Application Note

Using the Sitara MCU AM263x in an Automotive SiC Traction Inverter

ABSTRACT

The traction inverter is the main power processing system in an electric vehicle. The real time control of this power processing block being a critical task, needs a microcontroller unit (MCU) with low latency peripherals and deterministic processing time. The newly introduced Texas Instruments MCU AM263x is ideally suited for this application. However, to accelerate market adoption of the MCU, it is important to demonstrate it in a reference design closely matching the electric vehicle traction inverter. This demonstration will also significantly help customers to reduce software development time and focus on development of their product. This document focuses on presenting the software framework for traction inverters and demonstrating the capabilities of the real time control modules of AM263x with the help of the reference design *Automotive, High-Power, High-Performance SiC Traction Inverter Reference Design*.

Table of Contents

List of Figures

Trademarks

1 Introduction

The primary aim of this document is to describe how the Texas Instruments Sitara™ AM263x series of microcontrollers (MCUs) can be used in an automotive traction inverter reference design. This reference design is a 800 V, 300 kW SiC based inverter reference design from TI and Wolfspeed that attempts to provide a starting point for designers and engineers to achieve a high-performance, high-efficiency traction inverter system (*Automotive, High-Power, High-Performance SiC Traction Inverter Reference Design*). The application note to this reference design explains the hardware and software design considerations in detail. However, the application note does not cover the software design aspects related to AM263x. This document tries to cover that area so that the reference design documentation can be complete in all aspects.

WARNING

TI intends this application note to be operated in a lab environment only and does not consider the reference design to be a finished product for general consumer use.

TI intends this application note to be used only by qualified engineers and technicians familiar with risks associated with handling high-voltage electrical and mechanical components, systems, and subsystems.

High voltage! There are accessible high voltages present on the board. The board operates at voltages and currents that can cause shock, fire, or injury if not properly handled or applied. Use the equipment with necessary caution and appropriate safeguards to avoid injuring yourself or damaging property.

CAUTION

Do not leave the design powered when unattended.

1.1 Key System Specifications

The key system specifications are summarized in Table 1-1.

Table 1-1. Key System Specifications

PARAMETER	SPECIFICATIONS (UNITS)	NOTES
P_{OUT}	300 kW	Rated output power
V_{DSmax}	1200 V	Maximum Drain-Source Voltage
V_{DC}	800 V	DC bus voltage recommended
I_{DC}	300 A	DC Bus current
f_{SWmax}	60 kHz	Based on the gate driver bias power
I_L	360 A	AC output RMS current
L_{PL}	5.3 nH	Parasitic Inductance including DC link capacitors and Bus bar
C_{DC}	300 µF	DC link capacitor
L_{DC}	3.5 nH	DC Bus capacitor ESL
Power Density	32 kW/L	
Dimensions	28 cm × 29 cm × 11.5 cm	
Weight	6.2 kg	
Volume	9.3 L	
Area	812 cm^2	
P	5 bar	Coolant Operating Pressure
ΔP	200 mbar	Pressure Drop

- For information on the isolated gate driver, see the UCC5880-Q1 data sheet.
- For information on the Microcontroller, refer to the AM2634-Q1 and TMSF280039C-Q1 data sheet.
- For information on the bias supply, refer to UCC14240-Q1 data sheet.

- For information on the integrated modules, see the CAB450M12XM3 data sheet.
- For higher ambient temperatures, the DC-Link voltage and DC-Link current must be de-rated according to the included DC-Link capacitor ratings. See the 1100 V / 100 µF CX100µ1100d51KF6 data sheet provided by FTCAP GmbH for more detailed information.
- The included cold plate is a Wieland MicroCool CP3012-XP. To calculate the thermal resistance (°C/W) and pressure drop (bar) versus flow rate (liters/min.), refer to the CP3012-XP data sheet provided by Wieland MicroCool Inc. for more detailed information.
- The included current sensor board uses the LEM LF 510-S. See the LF 510-S data sheet provided by LEM USA Inc. for more detailed information.

2 AM263x Overview

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. The accelerator is named as control subsystem and includes interface modules like ADC, DAC, and PWM. This document describes how one R5F core and the group of interface modules required for one traction inverter power stage can be set-up and the control of the power stage implemented with the reference design TIDM-02014. More details of the SoC can be found in the *AM263x Sitara™ Microcontrollers* data sheet and the *AM263x Sitara Processors Technical Reference Manual*. Details of the inverter hardware can be found in the design guide of the *Automotive, High-Power, High-Performance SiC Traction Inverter 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 aggregates, 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 real time control with AM263x 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 it is via instructions running in the core. The memory view from integrated development environments (IDE) like Code Composer Studio™ (CCS) will not be able to read it. 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 is discussed in detail in this application note: *AM263x for Traction Inverters*. 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 quite fast, but that of program in on-Chip RAM is much slower. Also, the operation of transferring the program from on-Chip RAM to cache takes some time and introduces some non-predictive latency. If this latency is significant to the requirements of the application, it is highly recommended to store the application program in TCM. Details on TCM address can be found in AM263x technical reference manual. In this example, 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 Instruments 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 of these modules are available in AM263x technical reference manual. Configuring these modules can be done using an intuitive system configuration tool, SYSCONFIG, with reduced exposure to implementation details. AM263x *Software Development Kit* (SDK) provides details of the SYSCONFIG tool. The key for module synchronization is in configuring PWM synchronization input/output in EPWM Time Base section and ADC Start-of-Conversion (SOC) trigger in EPWM Event-Trigger section. Time Base is for aligning 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 this 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. As multiple ADC units can share the same SOC, 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 either at the start of one ADC conversion or at 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 below *\source\drivers*. The header files have more details in comments.

2.1 AM263x Control Card and Traction System Framework

The AM263x Control Card Evaluation Module (EVM) is an evaluation and development board for the Texas Instruments Sitara™ AM263x series of microcontrollers (MCUs). This EVM provides an easy way to start developing on the AM263x MCUs with on-board emulation for programming and debugging as well as buttons and LEDs for a simple user interface. The control card also enables header pin access to key signals through the use of a high speed edge connector (HSEC) that can directly be plugged onto the TIDM-02014 main control board.

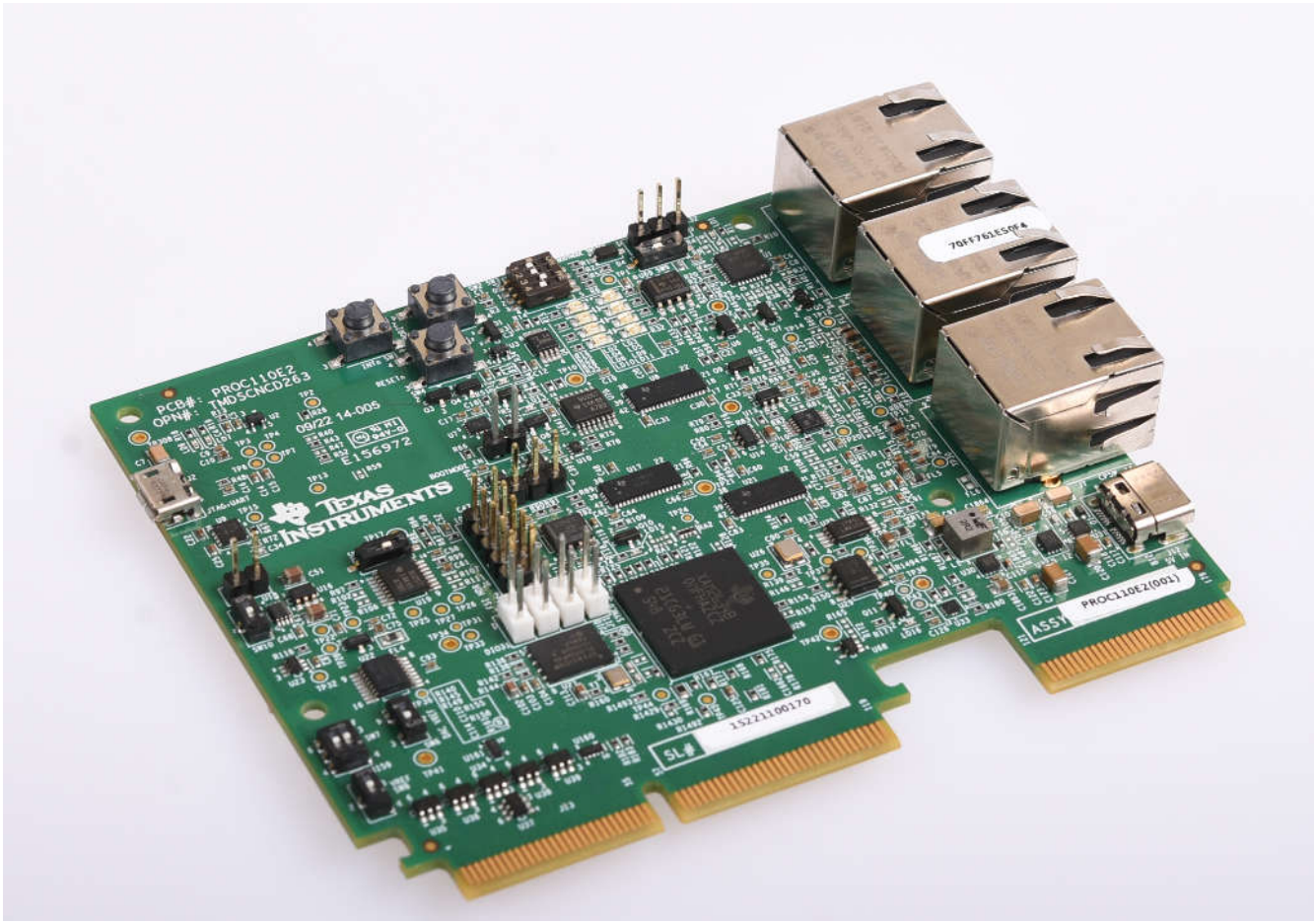

Figure 2-1. AM263x Sitara Control Card

The traction framework includes the resources highlighted in Figure 2-1 and is built into the traction system in Figure 2-2. TIDM-02014 is the traction inverter reference design hardware. It includes a power stage, gate drivers, current and voltage sampling, resolver analog front end, control system power tree and 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.

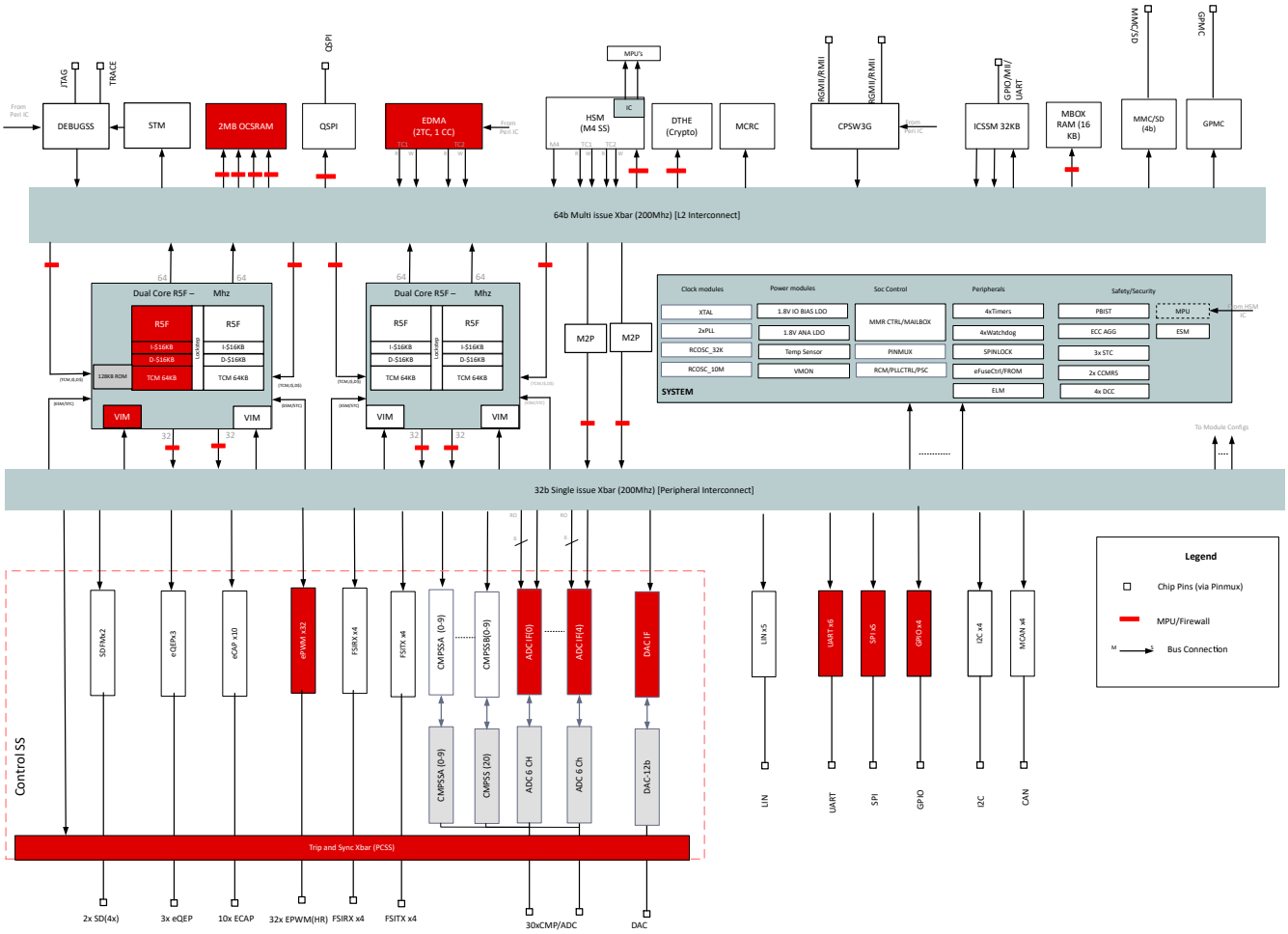

Figure 2-2. Traction Framework Resources

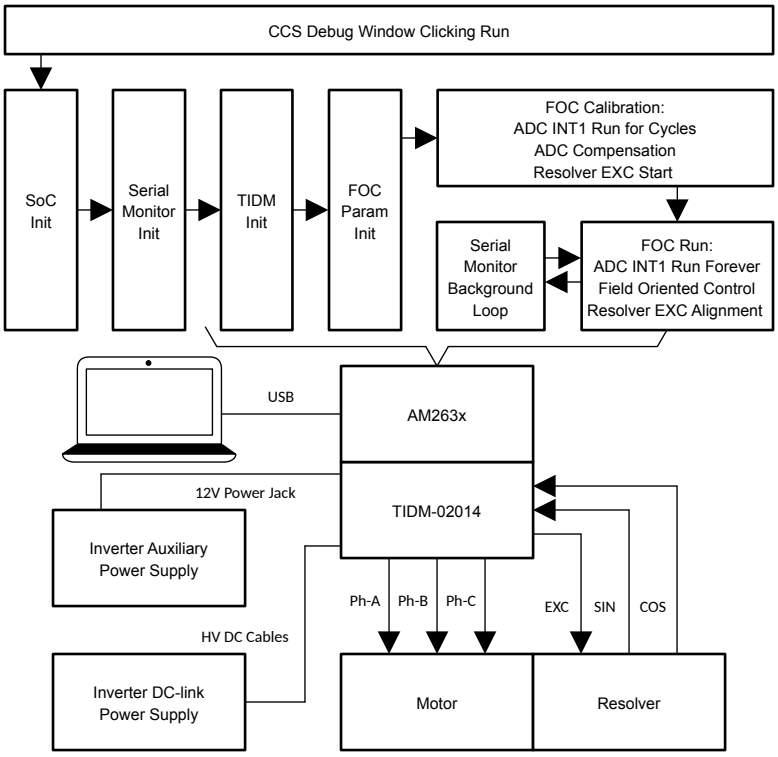

Figure 2-3. Traction System Diagram

3 Guide to Running TIDM-02014 Traction Inverter

3.1 Software Set-up

After setting up the traction system as shown in the Figure 2-3, the TIDM-02014 reference design needs the software environment to be set-up to run it. We used *Code Composer Studio* (CCS) as the integrated development environment for this purpose.

As a first step, please download and install CCS version 12.2.0 or newer that is recommended for this project. *CCS User's Guide* provides more details about CCS installation and use.

While it is not strictly necessary, establishing a real time debug interface helps a lot in debugging any power conversion application with an MCU. For AM263x, real time debug is enabled by UART connection between CCS and AM263x. The user interfaces of the AM263x control card are shown in Figure 3-1. The details on hardware connection can be found in *AM263x Control Card User's Guide*. The control card offers both JTAG and UART debugging ports in one USB Micro-B connector. A detailed explanation of creating a target configuration file for the debug ports can be found in the application note *AM263x for Traction Inverters*. As this application note also covers in detail how to configure control peripherals, interrupts and driver interfaces, we are not repeating those sections here.

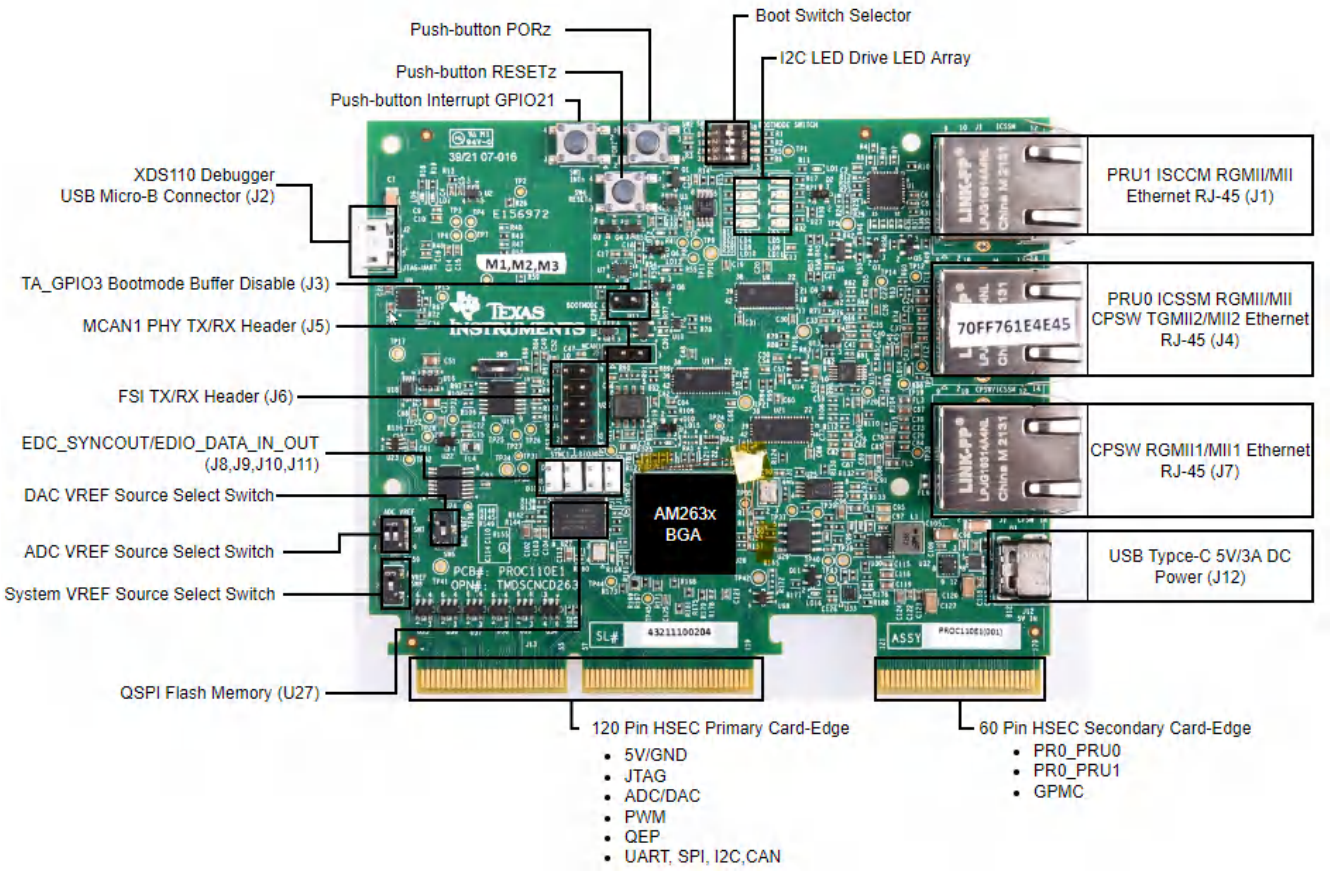

Figure 3-1. User Interfaces of AM263x Control Card

3.1.1 Code Composer Studio Project

To import the software project for TIDM-02014 into CCS, click Project → Import CCS Projects, and browse to *TractionDemo_am263x-cc-rev2_r5fss0-0_nortos_ti-arm-clang2* folder and click Select Folder. Select the project named TractionDemo_am263x-cc-rev2_r5fss0-0_nortos_ti-arm-clang2 and click Finish. The project is now be visible in the Project Explorer pane in CCS. The user's guide provides further details on *Importing Projects into CCS*.

The folder libraries/foc includes the typical FOC modules, including park and clarke transforms, PID functions, and estimators. These modules are independent of the specific device and board.

The folder Motor includes motor drive control files that call motor control core algorithm functions within the interrupt service routines and background tasks. The user can add their own code for system control, communication, and so forth. These modules are specific to this reference design project but are independent of the device and board.

Board-specific, motor-specific and device-specific files are in root folder and Motor folder. These files consist of device specific drivers to run the design. If you want to migrate the project to your own board or other devices, you only need to make changes to the files trinv.syscfg, Motor_param.h, and Resolver_param.h based on the pin assignments and features of your device or board.

3.1.2 Software Structure

The general structure of the project is shown in Figure 3-15. The device peripheral configuration is based on AM263x Driverlib and is partially generated using SysConfig, making the code portable across hardware and devices. To port the reference design software to a different board or device, the user only needs to change the trinv_hal.c, trinv_hal.syscfg, and trinv_hal.h files and the parameters in trinv_settings.h.

??? Figure shows the project software flow diagram of the firmware that includes one ISR for real time motor control, a main loop that allows the user to update motor control parameters through debug window. The ISR is

triggered by ADC End of Conversion (EOC). The functions that run in the main ISR are defined in trinv.h header file. In addition, in this design, accurate motor position is sensed through a resolver interface. The function that reads the ADC values for the resolvers signals and performs corresponding position, speed calculations runs in the control law accelerator (CLA) which is an independent processing core. The function is defined in trinv_cla_tasks_cpu1.cla file.

3.2 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.

3.2.1 Confirm CCS Features

The recommendation is to check the following CCS driver file if the CCS version is older than 11.1. The configuration of Cortex_R5 needs to be similar to Figure 3-2. If any line is missing, it is necessary to add the line showing in Figure 3-2. 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 3-2. CCS GTI UART Driver for R5F

3.2.2 Create Target Configuration File

The AM263x controCARD offers both JTAG and UART ports in one USB port. The details on hardware connection can be found in *AM263x Control Card User's Guide*. The control card offers both JTAG and UART debugging ports in one USB Micro-B connector. 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 3-3 to Figure 3-10. Briefly, a target configuration file is created and then configured with both JTAG and UART. The UART COM Port in Figure 3-10 need to match PC Device Manager COM Port for JTAG probe Application/User UART. The Baud Rate in Figure 3-10 need to be consistent with SoC UART Baud Rate configured in next step.

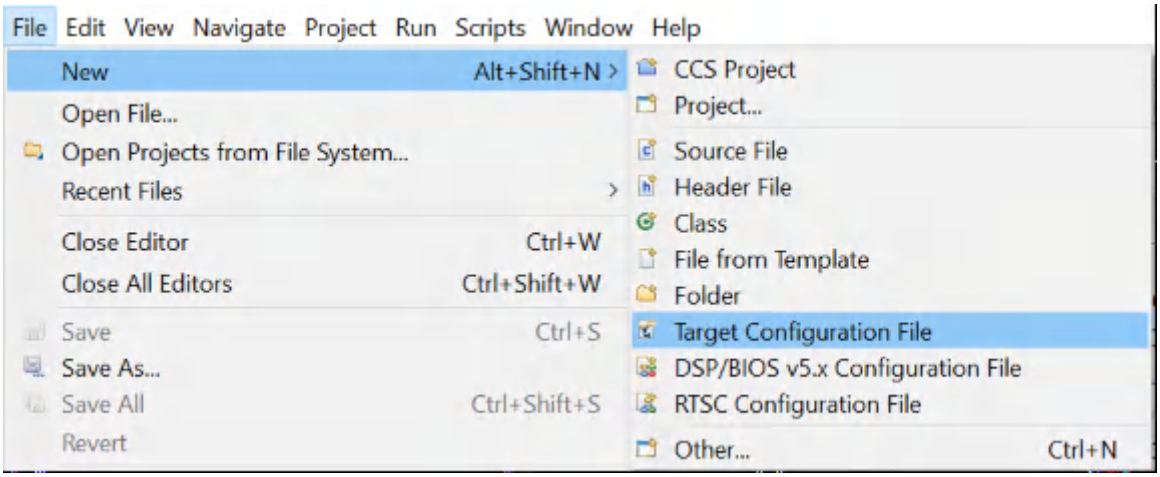

Figure 3-3. Create New Target Configuration File

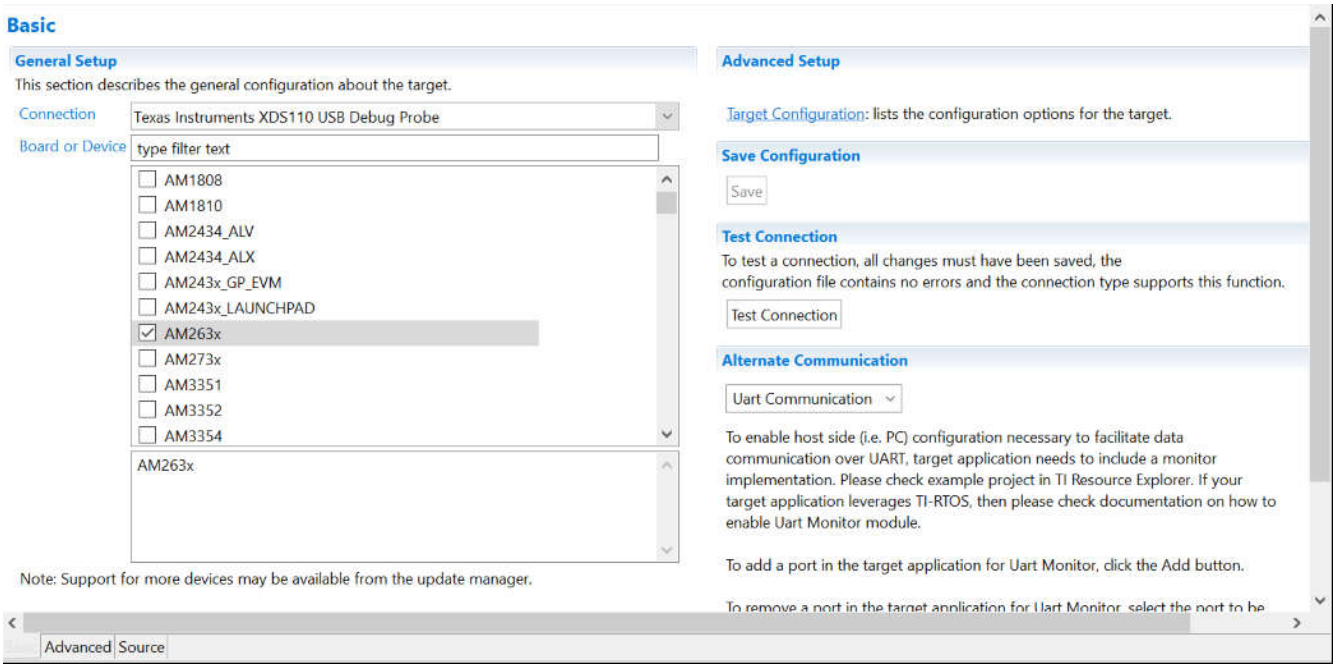

Figure 3-4. Select JTAG Connection and Device

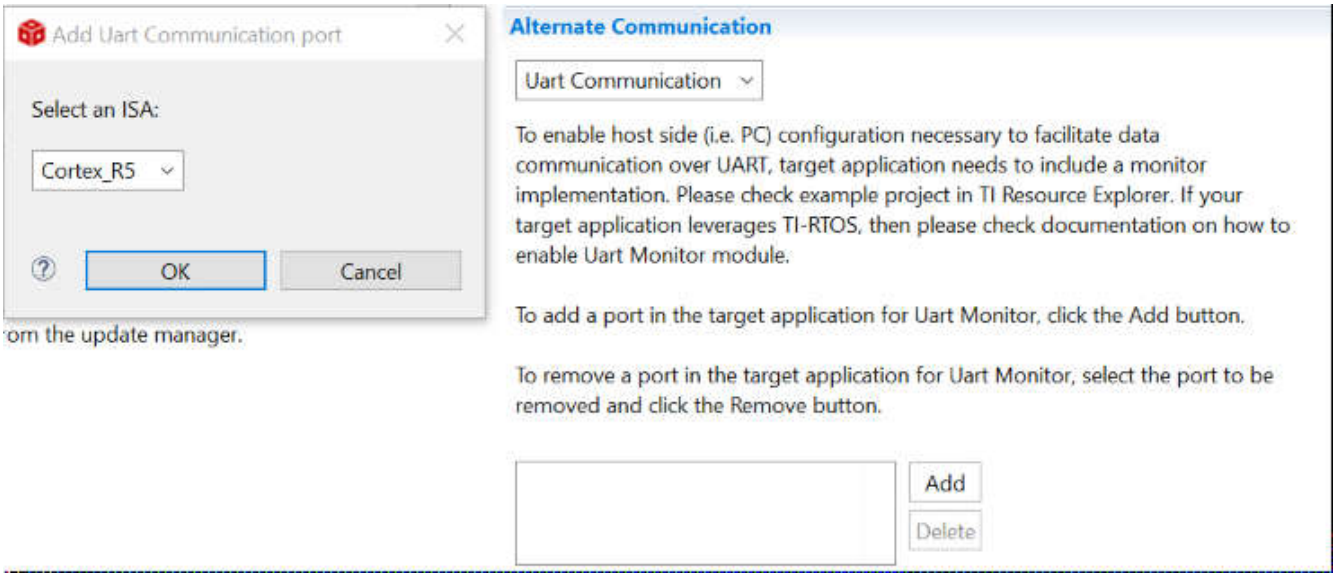

Figure 3-5. Add UART Communication Port

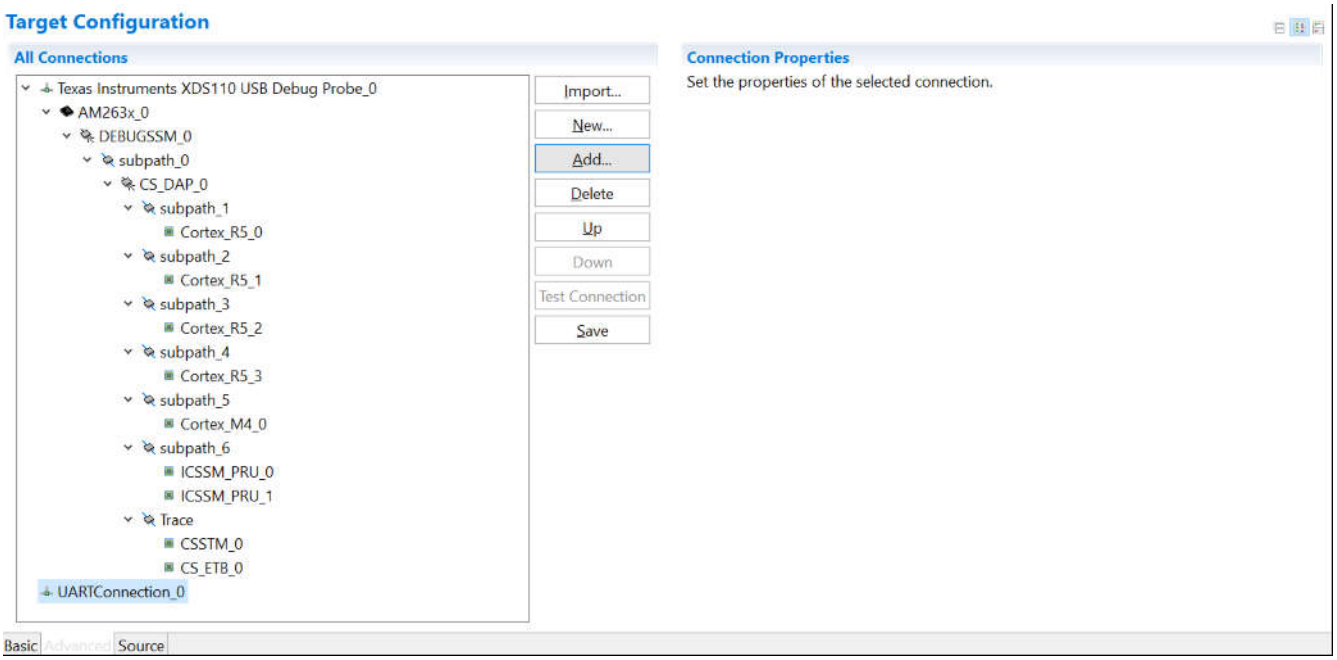

Figure 3-6. Open Advanced Target Configuration

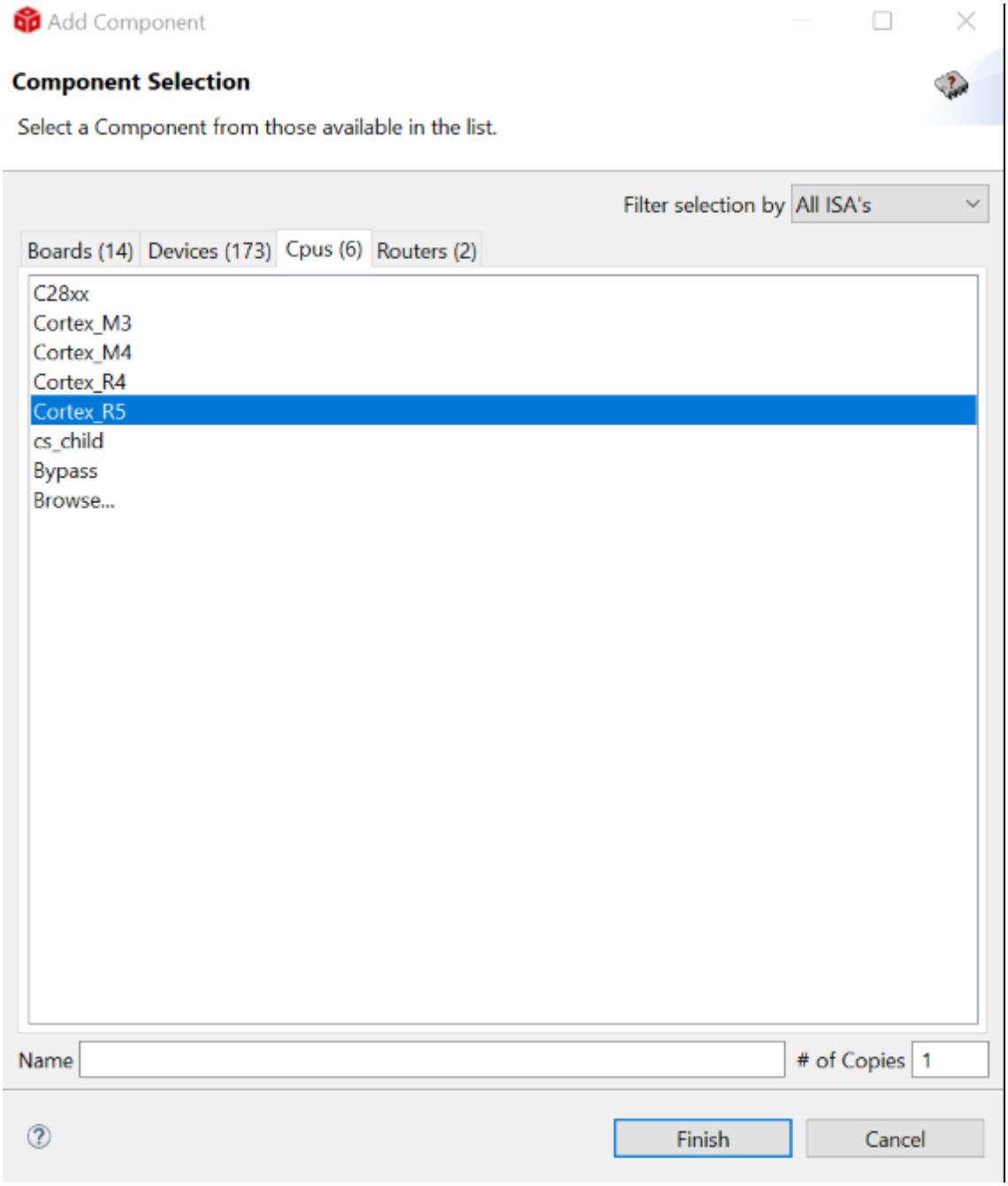

Figure 3-7. Add Component

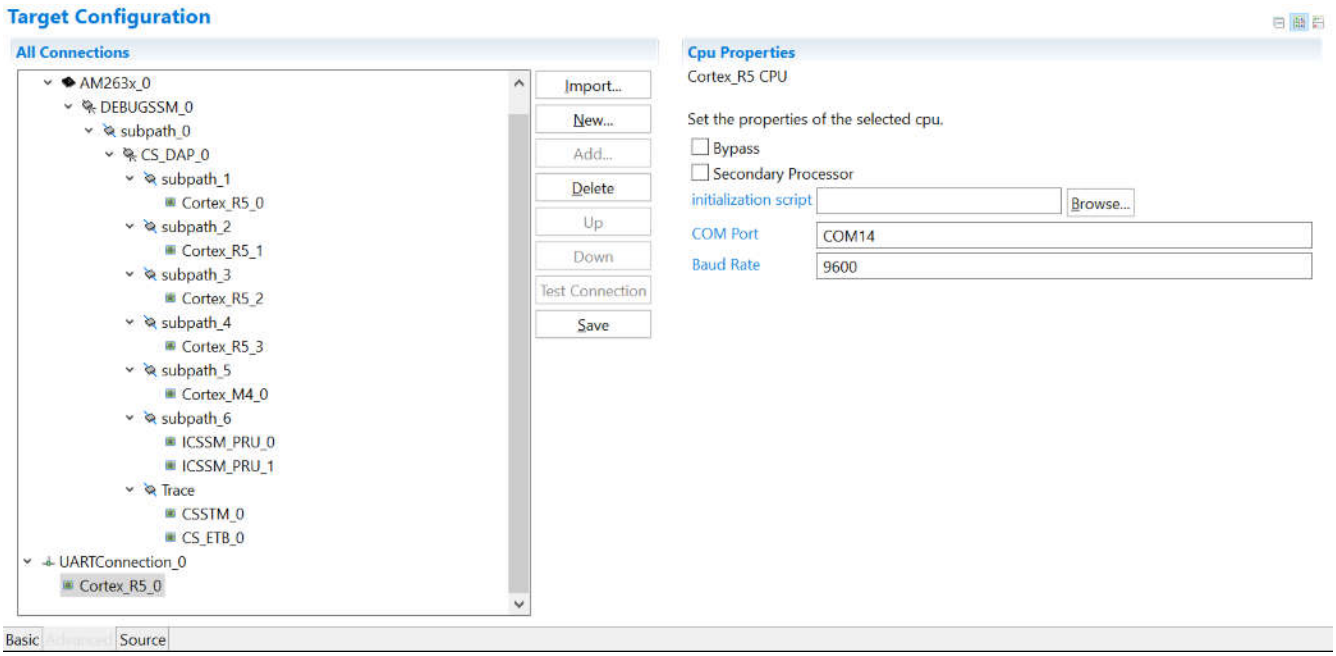

Figure 3-8. Select CPU Properties

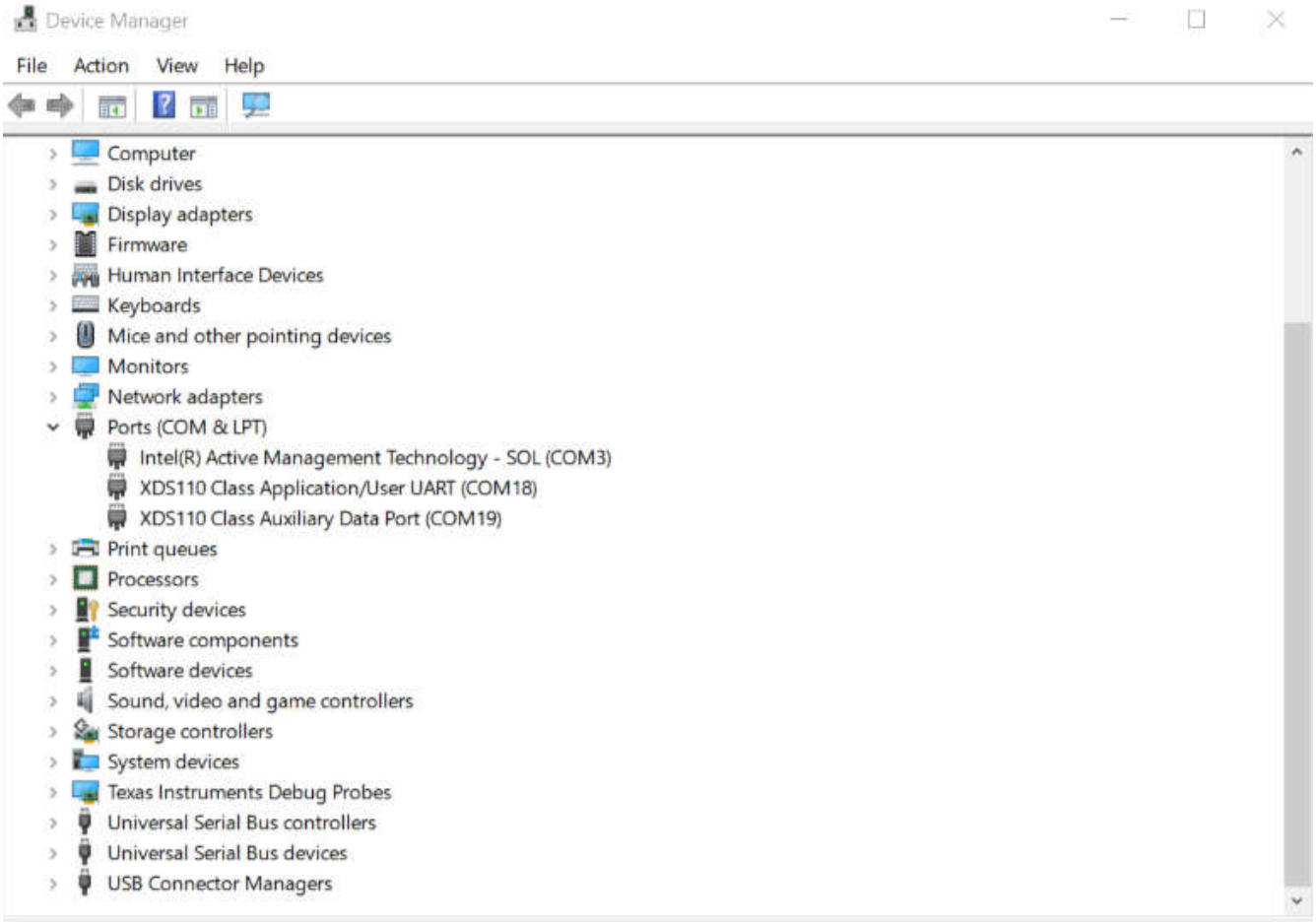

Figure 3-9. Find XDS110 UART COM Port

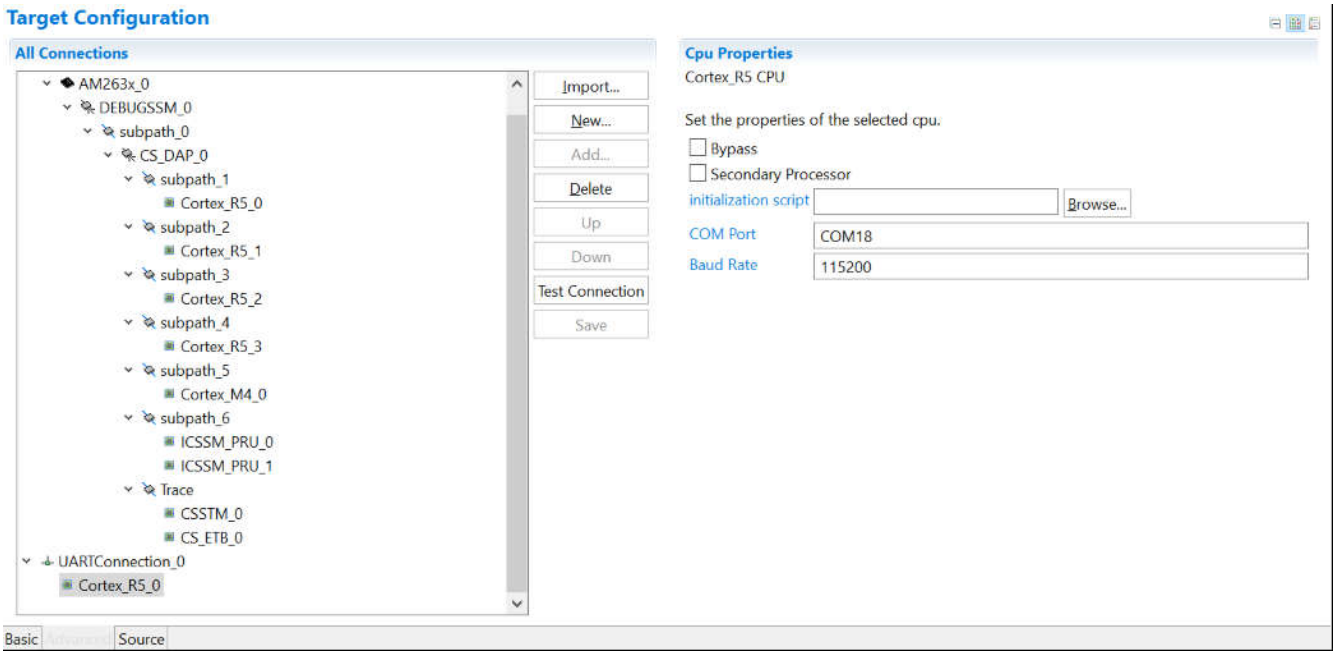

Figure 3-10. Update CPU Properties in Advanced Target Configuration

3.2.3 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 3-11 and configure UART0 instance for Serial Command Monitor at Figure 3-12. 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 3-13.

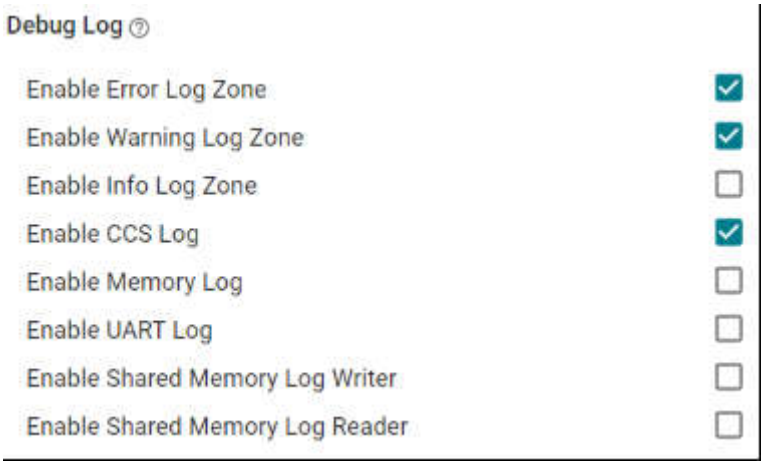

Figure 3-11. Disable UART Log in Debug Log

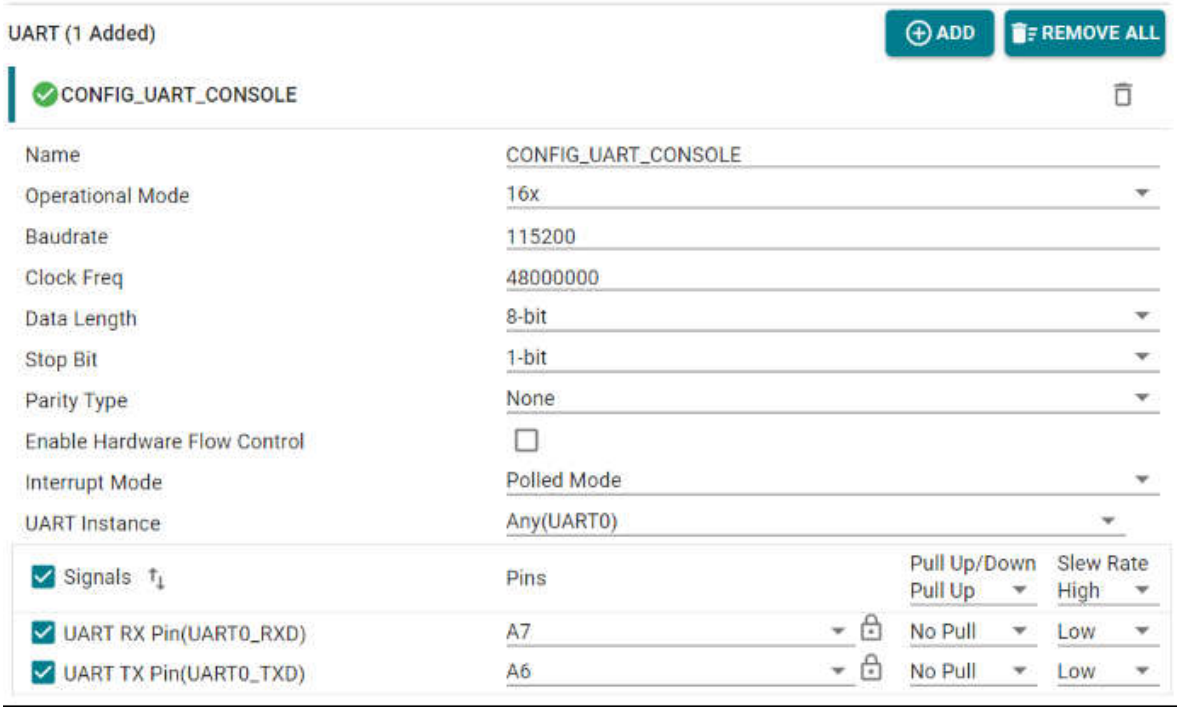

Figure 3-12. Configure UART0 Instance

```
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 3-13. Add Serial Monitor Function Calls

3.2.4 Launch Real Time Debug

After building the program, debug window should be opened with the target configuration file created in Section 3.2.2. If the created target configuration file is not already opened, it can be located by following Figure 3-14 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 3-15. 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 3-16 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 3-17.

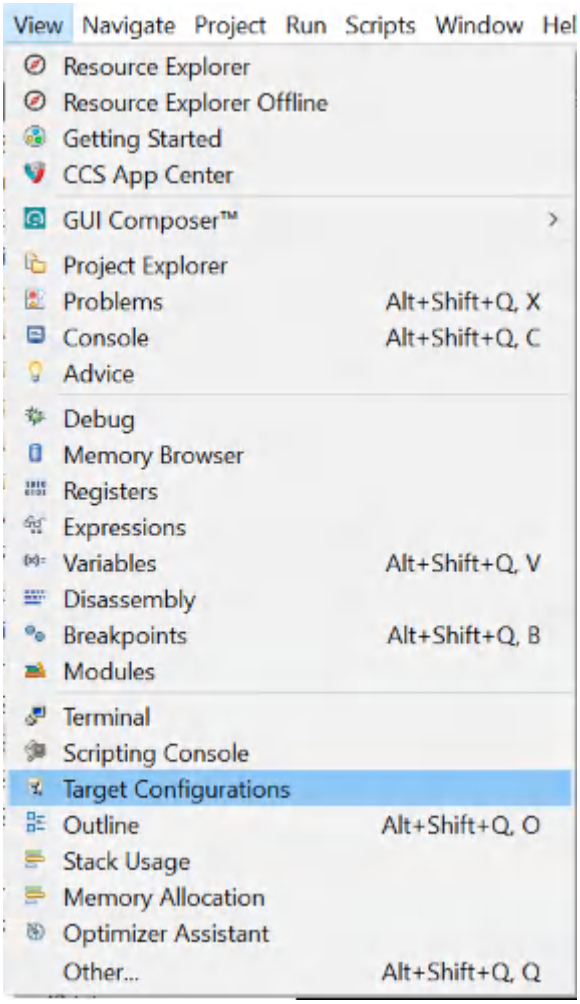

Figure 3-14. Locate Target Configuration File

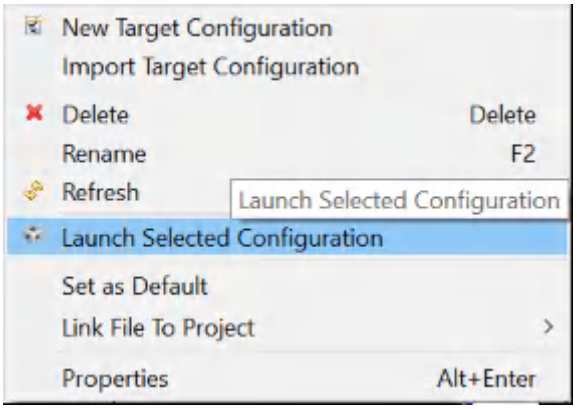

Figure 3-15. Launch Selected Configuration

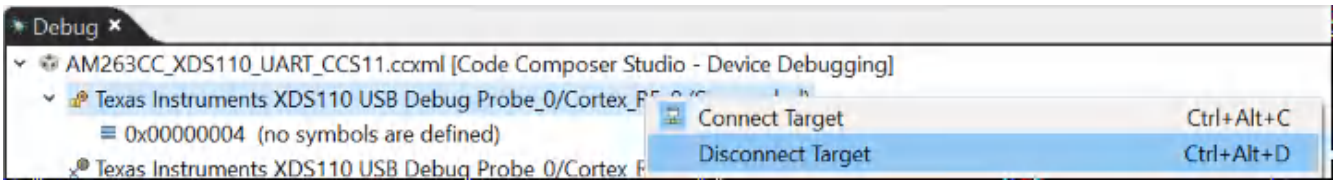

Figure 3-16. Disconnect JTAG Connection

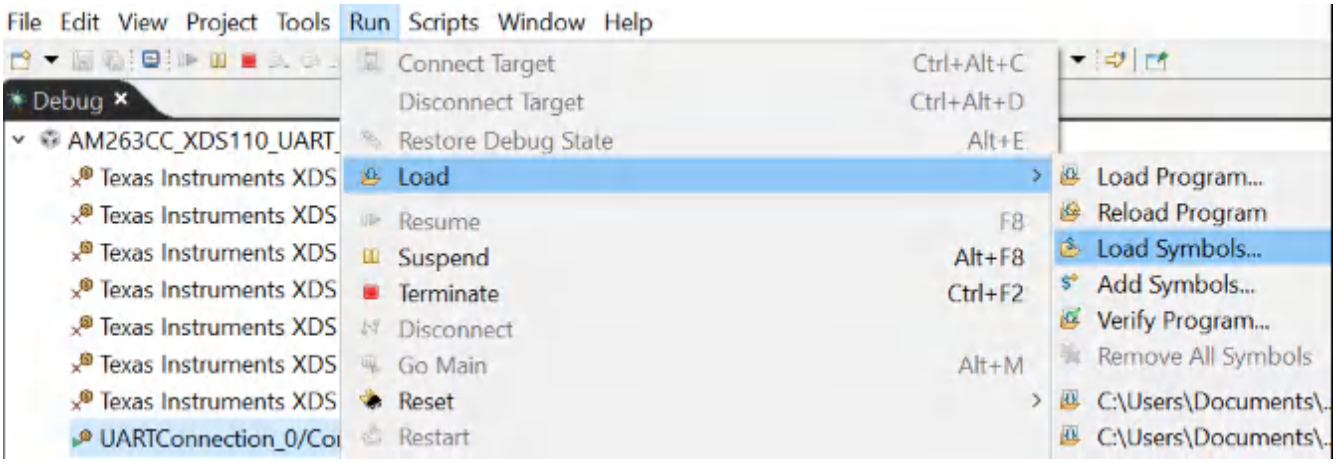

Figure 3-17. Establish UART Connection

3.3 Running the Code

The motor control software can be configured for different test modes to enable incremental software testing. For example, if neither of the CLOSED_CURRENT_LOOP or CLOSED_SPEED_LOOP symbols are defined, the Vd and Vq reference values are directly set to default test values. This mode can be used for testing the PWM setup and open-loop inverter operation. Similarly, only defining CLOSED_CURRENT_LOOP and not CLOSED_SPEED_LOOPallows setting up the id, iq references directly. In this case, the motor speed can be set by setting the ramp generator frequency. The user motor type and the corresponding parameters can be found in the trinv_setting.h file. The default setting can be changed by changing the value of the USER_MOTOR macro. If the user intends to use an outside the list of predefined motors, the user can copy the template used.

3.3.1 Project Setup

Import the project into CCS and select the appropriate build configuration. Right-click on the project in the Project Explorer and select Rebuild Project. Confirm that the Console pane shows that the project built without any errors.

On successful completion of the build, with the tidm_02014 project selected, go to Run → Debug or click the Debug button on the tool bar. By default the project will launch a debug session using the AM263x.ccxml file in

the project. AM263x.ccxml is configured to use the Texas Instruments XDS110 USB Debug Probe on board the TMDSCNCD263 control card.

After clicking Debug, CCS will automatically connect to the target, load the output file into the device, and change to the CCS Debug perspective. The program should be halted at the start of main().

Click the Continuous Refresh button in the Expressions window tool bar to tell CCS to update the data continuously at a rate defined in the CCS debug preferences.

3.3.2 Running the Application

Run the code by going to Run → Resume or clicking the Resume button in the tool bar. The project can now run and the variables display in the Expressions window. Check the following to confirm the application and hardware set up are working:

- The green power LEDs on the gate-drive boards must be on. If the gate-drivers are initialized without faults, none of the red, nFault LEDs are on. Gate drive initialization status can be checked through the tripFlagDMC.fault.UCC5880_status variable.
- Similarly other variables in the tripFlagDMC struct show the status of other faults. If no fault flags are set, to run the test motor, the runMotor can be set to RUN_MOTOR. Your variables need to appear similar to what is shown in Figure.
- If no faults are detected, the motor1.isrCount must increment continuously.
- Check the calibration offsets of the motor inverter board. The offset values of the motor phase current sensing values must be equal to approximately half of the scale current of ADC.
- The PWM output for motor drive can also be probed with an oscilloscope.

You can halt the CPU by first clicking the Suspend button on the toolbar or by selecting Target → Suspend. To run the application from the start again, reset the controller by clicking on the CPU Reset tool bar button or clicking Run → Reset → CPU Reset and then clicking on the Restart button or Run → Restart. You can close the CCS debug session by clicking the Terminate button or by clicking Run → Terminate. This will halt the program and disconnect CCS from the controller.

Note that it is not necessary to terminate the debug session each time you change the code. Instead you can go to Run → Load → Load Program... (or Reload Program... if it is the same file). CCS will also automatically prompts to ask if you want to reload your executable if it detects you have rebuilt it.

3.4 Get Samples From ADC and Read Samples Through 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.

3.4.1 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);

3.4.2 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();

3.4.3 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 3-18. 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];

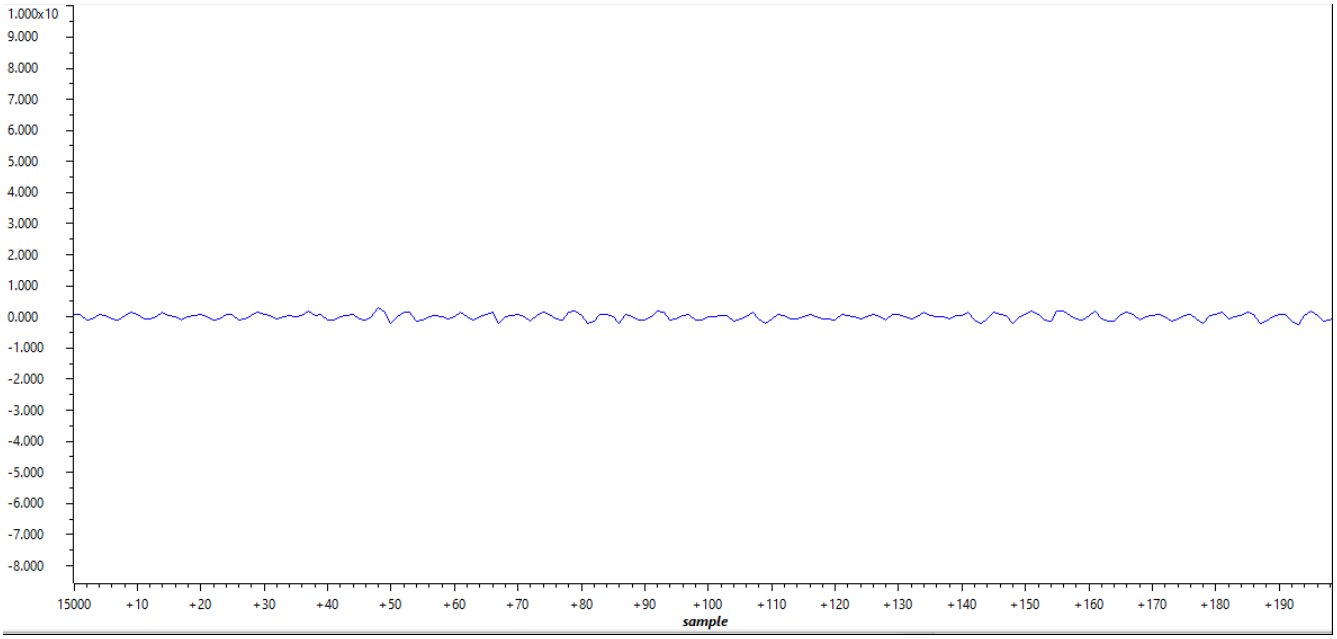

Figure 3-18. Plotted Phase A Current at No Load

During bring up of the system at low voltage, it is recommended to calibrate the offset of the DC bus voltage feedback, as there may be an offset voltage of 10-20 V due to the large measurement scale.

3.5 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.

3.5.1 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-02014. 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-02014. 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));

3.5.2 Read SVPWM Duty Cycles in Graph Window

Similar to plotting ADC samples in graph window in Section 3.4.3, 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 3-19.

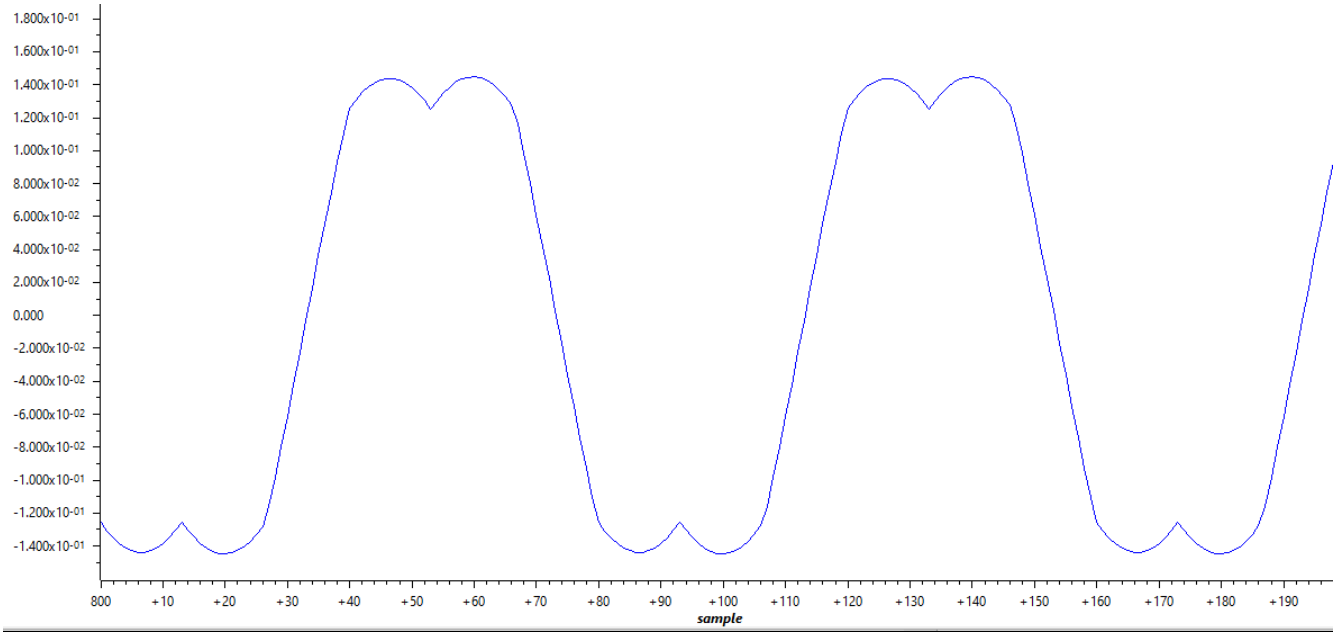

Figure 3-19. Phase A Duty Cycle

3.5.3 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 3.4.3. It is plotted in Figure 3-20. 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-02014. Control card details should be located in user guide.

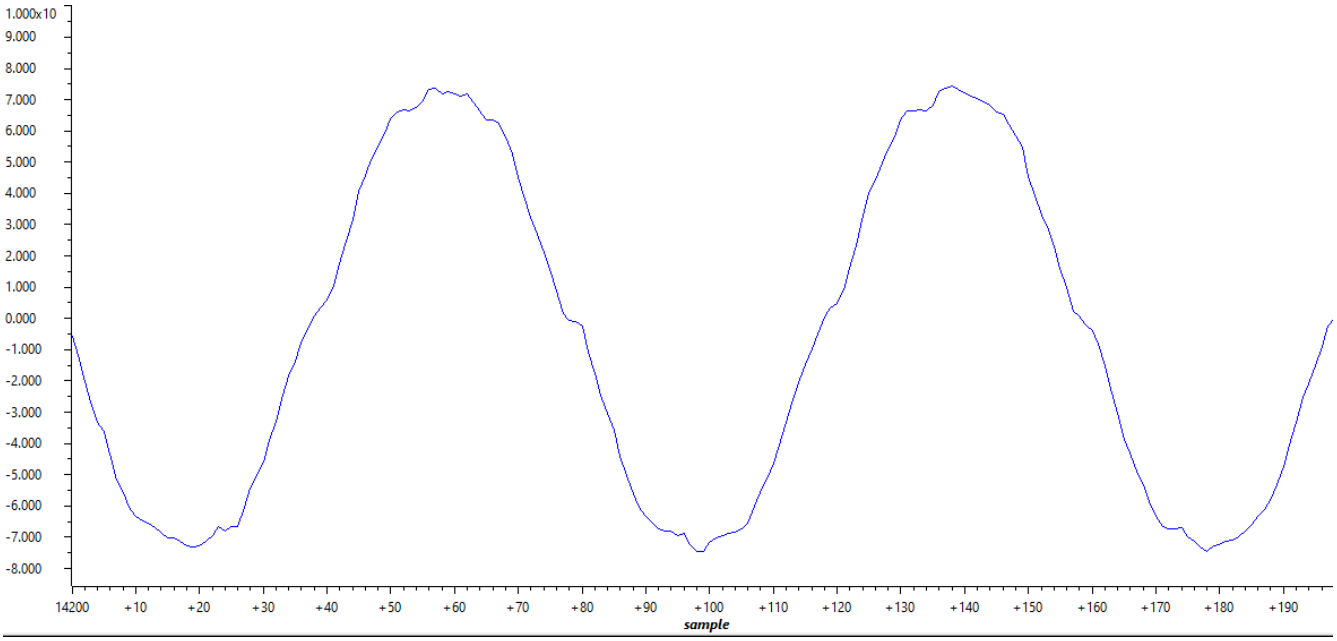

Figure 3-20. Phase A Current Open Loop

3.6 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 3.5.1. Id and Iq references are manually assigned.

3.6.1 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-02014.

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 3.4.3. 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 3-21 and Figure 3-22.

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

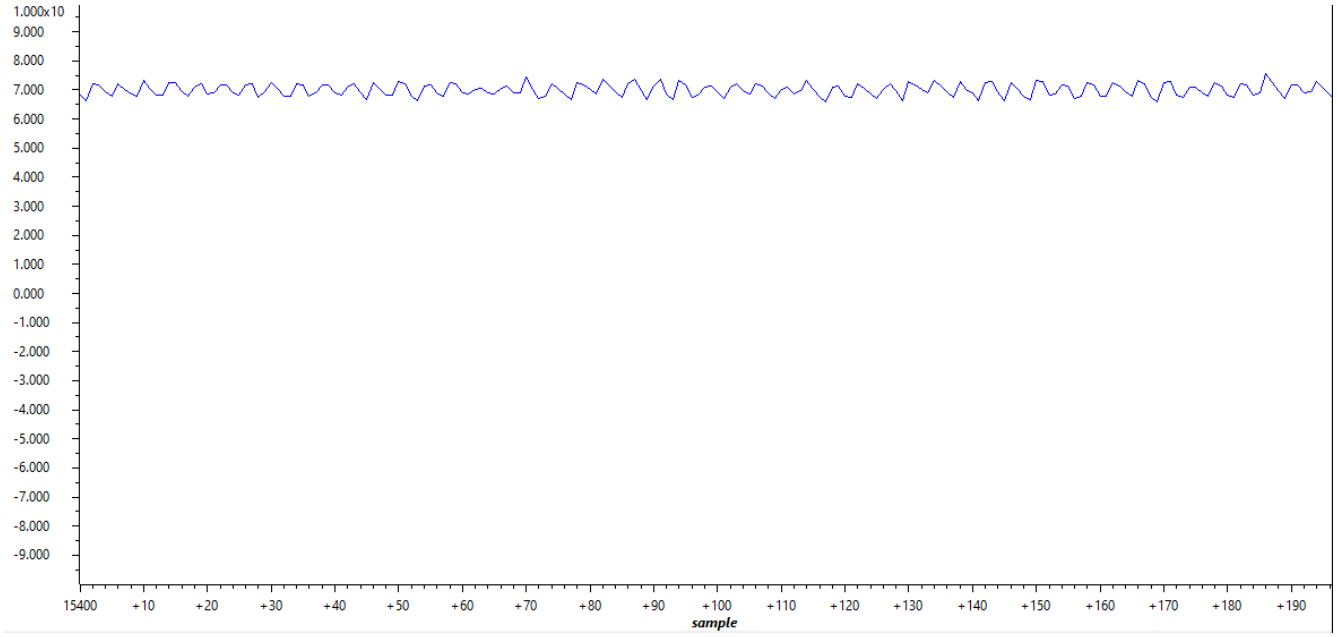

Figure 3-21. Open Loop Id

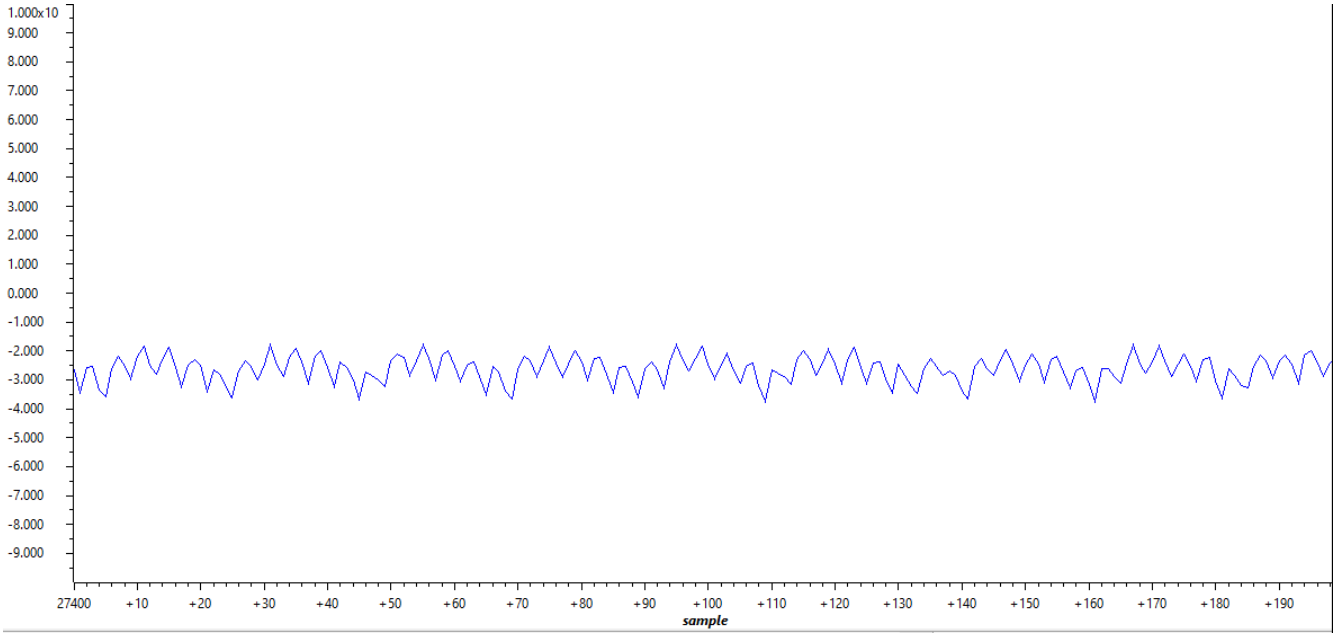

Figure 3-22. Open Loop Iq

3.6.2 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-02014. 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));

3.6.3 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 3.4.3. 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 3-23 and Figure 3-24.

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

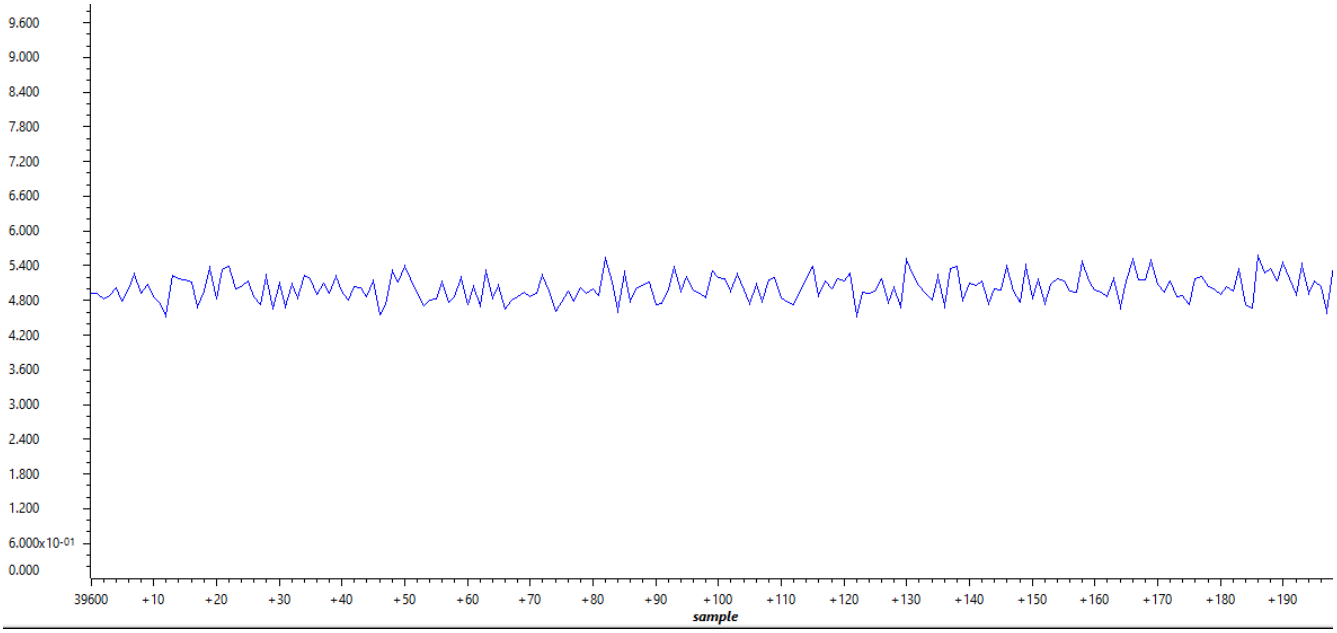

Figure 3-23. Closed Loop Id

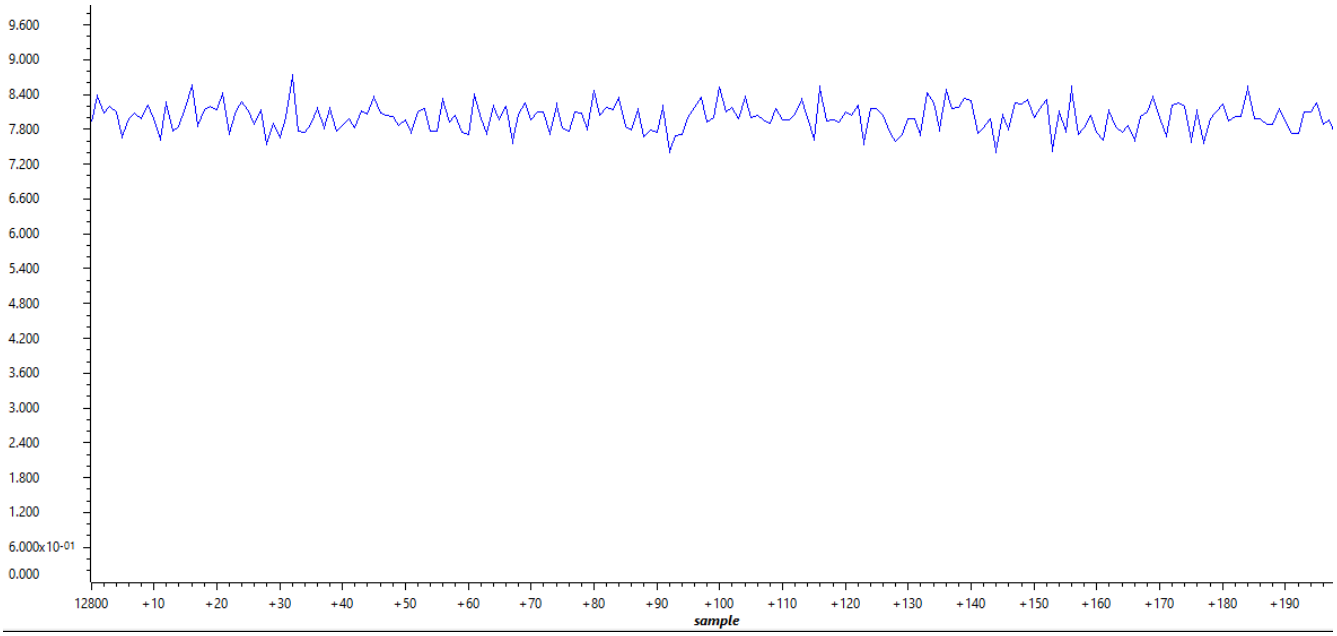

Figure 3-24. Closed Loop Iq

3.7 Add Software Resolver to Digital Converter

The software resolver to digital converter is implemented by sending excitation signal via DAC and DMA, and read sin and cos feedback with ADC. An overview of the configuration is summarized in Figure 3-25. In this work, PWM X is EPWM0 and PWM Y is EPWM7. EPWM0 at 10 kHz is the one triggering phase A power switches and the source of synchronization. EPWM7 at 20 kHz 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.

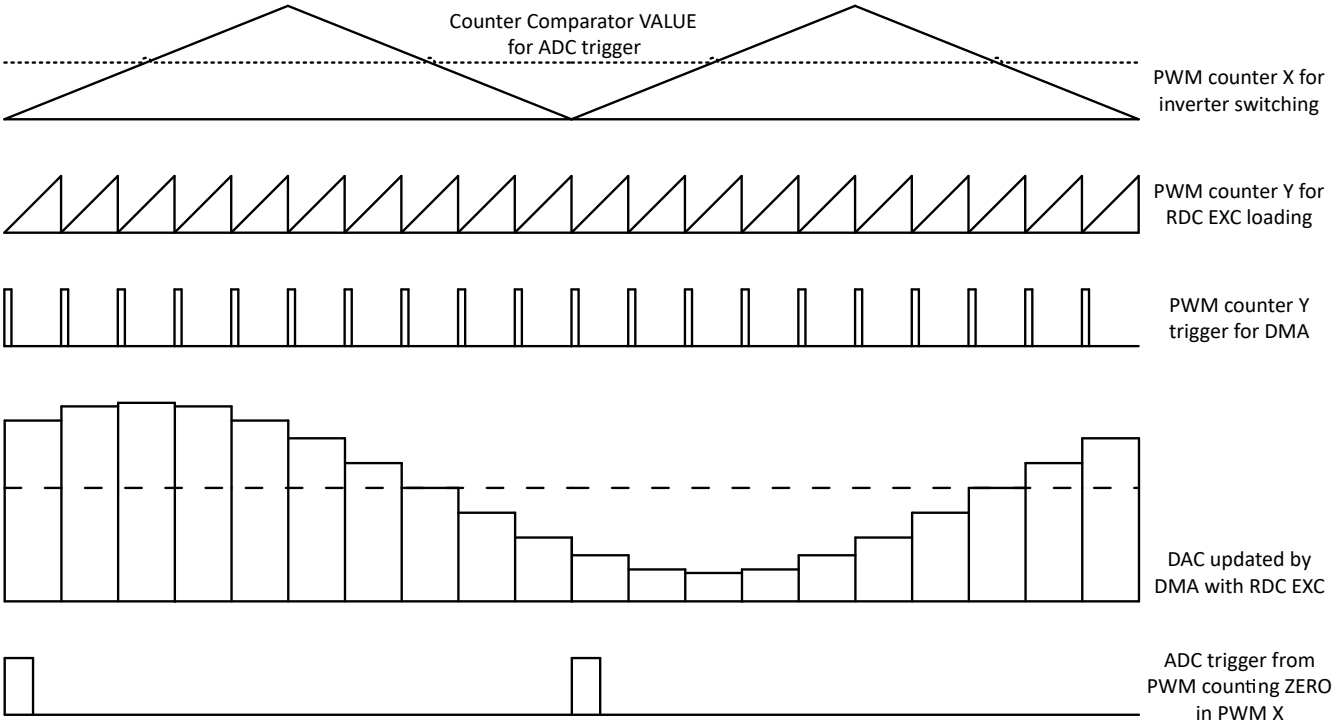

Figure 3-25. Software Resolver Synchronization and Excitation

3.7.1 Generate Excitation for Resolver Hardware

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 needs to be called before control loop running, and RDCexc_update can 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.

3.7.2 Add Resolver Software

The software resolver is implemented similar to C28 in TIDM-02014. 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);

Note 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);

3.7.3 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 3.4.3.

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

4 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.

4.1 SDK Resources Overview

A overview of SDK resources is summarized in Table 4-1. 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 4-1. 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

4.2 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 4-2. 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 3.

Table 4-2. 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)

Also, despite the similarities, there are some difference in SDK and some implementations with similar names. As shown in Section 4.1, 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.

4.3 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.

Also note 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 800 MHz while AM263x R5F cores are 400 MHz. 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.

5 Summary

As the heart of the electric vehicle, the traction inverter is a system requiring high CPU throughput and flexibility of control peripherals. This document demonstrate how the AM263x MCU can fit into the architecture of a traction inverter reference design with its hardware and software framework. Then the software set-up to use AM263x in TIDM-02014 is discussed in detail. The procedure to test the TIDM-02014 reference design with

the TMDSCNCD263 control card is presented. This document is a further addition to the efforts to reduce the learning curve and accelerate the adoption of AM263x in traction systems.

6 References

- Texas Instruments, *ASIL D Safety Concept-Assessed High-Speed Traction, Bi-directional DC/DC Conversion Reference Design*.
- Texas Instruments, *AM263x Sitara™ Microcontrollers*, data sheet.
- Texas Instruments, *AM263x Sitara Processors Technical Reference Manual*.
- Texas Instruments, *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