TI Microcontroller TMS320F28379D - Setup and Use Guide

Last Edit: 04/17/2022

Contributors:
Dylan Gaub
Jake Dorsett

Abstract

The TI Microcontroller TMS320F28379D is a powerful 32-bit floating-point microcontroller unit (MCU) designed for advanced closed-loop control applications such as industrial motor drives, solar inverters and digital power, electrical vehicles and transportation, and sensing and signal processing.

For our project, we are using this TI Microcontroller to create a controller-hardware-in the-loop (CHIL) experiment with an OPAL-RT system. Throughout this document, we will explain the process of setting up the microcontroller in both hardware and software, and how we accomplished our goals by interfacing with this board.

Table of Contents

Section 1.0 - Equipment Requirements

Hardware:

- TI Microcontroller – TMS320F28379D
- One USB Micro cable and one USB Mini-B cable **or** one USB Micro cable and one 5V DC power adapter **or** Two USB Mini-B cables
- Operational Computer with at least 2 USB ports

Optional Hardware:

- Digilent Analog Discovery 2 (for testing and debugging)
- Jumper Cables

Software:

- Code Composter Studio – Version 10 or higher

Optional Software:

- Digilent WaveForms

Note: This document is under the assumption that the user has a functional Windows OS and machine. Steps for Mac and Linux will not be documented at this time.

Section 1.1 - Initial Hardware Setup

The process of setting up the TI Microcontroller can be done by using one of the following options:

1. One USB Micro (1) and one USB Mini-B (2) connector

or

2. One USB Micro (1) and one DC power adapter (5V) (3)

or

3. Two USB Mini-B cables (2) ((3) Next to the DC Power Jack on the Dev Board)

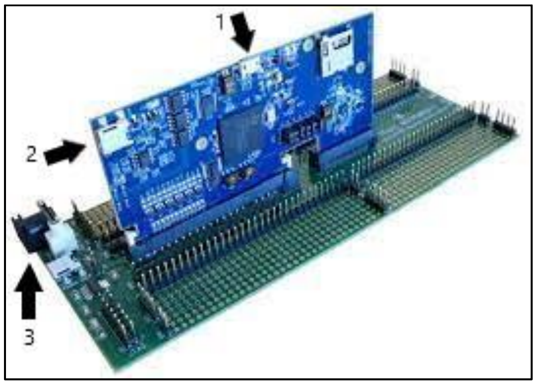


Figure 1: TI Microcontroller (Vertical) and Development Board (Horizontal)

As shown in Figure 1, we can see the proper connection of the TI Microcontroller and the Development Board. By carefully connecting the two boards, and latching the two tiny, white latches into place, the board is ready to be plugged in.

Taking the cables that you have chosen for your setup, follow the same numbers that correspond from your choice above.

1. Plug the USB Micro into the top of the TI Microcontroller (Labeled as 1 in Figure 1). Next, plug the USB Mini-B into the side of the TI Microcontroller (Labeled as 2 in Figure 1). After these connections are established, you should see two red LED light up on the microcontroller. The steps are the same for 3, except instead of the USB micro, you will plug the second USB Mini-B into the correct side on the Dev Board next to the DC Power Jack (Labeled as 3).

or

2. Plug the USB Micro into the top of the TI Microcontroller (Labeled as 1 in Figure 1). Next, plug the DC Power adapter in the side of the development board into the barrel connector (Labeled as 3 in Figure 1). Then flip the switch next to the barrel connector so that the LED on the development board turns on. After these connections are established, you should see two red LED light up on the microcontroller and a green LED light up on the development board.

Taking the ends of the cables you have decided to use (The DC Power Jack gets plugged into a wall outlet) and the USB Micro/Mini-B will be plugged into the USB ports on your machine.

If your connections have been made properly, you should see the appropriate LED indicators as mentioned per setup technique. If you are not seeing the correct LEDs light up, check that your cables are properly connected to your computer's USB ports and/or AC power source.

Section 1.2 - Initial Software Setup

The first step in setting up your software is to download Code Composer Studio. The download for the CCS IDE can be found here: https://www.ti.com/tool/CCSTUDIO

After navigating to the link above, click on **Download options** and download the proper OS and version.

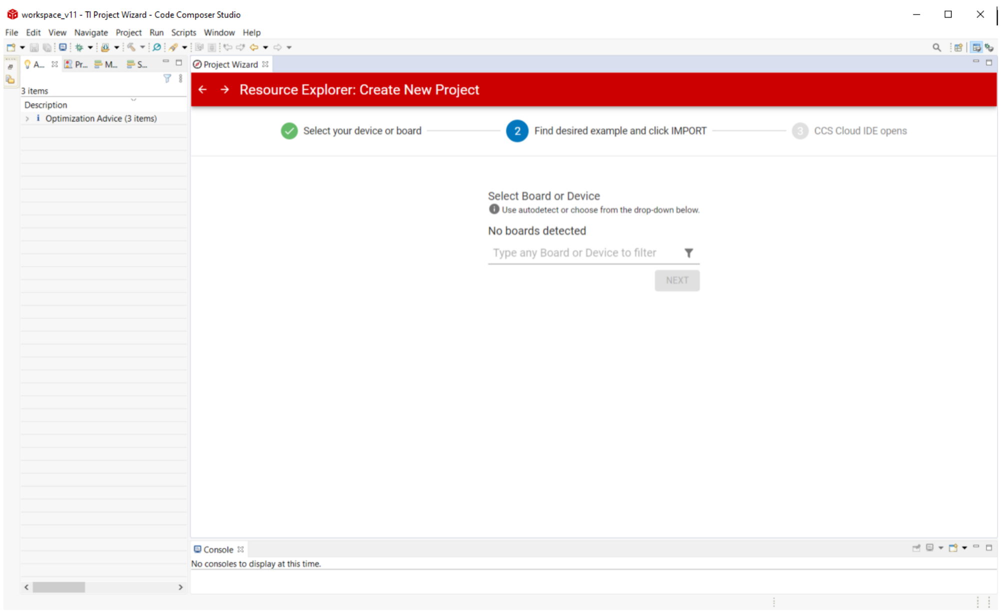
Note: Please keep in mind that CCS is a large program, so be sure that you have the space for it. The download may take a little while depending on your internet speed.

Follow the installation process and keep in mind important settings and options to enable during the installation:

- When the 'setup' window should appears click on 'next' follow the prompts and then select 'I accept the agreement'.
- Upon the next page, leave the settings as the default for the installation directory and use 'custom installation' for the setup type, then click next.
- Components to select 'SimpleLink MSP432 low power + performance MCUs' and 'C2000 real-time MCUs'
- The next option titled 'install debug probes' select the 'Spectrum Digital Debug Probes and Boards' option

Upon installation of CCS, launch the newly installed program.

After CCS has launched, you will come to a screen that looks like the following:


workspace_v11 - TI Project Wizard - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Wizard
Resource Explorer: Create New Project
Select your device or board
2 Find desired example and click IMPORT
3 CCS Cloud IDE opens
Select Board or Device
Type any Board or Device to filter
NEXT
Console
3 items
Description
Optimization Advice (3 items)

Use autodetect or choose from the drop-down below.

No boards detected

No consoles to display at this time.

Figure 2: Beginning screen on Code Composer Studio

Section 1.3 – Setting Up Code Composer Studio

Now that CCS is up and running, it is important to get the correct configurations to enable communication between Code Composer Studio and the TMS320F28379D.

You can simply set up an example project or follow the example files from the one-day workshop listed in the bottom sub note of this section. Refer to Figure 2 on Page 5 to search for an example project that is explained next.

To correctly bind the TI board with CCS for an example blinky led project, click on View → Getting Started, then click on Browse and Import Examples.

Next enter the following for the board type: LAUNCHXL F28379D

Then click Next

Now, you will be brought to a screen to find a piece of code you would like to execute, for our example we will use the "blinky_dc_cpu1", then click Import

Next, go to your project directory and click on the new imported project within the workspace.

At the top of your screen, you will see a hammer icon that looks like this: Click here and wait for your project to build and compile.

Upon a successful build, you can now enter debug mode, click on the bug icon that looks like this:

After your debug session has launched, you will be brought to the debug screen. Here, you can press the play button: Upon doing so your program will execute. In our case for the blinky program, our LED now blinks on the microcontroller.

When you are done executing a program, hit the stop button: After clicking on this button, you will be brought back to the initial CCS coding screen.

Note: Any of these steps can be clarified and followed by the setup tutorial created by TI which is found here: TI One Day Workshop

- See Pages 22-23 in the C2000 MCU 1-Day Workshop for more details on program execution

It is highly recommended to use this workshop link and follow the tutorials and labs to gain familiarity of how to interface with the TI.

Section 1.4 – Cleaning and restarting workspaces

In some rare occurrences, CCS will have issues building/debugging your code. Here are some provided solutions to try first if you are running into any issues related to your CCS throwing errors/having issues with building.

Note: This debugging process assumes that the initial code was building correctly, and now all the sudden has stopped. Sever issues may happen in the microcontroller's memory/cache.

It is good practice to do all these steps to begin your debugging process:

1. Completely close Code Composer Studio
2. Unplug devices and plug them back in
3. Relaunch Code Composer Studio
4. Clean and rebuild your project

To accomplish step 4 above, right click on the workspace that you are working in and click on 'Clean Project' as shown in the figure below:

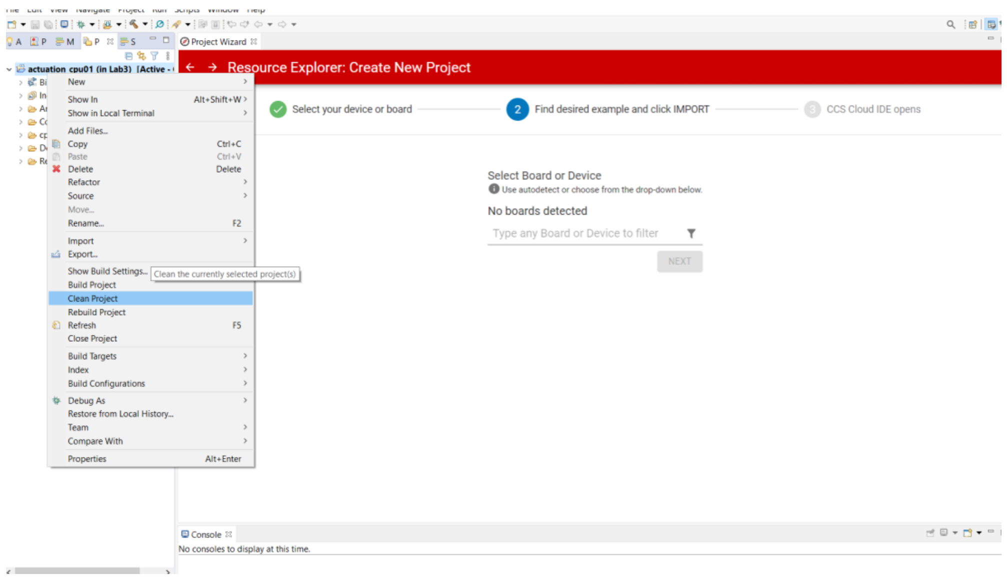


Figure 3: Cleaning a project inside of Code Composer Studio

Another way to clean your workspace is to click on Project → Clean:

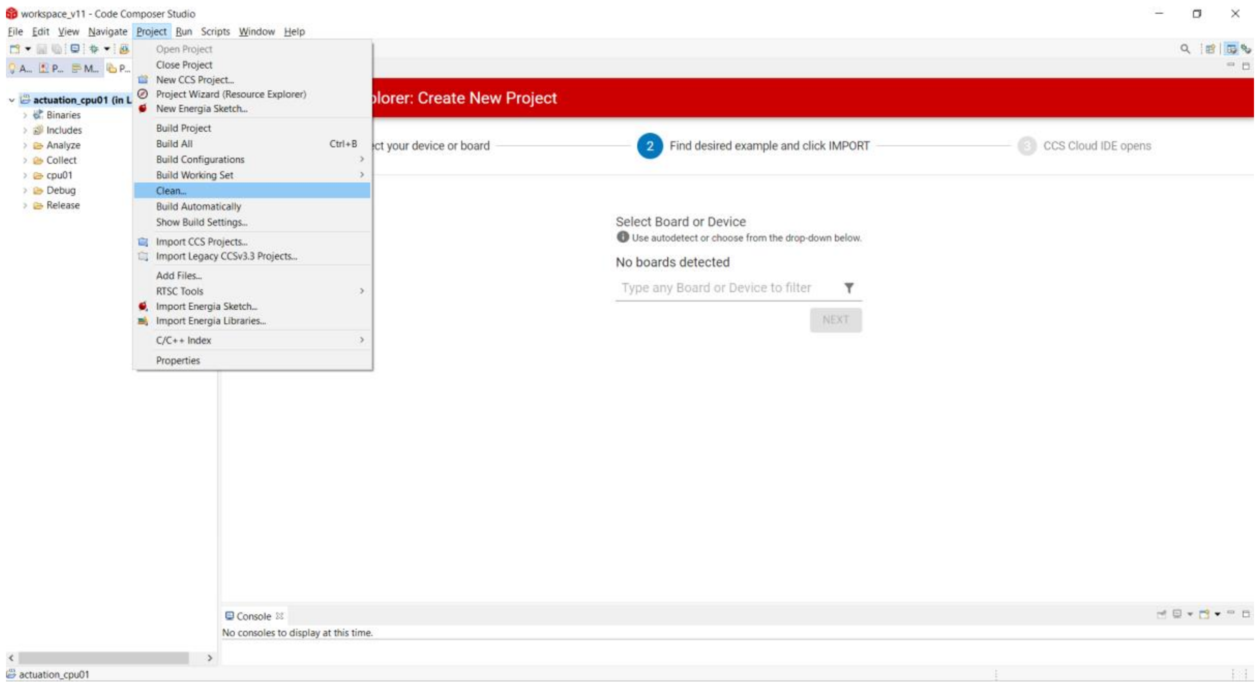
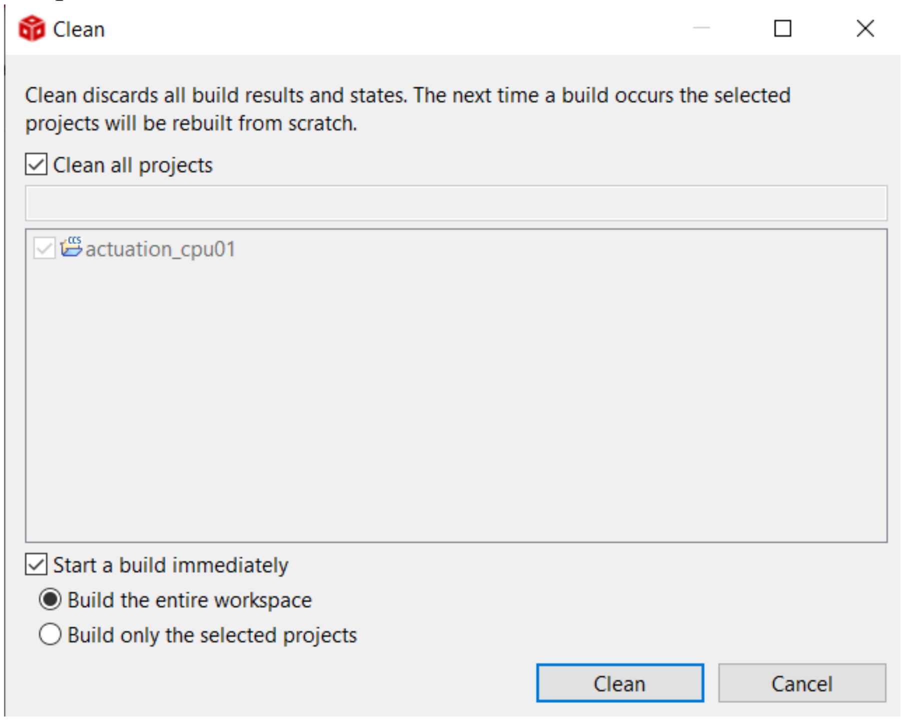


Figure 4: Another way to clean your workspace in Code Composer Studio

Then, when prompted, click on the 'Clean' button:


Clean

Clean discards all build results and states. The next time a build occurs the selected projects will be rebuilt from scratch.

☑ Clean all projects

☑ actuation_cpu01

☑ Start a build immediately

● Build the entire workspace

○ Build only the selected projects

Figure 5: Cleaning project

Note: These solutions are not the end all be all and may not solve all your issues. Be sure to check that there are no errors in your code as well.

Section 2.0 - Input and Output Operations

The TMS320F28379D TI Microcontroller uses the C programming language, so before starting a project with this microcontroller, be sure to gain at least a basic understanding of this programming language.

There are a few different communication options available on this microcontroller, but we will focus on simply acquiring data using analog to digital converters (ADCs) and sending data through digital to analog converters (DACs).

There are up to four 12-bit single-ended ADCs available, with up to 24 channels total, and three 12-bit buffered DACs. In this section we will discuss how to set up these systems in Code Composer and what to expect when running your code.

You can use your Analog Discover Kit with the Waveforms Software to enable I/O interaction with your microcontroller. Here you can enable a wave generation into the correct receiving pin on the dev board.

Setting up the ADC:

Once you have written the initial program setup (variables, headers, includes, etc.), we need to declare a function so that when your program is run, the ADC will be configured. This is done by writing the following code:

void ConfigureADC(void);   // Write ADC configurations and power up the ADC

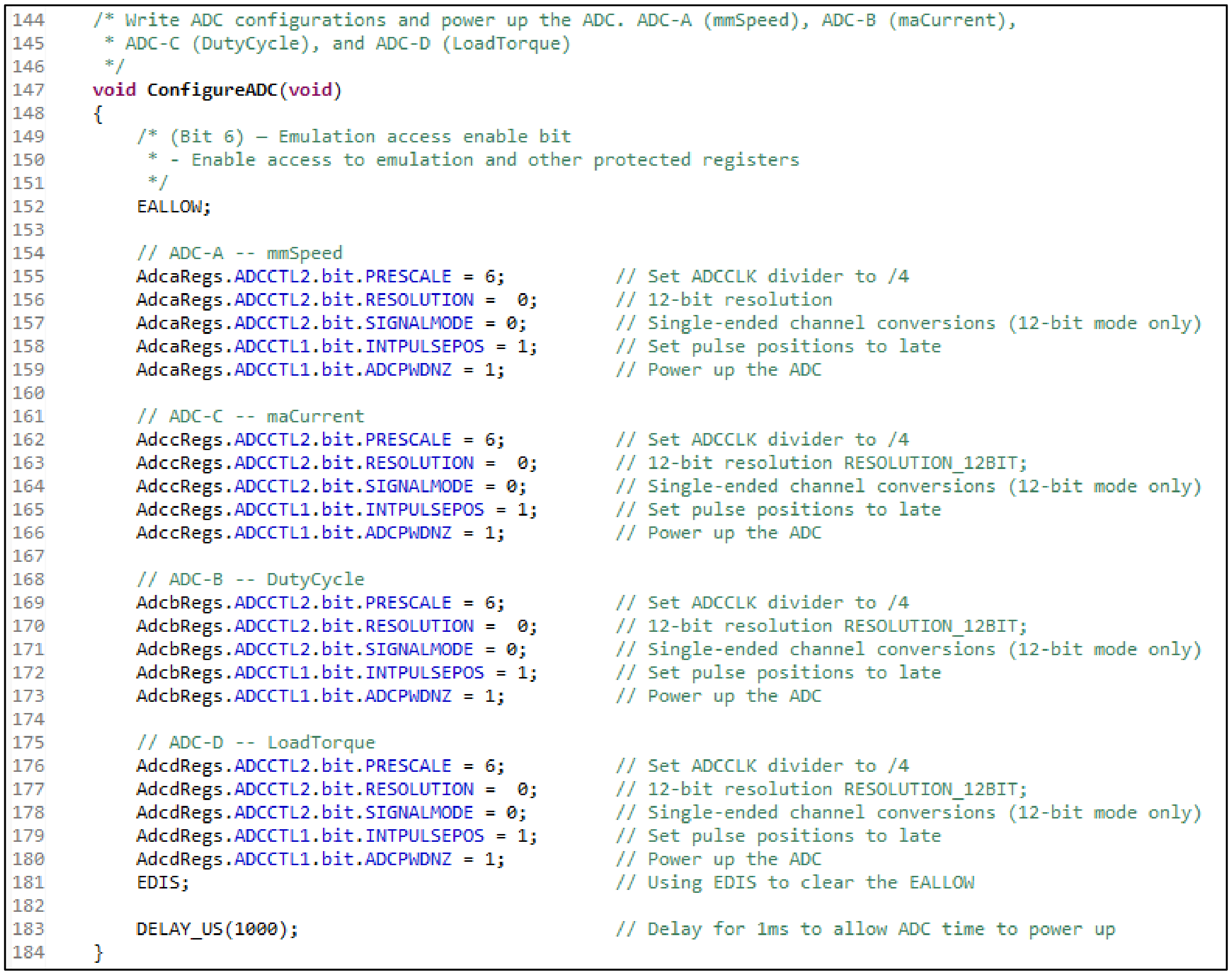
This line defines a function for the complier to recognize in the code. When the program gets to an instance of this function, the function is called, and the ADC is setup. As with most programming languages, this line of code needs to be located outside your main program, where other functions and variables are defined.

Inside your main program, typically where your program setup is located is where you will need to have a line calling this function. This line will look like this:

ConfigureADC();   // Configure the ADC and power it up

This line of code will jump the program to the ConfigureADC function. Inside this function is where you will set up which ADC will be used, how many will be used, and how they will convert the incoming data.

Below is an example of a ConfigureADC function:



```
/* Write ADC configurations and power up the ADC. ADC-A (mmSpeed), ADC-B (maCurrent),
 * ADC-C (DutyCycle), and ADC-D (LoadTorque)
 */
void ConfigureADC(void)
{
    /* (Bit 6) – Emulation access enable bit
     * - Enable access to emulation and other protected registers
     */
    EALLOW;

    // ADC-A -- mmSpeed
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6;          // Set ADCCLK divider to /4
    AdcaRegs.ADCCTL2.bit.RESOLUTION =  0;       // 12-bit resolution
    AdcaRegs.ADCCTL2.bit.SIGNALMODE = 0;        // Single-ended channel conversions (12-bit mode only)
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;       // Set pulse positions to late
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // Power up the ADC

    // ADC-C -- maCurrent
    AdccRegs.ADCCTL2.bit.PRESCALE = 6;          // Set ADCCLK divider to /4
    AdccRegs.ADCCTL2.bit.RESOLUTION =  0;       // 12-bit resolution RESOLUTION_12BIT;
    AdccRegs.ADCCTL2.bit.SIGNALMODE = 0;        // Single-ended channel conversions (12-bit mode only)
    AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;       // Set pulse positions to late
    AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // Power up the ADC

    // ADC-B -- DutyCycle
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6;          // Set ADCCLK divider to /4
    AdcbRegs.ADCCTL2.bit.RESOLUTION =  0;       // 12-bit resolution RESOLUTION_12BIT;
    AdcbRegs.ADCCTL2.bit.SIGNALMODE = 0;        // Single-ended channel conversions (12-bit mode only)
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;       // Set pulse positions to late
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // Power up the ADC

    // ADC-D -- LoadTorque
    AdcdRegs.ADCCTL2.bit.PRESCALE = 6;          // Set ADCCLK divider to /4
    AdcdRegs.ADCCTL2.bit.RESOLUTION =  0;       // 12-bit resolution RESOLUTION_12BIT;
    AdcdRegs.ADCCTL2.bit.SIGNALMODE = 0;        // Single-ended channel conversions (12-bit mode only)
    AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;       // Set pulse positions to late
    AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // Power up the ADC
    EDIS;                                       // Using EDIS to clear the EALLOW

    DELAY_US(1000);                             // Delay for 1ms to allow ADC time to power up
}
```

Figure 6: ConfigureADC function

When the function is called in the main program, the program jumps to the function instance; in this case it jumps to line 147 in our program. Next, we need to allow the registers to be written to and our ADC to be configured. In this example, we are setting up four ADCs, ADC-A through ADC-D. Writing to one of the ADC registers looks like what you see in line 155. In this line we are writing one bit (.bit) to the ADC-A (AdcaRegs.) Control 2 Register (.ADCCTL2), configuring the ADC clock prescaler value (.PRESCALE). By setting this register bit value to 6, we are telling the program to downscale the ADC clock by ¼ the input clock. The input clock in this case is 200MHz, so our ADC clock will be at 50MHz.

The next line is setting the resolution of the ADC to be 12-bit, whereas the following line is setting the signal mode to single-ended. On line 158, INTPULSEPOS is setting the ADC interrupt pulse position to be delayed slightly. This is essentially synchronizing the timing of the interrupt pulse to the latching of the ADC result. Finally, we simply power up the ADC with the command seen in line 159.

If your application requires identical ADCs like this one, simply follow the same setup scheme seen in lines 155 through 159 but be sure to write to a new ADC by changing the ADC identifier (Adc*Regs). Otherwise, the first step in your ADC setup is almost done.

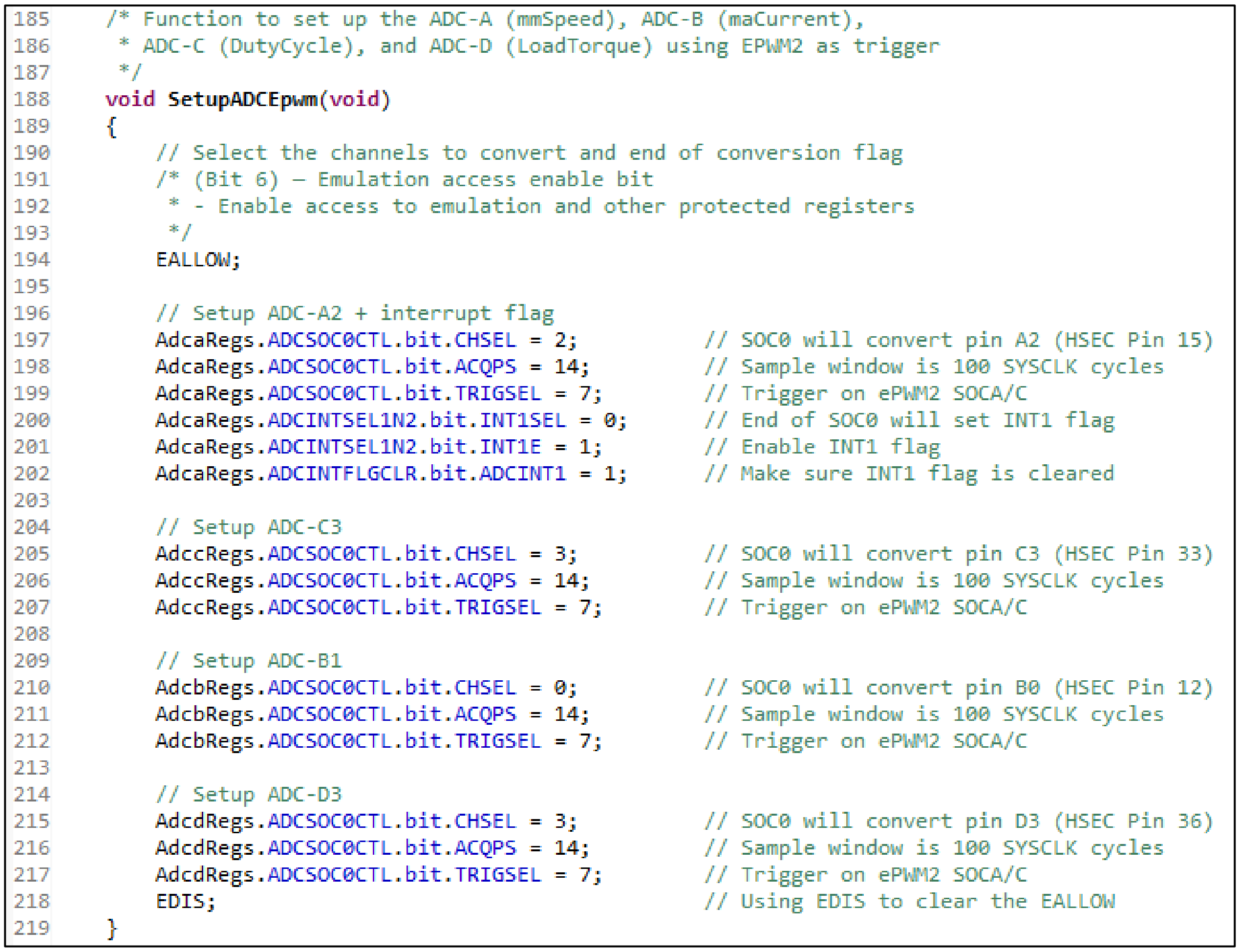
End the ConfigureADC function by writing the command "EDIS" followed by a short delay to allow the ADC(s) to power up.

Next, we need to declare another function that sets up the remainder of the ADCs. Just like before, declare a function alongside the previous function as so:

void SetupADCEpwm(void);   // Select the channels to convert and end of conversion flag for the Pulse Width Modulator

There are several ways to setup ADC triggers on this microcontroller, but we were able to get good results using EPWM, so will continue this tutorial using this scheme.

Below is an example of a SetupADCEpwm function:



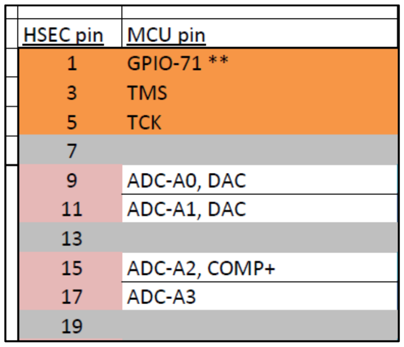
```
185   /* Function to set up the ADC-A (mmSpeed), ADC-B (maCurrent),
186    * ADC-C (DutyCycle), and ADC-D (LoadTorque) using EPWM2 as trigger
187    */
188   void SetupADCEpwm(void)
189   {
190       // Select the channels to convert and end of conversion flag
191       /* (Bit 6) – Emulation access enable bit
192        * - Enable access to emulation and other protected registers
193        */
194       EALLOW;
195
196       // Setup ADC-A2 + interrupt flag
197       AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2;          // SOC0 will convert pin A2 (HSEC Pin 15)
198       AdcaRegs.ADCSOC0CTL.bit.ACQPS = 14;         // Sample window is 100 SYSCLK cycles
199       AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 7;        // Trigger on ePWM2 SOCA/C
200       AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0;      // End of SOC0 will set INT1 flag
201       AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;        // Enable INT1 flag
202       AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;      // Make sure INT1 flag is cleared
203
204       // Setup ADC-C3
205       AdccRegs.ADCSOC0CTL.bit.CHSEL = 3;          // SOC0 will convert pin C3 (HSEC Pin 33)
206       AdccRegs.ADCSOC0CTL.bit.ACQPS = 14;         // Sample window is 100 SYSCLK cycles
207       AdccRegs.ADCSOC0CTL.bit.TRIGSEL = 7;        // Trigger on ePWM2 SOCA/C
208
209       // Setup ADC-B1
210       AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0;          // SOC0 will convert pin B0 (HSEC Pin 12)
211       AdcbRegs.ADCSOC0CTL.bit.ACQPS = 14;         // Sample window is 100 SYSCLK cycles
212       AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 7;        // Trigger on ePWM2 SOCA/C
213
214       // Setup ADC-D3
215       AdcdRegs.ADCSOC0CTL.bit.CHSEL = 3;          // SOC0 will convert pin D3 (HSEC Pin 36)
216       AdcdRegs.ADCSOC0CTL.bit.ACQPS = 14;         // Sample window is 100 SYSCLK cycles
217       AdcdRegs.ADCSOC0CTL.bit.TRIGSEL = 7;        // Trigger on ePWM2 SOCA/C
218       EDIS;                                       // Using EDIS to clear the EALLOW
219   }
```

Figure 7: SetupADCEpwm function

As with the previous function, we need to allow writing to these registers by using the command EALLOW.

Now we need to configure each ADC set up in the ConfigureADC function. By first writing to the ADC start of conversion 0 control register (.ADCSOC0CTL), we can select which channel (.CHSEL) of the chosen ADC we want to use. Line 197 in Figure 3 configures ADC-A to convert data on channel 2.

Note: when deciding which ADC channel, you want to configure, be sure to refer to the TMS320F28379D pinout to determine the best pin location for your application. Figure 4 on the next page shows a small section of the TMS320F28379D pinout. For example, ADC-A0 is ADC-A channel 0 and is located at HSEC pin 9 on the development board.



HSEC pin	MCU pin
1	GPIO-71 **
3	TMS
5	TCK
7	
9	ADC-A0, DAC
11	ADC-A1, DAC
13	
15	ADC-A2, COMP+
17	ADC-A3
19	

Figure 8: TMS320F28379D pinout example

The value written to the ACQPS bit in the SOC 0 Control register determines the sample and hold duration for the ADC.

Conversion time = 10.5*ADCCLK cycles = 10.5*(SYSCLK / 4) cycles = 10.5*20ns = 210ns
Sample and hold duration = ACQPS + 1*SYSCLK period = 15*5ns = 75ns

Next, the TRIGSEL bit in the example above was set to 7, which configures the ADC start conversion based on EPWM2 (yet to be set up). The following three lines then configure the ADC to set an interrupt flag when the conversion is finished (end of conversion – EOC), then ensure the flag is set to low.

We now have one ADC configured, along with its triggering mechanism. In this example, we needed three more ADC, so we simply duplicated this code above but changed the ADC we wanted to use (Adc*Regs), and the associated channel (.CHSEL). Since the interrupt portion was configured in lines 200-202, we do not need to reconfigure this per each ADC configured.

The code above essentially is an interrupt for the microcontroller to grab the correct values in real time and store them in the appropriate registers.

Setting up the DAC:

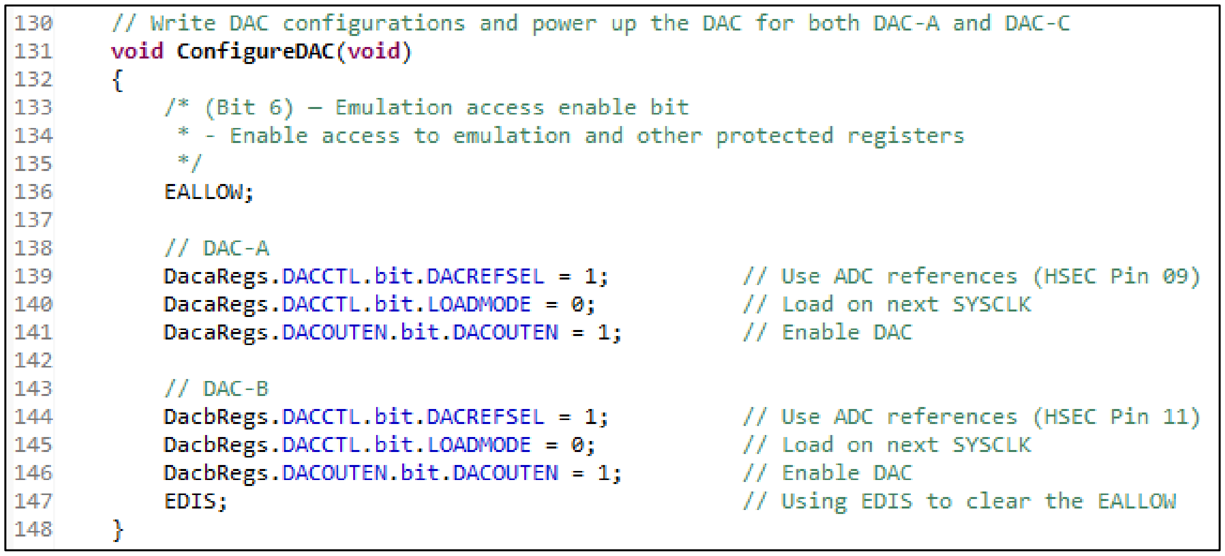
If your application requires data to be sent from the microcontroller in analog form, we need to set up and configure the DAC(s). Luckily this is a much more straightforward process than it was for the ADC.

First, like when setting up the ADC, we need to declare a function like this one:

void ConfigureDAC(void);   // Write DAC configurations and power up the both DAC A and DAC B

When the ConfigureDAC function is called, the program will jump to the ConfigureDAC function and set up the DAC(s). There are apparently three DACs located on the TMS320F28379D microcontroller, but on the pinout, we have only DAC A and DAC B located on developer board pins 9 and 11 respectively. Read through the TI microcontroller's technical reference manual if more than two DACs are needed.

Below is an example of setting up both DAC-A and DAC-B:



```
// Write DAC configurations and power up the DAC for both DAC-A and DAC-C
void ConfigureDAC(void)
{
    /* (Bit 6) — Emulation access enable bit
     * - Enable access to emulation and other protected registers
     */
    EALLOW;

    // DAC-A
    DacaRegs.DACCTL.bit.DACREFSEL = 1;          // Use ADC references (HSEC Pin 09)
    DacaRegs.DACCTL.bit.LOADMODE = 0;           // Load on next SYSCLK
    DacaRegs.DACOUTEN.bit.DACOUTEN = 1;         // Enable DAC

    // DAC-B
    DacbRegs.DACCTL.bit.DACREFSEL = 1;          // Use ADC references (HSEC Pin 11)
    DacbRegs.DACCTL.bit.LOADMODE = 0;           // Load on next SYSCLK
    DacbRegs.DACOUTEN.bit.DACOUTEN = 1;         // Enable DAC
    EDIS;                                       // Using EDIS to clear the EALLOW
}
```

Figure 9: ConfigureDAC function

First note from the pinout (Figure 4) that DAC-A is located on HSEC pin 9 and DAC-B on HSEC pin 11.

To set up the DAC, we first need to write to the DAC control register (.DACCTL) configuring the DAC reference voltage selection (.DACREFSEL). In this case we wanted to set the ADC VREFHI/VSSA as our reference voltages. Setting this to 0 sets the reference to VDAC/VSSA.

Next, we need to set when the DACVALA (active) register is loaded with a value from DACVALS (shadow). Setting this to 0 loads the value on next SYSCLK, whereas a 1 will set this to load on EPWMSYNCPER specified by SYNCSEL. SYNCSEL is another portion of the DAC control register which EPWMSYNCPER signal will update DACVALA. For simplicity, we will just ignore this and simply update DACVALA every SYSCLK cycle.

Lastly, we need to enable the DAC. If an additional DAC is needed, simply duplicate the setup but for DAC-B this time.

Note: the value in volts one should expect to see coming out of the DAC configured pin should follow closely to this formula found in the TI Technical Reference Manual:

$$\text{DACOUT} = (\text{DACVALA} * \text{DACREF}) / 4096$$

*ex: DACOUT = (2048 * 3V) / 4096 = 1.5V*

Section 2.1 – Graphing Outputs

To graph the outputs on Code Composer Studio, your program must be in the running state. Please see Section 1.3 if your project is not in this current state.

During the execution phase of your program (In debug mode), you can watch your values update real time in a graph: Click on Project → Graph → Single Time.

Here, you are given several options to decide how your graph will be displayed, and it is key to select the variable/register location where data is being stored.

You can save these graph settings by clicking on the 'Export' button, which is super helpful if you are continually graphing outputs.

Note: See Pages 60 in the C2000 MCU 1-Day Workshop for more details on graphing.

Section 3.0 – OPAL Use and Integration

To start up the opal, click on the 'Power' button with the red button next to it:

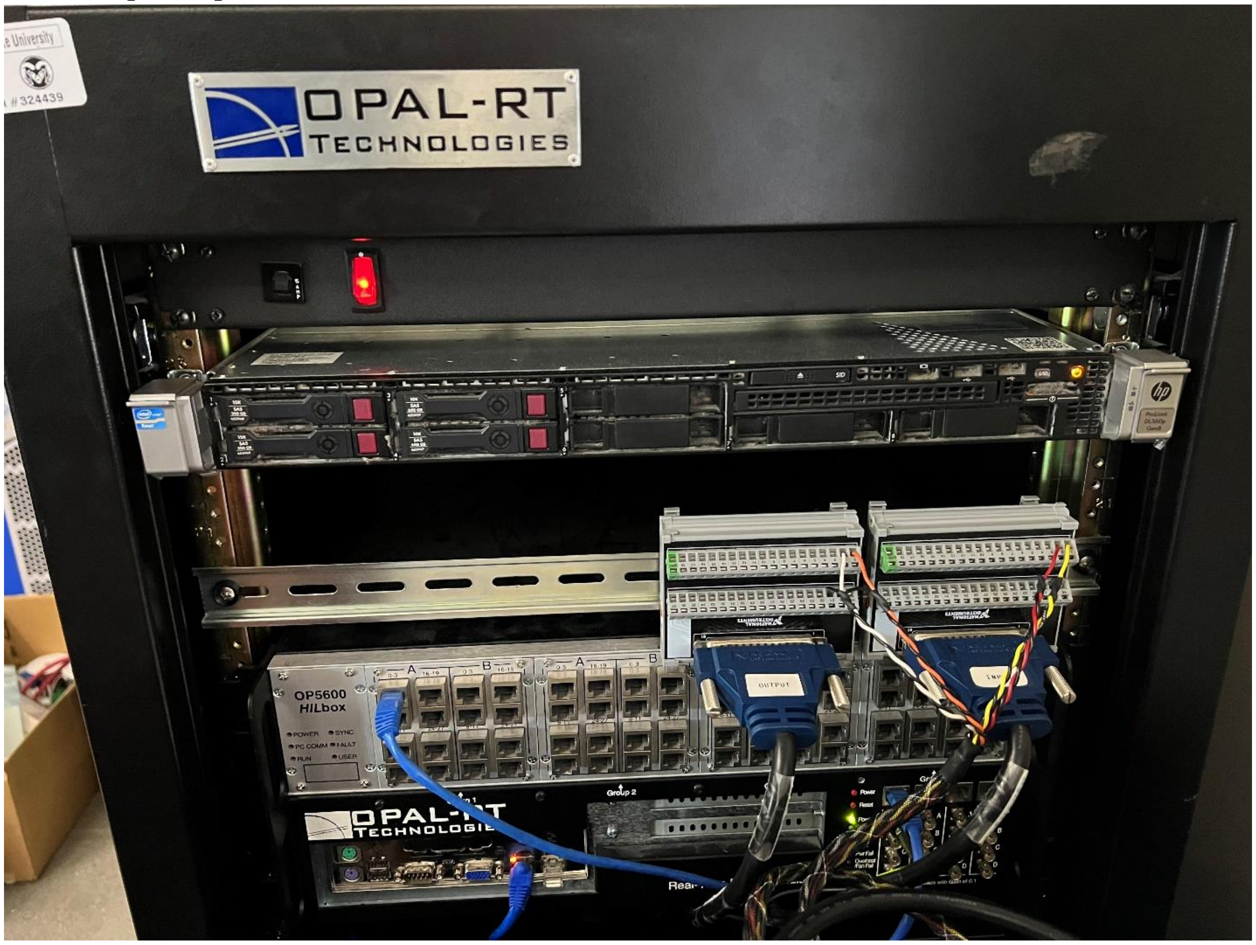


Figure 10: Picture of the OPAL RT (Power button in the lower right-hand side)

This will take upwards to 10-20 minutes for the OPAL to begin.
Note: As of 4/11/2022, the OPAL will have a loud fan noise when everything is up and running. Yes, this is normal.

While the OPAL is booting up, log into the connected computer and Open the RT-Lab application. When RT-Lab Opens, open your preferred model, here we like to use the 'DC Machine' model. After clicking on the workspace, you can load the model.

After the model is loading, you can click on the 'Execute' button, now you are ready to interface with the OPAL.

Note: As of 4/11/2022, the OPAL is connected to the Powerhouse ENS lab machine: For specific credentials related to this machine, please contact the owner of the machine: Chris Lute.
Note: For any other questions concerning the OPAL, reach out to Chris Lute.

The OPAL Threshold is 3.3V which is shown in the following Simulink figure:

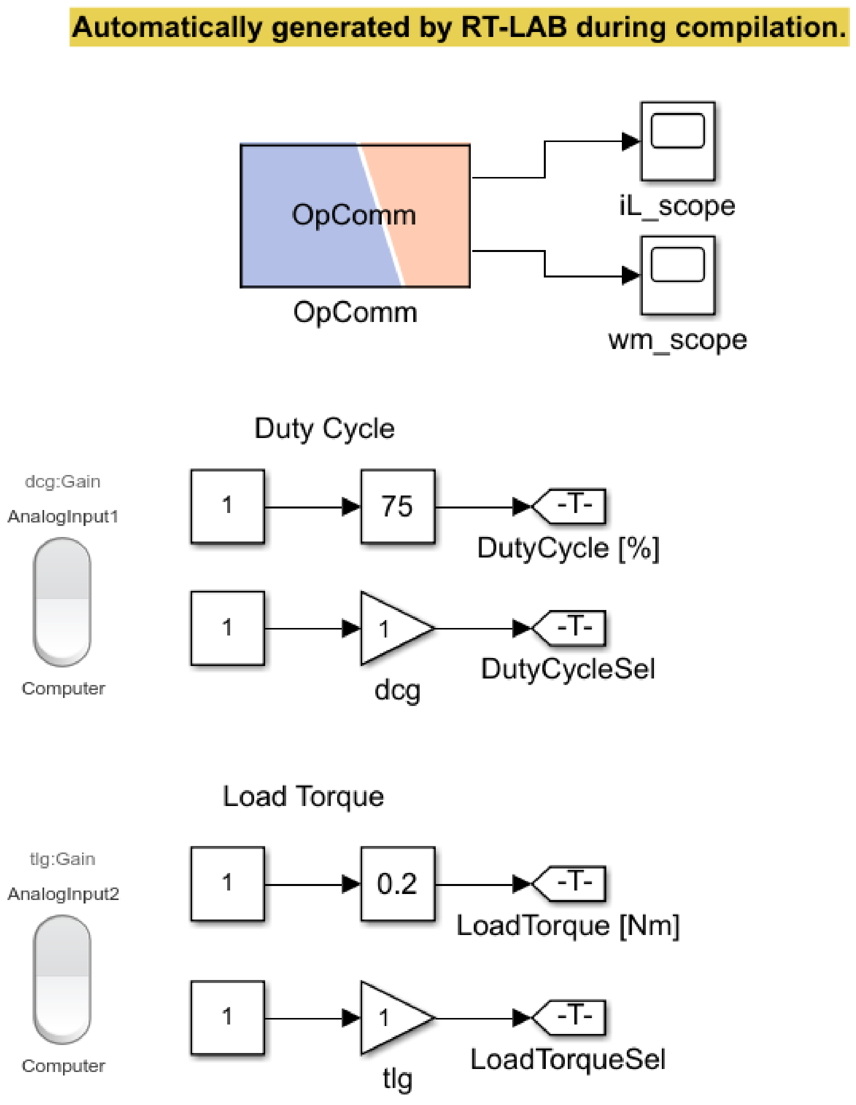


Figure 11: Simulink Model for DC Machine

Section 4.0 – Creating and using your own code within Code Composer Studio

Code Composer Studio is built off an unfamiliar TI library framework. With some experience and Knowledge in C, one can append their own code to the already preconstructed examples code that TI provides.

Section 5.0 – Setting up Code Coverage within Code Composer Studio

Code coverage is an extremely helpful concept that can help developers monitor their code and if it is performing fully and as it should.

Code Composer Studio has a helpful tool already built in that will generate spreadsheets to help you analyze the runtime of your code.

Note: Code coverage is only supported by versions of CCS 15.12-20.12, so a downgrade of versions for CCS will more than likely be required, follow Section 5.1 to downgrade your current version of CCS.

When you have the correct version of Code Composer Studio, Code coverage can easily be set up by following the tutorial from TI in the next note.

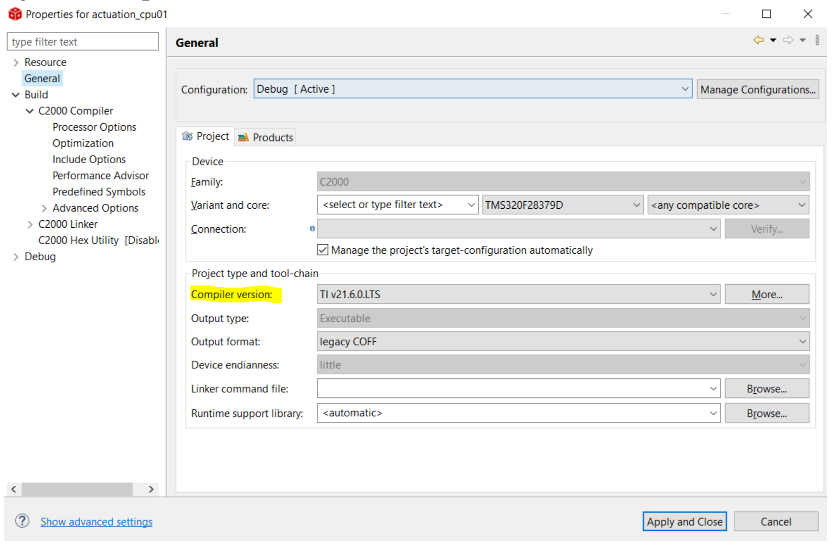
Note: A helpful code coverage tutorial can be found here: Code Coverage with TI Compilers.

Section 5.1 – Downgrading Code Composer Studio version

To downgrade your current Code Composer studio version, go to the link below and download the desired LTS file, and save it to a location that you can easily access within a couple of steps.
Code Composer Versions Link

After your download has completed, open Code Composer Studio and within your project workspace, click on Project → Properties.

Here you can see your compiler version:


Properties for actuation_cpu01

General

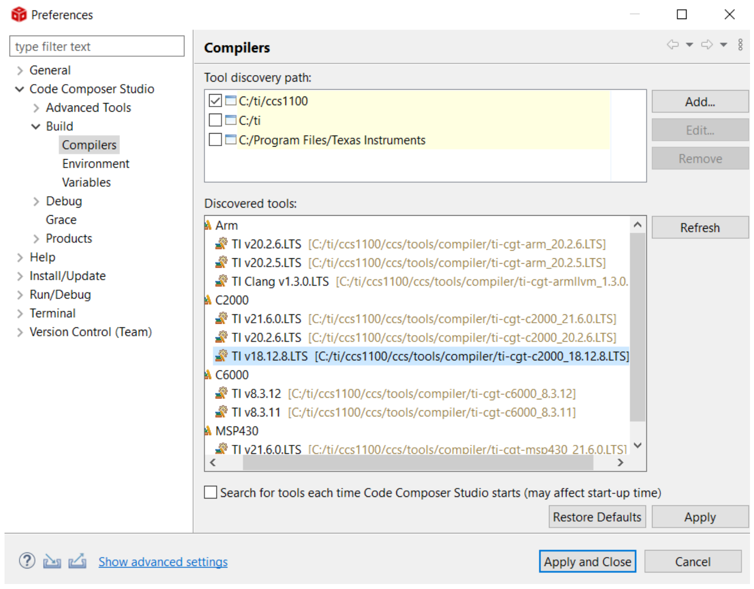
Configuration: Debug [Active]

Device
Family: C2000
Variant and core: <select or type filter text> TMS320F28379D <any compatible core>
Connection:
☑ Manage the project's target-configuration automatically

Project type and tool-chain
Compiler version: TI v21.6.0.LTS
Output type: Executable
Output format: legacy COFF
Device endianness: little
Linker command file:
Runtime support library: <automatic>

Figure 12: Compiler Version for CCS

Click on 'More' and here you can see the current compiler types and versions that are currently installed on your system:


Preferences

Compilers

Tool discovery path:
☑ C:/ti/ccs1100
☐ C:/ti
☐ C:/Program Files/Texas Instruments

Discovered tools:
Arm
TI v20.2.6.LTS [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-arm_20.2.6.LTS]
TI v20.2.5.LTS [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-arm_20.2.5.LTS]
TI Clang v1.3.0.LTS [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-armllvm_1.3.0.
C2000
TI v21.6.0.LTS [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-c2000_21.6.0.LTS]
TI v20.2.6.LTS [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-c2000_20.2.6.LTS]
TI v18.12.8.LTS [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-c2000_18.12.8.LTS]
C6000
TI v8.3.12 [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-c6000_8.3.12]
TI v8.3.11 [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-c6000_8.3.11]
MSP430
TI v21.6.0.LTS [C:/ti/ccs1100/ccs/tools/compiler/ti-cgt-msp430_21.6.0.LTS]

☐ Search for tools each time Code Composer Studio starts (may affect start-up time)

Figure 13: Compiler Directory and Locations

Note the given directory for the CCS compiler tools, follow the appropriate directory within your system and move the downloaded LTS file from the previous steps into this folder.

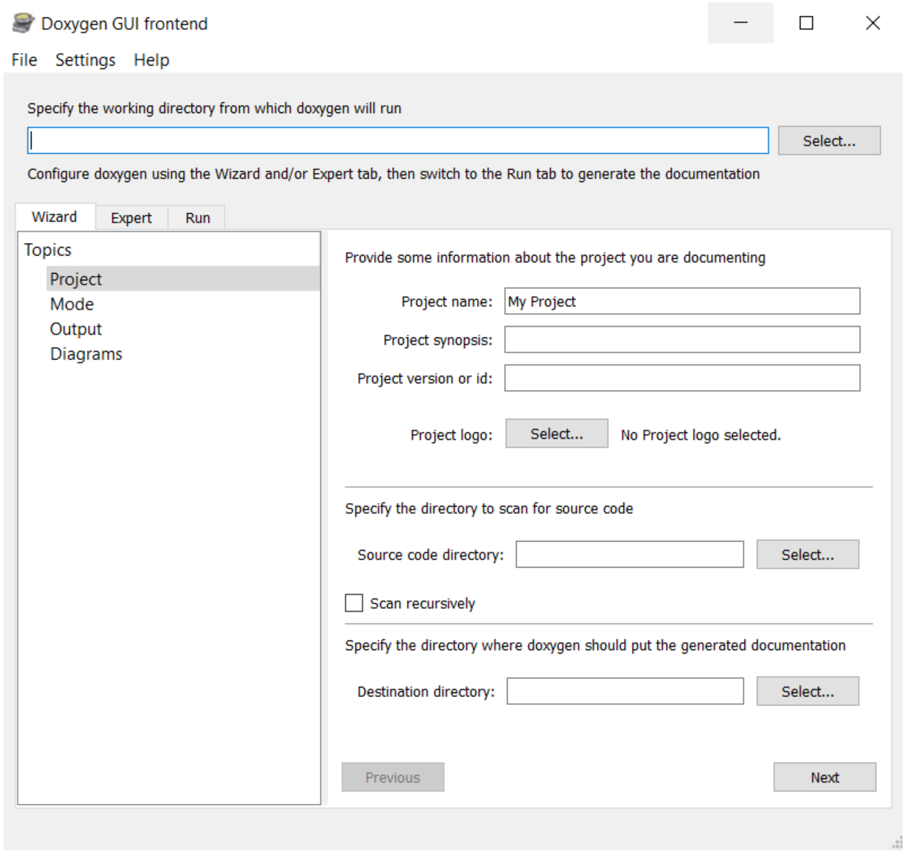
Now, you can select the compiler version that you would like in your build.

Section 6.0 – Documenting Code with Doxygen

Documenting code with Doxygen is an extremely helpful way to visualize the structure and descriptions of each line in your code.
It is an important programming practice to have detailed comments for your code and doing so will help you utilize Doxygen to its best potential.

To begin, download Doxygen here: Doxygen Download

After the download completes, launch Doxygen (Doxywizard), you will come to a screen that looks like this:



Specify the working directory from which doxygen will run

Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Provide some information about the project you are documenting

Project name: My Project

Project synopsis:

Project version or id:

Project logo: No Project logo selected.

Specify the directory to scan for source code

Source code directory:

☐ Scan recursively

Specify the directory where doxygen should put the generated documentation

Destination directory:

Figure 14: Doxygen application

Find the working directory where you downloaded you Doxygen application to. You can also provide relevant information, the code directory that you want to be documented and where you would like your HTML or LaTex saved to.

Click ‘Next’ and select the relevant code optimization, we are using C for CCS.
You can continue to click on ‘Next’, enabling relevant options that are helpful to your project, and then click on ‘Generate’.

Section 7.0 – Resources

Here are some resources related to the TI-Microcontroller that may be of interest: interest:

Code Composer Studio

TI C2000-F2837xD Microcontroller One Day Workshop

TMS320F2837xD Dual-Core Microcontrollers Datasheet

TMS320F2837xD Pinout and Other Useful Datasheets

Our Website - Holds demonstrations and more documentation from our Senior Design

Actuation GitHub Repository – From our Senior Design

Appendix A – Acronyms

TI	Texas Instruments
CCS	Code Composer Studio
PI	Proportional Integral
USB	Universal Serial Bus
uController	Microcontroller
LED	Light-Emitting Diode
Dev Board	Development Board
DC	Direct Current