Application Note

AM263x for Traction Inverters

Feng Qi

ABSTRACT

Traction inverters are the heart of electric vehicles. To accomplish real time control of a inverter, a micro-controller must have the combination of low latency peripherals and deterministic processing time. To accelerate market adoption of the micro-controller, it is critical to have a well architected inverter software framework demonstrating its potentials. Such a framework will also significantly help customers reduce software development time and focus on development of end equipment. This work focuses on presenting customers the framework for traction inverters and showing customers the real time control modules of AM263x. At the end of this publication, guidelines are summarized for customers moving to AM263x from other micro-controllers.

Table of Contents

1 Introduction ... 3
2 A Step-by-Step Guide to Running a Traction Inverter ... 5
2.1 Create Real Time Debug Interface ... 5
2.2 Configure Control Peripheral and ADC Interrupt With Sysconfig ... 14
2.3 Configure Gate Driver Interface With MSPI ... 23
2.4 Get Samples From ADC and Read Samples Via CCS ... 25
2.5 Generate Space Vector PWM and Drive Motor in Open Loop ... 27
2.6 Close Current Loop With Mock Speed ... 29
2.7 Add Software Resolver to Digital Converter ... 32
2.8 Close Speed Loop With Rotor Speed ... 35
3 A Brief Guide to Code Migration ... 38
3.1 SoC Architecture Overview ... 38
3.2 SDK Resources Overview ... 39
3.3 Code Migration From AM24 ... 39
3.4 Code Migration From C28 ... 40
4 Summary ... 40
5 References ... 40

List of Figures

Figure 1-1. Traction Framework Resources ... 4
Figure 1-2. Traction System Diagram ... 5
Figure 2-1. CCS GTI UART Driver for R5F ... 6
Figure 2-2. AM263x Control Card ... 6
Figure 2-3. Create New Target Configuration File ... 7
Figure 2-4. Select JTAG Connection and Device ... 7
Figure 2-5. Add UART Communication Port ... 8
Figure 2-6. Open Advanced Target Configuration ... 8
Figure 2-7. Add Component ... 9
Figure 2-8. Select CPU Properties ... 10
Figure 2-9. Find XDS110 UART COM Port ... 10
Figure 2-10. Update CPU Properties in Advanced Target Configuration ... 11
Figure 2-11. Disable UART Log in Debug Log ... 11
Figure 2-12. Configure UART0 Instance ... 12
Figure 2-13. Add Serial Monitor Function Calls ... 12
Figure 2-14. Locate Target Configuration File ... 13
Figure 2-15. Launch Selected Configuration ... 14
Figure 2-16. Disconnect JTAG Connection ... 14
Figure 2-17. Establish UART Connection ... 14

Figure 2-18. A Summary of EPWM7 Configuration....15
Figure 2-19. EPWM7 Time Base Configuration....16
Figure 2-20. EPWM7 Event Trigger Configuration....16
Figure 2-21. EPWM7 Counter Compare Configuration....17
Figure 2-22. EPWM0 Time Base Configuration....17
Figure 2-23. EPWM0 Event Trigger Configuration....18
Figure 2-24. EPWM0 Dead-Band Configuration....18
Figure 2-25. A Summary of ADC4 Configuration....19
Figure 2-26. ADC4 SOC Configuration....20
Figure 2-27. ADC4 INT Configuration....21
Figure 2-28. INT XBAR Configuration....21
Figure 2-29. A Summary of EDMA Configuration....22
Figure 2-30. EDMA Channel Trigger Configuration....23
Figure 2-31. Control Card Pinmux....24
Figure 2-32. A Summary of MCSPI Configuration....24
Figure 2-33. MCSPI Channel Configuration....25
Figure 2-34. Plotted Phase A Current at No Load....27
Figure 2-35. Phase A Duty Cycle....28
Figure 2-36. Phase A Current Open Loop....29
Figure 2-37. Open Loop Id....30
Figure 2-38. Open Loop Iq....31
Figure 2-39. Closed Loop Id....32
Figure 2-40. Closed Loop Iq....32
Figure 2-41. Software Resolver Synchronization and Excitation....33
Figure 2-42. Mother Board Top View....34
Figure 2-43. Mother Board Blue Wire for Resolver Excitation and DAC-A....34
Figure 2-44. Speed Loop Demonstration Program....36
Figure 2-45. Motor Forward Rotation....36
Figure 2-46. Motor Backward Rotation....37
Figure 2-47. Resolver Sine Envelope for a Transition from Forward to Backward....37
Figure 3-1. AM263x Block Diagram....38

List of Tables

Table 3-1. SDK directory structure....39
Table 3-2. Examples on Similarity of API Definitions....40

Trademarks

CoreSight™ is a trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
Code Composer Studio™ is a trademark of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
All trademarks are the property of their respective owners.

1 Introduction

AM263x is a system on chip (SoC) with Arm® Cortex®-R5F clusters and a dedicated accelerator for real time control. The clusters can be configured as either dual core mode or lockstep mode. And, the accelerator is named as control subsystem and includes interface modules like ADC, DAC, and PWM. This work takes core 0-0 as an example and demonstrates features of one R5F core and a group of interface modules for one traction inverter. This section gives a brief introduction on the R5F cluster and the control sub-system. More details on the SoC can be found in the *AM263x Sitara™ Microcontrollers Data Sheet* and the *AM263x Sitara Processors Technical Reference Manual*. Details on inverter hardware can be found in the *ASIL D Safety Concept-Assessed High-Speed Traction, Bi-directional DC/DC Conversion Reference Design*.

The Arm® Cortex®-R5F cluster includes two R5F cores accompanying memories like L1 cache and tightly-coupled memories (TCM), standard Arm CoreSight™ debug and trace architecture, integrated vectored interrupt manager (VIM), ECC aggregators, and various other modules for protocol conversion and address translation for easy integration into the SoC. More detailed block diagram can be found in AM263x technical reference manual. The key to solve real time control problem is to understand the impacts from cache and TCM. Instructions and data can be allocated to either On-Chip RAM or TCM by link command file when program is built. During execution, frequently used instructions and data in On-Chip RAM will be taken into cache automatically. As a result, execution time is significantly improved. But, the data in On-Chip RAM is not updated until it is written back from cache. When data is in cache, the only way to access is via instructions running in the core and memory view from integrated development environment (IDE) like Code Composer Studio™ (CCS) is not able to read cache. However, there is a way to read cache with CCS via a section of program operating universal asynchronous receiver/transmitter (UART) inside the core. Details on the UART method will be introduced in a following section. On the other hand, instructions and data allocated to TCM are kept at the address and available to memory view all the time. Generally, execution time of program in cache and TCM is very close but that of program in On-Chip RAM is much slower. And, the operation transferring program from On-Chip RAM to cache takes some time and introduces some non-predictive latency. If the latency is significant to the requirement from application, it is highly encouraged to store the application program in TCM. Details on TCM address can be found in AM263x technical reference manual. In this work, the interrupt program for field oriented control and software resolver to digital converter are located in TCM. The link command file is available as an example in CCS project folder.

The accelerator for real time control inherits Texas Instrument's classic C2000 control modules widely used around the world. It includes Analog-to-Digital Converter (ADC), Analog Comparator, Buffered Digital-to-Analog Converter, Enhanced Pulse Width Modulator (EPWM), Enhanced Capture, Enhanced Quadrature Encoder Pulse, Fast Serial Interface, Sigma Delta Filter Module, and Crossbar. Details on the modules are available in AM263x technical reference manual. With the help of an intuitive system configuration tool, SYSCONFIG, it is also possible to configure those modules with reduced exposure to implementation details. Details on SYSCONFIG can be found in AM263x Software Development Kit (SDK). The key for module synchronization is to configure PWM synchronization input/output in EPWM Time Base section and ADC Start-of-Conversion (SOC) trigger in EPWM Event-Trigger section. Time Base is to align multiple PWM channels while Event-Trigger is to synchronize features like ADC, DMA and Interrupt. One example for traction inverter is located in CCS project folder of traction inverter demo. In the example, one PWM channel is set to trigger updates for resolver excitation signal via DMA and DAC at higher frequency, and three PWM channels are used to create inverter signal and generate ADC SOC. In this way, resolver excitation signal from DAC is aligned to the desired phase for ADC samples. Multiple ADC units can share the same SOC. In other words, multiple samples can be taken simultaneously across multiple ADC units. Within one ADC unit, the sequence of samples can be configured in SOC Configuration section. And, ADC interrupt can be set in INT Configuration. The interrupt can be triggered at either the start of one ADC conversion or the end of one ADC conversion. Some simple examples on PWM and ADC are available in AM263x SDK under "\examples\drivers\epwm" and "\examples\drivers\adc". More details on the APIs can be found in AM263x SDK under "\source\drivers". The header files have great details in comments.

As we mentioned, there are some topics to be considered in development, like SoC architecture and SDK APIs. More details on those can be found in AM263x technical reference manual and SDK package. In the following sections, this work is going to focus on how to use the example project as a framework to accelerate development of traction inverter, and how to migrate existing project code to AM263x.

The framework includes the resources highlighted in Figure 1-1 and is built into the traction system in Figure 1-2. TIDM-02009 is the traction inverter reference design hardware. It includes a power model, gate drivers, current and voltage samples, resolver analog front end, and some connection interfaces. The motor includes a resolver taking sine wave excitation and sending modulated feedbacks for position sensing. Field Oriented Control is implemented with Cluster-0 Core-0. The real-time control section in ADC INT1 is allocated to TCM for the most deterministic execution time.

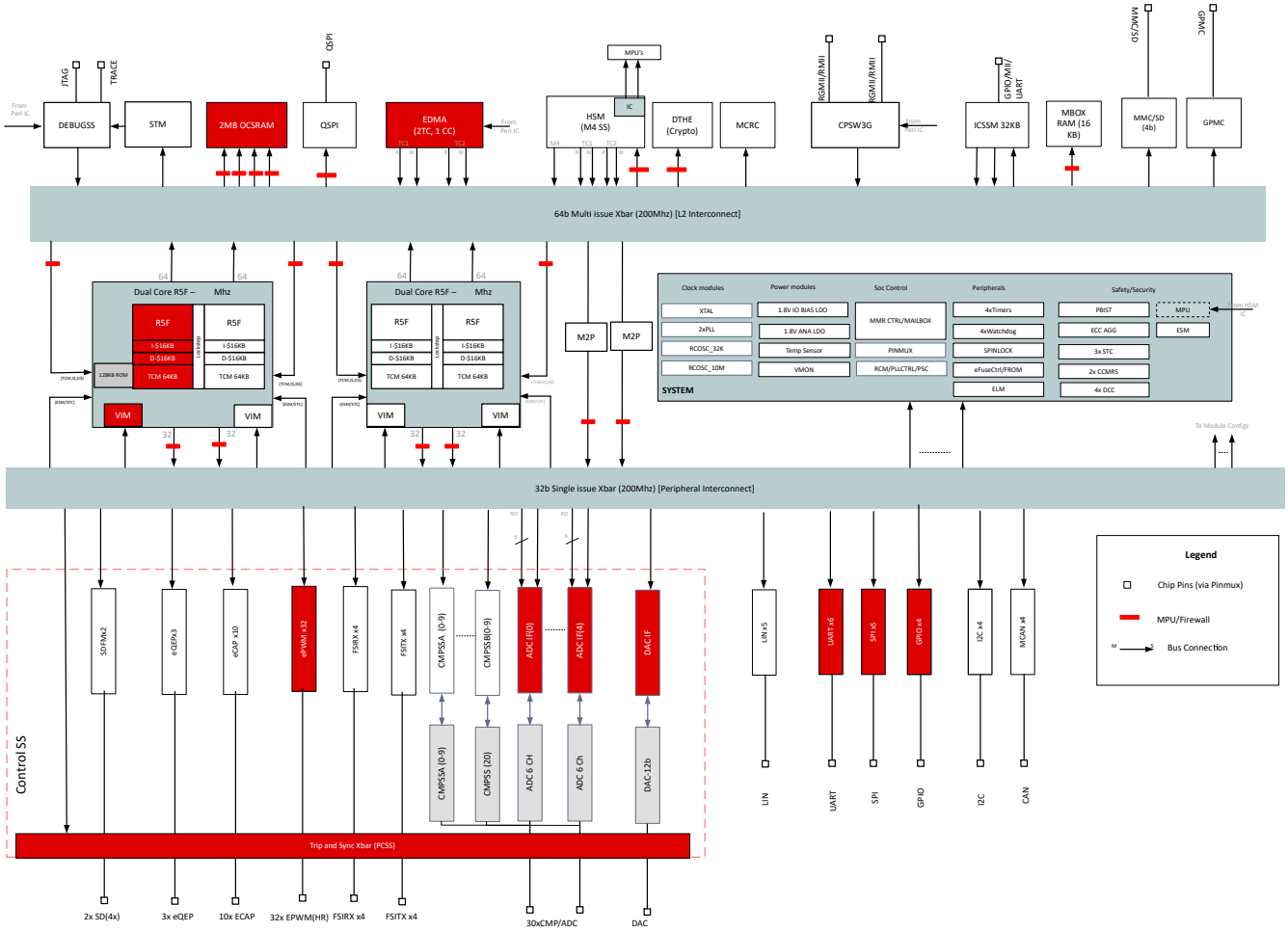

Figure 1-1. Traction Framework Resources

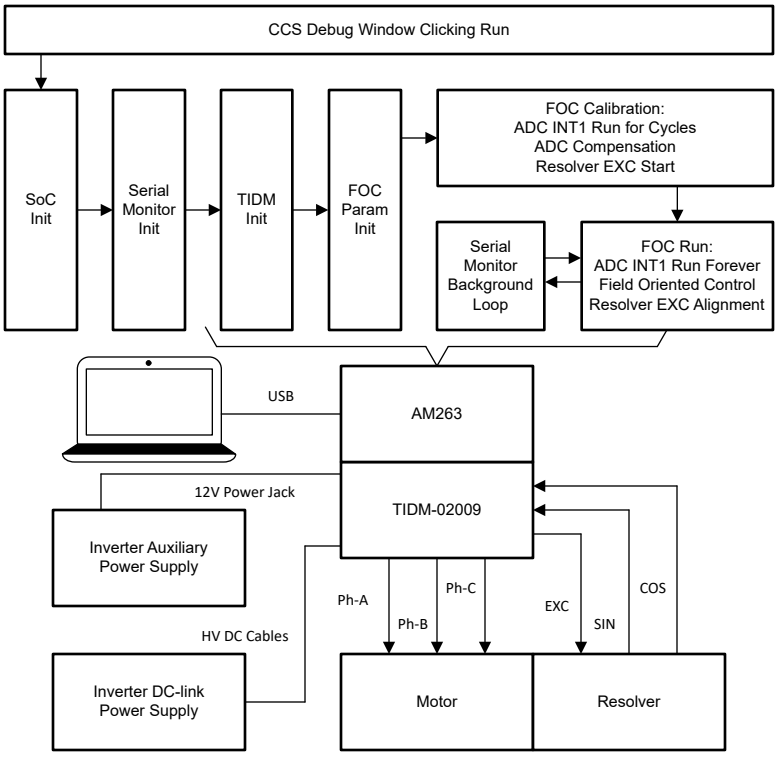

Figure 1-2. Traction System Diagram

2 A Step-by-Step Guide to Running a Traction Inverter

2.1 Create Real Time Debug Interface

For AM263x, real time debug is enabled by UART connection between CCS and AM263x. With real time debug, global variables can be added to expression window and ready for read/write during continuous run of the program. The connection is built by debug program in the listed files.

- Serial_Cmd_Monitor.c
- Serial_Cmd_Monitor.h
- Serial_Cmd_HAL.c
- Serial_Cmd_HAL.h

Even though there are four files listed here, there is only two functions required in application program. One is "SerialCmd_init()" called in initialization and the other is "SerialCmd_read()" called in background loop of BareMetal or low priority task of RTOS. This section focus on how to create the UART connection and how to launch real time debug in CCS.

2.1.1 Confirm CCS Features

It is recommended to check the following CCS driver file if the CCS version is older than 11.1. The configuration of Cortex_R5 should be similar to Figure 2-1. If any line is missing, it is necessary to add the line showing in Figure 2-1. As for content of the lines, COM Port and Baud Rate need to be updated in target configuration file, which is included in the next step.

- ccs\ccs_base\common\targetdb\drivers\gti_uart_driver.xml

```
<isa Type="Cortex_R5" ProcID="0x75803400">
    <driver file="../../../DebugServer/drivers/XPCOMToGTIAdapter.dvr">
        <property Type="stringfield" Value="COM14" id="COM Port" />
        <property Type="stringfield" Value="9600" id="Baud Rate" />
        <property Type="hiddenfield" Value="Little Endian" id="Endianness" />
        <property Type="hiddenfield" Value="32" id="Word Size Page 0" />
        <property Type="hiddenfield" Value="8" id="Minimum Addressable Size Page 0" />
        <property Type="hiddenfield" Value="@ti.com/UARTMonitor;1" id="XPCOM Class ID" />
        <property Type="hiddenfield" Value="Flash DLL Delegate" id="TargetAccess" />
        <connectionType Type="UARTConnection"/>
    </driver>
</isa>
```

Figure 2-1. CCS GTI UART Driver for R5F

2.1.2 Create Target Configuration File

For a control card in Figure 2-2, it offers both JTAG and UART ports in one USB port. Details on hardware connection can be found in *AM263x Control Card User's Guide*. It is necessary to create a target configuration file for the debug ports. A step-to-step guide is given in screen shots from Figure 2-3 to Figure 2-10. Briefly, a target configuration file is created and then configured with both JTAG and UART. The UART COM Port in Figure 2-10 should match PC Device Manager COM Port for JTAG probe Application/User UART. The Baud Rate in Figure 2-10 should be consistent with SoC UART Baud Rate configured in next step.

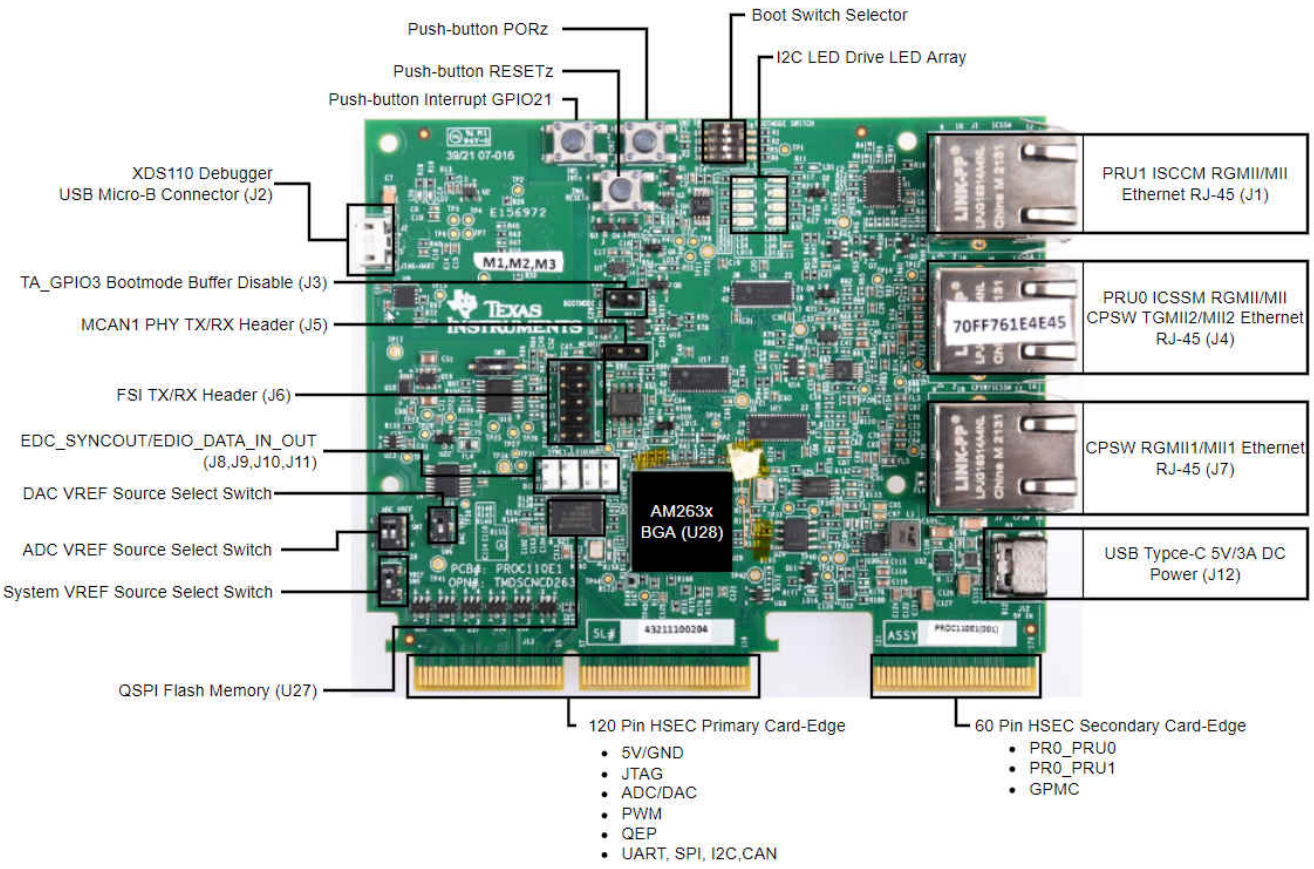

Figure 2-2. AM263x Control Card

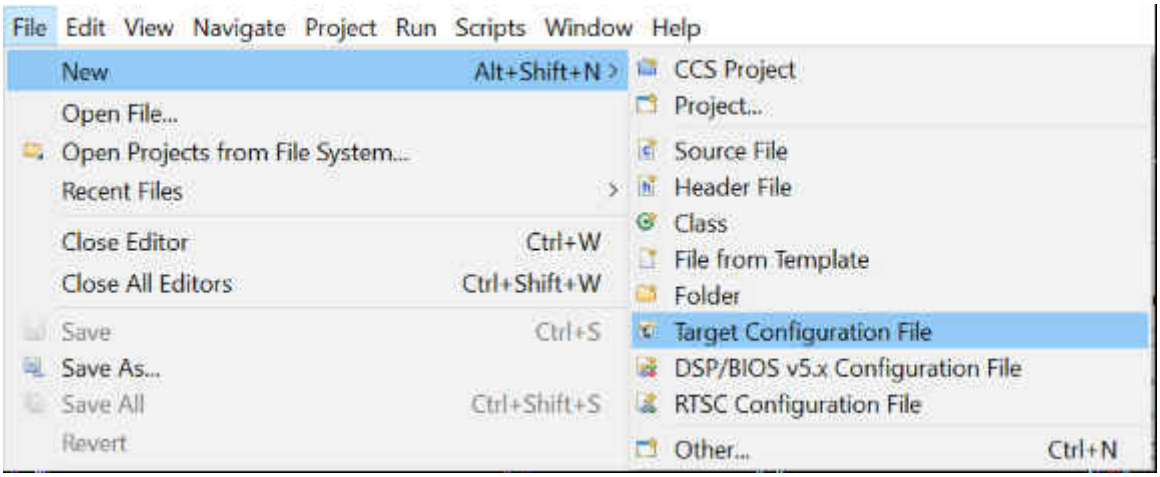

Figure 2-3. Create New Target Configuration File

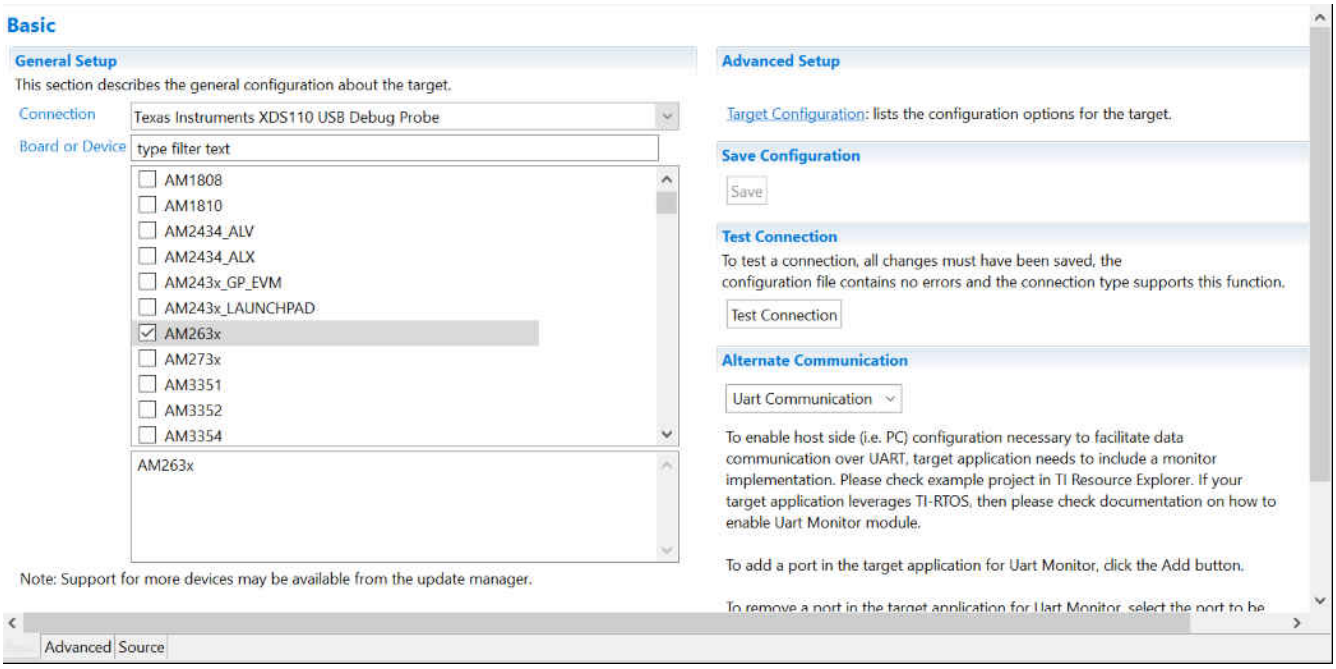

Figure 2-4. Select JTAG Connection and Device

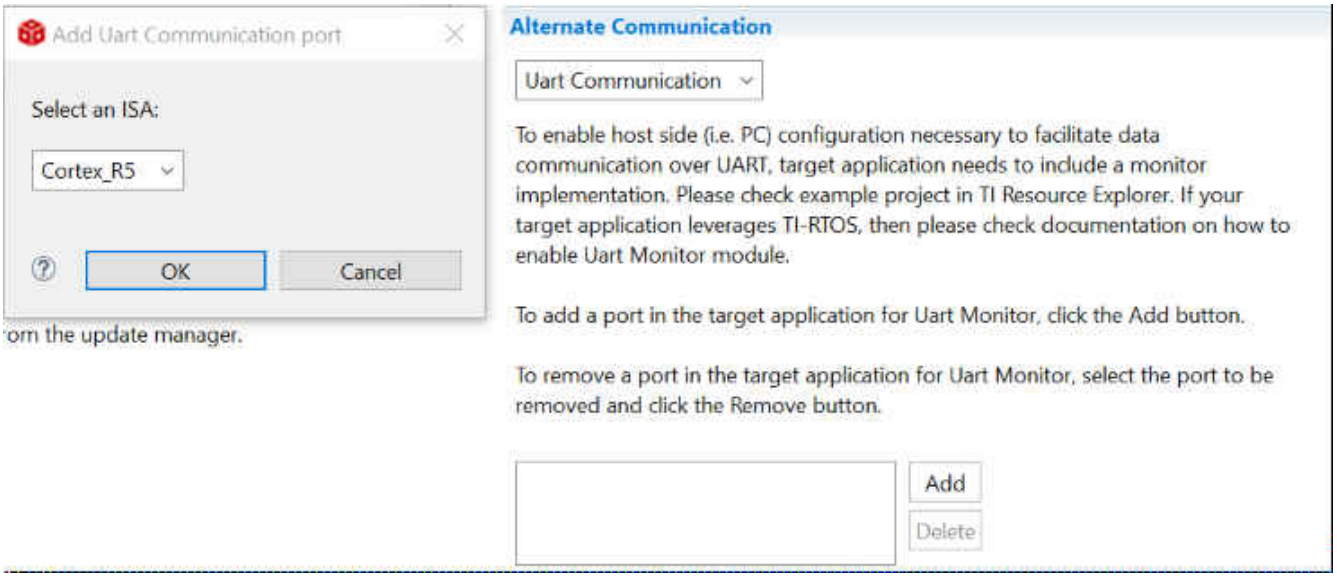

Figure 2-5. Add UART Communication Port

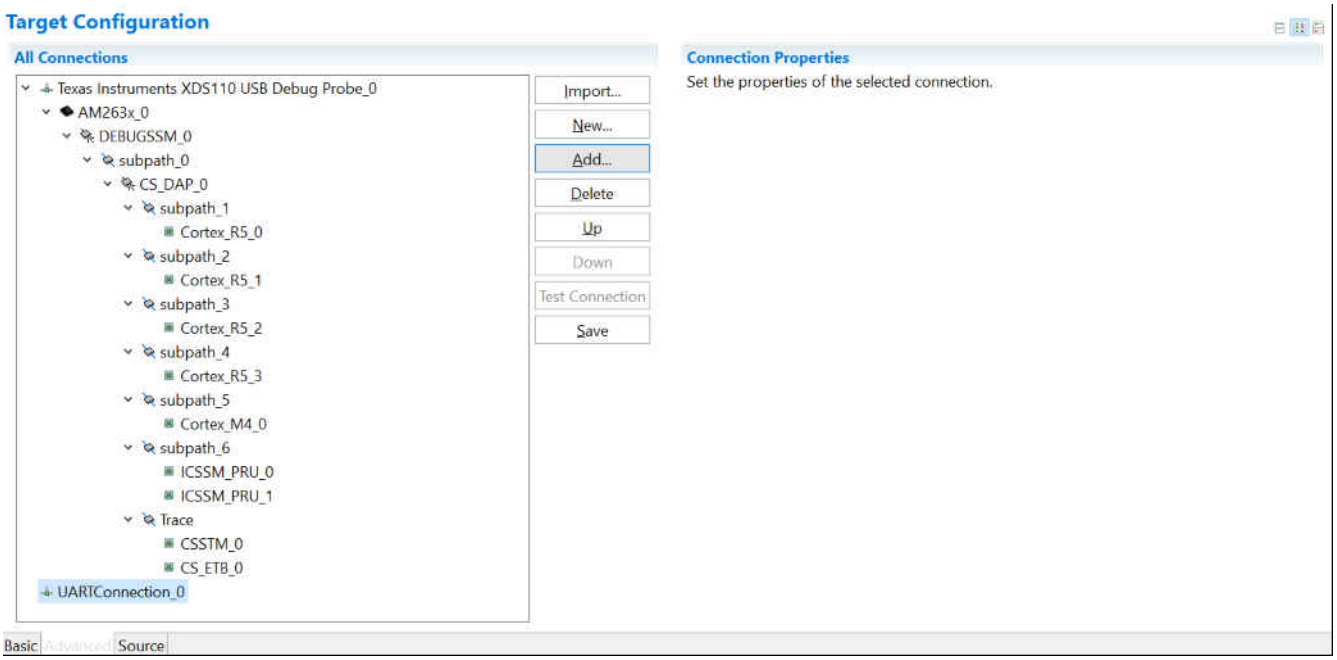

Figure 2-6. Open Advanced Target Configuration

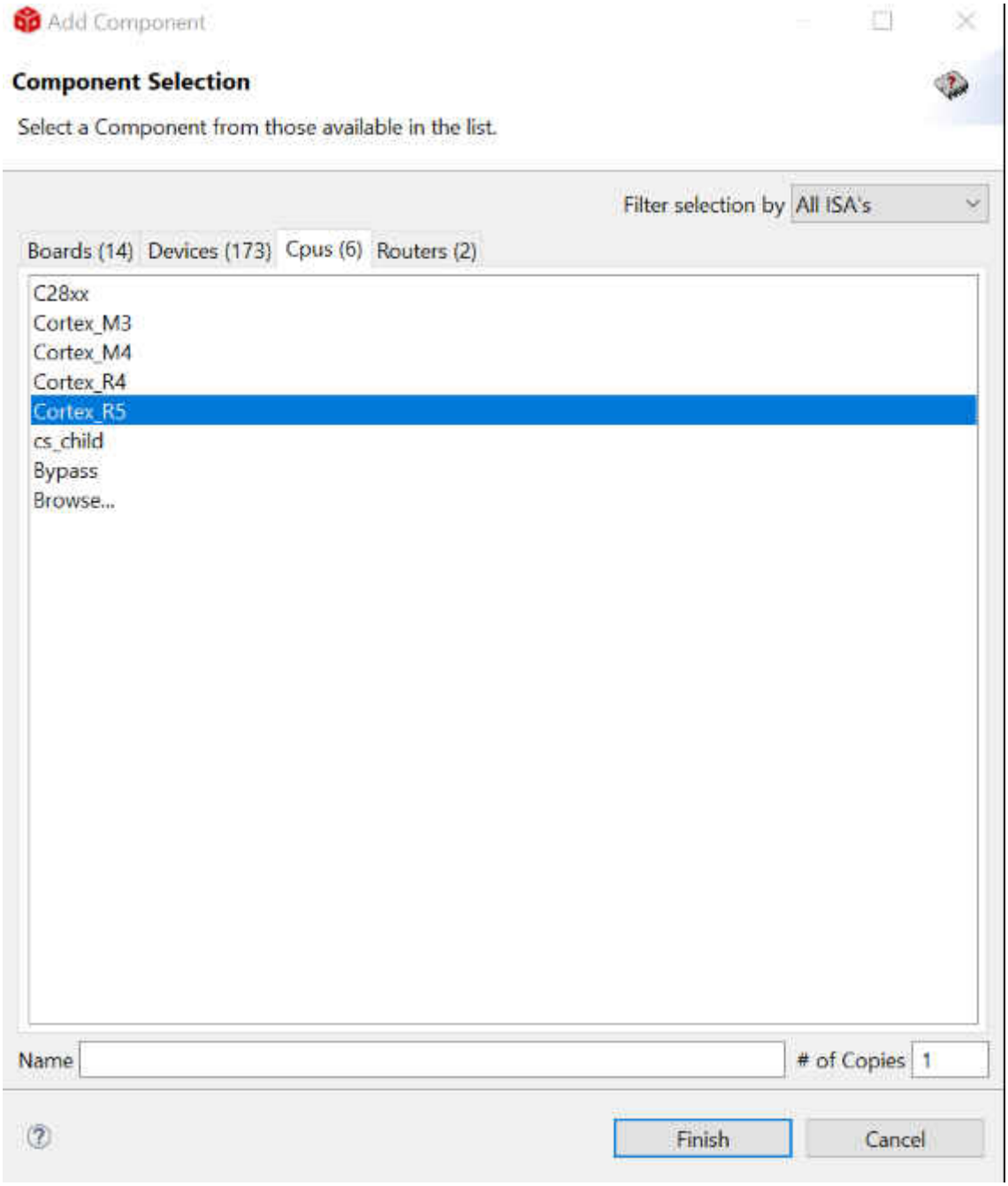

Figure 2-7. Add Component

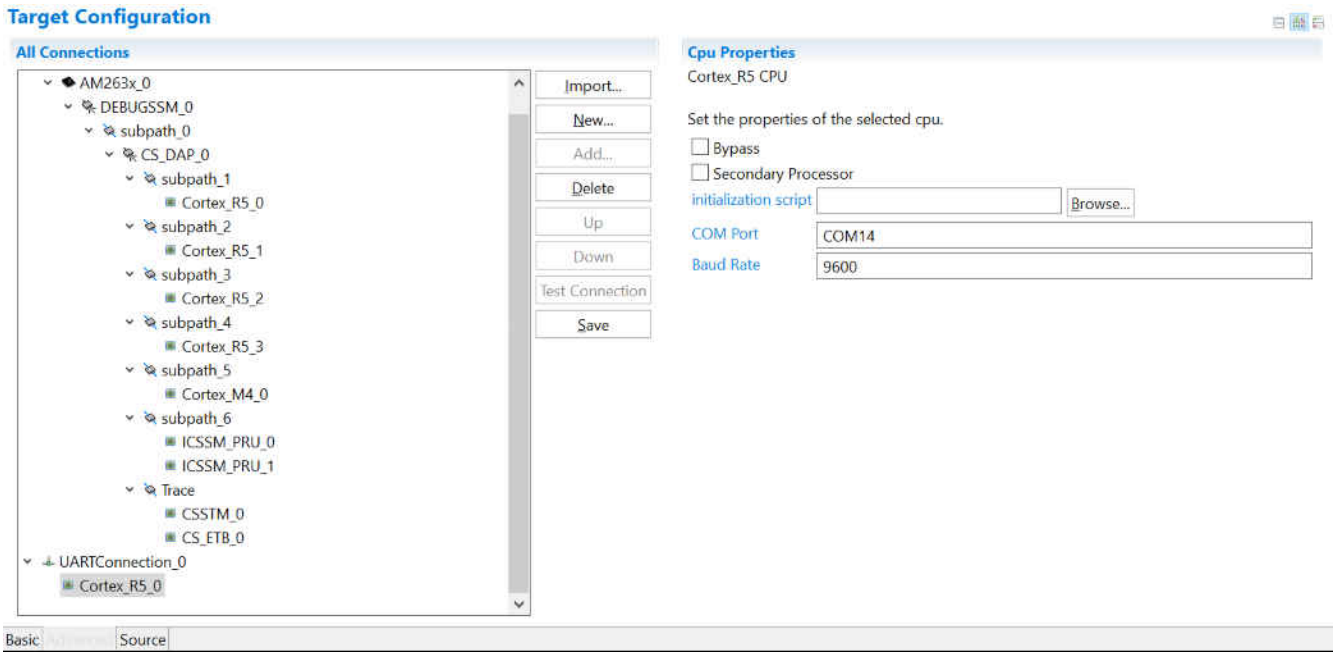

Figure 2-8. Select CPU Properties

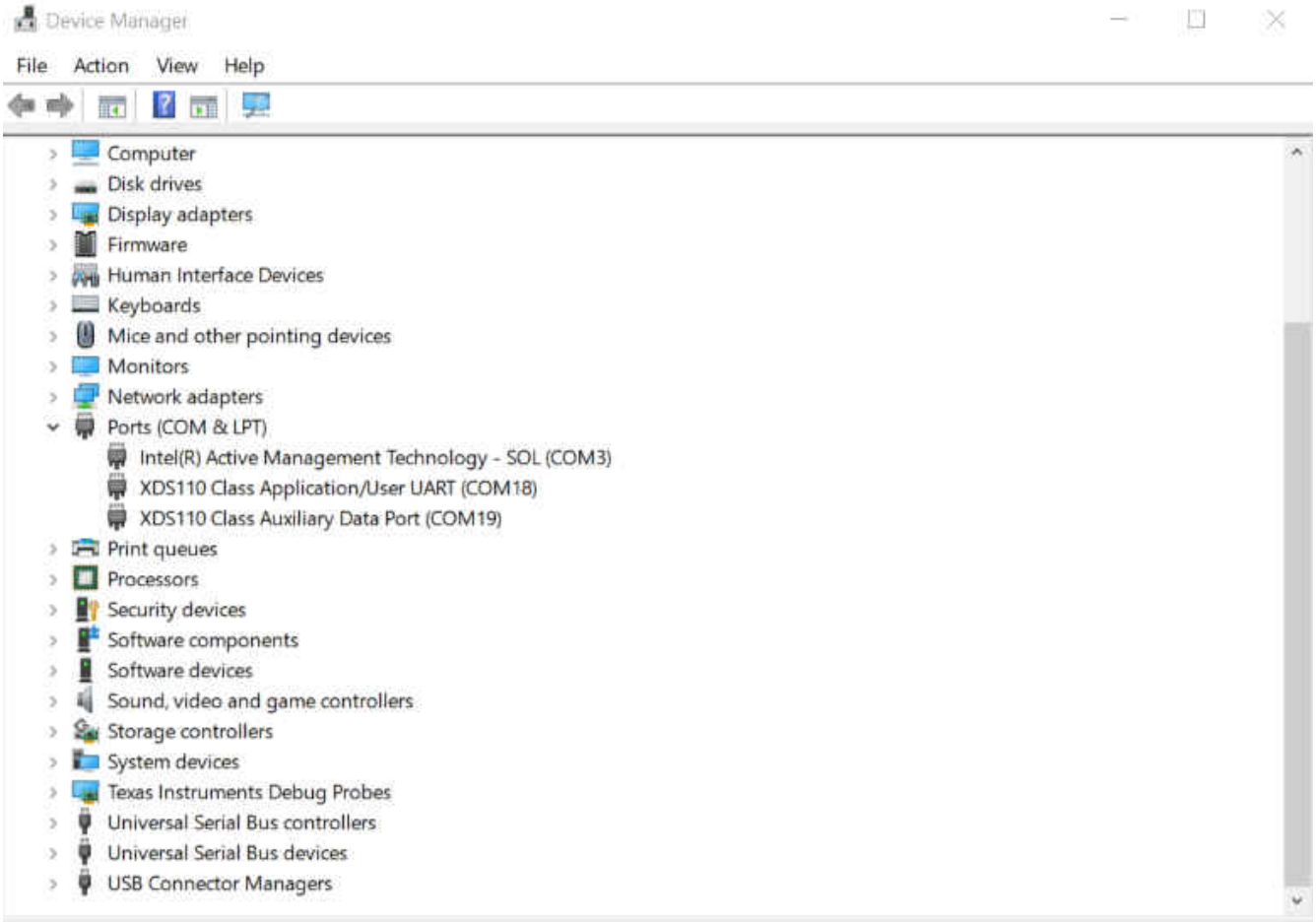

Figure 2-9. Find XDS110 UART COM Port

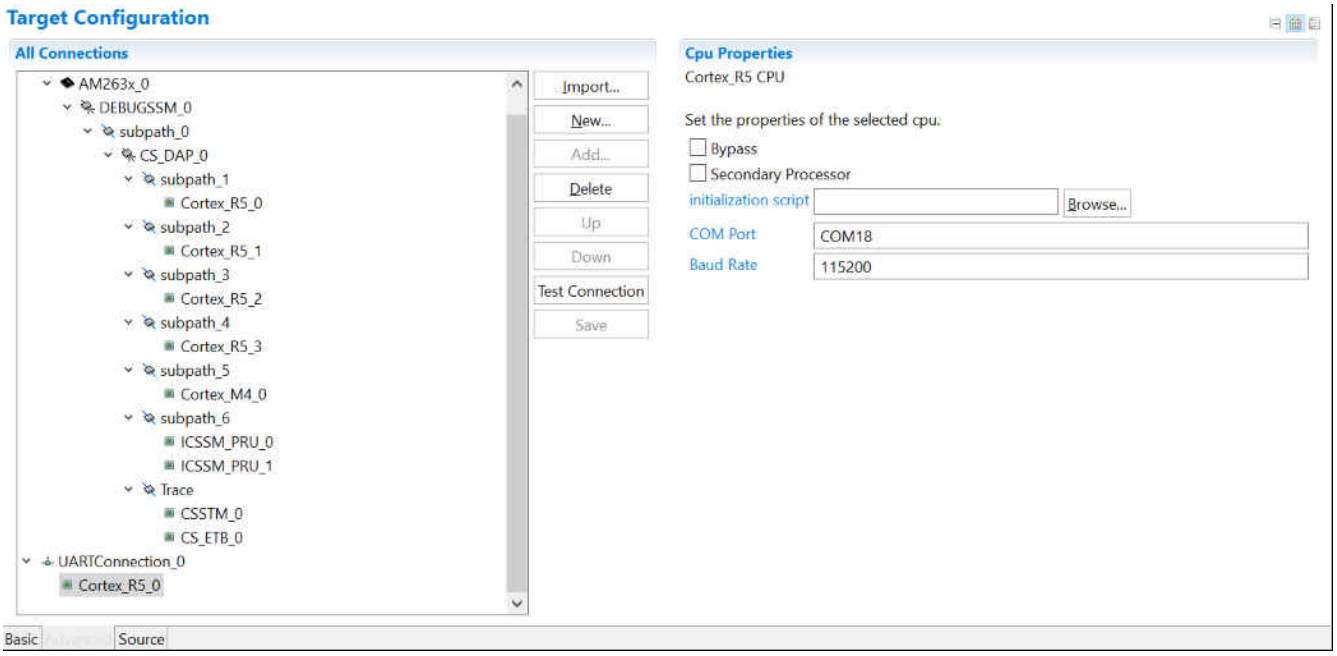

Figure 2-10. Update CPU Properties in Advanced Target Configuration

2.1.3 Add Serial Command Monitor Software

There are multiple ways to use UART0 as a debug interface. They are Debug Log and Serial Command Monitor. Debug Log is a built-in tool located at Driver Porting Layer of SDK. Like Serial Cmd Monitor, its function must be located out of interrupt callback. It is a handy tool enabling string input and output. But, input and output go through UART console only. There is no built-in GUI like Expression Window and Graph in CCS.
It is recommended to disable UART0 in Debug Log at Figure 2-11 and configure UART0 instance for Serial Command Monitor at Figure 2-12. As the name of UART in Sysconfig, "CONFIG_UART_CONSOLE", matches the handle name in "Serial_Cmd_HAL.c", it not necessary to modify the two functions required by initialization and background loop. They can be simply inserted as Figure 2-13.

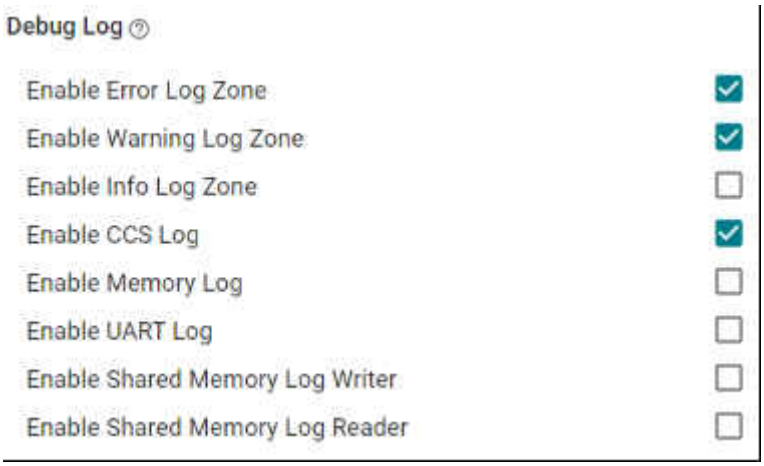

Figure 2-11. Disable UART Log in Debug Log

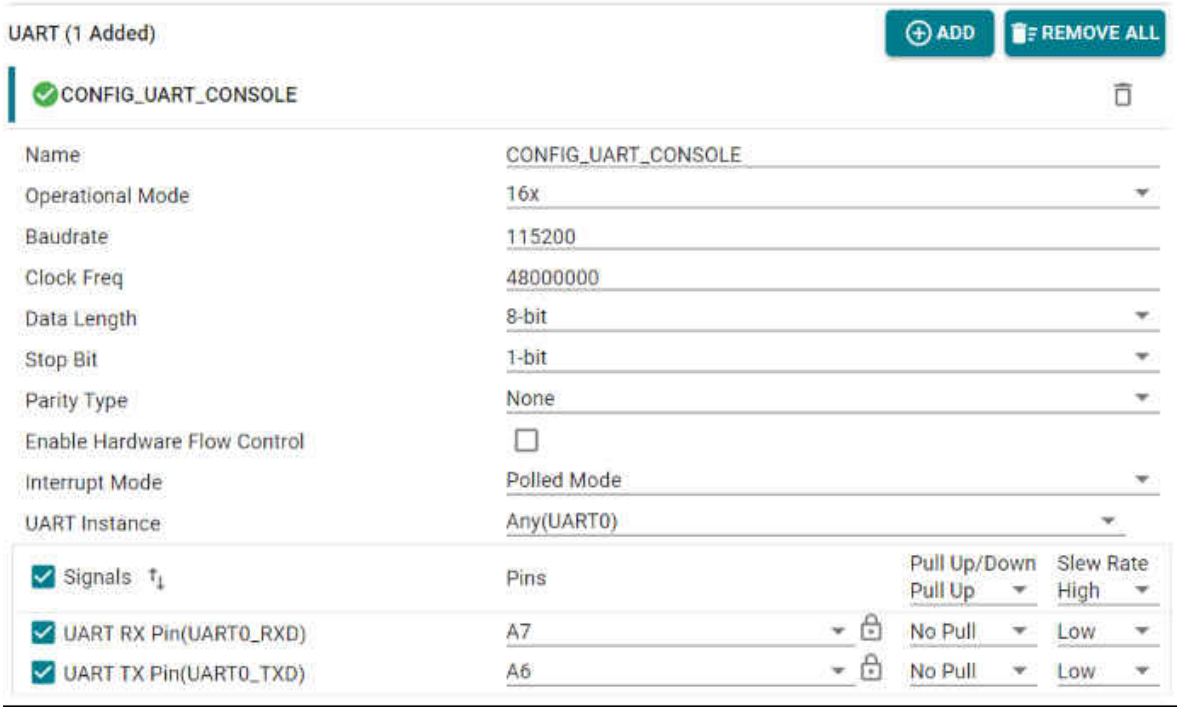

Figure 2-12. Configure UART0 Instance

```
void trinv_main(void *args)
{
    /* Open drivers to open the UART driver for console */
    Drivers_open();
    Board_driversOpen();

    SerialCmd_init();

    trinv_init();
    DebugP_log("trinv init done \r\n");
    FOC_init();
    DebugP_log("foc init done \r\n");
    FOC_cal();
    DebugP_log("foc cal done \r\n");
    FOC_run();
    DebugP_log("foc run done \r\n");

    while (gFlag){

        SerialCmd_read();

        gLoopTicker+=1;
        if (gLoopTicker>=100)
        {
            gLoopTicker=0;
        }
    }
    Board_driversClose();
    Drivers_close();
}
```

Figure 2-13. Add Serial Monitor Function Calls

2.1.4 Launch Real Time Debug

After building the program, debug window should be opened with the target configuration file created in Section 2.1.2. If the created target configuration file is not already opened, it can be located by following Figure 2-14 and looking into "User Defined" folder of the "Target Configuration" window. The created target configuration file should be under folder named as "User Defined". After right click on the file, a menu shows up and there is a option "Launch Selected Configuration" as shown in Figure 2-15. Then, debug window shows up. The steps to connect target, load image and run via JTAG can be found in many CCS tutorials. The processor must be running continuously before connecting to UART. As the UART connection is based on continuous operation of the program, UART connection will be broken and CCS will be frozen by Break-point, Suspend, Terminate or any other events stopping the Serial Command Monitor program from running. Sometimes, it is just a habit to use those features when they are available. It is recommended to disconnect target via JTAG as shown in Figure 2-16 while using UART connection. When the processor is running, UART connection can be established by simply select the UART connection → Run → Load → Load Symbols as shown in Figure 2-17.

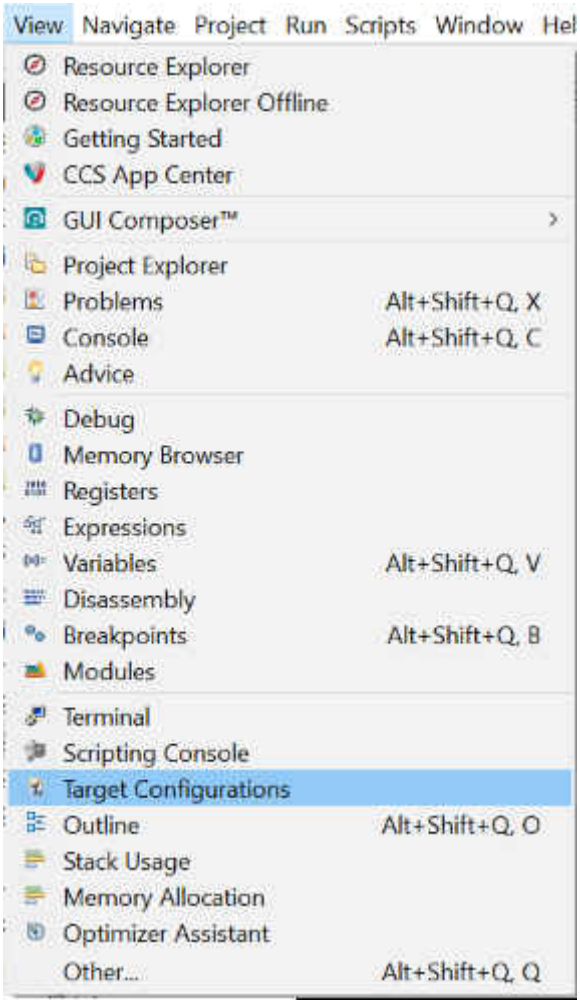

Figure 2-14. Locate Target Configuration File

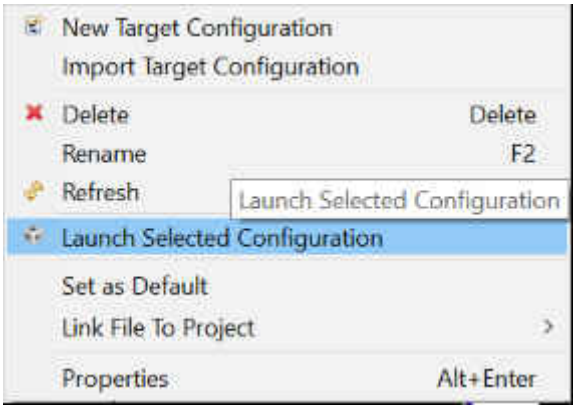

Figure 2-15. Launch Selected Configuration

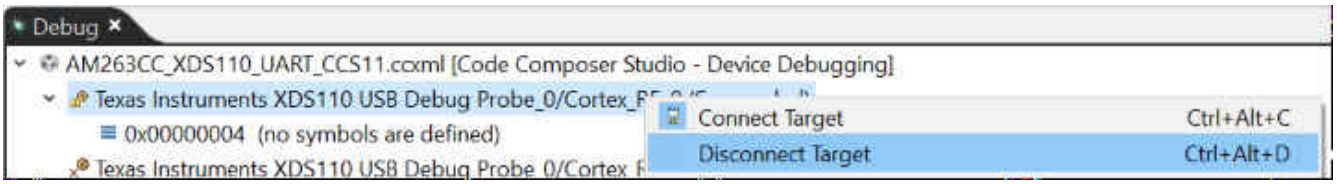

Figure 2-16. Disconnect JTAG Connection

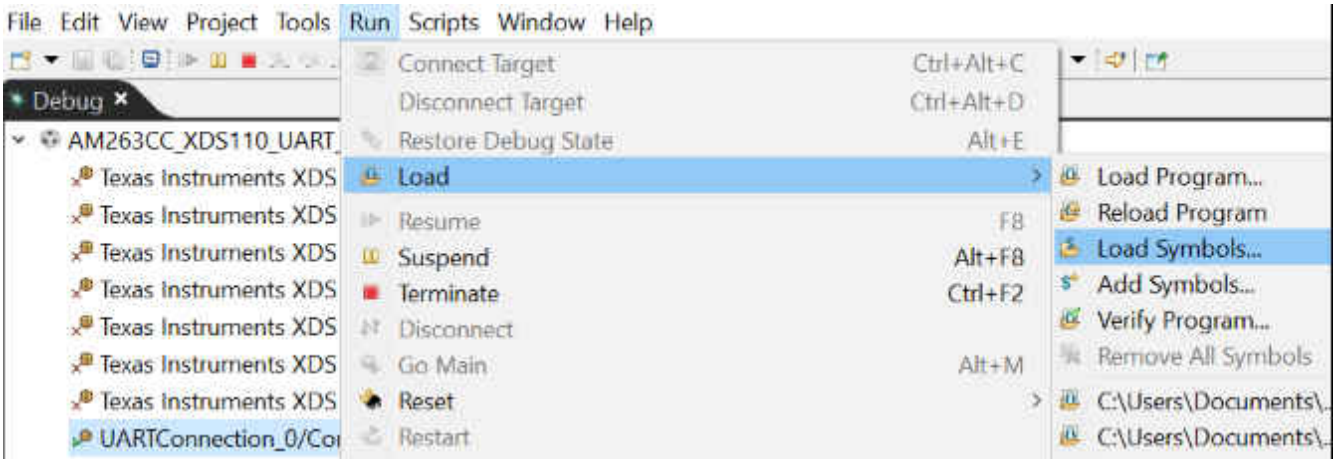

Figure 2-17. Establish UART Connection

2.2 Configure Control Peripheral and ADC Interrupt With Sysconfig

Sysconfig is a GUI-based configuration tool. It significantly simplifies the configuration of control peripherals and interrupts. The items inside Sysconfig are described in a very intuitive way. After the configuration is completed in Sysconfig, configuration program will be automatically generated and connected to main function during build. They are stored under the following files under a folder named "Generated Source". It is worth attention that the content of those files will be refreshed after a build with Sysconfig. It is fine to copy program from these files and reuse in other parts of the application program. More details on Sysconfig can be found from SDK documents. In the following parts of this section, the focus is on how to setup control peripherals and interrupt for a traction inverter.

- ti_board_config.c
- ti_board_config.h
- ti_board_open_close.c
- ti_board_open_close.h
- ti_dpl_config.c
- ti_dpl_config.h
- ti_drivers_config.c

- ti_drivers_config.h
- ti_drivers_open_close.c
- ti_drivers_open_close.h
- ti_pinmux_config.c
- ti_power_clock_config.c

2.2.1 Generate PWM for Time Reference

The PWM module of AM263x inherited features from TI classic C28 controllers. With a unique XBAR architecture of AM263x, its ADC SOC triggers are able to trigger not only ADC events but also various of other events like DMA. This part will give some details on how to synchronize PWM modules and how to configure PWM modules to trigger ADC and DMA. More details can be found in Technical Reference Manual.

Figure 2-18 presents a summary of ePWM7, which is used to trigger DMA for resolver excitation signal updating at 200 kHz via DAC. There are couple tabs available for configuration. But, for the DMA trigger purpose, there are only three tabs to be updated.

- EPWM Time Base
- EPWM Action Qualifier
- EPWM Event-Trigger

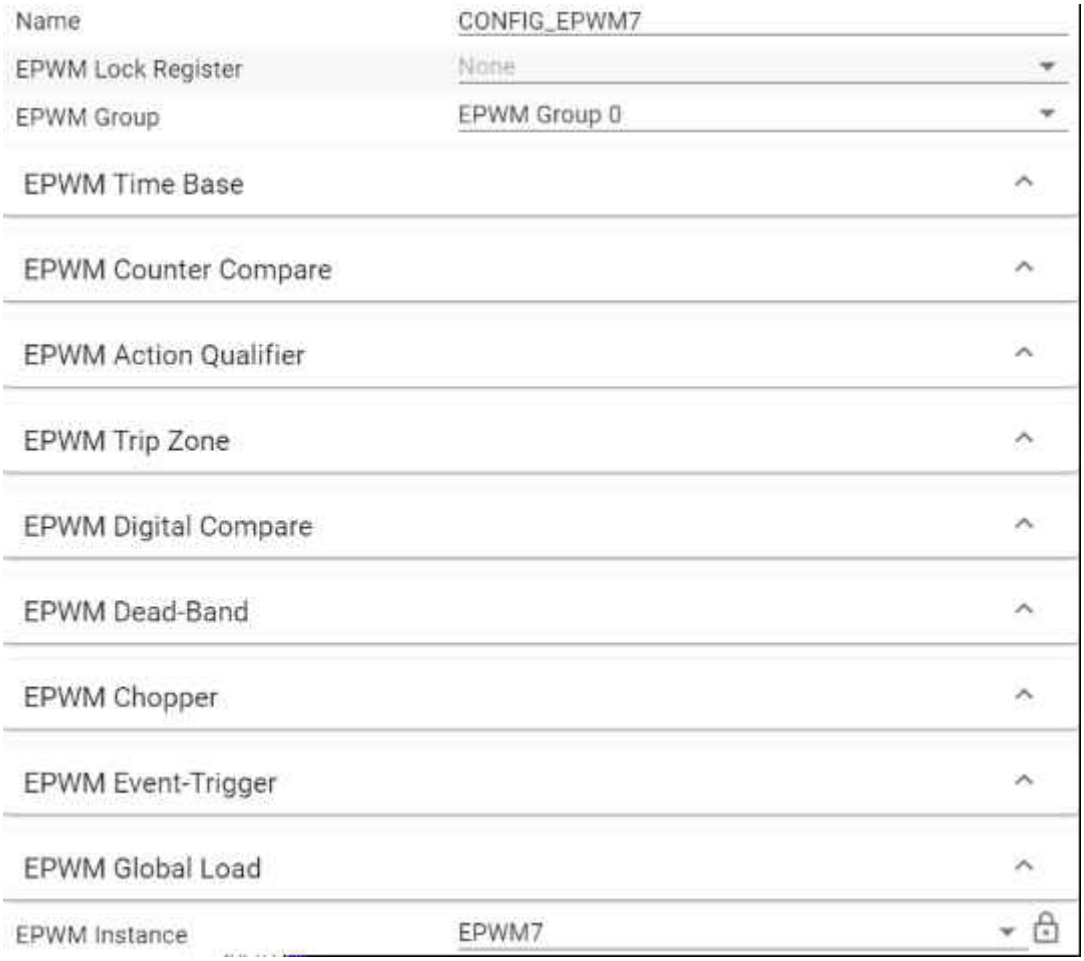

Figure 2-18. A Summary of EPWM7 Configuration

Figure 2-19 shows details on Time Base. EPWM7 is configured as 200 kHz, Up-Count, and following Sync Out Pulse from ePWM0. It is worth attention that peripherals are operated at 200 MHz while R5F cores are running at 400 MHz.

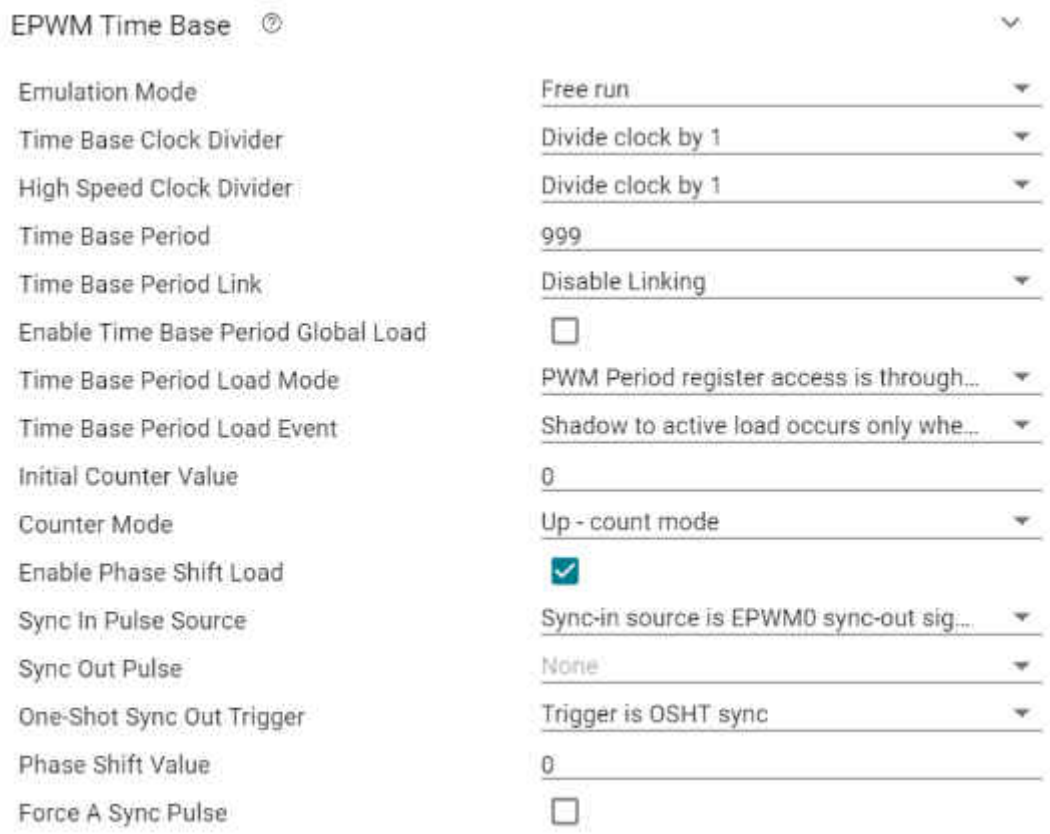

Figure 2-19. EPWM7 Time Base Configuration

Figure 2-20 shows details on Event Trigger. The key contents are listed below. ePWM7 is to generate two triggers, ADC SOCA and SOCB. The two triggers are from different source events The one related to CMPA is configured in the Counter Compare tab as shown in Figure 2-21.

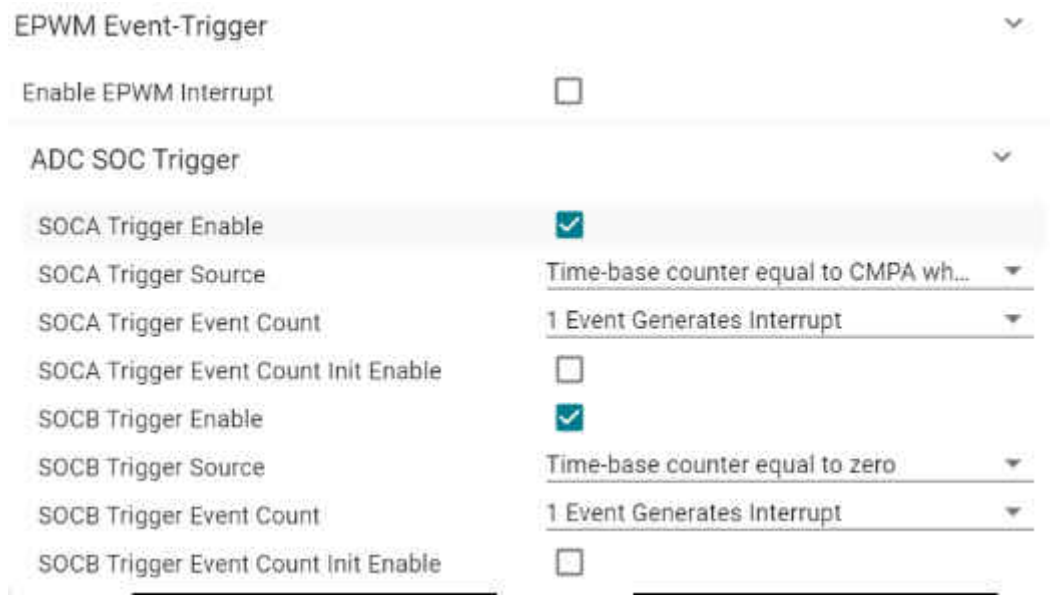

Figure 2-20. EPWM7 Event Trigger Configuration

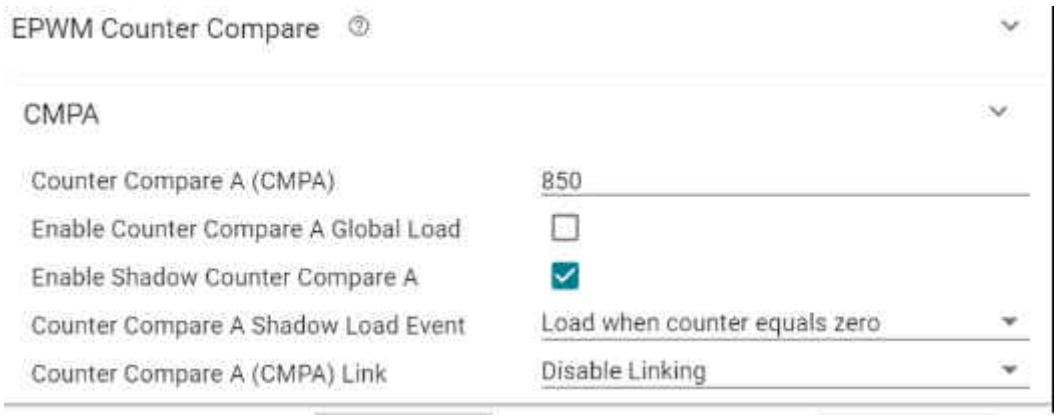

Figure 2-21. EPWM7 Counter Compare Configuration

EPWM0 is used as Phase A of traction inverter. All EPWM channels are synchronized to EPWM0. It is configured as 10 kHz, Up-Down Count, No Sync In, Sync Out and ADC SOC trigger at counter zero as shown by Figure 2-22 and Figure 2-23.

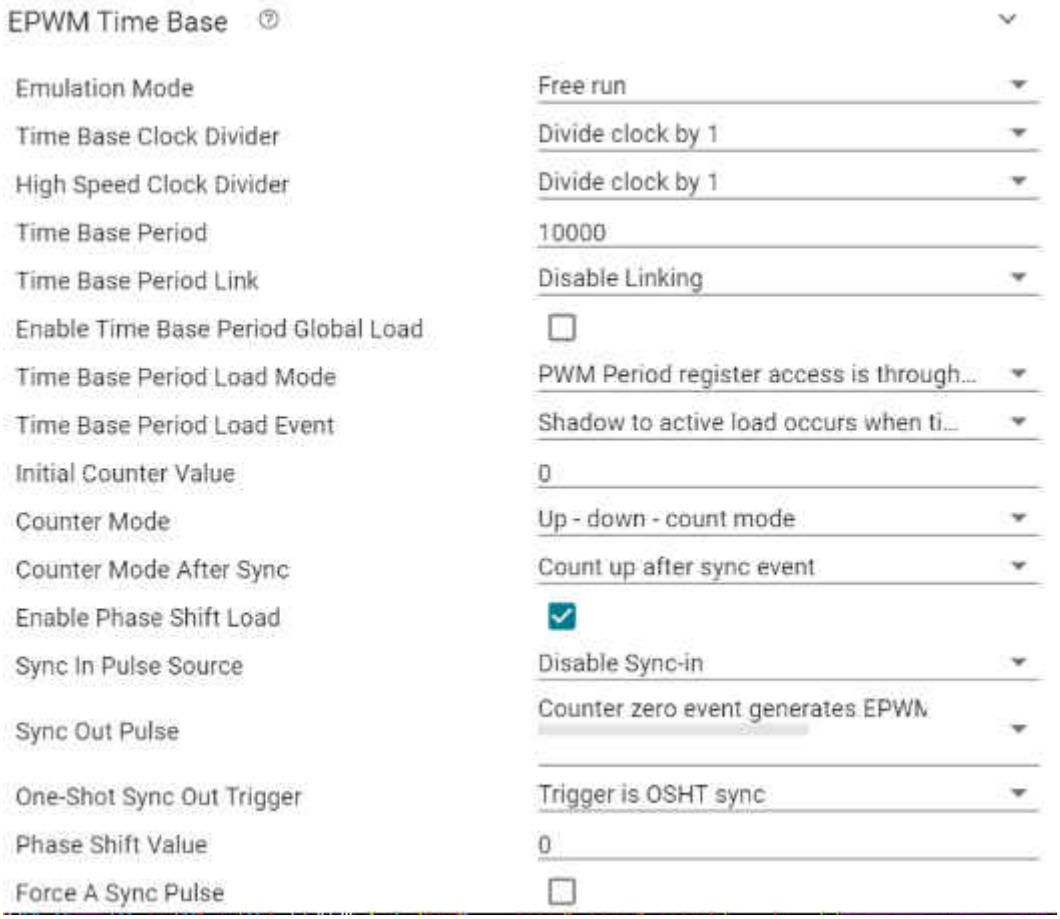

Figure 2-22. EPWM0 Time Base Configuration

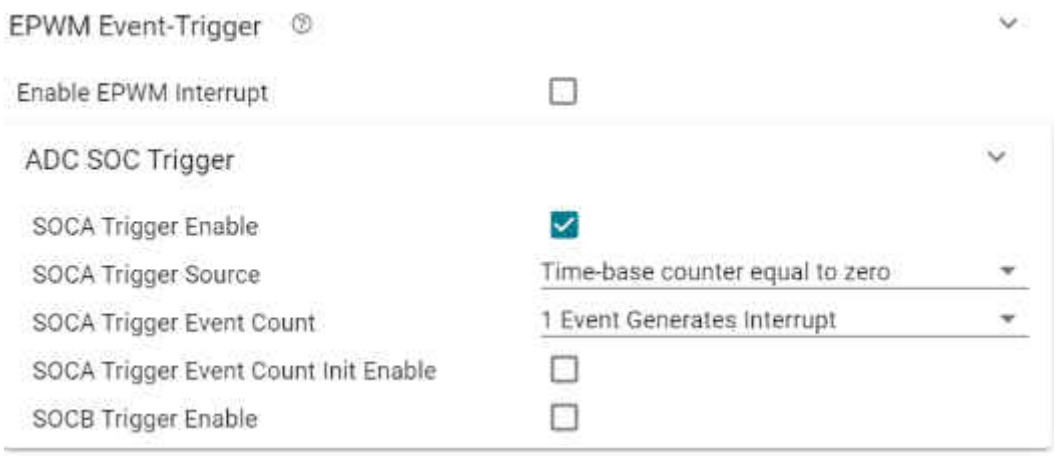

Figure 2-23. EPWM0 Event Trigger Configuration

Dead-Band of EPWM0 is configured as Figure 2-24. Delay value of 200 indicates 1000 ns because of the 200 MHz peripherals clock.

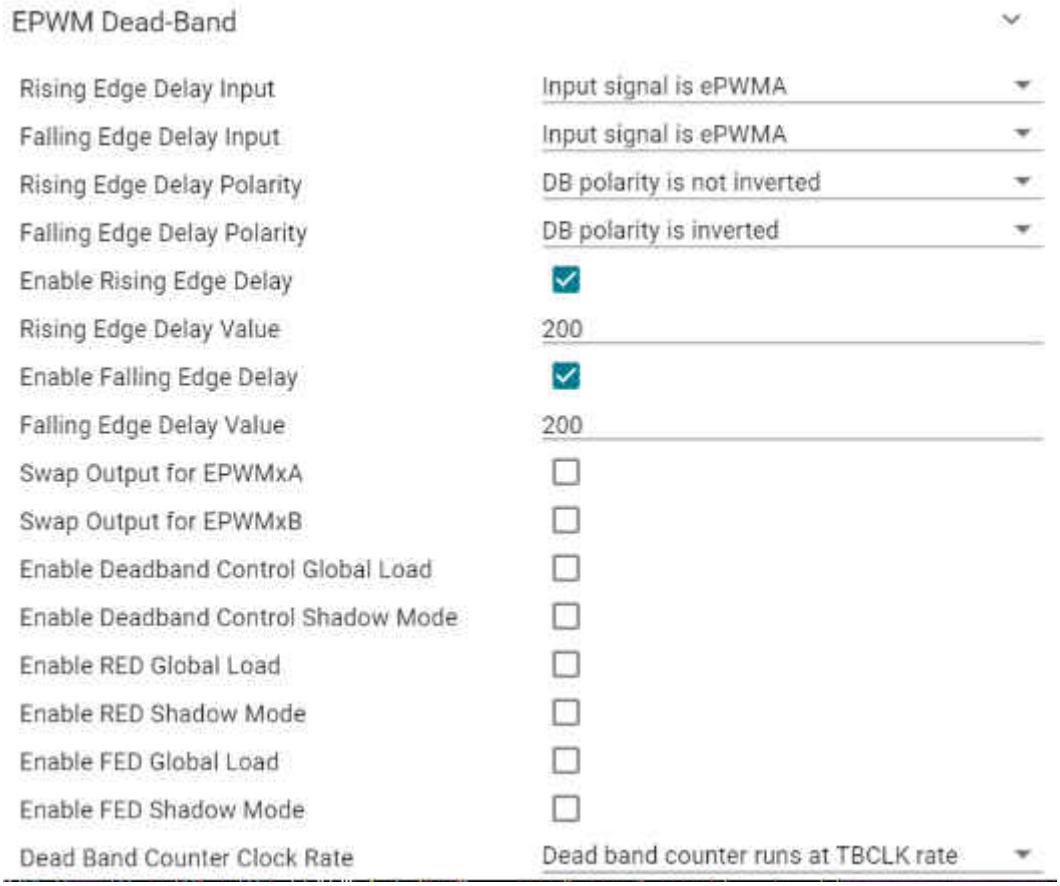

Figure 2-24. EPWM0 Dead-Band Configuration

2.2.2 Synchronize ADC Sampling and Interrupt Service Routine

ADC4 is taken as a example in this part. Figure 2-25 gives a summary of ADC4. It is similar to TI classic C28 controllers.

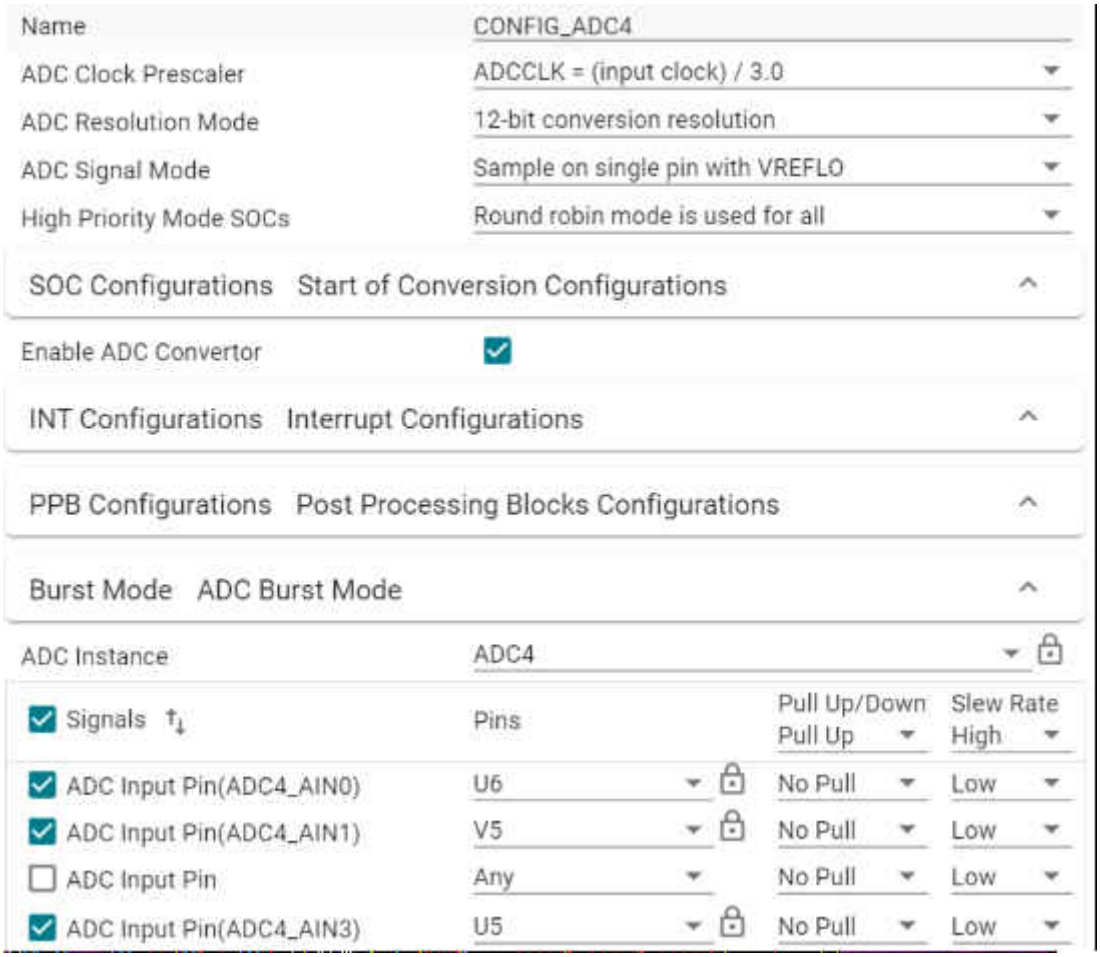

Figure 2-25. A Summary of ADC4 Configuration

Figure 2-26 gives an example on SOC0. There are 16 SOC for every ADC. The ADC channel can be configured as either single end or differential. Here, it is triggered by a PWM SOC event.

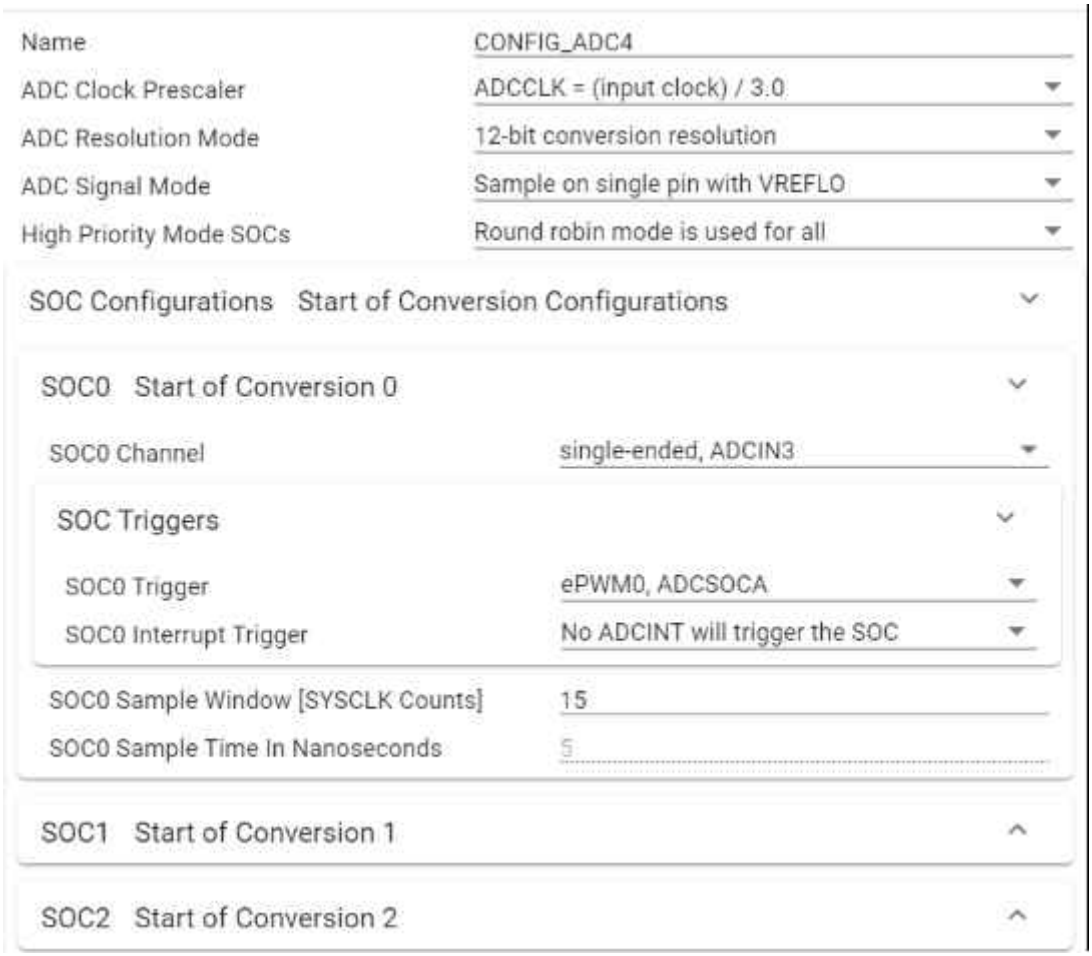

Figure 2-26. ADC4 SOC Configuration

Once the Start Of Conversions are configured, INTs can be updated like shown in Figure 2-27. The INT is to be created at the End Of Conversion for SOC5.

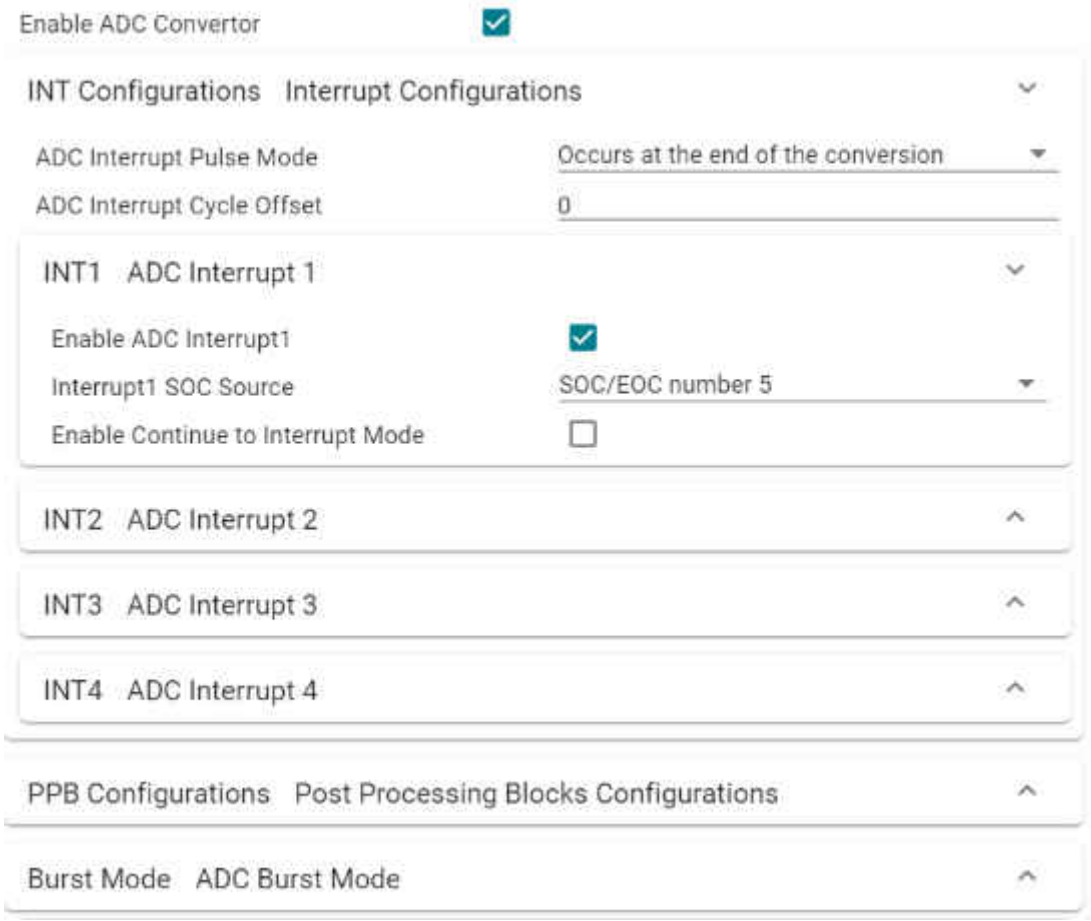

Figure 2-27. ADC4 INT Configuration

In Figure 2-28, ADC INT is connected to INT_XBAR_0 via XBAR system. The XBAR name is the one used in interrupt registration in later sections.

Figure 2-28. INT XBAR Configuration

2.2.3 Configure DMA for Resolver Excitation via DAC

In configuration, most of EDMA module shown in Figure 2-29 will be updated by user defined function during initialization of application. So, most contents can stay as the default.

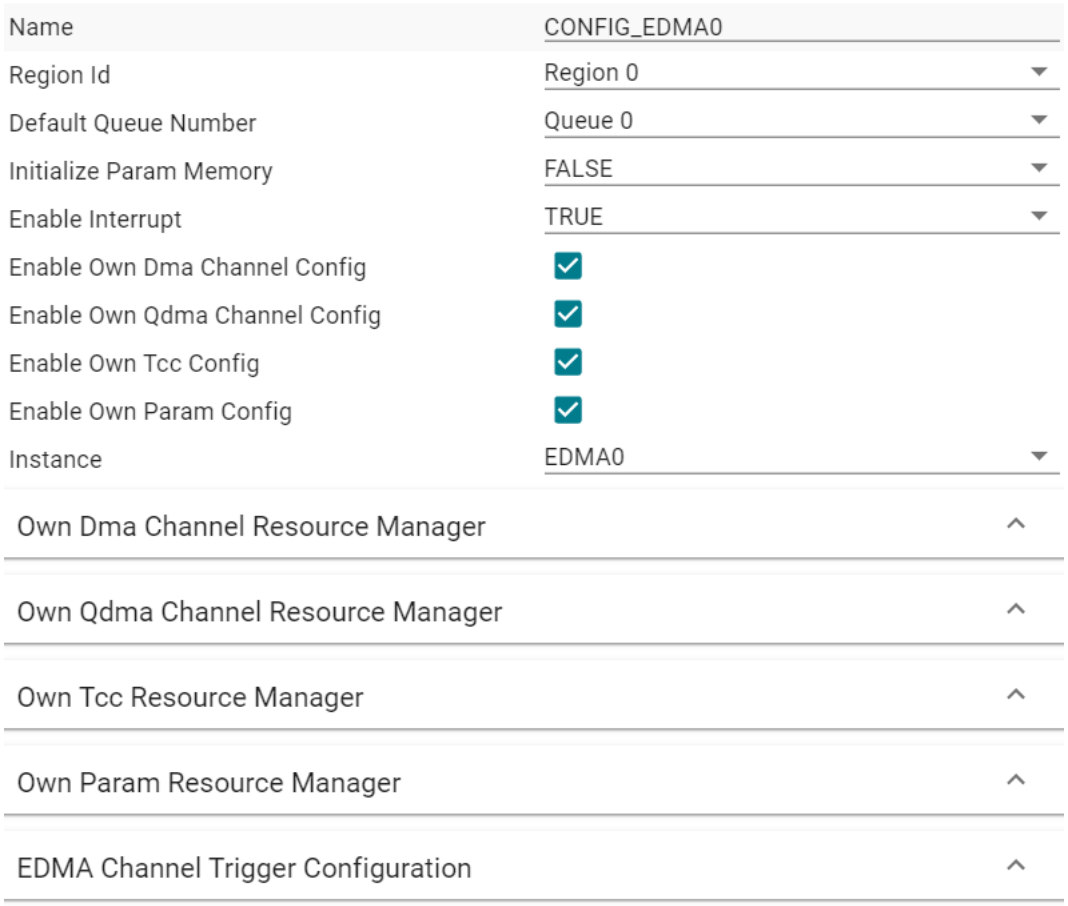

Figure 2-29. A Summary of EDMA Configuration

The key is the EDMA Channel Trigger Configuration in Figure 2-30. All trigger sources are listed in the Channel Trigger menu. As shown earlier, EPWM7 is configured as the trigger for EDMA to update DAC generating resolver excitation sine wave.

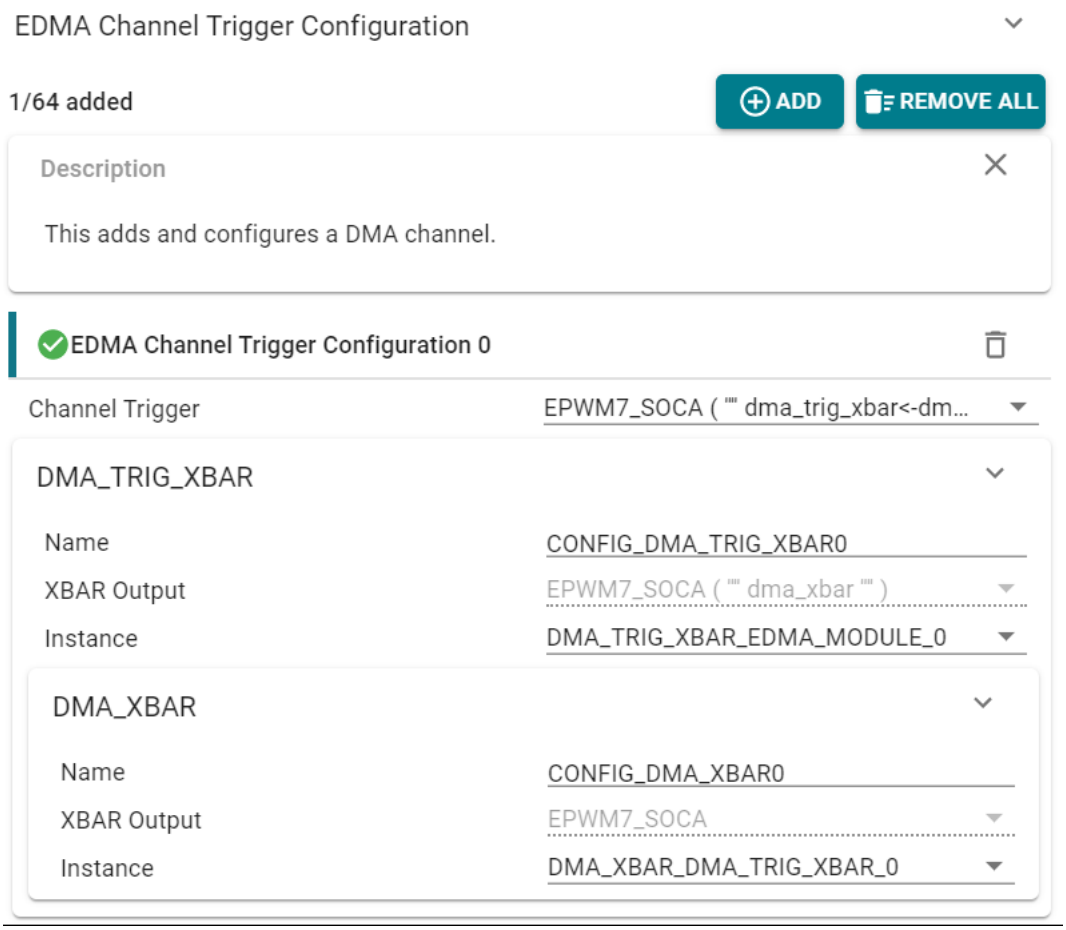

Figure 2-30. EDMA Channel Trigger Configuration

2.3 Configure Gate Driver Interface With MSPI

In TIDM-02009, UC5870-Q1 is designed to drive SiC MOSFETs. Besides PWM gating signal, it requires one SPI interface and couple GPIOs for communication and configuration. In this section, some pinmux on control card needs to be checked and modified if needed. Then, SPI is configured and the gate driver is initialized.

2.3.1 Confirm Control Card Hardware Configuration for Gate Drivers

As shown in Figure 2-31, there are 4 resistor arrays footprint and 2 resistor arrays used as pinmux between board edge interface, HSEC, and on-board SD card holder, MMC0. From TIDM-02009 and its design files, it is found that the board edge pins are required to connect some gate driver pins. The pinux configuration should be confirmed before moving forward.

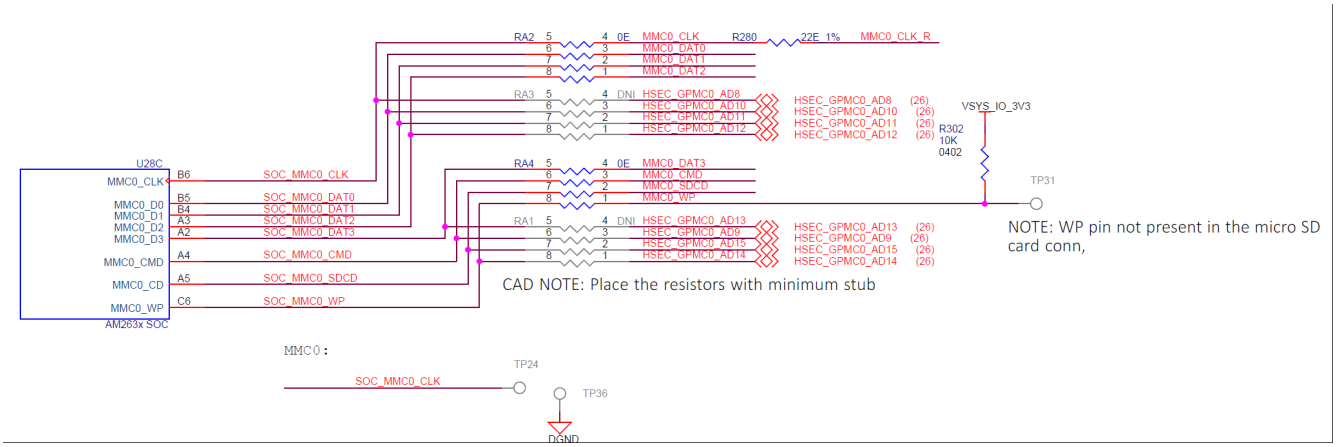

Figure 2-31. Control Card Pinmux

2.3.2 Configure MCSPI for UCC5870 Gate Drivers

MCSPI is configured in both Sysconfig and initialization program for UCC5870. The Sysconfig items are summarized in Figure 2-32. They cover most features except for data frame size. It is configured in the initialization program listed here. For UC5870-Q1, 16-bit data frame is used according to its data sheet.

- Init_UCC5870();
  - MCSPI_setDataWidth(spibaseAddr, spichNum, 16);

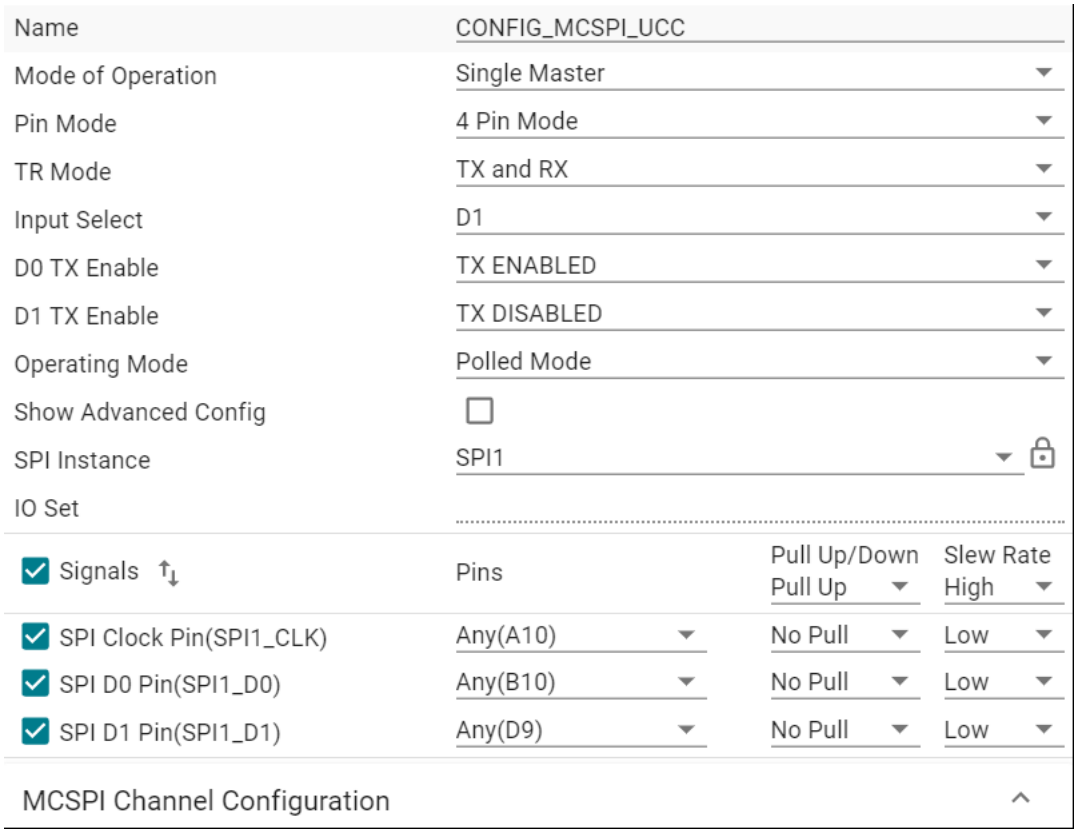

Figure 2-32. A Summary of MCSPI Configuration

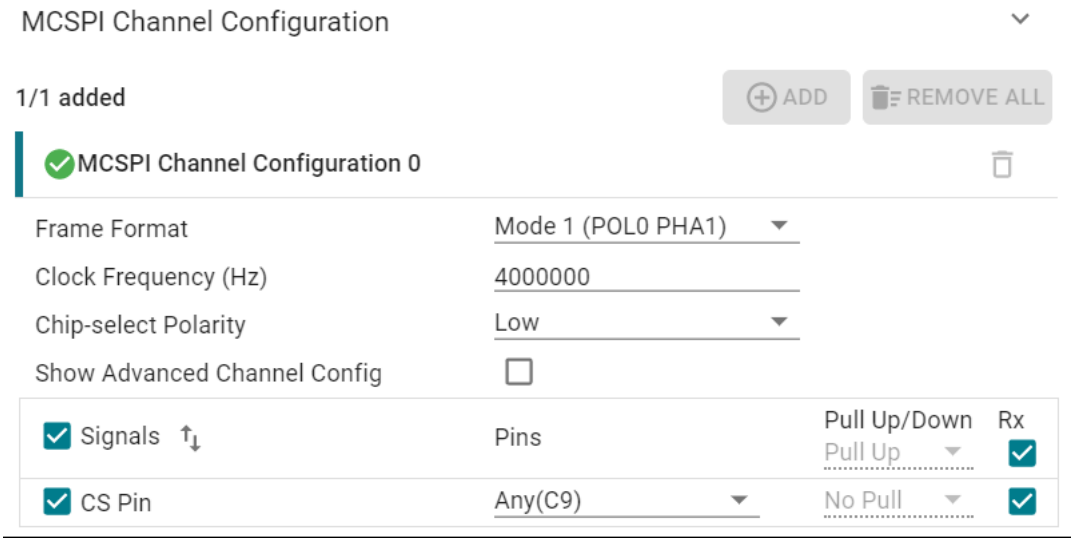

Figure 2-33. MCSPI Channel Configuration

2.3.3 Initialize UCC5870 Gate Drivers

UC5870-Q1 data sheet is the best resource to learn about device programming. TIDM-02009 adopts address-based configuration. The implementation of SPI communication follows the table of SPI message commands in UC5870-Q1. The initialization process follows the section of Device Functional Modes in UC5870-Q1 data sheet. At system level, there are two functions called by traction inverter initialization to initialize register values before writing and initialize gate drivers with the values. They are listed as follows.

- Init_UCC5870_Regs();
- Init_UCC5870();

2.4 Get Samples From ADC and Read Samples Via CCS

As configured in previous sections, ADCs sample signals at count zero of PWM0, and ADC INT is created at the selected End of Conversion every period of PWM0. This section presents how to register the INT, read samples, and plot them in Graph.

2.4.1 Register and Enable Interrupt

After the INT configuration, the following lines are necessary to register and enable interrupt. Line 1 is to initiate the hardware interrupt parameters with the defaults. Line 2 is to update the interrupt number with the one configured in previous section. More details on definition of the macro can be found in "cslr_intr_r5fss0_core0.h". If other clusters or cores are used, similar files can be located in the SDK. Line 3 assigns a callback function to the hardware interrupt. It only takes function addresses. Line 4 construct the hardware interrupt with the updated hardware interrupt parameters. If there is something wrong, line 5 will send fault message into CCS console via JTAG. If the interrupt status is not cleared, it will start to run after line 6 clear the status. In order to keep the interrupt running continuously, line 6 must be called every execution of the callback function.

1. HwiP_Params_init(&hwiPrms);
2. hwiPrms.intNum = CSLR_R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_0;
3. hwiPrms.callback = &FOCrun_ISR;
4. status = HwiP_construct(&gAdcHwiObject, &hwiPrms);
5. DebugP_assert(SystemP_SUCCESS == status);
6. ADC_clearInterruptStatus(CONFIG_ADC4_BASE_ADDR,ADC_INT_NUMBER1);

2.4.2 Add Log Code to Read Samples in Graph at Fixed Rate

As the time spent on communication between SoC and CCS is not deterministic, it is necessary to have a log recording values from the interrupt at a given sample rate. This is critical for observation data via graph window. A simple log is implemented. There are 16 pointers available to 16 global variables. Before the hardware interrupt starts, the pointers need to be assigned to either global variables or NULL. The following lines need to be called. Line 1 is called after the assignment of pointers. Line 2 is called after all computation is completed every interrupt. There is a scale, named gLogScaler, implemented in the line 2 function. It represents how many interrupts are skipped between two logging points. By setting the global variable, data can be logged at frequencies lower than or equal to the interrupt frequency.

1. LoopLog_init();
2. LoopLog_run();

2.4.3 Read ADC Samples in Expression and Graph Windows

ADC samples are read by the following lines 1 to 8 for 3-phase current, resolver sin/cos, and DC bus voltage. The SDK API is wrapped into Macros in one file. Couple simple "Ctrl + left click" on the variable name will help find the location where is defined. Example are given in lines 9 and 10. ADC_readResult is to read ADC result and ADC_readPPBResult is to read ADC result after Post Processing Blocks. Details on Post Processing Blocks can be find in Technical Reference Manual.

1. motor1.I_abc_A[0] = (float32_t)IFBU_PPB;
2. motor1.I_abc_A[1] = (float32_t)IFBV_PPB;
3. motor1.I_abc_A[2] = (float32_t)IFBW_PPB;
4. resolver1.sin_samples[0] = (float32_t)R_SIN1;
5. resolver1.sin_samples[1] = (float32_t)R_SIN2;
6. resolver1.cos_samples[0] = (float32_t)R_COS1;
7. resolver1.cos_samples[1] = (float32_t)R_COS2;
8. motor1.dcBus_V = (float32_t)VDC_EVT;
9. ADC_readResult(CSL_CONTROLSS_ADC1_RESULT_U_BASE, ADC_SOC_NUMBER0)
10. ADC_readPPBResult(CSL_CONTROLSS_ADC1_RESULT_U_BASE, ADC_PPB_NUMBER1)

To show an example on reading and plotting ADC in graph window, the log pointers are connected to 3-phase currents and log functions are called. Phase A current at no load is plotted into graph window by right clicking the gLog_CH[7] in expression window and selecting graph as shown in Figure 2-34. In this case, Phase A current is pointed to gLog_CH[7] as shown in the following list. It could be assigned to any log channel. To add gLog_CH into expression window, it is simply right clicking and adding to watch expression. More details can be found in CCS tutorial.

1. gLog_ptr[7] = &motor1.I_abc_A[0];
2. gLog_ptr[8] = &motor1.I_abc_A[1];
3. gLog_ptr[9] = &motor1.I_abc_A[2];

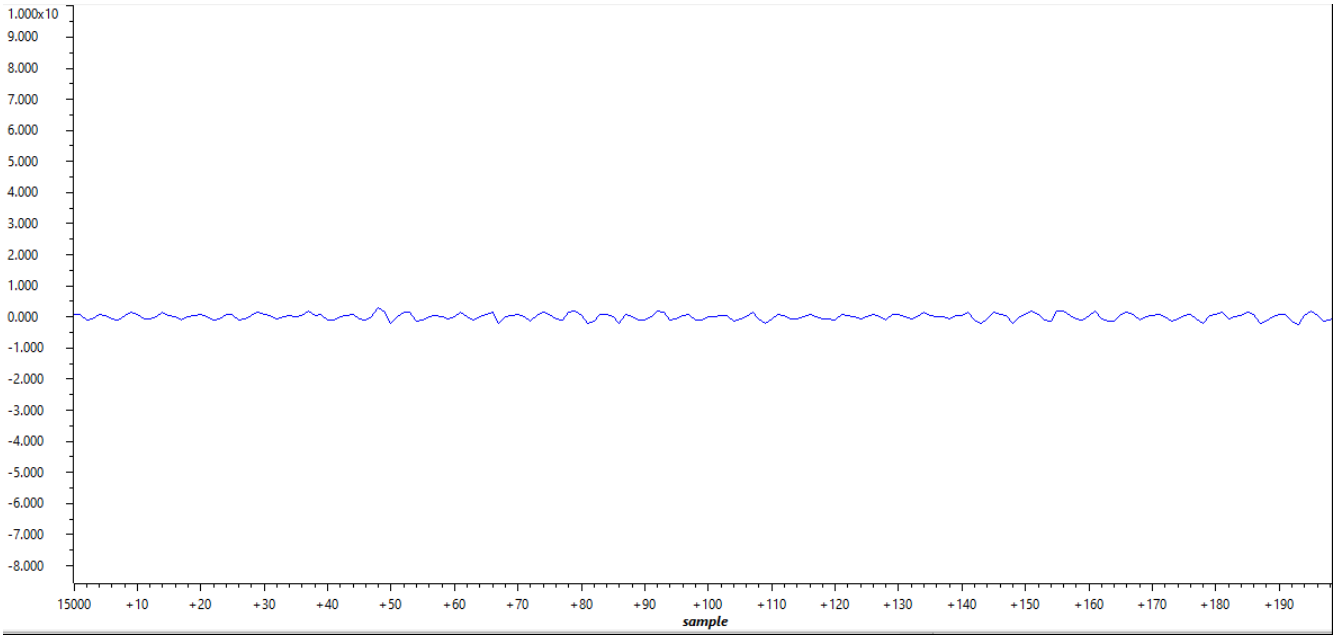

Figure 2-34. Plotted Phase A Current at No Load

During bring up at low voltage like 12V, it is suggested to overwrite the DC bus voltage with voltage reading from power supply as error from TIDM-02009 is not negligible at 12V.

2.5 Generate Space Vector PWM and Drive Motor in Open Loop

Space Vector PWM is a classic method for motor control. This section is to show high level functions other than text book details. The high level functions can be easily replaced by similar functions from either C28 libraries or other controller libraries. The key is to reduce the usage of local variables and allocate global variables into TCM for deterministic execution time.

2.5.1 Setup SVPWM Generator Inputs

The inputs to SVPWM generator is Vd and Vq. The following lines need to be called for assignment of the values. Motor1 is a structure stored in TCM. More details on its definition can be found in the program files. Couple simple "Ctrl + left click" on the variable name will help trace the location where is defined. The code shares the same logic as the C28 program for TIDM-02009. Vd and Vq are in real value other than per unit value.

1. motor1.Vout_dq_V[0] = VdTesting;
2. motor1.Vout_dq_V[1] = VqTesting;

Motor speed and motor angle are generated by the following lines. Lines 1 to 4 setup ramp controller, rc1, and ramp generator, rg1. SpdRef is per unit value between 0 and 1. The generated omega and theta are assigned to motor1 in lines 5 and 6. Line 7 limits the theta to a range from 0 to TWO_PI. The TWO_PI value is defined in the files. Couple simple "Ctrl + left click" on the variable name will help trace the location where is defined. It is worth attention that rc1, rg1, and motor1 need to be initialized accordingly before starting hardware interrupt.

1. rc1.TargetValue = SpdRef;
2. rampControl(&rc1);
3. rg1.Freq = rc1.SetpointValue;
4. rampGen(&rg1);
5. motor1.omega_e = rg1.Freq * BASE_FREQ * TWO_PI;
6. motor1.theta_e = rg1.Out * TWO_PI;
7. theta_limiter(&(motor1.theta_e));

The next couple lines are to feed the inputs into SVPWM generator and keep the output in per unit values. Line 1 keeps inputs within limits. Line 2 is inverse park transformation. Similar functions can be found in CMSIS DSP library and others. The angle information is already included in the structure of motor1. Line 3 is SVPWM generator. The logic is the same as C28 program for TIDM-02009. There are other implementations in previous C28 libraries. It is worth attention that there is a real value to per unit value conversion in this version. Line 4 keeps per unit output within limits.

1. dq_limiter_run(&motor1);
2. ipark_run(&motor1);
3. SVGEN_run(&motor1, &pwm1);
4. PWM_clamp(&pwm1);

After SVPWM generated, the per unit outputs are passed to EPWM Counter Compare by the function in following line 1. Line 2 gives details on setting EPWM0 Counter Compare. EPWM_setCounterCompareValue is the name of SDK API to set Counter Compare value. Here, the value is computed for Up-Down mode or Center-line mode.

1. TRINV_HAL_setPwmOutput(&pwm1);
2. EPWM_setCounterCompareValue(CONFIG_EPWM0_BASE_ADDR, EPWM_COUNTER_COMPARE_A, (uint16_t)((pwm->inv_half_prd * pwm->Vabc_pu[0]) +pwm->inv_half_prd));

2.5.2 Read SVPWM Duty Cycles in Graph Window

Similar to plotting ADC samples in graph window in Section 2.4.3, the following outputs in per unit value need to be connected with the log pointers.

1. gLog_ptr[4] = &pwm1.Vabc_pu[0];
2. gLog_ptr[5] = &pwm1.Vabc_pu[1];
3. gLog_ptr[6] = &pwm1.Vabc_pu[2];

The plotted graph of Phase A duty cycle is presented in Figure 2-35.

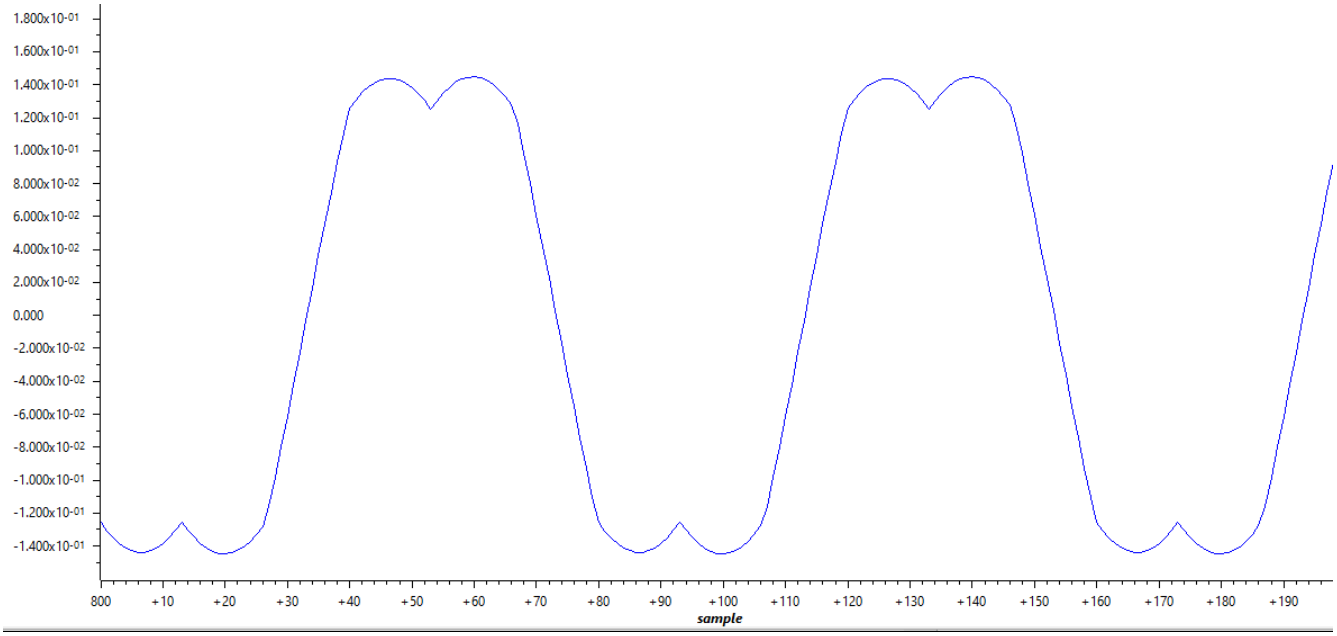

Figure 2-35. Phase A Duty Cycle

2.5.3 Power Up Inverter and Spin Motor in Open Loop

When the inverter is powered up, it is ready to spin a motor in open loop. It is recommended to start with low speed. The global variable SpdRef in per unit can be set around 0.01. There are couple flags controlling the execution of example program as listed. "runMotor" is the gate for "gTFlag_MockTheta" and "gTFlag_SpdDemo". Speed command will not be sent if "runMotor" is "FALSE". "gTFlag_MockTheta" is to use mock angle and speed for open loop and closed current loop. "gTFlag_SpdDemo" is to give speed command for demonstration of closed speed loop. "gTFlag_MockTheta" and "gTFlag_SpdDemo" should not be "TRUE" at the same time. The motor must be stopped with "runMotor" before switching between "gTFlag_MockTheta" and "gTFlag_SpdDemo". When "gTFlag_MockVdq" is "TRUE", Vd and Vq from program prior to SVPWM generation will be overwritten by manual inputs from expression window. When "gTFlag_MockId" or "gTFlag_MockIq" is "TRUE", current values at the input of current loop controller will be replaced by manual inputs from expression window.

- runMotor
- gTFlag_MockTheta
- gTFlag_MockVdq
- gTFlag_MockId
- gTFlag_MockIq
- gTFlag_SpdDemo

In this part, after setting "gTFlag_MockTheta" and "gTFlag_MockVdq" to "TRUE", "runMotor" can be changed to "TRUE". With properly selected "SpdRef", "VdTesting", and "VqTesting". It is worth attention that Vd and Vq are real value other than per unit value here. Phase A current can be read like Section 2.4.3. It is plotted in Figure 2-36. With some low frequency AC current, the motor should start spinning. If not, it is recommended to check motor, inverter, and control card. Inverter hardware details can be found at TIDM-02009. Control card details should be located in user guide.

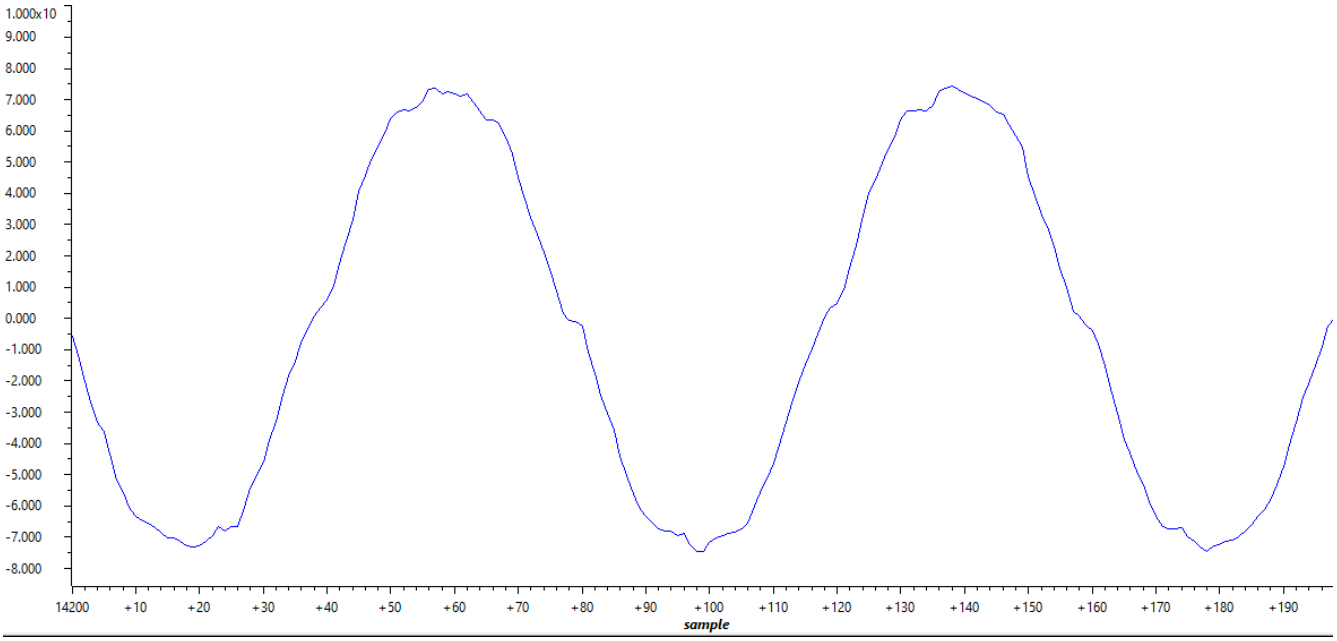

Figure 2-36. Phase A Current Open Loop

2.6 Close Current Loop With Mock Speed

This section is to close current loop with mock speed. Mock speed is created by the ramp presented in Section 2.5.1. Id and Iq references are manually assigned.

2.6.1 Add Transformations and Read Id-Iq in Open Loop

To close current loop, the following transformations need to be added. Line 1 is clark transformation, and line 2 is park transformation. Similar functions can be found in CMSIS DSP library and others. The angle information is already included in the structure of motor1. The implementation here is similar to C28 program for TIDM-02009.

1. clarke_run(&motor1);
2. park_run(&motor1);

Ideally, Id and Iq are close to constant values during open loop steady state operation. In most cases, it is not difficult to read them in expression window. If graph view is desired, they can be added to graph window by following Section 2.4.3. Here are the setup for log pointers. Line 1 is Id and line 2 is Iq. The plotted Id is in and Iq is in Figure 2-37 and Figure 2-38.

1. gLog_ptr[10] = &motor1.I_dq_A[0];
2. gLog_ptr[11] = &motor1.I_dq_A[1];

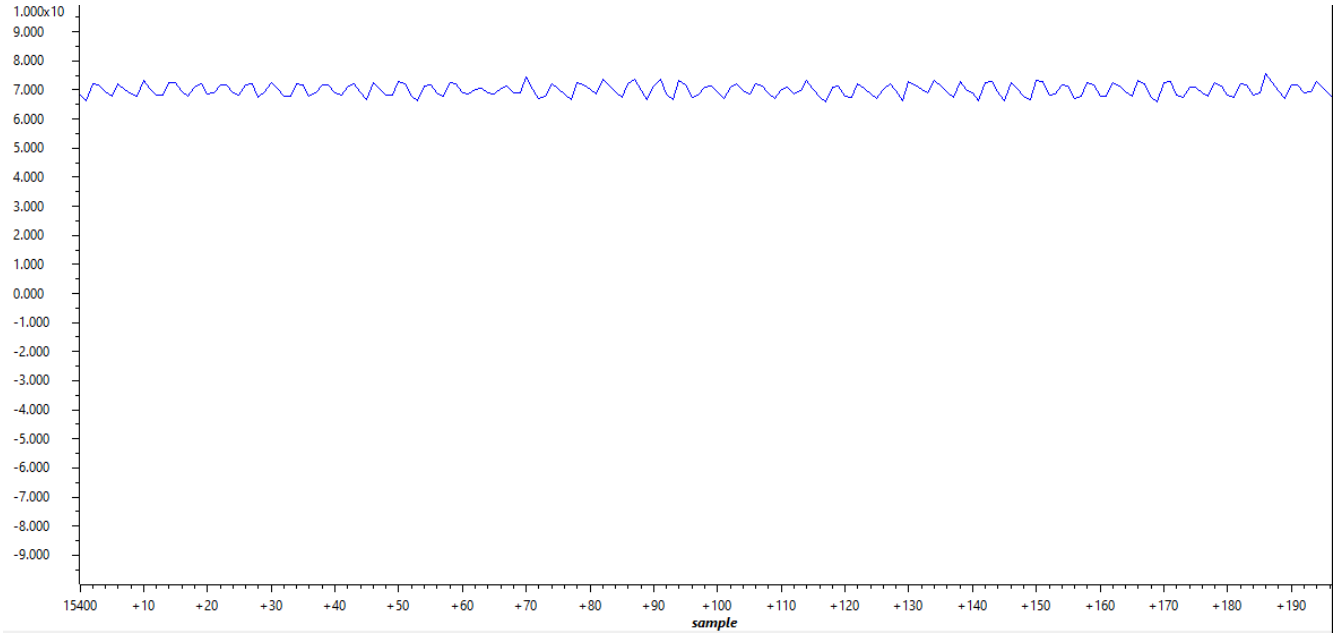

Figure 2-37. Open Loop Id

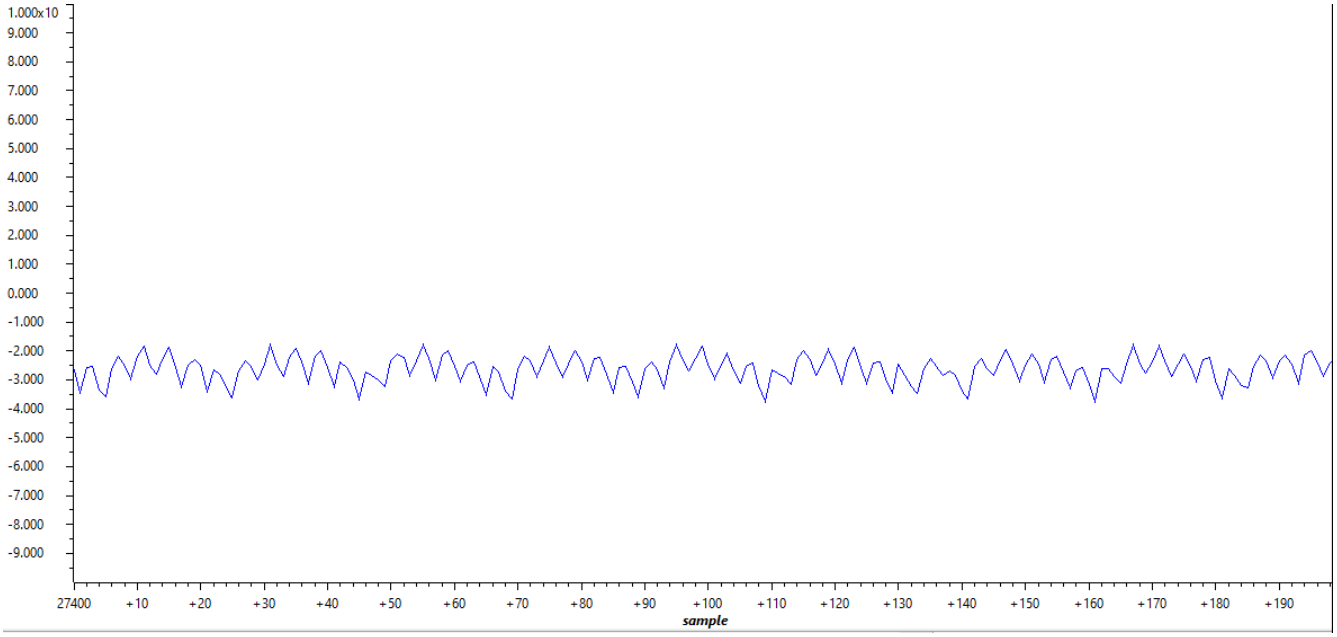

Figure 2-38. Open Loop Iq

2.6.2 Add Controllers to Close Current Loop

PI controllers must be added to close current loop. The PI controller here is implemented similar to C28 program for TIDM-02009. There are different implementations in CMSIS DSP library and others. The intend here is not to cover details on PI controllers. The key message is to make sure the controller is allocated in TCM. Running from TCM is critical for deterministic execution time. If not, significant amount of time will be taken from either content switching between cache and OCRAM or directly running in OCRAM. An example of attribute setting is given in line 1. Another example of one function call is shown in line 2 and 3. The attribute setting must match link command file and Memory Protection Unit configuration. More details can be found from the Technical Reference Manual.

```
1. __attribute__ ((section(".tcmb_code"))) static inline float32_t PI_run_series(PI_Obj * pi)
2. motor1.pi_id.fbackValue = motor1.I_dq_A[0];
3. motor1.Vout_dq_V[0] = PI_run_series(&(motor1.pi_id));
```

2.6.3 Read Id-Iq to Close Current Loop

Ideally, Id and Iq are constant values during closed current loop steady state operation. In most cases, it is not difficult to read them in expression window. If graph view is desired, they can be added to graph window by following Section 2.4.3. Here are the setup for log pointers. Line 1 is Id and line 2 is Iq. The plotted Id is in and Iq is in Figure 2-39 and Figure 2-40.

```
1. gLog_ptr[10] = &motor1.I_dq_A[0];
2. gLog_ptr[11] = &motor1.I_dq_A[1];
```

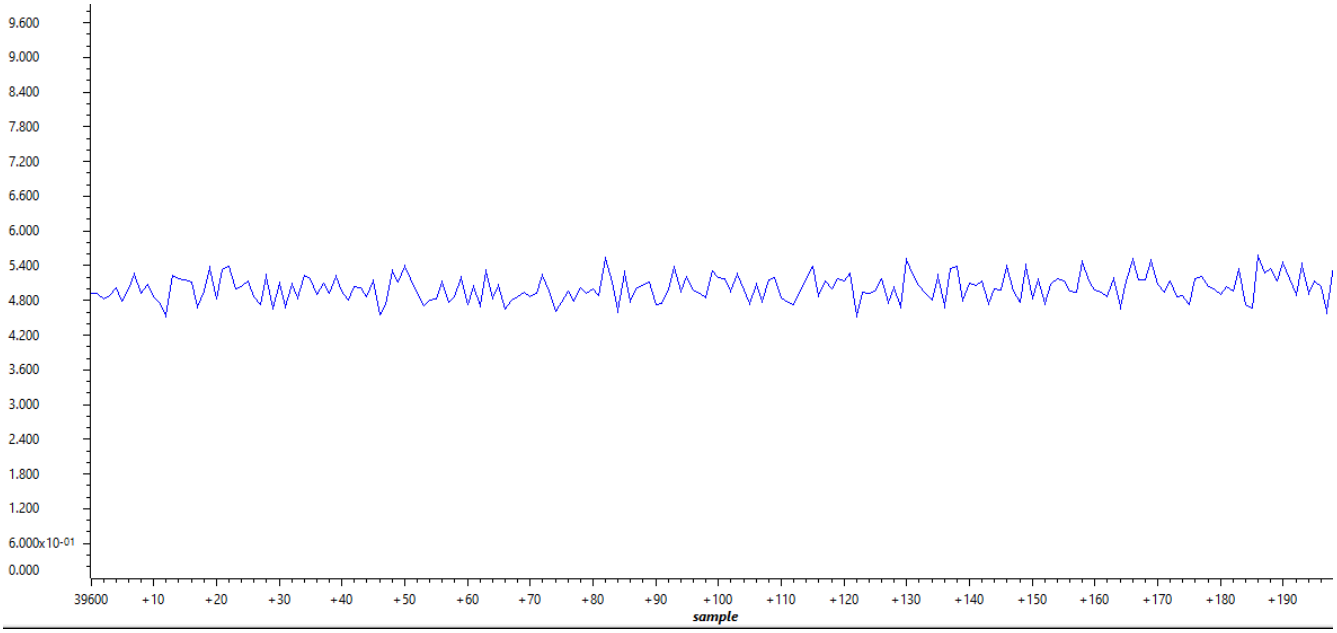

Figure 2-39. Closed Loop Id

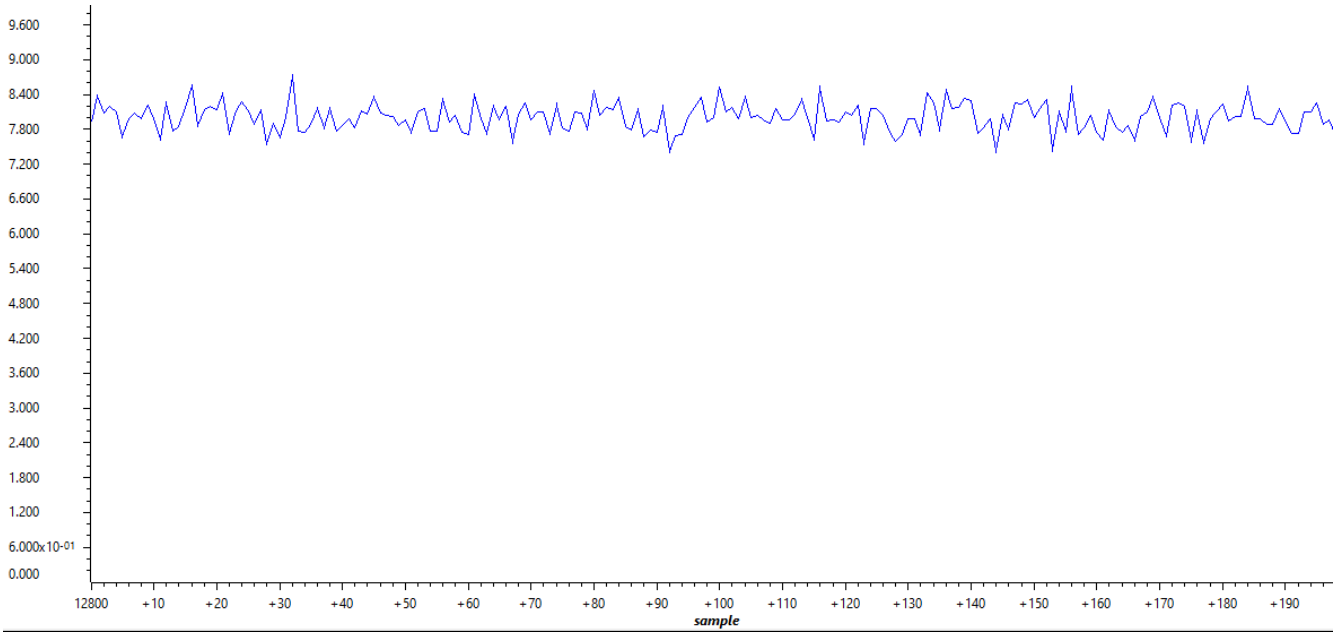

Figure 2-40. Closed Loop Iq

2.7 Add Software Resolver to Digital Converter

The software resolver to digital converter is implemented with ADC, DMA and DAC configured in Section 2.2.2 and Section 2.2.3. An overview of the configuration is summarized in Figure 2-41. In this work, PWM X is EPWM0 and PWM Y is EPWM7. EPWM0 at 10kHz is the one triggering phase A power switches and the source of synchronization. EPWM7 at 20kHz is dedicated to trigger EDMA0 for resolver excitation via DAC. ADC Start Of Conversion (SOC) is triggered at EPWM0 count zero. End Of Conversion (EOC) of ADC4 is the source of ADC INT1 containing the FOC loop. This section is to present functions and readings of the software resolver.

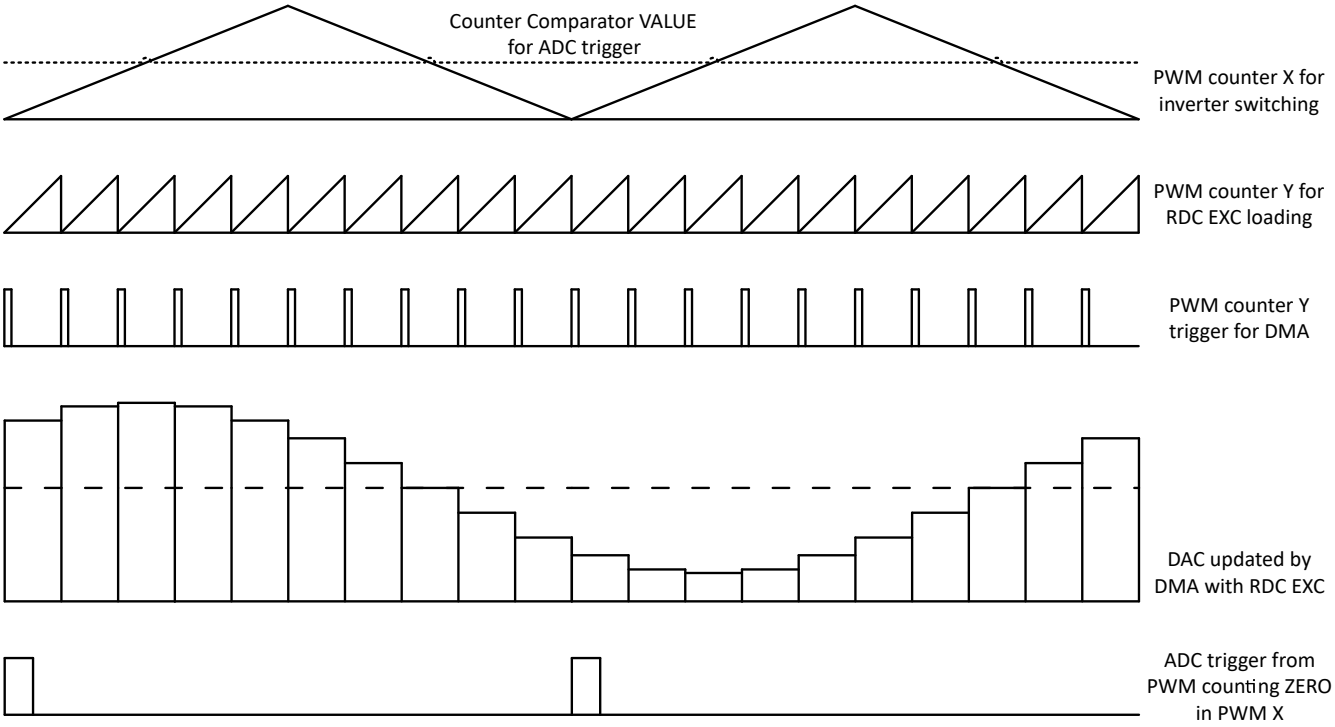

Figure 2-41. Software Resolver Synchronization and Excitation

2.7.1 Generate Excitation for Resolver Hardware

In Section 2.2.3, EDMA is configured to transfer data every EPWM7 SOCA event. To specify more details on the transfer, the function in the following line 1 is defined to initialize the EDMA with the location of data, size of data, and the location of DAC value register. To update the details, the function in the following line 2 is defined to change content of EDMA input table and DAC excitation output.

- uint16_t RDCexc_start(uint16_t *table, uint16_t table_size, EDMA_Handle dma_handle, uint32_t dma_ch, uint32_t dac_base)
- uint16_t RDCexc_update(uint16_t *table, uint16_t table_size, EDMA_Handle dma_handle, uint32_t dma_ch, uint32_t dac_base)

RDCexc_start should be called before control loop running, and RDCexc_update could be called while control loop running. The purpose of RDCexc_update is to reduce misalignment between excitation phase and sample time. Excitation phase can be adjusted by shifting the pointer of the input table before calling RDCexc_update. Examples are given in the following lines for both functions.

1. RDCexc_start(gRDCtable_ptr,20,gEdmaHandle[0],DMA_TRIG_XBAR_EDMA_MODULE_0,CONFIG_DAC0_BASE_ADDR);
2. RDCexc_update(gRDCtable_ptr,20,gEdmaHandle[0],DMA_TRIG_XBAR_EDMA_MODULE_0,CONFIG_DAC0_BASE_ADDR);

One element of a table in global scope is connected to gRDCtable_ptr. The table contains points of multiple full sine wave cycles. The element is in middle of the table so that the pointer can be shifted either right or left for excitation phase adjustment during operation. There are 20 points within one cycle. So, the length of EDMA input table is 20. EDMA handle, XBAR information and DAC address are available in configuration files.

It is worth attention that there could be pin mismatches in DAC channels between TIDM-02009 mother board and some versions of AM263x control cards. The workaround is to solder a blue wire on the mother board between resolver board excitation pin and DAC-A pin. The mother board top view is given in Figure 2-42 and the connection is shown in Figure 2-43.

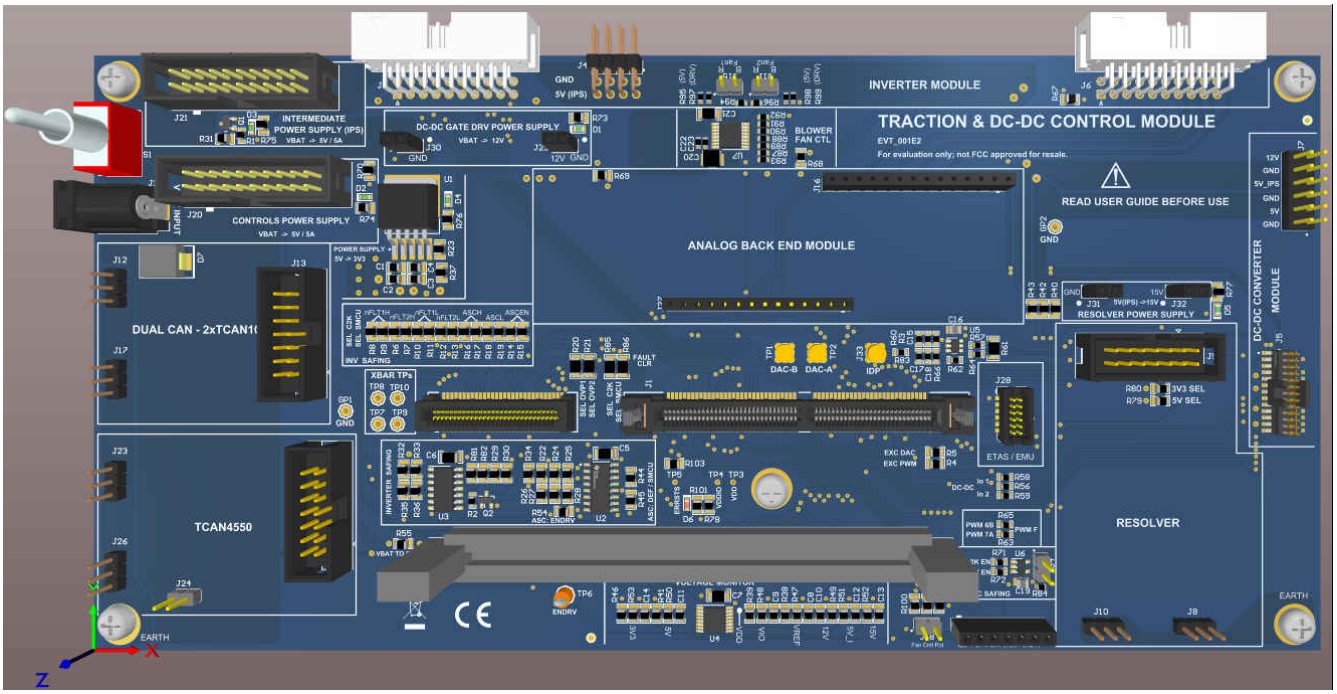

Figure 2-42. Mother Board Top View

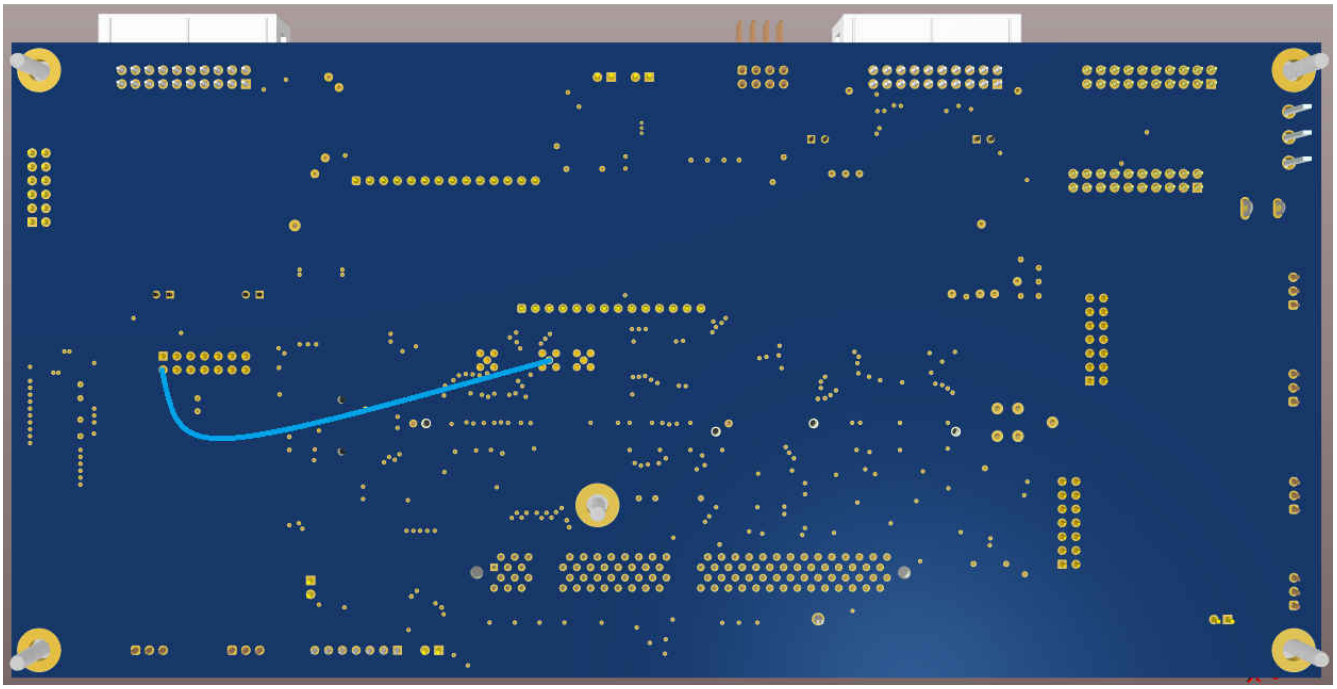

Figure 2-43. Mother Board Blue Wire for Resolver Excitation and DAC-A

2.7.2 Add Resolver Software

The software resolver is implemented similar to C28 in TIDM-02009. There are two functions listed as the following. Line 1 is to initialize resolver structure, and line 2 is to compute angle and speed. After initialize the structure, implementation details need to be updated according to features of resolver hardware. More details can be found in the structure definition.

1. static inline void resolver_init(Resolver_t *resolver);
2. static inline void resolver_run(Resolver_t *resolver);

It is worth attention that there are many ways to implement math functions like sin/cos. In this implementation, sin/cos functions for resolver phase lock loop inputs are moved out of "resolver_run" so that it is easier to find out all math functions and make changes. The following line 1 and 2 must be called before the above line 2. Standard C library functions are used here as an example.

1. resolver1.res_theta0_sin = sinf(resolver1.res_theta0);
2. resolver1.res_theta0_cos = cosf(resolver1.res_theta0);

2.7.3 Read Resolver Software Outputs

At initialization, outputs of software resolver are passed to global variables with pointers as shown in the following line 1 and 2. As motor angle is always changing during rotation, it will not be meaningful if it is not plotted in graph window at a fixed sample frequency. Line 3 and line 4 pass the addresses of resolver angle and speed to two log pointers. They are logged every assigned number of interrupts by the log functions described in Section 2.4.3.

1. resolver1.resolver_theta = &resolver_theta;
2. resolver1.resolver_omega = &resolver_omega;
3. gLog_ptr[12] = &resolver_theta;
4. gLog_ptr[13] = &resolver_omega;

2.8 Close Speed Loop With Rotor Speed

2.8.1 Add Speed Loop Controllers

PI controllers must be added to close speed loop. The PI controller here is implemented similar to C28 program for TIDM-02009. As the motor used in this work is an induction motor, there is another part, slip compensation, to close speed loop. It should be removed or replaced according to the actual motor in end equipment. The intend here is to show how to use the framework. The following line 1 and 2 shows the PI controller function calls, and line 3 to 6 shows the slip compensation. As there is division in line 6, line 4 and 5 are added to avoid zero in divisor.

1. motor1.pi_spd.fbackValue = resolver_omega;
2. motor1.pi_iq.refValue = PI_run_series(&(motor1.pi_spd));
3. aci1.IMDs += aci1.Kr * (motor1.I_dq_A[0] - aci1.IMDs);
4. aci1.IMDs = ((aci1.IMDs < 0.001) && (aci1.IMDs >= 0)) ? 0.001 : aci1.IMDs;
5. aci1.IMDs = ((aci1.IMDs > -0.001) && (aci1.IMDs < 0)) ? -0.001 : aci1.IMDs;
6. aci1.Wslip = aci1.Kt * motor1.I_dq_A[1] / aci1.IMDs;

2.8.2 Add Speed Loop Demo Program

In order to demonstrate speed loop, there is a short demonstration created to spin motor forward for 10 second, spin motor backward for 10 second, and stop. This is a infinite loop with a counter timing at interrupt frequency. This part of program is shown in Figure 2-44. As the input of speed loop reference is in rad/s of electrical speed, the demonstration input in RPM of mechanical speed need to be converted to electrical speed according to motor properties.

```
gDemoCnt++;
if(gDemoCnt < gDemoPos )
{
    gSpeedRef = 0;
}else if(gDemoCnt < gDemoNeg)
{
    gSpeedRef = gDemoSpd;
}else if(gDemoCnt < gDemoMax)
{
    gSpeedRef = -gDemoSpd;
}
else
{
    gDemoCnt = 0;
}
/* mech rpm to elec rad/s */
motor1.pi_spd.refValue = gSpeedRef / 60 * PAIRS * TWO_PI;
```

Figure 2-44. Speed Loop Demonstration Program

2.8.3 Read Motor Speed from Graph Window

When motor spinning, motor angle keeps changing. The best way to read it is via graph window. The procedure is similar to Section 2.4.3. The pointer is assigned as the following line 1.

1. gLog_ptr[14] = &gDemoRPM;

Motor angle at forward rotation is shown in Figure 2-45, and backward rotation is shown in Figure 2-46. The angle is created from software resolver. Sine envelope is recorded for a transition from forward to backward in .

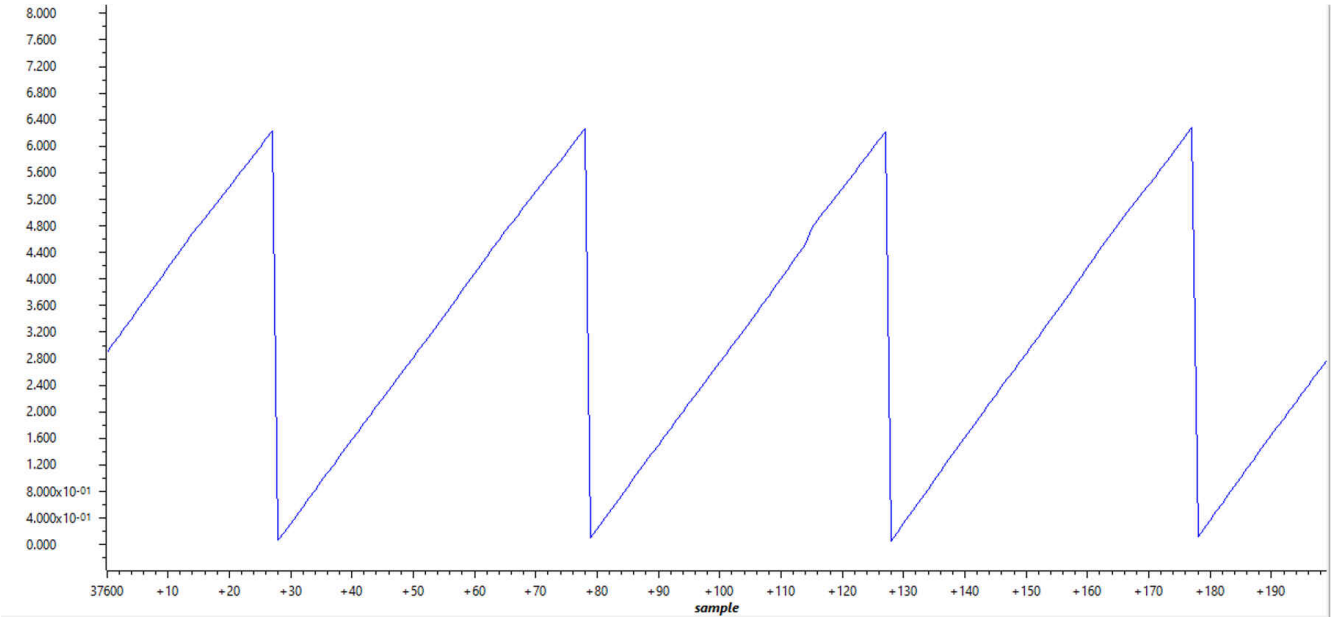

Figure 2-45. Motor Forward Rotation

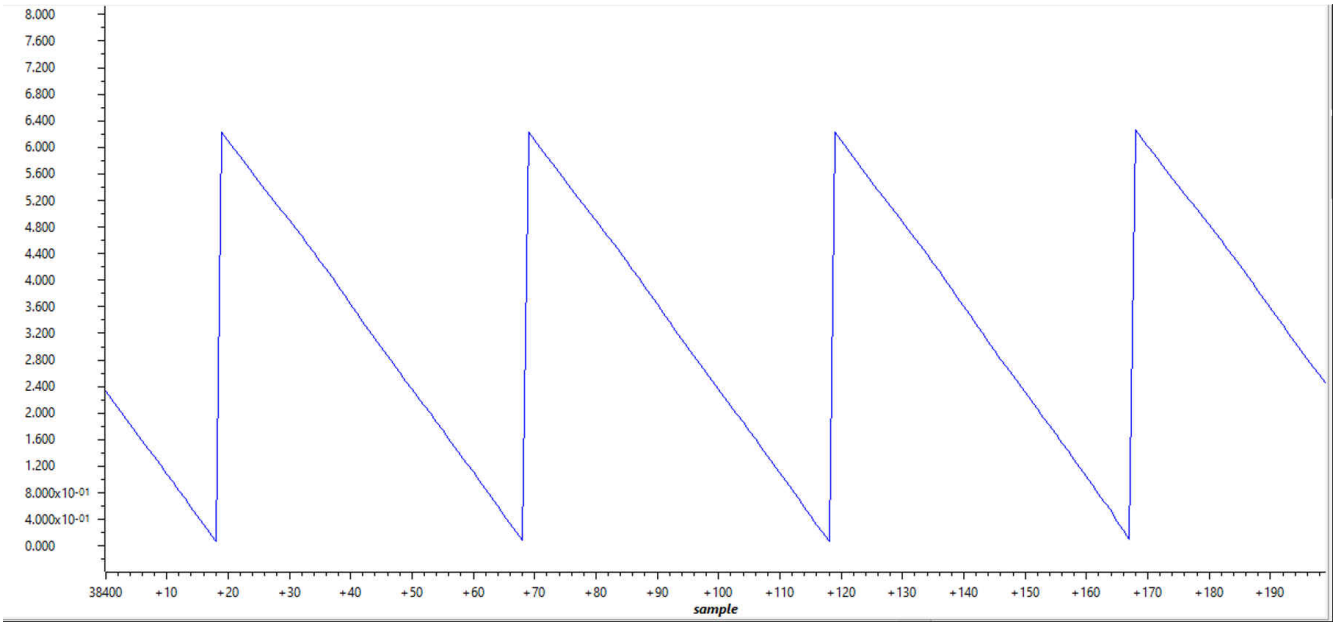

Figure 2-46. Motor Backward Rotation

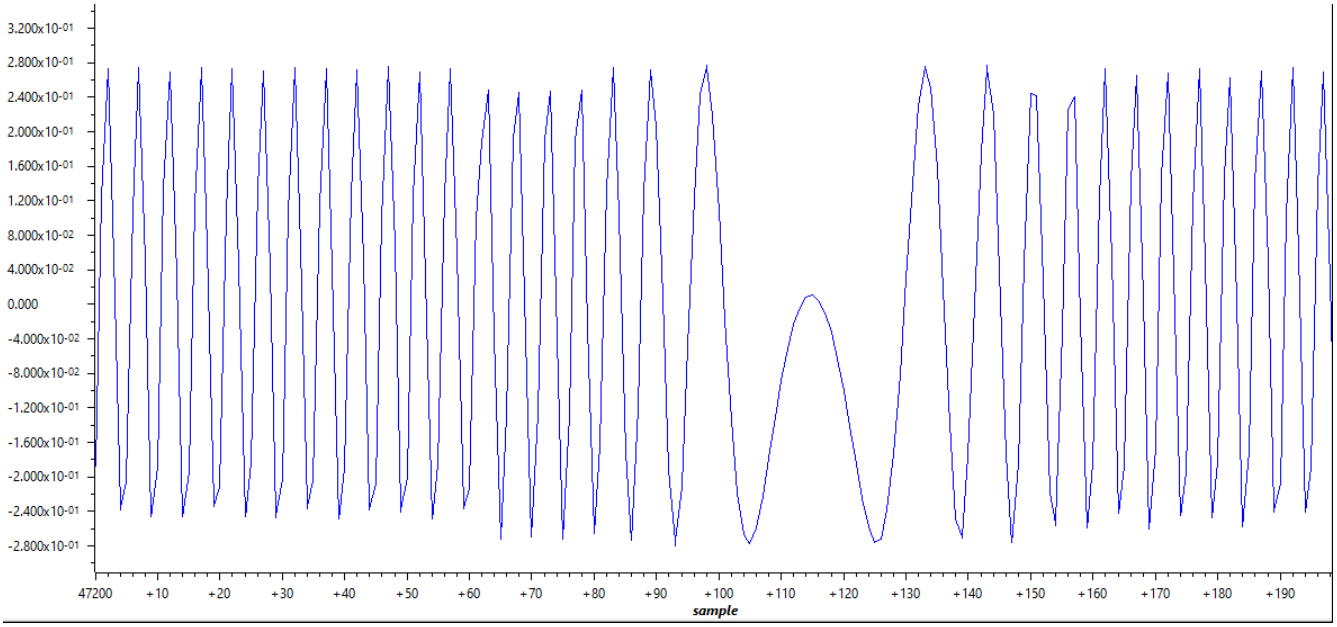

Figure 2-47. Resolver Sine Envelope for a Transition from Forward to Backward

3 A Brief Guide to Code Migration

This section starts with overviews on AM263x SoC architecture and SDK resources. With the available resources in mind, some tips are summarized in the following two parts on code migration from AM24 and C28 family.

3.1 SoC Architecture Overview

An overview of SoC architecture is presented in Figure 3-1. The SoC includes 2 clusters of R5F cores, 64-bit bus based L2 memory system, 32-bit bus based control and connectivity peripherals. There are also resources to help achieve security and safety features. The R5F clusters can be configured to either dual mode or lock-step mode, which gives opportunity to optimize computation and safety features. The L2 memory system supports many features like on-chip memory, off-chip memory, manipulation of memory, Standard Ethernet, Industrial Ethernet, and security. The control peripherals include all popular features, like PWM with shadow registers and SAR-ADC synchronized with PWM. The connectivity peripherals offer commonly used I2C, SPI, CAN, LIN, and UART.

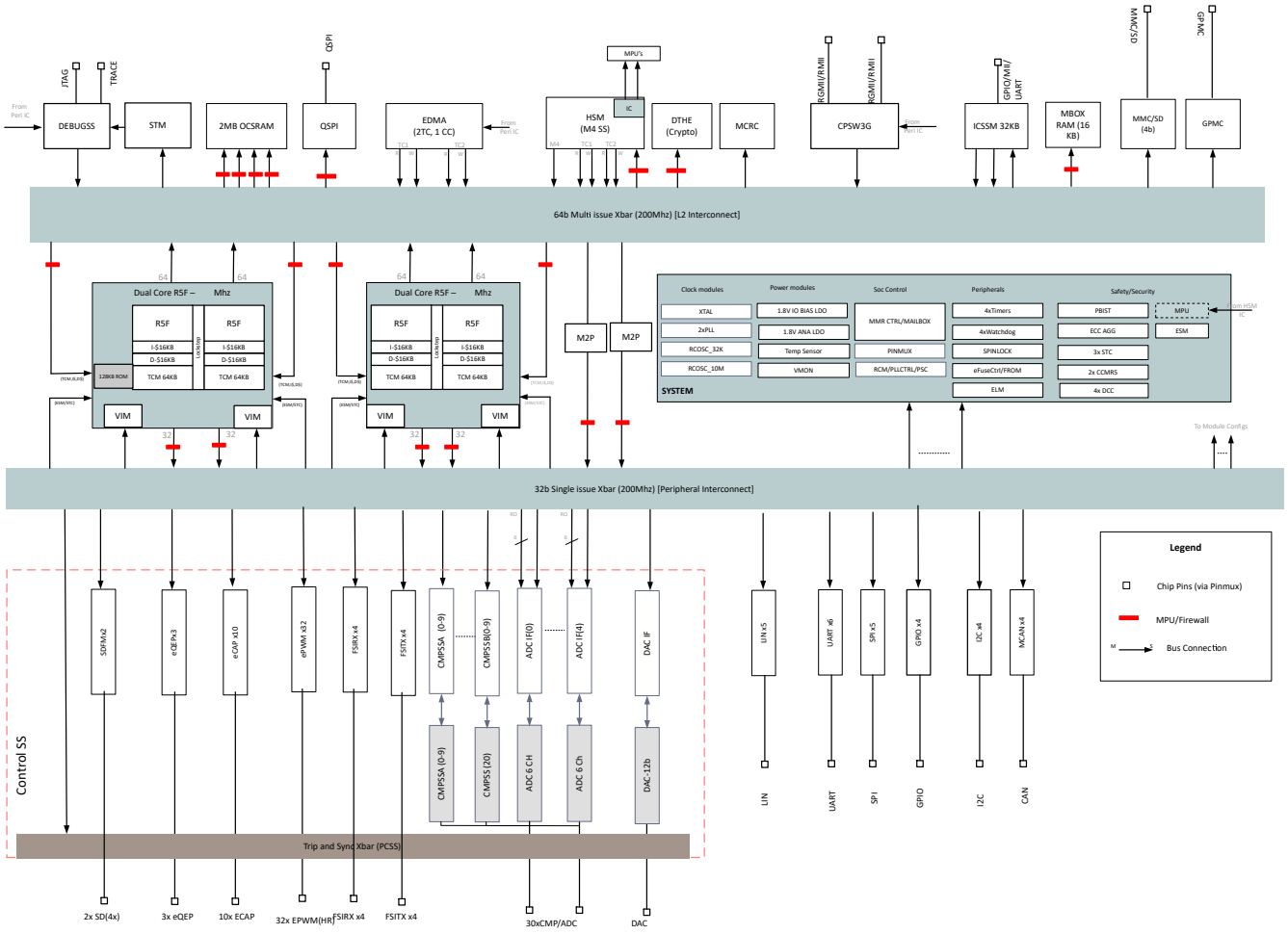

Figure 3-1. AM263x Block Diagram

3.2 SDK Resources Overview

A overview of SDK resources is summarized in Table 3-1. More details can be found in "README_FIRST_AM263X.html" under the SDK installation directory. By default, the directory is at a path like "C:\ti\mcu_plus_sdk_am263x_xx_xx_xx_xx". The "examples/" folder is the most important one for beginners. It includes examples matching AM24 for connectivity and C28 examples for control. In most cases, "Ctrl+left click" in CCS guides user to the API declaration headers. Due to the inheritance from C28 and AM24, control API headers includes both declaration and definition as static inline while connectivity API headers only shows declaration. And, the connectivity API definition must be found in source file. Both header and source files are kept in "source/" folder. If CCS can't reach the desired details, there is a good chance that the details stay in one of source files inside the "source/". Another approach is to look into "API Reference" of "README_FIRST_AM263X.html".

Table 3-1. SDK directory structure

Folder/Files	Description
${SDK_INSTALL_PATH}/	
README_FIRST_AM263X.html	Open this file in a web browser to reach SDK user guide
makefile	Top level makefile to build the whole SDK using "make"
imports.mak	Top level makefile to list paths to dependent tools
docs/	Offline HTML documentation
examples/	Example applications for AM263X, across multiple boards, CPUs, NO-RTOS, RTOS
source/	Device drivers, middleware libraries and APIs
tools/	Tools and utilities like CCS loading scripts, initialization scripts.
${SDK_INSTALL_PATH}/source/	
board/	Board peripheral device drivers
drivers/	SOC peripheral device drivers
industrial_comms/	Industrial Communication Protocol stacks and Industrial Protocol FW HAL(Firmware and Hardware Abstraction Layer)
kernel/	NO RTOS and RTOS kernel and Driver Porting layer (DPL) for these environments
${SDK_INSTALL_PATH}/examples/	
drivers/	SOC and board focused device driver examples. The examples are based on both NO-RTOS and RTOS.
empty/	Template projects to copy into workspace and customize based on application needs
industrial_comms/	Example for EtherCAT Slave

3.3 Code Migration From AM24

AM24 shares similar architecture and connectivity peripherals as AM263x. But, the control peripherals are completely different. Generally, programs related to connectivity can be migrated with little or none modification while those related to control peripherals must be updated for details in Technical Reference Manual of AM263x. As for SDK, every version is unique. But, APIs at Driver Porting Layer are almost identical for AM24 and AM263x. Difference can be found in SOC Specific Device Drivers.

It is worth attention that there are differences in architecture and connectivity between AM24 and AM263x despite of their similarities. For example, AM24 offers more powerful support for Industry Ethernet, Gigabit Industrial Communication Subsystems, and more flexibility expansion of external memory, DDR4 Subsystem. In AM263x, there are features for 100-Megabit Industry Ethernet and 16bit/32bit parallel bus. Programs related to features like those must be redesigned for migration from AM24 to AM263x.

Another point is that AM24 R5F cores are up to 800MHz while AM263x R5F cores are 400MHz. During code migration, significant change on execution time must be expected and properly handled. It is critical to make sure the execution time stay within requirements. However, for traction inverters, given that both 400MHz and 800MHz cores are much higher than the classic MCUs, the execution time taken by either of them should not be a concern in most cases.

3.4 Code Migration From C28

C28 shares similar control peripherals as AM263x. But, the architecture and connectivity peripherals are completely different. Generally, programs related to control peripherals can be migrated with little or none modification while those related to CPU, memory management, and connectivity peripherals must be updated for details in Technical Reference Manual of AM263x.

As well know, direct operation on registers has been widely used in C28 programs in the past. The movement from register operations to API calls happened in recent years. The change from register operation to API calls simplifies the adoption of more complex MCUs. But, it takes effort to migrate from register user to API user. The effort is inevitable for either C28 and AM263x. Once the effort is made, it is not difficult to work with AM263x control subsystem as the concept from modules, like ADC and PWM, are very similar. Some examples on the similarity of control APIs are given in Table 3-2. And also, the powerful Sysconfig is available in AM263x SDK. It offers intuitive user interface for system configuration. End users are enabled to directly apply their ideas on control peripheral into configuration without worrying about API details. The APIs widely used in control loop have been given as part of the framework and discussed in Section 2.

Table 3-2. Examples on Similarity of API Definitions

API Function	AM263x Definition	C28 Definition
Get ADC Result	static inline uint16_t ADC_readResult (uint32_t resultBase, ADC_SOCNumber socNumber)	static inline uint16_t ADC_readResult (uint32_t resultBase, ADC_SOCNumber socNumber)
Set PWM Duty Cycle	static inline void EPWM_setCounterCompareValue (uint32_t base, EPWM_CounterCompareModule compModule, uint16_t compCount)	static inline void EPWM_setCounterCompareValue (uint32_t base, EPWM_CounterCompareModule compModule, uint16_t compCount)

On the other hand, despite the similarities, there are some difference in SDK and some implementations with similar names. As shown in Section 3.2, the SDK structure of AM263x is very different from the SDK of C28. Although they share similar control peripherals and APIs are similar, it is still necessary to understand the different in SDK structure so that detail can be easily found during development. And, for some features like XBAR, both C28 and AM263x have XBAR synchronizing operation between modules but XBAR in AM263x is much more powerful than XBAR in C28. It comes with a challenge that it must be sufficiently understood and properly configured. XBAR program from C28 cannot be directly applied into AM263x projects.

4 Summary

Traction inverter is a typical application requiring both outstanding CPU throughput and flexible control peripherals. This work briefly introduces the architecture of a traction inverter reference design with AM263x software framework for traction inverters. Then, the focus stays on how to configure, change, and debug the traction inverter reference design with the software framework. A step-by-step guide is presented in order to help users understand details in the framework. On the other hand, a brief guide summarizes resources, tips, and tricks on code migration from AM24 and C28 family. It is critical to understand features and make adjustments during system design. After all, this work demonstrates a software framework to reduce the learning curve and accelerate the adoption of AM263x in traction systems.

5 References

- *ASIL D Safety Concept-Assessed High-Speed Traction, Bi-directional DC/DC Conversion Reference Design*
- *AM263x Sitara™ Microcontrollers Data Sheet*
- *AM263x Sitara Processors Technical Reference Manual*
- *AM263x Control Card User's Guide*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265