AN14326

3-phase Motor Control Kit with S32K396

Rev. 2.0 — 22 October 2024 Application note

Document information

Information	Content
Keywords	S32K396, Motor Control, FOC, eTPU
Abstract	This application note focuses on the peripheral configuration of the S32K396 motor control, especially the differences from the older platform. First, include a system conceptual description and a control flow diagram. Then, each module configuration is described in detail.

1 Introduction

S32K396 extends the K3xx MCUs in the automotive industry with the Cortex-M7 core at a higher frequency, advanced motor control coprocessors and extended analog including high-resolution PWM. It is developed to meet the next-generation SiC traction inverter requirements and to enable high efficiency and low latency features.

This application note does not describe the features of S32K396. Only a brief introduction to the features used in motor control is provided.

This application note aims to provide a guide for migrating the motor control project from the NXP PowerPC Cobra55 series, like MPC5777C, MPC5775E/B, and even MPC5744P, to the new S32K396 platform. This document only focuses on the main difference when migrating the motor control project from the Cobra55 platform to S32K396. There is no detailed description of the peripherals and features. And also, there is no detailed description of the 3-phase PMSM control. For the 3-phase PMSM control, see the AN12017: 3-Phase PMSM Development Kit with MPC5744P and/or AN13038: MCSPTR2A5775E 3-phase PMSM Motor Control Kit with MPC5775E. For more information of the chip, see S32K396 Reference Manual.

This application note covers three application scenarios: single motor in S32K396-BGA-DC, dual-motor in S32396-BGA-DC and S32X-MB, and single motor in S32K396-LQFP-DC. The configuration of the single motor in S32K396-LQFP-DC almost the same as the single motor in S32K396-BGA-DC. In addition to this, the differences in ADC are instanced caused by hardware. In this document, the default hardware is S32396-BGA-DC. If other applications differ, it is pointed out.

Development environment is based on Auto-Sar MCAL and config tool EB Tresos, motor control algorithm is from NXP Automotive Math and Motor Control Library (AMMCLIB), software resolver is implemented in eTPU based on S32k396 real-time Driver(RTD).

2 System concept

The system is based on the 3-phase PMSM control project on MPC5777C and/or MPC5744P. The application meets the following performance specifications:

- MC33937 MOSFETs predriver with an extensive set of functions and condition monitoring(see References)
- Incorporates a control technique with:
- Field Oriented Control of 3-phase PMSM with resolver position sensor
- Closed-loop speed control with action period 1 ms
- Closed-loop current control with action period 100 us
- Bidirection rotation
- Field weakening control extends the speed range of the PMSM beyond the base speed
- Position and speed are computed by eTPU
- Sensing of 3-phase motor currents
- FOC state variables sampled with period 100 us
- Automotive Math and Motor Control Library (AMMCLIB) – FOC algorithm built on blocks of precompiled SW library (see References)
- Use of eTPU Motor control function set to offload CPU
- FreeMASTER project for real-time debugging and data visualization
- Graphical control page to control the motor (motor start/stop, speed setup)
- High-speed recorder (reconstructed motor currents, vector control algorithm quantities)
- DC-Bus overvoltage and undervoltage, overcurrent, overload, and startup fail protection

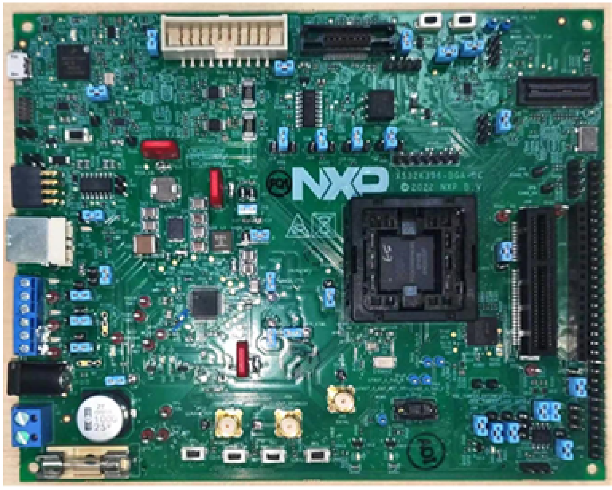


Figure 1. S32K396-BGA-DC

Figure 2. S32K396-BGA-DC and S32X-MB

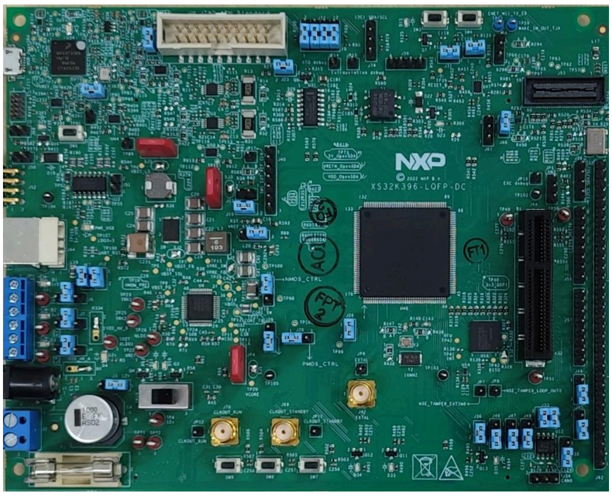


Figure 3. S32K396-LQFP-DC

Figure 4. MC33937 MOSFETs predriver

3 PMSM field-oriented control

For the 3-phase PMSM FOC part, see the AN12017: 3-Phase PMSM Development Kit with MPC5744P and/or AN13038: MCSPTR2A5775E 3-phase PMSM Motor Control Kit with MPC5775E.

4 Software implementation on S32K396

4.1 eTPU

The eTPU module in S32K396 has no much difference from the module on MPC5777x serials. In this application, the eTPU takes the same roles as it works on Cobra55. It is responsible for motor speed and position estimation and analog sensing. For this part, see Chapter 4.1 in AN13038: MCSPTR2A5775E 3-phase PMSM Motor Control Kit with MPC5775E.

4.2 S32K396 Key modules for PMSM FOC control

4.2.1 Interconnection with FETs Pre-Driver MC33937

Excitation of power FETs is ensured by NXP MC33937 pre-driver. This analog device is equipped with a charge pump that ensures external FETs drive at low power supply voltages. Moreover, three external bootstrap capacitors provide gate charge to the high-side FETs (see MC33937: 3-Phase Field Effect Transistor Pre-driver).

Configuration of MC33937 pre-driver is realized via SPI module. The MC33937 allows different operating modes to be set and locked by SPI commands. SPI commands also report the condition of the MC33937 based on the internal monitoring circuits and fault detection logic.

4.2.2 Module involvement in PMSM FOC control

The following figure shows the modules involved in the motor control, and also presents the connection of modules in this application.

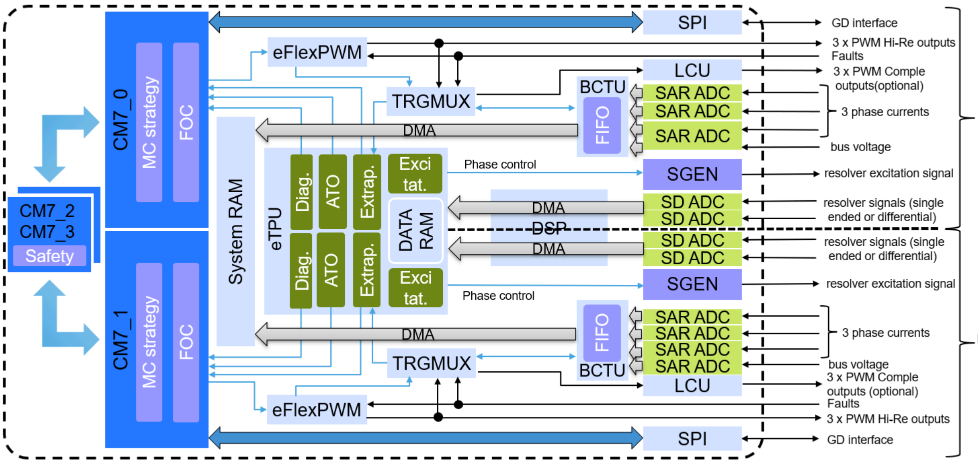


Figure 5. S32K396 motor control diagram

4.2.3 Motor trigger relationship

The following diagram shown that the motor trigger relationship. FlexPWM is used to generate a trigger signal that is connected to BCTU, this trigger signal can be delayed by LCU unit, and this configurable timing delay ensures accuracy of phase current sampling. At the same trigger point, the hardware triggers eTPU to extrapolate motor rotor position.

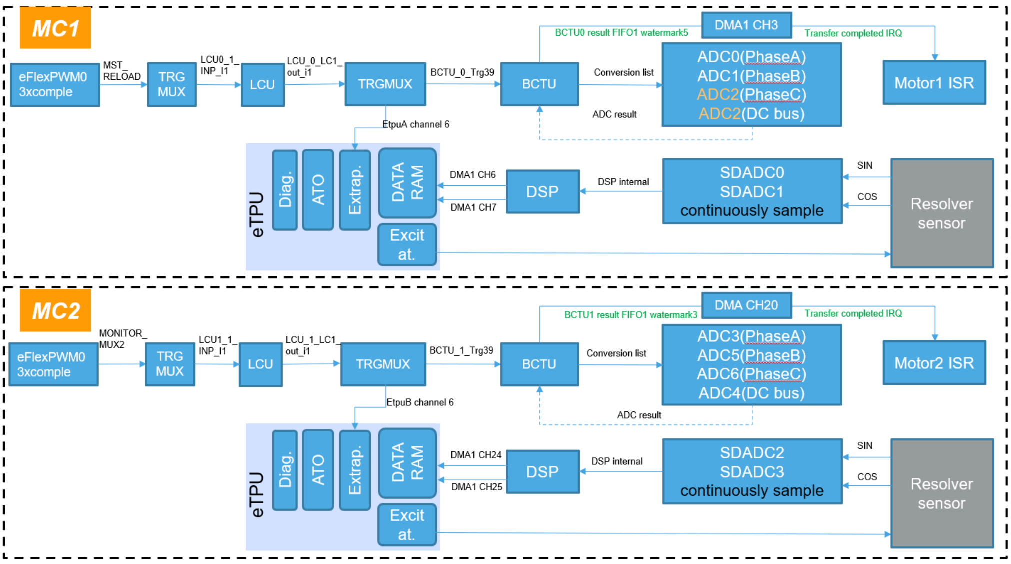


Figure 6. Motor trigger signal scheme

4.3 Device initialization

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. For more details, see S32K396 Reference Manual. The following Table 1 shows the clock configuration in this application.

Table 1. Clock configuration

Clock	Frequency (MHz)
CORE_CLK	160
M7_CORE_CLK	320
AIPS_PLAT_CLK	80
AIPS_SLOW_CLK	40
eTPU	M7_CORE_CLK
SDADC	80
DSP	160
SARADC	80
LCU	CORE_CLK
LPUART	AIPS_PLAT_CLK
LPSPI	AIPS_SLOW_CLK
eFlexPWM	CORE_CLK
SGEN	20

4.3.2 LPUART configuration

Low-Power Universal Asynchronous Receiver/Transmitter (LPUART) is used for communication between the S32K396 and FreeMASTER tool. Configured LPUART0 with baud rate 115200. See the following for configuration parameters.

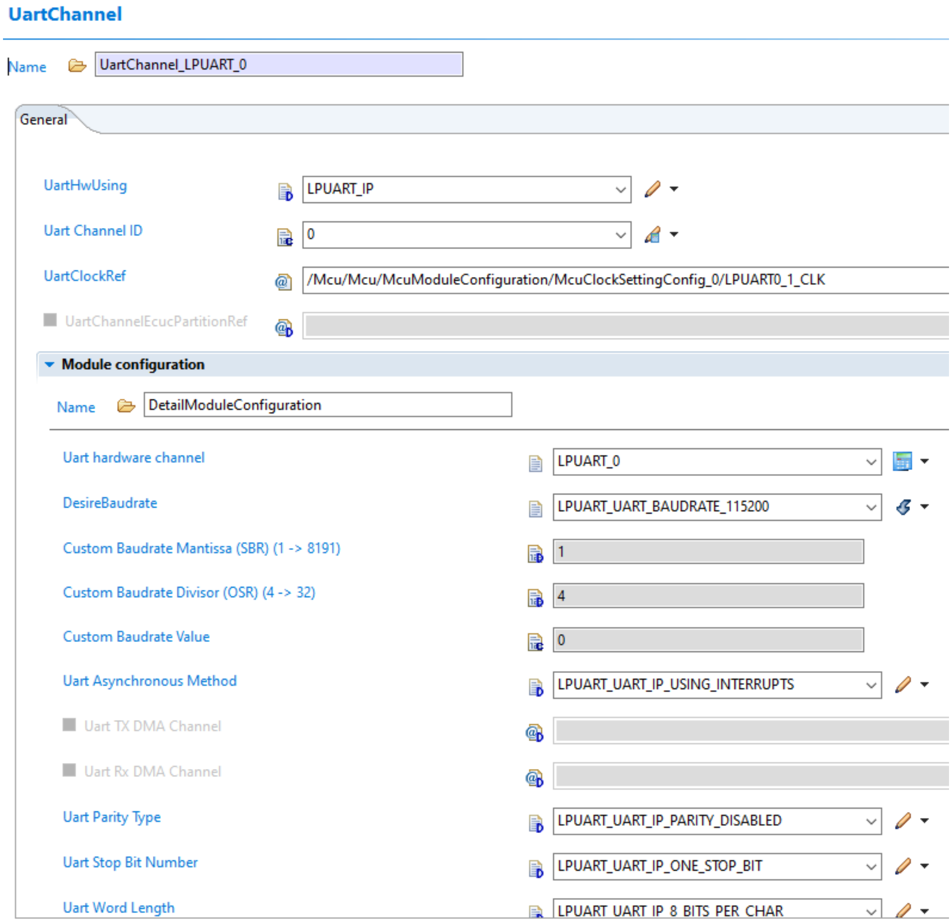


Figure 7. LPUART configuration

4.3.3 eTPU configuration

The configuration for eTPU is similar to the configuration on Cobra55. See the AN13038: MCSPTR2A5775E 3-phase PMSM Motor Control Kit with MPC5775E. In this application note, the eTPU generates excitation signal and resolver digital converter. The following list is part of the changes to adapt the S32K396, and these eTPU configuration parameters can be configured in EB Tresos based on S32K3_ETPU_SW_4.7_xxx.

```
Etpu_Resolver_Ip_ConfigType resolver_config =
{
    ETPU_RESOLVER_IP_SEMAPHORE_0,
    ETPU_RESOLVER_IP_OPTIONS_CALCULATION_ON +
    ETPU_RESOLVER_IP_OPTIONS_DIAG_MEASURES_ON +
    ETPU_RESOLVER_IP_OPTIONS_DIAG_MEASURES_ON +
    ETPU_RESOLVER_IP_OPTIONS_EXC_ADAPTATION_ON +
    ETPU_RESOLVER_IP_OPTIONS_EXC_GENERATION_ON +
    ETPU_RESOLVER_IP_OPTIONS_RDC_CHECKER_ON, /* options */
    NSEC2TCR1(100000),                       /* excitation_period */
    SFRACT24(0.070710678),                   /* ato_p_gain */
    SFRACT24(0.0025),                        /* ato_i_gain */
    SFRACT24(0.00000),                       /* exc_p_gain */
    SFRACT24(0.00012),                       /* exc_i_gain */
    SFRACT24(0.9)                            /* q_ewma_speed */
};
```

This project has adopted the latest eTPU RTD to implement software resolver, shown in the below figure.

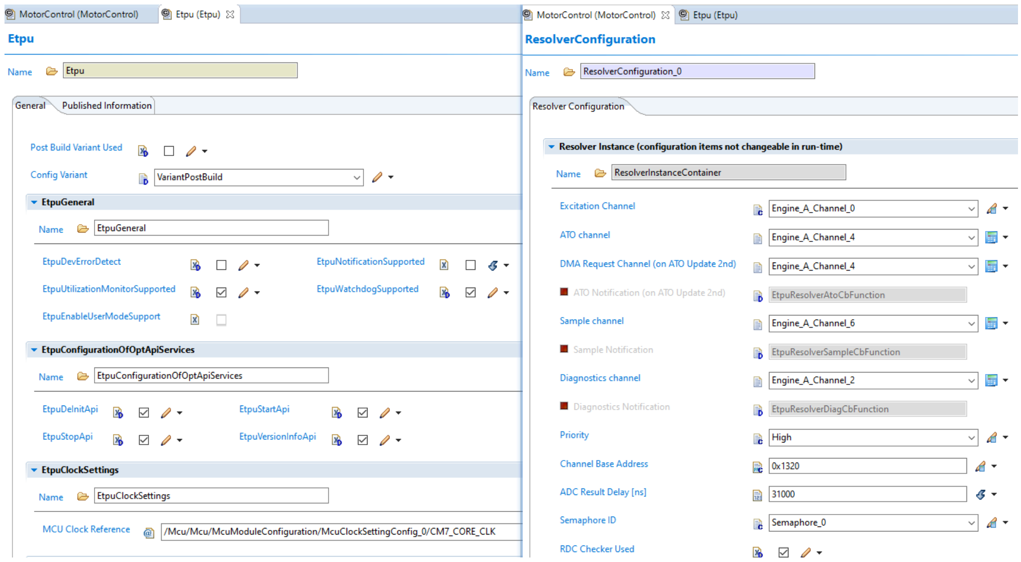


Figure 8. eTPU configuration

4.3.4 SDADC configuration

The Sine and Cosine analog feedback signals must 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 sigma-delta ADC (SDADC).

Two SDADC modules are used to continuously sample the Sine and Cosine signals in parallel, which are feedback by one motor. 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.

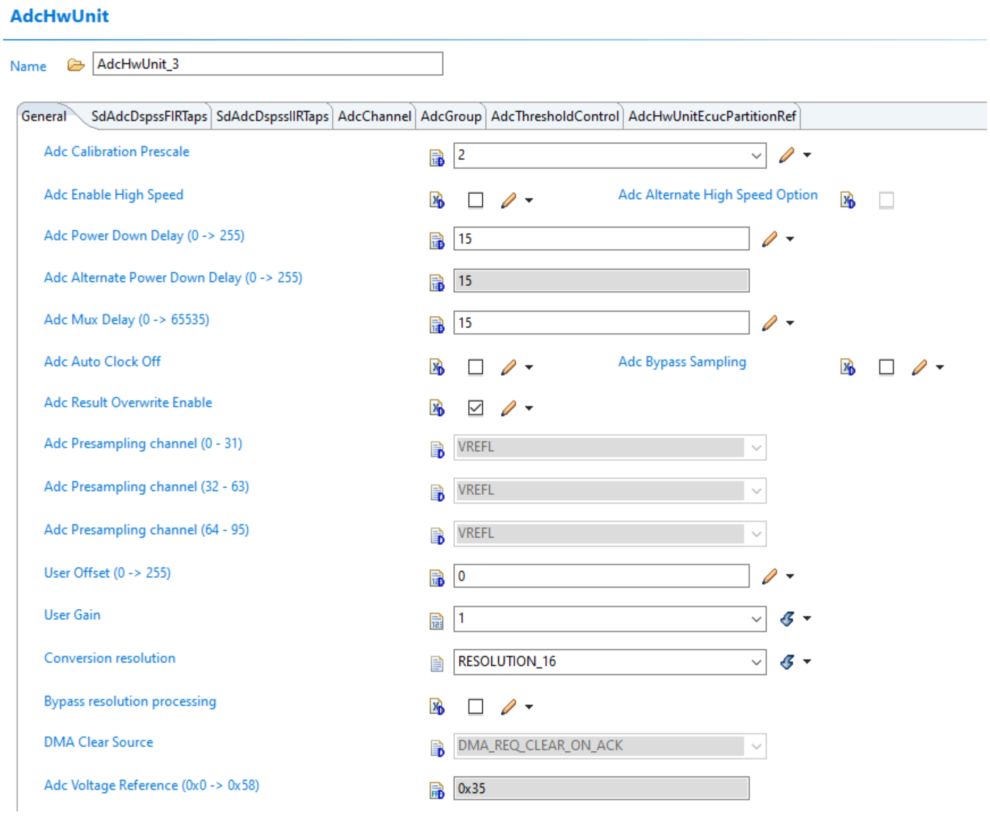
Table 2. SDADC configuration for resolver

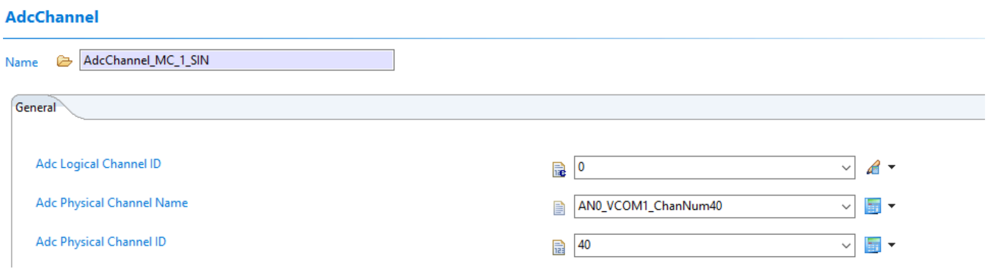
Configuration item	Value
SDADC clock	80 MHz
SDADC sampling frequency	40 MHz
ADC decimation rate	125
Result output data rate	40 MHz / 125 = 320 kHz
Input mode	Single-ended
FIFO size	16 words
FIFO threshold	8
DMA request on FIFO full	selected and enabled

Note: *In this application, a single-ended mode is configured for SDADC to sense resolver feedback signals (sine and cosine). This results from the Low voltage power stage circuitry allowing single-ended measurement for position feedback. However, it is recommended to use differential configuration for SDADC (and corresponding circuitry) when using SDADCs for eTPU Resolver signal digitization. This brings the advantage of noise rejection.*

Note: *The SDADC in the S32K396 platform removes the internal high-pass filter component, and a new IP COOLFLUX DSP SUBSYSTEM (DSPSS) is added for signal processing.*

SDADC_0 and SDADC_1 are used for Sine and Cosine signals respectively in single motor application. The configuration parameters are as follows, the same configuration parameters for SDADC_0 and SDADC_1.





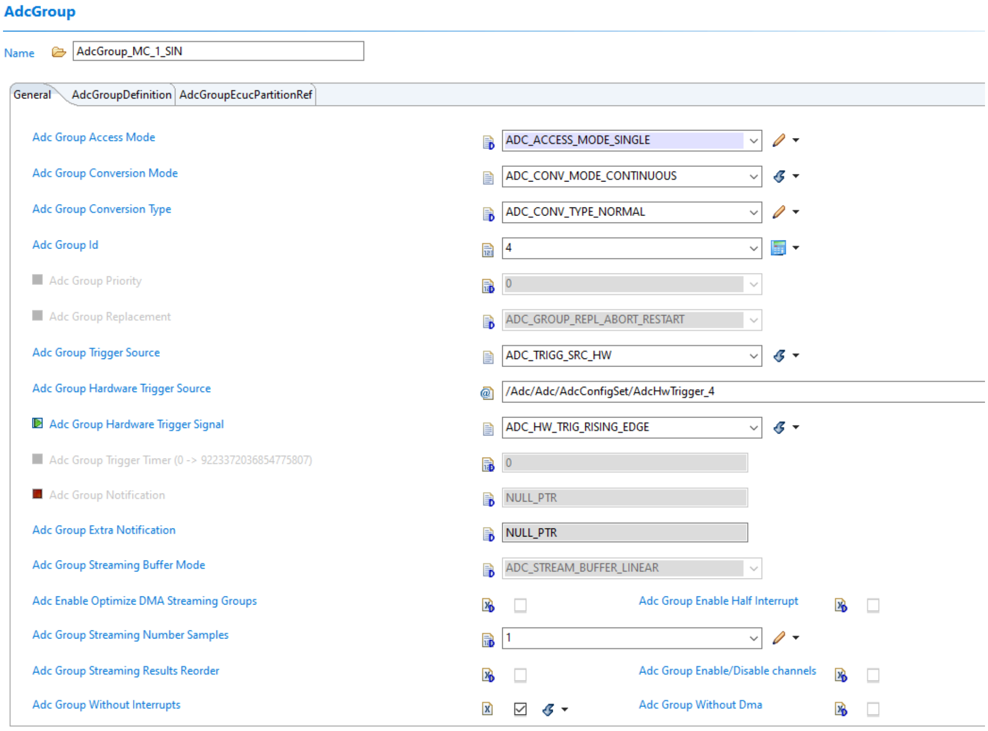


Figure 9. SDADC configuration

The below initialization process is needed by SDADC:

```
#define SDADC_RSER_CONF (SDADC_RSER_DFFDIRE_MASK | SDADC_RSER_DFFDIRS_MASK)
/* init SDADC_0 */
Sdadc_Ip_EnableHwTrigger(SDADC_INSTANCE_0);
Sdadc_Ip_EnableDmaEvents(SDADC_INSTANCE_0, SDADC_RSER_CONF);
/* set trigger SDADC_0_SW */
Sdadc_Ip_SetHwTrigger(SDADC_INSTANCE_0, SDADC_IP_TRIGGER_RISING_EDGE,
                      (Sdadc_Ip_TriggerSelectType)SDADC_SW_0_TrigNum0);
/* init SDADC_1 */
Sdadc_Ip_EnableHwTrigger(SDADC_INSTANCE_1);
Sdadc_Ip_EnableDmaEvents(SDADC_INSTANCE_1, SDADC_RSER_CONF);
/* set trigger SDADC_0_SW */
Sdadc_Ip_SetHwTrigger(SDADC_INSTANCE_1, SDADC_IP_TRIGGER_RISING_EDGE,
                      (Sdadc_Ip_TriggerSelectType)SDADC_SW_0_TrigNum0);
```

Note: *When enabled DSPSS, the SDADC converted data is transferred to the data memory (XMEM) of DSPSS with 16-bit aligned. And the data for the eTPU resolver should be obtained from DSPSS RAM. For more details, see S32K396 Reference Manual.*

If a dual-motor application is enabled, some modifications to the SDADC configuration are required. The second motor must enable SDADC_2 and SDADC_3. The configuration of SDADC_2 and SDADC_3 are the same as SDADC_0 and SDADC_1. The second motor needs the below initialization process:

```
/* init SDADC_2 */
Sdadc_Ip_EnableHwTrigger(SDADC_INSTANCE_2);
Sdadc_Ip_EnableDmaEvents(SDADC_INSTANCE_2, SDADC_RSER_CONF);
/* set trigger SDADC_2_SW */
Sdadc_Ip_SetHwTrigger(SDADC_INSTANCE_2, SDADC_IP_TRIGGER_RISING_EDGE,
(Sdadc_Ip_TriggerSelectType)SDADC_SW_2_TrigNum2);
/* init SDADC_3 */
```

```
Sdadc_Ip_EnableHwTrigger(SDADC_INSTANCE_3);
Sdadc_Ip_EnableDmaEvents(SDADC_INSTANCE_3, SDADC_RSER_CONF);
/* set trigger SDADC_2_SW */
Sdadc_Ip_SetHwTrigger(SDADC_INSTANCE_3, SDADC_IP_TRIGGER_RISING_EDGE,
                      (Sdadc_Ip_TriggerSelectType)SDADC_SW_2_TrigNum2);
```

4.3.5 DSP configuration

DSPSS is a signal processing block. It is built to implement FIR filtering capability for motor resolver application. CoolFlux DSP is the main processing block of this subsystem, which provides multithreaded processing capability (up to 4 threads). The subsystem has dedicated program and data memories for parallel access. This block processes sample obtained from the SDADC interface and then keeps the processed data in a local SRAM, which the core or DMA accesses. Figure 10 shows the diagram of the DSPSS. For details, see S32K396 Reference Manual.

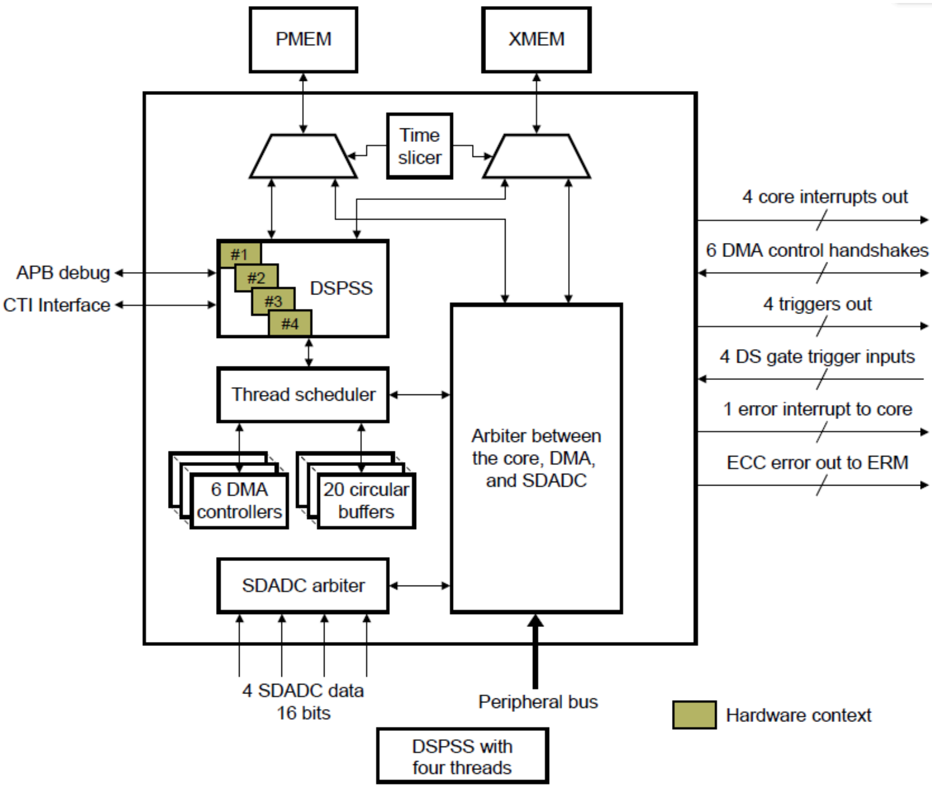
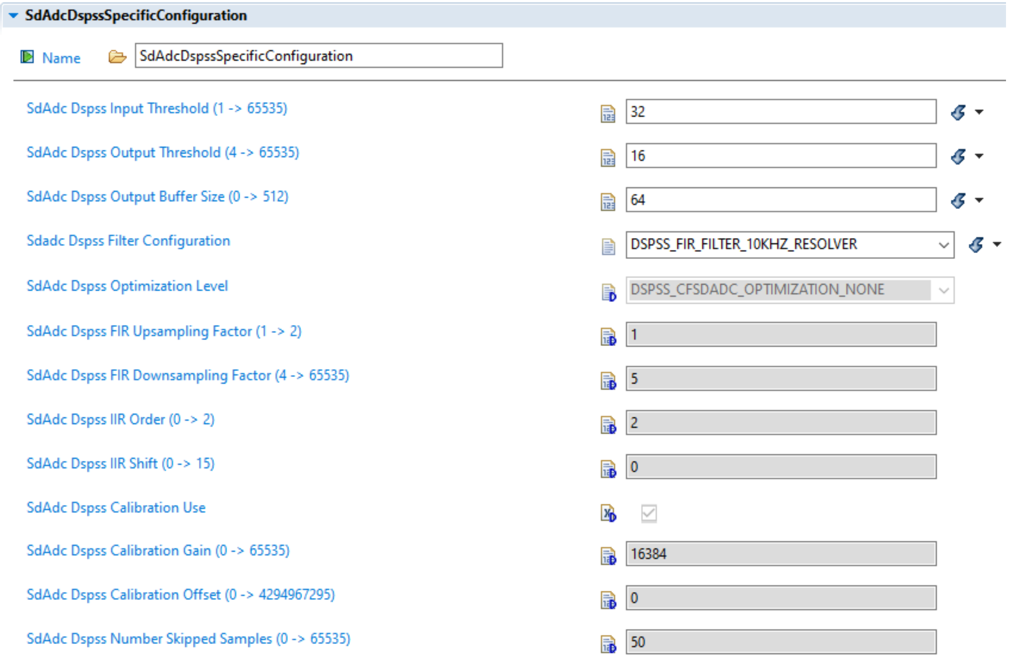
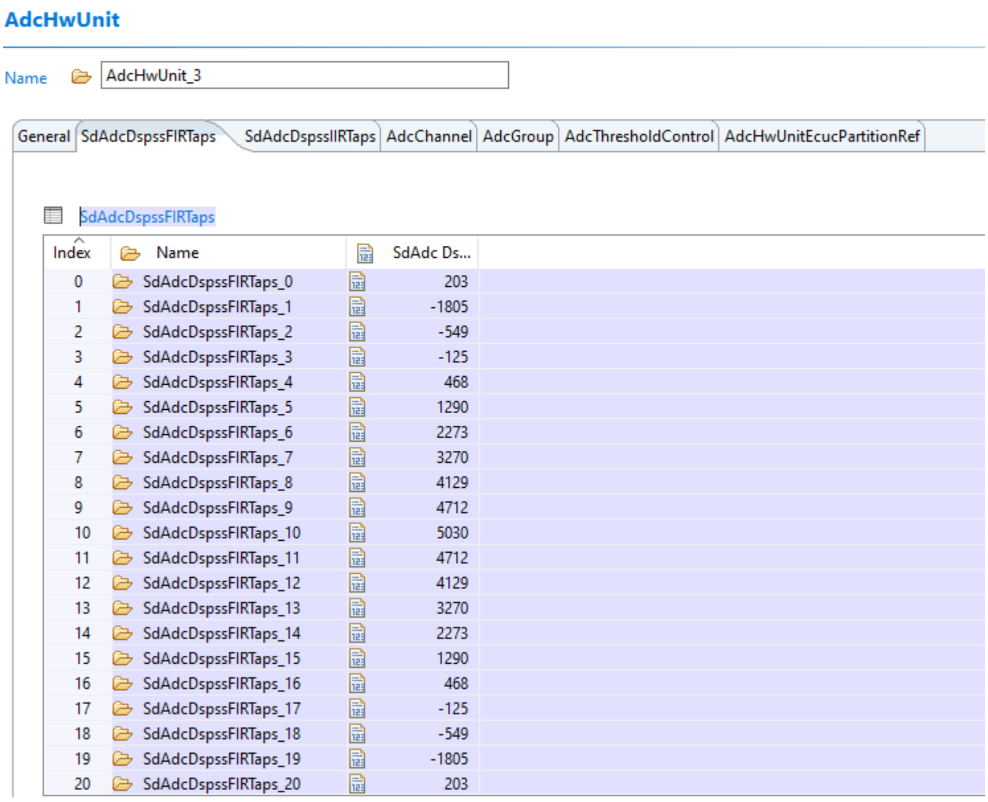


Figure 10. DSPSS block diagram

Below is the configuration of DSPSS. 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.


SdAdcDspssSpecificConfiguration
Name
SdAdcDspssSpecificConfiguration
SdAdc Dspss Input Threshold (1 -> 65535)
32
SdAdc Dspss Output Threshold (4 -> 65535)
16
SdAdc Dspss Output Buffer Size (0 -> 512)
64
Sdadc Dspss Filter Configuration
DSPSS_FIR_FILTER_10KHZ_RESOLVER
SdAdc Dspss Optimization Level
DSPSS_CFSDADC_OPTIMIZATION_NONE
SdAdc Dspss FIR Upsampling Factor (1 -> 2)
1
SdAdc Dspss FIR Downsampling Factor (4 -> 65535)
5
SdAdc Dspss IIR Order (0 -> 2)
2
SdAdc Dspss IIR Shift (0 -> 15)
0
SdAdc Dspss Calibration Use
SdAdc Dspss Calibration Gain (0 -> 65535)
16384
SdAdc Dspss Calibration Offset (0 -> 4294967295)
0
SdAdc Dspss Number Skipped Samples (0 -> 65535)
50


AdcHwUnit
Name
AdcHwUnit_3
General
SdAdcDspssFIRTaps
SdAdcDspssIIRTaps
AdcChannel
AdcGroup
AdcThresholdControl
AdcHwUnitEcucPartitionRef
SdAdcDspssFIRTaps
Index
Name
SdAdc Ds...
0 SdAdcDspssFIRTaps_0 203
1 SdAdcDspssFIRTaps_1 -1805
2 SdAdcDspssFIRTaps_2 -549
3 SdAdcDspssFIRTaps_3 -125
4 SdAdcDspssFIRTaps_4 468
5 SdAdcDspssFIRTaps_5 1290
6 SdAdcDspssFIRTaps_6 2273
7 SdAdcDspssFIRTaps_7 3270
8 SdAdcDspssFIRTaps_8 4129
9 SdAdcDspssFIRTaps_9 4712
10 SdAdcDspssFIRTaps_10 5030
11 SdAdcDspssFIRTaps_11 4712
12 SdAdcDspssFIRTaps_12 4129
13 SdAdcDspssFIRTaps_13 3270
14 SdAdcDspssFIRTaps_14 2273
15 SdAdcDspssFIRTaps_15 1290
16 SdAdcDspssFIRTaps_16 468
17 SdAdcDspssFIRTaps_17 -125
18 SdAdcDspssFIRTaps_18 -549
19 SdAdcDspssFIRTaps_19 -1805
20 SdAdcDspssFIRTaps_20 203

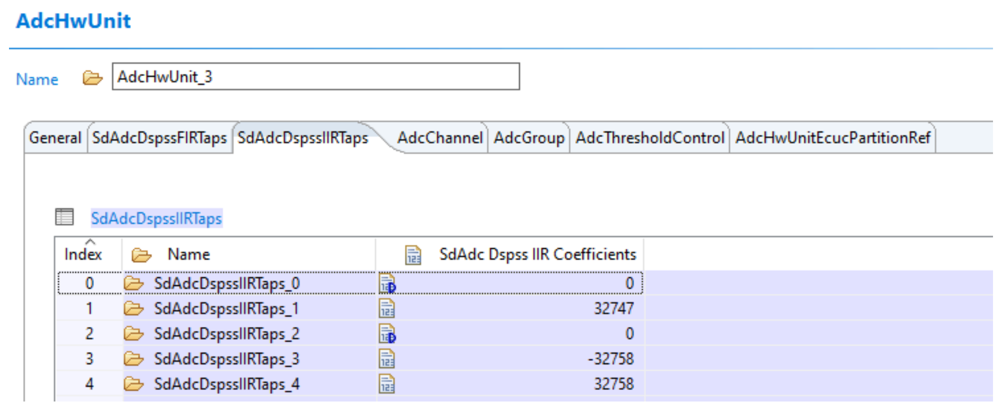


Figure 11. DSPSS configuration

To enable CoolFlux DSP, must load firmware to DSPSS's program memory(PMEM) and load the default configuration into the XMEM. The DSPSS's firmware and threads' configuration are done when calling the SDADC initialization. During the initialization, the DSPSS firmware is loaded into its RAM, the thread parameters are configured, and the thread is initialized. However, the threads are only ready to run.

There are 4 threads in the DSPSS, and the first two threads are used in the first motor. 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 RAM.

```
static void DSPSS_DMA_Config(void)
{
    /* configure core dma, transfer data which in dsp memory to etpu memory */
    /* each SDADC channel will call DSPSS_ThreadsInitialize function, it will
 causes DSP_COMP_LOW assert. */
    Dma_Ip_LogicChannelScatterGatherListType Dma_Cfg[2] =
    {
        {.Param = DMA_IP_CH_SET_SOURCE_ADDRESS,},
        {.Param = DMA_IP_CH_SET_DESTINATION_ADDRESS,}
    };
    /* Dspss output DMA configuration */
    Dma_Cfg[0].Value = DSPSS_ThreadGetOutputBufferStart(DSPSS_THREAD_ID0);
#if DEBUG_DSPSS
    Dma_Cfg[1].Value = Dspss_Sin_Out;
#else
    Dma_Cfg[1].Value = (uint32)RESOLVER_INSTANCE.pSignalsPba;
#endif
    Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_1, Dma_Cfg, 2);
    /* Dspss output DMA configuration */
    Dma_Cfg[0].Value = DSPSS_ThreadGetOutputBufferStart(DSPSS_THREAD_ID1);
#if DEBUG_DSPSS
    Dma_Cfg[1].Value = Dspss_Cos_Out;
#else
    Dma_Cfg[1].Value = (uint32)RESOLVER_INSTANCE.pSignalsPba + 0x80;
#endif
    Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_2, Dma_Cfg, 2);
    /* Resolver Excitation DMA configuration */
    Dma_Cfg[0].Value = (uint32)sResolverExcCmdDspBuf;
    Dma_Cfg[1].Value = (uint32)&pEtpu_Ip_Regs-
>CHAN[RESOLVER_INSTANCE.u8ChanNumAto].HSRR.R;
    Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_3, Dma_Cfg, 2);
}
```

Similarly, the DMA configuration for data transfer between the DSPSS and the eTPU RAM for the second motor.

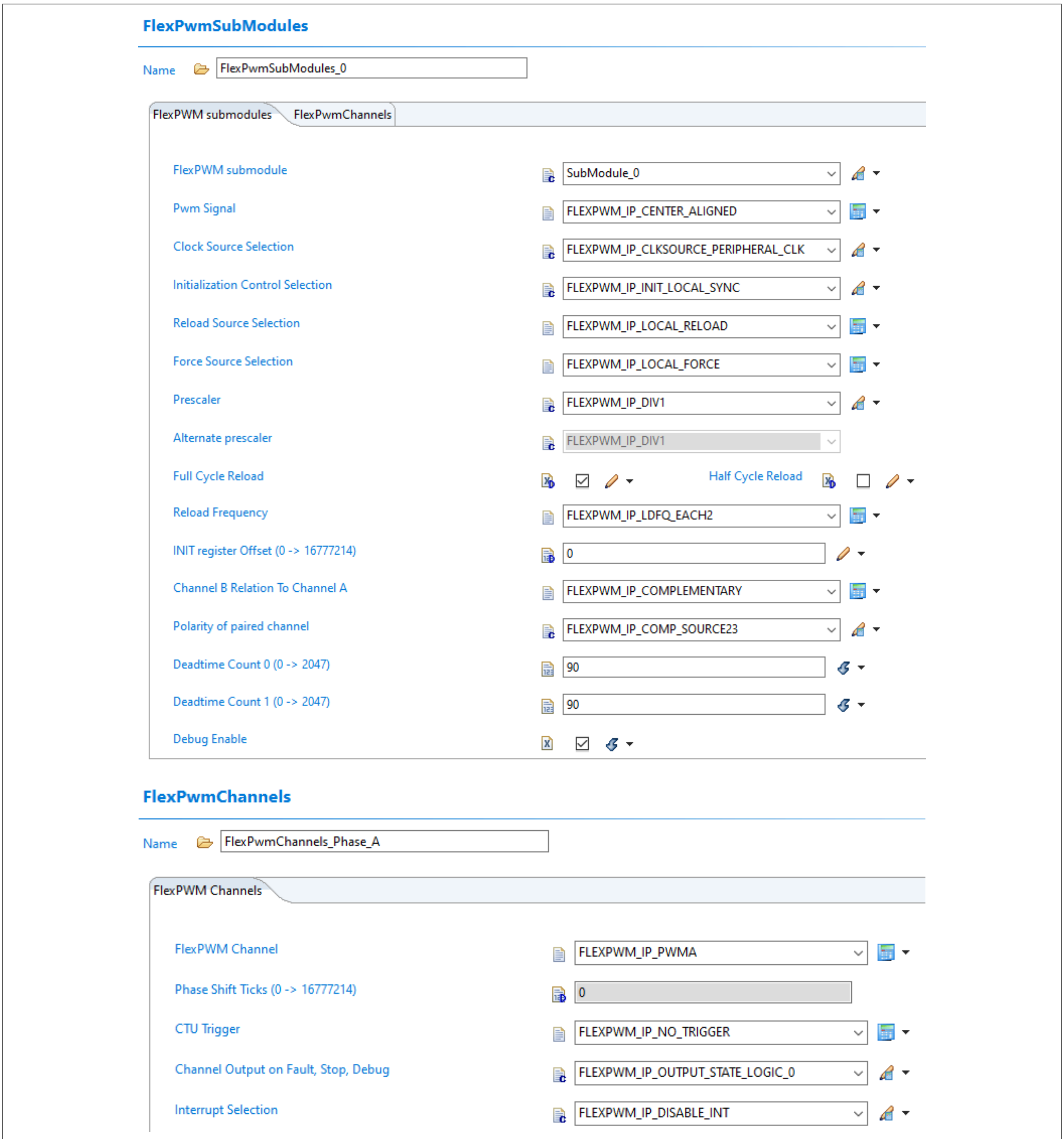
```
static void DSPSS_DMA_Config(void)
{
    /* configure core dma, transfer data which in dsp memory to etpu memory */
    /* each SDADC channel will call DSPSS_ThreadsInitialize function, it will
 causes DSP_COMP_LOW assert. */
    Dma_Ip_LogicChannelScatterGatherListType Dma_Cfg[2] =
    {
        {.Param = DMA_IP_CH_SET_SOURCE_ADDRESS,},
        {.Param = DMA_IP_CH_SET_DESTINATION_ADDRESS,}
    };
    /* Dspss output DMA configuration */
    Dma_Cfg[0].Value = DSPSS_ThreadGetOutputBufferStart(DSPSS_THREAD_ID2);
#if DEBUG_DSPSS
    Dma_Cfg[1].Value = Dspss_Sin_Out;
#else
    Dma_Cfg[1].Value = (uint32)RESOLVER_INSTANCE.pSignalsPba;
#endif
    Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_1, Dma_Cfg, 2);
    /* Dspss output DMA configuration */
    Dma_Cfg[0].Value = DSPSS_ThreadGetOutputBufferStart(DSPSS_THREAD_ID3);
#if DEBUG_DSPSS
    Dma_Cfg[1].Value = Dspss_Cos_Out;
#else
    Dma_Cfg[1].Value = (uint32)RESOLVER_INSTANCE.pSignalsPba + 0x80;
#endif
    Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_2, Dma_Cfg, 2);
    /* Resolver Excitation DMA configuration */
    Dma_Cfg[0].Value = (uint32)sResolverExcCmdDspBuf;
    Dma_Cfg[1].Value = (uint32)&pEtpu_Ip_Regs-
>CHAN[RESOLVER_INSTANCE.u8ChanNumAto].HSRR.R;
    Dma_Ip_SetLogicChannelTransferList(DMA_LOGIC_CH_3, Dma_Cfg, 2);
}
```

Then, enable the threads by calling ***DSPSS_ThreadEnable()***. In this project, the ***DSPSS_ThreadEnable()*** is called in the function ***Sdadc_Ip_EnableHwTrigger()***.

Note: *All used threads must be configured and initialized first before starting any thread to run. Otherwise, there might be problems or issues. Also, the threads should be started to run before the connected SDADC instances. If not, the phase shift might be found in the outputs.*

4.3.6 eFlexPWM configuration

The s32k396 has two eFlexPWM instances. Each eFlexPWM contains 4 PWM submodules, each set up to control a single half-bridge power stage. In the single motor project, the first eFlexPWM instance is used to control each phase of a three-phase motor on or off. Each submodule of eFlexPWM controls the three-phase circuit of the motor respectively. The configuration of submodule 0 and 1 are as follows, the same configuration parameters for submodule 1 and submodule 2. In the second motor, the eFlexPWM configuration is the same as in the first motor.


FlexPwmSubModules
Name FlexPwmSubModules_0
FlexPWM submodules
FlexPwmChannels
FlexPWM submodule SubModule_0
Pwm Signal FLEXPWM_IP_CENTER_ALIGNED
Clock Source Selection FLEXPWM_IP_CLKSOURCE_PERIPHERAL_CLK
Initialization Control Selection FLEXPWM_IP_INIT_LOCAL_SYNC
Reload Source Selection FLEXPWM_IP_LOCAL_RELOAD
Force Source Selection FLEXPWM_IP_LOCAL_FORCE
Prescaler FLEXPWM_IP_DIV1
Alternate prescaler FLEXPWM_IP_DIV1
Full Cycle Reload
Half Cycle Reload
Reload Frequency FLEXPWM_IP_LDFQ_EACH2
INIT register Offset (0 -> 16777214) 0
Channel B Relation To Channel A FLEXPWM_IP_COMPLEMENTARY
Polarity of paired channel FLEXPWM_IP_COMP_SOURCE23
Deadtime Count 0 (0 -> 2047) 90
Deadtime Count 1 (0 -> 2047) 90
Debug Enable
FlexPwmChannels
Name FlexPwmChannels_Phase_A
FlexPWM Channels
FlexPWM Channel FLEXPWM_IP_PWMA
Phase Shift Ticks (0 -> 16777214) 0
CTU Trigger FLEXPWM_IP_NO_TRIGGER
Channel Output on Fault, Stop, Debug FLEXPWM_IP_OUTPUT_STATE_LOGIC_0
Interrupt Selection FLEXPWM_IP_DISABLE_INT

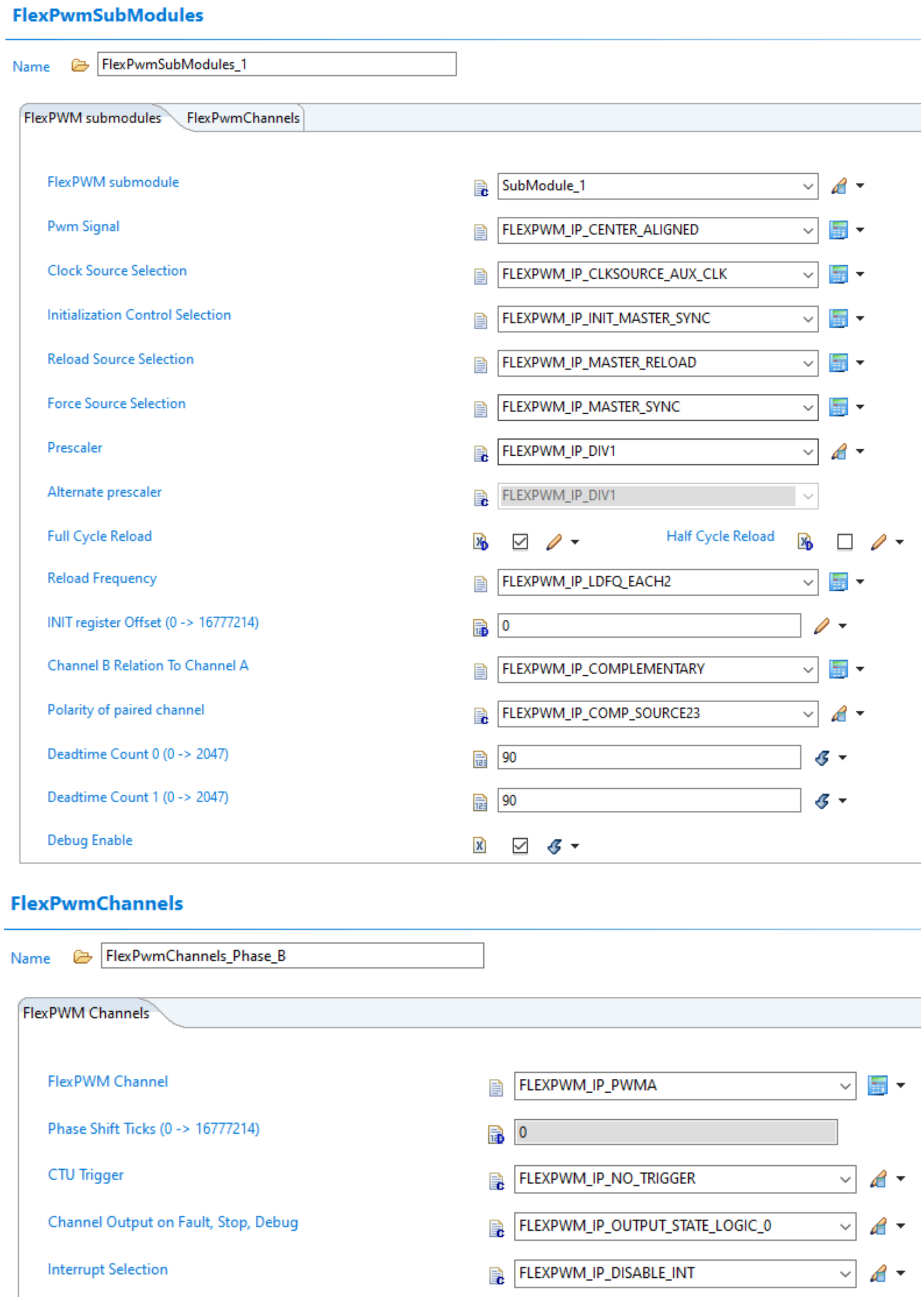


Figure 12. eFlexPWM configuration

eFlexPWM also provides fault channel support. Overcurrent and overvoltage are routed to eFlexPWM instance 0 from Pre-Driver. When the Pre-Driver triggers an OC or OV signal to eFlexPWM, the PWM generator continues to run, but the fault decoder forces the mapped out pins status based on fault configuration. Fault configuration includes eFlexPWM general fault configuration and fault configuration of each submodule.

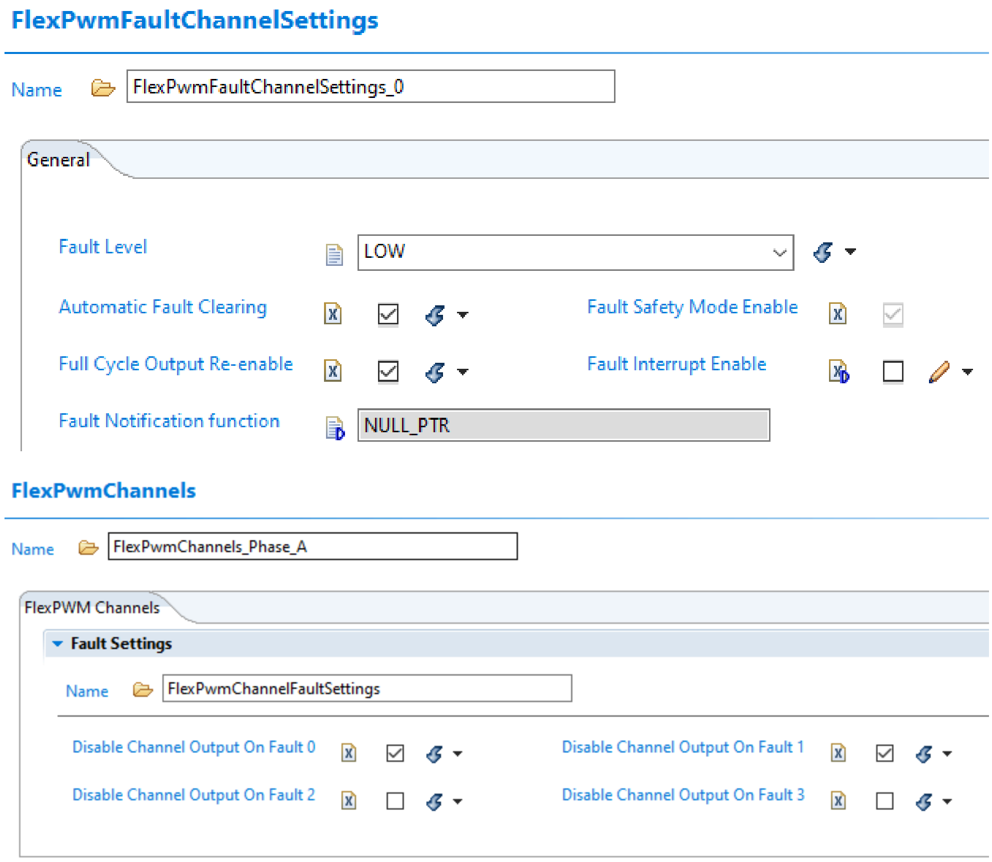


Figure 13. eFlexPWM fault configuration

4.3.7 ADC configuration

S32K396 has 7 ADC instances, which have 8 channels with 15-bit resolution. In one motor situation, three ADC instances are used to sample simultaneously Phase A, Phase B, and Phase C currents and DC bus voltage, with the trigger signal coming from eFlexPWM via TRGMUX. This is achieved by using 1 BCTU (Body Cross Trigger Unit) instances. The first BCTU can only support up to 3 ADCs, the second BCTU can support the remaining 4 ADCs.

The TRGMUX supports the triggering scheme between peripherals. In this project, the trigger signal from eFlexPWM must be routed to the BCTU. *Figure 14* shows the configuration of TRGMUX.

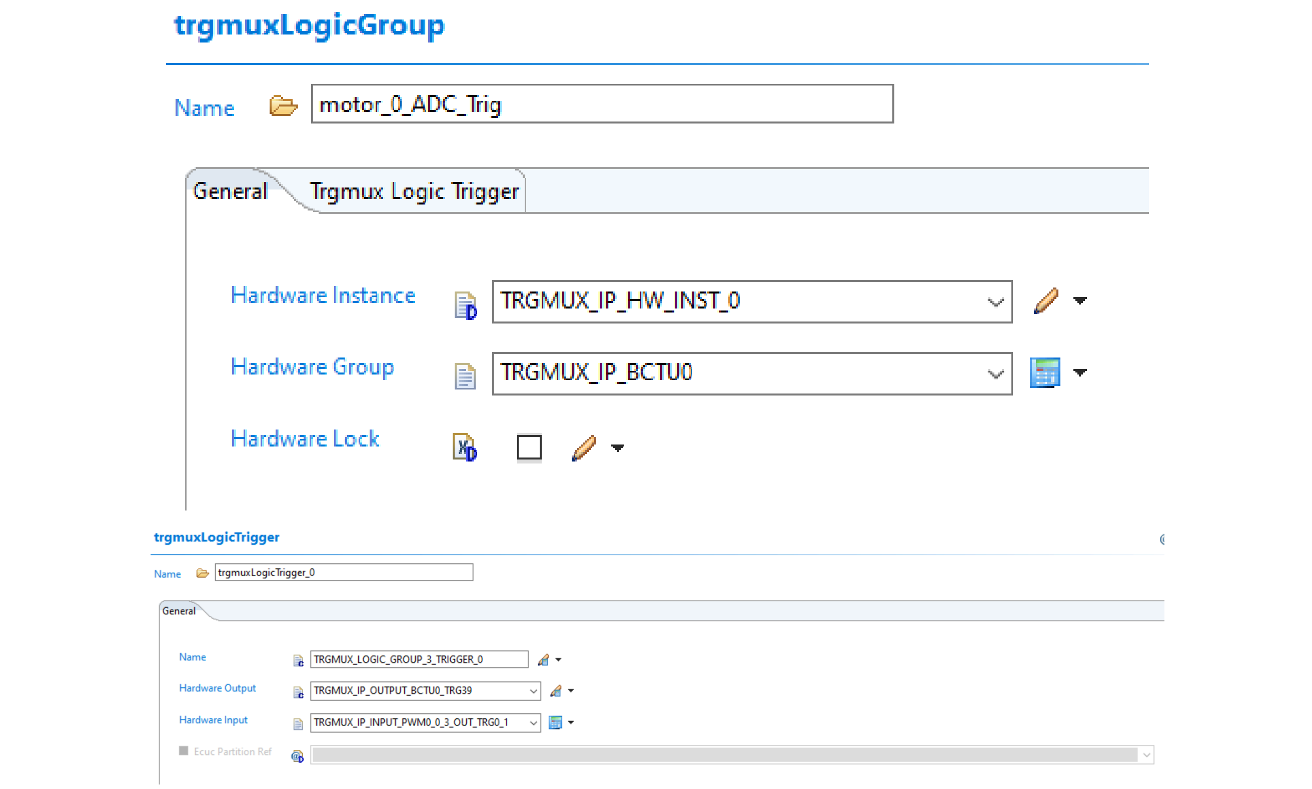
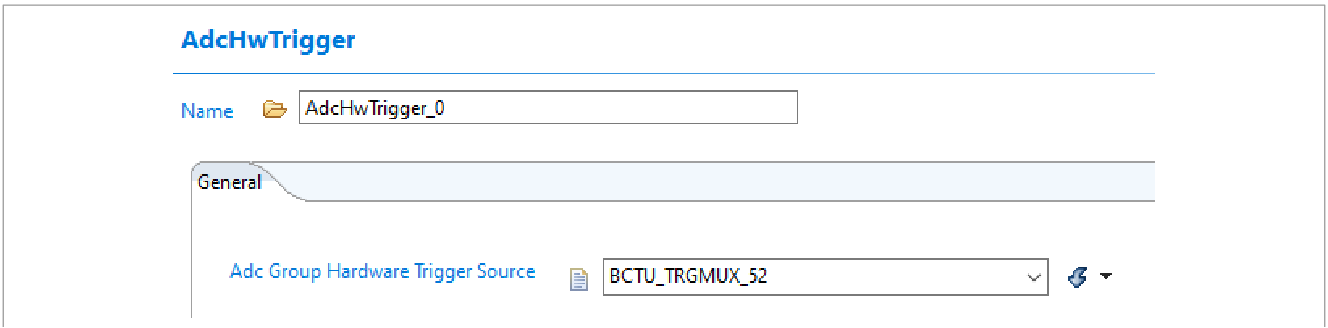
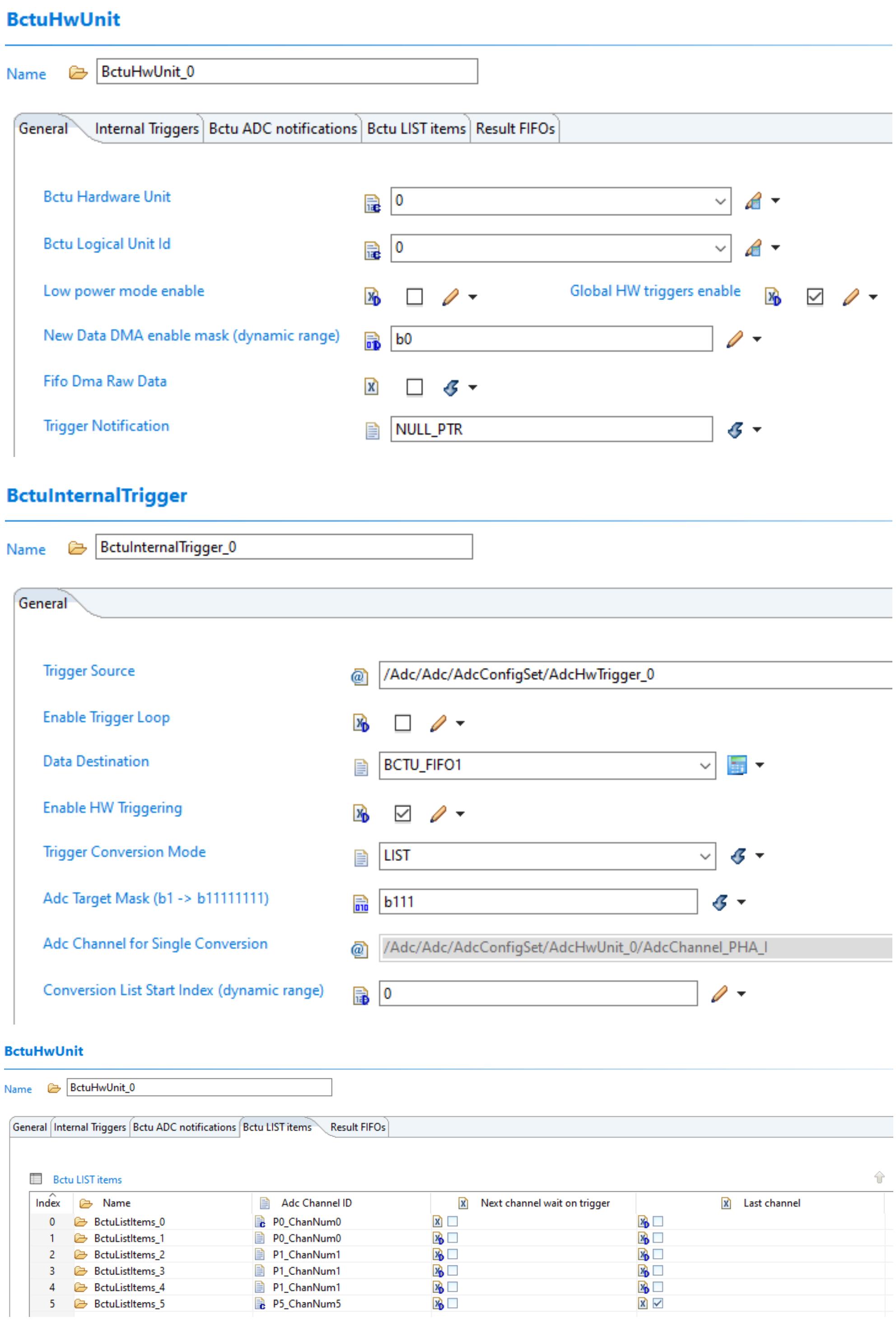


Figure 14. TRGMUX configuration

The BCTU can support associated ADCs that trigger simultaneously. Then, the ways to obtain ADCs conversion result like directly accessing ADCs result register, read the BCTU FIFO through DMA, and read 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.




BctuHwUnit
Name
BctuHwUnit_0
General
Internal Triggers
Bctu ADC notifications
Bctu LIST items
Result FIFOs
Bctu Hardware Unit
0
Bctu Logical Unit Id
0
Low power mode enable
Global HW triggers enable
New Data DMA enable mask (dynamic range)
b0
Fifo Dma Raw Data
Trigger Notification
NULL_PTR
BctuInternalTrigger
Name
BctuInternalTrigger_0
General
Trigger Source
/Adc/Adc/AdcConfigSet/AdcHwTrigger_0
Enable Trigger Loop
Data Destination
BCTU_FIFO1
Enable HW Triggering
Trigger Conversion Mode
LIST
Adc Target Mask (b1 -> b11111111)
b111
Adc Channel for Single Conversion
/Adc/Adc/AdcConfigSet/AdcHwUnit_0/AdcChannel_PHA_I
Conversion List Start Index (dynamic range)
0
BctuHwUnit
Name
BctuHwUnit_0
General
Internal Triggers
Bctu ADC notifications
Bctu LIST items
Result FIFOs
Bctu LIST items
Index
Name
Adc Channel ID
Next channel wait on trigger
Last channel
0
BctuListItems_0
P0_ChanNum0
1
BctuListItems_1
P0_ChanNum0
2
BctuListItems_2
P1_ChanNum1
3
BctuListItems_3
P1_ChanNum1
4
BctuListItems_4
P1_ChanNum1
5
BctuListItems_5
P5_ChanNum5

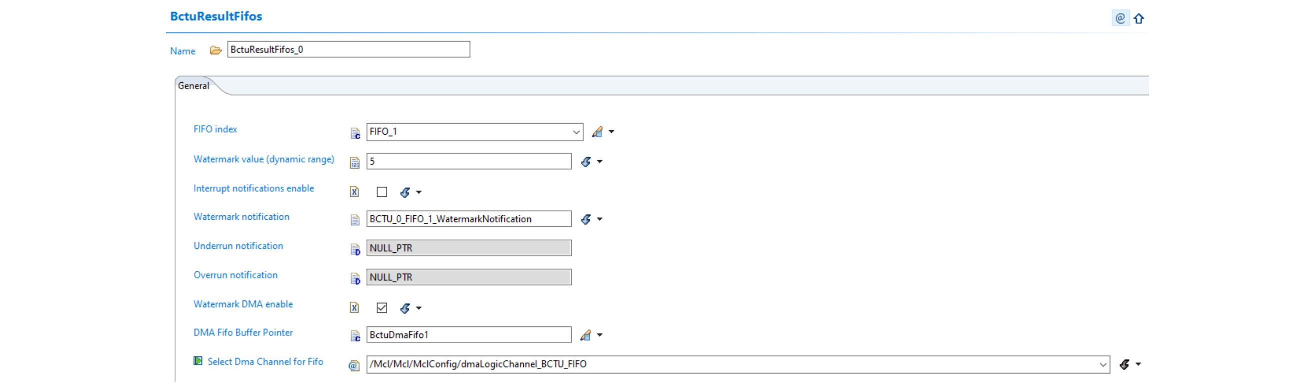
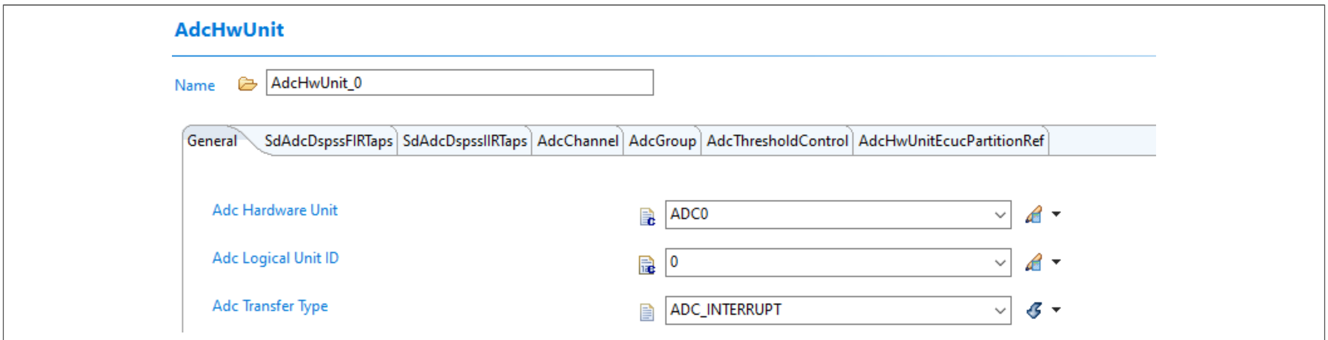


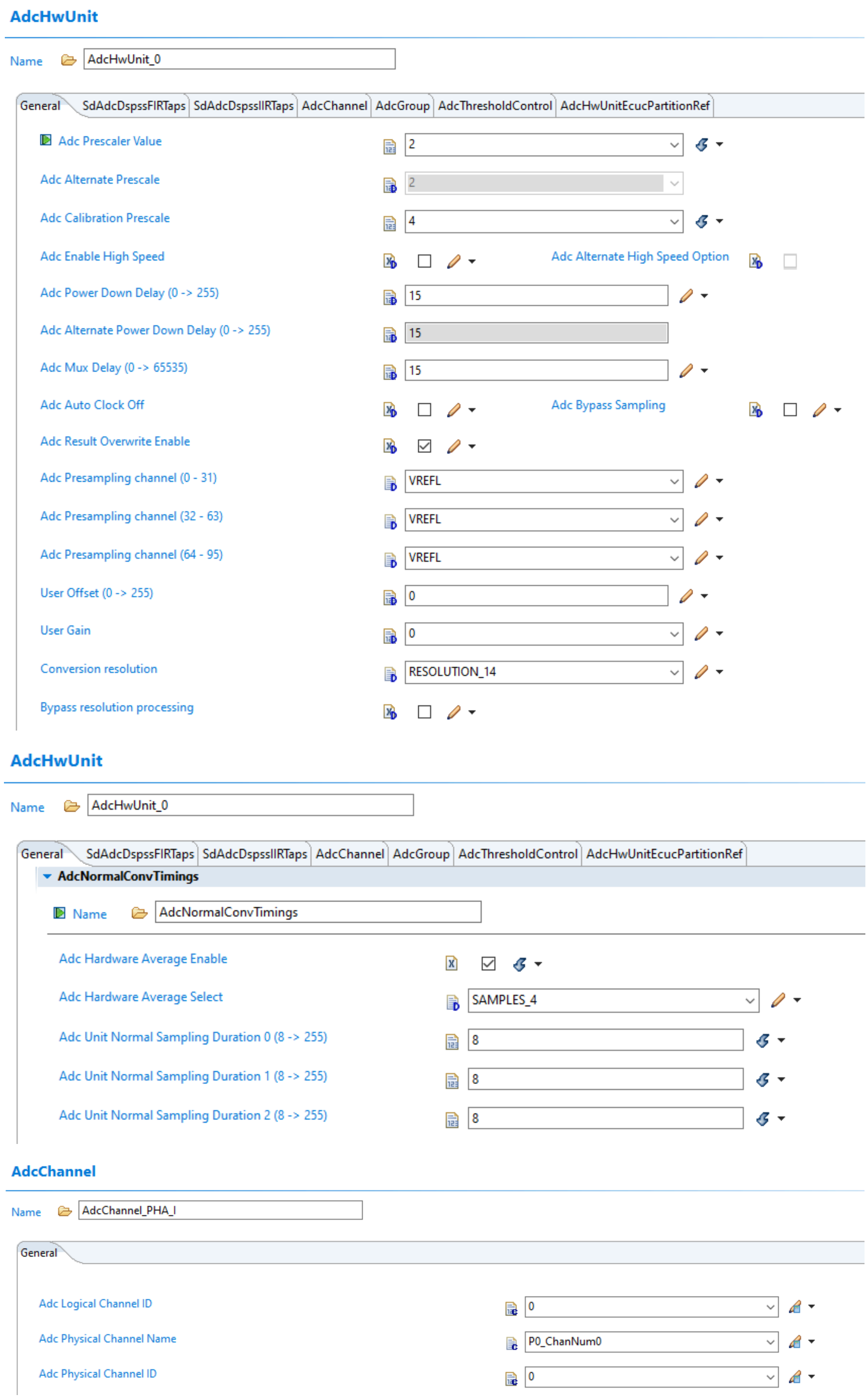
Figure 15. BCTU configuration

The below is the initialization of the first BCTU instance:

```
/* Initialize BCTU */
BCTU_Init();
static void BCTU_Init(void)
{
    uint8 instance = 0;
#if MOTOR_0
    /* Initialize BCTU0. */
    /* Disable global triggers. */
    Bctu_Ip_SetGlobalTriggerEn(instance, FALSE);
    Bctu_Ip_Init(instance, &BctuIpConfigControlMode_0);
    /* Enable global triggers. */
    Bctu_Ip_SetGlobalTriggerEn(instance, TRUE);
#elif MOTOR_1
    /* Initialize BCTU1. */
    instance = 1;
    /* Disable global triggers. */
    Bctu_Ip_SetGlobalTriggerEn(instance, FALSE);
    Bctu_Ip_Init(instance, &BctuIpConfigControlMode_0);
    /* Enable global triggers. */
    Bctu_Ip_SetGlobalTriggerEn(instance, TRUE);
#else
    #error "error bctu instance"
#endif
}   /* BCTU_Init */
```

The following gives the configuration for ADCs. Taking one ADC configuration as an example, the others are almost similar to the example one.




AdcHwUnit
Name
AdcHwUnit_0
General
SdAdcDspssFIRTaps
SdAdcDspssIIRTaps
AdcChannel
AdcGroup
AdcThresholdControl
AdcHwUnitEcucPartitionRef
Adc Prescaler Value
2
Adc Alternate Prescale
2
Adc Calibration Prescale
4
Adc Enable High Speed
Adc Alternate High Speed Option
Adc Power Down Delay (0 -> 255)
15
Adc Alternate Power Down Delay (0 -> 255)
15
Adc Mux Delay (0 -> 65535)
15
Adc Auto Clock Off
Adc Bypass Sampling
Adc Result Overwrite Enable
Adc Presampling channel (0 - 31)
VREFL
Adc Presampling channel (32 - 63)
VREFL
Adc Presampling channel (64 - 95)
VREFL
User Offset (0 -> 255)
0
User Gain
0
Conversion resolution
RESOLUTION_14
Bypass resolution processing
AdcHwUnit
Name
AdcHwUnit_0
General
SdAdcDspssFIRTaps
SdAdcDspssIIRTaps
AdcChannel
AdcGroup
AdcThresholdControl
AdcHwUnitEcucPartitionRef
AdcNormalConvTimings
Name
AdcNormalConvTimings
Adc Hardware Average Enable
Adc Hardware Average Select
SAMPLES_4
Adc Unit Normal Sampling Duration 0 (8 -> 255)
8
Adc Unit Normal Sampling Duration 1 (8 -> 255)
8
Adc Unit Normal Sampling Duration 2 (8 -> 255)
8
AdcChannel
Name
AdcChannel_PHA_I
General
Adc Logical Channel ID
0
Adc Physical Channel Name
P0_ChanNum0
Adc Physical Channel ID
0

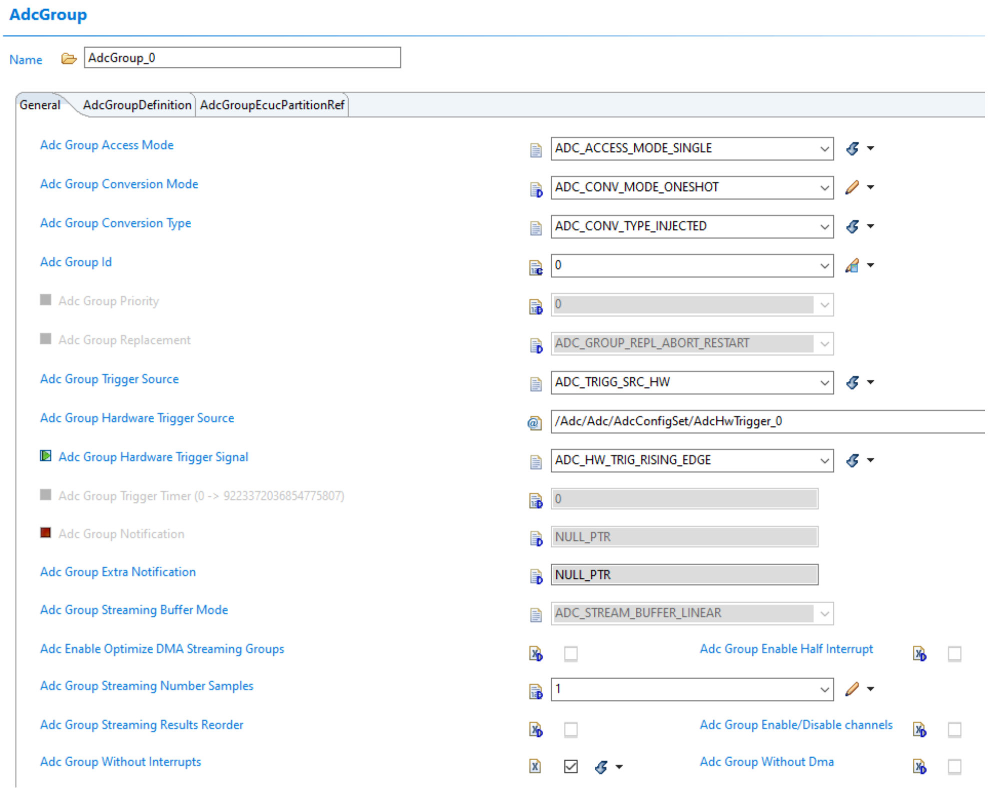


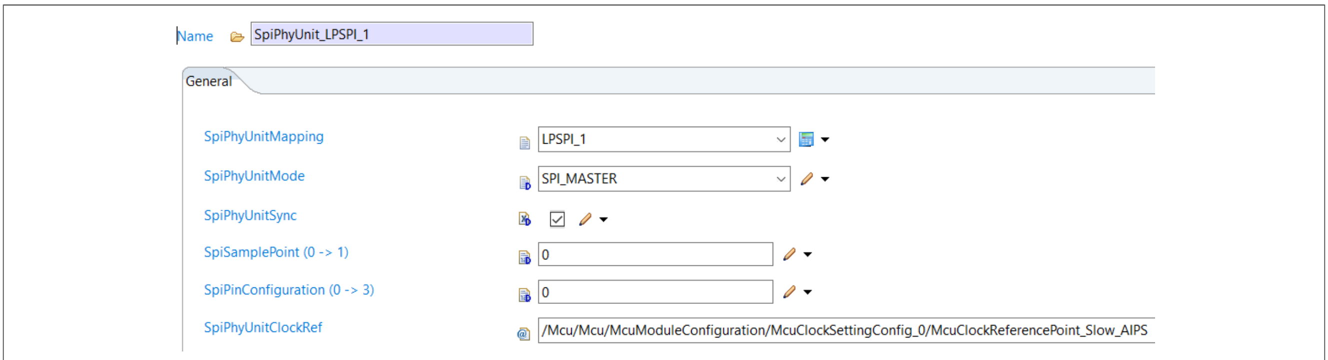
Figure 16. ADC configuration

```
static Std_ReturnType HAL_ADC_SARADC_Init(HAL_ADC_InstanceType adcInstance)
{
    DevAssert(adcInstance < HAL_ADC_INSTANCE_COUNT);
    (void)adcInstance;
    uint8 adcIndex, adcStartIndex, adcMaxIndex, adcCnt;
#if MOTOR_0
    adcStartIndex = MOTOR_0_SARADC_INIT_START;
    adcMaxIndex = MOTOR_0_SARADC_INIT_START + MOTOR_0_SARADC_INIT_CNT - 1;
    adcCnt = AdcHwUnit_2 - AdcHwUnit_0 + 1;
#endif
#if MOTOR_1
    adcStartIndex = MOTOR_1_SARADC_INIT_START;
    adcMaxIndex = MOTOR_1_SARADC_INIT_START + MOTOR_1_SARADC_INIT_CNT - 1; /* -1
 because of for judgement is <= */
    adcCnt = MOTOR_1_SARADC_INIT_CNT;
#endif
    Std_ReturnType returnValue = E_NOT_OK;
    Adc_CalibrationStatusType calibStatus;
    ADC_Type * const pAdcBase[ADC_INSTANCE_COUNT] = IP_ADC_BASE_PTRS;
    do
    {
        Adc_Init(&Adc_Config);
        /* Calibrate all used SARADC hardware units. */
        for (adcIndex = adcStartIndex; adcIndex <= adcMaxIndex; adcIndex++)
        {
#if USER_DEF_OLD_CHIP
            /* TODO: AMSIO must be 0, unknown reason. */
            pAdcBase[adcIndex]->AMSIO = 0;
            /* TODO: CALBISTREG must be 0xFF1F70, unknown reason. */
```

```
            pAdcBase[adcIndex]->CALBISTREG = 0xFF1F70;
#endif
            Adc_Calibrate(adcIndex - SARADC_START_INDEX, &calibStatus);
#if USER_DEF_OLD_CHIP
            /* TODO: Set AMSIO to default reset value, unknown reason. */
            pAdcBase[adcIndex]->AMSIO = 0x811;
#endif
            if (calibStatus.AdcUnitSelfTestStatus != E_OK)
            {
                break;
            }
        }
        /* Set all used SARADC hardware units to be BCTU triggered only. */
        for (adcIndex = adcStartIndex; adcIndex <= adcMaxIndex; adcIndex++)
        {
            Adc_EnableCtuControlMode(adcIndex - SARADC_START_INDEX);
        }
        /* Failed to calibrate all used SARADC hardware units. */
        if (adcIndex <= adcMaxIndex)
        {
            break;
        }
        /* Initialize BCTU */
        BCTU_Init();
        returnValue = E_OK;
    } while (0);
    return returnValue;
}   /* HAL_ADC_SARADC_Init */
```

4.3.8 LPSPI configuration

The LPSPI module is used as a communication interface between S32K396 MCU and FET predriver MC33937. LPSPI is configured as Master mode with a 2 MHz baud rate derived from a 40 MHz clock. The following shows the configuration.



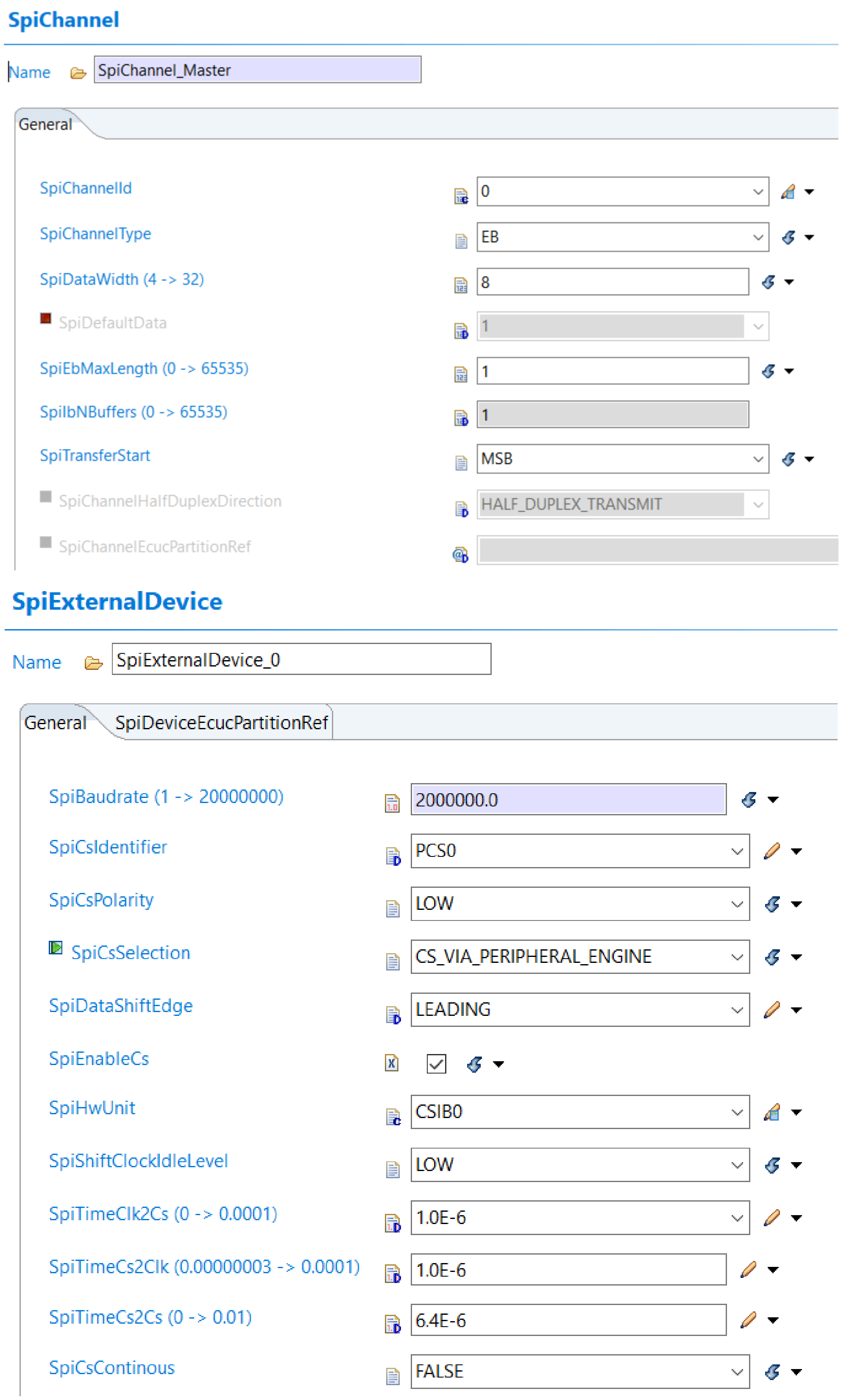


Figure 17. LPSPI configuration parameters

Note: *In the single motor in S32K396-LQFP-DC application, the LPSPI configuration is the same as above. In the dual-motor application, the second motor uses LPSPI_3 to communicate with the FET MC33937 predriver. Its configuration is the same as the first motor.*

4.3.9 DMA configuration

Within these application, 7 DMA channels are used to speed up the control loop operation and offload the CPU. DMA is used to transfer preprocessed Resolver feedback sine and cosine signals from DSPSS to eTPU data RAM and also transfer the phase currents and DC bus voltage measurements from ADC for the control loop.

All the channels used in this application are listed in the following table with respective DMA request sources.

Table 3. DMA channel usage in the application

Application	Module	Channel	Requesting source	Description
The single motor or the first motor of a dual-motor	Instance 1	6	DSPSS	Transferring filtered Resolver feedback sin signal from DSPSS
		7	DSPSS	Transferring filtered Resolver feedback cos signal from DSPSS
		8	-	This channel is linked by Instance 1 to channel 6
		3	BCTU0_FIFO0_REQUEST	Transferring Phase A current, Phase B current, Phase C current, and DC bus voltage measurement
The second motor of a dual-motor	Instance 1	24	DSPSS	Transferring filtered Resolver feedback sin signal from DSPSS
		25	DSPSS	Transferring filtered Resolver feedback cos signal from DSPSS
		26	-	This channel is linked by Instance 1 to channel 6
		20	BCTU1_FIFO0_REQUEST	Transferring Phase A current, Phase B current, Phase C current, and DC bus voltage measurement
Single motor in S32K396-LQFP-DC	Instance 1	6	DSPSS	Transferring filtered Resolver feedback sin signal from DSPSS
		7	DSPSS	Transferring filtered Resolver feedback cos signal from DSPSS
		8	-	This channel is linked by Instance 1 to channel 6
		16	BCTU1_FIFO0_REQUEST	Transferring Phase A current, Phase B current, Phase C current, and DC bus voltage measurement

Three DMA channels are used to ensure that Resolver feedback signals are delivered to eTPU data RAM and to trigger eTPU processing once all the data are transferred. DMA 1 channels 6 and 7 are configured to transfer preprocessed data (by DSPSS). DMA 1 channel 6 is configured to link DMA 1 channel 8 on subsequent transfer of constants into the eTPU Resolver ATO channel HSR register. In dual-motor application, the DMA workflow of the second motor is the same as that of the first motor. The array of the constants is as follows:

```
static const uint32 sResolverExcCmdDspBuf[2] =
{
    FS_ETPU_RESOLVER_HSR_UPDATE_1ST,
    FS_ETPU_RESOLVER_HSR_UPDATE_2ND
};
```

It means that after the first half of sine wave samples is transferred into eTPU data RAM, the *FS_ETPU_RESOLVER_HSR_UPDATE_1ST* constant is written to the HSR register. This writing operation initiates eTPU service requests for processing the first half period of the sampled sine and cosine wave. Similar applies for the second half period with *FS_ETPU_RESOLVER_HSR_UPDATE_2ND* that evokes processing of the second half period. Detailed configuration for DMA channels used by the eTPU Resolver function can be found in Table 4.

Table 4. DMA configuration for eTPU Resolver

Items	Sin signal	Cosin signal	Linked HSR
Source address	pThreadDescriptor[0]->buffer OutputStart + XMEM_0_ BASEADDRESS	pThreadDescriptor[1]->buffer OutputStart + XMEM_0_ BASEADDRESS	AndsResolverExcCmdDspBuf[0]
Destination address	resolver_instance. .signals_pba	resolver_instance.signals_pba + 0x80	&pEtpu_Ip_Regs->CHAN[RESOLVER_ INSTANCE.u8ChanNumAto]. HSRR.R
Source transfer size / modulo	16-bits / 2 bytes	16-bits / 2 bytes	32-bits / 4 bytes
Destination transfer size / modulo	32-bits / 4 bytes	32-bits / 4 bytes	32-bits / 4 bytes
Source address offset	2 bytes	2 bytes	4 bytes
Destination address offset	4 bytes	4 bytes	0 bytes
Minor loop byte count	64 bytes	64 bytes	4 bytes
Major loop count	2	2	2
Source last address adjustment	-64 bytes	-64 bytes	-16 bytes
Destination last address adjustment	-128 bytes	-128 bytes	0 bytes
Channel linking	Enabled	Disabled	Disabled
Linked channel	HSR DMA channel	-	-

Another DMA channel is used for 3-phase current and DC bus voltage measurements. In a single motor or the first motor of dual-motor applications, since the BCTU is triggered synchronously for its associated ADC and one of the ADCs must use its 2 channels, two rounds of sampling are required. The BCTU instance_0 FIFO_1 watermark value has been set to 5. It means that when the number of active FIFO entries exceeds 5, a BCTU instance_0 DMA request is raised. The DMA then transfers the ADC sample results from the BCTU FIFO to user-defined memory. In the second motor of dual-motor application and single motor in S32K396-LQFP-DC, three-phase current and DC bus voltage measurements are each assigned to a separate ADC instance. Table 5 lists the detailed configuration for DMA channels used by currents and voltage measurements.

Table 5. DMA configuration for phase currents and bus voltage

Applications	Items	Phase_A I Phase_B I Phase_C I DC bus V
The single motor or the first motor of a dual-motor	**Source address**	&IP_BCTU->FIFO1DR
	Destination address	Phase_A I: AndBctuDmaFifo1[0] Phase_B I: AndBctuDmaFifo1[1] Phase_C I: AndBctuDmaFifo1[2] DC bus V: AndBctuDmaFifo1[5]
The second motor of a dual-motor	**Source address**	&IP_BCTU_1->FIFO1DR
	Destination address	Phase_A I: AndBctuDmaFifo1[0] Phase_B I: AndBctuDmaFifo1[2] Phase_C I: AndBctuDmaFifo1[3] DC bus V: AndBctuDmaFifo1[1]
Single motor in S32K396-LQFP-DC	**Source address**	&IP_BCTU_1->FIFO1DR
	Destination address	Phase_A I: AndBctuDmaFifo1[1] Phase_B I: AndBctuDmaFifo1[2] Phase_C I: AndBctuDmaFifo1[3] DC bus V: AndBctuDmaFifo1[0]
All applications	**Source transfer size / modulo**	16-bits / 0 bytes
	Destination transfer size / modulo	16-bits / 0 bytes
	Source address offset	0 bytes
	Destination address offset	0 bytes
	Minor loop size	12
	Major loop count	1
	Source last address adjustment	0 bytes
	Destination last address adjustment	-12 bytes
	Channel linking	Disabled
	Linked channel	-

4.3.10 SGEN configuration

S32K396 has 2-sine wave generator (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–50 kHz in 16 Hz steps) is easily programmable through a simple register interface. The linearity and noise performance are carefully optimized through digital processing. For more details, see S32K396 Reference Manual.

The sine wave targets as the excitation signal resolver to replace the PWM signal as excitation. In this way, no more tuning circuits are needed and the cost can be decreased. Configured the output signal wave as 10 kHz with a maximum amplitude of 2 V. Also, configured the SGEN input phase align trigger mode as the hardware trigger. And the trigger signal is the AS signal from eTPU, which is routed by the TRGMUX module. The configuration parameters are as follows:

```
/* Address for SGEN Registers */
#define SGEN_0_CTRL_ADDR    0x406C8000U
```

```
#define SGEN_0_STATUS_ADDR  0x406C8004U
#define SGEN_1_CTRL_ADDR    0x406CC000U
#define SGEN_1_STATUS_ADDR  0x406CC004U
#define SGEN_0              (0U)
#define SGEN_1              (1U)
/* Masks & Offset for fields in SGEN_CTRL Registers */
#define SGEN_CTRL_LDOS_MASK         (0x1)
#define SGEN_CTRL_LDOS_SHIFT        (31U)
#define SGEN_CTRL_IOAMPL_MASK       (0xF)
#define SGEN_CTRL_IOAMPL_SHIFT      (26U)
#define SGEN_CTRL_WINDOW_MASK       (0x3)
#define SGEN_CTRL_WINDOW_SHIFT      (24U)
#define SGEN_CTRL_SEMASK_MASK       (0x1)
#define SGEN_CTRL_SEMASK_SHIFT      (23U)
#define SGEN_CTRL_TRIG_MODE_MASK    (0x1)
#define SGEN_CTRL_TRIG_MODE_SHIFT   (18U)
#define SGEN_CTRL_PDS_MASK          (0x1)
#define SGEN_CTRL_PDS_SHIFT         (16U)
#define SGEN_CTRL_IOFREQ_MASK       (0xFFFF)
#define SGEN_CTRL_IOFREQ_SHIFT      (0U)
/* User define for SGEN initialization params */
#define SGEN_WAVE_FREQ_WAIT             0x0     /* Wait for I/O sine wave
 frequency */
#define SGEN_WAVE_FREQ_LOAD             0x1     /* Load I/O sine wave frequency
 */
#define SGEN_IO_WAVE_AMPLITUDE          0xF
#define SGEN_PHA_ALIG_ACCEPT_WIN_01     0x0     /* +-1%  from the zero crossing
 */
#define SGEN_PHA_ALIG_ACCEPT_WIN_10     0x1     /* +-10% from the zero crossing
 */
#define SGEN_PHA_ALIG_ACCEPT_WIN_15     0x2     /* +-15% from the zero crossing
 */
#define SGEN_PHA_ALIG_ACCEPT_WIN_25     0x3     /* +-25% from the zero crossing
 */
#define SGEN_ERROR_INTERRUPT_OFF        0x0     /* Mask the error interrupt
 source */
#define SGEN_ERROR_INTERRUPT_ON         0x0     /* Enable the error interrupt
 source */
#define SGEN_IN_PHA_ALIG_TRIG_SW        0x0     /* Software trigger mode is
 selected */
#define SGEN_IN_PHA_ALIG_TRIG_HW        0x1     /* Hardware trigger mode is
 selected */
#define SGEN_POWER_DOWN_MODE_EXIT       0x0     /* Force SGEN to exit Power Down
 mode */
#define SGEN_POWER_DOWN_MODE_ENTER      0x1     /* Force SGEN to enter Power
 Down mode */
/* User pre-defined para for SWG output frequency */
#define SGEN_OUTPUT_FREQENCY            10486u  /* Output_Freq = (Clk * value) /
 16777216 */
void SGEN_Init(uint8_t instance)
{
    uint32_t SGEN_Addr;
    SGEN_Addr = (instance == 0) ? SGEN_0_CTRL_ADDR : SGEN_1_CTRL_ADDR;
    /* Exit power down mode */
    REG_BIT_CLEAR32(SGEN_Addr, SGEN_CTRL_PDS_MASK << SGEN_CTRL_PDS_SHIFT);

    /* Wait for load output freq */
    REG_BIT_SET32(SGEN_Addr, SGEN_WAVE_FREQ_WAIT << SGEN_CTRL_LDOS_SHIFT);
    /* Amplitude CFG */
    REG_BIT_SET32(SGEN_Addr, SGEN_IO_WAVE_AMPLITUDE << SGEN_CTRL_IOAMPL_SHIFT);
```

```
    /* Phase aligenment acceptance window width */
    REG_BIT_SET32(SGEN_Addr, SGEN_PHA_ALIG_ACCEPT_WIN_01 <<
 SGEN_CTRL_WINDOW_SHIFT);
    /* Error Interrupt Off */
    REG_BIT_SET32(SGEN_Addr, SGEN_ERROR_INTERRUPT_OFF <<
 SGEN_CTRL_SEMASK_SHIFT);
    /* Trigger mode selection */
    REG_BIT_SET32(SGEN_Addr, SGEN_IN_PHA_ALIG_TRIG_HW <<
 SGEN_CTRL_TRIG_MODE_SHIFT);
    /* Output wave frequency configuration */
    REG_BIT_SET32(SGEN_Addr, SGEN_OUTPUT_FREQENCY << SGEN_CTRL_IOFREQ_SHIFT);
    /* loaded output freq */
    REG_BIT_SET32(SGEN_Addr, SGEN_WAVE_FREQ_LOAD << SGEN_CTRL_LDOS_SHIFT);
}
```

4.3.11 Port Control and pin multiplexing

The following table shows the pins assignment for the S32K396 motor control application.

Table 6. Pins assignment for S32K396 PMSM FOC control

Module	Application	Signal name	Mode	Pin	Description
PWM	Single motor, the first motor of dual-motor, single motor in S32K396-LQFP-DC	PWMA_HS	FLEXPWM_0_A_0_OUT	GPIO120	PWM phase A HIGH-side output
		PWMA_LS	FLEXPWM_0_B_0_OUT	GPIO2	PWM phase A low-side output
		PWMB_HS	FLEXPWM_0_A_1_OUT	GPIO3	PWM phase B HIGH-side output
		PWMB_LS	FLEXPWM_0_B_1_OUT	GPIO119	PWM phases B low-side output
		PWMC_HS	FLEXPWM_0_A_2_OUT	GPIO98	PWM phase C HIGH-side output
		PWMC_LS	FLEXPWM_0_B_2_OUT	GPIO99	PWM phases C low-side output
		Fault_OC	FLEXPWM_0_FAULT_0_IN	GPIO47	Overcurrent signal for fast shut-off
		Fault_OV	FLEXPWM_0_FAULT_1_IN	GPIO48	Overvoltage signal for fast shut-off
	Single motor, The first motor of a dual-moto	MC33937_ Inerrupt	SIUL_EIRQ_12	GPIO196	external interrupt from MC33937
	The second motor of a dual-motor	PWMA_HS	FLEXPWM_1_A_0_OUT	GPIO95	PWM phase A HIGH-side output
		PWMA_LS	FLEXPWM_1_B_0_OUT	GPIO6	PWM phase A low-side output
		PWMB_HS	FLEXPWM_1_A_1_OUT	GPIO94	PWM phase B HIGH-side output
		PWMB_LS	FLEXPWM_1_B_1_OUT	GPIO7	PWM phases B low-side output
		PWMC_HS	FLEXPWM_1_A_2_OUT	GPIO93	PWM phase C HIGH-side output

Table 6. Pins assignment for S32K396 PMSM FOC control*...continued*

Module	Application	Signal name	Mode	Pin	Description
		PWMC_LS	FLEXPWM_1_B_2_OUT	GPIO72	PWM phases C low-side output
		Fault_OC	FLEXPWM_1_FAULT_0_IN	GPIO90	Overcurrent signal for fast shut-off
		Fault_OV	FLEXPWM_1_FAULT_1_IN	GPIO75	Overvoltage signal for fast shut-off
		MC33937_ Inerrupt	SIUL_EIRQ_20	GPIO76	external interrupt from MC33937
	Single motor in S32K396-LQFP-DC	MC33937_ Inerrupt	SIUL_EIRQ_4	GPIO132	external interrupt from MC33937
eTPU	Single motor, The first motor of dual-motor, Single motor in S32K396-LQFP-DC	EXC_SIG	ETPU_A_CH_0_OUT	GPIO44	eTPU excitation signal output
	The second motor of a dual-motor	EXC_SIG	ETPU_B_CH_7_OUT	GPIO32	eTPU excitation signal output
LPUART	All applications	UART_RX	LPUART0_RX	GPIO138	UART receives data
		UART_TX	LPUART0_TX	GPIO139	UART transmits data
SDADC	Single motor, The first motor of dual-motor, Single motor in S32K396-LQFP-DC	POS_SIN	SDADC0_AN_0	GPIO130	Resolver sine feedback signal
		POS_COS	SDADC1_AN_0	GPIO16	Resolver cosine feedback signal
	The second motor of a dual-motor	POS_SIN	SDADC2_AN_0	GPIO41	Resolver sine feedback signal
		POS_COS	SDADC3_AN_0	GPIO124	Resolver cosine feedback signal
SARADC	Single motor, The first motor of a dual-motor	PHA_I	ADC0_P0	GPIO97	Phase A current
		PHB_I	ADC1_P0	GPIO154	Phase B current
		PHC_I	ADC2_P1	GPIO24	Phase C current
		DCB_V	ADC2_P5	GPIO8	DC bus voltage
	The second motor of a dual-motor	PHA_I	ADC3_P4	GPIO15	Phase A current
		PHB_I	ADC5_P2	GPIO123	Phase B current
		PHC_I	ADC6_P1	GPIO40	Phase C current
		DCB_V	ADC4_S11	GPIO33	DC bus voltage
	Single motor in S32K396-LQFP-DC	PHA_I	ADC4_P1	GPIO15	Phase A current
		PHB_I	ADC5_P1	GPIO124	Phase B current
		PHC_I	ADC6_P1	GPIO40	Phase C current

Table 6. Pins assignment for S32K396 PMSM FOC control*...continued*

Module	Application	Signal name	Mode	Pin	Description
		DCB_V	ADC3_P1	GPIO8	DC bus voltage
LPSPI	Single motor, The first motor of dual-motor, S32K396-LQFP-DC	LPSPI1_SCOUT	LPSPI1_SCOUT_OUT	GPIO18	LPSPI data output
		LPSPI1_SCK	LPSPI1_SCK_OUT	GPIO19	LSPI clock
		LPSPI1_SIN	LPSPI1_SIN_IN	GPIO20	LSPI data input
		LPSPI1_PCS0	LPSPI1_PCS0_OUT	GPIO21	LSPI CS signal
	The second motor of a dual-motor	LPSPI3_SCOUT	LPSPI3_SCOUT_OUT	GPIO175	LPSPI data output
		LPSPI3_SCK	LPSPI3_SCK_OUT	GPIO173	LSPI clock
		LPSPI3_SIN	LPSPI3_SIN_IN	GPIO172	LSPI data input
		LPSPI3_PCS0	LPSPI3_PCS0_OUT	GPIO176	LSPI CS signal
GPIO	Single motor, The first motor of a dual-motor	MC_PreDrv_EN	GPIO	GPIO14	Enable signal for MC33937
		MC_PreDrv_RESET	GPIO	GPIO23	Reset signal for MC33937
	The second motor of a dual-motor	MC_PreDrv_EN	GPIO	GPIO206	Enable signal for MC33937
		MC_PreDrv_RESET	GPIO	GPIO96	Reset signal for MC33937
	Single motor in S32K396-LQFP-DC	MC_PreDrv_EN	GPIO	GPIO96	Enable signal for MC33937
		MC_PreDrv_RESET	GPIO	GPIO133	Reset signal for MC33937

4.4 Software architecture

The motor control application in S32K396 is based on the project on Cobra55. The software architecture is mostly the same as the architecture on Cobra55. See the AN13038: MCSPTR2A5775E 3-phase PMSM Motor Control Kit with MPC5775E and/or AN12017: 3-Phase PMSM Development Kit with MPC5744P.

5 Conclusion

The design described in this application note shows the simplicity and efficiency solution for PMSM motor control based on the S32K396 chip.

6 References

1. AN12017: 3-Phase PMSM Development Kit with MPC5744P.
2. AN13038: MCSPTR2A5775E 3-phase PMSM Motor Control Kit with MPC5775E.
3. S32K396 Reference Manual.
4. MC33937: 3-Phase Field Effect Transistor Pre-driver.
5. Automotive Math and Motor Control Library Set for NXP S32K3xx devices.
6. FreeMASTER Run-Time Debugging Tool.

7 Note about the source code in the document

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

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

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

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

8 Revision history

Revision history

Document ID	Release date	Description
AN14326 v.2.0	22 October 2024	Upgrade RTD and eTPU SW
AN14326 v.1.0	21 May 2024	Initial release

Legal information

Definitions

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

Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Trademarks

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

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

CoolFlux DSP — is a trademark of NXP B.V.

Tables

Figures

Contents

Date of release: 22 October 2024
Document identifier: AN14326