easily programmable through a simple register interface. The linearity and noise performance are carefully optimized through digital processing. For more details, see the S32K39, S32K37, and S32K36 Reference Manual ref.[4].

The sine wave targets as the excitation signal for resolver to replace the PWM signal as excitation. In this way, no more tuning circuits are needed and the cost can be decreased. The output signal wave is configured to 10 kHz with a maximum amplitude of 2 V. The trigger signal is from the eTPU, which is routed by the TRGMUX module. The SGEN peripheral does not have the RTD module. The example code below shows the configuration of SGEN in the EV-INVERTERGEN3 application:

```
void SGEN_Init(uint32_t freq, uint16_t amplitude, uint8_t trig)
{
    IP_SGEN_1->CTRL |= D_IP_SWG_SYN_CTRL_IOAMPL(amplitude);
    IP_SGEN_1->CTRL &= ~D_IP_SWG_SYN_CTRL_LDOS_MASK;
    IP_SGEN_1->CTRL |= D_IP_SWG_SYN_CTRL_IOFREQ((freq*16777216)/(INPUTFREQUENCY*1000));
    IP_SGEN_1->CTRL |= D_IP_SWG_SYN_CTRL_TRIG_MODE(trig);
    IP_SGEN_1->CTRL |= D_IP_SWG_SYN_CTRL_WINDOW(3);

    IP_SGEN_1->CTRL |=D_IP_SWG_SYN_CTRL_LDOS(1);
    IP_SGEN_1->CTRL &= ~D_IP_SWG_SYN_CTRL_LDOS_MASK;

    /* Enable output, start SGEN */
    IP_SGEN_1->CTRL &=~D_IP_SWG_SYN_CTRL_PDS_MASK;
}
```

4.3.7 DSP configuration

DSPSS is a signal processing block as shown in Figure 22. This block is built to implement FIR filtering capability for motor resolver application.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

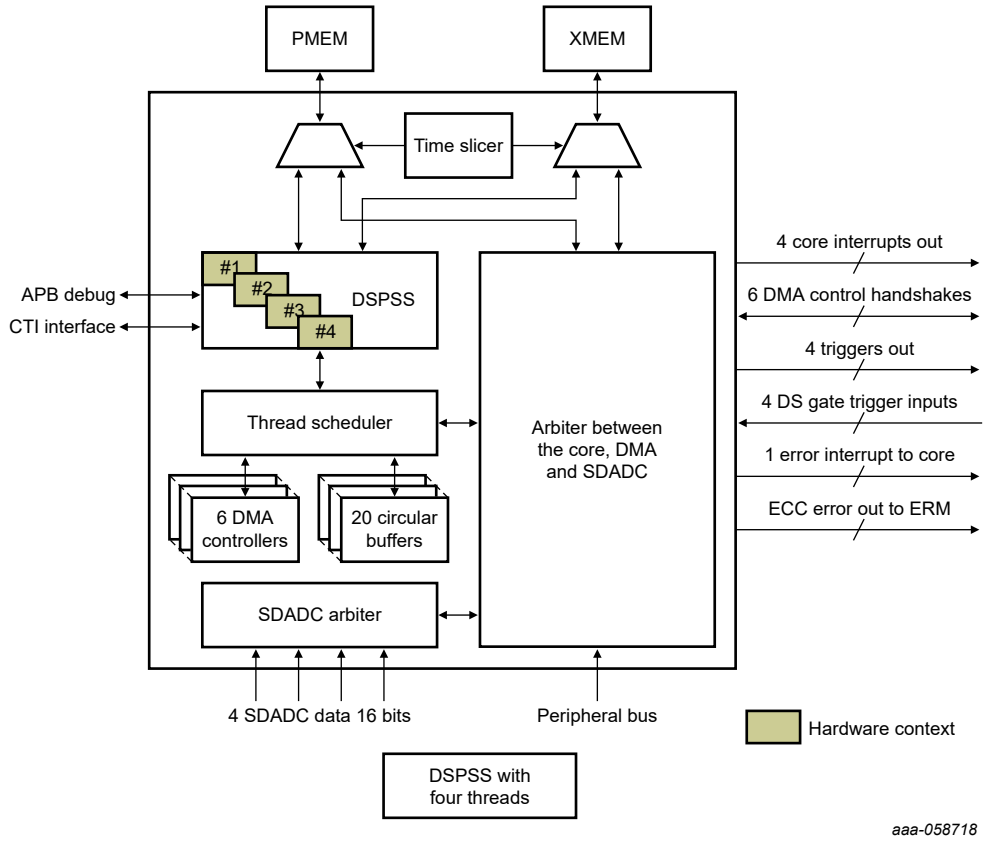


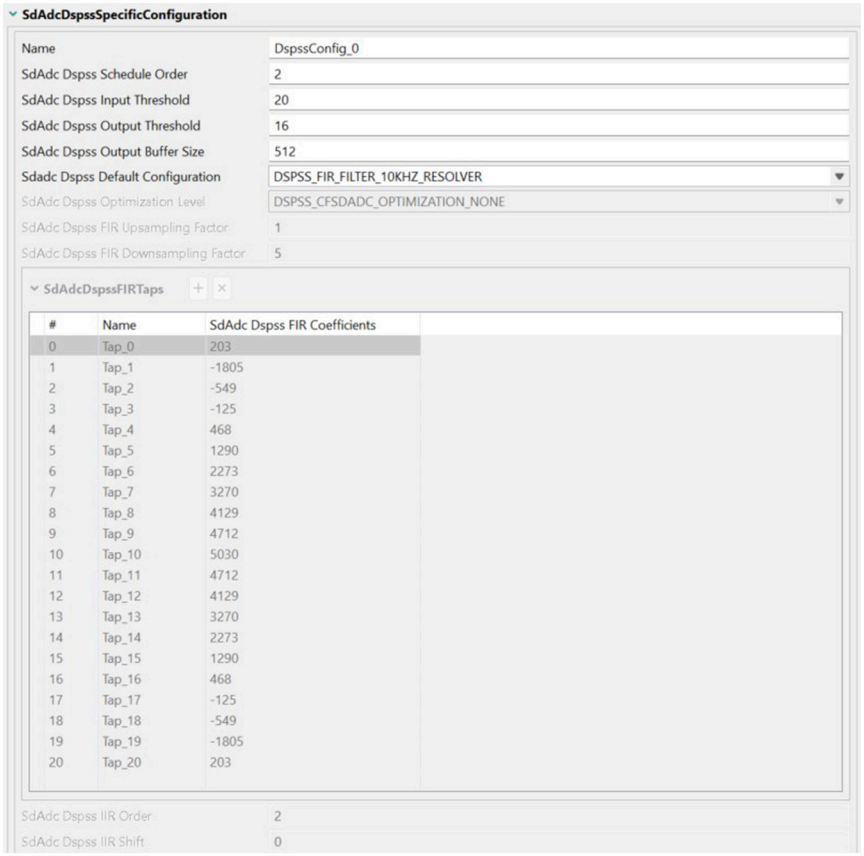
Figure 22. Block diagram of DSPSS

CoolFlux DSP is the main processing block of this subsystem, which provides multithreaded processing capability (up to four threads). The subsystem has dedicated program and data memories for parallel access. This block processes samples obtained from the SDADC interface and then keeps the processed data in a local SRAM, which is then accessed by core or DMA. Figure 22 shows the diagram of the DSPSS. For details, see the S32K39, S32K37, and S32K36 Reference Manual ref.[4].

Figure 23 shows the configuration of DSPSS with the coefficients of the FIR filter. The DSPSS input threshold defines the input block size that the thread processes when data is available in the input buffer. The DSPSS output threshold defines the number of processed samples in the output buffer needed to trigger output data transfer, such as triggering DMA to transfer data.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



aaa-060343

Figure 23. DSPSS configuration

Next, configure each used thread using thread descriptors and set the start address and length for the output buffer. The DMA can transfer the processed data from XMEM to the eTPU DATA RAM for the resolver.

There are 4 threads in the DSPSS, and only the two threads are used in this project. The configuration parameters for them are the same. After the thread configuration, the next step is configuring the DMA for transferring processed sine and cosine signals to eTPU DATA RAM. Then, deassert DSPSS reset to initialize the threads until the initialization has been finished and start the DSPSS execution by calling *DSPSS_Start()*.

```
/* Sine and cosine */
 DSPSS_CFSDADC_ThreadDescriptorType *pThreadDescriptorsArrStart =
 (DSPSS_CFSDADC_ThreadDescriptorType*)XMEM_0_BASEADDRESS;
 const uint8 sinThreadIdx = 2U;
 const uint8 cosThreadIdx = 3U;
 DSPSS_CFSDADC_ThreadDescriptorType *pThDescriptorSin = &pThreadDescriptorsArrStart[sinThreadIdx];
 DSPSS_CFSDADC_ThreadDescriptorType *pThDescriptorCos = &pThreadDescriptorsArrStart[cosThreadIdx];

 /* DMA CoolFlux -> eTPU - sine signal */
 Dma_Ip_LogicChannelTransferListType sin_dma[2] =
 {
         {DMA_IP_CH_SET_SOURCE_ADDRESS,      (uint32)pThDescriptorSin->BufferOutputStart +
 XMEM_0_BASEADDRESS},
         {DMA_IP_CH_SET_DESTINATION_ADDRESS,
 (uint32)Etpu_Resolver_Ip_Instance_BOARD_INITPERIPHERALS_I0.pSignalsPba},
 };

 /* DMA CoolFlux -> eTPU - cosine signal */
 Dma_Ip_LogicChannelTransferListType cos_dma[2] =
 {
         {DMA_IP_CH_SET_SOURCE_ADDRESS,      (uint32)pThDescriptorCos->BufferOutputStart +
 XMEM_0_BASEADDRESS},
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
        {DMA_IP_CH_SET_DESTINATION_ADDRESS,
 (uint32)Etpu_Resolver_Ip_Instance_BOARD_INITPERIPHERALS_I0.pSignalsPba +
 RES_SIGNAL_ADDRESS_OFFSET },
 };

 /* DMA for transferring resolver ATO trigger (write to eTPU HSRR) */
 Dma_Ip_LogicChannelTransferListType hrs_dma[2] =
     {
         {DMA_IP_CH_SET_SOURCE_ADDRESS,      (uint32)hrs},
         {DMA_IP_CH_SET_DESTINATION_ADDRESS, (uint32_t)(uint32)&pEtpu_Ip_Regs-
>CHAN[Etpu_Resolver_Ip_Instance_BOARD_INITPERIPHERALS_I0.u8ChanNumAto].HSRR.R},
     };

 Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_1, sin_dma, 2);
 Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_2, cos_dma, 2);
 Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_3, hrs_dma, 2);
 Dma_Ip_SetLogicChannelCommand(DMA_LOGIC_CH_1,DMA_IP_CH_SET_HARDWARE_REQUEST);
 Dma_Ip_SetLogicChannelCommand(DMA_LOGIC_CH_2,DMA_IP_CH_SET_HARDWARE_REQUEST);
 Dma_Ip_SetLogicChannelCommand(DMA_LOGIC_CH_3,DMA_IP_CH_SET_HARDWARE_REQUEST);
```

4.3.8 ADC configuration

S32K396 has seven ADC instances, which have eight channels with 15-bit resolution. In this application, four ADC instances are utilized to sample simultaneously phase A, phase B and phase C currents and DC bus voltage. This is achieved by utilizing a body cross-triggering unit (BCTU) instance that triggers at the same time as ADC3 (AdcHwUnit_3), ADC4 (AdcHwUnit_4), ADC5 (AdcHwUnit_5), and ADC6 (AdcHwUnit_6). Figure 24 and Figure 25 give the RTD configuration for ADCs, taking one ADC configuration as an example. The others are almost similar to the example one.

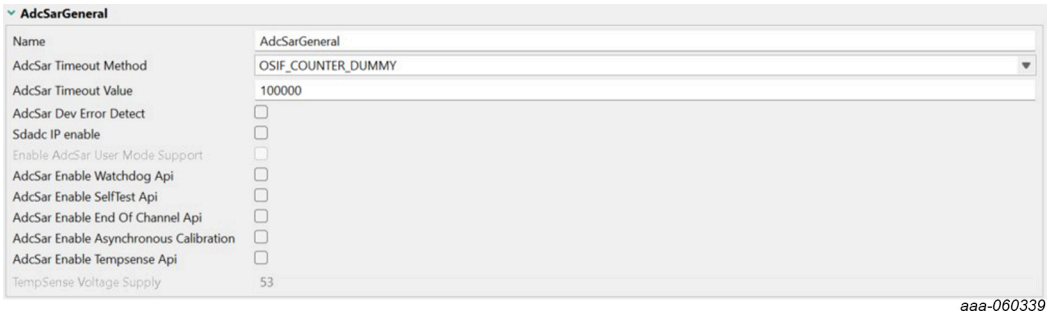


Figure 24. SARADC General configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

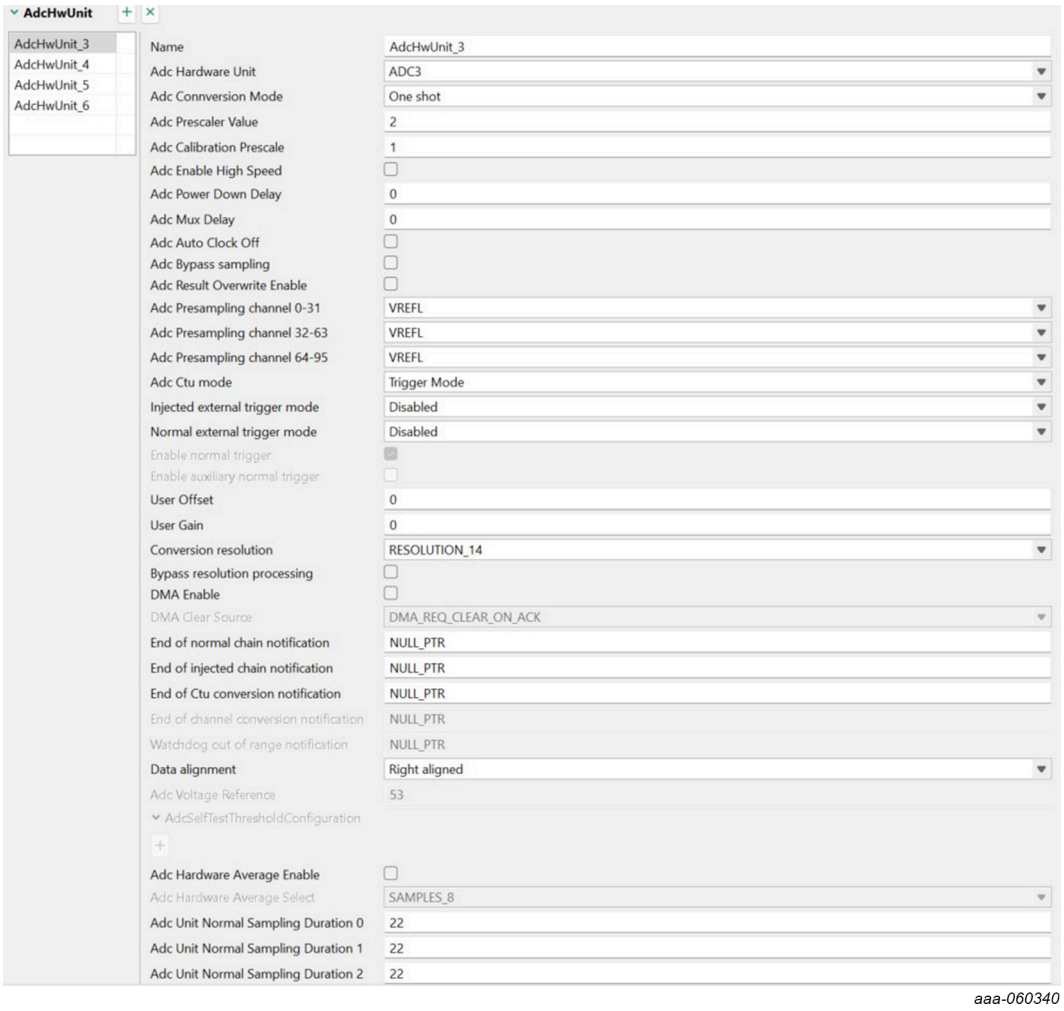


Figure 25. SARADC HW Unit configuration

4.3.9 BCTU configuration

The BCTU has 2 instances where the configured instance in the EV-INVERTERGEN3 project is BCTU_1. It can support associated ADCs that trigger simultaneously. It routes each ADC conversion result to memory through DMA. There are ways to obtain ADC conversion results such as directly accessing the ADC result register, reading the BCTU FIFO through DMA, and reading the BCTU FIFO through an interrupt. Reading the BCTU FIFO through DMA can improve the utility of the main core. Therefore, the BCTU FIFO and DMA must be configured. Figure 26 and Figure 27 show the RTD configuration for BCTU.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

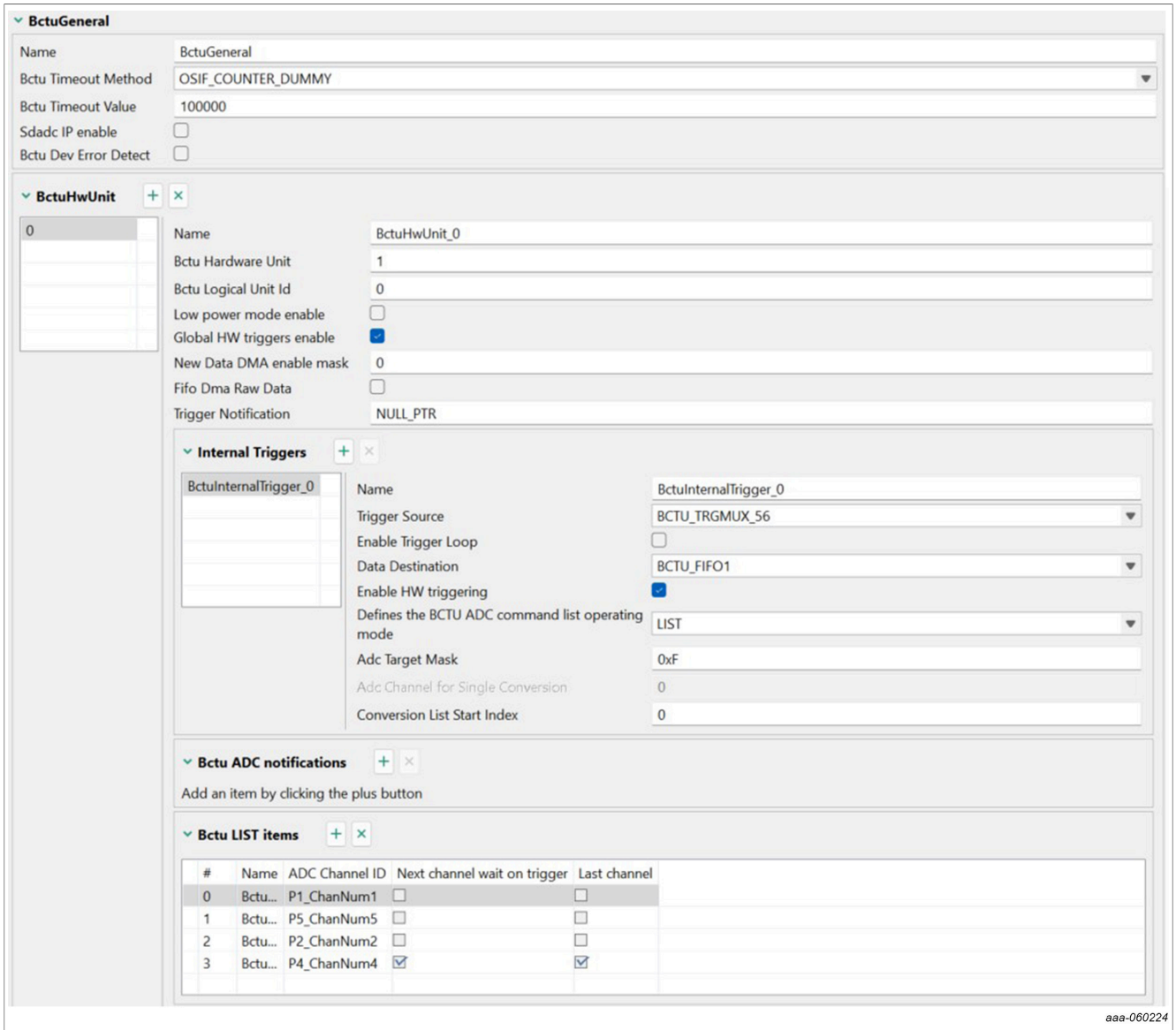


Figure 26. BCTU General configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

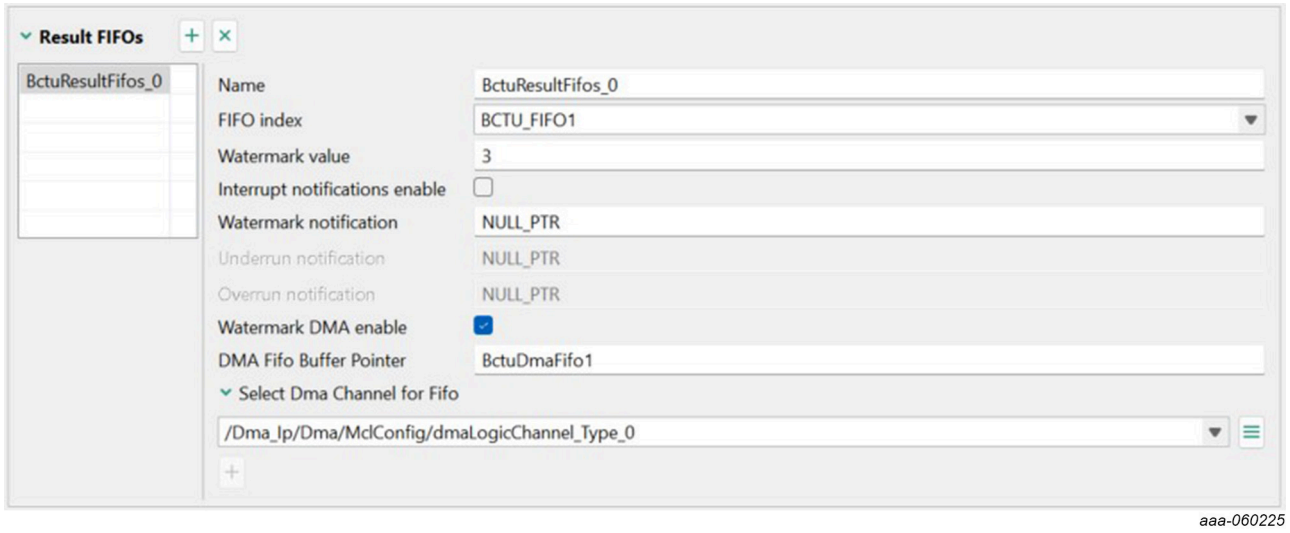


Figure 27. BCTU Results FIFOs

4.3.10 DMA configuration

In this application, seven DMA channels are used to speed up the control loop operation and offload the CPU. The first two DMA channels are used to transfer preprocessed resolver feedback sine and cosine signals from DSPSS to eTPU DATA RAM. The next four DMA channels are used to transfer the 3-phase currents and DC bus voltage measurement from ADC to the control loop. The last DMA is used for transferring resolver ATO trigger. The DMA configuration with source and destination addresses of respective signals are shown by the example code below.

```
void DMA_configuration (void)
{
 /* Set the proper destination and source addresses for the transfers */
 static volatile const uint32_t hsr[2]={2,3};

 /* BCTU */
 Dma_Ip_LogicChannelTransferListType list_bctu_dma[2] =
 {
   {DMA_IP_CH_SET_SOURCE_ADDRESS, (uint32_t)&IP_BCTU_1->FIFO1DR},
   {DMA_IP_CH_SET_DESTINATION_ADDRESS, (uint32_t)measured_currents_voltage+0x1000000},
 };

 /* Sine and cosine */
 DSPSS_CFSDADC_ThreadDescriptorType *pThreadDescriptorsArrStart =
 (DSPSS_CFSDADC_ThreadDescriptorType*)XMEM_0_BASEADDRESS;
 const uint8 sinThreadIdx = 2U;
 const uint8 cosThreadIdx = 3U;
 DSPSS_CFSDADC_ThreadDescriptorType *pThDescriptorSin = &pThreadDescriptorsArrStart[sinThreadIdx];
 DSPSS_CFSDADC_ThreadDescriptorType *pThDescriptorCos = &pThreadDescriptorsArrStart[cosThreadIdx];

 /* DMA CoolFlux -> eTPU - sine signal */
 Dma_Ip_LogicChannelTransferListType sin_dma[2] =
 {
        {DMA_IP_CH_SET_SOURCE_ADDRESS,      (uint32)pThDescriptorSin->BufferOutputStart + XMEM_0_BASEADDRESS},
        {DMA_IP_CH_SET_DESTINATION_ADDRESS,
 (uint32)Etpu_Resolver_Ip_Instance_BOARD_INITPERIPHERALS_I0.pSignalsPba},
 };

 /* DMA CoolFlux -> eTPU - cosine signal */
 Dma_Ip_LogicChannelTransferListType cos_dma[2] =
 {
        {DMA_IP_CH_SET_SOURCE_ADDRESS,      (uint32)pThDescriptorCos->BufferOutputStart + XMEM_0_BASEADDRESS},
        {DMA_IP_CH_SET_DESTINATION_ADDRESS, (uint32)Etpu_Resolver_Ip_Instance_BOARD_INITPERIPHERALS_I0.pSignalsPba
 + RES_SIGNAL_ADDRESS_OFFSET },
 };

 /* DMA for transferring resolver ATO trigger (write to eTPU HSRR) */
 Dma_Ip_LogicChannelTransferListType hsr_dma[2] =
     {
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
          {DMA_IP_CH_SET_SOURCE_ADDRESS,      (uint32)hsr},
          {DMA_IP_CH_SET_DESTINATION_ADDRESS, (uint32_t)(uint32)&pEtpu_Ip_Regs-
>CHAN[Etpu_Resolver_Ip_Instance_BOARD_INITPERIPHERALS_I0.u8ChanNumAto].HSRR.R},
      };

  Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_0, list_bctu_dma, 2);
  Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_1, sin_dma, 2);
  Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_2, cos_dma, 2);
  Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_3, hsr_dma, 2);
  Dma_Ip_SetLogicChannelCommand(DMA_LOGIC_CH_0,DMA_IP_CH_SET_HARDWARE_REQUEST);
  Dma_Ip_SetLogicChannelCommand(DMA_LOGIC_CH_1,DMA_IP_CH_SET_HARDWARE_REQUEST);
  Dma_Ip_SetLogicChannelCommand(DMA_LOGIC_CH_2,DMA_IP_CH_SET_HARDWARE_REQUEST);
  Dma_Ip_SetLogicChannelCommand(DMA_LOGIC_CH_3,DMA_IP_CH_SET_HARDWARE_REQUEST);
 }
```

Detailed configuration from the config tool RTD settings for DMA channels used for 3-phase currents, DC bus voltage measurement, and eTPU resolver function can be found in Table 4.

Table 4. DMA configuration

Items	3-phase currents and DC bus voltage signals	Sin signal	Cos signal	Linked HSR
Instance	1	1	1	1
Channel	16	17	18	19
Source address	[1]	[2]	[3]	hsr
Destination address	[4]	[5]	[6]	[7]
Source transfer size	2 bytes	2 bytes	2 bytes	4 bytes
Destination transfer size	2 bytes	2 bytes	2 bytes	4 bytes
Source signed offset	0 bytes	2 bytes	2 bytes	4 bytes
Destination signed offset	2 bytes	4 bytes	4 bytes	0 bytes
Transfer minor loop size	8 bytes	32 bytes	32 bytes	4 bytes
Transfer major loop count	10	2	2	2
Source last address adjustment	0 bytes	0 bytes	0 bytes	- 8 bytes
Destination last address adjustment	- 80 bytes	- 128 bytes	- 128 bytes	0 bytes
Source modulo	—	512 bytes	512 bytes	—
Destination modulo	—	—	—	—
Channel linking	Disabled	Disabled	Enabled	Disabled
Linked channel	—	—	HSR DMA channel	—

[1] &IP_BCTU_1 → FIFO1DR
[2] pThDescriptorSin → BufferOutputStart + XMEM_0_BASEADDRESS
[3] pThDescriptorCos → BufferOutputStart + XMEM_0_BASEADDRESS
[4] measured_currents_voltage +0x1000000
[5] Etpu_Resolver_Ip_Instance_BOARD_ INITPERIPHERALS_I0.pSignalsPba
[6] Etpu_Resolver_Ip_Instance_BOARD_ INITPERIPHERALS_I0.pSignalsPba + RES_SIGNAL_ADDRESS_OFFSET
[7] &pEtpu_Ip_Regs → CHAN[Etpu_Resolver _Ip_Instance_BOARD_INITPERIPHERALS_ I0.u8ChanNumAto].HSRR.R

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.3.11 eFlexPWM configuration

In the EV-INVERTERGEN3 application, the eFlexPWM1 module is used for generating the PWMs and synchronization. The eFlexPWM Submodules 0,1, and 2 are employed to generate center aligned complementary PWMs for the module. The eFlexPWM Submodule 3 is utilized to generate triggers for the 3-phase currents and DC bus voltage measurement. In this application, five triggers are generated per PWM period. These are further doubled by the LCU module to achieve oversampling of 10 per PWM period, which is set to 100 µs with the HW deadtime insertion of 1 µs. The RTD settings of the individual eFlexPWM Submodules are shown in the following figures.

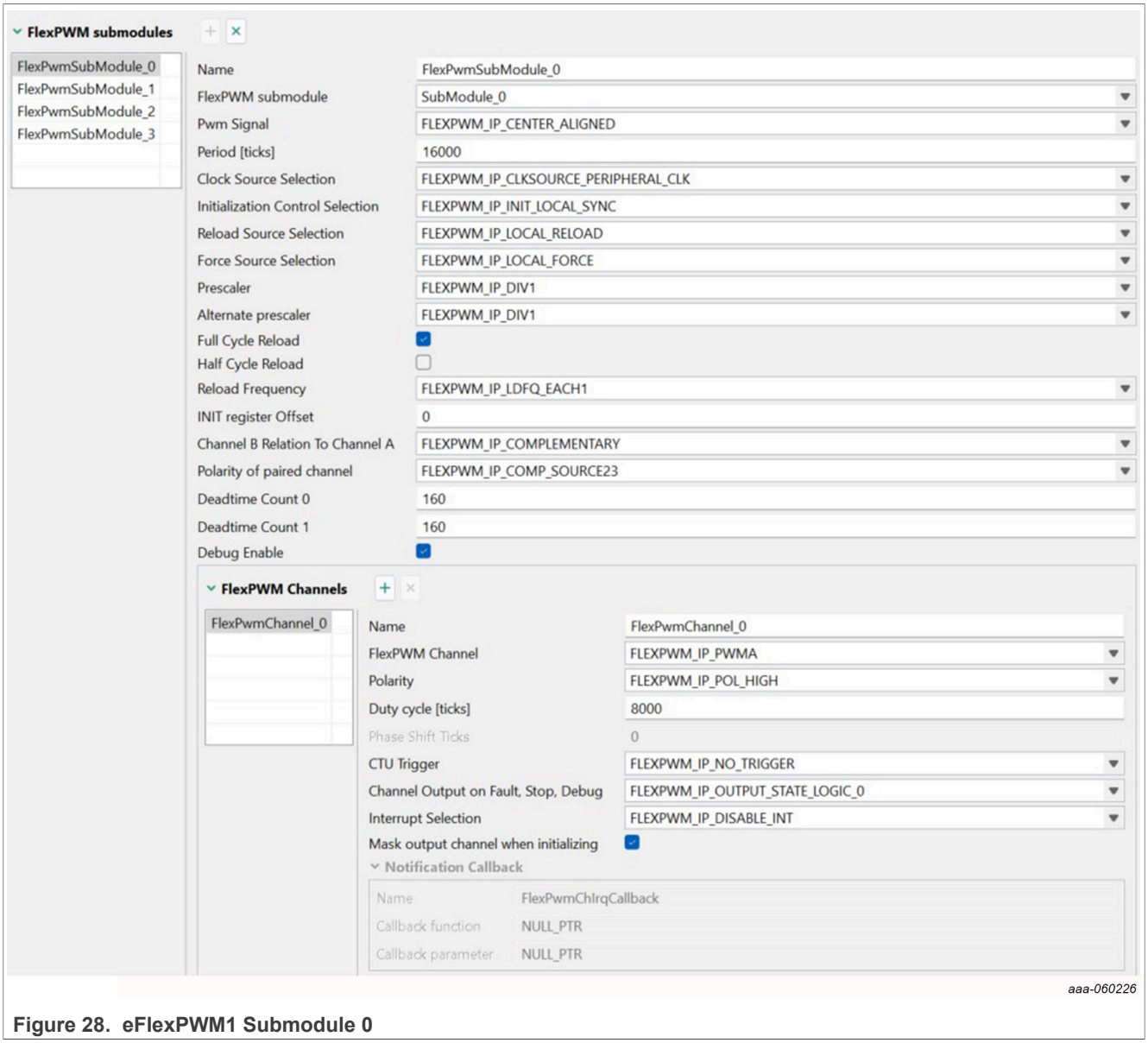


Figure 28. eFlexPWM1 Submodule 0

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

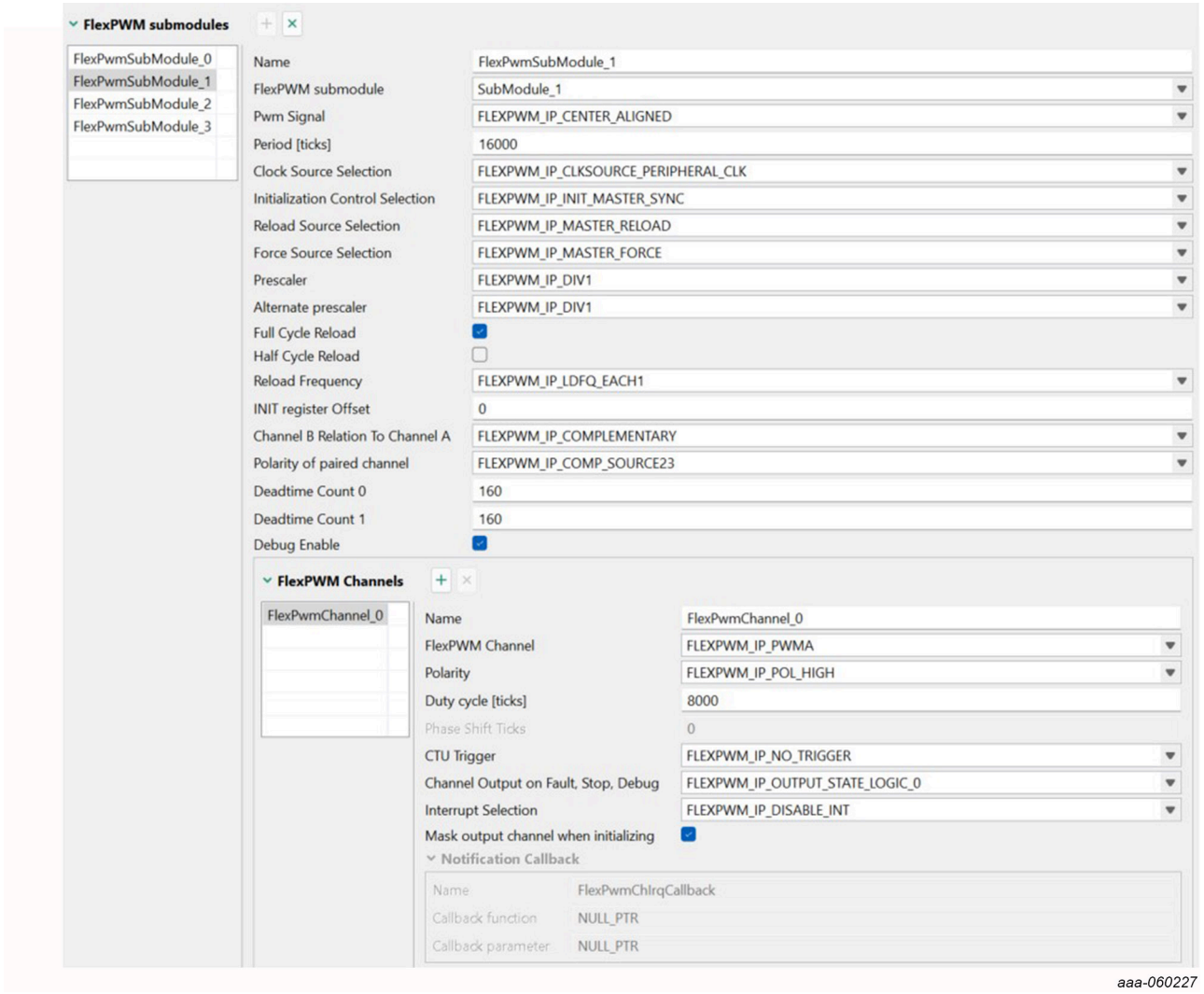


Figure 29. eFlexPWM1 Submodule 1

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

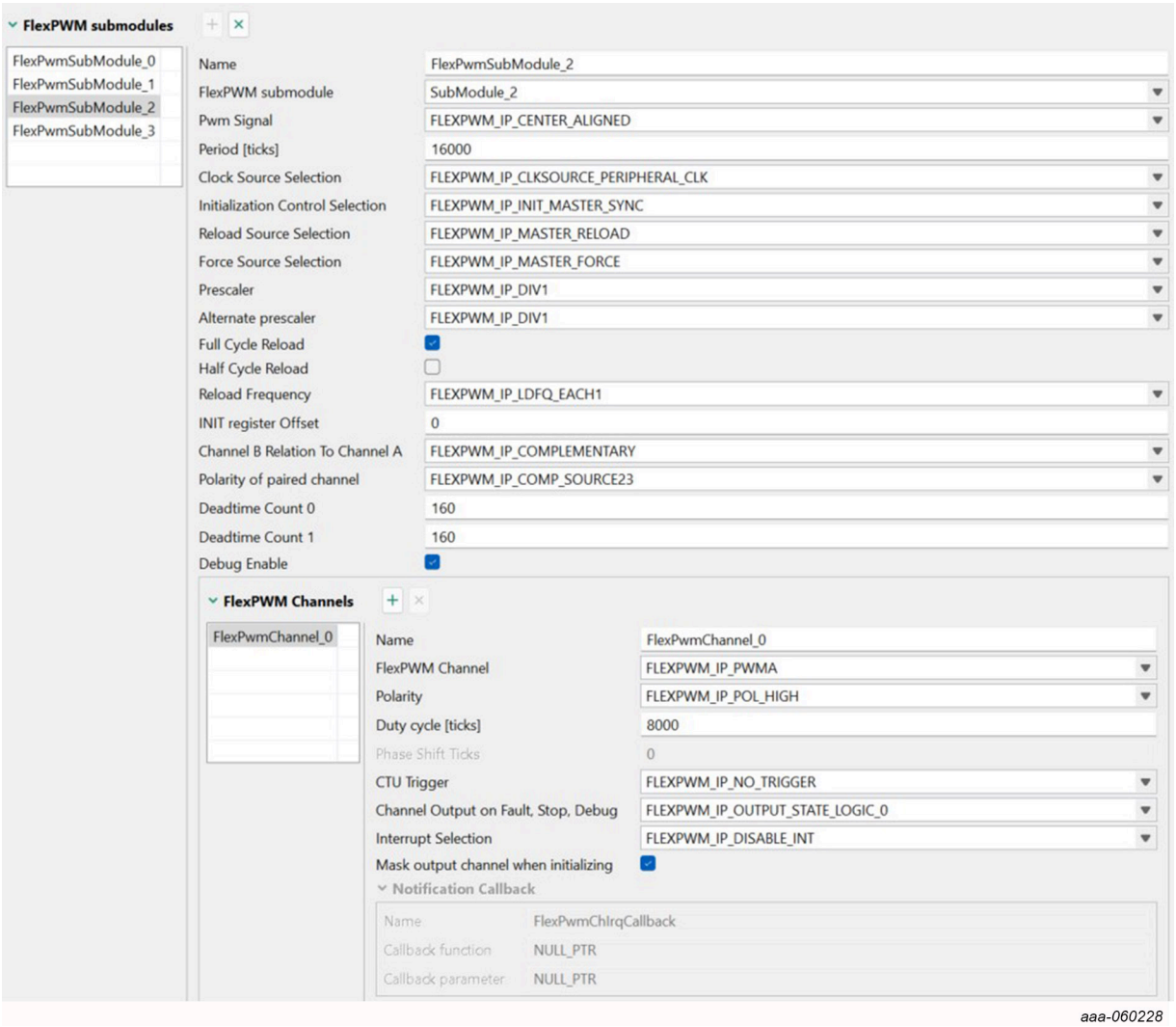


Figure 30. eFlexPWM1 Submodule 2

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

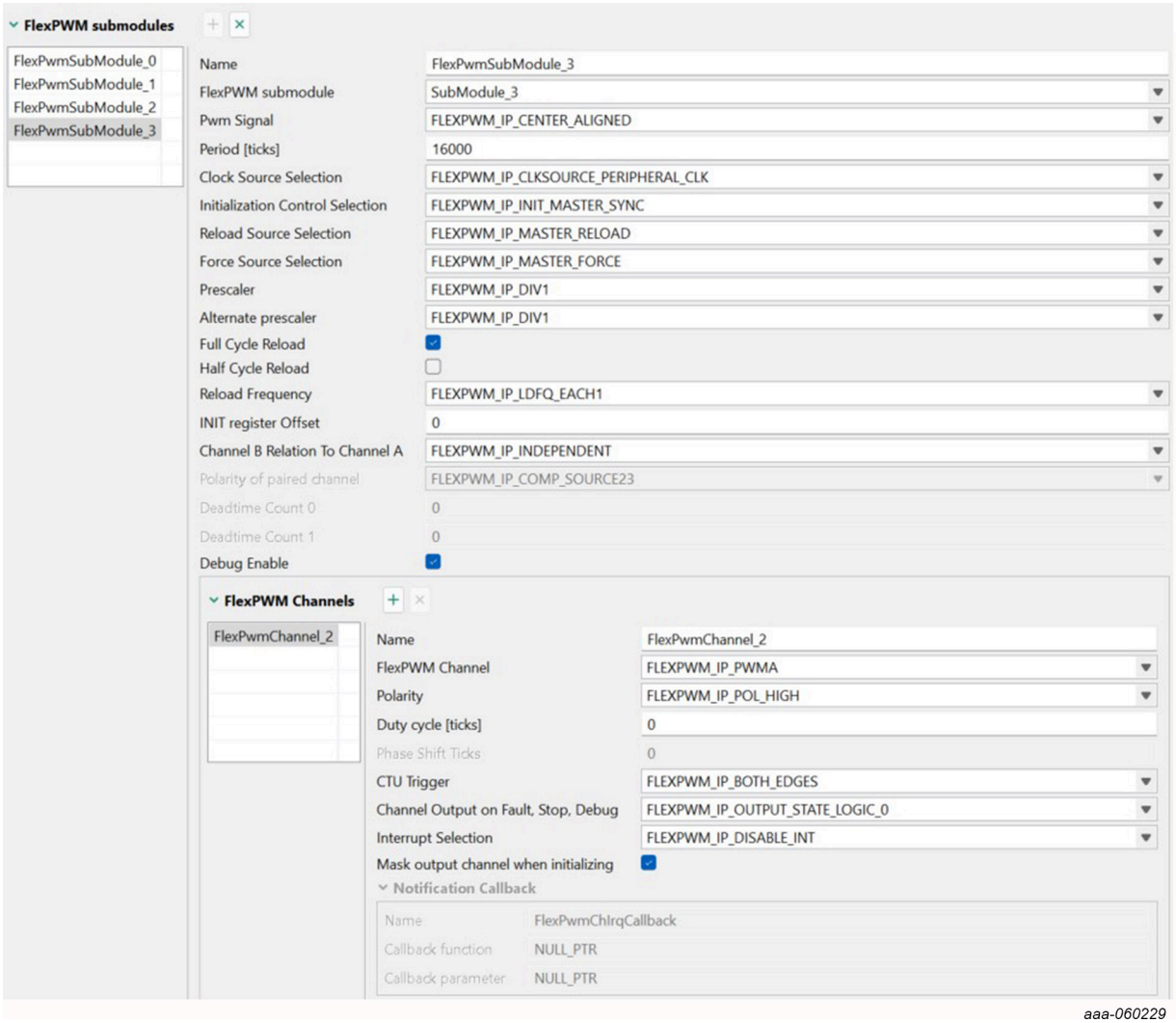


Figure 31. eFlexPWM1 Submodule 3

4.3.12 LCU configuration

The LCU is a module that can create synchronous and asynchronous logical operations of the input signals according to the internal LUT. The LUT configures the logic function of the logic cell. It delays the signals and combines them with AND/OR functions and other functions.

In the EV-INVERTERGEN3 application, the LCU is employed for the following operations:

- Creates the triggers for the 3-phase currents and DC bus voltage measurement oversampling (10 times per PWM period). This is done by the Logic Cell 0 of the LCU0.
- Extends the pulse width for triggering the eTPU extrapolation generated by the eFlexPWM val0 compare event. This is done by the part of Logic Cell 1 of the LCU0 module.

Figure 32 shows the LCU module functional configuration diagram of Logic Cell 0 and Logic Cell 1. The inputs/outputs of the corresponding logic cells are defined in the LCU RTD configuration.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

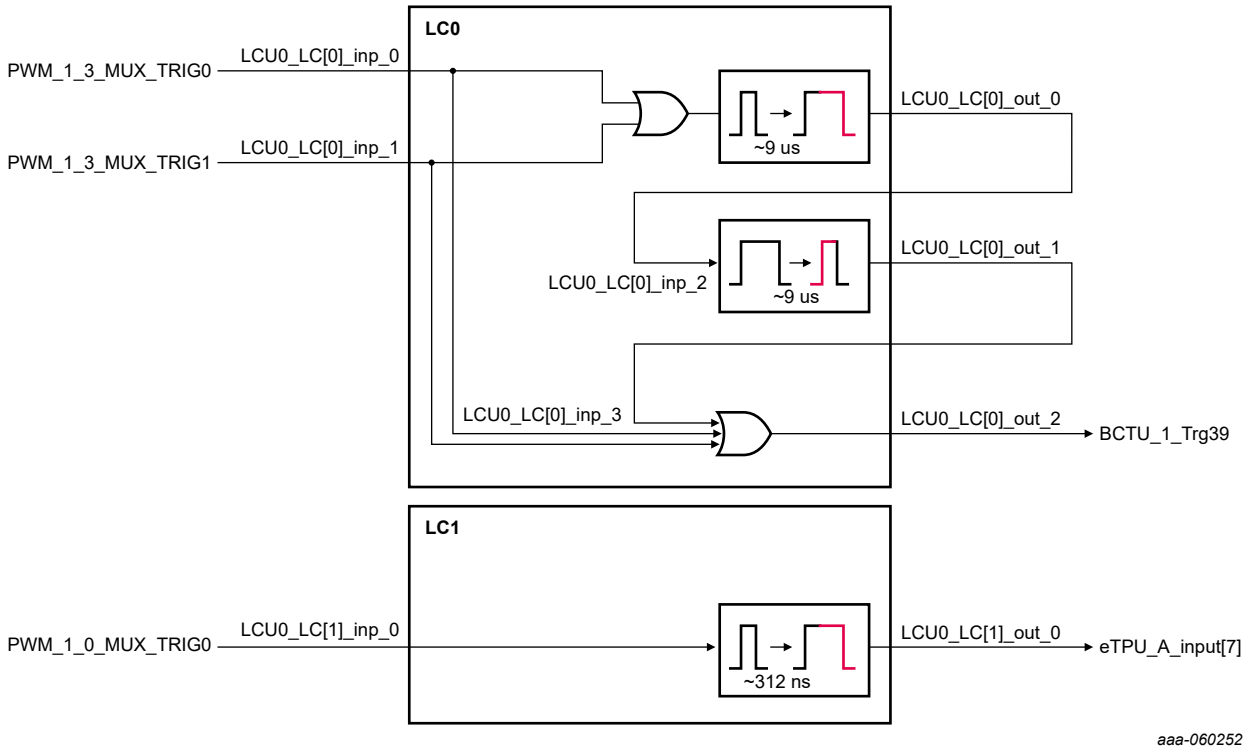


Figure 32. LCU module functional configuration diagram

Figure 33 shows the logical instances configuration where the configured LCU instance in EV-INVERTERGEN3 application is LCU0.

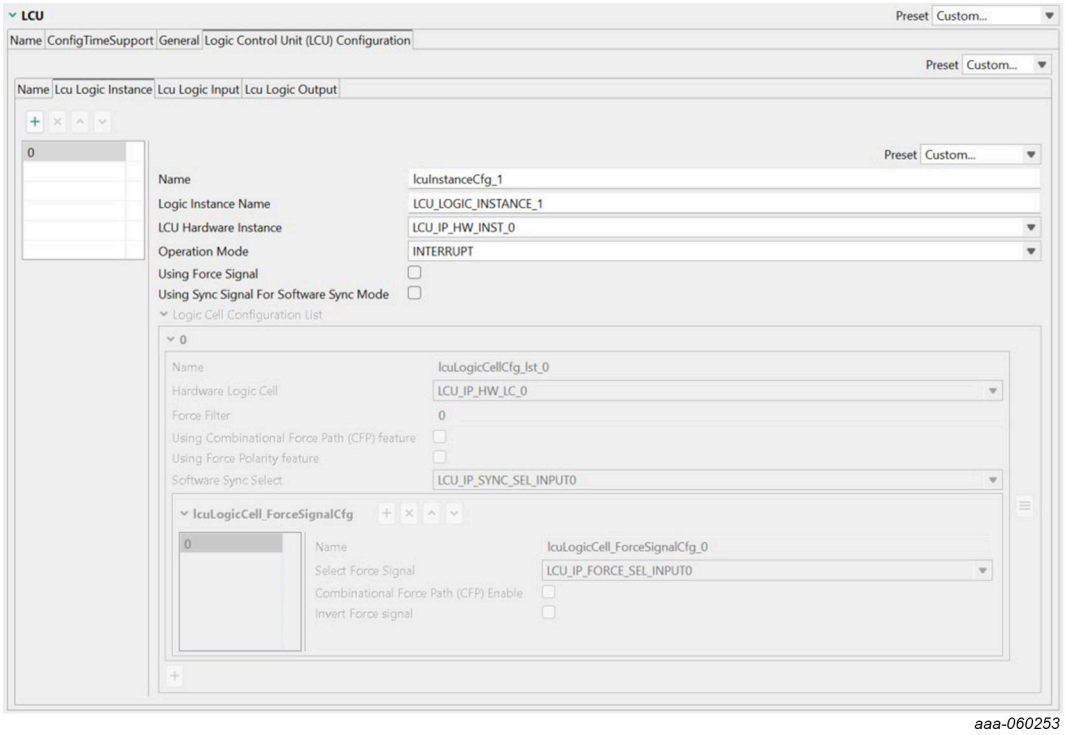


Figure 33. LCU0 Logic instances configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Figure 34, Figure 35, Figure 36, and Figure 37 define the LCU logic inputs (Input 0 - Input 3) configuration of Logic Cell 0. Figure 38 illustrates the LCU logic input (Input 0) configuration of Logic Cell 1.

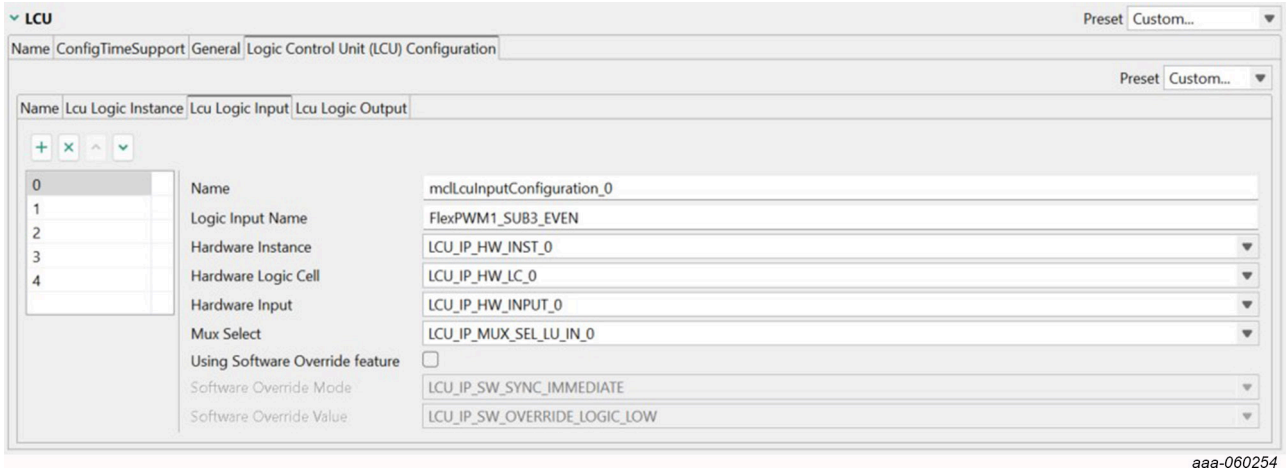


Figure 34. LCU0 Logic Cell 0 Input 0 configuration

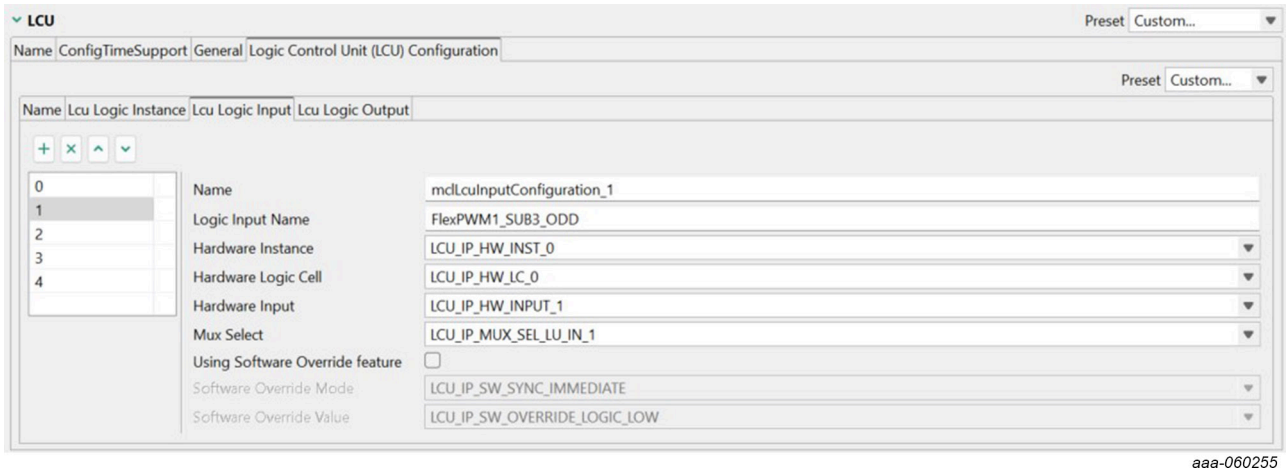


Figure 35. LCU0 Logic Cell 0 Input 1 configuration

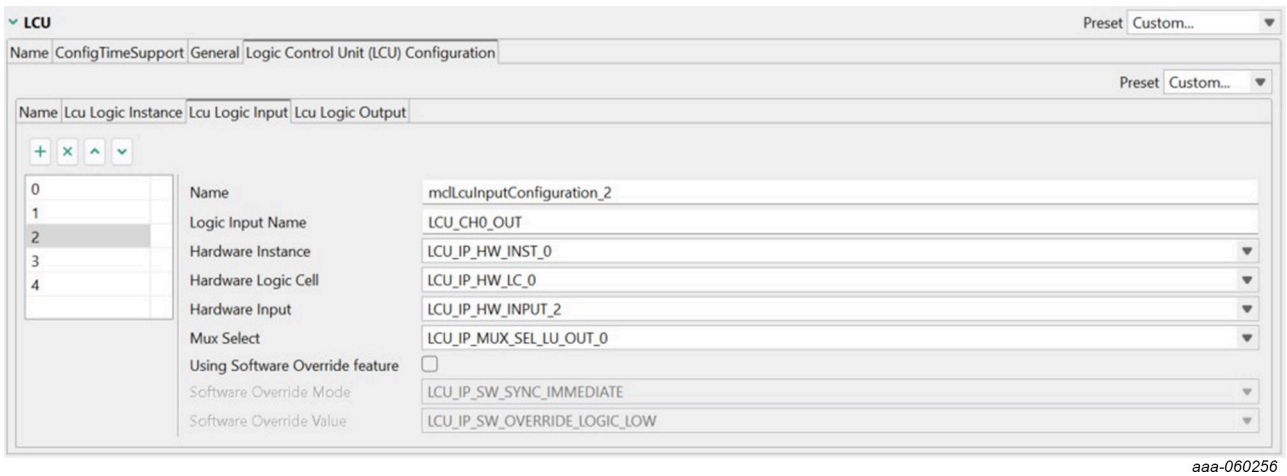


Figure 36. LCU0 Logic Cell 0 Input 2 configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

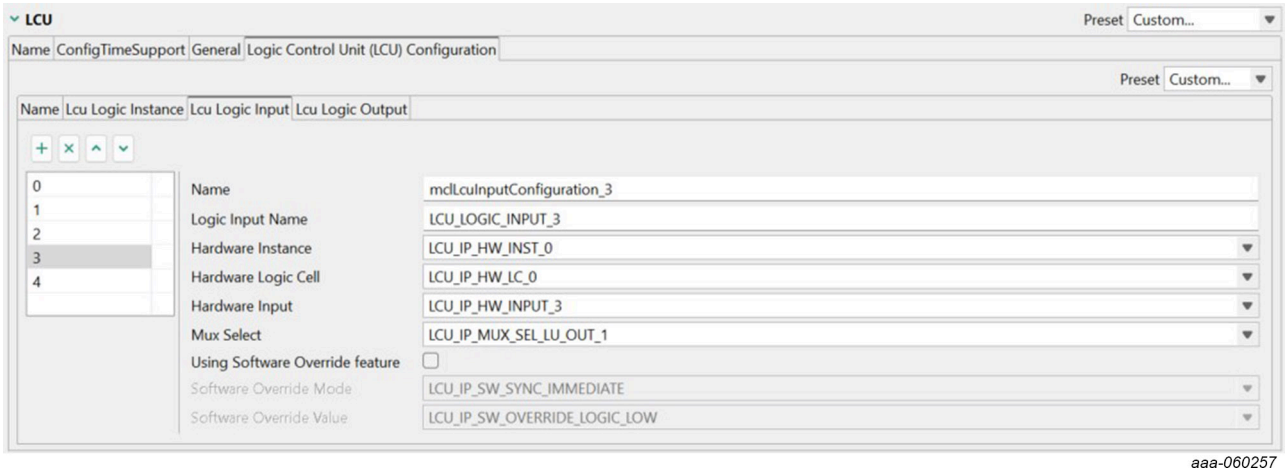


Figure 37. LCU0 Logic Cell 0 Input 3 configuration

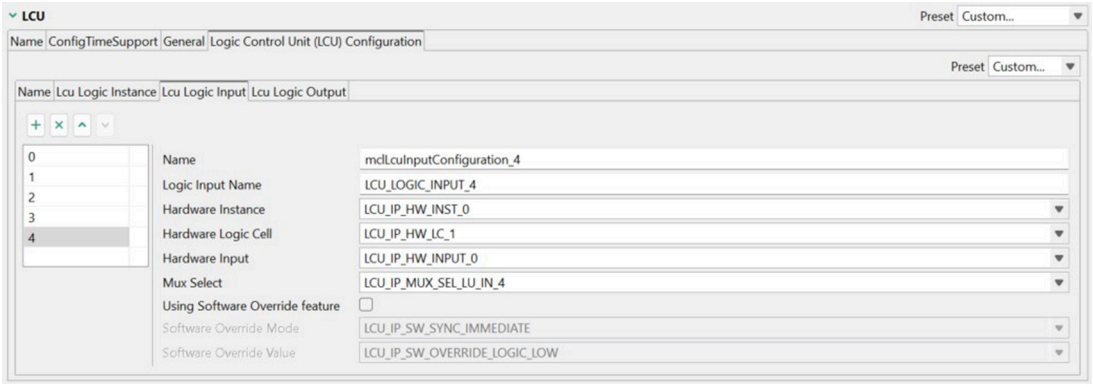
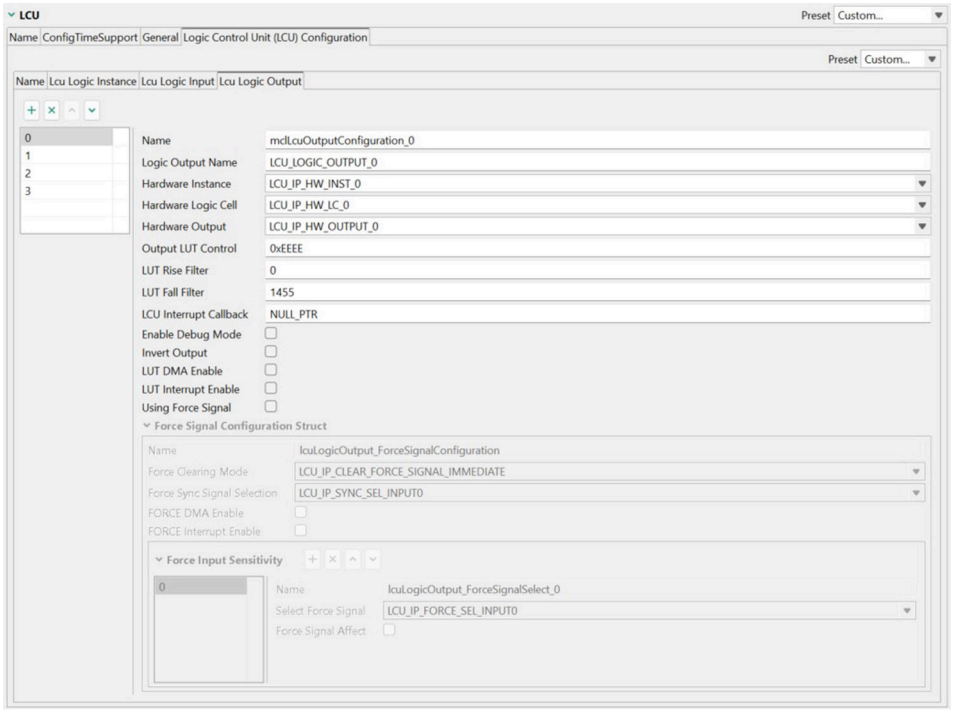


Figure 38. LCU0 Logic Cell 1 Input 0 configuration

Figure 39, Figure 40, and Figure 41 show the LCU logic outputs (Output 0 - Output 2) configuration of Logic Cell 0. Figure 42 illustrates the LCU logic output (Output 0) of Logic Cell 1.

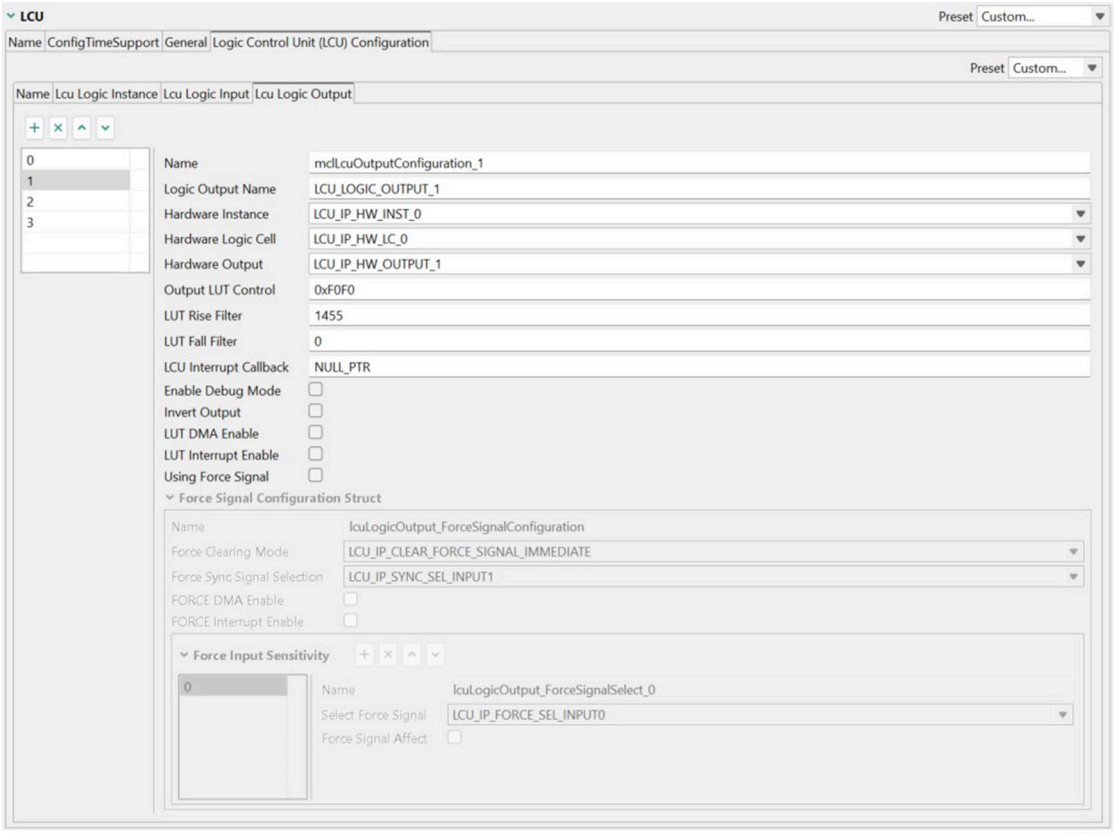
All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



aaa-060258

Figure 39. LCU0 Logic Cell 0 Output 0 configuration



aaa-060259

Figure 40. LCU0 Logic Cell 0 Output 1 configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

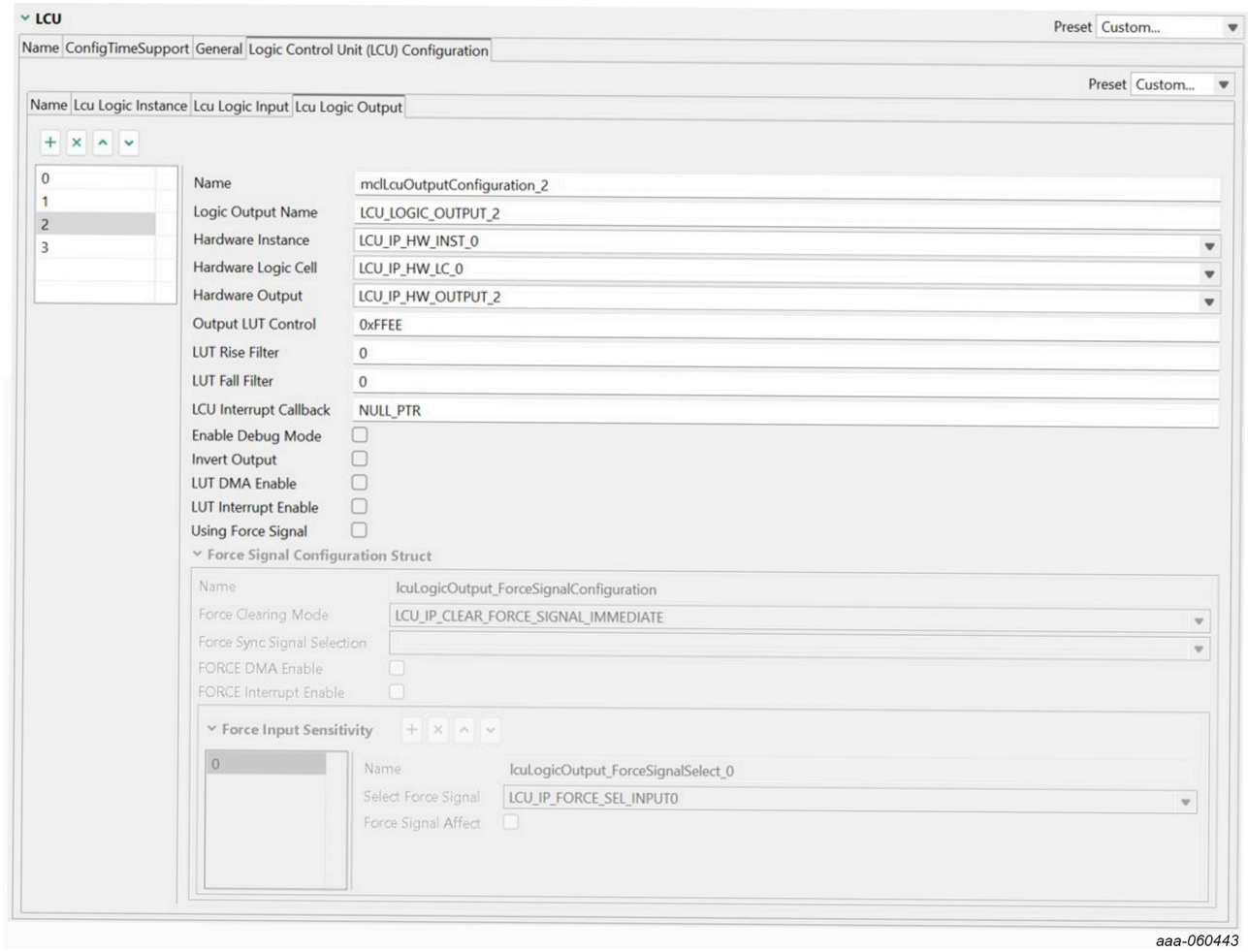


Figure 41. LCU0 Logic Cell 0 Output 2 configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

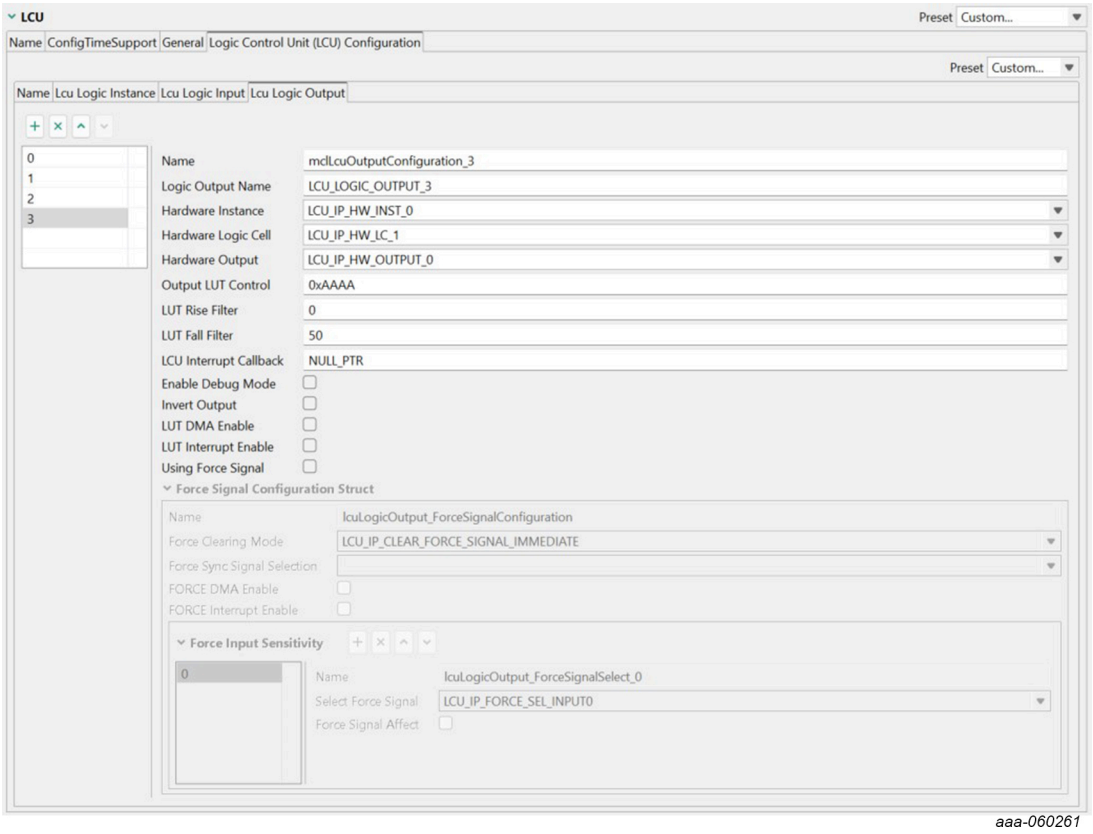


Figure 42. LCU0 Logic Cell 1 Output 0 configuration

4.3.13 TRGMUX configuration

For the variable configuration of the S32K396 device lot of internal trigger signals are configurable over the TRGMUX modules (TRGMUX_APP and TRGMUX_MSC).

In this application, the synchronous measurement of the motor phase currents, DC bus voltage, and the position are necessary. As the source of synchronization, the eFlexPWM1 module is used. The eFlexPWM1 submodule 3 in conjunction with the LCU is used to sample the motor phase currents and DC bus voltage 10 times per PWM period (TRGMUX configuration 0 and 1). The RTD configuration of TRGMUX 0 and TRGMUX 1 is shown in Figure 43 and Figure 44.

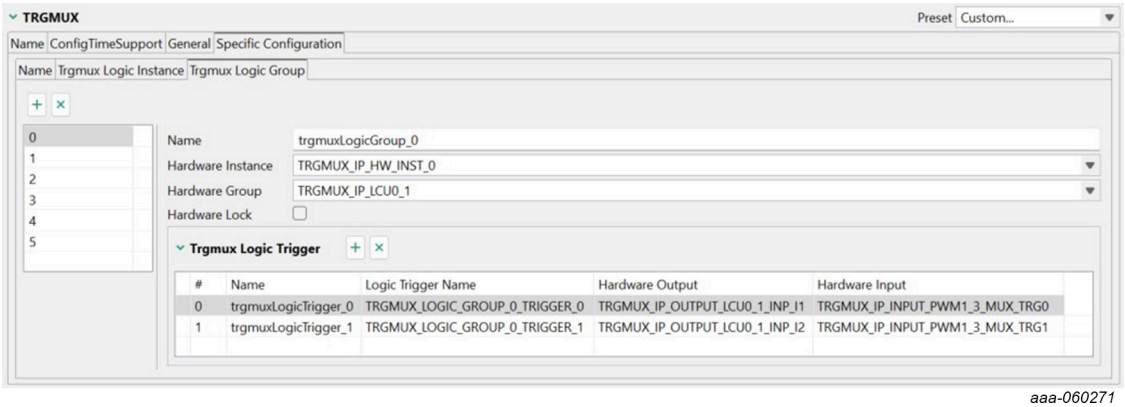


Figure 43. TRGMUX 0 configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

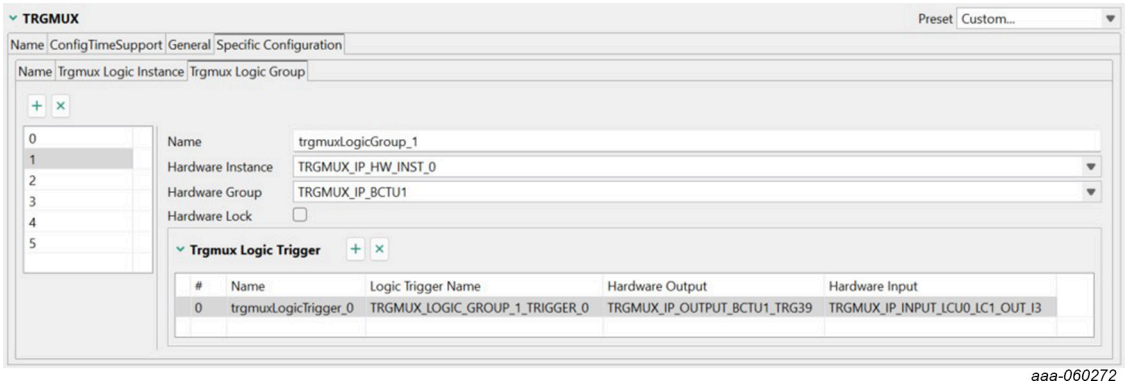


Figure 44. TRGMUX 1 configuration

The position extrapolation is triggered by the eFlexPWM1 submodule 0 (TRGMUX configuration 2 and 5). The RTD configuration of TRGMUX 2 and TRGMUX 5 is shown in Figure 45 and Figure 46.

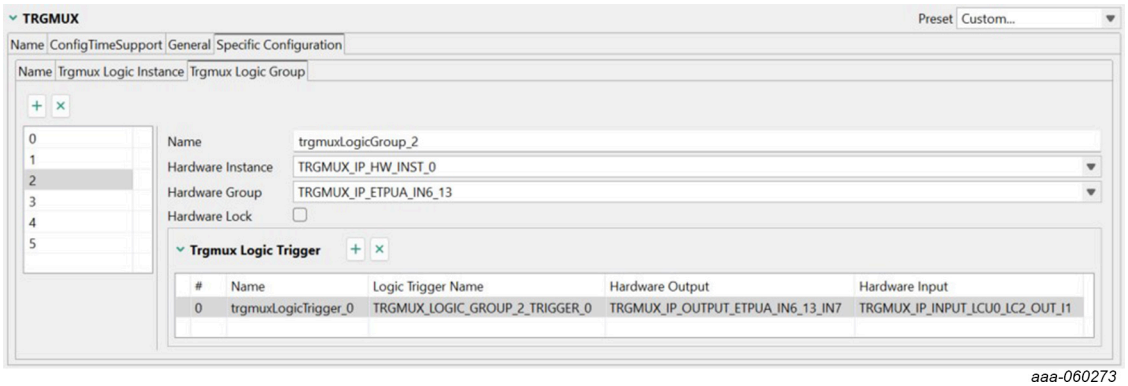


Figure 45. TRGMUX 2 configuration

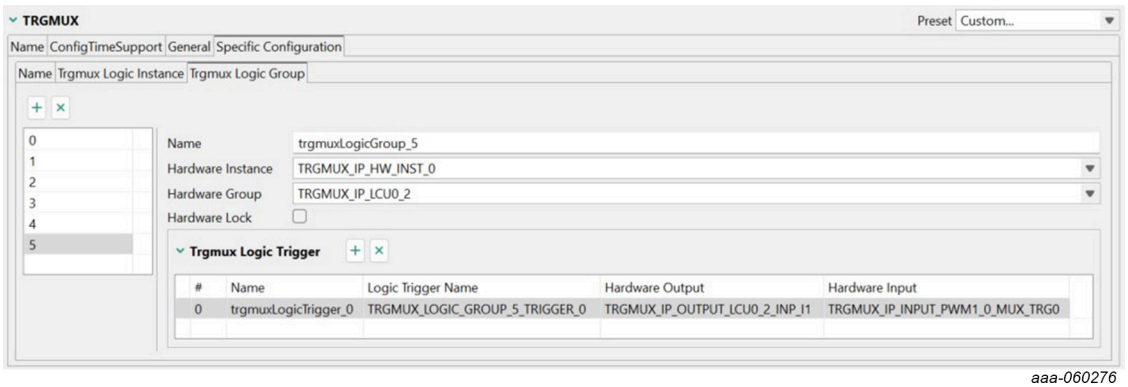


Figure 46. TRGMUX 5 configuration

The last necessary routing is the extrapolation signal phase synchronization, which is driven by the eTPU (TRGMUX configuration 3 and 4). The RTD configuration of TRGMUX 3 and TRGMUX 4 is shown in Figure 47 and Figure 48.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

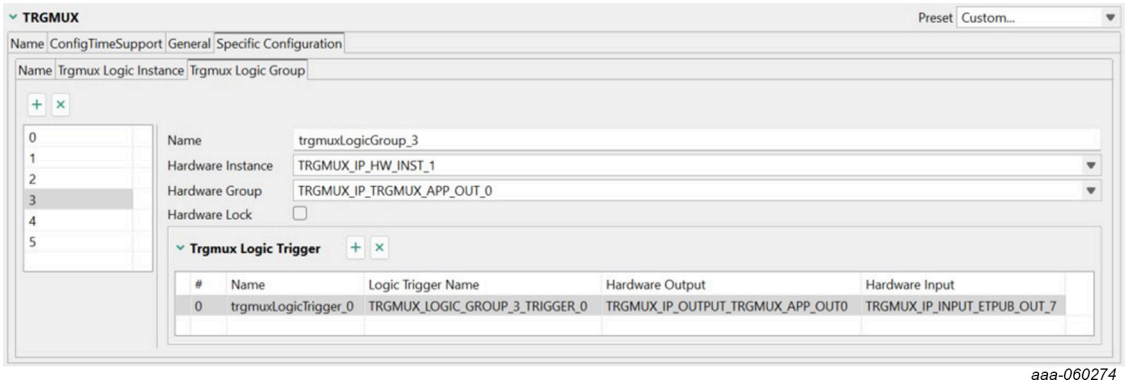


Figure 47. TRGMUX 3 configuration

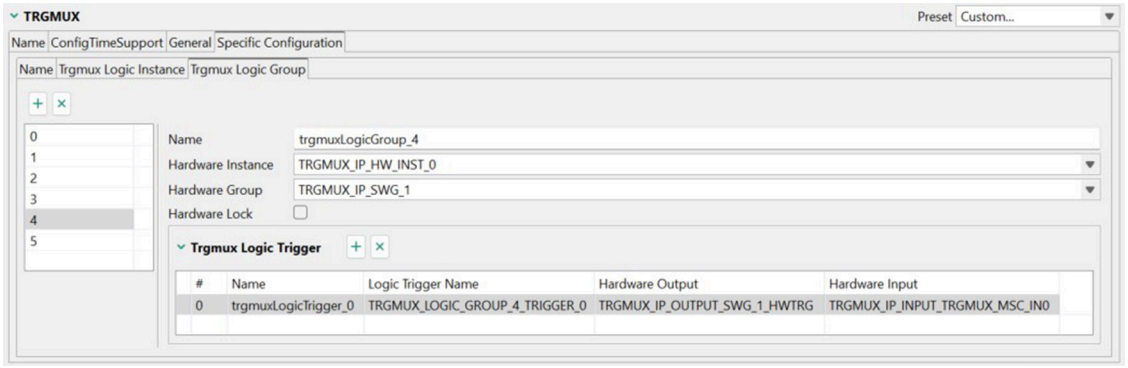


Figure 48. TRGMUX 4 configuration

A summary of TRGMUX modules configuration including TRGMUX output, selected trigger, functionality, and configuration number is shown in Table 5.

Table 5. TRGMUX configuration

TRGMUX	TRGMUX_APP Output	Selected trigger	Functionality	Configuration number
APP	LCU_0_lc1_inp_i1	PWM_1_3_MUX_TRIG0	Routing the triggers for the currents and DC bus voltage to the LCU for multiplication	0
APP	LCU_0_lc1_inp_i2	PWM_1_3_MUX_TRIG1		0
APP	BCTU_1_Trg39	LCU_0_LC1_out_i3	Currents and DC bus voltage conversion trigger	1
APP	eTPU_A_input[7]	LCU_0_LC2_out_i1	Extrapolation trigger	2
APP	LCU_0_lc2_inp_i1	PWM_1_0_MUX_TRIG0		5
MSC	TRGMUX_APP_ TRGMUX_OUT0	eTPU_B_output[7]	Resolver excitation signal phase shift	3
APP	SWG_1_Hardware trigger	TRGMUX_MSC_ TRGMUX_IN0		4

4.3.14 LPSPI configuration

LPSPI provides an efficient interface to an SPI bus, which is a synchronous serial communication interface used in embedded systems. This interface is typically used to perform short distance communications between MCUs and peripheral devices.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

In this application, LPSPI is used to perform communication for GD3162 LS, GD3162 HS, and FS26. The RTD configuration of LPSPI for these features is shown in the following figures. Figure 49, Figure 50, and Figure 51 show the LPSPI driver configuration of GD3162 LS, GD3162 HS, and FS26. Figure 52, Figure 53, and Figure 54 define the LPSPI general configuration of FS26, GD3162 HS, and GD3162 LS.

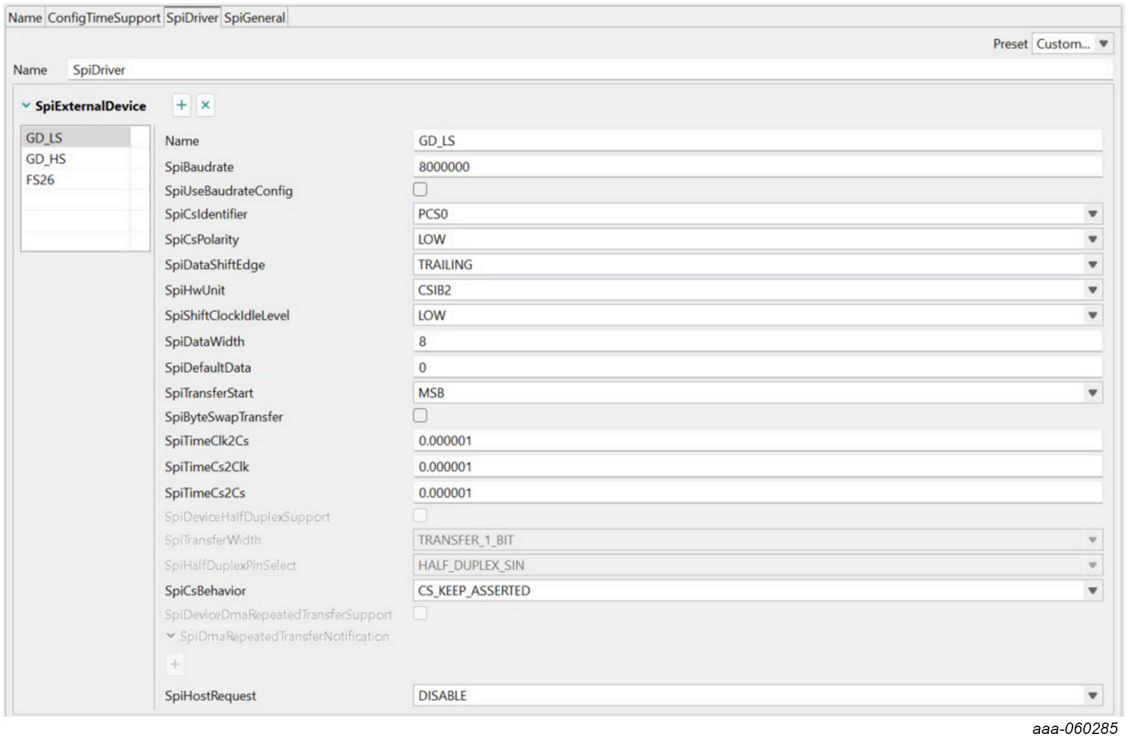
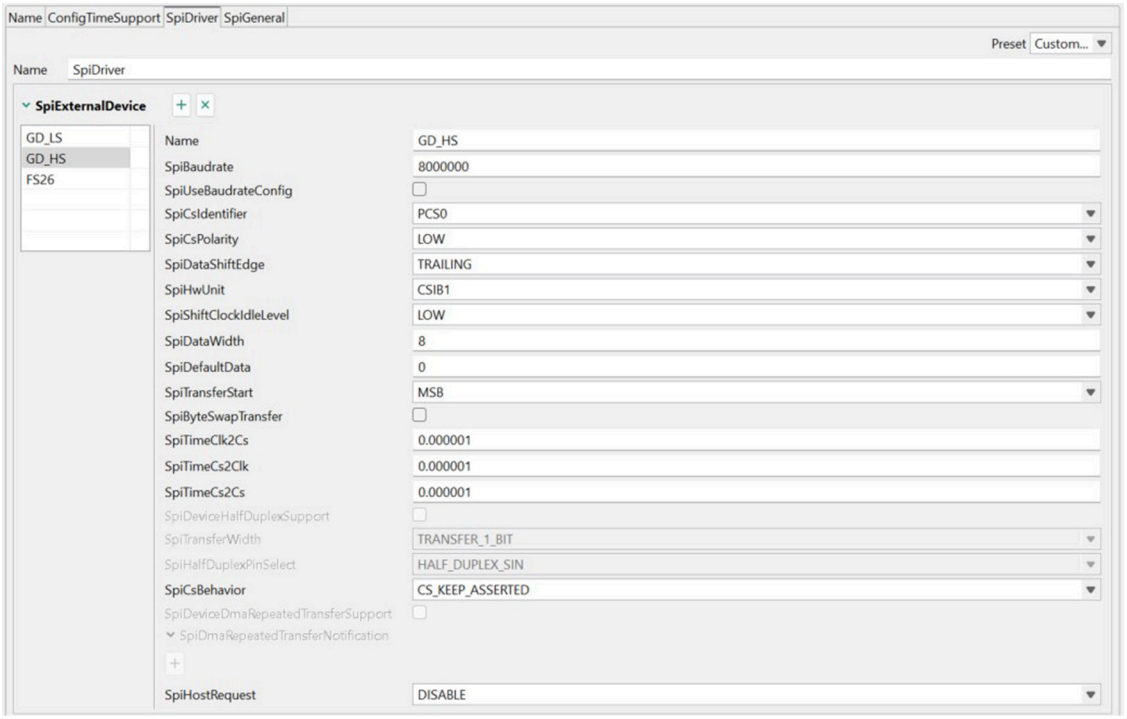


Figure 49. LPSPI Driver GD3162 LS configuration

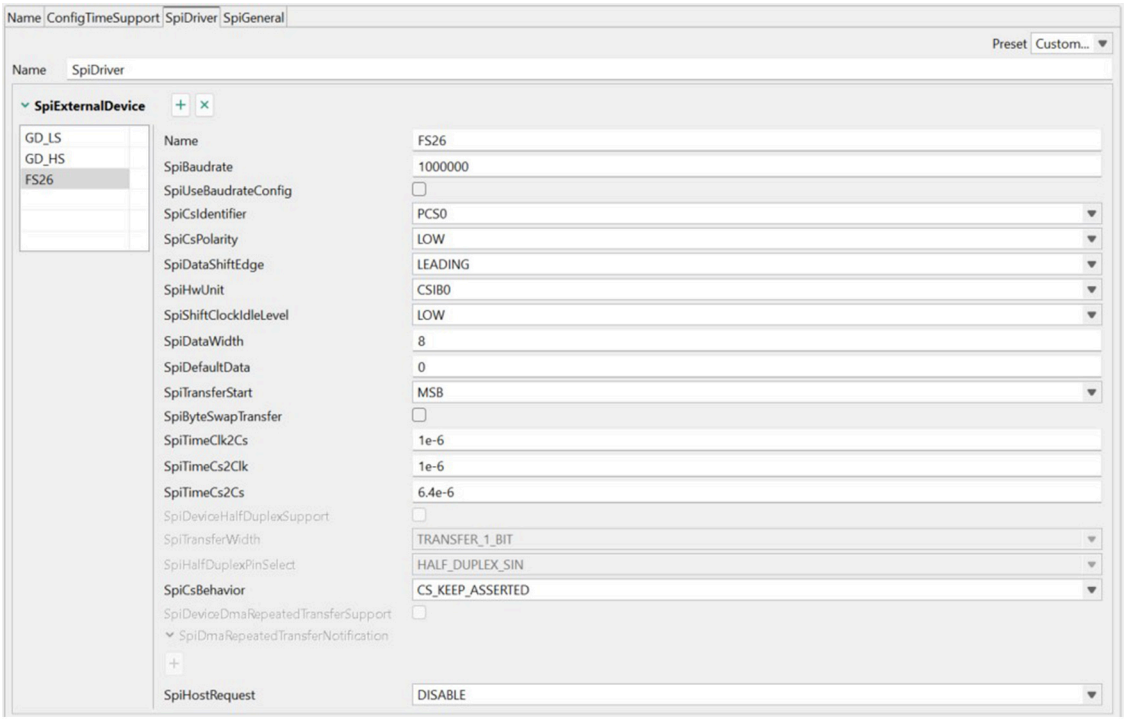
All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



aaa-060286

Figure 50. LPSPI Driver GD3162 HS configuration

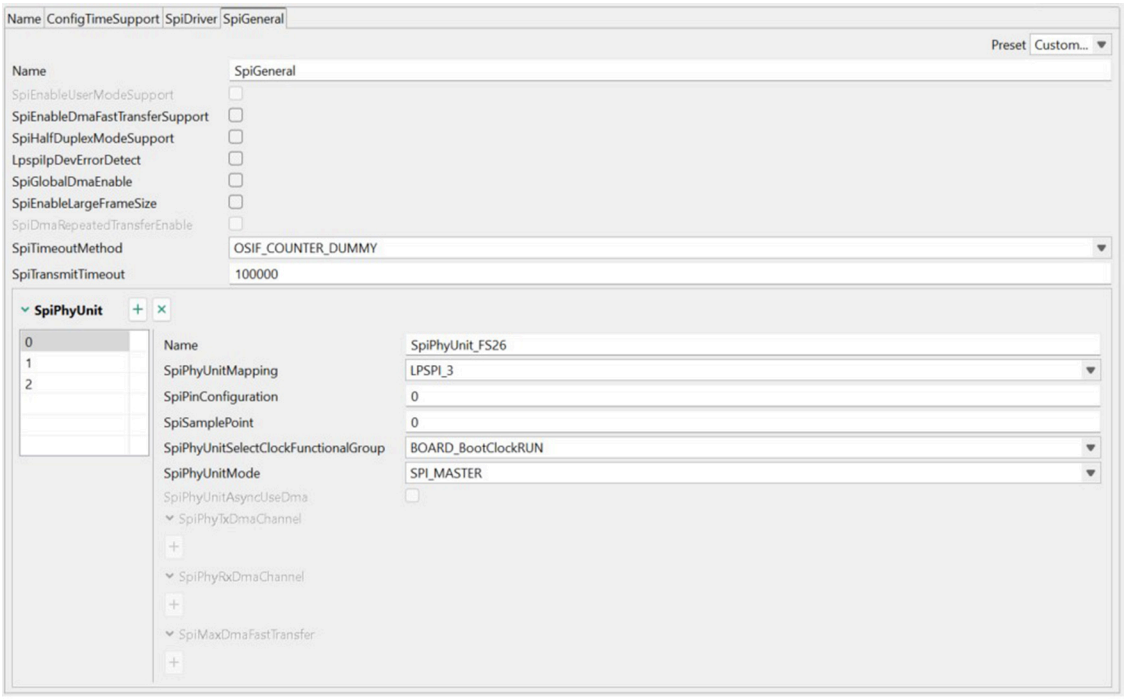


aaa-060287

Figure 51. LPSPI Driver FS26 configuration

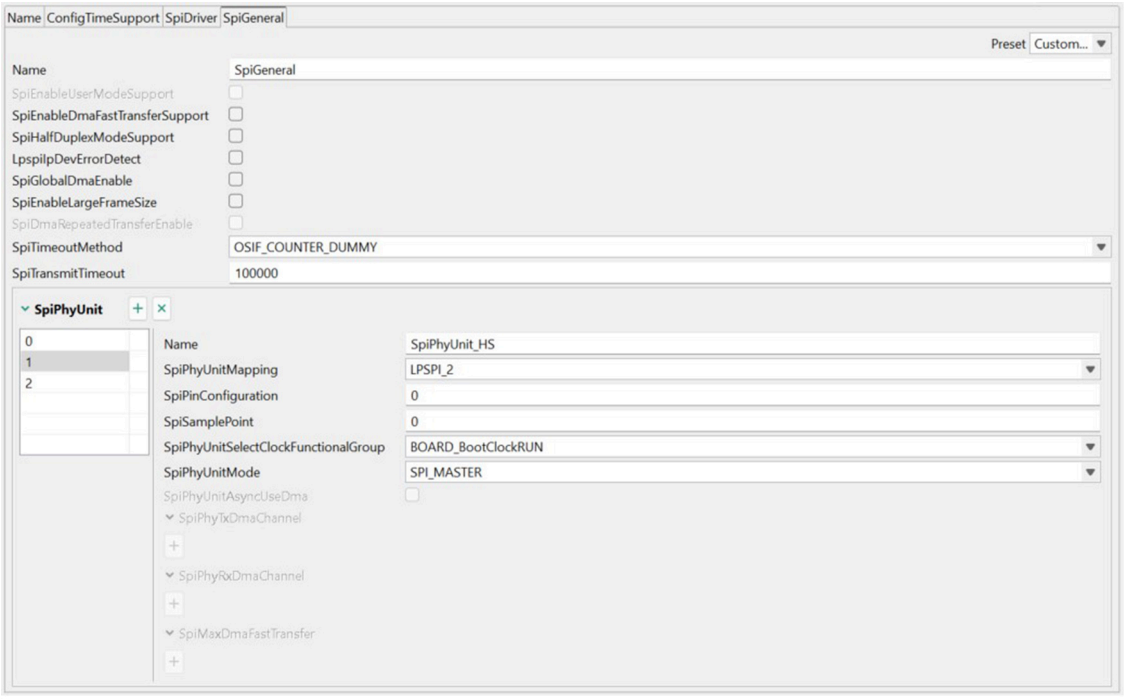
All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



aaa-060288

Figure 52. LPSPI General FS26 configuration



aaa-060289

Figure 53. LPSPI General GD3162 HS configuration

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

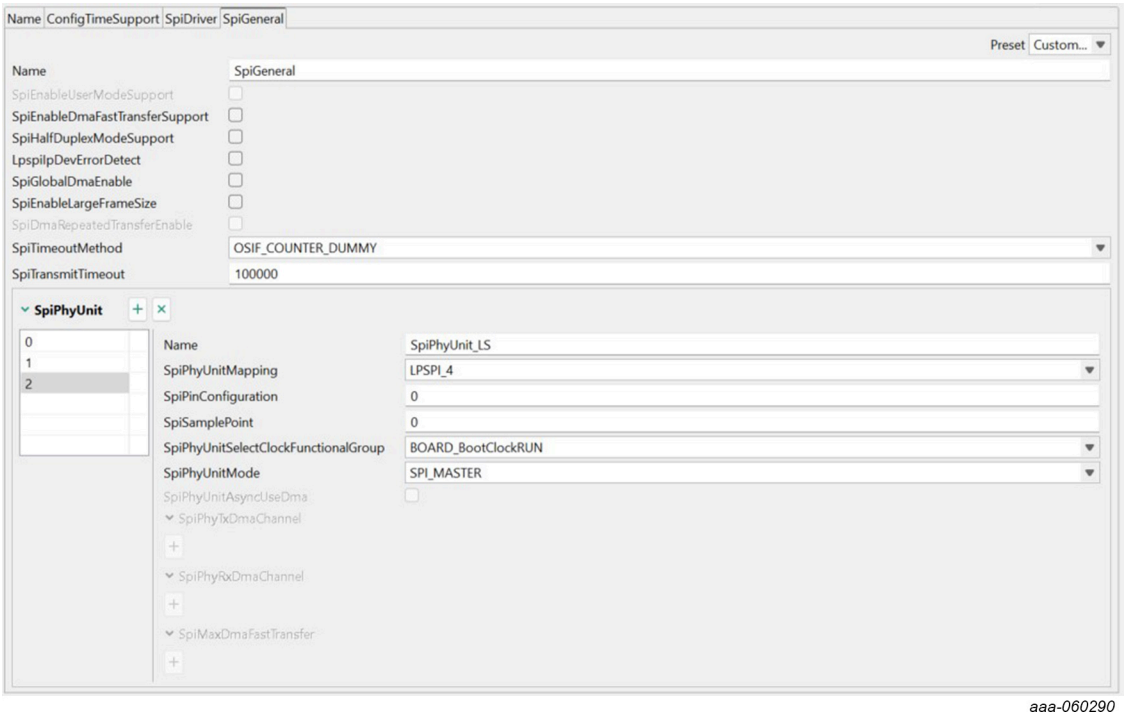


Figure 54. LPSPI General GD3162 LS configuration

4.3.15 GD3162 configuration

In this application, six GD3162 drivers are used, which are configurable over the SPI interface in two hardware chains. The GD3162 drivers also support the dynamic gate strength (GS) control, which increases the performance of the inverter. In the EV-INVERTERGEN3, the GS can be controlled over the pins or via SPI. GS control over the pins allows separate adjustment of rising and falling edges per phase. This solution is considered to be faster. Although GS control over the SPI is slower, it allows to configure GS per each transistor switch. Additionally, a safety feature is implemented to ensure that if the MCU becomes unresponsive, the GD is set to a predefined state based on the FS26 outputs. For PIN and LPSPI configuration details, check Section 4.3.2 "Port control and pin multiplexing" and Section 4.3.14 "LPSPI configuration". The high-level peripheral configuration for GD3162 is shown in Figure 55.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

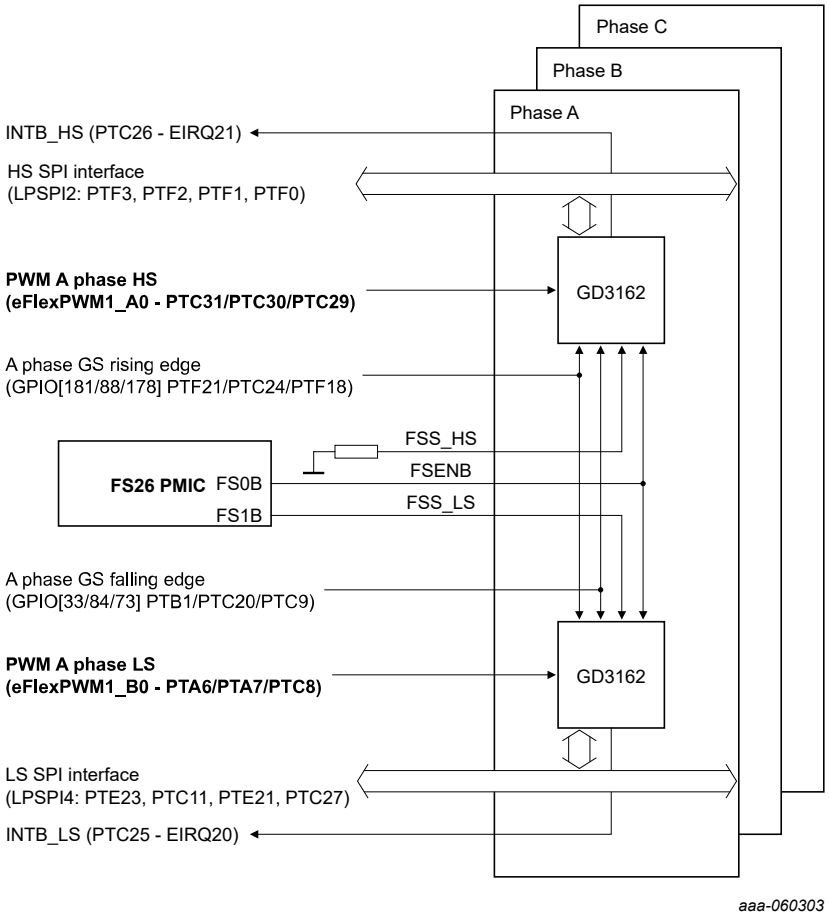


Figure 55. High-level peripheral configuration for GD3162

The RTD configuration of GD3162 is comprehensive. It is shown through multiple figures, starting with the general configuration in Figure 56. The SPI configuration is illustrated in Figure 57. The PIN configuration is defined in Figure 58 and Figure 59. The gate device configuration begins with Figure 60 and continues through multiple settings, ending with Figure 75.

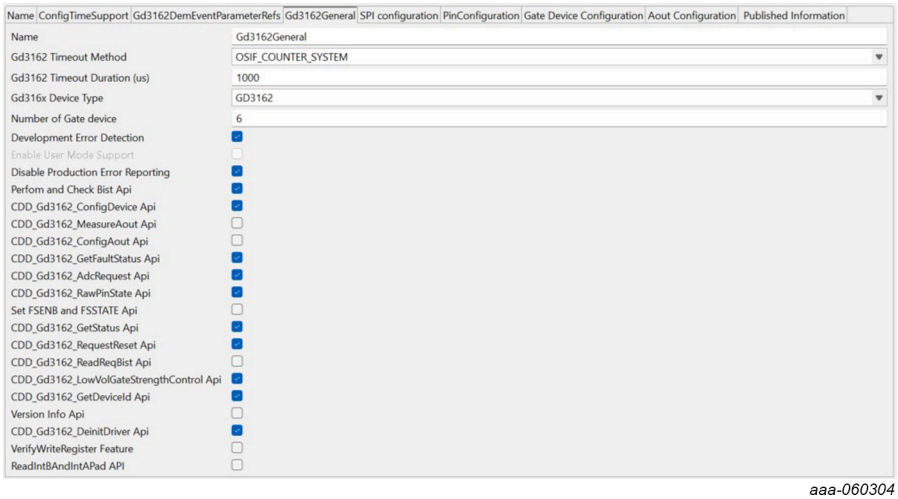
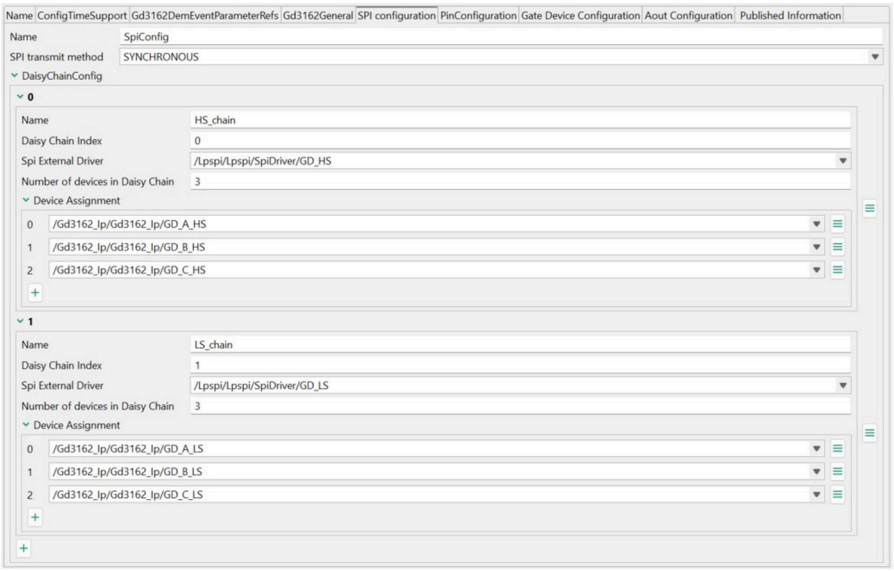


Figure 56. GD3162 General Configuration

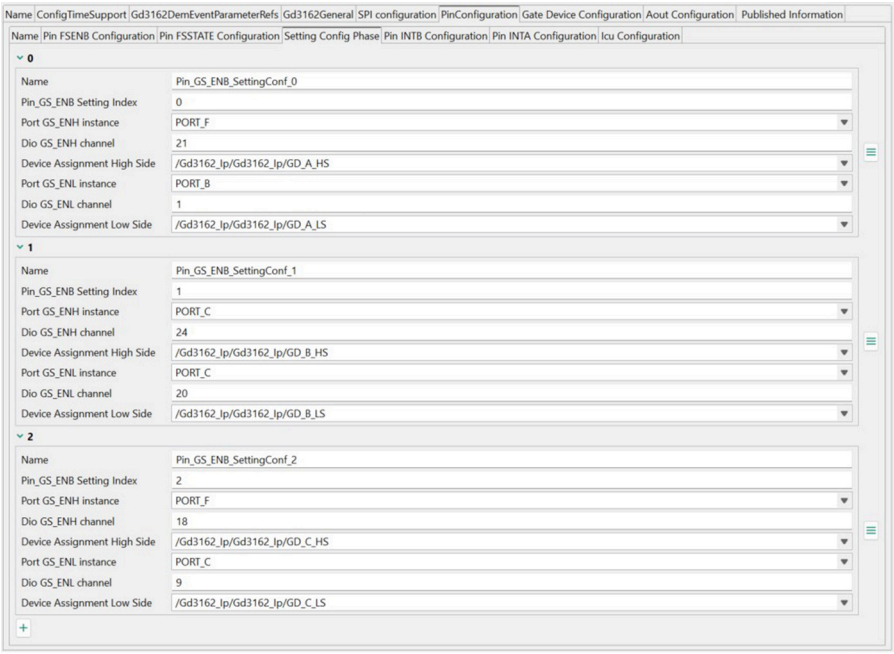
All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



aaa-060305

Figure 57. GD3162 SPI Configuration

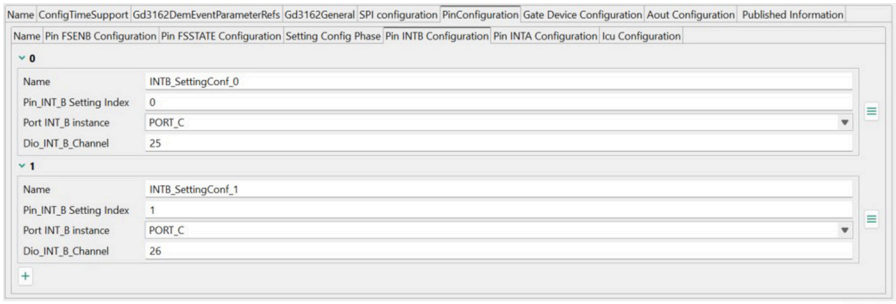


aaa-060306

Figure 58. GD3162 Pin Configuration Setting Config Phase

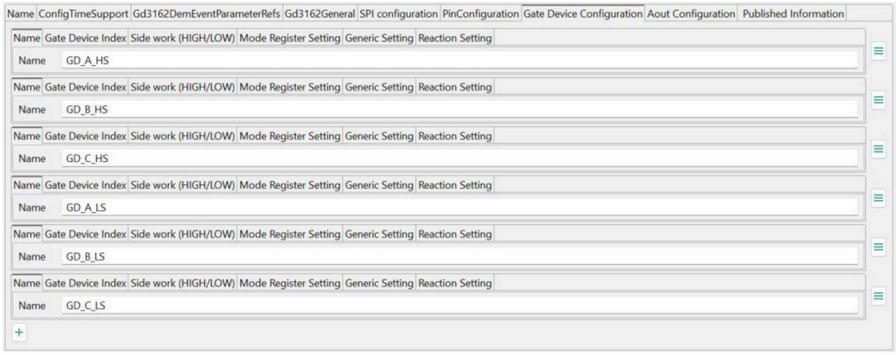
All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



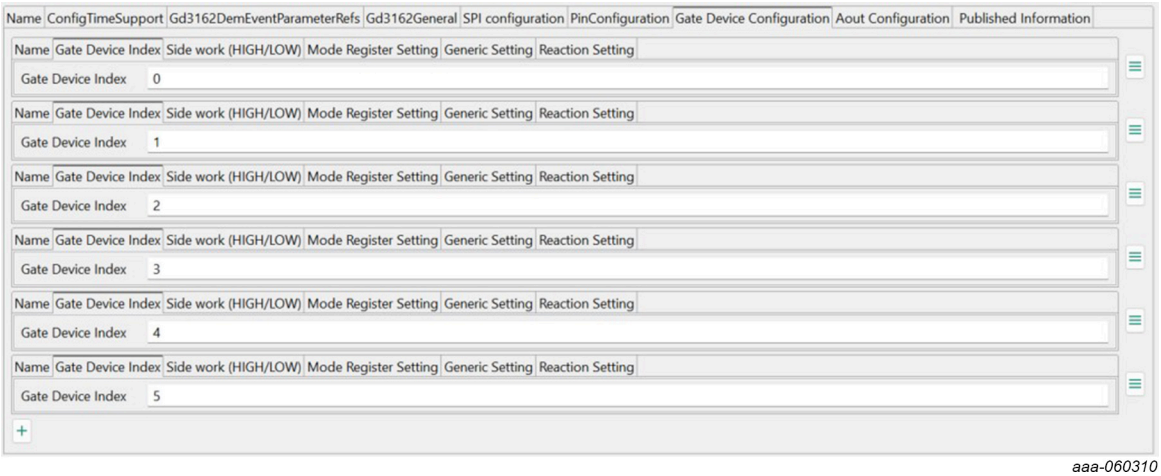
aaa-060307

Figure 59. GD3162 Pin Configuration INTB



aaa-060309

Figure 60. GD3162 Gate Device Configuration Name



aaa-060310

Figure 61. GD3162 Gate Device Configuration Index

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

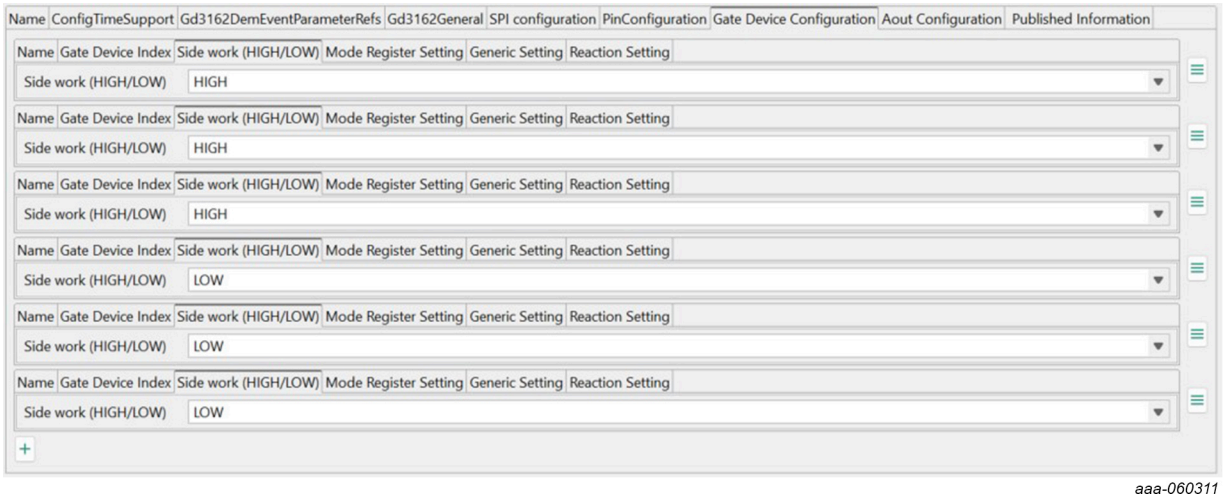


Figure 62. GD3162 Gate Device Configuration Side work

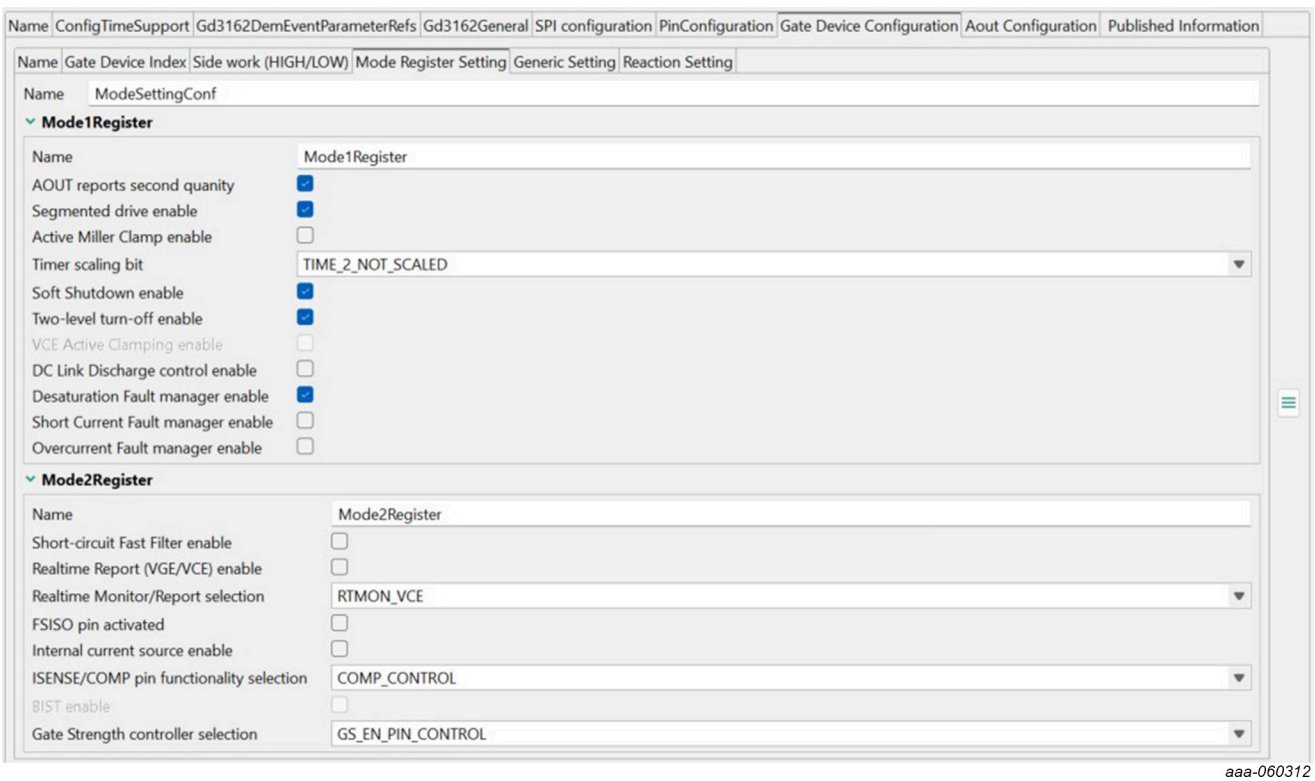


Figure 63. GD3162 Gate Device Configuration Mode Register

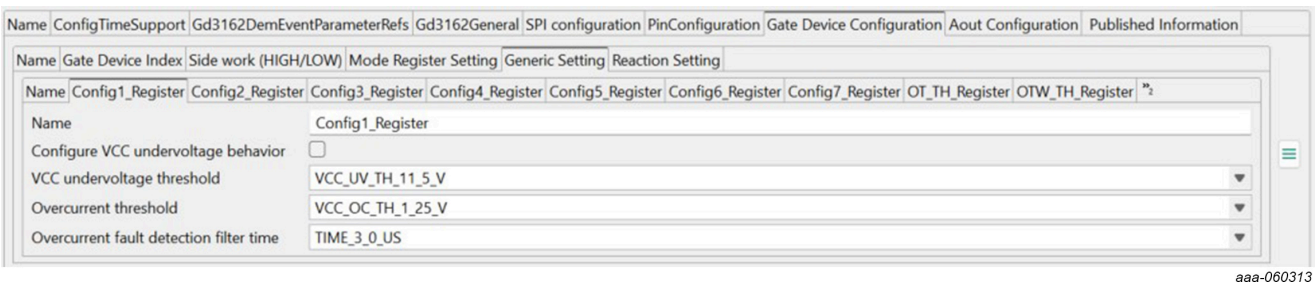
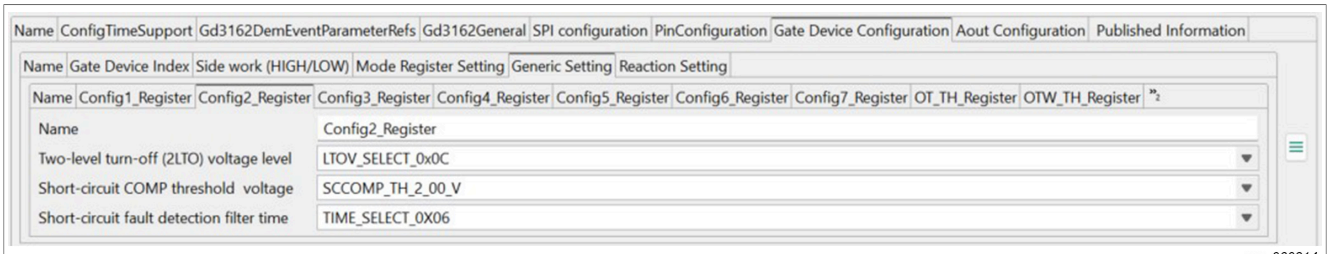


Figure 64. GD3162 Gate Device Configuration Config1

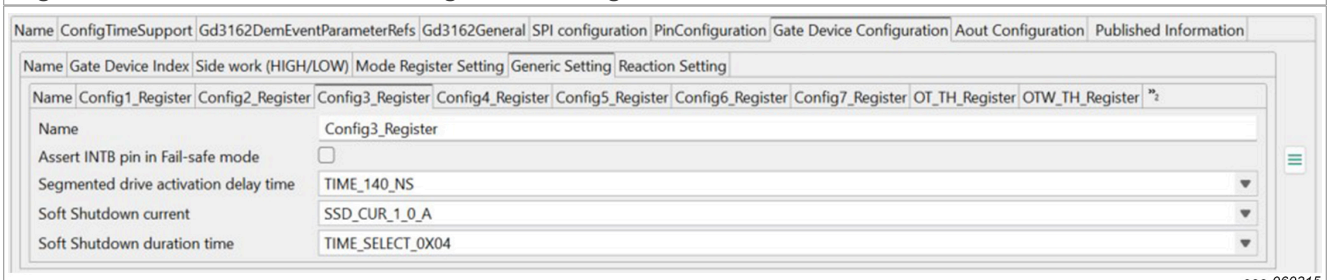
All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



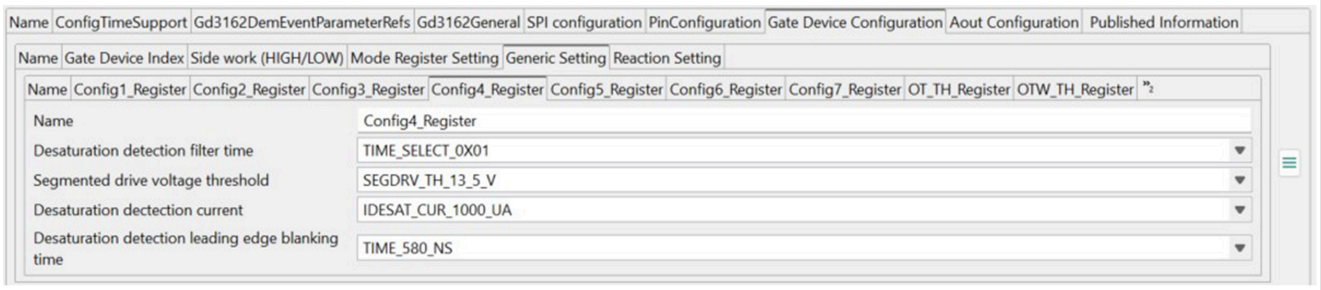
aaa-060314

Figure 65. GD3162 Gate Device Configuration Config2



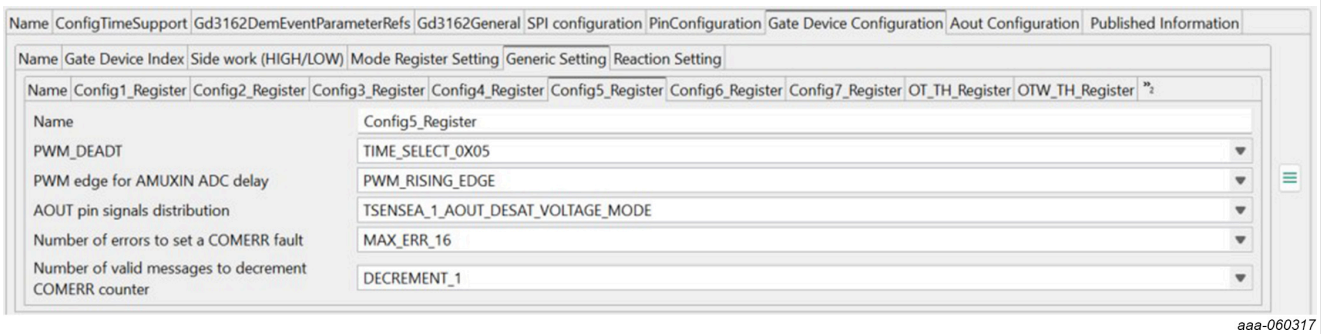
aaa-060315

Figure 66. GD3162 Gate Device Configuration Config3



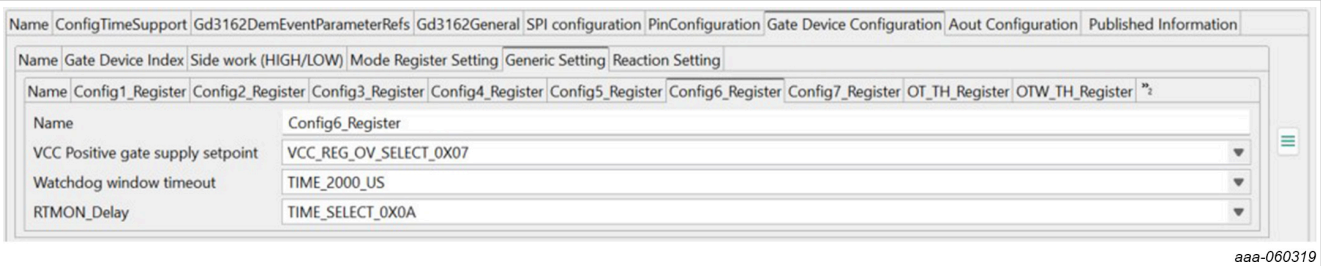
aaa-060316

Figure 67. GD3162 Gate Device Configuration Config4



aaa-060317

Figure 68. GD3162 Gate Device Configuration Config5

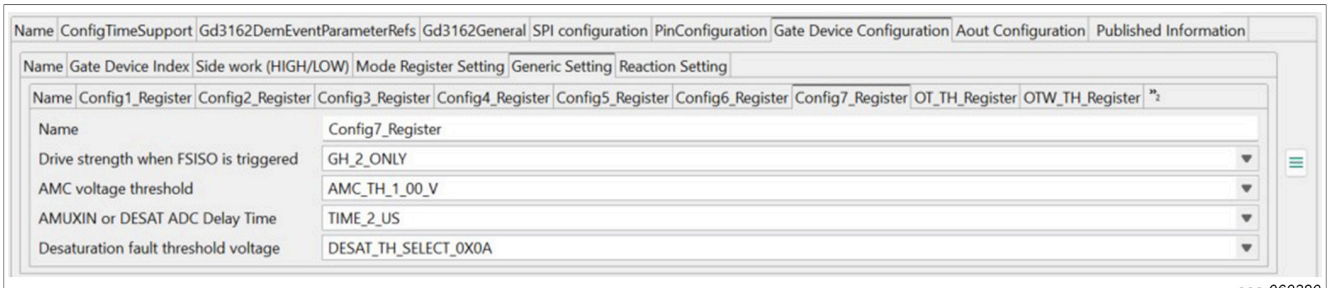


aaa-060319

Figure 69. GD3162 Gate Device Configuration Config6

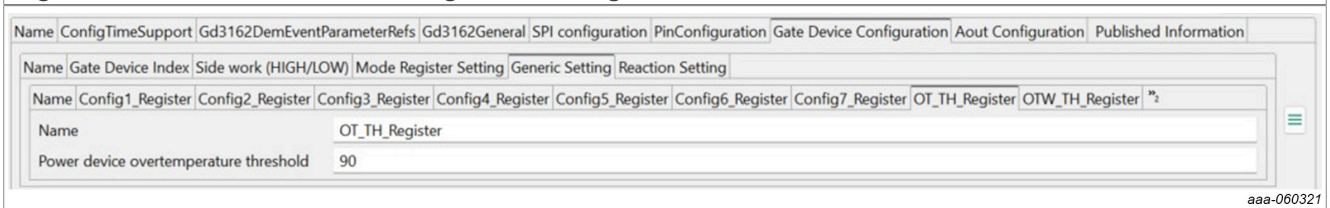
All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.



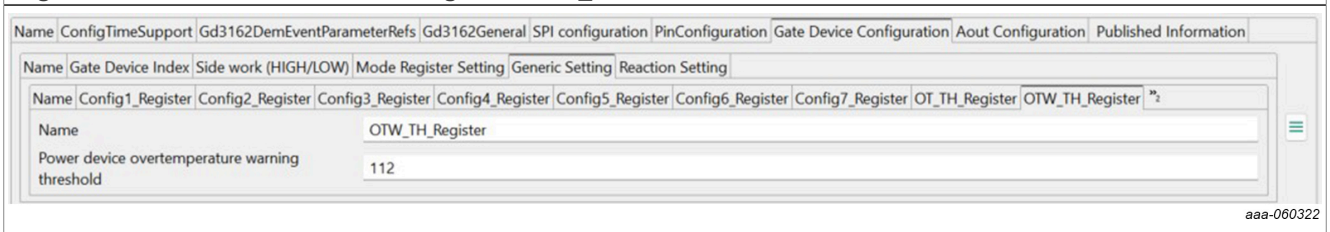
aaa-060320

Figure 70. GD3162 Gate Device Configuration Config7



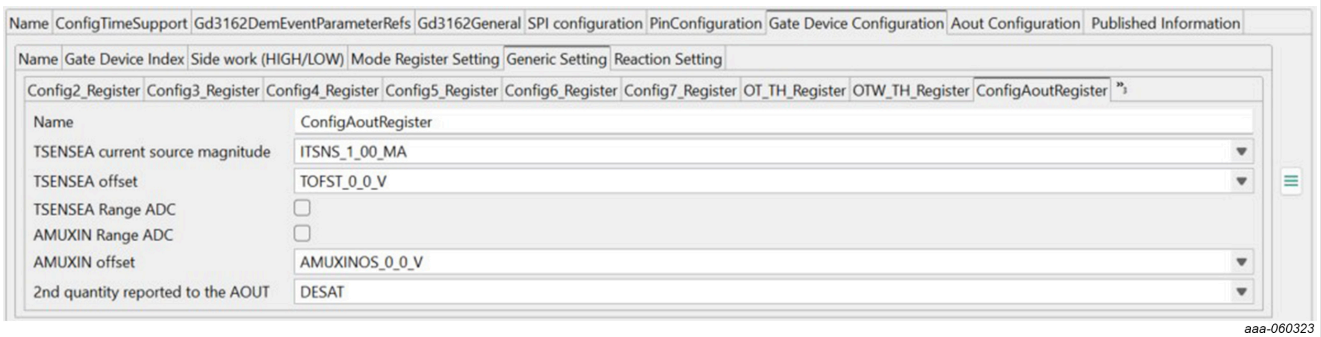
aaa-060321

Figure 71. GD3162 Gate Device Configuration OT_TH



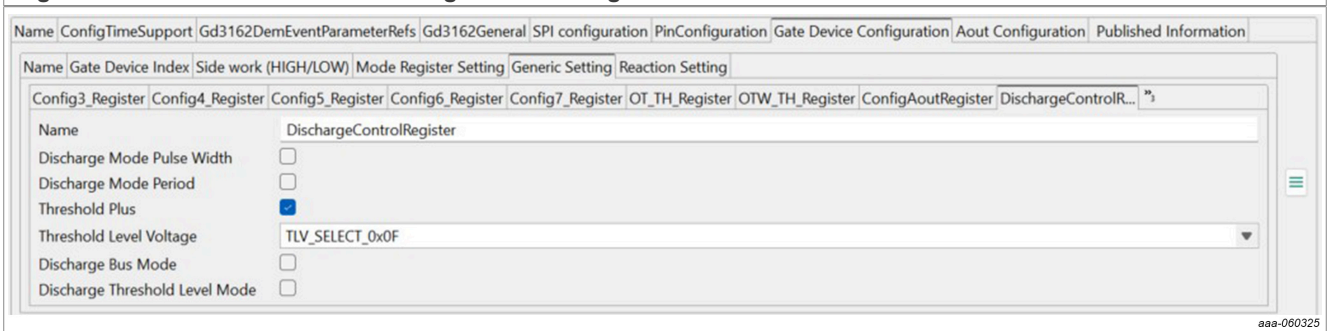
aaa-060322

Figure 72. GD3162 Gate Device Configuration OTW_TH



aaa-060323

Figure 73. GD3162 Gate Device Configuration ConfigAout



aaa-060325

Figure 74. GD3162 Gate Device Configuration DischargeControl

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

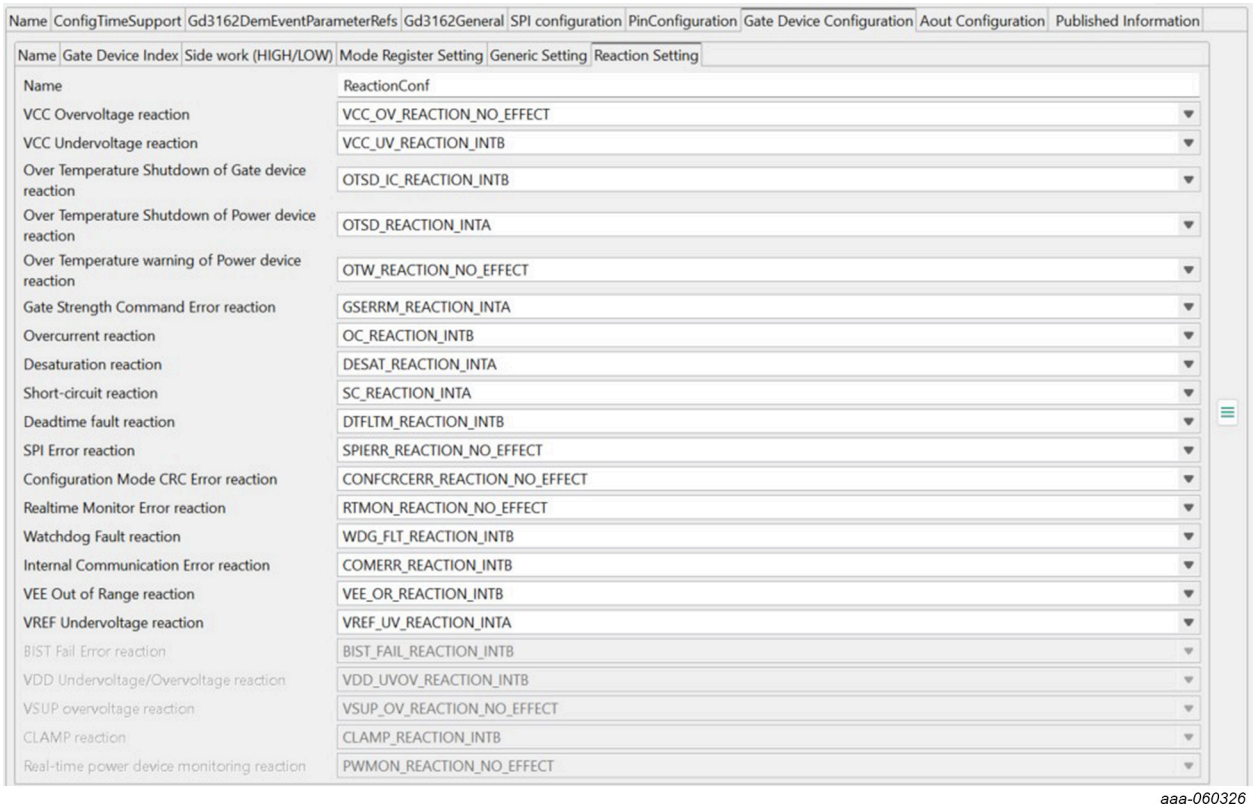


Figure 75. GD3162 Gate Device Configuration Reaction Setting

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.4 Software architecture

4.4.1 Introduction

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

4.4.2 Application data flow overview

The application software is interrupt-driven running in real-time. There is one periodic interrupt handler associated with the eTPU analog sensing interrupt, executing all motor control tasks. This includes both current and speed loop control. All tasks are performed in an order described by the application state machine shown in Figure 78, and application flowcharts shown in Figure 76 and Figure 77.

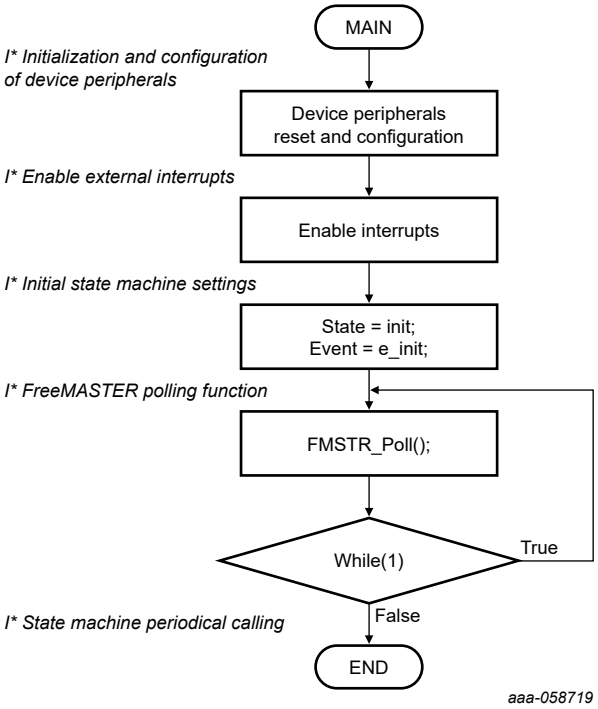


Figure 76. Flowchart diagram of main function with background loop

To achieve a 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 handler. Therefore, to call state machine functions, the peripheral causing this periodic interrupt must be properly configured and the interrupt enabled. As described in Section 4.3 "S32K396 device initialization", all peripherals are initially configured, and all interrupts are enabled after a RESET of the device. When interrupts are enabled and all S32K396 peripherals are correctly configured, the state machine functions are called from the ISR. The background loop handles noncritical timing tasks, such as the FreeMASTER communication polling.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

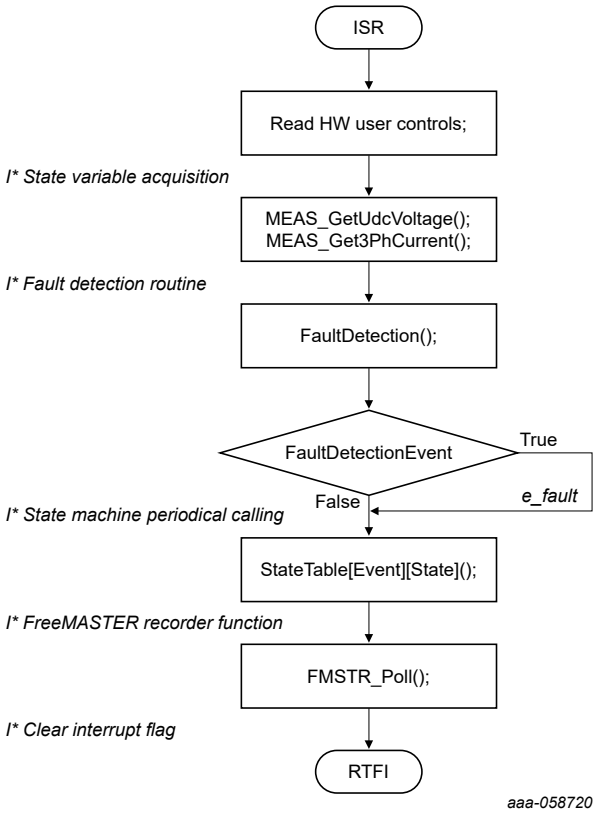


Figure 77. Flowchart diagram of periodic interrupt handler

4.4.3 State machine

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

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

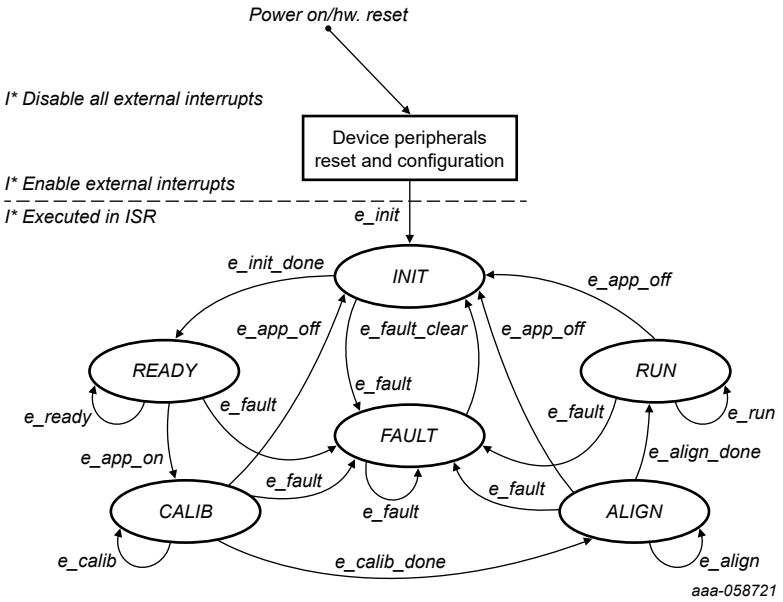


Figure 78. Application state machine

The application state machine consists of the 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, twelve 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

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.4.3.1 State - FAULT

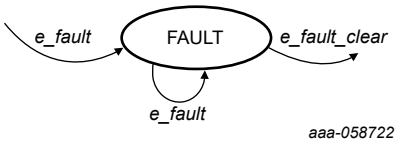


Figure 79. FAULT state with transitions

The application goes immediately to the FAULT state when a fault is detected. The system allows all states to pass into the FAULT state by setting *cntrState.event* = *e_fault*. FAULT is a state that transitions back to itself if the fault is still present in the system and the user does not request the 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. The fault register *permFaults* represents a fault flag, which is set and puts the application immediately into the 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 the application variable *switchFaultClear* is manually set to *true* (using FreeMASTER). That is, the user has acknowledged that the fault source has been removed and the application can be restarted. When the user sets *switchFaultClear* = *true*, the following sequence is automatically executed. See the code below.

```
if (ptr->cntrState.usrControl.switchFaultClear)
    {
  /* Clear permanent and temporary SW faults */
     ptr->permFaults.motor.R      = 0;     // Clear motor faults
     ptr->permFaults.stateMachine.R   = 0;     // Clear state machine faults
     FlexPwm_Ip_ClearFaultFlagHw(ptr->flexpwm_instance==0?
IP_EFLEXPWM_0:IP_EFLEXPWM_1,0xF); // Clear PWM faults

     /* Clearing the GD faults */
     if ((ptr->permFaults.GD3162_LS)||(ptr->permFaults.GD3162_HS))
     {
      ptr->permFaults.GD3162_LS = FALSE;
      ptr->permFaults.GD3162_HS = FALSE;
     }

     /* Clearing resolver fault */
     ptr->permFaults.resolver = FALSE;

  /* 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 an event to *cntrState.event* = *e_fault_clear* while in FAULT state represents a new request to proceed to INIT state. This request is purely a 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* to true. However, according to the interrupt data flow diagram shown in Figure 77, function *FaultDetection()* is called before state machine function *StateTable[Event][State]()*. Therefore, all faults are checked again and if there is any fault condition remaining in the system, the respective bits in *permFaults* and *tempFaults* variables are set. As a consequence of *permFaults* not being equal to zero, the function *FaultDetection()* modifies the application event from *e_fault_clear* back to *e_fault*, which means a jump to fault state when the state machine function *StateTable[Event][State]()* is called. Therefore, the INIT state is not entered, even though the user tries to clear the fault flags using

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

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 scans for the following system warnings and errors:

- DC bus overvoltage
- DC bus undervoltage
- Phase A, phase B, and phase C overcurrent
- Overspeed

The thresholds for fault detection can be modified in the MCAT. In addition, the fault state is entered if the following errors are detected:

- Resolver error (check resolver states)
- GD3162 errors (HS, LS)
- FOC error (irrelevant event call in the state machine)

4.4.3.2 State - INIT

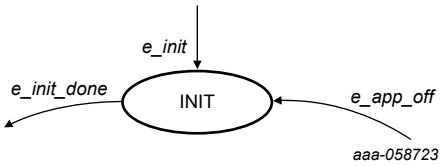


Figure 80. INIT state with transitions

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

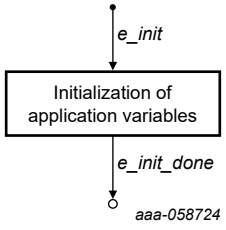


Figure 81. Flowchart of state INIT

4.4.3.3 State - READY

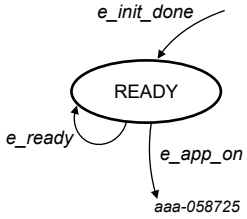


Figure 82. READY state with transitions

In the READY state, the application is waiting for the user command to start the motor. The application is released from waiting mode by the FreeMASTER interface setting the variable *switchAppOnOff* = *true*. See the flowchart in Figure 83.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

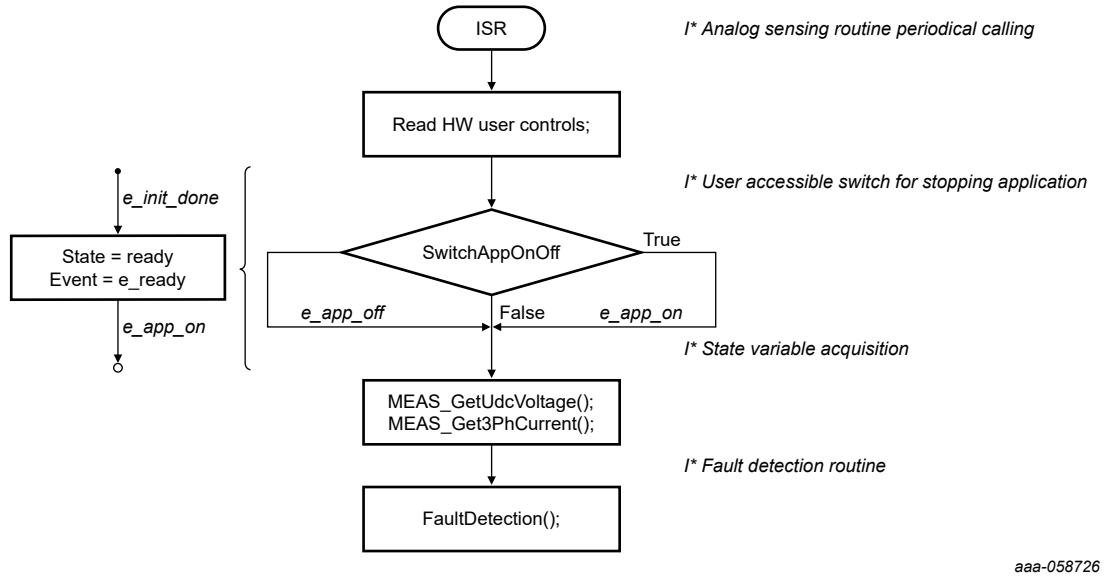


Figure 83. Flowchart of state READY

4.4.3.4 State - CALIB

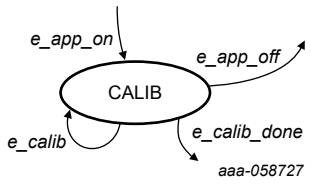


Figure 84. CALIB state with transitions

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

CALIB is a state that allows transition back to itself, provided no faults are present, the user does not request a stop of the application (by *switchAppOnOff = false*), and the calibration process has not finished. When the application event is automatically set to *cntrState.event = e_calib_done,* the state machine can proceed to state ALIGN.

A transition to the 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.

4.4.3.5 State - ALIGN

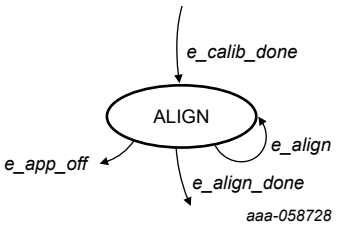


Figure 85. ALIGN state with transitions

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

This state shows the alignment of the rotor and stator flux vectors to mark zero position. The zero position is obtained at the ALIGN state, where a DC voltage is applied to the *d*-axis voltage for a certain period. This application of voltage causes the rotor to rotate to an 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 a zero position. To wait for the rotor to stabilize in an aligned position, a certain time period is selected during which the DC voltage is constantly applied. The period of time and the amplitude of the DC voltage can be modified in the INIT state. Timing is implemented using a software counter that counts from a predefined value down to zero. During this time, the event remains set to *cntrState.event* = *e_align*. When the counter reaches zero, the counter is reset back to the predefined value, and the event is automatically set to *cntrState.event* = *e_align_done*. This reset enables a transition to the RUN state. See the flowchart in Figure 86.

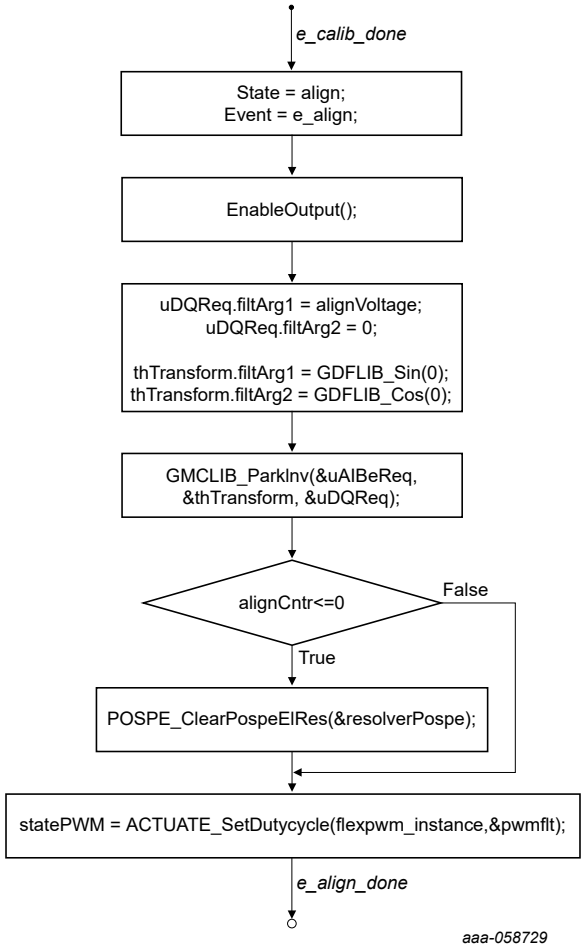


Figure 86. Flowchart of state ALIGN

A transition to the 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.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.4.3.6 State - RUN

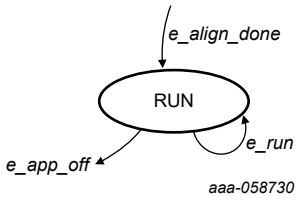


Figure 87. RUN state with transitions

In the RUN state, the FOC algorithm is calculated, as described in Section 3 "PMSM field oriented control".

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 the falling edge of *switchAppOnOff = false* using FreeMASTER.

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 for 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. The value of N depends on the ratio of speed and current loop sample time, which are set in MCAT.

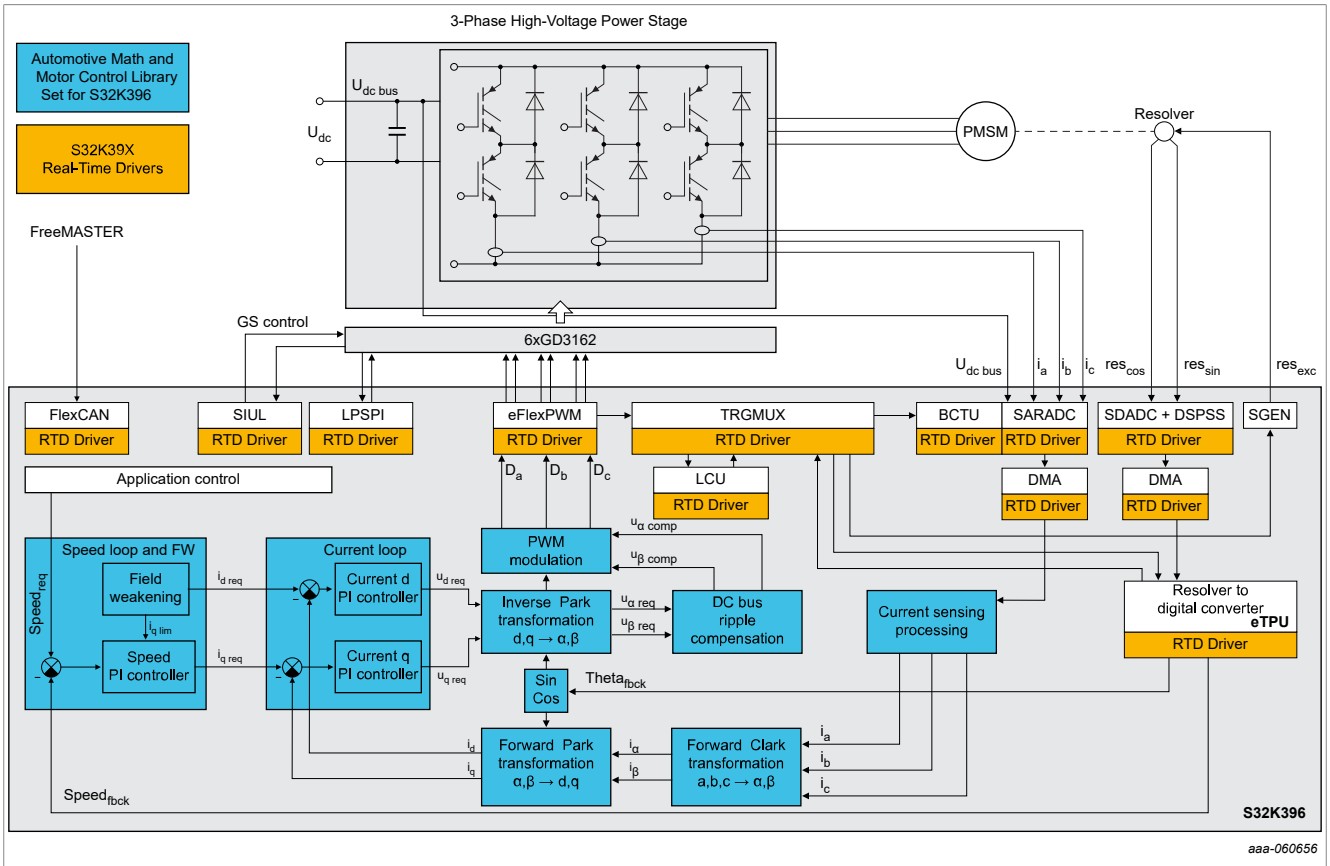


Figure 88. Module interconnection

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.4.4 AMMCLib integration

The application software of the FOC with resolver position sensing and field weakening is built using NXP's AMMCLib set. This set is 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 88. 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 S32K396 core, floating point arithmetic is used as a reference for this motor control application with PMSM.

The current loop function, AMCLIB_CurrentLoop, unites and optimizes the inner loop of the FOC cascade structure. This function consists of two PI controllers and basic mathematical operations that 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 89.

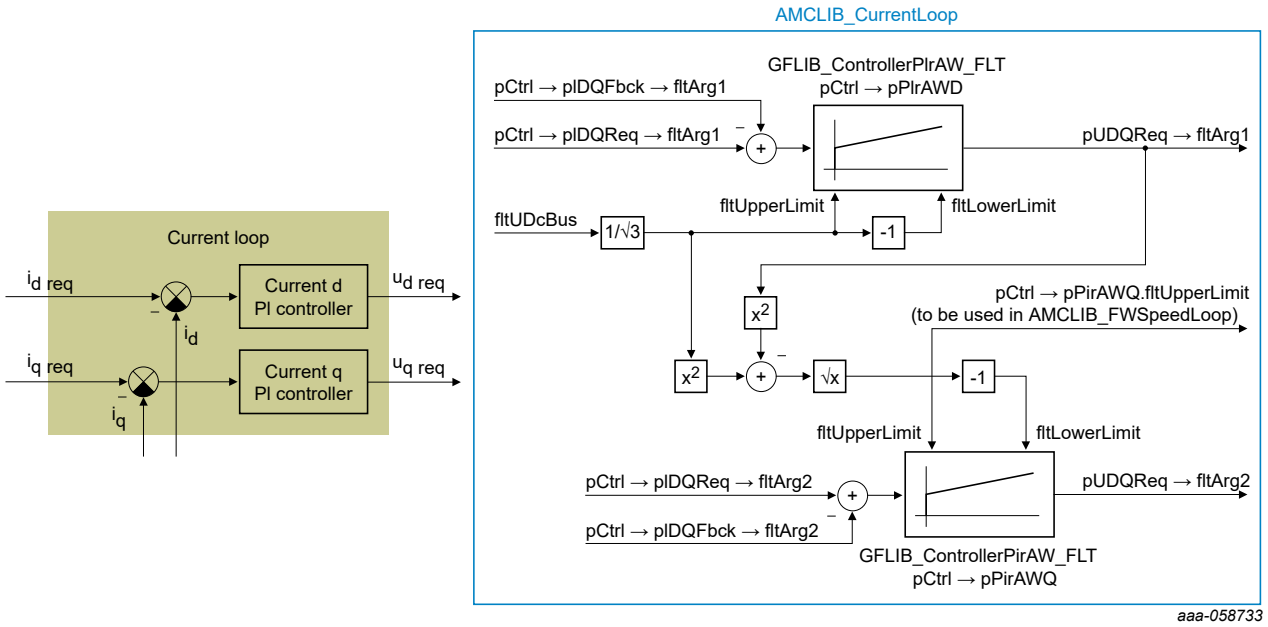


Figure 89. Functions and data structures in AMCLIB_CurrentLoop

Required *d*- and *q*-axis stator currents can be either manually modified or generated by the outer loop of the cascade structure consisting of speed loop and field weakening, as shown in Figure 90. To achieve a highly optimized level, AMCLIB_FWSpeedLoop merges two functions of the AMMCLib, namely speed control loop AMCLIB_SpeedLoop and field weakening control AMCLIB_FW. AMCLIB_SpeedLoop consists of speed PI controller GFLIB_ControllerPIpAW, speed ramp GFLIB_Ramp placed in the 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. All functions and data structures used in the AMCLIB_FW function are shown in Figure 90.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

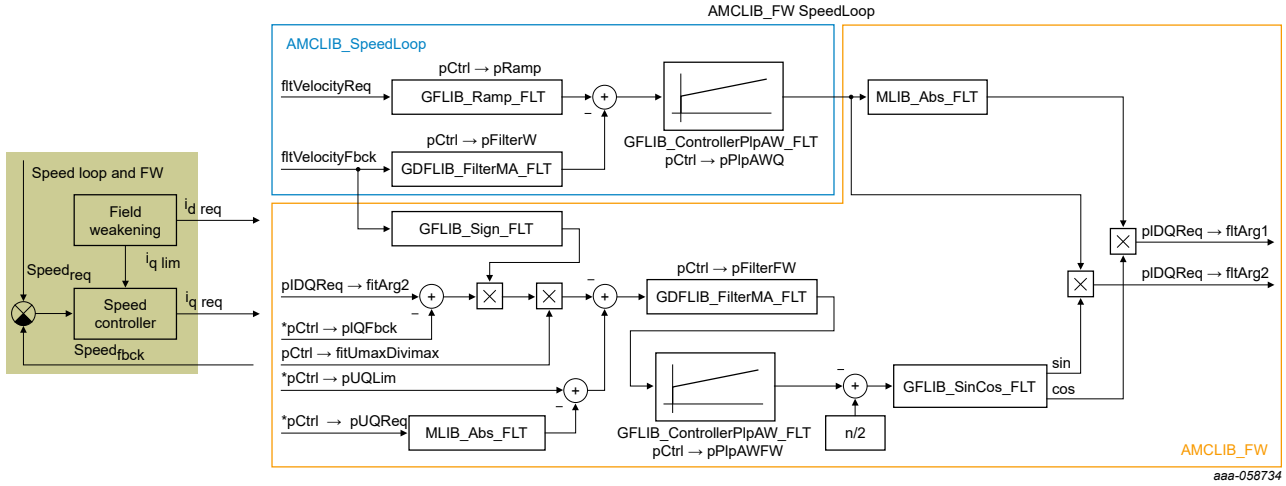


Figure 90. Functions and data structures in AMCLIB_FWSpeedLoop

AMCLIB_FW key advantages:

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

More details related to AMMCLib FOC functions can be found in the S32K396 AMMCLib User Manual ref.[2]. Parameters of the PI controllers placed in the speed and current control loop can be tuned by using NXP's MCAT tool. Detailed instructions on how to tune the parameters of the FOC structure by MCAT are presented in AN4642 ref.[5] and AN4912 ref.[6].

4.4.5 MCAT integration

MCAT is a graphical tool dedicated to motor control developers and 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 the new electric drive setup becomes easier because the MCAT tool offers the possibility to split the control structure and therefore control the motor at various levels of the cascade control structure.

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

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 file contains the following:

- Application scales
- Fault triggers
- Control loops parameters
- Speed sensor and/or observer settings

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- FreeMASTER scales

The PMSM_appconfig.h file should be linked to the project, and the constants should be used for the variables initialization.

FreeMASTER enables an online tuning of the motor control variables using the MCAT control and tuning view. However, 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

In the EV-INVERTERGEN3 application, communication between FreeMASTER and the S32K396 is provided via the CAN communication plugin. By default, the CAN interface for USB is used to communicate with the host PC via USB. The driver for PCAN-USB adapter can be downloaded from www.peak-system.com.

FreeMASTER and MCAT user interface are described in application note AN12235: 3-phase Sensorless PMSM Motor Control Kit with S32K144 ref.[7].

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 ref.[5].

6 Conclusion

The design described in this application note shows the simplicity and efficiency of using the S32K396 microcontroller for PMSM motor control and introduces it as an appropriate candidate for various applications in the automotive area. The MCAT tool provides an interactive online tool that makes the PMSM drive application tuning friendly and intuitive.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

7 Abbreviations

Table 6. Abbreviations

Acronym	Description
ADC	analog-to-digital converter
AMMCLib	automotive math and motor control library
ATO	angle tracking observer
BCTU	body cross-triggering unit
DMA	direct memory access
DSPSS	digital signal processing subsystem
EMF	electromotive force
eTPU	enhanced time processing unit
FIR	finite impulse response
FOC	field oriented control
ISR	interrupt service routine
LCU	logic control unit
LPSPI	low-power serial peripheral interface
MCAT	motor control application tuning
PMSM	permanent magnet synchronous motor
PWM	pulse-width modulation
RDC	resolver to digital converter
RTD	real-time drivers
SARADC	SARADC successive approximation register analog-to-digital converter
SDADC	sigma-delta ADC
SGEN	sine wave generator
TCM	tightly coupled memory

8 References

[1] S32 Design Studio for Arm
https://www.nxp.com/s32ds

[2] AMMCLib – Automotive Math and Motor Control Library Set for S32K3xx
https://www.nxp.com/ammclib

[3] eTPURDCCUG – eTPU RDC and RDC Checker User Guide
https://www.nxp.com/docs/en/user-guide/eTPURDCCUG.pdf

[4] S32K396RM – S32K39, S32K37, and S32K36 Reference Manual
https://www.nxp.com/webapp/Download?colCode=S32K396RM

[5] AN4642 – Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM
https://www.nxp.com/docs/en/application-note/AN4642.pdf

[6] AN4912 – Tuning 3-Phase PMSM Sensorless Control Application Using MCAT Tool
https://www.nxp.com/docs/en/application-note/AN4912.pdf

[7] AN12235 – 3-Phase Sensorless PMSM Motor Control Kit with S32K144
https://www.nxp.com/docs/en/application-note/AN12235.pdf

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

9 Note about the source code in the document

Copyright 2026 NXP. NXP Confidential. This software is owned or controlled by NXP and may only be used strictly in accordance with the applicable license terms found at https://www.nxp.com/LA_OPT_NXP_SW. Only the "internal use license" in Section 2.2 in the NXP SOFTWARE LICENSE AGREEMENT is granted for this software.

10 Revision history

Table 7. Revision history

Document ID	Release date	Description
AN14504 v.2.0	27 October 2026	• Section 2 "System concept", Control methodology includes, added the following items: – Dynamic gate strength control – DC link discharge control • Section 3.4 "Field weakening": Added note " Field weakening functionality is implemented…" • Section 4.2.1 "Module involvement in PMSM FOC control" – Revised Figure 17 – Revised Figure 18 – Removed figure "S32K396 motor control configuration" • Section 4.3 "S32K396 device initialization": Revised entire list • Table 3: Changed Input mode to "Differential" from "Single-ended" • Revised Figure 88
AN14504 v.1.0	09 May 2025	• Initial version

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

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.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

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.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

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.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Tables

Tab. 1. S32K396 clocking configuration 21
Tab. 2. Port control and pin multiplexing 23
Tab. 3. SDADC configuration for resolver 27
Tab. 4. DMA configuration 35
Tab. 5. TRGMUX configuration 47
Tab. 6. Abbreviations 70
Tab. 7. Revision history 72

Figures

Fig. 1. EV Traction Inverter Control Reference Design Gen 3 4
Fig. 2. FOC transformations 5
Fig. 3. Orientation of stationary and rotational reference frames, with current components transformed into both frames 6
Fig. 4. FOC current loop structure 8
Fig. 5. FOC speed loop structure 10
Fig. 6. Board layout 10
Fig. 7. Signal conditioning circuit for Hall effect sensor HC5FW 900-S 11
Fig. 8. Signal conditioning circuit for phase A current sensing 11
Fig. 9. Isolated voltage sensing circuit 12
Fig. 10. Excitation circuit 12
Fig. 11. Constant torque/power operating regions 13
Fig. 12. Constant flux/voltage operational regions 13
Fig. 13. Steady-state phasor diagram of PMSM operation up to base speed (left) and above speed (right) 14
Fig. 14. Voltage (left) and current (right) limits for PMSM drive operation 15
Fig. 15. eTPU resolver digital interface block diagram 16
Fig. 16. Oversampling and demodulation of resolver feedback signals 17
Fig. 17. S32K396 module interconnection block diagram 18
Fig. 18. Timing diagram of PWM and ADC synchronization 19
Fig. 19. FlexCAN configuration 25
Fig. 20. eTPU Resolver configuration 26
Fig. 21. SDADC General configuration 27
Fig. 22. Block diagram of DSPSS 29
Fig. 23. DSPSS configuration 30
Fig. 24. SARADC General configuration 31
Fig. 25. SARADC HW Unit configuration 32
Fig. 26. BCTU General configuration 33
Fig. 27. BCTU Results FIFOs 34
Fig. 28. eFlexPWM1 Submodule 0 36
Fig. 29. eFlexPWM1 Submodule 1 37
Fig. 30. eFlexPWM1 Submodule 2 38
Fig. 31. eFlexPWM1 Submodule 3 39
Fig. 32. LCU module functional configuration diagram 40
Fig. 33. LCU0 Logic instances configuration 40
Fig. 34. LCU0 Logic Cell 0 Input 0 configuration 41
Fig. 35. LCU0 Logic Cell 0 Input 1 configuration 41
Fig. 36. LCU0 Logic Cell 0 Input 2 configuration 41
Fig. 37. LCU0 Logic Cell 0 Input 3 configuration 42
Fig. 38. LCU0 Logic Cell 1 Input 0 configuration 42
Fig. 39. LCU0 Logic Cell 0 Output 0 configuration 43
Fig. 40. LCU0 Logic Cell 0 Output 1 configuration 43
Fig. 41. LCU0 Logic Cell 0 Output 2 configuration 44
Fig. 42. LCU0 Logic Cell 1 Output 0 configuration 45
Fig. 43. TRGMUX 0 configuration 45
Fig. 44. TRGMUX 1 configuration 46
Fig. 45. TRGMUX 2 configuration 46
Fig. 46. TRGMUX 5 configuration 46
Fig. 47. TRGMUX 3 configuration 47
Fig. 48. TRGMUX 4 configuration 47
Fig. 49. LPSPI Driver GD3162 LS configuration 48
Fig. 50. LPSPI Driver GD3162 HS configuration 49
Fig. 51. LPSPI Driver FS26 configuration 49
Fig. 52. LPSPI General FS26 configuration 50
Fig. 53. LPSPI General GD3162 HS configuration 50
Fig. 54. LPSPI General GD3162 LS configuration 51
Fig. 55. High-level peripheral configuration for GD3162 52
Fig. 56. GD3162 General Configuration 52
Fig. 57. GD3162 SPI Configuration 53
Fig. 58. GD3162 Pin Configuration Setting Config Phase 53
Fig. 59. GD3162 Pin Configuration INTB 54
Fig. 60. GD3162 Gate Device Configuration Name 54
Fig. 61. GD3162 Gate Device Configuration Index 54
Fig. 62. GD3162 Gate Device Configuration Side work 55
Fig. 63. GD3162 Gate Device Configuration Mode Register 55
Fig. 64. GD3162 Gate Device Configuration Config1 55
Fig. 65. GD3162 Gate Device Configuration Config2 56
Fig. 66. GD3162 Gate Device Configuration Config3 56
Fig. 67. GD3162 Gate Device Configuration Config4 56
Fig. 68. GD3162 Gate Device Configuration Config5 56
Fig. 69. GD3162 Gate Device Configuration Config6 56
Fig. 70. GD3162 Gate Device Configuration Config7 57
Fig. 71. GD3162 Gate Device Configuration OT_TH ... 57

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Fig. 72. GD3162 Gate Device Configuration OTW_TH 57
Fig. 73. GD3162 Gate Device Configuration ConfigAout 57
Fig. 74. GD3162 Gate Device Configuration DischargeControl 57
Fig. 75. GD3162 Gate Device Configuration Reaction Setting 58
Fig. 76. Flowchart diagram of main function with background loop 59
Fig. 77. Flowchart diagram of periodic interrupt handler 60
Fig. 78. Application state machine 61
Fig. 79. FAULT state with transitions 62
Fig. 80. INIT state with transitions 63
Fig. 81. Flowchart of state INIT 63
Fig. 82. READY state with transitions 63
Fig. 83. Flowchart of state READY 64
Fig. 84. CALIB state with transitions 64
Fig. 85. ALIGN state with transitions 64
Fig. 86. Flowchart of state ALIGN 65
Fig. 87. RUN state with transitions 66
Fig. 88. Module interconnection 66
Fig. 89. Functions and data structures in AMCLIB_CurrentLoop 67
Fig. 90. Functions and data structures in AMCLIB_FWSpeedLoop 68

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Contents

1 Introduction 3
2 System concept 3
3 PMSM field oriented control 4
3.1 Fundamental principle of PMSM FOC 4
3.2 PMSM model in rotational reference frame dq 5
3.2.1 Current control 6
3.2.2 Speed control 8
3.3 Current, voltage, and position sensing 10
3.4 Field weakening 12
4 Software implementation on the S32K396 15
4.1 eTPU 15
4.1.1 eTPU based resolver to digital converter (RDC) 15
4.2 S32K396 - key modules for PMSM FOC control 17
4.2.1 Module involvement in PMSM FOC control 18
4.3 S32K396 device initialization 20
4.3.1 Clock configuration 21
4.3.2 Port control and pin multiplexing 23
4.3.3 FlexCAN configuration 24
4.3.4 eTPU Resolver configuration 25
4.3.5 SDADC configuration 26
4.3.6 SGEN configuration 28
4.3.7 DSP configuration 28
4.3.8 ADC configuration 31
4.3.9 BCTU configuration 32
4.3.10 DMA configuration 34
4.3.11 eFlexPWM configuration 36
4.3.12 LCU configuration 39
4.3.13 TRGMUX configuration 45
4.3.14 LPSPI configuration 47
4.3.15 GD3162 configuration 51
4.4 Software architecture 59
4.4.1 Introduction 59
4.4.2 Application data flow overview 59
4.4.3 State machine 60
4.4.3.1 State - FAULT 62
4.4.3.2 State - INIT 63
4.4.3.3 State - READY 63
4.4.3.4 State - CALIB 64
4.4.3.5 State - ALIGN 64
4.4.3.6 State - RUN 66
4.4.4 AMMCLib integration 67
4.4.5 MCAT integration 68
5 FreeMASTER and MCAT user interface 69
6 Conclusion 69
7 Abbreviations 70
8 References 70
9 Note about the source code in the document 71
10 Revision history 72
Legal information 73

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2026 NXP B.V. All rights reserved.