7 Hardware Migration Guide

This section helps users to migrate the SDK example project to their own customer board.

7.1 Hardware Layer Overview

Figure 7-1 shows the hardware configuration relationships for FOC project. The hardware layer contains three parts: Board layer, SysConfig layer and HAL layer.

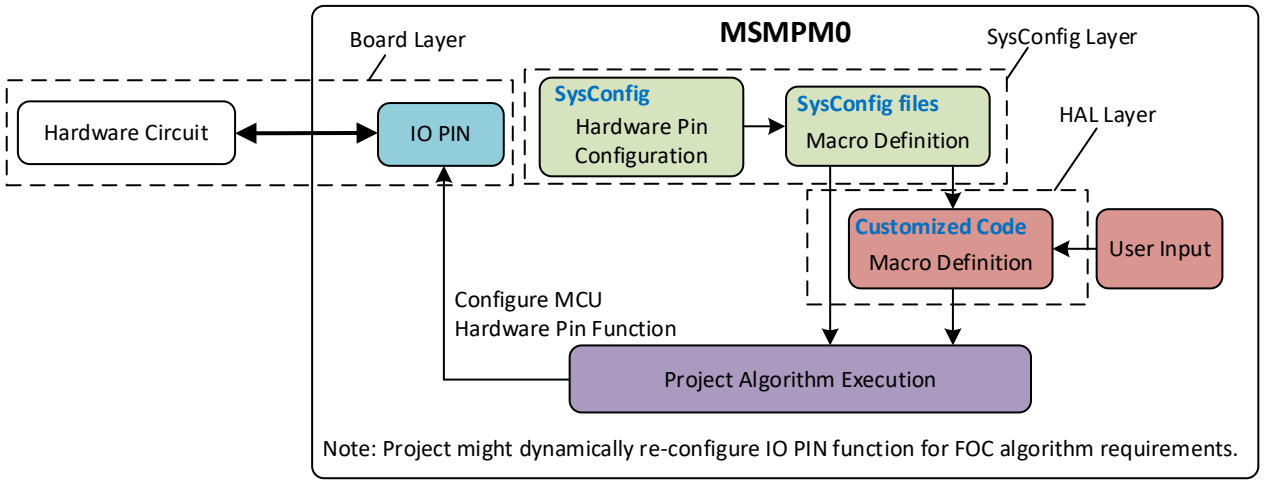


Figure 7-1. Hardware Configuration Relationship

The Board layer is the layer that users' connection for the MCU with other components in hardware circuit. For FOC applications, it is mainly for the connection to the used gate driver device and ADC sampling channels.

The SysConfig layer sets the MCU peripheral initialization features in the ***.syscfg*** file. The IO PIN connected to the external circuit needs to be set as the appropriate peripheral function to run FOC application sucessfully. The SDK project provides several peripheral default configurations to fit the DRV EVM Board or TIDA Reference Board. Users need to configure the ***.syscfg*** file to manually adapt the hardware circuit. The SysConfig tool will automatically generate MCU peripheral initialization file, including macro definitions **mapped to MCU hardware peripherals**.

The second HAL layer is the **Customized Macro Definition** in the SDK FOC project. Lots of FOC functions call customized macro definitions to determine the implementation of the algorithm. Therefore, users need manually to manage these customized macro definitions in header files to adapt the hardware circuit and SysConfig generated macro definition.

Follow the steps below to migrate the hardware configurations for customized board:

1. Check the MSPM0 IO pin used for each module on the customized board.
2. Modify SysConfig configurations to fit the hardware design.
3. Modify HAL layer macro definitions to fit the hardware design.

7.2 MCU Peripheral Configuration

7.2.1 Gate Driver Module

FOC application uses a pre-defined symbol to determine which gate driver board is used to configure the board parameters and HAL layer properly, as shown in Figure 7-2.

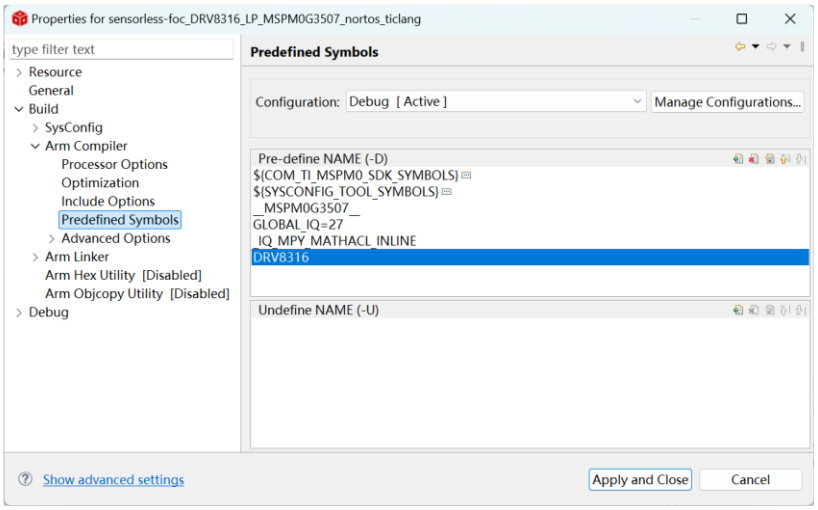


Figure 7-2 DRV8316 Pre-define Symbol in Project Properties

The default gate driver interface is configured for the EVM hardware board. Users can overwrite the relevant configuration set for DRV8316 to their own board configuration. While recommend using **CUSTOM** symbol for users' specific hardware design if use another gate driver for better code management.

Follow the steps below to configure the Custom Gate Driver Module:

7.2.1.1 Select Reference Projects

MSPM0 SDK provides various FOC example projects for different hardware boards. Refer to Table 7-1 to select the suitable project to do the migration for your own hardware board.

Table 7-1 Project Recommendation for Migration

FOC Type	Customized Hard Board		Project Recommendation	Migration Effort
	Gate Driver	Current Sensing Type		
Sensorless / Universal FOC	DRV8323	Single Shunt	sensorless-foc_DRV8323RS	Porting single shunt configuration.
			sensorless-foc_DRV8329	Porting gate driver configuration.
		Dual or Three Shunt	sensorless-foc_DRV8323RS	No significant effort.
	DRV8316	Single Shunt	sensorless-foc_DRV8316	Porting single shunt configuration.
			sensorless-foc_DRV8329	Porting gate driver configuration.
		Dual or Three Shunt	sensorless-foc_DRV8316	No significant effort.
	Others	Single Shunt	sensorless-foc_DRV8329	No significant effort.
		Dual or Three Shunt	sensorless-foc_TIDA010250	No significant effort.
Sensored FOC	DRV8316	Single Shunt	hall_sensored-foc_DRV8316	Porting single shunt configuration.
			hall_sensored-foc_TIDA010251	Porting gate driver configuration.
		Dual or Three Shunt	hall_sensored-foc_DRV8316	No significant effort.
	Others	Single Shunt	hall_sensored-foc_TIDA010251	No significant effort.
		Dual or Three Shunt	hall_sensored-foc_DRV8316	No significant effort.

Below uses the TIDA010250 as an example to introduce the flow for gate driver module migration.

7.2.1.2 Modify Pre-defined Symbols

In the project properties, modify the Default Pre-defined Symbols: TIDA010250->CUSTOM.

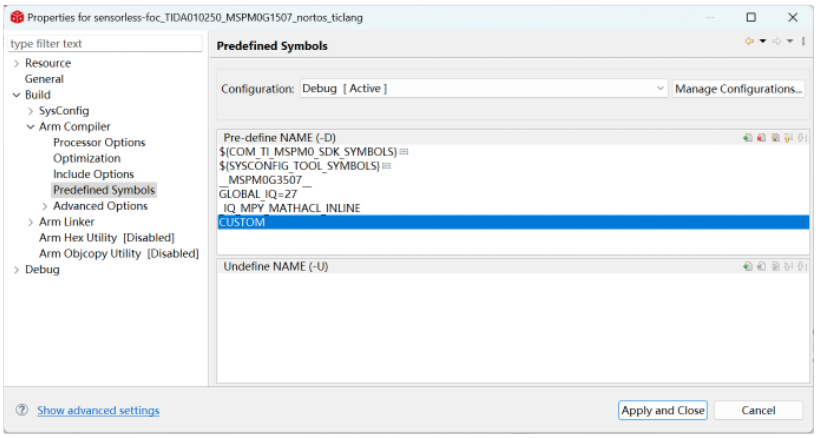


Figure 7-3 Modify Pre-define Symbol in Project Properties

7.2.1.3 Add Custom Source Files

By default, FOC project has integrated SPI communication for gate drivers in FOC library, users are owned to maintain the source code for gate driver communication if use customized gate drivers. To help users quickly remove the default SPI communication, FOC projects provide custom source files in SDK folder. Users could manually add source files into the customized project with **CUSTOM** pre-defined symbol.

7.2.1.3.1 Gate Driver Comm Folder

The preset custom source files for Gate Driver Communication are in the following SDK path:

```
...\ti\mspm0_sdk_<SDK_Version>\source\ti\motor_control_pmsm_foc\common_modules\hal\LP_MSPM0Gx5xx\gateDriverInterface\gateDriverLib
```

Figure 7-4 shows the CUSTOM Gate Driver Library folder. The library deletes default codes for SPI communications for gate drivers in FOC application, so that users can add their own gate driver communication codes (**Section 7.2.1.5**) in their customized files.

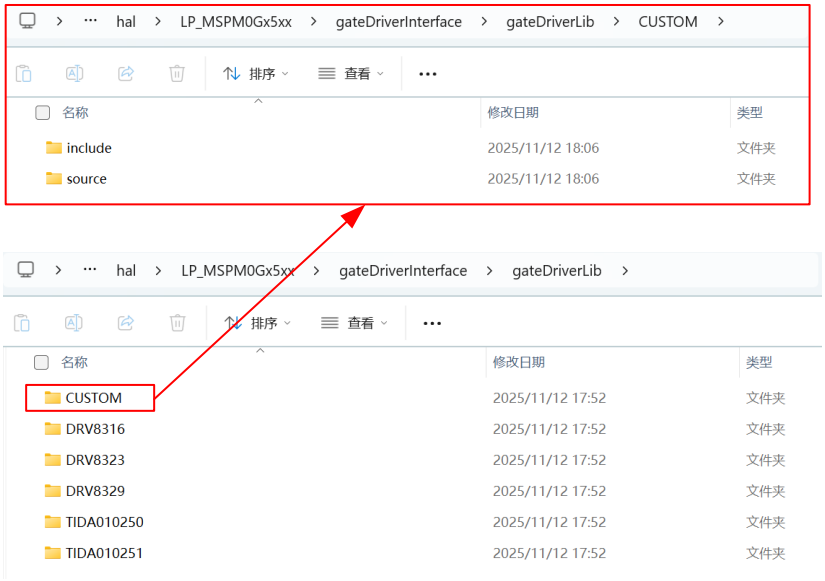


Figure 7-4 CUSTOM Gate Driver Library

Add the CUSTOM folder (copy and paste) into the example FOC project as shown in Figure 7-5. The default Gate Driver folder (TIDA010250) can be deleted (optional).

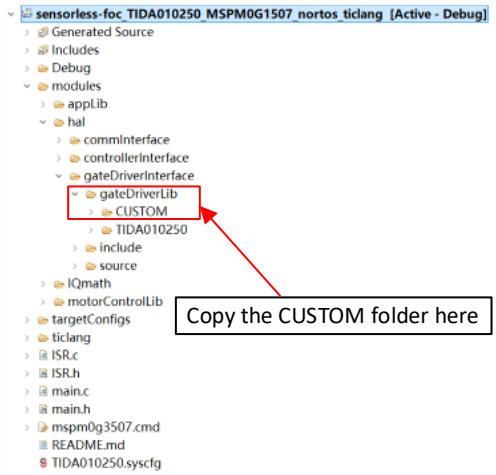


Figure 7-5 Add CUSTOM Folder into FOC Project

Add the CUSTOM folder path into the Include Options in Project Properties below:

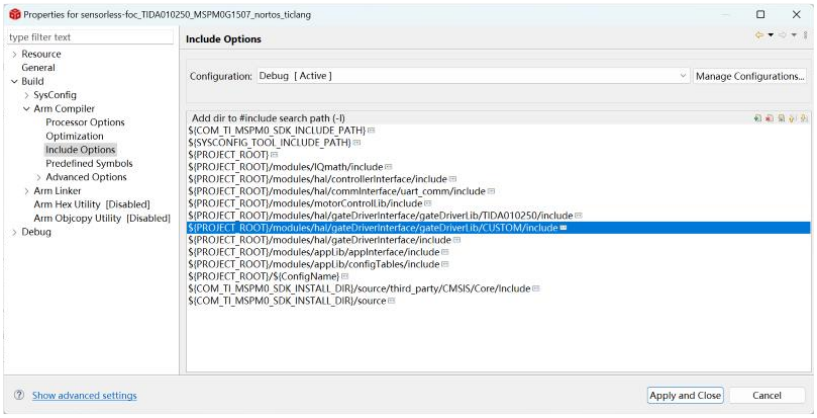


Figure 7-6 Add CUSTOM Folder into Include Options

7.2.1.3.2 HAL Layer File

The preset custom source files for HAL layer are in the following SDK path:

`...\ti\mspm0_sdk_<SDK_Version>\source\ti\motor_control_pmsm_foc\common_modules\hal\LP_MSPM0Gx5xx\gateDriverInterface\[LAUNCHPAD]\source`

Figure 7-7 shows the CUSTOM HAL layer file.

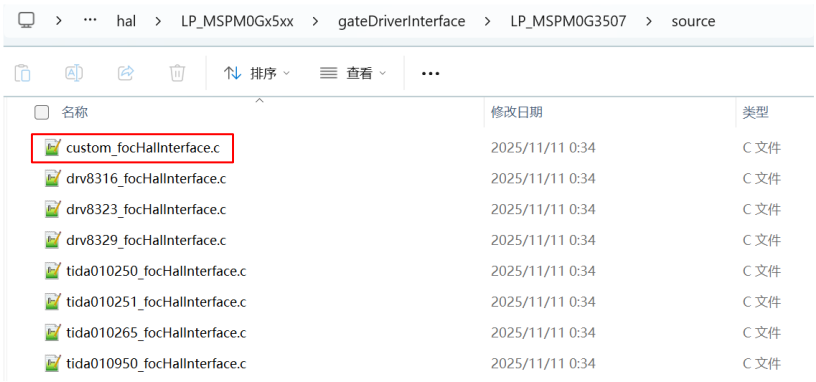


Figure 7-7 CUSTOM HAL Layer

Add the ***custom_focHallInterface.c*** file into the ***gateDriverLib*** folder of the FOC project below:

Then copy it to the path corresponding to the SDK project:

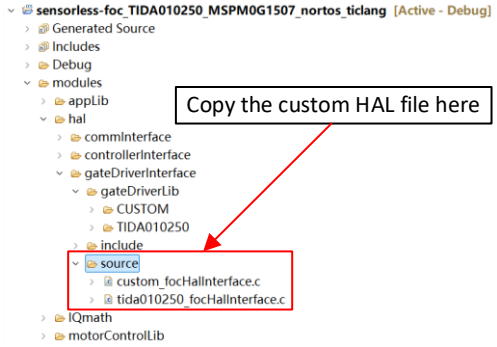


Figure 7-9 Add the Custom HAL file into FOC project

7.2.1.4 Add Custom Comm Interface

Gate drivers support different Comm interfaces to configurate, including resistor based, SPI based, IIC based, and others. It is complex for users to integrate a new Comm interface into FOC HAL library and debug. An easier way is introduced to disable default SPI based communication (DRV8316 or DRV8323 FOC project), so that the FOC algorithm does not use any MCU pins or peripherals to do communication.

Follow the steps in Section **7.2.1.3.1** to use CUSTOM communication interface. Then the gate driver communication APIs is set to empty and will not impact normal FOC applications, as shown in Figure 7-10. Users can also manually modify the source code as empty for the gate driver communication interface.

custom.c

```
#include <math.h>
#include "custom.h"

#include "main.h"
#include "ISR.h"
#include "focHALInterface.h"

#ifdef CUSTOM
uint32_t gateDrivernFaultReport = 0;
uint32_t gateDriverFaultAction = 0;

void gateDriverConfig(void)
{


}


void gateDriverClearFault(void)
{

}


uint32_t gateDriverGetFaultStatus(void)
{
    return 0;
}
/*Update the Gate Driver Parameters when the Config Enable Flg is Set */
void gateDriverParamsUpdate(HV_DIE_EEPROM_INTERFACE_T *pGateDriverConfig)
{

}

```

Figure 7-10 Custom Gate Driver File

Users can then add their own gate driver files out of FOC application to suit their own hardware board.

Note

Directly deleting APIs shown in Figure 7-10 results in compiler error as Sensorless FOC library calls these APIs in a static library.

7.2.1.5 Overwrite Default Macro Definitions

7.2.1.5.1 main.h File

The ***main.h*** misses the CUSTOM macro for user customization. Add the related code below (copy from other macro and overwrite with CUSTOM):

Manually add CUSTOM parts

```
217
218 #elif defined CUSTOM
219
220 /*! @brief TIDA010250 has inverting isense */
221 #define _INVERT_ISEN
222 /*! @brief IPD feature is enabled */
223 #define __IPD_ENABLE
224 /*! @brief TIDA010250 propagation delay */
225 #define DRIVER_PROPAGATION_DELAY_nS                500
226 /*! @brief TIDA010250 minimum on time (rise time + settling time) */
227 #define DRIVER_MIN_ON_TIME_nS                      8000
228 /*! @brief DC voltage base value */
229 #define DC_VOLTAGE_BASE                             448.0
230 /*! @brief Full scale readable current used as current base value,
231 calculated using (FULL Scale Voltage(3.3)/2* CSA Gain) */
232 #define FULL_SCALE_CURRENT_BASE                     8.25
233 /*! @brief Current shunt configuration */
234 #define __CURRENT_THREE_SHUNT_DYNAMIC
235 /*! @brief Enable dynamic current shunt changing */
236 #define DYNAMIC_CURRENT_SHUNT_CONFIG_EN             FALSE//TRUE
237
238 #endif
```

Figure 7-11 Add the Custom Macro into main.h File

Users should take care of the macro definitions modification based on the hardware board features, refers to Table 7-2 below.

Table 7-2 User Status Registers

Hardware Feature	Macro	Description
Current Sensing Type	_INVERT_ISEN	The board has an inverting or non-inverting current sensing hardware circuit. See **Section 7.2.3.1**.
	_NONINVERT_ISEN	
Current Sensing Method	__CURRENT_XX_SHUNT	Current sensing method includes Single Shunt, Dual Shunt and Three Shunt. See **Section 7.2.3.2**.
Board Parameters	DC_VOLTAGE_BASE	Maximum measurable bus voltage. See **Section 6.1.1**.
	FULL_SCALE_CURRENT_BASE	Base current. See **Section 6.1.2**.
	DRIVER_PROPAGATION_DELAY_nS	Defines the Time delay in ns between the Input PWM logic edge fed to the gate driver and actual Gate Driver output.
	DRIVER_MIN_ON_TIME_nS	Defines the combined rise time and settling time of the current sense amplified output.
IPD Function	__IPD_ENABLE	Enable IPD function for FOC application. The IPD configuration register should be set properly, See **Section 6.6.1.1.3**.

7.2.1.5.1.1 Delay Component in Current Sensing Path

Figure 7-12 shows the delay components in the current measurement path.

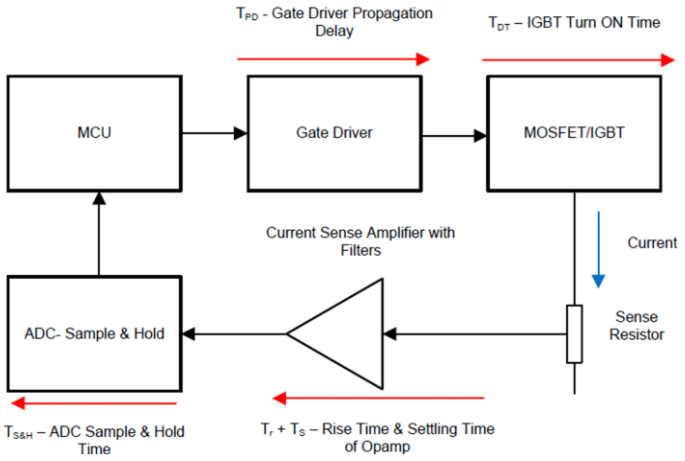


Figure 7-12 Delay in the Current Sensing Loop

FOC application uses DRIVER_PROPAGATION_DELAY to define the time delay in ns between the Input PWM logic edge fed to the gate driver and actual gate driver Output PWM. This delay impacts the Current Sense sampling instance on the actual gate driver output and has to be fed to the algorithm for accurate Current Sensing. Users can get the propagation delay from the used gate driver's data sheet or measure the time with hardware.

FOC application uses DRIVER_MIN_ON_TIME to define the combined rise time and settling time of the current sense amplified output. This value has to be captured independently for a full-scale change in voltage across the current shunt. For accurate current sense reading, the current sense amplifier output to be settled before the current signal is captured. Refer to **Equation 16** to calculate the DRIVER_MIN_ON_TIME.

DRIVER_MIN_ON_TIME = CSA Settling Time + CAS Rise Time (16)

Note

DRIVER_MIN_ON_TIME limits the maximum FOC output PWM duty cycle to reserve sufficient time for CSA to establish the stable current signal to feed ADC sampling channel.

7.2.1.5.2 gateDriver.h File

The ***gateDriver.h*** file defines the HAL layer macros. It is essential for users to overwrite the macros according to their hardware board circuit.

For PWM related macro definition overwriting, refer to **Section 7.2.2**.

For ADC related macro definition overwriting, refer to **Section 7.2.3**.

7.2.2 PWM Module

Using TIMA0 of MSPM0 to enable three pairs of complementary PWM output and deadband time insertion for FOC application. There have four output channels of TIMA0, and users can select any three of them for FOC application.

When the hardware circuit is finalized, users should pay attention to the correspondence between the motor phase sequence and PWM channel setting. The default mapping relationship for **sensorless-foc_DRV8316_LP_MSPM0G3507** project is shown in Table 7-3.

Table 7-3 PWM Mapping

Board Layer		HAL Layer		Descriptions
	SysConfig Layer			
Board Macro	IO PIN	PWM Channel	HAL Layer Macro	
INHA	PB4	TIMA0_C2	FOC_PWMA0_U_IDX	Phase A PWM output.
INHB	PA28	TIMA0_C3	FOC_PWMA0_V_IDX	Phase B PWM output.
INHC	PB20	TIMA0_C1	FOC_PWMA0_W_IDX	Phase C PWM output.

FAULT	PA27	Fault Pin 2	NOT USED	Gate driver output fault pin.
NA	NA	TIMA0_C0	FOC_PWMA0_ADC_TRIG_IDX	Trigger channel from PWM to ADC.

The HAL Layer Macro always keeps the same mapping relationship to the Board Macro, and the SysConfig layer should be modified accordingly.

Note

In current FOC applications, the unused PWM channel is configured as ADC trigger channel, but the PIN output function is enabled in the project. Users can disable it manually in the main loop code, or use the non-output channel (CC4 or CC5), to get rid of this PWM PIN output impact

7.2.2.1 Different Pin Used for PWM Output

For the case that users only change different PWM output pins in hardware while the mapping relationship keeps the same, users can just modify the PINMUX selection of the TIMA0 output channel in SysConfig tools, as shown in Figure 7-13.

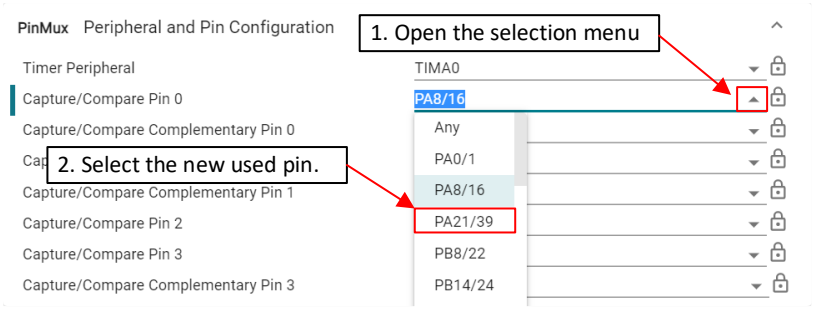


Figure 7-13 Modify PWM Output Pin in SysConfig

7.2.2.2 Different Pin for PWM Fault Pin

For the case that users change different PWM fault pins in hardware, users can just modify the PINMUX selection of the TIMA0 fault pin in SysConfig tools, as shown in Figure 7-14.

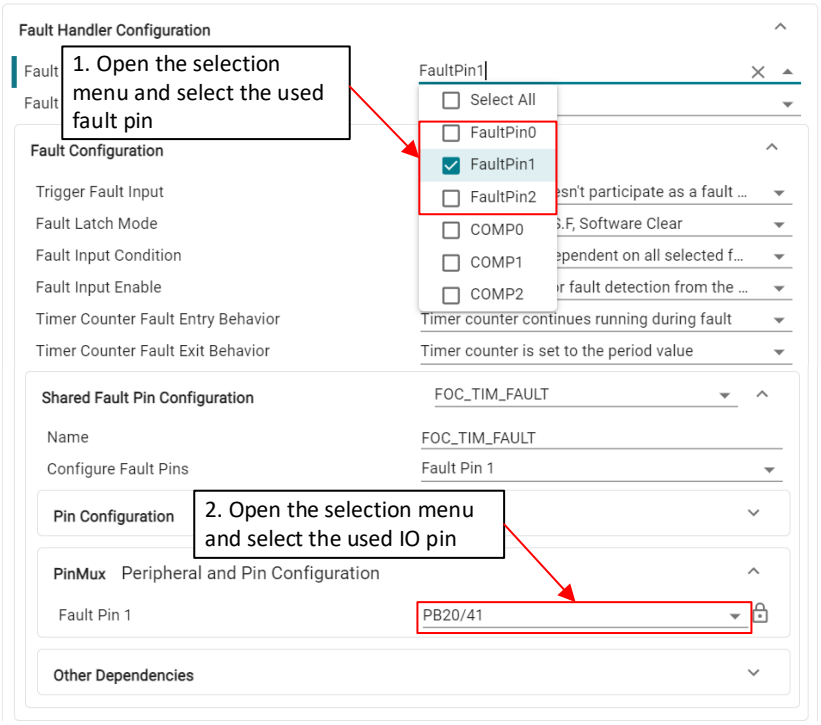


Figure 7-14 Modify PWM Fault Pin in SysConfig

7.2.2.3 Different Mapping to PWM Output Channel

For the case that the PWM output channel mapping in hardware is different, users should first refer to **Section 7.2.2.1** to modify the IO PIN used in SysConfig. Then position to ***gateDriver.h*** file to overwrite the HAL layer macro.

For example, if users have a new mapping table below:

Table 7-4 Custom PWM Mapping

Board Macro	IO PIN	PWM Channel	HAL Layer Macro
INHA	TBD	TIMA0_C0	FOC_PWMA0_U_IDX
INHB	TBD	TIMA0_C1	FOC_PWMA0_V_IDX
INHC	TBD	TIMA0_C2	FOC_PWMA0_W_IDX
NA	NA	TIMA0_C3	FOC_PWMA0_ADC_TRIG_IDX

The corresponding modifications in ***gateDriver.h*** file are shown in Figure 7-15:

Figure 7-15 Modify HAL Layer Macro

The remaining unused PWM output channel is always used to trigger ADC sampling in dual or three shunt current sensing method. And the ADC trigger EVENT should be set as the event of the trigger PWM channel.

For single shunt current sensing method, FOC application uses the secondary TIMA1 to trigger the ADC, so that no trigger channel modification is required.

7.2.3 ADC Module

When migrating ADC module to customized setting, users should pay attention to the mapping relationship as well as the current sensing method used in hardware circuit.

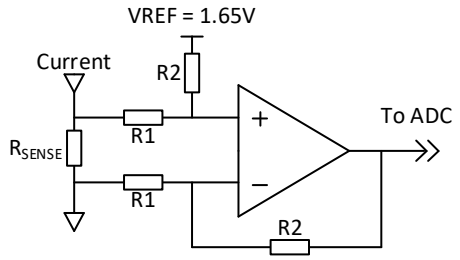
7.2.3.1 Current Sensing Type

Users should overwrite the macro definitions in the project according to the hardware sensing circuit type in ***main.h*** file, as shown in Table 7-5.

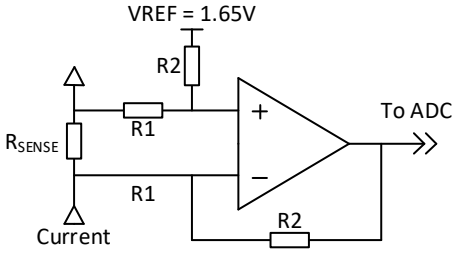
Table 7-5 Current Sensing Type

File	Macro Definition	Description
main.h	_NONINVERT_ISEN	Hardware board uses non-inverting current sensing circuit
	_INVERT_ISEN	Hardware board uses inverting current sensing circuit

Figure 7-16 shows the typical bidirectional current sensing circuit block diagram for users' reference.



a. Non-inverting Current Sensing Circuit



b. Inverting Current Sensing Circuit

Figure 7-16. Current Sensing Type

7.2.3.2 Current Sensing Method

The SDK FOC example can be configured for various shunt configuration options such as Single Shunt, Dual Shunt and Three Shunt. Based on the HW design the appropriate shunt configuration has to be selected for proper operation of algorithm. FOC application supports simultaneously sampling the two phases at a given instance to optimize the current sampling time. By default, in all shunt configurations, both the ADC instances are used to utilize the simultaneous sampling feature.

7.2.3.2.1 Three Shunt Configuration

Table 7-6 shows the macro for three shunt configurations.

Table 7-6 Macro for Three Shunt Sensing

File	Macro Definition	Description
main.h	__CURRENT_THREE_SHUNT_AB_C	Select this configuration if A and B phases are sensed through ADC0 and C phase is sensed through ADC1.
	__CURRENT_THREE_SHUNT_A_BC	Select this configuration if A phase is sensed through ADC0 and B, C phases are sensed through ADC1.

7.2.3.2.2 Three Shunt Configuration with Simultaneously Sampling

Users can also route one of the Phases say 'B' to both the ADC 0 and 1 instance and the other two phases to two different ADC instances.

For example, Phase A is routed to ADC0, and Phase C is routed to ADC1. Phase B is routed to both ADC0 and ADC1 instances. Then algorithm can dynamically switch to the two samples which gives the best current sampling time based on the given sector if enabled simultaneously sampling.

Under simultaneously sampling mode, FOC application supports shifting the current sensing estimation dynamically to the two phases for maximizing the modulation index. As in a balanced three phase Motor, any one of the three-phase currents can be estimated with the other two-phase currents in **Equation 17**.

$$Ia + Ib + Ic = 0 \quad (17)$$

Based on the operational sector the two phases with lowest modulation index are selected for current measurement and third phase with highest modulation index is estimated using the other two-phase currents. This method helps in extending the modulation index to higher limits with continuous SVM operation.

Table 7-7 shows the example to enable simultaneously sampling.

Table 7-7 Macro for Simultaneously Sampling

File	Macro Definition	Description
main.h	__CURRENT_THREE_SHUNT_DYNAMIC	Add macro to the file.
	DYNAMIC_CURRENT_SHUNT_CONFIG_EN	Add macro to the file and set it to TRUE.

7.2.3.2.3 Dual Shunt Configuration

Table 7-8 shows the macro for dual shunt configurations.

Table 7-8 Macro for Three Shunt Sensing

File	Macro Definition	Description
main.h	__CURRENT_TWO_SHUNT_A_B	Select this configuration if only two shunt sense across phase A and B are available current sampling where A is channeled to ADC 0 and B to ADC1.
	__CURRENT_TWO_SHUNT_B_C	Select this configuration if only two shunt sense across phase B and C are available current sampling where B is channeled to ADC 0 and C to ADC1.
	__CURRENT_TWO_SHUNT_A_C	Select this configuration if only two shunt sense across phase A and B are available current sampling and A is channeled to ADC 0 and C to ADC1.

If user has a different combination of phases routed to the ADC 0 and ADC1 instances than the default connections in SDK. Appropriate changes are required to be made. See **Section 7.2.3.4**.

7.2.3.2.4 Single Shunt Configuration

Using the macro __CURRENT_SINGLE_SHUNT for dual shunt configurations.

Single shunt configuration requires users to add additional TIMA1 instance, enable cross trigger function with TIMA0. Recommend porting configurations from SDK example project single shunt configuration as shown in Table 7-1 and following the instructions in **Section 7.2.3.4.1.3**.

7.2.3.3 CSA Offset Scaling Factor

FOC application converts the sampled currents through ADC into PU system based on the maximum current that can be sensed through the ADC. This depends on the CSA offset introduced from the amplifier.

Typically for bipolar current sense measurement, full scale value of ADC 3.3V / 2 = 1.65V is given as offset. For applications where the current sensing is always unipolar, offset values are set at less than 0.5V to use the maximum full scale ADC output for positive current measurement and small margin is left for negative current measurement.

FOC application requires this scaling to be specified for appropriate functionality. As the ADC 12-bit value is converted to PU value, if the offset is set as 0: Then the scaling factor to be set as _IQ(1). If the CSA offset in HW is set as 1.65V (3.3V / 2) for bipolar current sense measurement the scaling factor to be set as _IQ(2). For any arbitrary offset values, the scaling values to be specified as

_IQ(CSA_OFFSET [PU]) = _IQ(3.3V/(3.3V – CSA_OFFSET [V])) (18)

FOC application provides a macro for users to change the preset CSA offset defined for single shunt current sensing method, as shown in Table 7-9.

Table 7-9 Macro for CSA Offset

File	Macro Definition	Description
Drv8329_focHalInterface.c	DRV8329_CURRENT_SF_IQ	Define the CSA offset scaling factor for single shunt current sensing method.

For dual or three shunt current sensing method, FOC application hardcode the scaling factor to _IQ(2) in ***focHALInterface.c*** file.

7.2.3.4 ADC Mapping

Table 7-10 shows the default mapping relationship for **sensorless-foc_DRV8316_LP_MSPM0G3507** project.

Table 7-10 ADC Mapping

Board Layer		HAL Layer		SysConfig Configurations
	SysConfig Layer			
Board Connection	**ADC CHAN**	**ADC MEMRES**	**HAL Layer Macro**	
Phase A Current	A0_3	A0_MEM0	ADC0_CURRENT_U_CH	Initializes Memory 0/1 of ADC0/ADC1. **The channel selection does not matter** as the FOC application dynamically overwrites the channel settings according to HAL layer macro in runtime.
Phase B Current 1	A0_2	A0_MEM1	ADC0_CURRENT_V_CH	
Phase B Current 2	A1_2	A1_MEM0	ADC1_CURRENT_V_CH	
Phase C Current	A1_1	A1_MEM0	ADC1_CURRENT_W_CH	
Bus Voltage	A1_3	A1_MEM2	FOC_ADC_VOLT_DC_INST ADC_VOLT_DC_IDX	Assign the ADC1 channel 3 to Memory 2.
Phase A Voltage	A1_6	A1_MEM0	ADC_VOLTAGE_U_INST ADC_VOLTAGE_U_IDX ADC_VOLTAGE_U_CH	Not initialized in SysConfig and dynamically initialized in the FOC application if run ISD function.
Phase B Voltage	A0_7	A0_MEM0	ADC_VOLTAGE_V_INST ADC_VOLTAGE_V_IDX ADC_VOLTAGE_V_CH	
Phase C Voltage	A1_5	A1_MEM1	ADC_VOLTAGE_W_INST ADC_VOLTAGE_W_IDX ADC_VOLTAGE_W_CH	
ADC Interrupt	NA	A0_MEM1	FOC_ADC_ISR_INST	Initialize ADC0 interrupt configuration with MEM1 Result Loaded.
			FOC_ADC_MEM_RES_LOAD	

The FOC application dynamically modifies ADC channel and ADC memory result register mapping for different modes. In ISD mode, FOC applications use ADC0 MEM0 and ADC1 MEM0/1 to store sampled phase voltages. In other mode, FOC application use ADC0 MEM0/1 and ADC1 MEM0/1 to store sampled phase currents. Table 7-11 shows the API used by FOC application in ***Drv8316_focHalInterface.c*** file.

Table 7-11 APIs to Modify ADC Mapping

FOC API	HAL Layer Macro	Description
HAL_GD_ConfigureVoltageChannels()	ADC_VOLTAGE_X_INST ADC_VOLTAGE_X_IDX ADC_VOLTAGE_X_CH	Map the phase voltage channel (X_CH) to certain ADC MEM (X_IDX) of ADC Instance (X_INST). X = U / V / W.
HAL_GD_ReadVoltages()	ADC_VOLTAGE_U_INST ADC_VOLTAGE_U_IDX	Read the phase voltage from the ADC MEM (X_IDX) of the ADC Instance (X_INST). X = U / V / W.
HAL_GD_ConfigureCurrentChannels()	ADC0_CURRENT_X_CH	Map the phase current channel (X_CH) to the **hardcoded** ADC instance and ADC Memory. X = U / V / W.
HAL_GD_ReadCurrents()	NA	Read the phase current from the hardcoded ADC Memory of ADC Instance.
HAL_GD_ReadDCVBusVoltage()	FOC_ADC_VOLT_DC_INST ADC_VOLT_DC_IDX	Read the bus voltage from the pre-defined ADC memory of ADC instance.

Below sections introduce the flow if user's hardware does not follow the default mapping of SDK project.

7.2.3.4.1 Phase Current Channels

The modifiable macros for phase current channels are the ADC channels, the remaining configurations (ADC Instance and ADC Index) are hardcoded in the ***HAL_GD_ConfigureCurrentChannels()*** API, as shown in Figure 7-17.

```
168 void HAL_GD_ConfigureCurrentChannels(CURRENT_SHUNT_TYPES currentShunt)
169 {
170     /* Configure the ADC channels based on the ADC pin configurations */
171     switch(currentShunt)
172     {
173         case CURRENT_THREE_SHUNT_DYNAMIC:
174         case CURRENT_THREE_SHUNT_AB_C:
175             HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
176             HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_1, ADC0_CURRENT_V_CH);
177             HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
178             break;
179         case CURRENT_THREE_SHUNT_A_BC:
180             HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
181             HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_V_CH);
182             HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_1, ADC1_CURRENT_W_CH);
183             break;
184
185         case CURRENT_TWO_SHUNT_A_B:
186             HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
187             HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_V_CH);
188             break;
189
190         case CURRENT_TWO_SHUNT_A_C:
191             HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
192             HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
193             break;
194
195         case CURRENT_TWO_SHUNT_B_C:
196             HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_V_CH);
197             HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
198             break;
199
200         case CURRENT_SINGLE_SHUNT:
201             break;
202         default:
203             break;
204     }
205 }
```

Figure 7-17 Source Code for HAL_GD_ConfigureCurrentChannels

Figure 7-18 shows the ADC channel macros of phase currents.

```
680 /*! @brief Instance for ADC Interrupt */
681 #define FOC_ADC_ISR_INST        (ADC0_INST)
682 /*! @brief Memory Load Register for ADC Interrupt */
683 #define FOC_ADC_MEM_RES_LOAD    DL_ADC12_IIDX_MEM1_RESULT_LOADED
684
685 /*! @brief ADC instance for DC bus voltage */
686 #define FOC_ADC_VOLT_DC_INST    (ADC0_INST)
687 /*! @brief IDX for the DC bus voltage */
688 #define ADC_VOLT_DC_IDX         (ADC0_ADCMEM_3)
689
690 /*! @brief Channel for Phase U current */
691 #define ADC0_CURRENT_U_CH       (DL_ADC12_INPUT_CHAN_3)
692 /*! @brief Channel for Phase V current */
693 #define ADC0_CURRENT_V_CH       (DL_ADC12_INPUT_CHAN_1)
694 /*! @brief Channel for Phase V current */
695 #define ADC1_CURRENT_V_CH       (DL_ADC12_INPUT_CHAN_2)
696 /*! @brief Channel for Phase W current */
697 #define ADC1_CURRENT_W_CH       (DL_ADC12_INPUT_CHAN_1)
```

Define the ADC interrupt trigger source. Use the last ADC MEM for phase current as the trigger source

Define the ADC channel used by users' hardware circuit. Modify the CHAN accordingly.

Figure 7-18 Phase Current ADC Channel Macros

Depending on the current sensing method of the hardware design, users should overwrite source code of the ***HAL_GD_ConfigureCurrentChannels()*** API and ADC channel macros for phase currents in ***gateDriver.h*** file to sample phase current correctly.

7.2.3.4.1.1 Three Shunt Configuration

Figure 7-19 shows the three or dual shunt configuration signal chain. Recommend using two ADC instances to sample and convert three phase currents for MSPM0G series.

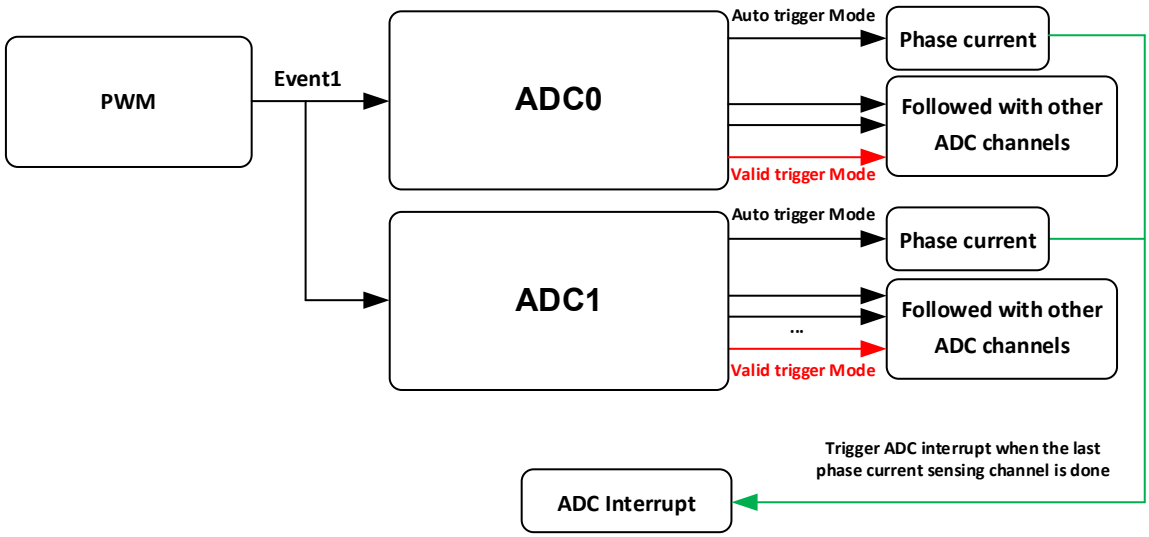


Figure 7-19. Three or Dual Shunt Configuration Signal Chain

Follow the steps below to modify phase current channels according to the hardware circuit.

1. Add the corresponding macro definition (**Section 7.2.3.2**) into ***main.h*** file
2. Modify the ADC channels in ***gateDriver.h*** file according to your hardware design
3. If Simultaneously Sampling is not enabled, users can set the ADC0_CURRENT_V_CH and ADC1_CURRENT_V_CH as the same
4. Select the last ADC MEM Index for phase current sampling as the interrupt trigger source. For example, if users set all three phase currents to the ADC0, then FOC application uses three ADC MEM Index (MEM0, MEM1, MEM2), so that the FOC_ADC_MEM_RES_LOAD should be set as ADC MEM2
5. Add macro FOC_ISR_ADC1 in ***gateDriver.h*** file if the Interrupt is used with ADC1 Instance
6. Initialize ADC Memory accordingly and set the ADC interrupt source in SysConfig. The ADC channel setting is optional as FOC application calls ***HAL_GD_ ConfigureCurrentChannels()*** to re-configure it
7. Overwrite the source code in ***HAL_GD_ConfigureCurrentChannels()*** API to meet the ADC instance and ADC Memory Index configuration implemented in step 2 ~ 6, as the ADC instance setting and memory read selection is hardcoded
8. Overwrite the source code in ***HAL_GD_HAL_GD_ReadCurrents()*** API to meet the ADC instance and ADC Memory Index configuration implemented in step 7

Figure 7-20 shows example of modifying all phase current channels mapping to ADC1. Example ADC channels setting refers to below:

Phase U current channel: ADC1.1 -> ADC MEM0

Phase V current channel: ADC1.2 -> ADC MEM1

Phase U current channel: ADC1.3 -> ADC MEM2 -> Trigger ADC1 Interrupt

Step 1: Add the current shunt configuration

```
#elif defined CUSTOM

/*! @brief TIDA010250 has inverting isense */
#define _INVERT_ISEN
/*! @brief IPD feature is enabled */
#define __IPD_ENABLE
/*! @brief TIDA010250 propagation delay */
#define DRIVER_PROPAGATION_DELAY_nS                500
/*! @brief TIDA010250 minimum on time (rise time + settling time) */
#define DRIVER_MIN_ON_TIME_nS                      8000
/*! @brief DC voltage base value */
#define DC_VOLTAGE_BASE                             448.0
/*! @brief Full scale readable current used as current base value,
calculated using (FULL Scale Voltage(3.3)/2* CSA Gain) */
#define FULL_SCALE_CURRENT_BASE                     8.25
/*! @brief Current shunt configuration */
#define __CURRENT_THREE_SHUNT_DYNAMIC
/*! @brief Enable dynamic current shunt changing */
#define DYNAMIC_CURRENT_SHUNT_CONFIG_EN             FALSE//TRUE

#endif
```

Step 2 & 3: Set ADC Current Channel

```
/*! @brief Channel for Phase U current */
#define ADC0_CURRENT_U_CH         (DL_ADC12_INPUT_CHAN_1)
/*! @brief Channel for Phase V current */
#define ADC0_CURRENT_V_CH         (DL_ADC12_INPUT_CHAN_2)
/*! @brief Channel for Phase V current */
#define ADC1_CURRENT_V_CH         (DL_ADC12_INPUT_CHAN_2)
/*! @brief Channel for Phase W current */
#define ADC1_CURRENT_W_CH         (DL_ADC12_INPUT_CHAN_3)
```

This example uses all ADC1 channels.
And simultaneously sampling is not enabled.

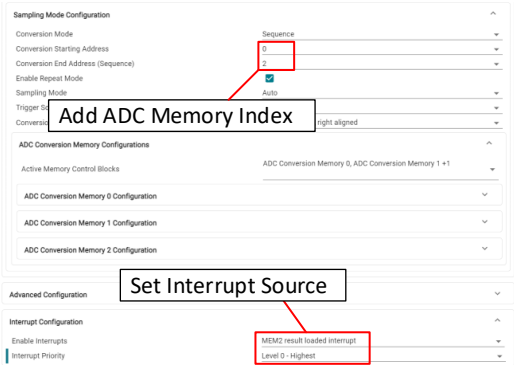
Step 4 & 5: Set ADC Interrupt Source

```
/*! @brief Instance for ADC Interrupt */
#define FOC_ADC_ISR_INST          (ADC1_INST)
/*! @brief Memory Load Register for ADC Interrupt */
#define FOC_ADC_MEM_RES_LOAD       DL_ADC12_IIDX_MEM2_RESULT_LOADED

#define FOC_ISR_ADC1
```

This example uses all ADC1 channels, so the ADC MEM is increased to 2 (0,1,2)
Due to ADC1 is used with larger Memory Index for phase current smapling, there add FOC_ISR_ADC1 macro for FOC application

Step 6: Configure SysConfig



Step 7: HAL_GD_ConfigureCurrentChannels()

```
void HAL_GD_ConfigureCurrentChannels(CURRENT_SHUNT_TYPES currentShunt)
{
    /* Configure the ADC channels based on the ADC pin configurations */
    switch(currentShunt)
    {
        case CURRENT_THREE_SHUNT_DYNAMIC:
        case CURRENT_THREE_SHUNT_AB_C:
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC0_IDX_1, ADC0_CURRENT_V_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_2, ADC1_CURRENT_W_CH);
            break;
        case CURRENT_THREE_SHUNT_A_BC:
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_1, ADC1_CURRENT_V_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_2, ADC1_CURRENT_W_CH);
            break;

        case CURRENT_TWO_SHUNT_A_B:
            HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_V_CH);
            break;

        case CURRENT_TWO_SHUNT_A_C:
            HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
            break;

        case CURRENT_TWO_SHUNT_B_C:
            HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_V_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
            break;

        case CURRENT_SINGLE_SHUNT:
            break;
        default:
            break;
    }
}
```

This example uses all ADC1 channels, so the ADC MEM is increased to 2 (0,1,2). The ADCX_INST and ADC_IDX should be both overwritten according to the hardware circuit.
As using three shunt method, here only requires to modify the THREE_SHUNT related branch.

Step 8: HAL_GD_ReadCurrents()

```
void HAL_GD_ReadCurrents(HAL_MEASURE_CURRENT_T *pCurrent)
{
    int32_t adc0Idx0,adc0Idx1,adc1Idx0,adc1Idx1;

    MC_ABC_T* iabcRaw = &pCurrent->iabcRaw;

    CURRENT_SHUNT_TYPES currentShunt = pCurrent->currentShunt;

    //ADC1-MEM0
    adc0Idx0 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC0_IDX_0);
    //ADC1-MEM1
    adc0Idx1 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC0_IDX_1);
    adc1Idx0 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC1_IDX_1);
    //ADC1-MEM2
    adc1Idx1 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC1_IDX_2);

    switch(pCurrent->currentShunt)
    {
        case CURRENT_THREE_SHUNT_DYNAMIC:
        case CURRENT_THREE_SHUNT_AB_C:
            iabcRaw->a = adc0Idx0;   //ADC1-MEM0
            iabcRaw->b = adc0Idx1;   //ADC1-MEM1
            iabcRaw->c = adc1Idx1;   //ADC1-MEM2
        break;

        case CURRENT_THREE_SHUNT_A_BC:
            iabcRaw->a = adc0Idx0;   //ADC1-MEM0
            iabcRaw->b = adc1Idx0;   //ADC1-MEM1
            iabcRaw->c = adc1Idx1;   //ADC1-MEM2
        break;

        case CURRENT_TWO_SHUNT_A_B:
            iabcRaw->a = adc0Idx0;
            iabcRaw->b = adc1Idx0;
        break;

        case CURRENT_SINGLE_SHUNT:

        break;
        default:
```

This example uses all ADC1 channels, so ADC MEM read back code should be modified according to Step 7.
The Phase U is read from ADC1 – MEM0 (iabcRaw->a)
The Phase V is read from ADC1 – MEM1 (iabcRaw->b)
The Phase W is read from ADC1 – MEM0 (iabcRaw->c)

Figure 7-20 Modify ADC Current Channel in Three Shunt Configuration

7.2.3.4.1.2 Dual Shunt Configuration

Refer to the flow for Three Shunt Configuration (**Section 7.2.3.4.1.1**) for dual shunt also. For Step 1, users should select dual shunt configuration macro from the available CURRENT_TWO_SHUNT_X_Y (X, Y = A, B, C). For Step 7 and Step 8, only the used CURRENT_TWO_SHUNT_X_Y (**Section 7.2.3.2.3**) branch requires to be overwritten, others can remain unchanged.

7.2.3.4.1.3 Single Shunt Configuration

Figure 7-21 shows the single shunt configuration flow.

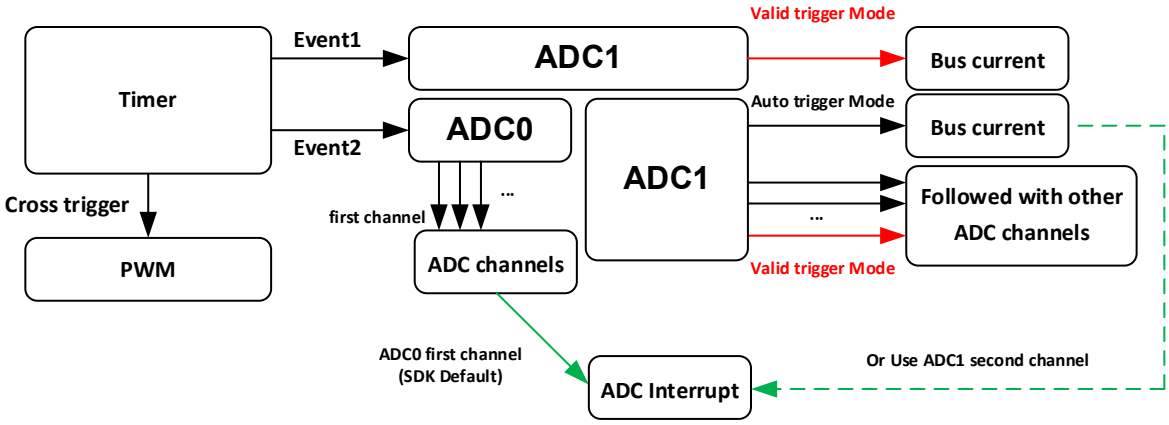


Figure 7-21. Single Shunt Configuration Signal Chain

Follow the steps below to modify phase current channels according to the hardware circuit.

1. Add the corresponding macro definition __CURRENT_SINGLE_SHUNT into ***main.h*** file
2. Modify the ADC channels in ***gateDriver.h*** file according to your hardware design. Single shunt requires one ADC channel and two ADC MEM Index as it samples twice in one PWM period
3. Select ADC MEM0 Index of the other ADC instance as the interrupt trigger source. For example, if users use ADC1 for bus current sampling, then use ADC0 instance for interrupt trigger
    a. If ADC1 interrupt is used, add macro FOC_ISR_ADC1 into ***gateDriver.h*** file
4. Initialize ADC Memory accordingly and set the ADC interrupt source in SysConfig. The ADC channel setting is optional as FOC application calls ***HAL_GD_ ConfigureCurrentChannels()*** to re-configure it
    a. If ADC1 interrupt is used, switch Event Trigger for ADC0 and ADC1 in TIMA1

Figure 7-22 shows an example of modifying ADC current channel mapping in Single Shunt Configuration.

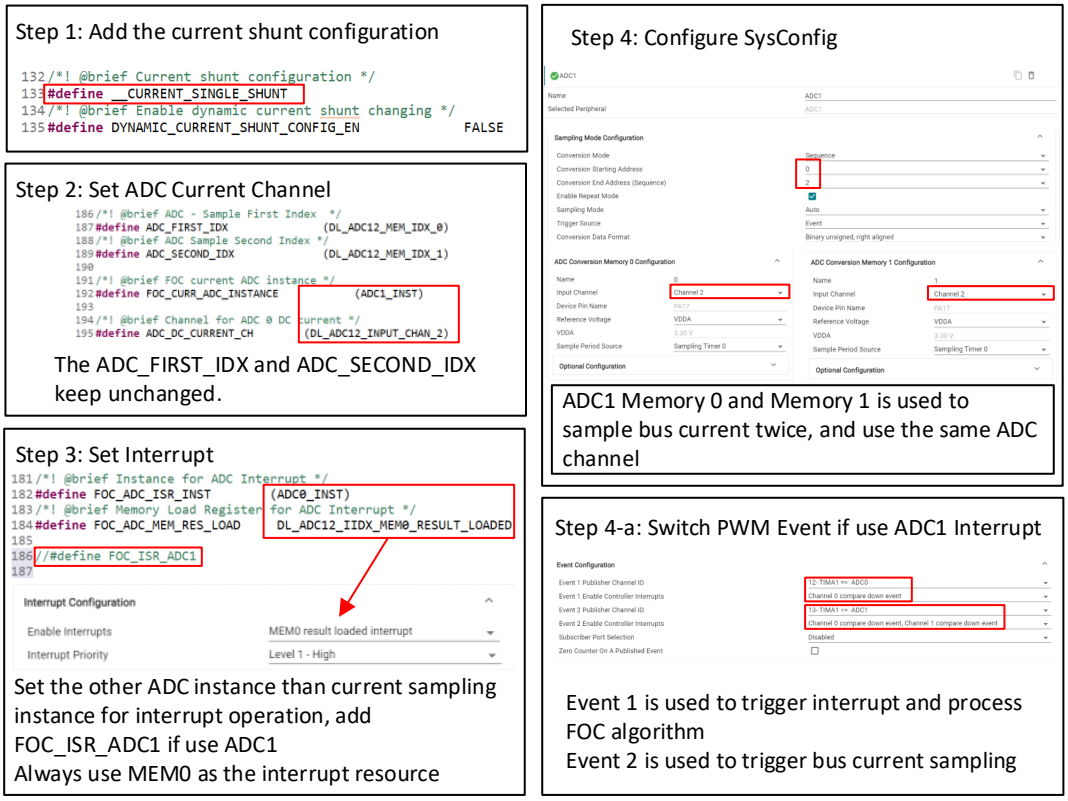


Figure 7-22 Modify ADC Current Channel in Single Shunt Configuration

Note

Migration is not restricted to one certain pattern. Users should take care of the interrupt setting in single shunt configuration, including setting proper event trigger in Timer, and setting proper trigger mode and interrupt trigger in ADC to meet the requirements defined in Figure 7-20.

7.2.3.4.2 Bus Voltage Channel

Follow the steps below to modify bus voltage channel according to the hardware circuit.

1. Assign the bus voltage channel to the desired ADC Instance and ADC Memory in SysConfig
2. Overwrite the HAL Layer Macro in ***gateDriver.h*** file accordingly

Figure 7-23 shows an example to modify voltage channel mapping: assign bus voltage channel (ADC0_4 defined in new hardware circuit) to ADC0 Memory 3.

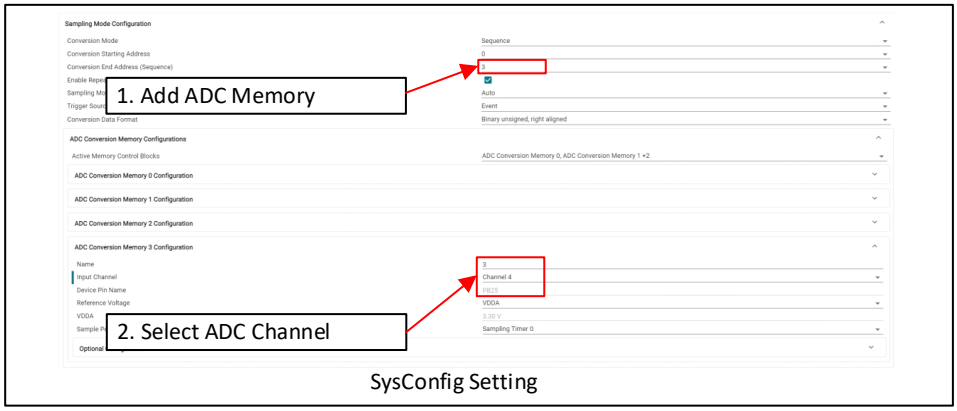


Figure 7-23 Modify Bus Voltage Channel

Recommend using ADC0 or ADC1 MEM2 for bus voltage sampling and keeping other ADC channel followed by bus voltage channel.

7.2.3.4.3 Phase Voltage Channels

Follow the steps below to modify phase voltage channels according to the hardware circuit.

1. Make sure the SysConfig has initialized the ADC Memory index. No need to be set with corresponding voltage channel, as FOC application calls ***HAL_GD_ConfigureVoltageChannels()*** to re-configure it
2. Overwrite the HAL Layer Macro in ***gateDriver.h*** file according to hardware circuit

Figure 7-24 shows an example of modifying phase voltage channel mapping.

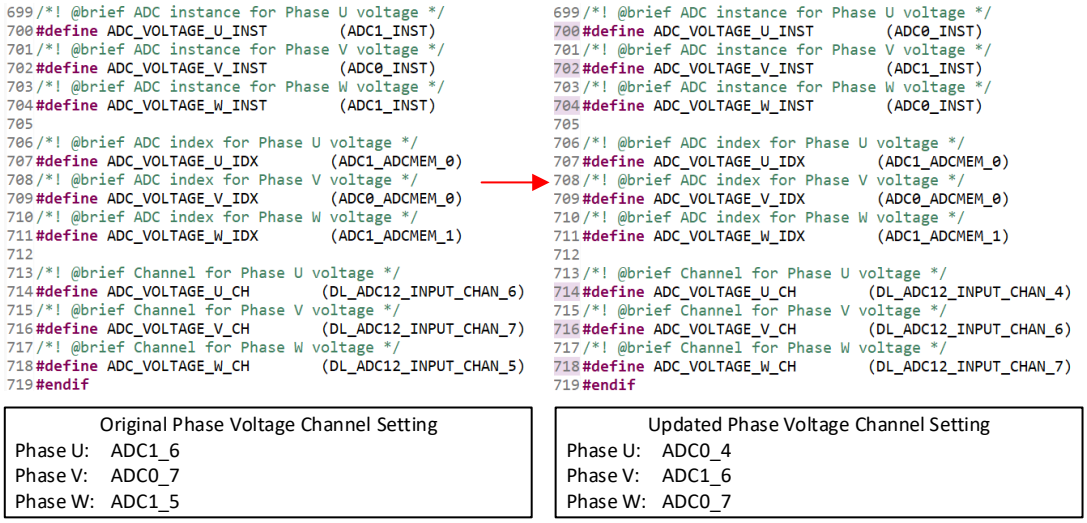


Figure 7-24 Modify Phase Voltage Channel

The ADC Memory Index can be left unchanged unless all Index is loaded to the same ADC Instance. If no voltage channel is used, users can set the same channel configuration as phase current to avoid compiler error.

7.2.3.5 Trigger Mode

FOC application start the ADC conversion by a hardware event generated by PWM module to avoid software delay. The ADC repeat mode is used with valid trigger mode to implement periodically conversion in every PWM cycle. Table 7-12 introduces the difference between auto step and valid trigger mode.

Table 7-12 ADC Trigger Mode Behavior

ADC Trigger Mode	Description
Auto step to next conversion	When ADC finishes current memory conversion, it automatically steps to next memory and starts conversion.
Valid trigger to next conversion	When ADC finishes current memory conversion, it waits for another trigger signal to start conversion at next memory. If no trigger signal occurs, ADC stays at current ADC memory and with no operation.

Follow the steps below to set proper trigger mode of the ADC channels for FOC application.

7.2.3.5.1 Three or Dual Shunt Configuration

For three or dual shunt configurations, FOC application converts phase currents one time in one PWM period. The ADC instance used for phase current sampling must set and *only* set the last ADC memory Index with Valid Trigger mode. So that in every PWM period cycle, PWM module generates an event to ADC module when low side MOSFET all ON, and this event triggers ADC to sample and convert phase currents. Figure 7-25 shows the SysConfig setting of ***sensorless-foc_DRV8316*** project.

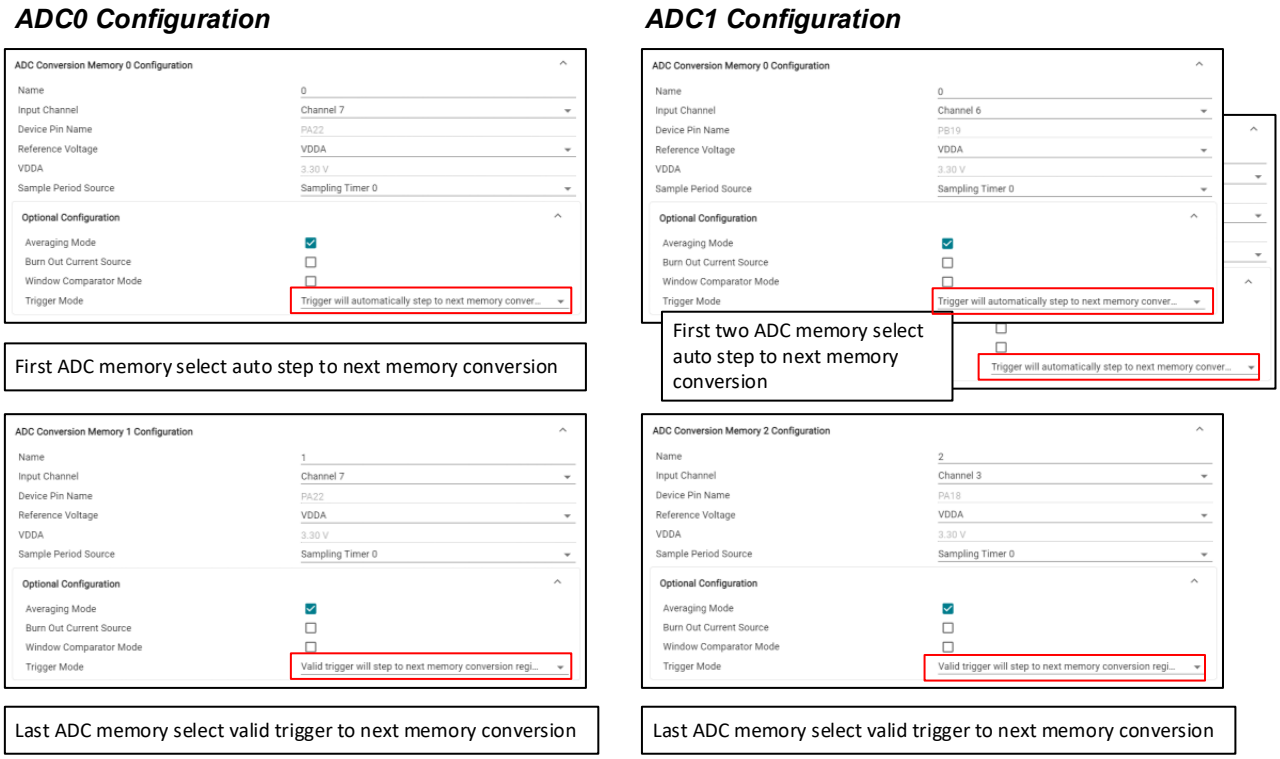


Figure 7-25 ADC Trigger Mode Setting in Three or Dual Shunt Configuration

7.2.3.5.2 Single Shunt Configuration

For single shunt configuration, FOC application converts phase currents two times in one PWM period and triggers the interrupt at the second time. The ADC instance used for phase current sampling must set and *only* set the **first** and **last** ADC memory Index with Valid Trigger mode. So that in every PWM period cycle, PWM module generates two events to ADC module according to space vector state, and events trigger ADC to sample and convert bus current in sequence. Figure 7-26 shows the SysConfig setting of ***sensorless-foc_DRV8329*** project.

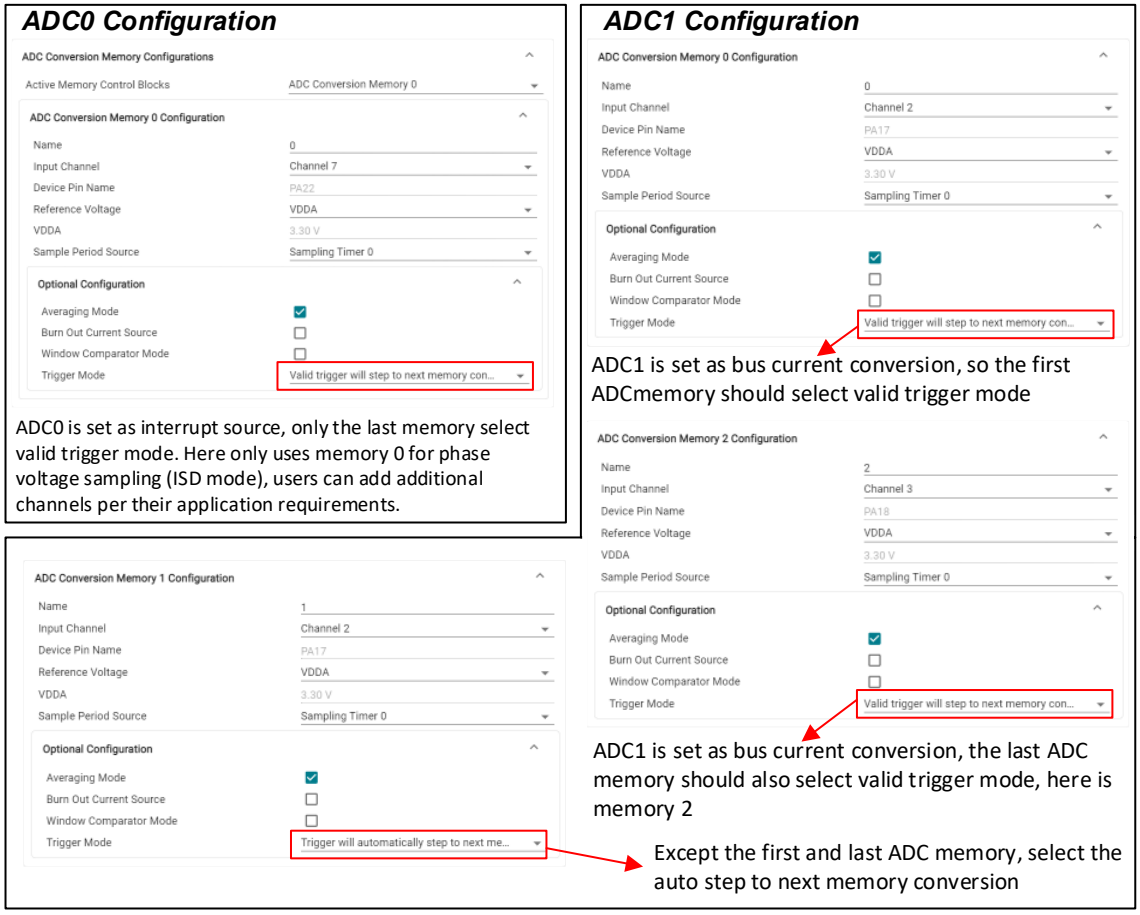


Figure 7-26 ADC Trigger Mode Setting in Single Shunt Configuration

Note

If using the ISD function in single shunt mode, the two phase voltage channels should be configured to the same ADC instance that does not sample the bus current; otherwise, it will not be possible to sample all three phase voltages simultaneously.

7.2.4 GPIO Pin

The GPIO used in FOC project mainly includes below parts:

1. HALL_GPIO_IN: Hall input signal, detailed in **Section 7.2.5**
2. FOC_GPIO_IN: to obtain direction and braking command input
3. FOC_GPIO_OUT: to control the gate driver power or indicate FOC fault
4. TST: test pins, not applied for FOC algorithm

Refers to **Section 7.2.5** if users want to reserve the GPIO function and assign new PINMUX. If users want to remove the GPIO pins, follow the steps below:

1. Delete the GPIO function in SysConfig
2. Remove the related API functions which use the related GPIO macros in ***focHALInterface.c*** file, shown in Figure 7-27.

```
927 void HAL_ClearNFault()
928 {
929 //    DL_GPIO_clearPins(FOC_GPIO_NFAULT_PORT, FOC_GPIO_NFAULT_PIN);
930 }
931
932 void HAL_SetNFault()
933 {
934 //    DL_GPIO_setPins(FOC_GPIO_NFAULT_PORT, FOC_GPIO_NFAULT_PIN);
935 }

1003 _Bool HAL_getDirPinStatus(void)
1004 {
1005     return 0; //DL_GPIO_readPins(FOC_GPIO_DIR_PORT, FOC_GPIO_DIR_PIN)?
1006               //          HAL_GPIO_HIGH : HAL_GPIO_LOW;
1007 }
1008
1009 _Bool HAL_getBrakePinStatus(void)
1010 {
1011     return 0; //DL_GPIO_readPins(FOC_GPIO_BRAKE_PORT, FOC_GPIO_BRAKE_PIN)?
1012               //          HAL_GPIO_HIGH : HAL_GPIO_LOW;
1013 }
```

Figure 7-27 Remove Preset GPIO Function

7.2.5 HALL Pin

The three Hall Sensor input signals are to be fed through general-purpose input/output (GPIO) inputs and Events are generated based on the changes in GPIO levels to a specific timer. Using SysConfig to overwrite the HALL pins according to your hardware design.

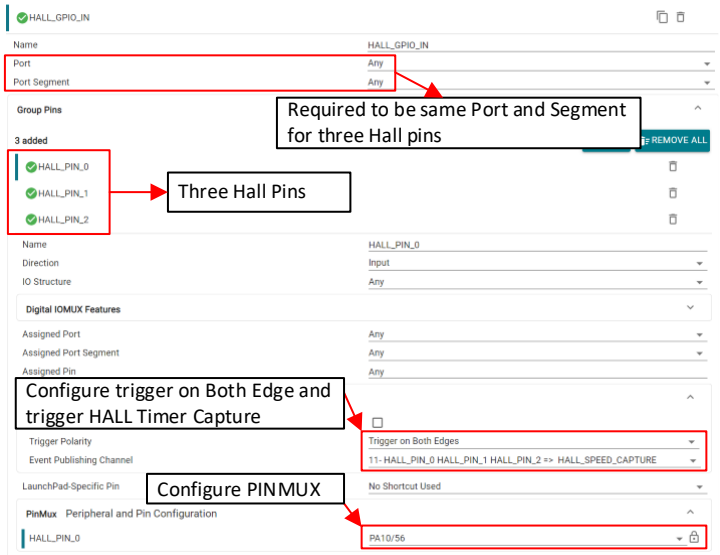


Figure 7-28 Modify HALL Pin

7.2.6 UART Module

The FOC project uses UART to communicate with GUI to support motor tuning without CCS IDE. The DMA is used to improve UART transmission efficiency and not impacted by FOC ISR.

Following the steps below to remove the UART module:

1. Delete the UART module and CRC module in SysConfig
2. Exclude the ***uart_comm.c*** file from build
3. Remove the related API functions which use the related UART or DMA macros in ***main.c*** and ***ISR.c*** files

Figure 7-29 shows example of removing UART module. Then users can then add UART module back and apply their own communication protocol.

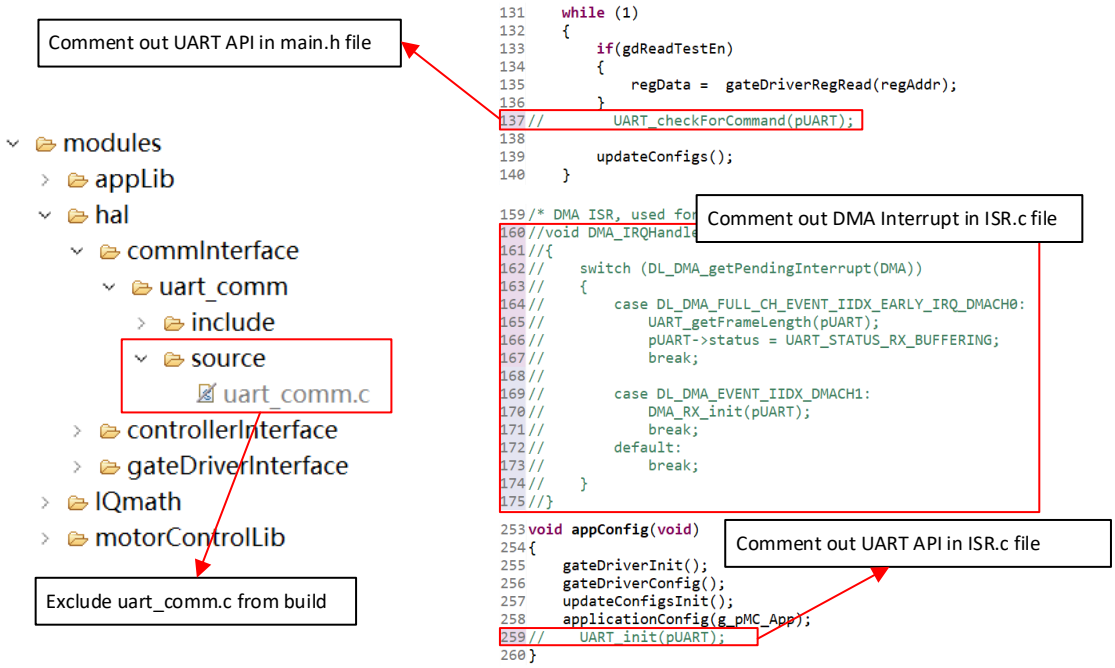


Figure 7-29 Remove UART Module

7.2.7 DAC12 Module

FOC application enables DAC12 module to track the variable in real-time running at MCU pin by default. Following the steps below to remove the DAC12 module:

1. Delete the DAC12 module in SysConfig
2. Comment out the code of DAC output generation in ***FOC_ADC_ISR()*** API (located in ***ISR.c*** file)

```
//    if(pUserCtrlRegs->dacCtrl.dacEn != 0)
//    {
//        if(pUserCtrlRegs->dacCtrl.dacScalingFactor != 0)
//        {
//            dacWriteData = _IQmpy_mathacl(
//                        *((uint32_t *)pUserCtrlRegs->dacCtrl.dacOutAddr),
//                         pUserCtrlRegs->dacCtrl.dacScalingFactor);
//            /* Adding offset value to half of reference voltage */
//            dacWriteData = _IQtoIQ12((dacWriteData >> 1) + _IQ(0.5));
//        }
//        else
//        {
//            dacWriteData = *((uint32_t *)pUserCtrlRegs->dacCtrl.dacOutAddr);
//            if((pUserCtrlRegs->dacCtrl.dacShift)>=0)
//            {
//                dacWriteData <<= pUserCtrlRegs->dacCtrl.dacShift;
//            }
//            else
//            {
//                dacWriteData >>= (-1*pUserCtrlRegs->dacCtrl.dacShift);
//            }
//        }
//        DL_DAC12_output12(DAC0, dacWriteData);
//    }
}
```

Figure 7-30 Remove DAC12 Module

7.2.8 CAPTURE Module

A timer is enabled in capture mode for IPD pulse time counting. If the IPD function is not used, follow the steps below to delete the CAPTURE module.

1. Delete the CAPTURE module in SysConfig
2. Comment out the macro __IPD_ENABLE in ***main.h*** file
3. Comment out the ***IPD_ADC_init()*** API (located in ***focHALInterface.c*** file)

main.h

```
159 /*! @brief DRV8316 has inverting isense */
160 #define _INVERT_ISEN
161 /*! @brief IPD feature is enabled */
162 //#define __IPD_ENABLE
163 /*! @brief DRV8323 propagation delay */
164 #define DRIVER_PROPAGATION_DELAY_nS                100
165 /*! @brief DRV8316 minimum on time (rise time + settling time) */
```

focHALInterface.c

```
1021 static void IPD_ADC_init(ADC12_Regs *adc12Inst, uint32_t chansel)
1022 {
1023 //    DL_ADC12_setClockConfig(adc12Inst, (DL_ADC12_ClockConfig *) &gADC_IPDClockConfig);
1024 //    DL_ADC12_initSingleSample(adc12Inst,
1025 //        DL_ADC12_REPEAT_MODE_ENABLED, DL_ADC12_SAMPLING_SOURCE_AUTO, DL_ADC12_TRIG_SRC_SOFTWARE,
1026 //        DL_ADC12_SAMP_CONV_RES_12_BIT, DL_ADC12_SAMP_CONV_DATA_FORMAT_UNSIGNED);
1027 //    DL_ADC12_configConversionMem(adc12Inst, DL_ADC12_MEM_IDX_0,
1028 //        chansel, DL_ADC12_REFERENCE_VOLTAGE_VDDA, DL_ADC12_SAMPLE_TIMER_SOURCE_SCOMP0, DL_ADC12_AVE
1029 //        DL_ADC12_BURN_OUT_SOURCE_DISABLED, DL_ADC12_TRIGGER_MODE_AUTO_NEXT, DL_ADC12_WINDOWS_COMP_MO
1030 //    DL_ADC12_setPowerDownMode(adc12Inst,DL_ADC12_POWER_DOWN_MODE_MANUAL);
1031 //    DL_ADC12_setSampleTime0(adc12Inst,2);
1032 //    DL_ADC12_setPublisherChanID(adc12Inst,FOC_IPD_EVENT_CH);
1033 //#ifdef _NONINVERT_ISEN
1034 //    DL_ADC12_enableEvent(adc12Inst,DL_ADC12_EVENT_WINDOW_COMP_HIGH);
1035 //#else
1036 //    DL_ADC12_enableEvent(adc12Inst,DL_ADC12_EVENT_WINDOW_COMP_LOW);
1037 //#endif
1038 //    DL_ADC12_enableConversions(adc12Inst);
1039 }
```

Figure 7-31 Remove CAPTURE Module

8 Frequently Asked Questions (FAQs)

8.1 Set Higher Deadband Time

Currently there is provision to set the deadband time as a 6-bit configurable value in ***pUserInputRegs***->*periphCfg1* register. So, the register provides a Max of 3.2uS. In cases where the higher deadband time is needed, users can update this register to increase the size of MCU_DEAD_TIME to higher number of bits in ***appInputCtrlInterface.h*** file to enable larger deadband time.

Figure 8-1 shows an example to configure deadband time to 8-bit.

```
458 /*! @brief userInputPeriCfg1 structure */
459 typedef struct
460 {
461     uint32_t
462     /*! Response to change of DIR pin status */
463     dirChangeMode:            1,
464     /*! DIR pin override */
465     dirInput:                 2,
466     /*! Bus current limit enable */
467     busCurrLimitEnable:       1,
468     /*! Bus current limit */
469     busCurrLimit:             5,
470     /*! deadtime for PWM outputs */
471     mcuDeadTime:              8,
472     /*! Reserved */
473     reserved:                 15;
474 }userInputPeriCfg1;
```

Figure 8-1 Set the Deadband Time Register to 8-bit

8.2 Reduce 1x ADC Pin for Simultaneously Sampling in Three Shunt Config

In FOC applications, the simultaneous sampling feature in three shunt current sensing method is enabled with four ADC channels.

For MSPM0GX50X series, there are two unique ADC channels that are mapped to ADC0 and ADC1 instance both as shown below. Select one of these channels for phase B current sampling.