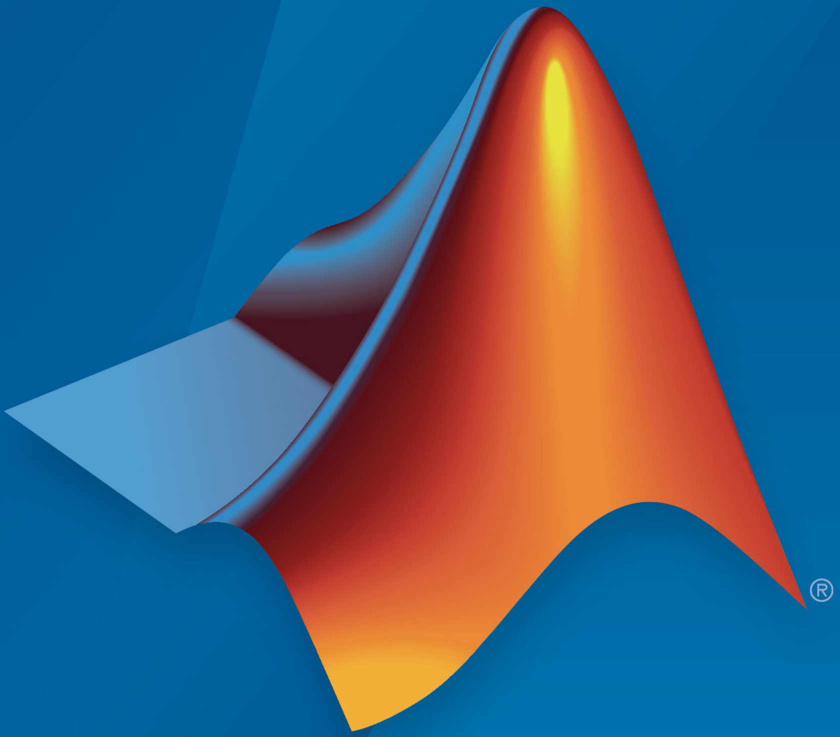
Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors

User's Guide



MATLAB® & SIMULINK®

R2019b

MathWorks®

Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors User's Guide

Revision History

October 2014	Online only	Revised for Version 14.2.0 (R2014b)
March 2015	Online only	Revised for Version 15.1.0 (R2015a)
September 2015	Online only	Revised for Version 15.2.0 (R2015b)
November 2015	Online only	Rereleased for Version 15.2.2 (R2015b)
March 2016	Online only	Revised for Version 16.1.0 (R2016a)
September 2016	Online only	Revised for Version 16.2.0 (R2016b)
March 2017	Online only	Revised for Version 17.1.0 (R2017a)
September 2017	Online only	Revised for Version 17.2.0 (R2017b)
March 2018	Online only	Revised for Version 18.1.0 (R2018a)
September 2018	Online only	Revised for Version 18.2.0 (R2018b)
October 2018	Online only	Revised for Version 18.2.1 (R2018b)
March 2019	Online only	Revised for Version 19.1.0 (R2019a)
September 2019	Online only	Revised for Version 19.2.0 (R2019b)
October 2019	Online only	Revised for Version 19.2.1 (R2019b)
January 2020	Online only	Revised for Version 19.2.2 (R2019b)

Contents

1 Working with Texas Instruments C2000 Processors

Working with Texas Instruments C2000 Processors

Install Support for Texas Instruments C2000 Processors

In this section...

Before you install the software, if you install the Texas Instruments (TI) Code Composer Studio™ on Windows®, then, make sure that:

- You have administrator privileges
- You have set the User Account Control (UAC) settings to the lowest level
- You install in a folder other than Program Files

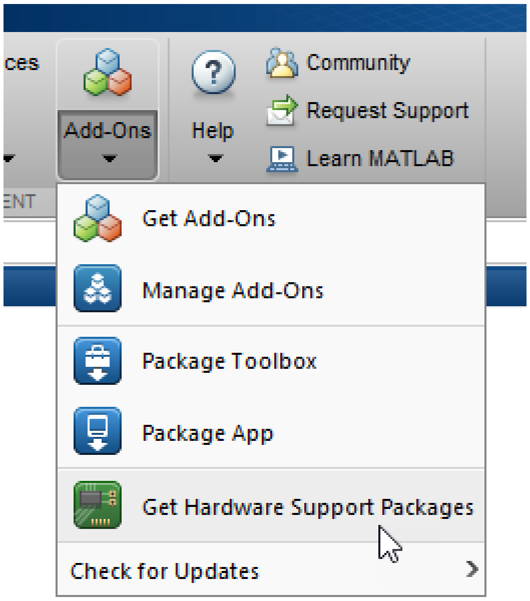
Using this installation process, you can download and install the following items on your host computer:

- Support for Texas Instruments C2000 processors and its features.
- Simulink® block library Embedded Coder Support Package for Texas Instruments C2000 Processors.
- Examples that show you how to use the Texas Instruments C2000 processor.

Install, Update, or Uninstall Support Package

Install Support Package

1. On the MATLAB® **Home** tab, in the **Environment** section, select **Add-Ons > Get Hardware Support Packages**.



2 In the Add-On Explorer window, click the support package and then click **Install**.

Update Support Package

On the MATLAB **Home** tab, in the **Environment** section, select **Help** > **Check for Updates**.

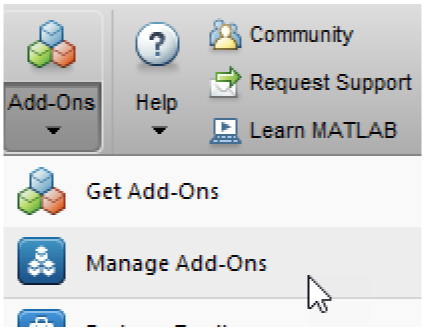
Uninstall Support Package

1 On the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons** > **Manage Add-Ons**.
2 In the **Add-On Manager** window, find and click the support package, and then click **Uninstall**.

Hardware Setup

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

If the support package is already installed, you can start the hardware setup by opening the Add-On Manager.



In the Add-On Manager, start the hardware setup process by clicking the **Setup** button, ⚙.

After starting, the Hardware Setup window provides instructions for configuring the support package 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.

Install Code Composer Studio

The C2000 support package 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.

See Also

"Modeling"

Supported Texas Instruments C2000 Processors

Processor Family	Processors
TI Delfino F28377S LaunchPad	F28377S
TI Delfino F2837xS	F28379S, F28377S, F28376S, F28375S, and F28374S
TI Delfino F28379D LaunchPad	F28379D
TI Delfino F2837xD	F28379D, F28377D, F28376D, F28375D, and F28374D
TI Delfino F2833x	F28335, F28334, and F28332
TI Delfino C2834x	C28346, C28345, C28344, C28343, C28342, and C28341
TI Piccolo F280049C LaunchPad	F280049C
TI Piccolo F28004x	F280049M, F280049C, F280049, F280048C, F280048, F280045, F280041C, F280041, F280040C, and F280040
TI Piccolo F2807x	F28075 and F28074
TI Piccolo F2806x	F28069M, F28069, F28068, F28067, F28066, F28065, F28064, F28063, and F28062
TI Piccolo F28069M LaunchPad	F28069M
TI Piccolo F2805x	F28055, F28054, F28053, F28052, F28051, and F28050
TI Piccolo F2803x	F28035, F28034, F28033, F28032, F28031, and F28030
TI Piccolo F2802x	F28027, F28026, F28023, F28022, F28021, F28020, and F280200
TI Piccolo F28027/F28027F LaunchPad	F28027
TI F280x	F2809, F2808, F2806, F2802, F2801, F28016, and F28015
TI F28044	F28044
TI F281x	F2812, F2811, and F2810

Processor Family	Processors
TI F2838x (C28x)	F28388D, F28388S, F28386D, F28386S, F28384D, and F28384S

See Also

"Install Support for Texas Instruments C2000 Processors" on page 1-2

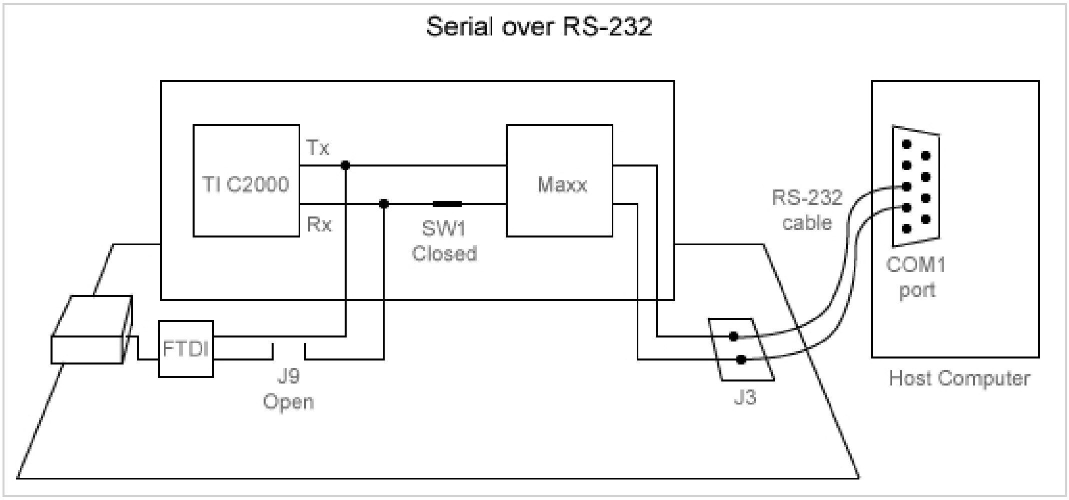
Set Up Serial Communication with Target Hardware

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

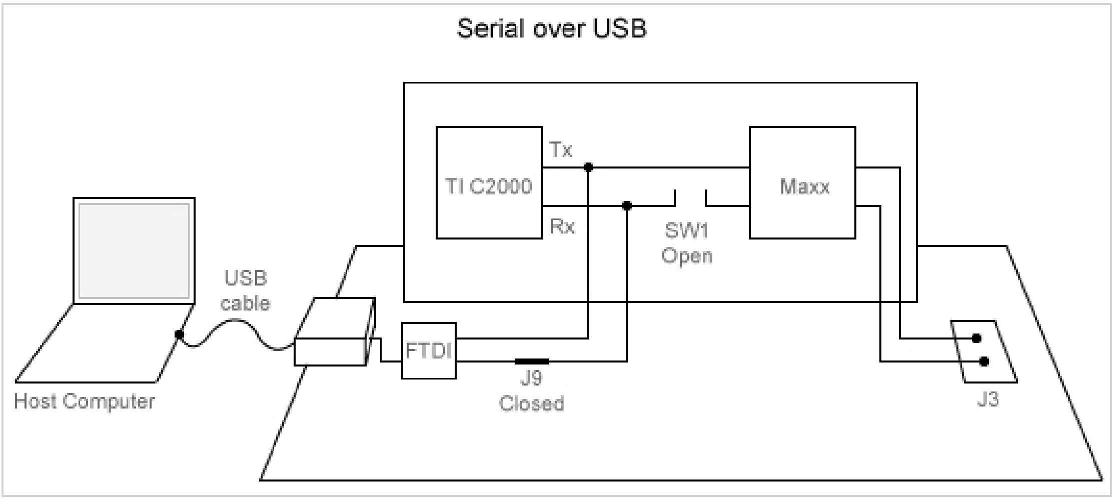
There are different control card versions available for C2000 processors. In some control cards, the SCI_A module pins are directly connected to the USB docking station. Some other control cards have a MAX32xx chip for RS–232 communication on the control card. These control cards have a ‘switch’ to connect 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 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.

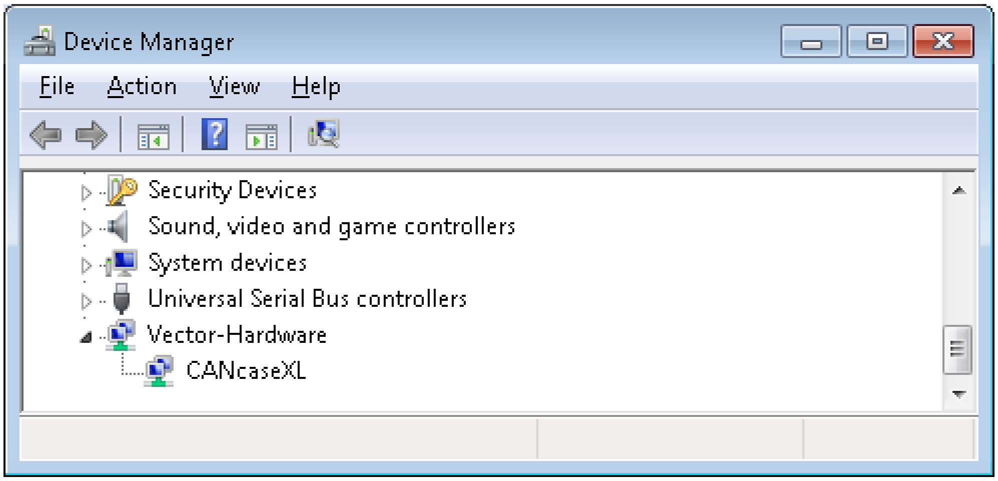
For more information, see http://processors.wiki.ti.com/index.php/
Using_the_serial_adapter_of_XDS100.

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 Open the `c28x_CAN_Tx` model that sends message from the board to the host.

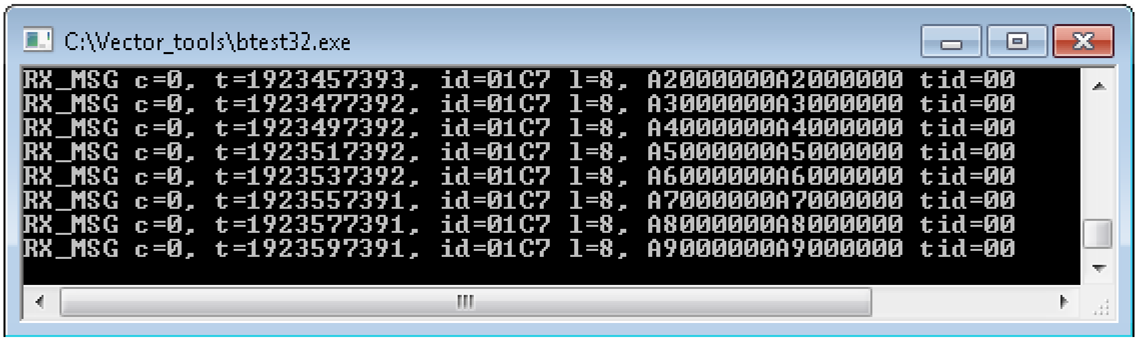
3 Select the target hardware that you are using. The model is set to use a CAN baud of 1Mbit/ sec; you can change the baud settings in the Target Preferences settings.

4 Use btest32.exe (Vector tool) on the host to receive the messages. 'btest32.exe' comes with the Vector drivers that you can find in the driver installation zip file from vector.

Make sure that 'btest32' returns the message values with no errors.

5 Set the baud for btest32 as follows:

For 1 Mbit/s: btest32.exe 1000000



```
RX_MSG c=0, t=1923457393, id=01C7 l=8, A2000000A2000000 tid=00
RX_MSG c=0, t=1923477392, id=01C7 l=8, A3000000A3000000 tid=00
RX_MSG c=0, t=1923497392, id=01C7 l=8, A4000000A4000000 tid=00
RX_MSG c=0, t=1923517392, id=01C7 l=8, A5000000A5000000 tid=00
RX_MSG c=0, t=1923537392, id=01C7 l=8, A6000000A6000000 tid=00
RX_MSG c=0, t=1923557391, id=01C7 l=8, A7000000A7000000 tid=00
RX_MSG c=0, t=1923577391, id=01C7 l=8, A8000000A8000000 tid=00
RX_MSG c=0, t=1923597391, id=01C7 l=8, A9000000A9000000 tid=00
```

This confirms the following:

The Vector CAN card drivers are working on the host.

The CAN settings, connection, and the cable on the target are working properly.

6. 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.

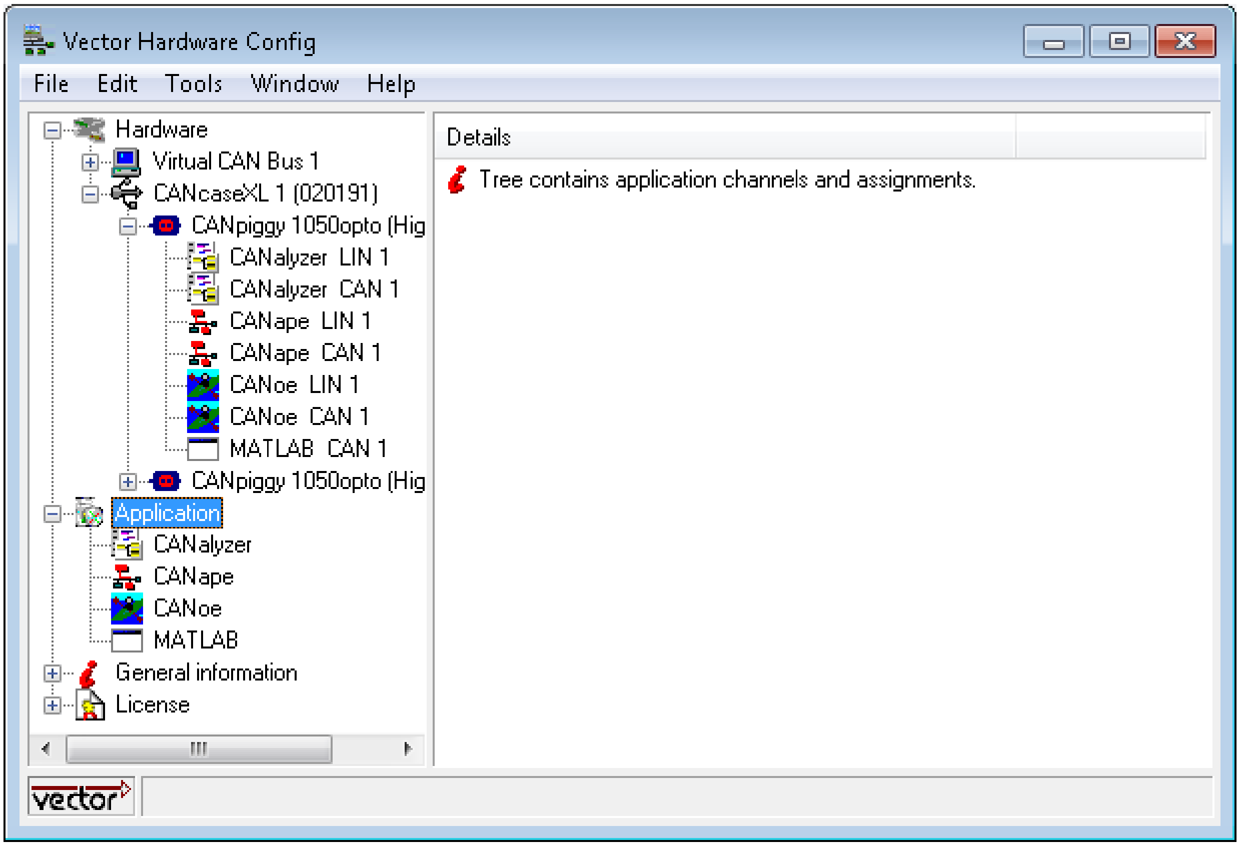
   This folder name may vary depending on the Windows OS (for example C:\Windows \Sys32).

7. Run the vcanconf.exe in the vector driver installation.
8. Right-click 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.

9 Open the CCP example model `c28x_ccp_ert`. Open the Configuration Parameters dialog box to select the desired processor and verify the CAN baud. The default baud is 1 Mbits/sec.

10 To run the model in External mode, click **Run**.

CAN communication is now set up between your host computer and the C2000 target hardware. Click the **Info** button on the model to get help on this example.

Data Type Support

Texas Instruments C2000 MCUs support 16-bit and 32-bit data types, but do not support native 8-bit data types. Simulink models and Embedded Coder software support many data types, including 8-bit data types.

If you select `int8` or `uint8` in your model, your simulation runs with 8-bit data, but in the generated code this data is represented as 16-bit. This may cause instances where data overflow and wraparound occurs in the simulation, but not in the generated code.

For example, to make the overflow behavior of the simulation and generated code match for a Simulink Add block in your model, select **Saturate on integer overflow** in that block.

In C2000 devices, in the generated code, the `double` data type is represented as single precision floating point values (32-bit). This representation results in a difference in data values in the simulation and the generated code.

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:

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 “Hardware Implementation Pane: Texas Instruments C2000 Processors”.

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.

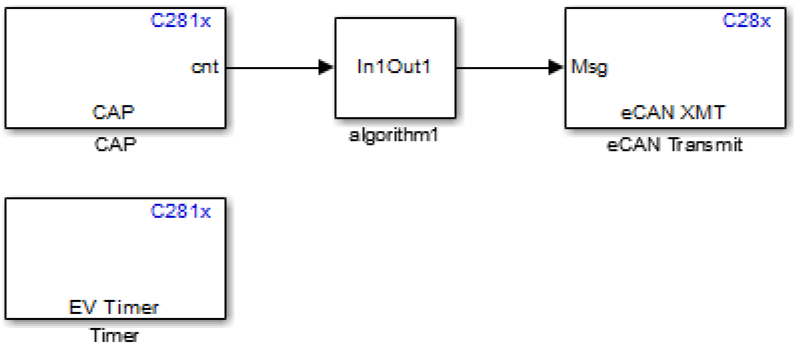
Each GP timer is available to multiple peripherals. For example:

- PWM1-PWM6 and CAP1-CAP3 share GP Timer 1

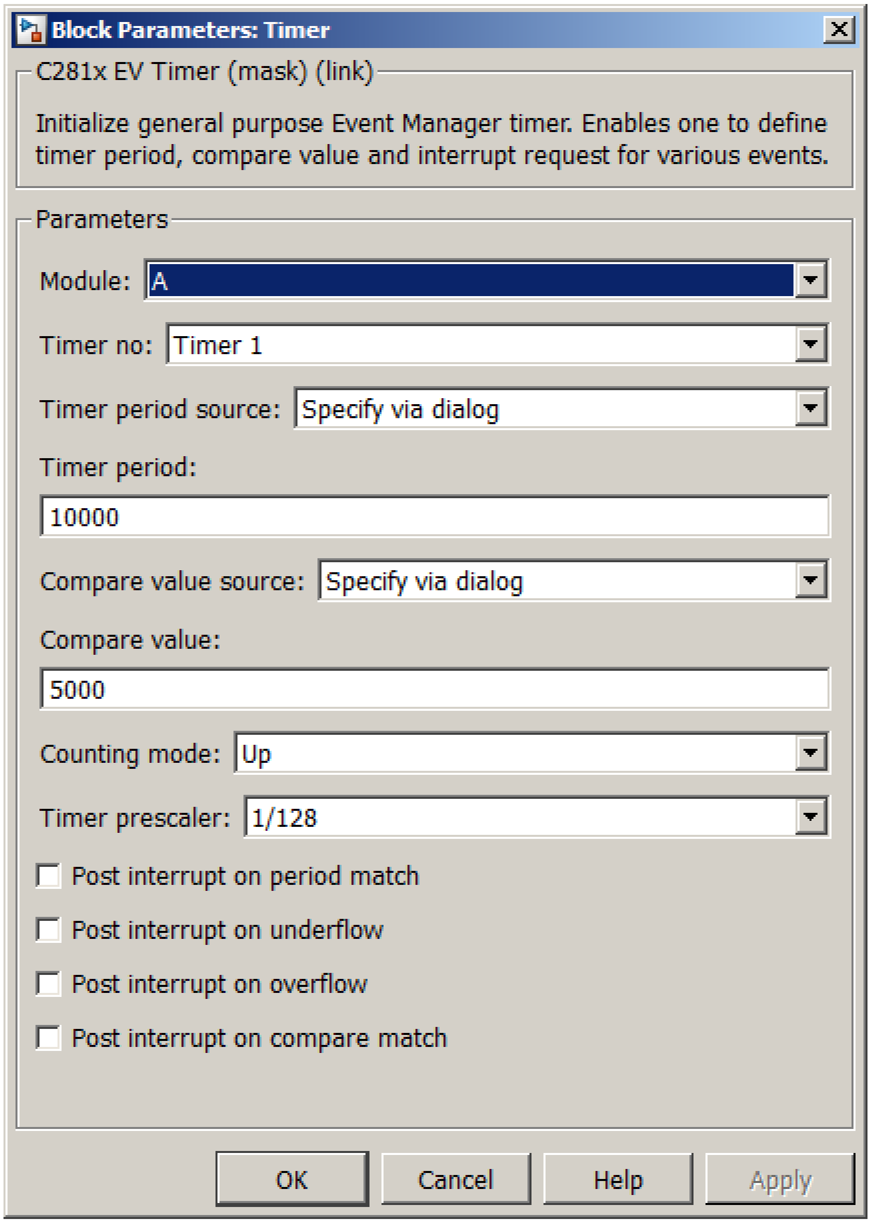
- PWM7-PWM12 and CAP4-CAP6 share GP Timer 3
- QEP1-QEP2 and CAP1-CAP3 share GP Timer 2
- QEP3-QEP4 and CAP4-CAP6 share GP Timer 4

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).



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

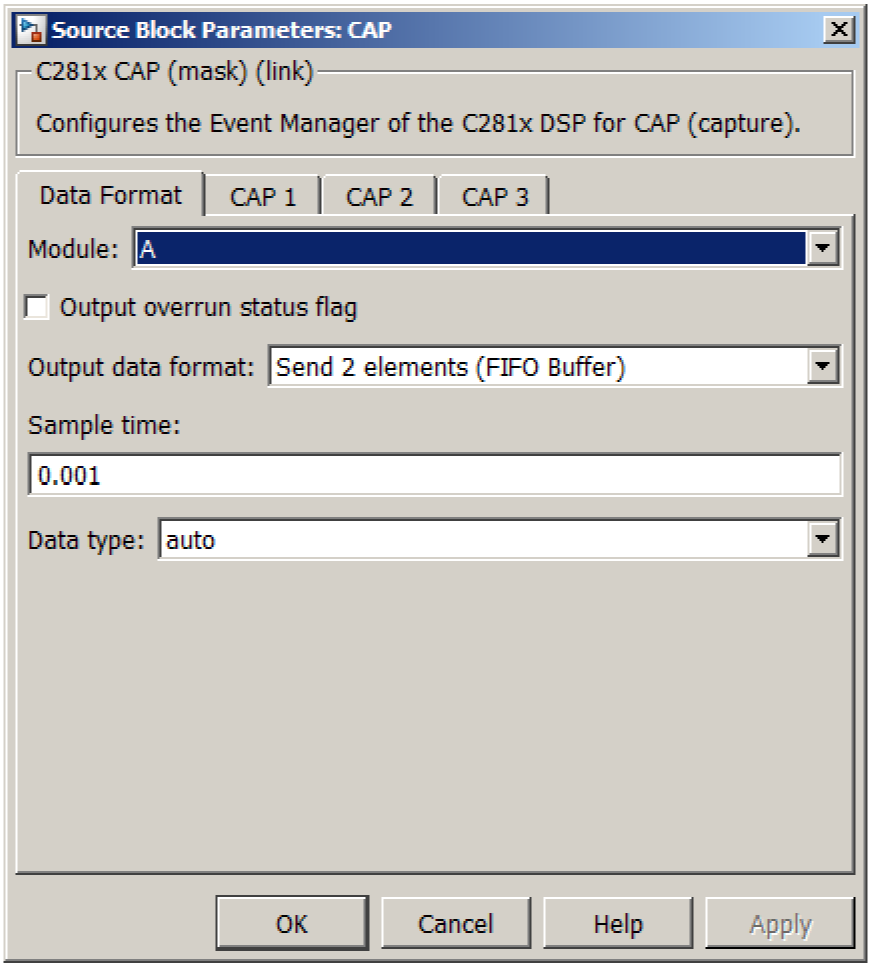
Timer prescaler: 1/128

☐ Post interrupt on period match

☐ Post interrupt on underflow

☐ Post interrupt on overflow

☐ Post interrupt on compare match


Source Block Parameters: CAP

C281x CAP (mask) (link)

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

Module: A

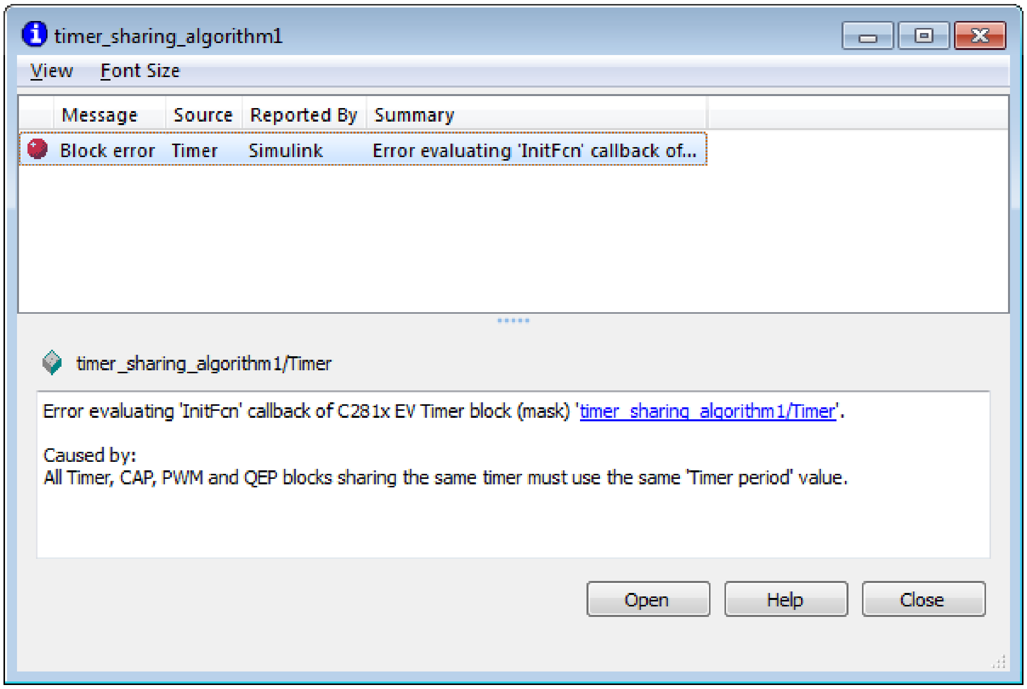
☐ Output overrun status flag

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...

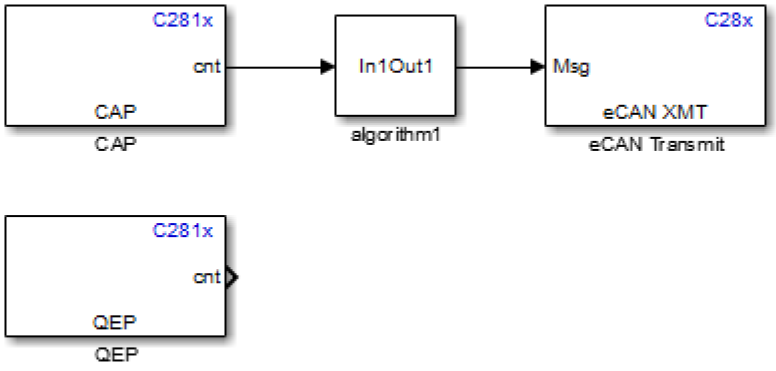
timer_sharing_algorithm1/Timer

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

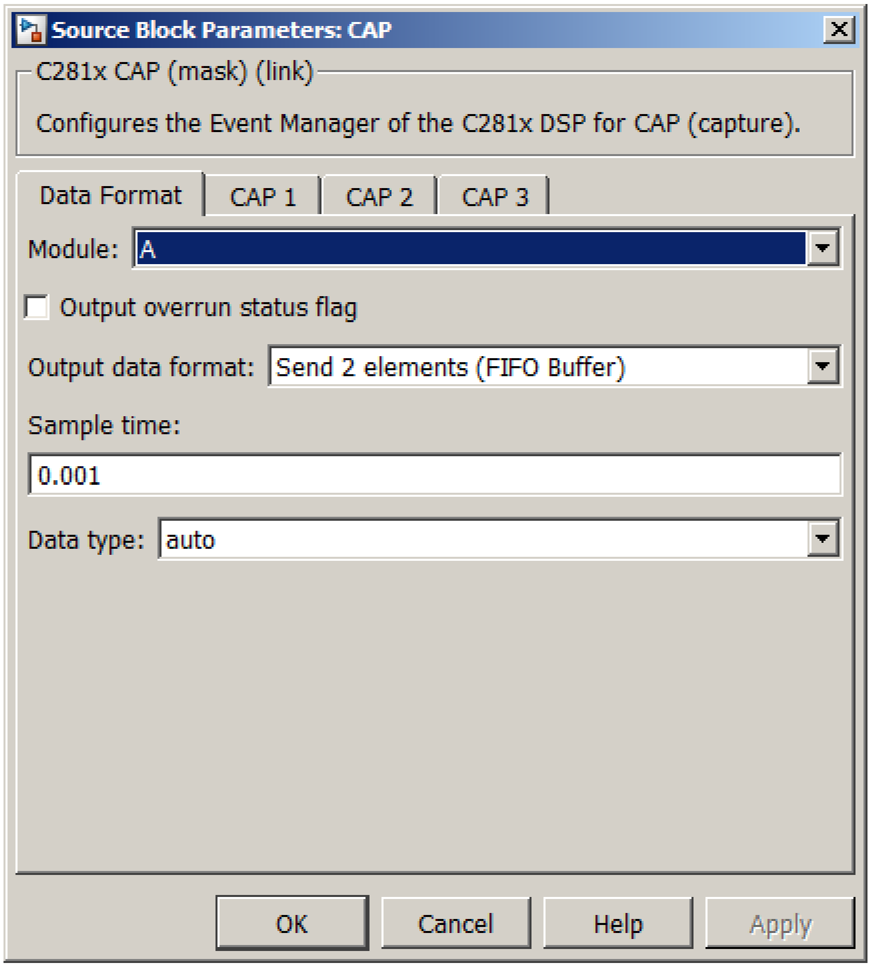
Caused by:
All Timer, CAP, PWM and QEP blocks sharing the same timer must use the same 'Timer period' value.

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.



C281x CAP (mask) (link)

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

Module: A

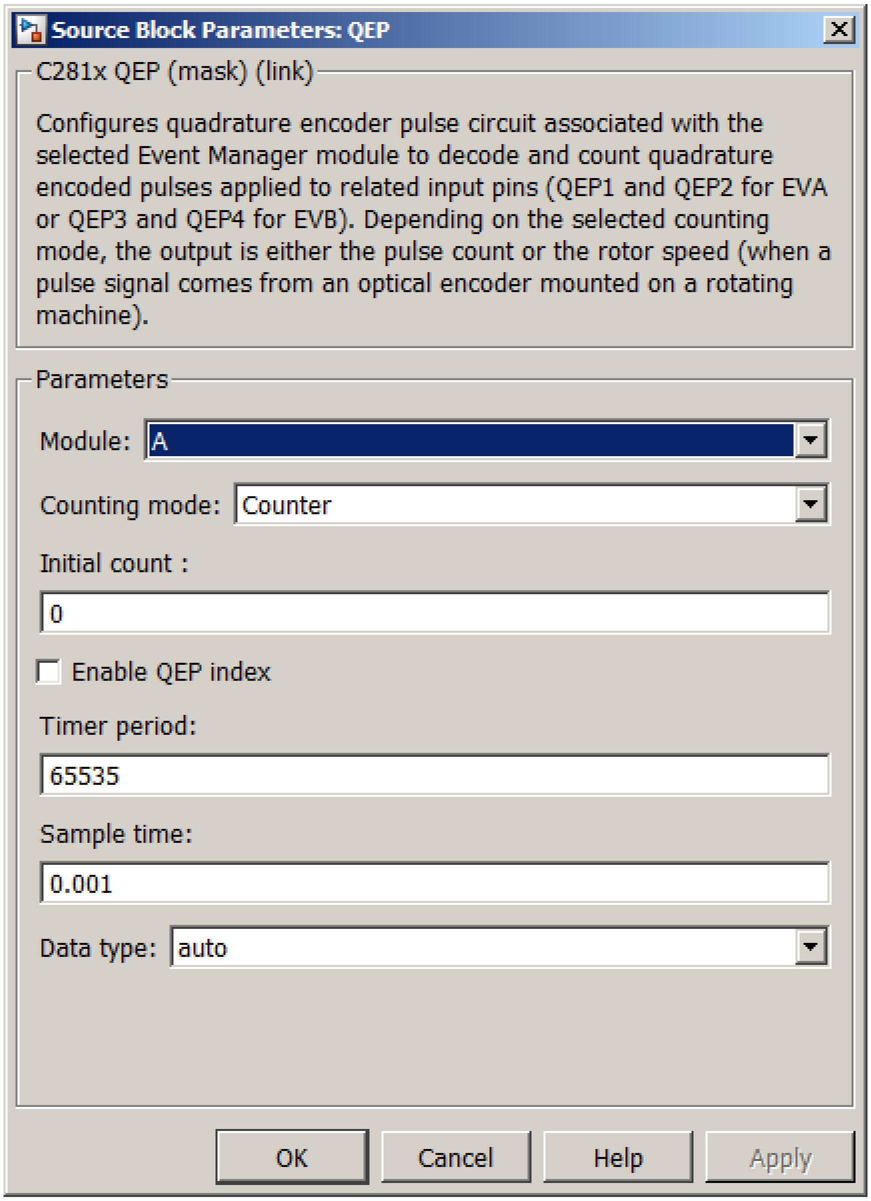
☐ Output overrun status flag

Output data format: Send 2 elements (FIFO Buffer)

Sample time:

0.001

Data type: auto



Source 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

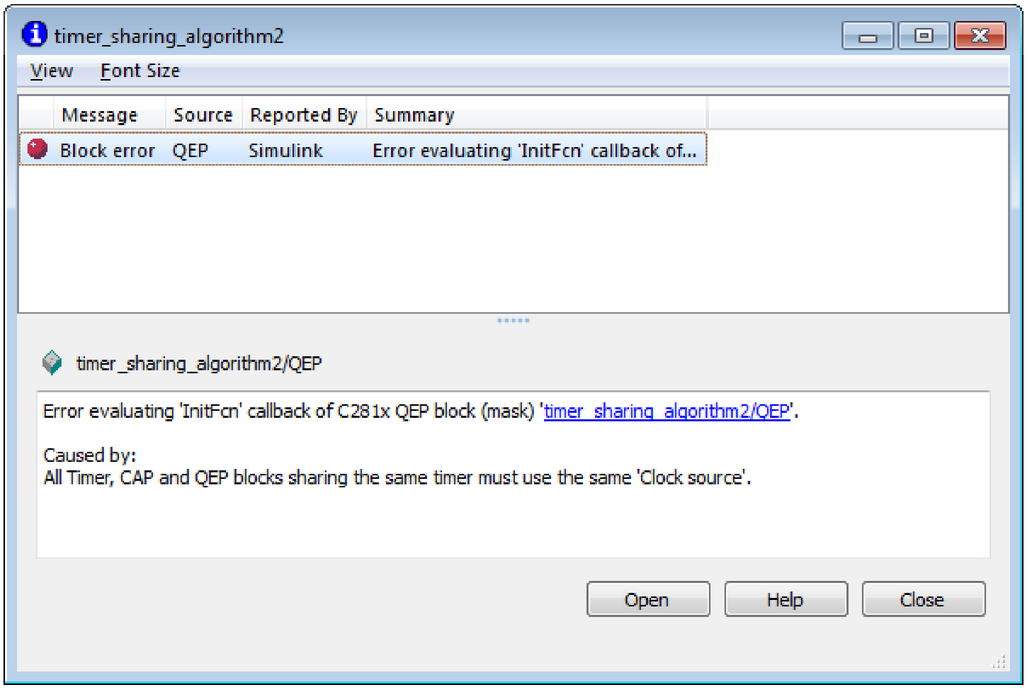
☐ Enable QEP index

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`.

Overview of Creating Models for C2000 Processors

In this section...
"Accessing the Embedded Coder Block Library" on page 1-24
"Building Your Model" on page 1-24

Accessing the Embedded Coder Block Library

1. After you have installed the supported development board, start MATLAB.
2. Open the `c2000lib` blockset in the Simulink library browser, or type in the following command at the MATLAB command prompt:

   ```
   c2000lib
   ```
3. Create your real-time model for your application the same way you create other Simulink models. Select blocks to build your model from the following sources or products:

   - The libraries in the `c2000lib` block library (for handling input and output functions for on your target hardware)
   - Simulink Coder™ software
   - Discrete time blocks from Simulink
   - Another blockset that meets your needs and operates in the discrete time domain

Note Rapid Accelerator simulation is not supported by Embedded Coder Support Package for Texas Instruments C2000 Processors.

Building Your Model

With this configuration, you can generate a real-time executable and download it to your Texas Instruments development board. Simulink Coder software automatically generates C code and inserts the I/O device drivers as specified by the hardware blocks in your block diagram. These device drivers are inserted in the generated C code.

When you are creating a custom device driver block using S-function, use the MATLAB_MEX_FILE macro to differentiate between simulation and code generation behaviors. For example, when you include the Texas Instruments header file in the generated code for creating the MEX file, use the `#else` section to avoid compilation errors, as shown:

```
#ifdef MATLAB_MEX_FILE /*
/* Simulation behavior */
#else
/* Code generation behavior*/
#endif
```

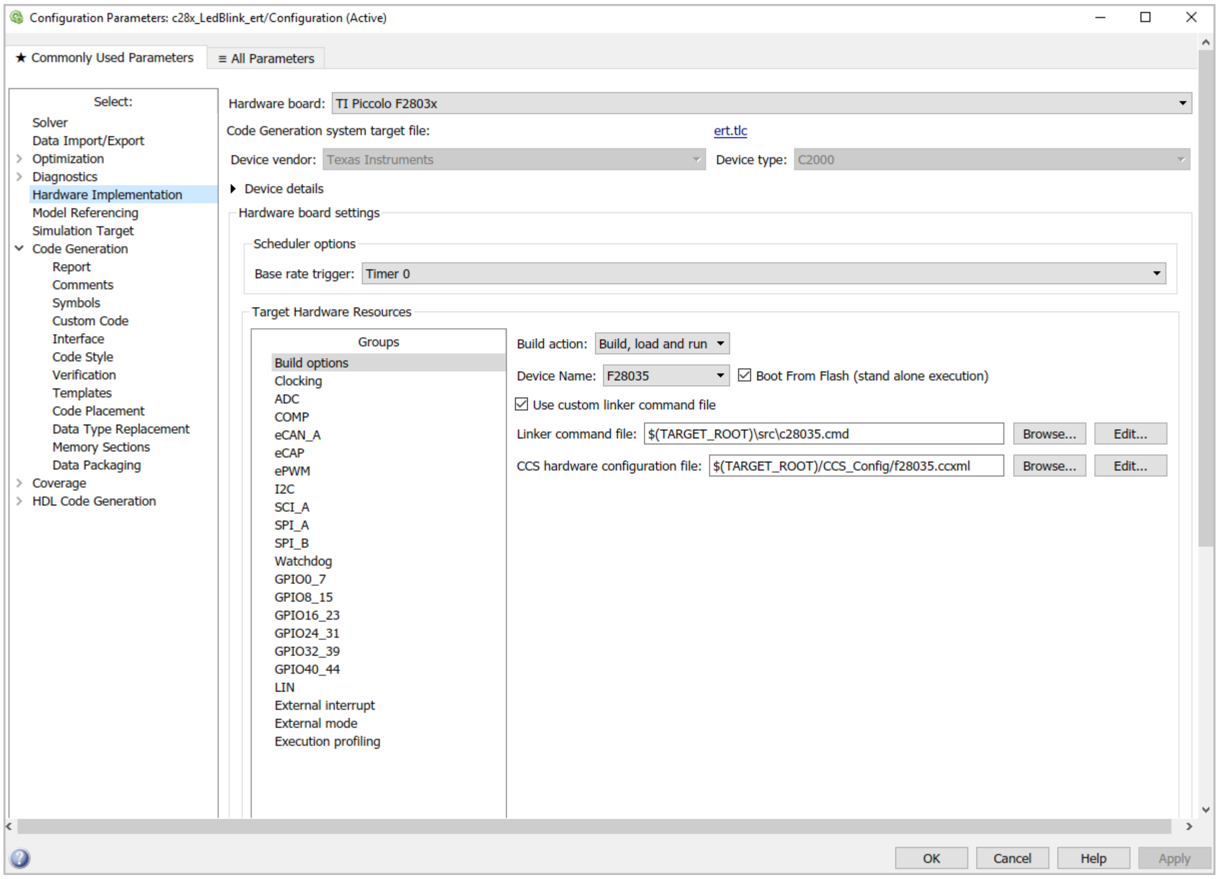
During the build operation, the Texas Instruments cross-compiler builds an executable file from the generated code. If you select the `Build, load and run option in Hardware Implementation > Target Hardware Resources > Build options > Build action parameter` then the generated executable is automatically downloaded to the target. For CCS v5 and the later versions, a CCS project file is also generated during the build process. You can use this project file for debugging in the CCS IDE.

Creating CCS Project from a Model

You can create a Code Composer Studio project from a model. To create CCS project and open the project, follow the steps given.

1. Install the TI C2000 support package and complete the additional setup tasks mentioned in “Install Support for Texas Instruments C2000 Processors” on page 1-2 .
2. Open the model **c28x_LedBlink_ert.slx**.

   This model is configured for a default target hardware (TI Piccolo F28035). To select a different target hardware, go to Configuration Parameters > Hardware Implementation > Hardware board.
3. If you select a different processor, make sure to replace the GPIO blocks and the GPIO pins connected to the LED with the GPIO blocks of the selected processor.


Configuration Parameters: c28x_LedBlink_ert/Configuration (Active)

Hardware board: TI Piccolo F2803x

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

Build action: Build, load and run

Device Name: F28035

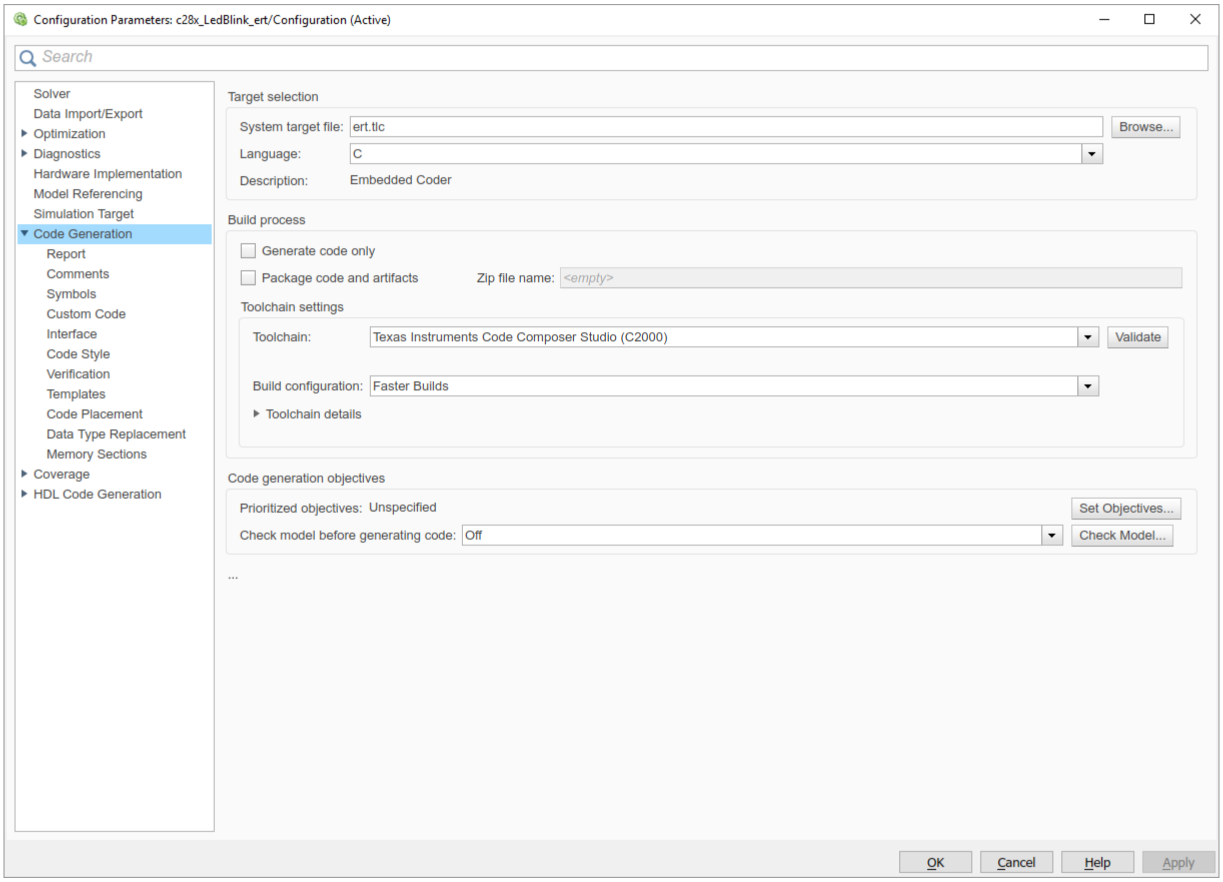
☑ Boot From Flash (stand alone execution)

☑ Use custom linker command file

Linker command file: $(TARGET_ROOT)\src\c28035.cmd

CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28035.ccxml

4 By default, the model is configured with the `Texas Instruments Code Composer Studio (C2000)` toolchain to build, load, and run.


Configuration Parameters: c28x_LedBlink_ert/Configuration (Active)

Target selection

System target file: ert.tlc

Language: C

Description: Embedded Coder

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

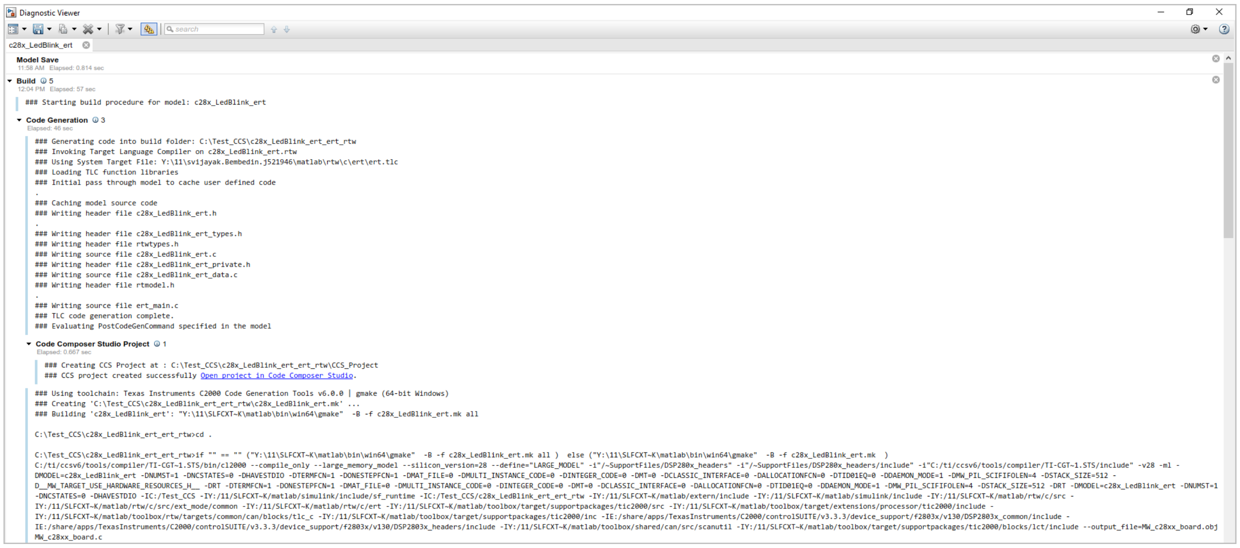
Prioritized objectives: Unspecified

Check model before generating code: Off

5 Click **Build Model** to build, load, and run the program and to create the CCS project.

6 Click **‘View Diagnostics’** to open the **Diagnostic Viewer**.

7 Under the Code Composer Studio Project section in the Diagnostic Viewer, click the link **'Open Project in Code Composer Studio'**.

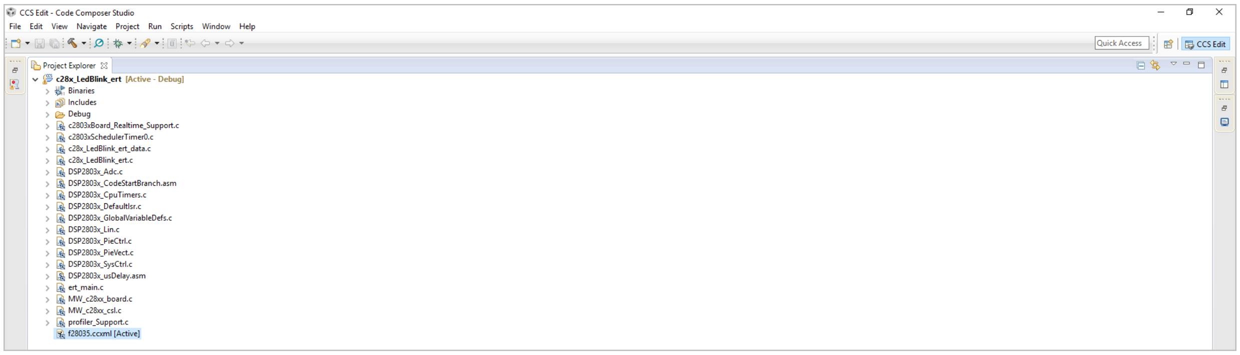


```
### Starting build procedure for model: c28x_LedBlink_ert

### Generating code into build folder: C:\Test_CCS\c28x_LedBlink_ert_ert_rtw
### Invoking Target Language Compiler on c28x_LedBlink_ert.rtw
### Using System Target File: Y:\11\svijayak.Bembedin.j521946\matlab\rtw\c\ert\ert.tlc
### Loading TLC function libraries
### Initial pass through model to cache user defined code
.
### Caching model source code
### Writing header file c28x_LedBlink_ert.h
.
### Writing header file c28x_LedBlink_ert_types.h
### Writing header file rtwtypes.h
### Writing source file c28x_LedBlink_ert.c
### Writing header file c28x_LedBlink_ert_private.h
### Writing source file c28x_LedBlink_ert_data.c
### Writing header file rtmodel.h
.
### Writing source file ert_main.c
### TLC code generation complete.
### Evaluating PostCodeGenCommand specified in the model

### Creating CCS Project at : C:\Test_CCS\c28x_LedBlink_ert_ert_rtw\CCS_Project
### CCS project created successfully Open project in Code Composer Studio.

### Using toolchain: Texas Instruments C2000 Code Generation Tools v6.0.0 | gmake (64-bit Windows)
### Creating 'C:\Test_CCS\c28x_LedBlink_ert_ert_rtw\c28x_LedBlink_ert.mk' ...
### Building 'c28x_LedBlink_ert': "Y:\11\SLFCXT~K\matlab\bin\win64\gmake"  -B -f c28x_LedBlink_ert.mk all

C:\Test_CCS\c28x_LedBlink_ert_ert_rtw>cd .

C:\Test_CCS\c28x_LedBlink_ert_ert_rtw>if "" == "" ("Y:\11\SLFCXT~K\matlab\bin\win64\gmake"  -B -f c28x_LedBlink_ert.mk all )  else ("Y:\11\SLFCXT~K\matlab\bin\win64\gmake"  -B -f c28x_LedBlink_ert.mk  )
C:/ti/ccsv6/tools/compiler/TI-CGT~1.STS/bin/cl2000 --compile_only --large_memory_model --silicon_version=28 --define="LARGE_MODEL" -i"/~SupportFiles/DSP280x_headers" -i"/~SupportFiles/DSP280x_headers/include" -i"C:/ti/ccsv6/tools/compiler/TI-CGT~1.STS/include" -v28 -ml -
DMODEL=c28x_LedBlink_ert -DNUMST=1 -DNCSTATES=0 -DHAVESTDIO -DTERMFCN=1 -DONESTEPFCN=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTEGER_CODE=0 -DMT=0 -DCLASSIC_INTERFACE=0 -DALLOCATIONFCN=0 -DTID01EQ=0 -DDAEMON_MODE=1 -DMW_PIL_SCIFIFOLEN=4 -DSTACK_SIZE=512 -
D__MW_TARGET_USE_HARDWARE_RESOURCES_H__ -DRT -DTERMFCN=1 -DONESTEPFCN=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTEGER_CODE=0 -DMT=0 -DCLASSIC_INTERFACE=0 -DALLOCATIONFCN=0 -DTID01EQ=0 -DDAEMON_MODE=1 -DMW_PIL_SCIFIFOLEN=4 -DSTACK_SIZE=512 -DRT -DMODEL=c28x_LedBlink_ert -DNUMST=1
-DNCSTATES=0 -DHAVESTDIO -IC:/Test_CCS -IY:/11/SLFCXT~K/matlab/simulink/include/sf_runtime -IC:/Test_CCS/c28x_LedBlink_ert_ert_rtw -IY:/11/SLFCXT~K/matlab/extern/include -IY:/11/SLFCXT~K/matlab/simulink/include -IY:/11/SLFCXT~K/matlab/rtw/c/src -
IY:/11/SLFCXT~K/matlab/rtw/c/src/ext_mode/common -IY:/11/SLFCXT~K/matlab/rtw/c/ert -IY:/11/SLFCXT~K/matlab/toolbox/target/supportpackages/tic2000/src -IY:/11/SLFCXT~K/matlab/toolbox/target/extensions/processor/tic2000/include -
IY:/11/SLFCXT~K/matlab/toolbox/rtw/targets/common/can/blocks/tlc_c -IY:/11/SLFCXT~K/matlab/toolbox/target/supportpackages/tic2000/inc -IE:/share/apps/TexasInstruments/C2000/controlSUITE/v3.3.3/device_support/f2803x/v130/DSP2803x_common/include -
IE:/share/apps/TexasInstruments/C2000/controlSUITE/v3.3.3/device_support/f2803x/v130/DSP2803x_headers/include -IY:/11/SLFCXT~K/matlab/toolbox/shared/can/src/scanutil -IY:/11/SLFCXT~K/matlab/toolbox/target/supportpackages/tic2000/blocks/lct/include --output_file=MW_c28xx_board.obj
MW_c28xx_board.c
```

The Code Composer Studio IDE launches with the generated project.

8 Open the **'Project Explorer'** from **View** tab in CCS.

9 Right-click the Target Configuration File (.ccxml) and click **'Set as Active Target Configuration'**.



```
c28x_LedBlink_ert [Active - Debug]
  Binaries
  Includes
  Debug
  c2803xBoard_Realtime_Support.c
  c2803xSchedulerTimer0.c
  c28x_LedBlink_ert_data.c
  c28x_LedBlink_ert.c
  DSP2803x_Adc.c
  DSP2803x_CodeStartBranch.asm
  DSP2803x_CpuTimers.c
  DSP2803x_DefaultIsr.c
  DSP2803x_GlobalVariableDefs.c
  DSP2803x_Lin.c
  DSP2803x_PieCtrl.c
  DSP2803x_PieVect.c
  DSP2803x_SysCtrl.c
  DSP2803x_usDelay.asm
  ert_main.c
  MW_c28xx_board.c
  MW_c28xx_csl.c
  profiler_Support.c
  f28035.ccxml [Active]
```

10 On Project Explorer pane, right-click the project in CCS and click **'Build Project'** to start the build process.

Make sure that the target hardware is connected to the host computer.

11 Click **Run** in CCS and click **Debug** (F11) to start the debug session.

12 Click the **Play** icon in the Debug mode to execute the code on the target hardware.

Note

- CCS project creation feature is **not supported with CCS v3.3 and CCS v4.**
- For CCS v5, the compiler and linker settings are not reflected in the CCS GUI, even though these flags are visible in the 'Summary of flags set' section. However, these flags are considered while building the CCS project.
- Compiler tools that are installed outside the CCS Installation directory are not detected automatically. Add the Code generation tool path manually from the CCS GUI path:

  Window > Preferences > Code Composer Studio> Build > Compilers > Tool Discovery.
- You can open only one instance of CCS at a time. To open a new project in CCS, use the '**Restart CCS'** option in the MATLAB prompt that shows while clicking the diagnostic Viewer link. The current instance is closed and a fresh instance of CCS with the new project is opened.

Apply the c2000lib Blockset

In this section...

Introduction

This section uses an example to show how to create a Simulink model that uses Embedded Coder Support Package for Texas Instruments C2000 Processors 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*, available from the Spectrum Digital website at `https://c2000.spectrumdigital.com/ezf2812/`

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)`

Pane	Field	Setting
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.

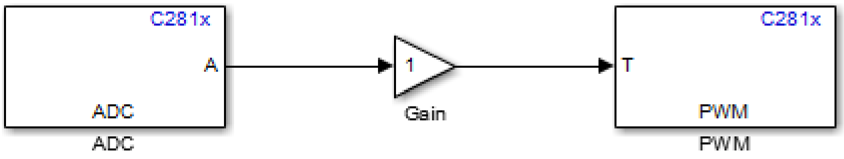
Pane	Field	Parameter
Timer	**Module**	`A`
	Waveform period source	`Specify via dialog`
	Waveform period units	`Clock cycles`
	Waveform period	`64000`
	Waveform type	`Asymmetric`
Outputs	**Enable PWM1/ PWM2**	Selected

Pane	Field	Parameter
	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

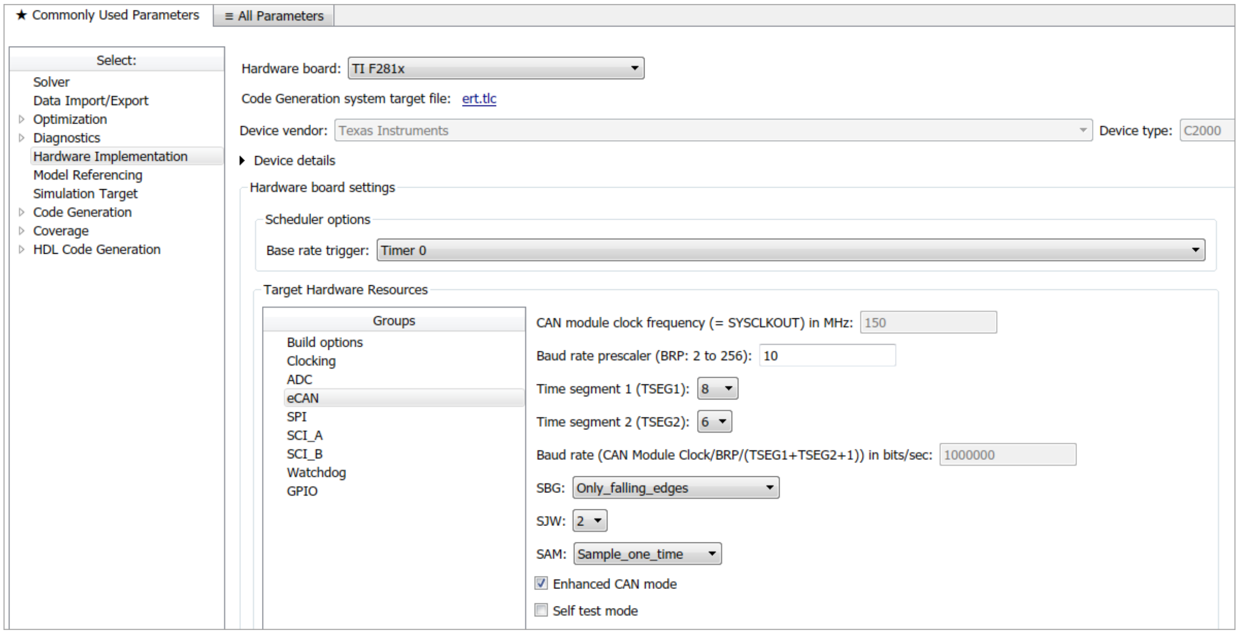
Setting Timing Parameters

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

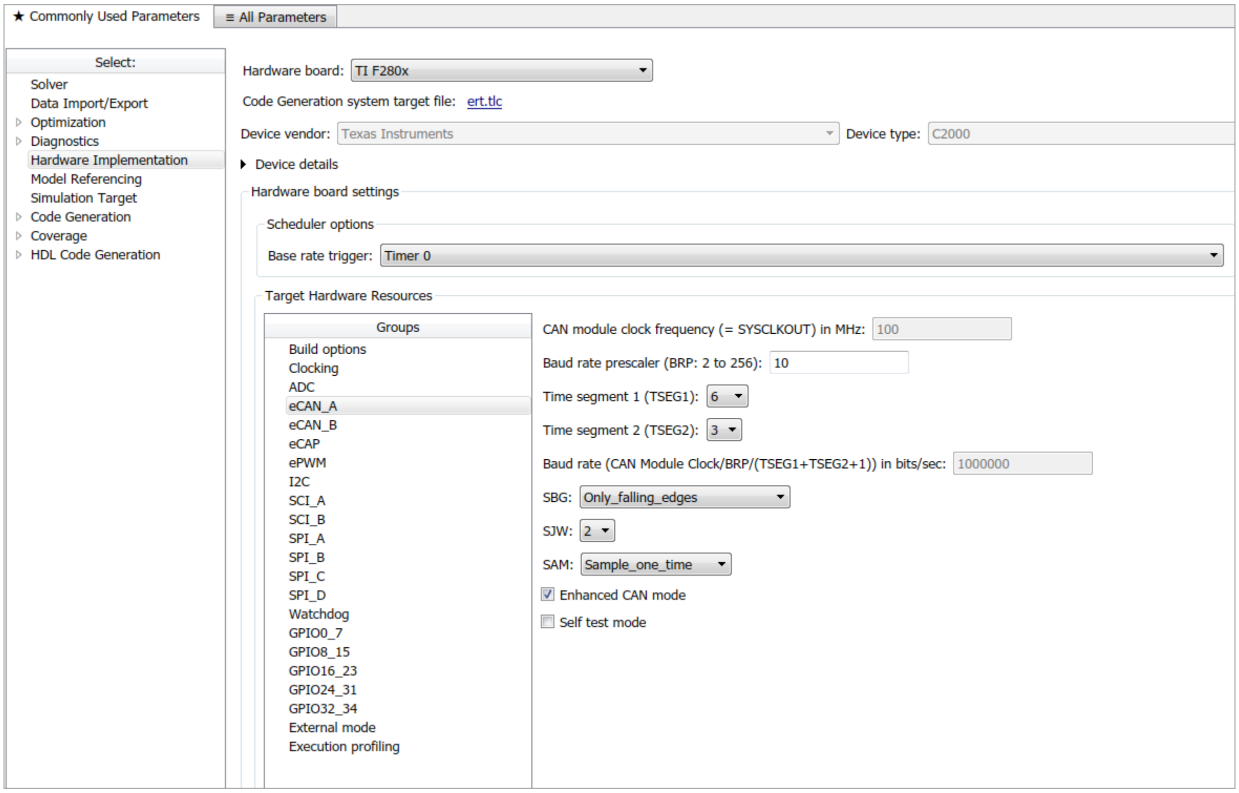
SJW: 2

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

Base rate trigger: Timer 0

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:

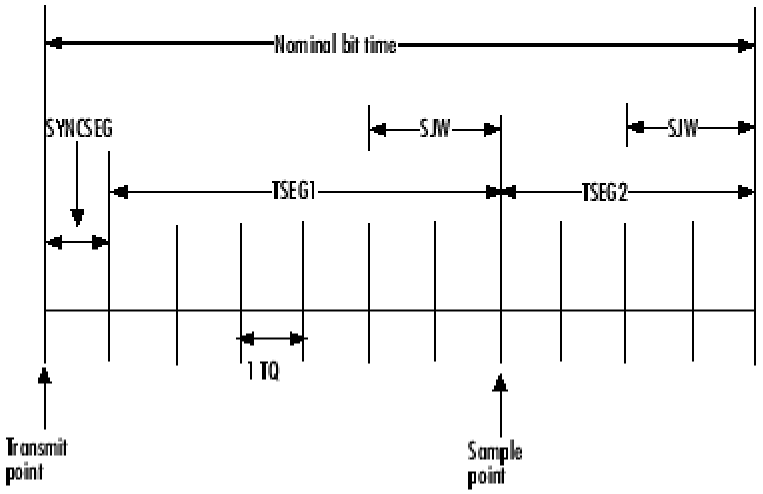
   Bitrate = CAN module clock frequency/(BRP * BitTime)

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.

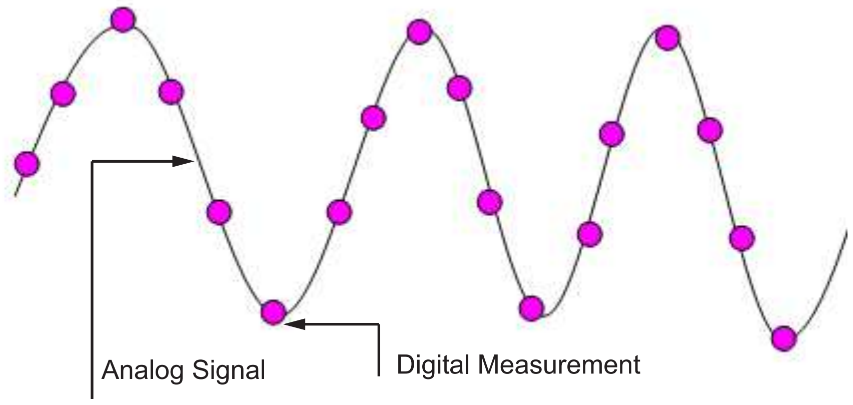
Bitrate	TSEG1	TSEG2	Bit Time	BRP	SJW
0.25 Mbit/s	8	6	10	40	2
0.5 Mbit/s	7	7	10	20	2
1 Mbit/s	8	6	10	10	2

Configuring Acquisition Window Width for ADC Blocks

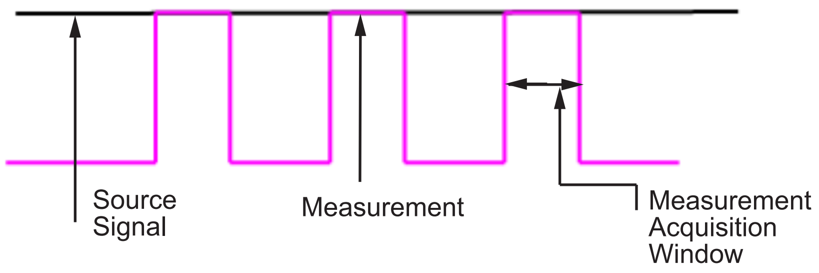
In this section...

What Is an Acquisition Window?

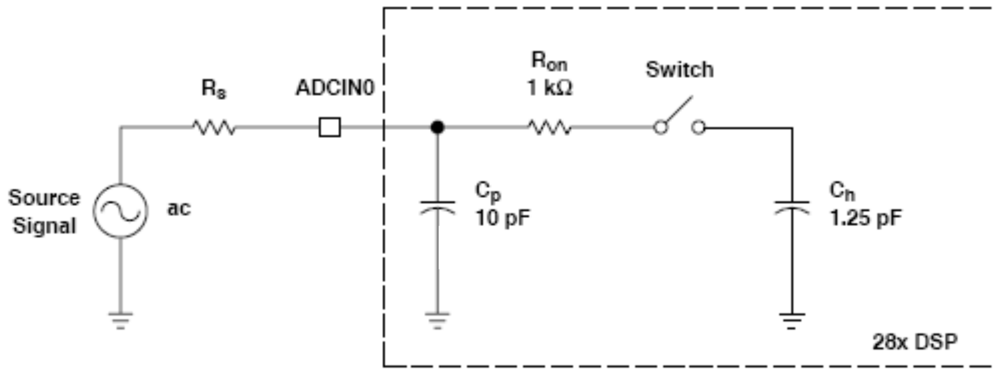
ADC blocks take a signal from an analog source and measure it with a digital device. The digital device does not measure in a continuous process, but in a series of discrete measurements, close enough together to approximate the source signal with the required accuracy.



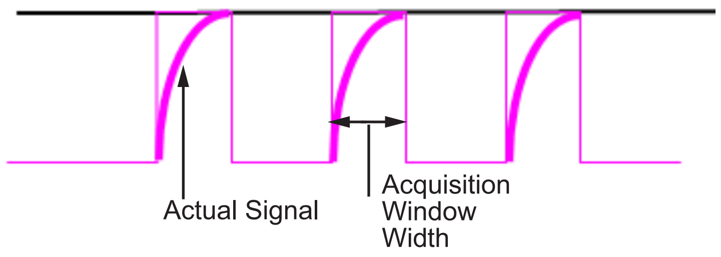
The digital measurement itself is not an instantaneous process, but is a measurement window, where the signal is acquired and measured, as shown in the following figure.



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.

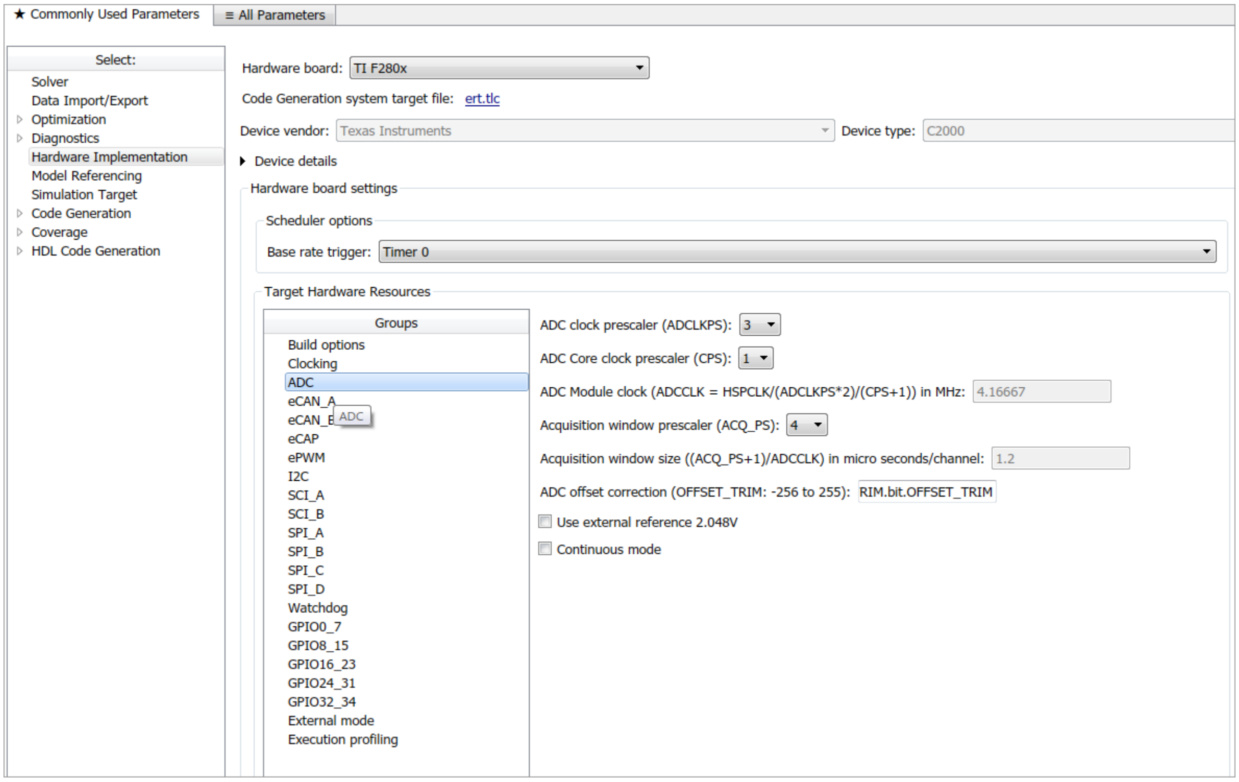
Configuring ADC Parameters for Acquisition Window Width

- “Accessing the ADC Parameters” on page 1-44
- “Configure Acquisition Window Width Using ADC Parameters” on page 1-46

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

Base rate trigger: Timer 0

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

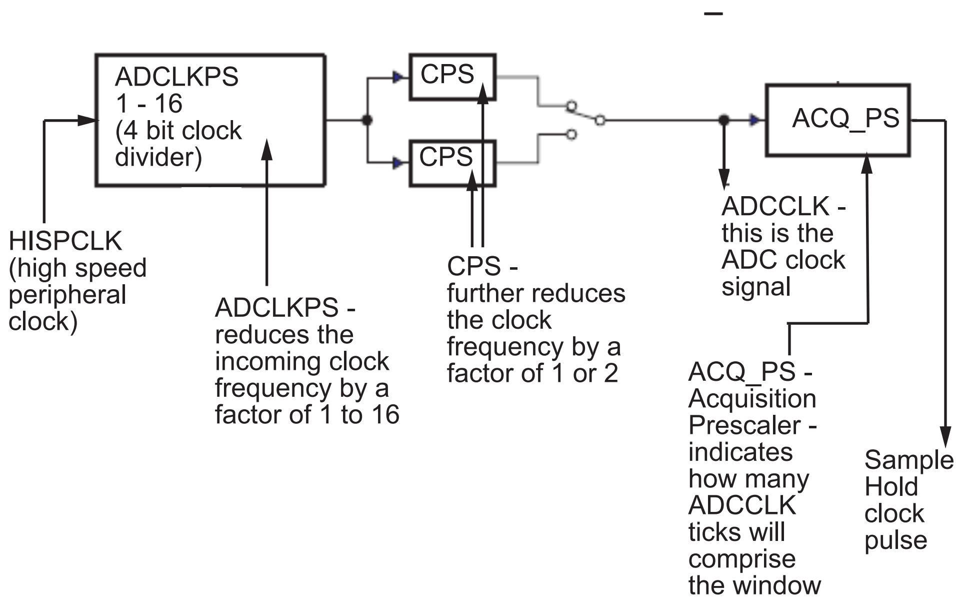
Acquisition window size ((ACQ_PS+1)/ADCCLK) in micro seconds/channel: 1.2

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.



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 1-14

Parameter Tuning and Signal Logging over Serial Communication

In this section...
"Monitor and Tune over Serial Communication" on page 1-48
"Monitor and Tune over CCP" on page 1-51

With **Monitor & Tune** (External Mode), you can log signals and tune parameters while the model is running on the target hardware in real time. When you change parameter values from within Simulink, the modified parameter values are communicated to the target hardware immediately. Also, you can monitor the effects of the parameters tuning activity by viewing the algorithm signals on the scopes.

You can run your model on the external mode via two communication interfaces, serial or CAN.

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 1-7. You can configure the external mode for different baud rates that your communication interface supports.

To run your model in external mode with CAN Calibration Protocol(CCP), use the CAN Calibration Protocol block and a Vector CAN case hardware. External mode uses the eCAN module to run external mode. You can configure the external mode with any available eCAN module. For more information, see "Set Up CAN Communication with Target Hardware" on page 1-10. You can configure the baud for eCAN.

The external mode with 'ert.tlc' downloads the executable, runs the model, and establishes external mode connection in a single step.

Monitor and Tune over Serial Communication

Monitor and Tune (External Mode) over Serial Communication can be done using either of the two available protocols:

- **Classic External Mode**

In this method, you will setup Serial communication between Simulink and Texas Instruments C2000 board to tune the parameters and monitor the signal of an algorithm running on the board.

- **External Mode over Universal Measurement and Calibration (XCP)**

  Simulink provides these additional features for the targets that support XCP:

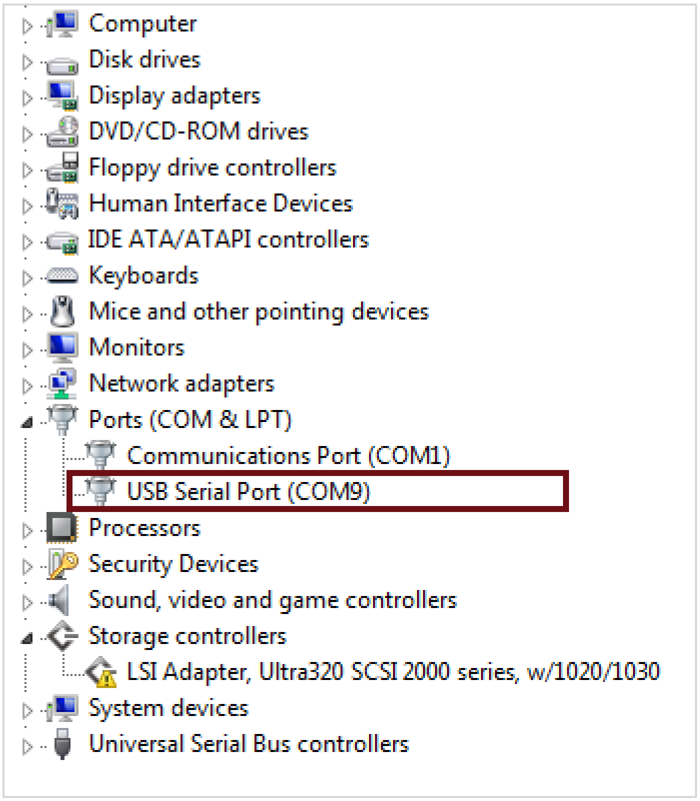
  - Dashboard objects such as Slider and Dashboard Scope. For more information, see Dashboard (Simulink)
  - Simulation Data Inspector (SDI) for visualizing the logged signals. For more information, see Simulation Data Inspector (SDI)
  - Running External mode over XCP has a few limitations. For a detailed list, see External Mode Simulation with XCP Communication(Simulink Coder)

Based on the maximum and the minimum baud that the SCI_A module supports, you can run your model in external mode over different baud rates.

To prepare your model for external mode with serial communication:

1. In the Configuration Parameters dialog box, select **Model Configuration Parameters > Hardware Implementation** .
2. Select a C2000 processor from the Hardware Implementation > **Hardware board** drop-down list.
3. Under **Target Hardware Resources**, select **External mode(Classic)/External Mode over Universal Measurement and Calibration (XCP)**.
4. In **Communication interface** drop-down list, select `serial`.
5. Specify the corresponding COM port that the target hardware uses.

   For the COM port 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

6 Make sure that the **Verbose** check box is selected to view the external mode execution progress and updates in the Diagnostic Viewer or in the Command Window.

7 To execute your model with a specific baud:

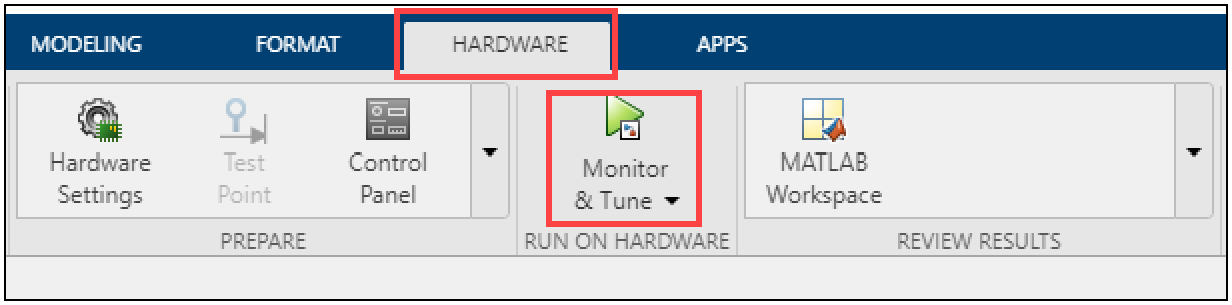
Under **Target Hardware Resources**, select the SCI_A pane and then specify the desired baud in **Desired baud rate in bits/sec** parameter.

The default baud is 115200 bits/sec. For better performance, you may increase the baud to a value that your hardware board permits.

Your model is now ready to perform Monitor and Tune action (External Mode) over serial communication.

To perform the Monitor and Tune action :

1 Open the Model.



2 Go to the**Hardware** tab and click **Monitor & Tune**.

For more information, see “Parameter Tuning and Signal Logging with Serial Communication”

Note

- For targets with small RAM such as F28027, the code has to be booted from flash. To boot the code from flash, select TI Piccolo F2802x (boot from flash) in **Hardware Implementation** > **Hardware board** drop-down list
- Parameter tuning and signal logging for 8-bit data type is not supported in external mode over serial

Monitor and Tune over CCP

You can tune parameters and log signals (External Mode) using Simulink external mode with CCP. For signal logging and parameter tuning with a third-party calibration tool, see “CAN Calibration Protocol with Third Party Tools” on page 1-60.

The external mode is supported using the CAN Calibration Protocol block and ASAP2 interface. The ASAP2 interface is used to get information about where a parameter or signal exists in the target memory. The CAN Calibration Protocol block is used to communicate with the target, download parameter updates, and upload signal information.

To prepare your model for external mode over CCP, perform these tasks:

1. To set up external mode, see “Set Up Your Model for External Mode” on page 1-52.
2. To run your model in external mode with ert.tlc, see “Monitor and Tune Your Model (External Mode)” on page 1-53.
3. To tune your parameters, see “Tuning Parameters” on page 1-55.

For more information, see:

- “Configuring the Host Vector CAN Application Channel” on page 1-53
- “Using Supported Objects and Data Types” on page 1-54
- “Tuning Parameters” on page 1-55
- “Viewing and Storing Signal Data” on page 1-55
- “Limitations” on page 1-59

Set Up Your Model for External Mode

1. Add a CCP driver block to your model from the **Simulink Block Library**.
2. Identify signals that you want to tune. Associate them with `Simulink.Parameter` or `canlib.Parameter` objects with `ExportedGlobal` storage class. Set the data type and value of the object. See “Using Supported Objects and Data Types” on page 1-54.
3. Identify signals you want to log. Associate them with `canlib.Signal` objects. Set the data type of the `canlib.Signal`. See “Using Supported Objects and Data Types” on page 1-54.

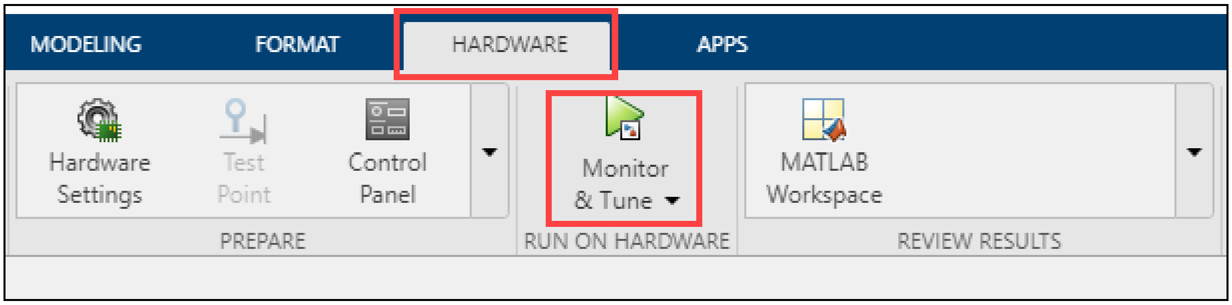
   For information about visualizing logged signal data, see “Viewing and Storing Signal Data” on page 1-55.
4. Load the `Simulink.Parameter` or `canlib.Parameter` and `canlib.Signal` data objects into the base workspace.
5. Select **Simulation** > **Model Configuration Parameters**.
6. In the Configuration Parameters dialog box, select **Optimization** > **Signals and Parameters** pane.
7. Select **Default parameter behavior** > **Inlined** and click **Configure**.
8. In the Model Parameter Configuration dialog box that opens, define the (global) tunable parameters for your models.
9. In the Configuration Parameters dialog box, select **Code Generation** > **Interface** pane.

10 Set the **Interface** parameter to ASAP2.

Monitor and Tune Your Model (External Mode)

To perform Monitor & Tune action :

1 Open the Model.



2 Go to the **Hardware** tab and click **Monitor & Tune**.

Your model now runs in the external mode over CCP.

Configuring the Host Vector CAN Application Channel

For external mode, the host-side CAN connection must use the `'MATLAB 1'` application channel. To configure the application channel that the Vector CAN drivers use, in the Command Window:

```
TargetsComms_VectorApplicationChannel.configureApplicationChannels
```

Use this tool to configure your host-side CAN channel settings.

If you try to connect using an application channel other than `'MATLAB 1'`, you see the following warning:

```
Warning:
It was not possible to connect to the target using CCP.
An error occurred when issuing the CONNECT command.
```

If you have not already installed the Vector CAN drivers, you get the following error message in the command window:

```
??? Error using ==>
TargetsComms_VectorApplicationChannel.TargetsComms_VectorApplicationChannel>
TargetsComms_VectorApplicationChannel.configureApplicationChannels at 40
```

```
Unable to launch the application channel configuration utility.
The "vcanconf" utility was not found on the Windows System Path.
To fix this error, make sure the required CAN drivers are installed on this computer;
refer to the product documentation for details.
```

If you want to use CAN to transmit or receive CAN messages between your host PC and your target, you need VN1600 supported by the Vector CAN Driver Library. Choose the driver libraries to support profiling, downloading, and external mode. Make sure that the library, `vcand32.dll`, is placed in the Windows `system32` folder.

Using Supported Objects and Data Types

Supported objects are:

- `Simulink.Parameter` or `canlib.Parameter` for parameter tuning
- `canlib.Signal` for signal logging

Supported data types are:

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

Define data objects for the signals and parameters of interest for ASAP 2 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` or `canlib.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 canlib.Signal objects.Set the data type of the canlib.Signal. The following code example shows how to declare such a canlib.Signal object for logging.

```
counter = canlib.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. `stepSize` and `counter` are the data objects defined in the code.

Tuning Parameters

1 In the workspace, set *`dataobject`*`.value` while the model is running in external mode. For example, to tune the parameter `stepSize` (that is, to change its value) from 1 to 2, at the command line, enter:

```
stepSize.value = 2
```

You see output similar to the following:

```
stepSize =

  Parameter with properties:

          Value: 2
      CoderInfo: [1×1 Simulink.CoderInfo]
    Description: ''
       DataType: 'uint8'
            Min: []
            Max: []
           Unit: ''
     Complexity: 'real'
     Dimensions: [1 1]
```

2 Update the model (press **Ctrl+D**) to apply the changed parameter.

Viewing and Storing Signal Data

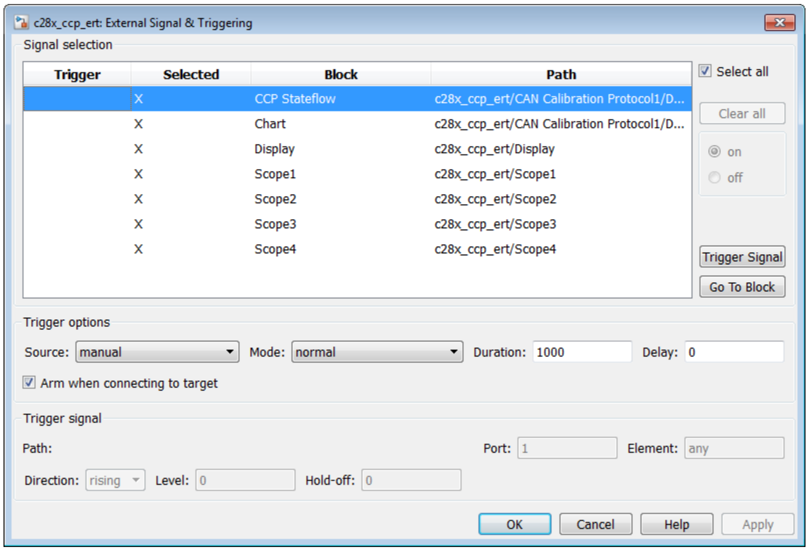
To view the logged signals, attach a supported scope type to the signal.

To customize the signals that are logged:

1 In the Simulink Editor, select **Code > External Mode Control Panel**.

2 In External Mode Control Panel, click **Signal & Triggering**.

3 In External Signal & Triggering box, clear the check boxes for scope data that you do not want to log.


c28x_ccp_ert: External Signal & Triggering

Signal selection

Trigger	Selected	Block	Path
	X	CCP Stateflow	c28x_ccp_ert/CAN Calibration Protocol1/D...
	X	Chart	c28x_ccp_ert/CAN Calibration Protocol1/D...
	X	Display	c28x_ccp_ert/Display
	X	Scope1	c28x_ccp_ert/Scope1
	X	Scope2	c28x_ccp_ert/Scope2
	X	Scope3	c28x_ccp_ert/Scope3
	X	Scope4	c28x_ccp_ert/Scope4

Select all ✓

Trigger options

Source: manual Mode: normal Duration: 1000 Delay: 0

✓ Arm when connecting to target

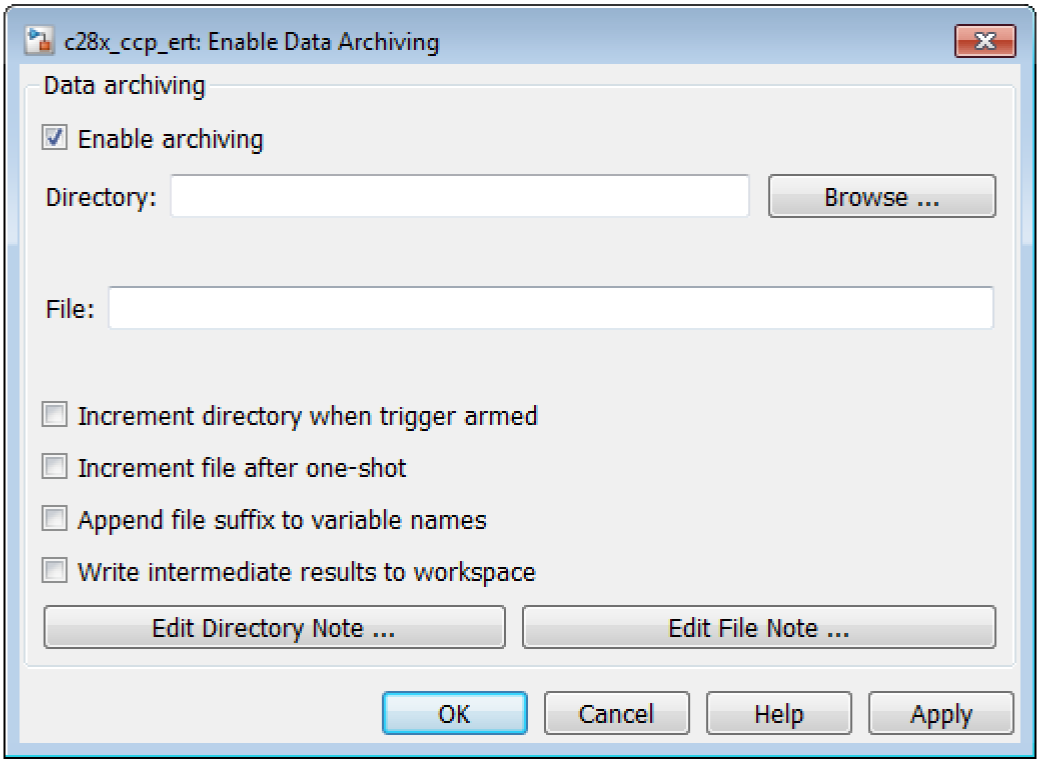
Trigger signal

Path: Port: 1 Element: any

Direction: rising Level: 0 Hold-off: 0

Storing signal data for further analysis. You can further analyze the logged data in MATLAB.

1 To use the data archiving in external mode, in the External Mode Control Panel, click **Data Archiving**.



☑ Enable archiving

Directory:

File:

☐ Increment directory when trigger armed

☐ Increment file after one-shot

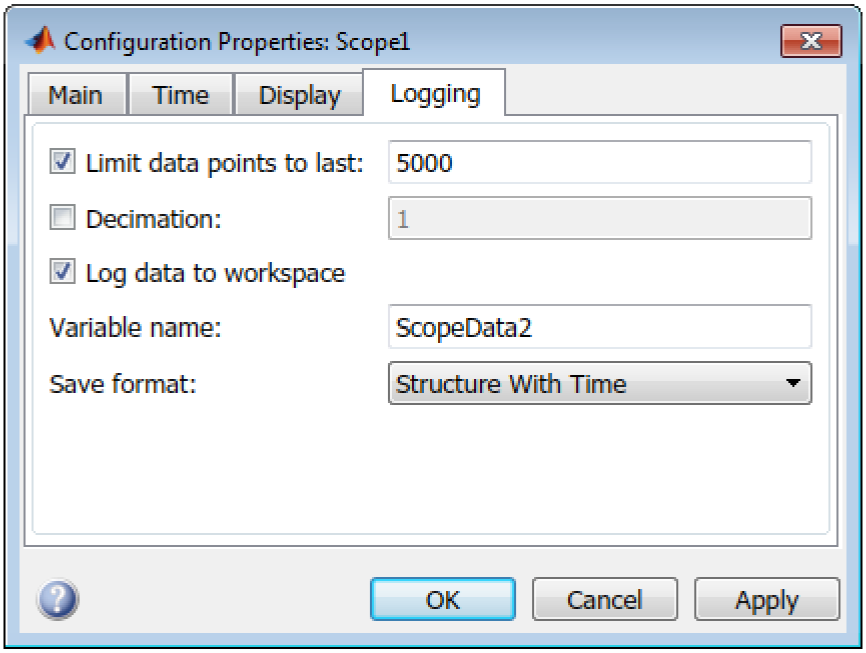
☐ Append file suffix to variable names

☐ Write intermediate results to workspace

- **a** In the Enable Data Archiving dialog box, select the check box **Enable archiving**.
- **b** Edit the Directory and File fields. Review the other settings.
- **c** Click **Apply** and close the dialog box.

2 Open the Scope parameters by clicking the scope parameters icon in the scope window toolbar.

Select the check box **Save data to workspace**.


Configuration Properties: Scope1

Main | Time | Display | Logging

- [✓] Limit data points to last: 5000
- [ ] Decimation: 1
- [✓] Log data to workspace

Variable name: ScopeData2

Save format: Structure With Time

3 In the **Variable name** field, edit the variable name. The data that is displayed in the scope at the end of the external mode session is available in the workspace with this variable name.

The data that was previously displayed in the scope is stored in `.mat` files.

For example, at the end of an external mode session, the following variable and files are available in the workspace and the current folder:

- A variable `ScopeData5` with the data currently displayed on the scope:

```
ScopeData5

ScopeData5 =

         time: [56x1 double]
      signals: [1x1 struct]
    blockName: 'c28x_ccp_ert/Scope1'
```

- In the current folder, .mat files for the three previous durations of scope data:

```
ExternalMode_0.mat
ExternalMode_2.mat
ExternalMode_1.mat
```

Limitations

Multiple signal sinks (for instance, scopes) are not supported.

Only the following kinds of scopes are supported with external mode logging:

- Simulink Scope block
- Simulink Display block
- Viewer type: scope — To use this option in your model, right-click a signal in the model, and select **Create & Connect Viewer** > **Simulink** > **Scope**. The other scope types listed are not supported (for instance, floating scope).

  Before connecting to external mode, right-click the signal and select **Signal Properties**. In the dialog box, select the **Test point** check box, and click **OK**.
- GRT is supported only for parameter tuning.
- You cannot log signals with sample rates more than 10 kHz.
- Top-level builds are supported for external mode. Subsystem builds are not supported for external mode.
- Logging and tuning of nonscalars is not supported. It is possible to log nonscalar signals by breaking down the signal into its scalar components. For an example of how to do this signal deconstruction, see the CCP example models, which use a Demux and Signal Conversion block with contiguous copy.
- Logging and tuning of complex numbers is not supported. 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

"Parameter Tuning and Signal Logging with Serial Communication"

CAN Calibration Protocol with Third Party Tools

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 1-63 and "Q Format Notation" on page 1-65 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 1-68.

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

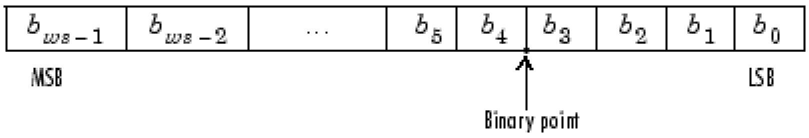
- "Notation" on page 1-63
- "Signed Fixed-Point Numbers" on page 1-64
- "Q Format Notation" on page 1-65

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

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

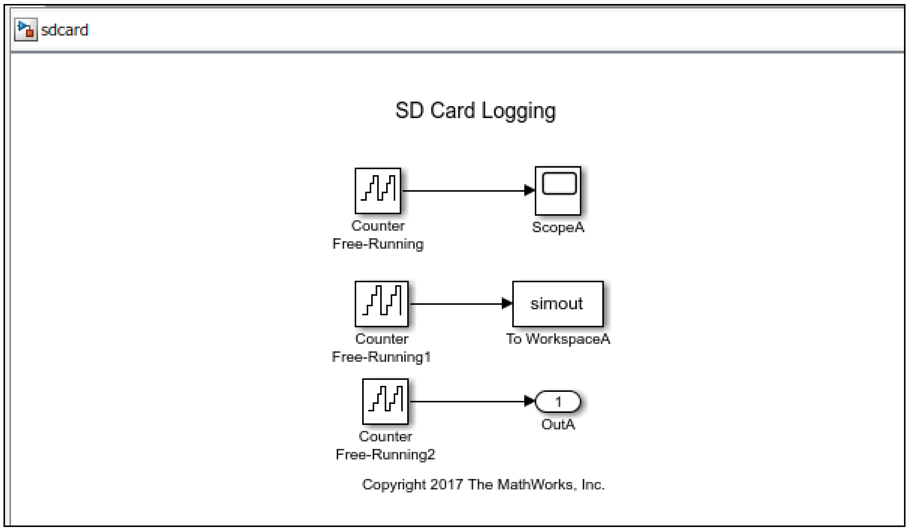
MAT-File Logging on SD Card

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 Model to Log Signals on SD Card” on page 2-7: 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 2-17: 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 2-18: After logging is complete, you can open MAT-files in MATLAB and use them for further analysis.

See Also

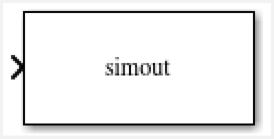
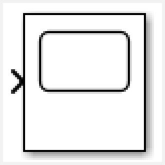
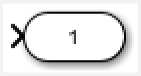
Related Examples

- “MAT-file Logging on SD Card for Texas Instruments C2000 Processors” on page 4-112
- “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

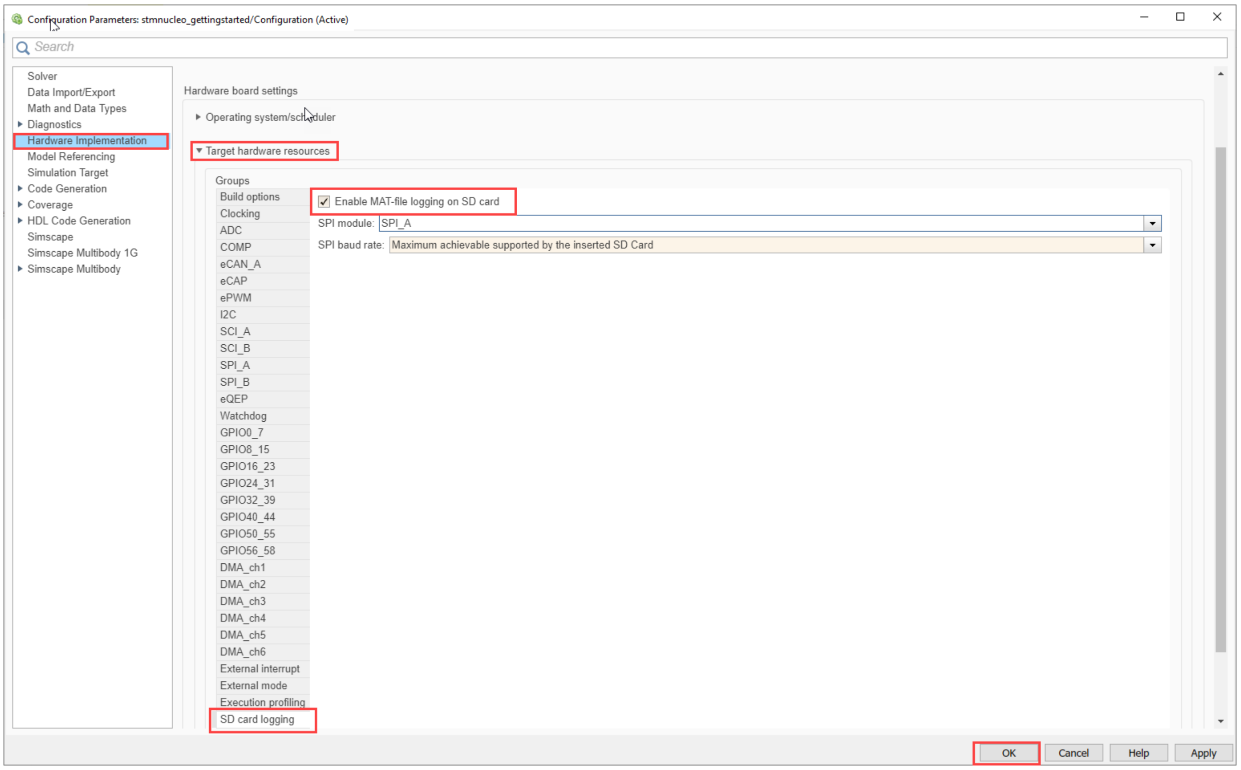
   - For C2000 hardware boards, it is advisable to limit the data points for the signals to be logged as C2000 hardware boards have limited memory. In case of memory allocation failure, set *Limit data points to last* to lesser than |512| value.
   - For F2803x, F2805x, F2833x, F280x and Concerto(35x and 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.


Configuration Parameters: stmnucleo_gettingstarted/Configuration (Active)

Hardware board settings

Target hardware resources

Groups: SD card logging

☑ 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.

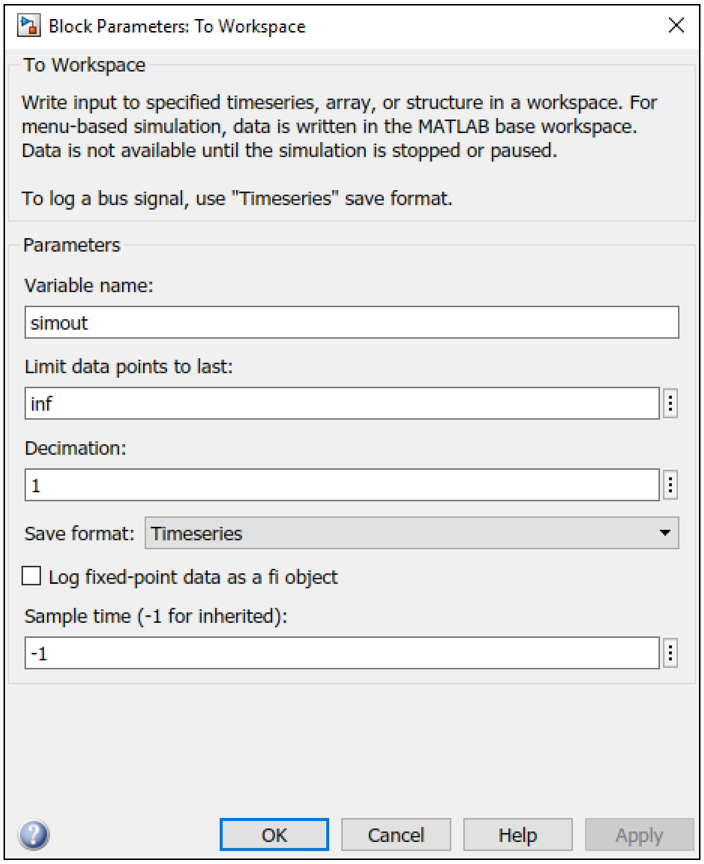
To Workspace Block

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.

Parameter	Description
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.
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

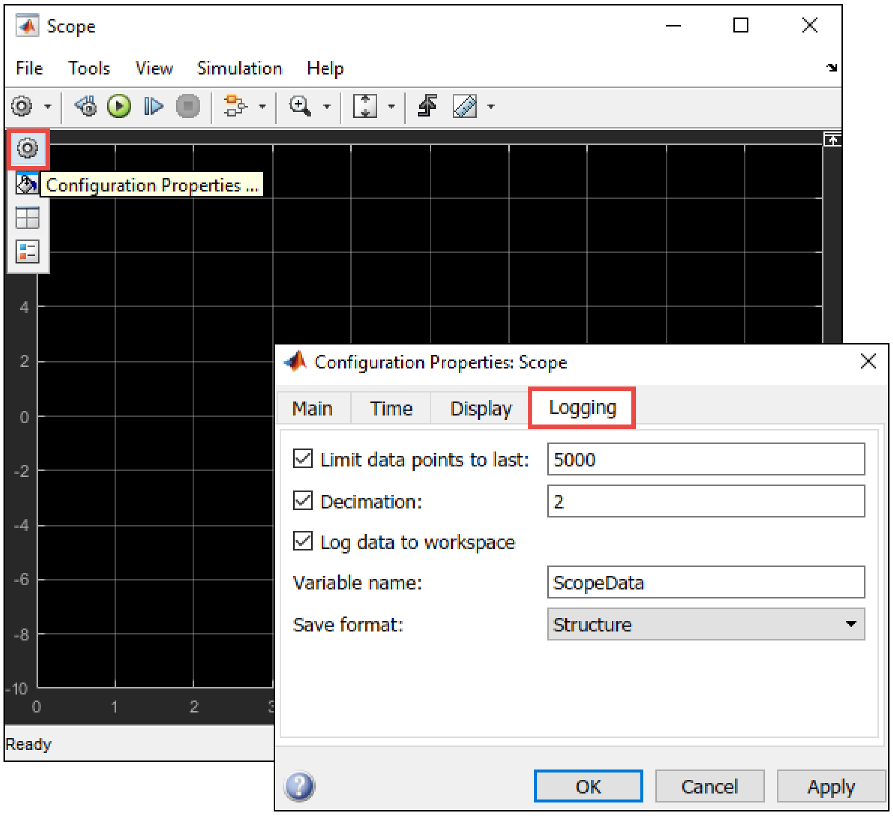
Scope Block

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.

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.

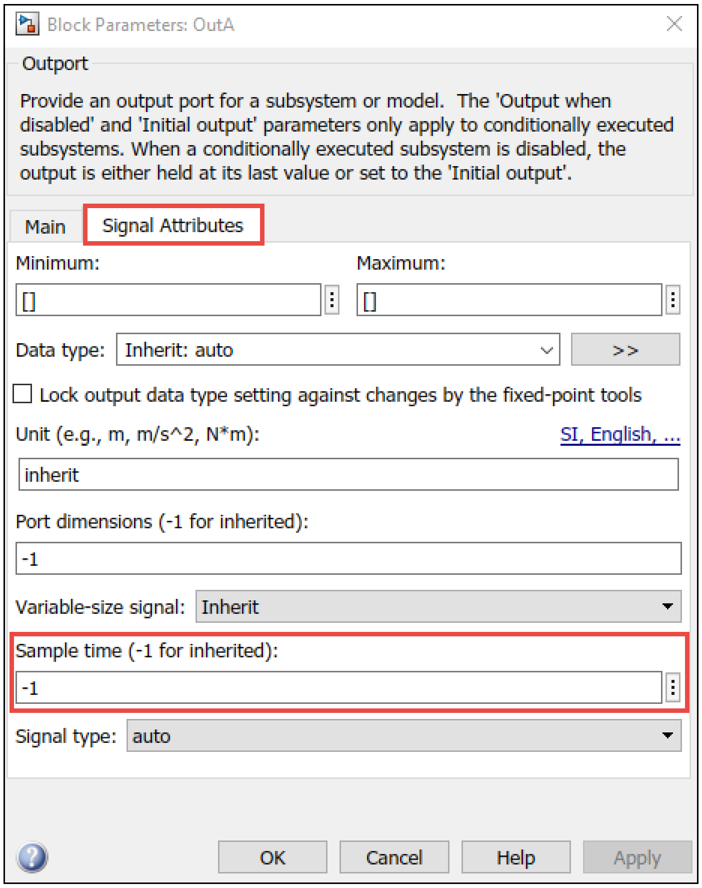

Scope
File Tools View Simulation Help
Configuration Properties ...
Ready
Configuration Properties: Scope
Main Time Display Logging
OK Cancel Apply

☑ 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, ...

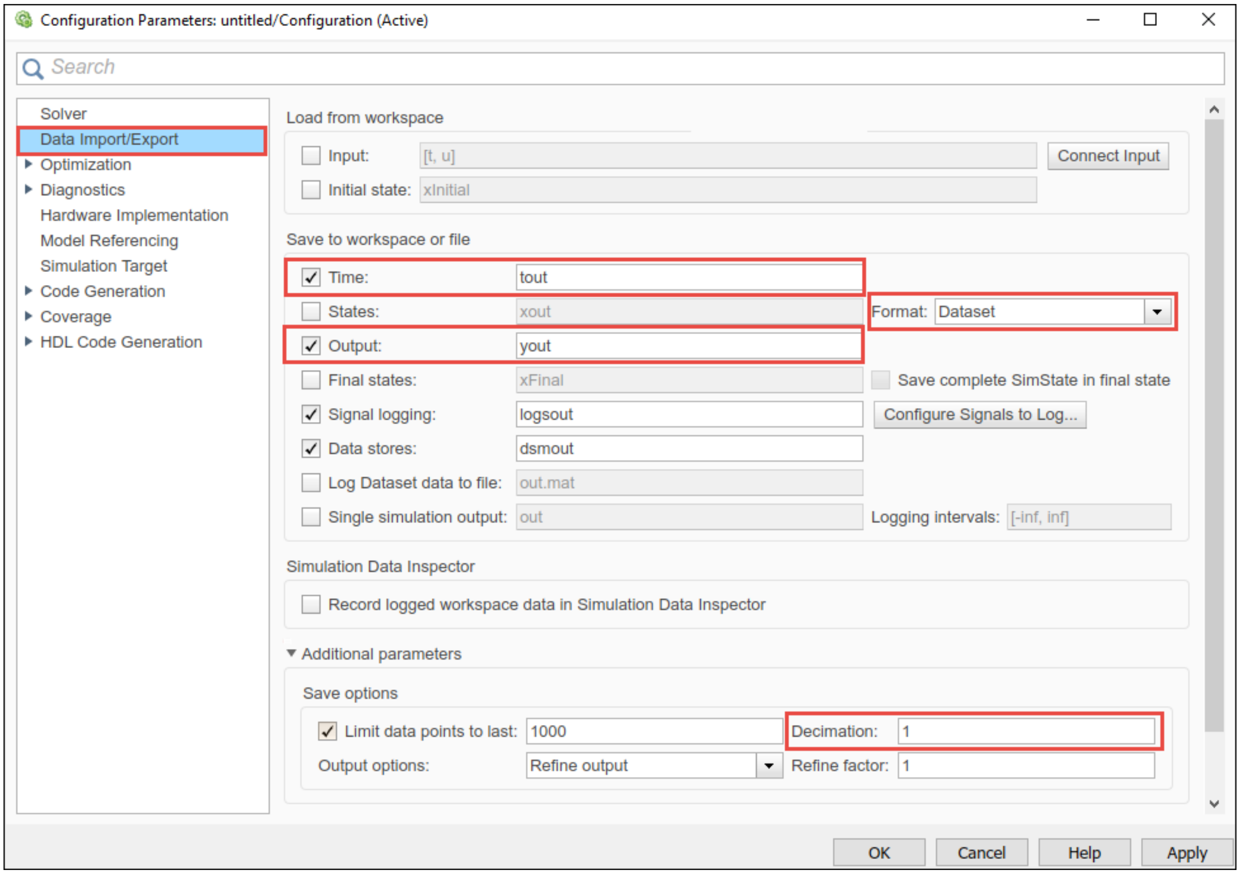
Port dimensions (-1 for inherited): -1

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**.


Configuration Parameters: untitled/Configuration (Active)

Search

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

Additional parameters

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.

Parameter	Description
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	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

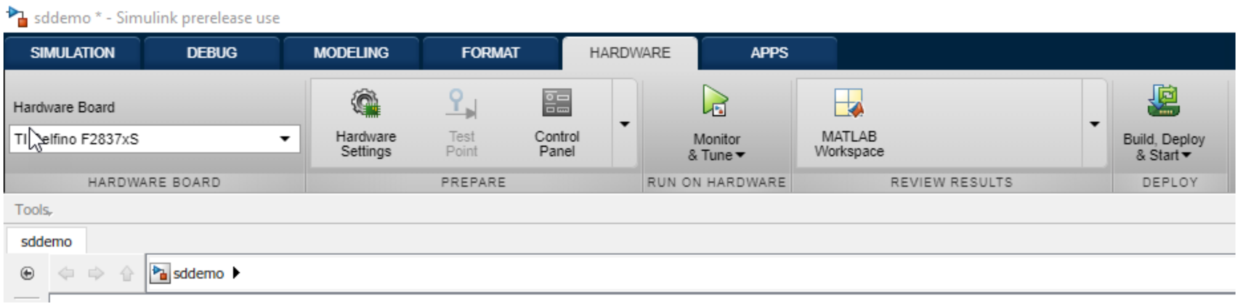
Related Examples

- "MAT-file Logging on SD Card for Texas Instruments C2000 Processors" on page 4-112
- "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.

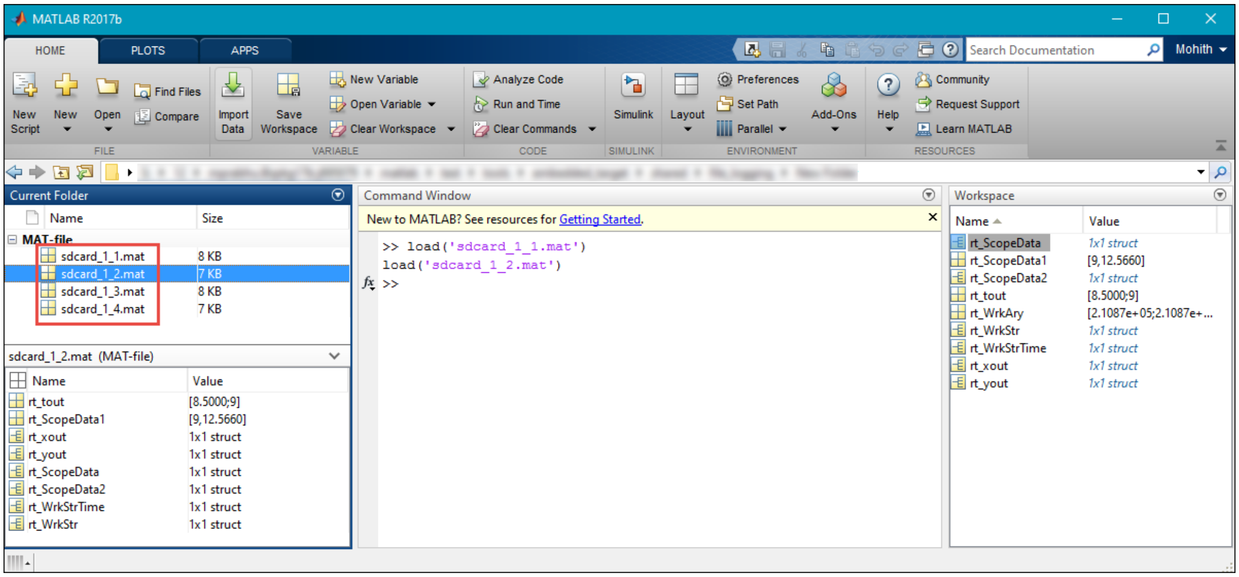
In C2000, remove the SD Card from the board, connect it to a computer and read the .mat files from the explorer.

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 mutiple 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.



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 4-112
- “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 Enable MAT-File Logging on page 2-5 .

- 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 4-112

Examples

```
<xi:include href="urn:mathworks:ex:texasinstrumentsc2000/PFCMultiAxisKitExample.xml" xm
<xi:fallback><!-- Fallback for texasinstrumentsc2000/PFCMultiAxisKitExample --><link xm
</xi:include>
```

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.

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

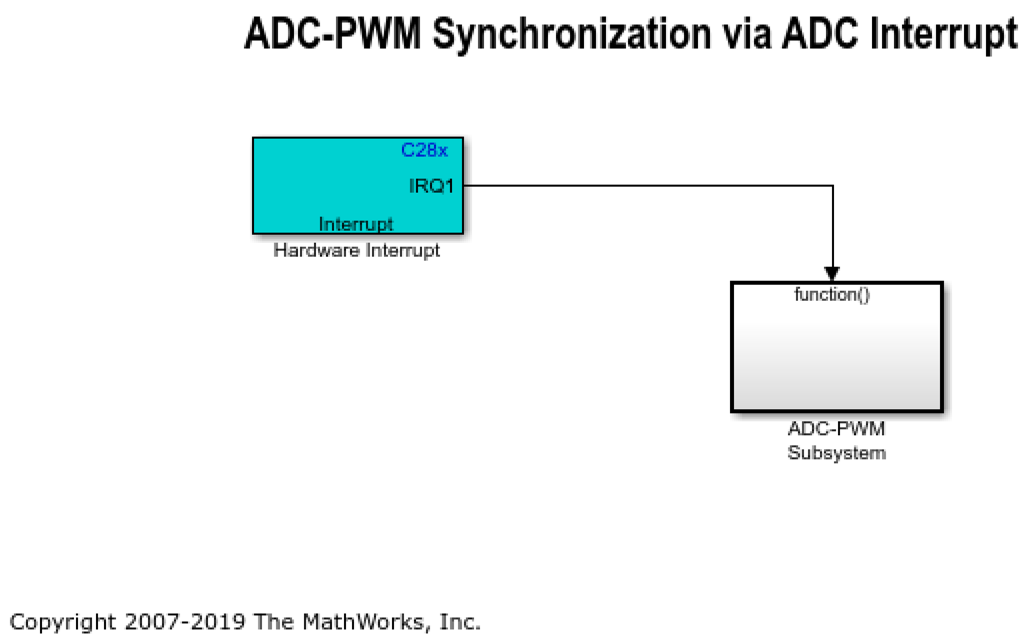
These are the Simulink models available for different C2000 processors:

- 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

Note: To use the F28M35x/F28M36x controlCARD, you need Embedded Coder® Support Package for Texas Instruments C2000™ F28M3x Concerto™ Processors.

Model

The following figure shows the example model.


ADC-PWM Synchronization via ADC Interrupt

Copyright 2007-2019 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 ePWM
- C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x ADC
- C28x Hardware Interrupt

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.

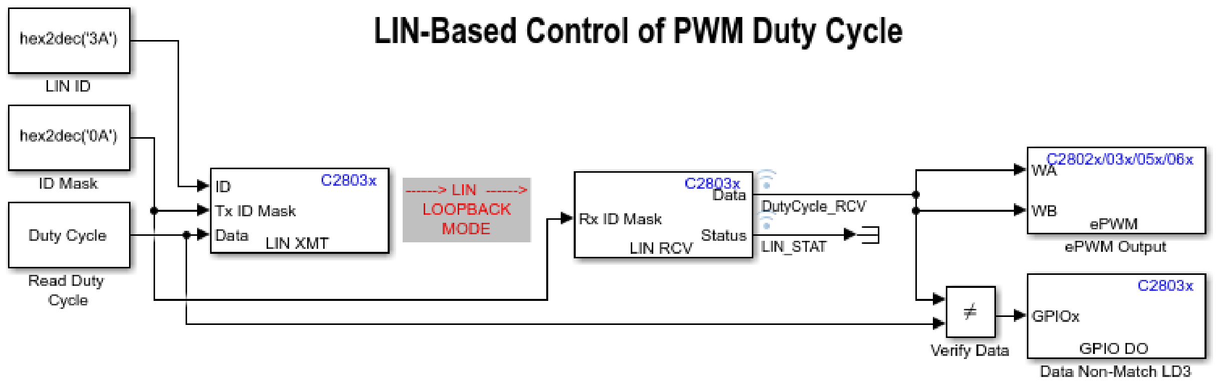
Required Hardware

- 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 1-70.

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. For more information, see

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

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

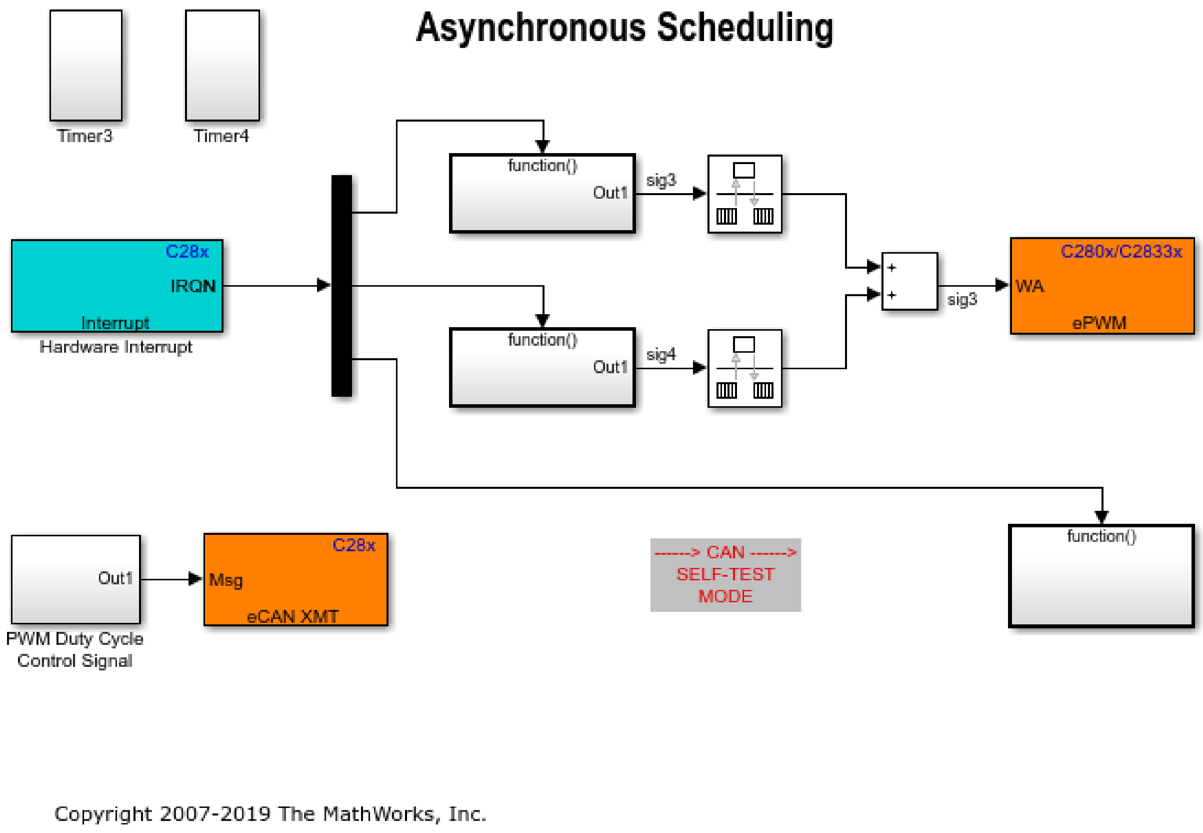
Available Models

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 Embedded Coder® Support Package for Texas Instruments C2000™ F28M3x Concerto™ Processors.

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:

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 ePWM
- C28x eCAN Transmit

Using CAN Calibration Protocol for Monitoring and Tuning

This example shows how to use the CAN Calibration Protocol (CCP) block to monitor model signals and tune parameter values in the application code running on the target hardware. You can use either use **Monitor and Tune** or a third-party calibration tool to interact with the application. The parameters can be loaded in the hardware flash memory and copied to the RAM during initialization.

Required Hardware

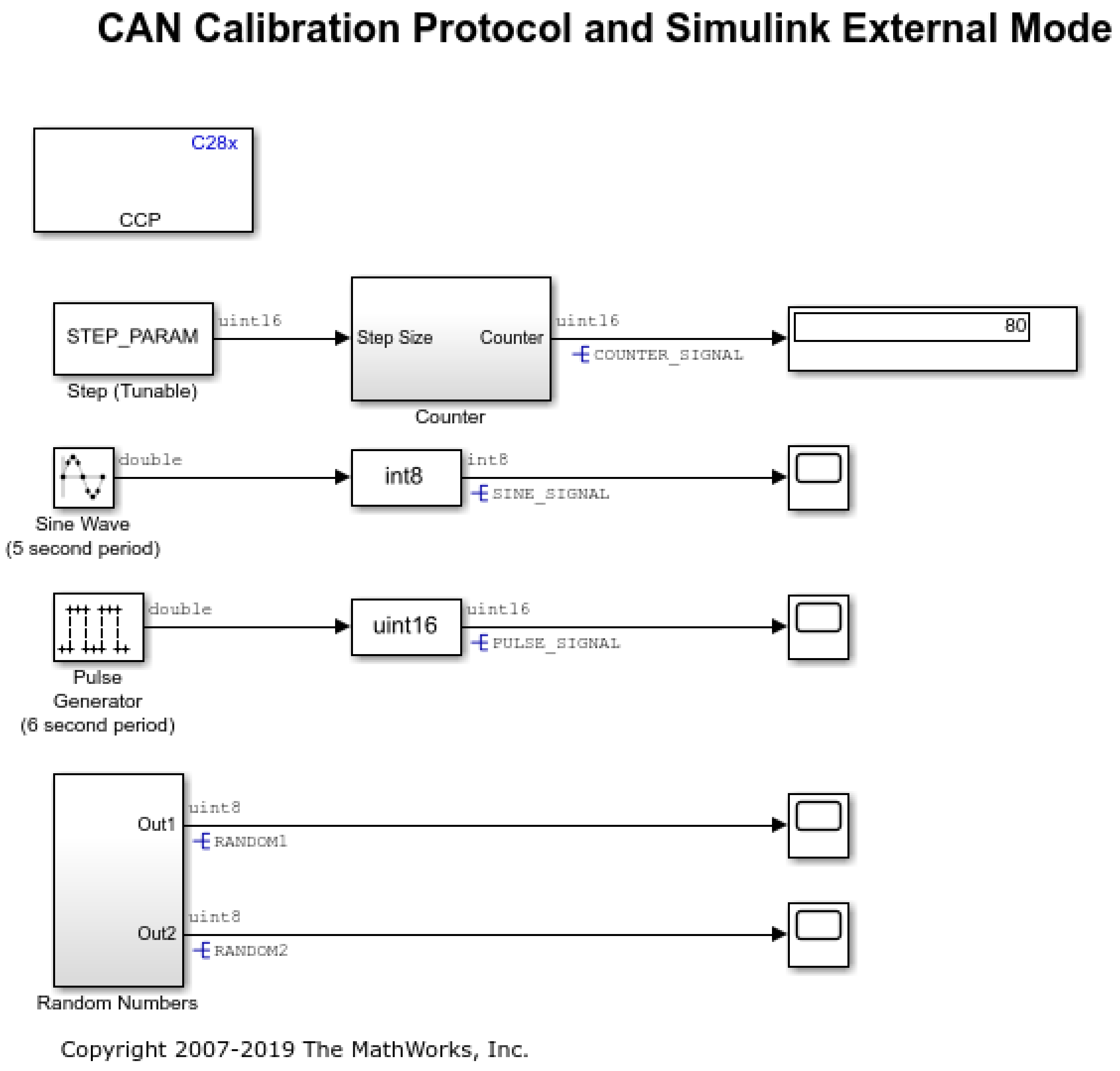
- Vector-Informatik CAN hardware and drivers must be installed on your host computer, and the baud rate must be set to 1 MBd to match the processor CAN configurations. For more information, see “Set Up CAN Communication with Target Hardware” on page 1-10.
- Spectrum Digital F2812/F2808/F28335 eZdsp board or F2808/F28035/F28044/F28069/F28335 controlCARD with Peripheral Explorer kit.

Available Models

These are the Simulink models available for different C2000 processors:

- The example model c28x_ccp_ert.slx can be used for Texas Instruments™ Piccolo F2803x/F2806x, F28x3x, F280x, F281x, F2807x, F2837x, and F28004x target hardware. By default, the F28335 target hardware is selected.
- To change the target hardware, browse to **Configuration Parameters > Hardware Implementation > Hardware board**.

Example Model



During code generation, an ASAP2 file is generated. The ASAP2 file contains symbol and memory address information. Either External mode or a third-party calibration tool can use the generated ASAP2 file to log the signals and update the parameters on the real-time application generated for the model.

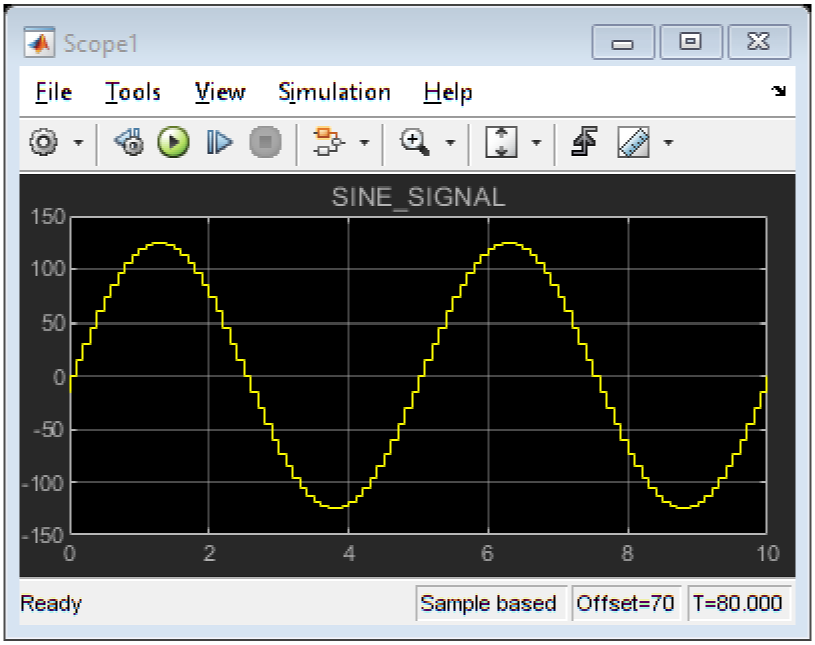
To enable `ASAP2` file generation, browse to **Configuration Parameters** > **Code Generation** > **Interface** and select the **ASAP2 interface** option.

Run the Model

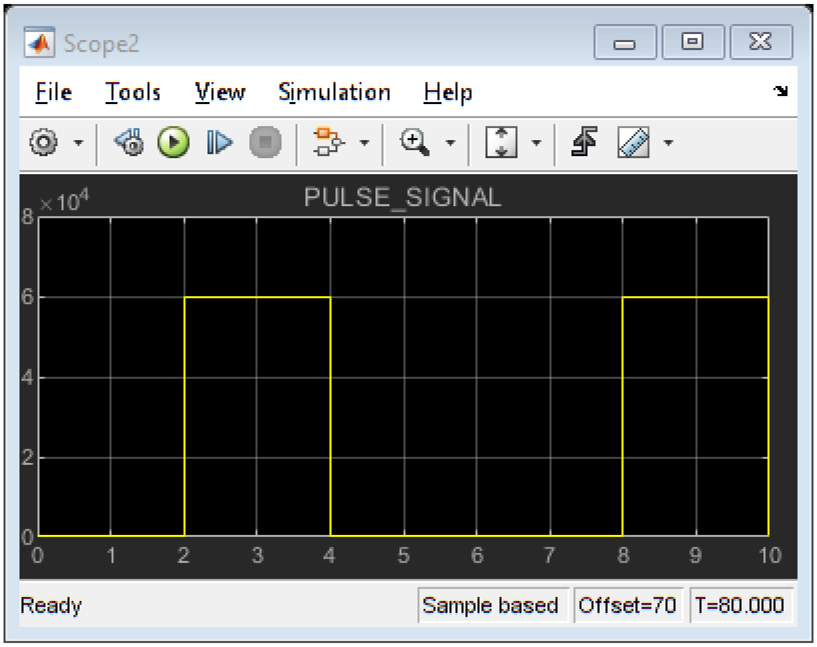
While opening the model c28x_ccp_ert.slx, a script is processed as a pre-load function callback. This callback can be viewed by selecting **File** > **Model Properties** > **Callbacks** > **PreLoadFcn**.

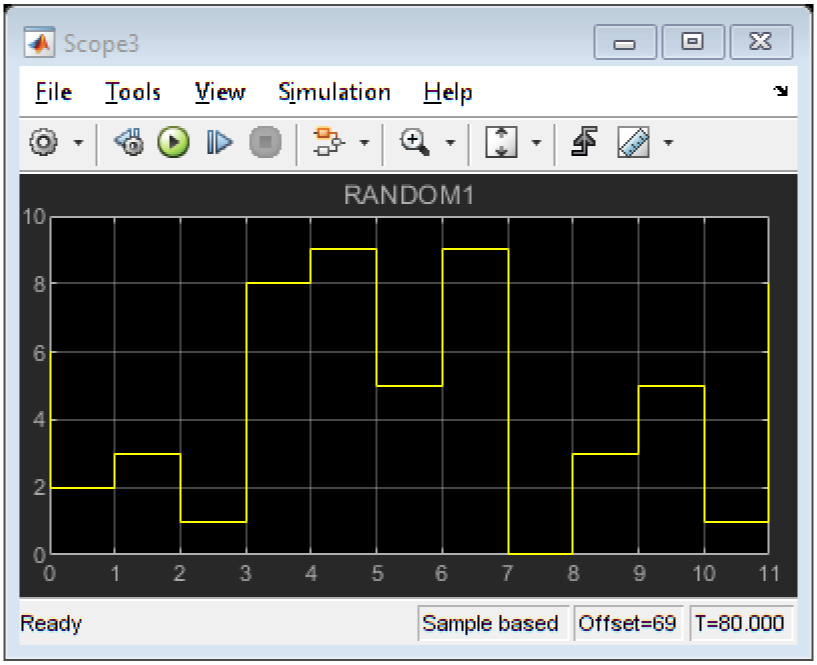
The `c2000_ccp_data.m` callback sets the Simulink® signals and parameters, thereby resolving the Simulink objects needed for the CCP DAQ lists and Simulink tunable parameters. Now, you can monitor the model signals and tune parameter values in real-time using CCP.

1. Browse to **Configuration Parameters** > **Hardware Implementation** > **Hardware board**, and select the required hardware board.
2. Ensure that **Communication interface** is set to **CAN**. The selection can be made at **Configuration Parameters** > **Hardware Implementation** > **Target hardware resources** > **External Mode** > **Communication interface**.
3. Open **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 logged data on the scope.



Sample based | Offset=70 | T=80.000


Scope2
File
Tools
View
Simulation
Help
PULSE_SIGNAL
×10^4
8
6
4
2
0
0
1
2
3
4
5
6
7
8
9
10
Ready
Sample based
Offset=70
T=80.000



Signal Logging and Parameter Tuning

The model c28x_ccp_ert.slx contains a variety of signals and parameters that can be logged or changed in real-time on the target hardware. Parameter tuning and signal logging are enabled by the CCP block included in the model. Additionally, when the model is built, it must be configured to generate an ASAP2 file. The ASAP2 file contains information about the signals and parameters that are available for logging or tuning in the real-time application.

The model contains a simple counter with parameterized step size, `STEP_PARAM`. `STEP_PARAM` takes advantage of a custom storage class package labeled `tic2000demospkg` to load the parameters on the flash memory and run them from the RAM for calibration using the `ramfuncs` section available in the memory map.

For this example, a custom storage class package labeled `tic2000demospkg` has been created. Run cscdesigner -advanced tic2000demospkg to view the definition of the `tic2000demospkg` package.

This model can be run from the flash memory (standalone) and also from the RAM. In both cases, `STEP_PARAM` is stored in the ramfuncs section. You can update the parameter

`STEP_PARAM` and log the output of the counter, COUNTER_SIGNAL. Additional signals available for logging are SINE_SIGNAL, PULSE_SIGNAL, and the RANDOMx signals.

To change the parameter `STEP_PARAM` in the real-time application:

1. Change its value, `STEP_PARAM.Value`, in the MATLAB workspace.
2. Update the model to apply this new value to the real-time application by selecting **Simulation** > **Update Diagram** or pressing **Ctrl+D**.

Signal Logging and Parameter Tuning Using a Third-Party Calibration Tool

As an alternative to **Monitor and Tune**, you can use a third-party calibration tool for signal logging and parameter tuning. The same signals and parameters that are available with External mode may be logged or updated using the third-party calibration tool. While using a third-party calibration tool, ensure that the tool is set in compliance with the word addressable nature of the Texas Instruments C2000 processors (16-bit addressable).

More About

- C28x CAN Calibration Protocol
- “Set Up CAN Communication with Target Hardware” on page 1-10

Simulation of FOC Using PMSM Model

This example shows how to simulate field-oriented control (FOC) using a Permanent Magnet Synchronous Machine (PMSM) model. The model is created using the SimPowerSystems™ toolbox, C28x peripherals, and DMC library blocks.

Using this example, you can:

- Simulate FOC using a PMSM model
- Generate code for the embedded controller

Required Hardware

- Spectrum Digital® F2808 or F2812 eZdsp board
- Digital Motor Controller board: Spectrum Digital® DM550
- Three-phase Permanent Magnet Synchronous Motor with quadrature encoder

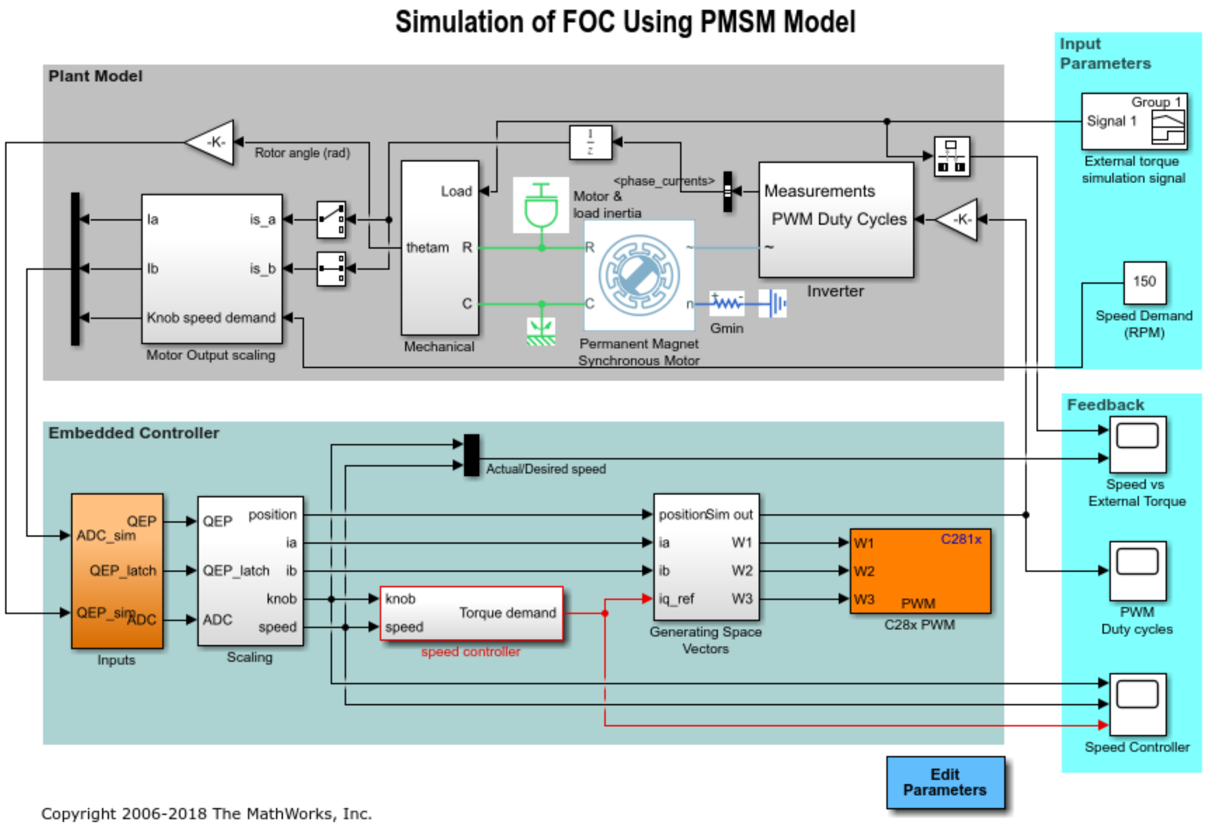
Note: The characteristics of the power supply and the amplifier must be the same as the input characteristics of the selected motor.

Available Models

- Spectrum Digital F2812 eZdsp: c2812pmsmsim.slx
- Spectrum Digital F2808 eZdsp: c2808pmsmsim.slx

Model

The following figure shows the Permanent Magnet Synchronous Motor Field-Oriented Control example model.



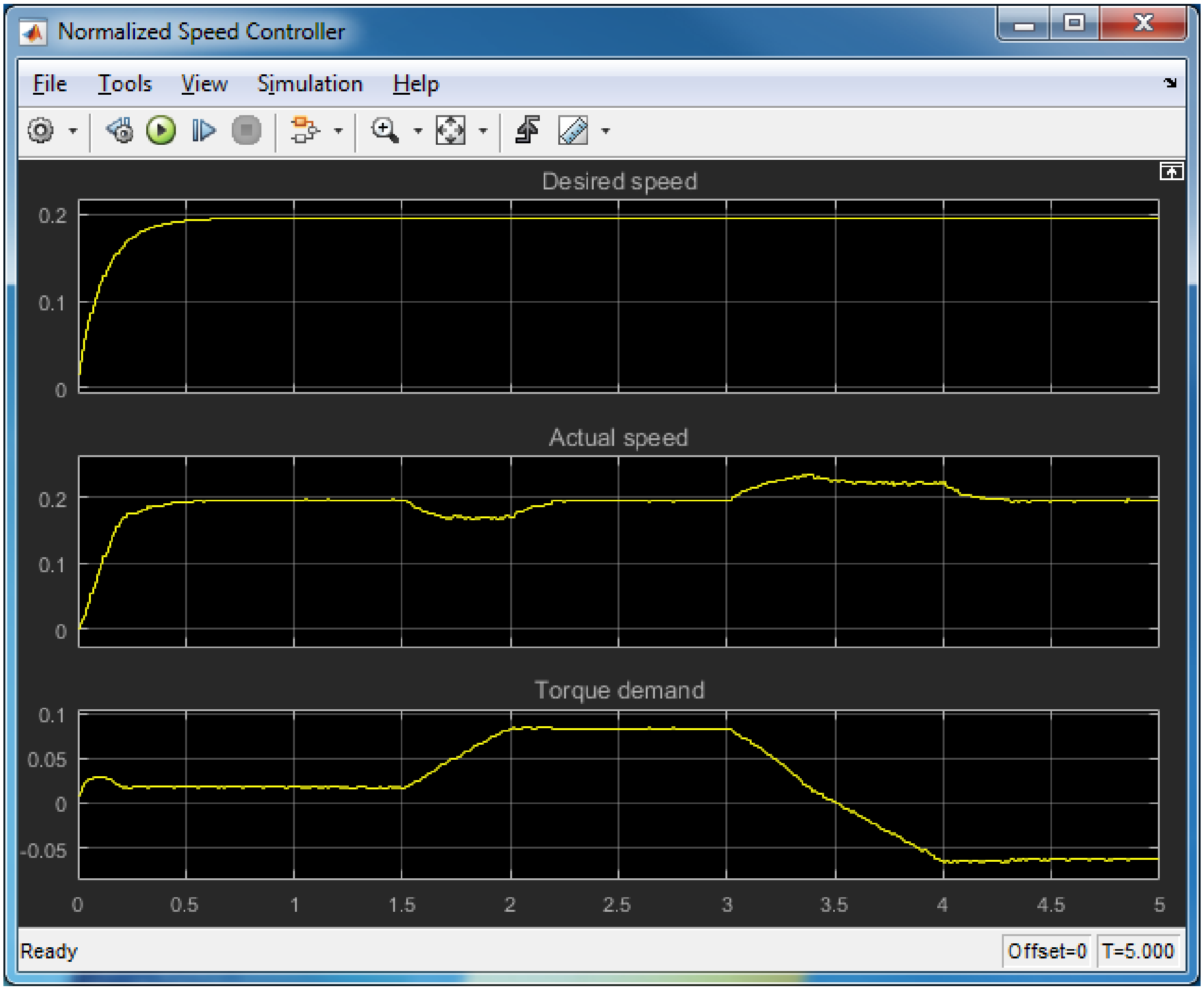
Simulate FOC Using a PMSM Model

The c2812 peripheral and DMC library blocks are used to control the speed and torque of a three-phase Permanent Magnet Synchronous Machine from SimPowerSystems in a speed-controlled closed-loop fashion using the field-oriented control technique.

To set up the Permanent Magnet Synchronous Machine, double-click the block and specify the parameters to match your hardware configuration. For more information on setting up the Permanent Magnet Synchronous Machine, see the SimPowerSystems documentation.

External torque can be added to the simulation to reproduce the mechanical action on the shaft of the motor. A default torque is applied at the beginning of the simulation and is reversed after 3 seconds of simulation. The controller model executes a speed controller algorithm and reacts to the torque changes to maintain the desired speed. You can achieve optimal results by tuning the PID controllers on the Embedded Controller side to suit the motor.

The following figure shows the speed of the PID controller in the speed controller subsystem. The desired speed is filtered and ramps up to a defined speed value. This value stays constant until the end of the simulation. At the beginning of the simulation, torque is applied on the shaft so that the motor spins by itself in one direction until enough opposite torque is accumulated in the controller to regulate the speed to the desired value. At time = 3 seconds, a different value of external torque is applied on the shaft of the motor, forcing the controller to change its output to maintain the desired speed.



Generate Code for the Embedded Controller

To generate code for the Embedded Controller, delete the **Plant Model** and the **Input Parameters** blocks and press **Ctrl+B**.

Connect the motor to the DM550 Digital Motor Controller and the eZdsp board to the DM550 Digital Motor Controller, using the standard cables. Follow the instructions given by the controller vendor. By default, the model is set up for motors with the following characteristics:

- 2000 slits/mechanical revolution
- 4 pole pairs
- Encoder index offset: 1850

Change the model parameters to suit the motor. Match the voltage and power characteristics of the motor with the controller parameters. Use the **Edit Parameters** button to change the model parameters of the motor.

The motor is driven by the conventional voltage-source inverter. The controller algorithm generates six pulse width modulation (PWM) signals using vector PWM technique for six power switching devices. Two input currents of the motor (ia and ib) are measured from the inverter and sent to the processor through two analog-to-digital converters (ADCs).

Run the Model

1. Open the model.
2. Remove the Plant Model and Input Parameters blocks.
3. Press Ctrl+B to generate, build, load, and run the controller code.

More About

C2000 PID Controller

Permanent Magnet Synchronous Motor Field-Oriented Control

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 (FOC)using the C28x peripherals and DMC library blocks.

Required Hardware

This example supports multiple hardware configurations:

DM550 Configuration

- Spectrum Digital® F2808-based board, F2808 or F2812 eZdsp board
- Digital Motor Controller board: Spectrum Digital DM550
- Three-phase Permanent Magnet Synchronous Motor with quadrature encoder

DRV8312 Configuration

- TI® DRV8312 Three-Phase Brushless Motor Control Kit (DRV8312-C2-KIT or DRV8312-69M-KIT) with F28035 or F28069 Piccolo or F28M35x or F28M36x Concerto processor(To use F28M35x/F28M36x you need Texas Instruments C2000 Concerto Support Package)
- Three-phase PMSM with optional Hall sensors attached to connector J10 of the DRV8312EVM board

Dual Motor Control Configuration

- TI Dual Motor Control and PFC Developer's kit TMDS2MTRPFCKIT with F28035
- 2 Three-phase PMSM, one with optional Hall sensors attached to connector J4 of the DRV8402-based Dual-Axis DMC EVM (TMDS2MTRPFCKIT)

High Voltage Motor Control Configuration

- TI High Voltage Motor Control and PFC Developer's kit TMDSHVMTRPFCKIT with F28035
- Three-phase PMSM with optional Hall sensors attached to connector H1 of the High Voltage Motor Control Kit (TMDSHVMTRPFCKIT)

TI C2000 LaunchPad with Motor Drive BoosterPack Configuration

- Motor Drive BoosterPack featuring DRV8301 (BOOSTXL-DRV8301) with F28027 LaunchPad (LAUNCHXL-F28027)
- Three-phase PMSM

TI LaunchXL-F28379D/F28377S with Motor Drive BoosterPack Configuration

- Motor Drive BoosterPack featuring DRV8305 (BOOSTXL-DRV8305EVM) with F28379D/F28377S LaunchPad (LAUNCHXL-F28379D/LAUNCHXL-F28377S)
- Two three-phase PMSM coupled to form a Dyno (2MTR-DYNO)

TI LaunchXL-F28069M with Motor Drive BoosterPack Configuration

- Motor Drive BoosterPack featuring DRV8301 (BOOSTXL-DRV8301) with F28069M LaunchPad (LAUNCHXL-F28069M)
- Two three-phase PMSM coupled to form a Dyno (2MTR-DYNO)

Note: Match the characteristics of the power supply and the amplifier with the input characteristics of the selected motor.

Available Models

The following are the example models available for different driver-controller combinations:

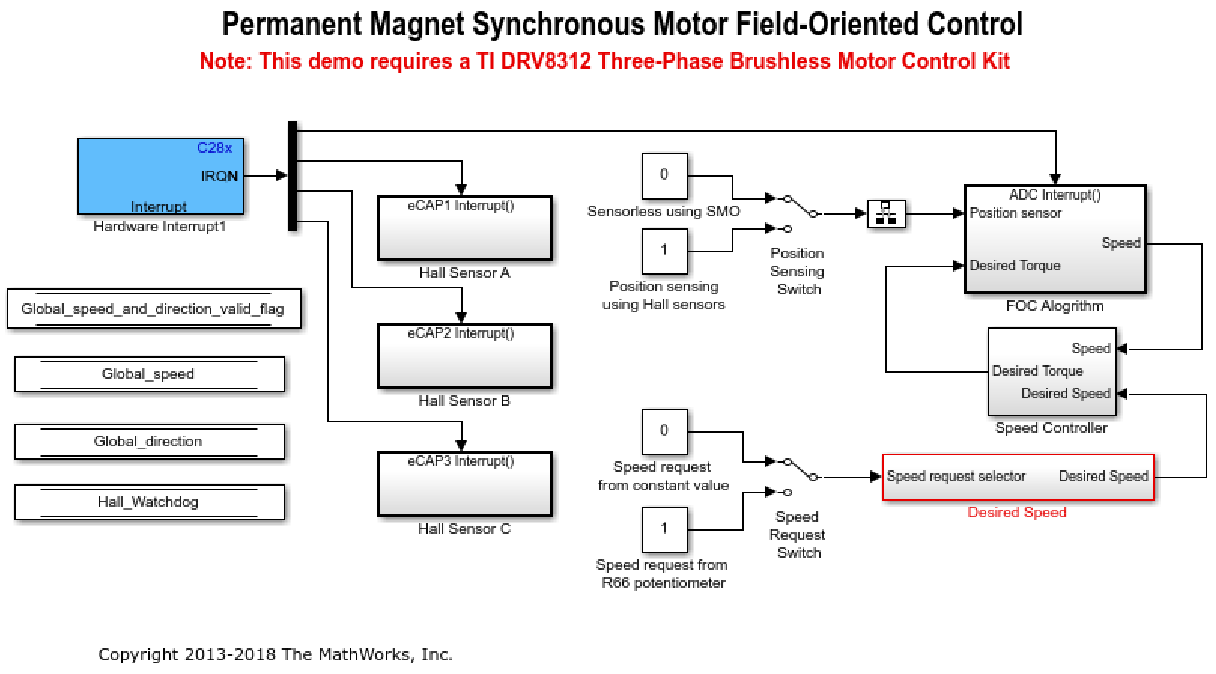
- DM550 + F2812 eZdsp: c2812pmsmfoc_ert.slx
- DM550 + F2812 eZdsp with flash-based memory map: c2812pmsmfocflash_ert.slx
- DM550 + F2808 eZdsp: c2808pmsmfoc_ert.slx
- DM550 + F2808 eZdsp with flash-based memory map: c2808pmsmfocflash_ert.slx
- DM550 + F28335 eZdsp: c28335pmsmfoc_ert.slx
- DM550 + F28335 eZdsp with flash-based memory map: c28335pmsmfocflash_ert.slx
- DRV8312EVM + F28035 Piccolo: c28035pmsmfoc_ert.slx
- DRV8312EVM + F28069 Piccolo: c28069pmsmfoc_ert.slx
- Dual-Axis DMC EVM + F28035 Piccolo: c28035pmsmfocdual.slx
- DRV8312EVM + F28M35x Concerto: c28M35xpmsmfoc_ert.slx
- DRV8312EVM + F28M36x Concerto: c28M36xpmsmfoc_ert.slx
- TMDSHVMTRPFCKIT + F28035 Piccolo: c28035pmsmfochvkit_ert.slx

- BOOSTXL-DRV8301 + LAUNCHXL-F28027 C2000 Piccolo LaunchPad: c28027pmsmfoc_ert.slx
- 2 BOOSTXL-DRV8301 + LAUNCHXL-F28069M C2000 Piccolo LaunchPad: c28069pmsmfocdual_ert.slx
- 2 BOOSTXL-DRV8305 + LAUNCHXL-F28377S C2000 LaunchPad: c28377Spmsmfocdual_ert.slx
- 2 BOOSTXL-DRV8305 + LAUNCHXL-F28379D C2000 LaunchPad: c28379Dpmsmfocdual_cpu1_ert.slx, c28379Dpmsmfocdual_cpu2_ert.slx

Note: In LaunchXL F28379D (Version 2.0), the ADC offset for Ia and Ib measurements are different from the earlier boards. Open the model "c28379Dpmsmfocdual_cpu1_ert.slx", "c28379Dpmsmfocdual_cpu2_ert.slx" and change the postion of the manual switch appropriately, depending on the version of the board. To change the manual switch setting in your model, browse to **FOC Alogrithm Motor** > **Torque Control Algorithm** > **Input Scaling**. It is recommended to calibrate the ADC offset, because the ADC offset varies for different boards.

Model

The following figure shows a Permanent Magnet Synchronous Motor Field-Oriented Control example model.



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 and C28x DMC library blocks from the Embedded Coder Support Package for Texas Instruments C2000 Processors.

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**. For more information, see “Hardware Implementation Pane: Texas Instruments C2000 Processors”.

The example models named *pmsmfocflash.slx, place the code in the flash memory section of the processor, allowing the application to run in a stand-alone fashion.

The model is set up for motors that have the following characteristics:

DM550 configuration

- 4 pole pairs
- 2000 slits/mechanical revolution
- Encoder index offset: 1850

DRV8312 Configuration

- 4 pole pairs
- Sensorless using sliding mode observer (SMO)
- 3 Hall sensors attached to connector J10 of the DRV8312EVM (optional)

Dual Motor Control Configuration

- 4 pole pairs
- Sensorless using sliding mode observer (SMO)
- 3 Hall sensors attached to connector J4 of the Dual-Axis DMC EVM (optional)

High Voltage Motor Control Kit Configuration

- 4 pole pairs
- Sensorless using sliding mode observer (SMO)
- 3 Hall sensors attached to connector H1 of the High Voltage Motor Control Kit (optional)

LaunchPad with Motor Drive BoosterPack Configuration

- 4 pole pairs
- Sensorless using sliding mode observer (SMO)

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.

Run the Model

1. Open the model that matches your hardware.

If you select the flash version, you must do the appropriate jumper settings on the eZdsp boards. Default jumper settings are set to execute the code from RAM, (refer to the eZdsp Technical Reference for more information).

2. Press **Ctrl+B** to build, load, and run the binary executable.

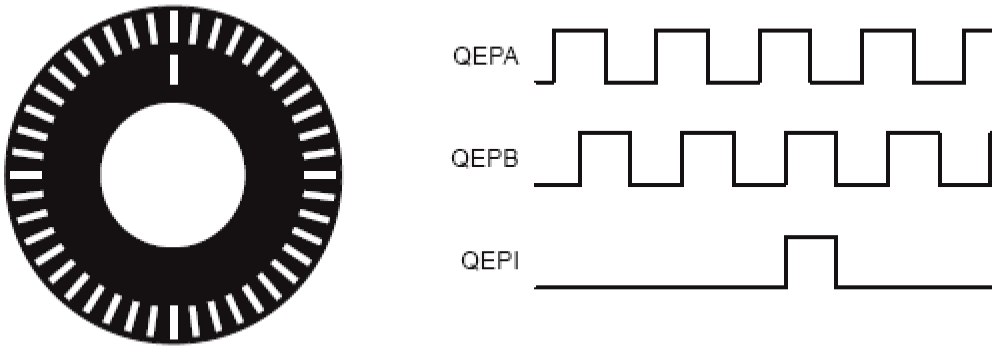
If you are using F28379D Launchpad, you must build and download CPU1 and CPU2 models to respective cores of the processor. Each core controls one motor on the dual motor control kit. Two cores communicate each other using IPC. CPU1 model sends Load Torque request to CPU2. CPU2 sends data to be logged to CPU1. For more information about IPC communication, see “Inter-Processor Communication Using IPC Blocks”.

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.

Model Calibration Using DM550 and QEP

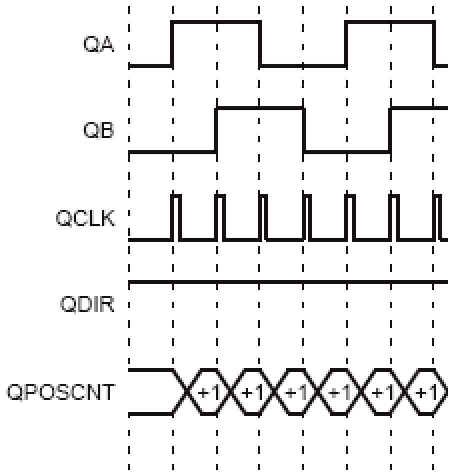
In this example, we are using a 2000 slit encoder with a four pole pair motor. The encoder has 2 channels (QEPA and QEPB) and an index pulse (QEPI).

Figure 1: Optical encoder disk



The timer driven by the QEP increments by 4 for each slit as shown in the following figure.

Figure 2: QEP signals and counter



The 4-pole pair motor in this example produces: * 8000 counts per mechanical * 2000 counts per electrical revolution

The following figure shows an A-N back EMF waveform and the position of the encoder index pulse relative to the back EMF of the motor. The neutral point is obtained by connecting a star resistor network on the 3 legs of the motor.

For F281x processors:

- Phase A is the one controlled by PWM1-2
- Phase B is the one controlled by PWM3-4
- Phase C is the one controlled by PWM5-6

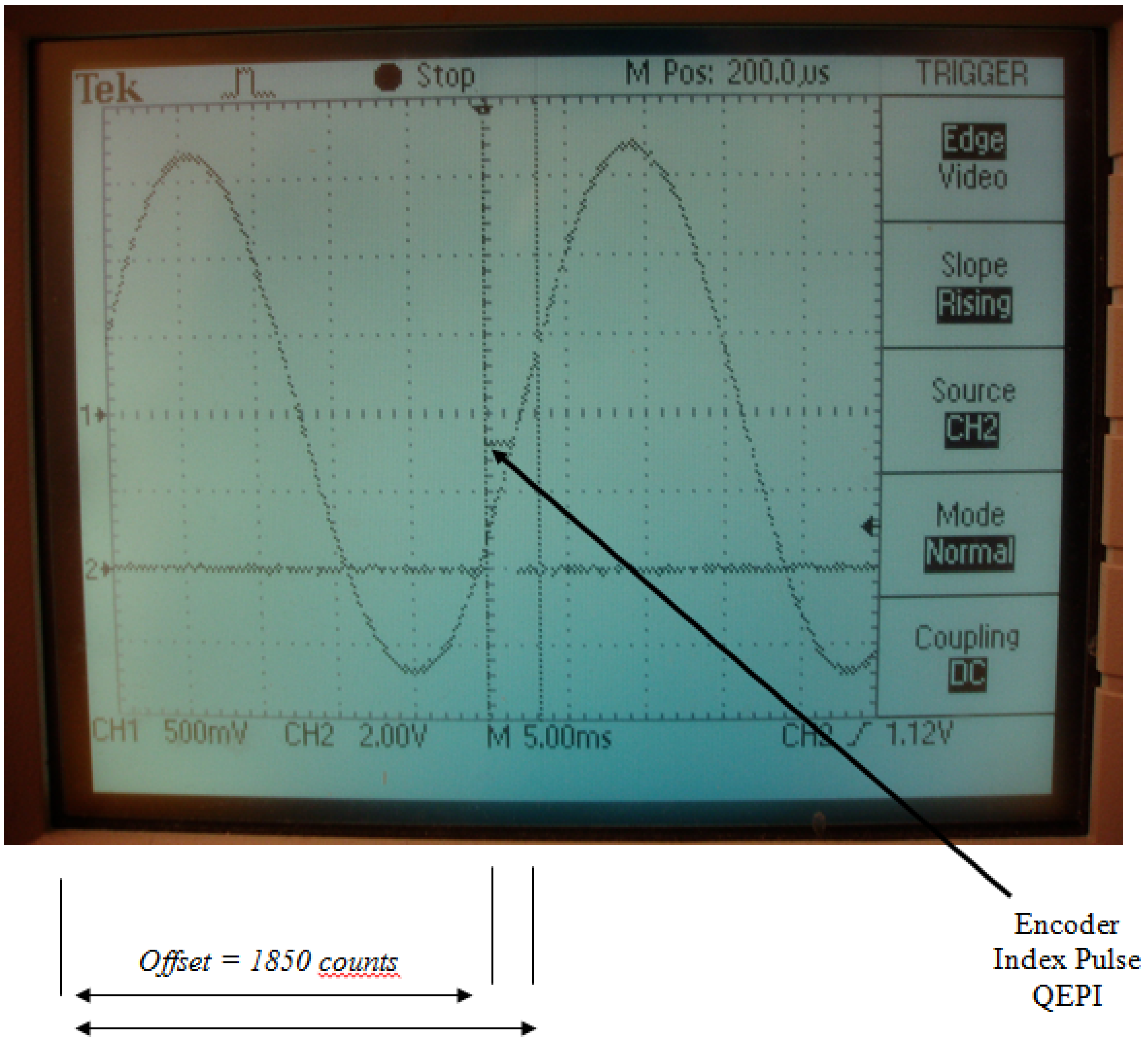
For F280x/F2833x processors:

- Phase A is the one controlled by PWM1A-1B
- Phase B is the one controlled by PWM2A-2B
- Phase C is the one controlled by PWM3A-3B

Spin the motor externally to get B-N 120° out of phase on the right-hand side of A-N on the oscilloscope (Figure 4). Make sure that the counter driven by the encoder is increasing while spinning in this direction. The reference position of the encoder is when

A-N crosses the 0V axis on its way up. Measure the time from that point to the next encoder index pulse on the right-hand side and convert that time into number of counts. Put the result in the "index offset" parameter of "Edit Parameters" block in the model.

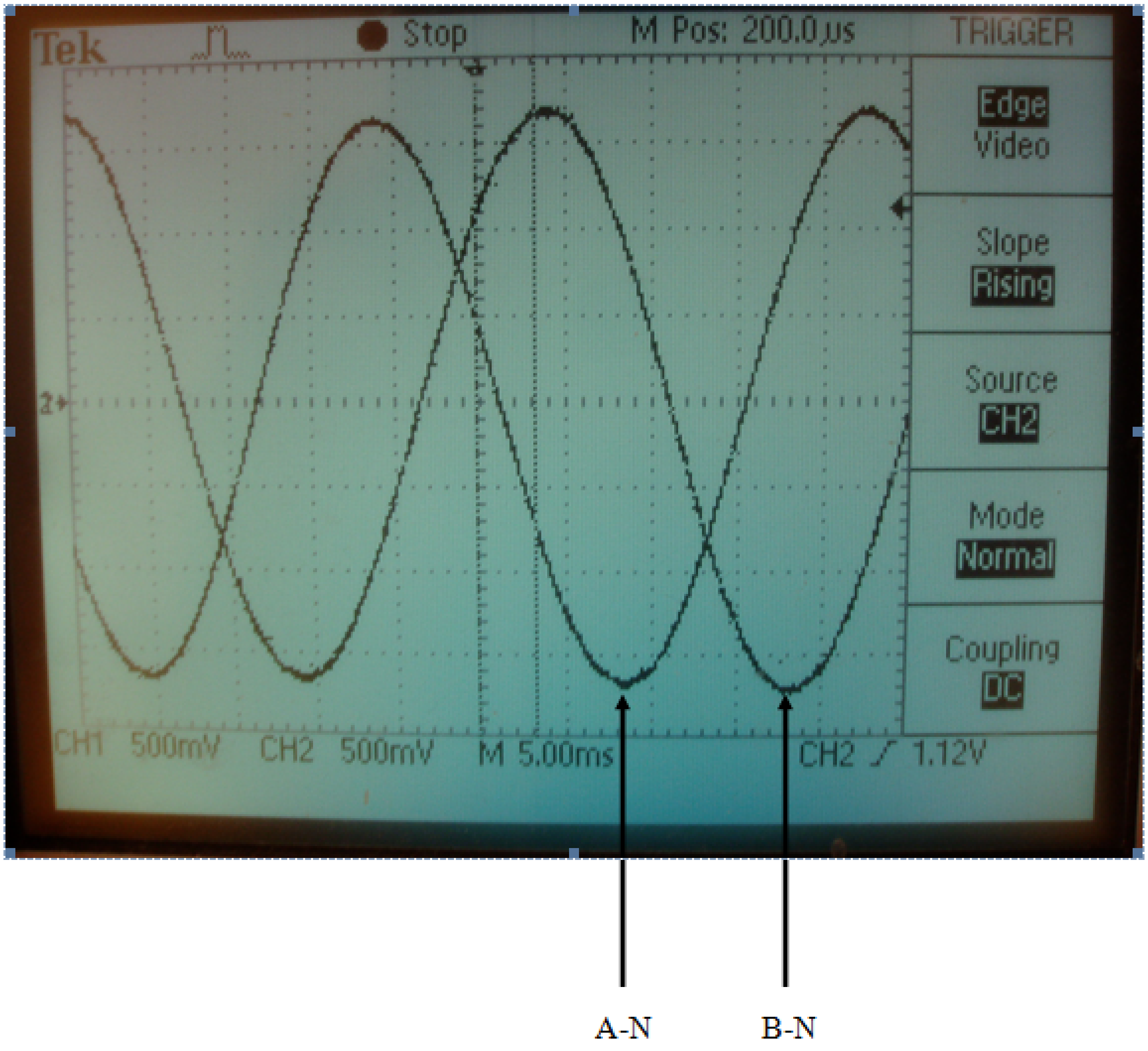
Figure 3: V_A-N waveform with index pulse



$$period = \frac{encoder\_slits * 4}{motor\_pole\_pairs} = 2000\_counts$$

The following picture shows the position of the B-N waveform compared to A-N:

Figure 4: V_A-N and V_B-N waveforms

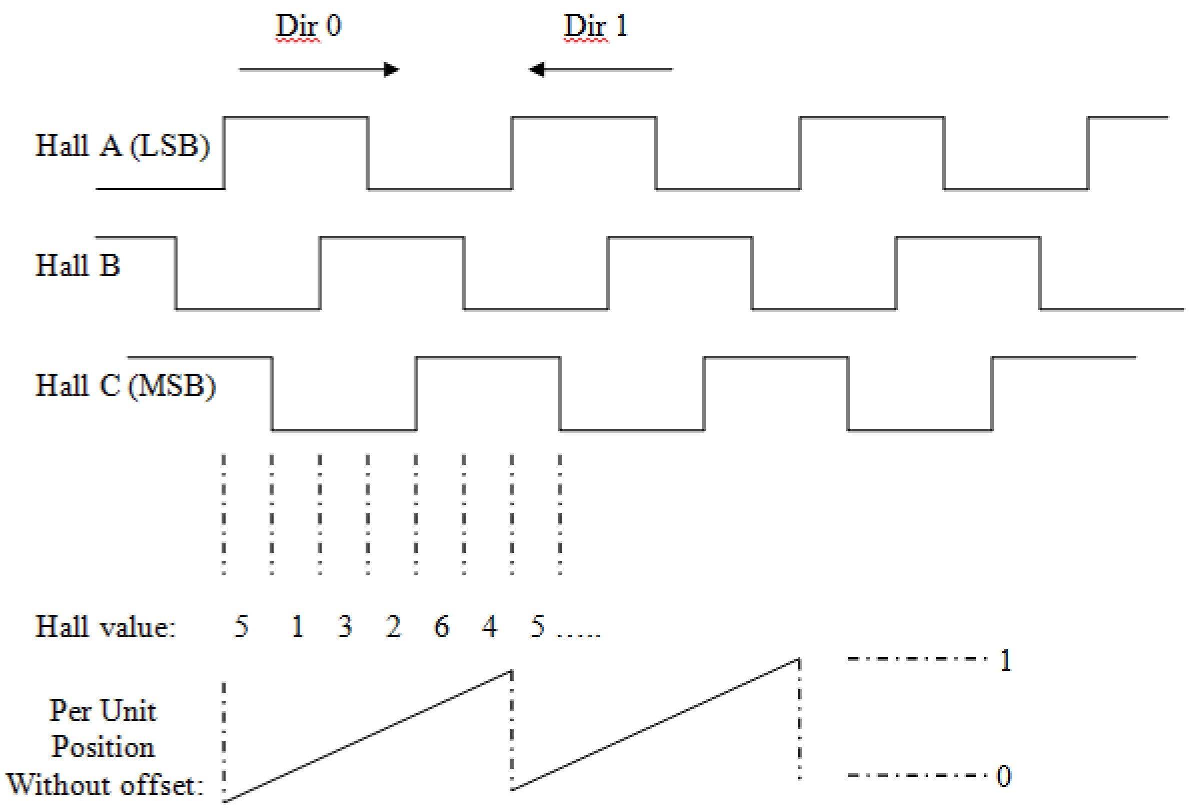


Model Calibration Using DRV8312EVM, Dual-Axis DMC EVM, and High Voltage Motor Control Kit

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, J4 of the Dual-Axis DMC EVM or H1 for the High Voltage Kit. 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.

Figure 5: 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:

- New_direction = Hall_direction
- New_valid_flag = Previous_direction == Hall_direction;
- 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:

- Read the halls.
- Get the value of the latest timer (timer captured from the last interrupt that triggered).
- Convert the time that elapsed from the previous edge to an electrical angle using the current speed.
- 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 and Dual-Axis DMC EVM kits.

For the Dual-Axis DMC EVM kit, c28035pmsmfocdual.slx controls 2 motors from the same F28035 processor. Motor2 can use position sensing coming from Hall sensors attached to J4 of the Dual-Axis DMC EVM.

Fast Serial Data Monitoring

Evaluation boards often provide serial over USB connections allowing fast serial transfers. The models running on the LaunchPads, are sending Ia and Ib currents over serial. Use c2000_host_read.slx to receive these signals on your host computer. For F28379D Launchpad example use c2000_host_read_F28379D.slx Select the appropriate COM port matching your board in the following blocks:

- c2000_host_read/Serial Configuration or c2000_host_read_F28379D/Serial Configuration
- c2000_host_read/Serial Receive or c2000_host_read_F28379D/Serial Receive
- c2000_host_read/Enabled Subsystem/Serial Send or c2000_host_read_F28379D/Enabled Subsystem/Serial Send

Adjust the Baud rate for your board:

- For F28027 Launchpad, set the baud rate to 3.75e6
- For F28069 Launchpad, set the baud rate to 5.625e6
- For F28377S Launchpad, set the baud rate to 5e6
- For F28379D Launchpad, set the baud rate to 5e6

Run the c2000_host_read model and observe the current signals updated at 20 kHz on the time scope. You can use the same technique to monitor other signals on other processors. Keep in mind that SCI_A is usually connected to the FTDI chip allowing serial transfers over USB on Launchpads, docking stations and ISO control cards. The FTDI FT2232D allows one way transfers up to 1.5 Mbps while the FTDI FT2232H allows one way transfers up to 6 Mbps.

Using the Control Law Accelerator (CLA)

This example shows how to use the Control Law Accelerator (CLA) available on some of the TI Piccolo processors.

Required Hardware:

For the LED blinking model:

- TI® Piccolo F28069, F28035, F28004x or Delfino F28377S board.

For the motor control application model:

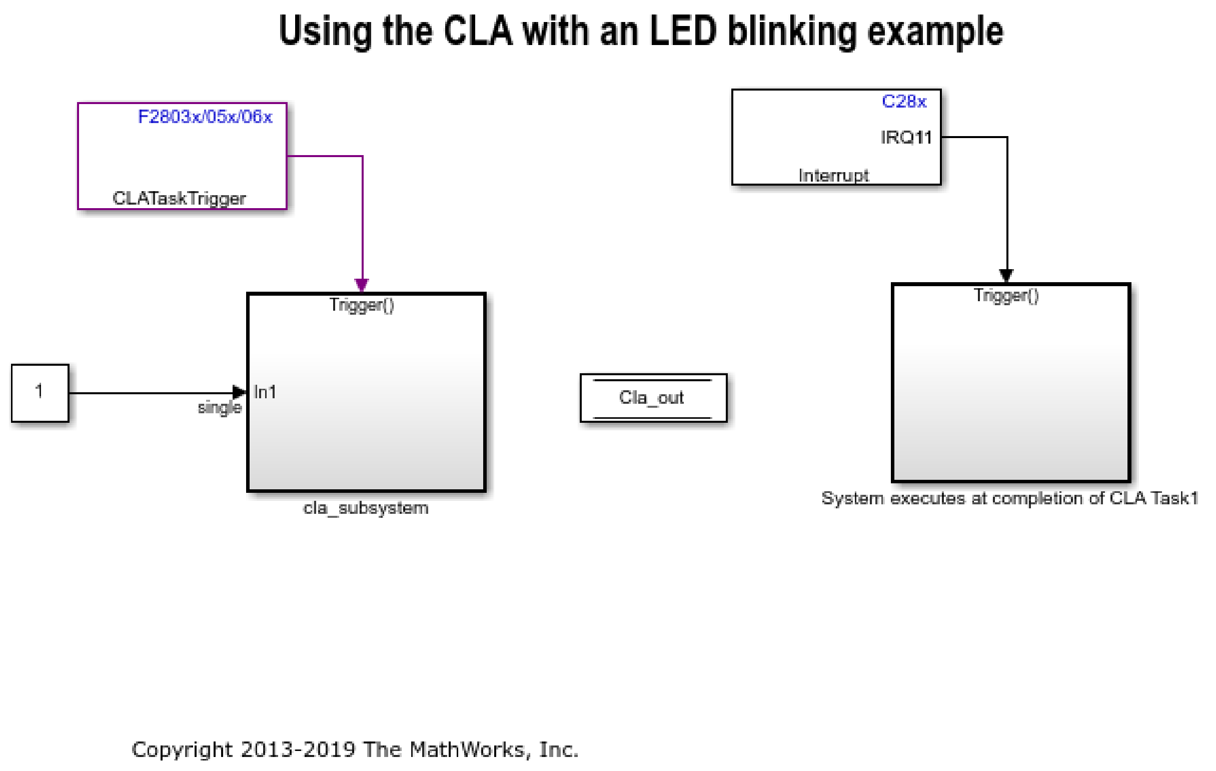
- TI® DRV8312 Three-Phase Brushless Motor Control Kit (DRV8312-C2-KIT or DRV8312-69M-KIT) with F28035 or F28069 Piccolo processor
- Three-phase PMSM with Hall sensors attached to connector J10 of the DRV8312EVM board

Available versions of example models:

- F28035 Piccolo: c28035blink_cla.slx
- F28069 Piccolo: c28069blink_cla.slx
- F28377S Delfino: c28377Sblink_cla.slx
- F28004x Piccolo: c28004xblink_cla.slx
- F28069 Piccolo: c28069_dataintegrity_cla.slx
- F28379D Delfino: c28379D_dataintegrity_cla.slx
- F28379D Delfino Launchpad: c28379D_cpu1_blink_cla.slx, c28379D_cpu2_blink_cla.slx
- DRV8312 + F28069 Piccolo: c28069pmsmfoc_cla.slx
- DRV8312 + F28035 Piccolo: 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.

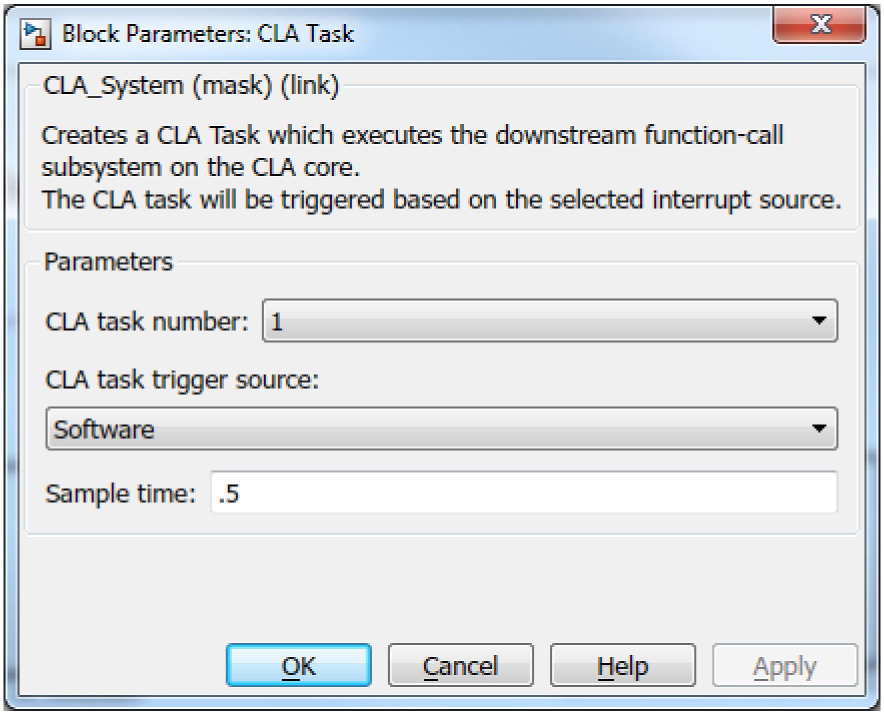

Using the CLA with an LED blinking example
F2803x/05x/06x
CLATaskTrigger
C28x
IRQ11
Interrupt
Trigger()
1
single
In1
cla_subsystem
Cla_out
Trigger()
System executes at completion of CLA Task1
Copyright 2013-2019 The MathWorks, Inc.

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.

The following section shows how to trigger a CLA task, exchange data from the C28x CPU to the CLA using memory sections like CpuToCla1MsgRAM and Cla1ToCpuMsgRAM, schedule C28x interrupts at the end of the CLA task, and replace the standard linker command file to include CLA specific linking attributes.

CLA Task Block

In the c28069blink_cla.slx model, a CLA Task block is provided to trigger a CLA task. This block runs the downstream function-call subsystem on the CLA. On the CLA Task block mask, you can specify the CLA task number and the associated interrupt triggering source. Ensure that you are using the CLA Trigger block from the library matching the 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. In this 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

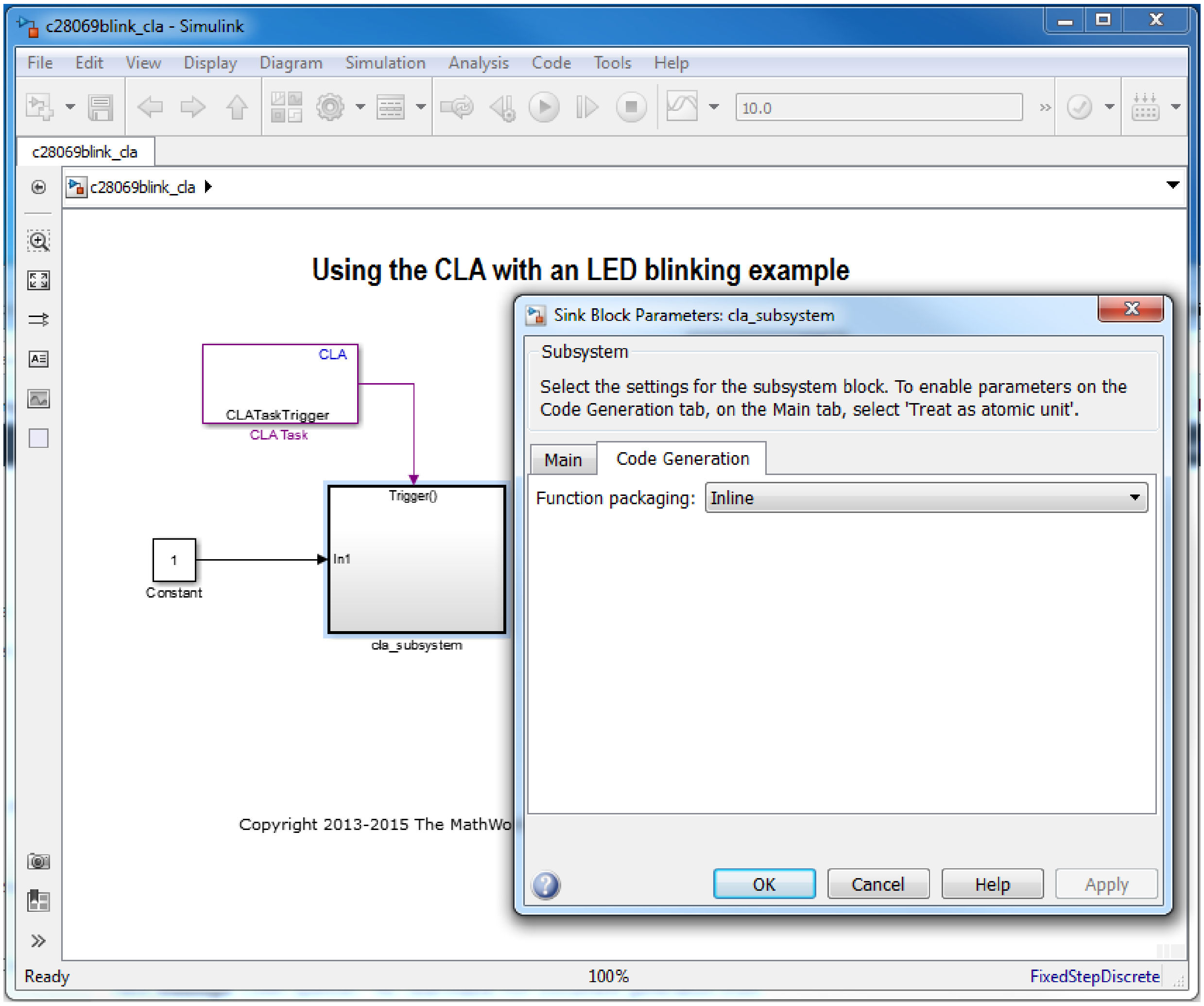
CLA task trigger source:

Software

Sample time: .5

Select Inline Code Generation for CLA Subsystem

For the CLA task subsystem, 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)".


c28069blink_cla - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
c28069blink_cla
Using the CLA with an LED blinking example
CLA
CLATaskTrigger
CLA Task
Trigger()
1
Constant
In1
cla_subsystem
Copyright 2013-2015 The MathWo
Ready
100%
FixedStepDiscrete

Sink Block Parameters: cla_subsystem

Subsystem

Select the settings for the subsystem block. To enable parameters on the Code Generation tab, on the Main tab, select 'Treat as atomic unit'.

Main | Code Generation

Function packaging: Inline

OK Cancel Help Apply

Changing the Standard Linker Command File

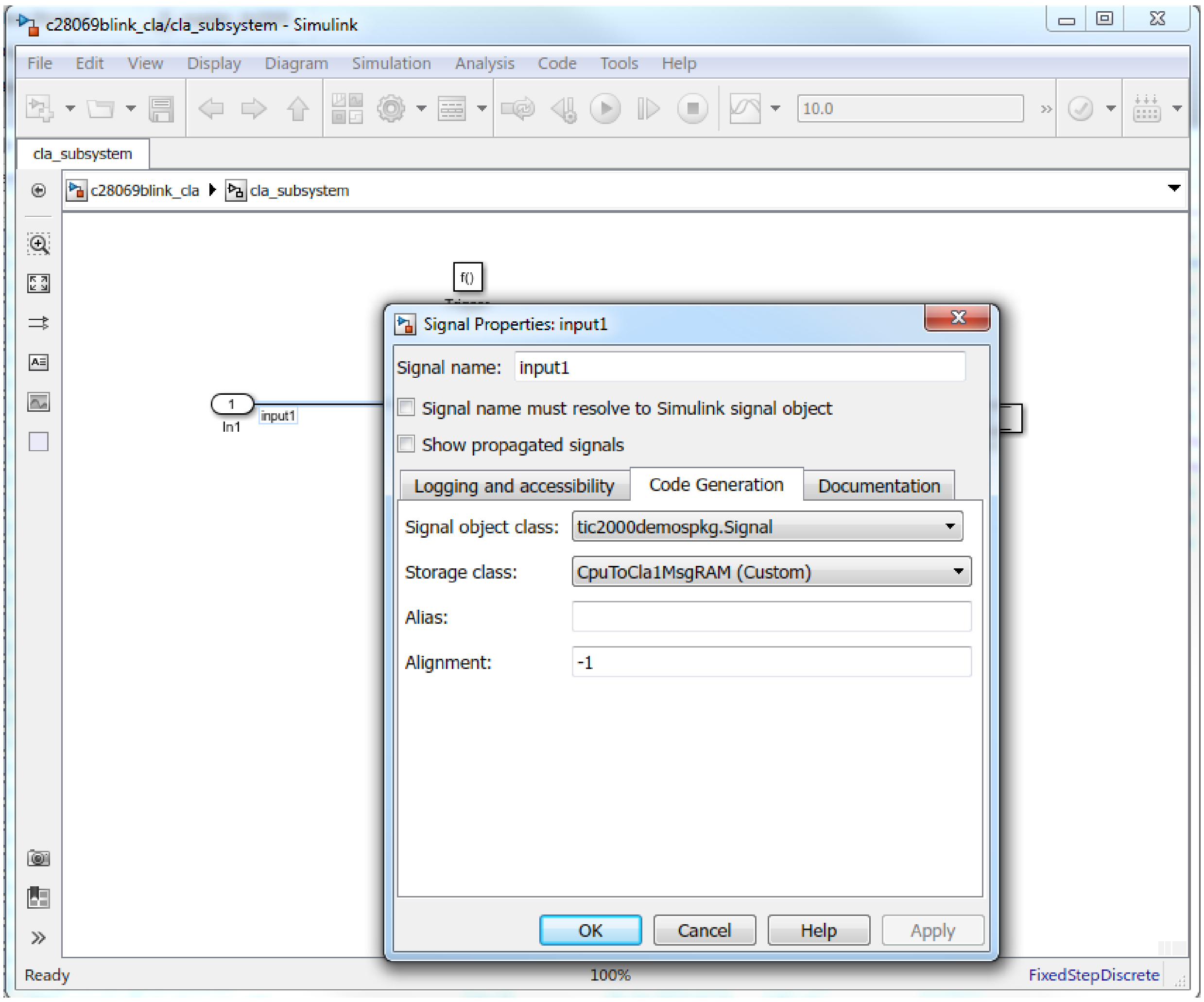
For F28035 and F28069 processors, a specific linker command file has to be selected to assign CLA memory sections. In the Model Configuration Parameters, under Hardware Implementation > Target hardware resources > Build options, "Use custom linker command file" is selected and the pre-configured "c28069_cla.cmd" 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 support package installation.

Interrupt Generation after CLA Task Completion

A CLA interrupt is generated when the CLA Task completes. CLA Task1 interrupt (CPU = 11, PIE = 1-8) is used as the interrupt source on the C28x Hardware Interrupt block. The function-call subsystem 'System executes at completion of CLA Task1' is executed when the interrupt occurs. 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.

Data Exchange between the CLA and the C28x CPU

All the interfaces between CLA and CPU need to be placed in specific memory sections. To gain specific access to these sections, custom storage classes CpuToCla1MsgRAM and Cla1ToCpuMsgRAM are defined in the dataclass package tic2000demospkg.


c28069blink_cla/cla_subsystem - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
cla_subsystem
c28069blink_cla
cla_subsystem
f()
1
In1
input1
Ready
100%
FixedStepDiscrete

Signal Properties: input1

Signal name: input1

☐ Signal name must resolve to Simulink signal object

☐ Show propagated signals

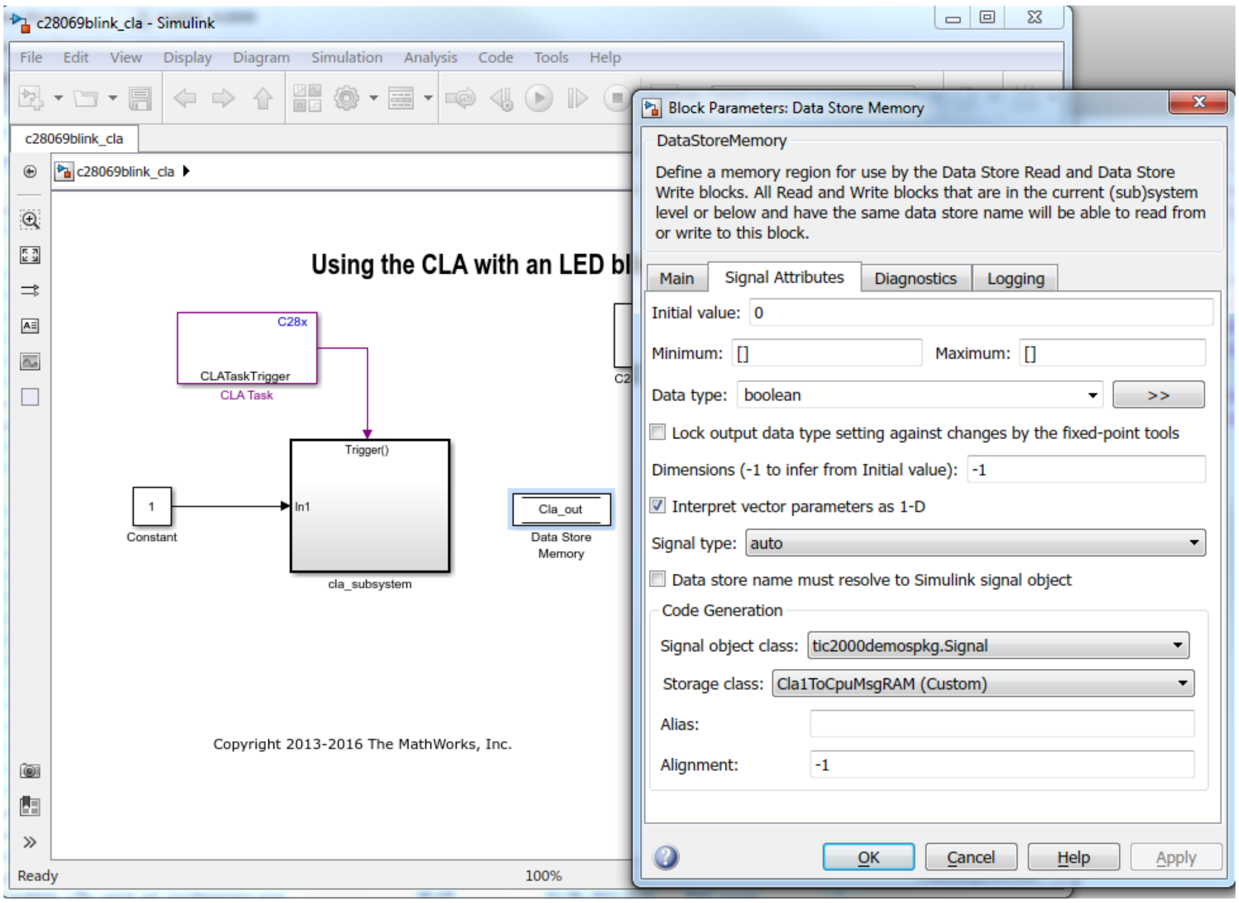
Logging and accessibility | Code Generation | Documentation

Signal object class: tic2000demospkg.Signal

Storage class: CpuToCla1MsgRAM (Custom)

Alias:

Alignment: -1


c28069blink_cla - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
c28069blink_cla
Using the CLA with an LED bl
C28x
CLATaskTrigger
CLA Task
Trigger()
1
Constant
In1
cla_subsystem
Cla_out
Data Store Memory
Copyright 2013-2016 The MathWorks, Inc.
Ready
100%
Block Parameters: Data Store Memory
Main
Signal Attributes
Diagnostics
Logging
OK
Cancel
Help
Apply

DataStoreMemory

Define a memory region for use by the Data Store Read and Data Store Write blocks. All Read and Write blocks that are in the current (sub)system level or below and have the same data store name will be able to read from or write to this block.

Initial value: 0

Minimum: [] Maximum: []

Data type: boolean

☐ Lock output data type setting against changes by the fixed-point tools

Dimensions (-1 to infer from Initial value): -1

☑ Interpret vector parameters as 1-D

Signal type: auto

☐ Data store name must resolve to Simulink signal object

Code Generation

Signal object class: tic2000demospkg.Signal

Storage class: Cla1ToCpuMsgRAM (Custom)

Alias:

Alignment: -1

Model Configuration Required while Using the CLA

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

Discrete State Variables

All discrete state variables used inside CLA function-call subsystems must be stored in 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 for an example.

Other Guidelines to Consider while Using the CLA

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

The specificity of the CLA may require more operations on the model or may prevent the use of Simulink features. Make sure to go through the list of limitations provided in the CLA C compiler documentation. Make sure to use the features of Embedded Coder to restrict the generated code in the limited syntax supported by the CLA C compiler.

Run the Model

1. Open the example model, c28069blink_cla.slx, c28035blink_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.

Task 2: Debug Code on the 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.
3. For debug instructions visit: http://processors.wiki.ti.com/index.php/C2000_CLA_C_Compiler#Debugging

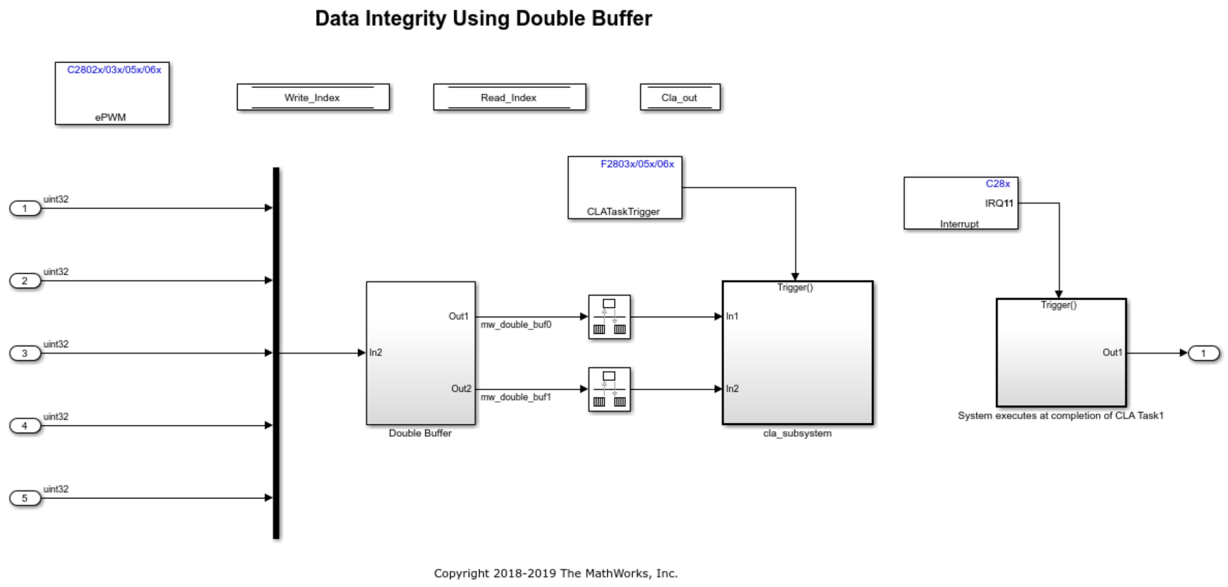
Task 3: Ensure Data Integrity Between CPU and CLA

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.



In the double buffer algorithm, two buffers 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.

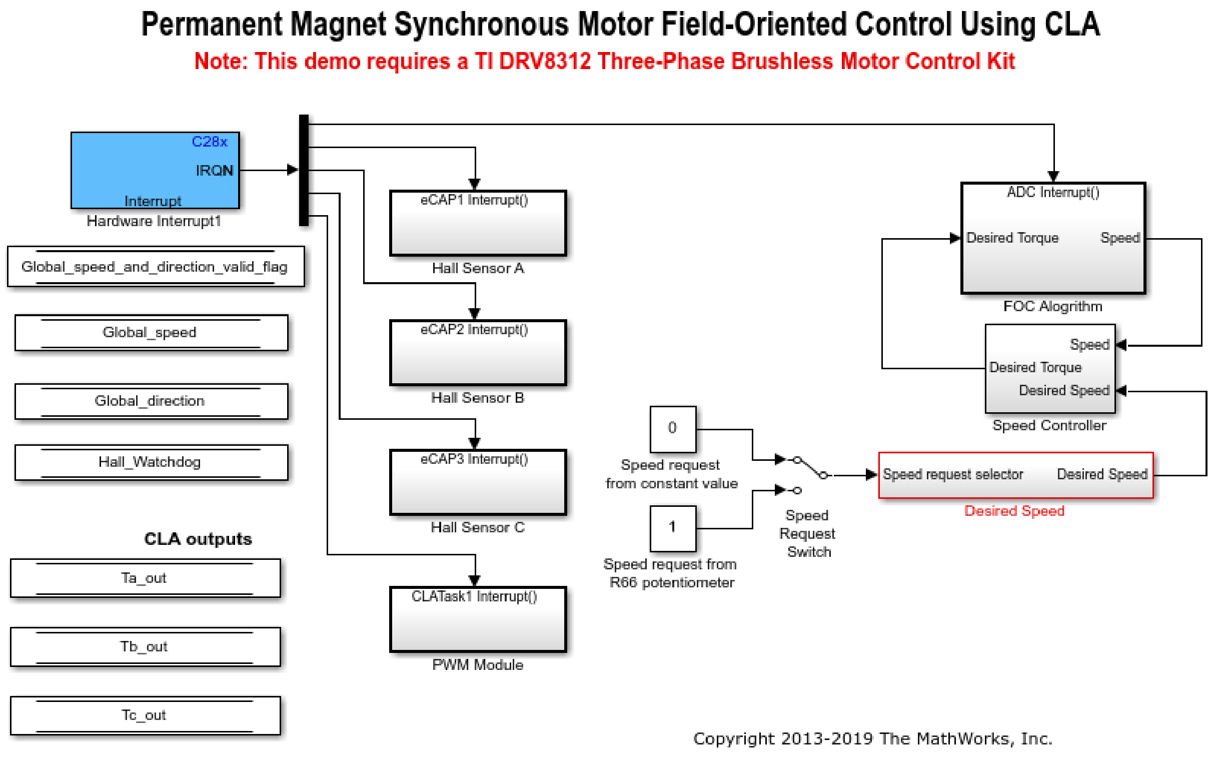
By using the double buffer logic, data integrity is ensured even though the CPU and the CLA are not synchronous.

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 4: Permanent Magnet Synchronous Motor Field-Oriented Control (FOC) Using CLA

The following figure shows a Permanent Magnet Synchronous Motor Field-Oriented Control (FOC) example model, which runs the FOC algorithm on the 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. This example is an extension of “Permanent Magnet Synchronous Motor Field-Oriented Control”, with the usage of CLA. 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.

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.

Limitations

Specific interactions between the CLA and the C28x CPU along with CLA C compiler limitations force the modeling practices to be followed. Below is a list of concepts that reflects modeling practices described in this example:

- The CLA application code can only be triggered by a C28x event. CLA Task can be triggered by C28x CPU via software or by different peripheral interrupts.
- All the interfaces between CLA and CPU need to be placed in specific memory locations. The CpuToCla1MsgRAM memory section is used to exchange data from the C28x to the CLA. The Cla1ToCpuMsgRAM memory section is used to exchange data from the CLA to the C28x.
- The CLA application code does not have access to global variables.
- Early versions of the CLA C compiler support only 2 levels of function calls. CLA interrupt service routines may call leaf functions only. Leaf functions may not call other functions.
- Recursive function calls are not supported.
- Integer division, modulus, and integer unsigned comparison are not supported with the CLA C compiler.

For more details and an exhaustive list of limitations visit: http://processors.wiki.ti.com/index.php/C2000_CLA_C_Compiler

These limitations must be considered when modeling in Simulink.

Troubleshooting

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

More About

- C28x CLA Task

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

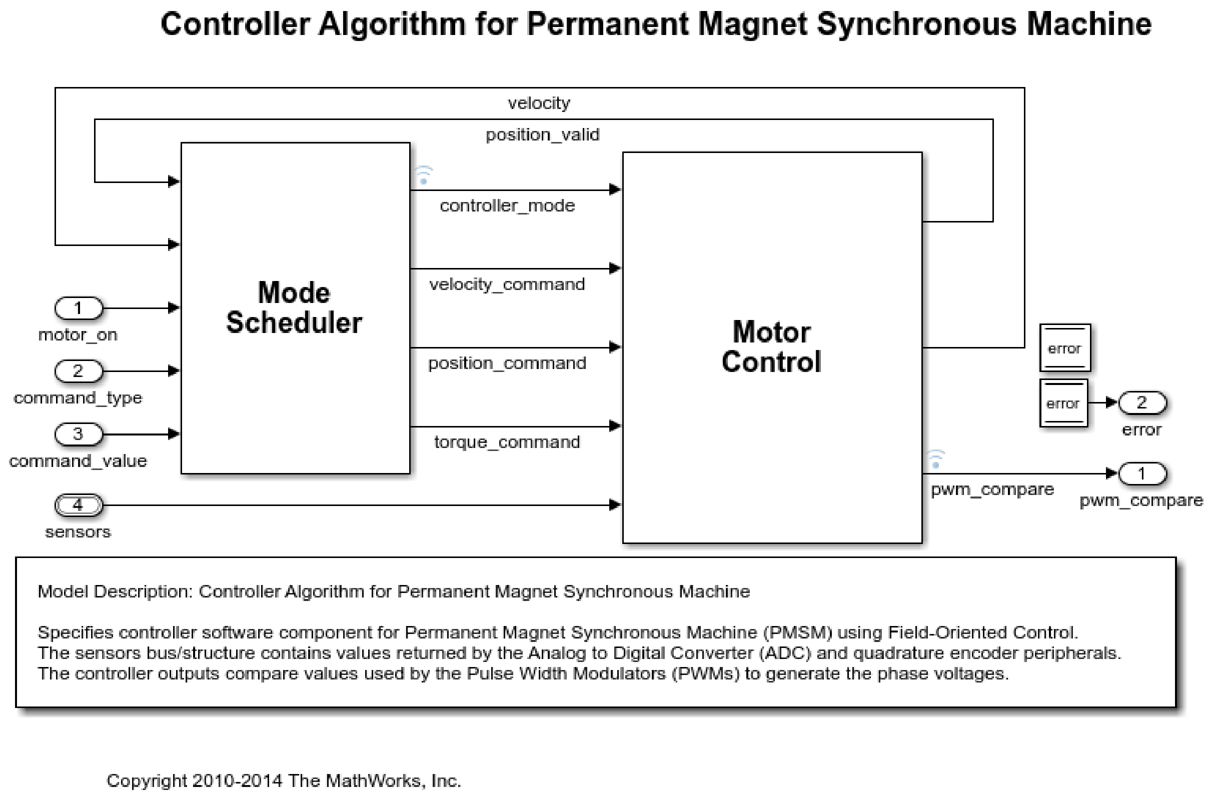
Note: Match the characteristics of the power supply and the amplifier with the input characteristics of the selected motor.

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

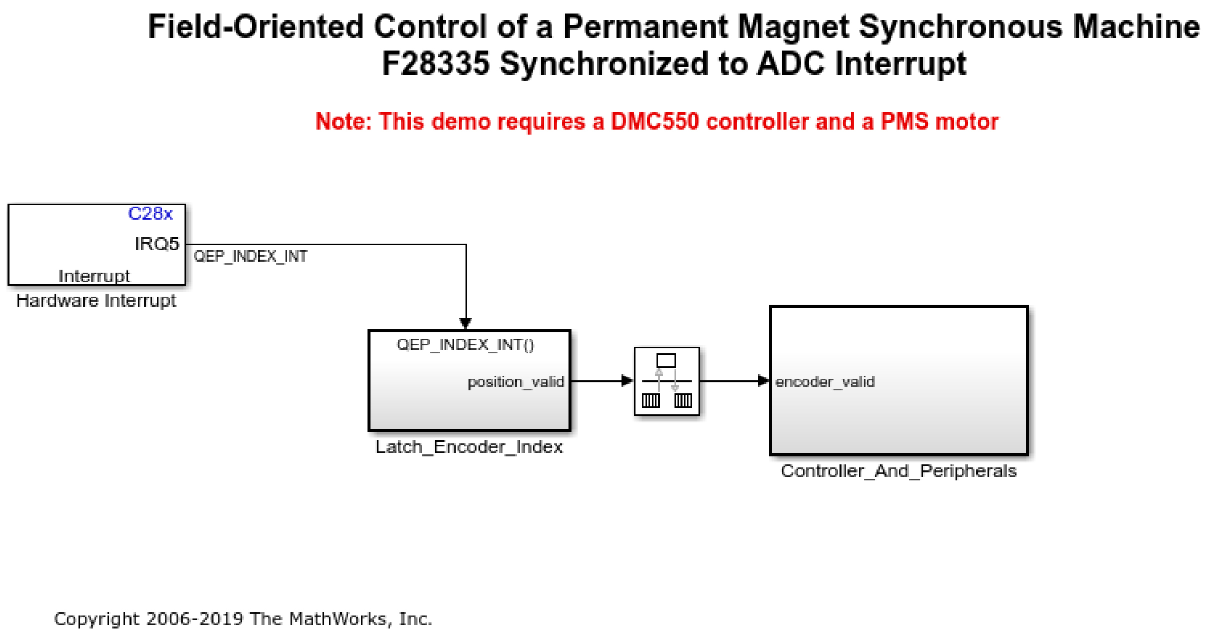


Model Description: Controller Algorithm for Permanent Magnet Synchronous Machine

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.

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 2006-2019 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

Code Verification and Validation with PIL

This example shows you how to use Embedded Coder Support Package for Texas Instruments C2000 processor for code verification and validation using PIL.

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 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 Embedded Coder Support Package for Texas Instruments C2000 supports serial communication interface for PIL over SCI-A.

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. Enable the settings for running PIL on Serial

- Set the COM Port as obtained above and replace the 'COM1' in the following command with the correct Serial port corresponding to your controlCARD:

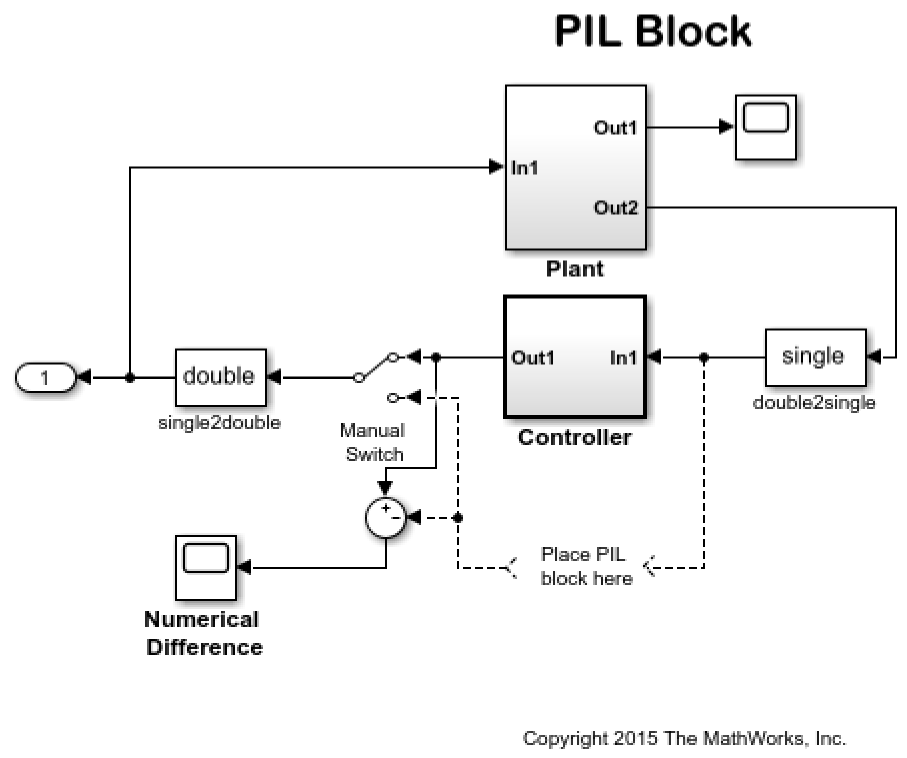
setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','COMPort','COM1');

- Set the baud rate for PIL communication by entering the baud rate as

setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','BaudRate',115200);

- Enable PIL over serial by the following command:

setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','enableserial',true);



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.

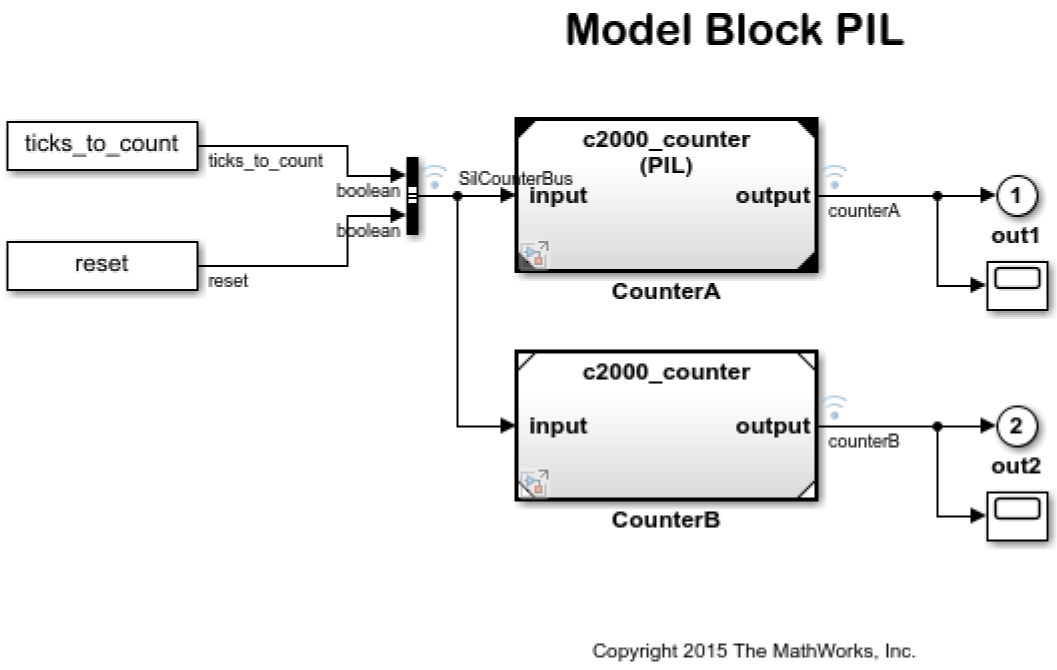
2. Choose a PIL communication interface by following the steps in Task 1 above.

3. Create a PIL block for the **Controller** subsystem as explained in Verify Generated Code Using Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) Simulation.

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 Verify Generated Code Using Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) Simulation.

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:

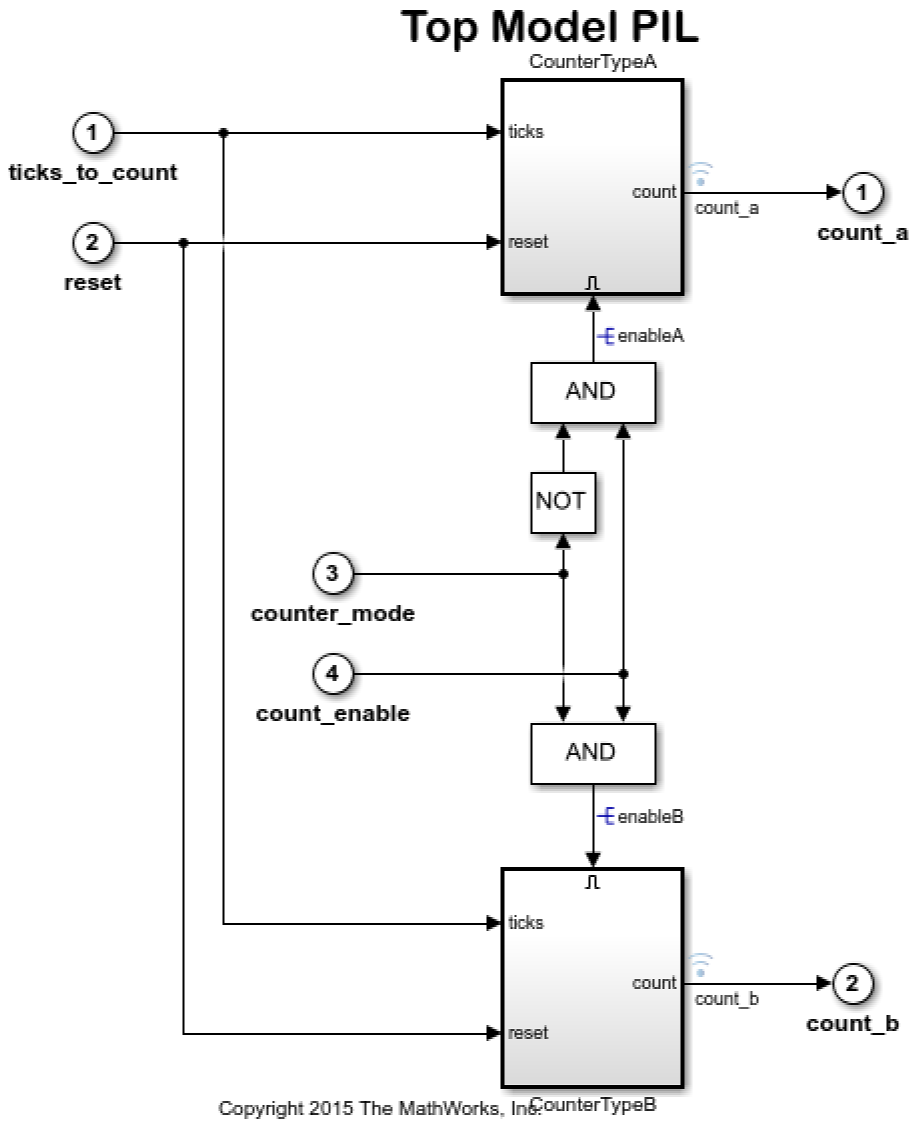
- You can verify code generated for referenced models
- You can easily switch a Model block between normal and PIL simulation mode

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.

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 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 TI Piccolo F2806x processor while **Scope2** shows the normal mode simulation output.



Top Model PIL
CounterTypeA
1
ticks_to_count
2
reset
ticks
reset
count
count_a
1
count_a
enableA
AND
NOT
3
counter_mode
4
count_enable
AND
enableB
ticks
reset
count
count_b
2
count_b
Copyright 2015 The MathWorks, Inc.
CounterTypeB

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.

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 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

Summary

This example introduced code verification workflows using PIL

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. See Embedded Coder Support Package for Texas Instruments C2000 Processors for details on workflows for Texas Instruments C2000 Processors.

Required Hardware

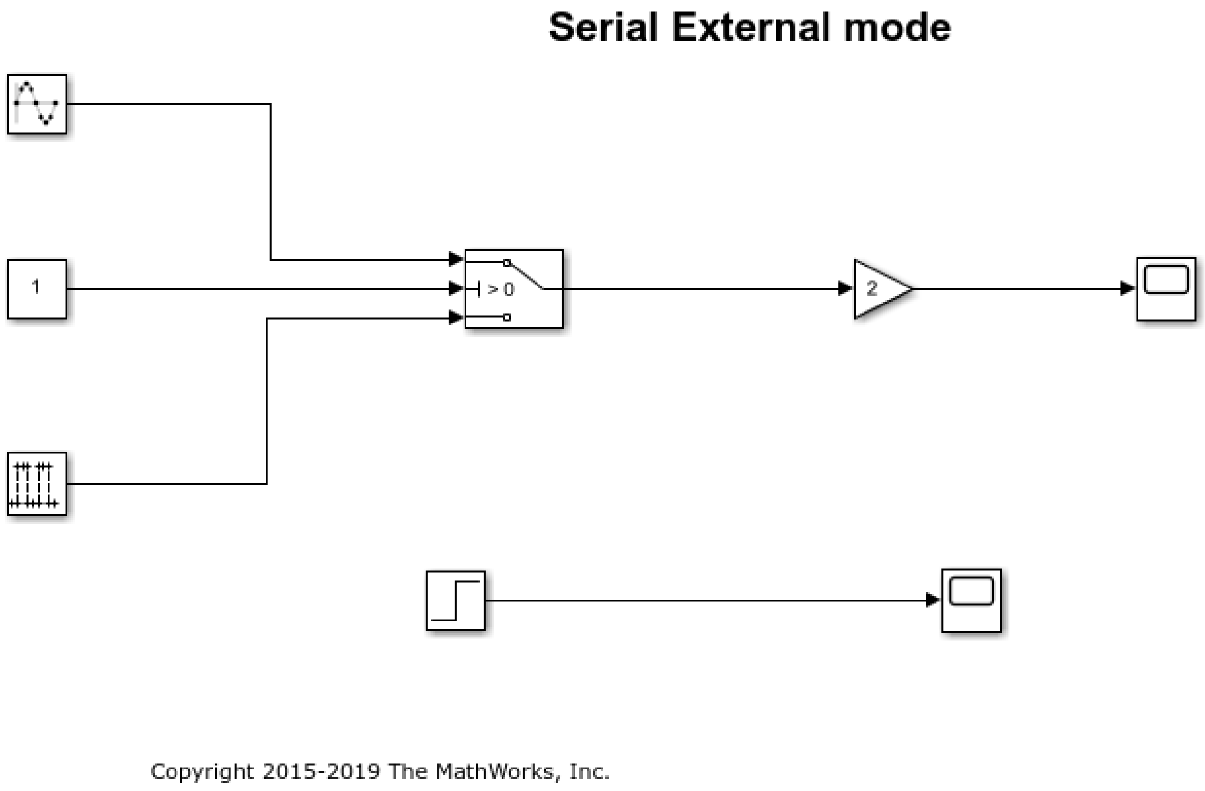
- Any of the Texas Instruments C2000 - based controlCARDS with docking station or Spectrum Digital eZdsp boards that have a serial interface with SCI_A.

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.

Model

This figure shows the example model.



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 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:

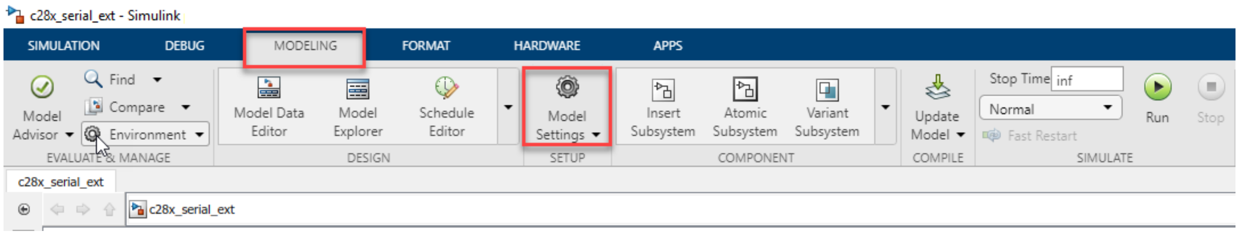
- Open Device Manager in Windows.
- Expand the Ports tab.
- Note the COM port associated with the target board.

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**. See also Parameter Tuning over Serial Communication for Texas Instruments C2000 Processors.

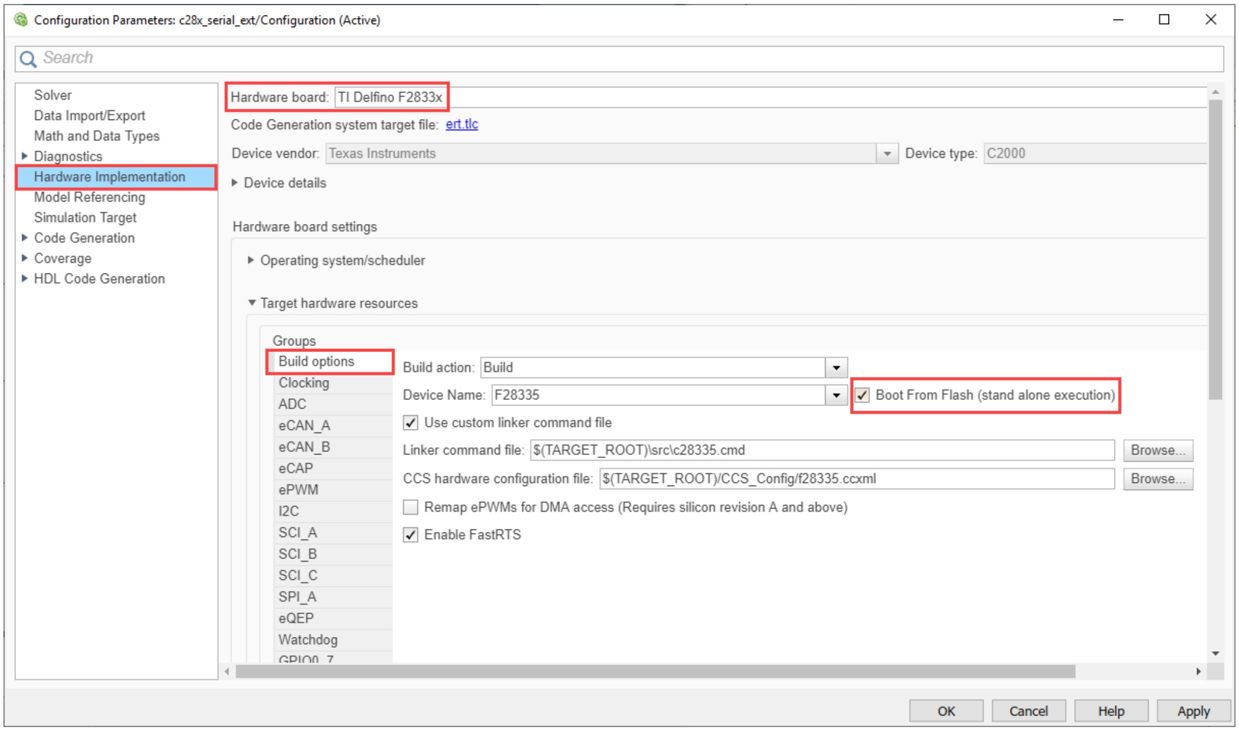
1. Open the Example Model.

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

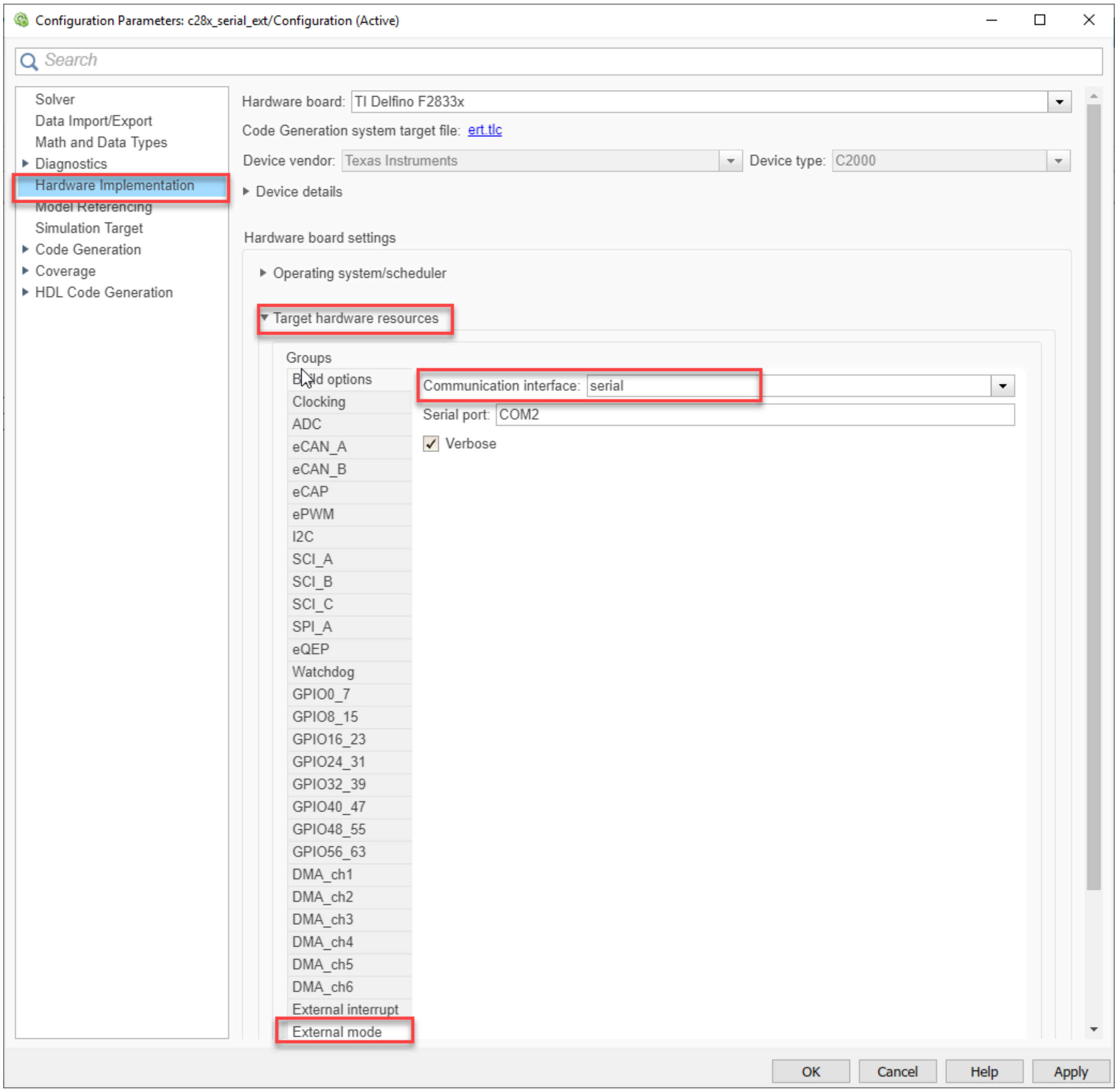
CCS hardware configuration file: $(TARGET_ROOT)/CCS_Config/f28335.ccxml

☐ Remap ePWMs for DMA access (Requires silicon revision A and above)

✓ Enable FastRTS

4. Navigate to the **Hardware Implementation > Target Hardware Resources > External mode**. Specify the **Communication interface** as **serial**. Specify the **Serial port** 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

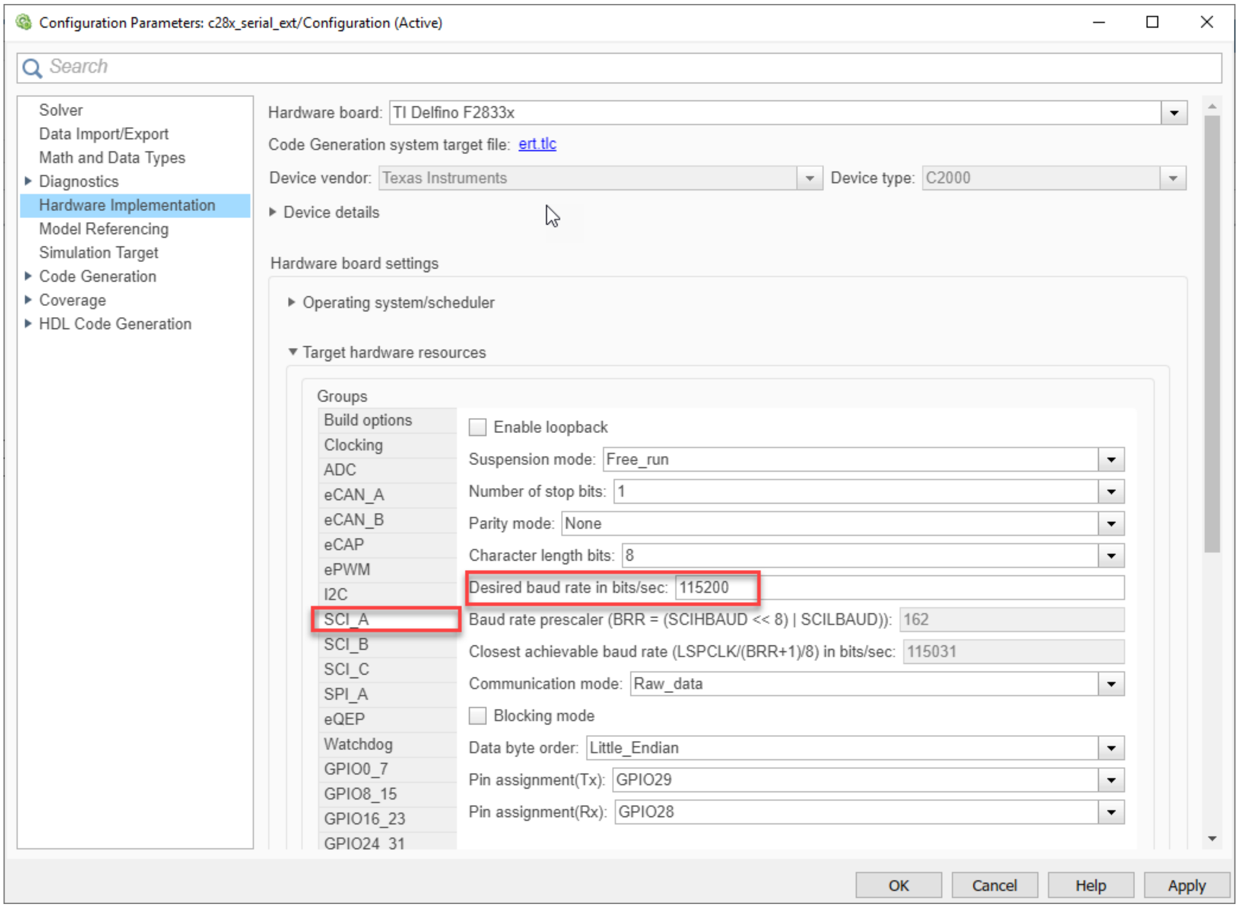
Device vendor: Texas Instruments

Device type: C2000

Communication interface: serial

Serial port: COM2

✓ Verbose



Configuration Parameters: c28x_serial_ext/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 Delfino F2833x

Code Generation system target file: ert.tlc

Device vendor: Texas Instruments   Device type: C2000

Device details

Hardware board settings

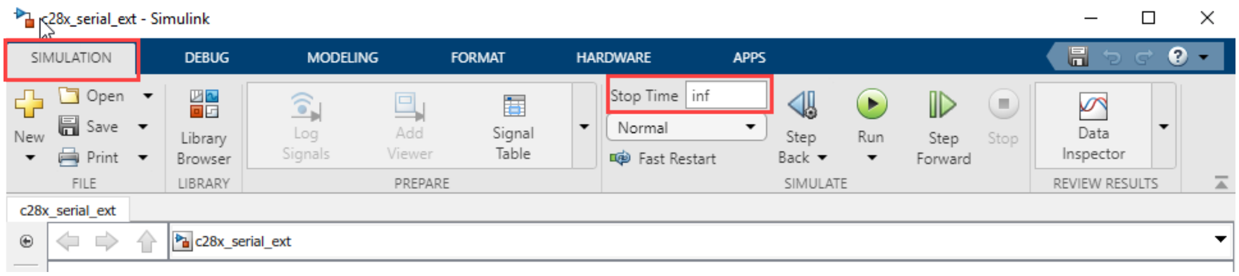
Operating system/scheduler

Target hardware resources

Groups: Build options, Clocking, ADC, eCAN_A, eCAN_B, eCAP, ePWM, I2C, SCI_A, SCI_B, SCI_C, SPI_A, eQEP, Watchdog, GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31

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

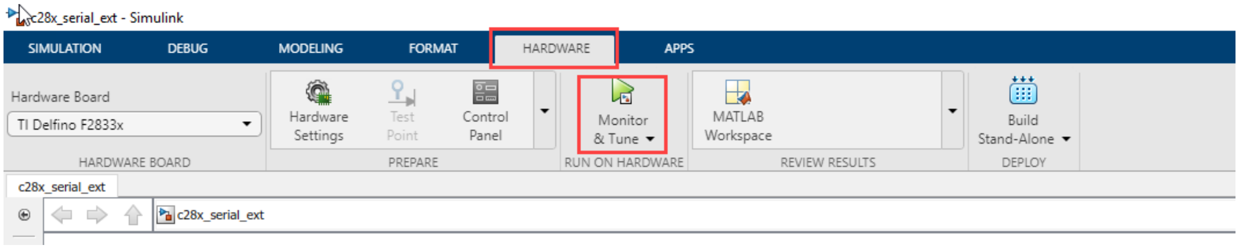
5. Open the **Simulation** tab and specify a value in the **Stop Time** text box. You can specify the stop time as 'inf' to run the model continuously on the target hardware.



Monitor and Tune the Model

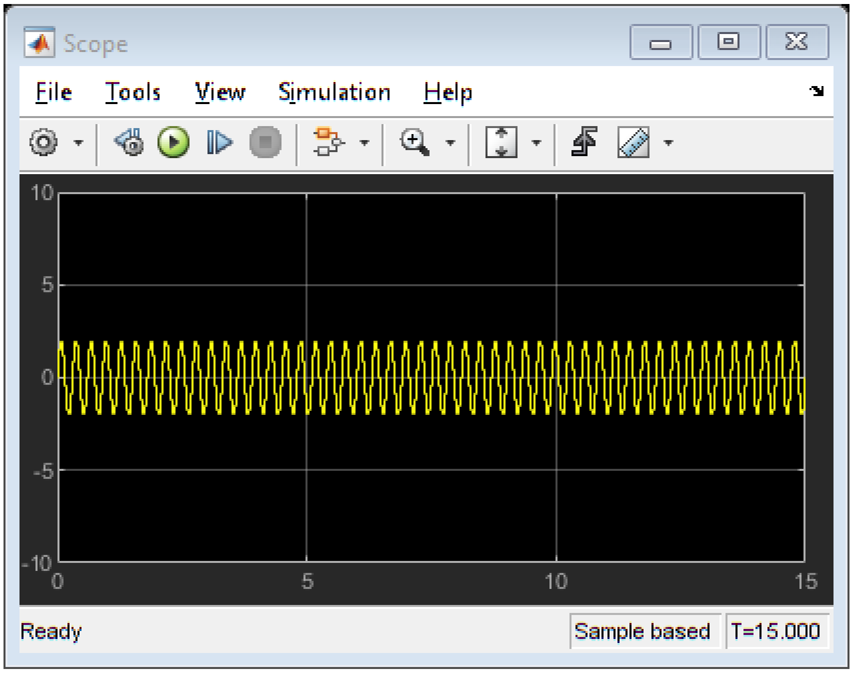
When you perform Monitor and Tune action for the model, the host computer communicates with the target, on which the generated executable runs. To perform Monitor and Tune in the Example Model:

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.

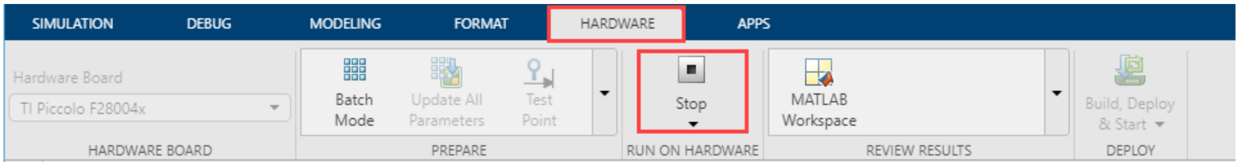


2. While the model is running, open the **Scope** connected to the **Gain** block to monitor its output.

```
Warning: Unrecognized function or variable
'codertarget.vexarmcortex.internal.getInstallDir'.
Warning: Unrecognized function or variable
'codertarget.vexarmcortex.internal.getInstallDir'.
Warning: Unrecognized function or variable
'codertarget.vexarmcortex.internal.getInstallDir'.
```



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 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.

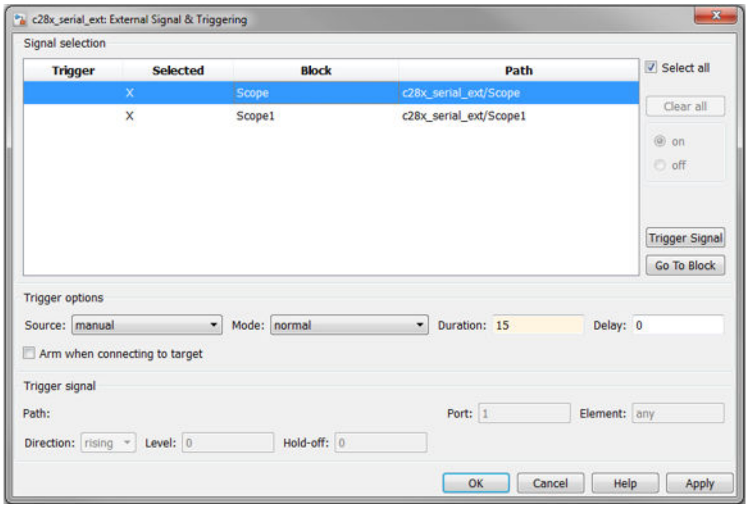
Logging Signals with a Manual Trigger

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.



c28x_serial_ext: External Signal & Triggering

Signal selection

Trigger	Selected	Block	Path
	X	Scope	c28x_serial_ext/Scope
	X	Scope1	c28x_serial_ext/Scope1

☑ Select all

Trigger options

Source: manual Mode: normal Duration: 15 Delay: 0

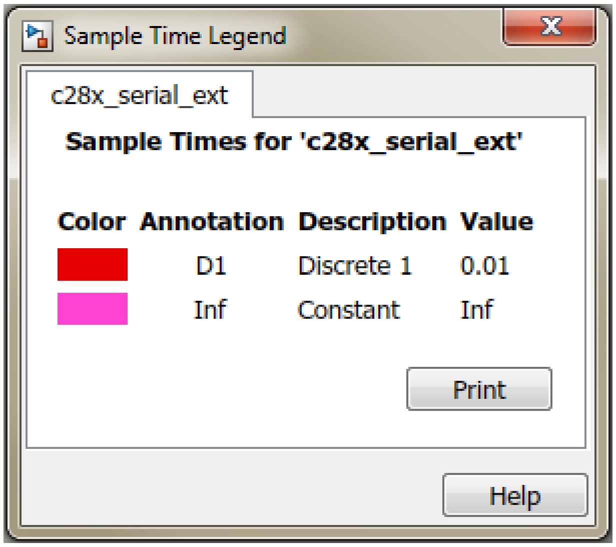
☐ Arm when connecting to target

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 Time Legend

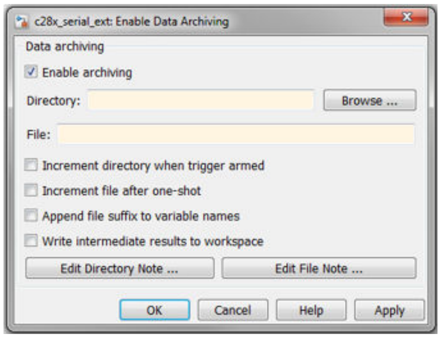
c28x_serial_ext

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:

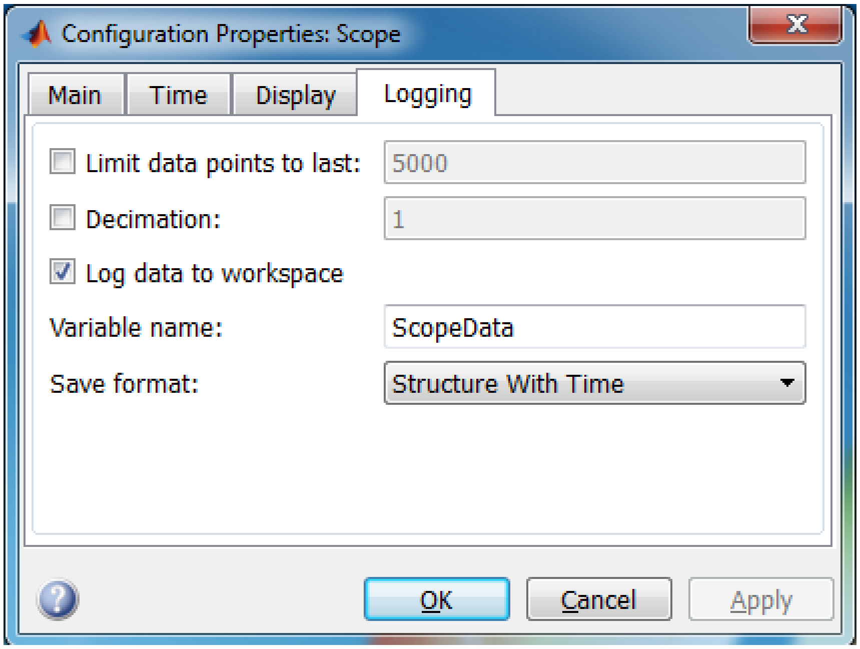
☐ Increment directory when trigger armed

☐ 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

Main | Time | Display | Logging

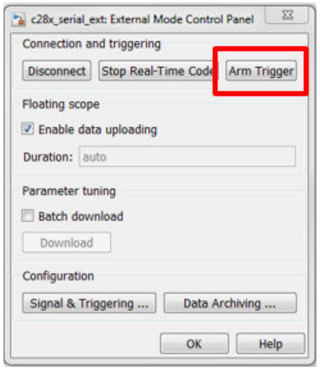
- [ ] Limit data points to last: 5000
- [ ] Decimation: 1
- [x] Log data to workspace

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.


c28x_serial_ext: External Mode Control Panel

Connection and triggering

Floating scope

- [x] Enable data uploading

Duration: auto

Parameter tuning

- [ ] Batch download

Configuration

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.

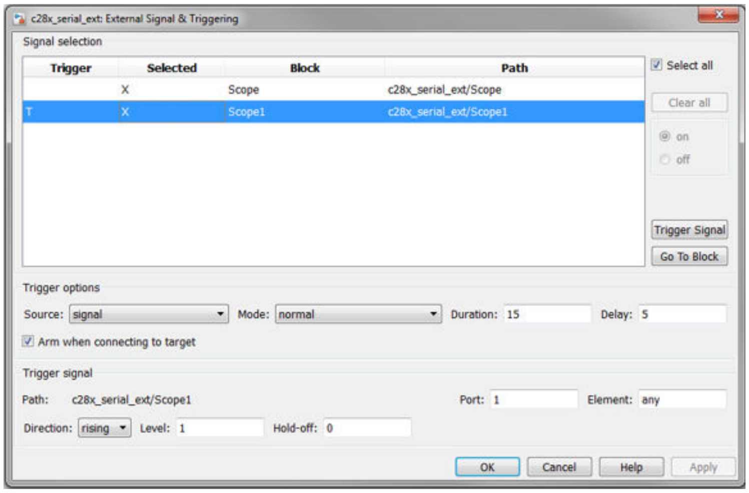
2. In this dialog box, select the Scope block connected to the **Step** input and click the **Trigger Signal** button to set the selected block signal as the trigger signal.

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.



Signal selection

Trigger	Selected	Block	Path
	X	Scope	c28x_serial_ext/Scope
T	X	Scope1	c28x_serial_ext/Scope1

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.

Limitations

- Increasing the **Duration** allows the capture of large real-time buffers. These buffers use heap section to allocate memory. A high value of **Duration** may result in *Not enough memory on the target to process the packet* warning, and consequently cause no data to be uploaded from the target to the host computer. To fix this problem, specify a larger heap size if your target has sufficient memory. To change the heap size, browse to **Configuration Parameters > Code Generation > Build Configuration > Specify > Linker** and modify the value of **--heap_size**.
- Parameter tuning and signal logging for 8-bit data types is not supported over serial communication interface for Texas Instruments C2000 processors. See Data Type Support for details on the data type support for Texas Instruments C2000 processors.

Troubleshooting

- An error stating *Attempting to establish connection with hostname 127.0.0.1 through port ...* indicates that either the COM port or the TCP/IP port 17725 on the local loopback are held by some other application. Ensure that the ports are available before you start Monitor and Tune, which uses a process in the background to buffer data and improve data logging. This process needs port 17725.
- 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

"Parameter Tuning and Signal Logging over Serial Communication" on page 1-48

Real-Time Code Execution Profiling

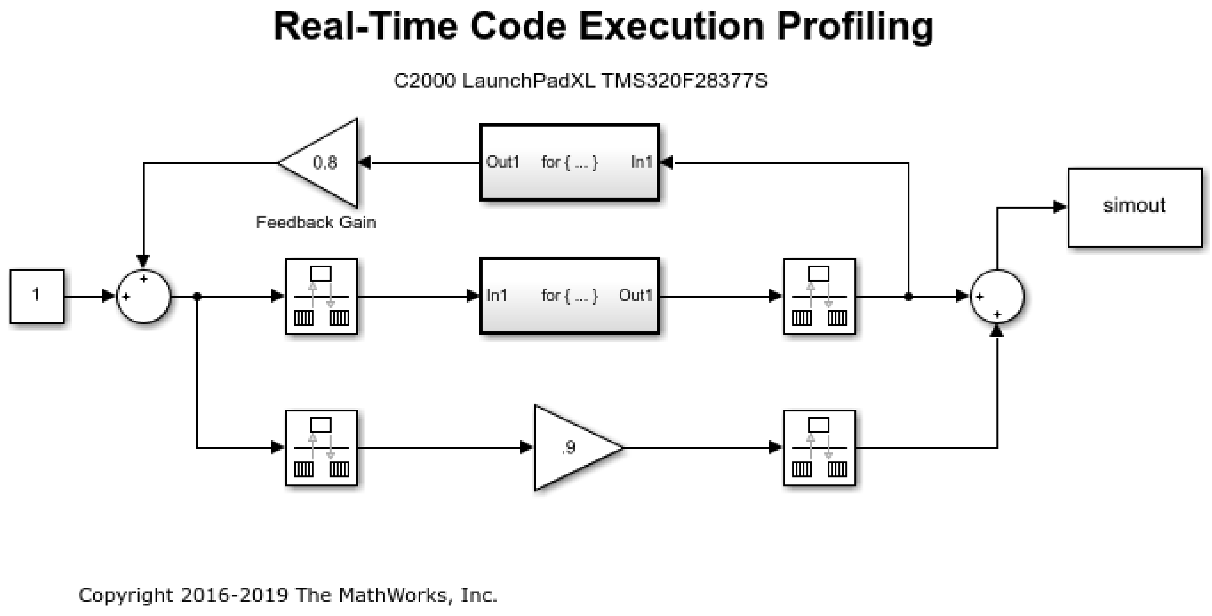
This example shows you how to use Embedded Coder™ Support Package for Texas Instruments C2000 Processors and Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors for real-time execution profiling of generated code.

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
Out1 for { ... } In1
In1 for { ... } Out1
1
.9
simout
Copyright 2016-2019 The MathWorks, Inc.

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 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. Open the **Hardware** tab and click **Build, Deploy & Start** to run the code inside the target hardware.

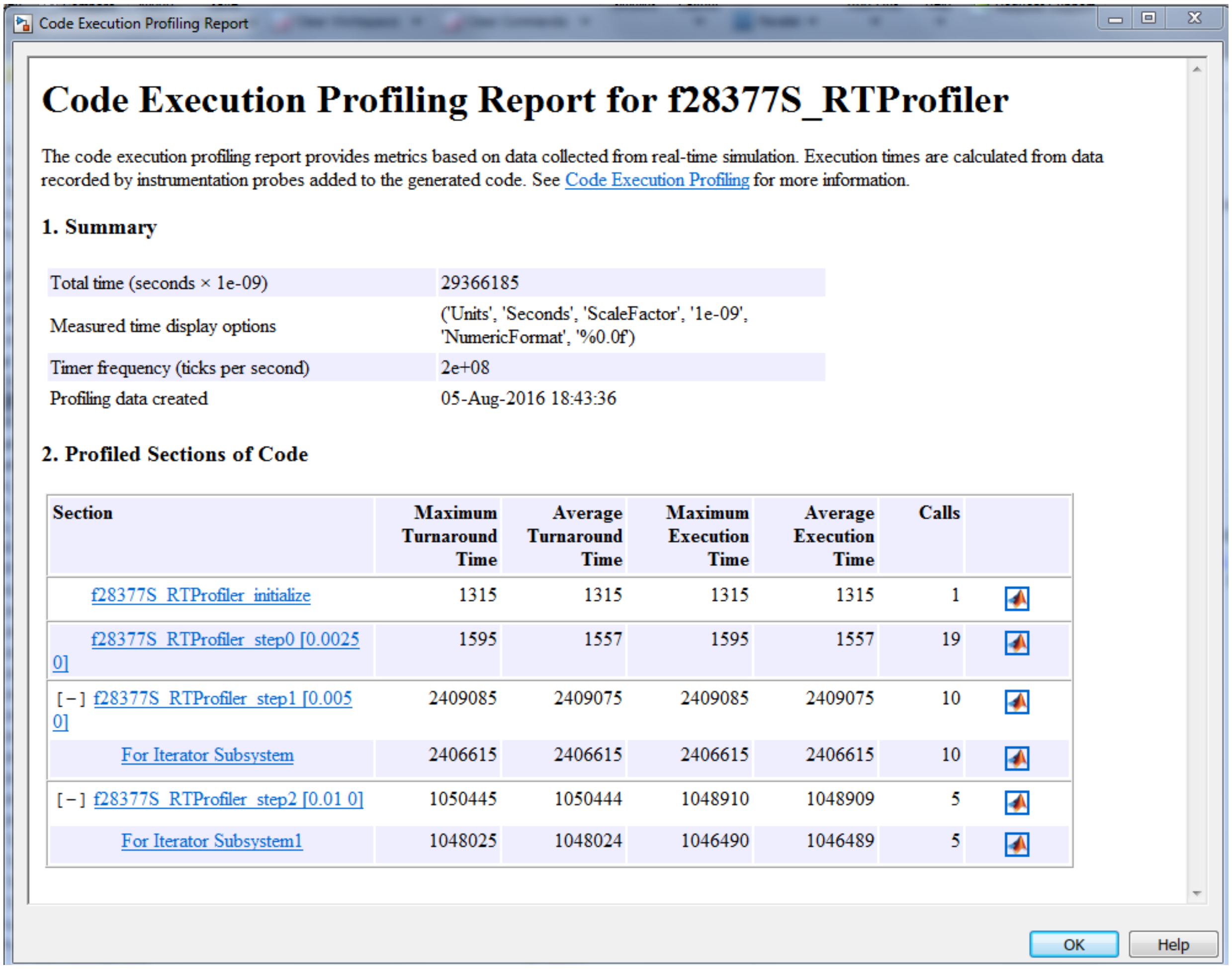
6. Run the following code in MATLAB Command Window to get the profiling data into MATLAB workspace:

```
codertarget.profile.getData('f28377S_RTProfiler')
```

The variable **executionProfile** is available in the MATLAB workspace at the end of execution of this command.

7. 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 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

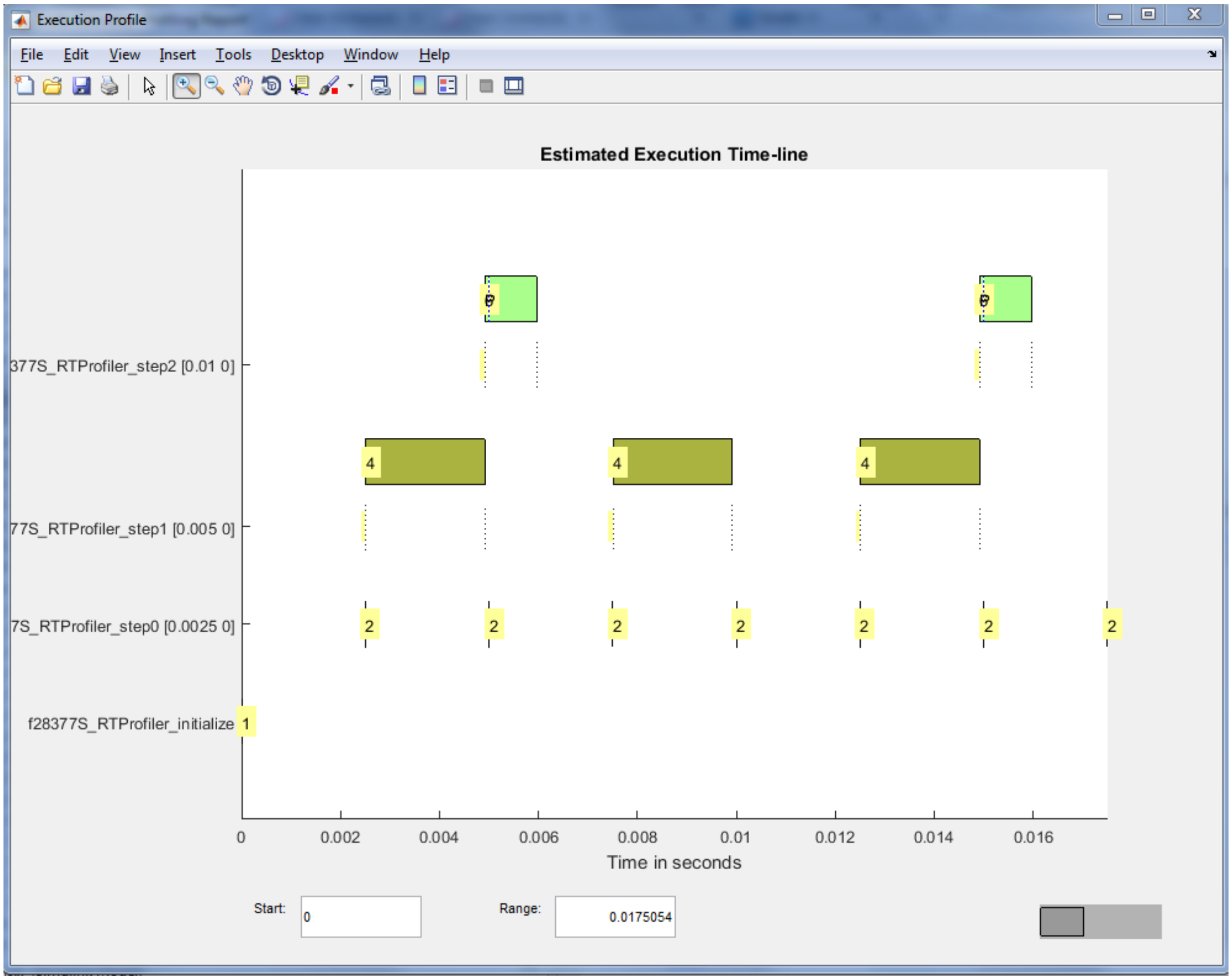
Total time (seconds × 1e-09)	29366185
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	2e+08
Profiling data created	05-Aug-2016 18:43:36

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	

8. 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
Time in seconds
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
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.
- Code execution profiling is currently not supported for models with asynchronously executed subsystems. For example, subsystems triggered by C28x Interrupt Block is not supported.

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

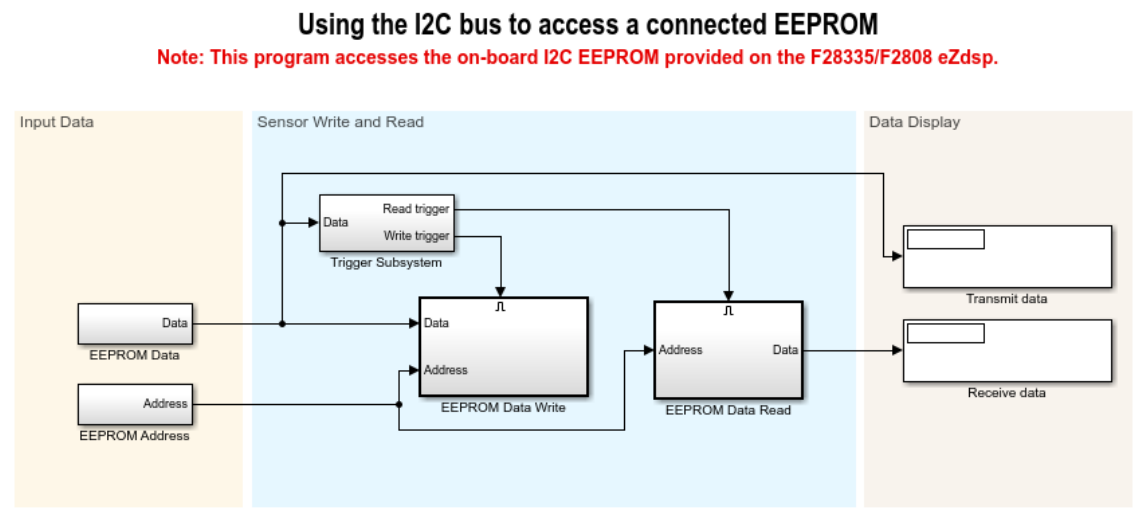
The model writes four bytes to EEPROM and reads back the data from the corresponding EEPROM address to show successful communication.

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 External mode.
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.

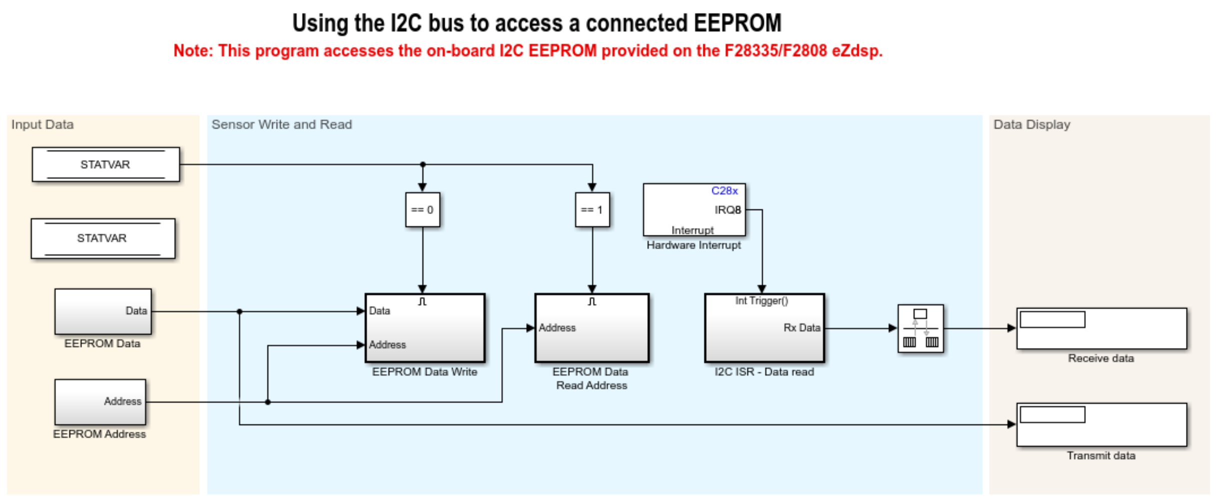
The model consists of these subsystems:

- **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 External mode.
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 External mode.

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

Modeling a Voltage Controller for the DC/DC Buck Converter

This example shows how to model a controller for the DC/DC buck converter using the Embedded Coder Support Package for Texas Instruments® C2000 Processors. The model runs on a TI F28377S or F28379D Launchpad connected to the C2000 DPS BoosterPack.

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.

Prerequisites

Before you start with this example, install these MathWorks® products:

- Simscape™
- Simscape™ Electrical™
- Instrument Control Toolbox™

Required Hardware

- F28377S LaunchPad or F28379D LaunchPad
- Digital Power Buck Converter BoosterPack

Available Models

- DCDC_Buck_Sim.slx can be used to simulate the controller for the DC/DC buck converter.
- f28379D_DCDC_Buck.slx can be used to generate code and load it on the F8379D LaunchPad.
- f28377s_DCDC_Buck.slx can be used to generate code and load it on the F8377S LaunchPad.
- c2000_host_read_12M.slx can be run on the host computer to log signals and tune parameters.

Simulate a Controller for the DC/DC Buck Converter Plant Model

The f28379D_DCDC_Buck model consists of these subsystems:

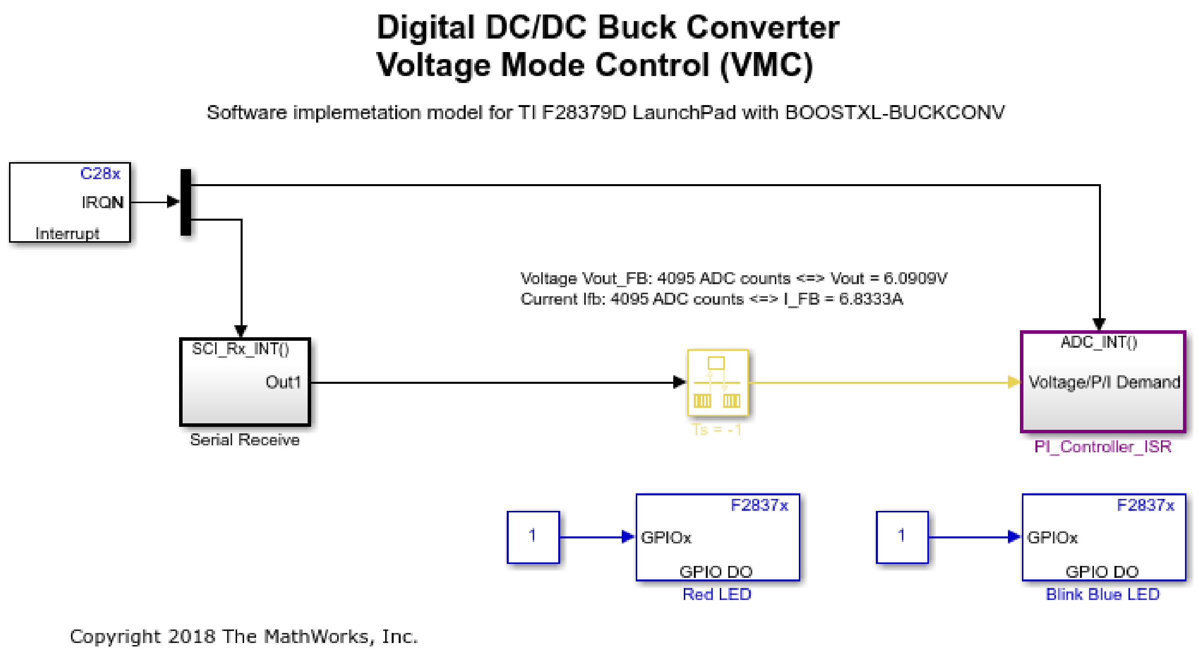
- **Controller**: The discrete proportional integral (PI) controller minimizes the error between the reference voltage and the output voltage. The duty cycle of the PI controller is limited to 60% of the PWM time period.
- **DC/DC Buck Converter**: Simscape™ blocks are used to model the DC/DC buck converter circuitry.
- **Dashboard Controls**: Used to set the reference voltage, switch on/off active load, and tune proportional and integral gains.

Run the Model

1. Open the f28379D_DCDC_Buck model, and click the **Run** button to simulate the model. The initialization file `DCDC_Buck_Param.m` loads the variables required for the simulation.
2. Observe the output waveforms on the scope blocks. You can tune the input parameters using the dashboard controls.

Generate Code for the Controller and Load it on the LaunchPad

In the f28379D_DCDC_Buck model, the PI controller is configured to run on the F28379D Launchpad. The Hardware Interrupt block schedules the interrupt service routine (ISR) tasks.


Digital DC/DC Buck Converter
Voltage Mode Control (VMC)
Software implemetation model for TI F28379D LaunchPad with BOOSTXL-BUCKCONV
C28x
IRQN
Interrupt
SCI_Rx_INT()
Out1
Serial Receive
Voltage Vout_FB: 4095 ADC counts <=> Vout = 6.0909V
Current Ifb: 4095 ADC counts <=> I_FB = 6.8333A
Ts = -1
ADC_INT()
Voltage/P/I Demand
PI_Controller_ISR
1
F2837x
GPIOx
GPIO DO
Red LED
1
F2837x
GPIOx
GPIO DO
Blink Blue LED
Copyright 2018 The MathWorks, Inc.

Run the Model on the LaunchPad

1. Open the f28379D_DCDC_Buck model and generate code by pressing **Ctrl+B**.
2. Follow the build process by opening the diagnostic viewer using the link provided at the bottom of the model canvas. After the code is loaded on the board, a blue LED blinks on the LaunchPad, indicating that the code is running.

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.

Monitor Signals and Tune Parameters on the Host Computer

Configure and Run the Model on the 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_12M model to match the COM port of the host computer:

- c2000_host_read_12M > Serial Configuration
- c2000_host_read_12M > Serial Receive
- c2000_host_read_12M > Serial Send > Serial Send

3. Click on the **Run** button to run the model.

For more information on using the serial connection for your hardware, see http://processors.wiki.ti.com/index.php/Using_the_serial_adapter_of_XDS100.

Monitor the Signals

While the model runs, you can monitor the following signals on the scope:

- **I_FB Current**: The real-time current flowing through the inductor (L1). 4095 ADC counts is equivalent to 6.8333 A inductor current.
- **V_FB Voltage**: The measured output voltage of the system. 4095 ADC counts is equivalent to 6.0909 V output voltage.

Tune the 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.
- **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.
- **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 may 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 may lead to instability of the controller; apply changes smoothly.

More About

C28x SCI Receive

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 Master Transfer block
- Read and write data to SPI EEPROM using the Master 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
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

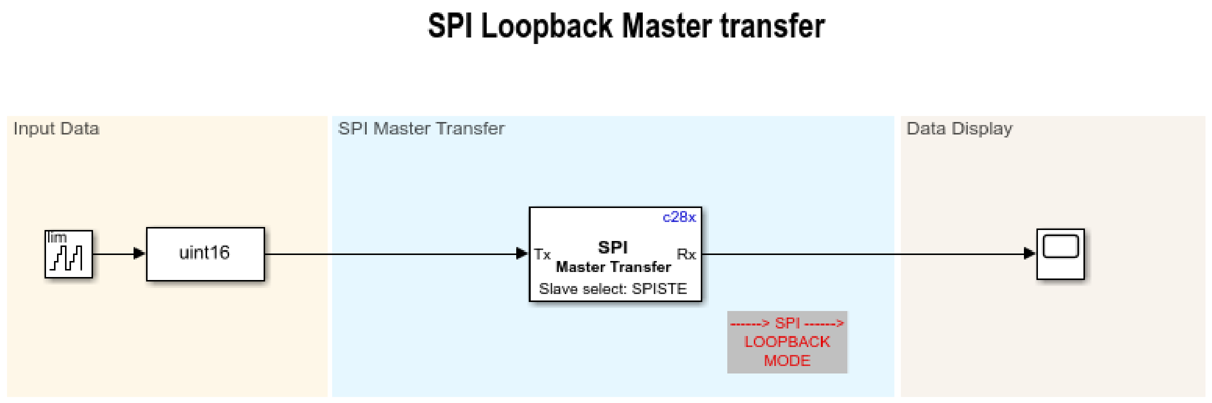
CAT25256 256kB SPI EEPROM Memory

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 Master Transfer Block

The Counter Limited block sends the counter value to the SPI Master 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.

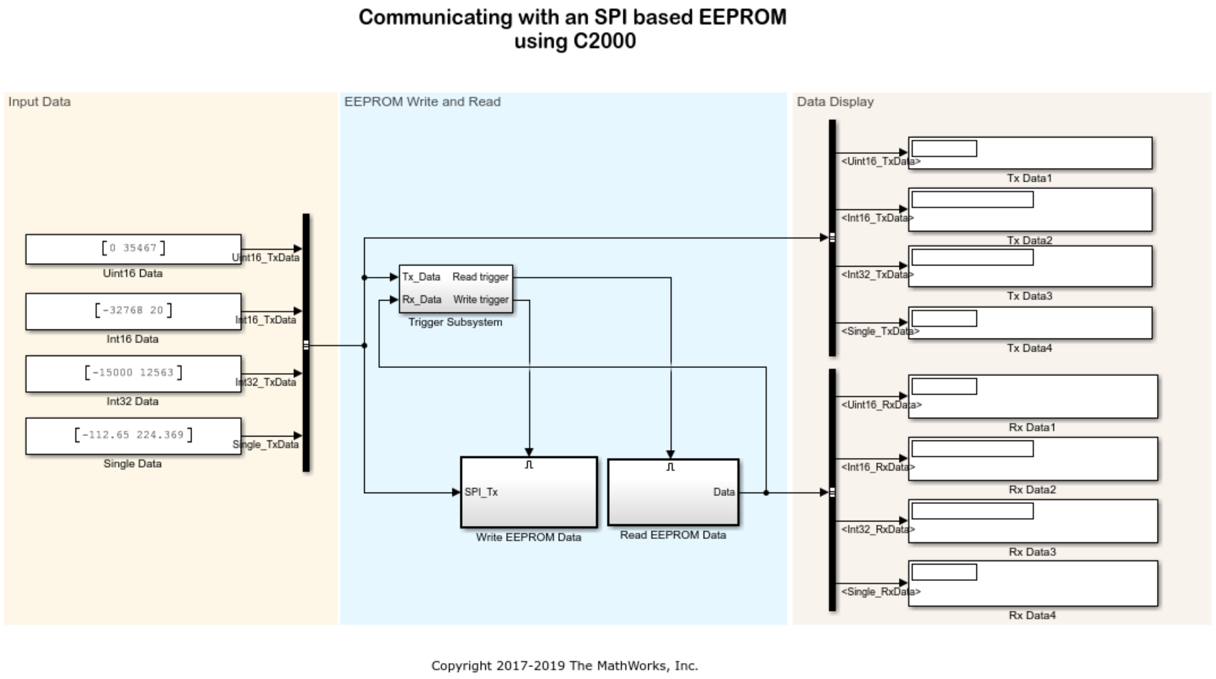


To run the model:

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 4-60.
5. Open **Hardware** tab and click **Monitor & Tune**. Observe the counter values on the scope.

Read and Write Data to SPI EEPROM Using the Master 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 slave is deselected between data transfers. In this case, the `Explicit GPIO calls` option is used to select **Slave select pin** as GPIO1 to ensure that the slave 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 Master Transfer block. After this, the program waits until the EEPROM write operation is complete by monitoring the status flag of the EEPROM.

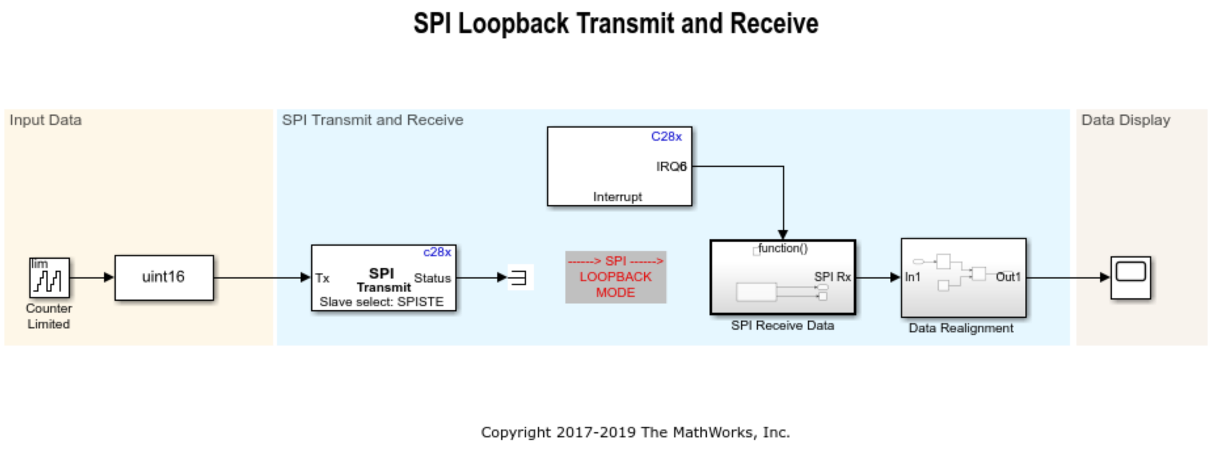
The Read EEPROM Data subsystem reads data back from the memory location 0x0020, and the 8-bit packet data is unpacked to data of required type using the Byte Unpack block. This data is then shown using Display blocks.

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 4-60.
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.

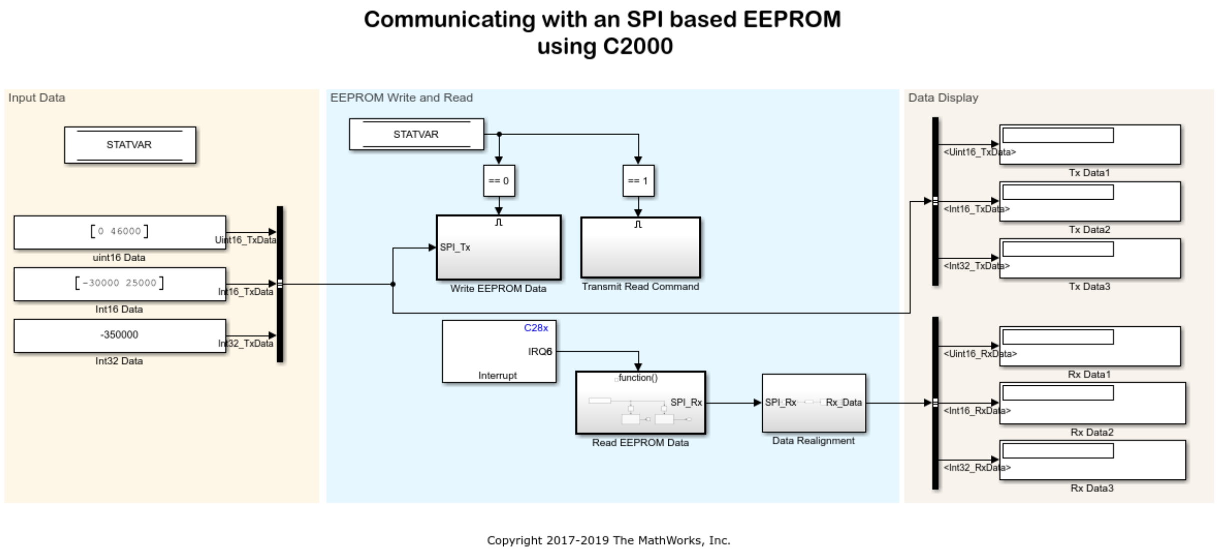
4 Select **Enable Rx interrupt**, and set **FIFO interrupt level(Rx)** to 4.

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 4-60.

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 slave is deselected between data transfers. In this case, the `Explicit GPIO calls` option is used to set **Slave select pin** to GPIO1 to ensure that the slave 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 4-60.
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 Master Transfer
- C28x SPI Receive
- C28x SPI Transmit

Inter-Processor Communication Using IPC Blocks

This example shows how to use the IPC blocks to communicate between the two CPUs of Texas Instruments™ Delfino F2837xD using Simulink® models.

In this example, you will learn how to:

- Send scalar and vector data from CPU1 using IPC Transmit block
- Receive data at CPU2 using IPC Receive block in polling and interrupt modes

Required Hardware

- Texas Instruments™ Delfino F2837xD controlCARDs or F28379D LaunchPad
- A USB serial cable, if your hardware provides serial over USB capabilities

This example is configured for the Texas Instruments F28377D controlCARD with a docking station. 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 1-7.

To select a different target hardware, in the Simulink Editor, browse to **Configuration Parameters > Hardware Implementation > Hardware board**.

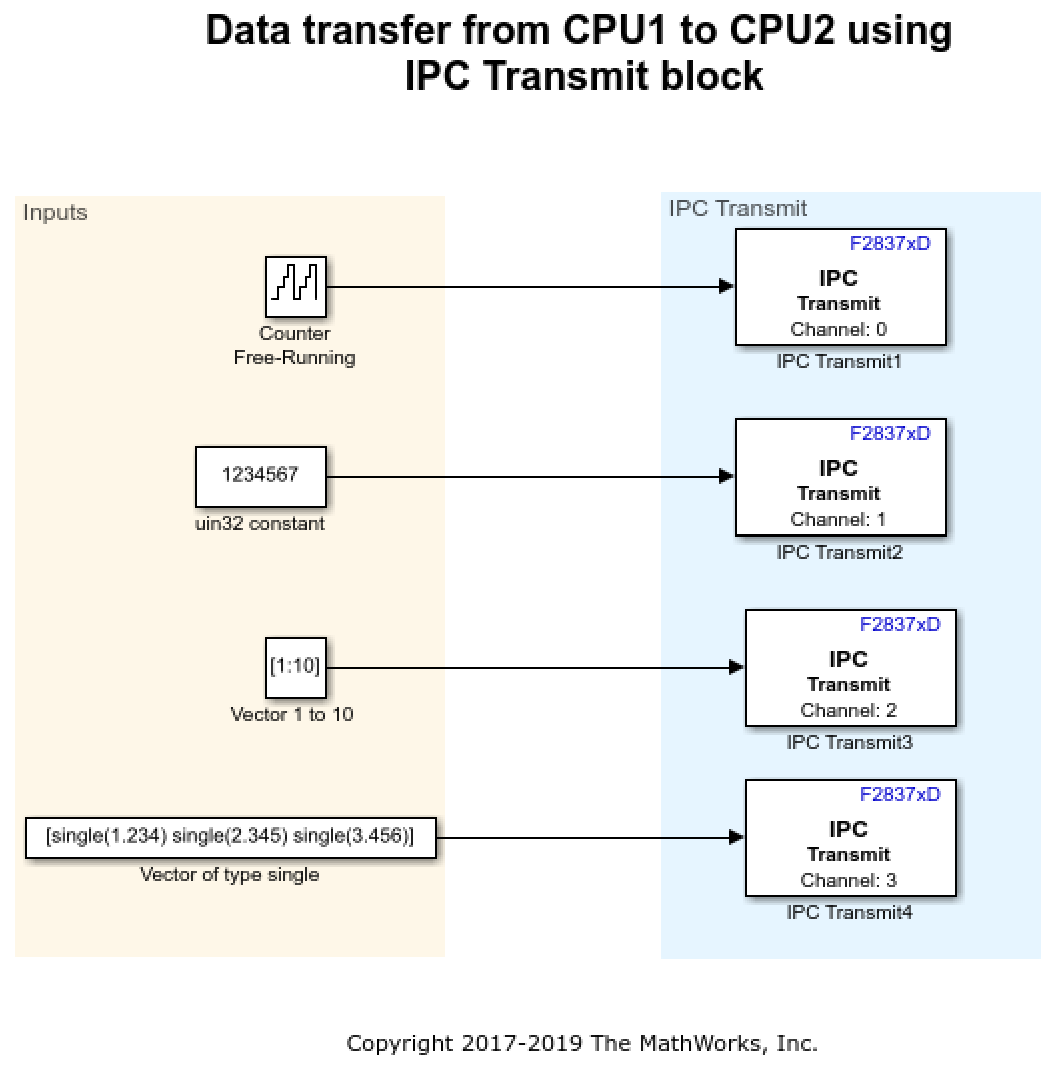
Send Scalar and Vector Data from CPU1 Using IPC Transmit block

This example uses four different channels to transmit data from CPU1 to CPU2. A maximum of 32 channnels can be used.

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.

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.



Receive Data at CPU2 Using IPC Receive Block in Polling and Interrupt Modes

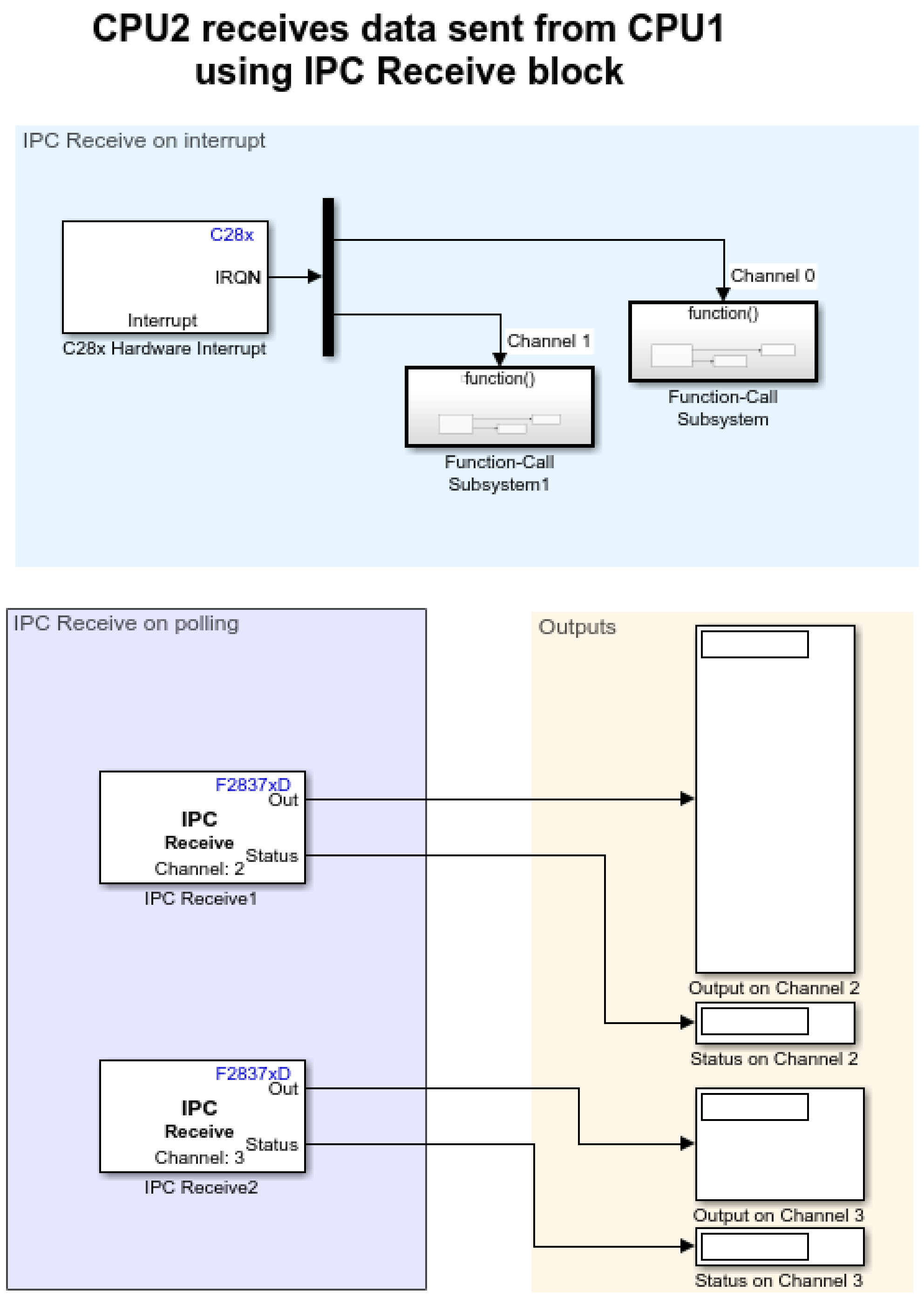
1. Open the c2837xd_ipc_cpu2_rx model.

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

2. Open **Hardware** tab and click **Monitor & Tune**.

3. 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.


CPU2 receives data sent from CPU1
using IPC Receive block
IPC Receive on interrupt
C28x
IRQN
Interrupt
C28x Hardware Interrupt
Channel 0
function()
Function-Call
Subsystem
Channel 1
function()
Function-Call
Subsystem1
IPC Receive on polling
F2837xD
Out
IPC
Receive
Status
Channel: 2
IPC Receive1
F2837xD
Out
IPC
Receive
Status
Channel: 3
IPC Receive2
Outputs
Output on Channel 2
Status on Channel 2
Output on Channel 3
Status on Channel 3

More About

For inter-processor communication, the first two blocks of the global shared RAM (2x16k) are used. 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**.

The current memory settings in the linker file are:

RAMGS_IPCBuffCPU1 : origin = 0x00C000, length = 0x001000

RAMGS_IPCBuffCPU2 : origin = 0x00D000, length = 0x001000

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 “Permanent Magnet Synchronous Motor Field-Oriented Control”. In this example, c28379Dpmsmfocdual_cpu1_ert.slx and c28379Dpmsmfocdual_cpu2_ert.slx models communicate using IPC.

For information about IPC blocks, see:

- F2837xD IPC Receive
- F2837xD IPC 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

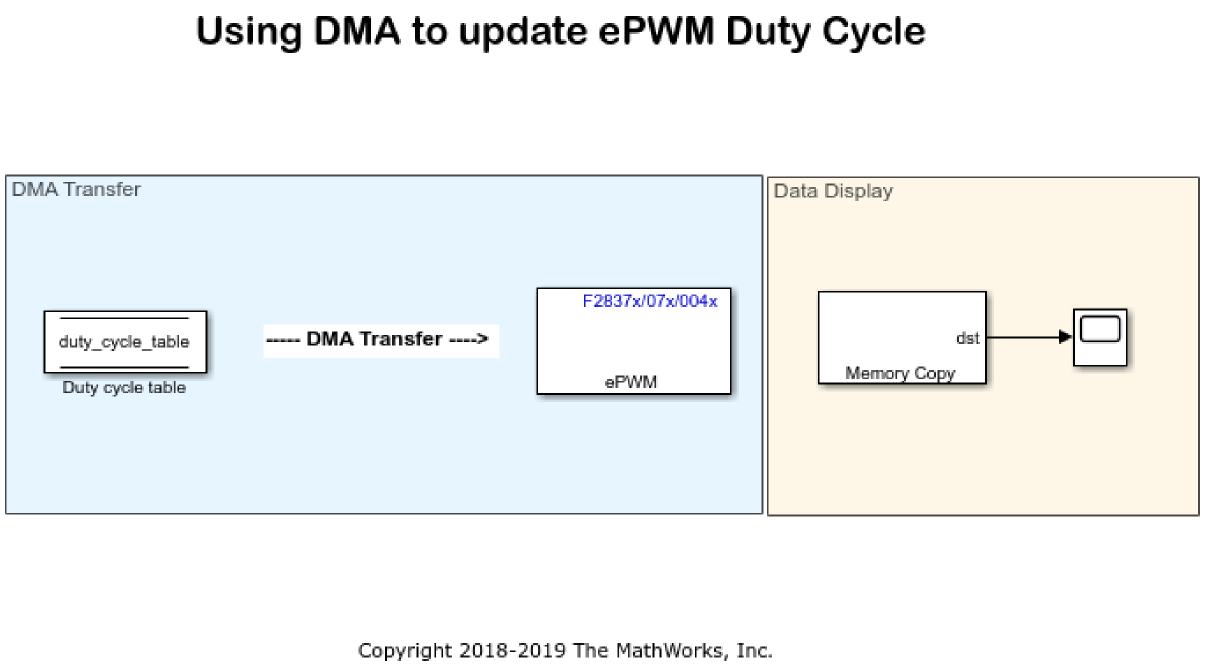
F2833x, F2806x, F2807x, F2837x, or F28004x controlCARD/LaunchPad.

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 Pane”.

The ePWM block is also configured to generate a start of conversion event for module A (SOCA). For more information, see “Event Trigger Pane”.

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”.

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 ePWM

Photovoltaic Inverter with MPPT Using Solar Explorer Kit

This example shows how to implement a photovoltaic (PV) inverter system using the Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors. 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

Prerequisites

Before you start with this example, install these MathWorks® products:

- Simscape™ Electrical™
- Instrument Control Toolbox™

Required Hardware

- Texas Instruments Solar Explorer Kit
- 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.

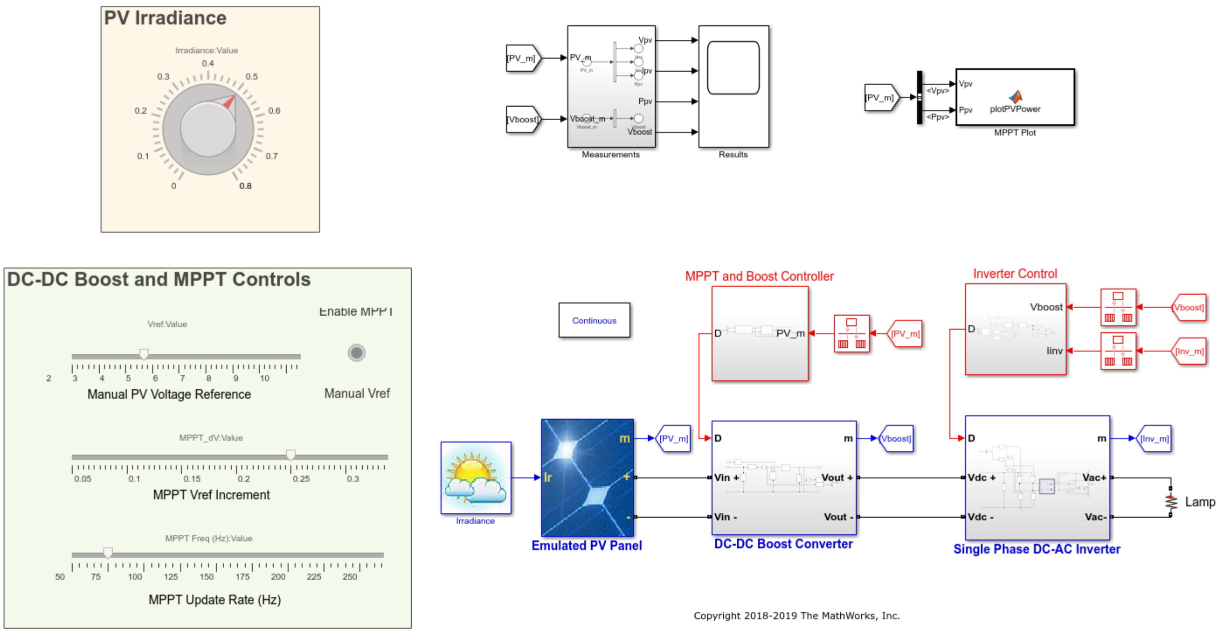
Simulate the Photovoltaic Inverter with MPPT

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.

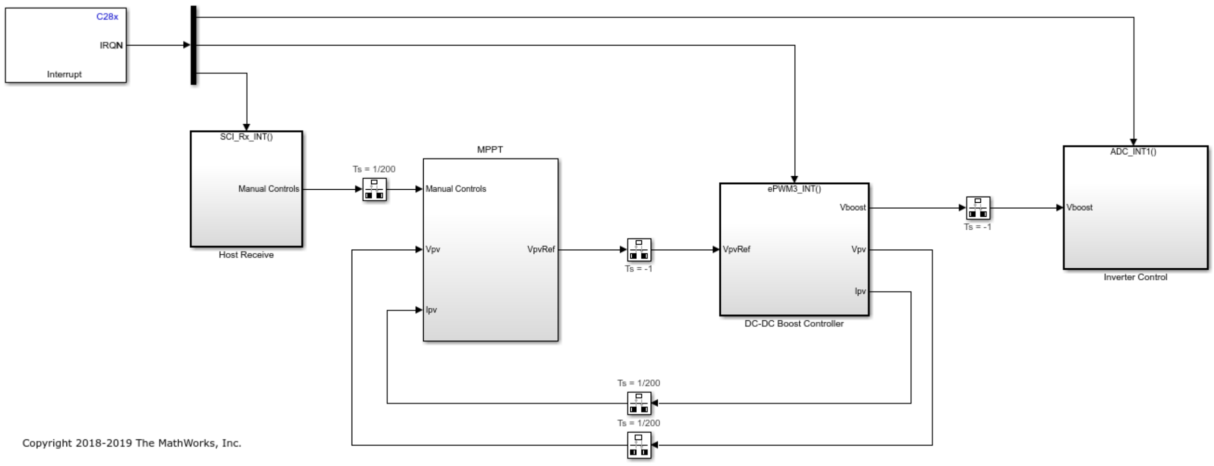
The input to the PV emulator can be changed to run the simulation for different values of irradiance. The controller parameters can be tuned to get a refined performance.

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:

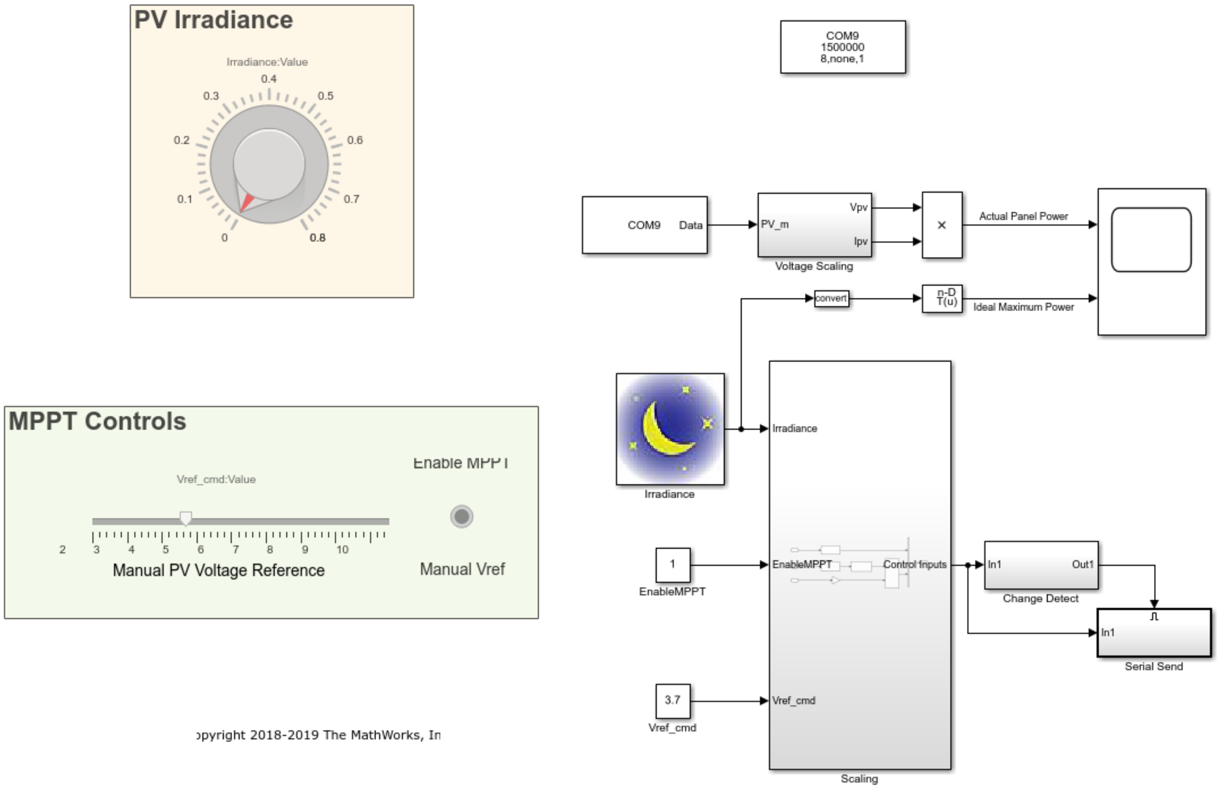
- Closed loop control of the DC-DC boost converter (50 kHz)
- Closed loop control of the DC-AC inverter (20 kHz)
- Setting the irradiance value sent by the user from a host model

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

- “ADC-PWM Synchronization Using ADC Interrupt” on page 4-2
- “Modeling a Voltage Controller for the DC/DC Buck Converter” on page 4-86

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 Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors.

Introduction

Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors 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.

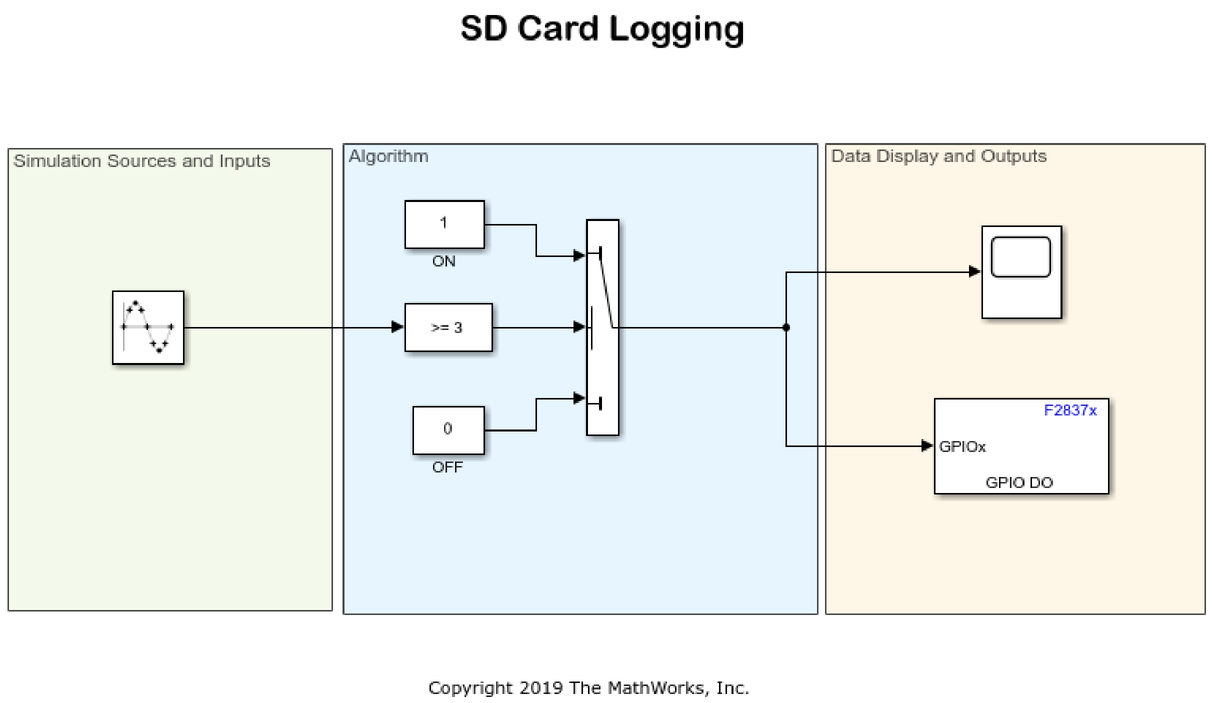
Step 1 - Prepare the Connection and Configure a Simulink Model for MAT-File Logging

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.

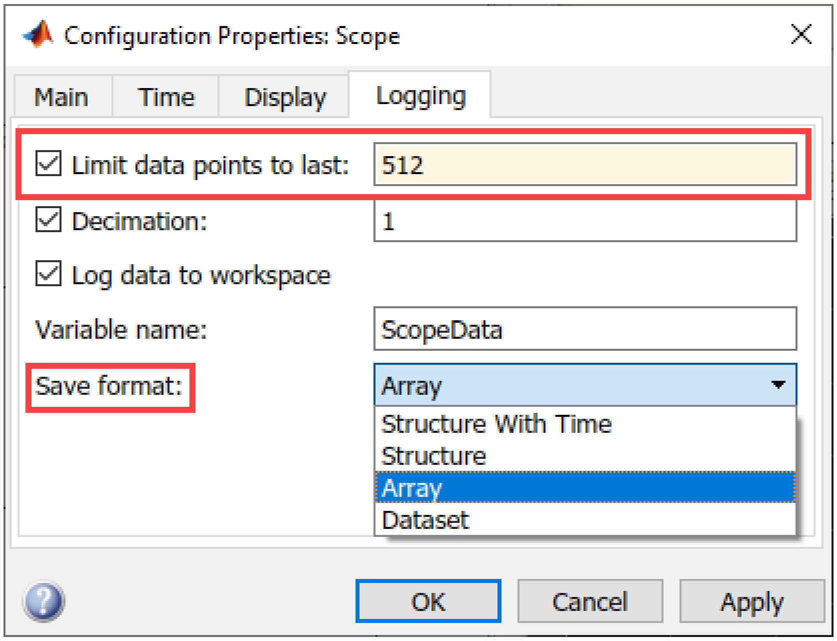
4. To enable MAT-file logging, go to **Target hardware resources > SD card logging**. Select the **Enable MAT-file logging on SD card** checkbox. Also select the desired **SPI module** and **SPI baud rate** and ensure in the SPI_x pane that all configurations are as required.

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**.


Configuration Properties: Scope

☑ Limit data points to last: 512
☑ Decimation: 1
☑ Log data to workspace
Variable name: ScopeData
Save format: Array

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.

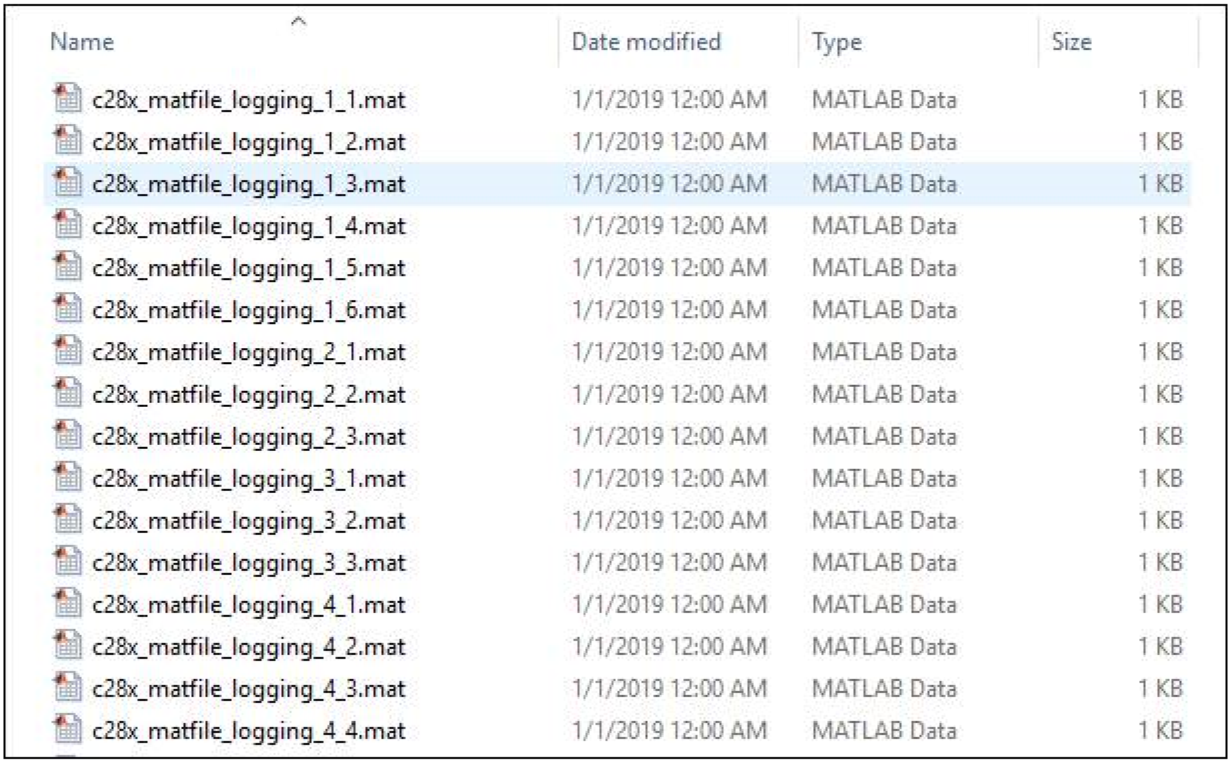
Tip: To learn more about logging data, see “Configure Model to Log Signals on SD Card” on page 2-7.

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

Power Factor Correction Using Boost Converter

This example shows how to implement power factor correction (PFC) using a boost converter with the Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors. The example uses the Texas Instruments Multi-Axis Digital Motor Control Kit along with the Texas Instruments F28035 controlCARD.

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

Prerequisites

Before you start this example, install these MathWorks® products:

- Simscape™ Electrical™ (for PFC simulation)
- Instrument Control Toolbox™ (for signal monitoring)

Required Hardware

- TI Dual Motor Control and PFC Developer's Kit (TMDS2MTRPFCKIT)
- F28035 controlCARD
- Motor Load (BLY171D-24V-4000 Brushless DC Motor and DRV8312-C2-KIT)

Available Models

- PFC_MultiAxisKit_Sim.slx simulates the plant model and controller for the PFC system.
- c28035pfc.slx generates code and loads it on the F28035 controlCARD.
- c2000_host_pfc.slx runs on the host computer to log signals.

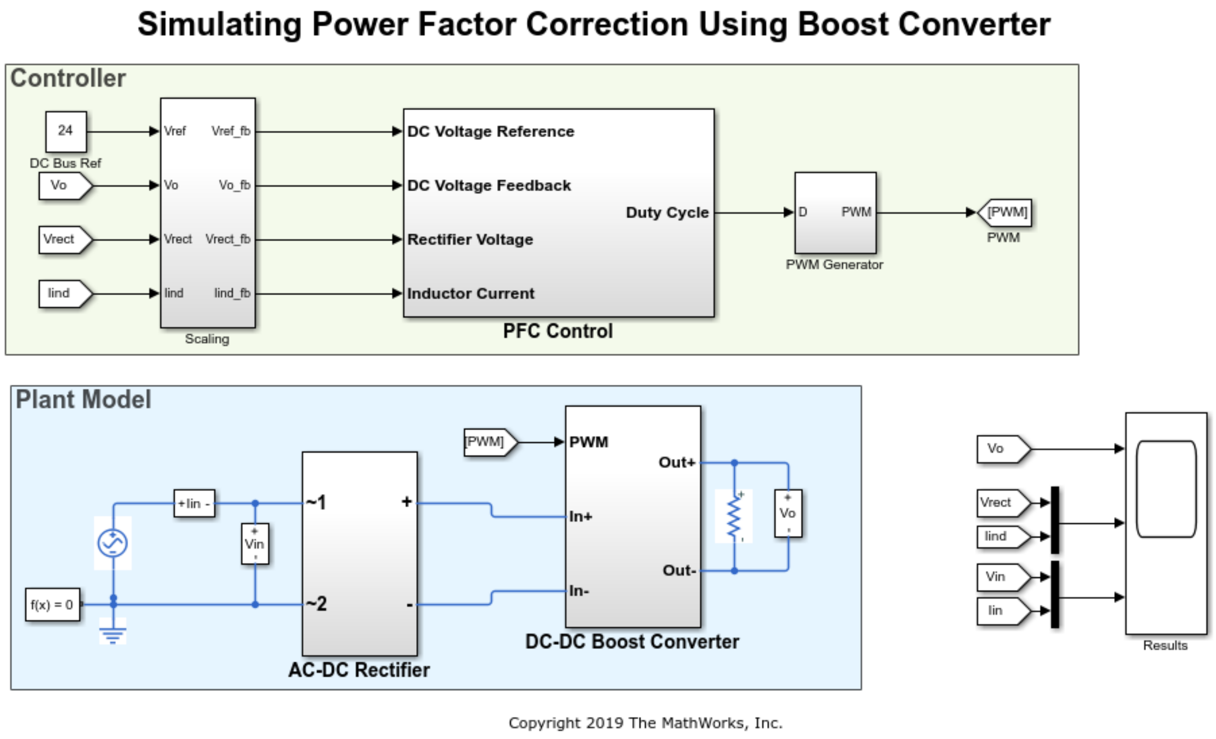
Simulate PFC Using Boost Converter

The simulation model consists of the plant model and the controller. The plant model consists of two major modules:

- **AC-DC Rectifier**: This module takes 13 to 16V 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('PFC_MultiAxisKit_Sim.slx');
```



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 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 outer the voltage loop.

The output of the PFC controller is the PWM duty cycle of the DC-DC boost converter. The controller operates at a rate of 50 kHz.

Run the Model

1. Open the PFCMultiAxisKitSim model, and click the Run button to simulate the model.

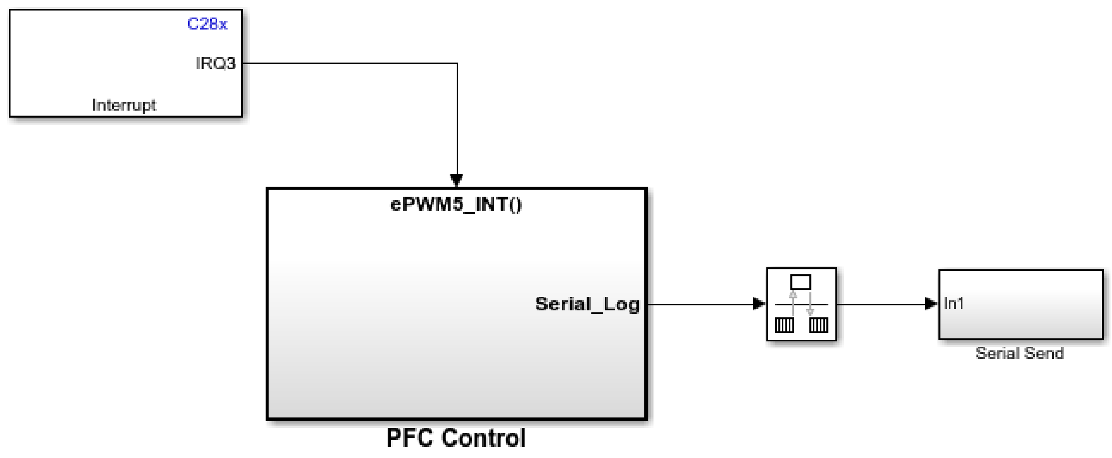
2. Observe the output waveforms on the `Scope` block.

Generate Code for the Controller and Load it on the controlCARD

The deployment model has a real-time interrupt service routines (ISR) configured to trigger PFC control at the rate of 50 kHz.

```
open_system('c28035pfc.slx');
```

PFC Controller for TI Multi-Axis Digital Motor Control Kit



Load the Model on the controlCARD

1. Turn on [M4]SW1 on the kit which turns on the power supply for smooth precharging of the DC bus.

2. Turn on [M4]SW3 on the kit to power on the controlCARD.

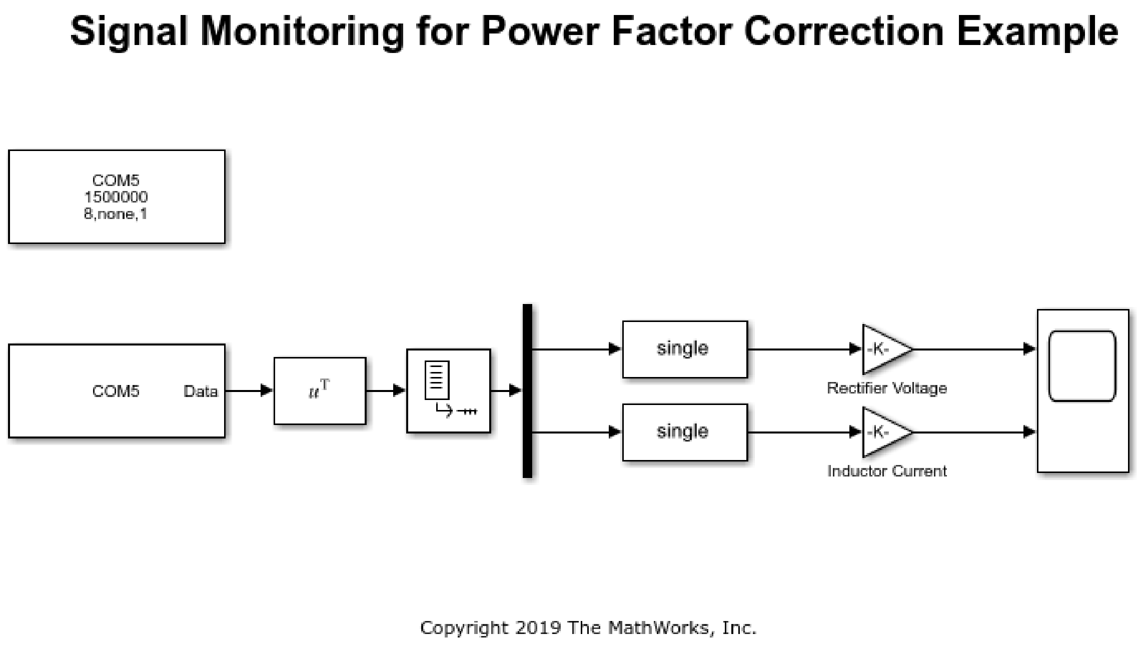
3. Open the c28035pfc model and generate code by pressing **Ctrl+B**.

4. Follow the build process by opening the diagnostic viewer using the link at the bottom of the model canvas.

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
COM5
1500000
8,none,1
COM5
Data
single
single
-K-
-K-
Rectifier Voltage
Inductor Current
Copyright 2019 The MathWorks, Inc.

While the model runs, you can monitor the rectified DC voltage and input boost converter current to analyze the performance of PFC control.

More About

- "ADC-PWM Synchronization Using ADC Interrupt" on page 4-2
- "Modeling a Voltage Controller for the DC/DC Buck Converter" on page 4-86
- "Photovoltaic Inverter with MPPT Using Solar Explorer Kit" on page 4-106

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.

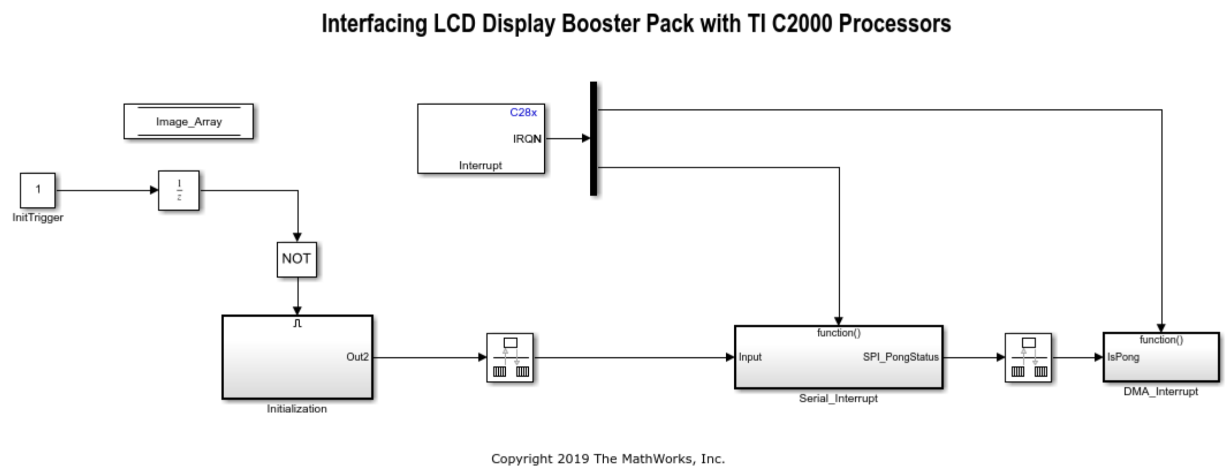
Kentec QVGA Display Pin	F28379D Pin	Application
SCS	GPIO 124	To select LCD for SPI communication
SDC	GPIO 22	To select whether to send data or commands to LCD
SCL	GPIO 60	SPI_A Clock pin
SDI	GPIO 58	SPI_A SIMO pin
PWM	GPIO 0	For backlight
RESET	GPIO 159	To reset LCD
GND	GND	Ground
VCC	VCC	Supply

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** colour 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

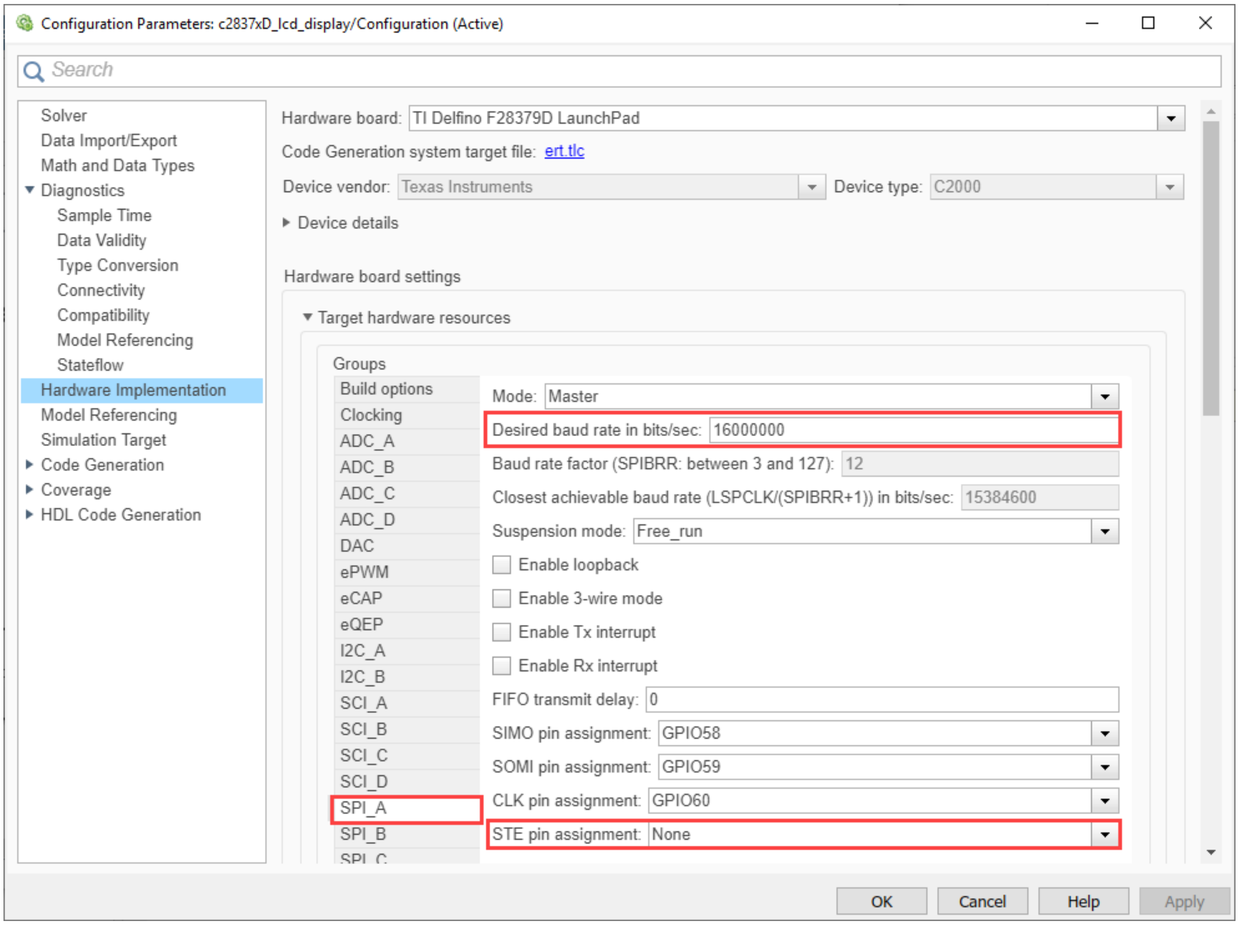
1. Open the LCD Display model.

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 **STE pin** to `None`.


Configuration Parameters: c2837xD_lcd_display/Configuration (Active)

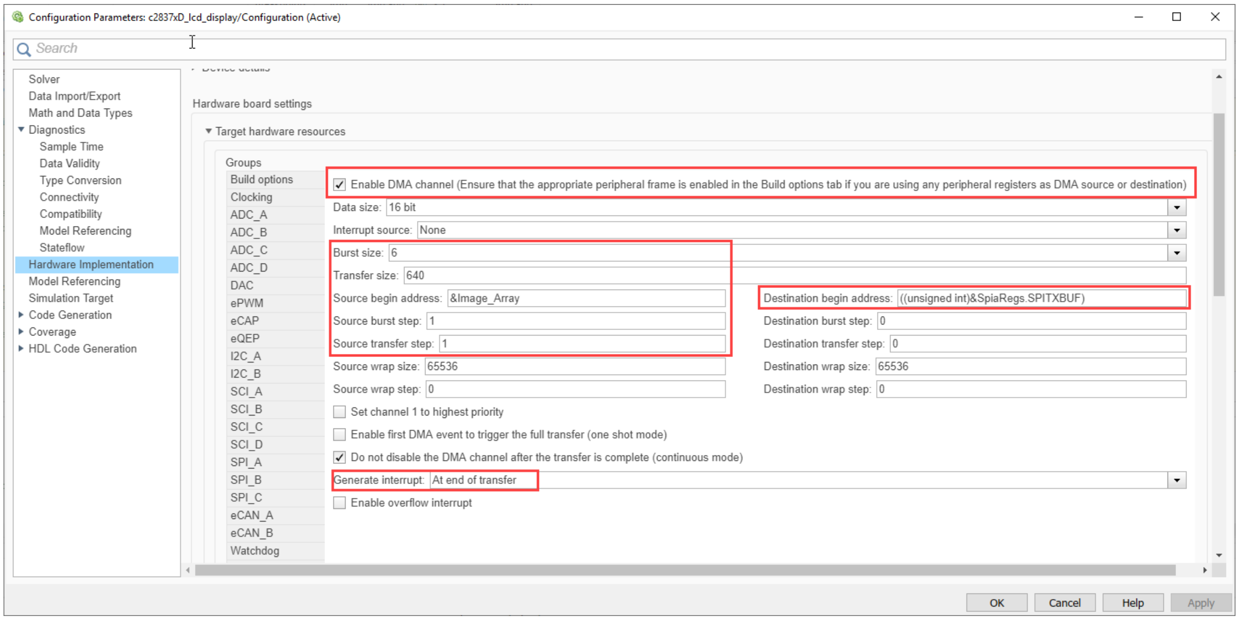
Hardware board: TI Delfino F28379D LaunchPad
Code Generation system target file: ert.tlc
Device vendor: Texas Instruments Device type: C2000
Device details

Hardware board settings

Target hardware resources

Mode: Master
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: ☐
FIFO transmit delay: 0
SIMO pin assignment: GPIO58
SOMI pin assignment: GPIO59
CLK pin assignment: GPIO60
STE pin assignment: None

6. Go to **Hardware Implementation > Target Hardware Resources > DMA_ch1** and update parameter values as shown.


Configuration Parameters: c2837xD_lcd_display/Configuration (Active)

Hardware board settings

Target hardware resources

Groups: Build options, Clocking, ADC_A, ADC_B, ADC_C, ADC_D, 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, Watchdog

✓ 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.