C2000™ Microcontroller Blockset

Getting Started Guide

MATLAB® & SIMULINK®

R2023**a**

MathWorks®

How to Contact MathWorks

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

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

C2000™ Microcontroller Blockset Getting Started Guide

Trademarks

Patents

Revision History

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

Contents

1 Product Overview

2 About C2000 Microcontroller Blockset

Product Overview

Design, simulate, and implement applications for Texas Instruments C2000 microcontrollers

C2000 Microcontroller Blockset enables you to model digital power conversion and motor control applications for TI C2000 microcontrollers (MCUs). The blockset includes peripheral blocks, such as digital IO, ADC, and ePWM, to perform simulations of control algorithms that require ADC-PWM synchronization in industrial and automotive applications using C2000 MCUs.

The blockset enables you to partition algorithms for multicore execution with inter-processor data communication (IPC) and co-processor (Control Law Accelerator) blocks. You can connect your Simulink® model directly to supported hardware for live I/O data exchange for rapid prototyping.

You can generate optimized code for C2000 MCUs for real-time and interrupt-driven execution of code using IQ Math and related optimization routines and perform real-time signal monitoring, parameter tuning, and processor-in-the-loop (PIL) testing (with Embedded Coder®). The blockset includes reference examples that help you build and deploy motor control applications on C2000 MCUs (with Motor Control Blockset™ and Embedded Coder).

C2000TM is a trademark of Texas Instruments®.

About C2000 Microcontroller Blockset

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
TI F2838x	F28388D, F28388S, F28386D, F28386S, F28384D, and F28384S
TI F28002x	F280025, F280025C, F280024, F280024C, F280023, F280023C, F280022 and F280021
TI F280025C LaunchPad	F280025C
TI F28003x	F280033, F280034, F280036, F280036C, F280037, F280037C, F280038, F280038C, F280039, and F280039C
TI Concerto F28M35x	F28M35H52C, F28M35H22C, F28M35M52C, F28M35M22C, F28M35M20B, and F28M35E20B
TI Concerto F28M36x	F28M36P63C, F28M36P53C, F28M36H53C, F28M36H53B, F28M36H33C, and F28M36H33B

See Also

“Hardware Setup for C2000 Microcontroller Blockset”

Overview of Creating a Model and Generating Executable for C2000 Processors

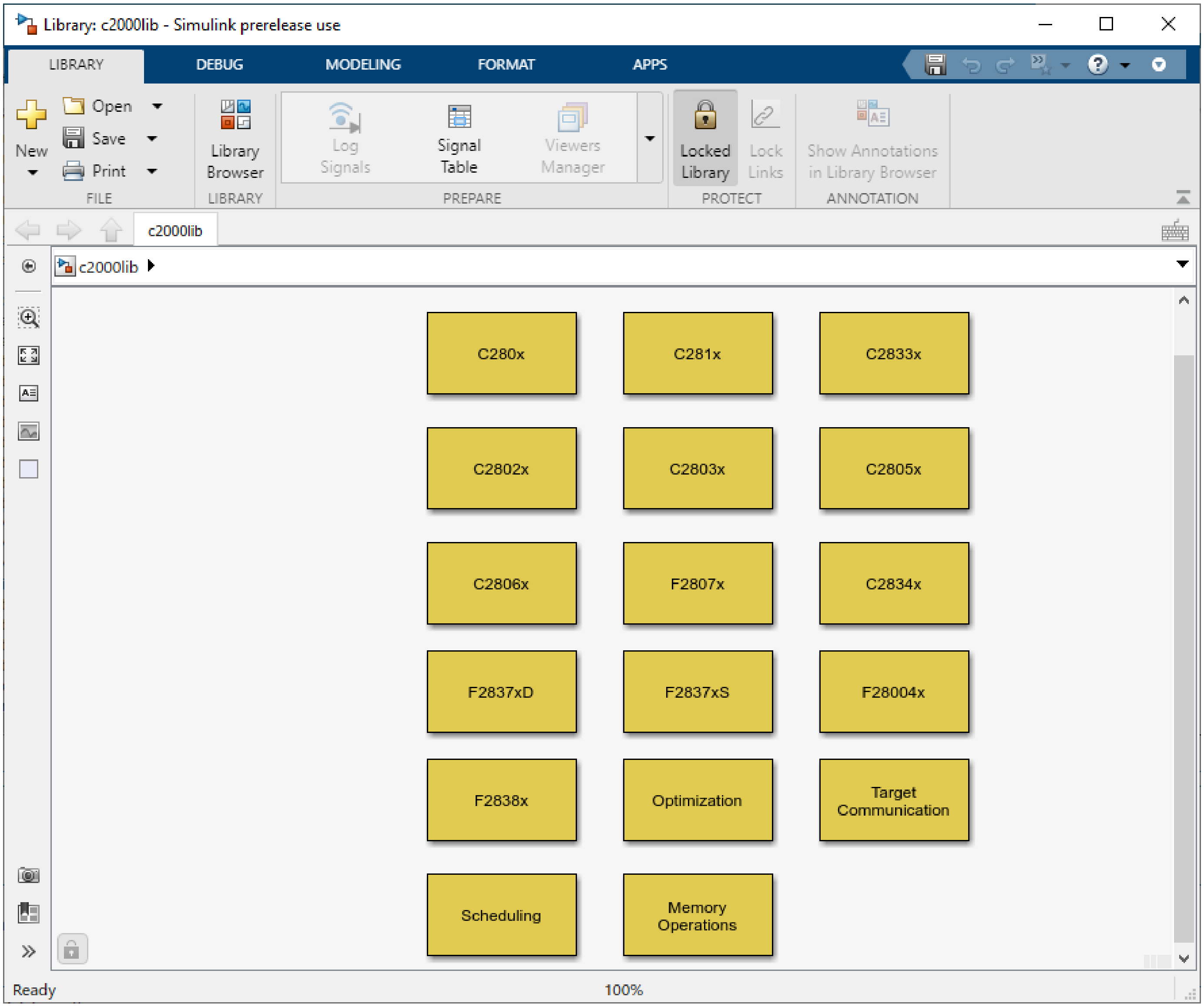
In this section...

Accessing the Embedded Coder Block Library

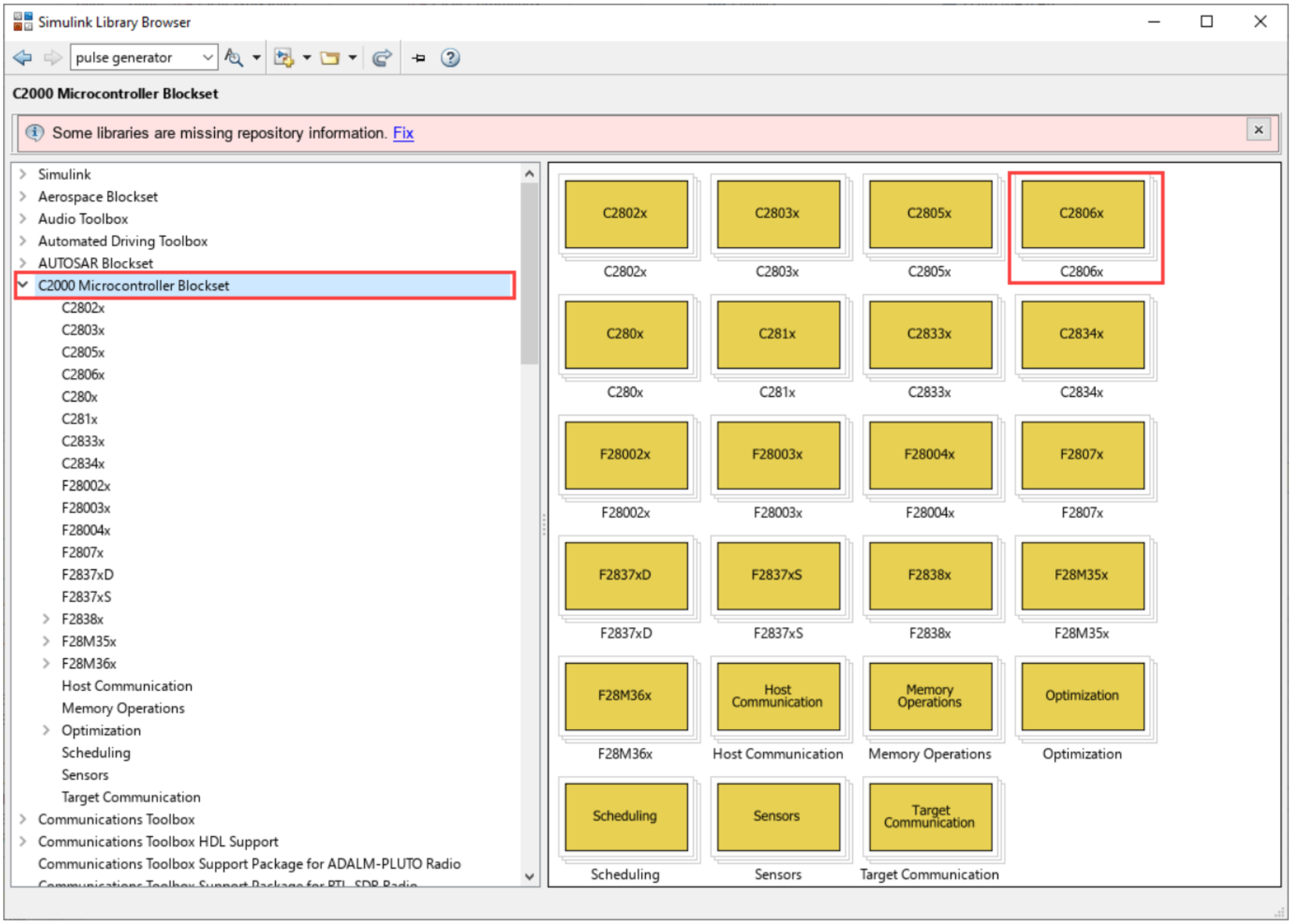
Describes how to access the block library and open the required target hardware.

1. After you have “Hardware Setup for C2000 Microcontroller Blockset”, you can open the block library for C2000 Microcontroller Blockset.
2. To open the C2000 Microcontroller Blockset Library, at the MATLAB® command prompt, enter the command:

   ```
   c2000lib
   ```



Alternately from the Simulink Library Browser, locate, and select C2000 Microcontroller Blockset.



Some libraries are missing repository information. Fix

3 Open the required target block to access the device driver blocks.

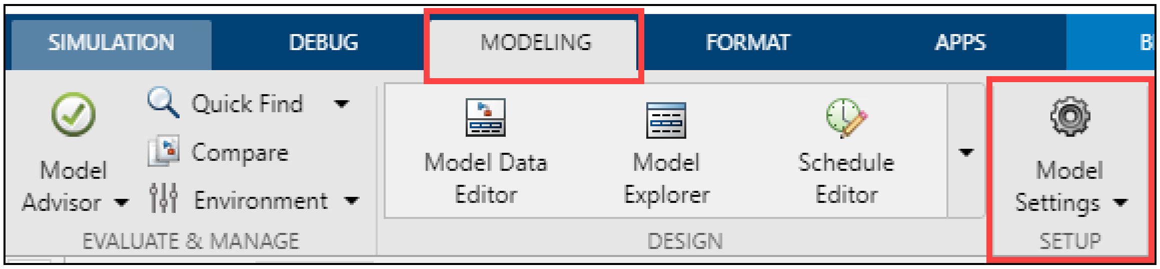
4 Create your real-time model for your application the same way you create other Simulink models. Select blocks to build your model from the following sources or products:

- 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

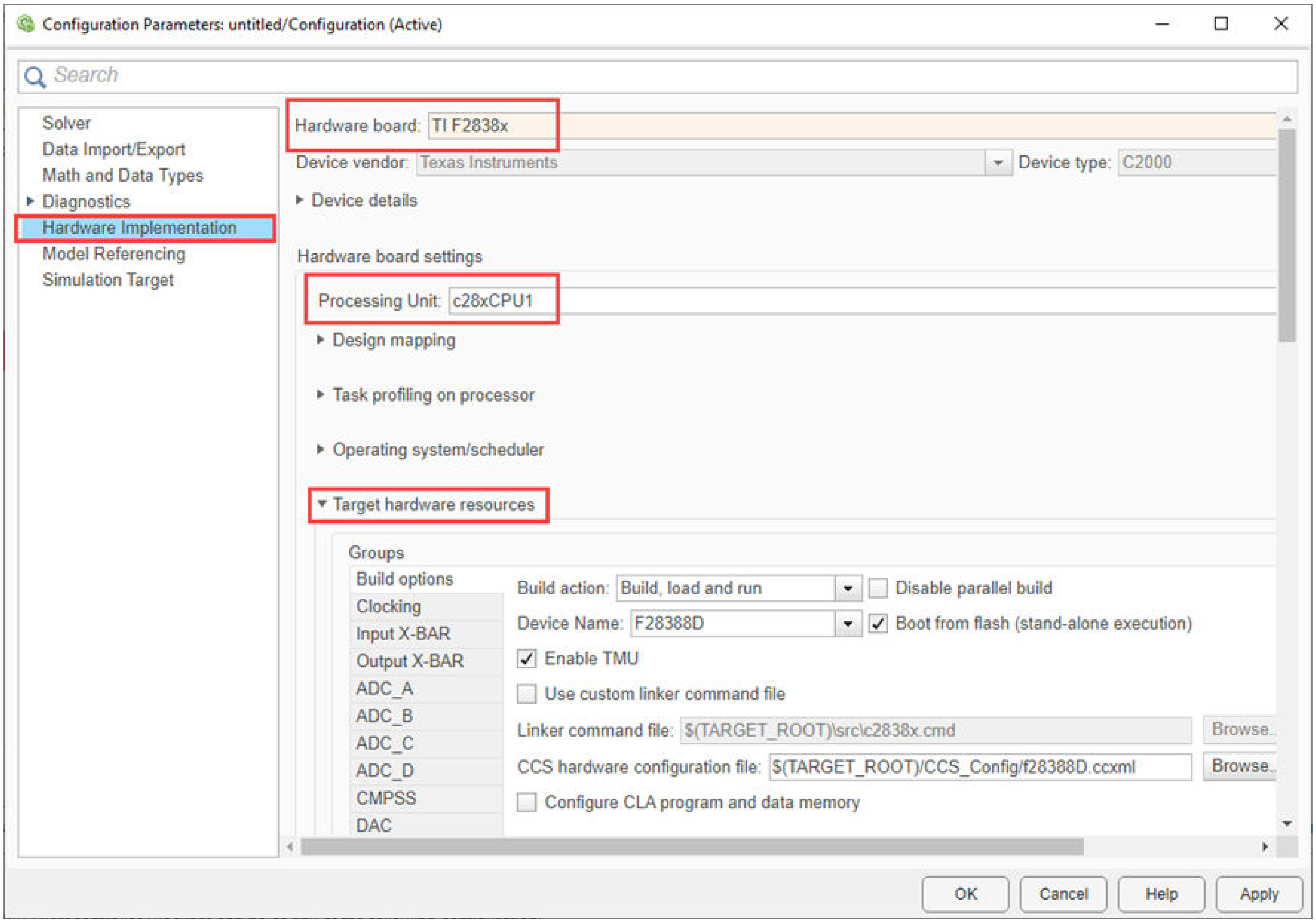
Configuring Target Hardware Resources

Describes how to configure target hardware resources using configuration parameters.

1 After adding the required blocks for the model, Open **Modeling** tab and Click **Model Settings**.



2 In the **Configuration Parameters** window, click **Hardware Implementation** and select the **Hardware Board**.


Configuration Parameters: untitled/Configuration (Active)

Hardware board: TI F2838x

Device vendor: Texas Instruments

Device type: C2000

Processing Unit: c28xCPU1

Build action: Build, load and run

□ Disable parallel build

Device Name: F28388D

☑ Boot from flash (stand-alone execution)

☑ Enable TMU

□ Use custom linker command file

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

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

□ Configure CLA program and data memory

3 Configure the **Target Hardware Resources** as required. For more details refer "Model Configuration Parameters for Texas Instruments C2000 Processors" and "Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)"

Note The peripherals supported in C2000 Microcontroller Blockset can be of any of the following configuration:

- Blocks only - features that can be configured only through the block. (Hardware Interrupt, etc.)

- Blocks + Configuration Parameters - features that can configured in both block and configuration parameters. (ADC, ePWM, SCI, SPI, etc)
- Configuration Parameters only - features that can be configured only through configuration parameters. (EMIF, DMA, Clock Configuration, etc.)

Configuring Additional Options for Hardware Selection

Following configuration settings are done automatically as part of a TI C2000 hardware selection. On selecting different hardware other than TI C2000 target, the below configurations are re-configured to default.

When changing **ModelReferenceNumInstancesAllowed** from `Multiple` to `One`, the name for the signal logged in child model (output signal) when running external mode using **XCP over CAN** will change.

- If set to `One`, you get `<ChildModelName>_B.<outputsignalname>`
- If set to `Multiple`, you get `<TopModelName>_DW.Model_InstanceData.rtb.<outputsignalname>`

`hCS` is the config set object for the model. You can use the following command to get the config set object.

```
hCS = getActiveConfigSet(mdlname);
```

- Go to **Configuration Parameters** > **Math and Data Types**and set the **Simulation behavior for denormal numbers** parameter to `Flush to Zero (FTZ)`.

  ```
  set_param(hCS,'DenormalBehavior', 'FlushToZero');
  ```
- Go to **Configuration Parameters** > **Math and Data Types**and set the **Default for underspecified data type** parameter to `Single`.

  ```
  set_param(hCS,'DefaultUnderspecifiedDataType','single');
  ```
- Go to **Configuration Parameters** > **Code Generation** > **Interface**and set the **Code replacement libraries** parameter to `TI C28x` for the C28x core of TI C2000 processors.

  ```
  set_param(hCS,'CodeReplacementLibrary','TI C28x');
  ```
- Go to **Configuration Parameters** > **Code Generation** > **Optimization** > **Advance parameters** and select the **Maximum stack size (bytes)** parameter depending on the processor.

Device Family

SL No	Device Family	Maximum Stack Size(bytes)
1	F281x	512
2	F2802x, F2803x, F2805x, F2834x, F2804x, F28M35x, F28M36x	768
3	F2806x, F2837x, F28004x, F28003x, F2838x, F28002x	1024
4	ARM-Core	2560

```
set_param(hCS,'MaxStackSize','1024');
```

- Go to **Configuration Parameters** > **Hardware Implementation** > **Device details**and set the **Support long long** parameter to `On`.

  ```
  set_param(hCS,'ProdLongLongMode','on');
  ```

  Note Models using CLA requires long long mode to be set to `off`.

- Go to **Configuration Parameters** > **Code Generation** > **Optimization**and set the **Remove root level I/O zero initialization** parameter to `On`.

  ```
  set_param(hCS,'ZeroExternalMemoryAtStartup', 'on');
  ```

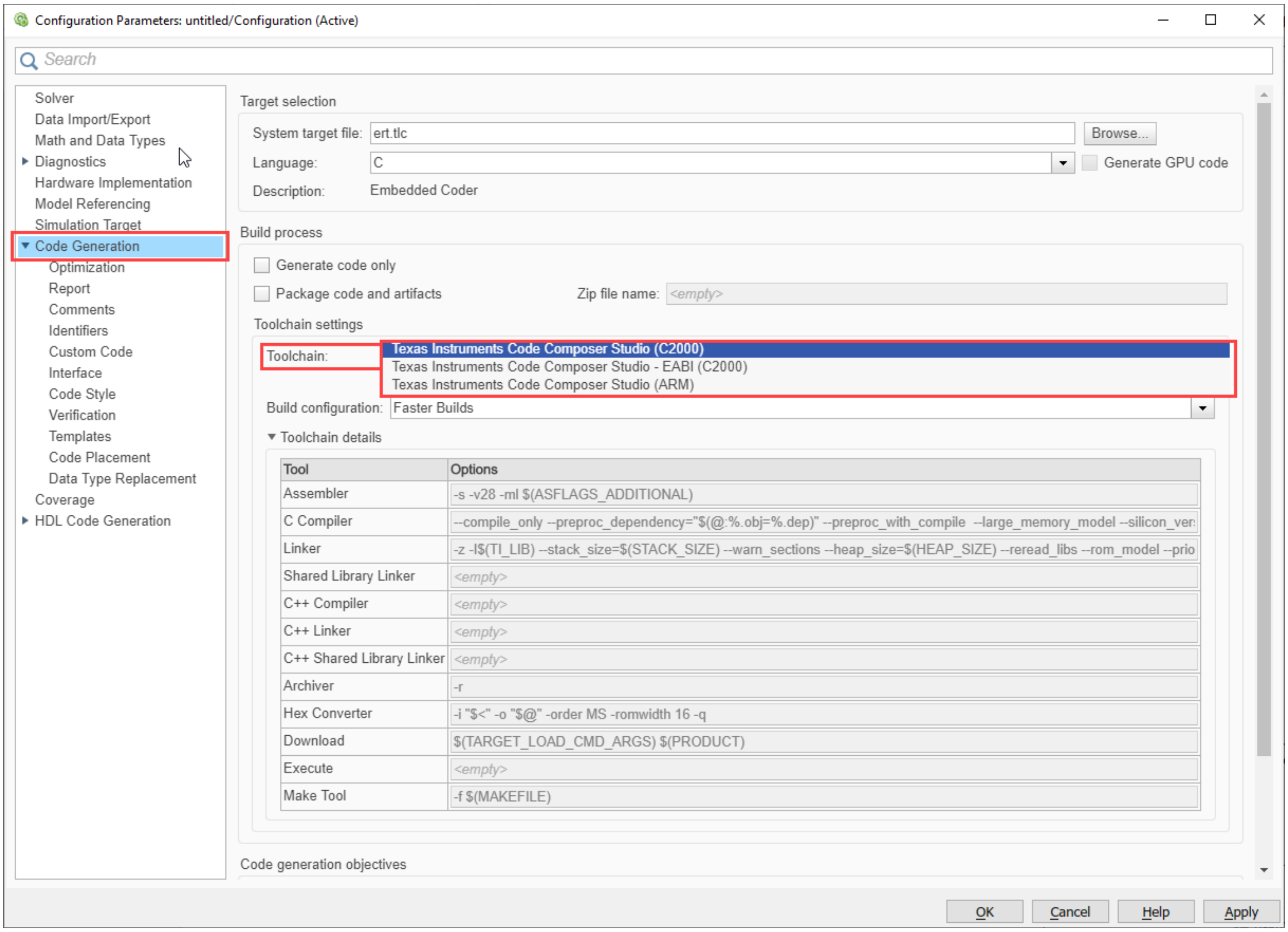
Ensure the above mentioned configurations are done when the model has to run in SIL mode for a custom board. In SIL mode, ensure following additional setting is done:

Go to **Configuration Parameters** > **Code Generation** > **Verification**and set the **Enable portable word sizes** parameter to `On`. This will ensure identical code generation for a model in PIL and SIL mode.

Configuring Additional Options for Code Generation

You can follow the steps below to configure additional options like Compiler flag, CRL and Custom Code, etc required for code generation in **Code Generation** pane.

1 In the **Configuration Parameters** window, click **Code Generation** pane.



Target selection

System target file: ert.tlc

Language: C

□ Generate GPU code

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)
Texas Instruments Code Composer Studio - EABI (C2000)
Texas Instruments Code Composer Studio (ARM)

Build configuration: Faster Builds

Toolchain details

Tool	Options
Assembler	-s -v28 -ml $(ASFLAGS_ADDITIONAL)
C Compiler	--compile_only --preproc_dependency="$(@:%.obj=%.dep)" --preproc_with_compile --large_memory_model --silicon_ver:
Linker	-z -I$(TI_LIB) --stack_size=$(STACK_SIZE) --warn_sections --heap_size=$(HEAP_SIZE) --reread_libs --rom_model --prio
Shared Library Linker	<empty>
C++ Compiler	<empty>
C++ Linker	<empty>
C++ Shared Library Linker	<empty>
Archiver	-r
Hex Converter	-i "$<" -o "$@" -order MS -romwidth 16 -q
Download	$(TARGET_LOAD_CMD_ARGS) $(PRODUCT)
Execute	<empty>
Make Tool	-f $(MAKEFILE)

Code generation objectives

2 By default,

- **Toolchain** for **C28x core** of all TI C2000 processors expect TI F2838x processor is **Texas Instruments Code Composer Studio (C2000)**. This generates an executable in Common Object File Application Binary Interface (**COFF ABI**) format.
- **Toolchain** for **TI F2838x (C28x)** processor is **Texas Instruments Code Composer Studio - EABI (C2000)**. This generates an executable in ELF Application Binary Interface (**EABI**) format.
- **Toolchain** for **ARM core** is **Texas Instruments Code Composer Studio (ARM)**.

COFF ABI format will interpret double datatype in Simulink as 32-bit floating point for C28x processors and ELF ABI format will interpret double datatype in Simulink as 64-bit floating point for TI F2838x (C28x) processor. For more information related to COFF ABI and ELF ABI, refer to the TI documentation.

Note

- ELF ABI (EABI) format will only work with TI F2838x processor and rest of the TI C2000 processors with COFF ABI format.

- By default, the code generation tools shipped with the blockset has EABI support. If you want to use a different version of CGT ensure it has EABI support.
- All the files referred (libraries - .lib, source - .c/.h/.asm and linker command files - .cmd) should be in the same format as supported by the toolchain. Else linker errors is seen during build process.
- If your are adding any custom files, ensure that the custom files matches the toolchain format.

---

3 **Generate code only** option under **Build process** enables you to specify code generation versus an executable build. For more information, refer Generate code only (Simulink Coder)

4 You can choose the **build configuration** options to get the required compiler flags for the code compilation.

   - Faster builds
   - Faster run
   - Debug
   - Specify

   Use **Specify** option to edit column **Tool Options** to add any additional flags. For more information, refer to Build configuration (Simulink Coder)

5 The **Code Generation** > **Optimization** category includes parameters for improving the simulation speed of your models and improving the performance of the generated code. For more information, refer “Model Configuration Parameters: Code Generation Optimization” (Simulink Coder).

6 The **Code Generation** > **Report** category includes parameters for generating and customizing the code generation report. For more information, refer “Model Configuration Parameters: Code Generation Report” (Simulink Coder)

7 The **Code Generation** > **Custom Code** category includes parameters for inserting custom C code into the generated code. For more information, refer “Model Configuration Parameters: Code Generation Custom Code” (Simulink Coder).

8 The **Code Generation** > **Interface** category includes parameters for configuring the interface of the generated code and code replacement library(CRL). For more information, refer “Model Configuration Parameters: Code Generation Interface” (Simulink Coder).

9 The **Code Generation** > **Verification** category includes code verification and performance analysis parameters for SIL and PIL simulations and also configuring profiling options. Additionally you can configure the following options and perform profiling which is independent of SIL and PIL simulations:

   - Measure task execution time - by enabling this parameter in Configuration Parameters, you can perform real time execution of profiling.
   - Measure function execution times - configure to get different levels of profiling (tasks, subsystems or functions) in Configuration Parameters. The profiling logic is added accordingly. There will be additional overhead due to profiling logic on the measure time. Currently only hardware interrupts are supported for asynchronous system profiling.
     - off - measure time with respective task
     - course - tasks, reference models and subsystem only
     - detailed - tasks,reference models, subsystem, block/function internalization and step logic
   - Save options - select to save only as summary data only or all data to get detailed time line in Configuration Parameters. Metrics option is not supported for C2000.

Profiling with Build, Load and Run

- Profiling data - Run the following code in MATLAB Command Window to obtain the profiling data, report and time line.
  - Profiling data - gathers profiling data from the target hardware and stores it in a variable.

    ```
    codertarget.profile.getData('ModelName')
    ```
  - Profiling report - provides HTML report based on the profiling data.

    ```
    executionProfile.report
    ```
  - Profiling timeline - provides the timeline report of the tasks or profiled functions.

    ```
    executionProfile.timeline
    ```

    For more information, refer "Model Configuration Parameters: Code Generation Verification" (Embedded Coder) and "Real-Time Code Execution Profiling".
- **Profiling with Monitor & Tune**

  You can use the C2000 Microcontroller Blockset to profile the real-time execution of the generated code running as an executable on a Texas Instruments C2000 board with XCP on Serial and XCP on TCP/IP Interface. For more information, see "Code Execution Profiling on Texas Instruments C2000".
- **Profiling with PIL**

  You can configure the profiling options and use it with PIL mode.

Generate Code, Build and Download the Executable

After configuring hardware resource and code generation for a model, you can generate the code, build the 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 along with configuration parameters in your Simulink model.

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. The build operation will generate dependency files **(.dep)** for each of the source files **(.c)** which it will use to create the object files **(.obj)**. The object files are then linked using linker command file to create the map file **(.map)** and the executable file **(.exe)** files.

The default linker command file can be modified, or different linker command file can be provided if required using the option `Hardware Implementation > Target Hardware Resources > Build options > Use custom linker`.

Dependency Build

Subsequent builds will use dependency files to recompile the object files only if any source or header files are changed or make file is updated. The dependency build makes use of Windows® PowerShell to update the dependency files. Hence the dependency build will not be used and object files is recompiled, if Windows PowerShell is not present in the computer.

Parallel Build

The build will make use of multiple cores in host computer by enabling parallel build to compile the object files faster. This may cause issues in the order of the messages displayed in build log if there are any build failures. You can disable this option using **Hardware Implementation > Target Hardware Resources > Build options > Disable parallel build** .

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.

Select option `Boot from Flash` if the application has to load to the flash. If you do not select this option, the application loads to the RAM. **Hardware Implementation > Target Hardware Resources > Build options > Boot from Flash**.

By default, the target configuration file will be provided to support download to Control card or launchpad of the selected hardware board. If you have any custom board, you can provide your own target configuration file in the option in **Hardware Implementation > Target Hardware Resources > Build options > Select target configuration**

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.

Note Rapid Accelerator simulation is not supported by C2000 Microcontroller Blockset.

See Also

"Creating CCS Project from a Model" on page 2-13

Creating CCS Project from a Model

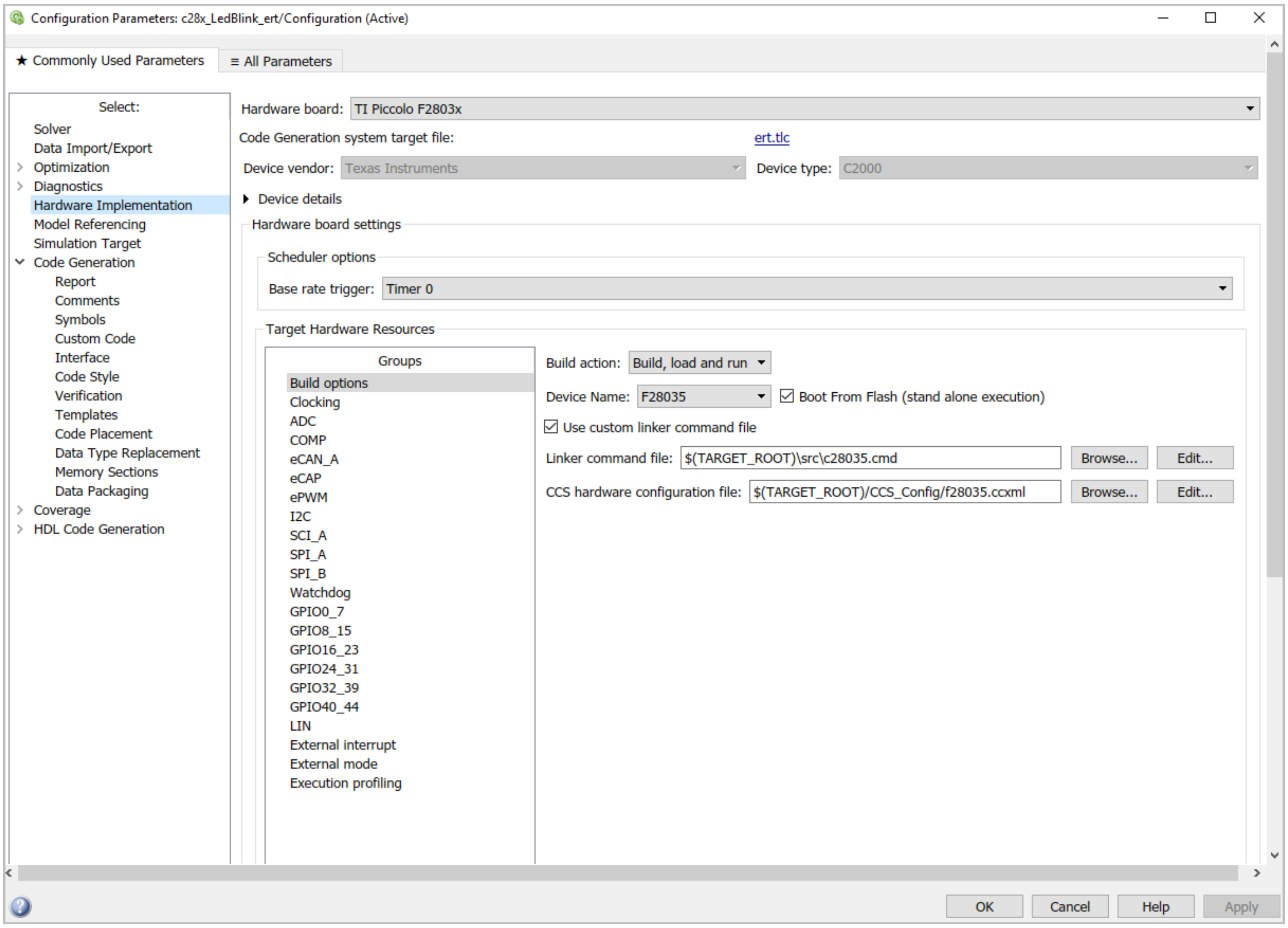
You can create a Code Composer Studio™ project from a model. To create CCS project and open the project, follow the steps given.

1 Install the C2000 Microcontroller Blockset and complete the additional setup tasks mentioned in “Hardware Setup for C2000 Microcontroller Blockset” .

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

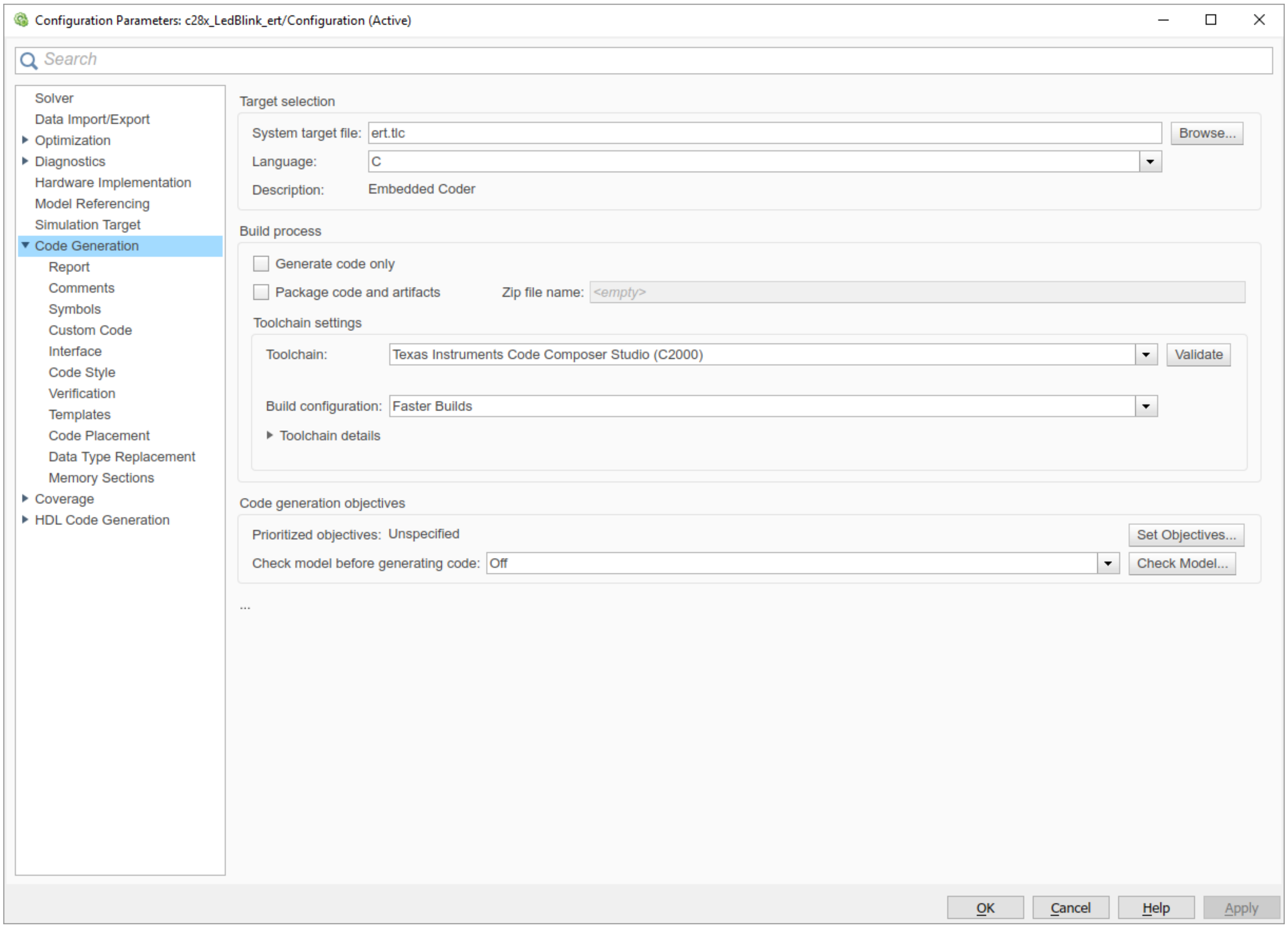
☑ Boot From Flash (stand alone execution)

☑ 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

Code generation objectives

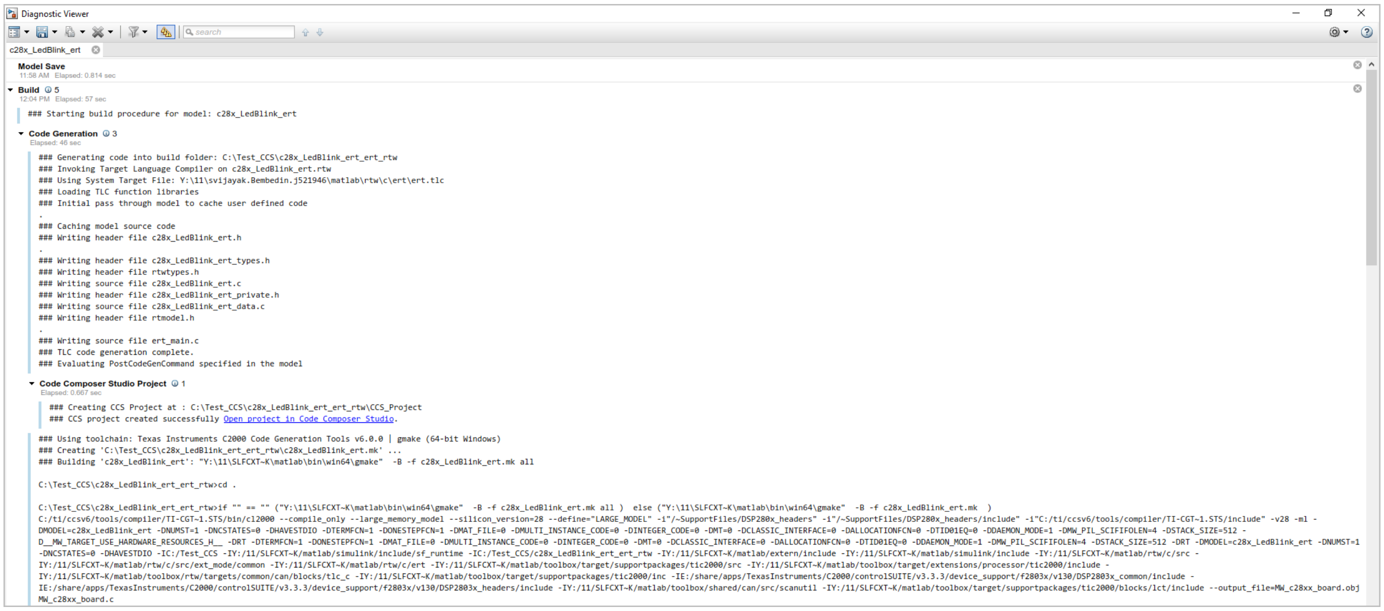
Prioritized objectives: Unspecified

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'**.

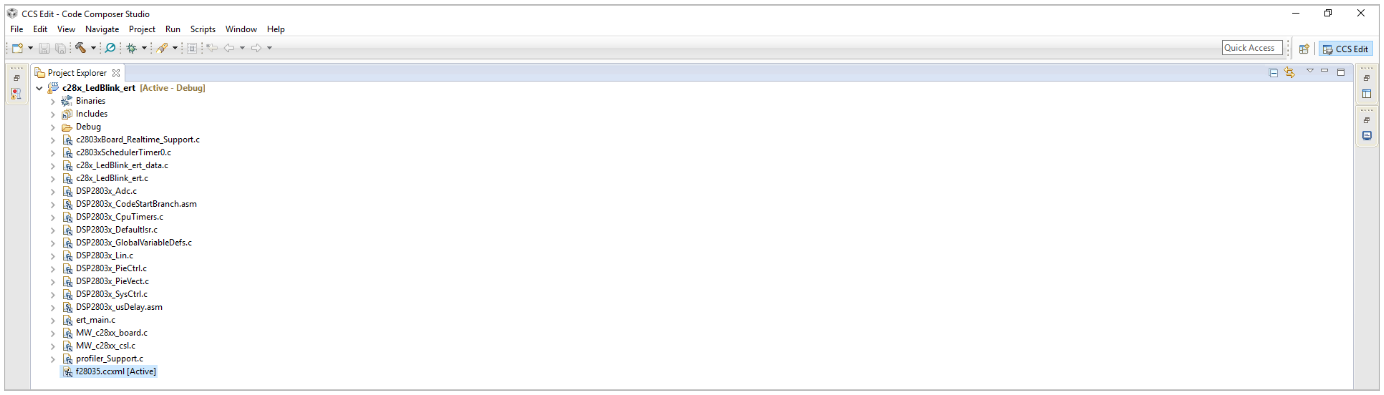


```
Model Save
11:58 AM  Elapsed: 0.814 sec

Build ⓘ 5
12:04 PM  Elapsed: 57 sec

### Starting build procedure for model: c28x_LedBlink_ert

Code Generation ⓘ 3
Elapsed: 46 sec

### 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

Code Composer Studio Project ⓘ 1
Elapsed: 0.667 sec

### 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.

See Also

“Overview of Creating a Model and Generating Executable for C2000 Processors” on page 2-3

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) when configured with COFF ABI format.
- The `double` data type is represented as double precision floating point values (64-bit) when configured with ELF ABI format.

This representation results in a difference in data values in the simulation and the generated code. For more information on ABI format selection, refer to the toolchain section in “Configuring Additional Options for Code Generation” on page 2-8

Model Reference support for C2000 Processors

A model reference is a reference to another model using a Model block. These references create model hierarchy. Each referenced model has a defined interface that specifies the properties of its inputs and outputs. The defined interface makes the behavior of the referenced model independent of its context in the model hierarchy. For simulation and code generation, a referenced model executes like a single block, or atomic unit, when the parent model executes. Model references are ideal for code reuse, unit testing, parallel builds, and large components. They can also reduce file contention and merge issues. For more information, see Model References.

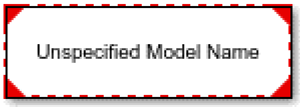
In C2000 Microcontroller Blockset, you can now use all the driver blocks from C2000 library inside model reference expect for Software and Hardware interrupt, CLA task trigger, CLA subsystem, and Idle task.

Reference Existing Models

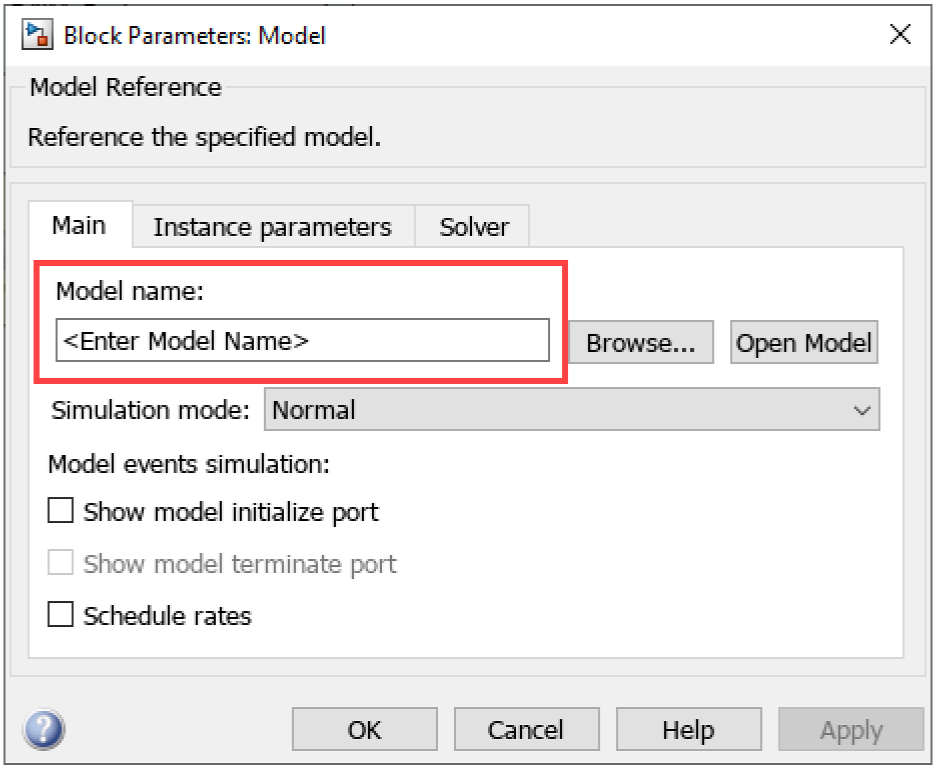
A model becomes a referenced model when a Model block in another model references it. Any model can function as a referenced model, and can continue to function as a separate model.

To reference an existing model in another model, follow these steps.

1. If the folder containing the model you want to reference is not on the MATLAB path, add the folder to the MATLAB path.
2. In the referenced model, set Total number of instances allowed per top model to:
   - `One` - to use the model at most once in a model hierarchy.
   - `Multiple` - to use the model more than once in a model hierarchy. To reduce overhead, specify Multiple only when necessary.
   - `Zero` - to preclude referencing the model.
3. Create an instance of the Model block in the parent model. The new block is initially unresolved because it does not specify a referenced model.



4. To open the Block Parameters dialog box, double-click the unresolved Model block.



Block Parameters: Model

Model Reference

Reference the specified model.

Model name: <Enter Model Name>

Simulation mode: Normal

Model events simulation:

☐ Show model initialize port

☐ Show model terminate port

☐ Schedule rates

5 Enter the name of the referenced model in the **Model name** field. This name must contain fewer than 60 characters, exclusive of the file extension.

6 Click **OK**. If the referenced model contains root-level inputs or outputs, the Model block displays corresponding input and output ports. For more, see Reference Existing Models

Things to Consider during Reference Modeling

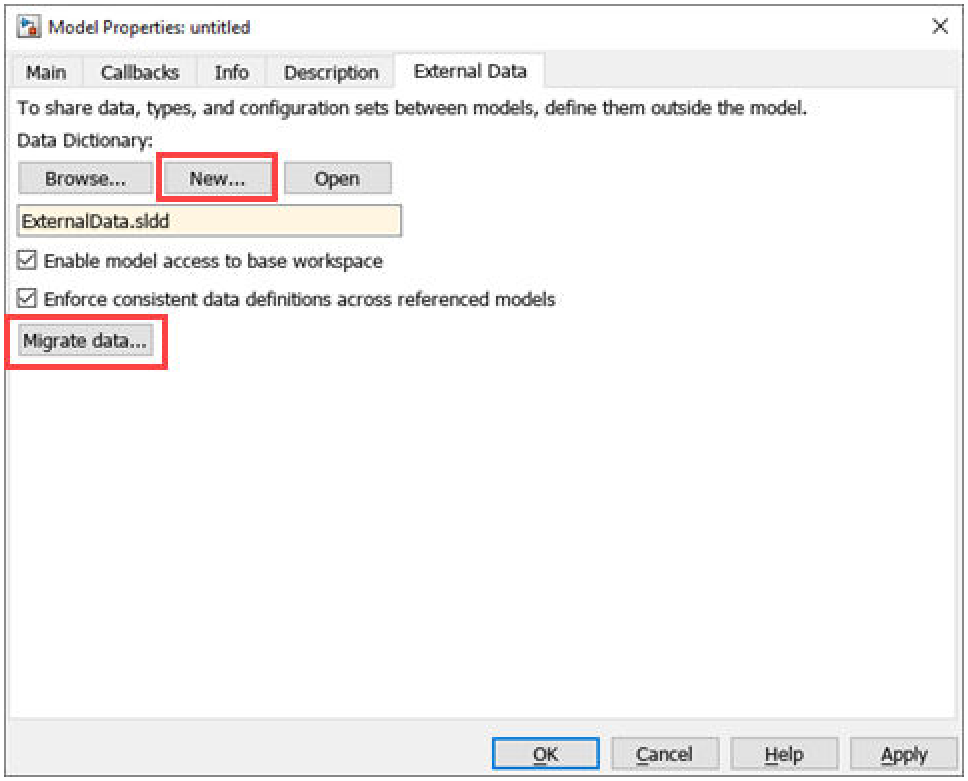
- Scheduling is considered only from the top model.
- All the configuration parameters referred by driver blocks must be same across all the models . If they are not same, then behavior is undefined as we do not have control on which configuration will be picked by the block.
- When using model reference and when function block is present in top level of child model, then you will not be able to generate the code for the child model directly.
- Total number of instances allowed per top model is `One`.
- For referenced models, go to **Configuration Parameters** > **Diagnostics** and set **Insufficient maximum identifier length** to `None`, when **maximum identifier length** does not provide enough space to make global identifiers unique across models.
- When naming the top model and reference model, if the first 31 characters of the top and reference model name matches, you will encounter an compilation error. since all 31 characters are used to name the variables in the generated code.

  So always try to have a unique names for top and reference model and try to keep the naming short.

Create Data Dictionary and Migrate the Model

This section describes how to create data dictionary to store model parameters and common configuration parameters references which can be used across parent and child models.

1. Configure your reference model with all the settings described in “Reference Existing Models” on page 2-18.
2. Open Model Properties. In the **Model Properties** dialog box, click **New** to create a data dictionary.



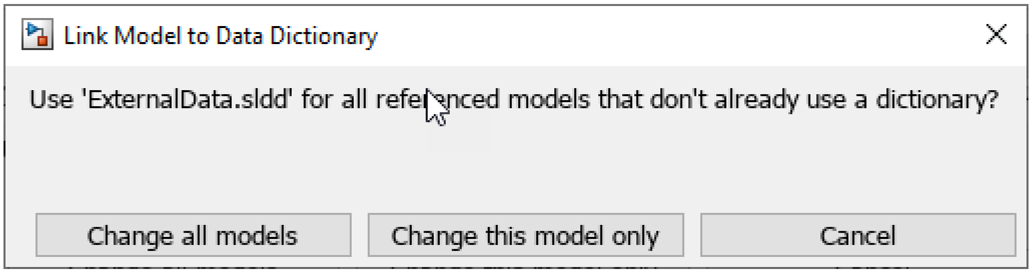
To share data, types, and configuration sets between models, define them outside the model.

Data Dictionary: ExternalData.sldd

☑ Enable model access to base workspace

☑ Enforce consistent data definitions across referenced models

3. Name the data dictionary, save it, and click **Apply**.
4. Link model to data dictionary dialog appears. Select the appropriate option.



Use 'ExternalData.sldd' for all referenced models that don't already use a dictionary?

5. Click **Migrate data**.
6. Click **Migrate** in response to the message about copying referenced variables.
7. (Optional) Clear **Enable model access to base workspace**.
8. Click **OK**.

9 To open the dictionary, in the Simulink Editor, click the model data badge in the bottom left corner, then click the **External Data** link.

To inspect the contents of the dictionary, in the Model Explorer **Model Hierarchy** pane under the **External Data** node, expand the dictionary node.

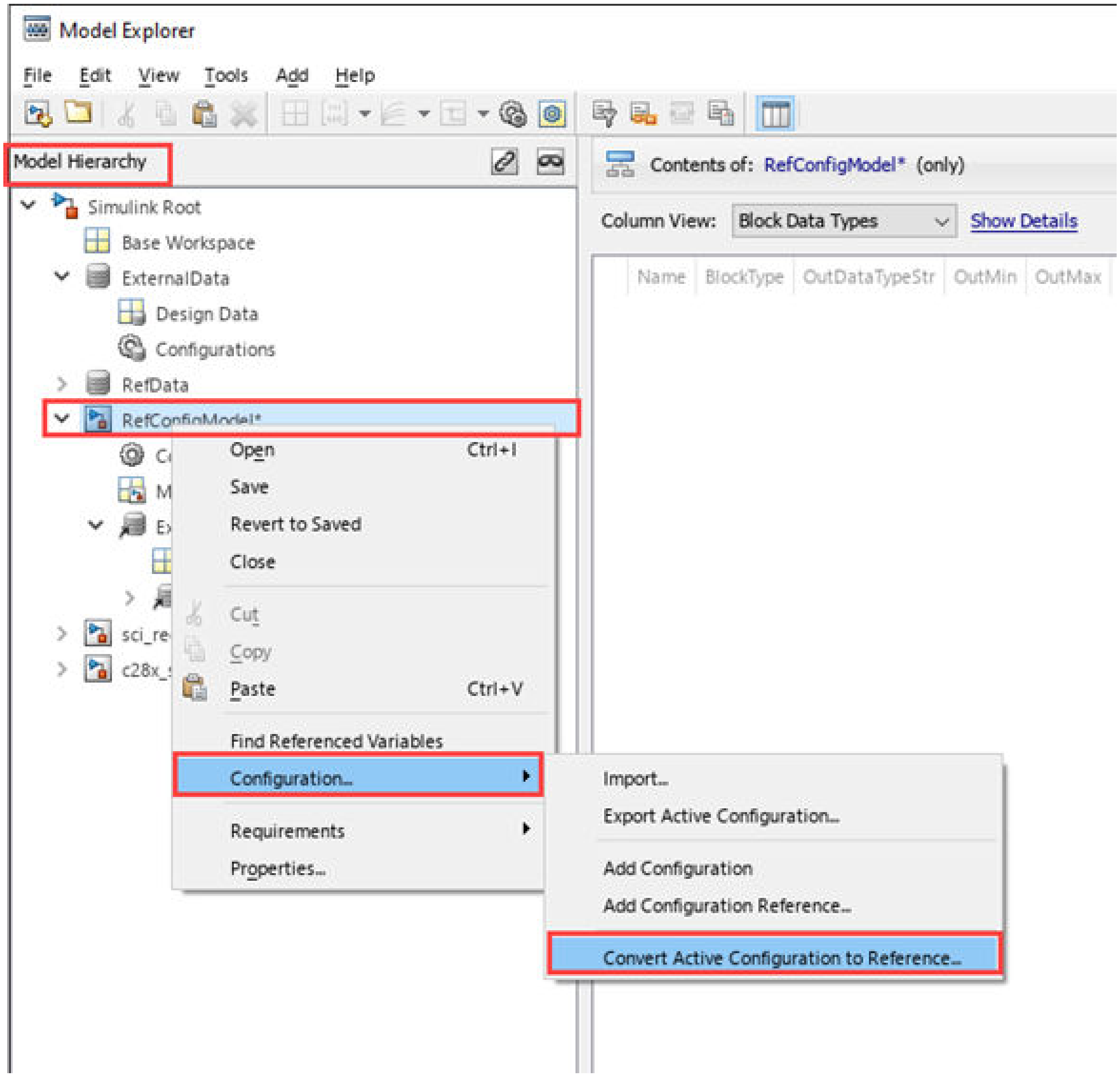
Convert Configuration Set to Configuration Reference and Include in a new model

As explained in the above section, all the configuration parameters referred by driver blocks must be same across all the models . If they are not same, then behavior is undefined as we do not have control on which configuration will be picked by the block. One way of doing this is to create configuration parameter references and refer the same across parent and child models. This ensures when ever there is change in the parameters, it is reflected across all the models.

In the top model, you must convert the active configuration set to a configuration reference:

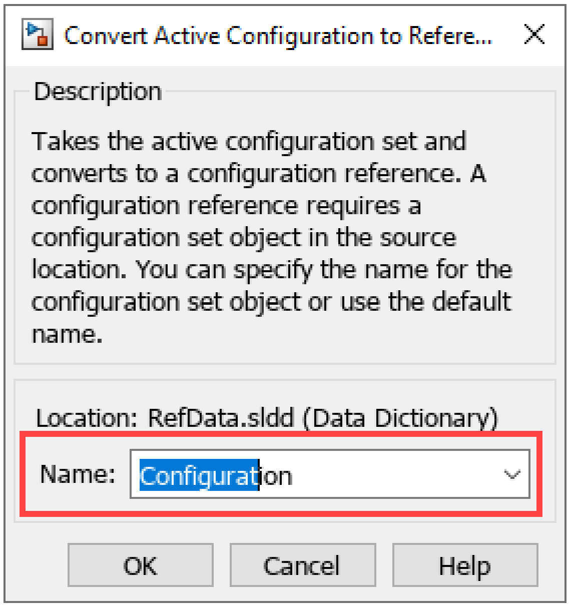
1 Open the Model Explorer.

2 Go to **Model Hierarchy** > **RefConfigModel** > **Configuration** > **Convert Active configuration to Reference**.


Model Explorer
File Edit View Tools Add Help
Model Hierarchy
Contents of: RefConfigModel* (only)
Column View: Block Data Types
Show Details
Name BlockType OutDataTypeStr OutMin OutMax
Simulink Root
Base Workspace
ExternalData
Design Data
Configurations
RefData
RefConfigModel*
sci_re
c28x_
Open Ctrl+I
Save
Revert to Saved
Close
Cut
Copy
Paste Ctrl+V
Find Referenced Variables
Configuration...
Requirements
Properties...
Import...
Export Active Configuration...
Add Configuration
Add Configuration Reference...
Convert Active Configuration to Reference...

3 To add it in the model, navigate to **Model Hierarchy** > **RefConfigModel** > **Configuration** > **Add configuration Reference** and click **Activate**.

4 Name the configuration reference. The associated sldd is selected to create a reference configuration set.



Convert Active Configuration to Refere...

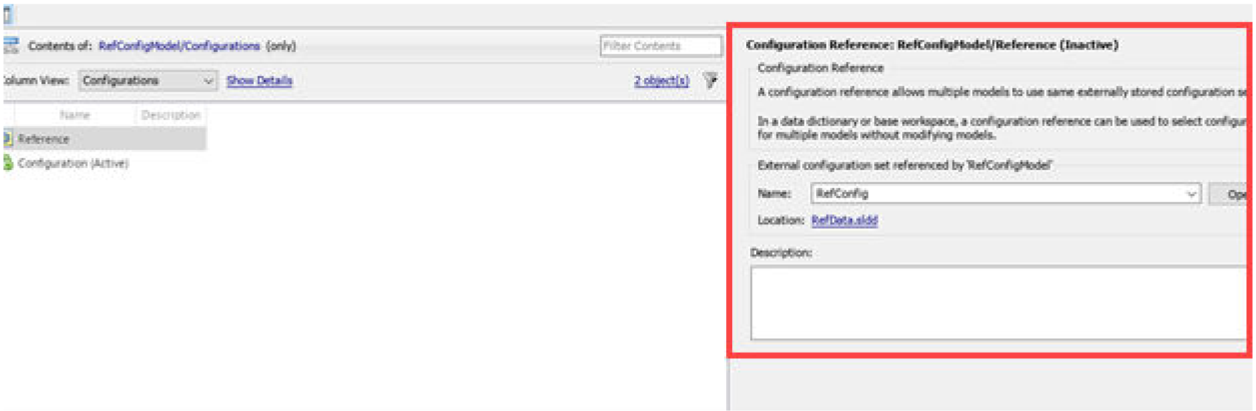
Description

Takes the active configuration set and converts to a configuration reference. A configuration reference requires a configuration set object in the source location. You can specify the name for the configuration set object or use the default name.

Location: RefData.sldd (Data Dictionary)

Name: Configuration

A configuration reference allows multiple models to use same externally stored configuration set.



Contents of: RefConfigModel/Configurations (only)

Column View: Configurations

Name	Description
Reference	
Configuration (Active)	

Configuration Reference: RefConfigModel/Reference (Inactive)

Configuration Reference

A configuration reference allows multiple models to use same externally stored configuration se

In a data dictionary or base workspace, a configuration reference can be used to select configu for multiple models without modifying models.

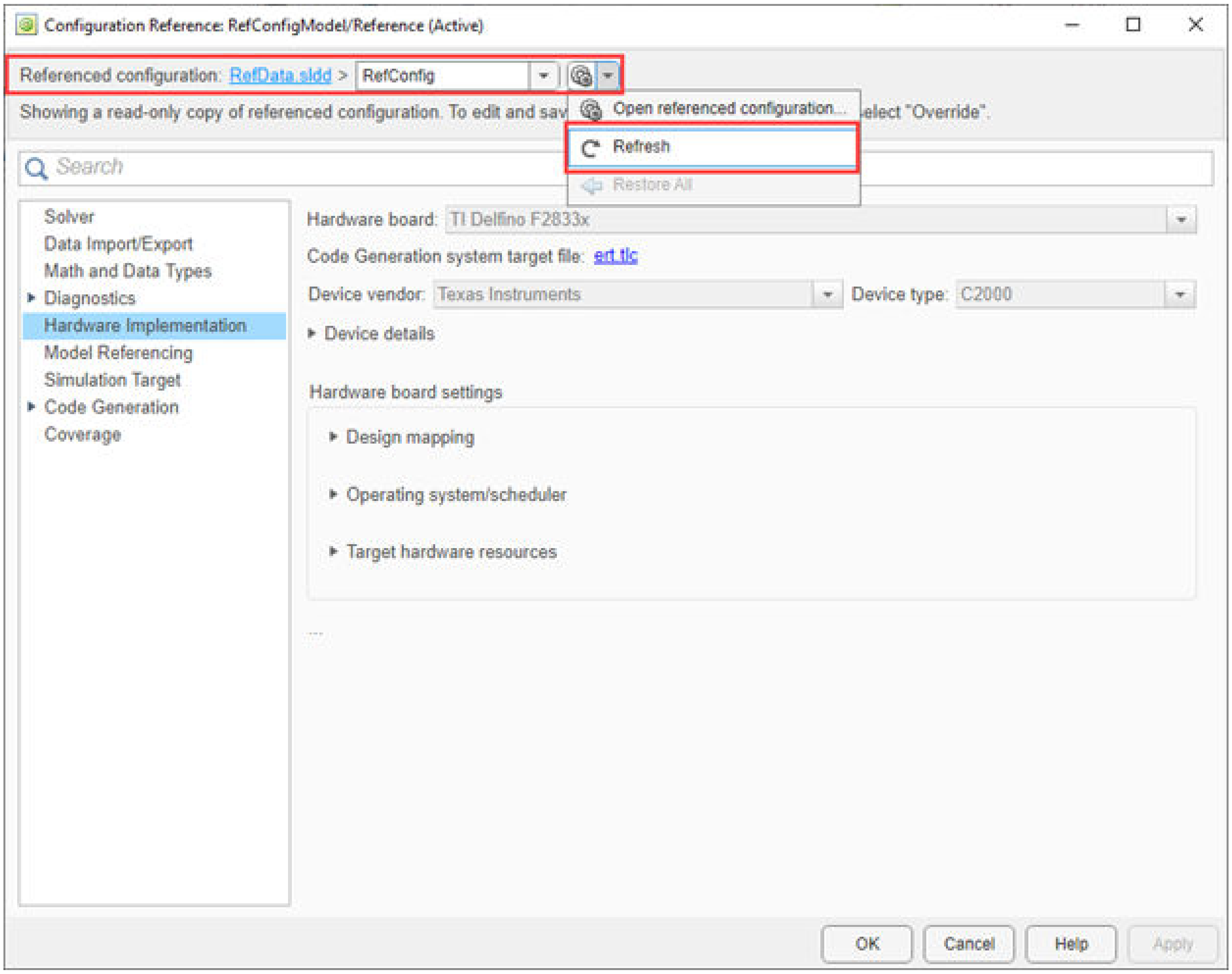
External configuration set referenced by 'RefConfigModel'

Name: RefConfig

Location: RefData.sldd

Description:

5 You can view the newly created reference configuration in Configuration Reference.


Configuration Reference: RefConfigModel/Reference (Active)
Referenced configuration: RefData.sldd > RefConfig
Showing a read-only copy of referenced configuration. To edit and sav ...elect "Override".
Open referenced configuration...
Refresh
Restore All
Search
Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
Hardware board: TI Delfino F2833x
Code Generation system target file: ert.tlc
Device vendor: Texas Instruments
Device type: C2000
Device details
Hardware board settings
Design mapping
Operating system/scheduler
Target hardware resources
OK
Cancel
Help
Apply

You can modify the reference configuration and share a Configuration with Multiple Models. For more information, see Share a Configuration with Multiple Models.

A predefined synchronous Referenced Model is available in “Serial Communication Using SCI Blocks” .

Configure Reference Model for Synchronous and Asynchronous Task

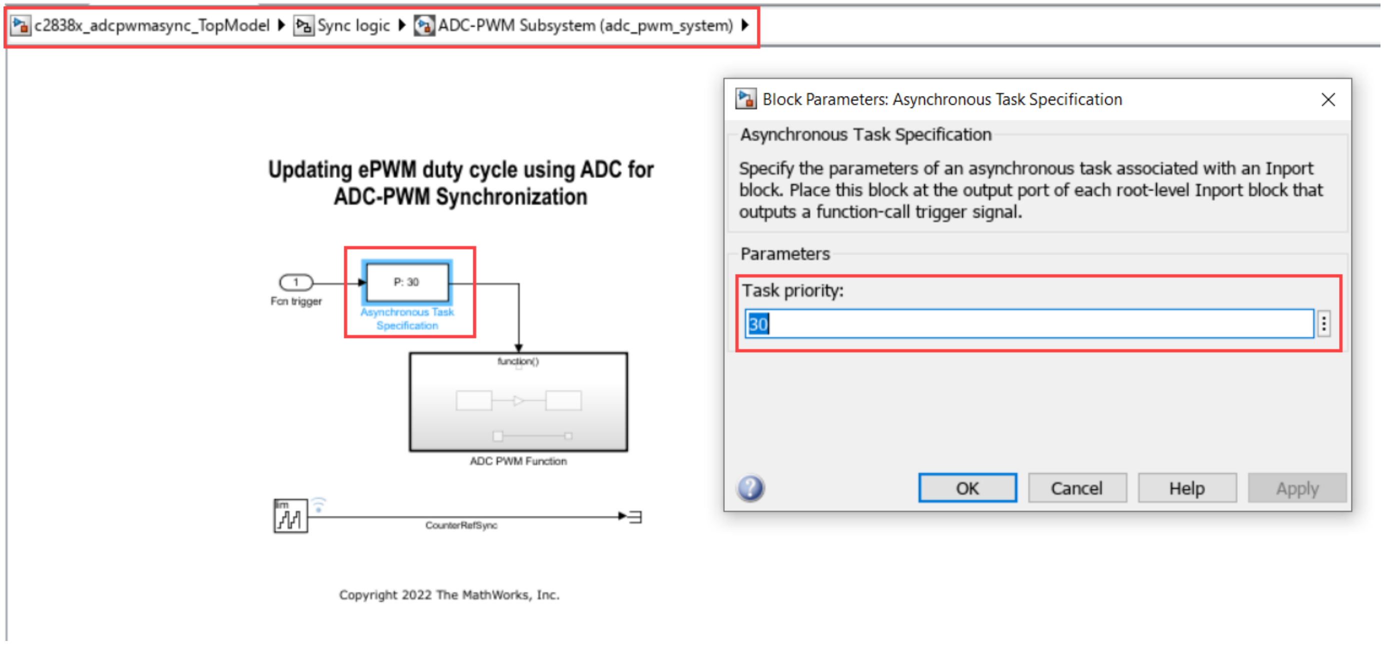
When you are working with reference models, the scheduling is considered only from the top model. When there is only synchronous task in the reference model, the sample time at which the reference model is called in the top model will determine the scheduling. However, if there are asynchronous task in the reference model, additional steps needs to be performed to trigger the same from the top model. This section explains how to configure a reference model when there is asynchronous task present in the reference model.

1 Open the top model. Enter the following command in MATLAB command prompt.

```
c2838x_adcpwmasync_TopModel
```

2 Navigate to the path in the model where you are configuring the reference model (adc_pwm_system).

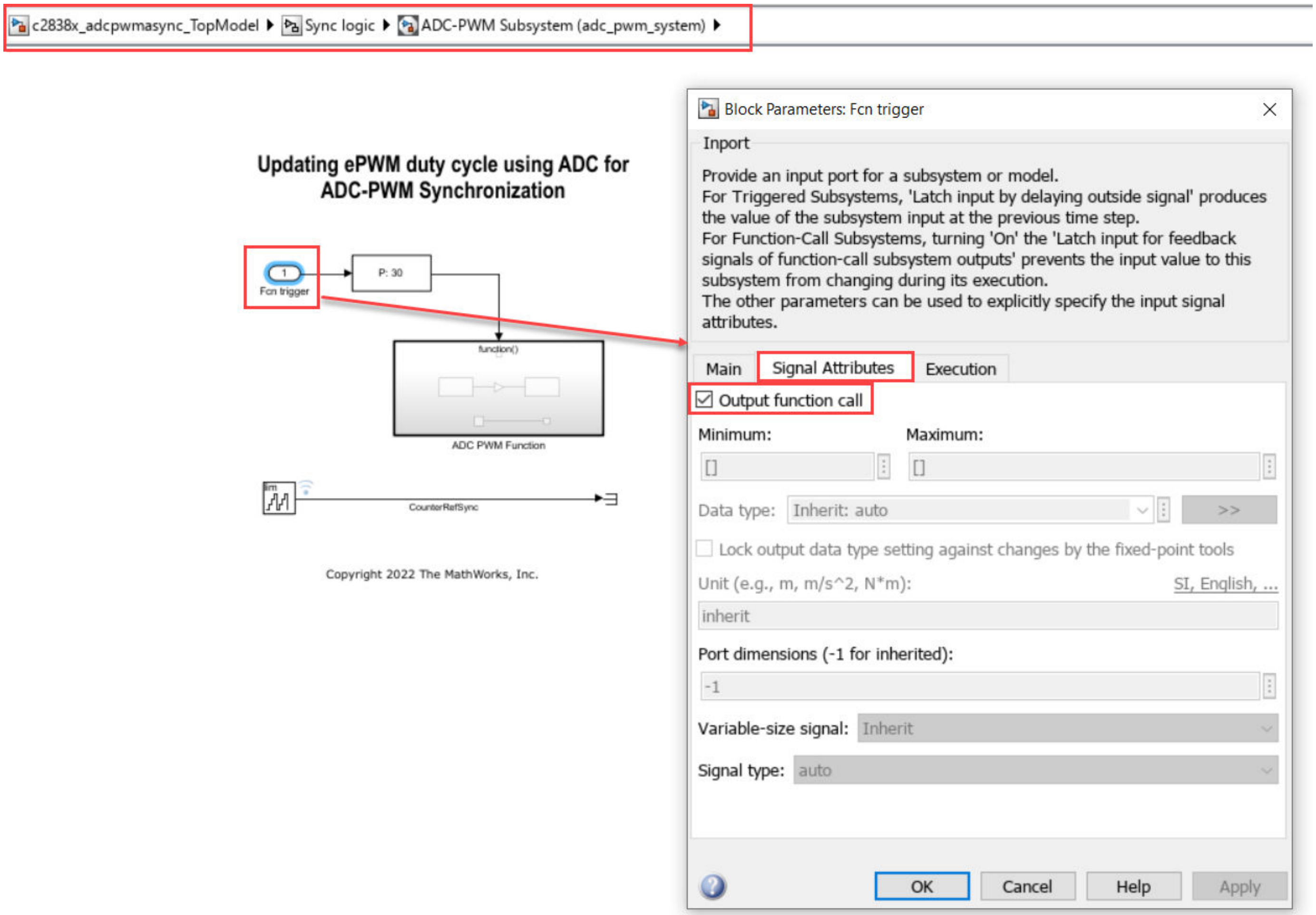
3 Verify that the asynchronous task specification block is added and it matches the task priority of the Hardware Interrupt triggering it in the top model.


c2838x_adcpwmasync_TopModel ▸ Sync logic ▸ ADC-PWM Subsystem (adc_pwm_system) ▸
Updating ePWM duty cycle using ADC for ADC-PWM Synchronization
1
Fcn trigger
P: 30
Asynchronous Task Specification
function()
ADC PWM Function
CounterRefSync
Copyright 2022 The MathWorks, Inc.
Block Parameters: Asynchronous Task Specification
Asynchronous Task Specification
Parameters
OK
Cancel
Help
Apply

Specify the parameters of an asynchronous task associated with an Inport block. Place this block at the output port of each root-level Inport block that outputs a function-call trigger signal.

Task priority: 30

4 Verify that the inport triggering the asynchronous task has **Output function call** enabled.


c2838x_adcpwmasync_TopModel ▸ Sync logic ▸ ADC-PWM Subsystem (adc_pwm_system) ▸
Updating ePWM duty cycle using ADC for ADC-PWM Synchronization
1
Fcn trigger
P: 30
function()
ADC PWM Function
CounterRefSync
Copyright 2022 The MathWorks, Inc.
Block Parameters: Fcn trigger
Main
Signal Attributes
Execution
>>
SI, English, ...
OK
Cancel
Help
Apply

Inport

Provide an input port for a subsystem or model.
For Triggered Subsystems, 'Latch input by delaying outside signal' produces the value of the subsystem input at the previous time step.
For Function-Call Subsystems, turning 'On' the 'Latch input for feedback signals of function-call subsystem outputs' prevents the input value to this subsystem from changing during its execution.
The other parameters can be used to explicitly specify the input signal attributes.

☑ Output function call

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

Port dimensions (-1 for inherited): -1

Variable-size signal: Inherit

Signal type: auto

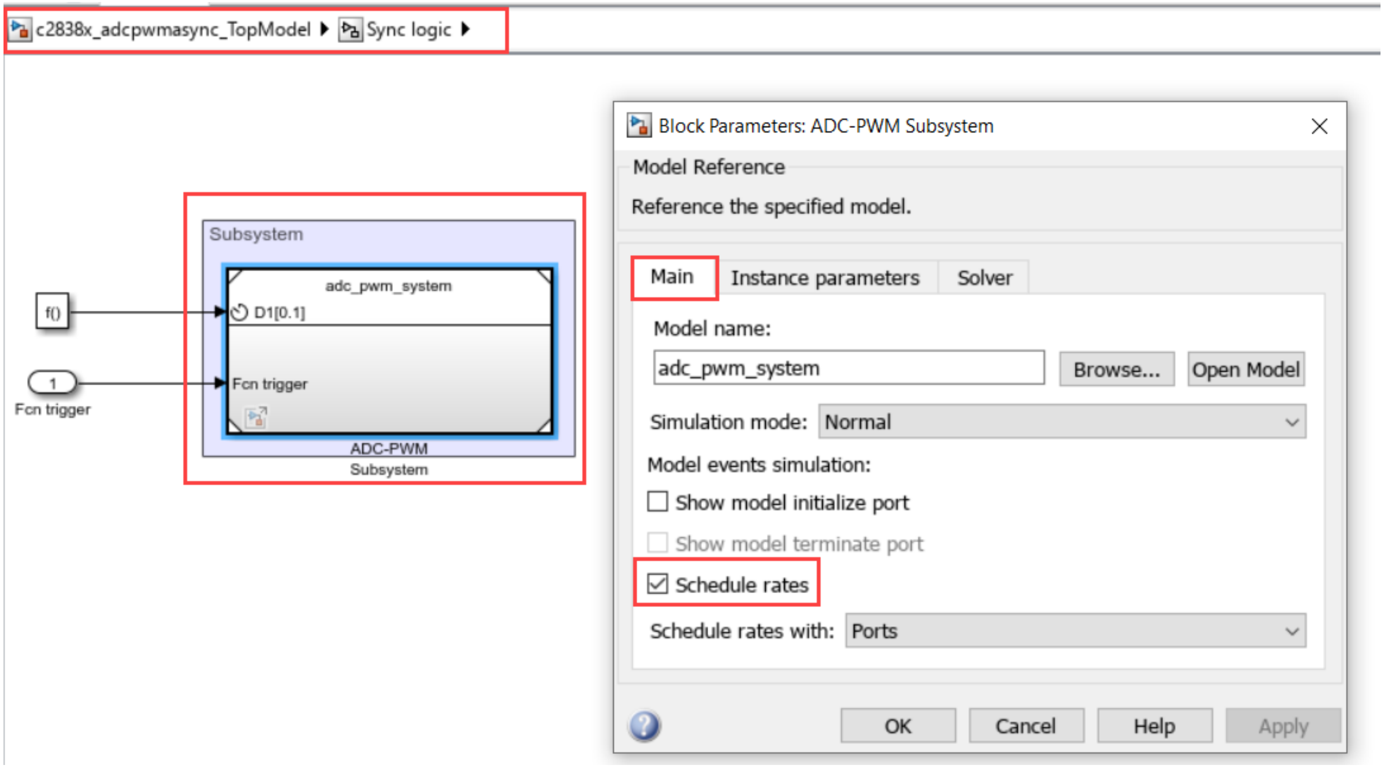
5 Ensure the periodic rate in the reference model matches the top model.

6 Now this reference model contains both synchronous and asynchronous task.

7 Schedule the model through base rate for synchronous task (periodic) and interrupt for asynchronous task.

8 In the top model, right click on **ADC-PWM Subsystem** and select **Block parameter (ModelReference)**. Verify **Schedule rates** parameter is enabled.

This options grows an inport with a scheduling icon and this can be configured to be triggered by a periodic task (in this example, function-call generator).



Block Parameters: ADC-PWM Subsystem

Model Reference

Reference the specified model.

Model name: adc_pwm_system

Simulation mode: Normal

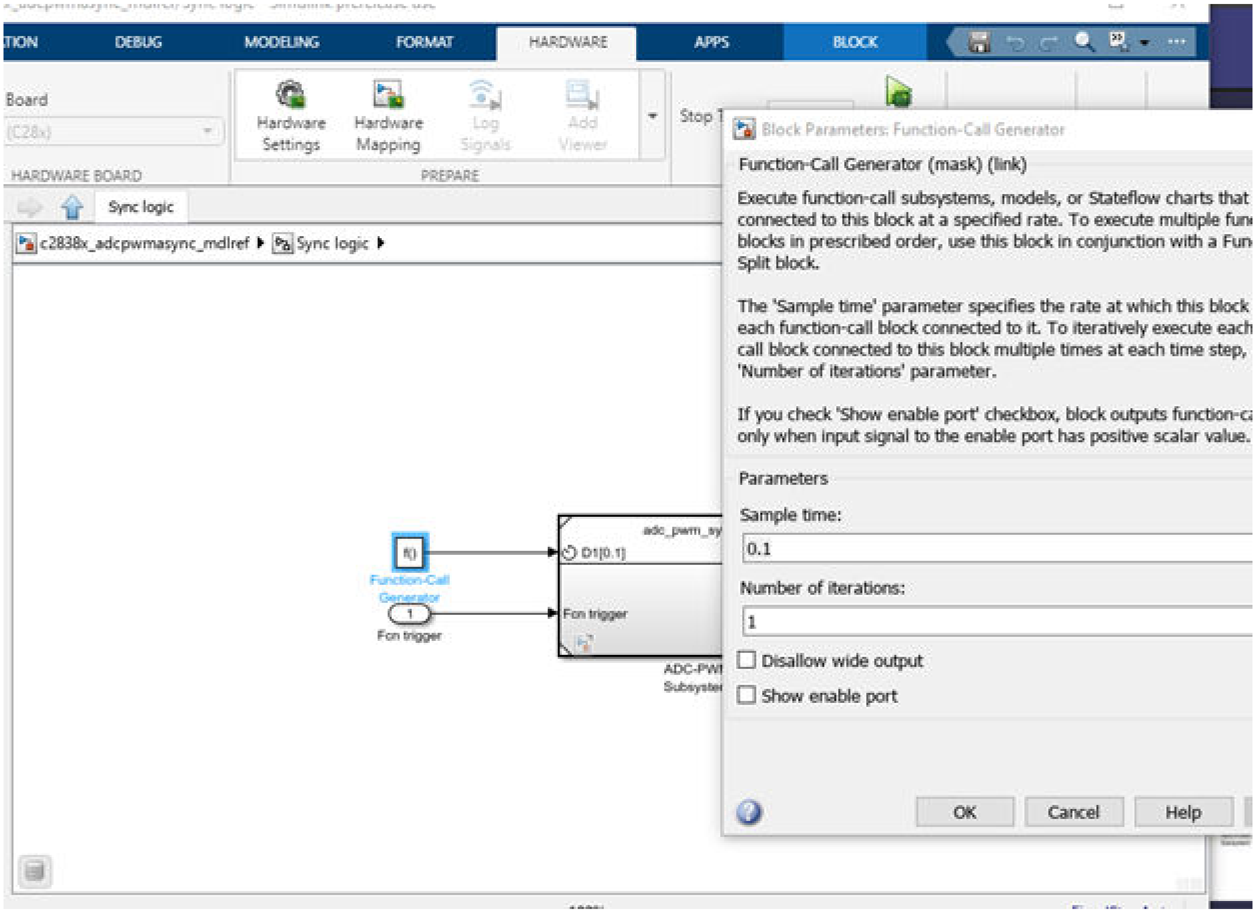
Model events simulation:

☐ Show model initialize port

☐ Show model terminate port

☑ Schedule rates

Schedule rates with: Ports



Block Parameters: Function-Call Generator

Function-Call Generator (mask) (link)

Execute function-call subsystems, models, or Stateflow charts that connected to this block at a specified rate. To execute multiple fun blocks in prescribed order, use this block in conjunction with a Fun Split block.

The 'Sample time' parameter specifies the rate at which this block each function-call block connected to it. To iteratively execute each call block connected to this block multiple times at each time step, 'Number of iterations' parameter.

If you check 'Show enable port' checkbox, block outputs function-ca only when input signal to the enable port has positive scalar value.

Parameters

Sample time:

0.1

Number of iterations:

1

□ Disallow wide output

□ Show enable port

9 The output trigger from Hardware Interrupt block, is passed as an input to the previously configured asynchronous port in the reference model. With these configurations, the reference model with both synchronous and asynchronous task can be triggered from the top model.

Note With the asynchronous task specification ,you will not be able to generate code independently from the reference model.

See Also

Model References | Reference Existing Models | Share a Configuration with Multiple Models

Related Examples

- “ADC-PWM Synchronization Using ADC Interrupt”
- “Serial Communication Using SCI Blocks”

Supported Third-Party Tools for Texas Instruments C2000 Processors

CCS versions supported for MATLAB (Support package until R2022b)

MATLAB Version	CCS Supported Version	Verified CCS Version	Verified C2000Ware	Verified Code Generation Tools (CGT)
MATLAB 9.2 (R2017b)	CCSv3 to CCSv7	Code Composer Studio v7.2	C2000Ware 1.00.01.00	CGT 16.9.2
MATLAB 9.4 (R2018a)				
MATLAB 9.5 (R2018b)	CCSv3 to CCSv8	Code Composer Studio Studio v8	C2000Ware 1.00.05.00	CGT 16.9.2
MATLAB 9.6 (R2019a)	CCSv3 to CCSv8	Code Composer Studio v8.2	C2000Ware 1.00.06.00	CGT 16.9.9
MATLAB 9.7 (R2019b)	CCSv3 to CCSv9	Code Composer Studio v9.1	C2000Ware 2.00.00.02	CGT 18.12.2
MATLAB 9.8 (R2020a)				
MATLAB 9.9 (R2020b)				
MATLAB 9.10 (R2021a)	CCSv3 to CCSv10	Code Composer Studio v10.1	C2000Ware 3.2.0.0	TI C28x™ CGT 20.2.1 and TI ARM CGT 20.2.1
MATLAB 9.11 (R2021b)	CCSv3 to CCSv10	Code Composer Studio v10.2	C2000Ware 3.3.0.0	TI C28x CGT 20.2.1 and TI ARM CGT 20.2.1
MATLAB 9.12 (R2022a)	CCSv3 to CCSv11	Code Composer Studio v11.0	C2000Ware 4.0.0.0	TI C28x CGT 20.2.1 and TI ARM CGT 20.2.1
MATLAB 9.13 (R2022b)	CCSv3 to CCSv11	Code Composer Studio v11.0	C2000Ware 4.0.0.0	TI C28x CGT 20.2.1 and TI ARM CGT 20.2.1
MATLAB 9.14 (R2023a)	CCSv3 to CCSv11	Code Composer Studio v11.0	C2000Ware 4.0.0.0	TI C28x CGT 20.2.1 and TI ARM CGT 20.2.1

CCS versions supported for MATLAB (C2000 Microcontroller Blockset starting R2023a)

MATLAB Version	CCS Supported Version	Verified CCS Version	Verified C2000Ware	Verified Code Generation Tools (CGT)
MATLAB 9.14 (R2023a)	CCSv3 to CCSv11	Code Composer Studio v11.0	C2000Ware 4.0.0.0	TI C28x CGT 20.2.1 and TI ARM CGT 20.2.1

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

- The C2000 blockset supports CCS v3.3 and the later versions. However, CCS v3.3 does not support auto-download feature.
- Install the Code Composer Studio (CCS) version that supports your hardware board. For example, install CCS v6 or later versions to work on Texas Instruments C2000 F2807x, Texas Instruments C2000 F2837xD, and Texas Instruments C2000 F2837xS processors.
- Install CCS v9 and later versions to work on Texas Instruments C2000 F2838x. CCS v11 and later versions to work on Texas Instruments C2000 F28003x

See Also

Related Examples

- “Creating CCS Project from a Model” on page 2-13