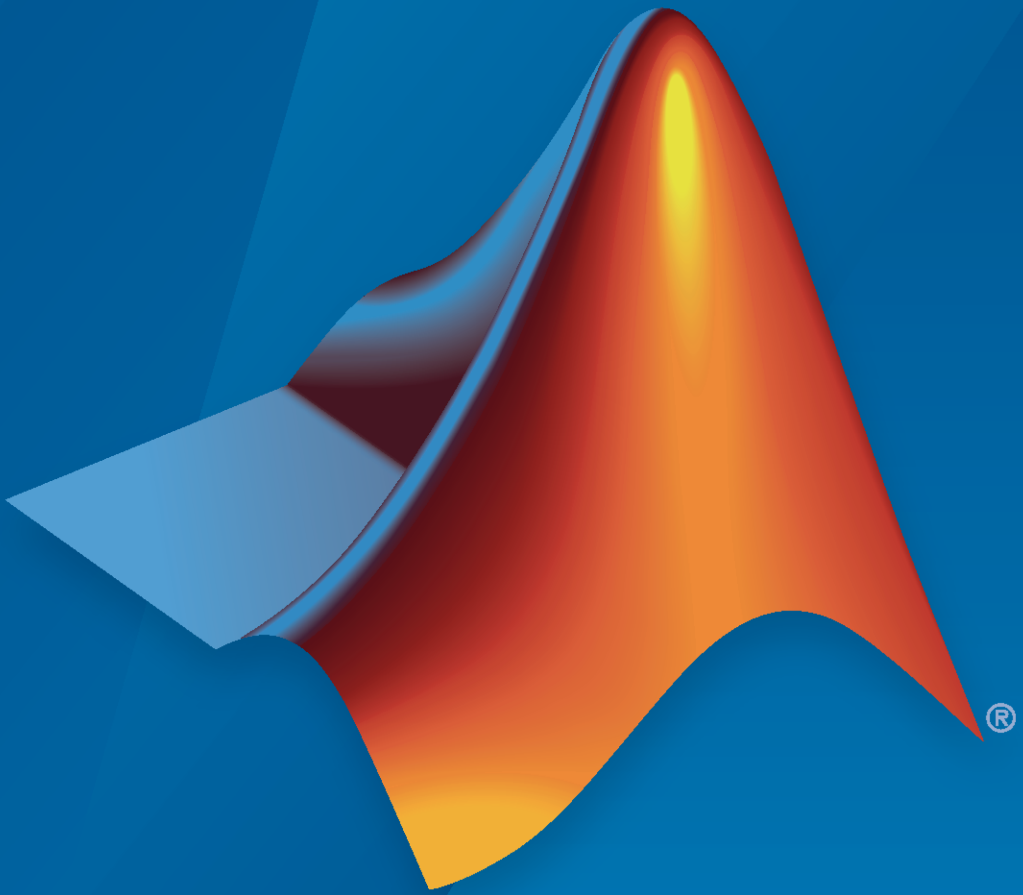
C2000™ Microcontroller Blockset

User's Guide



MATLAB® & SIMULINK®

R2023**a**

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

C2000™ Microcontroller Blockset User's Guide

Revision History

March 2023	Online only	New for Version 1.0 (Release R2023a)

Contents

2 MAT-File Logging on SD Card

3 SD Card Logging Troubleshooting

4 Examples

Working with C2000 Microcontroller Blockset

Hardware Setup for C2000 Microcontroller Blockset

Hardware boards and devices supported by MathWorks® require additional configuration and setup steps to connect to MATLAB and Simulink. C2000 Microcontroller Blockset provides a hardware setup process that guides you through registering, configuring, and connecting to your hardware board.

Hardware Setup for Third-Party Tools

The third party tools in hardware setup provides the following capabilities.

- **TI C28x CGT and TI ARM CGT -** To compile the generated code and create executable (Build, PIL , Monitor & Tune, and Connected IO)
- **TI Code Composer Studio -** Downloading the executable to hardware, CCS project creation and Connected IO
- Few processors require TI Code Composer Studio or TI C2000ware third party tool depending on the processor selected.
- TI F281x, F280x, F28044 processors requires driver installation

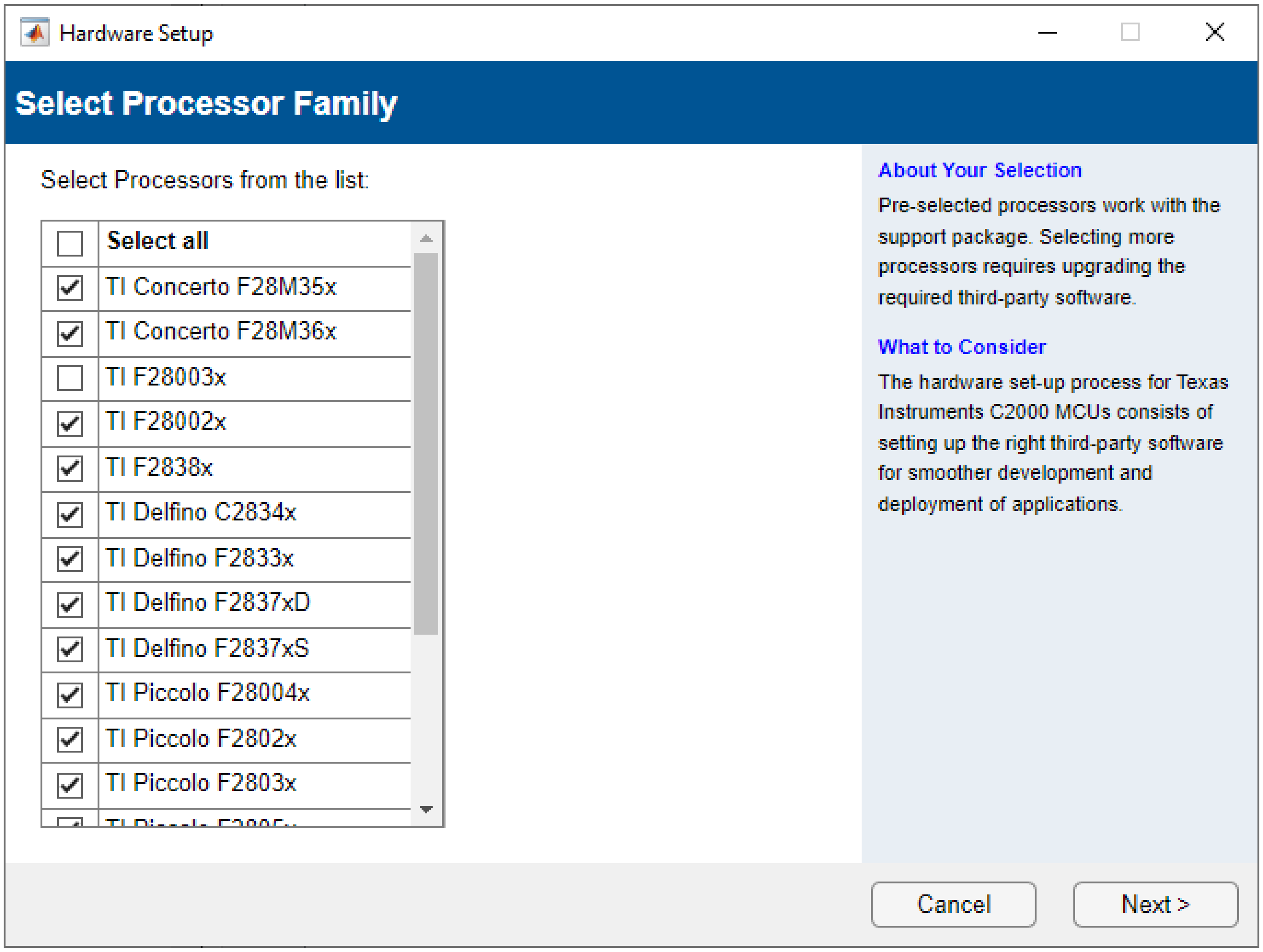
Note Control Suite and F28044x DSP installable are available only for windows. User must install in the windows platform and later copy the folder to the Linux platform.

To utilize the complete capabilities of C2000 Microcontroller Blockset, run the hardware setup. Follow one of the steps to launch the hardware setup.

Enter

```
c2000setup
```

at the MATLAB® prompt. This opens the Hardware Setup wizard.



Select Processors from the list:

☐	Select all
✓	TI Concerto F28M35x
✓	TI Concerto F28M36x
☐	TI F28003x
✓	TI F28002x
✓	TI F2838x
✓	TI Delfino C2834x
✓	TI Delfino F2833x
✓	TI Delfino F2837xD
✓	TI Delfino F2837xS
✓	TI Piccolo F28004x
✓	TI Piccolo F2802x
✓	TI Piccolo F2803x

About Your Selection

Pre-selected processors work with the support package. Selecting more processors requires upgrading the required third-party software.

What to Consider

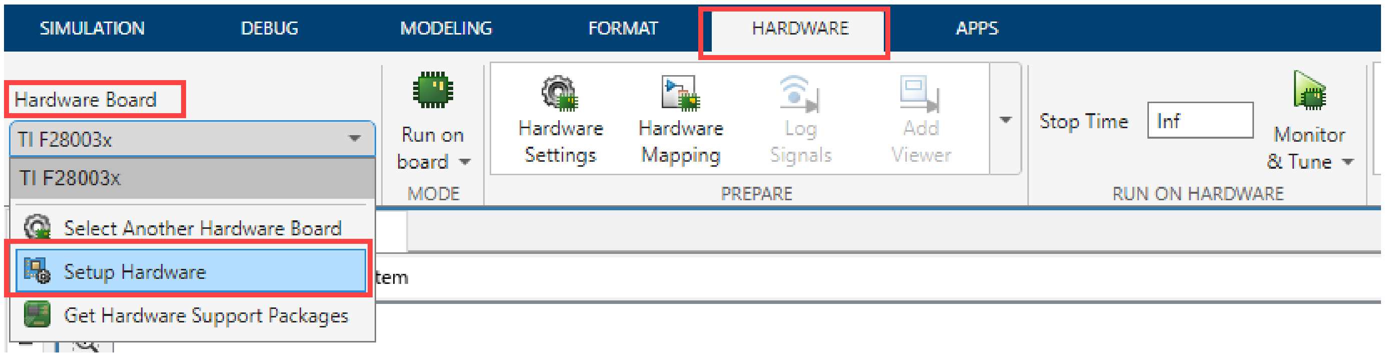
The hardware set-up process for Texas Instruments C2000 MCUs consists of setting up the right third-party software for smoother development and deployment of applications.

The Hardware Setup window provides instructions for configuring the C2000 blockset to work with your hardware.

Follow the instructions on each page of the Hardware Setup window. When the hardware setup process completes, you can open the examples to get familiar with the product and its features.

Alternatively you can also launch the hardware setup by:

1. Open any model which is configured with TI’s C2000™ hardware board. For more information, see “Configuring Target Hardware Resources”.
2. In the Simulink® toolstrip, navigate to **Hardware** > **Hardware Board** and select **Setup Hardware**.



Note

- The C2000 blockset supports CCS v3.3 and the later versions. However, CCS v3.3 does not support auto-download feature.

  Install the Code Composer Studio™ (CCS) version that supports your hardware board. For example, install CCS v6 or later versions to work on Texas Instruments™ C2000 F2807x, Texas Instruments C2000 F2837xD, and Texas Instruments C2000 F2837xS processors and install CCS v9 and later versions to work on Texas Instruments C2000 F2838x
- For more information related to CCS versions supported for MATLAB, see “Supported Third-Party Tools for Texas Instruments C2000 Processors”

See Also

`c2000setup` | “Supported Third-Party Tools for Texas Instruments C2000 Processors”

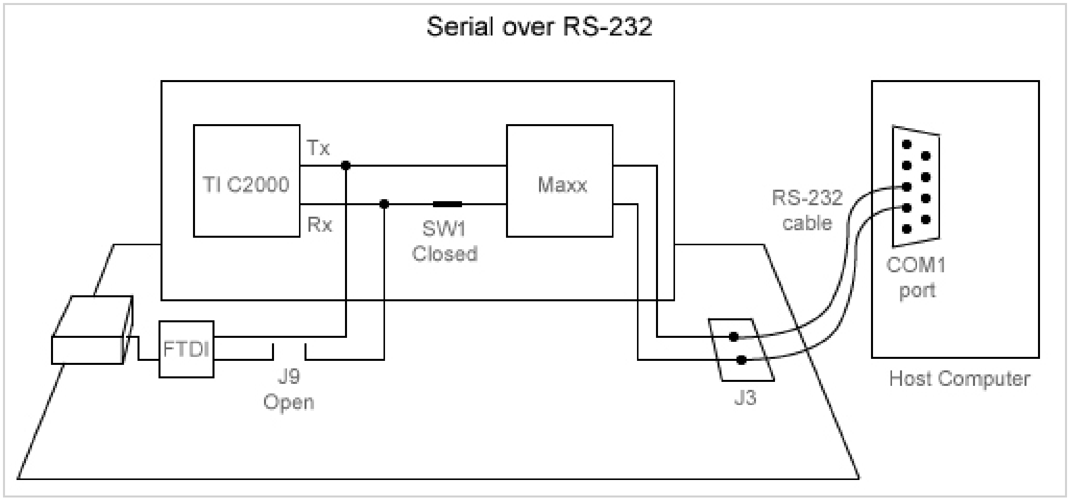
Set Up Serial Communication with Target Hardware

This section explains how to establish the serial communication between the host computer and the C2000 target hardware board.

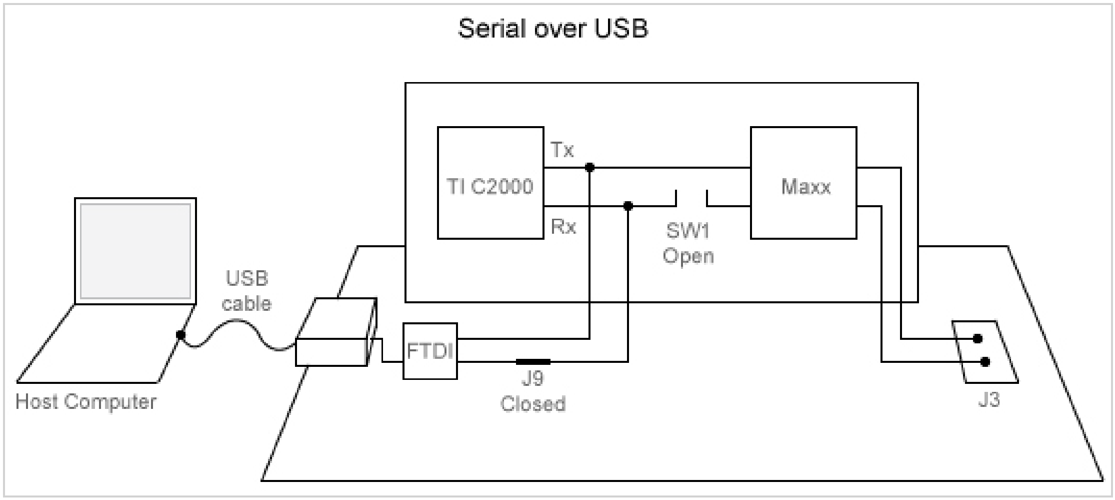
There are different control card versions available for C2000 processors. In some control cards, the SCI_A module pins are directly connected to the USB docking station and other control cards have a MAX32xx chip for RS–232 communication on the control card. These control cards have a 'switch' to connect or disconnect the Rx (Receive) line between the USB docking station and the MAX32xx chip.

You can establish a serial communication with the target hardware using RS-232 or the Serial over USB as shown below.

Serial over RS-232



Serial over USB



Note To use a USB JTAG on the USB docking station that is connected to SCI_A, use the switch to disconnect the Rx line that comes from the MAX3221 as it conflicts with the data you send to SCI_A.

While connecting a C2000 Launchpad to the host computer, ensure that:

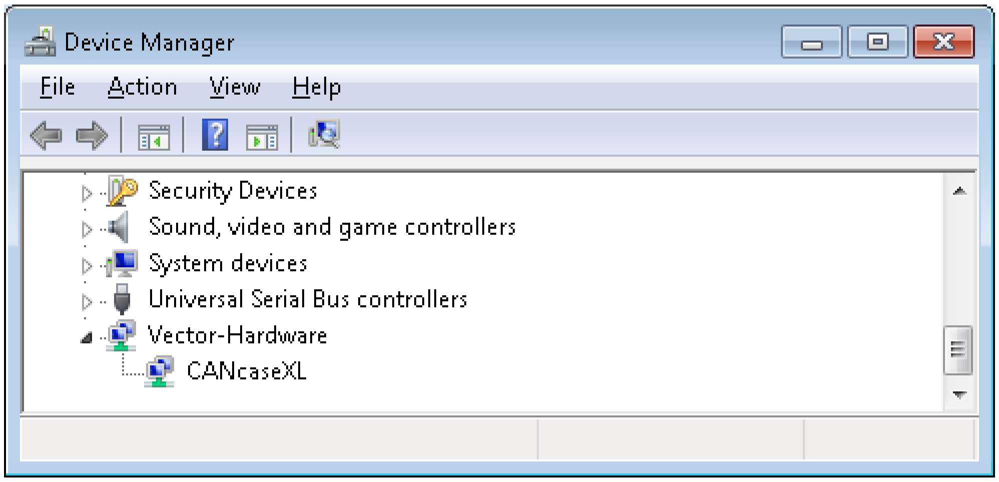
- For F28027, the S4 switch is on.
- For F28069, if you are using the default GPIO pins for transmit/receive (GPIO28/GPIO29), the jumper pin JP6 is open and the jumper pin JP7 is closed.
- The DIP switch (SW1) on the control card is in off position to enable the serial emulation using the FTDI chip.
- The J9 jumper on the docking station is closed.
- The GPIO pins using which the SCI_A module connects to the FTDI chip are configured correctly.
- The CCS tool is closed before running the program in external mode. You can use a tool such as PuTTY to test the basic working of Rx and Tx before trying the external mode.
- The COM port set at **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources** > **External mode** is the same as the COM port of the serial interface on Windows.

Set Up CAN Communication with Target Hardware

This section describes how to set up CAN communication with C2000 target hardware.

Follow these steps to set up CAN communication between the target hardware and your host computer.

1 Make sure that the Vector Hardware is installed properly in the Windows® Device Manager.



2 Download and install the latest version of the Vector XL Library from the `Vector Web site`. After installing the library, copy the file vxlapi64.dll from the installation folder (for example C:\Softwares\Vector_Driver_Setup_9_3_0\Common) to the windows root\system32 folder.

   This folder name may vary depending on the Windows OS (for example C:\Windows\Sys32).

3 Run the vcanconf.exe in the vector driver installation.

4 Right-click on the application and select "Add application".

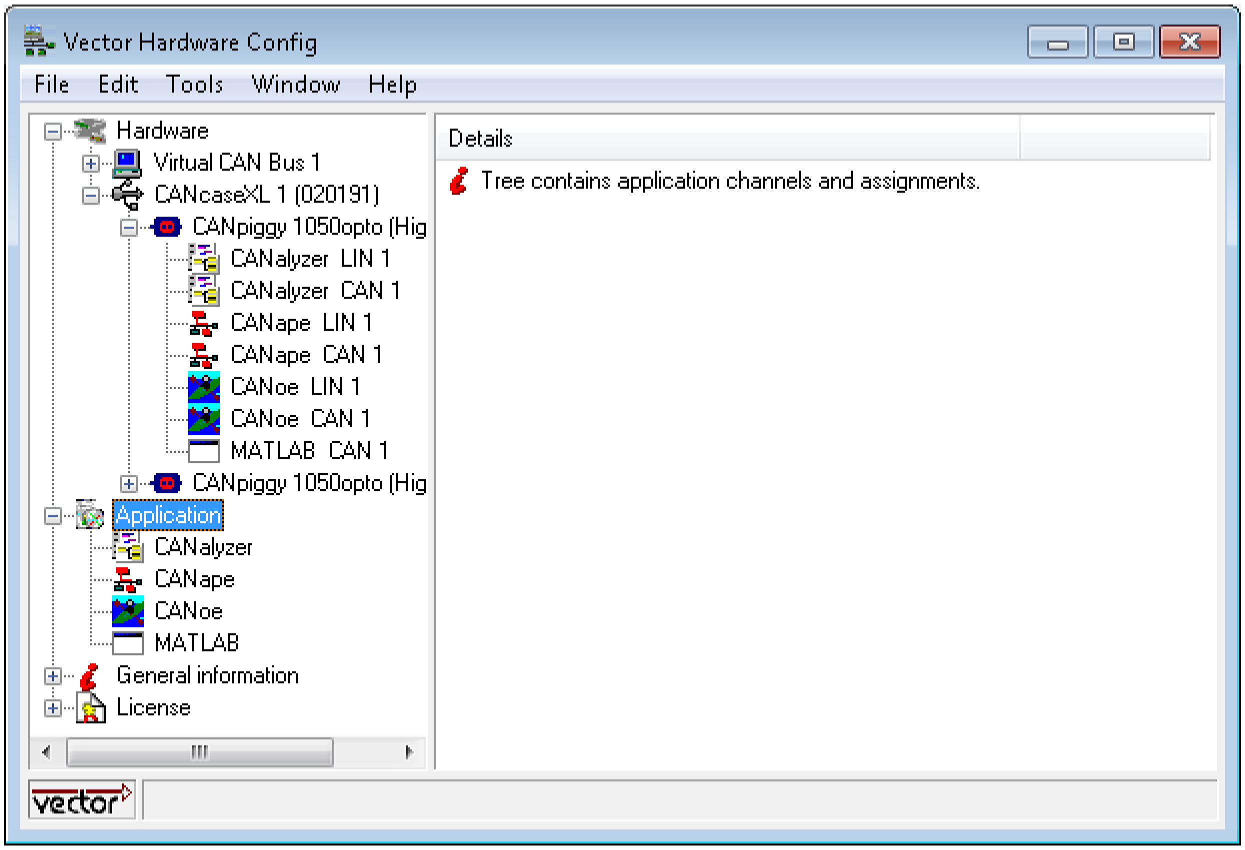
   a Name the application as ‘MATLAB’ and keep the default settings.

   b Right-click the first CAN piggy hardware device, select MATLAB, and pick CAN 1.

   c Right-click the first CAN piggy hardware device again and click **Default baud rate** to change the baud.

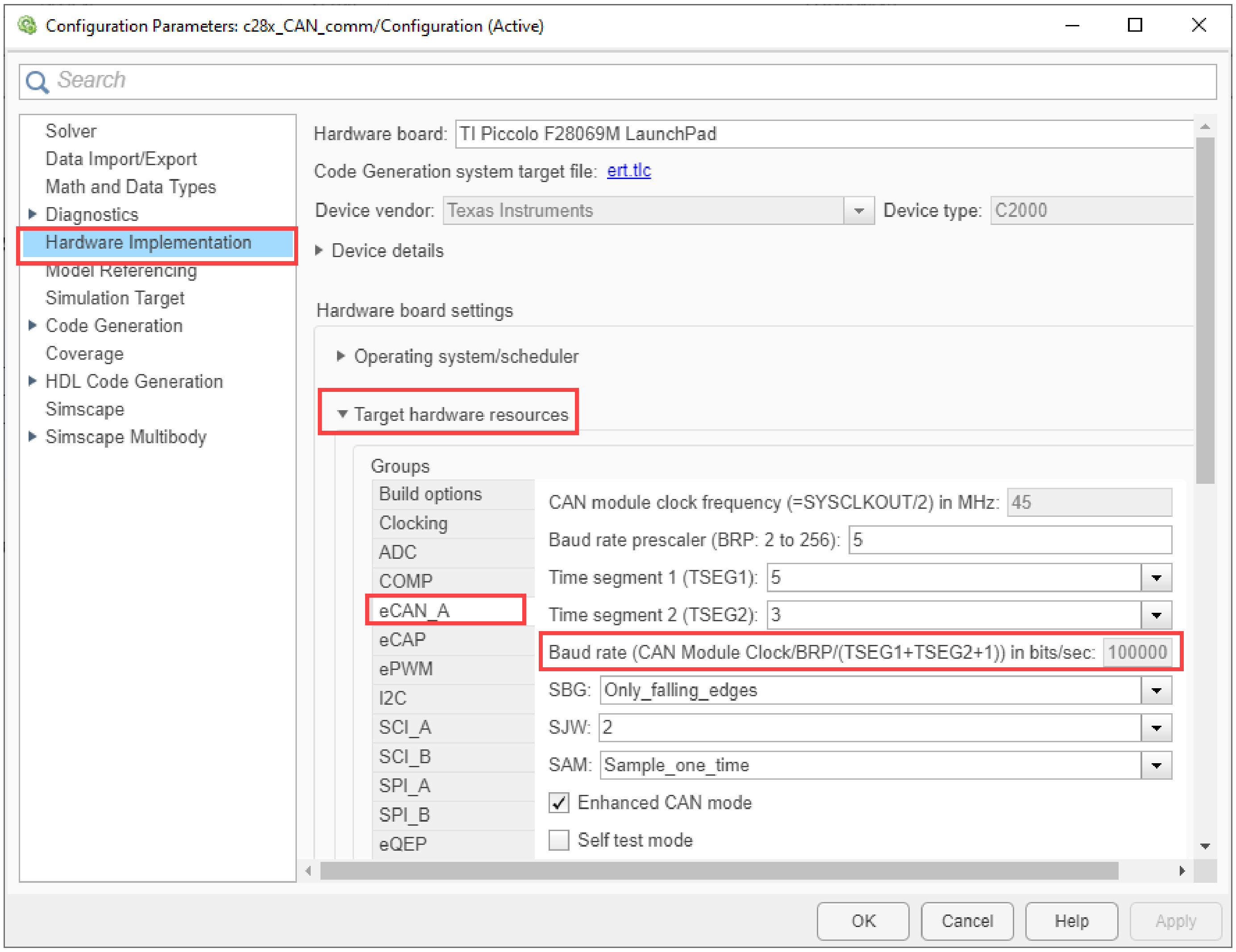
      This baud has to match your Simulink model CAN baud that you can change using the Target Preferences settings.

      The **Vector Hardware Config** window is shown.



Tree contains application channels and assignments.

5 Open the model. Go to Configuration Parameters dialog box, select the desired **Hardware board** and verify the CAN baud rate. The default baud is `1 Mbits/sec`.


Configuration Parameters: c28x_CAN_comm/Configuration (Active)

Hardware board: TI Piccolo F28069M LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments Device type: C2000

Hardware board settings

Target hardware resources

Groups: eCAN_A

CAN module clock frequency (=SYSCLKOUT/2) in MHz: 45

Baud rate prescaler (BRP: 2 to 256): 5

Time segment 1 (TSEG1): 5

Time segment 2 (TSEG2): 3

Baud rate (CAN Module Clock/BRP/(TSEG1+TSEG2+1)) in bits/sec: 100000

SBG: Only_falling_edges

SJW: 2

SAM: Sample_one_time

Enhanced CAN mode ✓

Self test mode

CAN communication is now set up between your host computer and the C2000 target hardware. To know more about the CAN communication using eCAN, see “CAN Communication Using eCAN Blocks” on page 4-104

Scheduling and Timing

Often, developers choose to run the code generated by Embedded Coder® in the context of a timer interrupt. Model blocks run in a periodical fashion clocked by the periodical interrupt whose period is tied to the base sample time of the model.

This execution scheduling model is not flexible enough for many systems, especially control and communication systems, which must respond to external events in real time. Such systems require the ability to handle various hardware interrupts in an asynchronous fashion.

Embedded Coder software lets you model and generate code for such systems by creating tasks driven by Hardware Interrupt blocks in addition to the tasks that are left to be handled in the context of the timer interrupt.

Timer-Based Interrupt Processing

For code that runs in the context of the timer interrupt, each iteration of the model solver is run after an interrupt has been posted and serviced by an interrupt service routine (ISR). The code generated for the C2000 processors uses `CPU_timer0` by default.

The timer is configured so that the base rate sample time of the model corresponds to the interrupt rate. The timer period and prescaler are calculated and set up to produce the desired rate as follows:

$$BaseRateSampleTime = \frac{TimerPeriod}{TimerClockSpeed}$$

The minimum achievable base rate sample time depends on the model complexity. The maximum value depends on the maximum timer period value (2^{32}-1) and the CPU clock speed .

If the blocks in the model inherit their sample time value, and a sample time is not explicitly defined, the default value is 0.2 s.

For more information about timer-based interrupt processing, see “C28x-Scheduler Options”.

High-Speed Peripheral Clock

The Event Managers and their general-purpose timers, which drive PWM waveform generation use the high-speed peripheral clock (HISCLK). By default, this clock is selected in Embedded Coder software. This clock is derived from the system clock (SYSCLKOUT):

HISCLK = [SYSCLKOUT / (high-speed peripheral prescaler)]

The high-speed peripheral prescaler is determined by the HSPCLK bits set in SysCtrl. The default value of HSPCLK is 1, which corresponds to a high-speed peripheral prescaler value of 2.

For example, on the F2812, the HISCLK rate becomes

HISCLK = 150 MHz / 2 = 75 MHz

External Interrupt Processing

For code that runs in the context of an external interrupt, the model uses the C28x Hardware Interrupt block. Configure the interrupt operation with **Configuration Parameters > Hardware**

Implementation > **Hardware board settings** > **Target hardware resources** > **External Interrupt**. For more information, see “Model Configuration Parameters for Texas Instruments C2000 Processors”.

ADC Interrupt Based Scheduling

You can set ADC interrupt as the Simulink model base rate. This means that every periodic event in the model will occur at the rate decided by ADC interrupt.

In order to set the base rate, ADC interrupt must be triggered. In general, an ePWM module can be used to trigger an ADC interrupt. An ePWM is free running counter which can be configured to trigger Start of Conversion for ADC module at a periodic interval. The ADC module can trigger an interrupt at the end of its conversion.

Other sources for ADC Start of Conversion include external interrupt and software.

In any model with base rate trigger:

- Any task in the model should not have base rate less than the base rate trigger.
- The sample time of ADC block should match with the base rate of the model.

For more information about ADC Interrupt based scheduling, see “C28x-Scheduler Options” and “Field-Oriented Control of PMSM with Quadrature Encoder Using C2000 Processors” on page 4-16.

Sharing General Purpose Timers Between C281x Peripherals

In this section...

TMS320x281x DSP devices have four General Purpose (GP) timers. Each Event Manager (EV) module includes two GP timers:

- EVA includes GP Timer 1 and GP Timer 2.
- EVB includes GP Timer 3 and GP Timer 4.

You can use the GP Timers independently or to operate peripherals associated with the EV Manager, such as PWM, QEP, and CAP.

The following table describes the timer-peripheral mapping of the c281xlib block library.

GP Timer Use for C281x Peripheral Blocks

	GP Timer 1	GP Timer 2	GP Timer 3	GP Timer 4
PWM1-PWM6	✓			
PWM7-PWM12			✓	
QEP1-QEP2		✓		
QEP3-QEP4				✓
CAP1-CAP3	✓	✓		
CAP4-CAP6			✓	✓

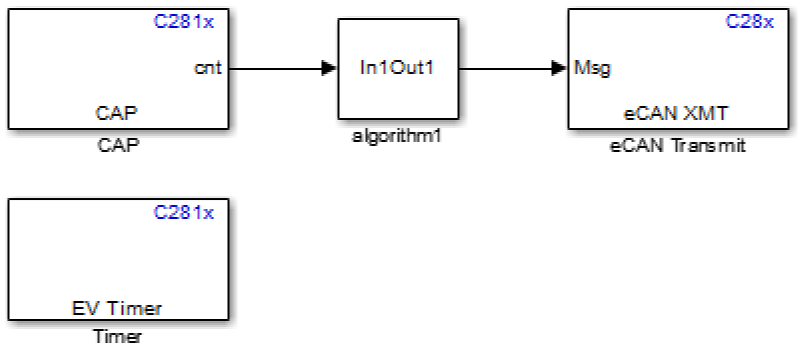
Each PWM or QEP peripheral has access to only one timer, while each CAP peripheral has access to two timers. In the PWM and QEP blocks, you can set the **Module** option to A or B to determine which unique timer-peripheral combination the block configures. By comparison, in the CAP block, you can use the **Time base** option to select one of two timers for each CAP peripheral.

Each GP timer is available to multiple peripherals. For example:

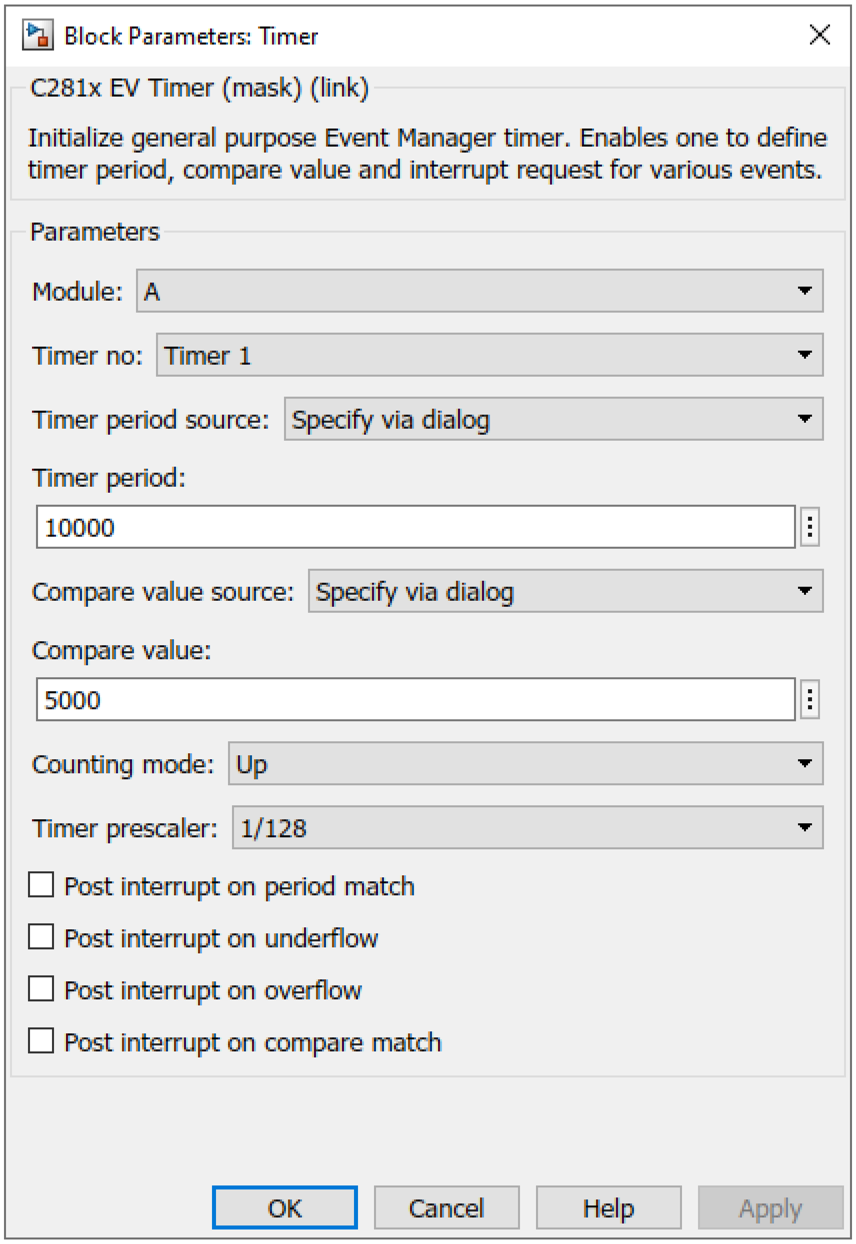
- PWM1-PWM6 and CAP1-CAP3 share GP Timer 1
- PWM7-PWM12 and CAP4-CAP6 share GP Timer 3
- QEP1-QEP2 and CAP1-CAP3 share GP Timer 2
- QEP3-QEP4 and CAP4-CAP6 share GP Timer 4

The PWM, QEP, CAP, and Timer blocks each provide independent access to key timer registers. If the blocks in your model share a specific GP timer, check that the timer-related settings are compatible. If the peripheral settings for a shared timer are not compatible, the software generates an error when you update the model or generate code.

Sharing General Purpose Timers Between CAP and eCAN



The model contains Timer and CAP blocks that both use Timer 1 (GP Timer 1).



Block Parameters: Timer

C281x EV Timer (mask) (link)

Initialize general purpose Event Manager timer. Enables one to define timer period, compare value and interrupt request for various events.

Parameters

Module: A

Timer no: Timer 1

Timer period source: Specify via dialog

Timer period:

10000

Compare value source: Specify via dialog

Compare value:

5000

Counting mode: Up

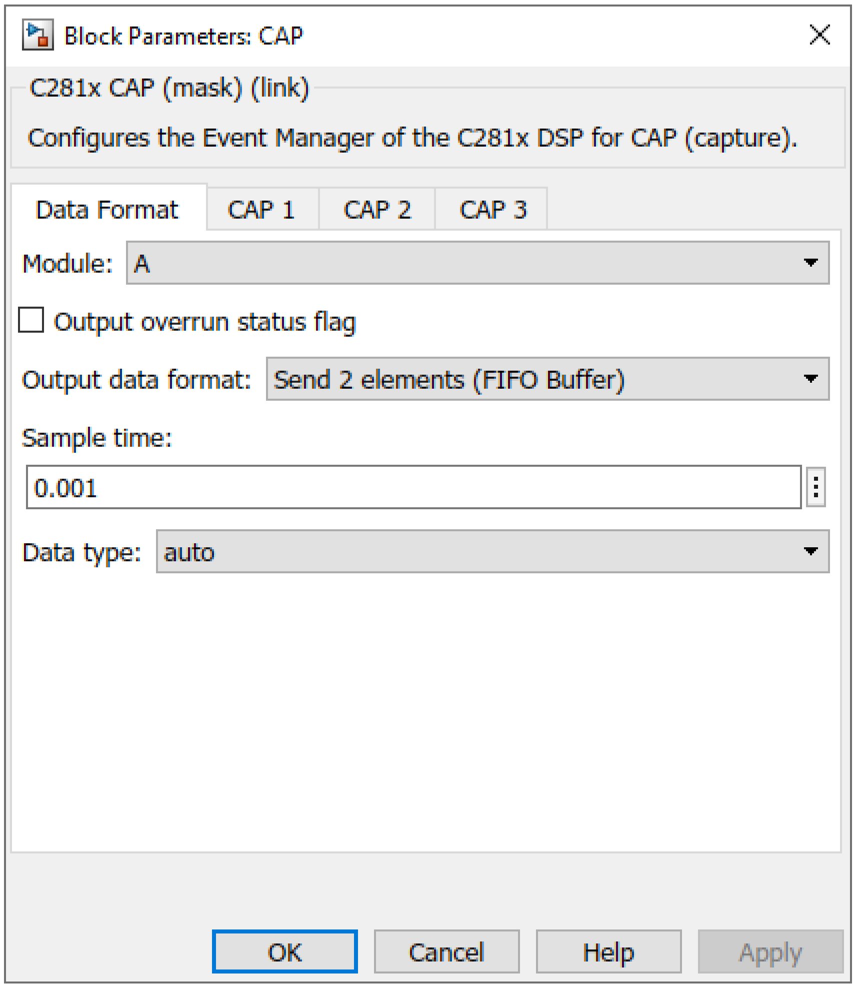
Timer prescaler: 1/128

☐ Post interrupt on period match

☐ Post interrupt on underflow

☐ Post interrupt on overflow

☐ Post interrupt on compare match


Block Parameters: CAP

C281x CAP (mask) (link)

Configures the Event Manager of the C281x DSP for CAP (capture).

Module: A

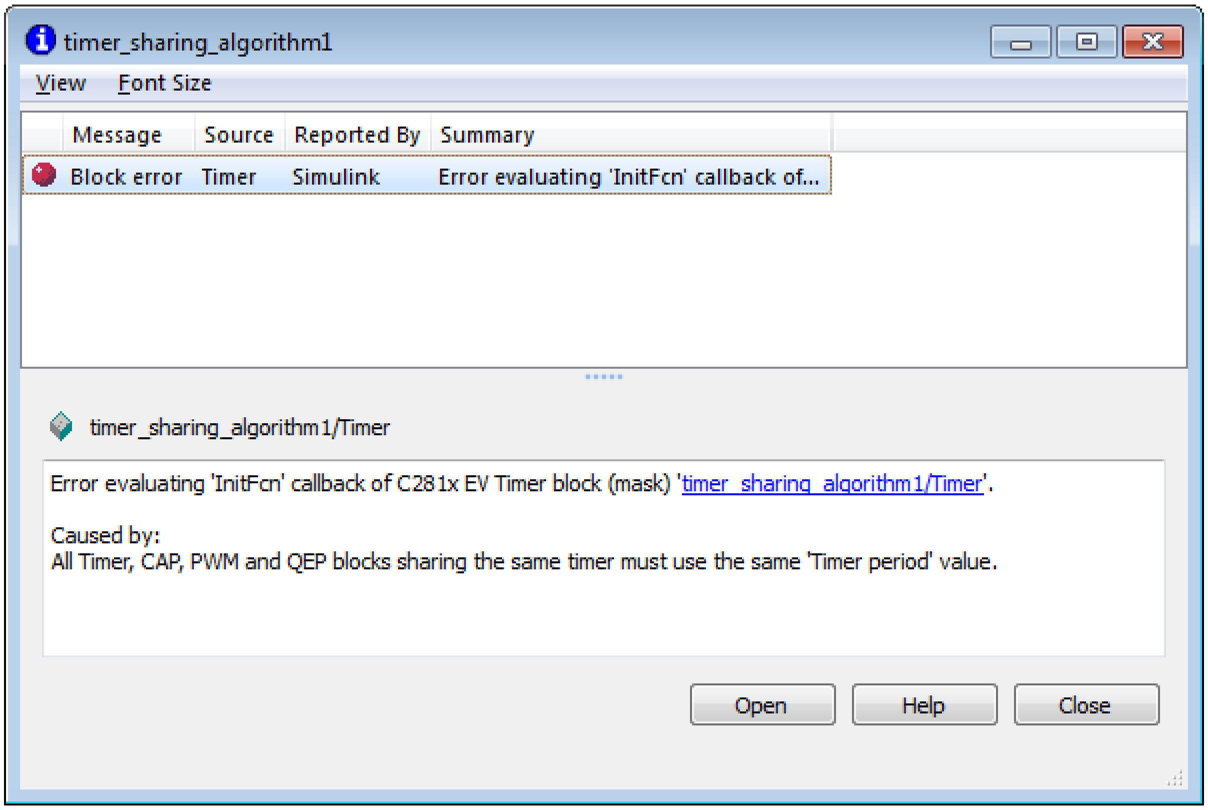
☐ Output overrun status flag

Output data format: Send 2 elements (FIFO Buffer)

Sample time: 0.001

Data type: auto

Both blocks have the same values for **Timer prescaler** and **Counting mode**. However, each block has different values for **Timer period**. The value of **Timer period** for Timer 1 is `65535` in the CAP block and `10000` in the Timer block.



Message	Source	Reported By	Summary
Block error	Timer	Simulink	Error evaluating 'InitFcn' callback of...

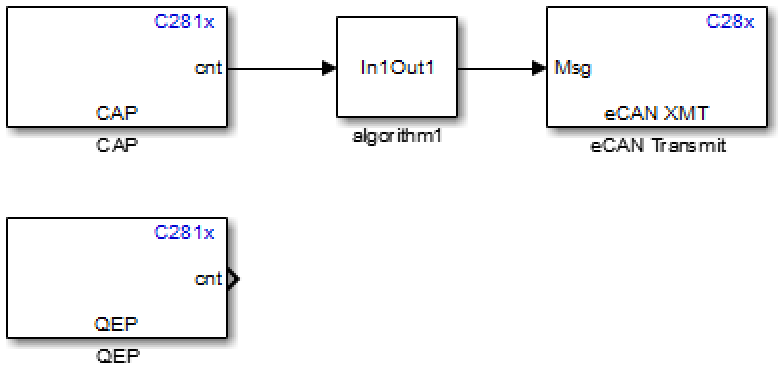
timer_sharing_algorithm1/Timer

Error evaluating 'InitFcn' callback of C281x EV Timer block (mask) 'timer_sharing_algorithm1/Timer'.

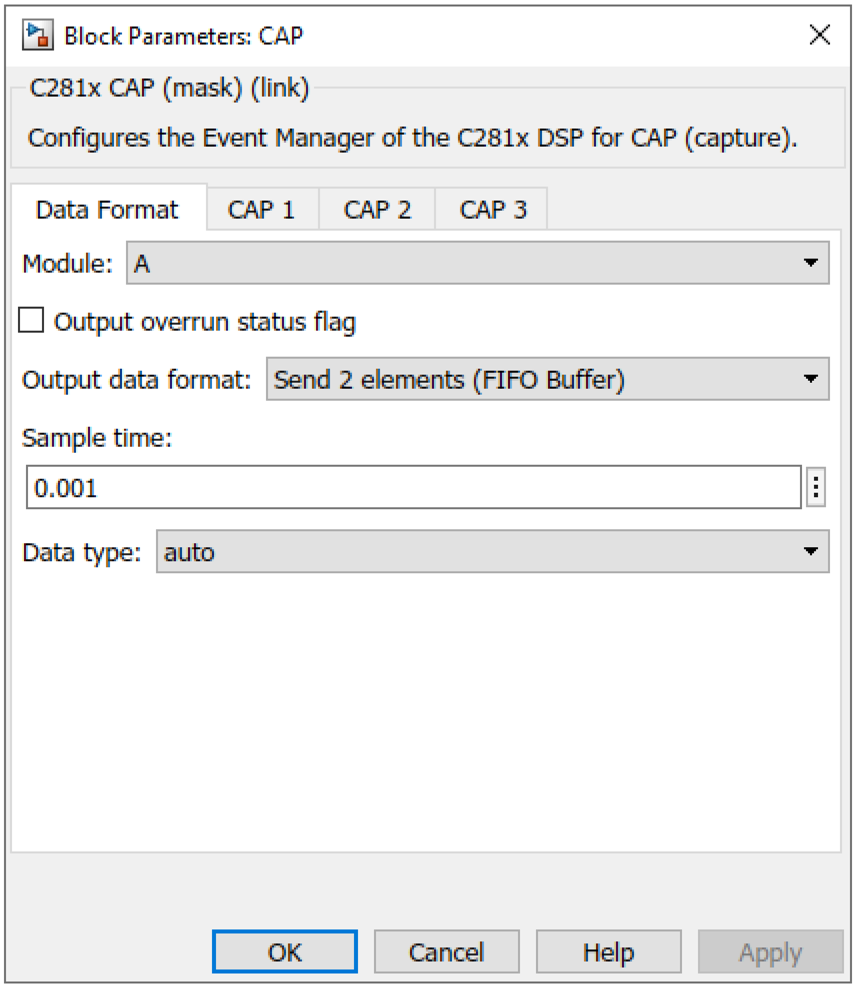
Caused by:
All Timer, CAP, PWM and QEP blocks sharing the same timer must use the same 'Timer period' value.

Since both blocks configure the same timer, and settings conflict, the software generates an error when you update the model.

Sharing General Purpose Timers Between CAP and SPI



The model contains QEP and CAP blocks that both use Timer 2. In the CAP block, the **Time base** option shows which timer the block uses. In the QEP block, setting **Module** to A configures the block to use QEP1–QEP2. GP Timer Use for C281x Peripheral Blocks shows that QEP1–QEP2 use Timer 2.


Block Parameters: CAP

C281x CAP (mask) (link)

Configures the Event Manager of the C281x DSP for CAP (capture).

Module: A

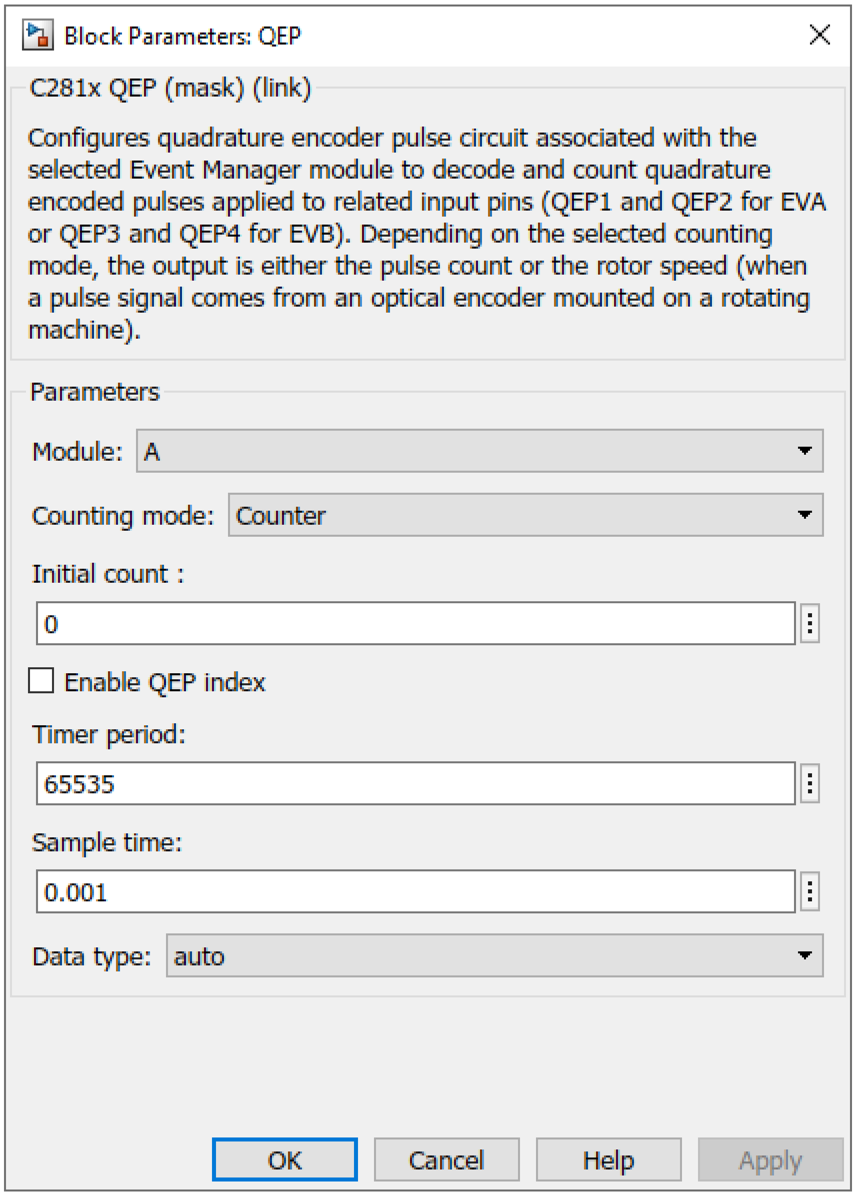
☐ Output overrun status flag

Output data format: Send 2 elements (FIFO Buffer)

Sample time:

0.001

Data type: auto


Block Parameters: QEP

C281x QEP (mask) (link)

Configures quadrature encoder pulse circuit associated with the selected Event Manager module to decode and count quadrature encoded pulses applied to related input pins (QEP1 and QEP2 for EVA or QEP3 and QEP4 for EVB). Depending on the selected counting mode, the output is either the pulse count or the rotor speed (when a pulse signal comes from an optical encoder mounted on a rotating machine).

Parameters

Module: A

Counting mode: Counter

Initial count : 0

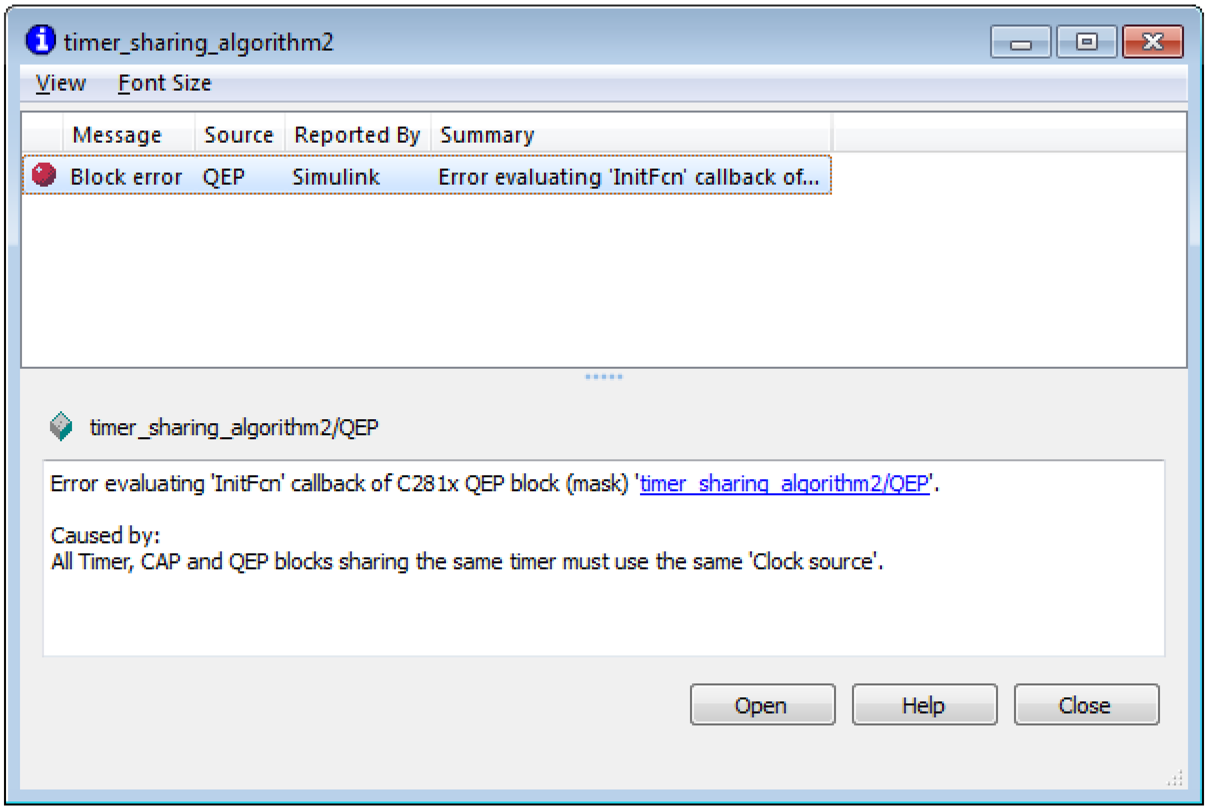
☐ Enable QEP index

Timer period: 65535

Sample time: 0.001

Data type: auto

Currently, both blocks define different clock sources for Timer 2. The CAP block uses `Internal` as a **Clock source**. The QEP block, which does not have a configurable **Clock source** setting, uses the QEP circuit as a clock source. If you build the model, the software generates the following error message.



Message	Source	Reported By	Summary
Block error	QEP	Simulink	Error evaluating 'InitFcn' callback of...

timer_sharing_algorithm2/QEP

Error evaluating 'InitFcn' callback of C281x QEP block (mask) 'timer_sharing_algorithm2/QEP'.

Caused by:
All Timer, CAP and QEP blocks sharing the same timer must use the same 'Clock source'.

To avoid generating errors when you build the model, change **Clock source** in the CAP block to `QEP device`.

Apply the c2000lib Blockset

In this section...

Introduction

This section uses an example to show how to create a Simulink model that uses C2000 Microcontroller Blockset blocks to target your board. The example creates a model that performs PWM duty cycle control via pulse width change. It uses the C2812 ADC block to sample an analog voltage and the C2812 PWM block to generate a pulse waveform. The analog voltage controls the duty cycle of the PWM and you can observe the duty cycle change on the oscilloscope.

Hardware Setup

The following hardware is required for this example:

- Spectrum Digital eZdsp F2812
- Function generator
- Oscilloscope and probes

To connect the target hardware:

1. Connect the function generator output to the ADC input ADCINA0 on the eZdsp F2812.
2. Connect the output of PWM1 on the eZdsp F2812 to the analog input of the oscilloscope.
3. Connect VREFLO to AGND on the eZdsp F2812. See the section on the Analog Interface in Chapter 2 of the *eZdsp F2812 Technical Reference*.

Starting the c2000lib Library

At the MATLAB prompt, type the following command:

```
c2000lib
```

This command opens the `c2000lib` library blockset, which contains libraries of blocks designed for targeting your board.

Setting Up the Model

Preliminary tasks for setting up a new model include configuring your model for the target hardware and setting the simulation parameters.

To configure your model with `ert.tlc` system target file:

1. In the Simulink Editor of your model, select **Simulation** > **Model Configuration Parameters** > **Hardware Implementation**.
2. Select a TI C2000 **Hardware board**. The `Texas Instruments Code Composer Studio (c2000)` option automatically gets selected for **Toolchain**.

   The following default settings in the **Simulation** > **Model Configuration Parameters** dialog box appear when you select your C2000 hardware board.

Pane	Field	Setting
Solver	**Type**	`Fixed-step`
Solver	**Solver**	`discrete (no continuous states)`
Solver	**Higher priority value indicates higher task priority**	`Value is 'off' and the parameter is disabled`
Code Generation > Optimization	**Remove internal data zero initialization**	`Value is 'off' and the parameter is disabled`
Code Generation > Optimization	**Maximum stack size (bytes)**	`512 (MAU)`
Hardware Implementation	**Device vendor**	`Texas Instruments`
Hardware Implementation	**Device type**	`C2000`
Hardware Implementation	**Support long long**	`Selected by default`
Code Generation > Interface	**Code replacement library**	`TI C28x`

3. Click **Apply**.

Note The generated code does not honor Simulink ‘stop time’ from the simulation. The ‘stop time’ is interpreted as `inf`. To implement a stop in the generated code, you must put a Stop Simulation block in your model.

Adding Blocks to the Model

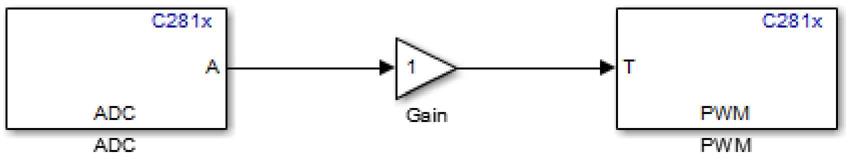
1. Open or double-click the C281x library, `c281xlib`.
2. Drag the ADC block into your model. Double-click the ADC block in the model and set **Sample time** to `64/80000`.
3. Drag the PWM block into your model. Double-click the PWM block in the model and set the following parameters.

Pane	Field	Parameter
Timer	**Module**	`A`
	Waveform period source	`Specify via dialog`
	Waveform period units	`Clock cycles`
	Waveform period	`64000`
	Waveform type	`Asymmetric`
Outputs	**Enable PWM1/PWM2**	Selected
	Duty cycle source	`Input port`
Logic	**PWM1 control logic**	`Active high`
	PWM2 control logic	`Active low`
Deadband	**Use deadband for PWM1/PWM2**	Selected
	Deadband prescaler	`16`
	Deadband period	`12`
ADC Control	**ADC start event**	`Period interrupt`

4 Enter `simulink` in the MATLAB Command Window to open the Simulink Library browser. Drag a Gain block from the Math Operations library into your model. Double-click the Gain block in the model and set the following parameters in the Function Block Parameters dialog box. Click **OK**.

Pane	Field	Parameter
Main	**Gain**	`30`
	Multiplication	`Element-wise(K.*u)`
	Sample time	`-1`
Signal Attributes	**Output data type mode**	`uint(16)`
	Integer rounding mode	`Floor`
Parameter Attributes	**Parameter data type mode**	`Inherit from input`

5 Connect the ADC block to the Gain block and the Gain block to the PWM block.



Generating Code from the Model

This section summarizes how to generate code from your real-time model.

There are three ways to start the automatic code generation process:

- In the Simulink Editor, click **Build Model** or **Deploy to Hardware**.
- On your model, press **Ctrl+B**.
- Press the Build button on the **Code Generation** pane of the Configuration Parameters dialog box.

Configuring Timing Parameters for CAN Blocks

In this section...

The CAN Blocks

The bit rate of these four CAN blocks cannot be set directly:
C281x eCAN Receive
C281x eCAN Transmit
C280x/C28x3x eCAN Receive
C280x/C28x3x eCAN Transmit

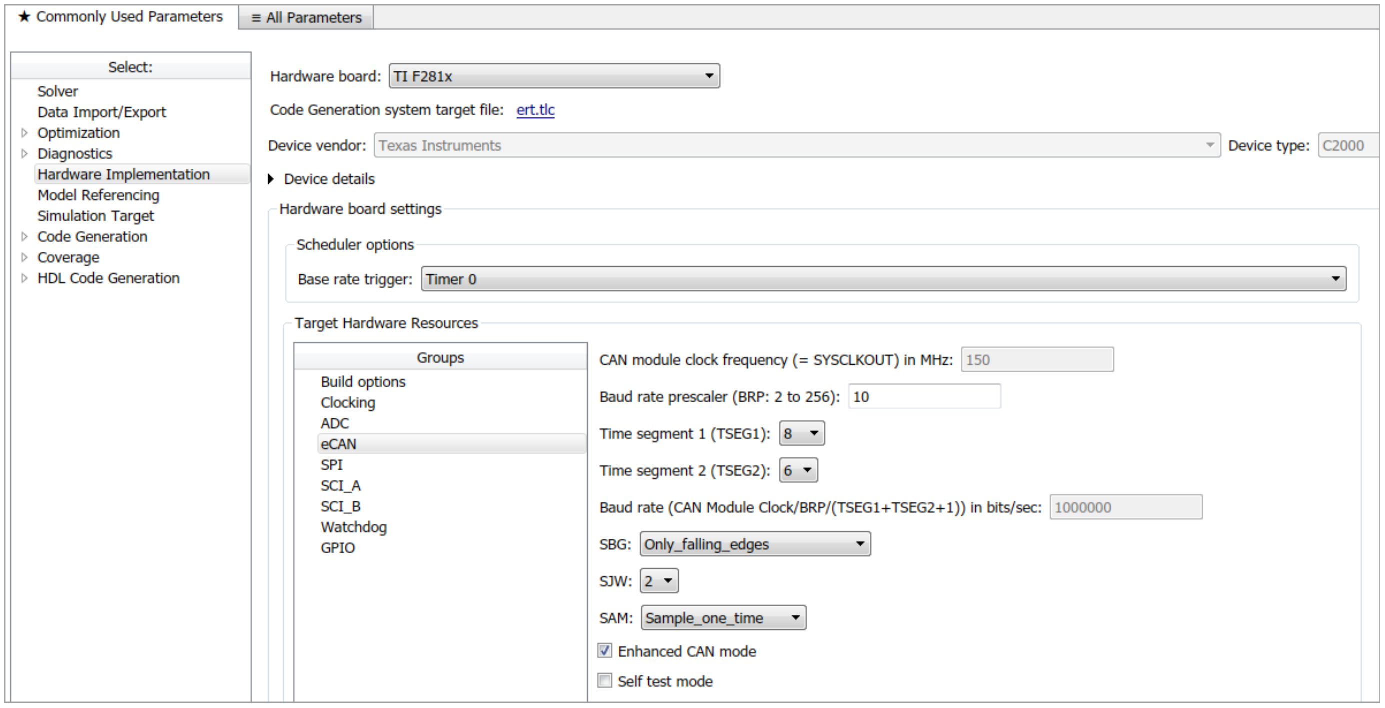
Setting Timing Parameters

Accessing the Timing Parameters

To set the Bitrate for a block whose bitrate cannot be set directly:

1. Configure the Target Hardware Resources tab.
2. Under the **Peripherals** tab, use the **TSEG1**, **TSEG2**, and **BaudRatePrescaler (BRP)** parameters to set the bitrate.

For example, the Target Hardware Resources tab for the F2812 eZdsp shown in the following figure.



★ Commonly Used Parameters | ≡ All Parameters

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- Coverage
- HDL Code Generation

Hardware board: TI F281x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Device details

Hardware board settings

Scheduler options

Base rate trigger: Timer 0

Target Hardware Resources

Groups
- Build options
- Clocking
- ADC
- eCAN
- SPI
- SCI_A
- SCI_B
- Watchdog
- GPIO

CAN module clock frequency (= SYSCLKOUT) in MHz: 150

Baud rate prescaler (BRP: 2 to 256): 10

Time segment 1 (TSEG1): 8

Time segment 2 (TSEG2): 6

Baud rate (CAN Module Clock/BRP/(TSEG1+TSEG2+1)) in bits/sec: 1000000

SBG: Only_falling_edges

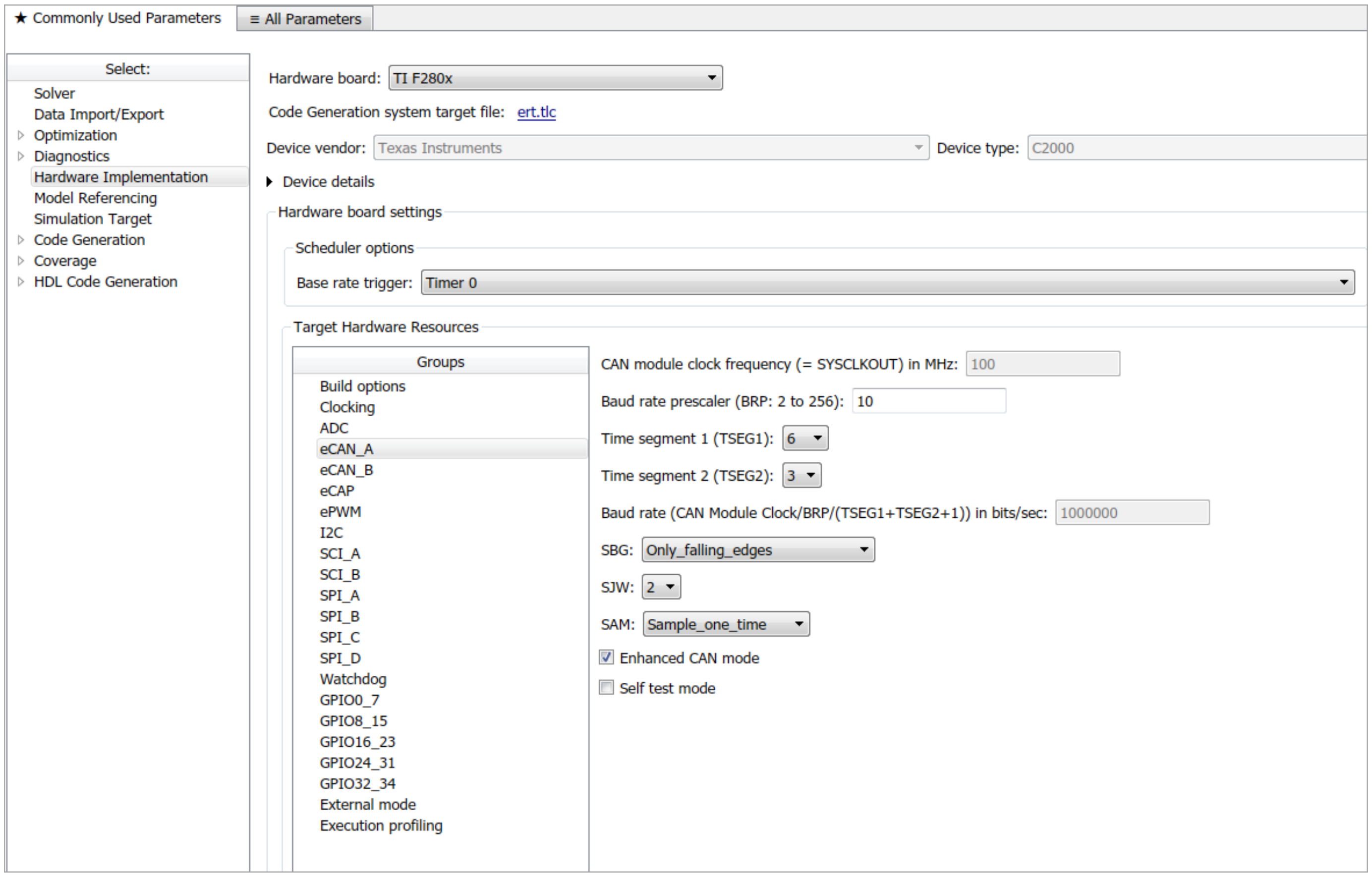
SJW: 2

SAM: Sample_one_time

☑ Enhanced CAN mode

☐ Self test mode

The C280x/C28x3x blocks have two independent eCAN modules.



Hardware board: TI F280x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments   Device type: C2000

Device details

Hardware board settings

Scheduler options

Base rate trigger: Timer 0

Target Hardware Resources

CAN module clock frequency (= SYSCLKOUT) in MHz: 100

Baud rate prescaler (BRP: 2 to 256): 10

Time segment 1 (TSEG1): 6

Time segment 2 (TSEG2): 3

Baud rate (CAN Module Clock/BRP/(TSEG1+TSEG2+1)) in bits/sec: 1000000

SBG: Only_falling_edges

SJW: 2

SAM: Sample_one_time

☑ Enhanced CAN mode

☐ Self test mode

The following sections describe the series of steps and rules that govern the process of setting these timing parameters.

Determining Timing Parameter Values

To determine the values for the timing parameters, complete the following steps:

1 Determine the CAN Bitrate specification based on your application.
2 Determine the frequency of the **CAN module clock**. For example:

   - CAN module clock = 100 MHz for the F2808 (Same as SYSCLKOUT)
   - CAN module clock = 150 MHz for the F2812 (Same as SYSCLKOUT)
   - CAN module clock = 75 MHz for the F28x3x (150 SYSCLKOUT/2)

3 Estimate the value of the **BaudRatePrescaler (BRP)**.
4 Solve this equation for BitTime:

   BitTime = CAN module clock frequency/(BRP * Bitrate)

5 Solve this equation for Bitrate:

   Bitrate = CAN module clock frequency/(BRP * BitTime)

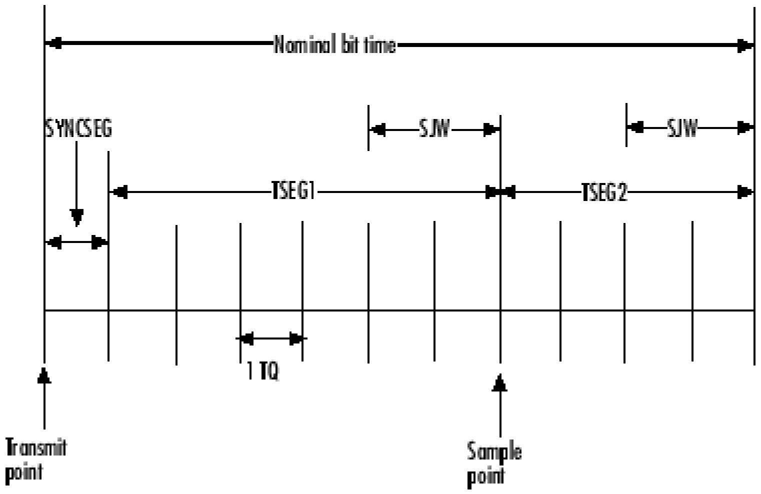
6 Estimate values for **TSEG1** and **TSEG2** that satisfy BitTime = TSEG1 + TSEG2 + 1.
7 Use the following rules to determine the values of **TSEG1** and **TSEG2**:

TSEG1 >= **TSEG2**
IPT (Information Processing Time) = 3/**BRP**
IPT <= **TSEG1** <= 16 TQ
IPT <= **TSEG2** <= 8 TQ
1 TQ <= **SJW** <= min (4 TQ, **TSEG2**)

where IPT is Information Processing Time, TQ is Time Quanta, and **SJW** is Synchronization Jump Width, also set in the Target Hardware Resources dialog box.

8 Iterate steps 4 through 7 until the values selected for TSEG1, TSEG2, and BRP meet the criteria.

The following illustration shows the relationship between the eCAN bit timing parameters.



Working with CAN Bit Timing

Assume that CAN Module Clock Frequency = 75 MHz, and a Bitrate of 1 Mbits/s is required.

1 Set the BRP to 5. Then substitute the values of CAN Module Clock Frequency, BRP, and Bitrate into the following equation, solving for BitTime:

BitTime = CAN Module Clock Frequency / (BRP * Bitrate)

BitTime = 75e6/(5 *1e6) = 15TQ

2 Set the values of **TSEG1** and **TSEG2** to 8TQ and 6TQ respectively. Substitute the values of *BitTime* from the previous equation, and the chosen values for *TSEG1* and **TSEG2** into the following equation:

BitTime = TSEG1 + TSEG2 + 1

15TQ = 8TQ + 6TQ + 1

3 Finally, check the selected values against the rules:
IPT = 3/**BRP** = 3/10 = .3
IPT <= **TSEG1** <= 16 TQ True! .3<=8TQ<=16TQ
IPT <= **TSEG2** <= 8TQ True! .3 <= 6TQ <= 8TQ
1TQ <= **SJW** <= min(4TQ, **TSEG2**) which means that **SJW** can be set to either 2, 3, or 4

4 When the chosen values satisfy the criteria, so further iteration is not required.

The following table provides example values for several bit rates when CAN Module Clock Frequency = 75 MHz, as it is with the F28335. Other combinations of the register values are possible.

Bitrate	TSEG1	TSEG2	Bit Time	BRP	SJW
0.25 Mbit/s	8	6	15	20	2
0.5 Mbit/s	8	6	15	10	2
1 Mbit/s	8	6	15	5	2

The following table provides example values for several bit rates when CAN Module Clock Frequency = 100 MHz, as it is with the F2808. Other combinations of the register values are possible.

Bitrate	TSEG1	TSEG2	Bit Time	BRP	SJW
0.25 Mbit/s	6	3	10	40	2
0.5 Mbit/s	5	4	10	20	2
1 Mbit/s	6	3	10	10	2

The following table provides example values for several bit rates when CAN Module Clock Frequency = 150 MHz, as it is with the F2812. Other combinations of the register values are possible.

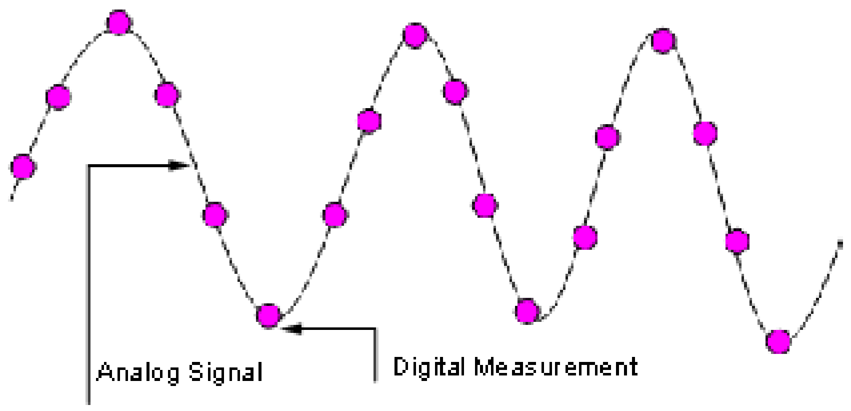
Bitrate	TSEG1	TSEG2	Bit Time	BRP	SJW
0.25 Mbit/s	8	6	10	40	2
0.5 Mbit/s	7	7	10	20	2
1 Mbit/s	8	6	10	10	2

Configuring Acquisition Window Width for ADC Blocks

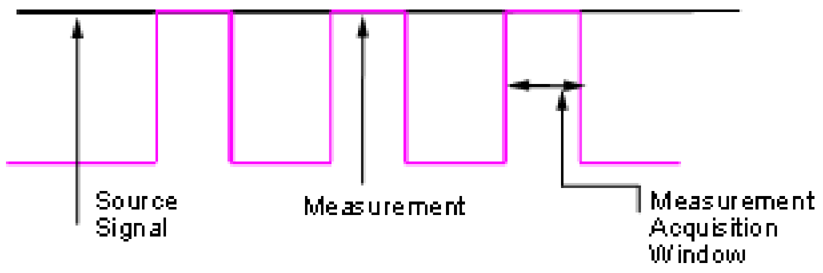
In this section...

What Is an Acquisition Window?

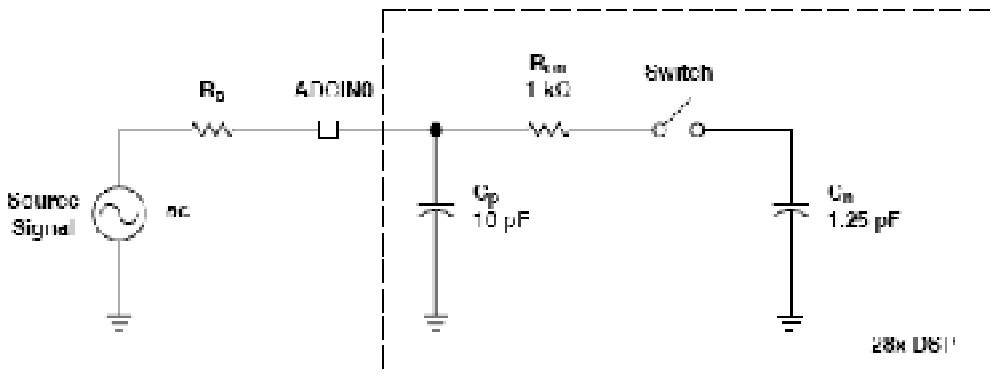
ADC blocks take a signal from an analog source and measure it with a digital device. The digital device does not measure in a continuous process, but in a series of discrete measurements, close enough together to approximate the source signal with the required accuracy.



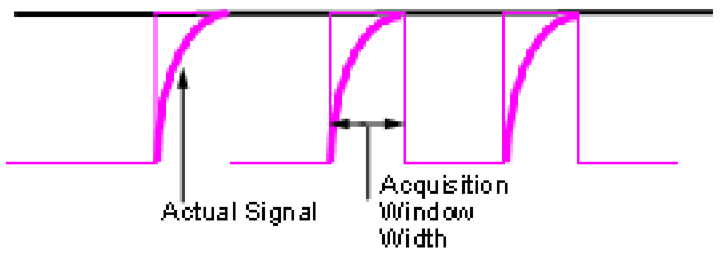
The digital measurement itself is not an instantaneous process, but is a measurement window, where the signal is acquired and measured, as shown in the following figure.



Ideally, when the measurement window is opened, the actual signal coming in would be measured perfectly. In reality the signal does not reach its full magnitude immediately. The measurement process can be modeled by a circuit similar to the one shown in the following figure for the ADC found on the F2812 eZdsp,



where the measurement circuit is characterized by a certain capacitance. In the preceding figure, when the switch is closed, the measurement begins. In this circuit, which is characterized by its capacitance, the signal received is not in a form of a step function as shown by the ideal measurement, but a ramp up to the true signal magnitude. The following figure shows what happens to the signal when the sampler switch is closed and the signal is received to be measured.



Because the signal acquisition is not instantaneous, it is very important to set a wide enough acquisition window to allow the signal to ramp up to full strength before the measurement is taken. If the window is too narrow, the measurement is taken before the signal has reached its full magnitude, resulting in erroneous data. If the window is too wide, the source signal itself may change, and the sampling may be too infrequent to reflect the actual value, also resulting in erroneous data. You must calculate the width of the acquisition window based on the circuit characteristics of resistance and capacitance of your specific circuit. Then, using the ADC parameters described in the following section, you can configure the acquisition window width.

Configuring ADC Parameters for Acquisition Window Width

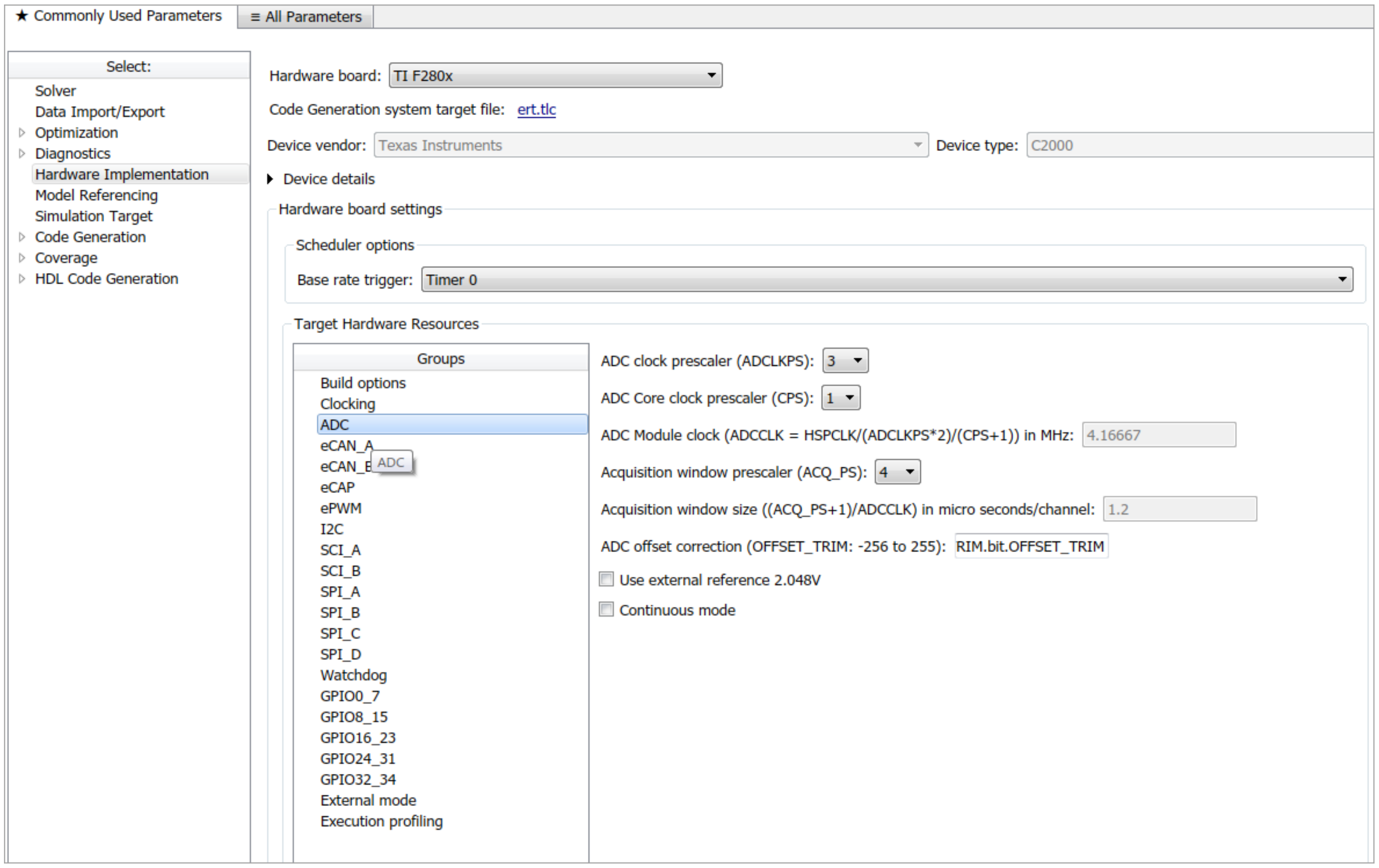
- "Accessing the ADC Parameters" on page 1-31
- "Configure Acquisition Window Width Using ADC Parameters" on page 1-33

Accessing the ADC Parameters

The ADC parameters can be set from the **Peripherals tab** of the Target hardware resources tab.

- You can set **ACQ_PS** — Acquisition Prescaler — to a value from 0 to 15. To obtain the actual value, increment the setting by 1. This increment produces an actual range from 1 to 16.
- You can set **ADCLKPS** — AD Clock Prescaler — to a value from 0 to 15. To obtain the actual value, increment the setting by 1. This increment produces an actual range from 1 to 16.

- You can set **CPS** — Clock Prescaler — to a value from 0 to 1. To obtain the actual value, increment the setting by 1. This increment produces an actual range from 1 to 2.



Hardware board: TI F280x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments Device type: C2000

Hardware board settings

Scheduler options

Base rate trigger: Timer 0

Target Hardware Resources

ADC clock prescaler (ADCLKPS): 3

ADC Core clock prescaler (CPS): 1

ADC Module clock (ADCCLK = HSPCLK/(ADCLKPS*2)/(CPS+1)) in MHz: 4.16667

Acquisition window prescaler (ACQ_PS): 4

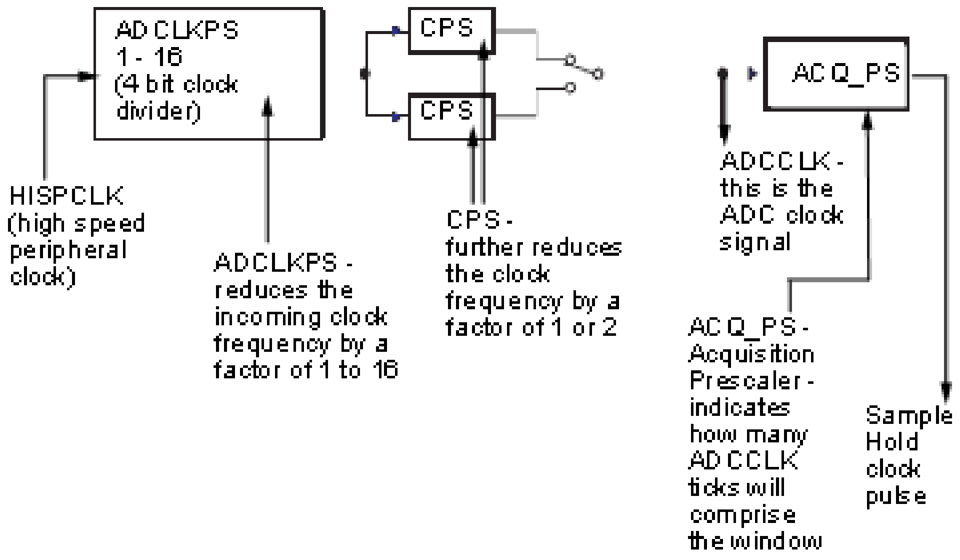
Acquisition window size ((ACQ_PS+1)/ADCCLK) in micro seconds/channel: 1.2

ADC offset correction (OFFSET_TRIM: -256 to 255): RIM.bit.OFFSET_TRIM

☐ Use external reference 2.048V

☐ Continuous mode

These three prescalers serve to reduce the speed of the clock and to set the acquisition window width. The following diagram shows how these prescalers are used.


ADCLKPS 1 - 16 (4 bit clock divider)
CPS
CPS
ACQ_PS
HISPCLK (high speed peripheral clock)
ADCLKPS - reduces the incoming clock frequency by a factor of 1 to 16
CPS - further reduces the clock frequency by a factor of 1 or 2
ADCCLK - this is the ADC clock signal
ACQ_PS - Acquisition Prescaler - indicates how many ADCCLK ticks will comprise the window
Sample Hold clock pulse

In the preceding diagram, the high-speed peripheral clock frequency is received and then divided by the **ADCLKPS**. The reduced clock frequency is then further divided by **CPS**. The resulting frequency is the **ADCCLK** signal. The value of **ACQ_PS** then determines how many **ADCCLK** ticks comprise one S/H (sample and hold) period, or in other words, the length of the acquisition window.

Configure Acquisition Window Width Using ADC Parameters

The following examples show how you can use ADC parameters to configure the acquisition window width:

Example 1:

If the HISPCLK = 30 MHz, and **ADCLKPS**=1 (which is a value of 2), the result is 15 MHz.

If **CPS**= 1 (which is a value of 2), then ADCCLK = 7.5 MHz.

If **ACQ_PS** = 0 (which is a value of 1), then the sample/hold period is 1 ADCCLK tick, or .1333 microseconds.

Example 2:

If the HISPCLK = 30 MHz, and **ADCLKPS**=1 (which is a value of 2), the result is 15 MHz.

If **CPS**= 1 (which is a value of 2), then ADCCLK = 7.5 MHz.

If **ACQ_PS** = 15 (which is a value of 16), then the sample/hold period is 16 ADCCLK ticks, or 2.1333 microseconds.

Note HISPCLK is set automatically for the user, and it is not possible to change the rate. For more information, see "High-Speed Peripheral Clock" on page 1-11

CAN Calibration Protocol with Third Party Tools

Warning Support for Monitor & Tune (External mode) with Communication interface as **Serial** and **CAN Calibration Protocol (CCP)** will be removed in a future release of MATLAB. It is recommended that you use Monitor and Tune with Communication interface as **XCP on Serial** and **XCP on CAN**.

Embedded Coder allows an ASAP2 data definition file to be generated during the code generation process. This file can be used by a third-party tool to access data from the real-time application while it is executing.

ASAP2 is a data definition standard by the Association for Standardization of Automation and Measuring Systems (ASAM). ASAP2 is a standard description for data measurement, calibration, and diagnostic systems. Embedded Coder software lets you export an ASAP2 file containing information about your model during the code generation process.

Before you generate ASAP2 files with Embedded Coder software, see “Generating an ASAP2 File” in the Simulink Coder™ help. The help describes how to define the signal and parameter information required by the ASAP2 file generation process.

Before the build process, select the ASAP2 option:

1. Select **Simulation** > **Model Configuration Parameters**.

   The Configuration Parameters dialog box appears.
2. In the Configuration Parameters dialog box, select Code Generation > Interface pane.
3. From the **Interface** drop-down list, in the **Data exchange** frame, select the `ASAP2` option.
4. Click **Apply**.

The build process creates an ASAM-compliant ASAP2 data definition file for the generated C code.

- The standard ASAP2 file generation does not include the memory address attributes in the generated file. Instead, it leaves a placeholder that you must replace with the actual address by postprocessing the generated file.
- The map file options in the project template has to be set up a certain way for this procedure to work. If you have created your own project templates, and you do not have the correct settings, you see the following instructions:

  ```
  Warning: It was not possible to do ASAP2 processing on your
  .map file.This is because your IDE project template is not
  configured to generate a .map file in the correct format.
  To generate a .map file in the correct format you need to
  setup the following options in your IDE project template:
  Generate section map should be checked on
  Generate register map should be checked off
  Generate symbol table should be checked on
  Format list file into pages should be checked off
  Generate summary should be checked off
  Page width should be equal to 132 characters
  Symbol colums should be 1
  You can change these options via Project -> Project Options
   -> Linker/Locator -> Map File -> Map File Format.
  ```

Embedded Coder software performs this postprocessing for you. To do the postprocessing, it first extracts the memory address information from the map file generated during the link process. Then, it replaces the placeholders in the ASAP2 file with the actual memory addresses. This postprocessing is performed automatically.

Using the IQmath Library

In this section...

About the IQmath Library

Introduction

The C28x IQmath Library blocks perform processor-optimized fixed-point mathematical operations. These blocks correspond to functions in the Texas Instruments C28x IQmath Library, an assembly-code library for the TI C28x family of digital signal processors.

Note Implementation of this library for the TI C28x processor produces the same simulation and code-generation output as the TI version of this library, but it does not use a global Q value, as does the TI version. The Q format is dynamically adjusted based on the Q format of the input data.

The IQmath Library blocks generally input and output fixed-point data types and use numbers in Q format. The C28x IQmath Library block reference pages discuss the data types accepted and produced by each block in the library. For more information, consult the "Fixed-Point Numbers" on page 1-37 and "Q Format Notation" on page 1-38 topics, as well as the Fixed-Point Designer™ product documentation, which includes more information on fixed-point data types, scaling, and precision issues.

You can use IQmath Library blocks with some core Simulink blocks and Fixed-Point Designer blocks to run simulations in Simulink models before generating code. Once you develop your model, you can generate equivalent code that is optimized to run on a TI C28x™ DSP. During code generation, a call is made to the IQmath Library for each IQmath Library block in your model to create target-optimized code. To learn more about creating models that include IQmath Library blocks and blocks from other blocksets, consult "Building Models" on page 1-40.

Common Characteristics

The following characteristics are common to IQmath Library blocks:

- Sample times are inherited from driving blocks.
- Blocks are single rate.
- Parameters are not tunable.
- Blocks support discrete sample times.

To learn more about characteristics particular to each block in the library, see the individual block reference pages.

References

For detailed information on the IQmath library, see the user guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments website. The user guide is included in the zip file download that also contains the IQmath library (registration required).

Fixed-Point Numbers

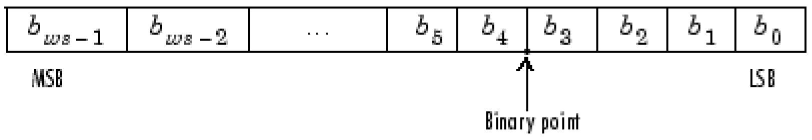
- "Notation" on page 1-37
- "Signed Fixed-Point Numbers" on page 1-37
- "Q Format Notation" on page 1-38

Notation

In digital hardware, numbers are stored in binary words. A binary word is a fixed-length sequence of binary digits (1s and 0s). How hardware components or software functions interpret this sequence of 1s and 0s is defined by the data type.

Binary numbers are used to represent either fixed-point or floating-point data types. A fixed-point data type is characterized by the word size in bits, the binary point, and whether it is signed or unsigned. The position of the binary point is the means by which fixed-point values are scaled and interpreted.

For example, a binary representation of a fractional fixed-point number (either signed or unsigned) is shown below:



where

- b_i is the *i*th binary digit.
- *ws* is the word size in bits.
- b_{ws-1} is the location of the most significant (highest) bit (MSB).
- b_0 is the location of the least significant (lowest) bit (LSB).
- The binary point is shown four places to the left of the LSB. In this example, therefore, the number is said to have four fractional bits, or a fraction length of 4.

Note For Embedded Coder, the results of fixed-point and integer operations in MATLAB/Simulink match the results on the hardware target down to the least significant bit (bit-trueness). The results of floating-point operations in MATLAB/Simulink do not match those on the hardware target, because the libraries used by the third-party compiler may be different from those used by MATLAB/Simulink.

Signed Fixed-Point Numbers

Signed binary fixed-point numbers are typically represented in one of three ways:

- Sign/magnitude
- One's complement
- Two's complement

Two's complement is the most common representation of signed fixed-point numbers and is used by TI digital signal processors.

Negation using signed two's complement representation consists of a bit inversion (translation to one's complement representation) followed by the binary addition of a 1. For example, the two's complement of 000101 is 111011, as follows:

000101 ->111010 (bit inversion) ->111011 (binary addition of a 1 to the LSB)

Q Format Notation

The position of the binary point in a fixed-point number determines how you interpret the scaling of the number. When it performs basic arithmetic such as addition or subtraction, hardware uses the same logic circuits regardless of the value of the scale factor. In essence, the logic circuits do not have knowledge of a binary point. They perform signed or unsigned integer arithmetic — as if the binary point is to the right of b_0. Therefore, you determine the binary point.

In the IQmath Library, the position of the binary point in the signed, fixed-point data types is expressed in and designated by Q format notation. This fixed-point notation takes the form

Qm.n

where

- *Q* designates that the number is in Q format notation — the Texas Instruments representation for signed fixed-point numbers.
- *m* is the number of bits used to designate the two's complement integer portion of the number.
- *n* is the number of bits used to designate the two's complement fractional portion of the number, or the number of bits to the right of the binary point.

In Q format, the most significant bit is designated as the sign bit. Representing a signed fixed-point data type in Q format requires m+n+1 bits to account for the sign.

Note The range and resolution varies for different Q formats. For specific details, see Section 3.2 in the *Texas Instruments C28x Foundation Software, IQmath Library Module User's Guide.*

When converting from Q format to floating-point format, the accuracy of the conversion depends on the values and formats of the numbers. For example, for single-precision floating-point numbers that use 24 bits, the resolution of the corresponding 32-bit number cannot be achieved. The 24-bit number approximates its value by truncating the lower end. For example:
32-bit integer 11110000 11001100 10101010 00001111
Single-precision float +1.1110000 11001100 10101010 x 231
Corresponding value 11110000 11001100 10101010 00000000

Expressing Q Format — Q.15

For example, a signed 16-bit number with n = 15 bits to the right of the binary point is expressed as

```
Q0.15
```

in this notation. This is (1 sign bit) + (m = 0 integer bits) + (n = 15 fractional bits) = 16 bits total in the data type. In Q format notation, the m = 0 is often implied, as in

```
Q.15
```

In Fixed-Point Designer software, this data type is expressed as

```
sfrac16
```

or

```
sfix16_En15
```

In DSP System Toolbox™ software, this data type is expressed as

```
[16 15]
```

Expressing Q Format — Q1.30

Multiplying two Q0.15 numbers yields a product that is a signed 32-bit data type with n = 30 bits to the right of the binary point. One bit is the designated sign bit, thereby forcing m to be 1:

m+n+1 = 1+30+1 = 32 bits total

Therefore, this number is expressed as

```
Q1.30
```

In Fixed-Point Designer software, this data type is expressed as

```
sfix32_En30
```

In DSP System Toolbox software, this data type is expressed as

```
[32 30]
```

Expressing Q Format — Q-2.17

Consider a signed 16-bit number with a scaling of $2^{(-17)}$. This requires n = 17 bits to the right of the binary point, meaning that the most significant bit is a *sign-extended* bit.

Sign extension fills additional bits with the value of the MSB. For example, consider a 4-bit two's complement number 1011. When this number is extended to 7 bits with sign extension, the number becomes 1111101 and the value of the number remains the same.

One bit is the designated sign bit, forcing m to be -2:

m+n+1 = -2+17+1 = 16 bits total

Therefore, this number is expressed as

```
Q-2.17
```

In Fixed-Point Designer software, this data type is expressed as

```
sfix16_En17
```

In DSP System Toolbox software, this data type is expressed as

```
[16 17]
```

Expressing Q Format — Q17.-2

Consider a signed 16-bit number with a scaling of 2^(2) or 4. This means that the binary point is implied to be 2 bits to the right of the 16 bits, or that there are n = -2 bits to the right of the binary point. One bit must be the sign bit, thereby forcing m to be 17:

m+n+1 = 17+(-2)+1 = 16

Therefore, this number is expressed as

```
Q17.-2
```

In Fixed-Point Designer software, this data type is expressed as

```
sfix16_E2
```

In DSP System Toolbox software, this data type is expressed as

```
[16 -2]
```

Building Models

- “Overview” on page 1-40
- “Converting Data Types” on page 1-40
- “Using Sources and Sinks” on page 1-41
- “Choosing Blocks to Optimize Code” on page 1-41
- “Double and Single-Precision Parameter Values” on page 1-41

Overview

You can use IQmath Library blocks in models along with certain core Simulink, Fixed-Point Designer, and other blockset blocks. This section discusses issues you should consider when building a model with blocks from these different libraries.

Converting Data Types

It is vital to make sure that blocks you connect in a model have compatible input and output data types. In most cases, IQmath Library blocks handle only a limited number of specific data types. You can refer to the block reference page for a discussion of the data types that the block accepts and produces.

When you connect IQmath Library blocks and Fixed-Point Designer blocks, you often need to set the data type and scaling in the block parameters of the Fixed-Point Designer block to match the data type of the IQmath Library block. Many Fixed-Point Designer blocks allow you to set their data type and scaling through inheritance from the driving block, or through backpropagation from the next block. This can be a good way to set the data type of a Fixed-Point Designer block to match a connected IQmath Library block.

Some DSP System Toolbox blocks and core Simulink blocks also accept fixed-point data types. Choose the right settings in these blocks' parameters when you connect them to an IQmath Library block.

Using Sources and Sinks

The IQmath Library does not include source or sink blocks. Use source or sink blocks from the core Simulink library or Fixed-Point Designer in your models with IQmath Library blocks.

Choosing Blocks to Optimize Code

In some cases, blocks that perform similar functions appear in more than one blockset. For example, the IQmath Library and Fixed-Point Designer software have a Multiply block. When you are building a model to run on C2000 DSP, choosing the block from the IQmath Library yields better optimized code. You can use a similar block from another library if it gives you functionality that the IQmath Library block does not support, but you will generate code that is less optimized.

Double and Single-Precision Parameter Values

When you enter double-precision floating-point values for parameters in the IQ Math blocks, the software converts them to single-precision values that are compatible with the behavior on C28x processor. For example, with the **Ramp Generator** block, the software converts the value of the **Maximum step angle** parameter to a single-precision value.

Configuring LIN Communications

In this section...

Overview

The LIN communications architecture supports a single master node and up to 16 slave nodes on a LIN network.

LIN nodes use message frames to exchange data. The message has two parts:

- Frame header, generated by the Master node.
- Frame response, which contains data generated by either Slave node or a slave task on a Master node (but not both).

Configuring Your Model

First, study, and understand the LIN addressing system. See the "Message Filtering and Validation" topic in the TMS320F2803x Piccolo Local Interconnect Network (LIN) Module, Literature Number: SPRUGE2A.

Configure the LIN node in your model as a master or slave node:

1. Configure the Target Hardware Resources tab, as described in "Configure Target Hardware Resources" (Embedded Coder).
2. In the Target Hardware Resources tab, select the **Peripherals** tab, and then select **LIN**.
3. Set **LIN mode** to `Master` or `Slave`.

If the LIN node is a Master node:

- Add a LIN Transmit block to the model. This block enables the Master to generate message headers.
- To send data, set the **ID** input and **Tx ID Mask** input to make Tx ID Match happen on this node.
- To receive data, place LIN Receive block in the model. Set the **Rx ID Mask** input to make Rx ID Match happen on this node.

For example, to configure a model with a master node that receives data from a slave node:

- Add a LIN Transmit block and a LIN Receive block to the model.
- In the Target Hardware Resources tab, configure the **ID Slave Task Byte**.
- For the LIN Transmit block, set the **ID** input.
- For the LIN Receive block, set the **Rx ID Mask** input so that: **Rx ID Mask** = **ID** XOR **Slave Task ID Byte**.

If the LIN node is a Slave node:

- To send data, place LIN Transmit block in the model. Set the **ID** input to match the LIN frame header issued by the remote Master. Set **Tx ID Mask** to make a Tx ID Match happen on this node.
- To receive data, place LIN Receive block in the model. Set the **Rx ID Mask** input to make an Rx ID Match happen on this node.

For example, to configure a model with a slave node that transmits data to a master node:

- Add a LIN Transmit block to the model.
- In the Target Hardware Resources tab, configure the **ID byte** or **ID Slave Task Byte** (depending on the **ID filtering** option).
- In the LIN Transmit block, set the **ID** input and **Tx ID Mask** input so that: **Tx ID Mask = ID** XOR (**ID Byte** or **ID Slave Task Byte**).

Set the **Data type** and **Data length** values in your LIN Receive blocks to match the type and length of the transmitted data. These values enable the receive block reconstruct the data from the message frames.

Note The LIN Transmit block inherits the data type and length from its input.

Tips and Limitations

Signal Monitoring and Parameter Tuning over XCP on CAN Using Third-Party Calibration Tools

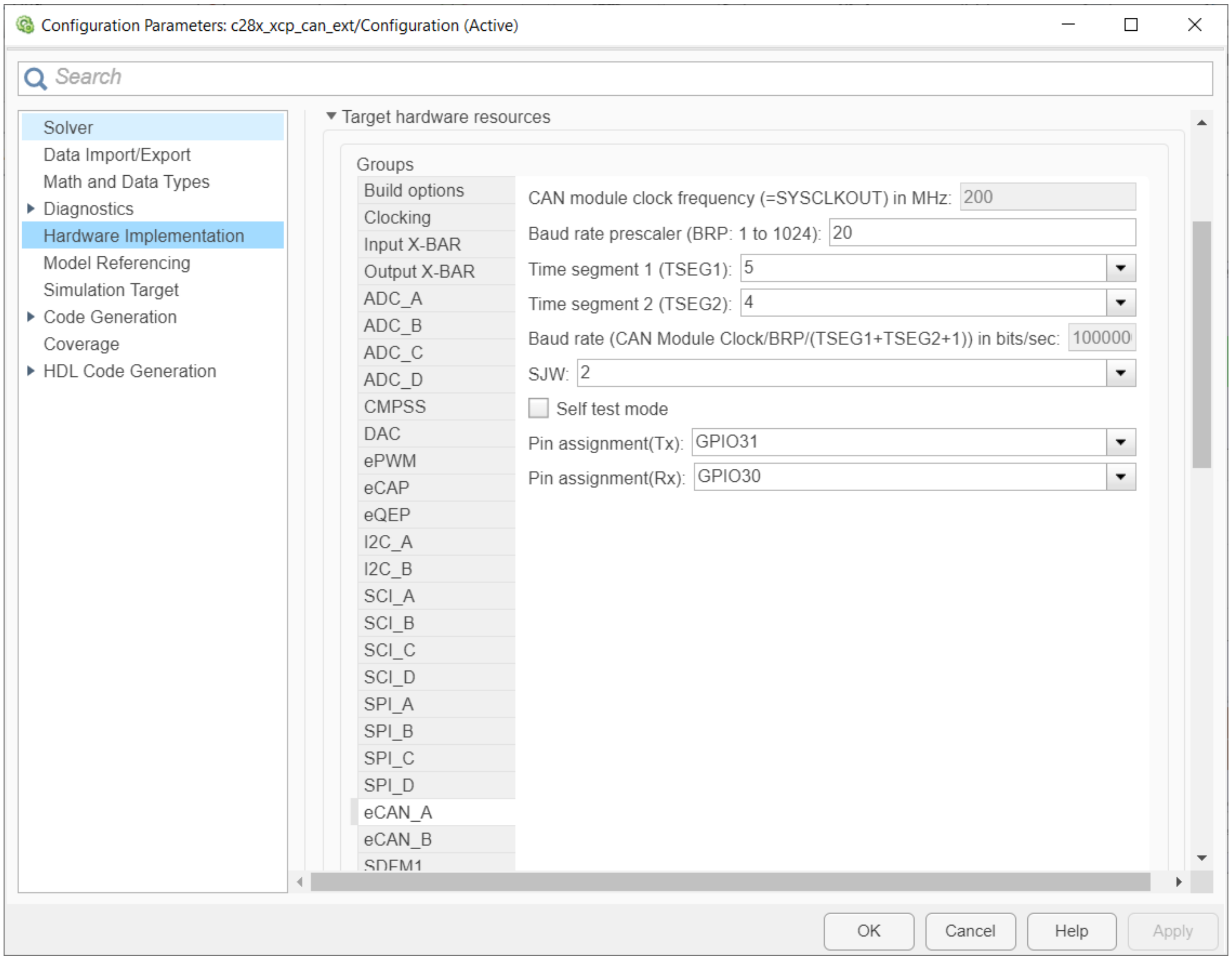
You can use XCP-based External mode simulation over CAN to connect to the Texas Instruments C2000 board from a third-party calibration software such as CANape and perform signal monitoring and parameter calibration. For information to configure the external mode with any available eCAN module, see “Set Up CAN Communication with Target Hardware” on page 1-8.

Note Refer to the corresponding third-party tools vendor to know more about the supported platforms.

To configure the Simulink model for signal monitoring and parameter tuning using third-party calibration tools:

1. In the Simulink model, enable the signals for logging. For more information, see Configure a Signal for Logging.
2. Identify the parameters for tuning. For more information, see Create Tunable Calibration Parameter in the Generated Code (Simulink Coder).
3. Open Configuration Parameters dialog box, go to the **Hardware Implementation** pane and select the connected Texas Instruments C2000 board from the **Hardware board** list.
4. Go to **Target hardware resources** > **eCAN** tab to configure different parameters related to the CAN module on the target.

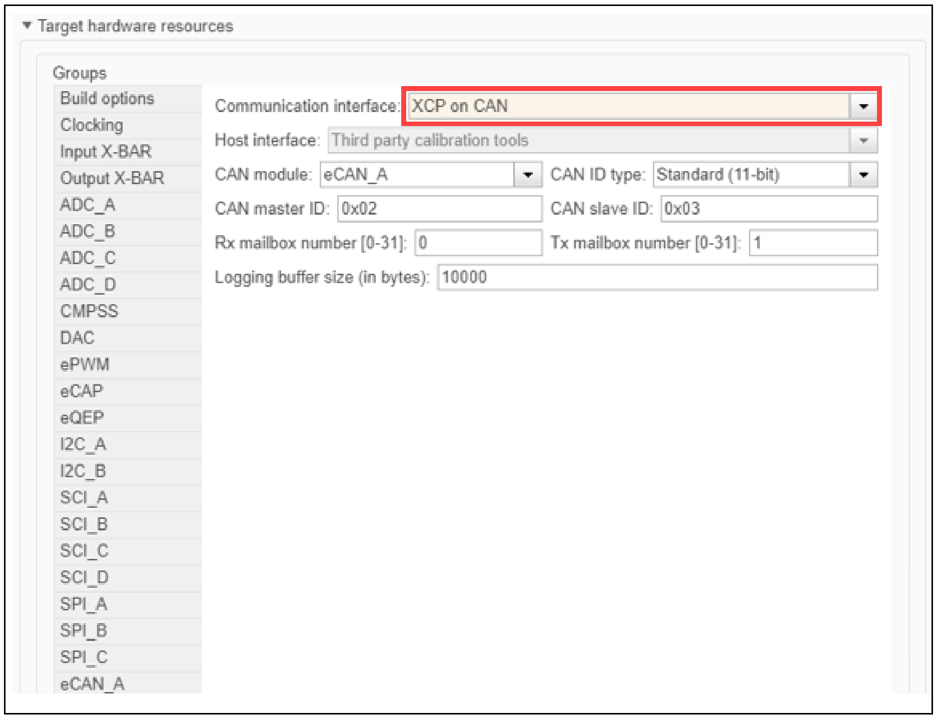
   Note If the target supports multiple eCAN modules, select the eCAN module to be used with external mode and then configure different parameters for the selected CAN module.



CAN module clock frequency (=SYSCLKOUT) in MHz: 200
Baud rate prescaler (BRP: 1 to 1024): 20
Time segment 1 (TSEG1): 5
Time segment 2 (TSEG2): 4
Baud rate (CAN Module Clock/BRP/(TSEG1+TSEG2+1)) in bits/sec: 100000
SJW: 2
Self test mode
Pin assignment(Tx): GPIO31
Pin assignment(Rx): GPIO30

5 Go to **Target hardware resources** > **External mode** tab, and choose `XCP on CAN` as the **Communication interface**.

Note `XCP on CAN` **Communication interface** supports only `Third party calibration tools` as **Host interface**.



Communication interface: XCP on CAN
Host interface: Third party calibration tools
CAN module: eCAN_A
CAN ID type: Standard (11-bit)
CAN master ID: 0x02
CAN slave ID: 0x03
Rx mailbox number [0-31]: 0
Tx mailbox number [0-31]: 1
Logging buffer size (in bytes): 10000

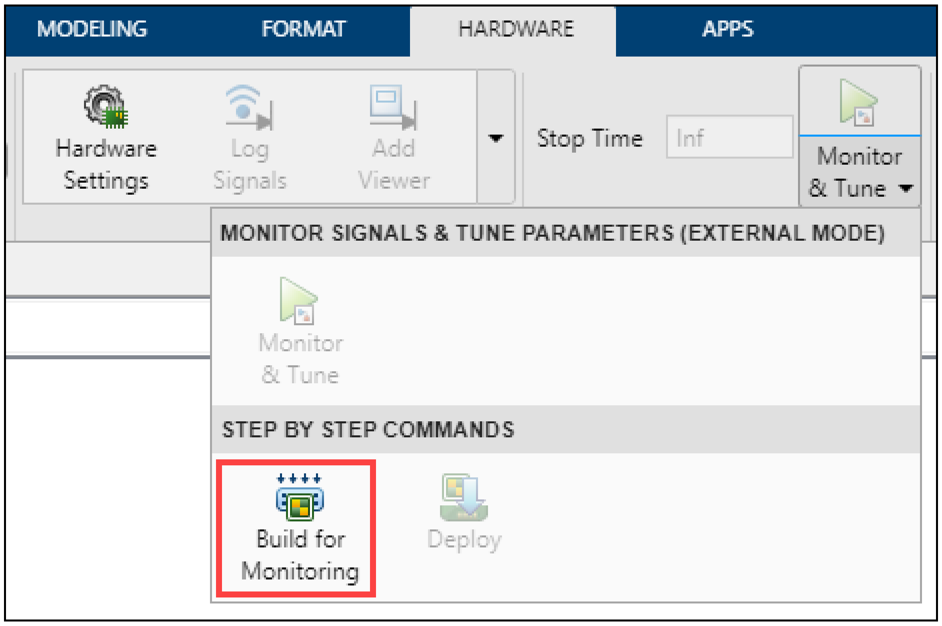
6 Configure the eCAN module on the target.

- **a** If the target supports multiple eCAN modules, select the `CAN module` to be used with external mode.
- **b** Select the `CAN ID type`.
- **c** Enter the values for **CAN Master ID** and **CAN slave ID**.
- **d** Enter the values for **Rx mailbox number** and **Tx mailbox number**.
- **e** Enter a suitable value for **Logging buffer size**.

7 Go to **Code Generation** > **Optimization** and then set **Default parameter behavior** to `Inlined`.

8 Click **Apply** and **OK**.

After you configure the Simulink model, you can initiate the Build for Monitoring action in Simulink and use the A2L file that is generated, for parameter calibration using third-party calibration tools.



1 In the **Hardware** tab of Simulink toolstrip, click **Build for Monitoring**. This action builds and deploys the model on the target Texas Instruments C2000 board, and also generates an A2L file in the current MATLAB folder path. The A2L file contains XCP slave information for using in third-party calibration tools. The file name of the A2L file is in this format: <*modelname*>`.a2l`.

   For more information on A2L file generation, see Export ASAP2 File for Data Measurement and Calibration.

   Note **Monitor & Tune** option is disabled during this process because `XCP on CAN` **Communication interface** supports only `Third party calibration tools` as **Host interface**.

2 Click **Deploy** in the Simulink Toolstrip to deploy the executable onto the target.

3 Import the A2L file into third-party calibration tools, connect to the XCP slave, and start monitoring of signals and calibration of parameters using the interface in third-party calibration tool. For more information, refer to the example “Calibrate ECU Parameters from Third-party Calibration Tools Using XCP-based CAN Interface” on page 4-327.

Supported Objects and Data Types

The supported objects are:

- Simulink.Parameter for parameter tuning
- Simulink.Signal for signal logging

Define data objects for the signals and parameters of interest for ASAP2 file generation. For ease of use, create a MATLAB file to define the data objects so that you only have to set up the objects only once.

To set up tunable parameters and signal logging:

1. Associate the parameters that you want to tune with `Simulink.Parameter` objects with 'ExportedGlobal' storage class. It is important to set the data type and value of the parameter object. For an example of how to create such a `Simulink.Parameter` object for tuning, see the following code:

   ```
   stepSize = Simulink.Parameter;
   stepSize.DataType = 'uint8';
   stepSize.CoderInfo.StorageClass = 'ExportedGlobal';
   stepSize.Value = 1;
   ```
2. Associate the signals that you want to log with `Simulink.Signal` objects. Set the data type of the `Simulink.Signal`. The following code example shows how to declare such a `Simulink.Signal` object for logging.

   ```
   counter = Simulink.Signal;
   counter.DataType = 'uint8';
   ```
3. Associate the data objects that you defined in the MATLAB file with parameters or signals in the model. For the previous code examples, you can set the **Constant value** in a Source block to `stepSize`, and set a **Signal name** to `counter` in the Signal Properties dialog box. The `stepSize` and `counter` are the data objects defined in the code.

The supported data types are:

- uint8, int8
- uint16, int16
- uint32, int32
- single

Note Logging and tuning support of 64-bit double variables is supported only on **TI F2838x (C28x)** board.

Limitations

- Logging and tuning of 32-bit double variables is not supported. Typecast 32-bit double to single variables using Data Type Conversion blocks as logging and tuning of single variables is supported.
- Logging and tuning of complex numbers is not supported. However, it is possible to work with complex numbers by breaking down the complex number into its real and imaginary components. You can perform this breakdown using the following blocks in the Simulink Math Operations library: Complex to Real-Imag, Real-Imag to Complex, Magnitude-Angle to Complex, and Complex to Magnitude-Angle.

See Also

"Data Type Support"

Signal Monitoring and Parameter Tuning over XCP on CAN

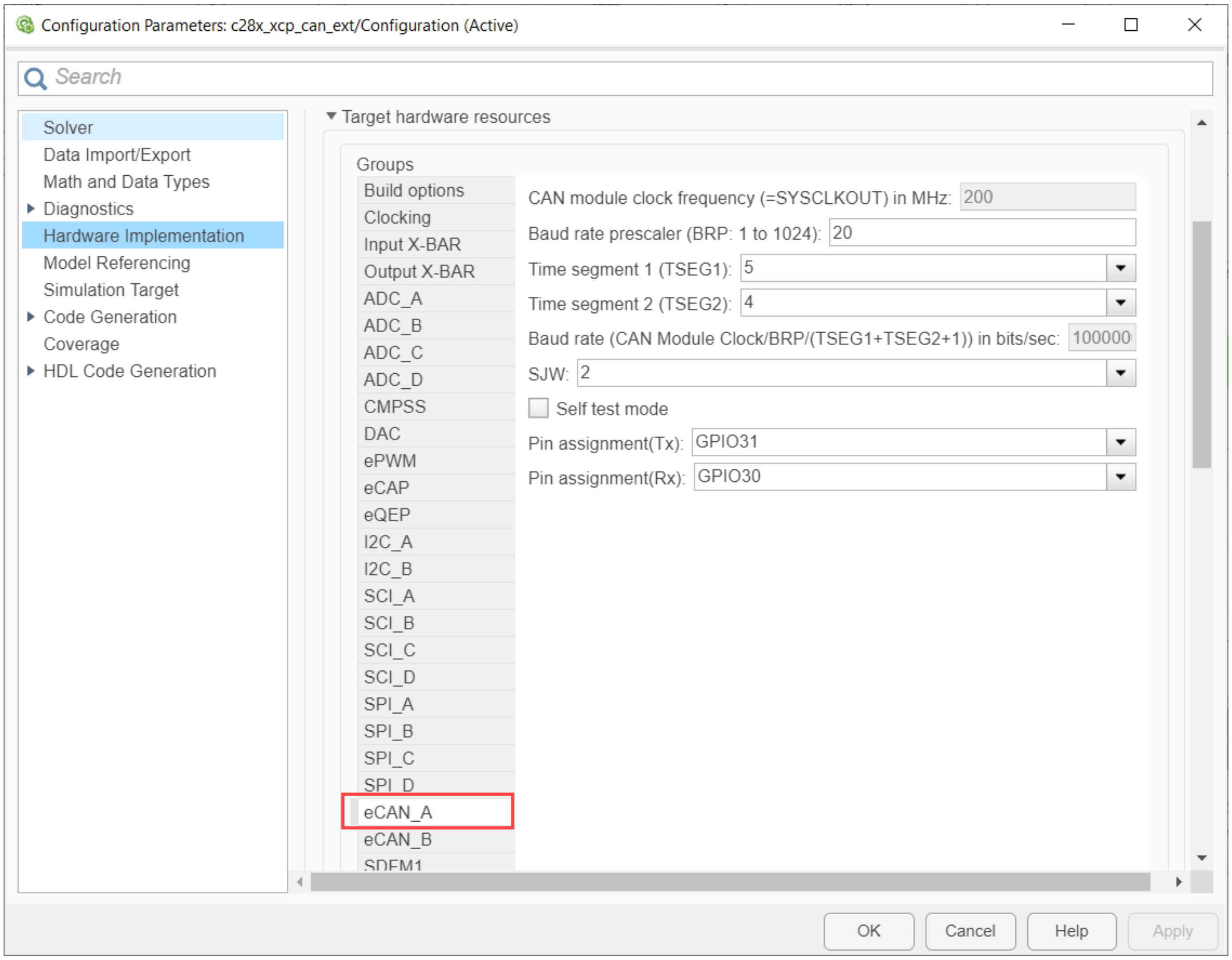
You can use XCP-based External mode simulation over CAN to connect to the Texas Instruments C2000 board and perform signal monitoring and parameter tuning. For information on setting up Vector devices, see “Set Up CAN Communication with Target Hardware” on page 1-8. Multiple Third-party CAN vendors are supported. You can also install device drivers for CAN interfaces from respective Third-party vendor websites.

Note Refer to the corresponding third-party tools vendor to know more about the supported platforms.

To configure the Simulink model for signal monitoring and parameter tuning perform these steps:

1. In the Simulink model, enable the signals for logging. For more information, see Configure a Signal for Logging.
2. Identify the parameters for tuning. For more information, see Create Tunable Calibration Parameter in the Generated Code (Simulink Coder).
3. Open Configuration Parameters dialog box, go to the **Hardware Implementation** pane and select the connected Texas Instruments C2000 board from the **Hardware board** list.
4. Go to **Target hardware resources > eCAN** tab to configure different parameters related to the CAN module on the target.

   Note If the target supports multiple eCAN modules, select the eCAN module to be used with external mode and then configure different parameters for the selected CAN module.

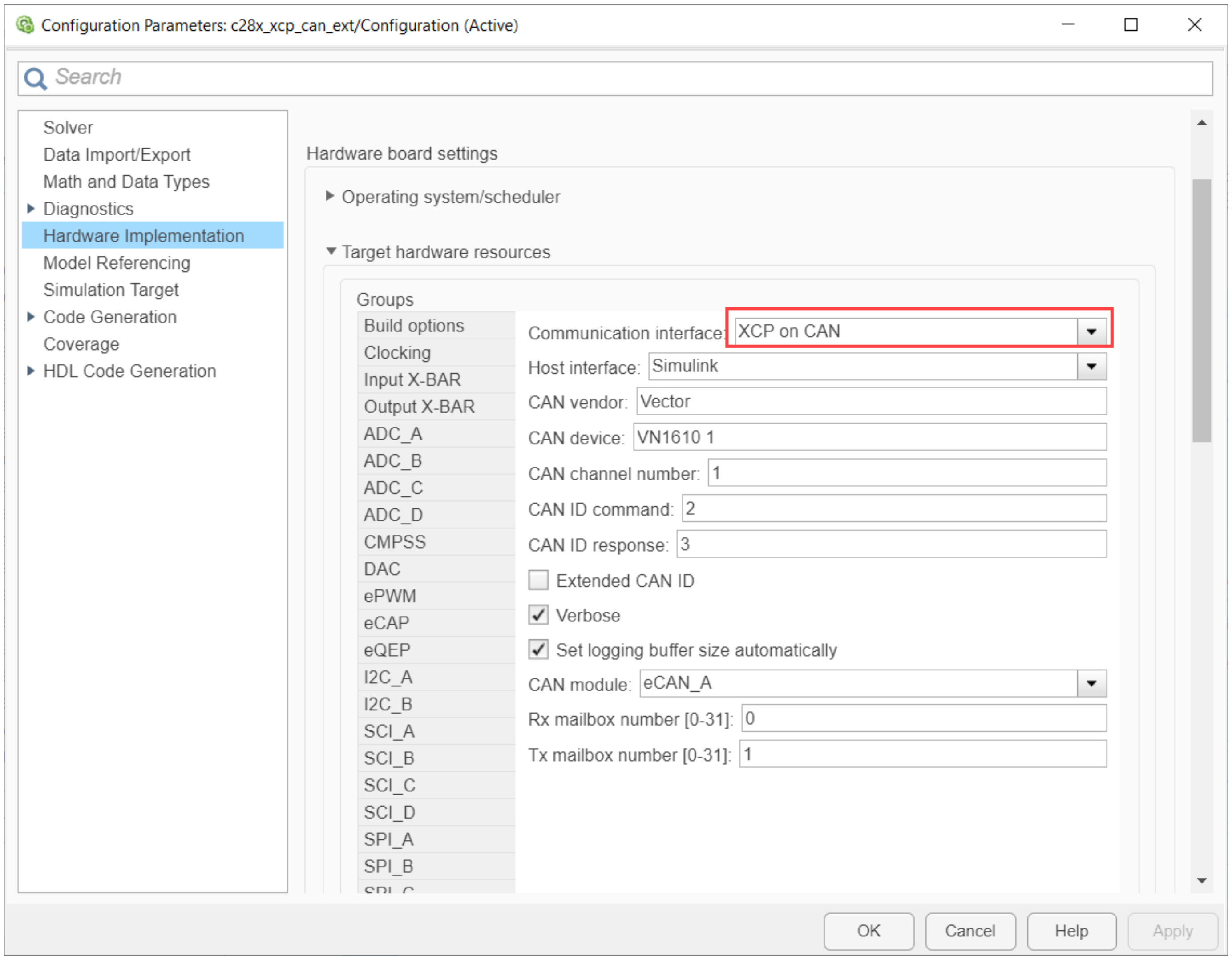

Configuration Parameters: c28x_xcp_can_ext/Configuration (Active)

Search

Target hardware resources

CAN module clock frequency (=SYSCLKOUT) in MHz: 200
Baud rate prescaler (BRP: 1 to 1024): 20
Time segment 1 (TSEG1): 5
Time segment 2 (TSEG2): 4
Baud rate (CAN Module Clock/BRP/(TSEG1+TSEG2+1)) in bits/sec: 100000
SJW: 2
☐ Self test mode
Pin assignment(Tx): GPIO31
Pin assignment(Rx): GPIO30

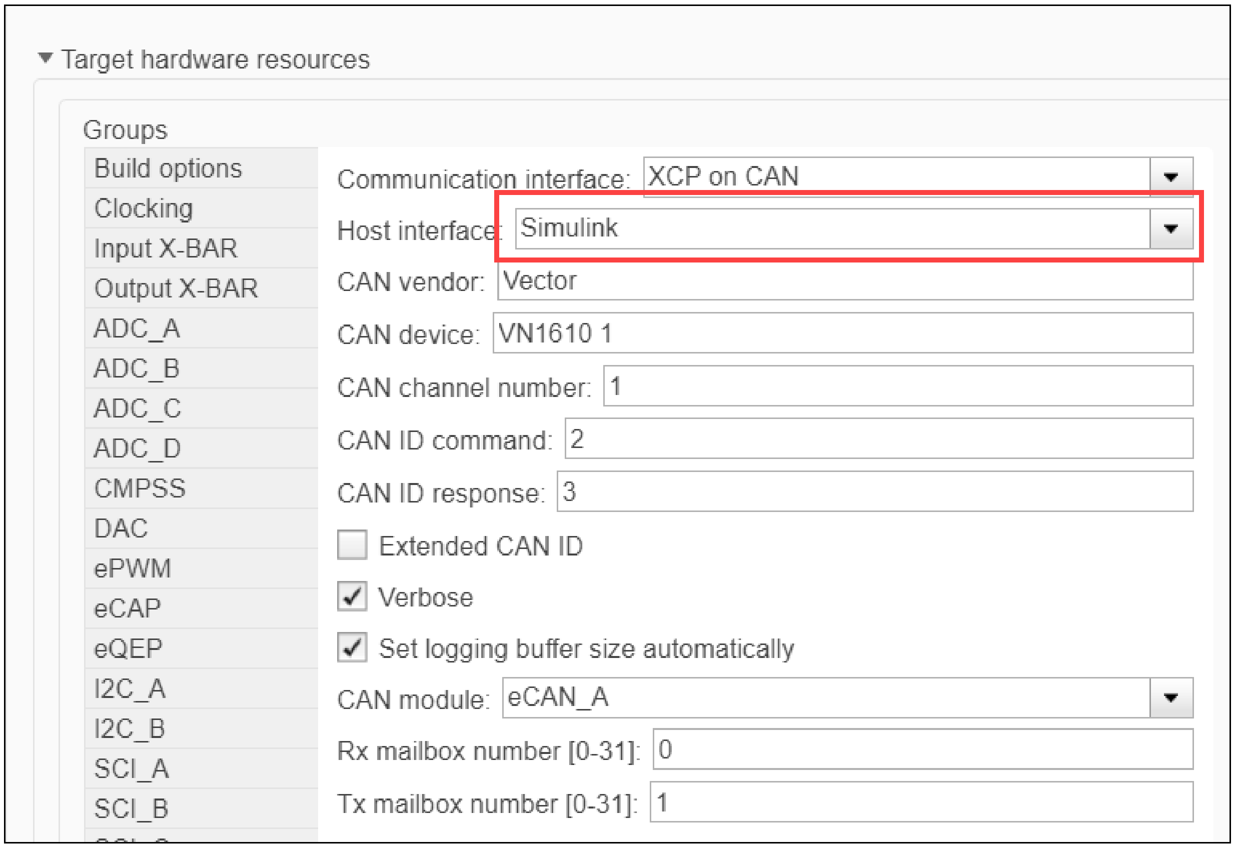
5 Go to **Target hardware resources** > **External mode** tab, and choose `XCP on CAN` as the **Communication interface**.



Communication interface: XCP on CAN
Host interface: Simulink
CAN vendor: Vector
CAN device: VN1610 1
CAN channel number: 1
CAN ID command: 2
CAN ID response: 3
Extended CAN ID: ☐
Verbose: ✓
Set logging buffer size automatically: ✓
CAN module: eCAN_A
Rx mailbox number [0-31]: 0
Tx mailbox number [0-31]: 1

6 Select the required option for **Host interface**.

- To perform signal monitoring and parameter tuning using Simulink, select `Simulink` as the **Host interface**.

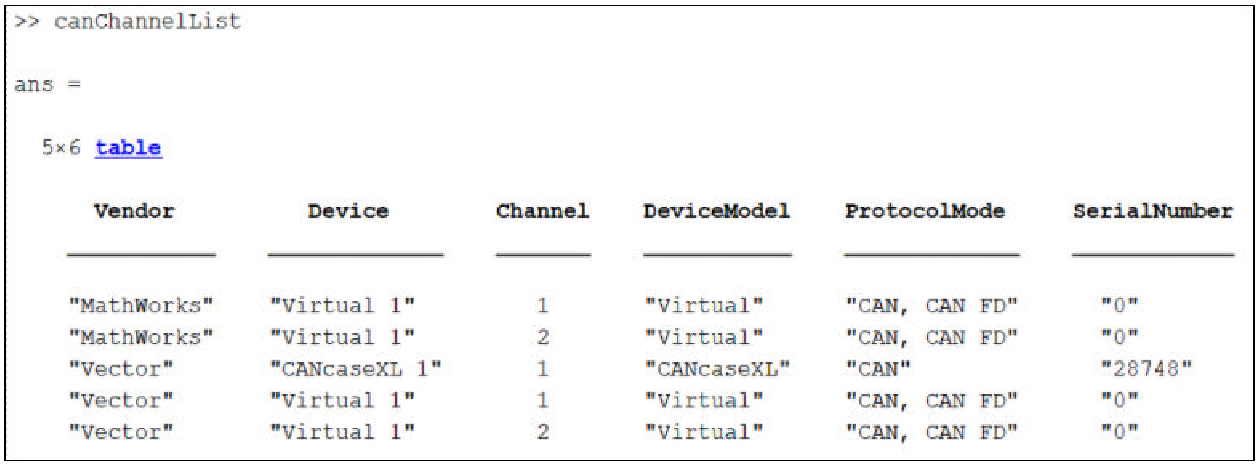


Communication interface: XCP on CAN
Host interface: Simulink
CAN vendor: Vector
CAN device: VN1610 1
CAN channel number: 1
CAN ID command: 2
CAN ID response: 3
☐ Extended CAN ID
☑ Verbose
☑ Set logging buffer size automatically
CAN module: eCAN_A
Rx mailbox number [0-31]: 0
Tx mailbox number [0-31]: 1

Note Vehicle Network Toolbox™ is required to perform external mode simulation using XCP on CAN and `Simulink` as **Host interface**.

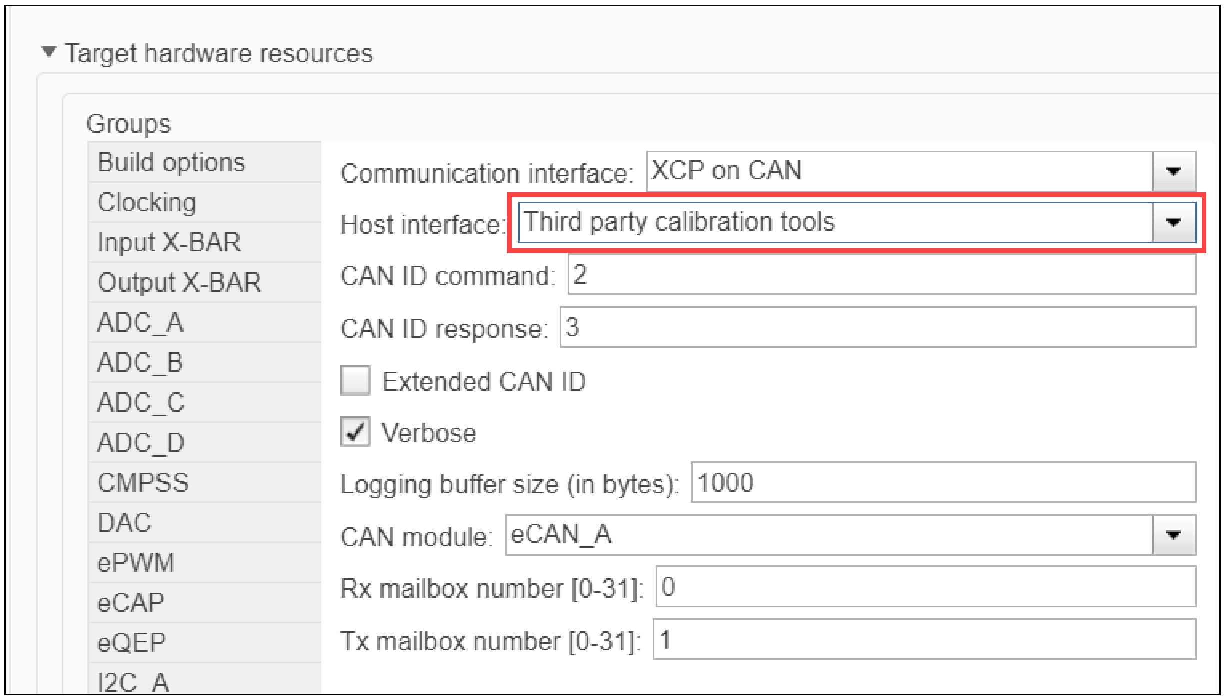
- Enter the values for **CAN vendor**, **CAN device**,and **CAN channel number**.

  Use theVehicle Network Toolbox function `canChannelList()` to get values for CAN vendor, CAN device, CAN channel . This function returns a list of all the CAN interfaces whose drivers have been installed and connected to the computer. In the MATLAB command window, type `canChannelList()` and press enter. A sample screen is shown here.



```
>> canChannelList

ans =

  5×6 table

      Vendor          Device        Channel     DeviceModel      ProtocolMode      SerialNumber
    ___________    _____________    _______    ___________    _______________    ____________

    "MathWorks"    "Virtual 1"         1       "Virtual"      "CAN, CAN FD"        "0"
    "MathWorks"    "Virtual 1"         2       "Virtual"      "CAN, CAN FD"        "0"
    "Vector"       "CANcaseXL 1"       1       "CANcaseXL"    "CAN"                "28748"
    "Vector"       "Virtual 1"         1       "Virtual"      "CAN, CAN FD"        "0"
    "Vector"       "Virtual 1"         2       "Virtual"      "CAN, CAN FD"        "0"
```

- To perform signal monitoring and parameter tuning using third-party calibration software such as CANape , select `Third party calibration tools` as the **Host interface**.



Communication interface: XCP on CAN
Host interface: Third party calibration tools
CAN ID command: 2
CAN ID response: 3
☐ Extended CAN ID
☑ Verbose
Logging buffer size (in bytes): 1000
CAN module: eCAN_A
Rx mailbox number [0-31]: 0
Tx mailbox number [0-31]: 1

7 Configure the eCAN module on the target.

   a Enter the values for **CAN ID command** and **CAN ID response**.

   b Select **Extended CAN ID** option, if you want to use extended ID.

   c Select **Verbose** option to view the External Mode execution progress and updates in the Diagnostic Viewer or in the MATLAB Command Window.

   d Select **Set logging buffer size automatically** option to set the optimal buffer size for logging data.

   e Select the **CAN module** to use with external mode.

   f Enter the values for **Rx mailbox number** and **Tx mailbox number**.

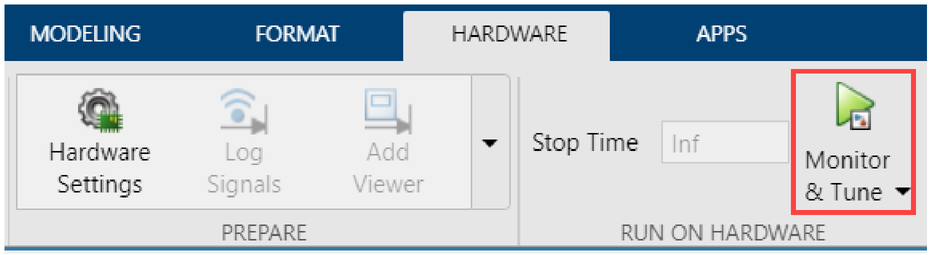
8 Click **Apply** and **OK**.

After you configure the Simulink model, initiate the Parameter Tuning and Signal Logging action using the steps listed in “Model with Simulink as Host Interface” on page 1-55. If you are using a third-party calibration software such as CANape to perform signal monitoring and parameter tuning, then perform the steps listed in “Model with Third party calibration tools as Host Interface” on page 1-55.

Deploy Model on Target Hardware

Model with Simulink as Host Interface

After you configure the Simulink model, initiate the Parameter Tuning and Signal Logging action. In the **Hardware** tab of Simulink toolstrip, click **Monitor & Tune**.



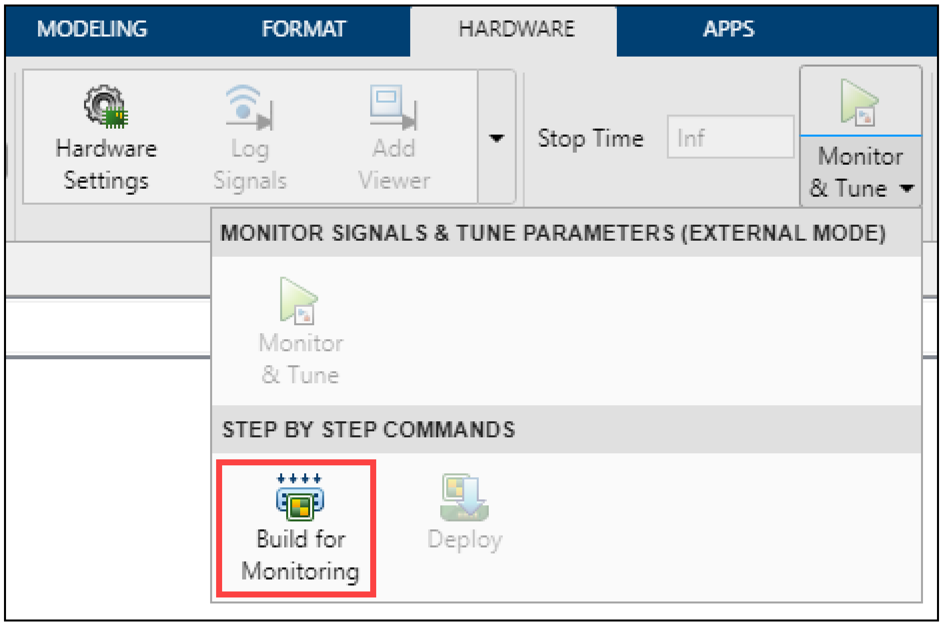
For more information, see the example “Signal Monitoring and Parameter Tuning Over XCP-based CAN Interface” on page 4-333.

Model with Third party calibration tools as Host Interface

To configure the Simulink model for signal monitoring and parameter tuning using third-party calibration tools perform these steps:

1 In the **Hardware** tab of Simulink toolstrip, click **Build for Monitoring**. This action builds and deploys the model on the target Texas Instruments C2000 board, and also generates an A2L file in the current MATLAB folder path. The A2L file contains XCP slave information for using in third-party calibration tools. The file name of the A2L file is in this format: <*modelname*>`.a2l`.

   For more information on A2L file generation, see Export ASAP2 File for Data Measurement and Calibration.



2. Click **Deploy** in the Simulink Toolstrip to deploy the executable onto the target.
3. Import the A2L file into third-party calibration tools, connect to the XCP slave, and start monitoring of signals and calibration of parameters using the interface in third-party calibration tool. For more information, see the example "Calibrate ECU Parameters from Third-party Calibration Tools Using XCP-based CAN Interface" on page 4-327.

You can use XCP-based External mode simulation over CAN to connect to the Texas Instruments C2000 board from a third-party calibration software such as CANape and perform signal monitoring and parameter calibration. For information to configure the external mode with any available eCAN module, see "Set Up CAN Communication with Target Hardware" on page 1-8.

Supported Objects and Data Types

The supported objects and data types for Signal Monitoring and Parameter Tuning over XCP on CAN using Simulink and third-Party Calibration Tools are:

- Simulink.Parameter for parameter tuning
- Simulink.Signal for signal logging

Define data objects for the signals and parameters. For ease of use, create a MATLAB file to define the data objects so that you only have to set up the objects only once.

To set up tunable parameters and signal logging:

1. Associate the parameters that you want to tune with `Simulink.Parameter` objects with 'ExportedGlobal' storage class. It is important to set the data type and value of the parameter object. For an example of how to create such a `Simulink.Parameter` object for tuning, see the following code:

   ```
   stepSize = Simulink.Parameter;
   stepSize.DataType = 'uint8';
   stepSize.CoderInfo.StorageClass = 'ExportedGlobal';
   stepSize.Value = 1;
   ```

2 Associate the signals that you want to log with `Simulink.Signal` objects. Set the data type of the `Simulink.Signal`. The following code example shows how to declare such a `Simulink.Signal` object for logging.

```
counter = Simulink.Signal;
counter.DataType = 'uint8';
```

3 Associate the data objects that you defined in the MATLAB file with parameters or signals in the model. For the previous code examples, you can set the **Constant value** in a Source block to `stepSize`, and set a **Signal name** to `counter` in the Signal Properties dialog box. The `stepSize` and `counter` are the data objects defined in the code.

The supported data types are:

- uint8, int8
- uint16, int16
- uint32, int32
- single

Note Logging and tuning support of 64-bit double variables is supported only on **TI F2838x (C28x)** board.

Troubleshooting

Memory overflow issue1 during compilation

Description

The following error might occur due to insufficient memory:

```
program will not fit into available memory, or the section contains a call
site that requires a trampoline that can't be generated for this section.
run placement with alignment/blocking fails for section ".ebss" size
0x98b1page 1.  Available memory ranges:
DRAM         size: 0x6000       unused: 0x6000       max hole: 0x6000
```

Action

This error occurs if the model contains a lot of signals and more memory is required than available on the target to log all the signals. To resolve the issue, reduce the number of signals being logged so that the memory available on the target is sufficient.

Memory overflow issue2 during compilation

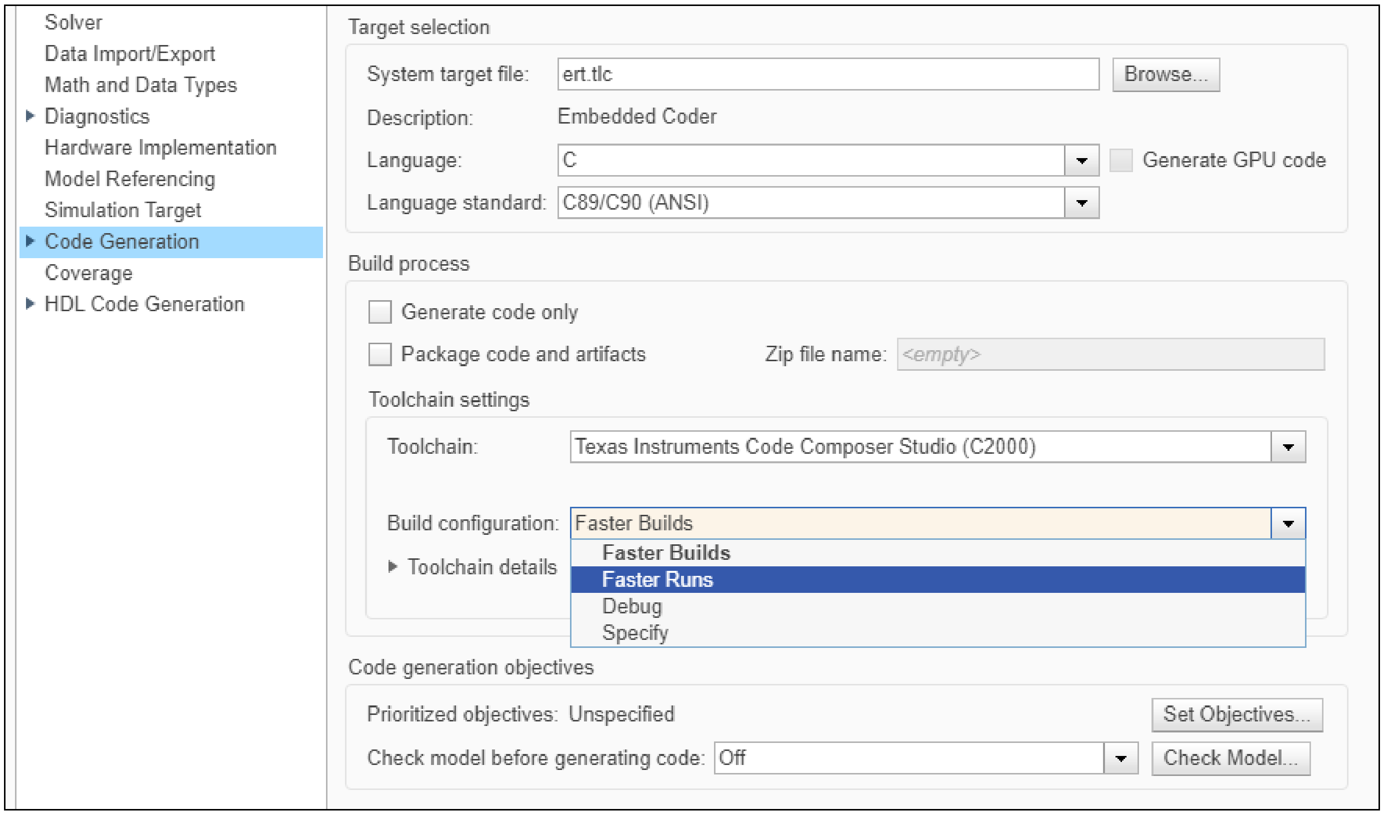
Description

The following error might occur due to insufficient memory:

```
program will not fit into available memory, or the section contains a call site that
requires a trampoline that can't be generated for this section. placement with
alignment/blocking fails for section ".text" size 0x360apage 0.
Available memory ranges: RAMLS_PROG size: 0x3000 unused: 0x2bd7 max hole: 0x2bd6
```

Action

This error shown at linking stage indicates that the memory is insufficient to fit the code section (`.text`) in the target memory available. External Mode simulation requires additional code to enable the communication between Simulink and the target, which might lead to this code section overflow. To mitigate this issue, select `Faster Runs` as the **Build Configuration** in the **Configuration Parameters** screen. This option compiles the code at `-O3` optimization level, resulting in a smaller code size which might fit the target memory.



Target selection

System target file: ert.tlc

Description: Embedded Coder

Language: C

Generate GPU code

Language standard: C89/C90 (ANSI)

Build process

Generate code only

Package code and artifacts

Zip file name: <empty>

Toolchain settings

Toolchain: Texas Instruments Code Composer Studio (C2000)

Build configuration: Faster Builds

Toolchain details

Code generation objectives

Prioritized objectives: Unspecified

Check model before generating code: Off

See Also

"Data Type Support"

Related Examples

- "Signal Monitoring and Parameter Tuning Over XCP-based CAN Interface" on page 4-333

Signal Monitoring and Parameter Tuning over XCP on Serial

In this section...

You can use Monitor and Tune (External Mode) action to tune parameters and monitor a Simulink model running on your target hardware.

Monitor and Tune enables you to tune model parameters and evaluate the effects of different parameter values on model results in real-time. When you change parameter values in a model, the modified parameter values are communicated to the target hardware immediately. You can monitor the effects of different parameter values by viewing the output signals on Sink blocks or in Simulation Data Inspector (SDI). Doing so helps you find the optimal values for performance. This process is called parameter tuning.

Monitor and Tune accelerates parameter tuning. You do not have to rerun the model each time you change parameters. You can also use Monitor and Tune to develop and validate your model using the actual data and hardware for which it is designed. This software-hardware interaction is not available solely by simulating a model.

For external mode with serial communication, you have a ready-to-use COM port on the USB drive if your target hardware supports it or the COM1 port on your computer with RS–232 cable. For more information on how to set up serial communication, see "Set Up Serial Communication with Target Hardware" on page 1-6. You can configure the external mode for different baud rates that your communication interface supports.

The blockset supports Monitor and Tune simulation over `XCP on Serial` communication interface.

In Universal Measurement and Calibration Protocol (XCP)-based External mode simulation over serial connections, you can use:

- "Dashboard" blocks: In addition to "Sources" and Sink blocks, you can use "Dashboard" blocks to change parameter values and to monitor the effects of parameter tuning. The Dashboard library contains set of blocks using which you can interactively control and visualize the model.
- Simulation Data Inspector (SDI): You can inspect and compare data from multiple simulations to validate model designs using Simulation Data Inspector.

Prepare a Simulink Model for External Mode

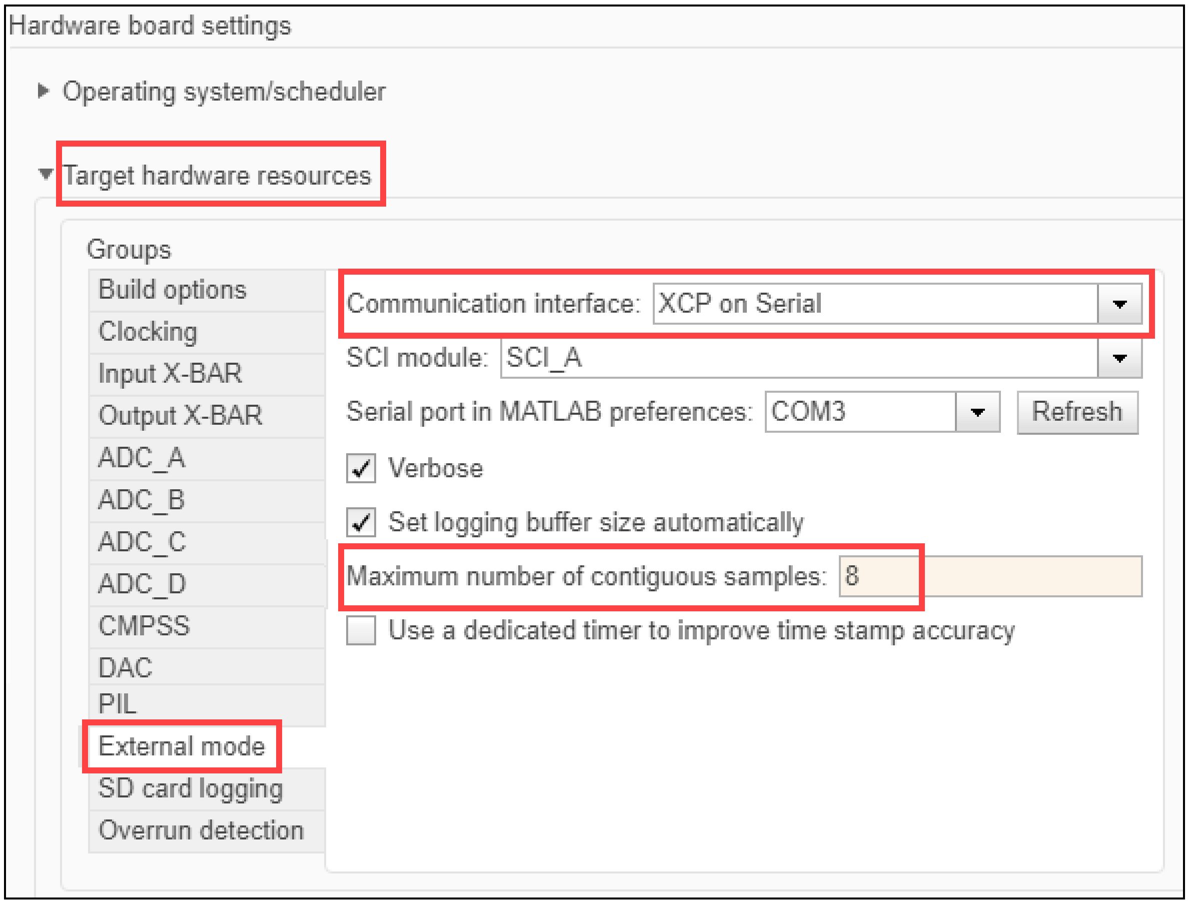
This section explains how to prepare a Simulink model to run in External mode.

To prepare your model for external mode/PIL with serial communication, refer "Serial Configuration for External Mode and PIL".

1. Configure the hardware network as described in "Hardware Setup for C2000 Microcontroller Blockset" on page 1-3.
2. Create or open a Simulink model.

3 Set the Simulation stop time parameter, located to the left of Simulation mode.

4 Configure the Model Configuration Parameters for the hardware as described in “Model Configuration Parameters for Texas Instruments C2000 Processors”. In **External mode**, set the **Communication interface** parameter based on the type of External mode simulation to run on the model.


Hardware board settings

▸ Operating system/scheduler

▾ Target hardware resources

Groups: Build options, Clocking, Input X-BAR, Output X-BAR, ADC_A, ADC_B, ADC_C, ADC_D, CMPSS, DAC, PIL, External mode, SD card logging, Overrun detection

Communication interface: XCP on Serial

SCI module: SCI_A

Serial port in MATLAB preferences: COM3 Refresh

☑ Verbose

☑ Set logging buffer size automatically

Maximum number of contiguous samples: 8

☐ Use a dedicated timer to improve time stamp accuracy

5 When you set the **Communication interface** to an XCP-based External mode, the **Set logging buffer size automatically** parameter becomes available. Select this parameter to automatically set the number of bytes to preallocate for the buffer in the hardware during simulation. By default, the **Set logging buffer size automatically** parameter is selected. If you clear this parameter, **Logging buffer size (in bytes)** parameter becomes available, where you can manually specify the memory buffer size for XCP-based External mode simulation.

The default baud rate is 115200. You can increase the baud of the serial over USB of your Launchpad or controlCARD. On Launchpads and controlCARDs using FTDI 2232H, you can select any baud less than or equal to 6 Mbps, or exactly 9 or 12 Mbps. On controlCARDs using FTDI 2232D, you can select any baud less than or equal to 1.5 Mbps, or exactly 2 or 3 Mbps.

6 You can send multiple contiguous samples in same packet to enhance signal logging performance in models containing signals of high sample rates. To do so, click **Hardware** tab, in the **Prepare gallery**, select **Control Panel** and then click **Signal & Triggering**. In the **External & Signal Triggering** dialog box, select **Send multiple contiguous samples in same packet**. For more information, see “Signal Logging and Parameter Tuning in XCP External Mode with Packed Mode” on page 4-348.

Note You might experience data drops in Packed mode, when data inside interrupt service routine (ISR) is logged.

7 In the **Configuration Parameters** screen, enter a value for **Maximum number of contiguous samples** parameter. This parameter dictates the maximum number of samples that can be filled in a single packet and the memory is allocated accordingly.

8 Click **Apply** and **OK**.

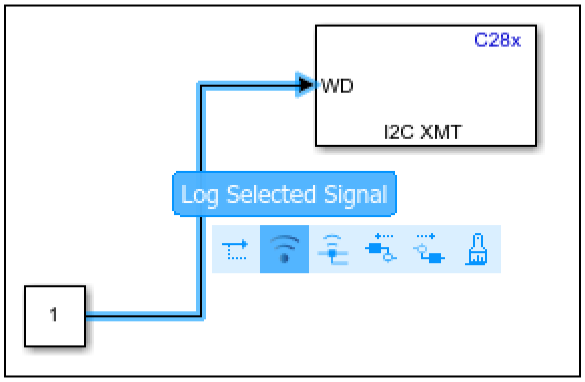
Signal Monitoring and Parameter Tuning of Simulink Model

XCP-Based External Mode Simulation over Serial Connection

Note This section applies only when you set the communication interface to XCP on Serial.

Before you begin, complete the “Prepare a Simulink Model for External Mode” on page 1-59 section.

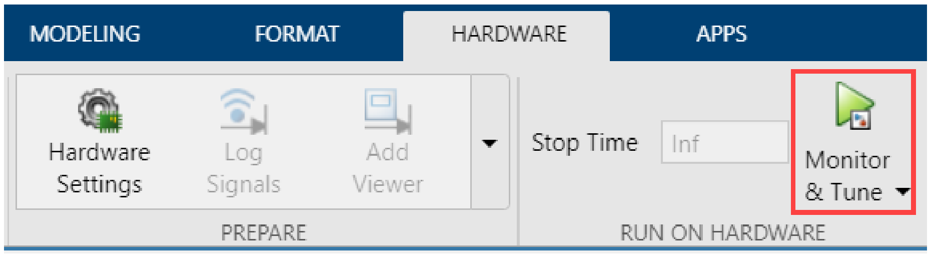
1 In the Simulink model, identify the signals to be logged for monitoring during simulation. Select the identified signal, open its context menu, and click the icon corresponding to **Enable Data Logging**.



Simulink displays a logged signal indicator for each logged signal.

2 (Optional) Place one or more Sink blocks in the model, and then mark the signals connected to them also for logging. For example, connect Display or Scope blocks and mark the signals connected to them for logging.

3 To start the simulation,open the **Hardware** tab and click the **Monitor & Tune**.



If none of the signals in the model is marked for logging, the MATLAB Command Window displays a warning message.

You can disregard this warning or mark signals for logging.

After several minutes, Simulink starts running the model on the hardware.

During simulation, when new simulation data becomes available in SDI, the Simulation Data Inspector button appears highlighted.

4. View the simulation output in Sink blocks or in SDI.

   - Sink blocks – To view the simulation output, double-click the Sink blocks in the model.
   - SDI – To view the new simulation data, perform these steps:

     a. Click the Simulation Data Inspector button.
     b. A new simulation run appears in the **Inspect** pane. The **Inspect** pane lists all logged signals in rows, organized by simulation run. You can expand or collapse any of the runs to view the signals in a run. For more information on signal grouping, see "Signal Grouping".

     We recommend you use SDI rather than using Sink blocks for the following reasons:

     - Streaming data to SDI does not store data in memory, making more efficient use of the memory available on the hardware. Sink blocks such as Scope stores data in buffers before sending the data to the host.
     - Using SDI, you can stream signals from top models and reference models simultaneously. Scope blocks can only log signals from a top-level model.

5. Change the parameter values in the model. Observe the corresponding changes in the simulation output.
6. Find the optimal parameter values by making adjustments and observing the results in the Sink blocks.
7. After you are satisfied with the results, stop the Monitor and Tune action, and save the model.

Note Monitor and Tune action increases the processing burden of the model running on the board. If the software reports an overrun, stop the Monitor and Tune action.

Stop Monitor and Tune

To stop the model that is running in Monitor and Tune, open the **Hardware** tab and click the **Stop** button.

If the Simulation stop time parameter is set to a specific number of seconds, Monitor and Tune stops when that time elapses.

When you finish using Monitor and Tune, set Simulation mode back to `Normal`.

Troubleshooting

Memory overflow issue during compilation

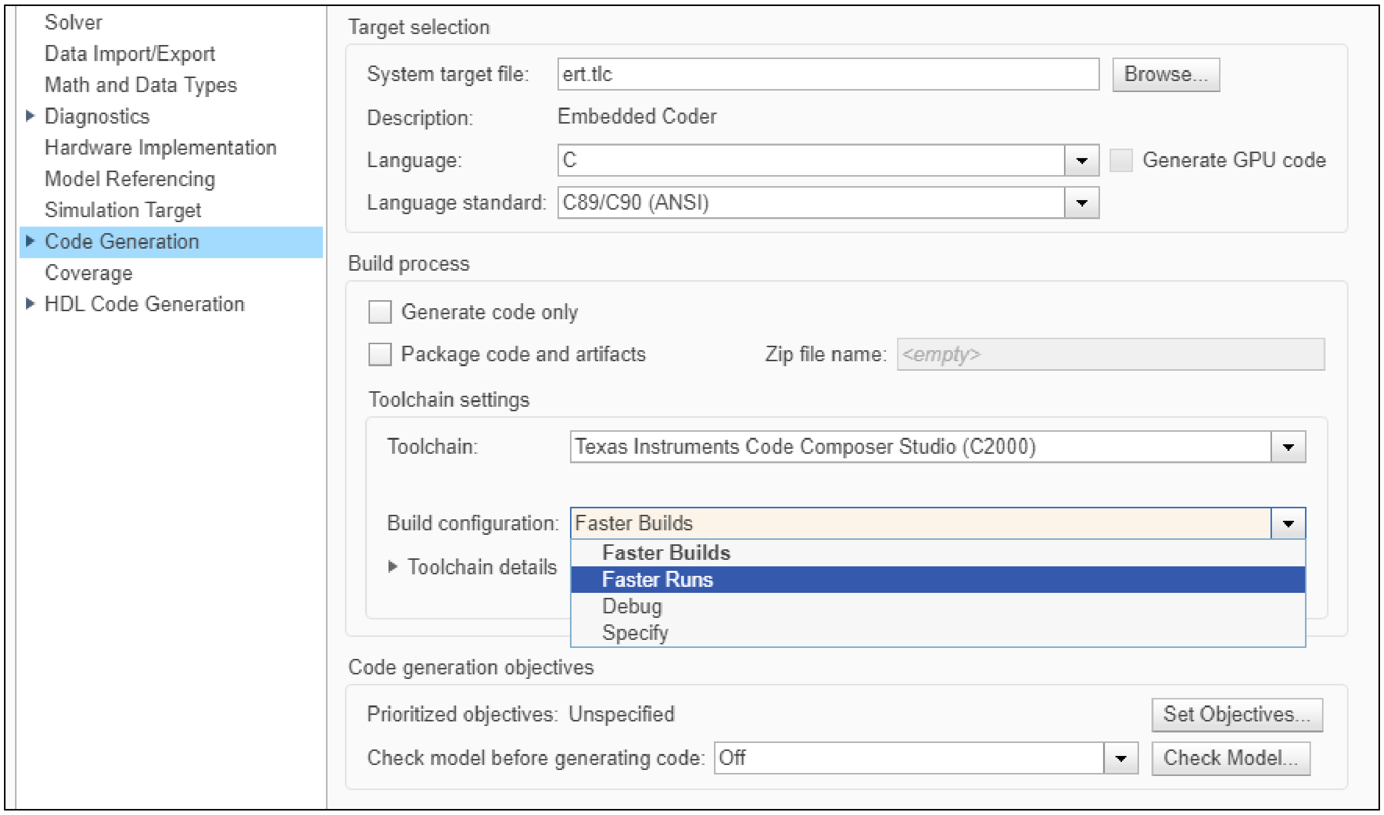
Description

The following error might occur due to insufficient memory:

```
program will not fit into available memory, or the section contains a call site that
requires a trampoline that can't be generated for this section. placement with
alignment/blocking fails for section ".text" size 0x360apage 0.
Available memory ranges: RAMLS_PROG size: 0x3000 unused: 0x2bd7 max hole: 0x2bd6
```

Action

This error shown at linking stage indicates that the memory is insufficient to fit the code section (`.text`) in the target memory available. External Mode simulation requires additional code to enable the communication between Simulink and the target, which might lead to this code section overflow. To mitigate this issue, select `Faster Runs` as the **Build Configuration** in the **Configuration Parameters** screen. This option compiles the code at `-O3` optimization level, resulting in a smaller code size which might fit the target memory.



Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation

Target selection

System target file: ert.tlc
Description: Embedded Coder
Language: C
Generate GPU code
Language standard: C89/C90 (ANSI)

Build process

Generate code only
Package code and artifacts
Zip file name: <empty>

Toolchain settings

Toolchain: Texas Instruments Code Composer Studio (C2000)
Build configuration: Faster Builds
Faster Builds
Faster Runs
Debug
Specify
Toolchain details

Code generation objectives

Prioritized objectives: Unspecified
Check model before generating code: Off

See Also

Related Examples

- “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301
- “Set Up Serial Communication with Target Hardware” on page 1-6

Signal Monitoring and Parameter Tuning over XCP on TCP/IP

In this section...

You can use Monitor and Tune (External Mode) action to tune parameters and monitor a Simulink model running on your TI ARM® Cortex®-M core hardware.

Monitor and Tune enables you to tune model parameters and evaluate the effects of different parameter values on model results in real-time. When you change parameter values in a model, the modified parameter values are communicated to the target hardware immediately. You can monitor the effects of different parameter values by viewing the output signals on Sink blocks or in Simulation Data Inspector (SDI). Doing so helps you find the optimal values for performance. This process is called parameter tuning.

Monitor and Tune accelerates parameter tuning. You do not have to rerun the model each time you change parameters. You can also use Monitor and Tune to develop and validate your model using the actual data and hardware for which it is designed. This software-hardware interaction is not available solely by simulating a model.

The blockset supports Monitor and Tune simulation over `XCP on TCP/IP` communication interface.

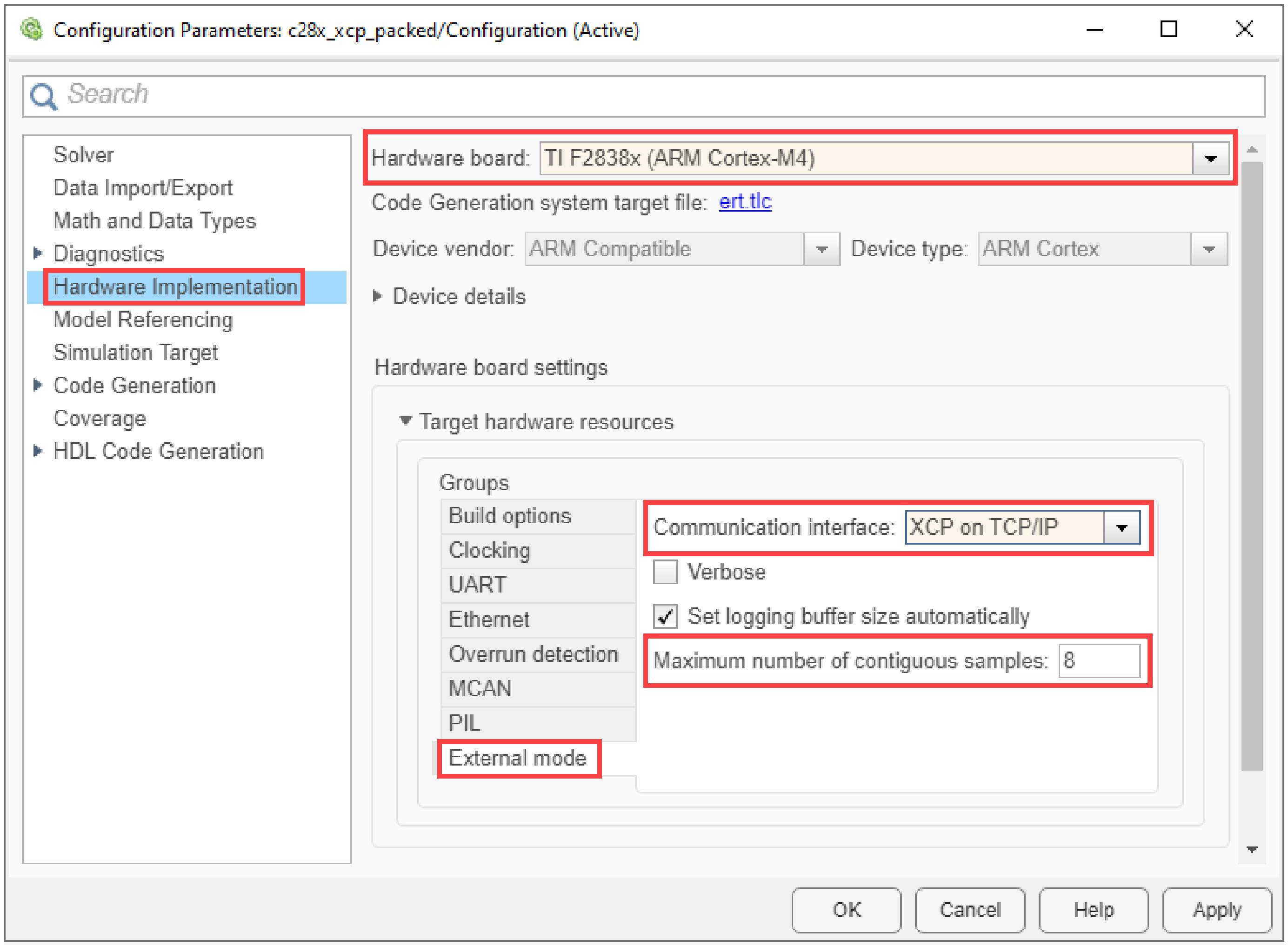
In Universal Measurement and Calibration Protocol (XCP)-based External mode simulation over TCP/IP connections, you can use:

- "Dashboard" blocks: In addition to "Sources" and Sink blocks, you can use "Dashboard" blocks to change parameter values and to monitor the effects of parameter tuning. The Dashboard library contains set of blocks using which you can interactively control and visualize the model.
- Simulation Data Inspector (SDI): You can inspect and compare data from multiple simulations to validate model designs using Simulation Data Inspector.

Prepare a Simulink Model for External Mode

This section explains how to prepare a Simulink model to run in External mode.

1. Configure the hardware network as described in "Hardware Setup for C2000 Microcontroller Blockset" on page 1-3.
2. Create or open a Simulink model.
3. Set the Simulation stop time parameter, located to the left of Simulation mode.
4. Configure the Model Configuration Parameters for the hardware as described in "Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)". In **External mode**, set the **Communication interface** parameter based on the type of External mode simulation to run on the model.


Configuration Parameters: c28x_xcp_packed/Configuration (Active)

Hardware board: TI F2838x (ARM Cortex-M4)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible   Device type: ARM Cortex

Communication interface: XCP on TCP/IP

Verbose: ☐

Set logging buffer size automatically: ✓

Maximum number of contiguous samples: 8

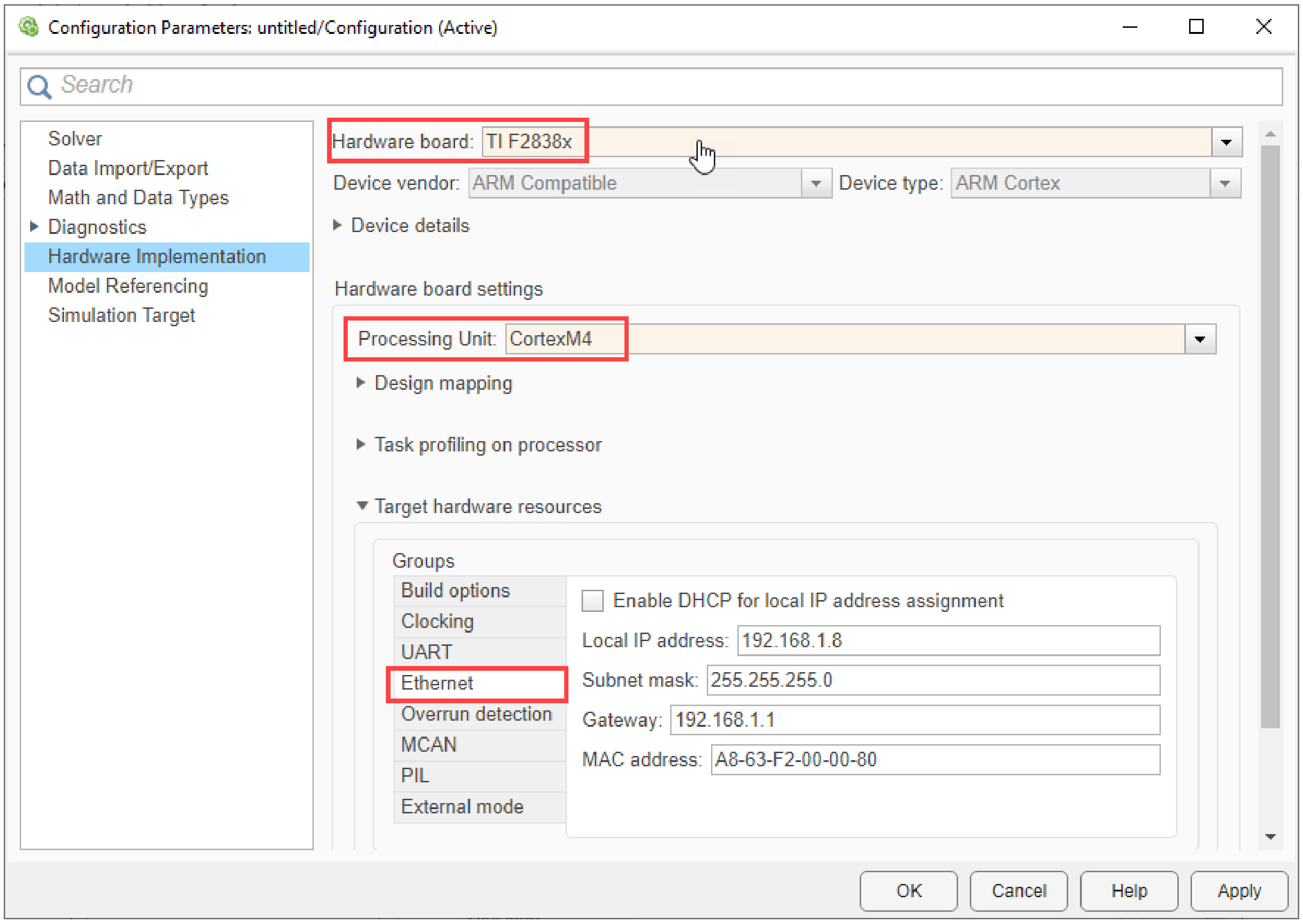
5 When you set the **Communication interface** to an XCP-based External mode, the **Set logging buffer size automatically** parameter becomes available. Select this parameter to automatically set the number of bytes to preallocate for the buffer in the hardware during simulation. By default, the **Set logging buffer size automatically** parameter is selected. If you clear this parameter, **Logging buffer size (in bytes)** parameter becomes available, where you can manually specify the memory buffer size for XCP-based External mode simulation.

6 You can send multiple contiguous samples in same packet to enhance signal logging performance in models containing signals of high sample rates. To do so, click **Hardware** tab, in the **Prepare gallery**, select **Control Panel** and then click **Signal & Triggering**. In the **External & Signal Triggering** dialog box, select **Send multiple contiguous samples in same packet**. For more information, see “Signal Logging and Parameter Tuning in XCP External Mode with Packed Mode” on page 4-348.

Note You might experience data drops in Packed mode, when data inside interrupt service routine (ISR) is logged.

7 In the **Configuration Parameters** screen, enter a value for **Maximum number of contiguous samples** parameter. This parameter dictates the maximum number of samples that can be filled in a single packet and the memory is allocated accordingly.

8 Go to **Ethernet** and specify the host addresses.



Hardware board: TI F2838x
Device vendor: ARM Compatible
Device type: ARM Cortex
Processing Unit: CortexM4
☐ Enable DHCP for local IP address assignment
Local IP address: 192.168.1.8
Subnet mask: 255.255.255.0
Gateway: 192.168.1.1
MAC address: A8-63-F2-00-00-80

- **Enable DHCP for local IP address assignment** - Select this parameter to configure the board to get an IP address from the local DHCP server on the network.
- **Local IP Address** - Select this parameter to set the IP address of the board.
- **Subnet mask** - Specify the subnet mask for the board. The subnet mask is a mask that designates a logical subdivision of a network.
- **Gateway** - Set the serial gateway to the gateway required to access the target computer.
- **MAC Address** - Specify the media access control (MAC) address, the physical network address of the board.

For more information on Ethernet parameter configurations, refer Ethernet.

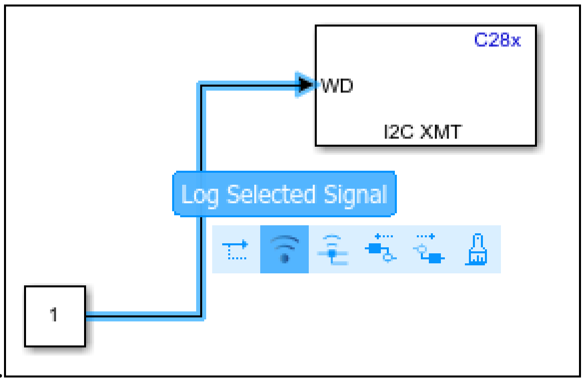
Signal Monitoring and Parameter Tuning of Simulink Model

XCP-Based External Mode Simulation over TCP/IP Connection

Note This section applies only when you set the communication interface to XCP on TCP/IP.

Before you begin, complete the “Prepare a Simulink Model for External Mode” on page 1-65 section.

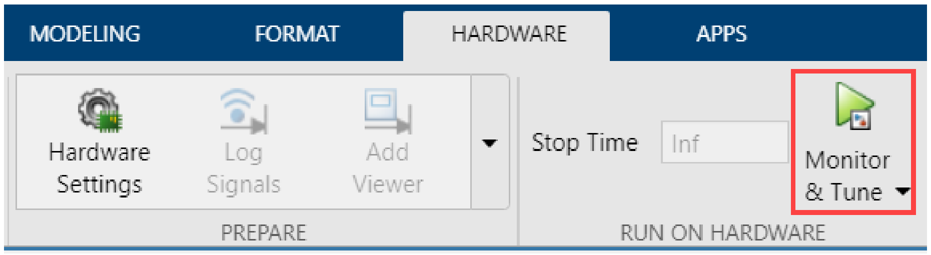
1 In the Simulink model, identify the signals to be logged for monitoring during simulation. Select the identified signal, open its context menu, and click the icon corresponding to **Enable Data Logging**.



Simulink displays a logged signal indicator for each logged signal.

2 (Optional) Place one or more Sink blocks in the model, and then mark the signals connected to them also for logging. For example, connect Display or Scope blocks and mark the signals connected to them for logging.

3 To start the simulation,open the **Hardware** tab and click the **Monitor & Tune**.



If none of the signals in the model is marked for logging, the MATLAB Command Window displays a warning message.

You can disregard this warning or mark signals for logging.

After several minutes, Simulink starts running the model on the hardware.

During simulation, when new simulation data becomes available in SDI, the Simulation Data Inspector button appears highlighted.

4 View the simulation output in Sink blocks or in SDI.

- Sink blocks – To view the simulation output, double-click the Sink blocks in the model.
- SDI – To view the new simulation data, perform these steps:

  a Click the Simulation Data Inspector button.

  b A new simulation run appears in the **Inspect** pane. The **Inspect** pane lists all logged signals in rows, organized by simulation run. You can expand or collapse any of the runs to view the signals in a run. For more information on signal grouping, see “Signal Grouping”.

We recommend you use SDI rather than using Sink blocks for the following reasons:

- Streaming data to SDI does not store data in memory, making more efficient use of the memory available on the hardware. Sink blocks such as Scope stores data in buffers before sending the data to the host.
- Using SDI, you can stream signals from top models and reference models simultaneously. Scope blocks can only log signals from a top-level model.

5. Change the parameter values in the model. Observe the corresponding changes in the simulation output.
6. Find the optimal parameter values by making adjustments and observing the results in the Sink blocks.
7. After you are satisfied with the results, stop the Monitor and Tune action, and save the model.

Note Monitor and Tune action increases the processing burden of the model running on the board. If the software reports an overrun, stop the Monitor and Tune action.

Stop Monitor and Tune

To stop the model that is running in Monitor and Tune, open the **Hardware** tab and click the **Stop** button .

If the Simulation stop time parameter is set to a specific number of seconds, Monitor and Tune stops when that time elapses.

When you finish using Monitor and Tune, set Simulation mode back to `Normal`.

Troubleshooting

Memory overflow issue during compilation

Description

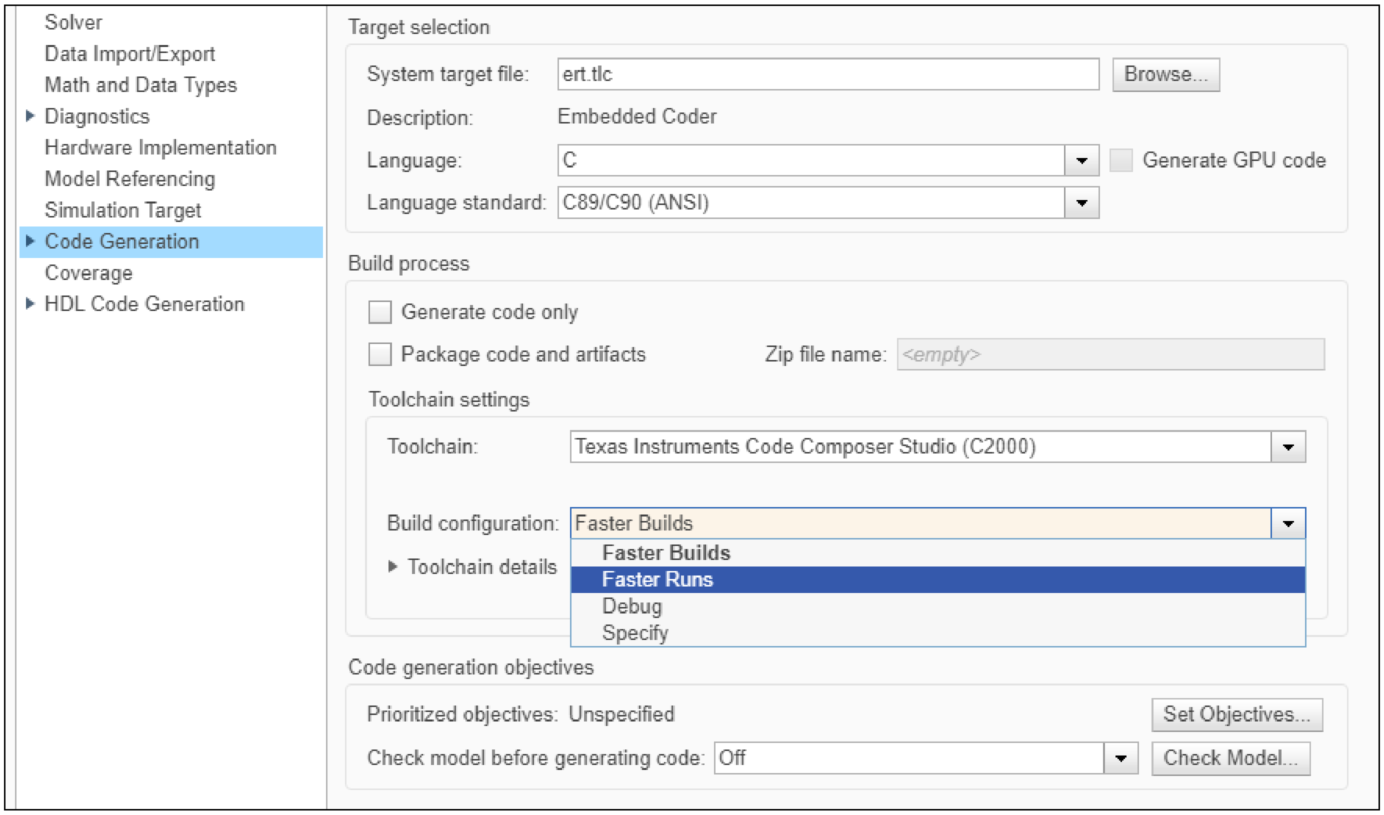
The following error might occur due to insufficient memory:

```
program will not fit into available memory, or the section contains a call site that
requires a trampoline that can't be generated for this section. placement with
alignment/blocking fails for section ".text" size 0x360apage 0.
Available memory ranges: RAMLS_PROG size: 0x3000 unused: 0x2bd7 max hole: 0x2bd6
```

Action

This error shown at linking stage indicates that the memory is insufficient to fit the code section (`.text`) in the target memory available. External Mode simulation requires additional code to enable the communication between Simulink and the target, which might lead to this code section overflow. To mitigate this issue, select `Faster Runs` as the **Build Configuration** in the **Configuration**

Parameters screen. This option compiles the code at `-O3` optimization level, resulting in a smaller code size which might fit the target memory.


Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation

Target selection

System target file: ert.tlc
Description: Embedded Coder
Language: C
Generate GPU code
Language standard: C89/C90 (ANSI)

Build process

Generate code only
Package code and artifacts
Zip file name: <empty>

Toolchain settings

Toolchain: Texas Instruments Code Composer Studio (C2000)
Build configuration: Faster Builds
Toolchain details

Faster Builds
Faster Runs
Debug
Specify

Code generation objectives

Prioritized objectives: Unspecified
Check model before generating code: Off

See Also

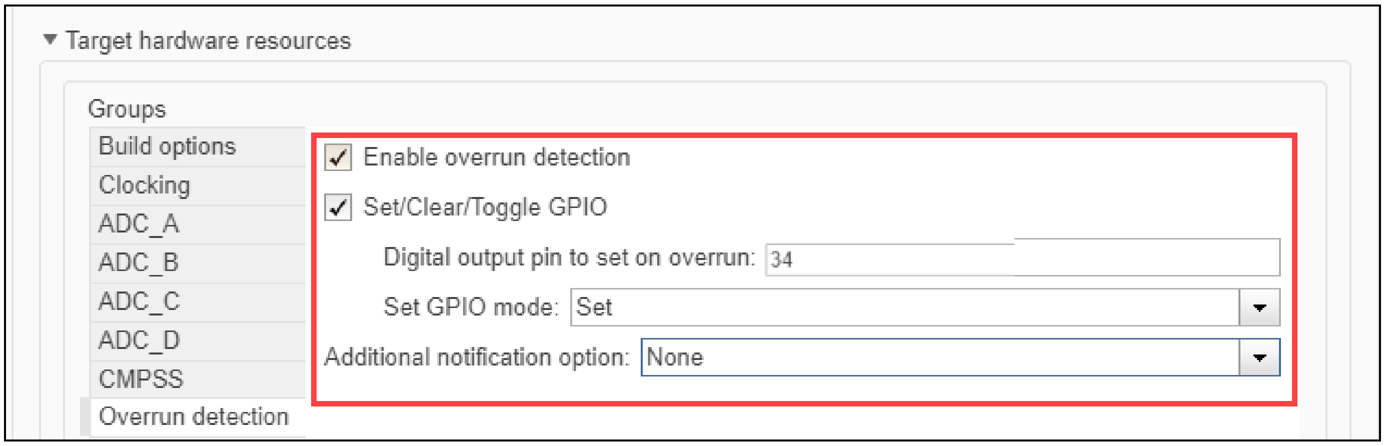
“ARM Cortex-M4 - Ethernet” | “External Mode” | “PIL”

Detect and Fix Task Overruns on Texas Instruments C2000 Hardware

You can configure a Simulink model running on the target hardware to detect and notify you when a task overrun occurs. A task overrun occurs if the target hardware is still performing one instance of a task when the next instance of that task is scheduled to begin. You can fix overruns by decreasing the frequency with which tasks are scheduled to run, and/or by reducing the number of tasks defined by your model.

To enable overrun detection:

1. Click the **Tools** menu in the model, and select **Run on Target Hardware** > **Options**.
2. In the Hardware Implementation pane that opens, select the **Overrun detection > Enable overrun detection** check box.
3. Use the **Digital output pin to set on overrun** parameter to specify the GPIO pin number of a digital output.



✓ Enable overrun detection

✓ Set/Clear/Toggle GPIO

Digital output pin to set on overrun: 34

Set GPIO mode: Set

Additional notification option: None

   - Select the GPIO mode: `Set, Clear or Toggle`.
   - If required select the additional notification option to notify when task overrun occurs: `Trigger interrupt or Call user-defined function`.
4. Click **OK**.

When a task overrun occurs:

- The state of the digital output pin specified by the **Digital output to set on overrun** parameter changes from low (0 Volts) to high (3.3 Volts).
- If Additional notification is selected as **Trigger Interrupt** then ISR with mentioned **PIE** and **CPU** number will be triggered.

  Note Set the priority value of the interrupt to less than 40 (default base rate priority) to ensure interrupt will trigger on overrun.

  If Additional notification is selected as **Call user-defined function** then C function will be called on overrun.

- The model continues running, but the effective sample time will be longer than specified.

To fix an overrun condition, reduce the processing burden of the model by applying one or more of the following solutions:

- Profiling can be used to measure execution time of each task and analyze in detail about the task which is overrunning.
- Increase the sample times for the model. For example, increase the values of the **Sample time** parameters in all of your data source blocks.
- Simplify the model.

If you are using External mode, and the preceding solutions do not fix the task overrun condition, consider disabling External mode. External mode adds a lightweight server to the model running on the target hardware. This server increases the processing burden upon the target hardware, which can contribute to a task overrun condition. There will be additional overhead due to profiling logic as well. This may also contribute to overrun condition.

See Also

Overrun detection | "Configuring Additional Options for Code Generation"

Related Examples

- "Real-Time Code Execution Profiling" on page 4-157

Quadrature Encoder Offset Calibration for PMSM Motor

This example calculates the offset between the d-axis of the rotor and encoder index pulse position as detected by the quadrature encoder sensor. The control algorithm (available in the field-oriented control and parameter estimation examples) uses this offset value to compute an accurate and precise position of the d-axis of rotor. The controller needs this position to implement the field-oriented control (FOC) correctly in the rotor flux reference frame (d-q reference frame), and therefore, run the permanent magnet synchronous motor (PMSM) correctly.

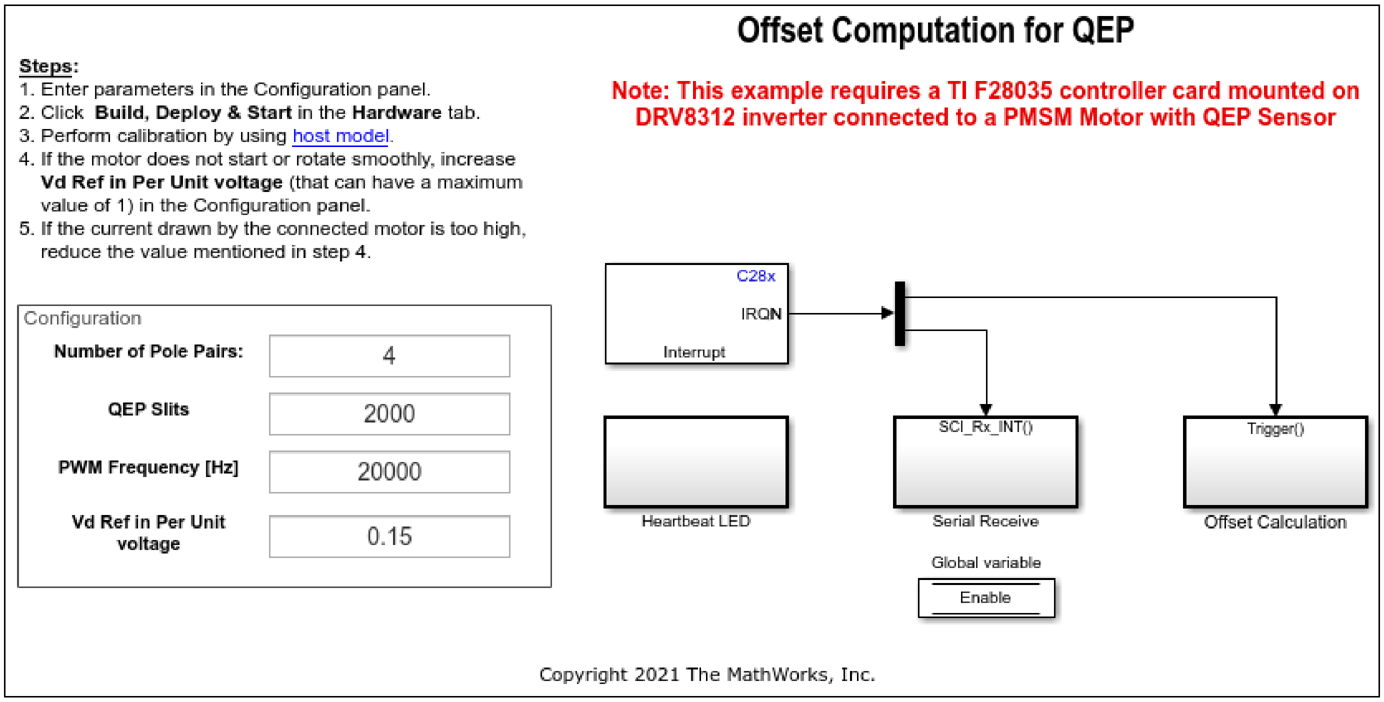
Models

The example includes this model:

- **mcb_pmsm_qep_offset_f28035**
- **mcb_pmsm_qep_offset_f280049C**
- For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Quadrature Encoder Offset Calibration for PMSM Motor” (Motor Control Blockset).

You can use these models only for code generation. You can use the open_system command to open the Simulink model. For example, use this command for a F28035 based controller:

```
open_system('mcb_pmsm_qep_offset_f28035.slx');
```



Offset Computation for QEP

Steps:
1. Enter parameters in the Configuration panel.
2. Click **Build, Deploy & Start** in the **Hardware** tab.
3. Perform calibration by using host model.
4. If the motor does not start or rotate smoothly, increase **Vd Ref in Per Unit voltage** (that can have a maximum value of 1) in the Configuration panel.
5. If the current drawn by the connected motor is too high, reduce the value mentioned in step 4.

Note: This example requires a TI F28035 controller card mounted on DRV8312 inverter connected to a PMSM Motor with QEP Sensor

Configuration

Number of Pole Pairs:	4
QEP Slits	2000
PWM Frequency [Hz]	20000
Vd Ref in Per Unit voltage	0.15

Copyright 2021 The MathWorks, Inc.

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks Products

To generate code and deploy model:

For the models: **mcb_pmsm_qep_offset_f28035/ mcb_pmsm_qep_offset_f280049C**

- Motor Control Blockset™
- Embedded Coder
- C2000 Microcontroller Blockset
- Fixed-Point Designer

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the motor by using open-loop control.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board.

The host model uses serial communication to command the target model and run the motor in an open-loop configuration. You can use the host model to control the motor rotations and validate the direction of rotation of motor. The **Incorrect motor direction** LED in the host model turns red to indicate that the motor is running in the opposite direction. When the LED turns red, you must reverse the motor phase connections (from ABC to CBA) to change the direction of rotation. The host model displays the calculated offset value.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB command prompt.

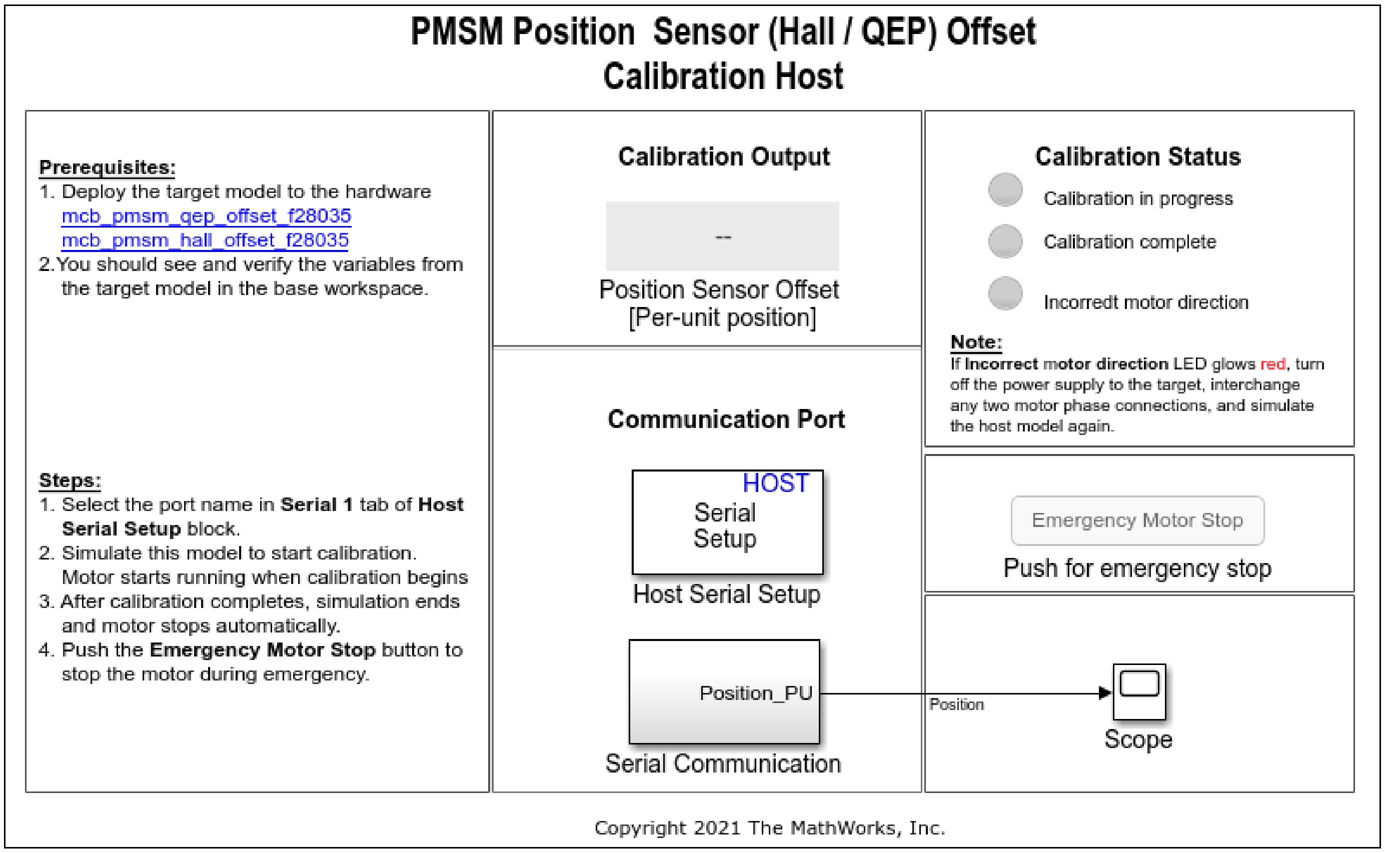
- F28035 control card + DRV8312-C2-KIT inverter: **mcb_pmsm_qep_offset_f28035**

  For connections related to the preceding hardware configuration, see “F28069/F28035/F28335 control card configuration” on page 1-81.
- F280049C control card + DRV8312-C2-KIT inverter: **mcb_pmsm_qep_offset_f280049C**

1. Complete the hardware connections.
2. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see “Model Configuration Parameters” (Motor Control Blockset).
3. Update these motor parameters in the **Configuration** panel of the target model.

   - Number of Pole Pairs
   - QEP Slits
   - PWM Frequency [Hz]
   - Vd Ref in Per Unit voltage
4. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.
5. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28035 based controller:

   ```
   open_system('mcb_pmsm_host_offsetComputation.slx');
   ```


PMSM Position Sensor (Hall / QEP) Offset Calibration Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_qep_offset_f28035
   mcb_pmsm_hall_offset_f28035
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select the port name in **Serial 1** tab of **Host Serial Setup** block.
2. Simulate this model to start calibration. Motor starts running when calibration begins
3. After calibration completes, simulation ends and motor stops automatically.
4. Push the **Emergency Motor Stop** button to stop the motor during emergency.

Note:
If **Incorrect motor direction** LED glows red, turn off the power supply to the target, interchange any two motor phase connections, and simulate the host model again.

Copyright 2021 The MathWorks, Inc.

For details about the serial communication between the host and target models, see “Host-Target Communication” (Motor Control Blockset).

You can use the Time Scope in the host model to monitor the rotor position and offset values.

6 Set the parameter **Port** of the following blocks in the `mcb_pmsm_host_offsetComputation` model to match the COM port of the host computer:

- `mcb_pmsm_host_offsetComputation` > Host Serial Setup
- `mcb_pmsm_host_offsetComputation` > Serial Communication > Host Serial Receive
- `mcb_pmsm_host_offsetComputation` > Serial Communication > SCI_TX > Host Serial Transmit

7 Click **Run** on the **Simulation** tab to run the host model. The motor runs and calibration begins when you start simulation. After calibration completes, simulation ends and the motor stops automatically.

8 See the **Calibration Status** section to know the status of calibration process:

- **Calibration in progress** - LED turns orange when the motor starts running. Notice the rotor position and offset value variations in the Time Scope (the position signal indicates a ramp signal with an amplitude between 0 and 1). After calibration completes this LED turns grey.
- **Calibration complete** - LED turns green when calibration completes. Then the **Calibration Output** field displays the computed offset value.
- **Incorrect motor direction** - LED turns red if the motor runs in the opposite direction. Then the Calibration Output field displays the value "NaN." Turn off the DC power supply (24V) and

reverse the motor phase connections from ABC to CBA. Repeat steps 5 to 8 and check if the Calibration complete LED is green. Verify that the Calibration Output field displays the offset value.

Note This example does not support simulation.

During emergency, click the **Emergency Motor Stop** button to stop the motor immediately.

For examples that implement FOC using a quadrature encoder sensor, you must update the computed quadrature encoder offset value in the pmsm.PositionOffset parameter in the model initialization script linked to the example. For instructions, see "Estimate Control Gains and Use Utility Functions" (Motor Control Blockset).

See Also

Related Examples

- "Field-Oriented Control of PMSM with Quadrature Encoder Using C2000 Processors" on page 4-16

Hall Offset Calibration for PMSM Motor

This example calculates the offset between the rotor direct axis (d-axis) and position detected by the Hall sensor. The field-oriented control (FOC) algorithm needs this position offset to run the permanent magnet synchronous motor (PMSM) correctly. To compute the offset, the target model runs the motor in the open-loop condition. The model uses a constant V_d (voltage along the stator's d-axis) and a zero V_q (voltage along the stator's q-axis) to run the motor (at a low constant speed) by using a position or ramp generator. When the position or ramp value reaches zero, the corresponding rotor position is the offset value for the Hall sensors.

The control algorithm (available in the field-oriented control and parameter estimation examples) uses this offset value to compute an accurate position of d-axis of the rotor. The controller needs this offset to optimally run the PMSM.

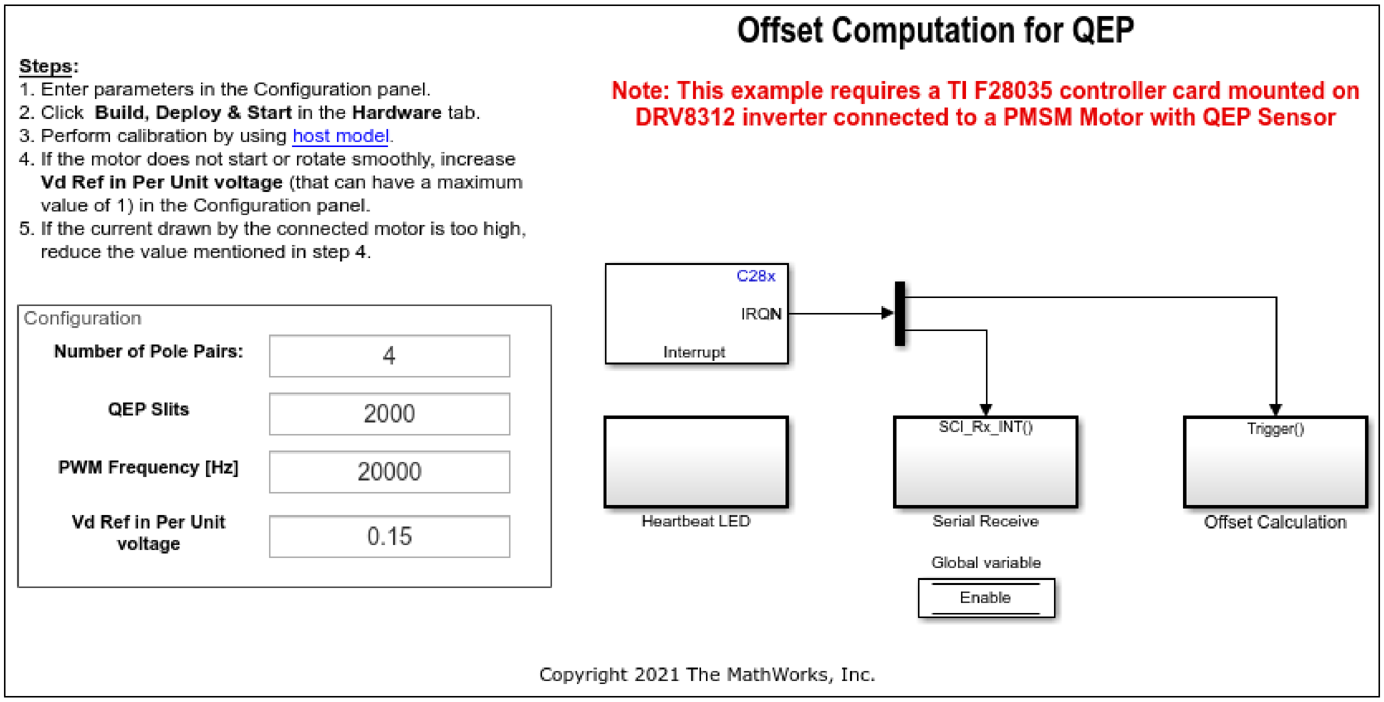
Models

This example includes this model:

- **mcb_pmsm_hall_offset_f28035**
- **mcb_pmsm_hall_offset_f280049C**
- For LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Field-Oriented Control of PMSM Using Hall Sensor” (Motor Control Blockset).

You can use these models only for code generation. You can use the open_system command to open the Simulink model. For example, use this command for a F28035 based controller:

```
open_system('mcb_pmsm_hall_offset_f28035.slx');
```



Offset Computation for QEP

Steps:
1. Enter parameters in the Configuration panel.
2. Click **Build, Deploy & Start** in the **Hardware** tab.
3. Perform calibration by using host model.
4. If the motor does not start or rotate smoothly, increase **Vd Ref in Per Unit voltage** (that can have a maximum value of 1) in the Configuration panel.
5. If the current drawn by the connected motor is too high, reduce the value mentioned in step 4.

Note: This example requires a TI F28035 controller card mounted on DRV8312 inverter connected to a PMSM Motor with QEP Sensor

Configuration

Number of Pole Pairs:	4
QEP Slits	2000
PWM Frequency [Hz]	20000
Vd Ref in Per Unit voltage	0.15

Copyright 2021 The MathWorks, Inc.

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks Products

To generate code and deploy model:

For the models: **mcb_pmsm_hall_offset_f28035**/ **mcb_pmsm_hall_offset_f280049C**

- Motor Control Blockset
- Embedded Coder
- C2000 Microcontroller Blockset
- Fixed-Point Designer

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the motor by using open-loop control.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board.

The host model uses serial communication to command the target model and run the motor in an open-loop configuration. You can use the host model to control the motor rotations and validate the direction of rotation of motor. The **Incorrect motor direction** LED in the host model turns red to indicate that the motor is running in the opposite direction. When the LED turns red, you must reverse the motor phase connections (from ABC to CBA) to change the direction of rotation. The host model displays the calculated offset value.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB command prompt.

- F28035 control card + DRV8312-C2-KIT inverter: **mcb_pmsm_hall_offset_f28035**

  For connections related to the preceding hardware configuration, see “F28069/F28035/F28335 control card configuration” on page 1-81.
- F280049 control card + DRV8312-C2-KIT inverter: **mcb_pmsm_hall_offset_f280049C**
- Three-phase PMSM with optional HALL sensors attached to connector J10 of the DRV8312EVM board

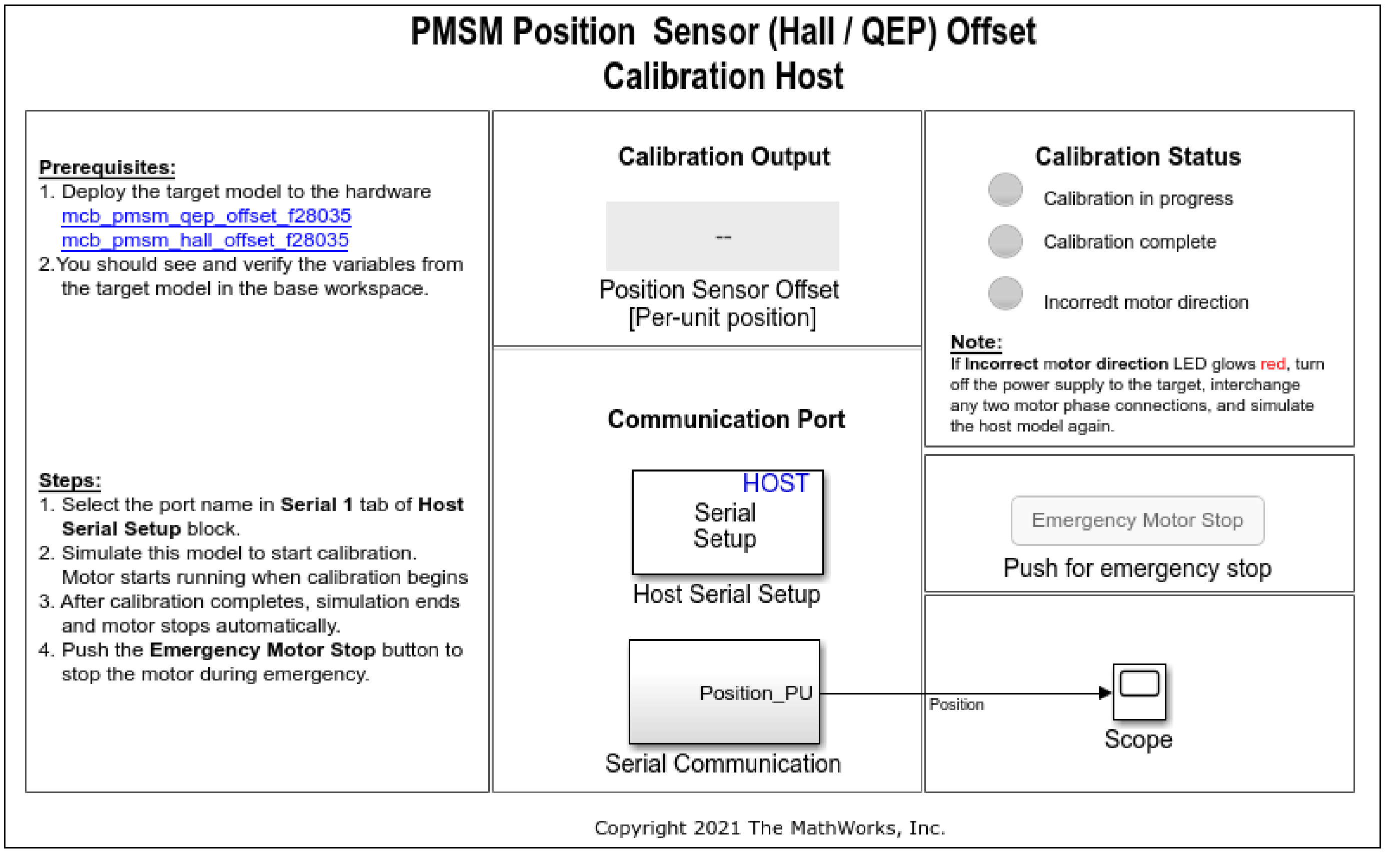
1. Complete the hardware connections.
2. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see “Model Configuration Parameters” (Motor Control Blockset).
3. Update these motor parameters in the **Configuration** panel of the target model.

   - Number of Pole Pairs

- PWM Frequency [Hz]
- Vd Ref in Per Unit voltage

4. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.
5. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28035 based controller:

```
open_system('mcb_pmsm_host_offsetComputation.slx');
```



PMSM Position Sensor (Hall / QEP) Offset Calibration Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_qep_offset_f28035
   mcb_pmsm_hall_offset_f28035
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select the port name in **Serial 1** tab of **Host Serial Setup** block.
2. Simulate this model to start calibration. Motor starts running when calibration begins
3. After calibration completes, simulation ends and motor stops automatically.
4. Push the **Emergency Motor Stop** button to stop the motor during emergency.

Note:
If **Incorrect motor direction** LED glows red, turn off the power supply to the target, interchange any two motor phase connections, and simulate the host model again.

Copyright 2021 The MathWorks, Inc.

For details about the serial communication between the host and target models, see “Host-Target Communication” (Motor Control Blockset).

You can use the Time Scope in the host model to monitor the rotor position and offset values.

6. Set the parameter **Port** of the following blocks in the `mcb_pmsm_host_offsetComputation` model to match the COM port of the host computer:
   - `mcb_pmsm_host_offsetComputation` > Host Serial Setup
   - `mcb_pmsm_host_offsetComputation` > Serial Communication > Host Serial Receive
   - `mcb_pmsm_host_offsetComputation` > Serial Communication > SCI_TX > Host Serial Transmit
7. Click **Run** on the **Simulation** tab to run the host model. The motor runs and calibration begins when you start simulation. After calibration completes, simulation ends and the motor stops automatically.

8 See the **Calibration Status** section to know the status of calibration process:

- **Calibration in progress** - LED turns orange when the motor starts running. Notice the rotor position and offset value variations in the Time Scope (the position signal indicates a ramp signal with an amplitude between 0 and 1). After calibration completes this LED turns grey.
- **Calibration complete** - LED turns green when calibration completes. Then the **Calibration Output** field displays the computed offset value.
- **Incorrect motor direction** - LED turns red if the motor runs in the opposite direction. Then the Calibration Output field displays the value "NaN." Turn off the DC power supply (24V) and reverse the motor phase connections from ABC to CBA. Repeat steps 5 to 8 and check if the Calibration complete LED is green. Verify that the Calibration Output field displays the offset value.

Note This example does not support simulation.

During emergency, click the **Emergency Motor Stop** button to stop the motor immediately.

For examples that implement FOC using a Hall sensor, you must update the computed offset value in the pmsm.PositionOffset parameter in the model initialization script linked to the example. For instructions, see “Estimate Control Gains and Use Utility Functions” (Motor Control Blockset).

See Also

Related Examples

- “Field-Oriented Control of PMSM with Hall Sensor Using C2000 Processors” on page 4-162

Hardware Connections

C2000 Microcontroller Blockset supports the following hardware configurations:

1. LAUNCHXL-F28069M configuration
2. LAUNCHXL-F28379D configuration
3. LAUNCHXL-F280049C configuration
4. F28069/F28035/F28335 control card configuration
5. LAUNCHXL-F2827/F28027D configuration
6. F28M35x Concerto® and F28M36x Concerto control card configuration
7. C2000 MCU Resolver Eval Kit [R2]

F28069/F28035/F28335 control card configuration

The configuration includes the following hardware components:

- Texas Instruments DRV8312-69M-KIT/DRV8312-C2-KIT inverter board
- Texas Instruments F28069/F28035/F28335 microcontroller control card
- Texas Instruments F28M35x Concerto/F28M36x Concerto microcontroller control card
- Motor BLY171D (supports both Hall and quadrature encoder sensors)
- Motor BLY172S (supports Hall sensor)
- Quadrature encoder
- DC power supply

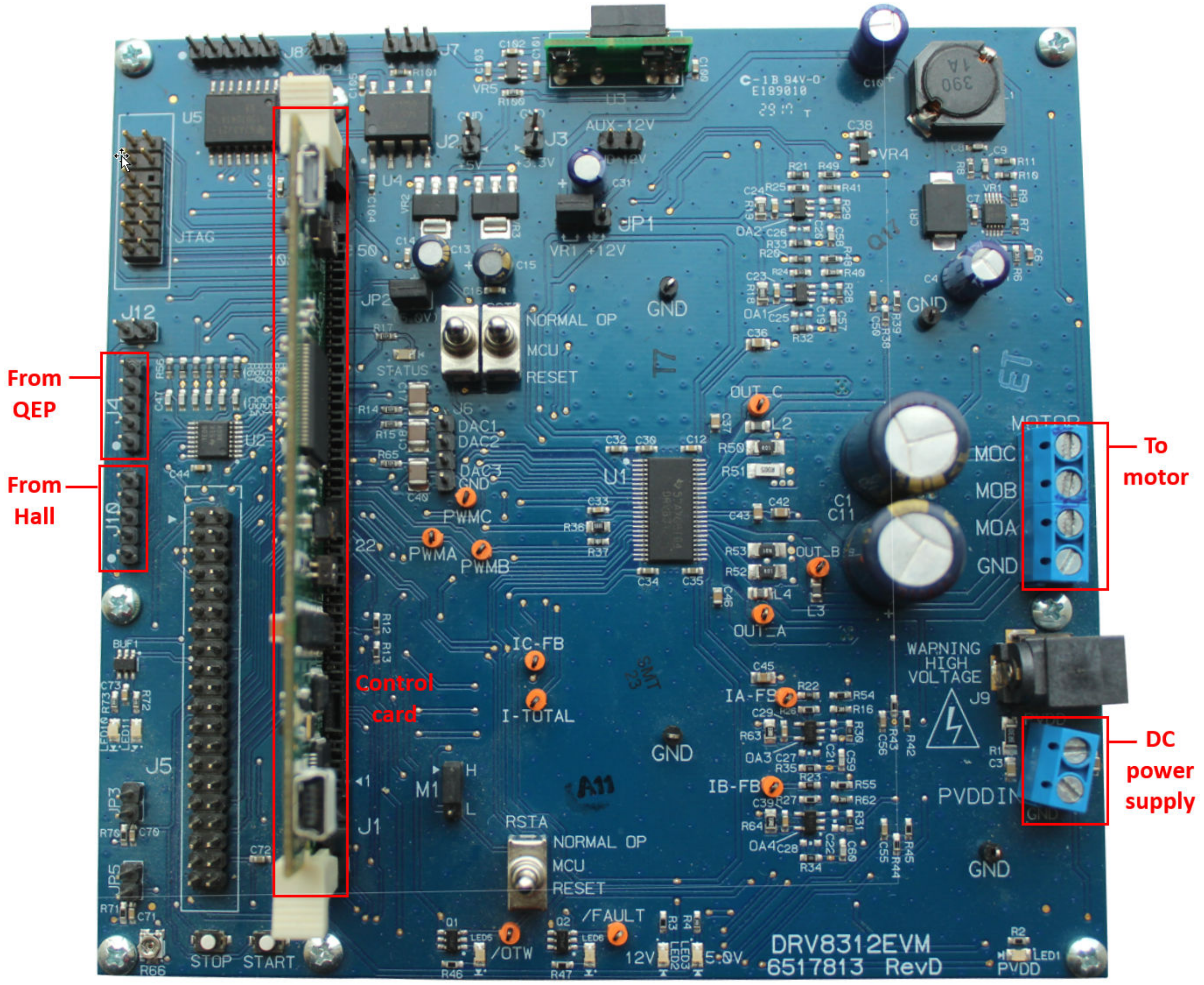
Note Due to auxiliary power supply related hardware issues, the DRV8312-69M-KIT/DRV8312-C2-KIT does not support the position sensors connected to some motors (for example, Teknic M-2310P motor).

The following steps describe the hardware connections for the F28069 control card configuration. You can use the same hardware connections for other control cards mentioned above.

Note For F28M36x Concerto use TMSADAP180TO100 card to connect to DRV8312-C2-KIT.

1. Connect the F28069 (or any of the above mentioned) control card to J1 of DRV8312-69M-KIT inverter board.
2. Connect the motor three phases, to MOA, MOB, and MOC on the inverter board.
3. Connect the DC power supply (24V) to PVDDIN on the inverter board.

Warning Be careful when connecting PVDD and GND to the positive and negative connections of the DC power supply. A reverse connection can damage the hardware components.

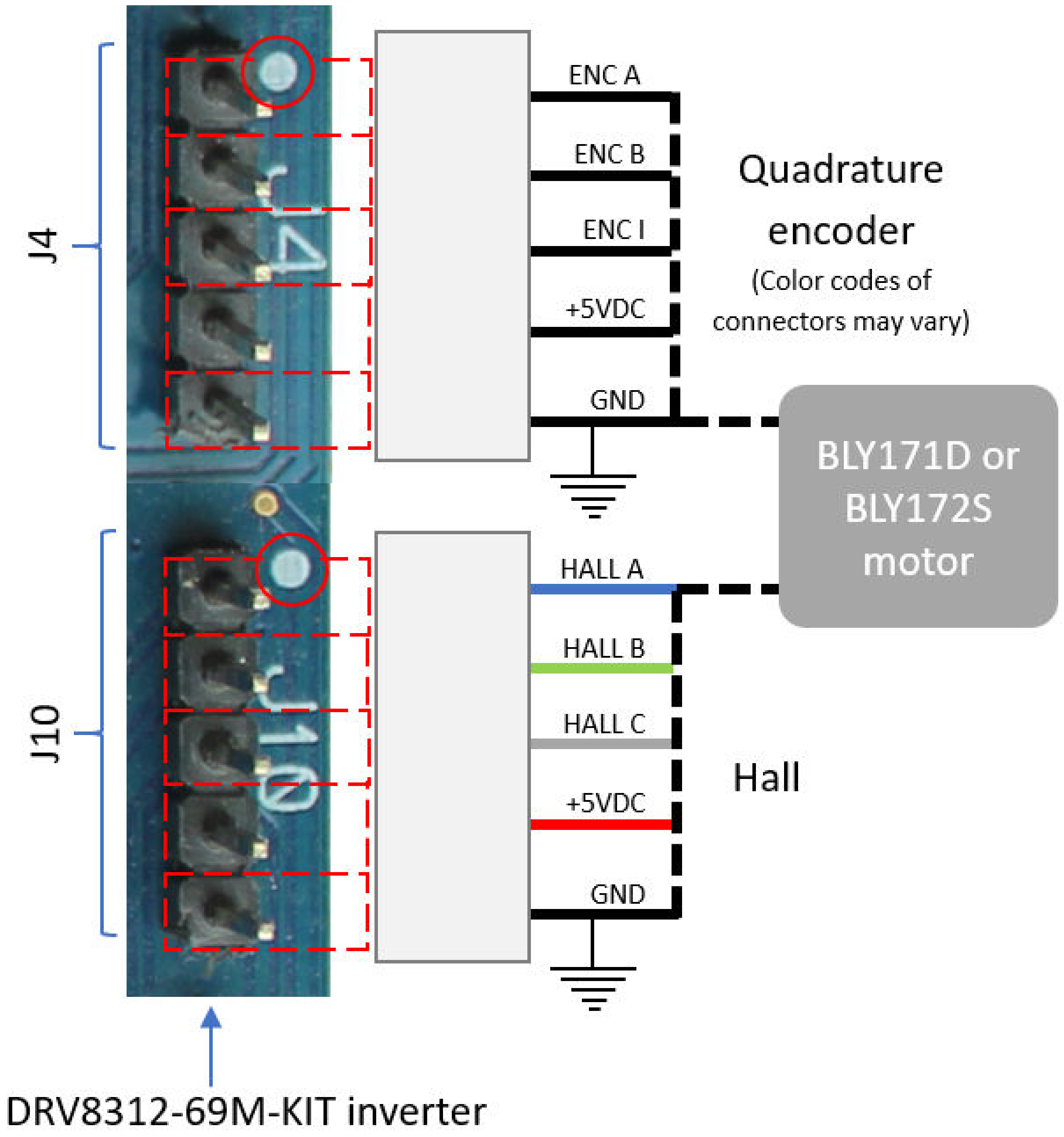


The following step describes about interfacing the quadrature encoder sensor:

- Connect the quadrature encoder pins (G, I, A, 5V, B) to J4 on the inverter board.

To implement position-sensing by using Hall sensor, use a motor that has inbuilt Hall sensors (for example, BLY171D and BLY172S). The following steps describe the steps to interface the Hall sensor:

- Connect the Hall sensor encoder output to J10 on the inverter board.



We recommend the following jumper settings for DRV8312-69M-KIT/DRV8312-C2-KIT inverter board when working with C2000 Microcontroller Blockset. You can customize these settings depending on the application requirements. For more information about these settings, see the device user guide available on Texas Instruments website.

- JP1 – VR1
- JP2 – ON
- JP3 – OFF
- JP4 – OFF

- JP5 – OFF
- M1 – H
- J2 – OFF
- J3 – OFF
- RSTA – MCU
- RSTB - MCU
- RSTC - MCU

LAUNCHXL-F28069M/F28379D/F28027/F28027D/F280049C Configurations

The LAUNCHXL-F28069M configuration includes the following hardware components:

- LAUNCHXL-F28069M controller
- BOOSTXL-DRV8305 (supported inverter)
- Teknic motor M-2310P (supports both Hall and quadrature encoder sensors)
- Motor BLY171D (supports both Hall and quadrature encoder sensors)
- Motor BLY172S (supports Hall sensor)
- DC power supply

The LAUNCHXL-F28379D configuration includes the following hardware components:

- LAUNCHXL-F28379D controller
- BOOSTXL-DRV8305 and BOOSTXL-3PHGANINV (supported inverters)
- Teknic motor M-2310P (supports both Hall and quadrature encoder sensors)
- Motor BLY171D (supports both Hall and quadrature encoder sensors)
- Motor BLY172S (supports Hall sensor)
- DC power supply

The LAUNCHXL-F28027/F28027D/F280049C configuration includes the following hardware components:

- LAUNCHXL-F28027 controller
- LAUNCHXL-F280049 controller
- BOOSTXL-DRV8305
- Teknic motor M-2310P
- Motor BLY171D
- Motor BLY172S
- DC power supply

Note LAUNCHXL-F28027/F28027D controller is sensorless and does not support both Hall and quadrature encoder sensors.

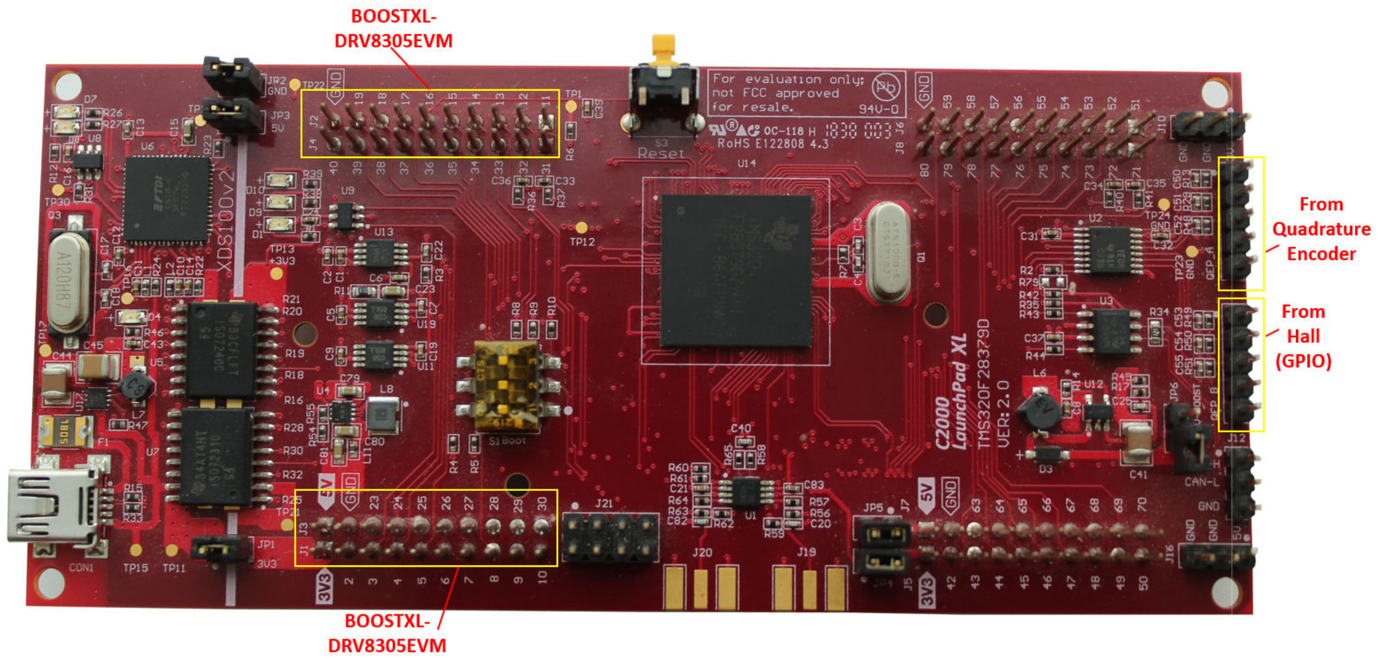
The following steps describe the hardware connections for the LAUNCHXL-F28069M, LAUNCHXL-F28379D, LAUNCHXL-F28027D, and LAUNCHXL-F280049C configurations:

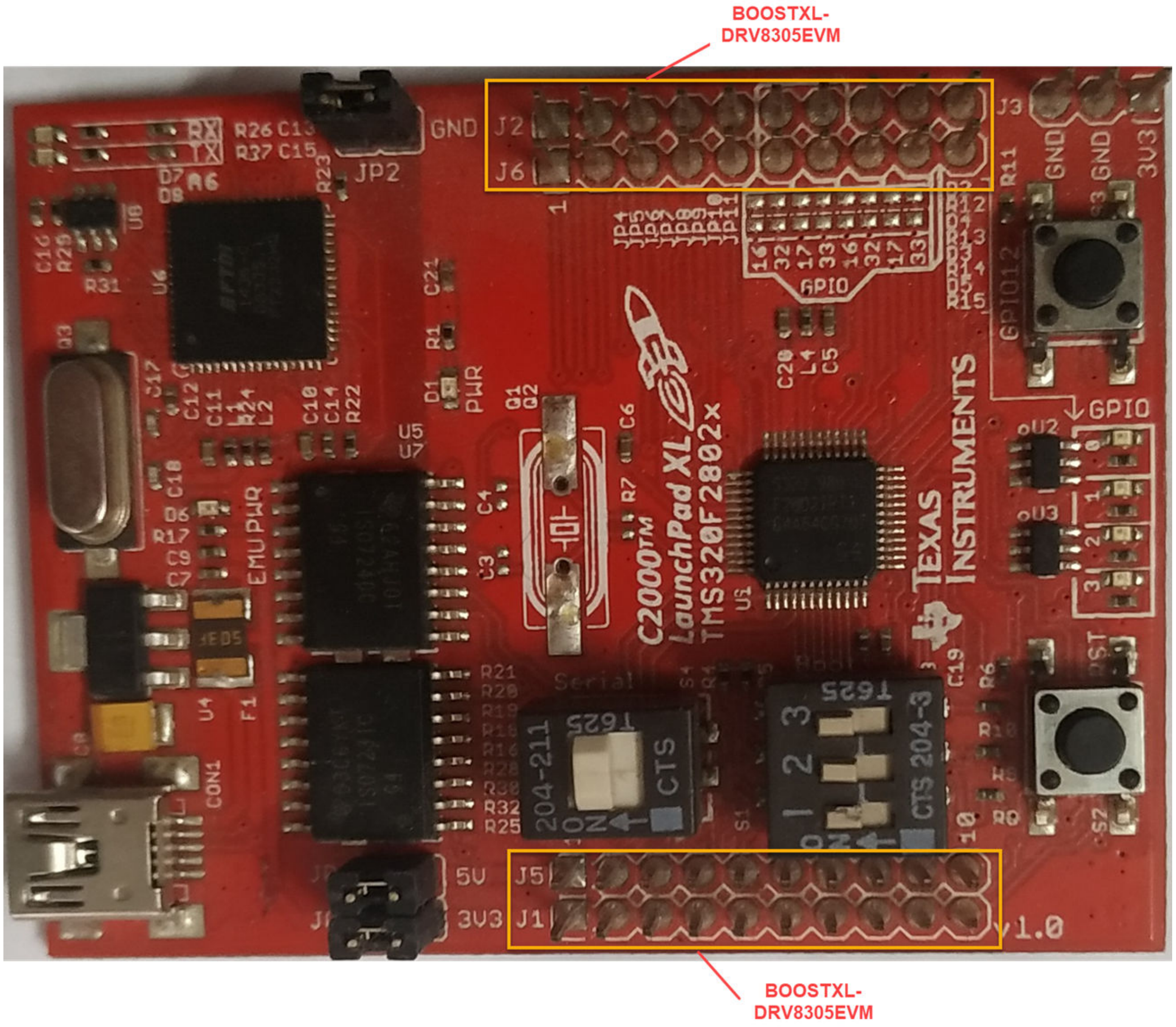
1. Attach the BOOSTXL inverter board to J1, J2, J3, J4 on the LAUNCHXL-F28069M or F28379D or F280049C and J1, J2, J5, J6 for LAUNCHXL -F28027/27F.

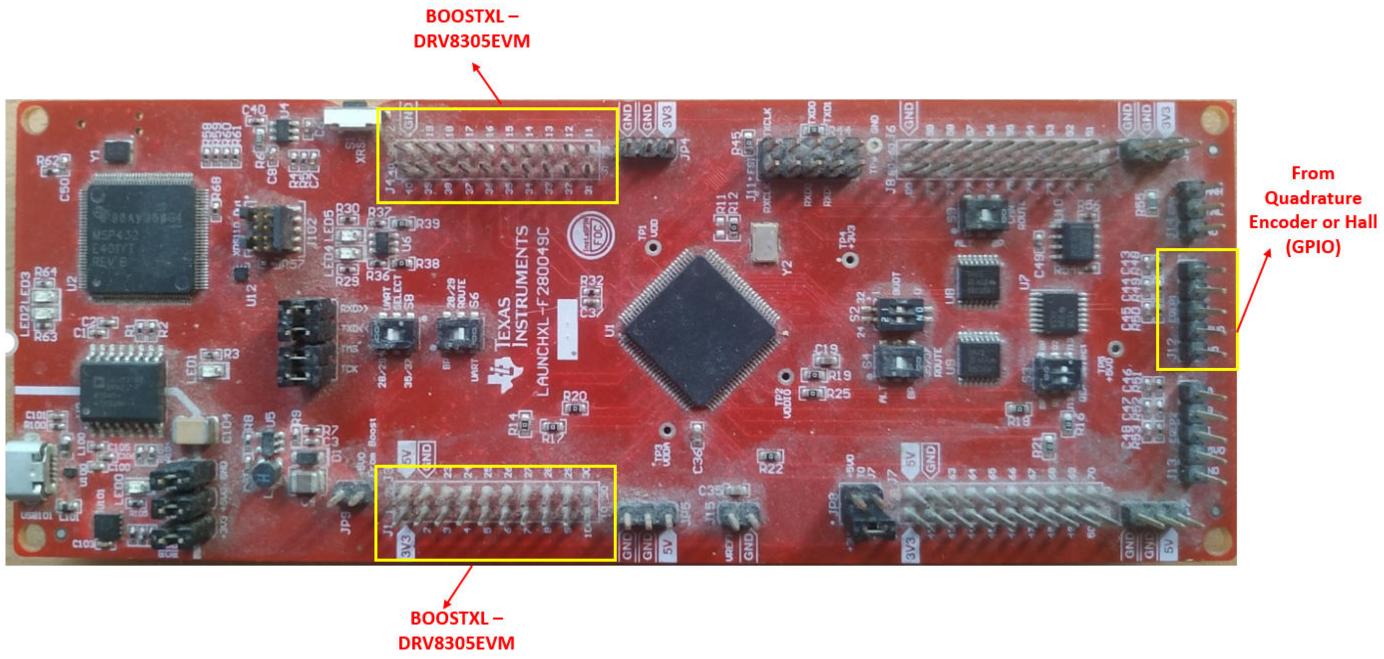
   Note Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J1, J2 of LAUNCHXL.

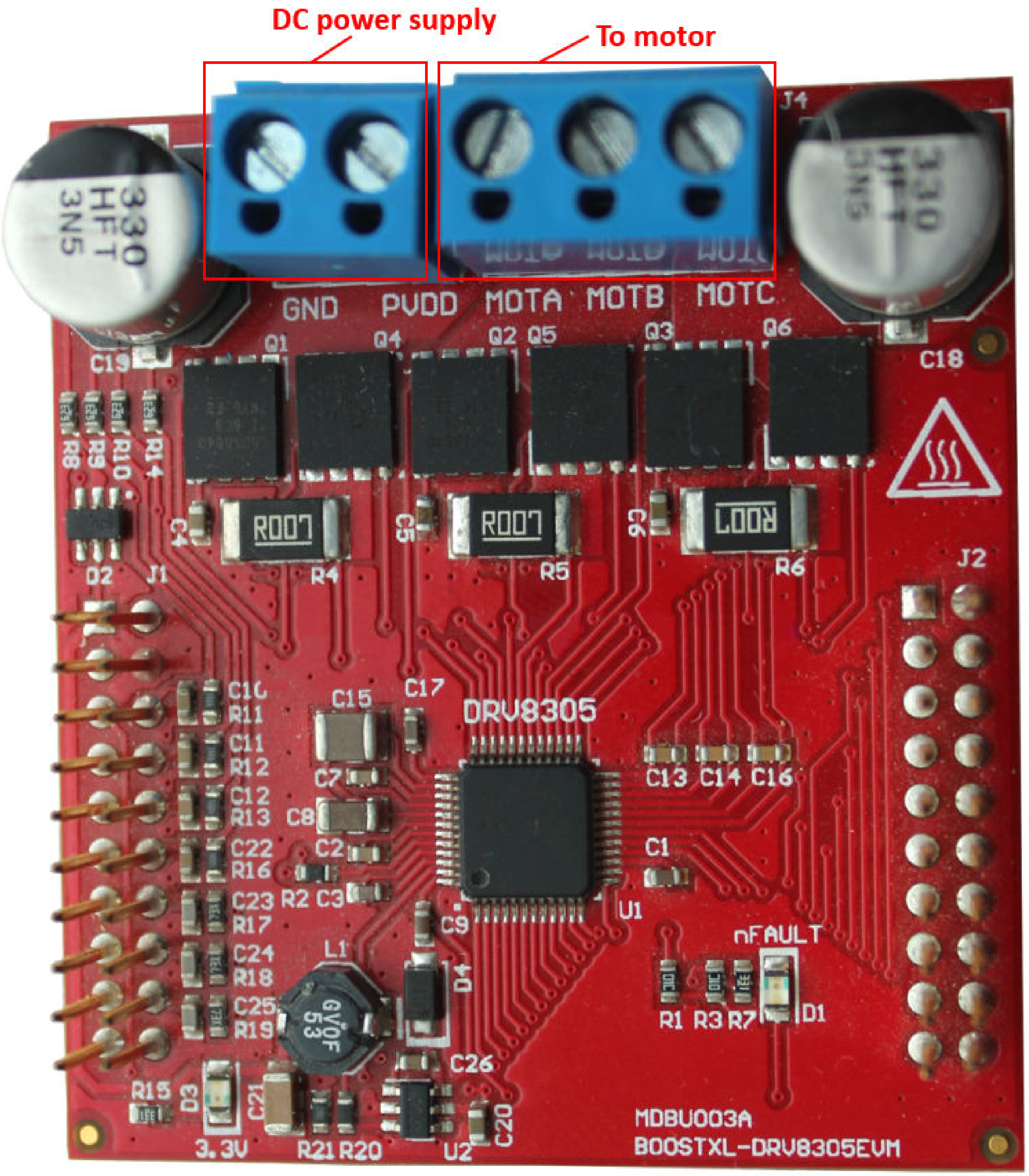
2. Connect the motor three phases, to MOTA, MOTB, and MOTC on the BOOSTXL inverter board.
3. Connect the DC power supply (24V) to PVDD and GND on the BOOSTXL inverter board.

Warning Be careful when connecting PVDD and GND to the positive and negative connections of the DC power supply. A reverse connection can damage the hardware components.




BOOSTXL-DRV8305EVM
BOOSTXL-DRV8305EVM


BOOSTXL – DRV8305EVM
From Quadrature Encoder or Hall (GPIO)
BOOSTXL – DRV8305EVM

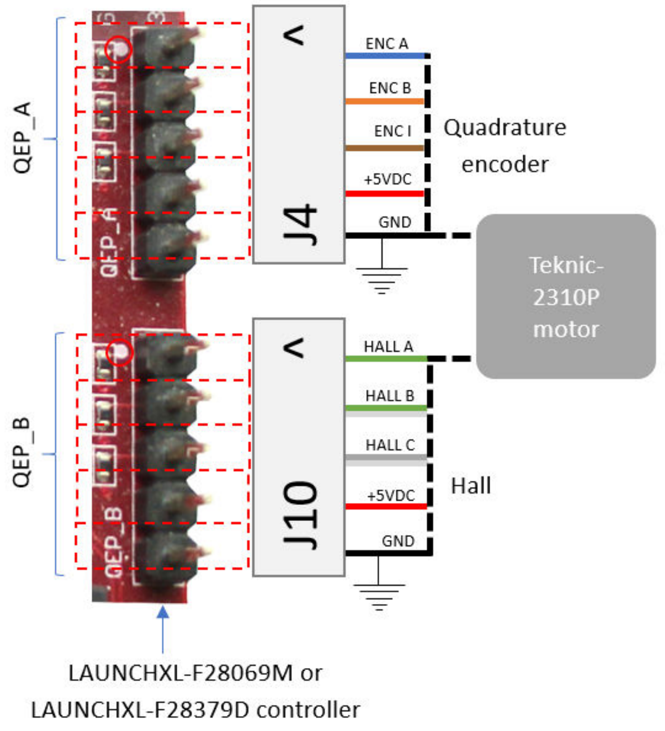


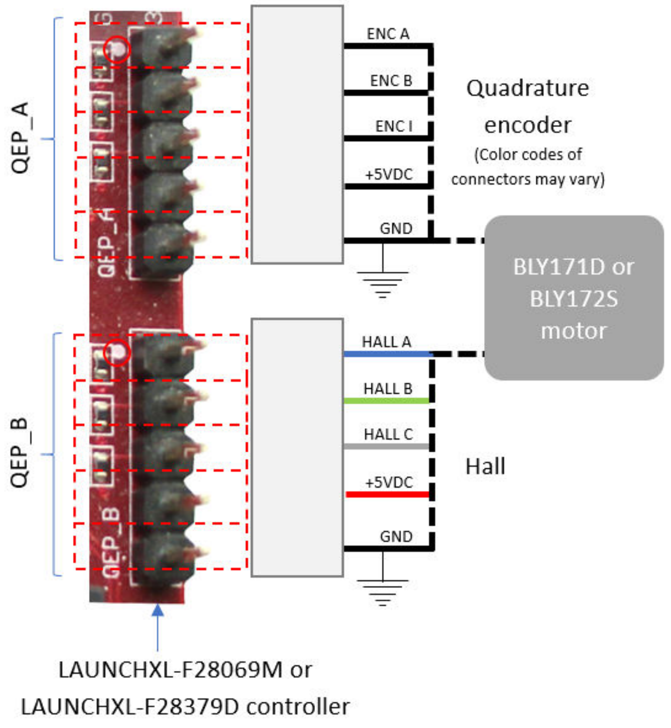
The following step describes about interfacing the quadrature encoder sensor:

- Connect the quadrature encoder pins (G, I, A, 5V, B) to QEP_A on the LAUNCHXL- F28379D/ F28069M controller board or EQEP1 (J12) of the LAUNCHXL-F280049C.

To implement position-sensing by using Hall sensor, use a motor that has inbuilt Hall sensors (for example, Teknic motor M-2310P, BLY171D and BLY172S). The following steps describe the steps to interface the Hall sensor:

- Connect the Hall sensor encoder output to a GPIO port that is configured as eCAP, on the LAUNCHXL controller board (QEP_B for LAUNCHXL-F28379D, EQEP1 (J12) for LAUNCHXL-F280049C).





We recommend the following jumper settings for the LAUNCHXL inverter boards when working with Motor Control Blockset. You can customize these settings depending on the application requirements. For more information about these settings, see the device user guide available on Texas Instruments website.

For LAUNCHXL-F28069M controller

- JP1 – ON
- JP2 – ON
- JP3 – ON
- JP4 – ON
- JP5 – ON
- JP6 – OFF
- JP7 – ON

For LAUNCHXL-F28379D controller

- JP1 – ON
- JP2 – ON
- JP3 – ON
- JP4 – ON
- JP5 – ON
- JP6 – OFF

For LAUNCHXL-F28027 controller

- S4 – ON

For LAUNCHXL-F280049C controller

- S8 – 0, S6 – 0 : Required for serial communication.
- For QEP1, S3 position 2 : down (0), S4 – UP (1)
- For QEP2, S3 position 1 : down (0)
- JP1, JP2, JP3, JP8: ON

Instructions for Dyno (Dual Motor) Setup

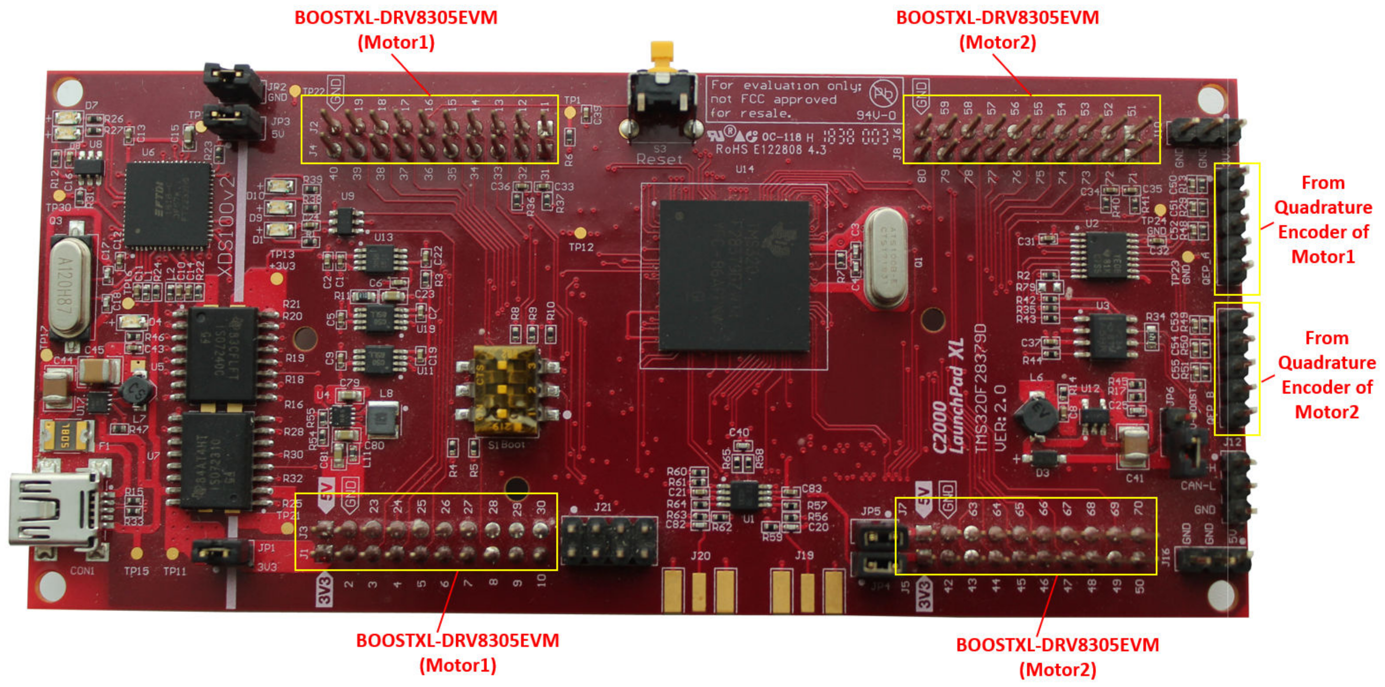
1. Connect the three phases of Motor1 and Motor2, to MOTA, MOTB, and MOTC on the corresponding BOOSTXL inverter boards.
2. Attach the BOOSTXL inverter board (connected to Motor1) to J1, J2, J3, J4 on the LAUNCHXL controller board.

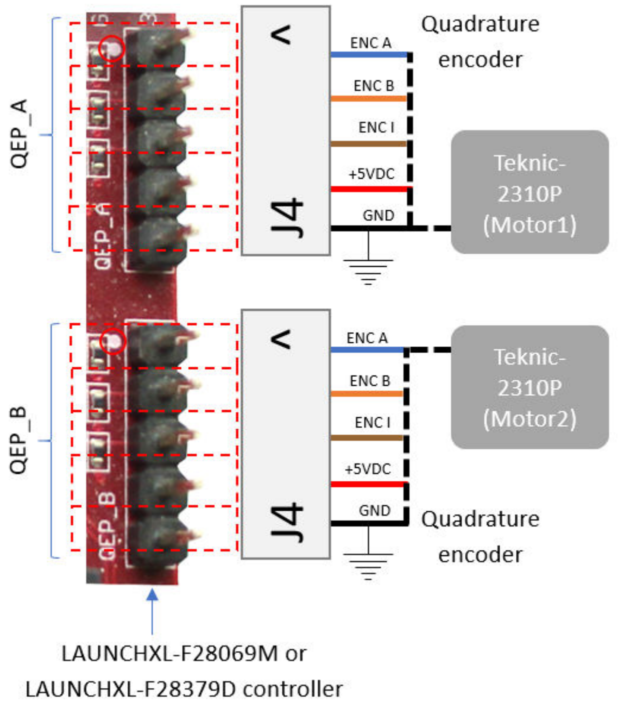
   Note Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J1, J2 of LAUNCHXL.
3. Attach the BOOSTXL inverter board (connected to Motor2) to J5, J6, J7, J8 on the LAUNCHXL controller board.

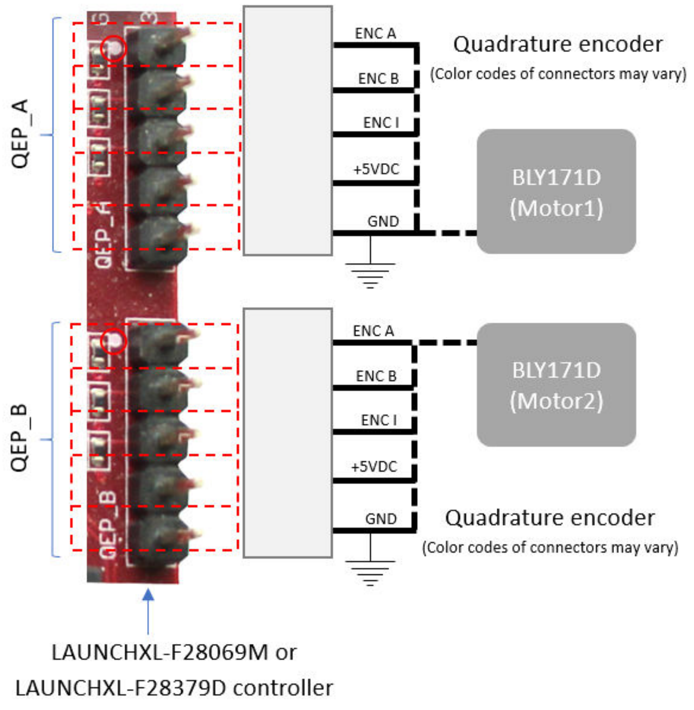
   Note Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J5, J6 of LAUNCHXL.
4. Connect the DC power supply (24V) to PVDD and GND on both BOOSTXL inverter boards.

   Note Connect the PVDD and GND on the BOOSTXL boards (for MOTOR1 and MOTOR2) to the same power supply. When one motor consumes power, the second motor generates power. If you connect both motors to the same power supply, the power generated by one motor is consumed by the other motor. The DC power supply delivers power only for the losses.
5. Connect the quadrature encoder pins of Motor1 (G, I, A, 5V, B) to QEP_A on the LAUNCHXL controller board.
6. Connect the quadrature encoder pins of Motor2 (G, I, A, 5V, B) to QEP_B on the LAUNCHXL controller board.

Warning Be careful when connecting PVDD and GND to the positive and negative connections of the DC power supply. A reverse connection can damage the hardware components.


BOOSTXL-DRV8305EVM (Motor1)
BOOSTXL-DRV8305EVM (Motor2)
From Quadrature Encoder of Motor1
From Quadrature Encoder of Motor2
BOOSTXL-DRV8305EVM (Motor1)
BOOSTXL-DRV8305EVM (Motor2)




QEP_A
QEP_B
ENC A
ENC B
ENC I
+5VDC
GND
Quadrature encoder
(Color codes of connectors may vary)
BLY171D (Motor1)
BLY171D (Motor2)
LAUNCHXL-F28069M or LAUNCHXL-F28379D controller

TMDSRSLVR C2000 Resolver to Digital Conversion Kit

The TMDSRSLVR C2000 Resolver to Digital Conversion Kit configuration includes the following hardware components:

- LAUNCHXL-F28069M controller

- BOOSTXL-DRV8305 (supported inverter)
- DC power supply
- TMDSRSLVR C2000 Resolver to Digital Conversion Kit (Resolver Eval Kit [R2])
- Resolver encoder

The following steps describe the hardware connections for the TMDSRSLVR board:

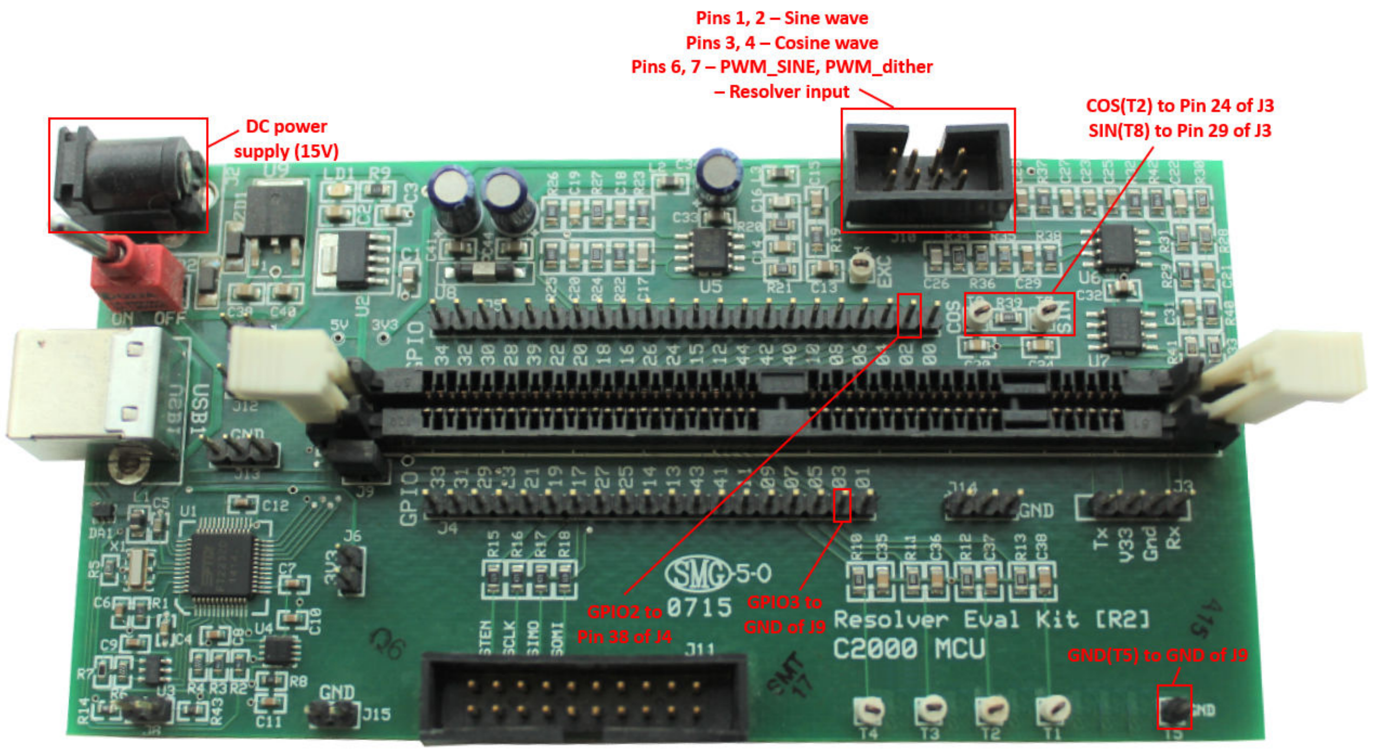
1. Connect DC power supply (15V) to J2 on the TMDSRSLVR board.
2. Connect the resolver output pins for sine wave to pins 1, 2 of J10 on the TMDSRSLVR board.
3. Connect the resolver output pins for cosine wave to pins 3, 4 of J10 on the TMDSRSLVR board.
4. Connect the resolver input pins to the PWM_dither and PWM_SINE pins of J10 on the TMDSRSLVR board.

The following step describes the hardware connection for the LAUNCHXL-F28069M controller board:

- Connect the LAUNCHXL-F28069M controller board to a computer via USB port.

The following steps describe the hardware connections between the MCU Resolver Eval Kit [R2] and LAUNCHXL-F28069M controller boards:

1. Connect the COS(T2) pin on the TMDSRSLVR board to pin 24 of J3 on the LAUNCHXL-F28069M controller board.
2. Connect the SIN(T8) pin on the TMDSRSLVR board to pin 29 of J3 on the LAUNCHXL-F28069M controller board.
3. Connect the GPIO2 pin on the TMDSRSLVR board to pin 38 of J4 on the LAUNCHXL-F28069M controller board.



Overview of CLA Configuration for C2000 Processors Using Subsystem

The control law accelerator (CLA) is a coprocessor available with the TI C2000 processor that allows parallel processing. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently.

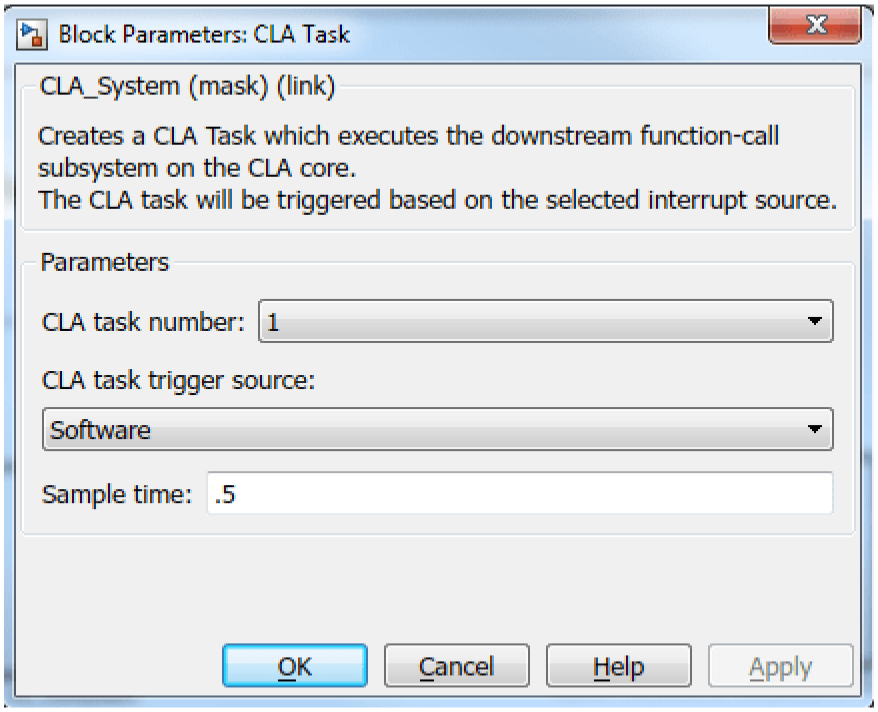
These sections explain:

In this section...

CLA Task

A CLA Task block is used here to trigger a CLA task. This block runs the downstream function-call subsystem on the CLA. In the block mask of the CLA Task block, specify the CLA task number and the associated interrupt triggering source. Ensure that you are using the CLA Trigger block from the library corresponding to your hardware board selected in the Configuration Parameters. CLA task trigger source options are different for different processors. The downstream function-call subsystem is executed using the selected CLA Task when the selected interrupt triggers. For more, see C28x CLA Task.

For example, CLA Task1 is used and **Software** is selected as the trigger source, which means that the CLA task is triggered at the sample rate provided in the sample time parameter.


Block Parameters: CLA Task

CLA_System (mask) (link)

Creates a CLA Task which executes the downstream function-call subsystem on the CLA core.
The CLA task will be triggered based on the selected interrupt source.

Parameters

CLA task number: 1

CLA task trigger source: Software

Sample time: .5

Interrupt Generation after CLA Task Completion

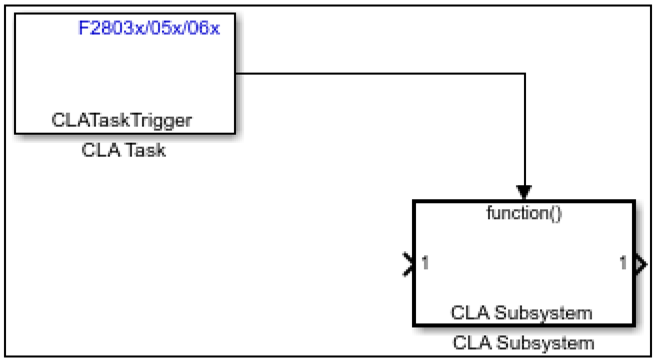
A CLA interrupt can be generated when the CLA Task completes. CLA Task1 interrupt is used as the interrupt source on the C28x Hardware Interrupt block. The function-call subsystem triggered by the Hardware Interrupt block with CLA task interrupt (CPU = 11, PIE = 1-8) is executed at the end of the CLA task when the particular interrupt occurs.

Note When a **CLA Task** is triggered by an interrupt other than **software**, ensure that the interrupt gets cleared at the end of the CLA task. This is achieved automatically when you use **Hardware Interrupt** block with CLA end of the task interrupt or the CLA source trigger interrupt.

CLA Subsystem

A CLA Subsystem block contains a subset of blocks within a model or system. CLA Subsystem is a triggered subsystem which is caused by a CLA Task block.

CLA Subsystem block can only be used along with CLA Task block. For more, see CLA Subsystem.



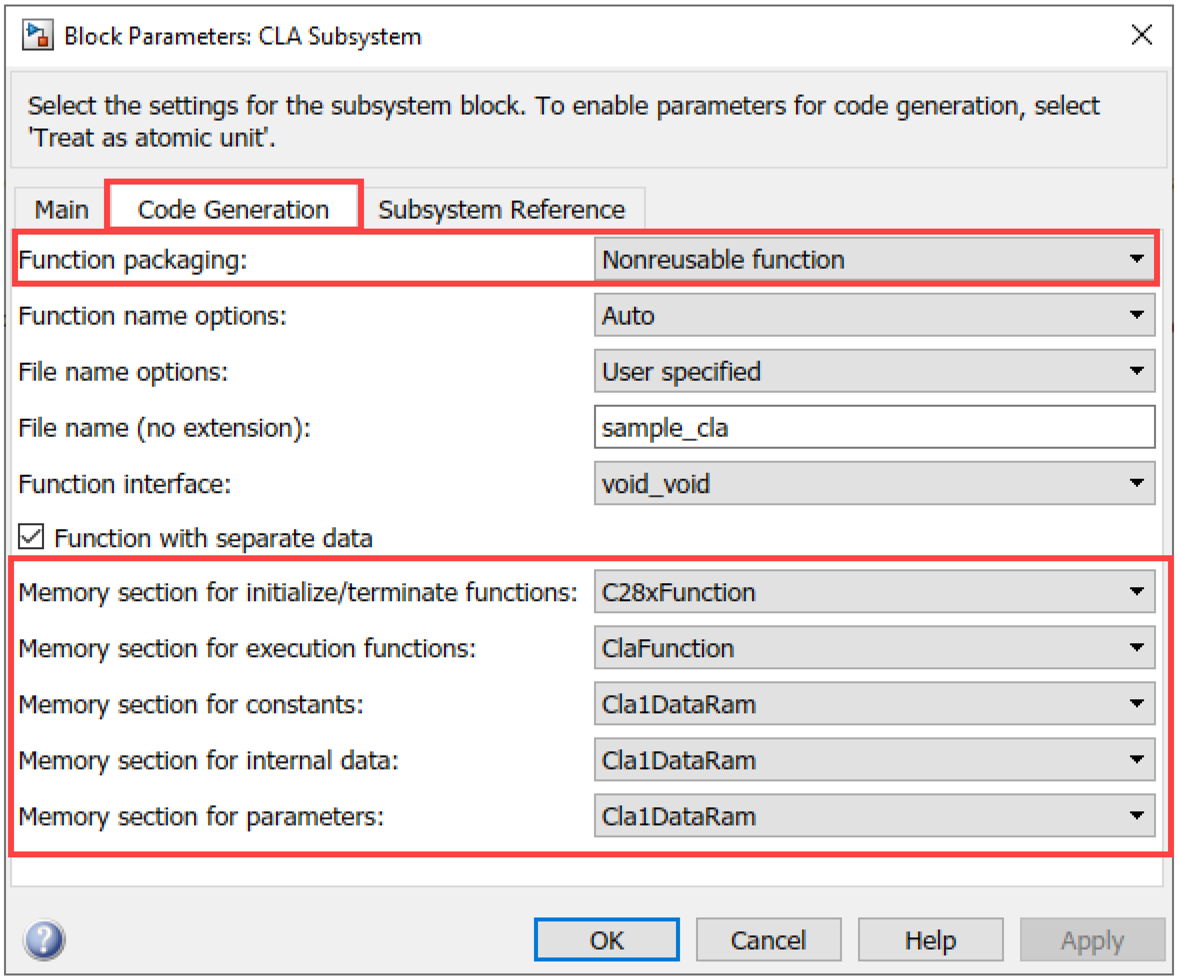
CLA Subsystem requires you to load `tic2000demospkg` package in the model with a valid TI C2000 based hardware board.

- Open **Configuration Parameters** > **Hardware Implementation** > **Hardware board** and change the parameter to a valid `TI C2000` board.
- After selecting TI C2000 hardware board, click **Manage packages** > **Refresh** > **Load** from Embedded Coder dictionary app to load `tic2000demospkg`. For more information, see “Data Exchange Between CLA and C28x CPU” on page 1-97.

Method 1 - Nonreusable Function Code Generation for CLA Subsystem (Recommended)

In the CLA Subsystem block, the **Function packaging** parameter on the Code Generation tab is set to is `Nonresuable` by default. To open the block parameters, right-click on the CLA subsystem and select **Block Parameters(Subsystem)**.

Note This method requires `tic2000demospkg` package to be loaded in the model with TI C2000 hardware board. For more, see “Data Exchange Between CLA and C28x CPU” on page 1-97.



Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Function packaging:	Nonreusable function
Function name options:	Auto
File name options:	User specified
File name (no extension):	sample_cla
Function interface:	void_void
✓ Function with separate data	
Memory section for initialize/terminate functions:	C28xFunction
Memory section for execution functions:	ClaFunction
Memory section for constants:	Cla1DataRam
Memory section for internal data:	Cla1DataRam
Memory section for parameters:	Cla1DataRam

- The CLA Subsystem block generates the CLA algorithm code as a separate `.C file` along with the data.

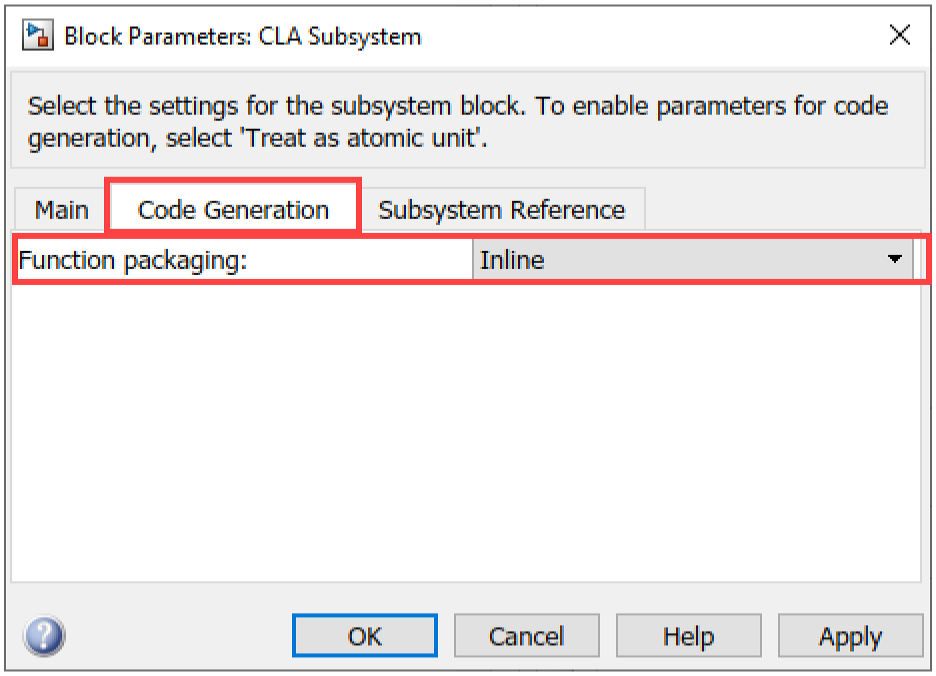
  The initialize/terminate functions are prefixed with compiler directives so it is compiled with only C complier and execution functions prefixed to ensure that it gets compiled by CLA compiler.
- The constants, parameters, and internal data are configured automatically to be stored in `Cla1DataRAM`.
- Save only the input signals from CPU to CLA as a signal and store them in the `CpuToCla1MsgRAM` storage class using the Code Mappings editor.
- Use this method to **Monitor & Tune (External mode)** signals outside the CLA subsystem.

Compatibility Considerations

The **Nonreusable Function Code Generation for CLA Subsystem** feature is available starting R2021b.

Method 2 - Inline Code Generation for CLA Subsystem

In the CLA Subsystem block, select the **Function packaging** as `Inline` under the **Code Generation** tab of the Block Parameters. To open the Block Parameters, right-click on the CLA subsystem and select **Block Parameters(Subsystem)**.


Block Parameters: CLA Subsystem

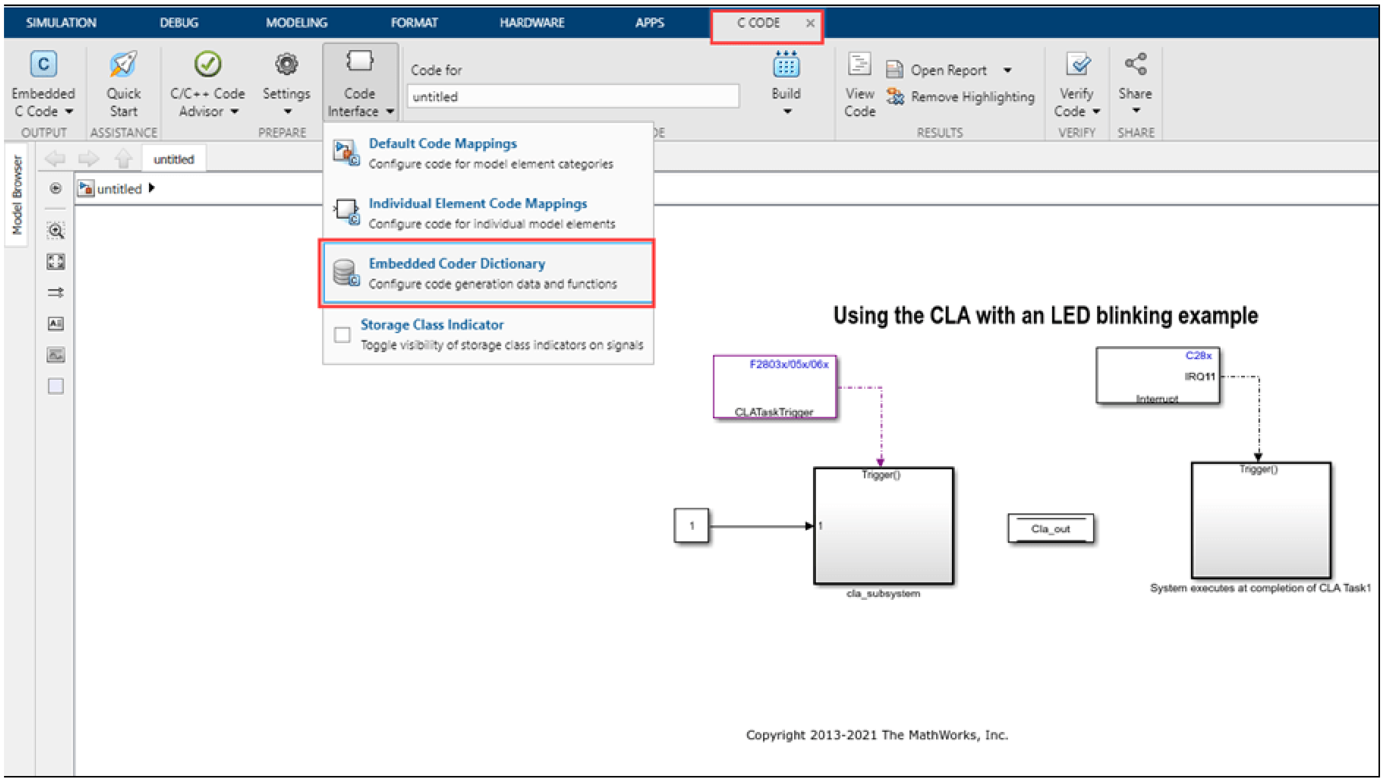
Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

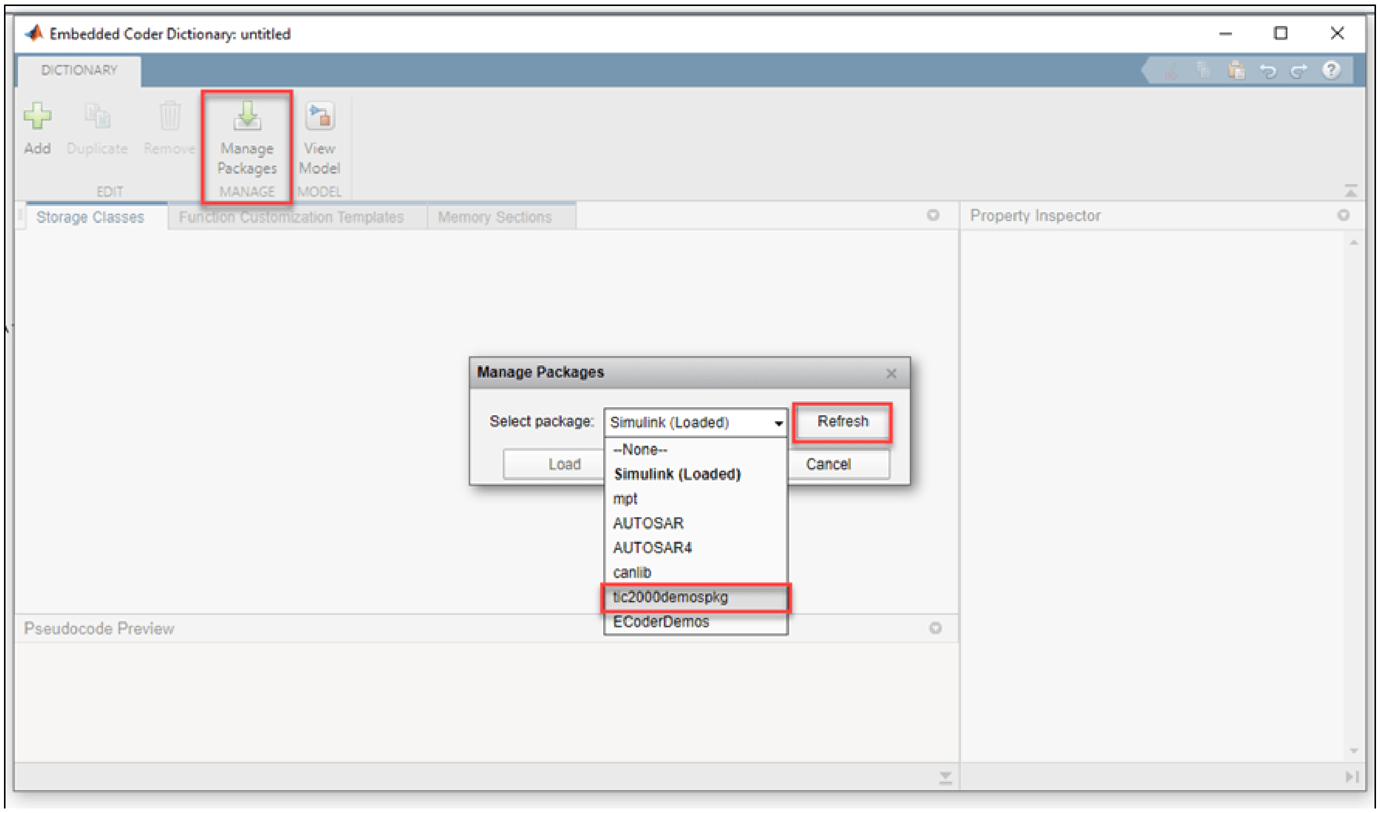
Function packaging: Inline

- In this method, the entire algorithm inside CLA gets `inlined` and is part of CLA task file (cla_task.cla).
- All the input signals to the CLA must be configured manually to store `CpuToCla1MsgRAM` storage class.
- All the output signals from the CLA to CPU configured manually to store `Cla1ToCpuMsgRAM` storage class.
- All discrete state variables used inside CLA function-call subsystems configured manually to store `Cla1DataRam`. For example, delay blocks and integrators must be set to use the Cla1DataRam storage class for state variables. See the **State Attributes** of the Delay block inside `cla_subsystem` in `c28035blink_cla.slx` example.
- With this method, you cannot perform **Monitor & Tune (External mode)** for the signals outside CLA subsystem.

Data Exchange Between CLA and C28x CPU

CLA storage classes can be loaded in the model (**Embedded Coder** > **Code Interface** > **Embedded Coder Dictionary** > **Manage Packages**) as shown.


SIMULATION
DEBUG
MODELING
FORMAT
HARDWARE
APPS
C CODE
Embedded C Code
Quick Start
C/C++ Code Advisor
Settings
Code Interface
Code for
untitled
Build
View Code
Open Report
Remove Highlighting
Verify Code
Share
OUTPUT
ASSISTANCE
PREPARE
RESULTS
VERIFY
SHARE
Default Code Mappings
Configure code for model element categories
Individual Element Code Mappings
Configure code for individual model elements
Embedded Coder Dictionary
Configure code generation data and functions
Storage Class Indicator
Toggle visibility of storage class indicators on signals
Using the CLA with an LED blinking example
F2803x/05x/06x
CLATaskTrigger
C28x
IRQ11
Interrupt
Trigger()
cla_subsystem
Cla_out
System executes at completion of CLA Task1
Copyright 2013-2021 The MathWorks, Inc.


Embedded Coder Dictionary: untitled
DICTIONARY
Add
Duplicate
Remove
Manage Packages
View Model
EDIT
MANAGE
MODEL
Storage Classes
Function Customization Templates
Memory Sections
Property Inspector
Pseudocode Preview
Manage Packages
Select package:
Simulink (Loaded)
Refresh
Load
Cancel
--None--
Simulink (Loaded)
mpt
AUTOSAR
AUTOSAR4
canlib
tic2000demospkg
ECoderDemos

The Code Mappings editor is the primary location to configure model data elements for code generation. For more information, refer to Code Mappings Editor – C (Simulink Coder).

All the interfaces between the CLA and the CPU need to be placed in specific memory sections. To gain specific access to these sections, Select **Embedded Coder > Code Mappings > Data stores** and **Signal/States** for CpuToCla1MsgRAM and Cla1ToCpuMsgRAM. For more information, refer to Code Mappings Editor – C (Simulink Coder).

Memory Section Data

The data in the model is stored in the various memory sections which has the following access rules.

Memory Storage Class

Memory Section	CPU	CLA
CpuToCla1MsgRAM	Read & Write	Read only
Cla1ToCpuMsgRAM	Read only	Read & Write
Cla1DataRAM	Read & Write	Read & Write

Note

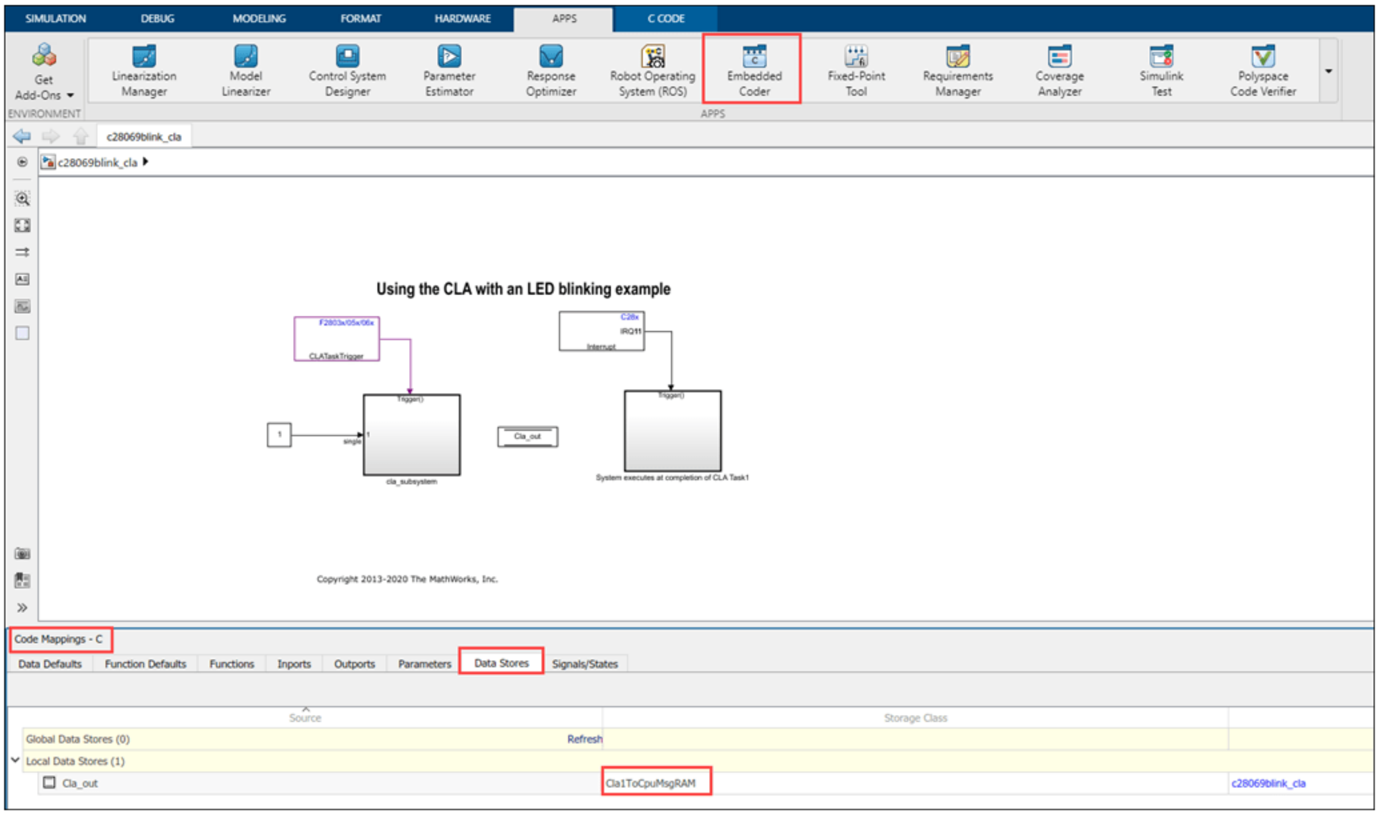
- Signal or states with `Cla1ToCpuMsgRAM` storage class will add initialization in the function compiled by CPU (model initialization function) but CPU does not have write access to the above memory section. If initialization is required for a particular variable and needs to be editable by CLA as well, it is recommended to use `Cla1DataRAM`.
- For F2803x processor, the `Cla1DataRAM` behave similar to `Cla1ToCpuMsgRam` i.e. CPU does not have write access to any of the above storage class except `CpuToCla1MsgRAM`.

Memory Section Code

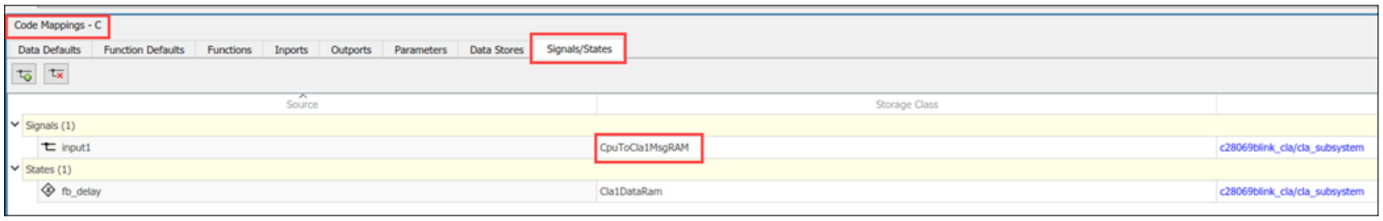
The functions with the following memory section are prefixed with compiler directives to control the compilation of the code.

- **C28xFunction** is compiled by `C28x compiler`
- **ClaFunction** is compiled by `CLA compiler`.

The following figures show the configurations of specific memory section between CPU and CLA in the model.



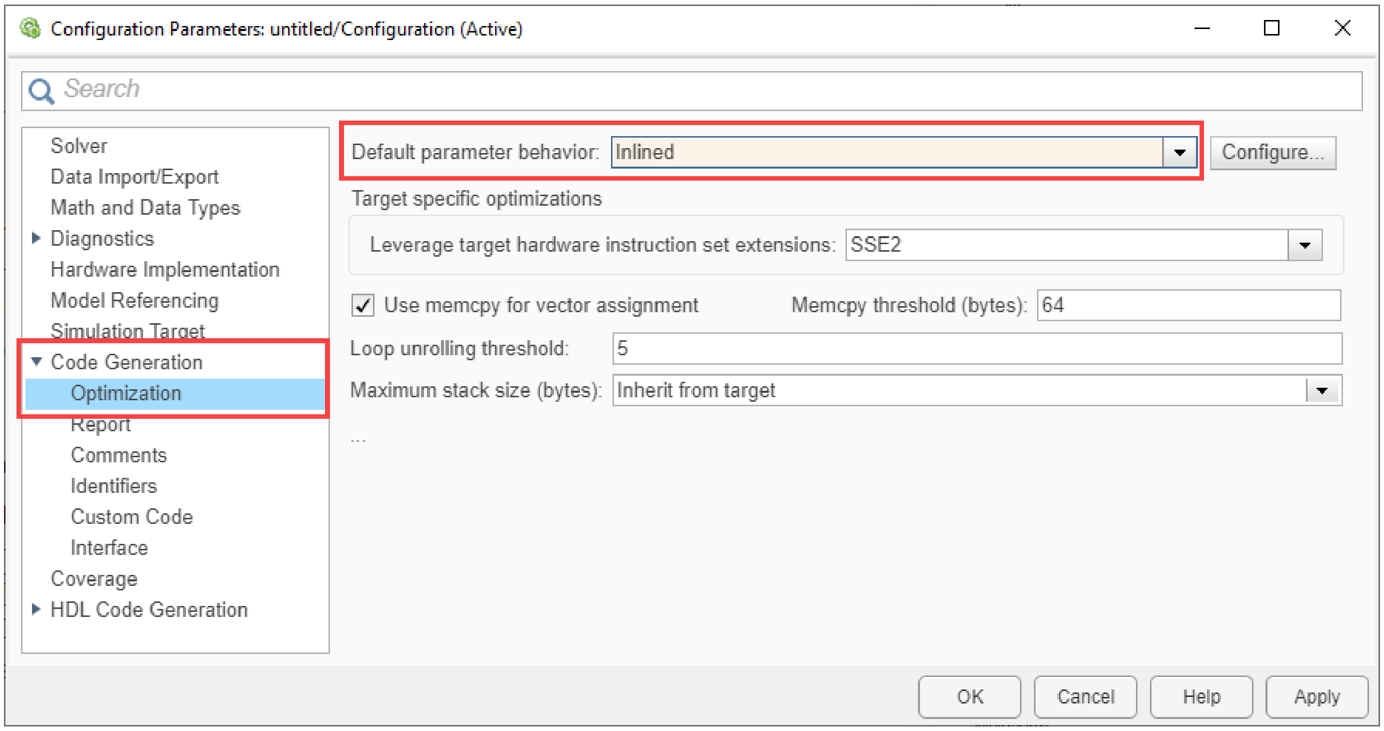
Source	Storage Class	
Global Data Stores (0) Refresh		
Local Data Stores (1)		
Cla_out	Cla1ToCpuMsgRAM	c28069blink_cla



Source	Storage Class	
Signals (1)		
input1	CpuToCla1MsgRAM	c28069blink_cla/cla_subsystem
States (1)		
fb_delay	Cla1DataRam	c28069blink_cla/cla_subsystem

Required Model Configuration When Using CLA

In **Model Configuration Parameters** > **Code Generation** > **Optimization**, the **Default parameter behavior** parameter must be set to `Inlined`. This avoids the creation of model structure global variables, as CLA cannot access global data.


Configuration Parameters: untitled/Configuration (Active)

Default parameter behavior: Inlined

Target specific optimizations

Leverage target hardware instruction set extensions: SSE2

☑ Use memcpy for vector assignment

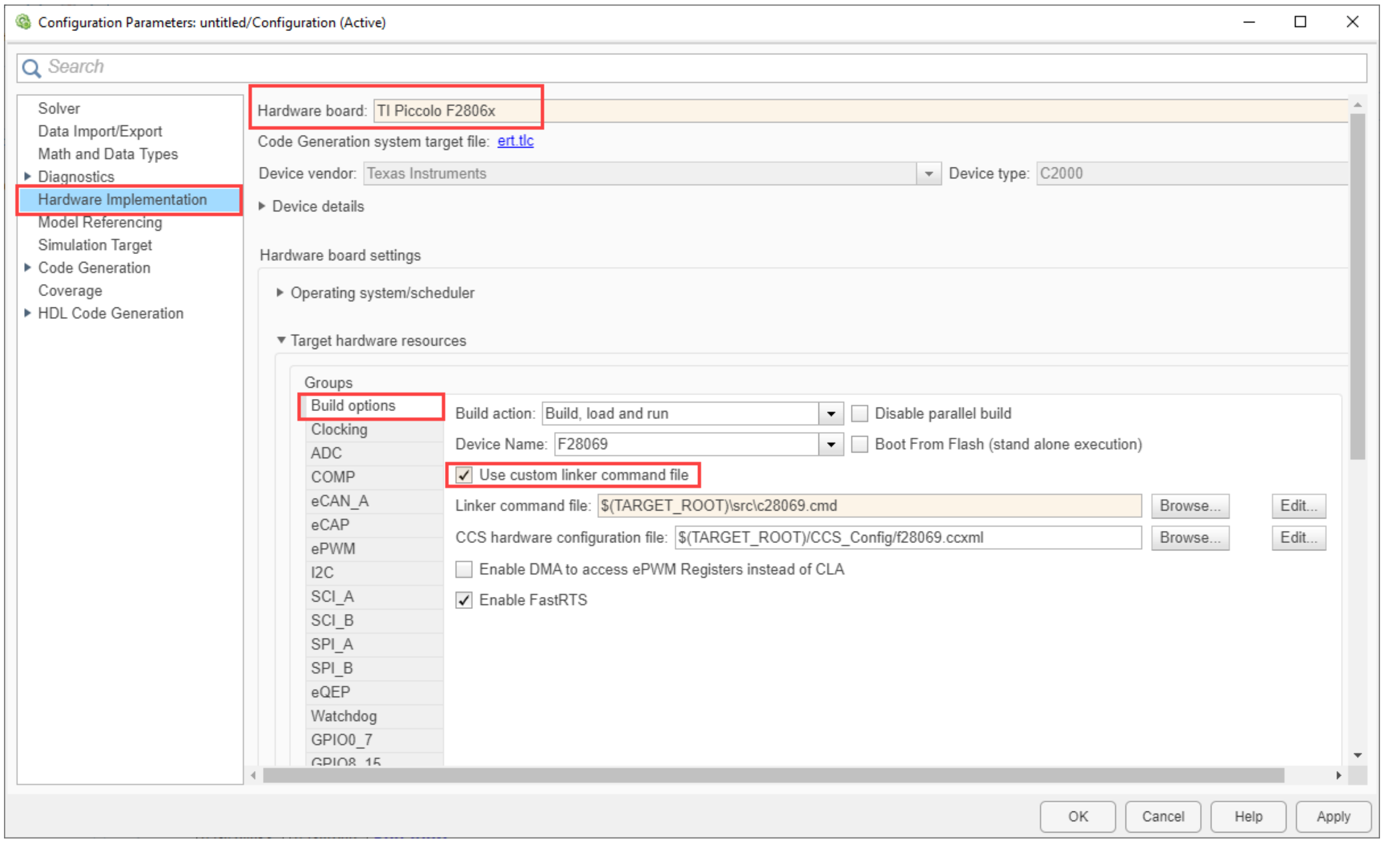
Memcpy threshold (bytes): 64

Loop unrolling threshold: 5

Maximum stack size (bytes): Inherit from target

Change Standard Linker Command File

For F28035 and F28069 processors, a specific linker command file has to be selected to assign CLA memory sections. In the Model Configuration Parameters, under **Hardware Implementation > Target hardware resources > Build options**, and select **Use custom linker command file**. A preconfigured `c28069_cla.cmd` is used as linker command file. This file adds CLA memory sections description and can be found in the `src` directory at the root of the blockset installation.


Configuration Parameters: untitled/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation

Hardware board: TI Piccolo F2806x
Code Generation system target file: ert.tlc
Device vendor: Texas Instruments   Device type: C2000
Device details

Hardware board settings
Operating system/scheduler
Target hardware resources

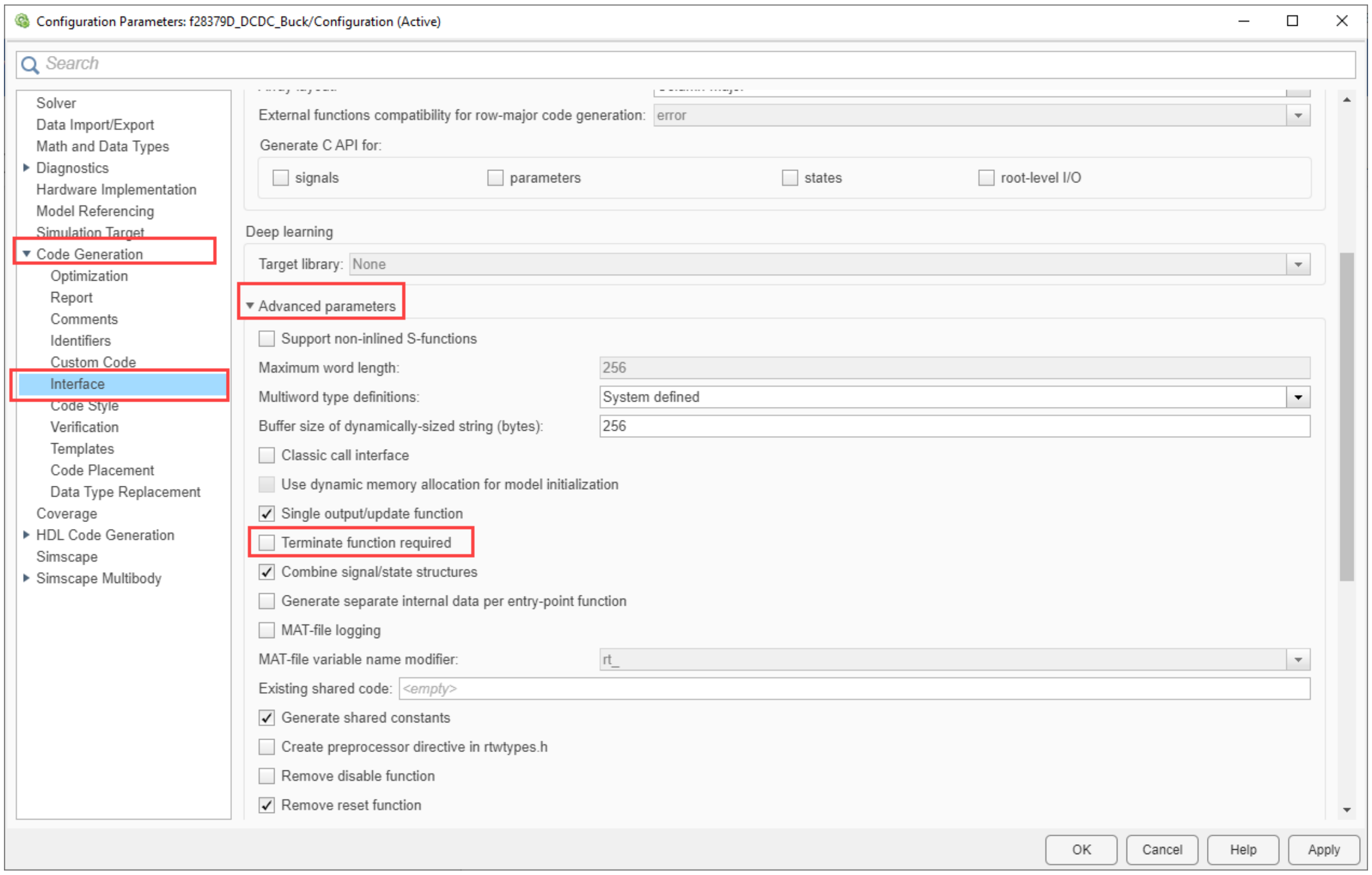
Groups
Build options
Clocking
ADC
COMP
eCAN_A
eCAP
ePWM
I2C
SCI_A
SCI_B
SPI_A
SPI_B
eQEP
Watchdog
GPIO0_7
GPIO8_15

Build action: Build, load and run   ☐ Disable parallel build
Device Name: F28069   ☐ Boot From Flash (stand alone execution)
☑ Use custom linker command file
Linker command file: $(TARGET_ROOT)\src\c28069.cmd   Browse...   Edit...
CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28069.ccxml   Browse...   Edit...
☐ Enable DMA to access ePWM Registers instead of CLA
☑ Enable FastRTS

OK   Cancel   Help   Apply

Profiling with XCP External Mode

When performing profiling with XCP External mode for a model containing CLA subsystem, ensure to disable the parameter **Terminate function required** under **Configuration Parameters > Code Generation > Interface > Advanced parameters**.


Configuration Parameters: f28379D_DCDC_Buck/Configuration (Active)

External functions compatibility for row-major code generation: error

Generate C API for:

- [ ] signals
- [ ] parameters
- [ ] states
- [ ] root-level I/O

Deep learning

Target library: None

Advanced parameters

- [ ] Support non-inlined S-functions

Maximum word length: 256

Multiword type definitions: System defined

Buffer size of dynamically-sized string (bytes): 256

- [ ] Classic call interface
- [ ] Use dynamic memory allocation for model initialization
- [x] Single output/update function
- [ ] Terminate function required
- [x] Combine signal/state structures
- [ ] Generate separate internal data per entry-point function
- [ ] MAT-file logging

MAT-file variable name modifier: rt_

Existing shared code: <empty>

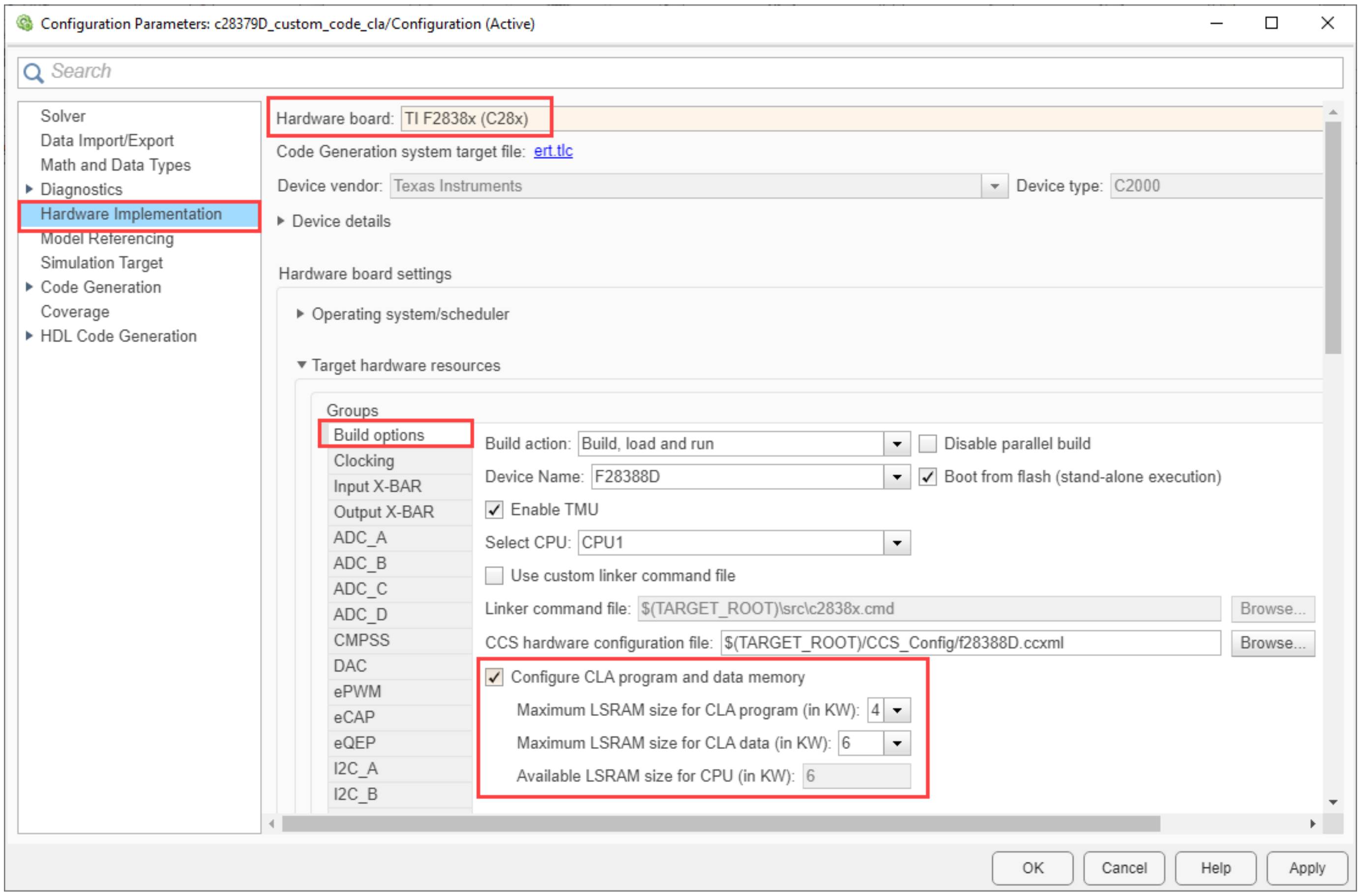
- [x] Generate shared constants
- [ ] Create preprocessor directive in rtwtypes.h
- [ ] Remove disable function
- [x] Remove reset function

Note For a model containing CLA, running profiling with XCP External mode, will not profile the tasks executed in CLA.

CLA LSRAM Memory Configuration

In the Model Configuration Parameters, you can configure the local shared RAM (LSRAM) as CLA program and data memory.

In Model Configuration Parameters, navigate to **Hardware Implementation** > **Target hardware resources** > **Build options** and select **Configure CLA program and data memory**. For information, refer to “C28x-Build Options”.


Configuration Parameters: c28379D_custom_code_cla/Configuration (Active)

Hardware board: TI F2838x (C28x)

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments   Device type: C2000

Hardware board settings

Target hardware resources

Build action: Build, load and run   ☐ Disable parallel build

Device Name: F28388D   ☑ Boot from flash (stand-alone execution)

☑ Enable TMU

Select CPU: CPU1

☐ Use custom linker command file

Linker command file: $(TARGET_ROOT)\src\c2838x.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28388D.ccxml

☑ Configure CLA program and data memory

Maximum LSRAM size for CLA program (in KW): 4

Maximum LSRAM size for CLA data (in KW): 6

Available LSRAM size for CPU (in KW): 6

The following table describes the total LSRAM memory available for the processors.

CLA LSRAM Memory Allocation

Processor	Total LSRAM Memory available in KiloWords (KW)
F2837x/F2807x	12
F28004x/F2838x	16

Debug CLA

The debug function `/*__mdebugstop()*/` is present at the beginning of the CLA task (`__interrupt void Cla1Task1 ( void )`) in the generated `cla_task.cla` file.

1. Enable the debug function `__mdebugstop()` by removing the block comments around it.
2. Compile the source code or build the CCS project.

Code Replacement Library in CLA

The following trigonometric function blocks from Simulink will be replaced by the corresponding CLA math library functions if used within the subsystem triggered by CLA task.

Note For CRL replacement inside CLA, a tag is added to the function call subsystem triggered by CLA task. Do not use this tag for any other subsystem.

- Sin with **approximation** set as `none`
- Cos with **approximation** set as `none`
- asin, acos, atan, atan2, log to the base 10, log to the base e, exponential function, and square root function.

CLA Limitations and Troubleshooting

Limitations

You need to follow certain modeling practices because of specific interactions between the CLA and the C28x CPU and because of CLA C compiler limitations.

- Only a C28x event can trigger the CLA application code. The C28x CPU can trigger a CLA Task via software or by using different peripheral interrupts.
- Place all the interfaces between the CLA and the CPU in specific memory locations. Use the CpuToCla1MsgRAM memory section to exchange data from the C28x to the CLA. Use the Cla1ToCpuMsgRAM memory section to exchange data from the CLA to the C28x.
- Constants with sample time as `inf` and mapped as `CpuToCla1MsgRAM` or `Cla1ToCpuMsgRAM` is always initialized to `0`. Ensure that the sample time for such constant is set to `-1` or `any positive integer`.
- For a model containing CLA subsystem running in external mode, the data cannot be logged inside the CLA subsystem. You have to move the data out of the CLA subsystem to monitor the data.
- The CLA application code does not have access to global variables.
- Recursive function calls are not supported.
- The CLA C compiler does not support integer division and integer unsigned comparison. Use MATLAB function to access CLA math library functions provided by TI for the above operation.
- You cannot use more than one instance of DAC block with same module when one instance is used inside CLA.
- **SPI**, **CAN** and **SCI** blocks are not supported inside CLA.
- The blocks which create reset functions cannot be used inside CLA. It is not possible to control the definition and compilation of the reset function to CLA core from Simulink. For example, PID integrator and conditional blocks which resets the states.
- When you use constants inside MATLAB function, it is stored in constant section of C28x which CLA is unable to access. You can overcome this problem by using constants in Simulink with proper storage classes configured and using the same as input to MATLAB function.

Consider these limitations when creating a model in Simulink.

Other Guidelines to Consider While Using CLA

Avoid the creation of sub-functions using atomic subsystems inside CLA function-call subsystems as this creates nested functions that are not supported on the CLA.

Certain CLA configurations might require more operations on the model or might prevent you from using Simulink features. See the list of limitations provided in the CLA C compiler documentation.

Use the Embedded Coder supported features to restrict the generated code in the limited syntax supported by the CLA C compiler.

Troubleshooting

- Ensure that the Code Generation Tools version your are using supports the CLA C Compiler.
- Ensure that you have installed the latest C/C++ header files with CLA support on your system.

See Also

C28x CLA Task | CLA Subsystem

Related Examples

- “Using the Control Law Accelerator (CLA)” on page 4-270

Code Execution Profiling on Texas Instruments C2000

Sample times you specify in a Simulink model determine the time schedule for running generated code on target hardware. If the target hardware has sufficient computing power, the code runs according to specified sample times in real-time. With real-time execution profiling, you can check if the generated code meets your real-time performance requirements.

You can also use code execution profiling results to enhance the design of your system. For example, if the code easily meets the real-time requirements, you can consider adding more functionality to your system to exploit available processing power. If the code does not meet real-time requirements, you can look for ways to reduce execution time. For example, you can identify tasks that require the most time and then investigate whether trade-off between functionality and speed is possible.

This topic introduces a workflow for real-time code execution profiling by showing you how to:

- Configure the model for code execution profiling, and generate code.
- Run generated code on target hardware.
- Analyze performance through code execution profiling plots and reports.

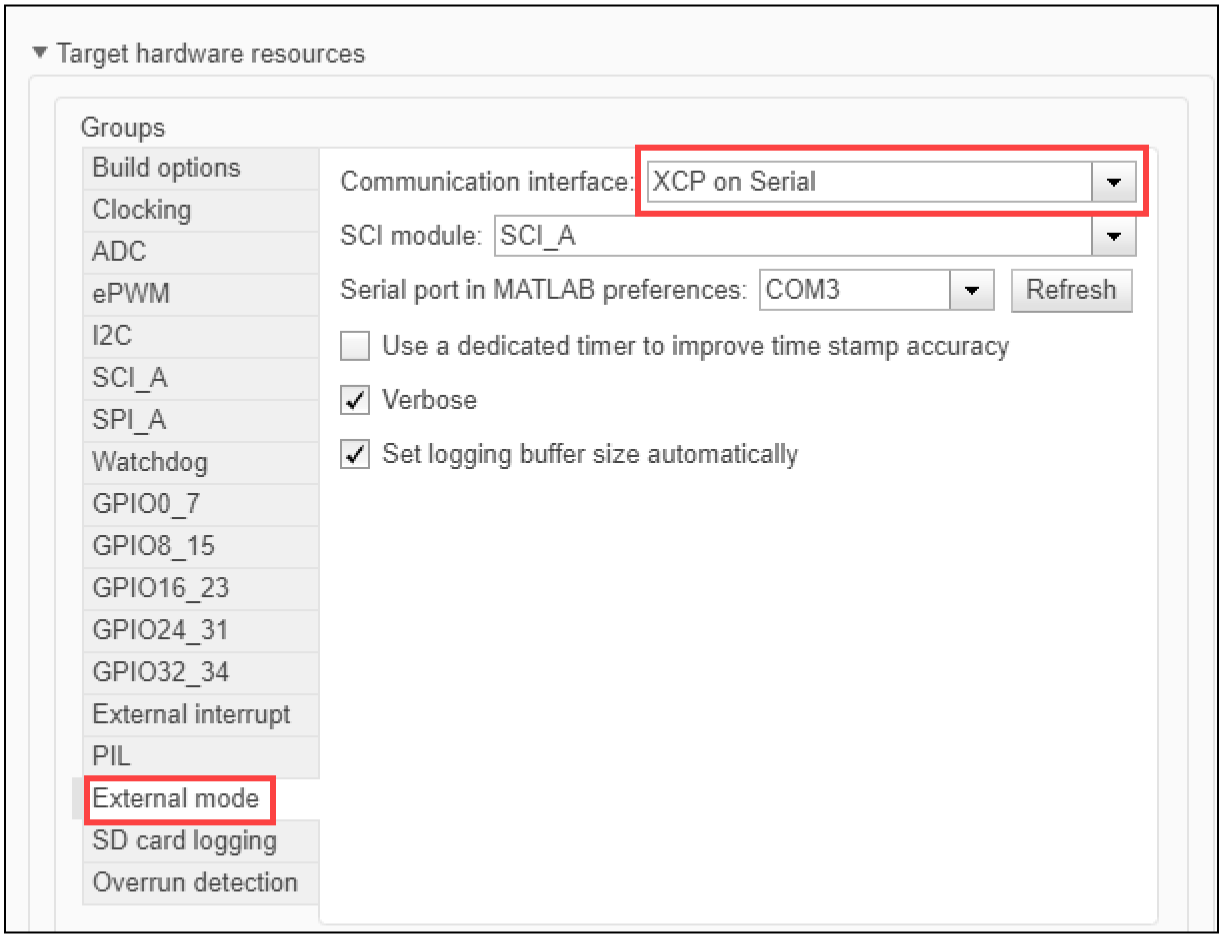
Profiling in XCP External Mode

You can obtain real time profiling data by using the XCP External Mode infrastructure. To configure a Simulink model for real-time profiling perform these steps:

1. In the Simulink Editor, select **Modeling** > **Model Configuration**. In the **Configuration Parameter** dialog box, click **External mode**.

   Note

   - Real-time profiling is supported only with `XCP on Serial` and `XCP on TCP/IP` external communication modes.
   - Profiling for `XCP on CAN` is now supported only with **Metrics only**.



Communication interface: XCP on Serial

SCI module: SCI_A

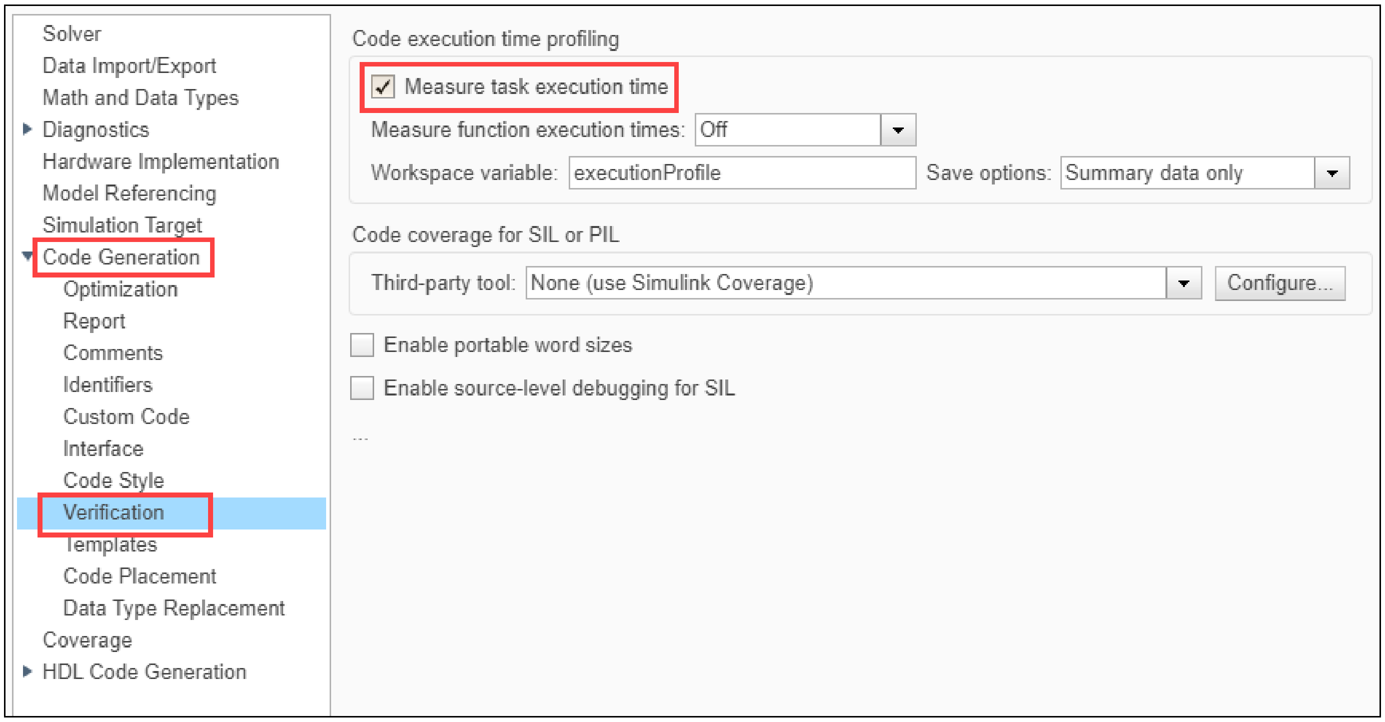
Serial port in MATLAB preferences: COM3

☐ Use a dedicated timer to improve time stamp accuracy

✓ Verbose

✓ Set logging buffer size automatically

2 Navigate to **Code generation** > **Verification** and select **Measure task execution time**.



Code execution time profiling

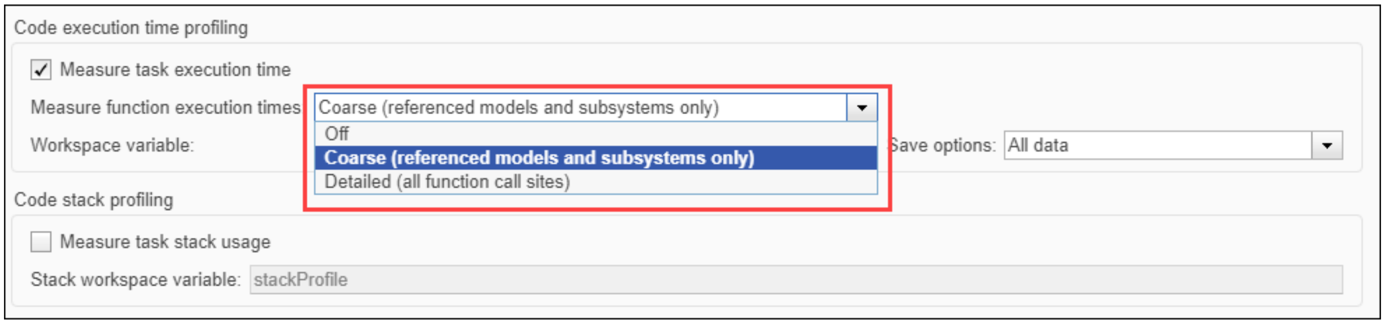
- [✓] Measure task execution time
- Measure function execution times: Off
- Workspace variable: executionProfile
- Save options: Summary data only

Code coverage for SIL or PIL

- Third-party tool: None (use Simulink Coverage)

- [ ] Enable portable word sizes
- [ ] Enable source-level debugging for SIL

3 Select the required option for **Measure function execution time**.


Code execution time profiling

- [✓] Measure task execution time
- Measure function execution times: Coarse (referenced models and subsystems only)
- Workspace variable:
- Save options: All data

Off
Coarse (referenced models and subsystems only)
Detailed (all function call sites)

Code stack profiling

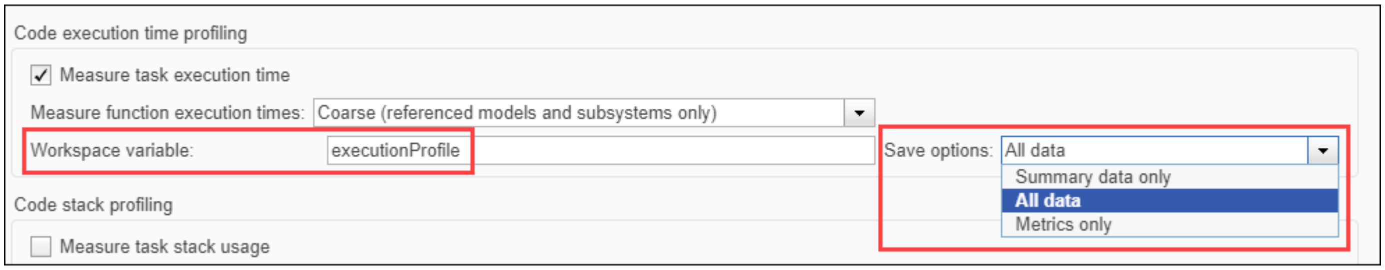
- [ ] Measure task stack usage
- Stack workspace variable: stackProfile

- `Off` – Select this option to disable function Profiling. Only Task profiling is available in this option.
- `Coarse (referenced models and subsystems only)` – Select this option to analyse generated function code for the main model components.
- `Detailed (all function call sites)` – Select this option to analyse generated function code for all blocks in the model

Note Selecting `Detailed (all function call sites)` option introduces a lot of overhead due to the profiling functions.

4 Enter the required value for **Workspace variable**. It is the variable in the MATLAB workspace used for storing data received from the target.

5 Select the required option for **Save options**. For help on selecting the save options, see "Save Options" on page 1-110.


Code execution time profiling

☑ Measure task execution time

Measure function execution times: Coarse (referenced models and subsystems only)

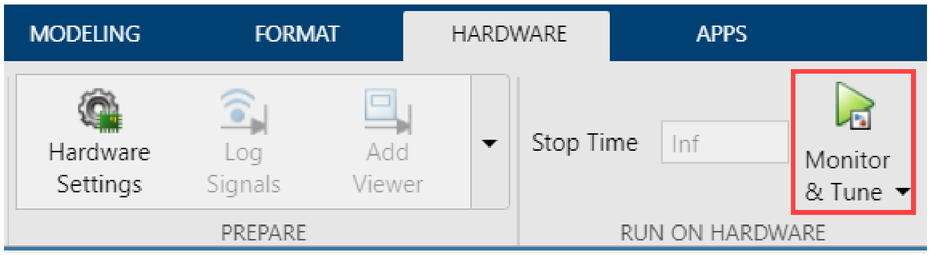
Workspace variable: executionProfile

Save options: All data

Code stack profiling

☐ Measure task stack usage

6 Click **Monitor & Tune** from the **Hardware** tab of Simulink toolstrip to generate the profiling report.



After the simulation ends, a profiling report is generated with profiling metrics of different tasks/functions that are being profiled. For more information, see “Code Execution Profiling on Texas Instruments C2000 Targets in XCP External Mode” on page 4-337.

For information on code execution profiling with SIL and PIL, see Code Execution Profiling with SIL and PIL.

Save Options

Select a save option based on the type of report you want to generate. This table explains the differences between the available save options.

	Summary data only	**All data**	**Metrics only**
Real-time data	Available	Available	Not available. Target sends profiling data only at the end of the simulation.
Host memory requirement	This option requires less memory as the host stores only the summary metrics of the profiling data. For example, 11 KB data for a model running for 50 seconds.	This option requires large memory as the host stores all the data sent by the target. For example, 1500 KB data for a model running for 50 seconds.	This option requires less memory as the host stores only the metrics data sent by the target. For example, 12 KB of data for a model running for 50 seconds.
SDI streaming	Available	Available	Not available
Bandwidth requirement	Requires additional bandwidth.	Requires additional bandwidth.	Does not require additional bandwidth.

Selecting Save Options

This section helps you to select the recommended save options in different scenarios.

- `All data` – Select this option, if the host has enough memory and the target has the required bandwidth to stream data.
- `Summary data only` – Select this option, if the simulation is running for a long time and host does not have a lot of memory.
- `Metrics only` – Select this option, if the target does not have enough time/bandwidth to stream profiling data.

Limitations

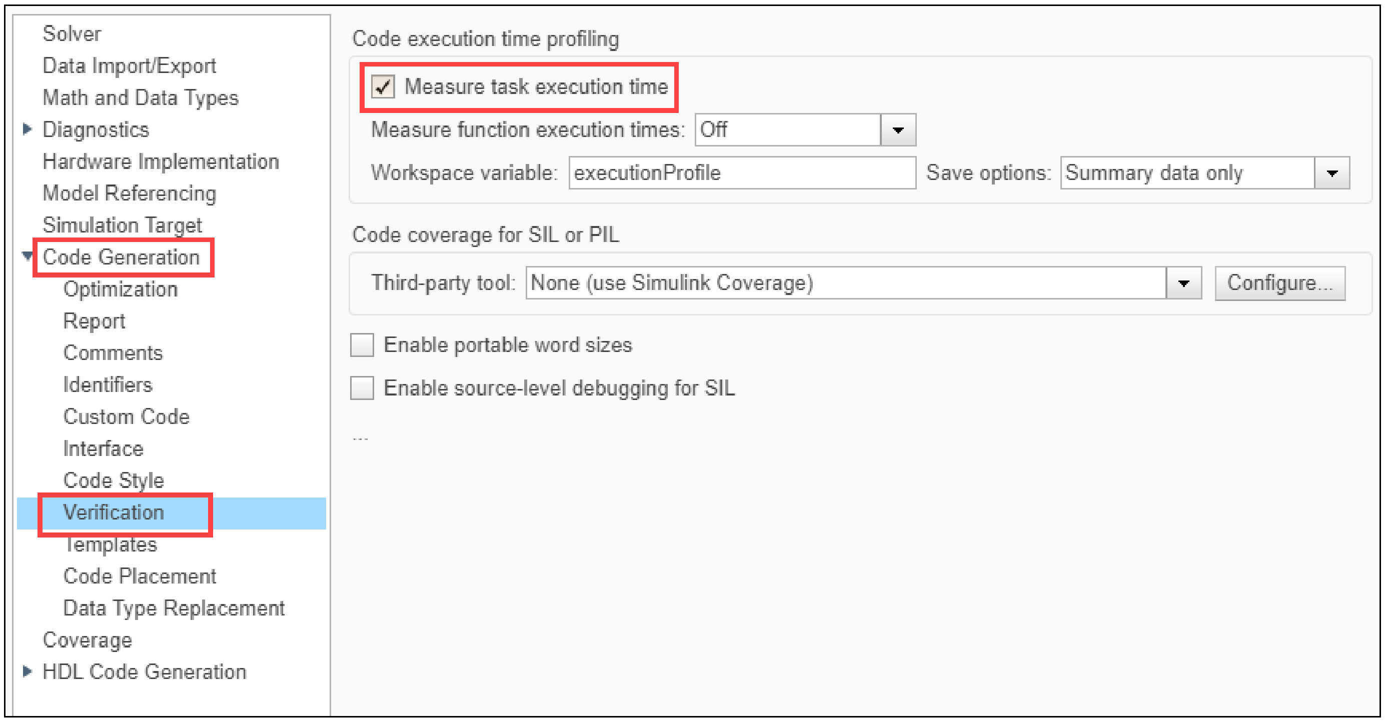
- XCP stack takes up high amount of RAM and flashes on the target. This issue might occur if the target has a less amount of memory (F2802x) and a complex model is being simulated. In this case, use “Profiling with Build” on page 1-111 as it has lighter memory footprint
- You need a serial connection to establish an external mode connection. If Serial connection is not available on the boards, then use FTDI chips.

Profiling with Build

You can use the Profiling with Build feature when the target hardware does not have enough memory to run a Simulink model for real-time profiling in XCP External Mode

This section helps to get profiling data from the target hardware. Perform the following steps.

1 In the **Configuration Parameter** dialog box, navigate to **Code generation** > **Verification** and select **Measure task execution time**.



Code execution time profiling

[✓] Measure task execution time

Measure function execution times: Off

Workspace variable: executionProfile   Save options: Summary data only

Code coverage for SIL or PIL

Third-party tool: None (use Simulink Coverage)

[ ] Enable portable word sizes

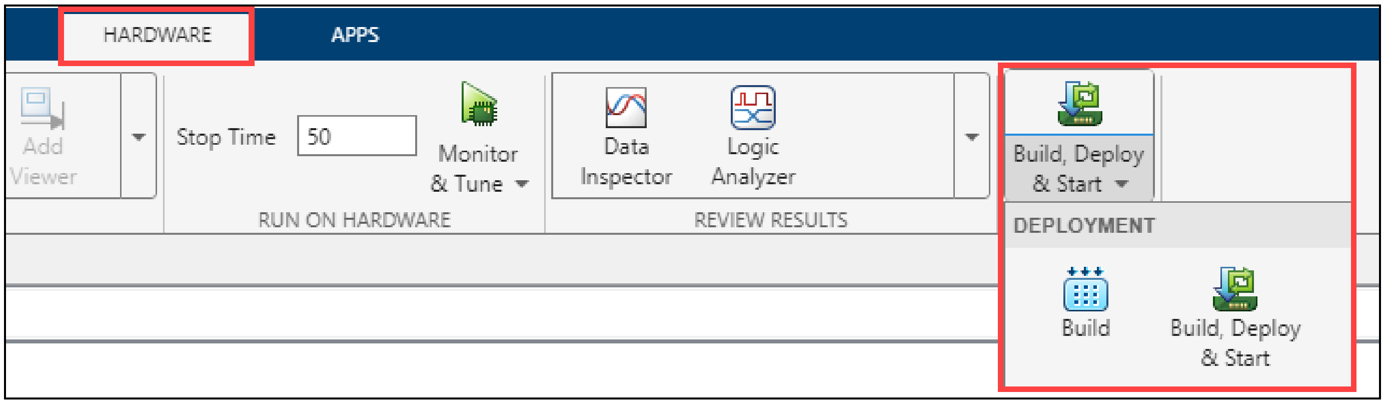
[ ] Enable source-level debugging for SIL

2 Set the number of profiling samples to be collected on the target using this command.

```
codertarget.tic2000.setExecutionProfileBufferLength(<modelName>, 100)
```

In the above command, name of the model is the first argument and number of profiling samples is the second argument.

3 Click **Build** or **Build, Deploy & Start** from the **Hardware** tab of Simulink toolstrip.



4 Get profiling data from the target by executing the following command.

```
codertarget.profile.getData(<modelName>)
```

5 Execute the following code in MATLAB Command Window to obtain the **Profiling Timeline** for the session you just ran. Analyze the **execution timeline** of different tasks and functions. Notice where the faster task pre-empts the slower one and where different functions start and end. Close the timeline when you are done.

```
executionProfile.timeline
```

For information on using C2000 Microcontroller Blockset for real-time execution profiling of generated code, see “Real-Time Code Execution Profiling” on page 4-157.

Limitations

- Real-time streaming of profiling data and SDI visualization are not supported.
- Number of profiling samples to be collected on the targets must be set manually. Also, the MATLAB command must be run manually to get the data from the target.
- Memory available on the target will limit the number of profiling samples that can be collected on the target.

Troubleshooting

Data Drop in Signal Logging or Code Execution Profiling

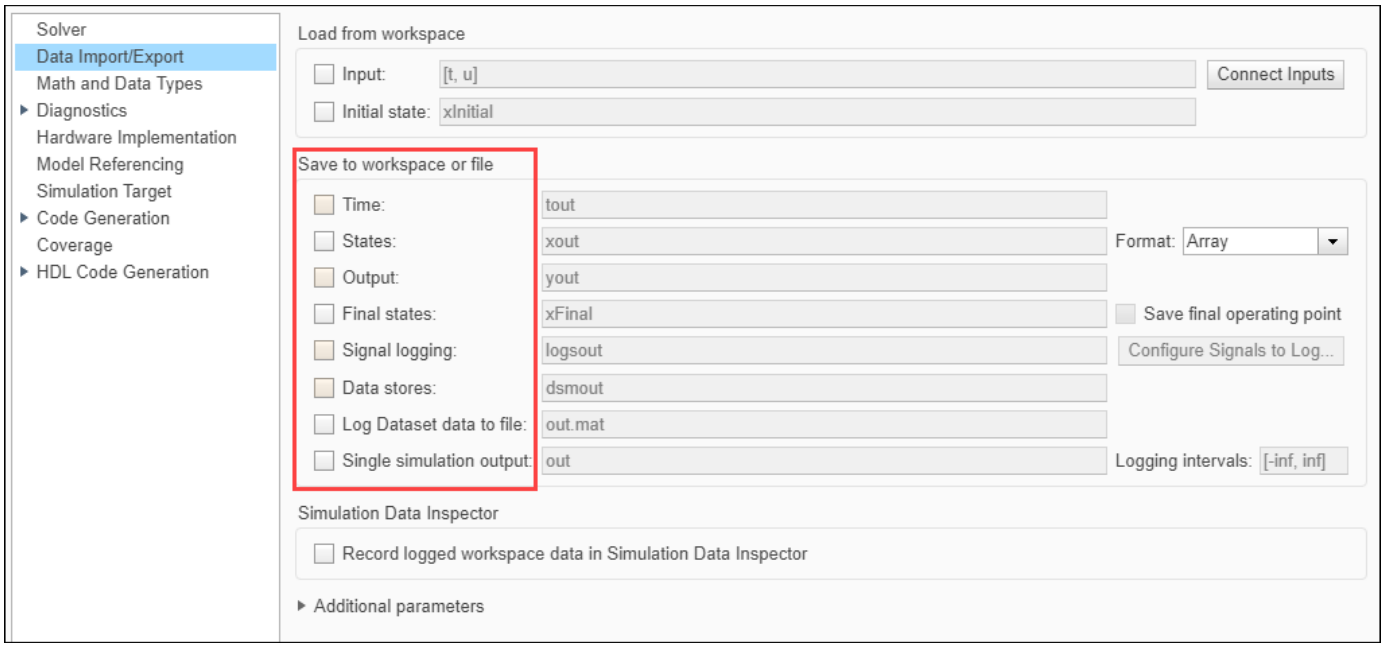
Description

Data drops can occur either in signal logging or profiling.

Action

Both Signal Logging and Profiling data streaming use the same communication channel to send data from the target. As channel bandwidth is limited, there could be data drops at high sample rates. This issue can be mitigated by performing these steps.

1 Stream only the data you need. If only profiling data is required, disable signal logging by clearing all the check boxes in **Configuration Parameters > Data Import/Export > Save to workspace or file**.



Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation

Load from workspace

☐ Input: [t, u] Connect Inputs
☐ Initial state: xInitial

Save to workspace or file

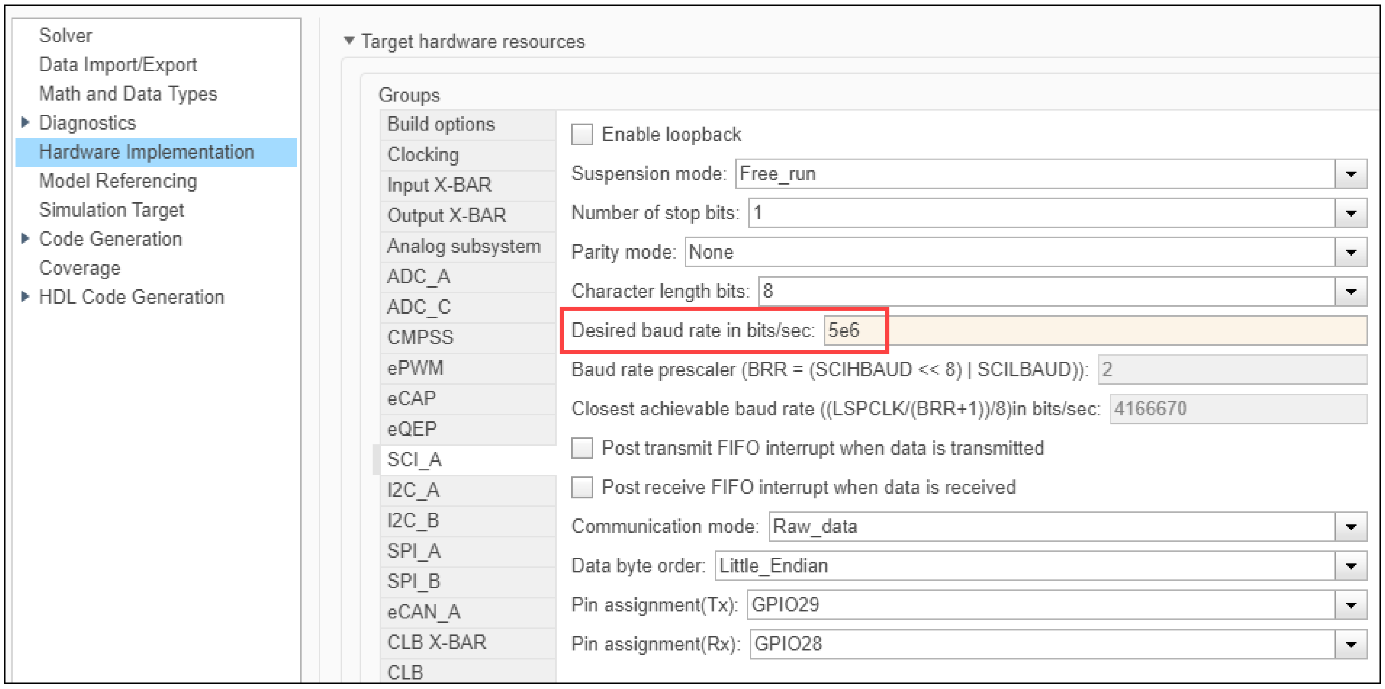
☐ Time: tout
☐ States: xout Format: Array
☐ Output: yout
☐ Final states: xFinal ☐ Save final operating point
☐ Signal logging: logsout Configure Signals to Log...
☐ Data stores: dsmout
☐ Log Dataset data to file: out.mat
☐ Single simulation output: out Logging intervals: [-inf, inf]

Simulation Data Inspector

☐ Record logged workspace data in Simulation Data Inspector

Additional parameters

2 Increase the desired baud rate from SCI_x tab of **Target hardware resources** to get additional bandwidth to stream data at higher sample rates.



Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation

Target hardware resources

Groups
Build options
Clocking
Input X-BAR
Output X-BAR
Analog subsystem
ADC_A
ADC_C
CMPSS
ePWM
eCAP
eQEP
SCI_A
I2C_A
I2C_B
SPI_A
SPI_B
eCAN_A
CLB X-BAR
CLB

☐ Enable loopback
Suspension mode: Free_run
Number of stop bits: 1
Parity mode: None
Character length bits: 8
Desired baud rate in bits/sec: 5e6
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)): 2
Closest achievable baud rate ((LSPCLK/(BRR+1))/8)in bits/sec: 4166670
☐ Post transmit FIFO interrupt when data is transmitted
☐ Post receive FIFO interrupt when data is received
Communication mode: Raw_data
Data byte order: Little_Endian
Pin assignment(Tx): GPIO29
Pin assignment(Rx): GPIO28

See Also

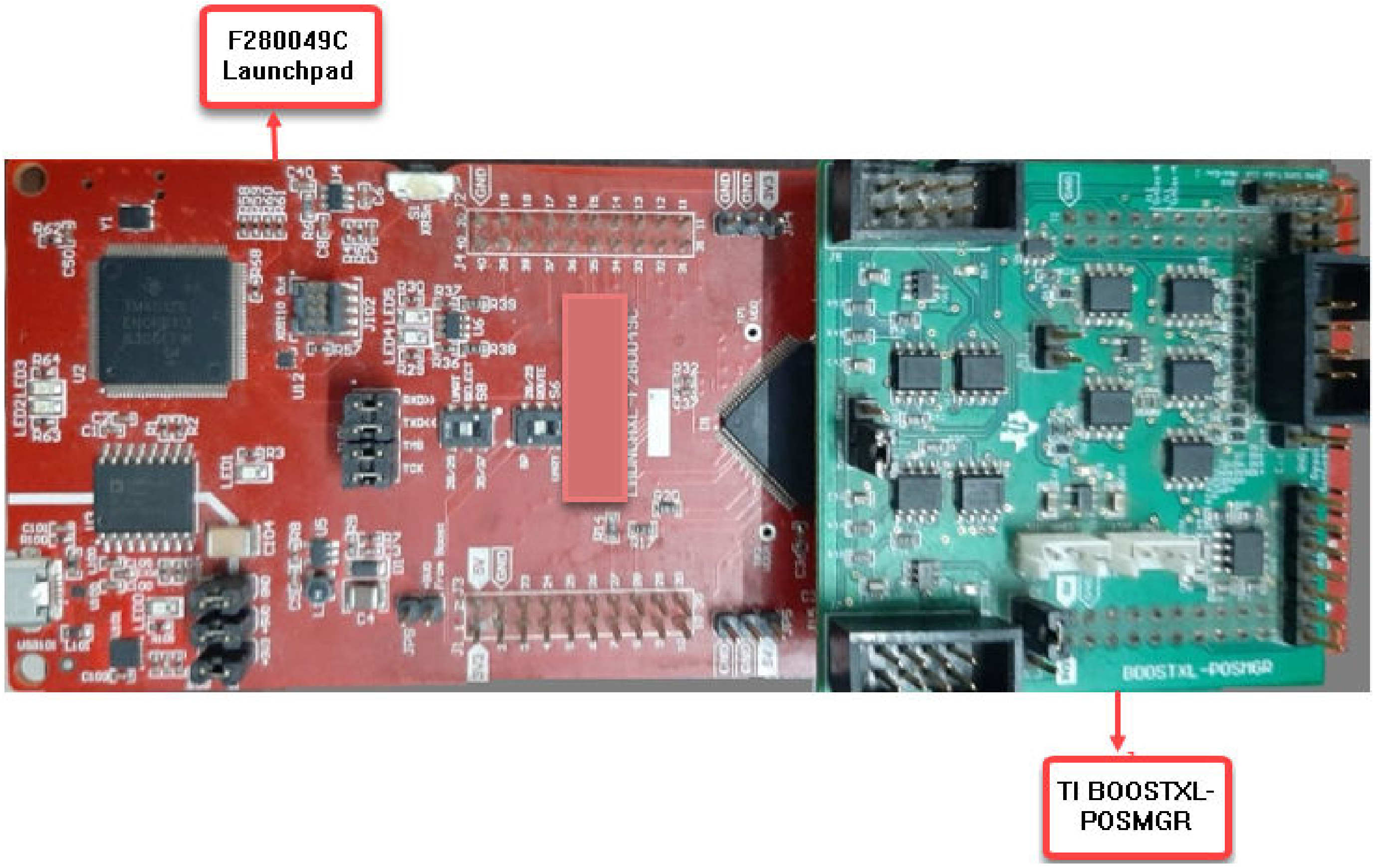
Related Examples

- “Code Execution Profiling on Texas Instruments C2000 Targets in XCP External Mode” on page 4-337

Hardware Connections

Connect F280049C LaunchPad to BOOSTXL-POSMGR

Mount the TI BOOSTXL-POSMGR on top of F280049C LaunchPad as shown in the following figure.



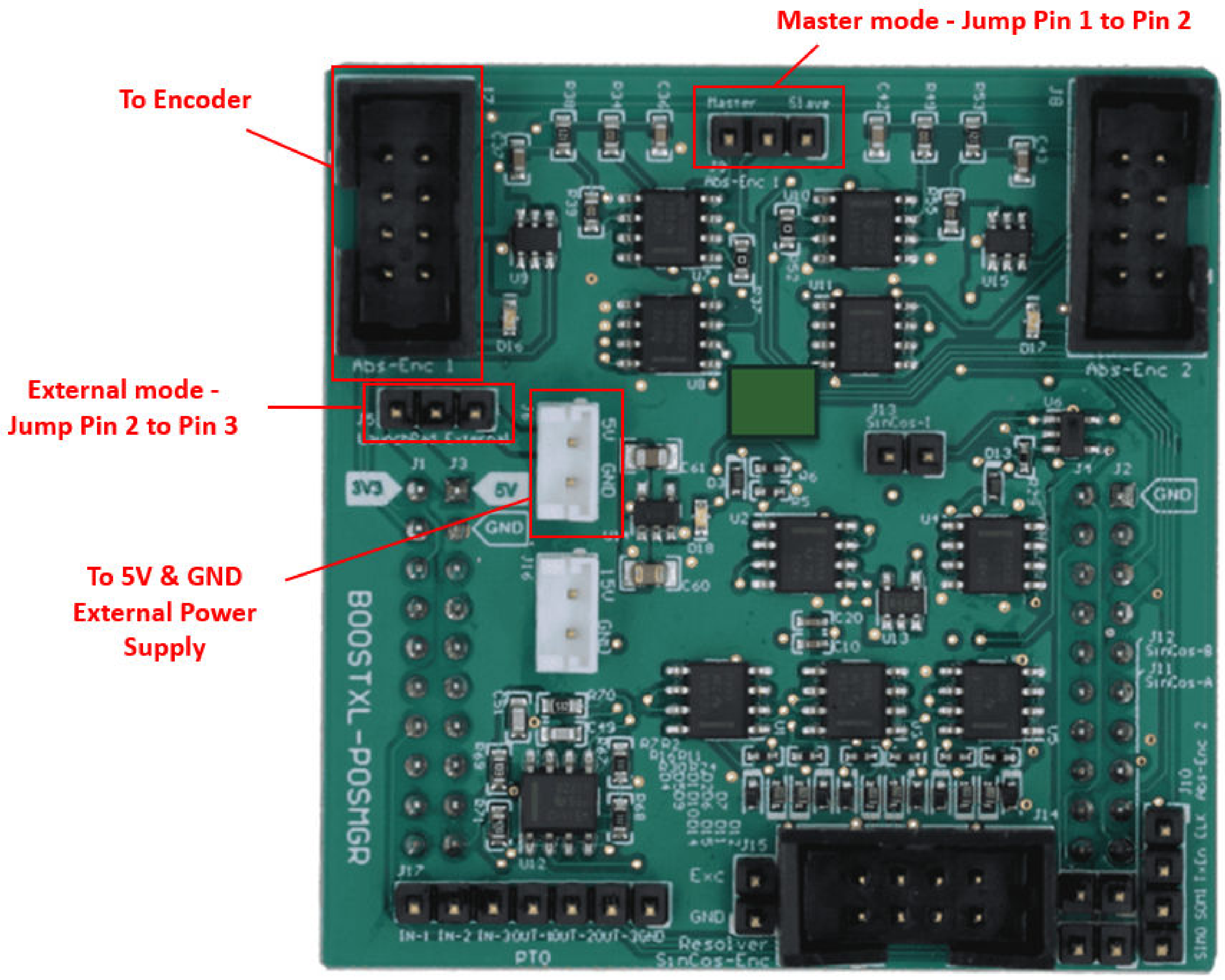
Ensure the following pin map connections are made between the Texas Instruments Piccolo F280049C LaunchPad and Texas Instruments BOOSTXL-POSMGR.

Pin Mapping

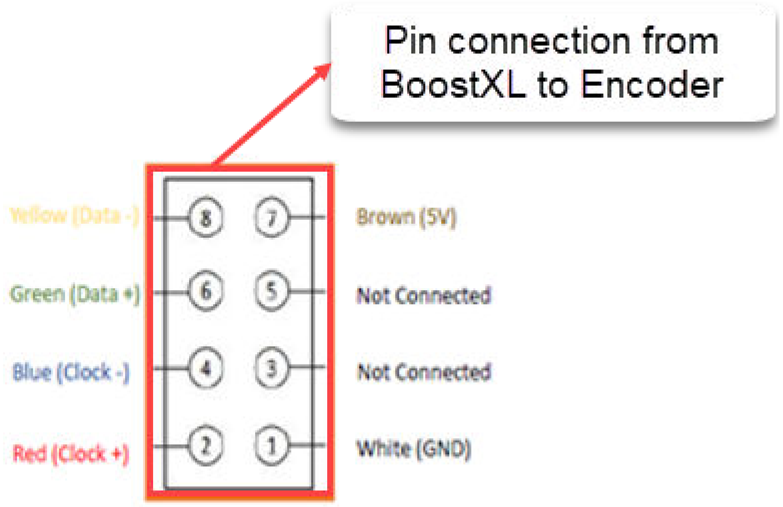
BOOSTXL-POSMGR	Launchpad-F280049c	Description
J4 [pin 2] Enc1-CLK	J8 [pin 80] (ePWM1A)	BiSS-C clock from master to encoder
J4 [pin 4] eQEP1B	J8 [pin 79] (ePWM1B)	SPI clock generated from MCU
J1 [pin 14] SPI-1-CLK	J5 [pin 47] (SPIB_CLK)	One SPI instance to emulate the BISS-C interface (SPIB on Launchpad)
J2 [pin 11] Enc1-DI	J6 [pin 55] (SPIB_MOSI)	
J2 [pin 13] Enc1-DO	J6 [pin 54] (SPIB_MISO)	
J2 [pin 3] SPISTE	J6 [pin 59] (SPIB_STE)	
J1 [pin 6] PWREN1	J5 [pin 43] (GPIO28_BP)	Encoder power enable

Connect BOOSTXL-POSMGR to Absolute Encoder

Connect the Texas Instruments BOOSTXL-POSMGR to Absolute Encoder as detailed in the following figure.


Master mode - Jump Pin 1 to Pin 2
To Encoder
External mode - Jump Pin 2 to Pin 3
To 5V & GND External Power Supply
BOOSTXL-POSMGR
Master
Slave
Abs-Enc 1
Abs-Enc 2
5V
GND
3V3
15V
PTO
Resolver
SinCos-Enc

- **To Encoder** - Use female to female hookup wires to wire the encoder to the header as shown in the following figure. The colors correspond to the encoder wires that are soldered to the connector.



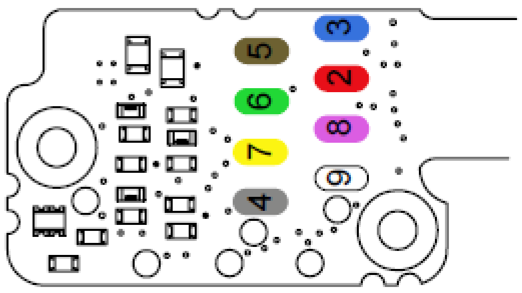
- **5V External Power Supply** - Connect 5V and GND to the external power supply. Use 2 banana clip to mini grabber wires. Use 2 male to female hookup wires to connect from mini grabbers to the external 5V and GND.
- **Master mode** - Use female to female hookup wire to jump pin 1 to pin 2 which sets the booster pack to master mode.
- **External power mode** - Use female to female hookup wire to jump pin 2 to pin 3 which sets the booster pack to external power mode.

Encoder Connections

Ensure the following wire connections are mapped to the absolute shaft encoder (BiSS).

Encoder Electrical Connection

Pin / Pad	Wire Colour	BiSS
Housing	**Outer shield**	**Encoder/machine case (Earth connection)**
1	Inner shield	0 V (GND)
2	Red	MA+
3	Blue	MA–
4	Grey	-
5	Brown	5 V supply
6	Green	SLO+
7	Yellow	SLO–
8	Pink	-
9	White	0 V (GND)


5
3
6
2
8
7
9
4

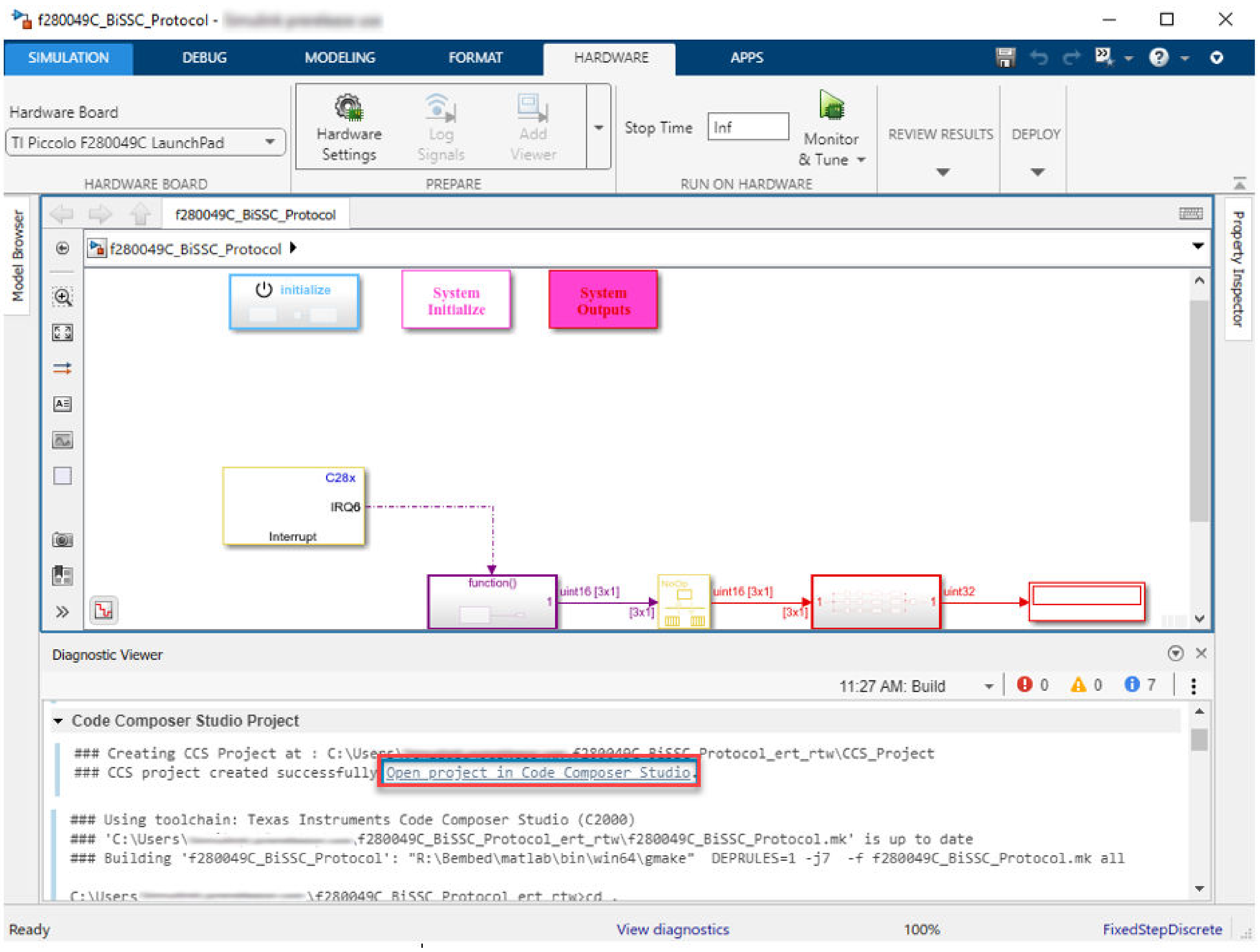
CLB Logic for Clock Generation

In this task you will learn how to:

- **Update** or **Modify** the CLB Tile configurations.
- Generate CLB configuration files.
- Generate CLB BiSS clock generation HTML file.

CLB Tile Configuration

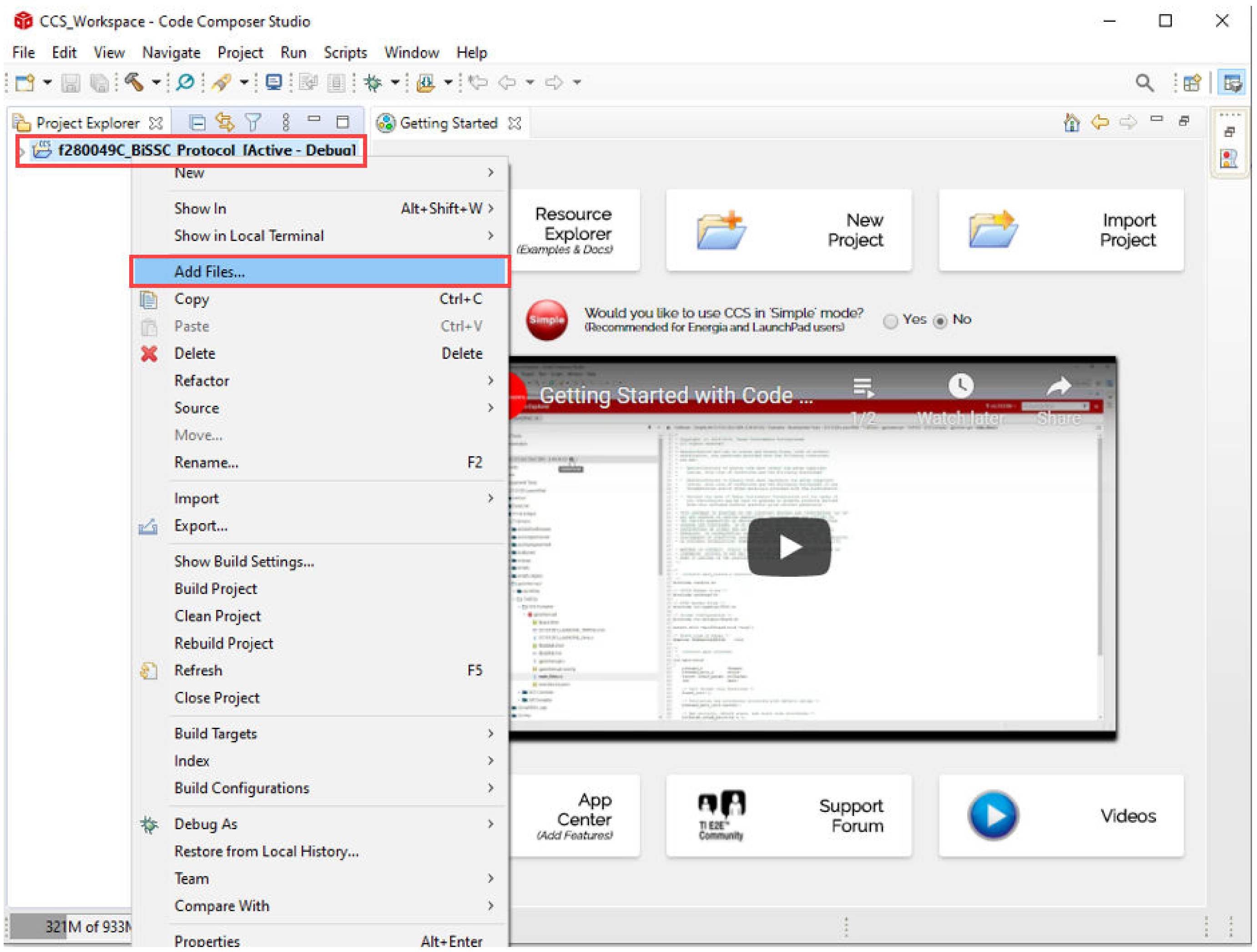
1 To generate the code for the model, press **Ctrl+B** or click **Build, Deploy & Start**. Follow the build process by opening the diagnostic viewer using the link provided at the bottom of the model canvas.



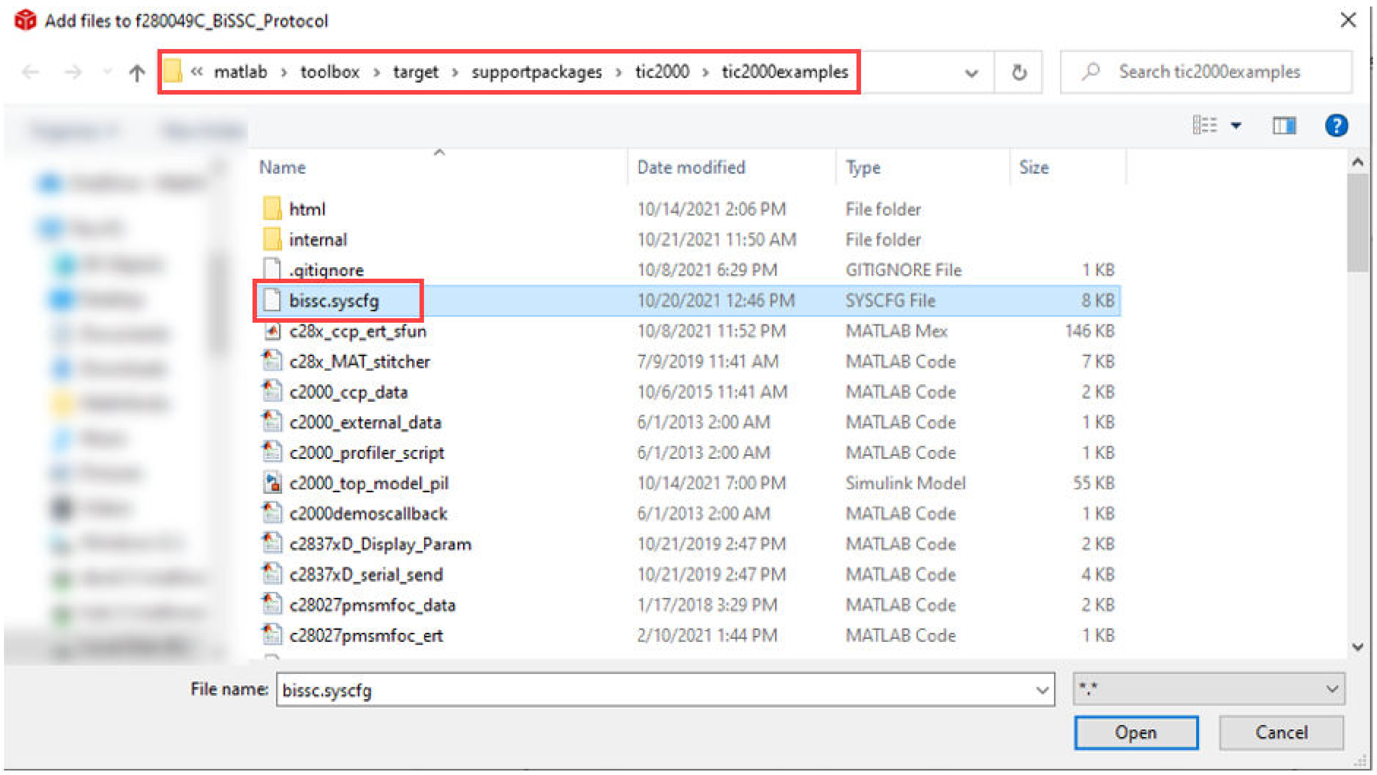
```
### Creating CCS Project at : C:\Users\[redacted]f280049C_BiSSC_Protocol_ert_rtw\CCS_Project
### CCS project created successfully Open project in Code Composer Studio.

### Using toolchain: Texas Instruments Code Composer Studio (C2000)
### 'C:\Users\[redacted]f280049C_BiSSC_Protocol_ert_rtw\f280049C_BiSSC_Protocol.mk' is up to date
### Building 'f280049C_BiSSC_Protocol': "R:\Bembed\matlab\bin\win64\gmake"  DEPRULES=1 -j7  -f f280049C_BiSSC_Protocol.mk all

C:\Users[redacted]\f280049C_BiSSC_Protocol_ert_rtw>cd .
```

2 Click to open the project in CCS.

3 Right click on the project and select **Add files**.



4 Locate the `bissc.sysconfig` in the `\tic2000examples` folder path of your installed blockset.


Add files to f280049C_BiSSC_Protocol

« matlab › toolbox › target › supportpackages › tic2000 › tic2000examples

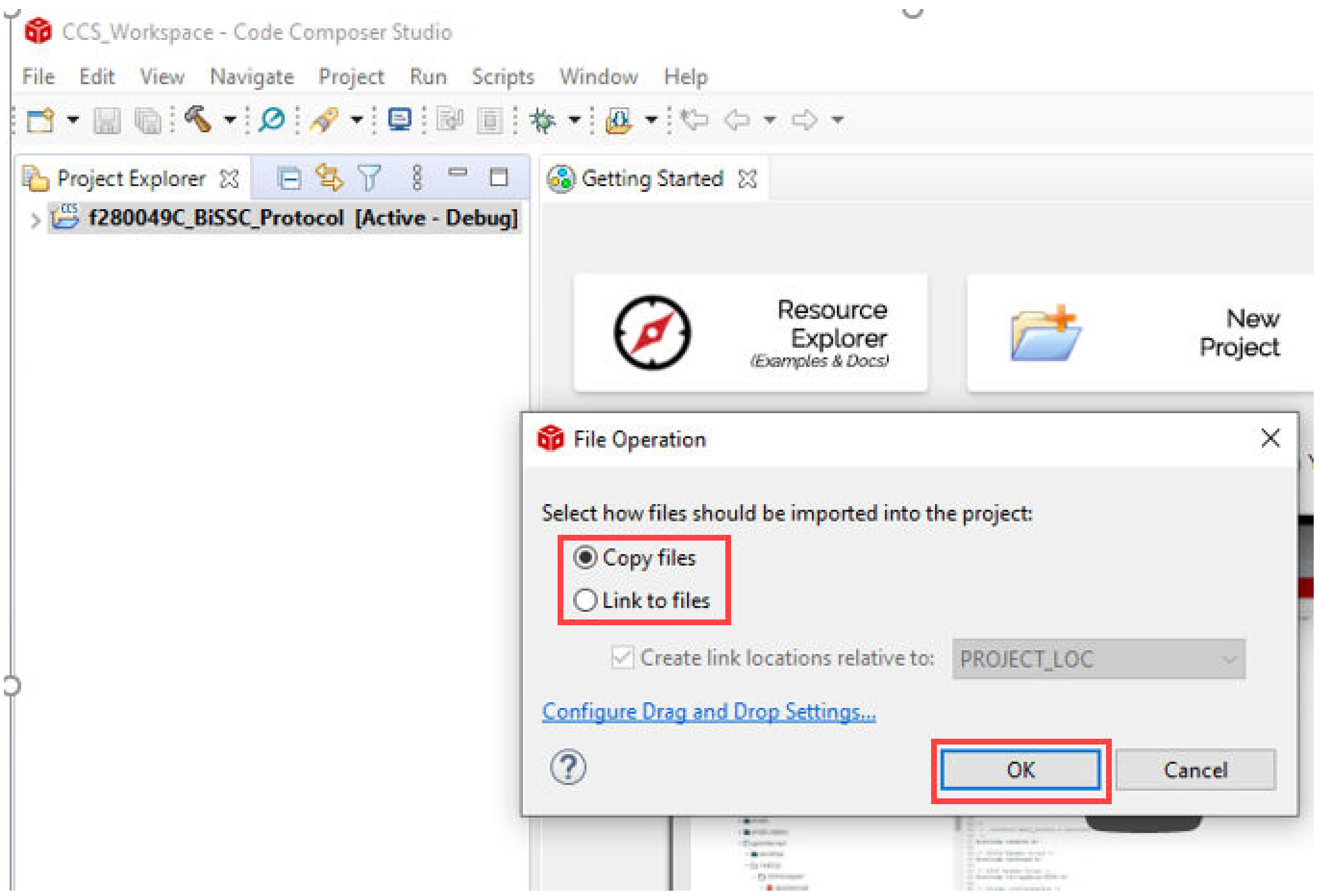
Search tic2000examples

Name	Date modified	Type	Size
html	10/14/2021 2:06 PM	File folder	
internal	10/21/2021 11:50 AM	File folder	
.gitignore	10/8/2021 6:29 PM	GITIGNORE File	1 KB
bissc.syscfg	10/20/2021 12:46 PM	SYSCFG File	8 KB
c28x_ccp_ert_sfun	10/8/2021 11:52 PM	MATLAB Mex	146 KB
c28x_MAT_stitcher	7/9/2019 11:41 AM	MATLAB Code	7 KB
c2000_ccp_data	10/6/2015 11:41 AM	MATLAB Code	2 KB
c2000_external_data	6/1/2013 2:00 AM	MATLAB Code	1 KB
c2000_profiler_script	6/1/2013 2:00 AM	MATLAB Code	1 KB
c2000_top_model_pil	10/14/2021 7:00 PM	Simulink Model	55 KB
c2000demoscallback	6/1/2013 2:00 AM	MATLAB Code	1 KB
c2837xD_Display_Param	10/21/2019 2:47 PM	MATLAB Code	2 KB
c2837xD_serial_send	10/21/2019 2:47 PM	MATLAB Code	4 KB
c28027pmsmfoc_data	1/17/2018 3:29 PM	MATLAB Code	2 KB
c28027pmsmfoc_ert	2/10/2021 1:44 PM	MATLAB Code	1 KB

File name: bissc.syscfg

.

5 Click **Ok** to Copy file.


CCS_Workspace - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Explorer
f280049C_BiSSC_Protocol [Active - Debug]
Getting Started
Resource Explorer (Examples & Docs)
New Project
File Operation
OK
Cancel

Select how files should be imported into the project:

[✓ Copy files] Link to files

Create link locations relative to: PROJECT_LOC

Configure Drag and Drop Settings...

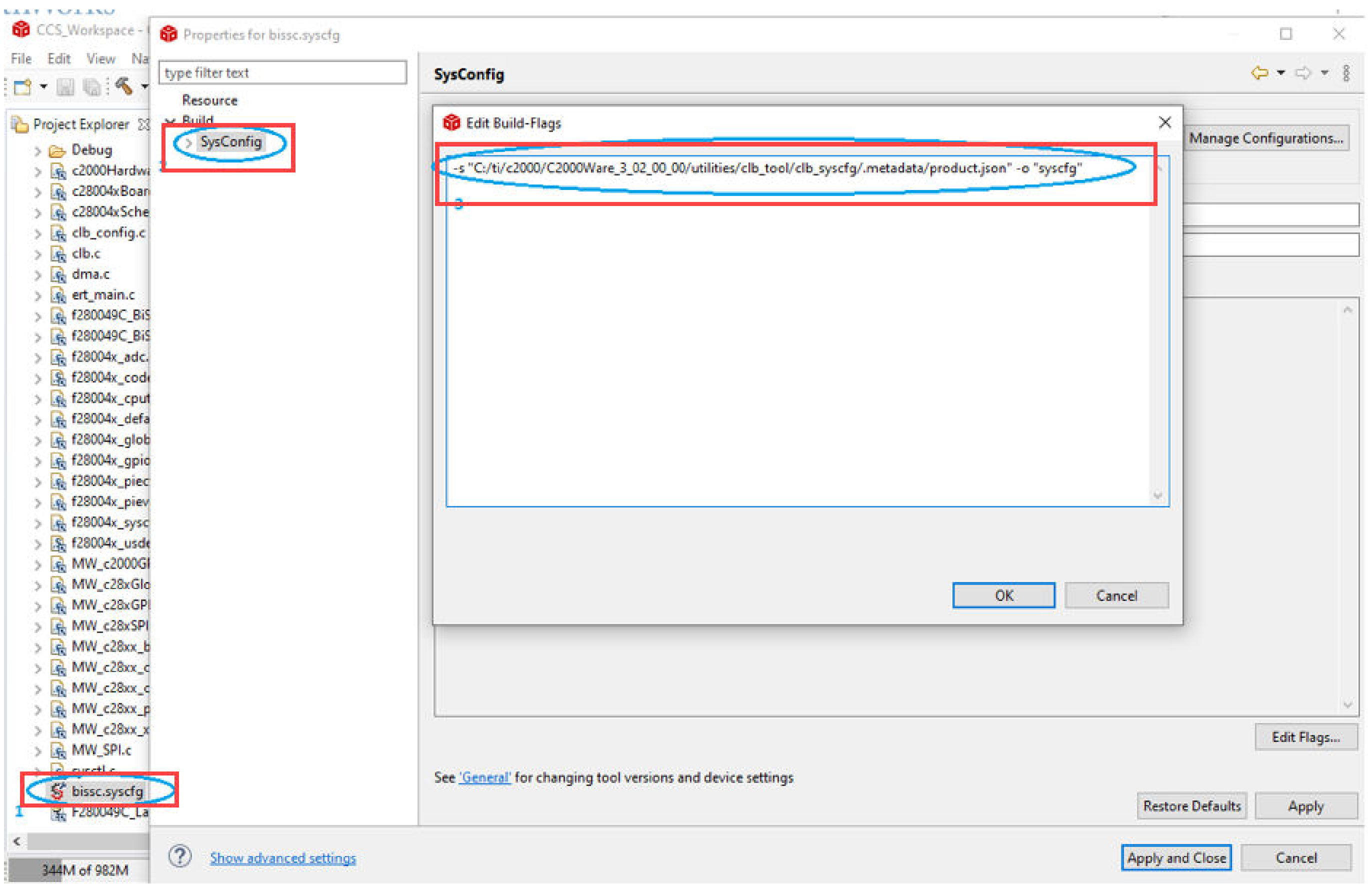
6 Configure the BiSS system configuration file (`bissc.syscfg`).

a Right click on `bissc.sysconfig` and select properties.

b Under **Build**, select **SysConfig**and Click **Edit Flags**.

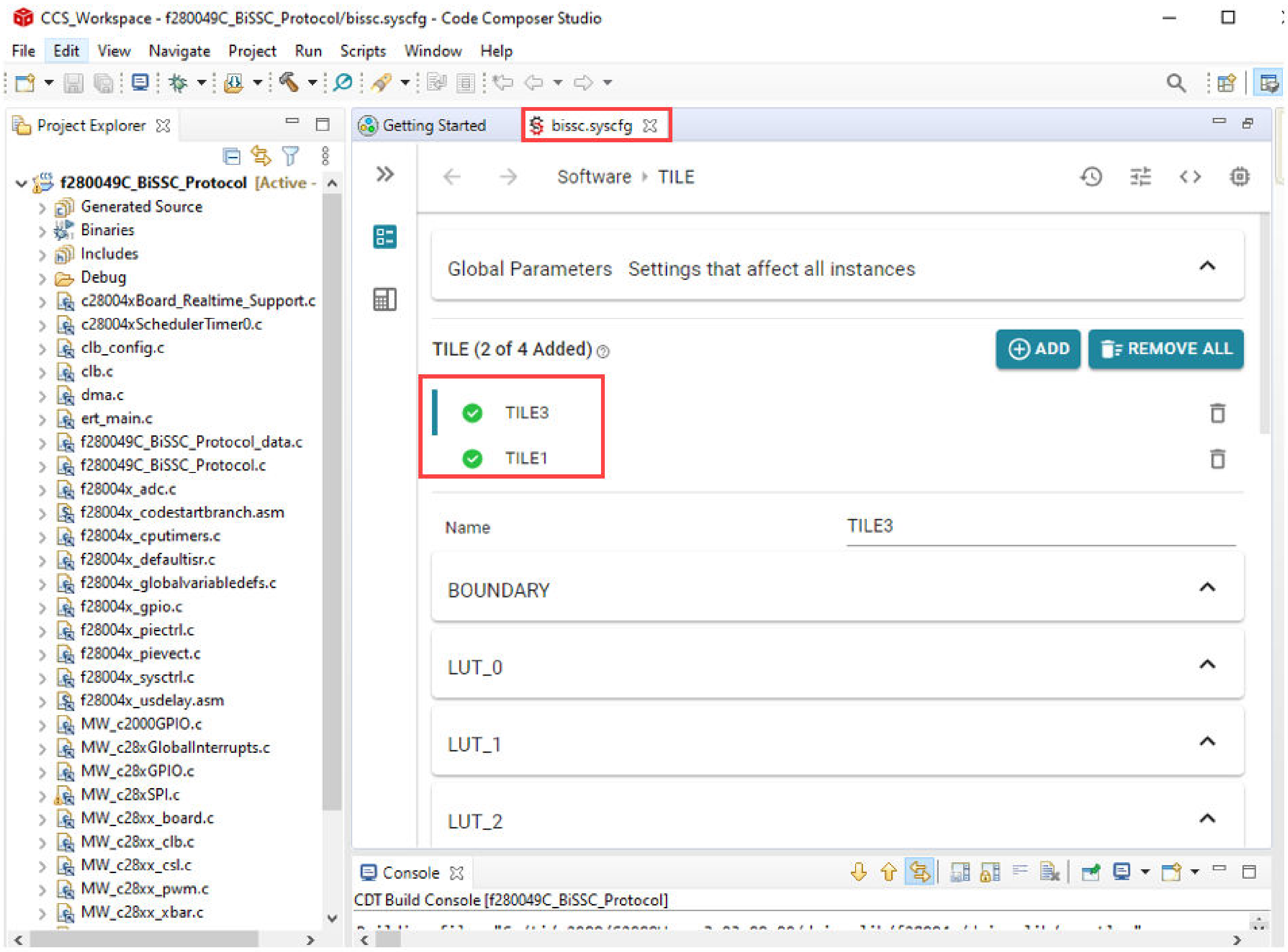
c Update the path to your C2000Ware install as shown, click **Apply** and **Close**.

Note Provide the installation path of the C2000ware depending upon where it is installed in your PC.



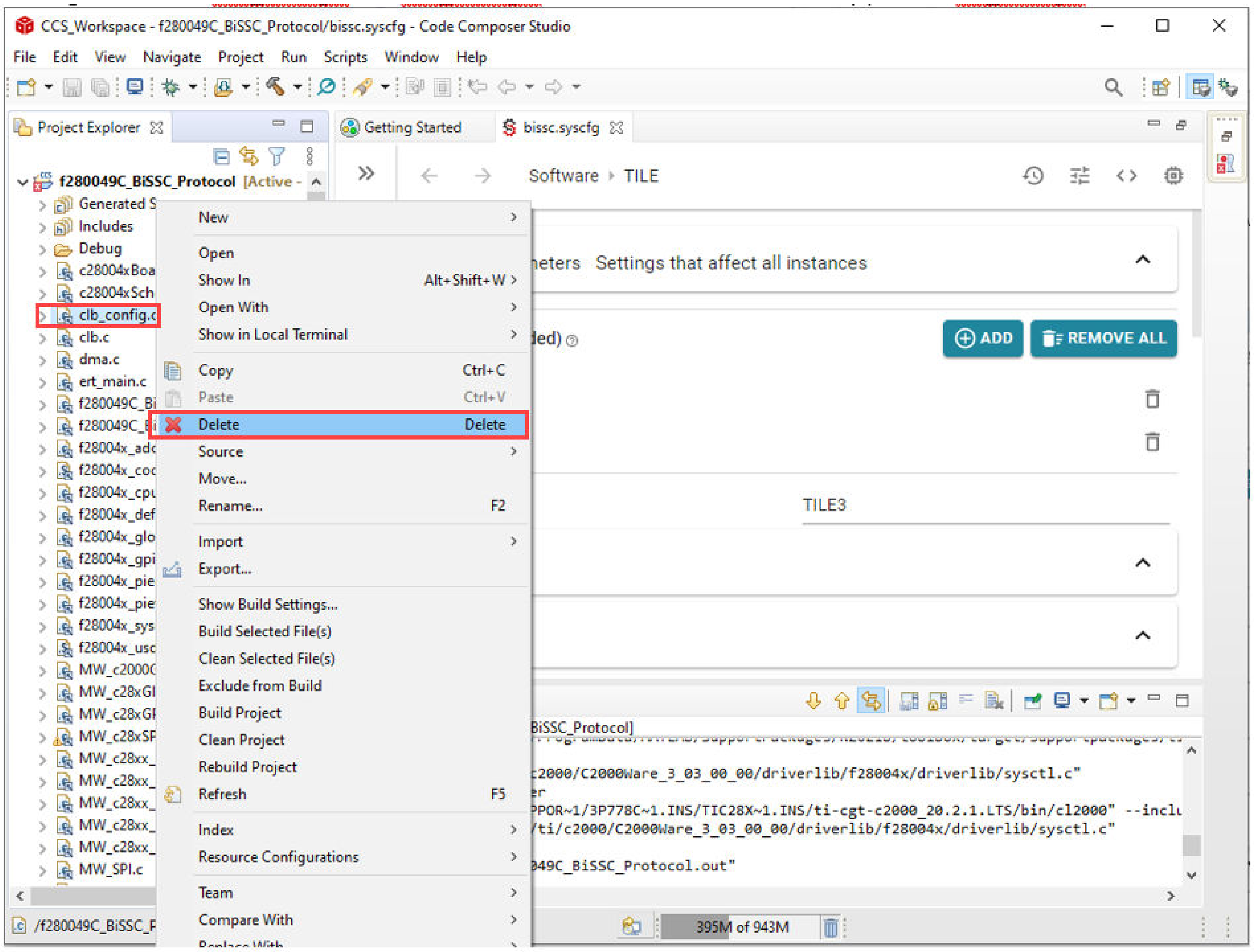
```
-s "C:/ti/c2000/C2000Ware_3_02_00_00/utilities/clb_tool/clb_syscfg/.metadata/product.json" -o "syscfg"
```

7 Double click on `bissc.syscfg` to view the TILE configurations.

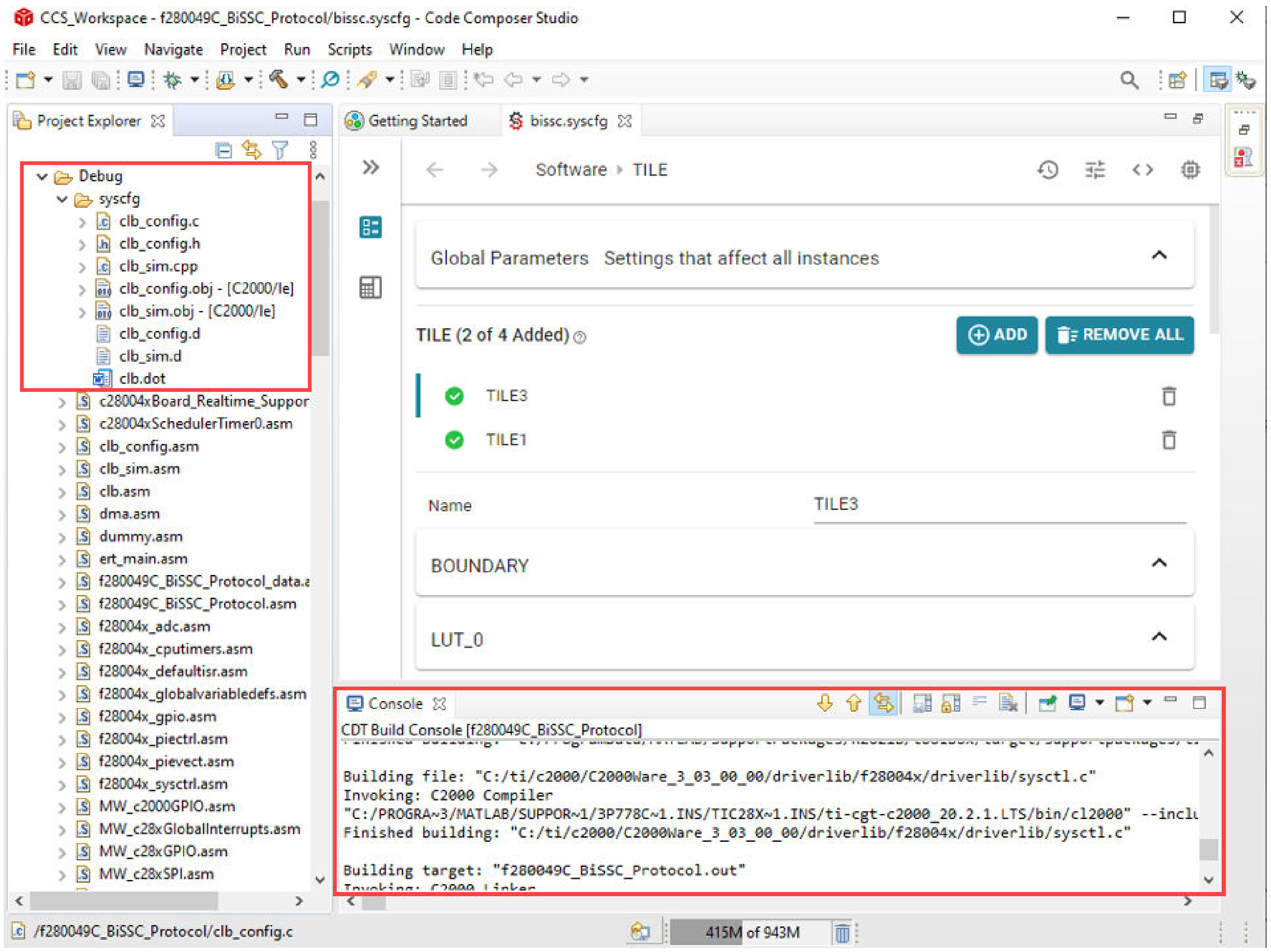


Generate CLB Configuration File

1 To re-generate the **CLB_config.c** and **CLB_config.h** file, First delete the existing **clb_config.c** as shown below.



2 Build the project. After successful build, the new **CLB_config.c** and **CLB_config.h** are generated as shown below. You can choose to debug from CCS or use these files in MATLAB.



```
Building file: "C:/ti/c2000/C2000Ware_3_03_00_00/driverlib/f28004x/driverlib/sysctl.c"
Invoking: C2000 Compiler
"C:/PROGRA~3/MATLAB/SUPPOR~1/3P778C~1.INS/TIC28X~1.INS/ti-cgt-c2000_20.2.1.LTS/bin/cl2000" --inclu
Finished building: "C:/ti/c2000/C2000Ware_3_03_00_00/driverlib/f28004x/driverlib/sysctl.c"

Building target: "f280049C_BiSSC_Protocol.out"
Invoking: C2000 Linker
```

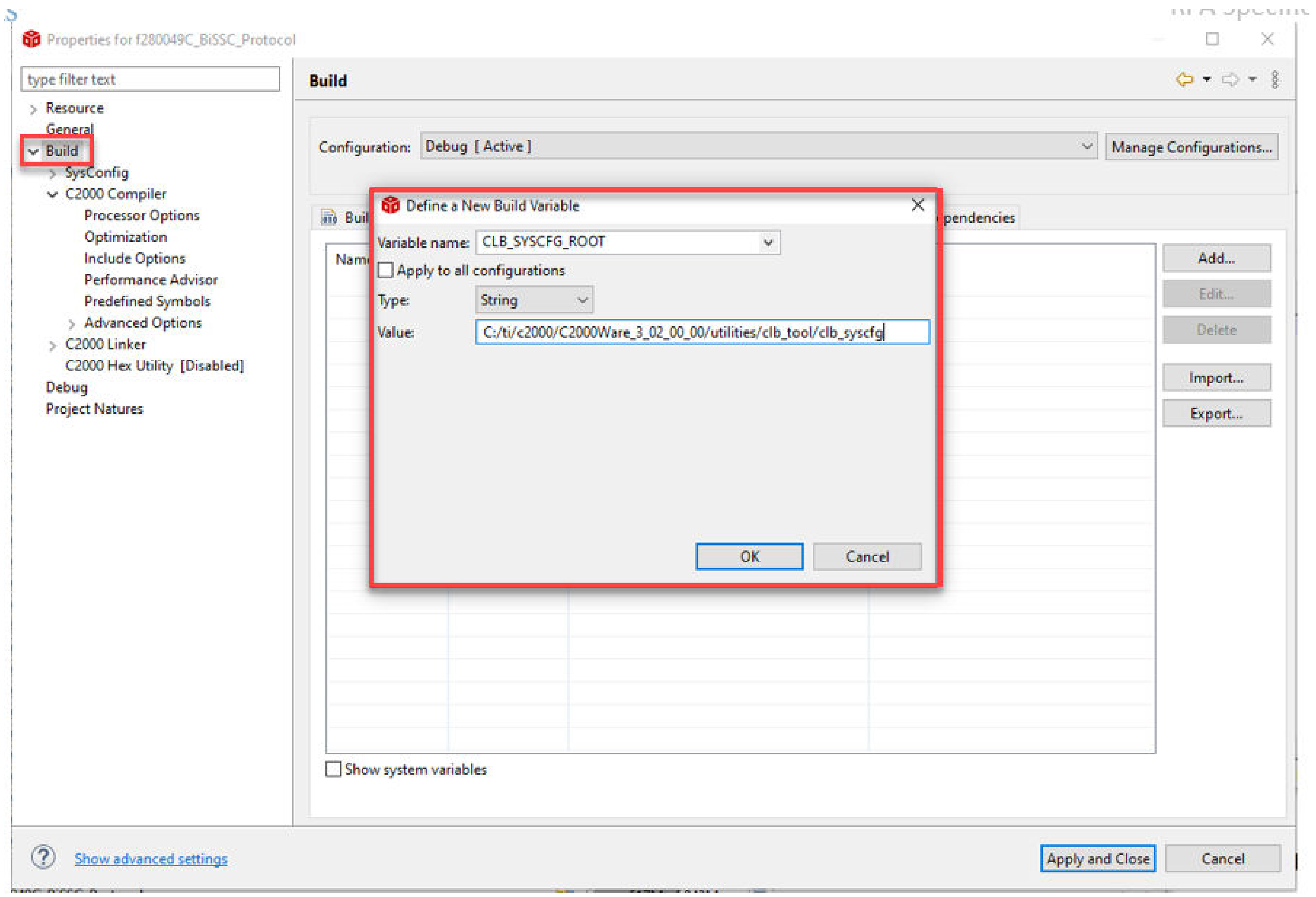
Generate CLB Configuration in HTML

The CLB tool generates CLB configuration HTML format in CCS. It shows the sub-module interconnections in diagram form and can be used to verify the design.

1 Under the project properties, add the following path:

```
C:\ti\c2000\C2000Ware_3_02_00_00\utilities\clb_tool\clb_syscfg\
```

Note Provide the installation path of the C2000ware depending upon where it is installed in your PC.


Properties for f280049C_BiSSC_Protocol

Build

Configuration: Debug [Active]

Define a New Build Variable

Variable name: CLB_SYSCFG_ROOT

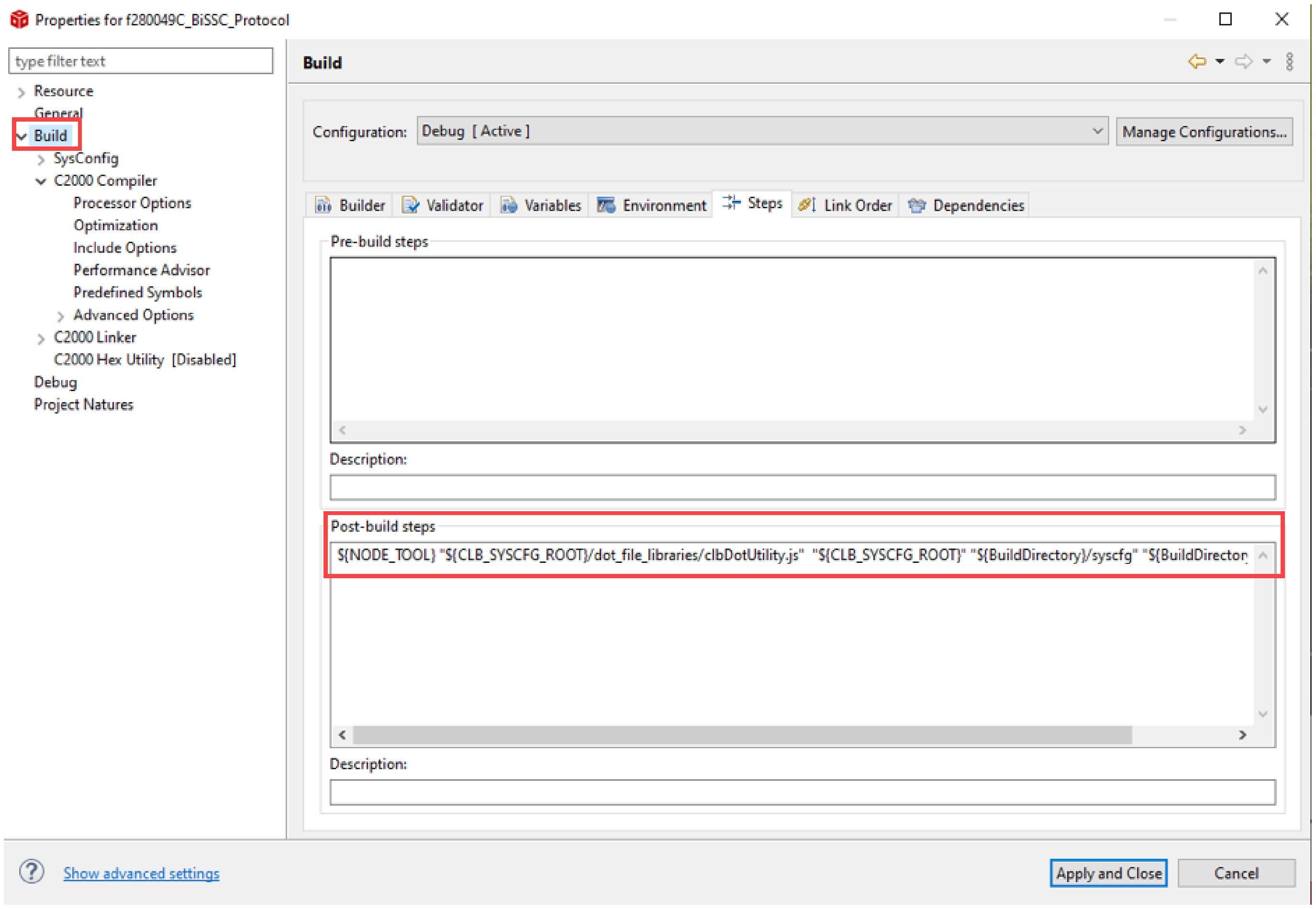
Apply to all configurations

Type: String

Value: C:/ti/c2000/C2000Ware_3_02_00_00/utilities/clb_tool/clb_syscfg

2 Enter the following command under the post build option as shown:

```
${NODE_TOOL} "${CLB_SYSCFG_ROOT}/dot_file_libraries/clbDotUtility.js"  "${CLB_SYSCFG_ROOT}"
"${BuildDirectory}/syscfg" "${BuildDirectory}/syscfg/clb.dot"
```

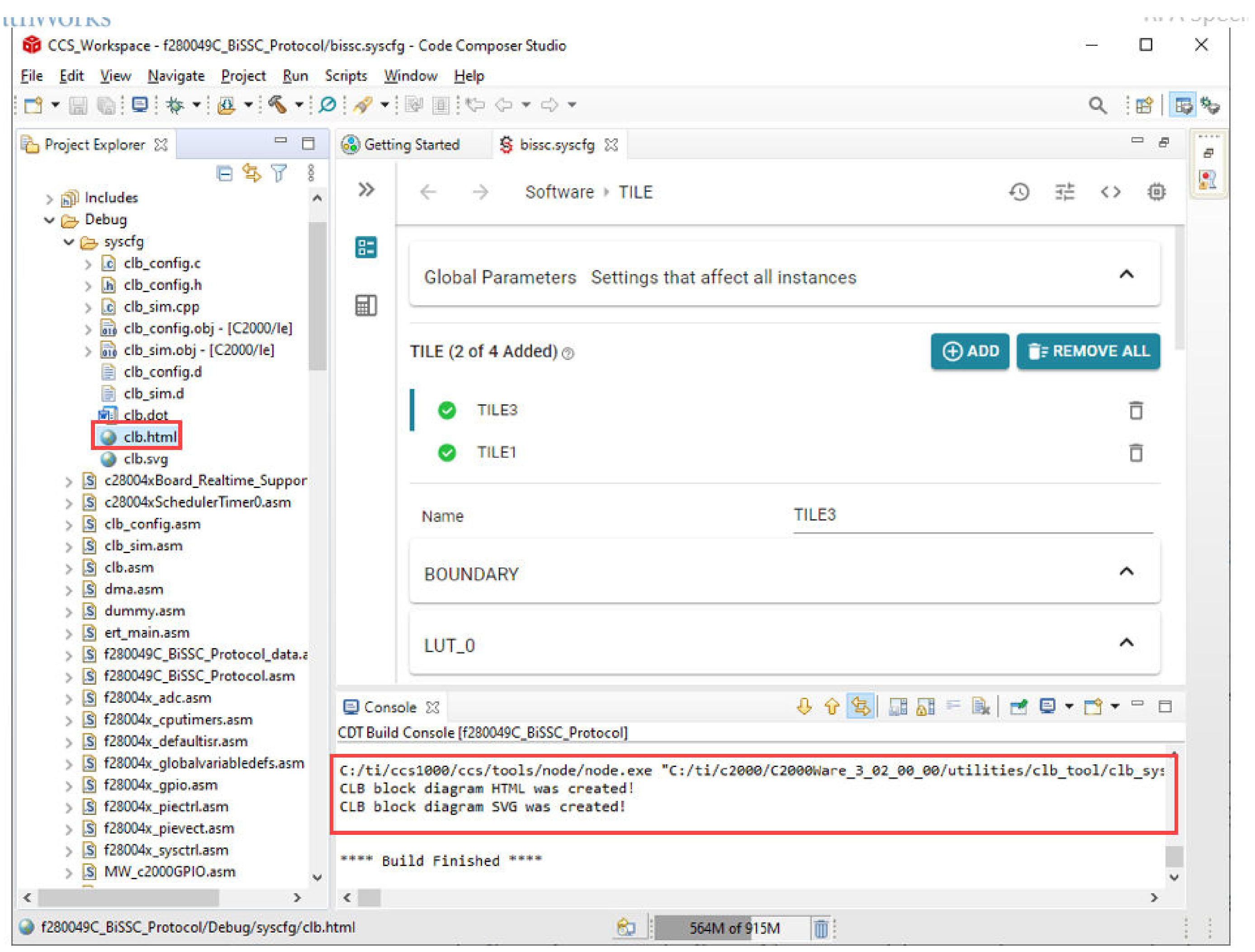


Configuration: Debug [Active]

Post-build steps:

```
${NODE_TOOL} "${CLB_SYSCFG_ROOT}/dot_file_libraries/clbDotUtility.js"  "${CLB_SYSCFG_ROOT}" "${BuildDirectory}/syscfg" "${BuildDirector
```

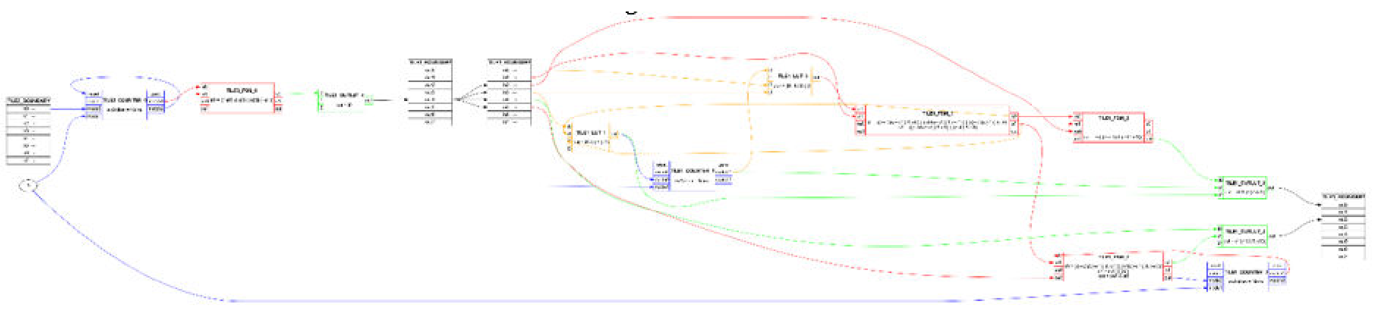
3 Build the project. After successful build the **clb.html** is generated.



Name TILE3

```
C:/ti/ccs1000/ccs/tools/node/node.exe "C:/ti/c2000/C2000Ware_3_02_00_00/utilities/clb_tool/clb_sys
CLB block diagram HTML was created!
CLB block diagram SVG was created!

**** Build Finished ****
```

4 Double click on the **clb.html** to view the CLB tile design.



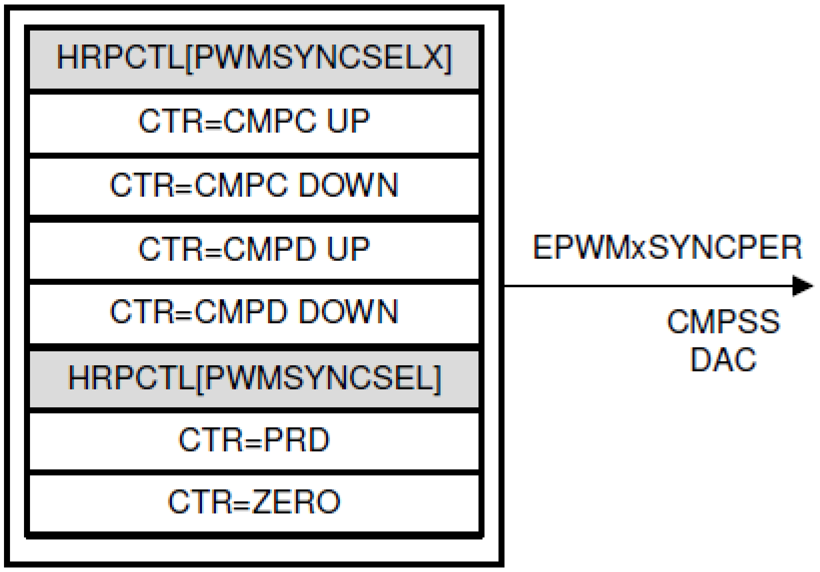
Overview of Time-Base Synchronization in ePWM Type 4

Time-Base Clock Synchronization

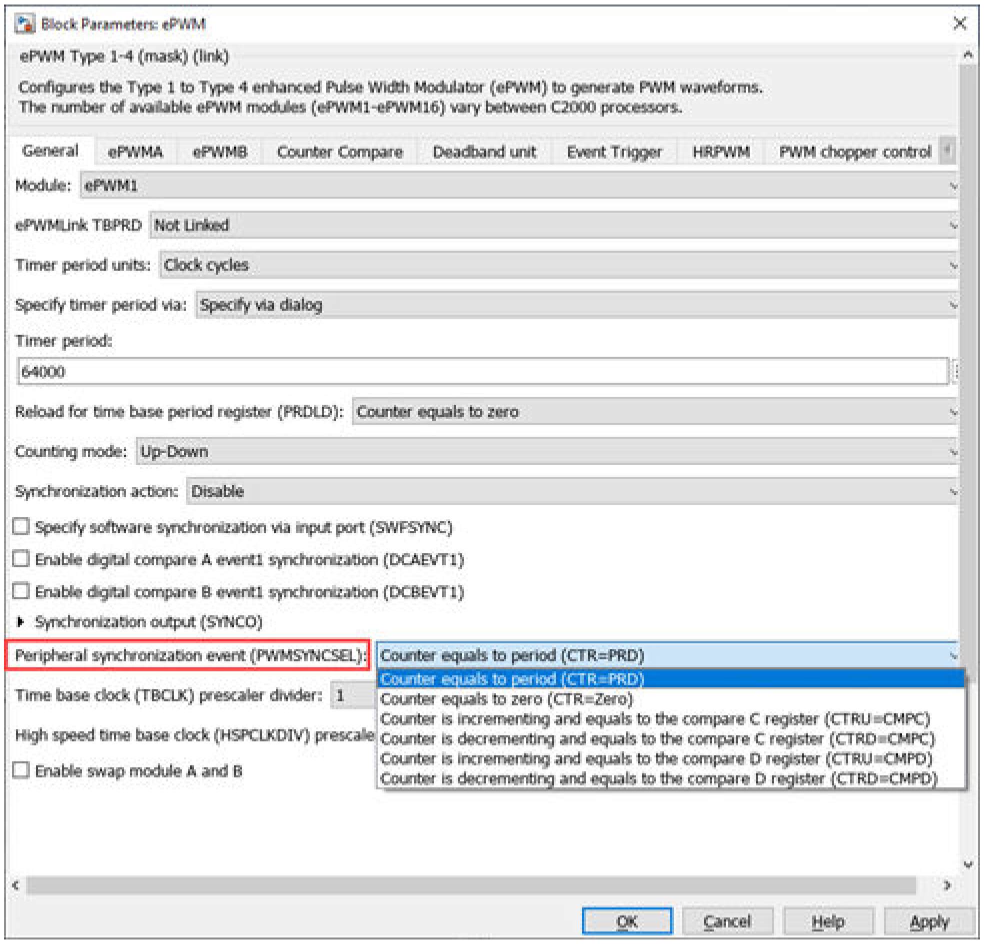
The TBCLKSYNC bit in the peripheral clock enable registers allows all users to globally synchronize all enabled ePWM modules to the time-base clock (TBCLK). When set, all enabled ePWM module clocks are started with the first rising edge of TBCLK aligned. This is done by default for the ePWM blocks.

Time-Base Counter Peripheral Synchronization

Each ePWM module has a peripheral synchronization output (SYNCPER). This output signal is used to synchronize the CMPSS to the EPWM.



The source of this signal can be configured in the ePWM block as shown below:



ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

Module: ePWM1

ePWMLink TBPRD Not Linked

Timer period units: Clock cycles

Specify timer period via: Specify via dialog

Timer period:

64000

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up-Down

Synchronization action: Disable

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

☐ Enable digital compare B event1 synchronization (DCBEVT1)

▸ Synchronization output (SYNCO)

Peripheral synchronization event (PWMSYNCSEL): Counter equals to period (CTR=PRD)

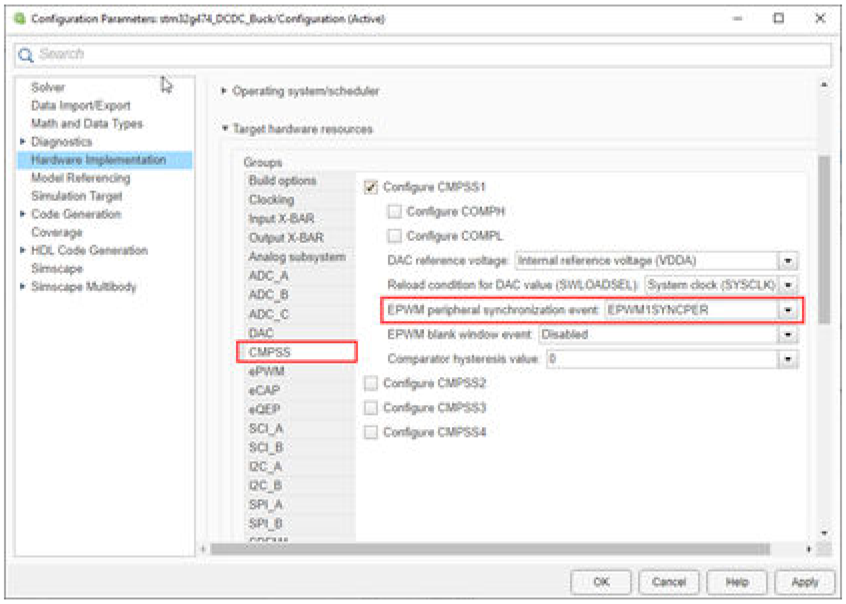
Counter equals to period (CTR=PRD)
Counter equals to zero (CTR=Zero)
Counter is incrementing and equals to the compare C register (CTRU=CMPC)
Counter is decrementing and equals to the compare C register (CTRD=CMPC)
Counter is incrementing and equals to the compare D register (CTRU=CMPD)
Counter is decrementing and equals to the compare D register (CTRD=CMPD)

Time base clock (TBCLK) prescaler divider: 1

High speed time base clock (HSPCLKDIV) prescale

☐ Enable swap module A and B

The corresponding signal can be used with CMPSS using the **EPWM peripheral synchronization event** parameter under **Hardware Implementation > Target hardware resources > CMPSS**.



☑ Configure CMPSS1

☐ Configure COMPH

☐ Configure COMPL

DAC reference voltage: Internal reference voltage (VDDA)

Reload condition for DAC value (SWLOADSEL): System clock (SYSCLK)

EPWM peripheral synchronization event: EPWM1SYNCPER

EPWM blank window event: Disabled

Comparator hysteresis value: 0

☐ Configure CMPSS2

☐ Configure CMPSS3

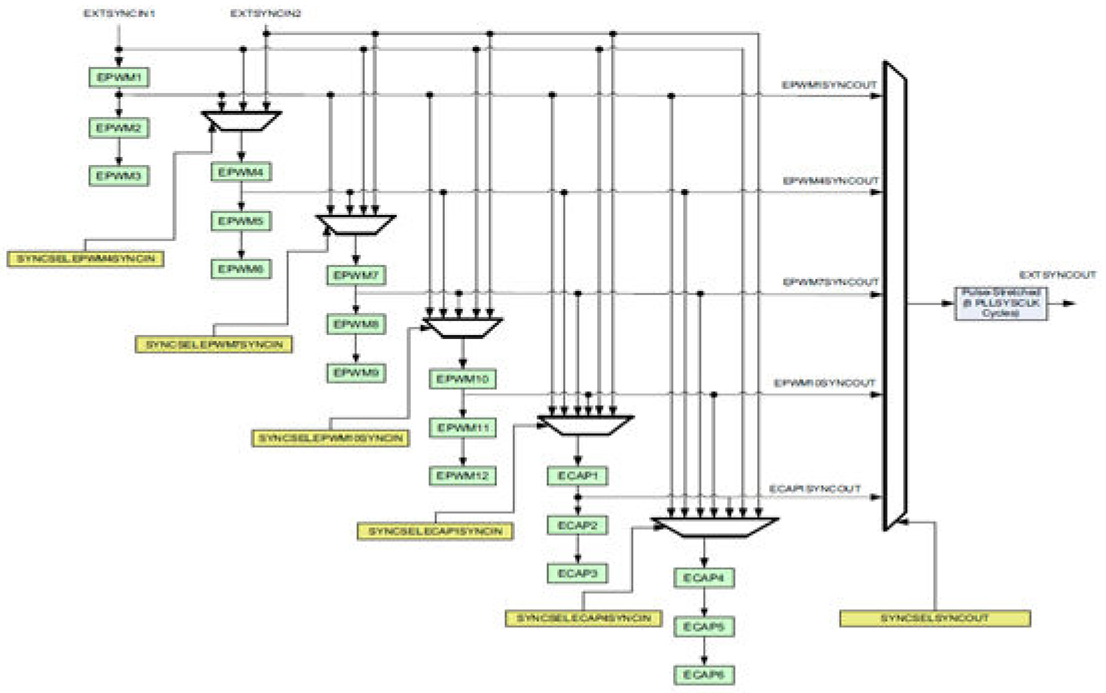
☐ Configure CMPSS4

Time-Base Counter Synchronization

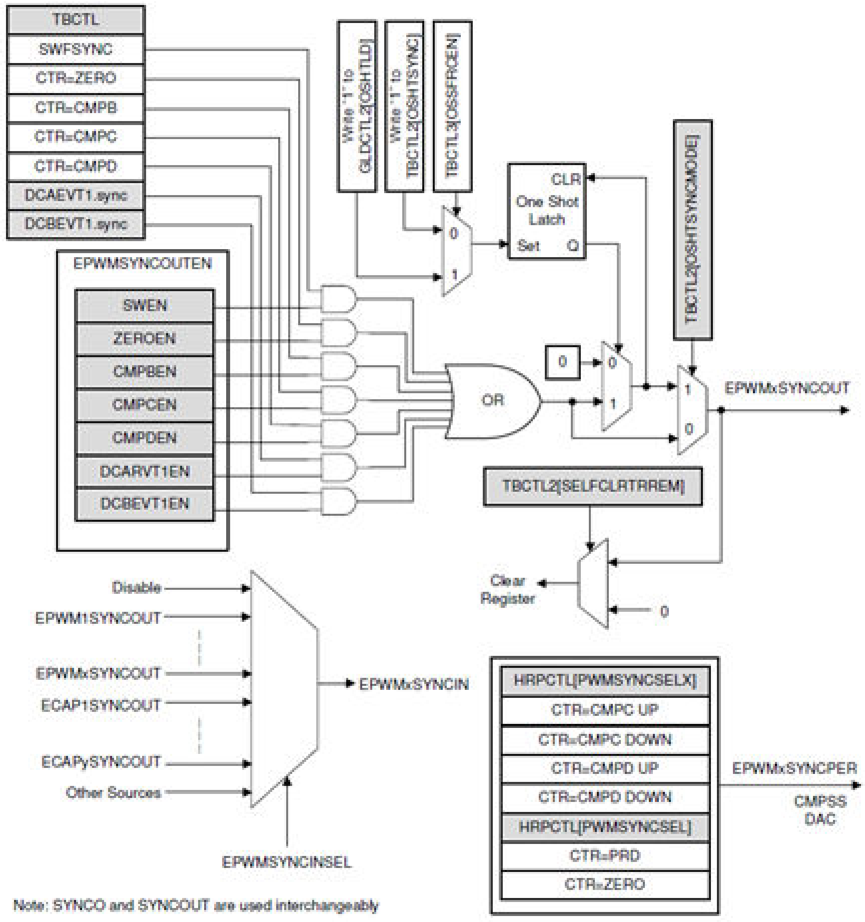
Time-base counter synchronization allows for increased flexibility of synchronization of the ePWM modules. The clock synchronization scheme allows ePWM modules to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral submodules (eCAP). In Type 4 ePWM, there are two types of time-base counter synchronization scheme available.

Time base counter synchronization options in ePWM

In processors F2837x, F2807x and F28004x, the ePWM modules are chained together as shown below.



In processors F2838x, F28002x and F28003x, there is no fixed chain of synchronization signals but each ePWM or eCAP module can be configured to use or ignore the synchronization input and can be the source of synchronization signal i.e. any ePWM or eCAP can be synchronized with any other ePWM or eCAP.

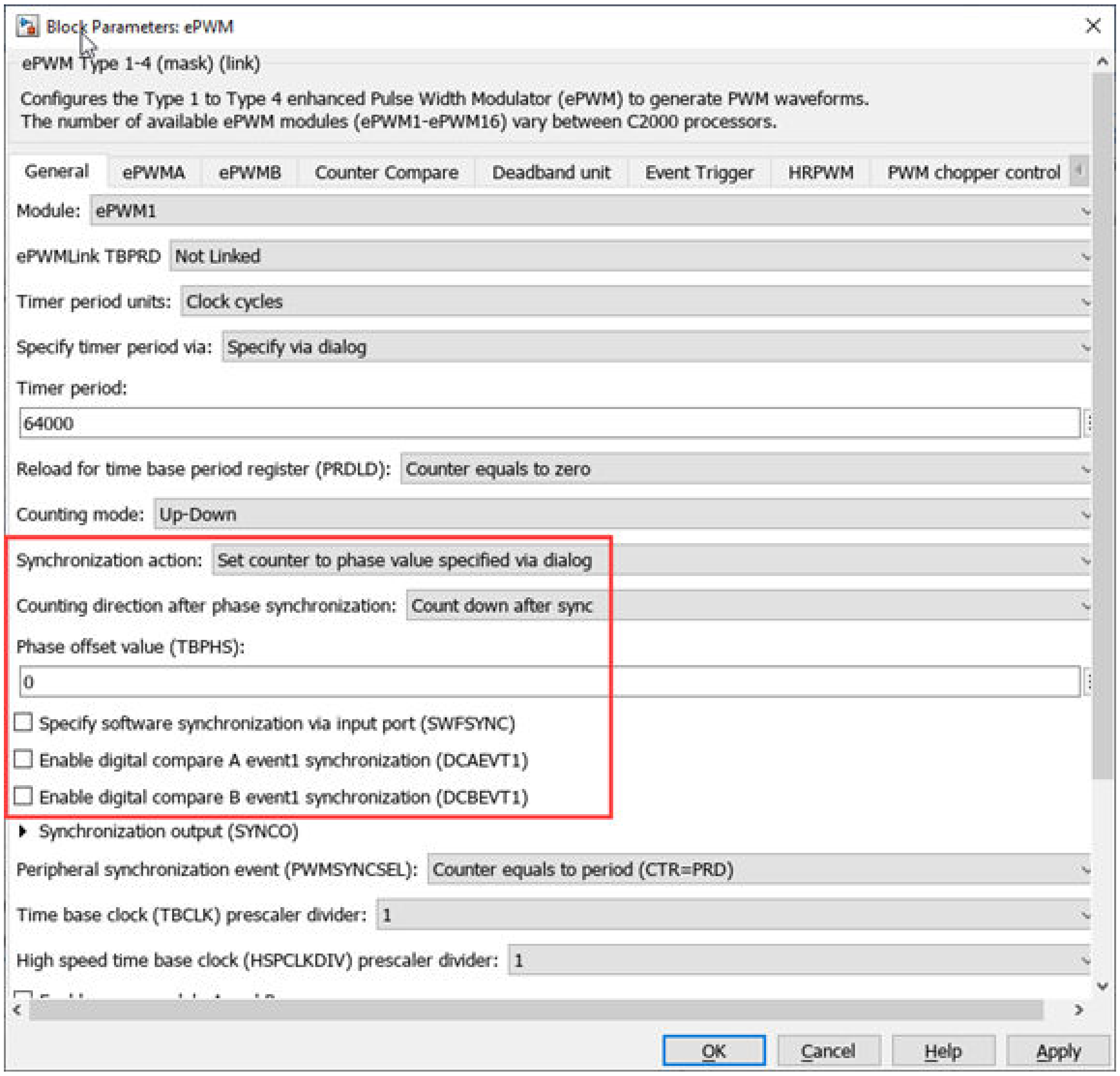


When the `PHSEN` bit in `TBCTL` is set the time-base counter (`TBCTR`) of the ePWM module is automatically loaded with the phase register (`TBPHS`) contents when one of the following conditions occur:

- `EPWMxSYNCI/Synchronization Input Pulse` - The value of the phase register is loaded into the counter register when an input synchronization pulse is detected (`TBPHS` → `TBCTR`). This operation occurs on the next valid time-base clock (TBCLK) edge.
- `Software Forced Synchronization Pulse` - Writing a 1 to the `SWFSYNC` control bit in TBCTL invokes a software forced synchronization. This pulse is **ORed** with the synchronization input signal, and therefore has the same effect as a pulse on `EPWMxSYNCI`.
- `Digital Compare Event Synchronization Pulse` - `DCAEVT1` and `DCBEVT1` digital compare events can be configured to generate synchronization pulses which have the same effect as `EPWMxSYNCI`.

  In up-down-count mode, the `PSHDIR` bit in `TBCTL` register configures the direction of the time-base counter immediately after a synchronization event. The new direction is independent of the direction prior to the synchronization event. The `PHSDIR` bit is ignored in count-up or count-down modes.

These options are set in ePWM block as shown below.


Block Parameters: ePWM

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

Module: ePWM1

ePWMLink TBPRD Not Linked

Timer period units: Clock cycles

Specify timer period via: Specify via dialog

Timer period:

64000

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up-Down

Synchronization action: Set counter to phase value specified via dialog

Counting direction after phase synchronization: Count down after sync

Phase offset value (TBPHS):

0

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

☐ Enable digital compare B event1 synchronization (DCBEVT1)

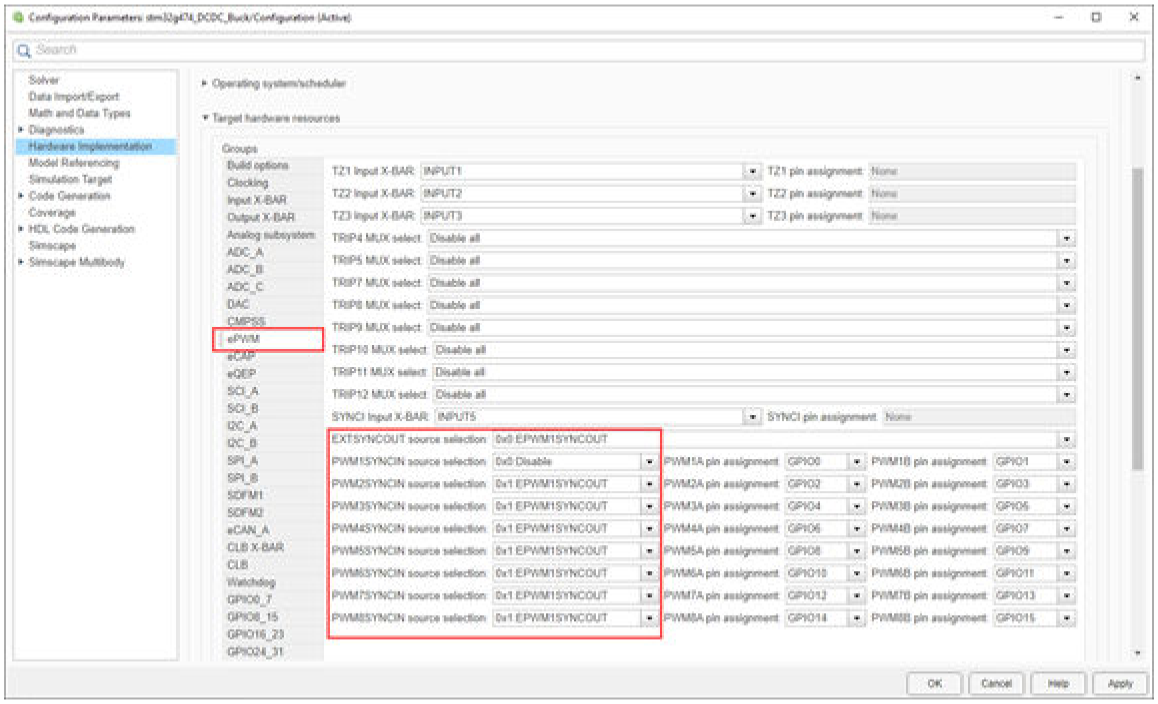
Synchronization output (SYNCO)

Peripheral synchronization event (PWMSYNCSEL): Counter equals to period (CTR=PRD)

Time base clock (TBCLK) prescaler divider: 1

High speed time base clock (HSPCLKDIV) prescaler divider: 1

The source for EXTSYNCOUT and the EPWMSYNCI signal can be selected under **Hardware Implementation > Target hardware resources > ePWM** as shown.

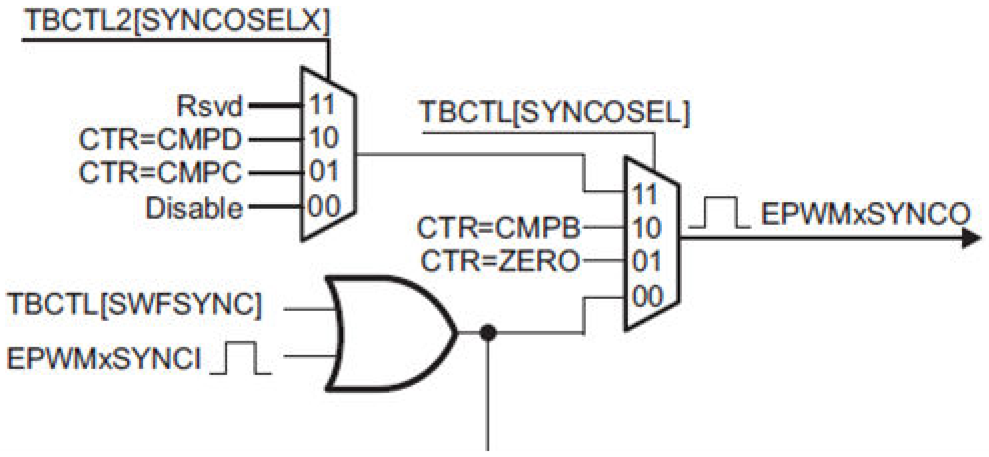


TZ1 Input X-BAR: INPUT1 | TZ1 pin assignment: None
TZ2 Input X-BAR: INPUT2 | TZ2 pin assignment: None
TZ3 Input X-BAR: INPUT3 | TZ3 pin assignment: None
TRIP4 MUX select: Disable all
TRIP5 MUX select: Disable all
TRIP7 MUX select: Disable all
TRIP8 MUX select: Disable all
TRIP9 MUX select: Disable all
TRIP10 MUX select: Disable all
TRIP11 MUX select: Disable all
TRIP12 MUX select: Disable all
SYNCI Input X-BAR: INPUT5 | SYNCI pin assignment: None
EXTSYNCOUT source selection: 0x0 EPWM1SYNCOUT
PWM1SYNCIN source selection: 0x0 Disable | PWM1A pin assignment: GPIO0 | PWM1B pin assignment: GPIO1
PWM2SYNCIN source selection: 0x1 EPWM1SYNCOUT | PWM2A pin assignment: GPIO2 | PWM2B pin assignment: GPIO3
PWM3SYNCIN source selection: 0x1 EPWM1SYNCOUT | PWM3A pin assignment: GPIO4 | PWM3B pin assignment: GPIO5
PWM4SYNCIN source selection: 0x1 EPWM1SYNCOUT | PWM4A pin assignment: GPIO6 | PWM4B pin assignment: GPIO7
PWM5SYNCIN source selection: 0x1 EPWM1SYNCOUT | PWM5A pin assignment: GPIO8 | PWM5B pin assignment: GPIO9
PWM6SYNCIN source selection: 0x1 EPWM1SYNCOUT | PWM6A pin assignment: GPIO10 | PWM6B pin assignment: GPIO11
PWM7SYNCIN source selection: 0x1 EPWM1SYNCOUT | PWM7A pin assignment: GPIO12 | PWM7B pin assignment: GPIO13
PWM8SYNCIN source selection: 0x1 EPWM1SYNCOUT | PWM8A pin assignment: GPIO14 | PWM8B pin assignment: GPIO15

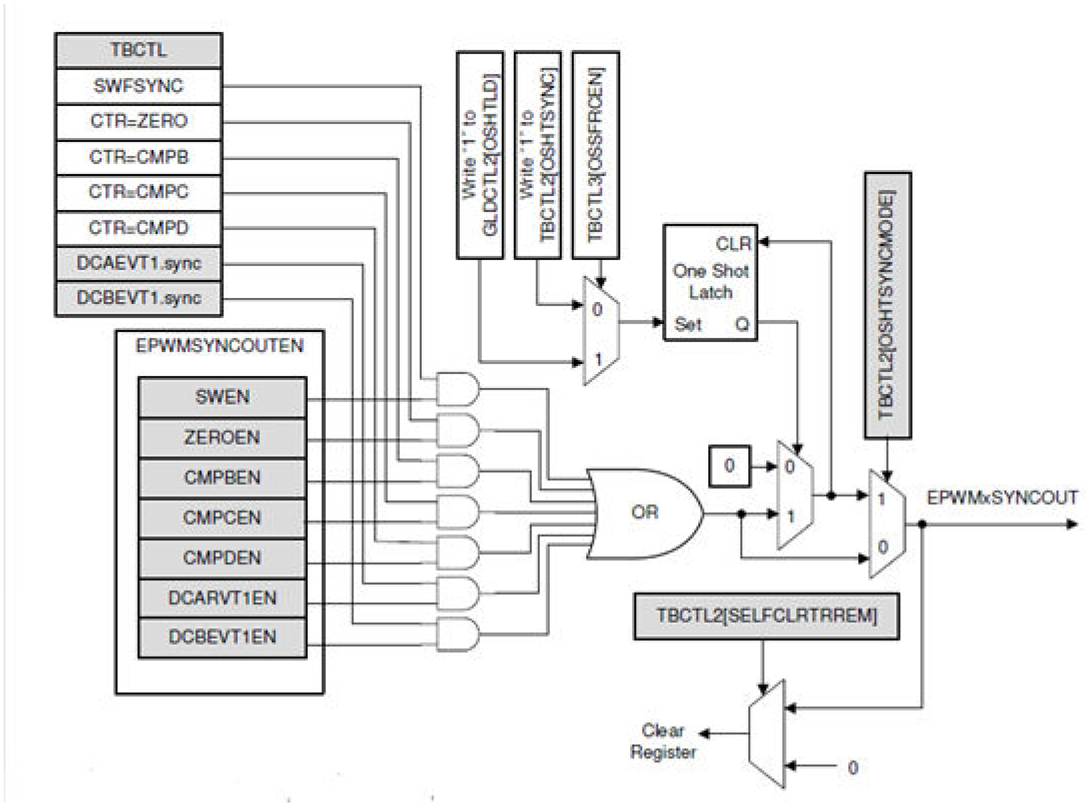
Note

- `EXTSYNCOUT` and `EPWMSYNCI` parameters vary based on the processor.
- For processors F28004x/F2837xD/F2837xS/F2807x, EXTSYNCIN1 and EXTSYNCIN2 are mapped to Input X-BAR 5 and Input X-BAR 6 respectively.
- For processors F2838x/F28003x/F28002x, EXTSYNCOUT option can be used to send synchronization output from eCAP in a model without ePWM blocks.

`EPWMxSYNCO` signal is the output pulse is used to synchronize the counter of other ePWM modules. For processors F2837x/F2807x and F28004x, `EPWMxSYNCO` signal can be generated as shown in the diagram below.



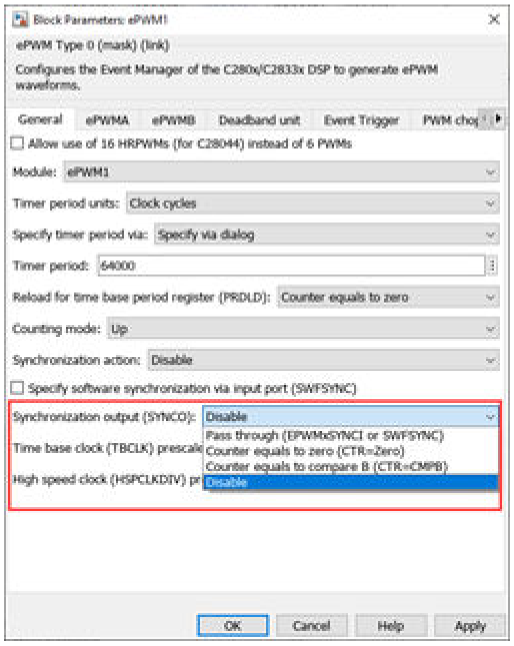
For processors F2838x, F28002x and F28003x, `EPWMxSYNCO` signal can be generated under multiple conditions as shown below.



Note Configuration of one-shot sync mode is currently not supported.

`EPWMxSYNCO` can be configured in the ePWM block as shown below.

For **F2837x/07x/004x** processors.


Block Parameters: ePWM1

ePWM Type 0 (mask) (link)

Configures the Event Manager of the C280x/C2833x DSP to generate ePWM waveforms.

☐ Allow use of 16 HRPWMs (for C28044) instead of 6 PWMs

Module: ePWM1

Timer period units: Clock cycles

Specify timer period via: Specify via dialog

Timer period: 64000

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up

Synchronization action: Disable

☐ Specify software synchronization via input port (SWFSYNC)

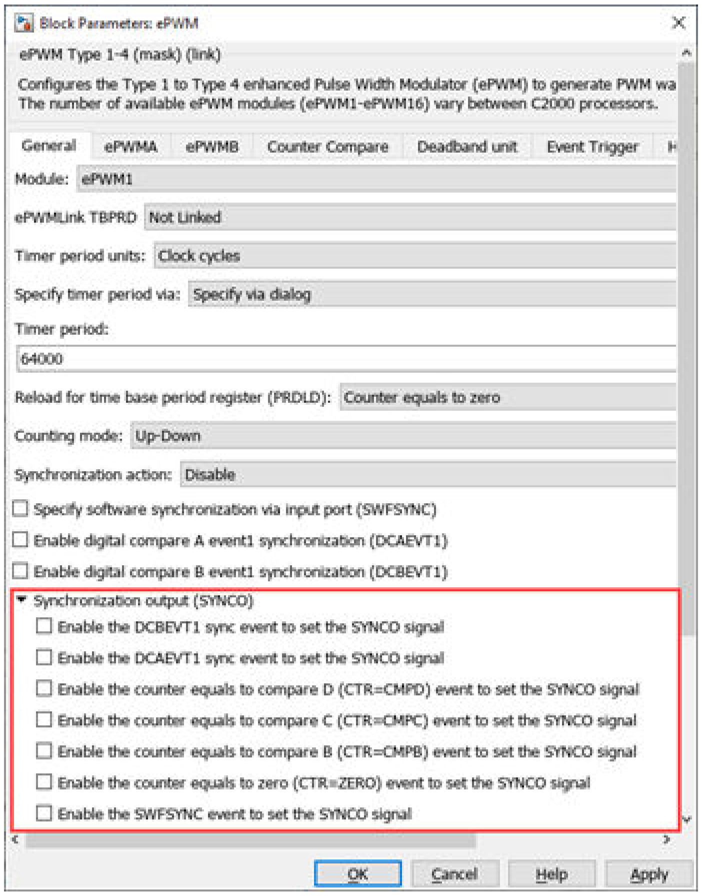
Synchronization output (SYNCO): Disable

- Pass through (EPWMxSYNCI or SWFSYNC)
- Counter equals to zero (CTR=Zero)
- Counter equals to compare B (CTR=CMPB)
- Disable

Time base clock (TBCLK) prescale

High speed clock (HSPCLKDIV) pr

For F2838x/002x and 003x processors.


Block Parameters: ePWM

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM wa
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

Module: ePWM1

ePWMLink TBPRD Not Linked

Timer period units: Clock cycles

Specify timer period via: Specify via dialog

Timer period:
64000

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up-Down

Synchronization action: Disable

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

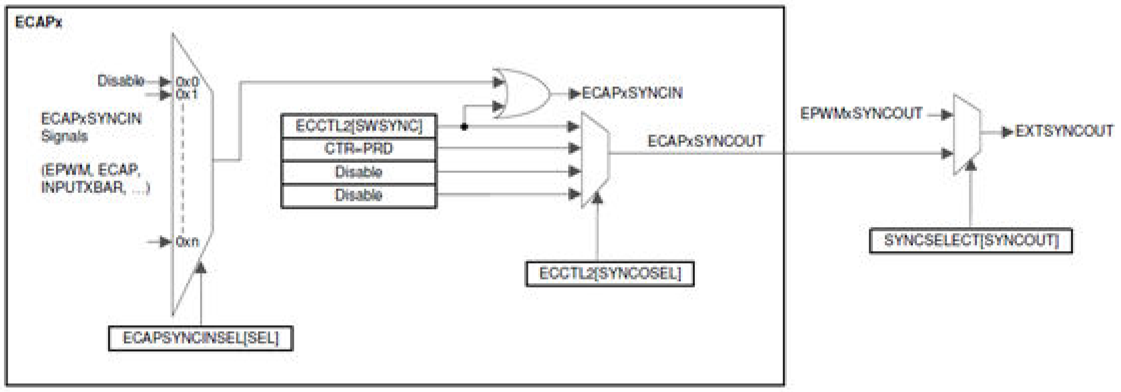
☐ Enable digital compare B event1 synchronization (DCBEVT1)

▾ Synchronization output (SYNCO)

- ☐ Enable the DCBEVT1 sync event to set the SYNCO signal
- ☐ Enable the DCAEVT1 sync event to set the SYNCO signal
- ☐ Enable the counter equals to compare D (CTR=CMPD) event to set the SYNCO signal
- ☐ Enable the counter equals to compare C (CTR=CMPC) event to set the SYNCO signal
- ☐ Enable the counter equals to compare B (CTR=CMPB) event to set the SYNCO signal
- ☐ Enable the counter equals to zero (CTR=ZERO) event to set the SYNCO signal
- ☐ Enable the SWFSYNC event to set the SYNCO signal

Time base counter synchronization options in eCAP

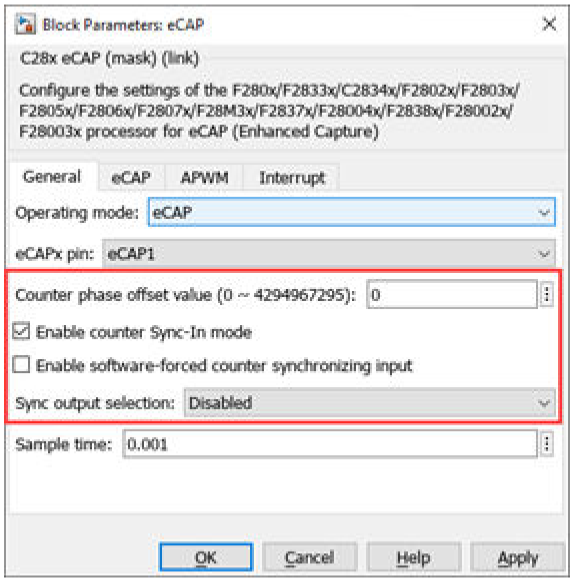
eCAP modules can be synchronized with each other by selecting a common `SYNCIN` source. `SYNCIN` source for eCAP can be either software sync-in or external sync-in. The external sync-in signal can come from eCAP, X-Bar or EPWM.



`SYNCOUT` signal from eCAP can be passed through or generated when `CTR=PRD`.

Although synchronization is possible in both `eCAP` as well as `APWM` mode for eCAP, `CTR=PRD` source for `SYNCOUT` signal is valid only in case of eCAP in `APWM` mode.

The phase offset, software sync-in and the syncout option can be selected in eCAP block as shown below.


Block Parameters: eCAP

C28x eCAP (mask) (link)

Configure the settings of the F280x/F2833x/C2834x/F2802x/F2803x/F2805x/F2806x/F2807x/F28M3x/F2837x/F28004x/F2838x/F28002x/F28003x processor for eCAP (Enhanced Capture)

Operating mode: eCAP

eCAPx pin: eCAP1

Counter phase offset value (0 ~ 4294967295): 0

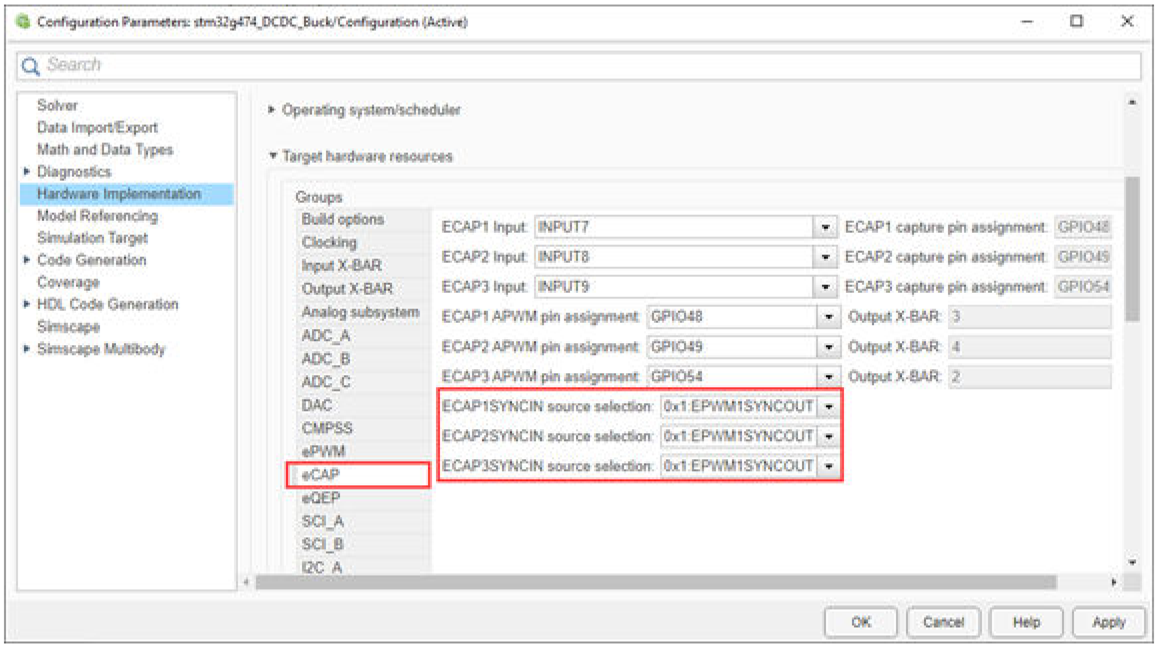
☑ Enable counter Sync-In mode

☐ Enable software-forced counter synchronizing input

Sync output selection: Disabled

Sample time: 0.001

The `eCAP SYNCIN` signal can be selected from the eCAP tab in configuration parameter. Ensure that the `Enable counter Sync-In mode` is enabled in the eCAP block for synchronization input to have effect.


Configuration Parameters: stm32g474_DCDC_Buck/Configuration (Active)

ECAP1 Input: INPUT7 ECAP1 capture pin assignment: GPIO48
ECAP2 Input: INPUT8 ECAP2 capture pin assignment: GPIO49
ECAP3 Input: INPUT9 ECAP3 capture pin assignment: GPIO54
ECAP1 APWM pin assignment: GPIO48 Output X-BAR: 3
ECAP2 APWM pin assignment: GPIO49 Output X-BAR: 4
ECAP3 APWM pin assignment: GPIO54 Output X-BAR: 2
ECAP1SYNCIN source selection: 0x1:EPWM1SYNCOUT
ECAP2SYNCIN source selection: 0x1:EPWM1SYNCOUT
ECAP3SYNCIN source selection: 0x1:EPWM1SYNCOUT

See Also

c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM

Related Examples

- “ADC-PWM Synchronization Using ADC Interrupt” on page 4-33
- “Using Time-Base Counter Synchronization to Synchronize ePWMs and eCAPS” on page 4-289

Communicate with Hardware Using Connected IO

You can use Connected IO to communicate with the IO peripherals on the hardware during simulation.

Simulation with Connected IO is an intermediate step in the Model-Based Design workflow that bridges the gap between simulation and code generation by enabling Simulink to communicate with the hardware before deploying the model on the hardware. Connected IO enables you to modify your model design and monitor the effect of the modified design using the peripheral data from the hardware in a near real-time environment. You are not required to deploy the model on the hardware to monitor the effect of the modified design, which accelerates the simulation process.

Note We recommend completing the hardware setup by entering the `c2000setup` at the MATLAB prompt before performing Connected IO.

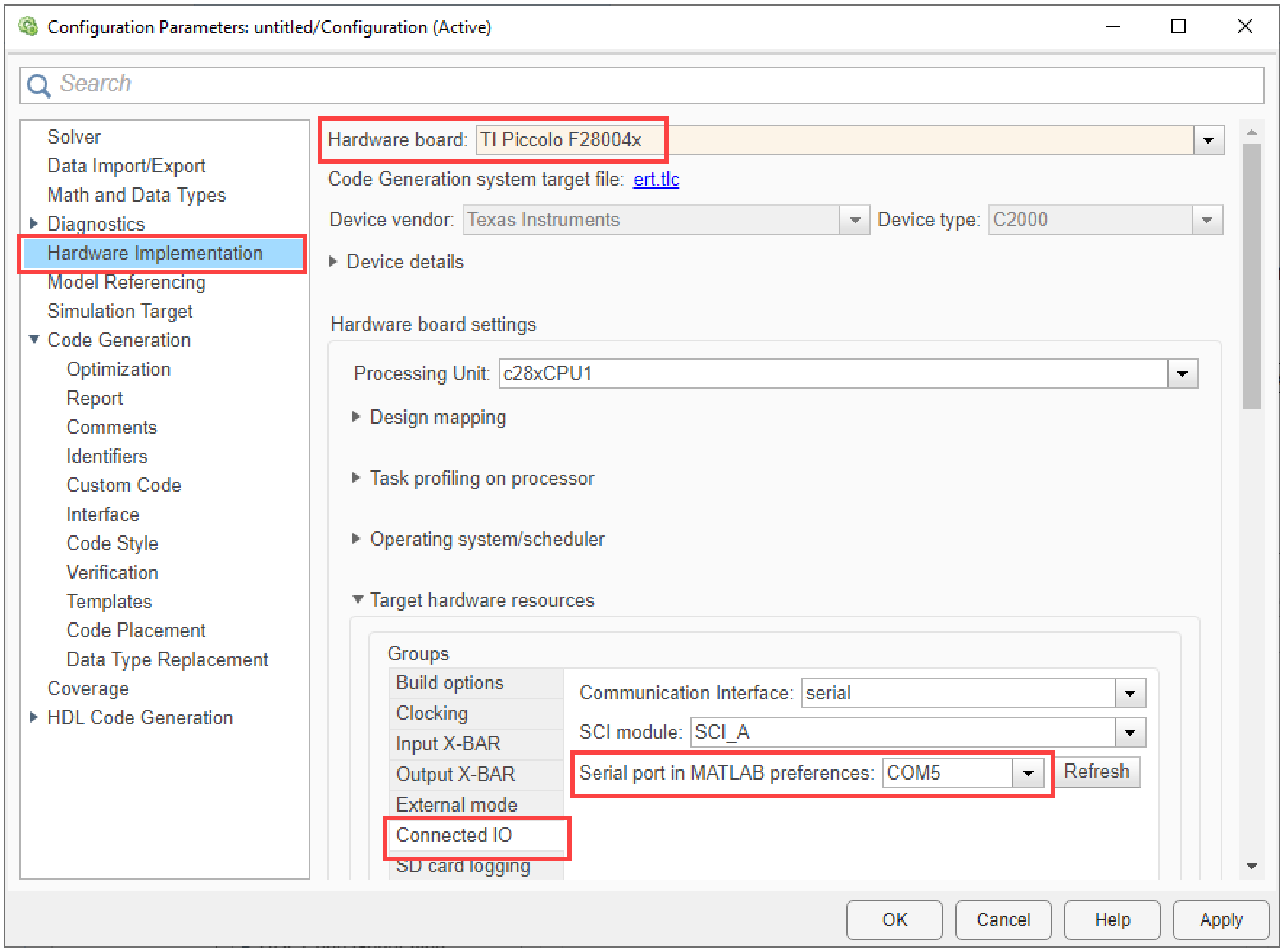
These sections explain:

In this section...

How to Enable Connected IO

To simulate a model in Connected IO, perform the following steps:

1. Open a Simulink model.
2. Press **CTRL+E** to open the Configuration Parameters dialog box.
3. In the Configuration Parameters dialog box, select **Hardware Implementation**.
4. Set the **Hardware board** parameter to any TI's C2000 board listed in the "Supported TI's C2000 Boards and Blocks with Connected IO" on page 1-142 section. For example, `TI Piccolo F28004x`. This selection automatically populates the parameters in the **Hardware board** settings with the default values for the TI's C2000 hardware.
5. From the **Groups** list under **Target hardware resources**, select **Connected IO**.
6. Specify the port to which the board is connected, for example `COM5`.


Configuration Parameters: untitled/Configuration (Active)

Hardware board: TI Piccolo F28004x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments   Device type: C2000

Hardware board settings

Processing Unit: c28xCPU1

Target hardware resources

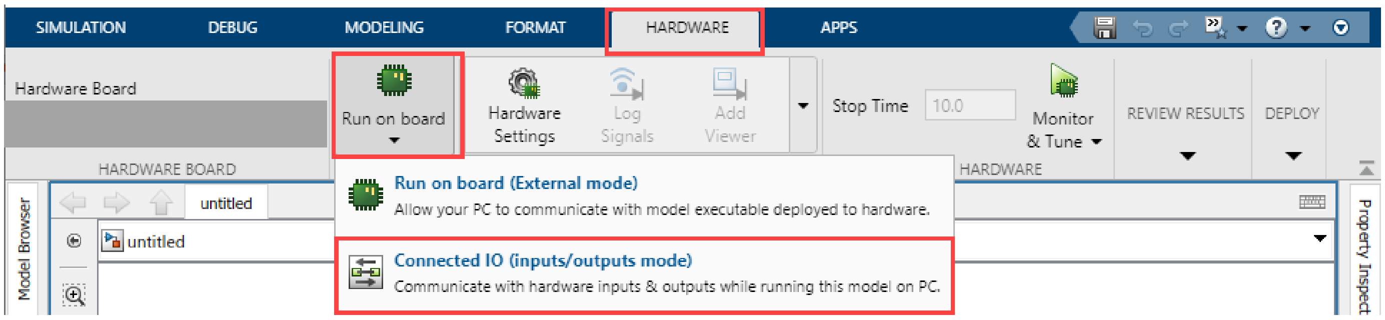
Communication Interface: serial

SCI module: SCI_A

Serial port in MATLAB preferences: COM5

7 Click **Apply**. Click **OK** to close the dialog box.

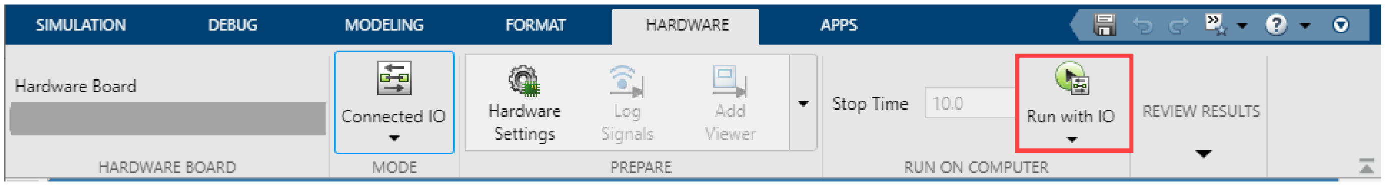
8 Go to **Hardware** tab, click on `Run on Board` and select **Connected IO (inputs/outputs mode)**.



Run on board (External mode)
Allow your PC to communicate with model executable deployed to hardware.

Connected IO (inputs/outputs mode)
Communicate with hardware inputs & outputs while running this model on PC.

9 Optionally, you can change the rate of simulation by clicking **Run with IO**. For more, see "Simulation Pacing".



Note The server for connected IO is built using `GNU Tools for ARM Embedded Processors` toolchain irrespective of the toolchain configured in Model Configuration Parameters.

Supported TI's C2000 Boards and Blocks with Connected IO

The Connected IO described here applies to the C2000 Microcontroller Blockset on these TI's C2000 boards and blocks:

- Source blocks: Without Connected IO, these source blocks output zero during simulation. With Connected IO, these blocks read data from the peripherals of the hardware during simulation.
- Sink blocks: Without Connected IO, these sink blocks do not have any role during simulation. With Connected IO, these blocks write data to the peripherals of the hardware during simulation.

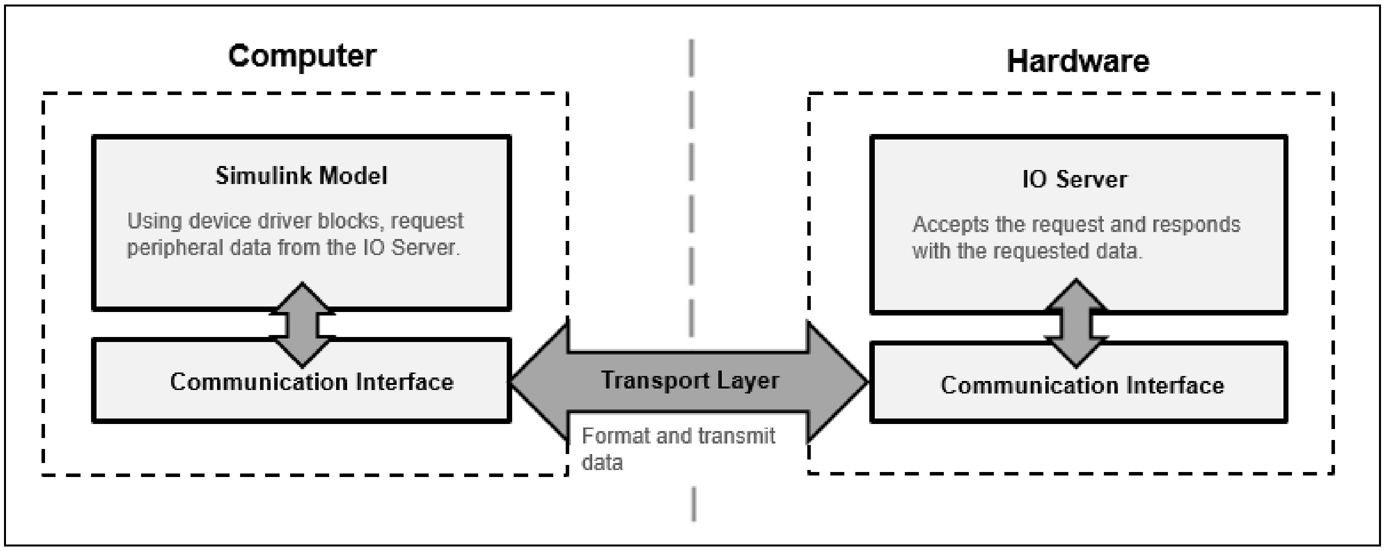
TI's C2000 Boards	Source Blocks	Sink Blocks
TI Delfino F28377S LaunchPad TI Delfino F2837xS TI Delfino F28379D LaunchPad TI Delfino F2837xD TI Delfino F2833x TI Delfino C2834x TI Piccolo F280049C LaunchPad TI Piccolo F28004x TI Piccolo F2807x TI Piccolo F2806x TI Piccolo F28069M LaunchPad TI Piccolo F2805x TI Piccolo F2803x TI Piccolo F2802x TI Piccolo F28027/F28027F LaunchPad TI F280x TI F28044 TI F2838x	C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Output C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ADC C280x/C2833x ADC	C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Input c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM

TI’s C2000 Boards	Source Blocks	Sink Blocks
TI F28002x		
TI F280025C LaunchPad		

How Connected IO Works

Connected IO creates a communication interface that enables the Simulink model and the IO Server to communicate with each other. The Simulink model resides in your computer, and the IO Server is an engine on the hardware that contains all peripheral functions. The transport layer formats and transmits the data using the communication interface.

This diagram shows the connection that the Connected IO creates between your computer and the hardware.


Computer
Hardware
Simulink Model
Using device driver blocks, request peripheral data from the IO Server.
IO Server
Accepts the request and responds with the requested data.
Communication Interface
Transport Layer
Format and transmit data
Communication Interface

Communication with Connected IO

When you simulate a Simulink model in with Connected IO:

1. The device driver blocks (for example, C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Input and C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Output blocks) in the model request peripheral data from the IO Server.
2. The IO Server accepts the request and responds with the requested data. You can use any Simulink sink or dashboard block to view the received data. Using the peripheral data received, you can verify that your model design meets the requirements.
3. If necessary, you can modify the design by adding, removing, or replacing any block in the Simulink model.
4. After the model is modified, resimulate the model. During simulation, the data request from the model is communicated to the hardware. You can continue to modify and simulate the model until the expected behavior is achieved.

Note

- The communication in Connected IO is an on-demand process. The hardware sends data only when receiving a data request from the Simulink model.
- You do not have to build, deploy, and run the model on the hardware to monitor the effects of your changes in your model design.

How Connected IO Differs from External Mode

Connected IO and External mode both enable you to communicate with the hardware during simulation. However, you use Connected IO and External mode for different purposes. The table shows the actions that you can perform with each mode.

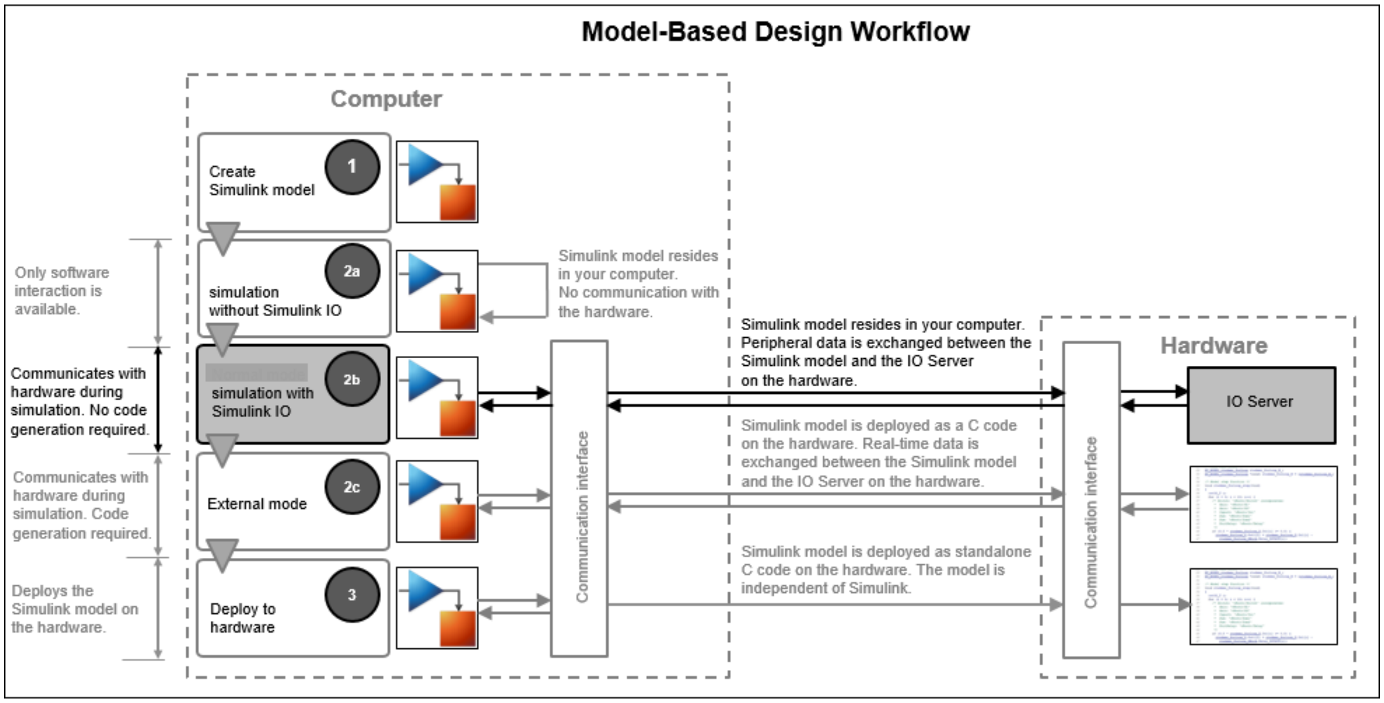
Action	External Mode	Connected IO
Obtain real-time data	You can obtain real-time data with External mode.	Enable the Simulink Pacing Option to get near real-time data.
Timing analysis of real-time data	Timing analysis of real-time data is possible because the Simulink model is running on the hardware in real-time.	Timing analysis of real-time data is not possible because the Simulink model is running in your computer and not on the hardware.
Code generation	Code is generated on the hardware.	No code is generated.

Connected IO in Model-Based Design

Simulink communicates with the hardware only when the code is generated and the model is deployed on the hardware in External mode. Simulation with Connected IO is an intermediate step in the model-based design workflow that bridges the gap between simulation and code generation by enabling Simulink to communicate with the hardware before deploying the model on the hardware.

This Model-Based Design Workflow diagram displays a model-based workflow:

1. Create a Simulink model.
2. Simulate the model in:
   a. Normal mode simulation without Connected IO: There is no hardware interaction and no code generation.
   b. Simulation with Connected IO: The model communicates with the hardware. There is no code generation.
   c. External mode: The model is deployed on the hardware and generates code.
3. Deploy the model to the hardware.


Model-Based Design Workflow
Computer
Create Simulink model
1
simulation without Simulink IO
2a
simulation with Simulink IO
2b
External mode
2c
Deploy to hardware
3
Only software interaction is available.
Communicates with hardware during simulation. No code generation required.
Communicates with hardware during simulation. Code generation required.
Deploys the Simulink model on the hardware.
Simulink model resides in your computer. No communication with the hardware.
Simulink model resides in your computer. Peripheral data is exchanged between the Simulink model and the IO Server on the hardware.
Simulink model is deployed as a C code on the hardware. Real-time data is exchanged between the Simulink model and the IO Server on the hardware.
Simulink model is deployed as standalone C code on the hardware. The model is independent of Simulink.
Communication interface
Hardware
IO Server

Model-Based Design Workflow

Limitations

- In the ePWM block, only initialization is supported. The inports, outports, HRPWM and interrupts are currently not supported for Connected IO.
- Connected IO is currently supported only for CPU1 core in TI C2000 processors.
- Connected IO is not supported for multiprocessor modelling.
- Connected works with only **timer 0** set as `scheduler interrupt source`.

See Also

"Troubleshooting Connected I/O" on page 1-146

Related Examples

- "Getting Started with Connected IO" on page 4-354

Troubleshooting Connected I/O

Description

Simulink timer is very slow while running Connected I/O simulation.

Action

This is due to presence of blocks with very less sample times. There is a two-way communication between Simulink and the Autopilot while running Connected I/O simulation. This introduces an inherent time overhead for a particular block execution. To resolve the issue, set the sample time of blocks to more than the block execution time for connected I/O simulation.

Description

Build issue due to space in temp directory while running Connected I/O.

Action

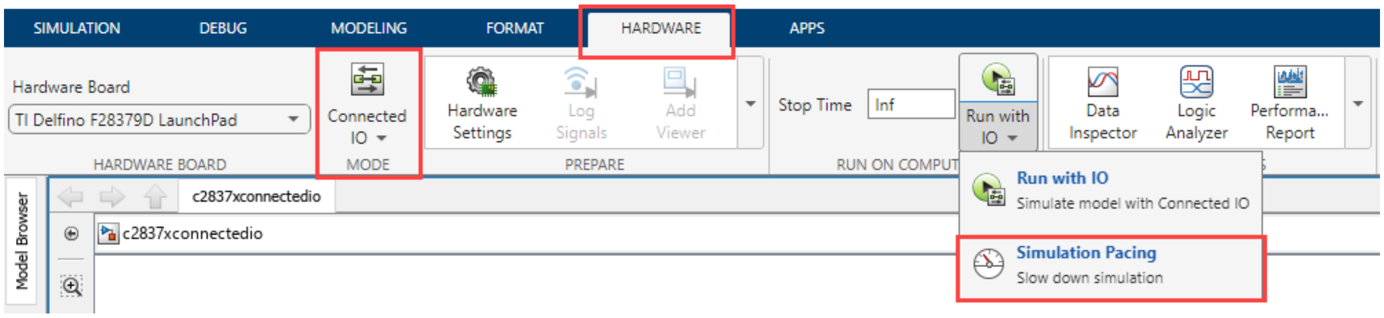
Ensure temp directory does not have any space in the full path.

Description

The scope data may not match exactly with the real-time data as it depends on SCI baud rate, simulation speed, and number of tasks performed in step function.

Action

- Ensure the SCI baud rate is configured correctly.
- Turn on the simulation pacing.



See Also

"Communicate with Hardware Using Connected IO" on page 1-140

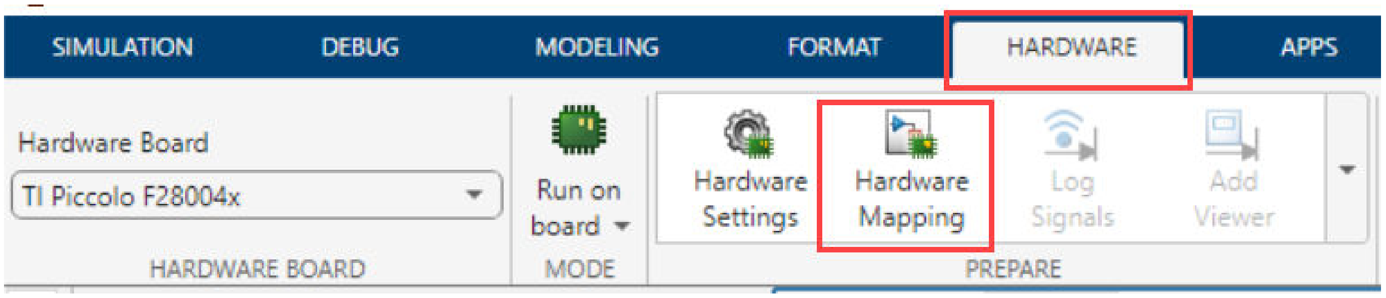
Related Examples

- "Getting Started with Connected IO" on page 4-354

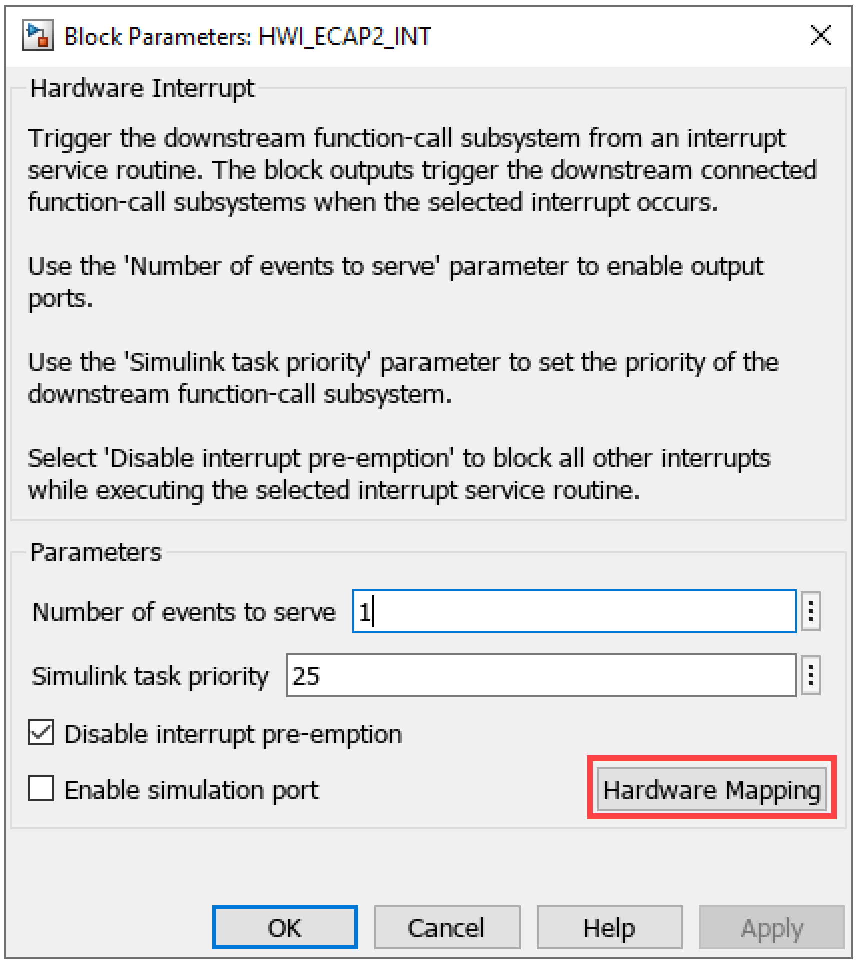
Configure Interrupts and Events Using Hardware Mapping

The **Hardware Mapping** tool allows you to configure the hardware interrupts tasks for the selected hardware board. With this tool, you can map the tasks in your software model to the available event sources and interrupts.

1 Simulink Toolstrip: On the **Hardware** tab, click **Hardware Mapping**.



Alternatively, you can also launch it from the Hardware Interrupt block.



Block Parameters: HWI_ECAP2_INT

Hardware Interrupt

Trigger the downstream function-call subsystem from an interrupt service routine. The block outputs trigger the downstream connected function-call subsystems when the selected interrupt occurs.

Use the 'Number of events to serve' parameter to enable output ports.

Use the 'Simulink task priority' parameter to set the priority of the downstream function-call subsystem.

Select 'Disable interrupt pre-emption' to block all other interrupts while executing the selected interrupt service routine.

Parameters

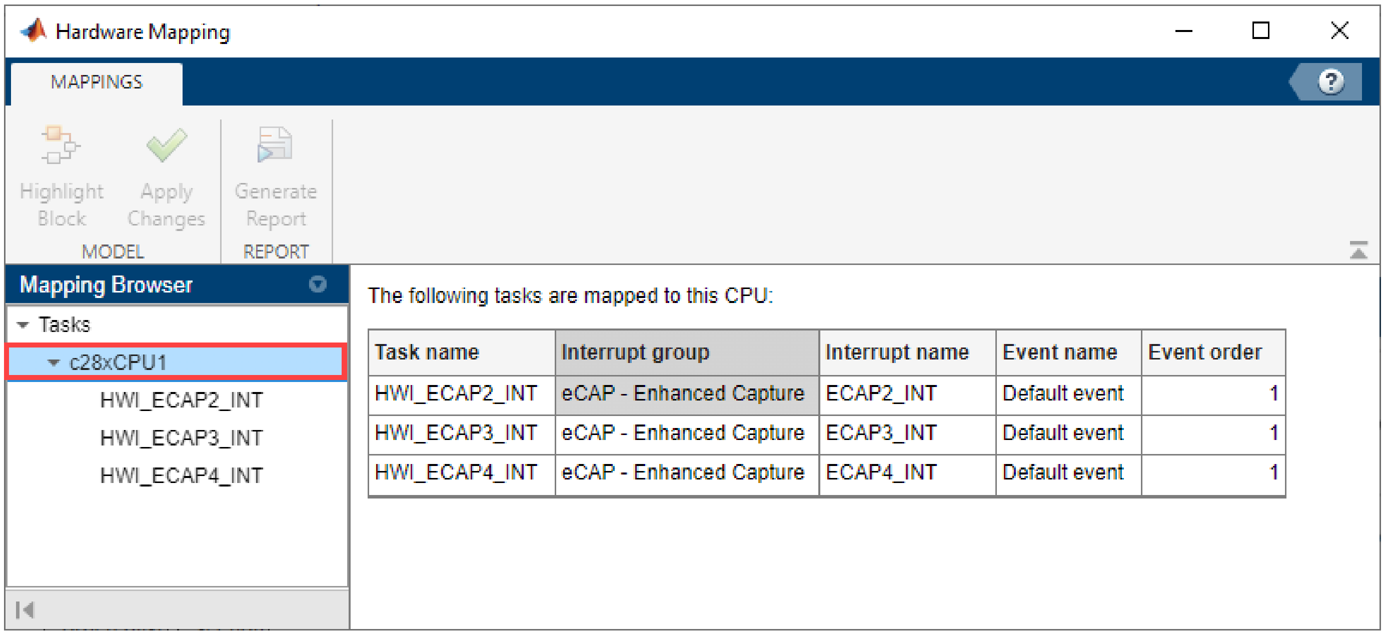
Number of events to serve: 1

Simulink task priority: 25

[✓] Disable interrupt pre-emption

[ ] Enable simulation port

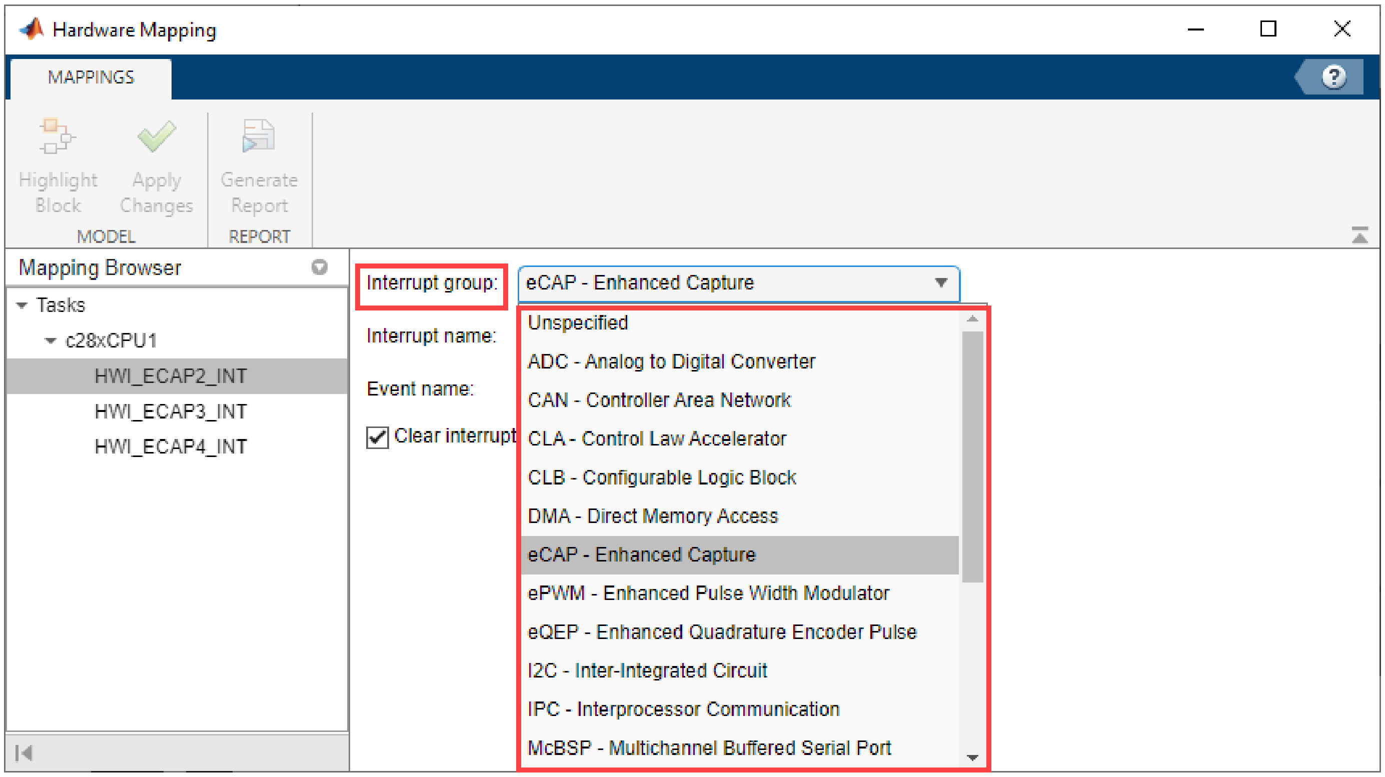
2 Manually select the task in **Browser** > **Tasks** > **`CPU name`** (c28xCPU1, c28xCPU1 or CortexM4).



The following tasks are mapped to this CPU:

Task name	Interrupt group	Interrupt name	Event name	Event order
HWI_ECAP2_INT	eCAP - Enhanced Capture	ECAP2_INT	Default event	1
HWI_ECAP3_INT	eCAP - Enhanced Capture	ECAP3_INT	Default event	1
HWI_ECAP4_INT	eCAP - Enhanced Capture	ECAP4_INT	Default event	1

3 Select the desired **Interrupt group**, **Interrupt name** and **Event name** for the corresponding task in the Hardware Interrupt block.


Hardware Mapping
MAPPINGS
Highlight Block
Apply Changes
MODEL
Generate Report
REPORT
Mapping Browser
Tasks
c28xCPU1
HWI_ECAP2_INT
HWI_ECAP3_INT
HWI_ECAP4_INT
Interrupt group:
eCAP - Enhanced Capture
Interrupt name:
Event name:
Clear interrupt
Unspecified
ADC - Analog to Digital Converter
CAN - Controller Area Network
CLA - Control Law Accelerator
CLB - Configurable Logic Block
DMA - Direct Memory Access
eCAP - Enhanced Capture
ePWM - Enhanced Pulse Width Modulator
eQEP - Enhanced Quadrature Encoder Pulse
I2C - Inter-Integrated Circuit
IPC - Interprocessor Communication
McBSP - Multichannel Buffered Serial Port

- **Interrupt group** - the TI C2000 peripheral and non-peripheral interrupts are classified into a various number of groups depending upon the peripheral causing the interrupt. The interrupt groups are populated depending upon the hardware board selected in the configuration parameters. Selecting an interrupt group changes the list of values in the **Interrupt name** parameter.

- **Interrupt name -** corresponds to the specific entry in the processor's interrupt vector table. The available ISRs depend on the interrupt group.
- The **Event name** lists all the events corresponding to the **Interrupt group** selected.
- **Event name -** lists the events that causes the interrupts in the **Interrupt name** parameter. Additionally, you can configure **Default event** and **Custom event** for the interrupts having multiple events.

  - **Custom event -** This event can be used when interrupt is triggered using software. This is used to acknowledge interrupts which are triggered without peripheral intervention.

    In this event, neither interrupt flag check nor interrupt clearing is done. The code snippet for custom event will look like:

    ```
    // custom event

    interrupt void ISRNAME()
    {

    {
    //body of the event
    }

    //Acknowledge ISR

    }
    ```
  - **Default event -** This event is selected by default. In this event all the unselected events causing the interrupt is cleared. The subsystem corresponding to this event is executed only for the unselected events. If only default event is selected, no interrupt flag is checked.

    The code snippet for default event will look like:

    ```
    // Default event

    interrupt void ISRNAME()
    {

    if(selectedevents)
    {
    //body of the event;
    //clear selected events
    }
    if(nonselectedevents)
    {
    //body of the default event
    //clear all non selected events
    }
    //acknowledgeISR;
    }
    ```

- **Event Order -** You can select multiple events for an interrupt. On the Hardware Interrupt block, for parameter **Number of events to serve** provide the number that you want to acknowledge.

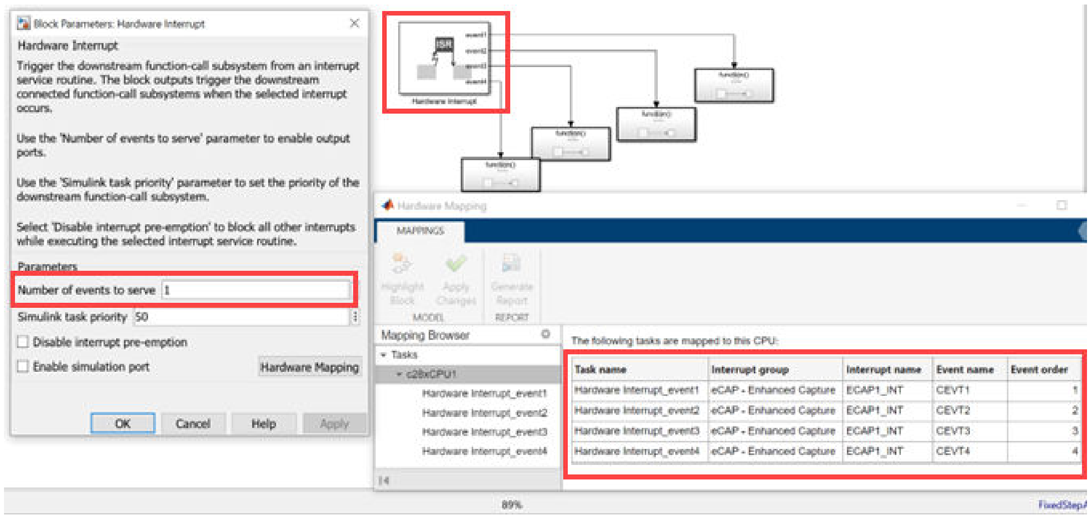
Specifying a value more than 1 in **Number of events to serve** parameter, will add that many out ports to the block. Each of the event can be configured in the hardware mapping. While configuring ensure the execution order is different for each event.

The event order decides the execution order of the events in the ISR. The code snippet for multiple events will look like:

```
interrupt ISRName()
{
//Execution order for event 1 is 1
if(event1)
{
//subsytem1
//clear event1;
}
//Execution order for event 1 is 2
if(event2)
{
//subsytem2
//clear event2;
}
...
//Acknowledge ISR
}
```

For example, to create an ISRNAME for multiple events with default event is selected, the code snippet will look like:

```
interrupt ISRName()
{
//Execution order for event 1 is 1
if(event1)
{
//code for the subsystem triggered by event1
//clear event1;
}
//Execution order for event 1 is 2
if(event2)
{
//code for the subsystem triggered by event2
//clear event2;
}
//Default event
if(event3)
{
//code for the subsystem triggered by default event
//clear event3;
}
...
//Acknowledge ISR
}
```



Trigger the downstream function-call subsystem from an interrupt service routine. The block outputs trigger the downstream connected function-call subsystems when the selected interrupt occurs.

Use the 'Number of events to serve' parameter to enable output ports.

Use the 'Simulink task priority' parameter to set the priority of the downstream function-call subsystem.

Select 'Disable interrupt pre-emption' to block all other interrupts while executing the selected interrupt service routine.

Number of events to serve: 1

Simulink task priority: 50

The following tasks are mapped to this CPU:

Task name	Interrupt group	Interrupt name	Event name	Event order
Hardware Interrupt_event1	eCAP - Enhanced Capture	ECAP1_INT	CEVT1	1
Hardware Interrupt_event2	eCAP - Enhanced Capture	ECAP1_INT	CEVT2	2
Hardware Interrupt_event3	eCAP - Enhanced Capture	ECAP1_INT	CEVT3	3
Hardware Interrupt_event4	eCAP - Enhanced Capture	ECAP1_INT	CEVT4	4

The sources of events or interrupts depend on the choice of hardware board available in the model.

4 Click the **Apply Changes** button in the toolstrip.

See Also

Hardware Mapping | Hardware Interrupt | C28x Hardware Interrupt

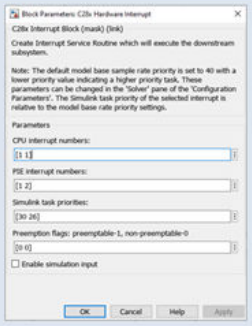
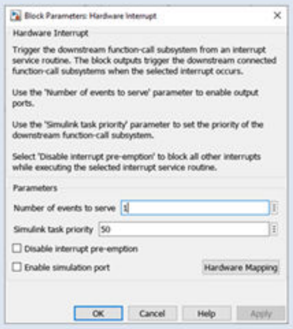
Modeling and Code Generation Using Hardware Interrupt

Use the Hardware Interrupt block to create an interrupt service routine (ISR) automatically in the generated code of your model. The ISR executes the downstream function-call subsystem associated with event ports of the block.

C2000 Microcontroller Blockset provides 3 workflows to configure the interrupts and generate code in Simulink.

- Modeling and Code generation using the C28x Hardware Interrupt block.
- Configuring interrupts with events using Hardware Interrupt block and Hardware Mapping tool.
- Multiprocessor modeling using Task Manager block and Hardware Mapping tool.

The following table describes the capabilities of C28x Hardware Interrupt and Hardware Interrupt blocks during modeling and code generation.

Feature	C28x Hardware Interrupt Block	Hardware Interrupt Block
	 Block Parameters: C28x Hardware Interrupt C28x Interrupt Block (mask) (link) Create Interrupt Service Routine which will execute the downstream subsystem. Note: The default model base sample rate priority is set to 40 with a lower priority value indicating a higher priority task. These parameters can be changed in the 'Solver' pane of the 'Configuration Parameters'. The Simulink task priority of the selected interrupt is relative to the model base rate priority settings. Parameters CPU interrupt numbers: [1 1] PIE interrupt numbers: [1 2] Simulink task priorities: [30 26] Preemption flags: preemptable-1, non-preemptable-0: [0 0] ☐ Enable simulation input OK Cancel Help Apply	 Block Parameters: Hardware Interrupt Hardware Interrupt Trigger the downstream function-call subsystem from an interrupt service routine. The block outputs trigger the downstream connected function-call subsystems when the selected interrupt occurs. Use the 'Number of events to serve' parameter to enable output ports. Use the 'Simulink task priority' parameter to set the priority of the downstream function-call subsystem. Select 'Disable interrupt pre-emption' to block all other interrupts while executing the selected interrupt service routine. Parameters Number of events to serve: 1 Simulink task priority: 50 ☐ Disable interrupt pre-emption ☐ Enable simulation port Hardware Mapping OK Cancel Help Apply
Block capability	Only one block is required for all the interrupts used in the model	For multiple interrupts use multiple Hardware Interrupt blocks with each corresponding to one interrupt.
Task priority	Simulink task priorities are added as a vector	Specify the priority in the Simulink task priority parameter
Preemption Flags	Preemption flags are added as a vector. • Preemptable – 1 • Non-preemptable - 0	Preemption is available as a check box (Disable interrupt pre-emption). • By default, the parameter is selected • Preemptable when selected • Non-preemptable when disabled
Interrupt	Interrupt is described by CPU and PIE number	You can select the interrupt names in the hardware mapping tool.

Modeling and Code generation using Hardware Interrupt

- In this workflow, you can configure interrupts using CPU and PIE numbers. For more, see C28x Hardware Interrupt
- This workflow is used for flat model approach.
- C28x Hardware Interrupt block can be used in model config set reference.
- No event selection capability is available.

- On using C28x Hardware Interrupt block, the task priority order is negative. Lower priority value indicates a higher priority task.

Modeling and Code Generation Using Hardware Interrupt and Hardware Mapping

- Hardware Interrupt blocks allows you to configure the interrupts along with the source triggers using the Hardware Mapping tool. For more information, see “Configure Interrupts and Events Using Hardware Mapping” on page 1-147
- Data of Hardware Interrupt block is stored as part of the reference config set and hence reference config set cannot be used.

Multiprocessor Modeling and Code Generation Using Task Manager and Hardware Mapping

- Task Manager blocks allows you to configure the interrupts using the Hardware Mapping tool.
- Using Task Manager block you can simulate hardware interrupts in Simulink environment and also you can implement timer-driven or event-driven. For more information, see Task Manager
- Task manager can be used in multiprocessor modeling approach, where task in a particular core is implemented using Model block.
- In multiprocessor modeling approach, higher priority value indicates a higher priority task. Navigate to **Configuration Parameters > Solver > Tasking and sample time options** and enable the **Higher priority value indicates a higher priority task**.
- Event configuration for hardware interrupts other than the default event does not work with task manager.

See Also

“Configure Interrupts and Events Using Hardware Mapping” on page 1-147 | **Hardware Mapping** | Hardware Interrupt | C28x Hardware Interrupt | Task Manager

Modeling and Code Generation Using Control Law Accelerator (CLA)

The control law accelerator (CLA) is a coprocessor available with the TI C2000 processor that allows parallel processing. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently.

C2000 Microcontroller Blockset provides two workflows to model CLA in Simulink.

- Modeling and code generation of CLA using subsystem
- Multiprocessor modeling of CLA using model reference

Modeling and Code Generation of CLA Using Subsystem

- Using subsystem workflow, you can generate and deploy the executable on the processor having CLA. For more information, see “Overview of CLA Configuration for C2000 Processors Using Subsystem” on page 1-93.
- The code generated requires less RAM (memory).
- Use the C28x CLA Task block to trigger a CLA task. The block runs the downstream function-call subsystem on the CLA.
- In subsystem workflow, the data must be configured in a memory accessed by CLA using Embedded Coder. For more information, see “CLA LSRAM Memory Configuration” on page 1-103
- The IPC blocks cannot be used to transfer data between CPU and CLA in a subsystem. For more information, see “Data Exchange Between CLA and C28x CPU” on page 1-97

Multiprocessor Modeling of CLA Using Model Reference

- Using CLA model reference workflow, you can simulate tasks according to the task priorities configured. For more information, “Modeling Control Law Accelerator (CLA) Using Model Reference” on page 1-167.
- The algorithm within CLA is modelled using model reference.
- A C2000 processor with a CLA coprocessor consists of a CLA Task Manager block and reference Model blocks containing the tasks for execution on each processor in the system. The tasks are configured using Hardware Mapping.
- Data communication between CPU and CLA requires Interprocess Data Read, Interprocess Data Write, and Interprocess Data Channel blocks. For more information, see “Link Task Execution Using Interprocess Data Channels” on page 1-169.
- The memory configuration is taken care automatically.
- The code generated requires more RAM (memory) compare to the subsystem CLA modeling. You can configure the local shared RAM (LSRAM) as CLA program and data memory. For more information, see “CLA LSRAM Memory Configuration” on page 1-103.

See Also

“Overview of CLA Configuration for C2000 Processors Using Subsystem” on page 1-93 | “Modeling Control Law Accelerator (CLA) Using Model Reference” on page 1-167

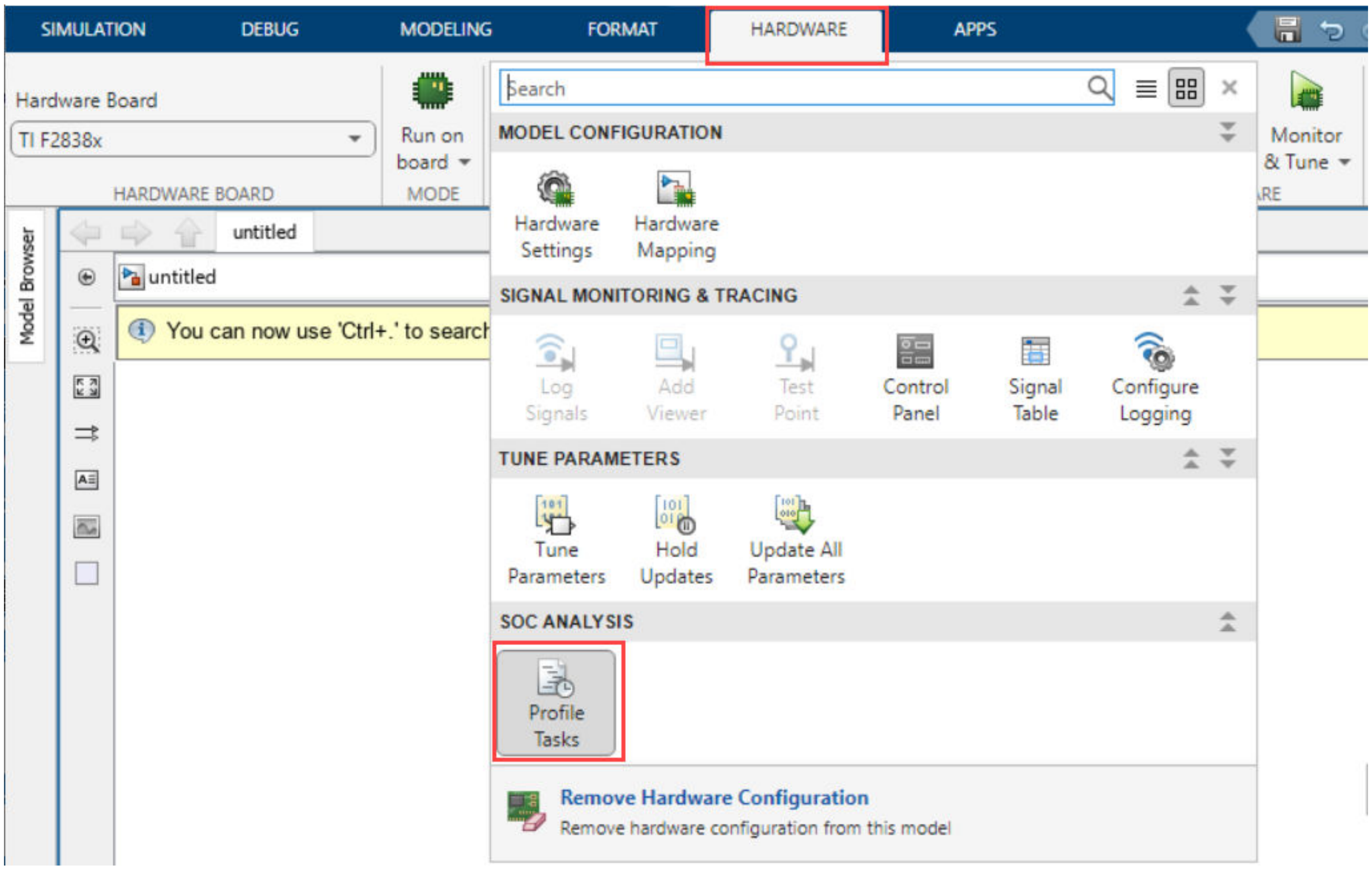
Related Examples

- “Using the Control Law Accelerator (CLA)” on page 4-270
- “Control Law Accelerator in DC-DC Power Conversion” on page 4-369

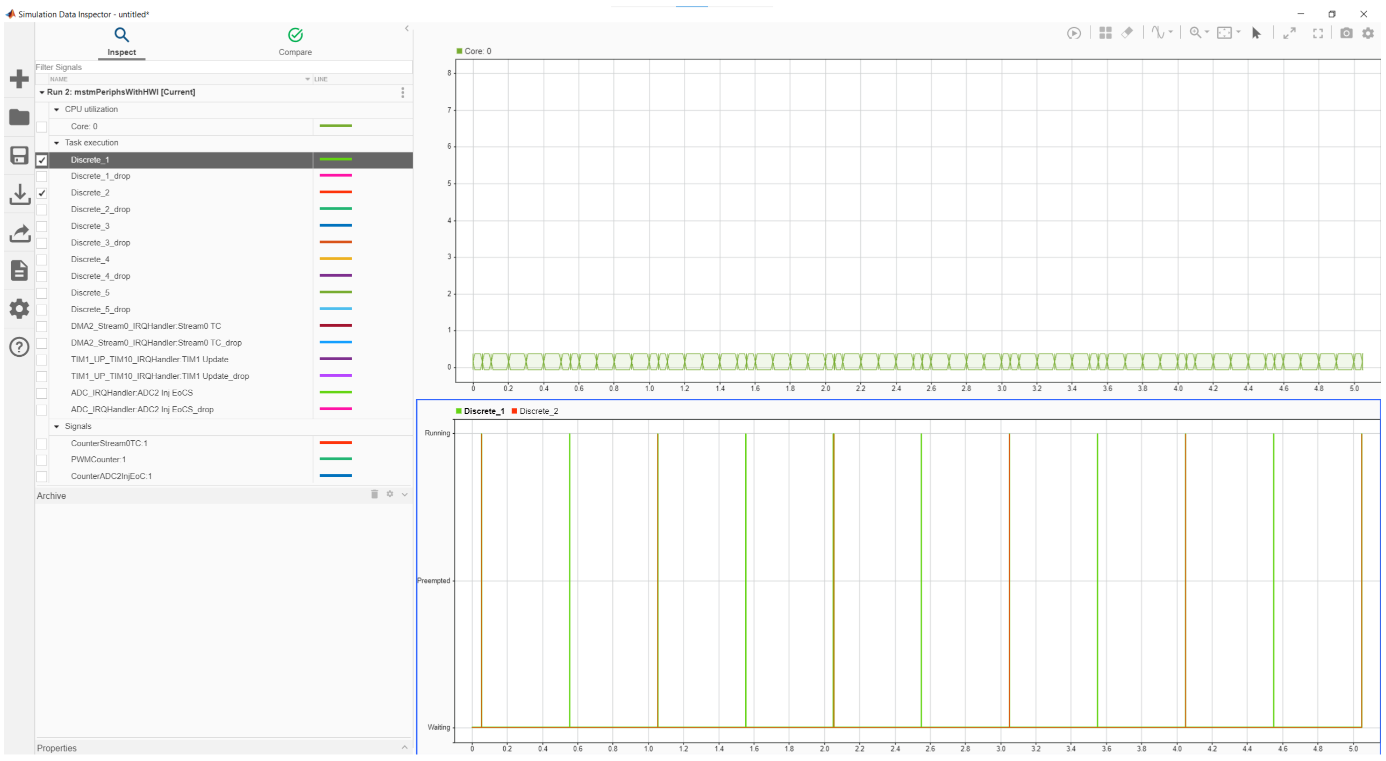
Streaming Task Profiling

Use task profiler to measure the timing of the interrupts on the TI C2000 processor. To profile interrupt timing, follow these steps.

1. Create or open an C2000 Microcontroller Blockset model.
2. On the **Hardware** tab, click **Hardware Settings**.
3. On the **Code Generation** > **Verification** > **Code execution time profiling** pane, select **Measure task execution time**. Close the Configuration Parameters window.
4. On the **Hardware** tab, enable interrupt profiling by clicking **Profile Tasks**.



5. Run the model on the hardware board by clicking **Monitor & Tune**.
6. When the run completes, open the **Simulation Data Inspector** to see the data signals and the interrupts on the C2000 Microcontroller Blockset processor. This figure shows a sample of the expected interrupt data shown. For more information on tasks in the **Simulation Data Inspector**, see “Task Visualization in Simulation Data Inspector” (SoC Blockset).



7 Using the data collected in the **Simulation Data Inspector**, you can use the **Task Execution Report** tool to generate an aggregation of the timing data from the interrupts.

See Also

Task Execution Report | **Simulation Data Inspector**

Use C2000 blocks with Multiprocessor Modeling

You can use the C2000 Microcontroller Blockset blocks with multiprocessor.

Multiprocessor blocks that interact with external hardware (for example ADC or PWM) come in pairs. The pair consists of a data input or output block, such as ADC Read, that simulates the behavior of the software drivers and an interface block, such as ADC Interface, that simulates the physical behavior of the hardware. During code generation, the data input or output block becomes the driver code for the given hardware peripheral and the interface block gets removed and replaced by the actual device hardware.

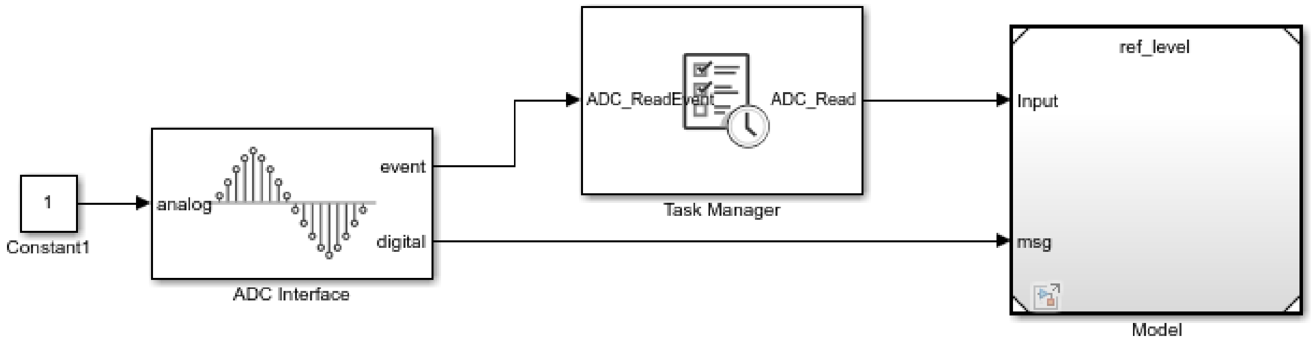
C2000 Microcontroller Blockset blocks are equivalent to the data input or output block with the multiprocessor block pairs. In code generation, the blockset generate the appropriate driver code. In simulation, due to the lack of an interface pair, the blocks do not generate any data and can be treated equivalent to the Constant block for input blocks or the Terminator block for output blocks.

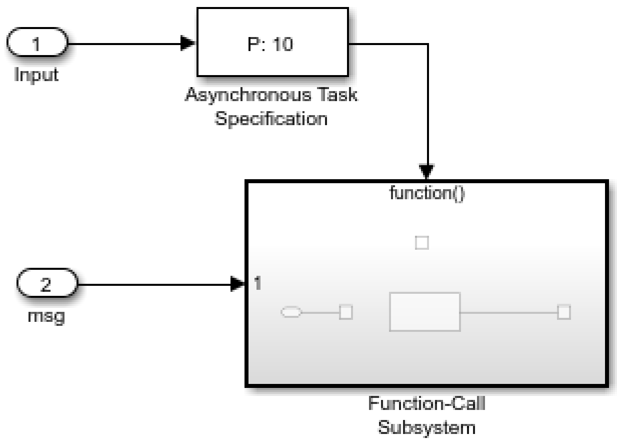
Note You can add a simulation behavior by using Variant Subsystem, Variant Model block.

Add Embedded Coder into Task

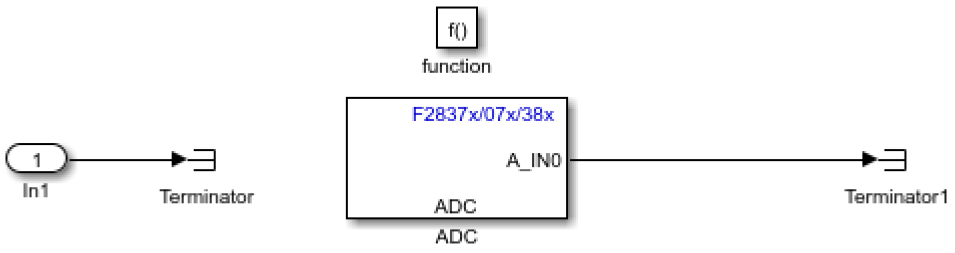
To configure an multiprocessor model to use the C2000 Microcontroller Blockset driver blocks, follow these steps.

1. Create a new multiprocessor model and set the **Hardware board** parameter to `TI F2837D Launchpad`.
2. Construct an empty event driven task that is driven by an ADC Interface block. This figure shows the sample model.





3 In the Simulink library browser, from the **C2000 Microcontroller Blockset** > **F2837xD** group, add the ADC block to the task in the model.



4 In the ADC block, set the **trigger source** parameter to `Software`.

5 Launch the **Hardware Mapping** tool. Set the **ADC_Task** parameter to `ADCA1_isr`. This setting maps the `ADCA1_isr` interrupt to the task containing the ADC block.

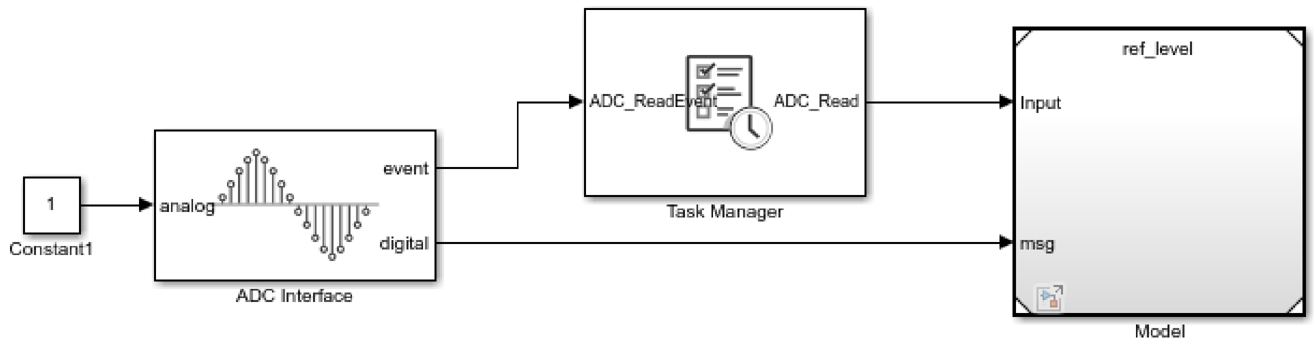
6 Use the **SoC Builder** tool to deploy the model to the connected `TI F2837D Launchpad`. The ADC block can now sample the ADCs from the hardware.

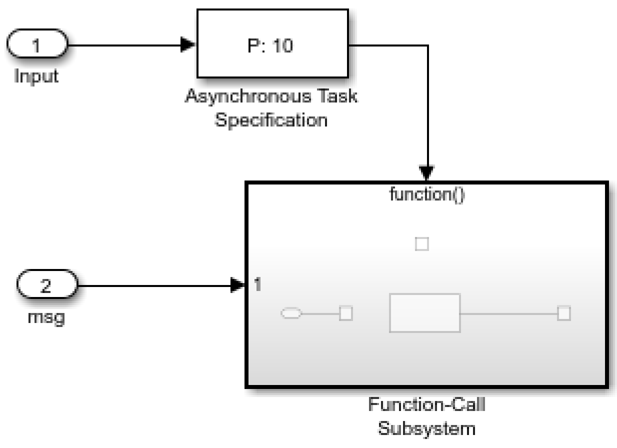
Link Embedded Coder Interrupts to Events

To configure an multiprocessor model to generate interrupts mapped to the C2000 Microcontroller Blockset blocks, follow these steps.

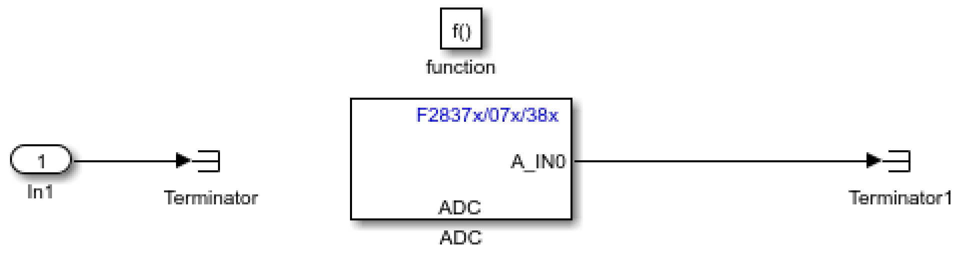
1 Create a new multiprocessor model and set the **Hardware board** parameter to `TI F2837D Launchpad`.

2 Construct an empty event driven task that is driven by an ADC Interface block. This figure shows the sample model.



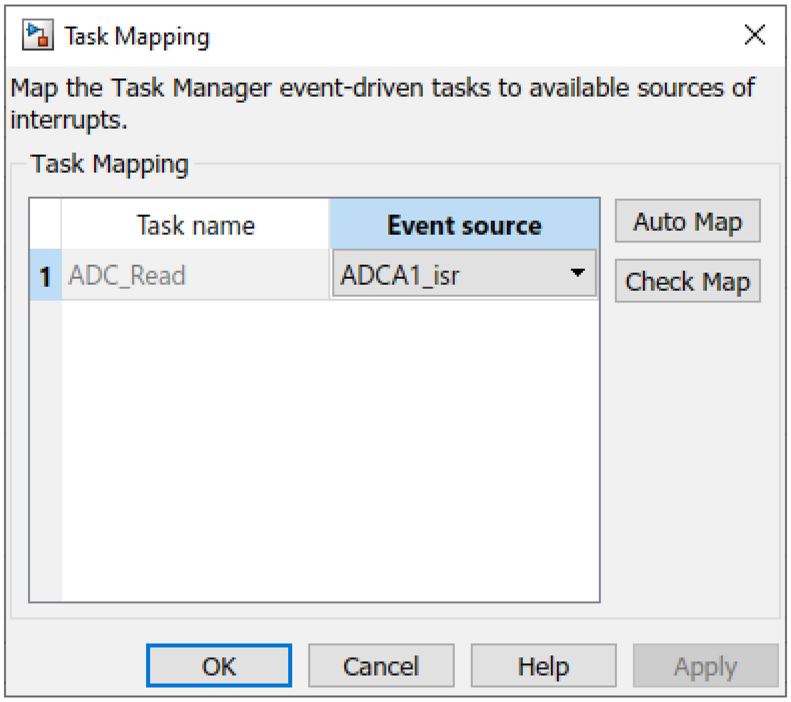


3 In the Simulink library browser, from the **C2000 Microcontroller Blockset** > **F2837xD** group, add the ADC block to the task in the model.



4 In the ADC block, set the **trigger source** parameter to `Software`.

5 Launch the **Hardware Mapping** tool. Set the **ADC_Task** parameter to `ADCA1_isr`. This maps the `ADCA1_isr` interrupt to the task containing the ADC block.



Task Mapping

Map the Task Manager event-driven tasks to available sources of interrupts.

Task Mapping

	Task name	Event source
1	ADC_Read	ADCA1_isr

6 Use the **SoC Builder** tool to deploy the model to the connected `TI F2837D Launchpad`. The ADC blocks can now samples the ADCs from the hardware on the specified interrupt.

This table shows the mapping between the interrupts in the C2000 Microcontroller Blockset blocks and the interrupts in the **Hardware Mapping** tool.

Interrupt Type	**Block Interrupt Parameter**	**Task Mapping Interrupt Parameter**
Analog to Digital Converter (ADC)	`ADCA`*`n`*	`ADCA`*`n`*`_isr`
Pulse Width Modulator (PWM)	`EPWM`*`n`*`_TZ`	`EPWM`*`n`*`_TZ_isr`
	`EPWM`*`n`*	`EPWM`*`n`*`_isr`
Electrically Evoked Compound Action Potential (ECAP)	`ECAP`*`n`*	`ECAP`*`n`*`_isr`
Enhanced Quadrature Encoder Pulse (eQEP)	`EQEP`*`n`*	`EQEP`*`n`*`_isr`
Serial Peripheral Interface (SPI)	`SPIA_RX_isr`	`SPIA_RX_isr`
	`SPIA_TX_isr`	`SPIA_TX_isr`
Direct Memory Address (DMA)	`DMA_CH`*`n`*	`DMA_CH`*`n`*`_isr`
Inter-Integrated Circuit (I2C)	`I2CA`	`I2CA_INT_isr`
	`I2CA_FIFO`	`I2CA_FIFO_isr`
Serial Communication Interface (SCI)	`SCIA_RX`	`SCIA_RX_isr`
	`SCIA_TX`	`SCIA_TX_isr`
Controller Area Network (CAN)	`CANA_`*`n`*	`CANA_`*`n`*`_isr`

Interrupt Type	Block Interrupt Parameter	Task Mapping Interrupt Parameter
External Interrupt (XINT)	`XINT`*n*	`XINT`*n*`_isr`
Fixed Point Unit (FPU)	`FPU_OVERFLOW`	`FPU_OVERFLOW_isr`
	`FPU_UNDERFLOW`	`FPU_UNDERFLOW_isr`
Viterbi, Complex Math, and CRC Unit (VCU)	`VCU`	`VCU_isr`

See Also

“Timer-Driven Task” (SoC Blockset) | “Event-Driven Tasks” (SoC Blockset) | **Hardware Mapping**

Using ARM Cortex-M Coprocessor

The ARM Cortex-M processor provides higher-level external connectivity, such as network communication. Using the ARM Cortex-M processor for asynchronous communication and supervisory tasks frees up the other CPUs to perform time-critical tasks concurrently. The ARM Cortex-M processor supports several higher-level communication APIs, such as UDP and TCP, that are not available on the C2000 processors.

Configure C2000 Model with ARM Cortex-M Processor

A C2000 processor with ARM Cortex-M processor consists of a Task Manager block and reference Model block containing the tasks for execution on each processor in the system. This model contains sample models of the C2000 processor and ARM Cortex-M processor, each with a timer-driven task.

Configure C2000 Top-Level Model

1. Open a new Simulink model. Save the model as `tif2838xD_top.slx`.
2. Configure the `tif2838xD_top.slx` model to be a C2000 application.
3. Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
4. Go to **Hardware Implementation > Hardware board** and select **TI F2838x**.
5. In the **Hardware board settings**> **Processing unit**, select `None`.
6. Click **Finish**.

Configure C2000 Processor Model

1. Open a new Simulink model. Save the model as `tif2838xD_c28xCPU1.slx`.
2. Configure the `tif2838xD_c28xCPU1.slx` model to be a C2000 application.
3. Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
4. Go to **Hardware Implementation > Hardware board** and select **TI F2838x**.
5. In the **Hardware board settings**> **Processing unit**, select `c28xCPU1`.
6. Click **Finish**.
7. Add the event- and timer-driven tasks to the C2000 processor model.
8. In the `tif2838xD_top.slx` model, add Task Manager and Model blocks.
9. Assign the `tif2838xD_c28xCPU1.slx` model to the Model block and then connect the tasks to the Task Manager block.

Configure ARM Cortex-M Processor Model

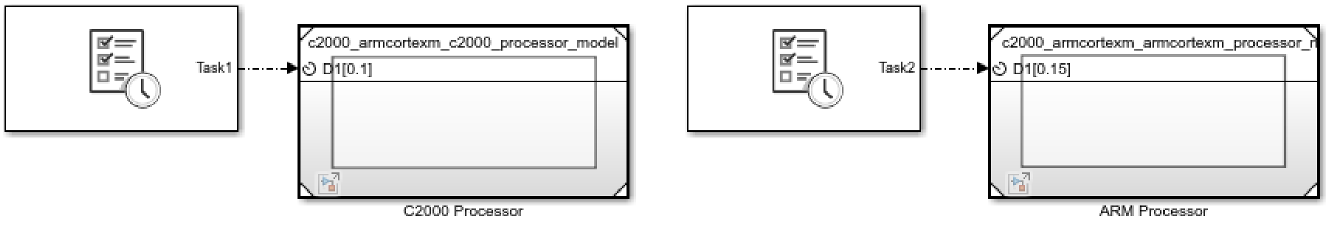
1. Open a new Simulink model. Save the model as `tif2838xD_ARMCPU3.slx`.
2. Configure the `tif2838xD_ARMCPU3.slx` model to be a C2000 application.
3. Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
4. Go to **Hardware Implementation > Hardware board** and select **TI F2838x**.
5. In the **Hardware board settings**> **Processing unit**, select `CortexM4`.

6 Click **Finish**.
7 Add event- or timer-driven tasks to the ARM Cortex-M model.
8 In the `tif2838xD_top.slx` model, add Task Manager and Model blocks.
9 Assign the `tif2838xD_ARMCPU3.slx` model to the Model block and then connect the tasks to the Task Manager block.

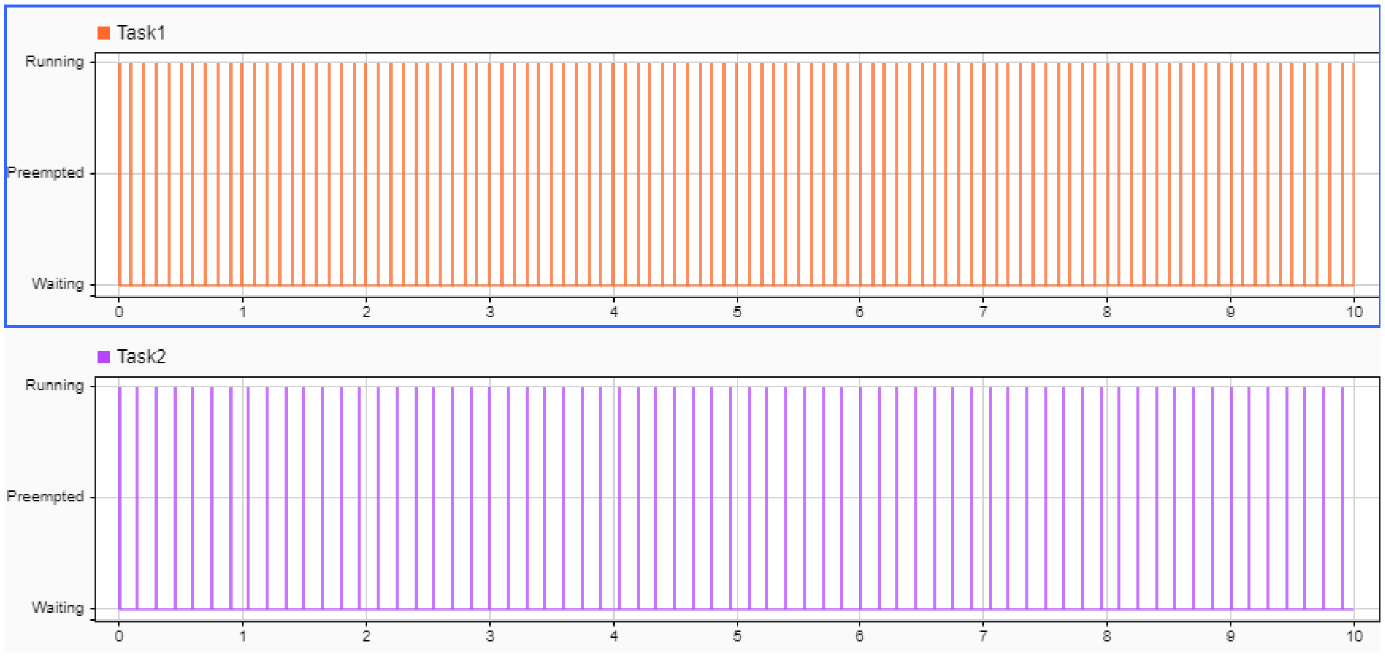
Configure C2000 and ARM Cortex-M Processor Model

This example shows how to create a C2000 model that contains the C2000 processor and the ARM Cortex-M connectivity manager (CM) processor. The C2000 and ARM Cortex-M processors contain time-varying tasks that execute at rates of 0.1 and 0.15 seconds, respectively. Open the model.

```
open_system("c2000_armcortexm_top_level_model.slx")
```


Task1
c2000_armcortexm_c2000_processor_model
D1[0.1]
C2000 Processor
Task2
c2000_armcortexm_armcortexm_processor_m
D1[0.15]
ARM Processor

Run the model and inspect the task execution in the Simulation Data Inspector. As separate processors, the tasks run without interfering with each other's execution.



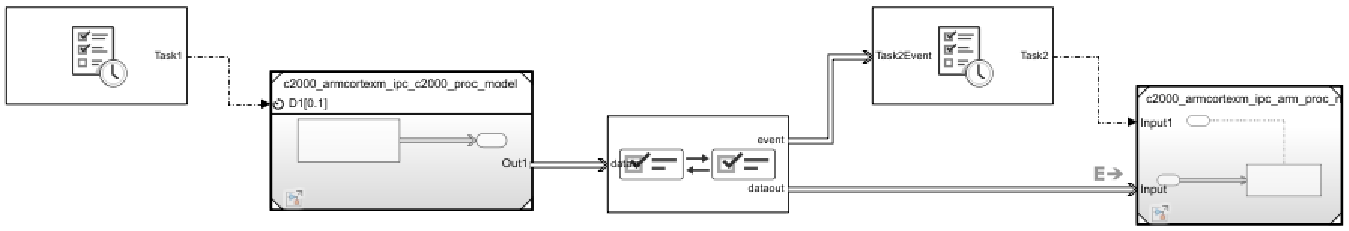
Connection Between C2000 and ARM Cortex-M Processors

The C2000 processors can connect to the ARM Cortex-M processor using the Interprocess Data Read, Interprocess Data Channel, and Interprocess Data Write block triplet. The block triplet simulates the exchange of data between the C2000 processor and the ARM Cortex-M processor. When the model generates code, custom flags in the `CPU`*`x`*`-CM IPC` module with data being transferred through shared memory, such as the `S`*`n`* RAM, replaces the block triplet for the specified C2000 processor.

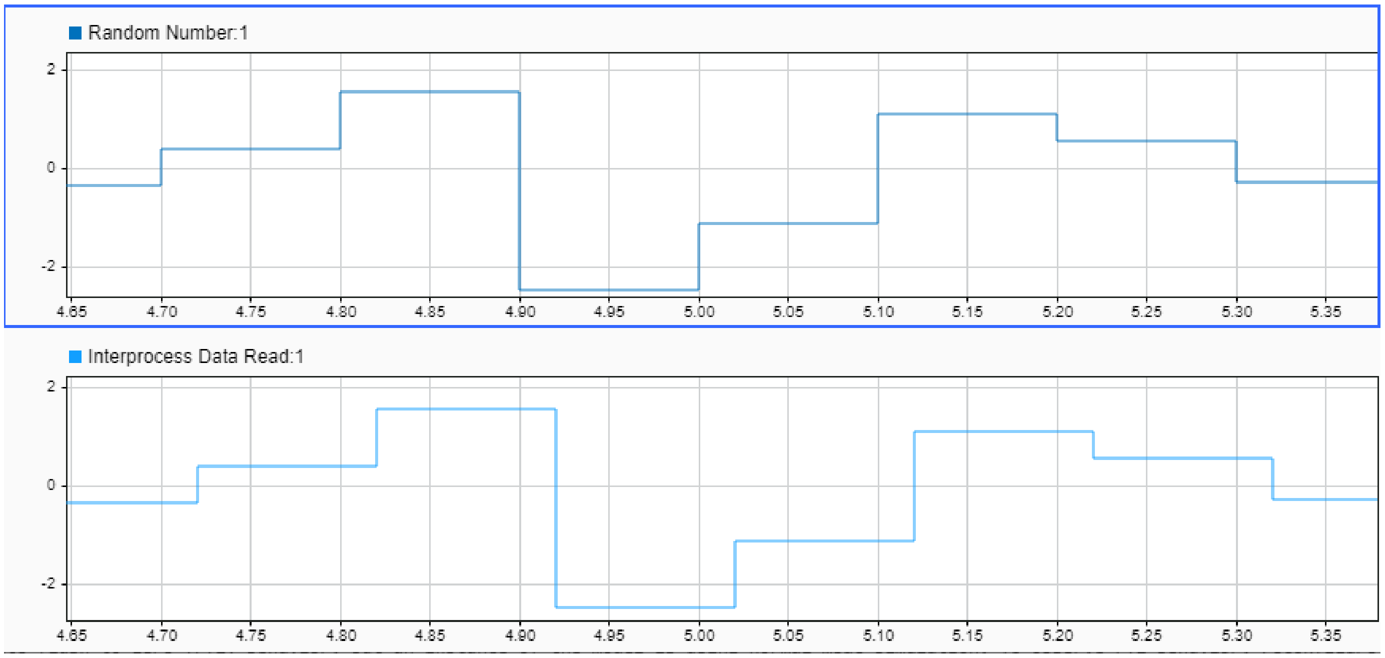
Data Exchange Between CPU and ARM Processors

This example shows how to transfer data between the C2000 processor and the ARM Cortex-M processor. The C2000 processor and ARM coprocessor can share data with the Interprocess Data Channel (SoC Blockset), Interprocess Data Read (SoC Blockset), and Interprocess Data Write (SoC Blockset) blocks. This model shows a timer-driven task on the C2000 processor that transmits a random number over the **Interprocessor Data Channel** block to the ARM processor. When the ARM processor receives the data, the task triggers and reads the data packet. Open the model.

```
open_system("c2000_armcortexm_ipc_top_level_model.slx");
```


Task1
c2000_armcortexm_ipc_c2000_proc_model
D1[0.1]
Out1
datain
event
dataout
Task2Event
Task2
c2000_armcortexm_ipc_arm_proc_
Input1
Input

This figure shows the Simulation Data Inspector view of the data and tasks in the applications. The delay between the CPU and CLA mesurements is due to the CPU completing the transmission before the start of the CLA task.



See Also

Task Manager | Interprocess Data Read | Interprocess Data Channel | Interprocess Data Write | UDP Read | UDP Write

Modeling Control Law Accelerator (CLA) Using Model Reference

The control law accelerator (CLA) is a coprocessor that supports parallel processing. Using the CLA for time-critical tasks frees up the other CPUs to perform other system- and higher-level asynchronous tasks concurrently. The CLA coprocessor also can directly connect to the hardware peripherals, such as analog-to-digital-conversion (ADC) and pulse-width-modulation (PWM), to minimize latency in time-critical tasks.

Configure C2000 Model with CLA Coprocessor

A C2000 processor with a CLA coprocessor consists of a Task Manager or CLA Task Manager block and reference Model blocks containing the tasks for execution on each processor in the system. This model contains sample models of the C2000 processor, and CLA processor each with an event-driven task.

Note Models using CLA requires **long long** mode to be set to `off`.

Configure C2000 Top-Level Model

1. Open a new Simulink model. Save the model as `tif2838xD_top.slx`.
2.
3. Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
4. Go to **Hardware Implementation > Hardware board** and select **TI F2838x**.
5. In the **Hardware board settings> Processing unit**, select `None`.
6. Click **Finish**.

Configure C2000 Processor Model

1. Open a new Simulink model. Save the model as `tif2838xD_c28xCPU1.slx`.
2. Configure the `tif2838xD_c28xCPU1.slx` model to be a C2000 application.
3. Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
4. Go to **Hardware Implementation > Hardware board** and select **TI F2838x**.
5. In the **Hardware board settings> Processing unit**, select `c28xCPU1`.
6. Click **Finish**.
7. Add the event- and timer-driven tasks to the C2000 processor model.
8. In the `tif2838xD_top.slx` model, add CLA Task Manager and Model blocks.
9. Assign the `tif2838xD_c28xCPU1.slx` model to the Model block and then connect the tasks to the Task Manager block.

Configure CLA Coprocessor Model

1. Open a new Simulink model. Save the model as `tif2838xD_CLA.slx`.

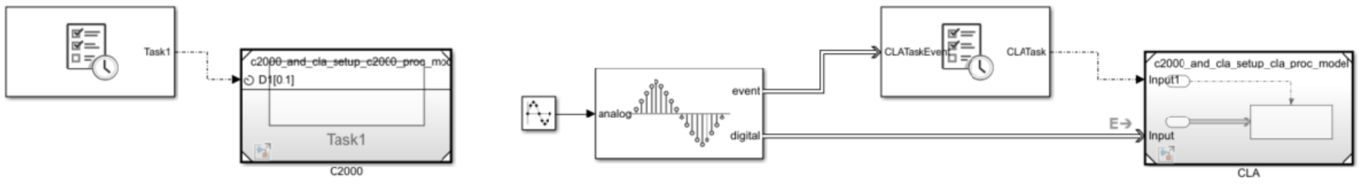
2 Configure the `tif2838xD_CLA.slx` model to be a C2000 application.
3 Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
4 Go to **Hardware Implementation > Hardware board** and select **TI F2838x**.
5 In the **Hardware board settings> Processing unit**, select `CPU1CLA1`.
6 Click **Finish**.
7 Add event-driven tasks to the CLA model.
8 In the `tif2838xD_top.slx` model, add CLA Task Manager and Model blocks.
9 Assign the `tif2838xD_CLA.slx` model to the Model block and then connect the tasks to the CLA Task Manager block.

The CLA coprocessor has restricted regional memory access. In code generation and deployment to the hardware board, the tasks automatically get mapped to the appropriate sections in the CLA ROM. You can inspect the memory sections for inports, outports, signals, states, and internal data assigned using the Code Mappings Editor – C (Embedded Coder) tool in the CLA reference model.

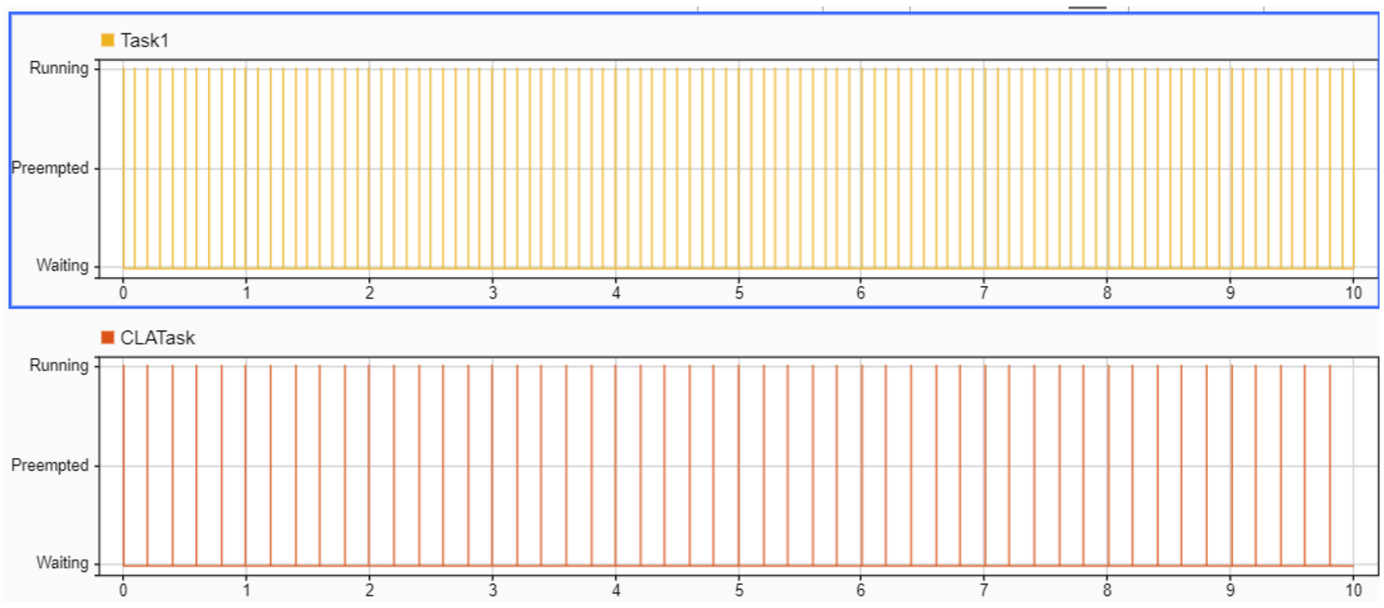
Configure C2000 Model with CLA Coprocessor

This example shows how to create a C2000 model that contains the C2000 processor and the control law accelerator (CLA) coprocessor. The C2000 processor model contains a timer-driven task that executes every 0.1 seconds. The CLA coprocessor model contains a task driven by events from the ADC, modelled by the ADC Read (SoC Blockset) and ADC Interface (SoC Blockset) block pair. Open the model.

```
open_system("c2000_and_cla_setup_top_level_model.slx")
```



Run the model and inspect the task execution in the Simulation Data Inspector. As separate processors, the tasks run without Interfering with each other's execution.



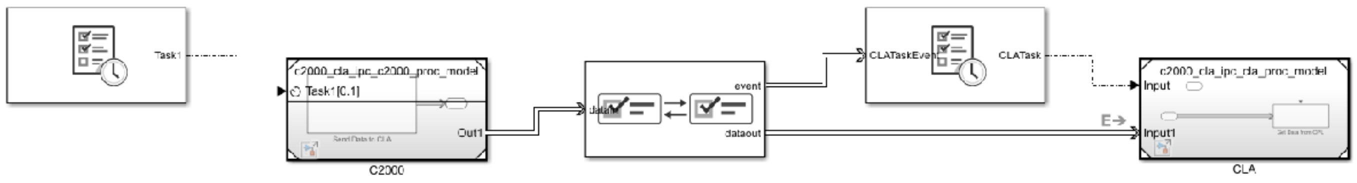
Link Task Execution Using Interprocess Data Channels

Exchange data between the two processors using the Interprocess Data Read, Interprocess Data Channel, and Interprocess Data Write block triplet. The block triplet simulates the exchange of data between the C2000 processor and the CLA coprocessor. When the model generates code, interrupts in the `ePIE` table with data being transferred through shared memory, such as the `LS`*n* `RAM`, replace the block triplet for the specified C2000 processor.

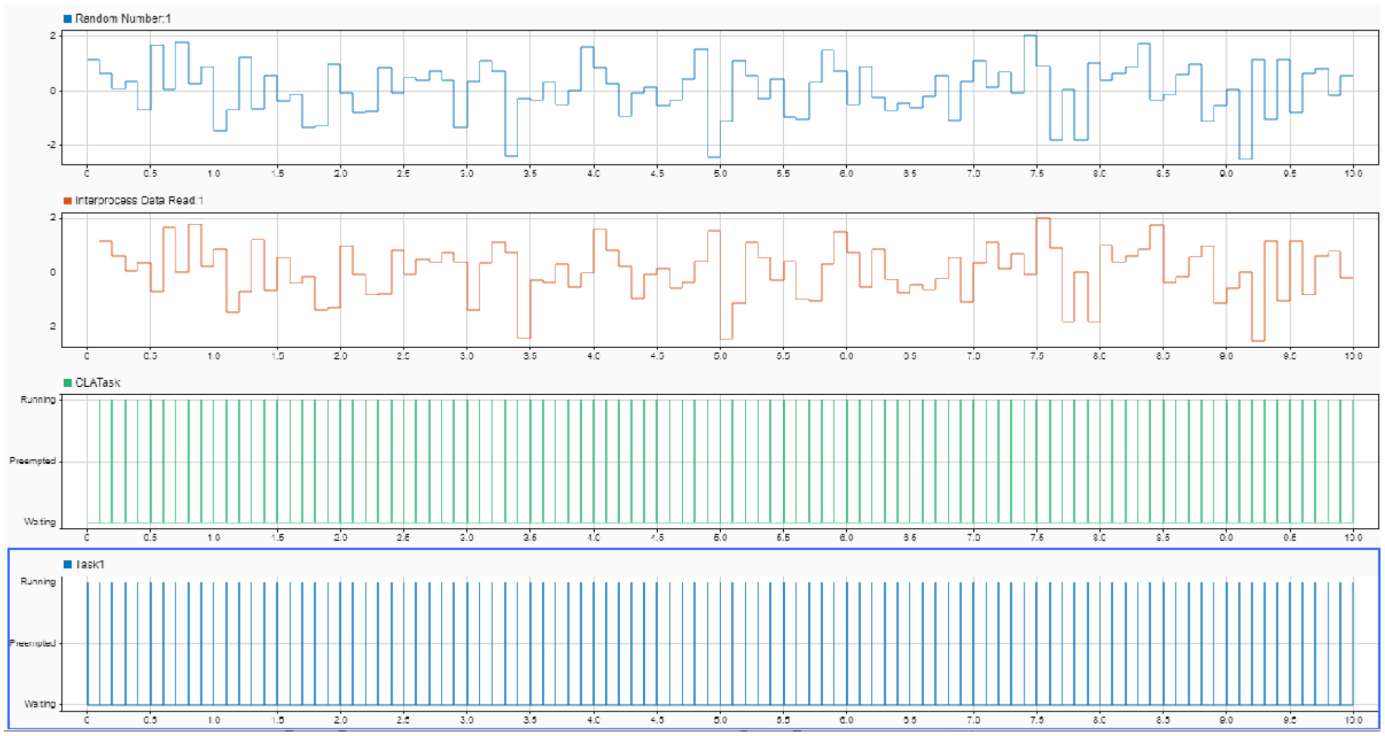
Data Exchange Between CPU and CLA Coprocessor

This example shows the basic connections available to transfer data between the C2000 processor and CLA coprocessors. The C2000 processor and CLA coprocessor can share data with the Interprocess Data Channel (SoC Blockset), Interprocess Data Read (SoC Blockset), Interprocess Data Write (SoC Blockset) blocks. This model shows a timer-driven task, driven by the Task Manager (SoC Blockset) block, on the C2000 processor that transmits a random number over the **Interprocessor Data Channel** block to the CLA coprocessor. When the CLA processor receives the data, the CLATask triggers and reads the data packet.

```
open_system("c2000_cla_ipc_top_level_model.slx");
```



This figure shows the Simulation Data Inspector view of the data and tasks in the applications. The delay between the CPU and CLA measurements are due to CPU completing transmission and the start of the CLA task.



Link Task Execution Using Software Triggers

The CPU can use software interrupts to launch tasks on the CLA coprocessor using the Software Trigger CPU<->CLA block. You can use this block to launch specified tasks from the C2000 on the processor, which frees up the C2000 processor to act in a supervisory role during system execution. The Software Trigger CPU<->CLA can also be used inside the CLA to launch tasks on the CPU.

You can add an Software Trigger CPU<->CLA to the CPU to launch tasks on the CLA by following these steps.

1. Choose an existing task in the CPU reference model.
2. Add the Software Trigger CPU<->CLA block inside the task.

   Note The Software Trigger CPU<->CLA block must be in an atomic subsystem block for periodic tasks.

3. Connect the input to a boolean signal. You can specify the Software Trigger CPU<->CLA block to trigger on a rising edge or when the signal is high.
4. Connect the output to an Outport block.
5. In the Software Trigger CPU<->CLA block, set the **Task number** to the CLA task.
6. In the CPU reference model, select the Outport block that connects the Software Trigger CPU<->CLA block to the top-level soc model.
7. In the **Signal Attributes** panel, refresh the **Data type** choices by choosing `--- Refresh Data Types ---`. Then set **Data type** to `Bus: rteEvent`.
8. Select **Output as nonvirtual bus in parent model**.

9 Click **OK** to apply the changes to the Outport block.

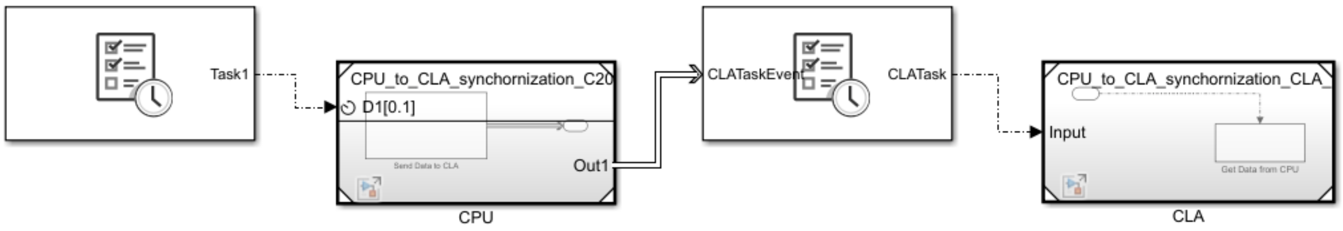
10 In the top-level model, connect the message output from the C2000 reference model to the CLA Task Manager block.

Note The assigned task must match the task number specified in the Software Trigger CPU<->CLA.

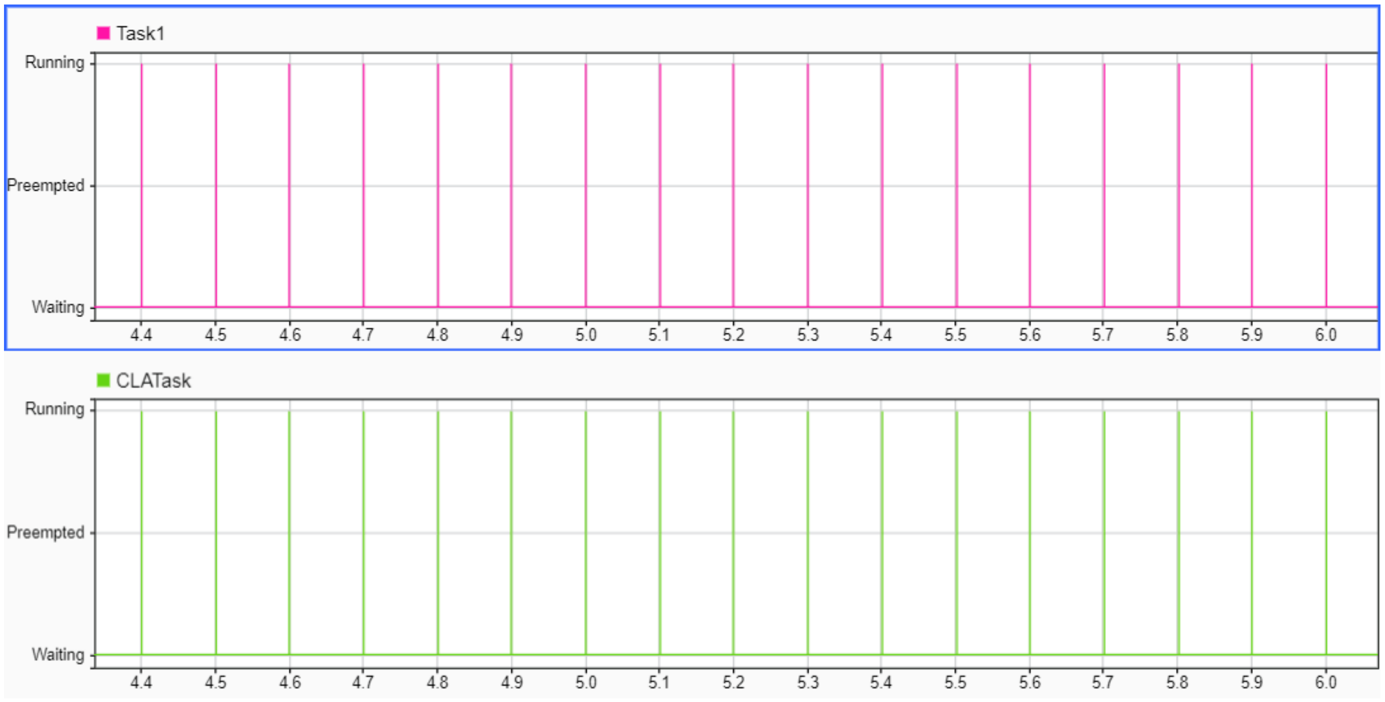
Synchronization Between CPU and CLA Coprocessor

This example shows how to launch tasks on the control law accelerator (CLA) coprocessors from the C2000 processor. The C2000 processor can launch tasks on the CLA coprocessor using the Software Trigger CPU<->CLA block. This model shows a timer-driven task on the C2000 processor that triggers and executes a CLA task on the CLA.

```
open_system("CPU_to_CLA_synchornization_top_level_model");
```



This figure shows the Simulation Data Inspector view of the tasks in the applications where the CPU task, `Task1`, triggers the CLA task, `CLATask`.



See Also

Task Manager | CLA Task Manager | Software Trigger CPU<->CLA | Interprocess Data Read | Interprocess Data Channel | Interprocess Data Write

Related Examples

- “Control Law Accelerator in DC-DC Power Conversion” (SoC Blockset)

Map Simulation Parameters to Peripheral Configuration Tool

SoC Blockset™ interface blocks, such as PWM Interface and ADC Interface, provide a simulation specification of PWM and ADC behavior. To build and deploy the model to a supported hardware board, the simulation parameters in the interface blocks must be equivalently set in the **Hardware Mapping** tool. These sections describe the mapping of parameters between the interface block and respective parameter groups in the **Hardware Mapping** tool.

ADC Parameter Mapping

The ADC Interface block includes several generic parameters that specify the ADC measurement in simulation. When connected to a matching ADC Read block, the **Hardware Mapping** tool provides an equivalent set of hardware parameters for the specified hardware board.

In the **Hardware Mapping** tool, set the **Simulink block** parameter to the ADC Read block that is connected to the ADC Interface block to be mapped. This table shows the relationship between the simulation parameters and the hardware parameters.

ADC Interface Block Parameter	Peripheral Configuration ADC Read Parameter	Conversion
Acquisition time (s)	**SOC*x* Acquisition window (cycles)**	The **SOC*x* Acquisition window (cycles)** parameter is measured in processor clock cycles. This expression shows how to derive the **SOC*x* Acquisition window (cycles)** parameter from the **Acquisition time (s)** parameter. **SOC*x* Acquisition window (cycles)** = **Acquisition time (s)** / (1 / **SYSCLKOUT**) - 1 The **SYSCLKOUT** parameter is defined in the “Model Configuration Parameters for Texas Instruments C2000 Processors” for the given CPU.

ADC Interface Block Parameter	Peripheral Configuration ADC Read Parameter	Conversion
Conversion time (s)	Not HW configurable	The **Conversion time (s)** is implicitly determined by the conversion granularity. For 12-bit conversion: **Conversion time (s)** = 10.5 x **SYSCLKOUT** / **ADCCLKDIV** and for 16-bit conversion: **Conversion time (s)** = 29.5 * **SYSCLKOUT** / 5 The **SYSCLKOUT** and **ADCCLKDIV** parameters defined in the “Model Configuration Parameters for Texas Instruments C2000 Processors” for the given CPU.

PWM Parameter Mapping

The PWM Interface block includes several generic parameters that specify the PWM waveform in simulation. When connected to a matching PWM Write block, the **Hardware Mapping** tool provides an equivalent set of hardware parameters for the specified hardware board.

In the **Hardware Mapping** tool, set the **Simulink block** parameter to the PWM Write block that is connected to the PWM Interface block to be mapped. This table shows the relationship between the simulation parameters and the hardware parameters.

PWM Interface Block Parameters	Peripheral Configuration PWM Write Parameters	Conversion
PWM Period	**Period (clock cycles)**	The **Period (clock cycles)** parameter is measured in processor clock cycles. This expression shows how to derive the **Period (clock cycles)** parameter from the **PWM Period** parameter. **Period (clock cycles) = PWM Period** x `PWMClockVal` `PWMClockVal` is calculated as: `PWMClockVal` = **SYSCLKOUT** / (**EPWMCLKDIV** x **HSPCLKDIV** x **TBCLK**) The **SYSCLKOUT** and **EPWM clock divider (EPWMCLKDIV)** parameters are defined in the “Model Configuration Parameters for Texas Instruments C2000 Processors” for the given CPU.

PWM Interface Block Parameters	Peripheral Configuration PWM Write Parameters	Conversion
Counter Mode	**Counting Mode**	The value of the **Counting Mode** parameter must match the value of the **Counter Mode** parameter for equivalent execution in simulation and on hardware.
Sampling Mode	• **Enable shadow mode** • **Reload CMP*x* register**	The sampling mode applies for the all input parameters in simulation. In the generated code, sampling mode can be configured for the individual comparator registers reload conditions (`counter=zero`, `counter is period`, `counter zero or period`)
Dead time (s)	**Dead band (cycles)**	The **Dead band (cycles)** parameter is given PWM clock cycles. This expression shows how to derive the **Dead band (cycles)** parameter from the **Dead time (s)** parameter. **Dead band (cycles) = Dead time (s)** x *PWMClockVal* *PWMClockVal* is calculated: *PWMClockVal* = **SYSCLKOUT** / (**EPWMCLKDIV** x **HSPCLKDIV** x **TBCLK**) The **SYSCLKOUT** and **EPWM clock divider (EPWMCLKDIV)** parameters are defined in the "Model Configuration Parameters for Texas Instruments C2000 Processors" for the given CPU.
Event trigger mode	• **ADC start of conversion** • **PWM Interrupt**	If the PWM Interface block event signal is connected to an ADC Interface block, then use the **ADC start of conversion** parameter. If the PWM Interface block event signal is connected to a Task Manager block, then use the **PWM Interrupt** parameter.
• **At *`position`* of period** • **At compare *`n`*** • **At compare *`n`* *`direction`* count**	• **Action on counter=zero** • **Action on counter=period** • **Action on counter=CMP*x* on *`direction`* count**	These parameters can be matched effectively one-to-one by using this conversion of parameter values. • `Clear` = `Low` • `Set` = `High` • `Do nothing` = `NoChange` • `Toggle` = `Change`

PWM Interface Block Parameters	Peripheral Configuration PWM Write Parameters	Conversion
Phase (degrees)	• **Enable phase offset** • **Timer phase offset**	The **Timer phase offset** parameter is given in clock cycles and is dependent on the **Enable phase offset** parameter. This expression shows how to derive the **Timer phase offset** parameter from the **Phase (degrees)** parameter. **Timer phase offset** = *PWMClockVal*/(360/**Phase (degrees)** *PWMClockVal* is calculated as: *PWMClockVal* = **SYSCLKOUT/EPWMCLKDIV/HSPCLKDIV/TBCLK** The **SYSCLKOUT** and **EPWM clock divider (EPWMCLKDIV)** parameters are defined in the “Model Configuration Parameters for Texas Instruments C2000 Processors” for the given CPU.

See Also

Hardware Mapping | PWM Interface | ADC Interface

PIL Simulation

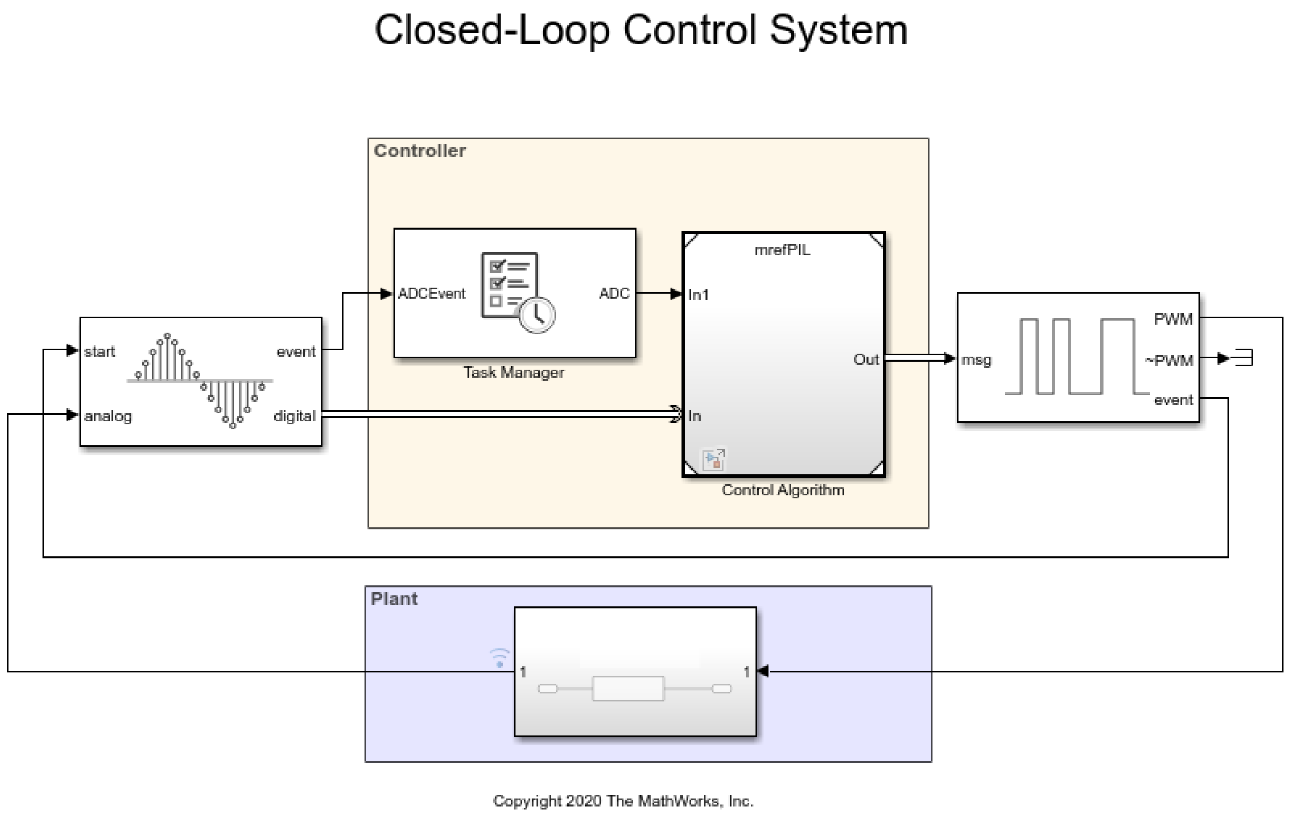
A processor-in-the-loop (PIL) simulation cross-compiles source code on your development computer, and then downloads and runs the object code on the processor in the hardware board. With SoC Blockset and C2000 Microcontroller Blockset features, you can run parts of your model on the supported Texas Instruments hardware board to get direct measurements of algorithm and task execution time from the processor. The measurements can be used to improve the quality of the simulations.

Two types of PIL simulations can be used with the SoC Blockset models and the supported Texas Instruments hardware board: block level simulation and top-model simulation. These sections describe the two types of PIL simulations available when using SoC Blockset models. For general information on PIL simulations, see “SIL and PIL Simulations” (Embedded Coder).

Note PIL simulation with C2000 Microcontroller Blockset is only supported using the `SCI_A` communication channel.

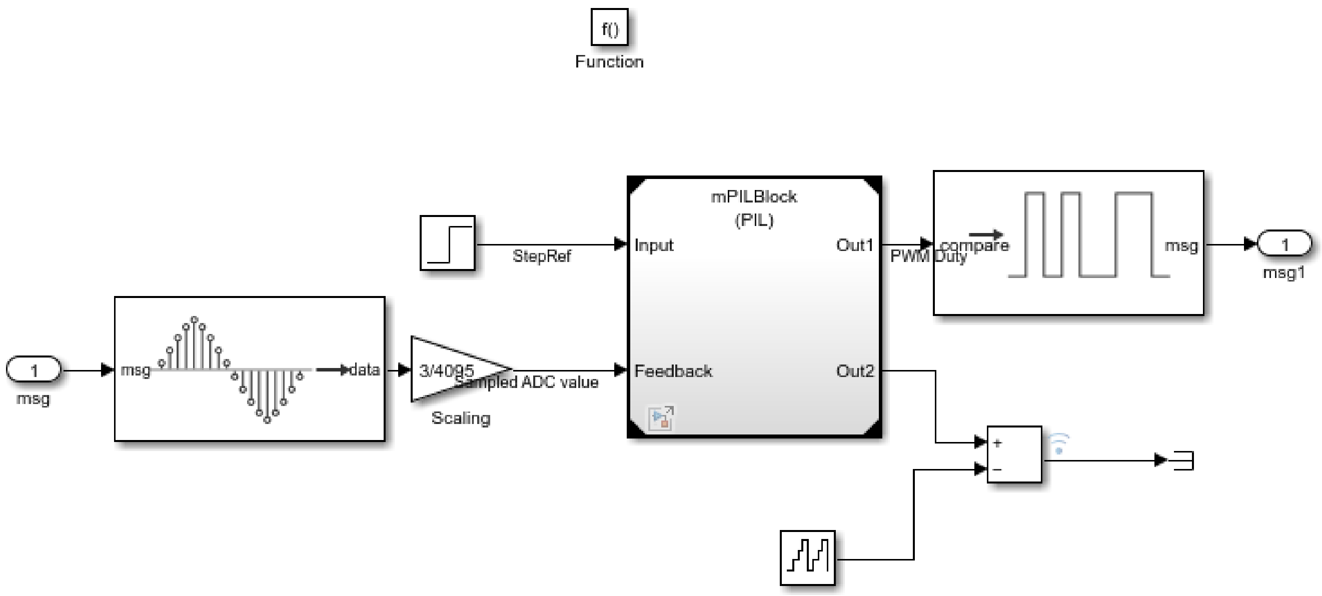
Block-Level PIL Simulation

1 Open an C2000 Microcontroller Blockset model, such as the closed-loop control system shown in this figure.



2 Open the Control Algorithm reference model that represents CPU1 on the hardware board. For more information on selecting CPUs for reference models, see “Hardware Board Settings”. The

control algorithm, mPILBlock, is a separate model used by the reference model block. The mPILBlock reference model connects to the ADC Read and PWM Write blocks but does not contain any driver blocks itself.



3. Select the mPILBlock reference model. On the **Model Block** tab, set **Simulation Mode** to `Processor-in-the-loop (PIL)`.
4. On the **Apps** tab, under **Code Verification, Validation, and Test**, click **SIL/PIL Manager**.
5. On the **SIL/PIL** tab, set **System Under Test** to `Model blocks in SIL/PIL mode`. With the mPILBlock model block configured for PIL operation, code generated for only that reference model block alone and then deployed to the processor on the hardware board.
6. Optionally, generate a PIL task profiling report. On the **SIL/PIL** tab, expand **Settings** and enable **Task Profiling**.
7. On the **SIL/PIL** tab, click **Run SIL/PIL** to run the simulation.

Top-Level PIL Simulation

In the top-level PIL simulation, code is generated for the top-level reference model and then deployed and executed on the connected hardware board. For detailed instructions on configuring a top-level PIL simulation, see “Configure and Run PIL Simulation” (Embedded Coder).

When using a top-level PIL simulation on C2000 Microcontroller Blockset models, the top-level reference model executes on the processor in the hardware board. However, due to the hybrid nature of the PIL setup, the model must meet these limitations.

- All tasks must be timer-driven. Event-driven tasks are not supported.
- Messages cannot be sent to the model block. As a result, driver blocks cannot be used in the models.

See Also

“SIL and PIL Simulations” (Embedded Coder) | “PIL Simulation Sequence” (Embedded Coder) | “Hardware Board Settings” | “Configure and Run PIL Simulation” (Embedded Coder)

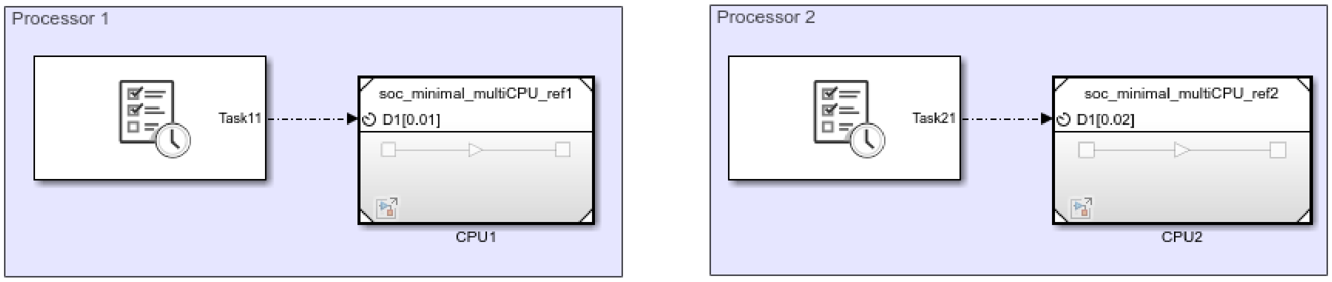
Run Multiprocessor Models in External Mode

In multiprocessor external mode simulation, each processor reference model can be deployed run simultaneously on the processors contained in the SoC or microcontroller. While model run on the separate processors, you can interact with each model to observe signals, tune model parameters, and evaluate the overall behavior of the multiprocessor system when running on the hardware. This section describes the typical workflow configuration used to setup an external mode simulation onto a supported multiprocessor hardware board, such as the `TI Delfino F2837xD`.

Note Hardware boards supporting multiprocessor deployment can be found in the C2000 Microcontroller Blockset.

Process to Run Multiprocessor Model

1. Create or open a multiprocessor SoC model and configure the model for a supported hardware board, such as the `TI Delfino F2837xD`. This figure shows an example of a minimal multiprocessor model. For more information on creating and configuring a multiprocessor model, see “Multiprocessor Execution” (SoC Blockset).



2. Connect each CPU in the hardware board to your host computer. In the `TI Delfino F2837xD`, CPU1 can be connected using the SCIA native port which connects to the USB port on the `TI Delfino F28379D Launch Pad` hardware board. CPU2 can be connected to the host computer using an external FTDI, a serial to USB converter, connect to the SCIB native port on the hardware board. Both the SCIA and SCIB ports are now exposed as the COM ports on the host computer. Different hardware boards will require their own connection setup to expose their own connection ports, one for each processor in your system.

   Note The SCIB ports are mapped to the pins on the hardware boards as follows:

   - `TI Delfino F28379D LaunchPad` – Rx `GPIO19` & Tx `GPIO18b`
   - `TI Delfino F2837xD` – Rx `GPIO11` & Tx `GPIO9`

3. Open one of the processor reference model. In the Simulink toolstrip, on the **System on Chip** tab, click **Hardware Settings** to open the **Configuration Parameter** window.
4. On the **Hardware Implementation** > **Target Hardware Resources** > **External Mode** tab, set the **Communication interface** to `serial(using xcp)`. Set the **Serial port** to match `COM` port previously defined for that processor. Set the **Baudrate** to the maximum supported by the hardware board or else data drop may be observed.

5. Check the **Hardware Implementation** > **Task profiling on processor** > **Show in SDI** to enable **Simulation Data Inspector** logging. Close the **Configuration Parameter** window.
6. In the Simulink toolstrip, on the **System on Chip** tab, click **Configure, Build & Deploy** to launch the **SoC Builder**. For more information on **SoC Builder**, see SoC Builder (SoC Blockset).
7. In the **SoC Builder** tool, on the **Select Build Action** screen, select **Build and load for external mode**. Click **Next**.
8. In the following screen, select the CPUs that will run the external mode models.
9. Complete the remaining steps and on the **Run Application** screen, click **Load and Run** to launch the external mode models on the processors. The models automatically start running in external mode.
10. To stop the external mode execution, in the Simulink toolstrip, on the **System on Chip** tab, click **Stop**.

Note You must stop all models. Stopping only one model while leaving others running can produce undefined behavior.

View External Mode Simulation Data

During and after running an external mode simulation on multiple processors, the tasks and signals can be viewed in the **Simulation Data Inspector**. Each processor records an independent run in the Simulation Data Inspector and contains all the tasks and signals that executed on that processor. Since the external mode is launched by the **SoC Builder**, all the runs for the separate processors share a common time, allowing comparison of the processor runs to each other to see the overall behavior of the software portion of your system on the SoC hardware.

Note External mode, profiler, and data logging use the same communication channel. To prevent data drops and gaps, do not run external mode simulations with profiling or data logging enabled, and vice-versa.

See Also

SoC Builder (SoC Blockset) | “Multiprocessor Execution” (SoC Blockset) | **Simulation Data Inspector**

More About

- “External Mode Simulations for Parameter Tuning, Signal Monitoring, and Code Execution Profiling” (Simulink Coder)

Hardware Mapping

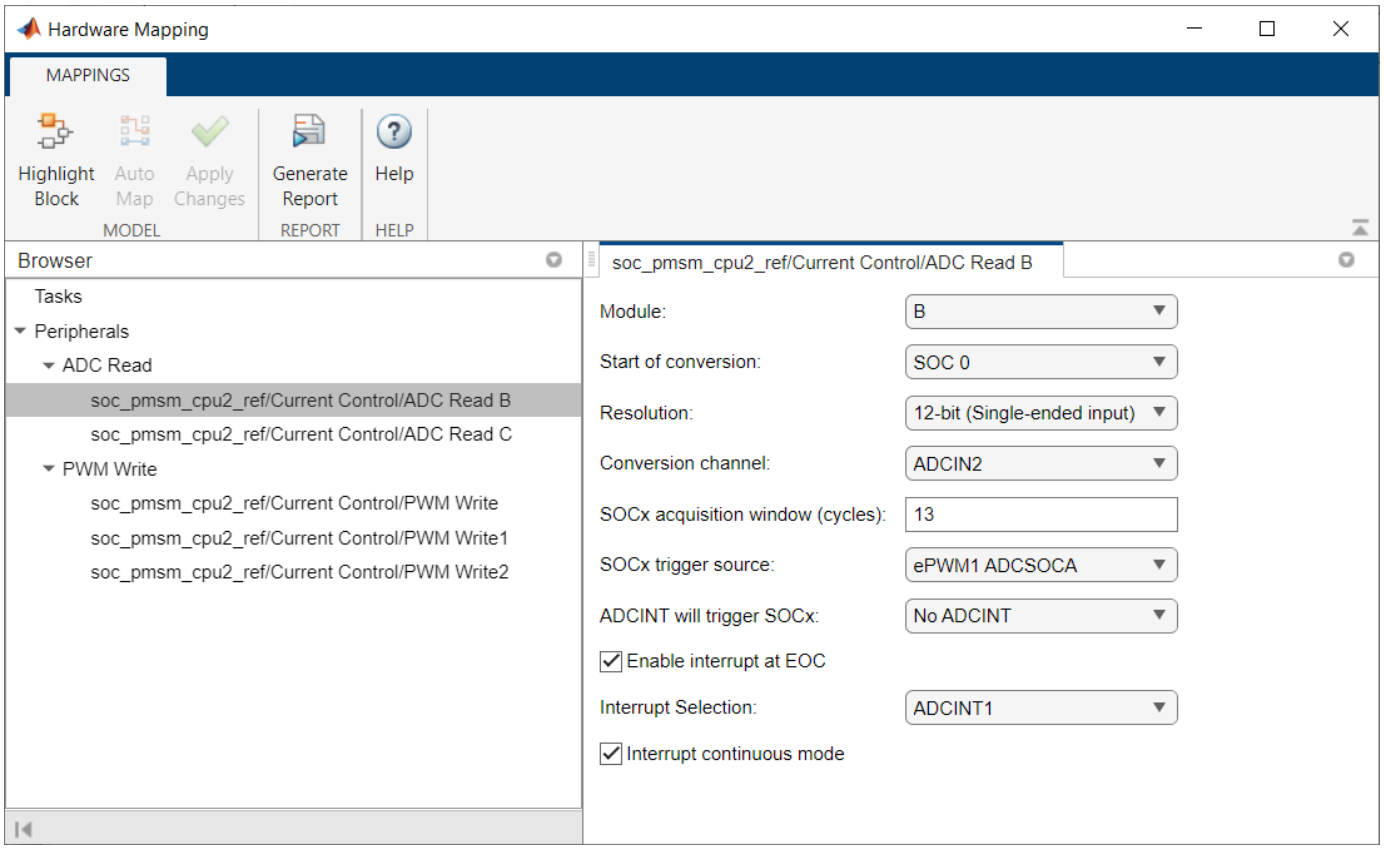
Map tasks and peripherals in a model to hardware board configurations

Description

The **Hardware Mapping** tool allows you to configure the software tasks and peripherals on the selected hardware board.

In this tool, you can map the tasks in your software model to the available event sources and interrupts:

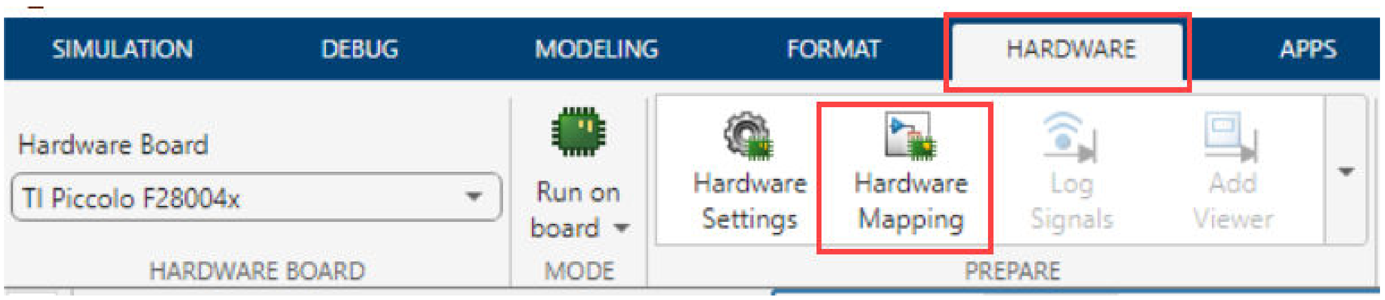
- Manually select the task in **Browser** > **Tasks** > ***task name***. Select the desired event or interrupt. source. Click the **Apply Changes** button in the toolstrip.
- Automatically by clicking the **Auto Map** button on the toolstrip.


Hardware Mapping
MAPPINGS
Highlight Block
Auto Map
Apply Changes
Generate Report
Help
MODEL
REPORT
HELP
Browser
Tasks
Peripherals
ADC Read
soc_pmsm_cpu2_ref/Current Control/ADC Read B
soc_pmsm_cpu2_ref/Current Control/ADC Read C
PWM Write
soc_pmsm_cpu2_ref/Current Control/PWM Write
soc_pmsm_cpu2_ref/Current Control/PWM Write1
soc_pmsm_cpu2_ref/Current Control/PWM Write2
soc_pmsm_cpu2_ref/Current Control/ADC Read B

Module:	B
Start of conversion:	SOC 0
Resolution:	12-bit (Single-ended input)
Conversion channel:	ADCIN2
SOCx acquisition window (cycles):	13
SOCx trigger source:	ePWM1 ADCSOCA
ADCINT will trigger SOCx:	No ADCINT
✓ Enable interrupt at EOC	
Interrupt Selection:	ADCINT1
✓ Interrupt continuous mode	

Open the Hardware Mapping

- Simulink Toolstrip: On the **Hardware** tab, click **Hardware Mapping**



- A screen within the **SoC Builder** app

Version History

Introduced in R2022b

See Also

Hardware Interrupt | “Configure Interrupts and Events Using Hardware Mapping” on page 1-147

SoC Builder

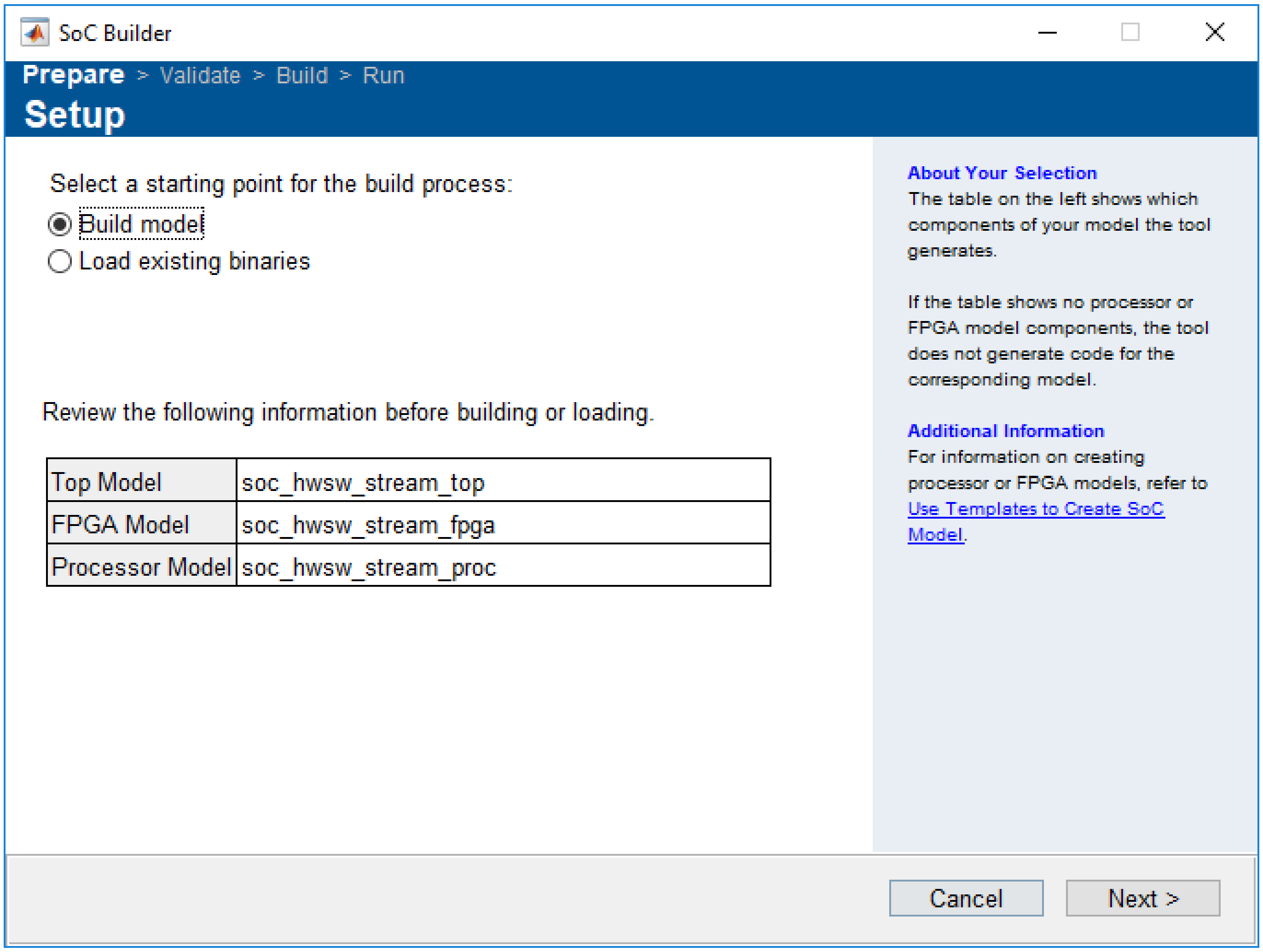
Build, load, and execute SoC model on SoC, FPGA, and MCU boards

Description

The **SoC Builder** tool steps through the various stages for building and executing an SoC model on an SoC, FPGA, or MCU board.

Using this tool, you can:

- Review the model information provided to the tool.
- Choose between different build actions.
- Set up a folder to store all generated files.
- Map model tasks to interrupt service routines.
- Configure the peripheral register settings.
- Review the memory map and edit it if needed.
- Validate that the model has all required components for generating a programming file.
- Build the model using Xilinx® Vivado®, Intel® Quartus®, Texas Instruments Code Composer Studio tool families.
- Configure the Ethernet connectivity.
- Load the programming file to your board.
- Run the application.


SoC Builder

Prepare > Validate > Build > Run

Setup

Select a starting point for the build process:

[✓ Build model]
Load existing binaries

Review the following information before building or loading.

Top Model	soc_hwsw_stream_top
FPGA Model	soc_hwsw_stream_fpga
Processor Model	soc_hwsw_stream_proc

About Your Selection
The table on the left shows which components of your model the tool generates.

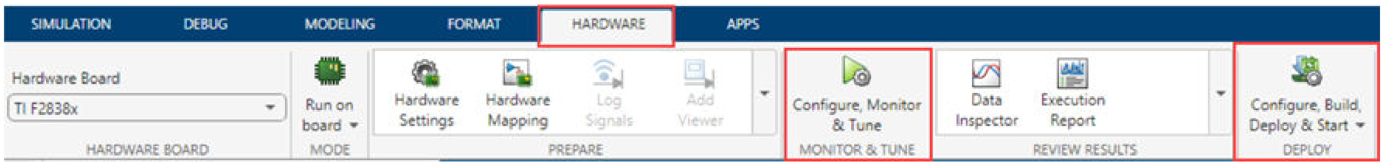
If the table shows no processor or FPGA model components, the tool does not generate code for the corresponding model.

Additional Information
For information on creating processor or FPGA models, refer to Use Templates to Create SoC Model.

Cancel | Next >

Open the SoC Builder

- Simulink Toolstrip: On the **Hardware** tab, click **Configure, Build, Deploy & Start** or **Configure, Monitor & Tune**.



- MATLAB command prompt: Enter `socBuilder('modelname')`.

Note If the **Configure, Build, Deploy & Start** or **Configure, Monitor & Tune** is not visible, on the **Hardware** tab, ensure that you have set the **Processing Unit** to **None** for the selected hardware board in the Configuration Parameters.

Programmatic Use

`socBuilder('modelname')` opens SoC Builder and loads the specified model into the tool.

Version History

Introduced in R2019a

See Also

Tools
Hardware Mapping

Objects
`socModelBuilder`

Hardware Mapping Peripherals for Texas Instruments C2000 Processors Properties

Select the configurations for the peripherals in the model deployed to the hardware board

Description

The peripheral configurations for devices listed in the **Hardware Mapping** tool for Texas Instruments C2000 processors appear in the selected block in the **Browser** > **Peripherals**. This table shows the association between the driver block, simulation interface block, and **Hardware Mapping** tool hardware configuration.

Driver Block	Interface Block	Hardware Configuration
ADC Read	ADC Interface	“ADC” (SoC Blockset)
PWM Write	PWM Interface	“PWM” (SoC Blockset)

Properties

ADC

`Module` — Hardware ADC Module
`A` (default) | `B` | `C` | `D`

Select the ADC module `A` through `D` on the hardware board.

`Start of conversion` — Start of conversion trigger
`SOC0` (default) | `SOC0` | ... | `SOC15`

Identify the start-of-conversion trigger by number.

`Resolution` — Resolution of digital conversion
`12-bit (Single-ended input)` (default) | `16-bit (Differential inputs)`

Select the resolution of the digital conversion output.

`Conversion channel` — Input channel to apply ADC
`Internal` (default) | `Undefined` | `Interrupt name`

Select the input channel to which this ADC conversion applies.

`SOCx Acqusition window (cycles)` — Length of ADC acquisition period
positive scalar integer

Define the length of the acquisition period in ADC clock cycles. The value of this parameter depends on the `SYSCLK` and the minimum ADC sample time.

`SOCx Trigger source` — SoC trigger source
`Software` | `Timer` *x* `TINT`*x*`n` | `GPIO ADCEXTSOC` | `ePWM`*x* `ADCSOCA`

Select the event source that triggers the start of the conversion.

ADCINT will trigger SOCx — Use ADCINT interrupt to trigger start of conversion
No ADCINT (default) | ADCINT1 | ADCINT2

At the end of conversion, use the ADCINT1 or ADCINT2 interrupt to trigger a start of conversion. This loop creates a continuous sequence of conversions. The default selection, No ADCINT disables this parameter. To set the interrupt, select the Post interrupt at EOC trigger option, and choose the appropriate interrupt.

Enable interrupt at EOC — Enable post interrupts when the ADC triggers end of conversion pulses
false (default) | true

Enable post interrupts when the ADC triggers EOC pulses. When you select this option, the dialog box displays the **Interrupt selection** and **Interrupt continuous mode** options.

Interrupt selection — ADC interrupt selection
ADCINT1 (default) | ADCINT2 | ADCINT3 | ADCINT4

Select which ADCINT# interrupt the ADC posts to after triggering an EOC pulse.

Interrupt continuous mode — Generate new EOC signal overriding previous interrupt flag status
false (default) | true

When the ADC generates an end of conversion (EOC) signal, generate an ADCINT# interrupt, whether the previous interrupt flag has been acknowledged or not.

PWM

PWM Module — Indicates which ePWM module to use
ePWM1 (default) | ePWM2 | ... | ePWM*x*

Select the appropriate ePWM module, ePWM*x*, where x is a positive integer.

High speed clock divider — High speed time base clock prescaler divider HSPCLKDIV
1 (default) | 2 | 4 | 6 | 8 | 10 | 12 | 14

Set the high speed time base clock prescaler divider, HSPCLKDIV.

Timerbase clock divider — Time base clock TBCLK prescaler divider corresponding to CLKDIV
1 (default) | 2 | 4 | 8 | 16 | 32 | 64 | 128

Use the Time base clock, TBCLK, prescaler divider, CLKDIV, and the high speed time base clock, HSPCLKDIV, prescaler divider, HSPCLKDIV, to configure the Time-base clock speed, TBCLK, for the ePWM module. Calculate TBCLK using this equation: TBCLK = PWM clock/(HSPCLKDIV * CLKDIV).

For example, the default values of both CLKDIV and HSPCLKDIV are 1, and the default frequency of PWM clock is 200 MHz, so: TBCLK in Hz = 200 MHz/(1 * 1) = 200 MHz TBCLK in seconds = 1/TBCLK in Hz = 1/200 MHz = 0.005 µs.

Period (clock cycles) — Period of ePWM counter
1 (default) | 2 | 4 | 8 | 16 | 32 | 64 | 128

Set the period of the ePWM counter waveform.

The timer period is in clock cycles:

Count Mode	Calculation	Example
`Up or down`	The value entered in clock cycles is used to calculate time-base period, `TBPRD`, for the `ePWM` timer register. The period of the `ePWM` timer is `TCTR = (TBPRD + 1) * TBCLK`, where `TCTR` is the timer period in seconds, and `TBCLK` is the time-base clock.	For `ePWM` clock, `EPWMCLK`, frequency = 200 MHz, and `TBCLK` = 5 ns. `EPWMCLK` will be equal to `SYSCLKOUT` or `SYSCLKOUT`/2 depending on the `ePWM` clock divider, `EPWMCLKDIV`, parameter setting. When the timer period is entered in clock cycles `TBPRD` = 9999, and the `ePWM` timer period is calculated as `TCTR` = 50 µs. For the default action settings on the `ePWMx` tab, the `ePWM` period = 50 µs.
`Up-down`	The value entered in clock cycles is used to calculate the time-base period, `TBPRD`, for the `ePWM` timer register. The period of the `ePWM` timer is `TCTR = 2 * TBPRD * TBCLK`, where `TCTR` is the timer period in seconds and `TBCLK` is the time-base clock.	For `EPWMCLK` frequency = 200 MHz and `TBCLK` = 5 ns. When the timer period is entered in clock cycles, `TBPRD` = 10000, and the `ePWM` timer period is calculated as `TCTR` = 100 µs. For the default action settings on the `ePWMx` tab, the `ePWM` period = 100 µs.

The initial duty cycle of the waveform from the time the PWM peripheral starts operation until the ePWM input port receives a new value for the duty cycle is Timer period / 2.

`Initialize CMPx count (clock cycles)` — Initialize the CMP*x* count
`0` (default) | positive integer

Set the initial count value of the comparator in clock cycles.

`Enable phase offset` — Enable the timer phase offset
false (default) | true

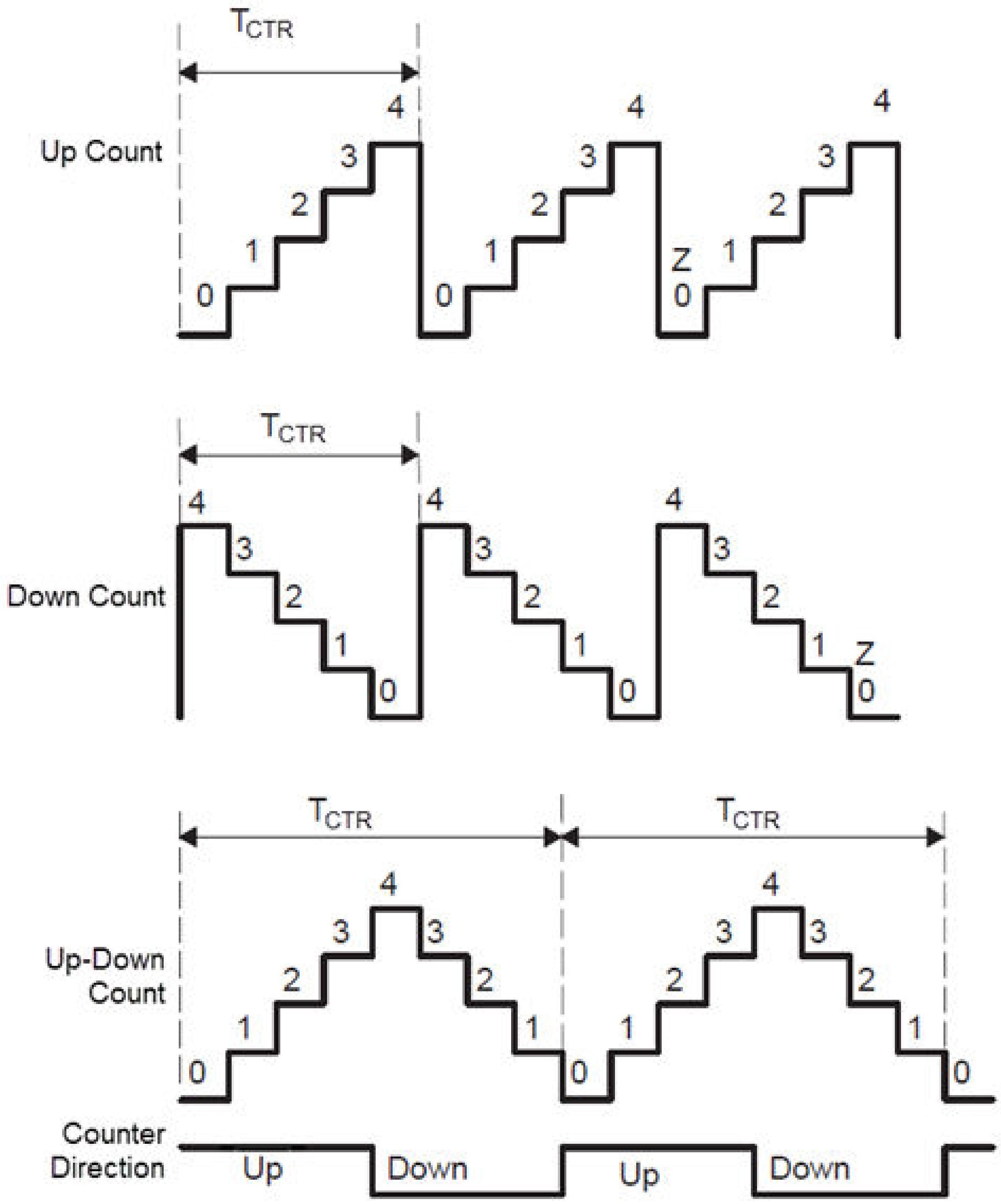
Enables to provide a timer phase offset value.

`Timer phase offset` — Timer phase offset
`0` (default) | integer between `0` and `65535`

The specified offset value is loaded in the time base counter on a synchronization event. Enter the phase offset value, `TBPHS`, in `TBCLK` cycles from 0 to 65535.

`Count mode` — Indicates counting mode of ePWM counter
`Up-Down` (default) | `Down` | `Up`

Specify the counting mode of the PWM internal counter. This figure shows three counting waveforms.


T_{CTR}
Up Count
0
1
2
3
4
Z
T_{CTR}
Down Count
4
3
2
1
0
Z
T_{CTR}
T_{CTR}
Up-Down Count
0
1
2
3
4
3
2
1
0
Counter Direction
Up
Down
Up
Down

`Action on counter=zero` — Behavior of action qualifier (AQ) submodule at zero count
`Do nothing` (default) | `Clear` | `Set` | `Toggle`

This group determines the behavior of the action qualifier (AQ) submodule. The AQ module determines which events are converted into one of the various action types, producing the required switched waveforms of the `ePWMA` circuit. The `ePWMB` always generates a complement signal of `ePWMA`.

`Action on counter=period` — Behavior of action qualifier (AQ) submodule at period count
`Do nothing` (default) | `Clear` | `Set` | `Toggle`

This group determines the behavior of the Action Qualifier (AQ) submodule. The AQ module determines which events are converted into one of the various action types, producing the required switched waveforms of the `ePWMA` circuit. The `ePWMB` always generates a complement signal of `ePWMA`.

`Action on counter=CMP`*`x`*` on `*`direction`*` count` — Behavior of Action Qualifier (AQ) submodule for the comparator (CMP) on for the given direction count
`Clear` (default) | `Do nothing` | `Set` | `Toggle`

This group determines the behavior of the action qualifier (AQ) submodule. The AQ module determines which events are converted into one of the various action types, producing the required switched waveforms of the `ePWMA` circuit. The `ePWMB` always generates a complement signal of `ePWMA`.

`Enable shadow mode` — Enable the shadow mode
`Disable` (default) | `Enable`

When shadow mode is not enabled, the `CMPA` register refreshes immediately. Provide different reload mode for `CMPA` register.

`Reload CMP`*`x`*` register` — Time at which the counter period is reset
`Counter equals to zero (CTR=Zero)` (default) | `Counter equals to period (CTR=PRD)` | `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` | `Freeze`

The time when the counter period resets based on the following condition:

- `Counter equals to zero (CTR=Zero)` – Refreshes the counter period when the value of the counter is 0.
- `Counter equals to period (CTR=PRD)` – Refreshes the counter period when the value of the counter is period.
- `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` – Refreshes the counter period when the value of the counter is 0 or period.
- `Freeze` – Refreshes the counter period when the value of the counter is freeze.

`ADC Start of conversion for ePWM module` — Trigger condition for an ADC start of the conversion event
`Counter equals to zero (CTR=Zero)` (default) | `Counter equals to period (CTR=PRD)` | `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` | `Disable` | `Counter is `*`direction`*` and equal to CMP`*`x`*

This parameter specifies the counter match condition that triggers an ADC start of the conversion event. The choices are:

- `Counter equals to zero (CTR=Zero)` – Triggers an ADC start of the conversion event when the `ePWM` counter reaches 0.
- `Counter equals to period (CTR=PRD)` – Triggers an ADC start of the conversion event when the `ePWM` counter reaches the period value.
- `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` – Triggers an ADC start of the conversion event when the time base counter, `TBCTR`, reaches zero or when the time base counter reaches the period, `TBCTR = TBPRD`.
- `Disable` – Disable ADC start of conversion event.
- `Counter is` *`direction`* `and equal to CMP`*`x`* – Triggers an ADC start of the conversion event when the counter equals the specified comparator and the counter *`direction`* is either `incrementing` or `decrementing`.

`ePWM interrupt` — Generate ISR for ePWM
`Disable` (default) | `Counter equals to zero (CTR=Zero)` | `Counter equals to period (CTR=PRD)` | `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` | `Counter is` *`direction`* `and equal to CMP`*`x`*

This parameter registers that an interrupt occurs for the specified event and generates interrupt service routine (ISR) code to be used by the Task Manager. The choices are:

- `Counter equals to zero (CTR=Zero)` – Generates an ISR for when the `ePWM` counter reaches 0.
- `Counter equals to period (CTR=PRD)` – Generates an ISR for when the `ePWM` counter reaches the period value.
- `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` – Generates an ISR for when the time base counter, `TBCTR`, reaches zero or when the time base counter reaches the period, `TBCTR = TBPRD`.
- `Disable` – Disable ISR generation.
- `Counter is` *`direction`* `and equal to CMP`*`x`* – Generates an ISR for when the counter equals the specified comparator and the counter *`direction`* is either `incrementing` or `decrementing`.

`Dead band (cycles)` — Enables the phase offset
`0` (default) | integer between `0` and `65535`

This parameter specifies the deadband delay for rising edge and falling edge in time-base clock cycles.

Version History

Introduced in R2022b

Interprocess Data Communication via Dedicated Hardware Peripheral

A variety of microcontroller units (MCUs) and SoCs provide dedicated hardware peripherals to enable processes executing on separate processors to communicate. The dedicated hardware connection eliminates the need to develop conventional channels through shared memory or through peripheral buses. Dedicated interprocess data communication in hardware is used in embedded MCUs that either support or do not support an operating system (OS). Without an OS, the process occupies the entirety of the processor resources. In this case, multiprocess systems require distribution across multiple processors within the single MCU. For example, the F2838xD family of processors from Texas Instruments contains a pair of interprocessor communication (IPC) peripherals that directly connect the C28 CPUs. For more information on the F2838xD processors and their IPC peripherals, see the Texas Instruments website TMS320F2838x Microcontrollers with Connectivity Manager.

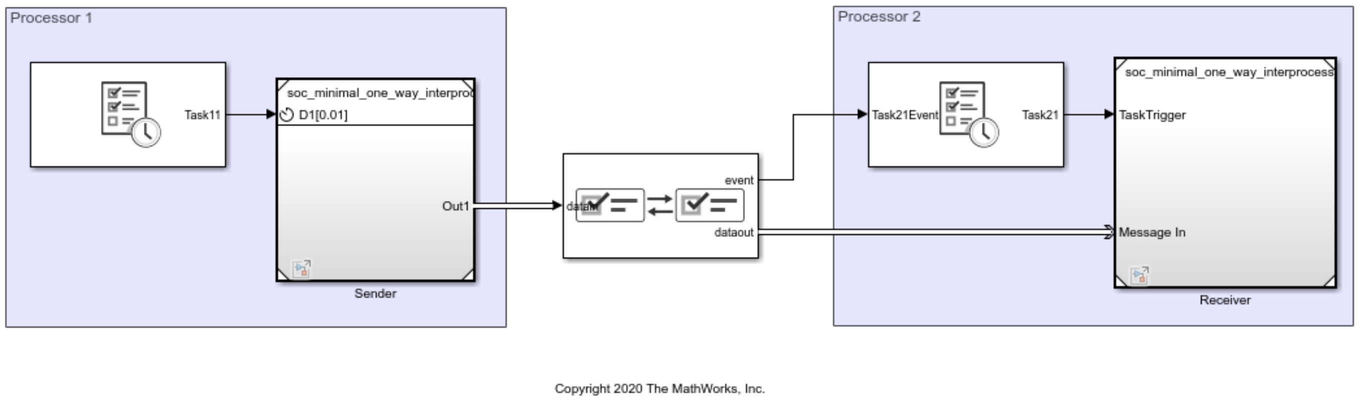
The blockset Interprocess Data Read, Interprocess Data Channel, and Interprocess Data Write blocks simulate communication between bare metal processes executing on separate processors. To create a mono direction data channel between two processors, add an Interprocess Data Write block into the processor reference model that sends data. Next, add an Interprocess Data Read block into the processor reference model that receives data. In each model, expose the event ports to the top-level model using the Outport and Inport blocks, respectively. Finally, connect the event ports in the top-level model using the Interprocess Data Channel block.

If the SoC models are built for a supported processor, such as those in the F2838xD family of processors, code is automatically generated for the hardware IPC peripherals.

One Way Interprocess Communication

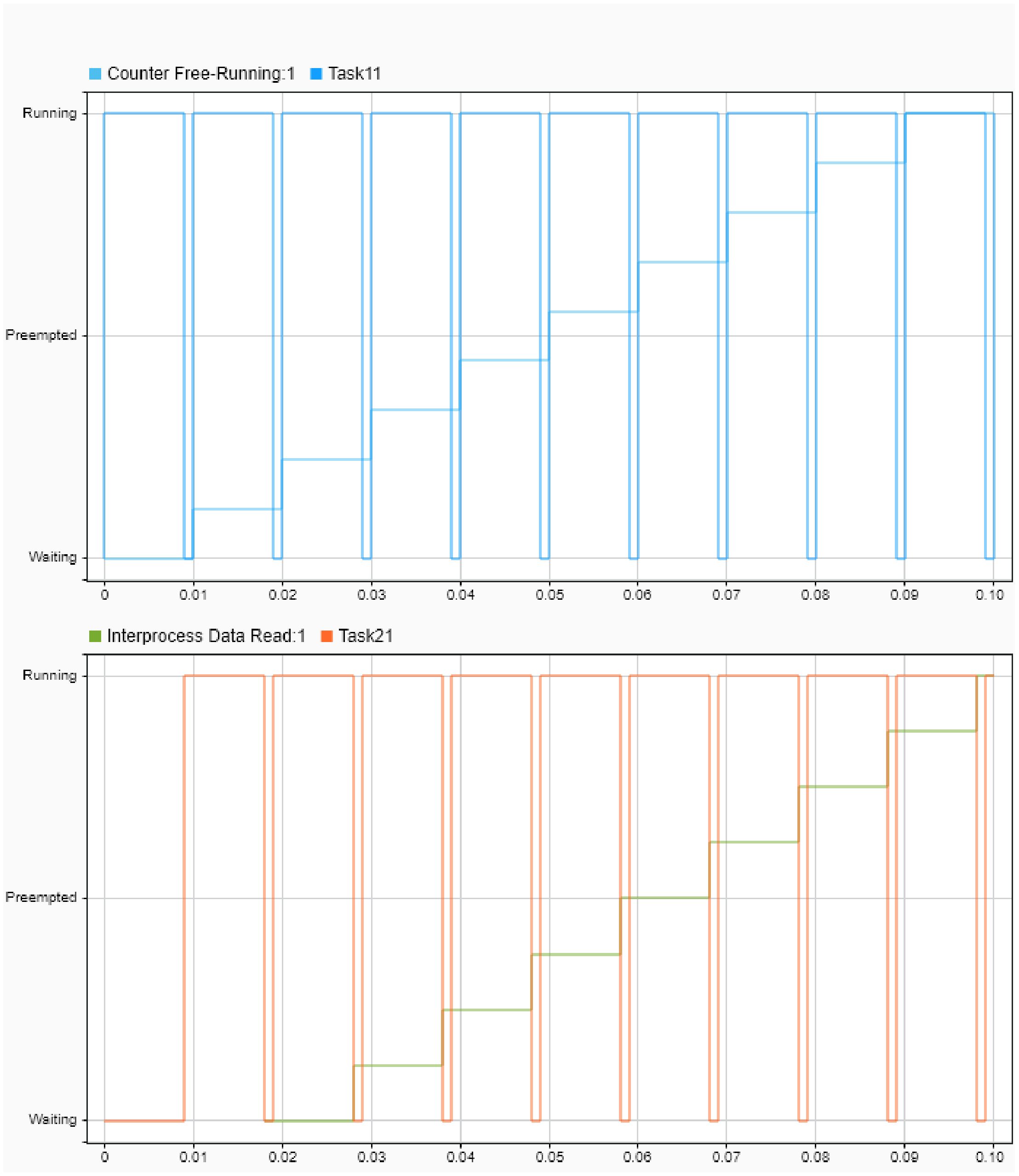
This example shows one-way interprocess data communication between two bare metal processors.

An algorithm in Processor1 sends a data message, using the Interprocess Data Write block, to the Interprocess Data Channel block at a 0.01 second interval. Processor2 two receives and processes the data messages asynchronously, using the Interprocess Data Read block.



Results

In the Simulation tab, click Run. When the simulation completes, open the Simulation Data Inspector to view the resulting signals and tasks. From the graphs, Processor1 sends the data value at the completion of the first task, Task11, instance. The data then gets received by Processor2, triggering the event driven task, Task21. At the completion of Task21 instance, the final value gets emitted in Processor2, potentially for additional processing by other tasks.

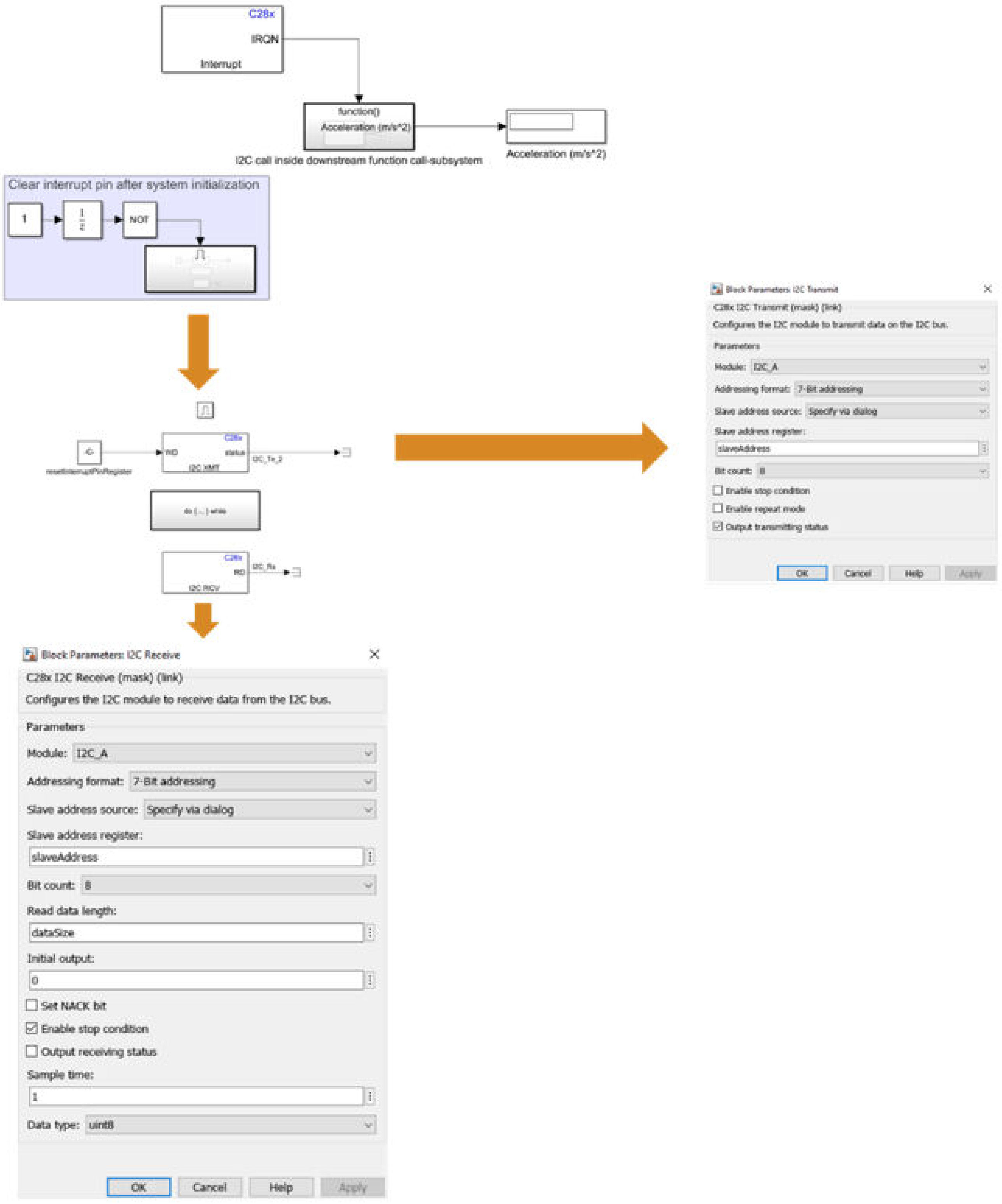


See Also

Interprocess Data Read | Interprocess Data Write | Task Manager | Interprocess Data Channel

Workaround to reset interrupt pin of sensors supporting latched interrupts

This topic helps you with the workaround to reset the interrupt pin of sensors which support latched interrupts. Sensor supporting latched interrupts provide register to reset the interrupt pin. The interrupt pin has to be reset after the system initialization. This ensures occurrence of continuous interrupts from the sensor. Use a model as shown in the image below to clear the interrupt pin of sensors that support latched interrupts.



Block Parameters: I2C Transmit

C28x I2C Transmit (mask) (link)

Configures the I2C module to transmit data on the I2C bus.

Parameters

Module: I2C_A

Addressing format: 7-Bit addressing

Slave address source: Specify via dialog

Slave address register:

slaveAddress

Bit count: 8

☐ Enable stop condition

☐ Enable repeat mode

☑ Output transmitting status

Block Parameters: I2C Receive

C28x I2C Receive (mask) (link)

Configures the I2C module to receive data from the I2C bus.

Parameters

Module: I2C_A

Addressing format: 7-Bit addressing

Slave address source: Specify via dialog

Slave address register:

slaveAddress

Bit count: 8

Read data length:

dataSize

Initial output:

0

☐ Set NACK bit

☑ Enable stop condition

☐ Output receiving status

Sample time:

1

Data type: uint8

The following table shows the slave address, reset interrupt pin register address, and data size for different sensors.

Sensor	Slave address	Reset interrupt pin register address	Data size (uint8)
ADXL345	0x53 (default) | 0x1D	0x32	6
LIS3DH	0x18 (default) | 0x19	0xA8	6

2

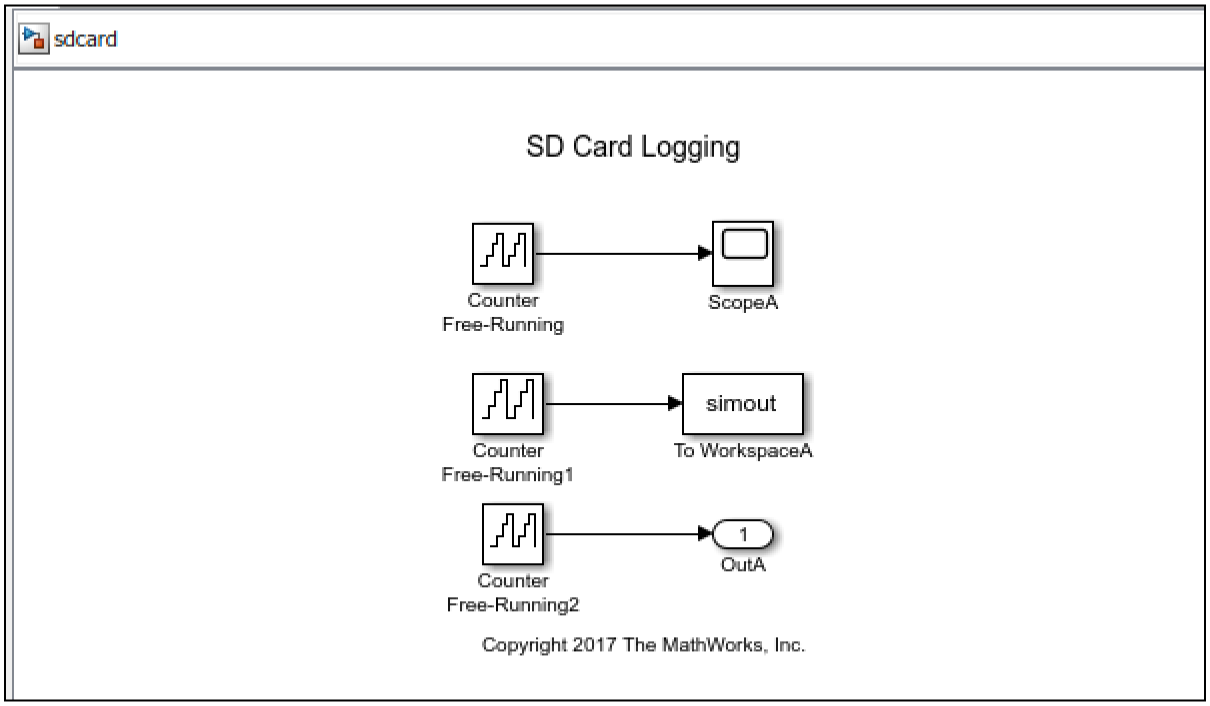
MAT-File Logging on SD Card

Log Signals on an SD Card

You can use SD card logging to save signals from Simulink models on an SD card mounted on a Texas Instruments C2000 hardware. The signals from these models are saved as data points in MAT-files. With the data you log, you can apply fault analysis, search for transient behavior, or analyze sensor data collected over a long period. The data points can be saved in: `Structure`, `Structure with time`, or `Array` format. Simulink supports logging signals on the target hardware from only these three blocks:

- Scope
- To Workspace
- Outport

This topic is for Embedded Coder Support Package for Texas Instruments C2000 Processors. This uses Embedded Coder



Without MAT-file logging on SD card, you can only log and analyze the data on the target hardware through External mode or by sending the signal to a computer (using Serial) and storing it in MATLAB. These mechanisms need an active connection between a computer and the target hardware when logging data. With SD card logging, you can log data without any need to connect the target hardware to a computer. Other advantages of logging data to SD card over other mechanisms are:

- The ability to collect data over a long duration for analysis.
- The ability to store the data in a well-structured MAT-file, including timestamp information.

Note

- The SD card should be formatted with FAT32 format to support the logging from C2000 processors.

- MAT-File Logging on SD Card does not support PIL (Processor in the loop) Mode simulation.
- SD card logging does not support F2802x and F281x processors.

Before you start to save the signals from Simulink models on an SD card, complete the steps listed in “Prerequisites for Logging Signals” on page 2-4.

1. “Configure Board Parameters and Enable MAT-File Logging” on page 2-5: To save MAT-files on an SD card, the **MAT-file logging** option in the **Configuration Parameters** dialog box must be selected. Also, the target hardware parameters must be specified.
2. “Configure Board Parameters and Enable MAT-File Logging” on page 2-5: SD card logging is supported in models containing To Workspace, Scope, or Outport blocks. You must specify the values for several block parameters.
3. “Run Model on Target Hardware” on page 2-15: Simulink deploys code and logs signals on the SD card. These signals are saved as MAT-files on the target hardware.
4. “Import MAT-Files into MATLAB” on page 2-16: After logging is complete, you can open MAT-files in MATLAB and use them for further analysis.

See Also

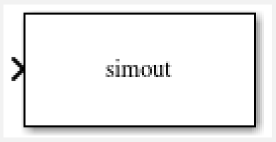
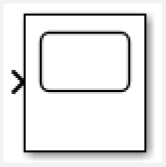
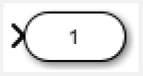
Related Examples

- “MAT-file Logging on SD Card for Texas Instruments C2000 Processors” on page 4-266
- “Memory and Signal Logging limitations on SD Card” on page 3-2

Prerequisites for Logging Signals

Before logging signals:

1. Connect the target hardware to a computer.
2. Create or open a Simulink model. To log signals, the model must have at least one of these blocks.

Block Name	Block Icon
To Workspace block	
Scope block	
Outport block	

Configure Board Parameters and Enable MAT-File Logging

To save signals on an SD card, the target hardware details must be specified.

1. In your model window, open the **Configuration Parameters** dialog box, go to the **Hardware Implementation** pane, and select the name of the target hardware from the **Hardware board** list.
2. In the **Hardware board settings** pane, expand **Target hardware resources** and select **SD card logging**.
3. Select **Enable MAT-file logging on SD card** option.

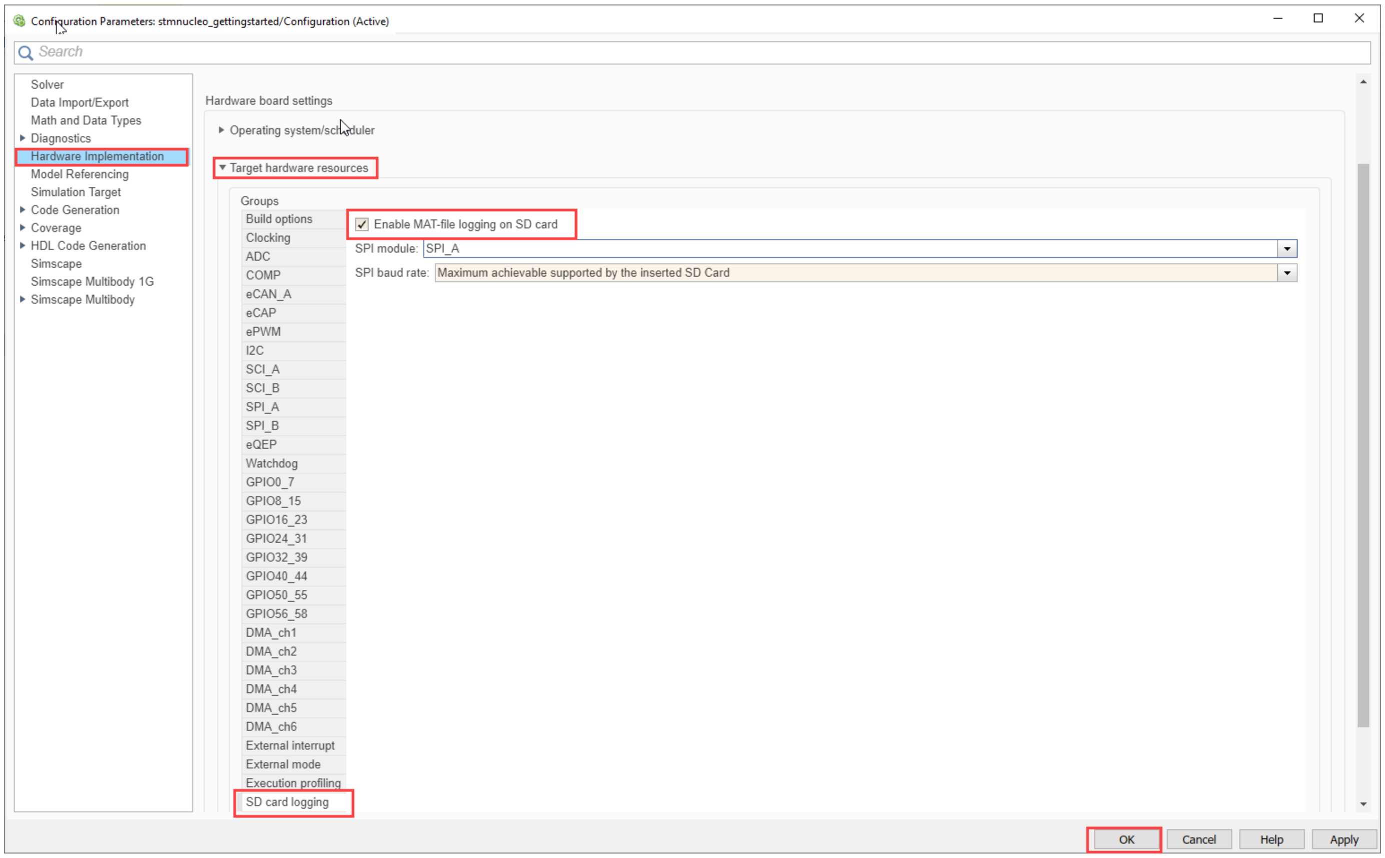
Note

- For C2000 hardware boards, it is advisable to limit the data points for the signals to be logged as C2000 hardware boards have limited memory. In case of memory allocation failure, set *Limit data points to last* to lesser than |512| value.
- For F2803x, F2805x, F2833x, F280x and Concerto(35x and 36x) processors, select *Save format* as |Array| to avoid memory overflow.

4. From the **SPI module** list, select the desired SPI module.

   The default values vary based on the **Hardware board** selected.
5. From the **SPI baud rate** list, select

   - **Inherit from SPI settings -** Inherits the value from the **Desired baud rate in bits/sec** set in the corresponding **SPI_**x pane.
   - **Maximum achievable supported by the inserted SD Card -** The baud rate for SPI interface is automatically selected based on the SD card inserted on the C2000 hardware. In this case, the **Desired baud rate in bits/sec** set in the corresponding **SPI_**x pane is overwritten.
6. Click **Apply**. Click **OK** to save your changes.



✓ Enable MAT-file logging on SD card

SPI module: SPI_A

SPI baud rate: Maximum achievable supported by the inserted SD Card

See Also

Related Examples

- “Memory and Signal Logging limitations on SD Card” on page 3-2

Configure Model to Log Signals on SD Card

SD card logging is supported in models containing To Workspace, Scope, or Outport blocks. You must specify the values for several block parameters.

To configure a Simulink model to run on the target hardware, perform these steps:

1. On the **Modeling** tab, in the **Simulate** section, set the **Stop Time**. The signals are logged for the time period specified in the **Stop Time** parameter. The default value is `Inf`. Enter time in seconds to log signals for that time period.

   Inf

2. In the Simulink model, set the parameter values of To Workspace, Scope, and Outport blocks.

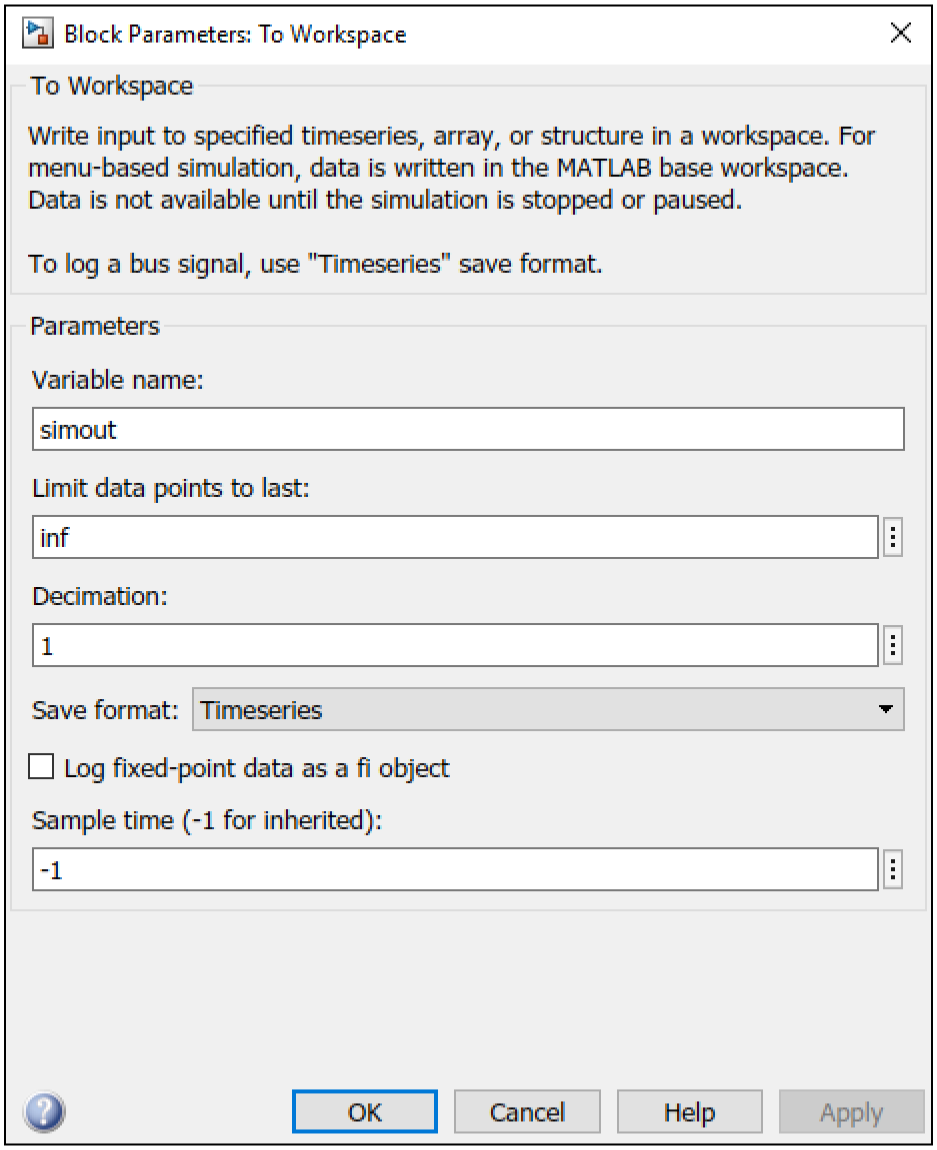
To Workspace Block

To set the parameter values of the To Workspace block:

1. Double-click the block, and specify these parameters in the **Block Parameter** dialog box.

Parameter	Description
Variable name	Specify a variable name for the logged data.
Limit data points to last	Specify the number of data points to be logged in the MAT-file. The maximum number of data points that a MAT-file can contain is 512.
Decimation	Use this parameter for the block to write data points at every *n*th sample, where *n* is the decimation factor. The default decimation, `1`, writes data at every time sample. For example, if you specify **Decimation** as `5`, the block writes data at every fifth sample. For example, if the block sample time is `0.1` and **Decimation** is `5`, the data points at 0, 0.5, 1, 1.5, ... seconds are logged. The data points are logged until the **Simulation stop time** is reached.

Parameter	Description
Save format	Select a format of the variable to which you save data. SD card logging supports only these three formats: `Array`, `Structure with Time`, or `Structure`. • `Array`: Save data as an array with associated time information. This format does not support variable-size data. • `Structure with Time`: Save data as a structure with associated time information. • `Structure`: Save data as a structure.
Sample time (-1 for inherited)	Specify an interval at which the block reads data. When you specify this parameter as `–1`, the sample time is inherited from the driving block.


Block Parameters: To Workspace

To Workspace

Write input to specified timeseries, array, or structure in a workspace. For menu-based simulation, data is written in the MATLAB base workspace. Data is not available until the simulation is stopped or paused.

To log a bus signal, use "Timeseries" save format.

Parameters

Variable name: simout

Limit data points to last: inf

Decimation: 1

Save format: Timeseries

☐ Log fixed-point data as a fi object

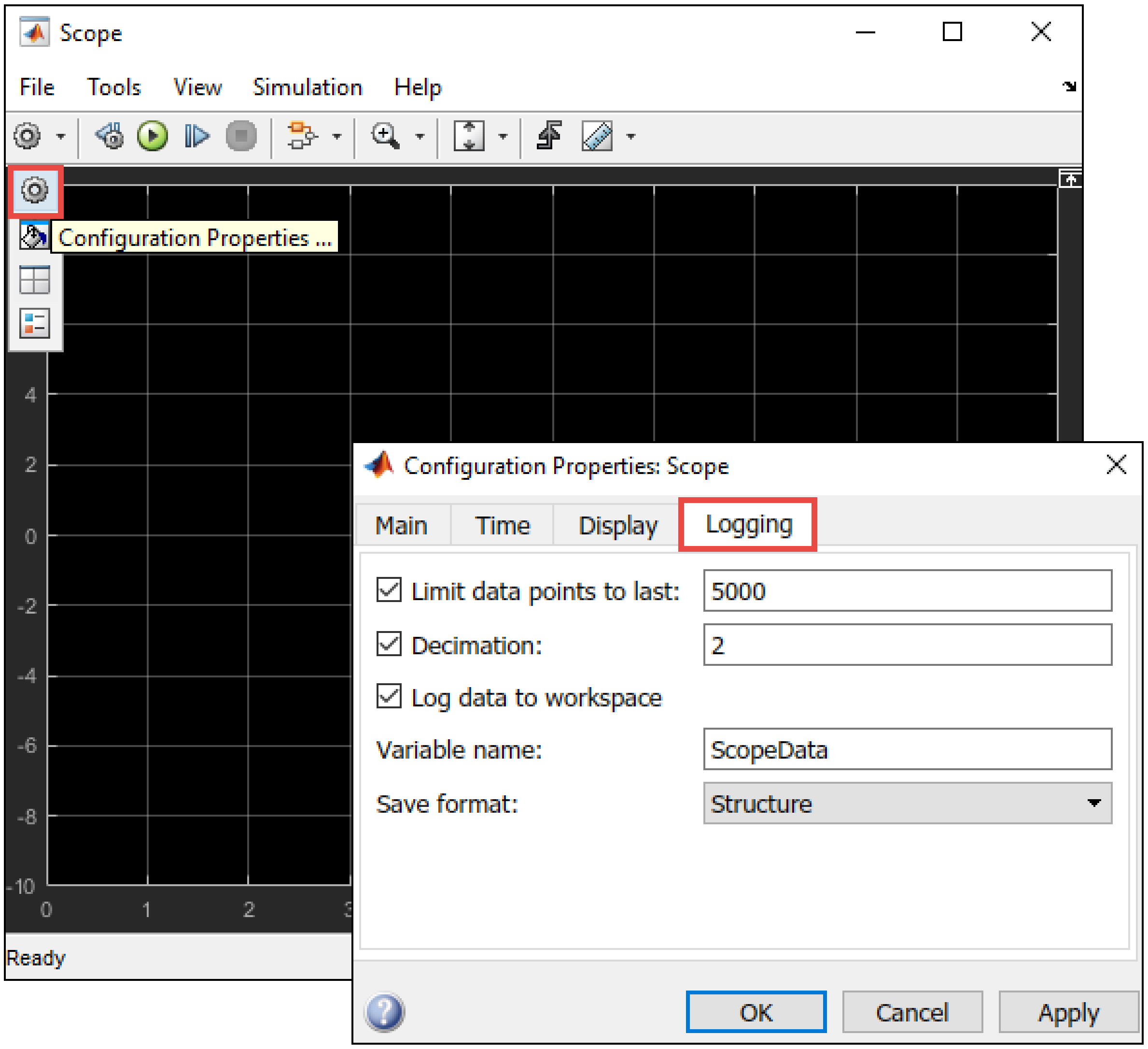
Sample time (-1 for inherited): -1

Scope Block

To set the parameter values of the Scope block:

1. Double-click the block, and click the Configuration Properties button.
2. In the **Main** tab, specify the **Sample time** parameter. When you specify this parameter as –1, the sample time is inherited from the driving block.
3. In the **Logging** tab, set the block parameters listed in this table.

Parameter	Description
Limit data points to last	Specify the number of data points to be logged in the MAT-file. The maximum number of data points that a MAT-file can contain is 512.
Decimation	Use this parameter for the block to write data points at every *n*th sample, where *n* is the decimation factor. The default decimation, `1`, writes data at every time sample. For example, if you specify **Decimation** as `5`, the block writes data at every fifth sample. For example, if the block sample time is `0.1` and **Decimation** is `5`, the data points at 0, 0.5, 1, 1.5, ... seconds are logged. The data points are logged until the **Simulation stop time** is reached.
Log data to workspace	Select this parameter to enable data logging. When you select this parameter, the **Variable name** and **Save format** parameters become available.
Variable name	Specify a variable name for the logged data.
Save format	Select a format of the variable to which you save data. SD card logging supports only these three formats: `Array`, `Structure with Time`, or `Structure`. • `Array`: Save data as an array with associated time information. This format does not support variable-size data. • `Structure with Time`: Save data as a structure with associated time information. • `Structure`: Save data as a structure.



☑ Limit data points to last: 5000

☑ Decimation: 2

☑ Log data to workspace

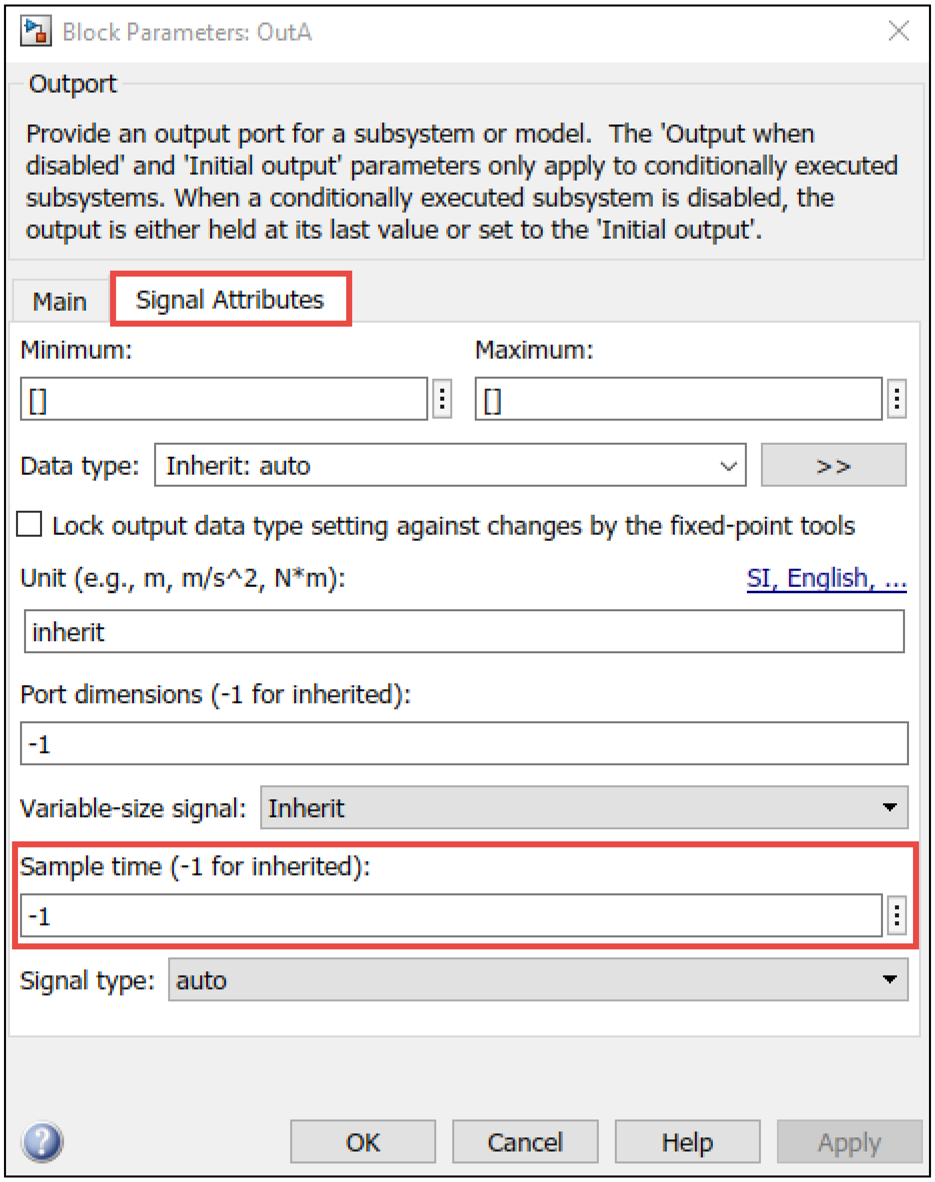
Variable name: ScopeData

Save format: Structure

Outport Block

To set the parameter values of the Outport block:

1 Double-click the block, select the **Signal Attributes** tab, and specify the **Sample time** parameter. When you specify this parameter as –1, the sample time is inherited from the driving block.


Block Parameters: OutA

Outport

Provide an output port for a subsystem or model. The 'Output when disabled' and 'Initial output' parameters only apply to conditionally executed subsystems. When a conditionally executed subsystem is disabled, the output is either held at its last value or set to the 'Initial output'.

Main | Signal Attributes

Minimum: []

Maximum: []

Data type: Inherit: auto

☐ Lock output data type setting against changes by the fixed-point tools

Unit (e.g., m, m/s^2, N*m): inherit

SI, English, ...

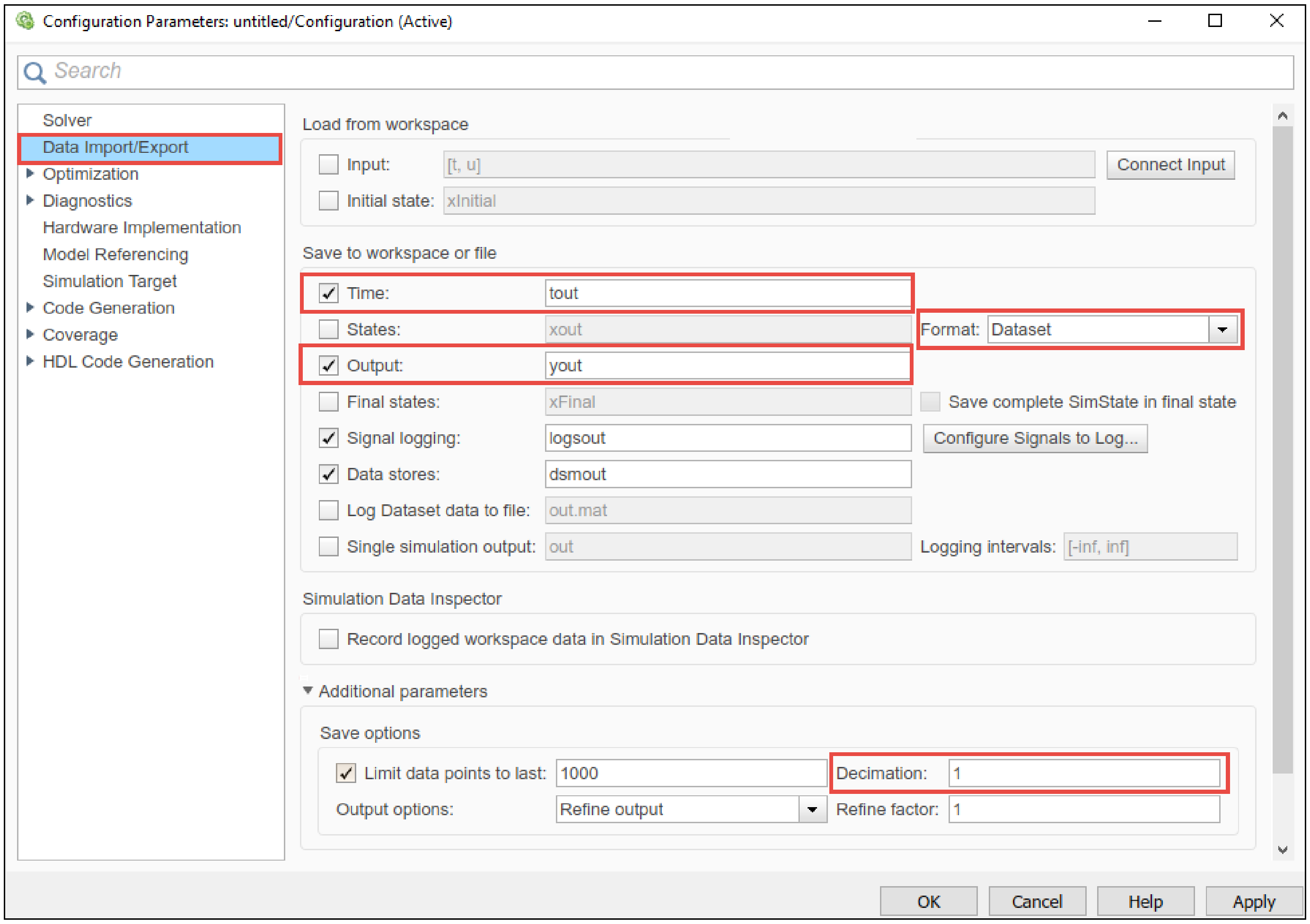
Port dimensions (-1 for inherited): -1

Variable-size signal: Inherit

Sample time (-1 for inherited): -1

Signal type: auto

2 In the model window, open the **Configuration Parameters** dialog box and select **Data Import/ Export**.



Load from workspace
- ☐ Input: [t, u]
- ☐ Initial state: xInitial

Save to workspace or file
- ☑ Time: tout
- ☐ States: xout — Format: Dataset
- ☑ Output: yout
- ☐ Final states: xFinal — ☐ Save complete SimState in final state
- ☑ Signal logging: logsout
- ☑ Data stores: dsmout
- ☐ Log Dataset data to file: out.mat
- ☐ Single simulation output: out — Logging intervals: [-inf, inf]

Simulation Data Inspector
- ☐ Record logged workspace data in Simulation Data Inspector

Save options
- ☑ Limit data points to last: 1000 — Decimation: 1
- Output options: Refine output — Refine factor: 1

3 Set the block parameters listed in this table:

Parameter	Description
Time	Saves time data to the specified variable.
Output	Saves signal data to the specified variable.
Format	Select a format of the variable to which you save data. SD card logging supports only these three formats: `Array`, `Structure with Time`, or `Structure`. • `Array`: Save data as an array with associated time information. This format does not support variable-size data. • `Structure with Time`: Save data as a structure with associated time information. • `Structure`: Save data as a structure.

Parameter	Description
Limit data points to last	Specify the number of data points to be logged in a MAT-file. The maximum number of data points that a MAT-file can contain is 512.
Decimation	Use this parameter for the block to write data points at every *n*th sample, where *n* is the decimation factor. The default decimation, `1`, writes data at every time sample. For example, if you specify **Decimation** as `5`, the block writes data at every fifth sample. For example, if the block sample time is `0.1` and **Decimation** is `5`, the data points at 0, 0.5, 1, 1.5, ... seconds are logged. The data points are logged until the **Simulation stop time** is reached.

See Also

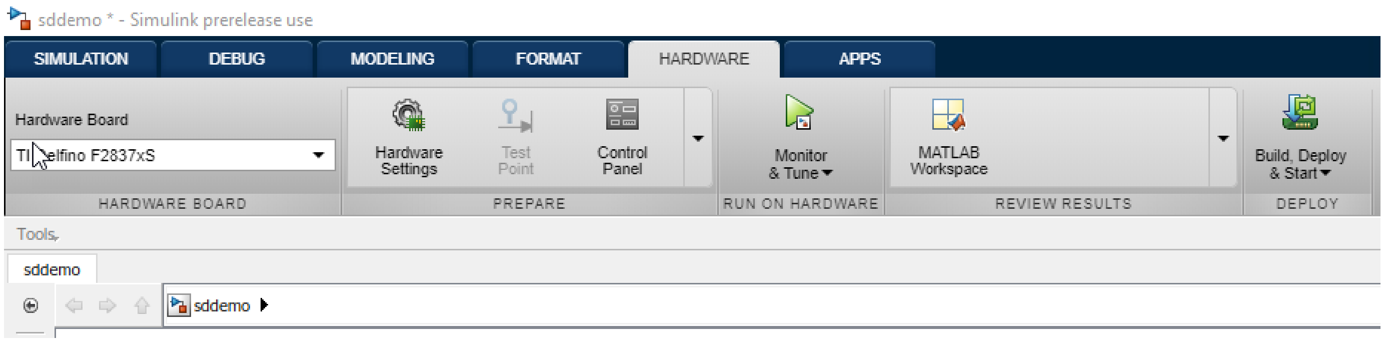
Related Examples

- “MAT-file Logging on SD Card for Texas Instruments C2000 Processors” on page 4-266
- “Memory and Signal Logging limitations on SD Card” on page 3-2

Run Model on Target Hardware

Simulink deploys code and logs signals on an SD card. These signals are saved as MAT-files on the target hardware.

To deploy the code on the target hardware, in the model window, go to **Hardware** tab. In the **Deploy** section, click the **Build Deploy & Start** button. The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the target hardware. Wait for the logging to stop.



Note After the **Simulation stop time** elapses, the logging of signal stops. However, the model continues to run. For example, if the **Simulation stop time** parameter is specified as `10.0` seconds, the logging stops after 10.0 seconds. However, the model continues to run for an indefinite time. If the **Simulation stop time** parameter is specified as `Inf`, the logging continues until the SD card memory is full, or you remove the SD card from the target hardware.

Import MAT-Files into MATLAB

After logging is complete, you can open MAT-files in MATLAB, and use them for further analysis. Since the data points are stored in MAT files, you can directly open the files in MATLAB without converting them into any other format.

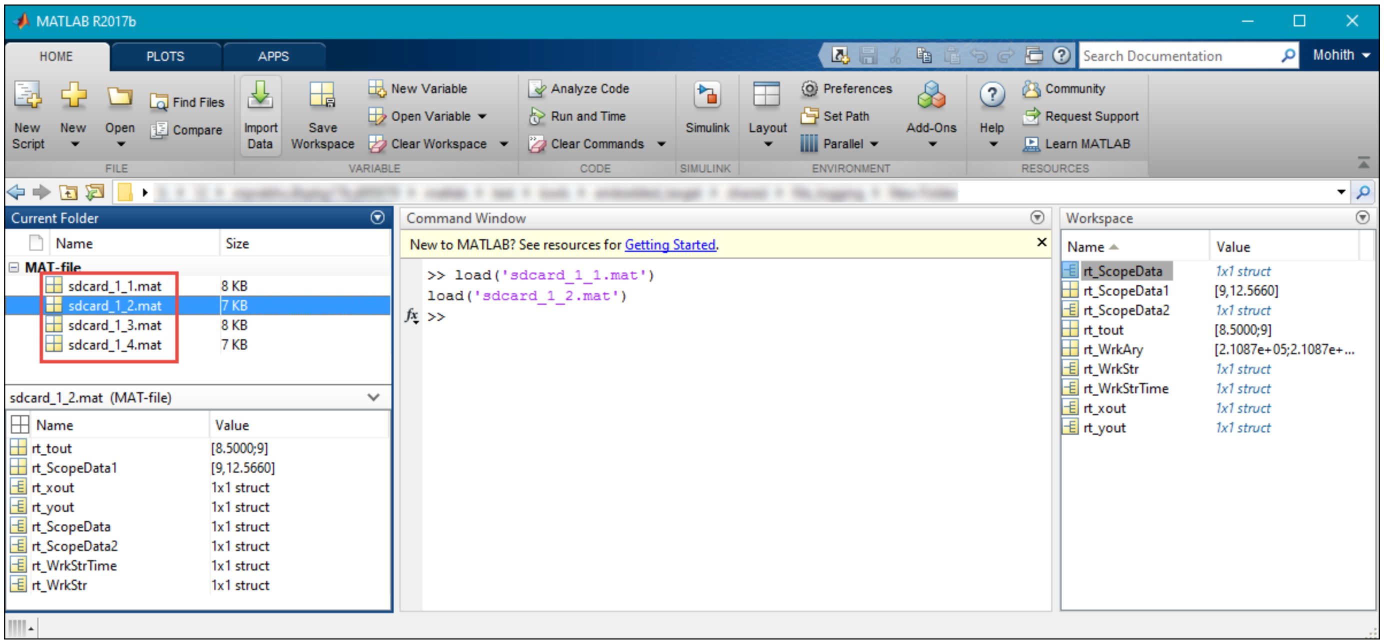
In C2000, remove the SD Card from the board, connect it to a computer and read the .mat files from the explorer.

The files are named as `<modelname>_<runnumber>_<indexnumber>.mat`. The name of your Simulink model is `modelname`. `runnumber` is the number of times the model is run. `runnumber` starts with `1` and is incremented by one for every successive run. `indexnumber` is the MAT-file number in a run. `indexnumber` starts with `1` and is incremented by one for every new file that is created in the same run.

Suppose that the name of the model is `sdcard`. In the first run, Simulink creates `sdcard_1_1.mat` file and starts logging data in this file. After the logging in the first file is completed, Simulink creates `sdcard_1_2.mat` file and continues logging data in this file. Likewise, the logging continues in multiple MAT-files until the **Simulation stop time** is elapsed. If the same model is run again, the new files are named as `sdcard_2_1.mat`, `sdcard_2_2.mat`, and so on.

Note Data loss occurs when:

- The total file size exceeds the available SD card storage capacity.
- The signal logging rate is faster than the SD card writing speed.



[redacted]

Name	Size
MAT-file	
sdcard_1_1.mat	8 KB
sdcard_1_2.mat	7 KB
sdcard_1_3.mat	8 KB
sdcard_1_4.mat	7 KB

sdcard_1_2.mat (MAT-file)

Name	Value
rt_tout	[8.5000;9]
rt_ScopeData1	[9,12.5660]
rt_xout	1x1 struct
rt_yout	1x1 struct
rt_ScopeData	1x1 struct
rt_ScopeData2	1x1 struct
rt_WrkStrTime	1x1 struct
rt_WrkStr	1x1 struct

New to MATLAB? See resources for Getting Started.

```
>> load('sdcard_1_1.mat')
load('sdcard_1_2.mat')
>>
```

Name	Value
rt_ScopeData	1x1 struct
rt_ScopeData1	[9,12.5660]
rt_ScopeData2	1x1 struct
rt_tout	[8.5000;9]
rt_WrkAry	[2.1087e+05;2.1087e+...
rt_WrkStr	1x1 struct
rt_WrkStrTime	1x1 struct
rt_xout	1x1 struct
rt_yout	1x1 struct

See Also

Related Examples

- “MAT-file Logging on SD Card for Texas Instruments C2000 Processors” on page 4-266
- “Memory and Signal Logging limitations on SD Card” on page 3-2

SD Card Logging Troubleshooting

Memory and Signal Logging limitations on SD Card

SD Card Logging Memory Limitation: SD Card logging allocates static memory for all the signal that is logged. The memory allocated is based on the type of logging i.e. `Save Format`, `Sample Time, Model stop time, Decimation` and `Limit data points to last`. Since the RAM size for C2000 processors is limited, there are chances of memory allocation failure when SD card logging is enabled. See “MAT-file Logging on SD Card for Texas Instruments C2000 Processors” on page 4-266 .

- Data memory allocation failure when SD card logging is enabled

  Try the following steps, in case of data memory allocation failure:

  - Select lower value for `Limit data points to last` . Lower value can result in smaller static memory allocation
  - Set the logging type as `Array` instead of `structure or structure with time`.
  - Reduce the number of signals that can be logged in the model
  - Reduce the sample rate for logging either by using rate `transition blocks` or `decimation` parameter
- Code memory allocation failure

  Code memory allocation fails when the SD Card feature and the algorithm together is generating code bigger than the allocated code memory. This can happen for the processors with low RAM memory like F280x, F2803x, F2805x, F2833x and Concerto (35x and 36x).

  Try the following steps in case of code memory failure:

  - Use the `Boot From Flash` option for the target
  - Reduce the size of the model there by reducing the code generated

SD Card Logging Connection Limitation: SD card logging uses SPI (Serial Peripheral Interface) to send the data between processor and memory. The data logging will be affected based on the SPI connections and SPI parameter settings. Very few control cards offer dedicated SD card logging slots (like F2837xD) to place the SD card. You have to use external SD card module to log data for other control card.

If the signals are not logging on SD card, try the following steps:

- Ensure proper SPI module is selected for SD card logging
- If you are using the external SD card interface, ensure the connections for `SPIMO, SOMI, CLK` and `STE` pins are correct
- Ensure correct supply and ground is connected for the external SD card interface
- Ensure the GPIO pins configurations for the selected SPI_x module in configuration parameters are correct
- Start with lower baud rate for SPI. Change the SPI baud rate setting in Configuration parameters to `Inherit from SPI settings` and start with low value like 1000000 MHz (Desired baud rate in bits per sec in SPI_x module)
- Ensure that you are not using the GPIO pins and the SPI module dedicated to SD card with other peripherals . This will cause conflict and the logging will fail

See Also

"MAT-file Logging on SD Card for Texas Instruments C2000 Processors" on page 4-266

Examples

Getting Started with Texas Instruments C2000 Microcontroller Blockset

This example shows you how to use C2000™ Microcontroller Blockset to run a Simulink® model on Texas Instruments C2000 hardware.

Introduction

C2000™ Microcontroller Blockset enables you to generate a real-time executable and download it to your TI development board. The blockset includes a library of Simulink blocks for configuring and accessing Texas Instruments C2000 peripherals and communication interfaces. In this example you will learn how to configure a simple Simulink model to generate code for any TI C2000 hardware and to run the generated code on the board to periodically turn an LED on and off. This example shows you how to configure two different models to run in different CPU cores.

Prerequisites

If you are new to Simulink, we recommend completing the:

- "Hardware Setup for C2000 Microcontroller Blockset" on page 1-3
- Interactive Simulink® Tutorial.
- Reading the Getting Started section of the Simulink documentation
- Running Simulink Getting Started example.
- "Overview of Creating a Model and Generating Executable for C2000 Processors"

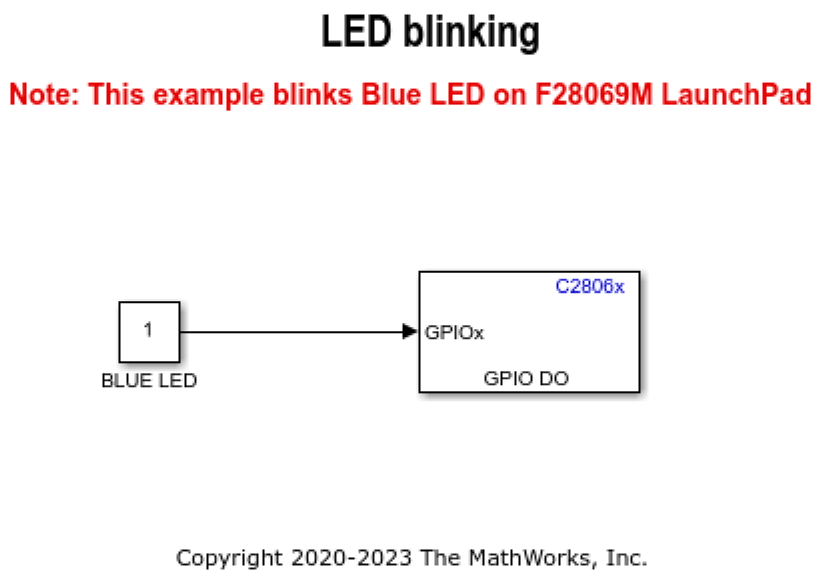
Required Hardware

To run this example, you can use any TI C2000 hardware. For your convenience, the following hardware models are preconfigured to execute the example:

- `TI Piccolo F28069M LaunchPad`
- `TI Delfino F28379D LaunchPad`
- `TI F2838x(C28x) ControlCard` and 180 pin ControlCard Docking Station

Model

```
open_system('c28069_blink.slx');
```

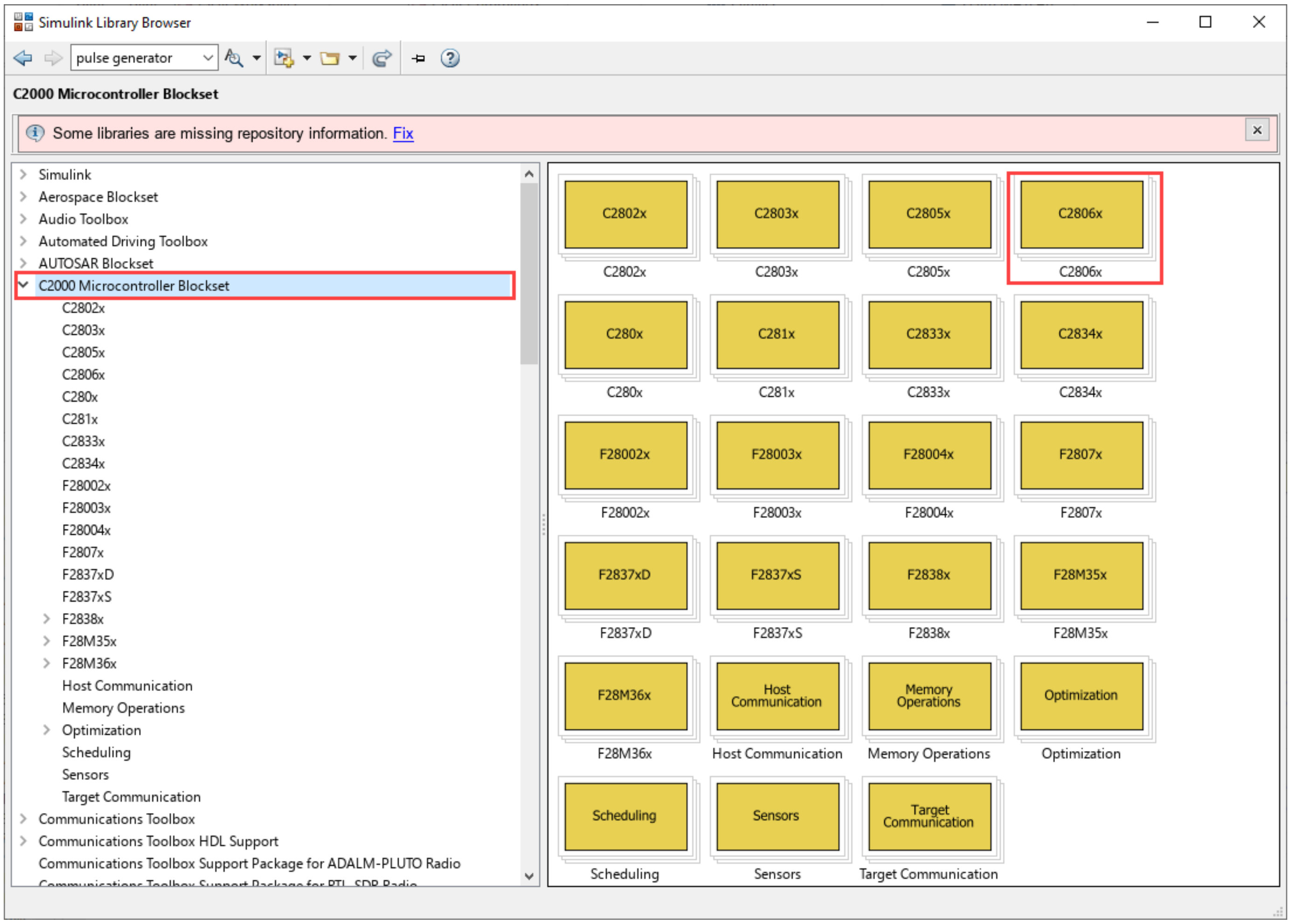

LED blinking
Note: This example blinks Blue LED on F28069M LaunchPad
1
BLUE LED
C2806x
GPIOx
GPIO DO
Copyright 2020-2023 The MathWorks, Inc.

Task 1 - Create, Configure and Run the Model for TI Piccolo F28069M LaunchPad (Single Core)

In this task you will create and configure a simple model blinking an on-board LED to run on TI Piccolo F28069M LaunchPad (Single core).

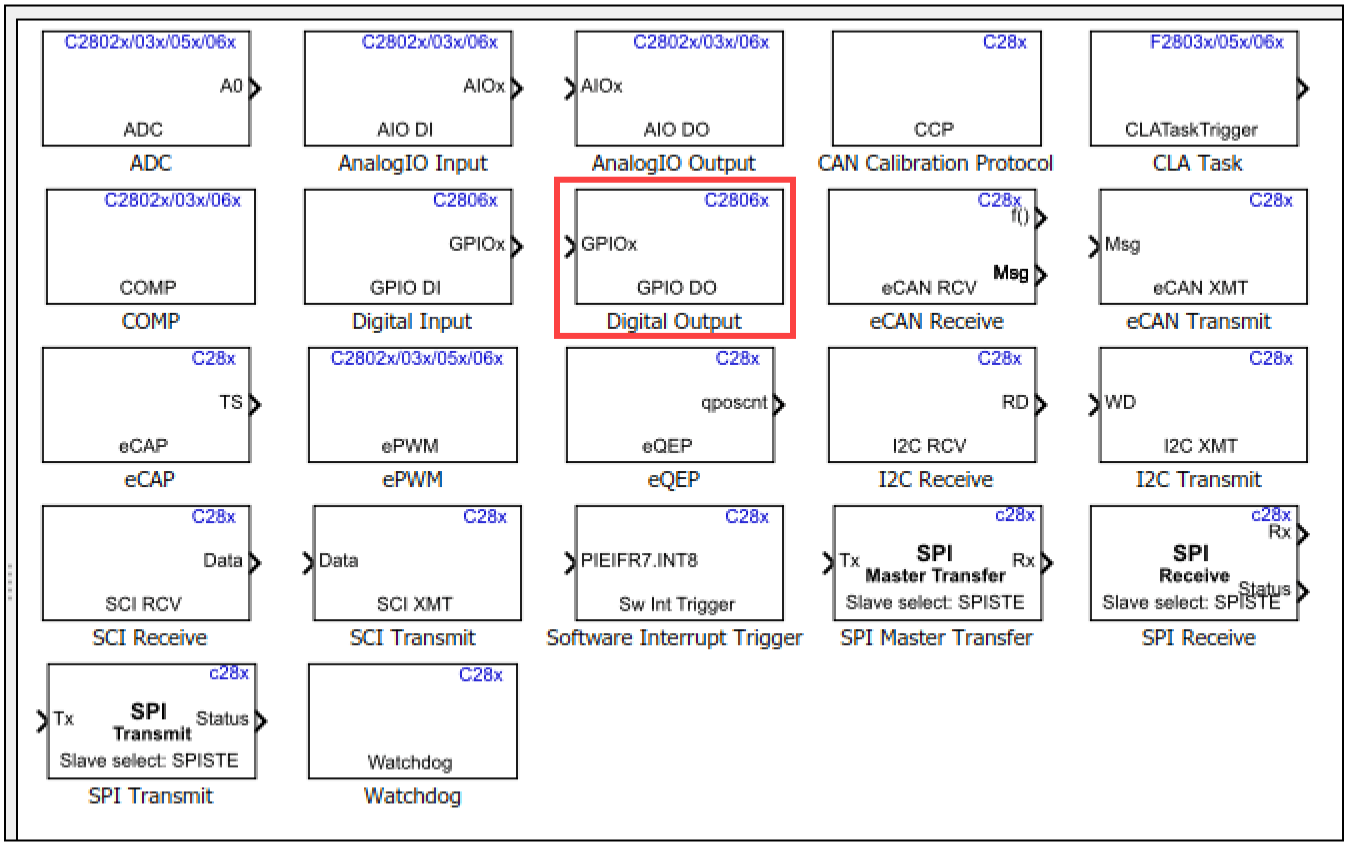
1. Enter `slLibraryBrowser` at the MATLAB® prompt. This opens the Simulink Library Browser.

2. In the Simulink Library Browser, navigate to **Libraries > C2000™ Microcontroller Blockset and select C2806x** processor.



Some libraries are missing repository information. Fix

3. Double-click the **GPIO** block. Review the block mask, which contains a description of the block and parameters for configuring the associated user GPIO.



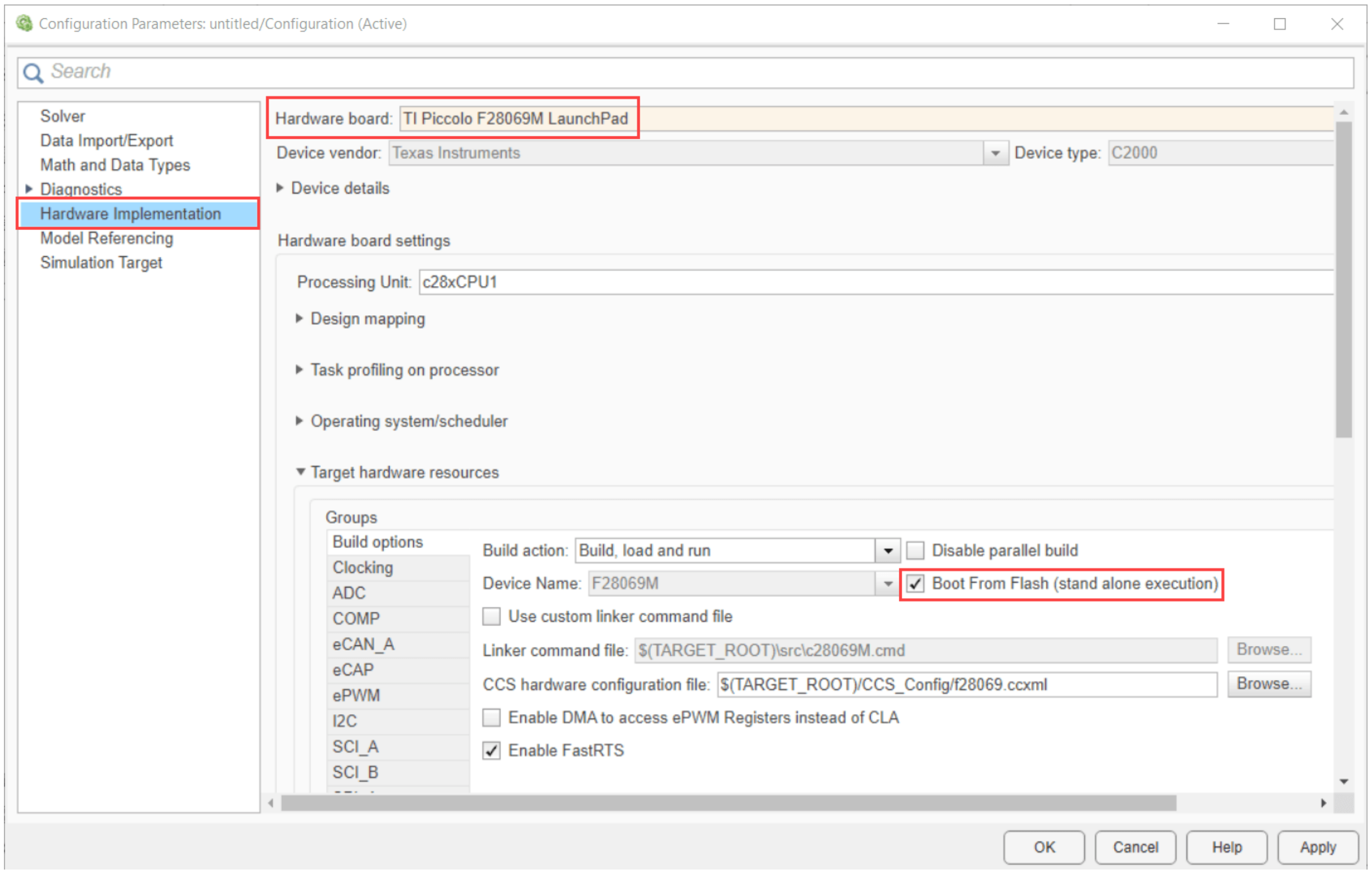
4. In MATLAB, select **HOME > New > Simulink Model**.

5. Drag the **GPIO** block from **C2806x** library to the model.

6. Drag the **Constant** block from **Simulink** library to the model.

7. Connect the output of the **Constant** block to either one of the corresponding input of each **GPIO** block. Configure the **GPIO and Constant** block as follows:

- a. Configure GPIO block with pin 39 for blue LED or with pin 34 for red LED.
- b. Ensure the Toggle GPIO option is enabled. Enabling this option will toggle the GPIO pin when input passed to this block is 1. The rate of toggle will be based on the sample time of the input.


Configuration Parameters: untitled/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target

Hardware board: TI Piccolo F28069M LaunchPad

Device vendor: Texas Instruments Device type: C2000

Device details

Hardware board settings

Processing Unit: c28xCPU1

Design mapping

Task profiling on processor

Operating system/scheduler

Target hardware resources

Groups

- Build options
- Clocking
- ADC
- COMP
- eCAN_A
- eCAP
- ePWM
- I2C
- SCI_A
- SCI_B

Build action: Build, load and run

Disable parallel build

Device Name: F28069M

[✓] Boot From Flash (stand alone execution)

Use custom linker command file

Linker command file: $(TARGET_ROOT)\src\c28069M.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28069.ccxml

Enable DMA to access ePWM Registers instead of CLA

[✓] Enable FastRTS

OK Cancel Help Apply

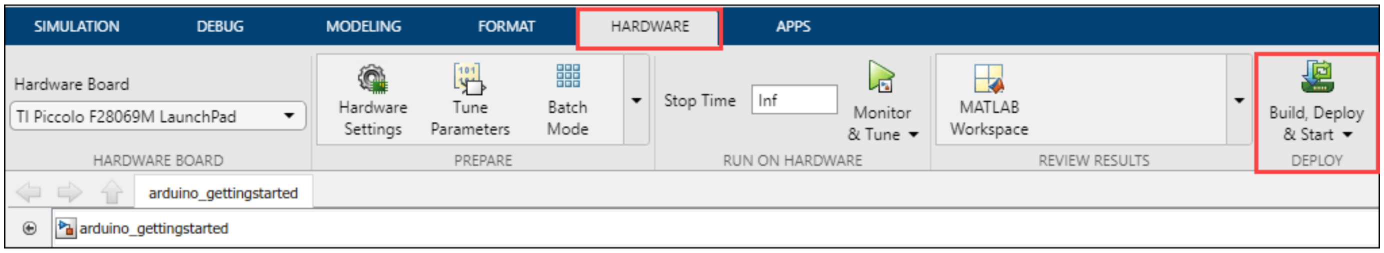
- c. For Constant block, specify the sample time (0.5). Since the toggle option is enabled in GPIO block, this ensures GPIO is 1 for period of 0.5 sec and 0 for 0.5 secs. So, the period of the waveform is 1 sec.

8. Save your model and connect the TI Piccolo F28069M LaunchPad to your computer with a USB cable.

9. Open the saved model to configure the model for F28069M Launchpad:

- a. Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
- b. Go to **Hardware Implementation > Hardware board** and select **TI Piccolo F2806M LaunchPad**.
- c. Ensure **Boot from Flash** is enabled if the application has to load to the flash. If you do not select this option, the application loads to the RAM.
- d. Click **OK**.

10. Go to the **Hardware** tab and click **Build, Deploy & Start** to generate code for the model and deploy the executable.



11. The generated code is built and run on the F28069M Launchpad automatically. When the model starts running on the F28069M Launchpad, observe that the user LED on the board blinks with a period of 1 second.

12. Save your model. A `preconfigured` model is included for your convenience.

Task 2 - Create, Configure and Run the Model for TI Delfino F28379D LaunchPad (Dual Core)

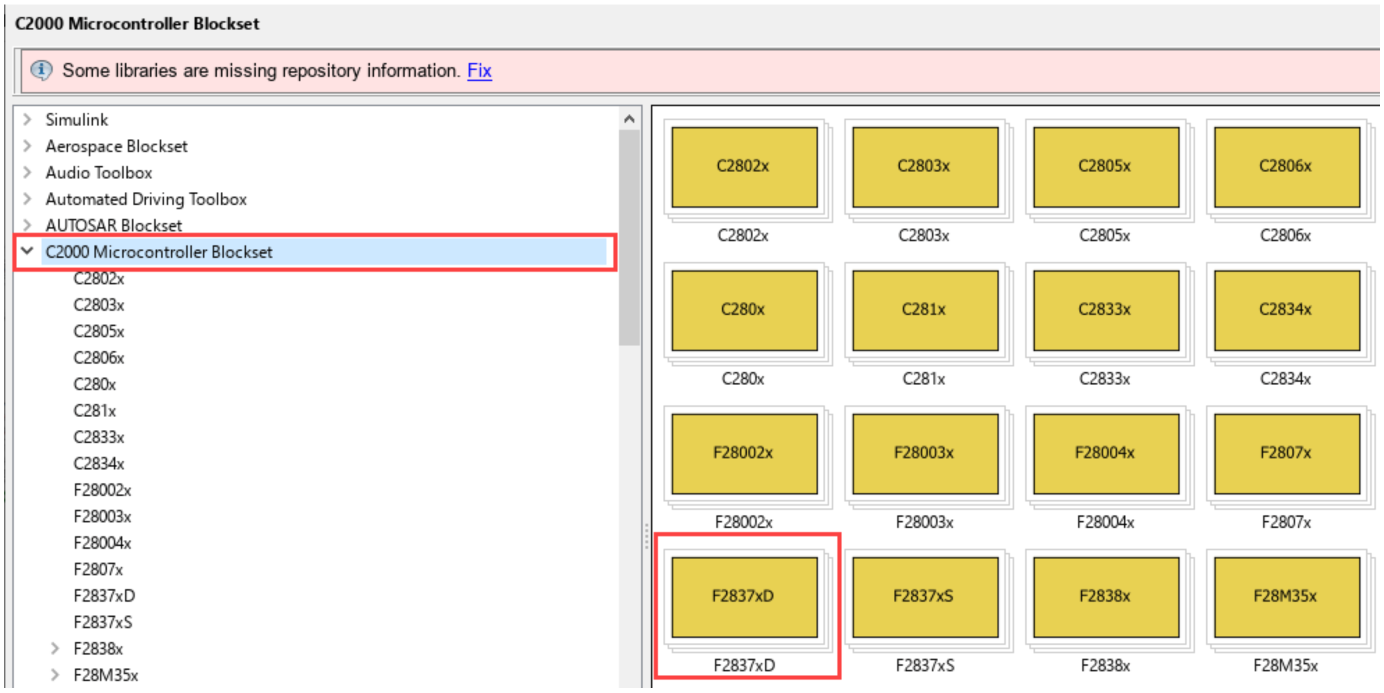
In this task you will create and configure a simple model blinking an on-board LED to run on F28379D Launchpad (Dual core).

F2837xD is a Dual core. You can create 2 models. One each for `CPU1` and `CPU2` to generate 2 separate executables.

Create a Model for CPU1 of F28379D

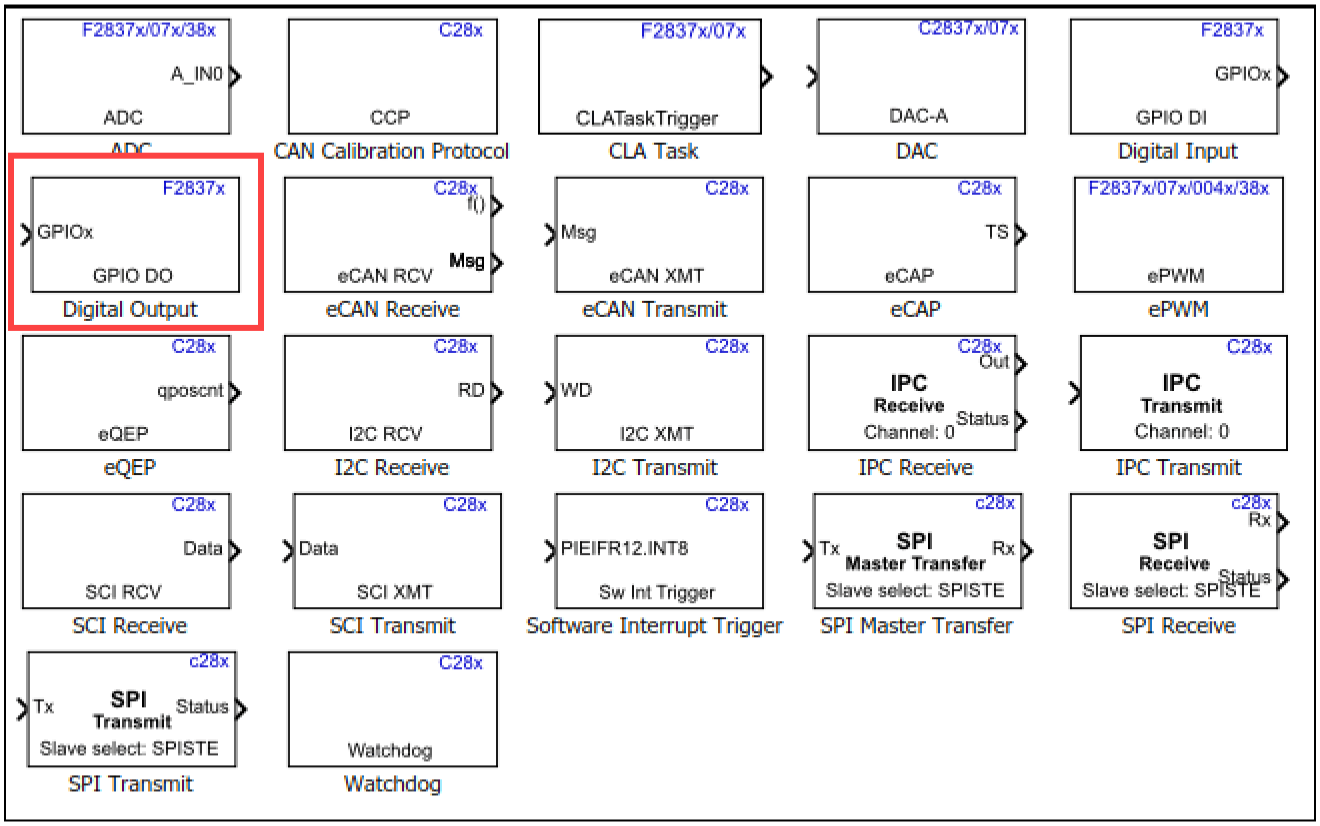
1. Enter `slLibraryBrowser` at the MATLAB prompt. This opens the Simulink Library Browser.

2. In the Simulink Library Browser, navigate to **Libraries > C2000™ Microcontroller Blockset** and select **F2837xD** processor.


C2000 Microcontroller Blockset

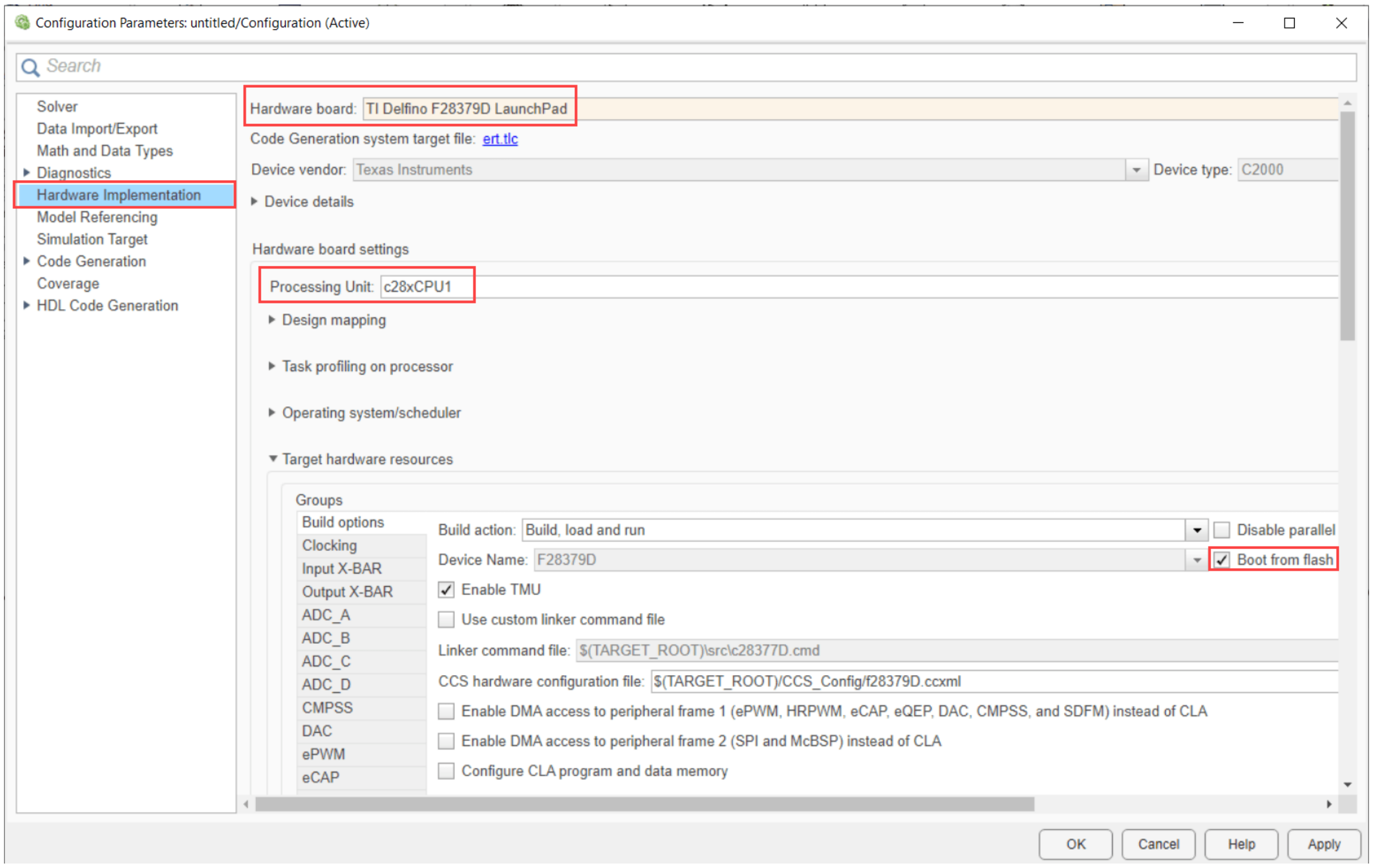
Some libraries are missing repository information. Fix

3. Double-click the **GPIO** block. Review the block mask, which contains a description of the block and parameters for configuring the associated user GPIO.



4. Perform the **steps 4 to 8 of Task 1** for TI Delfino F28379D LaunchPad. For this task, configure GPIO block with pin 34 for red LED.

5. Select the TI Delfino F28379D LaunchPad and verify the CPU used for your model. For this task, select CPU1. This selection creates and downloads the executable for CPU1.


Configuration Parameters: untitled/Configuration (Active)

Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Hardware board settings

Processing Unit: c28xCPU1

Target hardware resources

Build action: Build, load and run

Device Name: F28379D

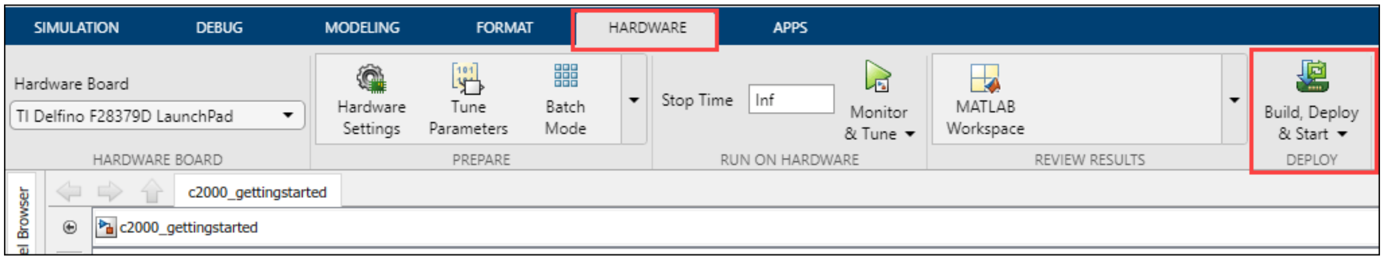
[✓] Boot from flash

[✓] Enable TMU

Linker command file: $(TARGET_ROOT)\src\c28377D.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28379D.ccxml

6. Go to the **Hardware** tab and click **Build, Deploy & Start** to generate code for the model and deploy the executable.


Hardware Board: TI Delfino F28379D LaunchPad

Stop Time: Inf

7. The generated code is built and run on the F28379D Launchpad automatically. When the model starts running on the F28379D Launchpad, observe that the user LED on the board blinks with a period of 1 second.

8. Save your model. A `preconfigured` model is included for your convenience.

Create a Model for CPU2 of F28379D

Since TI Delfino F28379D LaunchPad is a dual core processor, you can create a similar model using CPU2 option and blink the LED.

1. Perform the **steps 1 to 4** of **Task 2** for TI Delfino F28379D LaunchPad.

2. Select the TI Delfino F28379D LaunchPad and verify the CPU used for your model. For this task, select CPU2 and configure GPIO block with pin 31 for blue LED. This selection creates and downloads the executable for CPU2.

3. The generated code is built and run on the F28379D Launchpad automatically. When the model starts running on the F28379D Launchpad, observe that the user LED on the board blinks with a period of 1 second.

4. Save your model. A `preconfigured` model is included for your convenience.

Note:

- Do not use the GPIO pins is used in CPU1.
- Ensure that peripherals used in CPU1 are not used in CPU2. If same peripherals are used in CPU1 and CPU2, the model may not work as expected.

By creating 2 models for TI Delfino F28379D LaunchPad, you can blink 2 LED's. One from CPU1 and another with CPU2.

Task 3 - Create, Configure and Run the Model for F2838x (Multi-Core) Processor

In this task you will create and configure a simple model blinking an on-board LED to run on F2838x (Multi-core).

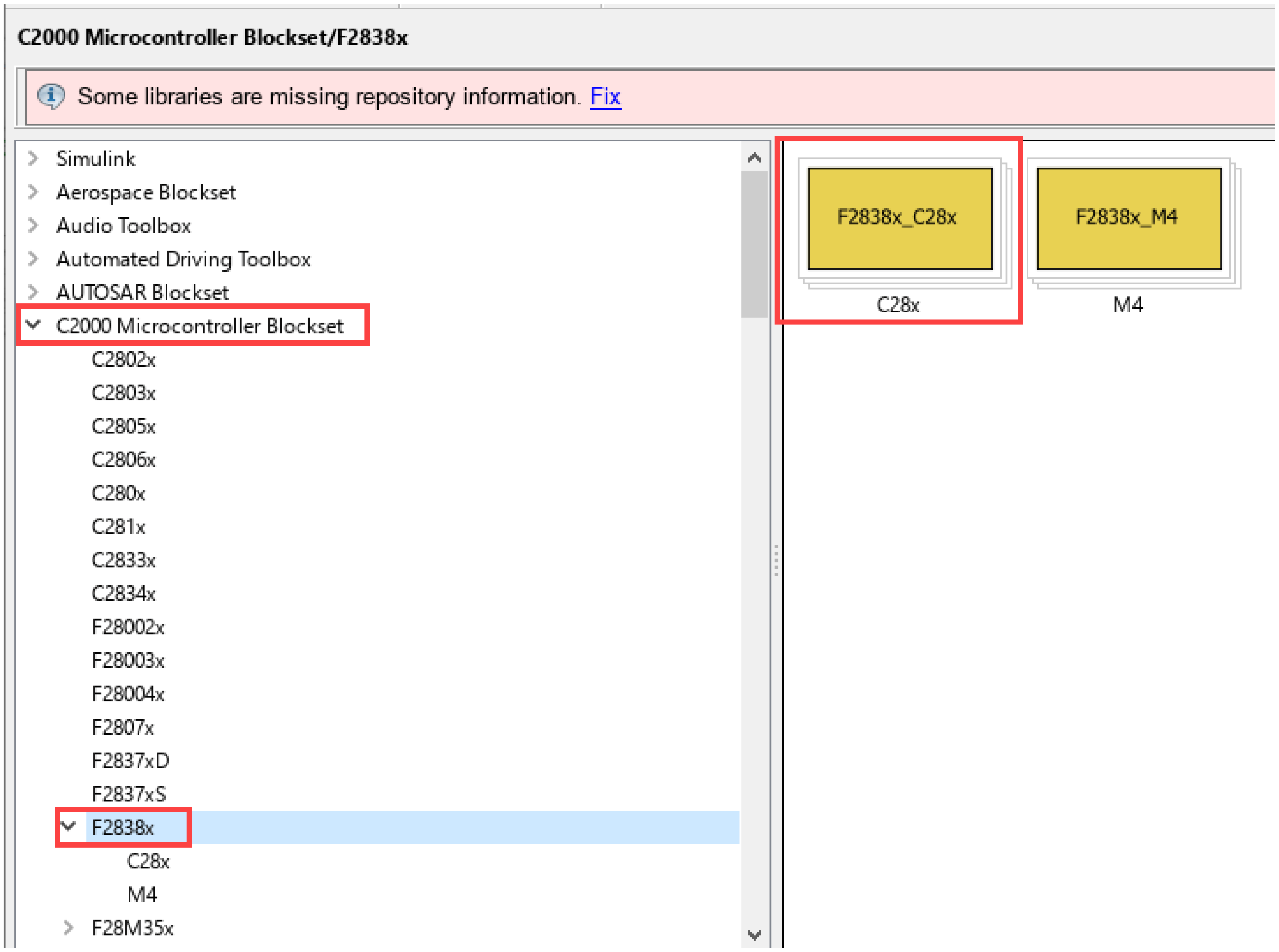
F2838x is a Multi-core processor. You can create 3 models to generate 3 separate executables for:

- `CPU1`
- `CPU2`
- `ARM Cortex-M4`

For CPU1 and CPU2 of F2838x(C28x)

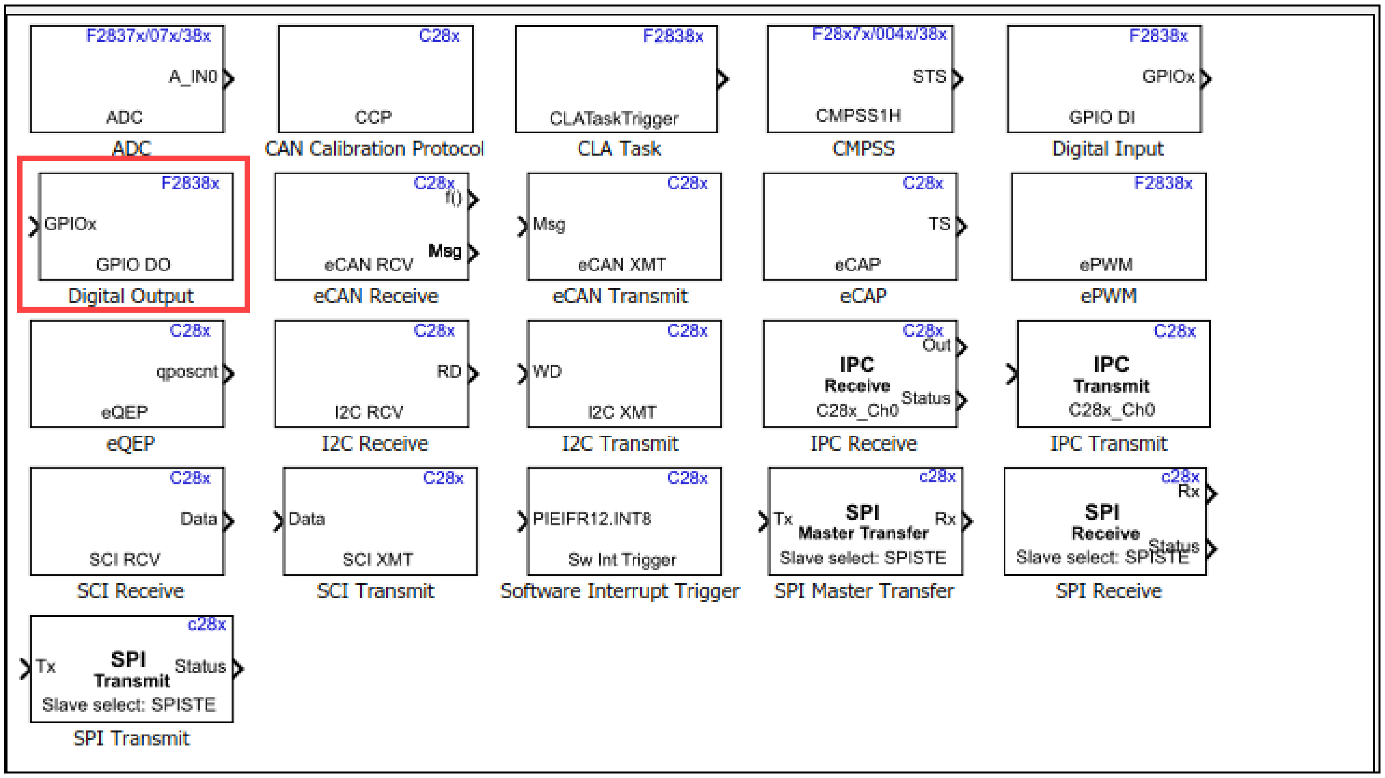
1. Enter `slLibraryBrowser` at the MATLAB prompt. This opens the Simulink Library Browser.

2. In the Simulink Library Browser, navigate to **Libraries > C2000™ Microcontroller Blockset > F2838x > Select F2838x_C28x** processor.


C2000 Microcontroller Blockset/F2838x

Some libraries are missing repository information. Fix

3. Double-click the **GPIO** block. Review the block mask, which contains a description of the block and parameters for configuring the associated user GPIO.



4. Create CPU1 and CPU2 model by performing the **steps 4 to 8** of **Task 2 for F2838x(C28x)** processor.

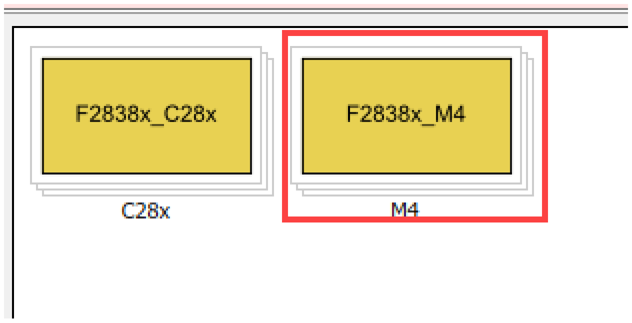
5. A preconfigured model (`CPU1` and `CPU2`) is included for your convenience.

Note: Do not use the same GPIO pins used in CPU1 and CPU2.

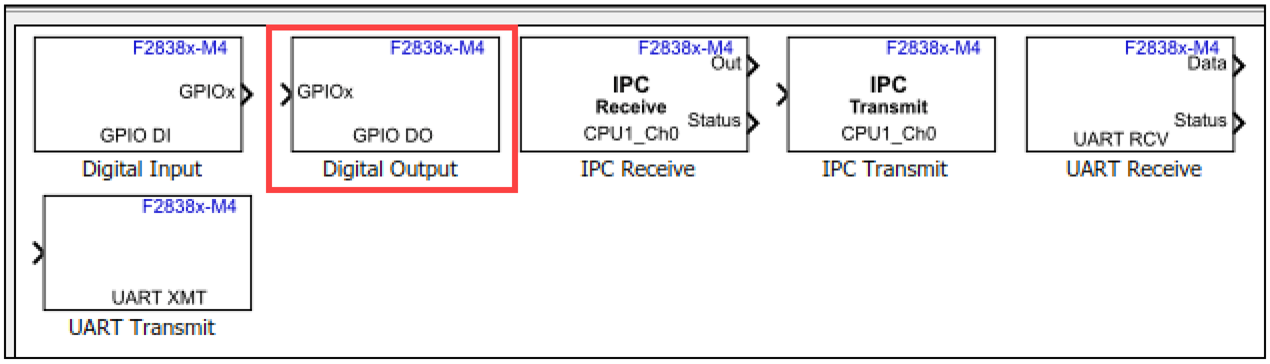
For ARM Cortex-M4 of F2838x_M4

1. Enter `slLibraryBrowser` at the MATLAB prompt. This opens the Simulink® Library Browser.

2. In the Simulink Library Browser, navigate to **Libraries > C2000™ Microcontroller Blockset > F2838x > Select F2838x_M4** processor.



3. Double-click the **GPIO** block. Review the block mask, which contains a description of the block and parameters for configuring the associated user GPIO.

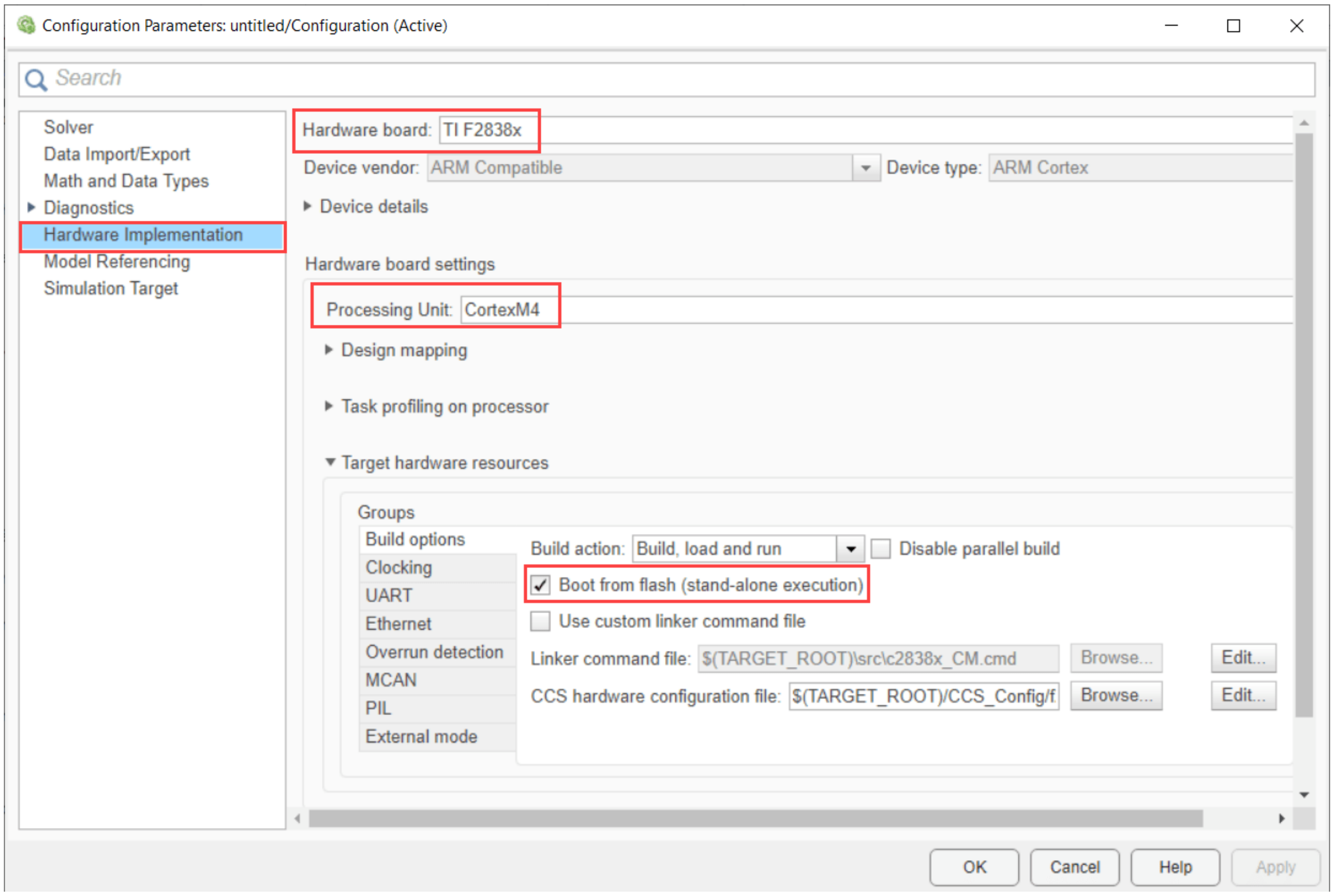


4. Perform the **steps 4 to 7** of **Task 1 for F2838x-M4** processor and save the model.

Note: While configuring the GPIO pin, select the appropriate pin applicable for ARM Cortex-M4 processor. Do not use same GPIO pins used in CPU1 or CPU2.

5. Open the saved model to configure the model for F2838x_M4:

- a. Open the **Modeling** tab and press **Ctrl+E** (Model settings) to open **Configuration Parameters** dialog box.
- b. Go to **Hardware Implementation > Hardware board** and select **TI F2838x. In the Processing Unit, select *CortexM4**.


Configuration Parameters: untitled/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target

Hardware board: TI F2838x

Device vendor: ARM Compatible Device type: ARM Cortex

Device details

Hardware board settings

Processing Unit: CortexM4

Design mapping

Task profiling on processor

Target hardware resources

Groups: Build options, Clocking, UART, Ethernet, Overrun detection, MCAN, PIL, External mode

Build action: Build, load and run ☐ Disable parallel build

☑ Boot from flash (stand-alone execution)

☐ Use custom linker command file

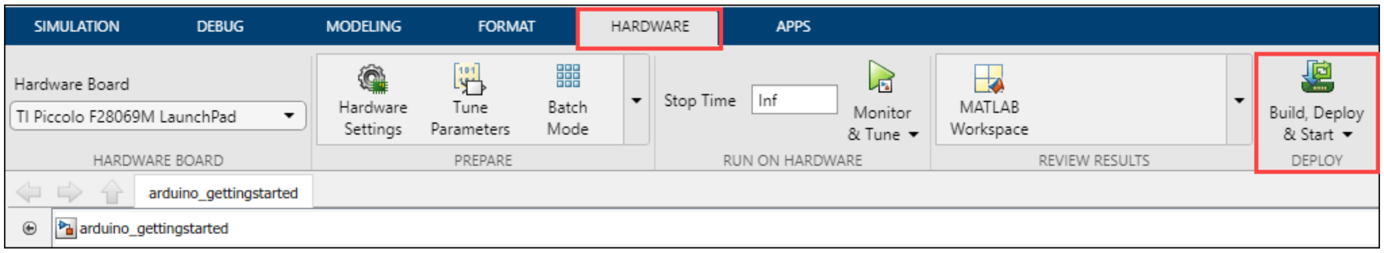
Linker command file: $(TARGET_ROOT)\src\c2838x_CM.cmd Browse... Edit...

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f Browse... Edit...

OK Cancel Help Apply

- c. Ensure **Boot from Flash** is enabled if the application has to load to the flash. If you do not select this option, the application loads to the RAM.
- d. Click **OK**.

6. Go to the **Hardware** tab and click **Build, Deploy & Start** to generate code for the model and deploy the executable.



7. The generated code is built on the F2838x (ARM Cortex-M4) and run automatically. When the model starts running on the F2838x (ARM Cortex-M4), observe GPIO pin toggling with a period of 1 second.

8. Save your model. A `preconfigured` model is included for your convenience.

Note: For Multicore processor like F2838x, try Interchanging the GPIO's between CPU2 and ARM Cortex-M4 or CPU1 and ARM Cortex-M4 and observe the LED blink.

Other Things to Try

Experiment with other blocks in the Texas Instruments C2000 Processors Block Library. For example:

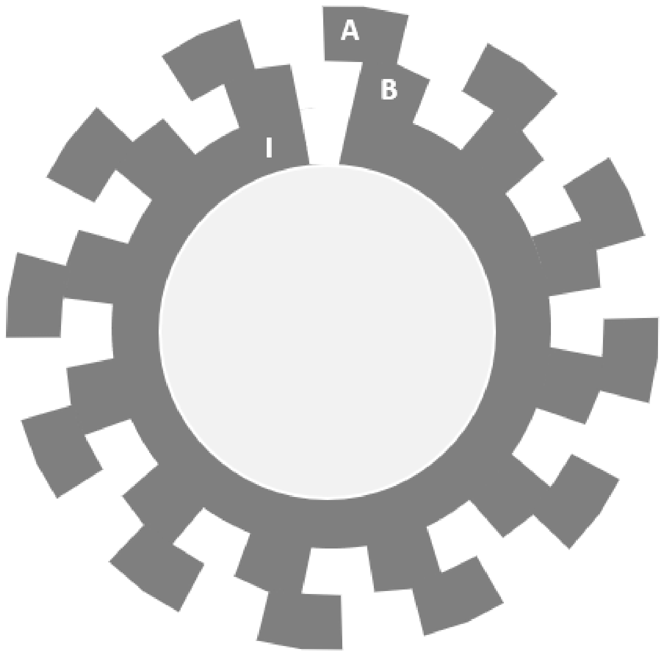
- Create and run a model that repeatedly blinks an LED. Hint: use the **Pulse Generator** block.
- Create and run a model that repeatedly blinks an LED. Hint: use the external mode example.

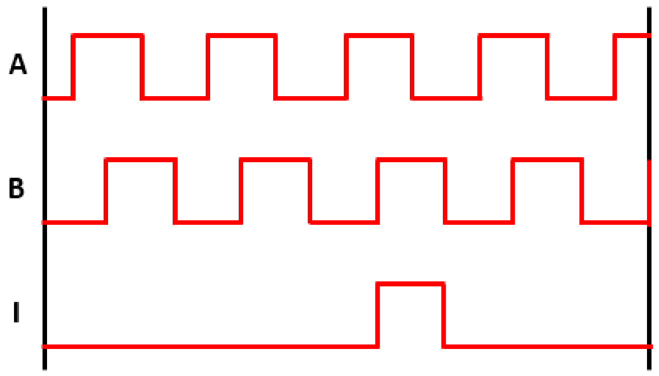
Field-Oriented Control of PMSM with Quadrature Encoder Using C2000 Processors

This example implements the field-oriented control (FOC) technique to control the speed of a three-phase permanent magnet synchronous motor (PMSM). The FOC algorithm requires rotor position feedback, which is obtained by a quadrature encoder sensor. For details about FOC, see "Field-Oriented Control (FOC)" (Motor Control Blockset).

A closed-loop FOC algorithm is used to regulate the speed and torque of a three-phase PMSM. This example uses C28x peripheral blocks from the C2000™ Microcontroller Blockset and MCB library blocks from Motor Control Blockset™.

This example uses the quadrature encoder sensor to measure the rotor position. The quadrature encoder sensor consists of a disk with two tracks or channels that are coded 90 electrical degrees out of phase. This creates two pulses (A and B) that have a phase difference of 90 degrees and an index pulse (I). Therefore, the controller uses the phase relationship between A and B channels and the transition of channel states to determine the direction of rotation of the motor.





Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- F28035 control card + DRV8312-C2-KIT inverter: mcb_pmsm_foc_qep_f28035
- F28335 control card + DRV8312-C2-KIT inverter: mcb_pmsm_foc_qep_f28335
- LAUNCHXL-F280049C controller + BOOSTXL-DRV8305 inverter: mcb_pmsm_foc_qep_f280049C

- Three-phase PMSM with optional QEP sensors attached to connector J4 of the DRV8312-C2-KIT inverter board or connector J12 of LAUNCHXL-F280049C.
- For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Field-Oriented Control of PMSM Using Quadrature Encoder” (Motor Control Blockset).

For connections related to the preceding hardware configuration, see “Hardware Connections” on page 1-81.

Available Models

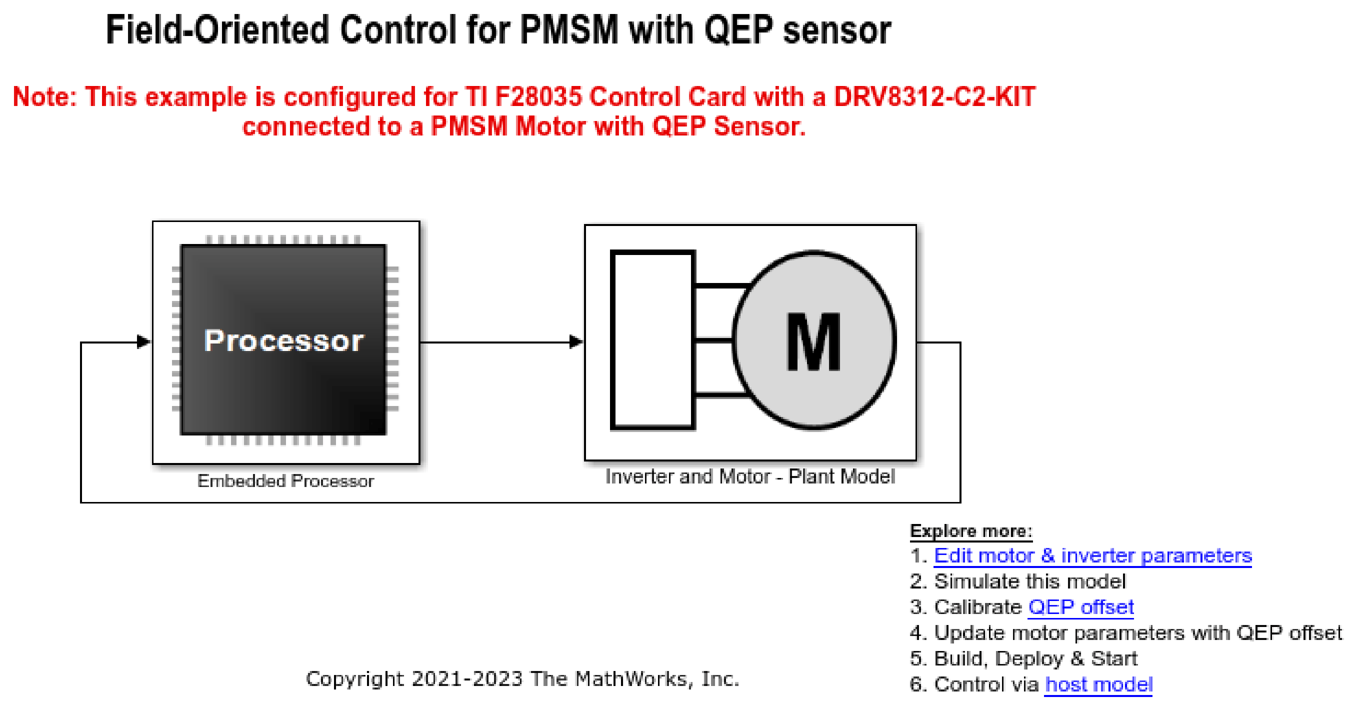
The example includes these models:

- mcb_pmsm_foc_qep_f28035
- mcb_pmsm_foc_qep_f28335
- mcb_pmsm_foc_qep_f280049C

Note: For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Field-Oriented Control of PMSM Using Quadrature Encoder” (Motor Control Blockset).

You can use these models for both simulation and code generation. You can also use the open_system command to open the Simulink® models. For example, use this command for a F28035 based controller.

```
open_system('mcb_pmsm_foc_qep_f28035.slx');
```



Field-Oriented Control for PMSM with QEP sensor

Note: This example is configured for TI F28035 Control Card with a DRV8312-C2-KIT connected to a PMSM Motor with QEP Sensor.

Explore more:
1. Edit motor & inverter parameters
2. Simulate this model
3. Calibrate QEP offset
4. Update motor parameters with QEP offset
5. Build, Deploy & Start
6. Control via host model

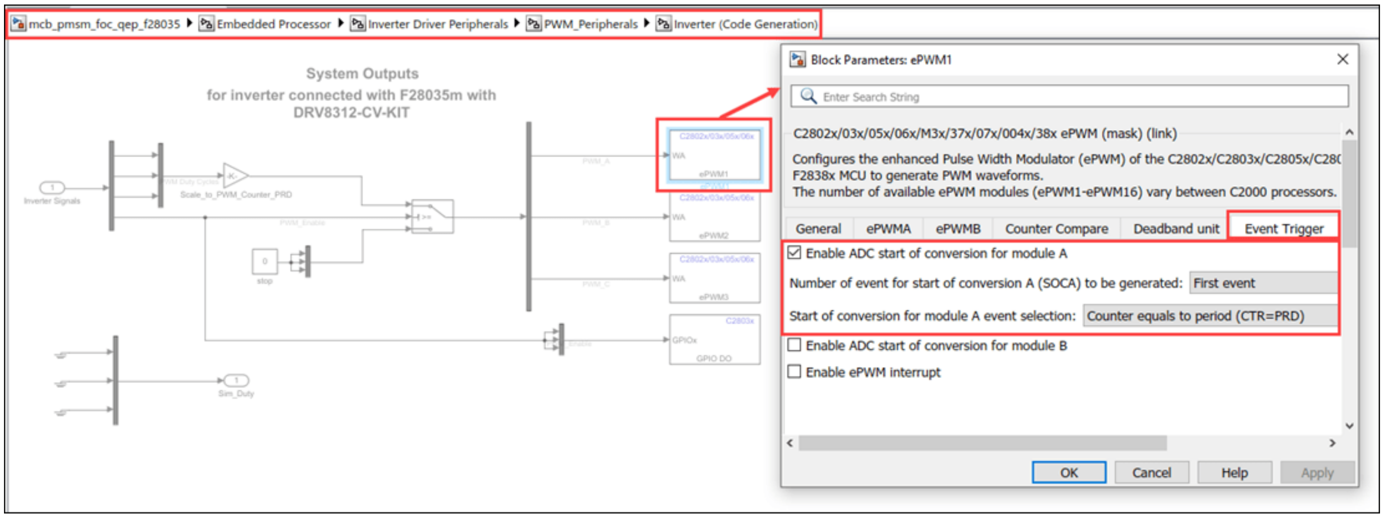
Copyright 2021-2023 The MathWorks, Inc.

You may need to change the model parameters to fit your specific motor. Match motor voltage and power characteristics to the controller. A conventional voltage-source inverter drives motor. The controller algorithm generates six pulse width modulation (PWM) signals using a vector PWM technique for six power switching devices. The inverter measures the current of the two motor inputs (ia and ib) input currents of the motor (ia and ib) using two analog-to-digital converters (ADCs) and sends the measurements to the processor.

Peripheral Block Configurations

Set the peripheral block configurations for this model. Double-click on the blocks to open block parameter configurations. You can use the same parameter values if you want to run this example for other hardware boards.

- **ePWM Block configuration**



C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the enhanced Pulse Width Modulator (ePWM) of the C2802x/C2803x/C2805x/C28
F2838x MCU to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

☑ Enable ADC start of conversion for module A

Number of event for start of conversion A (SOCA) to be generated: First event

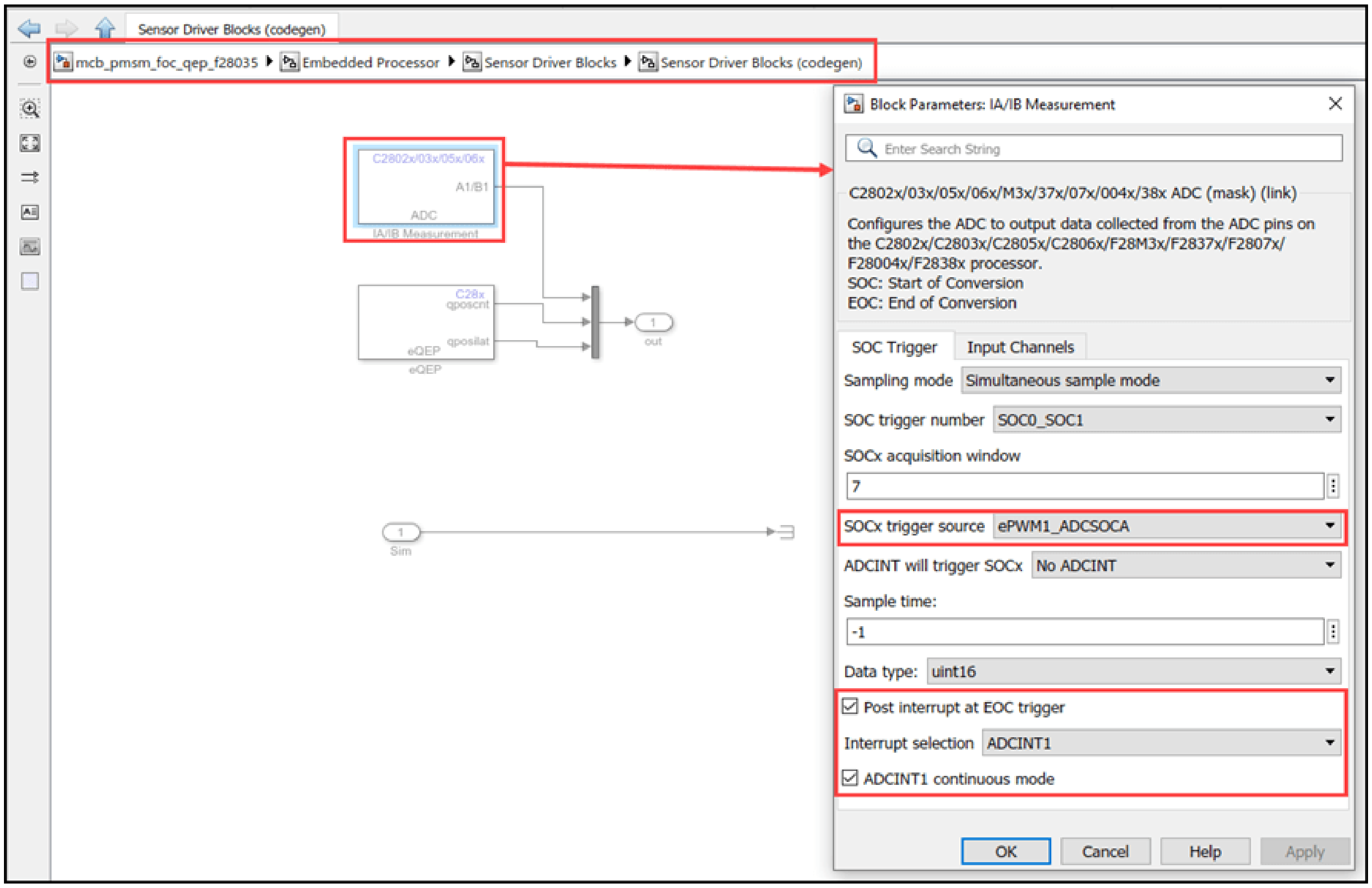
Start of conversion for module A event selection: Counter equals to period (CTR=PRD)

☐ Enable ADC start of conversion for module B

☐ Enable ePWM interrupt

- **ADC Block configuration**

The algorithm in this example uses an asynchronous scheduling. The pulse width modulation (PWM) block triggers the ADC conversion. At the end of conversion, the ADC posts an interrupt that triggers the main FOC algorithm. For more information, refer “ADC Interrupt Based Scheduling” on page 1-12.


Sensor Driver Blocks (codegen)
mcb_pmsm_foc_qep_f28035 ▸ Embedded Processor ▸ Sensor Driver Blocks ▸ Sensor Driver Blocks (codegen)
C2802x/03x/05x/06x
A1/B1
ADC
IA/IB Measurement
C28x
qposcnt
qposlat
eQEP
eQEP
1
out
1
Sim
Block Parameters: IA/IB Measurement
Enter Search String
SOC Trigger
Input Channels
OK
Cancel
Help
Apply

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ADC (mask) (link)

Configures the ADC to output data collected from the ADC pins on the C2802x/C2803x/C2805x/C2806x/F28M3x/F2837x/F2807x/F28004x/F2838x processor.
SOC: Start of Conversion
EOC: End of Conversion

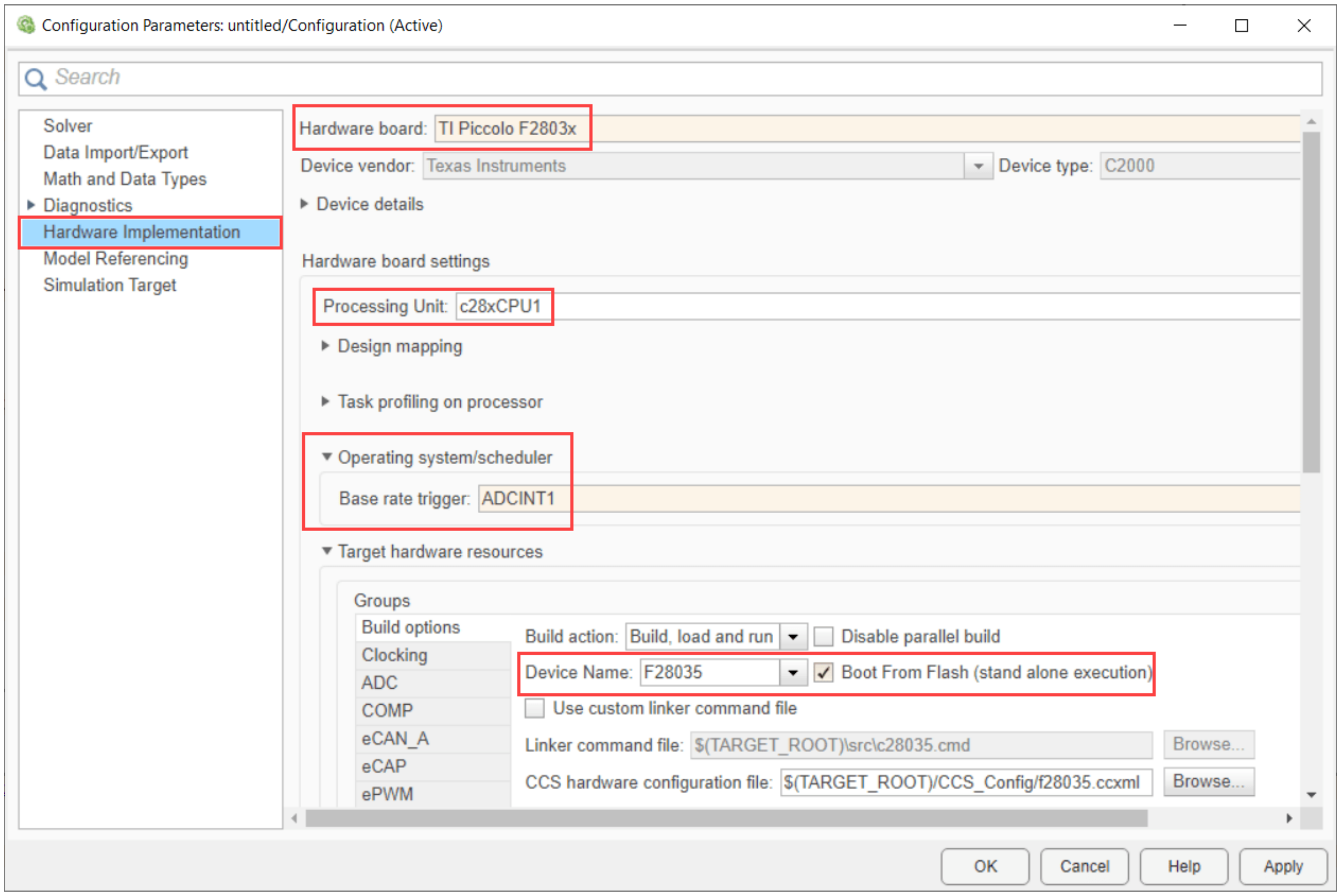
Sampling mode: Simultaneous sample mode
SOC trigger number: SOC0_SOC1
SOCx acquisition window: 7
SOCx trigger source: ePWM1_ADCSOCA
ADCINT will trigger SOCx: No ADCINT
Sample time: -1
Data type: uint16
☑ Post interrupt at EOC trigger
Interrupt selection: ADCINT1
☑ ADCINT1 continuous mode

Configure the Model

1. Open the mcb_pmsm_foc_qep_f28035 model. This model is configured for TI Piccolo F2803x hardware.

2. To run the model on other TI C2000 processors, first press **Ctrl+E** to open the Configuration Parameters dialog box. Then, select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

3. The following screenshots show the scheduler configurations performed in the model. You can use the same parameter values if you want to run this example for other hardware boards.


Configuration Parameters: untitled/Configuration (Active)

Hardware board: TI Piccolo F2803x

Device vendor: Texas Instruments

Device type: C2000

Processing Unit: c28xCPU1

Base rate trigger: ADCINT1

Build action: Build, load and run

Disable parallel build

Device Name: F28035

✓ Boot From Flash (stand alone execution)

Use custom linker command file

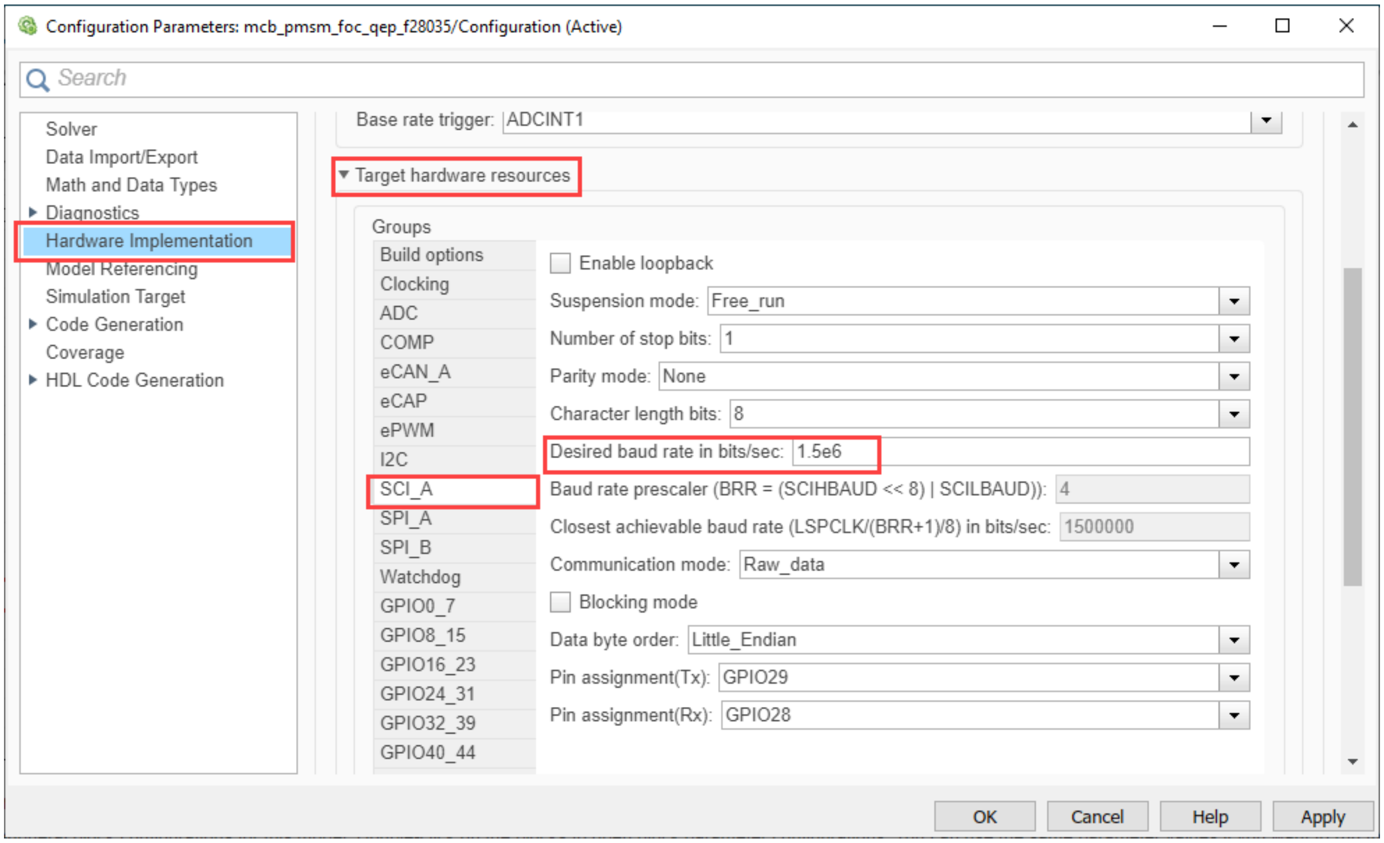
Linker command file: $(TARGET_ROOT)\src\c28035.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28035.ccxml

Note:

- Sampling rate of the ADC block should be same as the base rate of the model determined by the PWM period of the ePWM block.
- Base rate trigger selection should be same as the interrupt triggered by the ADC module. For more information, see Model Configuration Parameters for Texas Instruments C2000 Processors.
- Ensure that the **Default parameter behavior** (Configuration Parameters > Code Generation > Optimization) is set to **Inlined**.

4. Ensure that the baud rate is set to 1.5e6 bits/sec.


Configuration Parameters: mcb_pmsm_foc_qep_f28035/Configuration (Active)

Base rate trigger: ADCINT1

Target hardware resources

Groups: Build options, Clocking, ADC, COMP, eCAN_A, eCAP, ePWM, I2C, SCI_A, SPI_A, SPI_B, Watchdog, GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_44

Enable loopback
Suspension mode: Free_run
Number of stop bits: 1
Parity mode: None
Character length bits: 8
Desired baud rate in bits/sec: 1.5e6
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)): 4
Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec: 1500000
Communication mode: Raw_data
Blocking mode
Data byte order: Little_Endian
Pin assignment(Tx): GPIO29
Pin assignment(Rx): GPIO28

Required MathWorks® Products

To simulate model:

For the models: **mcb_pmsm_foc_qep_f28035**, **mcb_pmsm_foc_qep_f28335** and **mcb_pmsm_foc_qep_f280049C**

- Motor Control Blockset™
- Fixed-Point Designer™

To generate code and deploy model:

For the models: **mcb_pmsm_foc_qep_f28035**, **mcb_pmsm_foc_qep_f28335** and **mcb_pmsm_foc_qep_f280049C**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

Prerequisites

- If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see “Estimate Control Gains and Use Utility Functions” (Motor Control Blockset).

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Note: For F28335 processor you need to use external FTDI for serial communication.

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Open-Loop Control of 3-Phase AC Motors Using C2000 Processors" on page 4-171.

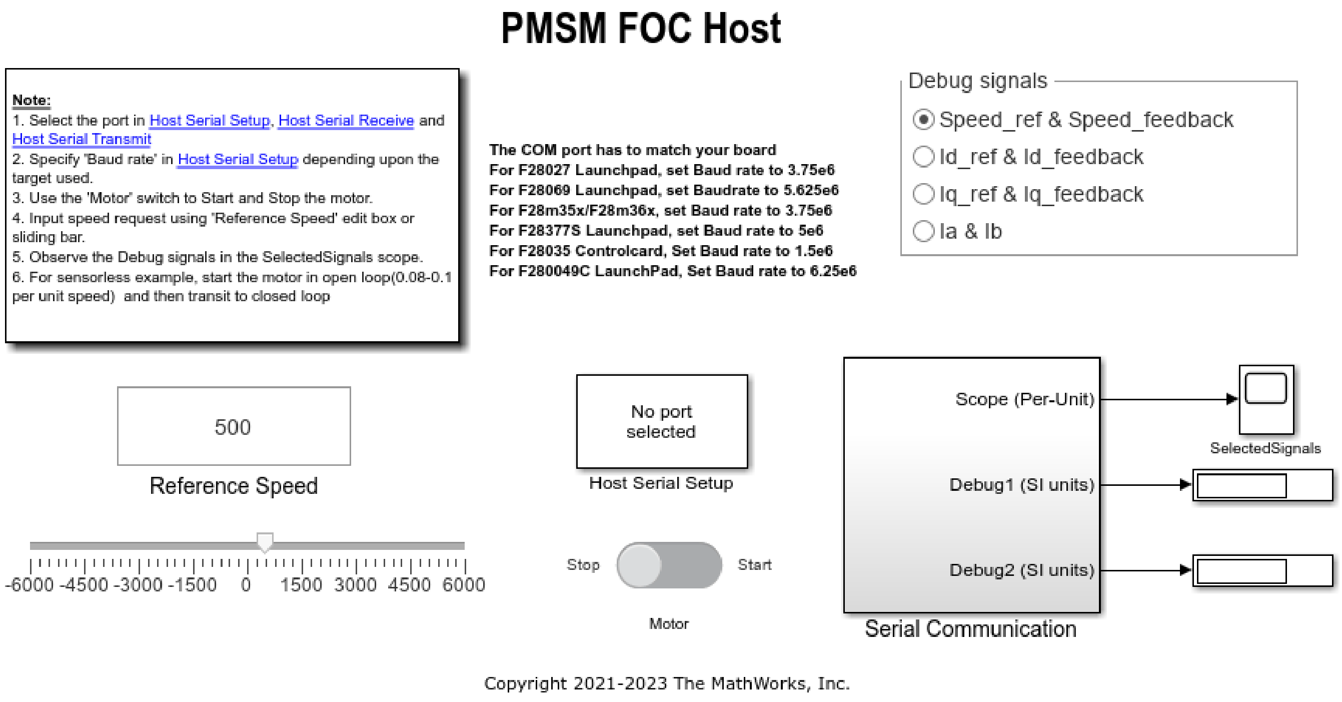
4. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 1-73.

5. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" (Motor Control Blockset).

6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

7. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a host model.

```
open_system('mcb_pmsm_foc_host_model.slx');
```



Note:
1. Select the port in Host Serial Setup, Host Serial Receive and Host Serial Transmit
2. Specify 'Baud rate' in Host Serial Setup depending upon the target used.
3. Use the 'Motor' switch to Start and Stop the motor.
4. Input speed request using 'Reference Speed' edit box or sliding bar.
5. Observe the Debug signals in the SelectedSignals scope.
6. For sensorless example, start the motor in open loop(0.08-0.1 per unit speed) and then transit to closed loop

The COM port has to match your board
For F28027 Launchpad, set Baud rate to 3.75e6
For F28069 Launchpad, set Baudrate to 5.625e6
For F28m35x/F28m36x, set Baud rate to 3.75e6
For F28377S Launchpad, set Baud rate to 5e6
For F28035 Controlcard, Set Baud rate to 1.5e6
For F280049C LaunchPad, Set Baud rate to 6.25e6

For details about the serial communication between the host and target models, see “Host-Target Communication” (Motor Control Blockset).

8. Set the parameter **Port** of the following blocks in the mcb_pmsm_foc_host_model model to match the COM port of the host computer:

- mcb_pmsm_foc_host_model > Host Serial Setup
- mcb_pmsm_foc_host_model > Serial Communication > Host Serial Receive
- mcb_pmsm_foc_host_model > Serial Communication > SCI_TX > Host Serial Transmit

9. Update the Reference Speed value in the host model.

10. Click **Run** on the **Simulation** tab to run the host model.

11. Change the position of the motor switch to Start, to start running the motor.

12. Observe the debug signals from serial communication, in the Time Scope of host model.

More About:

- “Host-Target Communication” (Motor Control Blockset)
- “Estimate PMSM Parameters Using Recommended Hardware” (Motor Control Blockset)
- For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Field-Oriented Control of PMSM Using Quadrature Encoder” (Motor Control Blockset)
- “Quadrature Encoder Offset Calibration for PMSM Motor” on page 1-73

Sensorless Field-Oriented Control of PMSM Using C2000 Processors

This example implements the field-oriented control (FOC) technique to control the speed of a three-phase permanent magnet synchronous motor (PMSM). For details about FOC, see “Field-Oriented Control (FOC)” (Motor Control Blockset).

This example uses the sensorless position estimation technique. You can select either the sliding mode observer or flux observer to estimate the position feedback for the FOC algorithm used in the example.

A closed-loop FOC algorithm is used to regulate the speed and torque of a three-phase PMSM. This example uses C28x peripheral blocks from the C2000™ Microcontroller Blockset and MCB library blocks from Motor Control Blockset.

The Sliding Mode Observer (SMO) block generates a sliding motion on the error between the measured and estimated position. The block produces an estimated value that is closely proportional to the measured position. The block uses stator voltages (V_α, V_β) and currents (I_α, I_β) as inputs and estimates the electromotive force (emf) of the motor model. It uses the emf to further estimate the rotor position and rotor speed. The Flux Observer block uses identical inputs $(V_\alpha, V_\beta, I_\alpha, I_\beta)$ to estimate the stator flux, generated torque, and the rotor position.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- F28035 control card + DRV8312-C2-KIT inverter: mcb_pmsm_foc_sensorless_f28035
- F28335 control card + DRV8312-C2-KIT inverter: mcb_pmsm_foc_sensorless_f28335
- F28069m control card + DRV8312-C2-KIT inverter: mcb_pmsm_foc_sensorless_f28069m
- LAUNCHXL-F280049C controller + BOOSTXL-DRV8305 inverter: mcb_pmsm_foc_sensorless_f280049C
- F28027 launch pad + DRV8301/DRV8305EVM inverter: mcb_pmsm_foc_sensorless_f28027LaunchPad
- F28069m control card + TMDSHVMTRINSPIN: mcb_pmsm_foc_sensorless_hvkit_f28069m
- For LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Sensorless Field-Oriented Control of PMSM” (Motor Control Blockset).

For connections related to the preceding hardware configuration, see “Hardware Connections” on page 1-81.

Available Models

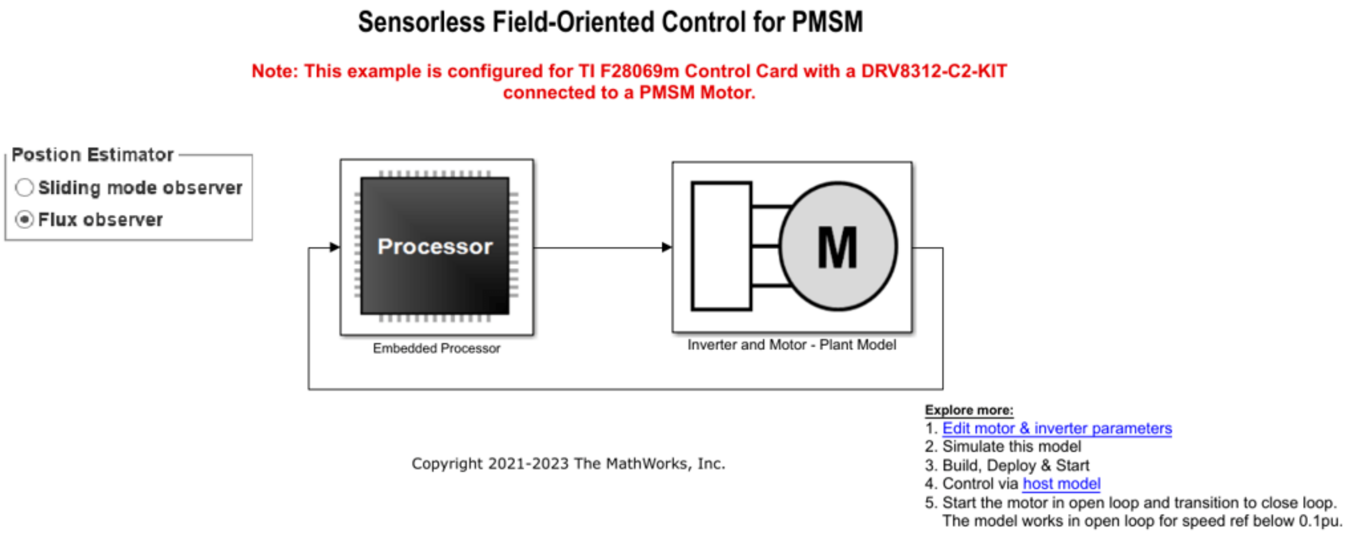
The example includes these models:

- mcb_pmsm_foc_sensorless_f28035
- mcb_pmsm_foc_sensorless_f28335
- mcb_pmsm_foc_sensorless_f28069m
- mcb_pmsm_foc_sensorless_f280049C
- mcb_pmsm_foc_sensorless_f28027LaunchPad
- mcb_pmsm_foc_sensorless_hvkit_f28069m

Note: For LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Sensorless Field-Oriented Control of PMSM” (Motor Control Blockset).

You can use these models for both simulation and code generation. You can also use the open_system command to open the Simulink® models. For example, use this command for a F28069m based controller.

```
open_system('mcb_pmsm_foc_sensorless_f28069m.slx');
```


Sensorless Field-Oriented Control for PMSM

Note: This example is configured for TI F28069m Control Card with a DRV8312-C2-KIT connected to a PMSM Motor.

Postion Estimator
( ) Sliding mode observer
(●) Flux observer

Copyright 2021-2023 The MathWorks, Inc.

Explore more:
1. Edit motor & inverter parameters
2. Simulate this model
3. Build, Deploy & Start
4. Control via host model
5. Start the motor in open loop and transition to close loop. The model works in open loop for speed ref below 0.1pu.

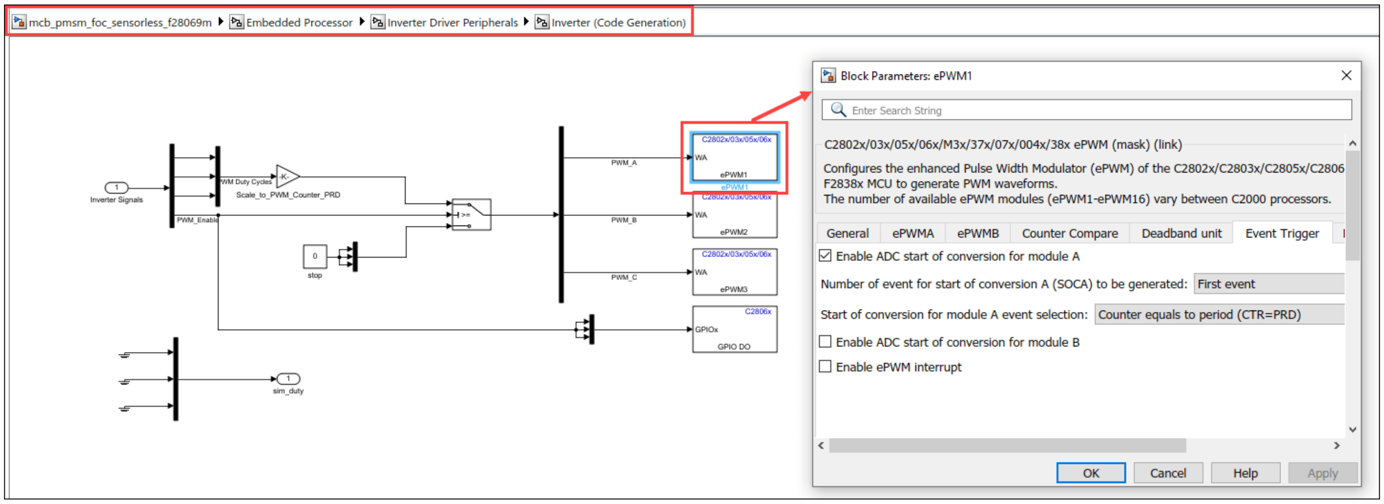
You may need to change the model parameters to fit your specific motor. Match motor voltage and power characteristics to the controller.

A conventional voltage-source inverter drives motor. The controller algorithm generates six pulse width modulation (PWM) signals using a vector PWM technique for six power switching devices. The inverter measures the current of the two motor inputs (ia and ib) input currents of the motor (ia and ib) using two analog-to-digital converters (ADCs) and sends the measurements to the processor.

Peripheral Block Configurations

Set the peripheral block configurations for this model. Double-click on the blocks to open block parameter configurations. You can use the same parameter values if you want to run this example for other hardware boards.

- **ePWM Block configuration**



Block Parameters: ePWM1

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the enhanced Pulse Width Modulator (ePWM) of the C2802x/C2803x/C2805x/C2806
F2838x MCU to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

☑ Enable ADC start of conversion for module A

Number of event for start of conversion A (SOCA) to be generated: First event

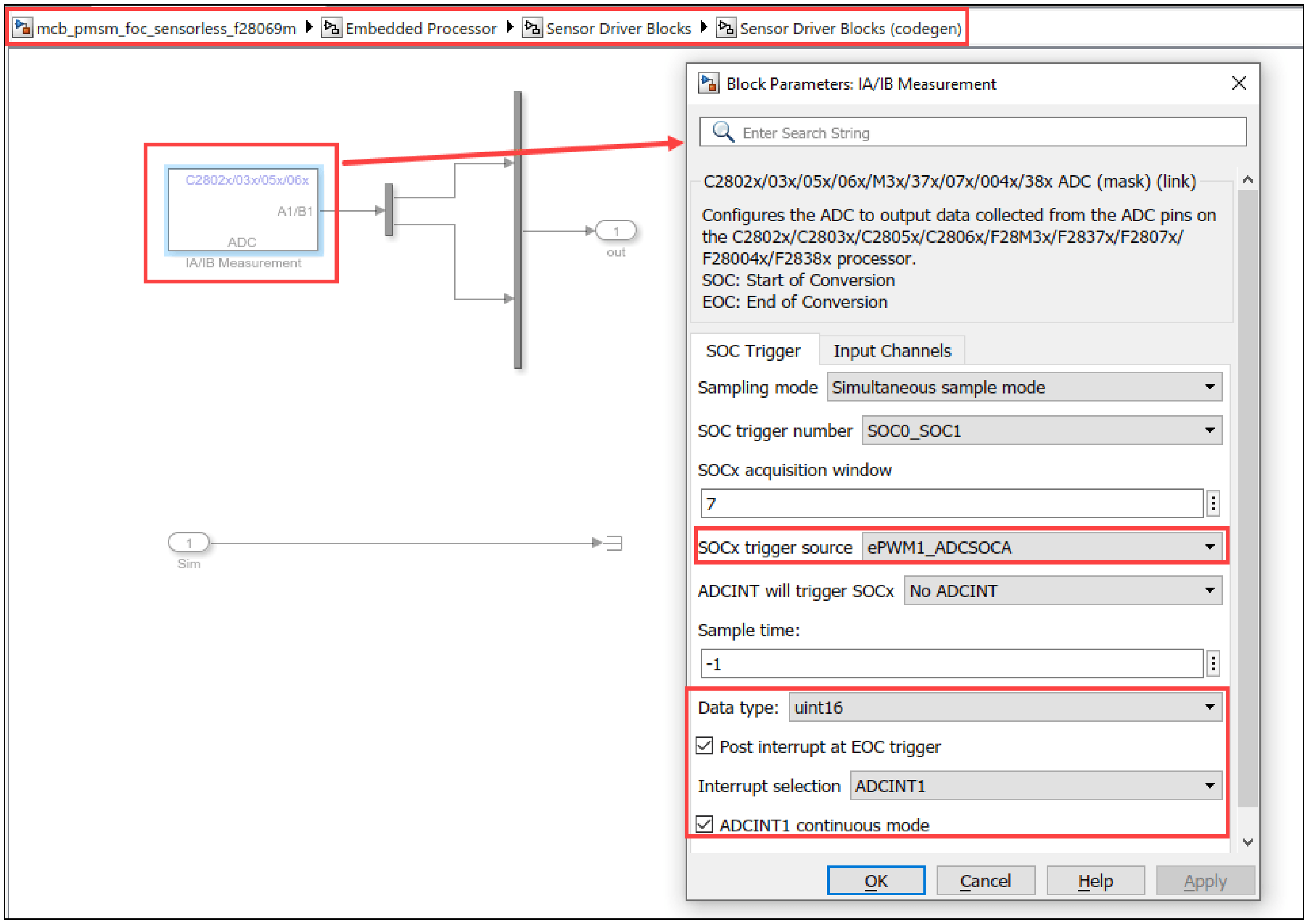
Start of conversion for module A event selection: Counter equals to period (CTR=PRD)

☐ Enable ADC start of conversion for module B

☐ Enable ePWM interrupt

- **ADC Block configuration**

The algorithm in this example uses an asynchronous scheduling. The pulse width modulation (PWM) block triggers the ADC conversion. At the end of conversion, the ADC posts an interrupt that triggers the main FOC algorithm. For more information, refer “ADC Interrupt Based Scheduling” on page 1-12.



C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ADC (mask) (link)

Configures the ADC to output data collected from the ADC pins on the C2802x/C2803x/C2805x/C2806x/F28M3x/F2837x/F2807x/F28004x/F2838x processor.
SOC: Start of Conversion
EOC: End of Conversion

Sampling mode: Simultaneous sample mode

SOC trigger number: SOC0_SOC1

SOCx acquisition window: 7

SOCx trigger source: ePWM1_ADCSOCA

ADCINT will trigger SOCx: No ADCINT

Sample time: -1

Data type: uint16

☑ Post interrupt at EOC trigger

Interrupt selection: ADCINT1

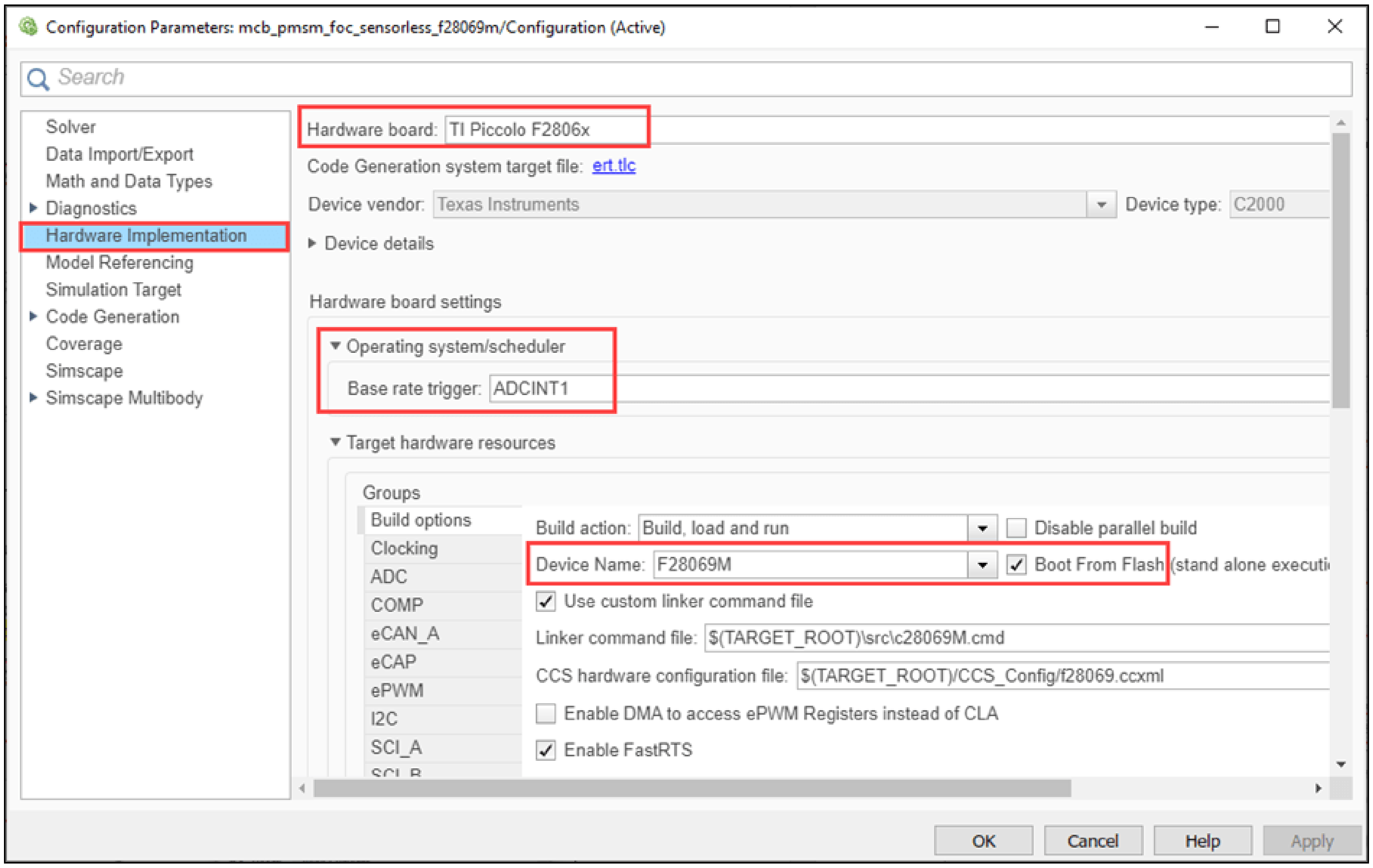
☑ ADCINT1 continuous mode

Configure the Model

1. Open the mcb_pmsm_foc_sensorless_f28069m model. This model is configured for TI Piccolo F28069x hardware.

2. To run the model on other TI C2000 processors, first press **Ctrl+E** to open the Configuration Parameters dialog box. Then, select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

3. The following screenshots show the scheduler configurations performed in the model. You can use the same parameter values if you want to run this example for other hardware boards.


Configuration Parameters: mcb_pmsm_foc_sensorless_f28069m/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- Coverage
- Simscape
- Simscape Multibody

Hardware board: TI Piccolo F2806x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Device details

Hardware board settings

Operating system/scheduler

Base rate trigger: ADCINT1

Target hardware resources

Groups: Build options, Clocking, ADC, COMP, eCAN_A, eCAP, ePWM, I2C, SCI_A, SCI_B

Build action: Build, load and run

☐ Disable parallel build

Device Name: F28069M

✓ Boot From Flash (stand alone executi

✓ Use custom linker command file

Linker command file: $(TARGET_ROOT)\src\c28069M.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28069.ccxml

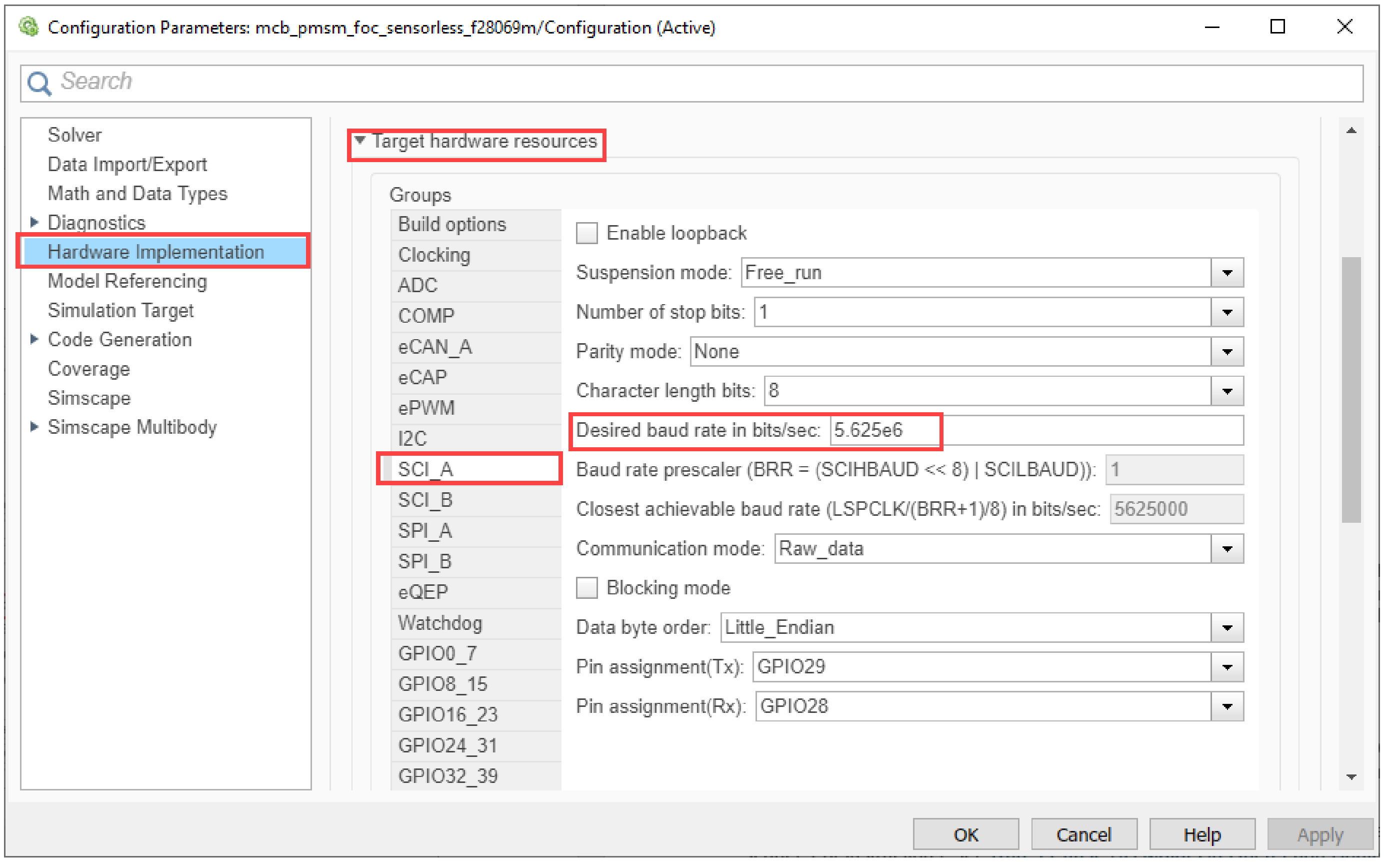
☐ Enable DMA to access ePWM Registers instead of CLA

✓ Enable FastRTS

Note:

- Sampling rate of the ADC block should be same as the base rate of the model determined by the PWM period of the ePWM block.
- Base rate trigger selection should be same as the interrupt triggered by the ADC module. For more information, see Model Configuration Parameters for Texas Instruments C2000 Processors.
- Ensure that the **Default parameter behavior** (Configuration Parameters > Code Generation > Optimization) is set to **Inlined**.

4. Ensure that the baud rate is set to 5.625e6 bits/sec.



Enable loopback

Suspension mode: Free_run

Number of stop bits: 1

Parity mode: None

Character length bits: 8

Desired baud rate in bits/sec: 5.625e6

Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)): 1

Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec: 5625000

Communication mode: Raw_data

Blocking mode

Data byte order: Little_Endian

Pin assignment(Tx): GPIO29

Pin assignment(Rx): GPIO28

Note:

- For F28335 processor you need to use external FTDI for serial communication.
- For F28035/F28335 processors sensorless example, the motor spins at a default speed of 0.5 per unit.

Required MathWorks® Products

To simulate model:

For the models: **mcb_pmsm_foc_sensorless_f28035** and **mcb_pmsm_foc_sensorless_f28335**

- Motor Control Blockset™
- Fixed-Point Designer™

To generate code and deploy model:

For the models: **mcb_pmsm_foc_sensorless_f28035** and **mcb_pmsm_foc_sensorless_f28335**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

Prerequisites

- If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see “Estimate Control Gains and Use Utility Functions” (Motor Control Blockset).

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the Analog-to-Digital Converter (ADC) or current offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

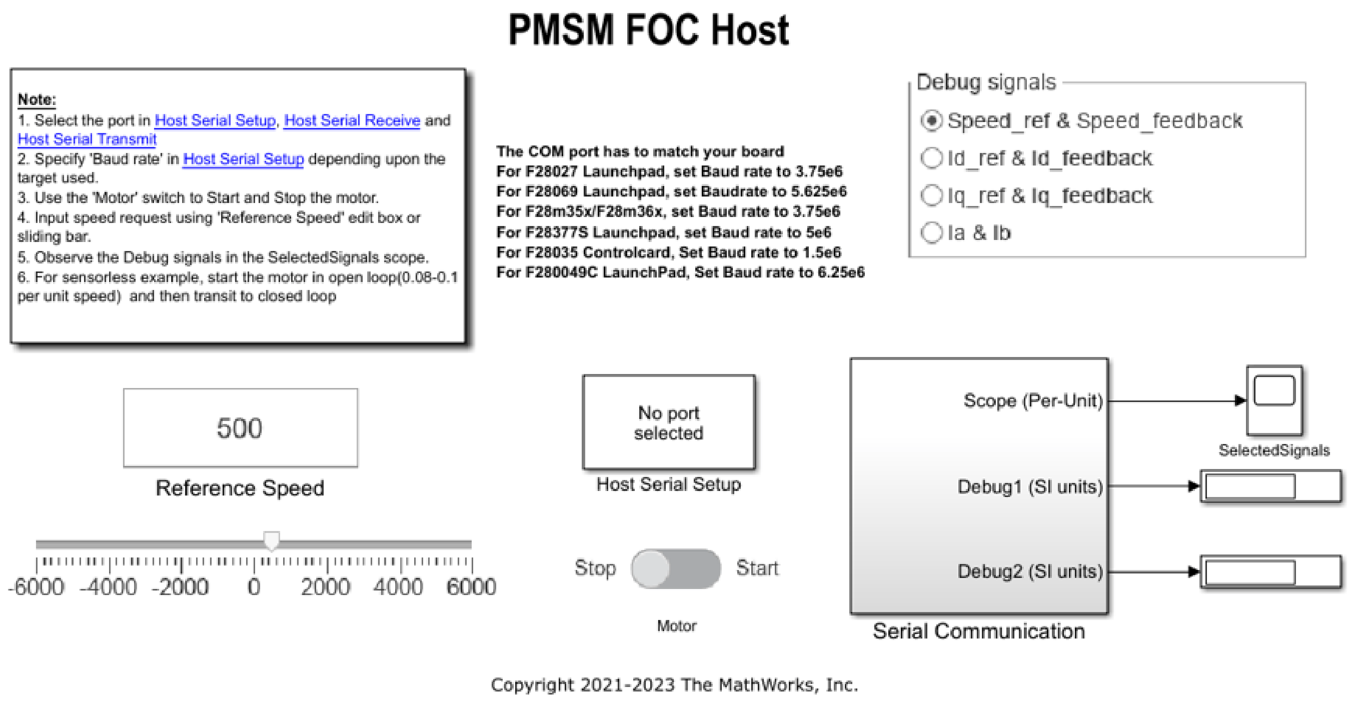
Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see “Open-Loop Control of 3-Phase AC Motors Using C2000 Processors” on page 4-171.

4. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the model, see “Model Configuration Parameters” (Motor Control Blockset).

5. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

6. In the target model, click the **host model** hyperlink to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_foc_host_model.slx');
```



PMSM FOC Host

Note:
1. Select the port in Host Serial Setup, Host Serial Receive and Host Serial Transmit
2. Specify 'Baud rate' in Host Serial Setup depending upon the target used.
3. Use the 'Motor' switch to Start and Stop the motor.
4. Input speed request using 'Reference Speed' edit box or sliding bar.
5. Observe the Debug signals in the SelectedSignals scope.
6. For sensorless example, start the motor in open loop(0.08-0.1 per unit speed) and then transit to closed loop

The COM port has to match your board
For F28027 Launchpad, set Baud rate to 3.75e6
For F28069 Launchpad, set Baudrate to 5.625e6
For F28m35x/F28m36x, set Baud rate to 3.75e6
For F28377S Launchpad, set Baud rate to 5e6
For F28035 Controlcard, Set Baud rate to 1.5e6
For F280049C LaunchPad, Set Baud rate to 6.25e6

Debug signals
[✓ Speed_ref & Speed_feedback]
Id_ref & Id_feedback
Iq_ref & Iq_feedback
Ia & Ib

Reference Speed: 500

Host Serial Setup: No port selected

Copyright 2021-2023 The MathWorks, Inc.

For details about the serial communication between the host and target models, see “Host-Target Communication” (Motor Control Blockset).

7. Set the parameter **Port** of the following blocks in the mcb_pmsm_foc_host_model model to match the COM port of the host computer:

- mcb_pmsm_foc_host_model > Host Serial Setup
- mcb_pmsm_foc_host_model > Serial Communication > Host Serial Receive
- mcb_pmsm_foc_host_model > Serial Communication > SCI_TX > Host Serial Transmit

8. Update the Reference Speed value in the host model.

NOTE: Before you run the motor at the required Reference Speed (by using either Sliding Mode Observer or Flux Observer), start running the motor at 0.1 x `pmsm.N_base` speed by using open-loop control. Then transition to closed-loop control by increasing the speed to 0.25 x `pmsm.N_base` (where, `pmsm.N_base` is the MATLAB workspace variable for base speed of the motor).

9. Click **Run** on the **Simulation** tab to run the host model.

10. Change the position of the motor switch to Start, to start running the motor in the open-loop condition.

NOTE: Do not run the motor (using this example) in the open-loop condition for a long time duration. The motor may draw high currents and produce excessive heat.

We designed the open-loop control to run the motor with a Reference Speed that is less than or equal to 10% of base speed.

When you run this example on the hardware at a low Reference Speed, due to a known issue, the PMSM may not follow the low Reference Speed.

11. Increase the motor Reference Speed beyond 10% of base speed to switch from open-loop to closed-loop control.

NOTE: To change the motor's direction of rotation, reduce the motor Reference Speed to a value less than 10% of the base speed. This brings the motor back to open-loop condition. Change the direction of rotation but keep the Reference Speed magnitude as constant. Then transition to the closed-loop condition.

12. Observe the debug signals from serial communication , in the Time Scope of host model.

Other Things to Try

- You can use SoC Blockset™ to implement a sensorless closed-loop motor control application that addresses challenges related to ADC-PWM synchronization, controller response, and studying different PWM settings. For details, see “Partition Motor Control for Multiprocessor MCUs” on page 4-194.
- You can also use SoC Blockset™ to develop a sensorless real-time motor control application that utilizes multiple processor cores to obtain design modularity, improved controller performance, and other design goals. For details, see “Integrate MCU Scheduling and Peripherals in Motor Control Application” on page 4-202.
- For LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Sensorless Field-Oriented Control of PMSM” (Motor Control Blockset).

ADC-PWM Synchronization Using ADC Interrupt

This example shows how to use the ADC block to sample an analog voltage and use the PWM block to generate a pulse waveform. This example also shows how to use the Hardware Interrupt block to synchronize the change in the PWM duty cycle with analog to digital conversion of voltage. In the generated code, changes in the voltage of the ADC input alter the duty cycle of the PWM output. The period of the PWM waveform remains constant.

You can also use the model c2838x_adcpwmasync_TopModel.slx to perform Model Reference workflow. For more information, see

Required Hardware

- Spectrum Digital F2808/F2812/F28335 eZdsp or Texas Instruments™ LaunchPad/controlSTICK/controlCARD with docking station
- Oscilloscope and probes
- Function generator

Hardware Connections

Connect the function generator output to the ADC input (ADCINA0) on the board. Connect the GPIO pin corresponding to PWM1A to the analog input of the oscilloscope.

Available Models

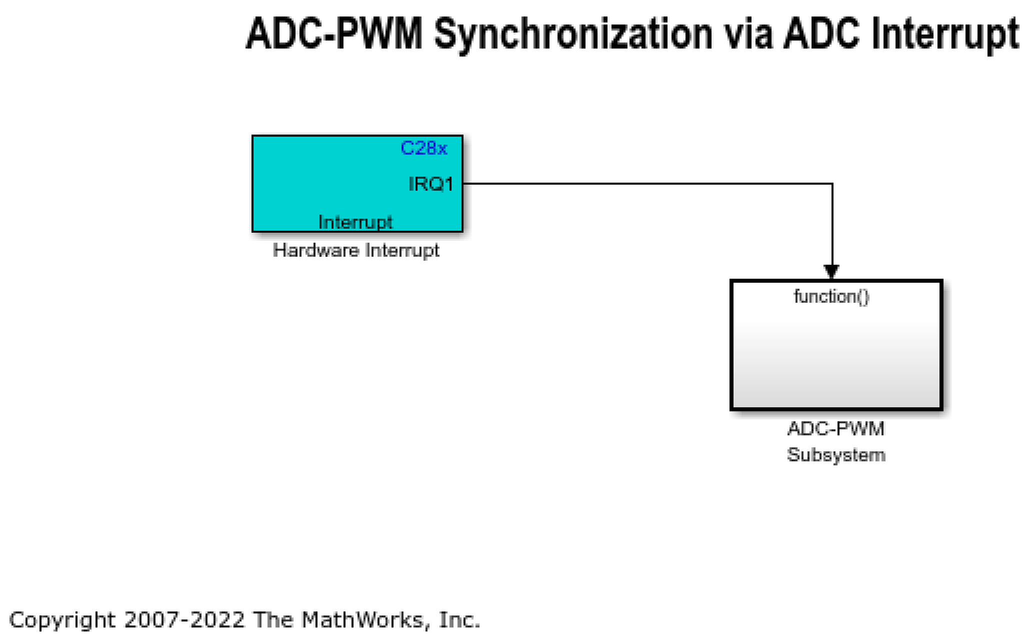
These are the Simulink models available for different C2000 processors:

- F281x-based board: c281x_adcpwmasynctest_ert.slx
- F280x/F2823x/F2833x-based board: c280x_2833x_adcpwmasynctest_ert.slx
- Piccolo F2802x/F2803x/F2806x or Concerto F28M35x/F28M36x-based board: c280xx_adcpwmasynctest_ert.slx
- Piccolo F2807x or Delfino F2837xS/F2837xD-based board: c2807x_2837xx_adcpwmasynctest_ert.slx
- Piccolo F28004x-based board: c28004x_adcpwmasynctest_ert.slx
- Model Reference workflow: c2838x_adcpwmasync_TopModel.slx

Note: To use the F28M35x/F28M36x controlCARD, you need C2000™ Microcontroller Blockset.

Model

The following figure shows the example model.


ADC-PWM Synchronization via ADC Interrupt
C28x
IRQ1
Interrupt
Hardware Interrupt
function()
ADC-PWM
Subsystem
Copyright 2007-2022 The MathWorks, Inc.

The analog voltage from the function generator controls the duty cycle of the PWM waveform. Duty cycle changes can be observed on the oscilloscope. The Hardware Interrupt block triggers an interrupt service routine (ISR) and schedules the execution of the connected subsystem (ADC-PWM Subsystem) when the processor receives the ADC interrupt (ADCINT).

The ADC-PWM Subsystem consists of an ADC block that drives the duty cycle input port of the PWM block. The PWM block is configured to trigger the start of conversion (SOC) of the ADC block.

Run the Model on the Hardware Board

1. Open the model corresponding to the target hardware you are using. Each model is configured for a default target hardware. To select a different target hardware, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**
2. On the **Hardware** tab, Click **Build, Deploy & Start** > **Build Stand-Alone** to generate, build, load, and run the program.
3. Observe the changes to the PWM waveform on the oscilloscope.

More About

- C281x ADC
- C281x PWM
- c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM
- C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ADC
- C28x Hardware Interrupt
-

Closed Loop Control of a DC-DC Buck Converter

This example shows how to model a closed loop control of a DC-DC buck converter in the C2000™ Microcontroller Blockset. The model runs on a F28379D Launchpad connected to the C2000 digital power supply booster pack.

Using this example you can model:

- Digital DC-DC synchronous buck converter voltage mode control (VMC).
- Digital DC-DC synchronous buck converter average current mode control using control law accelerator (CLA).

By running the models provided in this example on the host computer, you can:

- Simulate a controller for the DC-DC buck converter plant model.
- Generate code for the controller and load it on the LaunchPad.
- Monitor signals and tune parameters on the host computer.

Required Hardware

- F28379D LaunchPad
- Digital Power Buck Converter BoosterPack

Hardware Connections

Connect the Digital Power Buck Converter BoosterPack (BOOSTXL-BUCKCONV) to the Texas Instruments Delfino F28379D LaunchPad as detailed in the following table.

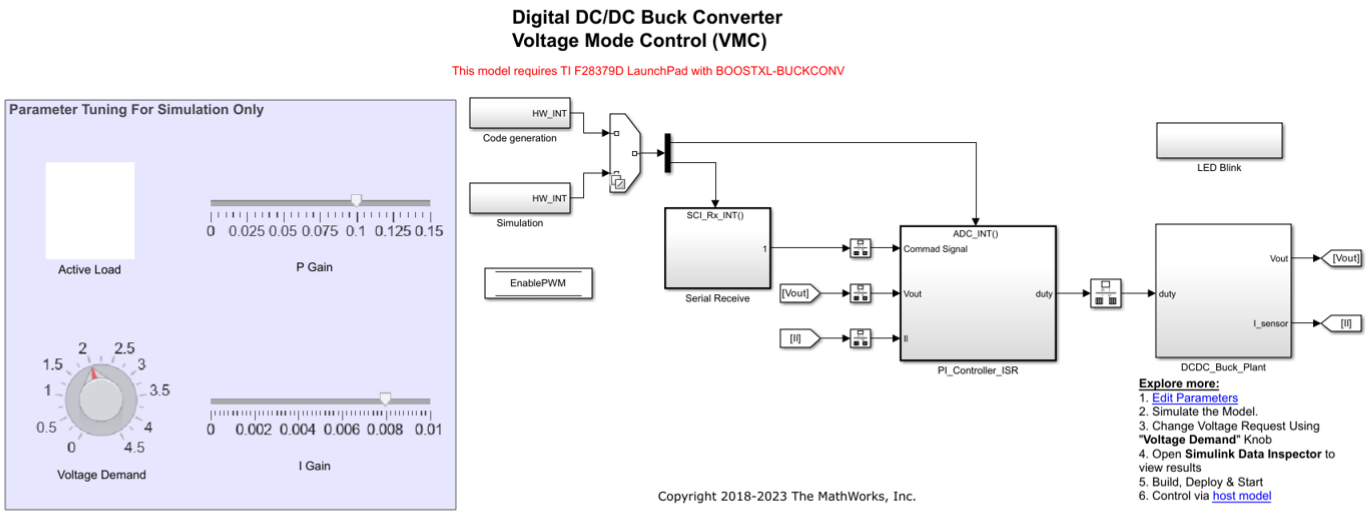
BOOSTXL-BUCKCONV	**LAUNCHXL-F28379D Site 2 (J5-J8)**	**Description**
H3[1] PWM-HI	J8[80] EPWM4A	High-side drive signal for synchronous buck
H3[2] PWM-LO	J8[79] EPWM4_B	Low-side drive signal for synchronous buck
H3[5] PWM Load	J8[76] GPIO10	PWM duty control signal for active load
H2[7] VoutFB	J7[67] ADCINC4	Output Voltage feedback
H2[9] ILFB	J7[69] ADCINA4	Inductor current feedback
H2[4] ILFB_AVG	J7[64] ADCINC5	Inductor current feedback (heavily filtered average)

Available Models

- f28379D_DCDC_Buck.slx and f28379D_DCDC_Buck_CLA.slx can be used to simulate the controller, generate code and load it on the F8379D LaunchPad.
- matlab:c2000_DCDC_Buck_host_model.slx and matlab:c2000_DCDC_Buck_CLA_host_model.slx can be run on the host computer to log signals and tune parameters.

You can use these models for both simulation and code generation. You can also use the `open_system` command to open the Simulink® models. For example, use this command for a F28379D based controller.

```
open_system('f28379D_DCDC_Buck.slx');
```


Digital DC/DC Buck Converter
Voltage Mode Control (VMC)
This model requires TI F28379D LaunchPad with BOOSTXL-BUCKCONV
Parameter Tuning For Simulation Only
Active Load
P Gain
Voltage Demand
I Gain
HW_INT
Code generation
Simulation
EnablePWM
SCI_Rx_INT()
Serial Receive
ADC_INT()
Commad Signal
Vout
Il
duty
PI_Controller_ISR
LED Blink
DCDC_Buck_Plant
I_sensor
[Vout]
[Il]
Explore more:
1. Edit Parameters
2. Simulate the Model.
3. Change Voltage Request Using "Voltage Demand" Knob
4. Open Simulink Data Inspector to view results
5. Build, Deploy & Start
6. Control via host model
Copyright 2018-2023 The MathWorks, Inc.

The f28379D_DCDC_Buck.slx and f28379D_DCDC_Buck_CLA.slx model consists of these subsystems:

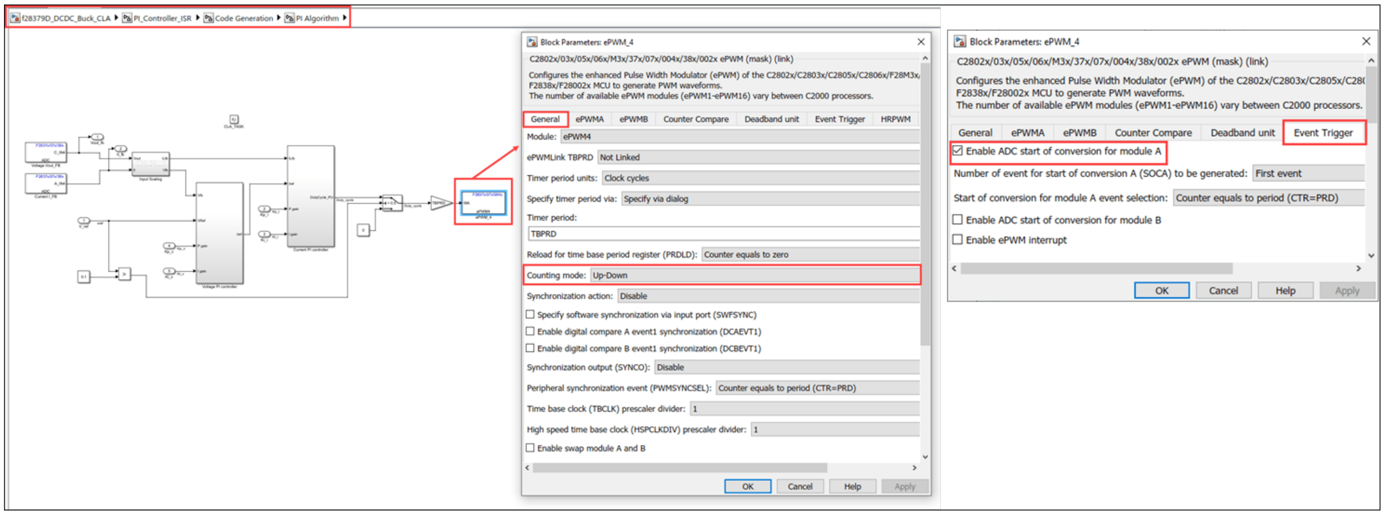
- **Controller**:
- Voltage mode control - f28379D_DCDC_Buck.slx model employs a discrete proportional integral (PI) controller to minimize the error between the reference voltage and the output voltage. The duty cycle of the PI controller is limited to 60% of the PWM time period.
- Average current mode control - f28379D_DCDC_Buck_CLA.slx model employs a discrete proportional integral (PI) controller to control the average inductor current, which produces the duty cycle. The current controller reference is the output of the voltage controller.
- **DC-DC Buck Converter**: Simscape™ blocks are used to model the DC-DC buck converter circuitry.
- **Dashboard Controls**: Used to set the reference voltage, switch the active load on and off active load, and tune proportional and integral gains.

Peripheral Block Configurations

Set the peripheral block configurations for this model. Double-click on the blocks to open block parameter configurations. You can use the same parameter values if you want to run this example for other hardware boards. The peripheral block configurations shown are for the f28379D_DCDC_Buck_CLA.slx model.

- **ePWM Block**

The ePWM block (in PI_Controller_ISR > Code Generation > PI Algrotihm) is configured to operate in **Up-Down** mode, and it triggers the start of conversion for ADC when the ePWM counter reaches the timer period.


f28379D_DCDC_Buck_CLA ▸ PI_Controller_ISR ▸ Code Generation ▸ PI Algorithm ▸

Block Parameters: ePWM_4

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x/002x ePWM (mask) (link)

Configures the enhanced Pulse Width Modulator (ePWM) of the C2802x/C2803x/C2805x/C2806x/F28M3x/
F2838x/F28002x MCU to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger | HRPWM

Module: ePWM4

ePWMLink TBPRD: Not Linked

Timer period units: Clock cycles

Specify timer period via: Specify via dialog

Timer period: TBPRD

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up-Down

Synchronization action: Disable

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

☐ Enable digital compare B event1 synchronization (DCBEVT1)

Synchronization output (SYNCO): Disable

Peripheral synchronization event (PWMSYNCSEL): Counter equals to period (CTR=PRD)

Time base clock (TBCLK) prescaler divider: 1

High speed time base clock (HSPCLKDIV) prescaler divider: 1

☐ Enable swap module A and B

OK | Cancel | Help | Apply

Block Parameters: ePWM_4

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x/002x ePWM (mask) (link)

Configures the enhanced Pulse Width Modulator (ePWM) of the C2802x/C2803x/C2805x/C280
F2838x/F28002x MCU to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

☑ Enable ADC start of conversion for module A

Number of event for start of conversion A (SOCA) to be generated: First event

Start of conversion for module A event selection: Counter equals to period (CTR=PRD)

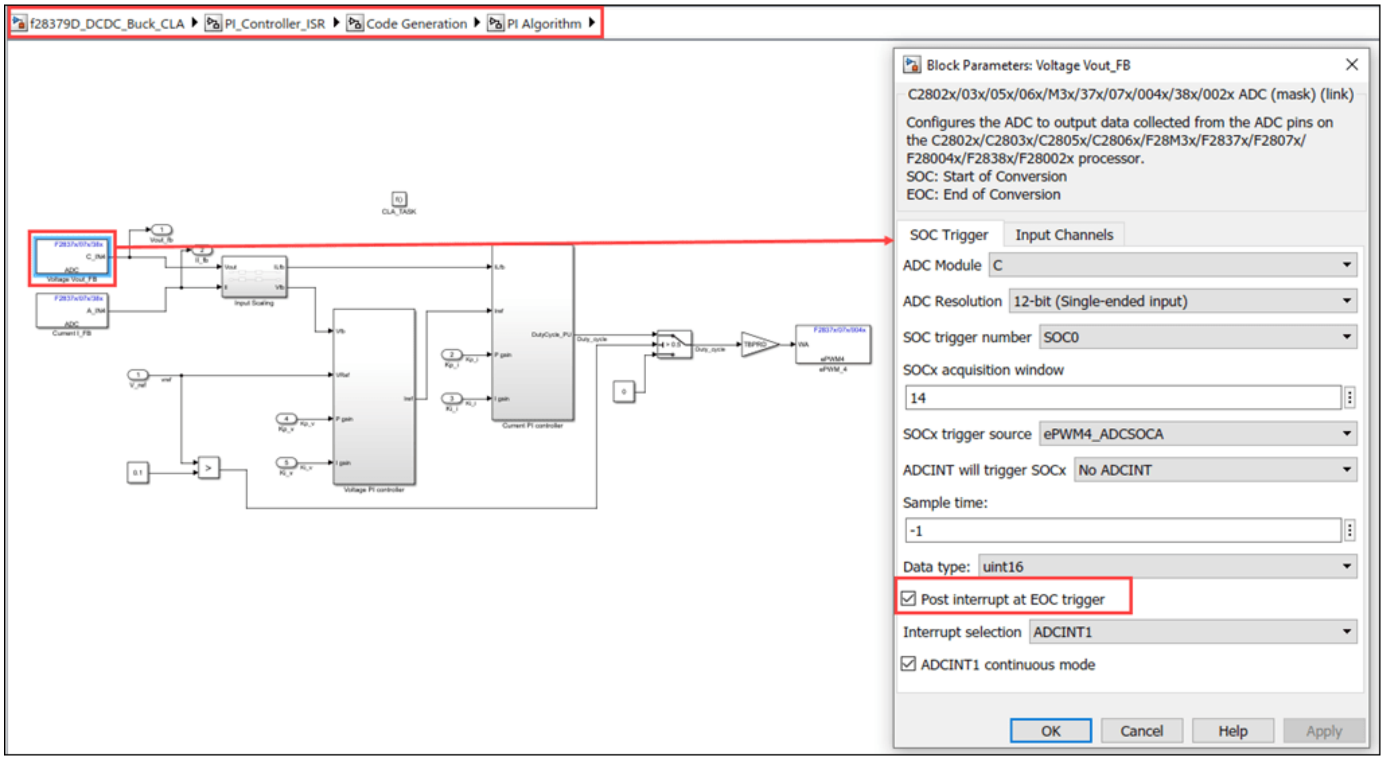
☐ Enable ADC start of conversion for module B

☐ Enable ePWM interrupt

OK | Cancel | Help | Apply

- **ADC Block**

The ADC block (in PI_Controller_ISR > Code Generation > PI Algrotihm) is configured with the **Post interrupt at EOC trigger** parameter enabled.



Block Parameters: Voltage Vout_FB

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x/002x ADC (mask) (link)

Configures the ADC to output data collected from the ADC pins on the C2802x/C2803x/C2805x/C2806x/F28M3x/F2837x/F2807x/F28004x/F2838x/F28002x processor.
SOC: Start of Conversion
EOC: End of Conversion

SOC Trigger | Input Channels

ADC Module: C

ADC Resolution: 12-bit (Single-ended input)

SOC trigger number: SOC0

SOCx acquisition window: 14

SOCx trigger source: ePWM4_ADCSOCA

ADCINT will trigger SOCx: No ADCINT

Sample time: -1

Data type: uint16

☑ Post interrupt at EOC trigger

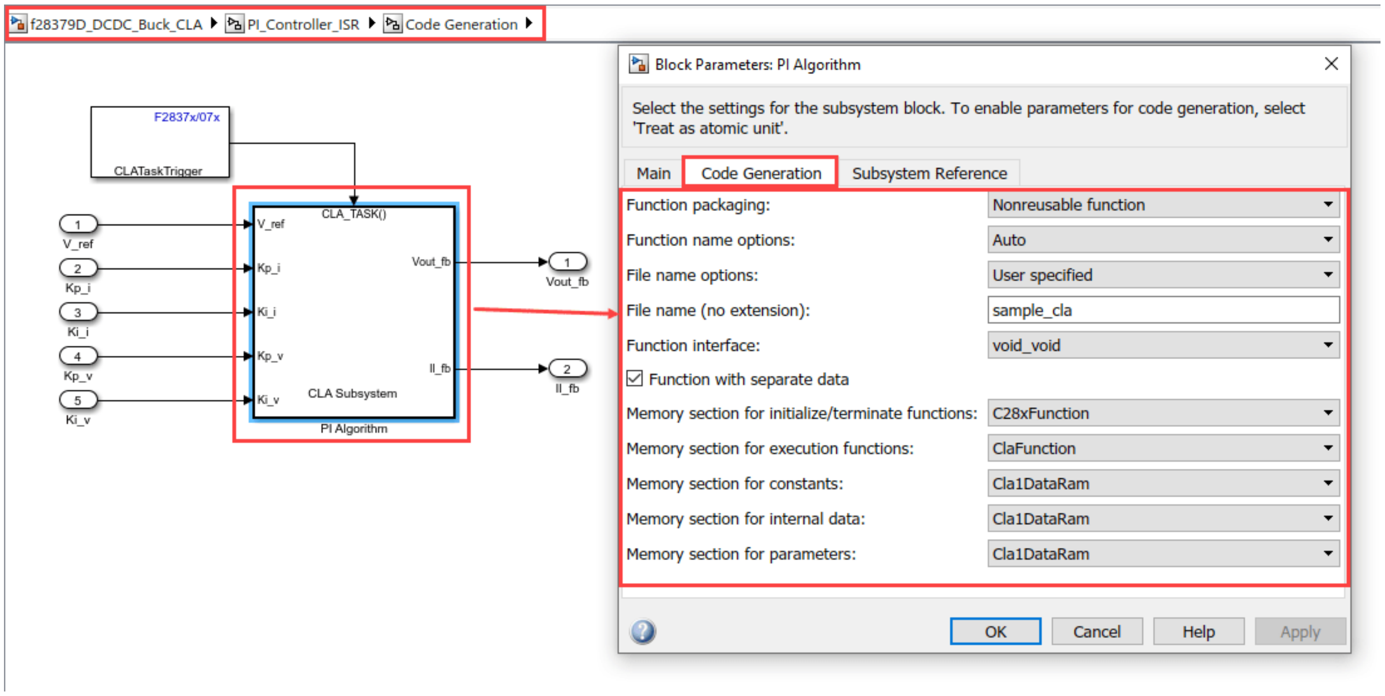
Interrupt selection: ADCINT1

☑ ADCINT1 continuous mode

OK | Cancel | Help | Apply

- **CLA Subsystem Block**

Inputs to the PI Algorithm block are stored in the `CpuToCla1MsgRAM`. CLA configuration is applicable only for the f28379D_DCDC_Buck_CLA.slx model. For more information, refer to "Overview of CLA Configuration for C2000 Processors Using Subsystem" on page 1-93.



Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Function packaging:	Nonreusable function
Function name options:	Auto
File name options:	User specified
File name (no extension):	sample_cla
Function interface:	void_void
☑ Function with separate data	
Memory section for initialize/terminate functions:	C28xFunction
Memory section for execution functions:	ClaFunction
Memory section for constants:	Cla1DataRam
Memory section for internal data:	Cla1DataRam
Memory section for parameters:	Cla1DataRam

Simulate Controller for DC/DC Buck Converter Plant Model

Run the Model

1. Open the f28379D_DCDC_Buck.slx or f28379D_DCDC_Buck_CLA.slx model.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results. You can tune the input parameters using the dashboard controls.

Generate Code for Controller and Load It on LaunchPad

In this section, you generate code and load it on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the BoostXL-BUCKCONV in a closed-loop control.

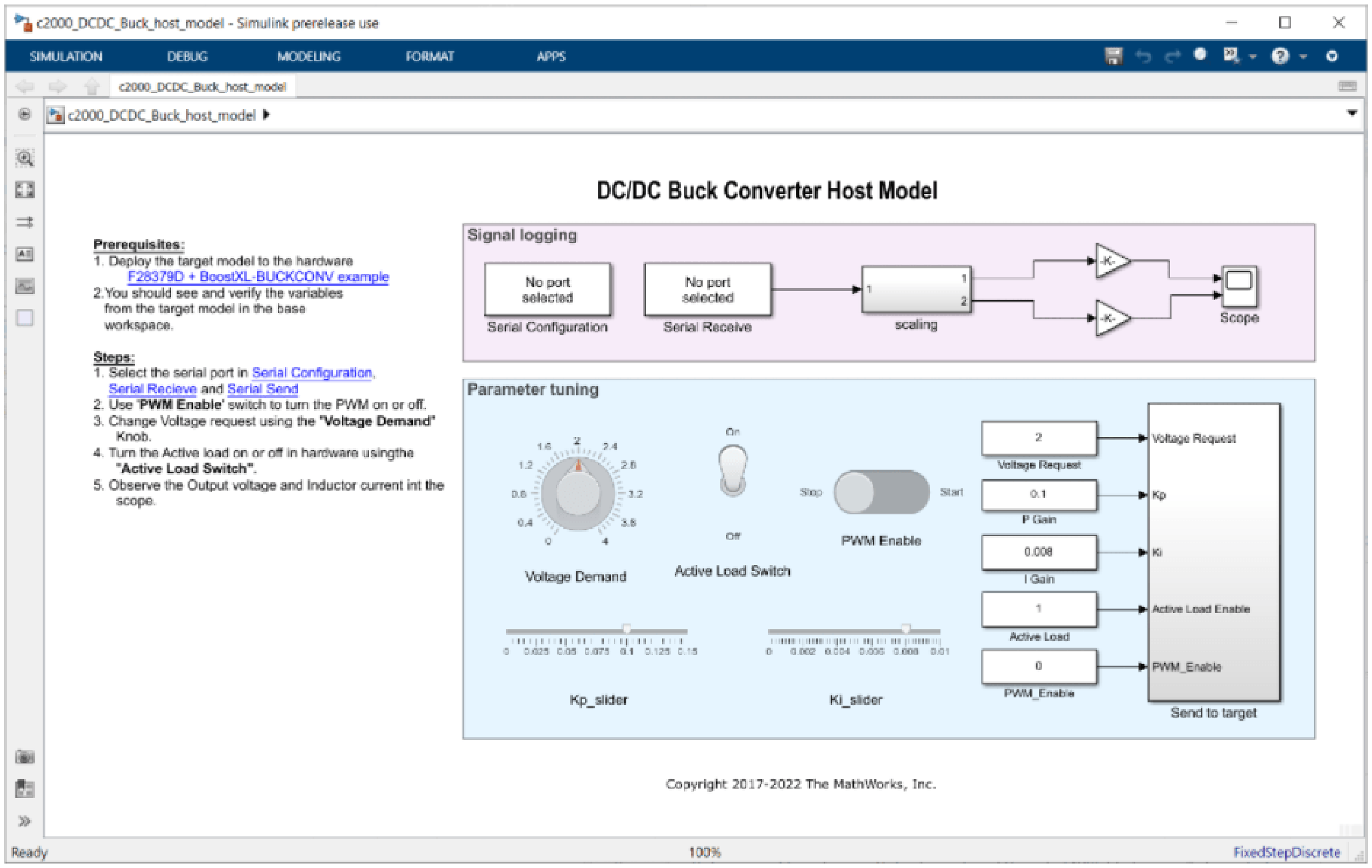
Run Model on LaunchPad

1. Open the f28379D_DCDC_Buck or f28379D_DCDC_Buck_CLA model and generate code by pressing **Ctrl+B** or clicking **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

Note: On the F28379D processor, this example runs on CPU1. Ensure that the program running on CPU2 is not using the peripherals that are used by CPU1.

2. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a host model.

open_system('c2000_DCDC_Buck_host_model.slx');



Prerequisites:
1. Deploy the target model to the hardware F28379D + BoostXL-BUCKCONV example
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select the serial port in Serial Configuration, Serial Recieve and Serial Send
2. Use '**PWM Enable**' switch to turn the PWM on or off.
3. Change Voltage request using the **"Voltage Demand"** Knob.
4. Turn the Active load on or off in hardware usingthe **"Active Load Switch"**.
5. Observe the Output voltage and Inductor current int the scope.

Copyright 2017-2022 The MathWorks, Inc.

Monitor Signals and Tune Parameters on Host Computer

Configure and Run Model on Host Computer

1. On the host computer, browse to **Device Manager** > **Ports (COM & LPT)** to find the COM port.

2. Set the parameter **Port** of the following blocks in the c2000_DCDC_Buck_host_model model to match the COM port of the host computer:

- c2000_DCDC_Buck_host_model > Serial Configuration
- c2000_DCDC_Buck_host_model > Serial Receive
- c2000_DCDC_Buck_host_model > Send to target > Serial Send > Serial Send

3. Click the **Run** button on the **Simulation** tab to run the host model.

4. While the model runs, you can tune parameters using the dashboard blocks such as Voltage Demand and Controller gains.

Note: If you encounter any non-real time communication between the host and the target, try closing the Scope block and run the simulation of the host model. You might encounter lag in uploading the data in the Scope block, if you have other applications running simultaneously in your machine.

For more information on using the serial connection for your hardware, see “Set Up Serial Communication with Target Hardware” on page 1-6.

Monitor the Signals

While the model runs, you can monitor the following signals on the scope:

- **Voltage Vout_FB**: The measured output voltage of the system. 4095 ADC counts is equivalent to 6.0909 V output voltage.
- **Current I_FB**: The real-time current flowing through the inductor (L1). 4095 ADC counts is equivalent to 6.8333 A inductor current.

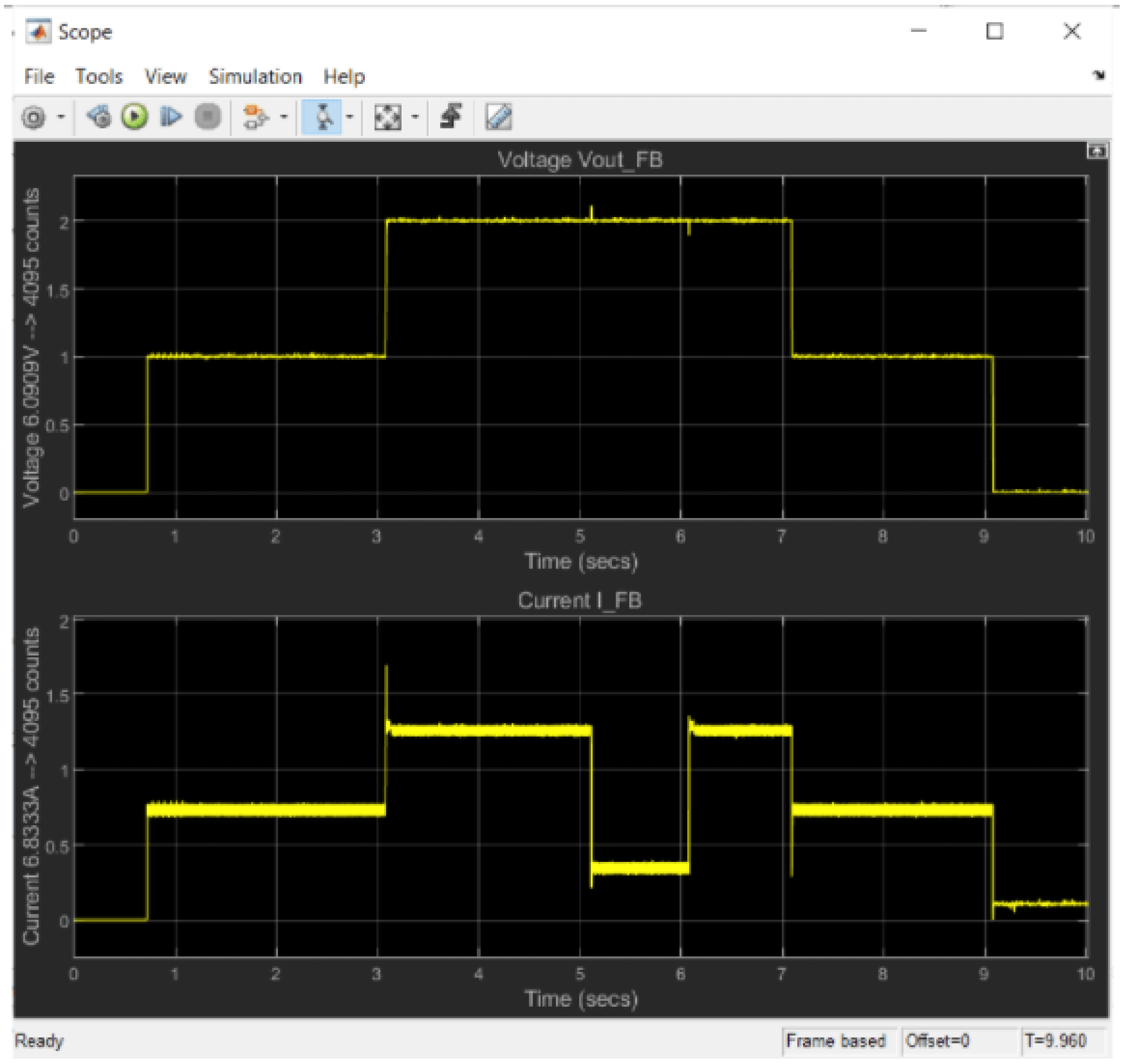
Tune the Parameters

While the model runs, you can tune parameters using the following dashboard blocks:

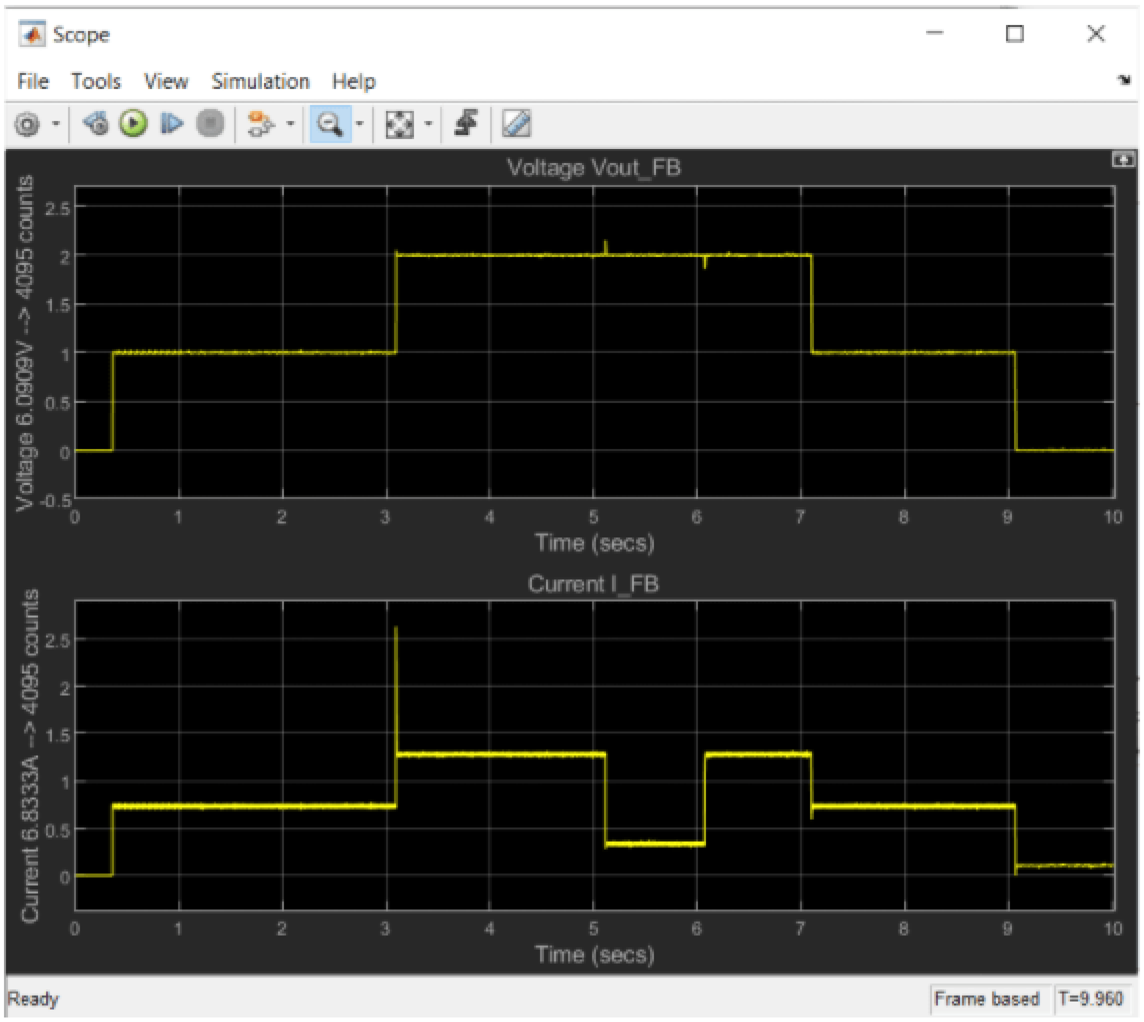
- **Voltage Demand**: Change the output voltage demand. This parameter is the main request for the control loop. The controller algorithm compares the voltage request value with the measured output voltage and adjusts the PWM duty cycle towards achieving the output voltage.
- **Active Load Switch**: Use this switch to turn on the active load present on the hardware on or off. This allows you to add an extra load resistor to study the effect of abrupt changes in the load circuit.
- **PWM Enable**: Use this slider to turn the PWM on or off for the synchronous buck converter.
- **Controller gains**: Change the controller gains (Kp and Ki values) using the available Slider blocks (Kp_slider and Ki_slider). You can change this parameter to study the robustness of the controller. Large, abrupt changes might lead to instability of the controller; apply changes smoothly.

Hardware Results

1. The following figure shows the hardware results for the model f28379D_DCDC_Buck.slx. Observe the output voltage and inductor current. You can also see the effect of voltage request change from 1 V to 2 V and effect of active load change.



2. The following figure shows the hardware results for the model f28379D_DCDC_Buck_CLA.slx Observe the output voltage and inductor current. You can also see the effect of voltage request change from 1 V to 2 V and effect of active load change.



Other Things to Try

- Run the example on the TI Delfino F280049C LaunchPad and analyze the results.
- Change the Kp and Ki values and observe the results.

More About

C28x SCI Receive

Digital DC/DC Buck Converter Using Peak Current Mode Control

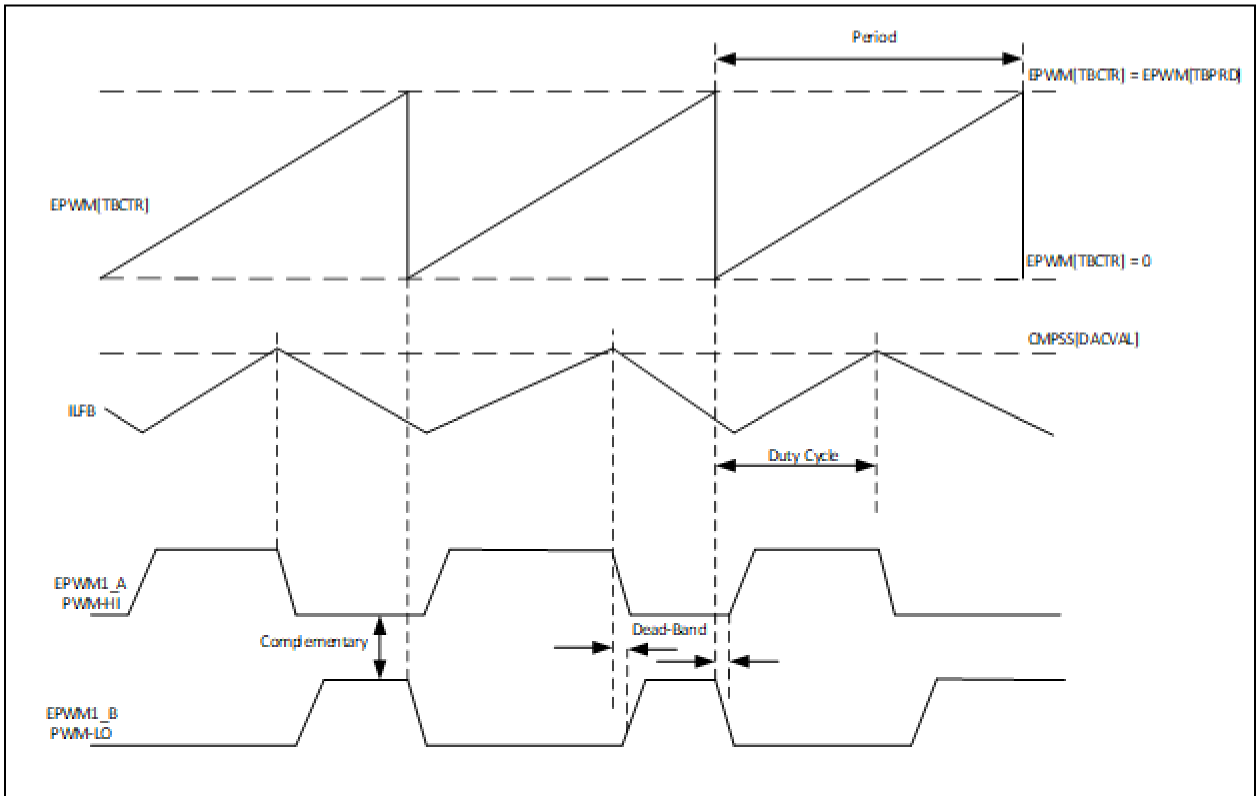
This example shows how to use the Comparator Subsystem (CMPSS) to regulate buck converter output voltage (BOOSTXL-BUCKCONV) using Peak Current Mode Control (PCMC) for Texas Instruments™ C2000™ Microcontroller Blockset. Using this example, you can:

- Configure the CMPSS to monitor the inductor current sense feedback (ILFB) signal.
- Configure the enhanced Pulse Width Modulator (ePWM) to trip on an overcurrent condition.
- Generate code for the controller and load it on the hardware board.
- Monitor signals and tune parameters on the host computer.

Introduction

This example uses a CMPSS, an ePWM, and ADC (Analog Digital Converter) subsystem to perform the synchronous buck operation in PCMC.

- Each PWM cycle starts by driving the ePWM high (PWM-HI) until the peak current threshold is reached after which the PWM-HI signal is driven low for the rest of the PWM cycle.
- The ePWM1 module triggers ADC conversions to sense the plant feedback signals at a frequency of 200kHz.
- The ADC ISR calculates changes to the peak current limit.
- An on-chip analog CMPSS is configured to continuously monitor the current sense feedback (ILFB) signal against set peak current limit.
- The Control Law Accelerator (CLA) executes the PCMC control algorithm.



PWM signal generated for Peak Current Mode Control

Required Hardware

- F280049C LaunchPad
- Digital Power Buck Converter BoosterPack
- 9V external power supply
- Micro-USB cable

Available Models

- `Digital DC/DC Buck Converter Peak Current Mode Control (PCMC)` can be used to generate code and load it on the F80049C LaunchPad.
- `Host PCMC Model` can be run on the host computer to log signals and tune parameters.

Hardware Connections

Connect the Digital Power Buck Converter BoosterPack (BOOSTXL-BUCKCONV) to the Texas Instruments Piccolo F280049C LaunchPad as detailed in the following table.

BOOSTXL-BUCKCONV	LAUNCHXL-F28379D Site 2 (J5-J8)	Description
H3[1] PWM-HI	J8[80] EPWM4A	High-side drive signal for synchronous buck
H3[2] PWM-LO	J8[79] EPWM4_B	Low-side drive signal for synchronous buck
H3[5] PWM Load	J8[76] GPIO10	PWM duty control signal for active load
H2[7] VoutFB	J7[67] ADCINC4	Output Voltage feedback
H2[9] ILFB	J7[69] ADCINA4	Inductor current feedback
H2[4] ILFB_AVG	J7[64] ADCINC5	Inductor current feedback (heavily filtered average)

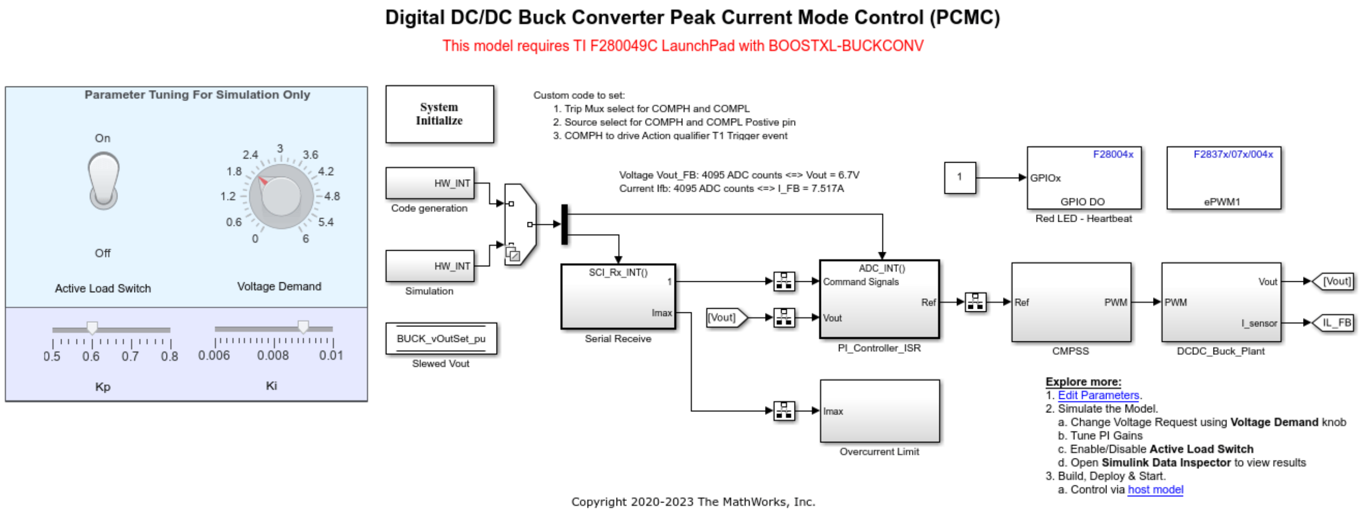
Note: Ensure that a jumper is present between pins **+3v3 TO J5** on the JP8 on the Launchpad.

Model

Digital DC/DC Buck Converter Peak Current Mode Control (PCMC) Model

To open the model, type the following command in the MATLAB® prompt:

```
open_system('f280049C_DCDC_Buck_PCMC');
```


Digital DC/DC Buck Converter Peak Current Mode Control (PCMC)

This model requires TI F280049C LaunchPad with BOOSTXL-BUCKCONV

Custom code to set:
1. Trip Mux select for COMPH and COMPL
2. Source select for COMPH and COMPL Postive pin
3. COMPH to drive Action qualifier T1 Trigger event

Voltage Vout_FB: 4095 ADC counts <=> Vout = 6.7V
Current Ifb: 4095 ADC counts <=> I_FB = 7.517A

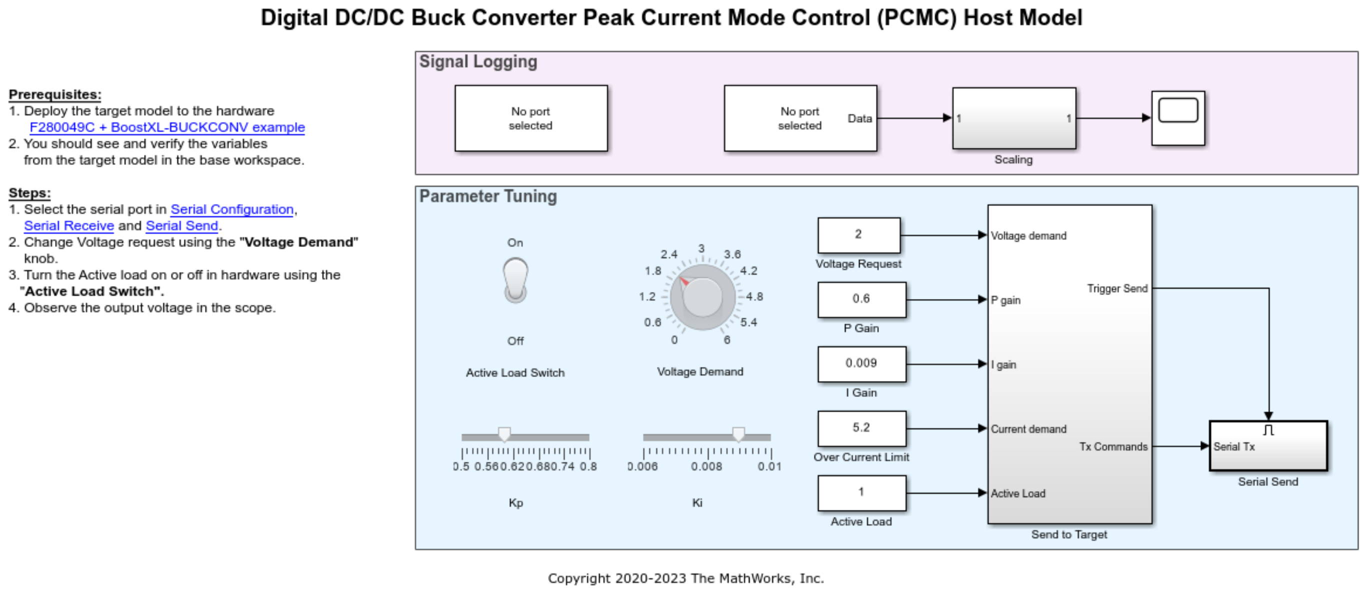
Explore more:
1. Edit Parameters.
2. Simulate the Model.
   a. Change Voltage Request using **Voltage Demand** knob
   b. Tune PI Gains
   c. Enable/Disable **Active Load Switch**
   d. Open **Simulink Data Inspector** to view results
3. Build, Deploy & Start.
   a. Control via host model

Copyright 2020-2023 The MathWorks, Inc.

C2000_host_read_PCMC Model

To open the model, type the following command in the MATLAB® prompt:

```
open_system('c2000_host_read_PCMC');
```



Digital DC/DC Buck Converter Peak Current Mode Control (PCMC) Host Model

Prerequisites:
1. Deploy the target model to the hardware F280049C + BoostXL-BUCKCONV example
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select the serial port in Serial Configuration, Serial Receive and Serial Send.
2. Change Voltage request using the **"Voltage Demand"** knob.
3. Turn the Active load on or off in hardware using the **"Active Load Switch"**.
4. Observe the output voltage in the scope.

Copyright 2020-2023 The MathWorks, Inc.

This example demonstrates the working of PCMC for a fixed input voltage of 9 V DC and an output voltage setpoint of 2 V DC. Hence a fixed slope compensation value is applicable for the above mentioned voltages is computed. For more information, refer to TI manual 'Digital Peak Current Mode Control With Slope Compensation'.

For simulation, the RAMP Generator has been implemented and SR flip flop block is used to generate PWMs to implement the comparator logic in Simulink®.

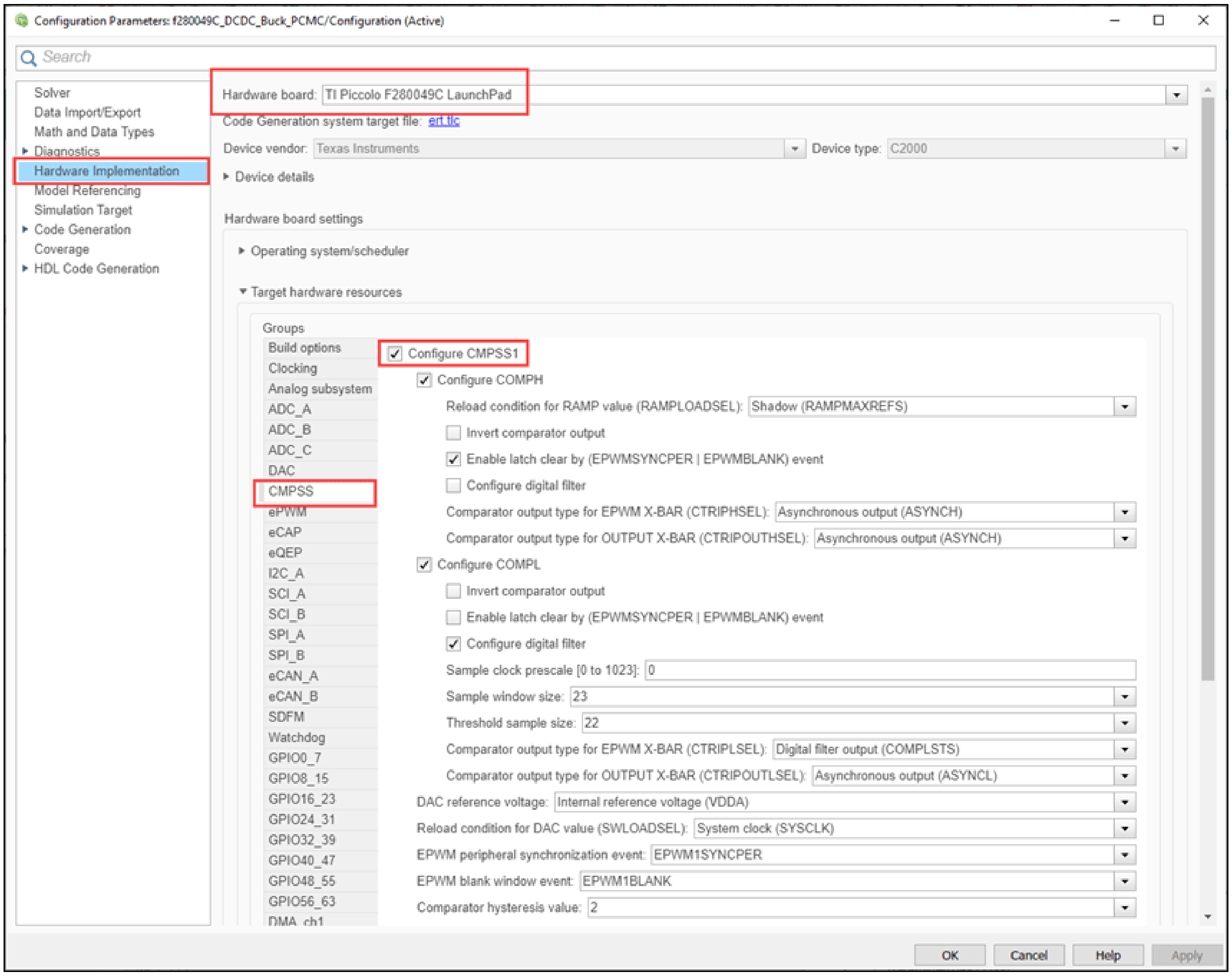
Configure and Run Digital DC/DC Buck Converter Peak Current Mode Control (PCMC) Model

1. Open the `Digital DC/DC Buck Converter Peak Current Mode Control (PCMC)` model. This model is configured for TI Piccolo F280049C LaunchPad hardware.

2. To run the model on other TI C2000 processors, first press **Ctrl+E** to open the Configuration Parameters dialog box. Then, select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

Note: Ensure that the Digital Power Buck Converter BoosterPack is connected to the selected hardware board with correct pin mappings.

3. The following screenshots show the CMPSS configurations performed in the model. You can use the same parameter values if you want to run this example for other hardware boards.


Configuration Parameters: f280049C_DCDC_Buck_PCMC/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation

Hardware board: TI Piccolo F280049C LaunchPad
Code Generation system target file: ert.tlc
Device vendor: Texas Instruments
Device type: C2000
Device details

Hardware board settings

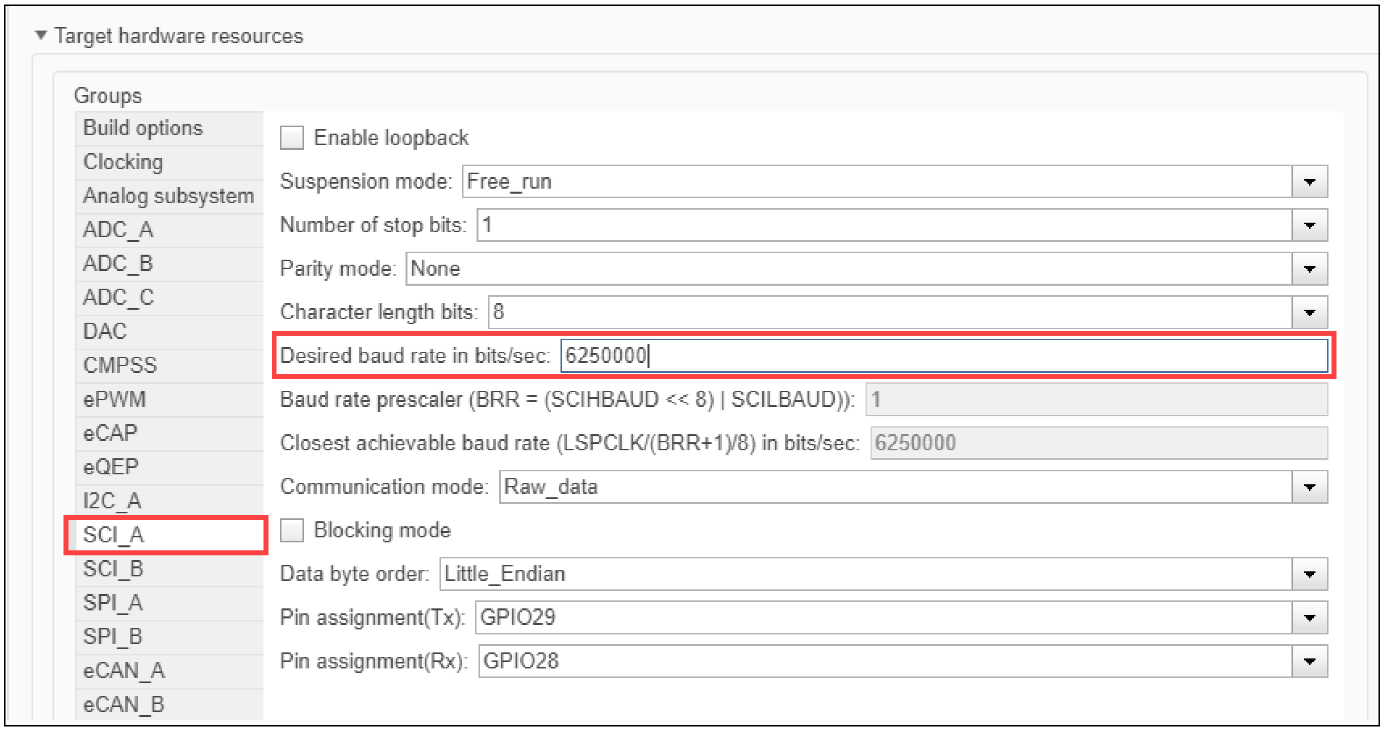
Operating system/scheduler

Target hardware resources

Groups: Build options, Clocking, Analog subsystem, ADC_A, ADC_B, ADC_C, DAC, CMPSS, ePWM, eCAP, eQEP, I2C_A, SCI_A, SCI_B, SPI_A, SPI_B, eCAN_A, eCAN_B, SDFM, Watchdog, GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, GPIO56_63, DMA_ch1

[✓] Configure CMPSS1
[✓] Configure COMPH
Reload condition for RAMP value (RAMPLOADSEL): Shadow (RAMPMAXREFS)
[ ] Invert comparator output
[✓] Enable latch clear by (EPWMSYNCPER | EPWMBLANK) event
[ ] Configure digital filter
Comparator output type for EPWM X-BAR (CTRIPHSEL): Asynchronous output (ASYNCH)
Comparator output type for OUTPUT X-BAR (CTRIPOUTHSEL): Asynchronous output (ASYNCH)
[✓] Configure COMPL
[ ] Invert comparator output
[ ] Enable latch clear by (EPWMSYNCPER | EPWMBLANK) event
[✓] Configure digital filter
Sample clock prescale [0 to 1023]: 0
Sample window size: 23
Threshold sample size: 22
Comparator output type for EPWM X-BAR (CTRIPLSEL): Digital filter output (COMPLSTS)
Comparator output type for OUTPUT X-BAR (CTRIPOUTLSEL): Asynchronous output (ASYNCL)
DAC reference voltage: Internal reference voltage (VDDA)
Reload condition for DAC value (SWLOADSEL): System clock (SYSCLK)
EPWM peripheral synchronization event: EPWM1SYNCPER
EPWM blank window event: EPWM1BLANK
Comparator hysteresis value: 2

4. Ensure that the baud rate is set to 6250000 bits/sec.



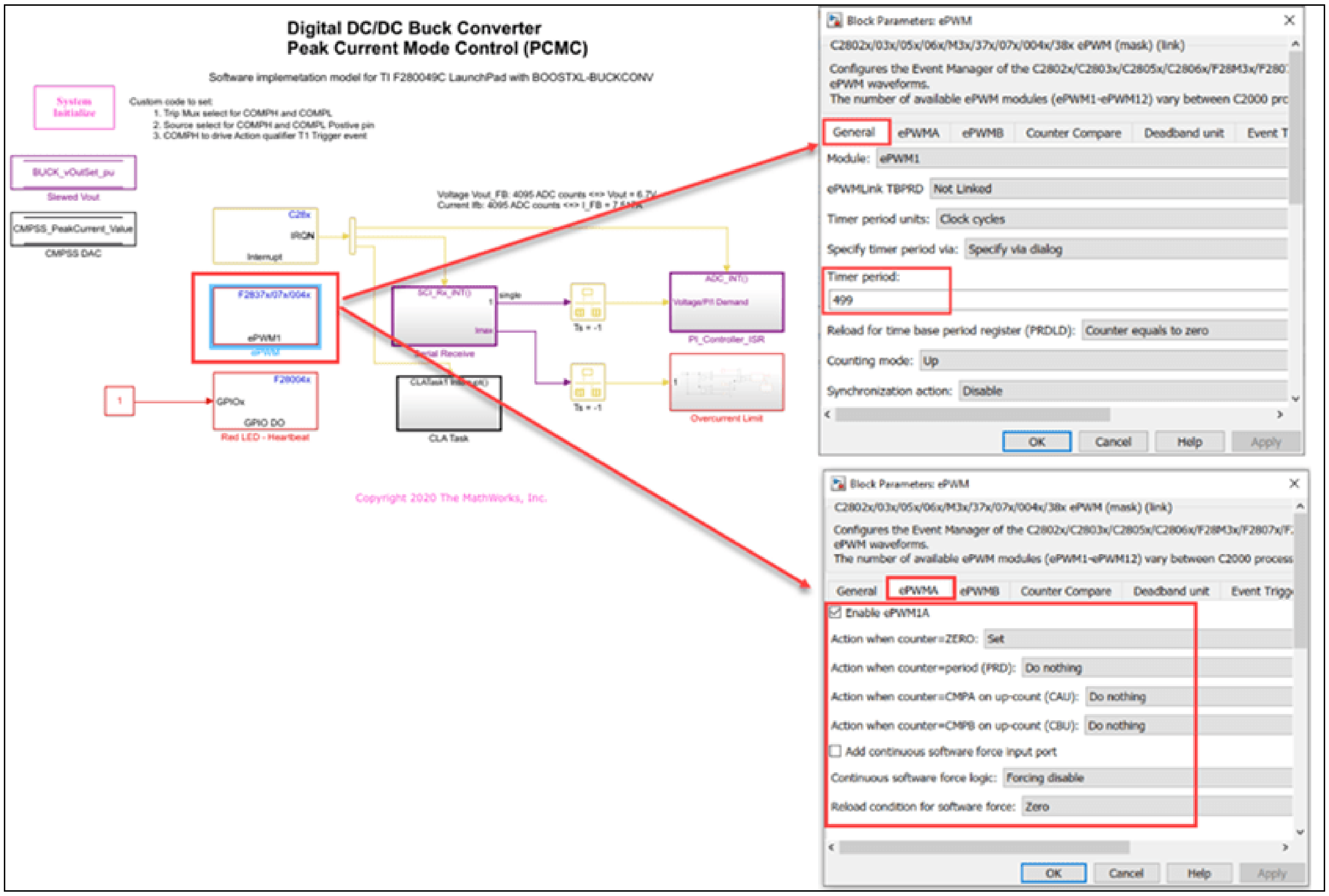
Enable loopback	☐
Suspension mode:	Free_run
Number of stop bits:	1
Parity mode:	None
Character length bits:	8
Desired baud rate in bits/sec:	6250000
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)):	1
Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec:	6250000
Communication mode:	Raw_data
Blocking mode	☐
Data byte order:	Little_Endian
Pin assignment(Tx):	GPIO29
Pin assignment(Rx):	GPIO28

Peripheral Block Configurations

Set the peripheral block configurations for this model. Double-click on the blocks to open block parameter configurations. You can use the same parameter values if you want to run this example for other hardware boards.

ePWM block

Configure the ePWM action qualifier to generate waveforms based on digital compare events. Configure the trip signal generated by the CMPSS module to trip ePWMA to low.


Digital DC/DC Buck Converter
Peak Current Mode Control (PCMC)
Software implementation model for TI F280049C LaunchPad with BOOSTXL-BUCKCONV
System Initialize
Custom code to set:
1. Trip Mux select for COMPH and COMPL
2. Source select for COMPH and COMPL Positive pin
3. COMPH to drive Action qualifier T1 Trigger event
BUCK_vOutSet_pu
Slewed Vout
CMPSS_PeakCurrent_Value
CMPSS DAC
Voltage Vout_FB: 4095 ADC counts <=> Vout = 6.7V
Current Ifb: 4095 ADC counts <=> I_FB = 7.5A
C28x
IRQN
Interrupt
F2837x/07x/004x
ePWM1
ePWM
SCI_Rx_INT()
Serial Receive
Ts = -1
ADC_INT()
Voltage/PI Demand
PI_Controller_ISR
F28004x
GPIOx
GPIO DO
Red LED - Heartbeat
CLATask1 Interrupt()
CLA Task
Overcurrent Limit
Copyright 2020 The MathWorks, Inc.

Block Parameters: ePWM

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the Event Manager of the C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837x/F28004x ePWM waveforms.
The number of available ePWM modules (ePWM1-ePWM12) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

Module: ePWM1
ePWMLink TBPRD: Not Linked
Timer period units: Clock cycles
Specify timer period via: Specify via dialog
Timer period: 499
Reload for time base period register (PRDLD): Counter equals to zero
Counting mode: Up
Synchronization action: Disable

OK | Cancel | Help | Apply

Block Parameters: ePWM

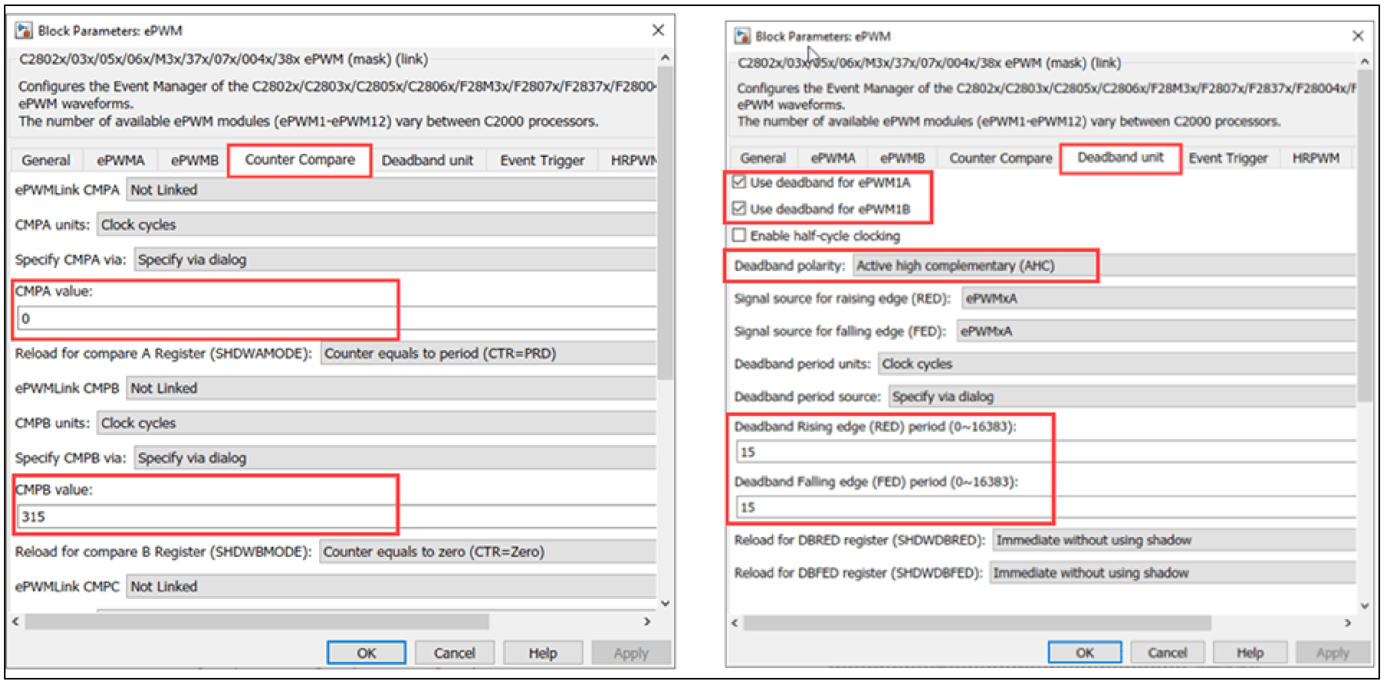
C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the Event Manager of the C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837x/F28004x ePWM waveforms.
The number of available ePWM modules (ePWM1-ePWM12) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

☑ Enable ePWM1A
Action when counter=ZERO: Set
Action when counter=period (PRD): Do nothing
Action when counter=CMPA on up-count (CAU): Do nothing
Action when counter=CMPB on up-count (CBU): Do nothing
☐ Add continuous software force input port
Continuous software force logic: Forcing disable
Reload condition for software force: Zero

OK | Cancel | Help | Apply


Block Parameters: ePWM

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the Event Manager of the C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837x/F28004x ePWM waveforms.
The number of available ePWM modules (ePWM1-ePWM12) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger | HRPWM

ePWMLink CMPA: Not Linked
CMPA units: Clock cycles
Specify CMPA via: Specify via dialog
CMPA value: 0
Reload for compare A Register (SHDWAMODE): Counter equals to period (CTR=PRD)
ePWMLink CMPB: Not Linked
CMPB units: Clock cycles
Specify CMPB via: Specify via dialog
CMPB value: 315
Reload for compare B Register (SHDWBMODE): Counter equals to zero (CTR=Zero)
ePWMLink CMPC: Not Linked

OK | Cancel | Help | Apply

Block Parameters: ePWM

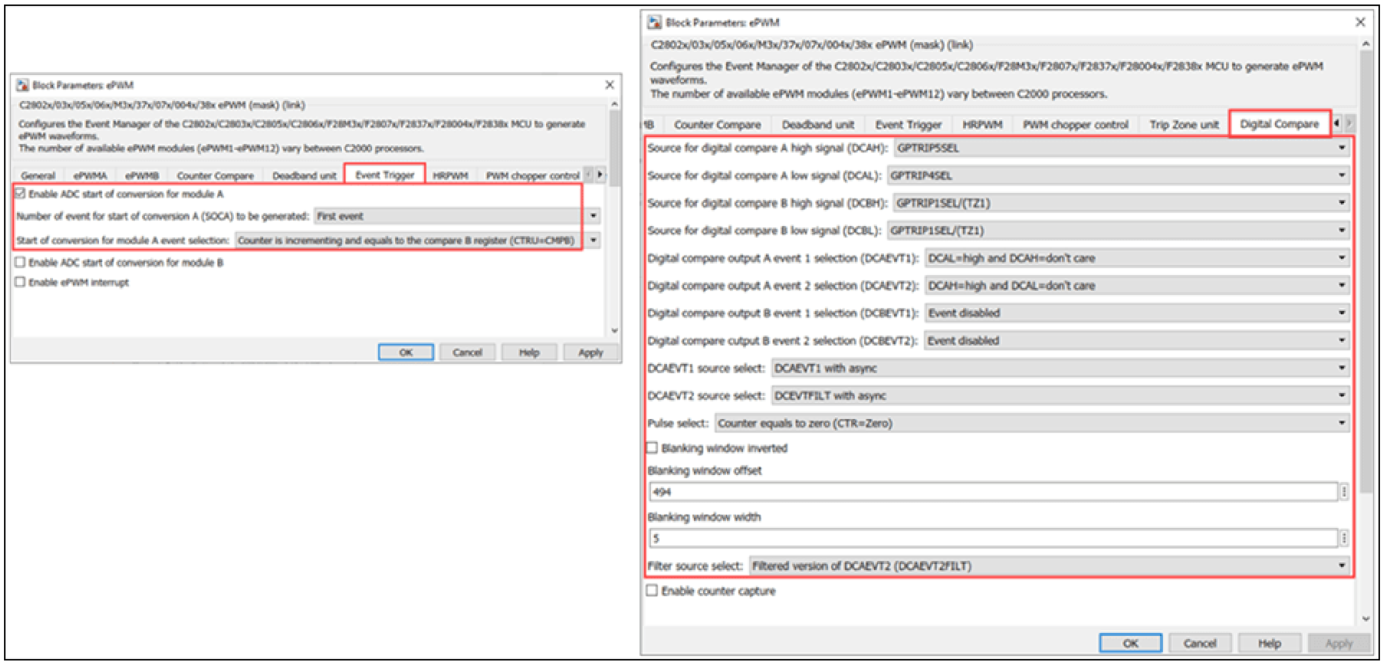
C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the Event Manager of the C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837x/F28004x ePWM waveforms.
The number of available ePWM modules (ePWM1-ePWM12) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger | HRPWM

☑ Use deadband for ePWM1A
☑ Use deadband for ePWM1B
☐ Enable half-cycle clocking
Deadband polarity: Active high complementary (AHC)
Signal source for raising edge (RED): ePWMxA
Signal source for falling edge (FED): ePWMxA
Deadband period units: Clock cycles
Deadband period source: Specify via dialog
Deadband Rising edge (RED) period (0~16383): 15
Deadband Falling edge (FED) period (0~16383): 15
Reload for DBRED register (SHDWDBRED): Immediate without using shadow
Reload for DBFED register (SHDWDBFED): Immediate without using shadow

OK | Cancel | Help | Apply



C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the Event Manager of the C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837x/F28004x/F2838x MCU to generate ePWM waveforms.
The number of available ePWM modules (ePWM1-ePWM12) vary between C2000 processors.

☑ Enable ADC start of conversion for module A

Number of event for start of conversion A (SOCA) to be generated:	First event
Start of conversion for module A event selection:	Counter is incrementing and equals to the compare B register (CTRU=CMPB)

☐ Enable ADC start of conversion for module B

☐ Enable ePWM interrupt

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

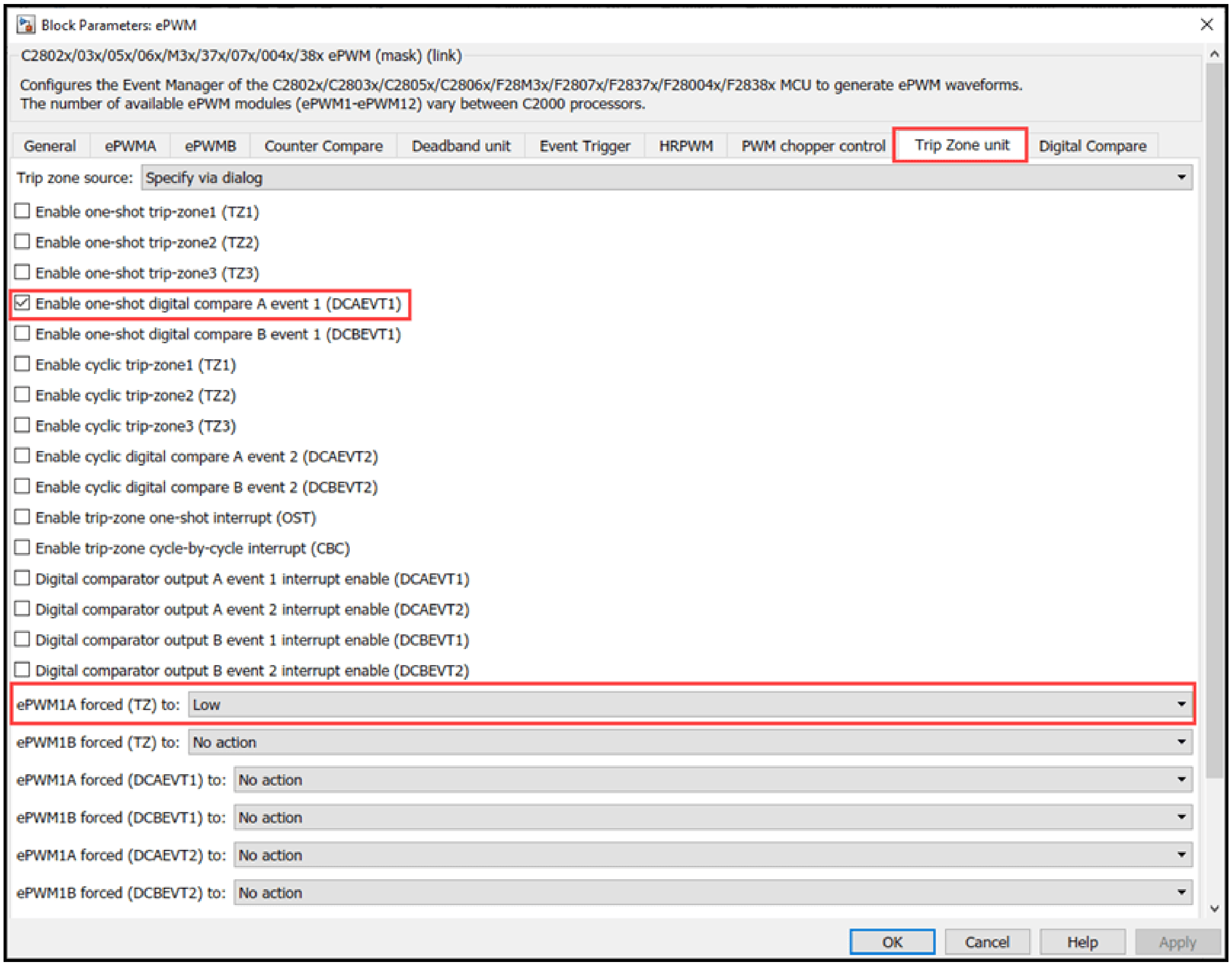
Configures the Event Manager of the C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837x/F28004x/F2838x MCU to generate ePWM waveforms.
The number of available ePWM modules (ePWM1-ePWM12) vary between C2000 processors.

Source for digital compare A high signal (DCAH):	GPTRIP5SEL
Source for digital compare A low signal (DCAL):	GPTRIP4SEL
Source for digital compare B high signal (DCBH):	GPTRIP1SEL/(TZ1)
Source for digital compare B low signal (DCBL):	GPTRIP1SEL/(TZ1)
Digital compare output A event 1 selection (DCAEVT1):	DCAL=high and DCAH=don't care
Digital compare output A event 2 selection (DCAEVT2):	DCAH=high and DCAL=don't care
Digital compare output B event 1 selection (DCBEVT1):	Event disabled
Digital compare output B event 2 selection (DCBEVT2):	Event disabled
DCAEVT1 source select:	DCAEVT1 with async
DCAEVT2 source select:	DCEVTFILT with async
Pulse select:	Counter equals to zero (CTR=Zero)

☐ Blanking window inverted

Blanking window offset	494
Blanking window width	5
Filter source select:	Filtered version of DCAEVT2 (DCAEVT2FILT)

☐ Enable counter capture



C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the Event Manager of the C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837x/F28004x/F2838x MCU to generate ePWM waveforms.
The number of available ePWM modules (ePWM1-ePWM12) vary between C2000 processors.

Trip zone source: Specify via dialog

- ☐ Enable one-shot trip-zone1 (TZ1)
- ☐ Enable one-shot trip-zone2 (TZ2)
- ☐ Enable one-shot trip-zone3 (TZ3)
- ☑ Enable one-shot digital compare A event 1 (DCAEVT1)
- ☐ Enable one-shot digital compare B event 1 (DCBEVT1)
- ☐ Enable cyclic trip-zone1 (TZ1)
- ☐ Enable cyclic trip-zone2 (TZ2)
- ☐ Enable cyclic trip-zone3 (TZ3)
- ☐ Enable cyclic digital compare A event 2 (DCAEVT2)
- ☐ Enable cyclic digital compare B event 2 (DCBEVT2)
- ☐ Enable trip-zone one-shot interrupt (OST)
- ☐ Enable trip-zone cycle-by-cycle interrupt (CBC)
- ☐ Digital comparator output A event 1 interrupt enable (DCAEVT1)
- ☐ Digital comparator output A event 2 interrupt enable (DCAEVT2)
- ☐ Digital comparator output B event 1 interrupt enable (DCBEVT1)
- ☐ Digital comparator output B event 2 interrupt enable (DCBEVT2)

ePWM1A forced (TZ) to: Low

ePWM1B forced (TZ) to: No action

ePWM1A forced (DCAEVT1) to: No action

ePWM1B forced (DCBEVT1) to: No action

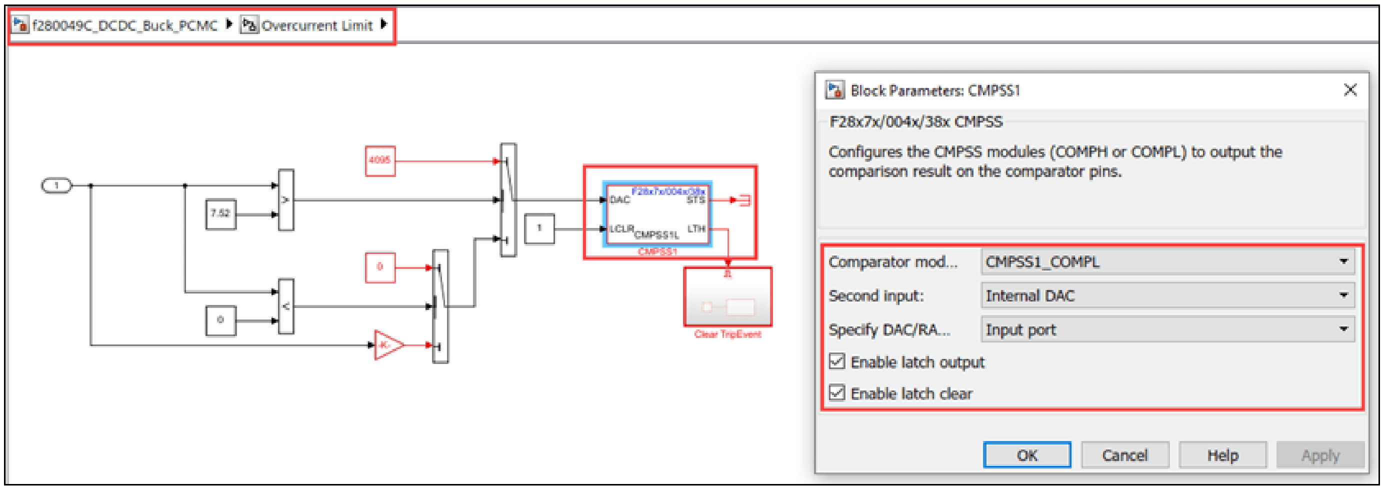
ePWM1A forced (DCAEVT2) to: No action

ePWM1B forced (DCBEVT2) to: No action

Parameter	Value	Description
Time Period – EPWM(TBPRD)	499	The value achieves an ePWM switching frequency of 200KHz, the PCMC is executed at this rate.
CMPA Value – EPWM(CMPA)	0	This enables the ePWM output to be driven to 100% DC which is then tripped OFF in PCMC.
CMPB Value – EPWM(CMPB)	315	Marks the start of the ADC SOC (Start Of Conversion) event well before next PWM cycle to update the peak current value.
Dead-band Value	15	The switching gap between EPWM1A and EPWM1B. The ePWM Dead-Band Generator submodule is configured to output a PWM-LO signal (controlled by EPWM_B) that is complementary to PWM-HI, with inserted dead-band.

CMPSS Block in Overcurrent Limit

The overcurrent logic trips the output on an overcurrent condition. Although overcurrent protection is inherent in the peak current loop, the application implements a redundant protection path using spare comparator resources on the MCU. This trip is configured for one shot trip in the ePWM. Clear the trip flags to re-enable the outputs.



Block Parameters: CMPSS1

F28x7x/004x/38x CMPSS

Configures the CMPSS modules (COMPH or COMPL) to output the comparison result on the comparator pins.

Comparator mod...	CMPSS1_COMPL
Second input:	Internal DAC
Specify DAC/RA...	Input port

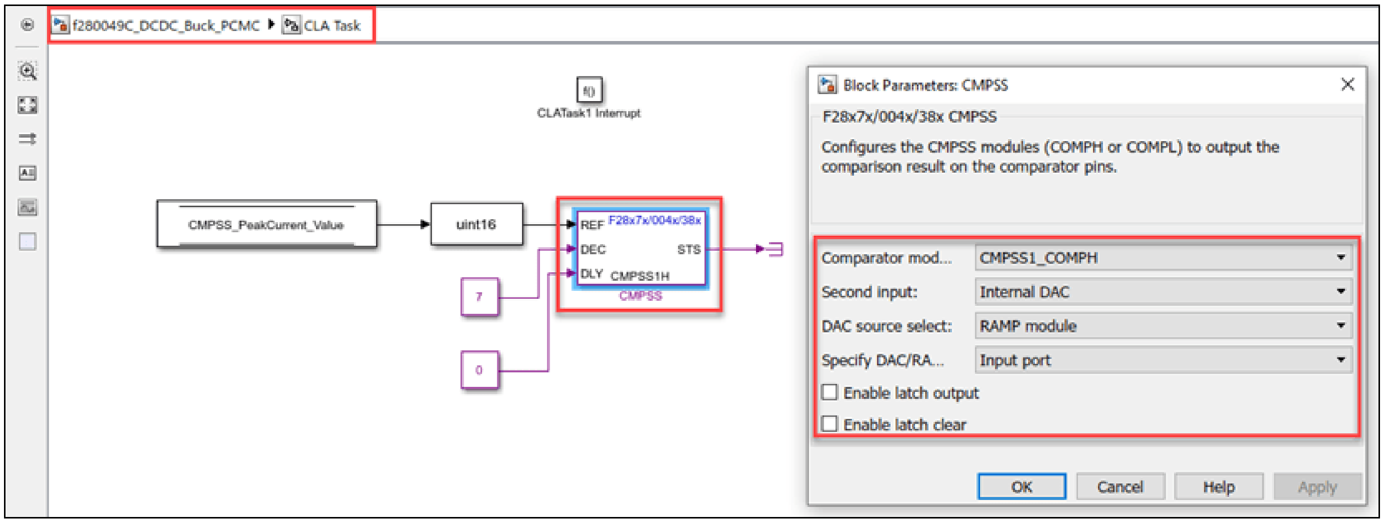
☑ Enable latch output

☑ Enable latch clear

CMPSS Block in CLA Task

The PCMC control algorithm is executed by CLA module, which generates an interrupt to the CPU when the algorithm is executed. The CMPSS with computed peak current threshold is updated within the CLA interrupt routine. For information on the CLA configuration, refer to “Using the Control Law Accelerator (CLA)” on page 4-270.

- The comparator monitors the inductor feedback current against the peak current threshold. It generates a trip signal for the ePWM when the current limit is met.
- The peak current threshold for the comparator is provided by a reference DAC. Use the ramp generator to control the reference DAC values, which are synchronized to a ePWM
- The RAMP decrement value, which defines the slope of the RAMP signal, is fixed to 7. This fixed slope might not cover the full dynamic range of the operation of the plant using PCMC.



Block Parameters: CMPSS

F28x7x/004x/38x CMPSS

Configures the CMPSS modules (COMPH or COMPL) to output the comparison result on the comparator pins.

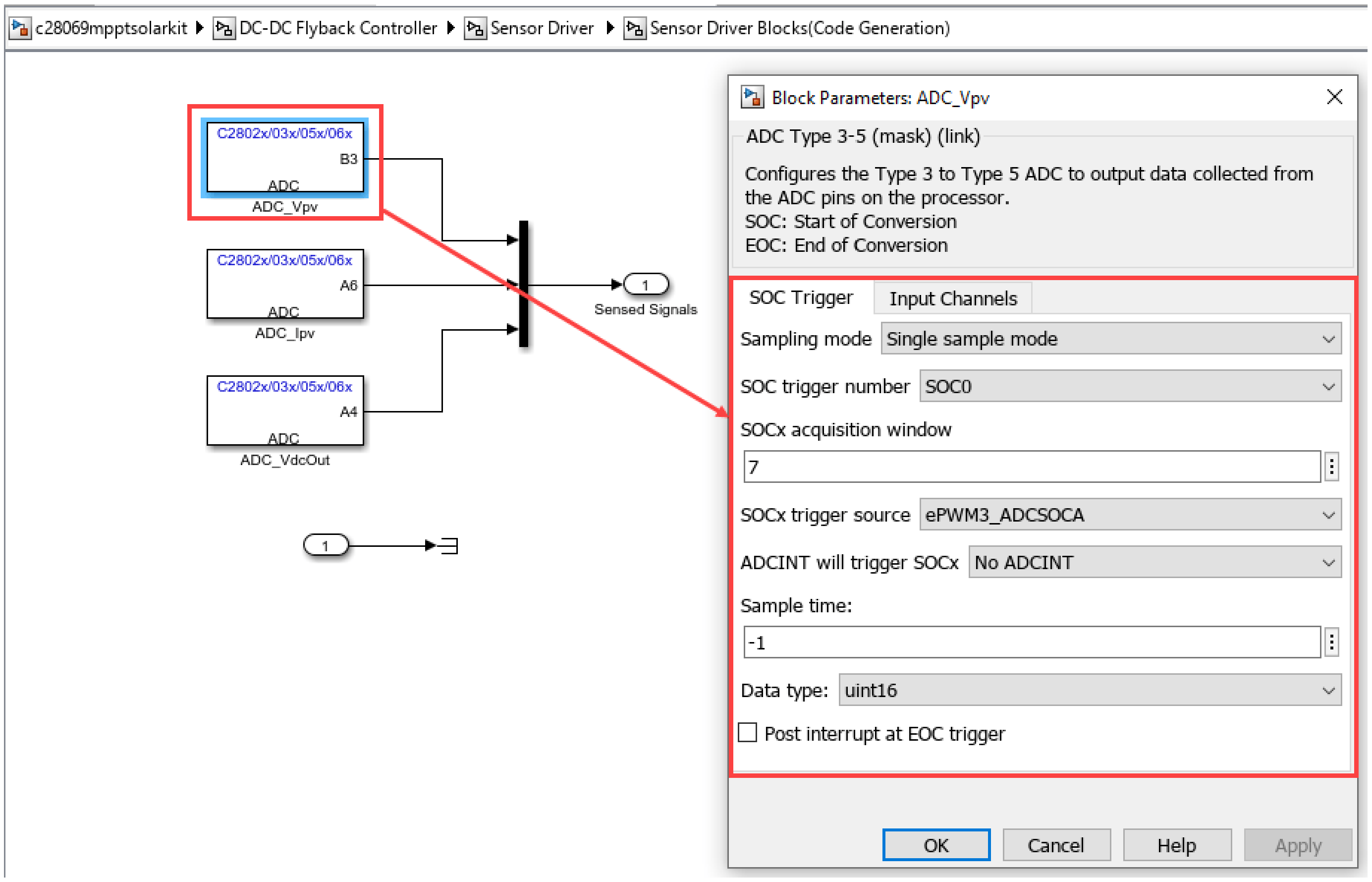
Comparator mod...	CMPSS1_COMPH
Second input:	Internal DAC
DAC source select:	RAMP module
Specify DAC/RA...	Input port

☐ Enable latch output

☐ Enable latch clear

ADC Block in PI Controller ISR

The ePWM triggers the ADC start of conversion (SOC) event to sample the output voltage. The ADC end of conversation (EOC) interrupt executes the digital control algorithm that computes the actuation value depending on the error in the system.



ADC Type 3-5 (mask) (link)

Configures the Type 3 to Type 5 ADC to output data collected from the ADC pins on the processor.
SOC: Start of Conversion
EOC: End of Conversion

Sampling mode: Single sample mode
SOC trigger number: SOC0
SOCx acquisition window: 7
SOCx trigger source: ePWM3_ADCSOCA
ADCINT will trigger SOCx: No ADCINT
Sample time: -1
Data type: uint16
☐ Post interrupt at EOC trigger

Simulate Peak current mode Control for DC/DC Buck Converter Plant Model

Run the Model

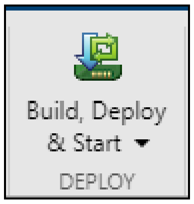
1. Open the f280049C_DCDC_Buck_PCMC.slx model.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results. You can tune the input parameters using the dashboard controls.

Generate Code for Controller and Load on Hardware Board

1. To generate the code for the model f280049C_DCDC_Buck_PCMC , press **Ctrl+B** or click **Build, Deploy & Start**.

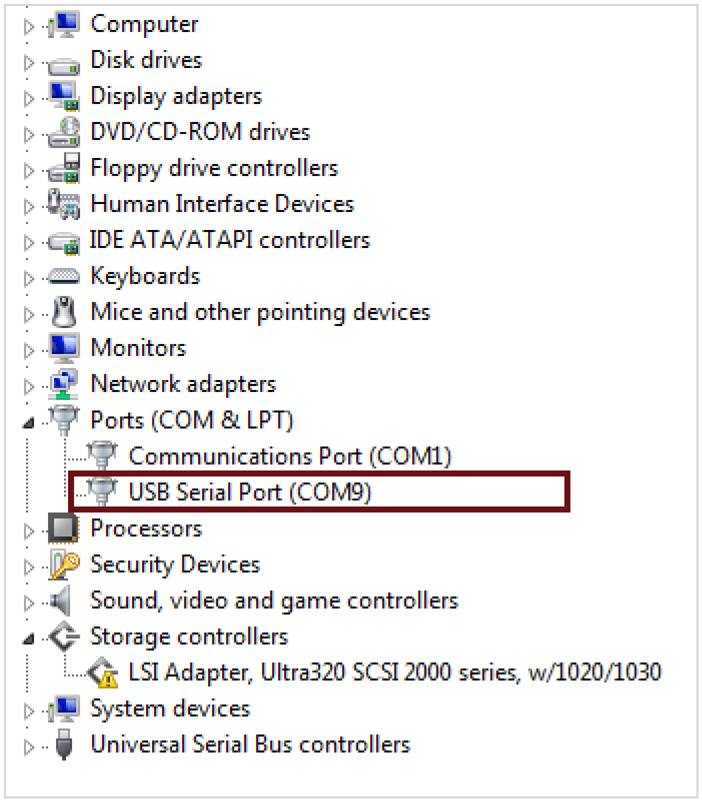


2. Follow the build process by opening the diagnostic viewer using the link provided at the bottom of the model canvas. After you load the code on the board, a red LED blinks on the hardware board, indicating that the code is running.

Monitor Signals and Tune Parameters on Host Computer

Configure and Run Model on Host Computer

1. On the host computer, browse to **Device Manager** > **Ports (COM & LPT)** to find the COM port.



2. Set the COM ports of the following blocks in the c2000_host_read_PCMC model to match the COM port of the host computer:

- **c2000_host_read_PCMC** > **Serial Receive**
- **c2000_host_read_PCMC** > **Serial Send** > **Serial Send1**

3. Click on the **Run** button to run the model.

Tune Parameters

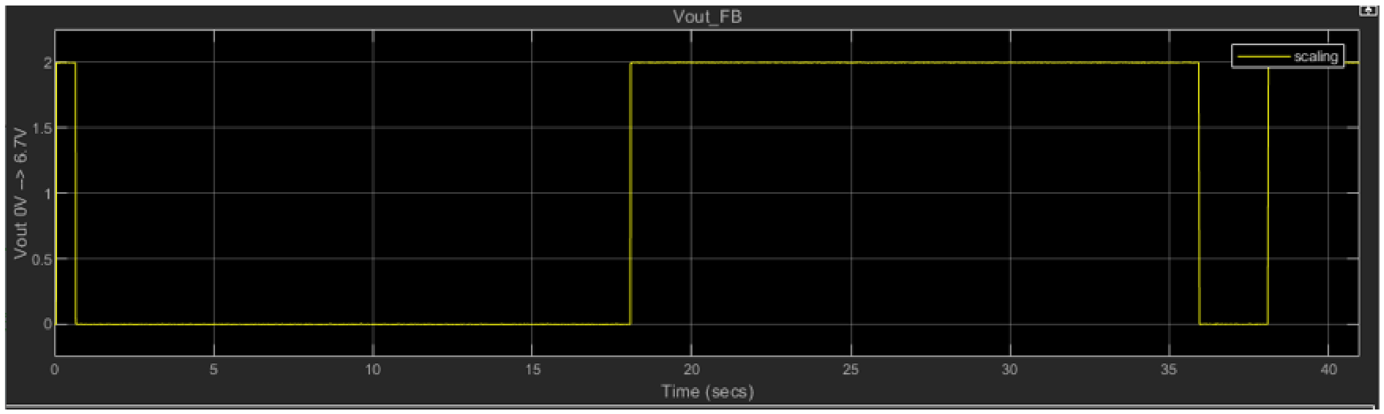
While the model runs, you can tune parameters using the following dashboard blocks:

- **Voltage Request** - Change the output voltage demand. This parameter is the main request for the control loop. The controller algorithm compares the Voltage Request value with the measured output voltage and adjusts the PWM duty cycle towards achieving the output voltage.
- **P Gain** - Change the proportional gain of the controller algorithm. You can change this parameter to study the robustness of the controller. Large, abrupt changes might lead to instability of the controller; apply changes smoothly.
- **I Gain** - Change the integral gain of the controller algorithm. You can change this parameter to study the robustness of the controller. Large, abrupt changes might lead to instability of the controller; apply changes smoothly.
- **Over Current Limit** - Change the overcurrent setpoint to be less than the current operating in the converter. For example, when the controller is regulating the Vout at 2 V, try to program the overcurrent limit to 1.2 A. This should trip the outputs to the plant as the over current condition occurred.
- **Active load** - Turn the active load present on the hardware on or off. This parameter allows you to add an extra load resistor to study the effect of abrupt changes in the load circuit.

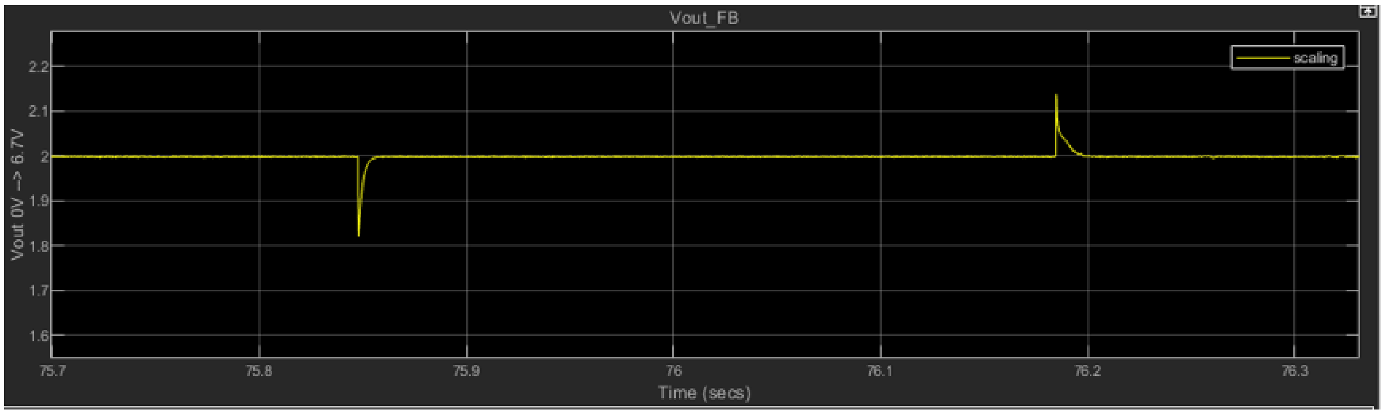
Monitor Signals

While the model runs, you can monitor the following signals on the **Scope** block:

- **V_FB Voltage** - The measured output voltage of the system is 2 V.



- **Active load**:



Troubleshooting

- Ensure the 9 V DC power supply input voltage is stable when active load is enabled. In the test bench setup, the active load works when input voltage is set to 10 V.
- If you face connectivity issues with the serial connection between host and target, try to disconnect and reconnect the TI Piccolo F280049C LaunchPad.
- When the current in the converter is more than the overcurrent setpoint, the output gets tripped and the P-I controller reaches saturation. To re-enable the output the below sequence can be followed:
- Set the overcurrent limit to a higher value such as 5.2 A.
- Set the voltage setpoint to 0 volts and then back to 2 V.

Other Things to Try

- Calculate the slope compensation for a different voltage setpoint, configure the comparator and run the example.

More About

- “CMPSS”
- c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM
- “Using the Control Law Accelerator (CLA)” on page 4-270

Using Comparator Subsystem (CMPSS) for Voltage Compare

This example shows how to use the Comparator Subsystem (CMPSS) to compare the analog voltage signals for Texas Instruments™ C2000™ processors and monitor the status output. Using this example, you can:

- Configure the CMPSS module to generate the digital output when there is a voltage difference on the input pins
- Configure Digital Filter to generate filtered output for CMPSS

Introduction

The CMPSS consists of two modules, Comparator High (COMPH) and Comparator Low (COMPL). Each module generates a high digital output when the voltage on the first input pin (positive input) is greater than the voltage on the second input pin (negative input). Vice versa it generates low digital output when the voltage on the first input pin (positive input) is less than the voltage on the second input pin (negative input).

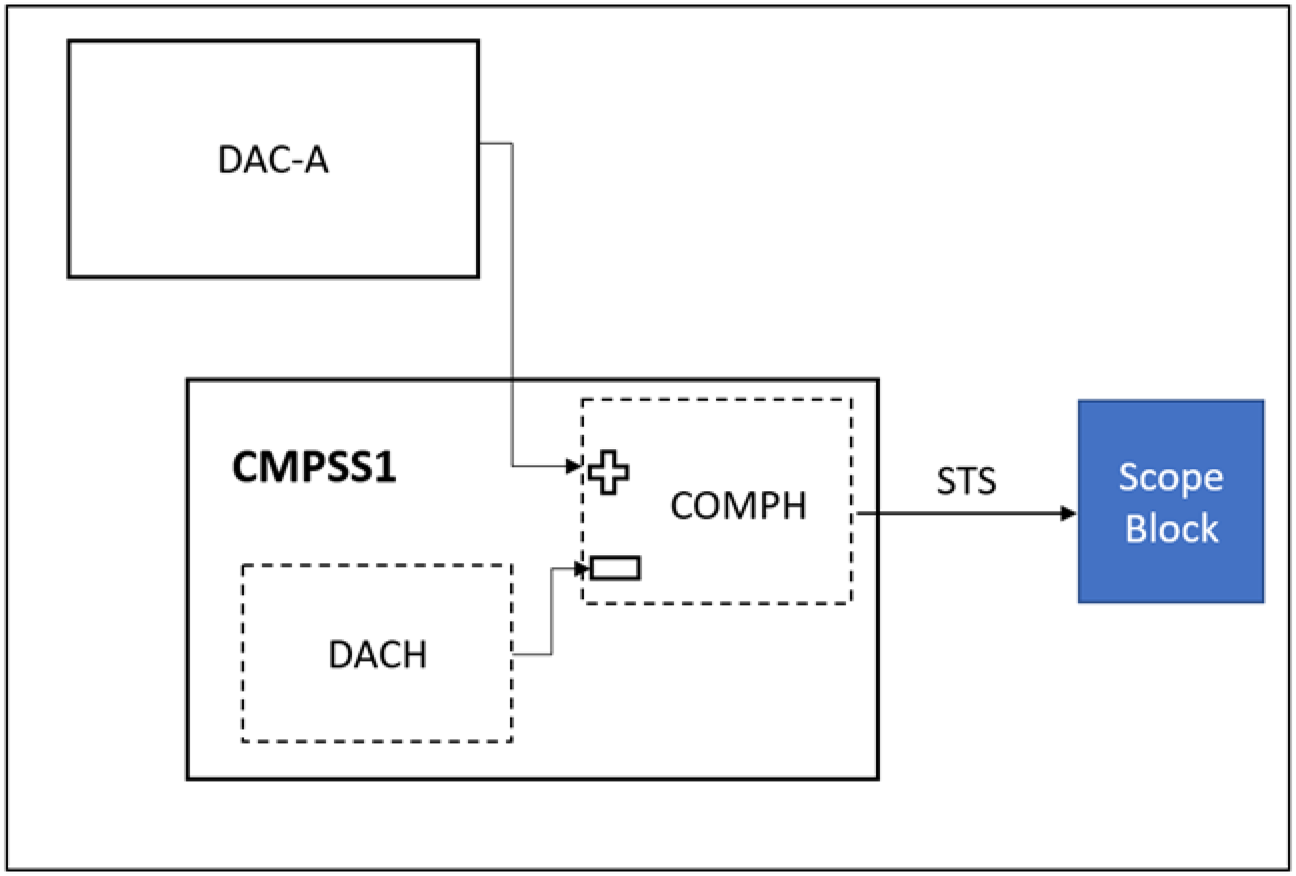
The second input pin can either be the external pin or the internal CMPSS DAC module.

Required Hardware

To run this example, you can use either TI Piccolo F280049C LaunchPad or TI Delfino F28379D LaunchPad. For your convenience, a preconfigured model for `TI Delfino F28379D LaunchPad` is used in this example.

Hardware Connections

CMPSS1

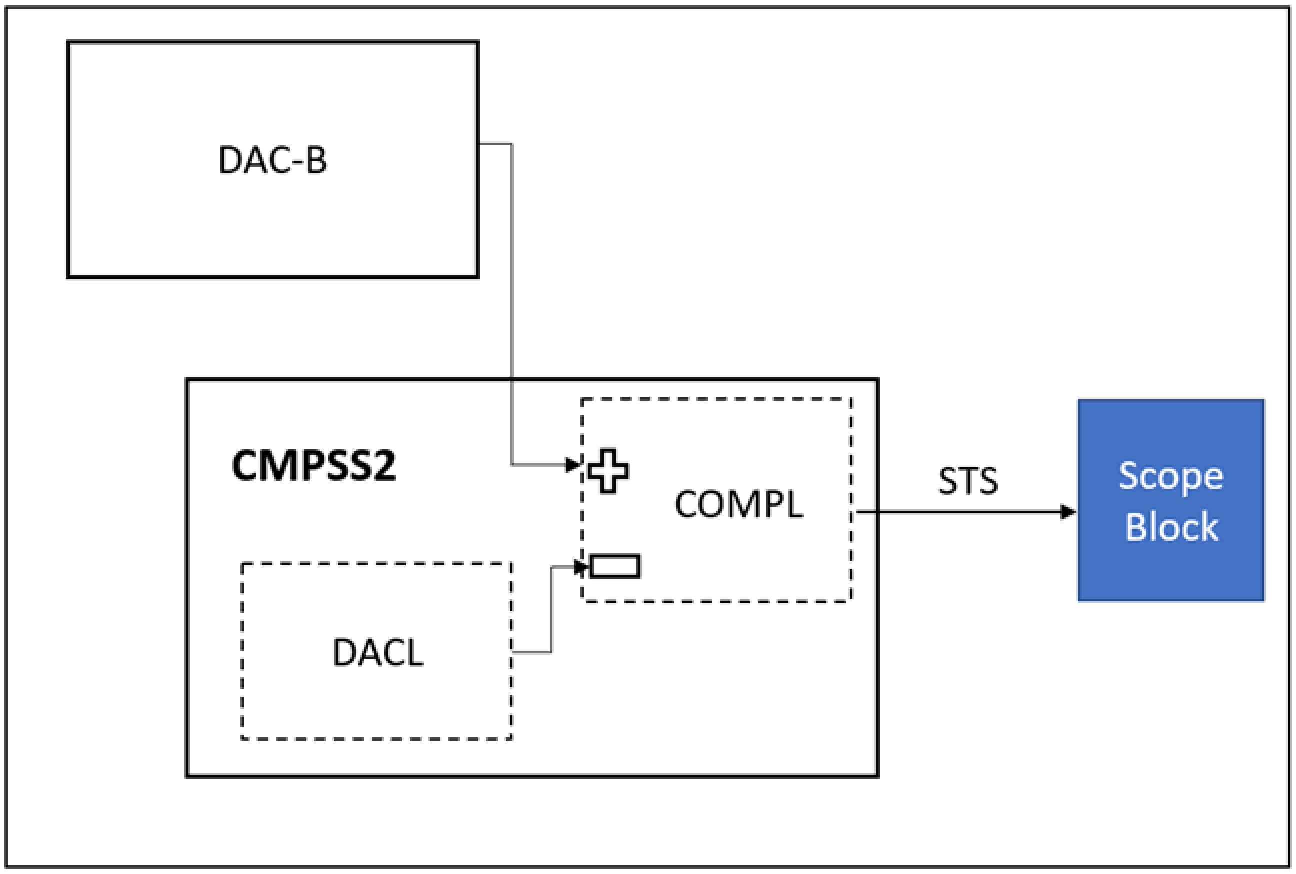


1. DAC- A is connected to the positive input of the CMPSS1H comparator (COMPH of CMPSS1).

2. Configure the negative input to the internal DAC of CMPSS1(DACH).

3. COMPH actively compares the input voltages on its input pins and provides the STS output.

CMPSS2



1. DAC- B is connected to the positive input of the CMPSS2L comparator (COMPL of CMPSS2).

2. Configure the negative input to the internal DAC of CMPSS2(DACL).

3. This COMPL actively compares the input voltages on its input pins and provides the STS output.

Connect the launchpad header pins for TI Delfino F28379D LaunchPad as listed below.

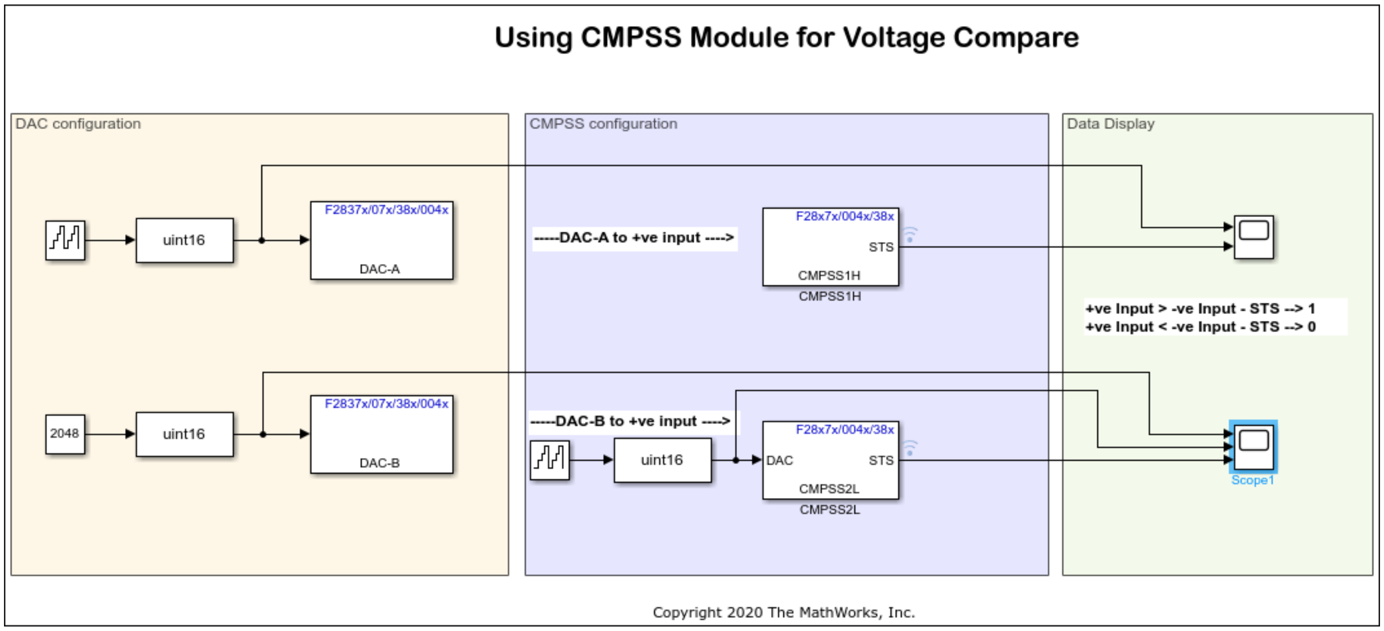
CMPSS1	DAC-A	COMPH
	J3[Pin 30]	J3[Pin 29]
CMPSS2	DAC-B	COMPL
	J7[Pin 70]	J7[Pin 69]

Model

Using CMPSS Module for Voltage Compare Model

To open the model, type the following command in the MATLAB® prompt:

open_system('f2837x_cmpss');


Using CMPSS Module for Voltage Compare
DAC configuration
CMPSS configuration
Data Display
uint16
F2837x/07x/38x/004x
DAC-A
2048
uint16
F2837x/07x/38x/004x
DAC-B
-----DAC-A to +ve input ---->
F28x7x/004x/38x
STS
CMPSS1H
CMPSS1H
-----DAC-B to +ve input ---->
uint16
F28x7x/004x/38x
DAC
STS
CMPSS2L
CMPSS2L
+ve Input > -ve Input - STS --> 1
+ve Input < -ve Input - STS --> 0
Scope1
Copyright 2020 The MathWorks, Inc.

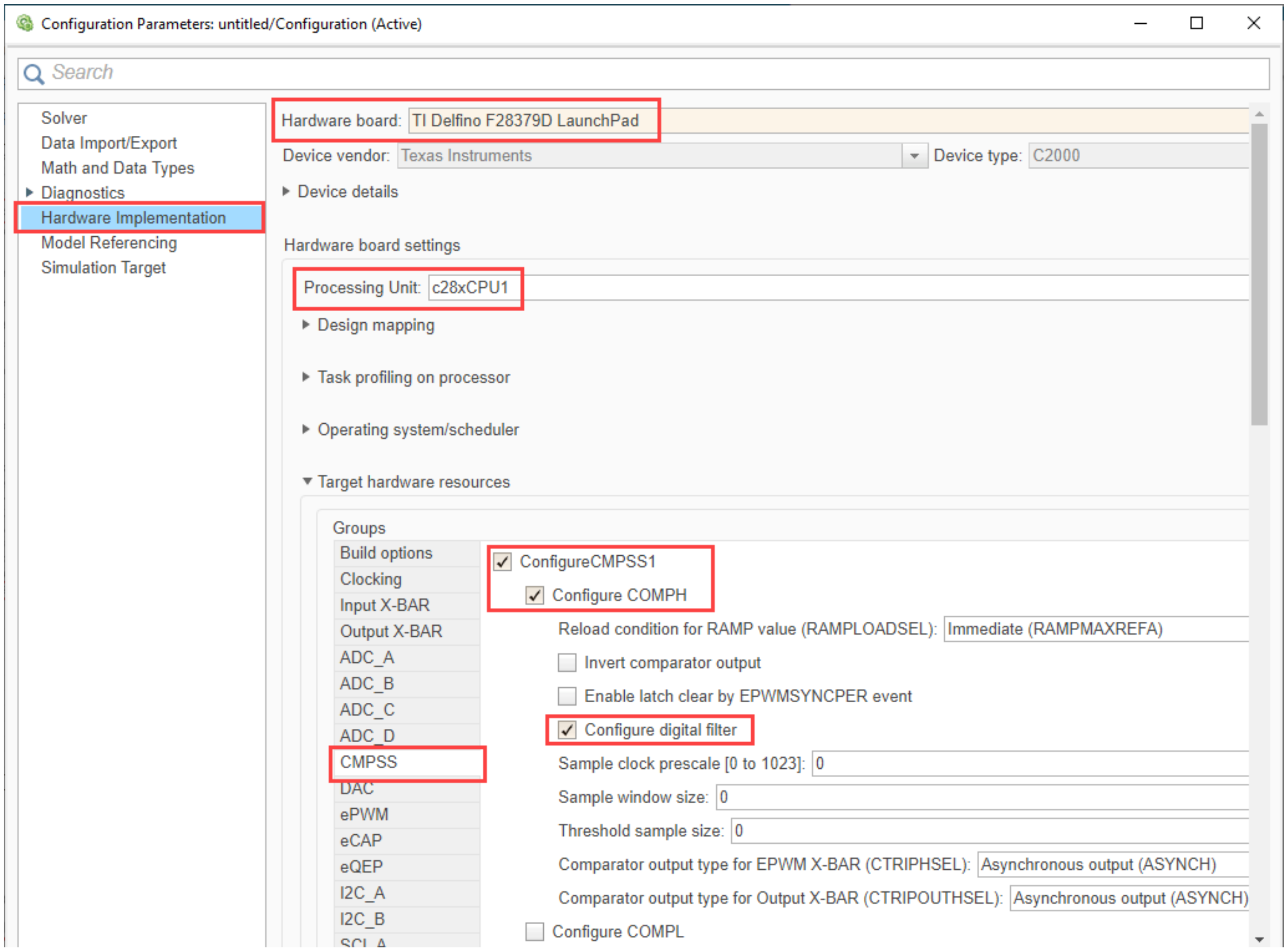
Configure the Model

1. Open the `Using CMPSS Module for Voltage Compare Model` model. This model is configured for TI Delfino F28379D LaunchPad hardware.

2. To run the model on other TI C2000 processors, press **Ctrl+E** to open the Configuration Parameters dialog box and select the required hardware board by navigating to **Hardware Implementation > Hardware board**.

3. The following are the comparator configurations performed in the model. Ensure that the specified parameter values are the same if you want to run this example for other hardware board.

- Configure CMPSS1 with COMPH
- Configure CMPSS2 with COMPL



Configuration Parameters: untitled/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target

Hardware board: TI Delfino F28379D LaunchPad

Device vendor: Texas Instruments | Device type: C2000

Device details

Hardware board settings

Processing Unit: c28xCPU1

Design mapping

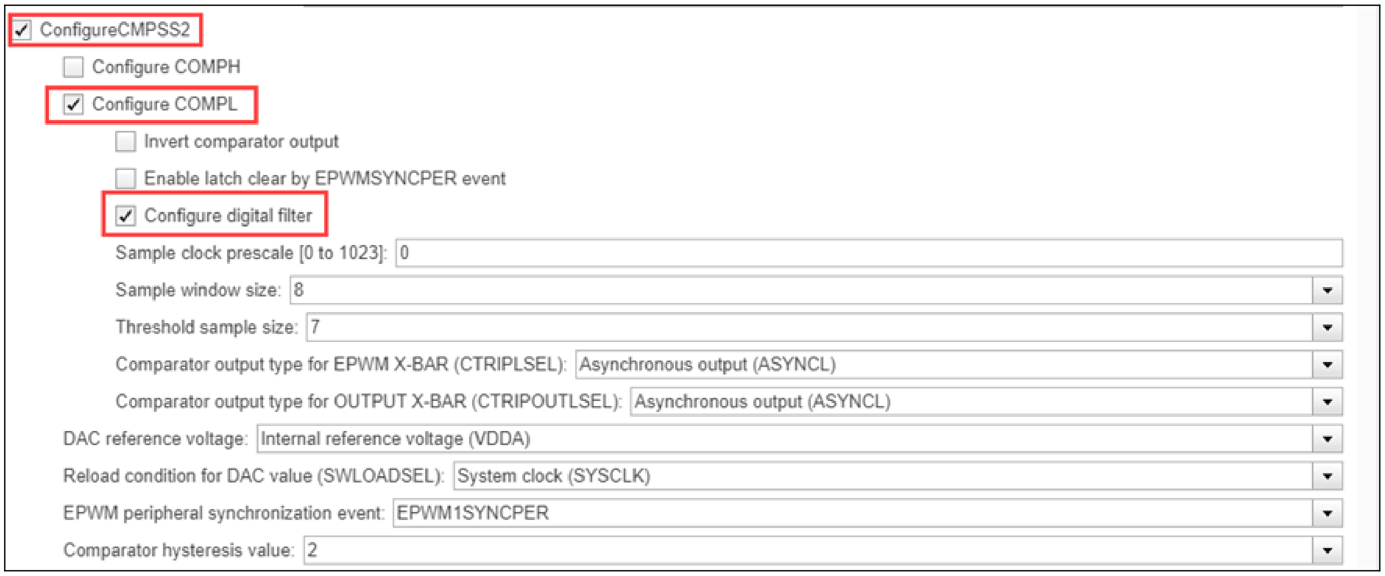
Task profiling on processor

Operating system/scheduler

Target hardware resources

Groups: Build options, Clocking, Input X-BAR, Output X-BAR, ADC_A, ADC_B, ADC_C, ADC_D, CMPSS, DAC, ePWM, eCAP, eQEP, I2C_A, I2C_B, SCI_A

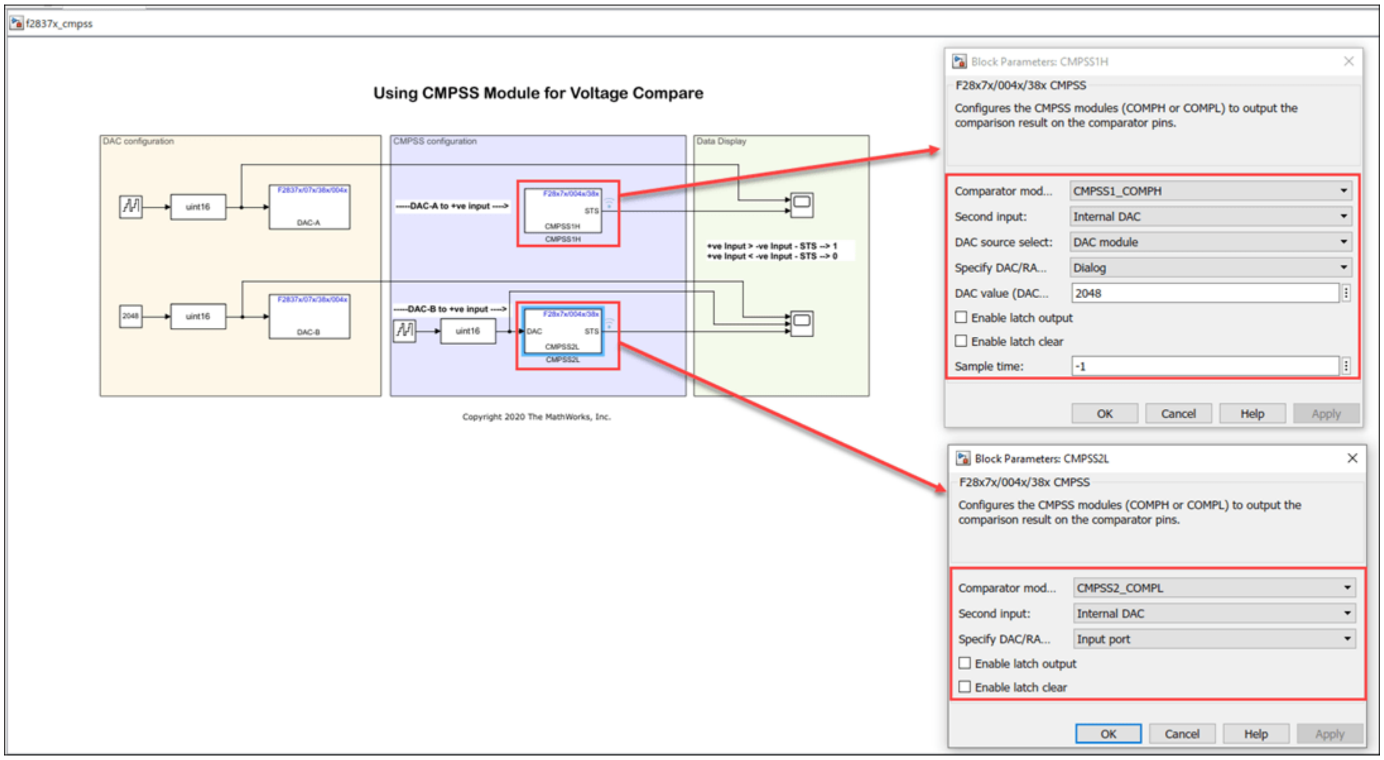
- [✓] ConfigureCMPSS1
  - [✓] Configure COMPH
    - Reload condition for RAMP value (RAMPLOADSEL): Immediate (RAMPMAXREFA)
    - [ ] Invert comparator output
    - [ ] Enable latch clear by EPWMSYNCPER event
    - [✓] Configure digital filter
    - Sample clock prescale [0 to 1023]: 0
    - Sample window size: 0
    - Threshold sample size: 0
    - Comparator output type for EPWM X-BAR (CTRIPHSEL): Asynchronous output (ASYNCH)
    - Comparator output type for Output X-BAR (CTRIPOUTHSEL): Asynchronous output (ASYNCH)
  - [ ] Configure COMPL



- [✓] ConfigureCMPSS2
  - [ ] Configure COMPH
  - [✓] Configure COMPL
    - [ ] Invert comparator output
    - [ ] Enable latch clear by EPWMSYNCPER event
    - [✓] Configure digital filter
    - Sample clock prescale [0 to 1023]: 0
    - Sample window size: 8
    - Threshold sample size: 7
    - Comparator output type for EPWM X-BAR (CTRIPLSEL): Asynchronous output (ASYNCL)
    - Comparator output type for OUTPUT X-BAR (CTRIPOUTLSEL): Asynchronous output (ASYNCL)
  - DAC reference voltage: Internal reference voltage (VDDA)
  - Reload condition for DAC value (SWLOADSEL): System clock (SYSCLK)
  - EPWM peripheral synchronization event: EPWM1SYNCPER
  - Comparator hysteresis value: 2

CMPSS Block Configurations

Following are the CMPSS block configurations done for this model. Double-click on the blocks to open block parameter configurations. Ensure the specified parameter values are the same if you want to run this example for other hardware board



Block Parameters: CMPSS1H

F28x7x/004x/38x CMPSS

Configures the CMPSS modules (COMPH or COMPL) to output the comparison result on the comparator pins.

Comparator mod...	CMPSS1_COMPH
Second input:	Internal DAC
DAC source select:	DAC module
Specify DAC/RA...	Dialog
DAC value (DAC...	2048
☐ Enable latch output	
☐ Enable latch clear	
Sample time:	-1

Block Parameters: CMPSS2L

F28x7x/004x/38x CMPSS

Configures the CMPSS modules (COMPH or COMPL) to output the comparison result on the comparator pins.

Comparator mod...	CMPSS2_COMPL
Second input:	Internal DAC
Specify DAC/RA...	Input port
☐ Enable latch output	
☐ Enable latch clear	

Run the Model

1. Open **Hardware** tab and click **Monitor & Tune**.

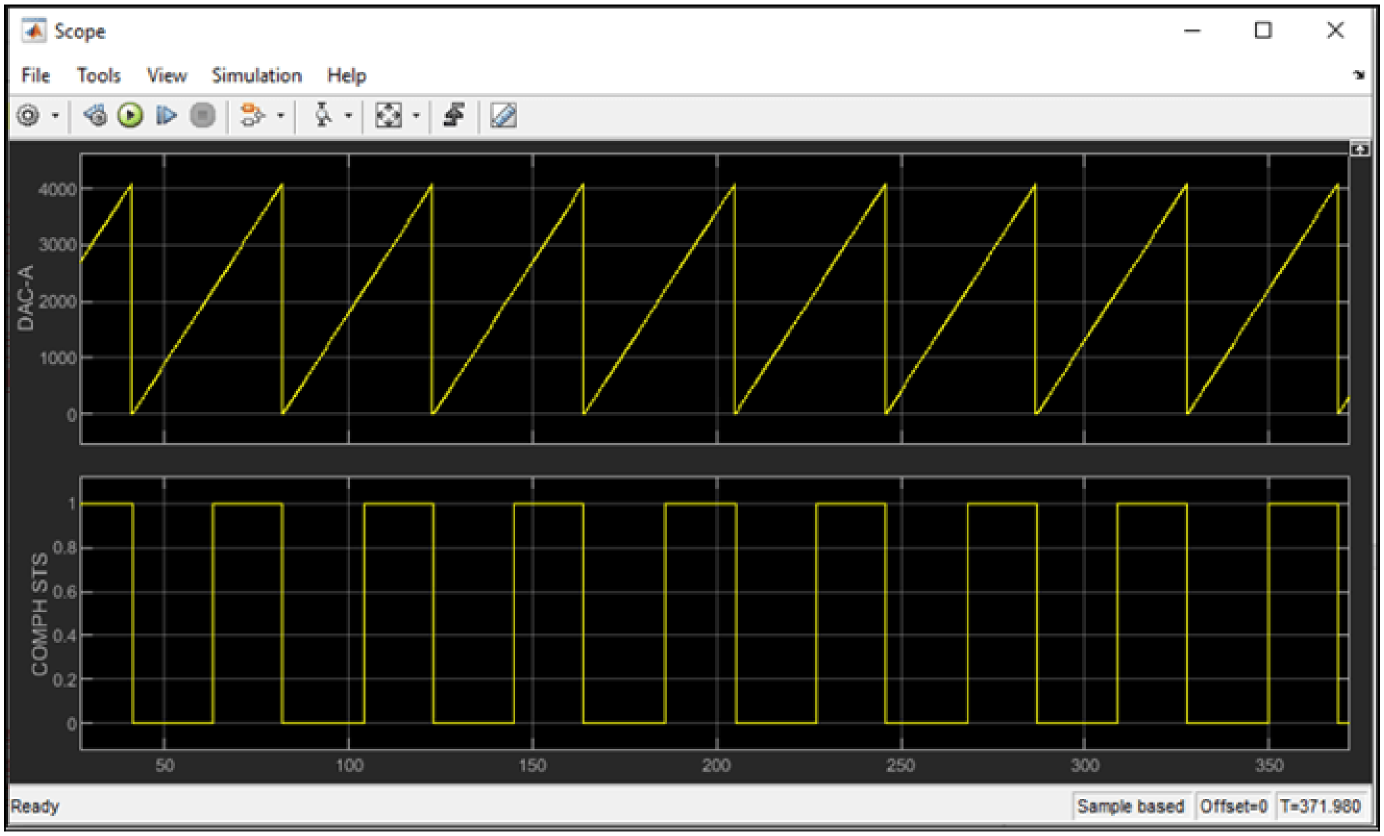
2. Use the diagnostic viewer to follow the build progress and wait until the code loads and runs on the target hardware.

3. Observe the logged data on the **Scope** block.

Analysis of DAC and CMPSS output signals

This section explains how to interpret the DAC-A and COMPH for CMPSS1.

- DAC-A voltage value ranges from 0-4095 which gives a Ramp voltage(0-3.3v) on the DAC-A pin. This voltage is fed to positive input of the CMPSS1 COMPH comparator.
- The internal DAC (DACH) of the CMPSS1 is configured to output a constant voltage of 2048(1.65v).
- The model is configured to run at a step time of 10ms. It will take 10ms * 4095 = 40.950 seconds for the DAC-A output to reach from 0 to 3.3V.
- The COMPH outputs **0** for the first half as the voltage on the positive input is less than the constant voltage (1.65v) on negative input. It then outputs **1** for second half as the voltage on the positive input is now more than the constant voltage (1.65v) on negative input.



Similar interpretation can be done for DAC-B and CMPSS2 from the Scope plot.

Other things to try

- Run the example on the `TI Delfino F280049C LaunchPad` hardware board.
- Connect the launchpad header pins for TI Delfino F280049C LaunchPad as listed below.

CMPSS1	DAC-A	COMPH
	J7[Pin 70]	J7[Pin 69]
CMPSS2	DAC-B	COMPL
	J3[Pin 30]	J3[Pin 23]

More About

- “CMPSS”
- F2807x/F2837xD/F2837xS/F28004x/F28003x/F2838x DAC

Read Data from IMU and Environmental Sensors

This example shows how to use C2000™ Microcontroller Blockset to read data from the BMI160 Inertial Measurement Unit (IMU) sensor and BME280 Environmental sensor that are part of the BOOSTXL-SENSORS BoosterPack™ plug-in module.

The example also shows how to read data from another BMM150 geomagnetic sensor connected as a breakout board to F28379D LaunchPad.

Required Hardware

- Texas Instruments™ F28379D LaunchPad
- BOOSTXL-SENSORS BoosterPack™ plug-in module, which includes built-in BMI160 Inertial Measurement Sensor, BMM150 Geomagnetic Sensor and BME280 Environmental Sensor.
- BMM150 Geomagnetic Sensor on an I2C-based board to be connected as a breakout board to F28379D LaunchPad.

Introduction

This example provides three Simulink models that use *TI Delfino F28379D Launchpad* as the Hardware board:

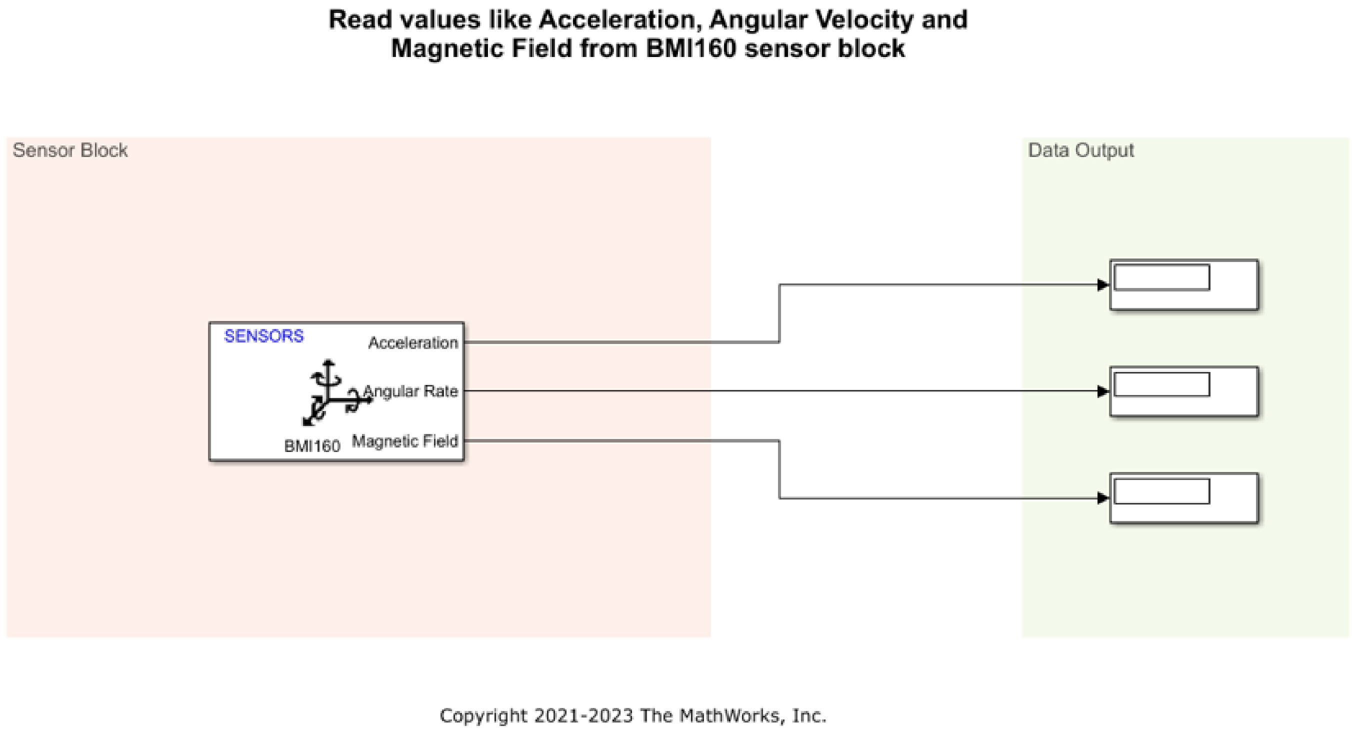
- c28x_i2c_bmi160_sensor helps you to read acceleration, angular rate and magnetic field as measured from both BMI160 and BMM150 sensors in the BOOSTXL-SENSORS module
- c28x_i2c_bme280_sensor helps you to read digital pressure and relative humidity as measured from a BME280 sensor in the BOOSTXL-SENSORS module
- c28x_i2c_bmm150_sensor helps you to read magnetic field as measured from a BMM150 sensor connected as a breakout board to F28379D LaunchPad

Model Configuration for Reading Data from BMI160 Sensor and BME280 Sensor

The model provided in this example for reading data from BMI160 sensor uses the corresponding block, BMI160 provided with the blockset.

To open the model, run this command at the MATLAB prompt:

```
open_system('c28x_i2c_bmi160_sensor');
```

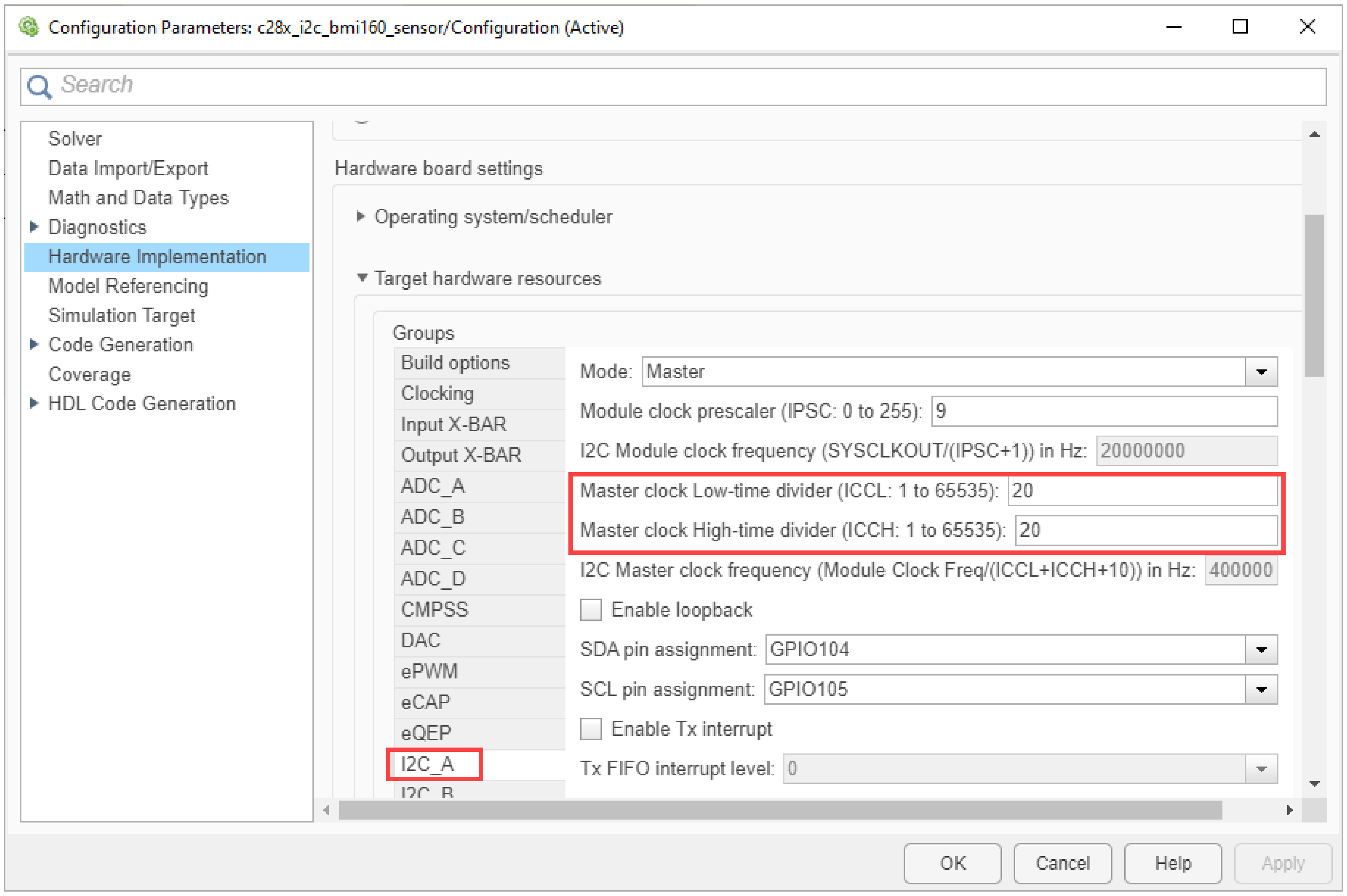

Read values like Acceleration, Angular Velocity and Magnetic Field from BMI160 sensor block
Sensor Block
Data Output
SENSORS
Acceleration
Angular Rate
Magnetic Field
BMI160
Copyright 2021-2023 The MathWorks, Inc.

In the BMI160 block in the model, the **I2C module** parameter is set to `I2C_A`. Therefore, to change the clock frequency, if required, change the settings for the same. To do this:

1. Go to Hardware tab, and click **Hardware Settings** to open the Configuration Parameters dialog box.

2. Go to **Hardware Implementation > Target hardware resources**, and select **I2C_A** tab.

3. Edit the values to change the clock frequency as required.


Configuration Parameters: c28x_i2c_bmi160_sensor/Configuration (Active)

Hardware board settings

Operating system/scheduler

Target hardware resources

Groups: Build options, Clocking, Input X-BAR, Output X-BAR, ADC_A, ADC_B, ADC_C, ADC_D, CMPSS, DAC, ePWM, eCAP, eQEP, I2C_A, I2C_B

Mode: Master

Module clock prescaler (IPSC: 0 to 255): 9

I2C Module clock frequency (SYSCLKOUT/(IPSC+1)) in Hz: 20000000

Master clock Low-time divider (ICCL: 1 to 65535): 20

Master clock High-time divider (ICCH: 1 to 65535): 20

I2C Master clock frequency (Module Clock Freq/(ICCL+ICCH+10)) in Hz: 400000

Enable loopback: unchecked

SDA pin assignment: GPIO104

SCL pin assignment: GPIO105

Enable Tx interrupt: unchecked

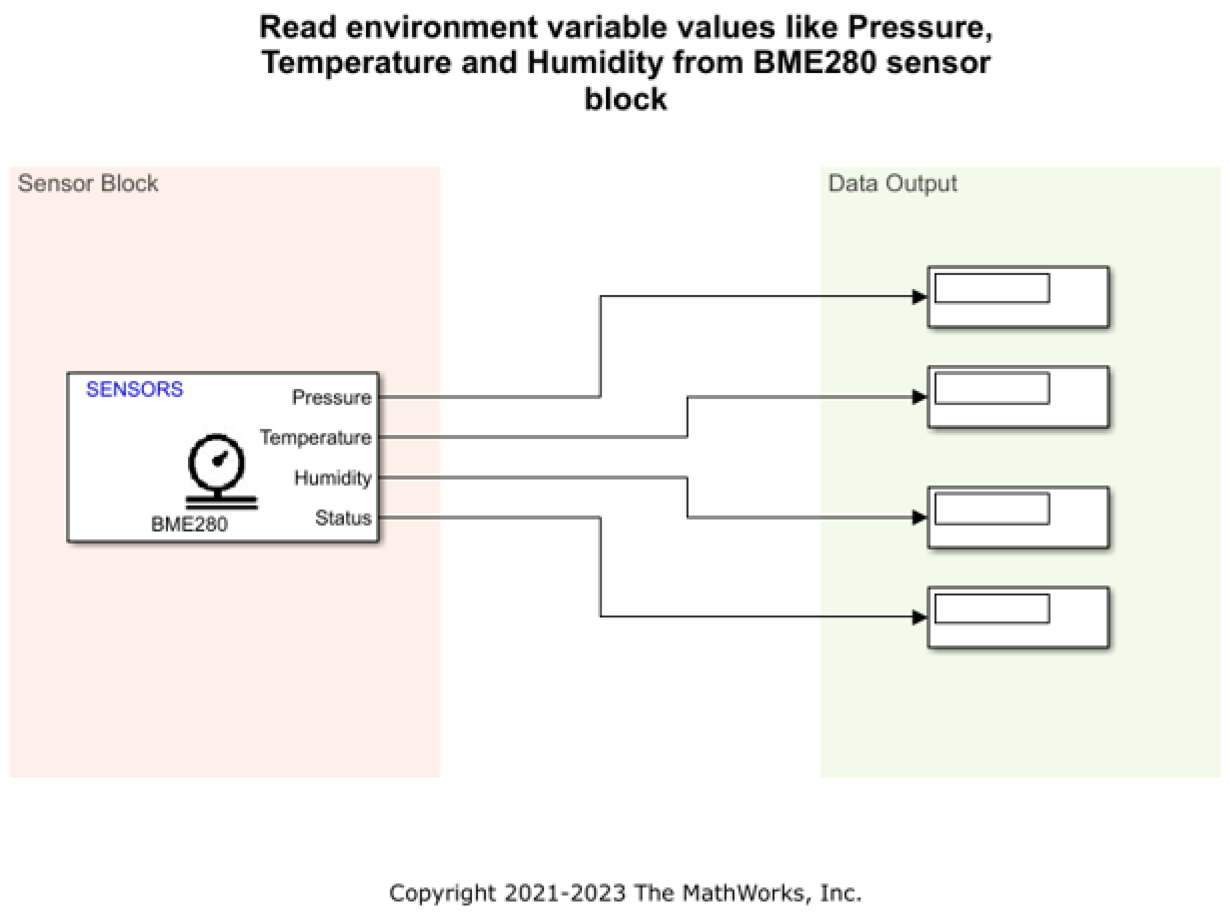
Tx FIFO interrupt level: 0

4. Click **Apply and then *OK**.

The other model provided in this example for reading data from BME280 sensor uses the corresponding block, BME280 provided with the blockset.

To open the model, run this command at the MATLAB prompt:

```
open_system('c28x_i2c_bme280_sensor');
```


Read environment variable values like Pressure, Temperature and Humidity from BME280 sensor block
Sensor Block
Data Output
SENSORS
Pressure
Temperature
Humidity
Status
BME280
Copyright 2021-2023 The MathWorks, Inc.

In the BME280 block in the above model, the **I2C module** parameter is set to `I2C_A`. Therefore, to change the clock frequency, if required, change the settings by following the same steps as described for BMI160.

You can also use the different options under **Advanced settings** inside the BME280 block to change the values for the filtering and sampling factors.

Note: The **I2C address** parameter value selected in each of the blocks (BMI160 and BME280) in the two models corresponds to the information provided in the **Schematics** section of the **BOOSTXL-SENSORS BoosterPack Plug-in Module User's Guide**.

Complete Hardware Connections and Read Data from BMI160 Sensor

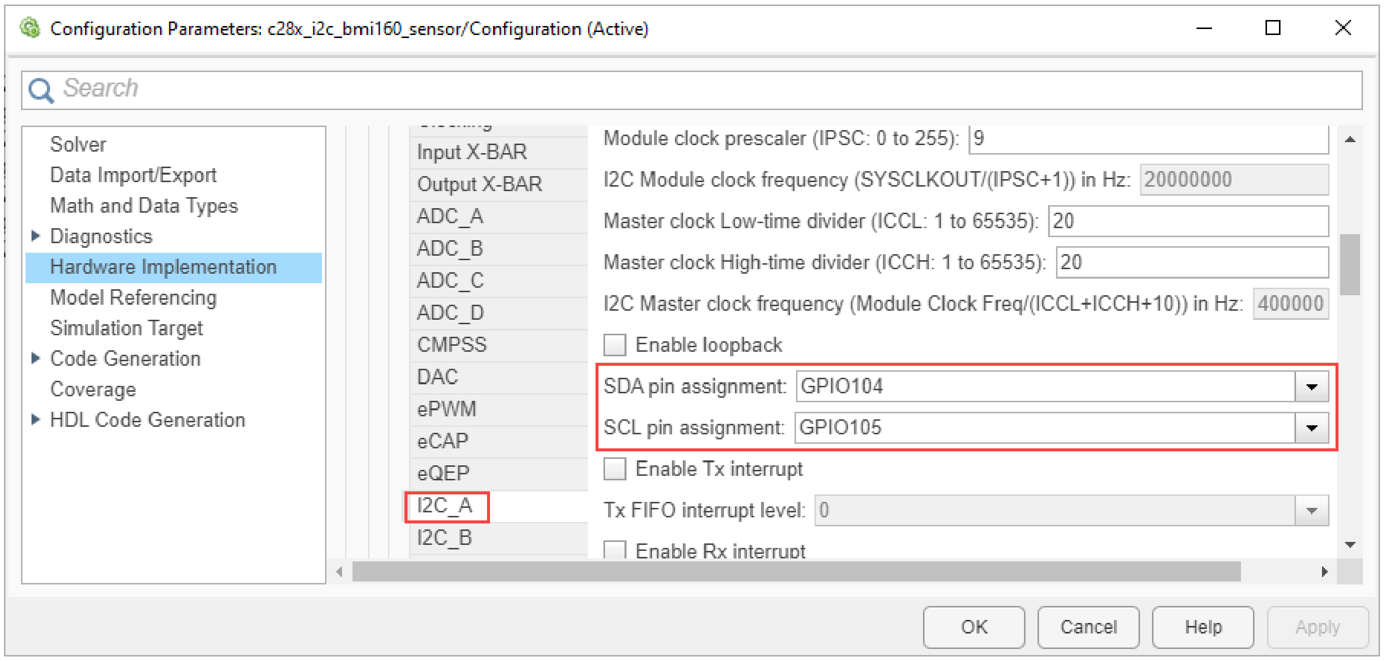
After you complete the configurations settings for the `c28x_i2c_bmi160_sensor` model, perform these steps:

1. Connect the BOOSTXL-SENSORS plug-in module to the F28379D LaunchPad. Connect GPIO104 and GPIO105 pins on the F28379D Launchpad to the J1.10 (SDA) and J1.9 (SCL) pins respectively on the BOOSTXL-SENSORS, and complete the other required connections like VDD and GND. For more details, refer to the the **Schematics** section of the **BOOSTXL-SENSORS BoosterPack Plug-in Module User's Guide**.

2. Connect the F28379D LaunchPad to the host computer.

2. In the **Configuration Parameters** window of `c28x_i2c_bmi160_sensor` model, click **Hardware Implementation** and navigate to **Target hardware resources > External mode**, and set the **Serial port in MATLAB Preferences** parameter to the corresponding COM port to which the Launchpad is connected. The COM port is available at **Device Manager > Ports** (COM & LTP) in Windows.

3. Select appropriate GPIO pins for SDA and SCL in **Hardware Implementation > I2C_A** pane, to communicate with BOOSTXL-SENSORS based on the actual hardware connections from the F28379D Launchpad.


Configuration Parameters: c28x_i2c_bmi160_sensor/Configuration (Active)

Module clock prescaler (IPSC: 0 to 255): 9
I2C Module clock frequency (SYSCLKOUT/(IPSC+1)) in Hz: 20000000
Master clock Low-time divider (ICCL: 1 to 65535): 20
Master clock High-time divider (ICCH: 1 to 65535): 20
I2C Master clock frequency (Module Clock Freq/(ICCL+ICCH+10)) in Hz: 400000
☐ Enable loopback
SDA pin assignment: GPIO104
SCL pin assignment: GPIO105
☐ Enable Tx interrupt
Tx FIFO interrupt level: 0
☐ Enable Rx interrupt

4. In the Hardware tab of Simulink model, click **Monitor & Tune**. You can observe from the Diagnostic Viewer that the code is generated for the model and the host connects to the target after loading the generated executable.

5. Rotate the board about its axis. You can observe that the value displayed in the Display block connected to **Angular Rate** output of the block is changing.

6. Change the orientation of the board. You can observe that the value displayed in the Display block connected to **Acceleration** output of the block is changing.

Complete Hardware Connections and Read Data from BME280 Sensor

After you complete the configurations settings for the `c28x_i2c_bme280_sensor` model, perform the same steps 1 to 4 as described in the previous section to specify the connections and run the model in External mode.

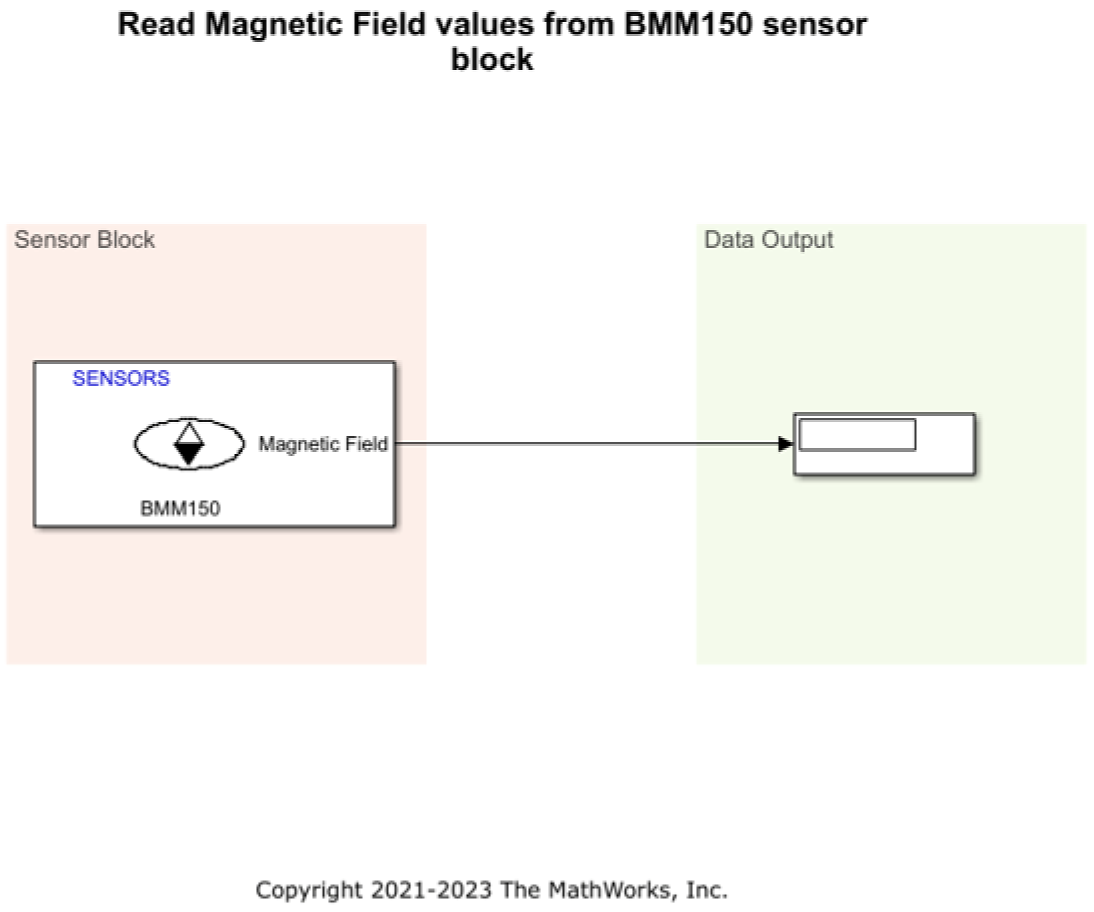
Then, observe the value displayed in the Display block connected to **Pressure**, **Temperature**, and **Humidity** output ports of the block. These values correspond to the current environmental conditions.

Model Configuration for Reading Data from BMM150 Sensor Connected as Breakout Board

The model provided in this example for reading data from BMM150 sensor uses the corresponding block, BMM150 provided with the blockset.

To open the model, run this command at the MATLAB prompt:

```
open_system('c28x_i2c_bmm150_sensor');
```


Read Magnetic Field values from BMM150 sensor block
Sensor Block
Data Output
SENSORS
Magnetic Field
BMM150
Copyright 2021-2023 The MathWorks, Inc.

In the BMM150 block in the above model, the **I2C module** parameter is set to `I2C_A`. Therefore, to change the clock frequency, if required, change the settings by following the same steps as described for BMI160.

You can also use the different options under **Preset value** parameter inside the BMM150 block to specify the optimum operating condition.

The board that you use can be a digital compass sensor based on BMM150 that uses an I2C interface. Refer to the board's specifications for I2C connection to F28379D LaunchPad, and specify the value for **I2C address** parameter accordingly.

Complete Hardware Connections and Read Data from BMM150 Sensor

After you complete the configurations settings for the `c28x_i2c_bmm150_sensor` model, perform these steps:

1. Connect the I2C-based board with the BMM150 sensor, to the F28379D LaunchPad, and complete the other required connections.

2. Connect the F28379D LaunchPad to the host computer.

2. In the **Configuration Parameters** window of `c28x_i2c_bmm150_sensor` model, click **Hardware Implementation** and navigate to **Target hardware resources > External mode**, and set the **Serial port in MATLAB Preferences** parameter to the corresponding COM port to which the Launchpad is connected. The COM port is available at **Device Manager > Ports** (COM & LTP) in Windows.

3. Select appropriate GPIO pins for SDA and SCL in **Hardware Implementation > I2C_A** pane, to communicate with I2C-based interface based on the actual hardware connections from the F28379D Launchpad.

4. In the Hardware tab of Simulink model, click **Monitor & Tune**. You can observe from the Diagnostic Viewer that the code is generated for the model and the host connects to the target after loading the generated executable.

5. Observe the current value in the Display block connected to **Magnetic Field** output of the block. Change the position of the board. You can observe that the value displayed in the Display block is changing.

Encode and Decode Serial Data Using C2000-based Hardware

This example shows how to use C2000™ Microcontroller Blockset to encode and decode serial data with TI's™ C2000-based hardware.

Introduction

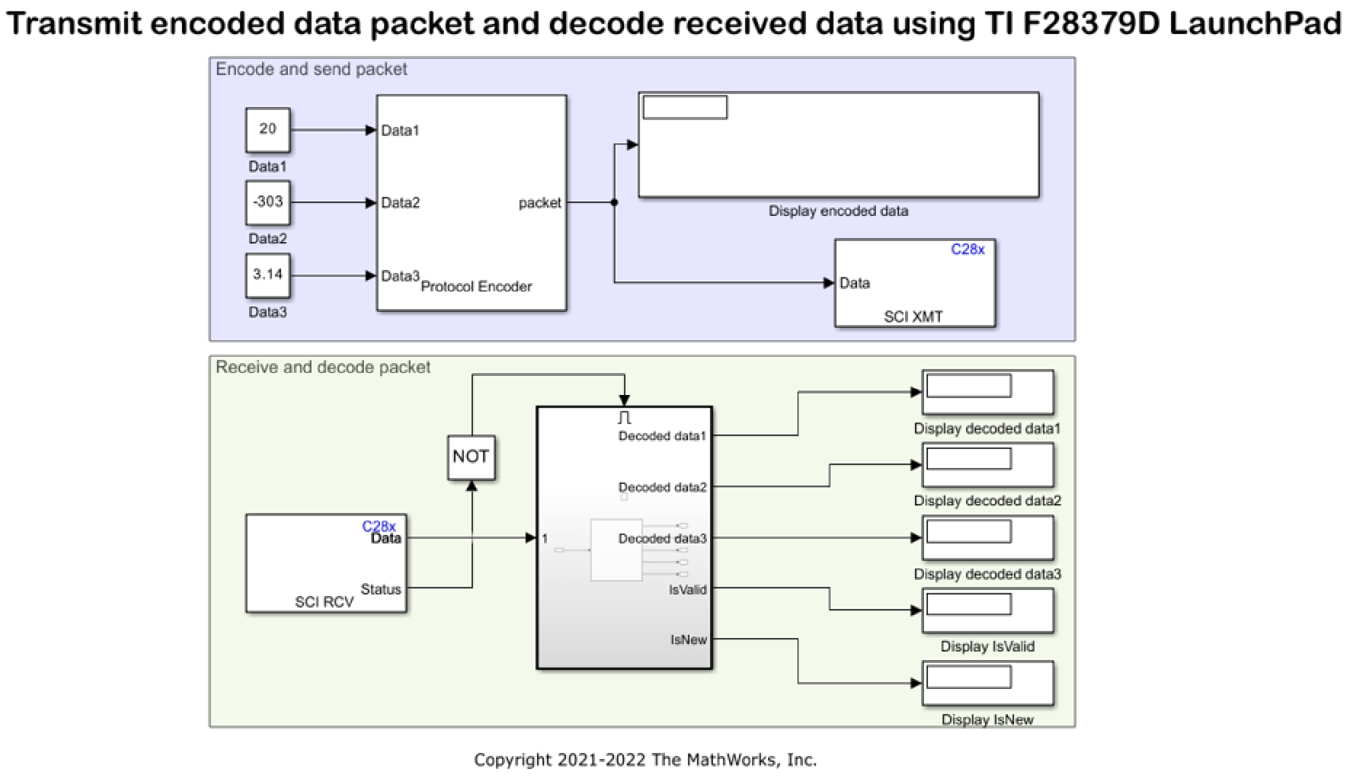
In this example, the Simulink model, performs these actions:

- At the transmission end, multiple fields are encoded into packet using Protocol Encoder block and the resulting uint8 byte stream is transmitted using C28x SCI Transmit block.
- At the Receiving end, the byte stream is received using C28x SCI Receive block and decoded into individual fields using Protocol Decoder block. The status output of SCI Receive block, which indicates a new data is available, is used to trigger the enabled subsystem containing Protocol Decoder block.

In this model, the Tx pin of SCI Module B sends serial data to the Rx pin of SCI Module B of the TI Delfino F28379D LaunchPad.

This model is configured to run in XCP-based External mode. For more information on External mode, see “Signal Monitoring and Parameter Tuning over XCP on Serial” on page 1-59.

```
open_system('c2000_encode_decode_packet');
```



The model provided in this example is preconfigured for the TI Delfino F28379D LaunchPad. You can run this model on any of the TI boards available under the **Hardware board** parameter in the

Simulink model. For more information on how to change the **Hardware board** parameter, see the **Step 2: Configure the Model for Connected Hardware** section of this example.

Required Hardware

To run this example, you must have the following hardware:

- Texas Instruments™ Delfino F28379D LaunchPad
- Connecting wires
- USB cable

Step 1: Connect TX and RX Pins on F28379D Launchpad

1. Connect your TI Delfino F28379D LaunchPad to your computer using the USB cable.

2. Connect the Tx pin of SCI B module (GPIO18 pin) to the Rx pin of SCI B module (GPIO19 pin). This connection is a loopback connection.

Step 2: Configure the Model for Connected Hardware

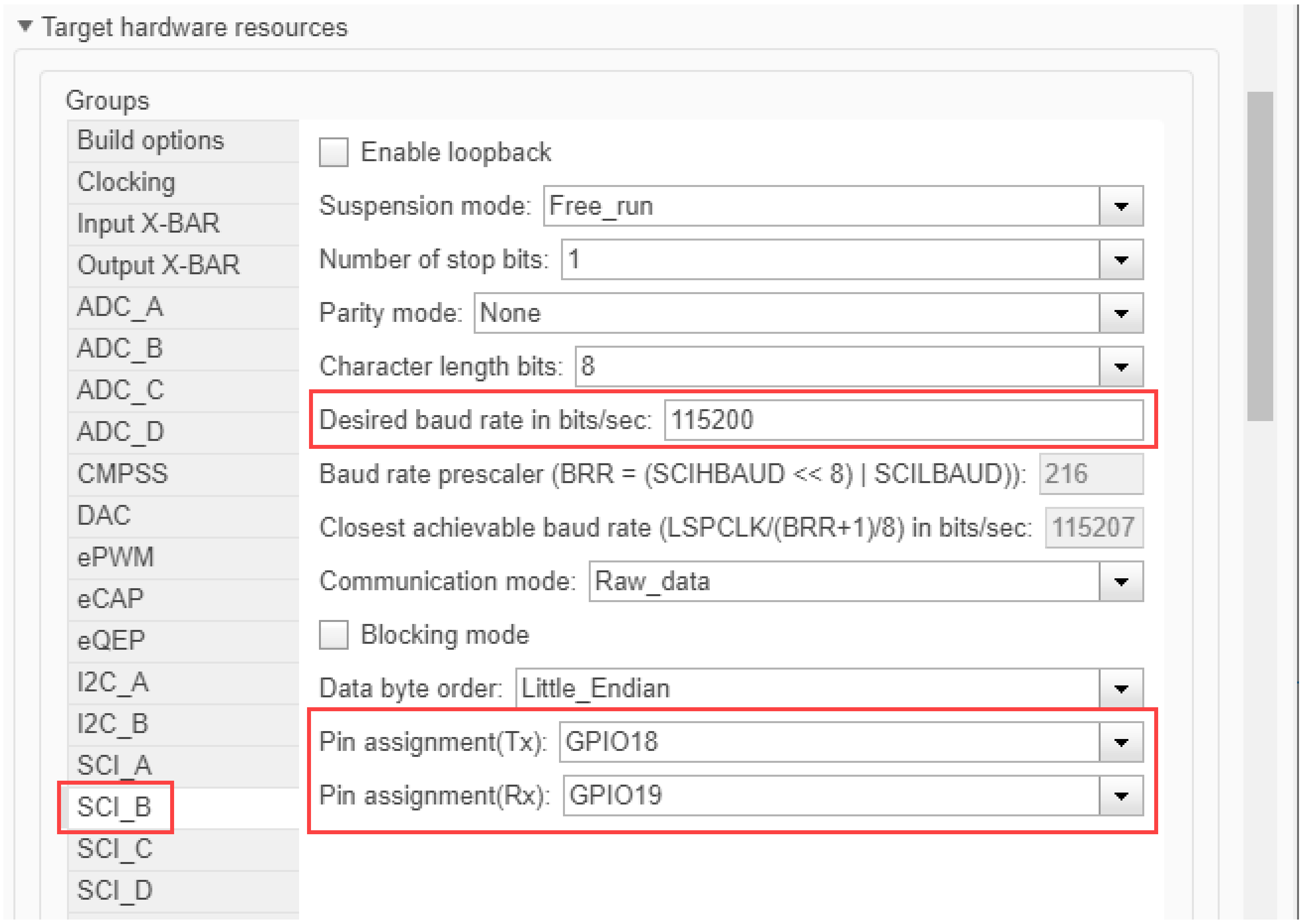
1. Open the Simulink model model. This model is configured to run on XCP-based External mode.

2. To configure the model, click **Hardware Settings** in the **Hardware** tab of the Simulink toolbar.

3. In the Configurations Parameters dialog box, select **Hardware Implementation**.

4. From the **Hardware board** list, select the TI's C2000-based processor that you are using.

5. Under **Target Hardware Resources**, click **SCI_B** tab and configure the properties including baud rate and pin assignments.



Target hardware resources

Groups: Build options, Clocking, Input X-BAR, Output X-BAR, ADC_A, ADC_B, ADC_C, ADC_D, CMPSS, DAC, ePWM, eCAP, eQEP, I2C_A, I2C_B, SCI_A, SCI_B, SCI_C, SCI_D

- [ ] Enable loopback
- Suspension mode: Free_run
- Number of stop bits: 1
- Parity mode: None
- Character length bits: 8
- Desired baud rate in bits/sec: 115200
- Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)): 216
- Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec: 115207
- Communication mode: Raw_data
- [ ] Blocking mode
- Data byte order: Little_Endian
- Pin assignment(Tx): GPIO18
- Pin assignment(Rx): GPIO19

If you are using any other GPIO pins for communication, change the **Pin assignment** parameter value selection accordingly.

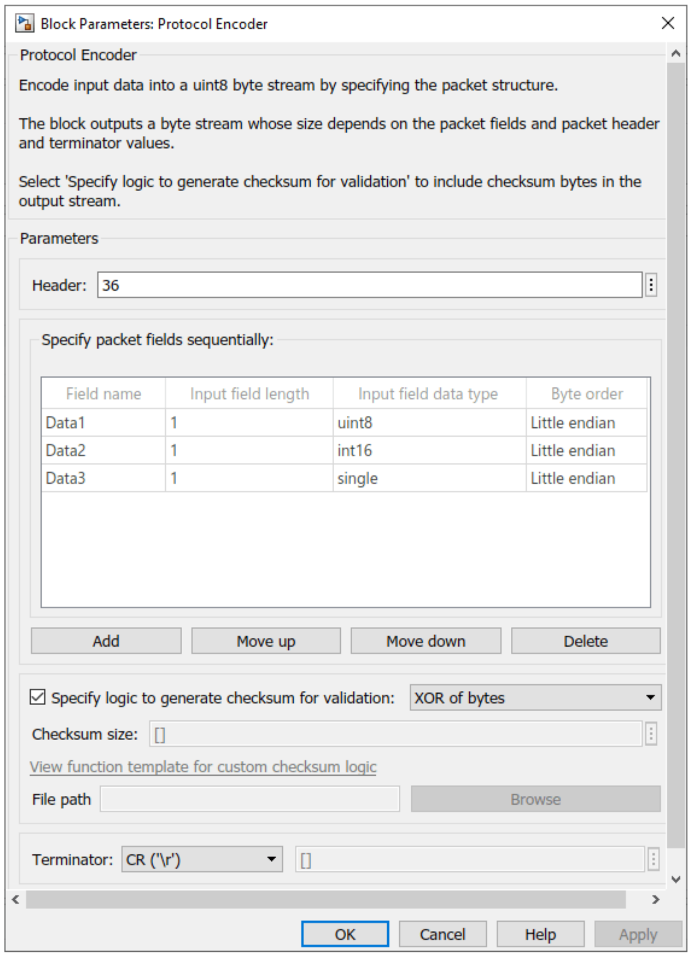
6. Click **Apply**. Click **OK** to close the dialog box.

Step 3: Configure Blocks in the Simulink Model

The packet structure used in this example is:

Header (36)	Data1 Data type: uint8	Data2 Data type: int16	Data1 Data type: single	Checksum (XOR based)	Terminator (CR)

Double-click the blocks and verify the parameter values specified in the Block Parameters dialog box.



Block Parameters: Protocol Encoder

Protocol Encoder

Encode input data into a uint8 byte stream by specifying the packet structure.

The block outputs a byte stream whose size depends on the packet fields and packet header and terminator values.

Select 'Specify logic to generate checksum for validation' to include checksum bytes in the output stream.

Parameters

Header: 36

Specify packet fields sequentially:

Field name	Input field length	Input field data type	Byte order
Data1	1	uint8	Little endian
Data2	1	int16	Little endian
Data3	1	single	Little endian

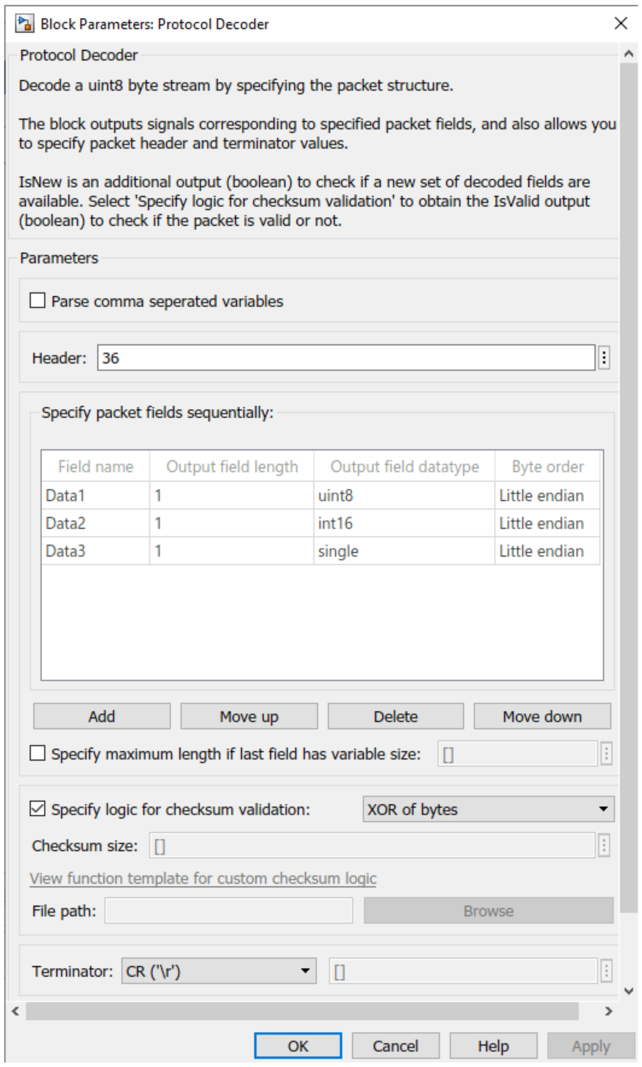
☑ Specify logic to generate checksum for validation: XOR of bytes

Checksum size: []

View function template for custom checksum logic

File path

Terminator: CR ('\r') []



Block Parameters: Protocol Decoder

Protocol Decoder

Decode a uint8 byte stream by specifying the packet structure.

The block outputs signals corresponding to specified packet fields, and also allows you to specify packet header and terminator values.

IsNew is an additional output (boolean) to check if a new set of decoded fields are available. Select 'Specify logic for checksum validation' to obtain the IsValid output (boolean) to check if the packet is valid or not.

Parameters

☐ Parse comma seperated variables

Header: 36

Specify packet fields sequentially:

Field name	Output field length	Output field datatype	Byte order
Data1	1	uint8	Little endian
Data2	1	int16	Little endian
Data3	1	single	Little endian

☐ Specify maximum length if last field has variable size: []

☑ Specify logic for checksum validation: XOR of bytes

Checksum size: []

View function template for custom checksum logic

File path:

Terminator: CR ('\r') []

For other blocks, the parameters are:

```
Block           | Parameter Name                     | Value
--------------------------------------------------------------------------------
Constant        | Interpret vector parameters as 1-D | selected
                | Sample time                        | inf
SCI Transmit    | SCI module                         | B
SCI Receive     | SCI module                         | B
                | Data type                          | uint8
                | Data length                        | 10
                | Sample time                        | 0.1
Display Status  | Format                             | short
```

The value for **Data length** parameter of SCI Receive block is set to *10*, which is the sum of these values:

```
Header size (1 byte) + Data1 (1 byte) + Data2 (2 bytes) + Data3 (4 bytes) + Checksum (1 byte) +
```

Step 4: Run the Model in XCP-based External Mode

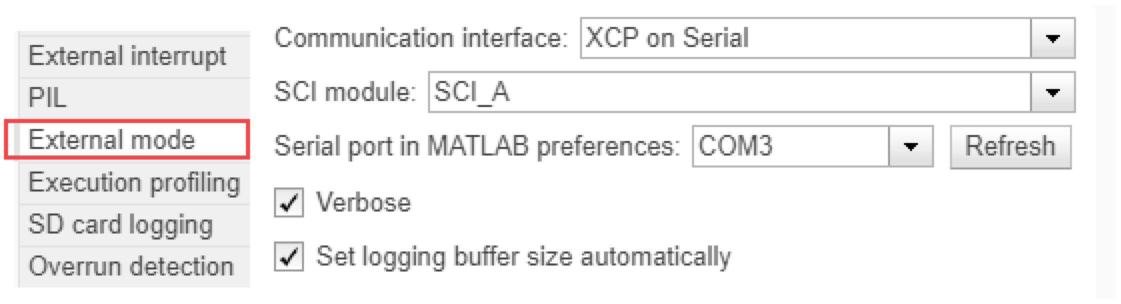
You can simulate the model in XCP-based External mode, which deploys the model as a C code on the hardware. The code obtains real-time data from the hardware.

Set Up the Model for XCP-based External Mode

1. To configure the model, click **Hardware Settings** in the **Hardware** tab of the Simulink toolbar.

2. In the Configurations Parameters dialog box, select **Hardware Implementation**.

3. Under **Target Hardware Resources**, click **External mode** tab and select **XCP on Serial** and **SCI_A** for *Communication interface* and *SCI module* parameters respectively. Select the corresponding value for the COM port of the host computer to which the C2000-based processor is connected.



Communication interface: XCP on Serial
SCI module: SCI_A
Serial port in MATLAB preferences: COM3
✓ Verbose
✓ Set logging buffer size automatically

4. Click **Apply**. Click **OK** to close the dialog box.

The Stop Time (under **Simulation** tab) is already set to `inf`.

Run the Model on XCP-based External Mode

1. To tune parameters and monitor signals in this model while the application runs on the target hardware, on the **Hardware** tab, click **Monitor & Tune**.

The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

At each time step, data specified in the `Constant` blocks are encoded into uint8 byte stream (packet) and transmitted by the TX1 pin to the RX1 pin of your C2000-based processor. You can see the uint8 byte stream generated as per the packet structure in the `Display encoded data` block.

The RX1 pin receives the uint8 byte stream which is decoded using `Protocol Decoder` block and displays it on the `Display Decoded data` blocks. Observe the output in the `Display Decoded data` blocks which will be same as the value given in the `Constant` blocks at the transmission end.

2. Try changing the values in the `Constant` blocks connected as input to the Protocol Encoder block, and observe if the same values are getting decoded by the Protocol Decoder block.

3. To stop running the model, click **Stop**.

Other Things to Try

- Try specifying a different packet structure using the block properties and observe the encoded data and decoded data in the `Display` blocks.

More About

- Protocol Encoder
- Protocol Decoder

Asynchronous Scheduling

This example shows how to use the Texas Instruments™ C28x™ peripherals and Hardware Interrupt blocks to control the real-time execution of Simulink® function-call subsystems in an asynchronous fashion.

Required Hardware

- Spectrum Digital F2808/F2812/F28335 eZdsp or Texas Instruments LaunchPad/controlSTICK/controlCARD with docking station
- Oscilloscope and probes

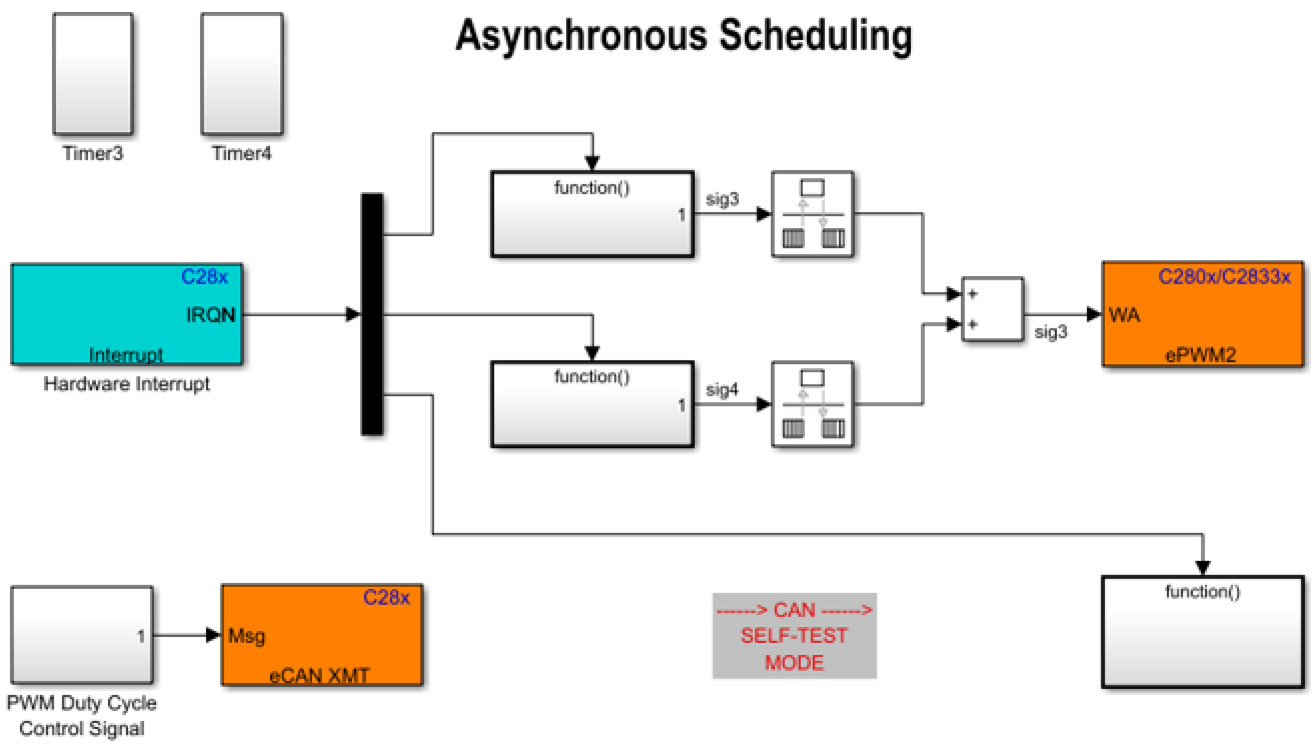
Available Models

These are the Simulink models available for different C2000 processors:

- F281x-based board: c281x_asyncscheduling_ert.slx
- F280x/F2823x/F2833x-based board: c280x_2833x_asyncscheduling_ert.slx
- Piccolo F2803x/F2806x-based board: c280xx_asyncscheduling_ert.slx
- Concerto F28M35x/F28M36x- based board: c28M3xx_asyncscheduling_ert.slx
- Piccolo F2807x/F28004x or Delfino F2837xS/F2837xD-based board: c2807x_2837xx_asyncscheduling_ert.slx

Note: To use the F28M35x/F28M36x controlCARD, you need C2000™ Microcontroller Blockset.

Example Model



The EV Timer or ePWM blocks are used to configure the timer interrupts. The timer interrupts are triggered based on the timer period, and the eCAN message receive interrupt is triggered when a message is received. The Hardware Interrupt block triggers the interrupt service routines (ISR) for the timer interrupts as well as for the eCAN message receive interrupt. The ISRs in turn call the function-call subsystems connected to the Hardware Interrupt block output ports.

The outputs of the first two subsystems are free-running counters. The sum of the counters is used to control the duty cycle of PWMB for F2812 or ePWM2 for F2808/F28335. The PWM waveform duty cycle increases linearly from 0 to 100%. The third subsystem contains an eCAN Receive block whose message output controls the duty cycle of the PWM block (PWMA for F2812 or ePWM1 for F2808/F28335). The duty cycle varies from 0 to 100% because the eCAN messages are received from the eCAN Transmit block.

Note: The **Self-Test** mode of eCAN_A is enabled to connect the eCAN_A transmitter and receiver internally to avoid external connection between the transmitter and the receiver. For disabling the **Self-Test** mode of eCAN_A, the transmitter and the receiver must be connected together externally.

Run the Model on the Board

To configure, build, and deploy the application that corresponds to your Simulink model:

“Model a Fault-Tolerant Fuel Control System” example on the STM32F746G Discovery board.

1. Open the model corresponding to your target hardware. Each model is configured for a default target hardware. To select a different target hardware, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**.
2. Go to **Target hardware resources** > **eCAN_A**, select **Self-Test Mode**, and click **OK**.
3. On the **Hardware** tab, Click **Build, Deploy & Start** > **Build Stand-Alone** to generate, build, load, and run the program.
4. Observe the changes of the PWM waveform on the oscilloscope.

More About

- c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM
- C28x eCAN Transmit

LIN-Based Control of PWM Duty Cycle

This example shows how to use the C28035 LIN Receive, LIN Transmit, and PWM blocks to generate a pulse waveform.

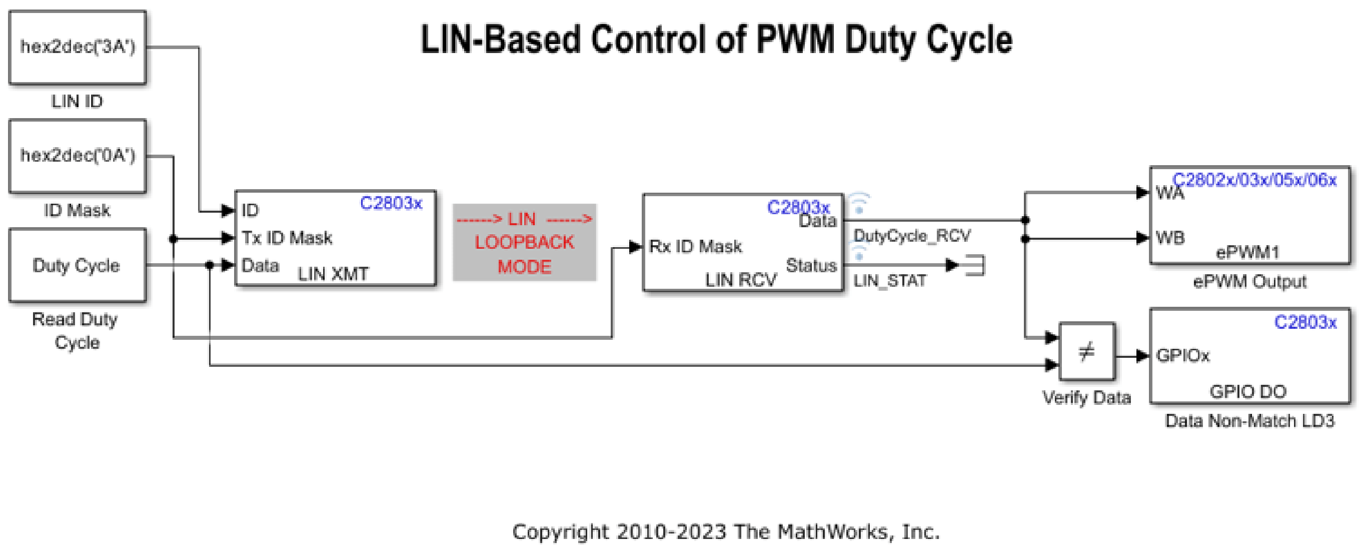
Required Hardware

- Texas Instruments™ F28035 controlCARD with docking station
- Oscilloscope and probes

Available Models

- Texas Instruments F28035 controlCARD: c28035lintest.slx

Example Model



This example model runs a single LIN node in loopback mode. The Read Duty Cycle subsystem passes the duty cycle values to the LIN Transmit block. The LIN Transmit block transmits the values to the LIN Receive block, which sends them to the ePWM Output block. The Verify Data block compares the original duty cycle values with the values from the LIN Receive block. If the values do not match, the Data Non-Match LD3 block flashes the LD3 LED on the C28035 controlCARD. In addition, a LIN_STAT signal is attached to the Status output from the LIN Receive block. To know the LIN communication status, you can check the LIN_STAT variable in the generated code running in Texas Instruments Code Composer Studio™.

The duty cycle of the generated pulse waveform is determined by the relative ratio of the received pulse width value and the pulse period, which is fixed at 64,000 clock cycles. The duty cycle toggles between 25%, 50%, and 75% based on the selection.

PWM duty cycle can be changed by double-clicking the Read Duty Cycle subsystem and selecting either 25%, 50%, or 75% value from the window that opens up.

Hardware Connections

Connect the output of PWM1 on the board to the analog input of the oscilloscope.

The LIN module is set to work in loopback mode. No external LIN hardware is needed because the LIN TX/RX signals are emulated in the software.

Setup LIN Loopback Mode

1. Browse to **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources**.
2. Set **LIN mode** to **Master** in LIN page.
3. Select **Enable loopback** on the LIN page.
4. Set **ID slave task byte** to a value between 0x00 and 0xFF.

For the LIN Transmit and LIN Receive blocks:

- Use the same ID Mask as input for both LIN TX ID Mask and RX ID Mask.
- Use **LIN ID** to make (`LIN ID` XOR `ID Mask`) == `ID-Slave Task Byte`.

For information about how to setup the LIN peripheral, see “Configuring LIN Communications” on page 1-42.

Monitor and Tune the Model

1. Open the model, double-click the **Duty Cycle** block, and select a new duty cycle value.
2. In the **Configuration Parameters** window, click **Hardware Implementation** and go to **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at **Device Manager** > **Ports** (COM & LTP) in Windows.
3. Go to **Hardware** tab and click **Monitor & Tune**.
4. Use the diagnostic viewer to follow the build progress, and wait until the code loads and runs on the target hardware.
5. Observe the change of the PWM waveform on the oscilloscope.
6. Change the duty cycle while in External mode and observe the changes in the PWM waveform on the oscilloscope.

More About

- C2803x LIN Receive
- C2803x LIN Transmit

Using the I2C Bus to Access Sensors

This example shows how to use the I2C blocks to communicate with I2C based devices.

Introduction

In this example you will learn how to configure and use I2C blocks to:

- Access the EEPROM
- Read the accelerometer and gyroscope data from I2C based sensor

Required Hardware

This example requires different hardware configurations for different tasks.

To access the EEPROM

Spectrum Digital F28335/F2808 eZdsp board

Note: Any C2000 (except c281x) controlSTICK or ControlCARD with docking station is not equipped with I2C EEPROM. You must add I2C EEPROM to this board to run the model.

Available models:

- c28x_i2c_eeprom.slx
- c28x_i2c_eeprom_interrupt.slx

To read accelerometer and gyroscope data from I2C based sensor

- Texas Instruments™ F28069/F28377S/F28379D/F28004x LaunchPad
- Sensors BoosterPack (BOOSTXL-SENSORS)

Available models:

c28x_i2c_sensor.slx

Read and Write Data to EEPROM

The model writes four bytes to EEPROM and reads back the data from the corresponding EEPROM address to show successful communication.

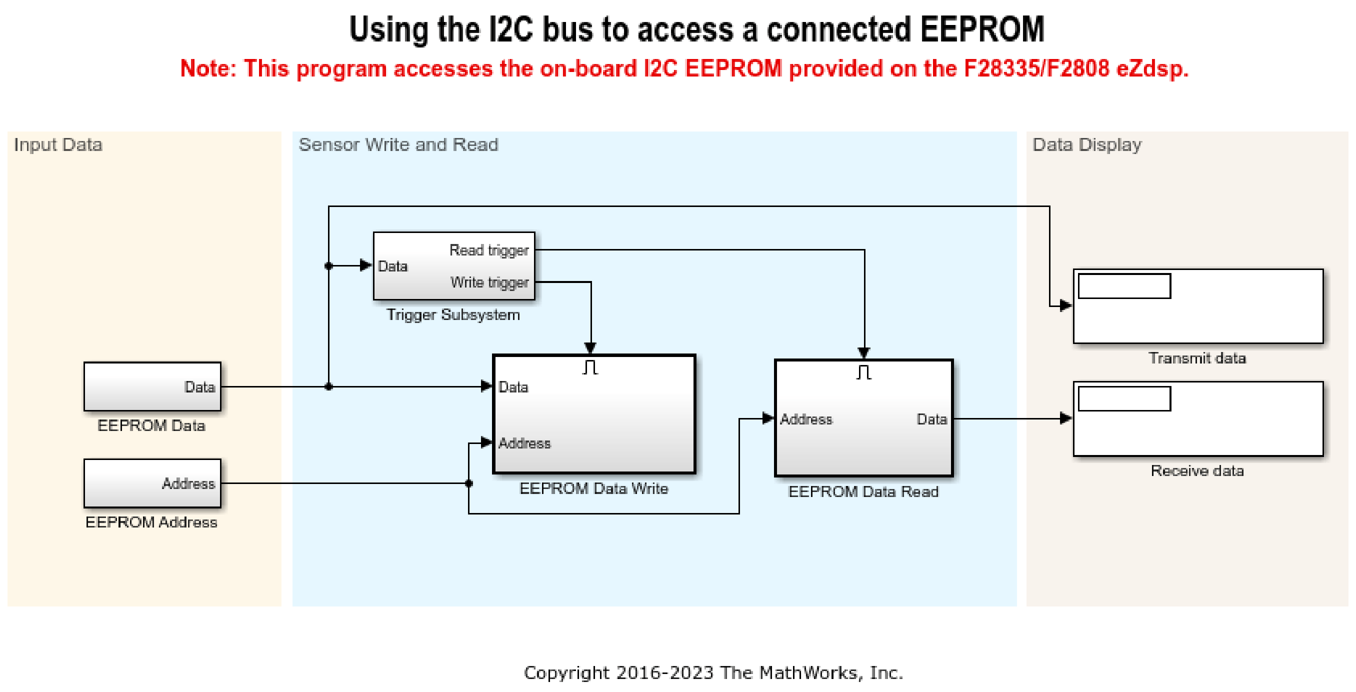
The model consists of these subsystems:

- **EEPROM Data**: Contains the data that is written to the EEPROM.
- **EEPROM Address**: Contains the EEPROM address where the data is written to.
- **Trigger Subsystem**: Triggers the EEPROM Data Write and EEPROM Data Read subsystems alternatively. The EEPROM Data Write Subsystem is triggered when there is a change in the EEPROM data. The EEPROM Data Read is triggered when there is no EEPROM write.
- **EEPROM Data Write Subsystem**: Consists of the I2C Transmit block, which sends the EEPROM address and the data along with a stop condition to the EEPROM at the end of the data write.
- **EEPROM Data Read Subsystem**: Consists of the I2C Transmit block, which sends the EEPROM address from which data is read. A wait logic using a while loop ensures that the address is sent to

the EEPROM from the Tx FIFO. The I2C Receive block reads the data of specified length along with a stop condition at the end of the data write.

You can run the model in External mode, monitor the transmitted and received data using display blocks. You can change the data in "EEPROM Data" subsystem and see the changes in the display blocks.

1. Open the c28x_i2c_eeprom model. This model is configured for **TI F28335** hardware board. To configure the model to run on other TI C2000 processors, you can change the 'Hardware board' parameter in the Configuration Parameters > Hardware Implementation pane.
2. In the **Configuration Parameters** window, click **Hardware Implementation** and navigate to **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at **Device Manager** > **Ports** (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301
3. Run the model and observe the "Receive data" and "Transmit data" display blocks.
4. Change the four byte data in "EEPROM Data" subsystem and observe the change getting reflected in "Receive data" block.



Use interrupts to Read and Write Data to EEPROM

The model writes four bytes to EEPROM and reads back the data from the corresponding EEPROM address to show successful communication.

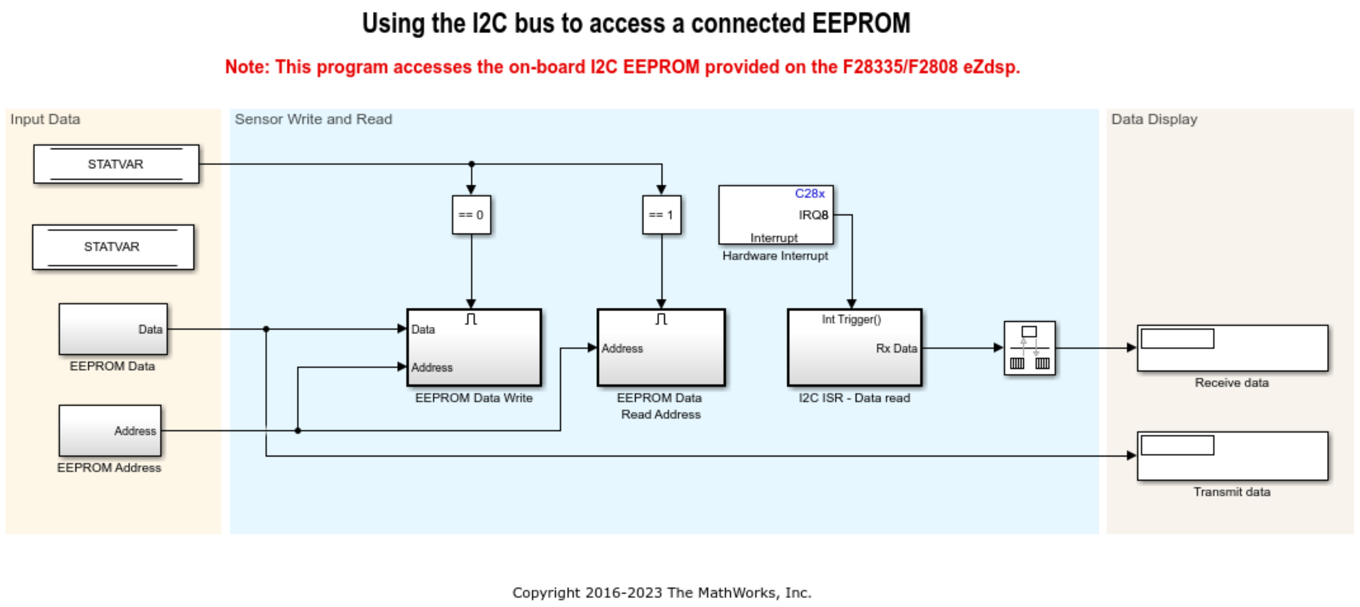
The model consists of these subsystems:

- **EEPROM Data**: Contains the free-running counter data that is written to the EEPROM.
- **EEPROM Address**: Contains the EEPROM address where the data is written to.

- **Trigger Logic**: Triggers EEPROM Data Write and EEPROM Data Read subsystems alternatively. The trigger logic consists of the STATVAR data store variable, which is updated to start a write/read operation in the I2C ISR block when the SCD interrupt is triggered.
- **EEPROM Data Write Subsystem**: Consists of the I2C Transmit block, which sends the EEPROM address and the data along with a stop condition to the EEPROM at the end of the data write.
- **EEPROM Data Read Subsystem**: Consists of the I2C Transmit block, which sends the EEPROM address to the EEPROM to initiate the EEPROM read operation. An ARDY interrupt is generated when the address is sent to the EEPROM from the TX FIFO. The I2C Receive block is then used in the interrupt service routine (ISR) to read the data of specified length along with a stop condition at the end of the data write.

You can run the model in External mode and monitor the transmitted and received data using display blocks.

1. Open the c28x_i2c_eeprom_interrupt model. This model is configured for **TI F28335** hardware board. To configure the model to run on other TI C2000 processors, you can change the target hardware in the Configuration Parameters > Hardware Implementation pane.
2. In the **Configuration Parameters** window, click **Hardware Implementation** > **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at **Device Manager** > **Ports** (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301
3. Ensure that **system interrupt**, **SCD interrupt**, and **ARDY interrupt** are selected in **Configuration Parameters > Hardware Implementation > I2C**.
4. Run the model and observe the free-running counter data in "Receive data" and "Transmit data" display blocks.



Read the Accelerometer and Gyroscope Data from I2C-based Sensor

The model configures the registers in Sensor Boosterpack reads the data from the accelerometer and gyroscope.

The model consists of these subsystems:

- **Initialization Subsystem**: The Initialization Subsystem is triggered only once at the beginning. The Initialization Subsystem performs the required sensor initialization by sending a series of commands with 1 ms delay between each command to the sensor. For each command, the I2C transmit block sends the address and the data along with the stop condition to the sensor. At the end of the initialization, the read cycle is initiated by sending the address of the register to be read to the sensor using the I2C Transmit block.
- **Sensor Read**: The Sensor Read Subsystem consists of the I2C Receive block, which reads the sensor data along with the stop condition that stops the current read cycle. Then, the I2C Transmit block sends the address to the sensor to initiate the data read for the next cycle. To ensure that the I2C Receive block is executed before the I2C Transmit block, set the block priorities for both the blocks by right-clicking and selecting **Properties** > **Priority**.
- **Data Realignment Subsystem**: Aligns the data as required before displaying it using a display block.

You can run the model in External mode and monitor the accelerometer and gyroscope data in the scope.

1. Open the c28x_i2c_sensor model. This model is configured for **TI Delfino F2837xS** hardware board. To configure the model for other TI C2000 processors, you can change the hardware board in the **Configuration Parameters > Hardware Implementation** pane.

2. In the **Configuration Parameters** window, click **Hardware Implementation** and navigate to **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at **Device Manager** > **Ports** (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301.

3. Ensure that the Sensor Boosterpack is connected to the Launchpad, and select the corresponding I2C module (A/B) on Tx and Rx blocks. I2C_B module exists only for F2807xs/F2837xS and F2837xD processors.

4. Select appropriate GPIO pins for **SDA** and **SCL** in Hardware Implementation > I2C_A/I2C_B to communicate with Sensor Boosterpack.

5. Run the model and observe the accelerometer and gyroscope sensor data in the scope.

More About

- C28x I2C Receive
- C28x I2C Transmit

Inter-Processor Communication Using IPC Blocks

This example shows how to use the IPC blocks to communicate between multiple cores of multi-core of Texas Instruments™ C2000™ Microcontroller Blockset using Simulink® models.

In this example, you will learn how to:

- Send scalar and vector data from one core using IPC Transmit block
- Receive data at other core using IPC Receive block in polling and interrupt modes

Required Hardware

- Texas Instruments™ Delfino F2837xD controlCARDs or F28379D LaunchPad
- Texas Instruments™ F2838x ControlCard
- A USB serial cable, if your hardware provides serial over USB capabilities

A USB serial cable is used to establish a serial connection between the host computer and the target hardware. For more information, see “Set Up Serial Communication with Target Hardware” on page 1-6.

To select a different target hardware, in the Simulink Editor, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**.

Task 1 - Communicate between CPU1 and CPU2 for TI Delfino F2837xD using IPC blocks

This task of an example is configured for TI Delfino F28377D ControlCard with a docking station. F2837xD is a dual-core processor. In this task, You will flash two different models for two cores:

- `CPU1 model`
- `CPU2 model`

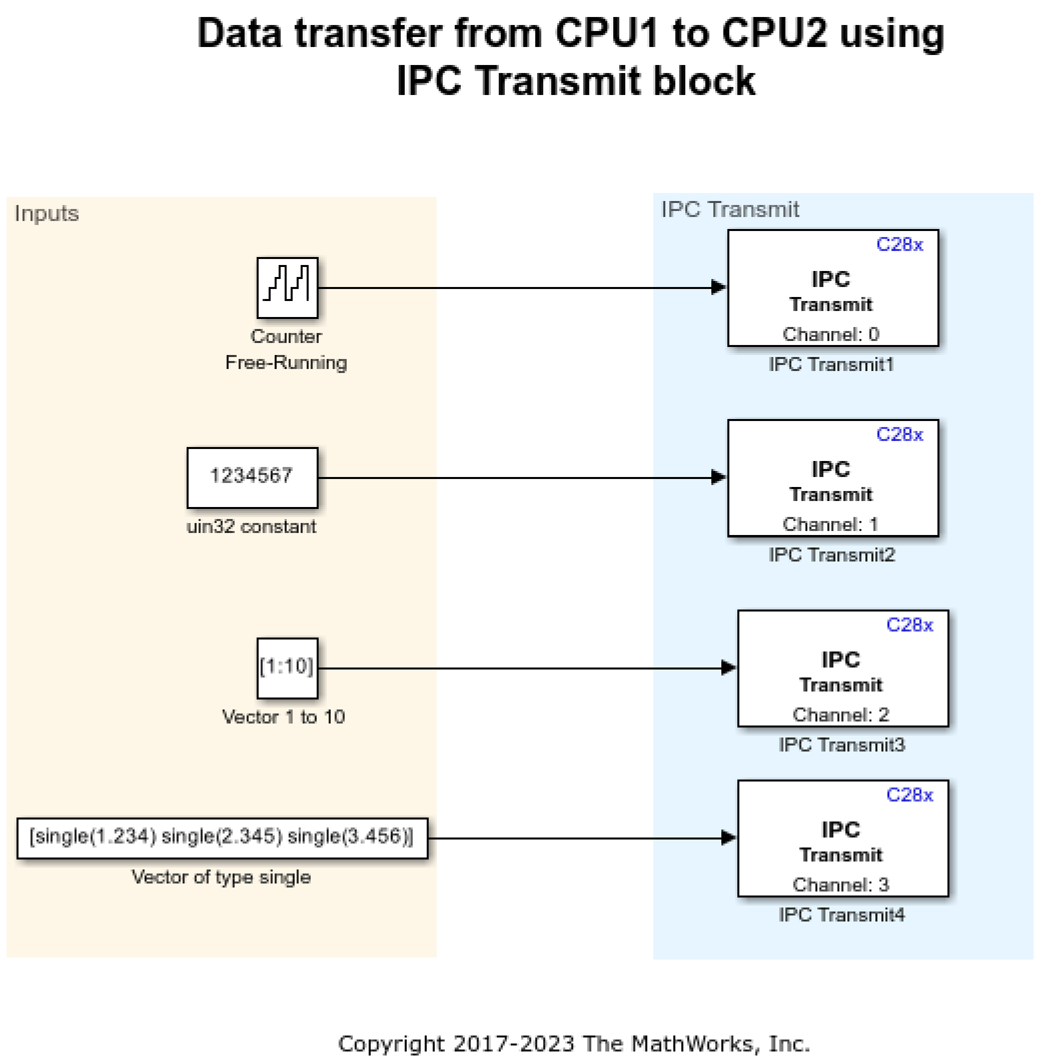
This example uses four different channels to transmit and receive data between cores. The following data is transmitted from CPU1 to CPU2:

- A scalar uint16 counter value is sent from channel 0
- Scalar data of type uint32 is sent from channel 1
- A vector [1x10] of type uint16 is sent from channel 2
- A vector [1x3] of type single is sent from channel 3

CPU1 model for TI F2837xD

Open the c2837xd_ipc_cpu1_tx model, and click **Build, Deploy & Start** under **Hardware** tab or press **Ctrl+B** to build and download the executable file on CPU1.

```
open_system('c2837xd_ipc_cpu1_tx');
```



CPU2 model for TI F2837xD

CPU2 receives data using IPC Receive Block in Polling and Interrupt Modes.

1. Open the c2837xd_ipc_cpu2_rx model.

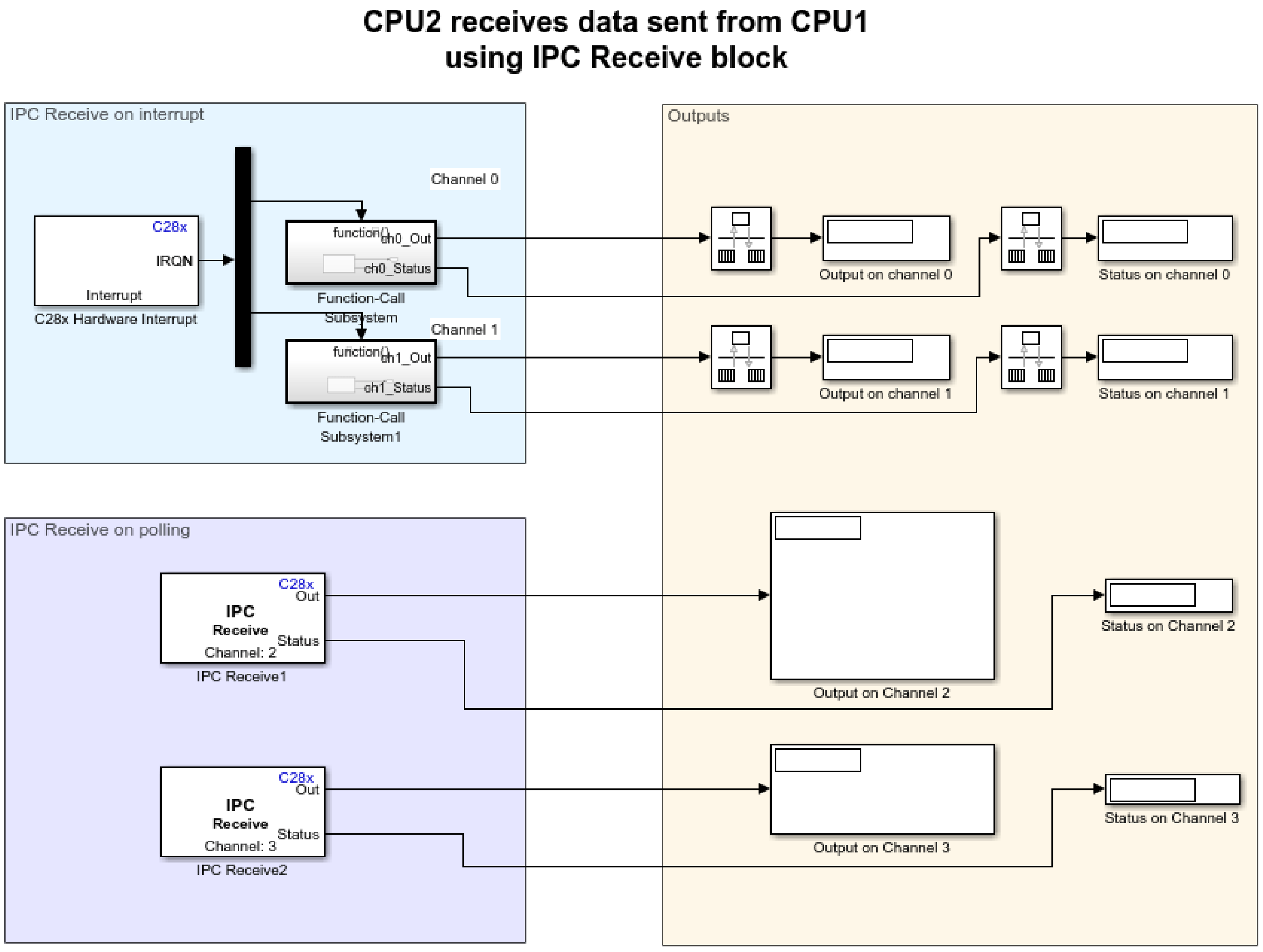
2. In the **Configuration Parameters** window, click **Hardware Implementation** and navigate to **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at **Device Manager** > **Ports** (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301.

3. Open **Hardware** tab and Click **Monitor & Tune**.

4. Observe the output data using display blocks. The data received in CPU2 must match the data transmitted from CPU1.

Note: The channel number of the IPC Receive block on the reading CPU must match the channel number of the IPC Transmit block on the writing CPU.

```
open_system('c2837xd_ipc_cpu2_rx');
```



Memory Details

For inter-processor communication:

- For scalar data transfers, message RAMs will be used. The current memory settings in the linker file are:

CPU2TOCPU1RAM: origin = 0x03F800, length = 0x000400

CPU1TOCPU2RAM : origin = 0x03FC00, length = 0x000400

- For vector data transfers, the two blocks of global shared RAM (2x16K) are used. The current memory settings in the linker file are:

RAMGS_IPCBuffCPU1 : origin = 0x00C000, length = 0x001000

RAMGS_IPCBuffCPU2 : origin = 0x00D000, length = 0x001000

You can access the linker file by browsing to **Configuration Parameters > Hardware Implementation > Target hardware resources > Build Options**.

For larger data transfers using IPC blocks, the memory must be increased using the linker file. To increase the IPC memory size, allocate more memory for RAMGS_IPCBuffCPU1 and RAMGS_IPCBuffCPU2. The size of the memory must be the same in both cases.

For dual motor control using IPC blocks, see “Control PMSM Loaded with Dual Motor (Dyno) Using C2000 Processors” on page 4-178. In this example, c28379Dpmsmfocdual_cpu1_ert.slx and c28379Dpmsmfocdual_cpu2_ert.slx models communicate using IPC.

Task 2 - Communicate between CPU1, CPU2 on TI F2838x and ARM Cortex-M4 using IPC blocks

This task of an example is configured with Texas Instruments™ F2838x ControlCard. F2838x is a Multi-core processor. You can flash three different models for three cores.

- `CPU1 model`
- `CPU2 model`
- `ARM Cortex-M4 model`

This example uses four different channels to transmit and receive data between cores.

- A Free-Running counter data is transmitted from CPU1 to CPU2. Data is transmitted at four different sample rates through four channels (0, 1, 2, 3)
- The data is received at CPU2 using IPC receive blocks. The channel numbers and sample time are configured to match the channel numbers and sample time used in IPC Transmit blocks in CPU1. The received data is then transferred from CPU2 to ARM Cortex - M4.
- The data is received at ARM Cortex - M4 using IPC receive blocks. The channel numbers and sample time are configured to match the channel numbers and sample time used in IPC Transmit blocks in CPU2. The received data is then transferred from ARM Cortex - M4 to CPU1.
- Run the CPU1 model in External mode and compare the data sent to CPU2 with the data received from ARM Cortex - M4.

In order to run the example. Download the CPU2 model for TI F2838x (C28x) and ARM Cortex -M4 model (TI F2838x (ARM Cortex - M4)) and then run CPU1 model for TI F2838x (C28x) in External mode.

CPU2 model for TI F2838x

Open the `CPU2 model`, and click **Build, Deploy & Start** under **Hardware** tab or press **Ctrl+B** to build and download the executable file on CPU2.

ARM Cortex - M4 model for ARM Cortex - M4

Open the `ARM Cortex-M4 model`, and click **Build, Deploy & Start** under **Hardware** tab or press **Ctrl+B** to build and download the executable file on ARM Cortex - M4.

CPU1 model for TI F2838x

1. Open `CPU1 model`.

2. In the **Configuration Parameters** window, click **Hardware Implementation** and navigate to **Target hardware resources > External mode** and set the **Serial port** parameter to the COM port at **Device Manager > Ports (COM &LTP)** in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301.

3. Open **Hardware** tab and click **Monitor & Tune**.

4. Observe the output data using Display blocks. The data received from ARM Cortex - M4 will be delayed value when compared with the data transmitted to CPU2.

Memory Details

For inter-processor communication between CPU1 and CPU2,

- For scalar data transfers, message RAMs will be used. The current memory settings in the linker file are:

CPU1TOCPU2RAM: origin = 0x03A000, length = 0x000800

CPU2TOCPU1RAM: origin = 0x03B000, length = 0x000800

- For vector data transfers, the two blocks of the global shared RAM (2x16k) are used. The current memory settings in the linker file are:

RAMGS_IPCBuffCPU1 : origin = 0x00D000, length = 0x001000

RAMGS_IPCBuffCPU2 : origin = 0x00E000, length = 0x001000

For larger data transfers using IPC blocks, the memory must be increased using the linker file. You can access the linker file by browsing to **Configuration Parameters > Hardware Implementation > Target hardware resources > Build Options**. To increase the IPC memory size, allocate more memory for RAMGS_IPCBuffCPU1 and RAMGS_IPCBuffCPU2. The size of the memory must be the same in both cases.

For inter-processor communication between CPU1/CPU2 and ARM Cortex - M4, as global shared RAM is not available, IPC message RAM is used. The amount of data transfer between CPU1/CPU2 and ARM Cortex - M4 is limited by the size of the message RAMs available.

The current message RAM settings in the linker file are:

CPUTOCMRAM : origin = 0x039000, length = 0x000800

CMTOCPURAM : origin = 0x038000, length = 0x000800

For information about IPC blocks, see:

- F2837xD/F2838x/F2838x-M4 IPC Receive
- F2837xD/F2838x/F2838x-M4 IPC Transmit

Using SPI to Read and Write Data to SPI EEPROM

This example shows how to configure and use SPI blocks to read and write data.

In this example, you will learn the following tasks:

- SPI loopback using the Controller Transfer block
- Read and write data to SPI EEPROM using the Controller Transfer block
- SPI loopback using the SPI Transmit block, the SPI Receive block, and interrupts
- Read and write data to SPI EEPROM using the SPI Transmit block, the SPI Receive block, and interrupts

Required Hardware

- Texas Instruments™ C2000 ControlCARD or LaunchPad
- Texas Instruments Peripheral Explorer or CAT25256 256kB SPI EEPROM Memory. This device uses a standard SPI protocol that is common to many other EEPROMs provided by different vendors. For more details about the device, refer to the CAT25256 data sheet.

Hardware Connections

The SPI EEPROM uses the following 8-bit opcodes for enable, write data, read data, and read status.

```
      Command  | Opcode | Operation
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
       WREN    | 6      | Enable Write Operations
       WRITE   | 2      | Write Data to Memory
       READ    | 3      | Read Data from Memory
       RDSR    | 5      | Read Status Register
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

The hardware connections made are dependent on the EEPROM usage. You can use the SPI EEPROM provided in the TI Peripheral Explorer or a separate EEPROM chip, such as CAT25256 256kB SPI EEPROM.

Texas Instruments Peripheral Explorer

- If the ControlCARD is inserted in the slot provided by the Peripheral Explorer, no additional connections are required.
- If the ControlCARD is inserted in a docking station or a LaunchPad is used, SPI connections must be made between the Peripheral Explorer and the docking station or LaunchPad as listed.

```
    Peripheral Explorer |  C2000
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
      ECAP-1  / SPISIMO |  SPISIMO
      ECAP-2  / SPISOMI |  SPISOMI
      ECAP-3  /  SPICLK |  SPICLK
      EPWM-6B /  SPISTE |  SPISTE
      GND               |  GND
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

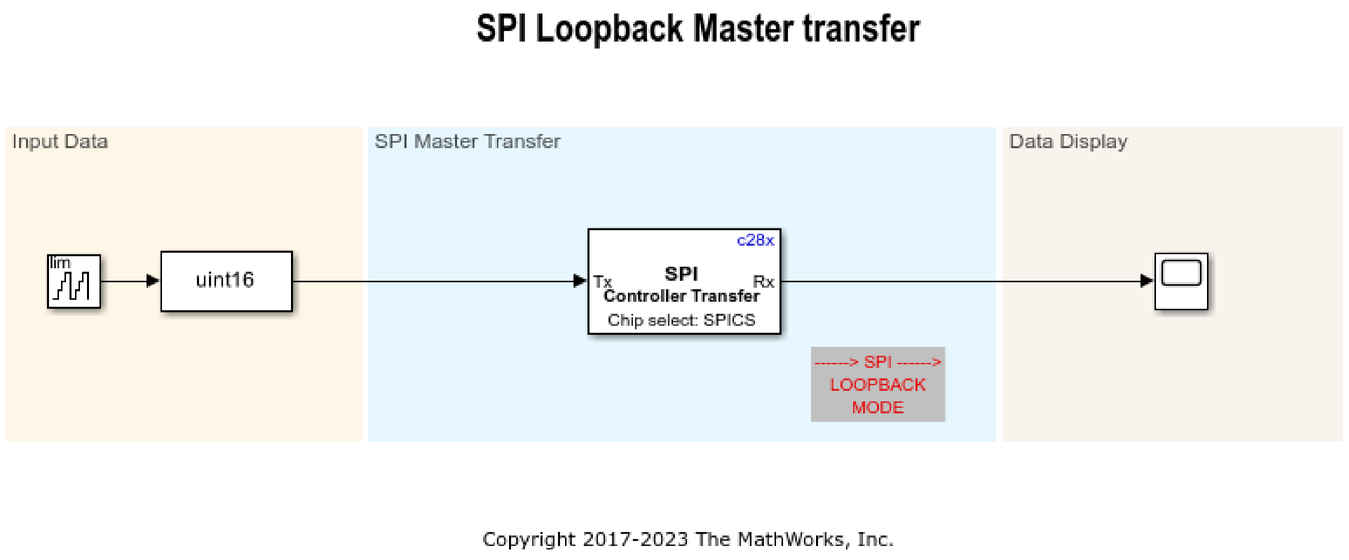
CAT25256 256kB SPI EEPROM Memory

Connect the chip to the C2000 LaunchPad or ControlCARD as listed.

```
     SPI EEPROM pin |  C2000
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
      /CS   (pin 1) |  SPISTE
      SO    (pin 2) |  SPIMISO
      /WP   (pin 3) |  3.3 V
      VSS   (pin 4) |  GND
      SI    (pin 5) |  SPIMOSI
      SCK   (pin 6) |  SPICLK
      /HOLD (pin 7) |  3.3 V
      VCC   (pin 8) |  3.3 V
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

SPI Loopback Using the Controller Transfer Block

The Counter Limited block sends the counter value to the SPI Controller Transfer block as input, and the output data received is sent to the scope. You can observe the counter value in the Scope block while running the model in External mode.



To run the model:

1 Open the c28x_spitest_ert model. This model is configured for the TI F28379D LaunchPad hardware board. To configure the model to run on other TI C2000 processors, change the **Hardware board** parameter in the **Configuration Parameters > Hardware Implementation** pane.

2 Browse to **Hardware Implementation > Target Hardware Resources > SPI_A**, and select **Enable loopback**. Alternatively, you can connect **SIMO** and **SOMI** physically using GPIO pins to enable external loopback.

3 Select appropriate values for **SIMO**, **SOMI**, **CLK**, and **STE pin assignment**, depending on the processor.

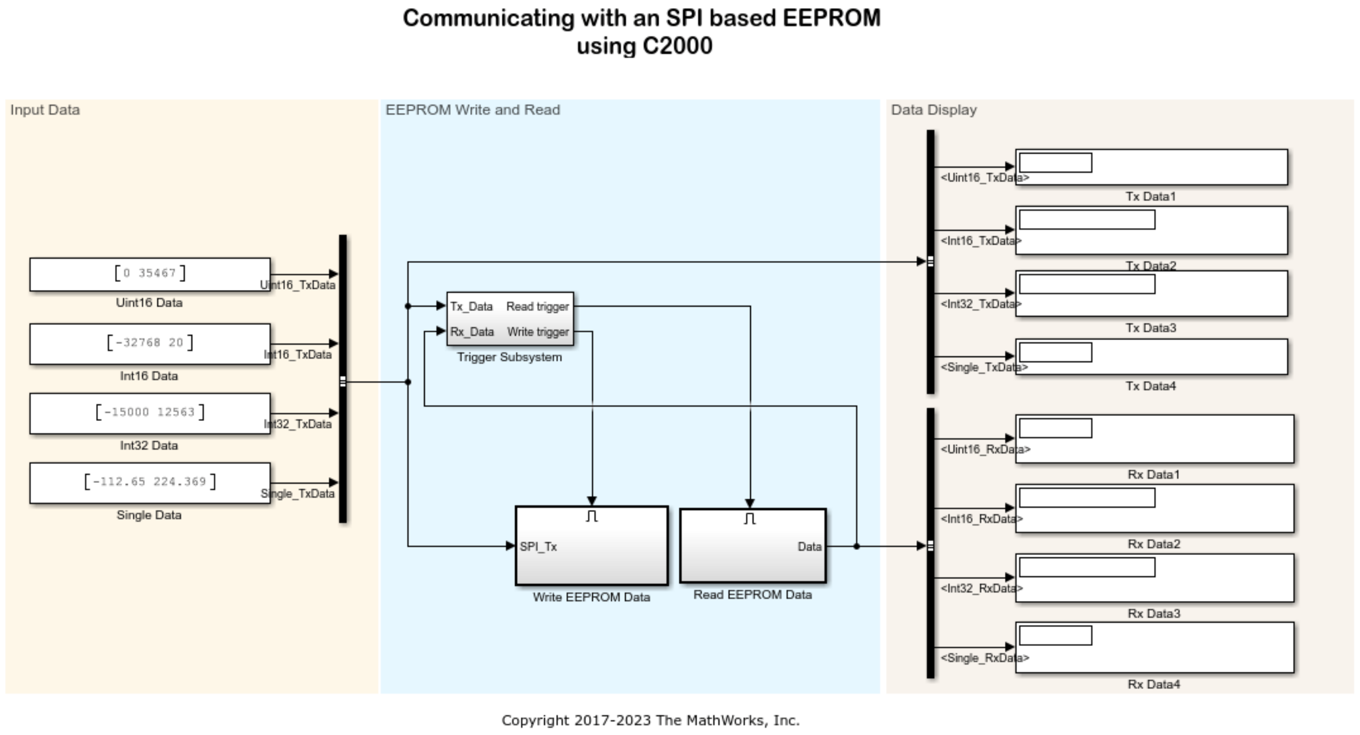
4 In the **Configuration Parameters** window, click **Hardware Implementation** > **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at

Device Manager > **Ports** (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301.

5 Open **Hardware** tab and click **Monitor & Tune**. Observe the counter values on the scope.

Read and Write Data to SPI EEPROM Using the Controller Transfer Block

The model writes data values of various types to the EEPROM and reads back the data from the corresponding EEPROM address to show successful communication.



SPI blocks are configured with the **Data bits** parameter set to 8 to send the 8-bit opcodes and write/read the 8-bit data. If you select the STE pin provided by the SPI peripheral in **Configuration Parameters**, the peripheral is deselected between data transfers. In this case, the `Explicit GPIO calls` option is used to select **Chip select (provided by SPI module) assignment** as GPIO1 to ensure that the peripheral is selected continuously for multiple data transfers.

The model consists of a Trigger Subsystem, Write EEPROM Data and Read EEPROM Data subsystems, and Display blocks. The Read EEPROM Data subsystem is always triggered, while the Write EEPROM Data subsystem is triggered only if the data read from the EEPROM does not match the input data to be written to the EEPROM.

The Write EEPROM Data subsystem performs EEPROM write enable operation before every write cycle. The input data of different types are packed in 8-bit packets using the Byte Pack block, and then the data is converted to uint16. The data is then written to the EEPROM at the address location 0x0020 using the Controller Transfer block. After this, the program waits until the EEPROM write operation is complete by monitoring the status flag of the EEPROM.

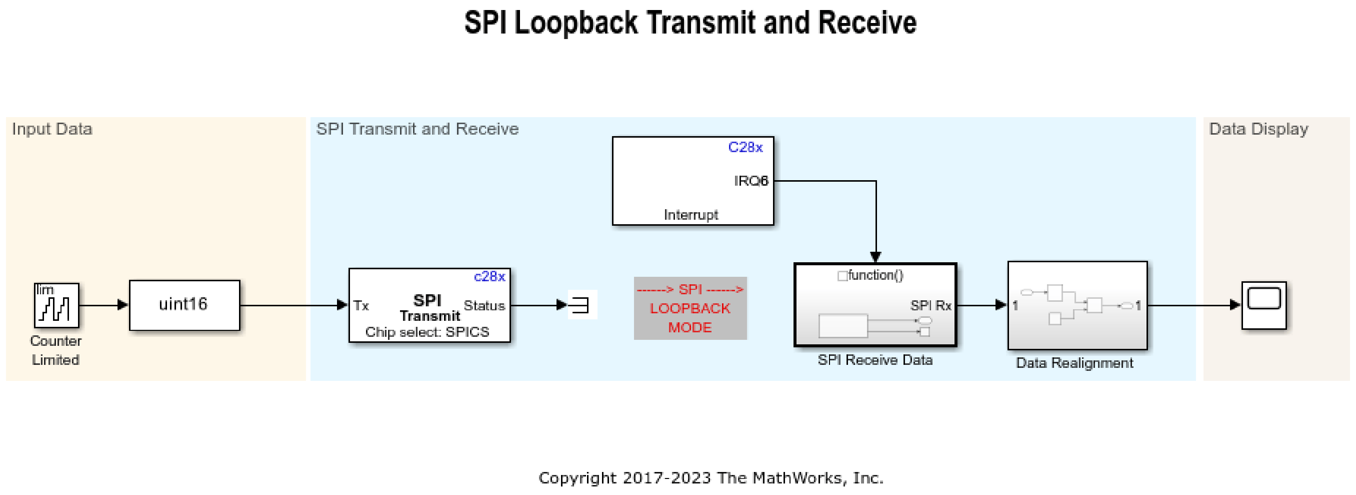
The Read EEPROM Data subsystem reads data back from the memory location 0x0020, and the 8-bit packet data is unpacked to data of required type using the Byte Unpack block. This data is then shown using Display blocks.

To run the model:

1. Open the c28x_spi_eeprom model. This model is configured for the **TI F28379D LaunchPad** hardware board. To configure the model to run on other TI C2000 processors, change the **Hardware board** parameter in the **Configuration Parameters > Hardware Implementation** pane.
2. Browse to **Hardware Implementation > Target Hardware Resources > SPI_A**.
3. Enter the value for **Desired baud rate in bits/sec** as `2000000`, and set **STE pin assignment** to `None`.
4. In the **Configuration Parameters** window, click **Hardware Implementation** and go to **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at **Device Manager** > **Ports** (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301.
5. Open **Hardware** tab and click **Monitor & Tune**. Observe output data in display blocks.
6. Change the input data values and observe the changes.

SPI Loopback Using the SPI Transmit Block, the SPI Receive Block, and Interrupts

The Counter Limited block sends the counter value to the SPI Transmit block as input. The receive FIFO is configured to trigger the interrupt for a FIFO length of 4. The output data of length 4 received from the SPI Receive block is realigned as a single stream and sent to the scope. You can observe the counter value in the Scope block while running in external mode.



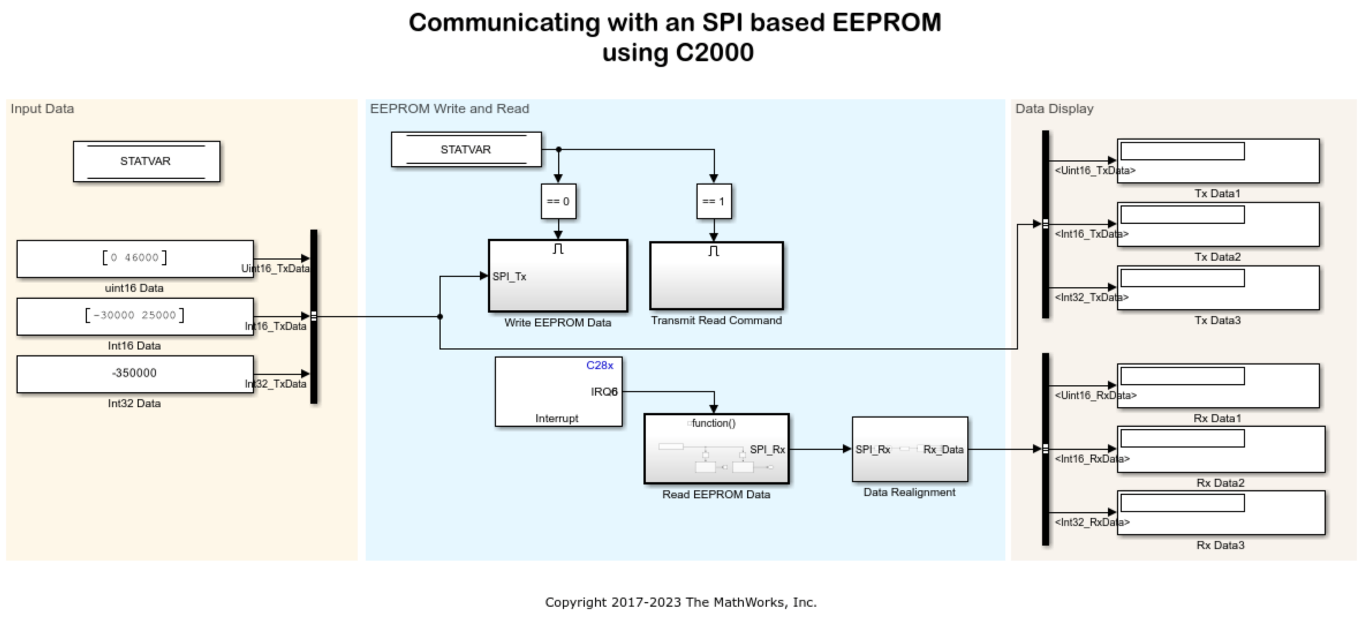
To run the model:

1. Open the c28x_spi_interrupt_test_ert model. This model is configured for the TI F28379D LaunchPad hardware board. To configure the model to run on other TI C2000 processors, you can change the **Hardware board** parameter in the **Configuration Parameters > Hardware Implementation** pane.
2. Browse to **Hardware Implementation > Target Hardware Resources > SPI_A**, and select **Enable loopback**. Alternatively, connect **SIMO** and **SOMI** physically using GPIO pins to enable external loopback.
3. Select appropriate values for **SIMO**, **SOMI**, **CLK**, and **STE pin assignment**, depending on the processor.

4 Select **Enable Rx interrupt**, and set **FIFO interrupt level(Rx)** to 4.
5 In the **Configuration Parameters** window, click **Hardware Implementation** and navigate to **Target hardware resources > External mode** and set the **Serial port** parameter to the COM port at **Device Manager > Ports** (COM & LTP) in Windows. For more information, see "Parameter Tuning and Signal Logging with Serial Communication" on page 4-301.
6 Open **Hardware** tab and click **Monitor & Tune**. Observe the counter values on the scope.

Read and Write Data to SPI EEPROM Using the SPI Transmit Block, the SPI Receive Block, and Interrupts

The model writes data values of various types to EEPROM and reads back data from the corresponding EEPROM address to show successful communication. This model can be used only for hardware boards with an SPI FIFO length of 15.



SPI blocks are configured with the **Data bits** parameter set to 8 to send the 8-bit opcodes and write/read the 8-bit data. If you select the STE pin provided by the SPI peripheral in **Configuration Parameters**, the peripheral is deselected between data transfers. In this case, the `Explicit GPIO calls` option is used to set **Chip select (provided by SPI module) assignment** to GPIO1 to ensure that the peripheral is selected continuously for multiple data transfers. The receive FIFO is configured to trigger the interrupt for a FIFO length of 15.

The model consists of Write EEPROM Data, Transmit Read Command, Read EEPROM Data, and Data Realignment subsystems along with Display blocks. Write and read data operations are triggered alternatively using the STATVAR variable.

The Write EEPROM Data subsystem performs the EEPROM write enable operation using SPI Transmit and SPI Receive blocks. Before every write cycle, block priorities are set to ensure that SPI Transmit is executed first. The input data of different types are packed in 8-bit packets using the Byte Pack block and converted to uint16. Data is then written to the EEPROM at the address location 0x0020 using the SPI Transmit block. After this, the program waits until the receive interrupt is triggered. When the receive interrupt is triggered, the data from the receive FIFO is read and discarded.

The Transmit Read Command subsystem is then triggered, which sends a read command and an address with dummy data of the size of the data to be read. After this, the program waits until the receive interrupt is triggered. When the receive interrupt is triggered, data from the receive FIFO is read.

The Data Realignment subsystem performs byte reordering to create 8-bit word packets, which are unpacked to data of required type using the Byte Unpack block. This data is then shown using display blocks.

To run the model:

1. Open the c28x_spi_eeprom_interrupt model. This model is configured for the TI F28379D LaunchPad hardware board. To configure the model to run on other TI C2000 processors, change the **Hardware board** parameter in the **Configuration Parameters > Hardware Implementation** pane.
2. Browse to **Hardware Implementation > Target Hardware Resources > SPI_A**.
3. Enter the value for **Desired baud rate in bits/sec** as `2000000`, and set **STE pin assignment** to `None`.
4. Select **Enable Rx interrupt**, and set **FIFO interrupt level(Rx)** to `15`.
5. In the **Configuration Parameters** window, click **Hardware Implementation** and navigate to **Target hardware resources** > **External mode** and set the **Serial port** parameter to the COM port at **Device Manager** > **Ports** (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301.
6. Open **Hardware** tab and click **Monitor & Tune**. Observe the output data using Display blocks.
7. Change the input data values and observe the changes.

More About

- C28x SPI Controller Transfer
- C28x SPI Receive
- C28x SPI Transmit

Modify Duty Cycle of ePWM Using DMA

This example shows how to configure the direct memory access (DMA) parameters to modify the ePWM duty cycle. Using DMA, the sine wave data is copied from a look-up table to the ePWM compare register.

You can observe the duty cycle changes by connecting the ePWM pin (GPIO2) to:

- An oscilloscope to monitor the changes in the duty cycle.
- The LED (GPIO31/GPIO34) to observe the dimming of the LED.

The example consists of a model and a callback script. The script runs when the model is initialized. The callback script (`sineTableCalculation.m`) generates a sine wave of 500 samples, and then scales the sine wave to the range of zero to the value of the ePWM period register.

Required Hardware

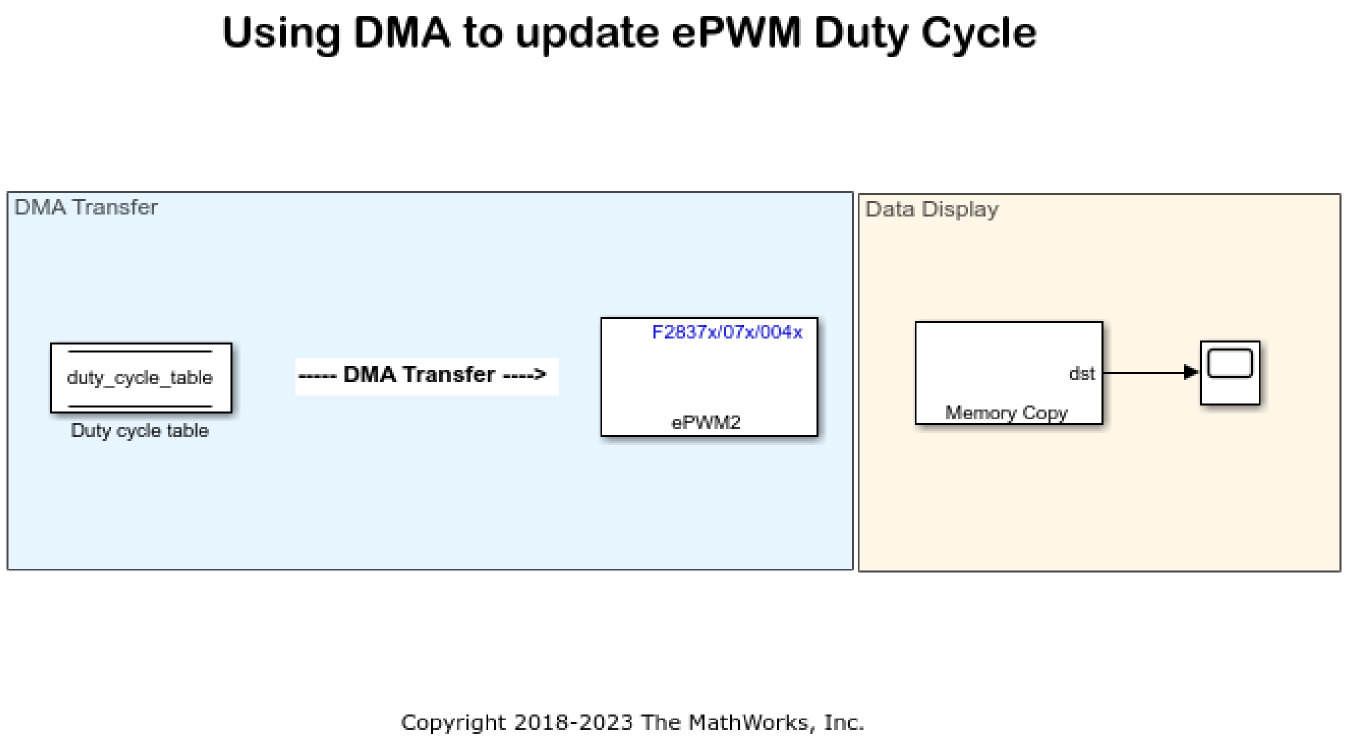
F2833x, F2806x, F2807x, F2837x, or F28004x controlCARD/LaunchPad.

Available Models

- F2833x: c2833x_dma_epwm.slx
- F2806x: c2806x_dma_epwm.slx
- F2807x, F2837x, and F28004x: c28x7x_c28004x_dma_epwm.slx

Model

The c28x7x_c28004x_dma_epwm model consists of the `duty_cycle_table` look-up table defined using the Data Store Memory, ePWM, and Memory Copy blocks. The `duty_cycle_table` look-up table stores the sine wave samples generated by the callback script.



The ePWM block is configured in up-down mode for a period of 0.002 seconds. For more information, see “General”.

The ePWM block is also configured to generate a start of conversion event for module A (SOCA). For more information, see “Event Trigger”.

The DMA parameters are configured to transfer 500 sine wave samples from the `duty_cycle_table` look-up table to the ePWM compare register. DMA transfers one sample at a time when triggered by the ePWM2SOCA event. To configure the DMA parameters, browse to **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources** > **DMA_ch1**.

The Memory Copy block provides the value of the ePWM compare register to the scope.

Configure and Run the Model

1. In the Configuration Parameters window, click **Hardware Implementation** > **Target hardware resources**.

2. Click **SCI_A**, and set the **Desired baud rate in bits/sec** parameter to `1.25e6`.

3. Click **External mode**, and set the **Serial port** parameter to the COM port at Device Manager > Ports (COM & LTP) in Windows. For more information, see “Parameter Tuning and Signal Logging with Serial Communication” on page 4-301.

4. To ensure that enough memory is present to run the model in External mode, click **Code Generation** > **Optimization** in the Configuration Parameters window.

5. Set the **Default parameter behavior** parameter to `Inlined`, and click **OK**.

6. Open **Hardware** tab and click **Monitor & Tune**. Observe the sine wave values on the scope.

7. You can connect the ePWM pin (GPIO2) to:

- An oscilloscope to monitor the changes in the duty cycle
- The LED (GPIO31/GPIO34) to observe the dimming of the LED

More About

c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM

Using Sigma Delta Filter Module (SDFM) to Measure the Analog Input Signal

This example shows how to use the Sigma Delta Filter Module (SDFM) to measure the analog input signal for Texas Instruments™ C2000™ Microcontroller Blockset. Using this example, you can:

- Configure the SDFM module to receive the digital bit stream from an external sigma delta modulator (SDM)
- Display the filtered digital data

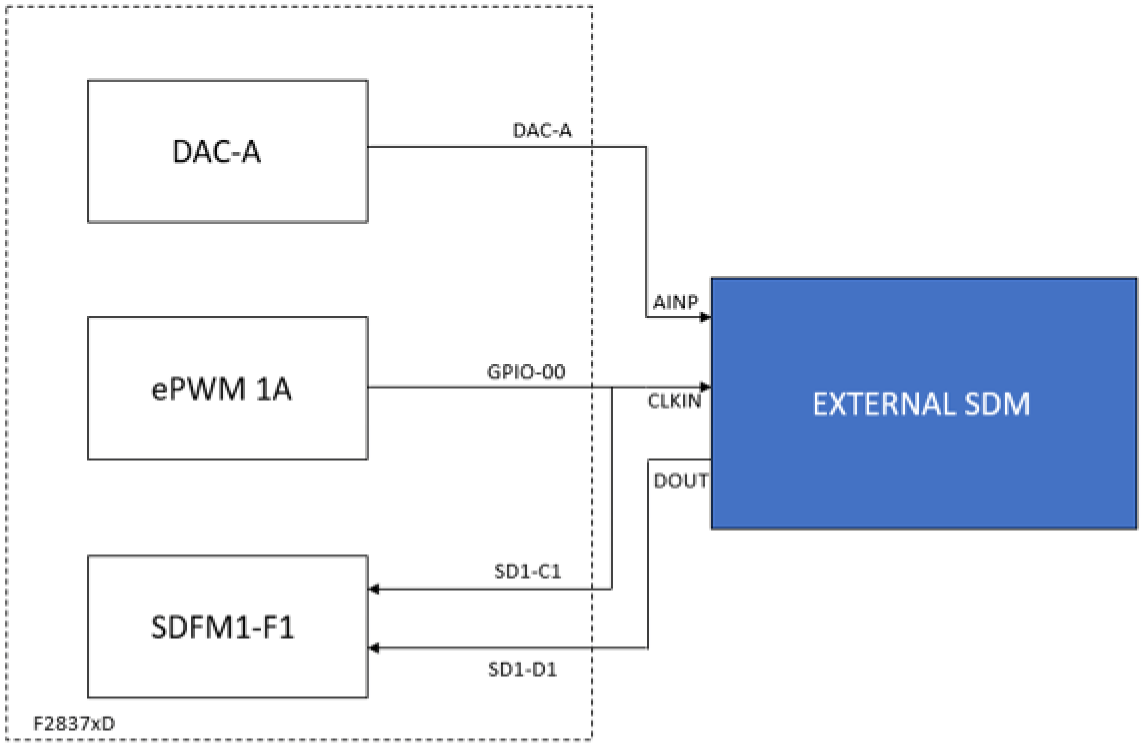
Introduction

This example uses **ePWM** and **DAC** blocks to generate clock and data signals which are provided as input to the external sigma delta modulator. The digital pulse stream from the sigma delta modulator is given as data input to the SDFM in the processor. The data filter output (DFLTx) is the digital 16-bit or 32-bit representation of the analog signal output from the DAC.

Required Hardware

- F2807x or F2837x controlCARD/LaunchPad.
- AMC1304EVM external Sigma Delta Modulator(SDM).

Hardware Connections



1. The output of the ePWM1A is provided as the clock input to the external SDM and the output of the DAC is provided as the analog data input to the external SDM.

2. The ePWM1A output is also provided as clock input to the SDFM module in the processor.

3. The digital data stream from the external sigma delta modulator is provided as the data input to the SDFM module.

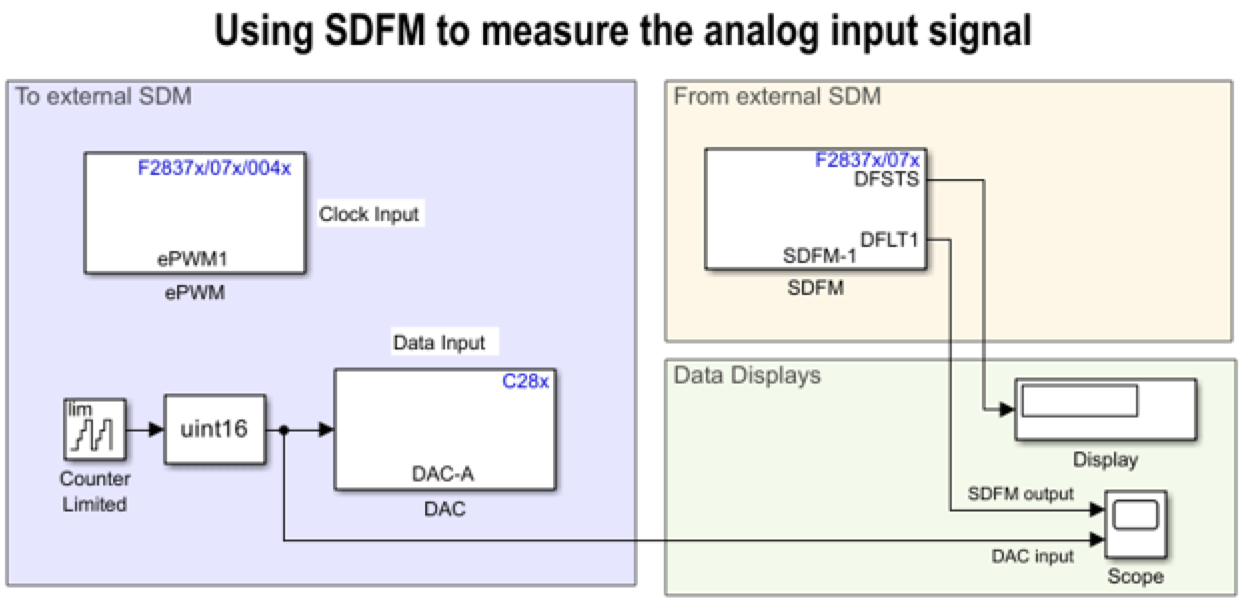
Connect the external SDM to the C2000 LaunchPad or ControlCARD as listed

SDM (AMC1304EVM)	C2000 Launchpad
AINP	DAC-A
AINN	GND
VDD1	3.3 V
AGND	GND
DGND	GND
DVDD	3.3 V
CLKIN	EPWM1A (GPIO-00)
CLKIN	SD1-C1 (GPIO-17)
DOUT	SD1-D1 (GPIO-16)

For the SDM AMC1304EVM, one must set jumper JP1 to the position labeled **Ext**.

Model

```
open_system('c28x7x_sdfm.slx');
```



Copyright 2020-2022 The MathWorks, Inc.

The ePWM1 is configured to give clock signal of 50% duty cycle and frequency in the range as required by external sigma delta modulator (SDM).

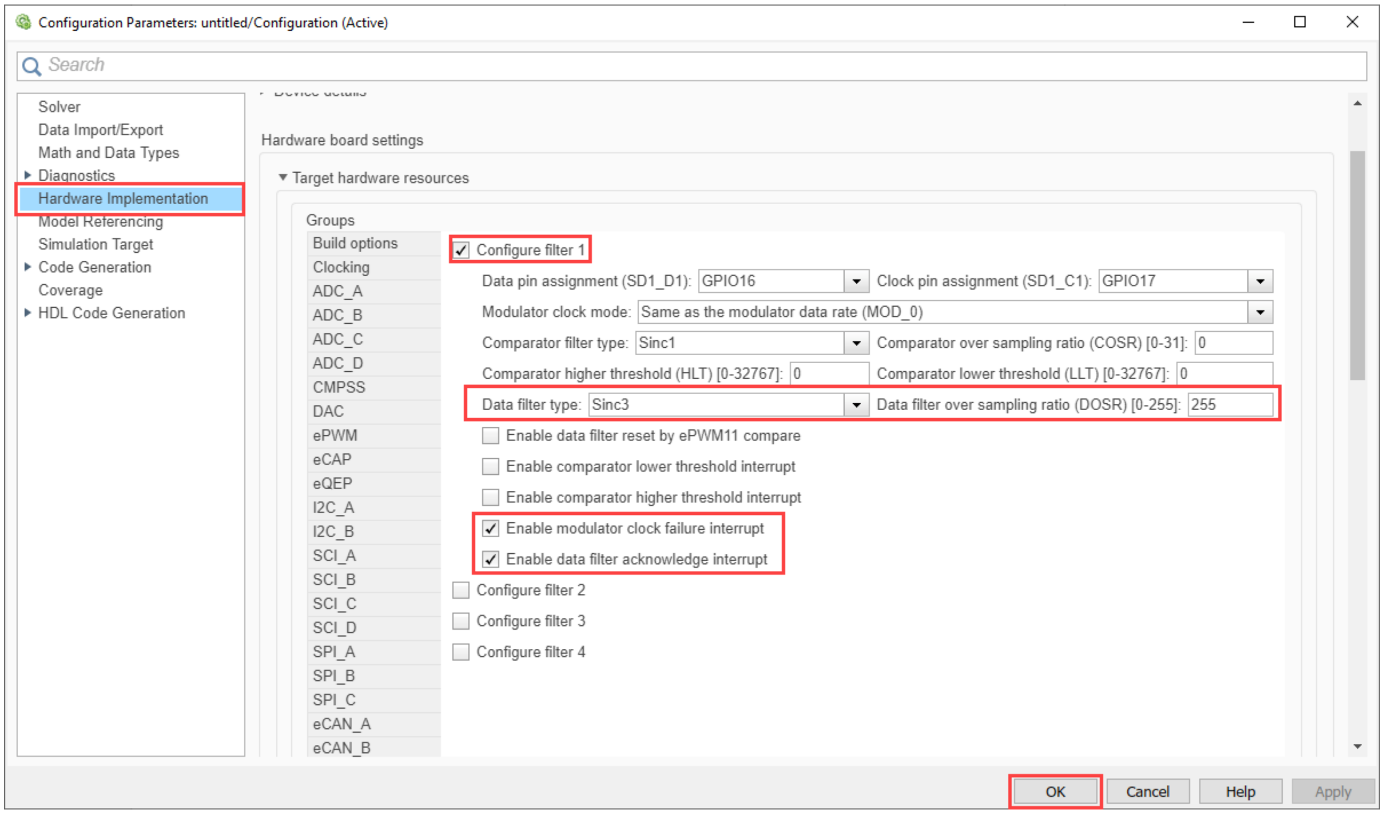
The input to DAC is varied from 1 to 300, and this will ensure that the analog output of DAC is varied in the input range as required by the SDM. The status of the data filter is viewed in the **Display** block. The varying input to the DAC and the digital data output from the SDFM can be viewed in the **Scope** block.

Configure and Run the Model

1. Open the SDFM example model. Go to the **Modeling** tab and press **Ctrl+E** to open the Configuration Parameters dialog box.

2. In the Configuration Parameters window, click **Hardware Implementation > Hardware board** and select the required hardware board.

3. Browse to **Target Hardware Resources > SDFM1** and select **Configure filter 1**. The digital filter settings for the filter channel 1 of SdfmReg1 is updated as shown below. Filter channel 1 of the SdfmReg1 with 16-bit data representation is configured for this example.


Configuration Parameters: untitled/Configuration (Active)

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation

Hardware board settings

Target hardware resources

Groups: Build options, Clocking, ADC_A, ADC_B, ADC_C, ADC_D, CMPSS, DAC, ePWM, eCAP, eQEP, I2C_A, I2C_B, SCI_A, SCI_B, SCI_C, SCI_D, SPI_A, SPI_B, SPI_C, eCAN_A, eCAN_B

☑ Configure filter 1

Data pin assignment (SD1_D1): GPIO16
Clock pin assignment (SD1_C1): GPIO17
Modulator clock mode: Same as the modulator data rate (MOD_0)
Comparator filter type: Sinc1
Comparator over sampling ratio (COSR) [0-31]: 0
Comparator higher threshold (HLT) [0-32767]: 0
Comparator lower threshold (LLT) [0-32767]: 0
Data filter type: Sinc3
Data filter over sampling ratio (DOSR) [0-255]: 255

☐ Enable data filter reset by ePWM11 compare
☐ Enable comparator lower threshold interrupt
☐ Enable comparator higher threshold interrupt
☑ Enable modulator clock failure interrupt
☑ Enable data filter acknowledge interrupt

☐ Configure filter 2
☐ Configure filter 3
☐ Configure filter 4

4. Ensure that Communication interface is set to **Serial**. The selection can be made at **Configuration Parameters > Hardware Implementation > Target hardware resources > External Mode > Communication interface**.

5. On the host computer, set the **Serial** port parameter to the **COM** port at **Device Manager > Ports (COM & LTP)** in Windows.

6. Click **Apply** and **OK**.

7. Open **Hardware** tab and click **Monitor & Tune**. Observe the input to DAC and the output of SDFM on the **Scope** block when the **DFSTS** output is 1.

Interpretation of DAC Analog Voltage Output and SDFM Digital Output

This section explains how to interpret the DAC output analog voltage and how to calculate the expected SDFM digital output. For example,

- Analog output voltage of the DAC = 190mV
- Input voltage range of the SDM = -250mV to 250mV
- Ratio of input voltage to the max voltage range of SDM = (190-(-250))/500 = 440/500 = 0.88 or 88% The output pulse stream of 1s and 0s from SDM will contain 88% 1s.

For the below SDFM configuration, the output data range is (-32,768 to 32,767).

- SDFM data filter type = Sinc3
- Data filter Over Sampling Ratio = 256
- Data representation = 16-bit

Since the input to SDFM contains a pulse stream with 88% 1s, the expected output (x) can be calculated as

(x-(-32768))/(32768*2) = 0.88

The expected SDFM output = 24,903 The actual SDFM output = 2.52 * 10^04 One can expect an error of around 1% or less in the digital output due to variations at the DAC output due to operations at such low voltages and due to the connections used.

Other things to try

You can configure the SDFM comparator filter in the configuration parameters, **Target Hardware Resources > SDFM1** and check the status of the comparator filter flags by enabling the **Enable comparator output** in the block parameter.

More About

F2807x/F2837xD/F2837xS/F28004x/F28003x/F2838x SDFM

CAN Communication Using eCAN Blocks

This example shows how to use the eCAN blocks to set up CAN communication between the target hardware and your host computer for Texas Instruments™ C2000™ Microcontroller Blockset.

Introduction

Using this example, you will:

- Learn how to establish a communication between target hardware and the host computer using eCAN Transmit and Receive blocks on the target hardware and Vehicle Network Toolbox™ (VNT) CAN Transmit and Receive blocks on the host.
- Demonstrate using acceptance filtering to receive specific messages with standard and extended IDs on target hardware.
- Demonstrate the use of the polling and interrupt methods to receive data on target hardware.

Prerequisites

Complete the following tutorials:

- “Set Up CAN Communication with Target Hardware” on page 1-8
- “Getting Started with Texas Instruments C2000 Microcontroller Blockset” on page 4-2

Required Hardware:

- Any Texas Instruments™ C2000™ board with CAN module
- Vector CAN hardware

Note: If you are trying to run this example on a Launchpad, connect the TX and RX pin to the vector CAN device. Incase of Control card, a Transceiver (TCAN EVM transceiver) is required between vector CAN device and Control card.

Task 1 - CAN Communication between Host Computer and Target Hardware

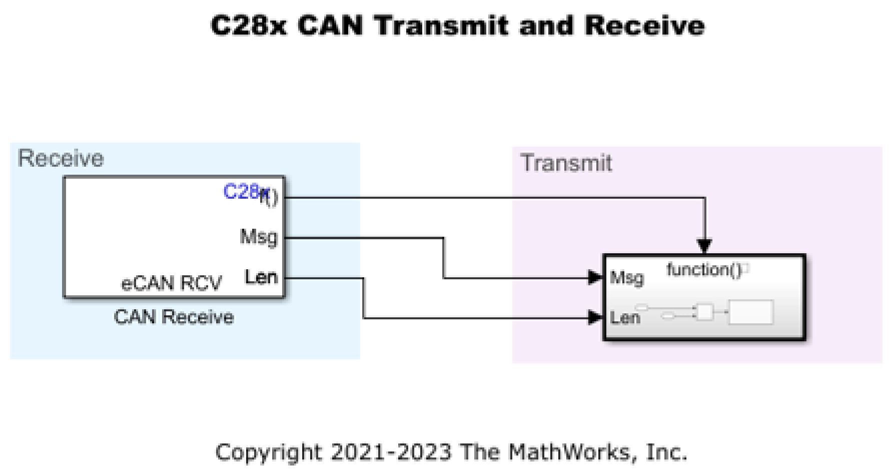
In this task, you will transmit counter data from the host to the target hardware in two parts.

- When the target hardware receives a new message from host, it transmits that data with a different message ID back to the host.
- The host then displays the transmitted and received CAN signals.

Target Model

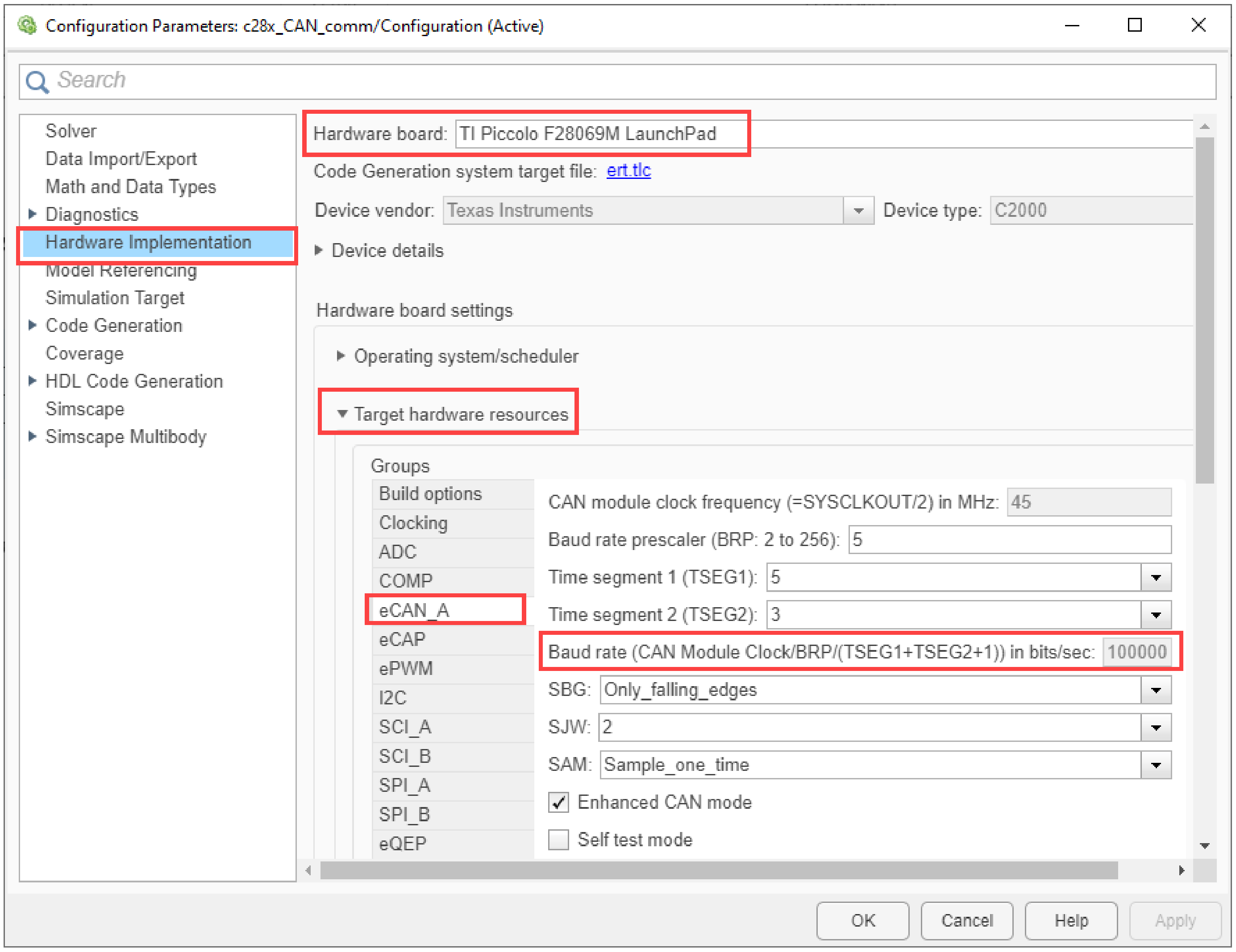
1. Open the target model.

```
open_system('c28x_CAN_comm.slx');
```


C28x CAN Transmit and Receive
Receive
C28x
f()
Msg
Len
eCAN RCV
CAN Receive
Transmit
Msg
Len
function()
Copyright 2021-2023 The MathWorks, Inc.

2. The example model is configured for the `TI Piccolo F28069M launchpad`. To select a different target hardware, in the Simulink® Editor, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**.

3. Navigate to **Hardware Implementation** > **Target Hardware Resources** > **eCAN_A** and ensure that you set the **Baud rate** to `1Mbps`.

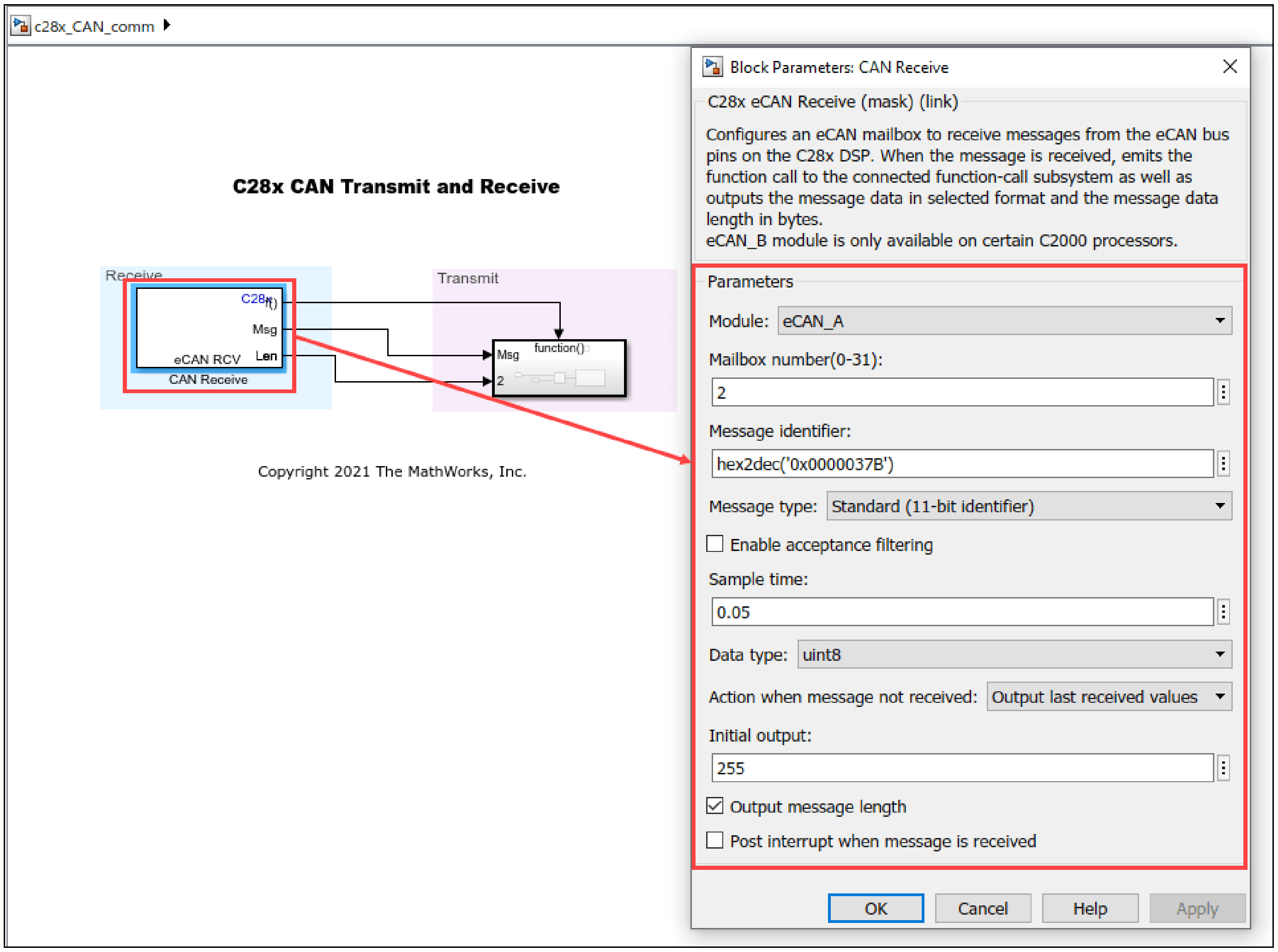


Hardware board: TI Piccolo F28069M LaunchPad
Code Generation system target file: ert.tlc
Device vendor: Texas Instruments
Device type: C2000

CAN module clock frequency (=SYSCLKOUT/2) in MHz: 45
Baud rate prescaler (BRP: 2 to 256): 5
Time segment 1 (TSEG1): 5
Time segment 2 (TSEG2): 3
Baud rate (CAN Module Clock/BRP/(TSEG1+TSEG2+1)) in bits/sec: 100000
SBG: Only_falling_edges
SJW: 2
SAM: Sample_one_time
✓ Enhanced CAN mode
☐ Self test mode

4. The model is configured to receive the data at sample time of `0.05s` and to transmit the same data with a different message ID once it receives the message.

5. Following are the **eCAN** block configurations done for target model. Double-click on the blocks to open block parameter configurations. Ensure the specified parameter values are the same if you want to run this example for other hardware board.



Block Parameters: CAN Receive

C28x eCAN Receive (mask) (link)

Configures an eCAN mailbox to receive messages from the eCAN bus pins on the C28x DSP. When the message is received, emits the function call to the connected function-call subsystem as well as outputs the message data in selected format and the message data length in bytes.
eCAN_B module is only available on certain C2000 processors.

Parameters

Module: eCAN_A

Mailbox number(0-31): 2

Message identifier: hex2dec('0x0000037B')

Message type: Standard (11-bit identifier)

☐ Enable acceptance filtering

Sample time: 0.05

Data type: uint8

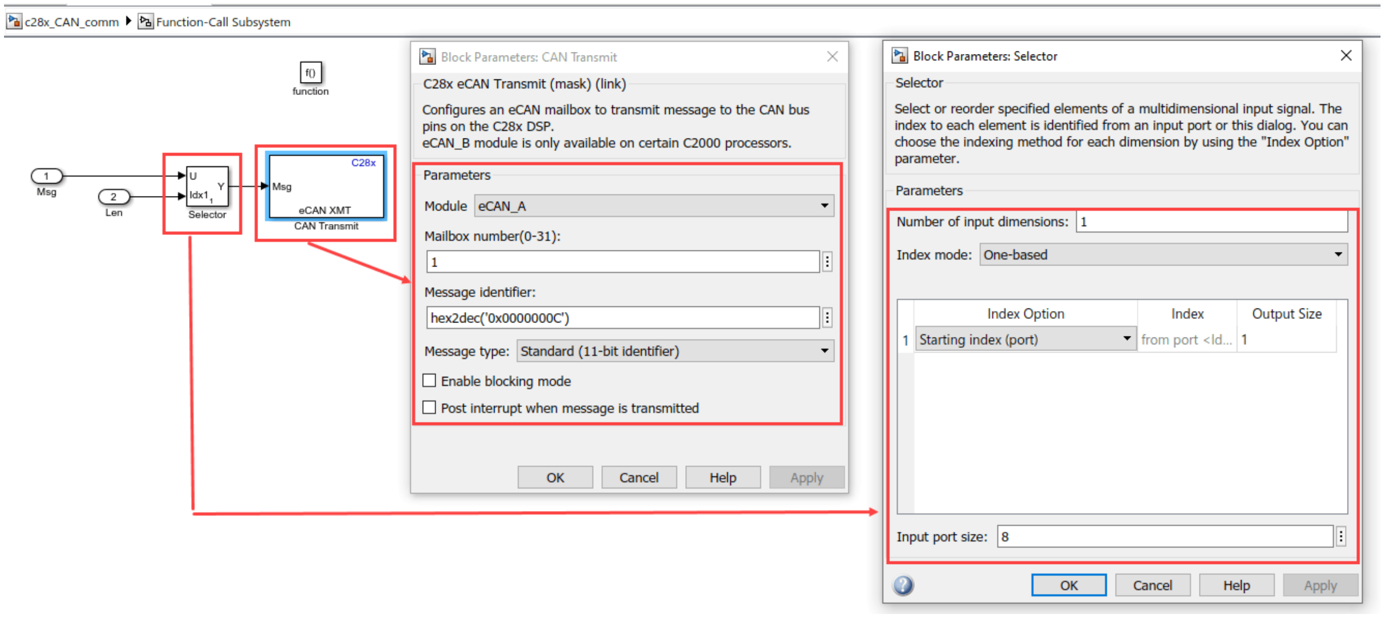
Action when message not received: Output last received values

Initial output: 255

☑ Output message length

☐ Post interrupt when message is received

- The received length is used as an input to extract specific data which is transmitted from host.



Block Parameters: CAN Transmit

C28x eCAN Transmit (mask) (link)

Configures an eCAN mailbox to transmit message to the CAN bus pins on the C28x DSP.
eCAN_B module is only available on certain C2000 processors.

Parameters

Module eCAN_A

Mailbox number(0-31):
1

Message identifier:
hex2dec('0x0000000C')

Message type: Standard (11-bit identifier)

☐ Enable blocking mode

☐ Post interrupt when message is transmitted

Block Parameters: Selector

Selector

Select or reorder specified elements of a multidimensional input signal. The index to each element is identified from an input port or this dialog. You can choose the indexing method for each dimension by using the "Index Option" parameter.

Parameters

Number of input dimensions: 1

Index mode: One-based

	Index Option	Index	Output Size
1	Starting index (port)	from port <Id...	1

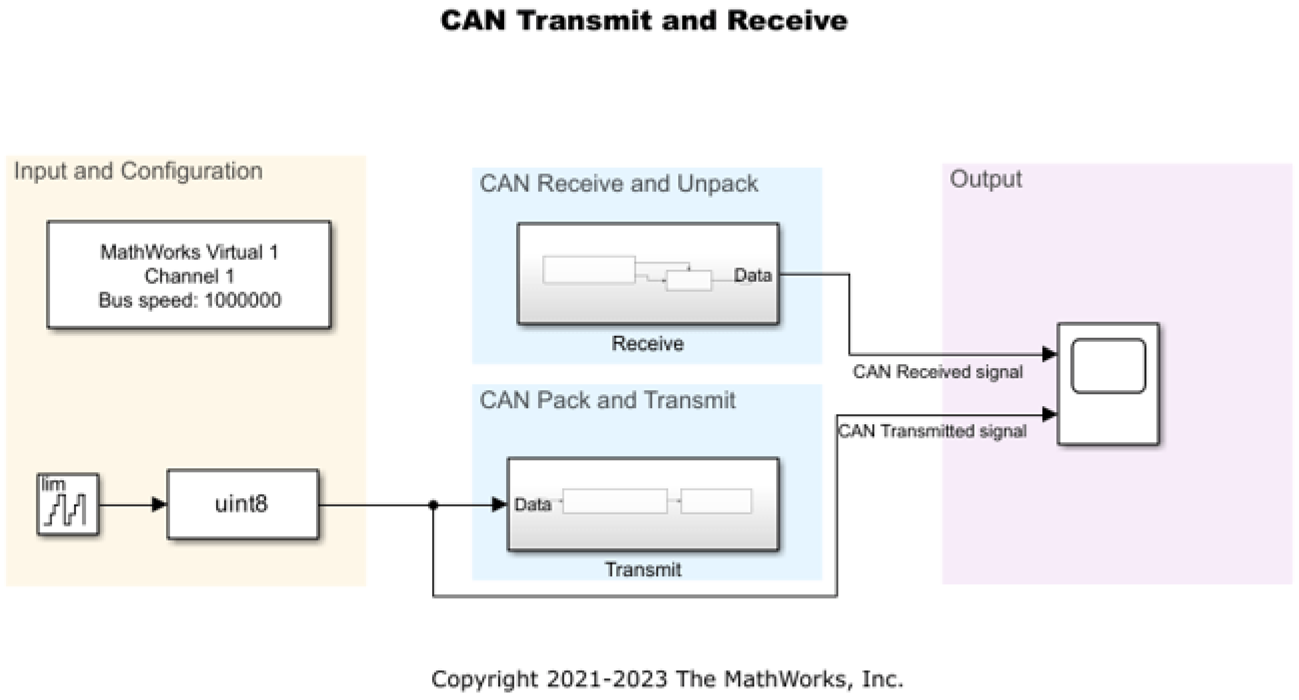
Input port size: 8

6. Click **Build, Deploy & Start** in the **Hardware** tab or press **Ctrl+B** to build and download the executable file.

Run Host Model

1. Open the host model.

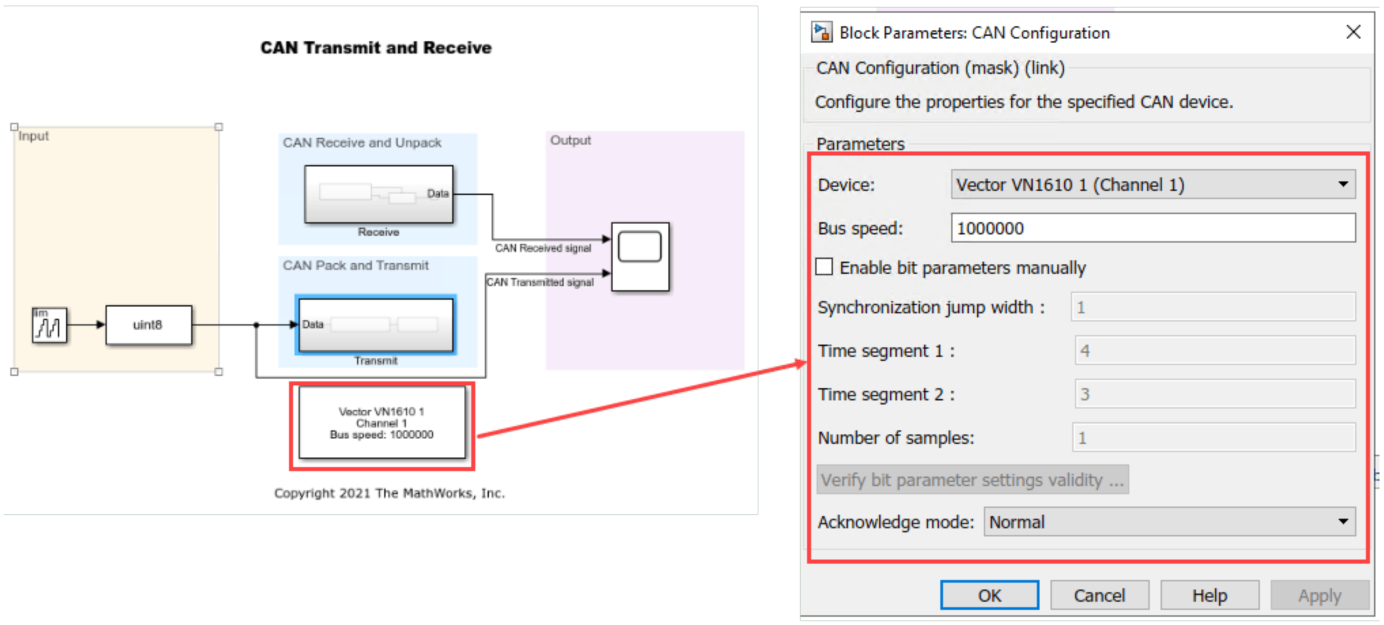
```
open_system('c2000_host_CAN_comm.slx');
```



2. Configure the device for CAN configuration, CAN Transmit and CAN Receive blocks. Ensure that you set the `baud rate` to **1Mbps** in CAN configuration block.

Note:

- The models in this example use the `Vector VN1610 1` hardware. However, you can connect your models to other supported hardware.
- Select the CAN device and channel as per your configuration.
- CAN Configuration, CAN Transmit and CAN Receive blocks on the host are from the Vehicle Network Toolbox™ library.



CAN Configuration (mask) (link)

Configure the properties for the specified CAN device.

Parameters

Device: Vector VN1610 1 (Channel 1)

Bus speed: 1000000

☐ Enable bit parameters manually

Synchronization jump width : 1

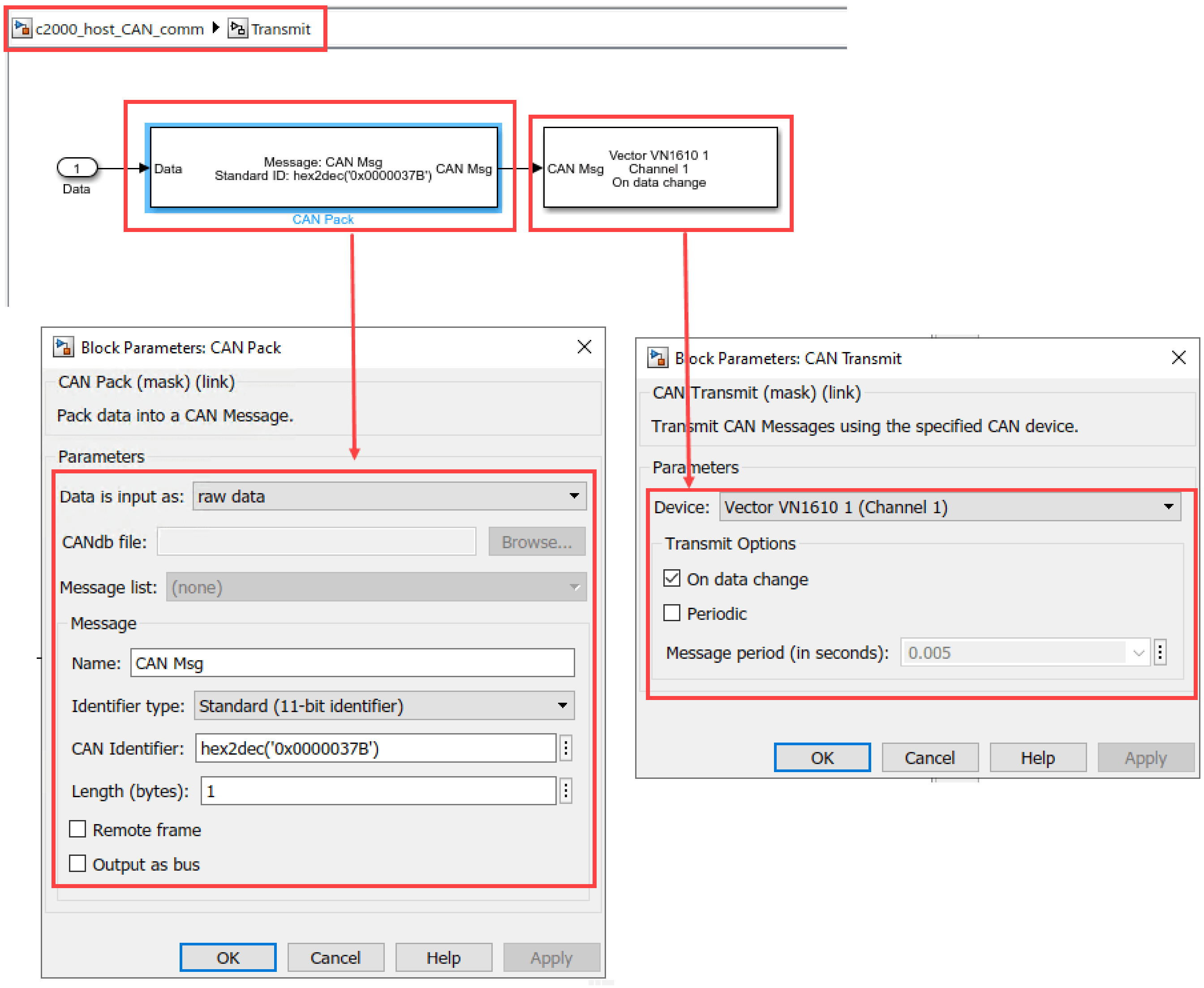
Time segment 1 : 4

Time segment 2 : 3

Number of samples: 1

Verify bit parameter settings validity ...

Acknowledge mode: Normal



Block Parameters: CAN Pack

CAN Pack (mask) (link)

Pack data into a CAN Message.

Parameters

Data is input as: raw data

CANdb file:

Message list: (none)

Message

Name: CAN Msg

Identifier type: Standard (11-bit identifier)

CAN Identifier: hex2dec('0x0000037B')

Length (bytes): 1

☐ Remote frame

☐ Output as bus

Block Parameters: CAN Transmit

CAN Transmit (mask) (link)

Transmit CAN Messages using the specified CAN device.

Parameters

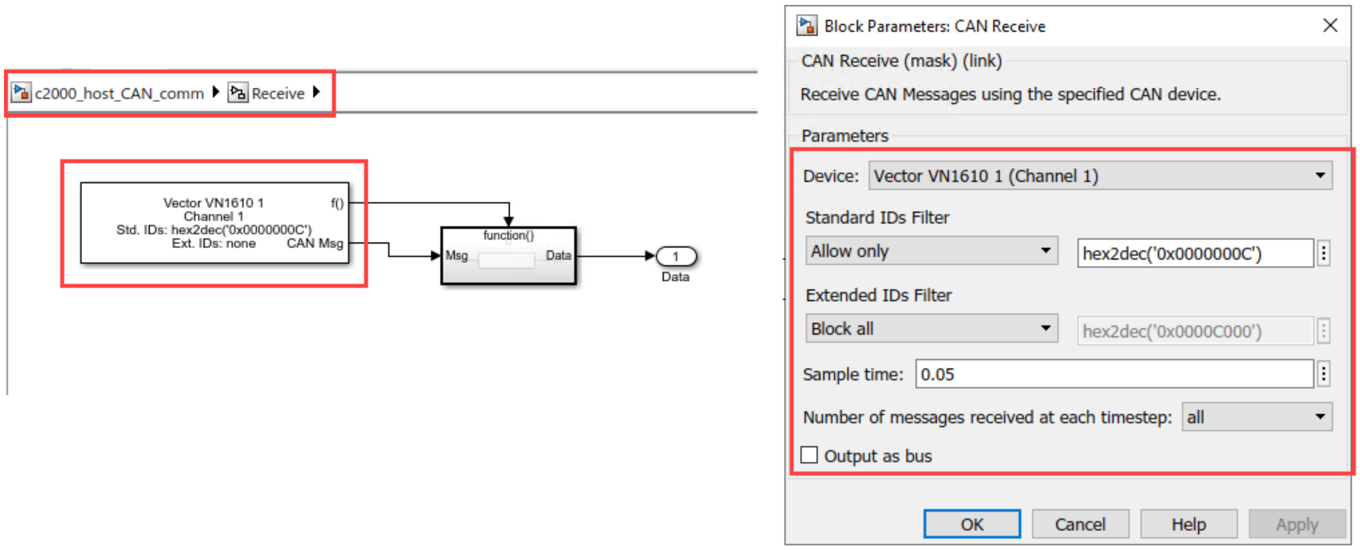
Device: Vector VN1610 1 (Channel 1)

Transmit Options

☑ On data change

☐ Periodic

Message period (in seconds): 0.005



Block Parameters: CAN Receive

CAN Receive (mask) (link)

Receive CAN Messages using the specified CAN device.

Parameters

Device: Vector VN1610 1 (Channel 1)

Standard IDs Filter: Allow only — hex2dec('0x0000000C')

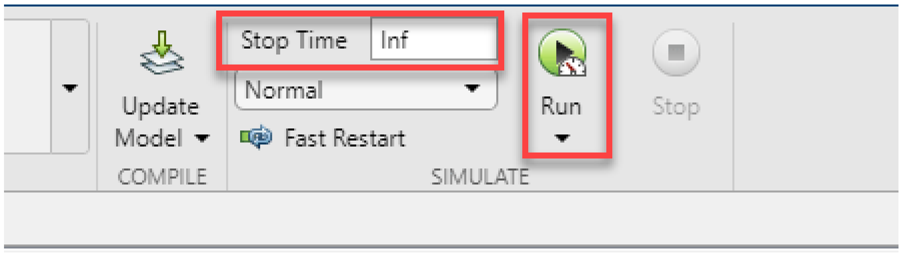
Extended IDs Filter: Block all — hex2dec('0x0000C000')

Sample time: 0.05

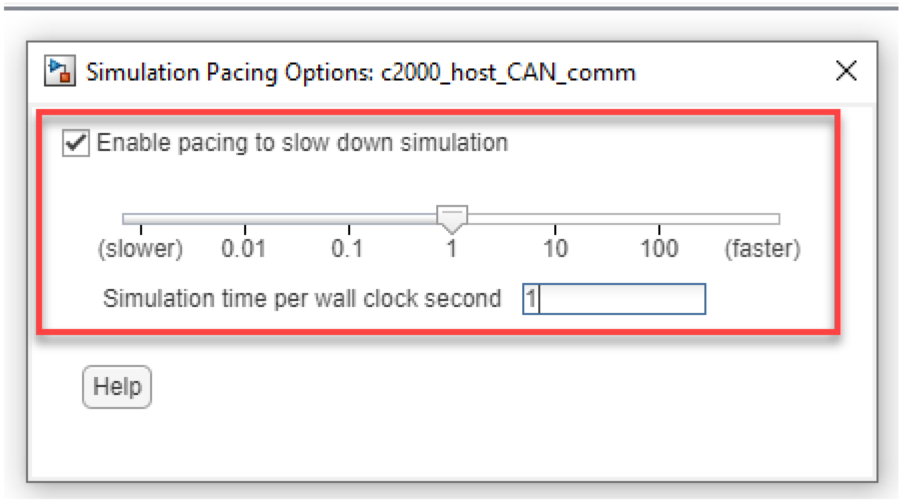
Number of messages received at each timestep: all

☐ Output as bus

3. Set the **Stop time** to **Inf** and click on **Run** drop-down and enable the **Simulation Pacing** in order to ensure real time communication with target.



Stop Time: Inf

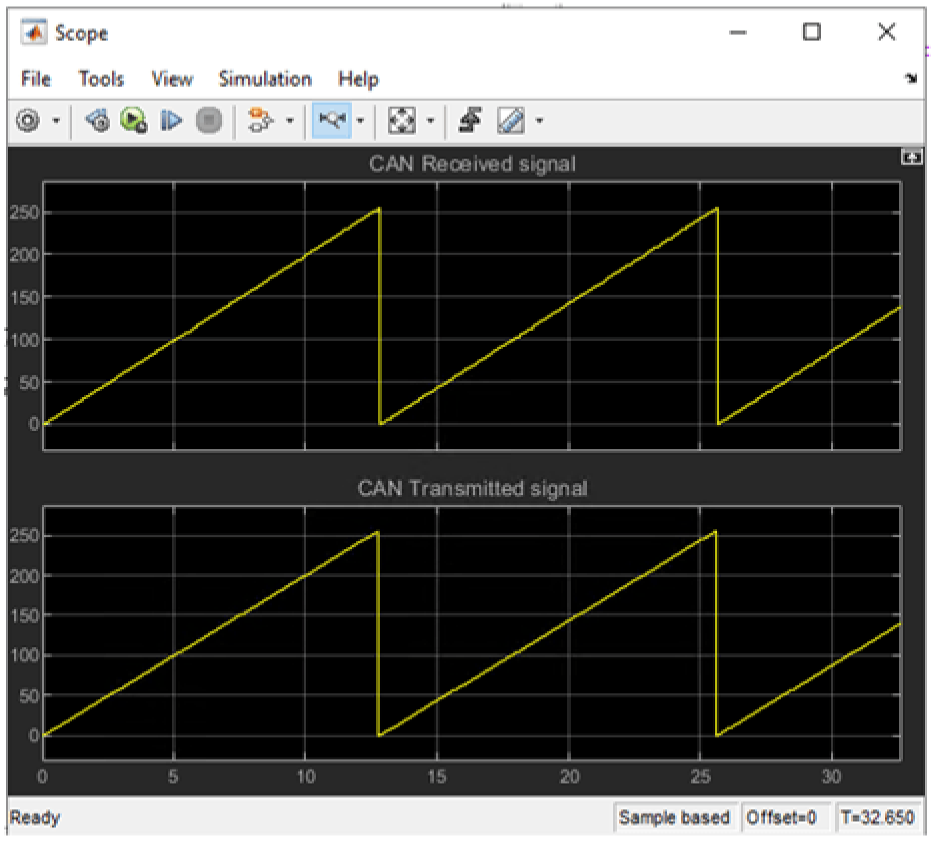

Simulation Pacing Options: c2000_host_CAN_comm

☑ Enable pacing to slow down simulation

Simulation time per wall clock second: 1

4. Click **Run**.

5. Observe and compare the CAN data received from target hardware and transmitted from host.



Task 2 - Use Acceptance Filtering on Target Hardware to Receive Specific Messages

In this task, you will do the following in the host and target models:

Host

- Transmit multiple standard and extended messages at different rates to the target.
- Receive messages from the target and compare.

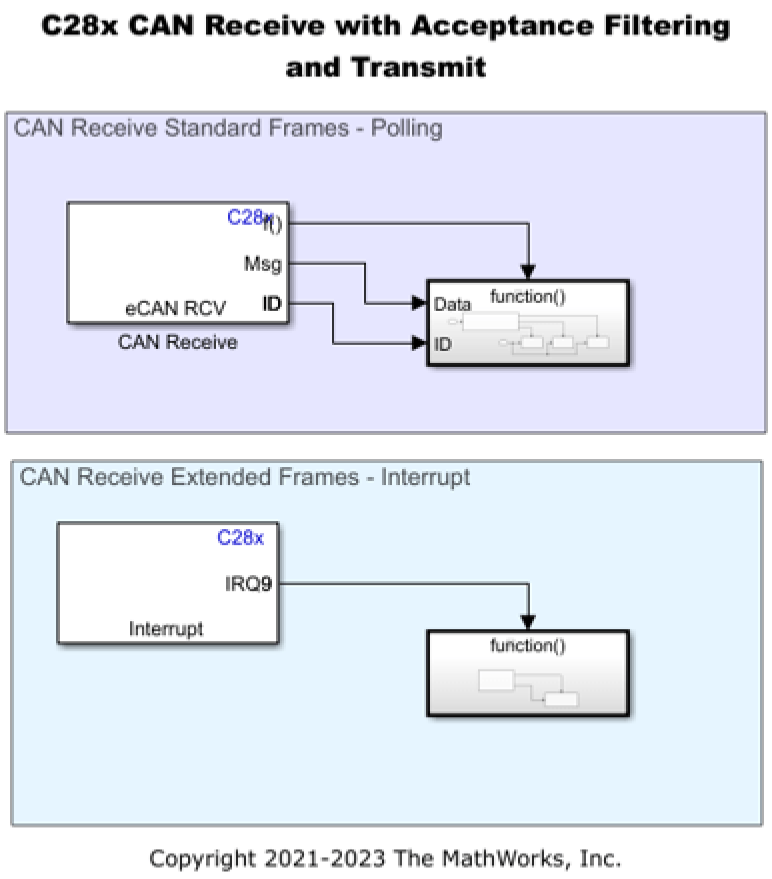
Target

- Receive the standard and extended messages from the host using two methods (polling and interrupt).
- Use ID filtering to receive only the required message from the host.
- Send the received data back to the host with a different ID for comparison.

Target Model

1. Open the target model.

```
open_system('c28x_CAN_acceptance_filtering.slx');
```


C28x CAN Receive with Acceptance Filtering and Transmit
CAN Receive Standard Frames - Polling
C28x
f()
Msg
ID
eCAN RCV
CAN Receive
Data
ID
function()
CAN Receive Extended Frames - Interrupt
C28x
IRQ9
Interrupt
function()
Copyright 2021-2023 The MathWorks, Inc.

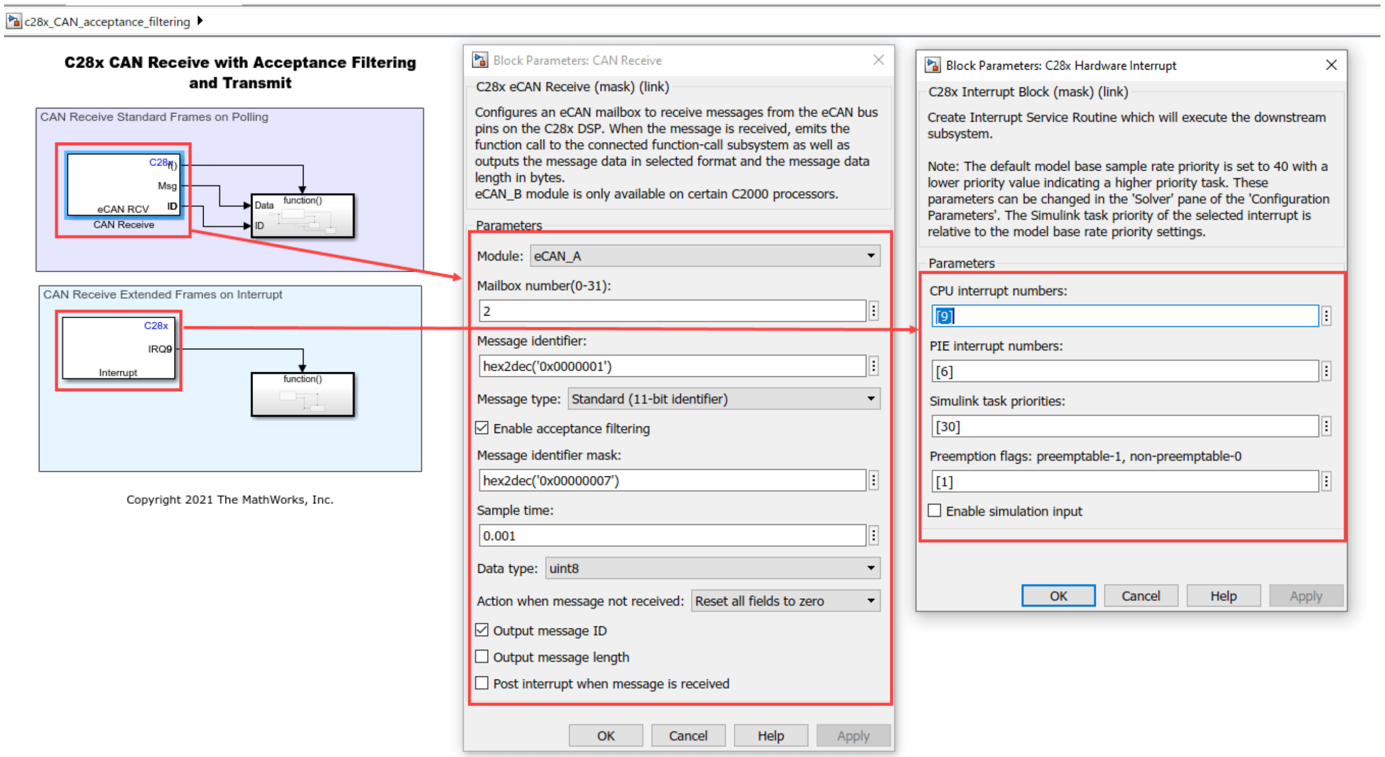
2. The example model is configured for the `TI Piccolo F28069M launchpad`. To select a different target hardware, in the Simulink Editor, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**.

3. Navigate to **Hardware Implementation** > **Target Hardware Resources** > **eCAN_A** and ensure that you set the **Baud rate** to `1Mbps`.

4. Target hardware receives standard ID messages using the polling method at the rate of 0.001 and extended ID messages using the interrupt method (ECAN1INTA is used to receive).

5. Following are the **eCAN** block configurations done for target model. Double-click on the blocks to open block parameter configurations. Ensure the specified parameter values are the same if you want to run this example for other hardware board

- The eCAN receive block is configured to receive standard messages as shown below:


c28x_CAN_acceptance_filtering
C28x CAN Receive with Acceptance Filtering and Transmit
CAN Receive Standard Frames on Polling
C28x
f()
Msg
eCAN RCV
ID
CAN Receive
Data
ID
function()
CAN Receive Extended Frames on Interrupt
C28x
IRQ9
Interrupt
function()
Copyright 2021 The MathWorks, Inc.

Block Parameters: CAN Receive

C28x eCAN Receive (mask) (link)

Configures an eCAN mailbox to receive messages from the eCAN bus pins on the C28x DSP. When the message is received, emits the function call to the connected function-call subsystem as well as outputs the message data in selected format and the message data length in bytes.
eCAN_B module is only available on certain C2000 processors.

Parameters

Module: eCAN_A

Mailbox number(0-31): 2

Message identifier: hex2dec('0x0000001')

Message type: Standard (11-bit identifier)

☑ Enable acceptance filtering

Message identifier mask: hex2dec('0x00000007')

Sample time: 0.001

Data type: uint8

Action when message not received: Reset all fields to zero

☑ Output message ID

☐ Output message length

☐ Post interrupt when message is received

OK Cancel Help Apply

Block Parameters: C28x Hardware Interrupt

C28x Interrupt Block (mask) (link)

Create Interrupt Service Routine which will execute the downstream subsystem.

Note: The default model base sample rate priority is set to 40 with a lower priority value indicating a higher priority task. These parameters can be changed in the 'Solver' pane of the 'Configuration Parameters'. The Simulink task priority of the selected interrupt is relative to the model base rate priority settings.

Parameters

CPU interrupt numbers: [9]

PIE interrupt numbers: [6]

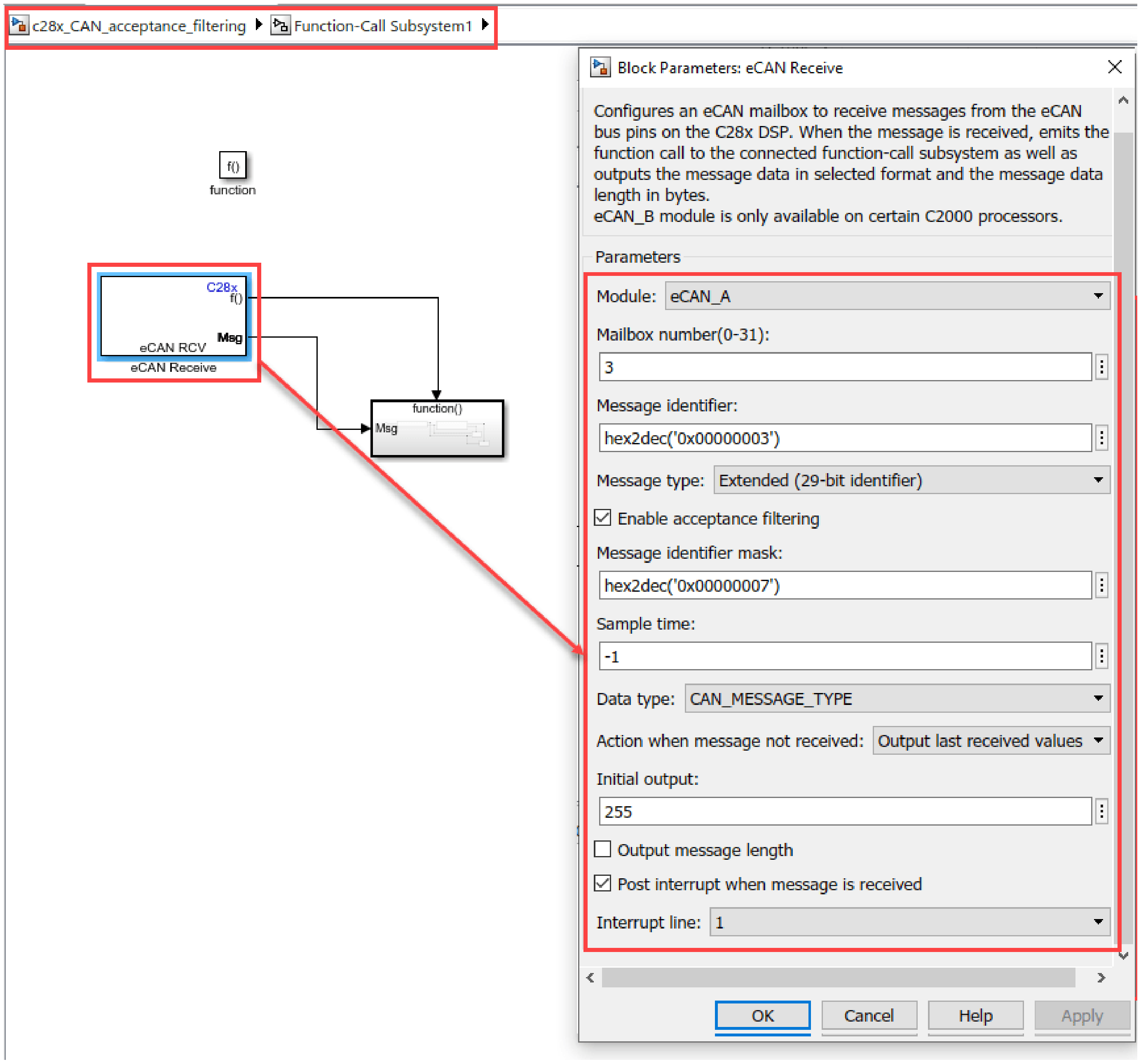
Simulink task priorities: [30]

Preemption flags: preemptable-1, non-preemptable-0: [1]

☐ Enable simulation input

OK Cancel Help Apply

- The eCAN receive block is configured to receive extended messages as shown below:



Block Parameters: eCAN Receive

Configures an eCAN mailbox to receive messages from the eCAN bus pins on the C28x DSP. When the message is received, emits the function call to the connected function-call subsystem as well as outputs the message data in selected format and the message data length in bytes.
eCAN_B module is only available on certain C2000 processors.

Parameters

Module: eCAN_A

Mailbox number(0-31): 3

Message identifier: hex2dec('0x00000003')

Message type: Extended (29-bit identifier)

☑ Enable acceptance filtering

Message identifier mask: hex2dec('0x00000007')

Sample time: -1

Data type: CAN_MESSAGE_TYPE

Action when message not received: Output last received values

Initial output: 255

☐ Output message length

☑ Post interrupt when message is received

Interrupt line: 1

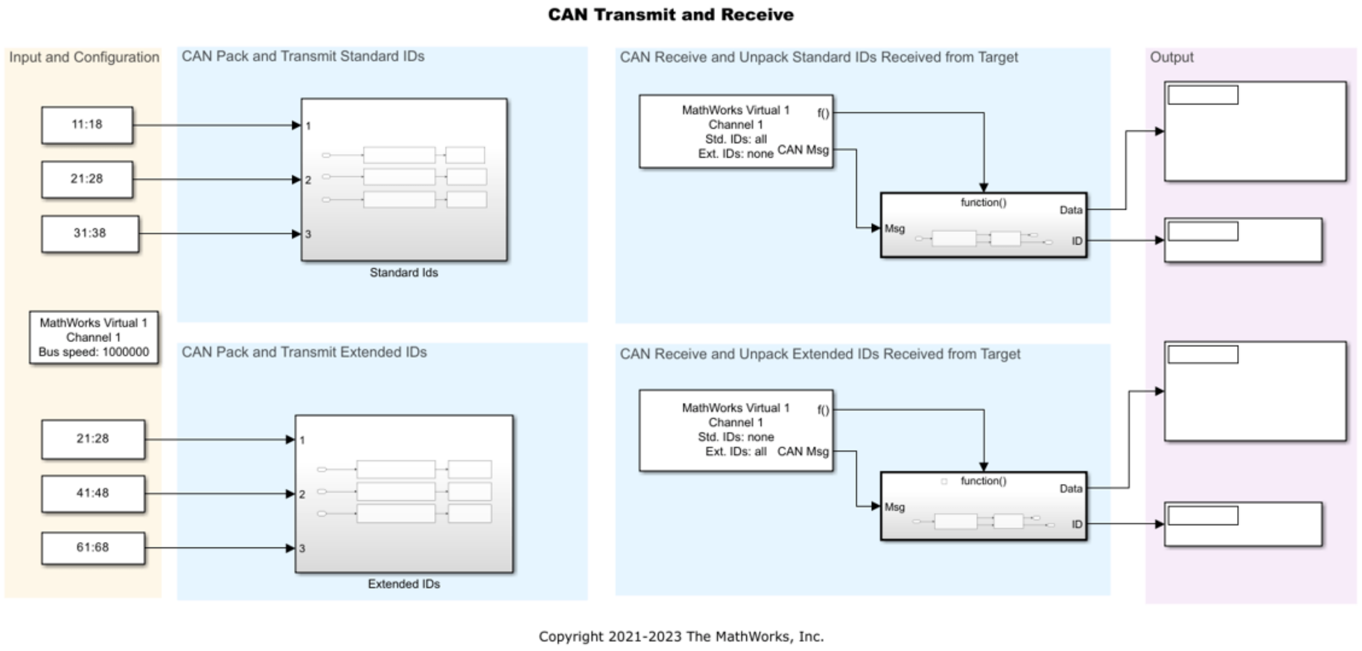
6. On receiving the new message, the target hardware transmits the received data with updated message IDs. There are multiple CAN Transmit blocks used to send the standard or extended messages with different **Message identifier** depending upon the received messages from the CAN receive block.

7. Click **Build, Deploy & Start** in the **Hardware** tab or press **Ctrl+B** to build and download the executable file.

Run on Host Model

1. Open the host model

```
open_system('c2000_host_CAN_acceptance_filtering.slx');
```

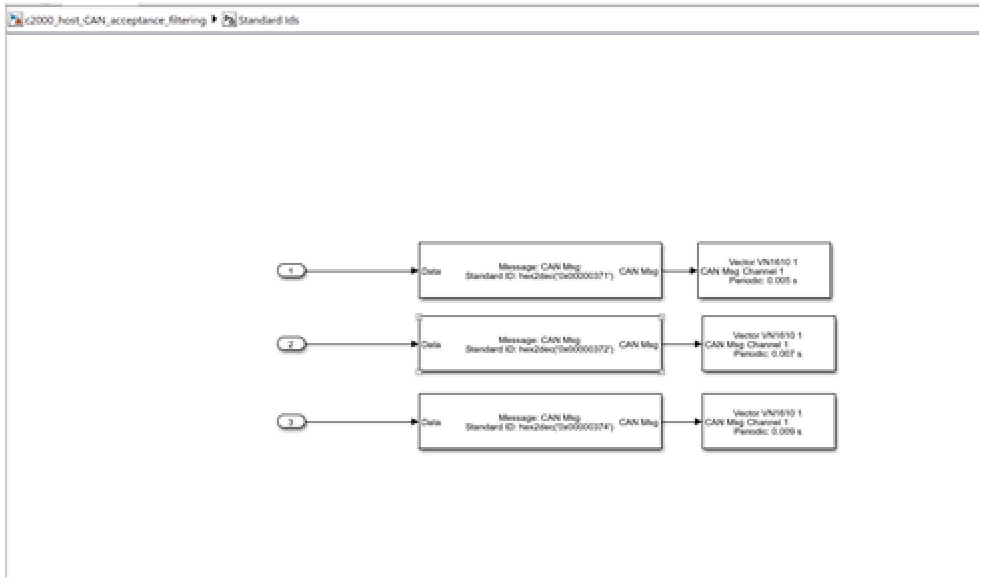


2. Configure the device for CAN configuration, CAN Transmit and CAN Receive blocks. Ensure that the baud rate is set to 1Mbps in CAN configuration block.

3. The model is configured to transmit three messages with different standard IDs and extended IDs.

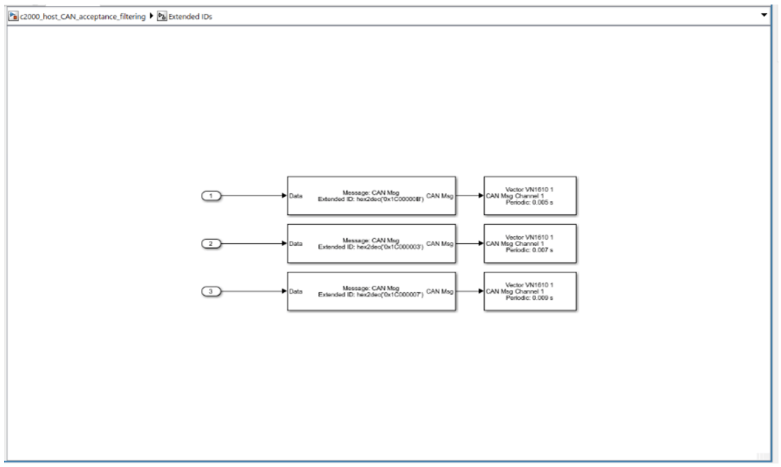
4. The CAN Transmit blocks are configured to transmit messages with `Standard IDs` at the following sample rates:

- hex2dec('0x00000371'): at 0.005
- hex2dec('0x00000372'): at 0.007
- hex2dec('0x00000374'): at 0.009



5. The CAN Transmit blocks are configured to transmit messages with `Extended IDs` at the following sample rates:

- hex2dec('0x1C00000B'): at 0.005
- hex2dec('0x1C000003'): at 0.007
- hex2dec('0x1C000007'): at 0.009



6. The host model has two Receive blocks, one receives all the standard messages and the other receives all the extended messages at the rate of sample time 0.001s.

7. Set the **Stop time** to **Inf** and click on **Run** drop-down and enable the **Simulation Pacing**.

8. Click **Run**

9. Observe the CAN data received on the host in the Display block. The data received is corresponding to the data of the messages received on the target hardware with different IDs.

Analysis of Acceptance Filtering Behavior

eCAN Receive block configured with Standard Message Type in Target Model

- **Message type** : `Standard`
- **Message identifier mask** : `hex2dec('0x00000007')`
- **Message identifier** : `hex2dec('0x00000001')`

As the **Message identifier mask** is set to `hex2dec('0x00000007')`, only the last 3 bits from **Message identifier** is considered for filtering.

These 3 bits in **Message identifier** is set to `1`, hence, it receives messages with only standard ID **hex2dec('0x00000371')** on the target hardware.

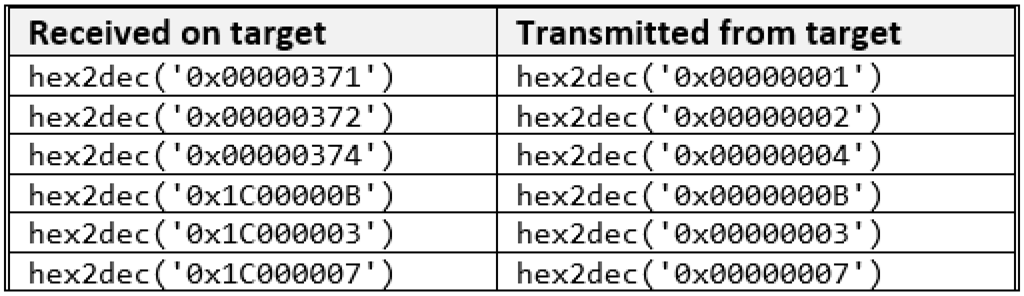
eCAN Receive block configured with Extended Message Type in Target Model

- **Message type** : `Extended`
- **Message identifier mask** : `hex2dec('0x00000007')`
- **Message identifier** : `hex2dec(''0x00000003'')`

As the **Message identifier mask** is set to `hex2dec('0x00000007')`, only the last 3 bits from **Message identifier** is considered for filtering.

These 3 bits in **Message identifier** is set to `3`, hence, it receives messages with only extended IDs **hex2dec('0x1C00000B')** and **hex2dec('0x1C000003')** on the target hardware.

When the target receives these messages, it transmits the same data with the modified message IDs to the host for comparison. The message IDs are modified in the target hardware are as shown here:



Received on target	Transmitted from target
hex2dec('0x00000371')	hex2dec('0x00000001')
hex2dec('0x00000372')	hex2dec('0x00000002')
hex2dec('0x00000374')	hex2dec('0x00000004')
hex2dec('0x1C00000B')	hex2dec('0x0000000B')
hex2dec('0x1C000003')	hex2dec('0x00000003')
hex2dec('0x1C000007')	hex2dec('0x00000007')

The host receives these corresponding IDs and data depending on the filtered IDs on the target hardware and it displays them.

Other Things to Try

- In the **eCAN Receive** block, which is configured to receive standard messages, change the **Message identifier** parameter to `hex2dec('0x00000002') or hex2dec('0x00000004')`, to receive messages with standard IDs of hex2dec('0x00000372') or hex2dec('0x00000374'), respectively, on the target hardware.
- In the eCAN Receive block, which is configured to receive extended messages, change the **Message identifier mask** parameter to `hex2dec('0x00000003')`, to receive all messages with extended IDs.

More About

- C28x eCAN Receive
- C28x eCAN Transmit
- “Set Up CAN Communication with Target Hardware” on page 1-8
- “Signal Monitoring and Parameter Tuning Over XCP-based CAN Interface” on page 4-333

Serial Communication Using SCI Blocks

This example shows how to use the SCI blocks to set up serial communication between the target hardware and your host computer for C2000™ Microcontroller Blockset.

Introduction

Using this example, you will:

- Establish communication between the target hardware and the host computer using SCI Transmit and Receive blocks on the target hardware and Instrumentation Control Toolbox™ Serial Transmit and Receive blocks on the host.
- Establish communication between the target hardware and the host computer using SCI Transmit and Receive blocks on the target hardware and a MATLAB® script on the host.
- Use the polling and interrupt methods to receive data on the target hardware.
- Use frame size to transmit data when the transmit rate is faster than the receive rate.

You can also use the model c2838x_adcpwmasync_TopModel.slx to perform Model Reference workflow. For more information, see

Prerequisites

Complete the following tutorials:

- "Getting Started with Texas Instruments C2000 Microcontroller Blockset" on page 4-2
- "Set Up Serial Communication with Target Hardware" on page 1-6. There are different control card versions available for C2000™ processors. In some control cards, the SCI_A module pins are directly connected to the USB docking station and other control cards have a MAX32xx chip for RS-232 communication on the control card.

Required Hardware

- Any Texas Instruments C2000 board

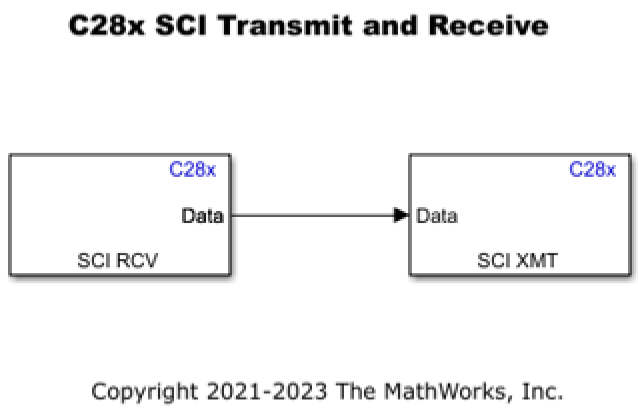
Task 1 - Serial Communication between Host Computer and Target Hardware using Serial Transmit and Receive Blocks on Host

In this task, you can transmit counter data from the host to the target hardware using Simulink® model on the host. When the target hardware receives data on polling, it transmits the same data back to the host. The host then displays the transmitted and received signals.

Target Model

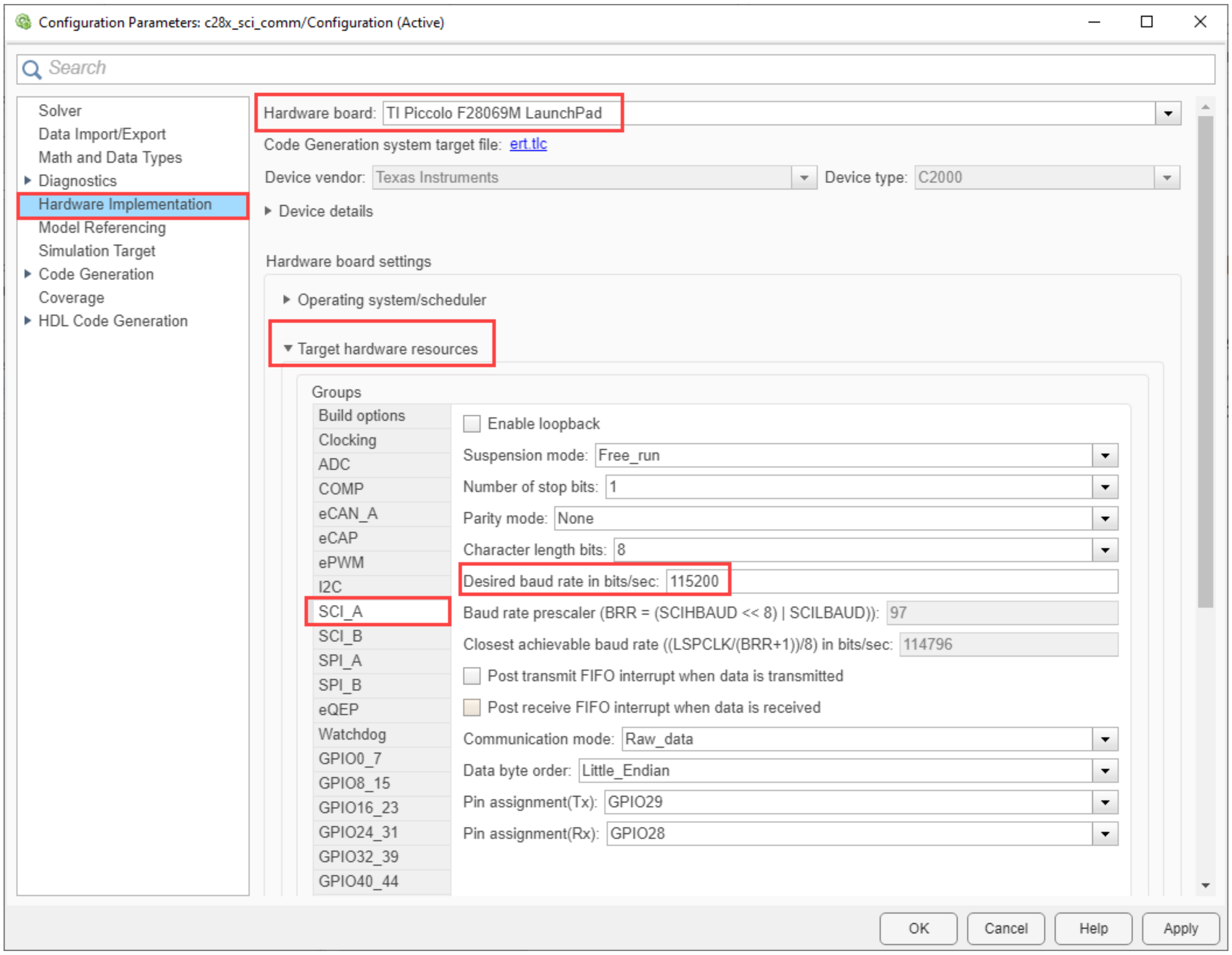
1. Open the target model.

```
open_system('c28x_sci_comm.slx');
```


C28x SCI Transmit and Receive
C28x
Data
SCI RCV
C28x
Data
SCI XMT
Copyright 2021-2023 The MathWorks, Inc.

2. The example model is configured for the TI Piccolo F28069M launchpad. To select different target hardware, in the Simulink® Editor, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**.

3. Navigate to **Hardware Implementation** > **Target Hardware Resources** > **SCI_A** and set the **Baud rate** to `115200`.


Configuration Parameters: c28x_sci_comm/Configuration (Active)

Hardware board: TI Piccolo F28069M LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments | Device type: C2000

Hardware board settings

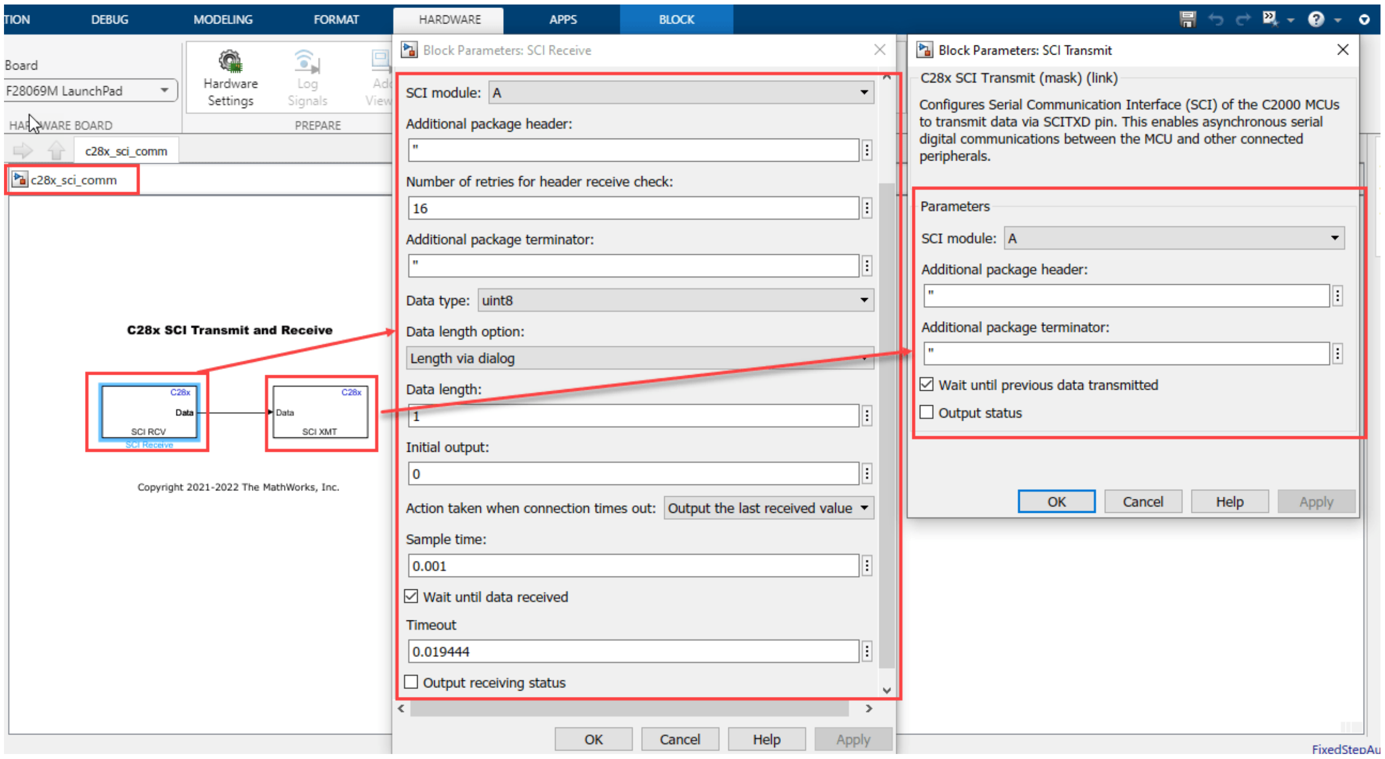
Target hardware resources

Groups: SCI_A

Enable loopback	☐
Suspension mode:	Free_run
Number of stop bits:	1
Parity mode:	None
Character length bits:	8
Desired baud rate in bits/sec:	115200
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)):	97
Closest achievable baud rate ((LSPCLK/(BRR+1))/8) in bits/sec:	114796
Post transmit FIFO interrupt when data is transmitted	☐
Post receive FIFO interrupt when data is received	☐
Communication mode:	Raw_data
Data byte order:	Little_Endian
Pin assignment(Tx):	GPIO29
Pin assignment(Rx):	GPIO28

4. The model is configured to receive the data without **header** and **terminator** at a sample time of `0.001 s` and to transmit the same data.

5. Following figure shows the SCI block configurations for the target model. Double-click on the blocks to open block parameter configurations. Ensure the specified parameter values are the same if you want to run this example for other hardware boards.



Block Parameters: SCI Receive

SCI module: A

Additional package header: ''

Number of retries for header receive check: 16

Additional package terminator: ''

Data type: uint8

Data length option: Length via dialog

Data length: 1

Initial output: 0

Action taken when connection times out: Output the last received value

Sample time: 0.001

☑ Wait until data received

Timeout: 0.019444

☐ Output receiving status

OK Cancel Help Apply

Block Parameters: SCI Transmit

C28x SCI Transmit (mask) (link)

Configures Serial Communication Interface (SCI) of the C2000 MCUs to transmit data via SCITXD pin. This enables asynchronous serial digital communications between the MCU and other connected peripherals.

Parameters

SCI module: A

Additional package header: ''

Additional package terminator: ''

☑ Wait until previous data transmitted

☐ Output status

OK Cancel Help Apply

Things to consider

You must consider the following things when configuring the SCI Receive block. In this example, the c28x_sci_comm model is configured to **Wait until data received** with timeout for blocking mode.

- **Blocking mode** - Select the **Wait until data received** parameter and set **Timeout** to `inf`. In this mode, if data is not available in FIFO to read, the block waits for infinite time until the data is available to read.
- **Blocking mode with Timeout** - Select the **Wait until data received** parameter and set **Timeout** to `any finite value greater than 0`. In this mode, if data is not available in FIFO to read, the block checks the FIFO status until the timeout value is reached. If data is not available in FIFO to read within that time, then the SCI Receive block outputs its status as timeout.
- **Non-Blocking mode** - Do not select the **Wait until data received** parameter. In this mode the SCI Receive block reads the data if the data is available in FIFO; otherwise the block outputs its status as **Data not available**.

In order to receive data of length more than the FIFO length, use either blocking mode or blocking mode with timeout. This ensures extra time to get the remaining data in FIFO after reading the entire FIFO.

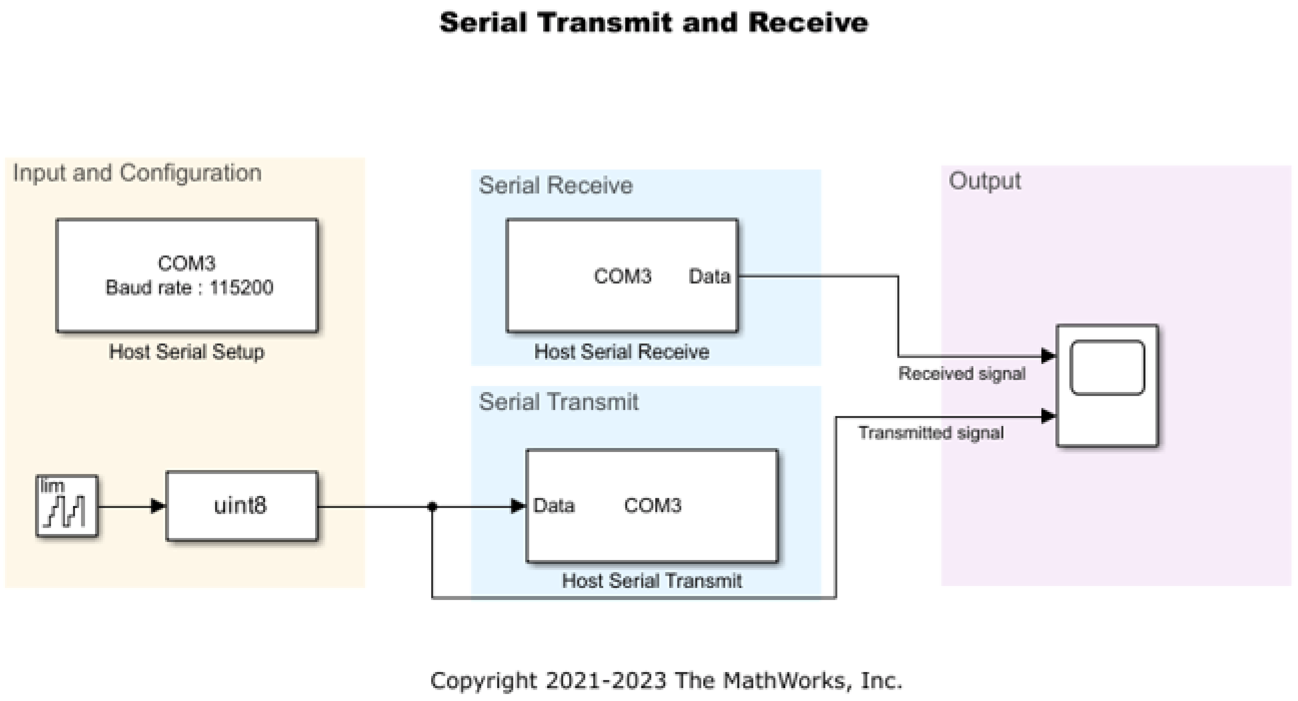
In blocking mode with and without timeout enabled, you might encounter task overrun as it waits for the data to read.

6. Click **Build, Deploy & Start** in the Hardware tab or press **Ctrl+B** to build and download the executable file.

Run Host Model

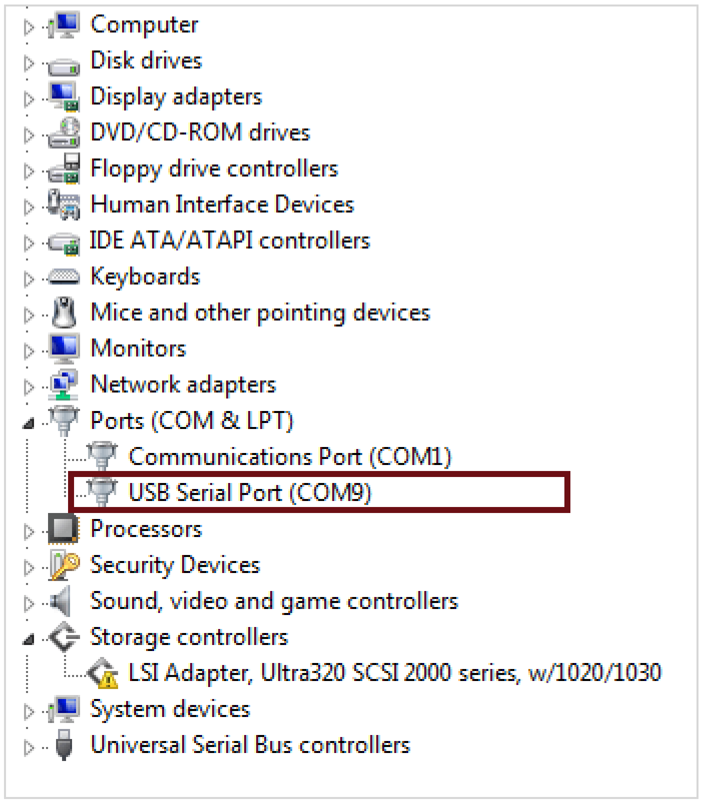
1. Open the host model.

```
open_system('c2000_host_serial_comm.slx');
```

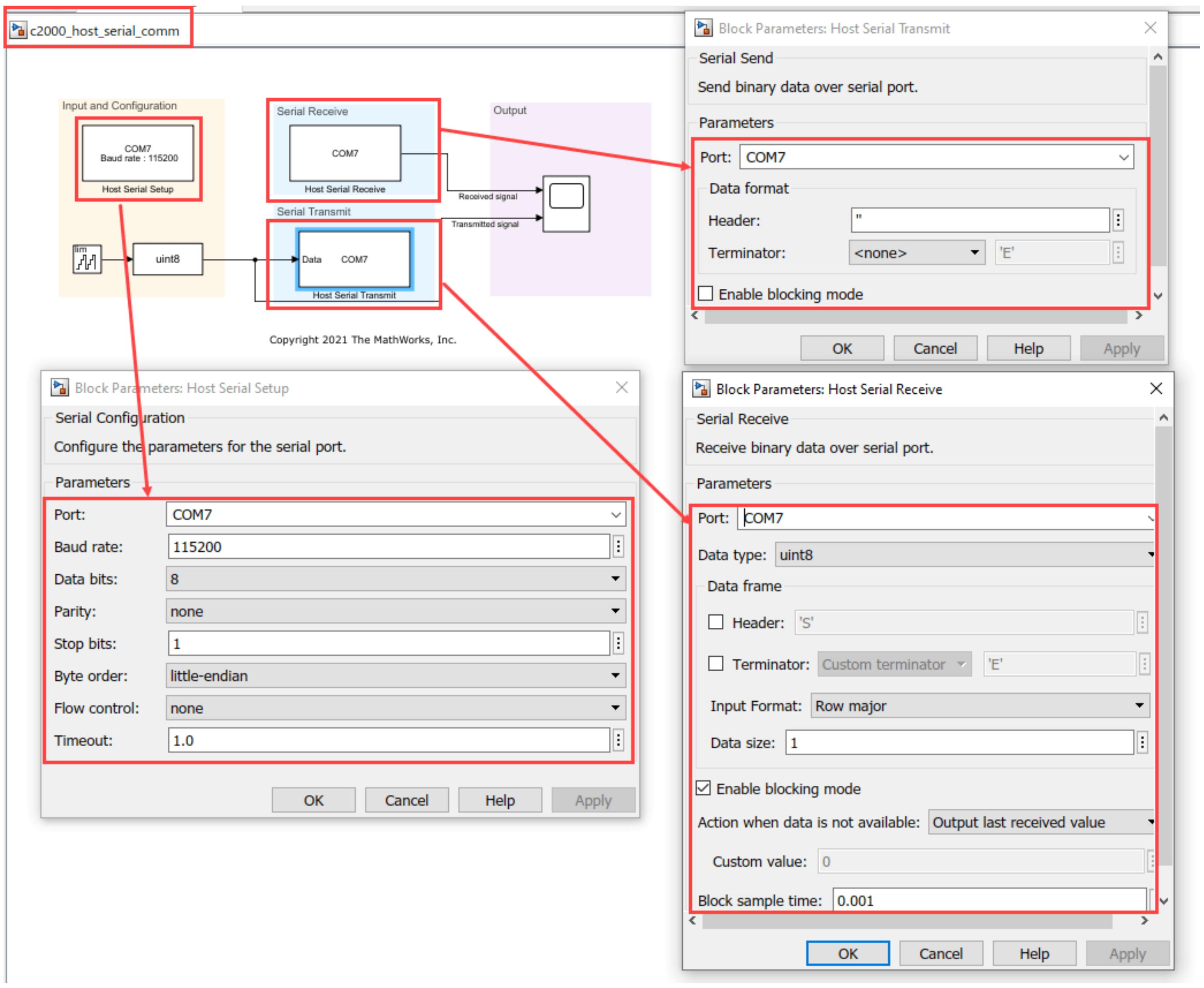


2. To see the list of available COM ports on your computer, select **Start > Control Panel > Device Manager > Ports (COM & LPT)**.

Note: The COM port shown in the figure is for illustration purpose and might vary for your computer. Select the COM port applicable for you computer.



3. Configure the device for **Host Serial Setup**, **Host Serial Transmit**, and **Host Serial Receive** blocks. Ensure that you set the **baud rate** to `115200` in the Host Serial Setup block.



Block Parameters: Host Serial Transmit

Serial Send

Send binary data over serial port.

Parameters

Port: COM7

Data format

Header: ''

Terminator: <none> 'E'

Enable blocking mode

Block Parameters: Host Serial Setup

Serial Configuration

Configure the parameters for the serial port.

Parameters

Port:	COM7
Baud rate:	115200
Data bits:	8
Parity:	none
Stop bits:	1
Byte order:	little-endian
Flow control:	none
Timeout:	1.0

Block Parameters: Host Serial Receive

Serial Receive

Receive binary data over serial port.

Parameters

Port: COM7

Data type: uint8

Data frame

Header: 'S'

Terminator: Custom terminator 'E'

Input Format: Row major

Data size: 1

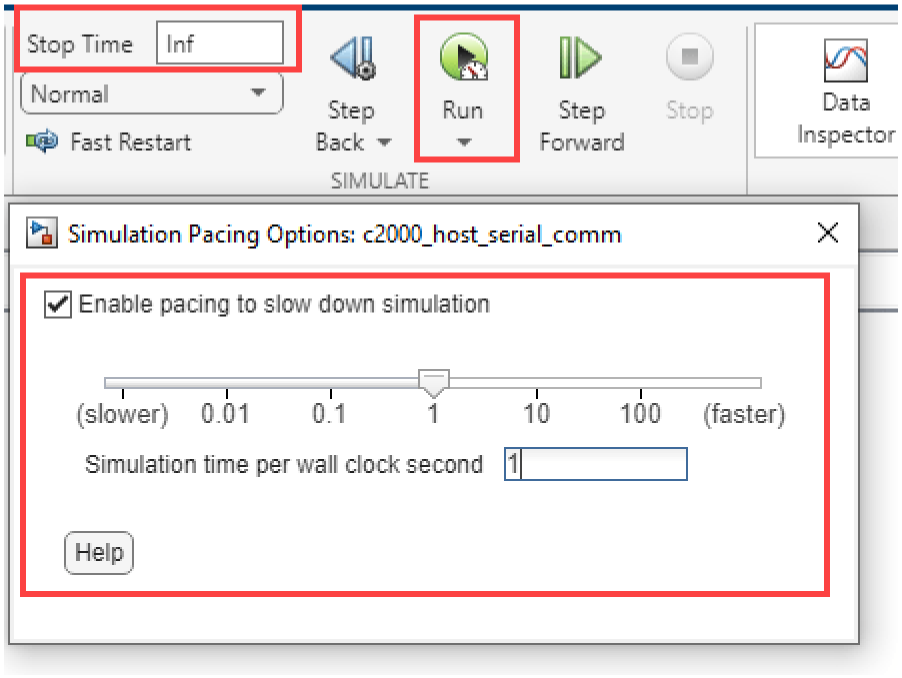
☑ Enable blocking mode

Action when data is not available: Output last received value

Custom value: 0

Block sample time: 0.001

4. Set the **Stop time** to `Inf` and click on **Run** drop-down and enable the **Simulation Pacing** in order to ensure real time communication with target.



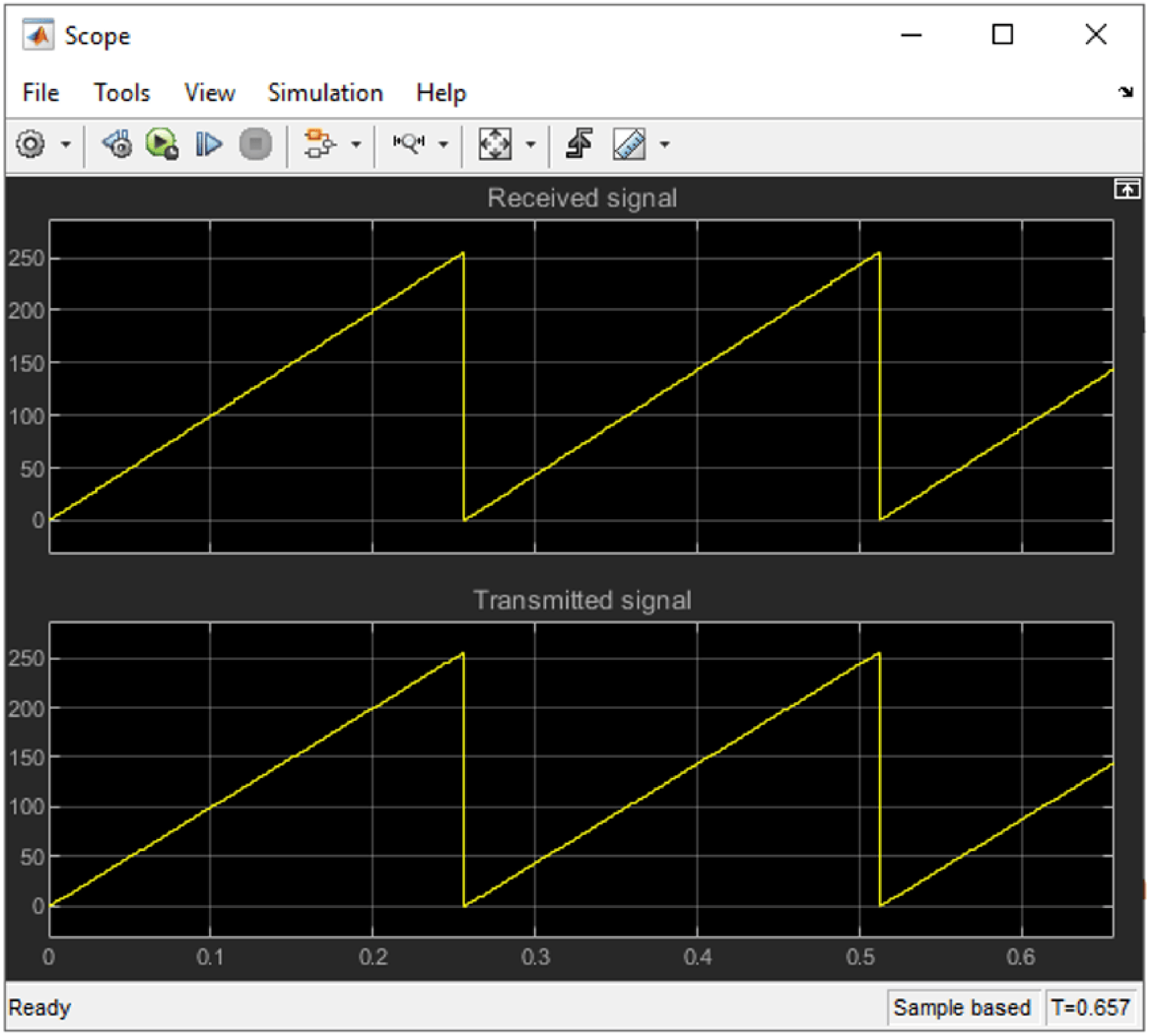
Simulation Pacing Options: c2000_host_serial_comm

✓ Enable pacing to slow down simulation

Simulation time per wall clock second: 1

5. Click **Run**.

6. Compare the serial data received from the target hardware and transmitted from the host.



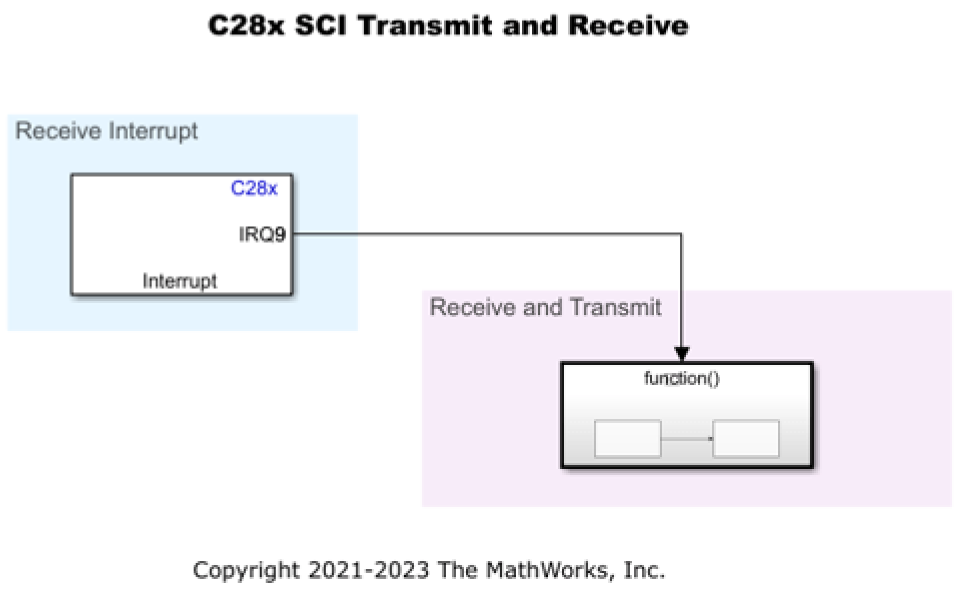
Task 2 - Serial Communication between Host Computer and Target Hardware Using MATLAB Script on Host

In this task, you will transmit **uint8** data from the host to the target hardware using a MATLAB® script on the host. When the target hardware receives the data on interrupt, it transmits the same data back to the host. The host then displays the received data in the MATLAB command window.

Target Model

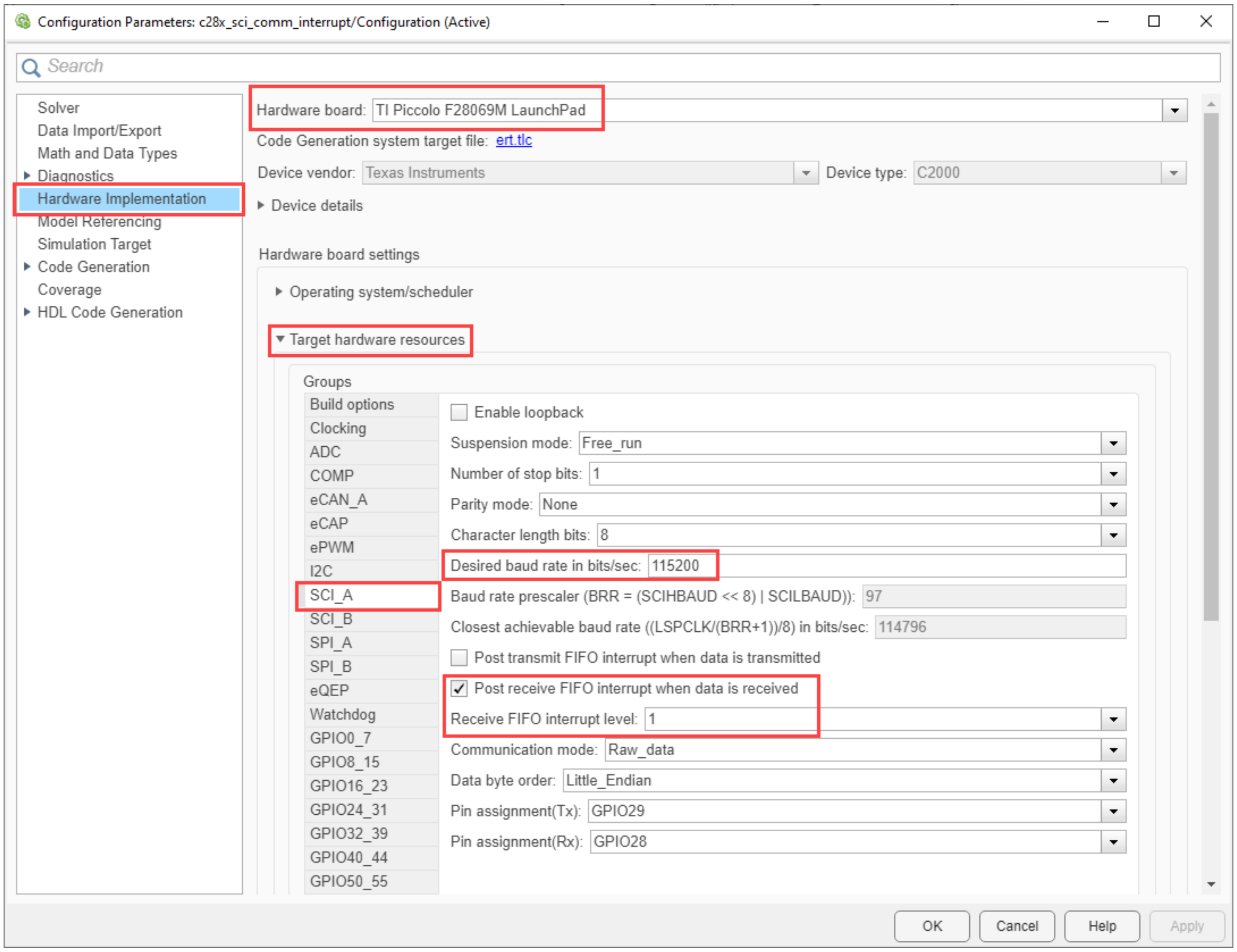
1. Open the target model.

```
open_system('c28x_sci_comm_interrupt.slx');
```


C28x SCI Transmit and Receive
Receive Interrupt
C28x
IRQ9
Interrupt
Receive and Transmit
function()
Copyright 2021-2023 The MathWorks, Inc.

2. The example model is configured for the TI Piccolo F28069M launchpad. To select a different target hardware, in the Simulink® Editor, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**.

3. Navigate to **Hardware Implementation** > **Target Hardware Resources** > **SCI_A** and ensure that you set the **Baud rate** to `115200`.



Configuration Parameters: c28x_sci_comm_interrupt/Configuration (Active)

Hardware board: TI Piccolo F28069M LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

☐ Enable loopback

Suspension mode: Free_run

Number of stop bits: 1

Parity mode: None

Character length bits: 8

Desired baud rate in bits/sec: 115200

Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)): 97

Closest achievable baud rate ((LSPCLK/(BRR+1))/8) in bits/sec: 114796

☐ Post transmit FIFO interrupt when data is transmitted

☑ Post receive FIFO interrupt when data is received

Receive FIFO interrupt level: 1

Communication mode: Raw_data

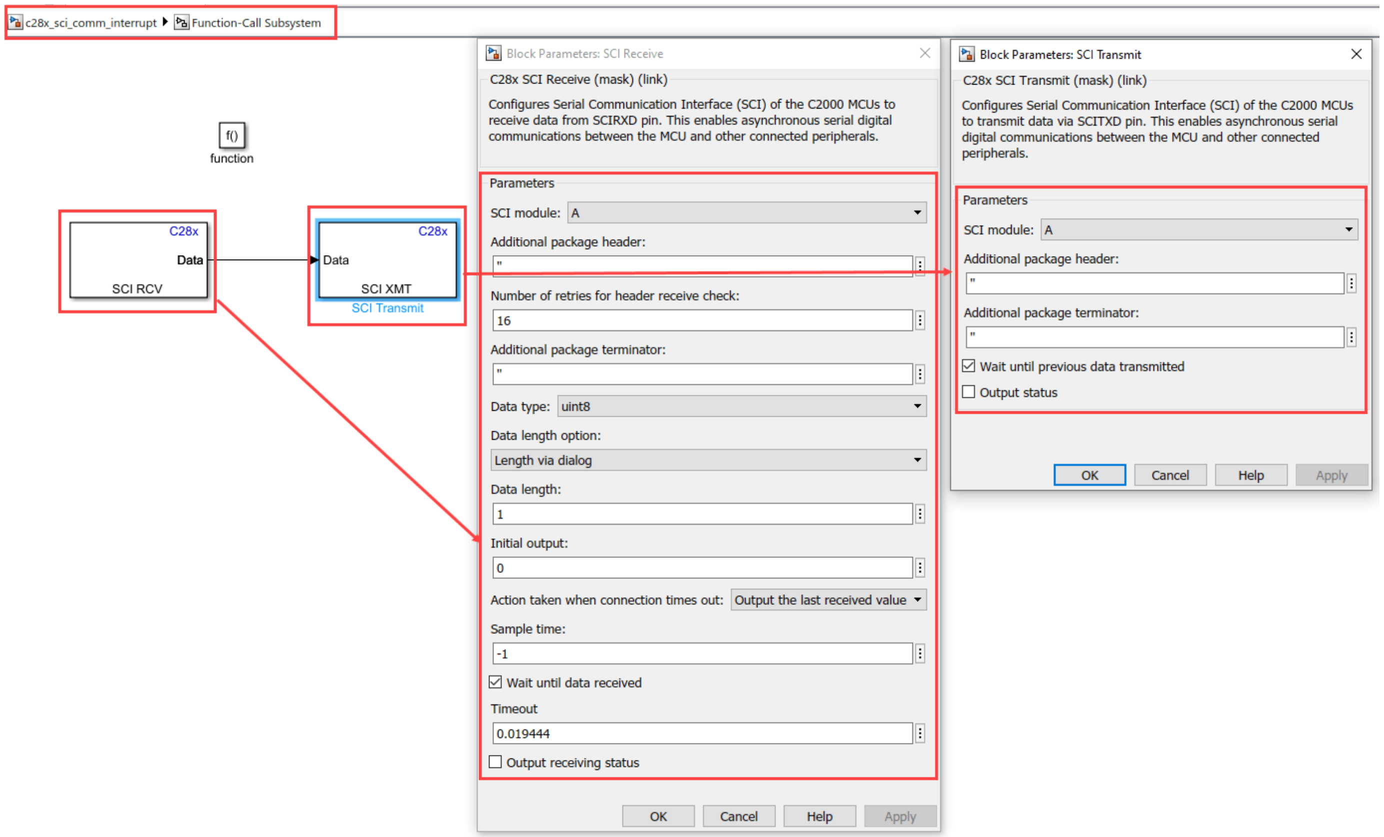
Data byte order: Little_Endian

Pin assignment(Tx): GPIO29

Pin assignment(Rx): GPIO28

4. Target hardware receives the data on interrupt. **SCIA_RX** is used to receive on interrupt level 1.

5. Following are the SCI block configurations for the target model. Double-click on the blocks to open block parameter configurations. Follow the section **Things to consider** from **Task 1**. Ensure that the specified parameter values are the same if you want to run this example for other hardware board. In this example, the c28x_sci_comm_interrupt model is configured to **Wait until data received** with timeout for blocking mode.



Block Parameters: SCI Receive

C28x SCI Receive (mask) (link)

Configures Serial Communication Interface (SCI) of the C2000 MCUs to receive data from SCIRXD pin. This enables asynchronous serial digital communications between the MCU and other connected peripherals.

Parameters

SCI module: A

Additional package header: ''

Number of retries for header receive check: 16

Additional package terminator: ''

Data type: uint8

Data length option: Length via dialog

Data length: 1

Initial output: 0

Action taken when connection times out: Output the last received value

Sample time: -1

[✓] Wait until data received

Timeout: 0.019444

[ ] Output receiving status

Block Parameters: SCI Transmit

C28x SCI Transmit (mask) (link)

Configures Serial Communication Interface (SCI) of the C2000 MCUs to transmit data via SCITXD pin. This enables asynchronous serial digital communications between the MCU and other connected peripherals.

Parameters

SCI module: A

Additional package header: ''

Additional package terminator: ''

[✓] Wait until previous data transmitted

[ ] Output status

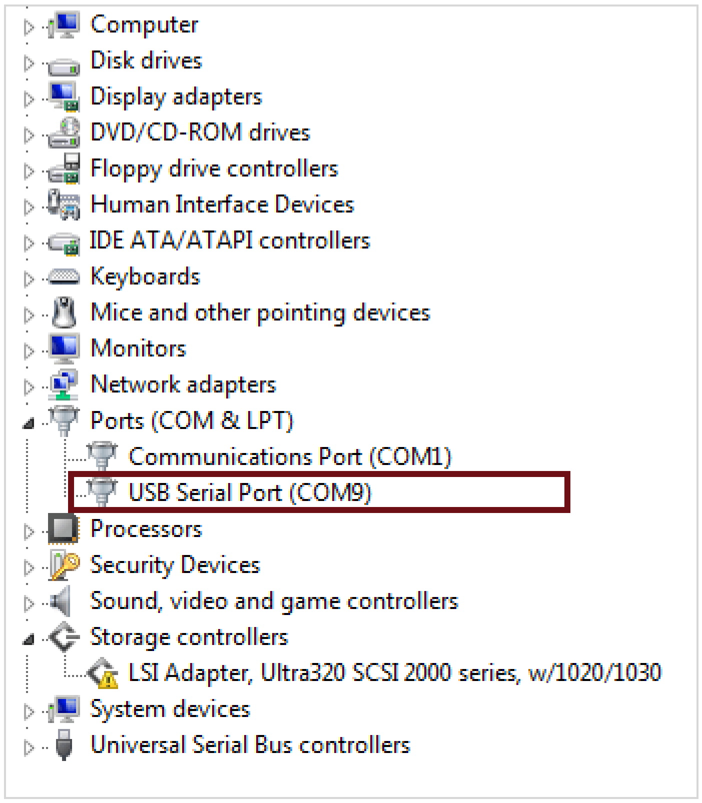
6. On receiving the new data, an interrupt is triggered, and the target hardware transmits the received data.

7. Click **Build, Deploy & Start** in the Hardware tab or press **Ctrl+B** to build and download the executable file.

Run on Host Using MATLAB Script

1. To run the model on host using MATLAB script, use **COM port** as an input argument and **baud rate** as an optional input argument. If baud rate is not provided, then by default a `115200` baud rate is used.

2. To see the list of available COM ports on your computer, select **Start > Control Panel > Device Manager > Ports (COM & LPT)**.

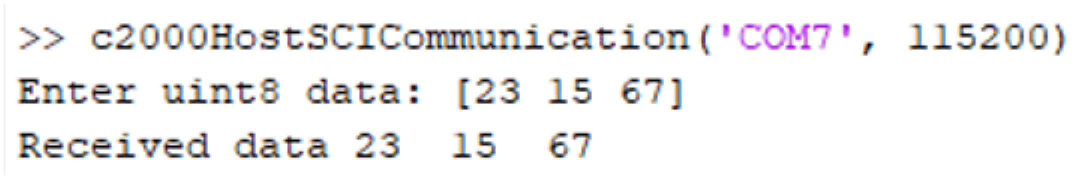


3. Run the following command at the MATLAB command prompt. Provide the COM port number of your computer. For example,

```
c2000HostSCICommunication('COM7', 115200);
```

4. Provide **uint8** data as an input in response to the prompt.

5. Observe that the same data is received in the MATLAB command window.



```
>> c2000HostSCICommunication('COM7', 115200)
Enter uint8 data: [23 15 67]
Received data 23   15   67
```

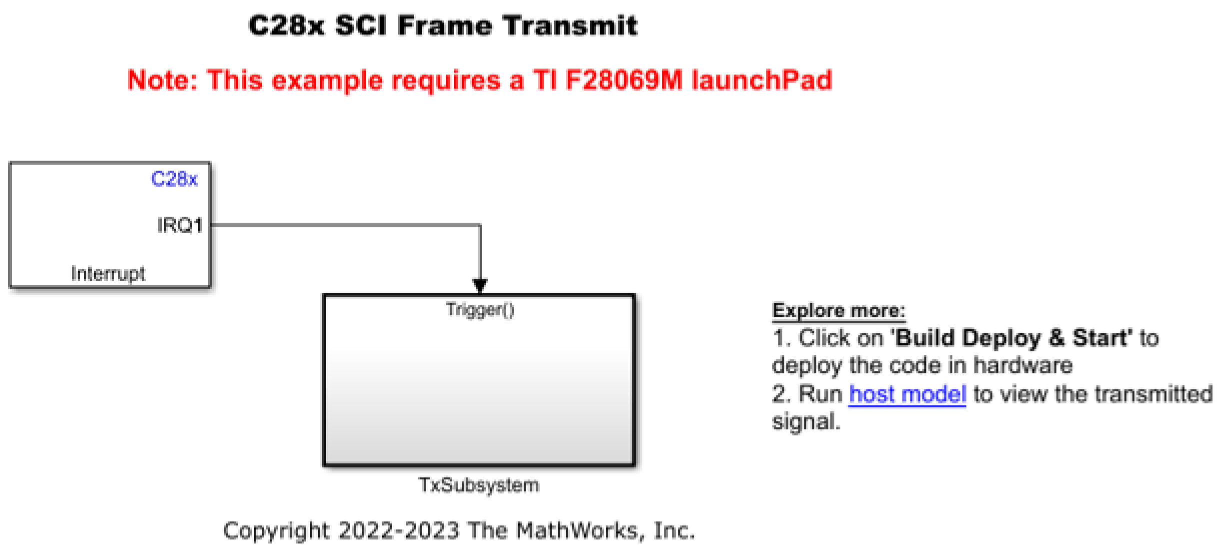
Task 3 - Use Frame Size to Transmit Data

In this task, you can transmit frame of data of size more than from the host to the target hardware using Simulink® model on the host. The host then displays the transmitted signal at a slower rate.

Target Model

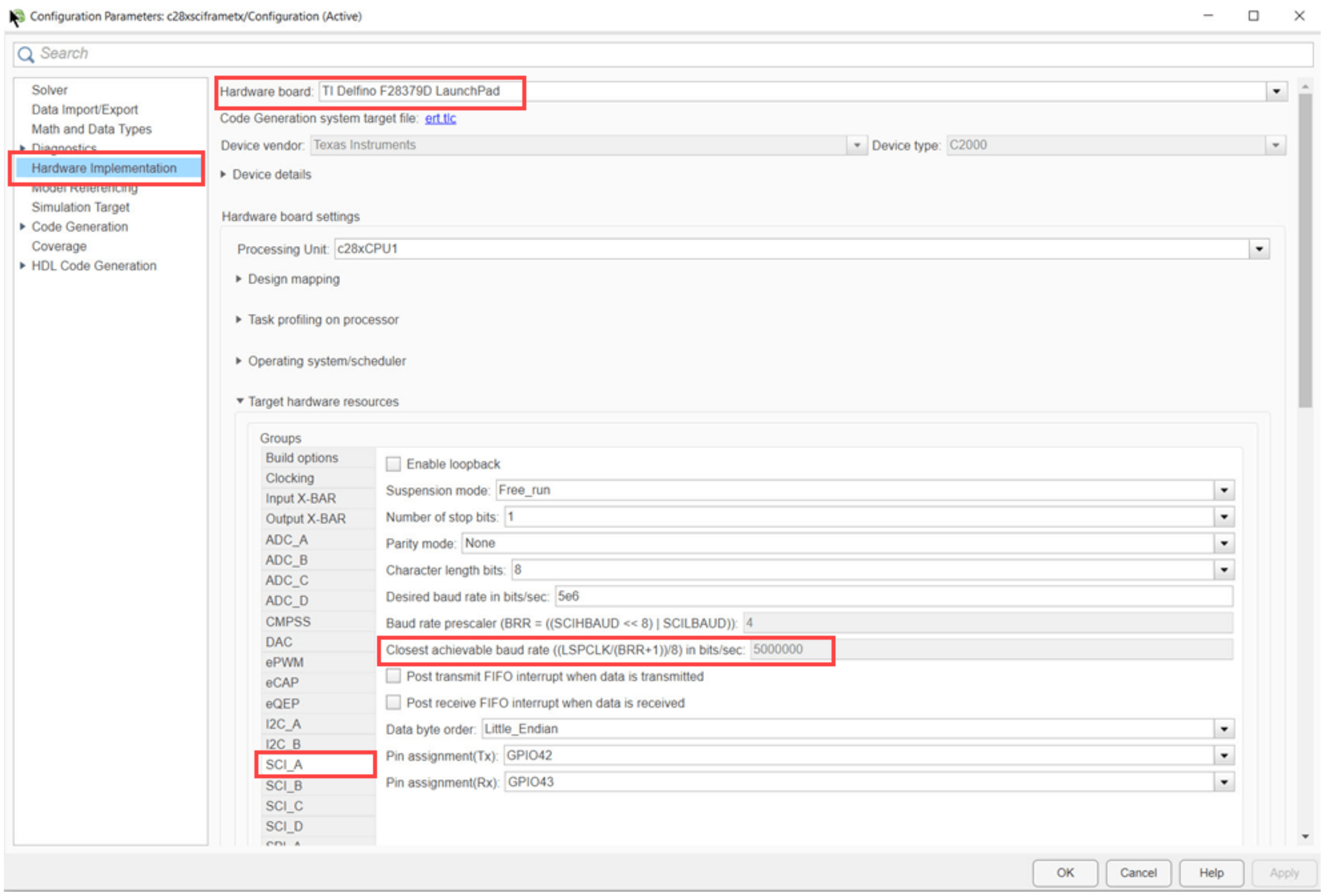
1. Open the target model.

```
open_system('c28xsciframetx.slx');
```


C28x SCI Frame Transmit
Note: This example requires a TI F28069M launchPad
C28x
IRQ1
Interrupt
Trigger()
TxSubsystem
Explore more:
1. Click on 'Build Deploy & Start' to deploy the code in hardware
2. Run host model to view the transmitted signal.
Copyright 2022-2023 The MathWorks, Inc.

2. The example model is configured for the TI Delfino F28379D launchpad. To select a different target hardware, in the Simulink® Editor, browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**.

3. Navigate to **Hardware Implementation** > **Target Hardware Resources** > **SCI_A** and ensure that you set the **Baud rate** to 5e6.



Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Device details

Hardware board settings

Processing Unit: c28xCPU1

Design mapping

Task profiling on processor

Operating system/scheduler

Target hardware resources

Enable loopback

Suspension mode: Free_run

Number of stop bits: 1

Parity mode: None

Character length bits: 8

Desired baud rate in bits/sec: 5e6

Baud rate prescaler (BRR = ((SCIHBAUD << 8) | SCILBAUD)): 4

Closest achievable baud rate ((LSPCLK/(BRR+1))/8) in bits/sec: 5000000

Post transmit FIFO interrupt when data is transmitted

Post receive FIFO interrupt when data is received

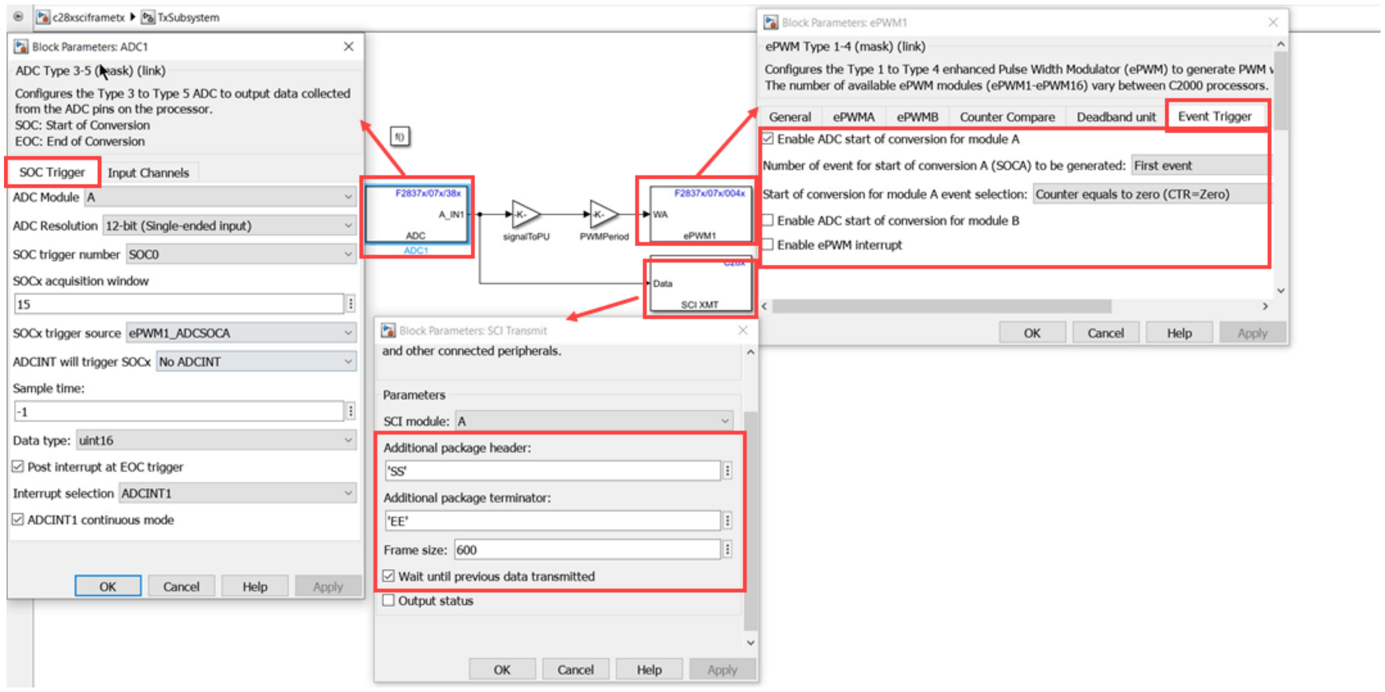
Data byte order: Little_Endian

Pin assignment(Tx): GPIO42

Pin assignment(Rx): GPIO43

4. The model is configured to receive the data with header and terminator at a sample time of `0.03s` and the data is transmitted at a rate determined by ADC interrupt which is `5e-6`.

5. Following are the SCI block configurations for the target model. Double-click on the blocks to open block parameter configurations. Ensure the specified parameter values are the same if you want to run this example for other hardware boards.



Block Parameters: ADC1

ADC Type 3-5 (mask) (link)

Configures the Type 3 to Type 5 ADC to output data collected from the ADC pins on the processor.
SOC: Start of Conversion
EOC: End of Conversion

SOC Trigger | Input Channels

ADC Module: A
ADC Resolution: 12-bit (Single-ended input)
SOC trigger number: SOC0
SOCx acquisition window: 15
SOCx trigger source: ePWM1_ADCSOCA
ADCINT will trigger SOCx: No ADCINT
Sample time: -1
Data type: uint16
☑ Post interrupt at EOC trigger
Interrupt selection: ADCINT1
☑ ADCINT1 continuous mode

Block Parameters: ePWM1

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM w
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

☑ Enable ADC start of conversion for module A
Number of event for start of conversion A (SOCA) to be generated: First event
Start of conversion for module A event selection: Counter equals to zero (CTR=Zero)
☐ Enable ADC start of conversion for module B
☐ Enable ePWM interrupt

Block Parameters: SCI Transmit

and other connected peripherals.

Parameters
SCI module: A
Additional package header: 'SS'
Additional package terminator: 'EE'
Frame size: 600
☑ Wait until previous data transmitted
☐ Output status

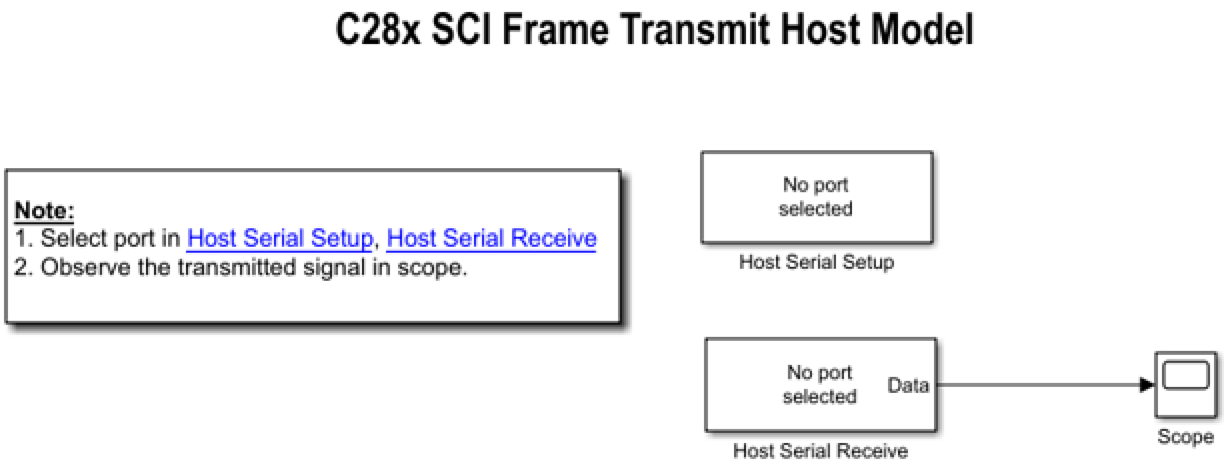
You will transmit data of frame size more than `1` at the rate determined by the ADC interrupt. The data fed to DAC is read by ADC which is then transmitted through SCI transmit block. The ADC then sends the interrupt after the conversion is completed. The start of conversion of ADC is triggered by ePWM1. The transmitted data is received in the host model at a slower rate `(0.03sec)`.

6. Click **Build, Deploy & Start** in the Hardware tab or press **Ctrl+B** to build and download the executable file.

Run on Host Model

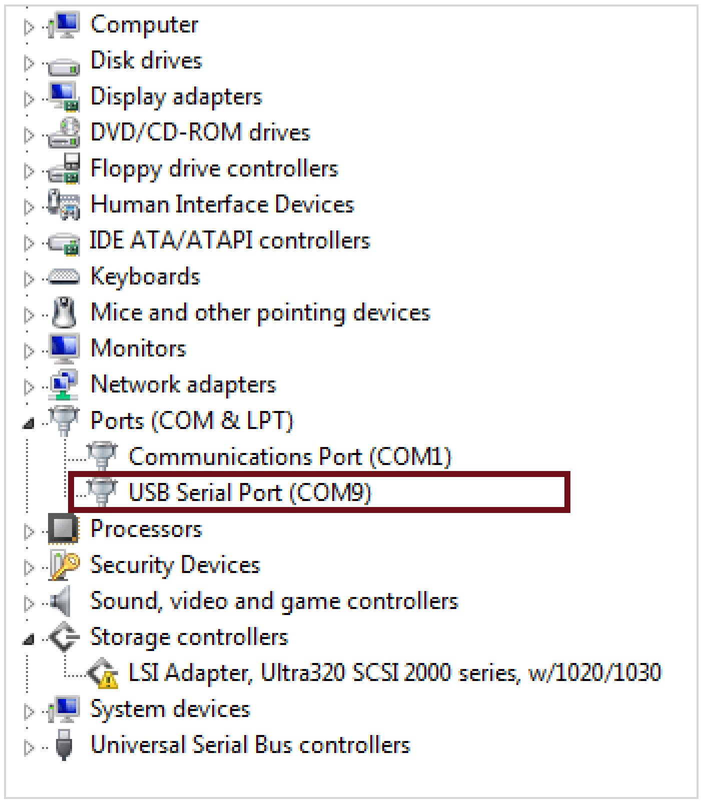
1. Open the host model.

```
open_system('c28xsciframehostmodel.slx');
```



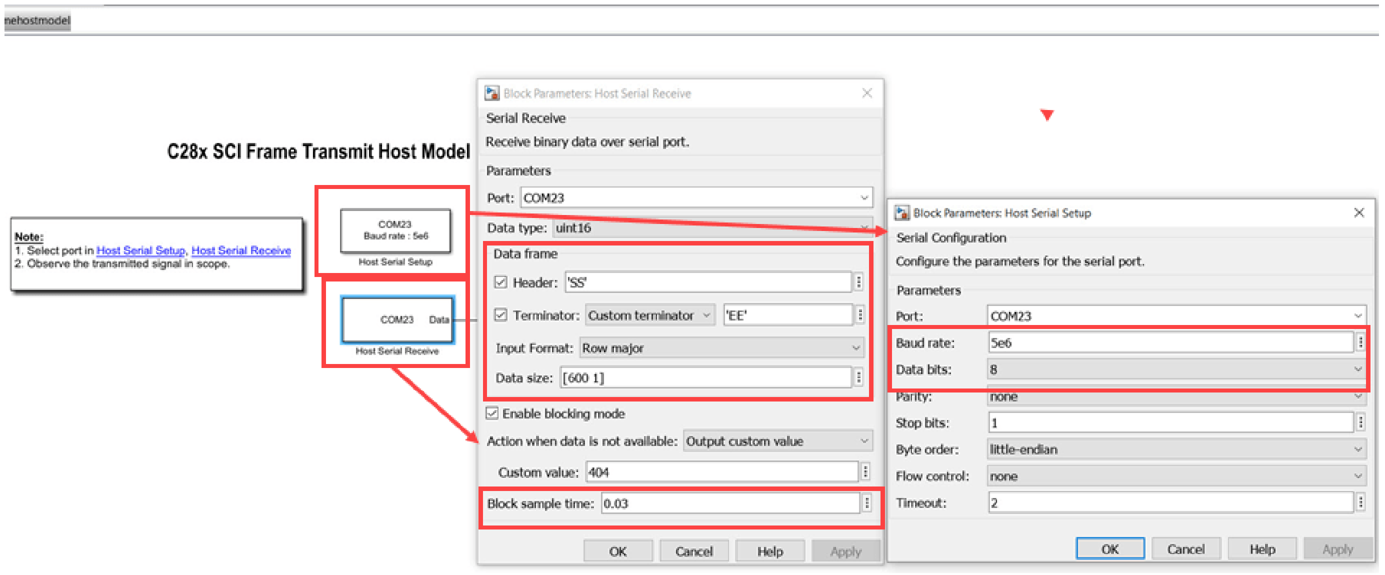
Note:
1. Select port in Host Serial Setup, Host Serial Receive
2. Observe the transmitted signal in scope.

2. To see the list of available COM ports on your computer, select **Start > Control Panel > Device Manager > Ports (COM & LPT)**.



- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Floppy drive controllers
- Human Interface Devices
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Ports (COM & LPT)
  - Communications Port (COM1)
  - USB Serial Port (COM9)
- Processors
- Security Devices
- Sound, video and game controllers
- Storage controllers
  - LSI Adapter, Ultra320 SCSI 2000 series, w/1020/1030
- System devices
- Universal Serial Bus controllers

3. Configure the device for **Host Serial Setup**, and **Host Serial Receive** blocks. Ensure that you set the **baud rate** to `5.625e6` in the Host Serial Setup block.



Note:
1. Select port in Host Serial Setup, Host Serial Receive
2. Observe the transmitted signal in scope.

Block Parameters: Host Serial Receive

Serial Receive

Receive binary data over serial port.

Parameters

Port: COM23

Data type: uint16

Data frame

☑ Header: 'SS'

☑ Terminator: Custom terminator 'EE'

Input Format: Row major

Data size: [600 1]

☑ Enable blocking mode

Action when data is not available: Output custom value

Custom value: 404

Block sample time: 0.03

Block Parameters: Host Serial Setup

Serial Configuration

Configure the parameters for the serial port.

Parameters

Port: COM23

Baud rate: 5e6

Data bits: 8

Parity: none

Stop bits: 1

Byte order: little-endian

Flow control: none

Timeout: 2

4. Click **Run**. Observe the data received on the Scope.

Other Things to Try

- Send and receive counter data using the Simulink® host model c2000_host_serial_comm while running c28x_sci_comm_interrupt on the target.
- Use the c2000HostSCICommunication MATLAB script to send and receive uint8 data while running c28x_sci_comm on the target.

More About

- C28x SCI Receive
- C28x SCI Transmit
- “Set Up Serial Communication with Target Hardware” on page 1-6

Code Verification and Validation with PIL

This example shows you how to use Texas Instruments™ C2000™ Processor for code verification and validation using PIL in C2000™ Microcontroller Blockset.

Introduction

In this example you will learn how to configure a Simulink® model to run Processor-In-the-Loop (PIL) simulation. In a PIL simulation, the generated code runs on the Texas Instruments C2000 processor. The results of the PIL simulation are transferred to Simulink to verify the numerical equivalence of the simulation and the code generation results. The PIL verification process is a crucial part of the development cycle to ensure that the behavior of the deployment code matches the design.

This example introduces how to configure a Simulink® model for code generation and verification using :

- **PIL Block**
- **Model Block PIL**
- **Top-Model PIL**

Required Hardware

To run this example you will need the following hardware: Texas Instruments C2000 processor based board with serial over USB capabilities

The Texas Instruments controlCard provides serial over USB capabilities. This allows serial communication from the target to your host computer. We will use this serial connection in this example to exchange data from Simulink to the target.

Some boards do not provide a FTDI chip and use the FTDI on the docking station and use the USB serial cable to establish a serial connection between the host computer and the target hardware. You can also use the COM1 port of your computer to establish an RS-232 serial connection with the board. See “Set Up Serial Communication with Target Hardware” on page 1-6 for details on establishing a serial connection between the target and the host computer.

Task 1 - Choose a Serial Communication Interface for PIL Simulation

The C2000™ Microcontroller Blockset supports serial communication interface for PIL over Serial.

After establishing a serial connection, find the COM port associated with the target hardware.

For more information on how to configure the Virtual COM port refer to this page. Note the COM port number of the USB Serial Port showing in your Windows Device Manager under Ports "(COM & LPT)"

1. Connect the target hardware to your host machine

2. Configure the Serial to run PIL. For more information, refer “Serial Configuration for External Mode and PIL”

Task 2 - Verify the Generated Code for a Subsystem Using a PIL Block

This example shows how to use a PIL block for subsystem code verification. With this approach:

- You can verify the code generated for a subsystem
- You must PIL block in the model as indicated by the comments in the model; make sure to avoid saving your model in this state as you would lose your original subsystem

1. Open the PIL Block model. This model is configured for the **TI Piccolo F2806x** target. You can configure the model to run on other TI C2000 processors or TI C2000 Concerto C28x core by changing the target hardware in the Configuration Parameters > Hardware Implementation pane. The objective here is to create a PIL block out of the **Controller** subsystem that you will run on the Texas Instruments C2000 processor.

Note: If you choose to select a different TI C2000 processors other than default configured model, ensure you set the **Total number of instances allowed per top model** to **Multiple** under **Configuration parameters** > **Model Referencing** > **Options for referencing this model**.

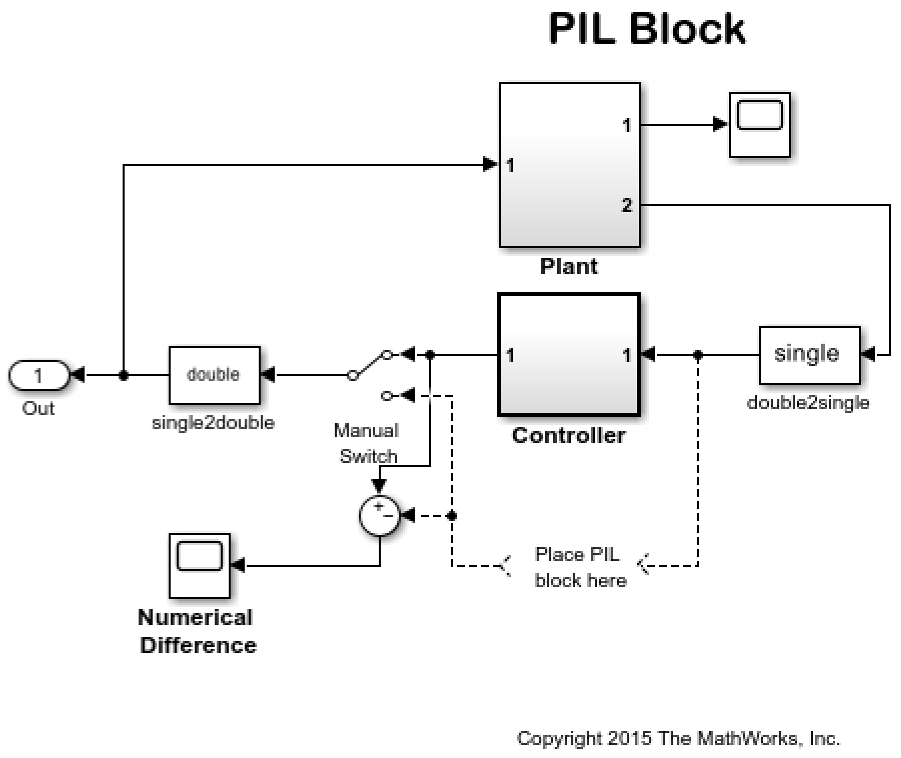
2. Choose a PIL communication interface by following the steps in Task 1 above.

3. Create a PIL block for the **Controller** subsystem as explained in “Test Generated Code with SIL and PIL Simulations” (Embedded Coder).

4. Place the PIL block created in the model as shown by the comments in the model.

5. Run the PIL simulation as explained in “Test Generated Code with SIL and PIL Simulations” (Embedded Coder).

6. You can switch between the original and PIL block subsystems by double clicking on the **Manual Switch** block. Double click on the **Numerical Difference** block to see the difference between the simulated **Controller** subsystem and the PIL block running on the target processor.



Task 3 - Verify Referenced Model Code Using PIL

This example shows how to verify the generated code for a referenced model by running a PIL simulation. With this approach:

- You can verify code generated for referenced models
- You can easily switch a Model block between normal and PIL simulation mode

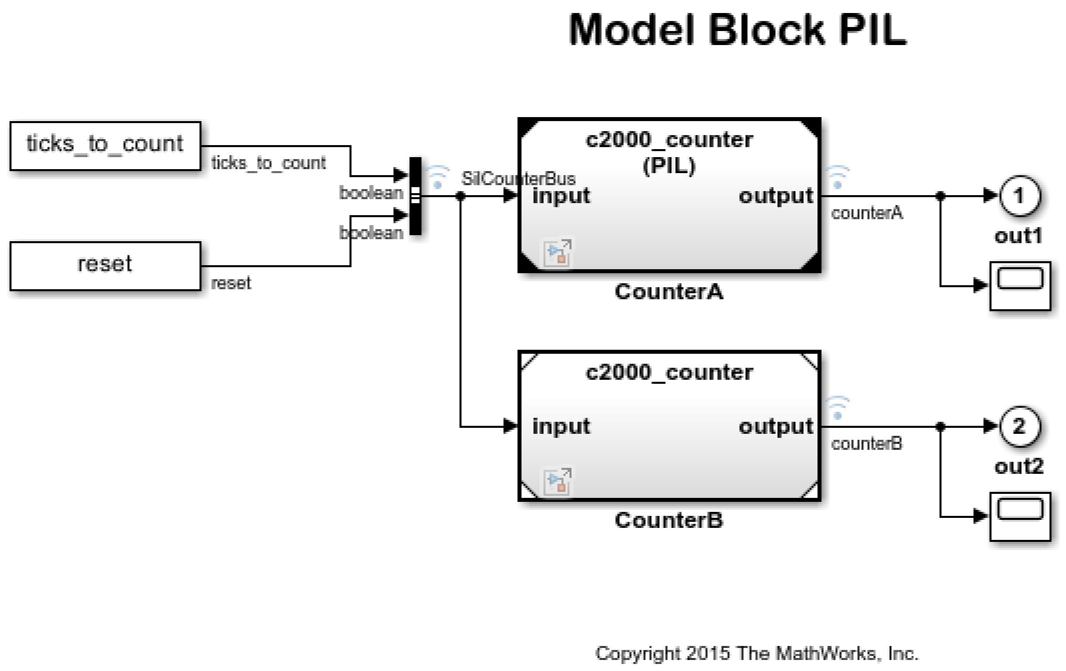
1. Open the Model Block PIL model. This model is configured for **TI Piccolo F2806x** target. To configure the model to run on other TI C2000 processors you can change the target hardware in the Configuration Parameters > Hardware Implementation pane. The model contains two Model blocks that both point at the same referenced model. You will configure one of the Model blocks to run in PIL simulation mode and the other in normal mode.

Note: If you choose to select a different TI C2000 processors other than default configured model, ensure you set the **Total number of instances allowed per top model** to **Multiple** under **Configuration parameters** > **Model Referencing** > **Options for referencing this model**.

2. Choose a PIL serial communication interface by following the steps in Task 1 above.

3. Configure and run **CounterA** Model block in PIL simulation as explained in “Test Generated Code with SIL and PIL Simulations” (Embedded Coder).

4. When the model starts running, **Scope1** displays the PIL simulation output running on the TI Piccolo F2806x processor while **Scope2** shows the normal mode simulation output.



Task 4 - Verify Top Model Code Using PIL

This example shows how to verify the generated code for a model by running a PIL simulation. With this approach:

- You can verify code generated for a top model
- You can easily switch the entire model between normal and PIL simulation mode

1. Open the Top Model PIL model. This model is configured for the **TI Piccolo F2806x** target.

Note: If you choose to select a different TI C2000 processors other than default configured model, ensure you set the **Total number of instances allowed per top model** to **Multiple** under **Configuration parameters > Model Referencing > Options for referencing this model**.

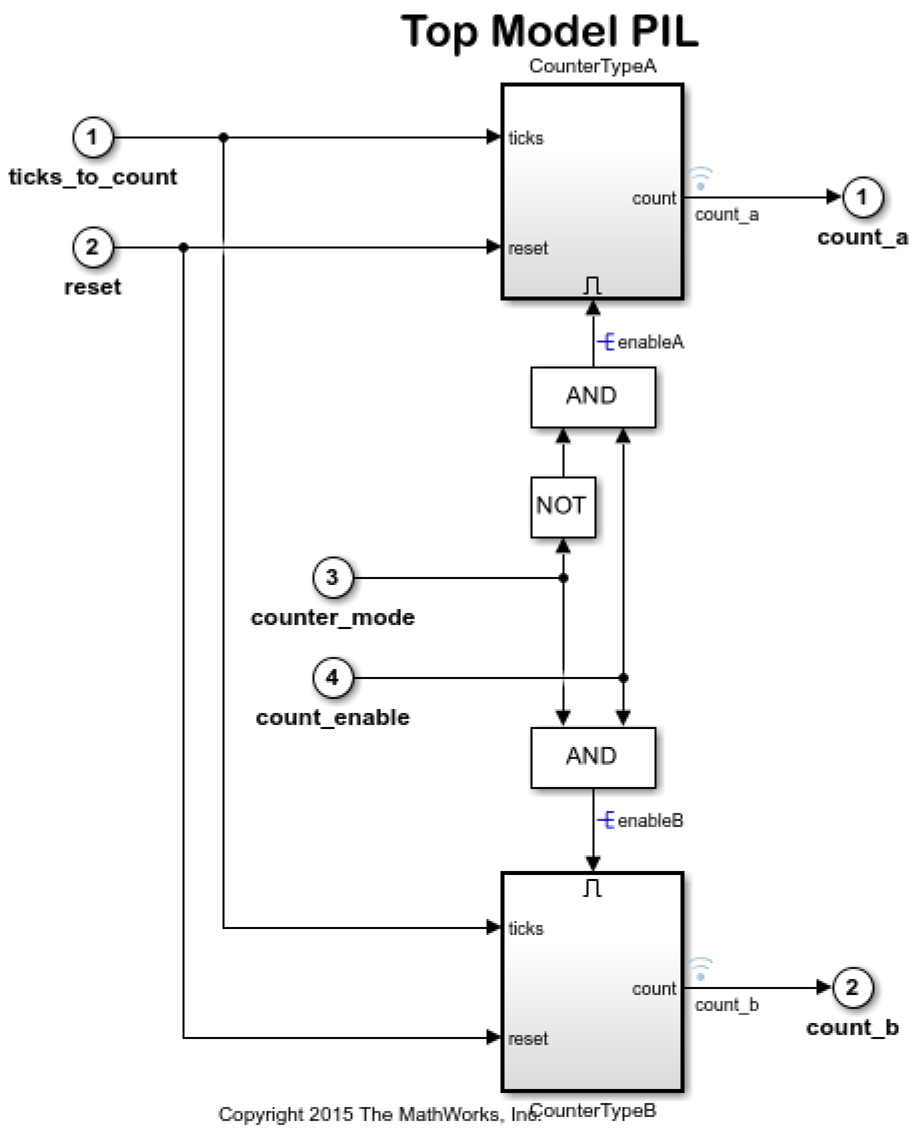
2. Choose a PIL serial communication interface by following the steps in Task 1 above.

3. Run the top model PIL simulation as explained in “Test Generated Code with SIL and PIL Simulations” (Embedded Coder).

4. When the PIL simulation is completed, a **logsOut** variable is created in the base workspace. The **logsOut** data contains PIL simulation results. You can access the logged data for signals **count_a** and **count_b** using the following commands:

- count_a = get(logsOut,'count_a');
- count_a.Values.Data

- count_b = get(logsOut,'count_b');
- count_b.Values.Data



Summary

This example introduced code verification workflows using PIL

Code Verification and Validation with External Mode

This example shows you how to use C2000™ Microcontroller Blockset for code verification and validation using External mode.

In this example you will learn how to configure a Simulink® model to run a simulation in External mode.

Simulink External mode feature enables you to accelerate the process of parameter tuning by letting you change certain parameter values while the model is running on target hardware, without stopping the model. When you change parameter values from within Simulink, the modified parameter values are communicated to the target hardware immediately. The effects of the parameters tuning activity may be monitored by viewing algorithm signals on scopes or displays in Simulink.

Prerequisites

This example runs on the ARM Cortex M3 CPU of the Concerto processor. We recommend completing the "Getting Started with C2000 Microcontroller Blockset for F28M3x Concerto Processors" on page 4-287.

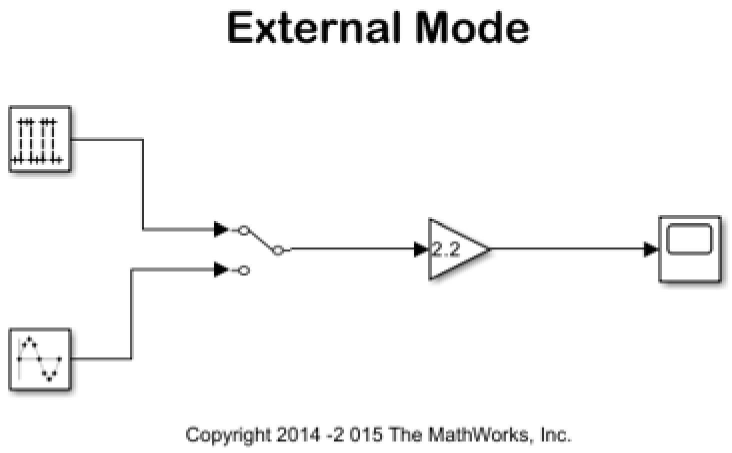
Required Hardware

To run this example you will need the following hardware:

- F28M36 Concerto Control Card or
- F28M35 Concerto Control Card

The Texas Instruments ControlCARDs provide serial over USB capabilities. This allows serial communication from the target to your host computer over the USB connection to the board. The controlCARDs also provide Ethernet capabilities. We will use both Ethernet and serial connections in this example to exchange data from Simulink to the target.

```
open_system('f28m35x_m3_external_mode');
```



Running the Model in External Mode

In this task, you will run a model in External mode. When you are prototyping and developing an algorithm, it is useful to monitor and tune the algorithm while it runs on hardware. The External mode feature in Simulink enables this capability.

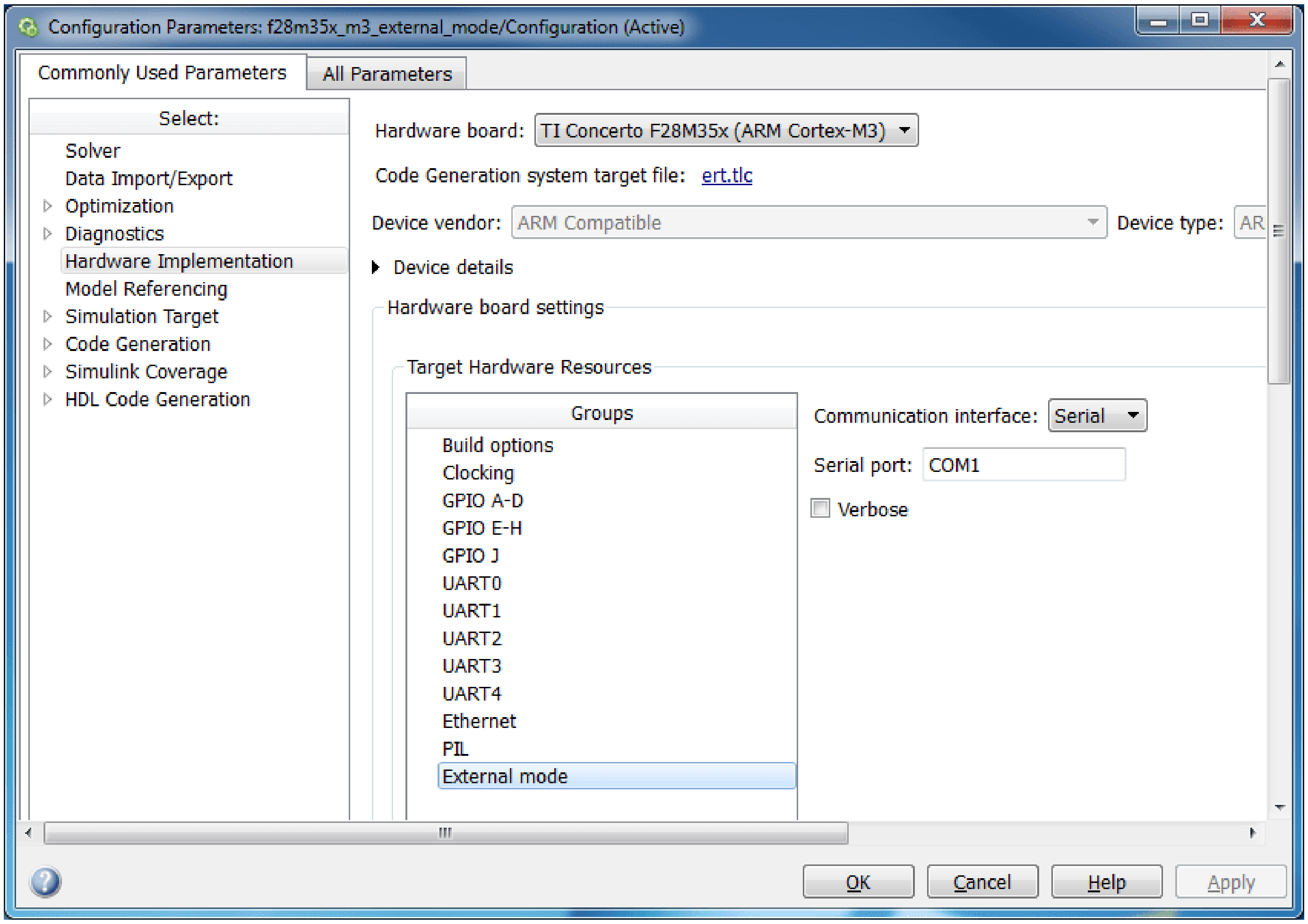
The blockset supports serial and TCP/IP communication interface for External Mode on the ARM Cortex M3 core. The serial communication interface uses UART0 for External Mode. The TCP/IP communication interface uses the Ethernet port provided on the C2000 Concerto controlCARD.

You can access UART0 via the USB cable connected to the controlCARD through a virtual COM port. For more information on how to configure the Virtual COM port refer to this page. Note the COM port number of the USB Serial Port showing in your Windows Device Manager under Ports "(COM & LPT)"

1. Open the External mode model. This model has already been configured for the **TI Concerto F28M35x (ARM Cortex-M3)** target. To configure the model to run on **TI Concerto F28M36x (ARM Cortex-M3)** you can change the Hardware board in the Configuration Parameters > Hardware Implementation pane.

2. Choose the communication interface by navigating through Configuration Parameters > Hardware Implementation > Target Hardware Resources > External Mode > Communication Interface.

3. If you choose Serial communication interface, enter the COM port number corresponding to your controlCARD. * As an example, see the settings in the External Mode model as shown below. In this example **Serial port** COM1 is selected.



Configuration Parameters: f28m35x_m3_external_mode/Configuration (Active)

Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

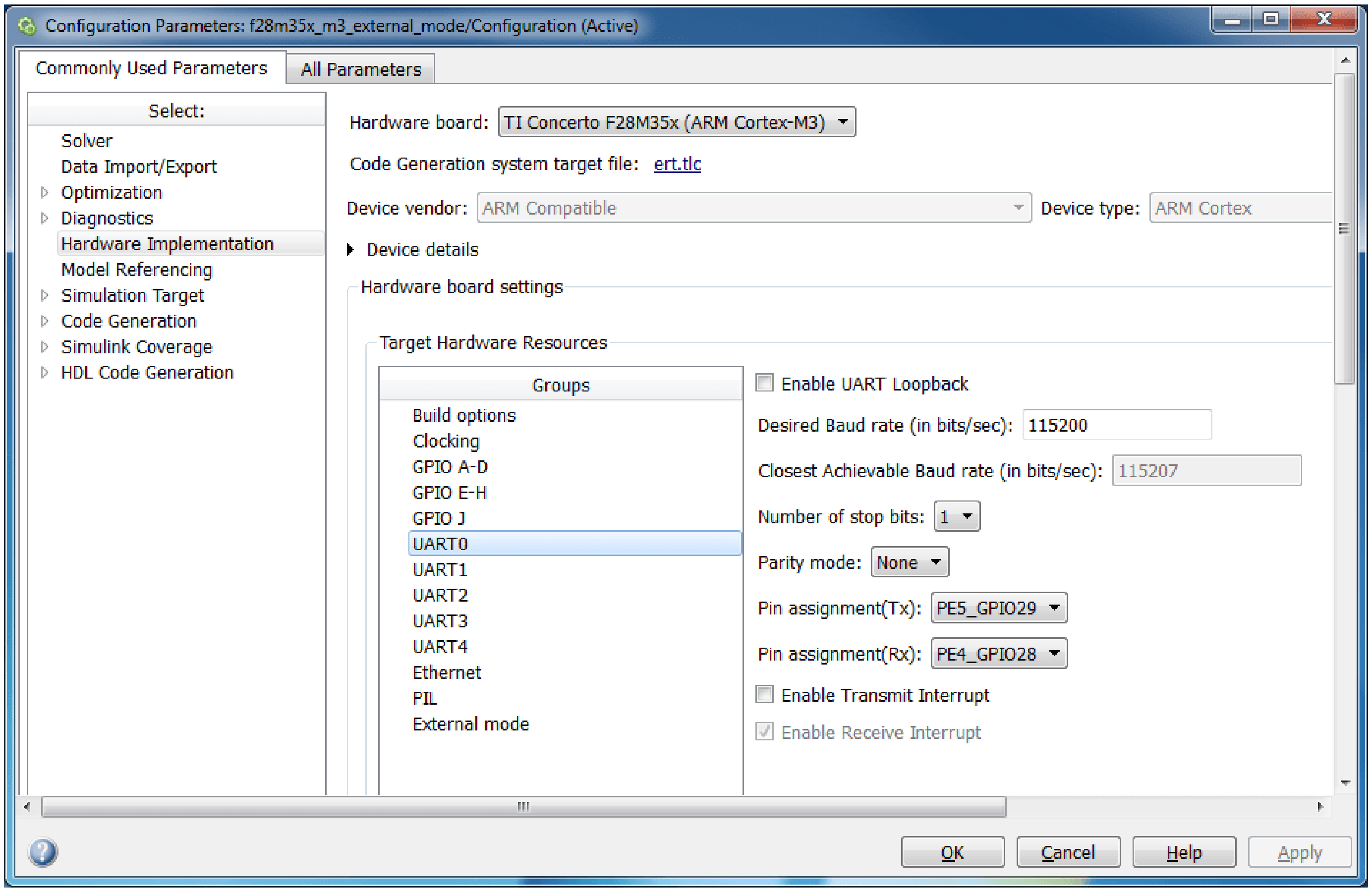
Device type: AR

Communication interface: Serial

Serial port: COM1

Verbose [ ]

- To set the serial baud rate, click on the **UART0** group in the configuration parameters and enter the desired baud rate as shown below.



Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible Device type: ARM Cortex

☐ Enable UART Loopback

Desired Baud rate (in bits/sec): 115200

Closest Achievable Baud rate (in bits/sec): 115207

Number of stop bits: 1

Parity mode: None

Pin assignment(Tx): PE5_GPIO29

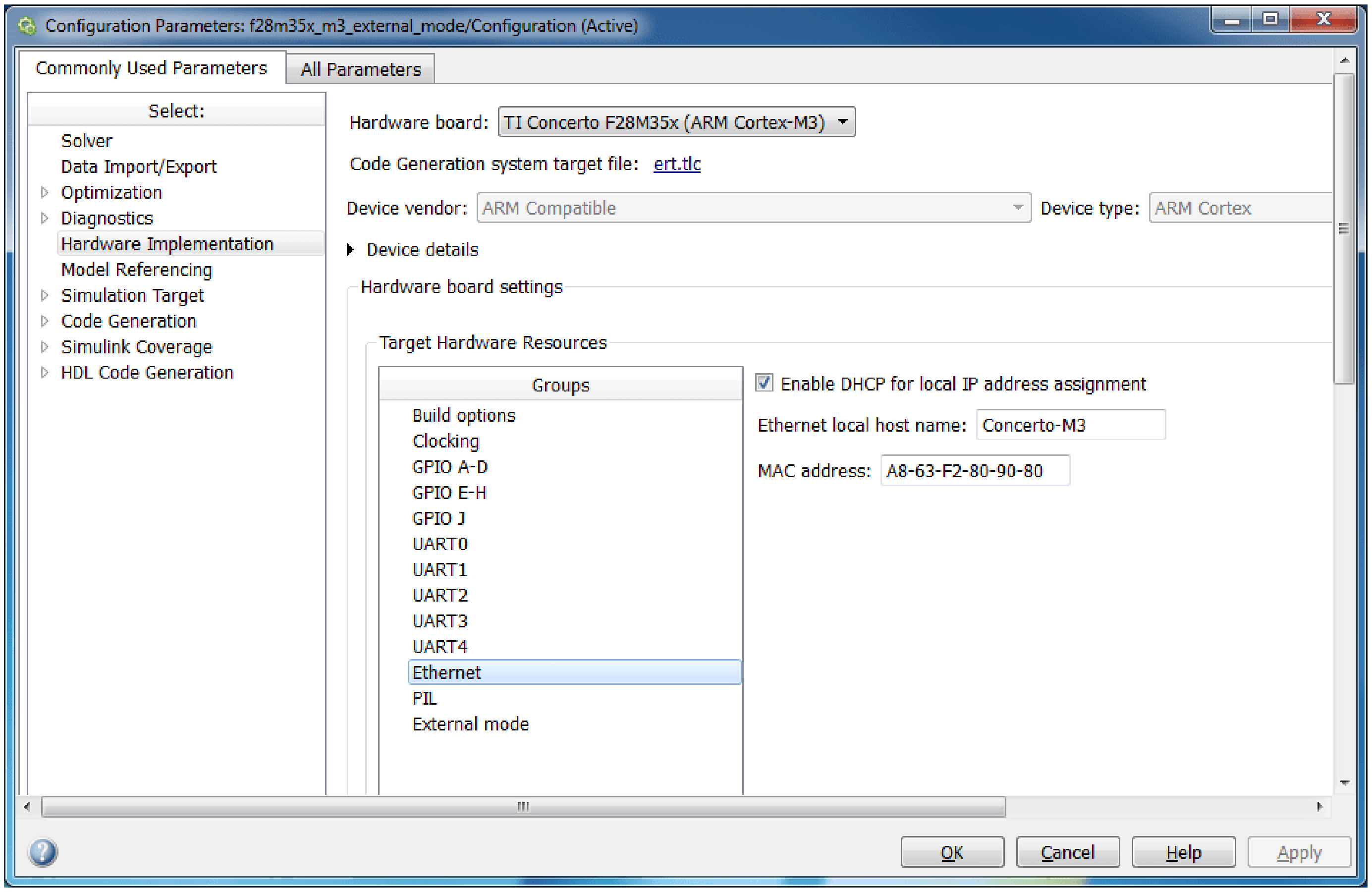
Pin assignment(Rx): PE4_GPIO28

☐ Enable Transmit Interrupt

☑ Enable Receive Interrupt

4. For External mode over Ethernet, the default Ethernet configuration uses DHCP for the target IP address assignment. To configure the target for Static IP address assignment, on the Ethernet tab, uncheck the **Enable DHCP for local IP address assignment** and enter the desired static IP address and subnet mask as shown below.

- The Ethernet communication interface uses port 17725.



Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

Device type: ARM Cortex

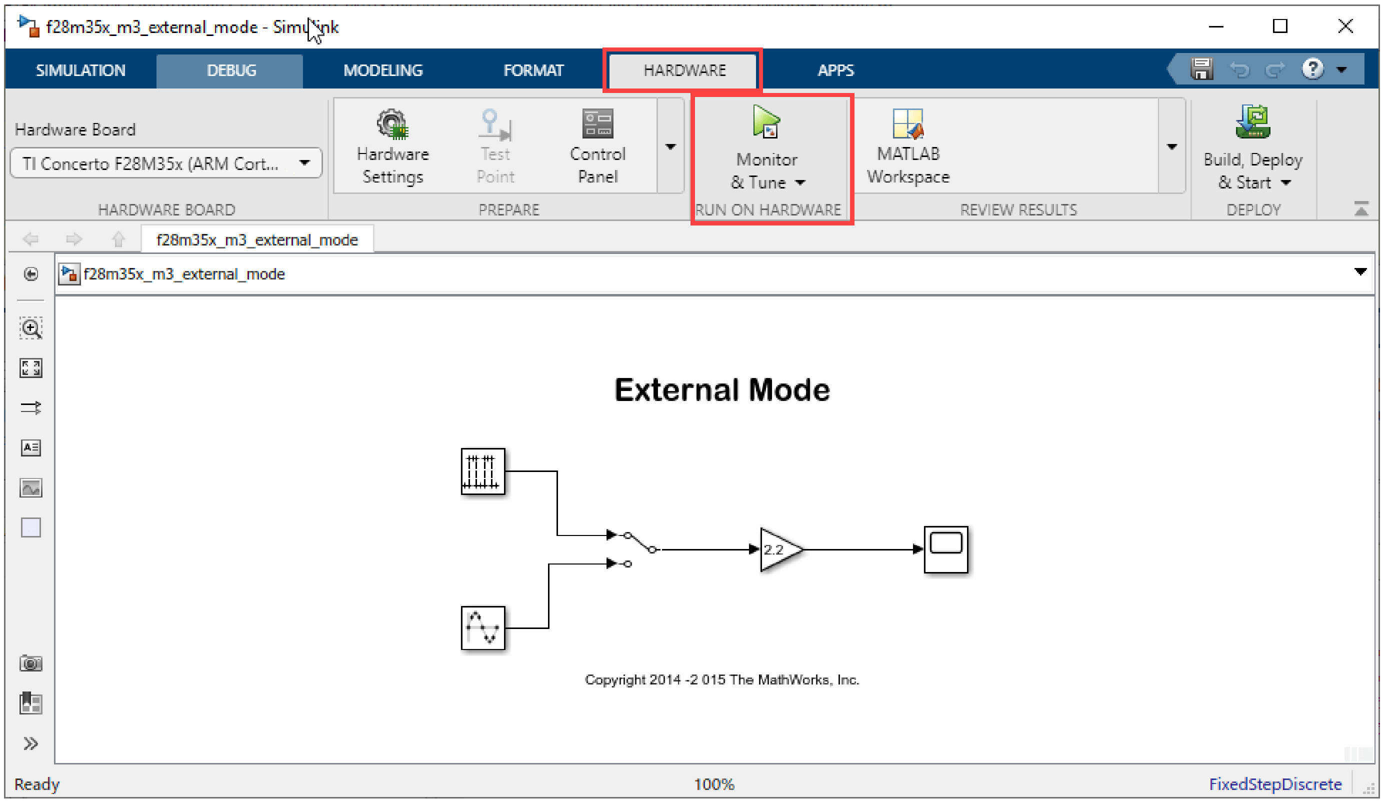
[✓] Enable DHCP for local IP address assignment

Ethernet local host name: Concerto-M3

MAC address: A8-63-F2-80-90-80

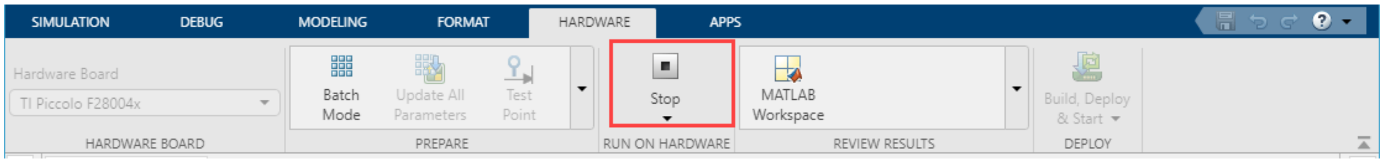
5. Click Apply and close the configuration parameters window.

6. Go to **Hardware** tab and click **Monitor & Tune**.



Wait for the model to build and load on the target. Once this is done, the External mode simulation begins. Double click the **Manual Switch** block while the simulation is running to change the input source. Double click on the **Gain** block to change the signal gain. Finally, double click on the **Scope** block to view the External mode simulation results. Note that the entire model is running on the target.

7. Stop External mode simulation:



Stopping External mode simulation terminates the execution of the code running on the F28M3x Concerto Processor. Before you can start another External mode simulation you need to run the generated code again.

Things to remember while setting up the model to run in External mode

- If running External Mode over UART0, ensure that the COM port number entered in the **Serial port** parameter at **Hardware Implementation** > **Target hardware resources** > **External mode** is correct.
- In the Model explorer, go to the Code > External Mode Control Panel and click on the 'Signal and Triggering' button. The default value of Duration parameter under Trigger Options Section is 10. You may need to change this value to 5 if the memory on the target is not enough to store 10 data points.

- The recommended baud rates for external mode communication over Serial is 115200 or 9600.

```
close_system('f28m35x_m3_external_mode', 0);
```

Code Verification and Validation with PIL on the ARM Cortex M3 Core of the F28M3x Concerto Processor

This example shows you how to use Texas Instruments C2000 F28M3x Concerto Processor for code verification and validation using PIL in C2000™ Microcontroller Blockset.

Introduction

In this example you will learn how to configure a Simulink® model to run Processor-In-the-Loop (PIL) simulations. In a PIL simulation, the generated code runs on the Texas Instruments C2000 Concerto processor. The results of the PIL simulation are transferred to Simulink® to verify the numerical equivalence of the simulation and the code generation results. The PIL verification process is a crucial part of the development cycle to ensure that the behavior of the deployment code matches the design.

This example introduces the Simulink code generation and verification workflow by showing you how to:

- Configure a Simulink model to run **Model Block PIL** simulations on the Texas Instruments C2000 Concerto ARM Cortex M3 Core.
- Configure a Simulink model to run **Top-Model PIL** simulations on the Texas Instruments C2000 Concerto ARM Cortex M3 Core.
- Configure a Simulink model to run **PIL Block** simulations on the Texas Instruments C2000 Concerto ARM Cortex M3 Core.

To run PIL on the C28x Core of Concerto, please refer to the “Code Verification and Validation with PIL” on page 4-139

Prerequisites

This example runs on the ARM Cortex M3 CPU of the Concerto processor. We recommend completing the “Getting Started with C2000 Microcontroller Blockset for F28M3x Concerto Processors” on page 4-287.

Required Hardware

To run this example you will need the following hardware:

- F28M36 Concerto Control Card or
- F28M35 Concerto Control Card

The Texas Instruments ControlCARDs provide serial over USB capabilities. This allows serial communication from the target to your host computer over the USB connection to the board. The controlCARDs also provide Ethernet capabilities. We will use both Ethernet and serial connections in this example to exchange data from Simulink to the target.

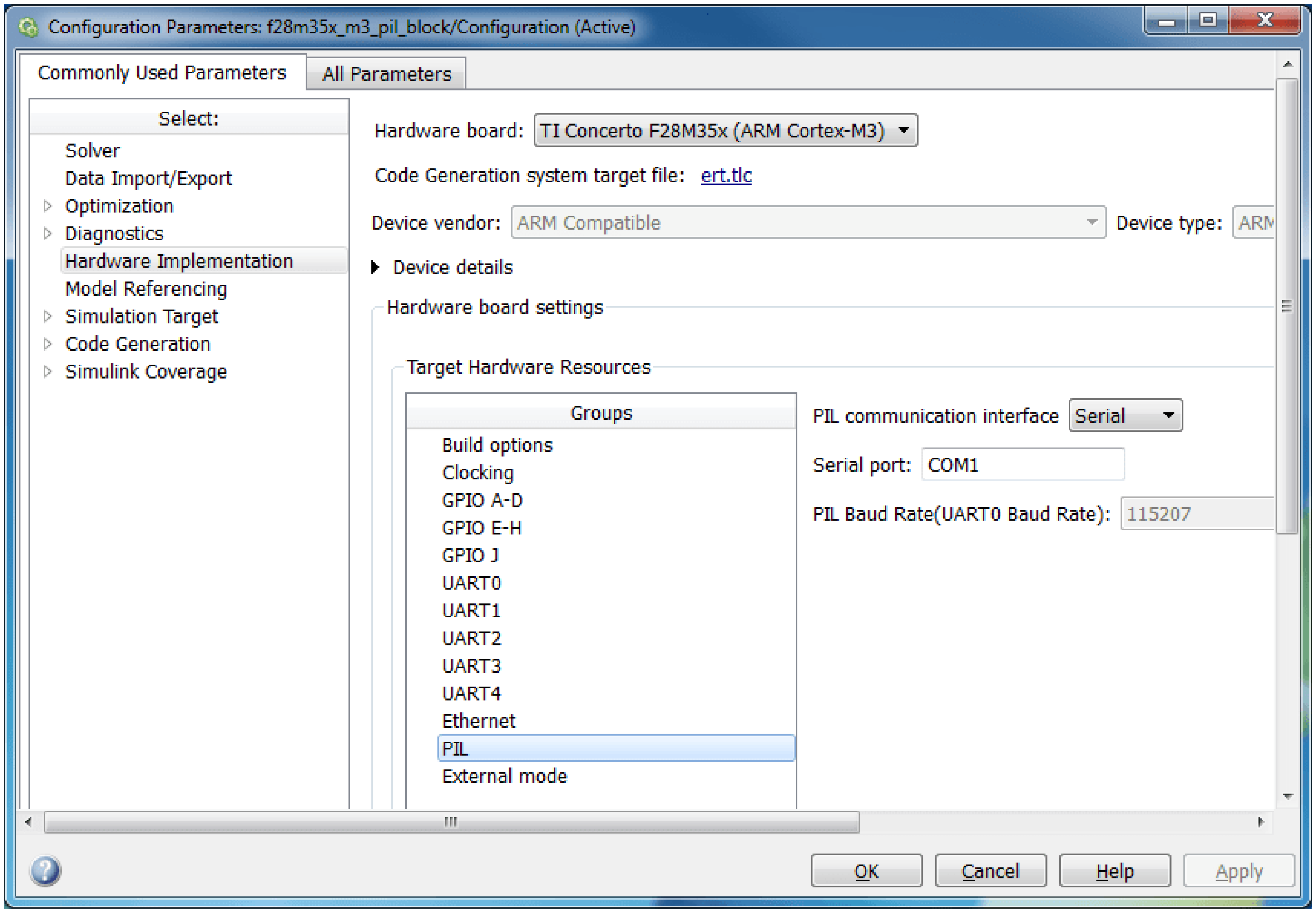
Choose a Communication Interface for PIL Simulation

The Texas Instruments C2000 Concerto processor supports serial and Ethernet communication interface for PIL. The Serial Communication interface uses UART0 and the Ethernet communication interface uses Ethernet port 17725 for PIL.

For Serial PIL: You can access UART0 via the USB cable connected to the controlCARD through a virtual COM port. For more information on how to configure the Virtual COM port refer to this page. Note the COM port number of the USB Serial Port showing in your Windows Device Manager under Ports "(COM & LPT)"

1. Connect the USB cable from your host machine to the controlCARD:

- Open your model configured for code generation on a Texas Instruments C2000 Concerto processor ARM Cortex M3 core. Browse to **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources** > **PIL** > **Serial Port** and enter the COM port number corresponding to your controlCARD.
- As an example, see the settings in the PIL Block model as shown below. For example, enter COM1 in the **Serial port** field


Configuration Parameters: f28m35x_m3_pil_block/Configuration (Active)

Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

Device type: ARM

Hardware board settings

Target Hardware Resources

PIL communication interface: Serial

Serial port: COM1

PIL Baud Rate(UART0 Baud Rate): 115207

For TCP/IP PIL:

1. Connect the USB cable from your host machine to the controlCARD:

- Open your model configured for code generation on a Texas Instruments C2000 Concerto processor ARM Cortex M3 core. By default this model is configured for Serial PIL over UART0. Go to Configuration Parameters > Hardware Implementation > Target Hardware Resources > PIL > PIL Communication Interface and change it to TCP/IP.

2. The default Ethernet configuration uses DHCP for Local IP Address Assignment. To configure for Static IP address assignment, In the Ethernet tab, uncheck the **Enable DHCP for local IP address assignment** and enter the desired Static IP Address and subnet mask.

Verify the Generated Code for a Subsystem Using a PIL Block

This example shows how to use a PIL block for subsystem code verification. With this approach:

- You can verify the code generated for a subsystem
- You must swap your original subsystem with a generated PIL block; make sure to avoid saving your model in this state as you would lose your original subsystem

1. Open the PIL Block model. This model is configured for the **TI Concerto F28M35x (ARM Cortex-M3)** target. To configure the model to run on **TI Concerto F28M36x (ARM Cortex-M3)** you can change the Hardware board in the Configuration Parameters > Hardware Implementation pane. The objective here is to create a PIL block out of the **Controller** subsystem that you will run on the Texas Instruments C2000 Concerto F28M35x processor ARM Cortex M3 core.

Note: If you choose to select a different TI C2000 processors other than default configured model, ensure you set the **Total number of instances allowed per top model** to **Multiple** under **Configuration parameters** > **Model Referencing** > **Options for referencing this model**.

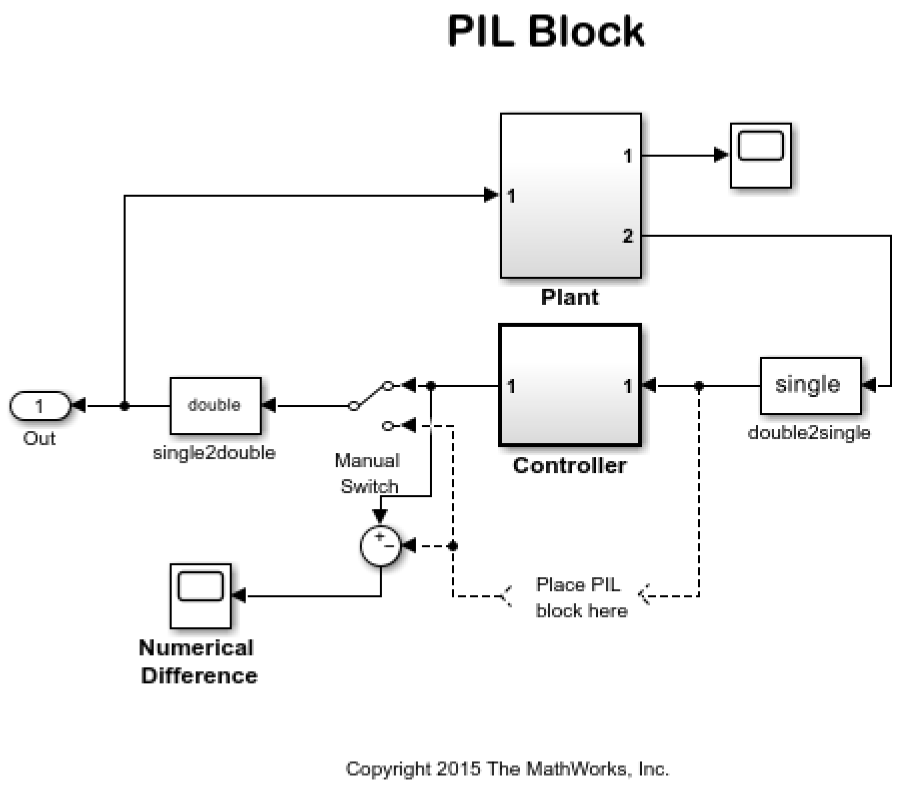
2. Choose a PIL communication interface by following the steps in Task 1 above.

3. Create a PIL block for the **Controller** subsystem by following Task Example 1 of the Verify Generated Code Using Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) Simulation.

4. Run the PIL simulation by following Example 1 of the Verify Generated Code Using Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) Simulation.

5. You can switch between the original and PIL block subsystems by double clicking on the **Manual Switch** block. Double click on the **Numerical Differences** block to see the difference between the simulated **Controller** subsystem and the PIL block running on the Texas Instruments C2000 Concerto board.

```
open_system('f28m35x_m3_pil_block');
```



Verify Referenced Model Code Using PIL

This example shows how to verify the generated code for a referenced model by running a PIL simulation. With this approach:

- You can verify code generated for referenced models
- You can easily switch a Model block between normal and PIL simulation mode

1. Open the Model Block PIL model. This model is configured for **TI Concerto F28M35x (ARM Cortex-M3)** target. To configure the model to run on **TI Concerto F28M36x (ARM Cortex-M3)** you can change the Hardware board in the Configuration Parameters > Hardware Implementation pane. The model contains two Model blocks that both point at the same referenced model. You will configure one of the Model blocks to run in PIL simulation mode and the other in normal mode.

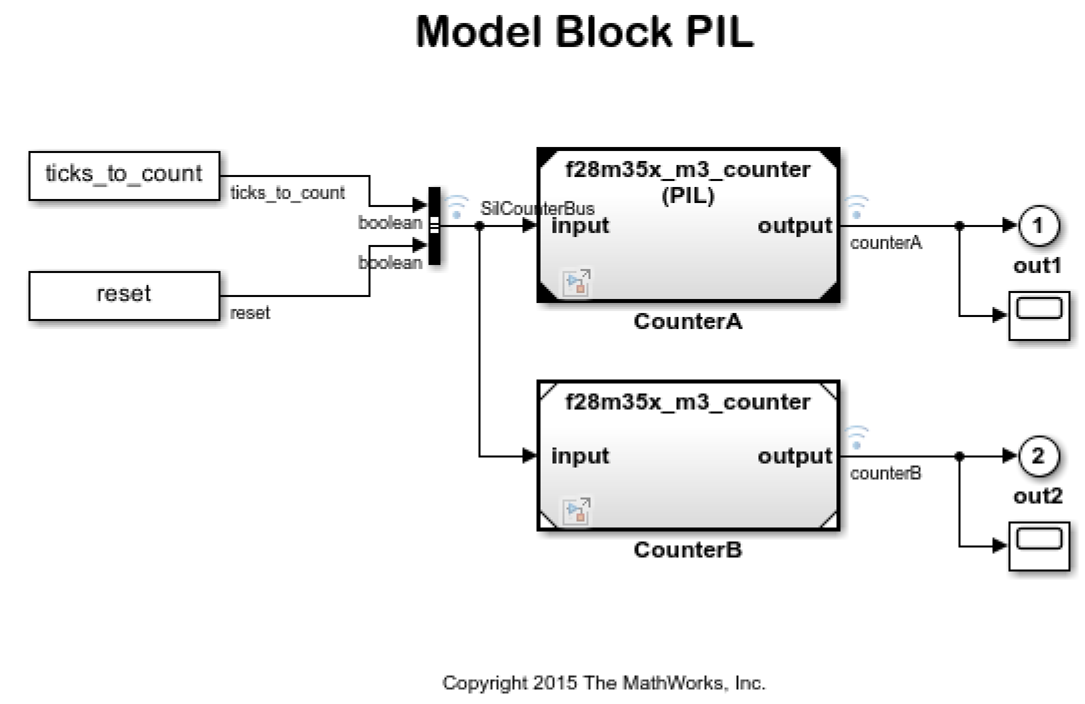
Note: If you choose to select a different TI C2000 processors other than default configured model, ensure you set the **Total number of instances allowed per top model** to **Multiple** under **Configuration parameters** > **Model Referencing** > **Options for referencing this model**.

2. Choose a PIL serial communication interface by following the steps in Task 1 above.

3. Configure and run **CounterA** Model block in PIL simulation mode by following Example 2 of the Verify Generated Code Using Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) Simulation.

4. When the model starts running, **Scope1** displays the PIL simulation output running on the Texas Instruments C2000 Concerto processor ARM Cortex M3 core while **Scope2** shows the normal mode simulation output.

```
open_system('f28m35x_m3_model_pil_block');
```



Verify Top Model Code Using PIL

This example shows how to verify the generated code for a model by running a PIL simulation. With this approach:

- You can verify code generated for a top model
- You can easily switch the entire model between normal and PIL simulation mode

1. Open the Top Model PIL model. This model is configured for the **TI Concerto F28M35x (ARM Cortex-M3)** target. To configure the model to run on **TI Concerto F28M36x (ARM Cortex-M3)** target, you can change the target hardware in the Configuration Parameters > Hardware Implementation pane.

Note: If you choose to select a different TI C2000 processors other than default configured model, ensure you set the **Total number of instances allowed per top model** to **Multiple** under **Configuration parameters** > **Model Referencing** > **Options for referencing this model**.

2. Choose a PIL serial communication interface by following the steps in Task 1 above.

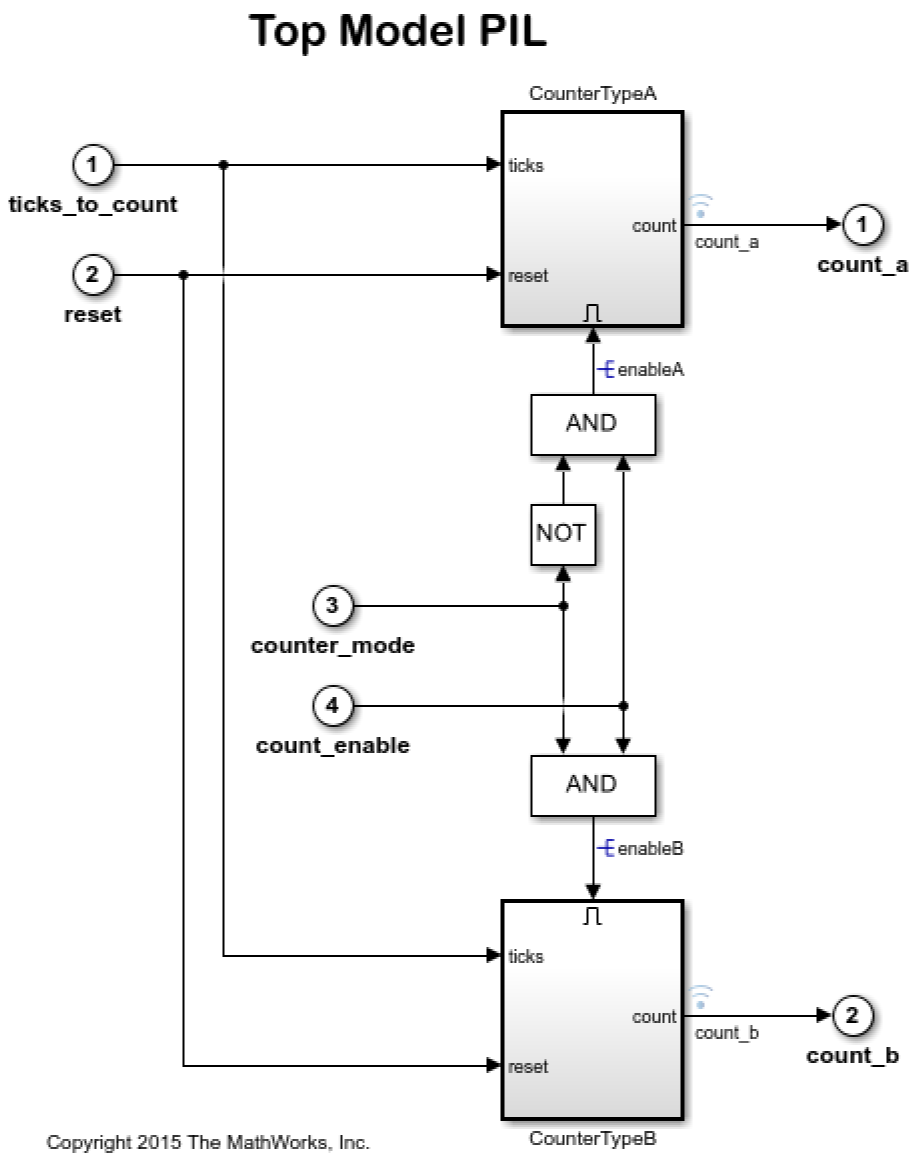
3. Run the top model PIL simulation by following Example 3 of the Verify Generated Code Using Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) Simulation.

4. When the PIL simulation is completed, a **logsOut** variable is created in the base workspace. The **logsOut** data contains PIL simulation results. You can access the logged data for signals **count_a** and **count_b** using the following commands:

- count_a = get(logsOut,'count_a');

- count_a.Values.Data
- count_b = get(logsOut,'count_b');
- count_b.Values.Data

```
open_system('f28m35x_m3_top_model_pil');
```



Real-Time Code Execution Profiling

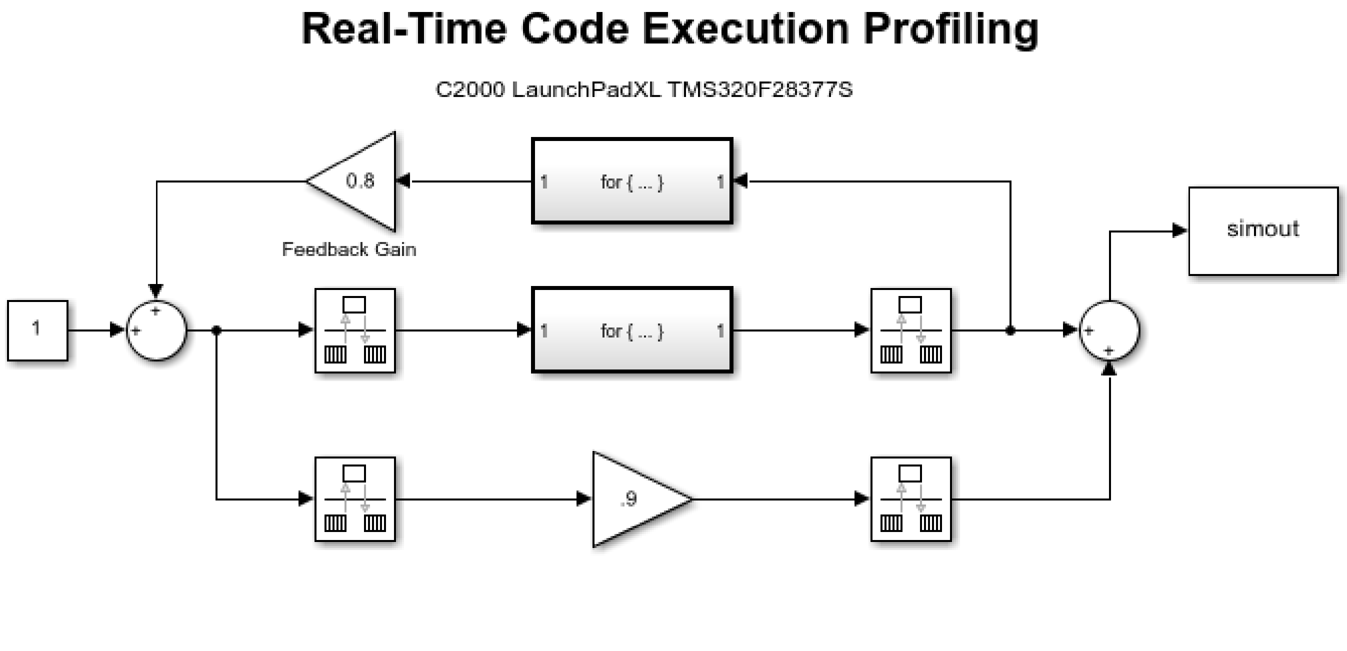
This example shows you how to use C2000™ Microcontroller Blockset for real-time execution profiling of generated code.

Available Versions of This Model:

- C2000 LaunchPadXL TMS320F28377S (LAUNCHXL-F28377S): f28377S_RTProfiler.slx

Model

The following figure shows the example model:


Real-Time Code Execution Profiling
C2000 LaunchPadXL TMS320F28377S
0.8
Feedback Gain
for { ... }
for { ... }
1
.9
simout

Introduction

Sample times specified in a Simulink® model determine the time schedule for running generated code on target hardware. If the target hardware has sufficient computing power, the code runs according to specified sample times in real-time.

You can use real-time execution profiling to check whether generated code meets real-time performance requirements. Execution Profiling results can also be used to take actions to enhance design of your system. For example, if the code easily meets the real-time requirements, you may consider adding more functionality to your system to exploit available processing power. If, on the other hand, the code does not meet real-time requirements, you may look for ways to reduce execution time. For example, you can identify the tasks that require the most time and then investigate whether trade-off between functionality and speed is possible.

This example introduces a workflow for real-time code execution profiling by showing you how to:

- Configure the model for code execution profiling, and generate code.
- Run generated code on target hardware.
- Analyze performance through code execution profiling plots and reports.

To Run the Example on the Board

1. Open the model. This model is configured for **C2000 LaunchPadXL TMS320F28377S (LAUNCHXL-F28377S)** target hardware.

2. Open the **Modeling** tab and click **Model Settings** or press **Ctrl+E** to open Configuration Parameters dialog box. Go to **Code Generation** > **Verification**.

3. Select **Measure task execution time**, and select **Measure function execution times** > **Detailed (all function call sites)** option. These options enable you to profile execution time for each task and function in generated code. Click **OK**.

4. Check that **Hardware Implementation** > **Build Options** > **Build, load and run** is selected. This option must be selected, or the generated executable does not download into the target.

5. Set the number of profiling samples to be collected on the target using this command.

```
codertarget.tic2000.setExecutionProfileBufferLength(<modelName>, 100)
```

6. Open the **Hardware** tab and click **Build, Deploy & Start** to run the code inside the target hardware.

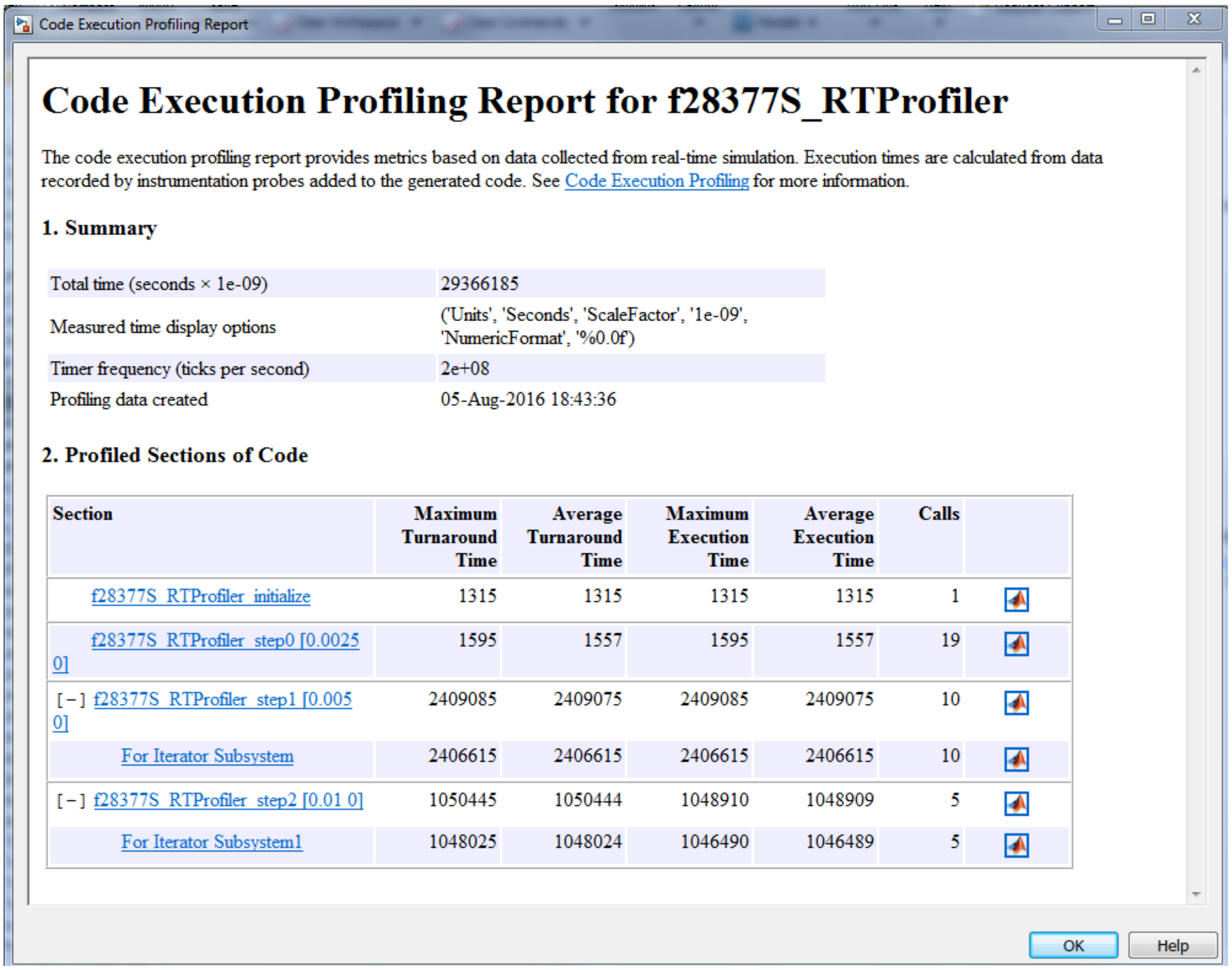
7. Run the following code in MATLAB® Command Window to get the profiling data into MATLAB workspace:

```
codertarget.profile.getData('f28377S_RTProfiler')
```

The variable **executionProfile** is available in the MATLAB workspace at the end of execution of this command.

8. Run the following code in MATLAB Command Window to obtain the **Profiling Report** for the session you just ran. Analyze the report on different turnaround and execution times for each task and function. Close the report when you are done.

```
executionProfile.report
```



Code Execution Profiling Report

Code Execution Profiling Report for f28377S_RTProfiler

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

1. Summary

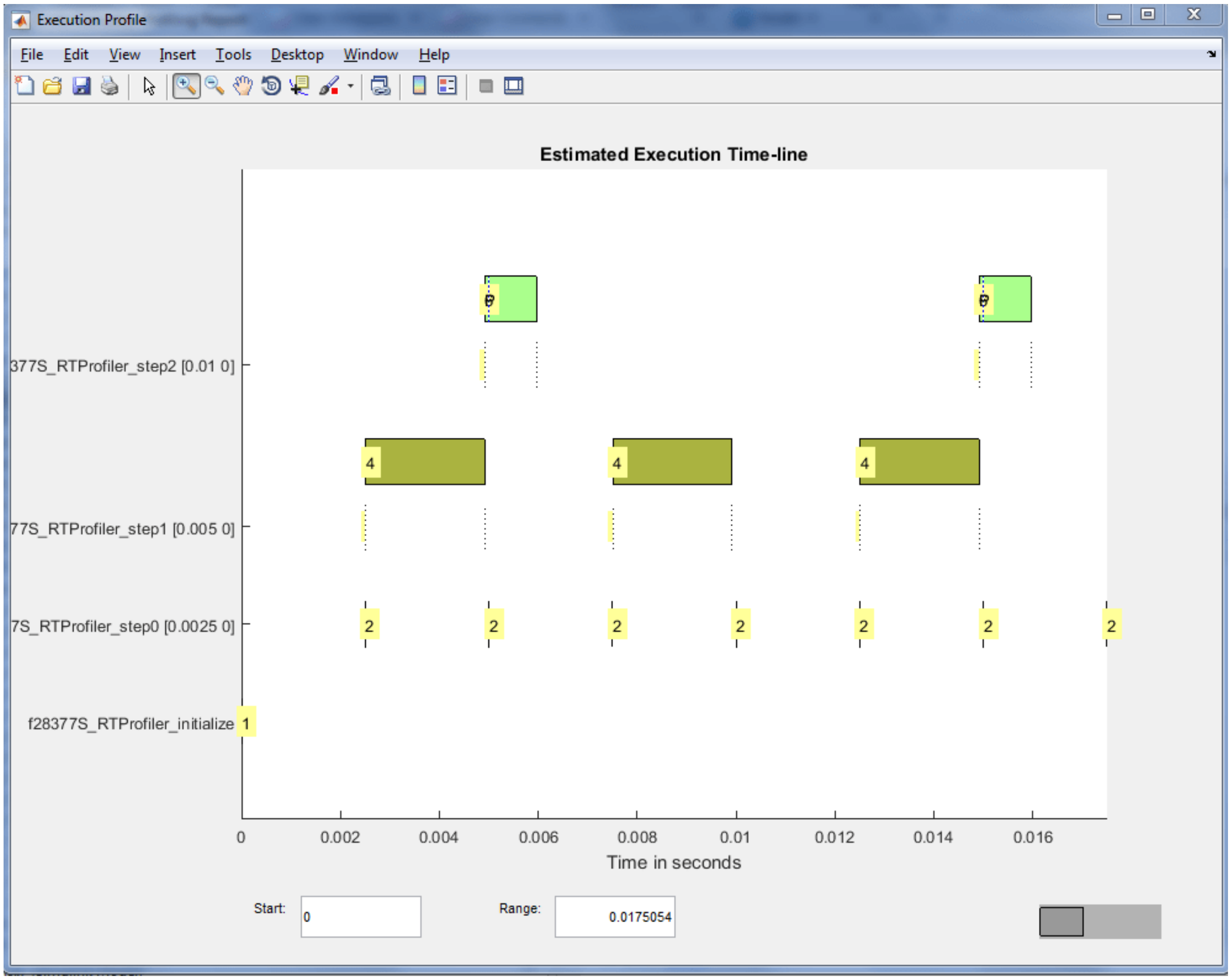
Total time (seconds × 1e-09)	29366185
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	2e+08
Profiling data created	05-Aug-2016 18:43:36

2. Profiled Sections of Code

Section	Maximum Turnaround Time	Average Turnaround Time	Maximum Execution Time	Average Execution Time	Calls	
f28377S_RTProfiler_initialize	1315	1315	1315	1315	1	
f28377S_RTProfiler_step0 [0.0025 0]	1595	1557	1595	1557	19	
[−] f28377S_RTProfiler_step1 [0.005 0]	2409085	2409075	2409085	2409075	10	
For Iterator Subsystem	2406615	2406615	2406615	2406615	10	
[−] f28377S_RTProfiler_step2 [0.01 0]	1050445	1050444	1048910	1048909	5	
For Iterator Subsystem1	1048025	1048024	1046490	1046489	5	

9. Execute the following code in MATLAB Command Window to obtain the **Profiling Timeline** for the session you just ran. Analyze the execution timeline of different tasks and functions. Notice where the faster task pre-empts the slower one and where different functions start and end. Close the timeline when you are done.

```
executionProfile.timeline
```


Execution Profile
File Edit View Insert Tools Desktop Window Help
Estimated Execution Time-line
377S_RTProfiler_step2 [0.01 0]
77S_RTProfiler_step1 [0.005 0]
7S_RTProfiler_step0 [0.0025 0]
f28377S_RTProfiler_initialize
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
Time in seconds
Start: 0
Range: 0.0175054

Notes

- The code execution profiler uses an on-chip timer. If you use a processor simulator, select one that can simulate processor timers.
- The model needs to run long enough to collect sufficient profiling data. This time depends on the number of profiling samples specified and the sample rates of the model. If you use a simulator, data collection can take considerably longer than when you use hardware.
- The shipped target configuration file (ccxml) removes RAM initializations done by the debugger while connecting to the target. Please make sure of this behavior on using a custom target configuration file.
- For asynchronously executed subsystems, code execution profiling is supported only for Hardware Interrupt and Idle Task blocks.
- Profiling probe functions add the finite amount of time in actual execution time.

- Once the profiling data is extracted, the execution will stop running in the hardware. You need to power cycle the board or flash the software from simulink to get it running.

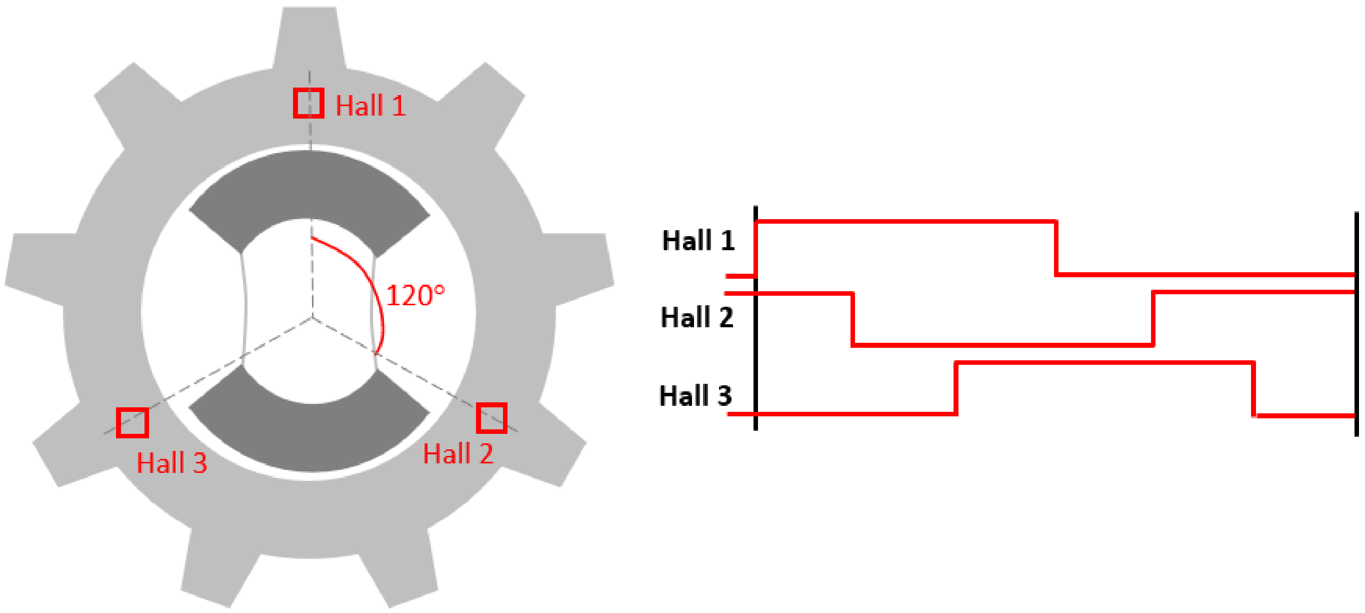
See Also

Field-Oriented Control of PMSM with Hall Sensor Using C2000 Processors

This example implements the field-oriented control (FOC) technique to control the speed of a three-phase permanent magnet synchronous motor (PMSM). The FOC algorithm requires rotor position feedback, which is obtained by a Hall sensor. For details about FOC, see “Field-Oriented Control (FOC)” (Motor Control Blockset).

A closed-loop FOC algorithm is used to regulate the speed and torque of a three-phase PMSM. This example uses C28x peripheral blocks from the C2000™ Microcontroller Blockset and MCB library blocks from Motor Control Blockset™.

This example uses the Hall sensor to measure the rotor position. A Hall effect sensor varies its output voltage based on the strength of the applied magnetic field. A PMSM consists of three Hall sensors located electrically 120 degrees apart. A PMSM with this setup can provide six valid combinations of binary states (for example, 001,010,011,100,101, and 110). The sensor provides the angular position of the rotor in the multiples of 60 degrees, which the controller uses to compute the angular velocity. The controller can then use the angular velocity to compute an accurate angular position of the rotor.



Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- F28035 control card + DRV8312-C2-KIT inverter: mcb_pmsm_foc_hall_f28035
- F28335 control card + DRV8312-C2-KIT inverter: mcb_pmsm_foc_hall_f28335
- LAUNCHXL-F280049C controller + BOOSTXL-DRV8305 inverter: mcb_pmsm_foc_hall_f280049

- F28069m control card + TMDSHVMTRINSPIN: mcb_pmsm_foc_hall_hvkit_f28069m
- Three-phase PMSM with optional HALL sensors attached to connector J4 of the DRV8312-C2-KIT board or connector J12 of LAUNCHXL-F280049C.
- For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Field-Oriented Control of PMSM Using Hall Sensor” (Motor Control Blockset).

For connections related to the preceding hardware configuration, see “Hardware Connections” on page 1-81.

Available Models

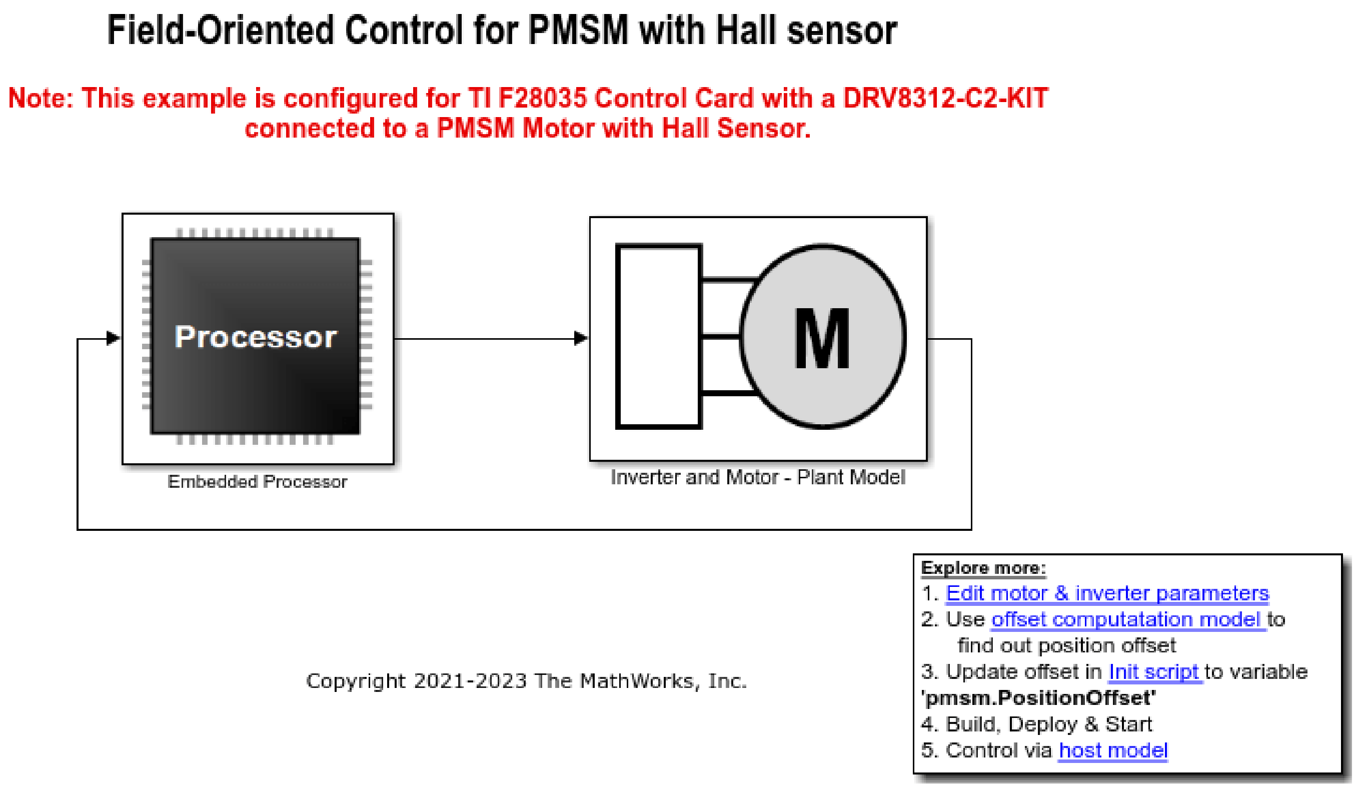
The example includes these models:

- mcb_pmsm_foc_hall_f28035
- mcb_pmsm_foc_hall_f28335
- mcb_pmsm_foc_hall_f280049C
- mcb_pmsm_foc_hall_hvkit_f28069m

Note: For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Field-Oriented Control of PMSM Using Hall Sensor” (Motor Control Blockset).

You can use these models for both simulation and code generation. You can also use the open_system command to open the Simulink® models. For example, use this command for a F28035 based controller.

```
open_system('mcb_pmsm_foc_hall_f28035.slx');
```



Field-Oriented Control for PMSM with Hall sensor

Note: This example is configured for TI F28035 Control Card with a DRV8312-C2-KIT connected to a PMSM Motor with Hall Sensor.

Copyright 2021-2023 The MathWorks, Inc.

Explore more:
1. Edit motor & inverter parameters
2. Use offset computatation model to find out position offset
3. Update offset in Init script to variable 'pmsm.PositionOffset'
4. Build, Deploy & Start
5. Control via host model

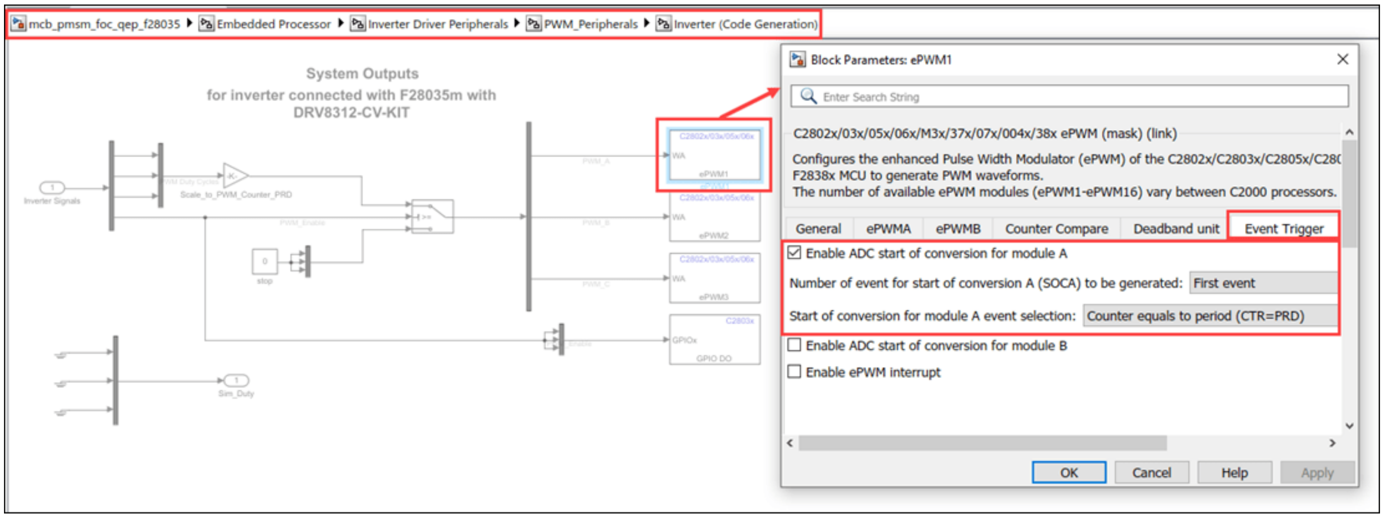
You may need to change the model parameters to fit your specific motor. Match motor voltage and power characteristics to the controller.

A conventional voltage-source inverter drives motor. The controller algorithm generates six pulse width modulation (PWM) signals using a vector PWM technique for six power switching devices. The inverter measures the current of the two motor inputs (ia and ib) input currents of the motor (ia and ib) using two analog-to-digital converters (ADCs) and sends the measurements to the processor.

Peripheral Block Configurations

Set the peripheral block configurations for this model. Double-click on the blocks to open block parameter configurations. You can use the same parameter values if you want to run this example for other hardware boards.

- **ePWM Block configuration**


mcb_pmsm_foc_qep_f28035 ▸ Embedded Processor ▸ Inverter Driver Peripherals ▸ PWM_Peripherals ▸ Inverter (Code Generation)
System Outputs
for inverter connected with F28035m with
DRV8312-CV-KIT
Inverter Signals
Scale_to_PWM_Counter_PRD
stop
Sim_Duty
ePWM1
ePWM2
ePWM3
GPIO DO

Block Parameters: ePWM1

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the enhanced Pulse Width Modulator (ePWM) of the C2802x/C2803x/C2805x/C28
F2838x MCU to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

☑ Enable ADC start of conversion for module A

Number of event for start of conversion A (SOCA) to be generated: First event

Start of conversion for module A event selection: Counter equals to period (CTR=PRD)

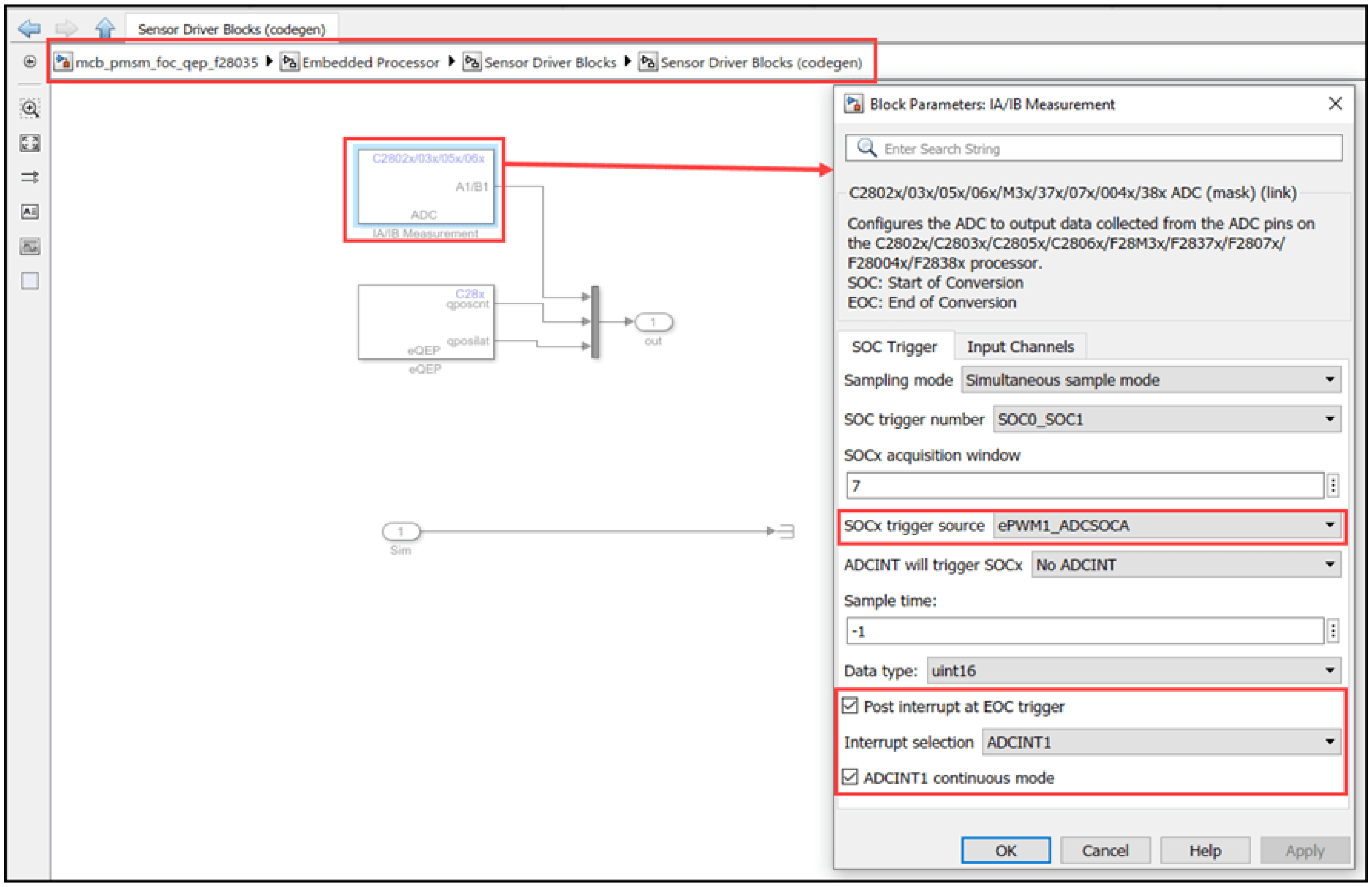
☐ Enable ADC start of conversion for module B

☐ Enable ePWM interrupt

OK | Cancel | Help | Apply

- **ADC Block configuration**

The algorithm in this example uses an asynchronous scheduling. The pulse width modulation (PWM) block triggers the ADC conversion. At the end of conversion, the ADC posts an interrupt that triggers the main FOC algorithm. For more information, refer “ADC Interrupt Based Scheduling” on page 1-12.


Sensor Driver Blocks (codegen)
mcb_pmsm_foc_qep_f28035 ▸ Embedded Processor ▸ Sensor Driver Blocks ▸ Sensor Driver Blocks (codegen)
C2802x/03x/05x/06x
A1/B1
ADC
IA/IB Measurement
C28x
qposcnt
qposlat
eQEP
eQEP
out
Sim

Block Parameters: IA/IB Measurement

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ADC (mask) (link)

Configures the ADC to output data collected from the ADC pins on the C2802x/C2803x/C2805x/C2806x/F28M3x/F2837x/F2807x/F28004x/F2838x processor.
SOC: Start of Conversion
EOC: End of Conversion

SOC Trigger | Input Channels

Sampling mode: Simultaneous sample mode

SOC trigger number: SOC0_SOC1

SOCx acquisition window: 7

SOCx trigger source: ePWM1_ADCSOCA

ADCINT will trigger SOCx: No ADCINT

Sample time: -1

Data type: uint16

☑ Post interrupt at EOC trigger

Interrupt selection: ADCINT1

☑ ADCINT1 continuous mode

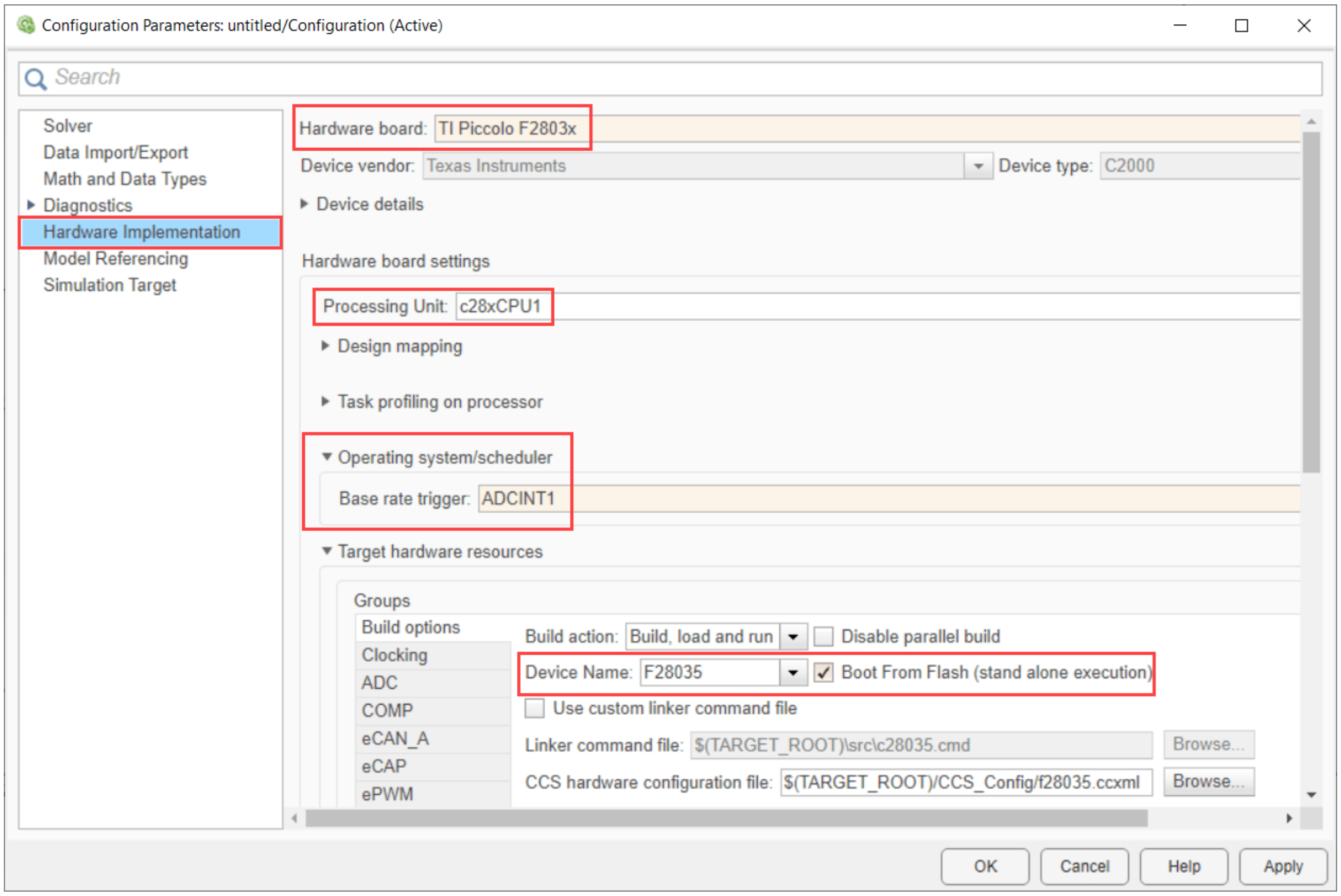
OK | Cancel | Help | Apply

Configure the Model

1. Open the mcb_pmsm_foc_hall_f28035 model. This model is configured for TI Piccolo F2803x hardware.

2. To run the model on other TI C2000 processors, first press **Ctrl+E** to open the Configuration Parameters dialog box. Then, select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

3. The following screenshots show the scheduler configurations performed in the model. You can use the same parameter values if you want to run this example for other hardware boards.


Configuration Parameters: untitled/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target

Hardware board: TI Piccolo F2803x
Device vendor: Texas Instruments
Device type: C2000
Device details

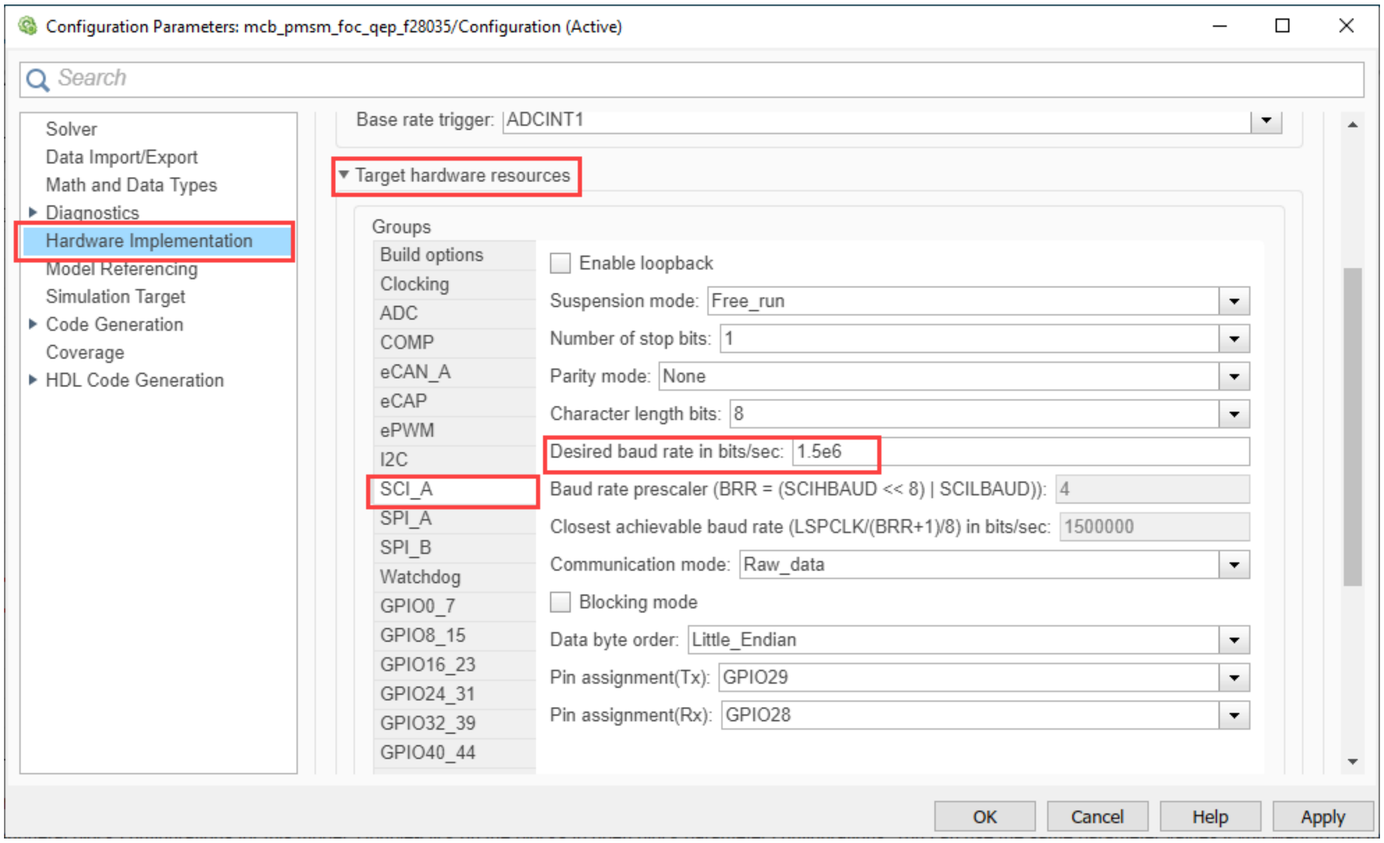
Hardware board settings
Processing Unit: c28xCPU1
Design mapping
Task profiling on processor
Operating system/scheduler
Base rate trigger: ADCINT1
Target hardware resources
Groups
Build options
Clocking
ADC
COMP
eCAN_A
eCAP
ePWM

Build action: Build, load and run
Disable parallel build
Device Name: F28035
[✓] Boot From Flash (stand alone execution)
Use custom linker command file
Linker command file: $(TARGET_ROOT)\src\c28035.cmd
CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28035.ccxml

Note:

- Sampling rate of the ADC block should be same as the base rate of the model determined by the PWM period of the ePWM block.
- Base rate trigger selection should be same as the interrupt triggered by the ADC module. For more information, see Model Configuration Parameters for Texas Instruments C2000 Processors.
- Ensure that the **Default parameter behavior** is set to **Inlined** (Configuration Parameters > Code Generation > Optimization).

4. Ensure that the baud rate is set to 1.5e6 bits/sec.


Configuration Parameters: mcb_pmsm_foc_qep_f28035/Configuration (Active)

Base rate trigger: ADCINT1

Target hardware resources

Enable loopback

Suspension mode: Free_run

Number of stop bits: 1

Parity mode: None

Character length bits: 8

Desired baud rate in bits/sec: 1.5e6

Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)): 4

Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec: 1500000

Communication mode: Raw_data

Blocking mode

Data byte order: Little_Endian

Pin assignment(Tx): GPIO29

Pin assignment(Rx): GPIO28

Note: For F28335 processor you need to use external FTDI for serial communication.

Required MathWorks® Products

To simulate model:

For the models: **mcb_pmsm_foc_hall_f28035**, **mcb_pmsm_foc_hall_f28335** and **mcb_pmsm_foc_hall_f283049c**

- Motor Control Blockset™
- Fixed-Point Designer™

To generate code and deploy model:

For the models: **mcb_pmsm_foc_hall_f28035**, **mcb_pmsm_foc_hall_f28335** and **mcb_pmsm_foc_hall_f283049c**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

Prerequisites

- If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the

Simulink® models. For instructions, see “Estimate Control Gains and Use Utility Functions” (Motor Control Blockset).

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see “Open-Loop Control of 3-Phase AC Motors Using C2000 Processors” on page 4-171.

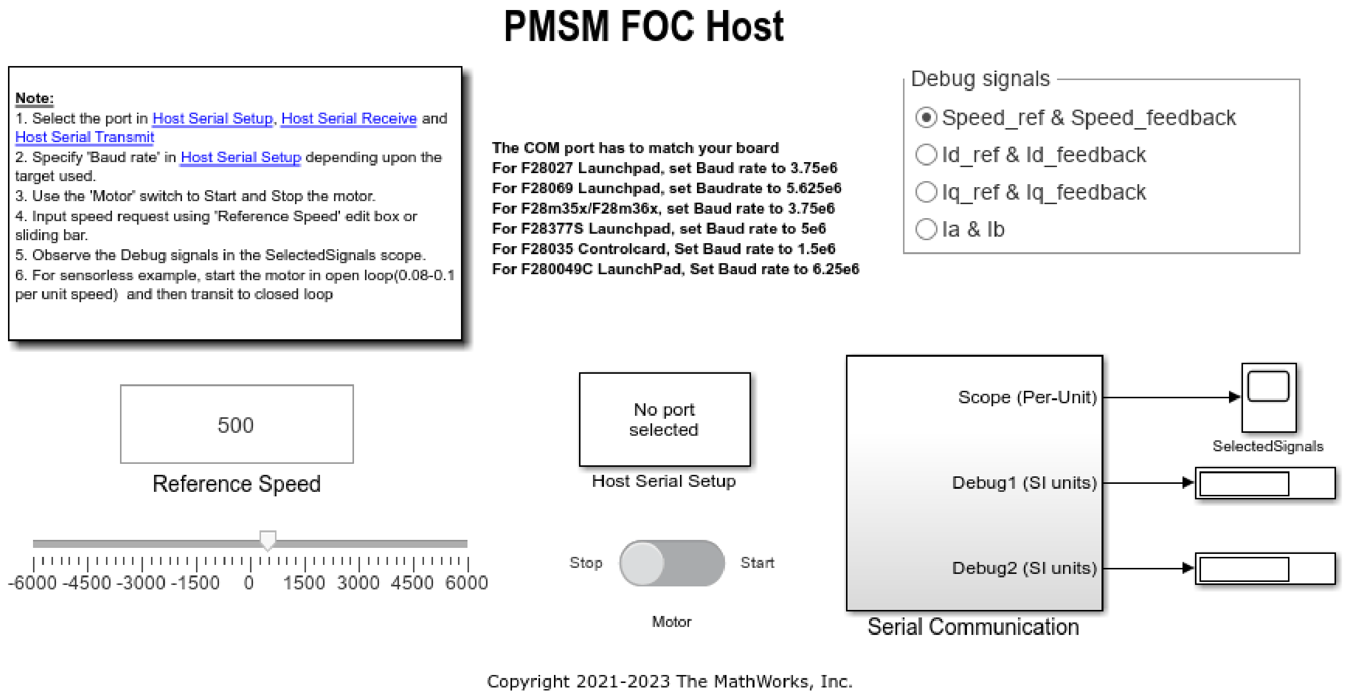
4. Compute the hall sensor offset value and update it in the model initialization scripts associated with the target model. For instructions, see “Hall Offset Calibration for PMSM Motor” on page 1-77.

5. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the model, see “Model Configuration Parameters” (Motor Control Blockset).

6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

7. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller.

```
open_system('mcb_pmsm_foc_host_model.slx');
```



Note:
1. Select the port in Host Serial Setup, Host Serial Receive and Host Serial Transmit
2. Specify 'Baud rate' in Host Serial Setup depending upon the target used.
3. Use the 'Motor' switch to Start and Stop the motor.
4. Input speed request using 'Reference Speed' edit box or sliding bar.
5. Observe the Debug signals in the SelectedSignals scope.
6. For sensorless example, start the motor in open loop(0.08-0.1 per unit speed) and then transit to closed loop

The COM port has to match your board
For F28027 Launchpad, set Baud rate to 3.75e6
For F28069 Launchpad, set Baudrate to 5.625e6
For F28m35x/F28m36x, set Baud rate to 3.75e6
For F28377S Launchpad, set Baud rate to 5e6
For F28035 Controlcard, Set Baud rate to 1.5e6
For F280049C LaunchPad, Set Baud rate to 6.25e6

Copyright 2021-2023 The MathWorks, Inc.

For details about the serial communication between the host and target models, see “Host-Target Communication” (Motor Control Blockset).

8. Set the parameter **Port** of the following blocks in the mcb_pmsm_foc_host_model model to match the COM port of the host computer:

- mcb_pmsm_foc_host_model > Host Serial Setup
- mcb_pmsm_foc_host_model > Serial Communication > Host Serial Receive
- mcb_pmsm_foc_host_model > Serial Communication > SCI_TX > Host Serial Transmit

9. Update the Reference Speed value in the host model.

10. Click **Run** on the **Simulation** tab to run the host model.

11. Change the position of the motor switch to Start, to start running the motor.

12. Observe the debug signals from the serial communication, in the Time Scope of host model. Use Debug signals to visualize different debug signals in **SelectedSignals**.

More About:

- “Host-Target Communication” (Motor Control Blockset)
- “Estimate PMSM Parameters Using Recommended Hardware” (Motor Control Blockset)
- For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Field-Oriented Control of PMSM Using Hall Sensor” (Motor Control Blockset)

- “Hall Offset Calibration for PMSM Motor” on page 1-77

Open-Loop Control of 3-Phase AC Motors Using C2000 Processors

This example uses open-loop control (also known as scalar control or Volts/Hz control) to run a motor. This technique varies the stator voltage and frequency to control the rotor speed without using any feedback from the motor. You can use this technique to check the integrity of the hardware connections. A constant speed application of open-loop control uses a fixed-frequency motor power supply. An adjustable speed application of open-loop control needs a variable-frequency power supply to control the rotor speed. To ensure a constant stator magnetic flux, keep the supply voltage amplitude proportional to its frequency.

Open-loop motor control does not have the ability to consider the external conditions that can affect the motor speed. Therefore, the control system cannot automatically correct the deviation between the desired and the actual motor speed.

This model runs the motor by using an open-loop motor control algorithm. The model helps you get started with Motor Control Blockset™ and verify the hardware setup by running the motor. The target model algorithm also reads the ADC values from the current sensors and sends the values to the host model by using serial communication.

You can use this model to:

- Check connectivity with the target.
- Check serial communication with the target.
- Verify the hardware and software environment.
- Check ADC offsets for current sensors.
- Run a new motor with an inverter and target setup for the first time.

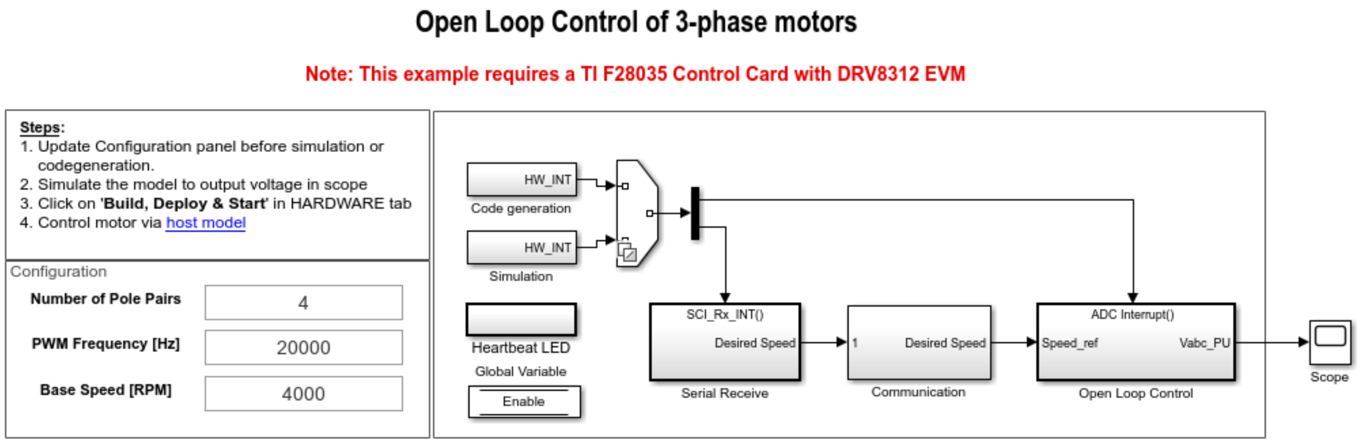
Models

The example includes these models:

- mcb_open_loop_control_f28035_DRV8312
- mcb_open_loop_control_f28027LaunchPad
- mcb_open_loop_control_f280049C_DRV8305
- For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset” (Motor Control Blockset).

You can use these models for both simulation and code generation. You can also use the open_system command to open the Simulink® models. For example, use this command for a F28035 based controller:

```
open_system('mcb_open_loop_control_f28035_DRV8312.slx');
```



Steps:
1. Update Configuration panel before simulation or codegeneration.
2. Simulate the model to output voltage in scope
3. Click on '**Build, Deploy & Start**' in HARDWARE tab
4. Control motor via host model

Configuration

Number of Pole Pairs	4
PWM Frequency [Hz]	20000
Base Speed [RPM]	4000

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks Products

To simulate model:

For the models: **mcb_open_loop_control_f28035_DRV8312**, **mcb_open_loop_control_f28027LaunchPad** and **mcb_open_loop_control_f280049C_DRV8305**

- Motor Control Blockset
- Fixed-Point Designer™

To generate code and deploy model:

For the models: **mcb_open_loop_control_f28035_DRV8312**, **mcb_open_loop_control_f28027LaunchPad** and **mcb_open_loop_control_f280049C_DRV8305**

- Motor Control Blockset
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer

Prerequisites

For BOOSTXL-DRV8301/8305, use these steps to update the model:

- Navigate to this path in the model: /Open Loop Control/Codegen/Hardware Initialization.
- For LAUNCHXL-F28027/F28027F: Update DRV830x Enable block from GPIO6.
- For DRV8312-F28035 ControlCard: Configure GPIO1, GPIO3 and GPIO5 to Active High enable the inverter.
- For DRV8305-F28049C LAUNCHXL: Configure GPIO39 to Active High enable the DRV830x inverter.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the motor by using open-loop control.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

This example supports these hardware configurations. You can also use the target model name to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- F28035 controller card + DRV8312-69M-KIT inverter: mcb_open_loop_control_f28035_DRV8312
- F280049C LAUNCHXL + BOOSTXL-DRV8305 inverter: mcb_open_loop_control_f280049C_DRV8305
- LAUNCHXL-F28027 controller + (BOOSTXL-DRV8301 or BOOSTXL-DRV8305 inverter: mcb_open_loop_control_f28027LaunchPad
- For F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller, refer to “Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset” (Motor Control Blockset).

For connections related to the preceding hardware configuration, see “Hardware Connections” on page 1-81.

NOTE:

- This example supports any type of three-phase AC motor (PMSM or induction) and any type of inverter attached to the supported hardware.
- Some PMSMs do not run at higher speeds, especially when the shaft is loaded. To resolve this issue, you should apply more voltages corresponding to a given frequency. You can use these steps to increase the applied voltages in the model:

1. Navigate to this path in the model: /Open Loop Control/Control_System/VabcCalc/.

2. Update the gain Correction_Factor_sinePWM as 20%.

3. For safety reasons, regularly monitor the motor shaft, motor current, and motor temperature.

Generate Code and Run Model to Implement Open-Loop Control

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see “Model Configuration Parameters” (Motor Control Blockset).

4. Update these motor parameters in the **Configuration** panel of the target model.

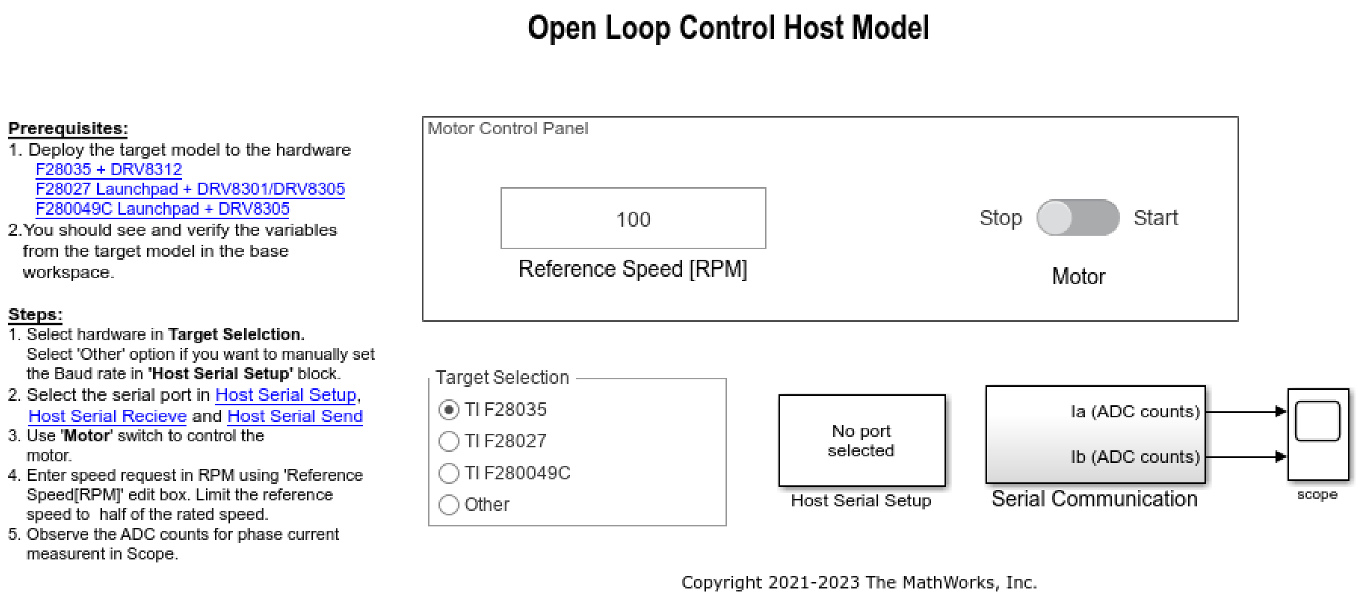
- **Number of Pole Pairs**
- **PWM Frequency [Hz]**
- **Base Speed [RPM]**
- **Data type for control algorithm**

5. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

NOTE: Ignore the warning message "Multitask data store option in the Diagnostics page of the Configuration Parameter Dialog is none" displayed by the model advisor, by clicking the Always Ignore button. This is part of the intended workflow.

6. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28035 based controller:

```
open_system('mcb_c2000_open_loop_control_host_model.slx');
```



Open Loop Control Host Model

Prerequisites:
1. Deploy the target model to the hardware
   F28035 + DRV8312
   F28027 Launchpad + DRV8301/DRV8305
   F280049C Launchpad + DRV8305
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select hardware in **Target Selelction.** Select 'Other' option if you want to manually set the Baud rate in **'Host Serial Setup'** block.
2. Select the serial port in Host Serial Setup, Host Serial Recieve and Host Serial Send
3. Use '**Motor**' switch to control the motor.
4. Enter speed request in RPM using 'Reference Speed[RPM]' edit box. Limit the reference speed to half of the rated speed.
5. Observe the ADC counts for phase current measurent in Scope.

Target Selection: [✓ TI F28035] TI F28027 TI F280049C Other

Copyright 2021-2023 The MathWorks, Inc.

For details about the serial communication between the host and target models, see “Host-Target Communication” (Motor Control Blockset).

7. Set the parameter **Port** of the following blocks in the mcb_c2000_open_loop_control_host_model model to match the COM port of the host computer:

- mcb_c2000_open_loop_control_host_model > Host Serial Setup

- mcb_c2000_open_loop_control_host_model > Serial Communication > Host Serial Receive
- mcb_c2000_open_loop_control_host_model > Serial Communication > Enabled Subsystem > Host Serial Transmit

8. Select a target (either TI F28035 or TI F28027D or TI F280049C) in the **Configuration Panel** of the host model.

9. Enter the Reference Speed value in the host model.

10. Click **Run** on the **Simulation** tab to run the host model.

11. Change the position of the Start / Stop Motor switch to On, to start running the motor.

12. After the motor is running, observe the ADC counts for the I_a and I_b currents in the Time Scope.

NOTE: This example may not allow the motor to run at full capacity. Begin running the motor at a small speed. In addition, it is recommended to change the Reference Speed in small steps (for example, for a motor having a base speed of 3000 rpm, start running the motor at 500 rpm and then increase or decrease the speed in steps of 200 rpm).

If the motor does not run, change the position of the Start / Stop Motor switch to Off, to stop the motor and change the Reference Speed in the host model. Then, change the position of the Start / Stop Motor switch to On, to run the motor again.

Generate Code and Run Model to Calibrate ADC Offset

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. Disconnect the motor wires for three phases from the hardware board terminals.

4. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see “Model Configuration Parameters” (Motor Control Blockset).

5. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

NOTE: Ignore the warning message "Multitask data store option in the Diagnostics page of the Configuration Parameter Dialog is none" displayed by the model advisor, by clicking the Always Ignore button. This is part of the intended workflow.

6. Click the **host model** hyperlink in the target model to open the associated host model.

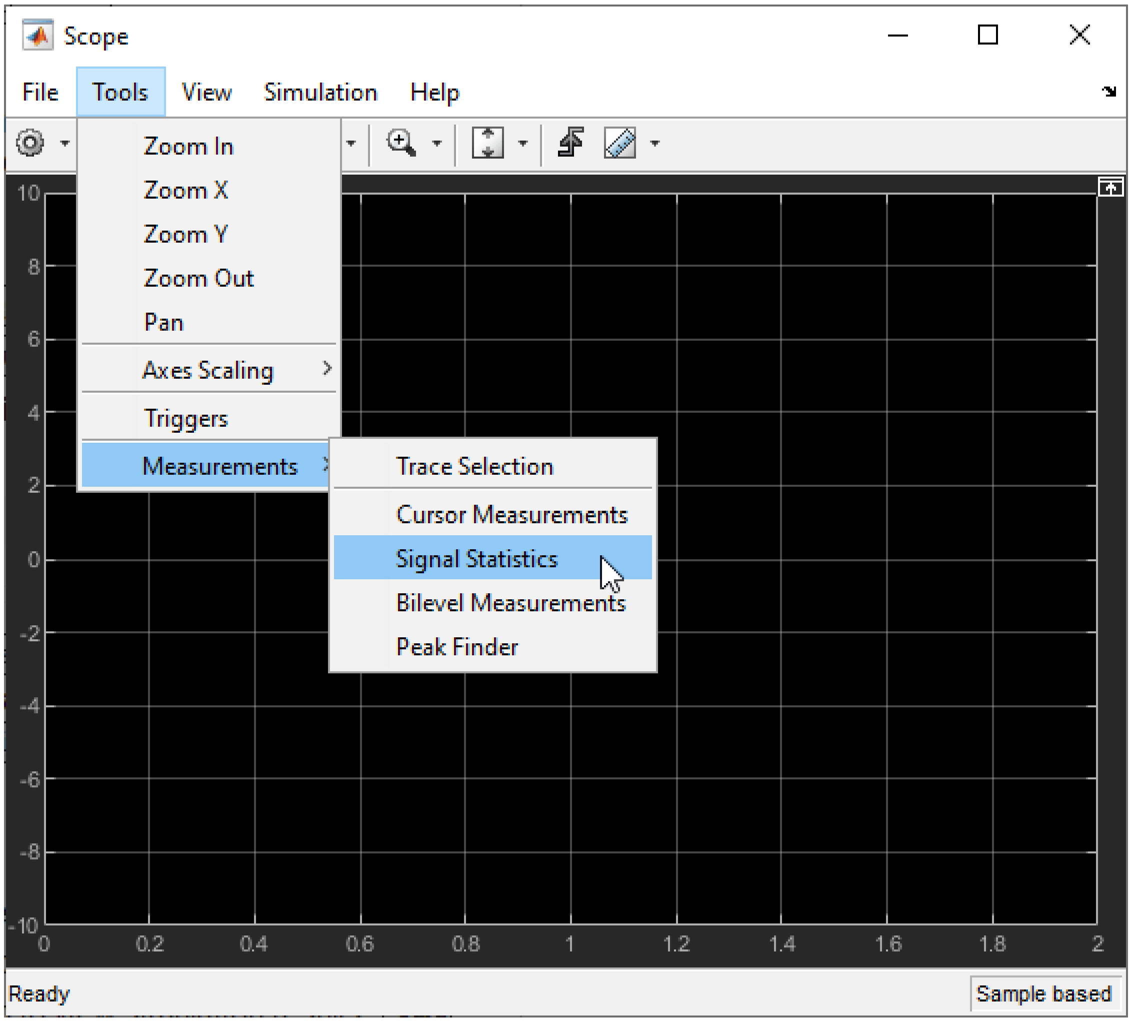
7. Set the parameter **Port** of the following blocks in the mcb_c2000_open_loop_control_host_model model to match the COM port of the host computer:

- mcb_c2000_open_loop_control_host_model > Host Serial Setup
- mcb_c2000_open_loop_control_host_model > Serial Communication > Host Serial Receive
- mcb_c2000_open_loop_control_host_model > Serial Communication > Enabled Subsystem > Host Serial Transmit

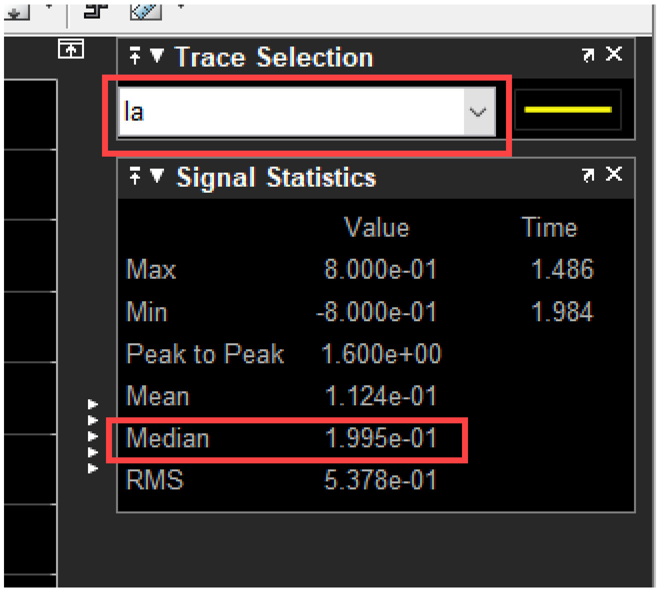
8. Click **Run** on the **Simulation** tab to run the host model.

9. Observe the ADC counts for the I_a and I_b currents in the Time Scope. The average values of the ADC counts are the ADC offset corrections for the currents I_a and I_b. To obtain the average (median) values of ADC counts:

- In the **Scope** window, navigate to **Tools** > **Measurements** and select **Signal Statistics** to display the **Trace Selection** and **Signal Statistics** areas.



- Under **Trace Selection**, select a signal (I_a or I_b). The characteristics of the selected signal are displayed in the **Signal Statistics** pane. You can see the median value of the selected signal in the Median field.



	Value	Time
Max	8.000e-01	1.486
Min	-8.000e-01	1.984
Peak to Peak	1.600e+00	
Mean	1.124e-01	
Median	1.995e-01	
RMS	5.378e-01	

For the Motor Control Blockset examples, update the computed ADC (or current) offset value in the inverter.CtSensAOffset and inverter.CtSensBOffset variables in the model initialization script linked to the example. For instructions, see “Estimate Control Gains and Use Utility Functions” (Motor Control Blockset).

Other Things to Try

- Try running this example on F28069M control card, LAUNCHXL-F28069M controller and LAUNCHXL-F28379D controller. For more information, refer to “Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset” (Motor Control Blockset).
- “Hardware Connections” on page 1-81.

Control PMSM Loaded with Dual Motor (Dyno) Using C2000 Processors

This example uses field-oriented control (FOC) to control two three-phase permanent magnet synchronous motors (PMSM) coupled in a dyno setup. Motor 1 runs in the closed-loop speed control mode. Motor 2 runs in the torque control mode and loads Motor 1 because they are mechanically coupled. You can use this example to test a motor in different load conditions.

The example simulates two motors that are connected back-to-back. You can use a different speed reference for Motor 1 and a different torque reference for Motor 2 (derived from the magnitude and electrical position of the Motor 2 reference stator current). Motor 1 runs at the reference speed for the load conditions provided by Motor 2 (with a different torque reference).

The example runs in the controller hardware board. You can input the speed reference for Motor 1 and current reference for Motor 2 using a host model. The host model uses serial communication to communicate with the controller hardware board.

Current control loops in Motor 1 and Motor 2 control algorithms are offset by Ts/2, where Ts is the control-loop execution rate.

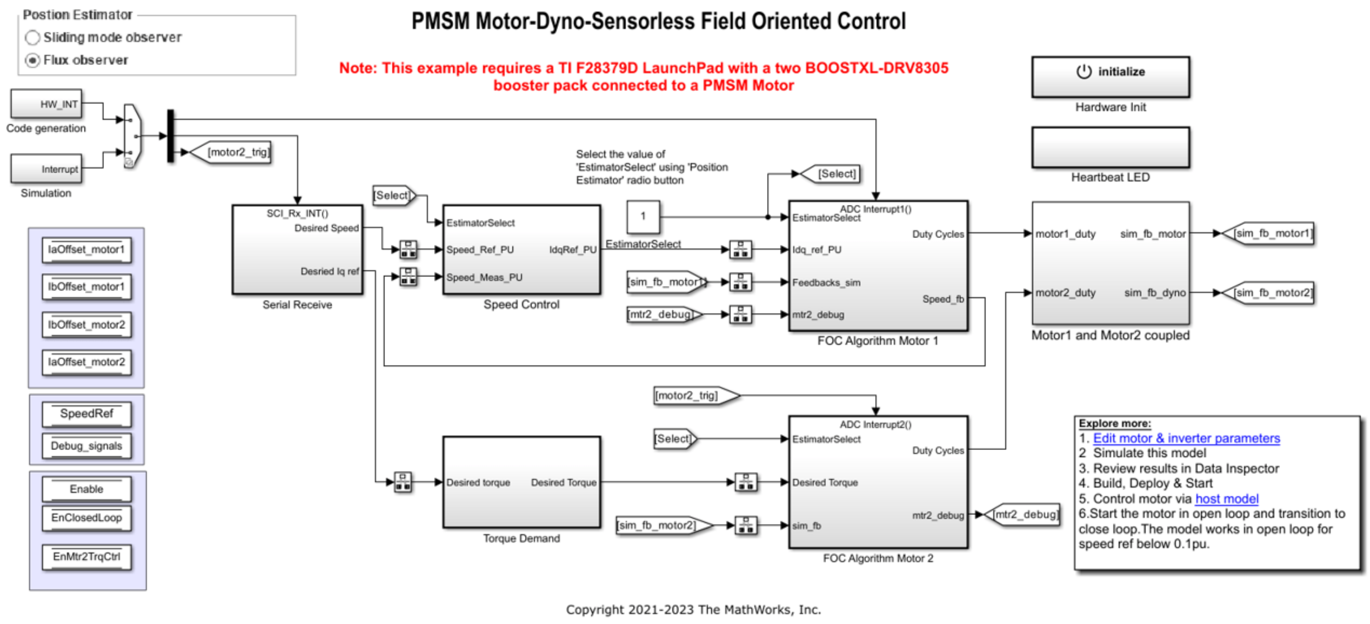
Single CPU Dyno Models

The example includes the following models:

- mcb_pmsm_foc_sensorless_dyno_f28379d.
- mcb_pmsm_foc_qep_dyno_f28069m.
- For QEP based dyno **mcb_pmsm_foc_f28379d_dyno** example, refer "Control PMSM Loaded with Dual Motor (Dyno)" (Motor Control Blockset) .

You can use mcb_pmsm_foc_sensorless_dyno_f28379d and mcb_pmsm_foc_qep_dyno_f28069m models for both simulation and code generation. You can also use the open_system command to open the Simulink® model. These models use timer0 as a base rate trigger. For example, use this command for a F28379D based controller:

```
open_system('mcb_pmsm_foc_sensorless_dyno_f28379d.slx');
```



Dual CPU Dyno Models

The example includes the following models:

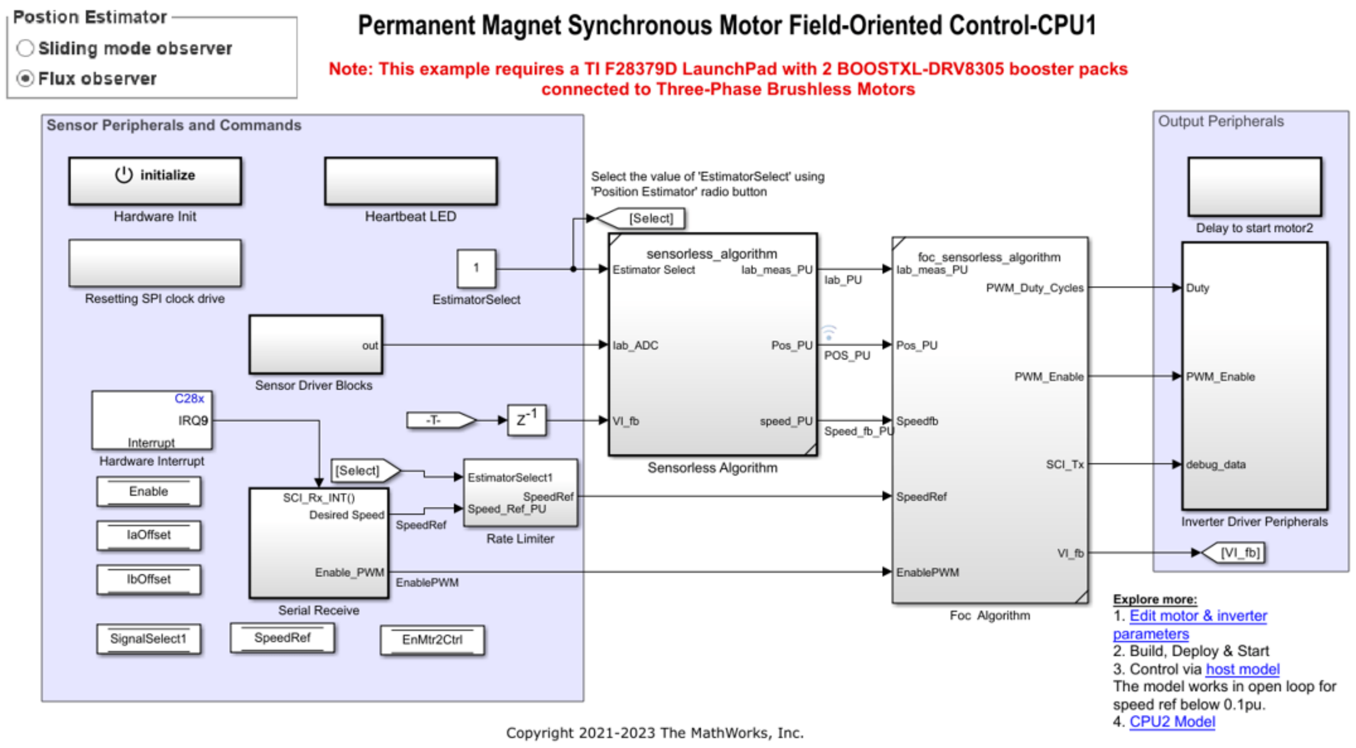
- mcb_pmsm_foc_dual_cpu1_f28379d.
- mcb_pmsm_foc_dual_cpu2_f28379d.

You can also use the open_system command to open the Simulink® model.

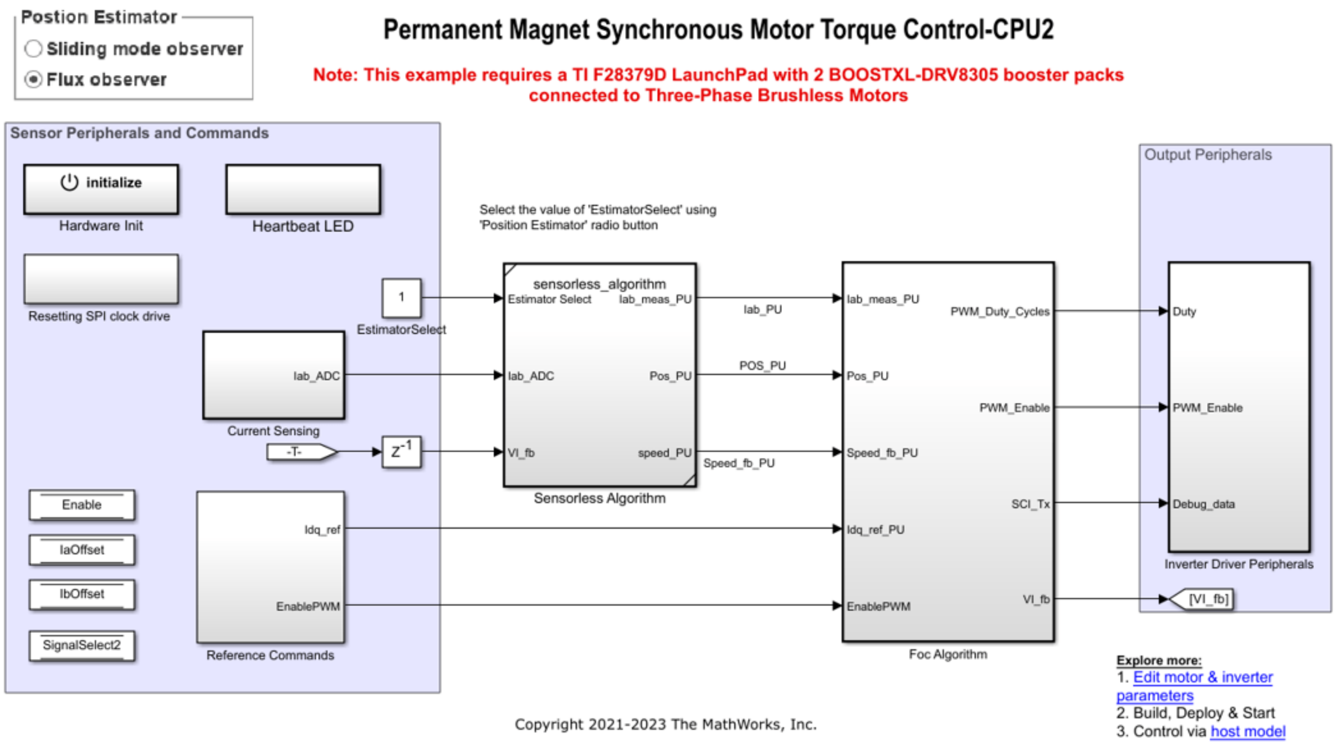
Note: Dual core CPU models does not support simulation.

For example, use this command for a F28379D based controller:

```
open_system('mcb_pmsm_foc_dual_cpu1_f28379d.slx');
```


Postion Estimator
Sliding mode observer
Flux observer
Permanent Magnet Synchronous Motor Field-Oriented Control-CPU1
Note: This example requires a TI F28379D LaunchPad with 2 BOOSTXL-DRV8305 booster packs connected to Three-Phase Brushless Motors
Sensor Peripherals and Commands
initialize
Hardware Init
Heartbeat LED
Resetting SPI clock drive
EstimatorSelect
Sensor Driver Blocks
Hardware Interrupt
Serial Receive
Rate Limiter
Enable
IaOffset
IbOffset
SignalSelect1
SpeedRef
EnMtr2Ctrl
Select the value of 'EstimatorSelect' using 'Position Estimator' radio button
sensorless_algorithm
Sensorless Algorithm
foc_sensorless_algorithm
Foc Algorithm
Output Peripherals
Delay to start motor2
Inverter Driver Peripherals
Explore more:
1. Edit motor & inverter parameters
2. Build, Deploy & Start
3. Control via host model
The model works in open loop for speed ref below 0.1pu.
4. CPU2 Model
Copyright 2021-2023 The MathWorks, Inc.

```
open_system('mcb_pmsm_foc_dual_cpu2_f28379d.slx');
```


Postion Estimator
Sliding mode observer
Flux observer
Permanent Magnet Synchronous Motor Torque Control-CPU2
Note: This example requires a TI F28379D LaunchPad with 2 BOOSTXL-DRV8305 booster packs connected to Three-Phase Brushless Motors
Sensor Peripherals and Commands
initialize
Hardware Init
Heartbeat LED
Resetting SPI clock drive
EstimatorSelect
Current Sensing
Enable
IaOffset
IbOffset
SignalSelect2
Reference Commands
Select the value of 'EstimatorSelect' using 'Position Estimator' radio button
sensorless_algorithm
Sensorless Algorithm
Foc Algorithm
Output Peripherals
Inverter Driver Peripherals
Explore more:
1. Edit motor & inverter parameters
2. Build, Deploy & Start
3. Control via host model
Copyright 2021-2023 The MathWorks, Inc.

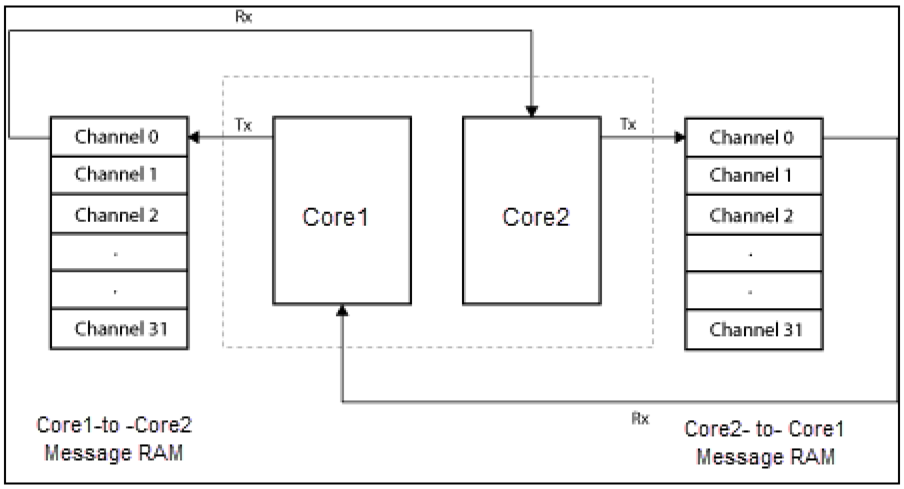
IPC Communication for Dual Core Processors

The CPU1 transmits the following data to CPU2 through IPC channels:

- Torque reference
- Signal select
- Enable Motor2(EnMtr2Ctrl)

The CPU2 transmits the debug signal according to the signal select of CPU1.

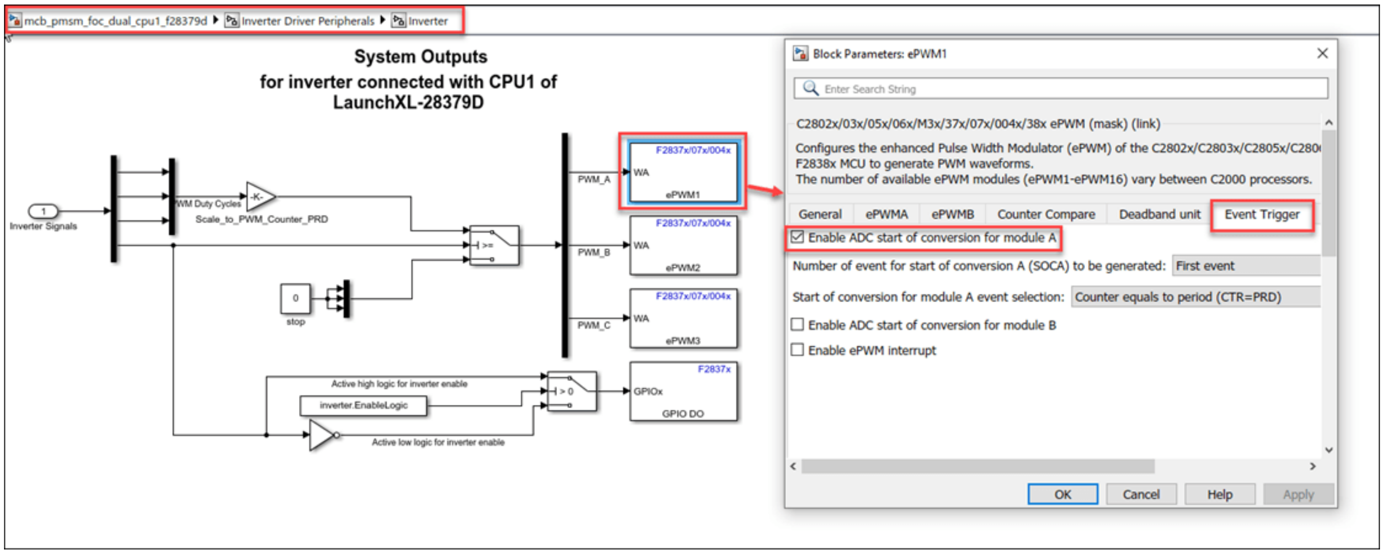
Core1 transmits data to its allocated memory (Core1-to-Core2 Message RAM) and receives data from the allocated memory of Core2 (Core2-to-Core1 Message RAM). Similarly, Core2 transmits data to its allocated memory (Core2-to-Core1 Message RAM) and receives data from allocated memory of Core1 (Core1-to-Core2 Message RAM). For more information, refer “Inter-Processor Communication Using IPC Blocks” on page 4-86.



Peripheral Block Configurations for Dyno Setup

Set the peripheral block configurations for this model. Double-click on the blocks to open block parameter configurations. You can use the same parameter values if you want to run this example for other hardware boards.

- **ePWM Block Configuration** The ePWM1/2/3 is offset by ePWM time period from ePWM4/5/6. This is achieved by sending a synchronization signal from ePWM3 to ePWM4 and setting the phase offset value of ePWM4 as `Target.PWM_counter_period`.


mcb_pmsm_foc_dual_cpu1_f28379d ▸ Inverter Driver Peripherals ▸ Inverter
System Outputs
for inverter connected with CPU1 of
LaunchXL-28379D
Inverter Signals
PWM Duty Cycles
Scale_to_PWM_Counter_PRD
stop
PWM_A
PWM_B
PWM_C
F2837x/07x/004x
WA
ePWM1
ePWM2
ePWM3
F2837x
GPIOx
GPIO DO
Active high logic for inverter enable
inverter.EnableLogic
Active low logic for inverter enable

Block Parameters: ePWM1

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ePWM (mask) (link)

Configures the enhanced Pulse Width Modulator (ePWM) of the C2802x/C2803x/C2805x/C280
F2838x MCU to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

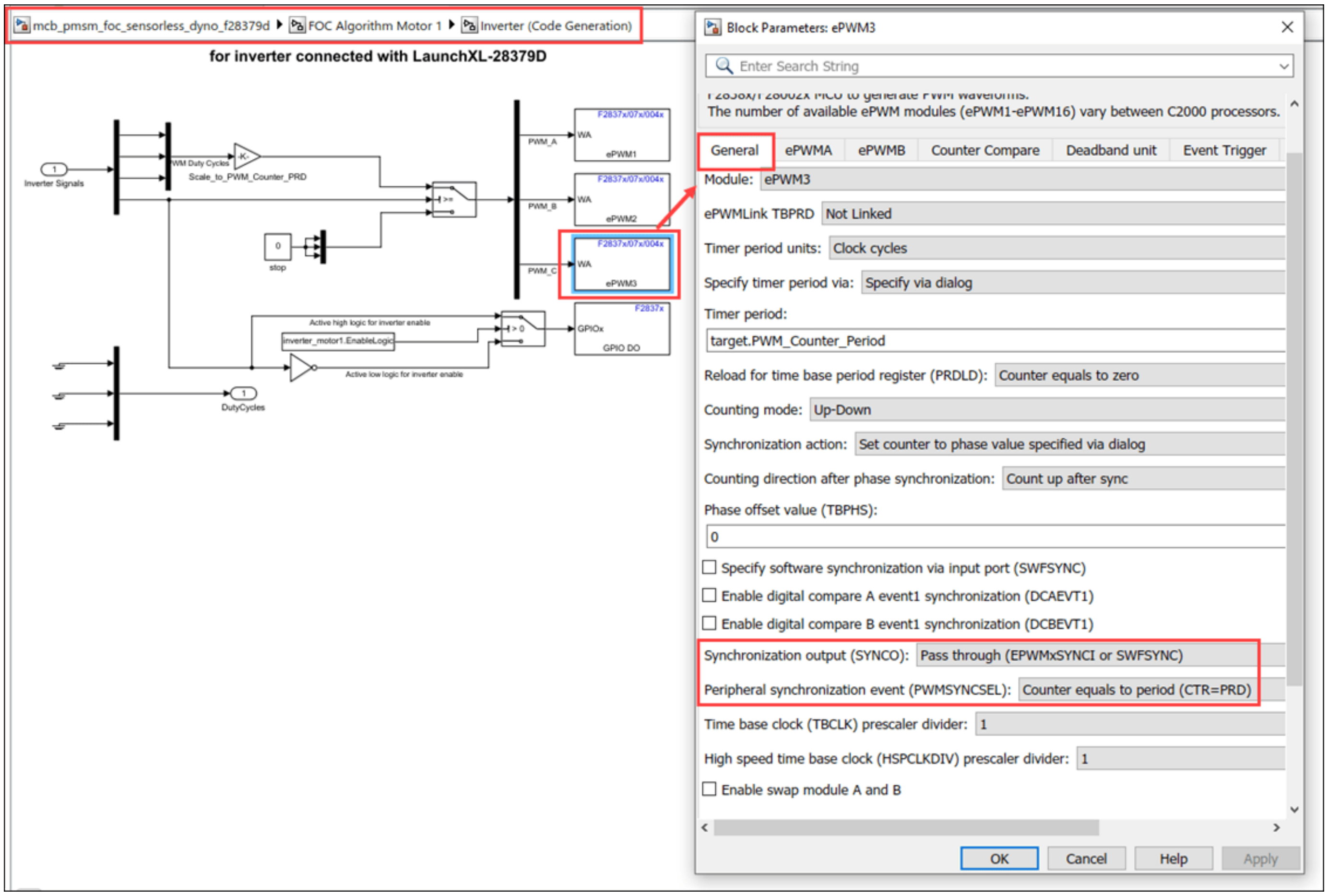
☑ Enable ADC start of conversion for module A

Number of event for start of conversion A (SOCA) to be generated: First event

Start of conversion for module A event selection: Counter equals to period (CTR=PRD)

☐ Enable ADC start of conversion for module B

☐ Enable ePWM interrupt


mcb_pmsm_foc_sensorless_dyno_f28379d ▸ FOC Algorithm Motor 1 ▸ Inverter (Code Generation)
for inverter connected with LaunchXL-28379D
Inverter Signals
PWM Duty Cycles
Scale_to_PWM_Counter_PRD
stop
PWM_A
PWM_B
PWM_C
F2837x/07x/004x
WA
ePWM1
ePWM2
ePWM3
F2837x
GPIOx
GPIO DO
Active high logic for inverter enable
inverter_motor1.EnableLogic
Active low logic for inverter enable
DutyCycles

Block Parameters: ePWM3

The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

Module: ePWM3

ePWMLink TBPRD: Not Linked

Timer period units: Clock cycles

Specify timer period via: Specify via dialog

Timer period:
target.PWM_Counter_Period

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up-Down

Synchronization action: Set counter to phase value specified via dialog

Counting direction after phase synchronization: Count up after sync

Phase offset value (TBPHS):
0

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

☐ Enable digital compare B event1 synchronization (DCBEVT1)

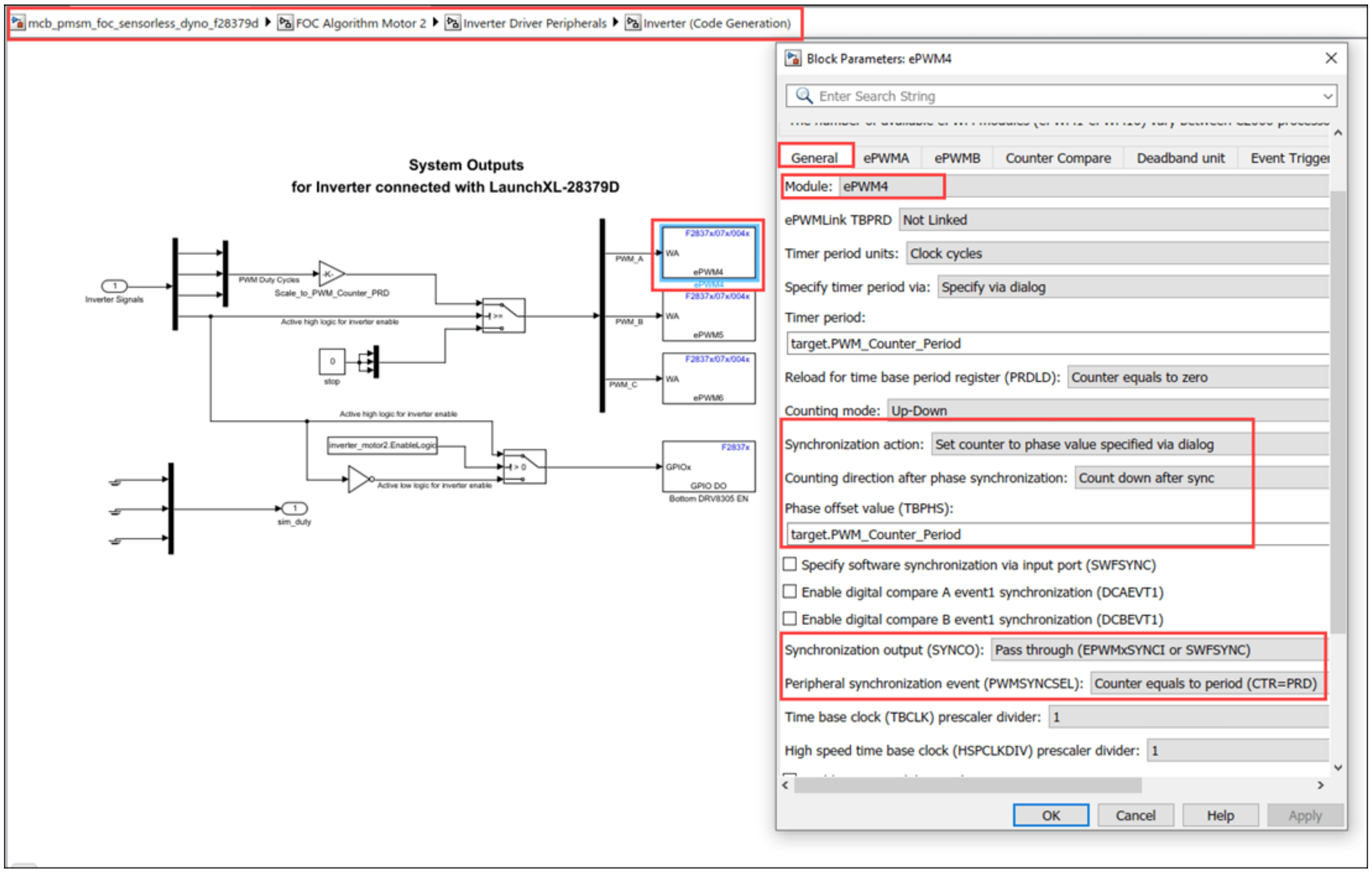
Synchronization output (SYNCO): Pass through (EPWMxSYNCI or SWFSYNC)

Peripheral synchronization event (PWMSYNCSEL): Counter equals to period (CTR=PRD)

Time base clock (TBCLK) prescaler divider: 1

High speed time base clock (HSPCLKDIV) prescaler divider: 1

☐ Enable swap module A and B


mcb_pmsm_foc_sensorless_dyno_f28379d ▸ FOC Algorithm Motor 2 ▸ Inverter Driver Peripherals ▸ Inverter (Code Generation)
System Outputs
for Inverter connected with LaunchXL-28379D
Inverter Signals
PWM Duty Cycles
Scale_to_PWM_Counter_PRD
Active high logic for inverter enable
stop
inverter_motor2.EnableLogic
Active low logic for inverter enable
sim_duty
PWM_A
PWM_B
PWM_C
ePWM4
ePWM5
ePWM6
GPIO DO
Bottom DRV8305 EN
OK
Cancel
Help
Apply

Block Parameters: ePWM4

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger

Module: ePWM4

ePWMLink TBPRD: Not Linked

Timer period units: Clock cycles

Specify timer period via: Specify via dialog

Timer period: target.PWM_Counter_Period

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up-Down

Synchronization action: Set counter to phase value specified via dialog

Counting direction after phase synchronization: Count down after sync

Phase offset value (TBPHS): target.PWM_Counter_Period

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

☐ Enable digital compare B event1 synchronization (DCBEVT1)

Synchronization output (SYNCO): Pass through (EPWMxSYNCI or SWFSYNC)

Peripheral synchronization event (PWMSYNCSEL): Counter equals to period (CTR=PRD)

Time base clock (TBCLK) prescaler divider: 1

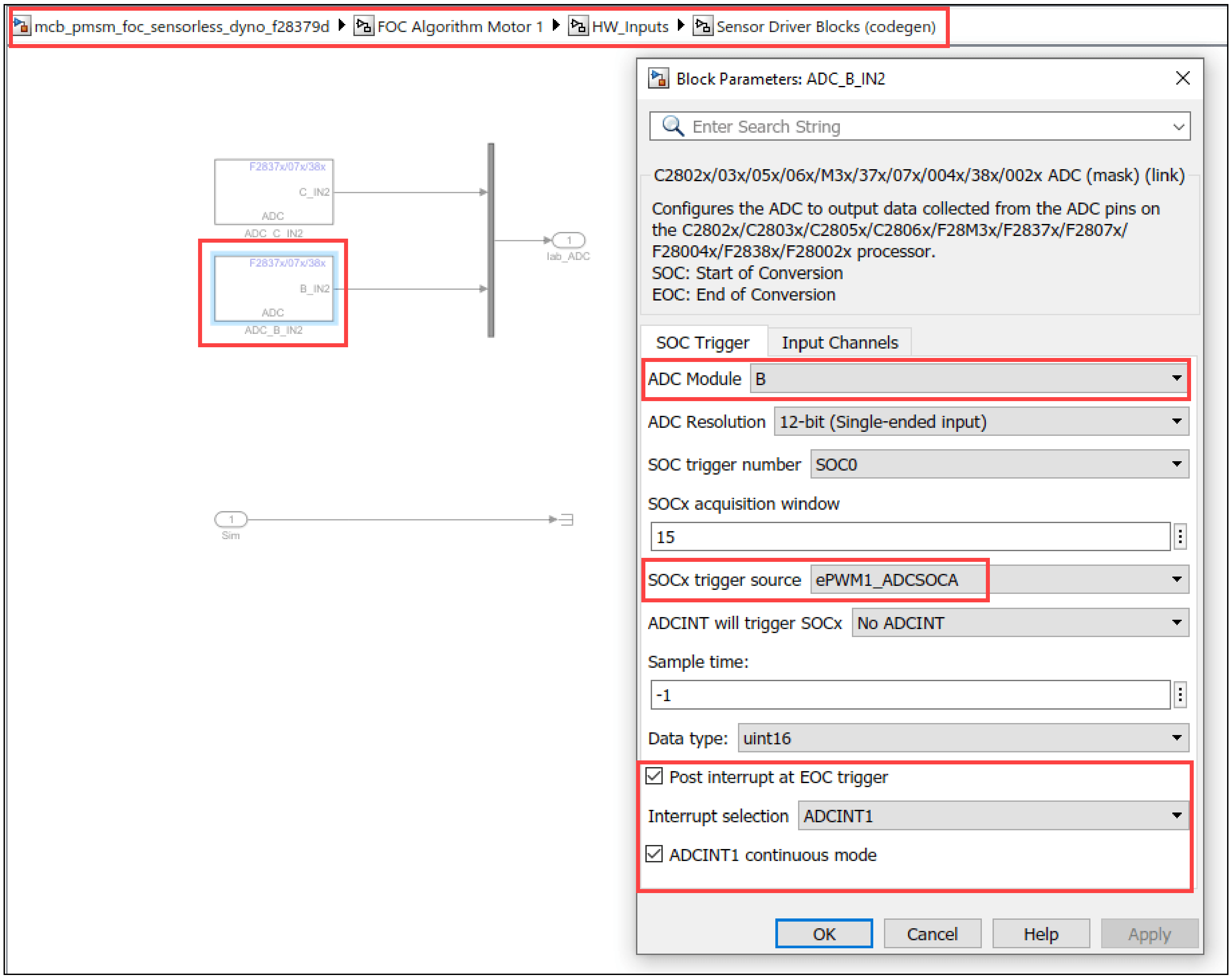
High speed time base clock (HSPCLKDIV) prescaler divider: 1

- **ADC Block Configuration**

The algorithm in this example uses an asynchronous scheduling. The pulse width modulation (PWM) block triggers the ADC conversion. At the end of conversion, the ADC posts an interrupt that triggers the main FOC algorithm. For more information, refer “ADC Interrupt Based Scheduling” on page 1-12.

For **Single CPU** model:

- ADC channel for Inverter 1



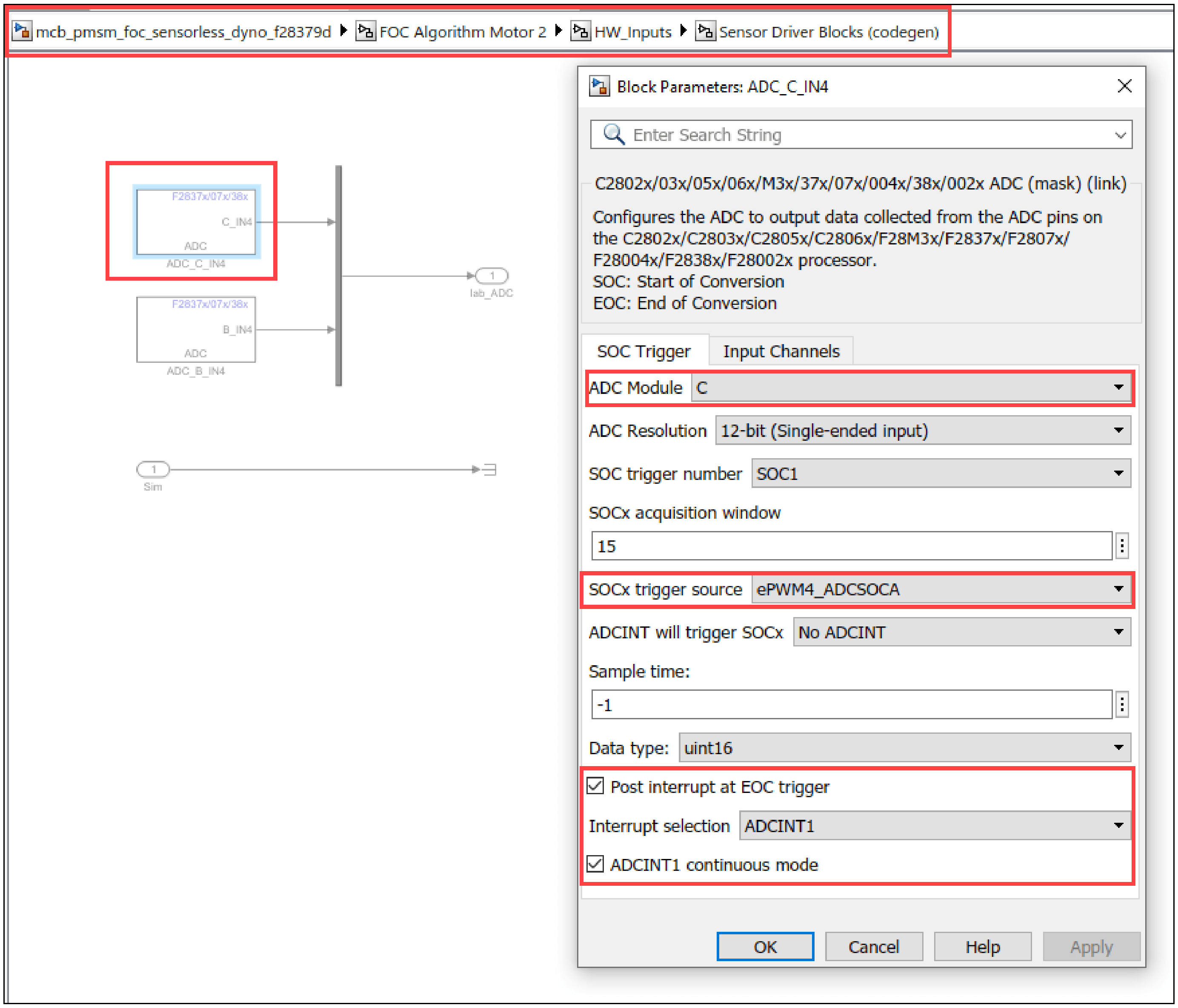
Block Parameters: ADC_B_IN2

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x/002x ADC (mask) (link)

Configures the ADC to output data collected from the ADC pins on the C2802x/C2803x/C2805x/C2806x/F28M3x/F2837x/F2807x/F28004x/F2838x/F28002x processor.
SOC: Start of Conversion
EOC: End of Conversion

ADC Module: B
ADC Resolution: 12-bit (Single-ended input)
SOC trigger number: SOC0
SOCx acquisition window: 15
SOCx trigger source: ePWM1_ADCSOCA
ADCINT will trigger SOCx: No ADCINT
Sample time: -1
Data type: uint16
☑ Post interrupt at EOC trigger
Interrupt selection: ADCINT1
☑ ADCINT1 continuous mode

- ADC channel for inverter 2



Block Parameters: ADC_C_IN4

C2802x/03x/05x/06x/M3x/37x/07x/004x/38x/002x ADC (mask) (link)

Configures the ADC to output data collected from the ADC pins on the C2802x/C2803x/C2805x/C2806x/F28M3x/F2837x/F2807x/F28004x/F2838x/F28002x processor.
SOC: Start of Conversion
EOC: End of Conversion

ADC Module: C

ADC Resolution: 12-bit (Single-ended input)

SOC trigger number: SOC1

SOCx acquisition window: 15

SOCx trigger source: ePWM4_ADCSOCA

ADCINT will trigger SOCx: No ADCINT

Sample time: -1

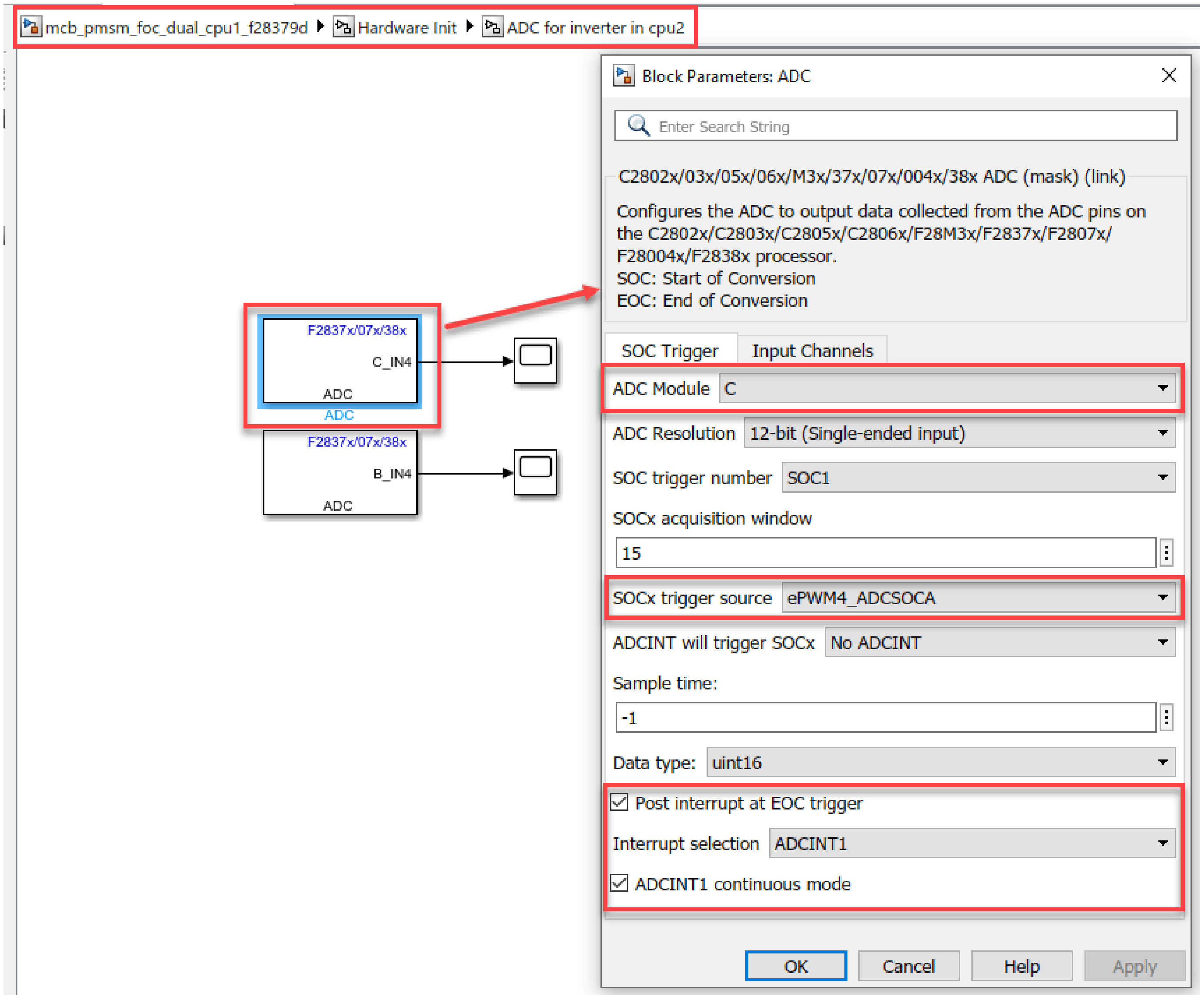
Data type: uint16

☑ Post interrupt at EOC trigger

Interrupt selection: ADCINT1

☑ ADCINT1 continuous mode

For **Dual CPU** model, the ADC channels for inverter 2 is defined in the initialize function of the CPU1 model. In CPU2 model, we use Memory Copy blocks, to access the value of the ADC registers.



C2802x/03x/05x/06x/M3x/37x/07x/004x/38x ADC (mask) (link)

Configures the ADC to output data collected from the ADC pins on the C2802x/C2803x/C2805x/C2806x/F28M3x/F2837x/F2807x/F28004x/F2838x processor.
SOC: Start of Conversion
EOC: End of Conversion

ADC Module: C

ADC Resolution: 12-bit (Single-ended input)

SOC trigger number: SOC1

SOCx acquisition window: 15

SOCx trigger source: ePWM4_ADCSOCA

ADCINT will trigger SOCx: No ADCINT

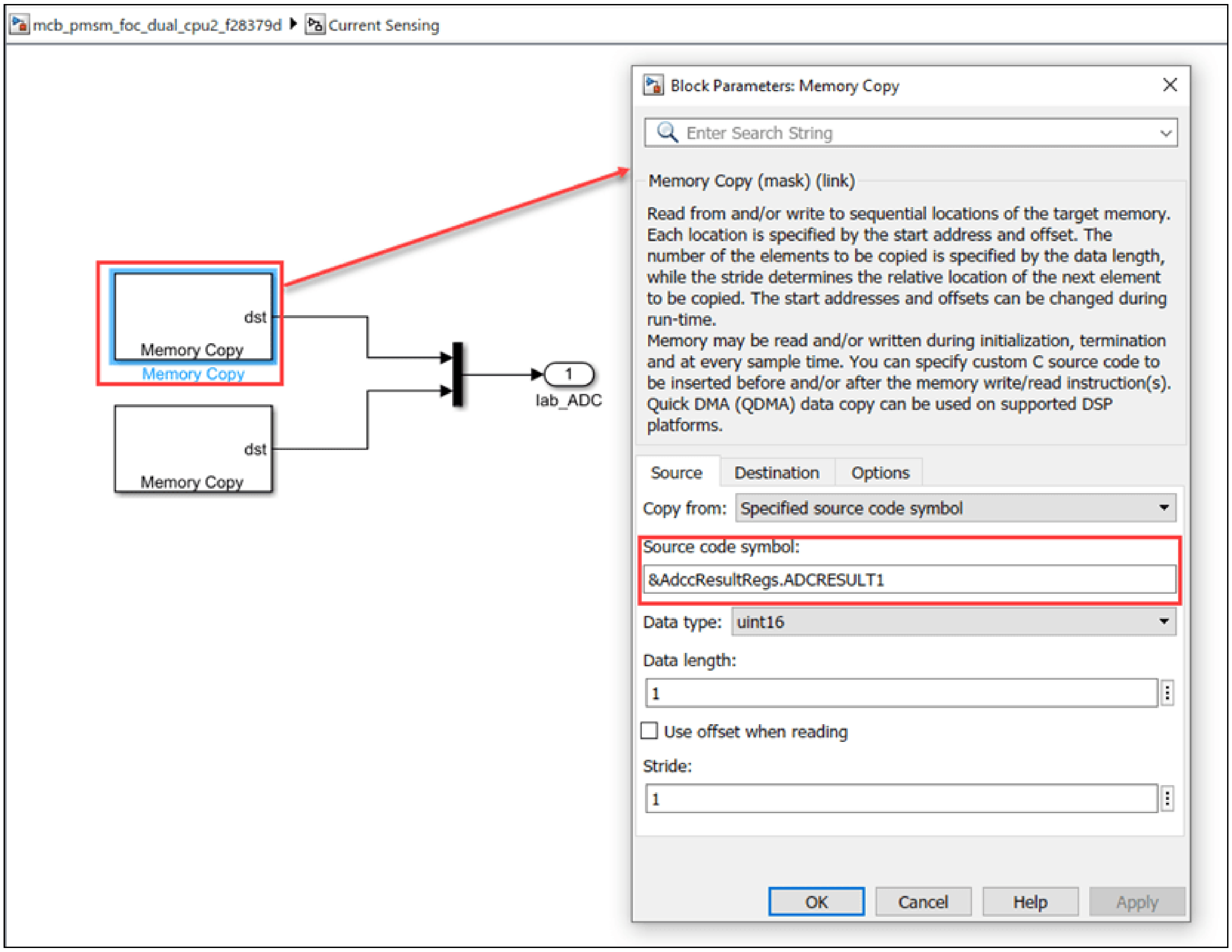
Sample time: -1

Data type: uint16

[✓] Post interrupt at EOC trigger

Interrupt selection: ADCINT1

[✓] ADCINT1 continuous mode



Memory Copy (mask) (link)

Read from and/or write to sequential locations of the target memory. Each location is specified by the start address and offset. The number of the elements to be copied is specified by the data length, while the stride determines the relative location of the next element to be copied. The start addresses and offsets can be changed during run-time.
Memory may be read and/or written during initialization, termination and at every sample time. You can specify custom C source code to be inserted before and/or after the memory write/read instruction(s). Quick DMA (QDMA) data copy can be used on supported DSP platforms.

Copy from: Specified source code symbol

Source code symbol: &AdccResultRegs.ADCRESULT1

Data type: uint16

Data length: 1

☐ Use offset when reading

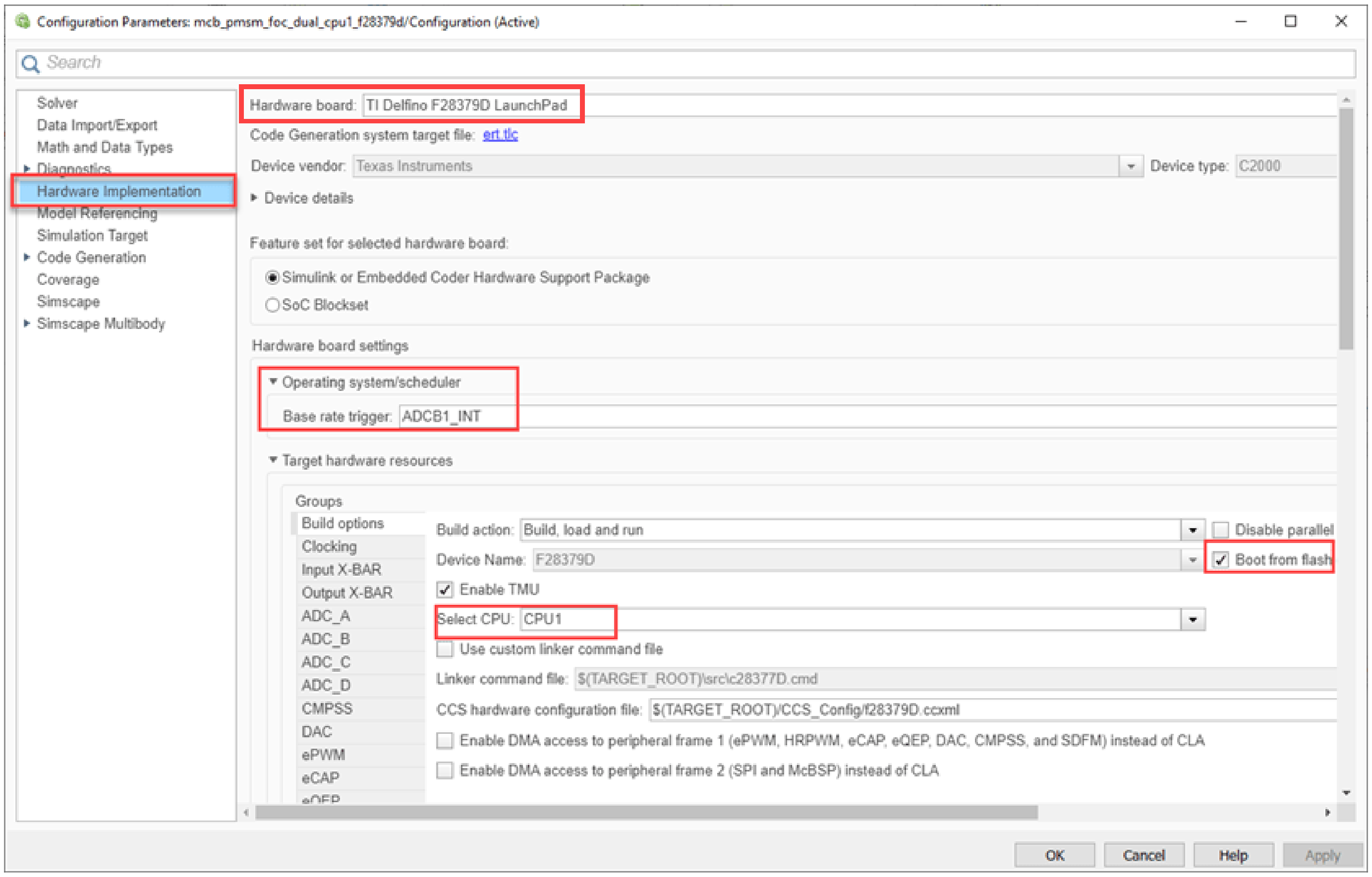
Stride: 1

Configure the Model

1. Open the 'mcb_pmsm_foc_dual_cpu1_f28379d' and 'mcb_pmsm_foc_dual_cpu2_f28379d' model. This model is configured for TI Delfino F28379D Launchpad.

2. To run the model on other TI C2000 processors, first press **Ctrl+E** to open the Configuration Parameters dialog box. Then, select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

3. The following screenshots show the scheduler configurations performed in the **Dual CPU** model.


Configuration Parameters: mcb_pmsm_foc_dual_cpu1_f28379d/Configuration (Active)

Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Feature set for selected hardware board: [✓ Simulink or Embedded Coder Hardware Support Package] SoC Blockset

Hardware board settings

Operating system/scheduler

Base rate trigger: ADCB1_INT

Target hardware resources

Build action: Build, load and run

Disable parallel: ☐

Device Name: F28379D

Boot from flash: ✓

Enable TMU: ✓

Select CPU: CPU1

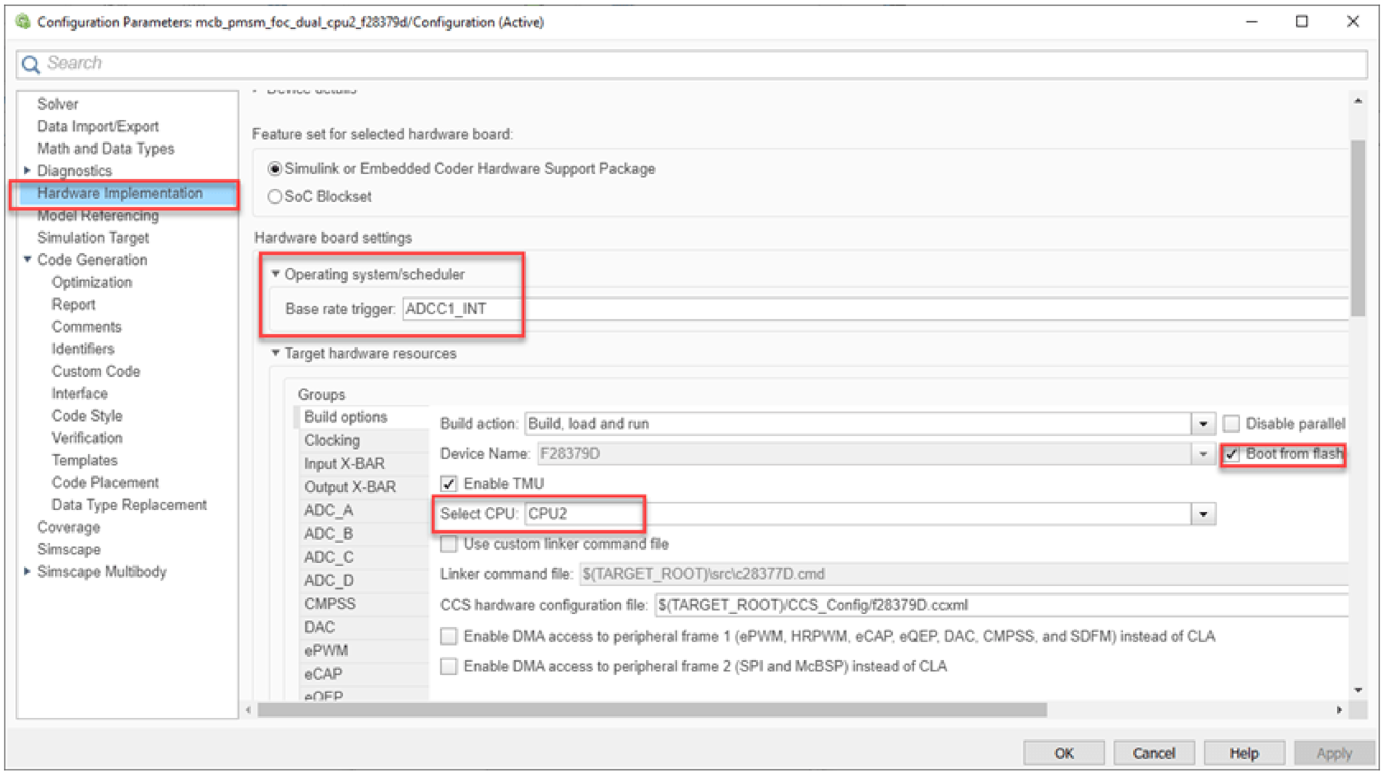
Use custom linker command file: ☐

Linker command file: $(TARGET_ROOT)\src\c28377D.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28379D.ccxml

Enable DMA access to peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC, CMPSS, and SDFM) instead of CLA: ☐

Enable DMA access to peripheral frame 2 (SPI and McBSP) instead of CLA: ☐


Configuration Parameters: mcb_pmsm_foc_dual_cpu2_f28379d/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Optimization
Report
Comments
Identifiers
Custom Code
Interface
Code Style
Verification
Templates
Code Placement
Data Type Replacement
Coverage
Simscape
Simscape Multibody

Feature set for selected hardware board:

(●) Simulink or Embedded Coder Hardware Support Package
( ) SoC Blockset

Hardware board settings

Operating system/scheduler

Base rate trigger: ADCC1_INT

Target hardware resources

Groups: Build options, Clocking, Input X-BAR, Output X-BAR, ADC_A, ADC_B, ADC_C, ADC_D, CMPSS, DAC, ePWM, eCAP, eQEP

Build action: Build, load and run

☐ Disable parallel

Device Name: F28379D

☑ Boot from flash

☑ Enable TMU

Select CPU: CPU2

☐ Use custom linker command file

Linker command file: $(TARGET_ROOT)\src\c28377D.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28379D.ccxml

☐ Enable DMA access to peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC, CMPSS, and SDFM) instead of CLA

☐ Enable DMA access to peripheral frame 2 (SPI and McBSP) instead of CLA

OK Cancel Help Apply

Note:

- For Single CPU model, ensure that the **base rate trigger** is **Timer0**.
- Ensure that the **Default parameter behavior** is set to **Inlined** (**Configuration Parameters > Code Generation > Optimization**).

4. Ensure that the baud rate is set to 12e6 bits/sec.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters for both Motor 1 and Motor 2. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset™ parameter estimation tool. For instructions, see “Estimate PMSM Parameters Using Recommended Hardware” (Motor Control Blockset).

2. Update the motor parameters (that you obtained from the datasheet, other sources, or parameter estimation tool) and inverter parameters in the model initialization script associated with the Simulink® model. For instructions, see “Estimate Control Gains and Use Utility Functions” (Motor Control Blockset).

For this example, update the motor parameters for both the motors in the model initialization script.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open mcb_pmsm_foc_sensorless_dyno_f28379d or mcb_pmsm_foc_qep_dyno_f28069m model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

4. Input a different speed reference for Motor 1 and a different current reference (load) for Motor 2. Observe the measured speed and other logged signals in the Data Inspector.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

The example supports this hardware configuration. You can also use the target model name to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28069M controller + 2 BOOSTXL-DRV8305 inverters: mcb_pmsm_foc_qep_dyno_f28069m
- LAUNCHXL-F28379D controller + 2 BOOSTXL-DRV8305 inverters: mcb_pmsm_foc_dual_cpu1_f28379d
- LAUNCHXL-F28379D controller + 2 BOOSTXL-DRV8305 inverters: mcb_pmsm_foc_dual_cpu2_f28379d
- LAUNCHXL-F28379D controller + 2 BOOSTXL-DRV8305 inverters: mcb_pmsm_foc_f28379d_dyno
- For LAUNCHXL-F28379D controller + 2 BOOSTXL-3PHGANINV inverters: mcb_pmsm_foc_f28379d_dyno, refer to “Control PMSM Loaded with Dual Motor (Dyno)” (Motor Control Blockset)

For connections related to the preceding hardware configuration, see “Hardware Connections” on page 1-81.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Open-Loop Control of 3-Phase AC Motors Using C2000 Processors" on page 4-171.

4. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 1-73.

For this example, update the QEP offset values in the pmsm_motor1.PositionOffset and pmsm_motor2.PositionOffset variables in initialization script.

5. Open the target model. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" (Motor Control Blockset).

6. To ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1, load a sample program to CPU2 of LAUNCHXL-F28379D, for example, a program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx).

Note:

- For dual CPU model, you must build and download CPU1 and CPU2 models to respective cores of the processor. Each core controls one motor on the dual motor control kit. Two cores communicate each other using IPC. CPU1 model sends Load Torque request to CPU2. CPU2 sends data to be logged to CPU1.

- After deploying the code, full power reset should be done on the launchpad to ensure proper execution of the code.

7. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the model to the hardware.

8. Click the **host model** hyperlink in the target model to open the associated host model. These are the following host models:

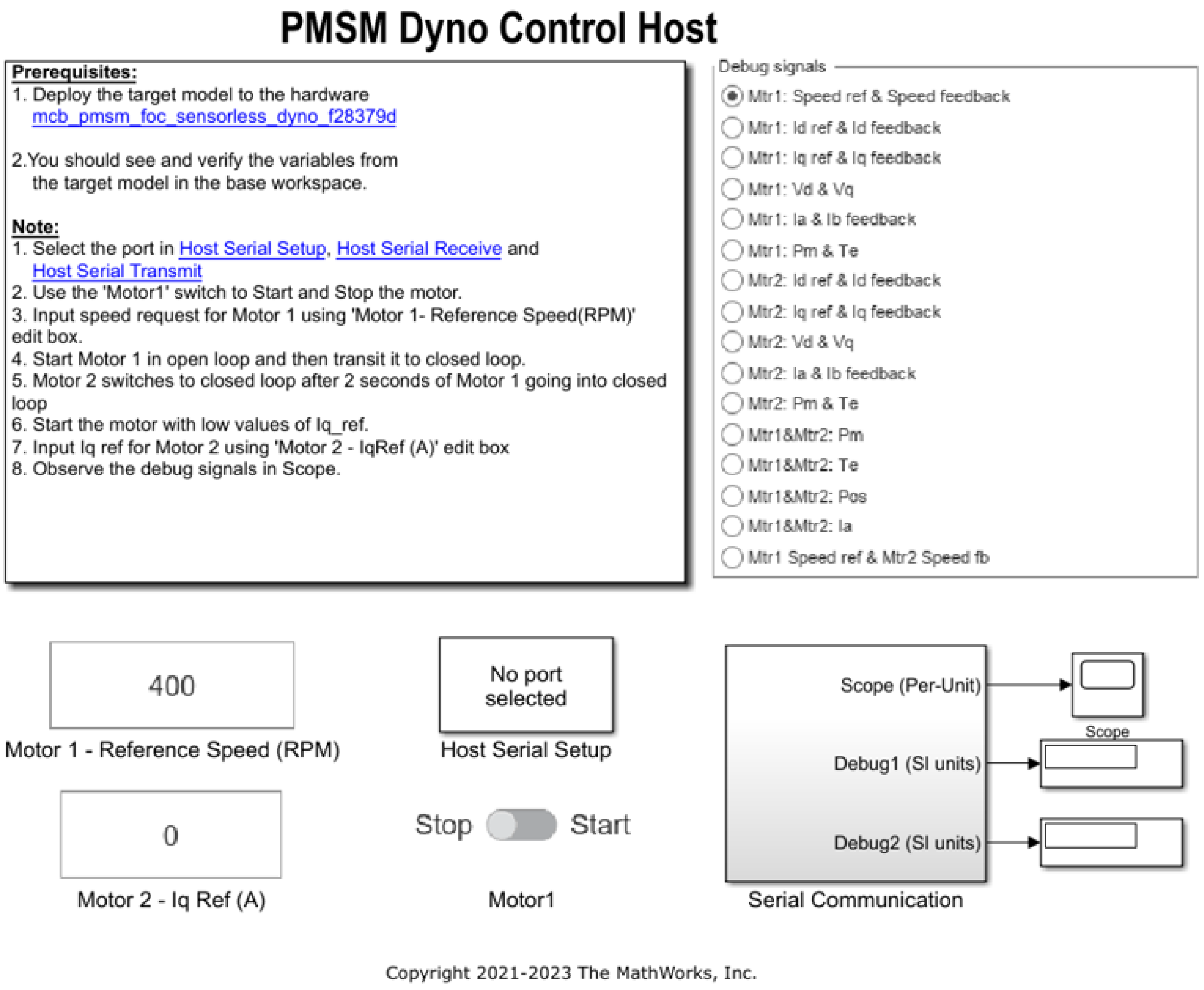
- mcb_pmsm_foc_dual_host_model.

- mcb_pmsm_foc_host_model_dyno_f28379d.

- mcb_pmsm_foc_host_model_dyno_f28069m.

You can also use the open_system command to open the host model:

```
open_system('mcb_pmsm_foc_host_model_dyno_f28379d.slx');
```



PMSM Dyno Control Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_foc_sensorless_dyno_f28379d

2.You should see and verify the variables from
   the target model in the base workspace.

Note:
1. Select the port in Host Serial Setup, Host Serial Receive and
   Host Serial Transmit
2. Use the 'Motor1' switch to Start and Stop the motor.
3. Input speed request for Motor 1 using 'Motor 1- Reference Speed(RPM)' edit box.
4. Start Motor 1 in open loop and then transit it to closed loop.
5. Motor 2 switches to closed loop after 2 seconds of Motor 1 going into closed loop
6. Start the motor with low values of Iq_ref.
7. Input Iq ref for Motor 2 using 'Motor 2 - IqRef (A)' edit box
8. Observe the debug signals in Scope.

Debug signals
- [✓ Mtr1: Speed ref & Speed feedback]
- Mtr1: Id ref & Id feedback
- Mtr1: Iq ref & Iq feedback
- Mtr1: Vd & Vq
- Mtr1: Ia & Ib feedback
- Mtr1: Pm & Te
- Mtr2: Id ref & Id feedback
- Mtr2: Iq ref & Iq feedback
- Mtr2: Vd & Vq
- Mtr2: Ia & Ib feedback
- Mtr2: Pm & Te
- Mtr1&Mtr2: Pm
- Mtr1&Mtr2: Te
- Mtr1&Mtr2: Pos
- Mtr1&Mtr2: Ia
- Mtr1 Speed ref & Mtr2 Speed fb

Motor 1 - Reference Speed (RPM): 400

Host Serial Setup: No port selected

Motor 2 - Iq Ref (A): 0

Motor1: Stop / Start

Copyright 2021-2023 The MathWorks, Inc.

9. Set the parameter **Port** of the following blocks in the mcb_pmsm_foc_host_model_dyno_f28379d model to match the COM port of the host computer. Similarly you can perform the same configuration for other host models in this example:

- mcb_pmsm_foc_host_model_dyno_f28379d > Host Serial Setup
- mcb_pmsm_foc_host_model_dyno_f28379d > Serial Communication > Host Serial Receive
- mcb_pmsm_foc_host_model_dyno_f28379d > Serial Communication > SCI_TX > Host Serial Transmit

10. Click **Run** on the **Simulation** tab to run the host model.

11. Change the position of the **Motor 1** switch to Start, to run the motor.

12. Update the **Motor 1 - Reference Speed (RPM)** and **Motor 2 - Iq Ref (A)** in the host model.

13. Select the debug signals that you want to monitor, to observe them in the Time Scope block of host model.

Other Things to Try

You can also use SoC Blockset™ to develop a real-time motor control application for a dual motor setup that utilizes multiple processor cores to obtain design modularity, improved controller performance, and other design goals. For details, see “Integrate MCU Scheduling and Peripherals in Motor Control Application” on page 4-202.

Partition Motor Control for Multiprocessor MCUs

This example shows how to partition real-time motor control application on to multiple processors to achieve design modularity and improved control performance.

Many MCUs provide multiple processor cores. These additional cores can be leveraged to achieve a variety of design goals:

- Divide the application into real-time tasks, such as control laws, and non-real time tasks, such as external communication, diagnostics, or machine learning
- Partition the control algorithm to run on multiple CPUs to achieve higher loop rate
- Run the same application in multiple CPU's for safety critical applications

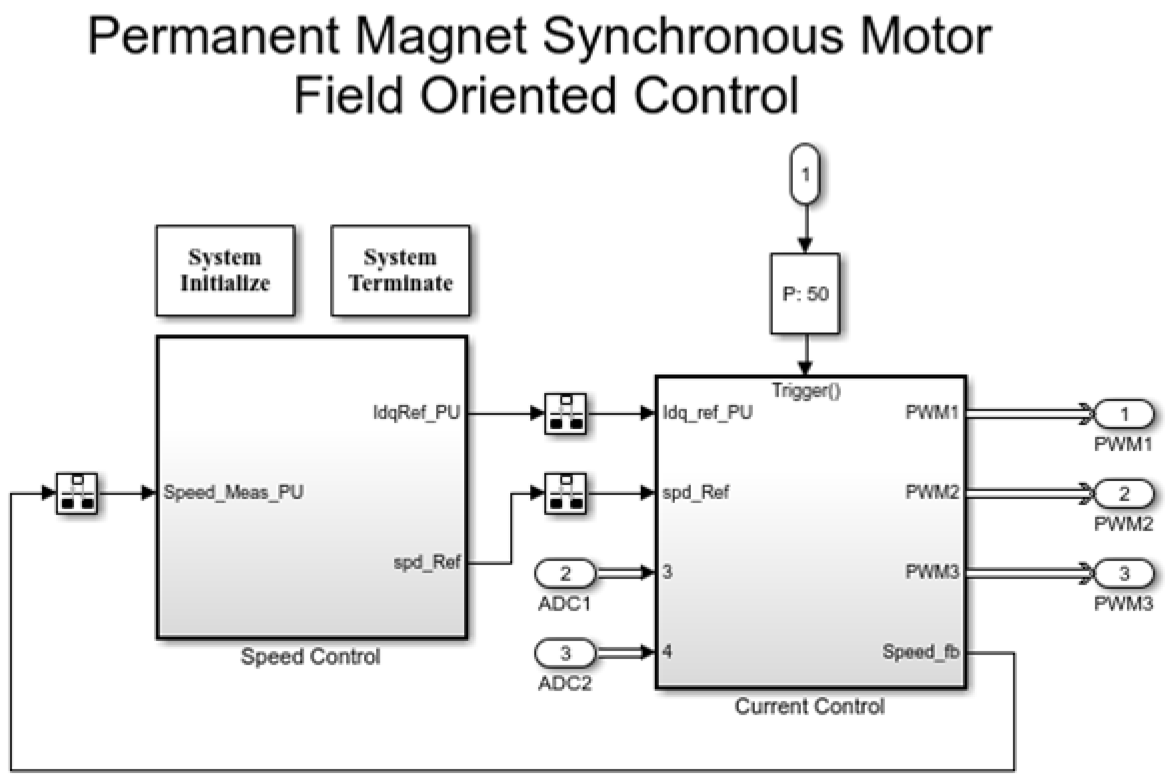
This example shows how to partition motor control application across two CPUs of the TI Delfino F28379D to achieve higher sampling time/PWM frequency.

Required hardware:

- TI Delfino F28379D LaunchPad or TI Delfino F2837xD based board
- BOOSTXL-DRV8305EVM motor driver board
- Teknic M-2310P-LN-04K PMSM motor

Partition Motor Control Algorithm

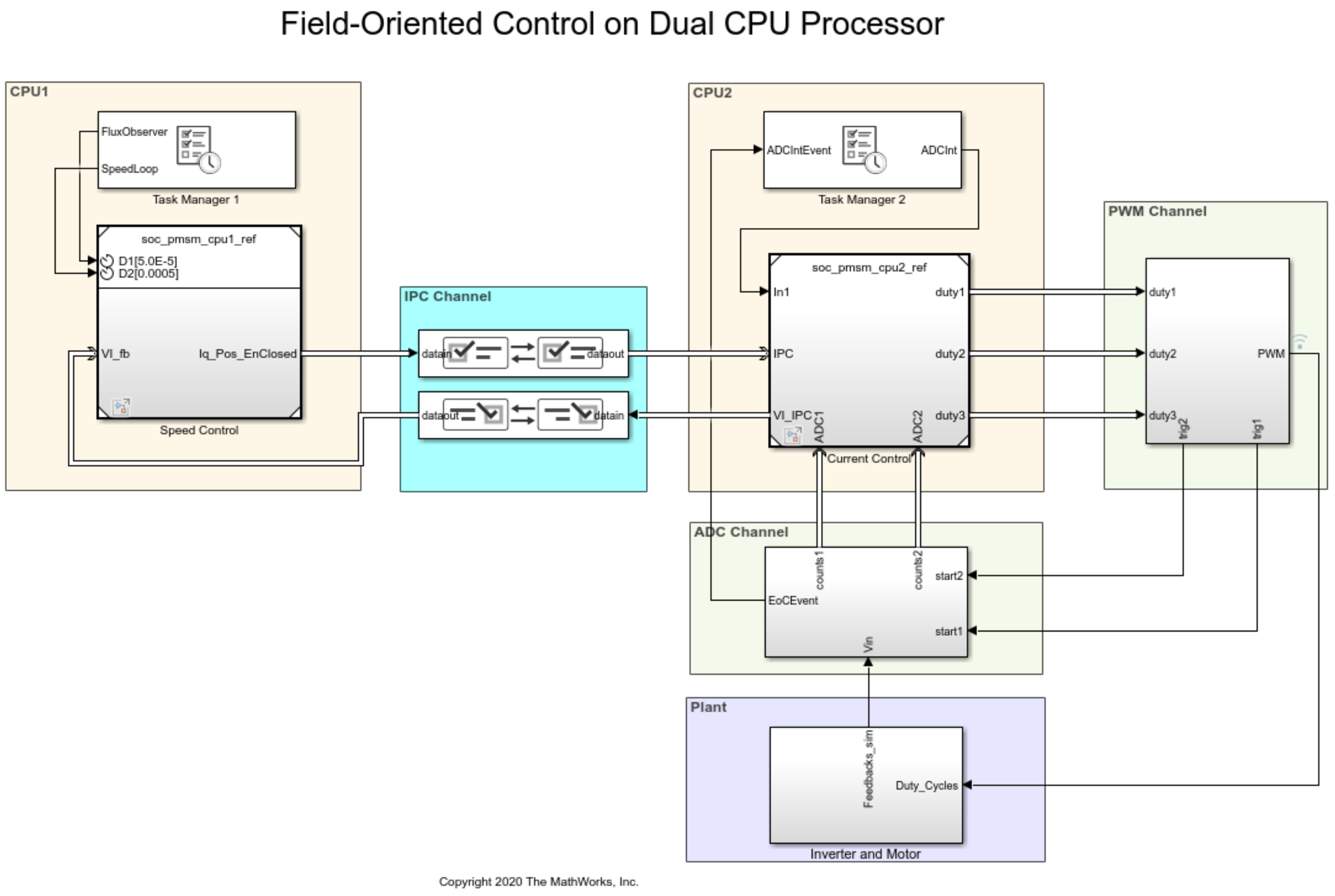
Open the soc_pmsm_singlecpu_foc model. This model simulates a single CPU motor controller, contained in the soc_pmsm_singlecpu_ref model, for a permanent magnet synchronous machine (PMSM).



We partition the control algorithm by executing current control on CPU2, and speed control and position estimation on CPU1 respectively. Data transfer between the CPU's are handled by Interprocess Data Channel block. For more information see “Interprocess Data Communication via Dedicated Hardware Peripheral” (SoC Blockset).

Open the soc_pmsm_dualcpu_foc model.

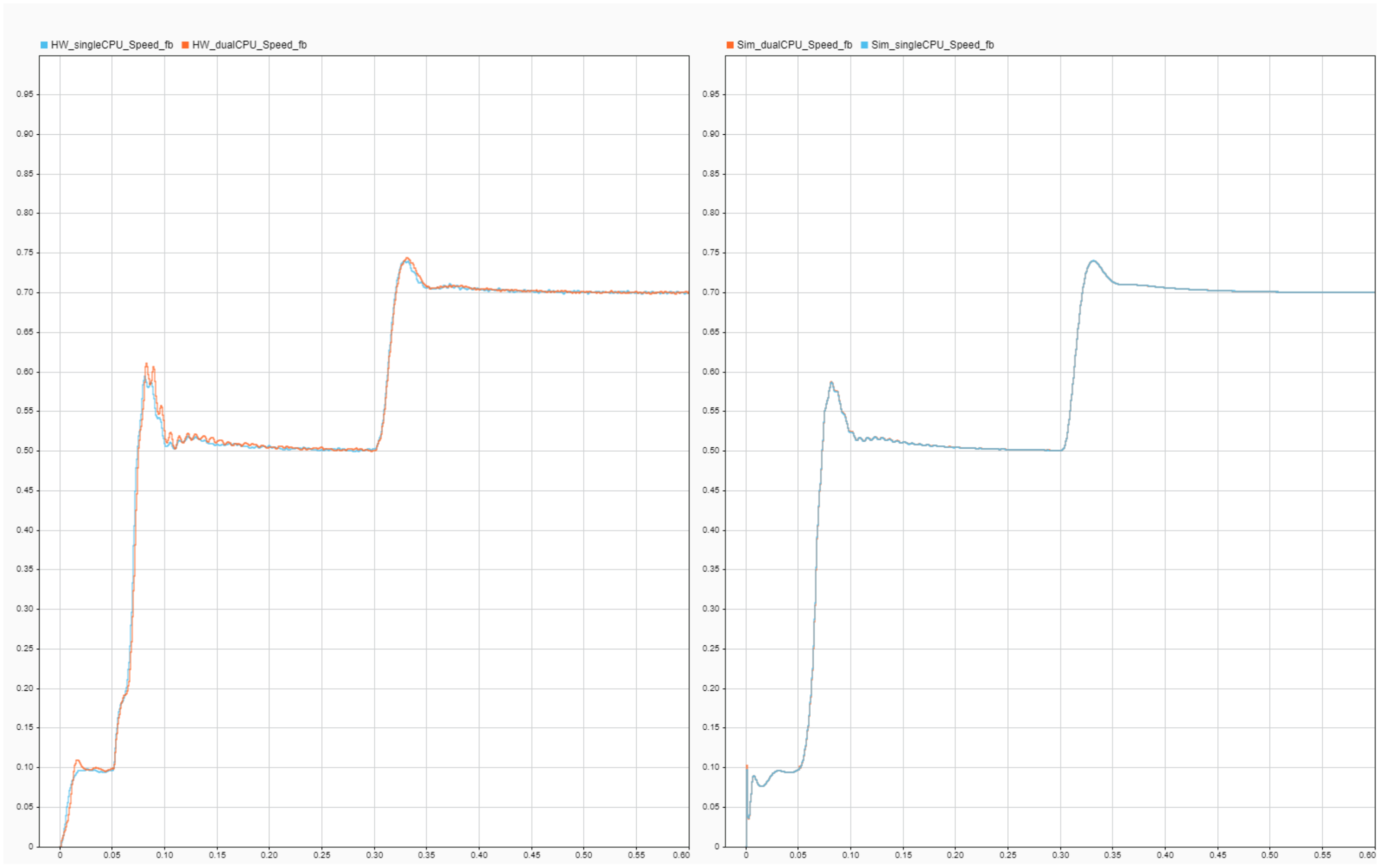
```
open_system('soc_pmsm_dualcpu_foc');
```



On the **System on Chip** tab, click **Hardware Settings** to open the **Configuration Parameters** window. In the **Hardware Implementation** tab, the **Processing Unit** parameter is configured to "None" indicating it is the top-level system model.

Open the soc_pmsm_cpu1_ref model and open the soc_pmsm_cpu2_ref model to view algorithms configured for each CPU. Model references contained within the system model are configured to run on c28xCPU1 (CPU1) and c28xCPU2 (CPU2).

On the Simulation tab, click **Run** to simulate the model. Open the Simulation Data Inspector and view signals. This figure shows results from the single and dual CPU models in simulation and deployment.



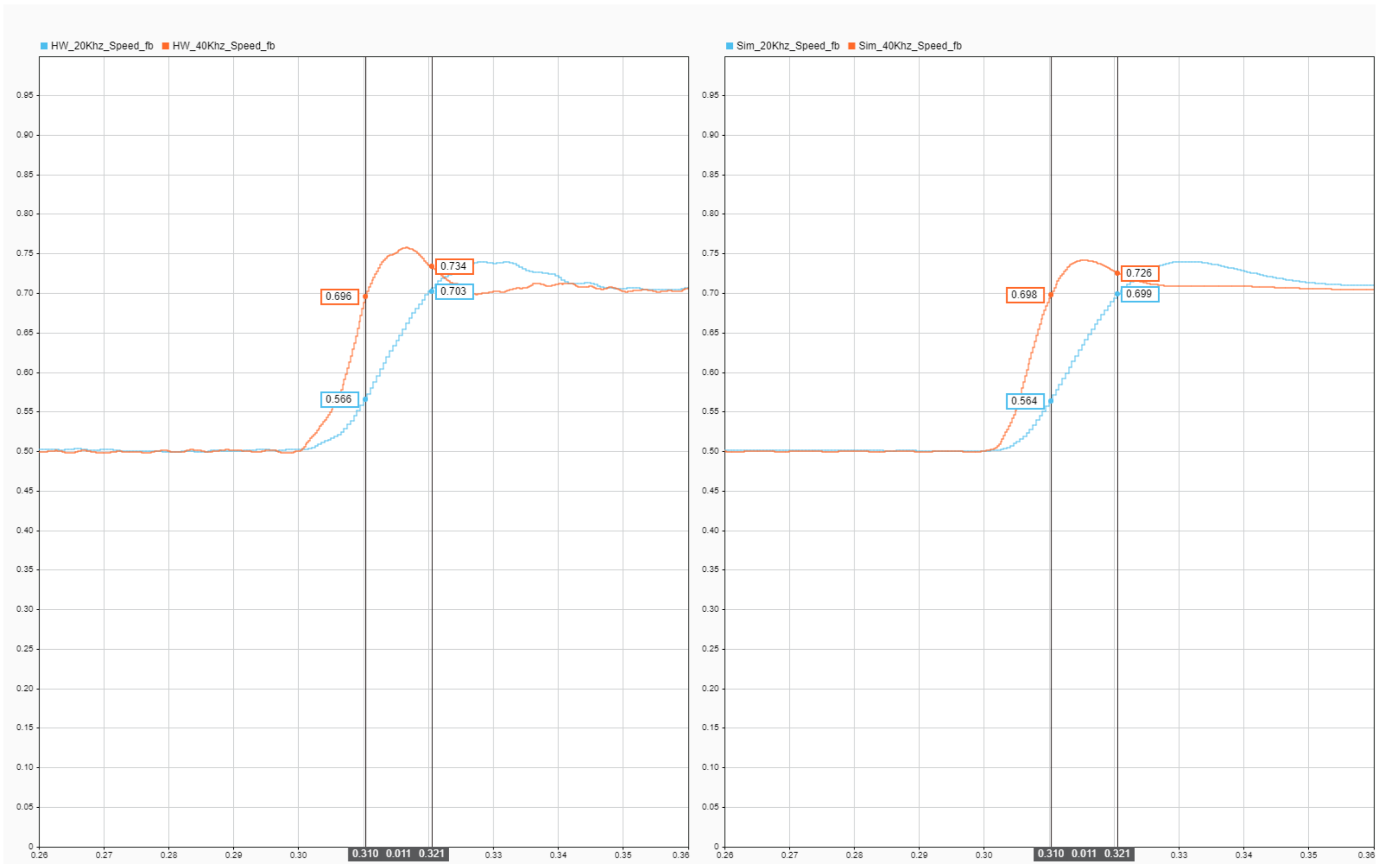
Performance Improvement with Concurrent Execution

Using both the CPUs to execute control algorithms allows us to achieve higher controller bandwidth. In the original single CPU model, the control algorithm takes just over 25us to execute. To provide a safety margin, single CPU model uses a PWM frequency of 20kHz, equivalent to 50us period.

After partitioning, the CPU1 and CPU2 execution times reduce to less than 20us. Allowing the PWM frequency to be increase to 40kHz. In the `soc_mcb_pmsm_foc_sensorless_f28379d_data.m` script, set PWM_frequency to 40e3 and run the script to configure the model to the new PWM frequency. With faster sampling of currents, controller gains can then be tuned to achieve faster response times.

Deploy the model to the TI Delfino F28379D LaunchPad using the SoC Builder (SoC Blockset) tool. To open the tool, on the **System on Chip** tab, click **Configure, Build, & Deploy** and follow the guided steps.

This figure shows the controller response from simulation and deployment at 25us current loop with 40kHz PWM frequency compared with 50us current loop at 20kHz frequency. As expected, the rise time in speed improves with faster current loop by approximately 50 percent.



Speed response is oscillatory because of sensorless algorithm, for more information see “Sensorless Field-Oriented Control of PMSM” (Motor Control Blockset)

For higher simulation granularity, set the PWM Interface block output to Switching Mode and change the plant model variant to use the MOSFET simulation.

See Also

- “Get Started with Multiprocessor Blocks on MCUs” (SoC Blockset)
- “Integrate MCU Scheduling and Peripherals in Motor Control Application” (Motor Control Blockset)

Permanent Magnet Synchronous Motor Field-Oriented Control Using Concerto Processors

This example shows how to control the speed of a three-phase Permanent Magnet Synchronous Motor in a closed-loop fashion via Field-Oriented Control using the C28x peripherals and MCB blocks from Motor Control Blockset™ library. For the complete hardware in the loop (HIL) PMSM example, see “Field-Oriented Control (FOC) of PMSM Using Hardware-In-The-Loop (HIL) Simulation” (Motor Control Blockset).

Required Hardware

This example supports the following hardware configuration:

DRV8312 Configuration:

- TI® DRV8312 Three-Phase Brushless Motor Control Kit (DRV8312-C2-KIT or DRV8312-69M-KIT) with F28M35x or F28M36x Concerto processor Note: To use the DRV8312 kit with F28M36x, you need an additional hardware for converting 180DIMM of F28M36x to 100 DIMM slot.
- Three-phase PMSM with optional Hall sensors attached to connector J10 of the DRV8312EVM board

For connections related to the preceding hardware configuration, see “Hardware Connections” on page 1-81

Note: Match the characteristics of the power supply and the amplifier with the input characteristics of the selected motor.

Available models of this example:

- DRV8312EVM + F28M35x Concerto: mcb_pmsm_foc_sensorless_f28m35x
- DRV8312EVM + TMSADAP180TO100 + F28M36x Concerto: mcb_pmsm_foc_sensorless_f28m36x

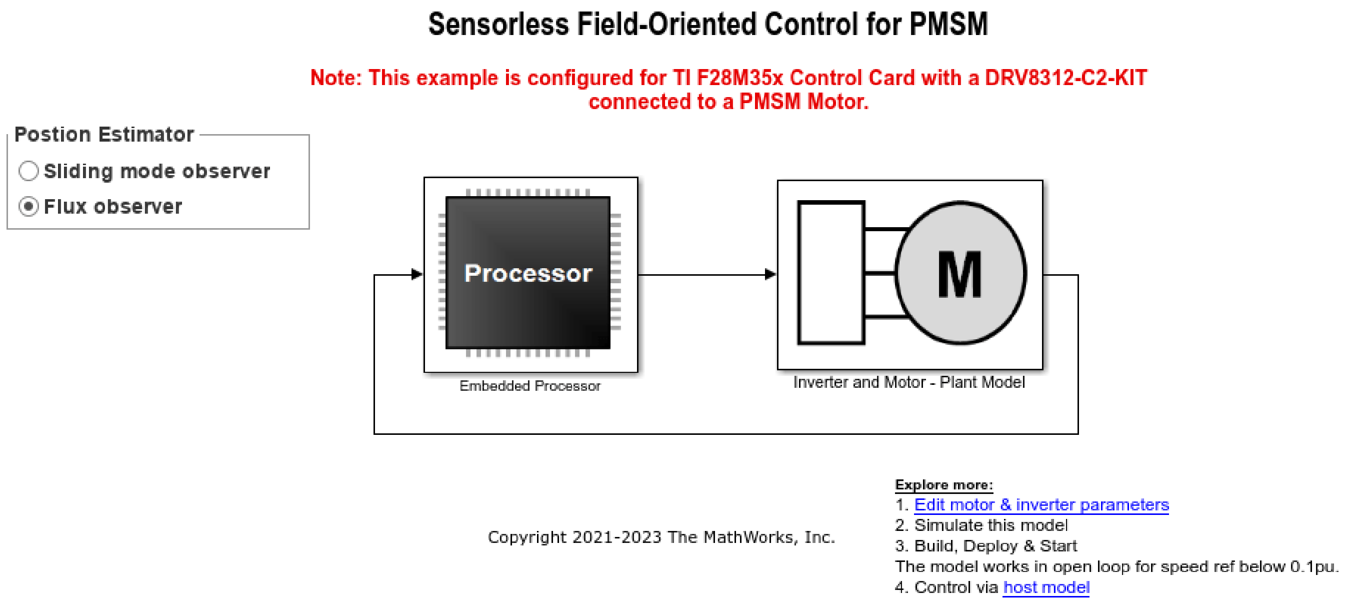
Prerequisites

For the model to work on the C28x CPU of Concerto Processor, the ARM core must be running already and all GPIOs necessary for C28x must be allocated to C28x from the ARM Cortex M3 CPU. For more information, see “Getting Started with C2000 Microcontroller Blockset for F28M3x Concerto Processors” on page 4-287.

Model

The following figure shows the Permanent Magnet Synchronous Motor Field-Oriented Control example model.

```
close
open_system('mcb_pmsm_foc_sensorless_f28m35x');
```


Sensorless Field-Oriented Control for PMSM
Note: This example is configured for TI F28M35x Control Card with a DRV8312-C2-KIT connected to a PMSM Motor.
Postion Estimator
Sliding mode observer
Flux observer
Processor
Embedded Processor
M
Inverter and Motor - Plant Model
Copyright 2021-2023 The MathWorks, Inc.
Explore more:
1. Edit motor & inverter parameters
2. Simulate this model
3. Build, Deploy & Start
The model works in open loop for speed ref below 0.1pu.
4. Control via host model

Postion Estimator: ( ) Sliding mode observer [✓ Flux observer]

Description

This example shows how to use a closed-loop Field-Oriented Control (FOC) algorithm to regulate the speed and torque of a three-phase permanent magnet synchronous motor (PMSM). To run this example, there needs to be an application code running on the ARM Cortex M3 CPU and allocate all GPIOs required for the C28x CPU core. This example uses C28x peripheral blocks from the F28M35x or F28M36x C28x Library blocks from the C2000™ Microcontroller Blockset and MCB blocks from Motor Control Blockset™ library.

The algorithm in this example uses an asynchronous scheduling. The pulse width modulation (PWM) block triggers the ADC conversion. At the end of conversion, the ADC posts an interrupt that triggers the main FOC algorithm.

The model is set up for motors that have the following characteristics:

DRV8312 Configuration:

- 4 pole pairs
- Sensorless using sliding mode observer (SMO)

You may need to change the model parameters to match to your specific motor. Match motor voltage and power characteristics to the controller.

A conventional voltage-source inverter drives the motor. The controller algorithm generates six pulse width modulation (PWM) signals using a vector PWM technique for six power switching devices. The inverter measures the current of the two motor input currents (ia and ib) of the motor using two analog-to-digital converters (ADCs) and sends the measurements to the processor.

```
close_system('c28M36xpmsmfoc_ert', 0);
```

How to Run the Example with Code Composer Studio v5.5/v6

- Open the example model that matches your hardware.

- Open Model Configuration parameters and select the required tool chain under Code Generation pane. Before running the code on the C28x core, make sure that code is running on the ARM Cortex M3 CPU. Refer to “Getting Started with C2000 Microcontroller Blockset for F28M3x Concerto Processors” on page 4-287.

For More Information on Motor Control kits supported on other C28x Targets, refer to “Sensorless Field-Oriented Control of PMSM Using C2000 Processors” on page 4-24

- Default settings are set to execute the code from RAM on C28x Core
- Press Ctrl+B to build a binary executable from example model.
- Ctrl+B automatically loads and runs the executable on the target selected.

High power must be applied to the Inverter before running the program. Stopping the program in the middle of its execution with high power on can damage the hardware. Use "Reset" to stop the execution of the program.

Integrate MCU Scheduling and Peripherals in Motor Control Application

This example shows how to identify and resolve issues with respect to peripheral settings and task scheduling early during development.

The following are typical challenges associated with MCU peripherals and scheduling:

- ADC-PWM synchronization to achieve current sensing at mid point of PWM period
- Incorporate sensor delays to achieve the desired controller response for the closed loop system
- Studying different PWM settings while designing special algorithms

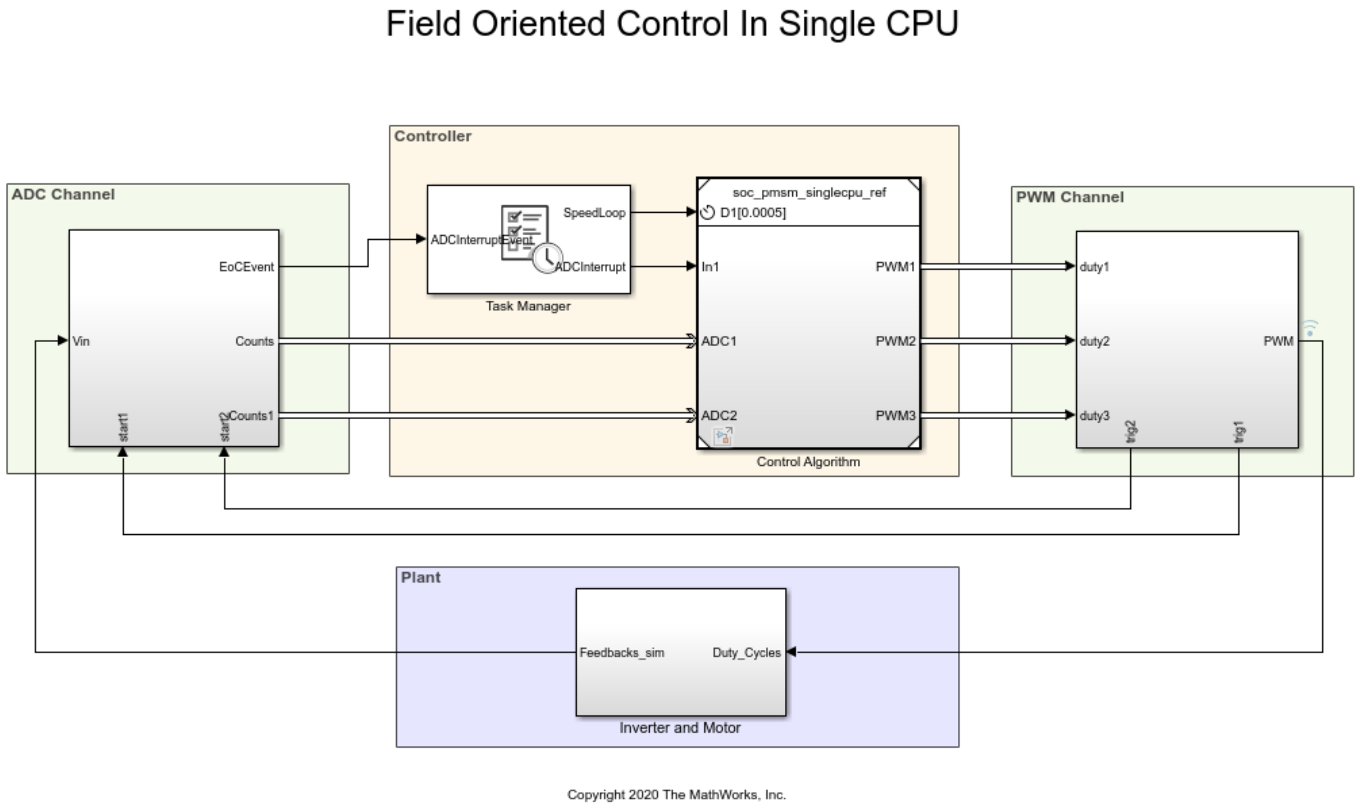
This example shows how to use SoC Blockset to address these challenges for a motor control closed-loop application in simulation and verify on hardware by deploying on to the TI Delfino F28379D LaunchPad.

Required hardware:

- TI Delfino F28379D LaunchPad or TI Delfino F2837xD based board
- BOOSTXL-DRV8305EVM motor driver board
- Teknic M-2310P-LN-04K PMSM motor

Model Structure

```
open_system('soc_pmsm_singlecpu_foc');
```

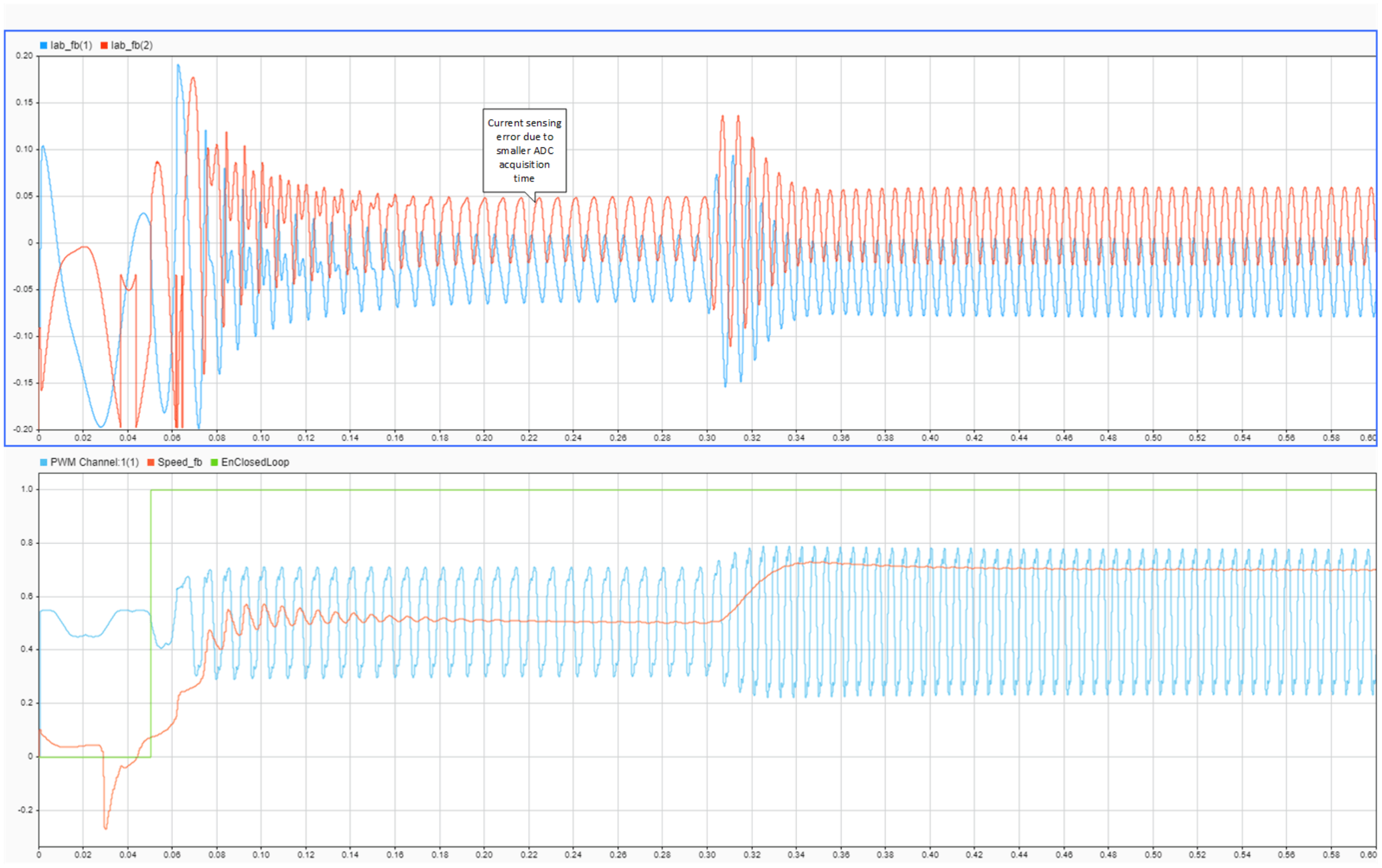


Open the soc_pmsm_singlecpu_foc model. This model simulates single CPU motor controller, contained in soc_pmsm_singlecpu_ref model, for a Permanent magnet synchronous motor inverter system. Controller senses the outputs from the plant using ADC Interface (SoC Blockset) and actuates using PWM Interface (SoC Blockset) that drives the inverter. Algorithm blocks from Motor Control Blockset™ is used in this example.

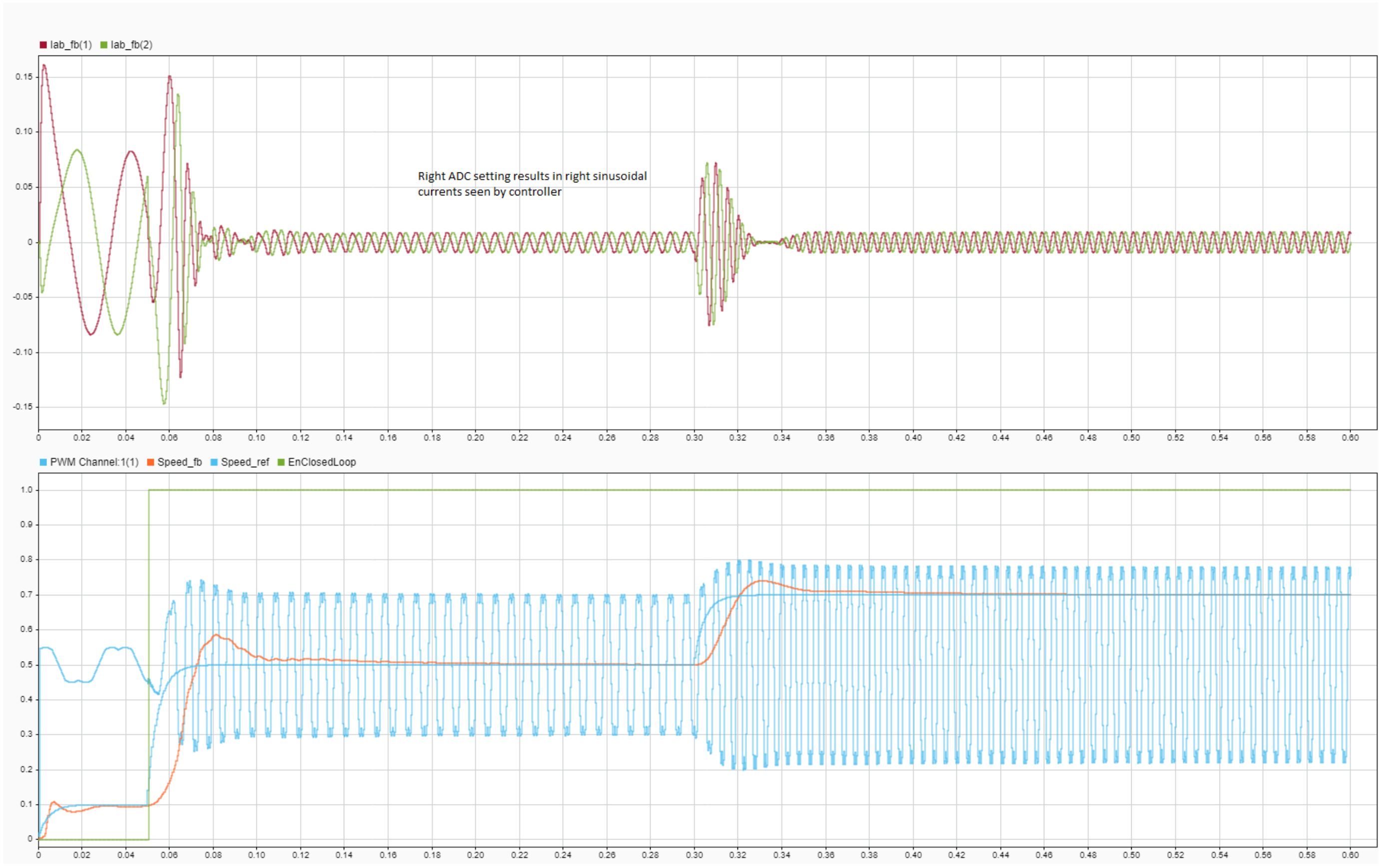
ADC Acquisition Time

ADC hardware contains a sample and hold circuit to sense the analog inputs. To ensure complete ADC measurement, the minimum acquisition time must be selected to account for the combined effects of input circuit and the capacitor in the sample and hold circuit.

Open ADC Interface block and change the default acquisition time to 100ns. Run the simulation and view the results in Simulation Data Inspector and observe there is a distortion in current waveforms. The low acquisition time resulted in ADC measurements not reaching their true value. As a result, the controller reacts by generating a relative duty cycle causing variations in current drawn by the motor. These figures show the reaction to the incorrect ADC measurement and overdraw in the phase A current channel, with phase A current in blue and phase B current in orange. The simulated speed feedback shows significant oscillations during open loop to closed loop transition, which in real world will halt the motor.

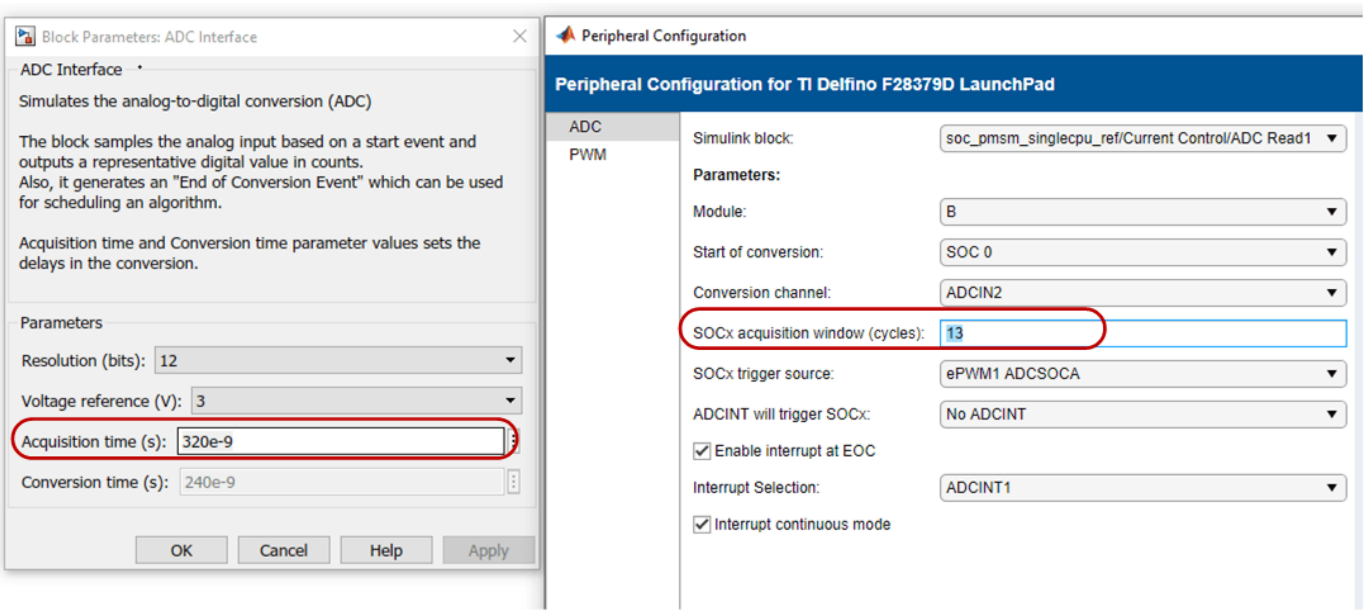


To fix this issue, open ADC Interface blocks change and change acquisition time to a larger value, 320ns. This value is above the minimum ADC acquistion time recommended in section ADC Operating Conditions (12-Bit Single-Ended Mode) of the TI Delfino F28379D LaunchPad data sheet. Run the simulation and view the results in Simulation Data Inspector. This figure shows the accurately sampled ADC values and the controller tracking the reference value as expected.



Verify the simulation results against hardware by deploying the model to the TI Delfino F28379D LaunchPad. On the **System on Chip** tab, click **Configure, Build, & Deploy** to open the SoC Builder (SoC Blockset) tool.

In the SoC Builder tool, on **Peripheral Configuration** tool, set **ADC > SOCx acquisition window cycles** parameter to `13 ADC clock ticks` for the ADC B and C modules. The ADC acquisition clock ticks parameter must be set to the simulation time value, set in the ADC Interface block, multiplied by the ADC clock frequency. You can get the ADC clock frequency from the model hardware settings. Open the soc_pmsm_singlecpu_ref model. On the **System on Chip** tab, click **Hardware Settings** to open the **Configuration Parameters** window. In the **Hardware Implementation > Target hardware resources > ADC_x** section, you can see the `ADC clock frequency in MHz` parameter value. This figure shows the ADC Interface block setting for simulation and peripheral app setting for deployment. Use same setting in simulation and codegen to ensure expected behavior.


Block Parameters: ADC Interface

ADC Interface

Simulates the analog-to-digital conversion (ADC)

The block samples the analog input based on a start event and outputs a representative digital value in counts.
Also, it generates an "End of Conversion Event" which can be used for scheduling an algorithm.

Acquisition time and Conversion time parameter values sets the delays in the conversion.

Parameters

Resolution (bits): 12

Voltage reference (V): 3

Acquisition time (s): 320e-9

Conversion time (s): 240e-9

Peripheral Configuration

Peripheral Configuration for TI Delfino F28379D LaunchPad

Simulink block: soc_pmsm_singlecpu_ref/Current Control/ADC Read1

Parameters:

Module: B

Start of conversion: SOC 0

Conversion channel: ADCIN2

SOCx acquisition window (cycles): 13

SOCx trigger source: ePWM1 ADCSOCA

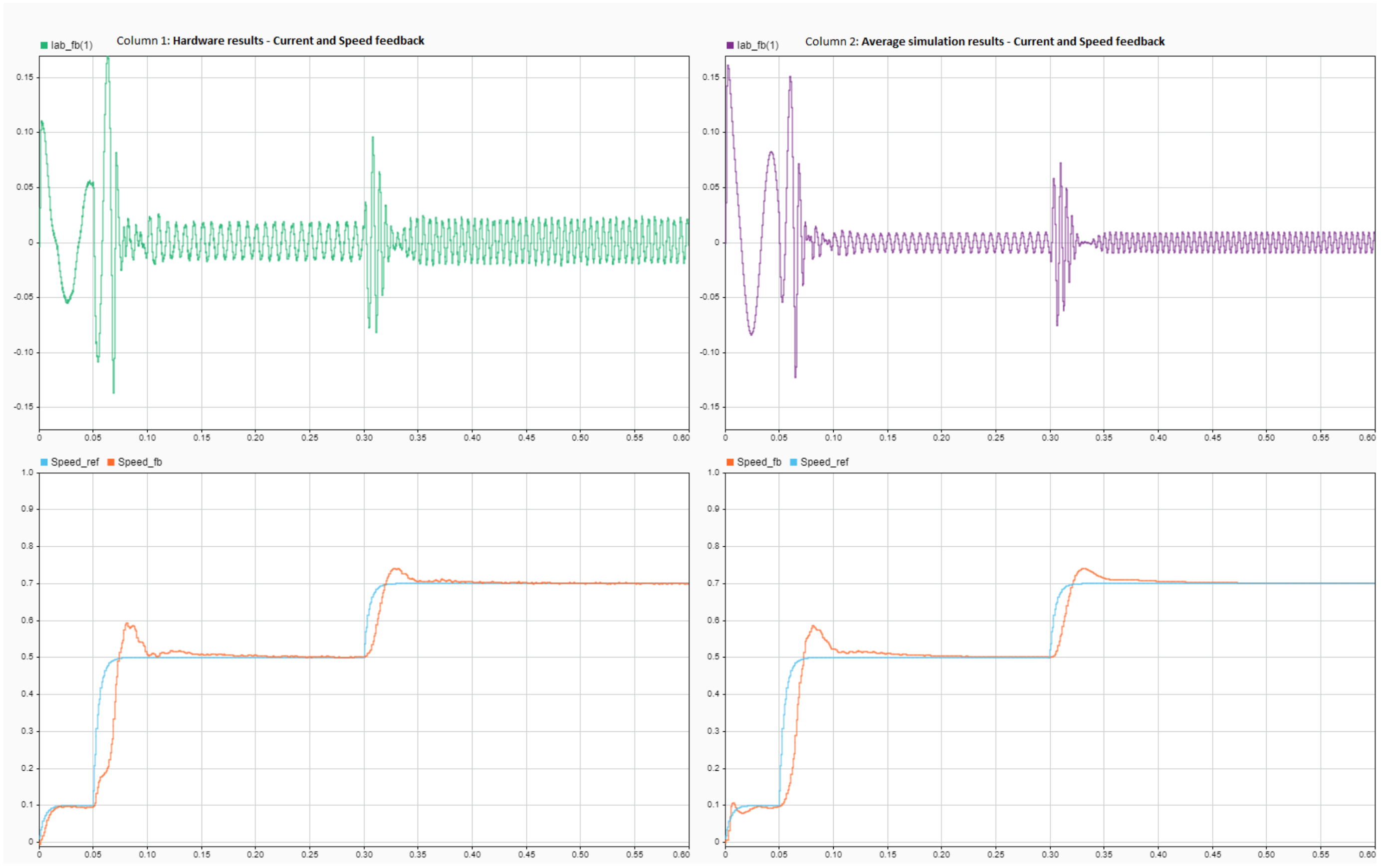
ADCINT will trigger SOCx: No ADCINT

✓ Enable interrupt at EOC

Interrupt Selection: ADCINT1

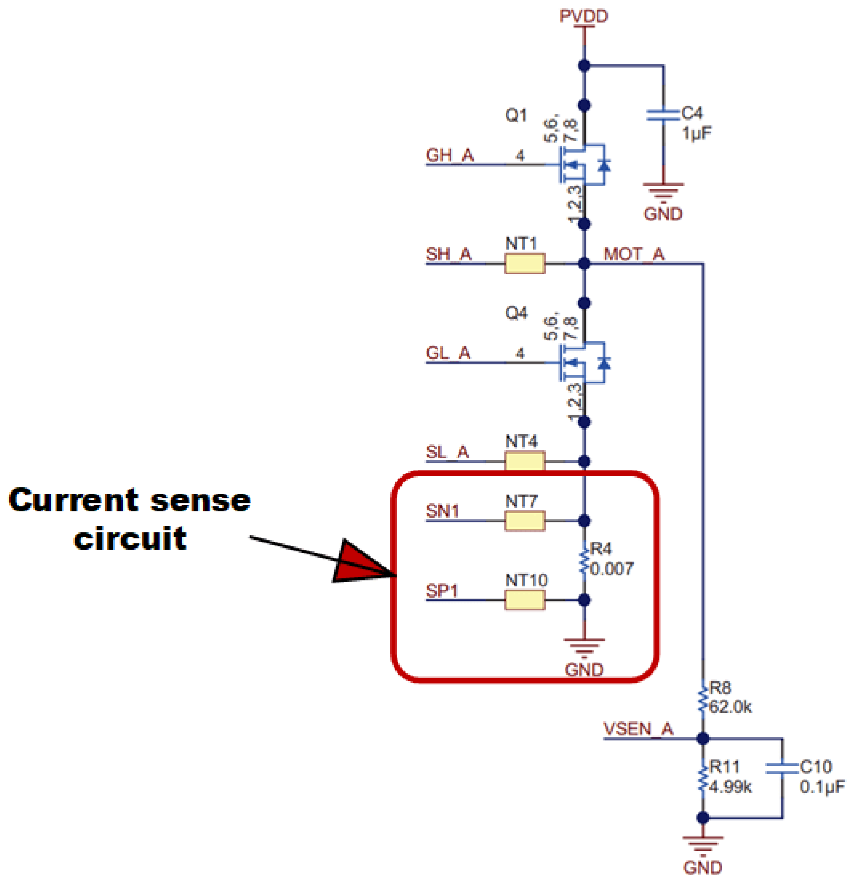
✓ Interrupt continuous mode

On **Select Build Action** page, to monitor data from hardware select `Build and load for External mode`. This figure shows the data from hardware with accurately sampled ADC values and the controller tracking the reference value as expected.



ADC-PWM Synchronization

The BOOSTXL-DRV8305EVM motor driver has a 3-phase inverter built using 6 power MOSFETS. This motor driver board uses a low-side shunt resistor to sense motor currents. The Current sense circuit amplifies the voltage drop across the shunt. This setup ensures low power dissipation, since the current only flows through the shunt when the bottom switches are on and away from PWM commutation noise. This figure shows the low-side shunt resistor circuit in BOOSTXL-DRV8305EVM motor drive.



For correct operation, current sensing must occur during the mid point of the PWM period when ADCs trigger. Specifically, the PWM counter must be at the maximum value when the bottom switches are active in the Up-Down counter mode. Current sampling at a different instance results in a measured currents of zero.

To analyze this case, switch the model to **high fidelity inverter simulation** mode. Change the plant variant to use detailed MOSFET based 3-phase inverter to replicate BOOSTXL-DRV8305EVM.

```
set_param('soc_pmsm_singlecpu_foc/Inverter and Motor/Average or Switching',...
'LabelModeActivechoice','SwitchingInverter');
```

Change the `Output mode` parameter of PWM Interface (SoC Blockset) to `Switching` and connect 6 PWMs to the Mux block.

```
set_param('soc_pmsm_singlecpu_foc/PWM Channel/PWM Interface', 'OutSigMode', 'Switching');
set_param('soc_pmsm_singlecpu_foc/PWM Channel/PWM Interface1', 'OutSigMode','Switching');
set_param('soc_pmsm_singlecpu_foc/PWM Channel/PWM Interface2', 'OutSigMode', 'Switching');
```

Delete existing connection between PWM Interface block and Mux.

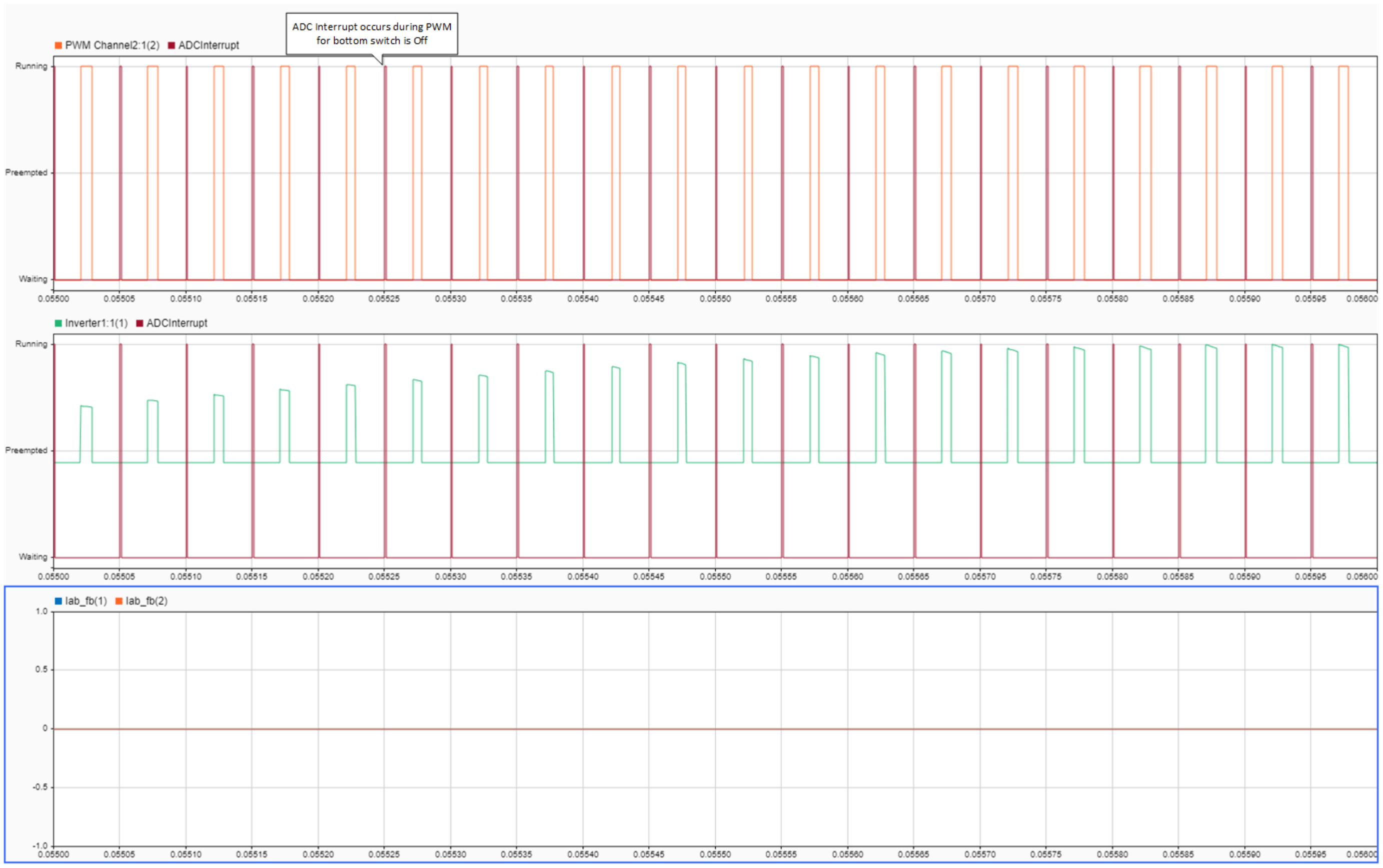
```
h = get_param('soc_pmsm_singlecpu_foc/PWM Channel/Mux','LineHandles');
delete_line(h.Inport);
```

As a last step, connect 6 PWM outputs to Mux.

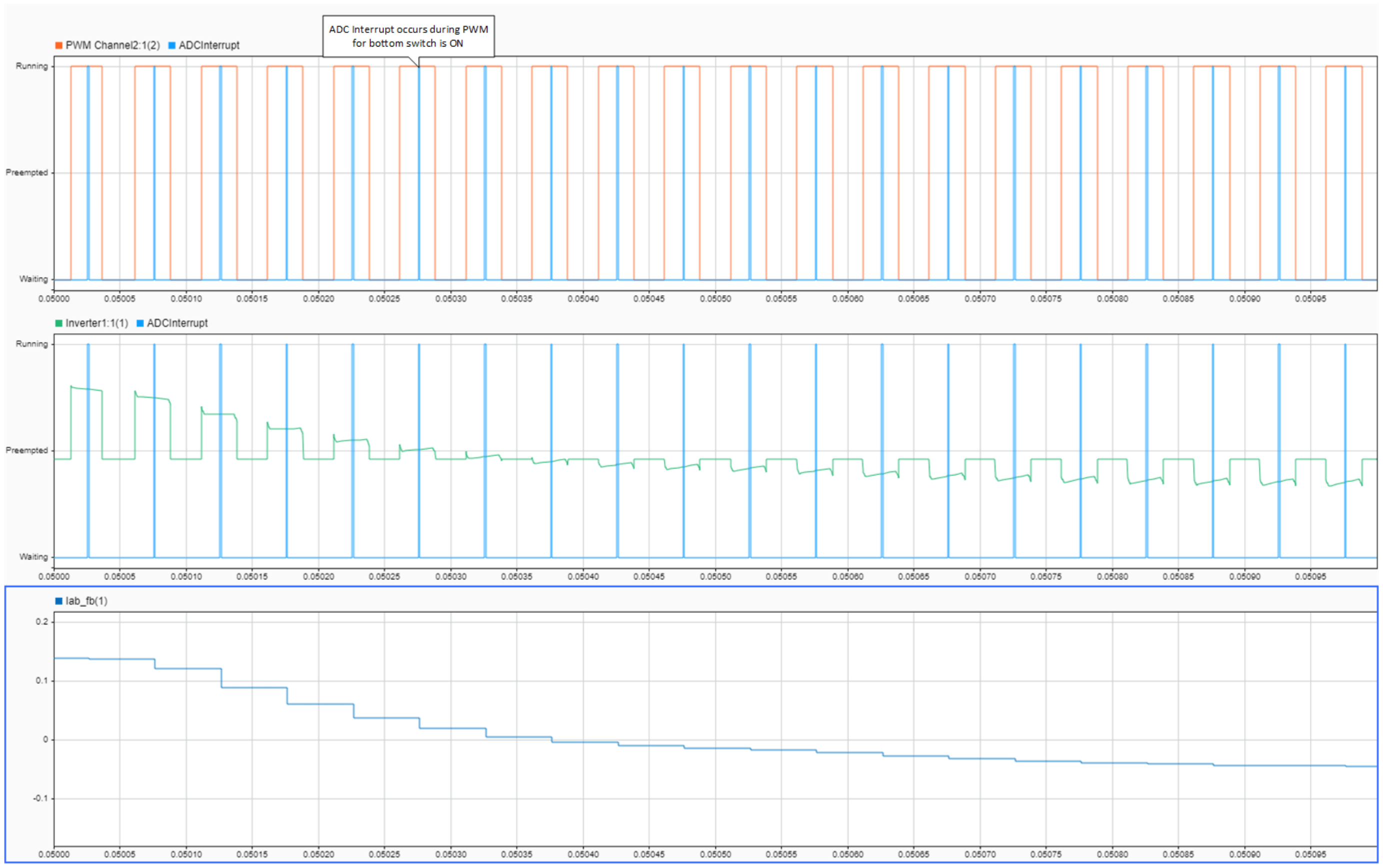
```
set_param('soc_pmsm_singlecpu_foc/PWM Channel/Mux','Inputs','6');
```

```
add_line('soc_pmsm_singlecpu_foc/PWM Channel', ...
{'PWM Interface/1', 'PWM Interface/2', 'PWM Interface1/1',...
'PWM Interface1/2', 'PWM Interface2/1', 'PWM Interface2/2'}, ...
{'Mux/1','Mux/2','Mux/3','Mux/4','Mux/5','Mux/6'}, 'autorouting', 'smart');
```

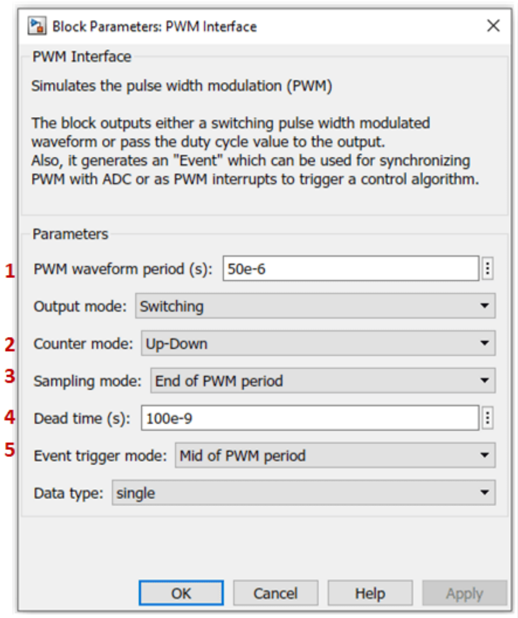
Open the PWM Interface blocks and set **Event trigger mode** to `End of PWM period`. Run the simulation and view the results in Simulation Data Inspector. In the figure, phase A and phase B currents are approximately zero current. This results in a loss of feedback and no actuation in the control loop. Select `Enable task simulation` in Task Manager block to simulate and visualize tasks in Simulation Data Inspector.



To fix this issue, change the **Event trigger mode** to `Mid point of PWM period`, equivalent to the PWM internal counter being at a maximum. Run the simulation and view the results in Simulation Data Inspector.



Deploy the model on to the TI Delfino F28379D LaunchPad using the SoC Builder (SoC Blockset) tool. In the SoC Builder tool, on **Peripheral configuration** tool, set **PWM event condition** to `Counter equals to period`. Use same setting in simulation and codegen to ensure expected behavior. This figure shows the PWM Interface block setting for simulation and the Peripheral Configuration tool setting for deployment.


Block Parameters: PWM Interface

PWM Interface

Simulates the pulse width modulation (PWM)

The block outputs either a switching pulse width modulated waveform or pass the duty cycle value to the output.
Also, it generates an "Event" which can be used for synchronizing PWM with ADC or as PWM interrupts to trigger a control algorithm.

Parameters

1 PWM waveform period (s): 50e-6

Output mode: Switching

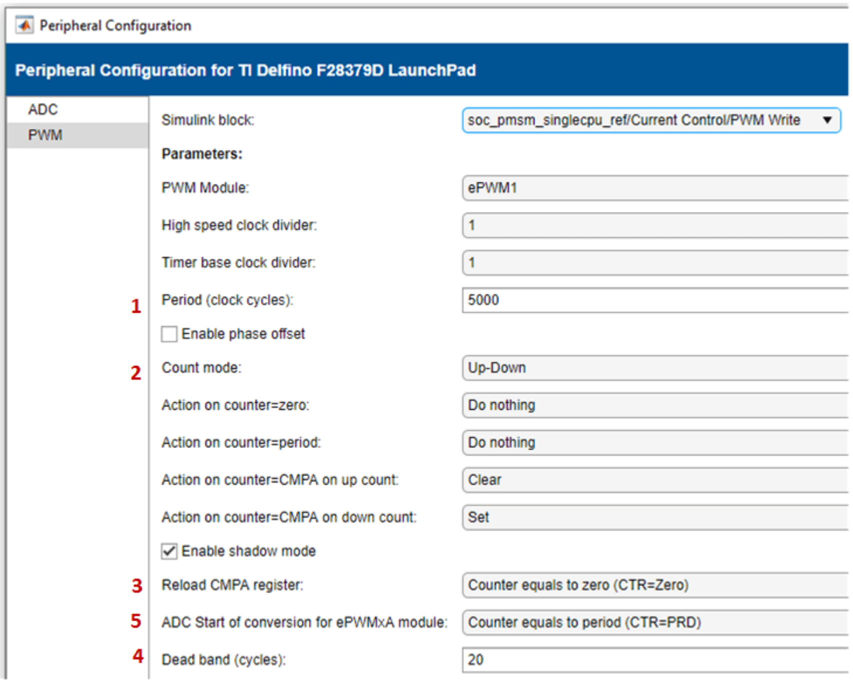
2 Counter mode: Up-Down

3 Sampling mode: End of PWM period

4 Dead time (s): 100e-9

5 Event trigger mode: Mid of PWM period

Data type: single


Peripheral Configuration

Peripheral Configuration for TI Delfino F28379D LaunchPad

Simulink block: soc_pmsm_singlecpu_ref/Current Control/PWM Write

Parameters:

PWM Module: ePWM1

High speed clock divider: 1

Timer base clock divider: 1

1 Period (clock cycles): 5000

☐ Enable phase offset

2 Count mode: Up-Down

Action on counter=zero: Do nothing

Action on counter=period: Do nothing

Action on counter=CMPA on up count: Clear

Action on counter=CMPA on down count: Set

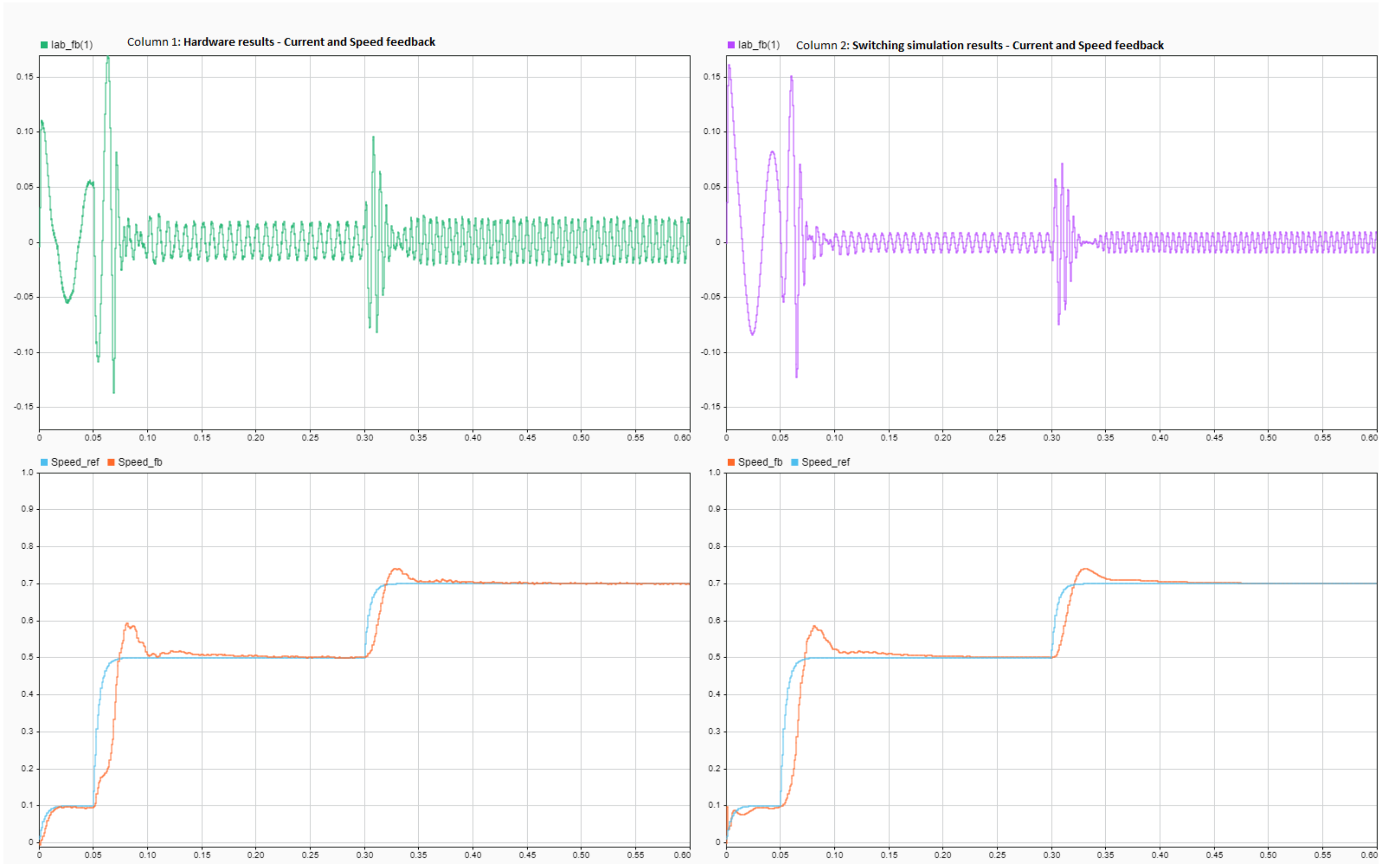
☑ Enable shadow mode

3 Reload CMPA register: Counter equals to zero (CTR=Zero)

5 ADC Start of conversion for ePWMxA module: Counter equals to period (CTR=PRD)

4 Dead band (cycles): 20

This figure shows the data from simulation and hardware with correct ADC-PWM synchronization and the controller tracking the reference value as expected.



See Also

- “Get Started with Multiprocessor Blocks on MCUs” (SoC Blockset)
- “Partition Motor Control for Multiprocessor MCUs” (Motor Control Blockset)

Schedule a Multi-Rate Controller for a Permanent Magnet Synchronous Machine

This example shows how to create a real-time executable for a Texas Instruments™ F28335 embedded target. You will build upon the algorithm specified in the example “Field-Oriented Control of Permanent Magnet Synchronous Machine” (Embedded Coder) by adding timing and peripheral specifications. You will learn an advanced technique to schedule the algorithm based on the periodic end-of-conversion event of the ADC.

Required hardware:

- Spectrum Digital® F28335 eZdsp board
- Digital Motor Controller board: Spectrum Digital DM550
- Three-phase Permanent Magnet Synchronous Motor with quadrature encoder

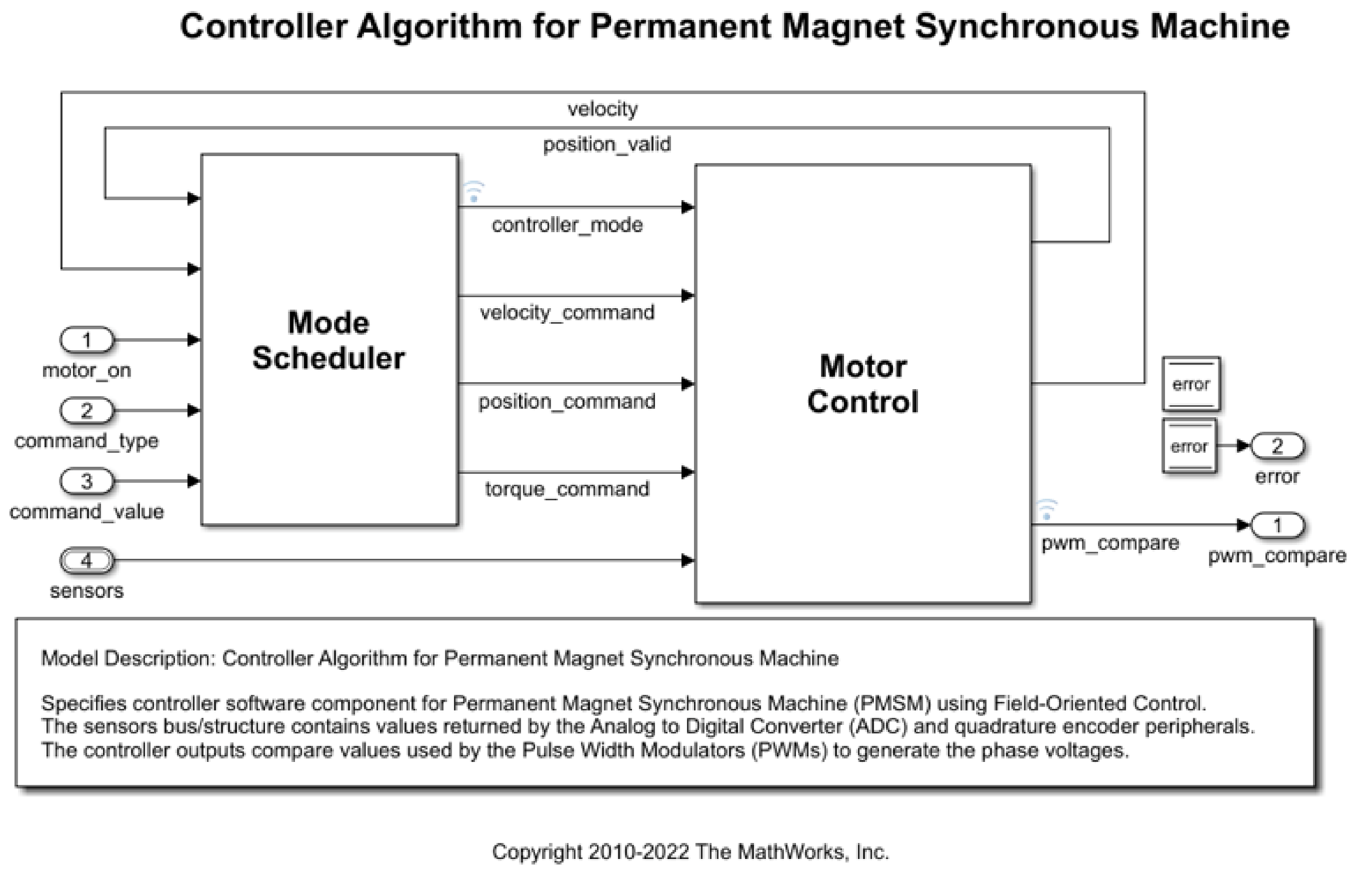
Note: Match the characteristics of the power supply and the amplifier with the input characteristics of the selected motor.

Introduction

The goal is to create a real-time executable for a Spectrum Digital F28335 eZdsp and schedule the controller execution using an ADC end-of-conversion interrupt. This goal is achieved using an advanced technique where the controller and peripherals are scheduled by a periodic ADC end-of-conversion interrupt.

This example builds upon the controller algorithm specified in the example “Field-Oriented Control of Permanent Magnet Synchronous Machine” (Embedded Coder). Refer to that example to explore how the controller algorithm behaves during system simulation with a model of a Permanent Magnet Synchronous Machine.

Configure the Controller Model for the TI F28335 Processor



Controller Algorithm for Permanent Magnet Synchronous Machine

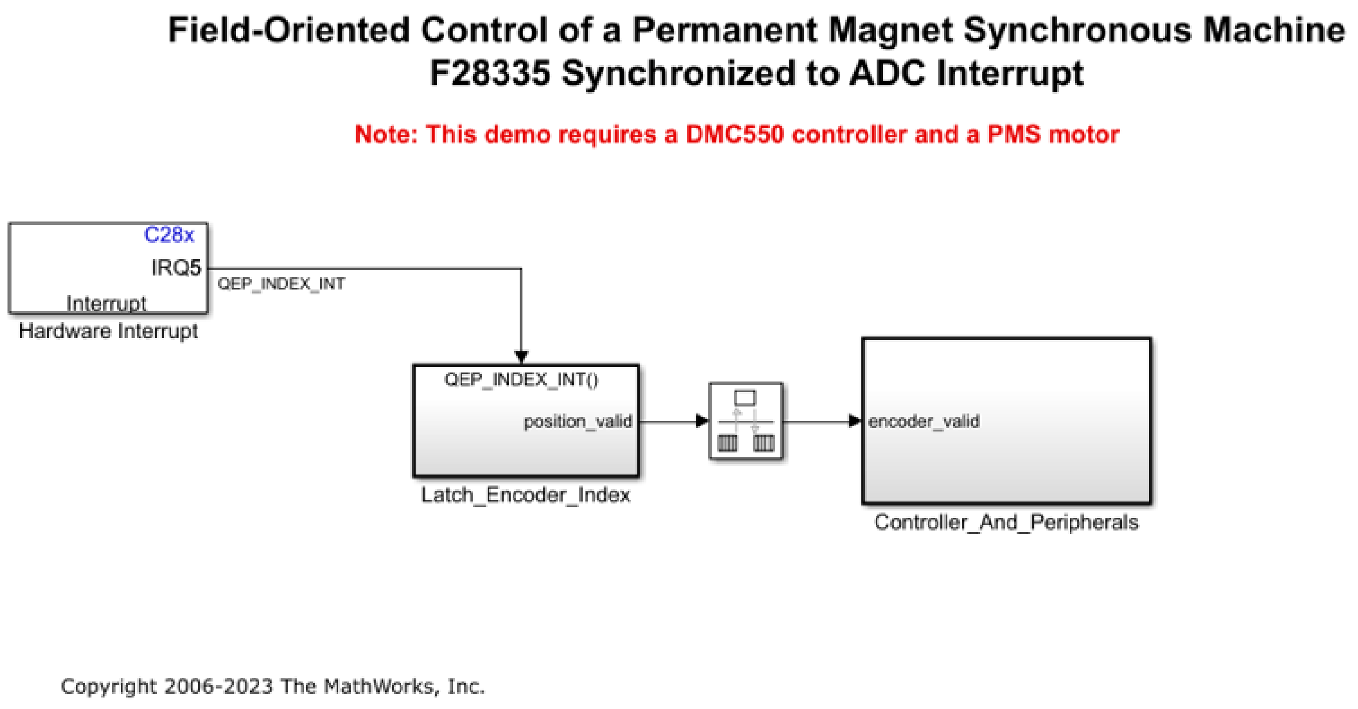
Model Description: Controller Algorithm for Permanent Magnet Synchronous Machine

Specifies controller software component for Permanent Magnet Synchronous Machine (PMSM) using Field-Oriented Control.
The sensors bus/structure contains values returned by the Analog to Digital Converter (ADC) and quadrature encoder peripherals.
The controller outputs compare values used by the Pulse Width Modulators (PWMs) to generate the phase voltages.

Copyright 2010-2022 The MathWorks, Inc.

To facilitate transitioning between simulation and code generation phases of a design, the controller algorithm is specified in its own c28335_pmsmfoc model. The controller algorithm is introduced in the example “Field-Oriented Control of Permanent Magnet Synchronous Machine” (Embedded Coder). In this example the controller model is configured for the TI F28335 target hardware. An archived library allowing incremental build will be automatically generated for the controller model.

Schedule Controller using ADC End-of-Conversion Interrupt


Field-Oriented Control of a Permanent Magnet Synchronous Machine
F28335 Synchronized to ADC Interrupt
Note: This demo requires a DMC550 controller and a PMS motor
C28x
IRQ5
Interrupt
Hardware Interrupt
QEP_INDEX_INT
QEP_INDEX_INT()
position_valid
Latch_Encoder_Index
encoder_valid
Controller_And_Peripherals
Copyright 2006-2023 The MathWorks, Inc.

In this section, the c28335_pmsmfoc_adcinterrupt model is used to call the controller model using a Model Block configured to c28335_pmsmfoc. Within the `Controller_And_Peripherals` subsystem, the `ePWM1` block is configured to trigger the ADC conversion so that sampling does not occur during a PWM edge transition (thus minimizing noise on the sampled signals). The controller is scheduled using the ADC end-of-conversion interrupt. This provides the shortest and most deterministic delay between the ADC conversion and the new values of PWM duty cycles.

By default, a model set for a processor such as TI F28335 will use a hardware timer to schedule all synchronous rates present in the model. If we keep this default behavior, the controller algorithm will not be synchronized with the ADC which could introduce latencies or drifts between the controller algorithm and the ADC conversions. In the c28335_pmsmfoc_adcinterrupt Model Configuration Parameters, under Hardware Implementation > Scheduler options, `ADCINT1` is selected for the Scheduler interrupt source. This option replaces the default scheduler interrupt source set to CPU Timer 0 interrupt with the ADC Interrupt coming from ADC module 1 (ADCINT1). On the ADC block, an end-of-conversion interrupt is enabled for ADC module 1. Since the ADC conversions are launched from the PWM module running at 25kHz, and since the base rate (fundamental sample time) of the model is set to 40us, it is safe to replace the scheduler interrupt source with the ADC interrupt triggering periodically at 40us. While using the default CPU Timer 0 as a scheduler interrupt source, the tool will automatically set the CPU Timer 0 to honor the base rate of the model. While replacing the scheduler interrupt source with ADCINT1, like in this example, it is the responsibility of the user to ensure that ADCINT1 will trigger periodically at the base rate used in the model.

This model showed an advanced technique for scheduling a multi-rate controller algorithm using an ADC end-of-conversion interrupt. This technique is valid for this example because we have configured the model to generate the ADC end-of-conversion interrupt at the same periodic rate as the sample time specified in the controller algorithm. If you do not ensure this sampling consistency, the behavior of generated code will be different than that of simulation.

Conclusion

This example showed how to create a real-time executable for a Texas Instruments F28335 embedded target using an advanced scheduling technique triggering the controller algorithm based on an ADC end-of-conversion interrupt. In this technique, the hardware is configured to generate a periodic interrupt with the same rate as the fastest rate specified in the controller.

More About

C28x eQEP

Photovoltaic Inverter with MPPT Using Solar Explorer Kit

This example shows how to implement a photovoltaic (PV) inverter system using the C2000™ Microcontroller Blockset. The example uses the Texas Instruments Solar Explorer Kit along with the Texas Instruments F28035 controlCARD.

Using this example, you can:

- Simulate a plant model for a PV inverter system
- Test the performance and tune the control algorithm
- Generate code for the controller and load it on the controlCARD
- Monitor signals and tune parameters using the host computer

Required Hardware

- Texas Instruments Solar Explorer Kit (TMDSSOLAR(P/C)EXPKIT)
- F28035 controlCARD

Available Models

- Solar_Inverter_Sim.slx can be used to simulate the plant model and controller for the PV inverter system.
- c28035solar_inverter.slx can be used to generate code and load it on the F28035 controlCARD.
- c2000_host_solar.slx can be run on the host computer to log signals and tune parameters.

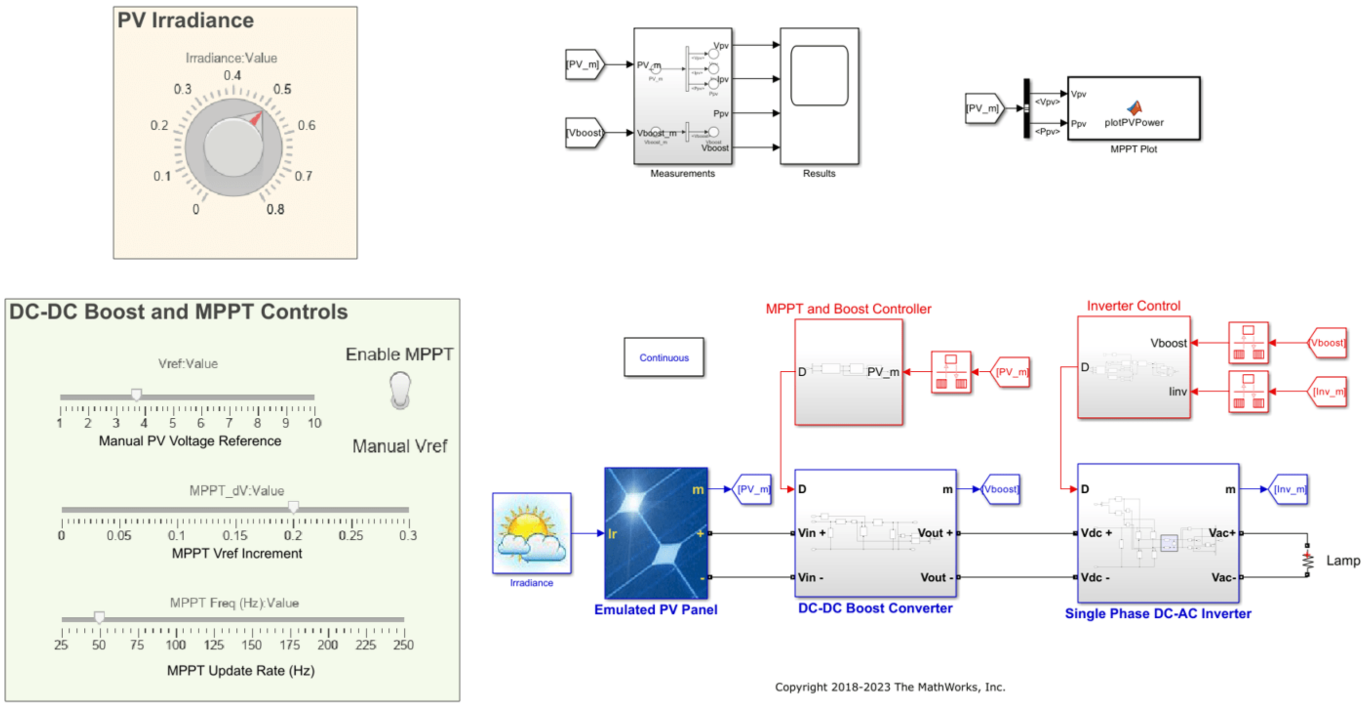
Simulate the Photovoltaic Inverter with MPPT

The simulation model consists of the plant model and the controllers. The plant model consists of three major components:

- Emulated PV Panel: This module takes the irradiance value as input (in kW/m2) and simulates the PV emulator implemented on the Texas Instruments Solar Explorer Kit.
- DC-DC Boost Converter: This module boosts up the input voltage based on the duty cycle of the PWM pulses.
- Single Phase DC-AC Inverter: This module generates a single-phase AC voltage waveform using the H-bridge topology. An AC load can be connected to the output.

The controllers in the simulation model are:

- Maximum power point tracking (MPPT)
- DC-DC boost controller
- DC-AC inverter controller



To make a solar panel energy efficient, the panel must be operated at its maximum power point. However, the maximum power point is not fixed because of the nonlinear nature of the PV cell and changes in temperature and light intensity. The perturb & observe (P&O) algorithm is implemented using a Stateflow® chart for calculating the reference voltage required for maximum power point operation. The reference voltage is achieved with the help of the DC-DC boost controller. The DC-DC boost controller implements a PI controller to track the reference voltage set by the MPPT algorithm. For tracking the reference voltage, the PV panel voltage (Vpv) is measured.

The DC-DC boost converter is a traditional single-phase converter with a single switching MOSFET, Q1. The duty cycle of the PWM output that drives Q1 determines the amount of boost imparted to the controlled parameter.

The DC-DC boost controller is realized using a PI controller. An increase in the duty cycle of the boost converter loads the panel and results in a panel output voltage drop. Accordingly, an increase in the output of the controller (duty cycle of the boost converter) causes an increase in the error input to the controller. To achieve the reference voltage value, the feedback and the voltage reference inputs of the PI controller are reversed. The controller operates at a rate of 50 kHz.

The DC-DC boost converter output voltage is not controlled using the DC-DC boost controller. However, the boost converter output voltage is regulated by the DC-AC inverter controller, which modulates the current drawn by the DC-AC inverter to keep this voltage regulated. The DC-AC inverter controller uses nested control loops — an outer voltage loop and an inner current loop.

The voltage loop generates the reference for the current loop. An increase in the current loads the DC bus and therefore causes a drop in the DC bus voltage. To achieve the reference voltage value, the feedback and the outer voltage compensator reference are reversed. The current reference is then multiplied by the sine reference to get the instantaneous current reference. The instantaneous current reference is then used by the current compensator along with the feedback current to provide duty cycle for the DC-AC inverter. The duty cycle is calculated using the unipolar sine PWM technique.

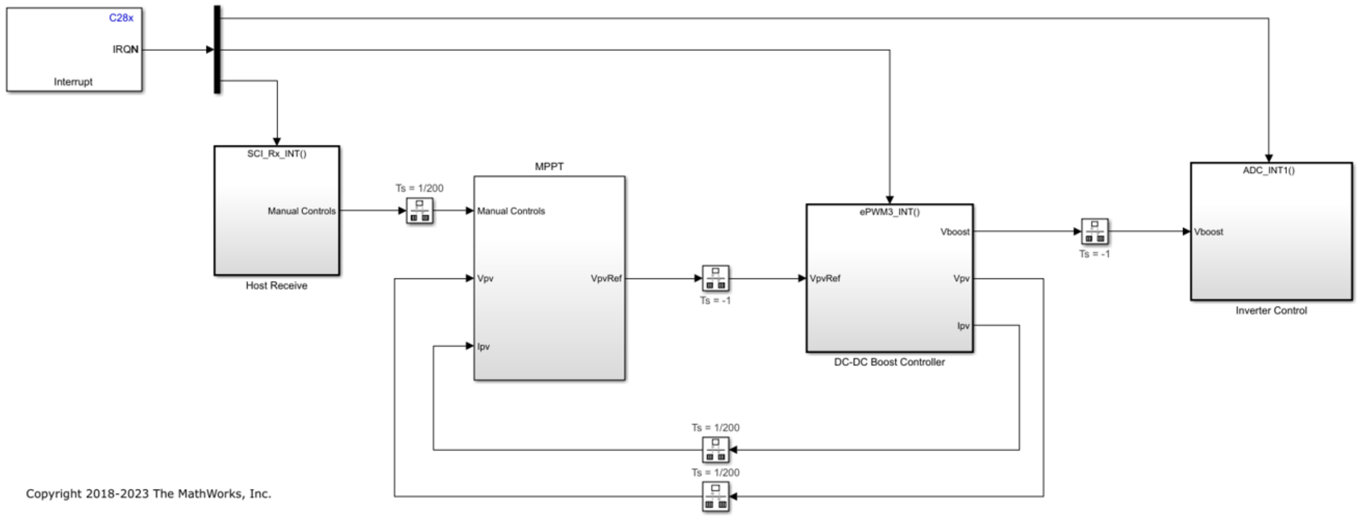
The input to the PV emulator can be changed to run the simulation for different values of irradiance. The controller parameters can be tuned to get a refined performance.

Generate Code for the Controller and Load It on the ControlCARD

The deployment model consists of three real-time interrupt service routines (ISR) used for:

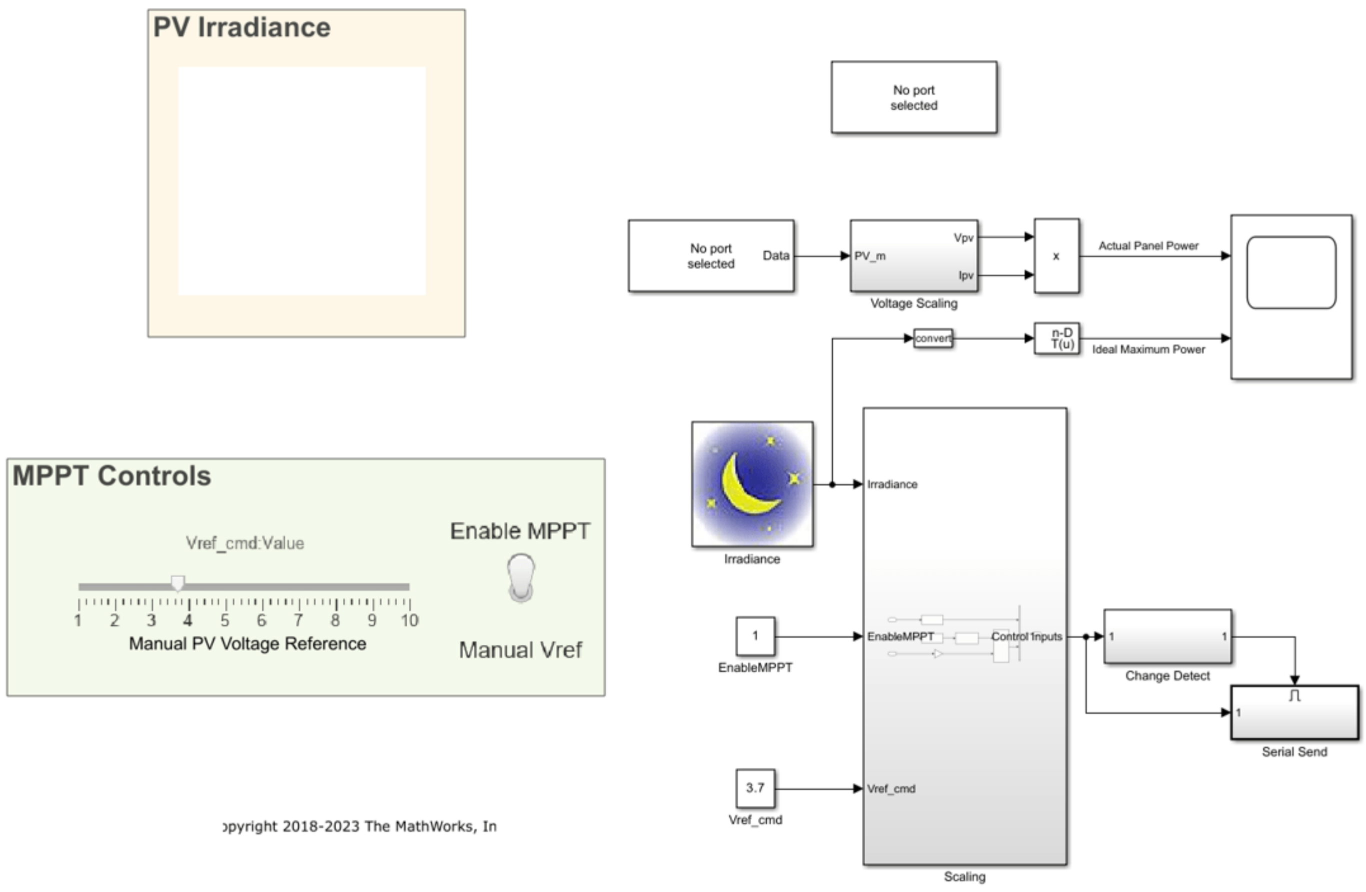
- Closed loop control of the DC-DC boost converter (50 kHz)
- Closed loop control of the DC-AC inverter (20 kHz)
- Setting the irradiance value sent by the user from a host model

The F28035 processor receives the irradiance value through serial communications interface (SCI) and sends it to the F28027 processor (PV emulator on the Solar Explorer Kit) using serial peripheral interface (SPI) communication. The F28027 processor is preconfigured to receive the updated irradiance value from the F28035 processor.



Monitor Signals and Tune Parameters Using the Host Computer

The host model receives the data from the kit and plots it to verify the performance of the MPPT and the control algorithms.



Monitor the Signals

While the model runs, you can monitor the Actual Panel Power signal on the scope. Actual Panel Power is the real-time power supplied by the PV emulator.

Tune the Parameters

While the model runs, you can tune parameters using the dashboard blocks:

- Irradiance — The irradiance value (in kW/m2) supplied to the PV emulator.
- Manual PV Voltage Reference — The value used to manually set the operating point of the PV emulator. This value is used when the toggle switch is set to the **Manual Vref** option. Toggling the switch turns off the MPPT algorithm and lets you choose the operating voltage of the PV emulator. The value must be less than or equal to the open-circuit voltage of the panel at the set irradiance, that is, `Vref_cmd <= Irradiance (in kW/m2) * 28`.

More About

- “MPPT Using Flyback Converter in TI Solar Micro Inverter Development Kit” on page 4-250
- “ADC-PWM Synchronization Using ADC Interrupt” on page 4-33
- “Closed Loop Control of a DC-DC Buck Converter” on page 4-35

Power Factor Correction Using Boost Converter

This example shows how to implement power factor correction (PFC) using a boost converter with the C2000™ Microcontroller Blockset. The example uses the Texas Instruments F28069M controlCARD with high voltage motor control kit (TMDSHVMTRINSPIN).

Using this example, you can:

- Simulate PFC using a boost converter.
- Generate code for the controller and load it on the controlCARD.
- Monitor signals using the host computer.

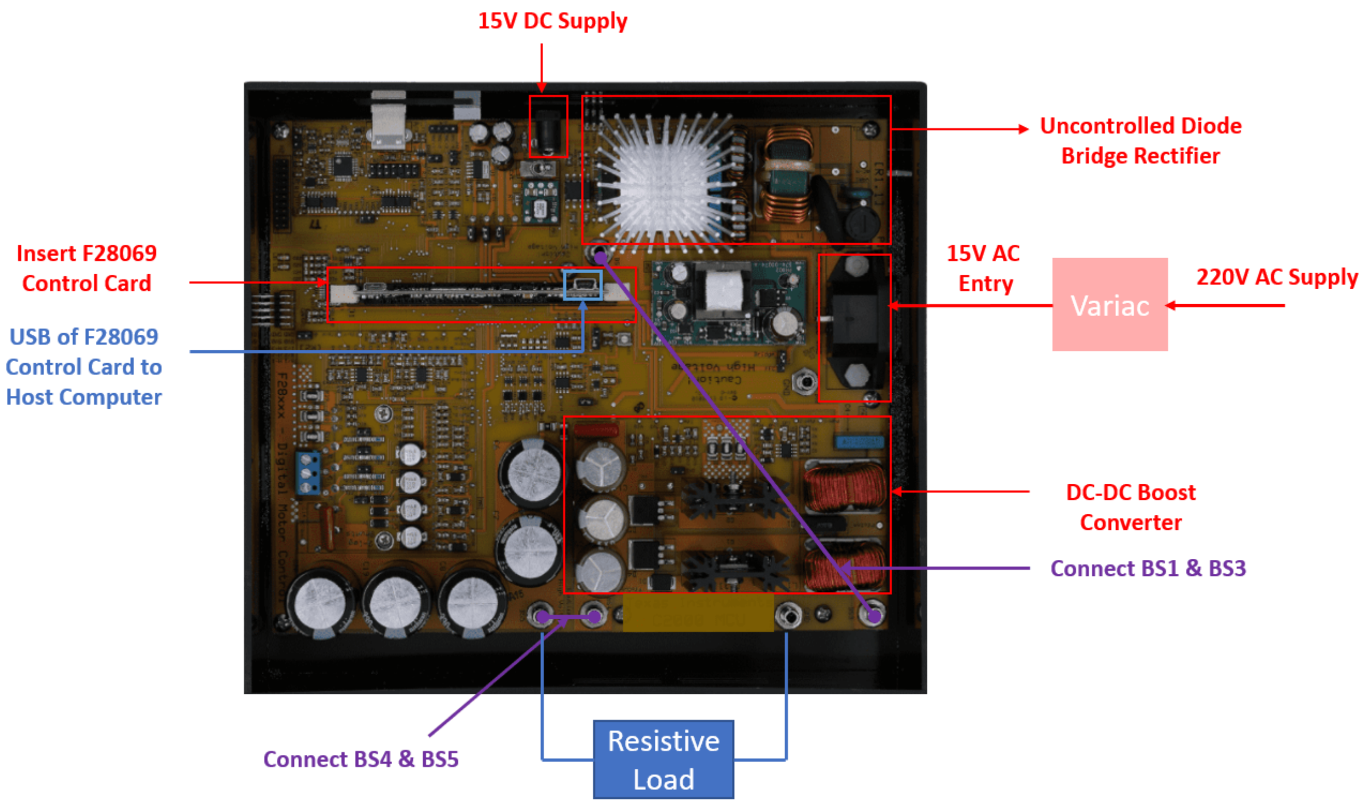
Required Hardware

- F28069M controlCARD
- High voltage motor control kit (TMDSHVMTRINSPIN)
- Resistive load of 30 ohms

Hardware Connections

The High voltage motor control kit (TMDSHVMTRINSPIN) configuration includes the following hardware components:

- F28069M controlCARD
- DC power supply
- Resistive Load
- Variac



The following steps describe the jumper connections, switch settings and power supply connections for the High voltage motor control kit (TMDSHVMTRINSPIN):

1. Install the Jumpers [Main]-J3, J4 and J5, J9 for 3.3V, 5V and 15V power rails.

2. Unpack the DIMM style controlCARD and place it in the connector slot of [Main]-J1.

3. For controlCARDs with on-card isolated emulation, ex: TMDSCNCD28069MISO:

- Populate M3-J5 on HV kit (disables power to HVKIT emulator)
- Do NOT populate J9 on HV kit (disables JTAG connection)

4. On the F28069 ISO control card, ensure the following switch settings,

- **SW3**: position 1 **ON** and position 2 **ON**.
- **SW2**: position 1 **OFF** and position 2 **OFF**.
- **SW1**: position 1 **ON** and position 2 **ON**.

5. Connect 15V DC power supply to [M6]-JP1 and ensure that [M6]-SW1 is in the "Off" position.

6. Connect banana cable from [Main]-BS1 to [Main]-BS3 (This will connect the rectifier output to the boost converter input).

7. Connect banana cable from [Main]-BS4 and [Main]-BS5 (This will include the inverter DC bus capacitors in parallel to the pfc output stage capacitors).

8. Connect AC power to the [Main]-P1 through a variac. Ensure AC input to the kit is in the range 15 to 18 V.

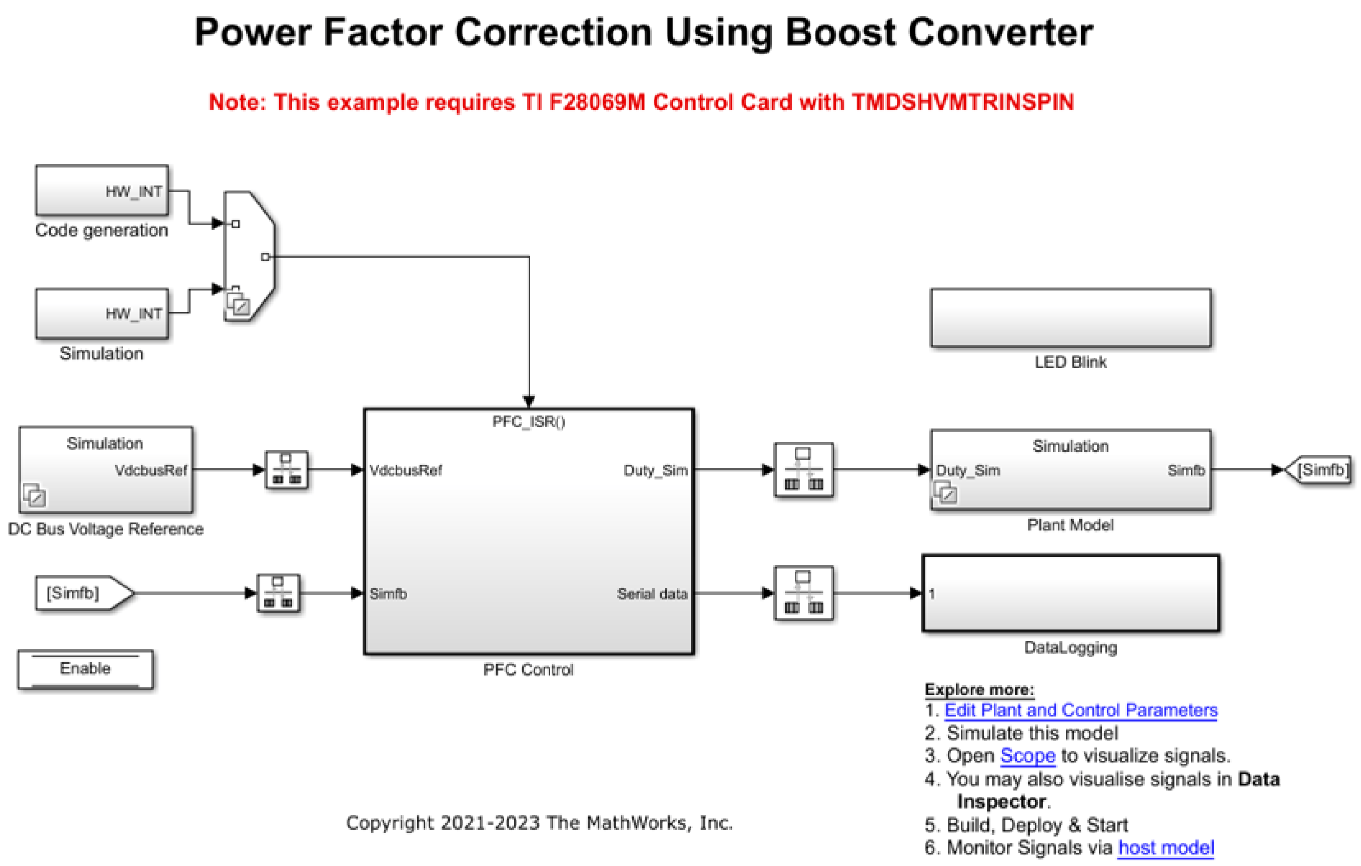
Available Models

- c28069pfchvkit.slx simulate, generate code, and load it on the F28069M controlCARD.
- c2000_host_pfc.slx runs on the host computer to log signals.

The model consists of the plant model and the controller. The plant model consists of two major modules:

- **AC-DC Rectifier**: This module takes 15 to 18 V AC input and outputs rectified DC voltage.
- **DC-DC Boost Converter**: This module boosts up the input voltage based on the duty cycle of the pulse width modulation (PWM) output.

```
open_system('c28069pfchvkit.slx');
```



Explore more:
1. Edit Plant and Control Parameters
2. Simulate this model
3. Open Scope to visualize signals.
4. You may also visualise signals in **Data Inspector**.
5. Build, Deploy & Start
6. Monitor Signals via host model

The AC-DC rectification stage uses a traditional uncontrolled H-bridge rectifier. The DC-DC boost converter on the kit has a two-phase interleaved topology. For simplicity, only one phase has been used for the boost operation. The duty cycle of the PWM output determines the amount of boost imparted to the input voltage.

The PFC controller provides current shaping of the AC input and regulates the DC bus. The outer voltage loop ensures that the output DC voltage is maintained at the set reference(24 V) by using a discrete proportional integral (PI) controller.

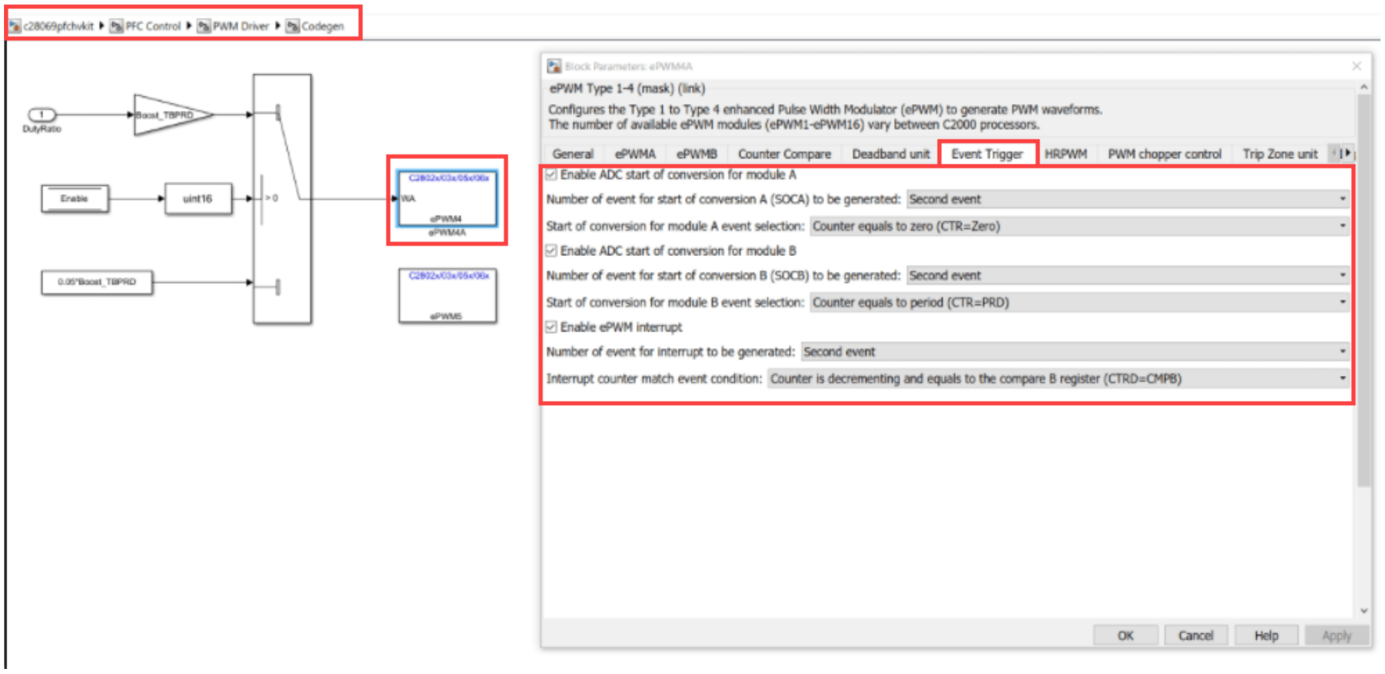
The inner current loop performs the wave shaping of the input AC current to maintain a high power factor. The reference for the current loop is generated by feed-forward of the rectified DC voltage as well as the output of the voltage control loop.

The output of the PFC controller is the PWM duty cycle of the DC-DC boost converter which is switched at 200 kHz. The controller operates at a rate of 100 kHz i.e. half of the PWM switching frequency.

Peripheral Block Configuration

ePWM block

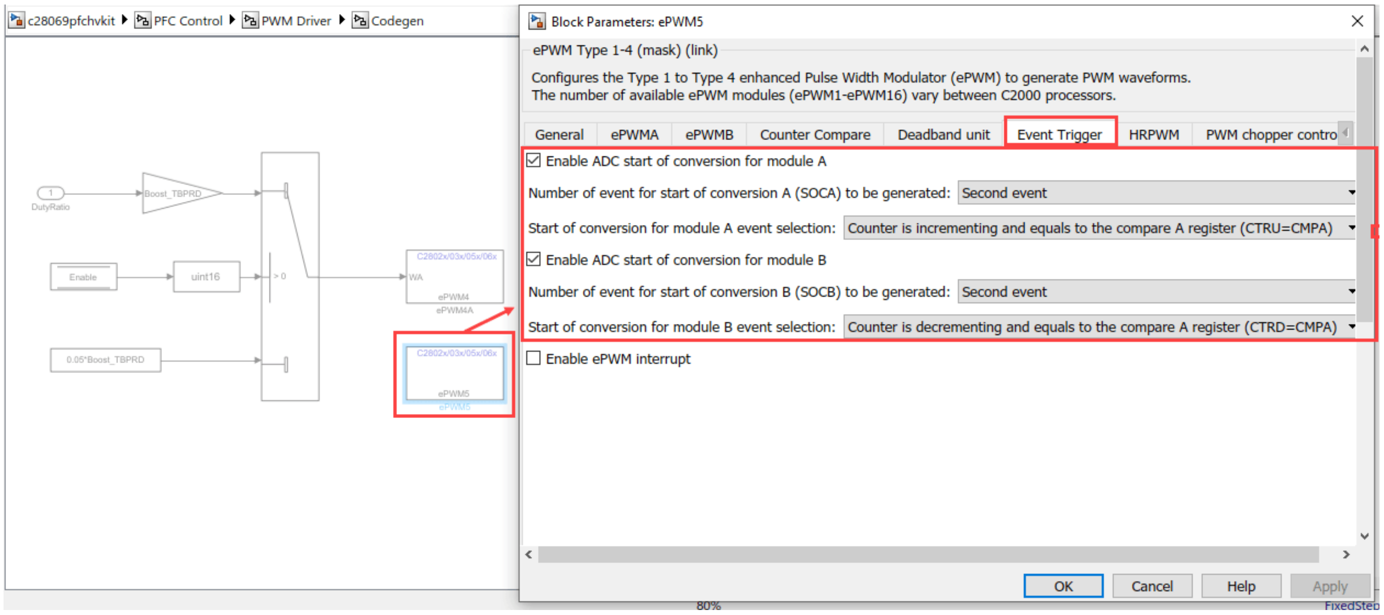
The PWM signals are generated at a frequency of 200 kHz. The ePWM4 module is configured to operate in up-down count mode. All ADC results are read in (100kHz) ISR labelled PFC_ISR. This ISR is triggered by ePWM4 on every second of CMPB (with a preset value) match event during down count, so that the ISR is triggered only after the ADC conversions are complete.



ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

- [✓] Enable ADC start of conversion for module A
- Number of event for start of conversion A (SOCA) to be generated: Second event
- Start of conversion for module A event selection: Counter equals to zero (CTR=Zero)
- [✓] Enable ADC start of conversion for module B
- Number of event for start of conversion B (SOCB) to be generated: Second event
- Start of conversion for module B event selection: Counter equals to period (CTR=PRD)
- [✓] Enable ePWM interrupt
- Number of event for interrupt to be generated: Second event
- Interrupt counter match event condition: Counter is decrementing and equals to the compare B register (CTRD=CMPB)



Block Parameters: ePWM5

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

General | ePWMA | ePWMB | Counter Compare | Deadband unit | Event Trigger | HRPWM | PWM chopper contro

☑ Enable ADC start of conversion for module A

Number of event for start of conversion A (SOCA) to be generated: Second event

Start of conversion for module A event selection: Counter is incrementing and equals to the compare A register (CTRU=CMPA)

☑ Enable ADC start of conversion for module B

Number of event for start of conversion B (SOCB) to be generated: Second event

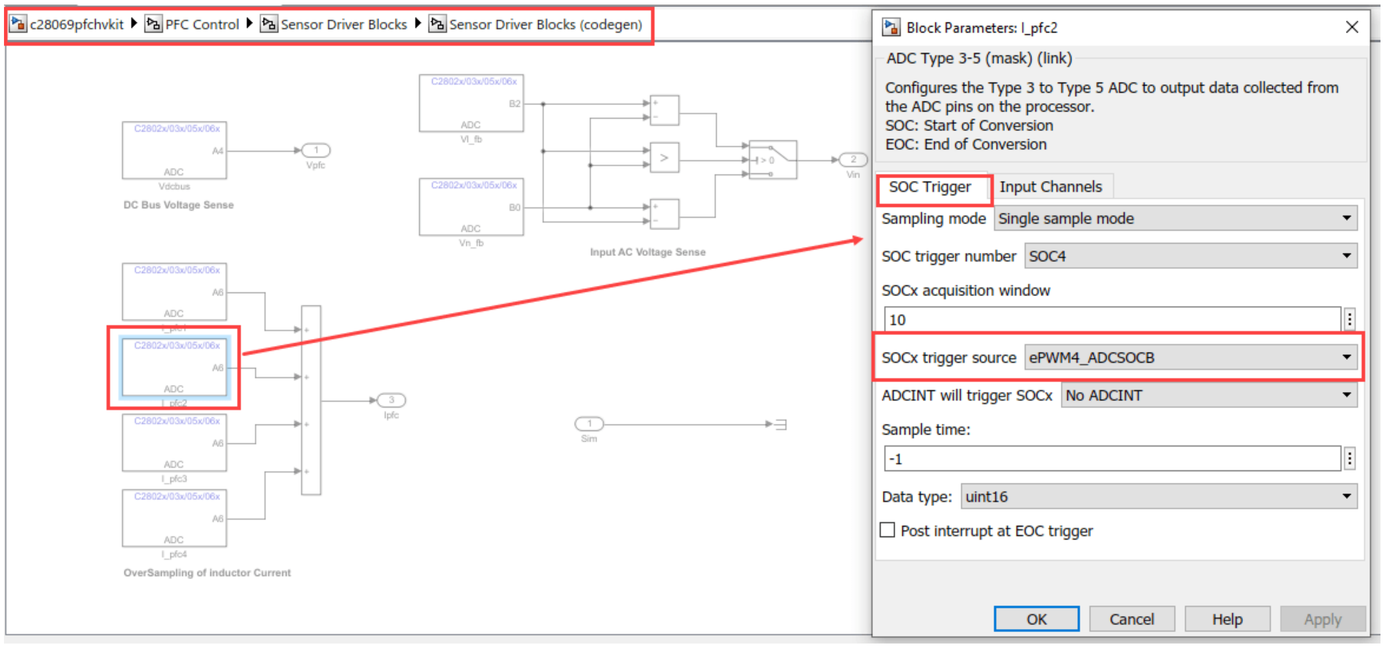
Start of conversion for module B event selection: Counter is decrementing and equals to the compare A register (CTRD=CMPA)

☐ Enable ePWM interrupt

ADC block

PFC total inductor current is sampled at the midpoint of the PWM ON pulse. Since the sampled value represents the average inductor current under CCM (continuous conduction mode) condition. PFC inductor current is also oversampled 4 times during each PWM time period. These 4 sampled values are then used to calculate the average PFC inductor current. The voltage signal conversions are also initiated when the PFC switch is on. The 4 SOC triggers are generated when,

- ePWM4 counter reaches zero and period.
- ePWM5 counter reaches a preset CMPA values during up and down count.


c28069pfchvkit ▸ PFC Control ▸ Sensor Driver Blocks ▸ Sensor Driver Blocks (codegen)
ADC
Vdcbus
Vpfc
DC Bus Voltage Sense
Vl_fb
Vn_fb
Vin
Input AC Voltage Sense
I_pfc2
I_pfc3
I_pfc4
Ipfc
Sim
OverSampling of Inductor Current
OK
Cancel
Help
Apply

Block Parameters: I_pfc2

ADC Type 3-5 (mask) (link)

Configures the Type 3 to Type 5 ADC to output data collected from the ADC pins on the processor.
SOC: Start of Conversion
EOC: End of Conversion

SOC Trigger | Input Channels

Sampling mode: Single sample mode

SOC trigger number: SOC4

SOCx acquisition window: 10

SOCx trigger source: ePWM4_ADCSOCB

ADCINT will trigger SOCx: No ADCINT

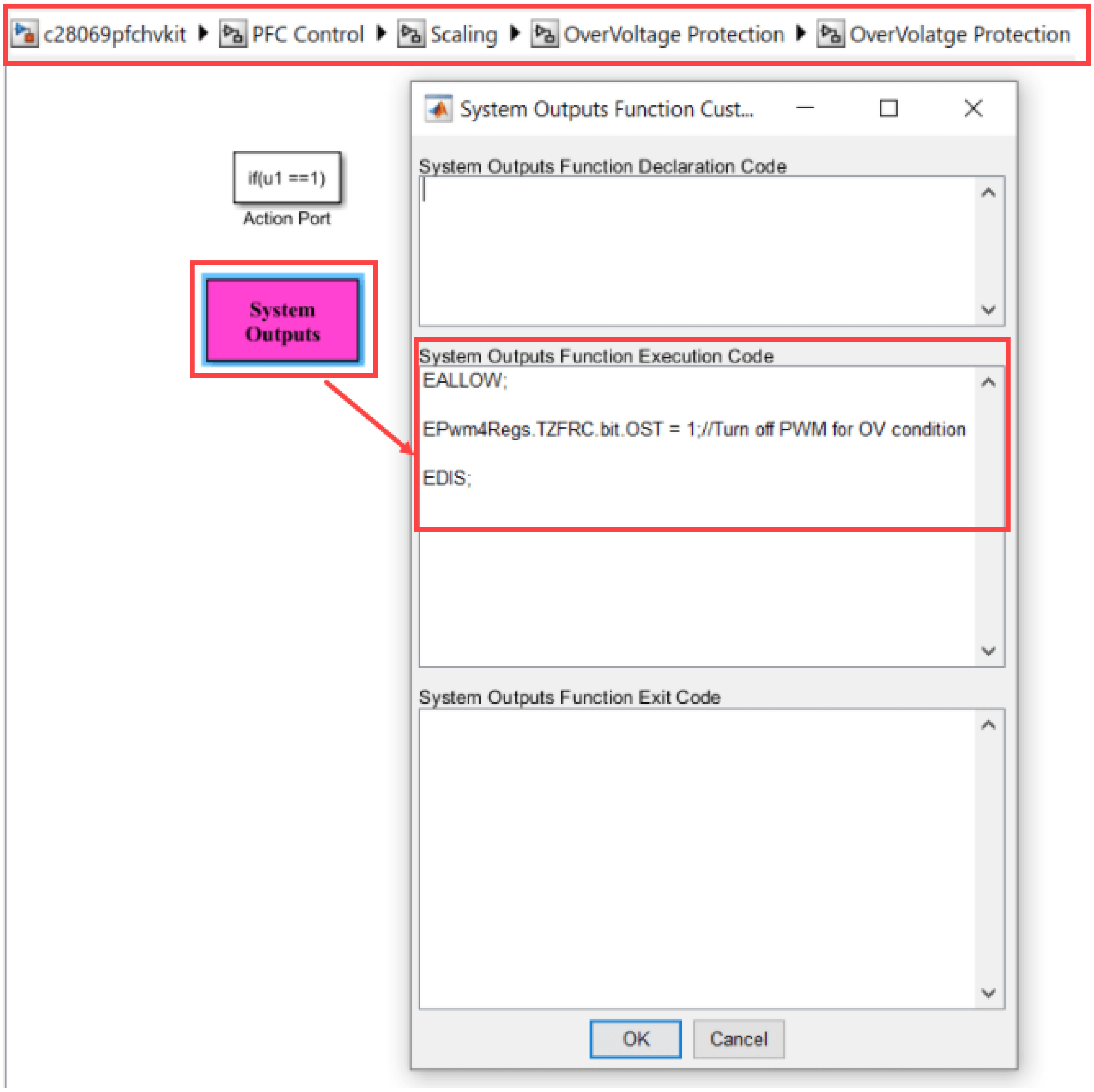
Sample time: -1

Data type: uint16

☐ Post interrupt at EOC trigger

Overvoltage Protection

An overvoltage protection mechanism is implemented using custom code in System Outputs block. The sensed DC bus output voltage from the ADC input is compared against the overvoltage protection threshold set in the initialization script file. `Vpfc_trip_level` is the overvoltage threshold point which is set to 35 V for the model. When overvoltage condition is met, the one-shot trip in ePWM is enabled.


c28069pfchvkit ▶ PFC Control ▶ Scaling ▶ OverVoltage Protection ▶ OverVolatge Protection
if(u1 ==1)
Action Port
System Outputs
System Outputs Function Cust...
System Outputs Function Declaration Code
System Outputs Function Execution Code
System Outputs Function Exit Code
OK
Cancel

```
EALLOW;

EPwm4Regs.TZFRC.bit.OST = 1;//Turn off PWM for OV condition

EDIS;
```

Simulate PFC Using Boost Converter

Run the Model

1. Open the c28069pfchvkit.slx model, and click the **Run** button to simulate the model.

2. Observe the output waveforms on the `Results` block from the following path **c28069pfchvkit** > **Plant Model** > **Simulation**.

Generate Code for Controller and Load It on controlCARD

The deployment model has a real-time interrupt service routines (ISR) configured to trigger PFC control at the rate of 100 kHz.

Load Model on controlCARD

1. Ensure the hardware connections are made as mentioned in the **Hardware Connections** section.

2. Turn on [M6]SW1 on the kit to power on the controlCARD.

3. Open the c28069pfchvkit.slx model and generate code by pressing **Ctrl+B**.

4. Follow the build process by opening the diagnostic viewer using the link at the bottom of the model canvas.

5. Connect an appropriate resistive load to the PFC DC output terminals (BS5 & BS6). Ensure the load resistance is in the range of 20 to 50 ohms.

6. Turn on the AC power supply and gradually increase the AC input ([Main]-P1) to the value between 15 V to 18 V.

- The PFC algorithm kicks in when the AC input is more than 14 V.

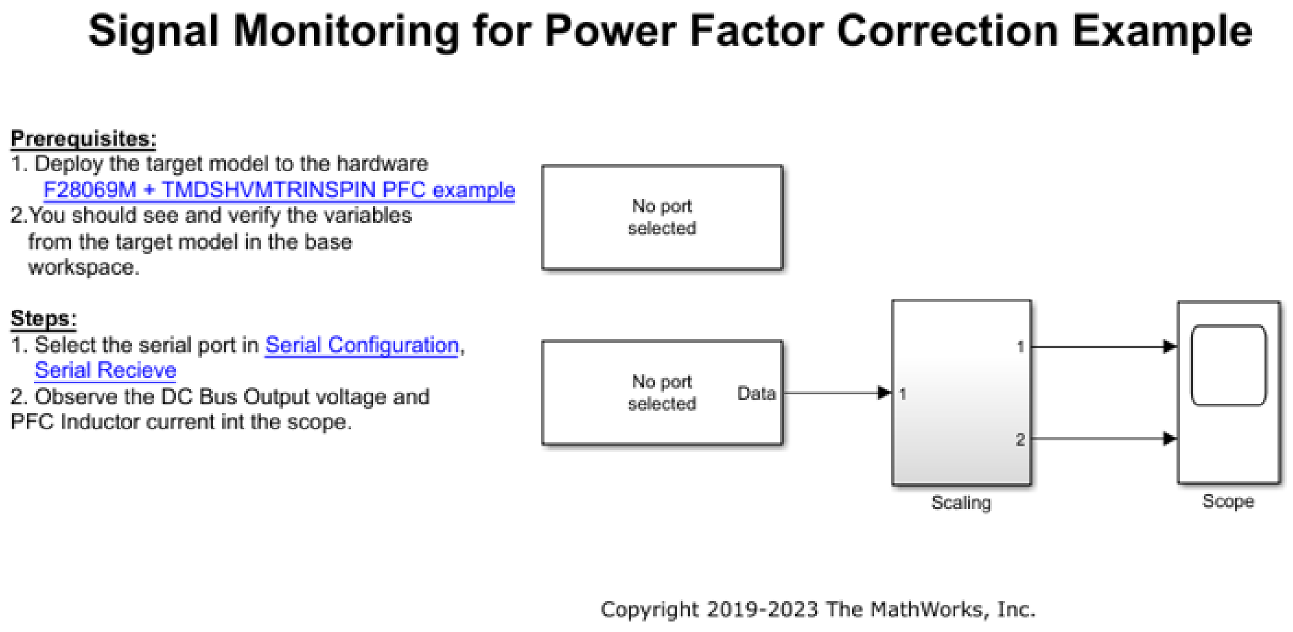
7. Run the host model and observe the output capacitor voltage & the PFC inductor current.

8. Turn off the AC power first and wait for a few minutes for the PFC DC bus capacitor to discharge completely.

Monitor Signals on Host Computer

The host model receives the data from the hardware kit and plots it to verify the performance of the PFC controller.

```
open_system('c2000_host_pfc.slx');
```


Signal Monitoring for Power Factor Correction Example

Prerequisites:
1. Deploy the target model to the hardware
   F28069M + TMDSHVMTRINSPIN PFC example
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select the serial port in Serial Configuration, Serial Recieve
2. Observe the DC Bus Output voltage and PFC Inductor current int the scope.

Copyright 2019-2023 The MathWorks, Inc.

Configure and Run Model on Host Computer

1. On the host computer, browse to **Device Manager** > **Ports (COM & LPT)** to find the COM port.

2. Set the parameter Port of the following blocks in the c2000_host_pfc model to match the COM port of the host computer:

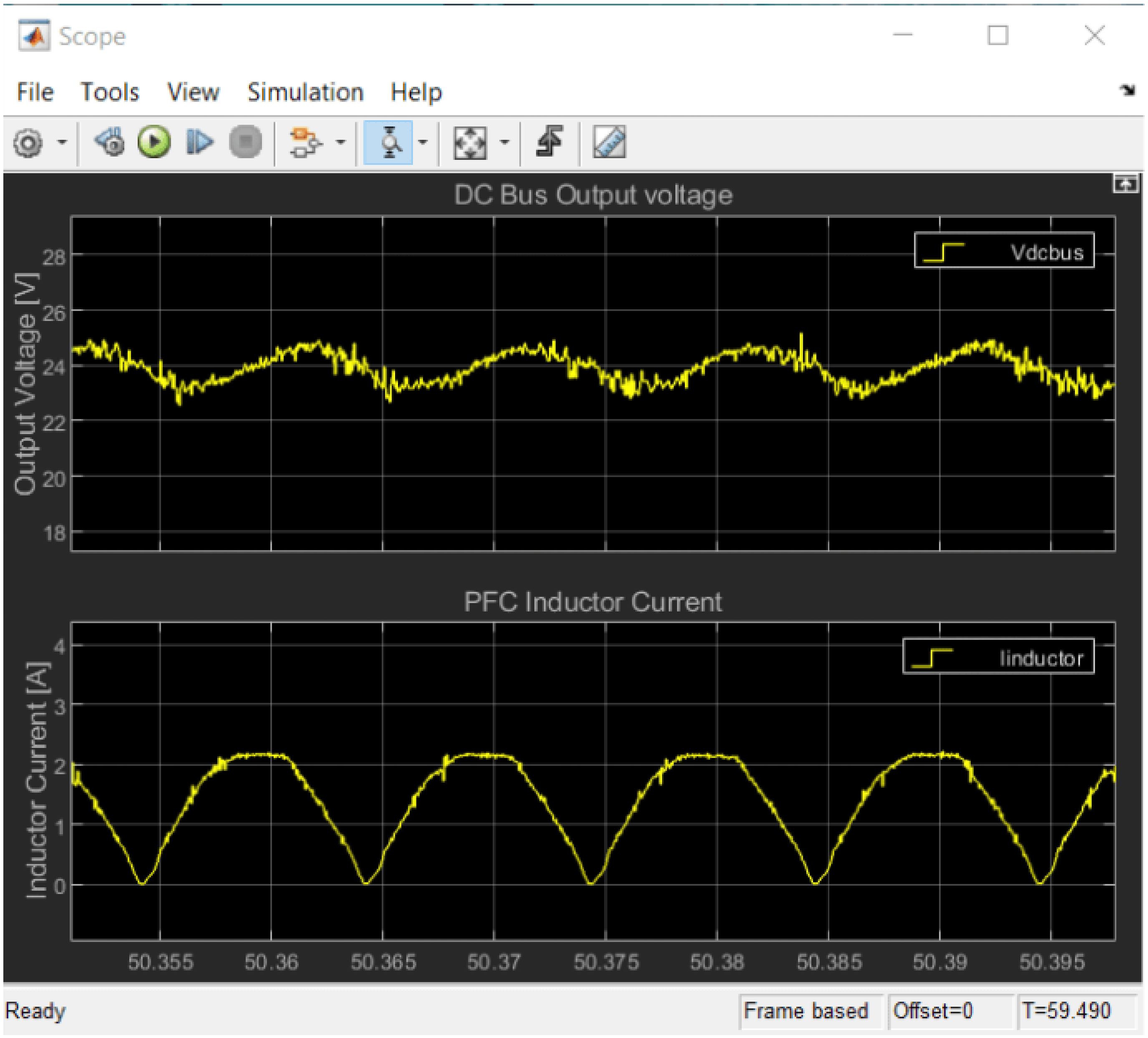
- c2000_host_pfc > Serial Configuration
- c2000_host_pfc > Serial Receive

3. Click the **Run** button on the Simulation tab to run the host model.

4. While the model runs, you can monitor the boost converter output capacitor voltage and input inductor current to analyze the performance of PFC control.

Hardware Results

The following figure shows the hardware results for the model matlab:c28069pfchvkit.slx. Observe the boost converter output capacitor voltage and input inductor current. The following result is taken at a resistive load of 30 ohm.



More About

- “ADC-PWM Synchronization Using ADC Interrupt” on page 4-33
- “Closed Loop Control of a DC-DC Buck Converter” on page 4-35
- “Photovoltaic Inverter with MPPT Using Solar Explorer Kit” on page 4-216

Interface LCD Booster Pack with Texas Instruments C2000 Processors

This example shows how to configure and use the Kentec QVGA Display Booster Pack to display an image using C28x™ peripherals for Texas Instruments™ C2000™ processors.

Using this example, you can:

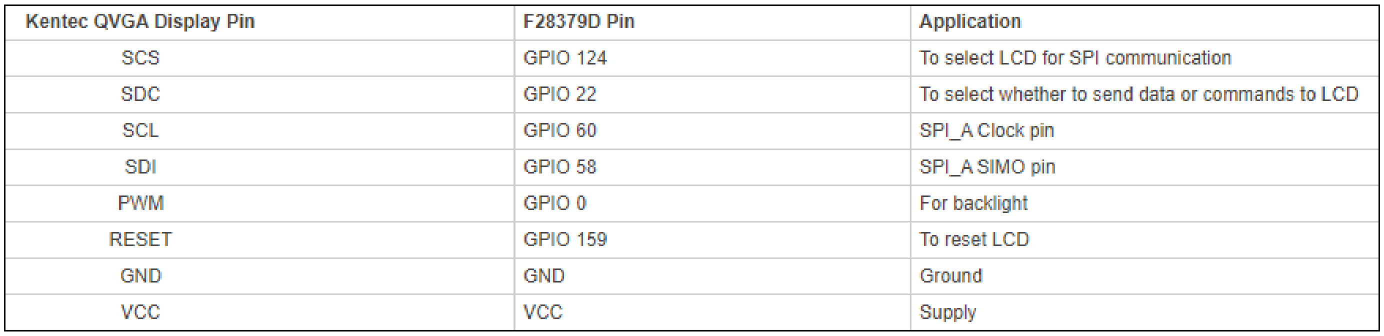
- Configure the Kentec QVGA display through the serial peripheral interface (SPI)
- Display the image through the SPI using direct memory access (DMA)

Required Hardware

- F28379D Launchpad
- BOOSTXL-K350QVG-S1 Kentec QVGA Display Booster Pack

Hardware Connections

Mount the Kentec QVGA Display Booster Pack on the F28379D Launchpad. The display uses the following pin mapping with the F28379D Launchpad.



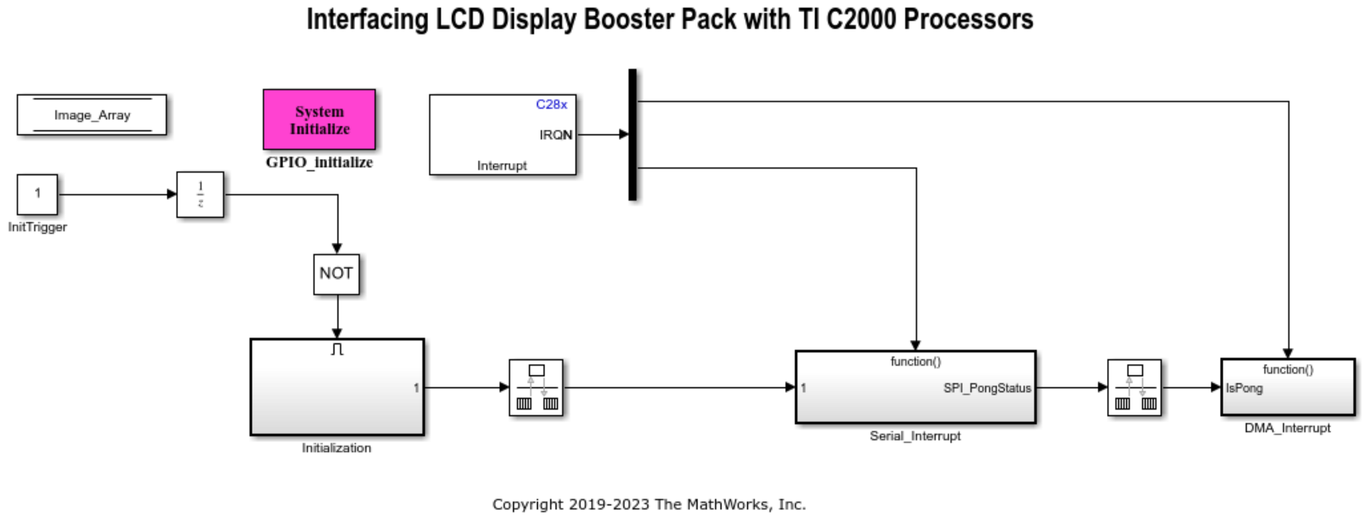
Kentec QVGA Display Pin	F28379D Pin	Application
SCS	GPIO 124	To select LCD for SPI communication
SDC	GPIO 22	To select whether to send data or commands to LCD
SCL	GPIO 60	SPI_A Clock pin
SDI	GPIO 58	SPI_A SIMO pin
PWM	GPIO 0	For backlight
RESET	GPIO 159	To reset LCD
GND	GND	Ground
VCC	VCC	Supply

Configure Kentec QVGA Display Through SPI

Before the image is transferred from host computer to F28379D Launchpad:

- Disable serial communication interface (SCI) interrupt ensure DMA channel is in halt state.
- Initialize the display.

```
open_system('c2837xD_lcd_display.slx');
```



Initialize the Kentec QVGA Display

1. Reset the LCD, turn ON the backlight, select the SCS pin and wait for 1 ms.

2. Configure the following registers: Power parameter, Pixel format, Display orientation, MCU interface parameter, Display control, Gamma correction and Power control. To configure Pixel format - Exit sleep mode, wait for 30 ms and then configure the register.

3. Display a **Green** color image on the LCD to indicate end of initialization.

4. Enable the SCI interrupt and the DMA channel.

Display the Image Through SPI Using DMA

For the image to display on the LCD, the image data must be transferred from the host computer to the target F28379D Launchpad through the SCI.

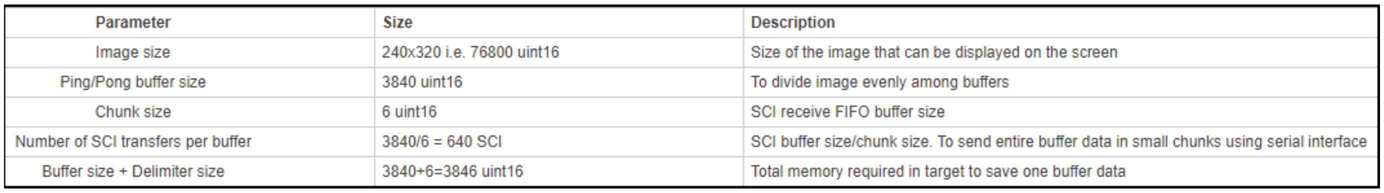
Due to low memory constraints on the target F28379D Launchpad, it is not possible to send the entire image data from the host computer to the target F28379D. Instead, the image data is transferred in chunks from the host computer to the target F28379D through the SCI. These chunks are stored in a small memory section `Image_Array` in the target. Double buffer is used in `Image_Array` to ensure data integrity between data read from the host using SCI and data transferred to the LCD display through the SPI interface. The buffers are referred as ping buffer and pong buffer.

Run the Function on the Host

The function **c2837xD_serial_send.m** performs the following operation:

- Runs on the host computer, reads the image, and sets the SCI parameters.
- Sends 16-bit image data for the entire buffer in small chunks to the target F28379D through the SCI.
- Adds a constant delimiter, of the same size as a chunk, for each buffer to indicate the beginning of new a buffer.

The following table explains the parameters used with their size.



Parameter	Size	Description
Image size	240x320 i.e. 76800 uint16	Size of the image that can be displayed on the screen
Ping/Pong buffer size	3840 uint16	To divide image evenly among buffers
Chunk size	6 uint16	SCI receive FIFO buffer size
Number of SCI transfers per buffer	3840/6 = 640 SCI	SCI buffer size/chunk size. To send entire buffer data in small chunks using serial interface
Buffer size + Delimiter size	3840+6=3846 uint16	Total memory required in target to save one buffer data

To send the entire image, buffer transmission must occur 20 times, i.e., Image size/buffer size. `Image Size` = 20 x buffer size = 20 x (chunk size x number of SCI transfers)

Run the Model on the Target

On the model side of the target, the following sequence of events occurs:

1. The target waits for the SCI interrupt. The SCI interrupt is triggered when the serial receive buffer receives the chunk size data.

2. In the serial interrupt service routines (ISR), the data is copied into the ping/pong buffer of `Image_Array`. The model uses the serial delimiter to identify the beginning of the data in the target. This process continues until the buffer transfer is complete.

3. The serial is now switched to the alternate buffer among the ping/pong to fill the data and enable the SPI event trigger for DMA transfer.

4. The DMA starts pushing the data from the first filled buffer to the SPI Transmit FIFO buffer. The data is transferred to the display through the SPI, and SPI Transmit triggers the DMA event when the buffer is empty. This process continues till the first filled buffer data is fully transferred to the display.

5. Once the transfer is complete:

- The DMA is halted, and the start address of DMA is changed to the alternate buffer
- An acknowledgement is sent to the host computer showing the first buffer is empty to receive new serial data

The above sequence continues until the entire image data is transferred to the display.

Configure and Run the Model

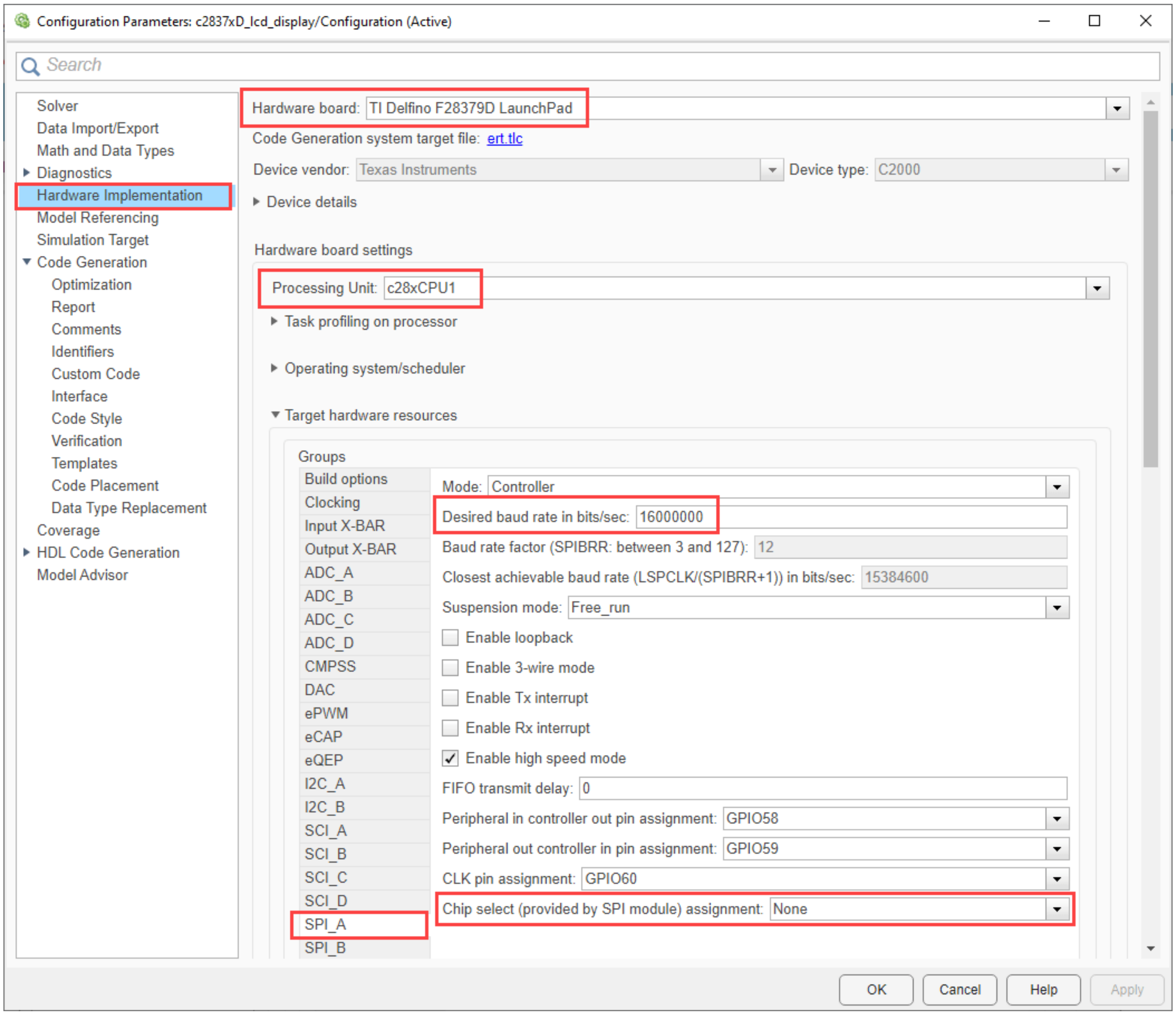
1. Open the LCD Display model.

2. Go to the **Modeling** tab and press **Ctrl+E** to open the Configuration Parameters dialog box.

3. Go to the **Hardware Implementation** and select the **Hardware board**.

4. Go to **Hardware Implementation > Target Hardware Resources > Build options** and select **Enable DMA access to peripheral frame 2 (SPI and McBSP) instead of CLA**.

5. Go to **Hardware Implementation > Target Hardware Resources > SPI_A** and set **Desired Baud rate in bits/sec** to `16000000` and **Chip select (provided by SPI module) assignment** to `None`.



Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Processing Unit: c28xCPU1

Mode: Controller

Desired baud rate in bits/sec: 16000000

Baud rate factor (SPIBRR: between 3 and 127): 12

Closest achievable baud rate (LSPCLK/(SPIBRR+1)) in bits/sec: 15384600

Suspension mode: Free_run

☐ Enable loopback

☐ Enable 3-wire mode

☐ Enable Tx interrupt

☐ Enable Rx interrupt

☑ Enable high speed mode

FIFO transmit delay: 0

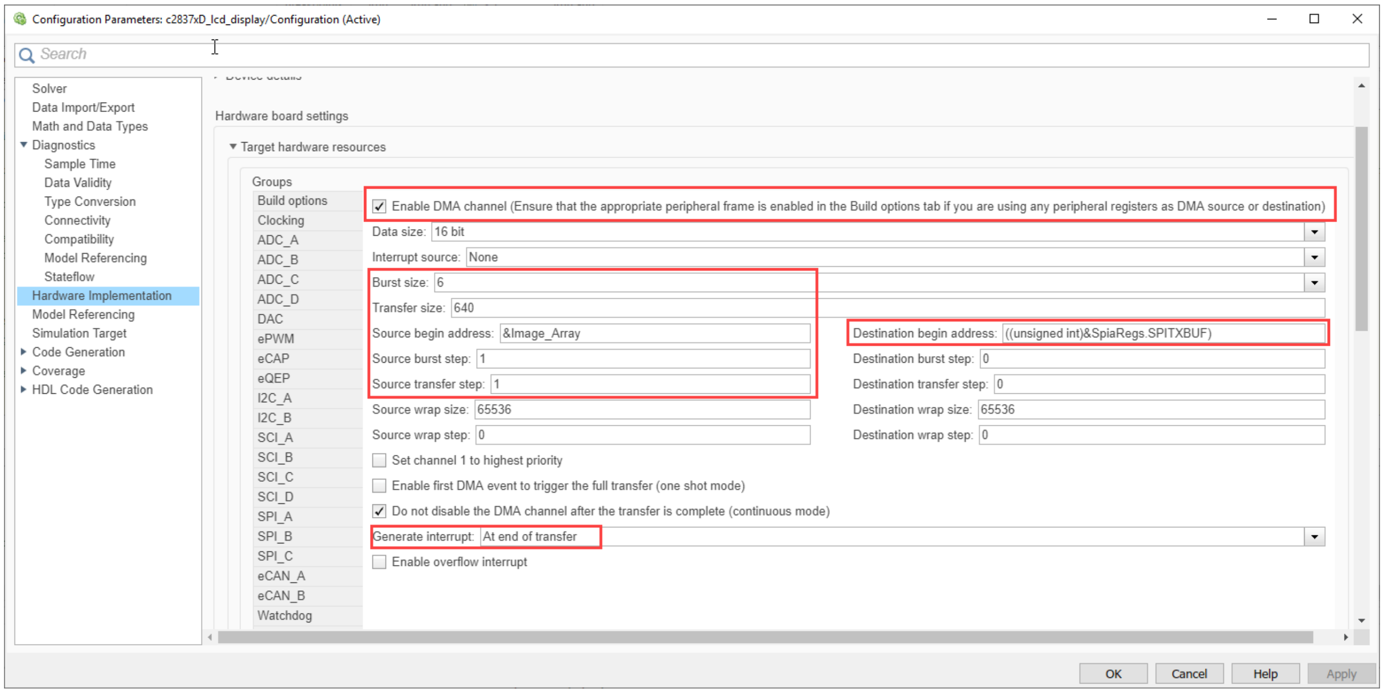
Peripheral in controller out pin assignment: GPIO58

Peripheral out controller in pin assignment: GPIO59

CLK pin assignment: GPIO60

Chip select (provided by SPI module) assignment: None

6. Go to **Hardware Implementation** > **Target Hardware Resources** > **DMA_ch1** and update parameter values as shown.



Hardware board settings

Target hardware resources

☑ Enable DMA channel (Ensure that the appropriate peripheral frame is enabled in the Build options tab if you are using any peripheral registers as DMA source or destination)

Data size: 16 bit

Interrupt source: None

Burst size: 6

Transfer size: 640

Source begin address: &Image_Array

Source burst step: 1

Source transfer step: 1

Source wrap size: 65536

Source wrap step: 0

Destination begin address: ((unsigned int)&SpiaRegs.SPITXBUF)

Destination burst step: 0

Destination transfer step: 0

Destination wrap size: 65536

Destination wrap step: 0

☐ Set channel 1 to highest priority

☐ Enable first DMA event to trigger the full transfer (one shot mode)

☑ Do not disable the DMA channel after the transfer is complete (continuous mode)

Generate interrupt: At end of transfer

☐ Enable overflow interrupt

7. Navigate to the **Hardware** tab and press **Ctrl+B**. If the LCD turns green, it indicates that the initialization of the Kentec QVGA display was successful.

8. On the host computer, run the function `c2837xD_serial_send`. The function takes two arguments:

- **COM port** - browse to **Device Manager** > **Ports (COM & LPT)** to find the COM port.
- **Image name** - file name of the image that you want to display.

Run the following command at the MATLAB® command prompt:

```
c2837xD_serial_send('COM2', 'mathworks.jpg');
```

9. Observe the image on the LCD.

Read Position of BiSS-C Absolute Encoder

This example shows how to read a position of an absolute encoder using the bidirectional serial/synchronous-continuous (BiSS-C) open protocol in the unidirectional mode. This example implements the BiSS-C protocol using TI C2000 peripherals such as CLB, SPI, ePWM, X-BAR, and GPIO using the C2000™ Microcontroller Blockset.

Using this example, you can:

- Configure peripheral modules such as CLB, SPI, ePWM, X-BAR, and GPIO to design and implement a BiSS-C protocol.
- Generate code for the controller and load it on the hardware board.
- Perform Monitor & Tune to monitor the position signals on the host computer.

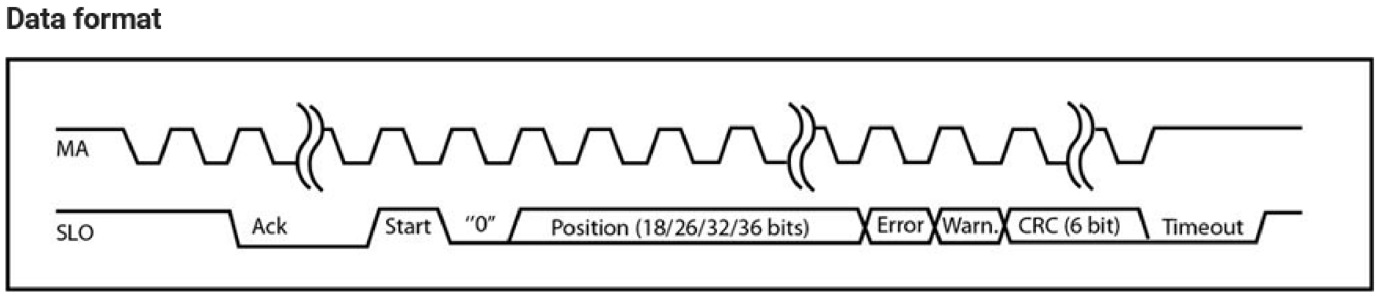
BiSS-C Interface

BiSS-C mode (unidirectional) is a fast synchronous serial interface for acquiring position data from an encoder. The controller controls the time of position acquisition and the data transmission speed, and the encoder transfers the position data to the controller.

The interface consists of two unidirectional differential pairs of lines:

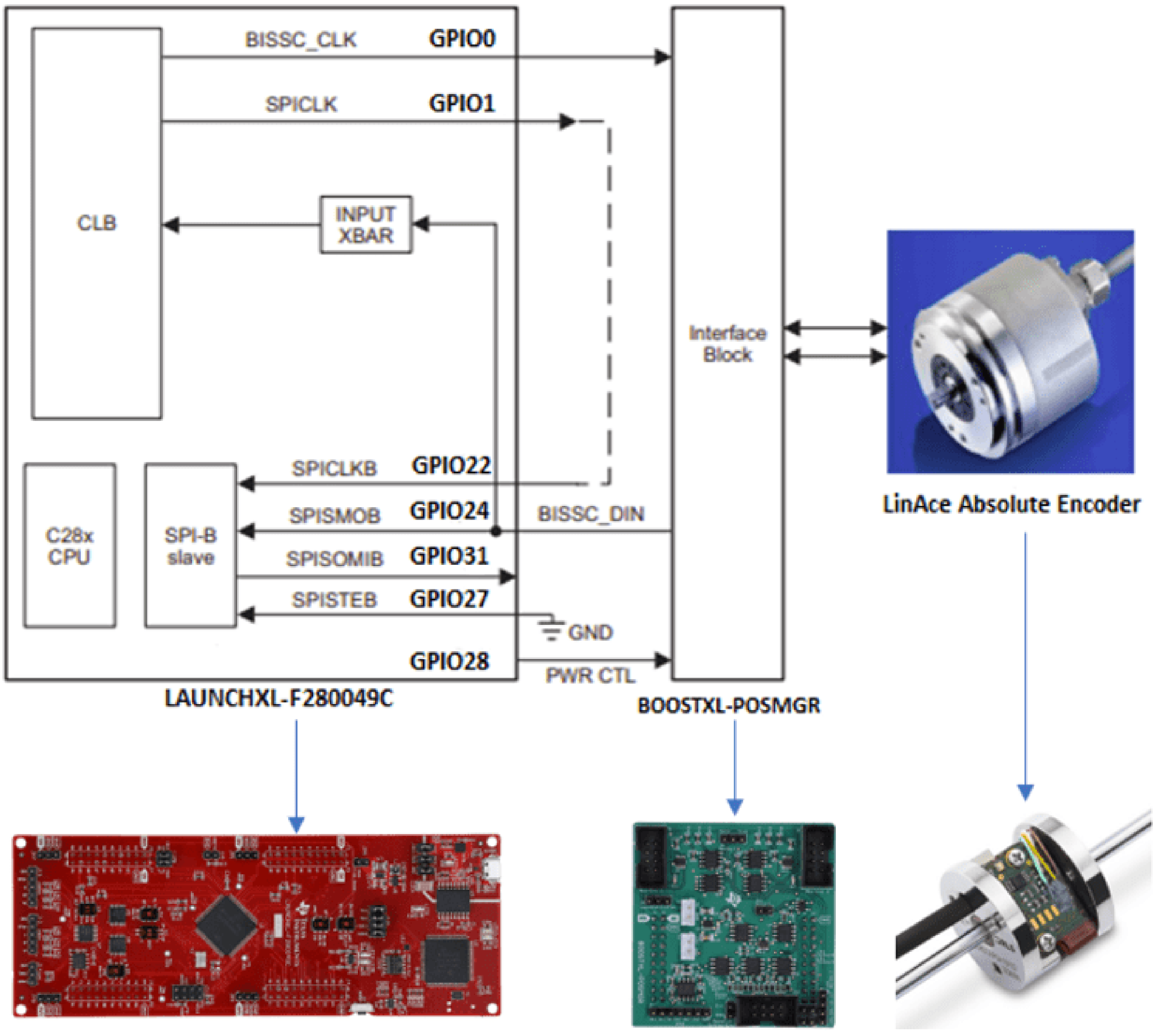
- Controller clock signals (represented as MA in the diagram) transmits position acquisition requests and timing information (clock) from controller to encoder.
- Encoder data out (represented as SLO in the diagram) transfers position data from encoder to controller, synchronized to MA.

This diagram shows the data format for BiSS-C protocol.



Introduction

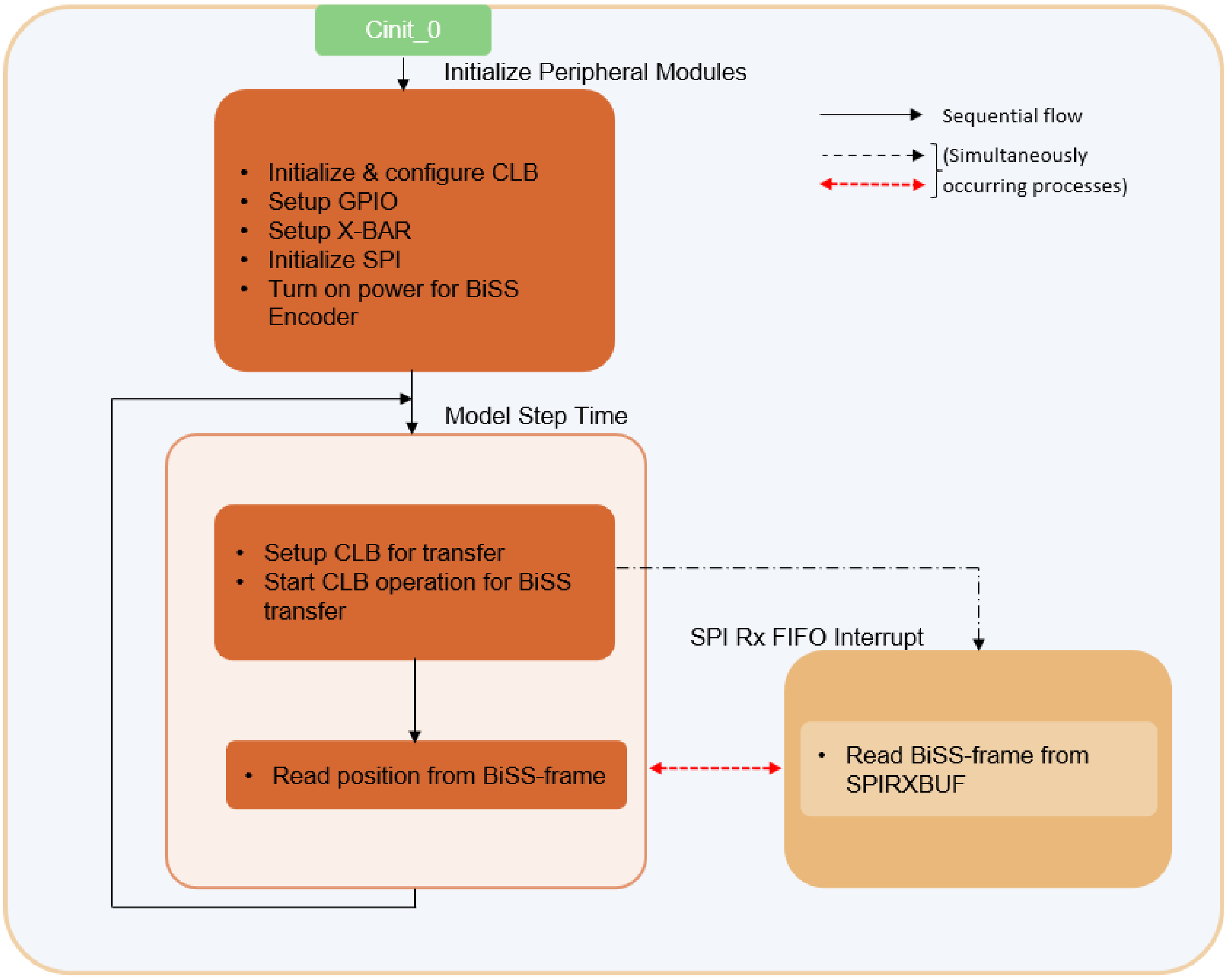
BiSS protocol implemented TI C2000 F280049C MCU through TI BOOSTXL-POSMGR Position Manager Booster Pack is used to interface the linear shaft absolute encoder from LinAce. Communication over BiSS-C interface is achieved primarily by CPU (C28x), configurable logic block (CLB), serial peripheral interface (SPI), and device interconnects (X-BARs).



- CLB is configured to generate the BiSS-C protocol clock signals (BiSSC_CLK and SPICLK).
- BiSS-C protocol clock signals are sent through ePWM channels(GPIO0 &1)
- SPI performs the encoder data receive functions.
- Input X-BAR is configured to enable the CLB to monitor the SPISIMO signal for detecting the start pulse and accordingly adjusts the phase of the receive clock (SPICLK).

Implementation

This workflow describes the BiSS-C implementation with respect to the TI F280049C Launchpad.


Cinit_0
Initialize Peripheral Modules
Initialize & configure CLB
Setup GPIO
Setup X-BAR
Initialize SPI
Turn on power for BiSS Encoder
Sequential flow
(Simultaneously occurring processes)
Model Step Time
Setup CLB for transfer
Start CLB operation for BiSS transfer
SPI Rx FIFO Interrupt
Read position from BiSS-frame
Read BiSS-frame from SPIRXBUF

- The CPU sets up a CLB for current BiSS-C command depending on the data packet format of the LinAce encoder as shown here:

Position	Status		CRC (inverted)
	Error	Warning	
26 bits	1 bit	1 bit	6 bits

- At each step time in the model, CPU sets up the CLB operation to generate BiSS-C command to read the encoder position.
- CLB TILE3 generates a 2MHz reference clock within the CLB module and number of clock pulses (BiSS_CLK-49 pulses and SPICLK-36 pulses), including the edge placement for these two clocks which is precisely controlled by the CLB TILE1 at this rate.
- A SPI interrupt is generated after receiving the data packet from the encoder. CPU then reads and decodes the encoder position from the SPI Receive FIFO.

Required Hardware

- F280049C LaunchPad
- Texas Instruments BOOSTXL-POSMGR
- LinAce™ absolute linear shaft encoder
- 5V external power supply
- Micro-USB cable

Available Models

You can use the Position Reading with a BiSS-C Absolute Encoder to generate code and load it on the F280049C LaunchPad.

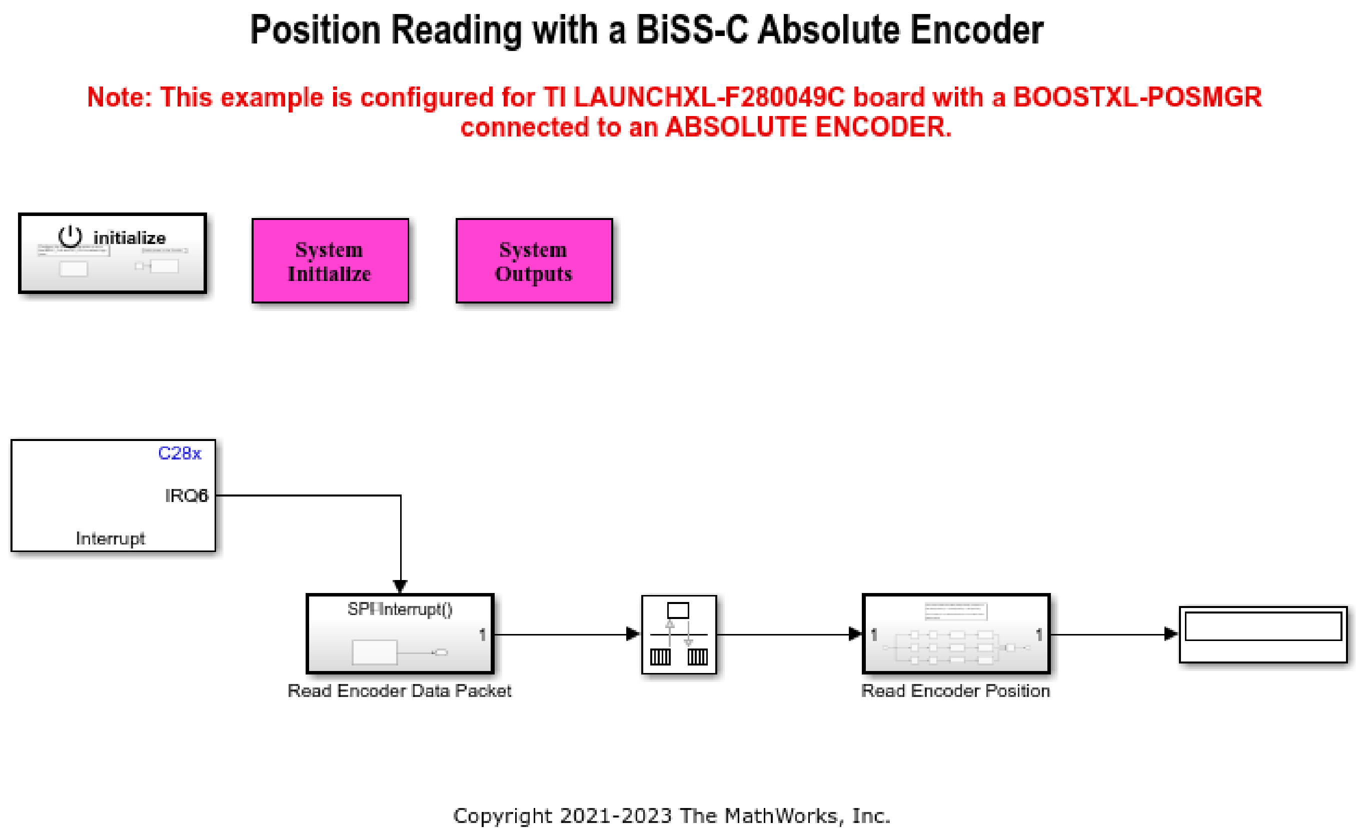
Hardware Connections

- Connect the **Texas Instruments Piccolo F280049C LaunchPad** to **Texas Instruments BOOSTXL-POSMGR**. For more information, refer to “Connect F280049C LaunchPad to BOOSTXL-POSMGR” on page 1-115.
- Connect the **Texas Instruments BOOSTXL-POSMGR** to **BiSS-C Absolute Encoder**. For more information, refer to “Connect BOOSTXL-POSMGR to Absolute Encoder” on page 1-116.
- Encoder Connections. For more information, refer to “Encoder Connections” on page 1-117.

Model

To open the model, type the following command in the MATLAB® prompt.

```
open_system('f280049C_BiSSC_Protocol');
```

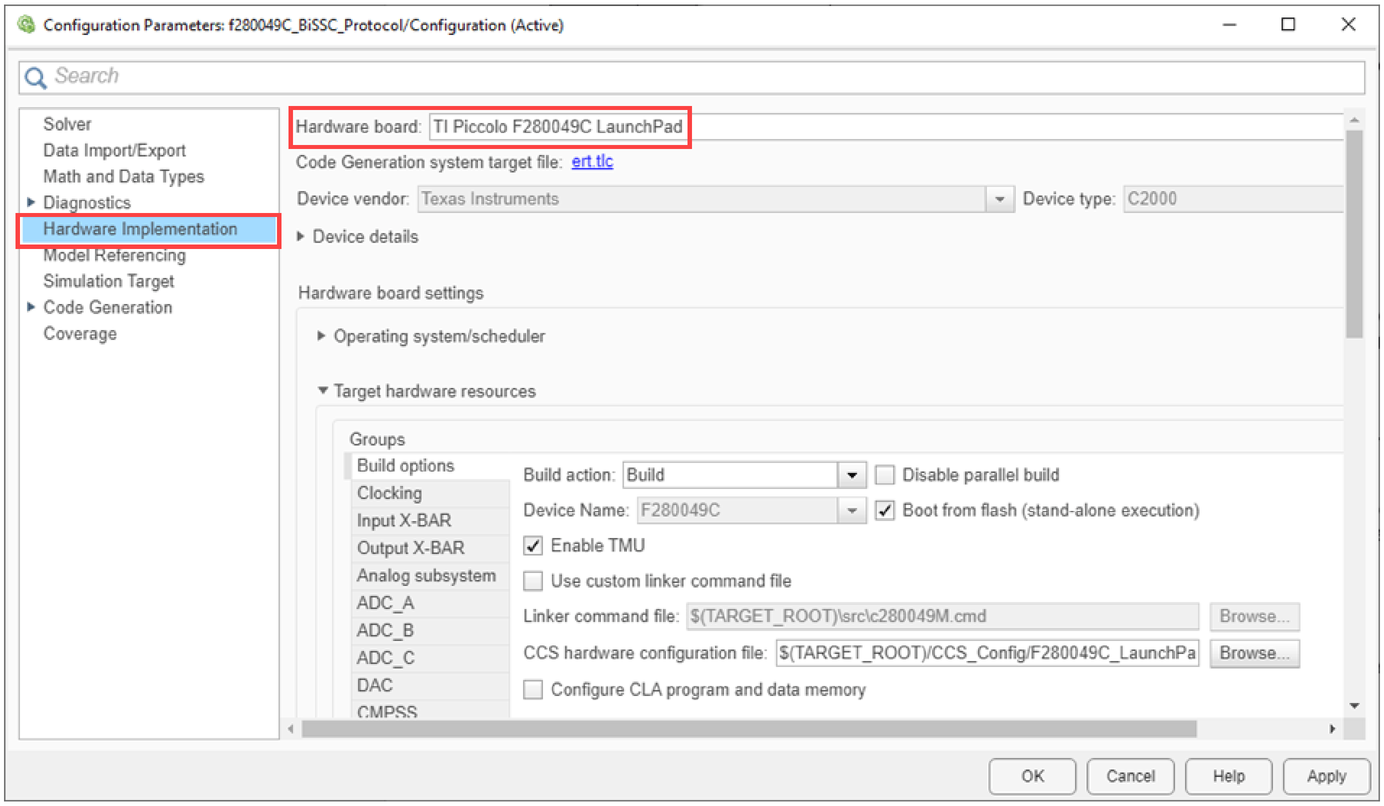

Position Reading with a BiSS-C Absolute Encoder
Note: This example is configured for TI LAUNCHXL-F280049C board with a BOOSTXL-POSMGR connected to an ABSOLUTE ENCODER.
initialize
System Initialize
System Outputs
C28x
IRQ6
Interrupt
SPIInterrupt()
1
Read Encoder Data Packet
1
1
Read Encoder Position
Copyright 2021-2023 The MathWorks, Inc.

This example demonstrates how to generate the BiSS-C command from the MCU (Controller) to read the position values from the encoder.

Task 1 - Configure and Run Position Reading with a BiSS-C Absolute Encoder Model

1. Open the Position Reading with a BiSS-C Absolute Encoder model. This model is configured for TI Piccolo F280049C LaunchPad hardware.

2. To run the model on other TI C2000 processors, first press **Ctrl+E** to open the Configuration Parameters dialog box. Then, select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.


Configuration Parameters: f280049C_BiSSC_Protocol/Configuration (Active)

Hardware board: TI Piccolo F280049C LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Build action: Build

Disable parallel build

Device Name: F280049C

☑ Boot from flash (stand-alone execution)

☑ Enable TMU

☐ Use custom linker command file

Linker command file: $(TARGET_ROOT)\src\c280049M.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/F280049C_LaunchPa

☐ Configure CLA program and data memory

Note: Ensure that the BOOSTXL-POSMGR is connected to the selected hardware board with correct pin mappings.

3. Perform the peripherals configurations in the model as shown in the Task 2. You can use the same parameter values if you want to run this example for other hardware boards.

Task 2 - Peripheral Configurations

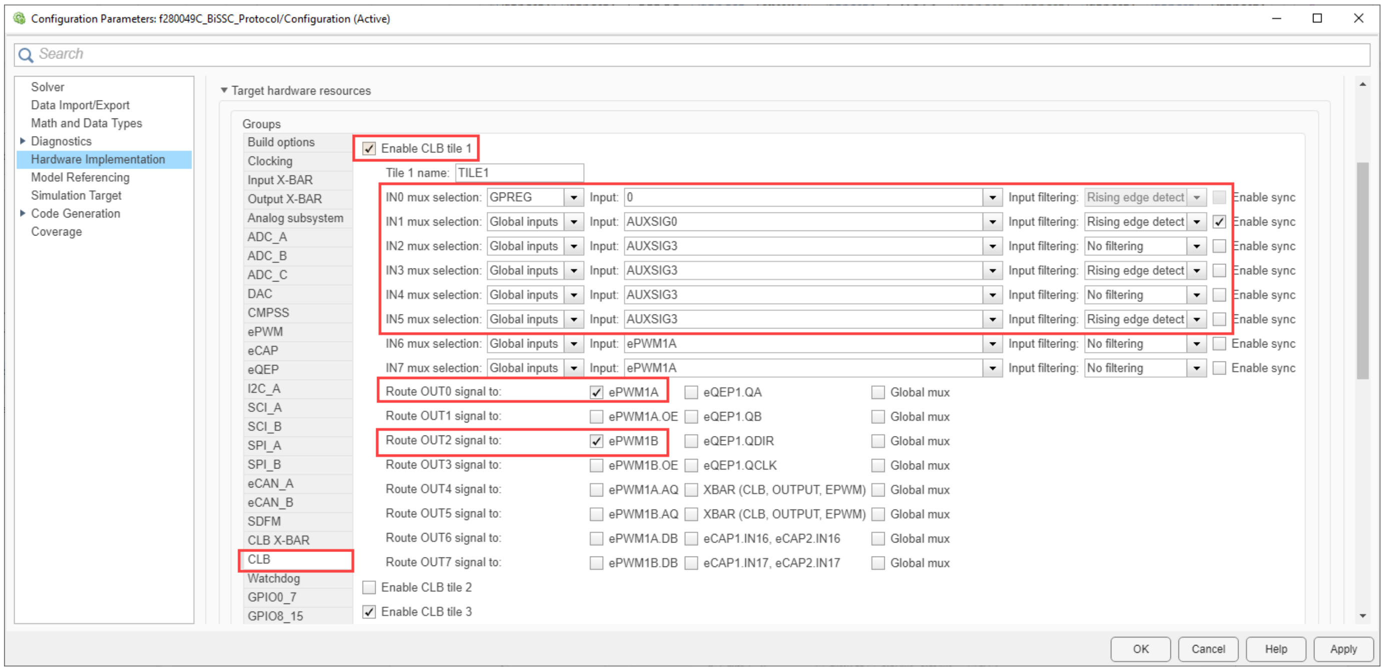
Set the peripheral block configurations for this model. Double-click the blocks to open block parameter configurations. You can use the same parameter values if you want to run this example on other hardware boards.

CLB Tile, CLB X-BAR, Input X-bar, ePWM are configured to generate `BiSS Clock` and `SPI clock`.

- **CLB**

For CLB Tile 1:

- Configure GPREG for Tile boundary IN0 to control the clock generation sequence.
- The generated BISSC_CLK and SPICLK clock outputs generated on TILE1 are routed to the ePWM1A and ePWM1B output pins. The remaining tile boundary configurations are explained in the subsequent sections.



Configuration Parameters: f280049C_BiSSC_Protocol/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage

Target hardware resources

Groups: Build options, Clocking, Input X-BAR, Output X-BAR, Analog subsystem, ADC_A, ADC_B, ADC_C, DAC, CMPSS, ePWM, eCAP, eQEP, I2C_A, SCI_A, SCI_B, SPI_A, SPI_B, eCAN_A, eCAN_B, SDFM, CLB X-BAR, CLB, Watchdog, GPIO0_7, GPIO8_15

☑ Enable CLB tile 1

Tile 1 name: TILE1

IN0 mux selection: GPREG	Input: 0	Input filtering: Rising edge detect	☐ Enable sync
IN1 mux selection: Global inputs	Input: AUXSIG0	Input filtering: Rising edge detect	☑ Enable sync
IN2 mux selection: Global inputs	Input: AUXSIG3	Input filtering: No filtering	☐ Enable sync
IN3 mux selection: Global inputs	Input: AUXSIG3	Input filtering: Rising edge detect	☐ Enable sync
IN4 mux selection: Global inputs	Input: AUXSIG3	Input filtering: No filtering	☐ Enable sync
IN5 mux selection: Global inputs	Input: AUXSIG3	Input filtering: Rising edge detect	☐ Enable sync
IN6 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync
IN7 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync

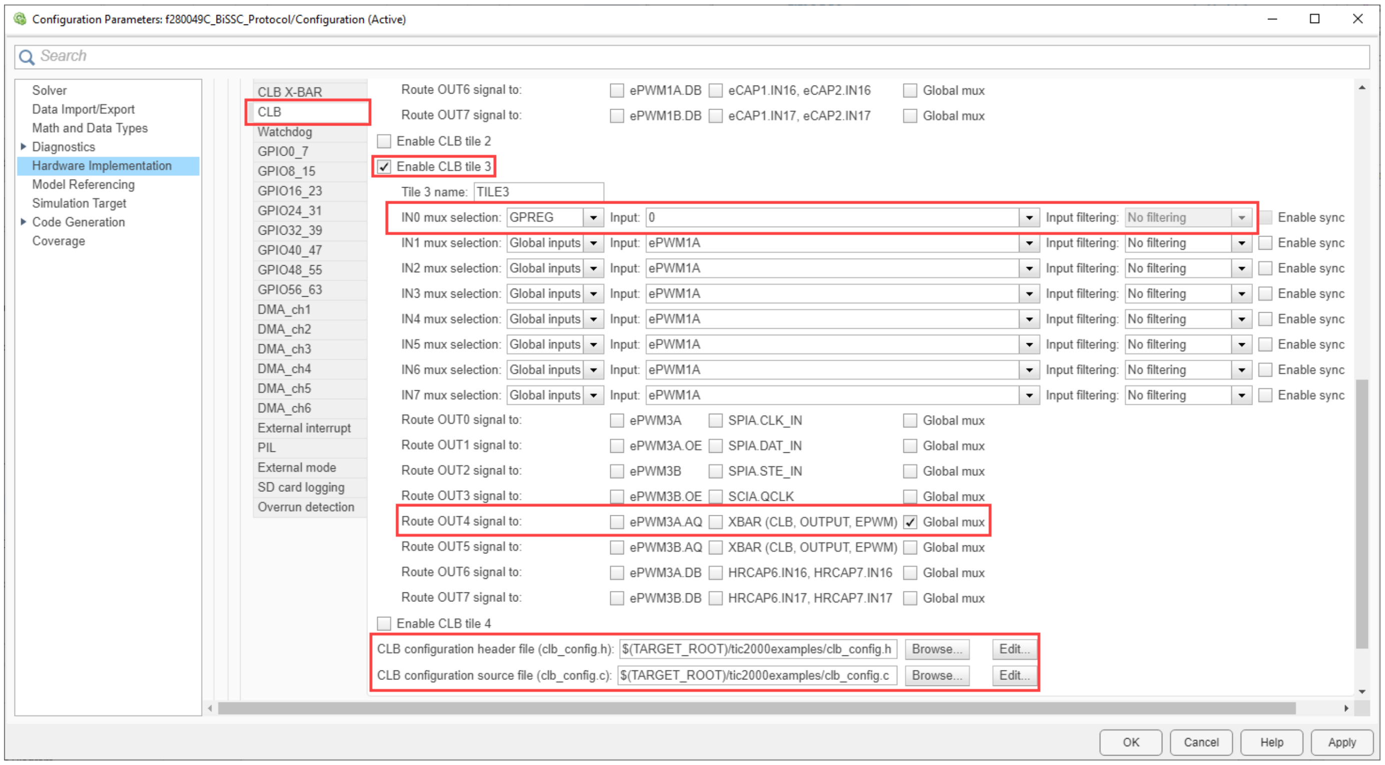
Route OUT0 signal to:	☑ ePWM1A	☐ eQEP1.QA	☐ Global mux
Route OUT1 signal to:	☐ ePWM1A.OE	☐ eQEP1.QB	☐ Global mux
Route OUT2 signal to:	☑ ePWM1B	☐ eQEP1.QDIR	☐ Global mux
Route OUT3 signal to:	☐ ePWM1B.OE	☐ eQEP1.QCLK	☐ Global mux
Route OUT4 signal to:	☐ ePWM1A.AQ	☐ XBAR (CLB, OUTPUT, EPWM)	☐ Global mux
Route OUT5 signal to:	☐ ePWM1B.AQ	☐ XBAR (CLB, OUTPUT, EPWM)	☐ Global mux
Route OUT6 signal to:	☐ ePWM1A.DB	☐ eCAP1.IN16, eCAP2.IN16	☐ Global mux
Route OUT7 signal to:	☐ ePWM1B.DB	☐ eCAP1.IN17, eCAP2.IN17	☐ Global mux

☐ Enable CLB tile 2

☑ Enable CLB tile 3

For CLB Tile 3:

- GPREG settings for TILE3 boundary IN0
- TILE3 OUT4 is routed to Global Mux which then is configured in CLB X-BAR to be used by TILE1.



Configuration Parameters: f280049C_BiSSC_Protocol/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage

Groups: CLB X-BAR, CLB, Watchdog, GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, GPIO56_63, DMA_ch1, DMA_ch2, DMA_ch3, DMA_ch4, DMA_ch5, DMA_ch6, External interrupt, PIL, External mode, SD card logging, Overrun detection

Route OUT6 signal to:	☐ ePWM1A.DB	☐ eCAP1.IN16, eCAP2.IN16	☐ Global mux
Route OUT7 signal to:	☐ ePWM1B.DB	☐ eCAP1.IN17, eCAP2.IN17	☐ Global mux

☐ Enable CLB tile 2

☑ Enable CLB tile 3

Tile 3 name: TILE3

IN0 mux selection: GPREG	Input: 0	Input filtering: No filtering	☐ Enable sync
IN1 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync
IN2 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync
IN3 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync
IN4 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync
IN5 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync
IN6 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync
IN7 mux selection: Global inputs	Input: ePWM1A	Input filtering: No filtering	☐ Enable sync

Route OUT0 signal to:	☐ ePWM3A	☐ SPIA.CLK_IN	☐ Global mux
Route OUT1 signal to:	☐ ePWM3A.OE	☐ SPIA.DAT_IN	☐ Global mux
Route OUT2 signal to:	☐ ePWM3B	☐ SPIA.STE_IN	☐ Global mux
Route OUT3 signal to:	☐ ePWM3B.OE	☐ SCIA.QCLK	☐ Global mux
Route OUT4 signal to:	☐ ePWM3A.AQ	☐ XBAR (CLB, OUTPUT, EPWM)	☑ Global mux
Route OUT5 signal to:	☐ ePWM3B.AQ	☐ XBAR (CLB, OUTPUT, EPWM)	☐ Global mux
Route OUT6 signal to:	☐ ePWM3A.DB	☐ HRCAP6.IN16, HRCAP7.IN16	☐ Global mux
Route OUT7 signal to:	☐ ePWM3B.DB	☐ HRCAP6.IN17, HRCAP7.IN17	☐ Global mux

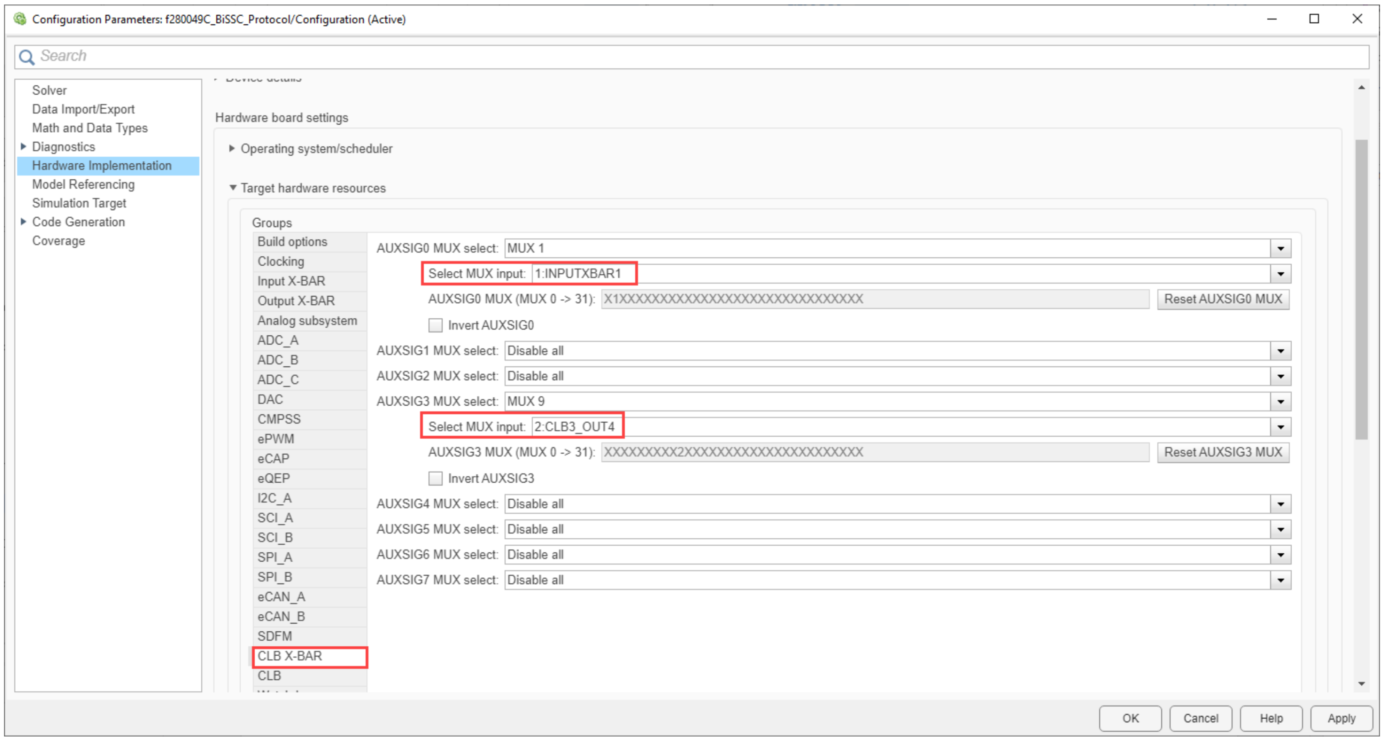
☐ Enable CLB tile 4

CLB configuration header file (clb_config.h): $(TARGET_ROOT)/tic2000examples/clb_config.h

CLB configuration source file (clb_config.c): $(TARGET_ROOT)/tic2000examples/clb_config.c

The CLB configuration files, that is the header file and the source file generated from the CCS CLB config tool, are imported into the Simulink model. For more information on how to configure the header file and the source file, see “CLB Logic for Clock Generation” on page 1-119.

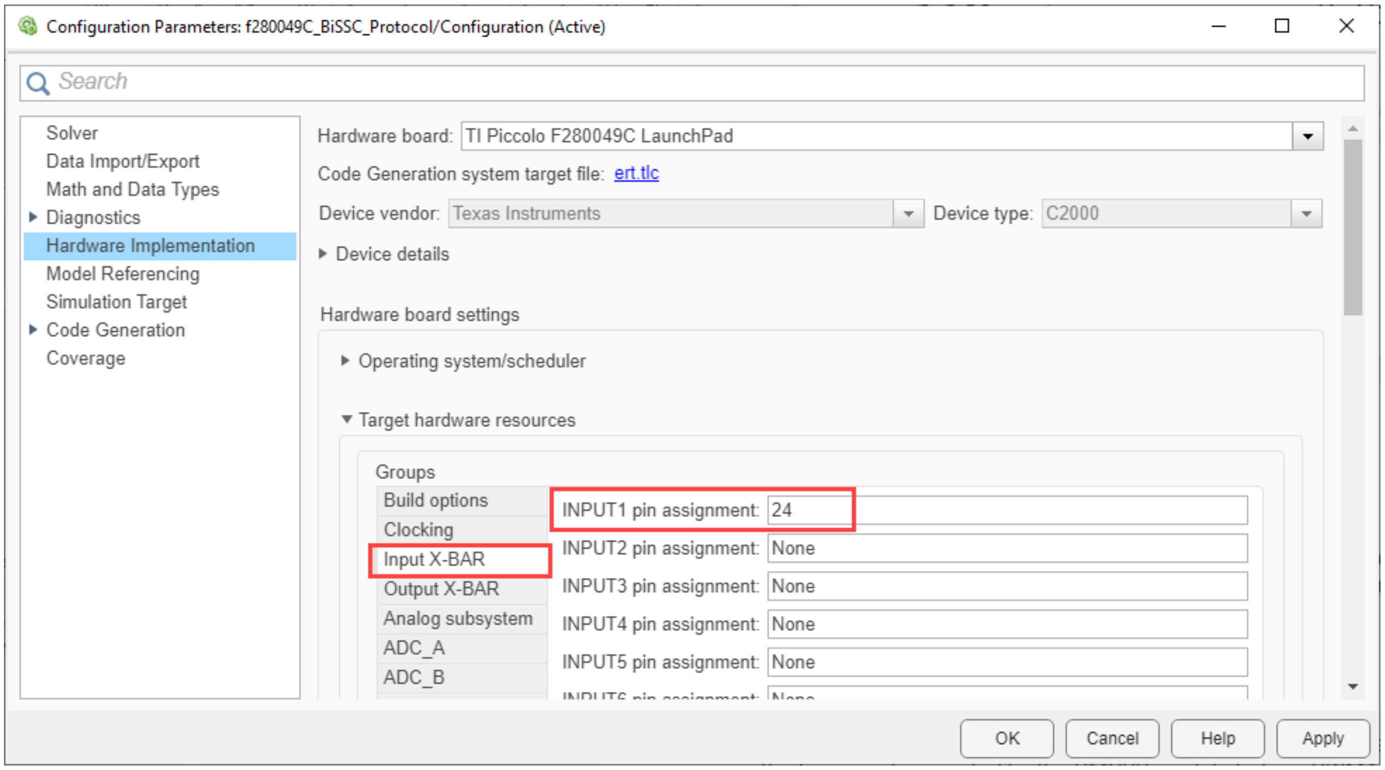
- **CLB X-BAR**
- AUXSIG0 Mux select is configured to connect the INPUTXBAR1 i.e BISSC_DIN line to the input of the TILE1 BOUNDARY IN1.
- AUXSIG3 Mux select is configured to connect the CLB3_OUT4 (clock source of 2MHz for TILE1) to the input of the TILE1_BOUNDARY in2, in3, in4 and in5.


Configuration Parameters: f280049C_BiSSC_Protocol/Configuration (Active)

Hardware board settings

AUXSIG0 MUX select: MUX 1
Select MUX input: 1:INPUTXBAR1
AUXSIG0 MUX (MUX 0 -> 31): X1XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Invert AUXSIG0
AUXSIG1 MUX select: Disable all
AUXSIG2 MUX select: Disable all
AUXSIG3 MUX select: MUX 9
Select MUX input: 2:CLB3_OUT4
AUXSIG3 MUX (MUX 0 -> 31): XXXXXXXXX2XXXXXXXXXXXXXXXXXXXXXX
Invert AUXSIG3
AUXSIG4 MUX select: Disable all
AUXSIG5 MUX select: Disable all
AUXSIG6 MUX select: Disable all
AUXSIG7 MUX select: Disable all

- **Input X-BAR**
- SPIB_SIMO on GPIO24(BISSC_DIN) line over which encoder response is generated is configured in INPUT X-BAR INPUT1.
- This setting enables encoder start pulse to be read inside TILE1 by the TILE1_FSM_1 followed by TILE1_FSM_2 to trigger the TILE1_COUNTER_2 to generate the SPICLK clock cycles to read the data packet.


Configuration Parameters: f280049C_BiSSC_Protocol/Configuration (Active)

Hardware board: TI Piccolo F280049C LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

INPUT1 pin assignment: 24

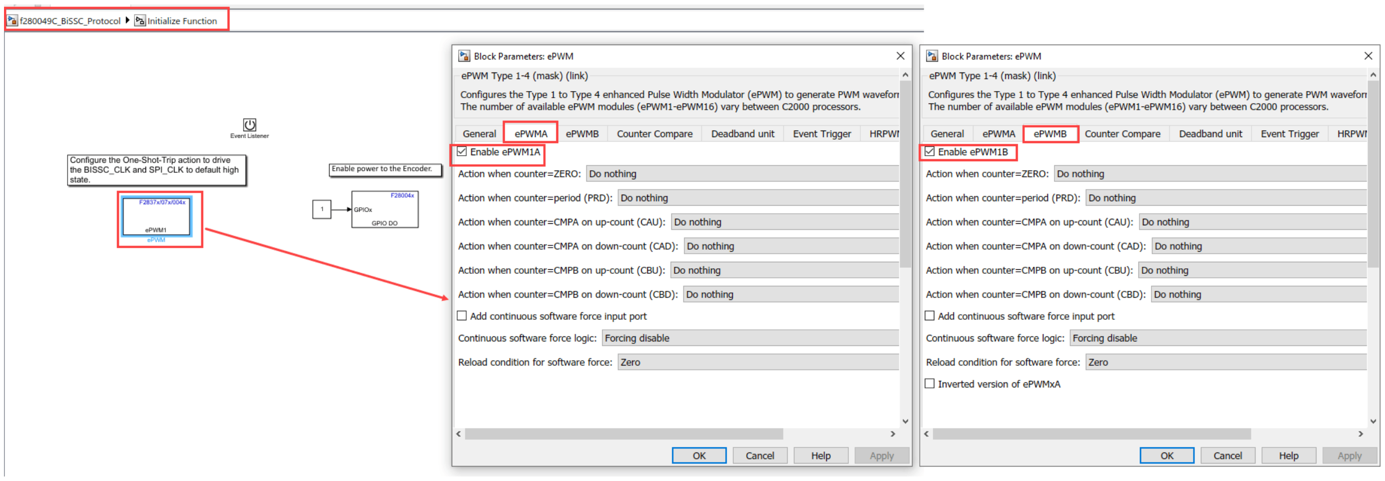
INPUT2 pin assignment: None

INPUT3 pin assignment: None

INPUT4 pin assignment: None

INPUT5 pin assignment: None

- **ePWM Block**
- Configure ePWM One Shot Trip option to drive the outputs GPIO0 (ePWM1A) and GPIO1 (ePWM1B) to high state i.e the idle state for the output.



Configure the One-Shot-Trip action to drive the BISSC_CLK and SPI_CLK to default high state.

Enable power to the Encoder.

Block Parameters: ePWM

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM wavefor
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

☑ Enable ePWM1A

Action when counter=ZERO: Do nothing

Action when counter=period (PRD): Do nothing

Action when counter=CMPA on up-count (CAU): Do nothing

Action when counter=CMPA on down-count (CAD): Do nothing

Action when counter=CMPB on up-count (CBU): Do nothing

Action when counter=CMPB on down-count (CBD): Do nothing

☐ Add continuous software force input port

Continuous software force logic: Forcing disable

Reload condition for software force: Zero

Block Parameters: ePWM

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM wavefor
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

☑ Enable ePWM1B

Action when counter=ZERO: Do nothing

Action when counter=period (PRD): Do nothing

Action when counter=CMPA on up-count (CAU): Do nothing

Action when counter=CMPA on down-count (CAD): Do nothing

Action when counter=CMPB on up-count (CBU): Do nothing

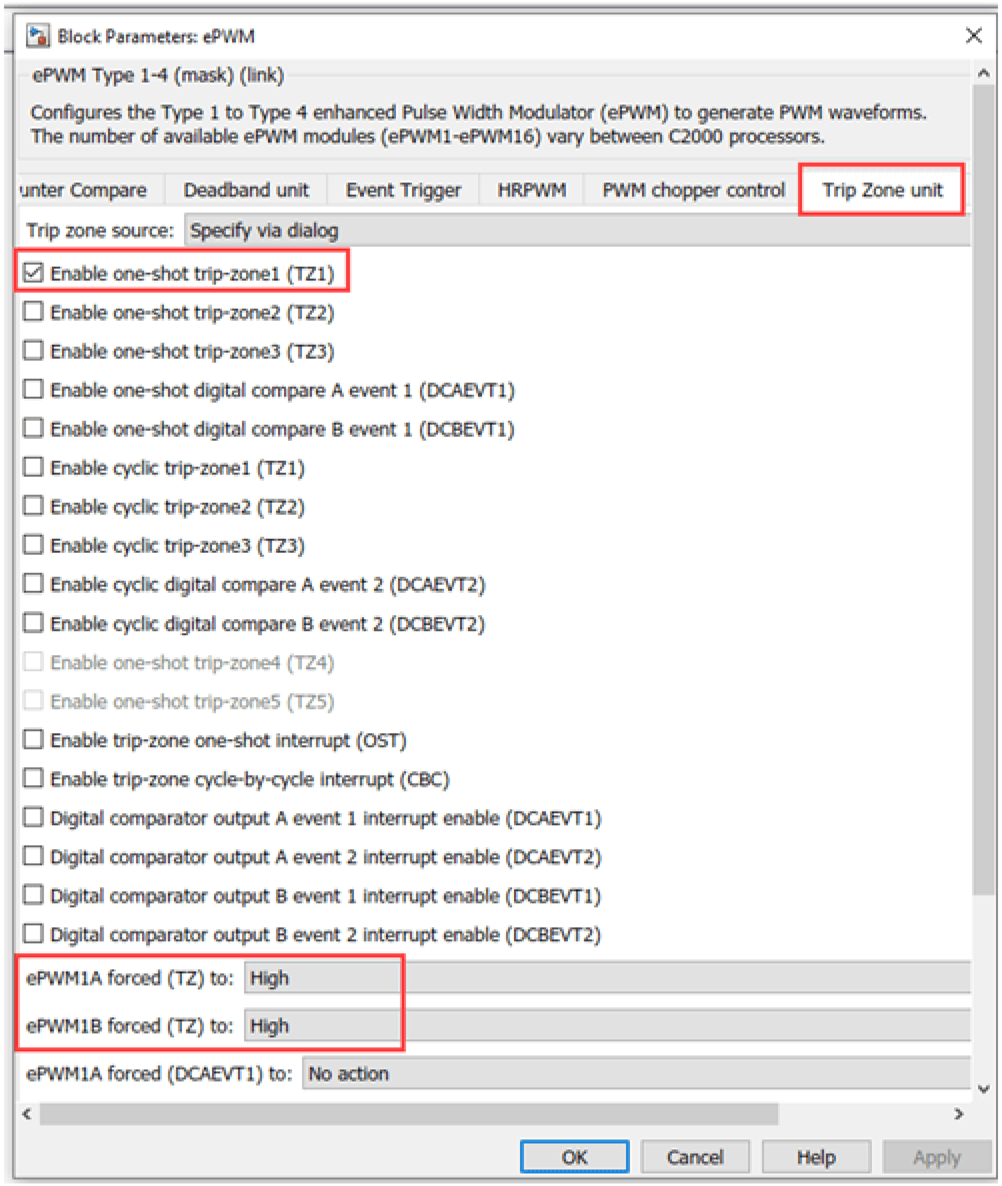
Action when counter=CMPB on down-count (CBD): Do nothing

☐ Add continuous software force input port

Continuous software force logic: Forcing disable

Reload condition for software force: Zero

☐ Inverted version of ePWMxA


Block Parameters: ePWM

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms. The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

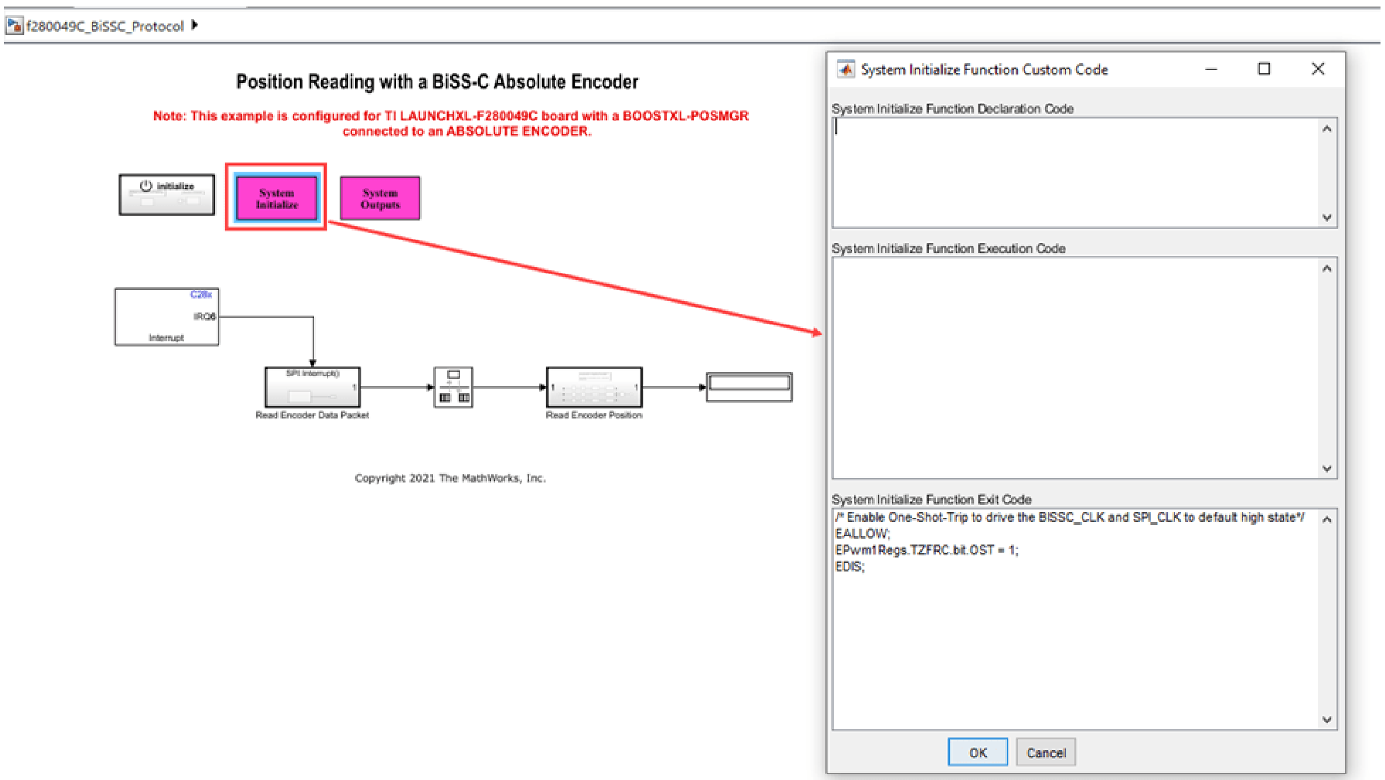
Trip zone source: Specify via dialog

- [✓] Enable one-shot trip-zone1 (TZ1)
- [ ] Enable one-shot trip-zone2 (TZ2)
- [ ] Enable one-shot trip-zone3 (TZ3)
- [ ] Enable one-shot digital compare A event 1 (DCAEVT1)
- [ ] Enable one-shot digital compare B event 1 (DCBEVT1)
- [ ] Enable cyclic trip-zone1 (TZ1)
- [ ] Enable cyclic trip-zone2 (TZ2)
- [ ] Enable cyclic trip-zone3 (TZ3)
- [ ] Enable cyclic digital compare A event 2 (DCAEVT2)
- [ ] Enable cyclic digital compare B event 2 (DCBEVT2)
- [ ] Enable one-shot trip-zone4 (TZ4)
- [ ] Enable one-shot trip-zone5 (TZ5)
- [ ] Enable trip-zone one-shot interrupt (OST)
- [ ] Enable trip-zone cycle-by-cycle interrupt (CBC)
- [ ] Digital comparator output A event 1 interrupt enable (DCAEVT1)
- [ ] Digital comparator output A event 2 interrupt enable (DCAEVT2)
- [ ] Digital comparator output B event 1 interrupt enable (DCBEVT1)
- [ ] Digital comparator output B event 2 interrupt enable (DCBEVT2)

ePWM1A forced (TZ) to: High

ePWM1B forced (TZ) to: High

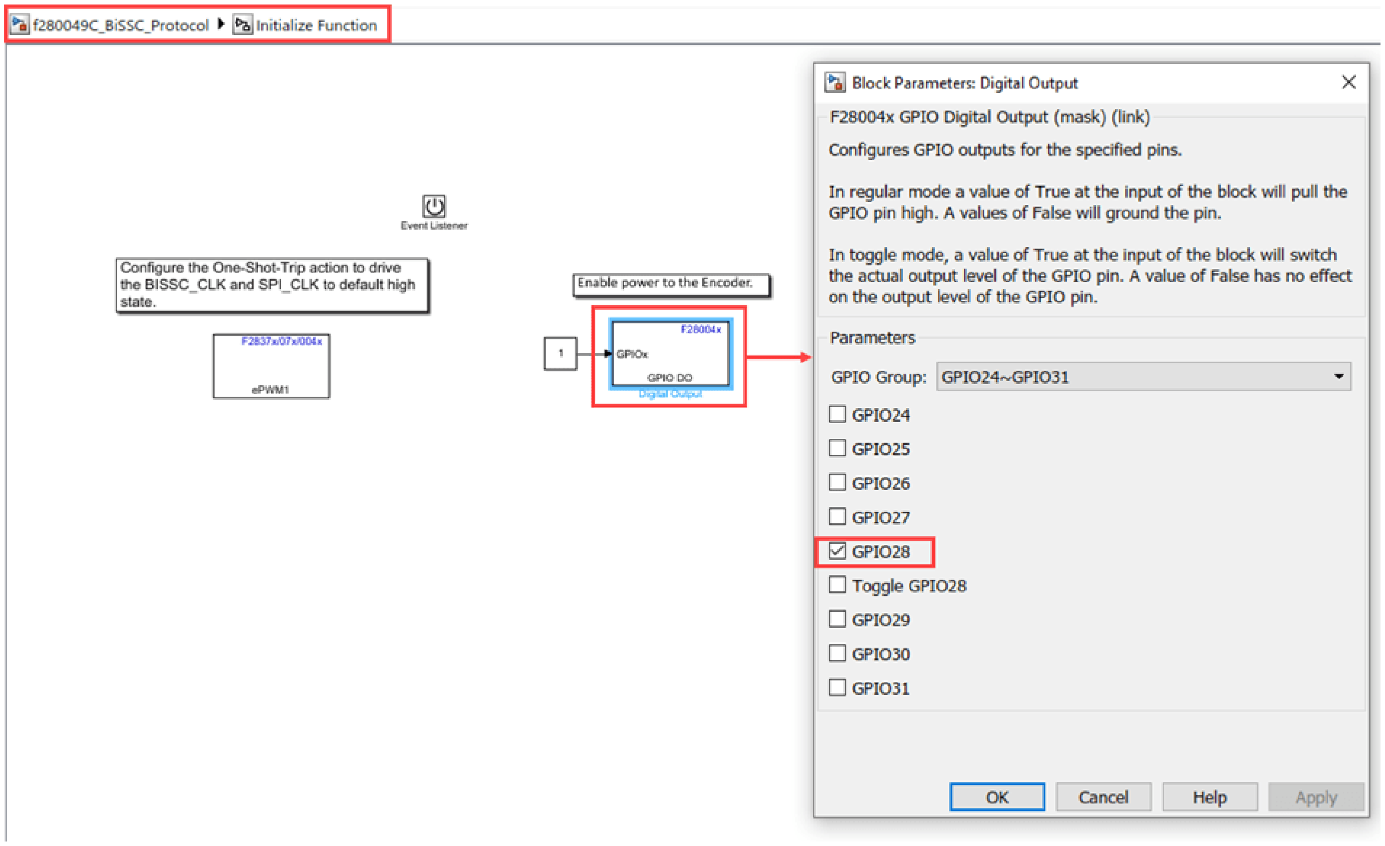
ePWM1A forced (DCAEVT1) to: No action


f280049C_BiSSC_Protocol
Position Reading with a BiSS-C Absolute Encoder
Note: This example is configured for TI LAUNCHXL-F280049C board with a BOOSTXL-POSMGR connected to an ABSOLUTE ENCODER.
initialize
System Initialize
System Outputs
C28x
IRQ6
Interrupt
SPI Interrupt()
Read Encoder Data Packet
Read Encoder Position
Copyright 2021 The MathWorks, Inc.
System Initialize Function Custom Code
System Initialize Function Declaration Code
System Initialize Function Execution Code
System Initialize Function Exit Code
OK
Cancel

```
/* Enable One-Shot-Trip to drive the BISSC_CLK and SPI_CLK to default high state*/
EALLOW;
EPwm1Regs.TZFRC.bit.OST = 1;
EDIS;
```

- **GPIO Block**

Enable the power to the encoder. Configure the GPIO28 output pin High to drive the encoder.



Configure the One-Shot-Trip action to drive the BISSC_CLK and SPI_CLK to default high state.

Enable power to the Encoder.

Block Parameters: Digital Output

F28004x GPIO Digital Output (mask) (link)

Configures GPIO outputs for the specified pins.

In regular mode a value of True at the input of the block will pull the GPIO pin high. A values of False will ground the pin.

In toggle mode, a value of True at the input of the block will switch the actual output level of the GPIO pin. A value of False has no effect on the output level of the GPIO pin.

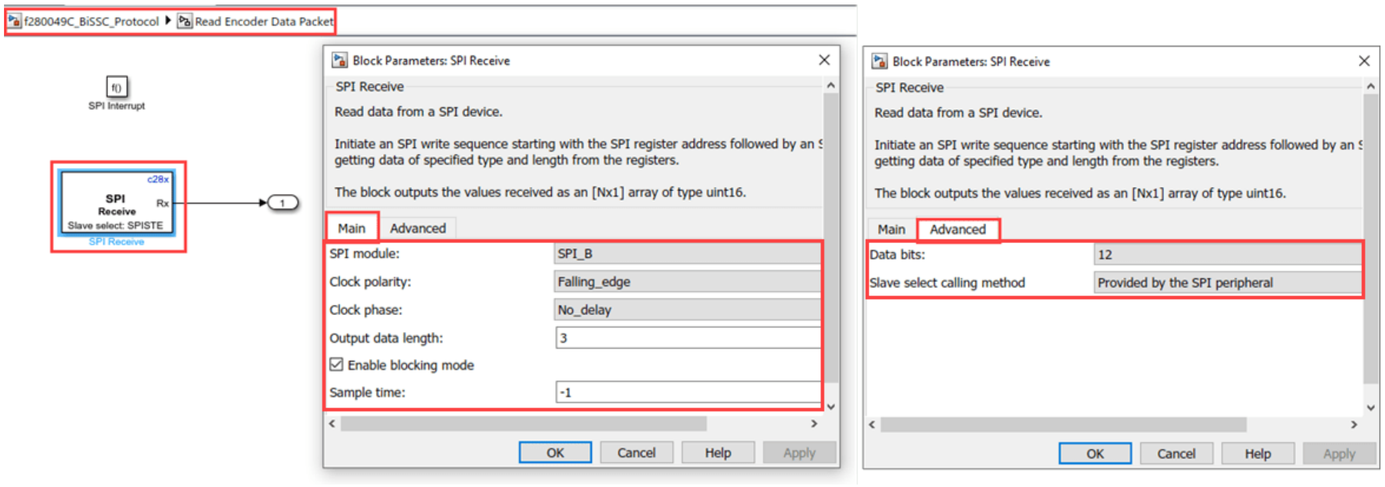
Parameters

GPIO Group: GPIO24~GPIO31

- [ ] GPIO24
- [ ] GPIO25
- [ ] GPIO26
- [ ] GPIO27
- [x] GPIO28
- [ ] Toggle GPIO28
- [ ] GPIO29
- [ ] GPIO30
- [ ] GPIO31

- **SPI Block**

Configure the SPI block to read the encoder position.



Block Parameters: SPI Receive

SPI Receive

Read data from a SPI device.

Initiate an SPI write sequence starting with the SPI register address followed by an S getting data of specified type and length from the registers.

The block outputs the values received as an [Nx1] array of type uint16.

Main | Advanced

SPI module:	SPI_B
Clock polarity:	Falling_edge
Clock phase:	No_delay
Output data length:	3
☑ Enable blocking mode	
Sample time:	-1

Block Parameters: SPI Receive

SPI Receive

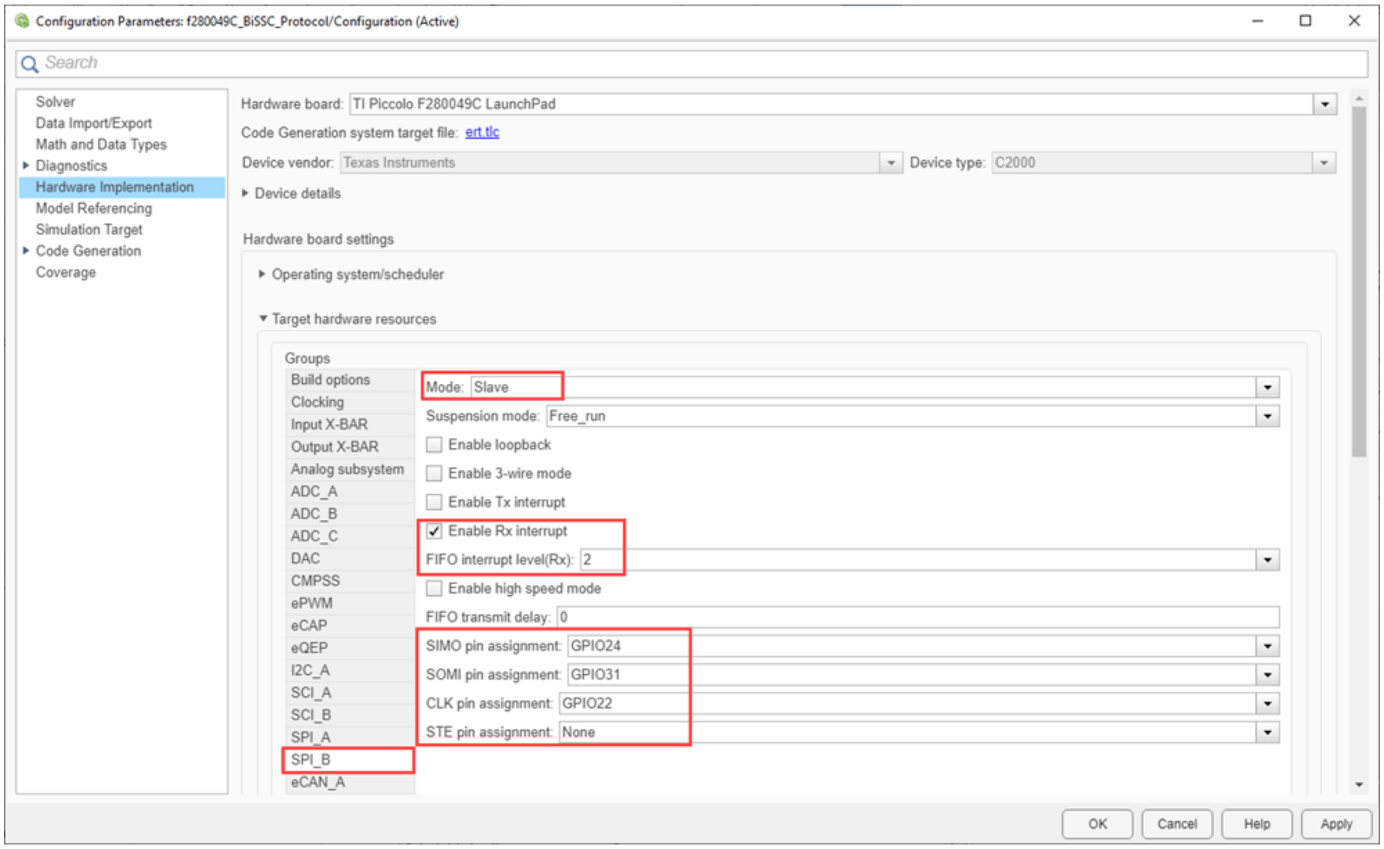
Read data from a SPI device.

Initiate an SPI write sequence starting with the SPI register address followed by an S getting data of specified type and length from the registers.

The block outputs the values received as an [Nx1] array of type uint16.

Main | Advanced

Data bits:	12
Slave select calling method	Provided by the SPI peripheral



Configuration Parameters: f280049C_BiSSC_Protocol/Configuration (Active)

Hardware board: TI Piccolo F280049C LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments   Device type: C2000

Hardware board settings

Target hardware resources

Groups: SPI_B

Mode: Slave

Suspension mode: Free_run

☐ Enable loopback

☐ Enable 3-wire mode

☐ Enable Tx interrupt

☑ Enable Rx interrupt

FIFO interrupt level(Rx): 2

☐ Enable high speed mode

FIFO transmit delay: 0

SIMO pin assignment: GPIO24

SOMI pin assignment: GPIO31

CLK pin assignment: GPIO22

STE pin assignment: None

Note: For SPI configuration, the STE pin on the BOOSTXL-POSMGR is connected to ground. Hence the pin is uninitialized from MCU.

Task 3 - CLB Logic for Clock Generation

In this task you will learn how to:

- `Update` or `Modify` the CLB Tile configurations. For more information, see “CLB Tile Configuration” on page 1-119.
- Generate CLB configuration file. For more information, see “Generate CLB Configuration File” on page 1-124.
- Generate CLB configuration HTML file. For more information, see “Generate CLB Configuration in HTML” on page 1-126.

Task 4 - Monitor Signals and Tune the Model

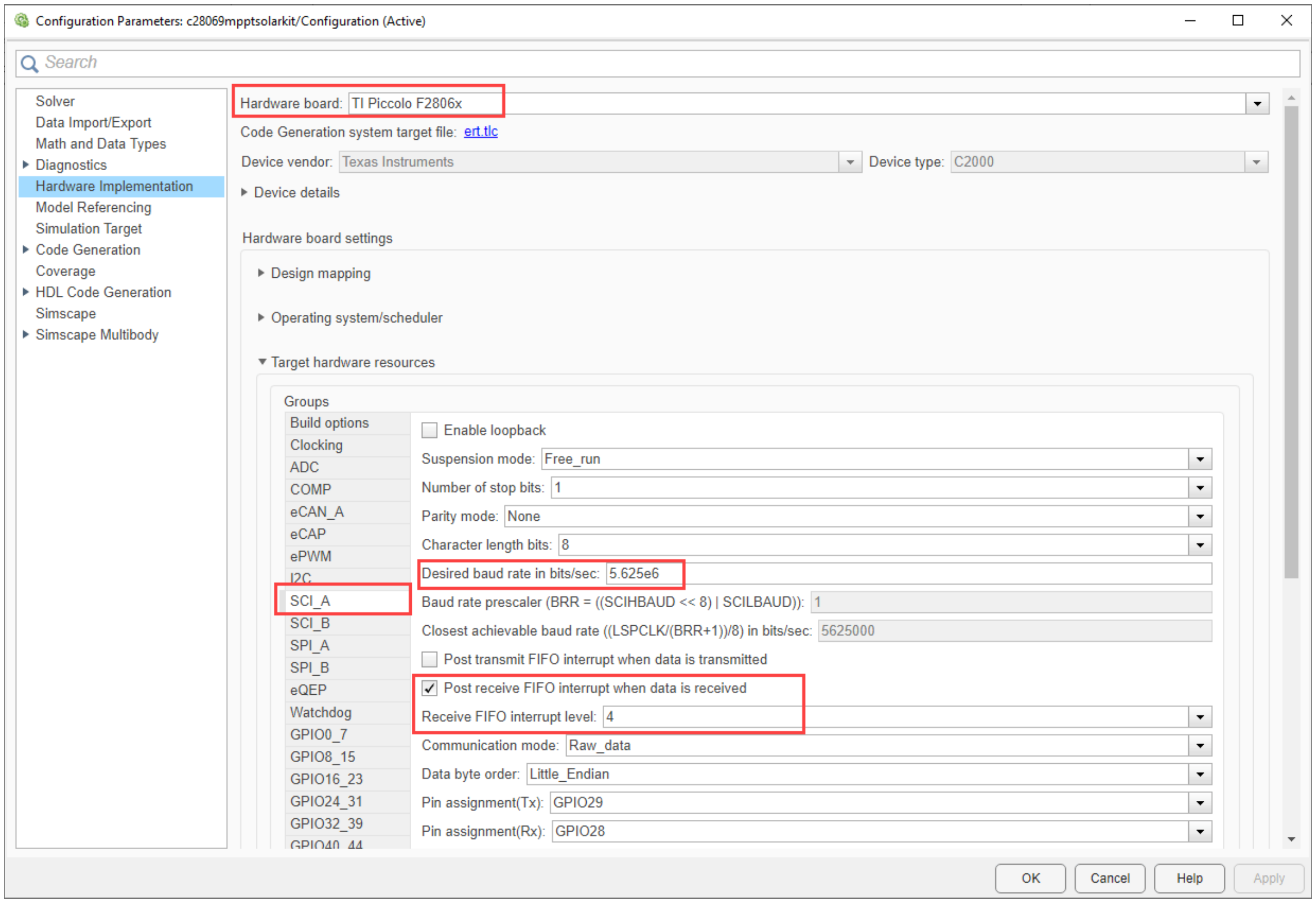
When you perform **Monitor & Tune** action for the model, the host computer communicates with the target on which the generated executable runs.

Prerequisites

Before executing Monitor & Tune, perform the following SCI configuration and Dip switch configuration.

SCI Configuration

As GPIO28 is used to enable the power to the encoder, use GPIO 35 or GPIO37 for UART to run the external mode.


Configuration Parameters: c28069mpptsolarkit/Configuration (Active)

Search

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
HDL Code Generation
Simscape
Simscape Multibody

Hardware board: TI Piccolo F2806x
Code Generation system target file: ert.tlc
Device vendor: Texas Instruments
Device type: C2000
Device details

Hardware board settings
Design mapping
Operating system/scheduler
Target hardware resources

Groups
Build options
Clocking
ADC
COMP
eCAN_A
eCAP
ePWM
I2C
SCI_A
SCI_B
SPI_A
SPI_B
eQEP
Watchdog
GPIO0_7
GPIO8_15
GPIO16_23
GPIO24_31
GPIO32_39
GPIO40_44

Enable loopback
Suspension mode: Free_run
Number of stop bits: 1
Parity mode: None
Character length bits: 8
Desired baud rate in bits/sec: 5.625e6
Baud rate prescaler (BRR = ((SCIHBAUD << 8) | SCILBAUD)): 1
Closest achievable baud rate ((LSPCLK/(BRR+1))/8) in bits/sec: 5625000
Post transmit FIFO interrupt when data is transmitted
✓ Post receive FIFO interrupt when data is received
Receive FIFO interrupt level: 4
Communication mode: Raw_data
Data byte order: Little_Endian
Pin assignment(Tx): GPIO29
Pin assignment(Rx): GPIO28

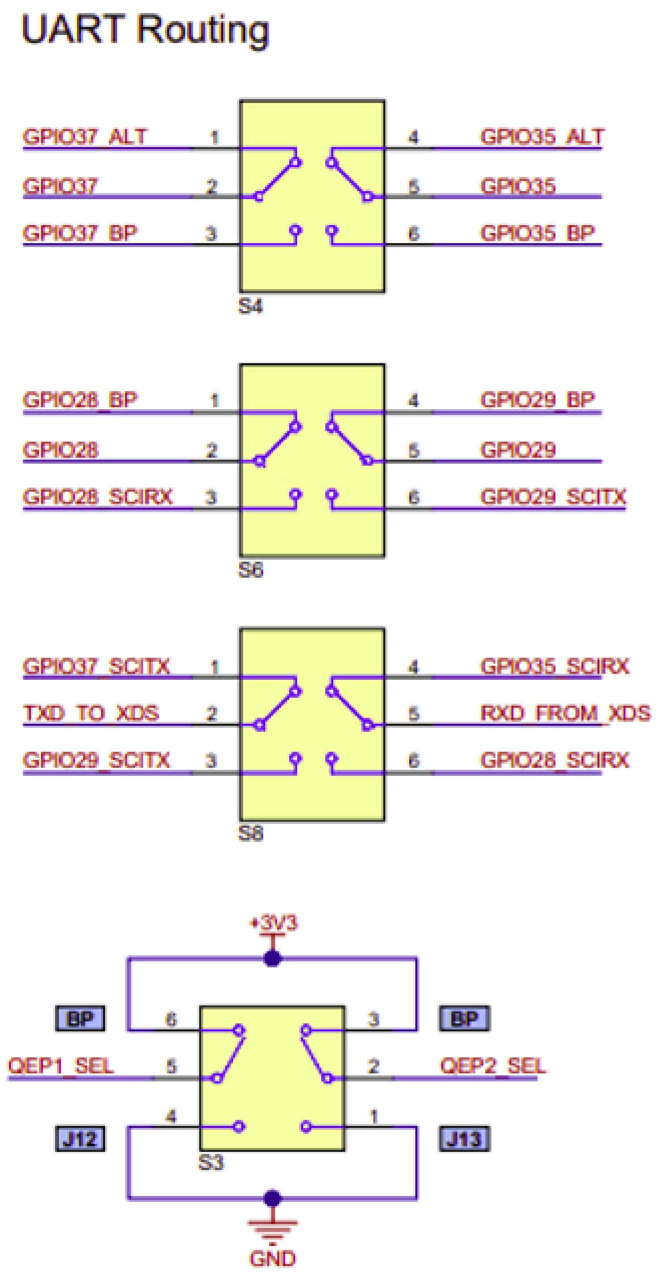
OK Cancel Help Apply

Dip Switch Configuration

Configure the dip switch to the following:

1. Dip switch S3, S4, S8: GPIO35 and GPIO37 routed to the virtual COM port for external mode operation.

2. Dip switch S6: Route the GPIO28-29 to BoosterPack Header. GPIO28 now enables the power to Encoder.



1. Open the Hardware tab and click **Monitor & Tune**. You can observe from the Diagnostic Viewer that the code is generated for the model and the host connects to the target after loading the generated executable.

Monitor Signals

While the model runs, you can monitor the following signals on the Display block. You can monitor the position counts of absolute encoder in the Display block.

Other Things to Try

- Try to run the example on F28379D target and analyze the results.
- BOOSTXL-POSMGR supports interfacing up-to 2 encoders. A second encoder can be interfaced with the similar approach.
- Perform Cyclic Redundancy Check(CRC) on the position data.

More About

- "Hardware Connections" on page 1-115
- "CLB Logic for Clock Generation" on page 1-119

MPPT Using Flyback Converter in TI Solar Micro Inverter Development Kit

This example shows how to implement a Maximum Power Point Tracking (MPPT) Algorithm along with control of DC-DC flyback converter using the C2000™ Microcontroller Blocket. The example uses the Texas Instruments™ Solar Micro Inverter Development Kit along with the Texas Instruments F28069M/F28035 controlCARD.

Using this example, you can:

- Simulate a plant model for a DC-DC flyback converter
- Simulate MPPT algorithm
- Generate code for the controller and load it on the controlCARD
- Monitor signals and tune parameters using the host computer

Required Hardware

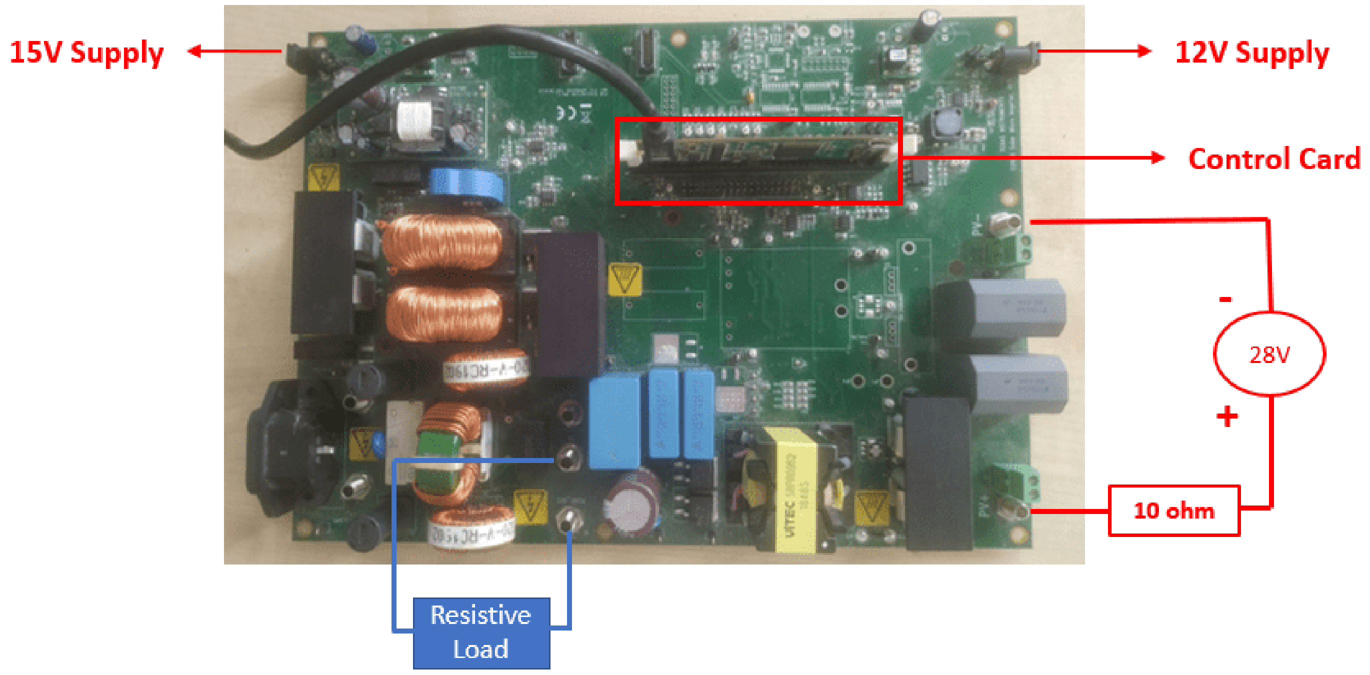
- Texas Instruments Solar Micro Inverter Development Kit (TMDSSOLARUINVKIT)
- F28035 controlCARD or F28069M controlCARD
- DC Voltage source 0-30V
- Resistance 10 ohm
- 60W Lamp

Available Models

- c28069mpptsolarkit can be used to simulate the plant model and generate code for the controller and deploy it on F28069 controlCARD.
- c2000_solarkit_host_model can be run on the host computer to log signals and tune parameters.

Hardware Connections

Make the hardware connection as shown in the diagram.



- Connect a DC Voltage source and a series resistance to `TP12(PV+)` and `TP13(PV-)`. You can also connect a Solar emulator or panel to the input terminals.
- Ensure you modify the controller parameters, MPPT parameters accordingly.
- Connect the external `+15VDC` isolated bias supply to `C:JP3` connector.
- Connect the external `+12VDC` isolated bias supply to `C:JP2` connector.
- Connect a USB cable from the ISO Control Card to the PC on which host model needs to run. Verify that the LED `LD4` on the control card is `ON` indicating USB connection.
- Connect a resistive load or lamp at the Flyback output stage TP8(VBus Flyback) and TP10(PGND).

Simulate the DC-DC Flyback Converter with MPPT

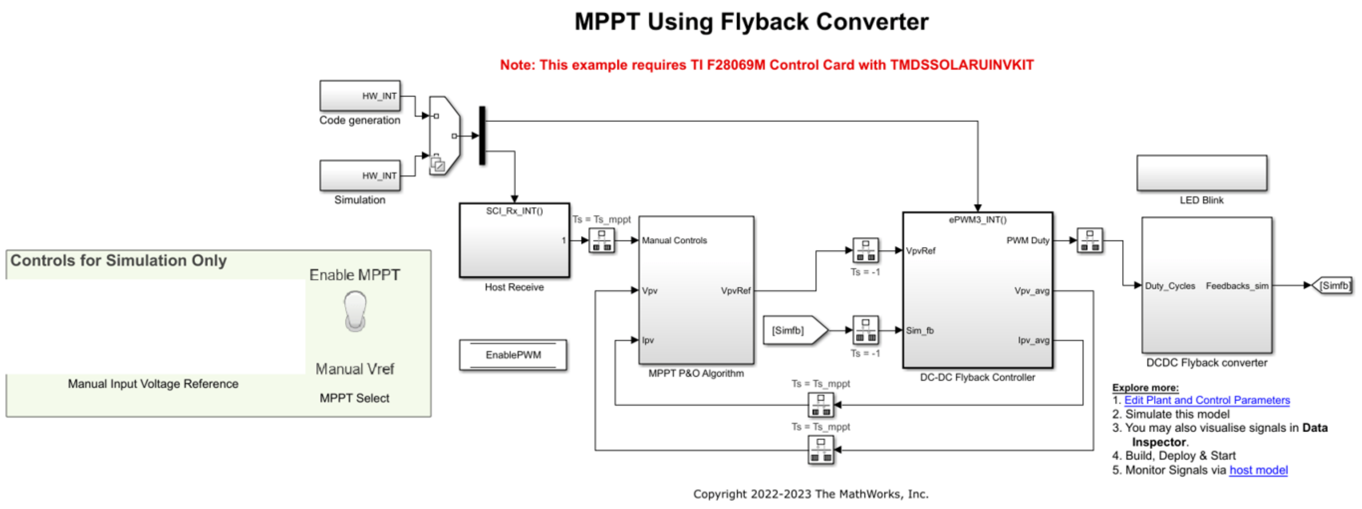
The simulation model consists of the plant model and the controllers. The plant model consists of two major components:

- Thevenin's Equivalent of PV Panel: This module uses a constant voltage source and a series resistor to emulate the PV characteristic.
- DC-DC Flyback Converter: This module simulates the flyback converter, which steps up or steps down the input voltage based on the duty cycle of the PWM pulses.

The controllers in the simulation model consists of:

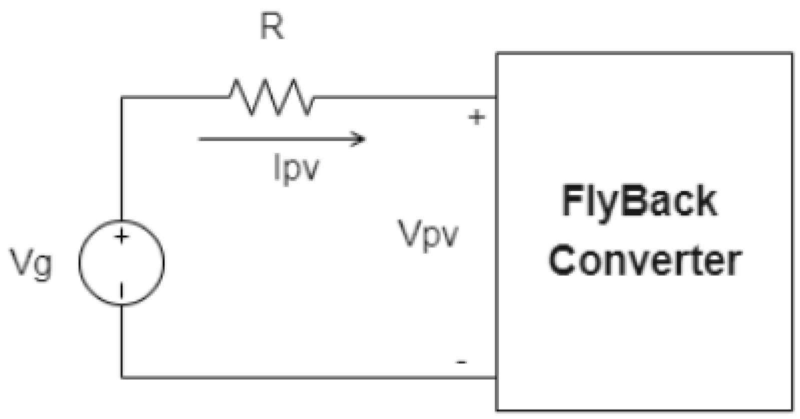
- Maximum power point tracking (MPPT) using Perturb & Observe (P&O) algorithm
- DC-DC flyback controller

```
open_system('c28069mpptsolarkit.slx');
```



Explore more:
1. Edit Plant and Control Parameters
2. Simulate this model
3. You may also visualise signals in **Data Inspector**.
4. Build, Deploy & Start
5. Monitor Signals via host model

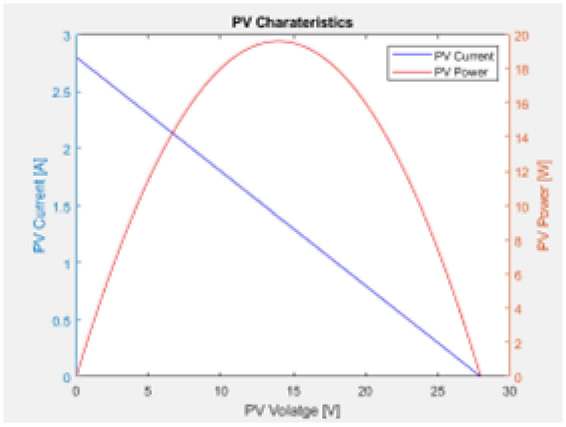
In this example, we are emulating the solar panel using the Thevenin's equivalent to provide input to the flyback converter. A fixed voltage source(28V) in series with a resistance (10 ohms) is used to create a variable voltage source depending on the current drawn from the input.



Here, Vpv=Vg-Ipv*R

PV characteristics of the equivalent solar panel:

The below PV characteristics is linear and obtained when the impedance across the input terminals of flyback converter is varied from 0(short circuit) to inf (open circuit).



The perturb & observe (P&O) algorithm is implemented using a Simulink® for calculating the reference voltage required for maximum power point operation. For the above type of characteristic, the maximum power point will occur at the midpoint of the above PV characteristic that is when input to the flyback converter is 14V. The reference voltage is achieved with the help of a voltage controller which implements a PI controller to track the reference voltage set by the MPPT algorithm. For tracking the reference voltage, the input to the flyback converter is measured. To achieve the reference voltage value, the feedback and the voltage reference inputs of the PI controller are reversed. The controller operates at a rate of 50 kHz.

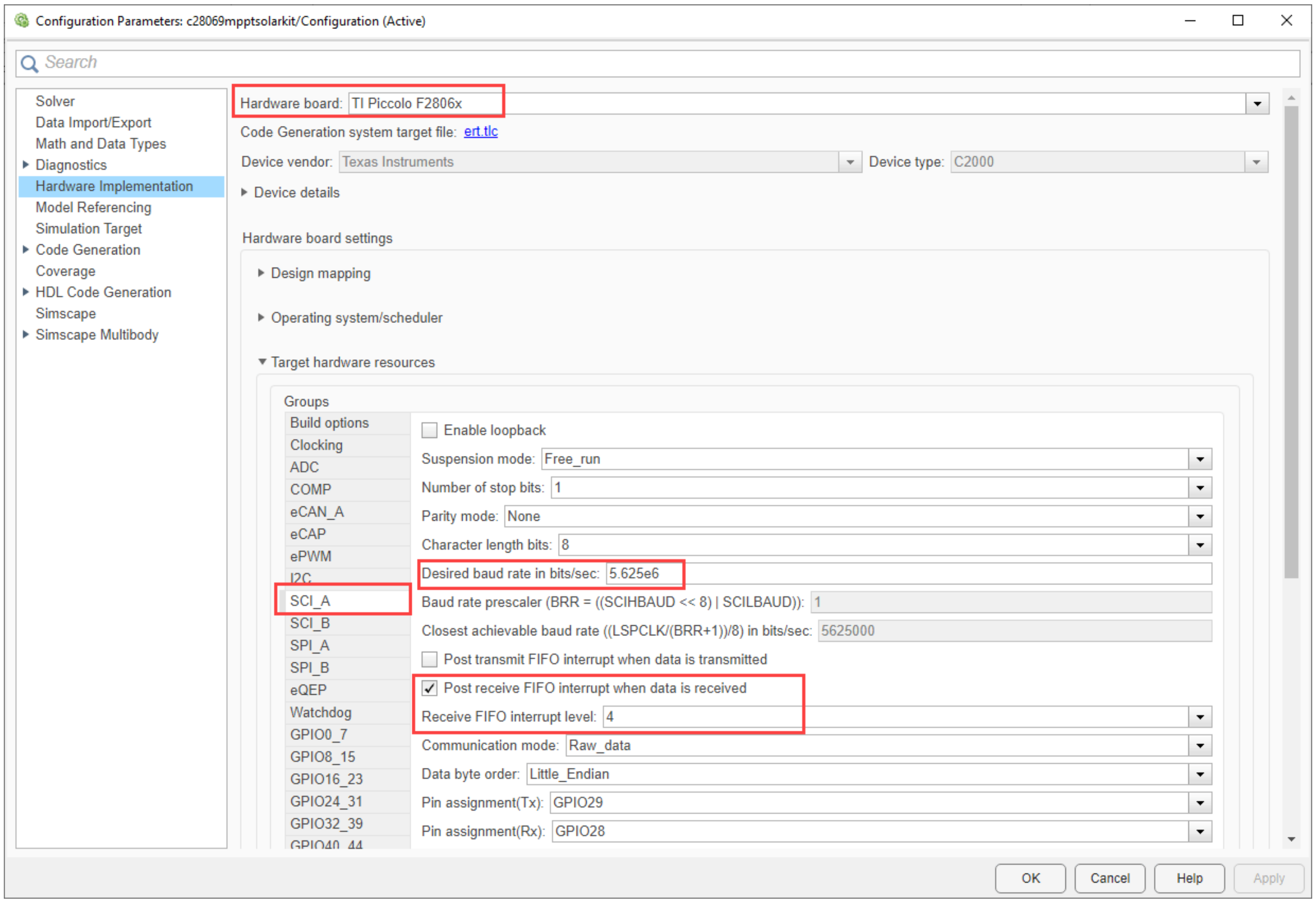
The flyback converter output voltage is not controlled. Ensure that a proper resistive load is connected to the output. In the example, a lamp of 60W has been used as a load.

Configure the Model

1. Open the model. The model is configured for TI F2806x hardware.

2. To run the model on other TI C2000 processors, press **Ctrl+E** to open the Configuration Parameters dialog box and select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

3. Ensure that SCI has desired baud rate of `5.625e6` Mbps for F28069 and `1.5Mbps` for F28035 and receive FIFO interrupt is enabled as shown.


Configuration Parameters: c28069mpptsolarkit/Configuration (Active)

Hardware board: TI Piccolo F2806x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Hardware board settings

Target hardware resources

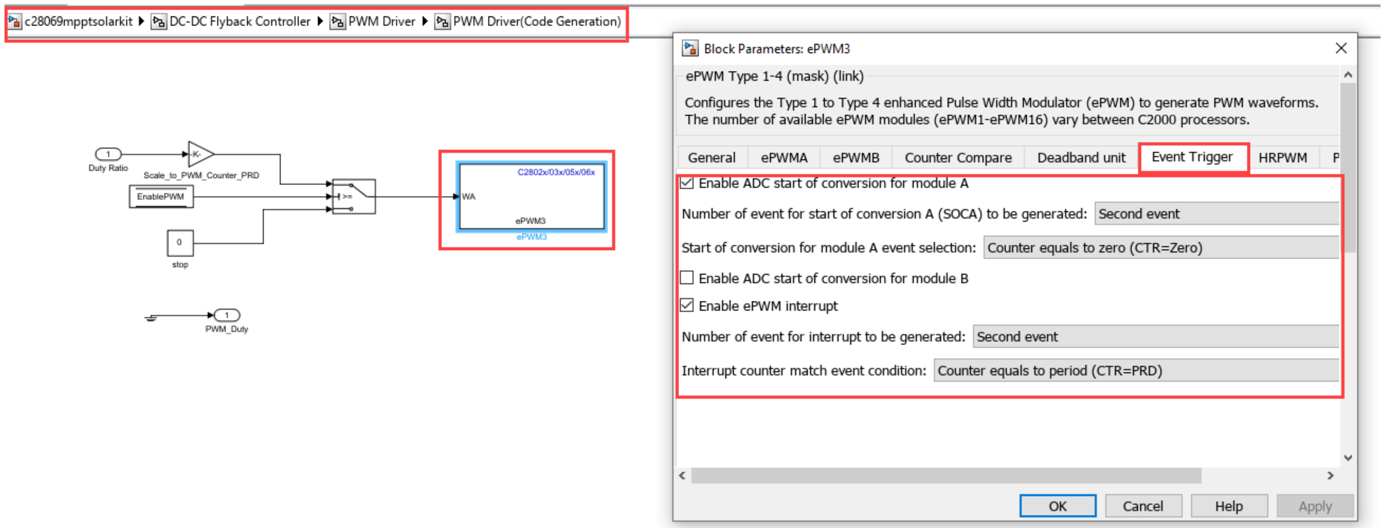
Groups: SCI_A

- [ ] Enable loopback
- Suspension mode: Free_run
- Number of stop bits: 1
- Parity mode: None
- Character length bits: 8
- Desired baud rate in bits/sec: 5.625e6
- Baud rate prescaler (BRR = ((SCIHBAUD << 8) | SCILBAUD)): 1
- Closest achievable baud rate ((LSPCLK/(BRR+1))/8) in bits/sec: 5625000
- [ ] Post transmit FIFO interrupt when data is transmitted
- [✓] Post receive FIFO interrupt when data is received
- Receive FIFO interrupt level: 4
- Communication mode: Raw_data
- Data byte order: Little_Endian
- Pin assignment(Tx): GPIO29
- Pin assignment(Rx): GPIO28

Peripheral Block Configuration

The PWM signals are generated at a frequency of 100 kHz. The ePWM3 module is configured to operate in up-down count mode. This ISR is triggered by ePWM3 on every second Time base period match event and ADC SoC is sent on every second event of ePWM3 counter equal to zero so that the ISR is triggered only after the ADC conversions are complete.

ePWM



Block Parameters: ePWM3

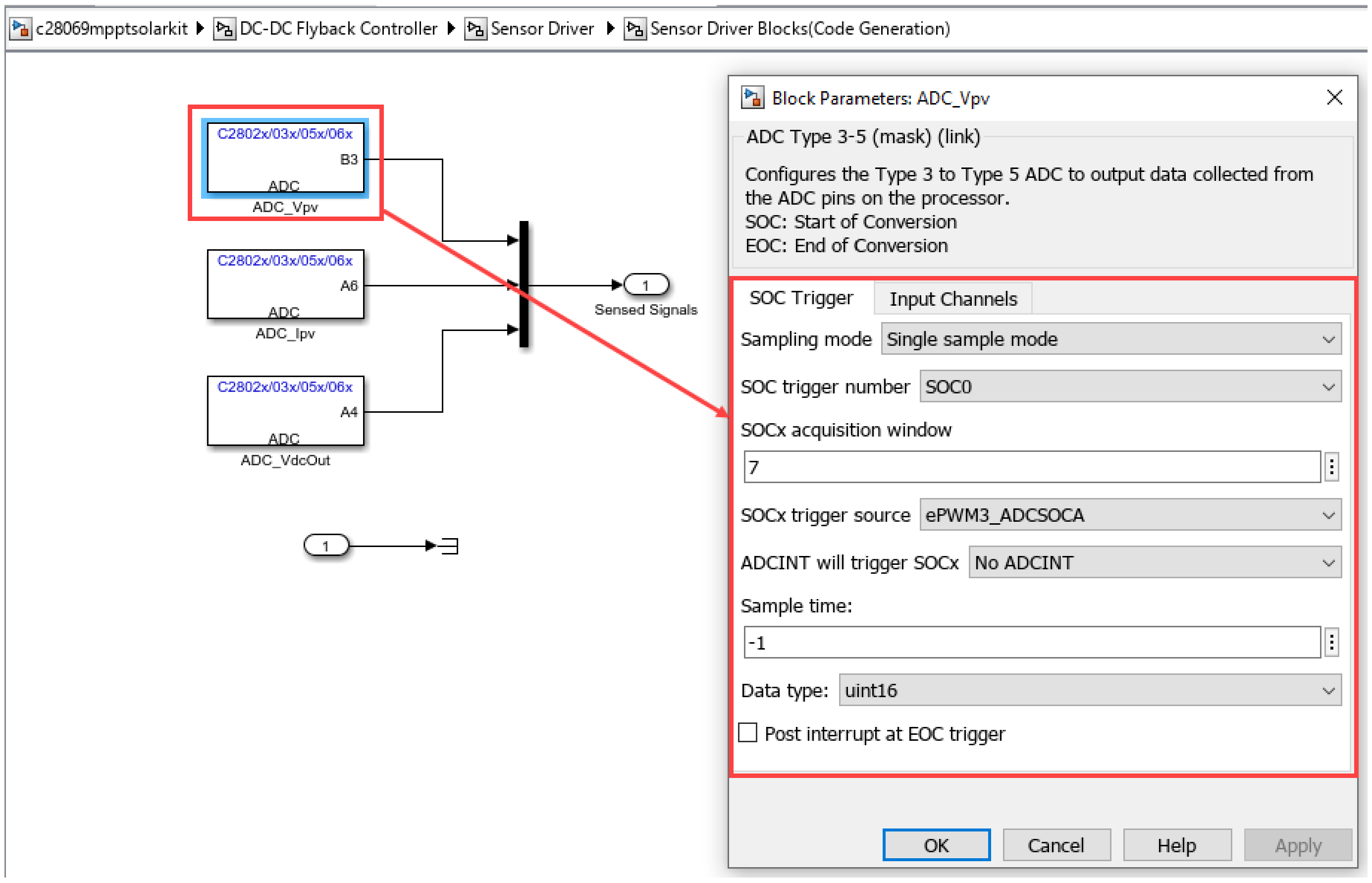
ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms. The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

- [✓] Enable ADC start of conversion for module A
- Number of event for start of conversion A (SOCA) to be generated: Second event
- Start of conversion for module A event selection: Counter equals to zero (CTR=Zero)
- [ ] Enable ADC start of conversion for module B
- [✓] Enable ePWM interrupt
- Number of event for interrupt to be generated: Second event
- Interrupt counter match event condition: Counter equals to period (CTR=PRD)

ADC

Input Voltage, input current to the flyback converter and Output Voltage are sampled when ePWM3 counter equals to zero.



Block Parameters: ADC_Vpv

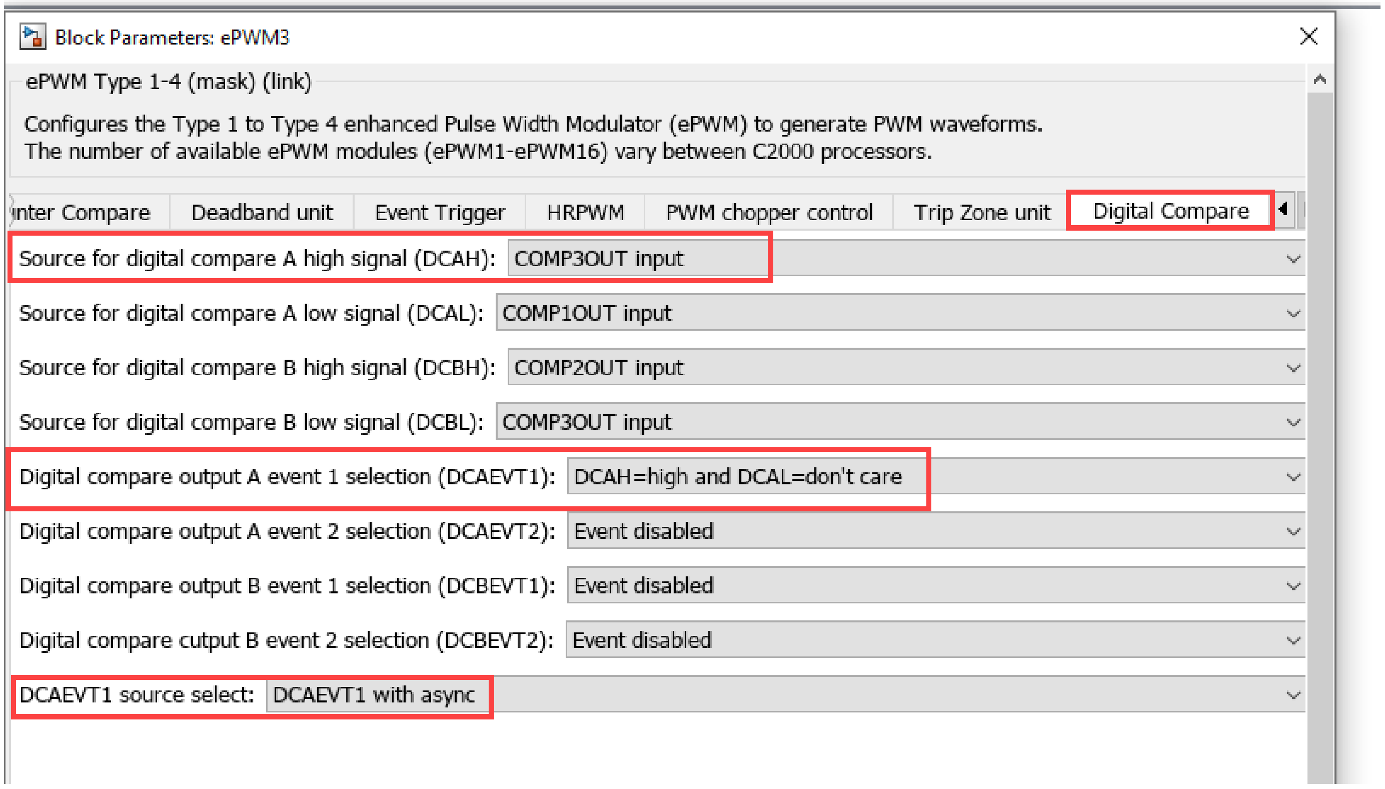
ADC Type 3-5 (mask) (link)

Configures the Type 3 to Type 5 ADC to output data collected from the ADC pins on the processor.
SOC: Start of Conversion
EOC: End of Conversion

- Sampling mode: Single sample mode
- SOC trigger number: SOC0
- SOCx acquisition window: 7
- SOCx trigger source: ePWM3_ADCSOCA
- ADCINT will trigger SOCx: No ADCINT
- Sample time: -1
- Data type: uint16
- [ ] Post interrupt at EOC trigger

Over Current Protection

The comparator3 is used to compare the switch current with a predefined current limit which drives the digital compare module A to trip the PWM.



Block Parameters: ePWM3

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

Source for digital compare A high signal (DCAH): COMP3OUT input

Source for digital compare A low signal (DCAL): COMP1OUT input

Source for digital compare B high signal (DCBH): COMP2OUT input

Source for digital compare B low signal (DCBL): COMP3OUT input

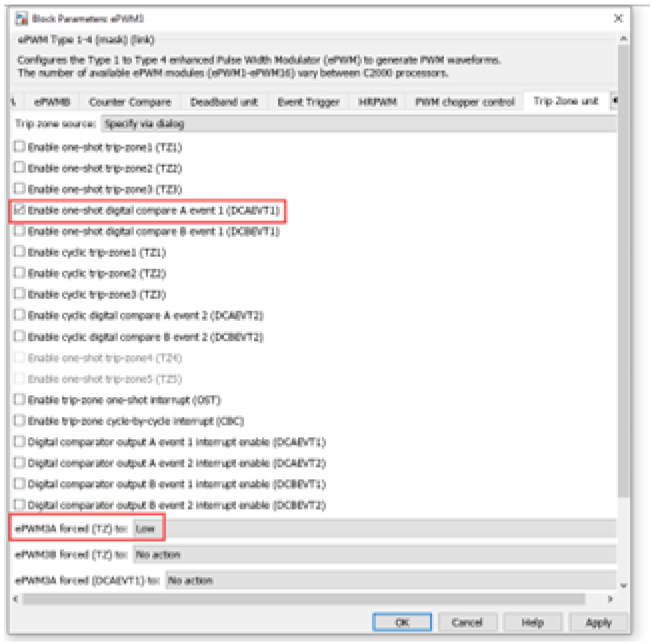
Digital compare output A event 1 selection (DCAEVT1): DCAH=high and DCAL=don't care

Digital compare output A event 2 selection (DCAEVT2): Event disabled

Digital compare output B event 1 selection (DCBEVT1): Event disabled

Digital compare cutput B event 2 selection (DCBEVT2): Event disabled

DCAEVT1 source select: DCAEVT1 with async



Block Parameters: ePWM3

ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms.
The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

Trip zone source: Specify via dialog

- ☐ Enable one-shot trip-zone1 (TZ1)
- ☐ Enable one-shot trip-zone2 (TZ2)
- ☐ Enable one-shot trip-zone3 (TZ3)
- ☑ Enable one-shot digital compare A event 1 (DCAEVT1)
- ☐ Enable one-shot digital compare B event 1 (DCBEVT1)
- ☐ Enable cyclic trip-zone1 (TZ1)
- ☐ Enable cyclic trip-zone2 (TZ2)
- ☐ Enable cyclic trip-zone3 (TZ3)
- ☐ Enable cyclic digital compare A event 2 (DCAEVT2)
- ☐ Enable cyclic digital compare B event 2 (DCBEVT2)
- ☐ Enable one-shot trip-zone4 (TZ4)
- ☐ Enable one-shot trip-zone5 (TZ5)
- ☐ Enable trip-zone one-shot interrupt (OST)
- ☐ Enable trip-zone cycle-by-cycle interrupt (CBC)
- ☐ Digital comparator output A event 1 interrupt enable (DCAEVT1)
- ☐ Digital comparator output A event 2 interrupt enable (DCAEVT2)
- ☐ Digital comparator output B event 1 interrupt enable (DCBEVT1)
- ☐ Digital comparator output B event 2 interrupt enable (DCBEVT2)

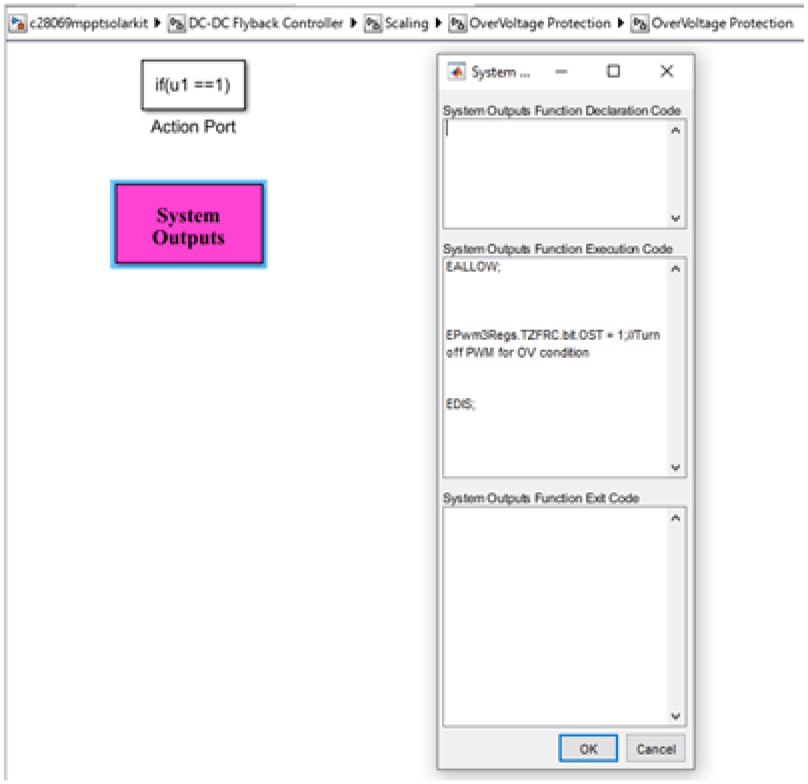
ePWM3A forced (TZ) to: Low

ePWM3B forced (TZ) to: No action

ePWM3A forced (DCAEVT1) to: No action

Over Voltage Protection

An overvoltage protection mechanism is implemented using custom code in System Outputs block. The sensed DC bus output voltage from the ADC input is compared against the overvoltage protection threshold set in the initialization script file. When overvoltage condition is met, the one-shot trip in ePWM is enabled.


c28069mpptsolarkit ▶ DC-DC Flyback Controller ▶ Scaling ▶ OverVoltage Protection ▶ OverVoltage Protection
if(u1 ==1)
Action Port
System Outputs
System ...
System Outputs Function Declaration Code
System Outputs Function Execution Code
System Outputs Function Exit Code
OK
Cancel

```
EALLOW;


EPwm3Regs.TZFRC.bit.OST = 1;//Turn
off PWM for OV condition


EDIS;
```

Simulate the Model

Run the Model

1. Open the c28069mpptsolarkit model and click the **Run** button to simulate the model.

2. Observe the output waveforms on the **Simulink Data Inspector (SDI)**.

Generate Code for Controller and Load It on controlCARD

The deployment model consists of two real-time interrupt service routines (ISR) used for:

- Closed-loop control of the DC-DC Flyback converter (50 kHz)
- Gets users commands such as Input Voltage reference in Manual Voltage mode and MPPT enable command

Load Model on controlCARD

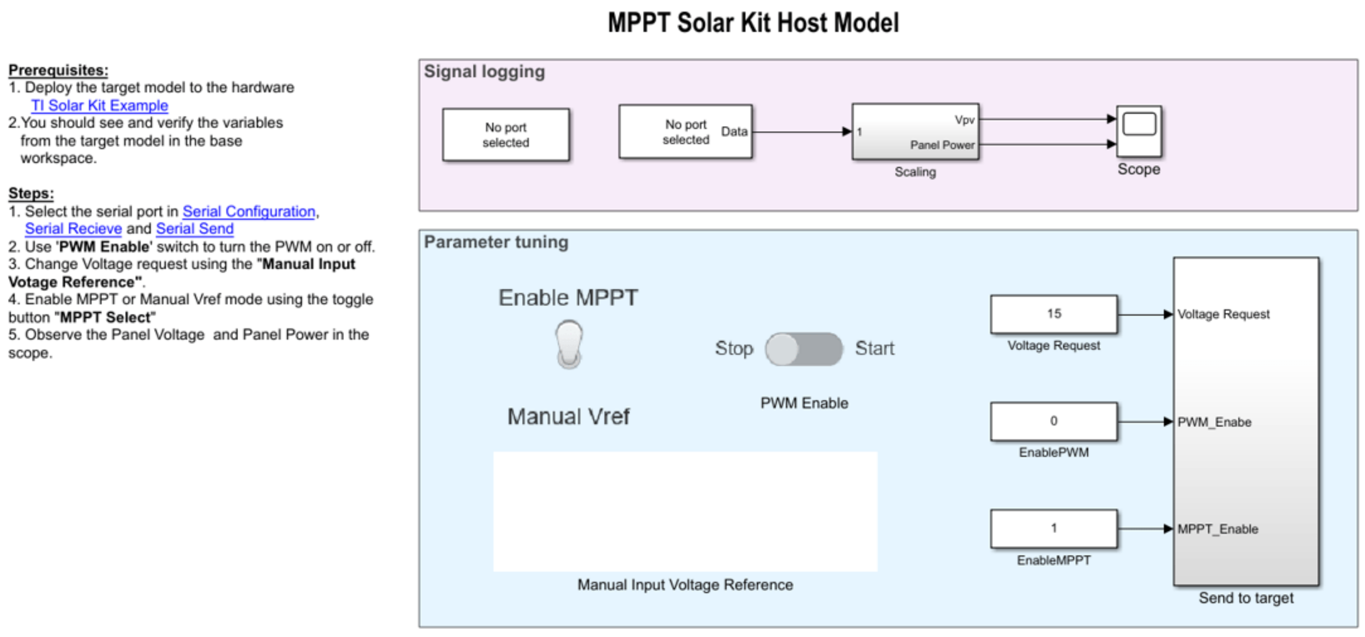
- **1.** Ensure the hardware connections are made as mentioned in the Hardware Connections section.
- **2.** Open the c28069mpptsolarkit model and generate code by pressing **Ctrl+B**.
- **3.** Follow the build process by opening the diagnostic viewer using the link at the bottom of the model canvas.

- **4.** Connect an appropriate resistive load to the flyback output terminals (TP8 & TP10). Currently, the model has been tested with a 60W bulb connected as load.
- **5.** Turn on the DC power supply and ensure it is 28V.
- **6.** Run the host model and observe the input voltage and input power of the flyback converter.

Monitor Signals and Tune Parameters Using the Host Computer

The host model receives the data from the kit and plots it to verify the performance of the MPPT and the control algorithms.

```
open_system('c2000_solarkit_host_model.slx');
```



Prerequisites:
1. Deploy the target model to the hardware
   TI Solar Kit Example
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select the serial port in Serial Configuration, Serial Recieve and Serial Send
2. Use '**PWM Enable**' switch to turn the PWM on or off.
3. Change Voltage request using the "**Manual Input Votage Reference**".
4. Enable MPPT or Manual Vref mode using the toggle button "**MPPT Select**"
5. Observe the Panel Voltage and Panel Power in the scope.

Configure and Run Model on Host Computer

- **1.** On the host computer, browse to **Device Manager > Ports (COM & LPT)** to find the COM port.
- **2.** Set the parameter Port of the following blocks in the c2000_solarkit_host_model to match the COM port of the host computer:

- **c2000_solarkit_host_model > Serial Configuration**
- **c2000_solarkit_host_model > Serial Receive**
- **c2000_solarkit_host_model > Send to target > Serial Send**

- **3.** Click the **Run** button on the Simulation tab to run the host model.
- **4.** While the model runs, you can monitor the input voltage and input power to the flyback converter to analyze and monitor the performance of the MPPT algorithm.

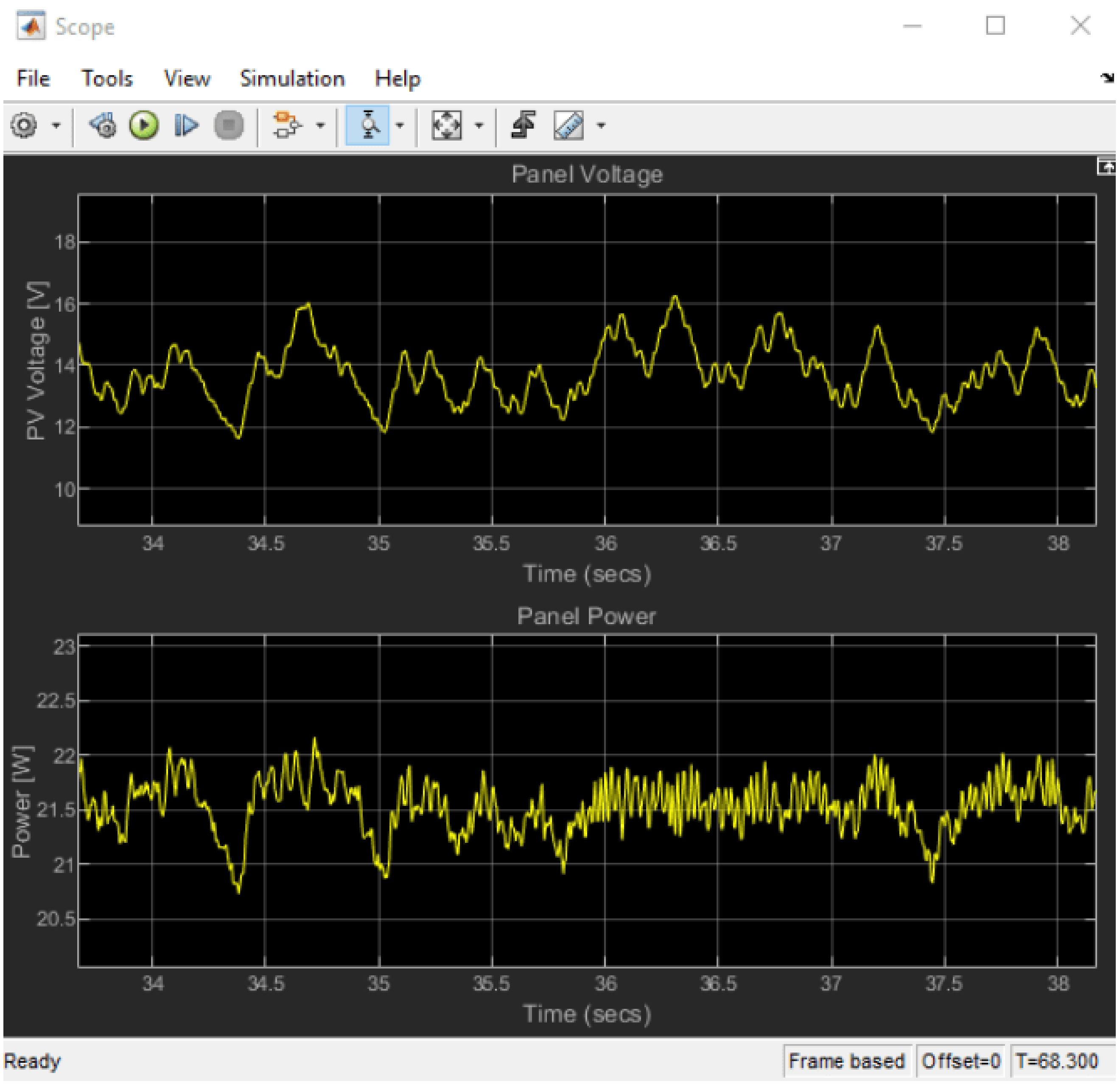
Tune the Parameters

While the model runs, you can tune parameters using the dashboard blocks:

- Manual Input Voltage Reference - The value used to manually set the operating point of the PV emulator. This value is used when the toggle switch is set to the Manual Vref option. Toggling the switch turns off the MPPT algorithm and lets you choose the operating voltage of the PV emulator.
- PWM Enable- This enables the PWM to the switch in flyback converter.

Hardware Results

The hardware results show that the input power is controlled at around 21 watts and input voltage to the fly back converter is controlled at around the MPPT voltage of 14V.



More About

“Photovoltaic Inverter with MPPT Using Solar Explorer Kit” on page 4-216

DC-DC Buck Converter Using MCU

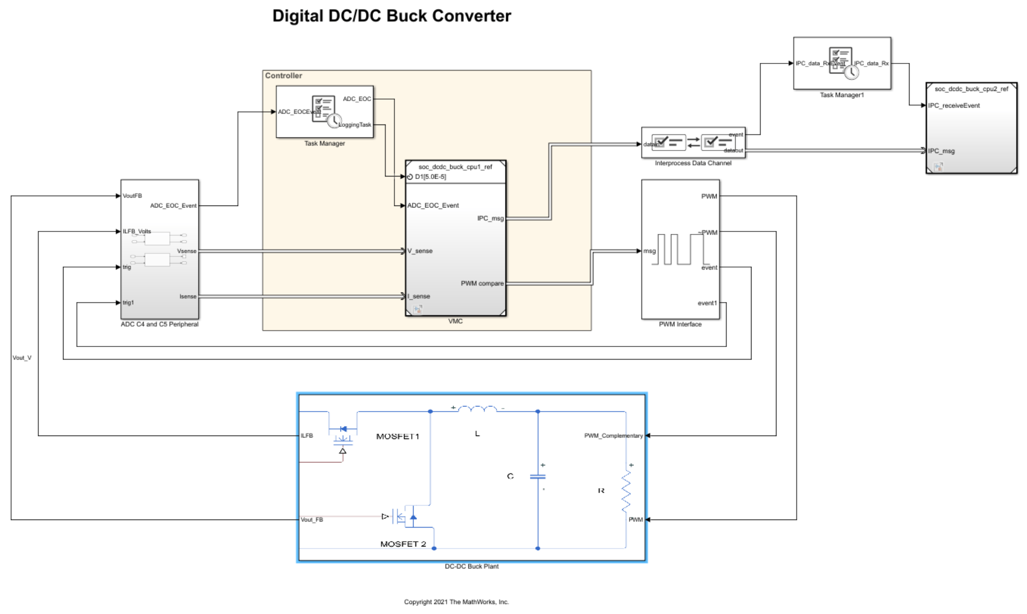
This example shows how to develop a DC-DC buck converter power regulator application.Typical challenges with power conversion simulation and deployment include:

- Modeling the analog circuit behavior of the buck converter circuit
- Modeling the timing behavior of PWM output and ADC sampling on an MCU
- Capturing signals in high CPU load controllers
- The amount of time required for controller validation, which is typically performed on hardware

These challenges are addressed in this example using SoC Blockset™ and Simscape™. Digital control type used in this example is voltage mode controller (VMC), verified on the TI Delfino F28379D LaunchPad and TI BOOSTXL-BUCKCONV kit.

This model shows the complete converter system, and the sections in this example will examine the individual challenges. To open this model, run the following code.

```
open_system('soc_dcdc_buck')
```

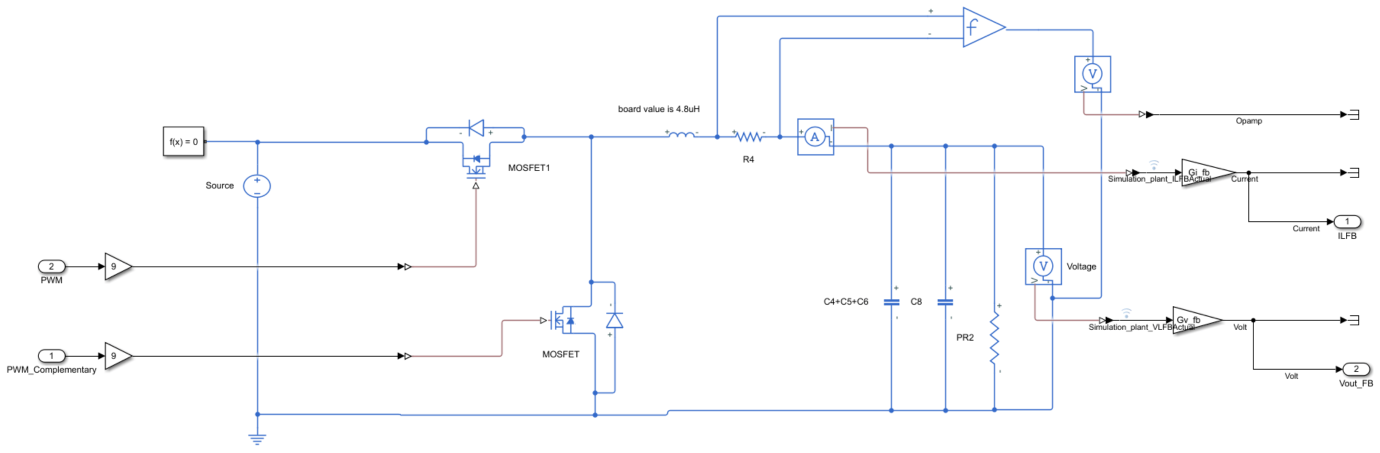


Required Hardware

- TI Delfino F28379D LaunchPad
- TI BOOSTXL-BUCKCONV kit

Model of DC-DC Buck Converter Kit

`DC-DC Buck Plant` subsystem is a Simscape reference model of the DC-DC buck converter analog circuitry.



The Simscape™ blocks in the model are selected and configured based on the original equipment manufacturer (OEM) specifications provided in the data sheet. To achieve computational efficiency in simulation without affect on behavior, the model includes these simplification relative to OEM specifications:

- Voltage and current sensing circuits are simplified to gain blocks.
- MOSFETS are simplified to ideal MOSFETS.
- Gate driver is not modeled and its propogation delays are not considered.
- Inductor is simplified to linear inductor.
- All parts are modeled with nominal values, and tolerances are not considered.
- DC supply is assumed to be constant.

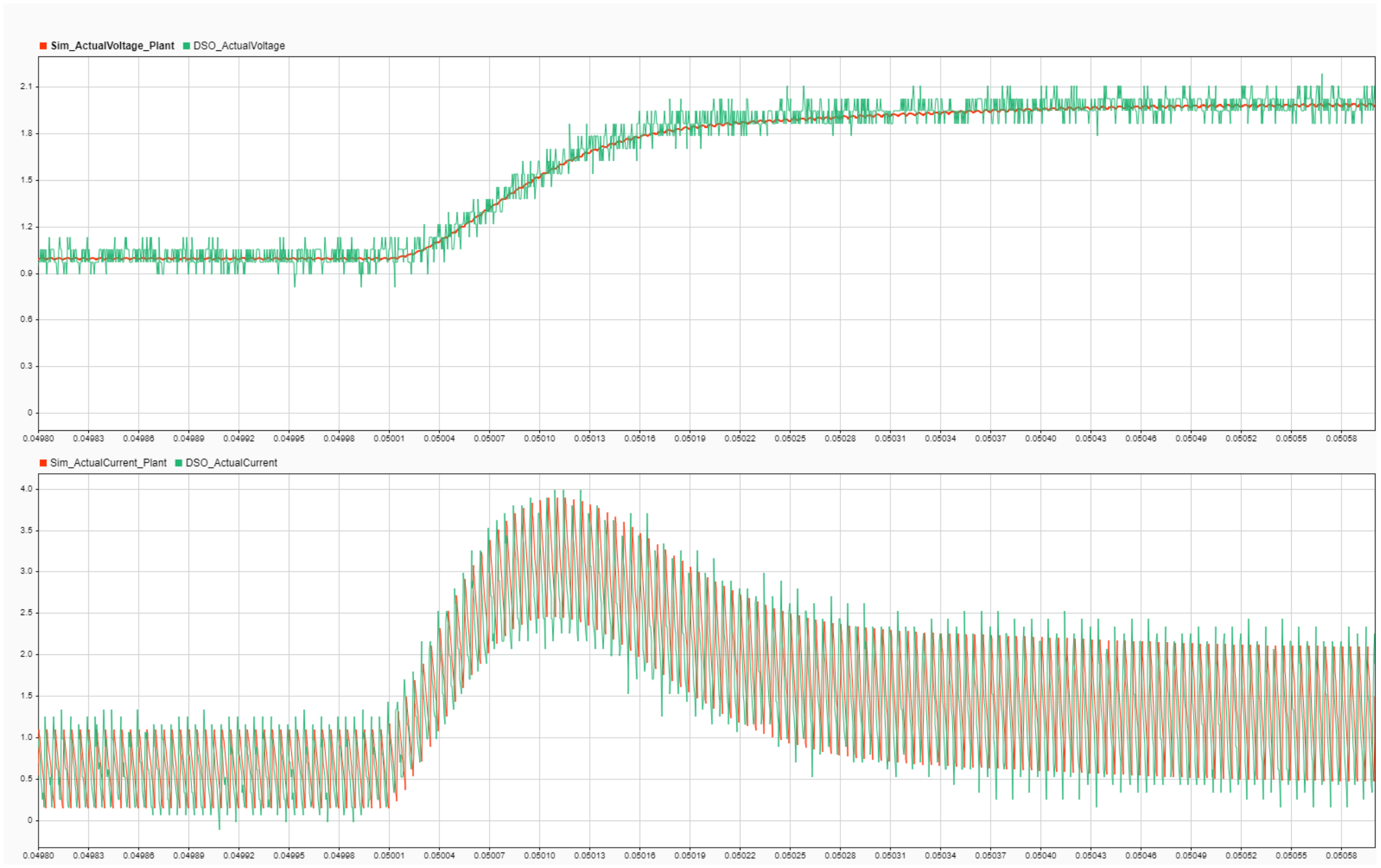
If needed, the open loop response of the Simscape model can be compared and verified against the physical hardware using a digital oscilloscope with results captured using the Data Acquistion Toolbox.

Voltage Mode Control on MCU

On the MCU, the output of the plant sampled by the ADC Interface (SoC Blockset) generaring an event on each end of conversion. The Task Manager (SoC Blockset) executes an event-driven task called ADC upon reception of each ADC end-of-conversion event. The ADC Interrupt task contains the feedback control algorithm that executes asynchronously in response to each ADC conversion event. The control algorithm receives feedback through ADC Read (SoC Blockset) and generates duty cycle values for PWM Write (SoC Blockset) block. The PWM Interface (SoC Blockset) block simulates PWM behavior including triggering an event to start the next ADC conversion. PWM frequency is set to 200 kHz. The discrete proportional integral (PI) controller minimizes the error between the reference voltage and the output voltage. The duty cycle of the PI controller is limited to 40% of the PWM time period.

The system starts with an initital voltage reference of 1 volt and allowed to reach steady state. This enables a fair control between the phsyical hardware and simulation to compare with a known state. The desired voltage step of 2 volts is then triggered at 50 ms to examine the step response of the closed loop controller. Click **Play** to simulate the model. Open the **Simulation Data Inspector** and view signals.

To verify simulation results against hardware, deploy the model to the TI Delfino F28379D LaunchPad. On the **System on Chip** tab, click **Configure, Build, & Deploy** to open the SoC Builder (SoC Blockset) tool. This figure shows the comparison of the controller response between the simulation and deployed model on the physical hardware. This signal on the hardware is captured using a digital oscilloscope. The high frequency operation of controller prevents the direct use of external mode on the same CPU. For this reason a digital oscilloscope is used to take these measurements.

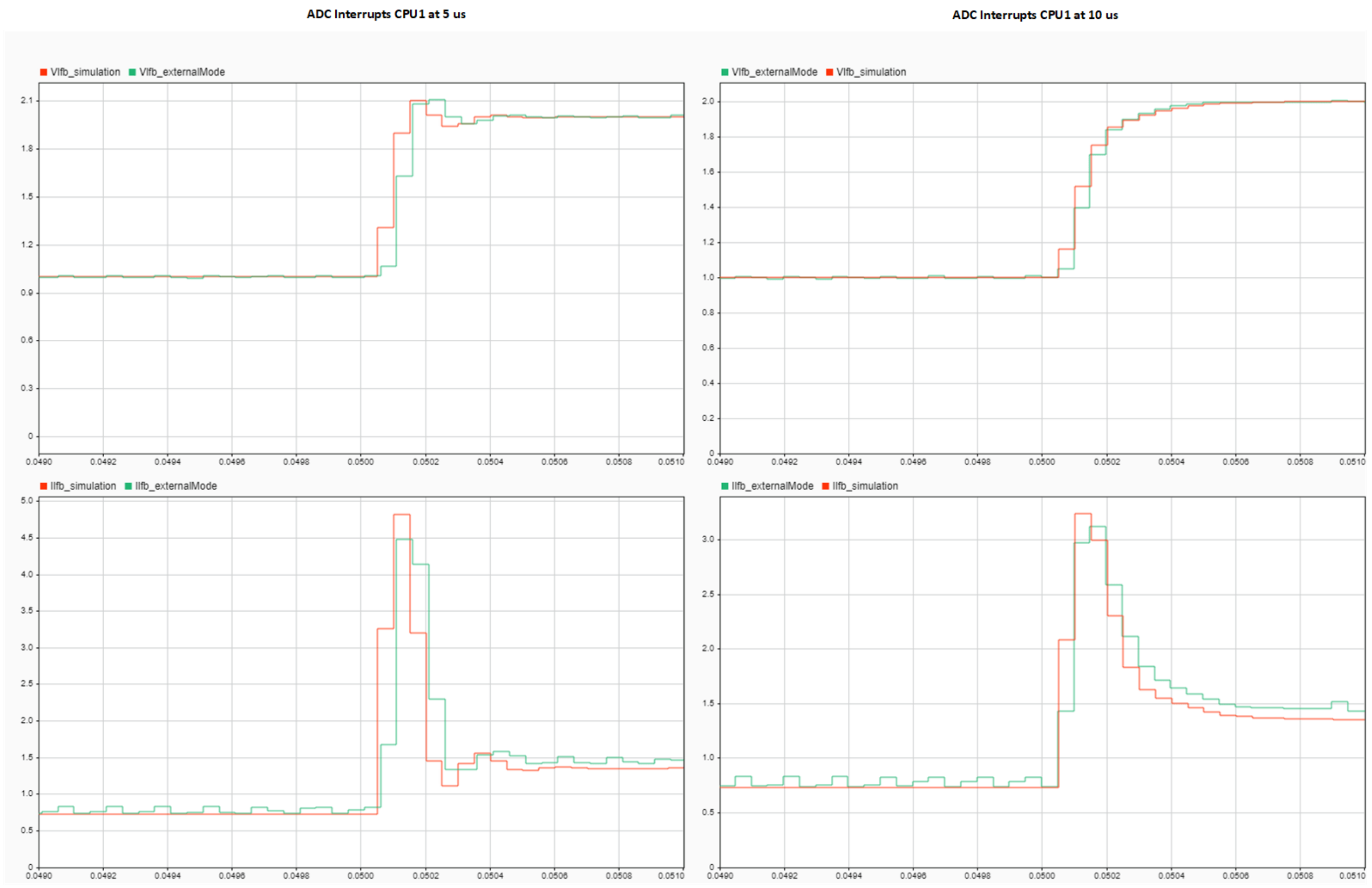


As expected, the voltage mode controller correctly tracks the desired the voltage output. Additionally the measurements from the deployed model match the simulation with greater than 95% accuracy for this type of system. The minor differences seen between the simulation and the deployed measurements can be attributed to the simplifications made in the Simscape model.

Taking Advantage of Multicore to Log Data in CPU2

CPU2 is configured to run external mode **SoC Builder** tool, to log and transmit the high frequency signals produced by control loop on CPU1. An Interprocess Data Channel (SoC Blockset) block connects CPU1 and CPU2, providing a low latency data transfer between the CPUs.

Use the SoC Builder (SoC Blockset) tool to deploy the model to the TI Delfino F28379D LaunchPad. A host-target communication connection, set up by the **SoC Builder** tool, logs the signal data from the executable running on CPU2 of the hardware board and sends the data to the **Simulation Data Inspector** in Simulink. Using CPU2 to own and manage the host-target communication and data logging, data can be captured from the resource intensive, high-priority task on CPU1 without interfering with its behavior and enabling that task to consume most of the CPU resources, and with maintaining the quality of data logging to Simulink. This figure shows the logged data signal from task 1 on CPU1, captured on task 2 on CPU2, of the model deployed to a TI Delfino F28379D LaunchPad.



ADC start of conversion trigger can be configured to generate at 1st PWM or 2nd PWM event. These settings are available in simulation and codegen. Observe simulation and codegen results match with greater than 95% accuracy.

Any resource intensive SoC Blockset model could use this setup to log data from hardware when the model is deployed to a TI Delfino F28379D LaunchPad. For more information on data logging techniques, see “Data Logging Techniques” (SoC Blockset).

Further Exploration

- Extend for high frequency switching applications involving Gallium Nitride (GaN) or Silicon Carbide (SiC)
- Variable PWM frequency and fixed duty cycle
- Variable phase offset
- Different PWM output schemes by using the PWM output control options
- Different PWM event generation techniques

See Also

- “Get Started with Multiprocessor Blocks on MCUs” (SoC Blockset)
- “Partition Motor Control for Multiprocessor MCUs” (SoC Blockset)
- “Data Logging Techniques” (SoC Blockset)

MAT-file Logging on SD Card for Texas Instruments C2000 Processors

This example shows you how to perform MAT-file logging using Simulink® model on a Micro SD card mounted on Texas Instruments™ C2000™ Microcontroller Blockset.

Introduction

C2000™ Microcontroller Blockset supports logging of signals from Simulink® model. These signals are logged as MAT-file(s) on a Micro SD card mounted on Texas Instruments™ C2000™ Processors.

Signal logging enables you to monitor the signal behavior and to perform any historical analysis of the data. The data can be saved in any of these formats: Structure, Structure with time, or Array. You can save the signals using the following blocks:

- To Workspace
- Scope
- Output port

This example provides you with the workflow to enable MAT-file logging and obtain the MAT-file on a Micro SD card mounted on Texas Instruments™ C2000™ Processors.

Required Hardware

To run this example, you need the following hardware:

- Texas Instruments™ C2000™ Processors
- External SD card interface(optional)
- Micro SD card

Hardware Connection(For External SD Card Interface)

- For the external SD card interface, based on the **SPI** module selected, the SIMO, SOMI, CLK and STE pin has to connected to corresponding GPIO pins of C2000 processor.
- Connect VCC and GND pins to appropriate supply and ground.

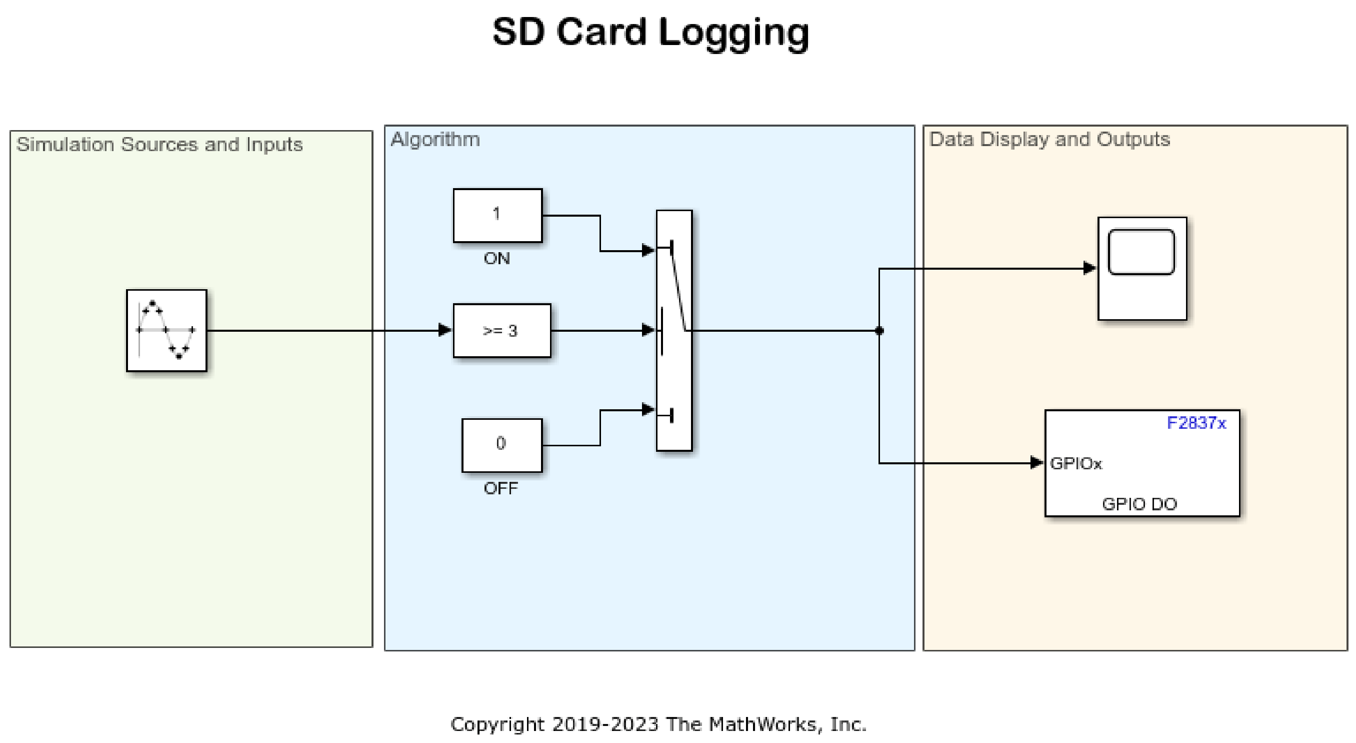
Step 1 - Prepare the Connection and Configure a Simulink Model for MAT-File Logging

The MAT-file logging needs to be configured on the Simulink® model so that the selected signals are logged on the Micro SD card on Texas Instruments™ C2000™ Processors.

1. Insert the Micro SD card in the slot provided on the board.

2. Open the SD Card Logging model. This Simulink model is pre-configured for the **TI Delfino F2837xD** with MAT-File logging enabled. In this example, signals from the Scope block are logged.

```
open_system('c28x_matfile_logging.slx');
```



3. In your model window, open the **Configuration Parameters** dialog box, go to the **Hardware Implementation** pane, and select the name of the target hardware from the **Hardware board** list. If required, you can change the Hardware board selection other than pre-configured F2837xD.

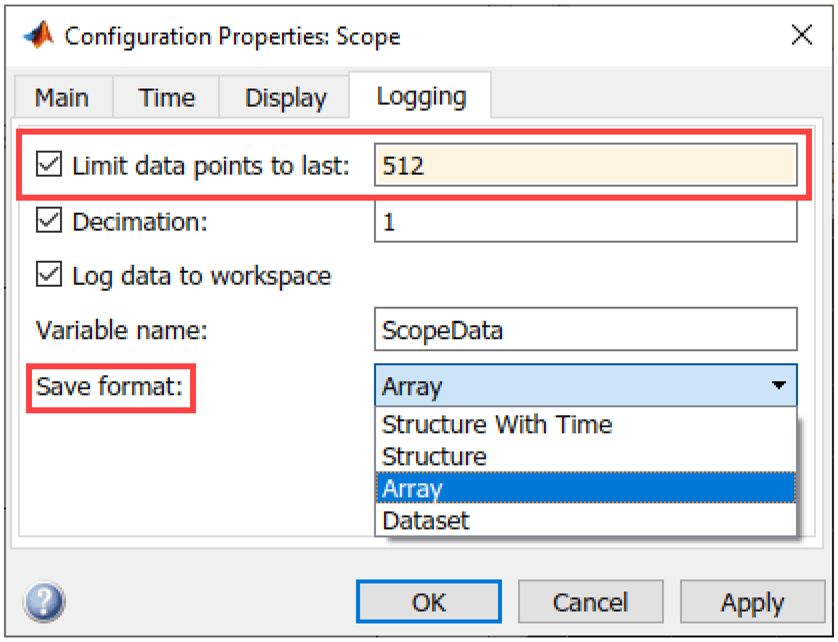
4. To enable MAT-file logging, go to **Target hardware resources > SD card logging**. Select the **Enable MAT-file logging on SD card** checkbox. Also select the desired **SPI module** and **SPI baud rate** and ensure in the SPI_x pane that all configurations are as required.

5. Click **Apply** and then **OK**.

6. Double-click the Scope block in the Simulink model. In the Scope dialog box, click the gear icon (Configuration Properties).

7. In the **Configuration Properties: Scope** dialog box, open the **Logging** tab, and select **Log data to Workspace**. Also select **Limit data points to last** and set a value of `512`.

8. Select the **Save format** as `Array/Structure with Time/Structure` and click **Apply** and then **OK**.



☑ Limit data points to last: 512

☑ Decimation: 1

☑ Log data to workspace

Variable name: ScopeData

Save format: Array

Structure With Time
Structure
Array
Dataset

9. On the **Modeling** tab, enter a value for the `stop time`. This is the duration for which the signal is logged. However, the model continues to run on the hardware and it is not affected by the time specified.

For example, in this demo model (SD Card Logging), the `stop time` is 10. The signal will be logged for 10 seconds. If the `stop time` is Inf, the signal will be logged till the Micro SD card is full or the board is disconnected.

Step 2 - Deploy the Model on Texas Instruments C2000 Processors and View the MAT-files

After the Simulink model is configured for MAT-file logging, you can deploy the model on the connected Texas Instruments C2000 Processors and view the MAT-file(s) created on the Micro SD card.

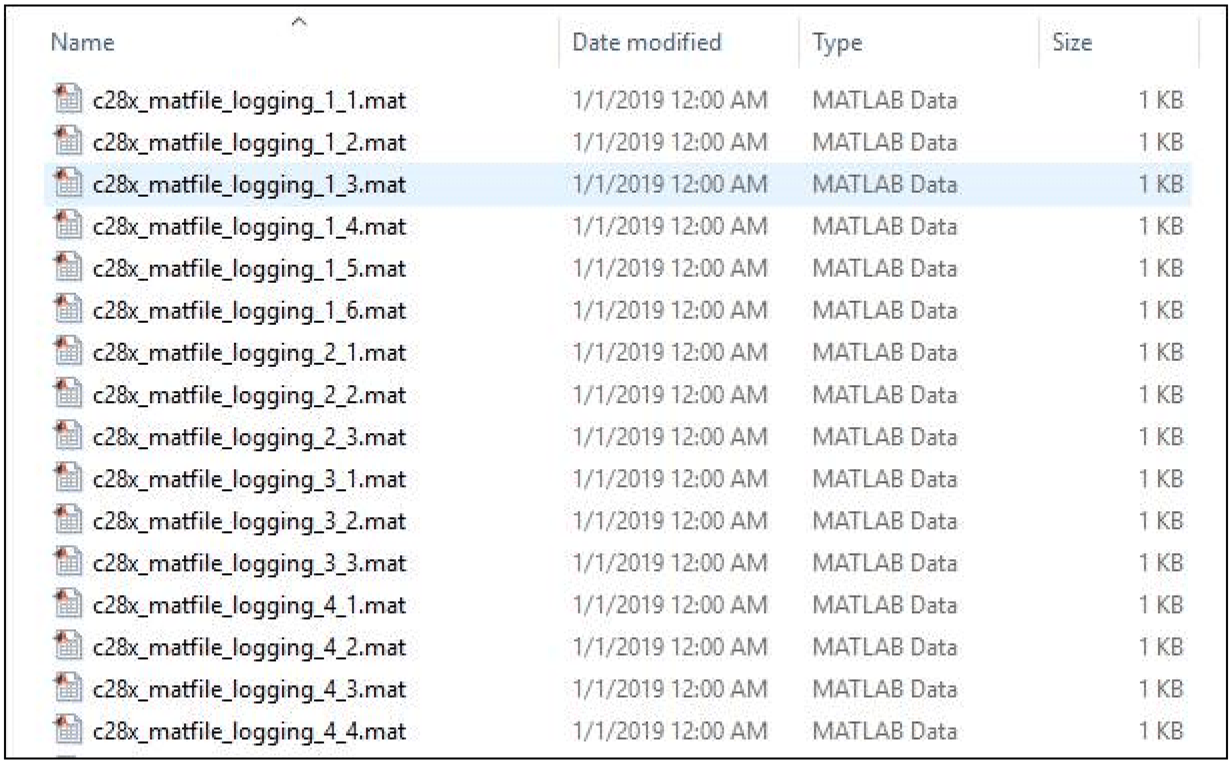
Tip: To learn more about logging data, see “Configure Model to Log Signals on SD Card” on page 2-7.

1. In the Simulink model, go to **Hardware** tab and click **Build, Deploy & Start** button. The build process for the model starts and it is deployed on the Texas Instruments C2000 Processors board. On successful deployment of the model, the LED on the board starts blinking.

Note: For processor other than F2837xD, you have to replace the **Digital Output** block and select the appropriate GPIO pin for LED blinking.

2. To view the logged MAT-files, remove the Micro SD card from the board after the Simulation time, and insert the card on your computer.

If the deployment of the model was successful, the MAT-files are generated on the Micro SD card as shown below:



Name	Date modified	Type	Size
c28x_matfile_logging_1_1.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_1_2.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_1_3.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_1_4.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_1_5.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_1_6.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_2_1.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_2_2.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_2_3.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_3_1.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_3_2.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_3_3.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_4_1.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_4_2.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_4_3.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB
c28x_matfile_logging_4_4.mat	1/1/2019 12:00 AM	MATLAB Data	1 KB

After importing the MAT-files to your computer, you can use it for further analysis in MATLAB®.

Because the model running on the hardware creates multiple MAT-files, the logged data points are distributed across the generated MAT-files. You can create a stitcher function to combine these MAT-files into a single MAT-file that contains all the logged data points. To view an example of a MAT-file stitcher and understand its usage, enter the following command in the MATLAB® command window.

```
edit('c28x_MAT_stitcher.m');
```

Other things to try with MAT-file Logging on SD Card

In this example, a Scope block is used to log signals. However, the signals in a Simulink® model can also be logged on the SD card using the `To Workspace` block or by logging the output(s) of the Simulink® model. Explore the MAT-file logging of a Simulink model using these two techniques.

Related Topics

"Configure Model to Log Signals on SD Card" on page 2-7

Using the Control Law Accelerator (CLA)

This example shows how to use the Control Law Accelerator (CLA) available on some of the TI® processors using C2000™ Microcontroller Blockset.

Introduction

C2000™ Microcontroller Blockset enables you:

- Using the CLA with an LED Blinking Example
- Data Integrity Between CPU and CLA
- Signal Monitoring of CLA Output
- Permanent Magnet Synchronous Motor Field-Oriented Control (FOC) Using CLA

Prerequisites

Complete the following tutorials:

- “Overview of CLA Configuration for C2000 Processors Using Subsystem” on page 1-93

Required Hardware

For the LED blinking model:

- TI® F28069, F28035, F28004x, F28379D or F28377S board.

For Signal Monitoring of CLA Output:

- TI F28379D LaunchPad

For the motor control application model:

- TI® DRV8312 Three-Phase Brushless Motor Control Kit (DRV8312-C2-KIT or DRV8312-69M-KIT) with F28035 or F28069 Piccolo processor
- Three-phase PMSM with Hall sensors attached to connector J10 of the DRV8312EVM board

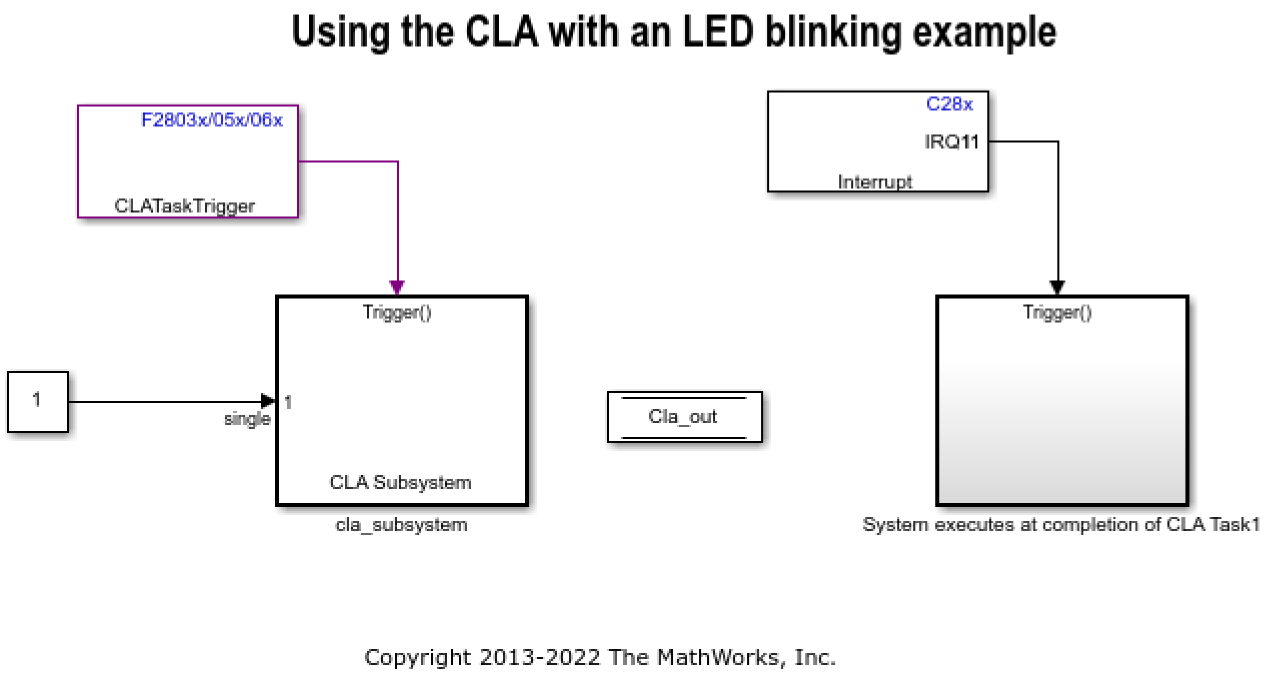
Available versions of example models:

- TI F28379D LaunchPad: c28379D_cmpss_cla_blink.slx
- F28035: c28035blink_cla.slx
- F28069: c28069blink_cla.slx
- F28377S: c28377Sblink_cla.slx
- F28004x: c28004xblink_cla.slx
- F28069: c28069_dataintegrity_cla.slx
- F28379D: c28379D_dataintegrity_cla.slx
- F28379D Launchpad: c28379D_cpu1_blink_cla.slx, c28379D_cpu2_blink_cla.slx
- F28379D: c28379D_custom_code_cla
- DRV8312 + F28069: c28069pmsmfoc_cla.slx
- DRV8312 + F28035: c28035pmsmfoc_cla.slx

Task 1: Using the CLA with an LED Blinking Example

The following figure shows an example model configured to use the CLA available on the hardware.

```
open_system('c28069blink_cla');
```


Using the CLA with an LED blinking example
F2803x/05x/06x
CLATaskTrigger
C28x
IRQ11
Interrupt
Trigger()
1
1
single
CLA Subsystem
cla_subsystem
Cla_out
Trigger()
System executes at completion of CLA Task1
Copyright 2013-2022 The MathWorks, Inc.

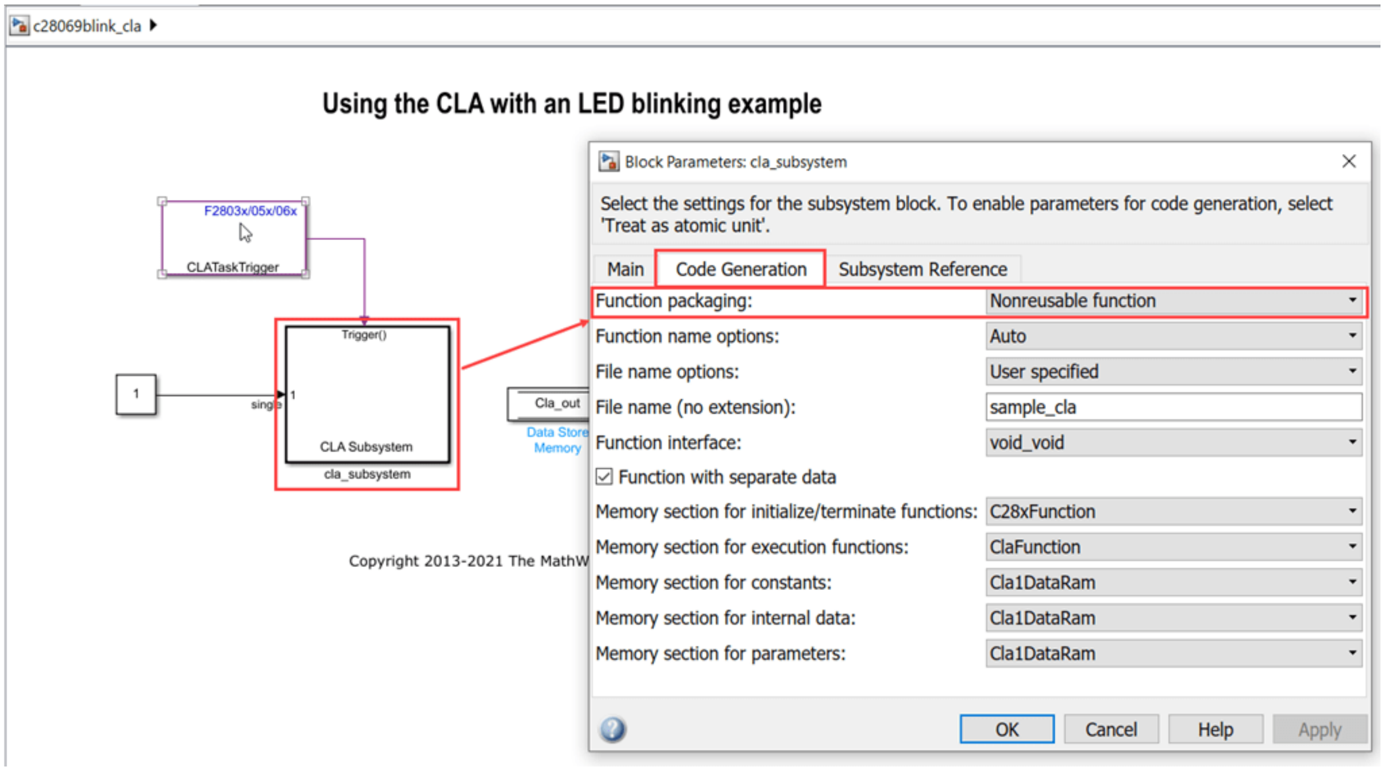
This model generates code for a Simulink® model where one part of the algorithm runs on the Control Law Accelerator (CLA) available on the device. The CLA is a co-processor that allows parallel processing. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently.

GPIO pin 34 (connected to the LED on control cards; for LaunchPad boards the GPIO pin number is different) toggles on every occurrence of the CLA Task1 interrupt.

Method 1 - Using Nonreusable Function Code Generation for CLA Subsystem to Blink an LED

In this method, `cla_subsystem` is triggered by a software at a rate of 0.5sec and is configured with **Nonreusable function** as `Function packaging`. For more, “Method 1 - Nonreusable Function Code Generation for CLA Subsystem (Recommended)” on page 1-95.

Following are the CLA Subsystem block configurations done for this model. To open the block Parameters,right-click on the CLA subsystem and select Block Parameters(Subsystem). Ensure the specified parameter values are the same if you want to run this example for other hardware board.

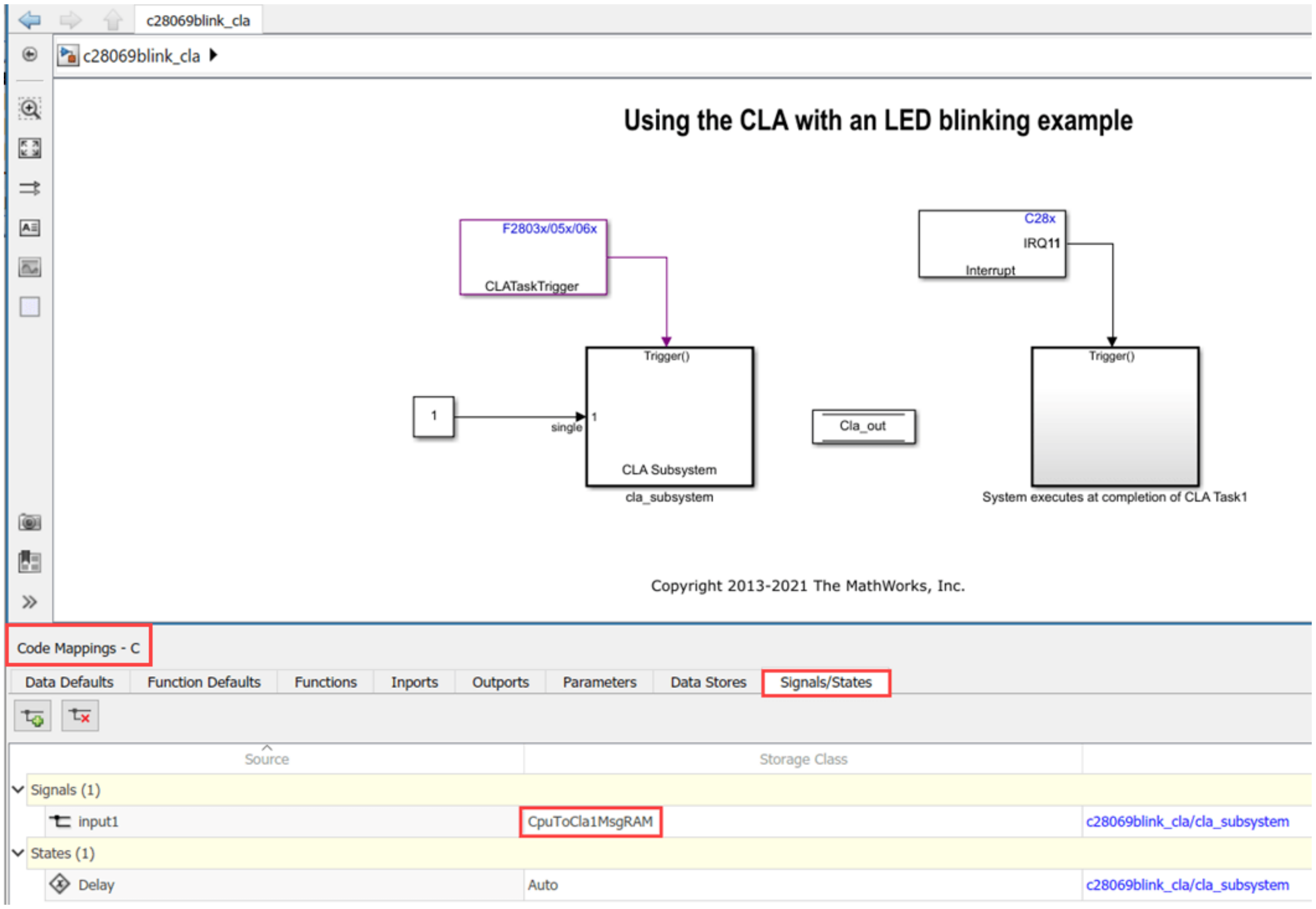


Block Parameters: cla_subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Main | Code Generation | Subsystem Reference

Function packaging:	Nonreusable function
Function name options:	Auto
File name options:	User specified
File name (no extension):	sample_cla
Function interface:	void_void
☑ Function with separate data	
Memory section for initialize/terminate functions:	C28xFunction
Memory section for execution functions:	ClaFunction
Memory section for constants:	Cla1DataRam
Memory section for internal data:	Cla1DataRam
Memory section for parameters:	Cla1DataRam



Source	Storage Class	
Signals (1)		
input1	CpuToCla1MsgRAM	c28069blink_cla/cla_subsystem
States (1)		
Delay	Auto	c28069blink_cla/cla_subsystem

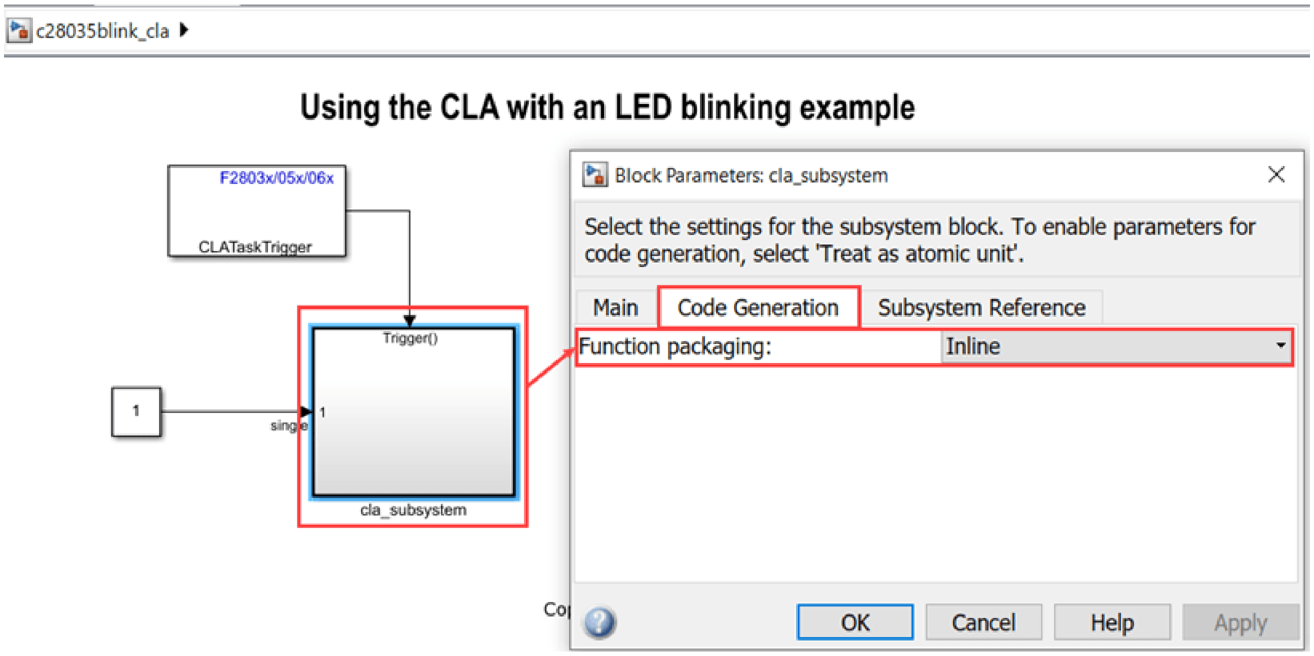
Run the Model

1. Open any one of the example model, c28069blink_cla.slx, c28004xblink_cla.slx, c28377Sblink_cla.slx, c28379D_cpu1_blink_cla.slx, or c28379D_cpu2_blink_cla.slx.
2. Open **Configuration Parameters** and select the required tool chain on the **Code Generation** pane.
3. Press **Ctrl+B** to build a binary executable and to automatically load and run the executable on the selected target.

Method 2 - Using Inline Code Generation for CLA Subsystem to Blink an LED

In this method, `cla_subsystem` is triggered by a software at a rate of 0.5 sec and is configured with **Inline** as `Function packaging`. For more, “Method 2 - Inline Code Generation for CLA Subsystem” on page 1-96.

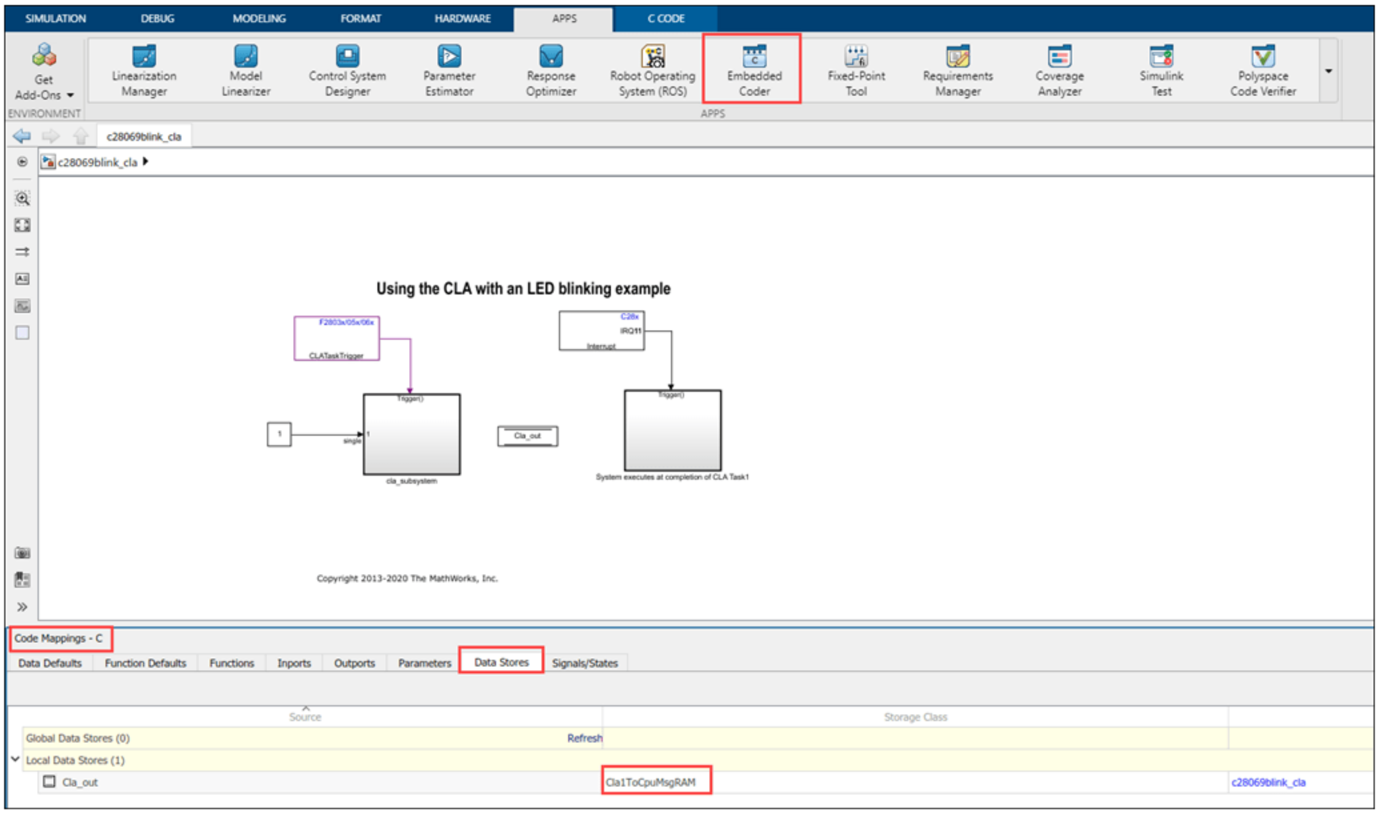
Following are the CLA Subsystem block configurations done for this model. To open the block Parameters,right-click on the CLA subsystem and select Block Parameters(Subsystem). Ensure the specified parameter values are the same if you want to run this example for other hardware board.



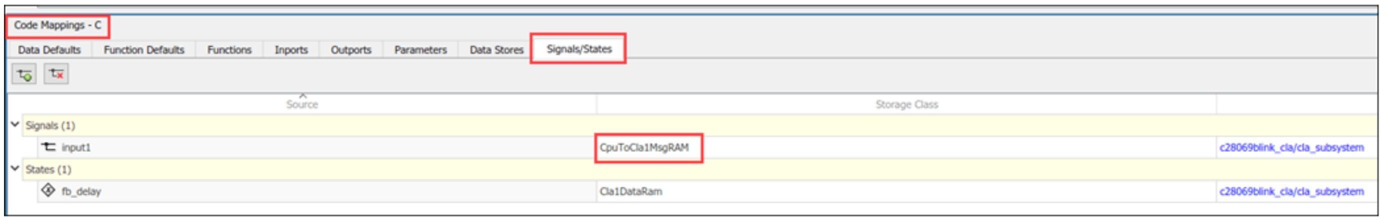
Block Parameters: cla_subsystem

Select the settings for the subsystem block. To enable parameters for code generation, select 'Treat as atomic unit'.

Function packaging: Inline



Source	Storage Class	
Global Data Stores (0) Refresh		
Local Data Stores (1)		
Cla_out	Cla1ToCpuMsgRAM	c28069blink_cla



Source	Storage Class	
Signals (1)		
input1	CpuToCla1MsgRAM	c28069blink_cla/cla_subsystem
States (1)		
fb_delay	Cla1DataRam	c28069blink_cla/cla_subsystem

Run the Model

1. Open the example model, c28035blink_cla.slx.
2. Open **Configuration Parameters** and select the required tool chain on the **Code Generation** pane.
3. Press **Ctrl+B** to build a binary executable and to automatically load and run the executable on the selected target.

Task 2: Ensure Data Integrity Between CPU and CLA

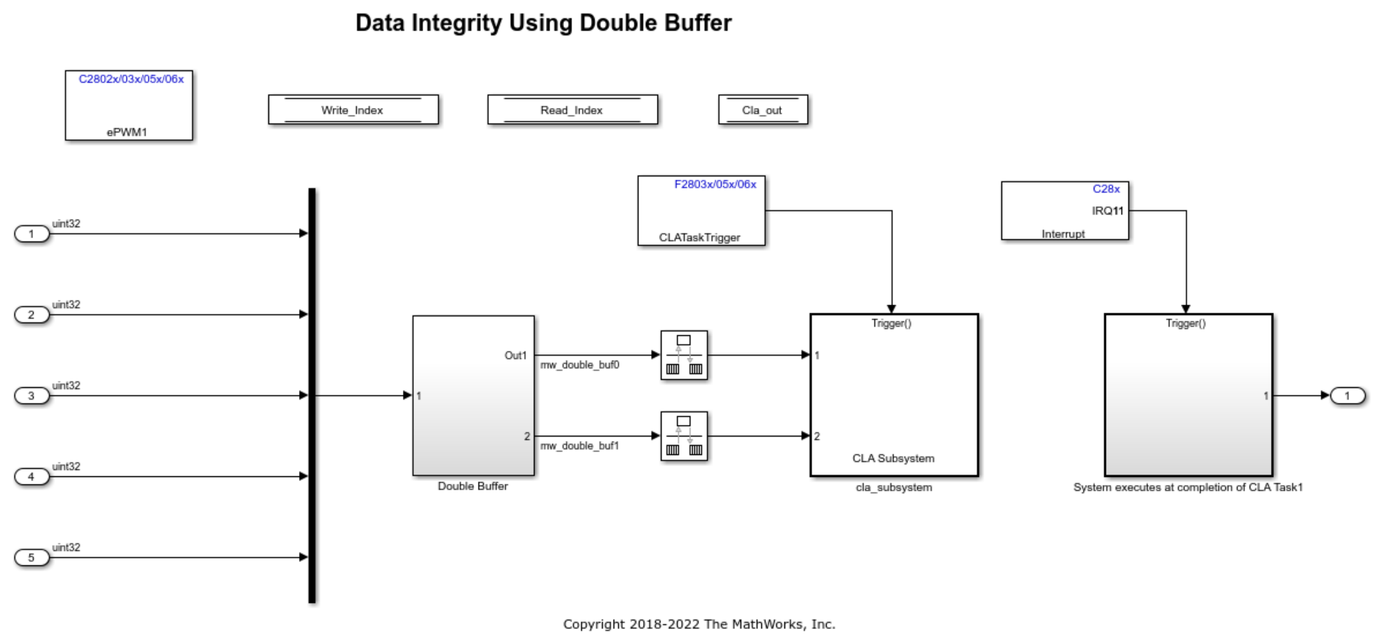
Data integrity issues may occur when:

- The data transfer is not atomic. For example, data transfers where the data size is more than the atomic size (uint16).
- Tasks are not synchronized. For example, the CLA is triggered asynchronously.
- Sample time of the CPU and the CLA are different.

To ensure data integrity, data must be protected either using a double buffer algorithm or by synchronizing both the CPU and the CLA.

In this model, the CPU and the CLA are not syncrhonized because the CLA is triggered asynchronously using an ePWM interrupt. To ensure data integrity, a double buffer algorithm along with control flags (Read_index and Write_index) are used.

```
open_system('c28069_dataintegrity_cla');
```



In the double buffer algorithm, two buffers mydoublebuf0 and mydoublebuf1 are used for data exchange. The CPU checks the buffer that is being read by the CLA using the Read_index flag and writes the data to the other buffer. Similarly, the CLA checks the buffer that is being written by the CPU using the Write_index flag and reads the data from the other buffer. For example, when Read_index is 0, data will be read from mydoublebuf0 in CLA.

By using the double buffer logic, data integrity is ensured even though the CPU and the CLA are not synchronous.

Run the Model

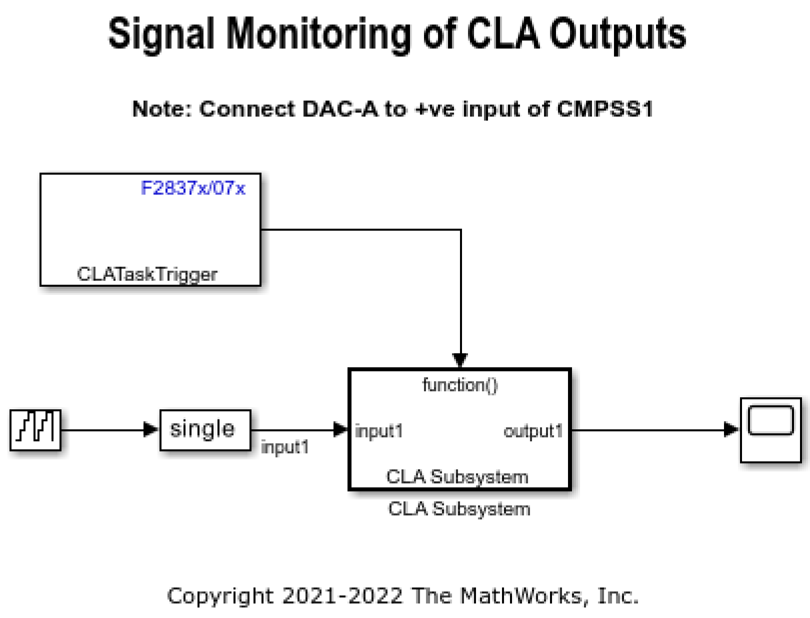
1. Open the c28069dataintegrity_cla.slx model.
2. On the **Hardware** tab, Click **Build, Deploy & Start** > **Build Stand-Alone** or press **Ctrl+B** to build and download the executable file on the CPU.

Task 3 : Signal Monitoring of CLA Output

In this task you will,

- Configure the model to read data from the DAC and CMPSS block and blink an LED inside the CLA task.
- Monitor & Tune the model to observe the CLA output.

```
open_system('c28379D_cmpss_cla_blink');
```

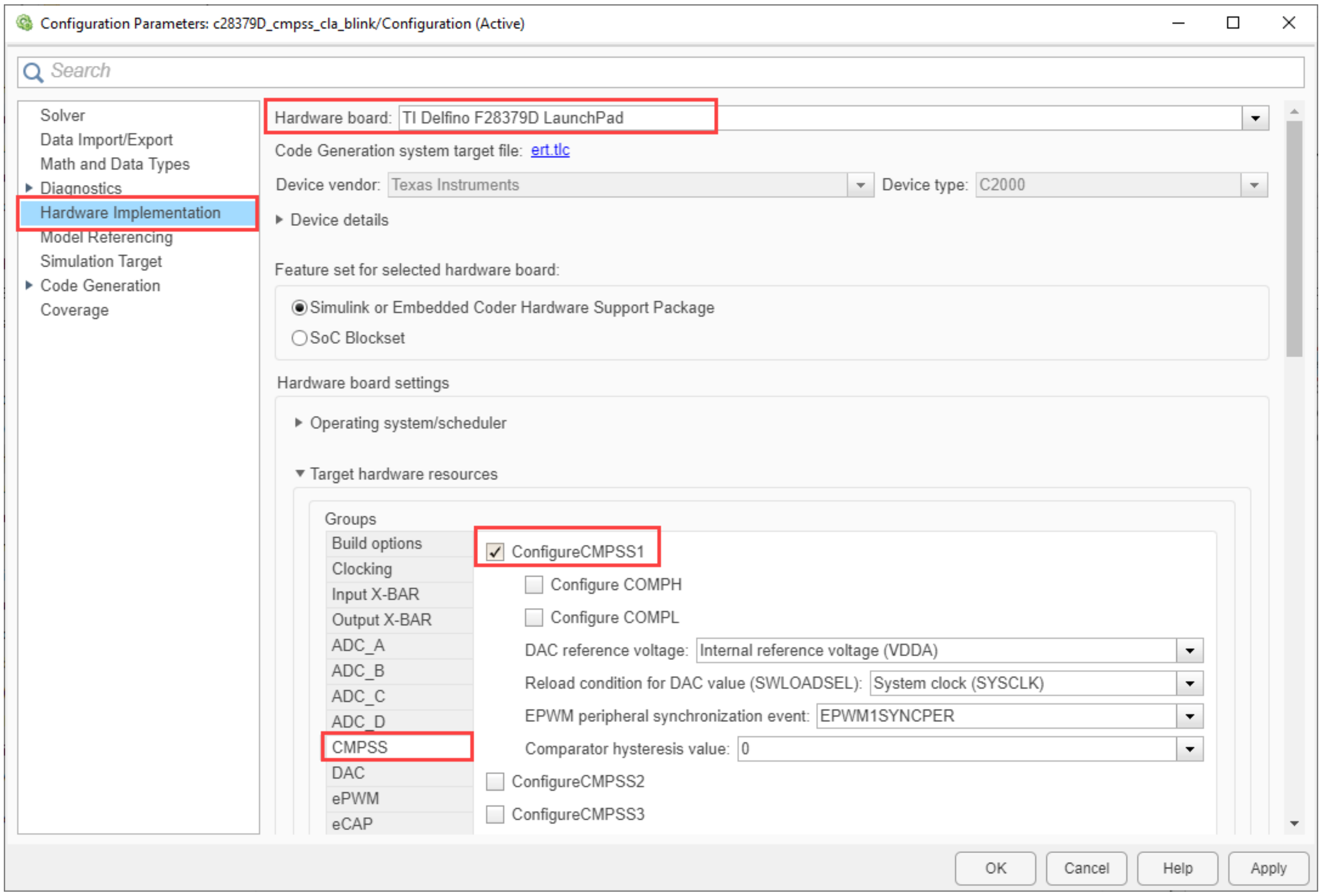

Signal Monitoring of CLA Outputs
Note: Connect DAC-A to +ve input of CMPSS1
F2837x/07x
CLATaskTrigger
single
input1
function()
input1
output1
CLA Subsystem
CLA Subsystem
Copyright 2021-2022 The MathWorks, Inc.

Input to the CLA Subsystem **input1** is stored in `CpuToCla1MsgRAM` using code mappings. CLA Subsystem is configured with `Nonreusable function` Function packaging. For more, see “Method 1 - Nonreusable Function Code Generation for CLA Subsystem (Recommended)” on page 1-95.

Configure the Model

1. Open the Using CMPSS Inside CLA to blink an LED model. This model is configured for TI Delfino F28379D LaunchPad hardware.

2. To run the model on other TI C2000 processors, press **Ctrl+E** to open the Configuration Parameters dialog box and select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.


Configuration Parameters: c28379D_cmpss_cla_blink/Configuration (Active)

Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Feature set for selected hardware board:

(●) Simulink or Embedded Coder Hardware Support Package

( ) SoC Blockset

Hardware board settings

Target hardware resources

Groups: CMPSS

[✓] ConfigureCMPSS1

[ ] Configure COMPH

[ ] Configure COMPL

DAC reference voltage: Internal reference voltage (VDDA)

Reload condition for DAC value (SWLOADSEL): System clock (SYSCLK)

EPWM peripheral synchronization event: EPWM1SYNCPER

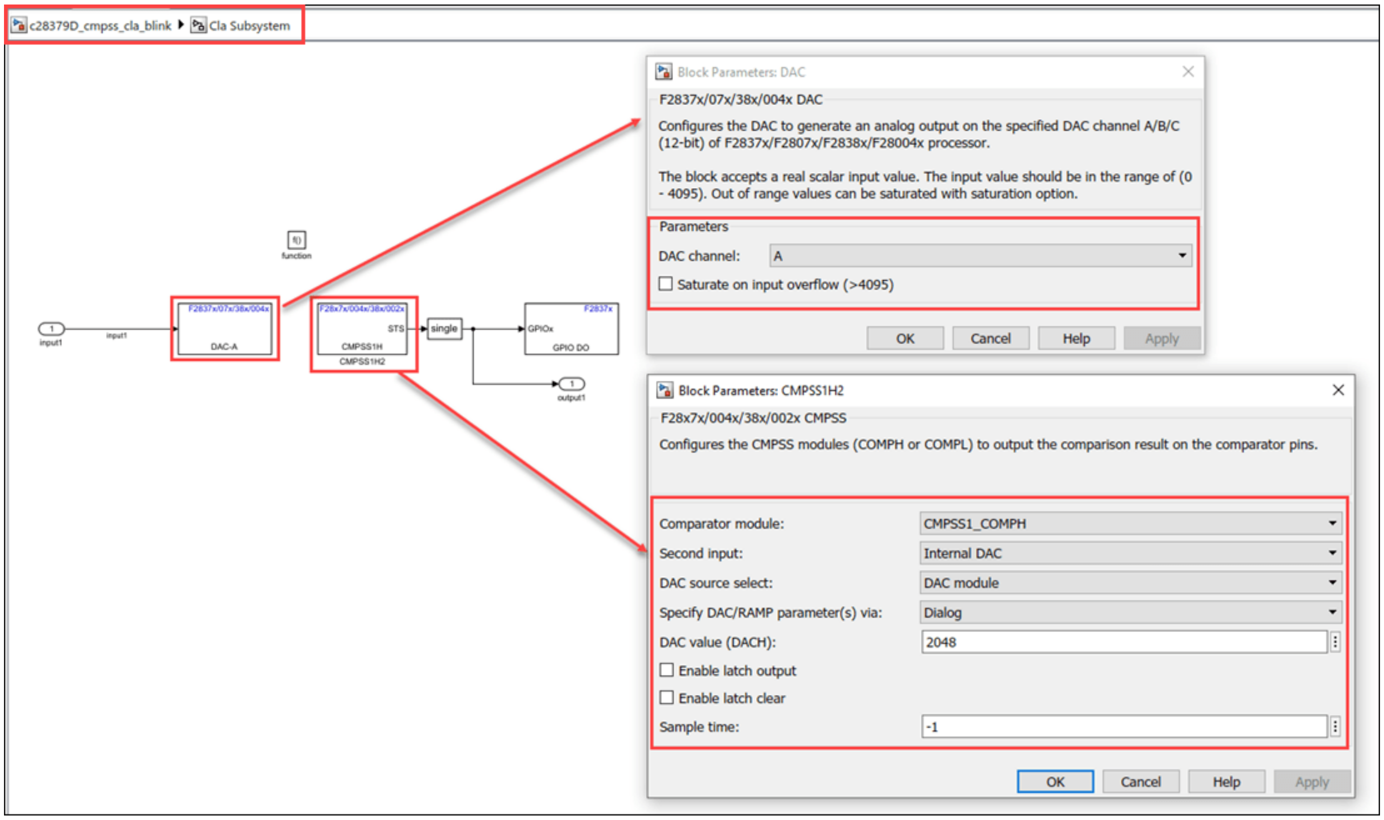
Comparator hysteresis value: 0

[ ] ConfigureCMPSS2

[ ] ConfigureCMPSS3

Driver block Configuration inside CLA

Following are the driver blocks CLA configurations done for this model. Double-click on the blocks to open block parameter configurations. Ensure the specified parameter values are the same if you want to run this example for other hardware board.



Block Parameters: DAC

F2837x/07x/38x/004x DAC

Configures the DAC to generate an analog output on the specified DAC channel A/B/C (12-bit) of F2837x/F2807x/F2838x/F28004x processor.

The block accepts a real scalar input value. The input value should be in the range of (0 - 4095). Out of range values can be saturated with saturation option.

Parameters

DAC channel: A

☐ Saturate on input overflow (>4095)

OK Cancel Help Apply

Block Parameters: CMPSS1H2

F28x7x/004x/38x/002x CMPSS

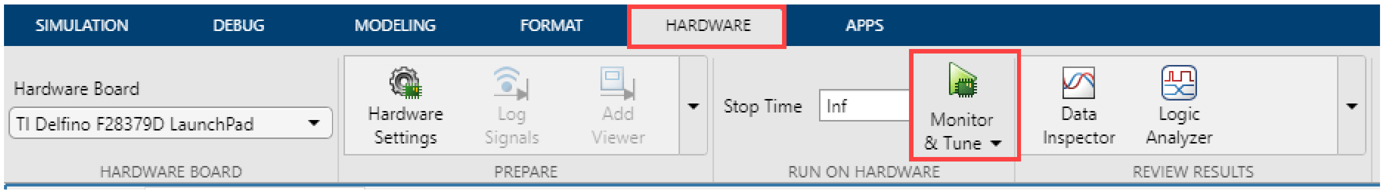
Configures the CMPSS modules (COMPH or COMPL) to output the comparison result on the comparator pins.

Comparator module:	CMPSS1_COMPH
Second input:	Internal DAC
DAC source select:	DAC module
Specify DAC/RAMP parameter(s) via:	Dialog
DAC value (DACH):	2048
☐ Enable latch output	
☐ Enable latch clear	
Sample time:	-1

OK Cancel Help Apply

Run the Model

1. Open **Hardware** tab and click **Monitor & Tune**.



2. Use the diagnostic viewer to follow the build progress and wait until the code loads and runs on the target hardware.

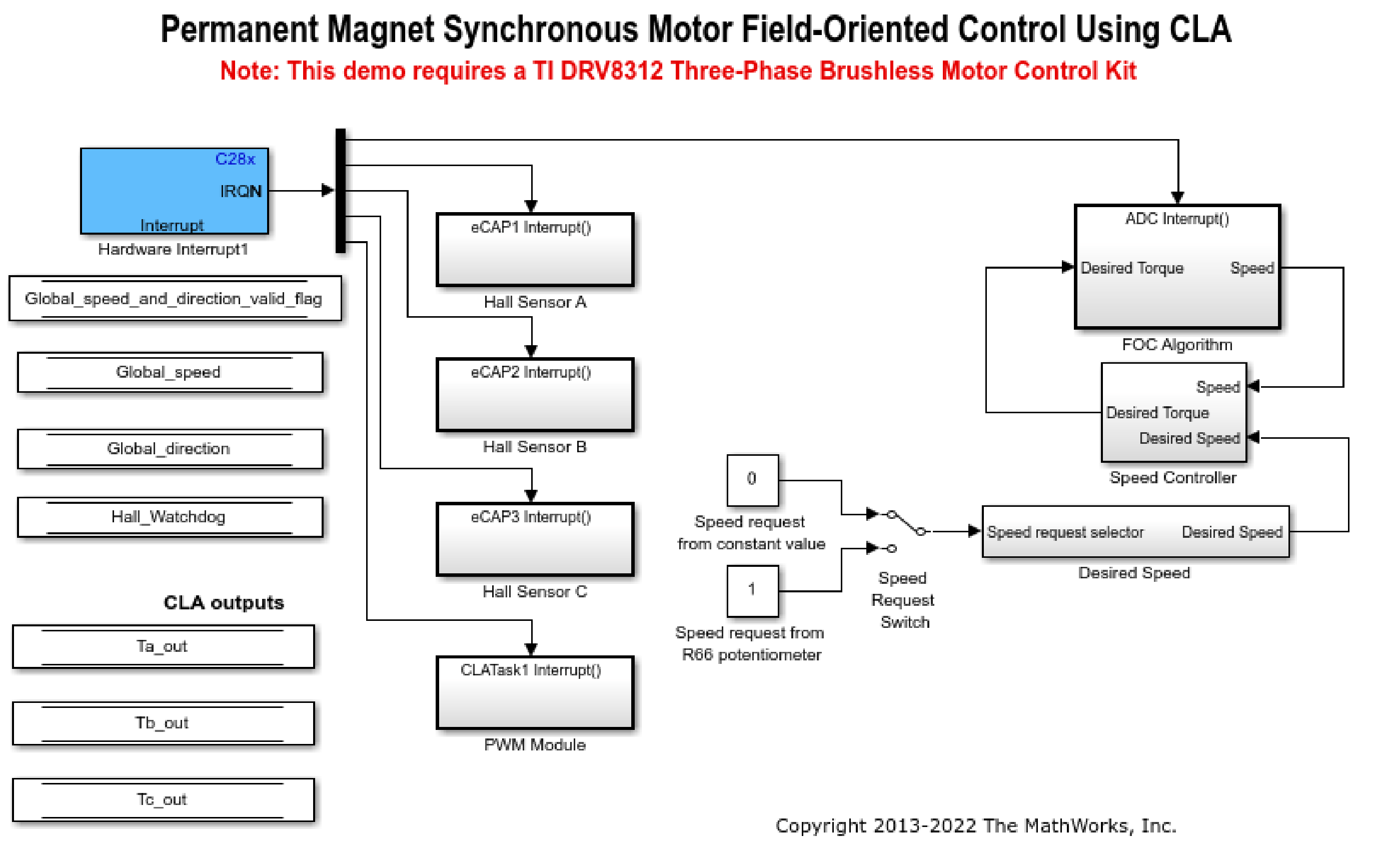
3. Observe the logged data on the Scope block and LED blinking.

Task 4: Permanent Magnet Synchronous Motor Field-Oriented Control (FOC) Using CLA

Input to this **Generating Raw Space Vectors** (c28035pmsmfoc_cla/FOC Algorithm/Torque Control Algorithm/Generating Space Vectors) is stored in `CpuToCla1MsgRAM` using code mappings. Generating Raw Space Vectors is configured with `Nonreusable function` Function packaging. For more, see “Method 1 - Nonreusable Function Code Generation for CLA Subsystem (Recommended)” on page 1-95.

The following figure shows a Permanent Magnet Synchronous Motor Field-Oriented Control (FOC) example model, which runs the FOC algorithm on the CLA.

```
open_system('c28069pmsmfoc_cla');
```



This model implements the Field-Oriented Control (FOC) of a Permanent Magnet Synchronous Motor using the CLA. The FOC algorithm is implemented with a CLA Task and PWM duty cycles are refreshed on completion of CLA Task using the corresponding interrupt. This example requires a DRV8312 kit with an F28069 or F28035 Control card. All the considerations listed in the blinking LED example are applied in this example.

In this model, data integrity is ensured by synchronizing the CPU and the CLA. The CLA is software triggered using the sample time inherited from the inputs. The Simulink scheduler ensures that all inputs are ready before the CLA subsystem is triggered.

In this example, a closed-loop Field-Oriented Control algorithm is used to regulate the speed and torque of a three-phase Permanent Magnet Synchronous Motor (PMSM). This example uses C28x peripheral blocks from the C2000™ Microcontroller Blockset. The algorithm in this example uses an asynchronous scheduling. The pulse width modulation (PWM) block triggers the ADC conversion. At the end of conversion, the ADC posts an interrupt that triggers the main FOC algorithm.

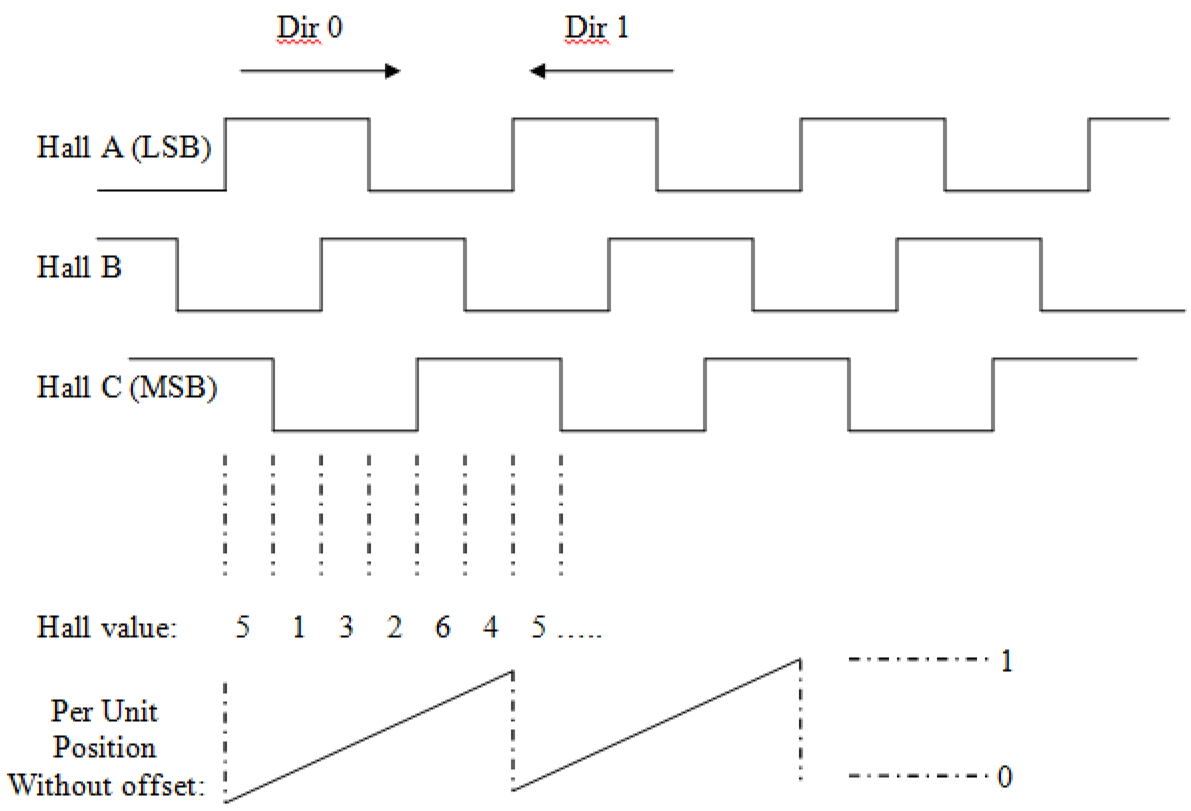
Configure the interrupt operation with Configuration Parameters > Hardware Implementation > Hardware board settings > Target hardware resources > External Interrupt.

Model Calibration Using DRV8312EVM

The 'Position Sensing Switch' in the model allows you to select sensorless position sensing or position sensing using Hall sensors. In case of sensorless (default), no calibration is needed. If using Hall sensors for position sensing, Hall sensors have to be connected to J10 of the DRV8312EVM board. The model concatenates the three Hall signals into a variable with Hall_A being the Least Significant Bit (LSB) and Hall_C being the Most Significant bit (MSB) of the variable.

The 'Speed Request Switch' allows you to select the source of the speed request. By default the speed will come from a normalized constant setting the speed request to half the acceptable speed range (0.5). For the DRV8312EVM, you can also select the speed request to come from potentiometer R66.

Interpretation of Hall Sensor Signals


Dir 0
Dir 1
Hall A (LSB)
Hall B
Hall C (MSB)
Hall value: 5 1 3 2 6 4 5 .....
Per Unit Position Without offset:
1
0

This section is not relevant if using sensor-less control. The model is configured with one interrupt for each Hall signal. In each Hall interrupt, there are four meaningful Hall values. Any other Hall value indicates a hardware problem. The Hall value read in a particular interrupt holds information about the motor's direction of rotation. For example, in Hall_A interrupt, reading a Hall value of 2 indicates that the motor is spinning in direction 0 and a falling edge has just occurred. If the model detects a direction change, it invalidates the direction and speed of the motor. For the speed to be valid, the model requires two consecutive edges with the same direction. Otherwise, the model sets a flag to invalidate the speed.

The following logic applies to the corresponding flag updates:

1. New_direction = Hall_direction
2. New_valid_flag = Previous_direction == Hall_direction;
3. Global_speed_and_direction_ready_flag = New_valid_flag && Old_valid_flag;

The Field Oriented Control algorithm takes a position signal from 0 to 1 reflecting an electrical revolution. If the speed signal is valid, the model performs a linear extrapolation from the Hall reading and accurately estimates the position.

Principle of the Hall based position estimation algorithm:

1. Read the halls.
2. Get the value of the latest timer (timer captured from the last interrupt that triggered).
3. Convert the time that elapsed from the previous edge to an electrical angle using the current speed.
4. If the speed information is not valid (speed is invalid after a direction change, at startup, when motor is stopped, when speed is too low...), the algorithm assumes that the position is in the middle of the 60 electrical degrees defined by the Hall reading. The maximum position signal error in these cases is therefore 30 electrical degrees.

The Hall decoder will be referenced (position = 0) when Hall_A is rising in direction 0. A Hall_position_offset variable is used to inform the FOC algorithm of the position difference between the Hall reference and the back EMF waveforms of the motor. Like the QEP example, this value has to be calibrated by comparing the Hall signals with the back EMF waveforms of the motor. In the DRV8312 example, Hall_position_offset is normalized on the electrical revolution and is set to 0.57, to match the characteristics of the motors included in the DRV8312EVM.

Run the Model

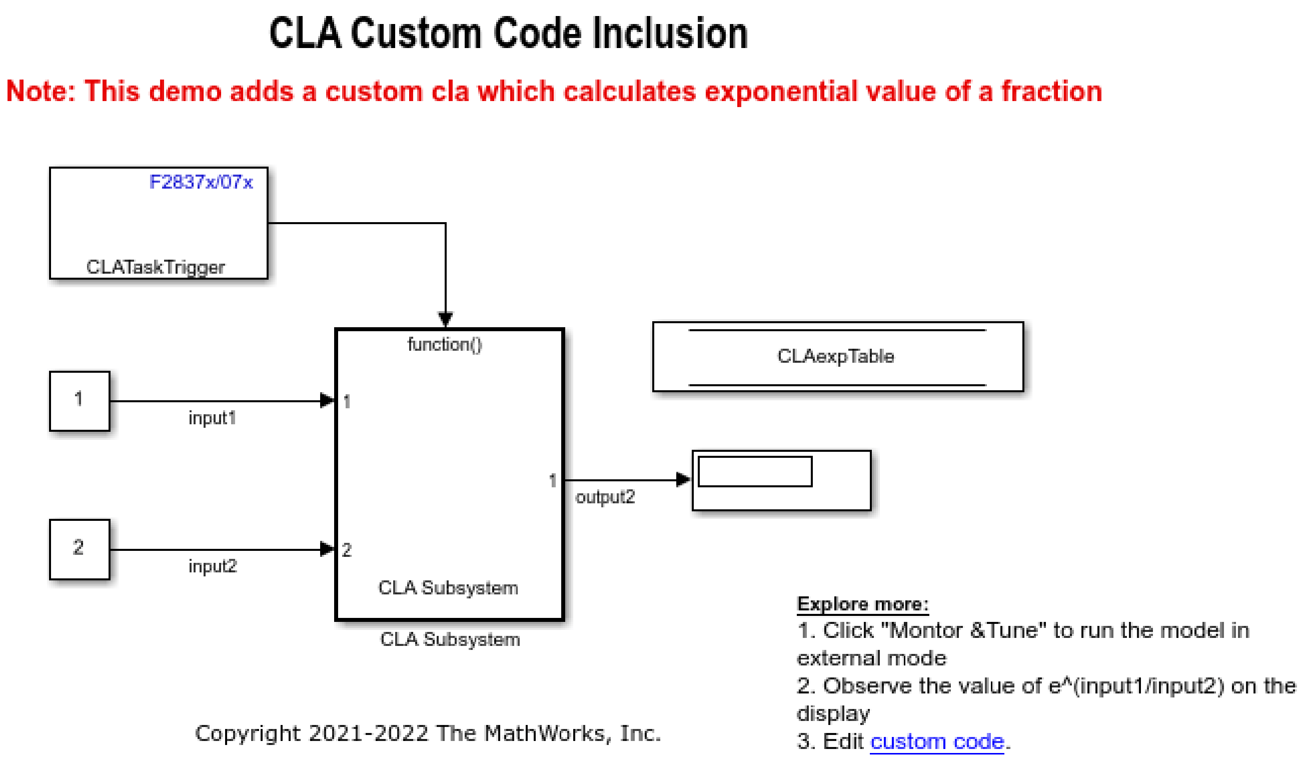
1. Open the example model c28069pmsmfoc_cla.slx or c28035pmsmfoc_cla.slx
2. Open **Configuration Parameters** and select the required tool chain on the **Code Generation** pane.
3. Press **Ctrl+B** to build a binary executable and to automatically load and run the executable on the selected target.

Task 5: CLA Custom Code Inclusion

In this task you will,

- Include a custom CLA file in the model
- Custom CLA file calculates the exponential value of a fraction and sends the output to Simulink.
- Monitor & Tune to observe the CLA output.

```
open_system('c28379D_custom_code_cla');
```


CLA Custom Code Inclusion
Note: This demo adds a custom cla which calculates exponential value of a fraction
F2837x/07x
CLATaskTrigger
function()
CLAexpTable
1
input1
1
1
output2
2
input2
2
CLA Subsystem
CLA Subsystem
Explore more:
1. Click "Montor &Tune" to run the model in external mode
2. Observe the value of e^(input1/input2) on the display
3. Edit custom code.
Copyright 2021-2022 The MathWorks, Inc.

In this model, we are accessing a function defined in a custom cla file from CLA Task. The user can configure his own custom CLA code and include it in the build process as shown in this example.

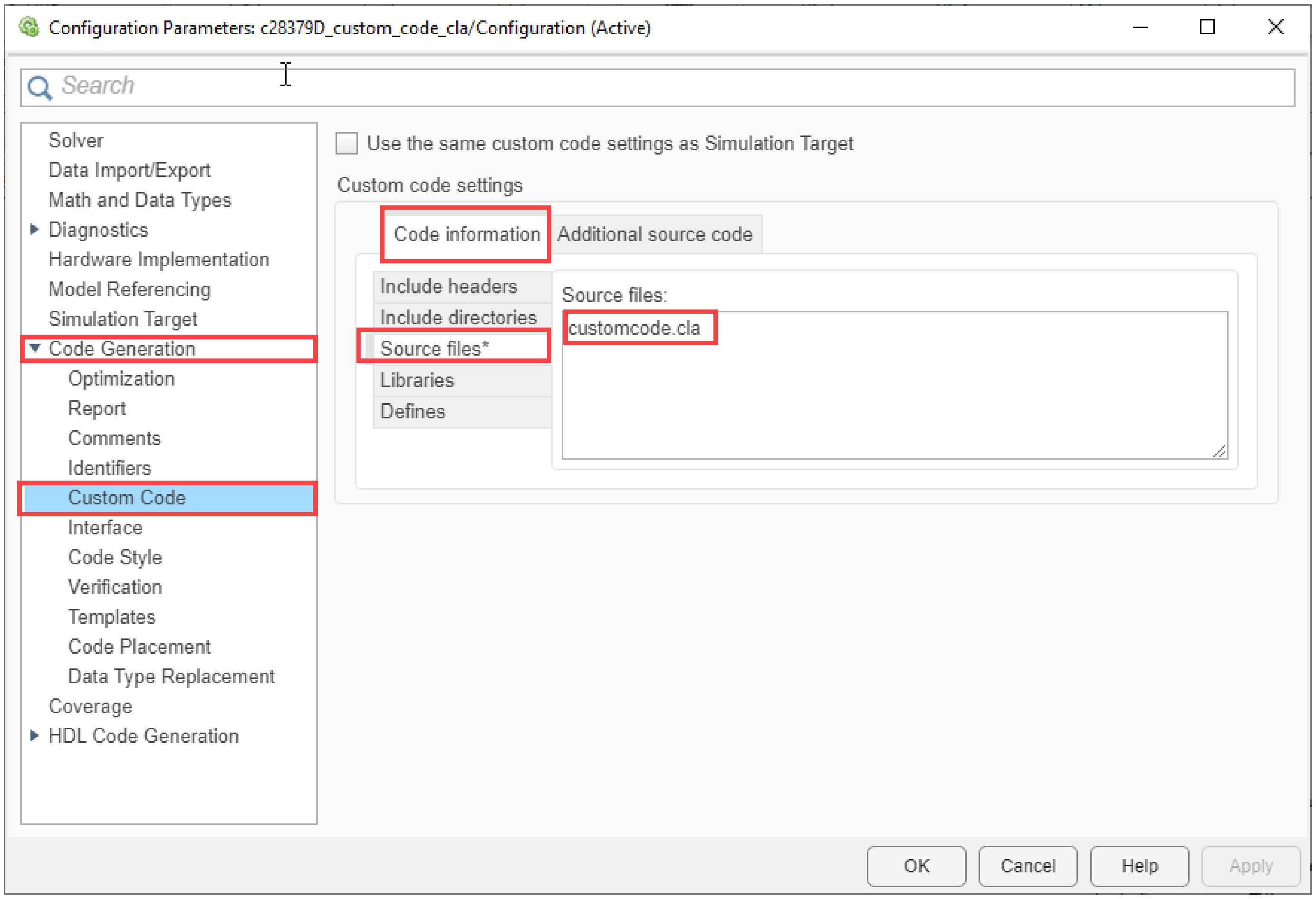
Inputs to the CLA Subsystem **input1** and **input2** are stored in `CpuToCla1MsgRAM` using code mappings. CLA Subsystem is configured with `Nonreusable function` Function packaging. For more, see “Method 1 - Nonreusable Function Code Generation for CLA Subsystem (Recommended)” on page 1-95.

Configure the Model

1. Open the Using CLA Custom Code Inclusion model. This model is configured for TI Delfino F28379D LaunchPad hardware.

2. To run the model on other TI C2000 processors, press **Ctrl+E** to open the Configuration Parameters dialog box and select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

3. In Configuration Parameters dialog box, navigate to **Code Generation** > **Custom Code** > **Code information** > **Source files** and select the **customcode.cla** file.


Configuration Parameters: c28379D_custom_code_cla/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- ▸ Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- ▾ Code Generation
  - Optimization
  - Report
  - Comments
  - Identifiers
  - Custom Code
  - Interface
  - Code Style
  - Verification
  - Templates
  - Code Placement
  - Data Type Replacement
- Coverage
- ▸ HDL Code Generation

☐ Use the same custom code settings as Simulation Target

Custom code settings

Code information | Additional source code

- Include headers
- Include directories
- Source files*
- Libraries
- Defines

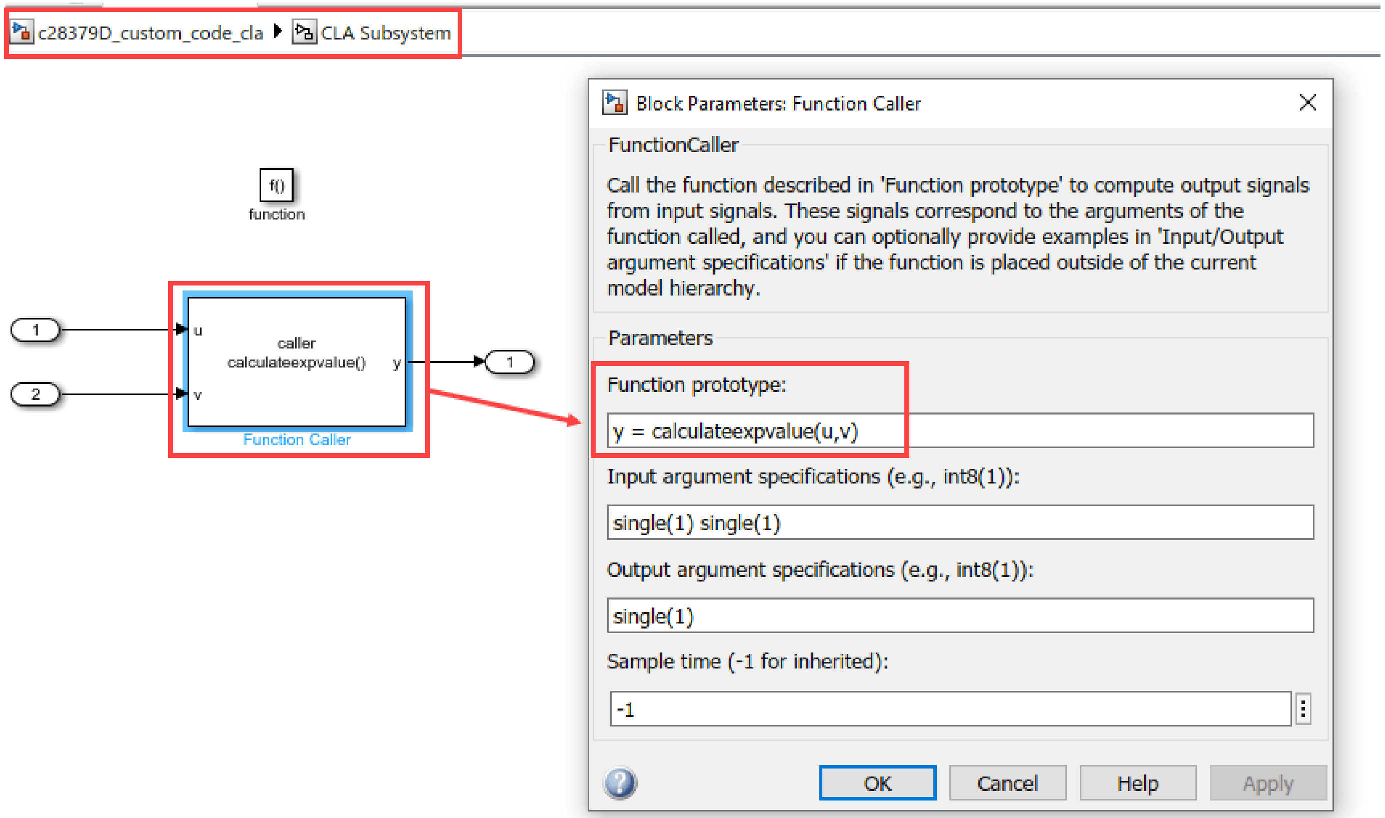
Source files:

customcode.cla

Function Caller block Configuration inside CLA

Following are the Function Caller block CLA configurations done for this model. Double-click on the blocks to open block parameter configurations. Ensure the specified parameter values are the same if you want to run this example for other hardware board.

calcuateexpvalue(u,v) is a user defined function, defined in a custom CLA file(customcode.cla) called from the CLA Task in the `cla_task.cla` file.



Call the function described in 'Function prototype' to compute output signals from input signals. These signals correspond to the arguments of the function called, and you can optionally provide examples in 'Input/Output argument specifications' if the function is placed outside of the current model hierarchy.

Function prototype: y = calculateexpvalue(u,v)

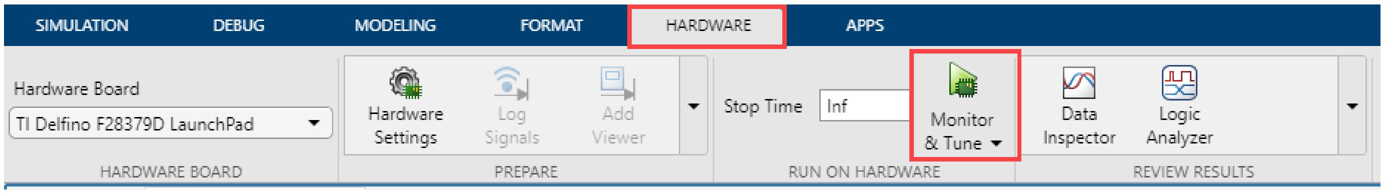
Input argument specifications (e.g., int8(1)): single(1) single(1)

Output argument specifications (e.g., int8(1)): single(1)

Sample time (-1 for inherited): -1

Run the Model

1. Open **Hardware** tab and click **Monitor & Tune**.



2. Use the diagnostic viewer to follow the build progress and wait until the code loads and runs on the target hardware.

3. Observe the value of exponential e^(input1/input2) on the Display block.

Other Things to Try

- Try to run the example with different driver blocks and analyse the results.

More About

- C28x CLA Task
- “Overview of CLA Configuration for C2000 Processors Using Subsystem” on page 1-93

- “Digital DC/DC Buck Converter Using Peak Current Mode Control” on page 4-43

Getting Started with C2000 Microcontroller Blockset for F28M3x Concerto Processors

In this example, you will learn how to configure a simple Simulink® model to generate code for ARM and C28x cores of Concerto F28M3x processors and run the generated code on the board to periodically turn LEDs on and off.

C2000™ Microcontroler Blockset enables you to create and run Simulink models on Texas Instruments C28x + ARM Cortex-M3 MCUs. The blockset includes a library of Simulink blocks for configuring and accessing F28M3x peripherals and communication interfaces.

Prerequisites

- If you are new to Simulink, we recommend completing the Interactive Simulink Tutorial.
- If you are new to Embedded Coder, see the Embedded Coder product page for an overview and tutorials.

Required Hardware

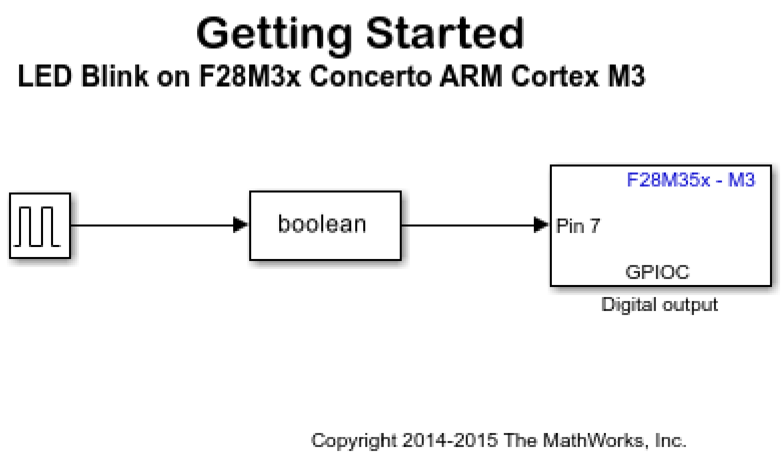
For this example, you will need one of the following hardware with Texas Instruments XDS100v2 USB Emulator connection:

- F28M36 Concerto Control Card or
- F28M35 Concerto Control Card

Configure the Model to Run on the ARM Cortex M3 Core of the F28M3x Processor

1. Open the ARM LED model. This model is configured to run on the F28M35x ARM Cortex M3 CPU. You can run the model on F28M36x by navigating to **Configuration Parameters** > **Hardware Implementation** > **Hardware board** and selecting **TI Concerto F28M36x (ARM Cortex-M3)**.
2. The red LED (LD3) on the F28M35H52C control card is driven by PC7_GPIO71. In the **Digital output** block, set the **GPIO port** to **GPIOC** and **Pin number** to 7.
3. The red LED (D2) on the F28M36P63C control card is driven by PF2_GPIO34. In the **Digital output** block, set the **GPIO port** to **GPIOF** and **Pin number** to 2.
4. Configure the GPIOs from the **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources** pane. By default, all pins are enabled on the C28x if not used by the ARM Simulink Model.
5. In **Configuration Parameters** > **Hardware Implementation**, ensure that **Build action** is set to **Build, load and run** and **Boot From Flash (stand alone execution)** is enabled.
6. Press Ctrl+B to build, load, and run the application on the hardware board. Observe that the red LED blinks once in every second.

```
open_system('led_blink_f28m35_m3');
```

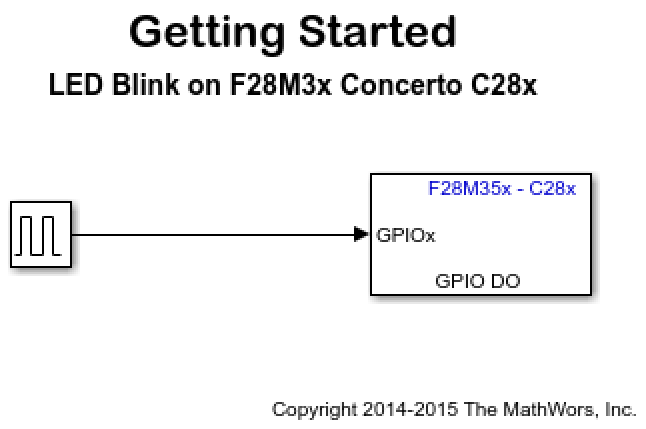

Getting Started
LED Blink on F28M3x Concerto ARM Cortex M3
boolean
F28M35x - M3
Pin 7
GPIOC
Digital output
Copyright 2014-2015 The MathWorks, Inc.

Configure the Model to Run on the C28x CPU of the F28M3x Processor

Before running the model on the C28x CPU (RAM or Flash), configure the clock settings for the C28x CPU. You can do this by running the above ARM LED model.

1. Open the C28x LED model. This model is configured to run on the F28M35x C28x CPU. You can run the model on F28M36x by navigating to **Configuration Parameters** > **Hardware Implementation** > **Hardware board** and selecting **TI Concerto F28M36x (C28x)**.
2. The red LED (LD2) on the F28M35H52C control card is driven by PC6_GPIO70. In the **Digital Output** block, set **GPIO Group** to `GPIO68~GPIO71` and select GPIO70.
3. The red LED (D1) on the F28M36P63C control card is driven by PE7_GPIO31. In the **Digital Output** block, set **GPIO Group** to `GPIO24~GPIO31` and select GPIO31.
4. In **Configuration Parameters > Hardware Implementation**, ensure that **Build action** is set to **Build, load and run** and **Boot From Flash (stand alone execution)** is enabled.
5. Press Ctrl+B to build, load, and run the application on the hardware board. Observe that the red LED LD2 on the board is ON for 0.5 second and OFF for 0.5 second.

```
open_system('led_blink_f28m35_c28');
```


Getting Started
LED Blink on F28M3x Concerto C28x
F28M35x - C28x
GPIOx
GPIO DO
Copyright 2014-2015 The MathWors, Inc.

Using Time-Base Counter Synchronization to Synchronize ePWMs and eCAPS

This example shows how to use time-base counter synchronization in C2000™ Microcontroller Blockset.

In this example you will learn how to:

- Synchronize ePWM4 with ePWM1 with a phase delay of 120 degrees
- Synchronize eCAP1 with ePWM4 with a phase delay of 120 degrees

Prerequisites

Complete the following tutorials:

- “Getting Started with Texas Instruments C2000 Microcontroller Blockset” on page 4-2
- “Overview of Time-Base Synchronization in ePWM Type 4” on page 1-130

Required Hardware

TI Delfino F28379D LaunchPad or TI F2838x control card

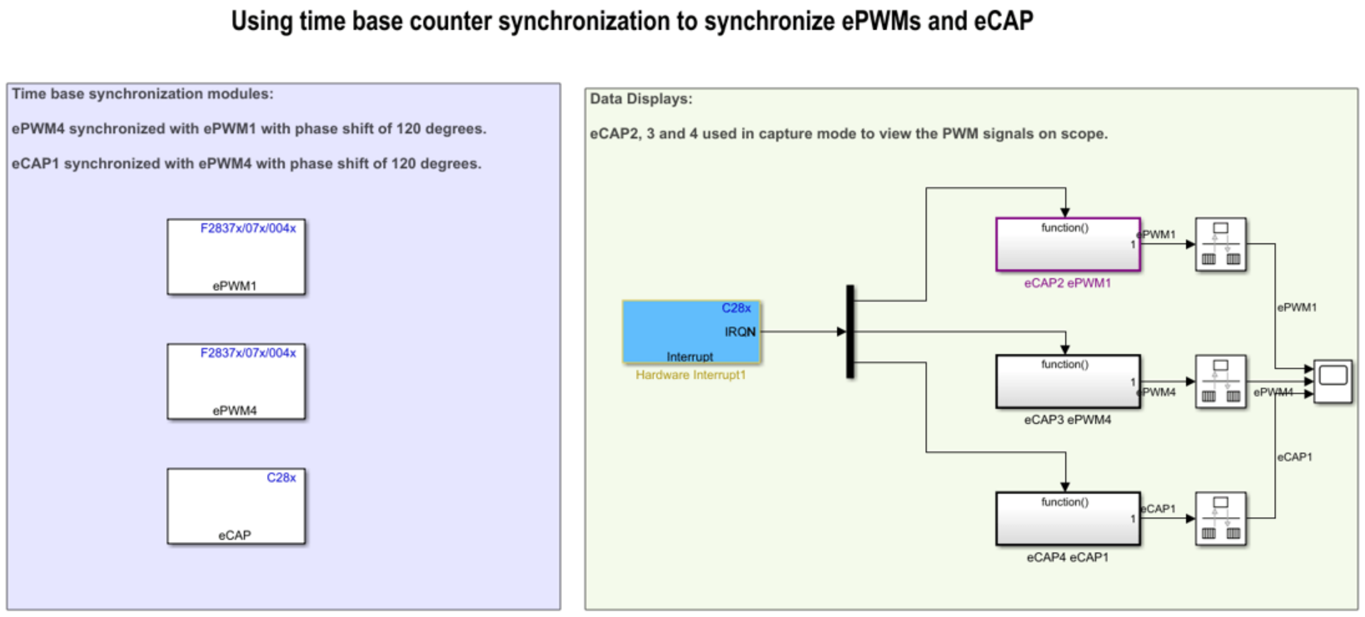
Available Models

- TI F28379D LaunchPad:f2837x_TimeBaseSync
- TI F2838x control card:f2838x_TimeBaseSync

Model

To open the model, type the following command in the MATLAB® command prompt.

```
open_system('f2837x_TimeBaseSync.slx');
```

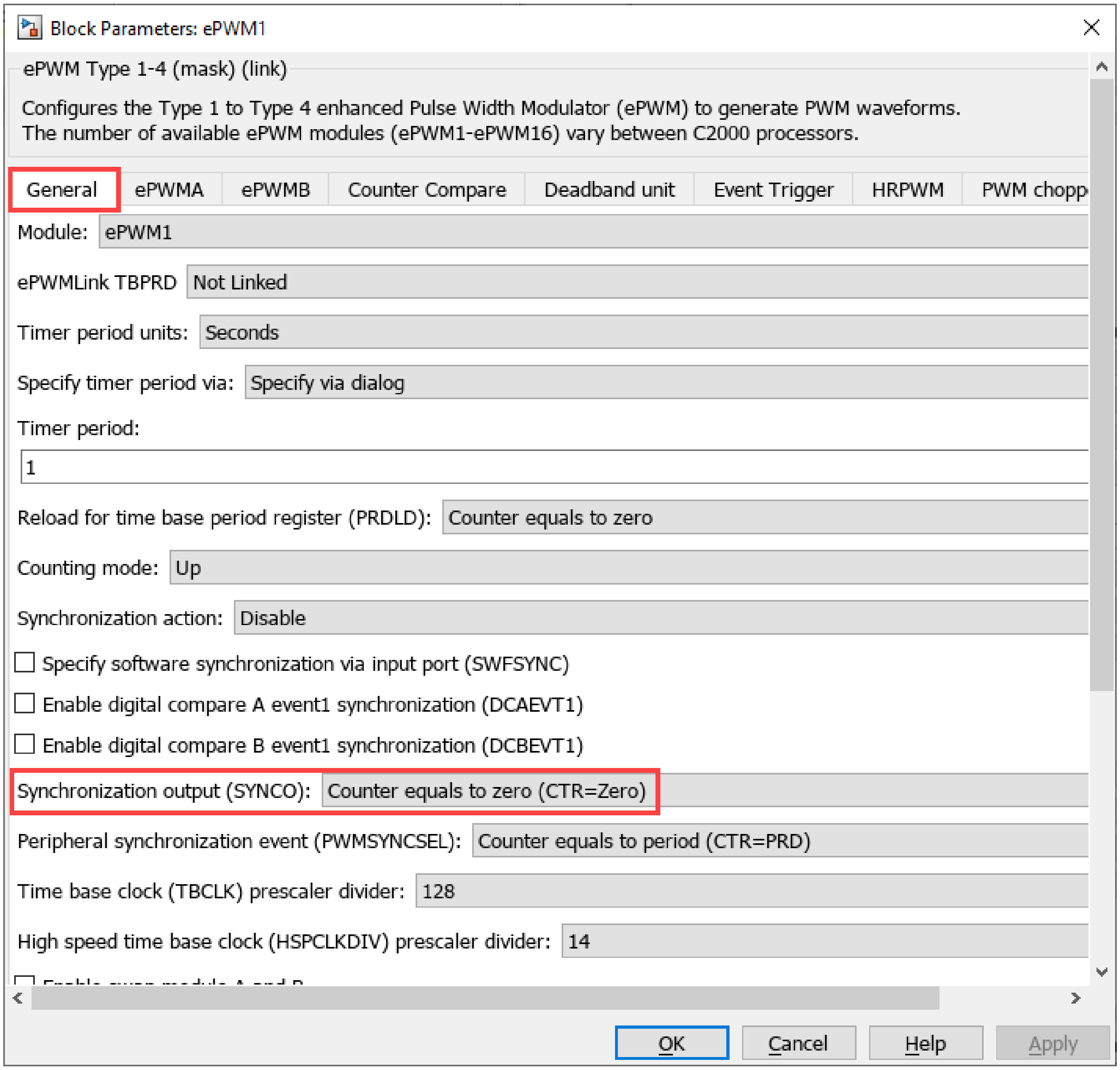

Using time base counter synchronization to synchronize ePWMs and eCAP
Time base synchronization modules:
ePWM4 synchronized with ePWM1 with phase shift of 120 degrees.
eCAP1 synchronized with ePWM4 with phase shift of 120 degrees.
F2837x/07x/004x
ePWM1
F2837x/07x/004x
ePWM4
C28x
eCAP
Data Displays:
eCAP2, 3 and 4 used in capture mode to view the PWM signals on scope.
C28x
IRQN
Interrupt
Hardware Interrupt1
function()
eCAP2 ePWM1
function()
eCAP3 ePWM4
function()
eCAP4 eCAP1
ePWM1
ePWM4
eCAP1

In the model, the ePWM1 and ePWM4 blocks are configured to generate a signal of time period 1 second and duty cycle of 50 %. Similarly, the eCAP1 is configured to work in the **APWM** mode and generate a signal of period 1 second and duty cycle of 50 %. The ePWM4 block is synchronized with the ePWM1 with a phase shift of 120 degrees, and eCAP1 is synchronized with the ePWM4 with a phase shift of 120 degrees.

eCAP2, eCAP3 and eCAP4 are used to capture the waveform generated by the time-base synchronization module and display it in the **Scope** block.

Configure ePWM and eCAP Modules for Time-Base Counter Synchronization

- Configure the ePWM1 block to send the signal by setting the Synchronization output (SYNCO) parameter to Counter equals to zero (CRT=Zero)



ePWM Type 1-4 (mask) (link)

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms. The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors.

Module: ePWM1

ePWMLink TBPRD Not Linked

Timer period units: Seconds

Specify timer period via: Specify via dialog

Timer period:

1

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up

Synchronization action: Disable

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

☐ Enable digital compare B event1 synchronization (DCBEVT1)

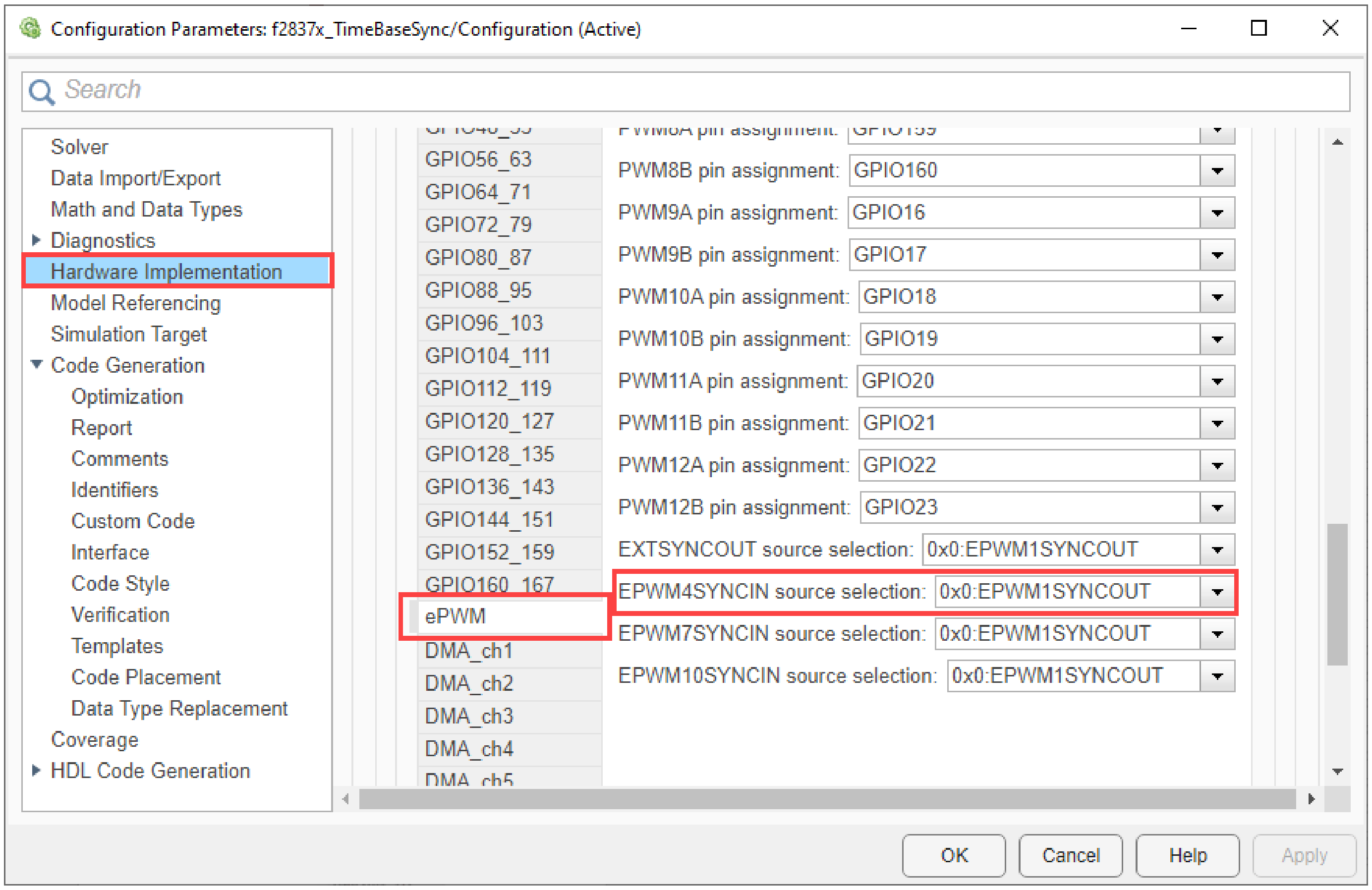
Synchronization output (SYNCO): Counter equals to zero (CTR=Zero)

Peripheral synchronization event (PWMSYNCSEL): Counter equals to period (CTR=PRD)

Time base clock (TBCLK) prescaler divider: 128

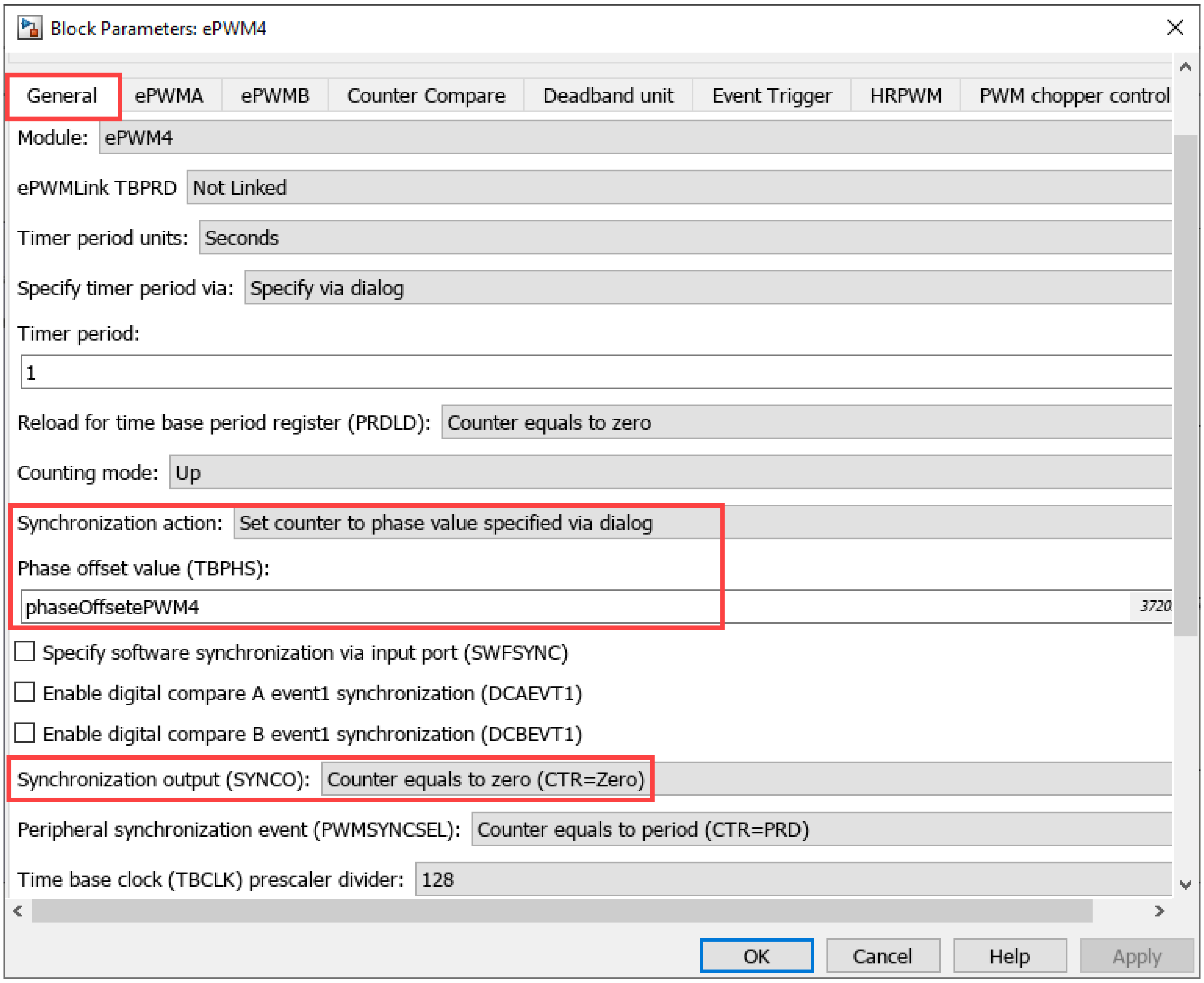
High speed time base clock (HSPCLKDIV) prescaler divider: 14

- ePWM4 is configured to have synchronization input from ePWM1 SYNCOUT in the Configuration Parameters.


Configuration Parameters: f2837x_TimeBaseSync/Configuration (Active)

PWM8A pin assignment: GPIO159
PWM8B pin assignment: GPIO160
PWM9A pin assignment: GPIO16
PWM9B pin assignment: GPIO17
PWM10A pin assignment: GPIO18
PWM10B pin assignment: GPIO19
PWM11A pin assignment: GPIO20
PWM11B pin assignment: GPIO21
PWM12A pin assignment: GPIO22
PWM12B pin assignment: GPIO23
EXTSYNCOUT source selection: 0x0:EPWM1SYNCOUT
EPWM4SYNCIN source selection: 0x0:EPWM1SYNCOUT
EPWM7SYNCIN source selection: 0x0:EPWM1SYNCOUT
EPWM10SYNCIN source selection: 0x0:EPWM1SYNCOUT

- The ePWM4 block is configured for phase offset value and send the SYNCOUT signal when `CTR=Zero`.


Block Parameters: ePWM4

Module: ePWM4

ePWMLink TBPRD Not Linked

Timer period units: Seconds

Specify timer period via: Specify via dialog

Timer period:

1

Reload for time base period register (PRDLD): Counter equals to zero

Counting mode: Up

Synchronization action: Set counter to phase value specified via dialog

Phase offset value (TBPHS):

phaseOffsetePWM4 3720

☐ Specify software synchronization via input port (SWFSYNC)

☐ Enable digital compare A event1 synchronization (DCAEVT1)

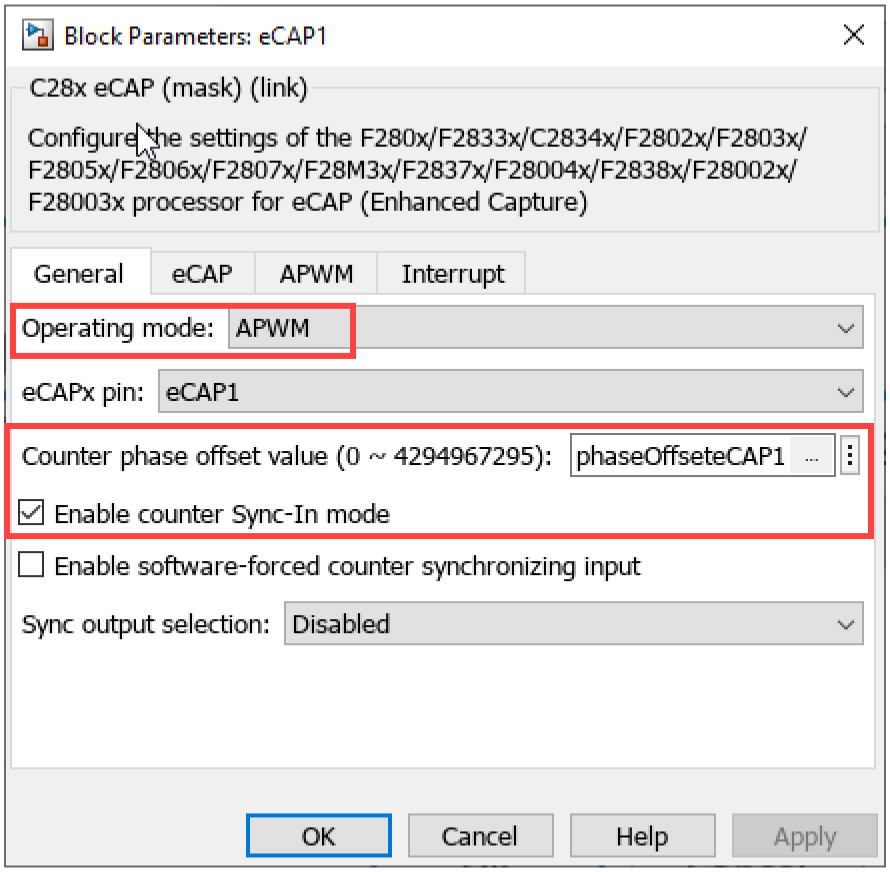
☐ Enable digital compare B event1 synchronization (DCBEVT1)

Synchronization output (SYNCO): Counter equals to zero (CTR=Zero)

Peripheral synchronization event (PWMSYNCSEL): Counter equals to period (CTR=PRD)

Time base clock (TBCLK) prescaler divider: 128

- eCAP1 block is configured to operate in APWM mode and enabled to have synchronization input in the block. The counter phase offset value is also set in the block parameter as shown.


Block Parameters: eCAP1

C28x eCAP (mask) (link)

Configure the settings of the F280x/F2833x/C2834x/F2802x/F2803x/F2805x/F2806x/F2807x/F28M3x/F2837x/F28004x/F2838x/F28002x/F28003x processor for eCAP (Enhanced Capture)

Operating mode: APWM

eCAPx pin: eCAP1

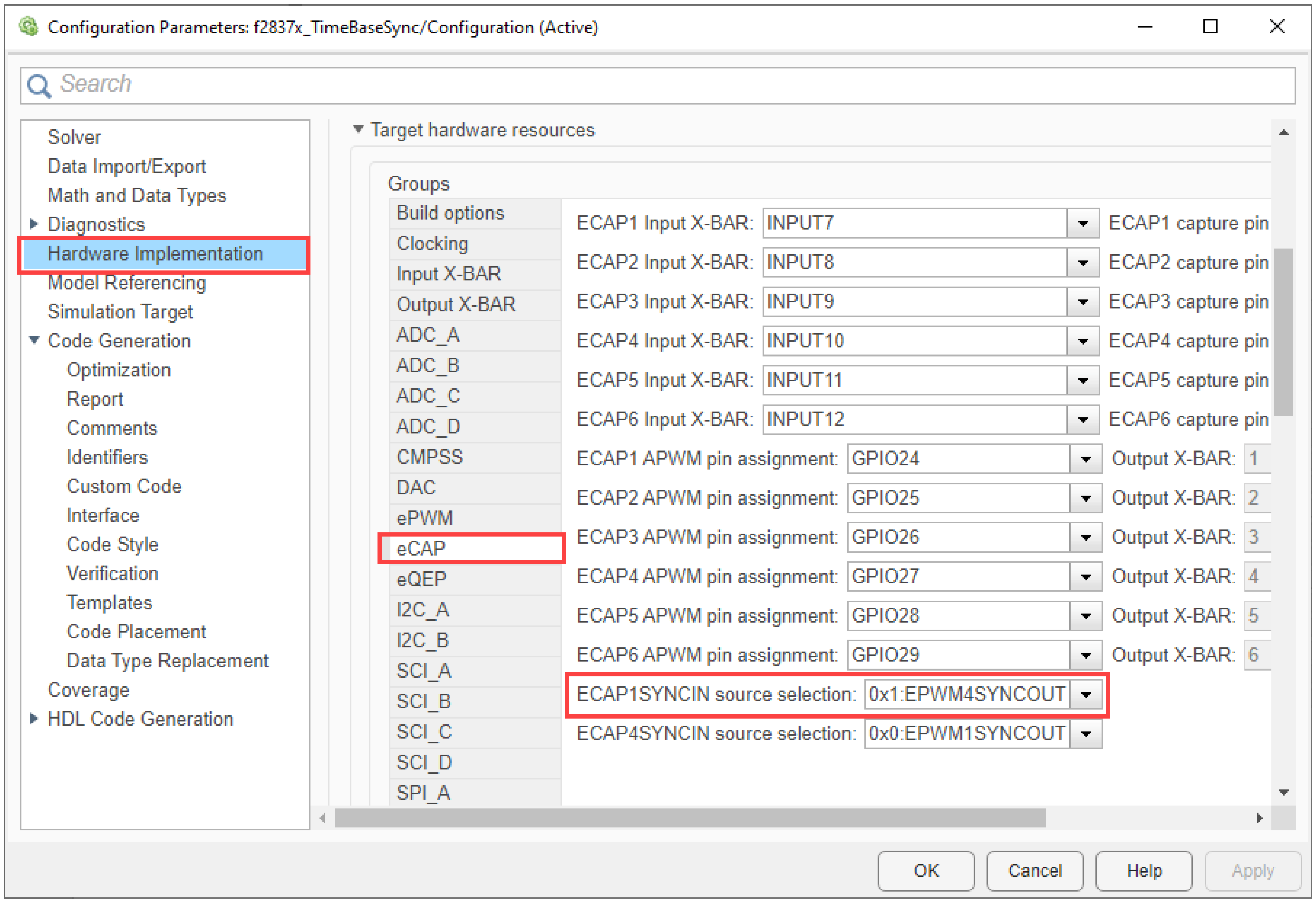
Counter phase offset value (0 ~ 4294967295): phaseOffseteCAP1

☑ Enable counter Sync-In mode

☐ Enable software-forced counter synchronizing input

Sync output selection: Disabled

- eCAP1 is configured to have synchronization input from ePWM4 SYNCOUT in Configuration Parameters.


Configuration Parameters: f2837x_TimeBaseSync/Configuration (Active)

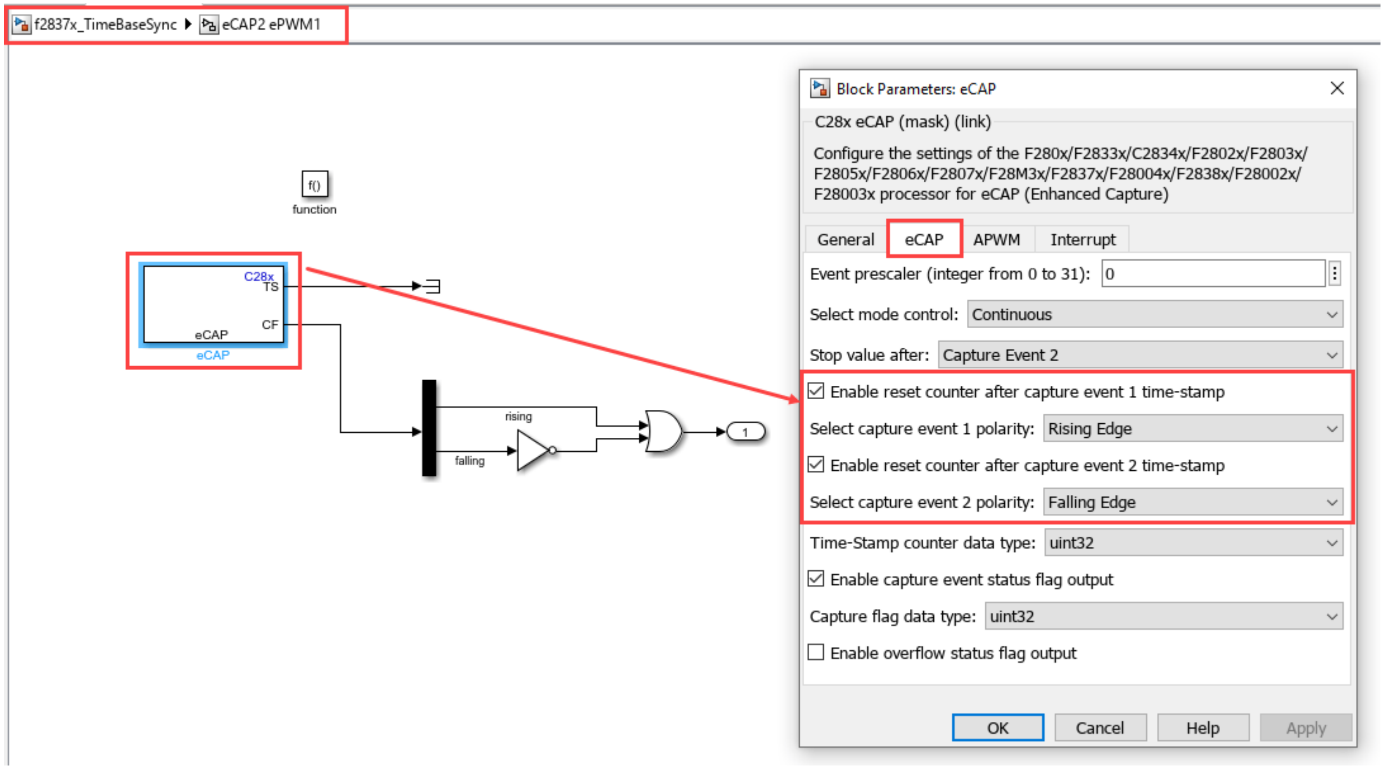
Target hardware resources

Groups: eCAP

ECAP1 Input X-BAR:	INPUT7	ECAP1 capture pin	
ECAP2 Input X-BAR:	INPUT8	ECAP2 capture pin	
ECAP3 Input X-BAR:	INPUT9	ECAP3 capture pin	
ECAP4 Input X-BAR:	INPUT10	ECAP4 capture pin	
ECAP5 Input X-BAR:	INPUT11	ECAP5 capture pin	
ECAP6 Input X-BAR:	INPUT12	ECAP6 capture pin	
ECAP1 APWM pin assignment:	GPIO24	Output X-BAR:	1
ECAP2 APWM pin assignment:	GPIO25	Output X-BAR:	2
ECAP3 APWM pin assignment:	GPIO26	Output X-BAR:	3
ECAP4 APWM pin assignment:	GPIO27	Output X-BAR:	4
ECAP5 APWM pin assignment:	GPIO28	Output X-BAR:	5
ECAP6 APWM pin assignment:	GPIO29	Output X-BAR:	6
ECAP1SYNCIN source selection:	0x1:EPWM4SYNCOUT		
ECAP4SYNCIN source selection:	0x0:EPWM1SYNCOUT		

Configure eCAP blocks to Capture PWM Waveforms

- Hardware interrupts eCAP2, eCAP3 and eCAP4 are configured to capture the generated PWM waveforms on the **Scope** block. The eCAP blocks are configured in eCAP mode to capture two events i.e. `Rising edge` and `Falling edge` of the signal.



Block Parameters: eCAP

C28x eCAP (mask) (link)

Configure the settings of the F280x/F2833x/C2834x/F2802x/F2803x/F2805x/F2806x/F2807x/F28M3x/F2837x/F28004x/F2838x/F28002x/F28003x processor for eCAP (Enhanced Capture)

General | eCAP | APWM | Interrupt

Event prescaler (integer from 0 to 31): 0

Select mode control: Continuous

Stop value after: Capture Event 2

☑ Enable reset counter after capture event 1 time-stamp

Select capture event 1 polarity: Rising Edge

☑ Enable reset counter after capture event 2 time-stamp

Select capture event 2 polarity: Falling Edge

Time-Stamp counter data type: uint32

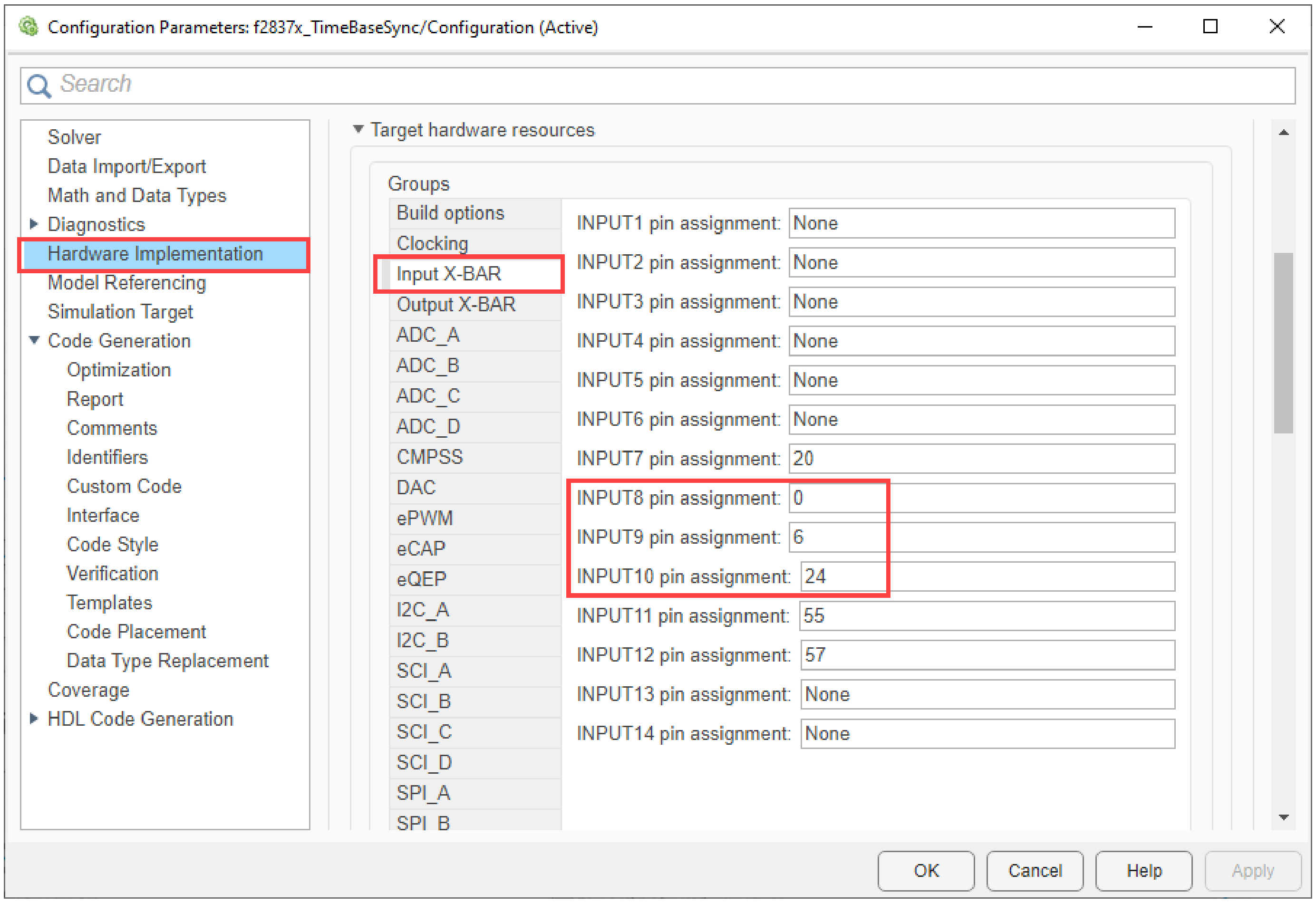
☑ Enable capture event status flag output

Capture flag data type: uint32

☐ Enable overflow status flag output

OK | Cancel | Help | Apply

- The PWM signals (ePWM1 at GPIO0, ePWM4 at GPIO6 and eCAP1 at GPIO24) are given as input to eCAP through Input X-BAR under **Hardware Implementation** > **Target hardware resources** > **Input X-BAR**.


Configuration Parameters: f2837x_TimeBaseSync/Configuration (Active)

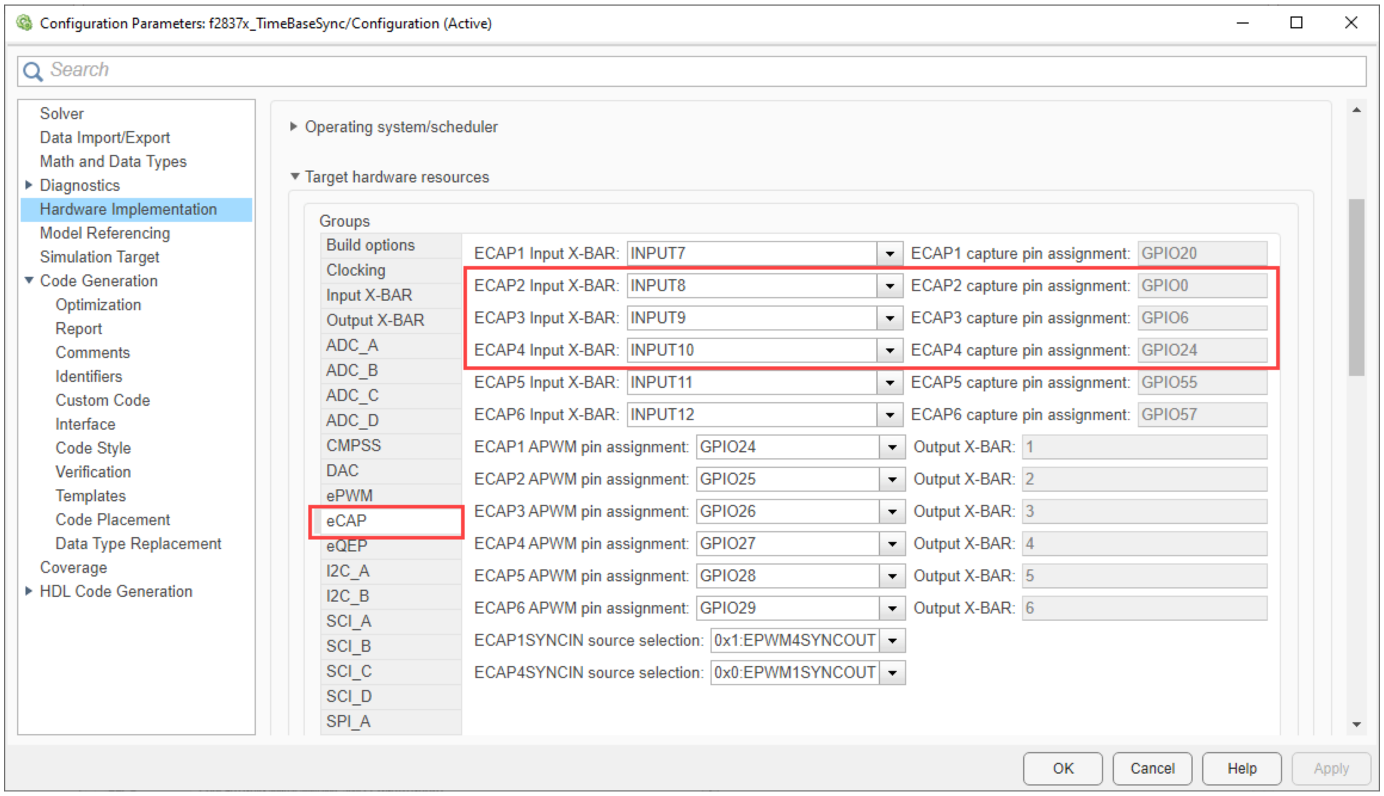
Search

Target hardware resources

Groups

INPUT1 pin assignment:	None
INPUT2 pin assignment:	None
INPUT3 pin assignment:	None
INPUT4 pin assignment:	None
INPUT5 pin assignment:	None
INPUT6 pin assignment:	None
INPUT7 pin assignment:	20
INPUT8 pin assignment:	0
INPUT9 pin assignment:	6
INPUT10 pin assignment:	24
INPUT11 pin assignment:	55
INPUT12 pin assignment:	57
INPUT13 pin assignment:	None
INPUT14 pin assignment:	None

- You can also verify the GPIO signals in Configuration Parameters.


Configuration Parameters: f2837x_TimeBaseSync/Configuration (Active)

ECAP1 Input X-BAR: INPUT7 | ECAP1 capture pin assignment: GPIO20

ECAP2 Input X-BAR: INPUT8 | ECAP2 capture pin assignment: GPIO0

ECAP3 Input X-BAR: INPUT9 | ECAP3 capture pin assignment: GPIO6

ECAP4 Input X-BAR: INPUT10 | ECAP4 capture pin assignment: GPIO24

ECAP5 Input X-BAR: INPUT11 | ECAP5 capture pin assignment: GPIO55

ECAP6 Input X-BAR: INPUT12 | ECAP6 capture pin assignment: GPIO57

ECAP1 APWM pin assignment: GPIO24 | Output X-BAR: 1

ECAP2 APWM pin assignment: GPIO25 | Output X-BAR: 2

ECAP3 APWM pin assignment: GPIO26 | Output X-BAR: 3

ECAP4 APWM pin assignment: GPIO27 | Output X-BAR: 4

ECAP5 APWM pin assignment: GPIO28 | Output X-BAR: 5

ECAP6 APWM pin assignment: GPIO29 | Output X-BAR: 6

ECAP1SYNCIN source selection: 0x1:EPWM4SYNCOUT

ECAP4SYNCIN source selection: 0x0:EPWM1SYNCOUT

Note: The Scope block displays signals only when you specify the time period in seconds. For higher frequency signals, use the digital source oscilloscope (DSO) to view the signals.

Phase Offset Calculation

The counter phase offset value determines the phase lead or lag.

Phase is calculated using the equation

$$\phi = \frac{PRD-CTRPHS}{PRD} * 360$$

where `PRD = Timer period`, `CTRPHS = counter phase offset value`.

Note: This formula is applicable in case of up or down count mode only. In case of up-down counter mode, the phase depends on the mode of the controller as well as the direction after synchronization.

Time delay based on the phase value is given by,

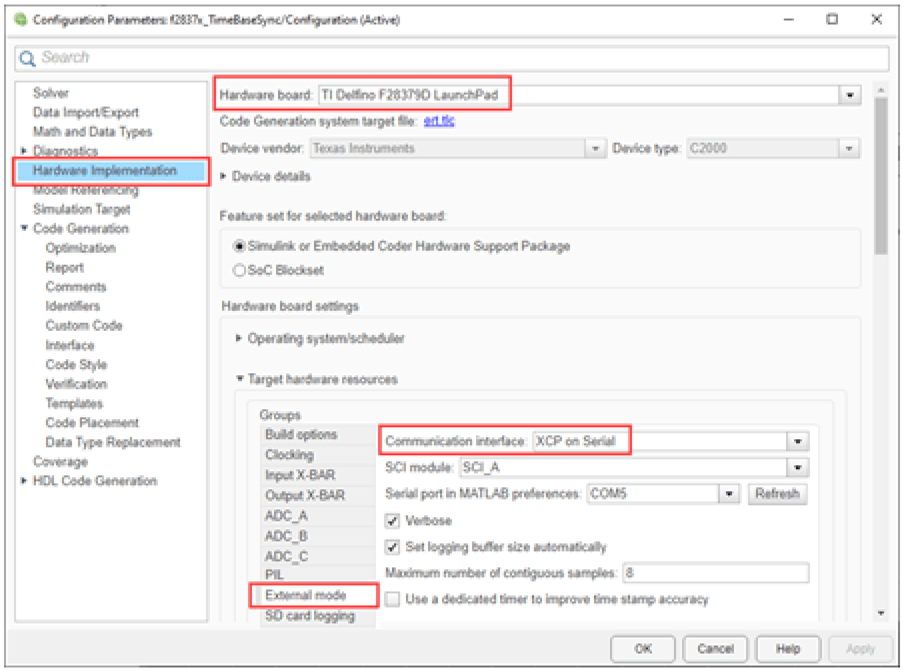
$$\text{Delay} = \frac{\phi}{360*\text{Frequency}}$$

Configure the Model

1. Open the model. The model in this example is configured for TI Delfino F28379D LaunchPad hardware.

2. To run the model on other TI C2000 processors, press **Ctrl+E** to open the Configuration Parameters dialog box and select the required hardware board by navigating to **Hardware Implementation** > **Hardware board**.

3. Ensure that **Communication interface** is set to `XCP on Serial`.


Configuration Parameters: f2837x_TimeBaseSync/Configuration (Active)

Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Feature set for selected hardware board:

(●) Simulink or Embedded Coder Hardware Support Package

( ) SoC Blockset

Communication interface: XCP on Serial

SCI module: SCI_A

Serial port in MATLAB preferences: COM5

☑ Verbose

☑ Set logging buffer size automatically

Maximum number of contiguous samples: 8

☐ Use a dedicated timer to improve time stamp accuracy

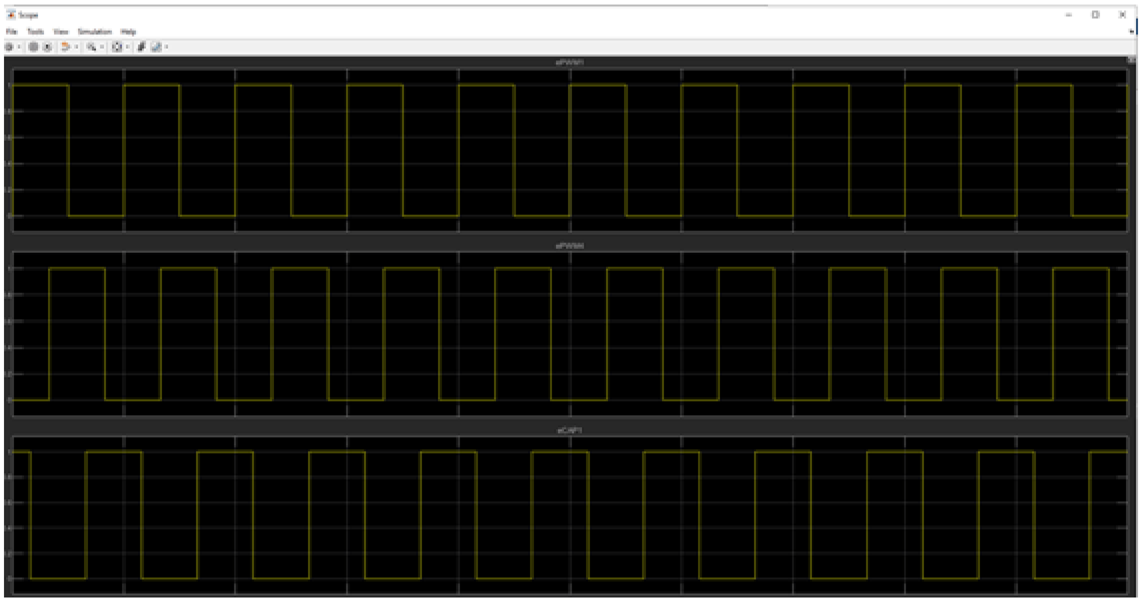
Run the Model

When you perform the **Monitor & Tune** action for the model, the host computer communicates with the target on which the generated executable runs.

1. On the **Hardware** tab of the model click **Monitor & Tune**.

2. Use the diagnostic viewer to follow the build progress and wait until the code loads and runs on the target hardware.

3. Observe the waveforms in the **Scope** block.



Other things to try

- Change the time Period to 20 kHz and monitor the signal in a DSO.
- Try synchronization between other ePWM and eCAP modules.
- Try different phase shifts and lead and lag by providing different counter phase offset value.
- In the Model Properties dialog box, navigate to the **Callbacks** tab > **c28xgetPhaseOffsetValue** to find the file for calculation of phase offset value based on phase delay needed between ePWM and eCAP modules.

More About

- “Overview of Time-Base Synchronization in ePWM Type 4” on page 1-130
- c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM
- “C28x-ePWM”

Parameter Tuning and Signal Logging with Serial Communication

This example shows how to perform parameter tuning and data logging with a Simulink® model running in Texas Instruments™ C2000 targets.

Required Hardware

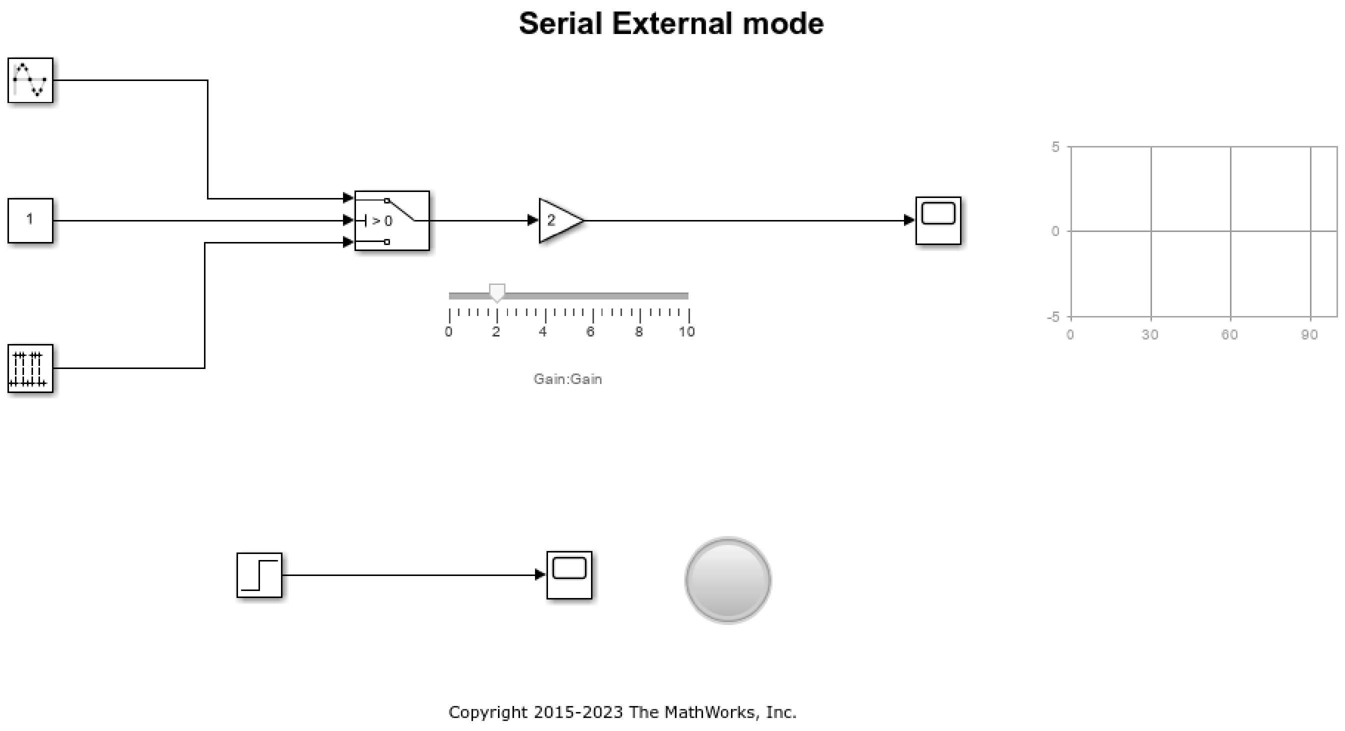
- Any of the Texas Instruments C2000 - based controlCARDS with docking station or Spectrum Digital eZdsp boards that have a serial interface with SCI_A.

Note: F281x based boards do not support parameter tuning over a serial communication interface. Refer to CAN based parameter tuning for these boards.

- A USB serial cable if your hardware provides serial over USB capabilities, or an RS-232 cable connected to the COM1 port of your computer. The CCS tool is closed before running the program in external mode. You can use a tool such as PuTTY to test the basic working of Rx and Tx before trying the external mode. For more, see “Set Up Serial Communication with Target Hardware” on page 1-6

Model

This figure shows the example model. SDI(Simulation Data Inspector) and HMI (Human Machine Interface) blocks in the model will work with External mode over XCP protocol.



Configure the Hardware and Model for Monitoring and Tuning

Set Up the Hardware

This example uses Texas Instruments F28335 controlCARD with docking station and USB serial cable to connect the host computer and the target hardware. You can also use the COM1 port of your computer to establish an RS-232 serial connection with the board. See “Set Up Serial Communication with Target Hardware” on page 1-6 for details on establishing a serial connection between the target and the host computer.

After establishing a serial connection, find the COM port associated with the target hardware:

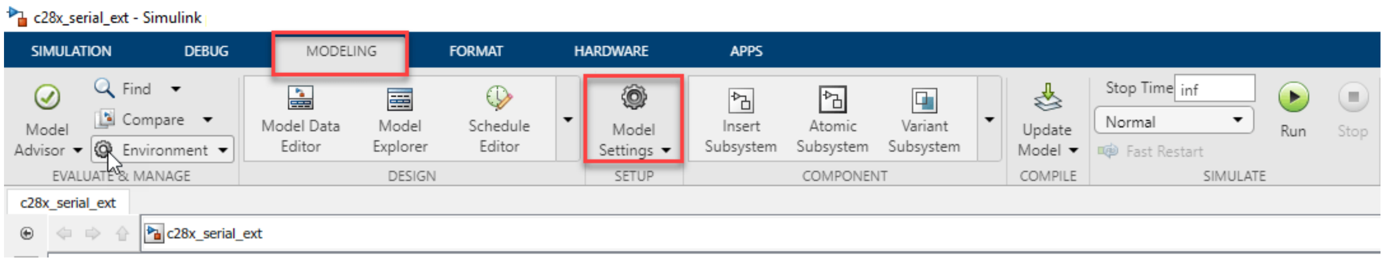
- Open Device Manager in Windows.
- Expand the Ports tab.
- Note the COM port associated with the target board.

Set up the model

The Example Model is configured for the Texas Instruments F28335 controlCARD with docking station, but you can follow the procedure for any other hardware board mentioned in **Required Hardware**.

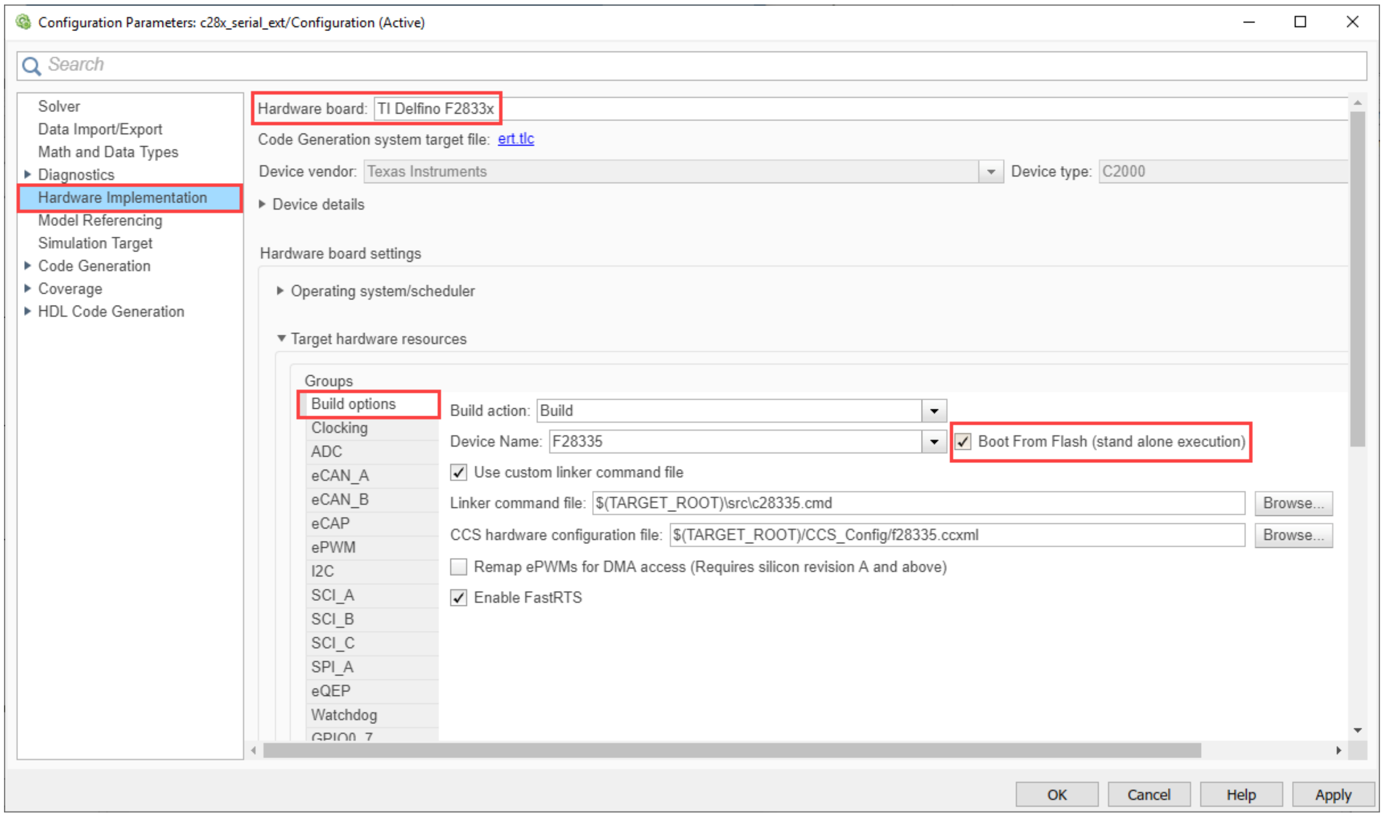
1. Open the Example Model.

2. Open the **Modeling** tab and press **Ctrl+E** to open Configuration Parameters dialog box and navigate to the **Hardware Implementation** pane.



3. Select your target hardware from the **Hardware board** drop-down list.

Note: To run the Example Model on targets with small amounts of RAM such as the F28027 or F28035, enable the **Boot From Flash (stand alone execution)** option in **Hardware Implementation > Target Hardware Resources > Build options** tab and increase the heap size as mentioned in the Limitations section.


Configuration Parameters: c28x_serial_ext/Configuration (Active)

Hardware board: TI Delfino F2833x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments   Device type: C2000

Hardware board settings

Target hardware resources

Groups: Build options

Build action: Build

Device Name: F28335   ✓ Boot From Flash (stand alone execution)

✓ Use custom linker command file

Linker command file: $(TARGET_ROOT)\src\c28335.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28335.ccxml

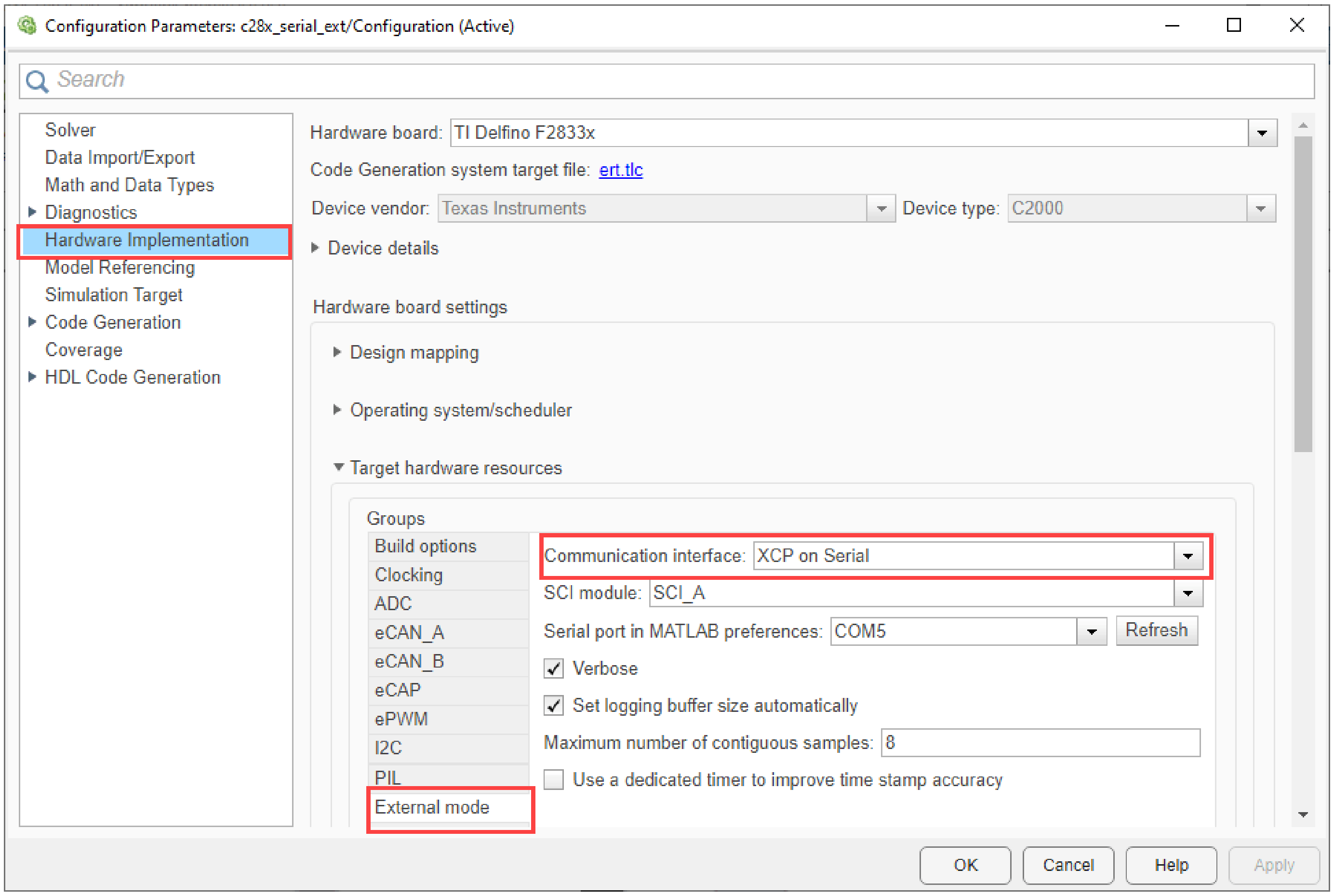
☐ Remap ePWMs for DMA access (Requires silicon revision A and above)

✓ Enable FastRTS

4. Navigate to the **Hardware Implementation** > **Target Hardware Resources** > **External mode**.

- Select the **Communication interface** as **serial** or **XCP on Serial**.
- Select the **SCI module**. By default SCI_A module is selected for Controlcards and Launchpads. For custom boards, select other serial modules to connect to FTDI.
- Select the **Serial port in MATLAB preference** with the COM port number associated with your target hardware.
- The **Verbose** option enables viewing the execution progress of the simulation on the **Diagnostic Viewer** and on the **MATLAB Command Window**.
- Navigate to **SCI_A** and specify the baud rate in **Desired baud rate in bits/sec**.

The default baud rate is `115200`. You can increase the baud of the serial over USB of your Launchpad or controlCARD. On Launchpads and controlCARDs using FTDI 2232H, you can select any baud less than or equal to 6 Mbps, or exactly 9 or 12 Mbps. On controlCARDs using FTDI 2232D, you can select any baud less than or equal to 1.5 Mbps, or exactly 2 or 3 Mbps.



Configuration Parameters: c28x_serial_ext/Configuration (Active)

Hardware board: TI Delfino F2833x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments Device type: C2000

Communication interface: XCP on Serial

SCI module: SCI_A

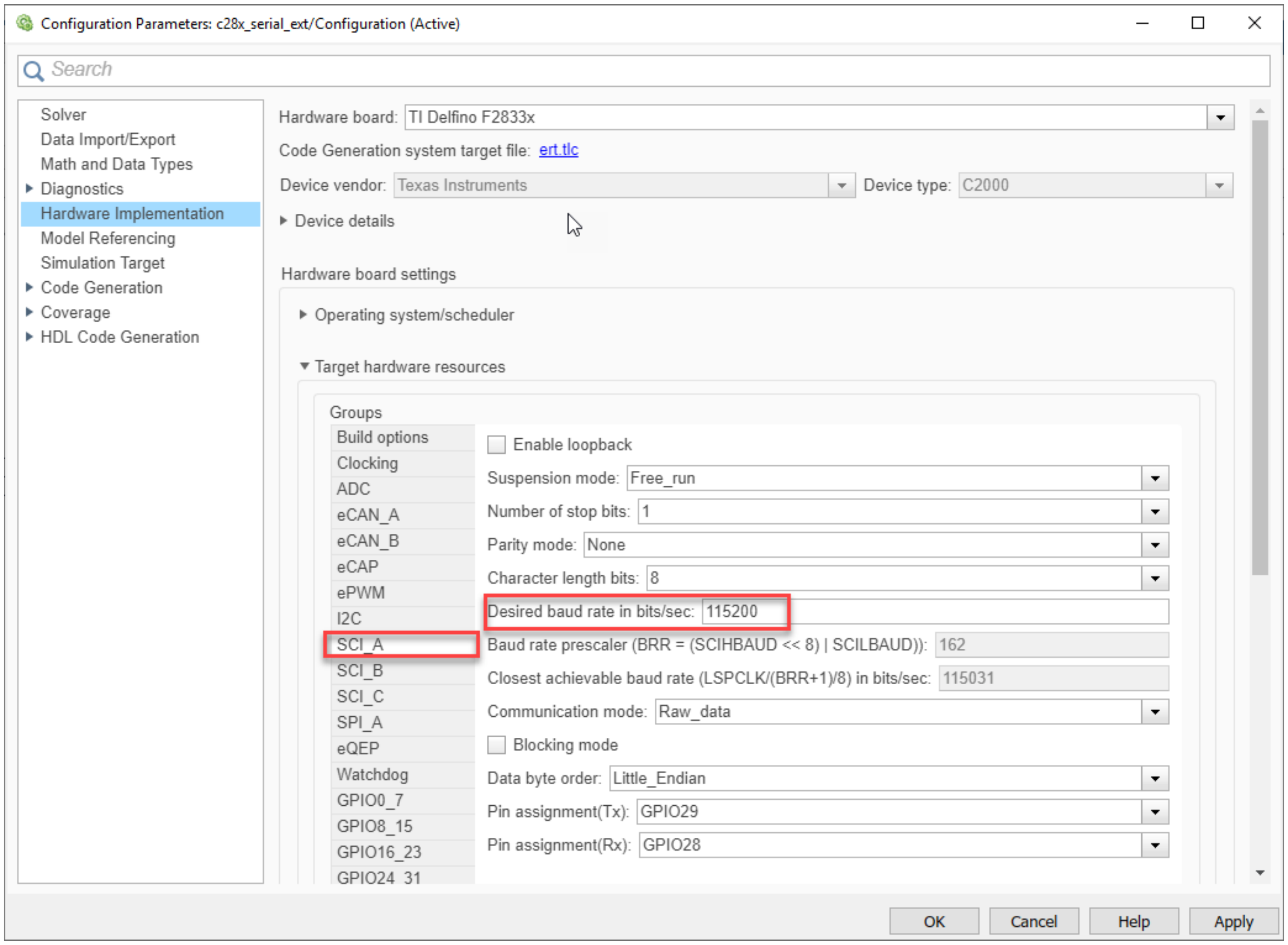
Serial port in MATLAB preferences: COM5

✓ Verbose

✓ Set logging buffer size automatically

Maximum number of contiguous samples: 8

☐ Use a dedicated timer to improve time stamp accuracy


Configuration Parameters: c28x_serial_ext/Configuration (Active)

Hardware board: TI Delfino F2833x
Code Generation system target file: ert.tlc
Device vendor: Texas Instruments
Device type: C2000

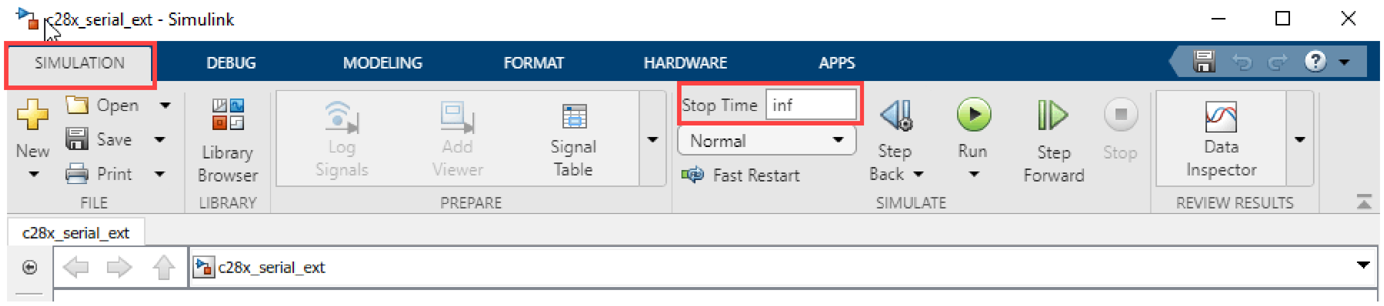
Hardware board settings

Target hardware resources

Groups: SCI_A

Enable loopback	☐
Suspension mode:	Free_run
Number of stop bits:	1
Parity mode:	None
Character length bits:	8
Desired baud rate in bits/sec:	115200
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)):	162
Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec:	115031
Communication mode:	Raw_data
Blocking mode	☐
Data byte order:	Little_Endian
Pin assignment(Tx):	GPIO29
Pin assignment(Rx):	GPIO28

5. Open the **Simulation** tab and specify a value in the **Stop Time** text box. You can specify the stop time as 'inf' to run the model continuously on the target hardware.


c28x_serial_ext - Simulink

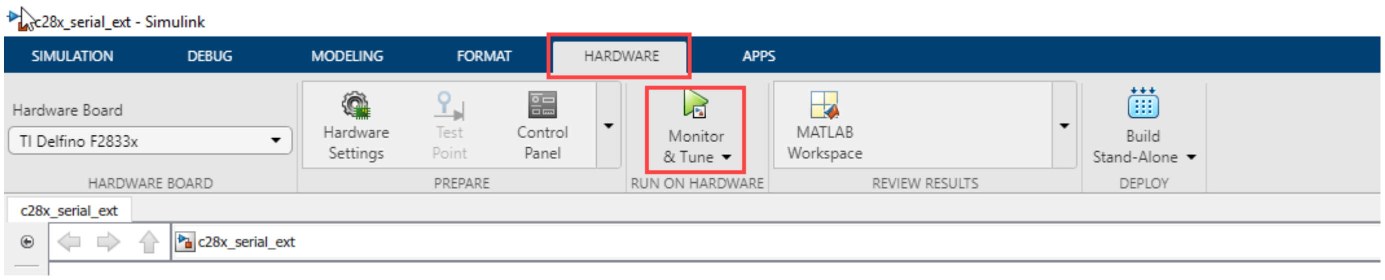
Stop Time: inf

Normal

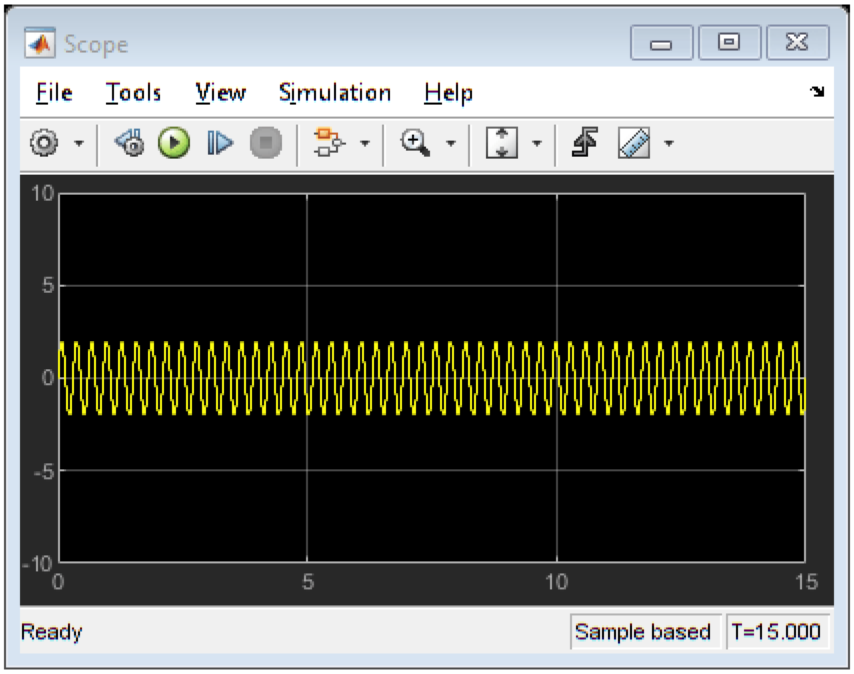
Monitor and Tune the Model

When you perform Monitor and Tune action for the model, the host computer communicates with the target, on which the generated executable runs. To perform Monitor and Tune in the Example Model:

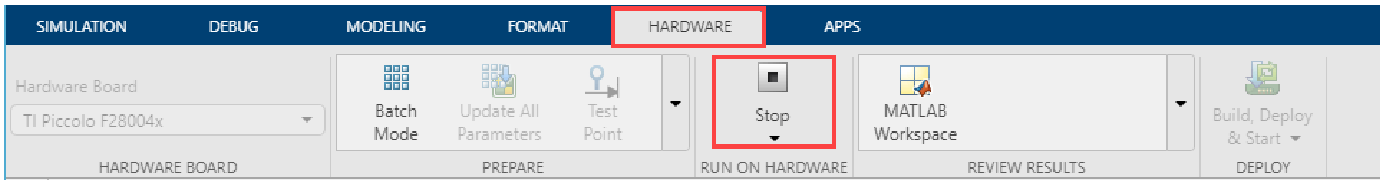
1. Open the **Hardware** tab and click **Monitor & Tune**. You can observe from the **Diagnostic Viewer** that the code is generated for the model and the host connects to the target after loading the generated executable.



2. While the model is running, open the **Scope** connected to the **Gain** block to monitor its output.



3. In the **Hardware** tab, click **Stop** button to terminate the simulation.



Parameter Tuning

You can tune parameter values while the generated executable is running on the target hardware. When the parameter values are changed in the Simulink model, the modified values are communicated to the target hardware.

1. While the Monitor and Tune action is in progress, double click the Gain block and change the value of the gain. You can use slider to change the gain value if using XCP protocol. You can observe that

the amplitude of the sine waveform has changed according to the new gain value. Change the value of the Constant block to `0` to switch the input source and observe the result on the Scope. If the **Verbose** option is selected in **Configuration Parameters > Hardware Implementation > External mode**, the status of the parameter change is displayed on the MATLAB command window.

2. Stop the simulation.

To modify more than one parameter and communicate the changes to the target hardware at once, use the **Batch download** option in the External Mode Control Panel. To open the External Mode Control Panel, go to the **Hardware** tab and click **Control Panel**. Refer to Parameter Downloading for details on the **Batch Download** option.

Data Logging

While the Monitor and Tune action is in progress, you can log data from the model to a file. You can either make use of the Scope or the To Workspace block.

Follow the steps below to manually trigger data logging or trigger data logging from a signal.

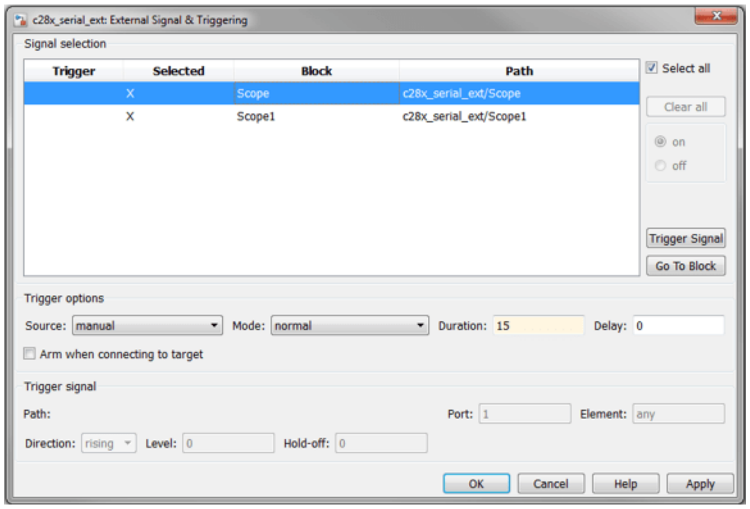
Logging Signals with a Manual Trigger

You can use the **Arm trigger** button on the External Mode Control Panel to trigger data logging. The **Arm when connecting to target** option enables the trigger automatically when the host connects to the target and data uploading begins. Otherwise, to start uploading the data, you must manually arm the trigger by clicking the **Arm trigger** button located in the External Mode Control Panel. Triggers can be useful when the communication channel speed does not allow live visualization of the desired signals. In this task, you will learn how to manually trigger data uploading from the target to the host computer.

1. Open the External Mode Control Panel and click on the **Signal & Triggering** button, which opens the External Signal & Triggering dialog box.

a. By default, the **Source** option is set to `manual`. Set the **Mode** to `normal` to collect contiguous data samples. External mode allocates sufficient memory to collect data for the length of the **Duration** for each signal. Depending on the communication speed and the processing time given to the External mode engine that is running as a background task, you may see a continuous data logging stream, or a stream of data containing gaps corresponding to the time required to send the acquired buffers.

b. Specify the **Duration** as `15` to collect 15 data samples for a base rate signal. Since the signal from the `Gain` block has the same sample rate as that of the model, the number of samples collected in each data set is 15. Uncheck the **Arm when connecting to target** option to enable manual triggering to upload data.



Signal selection

Trigger	Selected	Block	Path
	X	Scope	c28x_serial_ext/Scope
	X	Scope1	c28x_serial_ext/Scope1

Trigger options

Source: manual Mode: normal Duration: 15 Delay: 0

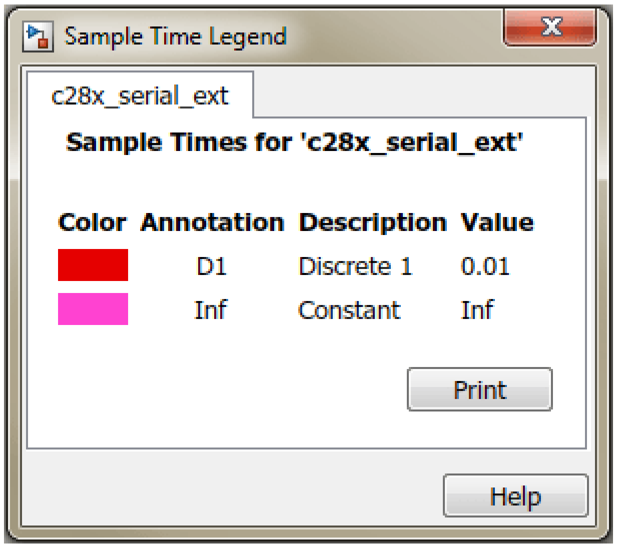
Arm when connecting to target

Trigger signal

Path: Port: 1 Element: any

Direction: rising Level: 0 Hold-off: 0

2. To view the sample time of the model, navigate to **Display** > **Sample Time** and select **Sample Time Legend**. For this example model, the sample time is 0.01 s.

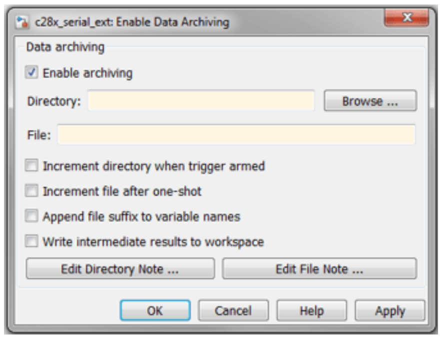


Sample Times for 'c28x_serial_ext'

Color	Annotation	Description	Value
🟥	D1	Discrete 1	0.01
🟪	Inf	Constant	Inf

3. Click on the **Data Archiving** button of the External Mode Control Panel to enable logging data to a MAT file from the Enable Data Archiving dialog box.

4. Check the **Enable archiving** option. Use the **File** and **Directory** parameters to specify the destination of your log file. In normal mode, file name incrementing happens automatically and new data sets are saved in new MAT files.


c28x_serial_ext: Enable Data Archiving

Data archiving

[✓] Enable archiving

Directory:

File:

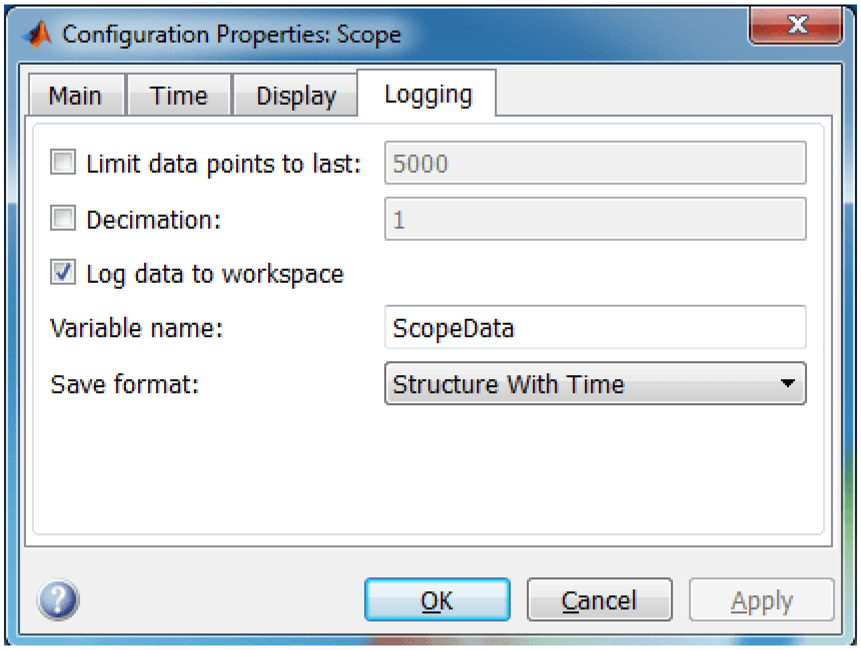
[ ] Increment directory when trigger armed

[ ] Increment file after one-shot

[ ] Append file suffix to variable names

[ ] Write intermediate results to workspace

5. Open the Scope connected to the Gain block and click on the **Configuration Properties** button. Navigate to the **Logging** tab and check the **Log data to workspace** option. Specify a name in **Variable name**. The logged data is saved in this variable. To save the time instant and the signal data values, select `Structure With Time` for **Save format**.


Configuration Properties: Scope

Logging

[ ] Limit data points to last: 5000

[ ] Decimation: 1

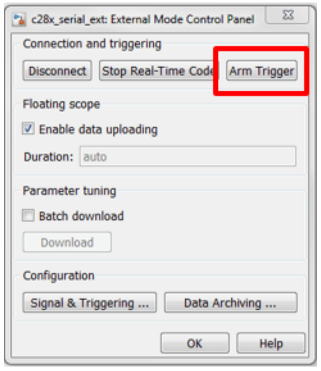
[✓] Log data to workspace

Variable name: ScopeData

Save format: Structure With Time

Note: If you do not select the **Save data to workspace option**, the MAT files for data logging are created, but they will be empty.

6. Click **Monitor & Tune**. Click the **Arm Trigger** button on the External Mode Control Panel to trigger data logging and the **Cancel Trigger** button to stop logging the data. Navigate to the folder specified for saving the logged data. Several MAT files are in this location, each containing a structure with 15 contiguous data samples.



[✓] Enable data uploading

Duration: auto

[ ] Batch download

7. Press the **Stop** button to terminate the simulation.

To collect only a single set of contiguous data samples, select `One-shot`. In this mode, check the option **Increment file after one-shot** in the Enable Data Archiving dialog box to save new data sets in new MAT files.

Logging Signal Data using a Signal Trigger

In signal trigger mode, Monitor and Tune uses a signal as a trigger to start logging data. The data uploading begins when the trigger event occurs. To analyze a signal when an error or fault condition occurs while the model is running, use a signal trigger to log data.

In this task, you will learn how to trigger data logging to a file when certain signal conditions are met. In signal trigger mode, depending on the **Delay** specified, you can choose to log data samples of a signal immediately at trigger, a few seconds after the occurrence of the trigger, or both before and after the occurrence of the trigger.

1. Open the External Signal & Triggering dialog box from the External Mode Control Panel.

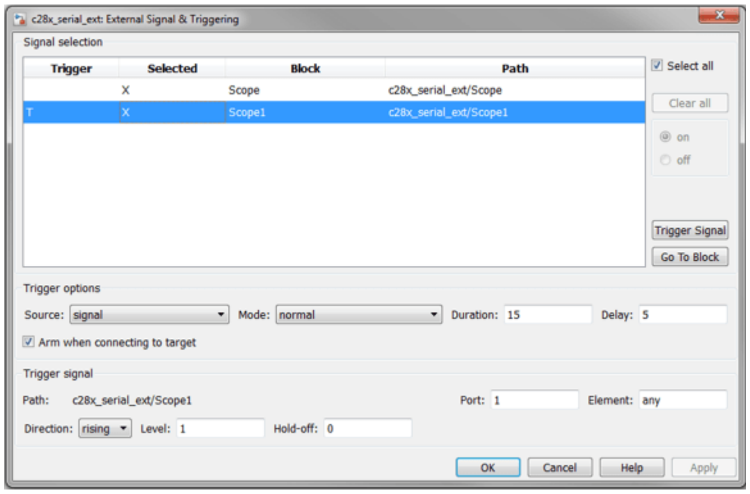
2. In this dialog box, select the Scope block connected to the **Step** input and click the **Trigger Signal** button to set the selected block signal as the trigger signal.

a. Select `signal` as the trigger **Source**. This enables the options in the **Trigger signal**. Set the **Mode** to `normal` to collect contiguous data samples.

b. Specify the **Duration** as `15` to collect 15 base rate samples whenever the trigger conditions are satisfied.

c. Specify the **Delay** as `5`. This causes data logging to begin 5 base rate samples after the occurrence of the signal trigger. Because the sample time for this model is 0.01 s and the step input is applied at the time instant 26 s, data logging begins from the time instant 26.05 s , (i.e 5 * 0.01 s after the trigger event at 26 s).

d. In the **Trigger signal** section, choose `rising` as the **Direction** and set **Level** to `1`. This causes the signal trigger conditions to be met whenever the signal connected to Scope1 increases in magnitude and crosses the threshold value of 1.


c28x_serial_ext: External Signal & Triggering

Signal selection

Trigger	Selected	Block	Path
	X	Scope	c28x_serial_ext/Scope
T	X	Scope1	c28x_serial_ext/Scope1

☑ Select all

Trigger options

Source: signal   Mode: normal   Duration: 15   Delay: 5

☑ Arm when connecting to target

Trigger signal

Path: c28x_serial_ext/Scope1   Port: 1   Element: any

Direction: rising   Level: 1   Hold-off: 0

3. Specify a folder and a file name in the Enable Data Archiving dialog box to save the logged data.

4. Click **Monitor & Tune**.

5. The trigger conditions are met at the time instant 26 s, so stop the simulation after, about 40 s. Navigate to the folder where the data is saved and load the MAT file into MATLAB workspace by double-clicking on it. If you examine the contents of the MAT file, the structure member `time` indicates, that data is logged from the time instant 26.05 s.

If the **Delay** `0`, logging of the signal data begins immediately when the signal trigger conditions are met. The **Delay** option can also be used with negative values to center the data acquisition around the event of interest. To log data samples before the trigger conditions occur for the Example Model, specify a negative value, `-8` for example in the **Delay** field. This captures 8 base rate samples before the occurrence of the trigger and the remaining data samples after the occurrence of trigger.

To collect a data set of base rate samples only when the signal trigger conditions are met for the first time, choose the **Mode** as `One Shot`. This mode is useful for viewing fast-changing data on a slow communication channel. To achieve this, increase the value of the **Duration** and set the signal trigger to acquire relevant data based on an event of interest, like a fault signal, an over voltage/current signal or a temperature warning signal.

Logging Data in SDI

XCP supports logging data in Simulation Data Inspector. For more, see “Save and Share Simulation Data Inspector Data and Views”

Troubleshooting

If you observe breaks in the scope, you can try the following to improve logging:

- Increase `Serial Baud Rate`
- Decrease the number of signals selected for logging
- Decrease the model base rate

More About

- “Signal Monitoring and Parameter Tuning over XCP on Serial” on page 1-59
- “Set Up Serial Communication with Target Hardware” on page 1-6

Serial Data Communication on F28M3x Concerto between ARM, C28x and the Host Computer

This example shows you how to use C2000™ Microcontroller Blockset for serial data exchange between the host computer and the target as well as serial data exchange between the ARM and c28x core.

In this example you will learn how to perform data exchange between the host and the target as well as between the ARM Cortex M3 and the C28x core using the serial internal loopback that exists between the two cores of the F28M3x Concerto processor.

The F28M3x Concerto processors supports data exchange between the ARM Cortex M3 core and the C28x Core by means of a UART4 to SCI-A Internal loopback which is physical connection between the 2 peripherals. The UART4 peripheral on the ARM Cortex M3 Core and SCI-A peripheral on the C28x core need to be configured to run on the same baud rate in both the models as shown below. The Internal loopback between the UART4 AND SCI-A can be enabled from the ARM Model by browsing to **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources** > **UART4**. Therefore, ensure that the UART4 baud rate matches the C28x SCI-A baud rate for internal loopback.

Prerequisites

We recommend completing the “Getting Started with C2000 Microcontroller Blockset for F28M3x Concerto Processors” on page 4-287.

Required Hardware

To run this example you will need the following hardware:

- F28M36 Concerto controlCARD or
- F28M35 Concerto controlCARD

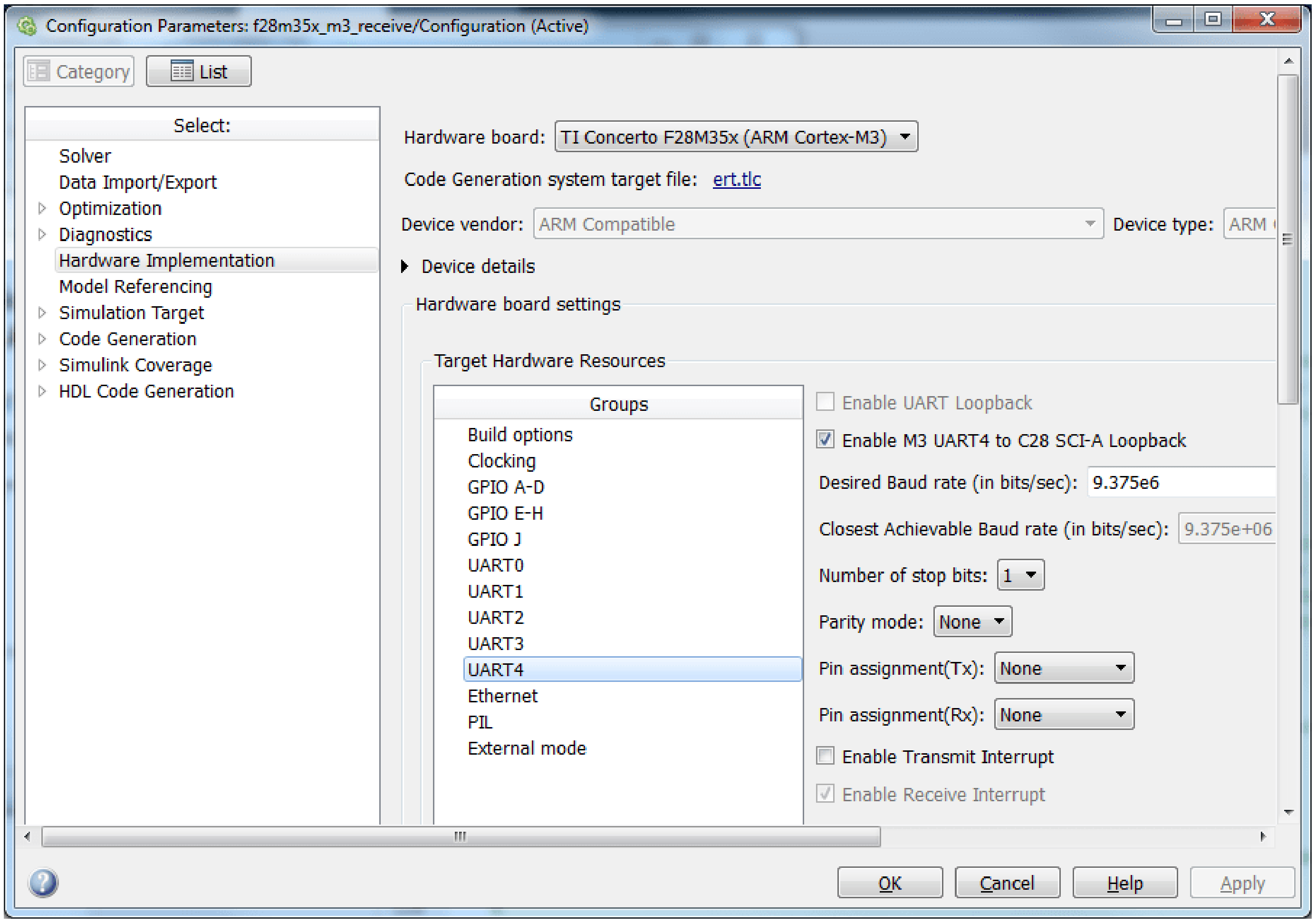
The Texas Instruments ControlCARDs provide serial over USB capabilities. This allows serial communication from the target to your host computer over the USB connection to the board.

ARM Cortex M3 UART4 to C28x SCI-A Internal Loopback

In this task, you will run a model on the C28x Core of the Concerto processor which sends data over the serial internal loopback to a model running on the ARM Cortex M3 Core. Every 1ms, 6 bytes are transmitted from C28x SCI-A to the ARM UART4 via the internal loopback. The data consists of 1 byte for packet header, 2 bytes for a 16-bit ADC value, 2 bytes for a 16-bit free-running counter and 1 byte for packet terminator. The ARM model is configured to generate an interrupt once 6 bytes are received. The UART4 serial receive block in the ARM model outputs the ADC and free-running counter values if a matching header and terminator are found. In the case where a header and terminator are not found at the right location, the block outputs values of 0 and the interrupt is resynchronized based on potential header and terminator found in the packet. In the interrupt service routine, the ADC data and the free-running counter values are sent to UART0 which is connected to the host computer. Task 2 explains this step.

1. Open the ARM UART model. This model is configured for the **TI Concerto F28M35x (ARM Cortex-M3)** target. To configure the model to run on **TI Concerto F28M36x (ARM Cortex-M3)**, change the Hardware board in the Configuration Parameters > Hardware Implementation pane.

2. The Internal Loopback is enabled in Configuration Parameters -> Hardware Implementation -> Target Hardware Resources -> UART4 by setting the **Enable M3 UART4 to C28 SCI-A loopback**. This parameter, when enabled, physically connects UART4 on the ARM core to SCIA on the C28x core and allows serial communication between the 2 cores. The Internal loopback configuration is controlled by the ARM model and doesn't require specific settings in the C28x model. The internal loopback, being a physical connection, does not need any GPIO for communication. This enables you to select the GPIOs to None for Transmit and Receive. This allows other peripherals to use these pins.


Configuration Parameters: f28m35x_m3_receive/Configuration (Active)

Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

Device type: ARM

Hardware board settings

Target Hardware Resources

- [ ] Enable UART Loopback
- [✓] Enable M3 UART4 to C28 SCI-A Loopback

Desired Baud rate (in bits/sec): 9.375e6

Closest Achievable Baud rate (in bits/sec): 9.375e+06

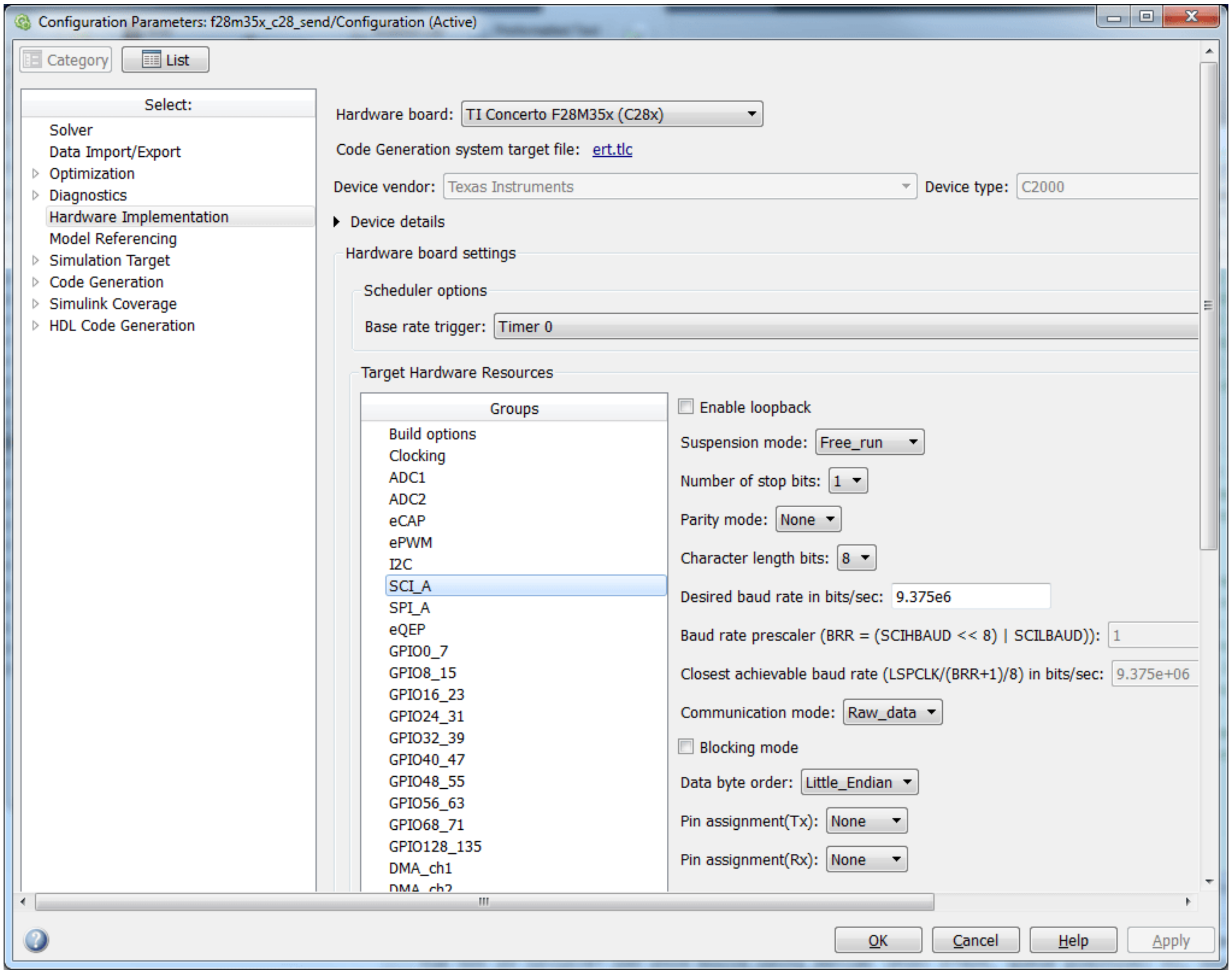
Number of stop bits: 1

Parity mode: None

Pin assignment(Tx): None

Pin assignment(Rx): None

- [ ] Enable Transmit Interrupt
- [✓] Enable Receive Interrupt


Configuration Parameters: f28m35x_c28_send/Configuration (Active)

Hardware board: TI Concerto F28M35x (C28x)

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments

Device type: C2000

Hardware board settings

Scheduler options

Base rate trigger: Timer 0

Target Hardware Resources

Groups: Build options, Clocking, ADC1, ADC2, eCAP, ePWM, I2C, SCI_A, SPI_A, eQEP, GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, GPIO56_63, GPIO68_71, GPIO128_135, DMA_ch1

Enable loopback	☐
Suspension mode:	Free_run
Number of stop bits:	1
Parity mode:	None
Character length bits:	8
Desired baud rate in bits/sec:	9.375e6
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)):	1
Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec:	9.375e+06
Communication mode:	Raw_data
Blocking mode	☐
Data byte order:	Little_Endian
Pin assignment(Tx):	None
Pin assignment(Rx):	None

3. Configure the baud rate of the UART4 peripheral and ensure it matches the baud rate of the C28x Model SCI-A peripheral. The internal loopback allows high serial baud rates. In these example models, the baud rates are set to 9.375 Mbps, which is the maximum achievable value by the hardware.

4. Click Apply and close the configuration parameters window.

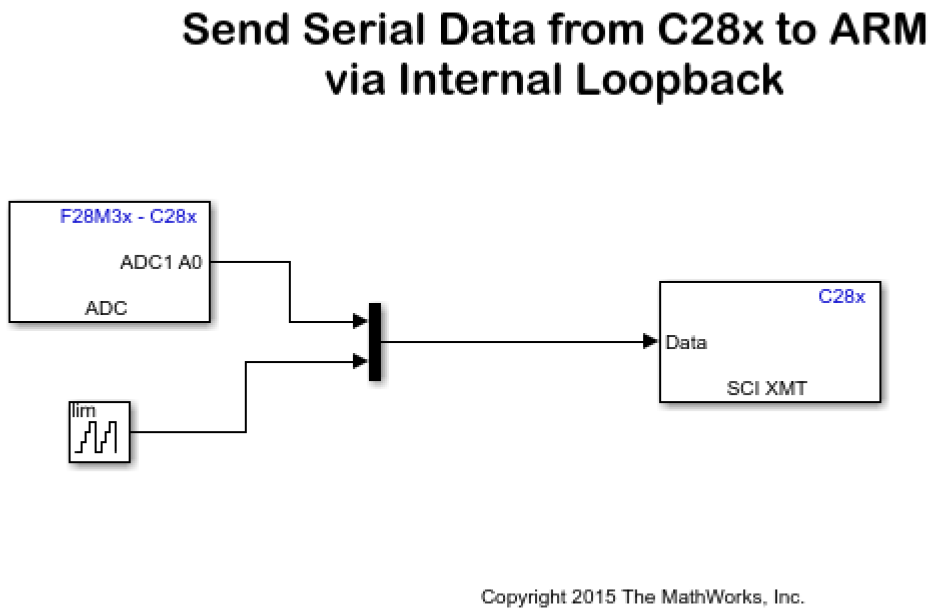
5. Open the C28x SCI model. This model is configured for the **TI Concerto F28M35x (C28x)** target. To configure the model to run on **TI Concerto F28M36x (C28x)**, change the Hardware board in the Configuration Parameters > Hardware Implementation pane. There are no settings for the internal loopback required on the C28x model.

6. Match the UART4 Closest Achievable Baud rate by selecting an appropriate desired baud rate for the SCI-A peripheral. 9.375 Mbps is used in this example.

7. Click Apply and close the configuration parameters window.

8. Build, load and run both the models. You can monitor the sent data through Task 2 and verify that the models are running correctly.

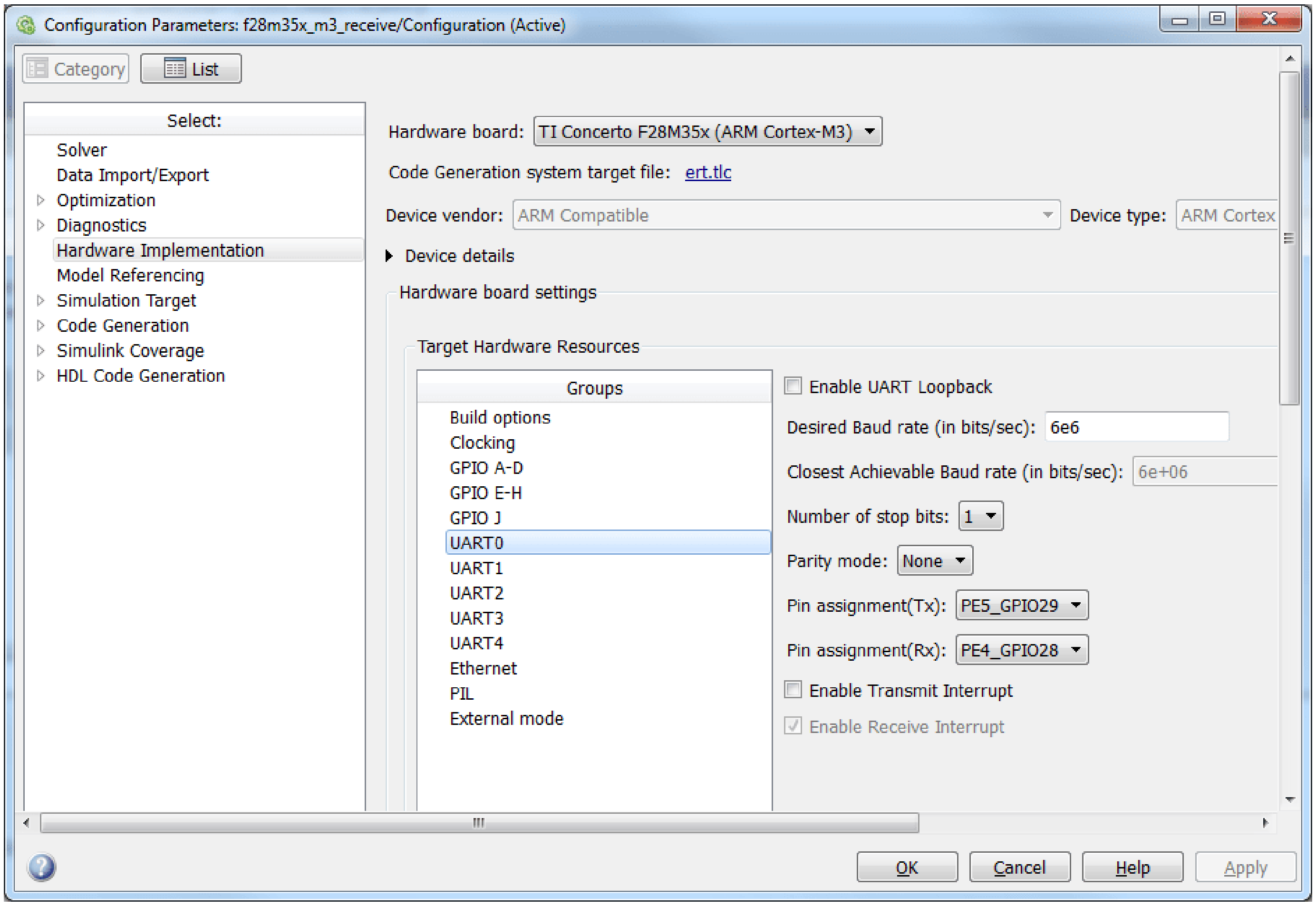
```
open_system('f28m35x_c28_send');
```


Send Serial Data from C28x to ARM via Internal Loopback
F28M3x - C28x
ADC1 A0
ADC
lim
C28x
Data
SCI XMT
Copyright 2015 The MathWorks, Inc.

Sending Data to the Host Computer

The Model running on the ARM Core transmits data to the host computer via UART0. You can access UART0 data on your host computer, via the USB cable connected to the controlCARD through a virtual COM port. For more information on how to configure the virtual COM port, refer to this page. Note the virtual COM port number of the USB Serial Port showing in your Windows Device Manager under Ports "(COM & LPT)", you will need it to configure your host receive program.

1. The ARM model sends data to the host via UART0 peripheral every 1ms. The turnaround time of Simulink does not allow code running on the host computer to receive data at that rate. The host computer can work on large data sets coming at a slower pace. The ARM model is constructed to add a header and terminator every 60ms. The host computer program will expect 244 bytes to be present in the serial buffer every 60ms. This technique works best with Simulink.


Configuration Parameters: f28m35x_m3_receive/Configuration (Active)

Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

Device type: ARM Cortex

Hardware board settings

Target Hardware Resources

Groups: UART0

Enable UART Loopback: ☐

Desired Baud rate (in bits/sec): 6e6

Closest Achievable Baud rate (in bits/sec): 6e+06

Number of stop bits: 1

Parity mode: None

Pin assignment(Tx): PE5_GPIO29

Pin assignment(Rx): PE4_GPIO28

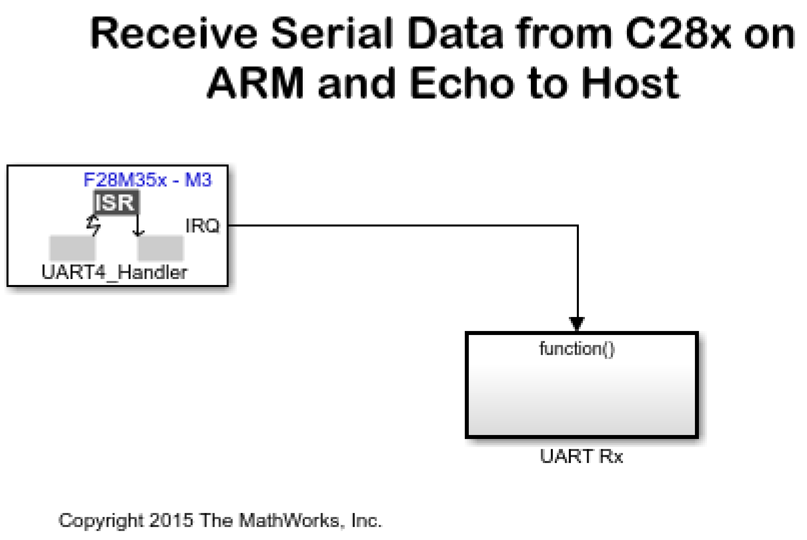
Enable Transmit Interrupt: ☐

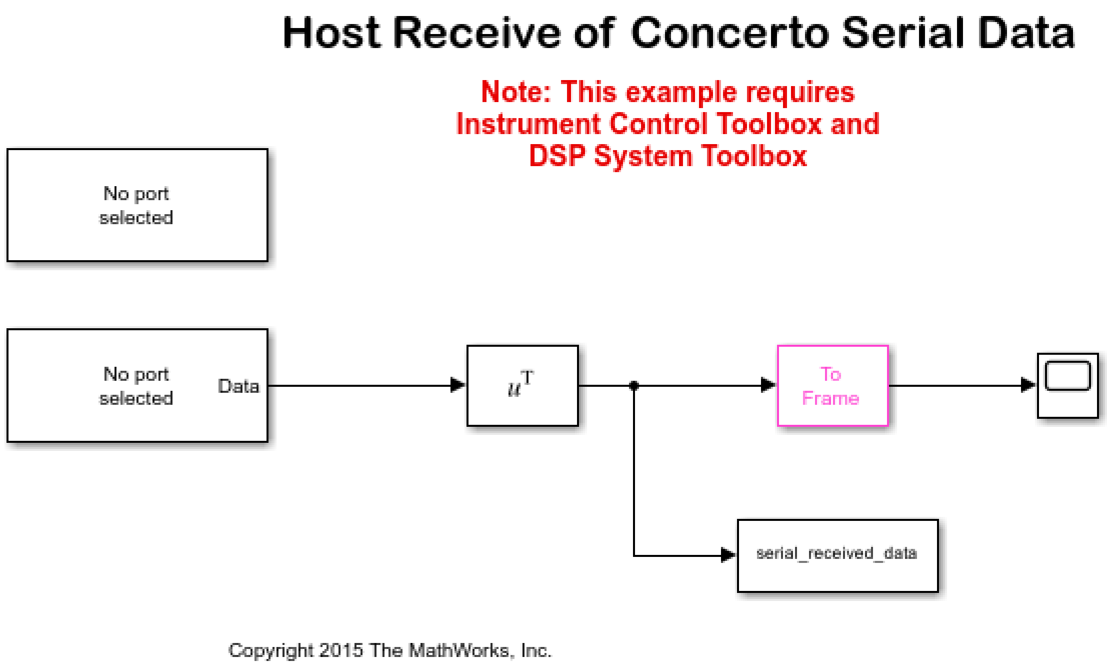
Enable Receive Interrupt: ✓

2. The serial over USB capability of the Concerto controlCARDs allows a maximum baud rate of 6Mbps. You can configure the baud rate and the GPIOs used for the serial connection on the UART0 pane of the Configuration Parameters -> Hardware Implementation tab. This example model uses a baud rate of 6Mbps along with PE5_GPIO29 and PE4_GPIO28 for transmit and receive respectively over UART0. These pins are connected to the serial over USB device present on the controlCARD.

3. You can read and plot the data using the Host Model. This model requires DSP System Toolbox and Instrument Control Toolbox. On the host model, enter the virtual COM port number and the baud rate in the Serial Configuration block. In the Serial Receive block, match the COM port number with the virtual COM port number. The Header and Terminator match with values specified in the ARM UART model. Note that the length and the sample time will ensure that the simulation will get a new frame every 60ms, although the ARM UART model sends a fresh data point every millisecond. This ensures that the simulation turnaround time is sufficient on the host. While Simulink can easily get one frame of 244 bytes every 60ms, it can not turnaround at a 1ms rate on a host computer and get one set of samples at a time. The model creates a variable named serial_received_data in the workspace that you can use to view and analyze the data. Note that MATLAB is set to use row-major order while the target uses column-major order which explains the need to transpose the data received with the Serial Receive block.

```
open_system('f28m35x_m3_receive');
open_system('Host_read_f28m35');
```


Receive Serial Data from C28x on ARM and Echo to Host
F28M35x - M3
ISR
IRQ
UART4_Handler
function()
UART Rx
Copyright 2015 The MathWorks, Inc.


Host Receive of Concerto Serial Data
Note: This example requires Instrument Control Toolbox and DSP System Toolbox
No port selected
No port selected
Data
u^T
To Frame
serial_received_data
Copyright 2015 The MathWorks, Inc.

Things to remember while setting up the model

Make sure that the closest achievable baud rate of UART 4 peripheral in the ARM Cortex M3 Model and in the SCI-A peripheral on the C28x model are matching.

```
close_system('f28m35x_m3_receive',0);
close_system('f28m35x_c28_send',0);
close_system('Host_read_f28m35',0);
```

Exchanging Ethernet Data with the F28M3x Concerto Processor

This example shows you how to use C2000™ Microcontroller Blockset to send and receive UDP and TCP messages using the ARM Cortex-M3 core of F28M3x Concerto processor.

In this example you will learn how to use TCP/IP Send, TCP/IP Receive, UDP Send and UDP Receive blocks to send and receive Ethernet data with the F28M3x Concerto processor.

The Ethernet port on the Texas Instruments C2000 Concerto processors supports the IEEE 802.3 standard.

Prerequisites

We recommend completing the “Getting Started with C2000 Microcontroller Blockset for F28M3x Concerto Processors” on page 4-287.

Required Hardware

To run this example you will need the following hardware:

- F28M36 Concerto controlCARD or
- F28M35 Concerto controlCARD
- RJ45 Ethernet cable.
- Optional router for automatic IP assignment.

The F28M3x Concerto controlCARDs provide an Ethernet port. This port can be used for data communication using the uIP TCP/IP stack provided by TI ControlSuite.

Hardware and Network Configuration

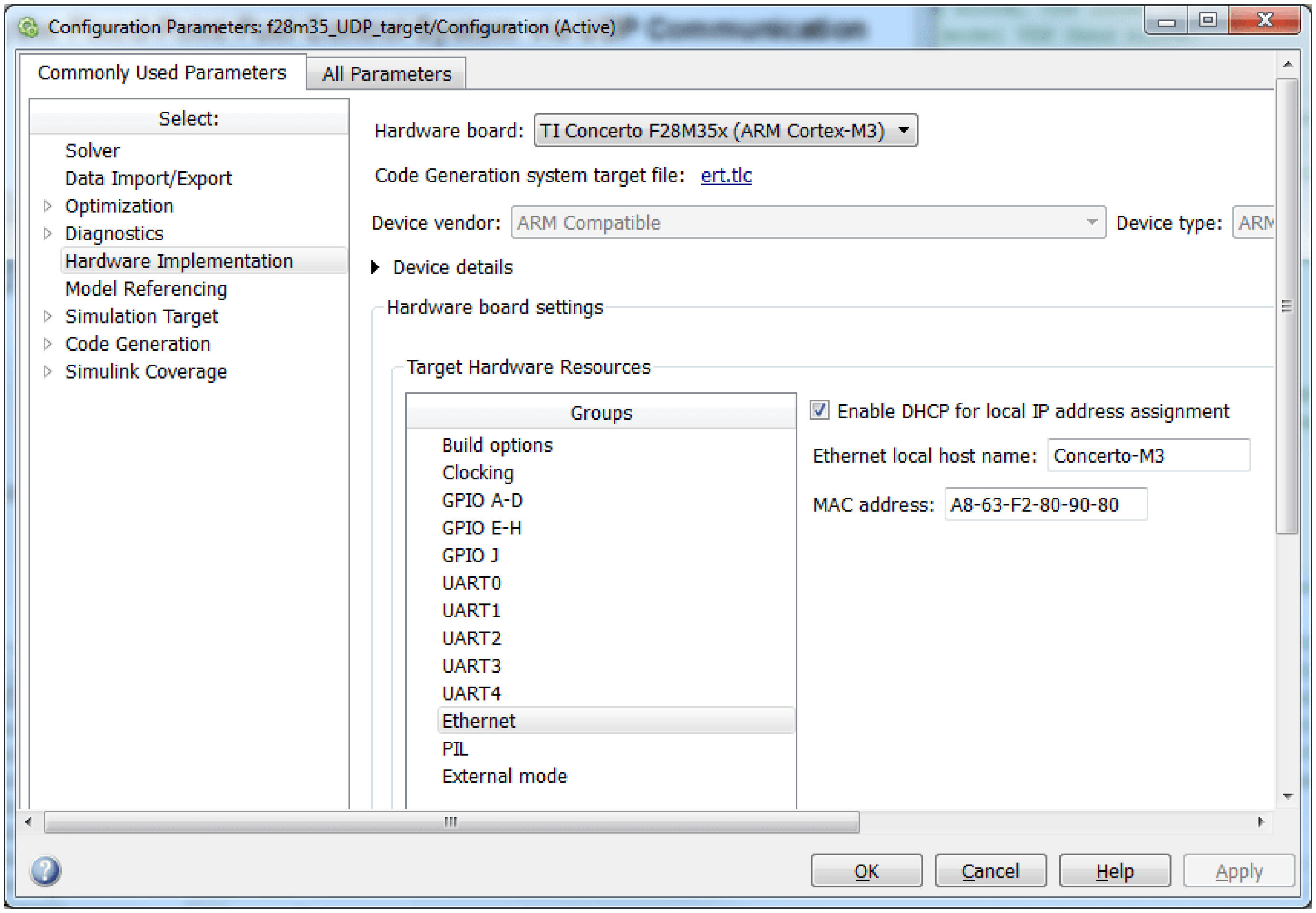
In this task you will configure the hardware and network properties needed to run this example.

- When using the F28M35 Concerto ControlCARD, ensure rows 1 to 15 of the connectivity MUX connector are shorted between the B and C lines using jumpers. This step routes the Ethernet pins to the ARM Cortex M3 core of the F28M35x Concerto processor. This step doesn't apply to the F28M36 controlCARD.

- The default Configuration of the target models present in this example are configured to get an IP address at startup using DHCP. This workflow is compatible when both the host computer and the target hardware are connected to a router. If you wish to connect the board straight into your host computer with the Ethernet cable, assign the F28M3x board with a static IP address. To change the Ethernet settings, browse to **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources** > **Ethernet**. To choose a static IP address, uncheck the **Enable DHCP for local IP address assignment**. This will prompt you to enter the static IP and subnet mask. Ensure that the IP address and subnet mask are unique and appropriate.

Settings for Automatic IP Address Assignment using DHCP:

- Connect the controlCARD with a network cable to a router for automatic IP address assignment. In this configuration, the host computer also needs to be connected to the same router. This can be done with a network cable or over WiFi.


Configuration Parameters: f28m35_UDP_target/Configuration (Active)

Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible Device type: ARM

Device details

Hardware board settings

Target Hardware Resources

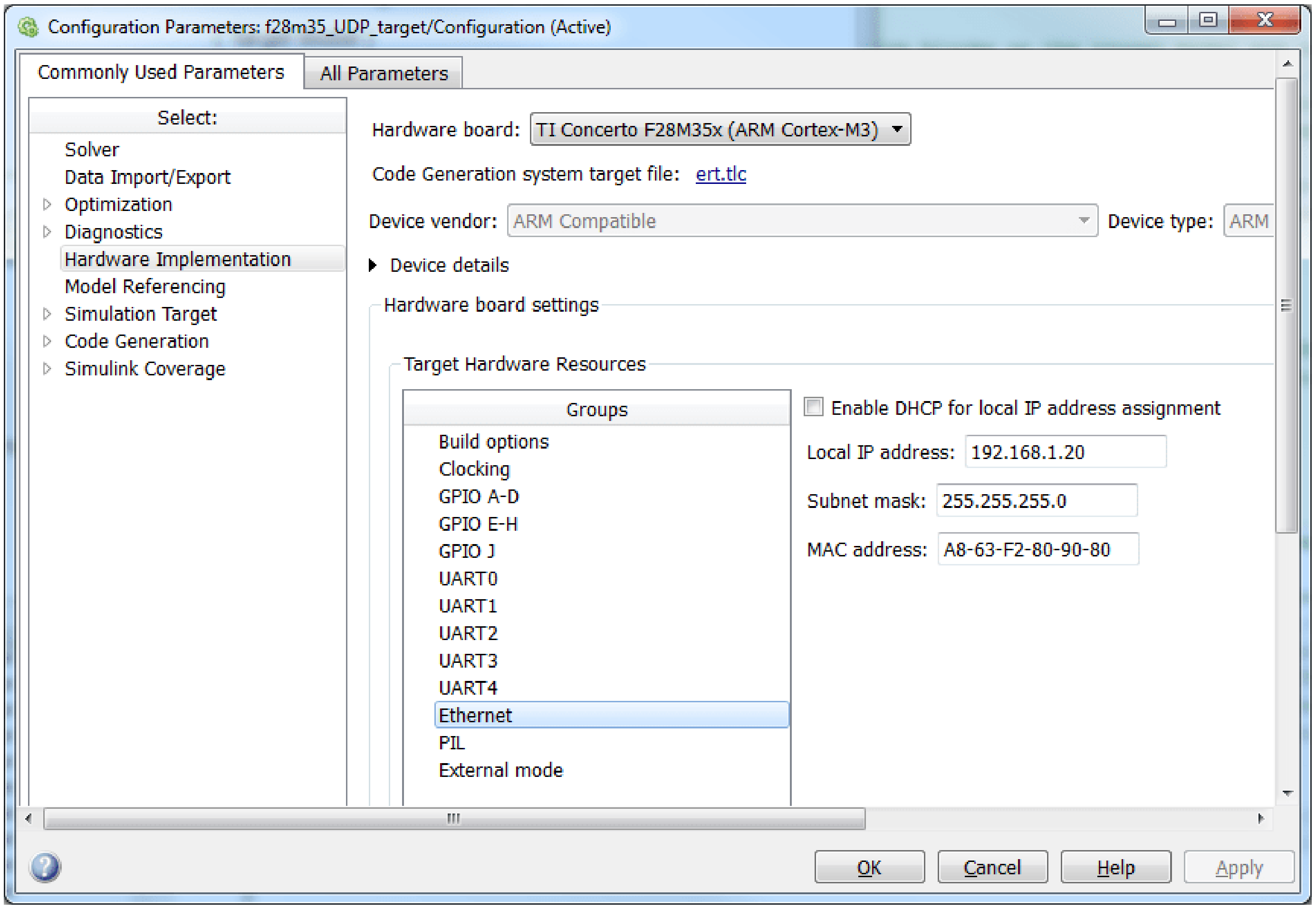
[✓] Enable DHCP for local IP address assignment

Ethernet local host name: Concerto-M3

MAC address: A8-63-F2-80-90-80

Ethernet Settings for Static IP Address Assignment:

- Connect the controlCARD with a network cable to the host computer. In this configuration, set the controlCARD with a static IP address.


Configuration Parameters: f28m35_UDP_target/Configuration (Active)

Hardware board: TI Concerto F28M35x (ARM Cortex-M3)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible

Device type: ARM

Hardware board settings

Target Hardware Resources

Groups: Ethernet

Enable DHCP for local IP address assignment: unchecked

Local IP address: 192.168.1.20

Subnet mask: 255.255.255.0

MAC address: A8-63-F2-80-90-80

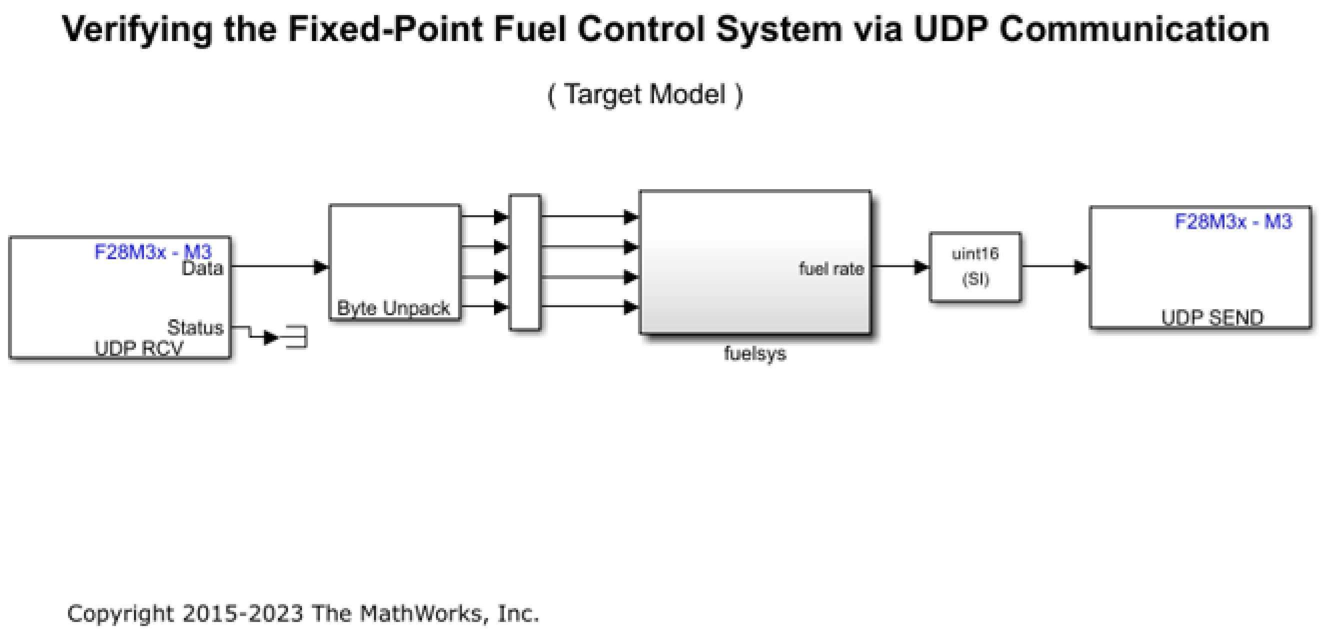
UDP Communication Between the Target and the Host Computer

In this task, you will run the Modeling a Fault-Tolerant Fuel Control System on the ARM Cortex M3 Core of the Concerto processor. The control algorithm runs on the target while the host runs a plant simulation of the system. The target and the host communicate via UDP. The target model receives the fuel system control sensors values from the host model and transmits back the calculated fuel mixture ratio to the host computer.

Models Involved in this Example

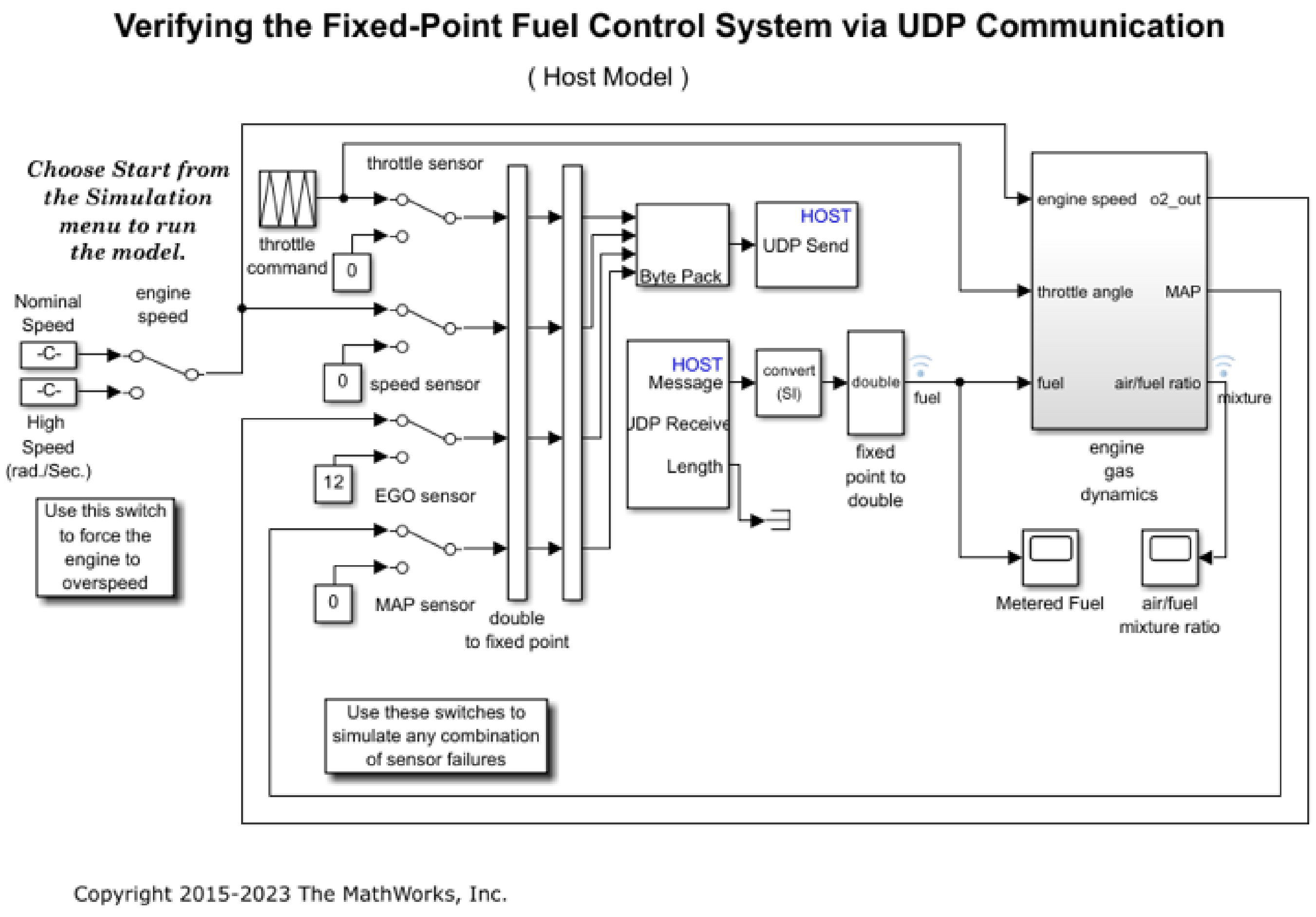
1. UDP Target Model

```
open_system('f28m35_UDP_target');
```


Verifying the Fixed-Point Fuel Control System via UDP Communication
(Target Model)
F28M3x - M3
Data
Status
UDP RCV
Byte Unpack
fuel rate
fuelsys
uint16
(SI)
F28M3x - M3
UDP SEND
Copyright 2015-2023 The MathWorks, Inc.

2. UDP Host model

```
open_system('f28m35_UDP_host');
```



Run the Model configured for the target on the hardware

1. Open the UDP target model. This model has been configured for **TI Concerto F28M35x (ARM Cortex-M3)**. To configure the model to run on a different board like **TI Concerto F28M36x (ARM Cortex-M3)**, change the selected **Hardware board** in the Configuration Parameters > Hardware Implementation pane.

2. Configure the network properties for the model as described in the previous task: **Hardware and Network Configuration**.

- On the target UDP Send block, the remote IP port has to match the local IP port of the host model UDP Receive block.
- On the target UDP Receive block, the local IP port has to match the remote IP port of the host model UDP Send block.

3. The UDP Send and Receive blocks on the target model are configured to do a transmit broadcast and receive from all IP address in the subnet. You can change this to send to a specific IP address and receive from a specific IP address if you know the IP address of the host computer.

4. On the UDP target model, press Ctrl+B or click on the **Deploy to Hardware** button to build, load and run the model on the target.

Run the Host Model and analyze the data

- While the target model is running, run the UDP Host model to send simulated sensor data to the target and visualize the calculated response by the controller.

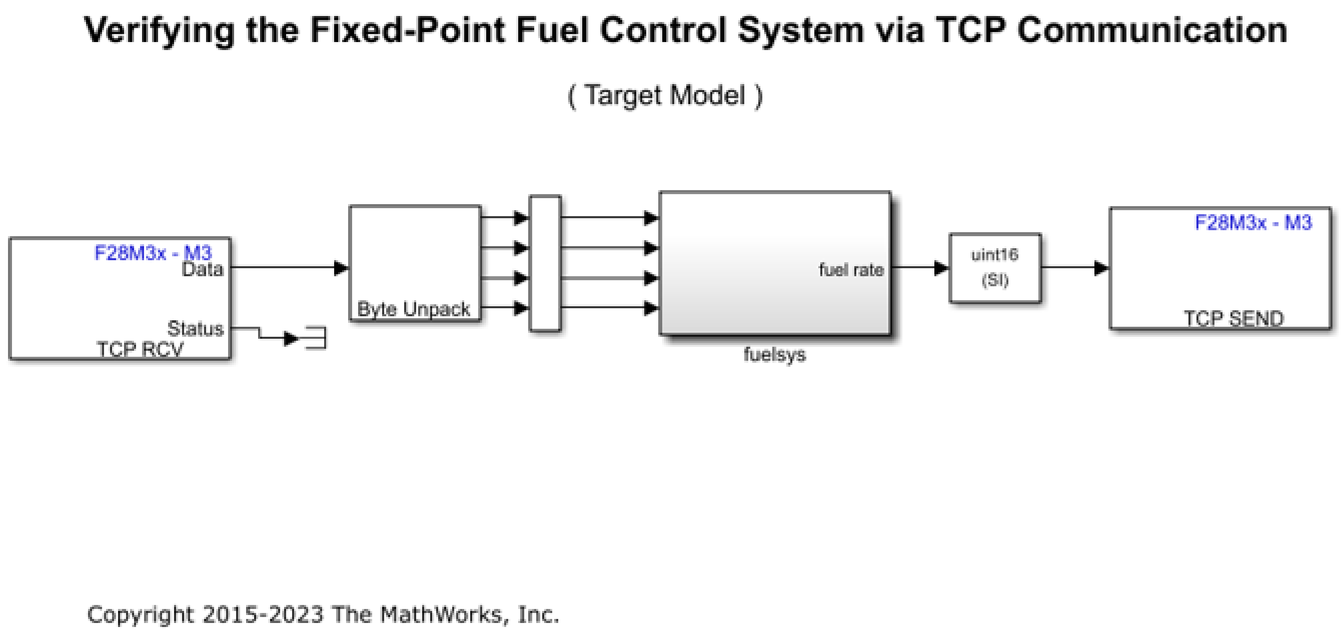
TCP Communication Between the Target and the Host Computer

In this task, you will run the Modeling a Fault-Tolerant Fuel Control System on the ARM Cortex M3 Core of the Concerto processor. The control algorithm runs on the target while the host runs a plant simulation of the system. The target and the host communicate via TCP. The target model receives the fuel system control sensors values from the host model and transmits back the calculated fuel mixture ratio to the host computer.

Models Involved in this Example

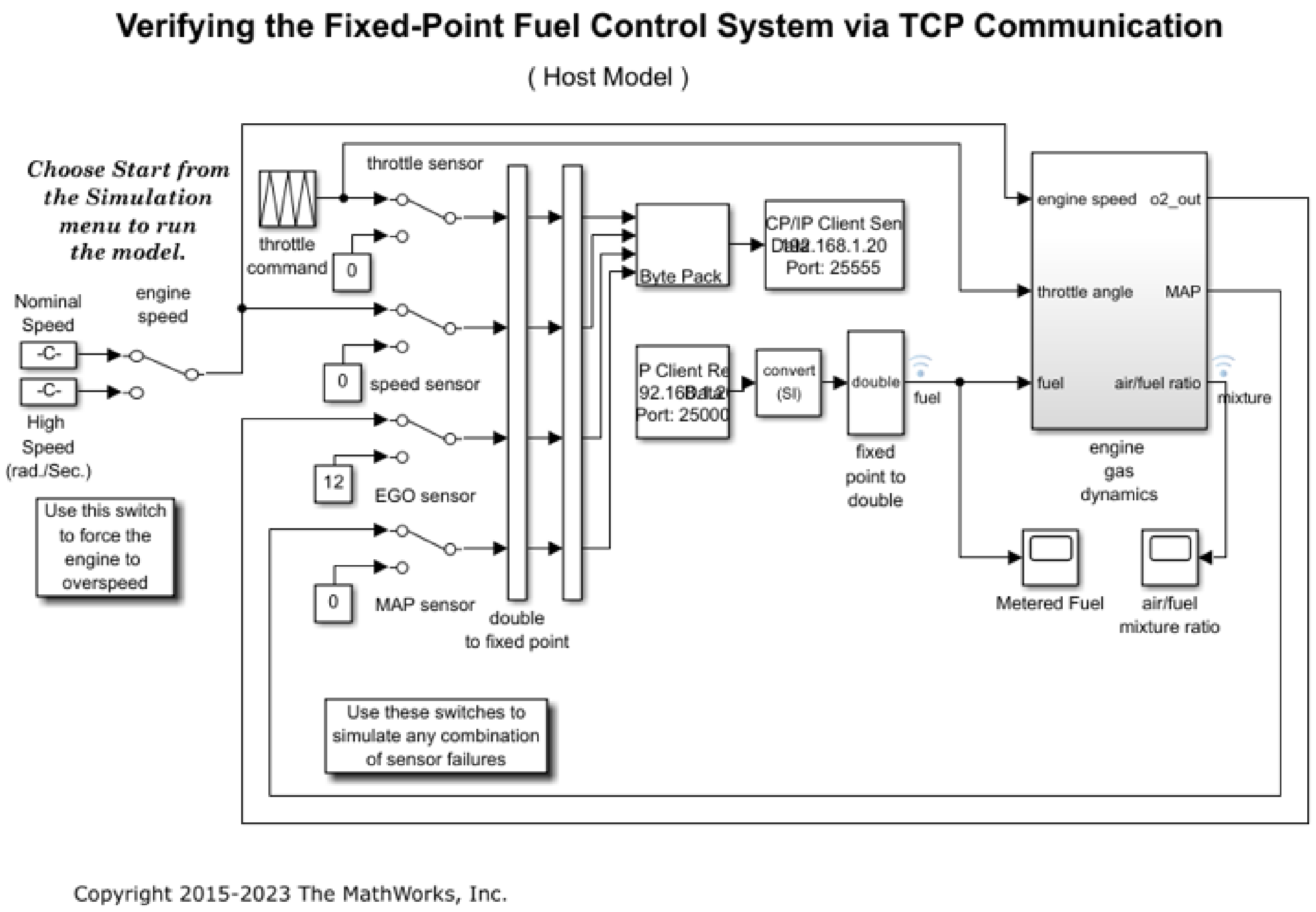
1. Target TCP model

```
open_system('f28m35_TCP_target');
```


Verifying the Fixed-Point Fuel Control System via TCP Communication
(Target Model)
F28M3x - M3
Data
Status
TCP RCV
Byte Unpack
fuel rate
fuelsys
uint16
(SI)
F28M3x - M3
TCP SEND
Copyright 2015-2023 The MathWorks, Inc.

2. Host TCP model - This model requires Instrument Control Toolbox.

```
open_system('f28m35_TCP_host');
```



Run the model configured for the target on the hardware

1. Open the Target TCP model. This model has been configured for the **TI Concerto F28M35x (ARM Cortex-M3)** target. To configure the model to run on a different board like **TI Concerto F28M36x (ARM Cortex-M3)**, change the selected **Hardware board** in the Configuration Parameters > Hardware Implementation pane.

2. Configure the network properties for the model as described in the previous task: **Hardware and Network Configuration**.

- On the target TCP Send block, the local IP port has to match the port of the host model TCP Receive block.
- On the target TCP Receive block, the local IP port has to match the port of the host model TCP Send block.

3. On the target TCP model, press Ctrl+B or click on the **Deploy to Hardware** button to build, load and run the model on the target.

Run the Host Model to receive and analyze the data

- While the target model is running, run the Host TCP model to send simulated sensor data to the target and visualize the calculated response by the controller.

```
close_system('f28m35_UDP_target',0);
close_system('f28m35_UDP_host',0);
close_system('f28m35_TCP_target',0);
close_system('f28m35_TCP_host',0);
```

Calibrate ECU Parameters from Third-party Calibration Tools Using XCP-based CAN Interface

This example shows how to monitor signals and tune parameters of a Simulink model on Texas Instruments™ C2000™ board using XCP-based CAN Interface.

Required Third-party Software

- Vector CANape®

Note: The integration of Simulink with CANape using Texas Instruments™ C2000™ board has been tested with CANape 16.0 SP6.

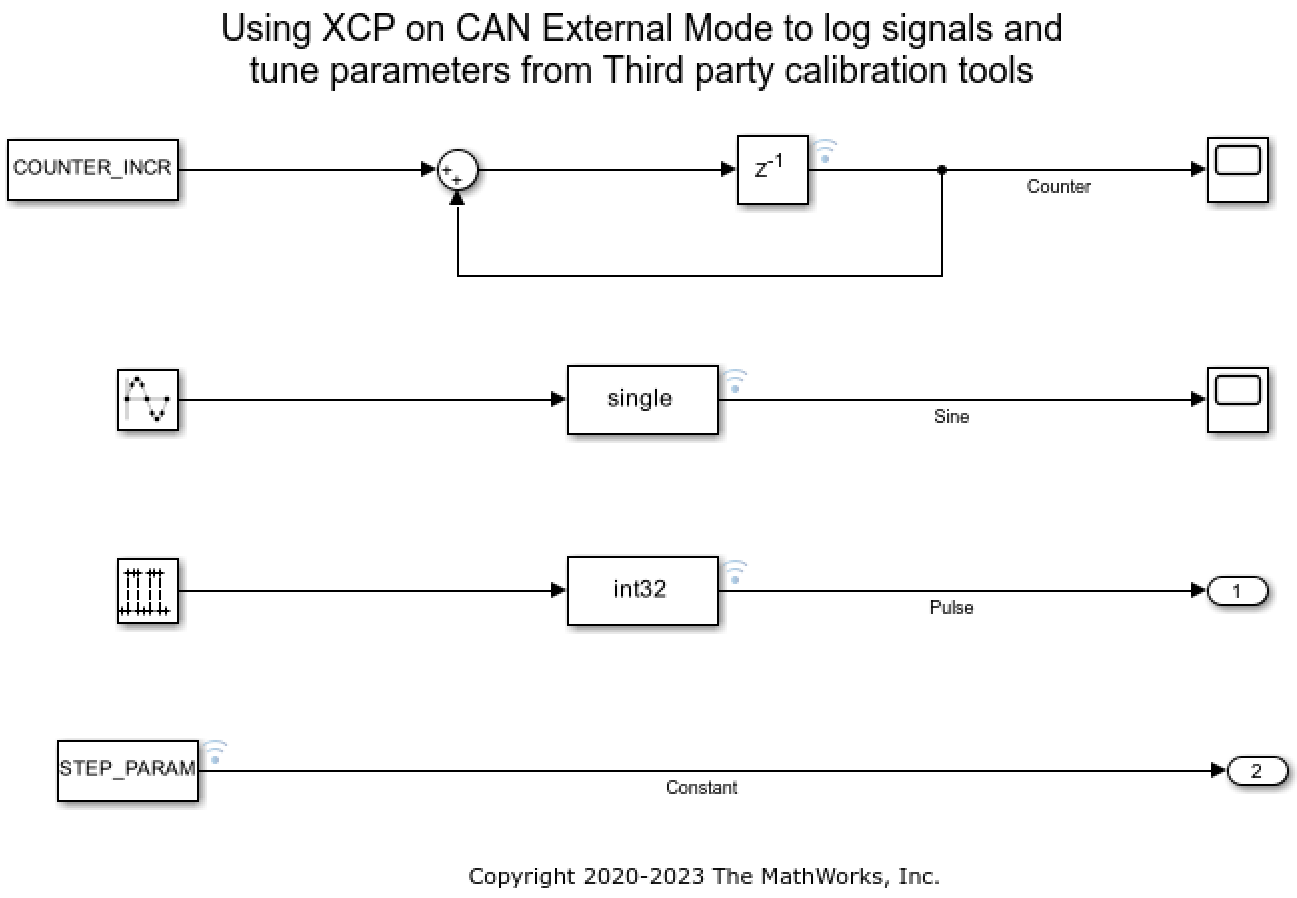
Required Hardware

Any Texas Instruments™ C2000™ board with CAN module

Model

To open the pre-configured model, run this command at the MATLAB prompt:

```
open_system('c28x_xcp_can_3pcaltool.slx');
```


Using XCP on CAN External Mode to log signals and tune parameters from Third party calibration tools
COUNTER_INCR
z^-1
Counter
single
Sine
int32
Pulse
1
STEP_PARAM
Constant
2
Copyright 2020-2023 The MathWorks, Inc.

This example contains four signals that are enabled for logging and two parameters for tuning. The four signals are:

- Counter - Increments the output by COUNTER_INCR at every time step and COUNTER_INCR is tunable
- Sine_Wave - A Sine Wave
- Pulse - A Pulse signal
- Constant - A Constant signal of value STEP_PARAM and STEP_PARAM is tunable

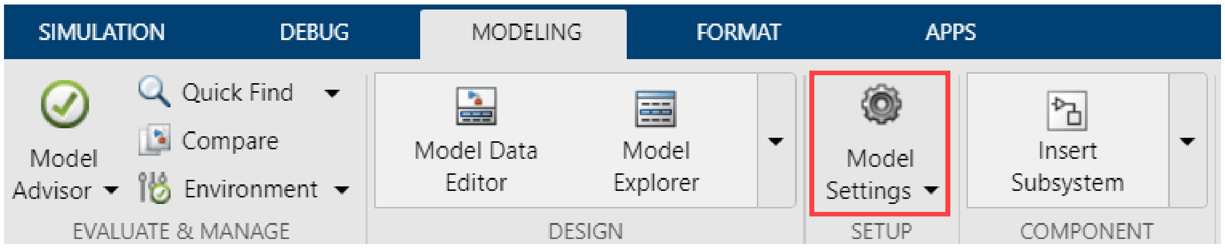
Configuring the Model

In this task, you will configure a Simulink model and enable calibration of parameters in third-party calibration tools.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

1. Open the model.

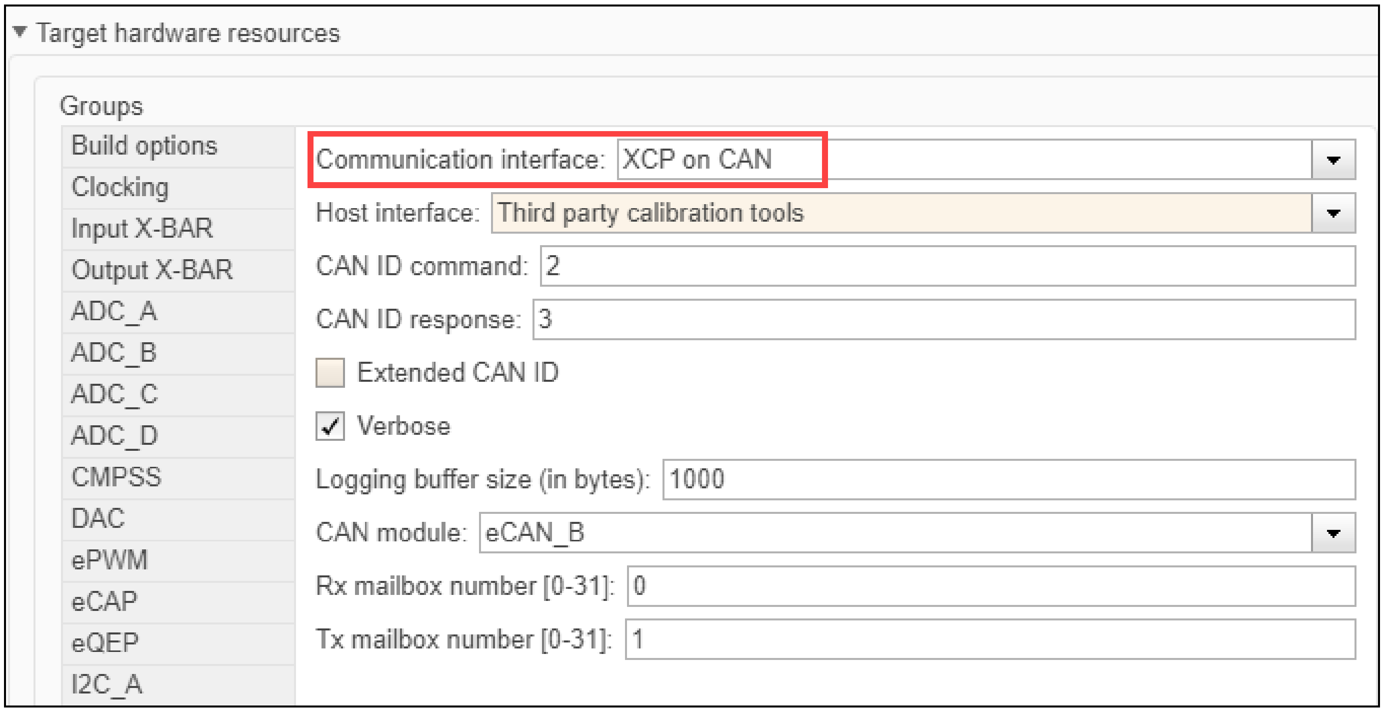
2. Go to **Modeling** > **Model Settings** to open the Configuration Parameters dialog box.



3. Open the **Hardware Implementation** pane, and select the required Texas Instruments C2000 board from the list in **Hardware board** parameter.

4. Expand **Target hardware resources** for that board.

5. Go to **External mode** tab and choose *XCP on CAN* as the **Communication interface**.



Communication interface: XCP on CAN
Host interface: Third party calibration tools
CAN ID command: 2
CAN ID response: 3
Extended CAN ID: ☐
Verbose: ✓
Logging buffer size (in bytes): 1000
CAN module: eCAN_B
Rx mailbox number [0-31]: 0
Tx mailbox number [0-31]: 1

6. Configure the eCAN module on the target.

a. Enter the values for **CAN ID command** and **CAN ID response**.

b. Select **Extended CAN ID** option, if you want to use extended ID.

c. Select **Verbose** option to view the External Mode execution progress and updates in the Diagnostic Viewer or in the MATLAB Command Window.

d. Enter the value for **logging buffer size**.

e. Select the CAN module to use with external mode, if the target supports multiple eCAN modules.

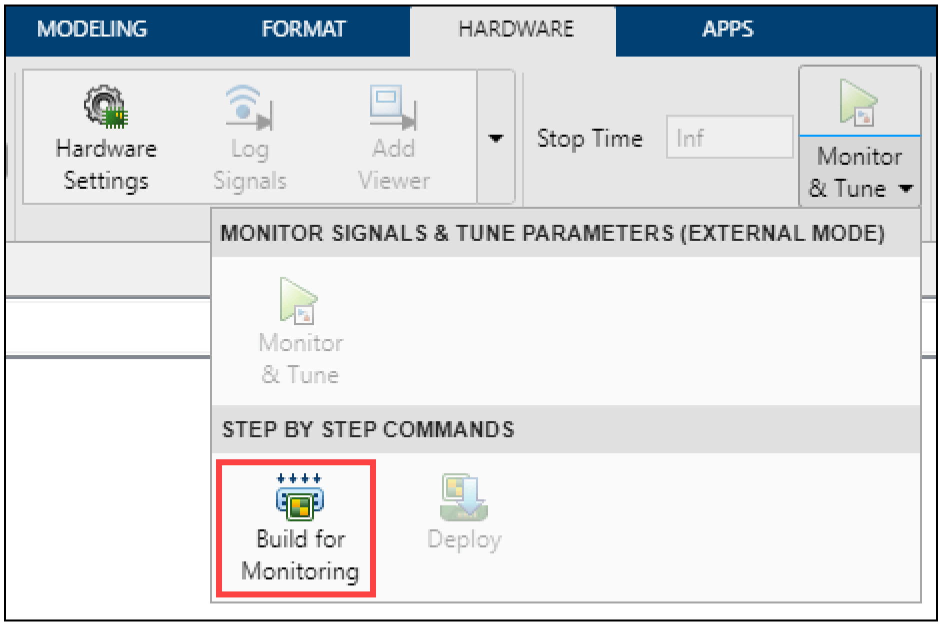
f. Enter the values for **Rx mailbox number** and **Tx mailbox number**.

7. Go to **Code Generation** > **Optimization** and then set **Default parameter behavior** to `Inlined`.

8. Click **Apply** and **OK**.

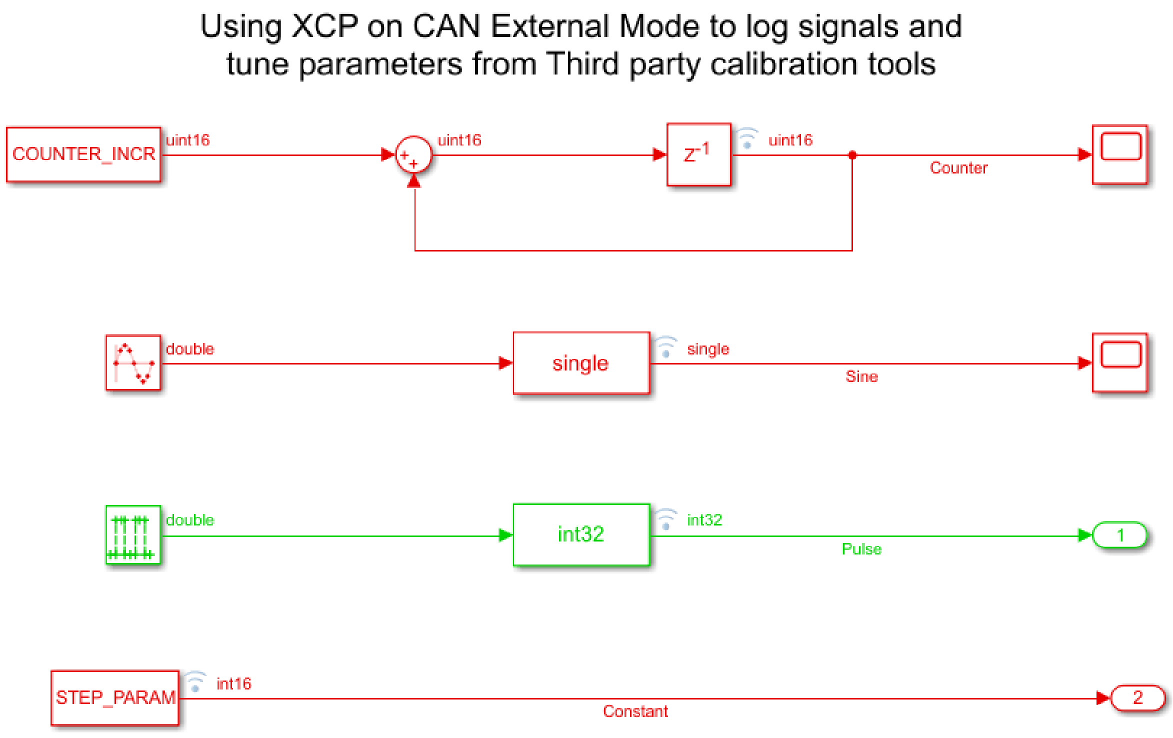
Initiate Build for Monitoring Action for the Model and Generate A2L File

On the **Hardware** tab of the Simulink toolstrip, click **Build for Monitoring**.



Click **Deploy** in the Simulink Toolstrip to deploy the executable onto the target.

The model is deployed on the Texas Instruments C2000 board and an A2L file named *modelname*.a2l is generated in the current folder path in MATLAB. After successful deployment, third-party calibration tools can be used to connect to the Texas Instruments C2000 board for monitoring signals and tuning parameters.



Create a New Project in CANape and Connect to the Texas Instruments C2000 board

1. Open Vector CANape software. Create a new Project.

2. Drag and drop the A2L file generated from the model into CANape. This opens a new dialog box which creates a new device. Click **Next**.

3. In Network settings, click **New network** button. This opens a dialog box where CAN Channel can be selected. Also, change the Baud Rate to the value that was used in Simulink model. Accept the changes and close the dialog.

4. Click **Next** and finally click **OK**. A new device is created, and the Settings dialog box for the newly created device opens.

5. Expand the **Protocol** tab in the Settings dialog box. Click **Transport Layer**.

6. Ensure that CAN Master ID, CAN Slave ID and other settings are same as the ones used in Simulink model.

7. Click on **Protocol** tab. Go to **Expert settings**. Change SHORT_DOWNLOAD_DISABLED and SHORT_UPLOAD_DISABLED options to yes.

8. Click **Accept all changes** in the top-left corner of the Settings dialog box and close it.

9. Click **Online** to connect to the Texas Instruments C2000 target.

Perform Measurement and Calibration from Third-party Tools

1. Open the list of signals and parameters by expanding the **Devices** tab in the **Explorer** pane.

2. Drag the signals that you want to monitor, to the Display area and select a graphic window.

3. Drag the parameters that you want to tune, to the Display area and select Parameter window.

4. Click **Start measurement** in the Start tab to start monitoring the selected signals.

5. Use the Parameter window to tune the parameters.

Troubleshoot Calibration in Third-party Tools

While you perform calibration of parameters using third-party, you may encounter these errors:

- CANape fails to connect with the error: `No response from the ECU`

  ```
  To resolve this issue, ensure that the CAN Channel Baud Rate, CAN Slave
  ID, CAN Master ID, CAN ID Type are correct and try to connect again.
  ```

- Data Acquisition does not start when you click **Start** for the first time.

  ```
  To resolve the issue, click Start again.
  ```

  ```
  The issue occurs because CANape sends command 0xD7 (GET_DAQ_EVENT_INFO) and 0xDC (GET_DAQ_CLOCK
  are not listed as supported optional commands in the ASAP2 file.
  By default, CANape has XCP_OPTIONAL_CMD_AUTO_LEARNING feature, which learns that the GET_DAQ_EV
  are not supported, and correctly sends DAQ on the second attempt. This information persists in
  does not occur again for the same project.
  ```

Related Topics

- “Set Up CAN Communication with Target Hardware” on page 1-8

Signal Monitoring and Parameter Tuning Over XCP-based CAN Interface

This example shows how to monitor signals and tune parameters of a Simulink® model on Texas Instruments™ C2000™ board using XCP-based CAN Interface.

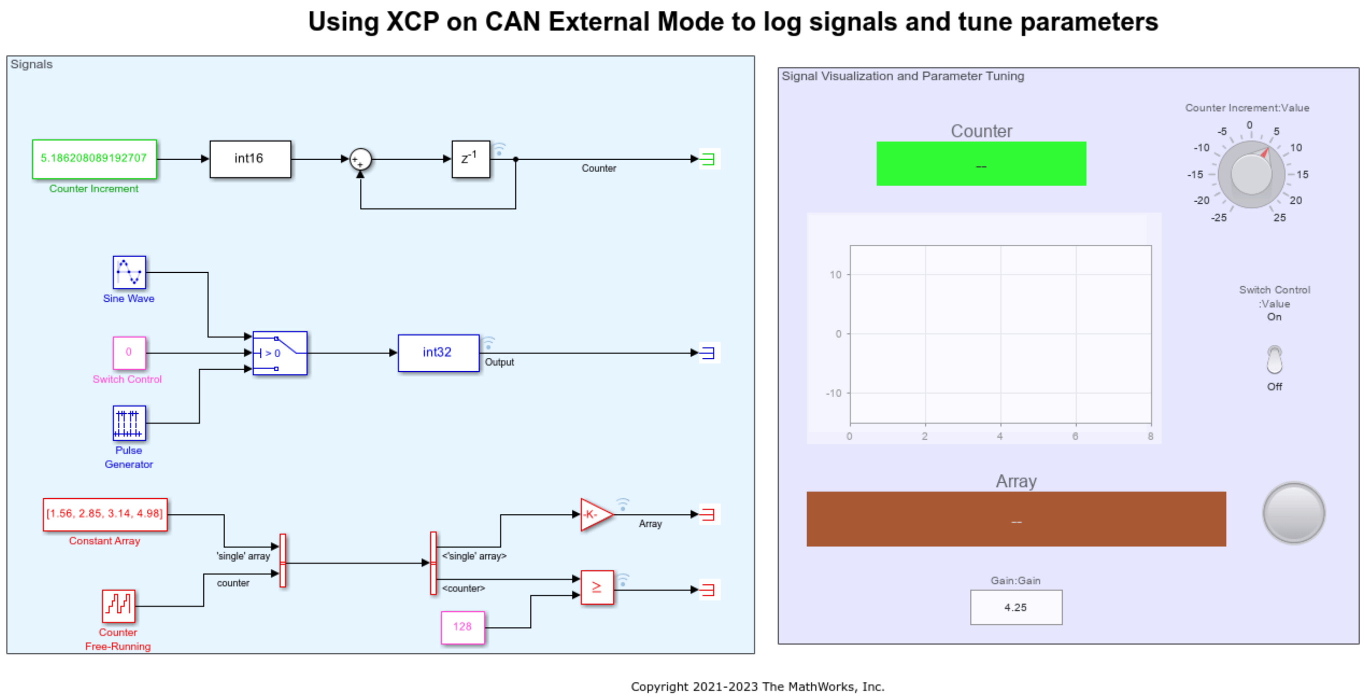
Required Hardware

Any Texas Instruments C2000 board with CAN module

Model

To open the pre-configured model, run this command at the MATLAB® prompt:

```
open_system('c28x_xcp_can_ext.slx');
```



In this example, multiple signals of different data types such as int36, int32, single are selected for logging. The visualization section on the right-hand side contains Dashboard Scope and Dashboard Display blocks which shows the values of different signals that are being logged.

Different parameters in the source blocks are tunable. Dashboard Knob, Toggle Switch and Edit boxes can also be used to tune the parameters.

Configuring the Model

In this task, you will configure a Simulink model and enable calibration of parameters in third-party calibration tools.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

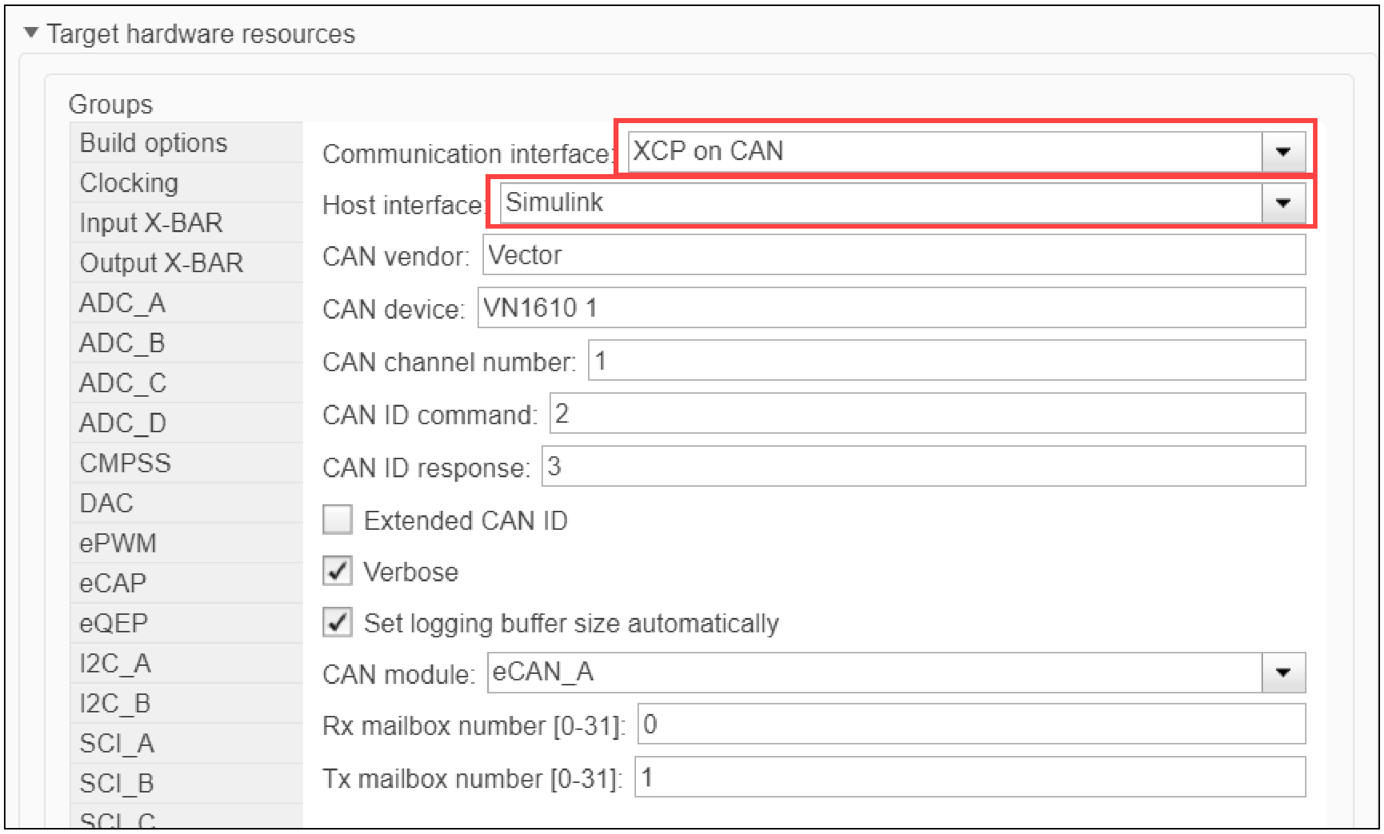
1. Open the model.

2. Go to **Modeling** > **Model Settings** to open the Configuration Parameters dialog box.

3. Open the **Hardware Implementation** pane, and select the required Texas Instruments C2000 board from the list in **Hardware board** parameter.

4. Expand **Target hardware resources** for that board.

5. Go to **External mode** tab and choose *XCP on CAN* as the **Communication interface**.

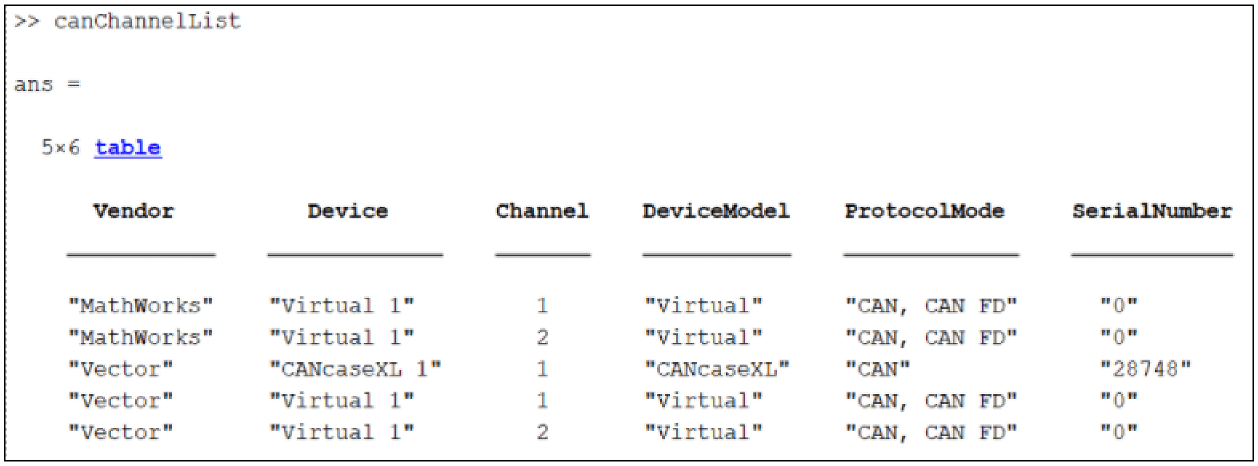


Communication interface: XCP on CAN
Host interface: Simulink
CAN vendor: Vector
CAN device: VN1610 1
CAN channel number: 1
CAN ID command: 2
CAN ID response: 3
☐ Extended CAN ID
☑ Verbose
☑ Set logging buffer size automatically
CAN module: eCAN_A
Rx mailbox number [0-31]: 0
Tx mailbox number [0-31]: 1

6. Select **Simulink** as the **Host interface**.

7. Configure the eCAN module on the target.

a. Enter the values for CAN vendor, CAN device, and CAN channel number. Use the Vehicle Network Toolbox function `canChannelList()` to get values for CAN vendor, CAN device, CAN channel. This function returns a list of all the CAN interfaces whose drivers have been installed and connected to the computer. In the MATLAB command window, type `canChannelList()` and press enter. A sample screen is shown here.



```
>> canChannelList

ans =

  5×6 table

      Vendor            Device          Channel     DeviceModel      ProtocolMode       SerialNumber
    ___________     _____________     _______     ___________     _____________     ____________

    "MathWorks"     "Virtual 1"          1        "Virtual"       "CAN, CAN FD"         "0"
    "MathWorks"     "Virtual 1"          2        "Virtual"       "CAN, CAN FD"         "0"
    "Vector"        "CANcaseXL 1"        1        "CANcaseXL"     "CAN"                 "28748"
    "Vector"        "Virtual 1"          1        "Virtual"       "CAN, CAN FD"         "0"
    "Vector"        "Virtual 1"          2        "Virtual"       "CAN, CAN FD"         "0"
```

b. Enter the values for **CAN ID command** and **CAN ID response**.

c. Select **Extended CAN ID** option, if you want to use extended ID.

d. Select **Verbose** option to view the External Mode execution progress and updates in the Diagnostic Viewer or in the MATLAB Command Window.

e. Select **Set logging buffer size automatically** option to set the optimal buffer size for logging data.

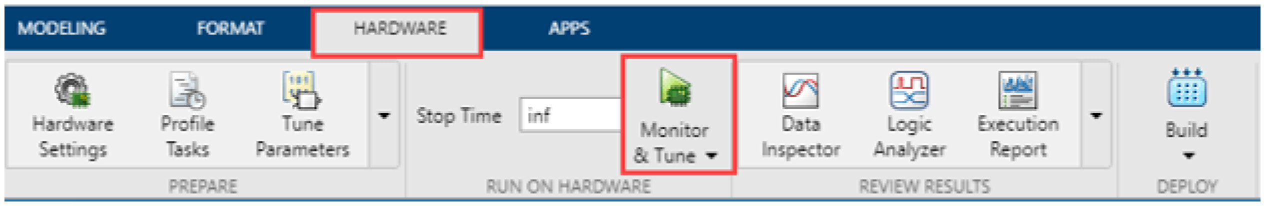
f. Select the CAN module to use with external mode, if the target supports multiple eCAN modules.

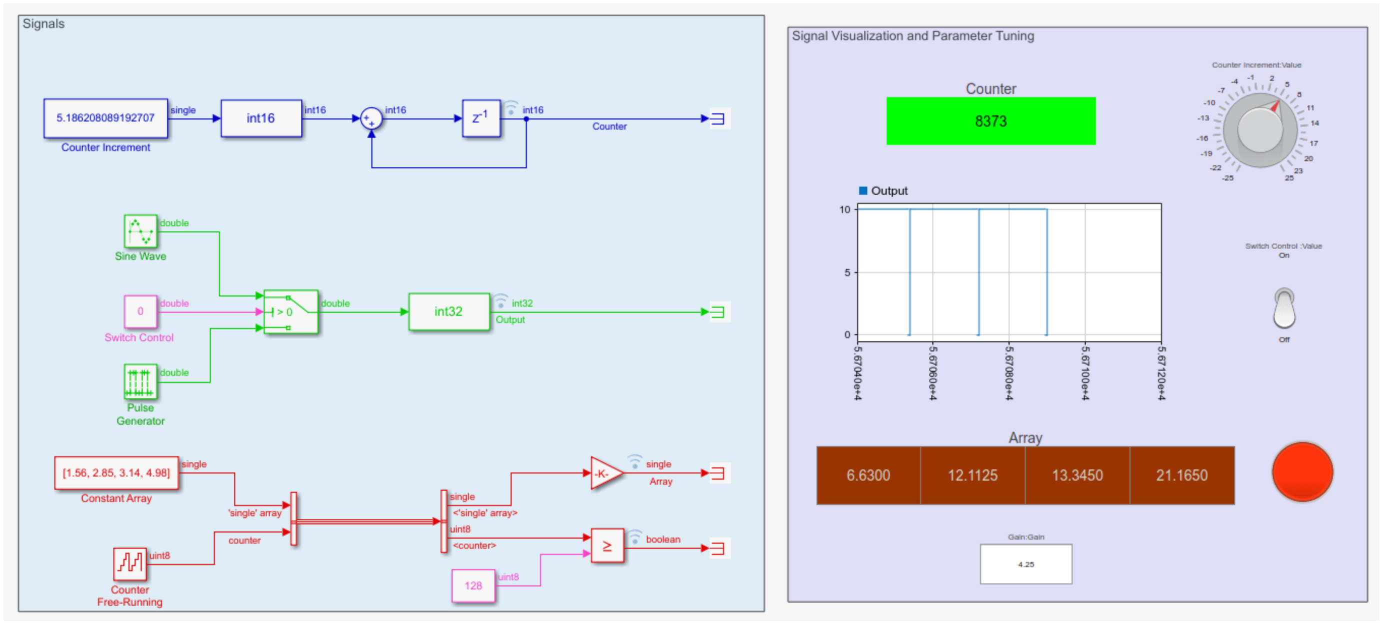
g. Enter the values for **Rx mailbox number** and **Tx mailbox number**.

8. Click **Apply** and **OK**.

Initiate Monitor and Tune Action for the Model

On the **Hardware** tab of the Simulink toolstrip, click **Monitor & Tune** to monitor signals and tune parameters.





Related Topics

- “Set Up CAN Communication with Target Hardware” on page 1-8

Code Execution Profiling on Texas Instruments C2000 Targets in XCP External Mode

This example shows how to use for Texas Instruments™ C2000™ Microcontroller Blockset to profile real-time execution of the generated code running as an executable on a Texas Instruments C2000 board with XCP on Serial and XCP on TCP/IP Interface.

Introduction

Sample times you specify in the Simulink® models determine the time schedule for running generated code on target hardware. With enough computing power on the hardware, the code runs in real-time according to the specified sample times. With real-time execution profiling, you can check if the generated code meets your real-time performance requirements. This support blockset supports the code execution profiling on any Texas Instruments C2000 board.

At the end of the Simulink model code profile execution, you can:

- View a report of code execution times.
- Access and analyze execution time profiling data.

Prerequisite

Complete the Getting Started with Embedded Coder Support Package for TI C2000 Processors video.

Note: Starting R2023a, the workflow for getting started tutorial for support package will continue to support in C2000™ Microcontroller Blockset.

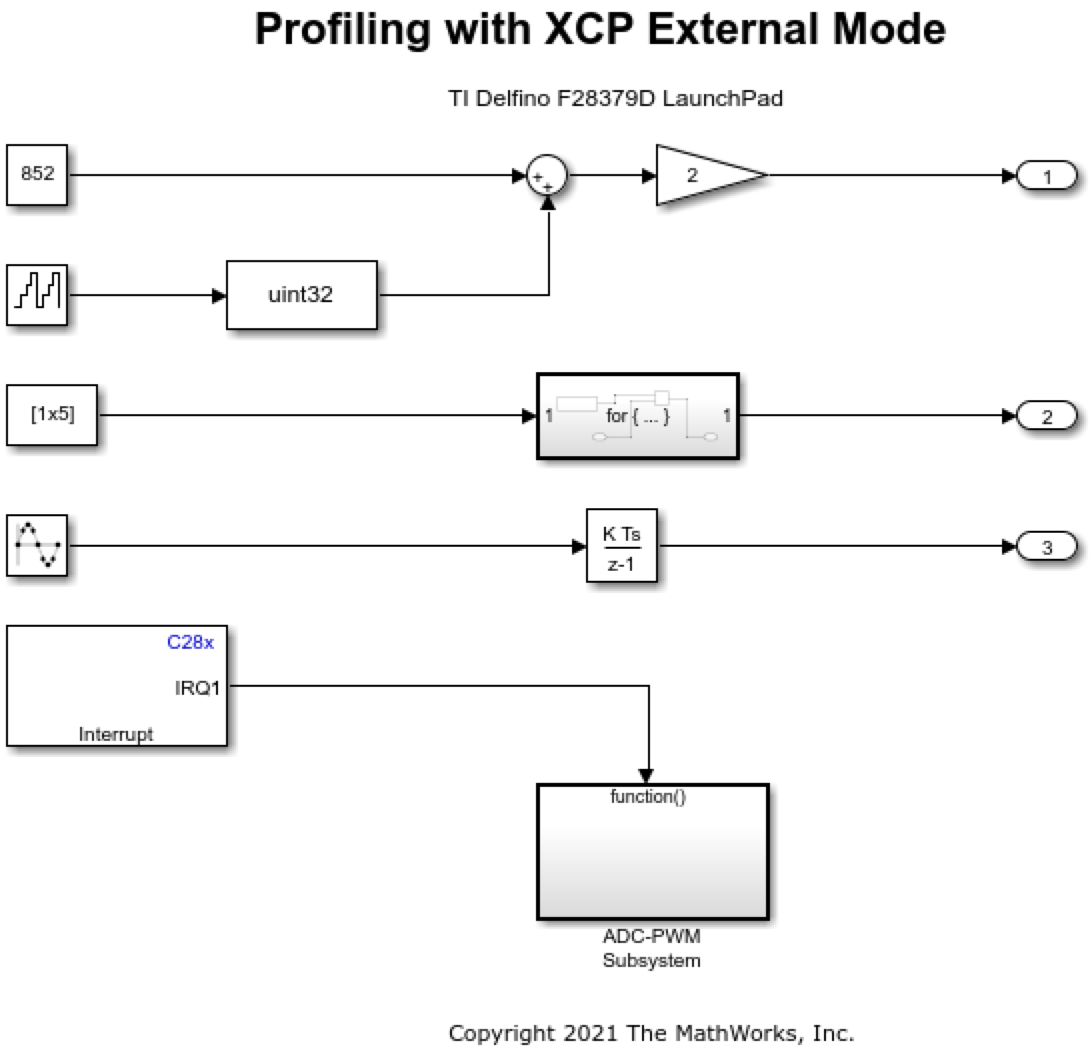
Required Hardware

Any Texas Instruments C2000 board

Model

This example uses a preconfigured model. Open the model using this command.

```
open_system('f28379D_profiling_xcp')
```


Profiling with XCP External Mode
TI Delfino F28379D LaunchPad
852
uint32
2
1
[1x5]
for { ... }
2
K Ts
z-1
3
C28x
IRQ1
Interrupt
function()
ADC-PWM Subsystem
Copyright 2021 The MathWorks, Inc.

Configuring the Model

In this example, you will configure a Simulink model and enable profiling.

Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

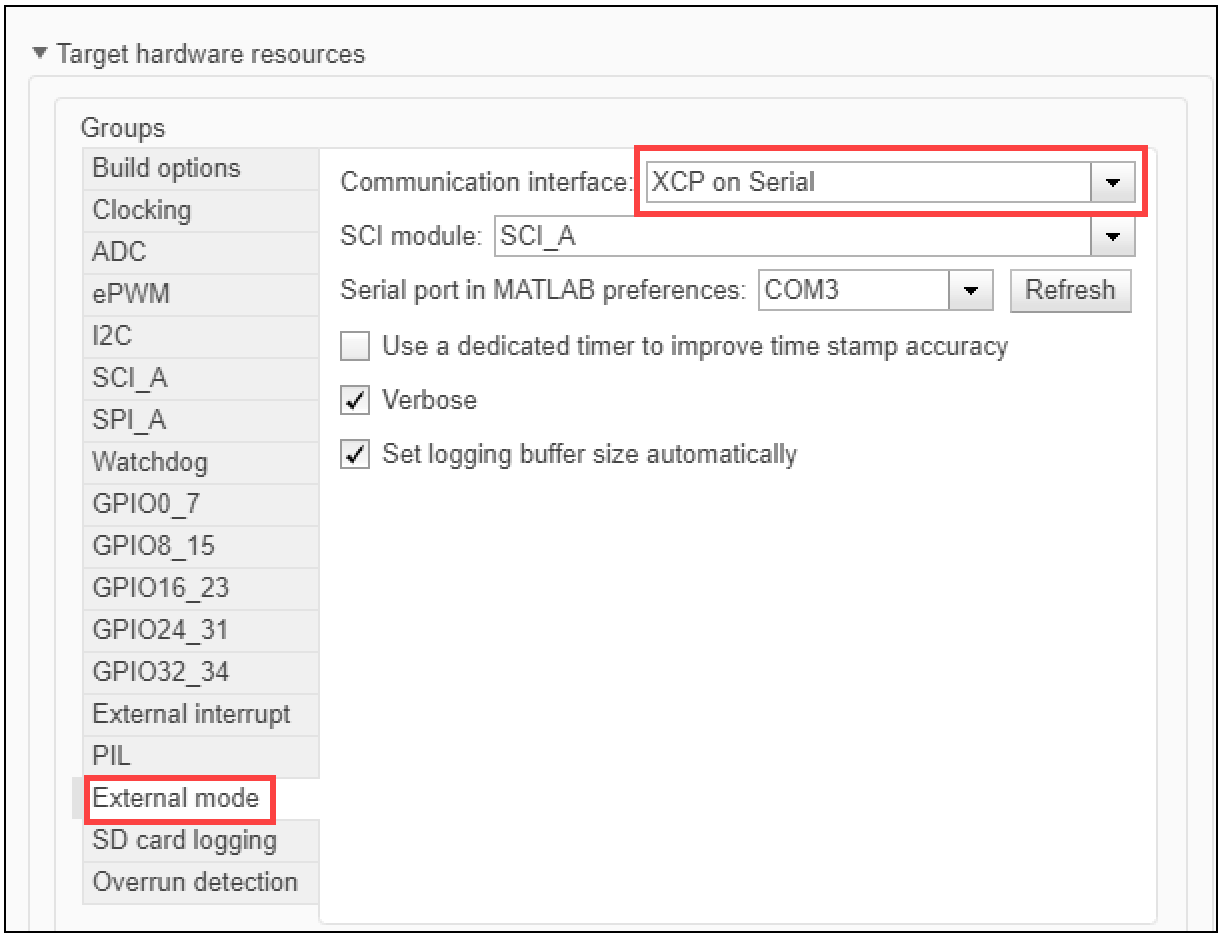
1. Open the model.

2. Go to **Modeling** > **Model Settings** to open the Configuration Parameters dialog box.

3. Open the **Hardware Implementation** pane, and select the required Texas Instruments C2000 board from the list in **Hardware board** parameter.

4. Expand **Target hardware resources** for that board.

5. Go to **External mode** tab and choose *XCP on Serial* as the **Communication interface**.



Communication interface: XCP on Serial

SCI module: SCI_A

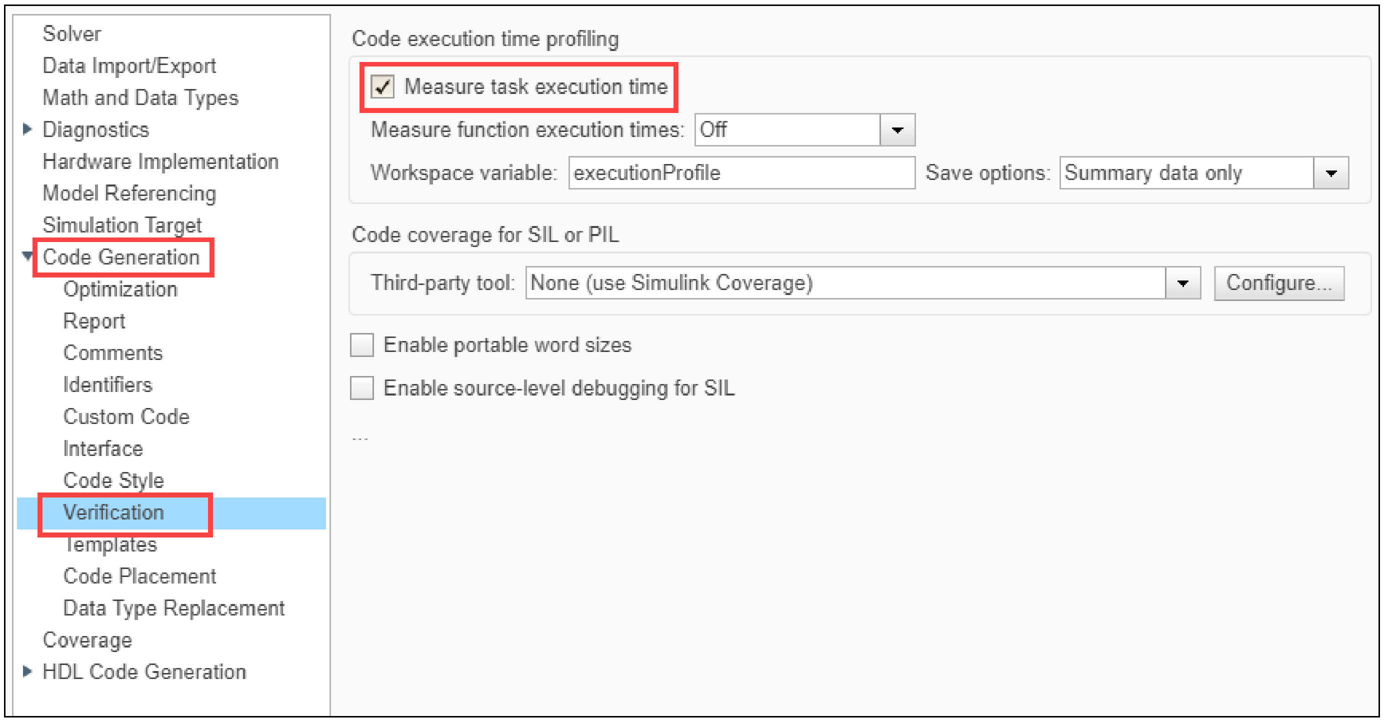
Serial port in MATLAB preferences: COM3

☐ Use a dedicated timer to improve time stamp accuracy

☑ Verbose

☑ Set logging buffer size automatically

6. Go to **Code Generation** > **Verification** > **Code execution time profiling** and select **Measure task execution time**.

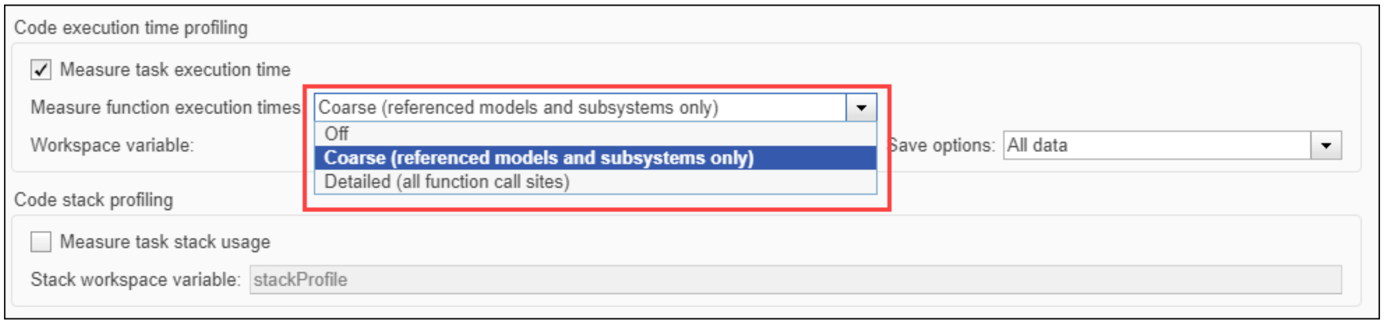

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Optimization
Report
Comments
Identifiers
Custom Code
Interface
Code Style
Verification
Templates
Code Placement
Data Type Replacement
Coverage
HDL Code Generation

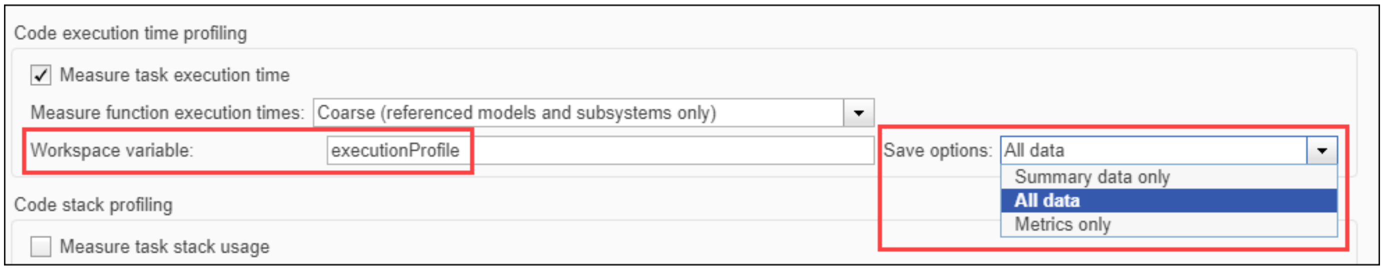
Code execution time profiling
✓ Measure task execution time
Measure function execution times: Off
Workspace variable: executionProfile
Save options: Summary data only

Code coverage for SIL or PIL
Third-party tool: None (use Simulink Coverage)

Enable portable word sizes
Enable source-level debugging for SIL
...

7. Select the required option for **Measure function execution times**, **Workspace Variable** and **Save Options**. For information on different save options, see “Save Options” on page 1-110.


Code execution time profiling
✓ Measure task execution time
Measure function execution times: Coarse (referenced models and subsystems only)
Off
Coarse (referenced models and subsystems only)
Detailed (all function call sites)
Workspace variable:
Save options: All data
Code stack profiling
Measure task stack usage
Stack workspace variable: stackProfile


Code execution time profiling
✓ Measure task execution time
Measure function execution times: Coarse (referenced models and subsystems only)
Workspace variable: executionProfile
Save options: All data

Code stack profiling
Measure task stack usage

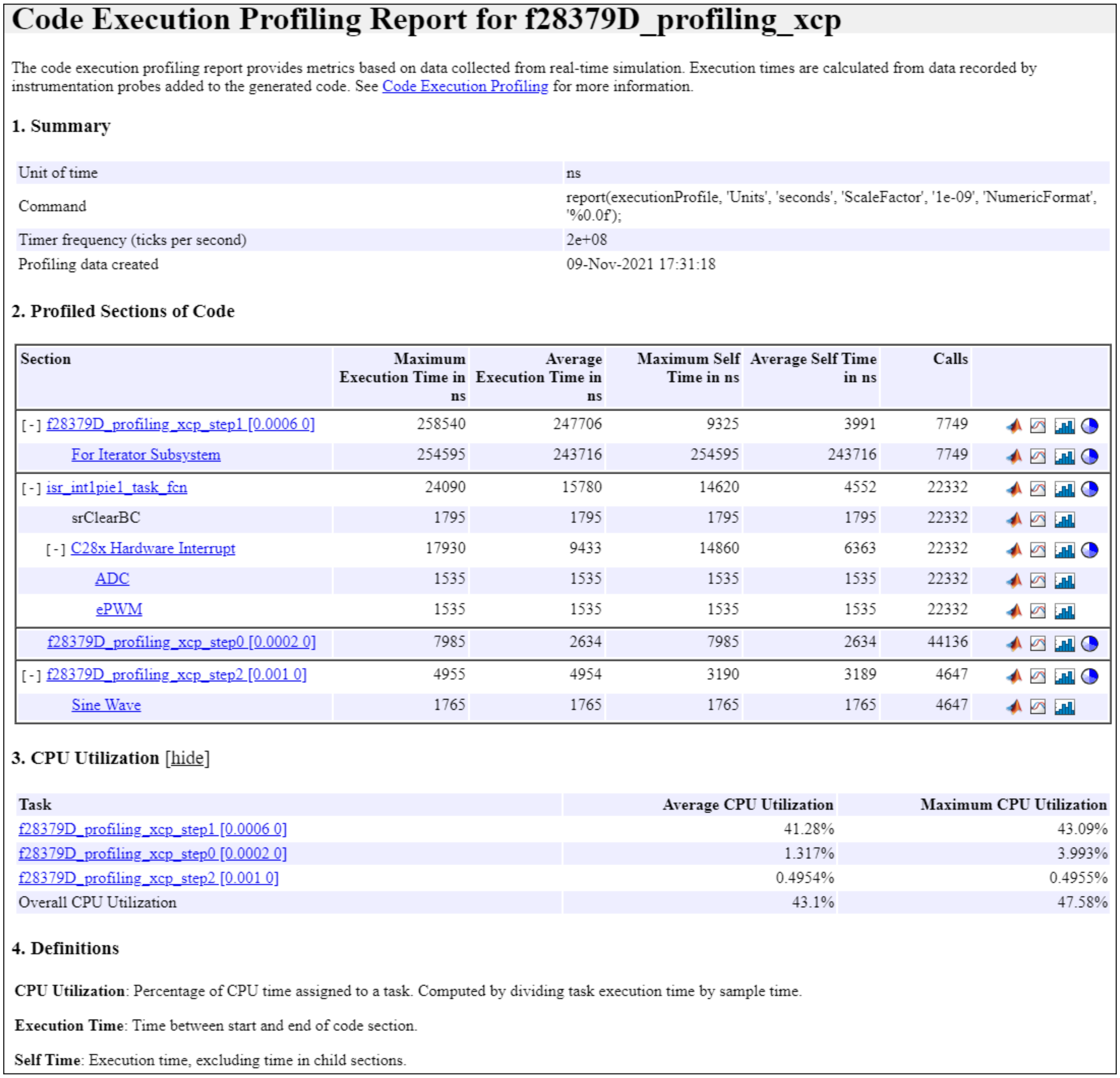
8. Click **Apply** and **OK**.

Initiate Monitor and Tune Action for the Model

On the **Hardware** tab of the Simulink toolstrip, click **Monitor & Tune** to monitor signals and tune parameters.

This generates a profiling report with profiling metrics of different tasks/functions that are being profiled. This also contains links to plot the data on SDI, bar charts, pie charts and CPU Utilization which provides further analysis of the data.

All data report



Code Execution Profiling Report for f28379D_profiling_xcp

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

1. Summary

Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	2e+08
Profiling data created	09-Nov-2021 17:31:18

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls
[-] f28379D_profiling_xcp_step1 [0.0006 0]	258540	247706	9325	3991	7749
For Iterator Subsystem	254595	243716	254595	243716	7749
[-] isr_int1pie1_task_fcn	24090	15780	14620	4552	22332
srClearBC	1795	1795	1795	1795	22332
[-] C28x Hardware Interrupt	17930	9433	14860	6363	22332
ADC	1535	1535	1535	1535	22332
ePWM	1535	1535	1535	1535	22332
f28379D_profiling_xcp_step0 [0.0002 0]	7985	2634	7985	2634	44136
[-] f28379D_profiling_xcp_step2 [0.001 0]	4955	4954	3190	3189	4647
Sine Wave	1765	1765	1765	1765	4647

3. CPU Utilization [hide]

Task	Average CPU Utilization	Maximum CPU Utilization
f28379D_profiling_xcp_step1 [0.0006 0]	41.28%	43.09%
f28379D_profiling_xcp_step0 [0.0002 0]	1.317%	3.993%
f28379D_profiling_xcp_step2 [0.001 0]	0.4954%	0.4955%
Overall CPU Utilization	43.1%	47.58%

4. Definitions

CPU Utilization: Percentage of CPU time assigned to a task. Computed by dividing task execution time by sample time.

Execution Time: Time between start and end of code section.

Self Time: Execution time, excluding time in child sections.

Code Execution Profiling Report Details

The profiling report is generated at the end of the simulation or when simulation is stopped manually. This example model contains three synchronous rates (200 microseconds, 600 microseconds, 1 microseconds) and an Asynchronous Hardware Interrupt block. The report is generated with **All data** save option and **Detailed** Function profiling. In the report following tasks/functions are profiled.

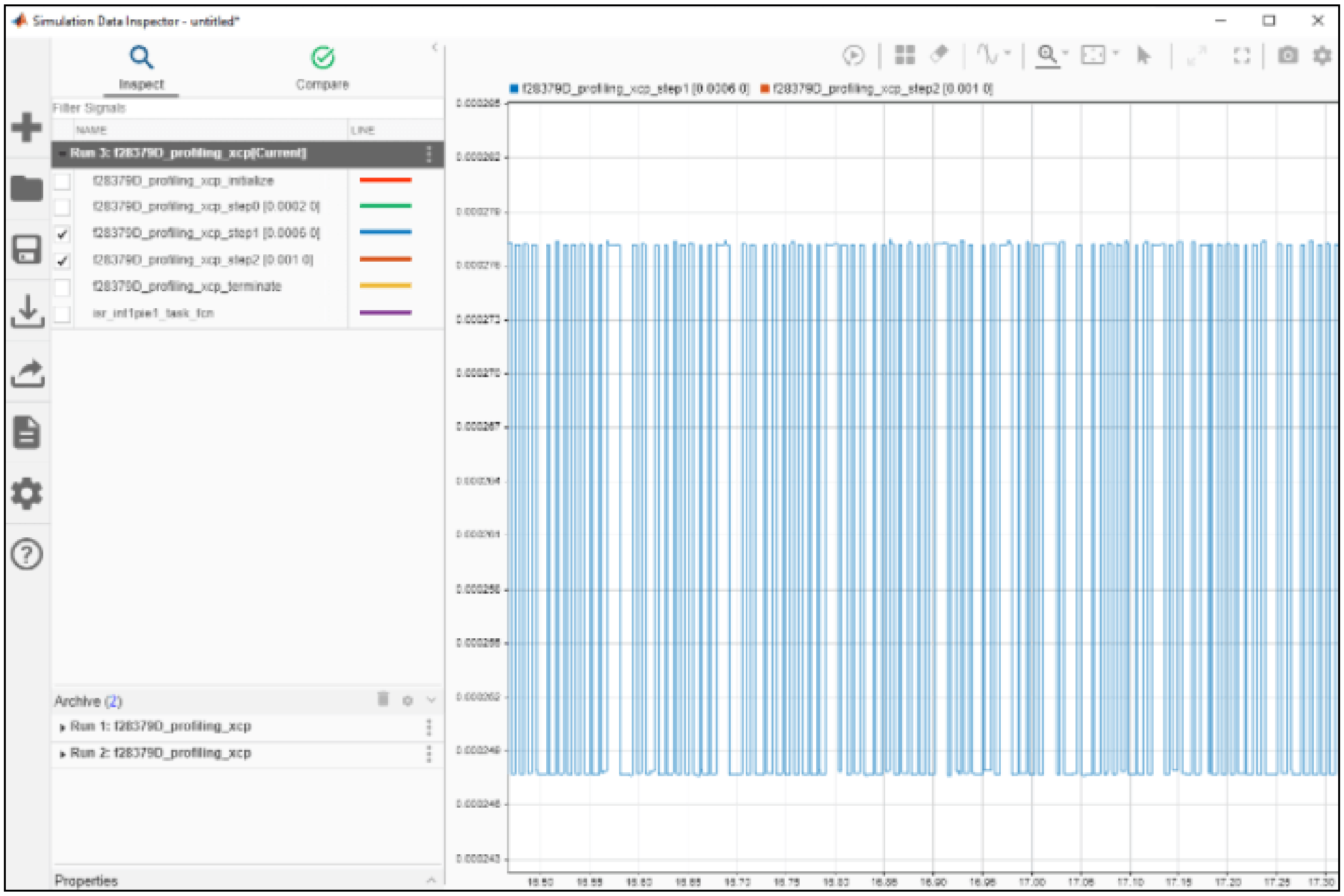
- All the three synchronous rates (step0, step1 and step2)
- Asynchronous rate (isr_int1pie1_task_fcn)
- ADC and PWM blocks inside the Hardware Interrupt block
- For Iterator Atomic subsystem
- Sine Wave block.

All the functions/tasks contain the following data:

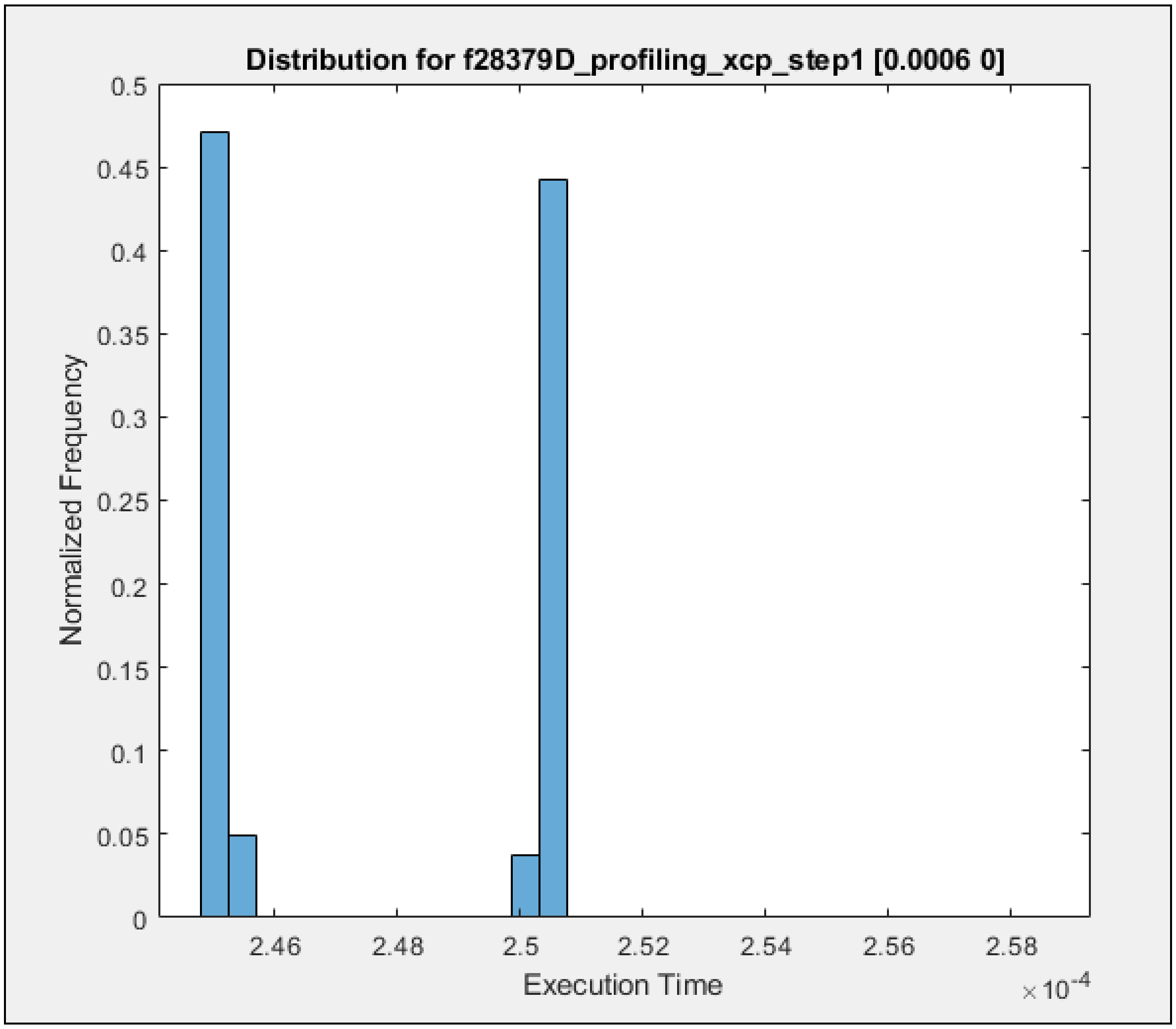
- Maximum and average execution time taken on the target.
- Maximum and average self time taken on the target.
- Number of times the function/task is called.

The report also contains links to:

- MATLAB Workspace variable containing the profiling data, which can be used to conduct a custom analysis of profiling data.
- Simulink Data Inspector: Profiling data can be visualized over the duration of simulation. A screen is shown below.


Simulation Data Inspector - untitled*
Inspect
Compare
Filter Signals
NAME
LINE
Run 3: f28379D_profiling_xcp[Current]
f28379D_profiling_xcp_initialize
f28379D_profiling_xcp_step0 [0.0002 0]
f28379D_profiling_xcp_step1 [0.0006 0]
f28379D_profiling_xcp_step2 [0.001 0]
f28379D_profiling_xcp_terminate
isr_int1pie1_task_fcn
Archive (2)
Run 1: f28379D_profiling_xcp
Run 2: f28379D_profiling_xcp
Properties
f28379D_profiling_xcp_step1 [0.0006 0]
f28379D_profiling_xcp_step2 [0.001 0]

- Bar chart: Contains Normalized frequency of the execution times of the task/function. A sample Bar chart is shown below.



- Pie chart: Shows the time taken by a function and its child functions. A sample Pie chart is shown below.

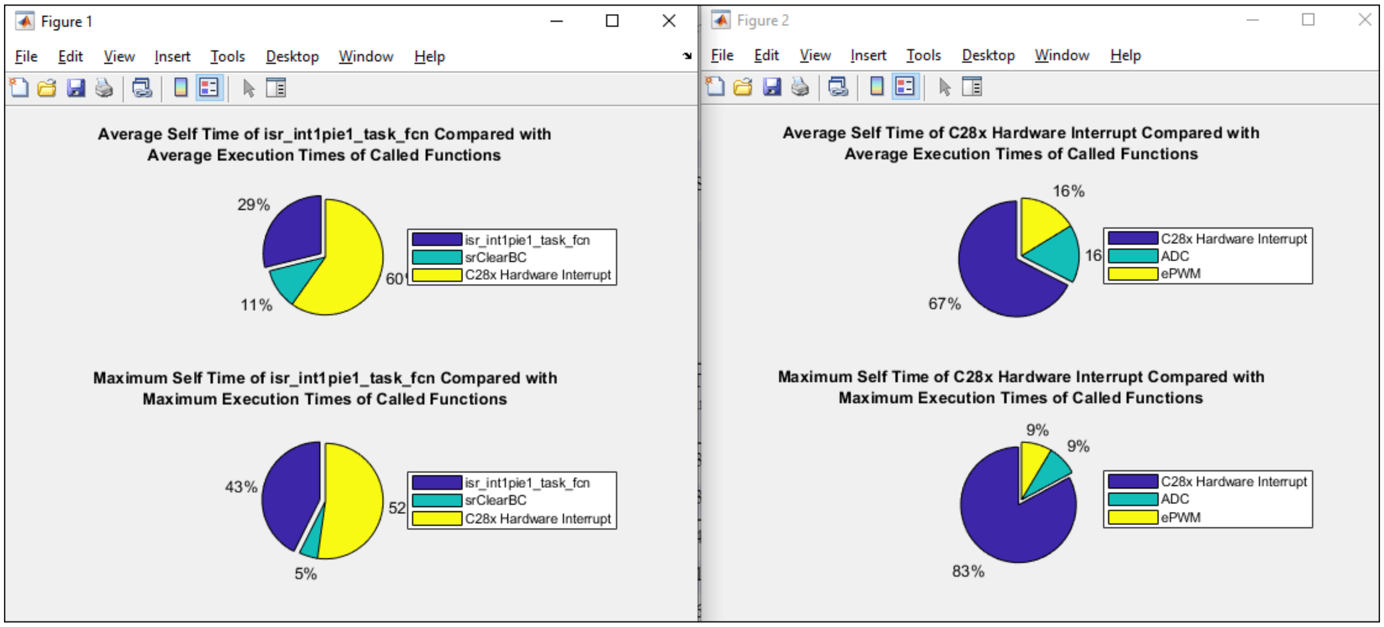
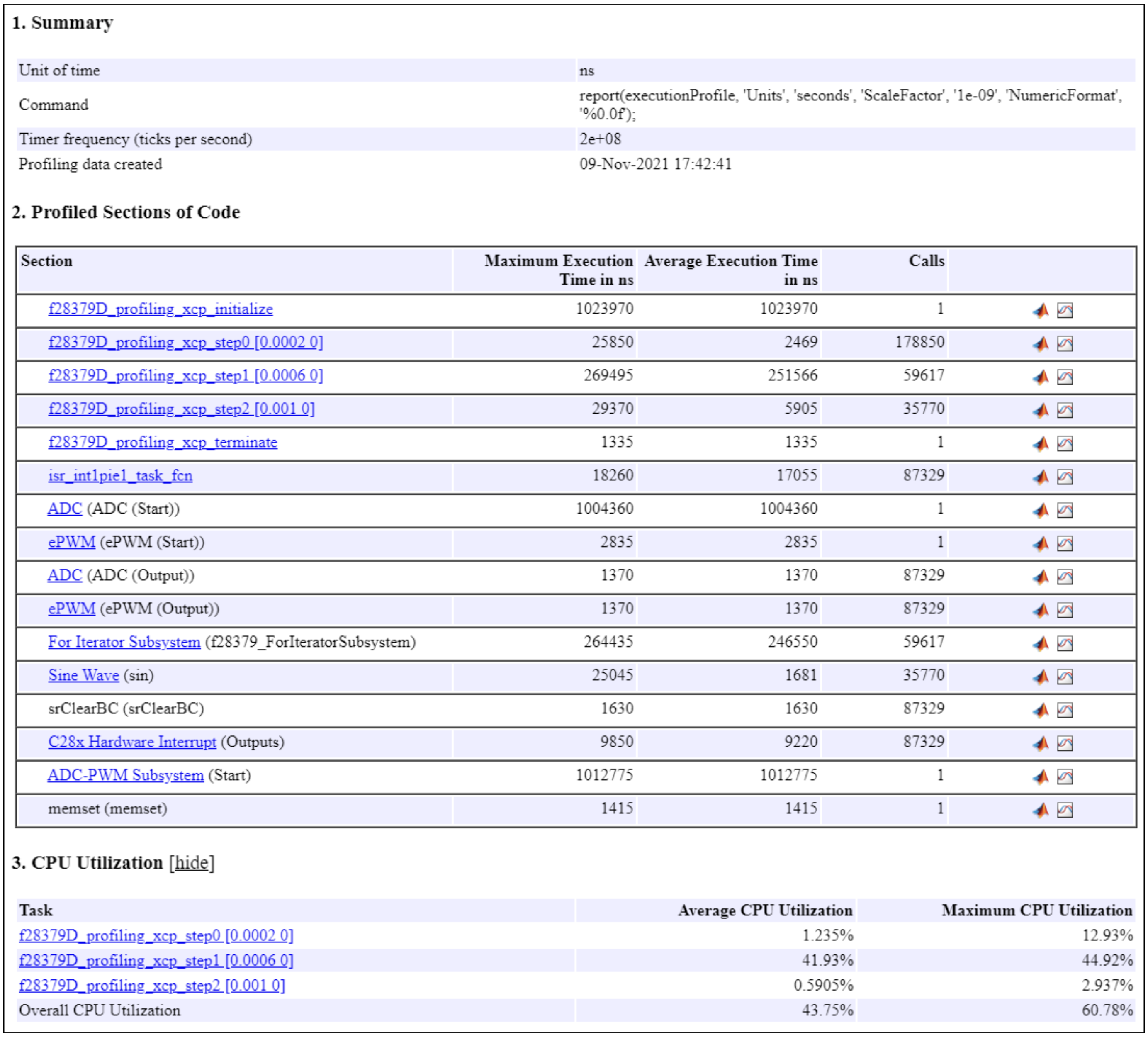


Figure 1
File Edit View Insert Tools Desktop Window Help
Average Self Time of isr_int1pie1_task_fcn Compared with Average Execution Times of Called Functions
29%
11%
60
isr_int1pie1_task_fcn
srClearBC
C28x Hardware Interrupt
Maximum Self Time of isr_int1pie1_task_fcn Compared with Maximum Execution Times of Called Functions
43%
5%
52
isr_int1pie1_task_fcn
srClearBC
C28x Hardware Interrupt
Figure 2
File Edit View Insert Tools Desktop Window Help
Average Self Time of C28x Hardware Interrupt Compared with Average Execution Times of Called Functions
16%
16
67%
C28x Hardware Interrupt
ADC
ePWM
Maximum Self Time of C28x Hardware Interrupt Compared with Maximum Execution Times of Called Functions
9%
9%
83%
C28x Hardware Interrupt
ADC
ePWM

Metrics only report



1. Summary

Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	2e+08
Profiling data created	09-Nov-2021 17:42:41

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Calls
f28379D_profiling_xcp_initialize	1023970	1023970	1
f28379D_profiling_xcp_step0 [0.0002 0]	25850	2469	178850
f28379D_profiling_xcp_step1 [0.0006 0]	269495	251566	59617
f28379D_profiling_xcp_step2 [0.001 0]	29370	5905	35770
f28379D_profiling_xcp_terminate	1335	1335	1
isr_int1pie1_task_fcn	18260	17055	87329
ADC (ADC (Start))	1004360	1004360	1
ePWM (ePWM (Start))	2835	2835	1
ADC (ADC (Output))	1370	1370	87329
ePWM (ePWM (Output))	1370	1370	87329
For Iterator Subsystem (f28379_ForIteratorSubsystem)	264435	246550	59617
Sine Wave (sin)	25045	1681	35770
srClearBC (srClearBC)	1630	1630	87329
C28x Hardware Interrupt (Outputs)	9850	9220	87329
ADC-PWM Subsystem (Start)	1012775	1012775	1
memset (memset)	1415	1415	1

3. CPU Utilization [hide]

Task	Average CPU Utilization	Maximum CPU Utilization
f28379D_profiling_xcp_step0 [0.0002 0]	1.235%	12.93%
f28379D_profiling_xcp_step1 [0.0006 0]	41.93%	44.92%
f28379D_profiling_xcp_step2 [0.001 0]	0.5905%	2.937%
Overall CPU Utilization	43.75%	60.78%

This report contains only summary metrics of Average/Maximum execution times of each task/function being profiled. Also, it contains the number of times each function/task is called during simulation.

Note: Profiling with **All data** or **Summary data** save options and **Detailed** option selected for Measure function execution times requires high bandwidth. This is because large amount of data must be sent in real-time. If the target does not have enough bandwidth to stream data in real-time, then select **Metrics Only** save option. Running profiling with this option stores summary profiling data on the target and will only send data to the host at the end of simulation.

Other Things to Try

Profile other Simulink models from the Texas Instruments C2000 blockset. Observe the time taken for implementing steps in the Simulink model that helps to improve its efficiency.

Related Topics

- “Code Execution Profiling on Texas Instruments C2000” on page 1-107

Signal Logging and Parameter Tuning in XCP External Mode with Packed Mode

This example shows how to perform parameter tuning and data logging with a Simulink® model enabled with Packed Mode running on Texas Instruments™ C2000™ targets.

Packed Mode leads to enhanced signal logging performance in models containing signals of high sample rates. However, the enhanced performance comes at an additional cost because Packed Mode requires higher RAM on the target as multiple samples need to be stored on the target before transmission.

Introduction

XCP External Mode is used to log signal data and tune parameters on the Texas Instruments C2000 targets in real-time. However, there is a limited bandwidth available for communication between the host and target. This limits the data that can be sent from the target. With Packed Mode, multiple data points of multiple signals are combined in a single packet which is sent over a transport. This reduces the overhead introduced by additional bytes of framing the packet.

Prerequisite

Complete the Getting Started with Embedded Coder Support Package for TI C2000 Processors video.

Note: Starting R2023a, the workflow for getting started tutorial for support package will continue to support in C2000™ Microcontroller Blockset.

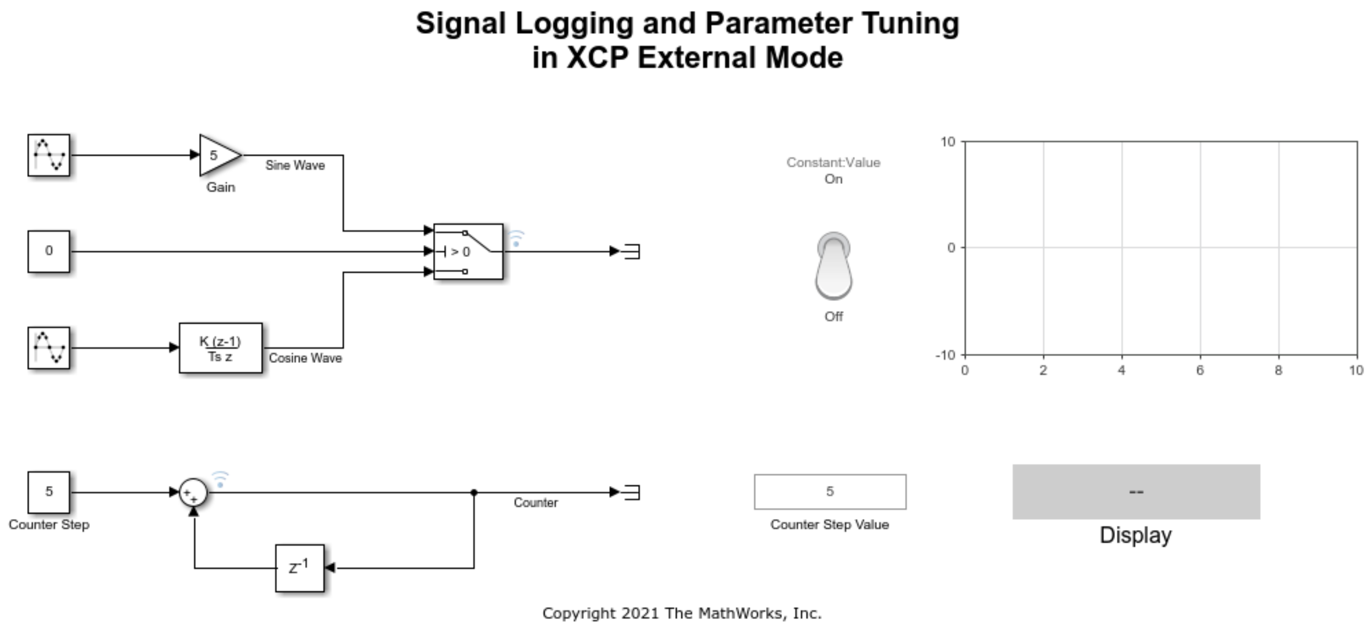
Required Hardware

Any Texas Instruments C2000 board

Model

To open the pre-configured model, run this command at the MATLAB prompt:

```
open_system('c28x_xcp_packed')
```



This is a multirate model where you log signals at the sample rates of 25 microseconds and 50 microseconds. It is preconfigured to run on the F28379D LaunchPad at a baud rate of 12 Mbps.

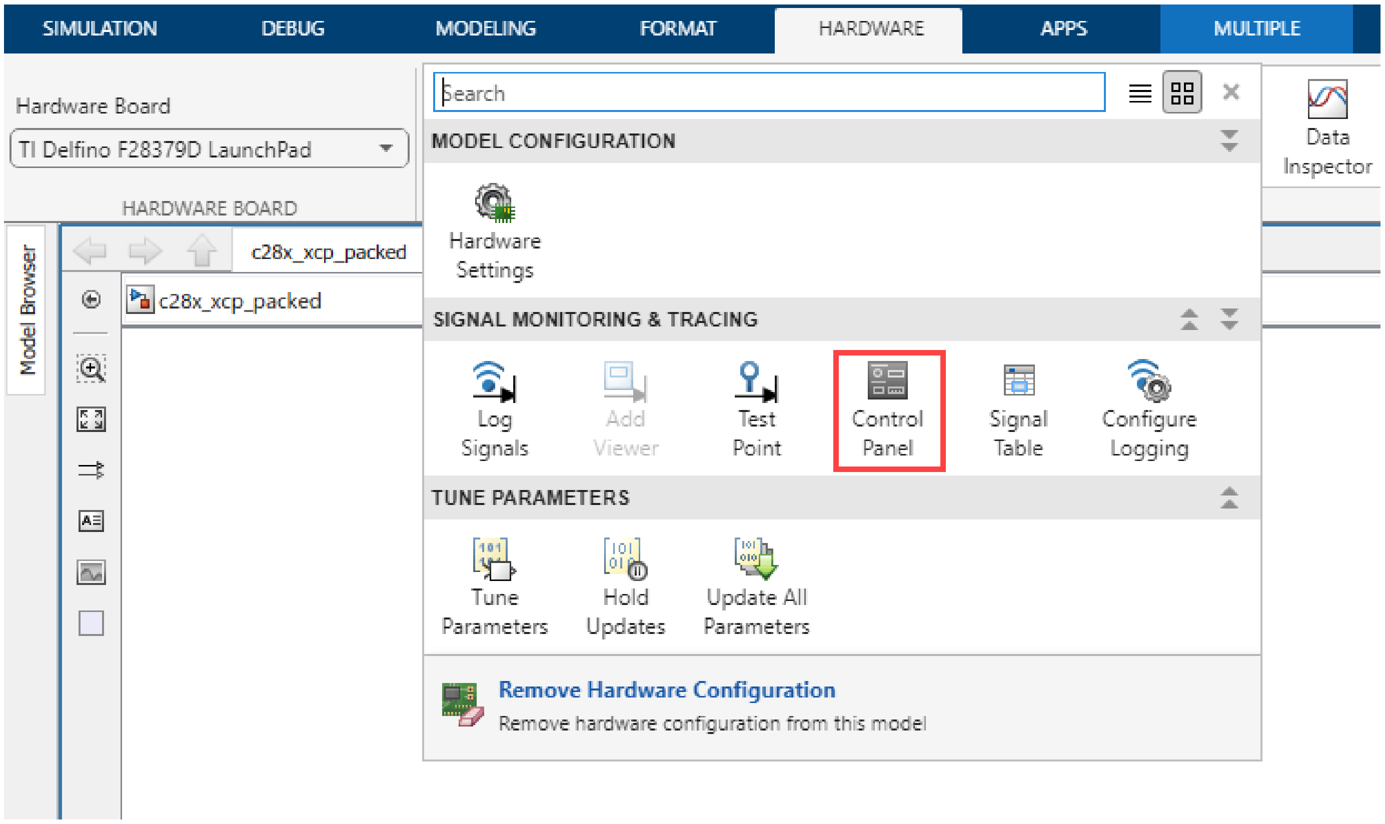
Configuring the Model

In this example, you will configure a Simulink model and enable packed mode.

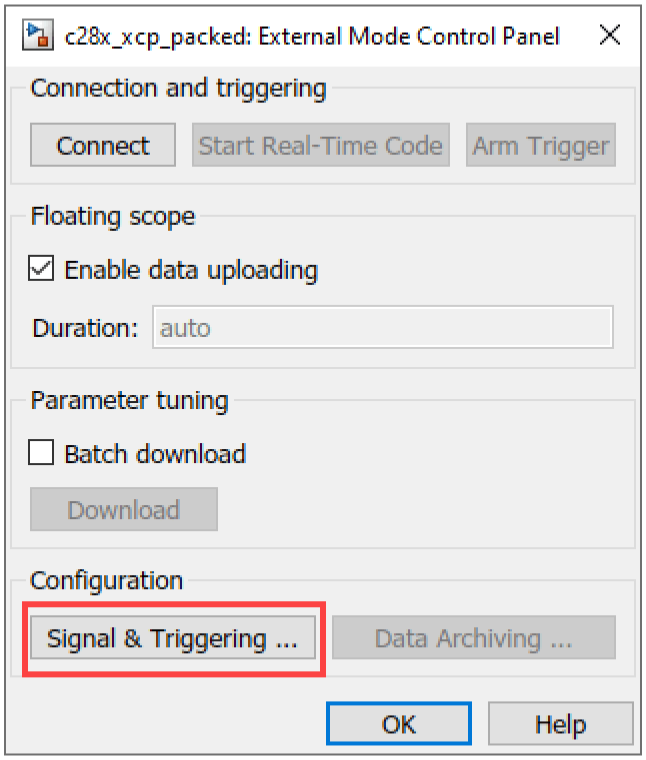
Note: These steps are not required in the pre-configured model. Perform these steps if you have changed the hardware or not using the pre-configured model.

1. Open the model.

2. In the **Hardware** tab, in the **Prepare** gallery, select **Control Panel**.



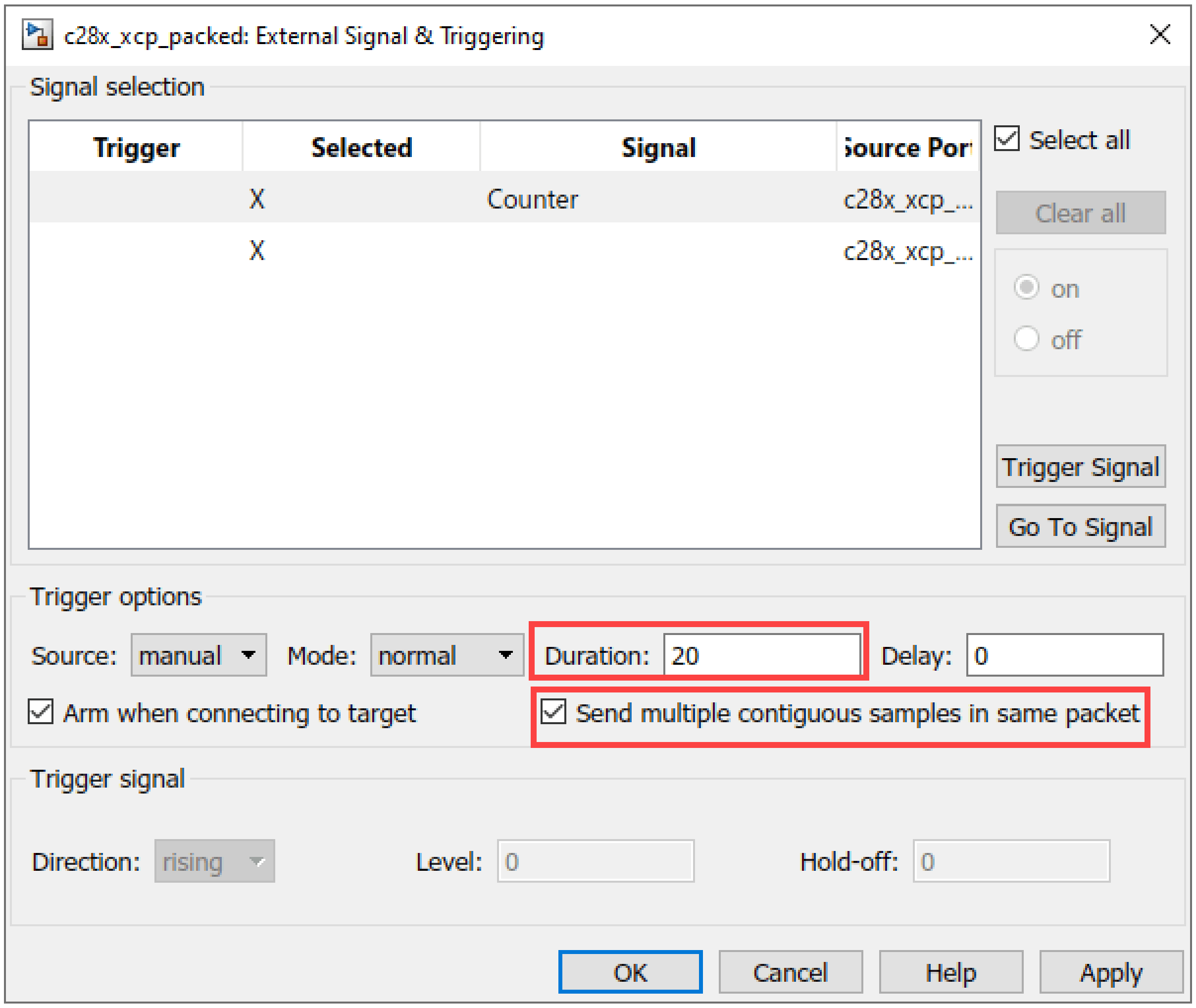
3. In the External Model Control Panel, click **Signal & Triggering**.



4. In the **External & Signal Triggering** dialog box, select **Send multiple contiguous samples in same packet** and enter a value for **Duration**.

This enables the XCP Packed mode and multiple contiguous samples are sent in a single packet whose length is the value specified for **Duration** parameter.

Note: Clearing the **Send multiple contiguous samples in same packet** option might lead to data drops. To optimize the performance and memory utilization, tune the value of **Duration** parameter.



c28x_xcp_packed: External Signal & Triggering

Signal selection

Trigger	Selected	Signal	Source Port
	X	Counter	c28x_xcp_...
	X		c28x_xcp_...

☑ Select all

Trigger options

Source: manual Mode: normal Duration: 20 Delay: 0

☑ Arm when connecting to target ☑ Send multiple contiguous samples in same packet

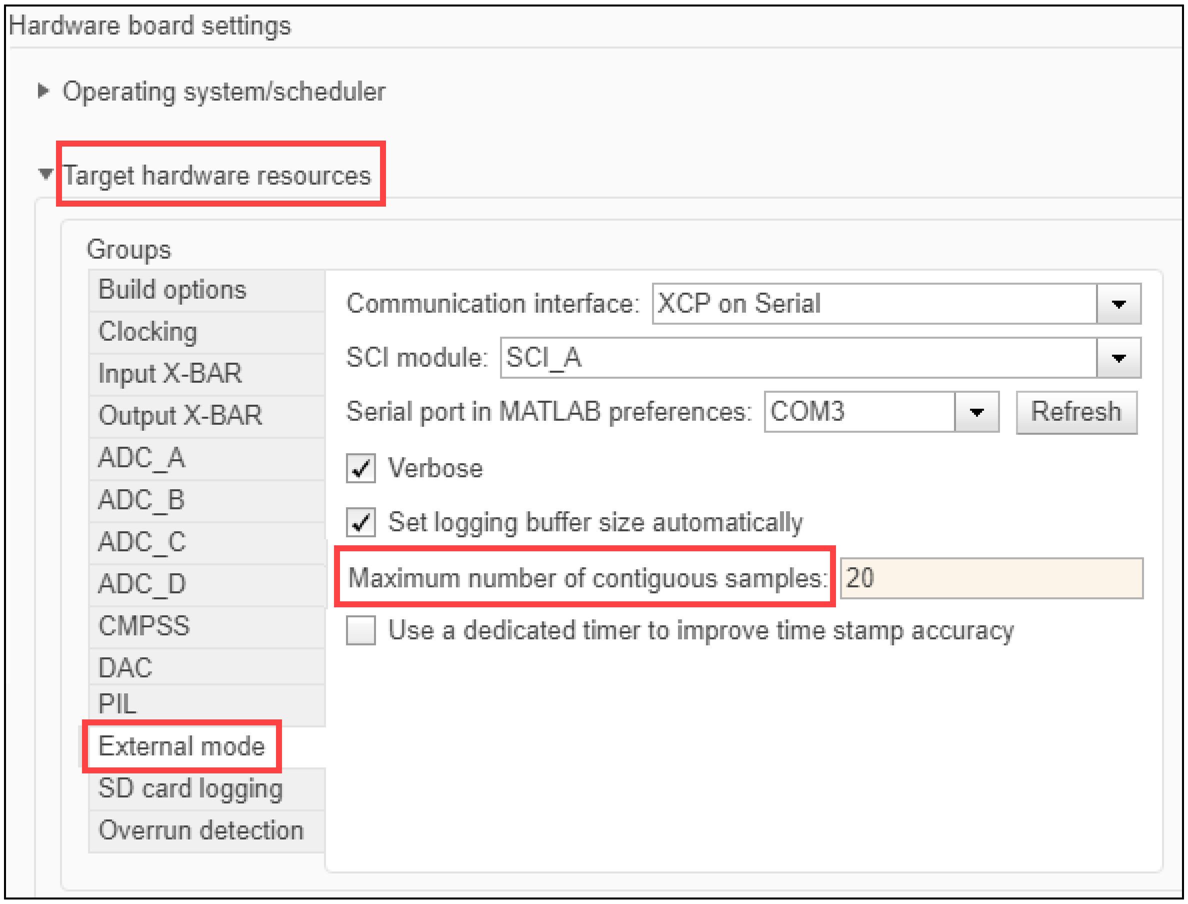
Trigger signal

Direction: rising Level: 0 Hold-off: 0

5. Click **Apply** and **OK**.

6. In the **Hardware** tab, click **Hardware Settings** to open the Configuration Parameters dialog box.

7. In the **Hardware Implementation** window, navigate to **Target hardware resources** > **External Mode**.


Hardware board settings

Operating system/scheduler

Target hardware resources

Groups

Build options
Clocking
Input X-BAR
Output X-BAR
ADC_A
ADC_B
ADC_C
ADC_D
CMPSS
DAC
PIL
External mode
SD card logging
Overrun detection

Communication interface: XCP on Serial

SCI module: SCI_A

Serial port in MATLAB preferences: COM3 Refresh

✓ Verbose

✓ Set logging buffer size automatically

Maximum number of contiguous samples: 20

☐ Use a dedicated timer to improve time stamp accuracy

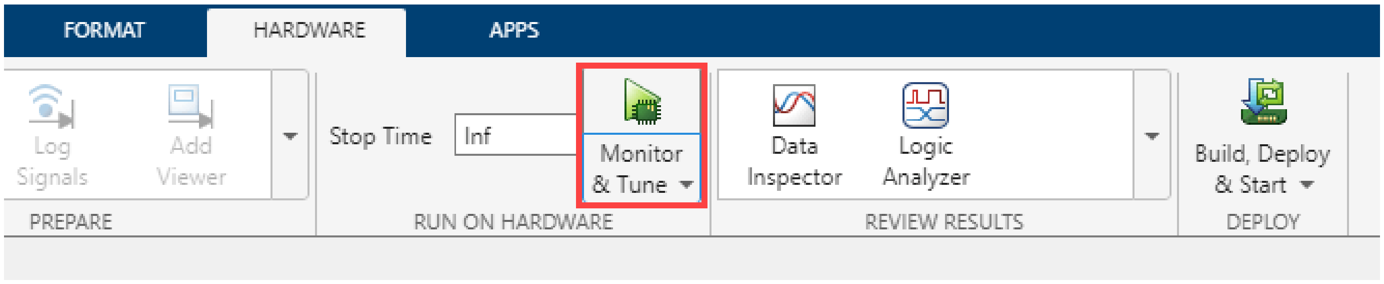
8. Set the parameter **Maximum number of contiguous samples** to a value greater than or equal to the value of **Duration** parameter set in External Mode Control Panel. This parameter dictates the maximum number of samples that can be filled in a single packet and the memory is allocated accordingly.

Note: The **Duration** parameter can be changed dynamically to any value that is less than or equal to **Maximum number of contiguous samples** without rebuilding the model.

9. Click **Apply** and **OK**.

Initiate Monitor and Tune Action for the Model

On the **Hardware** tab of the Simulink toolstrip, click **Monitor & Tune** to monitor signals and tune parameters.


FORMAT
HARDWARE
APPS
Log Signals
Add Viewer
PREPARE
Stop Time
Inf
Monitor & Tune
RUN ON HARDWARE
Data Inspector
Logic Analyzer
REVIEW RESULTS
Build, Deploy & Start
DEPLOY

Getting Started with Connected IO

In this example, you will learn how to to connect the Simulink® model directly to supported hardware for live I/O data exchange and blink an LED using connected IO in C2000™ Microcontroller Blockset.

Connected IO creates a communication interface that enables the Simulink model and the IO Server to communicate with each other. The model communicates with the hardware and no code generation is required.

Simulation with Connected IO is an intermediate step in the Model-Based Design workflow that bridges the gap between simulation and code generation by enabling Simulink to communicate with the hardware before deploying the model on the hardware.

Prerequisites

We recommend that you complete the following tutorials:

- Run `c2000setup` at the MATLAB® command prompt to complete the hardware setup.
- “Getting Started with Texas Instruments C2000 Microcontroller Blockset” on page 4-2
- “Communicate with Hardware Using Connected IO” on page 1-140

Required Hardware

- Any TI™ C2000™ board (This example is configured with TI Delfino F28379D Launchpad)
- You can use the input and output hardware peripherals corresponding to the respective source or sink blocks of the C2000™ Microcontroller Blockset that supports connected I/O in Normal mode of simulation.
- USB cable
- Connecting wires

Hardware Setup

1. Connect the TI™ C2000™ board to the host computer.

2. Connect ePWM1 pin (J4 pin 40) to ADC A0 pin (J3 pin 30)

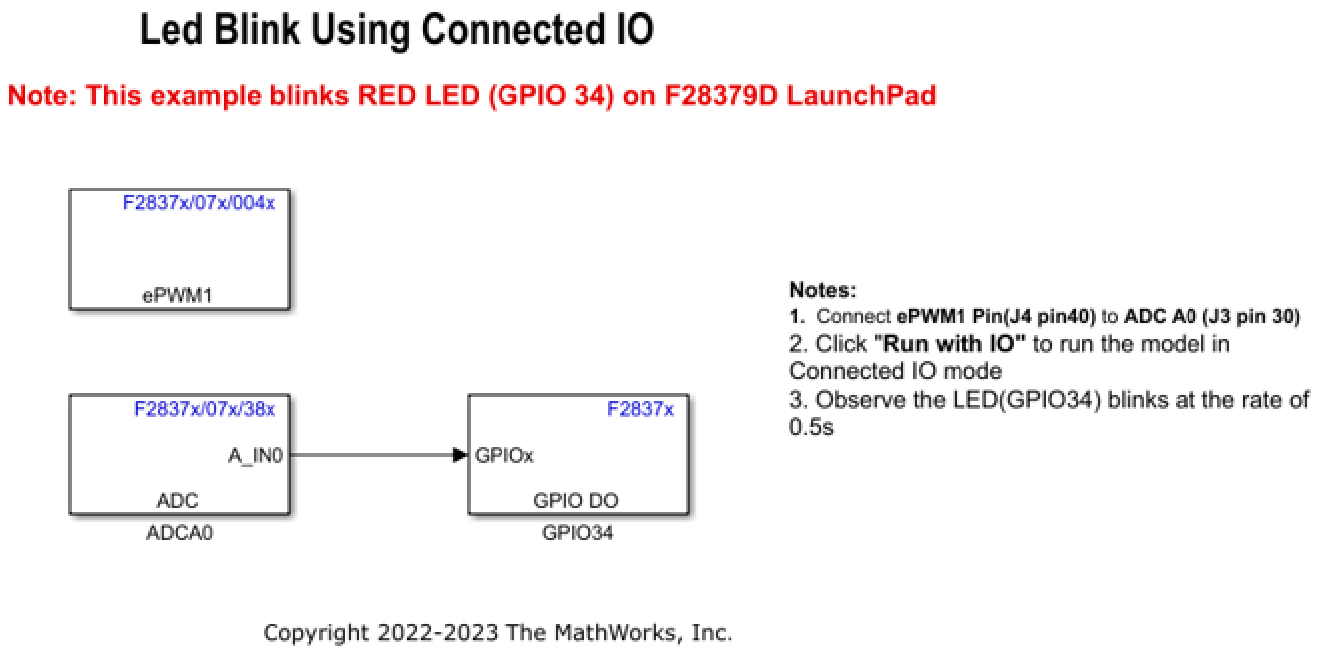
Simulink Model for Connected I/O

This example uses a preconfigured model from the C2000™ Microcontroller Blockset. In this model, the ePWM1 outputs a squared pulse of a particular duty ratio, which is being fed to an ADC channel. The ADC output in turn is fed to a digital output corresponding to the user LED(GPIO34) on the launchpad. On running the model in connected IO, you can observe the LED blink at the rate equal to the period of the ePWM output.

The LED on period is determined by the initial duty ration set in the ePWM block.

To open the model, run this command in the MATLAB (R) Command Window.

```
open_system('c2837xconnectedio.slx')
```


Led Blink Using Connected IO

Note: This example blinks RED LED (GPIO 34) on F28379D LaunchPad

Notes:
1. Connect **ePWM1 Pin(J4 pin40)** to **ADC A0 (J3 pin 30)**
2. Click **"Run with IO"** to run the model in Connected IO mode
3. Observe the LED(GPIO34) blinks at the rate of 0.5s

Copyright 2022-2023 The MathWorks, Inc.

Configure Parameters and Hardware Setup

Configure these parameters in the Configuration Parameters dialog box.

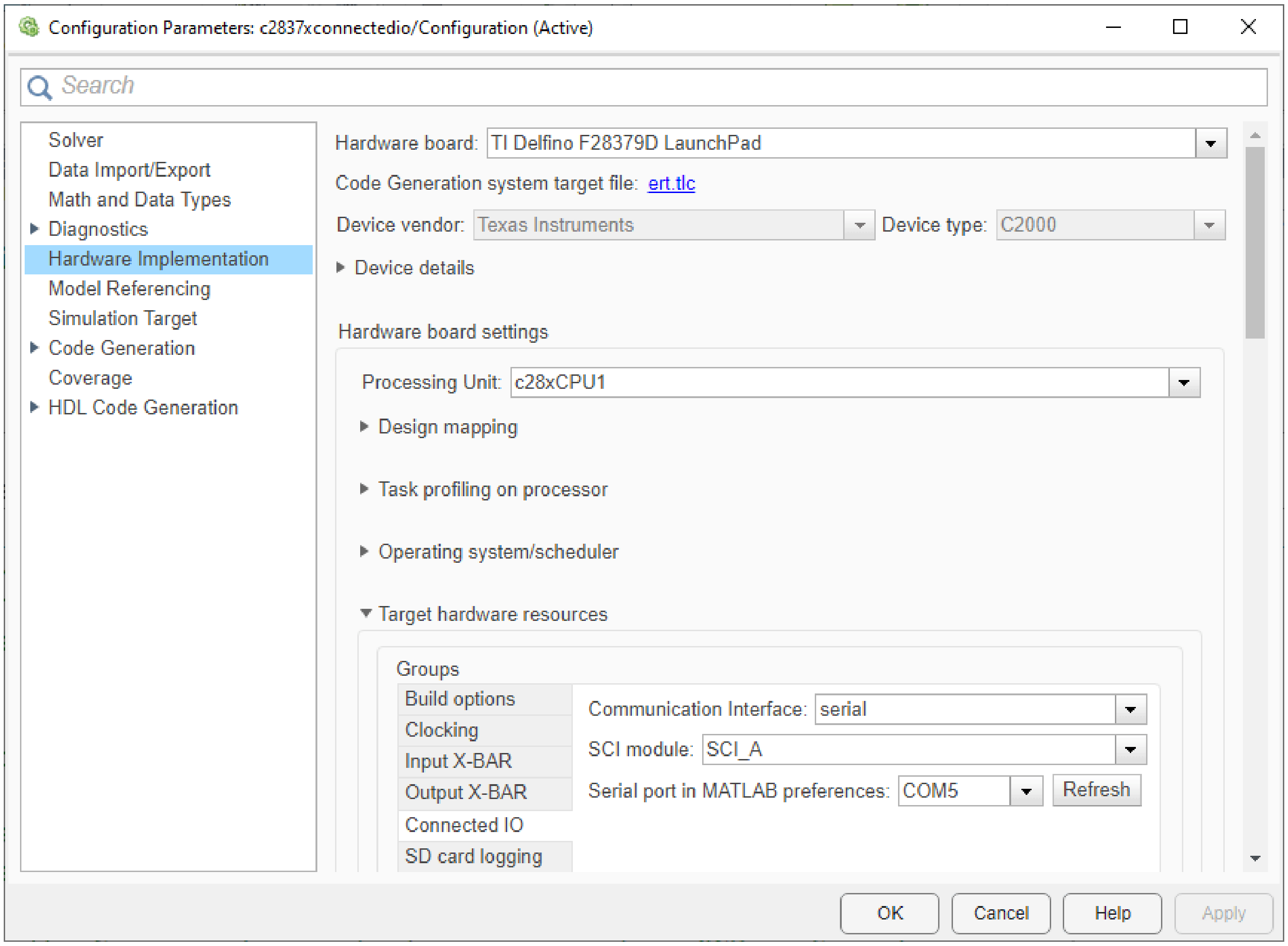
1. Open the Simulink® model.

2. On the **Modeling** tab, select **Model Settings**.

3. In the Configuration Parameters dialog box, select Hardware Implementation.

4. Set the Hardware board parameter to **TI Delfino F28379D Launchpad**. This selection populates the Hardware board settings parameters with the default values for the hardware.

5. Navigate to the **Hardware Implementation** > **Target Hardware Resources** > **Connected IO**


Configuration Parameters: c2837xconnectedio/Configuration (Active)

Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments Device type: C2000

Hardware board settings

Processing Unit: c28xCPU1

Target hardware resources

Groups: Build options, Clocking, Input X-BAR, Output X-BAR, Connected IO, SD card logging

Communication Interface: serial

SCI module: SCI_A

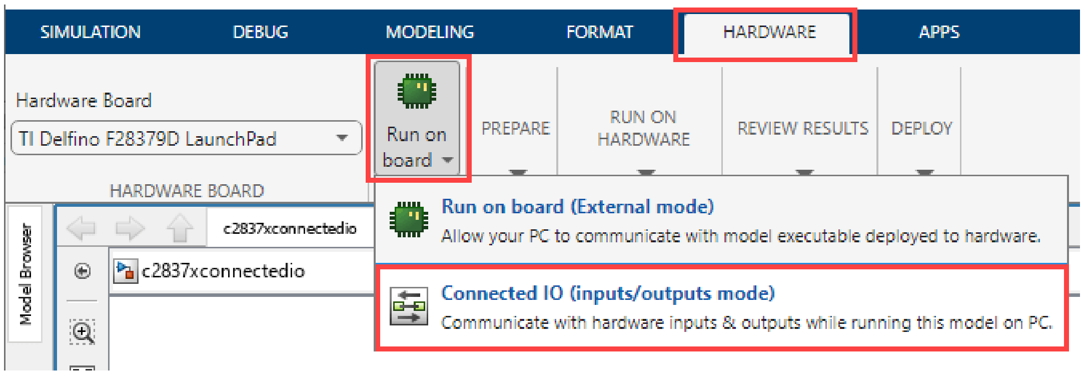
Serial port in MATLAB preferences: COM5

- Select the **Communication interface** as serial.
- Select the SCI module. By default SCI_A module is selected for Controlcards and Launchpads. For custom boards, select other serial modules to connect to FTDI.
- Select the Serial port in MATLAB preference with the COM port number associated with your target hardware.
- The Verbose option enables viewing the execution progress of the simulation on the Diagnostic Viewer and on the MATLAB Command Window.
- Navigate to **SCI_A** and specify the baud rate in Desired baud rate in bits/sec.

The default baud rate is 115200. You can increase the baud of the serial over USB of your Launchpad or controlCARD. On Launchpads and controlCARDs using FTDI 2232H, you can select any baud less than or equal to 6 Mbps, or exactly 9 or 12 Mbps. On controlCARDs using FTDI 2232D, you can select any baud less than or equal to 1.5 Mbps, or exactly 2 or 3 Mbps.

Run Simulink Model

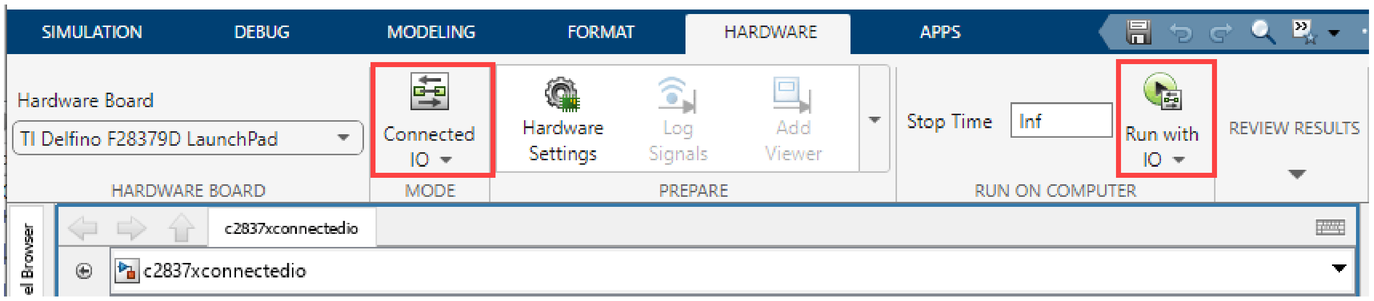
1. On the **Hardware** tab of the Simulink model, in the **Mode** section, select **Connected IO**.



Run on board (External mode)
Allow your PC to communicate with model executable deployed to hardware.

Connected IO (inputs/outputs mode)
Communicate with hardware inputs & outputs while running this model on PC.

2. Click **Run with IO**



3. Observe the output in the scope or SDI block

4. Observe the LED blinking

5. Results

Triple-Axis Tilt Calculation Using LIS3DH FIFO Data Ready Interrupt

This example shows how to use the FIFO data ready interrupt of LIS3DH linear accelerometer connected to an TI's™ C2000-based hardware to sense the tilt of the sensor. This is done using a downstream function-call subsystem that reads acceleration values from FIFO buffer sensor.

The sensor block is placed inside the downstream function-call subsystem. The output of the sensor block is sent to a MATLAB Function block where the data is averaged, and tilt is calculated.

For more information on tilt sensing using accelerometer values, see Using an Accelerometer for Inclination Sensing.

Supported Hardware

To run this example, you can use any TI C2000 hardware. For your convenience, the following hardware is preconfigured to execute the example:

- TI Delfino F28379D

Prerequisites

Before you start this example, complete the "Getting Started with Texas Instruments C2000 Microcontroller Blockset" on page 4-2.

Required Hardware

- Supported TI C2000 hardware
- LIS3DH sensor
- USB cable
- Breadboard wires

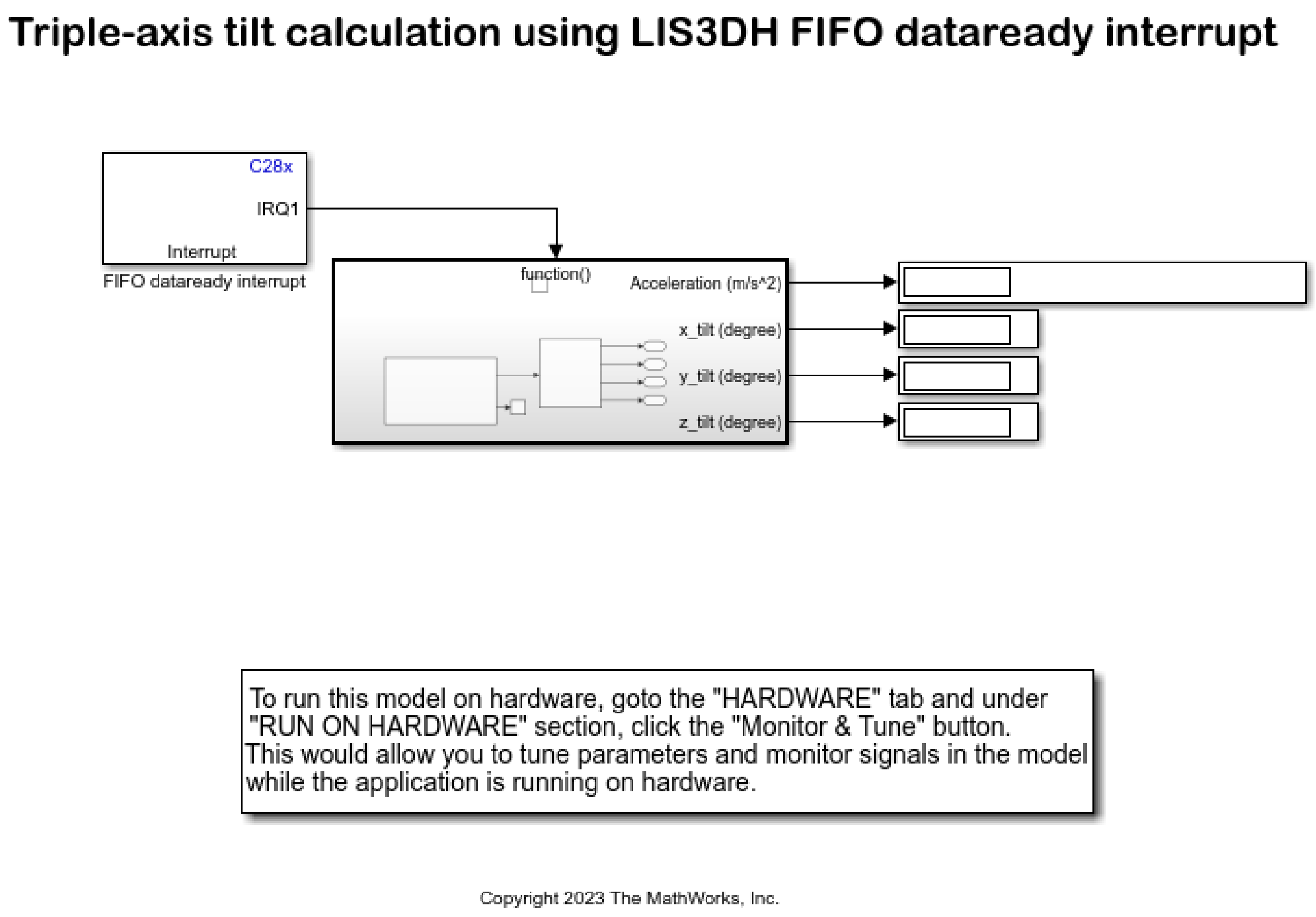
Hardware Connection

- Connect the INT1 pin of the LIS3DH sensor to pin 60 on the TI C2000 hardware.
- Connect the SDA, SCL, 3.3V, and GND pin of the TI C2000 hardware to the respective pins on the LIS3DH sensor.

Hardware Configuration in the Model

To open the model used in this example, enter the following command at MATLAB command prompt:

```
open_system('c28x_i2c_lis3dh_tiltdetection_fifointerrupt.slx');
```



Triple-axis tilt calculation using LIS3DH FIFO dataready interrupt

To run this model on hardware, goto the "HARDWARE" tab and under "RUN ON HARDWARE" section, click the "Monitor & Tune" button. This would allow you to tune parameters and monitor signals in the model while the application is running on hardware.

Copyright 2023 The MathWorks, Inc.

The model is preconfigured to work with TI Delfino F28379D LaunchPad hardware. If you are using a different C2000 hardware, change the hardware board by performing the following steps:

1. Navigate to **Modeling** > **Model Settings** to open the Configuration Parameters dialog box.

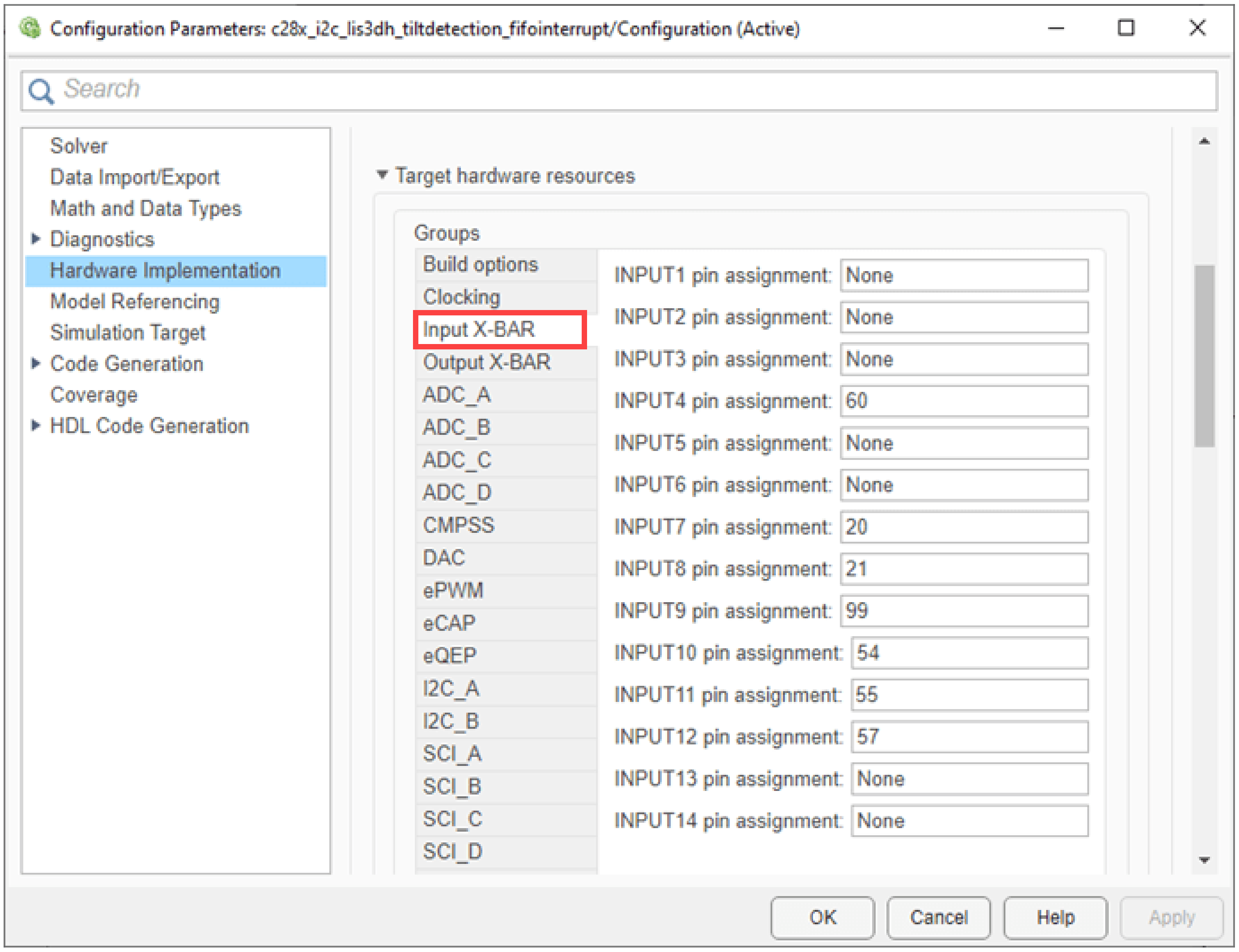
2. Open the **Hardware Implementation** pane, and from the **Hardware board** list, select the type of TI 2000 hardware that you are using.

3. Click **Apply**. Click **OK** to close the dialog box.

Configure the GPIO Pin for External Interrupt

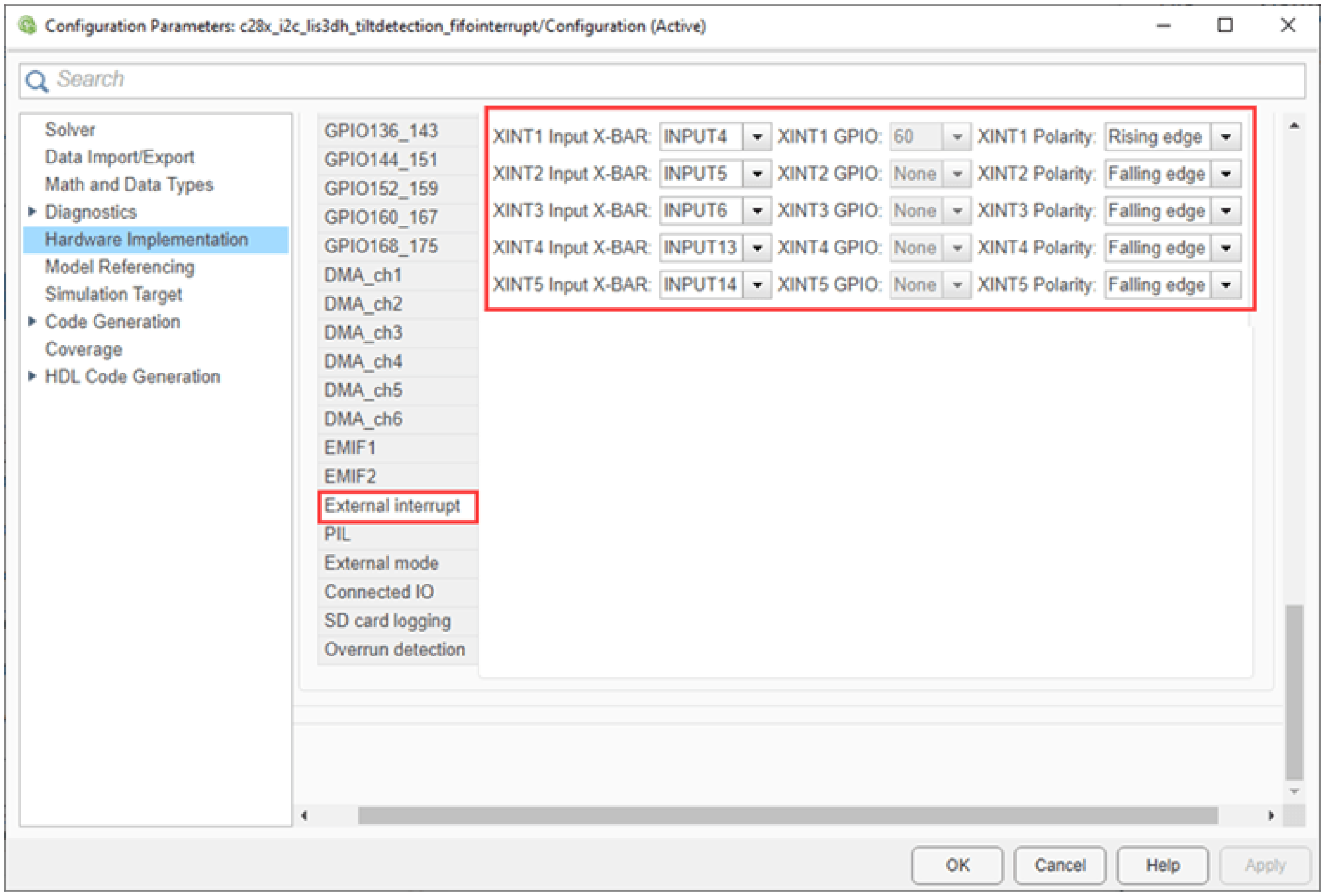
1. Navigate to **Modeling** > **Model Settings** > **Hardware Implementation** > **Target hardware resources**.

2. Click **Input X-BAR** and assign the pin to configure as interrupt.


Configuration Parameters: c28x_i2c_lis3dh_tiltdetection_fifointerrupt/Configuration (Active)

INPUT1 pin assignment:	None
INPUT2 pin assignment:	None
INPUT3 pin assignment:	None
INPUT4 pin assignment:	60
INPUT5 pin assignment:	None
INPUT6 pin assignment:	None
INPUT7 pin assignment:	20
INPUT8 pin assignment:	21
INPUT9 pin assignment:	99
INPUT10 pin assignment:	54
INPUT11 pin assignment:	55
INPUT12 pin assignment:	57
INPUT13 pin assignment:	None
INPUT14 pin assignment:	None

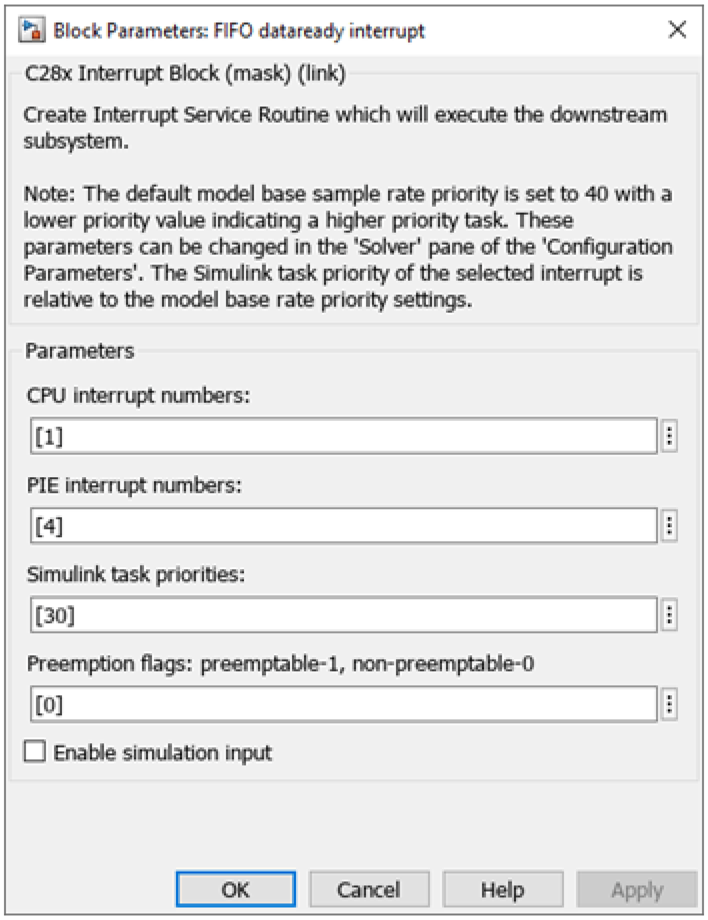
3. The pin selected is reflected in **External interrupt** tab. Select the polarity of the interrupt pin in this tab.


Configuration Parameters: c28x_i2c_lis3dh_tiltdetection_fifointerrupt/Configuration (Active)

XINT1 Input X-BAR: INPUT4	XINT1 GPIO: 60	XINT1 Polarity: Rising edge
XINT2 Input X-BAR: INPUT5	XINT2 GPIO: None	XINT2 Polarity: Falling edge
XINT3 Input X-BAR: INPUT6	XINT3 GPIO: None	XINT3 Polarity: Falling edge
XINT4 Input X-BAR: INPUT13	XINT4 GPIO: None	XINT4 Polarity: Falling edge
XINT5 Input X-BAR: INPUT14	XINT5 GPIO: None	XINT5 Polarity: Falling edge

4. Click **Apply**. Click **OK** to close the dialog box.

5. In the C28x Interupt Block parameters, specify the values for **CPU interrupt numbers**, **PIE interrupt numbers**, and **Simulink task priorities**, and **Preemption flags**.


Block Parameters: FIFO dataready interrupt

C28x Interrupt Block (mask) (link)

Create Interrupt Service Routine which will execute the downstream subsystem.

Note: The default model base sample rate priority is set to 40 with a lower priority value indicating a higher priority task. These parameters can be changed in the 'Solver' pane of the 'Configuration Parameters'. The Simulink task priority of the selected interrupt is relative to the model base rate priority settings.

Parameters

CPU interrupt numbers: [1]

PIE interrupt numbers: [4]

Simulink task priorities: [30]

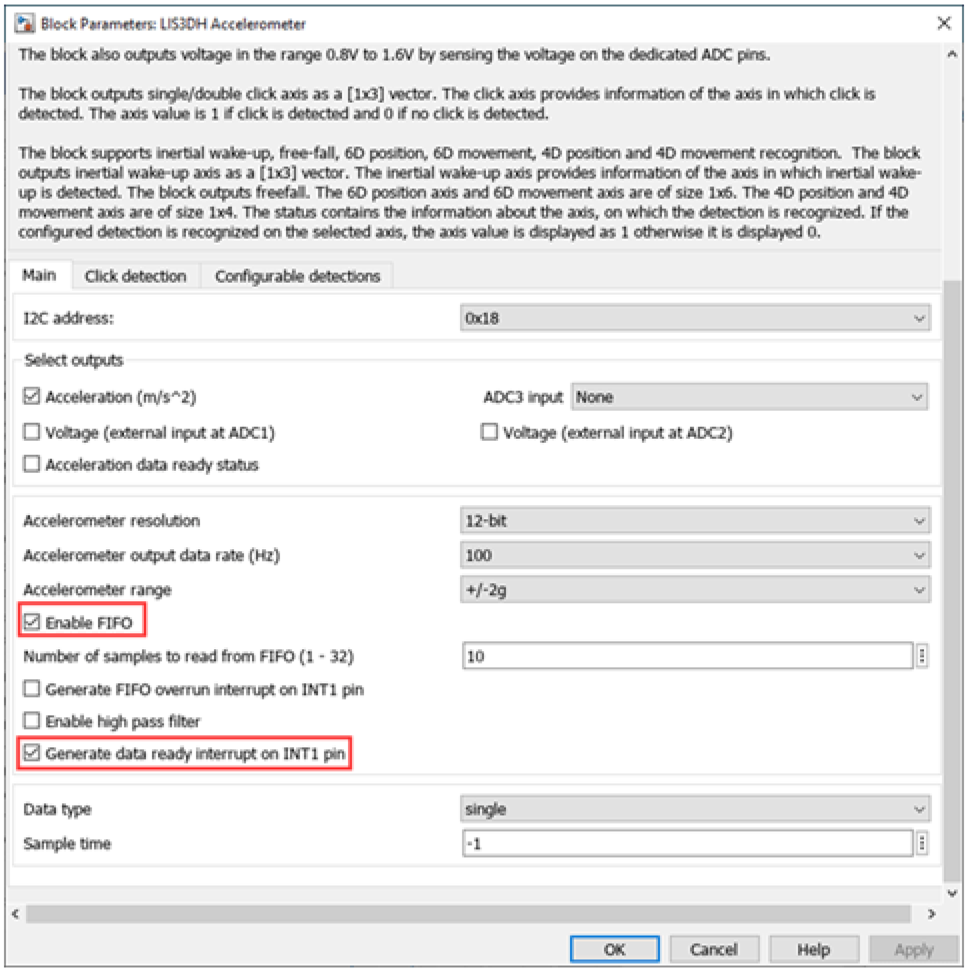
Preemption flags: preemptable-1, non-preemptable-0: [0]

☐ Enable simulation input

Configure the FIFO Data Ready Interrupt on LIS3DH for Interrupt Service Routine (ISR)

1. The LIS3DH sensor block is placed in the downstream function-call subsystem. The output of the block is connected to MATLAB function block. The FIFO data acquired by the MATLAB function is averaged, and tilt of each axis is calculated.

2. Ensure that the **Generate data ready interrupt on INT1 pin** parameter is selected, and also ensure that FIFO is enabled and Number of samples to read from FIFO (1-32) is provided. In this example, data update is expected at 100 Hz (the specified ODR) and Number of samples to read from FIFO is 10.



The block also outputs voltage in the range 0.8V to 1.6V by sensing the voltage on the dedicated ADC pins.

The block outputs single/double click axis as a [1x3] vector. The click axis provides information of the axis in which click is detected. The axis value is 1 if click is detected and 0 if no click is detected.

The block supports inertial wake-up, free-fall, 6D position, 6D movement, 4D position and 4D movement recognition. The block outputs inertial wake-up axis as a [1x3] vector. The inertial wake-up axis provides information of the axis in which inertial wake-up is detected. The block outputs freefall. The 6D position axis and 6D movement axis are of size 1x6. The 4D position and 4D movement axis are of size 1x4. The status contains the information about the axis, on which the detection is recognized. If the configured detection is recognized on the selected axis, the axis value is displayed as 1 otherwise it is displayed 0.

I2C address: 0x18

Select outputs

☑ Acceleration (m/s^2)

ADC3 input: None

☐ Voltage (external input at ADC1)

☐ Voltage (external input at ADC2)

☐ Acceleration data ready status

Accelerometer resolution: 12-bit

Accelerometer output data rate (Hz): 100

Accelerometer range: +/-2g

☑ Enable FIFO

Number of samples to read from FIFO (1 - 32): 10

☐ Generate FIFO overrun interrupt on INT1 pin

☐ Enable high pass filter

☑ Generate data ready interrupt on INT1 pin

Data type: single

Sample time: -1

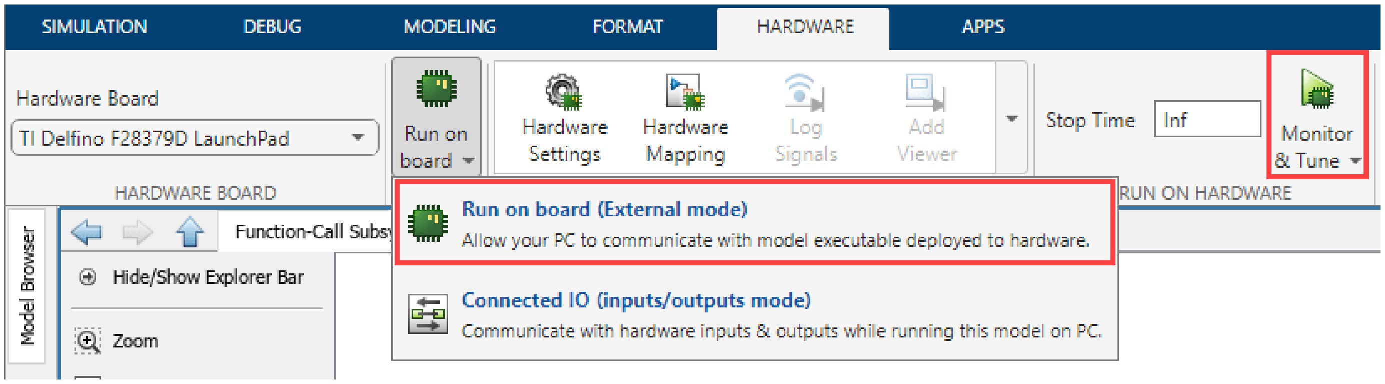
3. Connect the INT1 pin to GPIO 60 of TI C2000 hardware.

Initiate Monitor and Tune Action for the Model

During Monitor and Tune action, the model is deployed as a C code on the hardware. The code obtains real-time data from the hardware. The data acquisition and parameter tuning are done while the application is running on the hardware.

Note: Ensure that the TI 2000 hardware you are using has sufficient memory to run the application on hardware.

1. On the **Hardware** tab, in the **Mode** section, select **Run on board (External Mode)** and then click **Monitor & Tune**.



Run on board (External mode)
Allow your PC to communicate with model executable deployed to hardware.

Connected IO (inputs/outputs mode)
Communicate with hardware inputs & outputs while running this model on PC.

The lower-left corner of the model window displays status while Simulink prepares, downloads, and runs the Simulink model on the hardware. During simulation, the pin 60 on TI C2000 hardware generates an interrupt on every rising edge of the signal, which triggers the function-call subsystem.

2. Move the sensor and verify the changing values at each time step by double-clicking the Display blocks in the **Display data** area of the model.

???

Trigger Downstream Function-Call Subsystem Using C2000 External Interrupt Block with Single Tap Event on BMI160 Sensor

This example shows how to use the C2000™ Microcontroller Blockset to trigger a downstream function-call in Monitor and Tune action when Single tap event occurs on BMI160 sensor using a TI C2000 External Interrupt block. This example model for monitor and tune simulation, shows the capabilities of the External Interrupt block during single tap event on BMI160 sensor.

When you use the Monitor and Tune (External mode) action, a single tap event occurring on BMI160 sensor drives downstream subsystem through the digital pin externally. For every rising edge of this input pulse signal, the downstream function-call subsystem is triggered.

Prerequisite

If you are new to Simulink, we recommend completing the:

- Interactive Simulink® Tutorial.
- Reading the Getting Started section of the Simulink documentation
- Running Simulink Getting Started example.
-

Required Hardware

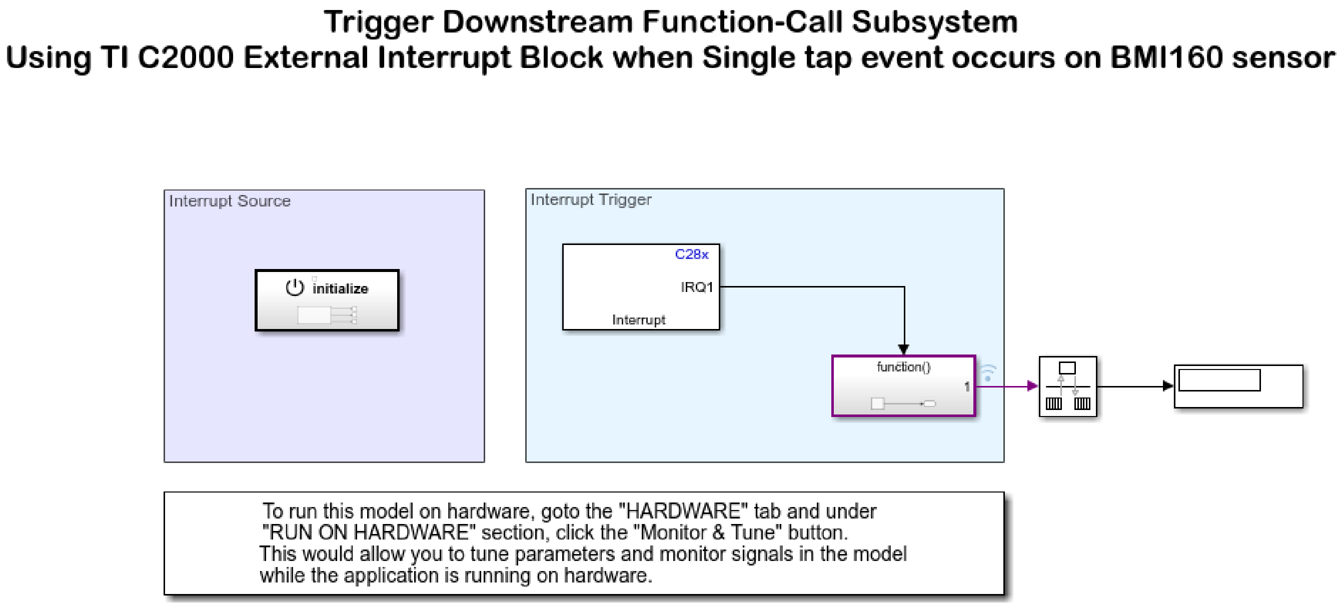
- Texas Instruments™ F280049C LaunchPad
- BMI160 Boosterpack
- Micro USB cable
- Connecting wires

Configure Simulink Model for Counting Single Taps on BMI160 Sensor

For the monitor and tune (external mode) action, this example uses a preconfigured Simulink model from the C2000® Microcontroller Blockset.

To open the Simulink model, run this command at the MATLAB Command Window:

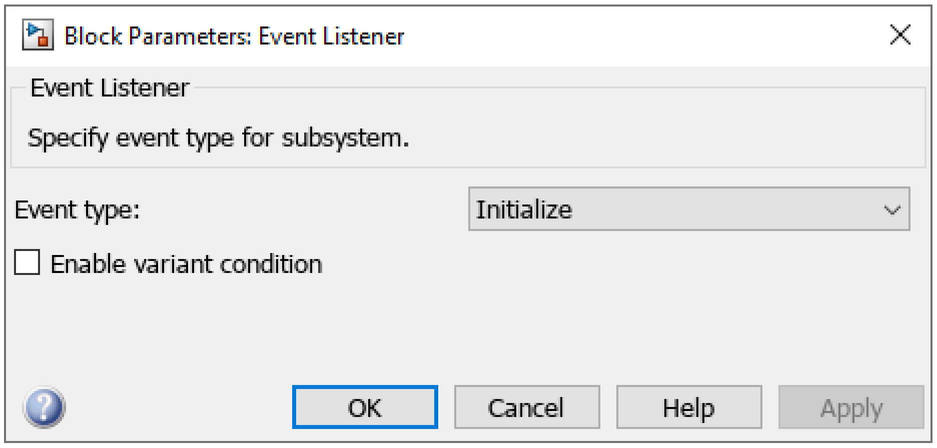
```
open_system('c28x_i2c_singletap_interrupt_detection.slx');
```


Trigger Downstream Function-Call Subsystem
Using TI C2000 External Interrupt Block when Single tap event occurs on BMI160 sensor
Interrupt Source
initialize
Interrupt Trigger
C28x
IRQ1
Interrupt
function()
1

To run this model on hardware, goto the "HARDWARE" tab and under "RUN ON HARDWARE" section, click the "Monitor & Tune" button. This would allow you to tune parameters and monitor signals in the model while the application is running on hardware.

Configure the following parameters:

1. In the **Interrupt Source** area, Initialize Function subsystem consists of BMI160 block and Event Listener blocks. In the Event Listener block, ensure that **Event type** is set to **Initialize**.


Block Parameters: Event Listener

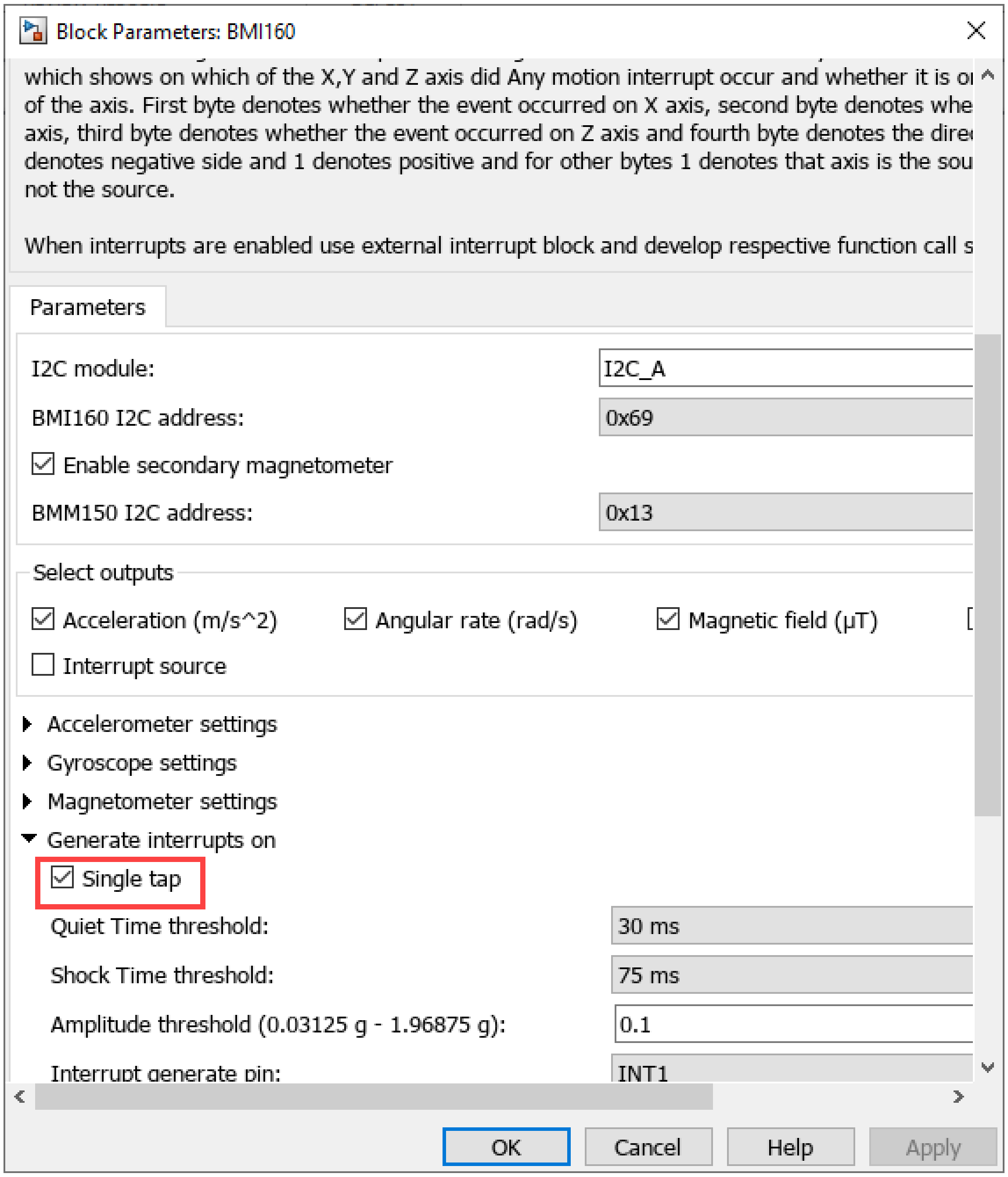
Event Listener

Specify event type for subsystem.

Event type: Initialize

☐ Enable variant condition

2. In the BMI160 Sensor, ensure that **Single tap** option is selected for **Generate interrupts on** parameter.


Block Parameters: BMI160

which shows on which of the X,Y and Z axis did Any motion interrupt occur and whether it is o
of the axis. First byte denotes whether the event occurred on X axis, second byte denotes whe
axis, third byte denotes whether the event occurred on Z axis and fourth byte denotes the dire
denotes negative side and 1 denotes positive and for other bytes 1 denotes that axis is the sou
not the source.

When interrupts are enabled use external interrupt block and develop respective function call s

Parameters

I2C module: I2C_A

BMI160 I2C address: 0x69

☑ Enable secondary magnetometer

BMM150 I2C address: 0x13

Select outputs

☑ Acceleration (m/s^2) ☑ Angular rate (rad/s) ☑ Magnetic field (μT)

☐ Interrupt source

Accelerometer settings

Gyroscope settings

Magnetometer settings

Generate interrupts on

☑ Single tap

Quiet Time threshold: 30 ms

Shock Time threshold: 75 ms

Amplitude threshold (0.03125 g - 1.96875 g): 0.1

Interrupt generate pin: INT1

3. Configure the External Interrupt block parameters to its default values. Ensure that the **Add simulation input port** option is cleared.

Signal Monitoring and Parameter Tuning

To run the model for signal monitoring and parameter tuning, on the **Hardware** tab, in the **Mode** section, select **Run on board** and then click **Monitor & Tune** to start signal monitoring and parameter tuning.

During simulation, the single tap event on the sensor generates interrupt on the C2000® Microcontroller Blockset. Observe the increment of the counter values in Display block for every rising edge of the taps.

You can notice a delay in the downstream function-call.

You can also view the output of the manual switch and rate transition block output on the Simulation Data Inspector and Logic Analyzer.

Control Law Accelerator in DC-DC Power Conversion

This example shows how to manage the voltage mode control (VMC) algorithm for a closed loop DC-DC power conversion system by using the Control Law Accelerator (CLA). An auxiliary software current protection loop is managed by C2000 CPU. In the TMS320F28379D and similar processor families, the CLAs execute the hard-realtime portions of the algorithm can connect with hardware perhipherals such as ADC's and PWM's. The C2000 CPU is better suited for hard real-time portions of the algorithm that require multitasking, handling of asynchronous events, and communication with other cores. While this example extends the DC-DC buck converter developed in the“DC-DC Buck Converter Using MCU” (SoC Blockset) example, the strategy presented can be applied to any design where a resource intensive hard real-time process executes on the CLA and real-time multi-tasking processes run on C2000 CPU.

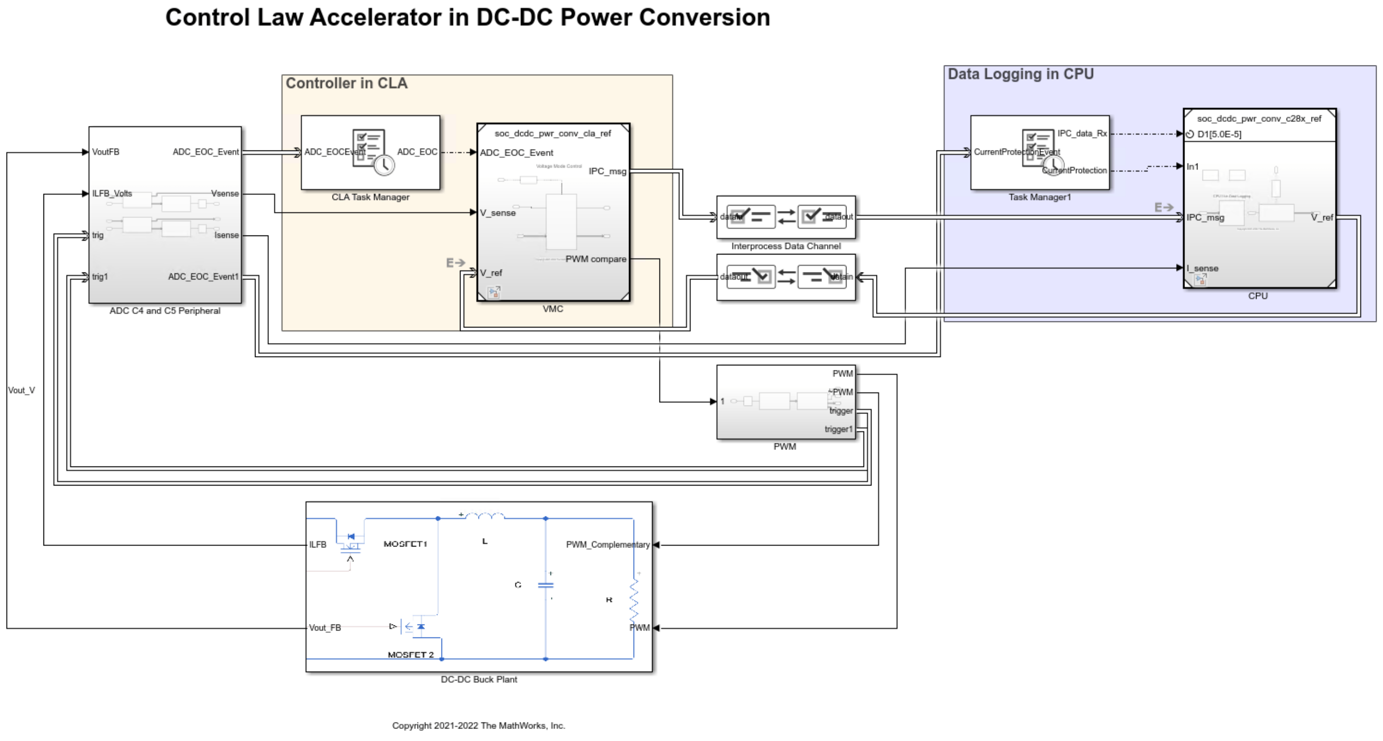
Required Hardware

- TI Delfino F28379D LaunchPad
- TI BOOSTXL-BUCKCONV kit

Voltage Mode Control Algorithm on CLA

This model shows a DC-DC power conversion system with the voltage mode control (VMC) algorithm executing on the CLA. The model assigns a separate Task Manager (SoC Blockset) block to the CLA and C2000 CPU reference models. For more information about scheduling tasks on the CLA, see the CLA Task Manager documentation. Open the model.

```
open_system('soc_dcdc_pwr_conv');
```



Top-level model `soc_dcdc_pwr_conv` does not build by having the **Processing Unit** parameter to `none`. In each reference model, set the **Processing Unit** parameter to a specific processor. On the **System on Chip** tab, click **Hardware Settings** to open the **Configuration Parameters** window.

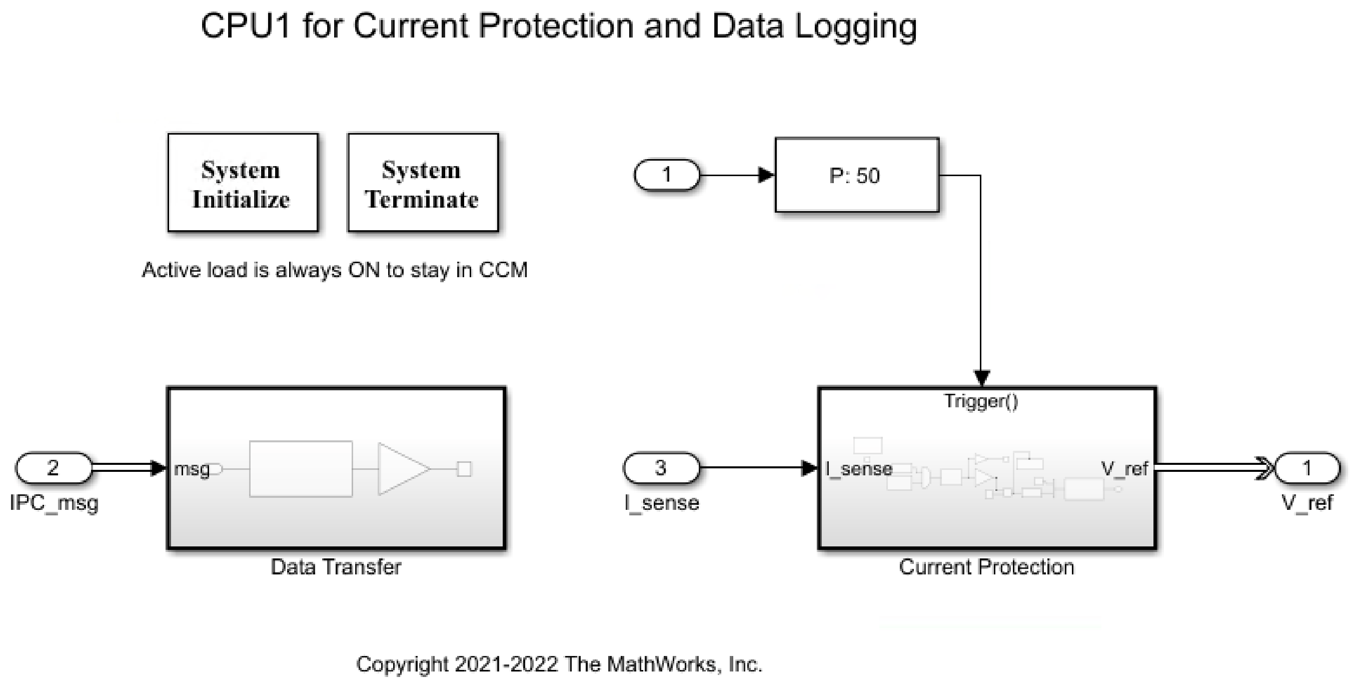
- For the CLA reference model, set **Processing Unit** to `CPU1CLA1` in the **Hardware Implementation** tab.
- For the C2000 CPU reference model, set **Processing Unit** to `c28xCPU1` in the **Hardware Implementation** tab.

Select the appropriate interrupt source for the CLA in the Task Mapper tool. This example uses `ADCCINT4` as the ADC interrupt to the CLA. For more information about designing the VMC and peripheral configuration, see “DC-DC Buck Converter Using MCU” (SoC Blockset).

The CLA has restricted regional memory access. Specify the memory sections for inports, outports, signals, states, and internal data by using the Code Mappings Editor – C (Embedded Coder) tool in the CLA reference model. You can ensure that the CLA has appropriate access to these sections by specifying the `initialize`, `execute`, and `terminate` function program spaces. By default, this model has appropriate code mapping for the CLA.

Current Protection Algorithm on C2000 CPU

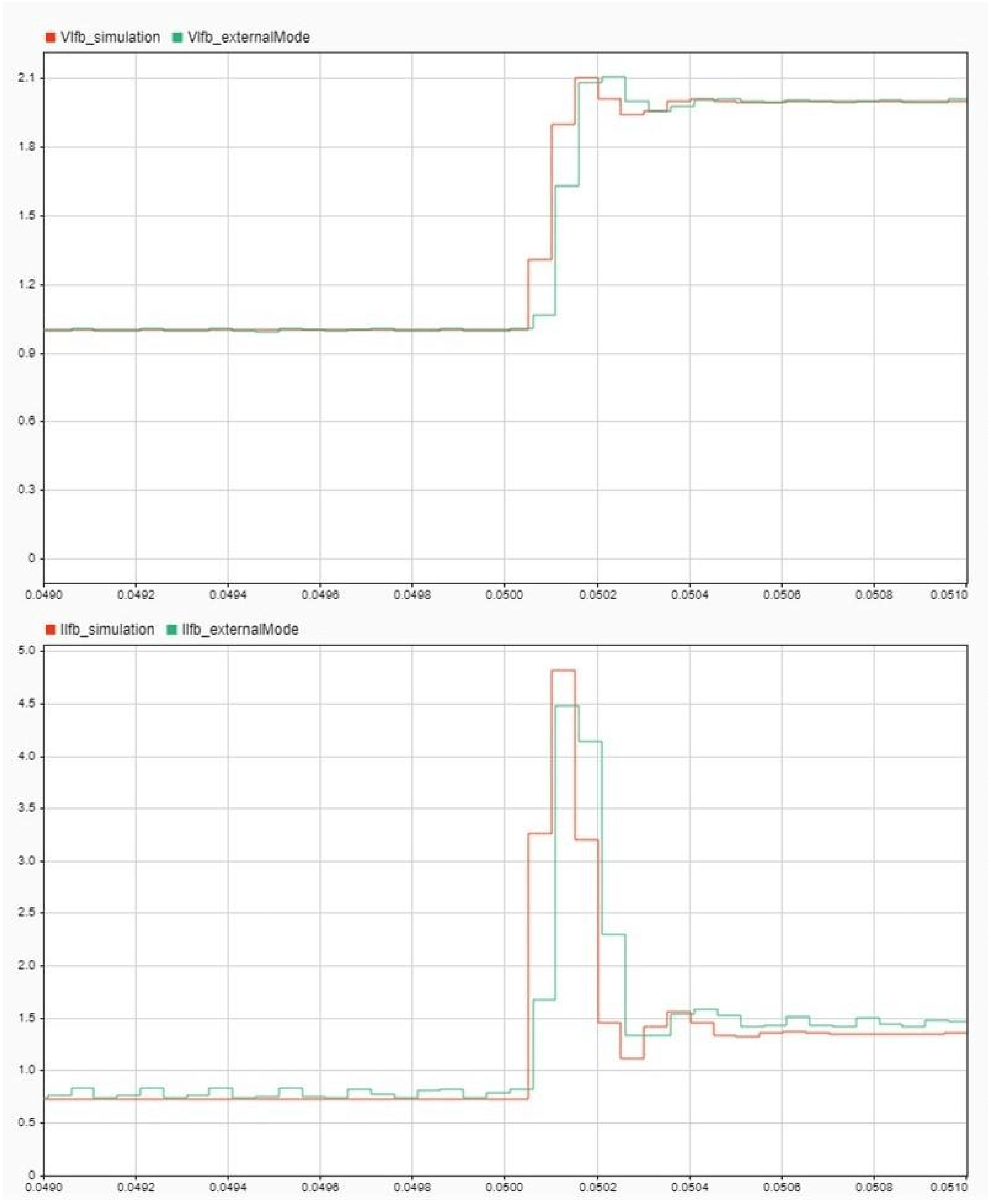
A current protection algorithm runs on the C2000 CPU. This algorithm monitors the current levels and protects the hardware if the current spikes beyond a set current threshold level. The CLA and the C2000 CPU can access the same peripheral modules. This algorithm is an auxillary asynchronous component of the power conversion system, the more computationally intensive VMC algorithm on the CLA.


CPU1 for Current Protection and Data Logging
System Initialize
System Terminate
Active load is always ON to stay in CCM
1
P: 50
Trigger()
2
IPC_msg
msg
Data Transfer
3
I_sense
I_sense
V_ref
1
V_ref
Current Protection
Copyright 2021-2022 The MathWorks, Inc.

The C2000 CPU sends the voltage reference signal to the CLA and receives and logs the values of current and voltage feedback. The Interprocess Data Channel (SoC Blockset) blocks model the data exchange between the CLA and the C2000 CPU. When deployed to hardware, these blocks utilize a section of shared memory for data transfer between the CLA and the C2000 CPU and ensure data integrity.

Results

The following graph shows the step response from 0 to 2 volts. The voltage mode controller correctly tracks the desired the voltage output. The currents are below the threshold and the algorithm continues without interruption.



When the input is changed from 0 to 3 volts, the current exceeds the threshold of 6 amps and triggers the current protection algorithm.

See Also

- “Get Started with Multiprocessor Blocks on MCUs” (SoC Blockset)
- “Partition Motor Control for Multiprocessor MCUs” (SoC Blockset)
- “Data Logging Techniques” (SoC Blockset)
- “Interprocess Data Communication via Dedicated Hardware Peripheral” (SoC Blockset)