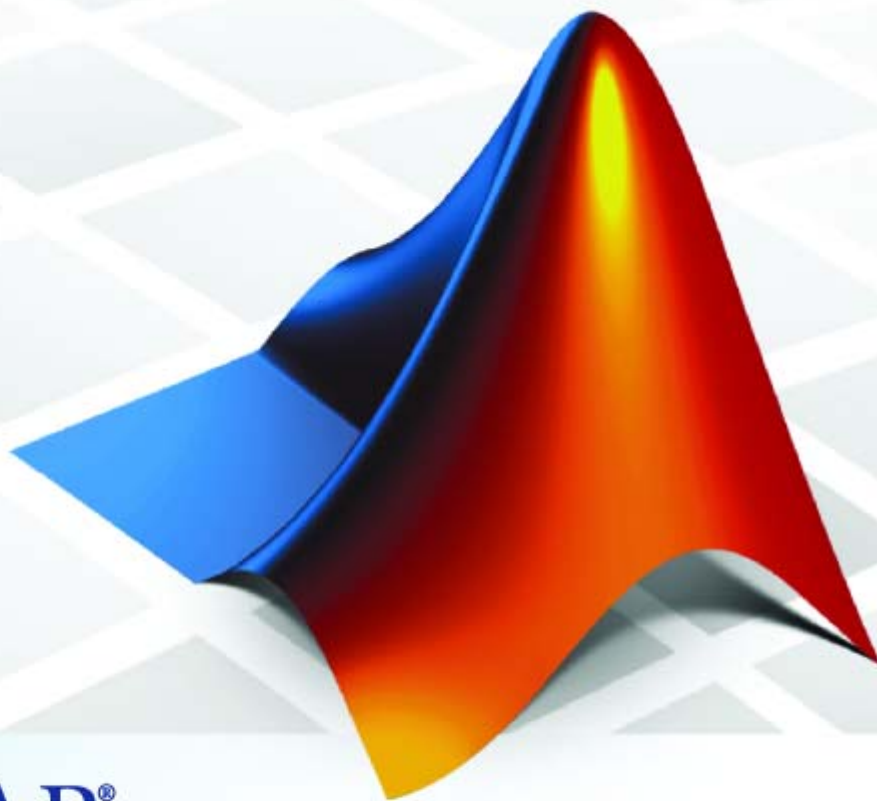


# Target for TI C2000™ 2

## User's Guide



**MATLAB®**  
& **SIMULINK®**

## How to Contact The MathWorks



[www.mathworks.com](http://www.mathworks.com)  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab)  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html)

Web  
Newsgroup  
Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com)  
[bugs@mathworks.com](mailto:bugs@mathworks.com)  
[doc@mathworks.com](mailto:doc@mathworks.com)  
[service@mathworks.com](mailto:service@mathworks.com)  
[info@mathworks.com](mailto:info@mathworks.com)

Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

For contact information about worldwide offices, see the MathWorks Web site.

*Target for TI C2000 User's Guide*

© COPYRIGHT 2003–2007 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, SimBiology, SimHydraulics, SimEvents, and xPC TargetBox are registered trademarks and The MathWorks, the L-shaped membrane logo, Embedded MATLAB, and PolySpace are trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

### Patents

The MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

**Revision History**

November 2003 Online only  
June 2003 Online only  
October 2004 Online only  
December 2004 Online only  
March 2005 Online only  
September 2005 Online only  
March 2006 Online only  
September 2006 Online only  
March 2007 Online only  
September 2007 Online only

New for Version 1.0 (Release 13SP1+)  
New for Version 1.1 (Release 14)  
Revised for Version 1.1.1 (Release 14SP1)  
Revised for Version 1.2 (Release 14SP1+)  
Revised for Version 1.2.1 (Release 14SP2)  
Revised for Version 1.3 (Release 14SP3)  
Revised for Version 2.0 (Release 2006a)  
Revised for Version 2.1 (Release 2006b)  
Revised for Version 2.2 (Release 2007a)  
Revised for Version 2.3 (Release 2007b)



## Getting Started

# 1

<b>What Is Target for TI C2000?</b> .....	<b>1-2</b>
Introduction .....	1-2
Overview of Target for TI C2000 .....	1-2
Suitable Applications .....	1-2
<b>Setting Up and Configuring</b> .....	<b>1-4</b>
Platform Requirements — Hardware and Operating System .....	1-4
Supported Hardware for Targets .....	1-4
Software Requirements .....	1-6
Verifying the Configuration .....	1-7
<b>Target for TI C2000 and Code Composer Studio</b> .....	<b>1-10</b>
Using Code Composer Studio with Target for TI C2000 ...	1-10
Default Project Configuration .....	1-10
<b>Data Type Support</b> .....	<b>1-12</b>
<b>Scheduling and Timing</b> .....	<b>1-13</b>
Overview .....	1-13
Timer-Based Interrupt Processing .....	1-13
Asynchronous Interrupt Processing .....	1-14
<b>Overview of Creating Models for Targeting</b> .....	<b>1-19</b>
Accessing the Target for TI C2000 Block Library .....	1-19
Online Help .....	1-20
Blocks with Restrictions .....	1-20
S-Function Builder Blocks .....	1-22
Setting Simulation Configuration Parameters .....	1-22
Building Your Model .....	1-23
<b>Using the c2000lib Blockset</b> .....	<b>1-25</b>
Introduction .....	1-25

Hardware Setup .....	1-25
Starting the c2000lib Library .....	1-26
Setting Up the Model .....	1-27
Adding Blocks to the Model .....	1-31
Generating Code from the Model .....	1-33

## Configuring Timing Parameters for CAN Blocks

### 2

<b>Blocks Where the Bit Rate Cannot Be Set Directly</b> ....	2-2
<b>Setting Timing Parameters</b> .....	2-3
Accessing the Timing Parameters .....	2-3
Equations for Bit Rate Calculation .....	2-5
CAN Bit Timing Examples .....	2-7
<b>Parameter Tuning and Signal Logging</b> .....	2-9
Overview .....	2-9
Using External Mode .....	2-9
Using a Third Party Calibration Tool .....	2-18

## Configuring Acquisition Window Width for ADC Blocks

### 3

<b>What Is an Acquisition Window?</b> .....	3-2
<b>Configuring ADC Parameters for Acquisition Window Width</b> .....	3-5
Accessing the ADC Parameters .....	3-5
Examples .....	3-7

## Creating Stand-Alone Applications by Saving Code into Flash Memory

### 4

<b>The Need for Stand-Alone Applications</b> .....	<b>4-2</b>
<b>Generating Code for Flash Memory</b> .....	<b>4-3</b>
<b>Running Code from Flash Memory</b> .....	<b>4-4</b>

## Using the IQmath Library

### 5

<b>About the IQmath Library</b> .....	<b>5-2</b>
Introduction .....	<b>5-2</b>
Common Characteristics .....	<b>5-3</b>
<b>Fixed-Point Numbers</b> .....	<b>5-4</b>
Notation .....	<b>5-4</b>
Signed Fixed-Point Numbers .....	<b>5-5</b>
Q Format Notation .....	<b>5-5</b>
<b>Building Models</b> .....	<b>5-9</b>
Overview .....	<b>5-9</b>
Converting Data Types .....	<b>5-9</b>
Using Sources and Sinks .....	<b>5-10</b>
Choosing Blocks to Optimize Code .....	<b>5-10</b>

## Blocks — By Category

### 6

<b>C2000 Target Preferences (c2000tgtpreflib)</b> .....	<b>6-2</b>
<b>Host-Side CAN Blocks (c2000canlib)</b> .....	<b>6-3</b>

<b>Host-Side SCI Blocks (c2000scilib) .....</b>	<b>6-4</b>
<b>C2000 RTDX Instrumentation (rtDXBlocks) .....</b>	<b>6-5</b>
<b>C280x DSP Chip Support (c280xdspchiplib) .....</b>	<b>6-6</b>
<b>C281x DSP Chip Support (c281xdspchiplib) .....</b>	<b>6-8</b>
<b>C28x Digital Motor Control (c28xdmclib) .....</b>	<b>6-10</b>
<b>C28x IQmath (tiiqmathlib) .....</b>	<b>6-11</b>

## Blocks — Alphabetical List

**7**

**Index**



# Getting Started

---

What Is Target for TI C2000? (p. 1-2)	Introduces Target for TI C2000 and describes some of its features and supported hardware
Setting Up and Configuring (p. 1-4)	Describes the software and hardware required to use Target for TI C2000™ and how to set them up
Target for TI C2000 and Code Composer Studio (p. 1-10)	Provides information about Code Composer Studio™
Data Type Support (p. 1-12)	Compares the data types supported by Simulink and the TI C2000 processors
Scheduling and Timing (p. 1-13)	Provides information about TI C2000 scheduling
Overview of Creating Models for Targeting (p. 1-19)	Summarizes the steps required to create models for your target
Using the c2000lib Blockset (p. 1-25)	Provides an example of creating a model and targeting hardware

## What Is Target for TI C2000?

In this section...
“Introduction” on page 1-2
“Overview of Target for TI C2000” on page 1-2
“Suitable Applications” on page 1-2

### Introduction

This chapter describes how to use Target for TI C2000™ to create and execute applications on Texas Instruments C2000 development boards. To use the targeting software, you should be familiar with using Simulink® to create models and with the basic concepts of Real-Time Workshop® automatic code generation. To read more about Real-Time Workshop, refer to the “Real-Time Workshop” documentation.

### Overview of Target for TI C2000

Target for TI C2000 integrates Simulink and MATLAB® with Texas Instruments eXpressDSP™ tools. You can use this product to develop and validate digital signal processing and control designs from concept through code.

Target for TI C2000 uses C code generated by Real-Time Workshop® and your TI development tools to generate a C language real-time implementation of your Simulink model. Real-Time Workshop builds a Code Composer Studio™ project from the C code.

You can compile, link, download, and execute the generated code on an F2808 or F2812 eZdsp™ DSP board from Spectrum Digital, Inc. or on a custom board based on a TI C280x or C281x chip.

### Suitable Applications

The Target for TI C2000 enables you to develop digital signal processing and control applications. Some important characteristics of the applications that you can develop are

- Asynchronous scheduling
- Flash-based standalone applications
- Fixed-point arithmetic
- Single rate
- Multirate
- Adaptive
- Frame based

## Setting Up and Configuring

In this section...
“Platform Requirements — Hardware and Operating System” on page 1-4
“Supported Hardware for Targets” on page 1-4
“Software Requirements” on page 1-6
“Verifying the Configuration” on page 1-7

### Platform Requirements — Hardware and Operating System

To run Target for TI C2000, your host PC must meet the following hardware configuration requirements:

- Intel Pentium or Intel Pentium processor-compatible PC
- One parallel printer port or one USB port to connect your target board to your PC
- DVD drive
- Windows 2000 or Windows XP

You may need additional hardware, such as signal sources and generators, oscilloscopes or signal display systems, and assorted cables to test and evaluate your application on your hardware.

### Supported Hardware for Targets

Target for TI C2000 supports the following boards:

- DSP Starter Kits (DSK) from Spectrum Digital, Inc.
  - TMS320F2812 eZdsp DSK — The F2812eZdsp DSP Starter Kit
  - TMS320F2808 eZdsp DSK — The F2808eZdsp DSP Starter Kit

The above DSKs help developers evaluate digital signal processing applications for the Texas Instruments DSP chips. You can create, test, and deploy your processing software and algorithms on the target processor

without the difficulties inherent in starting with the digital signal processor itself and building the support hardware to test the application on the processor. Instead, the development board provides the input hardware, output hardware, timing circuitry, memory, and power for the digital signal processor. Texas Instruments provides the software tools, such as the C compiler, linker, assembler, and integrated development environment, for PC users to develop, download, and test their algorithms and applications on the processor.

Refer to the documentation provided with your hardware for information on setting up and testing your target board.

---

**Note** To generate code, and download the code to your target board, you do not need to change any jumpers from their factory defaults on the F2812 target board.

However, if you want to run your code from flash memory on the F2808 or F2812, you do need to change settings on the board. For more information on this, see “Creating Stand-Alone Applications by Saving Code into Flash Memory”.

---

---

**Note** In factory default condition, the F2812 target board is set to operate in microcontroller mode. Target for TI C2000 does not support microprocessor mode.

---

- Custom boards based on any of the following Texas Instruments C2000 Digital Signal Controllers:
  - TMS320F2801
  - TMS320F2802
  - TMS320F2806
  - TMS320F2808
  - TMS320F2809
  - TMS320C2810

- TMS320F2810
- TMS320C2811
- TMS320F2811
- TMS320R2811
- TMS320C2812
- TMS320F2812
- F28015
- F28016
- F28044

## **Running Code from Flash Memory**

Running code from flash memory is supported on both the F2808 and F2812 eZdsp DSKs. Although you can generate and download code to the F2808 or F2812 eZdsp DSK with the board in factory default condition, you need to change hardware settings on the board before you can run code from flash memory. For more information, refer to “Creating Stand-Alone Applications by Saving Code into Flash Memory”

## **Software Requirements**

### **MathWorks Software**

For information about other MathWorks software required to use Target for TI C2000, refer to the MathWorks Web site — <http://www.mathworks.com>. Check the Products area for Target for TI C2000.

For information about the software required to use Link for Code Composer Studio Development Tools, refer to the Products area of the MathWorks Web site — <http://www.mathworks.com>.

### **Texas Instruments Software**

In addition to the required software from The MathWorks™, Target for TI C2000 requires that you install the Texas Instruments development tools and

software listed in the following table. Installing Code Composer Studio IDE for the C2000 series installs the software shown.

### Required TI Software for Targeting Your TI C2000 Hardware

Installed Product	Additional Information
Assembler	Creates object code (.obj) for C2000 boards from assembly code.
Compiler	Compiles C code from the blocks in Simulink models into object code (.obj). As a by-product of the compilation process, you get assembly code (.asm) as well.
Linker	Combines various input files, such as object files and libraries.
Code Composer Studio	Texas Instruments integrated development environment (IDE) that provides code debugging and development tools.
TI C2000 miscellaneous utilities	Various tools for developing applications for the C2000 digital signal processor family.
Code Composer Setup Utility	Program you use to configure your CCS installation by selecting your target boards or simulator.
Flash Plug-In	Plug-in you use in downloading generated code to flash memory. While this plug-in is not strictly required, it is very useful when working with flash memory. It is available through the CCS Web Update.

### Verifying the Configuration

To determine whether Target for TI C2000 is installed on your system, enter this command at the MATLAB prompt:

```
c2000lib
```

MATLAB displays the C2000 block library containing the following libraries and blocks that comprise the C2000 library:

- RTDX Instrumentation
- C2000 Target Preferences
- Host-side CAN Blocks
- Host-side SCI Blocks
- C281x DSP Chip Support
- C280x DSP Chip Support
- C28x IQMath Library
- C28x DMC Library

If you do not see the listed libraries, or MATLAB does not recognize the command, you need to install Target for TI C2000. Without the software, you cannot use Simulink and Real-Time Workshop to develop applications targeted to the TI boards.

---

**Note** For information about system requirements, refer to the system requirements page, available in the Products area at the MathWorks Web site (<http://www.mathworks.com>).

---

To verify that Code Composer Studio (CCS) is installed on your machine, enter this command at the MATLAB prompt:

```
ccsboardinfo
```

With CCS installed and configured, MATLAB returns information about the boards that CCS recognizes on your machine, in a form similar to the following listing:

Board Num	Board Name	Proc Num	Processor Name	Processor Type
1	F2812 Simulator	0	CPU	TMS320C28xx
0	F2812 PP Emulator	0	CPU_1	TMS320C28xx



---

If MATLAB does not return information about any boards, revisit your CCS installation and setup in your CCS documentation.

As a final test, launch CCS to ensure that it starts up successfully. For Target for TI C2000 to operate with CCS, the CCS IDE must be able to run on its own.

---

**Note** For any model to work in the targeting environment, you must select the discrete-time solver in the **Solver** pane of the Simulink Configuration Parameters dialog box. Targeting does not work with continuous-time solvers.

To select the discrete-time solver, from the main menu in your model window, select **Simulation > Configuration Parameters**. Then in the **Solver** pane, set the **Solver** option to discrete (no continuous states).

---

## Target for TI C2000 and Code Composer Studio

<b>In this section...</b>
“Using Code Composer Studio with Target for TI C2000” on page 1-10
“Default Project Configuration” on page 1-10

### Using Code Composer Studio with Target for TI C2000

Texas Instruments (TI) facilitates development of software for TI DSPs by offering Code Composer Studio (CCS) Integrated Development Environment (IDE). Used in combination with Target for TI C2000 and Real-Time Workshop, CCS provides an integrated environment that, once installed, requires no coding.

Executing code generated from Real-Time Workshop on a particular target requires that Real-Time Workshop generate target code that is tailored to the specific hardware target. Target-specific code includes I/O device drivers and interrupt service routines (ISRs). Generated source code must be compiled and linked using CCS so that it can be loaded and executed on a TI DSP. To help you to build an executable, Target for TI C2000 uses Link for Code Composer Studio to start the code building process within CCS. After you download your executable to your target and run it, the code runs wholly on the target. You can access the running process only from the CCS debugging tools or across a link using Link for Code Composer Studio Development Tools.

### Default Project Configuration

CCS offers two standard project configurations, Release and Debug. Project configurations define sets of project build options. When you specify the build options at the project level, the options apply to all files in your project. For more information about the build options, refer to your TI documentation. The models you build with Target for TI C2000 use a custom configuration that provides a third combination of build and optimization settings — customMW.

### Default Build Options in the custom\_MW Configuration

The default settings for custom\_MW are the same as the Release project configuration in CCS, except for the compiler options.

Your CCS documentation provides complete details on the compiler build options. You can change the individual settings or the build configuration within CCS.

## Data Type Support

TI C2000 DSPs support 16-bit data types and do not have native 8-bit data types. Simulink and Target for TI C2000 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, that data will be 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.

# Scheduling and Timing

In this section...
“Overview” on page 1-13
“Timer-Based Interrupt Processing” on page 1-13
“Asynchronous Interrupt Processing” on page 1-14

## Overview

Normally the code generated by Target for TI C2000 runs out of 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, however, 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.

For C280x and C281x-based boards, Target for TI C2000 lets you model systems that include asynchronous hardware interrupt processing 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 C280x or C281x uses `CPU_timer0`.

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

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

The minimum achievable base rate sample time depends on the model complexity. The maximum value depends on the maximum timer period value ( $2^{32}-1$  for the F2812 and F2808), and the CPU clock speed. The CPU clock speed for the F2808 it is 100 MHz, and for the F2812 it is 150 MHz.

If all the blocks in the model inherit their sample time value, and no sample time is explicitly defined, Simulink assigns a default of 0.2 s.

### **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 always selected in Target for TI C2000. This clock is derived from the system clock (SYSCLKOUT):

$$\text{HISCLK} = [\text{SYSCLKOUT} / (\text{high-speed peripheral prescaler})]$$

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

$$\text{HISCLK} = 150 \text{ MHz} / 2 = 75 \text{ MHz}$$

### **Asynchronous Interrupt Processing**

Simulink and Real-Time Workshop facilitate the modeling and generation of code for asynchronous event handling, including servicing of hardware-generated interrupts, by using the following special blocks:

- Hardware Interrupt block

This block enables selected hardware interrupts, generates the corresponding interrupt service routines (ISRs), and connects them to the corresponding interrupt service vector table entries. When you connect the output of the Hardware Interrupt block to the control input of a triggered subsystem (for example, a function-call subsystem), the generated subsystem code is called from the ISRs.

Target for TI C2000 provides a Hardware Interrupt block for each of the supported processor families: C280x Hardware Interrupt and C281x Hardware Interrupt.

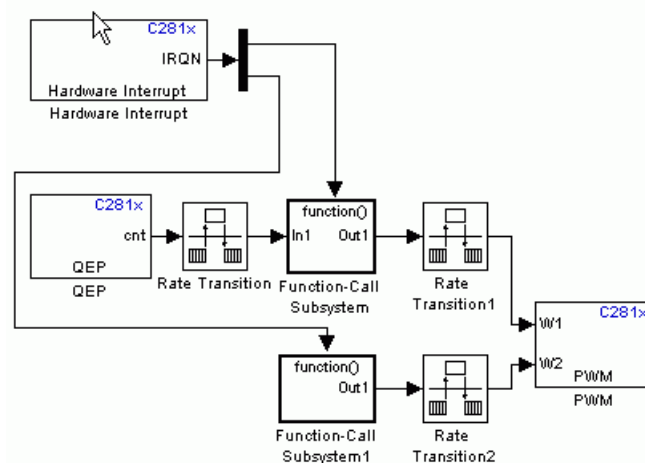
- Rate Transition blocks

These blocks support data transfers between blocks running with different priorities. The built-in Simulink Rate Transition blocks can be used for this purpose.

- Software Interrupt block

This block polls the input port for the input value, and when the input value is greater than a specified value, the block posts the interrupt to a Hardware Interrupt block in the model.

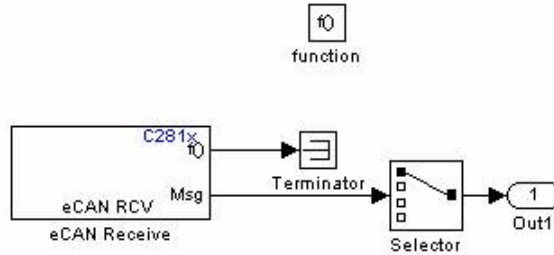
The following diagram illustrates a use case where a Hardware Interrupt block triggers two tasks, connected to other blocks that run periodically in the context of the synchronous scheduler.



In the preceding figure, the Hardware Interrupt block is set to react on two interrupts. Since only one Hardware Interrupt block is allowed in a model and the output of this block is a vector of length two, you must connect the Hardware Interrupt block to a Demux block to trigger the two function-called subsystems. The function-called subsystems contain the blocks that are executed asynchronously in the context of the hardware interrupt.

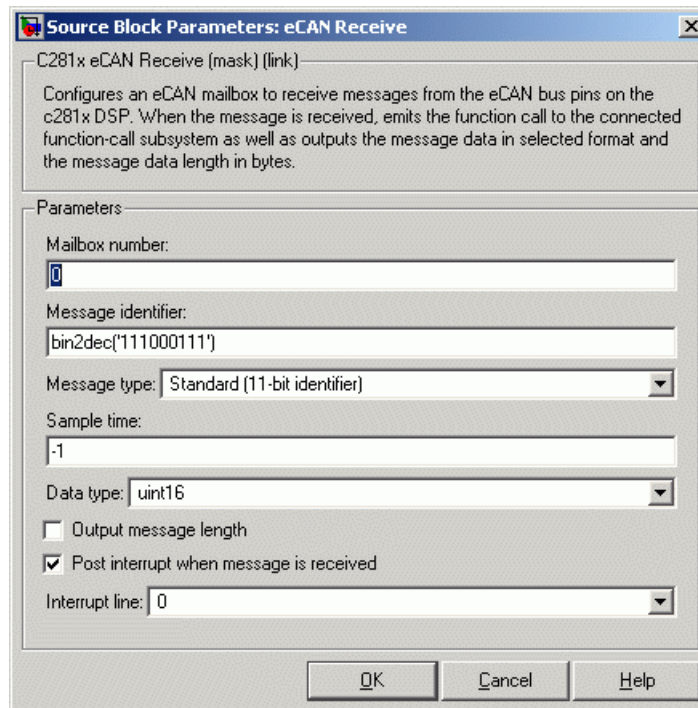
The following example shows how to build and configure a model to react on an eCAN message using a hardware interrupt and an asynchronous scheduler:

- 1 Place the eCAN Receive block in a function-called subsystem, as shown in the following figure.

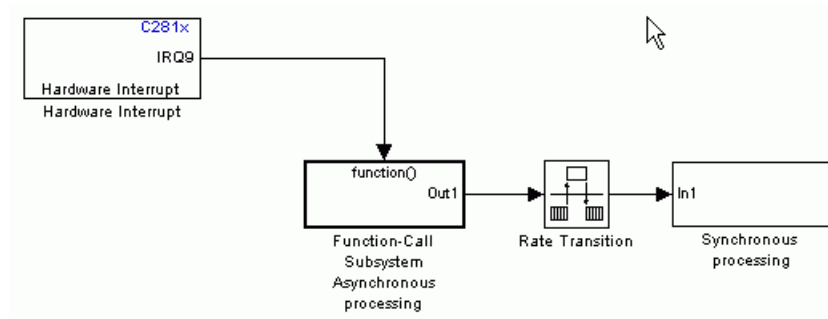


- 2 On the eCAN Receive block dialog, check the box labeled **Post interrupt when message is received**, as shown in the following figure.

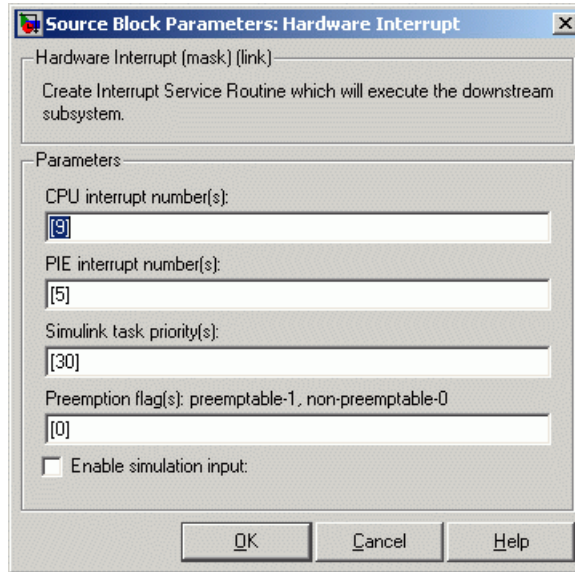




- 3 Set the **Sample Time** of the eCAN Receive block to -1 since the block will be triggered by the ISR, as shown in the preceding figure.
- 4 Add the C281x Hardware Interrupt block to your model, as shown in the following figure.



- 5 The eCAN interrupt on C281x chips is on CPU line 9 and PIE line 5 for module 0. These parameters can be found in the C281x Hardware Interrupt block, C281x Peripheral Interrupt Vector Values figure. Set the hardware interrupt parameters **CPU interrupt number(s)**: to 9, and **PIE interrupt number(s)**: to 5 as shown in the following figure.



- 6 Connect the output of the Hardware Interrupt block to the function-call subsystem containing the eCAN block.

At execution time, when a new eCAN message is received, the eCAN interrupt is triggered, and the code you placed in the function-called subsystem is executed. In this example, the eCAN Receive block is placed in the function-called subsystem, which means that the message is read and is passed to the rest of the code.

For more information, see the section on Asynchronous Support in the Real-Time Workshop documentation.

# Overview of Creating Models for Targeting

## In this section...

“Accessing the Target for TI C2000 Block Library” on page 1-19

“Online Help” on page 1-20

“Blocks with Restrictions” on page 1-20

“S-Function Builder Blocks” on page 1-22

“Setting Simulation Configuration Parameters” on page 1-22

“Building Your Model” on page 1-23

## Accessing the Target for TI C2000 Block Library

After you have installed the supported development board, start MATLAB. At the MATLAB command prompt, type

```
c2000lib
```

This opens the `c2000lib` Simulink blockset that includes libraries containing blocks predefined for C2000 input and output devices. As needed, add the blocks to your model. See “Using the `c2000lib` Blockset” on page 1-25 for an example of how to use this library.

Create your real-time model for your application the same way you create any other Simulink model — by using standard blocks and C-MEX S-functions. Select blocks to build your model from the following sources:

- Appropriate Target Preferences library block, to set preferences for your target and application
- From the appropriate libraries in the `c2000lib` block library, to handle input and output functions for your target hardware
- From Real-Time Workshop
- From Simulink Fixed Point
- Discrete time blocks from Simulink

- From any other blockset that meets your needs and operates in the discrete time domain

## Online Help

To get general help for using Target for TI C2000, use the help feature in MATLAB. At the command prompt, type

```
help tic2000
```

to list the functions and block libraries included in Target for TI C2000. Or select **Help > Full Product Family Help** from the menu bar in the MATLAB desktop. When you see the Contents in Help, select **Target for TI C2000**.

## Blocks with Restrictions

There are many blocks in different blocksets that communicate with your MATLAB workspace. Some blocks may not work on the target as they do on your desktop, and for that reason, you should avoid them altogether. Other blocks may have restrictions in their settings, which, when followed, ensure smooth communications. All the blocks that require this special consideration are listed in the following sections.

## Blocks to Avoid Using in Your Models

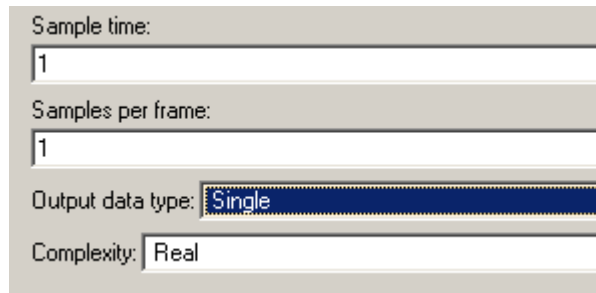
The blocks listed in the table below generate code, but they do not work on the target as they do on your desktop—in general, they slow your signal processing application without adding instrumentation value. For this reason, The MathWorks recommends that you *avoid* using certain blocks, such as the Scope block and some source and sink blocks, in SIMULINK models that you use for TI C2000 DSP targets.

<b>Library</b>	<b>Category</b>	<b>Block Name</b>
Simulink	Sinks	Scope
		To File
		To Workspace
	Sources	From File
		From Workspace
Signal Processing Blockset	Signal Operations	Triggered Signal From Workspace
	Signal Processing Sinks	Signal To Workspace
		Spectrum Scope
		Triggered to Workspace
		To Wave Device
		To Wave File
	Signal Processing Sources	Signal From Workspace
		From Wave Device
		From Wave File

### **Blocks That Require Specific Settings**

Any block listed in the following table can be used with all your models. However, such a block requires specific settings, as indicated under “Restriction.”

Library	Category	Block Name	Restriction
Signal Processing Blockset	Signal Processing Sources	Random Source Block	For this block, the only <b>Output data type</b> supported by the TI C2000 is <b>Single</b> . Be sure to set this parameter correctly in the <b>Block Parameters</b> dialog box. See the following figure.



## S-Function Builder Blocks

Simulink S-Function Builder can be used to create and add new blocks to your model. When you generate code for your model, related source code files are added to your Code Composer Studio project.

## Setting Simulation Configuration Parameters

When you drag a Target Preferences block into your model, you are given the option to set basic simulation parameters automatically.

To refine the automatic settings, or set the simulation parameters manually, open your model and select **Simulation > Configuration Parameters**.

If you are setting your simulation parameters manually, you must make at least the following two settings:

- You must specify discrete time by selecting **Fixed-step and discrete** (no continuous states) in the **Solver** pane of the Configuration Parameters dialog box.

- You must also specify the appropriate version of the system target file and template makefile in the **Real-Time Workshop** pane. For Target for TI C2000, specify one of the following system target files, or click **Browse** and select from the list of targets.

```
ccslink_grt.tlc  
ccslink_ert.tlc
```

The associated template filename is automatically filled in.

## System Target Types and Memory Management

There are two system target types that apply to Target for TI C2000. These correspond to the two system target files mentioned above.

A Generic Real-Time (GRT) target (such as `ccslink_grt.tlc`) is the target configuration that generates model code for a real-time system as if the resulting code was going to be executed on your workstation.

An Embedded Real-Time (ERT) target (such as `ccslink_ert.tlc`) is the target configuration that generates model code for execution on an independent embedded real-time system. This option requires Real-Time Workshop Embedded Coder.

The ERT target for Target for TI C2000 offers memory management features that give you a way manage the performance of your code while working with limited memory resources. For more information on this, see the chapter on Memory Sections in the *Real-Time Workshop Embedded Coder User's Guide*.

## Building Your Model

With this configuration, you can generate a real-time executable and download it to your TI development board by clicking **generate\_code** on the **Real-Time Workshop** pane. Real-Time Workshop automatically generates C code and inserts the I/O device drivers as specified by the hardware blocks in your block diagram, if any. These device drivers are inserted in the generated C code as inlined S-functions. For information about inlining S-functions, refer to your target language compiler documentation. For a complete discussion of S-functions, refer to your documentation about writing S-functions.

During the same build operation, block parameter dialog box entries are combined into a project file for CCS for your TI C2000 board. If you selected the **Build** and **execute build** action in the configuration settings, your makefile invokes the TI cross-compiler to build an executable file that is automatically downloaded via the parallel port to your target. After downloading the executable file to the target, the build process runs the file on the board's DSP.

---

**Note** After using the run-time **Build** option to generate and build code for your application, you must perform the following reset sequence before you can run that code on your board. If you want to rerun your application manually once it has been generated, you must also use this procedure.

---

## **F2812 eZdsp and F2808 eZdsp Reset Sequence**

- 1** Reset the board CPU.
- 2** Load your code onto the target.
- 3** Run your code on the target.



## Using the c2000lib Blockset

### In this section...

- “Introduction” on page 1-25
- “Hardware Setup” on page 1-25
- “Starting the c2000lib Library” on page 1-26
- “Setting Up the Model” on page 1-27
- “Adding Blocks to the Model” on page 1-31
- “Generating Code from the Model” on page 1-33

## Introduction

This section uses an example to demonstrate how to create a Simulink model that uses Target for TI C2000 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. This model is also provided in the Demos library. Note that the model in the Demos library also includes a model simulation.

## Hardware Setup

The following hardware is needed for this example:

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

To connect the 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.

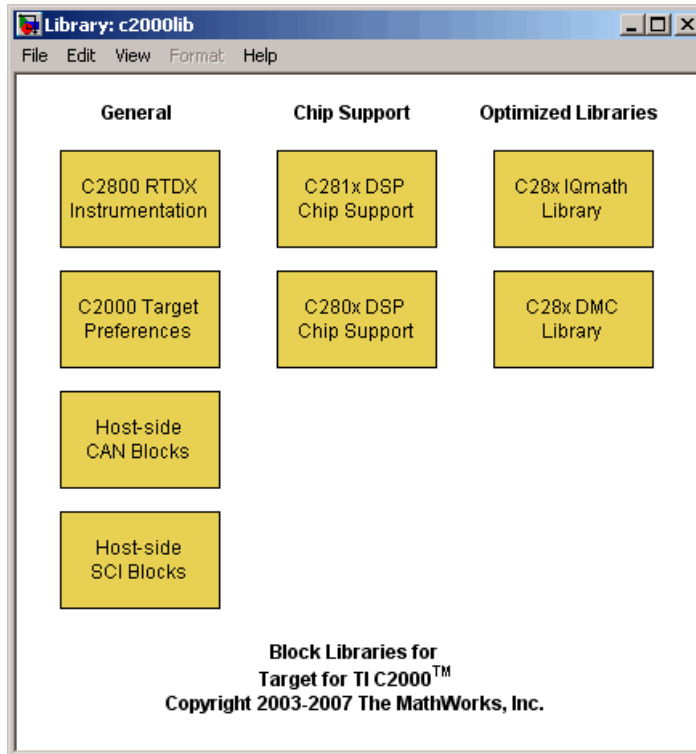
- 3 Connect VREFLO to AGND on the eZdsp F2812. See the section on the Analog Interface in Chapter 2 of the *eZdsp™ F2812 Technical Reference*, available from the Spectrum Digital Web site at <http://c2000.spectrumdigital.com/ezf2812/>

## Starting the c2000lib Library

At the MATLAB prompt, type

```
c2000lib
```

to open the c2000lib library blockset, which contains libraries of blocks designed for targeting your board.



The libraries are in three groups, plus Info and Demos blocks.

## General

- C2800 RTDX Instrumentation (rtdxBlocks) — Blocks for adding RTDX communications channels to Simulink models. See the tutorial in [Link for Code Composer Studio Development Tools documentation](#) for an example of using these blocks.
- C2000 Target Preferences (c2000tgtpreflib) — Blocks to specify target preferences and options. You do not connect this block to any other block in your model.
- Host-side CAN Blocks (c2000canlib) — Blocks to configure CAN message blocks and Vector CAN driver blocks
- Host-side SCI Blocks (c2000canlib) — Blocks to configure host-side serial communications interface to send and receive data from serial port

## Chip Support

- C281x DSP Chip Support (c281xdspchiplib) — Blocks to configure the codec on the F2812 eZdsp DSK or on C281x-based custom boards
- C280x DSP Chip Support (c280xdspchiplib) — Blocks to configure the codec on the F2808 eZdsp DSK or on C280x-based custom boards

## Optimized Libraries

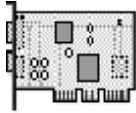
- C28x IQmath Library (tiiqmathlib) — Fixed-point math blocks for use with C28x targets
- C28x DMC Library (c28xdmclib) — Fixed-point math blocks for digital motor control with C28x DSPs

## Setting Up the Model

Preliminary tasks for setting up a new model include adding a Target Preferences block, setting or verifying Target Preferences, and setting the simulation parameters.

- 1 In the Library: c2000lib window, select **File > New > Model** to create a new Simulink model.

- 2 In the Library: c2000lib window, double-click the C2000 Target Preferences library block.
- 3 From the Target Preferences Library window, drag the F2812 eZdsp block into your new model.



F2812 eZdsp

- 4 Click **Yes** to allow automatic setup. The following settings are made, referenced in the table below by their locations in the **Simulation > Configuration Parameters** dialog box:

Pane	Field	Setting
Solver	Stop time	10
Solver	Type	Fixed-step
Data Import/Export	Save to workspace - Time	tout
Data Import/Export	Save to workspace - Output	yout
Hardware Implementation	Device type	C2000
Real-Time Workshop	Target selection - System target file	ccslink_grt.tlc or ccslink_ert.tlc

---

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

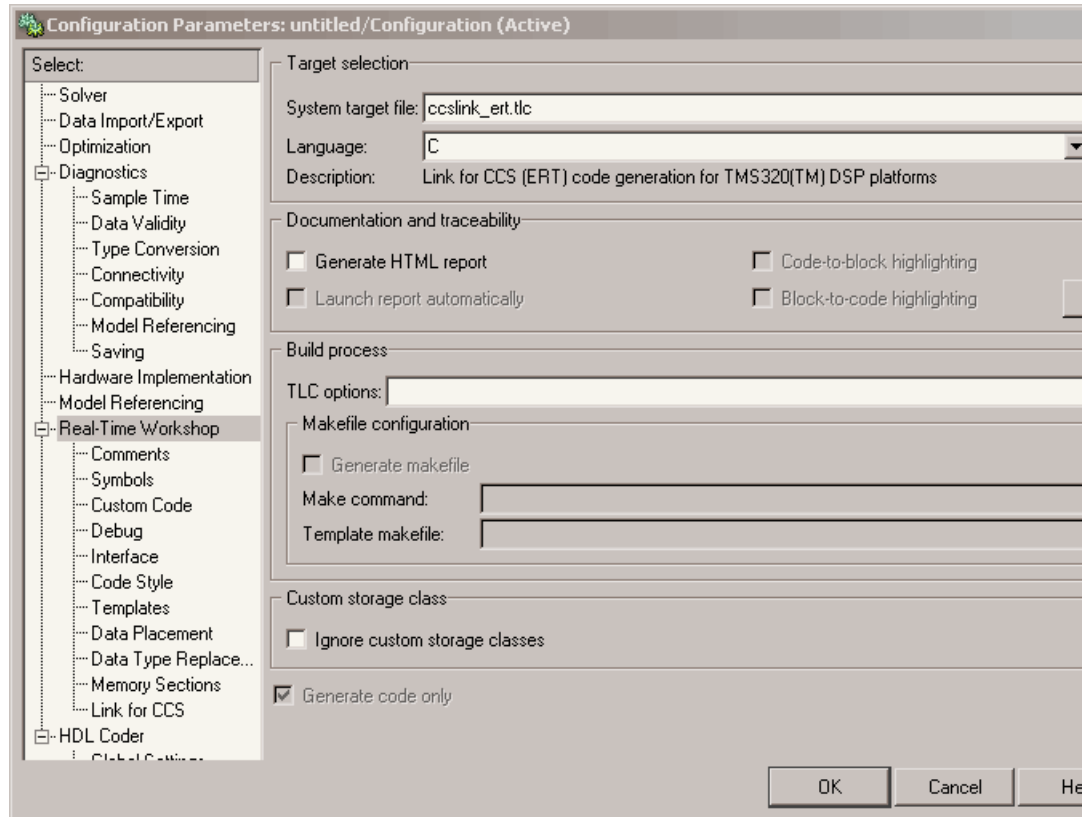
---

---

**Note** One Target Preferences block must be in each target model at the top level. It does not connect to any other blocks, but stands alone to set the target preferences for the model.

---

- 5** From your model's main menu, select **Simulation > Configuration Parameters** to verify and set the simulation parameters for this model. Parameters you set in this dialog box belong to the model you are building. They are saved with the model and stored in the model file. Refer to your Simulink documentation for information on the Configuration Parameters dialog box.
- 6** Use the **Real-Time Workshop** pane to set options for the real-time model. Refer to your "Real-Time Workshop" documentation for detailed information on the **Real-Time Workshop** pane options.

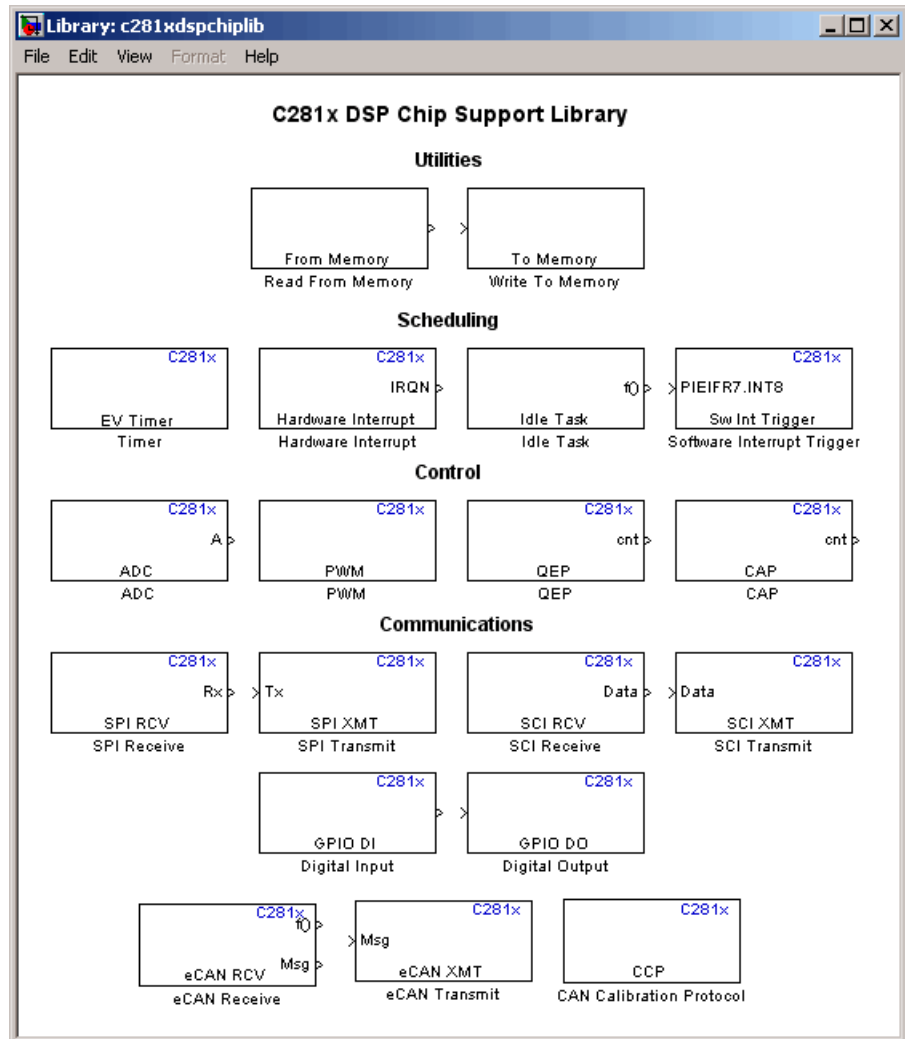


- **System target file.** Clicking **Browse** opens the **System target file browser** where you select `ccslink_grt.tlc` or `ccslink_ert.tlc`. When you select your target configuration, Real-Time Workshop chooses the appropriate system target file, template makefile, and make command. You can also enter the target configuration filename, and Real-Time Workshop will fill in the **Template makefile** and **Make command** selections.

- 7 Set the configuration parameters by typing **Ctrl-E** and adjust these parameters. For descriptions of these fields, see the Target Preferences reference page and “Setting Simulation Configuration Parameters” on page 1-22 in the section titled “Overview of Creating Models for Targeting” on page 1-19.

## Adding Blocks to the Model

- 1 Double-click the C281x DSP Chip Support Library to open it.



- 2 Drag the C281x ADC block into your model. Double-click the ADC block in the model and set **Sample time** to 64/80000. Use the default values

for all other fields. Refer to the C281x ADC reference page for information on these fields.

- 3** Drag the C281x PWM block into your model. Double-click the PWM block in the model and set the following parameters. Refer to the C281x PWM reference page for information on these fields.

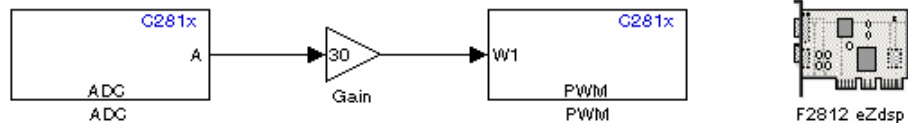
<b>Pane</b>	<b>Field</b>	<b>Parameter</b>
<b>Timer</b>	<b>Module</b>	A
	<b>Waveform period source</b>	Specify via dialog
	<b>Waveform period units</b>	Clock cycles
	<b>Waveform period</b>	64000
	<b>Waveform type</b>	Asymmetric
<b>Outputs</b>	<b>Enable PWM1/PWM2</b>	Selected
	<b>Duty cycle source</b>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



- 4 Enter Simulink at the MATLAB command line 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)
	Round integer calculations toward	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 as shown:



## Generating Code from the Model

This section summarizes how to generate code from your real-time model. For details about generating code from models in Real-Time Workshop, refer to the “Real-Time Workshop” documentation.

You start the automatic code generation process from the Simulink model window by clicking **Generate code** in the **Real-Time Workshop** pane of the Configuration Parameters dialog. Other ways of starting the code generation process are by clicking the **Incremental Build** button on the toolbar of your model, or by pressing the keyboard shortcut, **Ctrl+B**, while your model is open and in focus.

**Note** In CCS, you see your project with the files in place in the directory tree.

# Configuring Timing Parameters for CAN Blocks

---

Blocks Where the Bit Rate Cannot Be Set Directly (p. 2-2)

Lists the specific blocks whose timing parameters are set with the described process

Setting Timing Parameters (p. 2-3)

Describes how to set block timing parameters to obtain the required bit rate

Parameter Tuning and Signal Logging (p. 2-9)

How use Simulink external mode or a third party calibration tool for signal logging and parameter tuning.

### **Blocks Where the Bit Rate Cannot Be Set Directly**

There are four specific CAN blocks in the C2000 control where the bit rate cannot be set directly and require the setting of timing parameters. These blocks are:

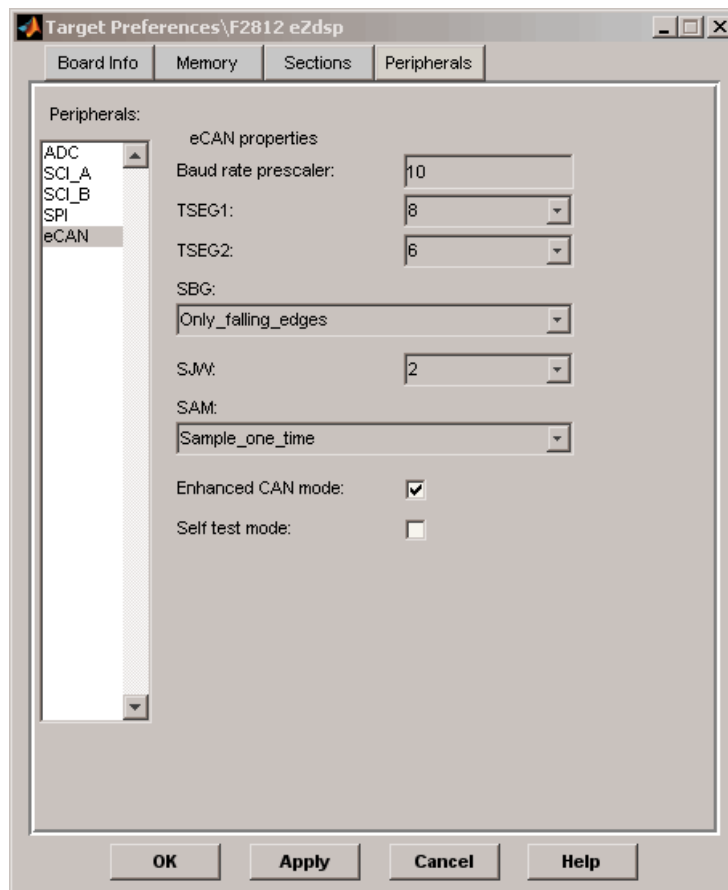
- C281x eCAN Receive
- C281x eCAN Transmit
- C280x eCAN Receive
- C280x eCAN Transmit

## Setting Timing Parameters

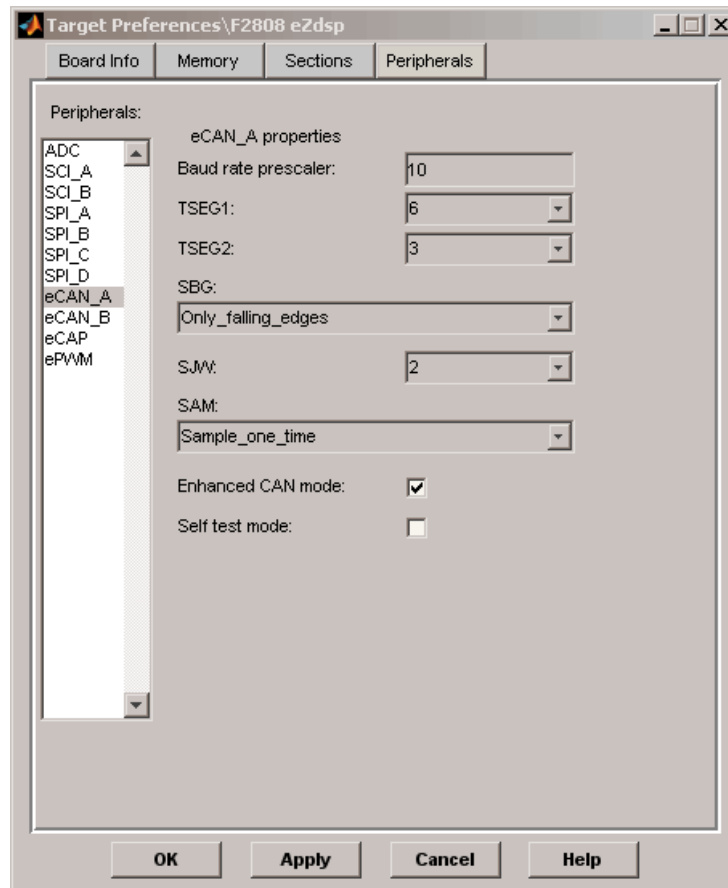
In this section...
“Accessing the Timing Parameters” on page 2-3
“Equations for Bit Rate Calculation” on page 2-5
“CAN Bit Timing Examples” on page 2-7

### Accessing the Timing Parameters

In “Blocks Where the Bit Rate Cannot Be Set Directly” you must use the following parameters: **TSEG1**, **TSEG2**, and **BaudRatePrescaler (BRP)** to set the required bit rate. These parameters are configured from the **Target Preference Setup** dialog for your specific model, in the **Peripherals** tab. To open the **Target Preference Setup** dialog, double click the target preferences block in your model. For example, for the C281x blocks, this dialog box is shown in the following figure:



For the C280x blocks, there are two separate eCAN modules that can be set independently, as shown by the Target Preferences Setup dialog box:



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

## Equations for Bit Rate Calculation

The following steps guide you through the process of configuring the required timing parameters.

- 1 Review the known entities:

### Bit Rate

This is the rate you want to set for your CAN.

### SYSCLOCKOUT

This is the CAN module system clock frequency.

- 2 Estimate the value of the **BaudRatePrescaler (BRP)** and substitute this value, along with the known values of *Bitrate* and *SYSCLOCKOUT*, into the equation below as follows:

$$\textit{Bitrate} = \textit{SYSCLOCKOUT} / (\textit{BRP} * \textit{BitTime})$$

Solve this equation for *BitTime* to obtain a value:

$$\textit{BitTime} = \textit{SYSCLOCKOUT} / (\textit{BRP} * \textit{Bitrate})$$

- 3 Estimate values of **TSEG1** and **TSEG2** that satisfy the following equation:

$$\textit{BitTime} = \textit{TSEG1} + \textit{TSEG2} + 1$$

Remember that *BitTime* is now a known quantity, calculated in the previous step.

- 4 Validate these estimated values of **BRP**, **TSEG1**, and **TSEG2** against the following rules:

$$\textbf{TSEG1} \geq \textbf{TSEG2}$$

$$\textit{IPT} (\text{Information Processing Time}) = 3/\textbf{BRP}$$

$$\textit{IPT} \leq \textbf{TSEG1} \leq 16 \textit{TQ}$$

$$\textit{IPT} \leq \textbf{TSEG2} \leq 8 \textit{TQ}$$

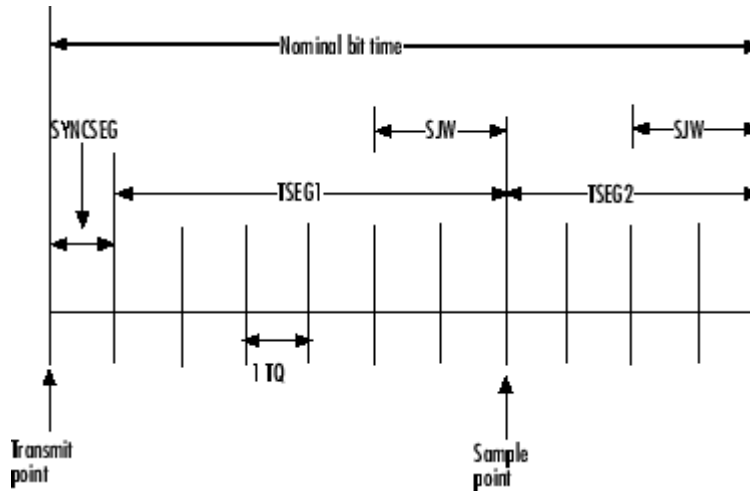
$$1 \textit{TQ} \leq \textbf{SJW} \leq \min(4 \textit{TQ}, \textbf{TSEG2})$$

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

- 5 Iterate steps two through four until the values selected for **TSEG1**, **TSEG2**, and **BRP** meet all of the criteria.

Another way to look at the eCAN bit timing parameters is shown in the following illustration.





## CAN Bit Timing Examples

Assume that  $SYSCLKOUT = 150$  MHz, and a bit rate of 1 Mbits/s is required.

- 1 Try to set the  $BRP$  to 10. Then substitute the values of bit rate,  $BRP$ , and  $SYSCLKOUT$  into the following equation, solving for  $BitTime$ :

$$BitTime = SYSCLKOUT / (BRP * Bitrate)$$

$$BitTime = 150 / (10 * 1) = 15TQ$$

- 2 Try to 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:

$$IPT = 3/BRP = 3/10 = .3$$

$$IPT \leq TSEG1 \leq 16 TQ \text{ True! } .3 \leq 8TQ \leq 16TQ$$

$$IPT \leq TSEG2 \leq 8TQ \text{ True! } .3 \leq 6TQ \leq 8TQ$$

$1TQ \leq \mathbf{SJW} \leq \min(4TQ, \mathbf{TSEG2})$  which means that **SJW** can be set to either 2, 3, or 4

**4** Because all chosen values satisfy the criteria, no further iteration is necessary.

The following table provides common timing parameter settings for 3 typical values of Bit Rate and SYSCLKOUT = 150MHz. This clock frequency is the maximum for the C281x blocks.

<b>Bit Rate</b>	<b>TSEG1</b>	<b>TSEG2</b>	<b>Bit Time</b>	<b>BRP</b>	<b>SJW</b>
.5 Mbit/s	8	6	15	20	2
1 Mbit/s	8	6	15	10	2
2 Mbit/s	8	6	15	5	2

The following table provides common timing parameter settings for 3 typical values of Bit Rate and SYSCLKOUT = 100MHz. This clock frequency is the maximum for the C280x blocks.

<b>Bit Rate</b>	<b>TSEG1</b>	<b>TSEG2</b>	<b>Bit Time</b>	<b>BRP</b>	<b>SJW</b>
.5	6	3	10	20	2
1	5	4	10	10	2
2	6	3	10	5	2

# Parameter Tuning and Signal Logging

In this section...
“Overview” on page 2-9
“Using External Mode” on page 2-9
“Using a Third Party Calibration Tool” on page 2-18

## Overview

Target for TI C2000 supports parameter tuning and signal logging either using Simulink external mode or with a third party calibration tool. In both cases the model must include a special block, the CAN Calibration Protocol block.

## Using External Mode

Simulink external mode enables you to log signals and tune parameters without requiring a calibration tool. This section describes the steps for converting a model to use external mode.

External mode is supported using the CAN Calibration Protocol block and ASAP2 interface. The CAN Calibration Protocol block is used to communicate with the target, downloading parameter updates and uploading signal information. The ASAP2 interface is used to get information about where in the target memory a parameter or signal lives.

---

**Note** You must configure the host-side CAN application channel. See “Configuring the Host Vector CAN Application Channel ” on page 2-11.

---

To prepare your model for external mode, follow these steps:

- 1 Add a CCP driver block.
- 2 Add a Switch External Mode Configuration Block (for ease of use; you can also make changes manually).

- 3 Identify signals you want to tune, and associate them with `Simulink.Parameter` objects with `ExportedGlobal` storage class. It is important to set the data type and value of the `Simulink.Parameter` object. See “Using Supported Objects and Data Types” on page 2-11.
- 4 Identify signals you want to log, and associate them with `canlib.Signal` objects. It is important to set the data type of the `canlib.Signal`. See “Using Supported Objects and Data Types” on page 2-11.

For information about visualizing logged signal data, see “Viewing and Storing Signal Data” on page 2-13.

- 5 Load the the `Simulink.Parameter` and `canlib.Signal` data objects into the base workspace.
- 6 Configure the model for building by double-clicking the `Switch External Mode Configuration` block. In the dialog box, select **Building an executable**, and click **OK**.
- 7 Build the model, and download the executable to the target
- 8 After downloading the executable to the target, you can switch the model to external mode by double-clicking the `Switch External Mode Configuration Block`. In the dialog box that appears, select **External Mode**, and click **OK**.
- 9 You can now connect to the target using external mode by clicking the **Connect** button.
- 10 If you have set up tunable parameters, you can now tune them. See “Tuning Parameters” on page 2-12.

If you do not want to use the `Switch External Mode Configuration` block, you can configure for building and then external mode manually. For instructions, see “Manual Configuration For External Mode” on page 2-16.

See the following topics for more information:

- “Configuring the Host Vector CAN Application Channel ” on page 2-11
- “Using Supported Objects and Data Types” on page 2-11
- “Tuning Parameters” on page 2-12

- “Viewing and Storing Signal Data” on page 2-13
- “Manual Configuration For External Mode” on page 2-16
- “Limitations” on page 2-17

## Configuring the Host Vector CAN Application Channel

External mode expects that the host-side CAN connection is using the 'MATLAB 1' application channel. To configure the application channel used by the Vector CAN drivers, enter the following at the MATLAB command line:

```
TargetsComms_VectorApplicationChannel.configureApplicationChannels
```

The Vector CAN Configuration tool appears. Use this tool to configure your host-side CAN channel settings.

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

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

## Using Supported Objects and Data Types

Supported objects:

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

Supported data types:

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

You need to define data objects for the signals and parameters of interest for ASAP 2 file generation. For ease of use, create an m-file to define the data objects, so that you only have to set up the objects once.

To set up tuneable parameters and signal logging:

- 1 Associate the parameters to be tuned with `Simulink.Parameter` objects with `ExportedGlobal` storage class. It is important to set the data type and value of the `Simulink.Parameter` object. See the following m-code for an example of how to create such a `Simulink.Parameter` object for tuning:

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

- 2 Associate the signals to be logged with `canlib.Signal` objects. It is important to set the data type of the `canlib.Signal`. The following m-code example shows how to declare such a `canlib.Signal` object for logging:

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

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

### Tuning Parameters

To tune a parameter, follow these steps:

- 1 Set `dataobject.value` in the workspace 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, enter the following at the command line:

```
stepSize.value = 2
```

You see output similar to the following:

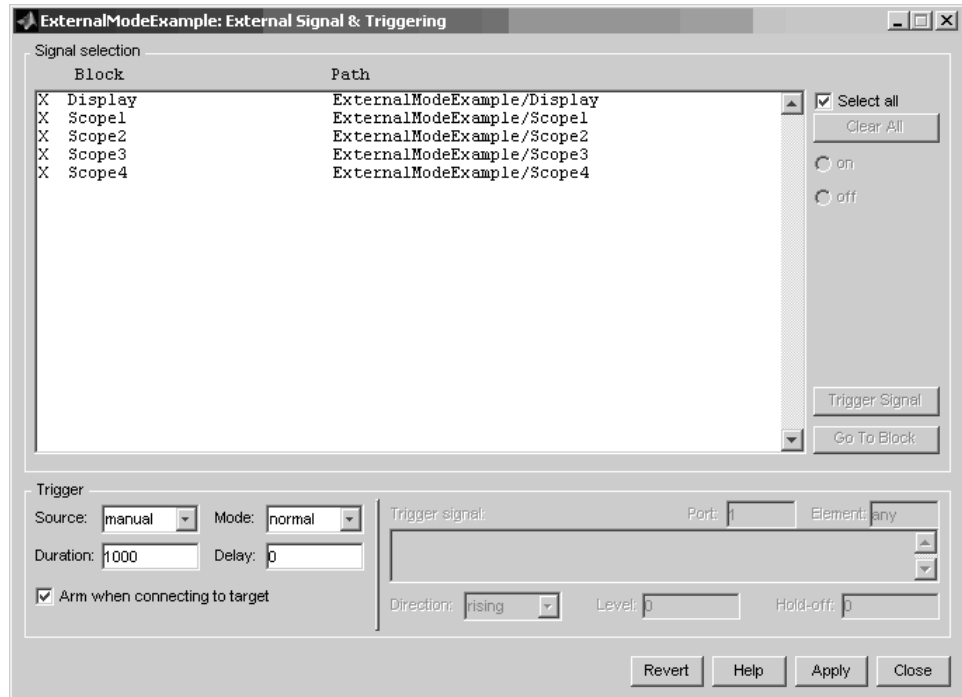
```
stepSize =  
  
Simulink.Parameter (handle)  
    RTWInfo: [1x1 Simulink.ParamRTWInfo]  
    Description: ''  
    DataType: 'uint8'  
    Min: -Inf  
    Max: Inf  
    DocUnits: ''  
    Value: 2  
    Complexity: 'real'  
    Dimensions: [1 1]
```

- 2 Return to your model, and 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 (see “Limitations” on page 2-17 for supported scope types).

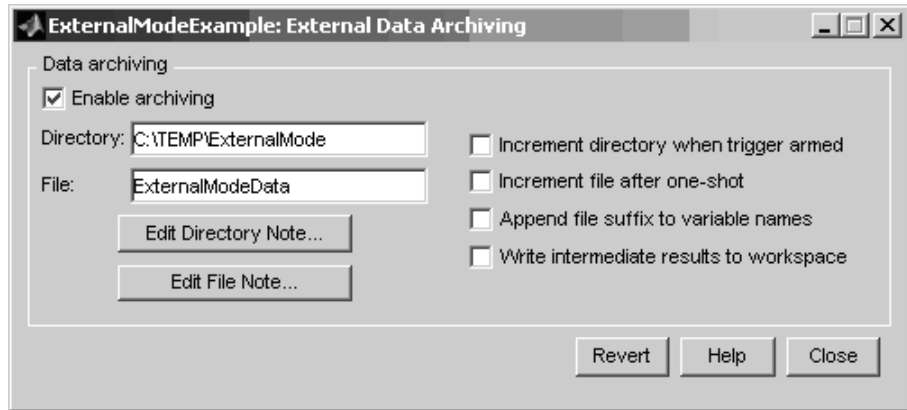
Select which signals you want to log by using the External Signal & Triggering dialog box. Access the External Mode Control Panel from the Tools menu, and click the **Signal & Triggering** button. By default, all displays appear as selected to be logged, as shown in the following example. Edit these settings if you do not want to log all displays. Individual displays can be selected manually.



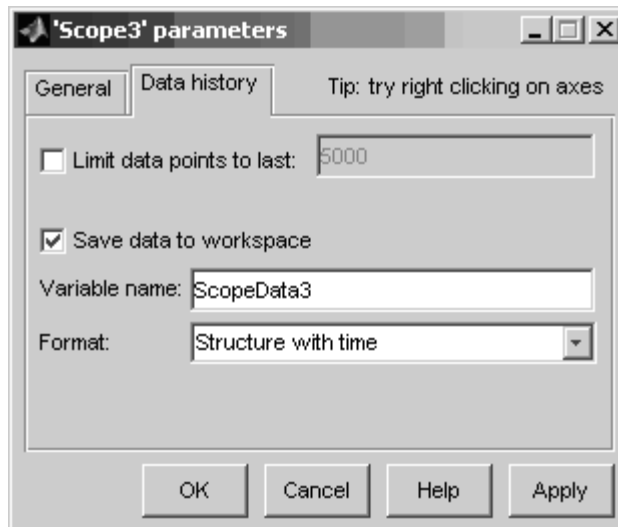
**Storing signal data for further analysis.** It is possible to store the logged data for further analysis in MATLAB.

- 1 To use the Data Archiving feature of external mode, click **Data Archiving** in the External Mode Control Panel. The External Data Archiving dialog box appears.





- a Select the check box **Enable archiving**
  - b Edit the **Directory** and **Filename** and any other desired settings.
  - c Close the dialog box.
- 2 Open the Scope parameters, and select the check box **Save data to workspace**.



- 3** You may want to edit the **Variable name** in the edit box. The data that is displayed on 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 as previously setup using Data Archiving.

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

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

```
ScopeData5

ScopeData5 =

        time: [56x1 double]
       signals: [1x1 struct]
   blockName: 'mpc555rt_ccp/Scope1'
```

- In the current directory, .mat files for the three previous **Durations** of scope data:

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

### Manual Configuration For External Mode

As an alternative to using the Switch External Mode Configuration block, you can configure models manually for build and execution with external mode.

To configure a model to be built for external mode:

- 1** Select **Inline parameters** (under Optimization in the Configuration Parameters dialog box). The **Inline parameters** option is required for ASAP2 generation.
- 2** Select **Normal** simulation mode (in either the Simulation menu, or the drop-down list in the toolbar).

- 3 Select ASAP2 as the **Interface** (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).

After you build the model, you can configure it for external mode execution:

- 1 Make sure **Inline parameters** are selected (under Optimization in the Configuration Parameters dialog box). The **Inline parameters** option is required for external mode.
- 2 Select **External** simulation mode (in either the Simulation menu, or the drop-down list in the toolbar).
- 3 Select External mode as the **Interface** (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).

### Limitations

Multiple signal sinks (e.g. 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, right-click a signal in the model, and select **Create & Connect Viewer > Simulink > Scope**. The other scope types listed there are not supported (e.g., floating scope).

Before connecting to external mode, you also need to right-click the signal, and select **Signal Properties**. In the dialog box, select the **Test point** check box, and click **OK**.

GRT is supported but only for parameter tuning.

It is not possible to log signals with very fast sample times (e.g., 0.0001) without losing data.

Subsystem builds are not supported for external mode, only top-level builds are supported.

Logging and tuning of nonscalars is not supported. It is possible to log nonscalar signals by breaking the signal down into its scalar components. For an example of how to do this signal deconstruction, see the CCP demo 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 the complex number down into its real and imaginary components. This breakdown can be performed using the following blocks in the Simulink Math Operations library: Complex to Real-Imag, Real-Imag to Complex, Magnitude-Angle to Complex, Complex to Magnitude-Angle.

### Using a Third Party Calibration Tool

Target for TI C2000 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. Target for TI C2000 lets you export an ASAP2 file containing information about your model during the code generation process. See also .

Before you begin generating ASAP2 files with Target for TI C2000, you should read the “Generating ASAP2 Files” section of the Real-Time Workshop documentation. That section describes how to define the signal and parameter information required by the ASAP2 file generation process.

Select the ASAP2 option before the build process as follows:

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

The Configuration Parameters dialog box appears.

- 2 Select **Interface** (under **Real-Time Workshop**) in the tree.

- 3 Select the ASAP2 option from the **Interface** drop-down menu, in the **Data exchange** frame.

- 4 Click **Apply**.

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

- The standard Real-Time Workshop ASAP2 file generation does not include the memory address attributes in the generated file. Instead, it leaves a placeholder that must be replaced with the actual address by postprocessing the generated file.
- The map file options in the template project need to be set up a certain way for this procedure to work. If you have created your own template projects, 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 columns should be 1
You can change these options via Project -> Project Options
-> Linker/Locator -> Map File -> Map File Format.
```

Target for TI C2000 performs this postprocessing for you. To do this, it first extracts the memory address information from the map file generated during the link process. Secondly, it replaces the placeholders in the ASAP2 file with the actual memory addresses. This postprocessing is performed automatically and requires no additional input from you.



# Configuring Acquisition Window Width for ADC Blocks

---

What Is an Acquisition Window?  
(p. 3-2)

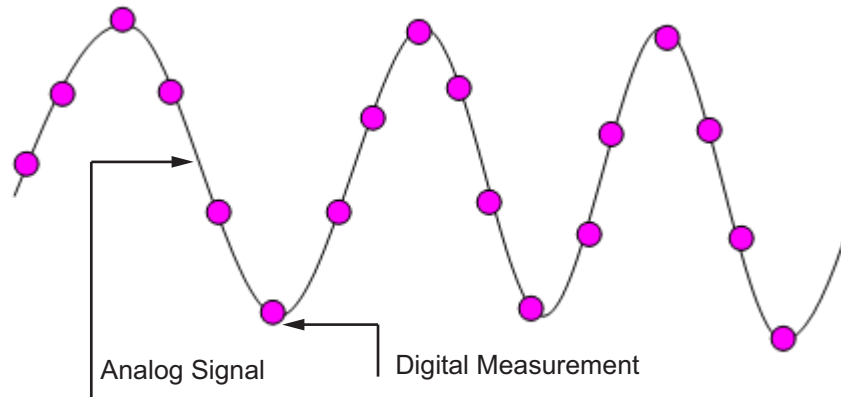
Explains the concept of the acquisition window and how it affects data validity

Configuring ADC Parameters for Acquisition Window Width (p. 3-5)

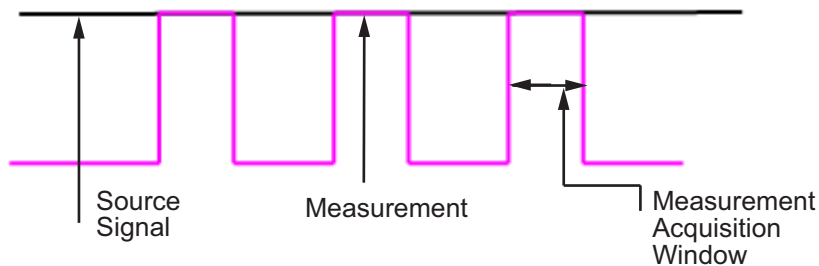
Describes how to set ADC parameters to obtain the proper acquisition window width

## 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, as shown in the following figure:



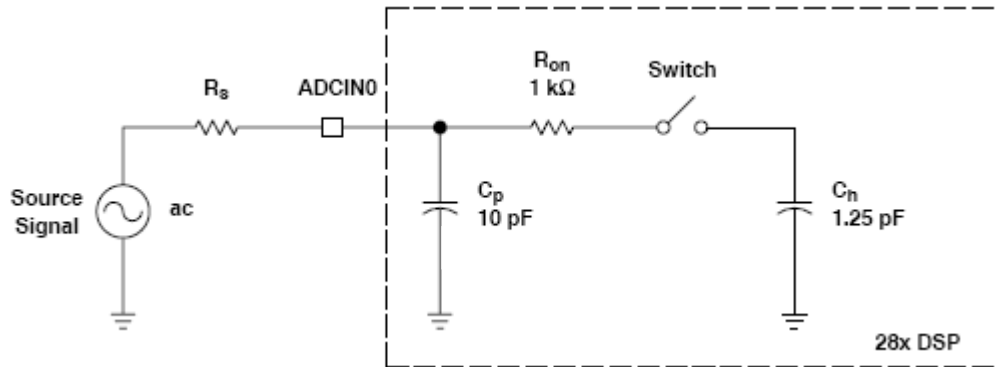
The digital measurement itself is not an instantaneous process, but is a measurement window, where the signal is acquired and measured, as shown below:



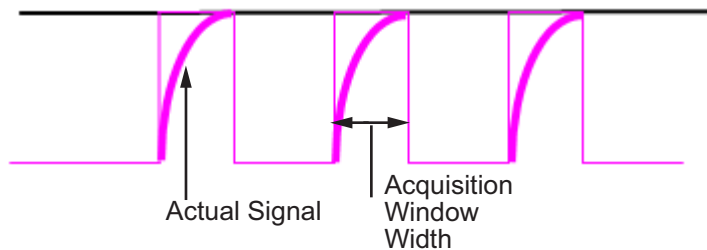
Ideally, as soon as the measurement window is opened, the actual signal coming in would be measured perfectly. In reality the signal does not reach its full magnitude immediately. The measurement process can be modeled by a



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 necessary

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 necessary acquisition window width.

## Configuring ADC Parameters for Acquisition Window Width

### In this section...

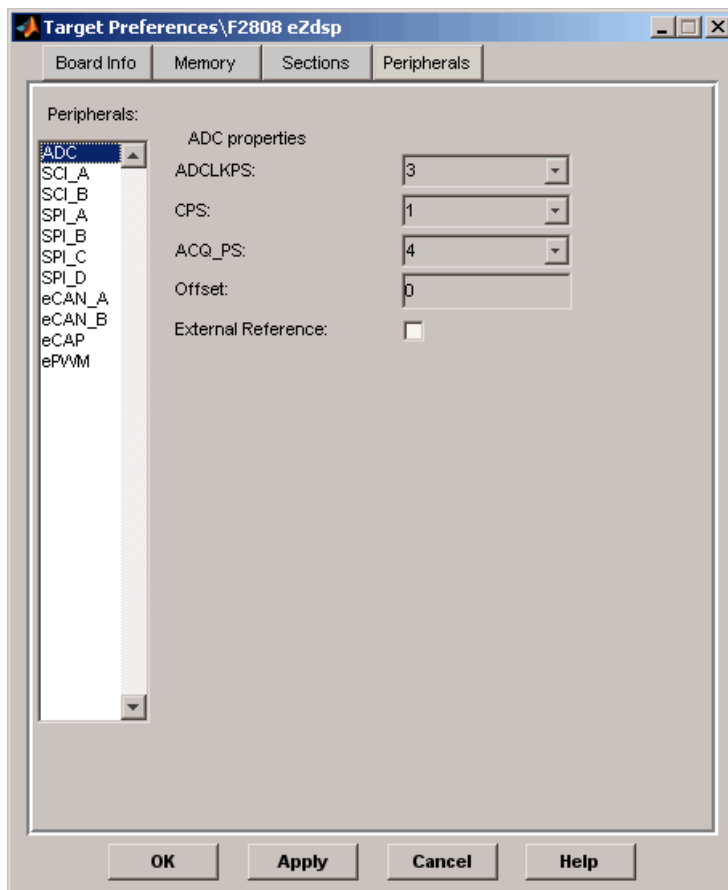
“Accessing the ADC Parameters” on page 3-5

“Examples” on page 3-7

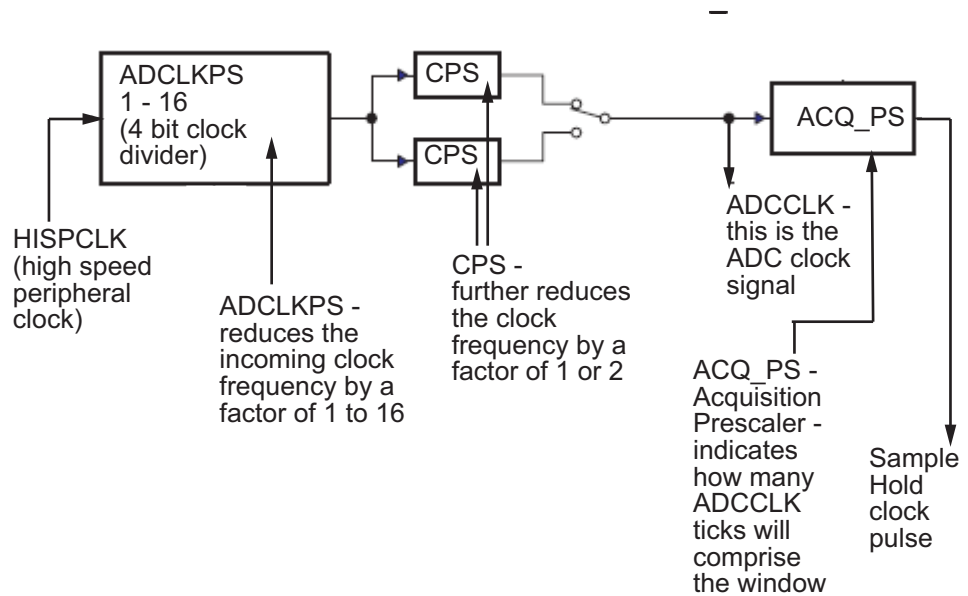
### Accessing the ADC Parameters

The ADC parameters can be set from the **Peripherals tab** of the Custom C280x Board configuration settings, or the configuration settings for the Custom C281x Board, or the F2808 eZdsp. These parameters are:

- **ACQ\_PS** — Acquisition Prescaler — can be set to a value from 0 to 15, however, the actual value is incremented by 1 to result in a range from 1 to 16.
- **ADCLKPS** — AD Clock Prescaler — can be set to a value from 0 to 15, however, the actual value is incremented by 1 to result in a range from 1 to 16.
- **CPS** — Clock Prescaler — can be set to a value from 0 to 1, however, the actual value is incremented by 1 to result in a range from 1 to 2.



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.

## Examples

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 15MHz.

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

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

Example 2:

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

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

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

---

**Note** HISPCLK is set automatically for the user, and it is not possible to change the rate.

---

# Creating Stand-Alone Applications by Saving Code into Flash Memory

---

The Need for Stand-Alone Applications (p. 4-2)

Explains the need and use for storing code in Flash memory

Generating Code for Flash Memory (p. 4-3)

Lists necessary steps to place generated code into on-chip Flash memory

Running Code from Flash Memory (p. 4-4)

Describes the required steps to run code from on-chip Flash memory

### **The Need for Stand-Alone Applications**

By default, the code generated through the Code Composer Studio (CCS) is stored in RAM on the applicable chip and is discarded when the chip is unplugged. However, there is often a need to save the generated code directly on the DSP chip nonvolatile memory to reuse it for a different application or environment. Flash memory allows this process to take place. Saving the code in Flash, directly on the chip, allows the chip to be unplugged and reused at a different time.



## Generating Code for Flash Memory

To store generated code in the internal Flash memory of the C28xx DSPs specific parameters need to be set. You also need a Flash Programmer (the TI Flash programmer is installed by default with CCS). The following process guides you through the necessary steps:

- 1** Drag the F2812 or F2808 **Stand alone using Flash Memory** Target Preferences block into the model.
- 2** Programming the on-chip Flash for TI C28xx DSPs requires a Flash Programmer. The two most commonly used options are the TI Flash Programmer, which is installed by default with CCS, or the Spectrum Digital™ SDFlash. Refer to the specific vendor's documentation for more information, and then download and install a Flash Programmer of your choice.
- 3** Build and generate code in CCS. Then, launch the Flash Programmer to erase, program, and verify the Flash. Your chip now contains the code in its Flash memory.

### Running Code from Flash Memory

Now that the code is saved in the C28xx DSP chip nonvolatile memory, you must set an indicator for the chip before you can run this code. This indicator is set by the Bootloader Modes of the particular chip. For example, on F2812 eZdsp, you need to adjust the jumper setting for JP7. On F2808 eZdsp, you need to adjust the switches 1 and 3 on bank SW1. For precise instructions, refer to the specific DSP Boot ROM Reference Guide found on the TI Web page and the Spectrum Digital™ Reference Guides for the eZdsp chips.

# Using the IQmath Library

---

About the IQmath Library (p. 5-2)

Introduces the IQmath Library

Fixed-Point Numbers (p. 5-4)

Representation of fixed-point numbers in the IQmath Library

Building Models (p. 5-9)

Issues to consider when you build models with the IQmath Library

## About the IQmath Library

In this section...
“Introduction” on page 5-2
“Common Characteristics” on page 5-3

### Introduction

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

---

**Note** The 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 on fixed-point numbers and Q format, see

- “Fixed-Point Numbers” on page 5-4. In addition, see the Simulink Fixed Point documentation, which includes more information on fixed-point data types and scaling and precision issues.
- “Q Format Notation” on page 5-5

You can use these blocks with some core Simulink blocks and Simulink Fixed Point blocks to run simulations in Simulink models before generating code. Once you develop your model, you can invoke Real-Time Workshop to 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 both IQmath Library blocks and blocks from other blocksets, refer to “Building Models” on page 5-9.

## **Common Characteristics**

The following characteristics are common to all IQmath Library blocks:

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

To learn more about characteristics particular to each block in the library, see “C28x IQmath (tiiqmathlib)” on page 6-11 for links to the individual block reference pages.

## Fixed-Point Numbers

### In this section...

“Notation” on page 5-4

“Signed Fixed-Point Numbers” on page 5-5

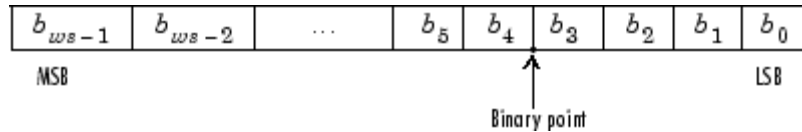
“Q Format Notation” on page 5-5

### 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.

## 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 have no 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 always designated as the sign bit. Representing a signed fixed-point data type in Q format always 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

---

### **Example – 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 Simulink Fixed Point, this data type is expressed as

sfrac16



or

`sfixed16_En15`

In Filter Design Toolbox, this data type is expressed as

[16 15]

### Example – 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:

$$m+n+1 = 1+30+1 = 32 \text{ bits total}$$

Therefore, this number is expressed as

Q1.30

In Simulink Fixed Point, this data type is expressed as

`sfixed32_En30`

In Filter Design Toolbox, this data type is expressed as

[32 30]

### Example – 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:

$$m+n+1 = -2+17+1 = 16 \text{ bits total}$$

Therefore, this number is expressed as

Q-2.17

In Simulink Fixed Point, this data type is expressed as

`sfixed16_E17`

In Filter Design Toolbox, this data type is expressed as

[16 17]

### **Example – 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 Simulink Fixed Point, this data type is expressed as

`sfixed16_E2`

In Filter Design Toolbox, this data type is expressed as

[16 -2]

## Building Models

In this section...
“Overview” on page 5-9
“Converting Data Types” on page 5-9
“Using Sources and Sinks” on page 5-10
“Choosing Blocks to Optimize Code” on page 5-10

### Overview

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

### Converting Data Types

As always, it is vital to make sure that any 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 any block reference page in the alphabetical block reference for a discussion of the data types that the block accepts and produces.

When you connect IQmath Library blocks and Simulink Fixed Point blocks, you often need to set the data type and scaling in the block parameters of the Simulink Fixed Point block to match the data type of the IQmath Library block. Many Simulink Fixed Point 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 Simulink Fixed Point block to match a connected IQmath Library block.

Some Signal Processing Blockset blocks and core Simulink blocks also accept fixed-point data types. Make the appropriate 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 Simulink Fixed Point 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, both the IQmath Library and Simulink Fixed Point have a Multiply block. When you are building a model to run on C2000 DSP, choosing the block from the IQmath Library always 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.

# Blocks — By Category

---

C2000 Target Preferences (c2000tgtpreflib) (p. 6-2)	Target preference blocks for C2000 boards
Host-Side CAN Blocks (c2000canlib) (p. 6-3)	Host-Side CAN blocks
Host-Side SCI Blocks (c2000scilib) (p. 6-4)	Host-Side SCI blocks
C2000 RTDX Instrumentation (rtdxBlocks) (p. 6-5)	RTDX blocks for C2000 boards
C280x DSP Chip Support (c280xdspchiplib) (p. 6-6)	Blocks that support C280x boards
C281x DSP Chip Support (c281xdspchiplib) (p. 6-8)	Blocks that support C281x boards
C28x Digital Motor Control (c28xdmclib) (p. 6-10)	Blocks that represent the functionality of the TI C28x DMC Library
C28x IQmath (tiiqmathlib) (p. 6-11)	Blocks that represent the functionality of the TI IQmath Library

## C2000 Target Preferences (c2000tgtplib)

Custom Board	Target preferences for custom C28xx board
F2808 eZdsp	F2808 eZdsp DSK target preferences
F2808 eZdsp Stand alone code using Flash Memory	
F2812 eZdsp	F2812 eZdsp DSK target preferences
F2812 eZdsp Stand alone code using Flash Memory	

## Host-Side CAN Blocks (c2000canlib)

See the *CAN Blockset Reference* for information on these blocks. See “Parameter Tuning and Signal Logging” on page 2-9 for information about using external mode with CCP.

## **Host-Side SCI Blocks (c2000scilib)**

SCI Receive

Configure host-side serial communications interface to receive data from serial port

SCI Setup

Configure COM ports for host-side SCI Transmit and Receive blocks

SCI Transmit

Configure host-side serial communications interface to transmit data to serial port



## **C2000 RTDX Instrumentation (rtdxBlocks)**

From RTDX

Add RTDX input channel

To RTDX

Add RTDX output channel

## C280x DSP Chip Support (c280xdspchilib)

C280x ADC	Analog-to-digital converter (ADC)
C280x eCAN Receive	Enhanced Control Area Network receive mailbox
C280x eCAN Transmit	Enhanced Control Area Network transmit mailbox
C280x eCAP	Receive and log capture input pin transitions or configure auxiliary pulse width modulator
C280x ePWM	Configure C280x Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms
C280x eQEP	Quadrature encoder pulse circuit
C280x GPIO Digital Input	Configure general purpose input pins
C280x GPIO Digital Output	Configure general purpose output pins
C280x Hardware Interrupt	Interrupt Service Routine to handle hardware interrupt on C280x processor
C280x SCI Receive	Receive data on target via serial communications interface (SCI) from host
C280x SCI Transmit	Transmit data from target via serial communications interface (SCI) to host
C280x SPI Receive	Receive data via serial peripheral interface (SPI) on target
C280x SPI Transmit	Transmit data via serial peripheral interface (SPI) to host
C280x SW Int Trigger	Generate software triggered nonmaskable interrupt

From Memory	Retrieve data from target memory
Idle Task	Free-running task that executes downstream subsystem
To Memory	Write data to target memory

## C281x DSP Chip Support (c281xdspchilib)

C281x ADC	Analog-to-digital converter (ADC)
C281x CAP	Receive and log capture input pin transitions
C281x eCAN Receive	Enhanced Control Area Network receive mailbox
C281x eCAN Transmit	Enhanced Control Area Network transmit mailbox
C281x GPIO Digital Input	General-purpose I/O pins for digital input
C281x GPIO Digital Output	General-purpose I/O pins for digital output
C281x Hardware Interrupt	Interrupt Service Routine to handle hardware interrupt on C281x processor
C281x PWM	Pulse width modulators (PWMs)
C281x QEP	Quadrature encoder pulse circuit
C281x SCI Receive	Receive data on target via serial communications interface (SCI) from host
C281x SCI Transmit	Transmit data from target via serial communications interface (SCI) to host
C281x SPI Receive	Receive data via serial peripheral interface on target
C281x SPI Transmit	Transmit data via serial peripheral interface (SPI) to host
C281x SW Int Trigger	Generate software triggered nonmaskable interrupt
C281x Timer	Configure up to four general-purpose, stand alone Event Manager timers

From Memory

Retrieve data from target memory

Idle Task

Free-running task that executes  
downstream subsystem

To Memory

Write data to target memory

## C28x Digital Motor Control (c28xdmclib)

Clarke Transformation	Convert balanced three-phase quantities to balanced two-phase quadrature quantities
Inverse Park Transformation	Convert rotating reference frame vectors to two-phase stationary reference frame
Park Transformation	Convert two-phase stationary system vectors to rotating system vectors
PID Controller	Digital PID controller
Ramp Control	Create ramp-up and ramp-down function
Ramp Generator	Generate ramp output
Space Vector Generator	Duty ratios for stator reference voltage
Speed Measurement	Motor speed

## C28x IQmath (tiqmathlib)

Absolute IQN	Absolute value
Arctangent IQN	Four-quadrant arc tangent
Division IQN	Divide IQ numbers
Float to IQN	Convert floating-point number to IQ number
Fractional part IQN	Fractional part of IQ number
Fractional part IQN x int32	Fractional part of result of multiplying IQ number and long integer
Integer part IQN	Integer part of IQ number
Integer part IQN x int32	Integer part of result of multiplying IQ number and long integer
IQN to Float	Convert IQ number to floating-point number
IQN x int32	Multiply IQ number with long integer
IQN x IQN	Multiply IQ numbers with same Q format
IQN1 to IQN2	Convert IQ number to different Q format
IQN1 x IQN2	Multiply IQ numbers with different Q formats
Magnitude IQN	Magnitude of two orthogonal IQ numbers
Saturate IQN	Saturate IQ number
Square Root IQN	Square root or inverse square root of IQ number
Trig Fcn IQN	Sine, cosine, or arc tangent of IQ number





# Blocks — Alphabetical List

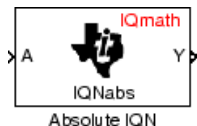
---

# Absolute IQN

**Purpose** Absolute value

**Library** `tiiqmathlib` in Target for TI C2000

**Description** This block computes the absolute value of an IQ number input. The output is also an IQ number.

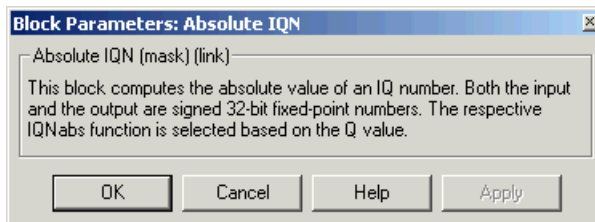


---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



**See Also** Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

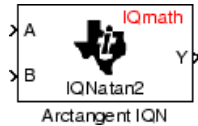
## Purpose

Four-quadrant arc tangent

## Library

tiiqmathlib in Target for TI C2000

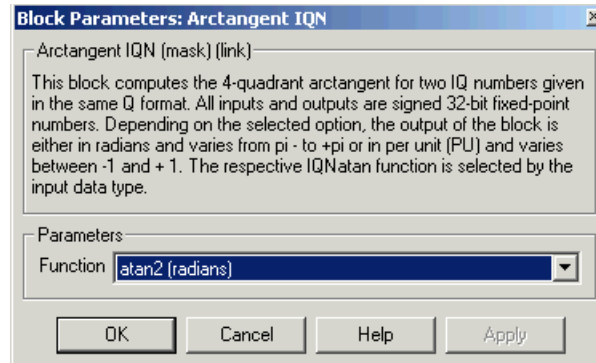
## Description



The Arctangent IQN block computes the four-quadrant arc tangent of the IQ number inputs and produces IQ number output.

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

## Dialog Box



## Function

Type of arc tangent to calculate:

- atan2 — Compute the four-quadrant arc tangent with output in radians with values from  $-\pi$  to  $+\pi$ .
- atan2PU — Compute the four-quadrant arc tangent per unit. If  $\text{atan2}(B,A)$  is greater than or equal to 0,  $\text{atan2PU}(B,A) = \text{atan2}(B,A) / 2 * \pi$ . Otherwise,  $\text{atan2PU}(B,A)$

# Arctangent IQN

---

=  $\text{atan2}(B,A)/2*\pi+1$ . The output is in per-unit radians with values from 0 to  $2*\pi$  radians.

---

**Note** The order of the inputs to the Arctangent IQN block correspond to the Texas Instruments convention, with argument 'A' at the top and 'B' at bottom.

---

## See Also

Absolute IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

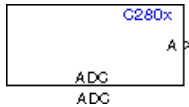
## Purpose

Analog-to-digital converter (ADC)

## Library

c280xdspchip1lib in Target for TI C2000

## Description



The C280x ADC block configures the C280x ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. The ADC block outputs digital values representing the analog input signal and stores the converted values in the result register of your digital signal processor. You use this block to capture and digitize analog signals from external sources such as signal generators, frequency generators, or audio devices.

## Output

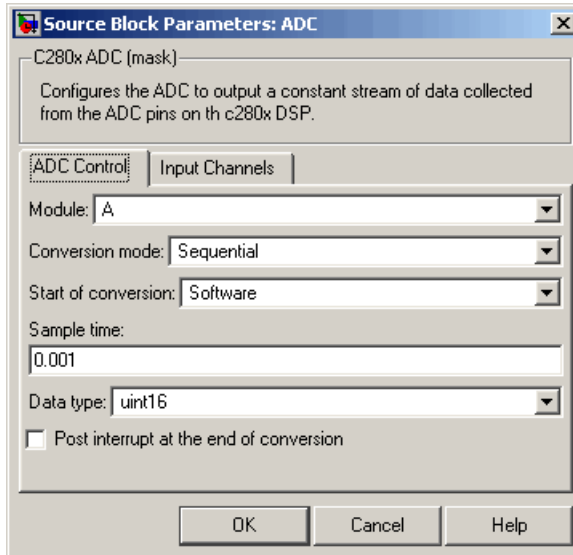
The output of the C280x ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the C280x ADC is 12-bit converter.

## Modes

The C280x ADC block supports ADC operation in dual and cascaded modes. In dual mode, either module A or module B can be used for the ADC block, and two ADC blocks are allowed in the model. In cascaded mode, both module A and module B are used for a single ADC block.

## Dialog Box

### ADC Control Pane



### Module

Specifies which DSP module to use:

- A — Displays the ADC channels in module A (ADCINA0 through ADCINA7).
- B — Displays the ADC channels in module B (ADCINB0 through ADCINB7).
- A and B — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7).

### Conversion mode

Type of sampling to use for the signals:

- Sequential — Samples the selected channels sequentially.

- Simultaneous — Samples the corresponding channels of modules A and B at the same time.

## Start of conversion

Type of signal that triggers conversions to begin:

- Software — Signal from software. Conversion values are updated at each sample time.
- ePWMxA / ePWMxB / ePWMxA\_ePWMxB — Start of conversion is controlled by user-defined PWM events.
- XINT2\_ADCSOC — Start of conversion is controlled by the XINT2\_ADCSOC external signal pin.

The choices available in **Start of conversion** depend on the **Module** setting. The following table summarizes the available choices. For each set of **Start of conversion** choices, the default is given first.

Module Setting	Start of Conversion Choices
A	Software, ePWMxA, XINT2_ADCSOC
B	ePWMxB, Software
A and B	Software, ePWMxA, ePWMxB, ePWMxA_ePWMxB, XINT2_ADCSOC

## Sample time

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. See “Scheduling and Timing” on page 1-13 for more information on timing. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt at the end of conversion** box, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

# C280x ADC

---

To set different sample times for different groups of ADC channels, you must add separate C280x ADC blocks to your model and set the desired sample times for each block.

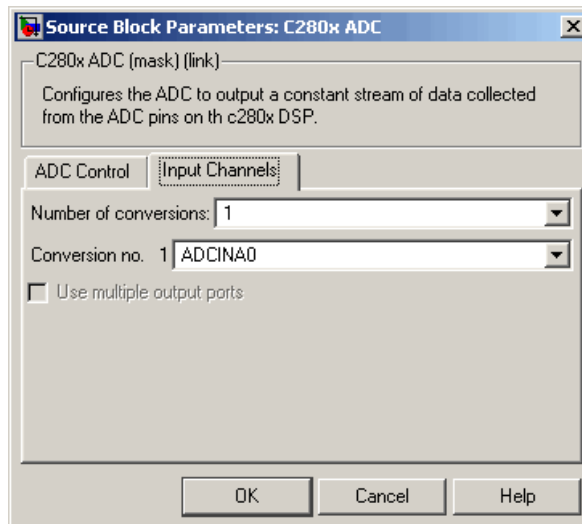
## Data type

Date type of the output data. Valid data types are auto, double, single, int8, uint8, int16, uint16, int32, or uint32.

## Post interrupt at the end of conversion

Select this check box to post an asynchronous interrupt at the end of each conversion. Note that the interrupt is always posted at the end of conversion. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

## Input Channels Pane



## Number of conversions

Number of ADC channels to use for analog-to-digital conversions.



**Conversion no.**

Specific ADC channel to associate with each conversion number.

In oversampling mode, a signal at a given ADC channel can be sampled multiple times during a single conversion sequence.

To oversample, specify the same channel for more than one conversion. Converted samples are output as a single vector.

**Use multiple output ports**

If more than one ADC channel is used for conversion, you can use separate ports for each output and show the output ports on the block. If you use more than one channel and do not use multiple output ports, the data is output in a single vector.

**See Also**

C280x ePWM, C280x Hardware Interrupt, “Configuring Acquisition Window Width for ADC Blocks”

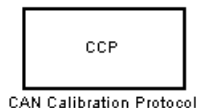
# CAN Calibration Protocol (C2000)

---

**Purpose** Implement CAN Calibration Protocol (CCP) standard

**Library** Target for TI C2000 Driver Library/ CAN Interface

**Description** The CAN Calibration Protocol (C2000) block provides an implementation of a subset of the CAN Calibration Protocol (CCP) Version 2.1. CCP is a protocol for communicating between the target processor and the host machine over CAN. In particular, a calibration tool (see ) running on the host can communicate with the target, allowing remote signal monitoring and parameter tuning.



This block processes a Command Receive Object (CRO) and outputs the resulting Data Transmission Object (DTO) and Data Acquisition (DAQ) messages.

For more information on CCP, refer to *ASAM Standards: ASAM MCD: MCD 1a* on the Association for Standardization of Automation and Measuring Systems (ASAM) Web site at <http://www.asam.de>.

## Using the DAQ Output

---

**Note** The CCP Data Acquisition (DAQ) List mode of operation is only supported with Real-Time Workshop Embedded Coder. If Embedded Coder is not available then custom storage classes `canlib.signal` are ignored during code generation: this means that the CCP DAQ Lists mode of operation cannot be used.

You can use the CCP Polling mode of operation with or without Real-Time Workshop Embedded Coder.

---

The DAQ output is the output for any CCP Data Acquisition (DAQ) lists that have been set up. You can use the ASAP2 file generation feature of the Real-Time (RT) target to

- Set up signals to be transmitted using CCP DAQ lists.

- Assign signals in your model to a CCP event channel automatically (see ).

Once these signals are set up, event channels then periodically fire events that trigger the transmission of DAQ data to the host. When this occurs, CAN messages with the appropriate CCP/DAQ data appear on the DAQ output, along with an associated function call trigger.

The calibration tool (see ) must use CCP commands to assign an event channel and data to the available DAQ lists, and interpret the synchronous response.

Using DAQ lists for signal monitoring has the following advantages over the polling method:

- There is no need for the host to poll for the data. Network traffic is halved.
- The data is transmitted at the correct update rate for the signal. Therefore, there is no unnecessary network traffic generated.
- Data is guaranteed to be consistent. The transmission takes place after the signals have been updated, so there is no risk of interruptions while sampling the signal.

---

**Note** Target for TI C2000 does not currently support event channel prescalers.

---

## Dialog Box

### CAN station address (16 bit integer)

The station address of the target. The station address is interpreted as a uint16. It is used to distinguish between different targets. By assigning unique station addresses to targets sharing the same CAN bus, it is possible for a single host to communicate with multiple targets.

# CAN Calibration Protocol (C2000)

---

## **CAN module**

Choose CAN module A or B.

## **CAN message identifier (CRO)**

Specify the CAN message identifier for the Command Receive Object (CRO) message you want to process.

## **CAN message type (CRO)**

The incoming message type. Select either Standard(11-bit identifier) or Extended(29-bit identifier).

## **CAN message identifier (DTO/DAQ)**

The message identifier is the CAN message ID used for Data Transmission Object (DTO) and Data Acquisition (DAQ) message outputs.

## **CAN message type (DTO/DAQ)**

The message type to be transmitted by the DTO and DAQ outputs. Select either Standard(11-bit identifier) or Extended(29-bit identifier).

## **Total number of Object Descriptor Tables (ODTs)**

The default number of Object Descriptor Tables (ODTs) is 8. These ODTs are shared equally between all available DAQ lists. You can choose a value between 0 and 254, depending on how many signals you wish to log simultaneously. You must make sure you allocate at least 1 ODT per DAQ list, or your build will fail. The calibration tool will give an error message if there are too few ODTs for the number of signals you specify for monitoring. Be aware that too many ODTs can make the sample time overrun. If you choose more than the maximum number of ODTs (254), the build will fail.

A single ODT uses 56 bytes of memory. Using all 254 ODTs would require over 14 KB of memory, a large proportion of the available memory on the target. To conserve memory on the target, the default number is low, allowing DAQ list signal monitoring with reduced memory overhead and processing power.

As an example, if you have five different rates in a model, and you are using three rates for DAQ, then this will create three DAQ lists and you must make sure you have at least three ODTs. ODTs are shared equally among DAQ lists and, therefore, you will end up with one ODT per DAQ list. With less than three ODTs, you get zero ODTs per DAQ list and the behavior is undefined.

Taking this example further, say you have three DAQ lists with one ODT each, and start trying to monitor signals in a calibration tool. If you try to assign too many signals to a particular DAQ list (that is, signals requiring more space than seven bytes (one ODT) in this case), then the calibration tool will report this as an error.

### **CRO sample time**

The sample time for CRO messages.

### **Supported CCP Commands**

The following CCP commands are supported by the CAN Calibration Protocol (C2000) block:

- CONNECT
- DISCONNECT
- DNLOAD
- DNLOAD\_6
- EXCHANGE\_ID
- GET\_CCP\_VERSION
- GET\_DAQ\_SIZE
- GET\_S\_STATUS
- SET\_DAQ\_PTR
- SET\_MTA
- SET\_S\_STATUS
- SHORT\_UP

# CAN Calibration Protocol (C2000)

---

- START\_STOP
- START\_STOP\_ALL
- TEST
- UPLOAD
- WRITE\_DAQ

## **Compatibility with Calibration Packages**

The above commands support

- Synchronous signal monitoring via calibration packages that use DAQ lists
- Asynchronous signal monitoring via calibration packages that poll the target
- Asynchronous parameter tuning via CCP memory programming

This CCP implementation has been tested successfully with the Vector-Informatik CANape calibration package running in both DAQ list and polling mode, and with the Accurate Technologies, Inc., Vision, calibration package running in DAQ list mode. (Note that Accurate Technologies, Inc., Vision does not support the polling mechanism for signal monitoring).

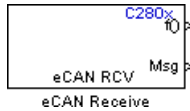
## Purpose

Enhanced Control Area Network receive mailbox

## Library

c280xdspchip1lib in Target for TI C2000

## Description



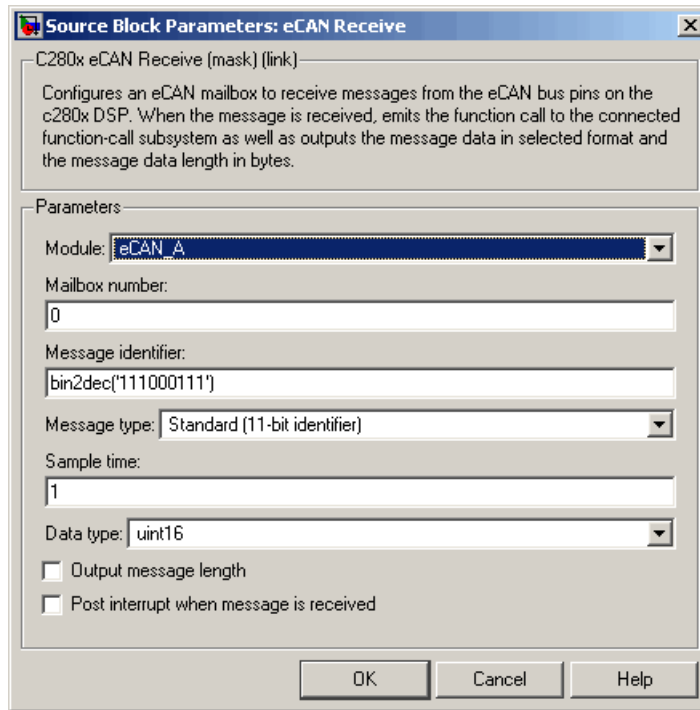
The C280x enhanced Control Area Network (eCAN) Receive block generates source code for receiving eCAN messages through an eCAN mailbox. The eCAN modules on the DSP chip provide serial communication capability and have 32 mailboxes configurable for receive or transmit. The C280x supports eCAN data frames in standard or extended format.

The C28x eCAN Receive block has up to two and, optionally, three output ports.

- The first output port is the function call port, and a function call subsystem should be connected to this port. When a new message is received, this subsystem is executed.
- The second output port is the message data port. The received data is output in the form of a vector of elements of the selected data type. The length of the vector is always 8 bytes. The message data port will always output data. When the block is used in polling mode, if there is no new message created between the consecutive executions of the block, then the old message, or the existing message, is repeated.
- The third output port is optional and appears only if **Output message length** is selected.

# C280x eCAN Receive

## Dialog Box



### Module

Determines which of the two eCAN modules is being configured by this instance of the C280x eCAN Receive block. Options are eCAN\_A and eCAN\_B.

### Mailbox number

Unique number from 0 to 15 for standard or from 0 to 31 for enhanced CAN mode. It refers to a mailbox area in RAM. In standard mode, the mailbox number determines priority.

### Message identifier

Identifier of length 11 bits for standard frame size or length 29 bits for extended frame size in decimal, binary, or hex. If in binary or hex, use `bin2dec(' ')` or `hex2dec(' ')`, respectively, to convert the entry. The message identifier is associated with a



receive mailbox. Only messages that match the mailbox message identifier are accepted into it.

## Message type

Select Standard (11-bit identifier) or Extended (29-bit identifier).

## Sample time

Frequency with which the mailbox is polled to determine if a new message has been received. A new message causes a function call to be emitted from the mailbox. If you want to update the message output only when a new message arrives, then the block needs to be executed asynchronously. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt when message is received** box, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

---

**Note** For information about setting the timing parameters of the CAN module see “Configuring Timing Parameters for CAN Blocks”.

---

## Data type

Type of data in the data vector. The length of the vector for the received message is at most 8 bytes. If the message is less than 8 bytes, the data buffer bytes are right-aligned in the output. Only uint16 (vector length = 4 elements) or uint32 (vector length = 8 elements) data are allowed. The data are unpacked as follows using the data buffer, which is 8 bytes.

For uint16 data,

```
Output[0] = data_buffer[1..0];
Output[1] = data_buffer[3..2];
Output[2] = data_buffer[5..4];
Output[3] = data_buffer[7..6];
```

# C280x eCAN Receive

---

For uint32 data,

```
Output[0] = data_buffer[3..0];  
Output[1] = data_buffer[7..4];
```

For example, if the received message has two bytes,

```
data_buffer[0] = 0x21  
data_buffer[1] = 0x43
```

the uint16 output would be:

```
Output[0] = 0x4321  
Output[1] = 0x0000  
Output[2] = 0x0000  
Output[3] = 0x0000
```

## **Output message length**

Select to output the message length in bytes to the third output port. If not selected, the block has only two output ports.

## **Post interrupt when message is received**

Select this check box to post an asynchronous interrupt when a message is received.

## **References**

Detailed information on the eCAN module is in *TMS320x281x, 280x Enhanced Controller Area Network (eCAN) Reference Guide (Rev. D)*, Literature Number SPRU074D, available at the Texas Instruments Web site.

## **See Also**

C280x eCAN Transmit, C280x Hardware Interrupt

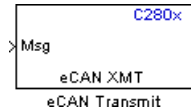
## Purpose

Enhanced Control Area Network transmit mailbox

## Library

c280xdspchiplib in Target for TI C2000

## Description



The C280x enhanced Control Area Network (eCAN) Transmit block generates source code for transmitting eCAN messages through an eCAN mailbox. The eCAN modules on the DSP chip provide serial communication capability and have 32 mailboxes configurable for receive or transmit. The C280x supports eCAN data frames in standard or extended format.

---

**Note** Fixed-point inputs are not supported for this block.

---

## Data Vectors

The length of the vector for each transmitted mailbox message is 8 bytes. Input data are always right-aligned in the message data buffer. Only `uint16` (vector length = 4 elements) or `uint32` (vector length = 8 elements) data are accepted. The following examples show how the different types of input data are aligned in the data buffer:

For input of type `uint32`,

```
inputdata [0] = 0x12345678
```

the data buffer is:

```
data buffer[0] = 0x78
data buffer[1] = 0x56
data buffer[2] = 0x34
data buffer[3] = 0x12
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

# C280x eCAN Transmit

---

For input of type uint16,

```
inputdata [0] = 0x1234
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x00
data buffer[3] = 0x00
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

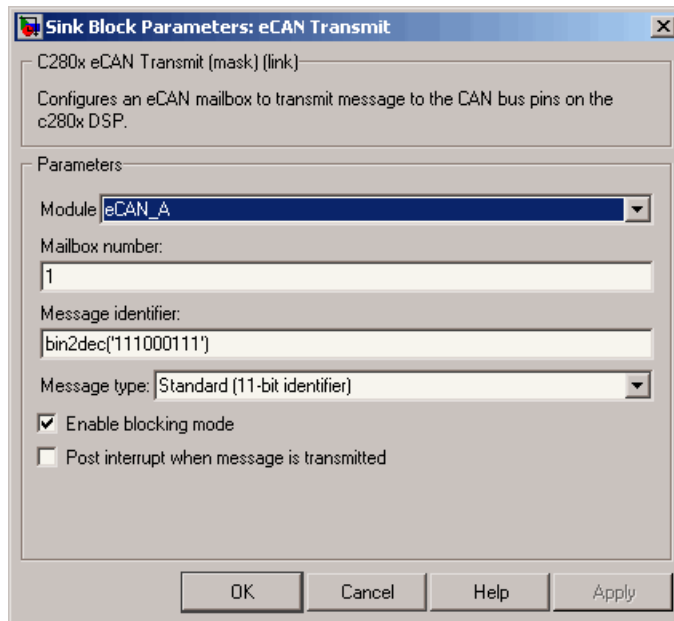
For input of type uint16[2], which is a two-element vector,

```
inputdata [0] = 0x1234
inputdata [1] = 0x5678
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x78
data buffer[3] = 0x56
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

## Dialog Box



### Module

Determines which of the two eCAN modules is being configured by this instance of the C280x eCAN Transmit block. Options are eCAN\_A and eCAN\_B.

### Mailbox number

Unique number from 0 to 15 for standard or from 0 to 31 for enhanced CAN mode. It refers to a mailbox area in RAM. In standard mode, the mailbox number determines priority.

### Message identifier

Identifier of length 11 bits for standard frame size or length 29 bits for extended frame size in decimal, binary, or hex. If in binary or hex, use `bin2dec(' ')` or `hex2dec(' ')`, respectively, to convert the entry. The message identifier is coded into a message that is sent to the CAN bus.

# C280x eCAN Transmit

---

## Message type

Select Standard (11-bit identifier) or Extended (29-bit identifier).

## Enable blocking mode

If selected, the CAN block code waits indefinitely for a transmit (XMT) acknowledge. If not selected, the CAN block code does not wait for a transmit (XMT) acknowledge, which is useful when the hardware might fail to acknowledge transmissions.

## Post interrupt when message is transmitted

If selected, an asynchronous interrupt will be posted when data is transmitted.

---

**Note** For information about setting the timing parameters of the CAN module see “Configuring Timing Parameters for CAN Blocks”.

---

## References

Detailed information on the eCAN module is in *TMS320x281x, 280x Enhanced Controller Area Network (eCAN) Reference Guide (Rev. D)*, Literature Number SPRU074D, available at the Texas Instruments Web site.

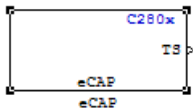
## See Also

C280x eCAN Receive

**Purpose** Receive and log capture input pin transitions or configure auxiliary pulse width modulator

**Library** c280xdspchip1lib in Target for TI C2000

## Description



## Dialog Box

The eCAP block dialog box provides configuration parameters on four tabbed panes:

- **General**—Set the operating mode for the block (whether the block performs eCAP or APWM processes, assign the pin associated, and set the sample time)
- **eCAP**—Configure eCAP functions such as prescaler value, capture pin, and mode control
- **APWM**—Configure waveform and duty cycle values for the pulse width modulation capability
- **Interrupt**—Specify when the block posts interrupts

You can add up to four C280x eCAP blocks to your model, one block for each capture pin. For example, you can have one block configured for eCAP mode with eCAP1 pin selected and three blocks configured for APWM mode with assigned pins eCAP2, eCAP3, and eCAP4. Or four blocks configured for eCAP mode with each block assigned a different eCAP pin. You cannot assign the same eCAP pin to two eCAP blocks in one model.

## Block Input and Output Ports

The C280x eCAP block has optional input and output ports as shown in the following table.

<b>Port</b>	<b>Description and When the Port is Enabled</b>
Input port SI	Synchronization input for input value from software. Enabled when you select <b>Enable software forced counter synchronizing input</b> in either operating mode.
Input port RA	One-shot arming starts the one-shot sequence. Enabled when you set the mode control to One shot.
Output port TS	When you enable the reset counter, this option resets the capture event counter after capturing the event time stamp. Enabled when you select <b>Enable reset counter after capture event1 time-stamp</b> .
Output port CF	This port reports the status of the capture event. Enabled when you select <b>Enable capture event status flag output</b> .
Output port OF	Enabled when you select <b>Enable overflow status flag output</b> .

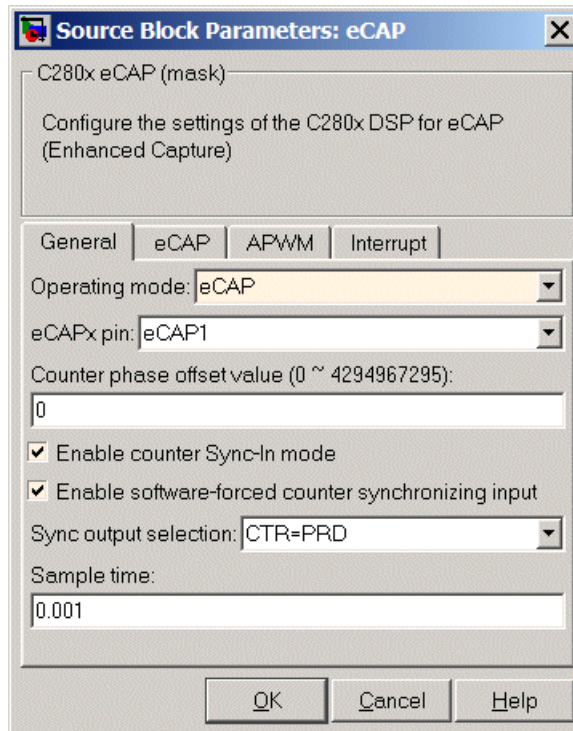
---

**Note** The outputs of this block can be vectorized.

---



## General Pane



### Operating mode

Select either eCAP or APWM from the list. Selecting eCAP puts the block in capture mode for the capture units. The capture units log pin transitions by logging the transitions in to a FIFO buffer. In APWM mode, the block generates asymmetric pulse width modulation (APWM) waveforms for driving downstream systems.

### eCAPx pin

The capture unit includes the following features:

- Four capture input pins—eCAP1 through eCAP4—one pin for each capture unit on the C281x processor.

- One maskable interrupt flag for each capture unit for a total of four flags.
- Ability to specify the transition detection—rising edge, falling edge, or both edges.

### **Counter phase offset value (0~4294967295)**

The value you enter here provide the time base for event captures, clocked by the system clock. A phase register is used to synchronize with other counters via the software or hardware forced sync (refer to **Enable counter Sync-In mode**). This is particularly useful in APWM mode when you need a phase offset between capture modules. Enter the phase offset as an integer from 0 (no offset) to 42949667295 ( $2^{32}$ ) counts.

### **Enable counter Sync-In mode**

Select this to enable the TSCTR counter to load from the TSCTR register when the block receives either the SYNC1 signal or a software force event (refer to **Enable software-forced counter synchronizing input**).

### **Enable software-forced counter synchronizing input**

This option provides a convenient software method for synchronizing one or more eCAP time bases.

### **Sync output selection**

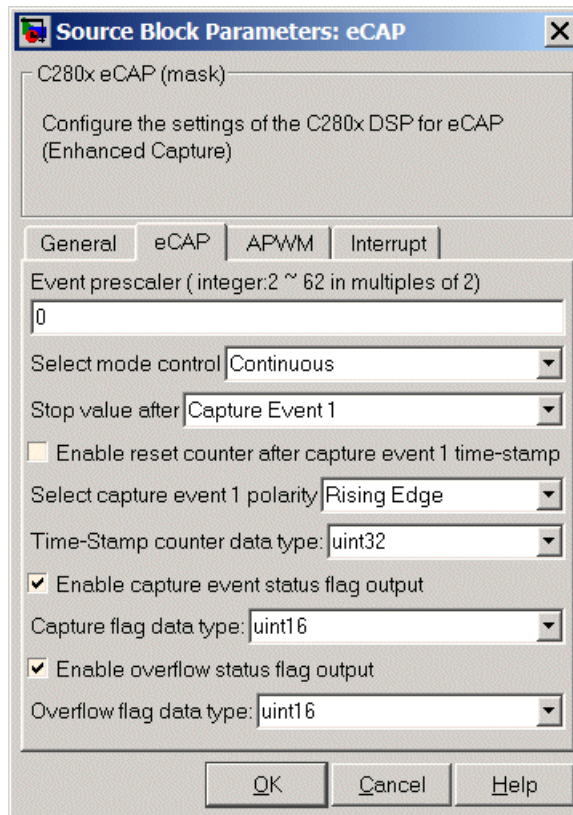
Select one of the list entries Pass through, CTR=PRD, or Disabled to synchronize with other counters.

### **Sample time**

Set the sample time for the block in seconds.

### **eCAP Pane**

To enable the configuration parameters on this pane, select eCAP from the **Operating mode** list on the **General** pane.



### Event prescaler (integer from 0 to 31)

You can prescale an input capture signal, called a pulse train, by a value that you set here. Enter an integer between 0 and 31. Entering a 0 bypasses the input prescaler, leaving the input capture signal unchanged.

### Select mode control

Provides continuous and one-shot mode control operations. The default setting of continuous mode enables continuous time-stamp captures using a circular buffer that captures events 1 through 4.

One shot mode disable continuous mode and enables the **Enable one-shot rearming control via input port** option so you can select it.

### **Enable one-shot rearming control via input port**

Select this to arm the one-shot sequence:

- 1** Reset the Mod4 counter to zero.
- 2** Unfreeze the Mod4 counter.
- 3** Enable capture register loading.

### **Stop value after**

#### **Enable reset counter after capture event 1 time-stamp**

Enables a reset after capture event 1. When you select this option, the eCAP process resets the counters after receiving a capture event 1 time-stamp.

#### **Select capture event 1 polarity**

This setting determines when the capture event triggers. Select Rising edge or Falling edge from the list.

#### **Time-Stamp counter data type**

Select the data type to represent the counter. The list includes integer and unsigned 8-, 16-, and 32-bit data types, double, single, and Boolean. Select one on the list that meets your needs.

#### **Enable capture event status flag output**

Select to output the capture event status flag. The block outputs a 0 until the event capture. After the event the flag value is one.

#### **Overflow capture event flag data type**

Select the data type to represent the capture event flag. The list includes integer and unsigned 8-, 16-, and 32-bit data types, double, single, and Boolean. Select one on the list that meets your needs.

## **Enable overflow status flag output**

Select to output the status of the elements of the FIFO buffer.

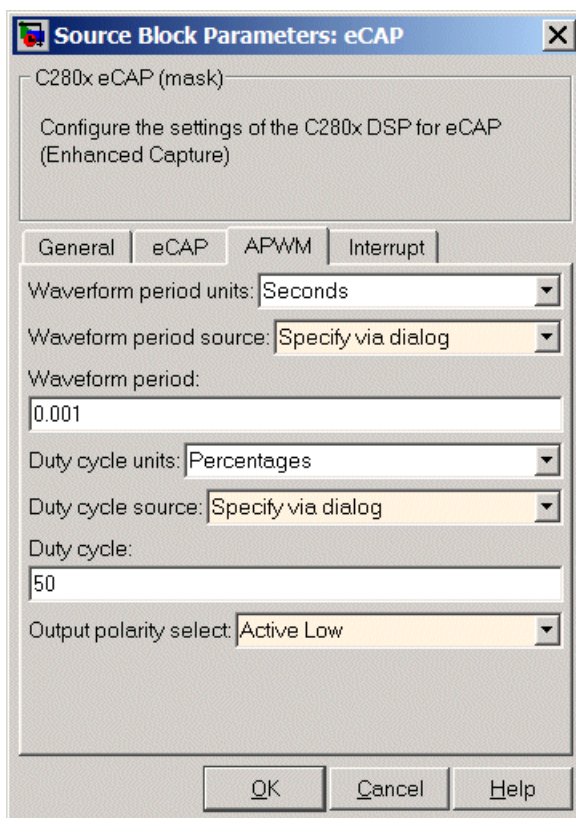
After you select this, set the data type for the flag in **Overflow flag data type**.

## **Overflow flag data type**

Select the data type to represent the status flag. The list includes integer and unsigned 8-, 16-, and 32-bit data types, double, single, and Boolean. Select one on the list that meets your needs.

## **APWM Pane**

To enable the configuration parameters on this pane, select APWM from the **Operating mode** list on the **General** pane.



### Waveform period units

Units in which to measure the waveform period. Options are **Clock cycles**, which refer to the high-speed peripheral clock on the F2812 chip (75 MHz), or **Seconds**. Note that changing these units changes the **Waveform period** value and the **Duty cycle** value and **Duty cycle units** selection.

### Waveform period source

Source from which the waveform period value is obtained. Select **Specify via dialog** to enter the value in **Waveform period** or select **Input port** to use a value from the input port.

## Waveform period

Period of the PWM waveform measured in clock cycles or in seconds, as specified in the **Waveform period units**.

---

**Note** The term *clock cycles* refers to the high-speed peripheral clock on the F2812 chip. This clock is 75 MHz by default because the high-speed peripheral clock prescaler is set to 2 (150 MHz/2).

---

## Duty cycle units

Units for the duty cycle. Select `Clock cycles` or `Percentages` from the list. Changing these units changes the **Duty cycle** value, the **Waveform period** value, and **Waveform period units** selection.

## Duty cycle source

Source from which the duty cycle for the specific PWM pair is obtained. Select `Specify via dialog` to enter the value in **Duty cycle** or select `Input port` to use a value from the input port.

## Duty cycle

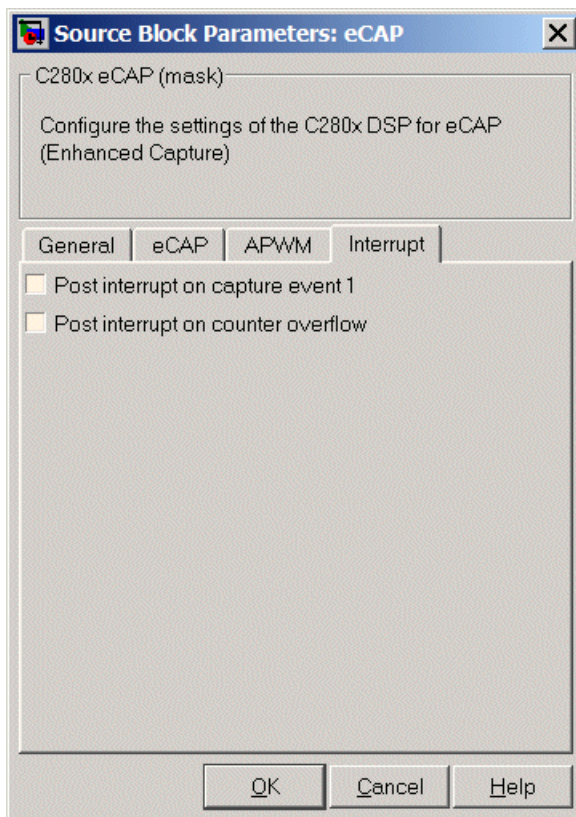
Ratio of the PWM waveform pulse duration to the PWM waveform period expressed in **Duty cycle units**.

## Output polarity select

Set the active level for the output. Choose `Active High` or `Active Low` from the list. When you select `Active High`, the compare value defines the high time. Selecting `Active Low` directs the compare value to define the low time.

## Interrupt Pane

In the following figure, you see the interrupt options when you put the block in eCAP mode by selecting eCAP for **Operating mode** on the **General** pane.



### **Post interrupt on capture event 1**

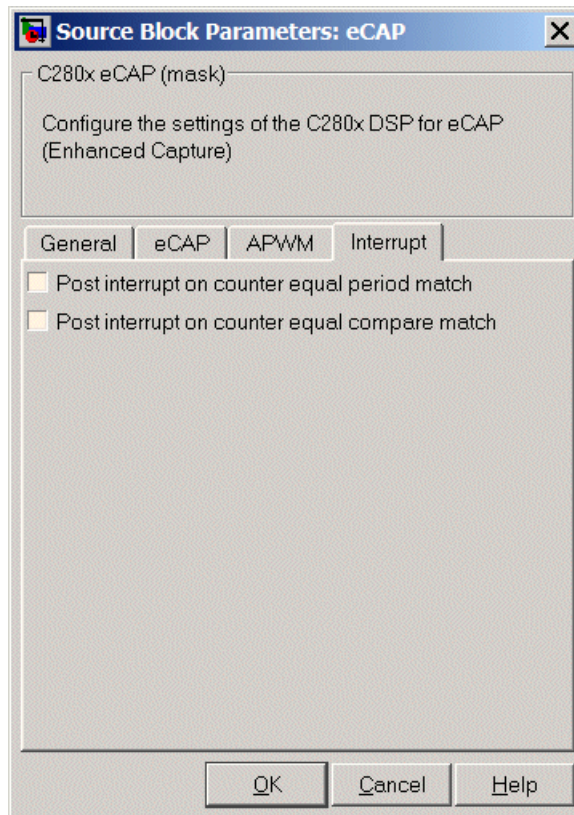
Enables capture event 1 as in interrupt source.

### **Post interrupt on counter overflow**

Enables counter overflow as an interrupt source.

The next figure presents the interrupt options when you put the block in APWM mode by selecting APWM for **Operating mode** on the **General** pane.





**Post interrupt on counter equal period match**

**Post interrupt on counter equal compare match**

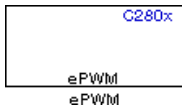
# C280x ePWM

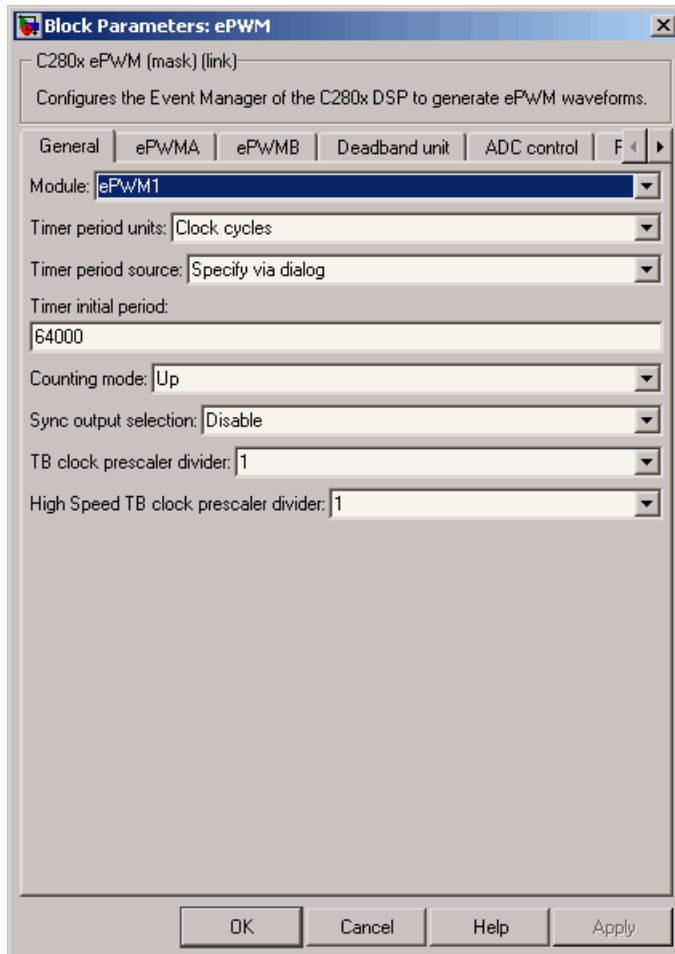
---

**Purpose** Configure C280x Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms

**Library** `c280xdspchip1lib` in Target for TI C2000

**Description** A C280x system contains multiple ePWM modules, each having two PWM outputs. The C280x ePWM block lets you configure up to six ePWM modules.



**Dialog  
Box****General Pane****Module**

Specifies which target ePWM module to use. Possible values are ePWM1 through ePWM6.

## Timer period units

Specifies the units in which the **Waveform period** is expressed. Choose Seconds (the default) or Clock cycles. The period register is a uint16, so when seconds are used for the **Timer period units** (a double) a conversion must be done. For best performance, It is recommended that you use clock cycles here as there will be fewer calculations and less risk of round-off error. For example, on the C2808 the PWM module is based on the system clock, or SYSCLOCK/2 (100/2 MHz = 50 MHz) so the compare value and the period register must be calculated using this timing.

## Timer period source

Source from which the waveform period value is obtained. Select Specify via dialog to enter the value in **Waveform period** or select Input port to use a value from the input port.

## Timer initial period

Period of the PWM waveform measured in clock cycles or in seconds, as specified in **Waveform period units**.

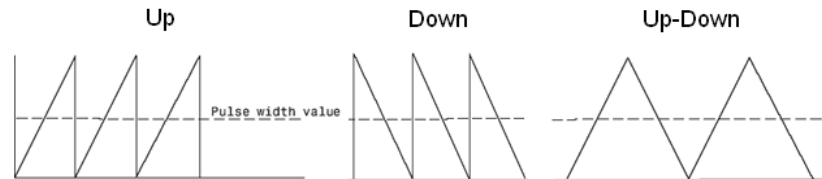
---

**Note** The term *clock cycles* refers to the Time-base Clock on the C280x chip. See the discussion of the **TB clock prescaler divider** below for an explanation of how the Time-base Clock speed is calculated.

---

## Counting mode

Specifies the counting mode in which to operate. C280x PWMs can operate in three distinct counting modes: Up, Down, and Up-Down. The following illustration shows the waveforms that correspond to these three modes:



### Sync output selection

Specifies the source that generates the EPWMxSYNCO signal, if any. The available choices are EPWMxSYNCI or SWFSYNC, CTR=Zero, CTR=CMPB, and Disable (the default).

### Enable S/W sync input port

This check box appears only when you choose EPWMxSYNCI or SWFSYNC in **Sync output selection**. Check to enable the input port.

### Enable phase offset source

Determines whether the ePWM module will use a phase offset and, if so, its source. Choices are Input port (the default), Specify via dialog, and Disable.

### Phase offset value

This field appears only when you select Specify via dialog in **Enable phase offset source**. Enter the counter phase offset value relative to the time-base that is supplying the sync-in signal.

### TB clock prescaler divider

This value, together with the **High Speed TB clock prescaler divider** value, determine the clock speed of the Time-Base submodule, which provides all event timing for the ePWM. The Time-base Clock's speed (TBCLK) is the result of dividing the system clock speed by the product of the **High Speed TB clock prescaler divider** (HSPCLKDIV) and the **TB clock prescaler divider** (CLKDIV) as in the following formula:

$$TBCLK = SYSCLKOUT / (HSPCLKDIV * CLKDIV)$$

Because the default values for both the **High Speed TB clock prescaler divider** and the **High Speed TB clock prescaler divider** are both 1, the default value of the Time-base Clock is equal to the system clock speed of 100 MHz

Choices are 1, 2, 4, 8, 16, 32, 64, and 128.

### **High Speed TB clock prescaler divider**

See the discussion of the **TB clock prescaler divider** above for an explanation of this value's role in setting the speed of the Time-base Clock. Choices are 1, 2, 4, 6, 8, 10, 12, and 14.

### **ePWMA and ePWMB panes**

The **ePWMA output** pane and **ePWMB output** pane include the same settings, although the default value is different in some cases, as noted below.

**Block Parameters: ePWM** [X]

C280x ePWM (mask) (link)  
Configures the Event Manager of the C280x DSP to generate ePWM waveforms.

General ePWMA ePWMB Deadband unit ADC control F < >

Enable ePWM1A

CMPA units: Percentages

CMPA source: Specify via dialog

CMPA value: 50

Action when counter=ZERO: Do nothing

Action when counter=PRD: Clear

Action when counter=CMPA on CAU: Set

Action when counter=CMPA on CAD: Do nothing

Action when counter=CMPB on CBU: Do nothing

Action when counter=CMPB on CBD: Do nothing

Compare value reload condition: Load on CTR=Zero

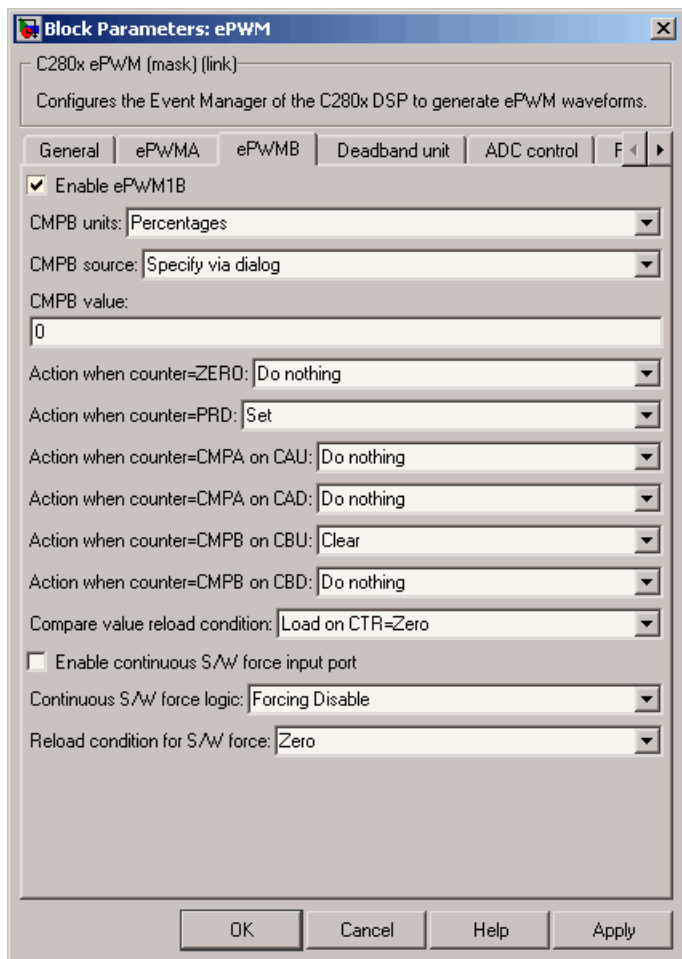
Enable continuous S/W force input port

Continuous S/W force logic: Forcing Disable

Reload condition for S/W force: Zero

Enable HRPWM

OK Cancel Help Apply



## Enable ePWMxA

## Enable ePWMxB

Select to enable the ePWMA and/or ePWMB output signals for the module that is currently chosen in the **General** pane. By default, both **Enable ePWMxA** and **Enable ePWMxB** are selected for each of the six ePWM modules you can select in the **General** pane.



## Use deadband for ePWMxA

## Use deadband for ePWMxB

Enables a deadband area of no signal overlap between pairs of ePWM output signals. In all cases, this check box is cleared by default.

## Duty cycle units

Specifies the units in which the **Duty cycle** value is expressed: Percentages (the default) or Clock cycles.

---

**Note** Using percentages may cause some additional computation time in generated code. This may or may not be noticeable in your application.

---

## Duty cycle source

Specifies the source from which the pulse width is to be obtained. Choose Specify via dialog (the default) to enter a value in the **Duty cycle** field, or Input port to use a value from the input port.

## Duty cycle

This field appears only when you choose Specify via dialog in **Duty cycle source**. Enter a value that specifies the pulse width, in the units specified in **Duty cycle units**.

## Action when counter=ZERO

## Action when counter=PRD

## Action when counter=CMPA on CAU

## Action when counter=CMPA on CAD

## Action when counter=CMPB on CBU

## Action when counter=CMPB on CBD

These settings, along with the other remaining settings in the **ePWMA output** and **ePWMB output** panes, determine the behavior of the Action Qualifier (AQ) submodule. Based on these settings, the AQ module decides which events are converted into

various action types, thereby producing the required switched waveforms of the ePWMxA and ePWMxB output signals.

For each of these four fields, the available choices are Do nothing, Clear, Set, and Toggle.

The default values for these fields vary between the **ePWMA output** and **ePWMB output** panes. The following table shows the defaults for each of these panes:

Action when counter=...	ePWMA output pane	ePWMB output pane
ZERO	Clear	Do nothing
PRD	Do nothing	Set
CMPA on CAU	Set	Do nothing
CMPA on CAD	Do nothing	Do nothing
CMPB on CBU	Do nothing	Clear
CMPB on CBD	Do nothing	Do nothing

For a detailed discussion of the AQ submodule, see the *TMS320x280x Enhanced Pulse Width Modulator (ePWM) Module Reference Guide* (SPRU791), available on the Texas Instruments Web site.

**Compare value reload condition**  
**Enable continuous S/W force input port**  
**Continuous S/W force logic**  
**Reload condition for S/W force**

These four settings determine how the AQ module handles the S/W force event, an asynchronous event initiated by software (CPU) via control register bits.

**Compare value reload condition** determines if and when the Action-qualifier S/W Force Register is reloaded from a shadow

register. Choices are Load on CTR=Zero (the default), Load on CTR=PRD, Load on either, and Freeze.

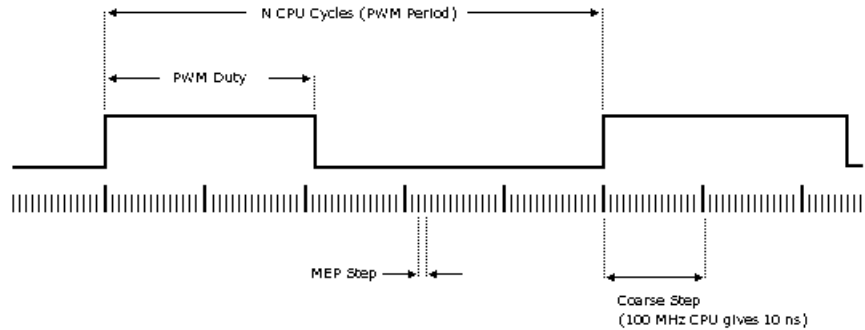
**Enable continuous S/W force input port** specifies the source from which the control logic is obtained. This check box is cleared by default. Select this check box to obtain the control logic from the input port

**Continuous S/W force logic** specifies what type of S/W force logic to use if the continuous S/W force input port is not enabled. Choices are Forcing Disable (the default), Forcing Low, and Forcing High.

**Reload condition for S/W force** — Choices are Zero (the default), Period, Either period or zero, and Immediate.

## **Enable HRPWM**

Select to enable High Resolution PWM settings. When the effective resolution for conventionally generated PWM is insufficient, you may want to consider High Resolution PWM (HRPWM). The resolution of PWM is normally dependent upon the PWM frequency and the underlying system clock frequency. To address this limitation, HRPWM uses **Micro Edge Positioner (MEP)**<sup>™</sup> technology to position edges more finely by dividing each coarse system clock. The accuracy of the subdivision is on the order of 150ps. The relationship between one system clock and edge position in terms of **MEP** steps is shown in the following figure:



MEP scale factor = Number of MEP steps in one coarse step

## HRPWM loading mode

Specify loading mode for HRPWM. This selects the time event that loads the CMPAHR shadow value into the active register.

## HRPWM control mode

Specify control mode for HRPWM. The **MEP** can be controlled using duty cycle control from the CMPAHR register, or using phase control from the TBPHSHR register. Rising edge or falling edge should be controlled from the CMPAHR register. For control of both edges, use the TBPHSHR register.

## HRPWM edge control mode

Specify edge of the PWM that is controlled by the micro-edge positioner™ (**MEP**) logic.

## CMPAHR

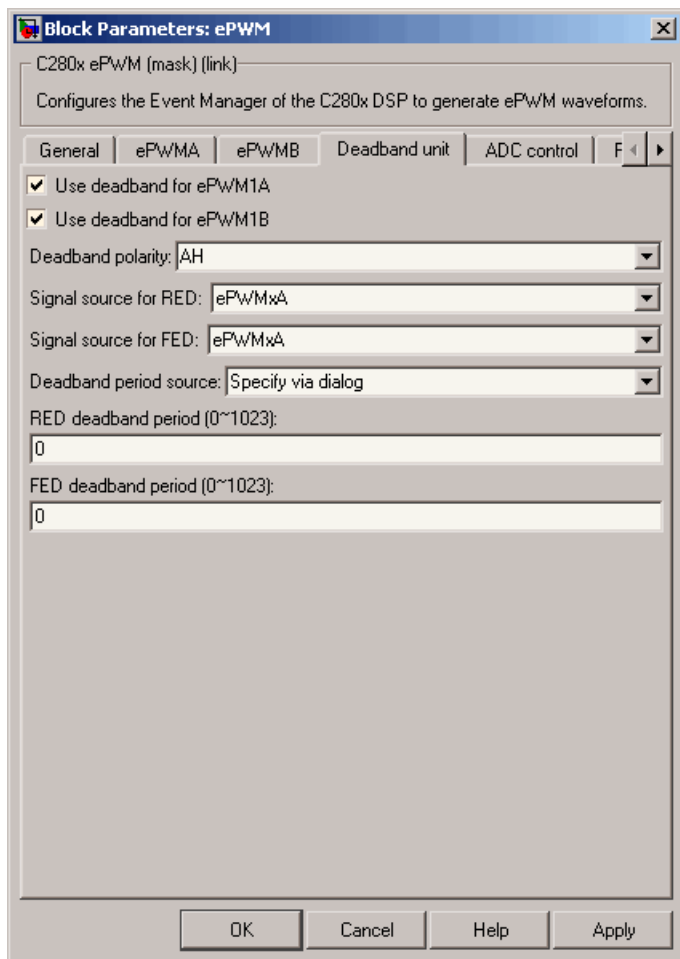
Specify Compare A (High Resolution) register

## Enable scale factor optimizer software™

Select to enable scale factor optimizer (SFO) software. The TI-supplied **MEP** scale factor optimizer software functions help to determine dynamically the optimum step size for the **MEP** based on operating temperature and voltage. It is recommended that applications that use the HRPWM feature should use the SFO software.

## Deadband Unit Pane

The **Deadband unit** pane lets you specify parameters for the Dead-Band Generator (DB) submodule. Since using the DB submodule is not required for generating a deadband in PWM output, this pane is empty by default. The elements of the **Deadband unit** pane shown in the following image appear only when you select either or both of the **Use deadband for ePWMxA** or **Use deadband for ePWMxB** check boxes in the **ePWMA output** or **ePWMB output** panes.



## Deadband polarity

Configures the deadband polarity as AH (active high, the default), AL (active low), AHC (active high complementary), or ALC (active low complementary).

## **Deadband period source**

Specifies the source from which the control logic is to be obtained. Choose `Specify via dialog` (the default) to enter explicit values, or `Input port` to use a value from the input port.

## **RED deadband period**

This field appears only when **Use deadband for ePWMxA** is selected in the **ePWMA output** pane. Enter a value from 0 to 1023 to specify a rising edge delay.

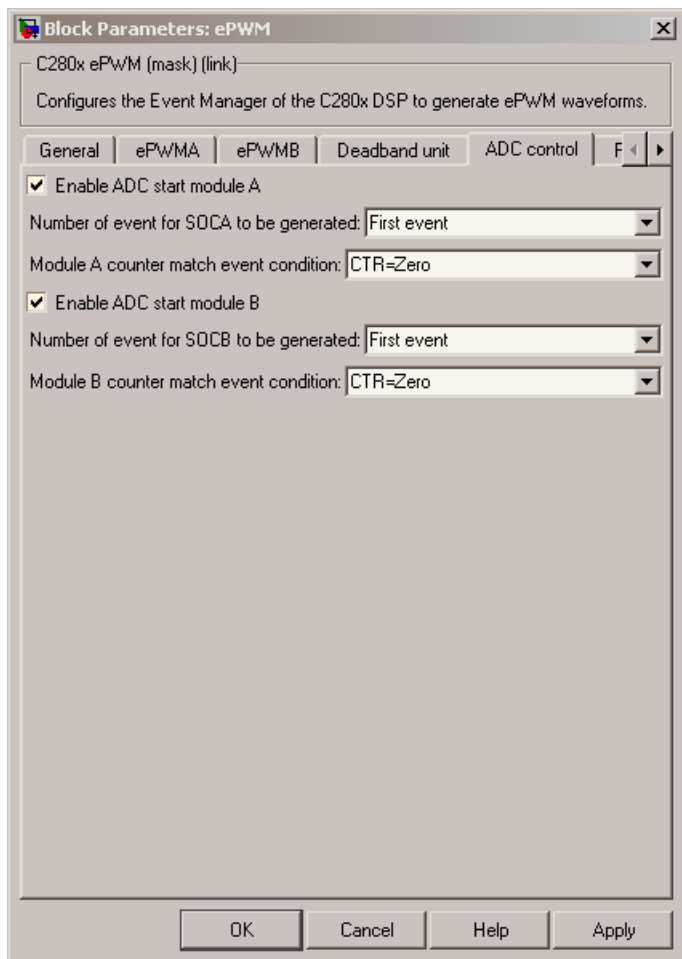
## **FED deadband period**

This field appears only when **Use deadband for ePWMxB** is selected in the **ePWMB output** pane. Enter a value from 0 to 1023 to specify a falling edge delay.

## **ADC Control Pane**

The **ADC control** pane lets you specify conditions under which ADC start of conversion is triggered by either or both of the ePWMA and ePWMB outputs.

# C280x ePWM



## Enable ADC start module A

Select to allow ePWMA to trigger ADC start of conversion. This check box is cleared by default.



## **Number of event for SOCA to be generated**

This field appears only when you check the **Enable ADC start module A** check box. Specify how often you want ADC start of conversion to be triggered. First event triggers ADC start of conversion with every event, Second event triggers ADC start of conversion with every second event, and Third event triggers ADC start of conversion with every third event.

## **Module A counter match event condition**

This field also appears only when you select the **Enable ADC start module A** check box. Specify the counter match condition that will trigger an ADC start of conversion event. Choices are CTR=Zero (the default), CTR=PRD, CTRU=CMPA, CTRD=CMPA, CTRU=CMPB, and CTRD=CMPB.

## **Enable ADC start module B**

Select to allow ePWMB to trigger ADC start of conversion. This check box is cleared by default.

## **Number of event for SOCB to be generated**

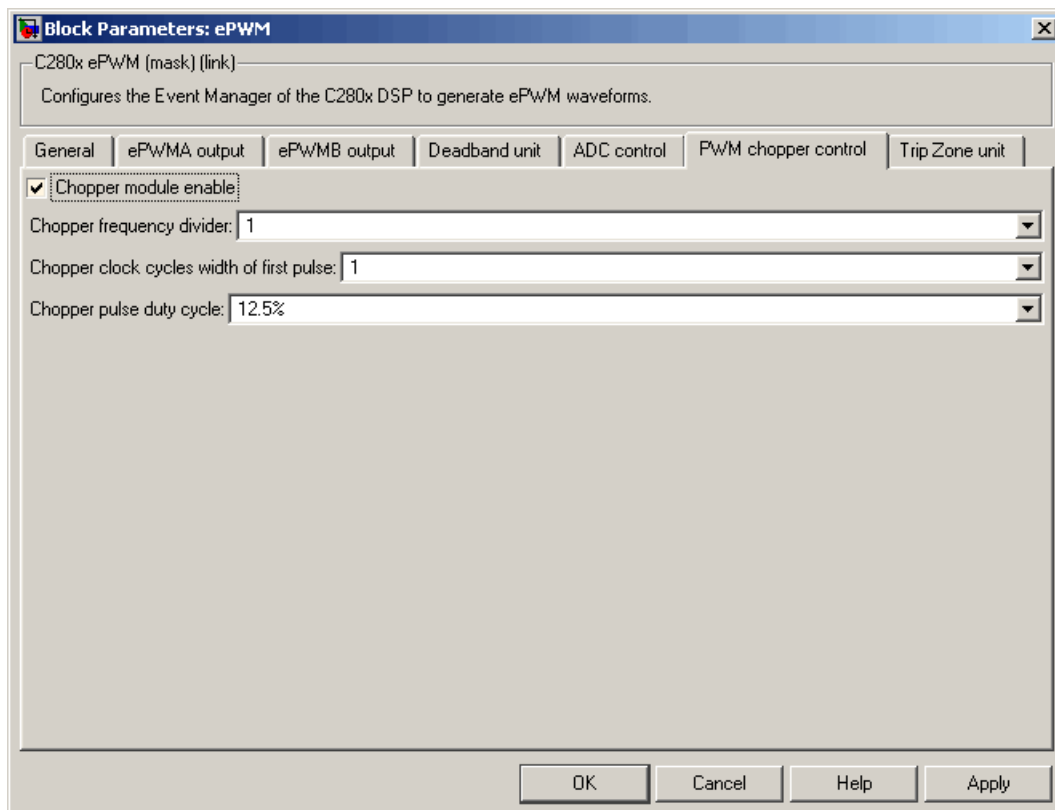
This field appears only when you select the **Enable ADC start module B** check box. Specify how often you want ADC start of conversion to be triggered. First event triggers ADC start of conversion with every event, Second event triggers ADC start of conversion with every second event, and Third event triggers ADC start of conversion with every third event.

## **Module B counter match event condition**

This field also appears only when you select the **Enable ADC start module B** check box. Specify the counter match condition that will trigger an ADC start of conversion event. Choices are CTR=Zero (the default), CTR=PRD, CTRU=CMPA, CTRD=CMPA, CTRU=CMPB, and CTRD=CMPB.

## **PWM Chopper Control Pane**

The **PWM chopper control** pane lets you specify parameters for the PWM-Chopper (PC) submodule. The PC submodule allows a high-frequency carrier signal to modulate the PWM waveform generated by the AQ and DB modules.



### **Chopper module enable**

Select to enable the chopper module. Use of the chopper module is optional, so this check box is cleared by default.

### **Chopper frequency divider**

**Chopper frequency divider** is a prescaler that is used to set the frequency of the chopper clock. The system clock speed is divided by this value to determine the chopper clock frequency. Choose an integer value from 1 to 8.

## **Chopper clock cycles width of first pulse**

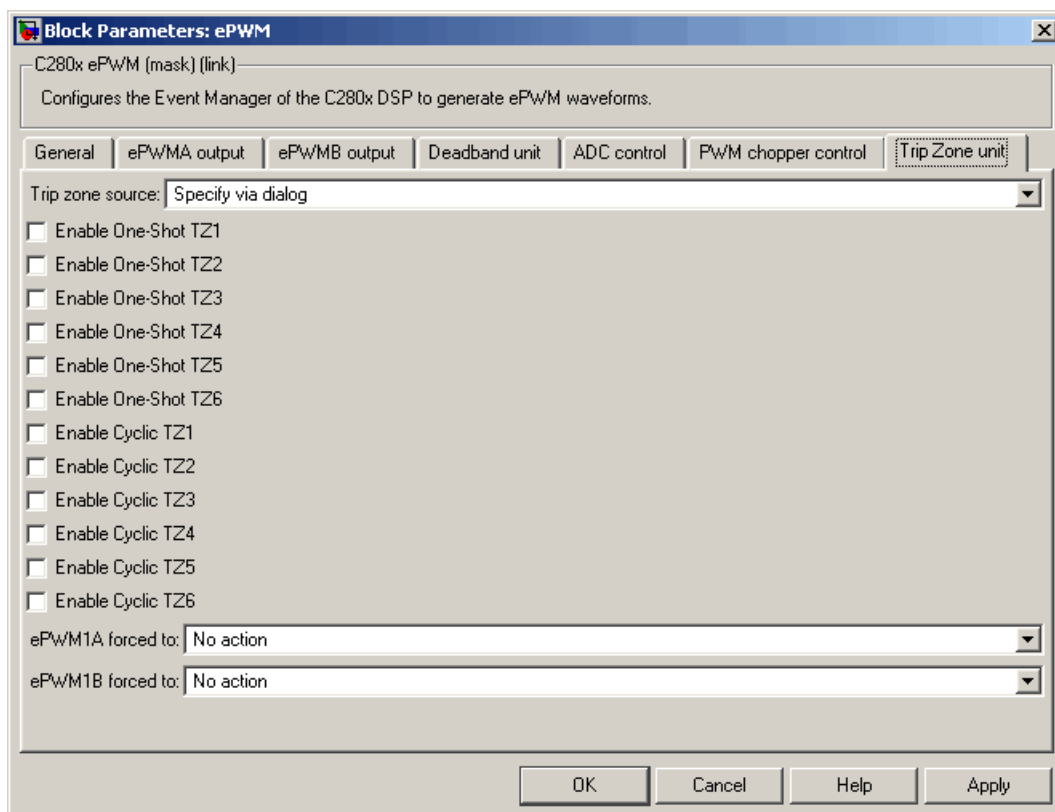
Choose an integer value from 1 to 16 to set the width of the first pulse. Use this feature to provide a high-energy first pulse to ensure hard and fast power switch turn on.

## **Chopper pulse duty cycle**

The duty cycles of the second and subsequent pulses are also programmable. Choices are 12.5%, 25%, 37.5%, 50%, 62.5%, 75%, and 87.5%.

## **Trip Zone Unit Pane**

The **Trip Zone unit** pane lets you specify parameters for the Trip-zone (TZ) submodule. Each ePWM module is connected to six TZ signals (TZ1 to TZ6) that are sourced from the GPIO MUX. These signals indicate external fault or trip conditions. Use the settings in this pane to program the EPWM outputs to respond when faults occur.



## Trip zone source

Specifies the source from which the control logic is to be obtained. Choose Specify via dialog (the default) to explicitly enable Trip-zone signals, or Input port to use information from the input port.

**Enable One-Shot TZ1**  
**Enable One-Shot TZ2**  
**Enable One-Shot TZ3**  
**Enable One-Shot TZ4**  
**Enable One-Shot TZ5**  
**Enable One-Shot TZ6**

Select any of these check boxes to enable the corresponding Trip-zone signal in One-Shot Mode. In this mode, when the trip event is active, the respective action on the EPWMxA/B output is carried out immediately and is latched. The condition remains latched and can only be cleared by the user under software control.

**Enable Cyclic TZ1**  
**Enable Cyclic TZ2**  
**Enable Cyclic TZ3**  
**Enable Cyclic TZ4**  
**Enable Cyclic TZ5**  
**Enable Cyclic TZ6**

Select any of these check boxes to enable the corresponding Trip-zone signal in Cycle-by-Cycle Mode. In this mode, when the trip event is active, the respective action on the EPWMxA/B output is carried out immediately and is latched. In Cycle-by-Cycle Mode, the condition is automatically cleared when the PWM Counter reaches zero. Therefore, in Cycle-by-Cycle Mode, the trip event is cleared or reset every PWM cycle.

**ePWMxA forced to**  
**ePWMxB forced to**

Upon a fault condition, the ePWMxA and/or ePWMxB output can be overridden and forced to one of the following: No action (the default), High, Low, or Hi-Z(High Impedance).

**See Also**

C280x ADC

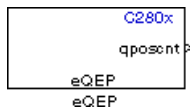
# C280x eQEP

---

**Purpose**                      Quadrature encoder pulse circuit

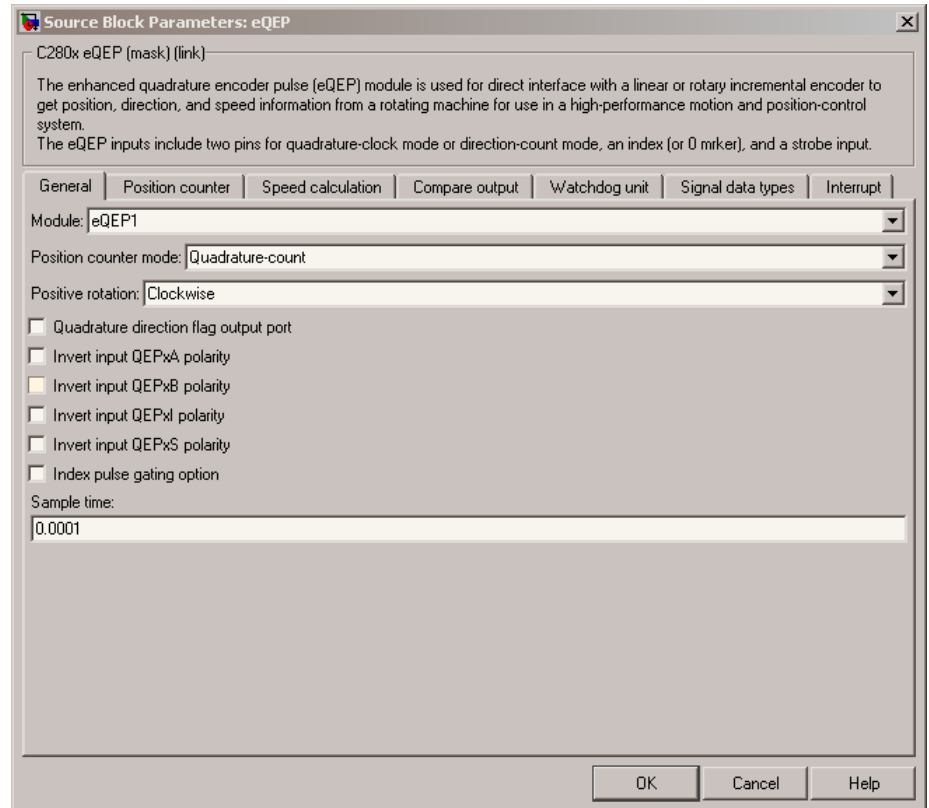
**Library**                      c280xdspchip1lib in Target for TI C2000

**Description**                The enhanced quadrature encoder pulse (eQEP) module is used for direct interface with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine for use in a high-performance motion and position-control system.



## Dialog Box

### General Pane



### Module

As many as two eQEP units are allowed on a single C280x-based board. Choose eQEP1 (the default) or eQEP2.

### Position counter mode

The input signals QEPA and QEPB are processed by the Quadrature Decoder Unit (QDU) to produce clock (QCLK) and direction (QDIR) signals. Choose the position counter mode appropriate to the way the input to the eQEP module is encoded.

Choices are Quadrature-count (the default), Direction-count, Up-count, and Down-count.

### **Positive rotation**

This field appears only when you choose Quadrature-count in **Position counter mode**. Choose the direction that represents positive rotation: Clockwise (the default) or Counterclockwise.

### **External clock rate**

This field appears only when you choose Direction-count, Up-count, or Down-count in **Position counter mode**. In these cases, you can program clock generation to the position counter to occur on both rising and falling edges of the QEPA input or on the rising edge only. The effect of choosing the former is increasing the measurement resolution by a factor of 2. Choices are 2x resolution: Count the rising/falling edge (the default) or 1x resolution: Count the rising edge only.

### **Quadrature phase error flag output port**

This check box appears only when you choose Quadrature-count in **Position counter mode**. Select this check box if you want to generate an interrupt when the QEPA and QEPB signals fall out of their normal state of being 90 degrees out of phase.

### **Quadrature direction flag output port**

This check box appears only when you choose Quadrature-count in **Position counter mode**. Select this check box if you want to generate an interrupt when the direction of counting is reversed by swapping the QEPA and QEPB input signals.

### **Invert input QEPxA polarity**

### **Invert input QEPxB polarity**

### **Invert input QEPxI polarity**

### **Invert input QEPxS polarity**

Select any of these check boxes to invert the polarity of the respective eQEP input signal.

### **Index pulse gating option**

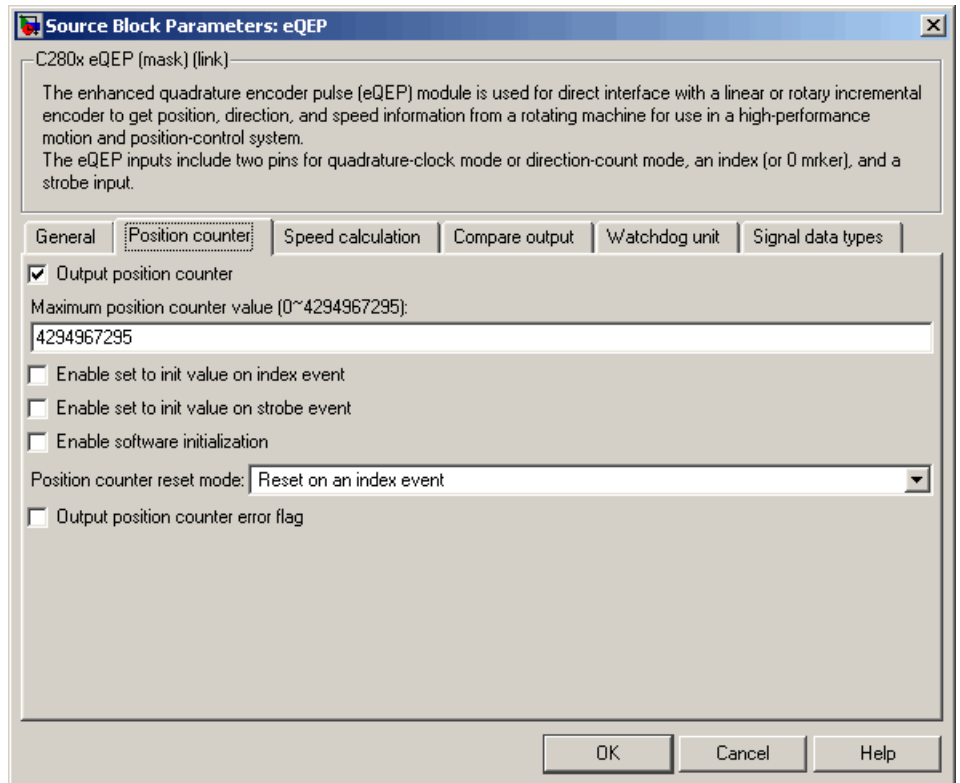
Select this check box to enable gating of the index pulse.



## Sample time

Enter the sample time in seconds.

## Position Counter Pane



## Output position counter

This check box is selected by default. Leave it selected to output the position counter signal PCSOUT from the position counter and control unit (PCCU).

## **Maximum position counter value**

Enter a maximum value for the position counter. Enter a value from 0 to 4294967295. The default is the maximum allowed value of 4294967295.

## **Enable set to init value on index event**

Select to set the position counter to its initialization value on an index event. This check box is cleared by default.

## **Set to init value on index event**

This field appears only when **Enable set to init value on index event** is selected. Choose to set the position counter to its initialization value on the **Rising** edge (the default) or the **Falling** edge of the index input.

## **Enable set to init value on strobe event**

Select to set the position counter to its initialization value on a strobe event. This check box is cleared by default.

## **Set to init value on strobe event**

This field appears only when **Enable set to init value on strobe event** is selected. Choose to set the position counter to its initialization value on the **Rising** edge (the default) or the **Falling** edge of the strobe input.

## **Enable software initialization**

Select to allow the position counter to be set to its initialization value via software. This check box is cleared by default.

## **Software initialization source**

This field appears only when **Enable software initialization** is selected. Choose **Set to init value at start up** (the default) or **Input port** to receive the control logic through the input port.

## **Initialization value**

This field appears only when **Enable set to init value on index event**, **Enable set to init value on strobe event**, or **Enable software initialization** check box is selected. Enter the initialization value for the position counter. Enter a value from 0 to 4294967295. The default is 2147483648.

**Position counter reset mode**

Choose a position counter reset mode, depending on the nature of the system the eQEP module is working with: Reset on an index event (the default), Reset on the maximum position, Reset on the first index event, or Reset on a time unit event.

**Output position counter error flag**

This check box appears only when **Position counter reset mode** is set to Reset on an index event. Select this check box to output the position counter error flag on error.

**Output latch position counter on index event**

This check box appears only when **Position counter reset mode** is set to Reset on the maximum position or Reset on the first index event. The eQEP index input can be configured to latch the position counter (QPOSCNT) into QPOSILAT on occurrence of a definite event on this pin. Select this check box to latch the position counter on each index event.

**Index event latch of position counter**

This field appears only when the **Output latch position counter on index event** check box is selected. Choose one of the following events to configure the eQEP position counter to latch on that event: Rising edge, Falling edge, or Software index marker via input port.

**Output latch position counter on strobe event**

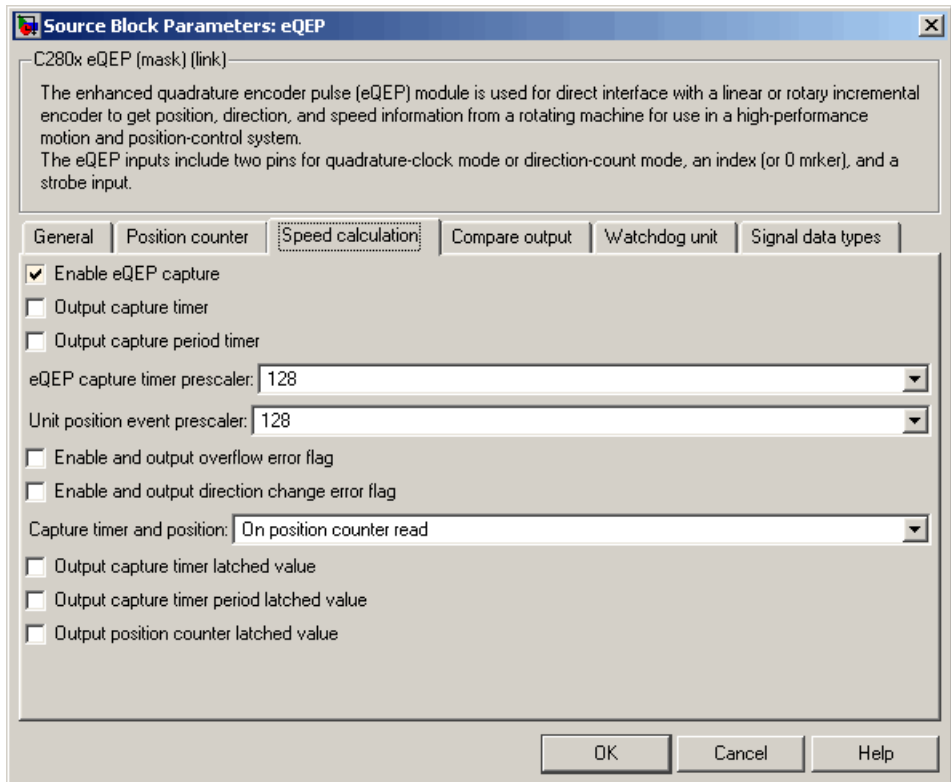
This check box appears only when **Position counter reset mode** is set to Reset on the maximum position or Reset on the first index event. The eQEP strobe input can be configured to latch the position counter (QPOSCNT) into QPOSSLAT on occurrence of a definite event on this pin. Select this check box to latch the position counter on each strobe event.

**Strobe event of latched position counter**

This field appears only when the **Output latch position counter on strobe event** check box is selected. Choose Rising edge to latch on the rising edge of the strobe event input, or Depending

on direction to latch on the rising edge in the forward direction and the falling edge in the reverse direction.

## Speed Calculation Pane



### Enable QEP capture

The eQEP peripheral includes an integrated edge capture unit to measure the elapsed time between the unit position events. Check this check box to enable the edge capture unit. This check box is cleared by default.

**Output capture timer**

Select this check box to output the capture timer into the capture period register. This check box is cleared by default.

**Output capture period timer**

Select this check box to output the capture period into the capture period register. This check box is cleared by default.

**eQEP capture timer prescaler**

The eQEP capture timer runs from prescaled SYSCLKOUT. The capture timer period is the value of SYSCLKOUT divided by the value you choose in this field. Choices are 1, 2, 4, 8, 16, 32, 64, and 128 (the default).

**Unit position event prescaler**

The timing of the unit position event is determined by prescaling the quadrature-clock (QCLK). QCLK is divided by the value you choose in this popup. Choices are 4, 8, 16, 32, 64, 128, 256, 512, 1024, and 2048 (the default).

**Enable and output overflow error flag**

Select this check box to enable and output the eQEP overflow error flag in the event of capture timer overflow between unit position events.

**Enable and output direction change error flag**

Select this check box to enable and output the direction change error flag.

**Capture timer and position**

Choose the event that triggers the latching of the capture timer and capture period register: On position counter read (the default) or On unit time-out event.

**Unit timer period**

This field appears only when you choose On unit time-out event in **Capture timer and position**. Enter a value for the unit timer period from 0 to 4294967295. The default is 100000000.

## Output capture timer latched value

Select this check box to output the capture timer latched value from the QCTMRLAT register.

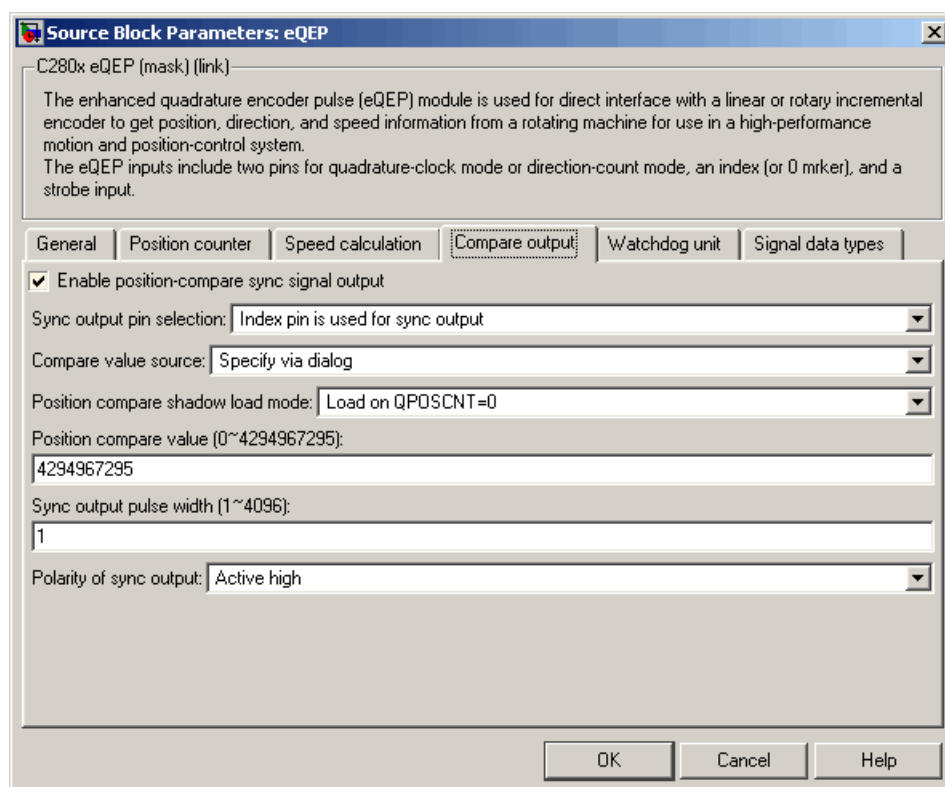
## Output capture timer period latched value

Select this check box to output the capture timer period latched value from the QCPRDLAT register.

## Output position counter latched value

Select this check box to output the position counter latched value from the QPOSLAT register.

## Compare Output Pane



**Enable position-compare sync signal output**

The eQEP peripheral includes a position-compare unit that is used to generate the position-compare sync signal on compare match between the position counter register (QPOSCNT) and the position-compare register (QPOSCMP). Select this check box to enable the position-compare sync signal output. This check box is cleared by default.

**Sync output pin selection**

Choose which pin is used for the sync signal output. Choices are Index pin is used for sync output (the default) and Strobe pin is used for sync output.

**Compare value source**

Choose the source of the value to use in the position comparison. Choose Specify via dialog (the default) to specify a fixed value or Input port to read the value from the input port.

**Position compare shadow load mode**

This field lets you enable or disable shadow mode for use in generating the position-compare sync signal (shadow mode is enabled by default). When shadow mode is enabled, you can also choose an event to trigger the loading of the shadow register value into the active register.

Choose Disable shadow mode to disable shadow mode. Choose Load on QPOSCNT=0 (the default) to load on the position-counter zero event. Choose Load on QPOSCNT=QPOSCMP to load on compare match.

**Position compare value**

This field appears only when you choose Specify via dialog in **Compare value source**. Enter a value from 0 to 4294967295. The default is 4294967295. This value is loaded into the position-compare register (QPOSCMP).

## **Sync output pulse width**

The pulse stretcher logic in the position-compare unit generates a programmable position-compare sync pulse output on the position-compare match.

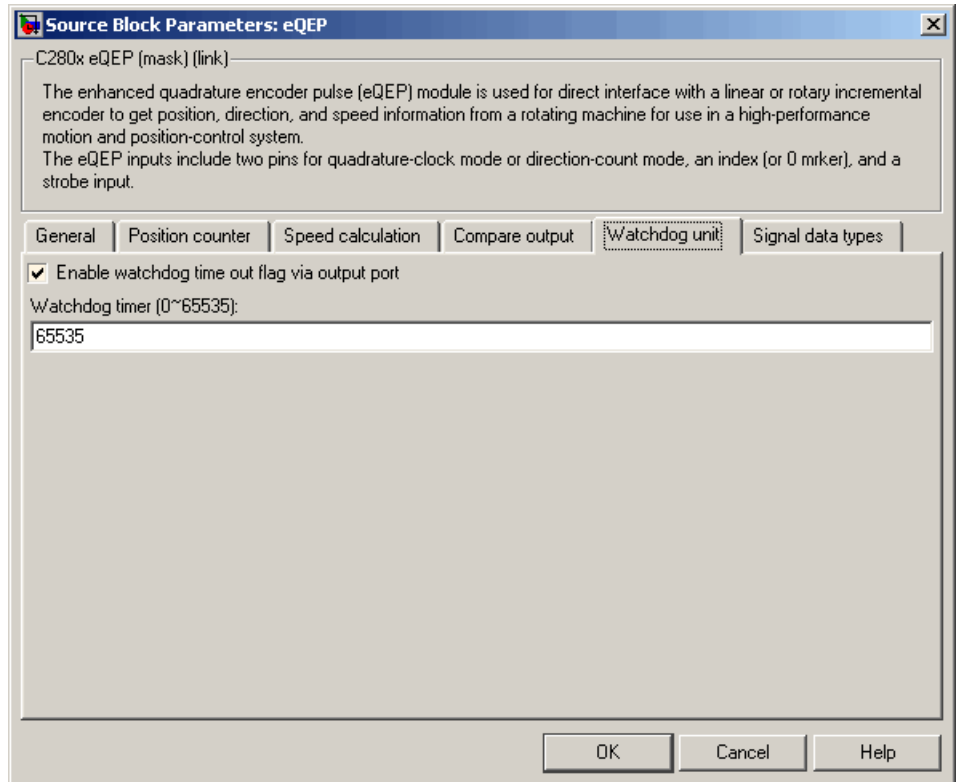
Enter a value from 1 to 4096 to determine the pulse width of the position-compare sync output signal. The default is 1.

## **Polarity of sync output**

Choose a value to determine the polarity of the sync output signal: Active high (the default) or Active low.



## Watchdog Unit Pane



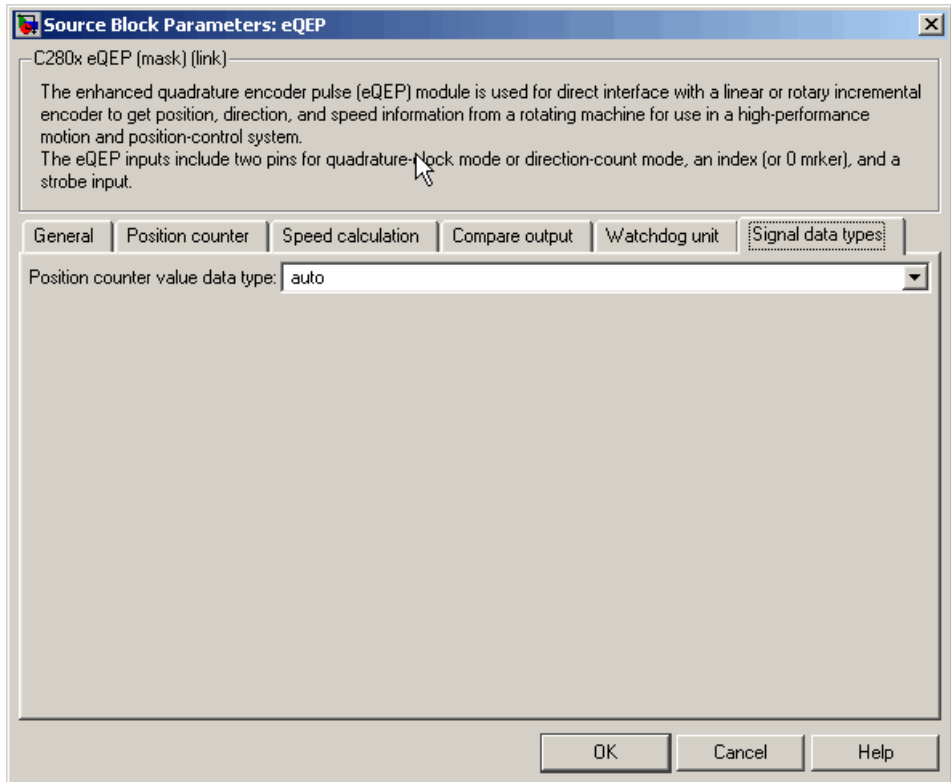
### Enable watchdog time out flag via output port

The eQEP peripheral contains a watchdog timer that monitors the quadrature-clock to indicate proper operation of the motion-control system. Select this check box to enable the watchdog time out flag.

### Watchdog timer

Enter the time-out value for the watchdog timer. Enter a value from 0 to 65535 (the default).

## Signal Data Types Pane



The image above shows the default condition of the **Signal data types** pane. Choosing any of a number of options in other panes of the C280x eQEP dialog box causes a corresponding popup to appear in the **Signal data types** pane.

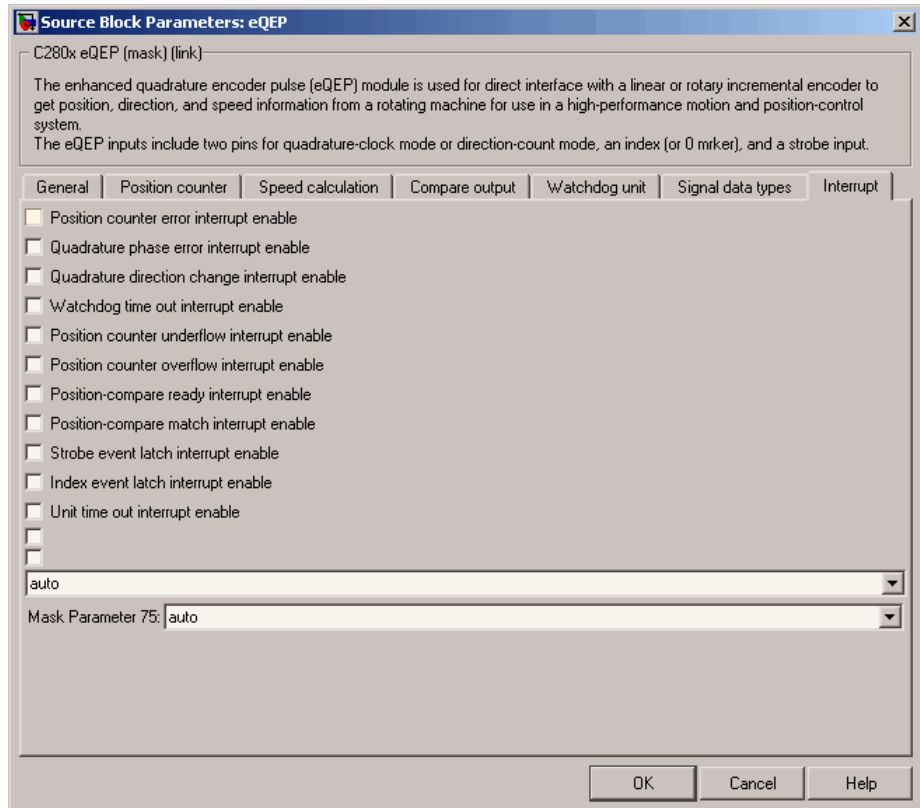
The following table summarizes the options for which you can set the data type in the **Signal data types** pane:

Pane	Option
General	Quadrature phase error flag output port
	Quadrature direction flag output port
Position counter	Output position counter (selected by default)
	Output position counter error flag
	Output latch position counter on index event
	Output latch position counter on strobe event
Speed calculation	Output capture timer
	Output capture period timer
	Enable and output overflow error flag
	Enable and output direction change error flag
	Output capture timer latched value
	Output capture timer period latched value
Watchdog unit	Output position counter latched value
	Enable watchdog time out flag via output port

The fields that appear on the **Signal data types** pane are named similarly to these options. For example, **Position counter value data type** on the **Signal data types** pane corresponds to the **Output position counter** option on the **Position counter** pane.

For all data type fields, valid data types are auto, double, single, int8, uint8, int16, uint16, int32, uint32, and boolean.

## Interrupt Pane



The image above shows the default condition of the **Interrupt** pane. Interrupts corresponding to specific events are enabled or disabled based on the settings in this pane.

### **Position counter error interrupt enable**

Check this box to enable position counter error interrupts. This checkbox is cleared by default.

**Quadrature phase error interrupt enable**

Check this box to enable quadrature phase error interrupts. This checkbox is cleared by default.

**Quadrature direction change interrupt enable**

Check this box to enable quadrature direction change interrupts for changes in the counting direction. This checkbox is cleared by default.

**Watchdog timeout interrupt enable**

The eQEP Peripheral contains a watchdog timer that monitors the quadrature clock. Check this box to enable watchdog timeout interrupts. This checkbox is cleared by default.

**Position counter underflow interrupt enable**

Check this box to enable position counter underflow interrupts. This checkbox is cleared by default.

**Position counter overflow interrupt enable**

Check this box to enable position counter overflow interrupts. This checkbox is cleared by default.

**Position-compare ready interrupt enable**

Check this box to enable position-compare ready interrupts. This checkbox is cleared by default.

**Position-compare match interrupt enable**

Check this box to enable position-compare match interrupts. This checkbox is cleared by default.

**Strobe event latch interrupt enable**

Check this box to enable strobe event latch interrupts. This checkbox is cleared by default.

**Index event latch interrupt enable**

Check this box to enable index event latch interrupts. This checkbox is cleared by default.

**Unit timeout interrupt enable**

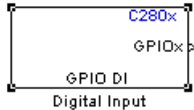
Check this box to enable unit timeout interrupts. This checkbox is cleared by default.

# C280x GPIO Digital Input

**Purpose** Configure general purpose input pins

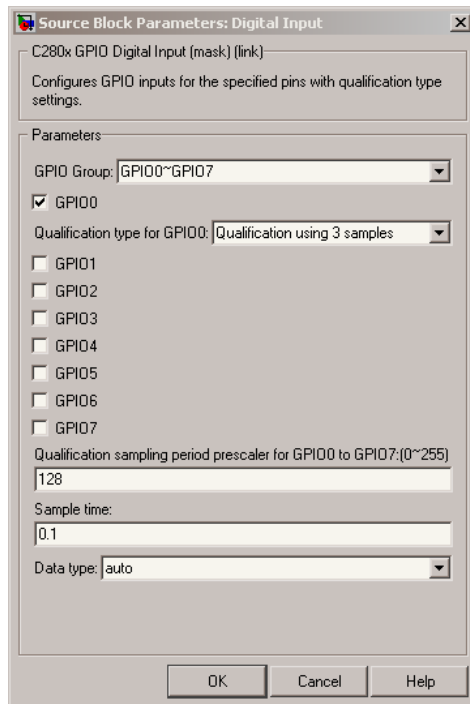
**Library** c280xdspchip1lib in Target for TI C2000

## Description



This block configures the general purpose I/O (GPIO) MUX registers that control the operation of GPIO shared pins for digital input. Each I/O port has one MUX register that selects peripheral operation or digital I/O operation. Reset configures all pins for I/O operation. There are 35 pins total. Any pin you select for input through the dialog box cannot be used simultaneously as an output. Peripherals connected to that pin must be disabled. For each pin selected for input operation, you can specify the type of signal qualification required.

## Dialog Box

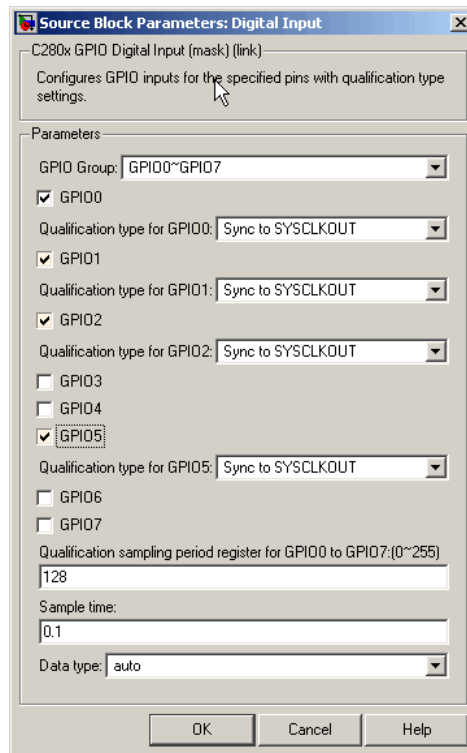


## GPIO Group

Each group contains eight ports, with the exception of the last one which contains two. Select the input group to use: GPIO0~GPIO7, GPIO8~GPIO15, GPIO16~GPIO23, GPIO24~GPIO31, or GPIO32~GPIO34.

## GPIO pins for input

For each **GPIO Group**, select the specific pins to enable for digital input. For example, for the GPIO0~GPIO7 group, you might wish to select the **GPIO0**, **GPIO1**, **GPIO2**, and **GPIO5** pins, as shown in the following figure:



## C280x GPIO Digital Input

Pins that you do not select are available for peripheral functionality or output. The following table shows the peripherals available for each pin.

<b>GPIO Name</b>	<b>Available Peripherals</b>
GPIO0	EPWM1A
GPIO1	EPWM1B; SPISIMOD
GPIO2	EPWM2A
GPIO3	EPWM2B; SPISOMID
GPIO4	EPWM3A
GPIO5	EPWM3B; SPICLKD; ECAP1
GPIO6	EPWM4A; EPWMSYNCI; EPWMSYNCO
GPIO7	EPWM4B; SPISTED; ECAP2
GPIO8	EPWM5A; CANTXB; ADCSOCAO
GPIO9	EPWM5B; SCITXB; ECAP3
GPIO10	EPWM6A; CANRXB; ADCSOCBO
GPIO11	EPWM6B; SCIRXB; ECAP4
GPIO12	TZ1; CANTXB; SPISIMOB
GPIO13	TZ2; CANRXB; SPISOMIB
GPIO14	TZ3; SCITXB; SPICLKB
GPIO15	TZ4; SCIRXB; SPISTEB
GPIO16	SPISIMOA; CANTXB; TZ5
GPIO17	SPISOMIA; CANRXB; TZ6
GPIO18	SPICLKA; SCITXB



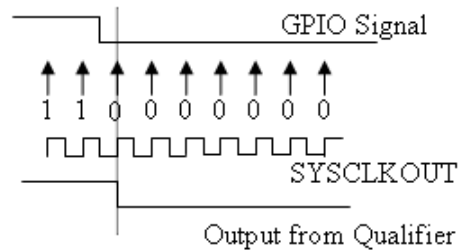
GPIO Name	Available Peripherals
GPIO19	SPISTEAA; SCIRXB
GPIO20	EQEP1A; SPISIMOC; CANTXB
GPIO21	EQEP1B; SPISOMIC; CANRXB
GPIO22	EQEP1S; SPICLKC; SCITXB
GPIO23	EQEP1I; SPISTEC; SCIRXB
GPIO24	ECAP1; EQEP2A; SPISIMOB
GPIO25	ECAP2; EQEP2B; SPISOMIB
GPIO26	ECAP3; EQEP2I; SPICLKB
GPIO27	ECAP4; EQEP2S; SPISTEB
GPIO28	SCIRXDA; TZ5
GPIO29	SCITXDA; TZ6
GPIO30	CANRXA
GPIO31	CANTXA
GPIO32	SDAA; EPWMSYNCI; ADCSOAO
GPIO33	SCLA; EPQMSYNCO; ADCSOAO
GPIO34	Reserved

### Qualification type for GPIO[pin#]

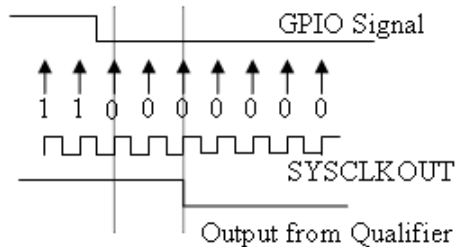
Each pin selected for input offers three signal qualification types:

- Sync to SYSCLKOUT — This setting is the default for all pins at reset. Using this qualification type, the input signal is synchronized to the system clock SYSCLKOUT. The following figure shows the input signal measured on each tick of the system clock, and the resulting output from the qualifier.

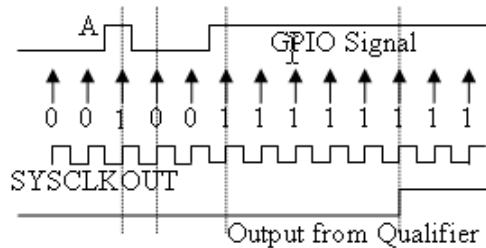
# C280x GPIO Digital Input



- Qualification using 3 samples — This setting requires three consecutive cycles of the same value for the output value to change. The following figure shows that, in the third cycle, the GPIO value changes to 0, but the qualifier output is still 1 because it waits for three consecutive cycles of the same GPIO value. The next three cycles all have a value of 0, and the output from the qualifier changes to 0 immediately after the third consecutive value is received.



- Qualification using 6 samples — This setting requires six consecutive cycles of the same GPIO input value for the output from the qualifier to change. In the following figure, the glitch **A** has no effect on the output signal. When the glitch occurs, the counting begins, but the next measurement is low again, so the count is ignored. The output signal does not change until six consecutive samples of the high signal are measured.



## Qualification sampling period prescaler

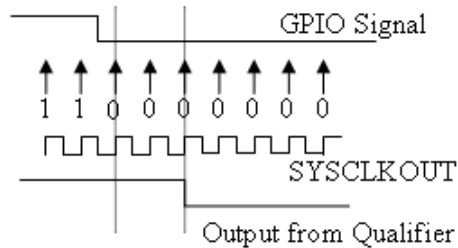
Visible only when an appropriate setting for **Qualification type for GPIO [pin#]** is selected. The qualification sampling period prescaler, with possible values of 0 to 255, calculates the frequency of the qualification samples or the number of system clock ticks per sample. The formula for calculating the qualification sampling frequency is:

$$\frac{SYSCLKOUT}{2 * Prescaler}$$

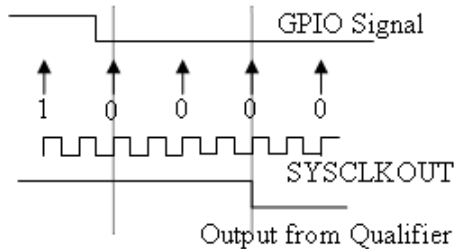
with the exception of zero. When **Qualification sampling period prescaler=0**, a sample is taken every SYSCLKOUT clock tick. For example, a prescale setting of 0 means that a sample is taken on each SYSCLKOUT tick.

The following figure shows the SYSCLKOUT ticks, a sample taken every clock tick, and the **Qualification type** set to Qualification using 3 samples. In this case, the **Qualification sampling period prescaler=0**:

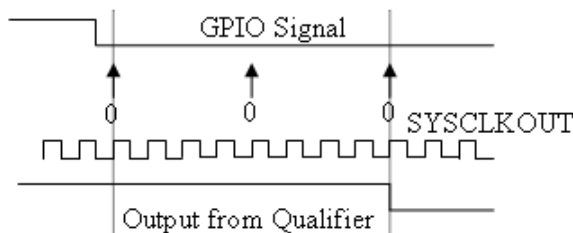
# C280x GPIO Digital Input



In the next figure **Qualification sampling period prescaler=1**. A sample is taken every two clock ticks, and the **Qualification type** is set to Qualification using 3 samples. The output signal changes much later than if **Qualification sampling period prescaler=0**.



In the following figure, **Qualification sampling period prescaler=2**. Thus, a sample is taken every four clock ticks, and the **Qualification type** is set to Qualification using 3 samples.



**Sample time**

Specifies the time interval between output samples. To inherit sample time from the upstream block, set this parameter to -1. For more information, refer to the section on “Specifying Sample Time” in the Simulink documentation.

**Data type**

Specifies the data type of the input. The input is read as 16-bit integer, and then cast to the selected data type. Valid data types are auto, double, single, int8, uint8, int16, uint16, int32, uint32 or boolean.

**See Also**

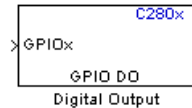
C280x GPIO Digital Output

# C280x GPIO Digital Output

**Purpose** Configure general purpose output pins

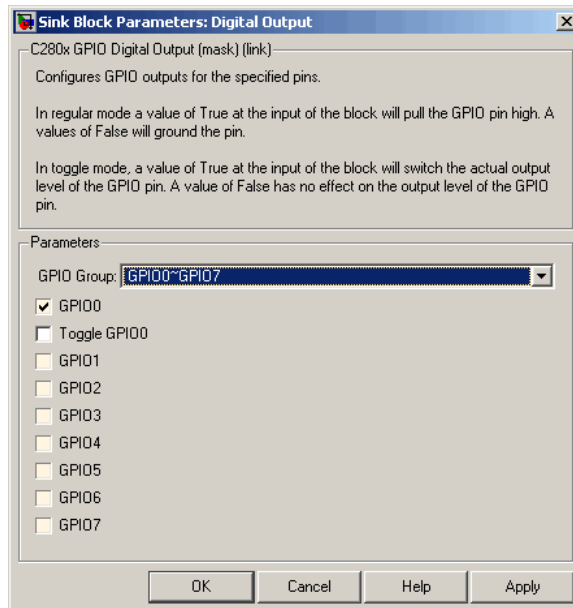
**Library** c280xdspchip1lib in Target for TI C2000

## Description



This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital output. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation. There are 35 pins total. Any pin selected for output through the dialog box cannot be used simultaneously as an input. For each specified output pin you select, you can elect to toggle the GPIO pin signal.

## Dialog Box



## GPIO Group

Each group contains eight ports, with the exception of the last group which contains two. There are 35 ports, or pins total.

Select the output group to use: GPIO0~GPIO7, GPIO8~GPIO15, GPIO16~GPIO23, GPIO24~GPIO31, or GPIO32~GPIO34.

## GPIO pins for output

Select the pins for output from each group. Pins that you do not select for output can be used for input or peripheral functionality. Refer to the C280x GPIO Digital Input block for a table of all available peripherals for each pin.

A value of `True` at the input of the block drives the selected GPIO pin high. A value of `False` at the input of the block grounds the selected GPIO pin.

## Toggle GPIO[bit#]

For each pin selected for output, you can elect to toggle the signal of that pin. In **Toggle** mode, a value of `True` at the input of the block switches the GPIO pin output level. Thus, if the GPIO pin was driven high, in **Toggle** mode, with the value of `True` at the input, the pin output level is driven low. If the GPIO pin was driven low, in **Toggle** mode, with the value of `True` at the input of the block, the same pin output level is driven high. If the input of the block is `False`, there is no effect on the GPIO pin output level.

---

**Note** The outputs of this block can be vectorized.

---

## See Also

C280x GPIO Digital Input

# C280x Hardware Interrupt

---

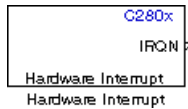
## Purpose

Interrupt Service Routine to handle hardware interrupt on C280x processor

## Library

c280xdspchip1lib in Target for TI C2000

## Description



For many systems, an execution scheduling model based on a timer interrupt is not sufficient to ensure a real-time response to external events. The C280x Hardware Interrupt block addresses this problem by allowing for the asynchronous processing of interrupts triggered by events managed by other blocks in the C280x DSP Chip Support Library.

The C280x blocks that can generate an interrupt for asynchronous processing are:

- C280x ADC
- C280x eCAN Receive
- C280x eCAN Transmit
- C280x eCAP
- C280x eQEP
- C280x SCI Receive
- C280x SCI Transmit
- C280x Software Interrupt Trigger
- C280x SPI Receive
- C280x SPI Transmit

Only one Hardware Interrupt block can be used in a model. To handle multiple interrupts, place a Demux block at the output of the Hardware Interrupt block to direct function calls to the appropriate function-call subsystems.

For details about this block, refer to C280x Hardware Interrupt block in your Link for Code Composer Studio Development Tools documentation.



## Vectorized Output

This block outputs a function call. The size of the function call line equals the number of interrupts the block is set to handle. The block dialog box displays four parameters for each interrupt. These parameters comprise a set of four vectors of equal length. Each interrupt is represented by one element from each parameter (four elements total), one from the same position in each of these vectors.

The following parameters describe the elements in the interrupt vector:

- CPU interrupt numbers
- PIE interrupt numbers
- Task priorities
- Preemption flags

Thus, one interrupt is described by a CPU interrupt number, a PIE interrupt number, a task priority, and a preemption flag.

The CPU and PIE interrupt numbers together uniquely specify a single interrupt for a single peripheral or peripheral module. The following table maps CPU and PIE interrupt numbers to these peripheral interrupts. The row numbers are CPU values and the column numbers are the PIE values.

---

**Note** The TINT0 (TIMER 0) interrupt is always reserved, and will generate errors if used.

---

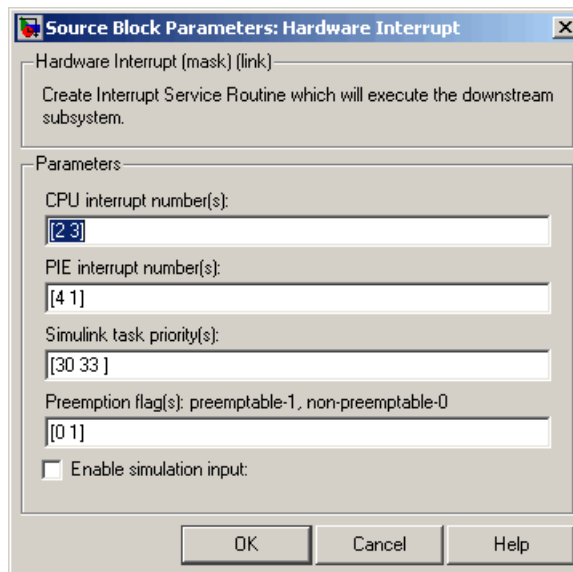
C280x Peripheral Interrupt Vector Values

	1	2	3	4	5	6	7	8
1	SEQ1INT (ADC)	SEQ2INT (ADC)	Reserved	XINT1	XINT2	ADCINT (ADC)	TINT0 (TIMER 0)	WAKEINT (LPM/WVD)
2	EPWM1_TZIN (ePWM1)	EPWM2_TZIN (ePWM2)	EPWM3_TZIN (ePWM3)	EPWM4_TZIN (ePWM4)	EPWM5_TZIN (ePWM5)	EPWM6_TZIN (ePWM6)	Reserved	Reserved
3	EPWM1_INT (ePWM1)	EPWM2_INT (ePWM2)	EPWM3_INT (ePWM3)	EPWM4_INT (ePWM4)	EPWM5_INT (ePWM5)	EPWM6_INT (ePWM6)	Reserved	Reserved
4	ECAP1_INT (eCAP1)	ECAP2_INT (eCAP2)	ECAP3_INT (eCAP3)	ECAP4_INT (eCAP4)	Reserved	Reserved	Reserved	Reserved
5	EQEP1_INT (eQEP1)	EQEP2_INT (eQEP2)	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
6	SPIRXINTA (SPI-A)	SPIRXINTA (SPI-A)	SPIRXINTB (SPI-B)	SPIRXINTB (SPI-B)	SPIRXINTC (SPI-C)	SPIRXINTC (SPI-C)	SPIRXINTD (SPI-D)	SPIRXINTD (SPI-D)
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
8	I2CINT2A (I2C-A)	I2CINT1A (I2C-A)	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
9	SCIRXINTA (SCI-A)	SCITXINTA (SCI-A)	SCIRXINTB (SCI-B)	SCITXINTB (SCI-B)	ECAN0INTA (CAN-A)	ECAN1INTA (CAN-A)	ECAN0INTB (CAN-B)	ECAN1INTB (CAN-B)
10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

The task priority indicates the relative importance tasks associated with the asynchronous interrupts. If an interrupt triggers a higher-priority task while a lower-priority task is running, the execution of the lower-priority task is suspended while the higher-priority task is executed. The lowest value represents the highest priority. The default priority value of the base rate task is 40, so the priority value for each asynchronously triggered task must be less than 40 for these tasks to cause the suspension of the base rate task.

The preemption flag determines whether a given interrupt is preemptable. Preemption overrides prioritization, such that a preemptable task of higher priority can be preempted by a nonpreemptable task of lower priority.

## Dialog Box



### CPU interrupt number(s)

Enter a vector of CPU interrupt numbers for the interrupts you want to process asynchronously.

# C280x Hardware Interrupt

---

See the table of C280x Peripheral Interrupt Vector Values on page 7-82 for a mapping of CPU interrupt number to interrupt names.

## **PIE interrupt number(s)**

Enter a vector of PIE interrupt numbers for the interrupts you want to process asynchronously.

See the table of C280x Peripheral Interrupt Vector Values on page 7-82 for a mapping of CPU interrupt number to interrupt names.

## **Simulink task priority(s)**

Enter a vector of task priorities for the interrupts you want to process asynchronously.

See the discussion of this block's "Vectorized Output" on page 7-81 for an explanation of task priorities.

## **Preemption flag(s)**

Enter a vector of preemption flags for the interrupts you want to process asynchronously.

See the discussion of this block's "Vectorized Output" on page 7-81 for an explanation of preemption flags.

## **Enable simulation input**

Select this check box if you want to be able to test asynchronous interrupt processing in the context of your Simulink model.

---

**Note** Using this check box is the only way you can test asynchronous interrupt processing behavior in Simulink.

---

## **References**

For detailed information about interrupt processing, refer to *TMS320x280x DSP System Control and Interrupts Reference Guide*, SPRU712B, available at the Texas Instruments Web site.

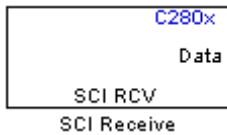
## **See Also**

C280x SW Int Trigger, Idle Task

**Purpose** Receive data on target via serial communications interface (SCI) from host

**Library** c280xdspchip1lib in Target for TI C2000

**Description** The C280x SCI Receive block supports asynchronous serial digital communications between the target and other asynchronous peripherals in nonreturn-to-zero (NRZ) format. This block configures the C280x DSP target to receive scalar or vector data from the COM port via the C280x target's COM port.



---

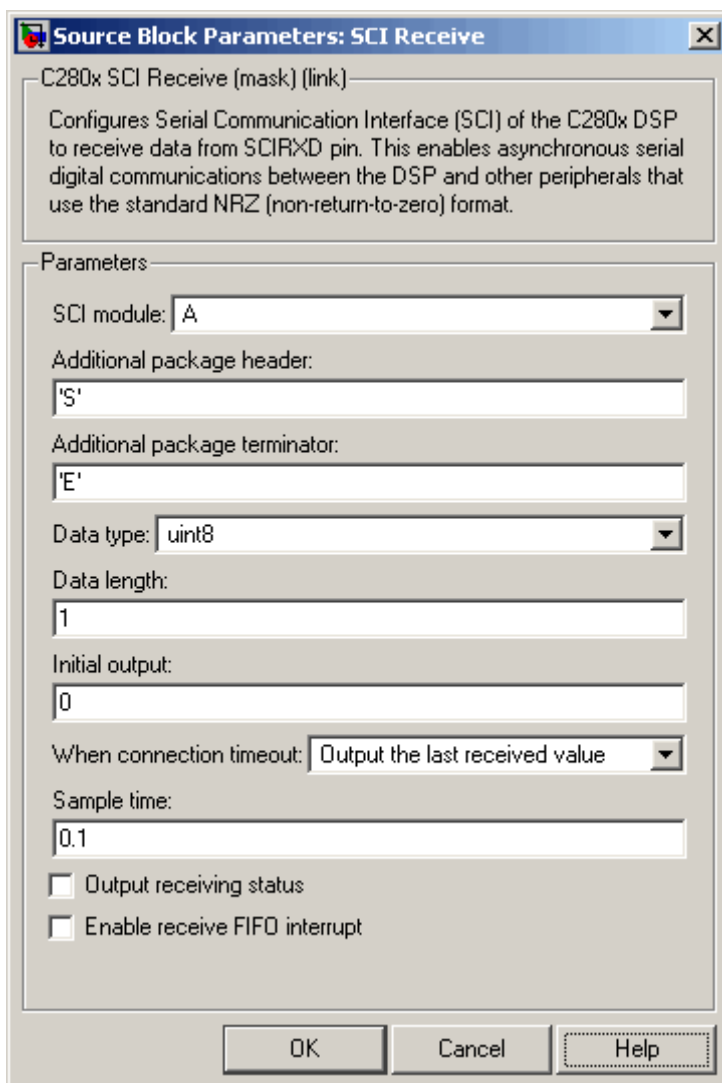
**Note** For any given model, you can have only one C280x SCI Receive block per module. There are two modules, A and B, which can be configured through the F2808 eZdsp target preferences block.

Many SCI-specific settings are in the **DSPBoard** section of the F2808 eZdsp target preferences block. You should verify that these settings are correct for your application.

---

# C280x SCI Receive

## Dialog Box



### SCI module

SCI module to be used for communications.

## Additional package header

This field specifies the data located at the front of the received data package, which is not part of the data being received, and generally indicates start of data. The additional package header must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not received nor are they included in the total byte count.

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the host SCI Transmit block.

---

## Additional package terminator

This field specifies the data located at the end of the received data package, which is not part of the data being received, and generally indicates end of data. The additional package terminator must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not received nor are they included in the total byte count.

## Data type

Data type of the output data. Available options are single, int8, uint8, int16, uint16, int32, or uint32.

## Data length

How many of **Data type** the block will receive (not bytes). Anything more than 1 is a vector. The data length is inherited from the input (the data length originally input to the host-side SCI Transmit block).

## Initial output

Default value from the c280x SCI Receive block. This value is used, for example, if a connection time-out occurs and the **When**

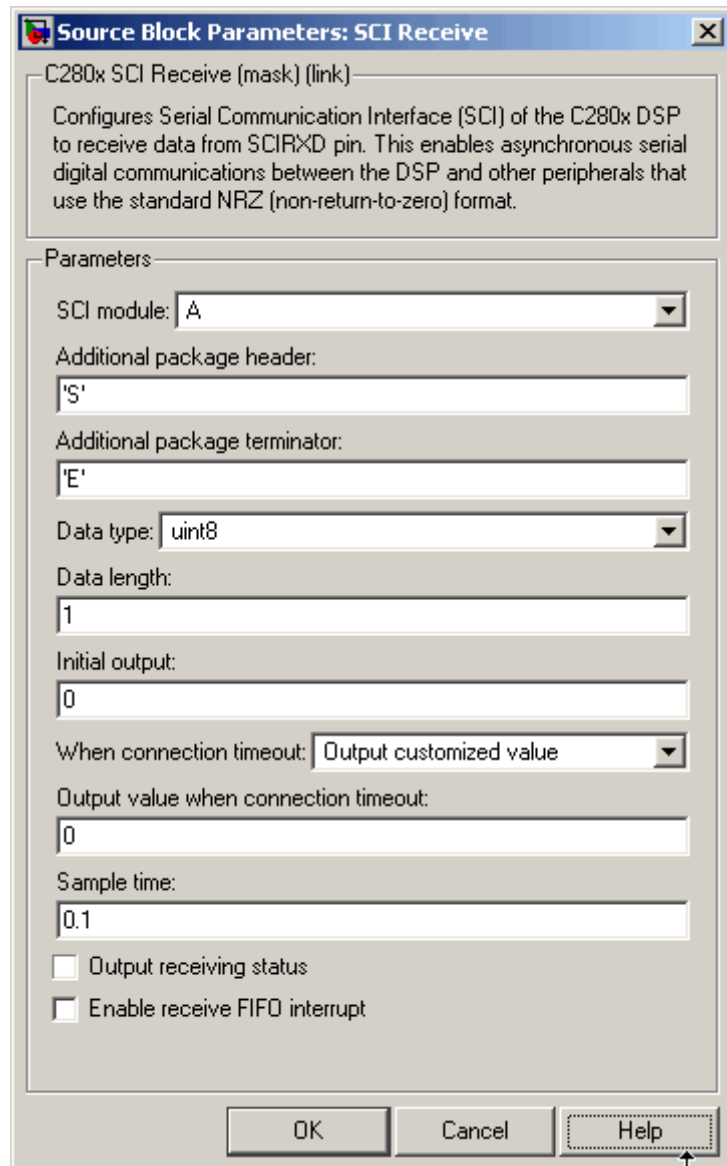
**connection timeout** field is set to “Output the last received value”, but nothing yet has been received.

### **When connection timeout**

Specifies what to output if a connection time-out occurs. If “Output the last received value” is selected, the last received value is what is output, unless none has been received yet, in which case the **Initial output** is considered the last received value.

If “Output customized value” is selected, a field for specifying a custom value is added to the dialog box (as shown in the following figure).





# C280x SCI Receive

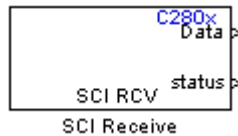
---

## Sample time

Sample time,  $T_s$ , for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

## Output receiving status

When this field is checked, the c280x SCI Receive block adds another output port for the transaction status, and appears as shown in the following figure.



The error status may be one of the following values:

- 0: No errors
- 1: A time-out occurred while the block was waiting to receive data
- 2: There is an error in the received data (checksum error)
- 3: SCI parity error flag — Occurs when a character is received with a mismatch
- 4: SCI framing error flag — Occurs when an expected stop bit is not found

## Enable receive FIFO interrupt

If this option is selected, an interrupt is posted when FIFO is full, allowing the subsystem to take some sort of action (for example, read data as soon as it is received). If this option is cleared, the block stays in polling mode. If the block is in polling mode and not blocking, it checks the FIFO to see if there is data to read. If data is present, it reads and outputs. If no data is present, it continues. If the block is in polling mode and blocking, it waits until data is available to read (after data length is reached).

### **Receive FIFO interrupt level**

This parameter is enabled when the **Enable receive FIFO interrupt** option is selected. Select an interrupt level from 0 to 16. The default level is 0.

### **References**

Detailed information on the SCI module is in *TMS320x281x, 280x DSP Serial Communication Interface (SCI) Reference Guide*, Literature Number SPRU051B, available at the Texas Instruments Web site.

### **See Also**

C280x SCI Transmit, C280x Hardware Interrupt

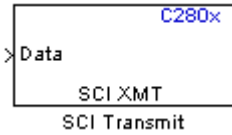
# C280x SCI Transmit

---

**Purpose** Transmit data from target via serial communications interface (SCI) to host

**Library** c280xdspchip1lib in Target for TI C2000

**Description** The C280x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C280x target's COM ports in nonreturn-to-zero (NRZ) format. You can specify how many of the six target COM ports to use. The sampling rate and data type are inherited from the input port. The data type of the input port must be one of the following: `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`. If no data type is specified, the default data type is `uint8`.



---

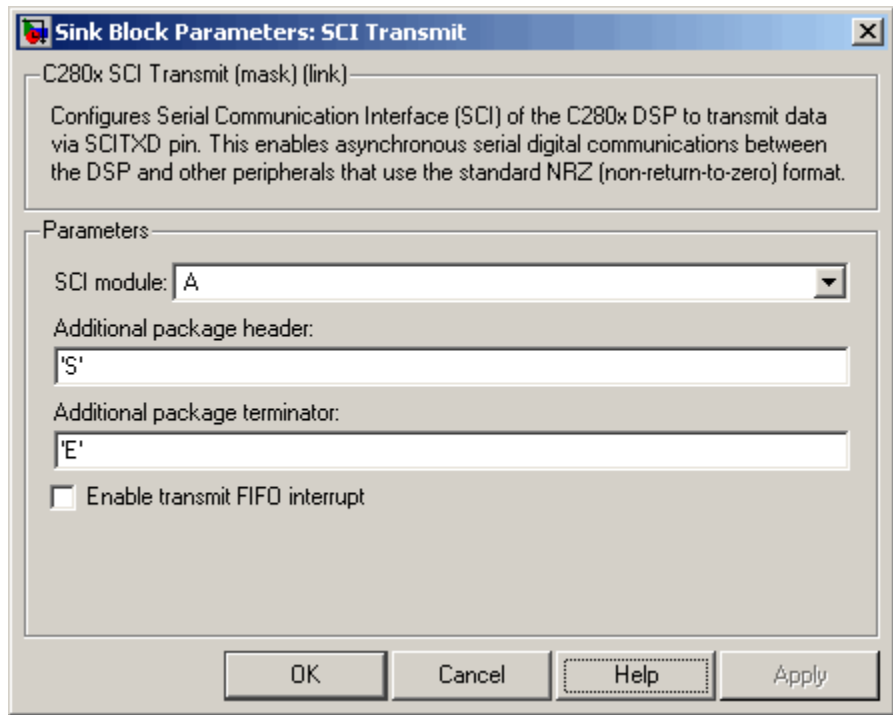
**Note** For any given model, you can have only one C280x SCI Transmit block per module. There are two modules, A and B, which can be configured through the F2808 eZdsp target preferences block.

Many SCI-specific settings are in the **DSPBoard** section of the Target Preferences block. You should verify that these settings are correct for your application.

Fixed-point inputs are not supported for this block.

---

## Dialog Box



### SCI module

SCI module to be used for communications.

### Additional package header

This field specifies the data located at the front of the sent data package, which is not part of the data being transmitted, and generally indicates start of data. The additional package header must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not sent nor are they included in the total byte count.

# C280x SCI Transmit

---

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the host SCI Receive block.

---

## **Additional package terminator**

This field specifies the data located at the end of the sent data package, which is not part of the data being transmitted, and generally indicates end of data. The additional package terminator must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not sent nor are they included in the total byte count.

## **Enable transmit FIFO interrupt**

If checked, an interrupt is posted when FIFO is full, allowing the subsystem to take some sort of action.

## **References**

Detailed information on the SCI module is in *TMS320x281x, 280x DSP Serial Communication Interface (SCI) Reference Guide*, Literature Number SPRU051B, available at the Texas Instruments Web site.

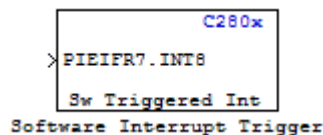
## **See Also**

C280x SCI Receive, C280x Hardware Interrupt

**Purpose** Generate software triggered nonmaskable interrupt

**Library** c280xdspchip1lib in Target for TI C2000

## Description



When you add this block to a model, the block polls the input port for the input value. When the input value is greater than the value in **Trigger software interrupt when input value is greater than**, the block posts the interrupt to a Hardware Interrupt block in the model.

To use this block, add a Hardware Interrupt block to your model to process the software triggered interrupt from this block into an interrupt service routine on the processor. Set the interrupt number in the Hardware Interrupt block to the value you set here in **CPU interrupt number**.

The CPU and PIE interrupt numbers together specify a single interrupt for a single peripheral or peripheral module. The following table maps CPU and PIE interrupt numbers to these peripheral interrupts. The row numbers are CPU values and the column numbers are the PIE values.

---

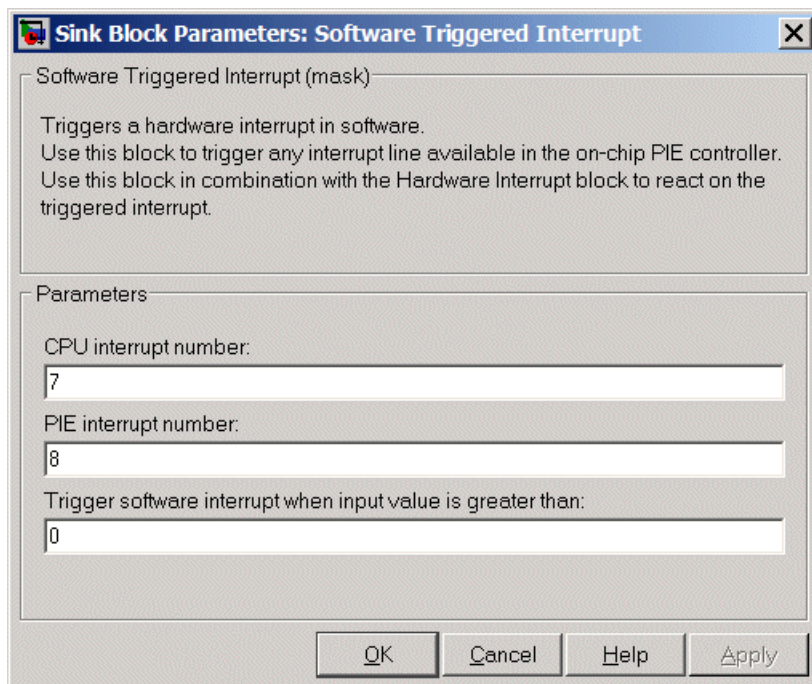
**Note** Fixed-point inputs are not supported for this block.

---





## Dialog Box



### **CPU interrupt number**

Specify the interrupt to which the block responds. Interrupt numbers are integers ranging from 1 to 12.

### **PIE interrupt number**

Enter an integer value from 1 to 8 to set the Peripheral Interrupt Expansion (PIE) interrupt number.

### **Trigger software interrupt when input value is greater than:**

Sets the value above which the block posts an interrupt. Enter the value for the level that indicates that the interrupt is asserted by a requesting routine.

# C280x SW Int Trigger

---

## **References**

For detailed information about interrupt processing, refer to *TMS320x280x DSP System Control and Interrupts Reference Guide*, SPRU712B, available at the Texas Instruments Web site.

## **See Also**

C280x Hardware Interrupt

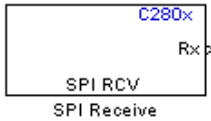
## Purpose

Receive data via serial peripheral interface (SPI) on target

## Library

c280xdspchip1lib in Target for TI C2000

## Description



The C280x SPI Receive supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode.

In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both master and slave to send and receive data simultaneously. The maximum for the clock is one quarter of the DSP controller's clock frequency.

For any given model, you can have only one C280x SPI Receive block per module. There are two modules, A and B, which can be configured through the F2808 eZdsp target preferences block.

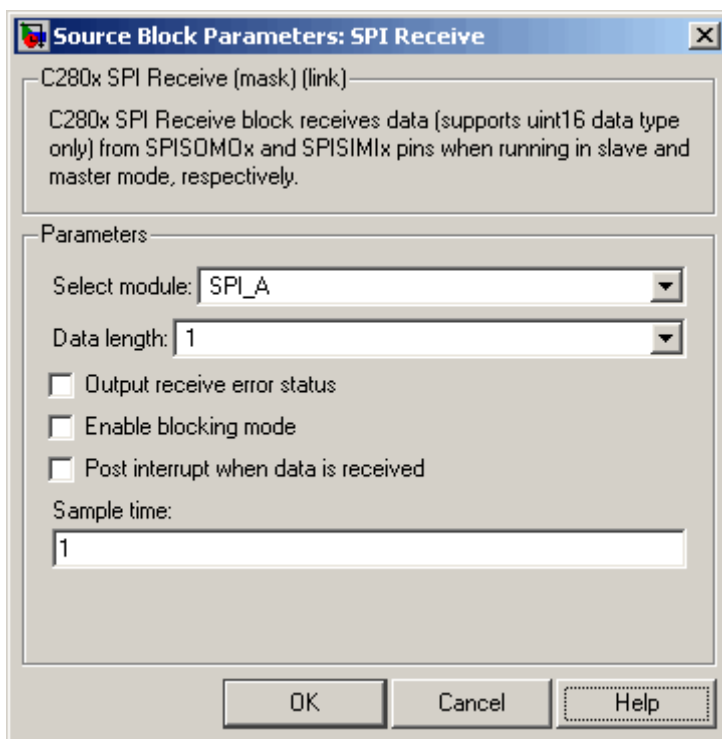
---

**Note** Many SPI-specific settings are in the **DSPBoard** section of the Target Preferences block. You should verify that these settings are correct for your application.

---

# C280x SPI Receive

## Dialog Box



### Select module

SPI module (A-D) to be used for communications.

### Data length

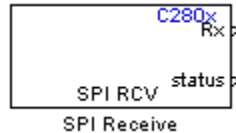
Specifies how many uint16s are expected to be received. Select 1 through 16.

### Enable blocking mode

If this option is selected, system waits until data is received before continuing processing.

## Output receive error status

When this field is checked, the c280x SPI Receive block adds another output port for the transaction status, and appears as shown in the following figure.



Error status may be one of the following values:

- 0: No errors
- 1: Data loss occurred, (Overrun: when FIFO disabled, Overflow when FIFO enabled)
- 2: Data not ready, a time out occurred while the block was waiting to receive data

## Post interrupt when data is received

Check this check box to post an asynchronous interrupt when data is received.

## Sample time

Sample time,  $T_s$ , for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt when message is received** box, and refer to "Asynchronous Interrupt Processing" on page 1-14 for a discussion of block placement and other necessary settings.

## See Also

C280x SPI Transmit, C280x Hardware Interrupt

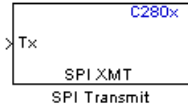
# C280x SPI Transmit

---

**Purpose** Transmit data via serial peripheral interface (SPI) to host

**Library** c280xdspchip1lib in Target for TI C2000

## Description



The C280x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both master and slave to send and receive data simultaneously. The maximum for the clock is one quarter of the DSP controller's clock frequency.

The sampling rate is inherited from the input port. The supported data type is `uint16`.

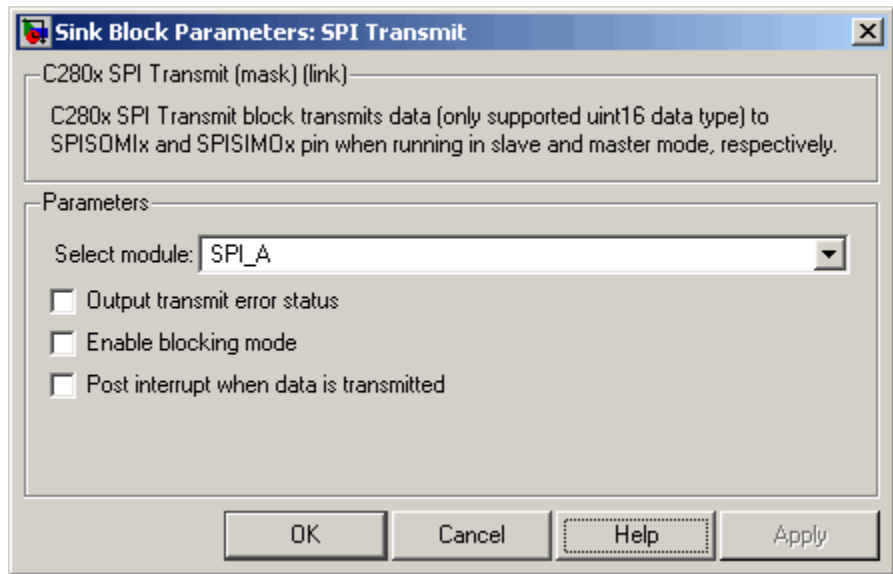
---

**Note** For any given model, you can have only one C280x SPI Transmit block per module. There are two modules, A and B, which can be configured through the F2808 eZdsp target preferences block.

Many SPI-specific settings are in the **DSPBoard** section of the target preferences block. You should verify that these settings are correct for your application.

---

## Dialog Box

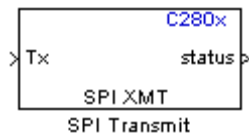


### Select module

SPI module (A-D) to be used for communications.

### Output transmit error status

When this field is checked, the c280x SPI Transmit block adds another output port for the transaction status, and appears as shown in the following figure.



Error status may be one of the following values:

- 0: No errors
- 1: A time-out occurred while the block was transmitting data

## C280x SPI Transmit

---

- 2: There is an error in the transmitted data (for example, header or terminator don't match, length of data expected is too big or too small)

### **Enable blocking mode**

If this option is selected, system waits until data is sent before continuing processing.

### **Post interrupt when data is transmitted**

Check this check box to post an asynchronous interrupt when data is transmitted.

### **See Also**

C280x SPI Receive, C280x Hardware Interrupt



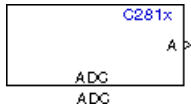
## Purpose

Analog-to-digital converter (ADC)

## Library

c281xdspchip1lib in Target for TI C2000

## Description



The C281x ADC block configures the C281x ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. The ADC block outputs digital values representing the analog input signal and stores the converted values in the result register of your digital signal processor. You use this block to capture and digitize analog signals from external sources such as signal generators, frequency generators, or audio devices.

## Triggering

The C281x ADC trigger mode depends on the internal setting of the source start-of-conversion (SOC) signal. In unsynchronized mode the ADC is usually triggered by software at the sample time intervals specified in the ADC block. For more information on configuring the specific parameters for this mode, see “Configuring Acquisition Window Width for ADC Blocks”.

In synchronized mode, the Event (EV) Manager associated with the same module as the ADC triggers the ADC. In this case, the ADC is synchronized with the pulse width modulator (PWM) waveforms generated by the same EV unit via the **ADC Start Event** signal setting. The **ADC Start Event** is set in the C281x PWM block. See that block for information on the settings.

---

**Note** The ADC cannot be synchronized with the PWM if the ADC is in cascaded mode (see below).

---

## Output

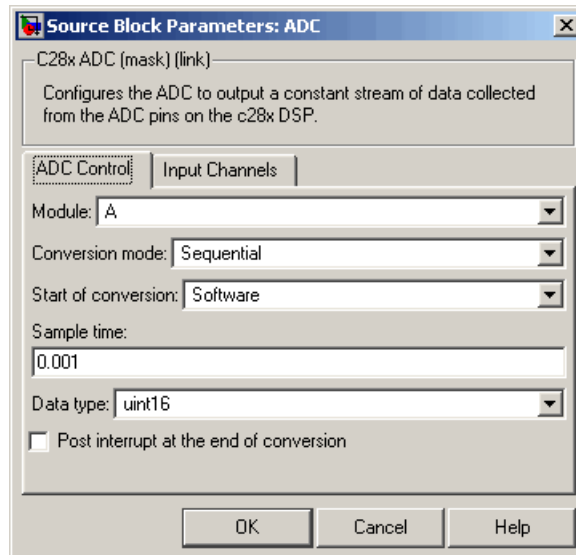
The output of the C281x ADC is a vector of uint16 values. The output values are in the range 0 to 4095 because the C281x ADC is 12-bit converter.

## Dialog Box

### Modes

The C281x ADC block supports ADC operation in dual and cascaded modes. In dual mode, either module A or module B can be used for the ADC block, and two ADC blocks are allowed in the model. In cascaded mode, both module A and module B are used for a single ADC block.

### ADC Control Pane



### Module

Specifies which DSP module to use:

- A — Displays the ADC channels in module A (ADCINA0 through ADCINA7).
- B — Displays the ADC channels in module B (ADCINB0 through ADCINB7).

- A and B — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7)

Then, use the check boxes to select the desired ADC channels.

### Conversion mode

Type of sampling to use for the signals:

- Sequential — Samples the selected channels sequentially
- Simultaneous — Samples the corresponding channels of modules A and B at the same time

### Start of conversion

Type of signal that triggers conversions to begin:

- Software — Signal from software
- EVA — Signal from Event Manager A
- EVB — Signal from Event Manager B
- External — Signal from external hardware

### Sample time

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. See “Scheduling and Timing” on page 1-13 for more information on timing. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt at the end of conversion** box, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

To set different sample times for different groups of ADC channels, you must add separate C281x ADC blocks to your model and set the desired sample times for each block.

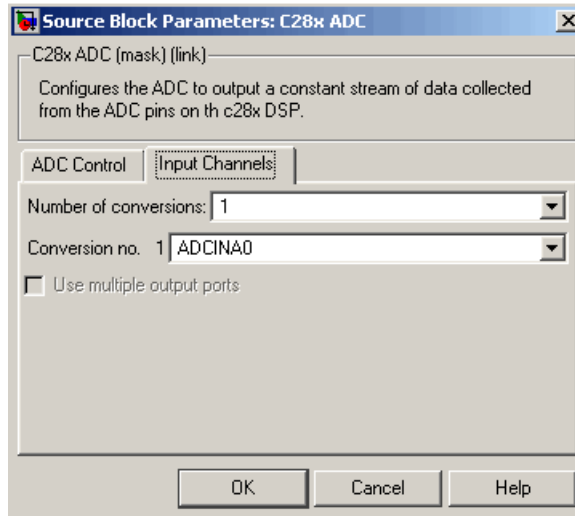
### Data type

Date type of the output data. Valid data types are auto, double, single, int8, uint8, int16, uint16, int32, or uint32.

## Post interrupt at the end of conversion

Check this check box to post an asynchronous interrupt at the end of each conversion. Note that the interrupt is always posted at the end of conversion.

## Input Channels Pane



## Number of conversions

Number of ADC channels to use for analog-to-digital conversions.

## Conversion no.

Specific ADC channel to associate with each conversion number.

In oversampling mode, a signal at a given ADC channel can be sampled multiple times during a single conversion sequence.

To oversample, specify the same channel for more than one conversion. Converted samples are output as a single vector.

## Use multiple output ports

If more than one ADC channel is used for conversion, you can use separate ports for each output and show the output ports on the

block. If you use more than one channel and do not use multiple output ports, the data is output in a single vector.

**See Also**

C281x PWM, C281x Hardware Interrupt

# C281x CAP

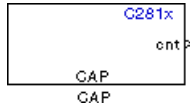
## Purpose

Receive and log capture input pin transitions

## Library

c281xdspchip1lib in Target for TI C2000

## Description



The C281x CAP block sets parameters for the capture units (CAPs) of the Event Manager (EV) module. The capture units log transitions detected on the capture unit pins by recording the times of these transitions into a two-level deep FIFO stack. The capture unit pins can be set to detect rising edge, falling edge, either type of transition, or no transition.

The C281x chip has six capture units — three associated with each EV module. Capture units 1, 2, and 3 are associated with EVA and capture units 4, 5, and 6 are associated with EVB. Each capture unit is associated with a capture input pin.

---

**Note** You can have up to two C281x CAP blocks in any one model—one block for each EV module.

---

Each group of EV module capture units can use one of two general-purpose (GP) timers on the target board. EVA capture units can use GP timer 1 or 2. EVB capture units can use GP timer 3 or 4. When a transition occurs, the value of the selected timer is stored in the two-level deep FIFO stack.

## Outputs

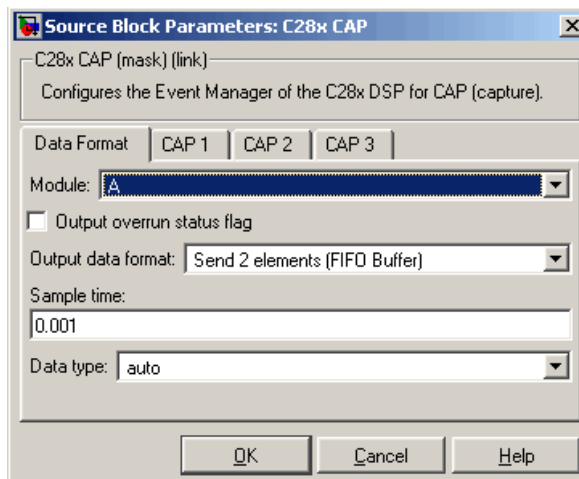
This block has up to two outputs: a cnt (count) output and an optional, FIFO status flag output. The cnt output increments each time a transition of the selected type occurs. The status flag outputs are

- 0 — The FIFO is empty. Either no captures have occurred or the previously stored capture(s) have been read from the stack. (The binary version of this flag is 00.)
- 1 — The FIFO has one entry in the top register of the stack. (The binary version of this flag is 01.)

- 2 — The FIFO has two entries in the stack registers. (The binary version of this flag is 10.)
- 3 — The FIFO has two entries in the stack registers and one or more captured values have been lost. This occurs because another capture occurred before the FIFO stack was read. The new value is placed in the bottom register. The bottom register value is pushed to the top of the stack and the top value is pushed out of the stack. (The binary version of this flag is 11.)

## Dialog Box

### Data Format Pane



### Module

Select the Event Manager (EV) module to use:

- A — Use CAPs 1, 2, and 3.
- B — Use CAPs 4, 5, and 6.

### Output overrun status flag

Select to output the status of the elements in the FIFO. The data type of the status flag is uint16.

## Send data format

The type of data to output:

- **Send 2 elements (FIFO Buffer)** — Sends the latest two values. The output is updated when there are two elements in the FIFO, which is indicated by bit 13 or 11 or 9 being sent (CAP x FIFO). If the CAP is polled when fewer than two elements are captures, old values are repeated. The CAP registers are read as follows:
  - a** The CAP x FIFO status bits are read and the value is stored in the status flag.
  - b** The top value of the FIFO is read and stored in the output at index 0.
  - c** The new top value of the FIFO (the previously stored bottom stack value) is read and stored in the output at index 1.
- **Send 1 element (oldest)** — Sends the older of the two most recent values. The output is updated when there is at least one element in the FIFO, which is indicated by any of the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:
  - a** The CAP x FIFO status bits are read and the value is stored in the status flag.
  - b** The top value of the FIFO is read and stored in the output.
- **Send 1 element (latest)** — Sends the most recent value. The output is updated when there is at least one element in the FIFO, which is indicated by any of the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:
  - a** The CAP x FIFO status bits are read and the value is stored in the status flag.
  - b** If there are two entries in the FIFO, the bottom value is read and stored in the output. If there is only one entry in the FIFO, the top value is read and stored in the output.



## Sample time

Time between outputs from the FIFO. If new data is not available, the previous data is sent.

## Data type

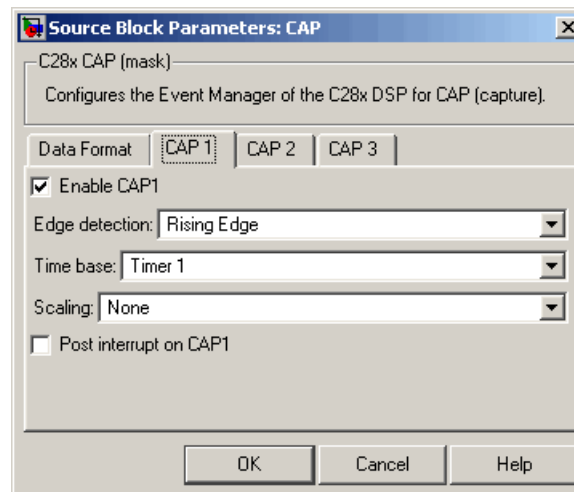
Data type of the output data. Available options are auto, double, single, int8, uint8, int16, uint16, int32, uint32, and boolean. The auto option uses the data type of a connected block that outputs data to this block. If this block does not receive any input, auto sets the data type to double.

---

**Note** The output of the C281x CAP block can be vectorized.

---

## CAP# Pane



The CAP# panes set parameters for individual CAPs. The particular CAP affected by a CAP# pane depends on the EV module you selected:

- **CAP1** controls CAP 1 or CAP 4, for EV module A or B, respectively.

- **CAP2** controls CAP 2 or CAP 5, for EV module A or B, respectively.
- **CAP3** controls CAP 3 or CAP 6, for EV module A or B, respectively.

### Enable CAP#

Select to use the specified capture unit pin.

### Edge Detection

Type of transition detection to use for this CAP. Available types are Rising Edge, Falling Edge, Both Edges, and No transition.

### Time Base

The target board GP timer to use. CAPs 1, 2, and 3 can use Timer 1 or Timer 2. CAPs 4, 5, and 6 can use Timer 3 or Timer 4.

### Scaling

Clock divider factor by which to prescale the selected GP timer to produce the desired timer counting rate. Available options are none, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, and 1/128. The resulting rate for each option is shown below.

Scaling	Resulting Rate ( $\mu$ s)
none	0.01334
1/2	0.02668
1/4	0.05336
1/8	0.10672
1/16	0.21344
1/32	0.42688
1/64	0.85376
1/128	1.70752

---

**Note** The above rates assume a 75 MHz input clock.

---

**Post interrupt on CAP#**

Check this check box to post an asynchronous interrupt on CAP#.

**See Also**

C281x Hardware Interrupt

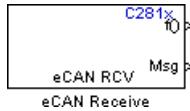
# C281x eCAN Receive

---

**Purpose** Enhanced Control Area Network receive mailbox

**Library** c281xdspchip1lib in Target for TI C2000

## Description



The C281x enhanced Control Area Network (eCAN) Receive block generates source code for receiving eCAN messages through an eCAN mailbox. The eCAN module on the DSP chip provides serial communication capability and has 32 mailboxes configurable for receive or transmit. The C281x supports eCAN data frames in standard or extended format.

The C281x eCAN Receive block has up to two and, optionally, three output ports.

- The first output port is the function call port, and a function call subsystem should be connected to this port. When a new message is received, this subsystem is executed.
- The second output port is the message data port. The received data is output in the form of a vector of elements of the selected data type. The length of the vector is always 8 bytes.
- The third output port is optional and appears only if **Output message length** is selected.

## Dialog Box

**Source Block Parameters: eCAN Receive**

C28x eCAN Receive (mask)

Configures an eCAN mailbox to receive messages from the eCAN bus pins on the c28x DSP. When the message is received, emits the function call to the connected function-call subsystem as well as outputs the message data in selected format and the message data length in bytes.

Parameters

Mailbox number:

Message identifier:

Message type:

Sample time:

Data type:

Output message length

Post interrupt when message is received

OK Cancel Help

### Mailbox number

Unique number between 0 and 15 for standard or between 0 and 31 for enhanced CAN mode. It refers to a mailbox area in RAM. In standard mode, the mailbox number determines priority.

### Message identifier

Identifier of length 11 bits for standard frame size or length 29 bits for extended frame size in decimal, binary, or hex. If in binary or hex, use `bin2dec('')` or `hex2dec('')`, respectively, to convert the entry. The message identifier is associated with a receive mailbox. Only messages that match the mailbox message identifier are accepted into it.

### Message type

Select Standard (11-bit identifier) or Extended (29-bit identifier).

## Sample time

Frequency with which the mailbox is polled to determine if a new message has been received. A new message causes a function call to be emitted from the mailbox. If you want to update the message output only when a new message arrives, then the block needs to be executed asynchronously. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt when message is received** box, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

---

**Note** For information about setting the timing parameters of the CAN module see “Configuring Timing Parameters for CAN Blocks”.

---

## Data type

Type of data in the data vector. The length of the vector for the received message is at most 8 bytes. If the message is less than 8 bytes, the data buffer bytes are right-aligned in the output. Only uint16 (vector length = 4 elements) or uint32 (vector length = 8 elements) data are allowed. The data are unpacked as follows using the data buffer, which is 8 bytes.

For uint16 data,

```
Output[0] = data_buffer[1..0];
Output[1] = data_buffer[3..2];
Output[2] = data_buffer[5..4];
Output[3] = data_buffer[7..6];
```

For uint32 data,

```
Output[0] = data_buffer[3..0];
Output[1] = data_buffer[7..4];
```

For example, if the received message has two bytes:

```
data_buffer[0] = 0x21
data_buffer[1] = 0x43
```

the uint16 output would be:

```
Output[0] = 0x4321
Output[1] = 0x0000
Output[2] = 0x0000
Output[3] = 0x0000
```

### **Output message length**

Select to output the message length in bytes to the third output port. If not selected, the block has only two output ports.

### **Post interrupt when message is received**

Check this check box to post an asynchronous interrupt when a message is received.

## **References**

Detailed information on the eCAN module is in *TMS320F28x DSP Enhanced Control Area Network (eCAN) Reference Guide*, Literature Number SPRU074A, available at the Texas Instruments Web site.

## **See Also**

C281x eCAN Transmit, C281x Hardware Interrupt

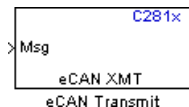
# C281x eCAN Transmit

---

**Purpose** Enhanced Control Area Network transmit mailbox

**Library** c281xdspchip1lib in Target for TI C2000

## Description



The C281x enhanced Control Area Network (eCAN) Transmit block generates source code for transmitting eCAN messages through an eCAN mailbox. The eCAN module on the DSP chip provides serial communication capability and has 32 mailboxes configurable for receive or transmit. The C28x supports eCAN data frames in standard or extended format.

---

**Note** Fixed-point inputs are not supported for this block.

---

## Data Vectors

The length of the vector for each transmitted mailbox message is 8 bytes. Input data are always right-aligned in the message data buffer. Only `uint16` (vector length = 4 elements) or `uint32` (vector length = 8 elements) data are accepted. The following examples show how the different types of input data are aligned in the data buffer

For input of type `uint32`,

```
inputdata [0] = 0x12345678
```

the data buffer is:

```
data buffer[0] = 0x78
data buffer[1] = 0x56
data buffer[2] = 0x34
data buffer[3] = 0x12
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```



For input of type uint16,

```
inputdata [0] = 0x1234
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x00
data buffer[3] = 0x00
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

For input of type uint16[2], which is a two-element vector,

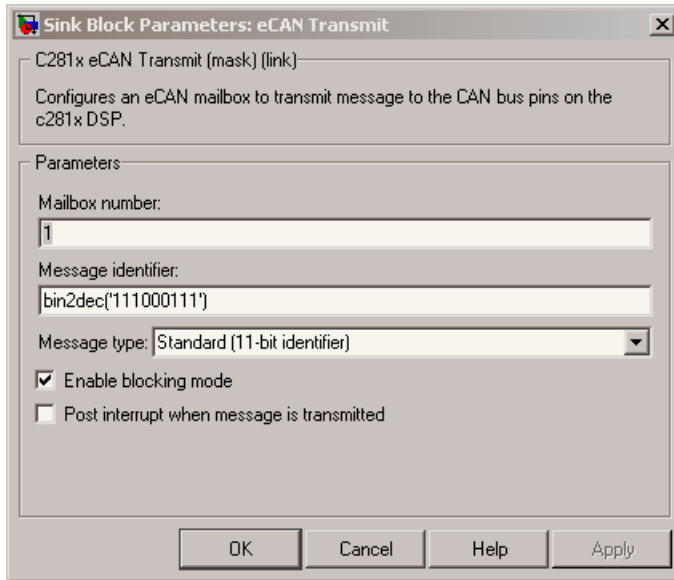
```
inputdata [0] = 0x1234
inputdata [1] = 0x5678
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x78
data buffer[3] = 0x56
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

# C281x eCAN Transmit

## Dialog Box



### Mailbox number

Unique number between 0 and 15 for standard or between 0 and 31 for enhanced CAN mode. It refers to a mailbox area in RAM. In standard mode, the mailbox number determines priority.

### Message identifier

Identifier of length 11 bits for standard frame size or length 29 bits for extended frame size in decimal, binary, or hex. If in binary or hex, use `bin2dec(' ')` or `hex2dec(' ')`, respectively, to convert the entry. The message identifier is coded into a message that is sent to the CAN bus.

### Message type

Select Standard (11-bit identifier) or Extended (29-bit identifier).

### Enable blocking mode

If selected, the CAN block code waits indefinitely for a transmit (XMT) acknowledge. If cleared, the CAN block code does not wait

for a transmit (XMT) acknowledge, which is useful when the hardware might fail to acknowledge transmissions.

**Post interrupt when message is transmitted**

If selected, an asynchronous interrupt is posted when data is transmitted.

---

**Note** For information about setting the timing parameters of the CAN module see “Configuring Timing Parameters for CAN Blocks”.

---

**References**

Detailed information on the eCAN module is in *TMS320F28x DSP Enhanced Control Area Network (eCAN) Reference Guide*, Literature Number SPRU074A, available at the Texas Instruments Web site.

**See Also**

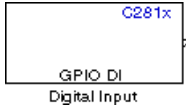
C281x eCAN Receive

# C281x GPIO Digital Input

**Purpose** General-purpose I/O pins for digital input

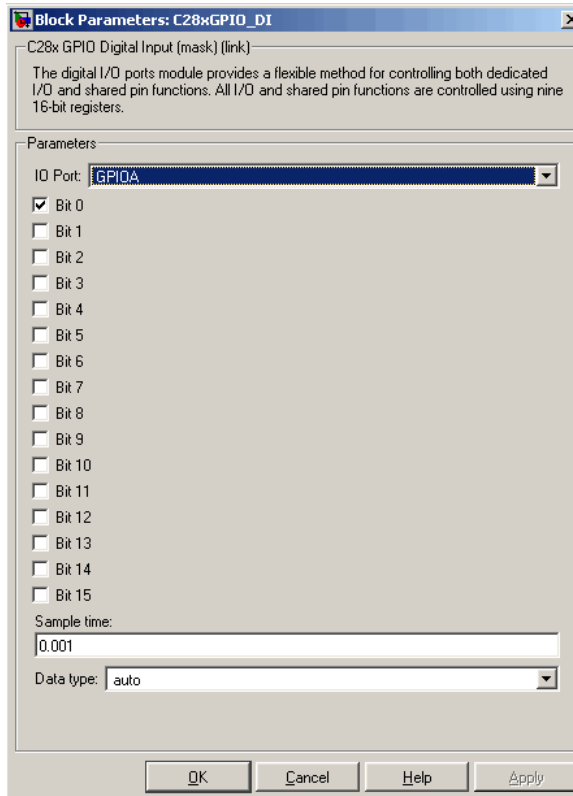
**Library** c281xdspchip1lib in Target for TI C2000

## Description



This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital input. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.

## Dialog Box



## IO Port

Select the input/output port to use: GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, or GPIOG and select the I/O Port bits to enable for digital input. (Note that there is no GPIOC port on the C281x.) If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Multiple GPIO DI blocks cannot share the same I/O port.

---

**Note** The input function of the digital I/O and the input path to the related peripheral are always enabled on the board. If you configure a pin as digital I/O, the corresponding peripheral function cannot be used.

---

The following tables show the shared pins.

## GPIO A MUX

Bit	Peripheral Name (Bit = 1)	GPIO Name (Bit = 0)
0	PWM1	GPIOA0
1	PWM2	GPIOA1
2	PWM3	GPIOA2
3	PWM4	GPIOA3
4	PWM5	GPIOA4
5	PWM6	GPIOA5
8	QEP1/CAP1	GPIOA8

# C281x GPIO Digital Input

## GPIO A MUX (Continued)

Bit	Peripheral Name (Bit = 1)	GPIO Name (Bit = 0)
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

## GPIO B MUX

Bit	Peripheral Name (Bit = 1)	GPIO Name (Bit = 0)
0	PWM7	GPIOB0
1	PWM8	GPIOB1
2	PWM9	GPIOB2
3	PWM10	GPIOB3
4	PWM11	GPIOB4
5	PWM12	GPIOB5
8	QEP3/CAP4	GPIOB8
9	QEP4/CAP5	GPIOB9
10	CAP6	GPIOB10

### Sample time

Time interval, in seconds, between consecutive input from the pins.

### Data type

Data type of the data to obtain from the GPIO pins. The data is read as 16-bit integer data and then cast to the selected data type. Valid data types are auto, double, single, int8, uint8, int16, uint16, int32, uint32 or boolean.

---

**Note** The width of the vectorized data output by this block is determined by the number of bits selected in the **Block Parameters** dialog box.

---

**See Also**      C281x GPIO Digital Output

# C281x GPIO Digital Output

---

**Purpose** General-purpose I/O pins for digital output

**Library** c281xdspchip1lib in Target for TI C2000

**Description** This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital output. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.



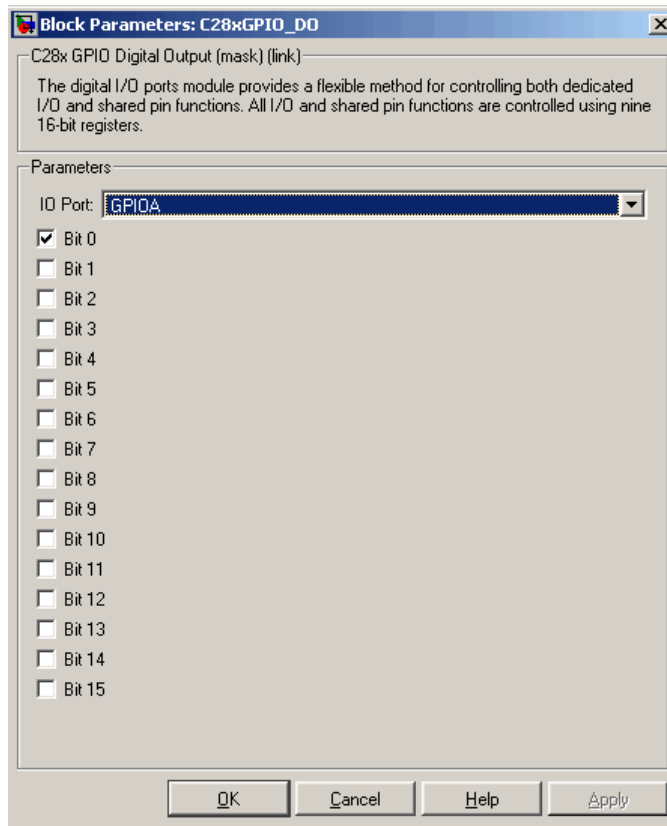
---

**Note** Fixed-point inputs are not supported for this block.

---



## Dialog Box



### IO Port

Select the input/output port to use: GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, or GPIOG and select the I/O Port bits to enable for digital input. (Note that there is no GPIOC port on the C281x.) If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Note that multiple GPIO DO blocks cannot share the same I/O port.

# C281x GPIO Digital Output

---

**Note** The input function of the digital I/O and the input path to the related peripheral are always enabled on the board. If you configure a pin as digital I/O, the corresponding peripheral function cannot be used.

---

The following tables show the shared pins.

## GPIO A MUX

Bit	Peripheral Name (Bit = 1)	GPIO Name (Bit = 0)
0	PWM1	GPIOA0
1	PWM2	GPIOA1
2	PWM3	GPIOA2
3	PWM4	GPIOA3
4	PWM5	GPIOA4
5	PWM6	GPIOA5
8	QEP1/CAP1	GPIOA8
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

## GPIO B MUX

Bit	Peripheral Name (Bit = 1)	GPIO Name (Bit = 0)
0	PWM7	GPIOB0
1	PWM8	GPIOB1

## GPIO B MUX (Continued)

Bit	Peripheral Name (Bit = 1)	GPIO Name (Bit = 0)
2	PWM9	GPIOB2
3	PWM10	GPIOB3
4	PWM11	GPIOB4
5	PWM12	GPIOB5
8	QEP3/CAP4	GPIOB8
9	QEP4/CAP5	GPIOB9
10	CAP6	GPIOB10

### See Also

C281x GPIO Digital Input

# C281x Hardware Interrupt

---

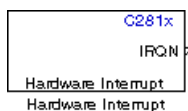
## Purpose

Interrupt Service Routine to handle hardware interrupt on C281x processor

## Library

c281xdspchip1lib in Target for TI C2000

## Description



For many systems, an execution scheduling model based on a timer interrupt is not sufficient to ensure a real-time response to external events. The C281x Hardware Interrupt block addresses this problem by allowing for the asynchronous processing of interrupts triggered by events managed by other blocks in the C281x DSP Chip Support Library.

The C281x blocks that can generate an interrupt for asynchronous processing are:

- C281x ADC
- C281x CAP
- C281x eCAN Receive
- C281x Timer
- C281x SCI Receive
- C281x SCI Transmit
- C281x Software Interrupt Trigger
- C281x SPI Receive
- C281x SPI Transmit

Only one Hardware Interrupt block can be used in a model. To handle multiple interrupts, place a Demux block at the output of the Hardware Interrupt block to direct function calls to the appropriate function-call subsystems.

For details about this block, refer to C281x Hardware Interrupt block in your Link for Code Composer Studio Development Tools documentation.

## Vectorized Output

This block outputs a function call. The size of the function call line equals the number of interrupts the block is set to handle. The block dialog box presents four parameters for each interrupt. These parameters comprise a set of four vectors of equal length. Each interrupt is represented by one element from each parameter (four elements total), one from the same position in each of these vectors.

The following parameters describe each interrupt:

- CPU interrupt numbers
- PIE interrupt numbers
- Task priorities
- Preemption flags

Thus, one interrupt is described by a CPU interrupt number, a PIE interrupt number, a task priority, and a preemption flag.

The CPU and PIE interrupt numbers together uniquely specify a single interrupt for a single peripheral or peripheral module. The following table maps CPU and PIE interrupt numbers to these peripheral interrupts. The row numbers are CPU values and the column numbers are the PIE values.

---

**Note** The TINT0 (TIMER 0) interrupt is always reserved, and will generate errors if used.

---

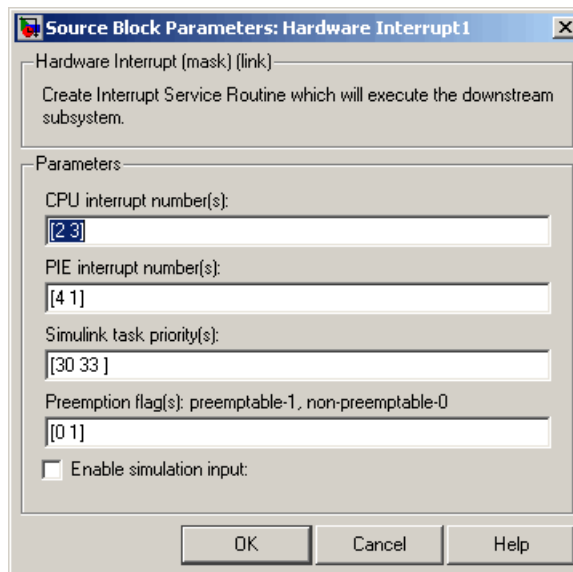
### C281x Peripheral Interrupt Vector Values

	1	2	3	4	5	6	7	8
1	PDPINTA (EV-A)	PDPINTB (EV-B)	Reserved	XINT1	XINT2	ADCINT (ADC)	TINT0 (TIMER 0)	WAKEINT (LPM/WD)
2	CMP1INT (EV-A)	CMP2INT (EV-A)	CMP3INT (EV-A)	T1PINT (EV-A)	T1CINT (EV-A)	T1UFINT (EV-A)	T1OFINT (EV-A)	Reserved
3	T2PINT (EV-A)	T2CINT (EV-A)	T2UFINT (EV-A)	T2OFINT (EV-A)	CAPINT1 (EV-A)	CAPINT2 (EV-A)	CAPINT3 (EV-A)	Reserved
4	CMP4INT (EV-B)	CMP5INT (EV-B)	CMP6INT (EV-B)	T3PINT (EV-B)	T3CINT (EV-B)	T3UFINT (EV-B)	T3OFINT (EV-B)	Reserved
5	T4PINT (EV-B)	T4CINT (EV-B)	T4UFINT (EV-B)	T4OFINT (EV-B)	CAPINT4 (EV-B)	CAPINT5 (EV-B)	CAPINT6 (EV-B)	Reserved
6	SPIRXINTA (SPI)	SPITXINTA (SPI)	Reserved	Reserved	MRINT (McBSP)	MXINT (McBSP)	Reserved	Reserved
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
8	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
9	SCIRXINTA (SCI-A)	SCITXINTA (SCI-A)	SCIRXINTB (SCI-B)	SCITXINTB (SCI-B)	ECAN0INT (CAN)	ECAN1INT (CAN)	Reserved	Reserved
10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

The task priority indicates the relative importance of tasks associated with the asynchronous interrupts. If an interrupt triggers a higher-priority task while a lower-priority task is running, the execution of the lower-priority task is suspended while the higher-priority task is executed. The lowest value represents the highest priority. The default priority value of the base rate task is 40, so the priority value for each asynchronously triggered task must be less than 40 for these tasks to cause the suspension of the base rate task.

The preemption flag determines whether a given interrupt is preemptable. Preemption overrides prioritization. A preemptable task of higher priority can be preempted by a lower priority nonpreemptable task.

## Dialog Box



### CPU interrupt number(s)

Enter a vector of CPU interrupt numbers for the interrupts you want to process asynchronously.

# C281x Hardware Interrupt

---

See the table of C281x Peripheral Interrupt Vector Values on page 7-134 for a mapping of CPU interrupt number to interrupt names.

## **PIE interrupt number(s)**

Enter a vector of PIE interrupt numbers for the interrupts you want to process asynchronously.

See the table of C281x Peripheral Interrupt Vector Values on page 7-134 for a mapping of CPU interrupt number to interrupt names.

## **Simulink task priority(s)**

Enter a vector of task priorities for the interrupts you want to process asynchronously.

See the discussion of this block's "Vectorized Output" on page 7-133 for an explanation of task priorities.

## **Preemption flag(s)**

Enter a vector of preemption flags for the interrupts you want to process asynchronously.

See the discussion of this block's "Vectorized Output" on page 7-133 for an explanation of preemption flags.

## **Enable simulation input**

Select this check box if you want to be able to test asynchronous interrupt processing in the context of your Simulink model.

---

**Note** Using this check box is the only way you can test asynchronous interrupt processing behavior in Simulink.

---

## **References**

For detailed information interrupt processing, refer to *TMS320x281x DSP System Control and Interrupts Reference Guide*, SPRU078C, available at the Texas Instruments Web site.

## **See Also**

C281x SW Int Trigger, C281x Timer, Idle Task



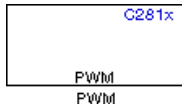
## Purpose

Pulse width modulators (PWMs)

## Library

c281xdspchip1ib in Target for TI C2000

## Description



F2812 DSPs include a suite of pulse width modulators (PWMs) used to generate various signals. This block provides options to set the A or B module Event Managers to generate the waveforms you require. The twelve PWMs are configured in six pairs, with three pairs in each module.

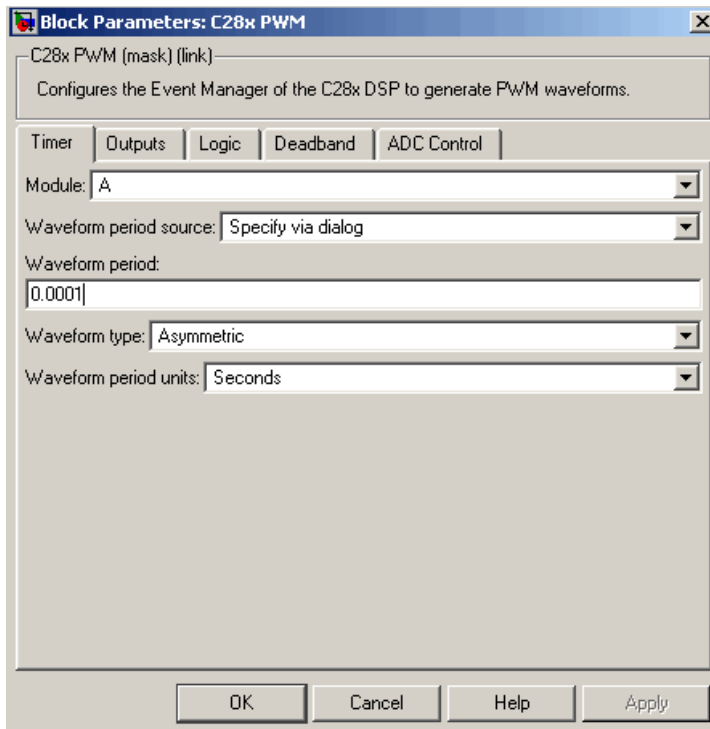
---

**Note** All inputs to the C281x PWM block must be scalar values.

---

## Dialog Box

### Timer Pane



### Module

Specifies which target PWM pairs to use:

- A — Displays the PWMs in module A (PWM1/PWM2, PWM3/PWM4, and PWM5/PWM6).
- B — Displays the PWMs in module B (PWM7/PWM8, PWM9/PWM10, and PWM11/PWM12).

---

**Note** PWMs in module A use Event Manager A, Timer 1, and PWMs in module B use Event Manager B, Timer 3.

---

## **Waveform period source**

Source from which the waveform period value is obtained. Select **Specify via dialog** to enter the value in **Waveform period** or select **Input port** to use a value from the input port.

## **Waveform period**

Period of the PWM waveform measured in clock cycles or in seconds, as specified in the **Waveform period units**.

---

**Note** The term *clock cycles* refers to the high-speed peripheral clock on the F2812 chip. This clock is 75 MHz by default because the high-speed peripheral clock prescaler is set to 2 (150 MHz/2).

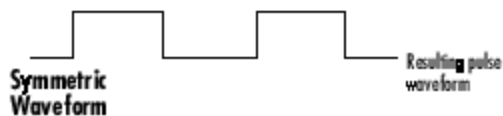
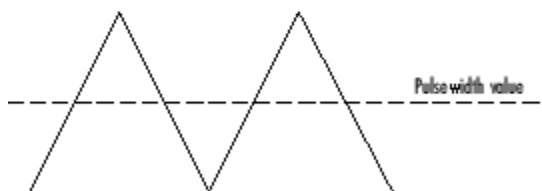
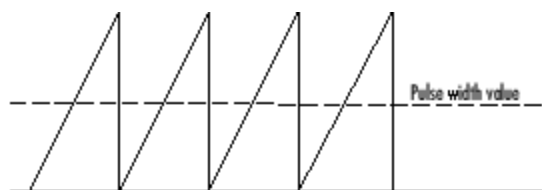
---

## **Waveform type**

Type of waveform to be generated by the PWM pair. The F2812 PWMs can generate two types of waveforms: **Asymmetric** and **Symmetric**. The following illustration shows the difference between the two types of waveforms.

# C281x PWM

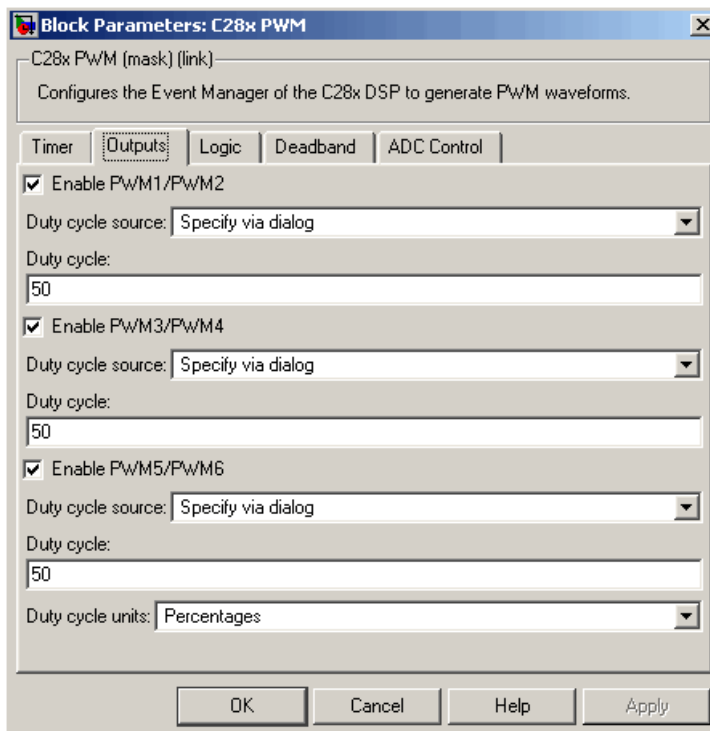
---



## Waveform period units

Units in which to measure the waveform period. Options are Clock cycles, which refer to the high-speed peripheral clock on the F2812 chip (75 MHz), or Seconds. Note that changing these units changes the **Waveform period** value and the **Duty cycle** value and **Duty cycle units** selection.

## Outputs Pane



### Enable PWM#/PWM#

Check to activate the PWM pair. PWM1/PWM2 are activated via the Output 1 pane, PWM3/PWM4 are on Output 2, and PWM5/PWM6 are on Output 3.

### Duty cycle source

Source from which the duty cycle for the specific PWM pair is obtained. Select **Specify via dialog** to enter the value in **Duty cycle** or select **Input port** to use a value from the input port.

**Duty cycle**

Ratio of the PWM waveform pulse duration to the PWM waveform period expressed in **Duty cycle units**.

**Duty cycle units**

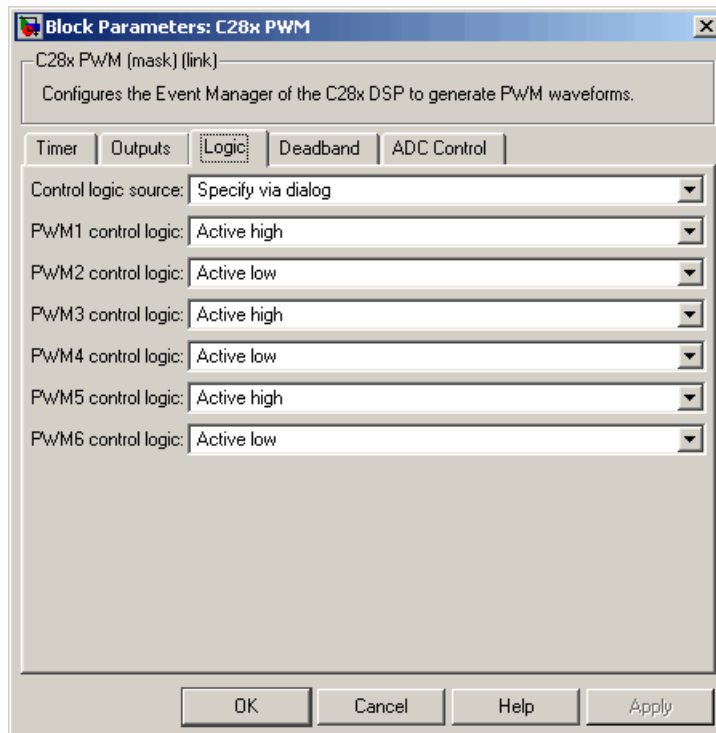
Units for the duty cycle. Valid choices are Clock cycles and Percentages. Note that changing these units changes the **Duty cycle** value, and the **Waveform period** value and **Waveform period units** selection.

---

**Note** Using percentages may cause some additional computation time in generated code. This may or may not be noticeable in your application.

---

## Logic Pane



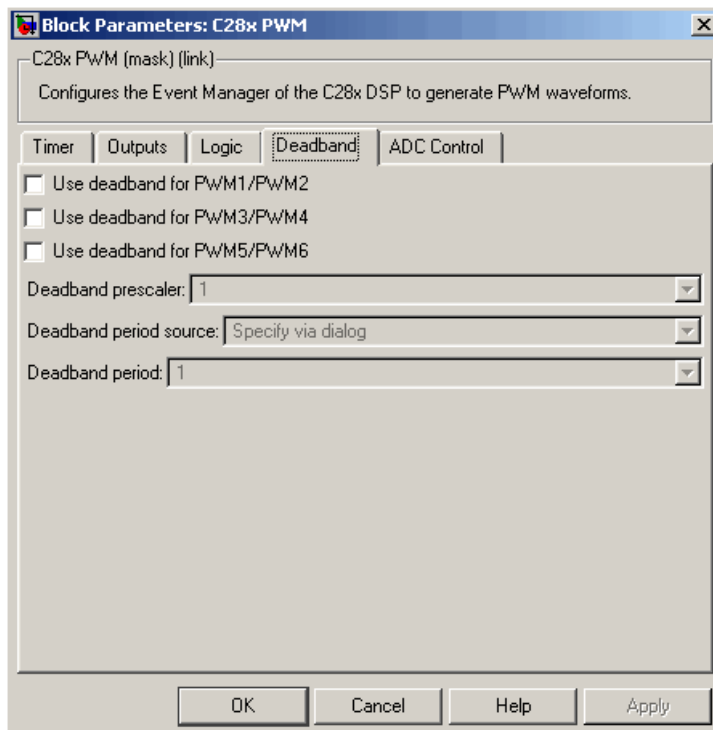
### Control logic source

Source from which the control logic is obtained for all PWMs. Select `Specify via dialog` to enter the values in the **PWM# control logic** fields or select `Input port` to use values from the input port.

### PWM# control logic

Control logic trigger for the PWM. `Forced high` causes the pulse value to be high. `Active high` causes the pulse value to go from low to high and `Active low` causes the pulse value to go from high to low. `Forced low` causes the pulse value to be low.

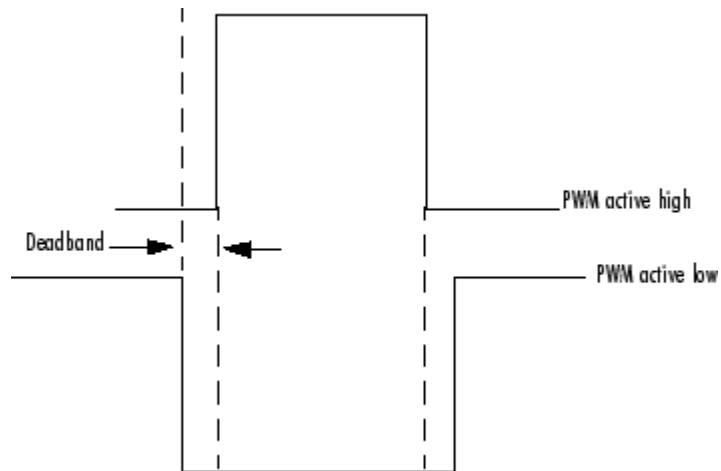
## Deadband Pane



### Use deadband for PWM#/PWM#

Enables a deadband area of no signal overlap at the beginning of particular PWM pair signals. The following figure shows the deadband area.





### **Deadband prescaler**

Number of clock cycles, which, when multiplied by the Deadband period, determines the size of the deadband. Selectable values are 1, 2, 4, 8, 16, and 32.

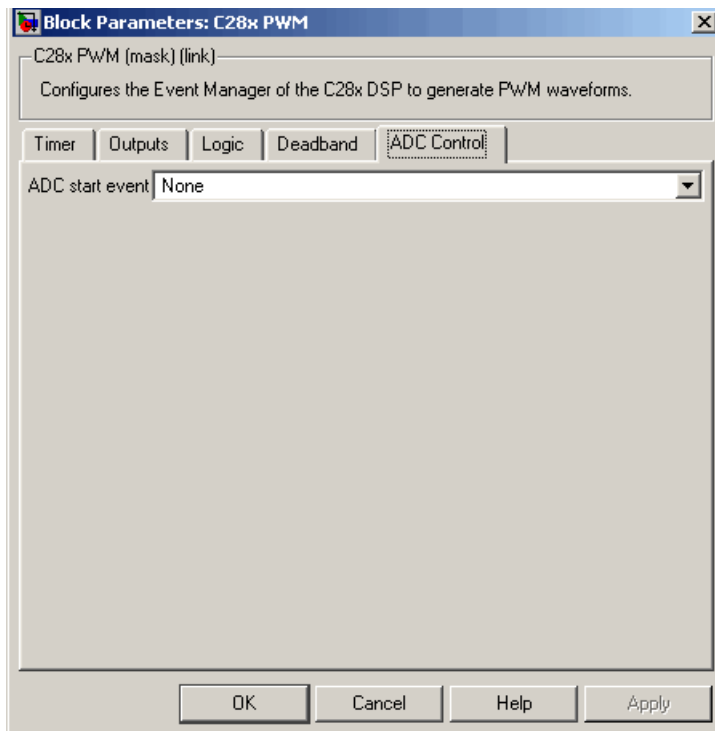
### **Deadband period source**

Source from which the deadband period is obtained. Select Specify via dialog to enter the values in the **Deadband period** field or select Input port to use a value, in clock cycles, from the input port.

### **Deadband period**

Value that, when multiplied by the Deadband prescaler, determines the size of the deadband. Selectable values are from 1 to 15.

## ADC Control Pane



### ADC start event

Controls whether this PWM and ADC associated with the same EV module are synchronized. Select None for no synchronization or select an interrupt to generate the source start-of-conversion (SOC) signal for the associated ADC.

- None — The ADC and PWM are not synchronized. The EV does not generate an SOC signal and the ADC is triggered by software (that is, the A/D conversion occurs when the ADC block is executed in the software).

- Underflow interrupt — The EV generates an SOC signal for the ADC associated with the same EV module when the board's General Purpose (GP) timer counter reaches a hexadecimal value of FFFF.
- Period interrupt — The EV generates an SOC signal for the ADC associated with the same EV module when the value in GP timer matches the value in the period register. The value set in **Waveform period** above determines the value in the register.

---

**Note** If you select Period interrupt and specify a sampling time less than the specified (**Waveform period**)/(**Event timer clock speed**), zero-order hold interpolation will occur. (For example, if you enter 64000 as the waveform period, the period for the timer is  $64000/75 \text{ MHz} = 8.5333\text{e-}004$ . If you enter a **Sample time** in the C281x ADC dialog box that is less than this result, it will cause zero-order hold interpolation.)

---

- Compare interrupt — The EV generates an SOC signal for the ADC associated with the same EV module when the value in the GP timer matches the value in the compare register. The value set in **Duty cycle** above determines the value in the register.

## See Also

C281x ADC

# C281x QEP

---

## Purpose

Quadrature encoder pulse circuit

## Library

c281xdspchip1lib in Target for TI C2000

## Description

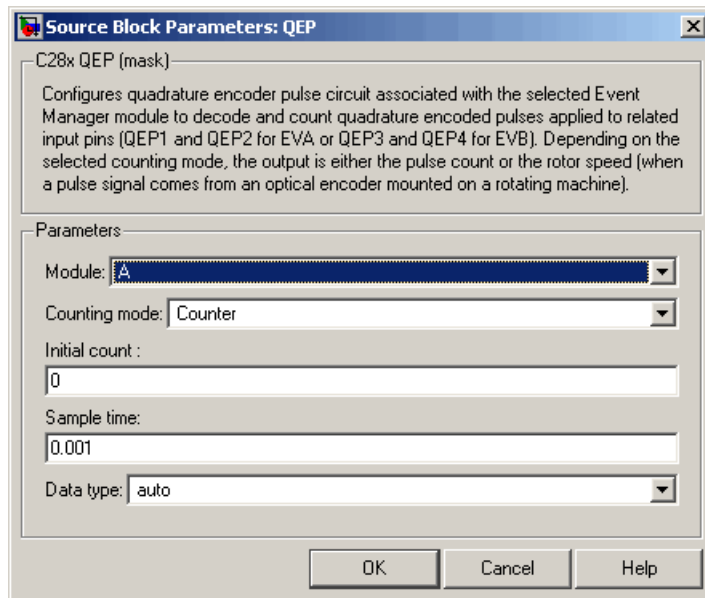


Each F2812 Event Manager has three capture units, which can log transitions on its capture unit pins. Event Manager A (EVA) uses capture units 1, 2, and 3. Event Manager B (EVB) uses capture units 4, 5, and 6.

The quadrature encoder pulse (QEP) circuit decodes and counts quadrature encoded input pulses on these capture unit pins. QEP pulses are two sequences of pulses with varying frequency and a fixed phase shift of 90 degrees (or one-quarter of a period). Both edges of the QEP pulses are counted so the frequency of the QEP clock is four times the input sequence frequency.

The QEP, in combination with an optical encoder, is particularly useful for obtaining speed and position information from a rotating machine. Logic in the QEP circuit determines the direction of rotation by which sequence is leading. For module A, if the QEP1 sequence leads, the general-purpose (GP) Timer counts up and if the QEP2 sequence leads, the timer counts down. The pulse count and frequency determine the angular position and speed.

## Dialog Box



### Module

Specifies which QEP pins to use:

- A — Uses QEP1 and QEP2 pins.
- B — Uses QEP3 and QEP4 pins.

### Counting mode

Specifies how to count the QEP pulses:

- Counter — Count the pulses based on the board's GP Timer 2 (or GP Timer 4 for EVB).
- RPM — Count the machine's revolutions per minute.

### Positive rotation

Defines whether to use Clockwise or Counterclockwise as the direction to use as positive rotation. This field appears only if you select RPM above.

**Encoder resolution**

Number of QEP pulses per revolution. This field appears only if you select RPM above.

**Initial count**

Initial value for the counter. The default is 0.

**Sample time**

Time interval, in seconds, between consecutive reads from the QEP pins.

**Data type**

Data type of the QEP pin data. The data is read as 16-bit data and then cast to the selected data type. Valid data types are auto, double, single, int8, uint8, int16, uint16, int32, uint32 or boolean.

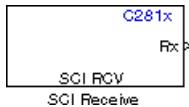
## Purpose

Receive data on target via serial communications interface (SCI) from host

## Library

c281xdspchip1lib in Target for TI C2000

## Description



The C281x SCI Receive block supports asynchronous serial digital communications between the target and other asynchronous peripherals in nonreturn-to-zero (NRZ) format. This block configures the C281x DSP target to receive scalar or vector data from the COM port via the C28x target's COM port.

---

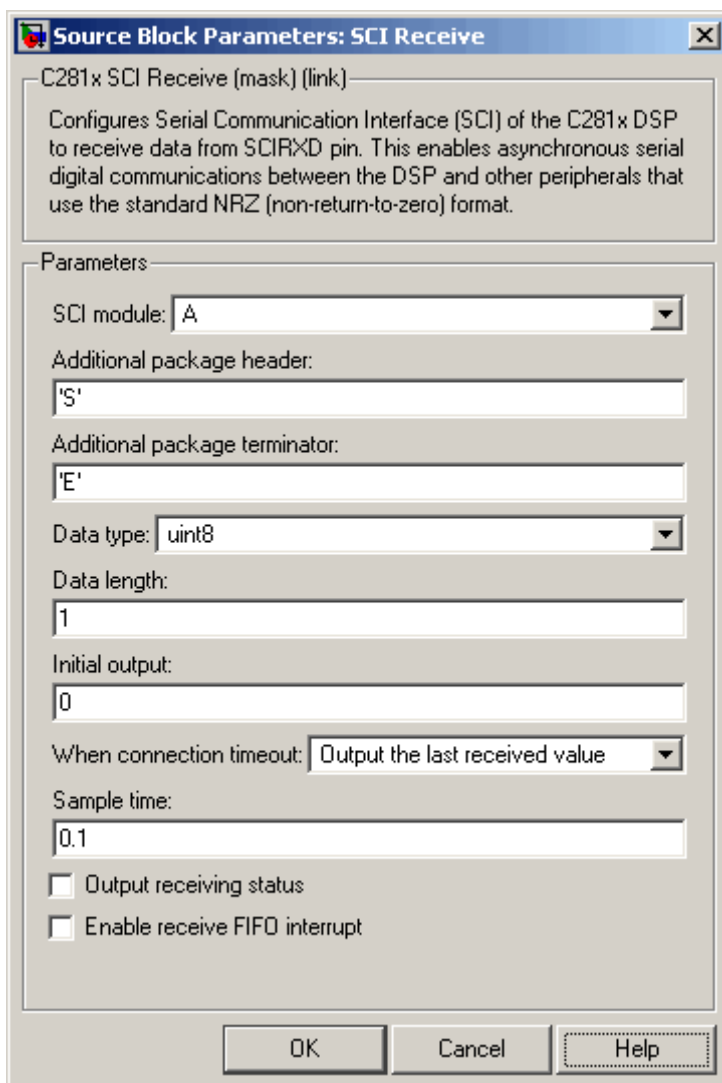
**Note** For any given model, you can have only one C281x SCI Receive block per module. There are two modules, A and B, which can be configured through the F2812 eZdsp target preferences block.

Many SCI-specific settings are in the **DSPBoard** section of the F2812 eZdsp target preferences block. You should verify that these settings are correct for your application.

---

# C281x SCI Receive

## Dialog Box



### SCI module

SCI module to be used for communications.



## Additional package header

This field specifies the data located at the front of the received data package, which is not part of the data being received, and generally indicates start of data. The additional package header must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not received nor are they included in the total byte count.

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the host SCI Transmit block.

---

## Additional package terminator

This field specifies the data located at the end of the received data package, which is not part of the data being received, and generally indicates end of data. The additional package terminator must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not received nor are they included in the total byte count.

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the host SCI Transmit block.

---

## Data type

Data type of the output data. Available options are single, int8, uint8, int16, uint16, int32, or uint32.

## Data length

How many of **Data type** the block will receive (not bytes). Anything more than 1 is a vector. The data length is inherited

## C281x SCI Receive

---

from the input (the data length originally input to the host-side SCI Transmit block).

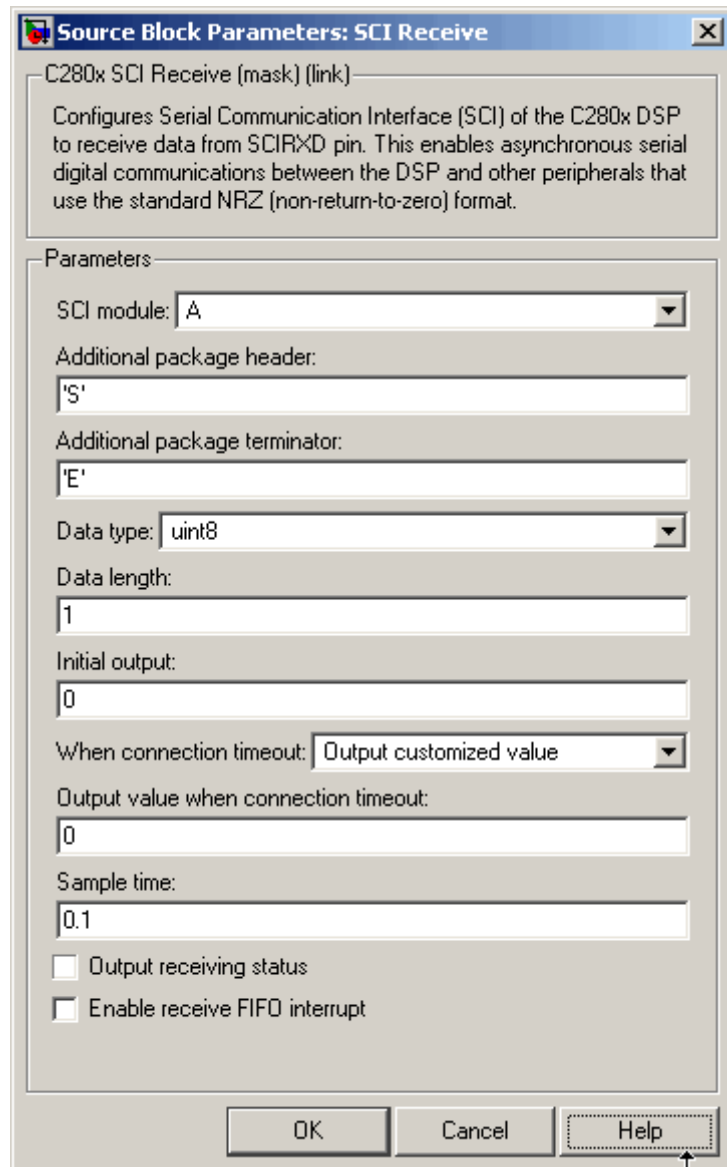
### **Initial output**

Default value from the c281x SCI Receive block. This value is used, for example, if a connection time-out occurs and the **When connection timeout** field is set to “Output the last received value”, but nothing yet has been received.

### **When connection timeout**

Specifies what to output if a connection time-out occurs. If “Output the last received value” is selected, the last received value is what is output, unless none has been received yet, in which case the **Initial output** is considered the last received value.

If “Output customized value” is selected, a field for specifying a custom value is added to the dialog box (as shown in the following figure).



# C281x SCI Receive

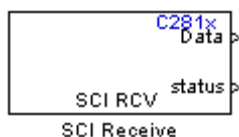
---

## Sample time

Sample time,  $T_s$ , for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

## Output receiving status

When this field is checked, the c281x SCI Receive block adds another output port for the transaction status, and appears as shown in the following figure.



Error status may be one of the following values:

- 0: No errors
- 1: A time-out occurred while the block was waiting to receive data
- 2: There is an in the received data (checksum error)
- 3: SCI parity-error flag — Occurs when a character is received with a mismatch between the number of 1s and its parity bit
- 4: SCI framing-error flag — Occurs when an expected stop bit is not found

## Enable receive FIFO interrupt

If this option is selected, an interrupt is posted when FIFO is full, allowing the subsystem to take some sort of action (for example, read data as soon as it is received). If this option is cleared, the block stays in polling mode. If the block is in polling mode and not blocking, it checks the FIFO to see if there is data to read. If data is present, it reads and outputs. If no data is present, it continues. If the block is in polling mode and blocking, it waits until data is available to read (when data length is reached).

### **Receive FIFO interrupt level**

This parameter is enabled when the **Enable receive FIFO interrupt** option is selected. Select an interrupt level from 0 to 16. The default level is 0.

### **References**

Detailed information on the SCI module is in *TMS320x281x, 280x DSP Serial Communication Interface (SCI) Reference Guide*, Literature Number SPRU051B, available at the Texas Instruments Web site.

### **See Also**

C281x SCI Transmit, C281x Hardware Interrupt

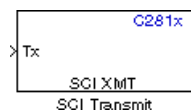
# C281x SCI Transmit

---

**Purpose** Transmit data from target via serial communications interface (SCI) to host

**Library** `c281xdspchip1lib` in Target for TI C2000

## Description



The C281x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C281x target's COM ports in nonreturn-to-zero (NRZ) format. You can specify how many of the six target COM ports to use. The sampling rate and data type are inherited from the input port. The data type of the input port must be one of the following: `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. If no data type is specified, the default data type is `uint8`.

---

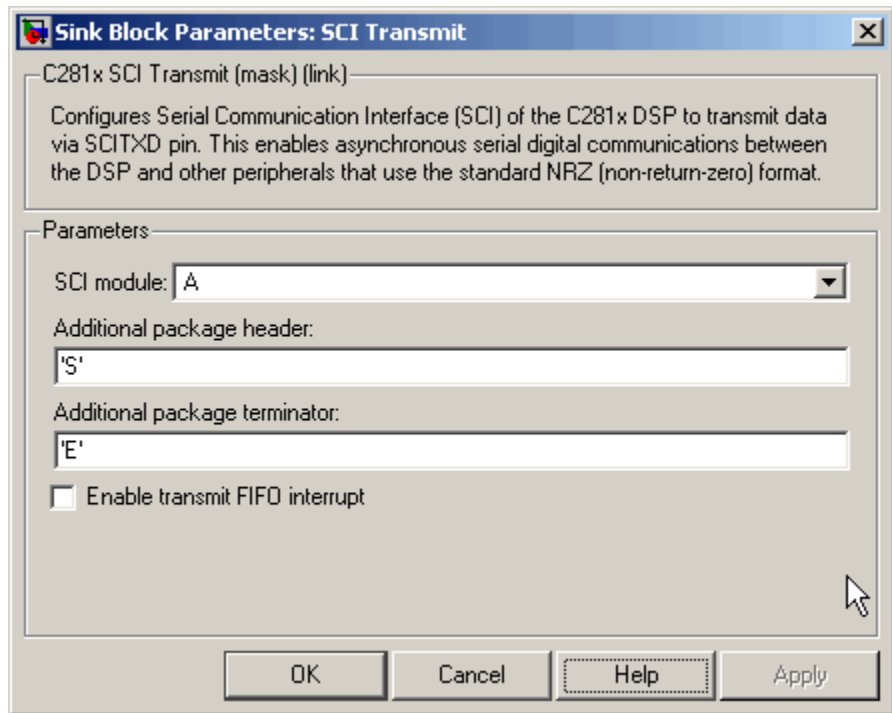
**Note** For any given model, you can have only one C281x SCI Transmit block per module. There are two modules, A and B, which can be configured through the F2812 eZdsp target preferences block.

Many SCI-specific settings are in the **DSPBoard** section of the F2812 eZdsp target preferences block. You should verify that these settings are correct for your application.

Fixed-point inputs are not supported for this block.

---

## Dialog Box



### SCI module

SCI module to be used for communications.

### Additional package header

This field specifies the data located at the front of the sent data package, which is not part of the data being transmitted, and generally indicates start of data. The additional package header must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not sent nor are they included in the total byte count.

# C281x SCI Transmit

---

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the host SCI Receive block.

---

## **Additional package terminator**

This field specifies the data located at the end of the sent data package, which is not part of the data being transmitted, and generally indicates end of data. The additional package terminator must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not sent nor are they included in the total byte count.

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the host SCI Receive block.

---

## **Enable transmit FIFO interrupt**

If this option is selected, an interrupt is posted when FIFO is full, allowing the subsystem to take some sort of action.

## **References**

Detailed information on the SCI module is in *TMS320x281x, 280x DSP Serial Communication Interface (SCI) Reference Guide*, Literature Number SPRU051B, available at the Texas Instruments Web site.

## **See Also**

C281x SCI Receive, C281x Hardware Interrupt



<b>Purpose</b>	Generate software triggered nonmaskable interrupt
<b>Library</b>	c281xdspchip1lib in Target for TI C2000
<b>Description</b>	<p>When you add this block to a model, the block polls the input port for the input value. When the input value is greater than the value in <b>Trigger software interrupt when input value is greater than</b>, the block posts the interrupt to a Hardware Interrupt block in the model.</p> <p>To use this block, add a Hardware Interrupt block to your model to process the software triggered interrupt from this block into an interrupt service routine on the processor. Set the interrupt number in the Hardware Interrupt block to the value you set here in <b>CPU interrupt number</b>.</p> <p>The CPU and PIE interrupt numbers together specify a single interrupt for a single peripheral or peripheral module. The following table maps CPU and PIE interrupt numbers to these peripheral interrupts. The row numbers are CPU values and the column numbers are the PIE values.</p>

---

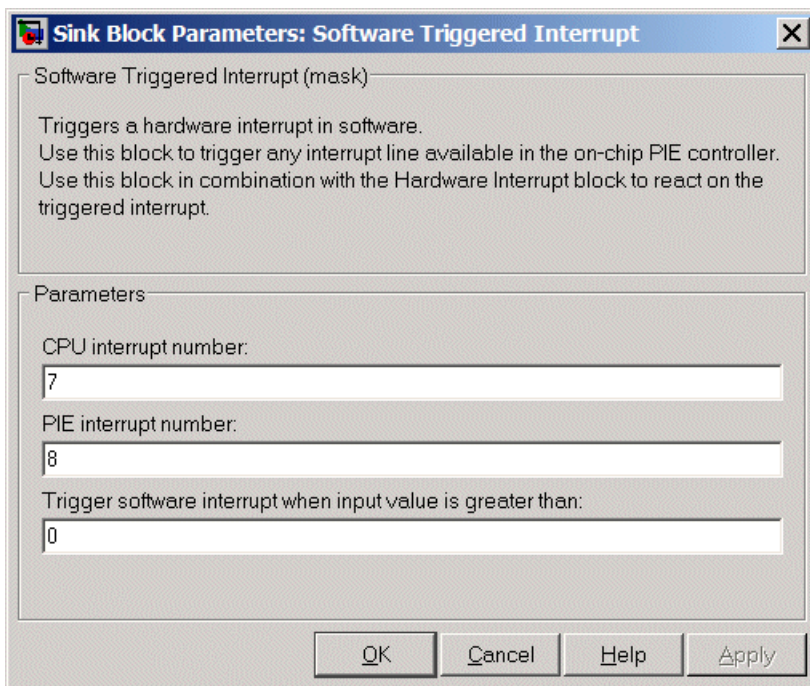
**Note** Fixed-point inputs are not supported for this block.

---

C281x Peripheral Interrupt Vector Values

	1	2	3	4	5	6	7	8
1	PDPINTA (EV-A)	PDPINTB (EV-B)	Reserved	XINT1	XINT2	ADCINT (ADC)	TINT0 (TIMER 0)	WAKEINT (LPM/WD)
2	CMP1INT (EV-A)	CMP2INT (EV-A)	CMP3INT (EV-A)	T1PINT (EV-A)	T1CINT (EV-A)	T1UFINT (EV-A)	T1OFINT (EV-A)	Reserved
3	T2PINT (EV-A)	T2CINT (EV-A)	T2UFINT (EV-A)	T2OFINT (EV-A)	CAPINT1 (EV-A)	CAPINT2 (EV-A)	CAPINT3 (EV-A)	Reserved
4	CMP4INT (EV-B)	CMP5INT (EV-B)	CMP6INT (EV-B)	T3PINT (EV-B)	T3CINT (EV-B)	T3UFINT (EV-B)	T3OFINT (EV-B)	Reserved
5	T4PINT (EV-B)	T4CINT (EV-B)	T4UFINT (EV-B)	T4OFINT (EV-B)	CAPINT4 (EV-B)	CAPINT5 (EV-B)	CAPINT6 (EV-B)	Reserved
6	SPIRXINTA (SPI)	SPITXINTA (SPI)	Reserved	Reserved	MRINT (McBSP)	MXINT (McBSP)	Reserved	Reserved
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
8	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
9	SCIRXINTA (SCI-A)	SCITXINTA (SCI-A)	SCIRXINTB (SCI-B)	SCITXINTB (SCI-B)	ECAN0INT (CAN)	ECAN1INT (CAN)	Reserved	Reserved
10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

## Dialog Box



### **CPU interrupt number**

Specify the interrupt the block responds to. Interrupt numbers are integers ranging from 1 to 12.

### **PIE interrupt number**

Enter an integer value from 1 to 8 to set the Peripheral Interrupt Expansion (PIE) interrupt number.

### **Trigger software interrupt when input value is greater than:**

Sets the value above which the block posts an interrupt. Enter the value to set the level that indicates that the interrupt is asserted by a requesting routine.

# C281x SW Int Trigger

---

## **References**

For detailed information about interrupt processing, refer to *TMS320x281x DSP System Control and Interrupts Reference Guide*, SPRU078C, available at the Texas Instruments Web site.

## **See Also**

C281x Hardware Interrupt

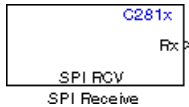
## Purpose

Receive data via serial peripheral interface on target

## Library

c281xdspchip1lib in Target for TI C2000

## Description



The C281x SPI Receive supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode.

In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both master and slave to send and receive data simultaneously. The maximum for the clock is one quarter of the DSP controller's clock frequency.

For any given model, you can have only one C281x SPI Receive block per module. There are two modules, A and B, which can be configured through the F2812 eZdsp target preferences block.

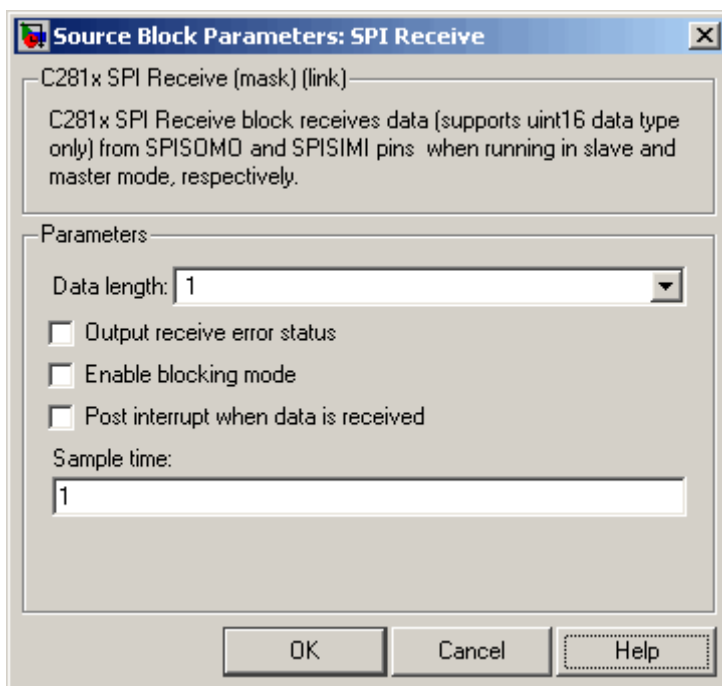
---

**Note** Many SPI-specific settings are in the **DSPBoard** section of the F2812 eZdsp target preferences block. You should verify that these settings are correct for your application.

---

# C281x SPI Receive

## Dialog Box



### Data length

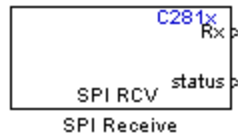
Specifies how many uint16s are expected to be received. Select 1 through 16.

### Enable blocking mode

If this option is selected, system waits until data is received before continuing processing.

### Output receive error status

When this field is checked, the c281x SPI Receive block adds another output port for the transaction status, and appears as shown in the following figure.



Error status may be one of the following values:

- 0: No errors
- 1: Data loss occurred (Overrun: when FIFO disabled, Overflow: when FIFO enabled)
- 2: Data not ready, a time-out occurred while the block was waiting to receive data

### **Post interrupt when data is received**

Check this check box to post an asynchronous interrupt when data is received.

### **Sample time**

Sample time,  $T_s$ , for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt when message is received** box, and refer to "Asynchronous Interrupt Processing" on page 1-14 for a discussion of block placement and other necessary settings.

## **See Also**

C281x SPI Transmit, C281x Hardware Interrupt

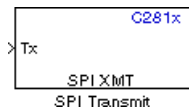
# C281x SPI Transmit

---

**Purpose** Transmit data via serial peripheral interface (SPI) to host

**Library** c281xdspchip1lib in Target for TI C2000

## Description



The C281x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both master and slave to send and receive data simultaneously. The maximum for the clock is one quarter of the DSP controller's clock frequency.

The sampling rate is inherited from the input port. The supported data type is `uint16`.

---

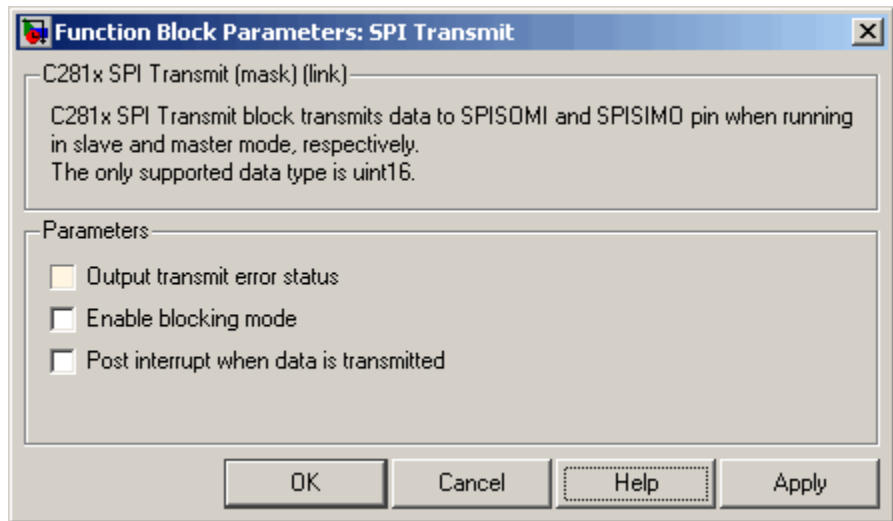
**Note** For any given model, you can have only one C281x SPI Transmit block per module. There are two modules, A and B, which can be configured through the F2812 eZdsp target preferences block.

Many SPI-specific settings are in the **DSPBoard** section of the F2812 eZdsp target preferences block. You should verify that these settings are correct for your application.

---

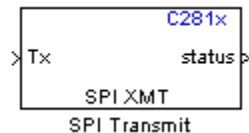


## Dialog Box



### Output transmit error status

When this field is checked, the c281x SPI Transmit block adds another output port for the transaction status, and appears as shown in the following figure.



Error status may be one of the following values:

- 0: No errors
- 1: A time-out occurred while the block was transmitting data
- 2: There is an error in the transmitted data (for example, header or terminator don't match, length of data expected is too big or too small)

# C281x SPI Transmit

---

## **Enable blocking mode**

If this option is selected, system waits until data is sent before continuing processing.

## **Post interrupt when data is transmitted**

Select this check box to post an asynchronous interrupt when data is transmitted.

## **See Also**

C281x SPI Receive

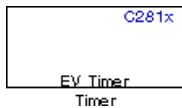
## Purpose

Configure up to four general-purpose, stand alone Event Manager timers

## Library

c281xdspchip1lib in Target for TI C2000

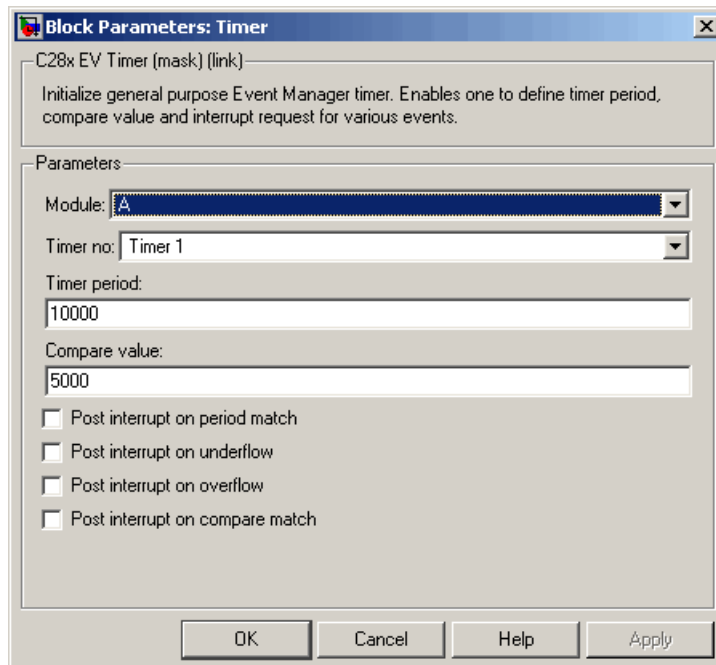
## Description



The C281x event-manager (EV) modules include general-purpose (GP) timers. There are two general-purpose (GP) timers in each module. These timers can be used as independent time bases in various applications.

The C281x Timer block lets you set the periodicity of the general-purpose timers, and configure them to post interrupts under specified conditions.

## Dialog Box



## Module

### Timer no

Select which of four possible timers to configure. Setting **Module** to A lets you select Timer 1 or Timer 2 in **Timer no**. Setting **Module** to B lets you select Timer 3 or Timer 4 in **Timer no**.

### Timer period

Set the length of the timer period in clock cycles. Enter a value from 0 to 65535. The default is 10000.

You can easily calculate how many clock cycles to set for the timer period if you know the length of a clock cycle. The calculation for the length of one clock cycle is as follows:

$$\text{Sysclk}(150\text{MHz}) \rightarrow \text{HISPCLK}(1/2) \rightarrow \text{InputClockPrescaler}(1/128)$$

where the System clock frequency of 150MHz is divided by the high speed clock prescaler of 2, and then divided by the timer control input clock prescaler, which is 128. The resulting frequency is .586MHz. Thus, one clock cycle is 1/.586MHz, which is 1.706µs.

### Compare value

Enter a constant value to be used for comparison to the running timer value for the purpose of generating interrupts. Enter a value from 0 to 65535. The default is 5000. Note that interrupts will be generated only if **Post interrupt on compare match** is selected.

### Post interrupt on period match

Select this check box to generate an interrupt whenever the value of the timer reaches its maximum value as specified in **Timer period**.

### Post interrupt on underflow

Select this check box to generate an interrupt whenever the value of the timer cycles back to 0.

### Post interrupt on overflow

Select this check box to generate an interrupt whenever the value of the timer reaches its maximum possible value of 65535. Note

that unless **Timer period** is set to 65535, this interrupt will never be generated even if this check box is selected.

**Post interrupt on compare match**

Select this check box to generate an interrupt whenever the value of the timer equals **Compare value**.

**See Also**

C281x Hardware Interrupt, Idle Task

# Clarke Transformation

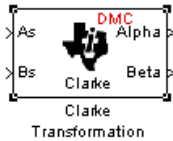
## Purpose

Convert balanced three-phase quantities to balanced two-phase quadrature quantities

## Library

c28xdmclib in Target for TI C2000

## Description

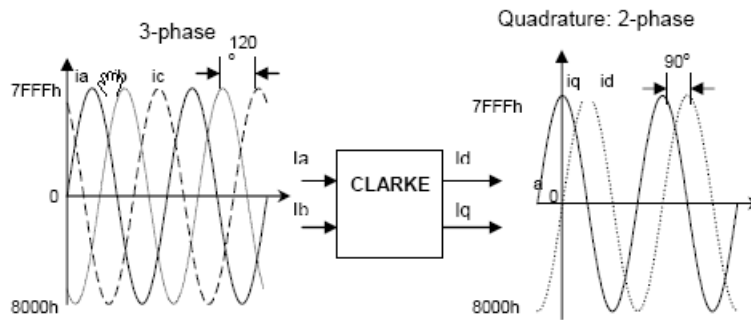


This block converts balanced three-phase quantities into balanced two-phase quadrature quantities. The transformation implements these equations

$$I_d = I_a$$

$$I_q = (2I_b + I_a) / \sqrt{3}$$

and is illustrated in the following figure.



The inputs to this block are the phase a (As) and phase b (Bs) components of the balanced three-phase quantities and the outputs are the direct axis (Alpha) component and the quadrature axis (Beta) of the transformed signal.

The instantaneous outputs are defined by the following equations and are shown in the following figure:

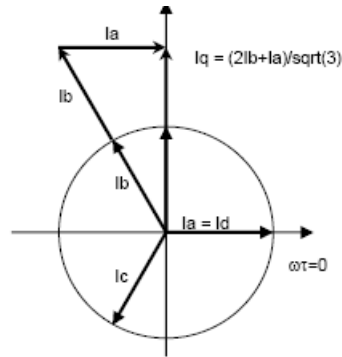
$$i_a = I * \sin(\omega t)$$

$$i_b = I * \sin(\omega t + 2\pi/3)$$

$$i_c = I * \sin(\omega t - 2\pi/3)$$

$$i_d = I * \sin(\omega t)$$

$$i_q = I * \sin(\omega t + \pi/2)$$



The variables used in the preceding equations and figures correspond to the variables on the block as shown in the following table:

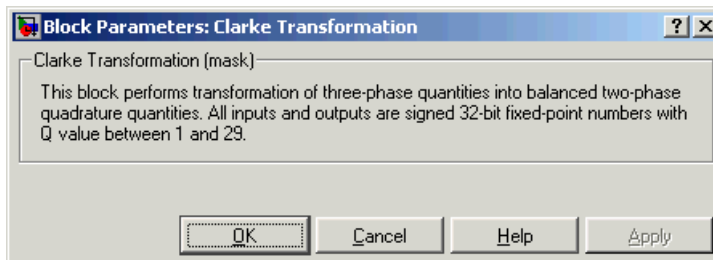
	Equation Variables	Block Variables
Inputs	ia	As
	ib	Bs
Outputs	id	Alpha
	iq	Beta

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

# Clarke Transformation

---

## Dialog Box



## References

Detailed information on the DMC library is in *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

## See Also

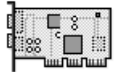
Inverse Park Transformation, Park Transformation, PID Controller, Space Vector Generator, Speed Measurement



**Purpose** Target preferences for custom C28xx board

**Library** c2000tgtpreflib in Target for TI C2000

## Description



Custom Board

Options on the block dialog box let you set features of code generation for your custom C280x or C281x processor-based target. Adding this block to your Simulink model provides access to the processor hardware settings you need to configure when you generate a project from a Simulink model or you generate code from Real-Time Workshop to run on your board.

Any model that you use to generate a project or that you target to custom hardware should include this block or the Target Preferences block from Link for Code Composer Studio Development Tools. Simulink or Real-Time Workshop return an error message if a target preferences block is not present in your model when you try to generate projects or code.

---

**Note** This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model. Simulink returns an error when your model does not include a target preferences block or has more than one.

---

The processor and target options you specify on this block are:

- Processor and board information
- Memory mapping and layout
- Allocation of the various code sections, such as compiler and custom sections
- Operating parameters for peripherals on C280x processors

Setting the options included in this dialog box results in identifying your target to Real-Time Workshop, Target for TI C2000, and Simulink, and

# Custom Board

---

configuring the memory map for your target. Both steps are essential for targeting any C28xx-based board that is custom or explicitly supported.

Unlike most other blocks, you cannot open the block dialog box for this block until you add the block to a model. When you try to open the block dialog, the block attempts to connect to your target. It cannot make the connection when the block is in the library and returns an error message. Also, if you do not have Code Composer Studio installed, you cannot open this block.

For details about the options for the Custom C28xx Board target preferences, refer to the Target Preferences block in Link for Code Composer Studio Development Tools.

## **Generating Code from Model Subsystems**

Real-Time Workshop provides the ability to generate code from a selected subsystem in a model. To generate code for a C28xx processor-based target from a subsystem, the subsystem model must include a Target Preferences block.

### **See Also**

C280x ADC, C280x eCAN Receive, C280x eCAN Transmit, C280x ePWM, C281x ADC, C281x eCAN Receive, C281x eCAN Transmit, C281x PWM

**Purpose** Divide IQ numbers

**Library** `tiiqmathlib` in Target for TI C2000

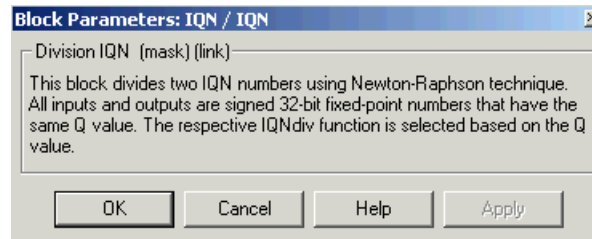
**Description** This block divides two numbers that use the same Q format, using the Newton-Raphson technique. The resulting quotient uses the same Q format at the inputs.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



**See Also** Absolute IQN, Arctangent IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

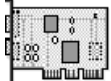
# F2808 eZdsp

---

**Purpose** F2808 eZdsp DSK target preferences

**Library** c2000tgtpref1lib in Target for TI C2000

## Description



F2808 eZdsp

Options on the block dialog box let you set features of code generation for your F2808 eZdsp target. Adding this block to your Simulink model provides access to the processor hardware settings you need to configure when you generate a project from a Simulink model or you generate code from Real-Time Workshop to run on your board.

Any model that you use to generate a project or that you target to F2808 eZdsp hardware should include this block or the Target Preferences block from Link for Code Composer Studio Development Tools. Simulink or Real-Time Workshop return an error message if a target preferences block is not present in your model when you try to generate projects or code.

---

**Note** This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model. Simulink returns an error when your model does not include a target preferences block or has more than one.

---

The processor and target options you specify on this block are:

- Processor and board information
- Memory mapping and layout
- Allocation of the various code sections, such as compiler and custom sections
- Operating parameters for peripherals on F2808 eZdsp hardware

Setting the options included in this dialog box results in identifying your target to Real-Time Workshop, Target for TI C2000, and Simulink,

and configuring the memory map for your target. Both steps are essential for targeting any F2808 eZdsp.

Unlike most other blocks, you cannot open the block dialog box for this block until you add the block to a model. When you try to open the block dialog, the block attempts to connect to your target. It cannot make the connection when the block is in the library and returns an error message. Also, if you do not have Code Composer Studio installed, you cannot open this block.

For details about the options for the F2808 eZdsp target preferences, refer to the Target Preferences block in Link for Code Composer Studio Development Tools.

## **Generating Code from Model Subsystems**

Real-Time Workshop provides the ability to generate code from a selected subsystem in a model. To generate code for a F2808 eZdsp from a subsystem, the subsystem model must include a Target Preferences block.

## **See Also**

C280x ADC, C280x eCAN Receive, C280x eCAN Transmit, C280x ePWM, C280x eQEP, C280x Hardware Interrupt, Idle Task

# F2808 eZdsp Stand alone code using Flash Memory

---

**Library** c2000tgtppreflib in Target for TI C2000

**Description** This block saves the generated code to nonvolatile flash memory for reuse. Saving the code in Flash, directly on the chip, allows the chip to be unplugged and reused at a different time. Options on the block dialog box let you set features of code generation for your F2808 eZdsp target. Adding this block to your Simulink model provides access to the processor hardware settings you need to configure when you generate a project from a Simulink model or you generate code from Real-Time Workshop to run on your board.

Any model that you use to generate a project or that you target to F2808 eZdsp hardware should include this block or the Target Preferences block from Link for Code Composer Studio Development Tools. Simulink or Real-Time Workshop return an error message if a target preferences block is not present in your model when you try to generate projects or code.

---

**Note** This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model. Simulink returns an error when your model does not include a target preferences block or has more than one.

---

The processor and target options you specify on this block are:

- Processor and board information
- Memory mapping and layout
- Allocation of the various code sections, such as compiler and custom sections
- Operating parameters for peripherals on F2808 eZdsp hardware

Setting the options included in this dialog box results in identifying your target to Real-Time Workshop, Target for TI C2000, and Simulink,

# F2808 eZdsp Stand alone code using Flash Memory

---

and configuring the memory map for your target. Both steps are essential for targeting any F2808 eZdsp.

Unlike most other blocks, you cannot open the block dialog box for this block until you add the block to a model. When you try to open the block dialog, the block attempts to connect to your target. It cannot make the connection when the block is in the library and returns an error message. Also, if you do not have Code Composer Studio installed, you cannot open this block.

For details about the options for the F2808 eZdsp target preferences, refer to the Target Preferences block in Link for Code Composer Studio Development Tools.

## Generating Code from Model Subsystems

Real-Time Workshop provides the ability to generate code from a selected subsystem in a model. To generate code for a F2808 eZdsp from a subsystem, the subsystem model must include a Target Preferences block.

## See Also

C280x ADC, C280x eCAN Receive, C280x eCAN Transmit, C280x ePWM, C280x eQEP, C280x Hardware Interrupt, Idle Task

# F2812 eZdsp

---

<b>Purpose</b>	F2812 eZdsp DSK target preferences
<b>Library</b>	c2000tgtpref1lib in Target for TI C2000
<b>Description</b>	<p>Options on the block dialog box let you set features of code generation for your F2812 eZdsp target. Adding this block to your Simulink model provides access to the processor hardware settings you need to configure when you generate a project from a Simulink model or you generate code from Real-Time Workshop to run on your board.</p> <p>Any model that you use to generate a project or that you target to F2812 eZdsp hardware should include this block or the Target Preferences block from Link for Code Composer Studio Development Tools. Simulink or Real-Time Workshop return an error message if a target preferences block is not present in your model when you try to generate projects or code.</p>

---

**Note** This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model. Simulink returns an error when your model does not include a target preferences block or has more than one.

---

The processor and target options you specify on this block are:

- Processor and board information
- Memory mapping and layout
- Allocation of the various code sections, such as compiler and custom sections
- Operating parameters for peripherals on F2812 eZdsp hardware

Setting the options included in this dialog box results in identifying your target to Real-Time Workshop, Target for TI C2000, and Simulink,



and configuring the memory map for your target. Both steps are essential for targeting any F2812 eZdsp.

Unlike most other blocks, you cannot open the block dialog box for this block until you add the block to a model. When you try to open the block dialog, the block attempts to connect to your target. It cannot make the connection when the block is in the library and returns an error message. Also, if you do not have Code Composer Studio installed, you cannot open this block.

For details about the options for the F2812 eZdsp target preferences, refer to the Target Preferences block in Link for Code Composer Studio Development Tools.

## **Generating Code from Model Subsystems**

Real-Time Workshop provides the ability to generate code from a selected subsystem in a model. To generate code for a F2812 eZdsp from a subsystem, the subsystem model must include a Target Preferences block.

For details about the options for the F2812 eZdsp target preferences, refer to the Target Preferences block in Link for Code Composer Studio Development Tools.

## **See Also**

C281x ADC, C281x eCAN Receive, C281x eCAN Transmit, C281x PWM

# F2812 eZdsp Stand alone code using Flash Memory

---

**Library** c2000tgtpreflib in Target for TI C2000

**Description** This block saves the generated code to nonvolatile flash memory for reuse. Saving the code in Flash, directly on the chip, allows the chip to be unplugged and reused at a different time. Options on the block dialog box let you set features of code generation for your F2812 eZdsp target. Adding this block to your Simulink model provides access to the processor hardware settings you need to configure when you generate a project from a Simulink model or you generate code from Real-Time Workshop to run on your board.

Any model that you use to generate a project or that you target to F2812 eZdsp hardware should include this block or the Target Preferences block from Link for Code Composer Studio Development Tools. Simulink or Real-Time Workshop return an error message if a target preferences block is not present in your model when you try to generate projects or code.

---

**Note** This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model. Simulink returns an error when your model does not include a target preferences block or has more than one.

---

The processor and target options you specify on this block are:

- Processor and board information
- Memory mapping and layout
- Allocation of the various code sections, such as compiler and custom sections
- Operating parameters for peripherals on F2812 eZdsp hardware

Setting the options included in this dialog box results in identifying your target to Real-Time Workshop, Target for TI C2000, and Simulink,

# F2812 eZdsp Stand alone code using Flash Memory

---

and configuring the memory map for your target. Both steps are essential for targeting any F2812 eZdsp.

Unlike most other blocks, you cannot open the block dialog box for this block until you add the block to a model. When you try to open the block dialog, the block attempts to connect to your target. It cannot make the connection when the block is in the library and returns an error message. Also, if you do not have Code Composer Studio installed, you cannot open this block.

For details about the options for the F2812 eZdsp target preferences, refer to the Target Preferences block in Link for Code Composer Studio Development Tools.

## **Generating Code from Model Subsystems**

Real-Time Workshop provides the ability to generate code from a selected subsystem in a model. To generate code for a F2812 eZdsp from a subsystem, the subsystem model must include a Target Preferences block.

For details about the options for the F2812 eZdsp target preferences, refer to the Target Preferences block in Link for Code Composer Studio Development Tools.

## **See Also**

C281x ADC, C281x eCAN Receive, C281x eCAN Transmit, C281x PWM

# Float to IQN

---

**Purpose** Convert floating-point number to IQ number

**Library** `tiiqmathlib` in Target for TI C2000

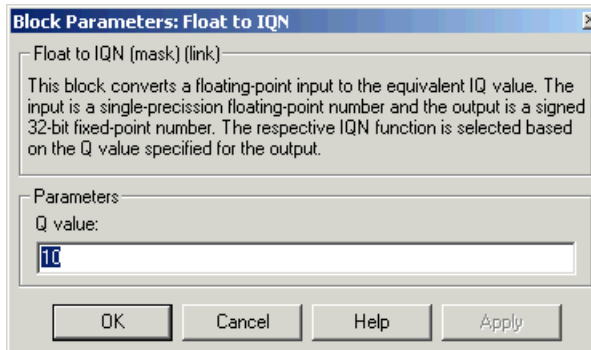
**Description** This block converts a floating-point number to an IQ number. The Q value of the output is specified in the dialog.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



### Q value

Q value from 1 to 30 that specifies the precision of the output

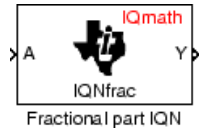
### See Also

Absolute IQN, Arctangent IQN, Division IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

**Purpose** Fractional part of IQ number

**Library** tiiqmathlib in Target for TI C2000

**Description** This block returns the fractional portion of an IQ number. The returned value is an IQ number in the same IQ format.

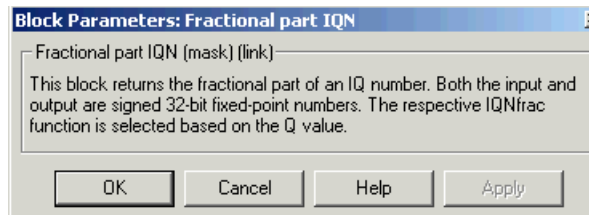


---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



**See Also** Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

# Fractional part IQN x int32

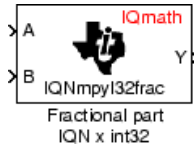
## Purpose

Fractional part of result of multiplying IQ number and long integer

## Library

tiiqmathlib in Target for TI C2000

## Description



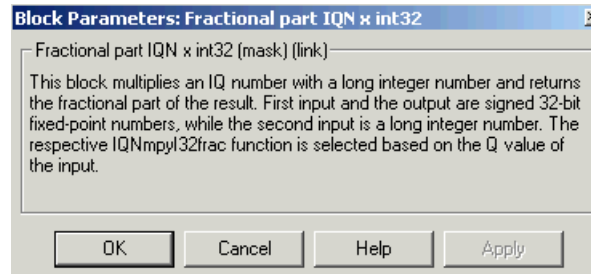
This block multiplies an IQ input and a long integer input and returns the fractional portion of the resulting IQ number.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



## See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

## Purpose

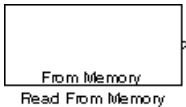
Retrieve data from target memory

## Library

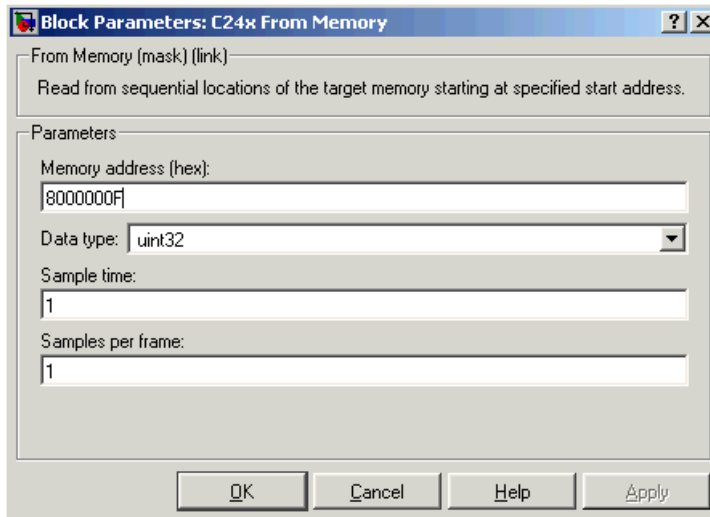
c280xspchiplib or c281xspchiplib in Target for TI C2000

## Description

This block retrieves data of the specified data type from a particular memory address on the target.



## Dialog Box



## Memory address

Address of the target memory location, in hexadecimal, from which to read data.

---

**Note** To ensure the correct operation of this block, you must specify exactly the desired memory location. Refer to your Linker CMD file for available memory locations.

---

# From Memory

---

**Data type**

Data type of the data to obtain from the above memory address. The data is read as 16-bit data and then cast to the selected data type. Valid data types are double, single, int8, uint8, int16, uint16, int32, and uint32.

**Sample time**

Time interval, in seconds, between consecutive reads from the specified memory location.

**Samples per frame**

Number of elements of the specified data type to be read from the memory region starting at the given address.

**See Also**

To Memory

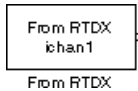


**Purpose**

Add RTDX input channel

**Library**

rtDXBlocks in Target for TI C2000

**Description**

When you generate code from Simulink in Real-Time Workshop with a From RTDX block in your model, code generation inserts the C commands to create an RTDX input channel on the target. Input channels transfer data from the host to the target.

The generated code contains this command:

```
RTDX_enableInput(&channelname)
```

where channelname is the name you enter in **Channel name**.

---

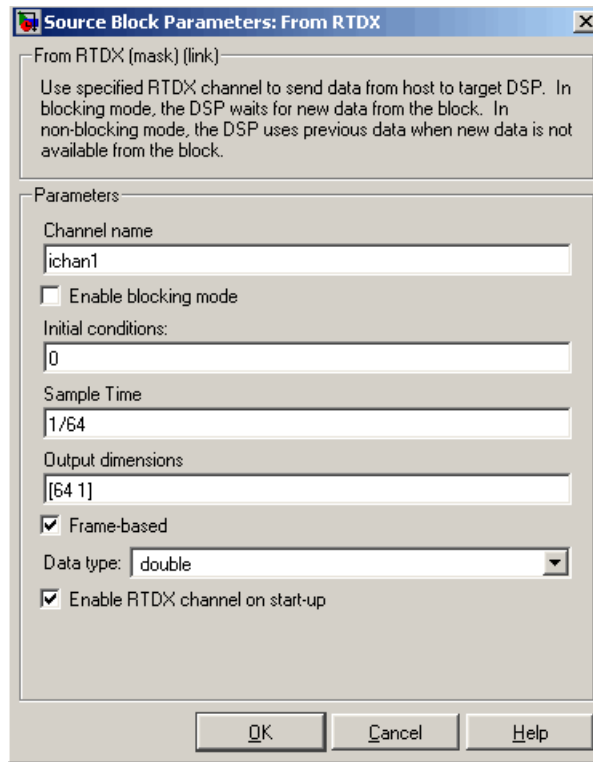
**Note** From RTDX blocks work only in code generation and when your model runs on your target. In simulations, this block does not perform any operations, except generating an output matching your specified initial conditions.

---

To use RTDX blocks in your model, you must do the following:

- 1** Add one or more To RTDX or From RTDX blocks to your model.
- 2** Download and run your model on your target.
- 3** Enable the RTDX channels from MATLAB or use **Enable RTDX channel on start-up** on the block dialog.
- 4** Use the readmsg and writemsg functions in MATLAB to send and retrieve data from the target over RTDX.

## Dialog Box



### Channel name

Name of the input channel to be created by the generated code. The channel name must meet C syntax requirements for length and character content.

### Enable blocking mode

Blocking mode instructs the target processor to pause processing until new data is available from the From RTDX block. If you enable blocking and new data is not available when the processor needs it, your process stops. In nonblocking mode, the processor uses old data from the block when new data is not available.

Nonblocking operation is the default and is recommended for most operations.

### **Initial conditions**

Data the processor reads from RTDX for the first read. If blocking mode is not enabled, you must have an entry for this option. Leaving the option blank causes an error in Real-Time Workshop. Valid values are 0, null ([ ]), or a scalar. The default value is 0.

0 or null ([ ]) outputs a zero to the processor. A scalar generates one output sample with the value of the scalar. If **Output dimensions** specifies an array, every element in the array has the same scalar or zero value. A null array ([ ]) outputs a zero for every sample.

### **Sample time**

Time between samples of the signal. The default is 1 second. This produces a sample rate of one sample per second (1/**Sample time**).

### **Output dimensions**

Dimensions of a matrix for the output signal from the block. The first value is the number of rows and the second is the number of columns. For example, the default setting [ 1 64 ] represents a 1-by-64 matrix of output values. Enter a 1-by-2 vector for the dimensions.

### **Frame-based**

Sets a flag at the block output that directs downstream blocks to use frame-based processing on the data from this block. In frame-based processing, the samples in a frame are processed simultaneously. In sample-based processing, samples are processed one at a time. Frame-based processing can increase the speed of your application running on your target. Note that throughput remains the same in samples per second processed. Frame-based operation is the default.

### **Data type**

Type of data coming from the block. Select one of the following types:

- **Double** — Double-precision floating-point values. This is the default. Values range from -1 to 1.
- **Single** — Single-precision floating-point values ranging from -1 to 1.
- **Uint8** — 8-bit unsigned integers. Output values range from 0 to 255.
- **Int16** — 16-bit signed integers. With the sign, the values range from -32768 to 32767.
- **Int32** — 32-bit signed integers. Values range from  $-2^{31}$  to  $(2^{31} - 1)$ .

### **Enable RTDX channel on start-up**

Enables the RTDX channel when you start the channel from MATLAB. With this selected, you do not need to use the enable function in Link for Code Composer Studio Development Tools to prepare your RTDX channels. This option applies only to the channel you specify in **Channel name**. You do have to open the channel.

### **See Also**

ccsdsp, readmsg, To RTDX, writemsg.

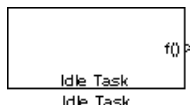
## Purpose

Free-running task that executes downstream subsystem

## Library

c280xspchiplib or c281xspchiplib in Target for TI C2000

## Description



The Idle Task block, and the subsystem to which it is connected, specify one or more functions to execute as background tasks. By definition, all tasks executed through the Idle Task block are of the lowest priority, lower than that of the base rate task.

## Vectorized Output

The output of this block includes a set of two vectors, the **Number of tasks** and the corresponding **Preemption flag(s)**. The **Preemption flag(s)** vector must be the same length as the **Number of tasks** vector unless it has only one element.

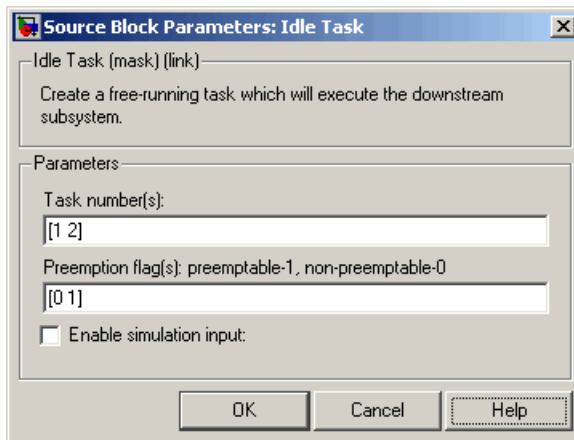
If the **Preemption flag(s)** vector does have one element, then that value applies to all functions in the downstream subsystem.

If the **Preemption flag(s)** vector has the same number of elements as the **Number of tasks** vector, then each task's preemption flag value is the value of the corresponding element in the **Preemption flag(s)** vector.

The preemption flag determines whether a given interrupt is preemptable or not. Preemption overrides prioritization, such that a preemptable task of higher priority can be preempted by a non-preemptable task of lower priority.

# Idle Task

## Dialog Box



### Number of tasks

The values you enter determine the order in which the functions in the downstream subsystem are to be executed, while the number of values you enter corresponds to the number of functions in the downstream subsystem.

Enter a vector containing the same number of elements as the number of functions in the downstream subsystem. This vector can contain no more than 16 elements, and the values must be from 0 to 15 inclusive.

The value of the first element in the vector determines the order in which the first function in the subsystem will be executed, and so on.

For example, if you enter `[2, 3, 1]` in this field, you are indicating that there are three functions to be executed, and that the third function will be executed first, the first function will be executed second, and the second function will be executed third.

When all functions have been executed, the Idle Task block cycles back and repeats the execution of the functions in the same order.

## **Preemption flag(s)**

The preemption flag determines whether a given interrupt is preemptable or not. Preemption overrides prioritization, so if you flag one of these functions as non-preemptable, its execution will not be suspended by another task even though the functions in the downstream subsystem all have the lowest priority by definition.

Enter either a vector of one element, in which case that preemption flag applies to all functions to be executed in the downstream subsystem, or a vector containing the same number of elements as the **Number of tasks** vector, in which case each preemption flag value applies to the task number in the corresponding position within its vector. All preemption flag values must be either 0 (non-preemptable) or 1 (preemptable).

## **Enable simulation input**

Select this check box to make it possible to test asynchronous interrupt processing in the context of your Simulink model.

---

**Note** Using this check box is the only way you can test asynchronous interrupt processing behavior in Simulink.

---

## **See Also**

C280x Hardware Interrupt, C281x Hardware Interrupt

# Integer part IQN

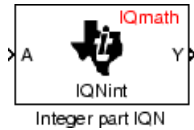
## Purpose

Integer part of IQ number

## Library

tiiqmathlib in Target for TI C2000

## Description



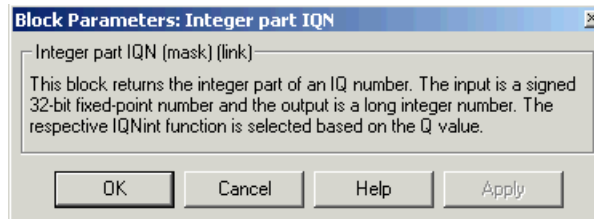
This block returns the integer portion of an IQ number. The returned value is a long integer.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



## See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN



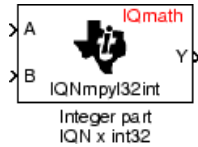
## Purpose

Integer part of result of multiplying IQ number and long integer

## Library

tiiqmathlib in Target for TI C2000

## Description



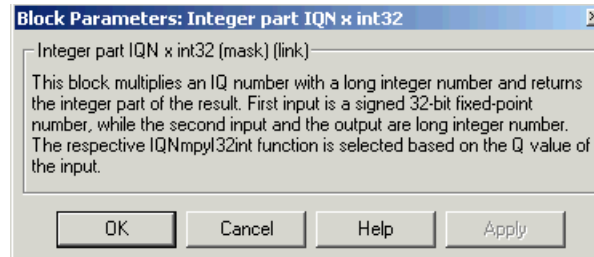
This block multiplies an IQ input and a long integer input and returns the integer portion of the resulting IQ number as a long integer.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



## See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

# Inverse Park Transformation

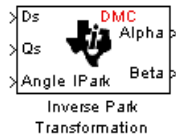
## Purpose

Convert rotating reference frame vectors to two-phase stationary reference frame

## Library

c28xdmclib in Target for TI C2000

## Description

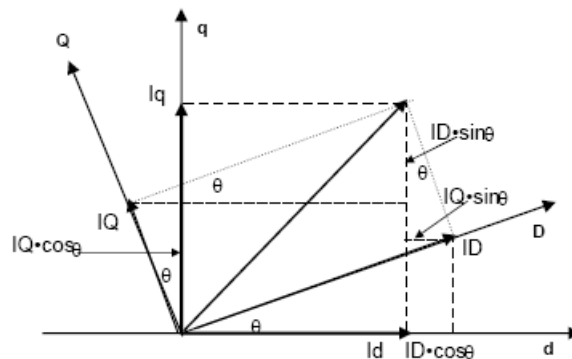


This block converts vectors in an orthogonal rotating reference frame to a two-phase orthogonal stationary reference frame. The transformation implements these equations:

$$I_d = I_D * \cos \theta - I_Q * \sin \theta$$

$$I_q = I_D * \sin \theta + I_Q * \cos \theta$$

and is illustrated in the following figure.



The inputs to this block are the direct axis ( $D_s$ ) and quadrature axis ( $Q_s$ ) components of the transformed signal in the rotating frame and the phase angle ( $Angle$ ) between the stationary and rotating frames.

The outputs are the direct axis ( $Alpha$ ) and the quadrature axis ( $Beta$ ) components of the transformed signal.

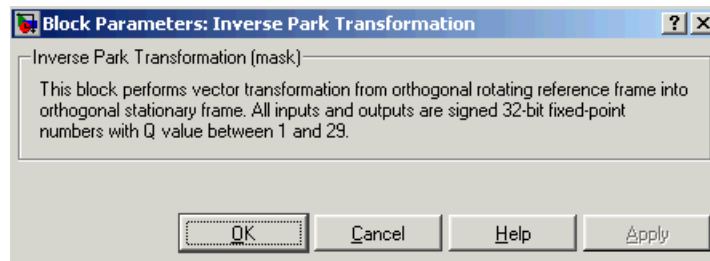
The variables used in the preceding figure and equations correspond to the block variables as shown in the following table:

# Inverse Park Transformation

	Equation Variables	Block Variables
Inputs	ID	Ds
	IQ	Qs
	$\theta$	Angle
Outputs	id	Alpha
	iq	Beta

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

## Dialog Box



## References

Detailed information on the DMC library is in *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

## See Also

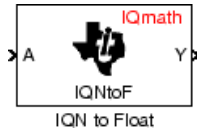
Clarke Transformation, Park Transformation, PID Controller, Space Vector Generator, Speed Measurement

# IQN to Float

**Purpose** Convert IQ number to floating-point number

**Library** `tiiqmathlib` in Target for TI C2000

**Description** This block converts an IQ input to an equivalent floating-point number. The output is a single floating-point number.

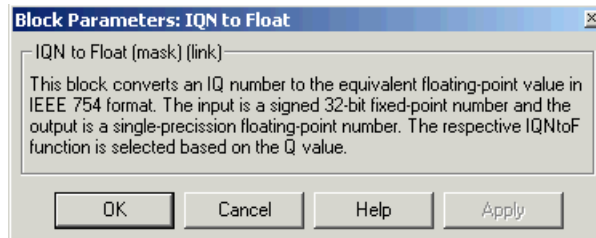


---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box

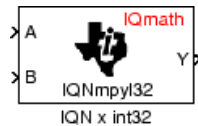


**See Also** Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

**Purpose** Multiply IQ number with long integer

**Library** `tiiqmathlib` in Target for TI C2000

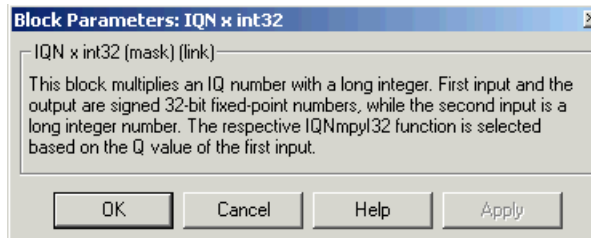
**Description**



This block multiplies an IQ input and a long integer input and produces an IQ output of the same Q value as the IQ input.

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

**Dialog Box**



**See Also**

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

# IQN x IQN

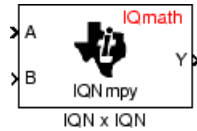
## Purpose

Multiply IQ numbers with same Q format

## Library

tiiqmathlib in Target for TI C2000

## Description



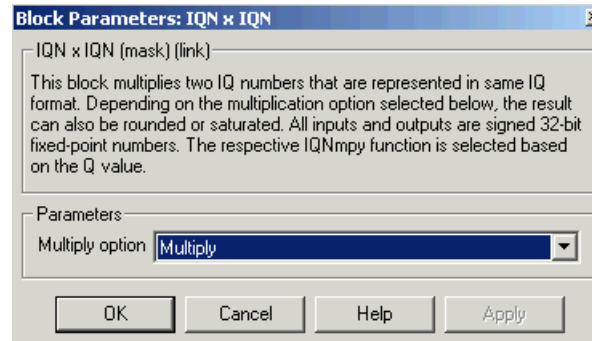
This block multiplies two IQ numbers. Optionally, it can also round and saturate the result.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



### Multiply option

Type of multiplication to perform:

- Multiply — Multiply the numbers.
- Multiply with Rounding — Multiply the numbers and round the result.
- Multiply with Rounding and Saturation — Multiply the numbers and round and saturate the result to the maximum value.

## **See Also**

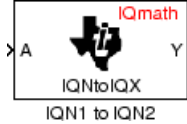
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

# IQN1 to IQN2

**Purpose** Convert IQ number to different Q format

**Library** `tiiqmathlib` in Target for TI C2000

**Description** This block converts an IQ number in a particular Q format to a different Q format.

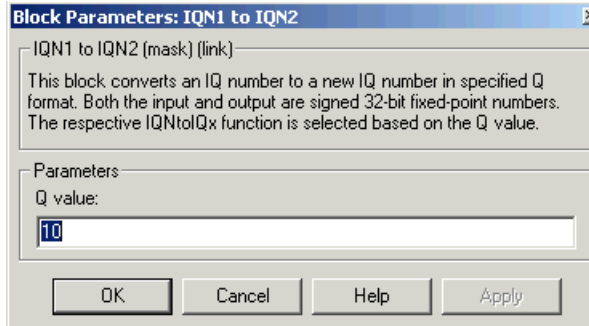


---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



### Q value

Q value from 1 to 30 that specifies the precision of the output

### See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN



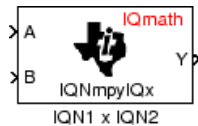
## Purpose

Multiply IQ numbers with different Q formats

## Library

tiiqmathlib in Target for TI C2000

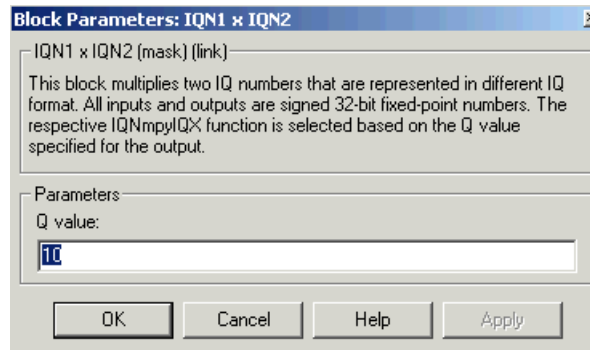
## Description



This block multiplies two IQ numbers when the numbers are represented in different Q formats. The format of the result is specified in the dialog box.

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

## Dialog Box



### Q value

Q value from 1 to 30 that specifies the precision of the output

## See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

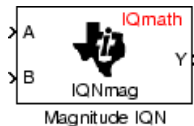
# Magnitude IQN

---

**Purpose** Magnitude of two orthogonal IQ numbers

**Library** `tiiqmathlib` in Target for TI C2000

**Description** This block calculates the magnitude of two IQ numbers using



$$\sqrt{a^2 + b^2}$$

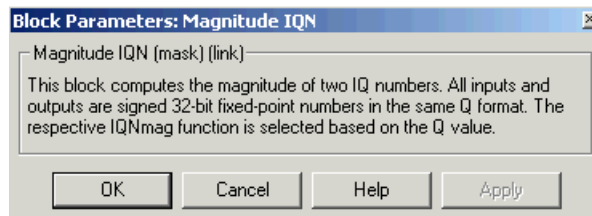
The output is an IQ number in the same Q format as the input.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



**See Also** Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Saturate IQN, Square Root IQN, Trig Fcn IQN

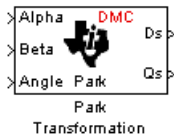
## Purpose

Convert two-phase stationary system vectors to rotating system vectors

## Library

c28xdmclib in Target for TI C2000

## Description

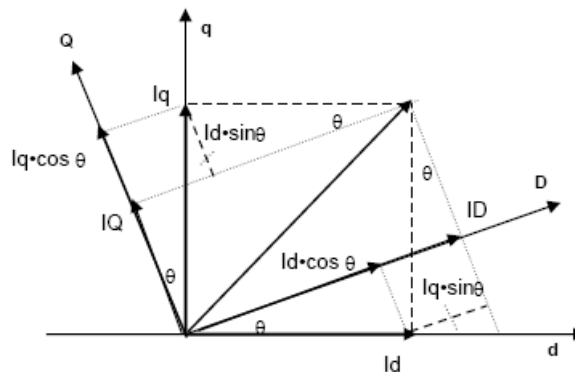


This block converts vectors in balanced two-phase orthogonal stationary systems into an orthogonal rotating reference frame. The transformation implements these equations

$$ID = Id * \cos \theta + Iq * \sin \theta$$

$$IQ = -Id * \sin \theta + Iq * \cos \theta$$

and is illustrated in the following figure.



The variables used in the preceding figure and equations correspond to the block variables as shown in the following table:

	Equation Variables	Block Variables
Inputs	id	Alpha
	iq	Beta
	$\theta$	Angle

# Park Transformation

	Equation Variables	Block Variables
Outputs	ID	Ds
	IQ	Qs

The inputs to this block are the direct axis (Alpha) and the quadrature axis (Beta) components of the transformed signal and the phase angle (Angle) between the stationary and rotating frames.

The outputs are the direct axis (Ds) and quadrature axis (Qs) components of the transformed signal in the rotating frame.

The instantaneous inputs are defined by the following equations:

$$id = I * \sin(\omega t)$$

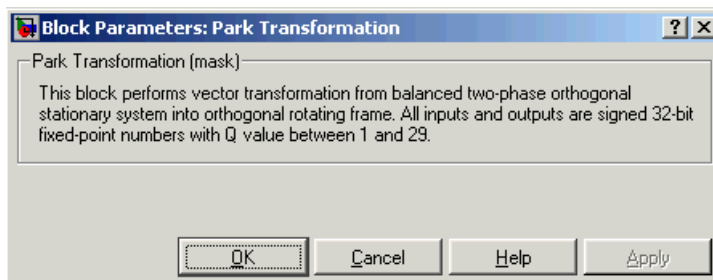
$$iq = I * \sin(\omega t + \pi / 2)$$

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



## References

Detailed information on the DMC library is in *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

## See Also

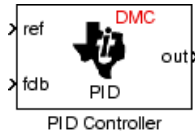
Clarke Transformation, Inverse Park Transformation, PID Controller, Space Vector Generator, Speed Measurement

# PID Controller

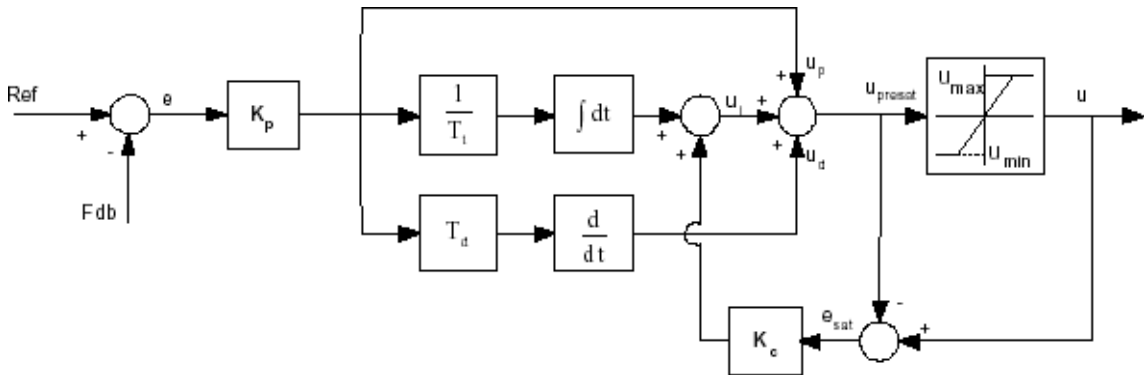
**Purpose** Digital PID controller

**Library** c28xdmclib in Target for TI C2000

## Description



This block implements a 32-bit digital PID controller with antiwindup correction. The inputs are a reference input (ref) and a feedback input (fdb) and the output (out) is the saturated PID output. The following diagram shows a PID controller with antiwindup.



The differential equation describing the PID controller before saturation that is implemented in this block is

$$u_{presat}(t) = u_p(t) + u_i(t) + u_d(t)$$

where  $u_{presat}$  is the PID output before saturation,  $u_p$  is the proportional term,  $u_i$  is the integral term with saturation correction, and  $u_d$  is the derivative term.

The proportional term is

$$u_p(t) = K_p e(t)$$

where  $K_p$  is the proportional gain of the PID controller and  $e(t)$  is the error between the reference and feedback inputs.

The integral term with saturation correction is

$$u_i(t) = \frac{K_p}{T_i} \int_0^t e(\zeta) d\zeta + K_c(u(t) - u_{presat}(t))$$

where  $K_c$  is the integral correction gain of the PID controller.

The derivative term is

$$u_d(t) = K_p T_d \frac{de(t)}{dt}$$

where  $T_d$  is the derivative time of the PID controller. In discrete terms, the derivative gain is defined as  $K_d = T_d/T$ , and the integral gain is defined as  $K_i = T/T_i$ , where  $T$  is the sampling period and  $T_i$  is the integral time of the PID controller.

The above differential equations are transformed into a difference equations by backward approximation.

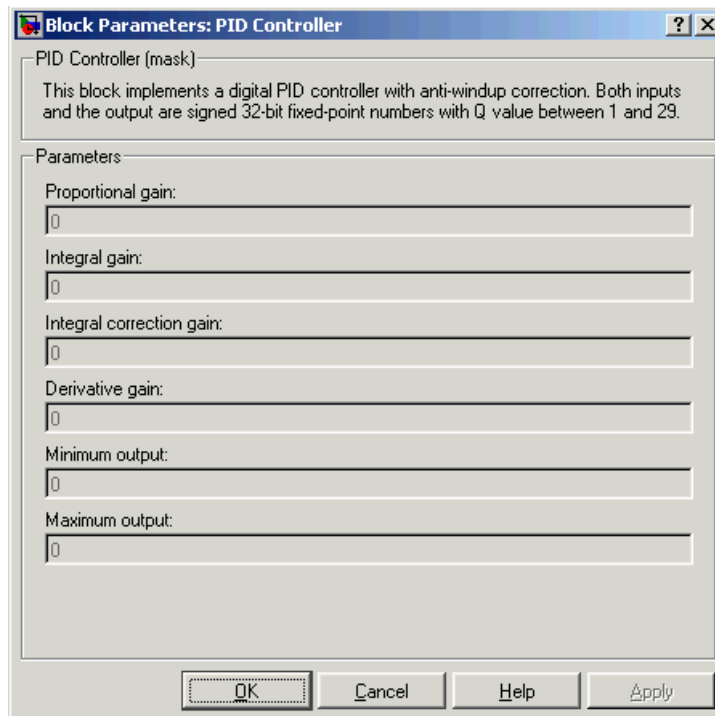
---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

# PID Controller

## Dialog Box



### **Proportional gain**

Amount of proportional gain ( $K_p$ ) to apply to the PID

### **Integral gain**

Amount of gain ( $K_i$ ) to apply to the integration equation

### **Integral correction gain**

Amount of correction gain ( $K_c$ ) to apply to the integration equation

### **Derivative gain**

Amount of gain ( $K_d$ ) to apply to the derivative equation.

### **Minimum output**

Minimum allowable value of the PID output



**Maximum output**

Maximum allowable value of the PID output

**References**

Detailed information on the DMC library is in *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

**See Also**

Clarke Transformation, Inverse Park Transformation, Park Transformation, Space Vector Generator, Speed Measurement

# Ramp Control

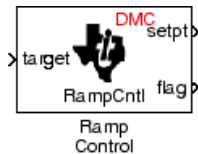
## Purpose

Create ramp-up and ramp-down function

## Library

c28xdmclib in Target for TI C2000

## Description

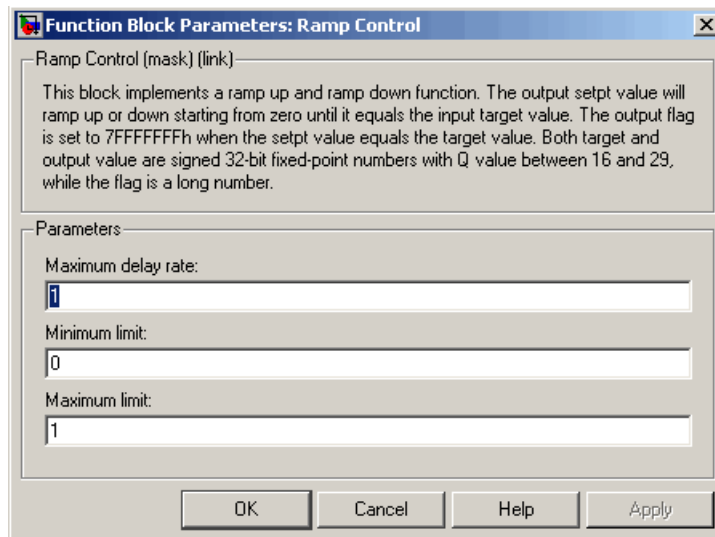


This block implements a ramp-up and ramp-down function. The input is a target value and the outputs are the set point value (setpt) and a flag. The flag output is set to 7FFFFFFFh when the output setpt value reaches the input target value. The target and setpt values are signed 32-bit fixed-point numbers with Q values between 16 and 29. The flag is a long number.

The target value is compared with the setpt value. If they are not equal, the output setpt is adjusted up or down by a fixed step size (0.0000305).

If the fixed step size is relatively large compared to the target value, the output may oscillate around the target value.

## Dialog Box



**Maximum delay rate**

Value that is multiplied by the sampling loop time period to determine the time delay for each ramp step. Valid values are integers greater than 0.

**Minimum limit**

Minimum allowable ramp value. If the input falls below this value, it will be saturated to this minimum. The smallest value you can enter is the minimum value that can be represented in fixed-point data format by the input and output blocks to which this Ramp Control block is connected in your model. If you enter a value below this minimum, an error occurs at the start of code generation or simulation. For example, if your input is in Q29 format, its minimum value is -4.

**Maximum limit**

Maximum allowable ramp value. If the input goes above this value, it will be reduced to this maximum. The largest value you can enter is the maximum value that can be represented in fixed-point data format by the input and output blocks to which this Ramp Control block is connected in your model. If you enter a value above this maximum, an error occurs at the start of code generation or simulation. For example, if your input is in Q29 format, its maximum value is 3.9999....

**See Also**

Ramp Generator

# Ramp Generator

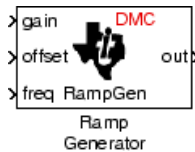
## Purpose

Generate ramp output

## Library

c28xdmclib in Target for TI C2000

## Description



This block generates ramp output (out) from the slope of the ramp signal (gain), DC offset in the ramp signal (offset), and frequency of the ramp signal (freq) inputs. All of the inputs and output are 32-bit fixed-point numbers with Q values between 1 and 29.

## Algorithm

The block's output (out) at the sampling instant  $k$  is governed by the following algorithm:

$$\text{out}(k) = \text{angle}(k) * \text{gain}(k) + \text{offset}(k) "$$

For  $\text{out}(k) > 1$ ,  $\text{out}(k) = \text{out}(k) - 1$ . For  $\text{out}(k) < -1$ ,  $\text{out}(k) = \text{out}(k) + 1$ .

Angle( $k$ ) is defined as follows:

$$\text{angle}(k) = \text{angle}(k-1) + \text{freq}(k) * \text{Maximum step angle}$$

$$\text{for } \text{angle}(k) > 1, \text{angle}(k) = \text{angle}(k) - 1$$

$$\text{for } \text{angle}(k) < -1, \text{angle}(k) = \text{angle}(k) + 1"$$

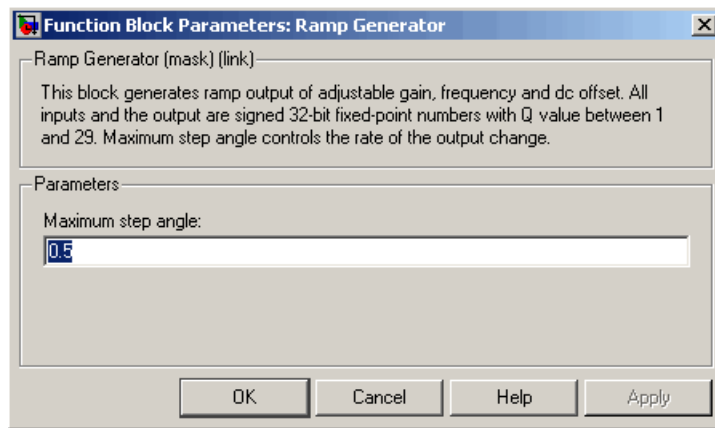
The frequency of the ramp output is controlled by a precision frequency generation algorithm that relies on the modulo nature of the finite length variables. The frequency of the output ramp signal is equal to

$$f = (\text{Maximum step angle} * \text{sampling rate}) / 2^m "$$

where  $m$  represents the fractional length of the data type of the inputs.

All math operations are carried out in fixed-point arithmetic, where the fixed-point fractional length is determined by the block's inputs.

## Dialog Box

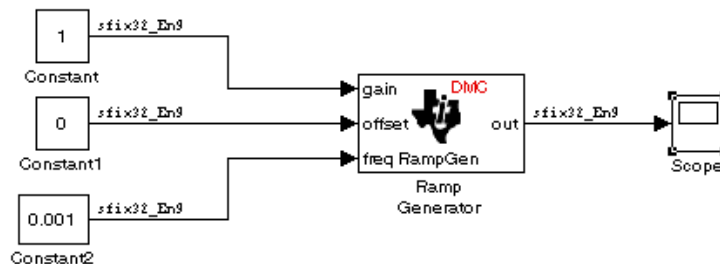


## Maximum step angle

The maximum step size, which determines the rate of change of the output (i.e., the minimum period of the ramp signal).

## Examples

The following model demonstrates the Ramp Generator block. The Constant and Scope blocks are available in Simulink Commonly Used Blocks.



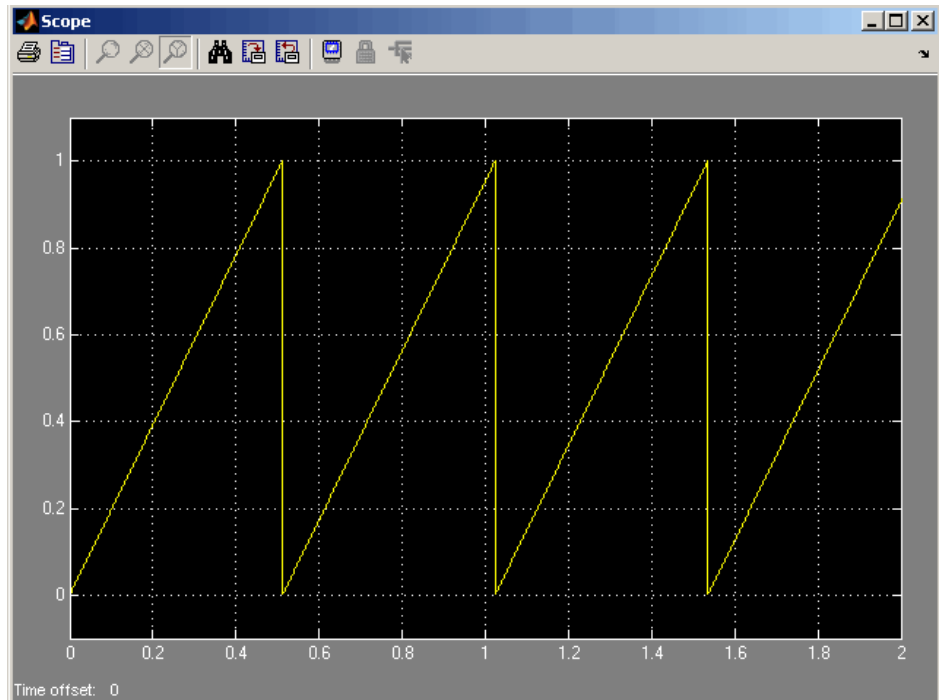
In your model, select **Simulation > Configuration Parameters**. On the **Solver** pane, set **Type** to Fixed-step and **Solver** to discrete (no continuous states). Set the parameter values for the blocks as shown in the following table.

# Ramp Generator

---

Block	Connects to	Parameter	Value
Constant	Ramp Generator - gain	Constant value Sample time Output data type Output scalig value	1 0.001 sfix(32) 2 <sup>-9</sup>
Constant	Ramp Generator - offset	Constant value Sample time Output data type Output scalig value	0 inf sfix(32) 2 <sup>-9</sup>
Constant	Ramp Generator - freq	Constant value Sample time Output data type Output scalig value	0.001 inf sfix(32) 2 <sup>-9</sup>
Ramp Generator	Scope (Simulink block)	Maximum step angle	1

When you run the model, the Scope block generates the following output (drag a zoom box around a portion of the output to change the display).



The expected frequency of the output is

$$f = (\text{maximum step angle} * \text{sampling rate}) / 2^m$$

$$f = (1 * 1000) / 2^9 = 1.9531 \text{ Hz}$$

The expected period is then

$$T = 1/f = 0.5120 \text{ s}$$

which is what the above Scope output shows.

## See Also

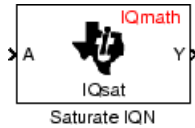
Ramp Control

# Saturate IQN

**Purpose** Saturate IQ number

**Library** `tiiqmathlib` in Target for TI C2000

## Description



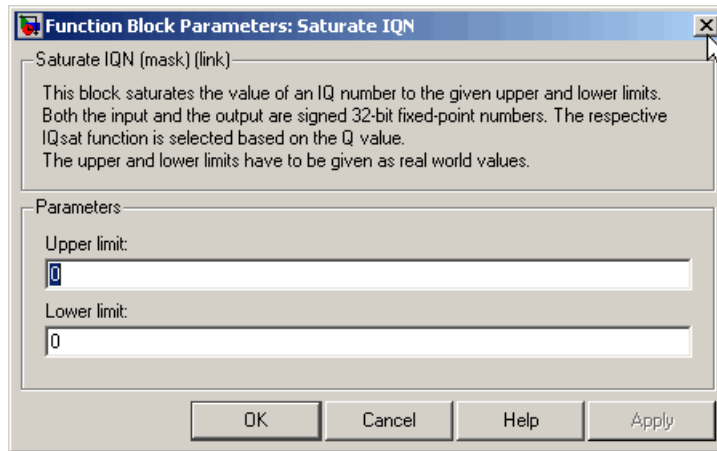
This block saturates an input IQ number to the specified upper and lower limits. The returned value is an IQ number of the same Q value as the input.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



### Upper Limit

Maximum real-world value to which to saturate

### Lower Limit

Minimum real-world value to which to saturate



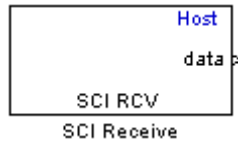
## See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Square Root IQN, Trig Fcn IQN

# SCI Receive

**Purpose** Configure host-side serial communications interface to receive data from serial port

**Library** c2000scilib in Target for TI C2000



## Description

Specify the configuration of data being received from the target by this block.

The data package being received is limited to 16 bytes of ASCII characters, including package headers and terminators. Calculate the size of a package by including the package header, or terminator, or both, and the data size.

Acceptable data types are single, int8, uint8, int16, uint16, int32, or uint32. The number of bytes in each data type is listed in the following table:

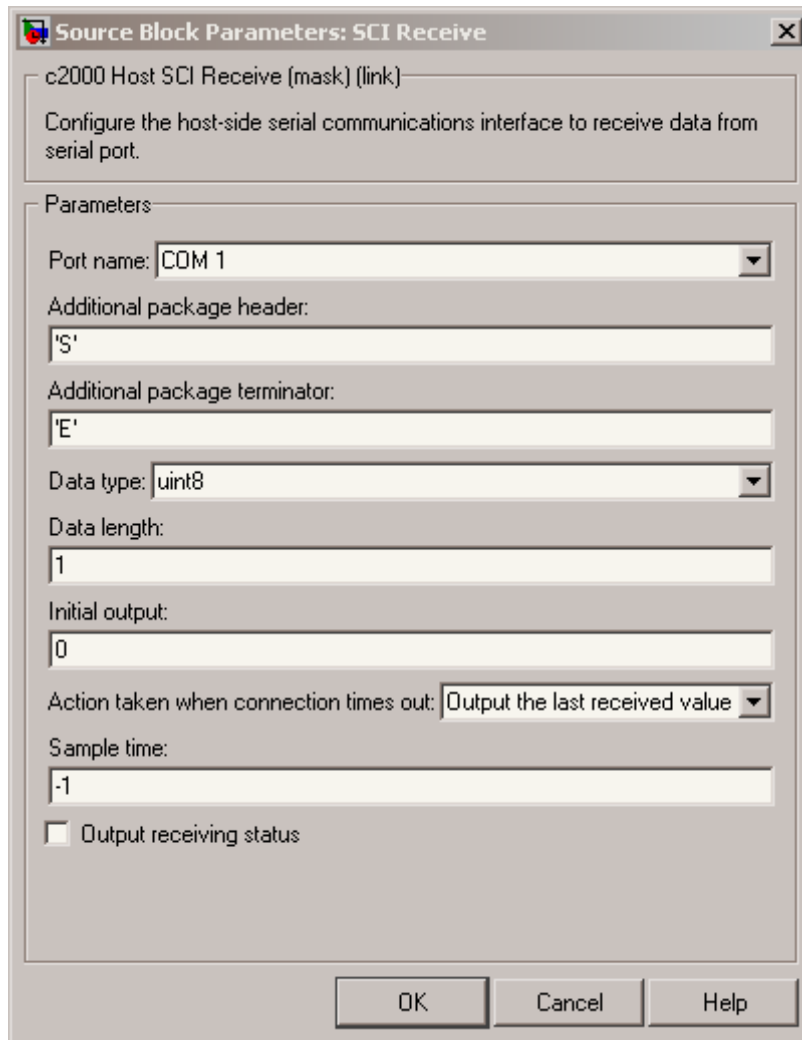
Data Type	Byte Count
single	4 bytes
int8 and uint8	1 byte
int16 and uint16	2 bytes
int32 and uint32	4 bytes

For example, if your data package has package header 'S' (1 byte) and package terminator 'E' (1 byte), that leaves 14 bytes for the actual data. If your data is of type int8, there is room in the data package for 14 int8s. If your data is of type uint16, there is room in the data package for 7 uint16s. If your data is of type int32, there is room in the data package for only 3 int32s, with 2 bytes left over. Even though you could fit two int8s or one uint16 in the remaining space, you may not, because you cannot mix data types in the same package.

The number of data types that can fit into a data package determine the data length (see **Data length** in the Dialog Box description). In the example just given, the 14 for data type `int8` and the 7 for data type `uint16` are the data lengths for each data package, respectively. When the data length exceeds 16 bytes, unexpected behavior, including run time errors, may result.

# SCI Receive

## Dialog Box



The dialog box is titled "Source Block Parameters: SCI Receive" and contains the following fields and options:

- Header: c2000 Host SCI Receive (mask) (link)
- Description: Configure the host-side serial communications interface to receive data from serial port.
- Parameters section:
  - Port name: COM 1 (dropdown menu)
  - Additional package header: 'S' (text field)
  - Additional package terminator: 'E' (text field)
  - Data type: uint8 (dropdown menu)
  - Data length: 1 (text field)
  - Initial output: 0 (text field)
  - Action taken when connection times out: Output the last received value (dropdown menu)
  - Sample time: -1 (text field)
  - Output receiving status (checkbox)
- Buttons: OK, Cancel, Help

### Port name

You may configure up to four COM ports (COM1 through COM4) for up to four host-side SCI Receive blocks.

## Additional package header

This field specifies the data located at the front of the received data package, which is not part of the data being received, and generally indicates start of data. The additional package header must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not received nor are they included in the total byte count.

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the target SCI transmit block.

---

## Additional package terminator

This field specifies the data located at the end of the received data package, which is not part of the data being received, and generally indicates end of data. The additional package terminator must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not received nor are they included in the total byte count.

## Data type

Choice of single, int8, uint8, int16, uint16, int32, or uint32.

The input port of the SCI Transmit block accepts only one of these values. Which value it accepts is inherited from the data type from the input (the data length is also inherited from the input). Data must consist of only one data type; you cannot mix types.

## Data length

How many of **Data type** the block receives (not bytes). Anything more than 1 is a vector. The data length is inherited from the input (the data length input to the SCI Transmit block).

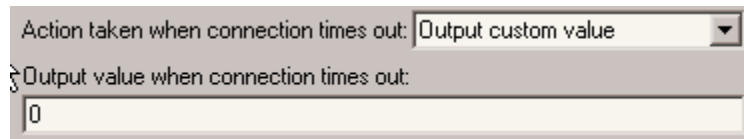
## Initial output

Default value from the Receive block. This value is used, for example, if a connection time-out occurs and the **When connection timeout** field is set to “Output the last received value”, but nothing yet has been received.

## Action Taken when connection times out

Specifies what to output if a connection time-out occurs. If “Output the last received value” is selected, the last received value is what is output, unless none has yet been received, in which case the **Initial output** is considered the last received value.

If “Output customized value” is selected, a field for specifying a custom value is added to the dialog box (as shown in the following figure).

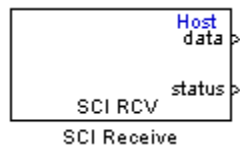


## Sample time

Determines how often the SCI Receive block is called (in seconds). A value of -1 indicates the time is inherited from the model parameters. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-14 for a discussion of block placement and other necessary settings.

## Output receiving status

When this field is checked, the SCI Receive block adds another output port for the transaction status, and appears as shown in the following figure.



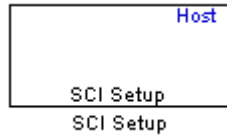
The error status may be one of the following values:

- 0: No errors
- 1: A time-out occurred while the block was waiting to receive data
- 2: There is an error in the received data (checksum error)
- 3: SCI parity error flag — Occurs when a character is received with a mismatch
- 4: SCI framing error flag — Occurs when an expected stop bit is not found

# SCI Setup

**Purpose** Configure COM ports for host-side SCI Transmit and Receive blocks

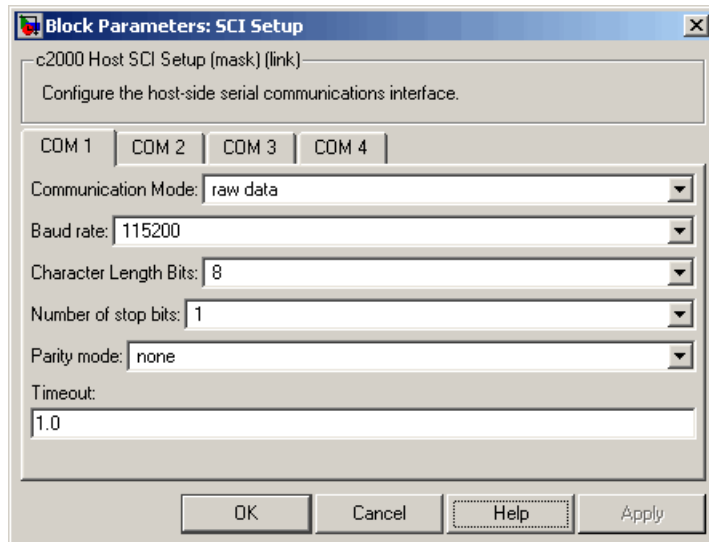
**Library** c2000scilib in Target for TI C2000



## Description

Standardize COM port settings for use by the host-side SCI Transmit and Receive blocks. Setting COM port configurations globally with the SCI Setup block avoids conflicts (e.g., the host-side SCI Transmit block cannot use COM1 with settings different than those the COM1 used by the host-side SCI Receive block) and requires that you set configurations only once for each COM port. The SCI Setup block is a stand alone block.

## Dialog Box





## Communication Mode

Raw data or protocol. Raw data is unformatted and sent whenever the transmitting side is ready to send, whether the receiving side is ready or not. No deadlock condition can occur because there is no wait state. Data transmission is asynchronous. With this mode, it is possible the receiving side could miss data, but if the data is noncritical, using raw data mode can avoid blocking any processes.

If you specify protocol mode, some handshaking between host and target occurs. The transmitting side sends \$SND indicating that it is ready to transmit. The receiving side sends back \$RDY indicating that it is ready to receive. The transmitting side then sends data and, when the transmission is completed, it sends a checksum.

Advantages to using protocol mode include

- Ensures that data is received correctly (checksum)
- Ensures that data is actually received by target
- Ensures time consistency; each side waits for its turn to send or receive

---

**Note** Deadlocks can occur if one SCI Transmit block is trying to communicate with more than one SCI Receive block on different COM ports when both are blocking (using protocol mode). Deadlocks cannot occur on the same COM port.

---

## Baud rate

Choose from 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

## Number of stop bits

Select 1 or 2.

**Parity mode**

Select none, odd, or even.

**Timeout**

Enter any value greater than or equal to 0, in seconds. When the COM port involved is using protocol mode, this value indicates how long the transmitting side waits for an acknowledgement from the receiving side or how long the receiving side waits for data. The system displays a warning message if the time-out is exceeded, every  $n$  number of seconds,  $n$  being the value in

**Timeout.**

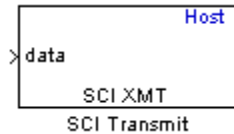
---

**Note** Simulink actually suspends processing for the length of the time-out, and you will not be able to perform any Simulink action. If the time-out is set for a long period of time, it may appear that Simulink has frozen.

---

**Purpose** Configure host-side serial communications interface to transmit data to serial port

**Library** c2000scilib in Target for TI C2000



## Description

Specify the configuration of data being transmitted to the target from this block.

The data package being sent is limited to 16 bytes of ASCII characters, including package headers and terminators. Calculate the size of a package by figuring in package header, or terminator, or both, and the data size.

Acceptable data types are `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. The byte size of each data type is as follows:

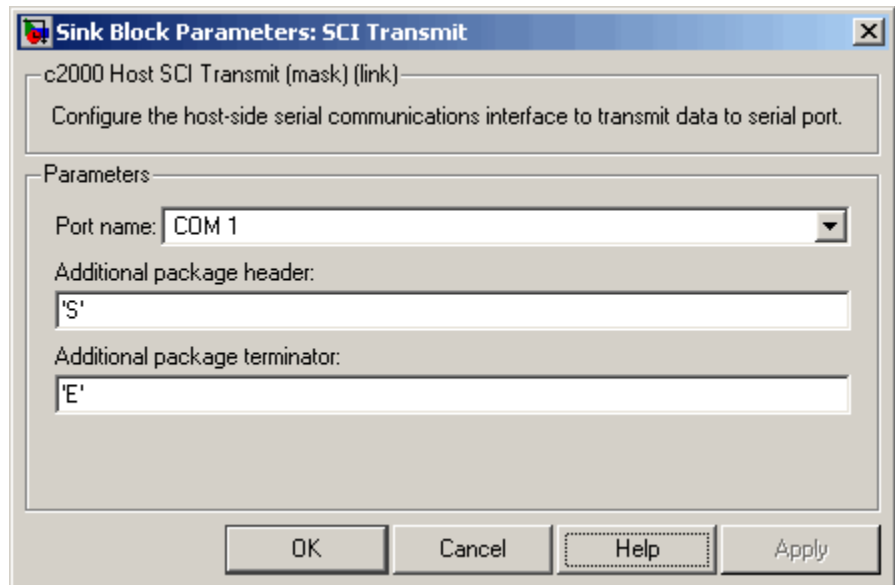
Data Type	Byte Count
<code>single</code>	4 bytes
<code>int8 &amp; uint8</code>	1 byte
<code>int16 &amp; uint16</code>	2 bytes
<code>int32 &amp; uint32</code>	4 bytes

For example, if your data package has package header “S” (1 byte) and package terminator “E” (1 byte), that leaves 14 bytes for the actual data. If your data is of type `int8`, there is room in the data package for 14 `int8`s. If your data is of type `uint16`, there is room in the data package for only 7 `uint16`s. If your data is of type `int32`, there is room in the data package for only 3 `int32`s, with 2 bytes left over. Even though you could fit two `int8`s or one `uint16` in the remaining space, you may not, because you cannot mix data types in the same package.

# SCI Transmit

The number of data types that can fit into a data package determine the data length (see **Data length** in the Dialog Box description). In the example just given, the 14 for data type int8 and the 7 for data type uint16 are the data lengths for each data package, respectively. When the data length exceeds 16 bytes, unexpected behavior, including run time errors, may result.

## Dialog Box



### Port name

You may configure up to four COM ports (COM1 through COM4) for up to four host-side SCI Transmit blocks.

### Additional package header

This field specifies the data located at the front of the transmitted data package, which is not part of the data being transmitted, and generally indicates start of data. The additional package header must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in

this field, but the quotes are not sent nor are they included in the total byte count.

---

**Note** Any additional packager header or terminator must match the additional package header or terminator specified in the target SCI receive block.

---

### **Additional package terminator**

This field specifies the data located at the end of the transmitted data package, which is not part of the data being sent, and generally indicates end of data. The additional package terminator must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in this field, but the quotes are not transmitted nor are they included in the total byte count.

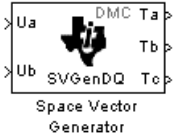
# Space Vector Generator

---

**Purpose** Duty ratios for stator reference voltage

**Library** c28xdmclib in Target for TI C2000

## Description



This block calculates appropriate duty ratios needed to generate a given stator reference voltage using space vector PWM technique. Space vector pulse width modulation is a switching sequence of the upper three power devices of a three-phase voltage source inverter and is used in applications such as AC induction and permanent magnet synchronous motor drives. The switching scheme results in three pseudosinusoidal currents in the stator phases. This technique approximates a given stator reference voltage vector by combining the switching pattern corresponding to the basic space vectors.

The inputs to this block are

- Alpha component — the reference stator voltage vector on the direct axis stationary reference frame ( $U_a$ )
- Beta component — the reference stator voltage vector on the direct axis quadrature reference frame ( $U_b$ )

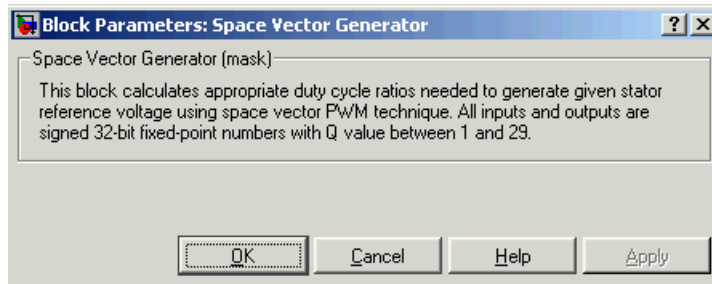
The alpha and beta components are transformed via the inverse Clarke equation and projected into reference phase voltages. These voltages are represented in the outputs as the duty ratios of the PWM1 ( $T_a$ ), PWM3 ( $T_b$ ), and PWM5 ( $T_c$ ).

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



## References

Detailed information on the DMC library is in *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

## See Also

Clarke Transformation, Inverse Park Transformation, Park Transformation, PID Controller, Speed Measurement

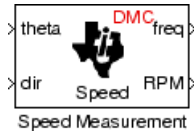
# Speed Measurement

---

**Purpose** Motor speed

**Library** c28xdmclib in Target for TI C2000

## Description



This block calculates the motor speed based on the rotor position when the direction information is available. The inputs are the electrical angle (`theta`) and the direction of rotation (`dir`) from the encoder. The outputs are the speed normalized from 0 to 1 in the Q format (`freq`) and the speed in revolutions per minute (`rpm`).

---

**Note** This block does not call the corresponding Texas Instruments library function during code generation. Instead, the MathWorks code uses the TI functions global Q setting to adjust dynamically the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Understanding the Theta Input to the Block

To indicate the rotational position of your motor, the block expects a 32-bit, fixed-point value that varies from 0 to 1.

Block input `theta` is defined by the following relations:

- A `theta` input signal equal to 0 indicates 0 degrees of rotation.
- A `theta` input signal equal to 1 indicates 360 degrees of rotation (one full rotation).

When the motor spins at a constant speed, `theta` (in counts) from your position sensor (encoder) should increase linearly from 0 to 1 and then abruptly return to 0, like a saw-shaped signal. Adjust the `theta` signal output from your encoder to get the correct input signal range for the Speed Measurement block. Then, convert your encoder signal to 32-bit fixed-point Q format that meets your resolution needs.

For example, if you are using a position sensor that generates 8000 counts for one full revolution of the motor, (0.0450 degrees per count),



you need to reset your counter to 0 after your counter reaches 8000. Each time you read your encoder position, you need to convert the position to a 32-bit, fixed-point Q format value knowing that 8000 is represented as a 1.0. In this example your format could be Q31.

## The Base Speed Parameter

*Base speed* is the maximum motor rotation rate to measure. This value is probably not the maximum speed the motor can achieve.

The Speed Measurement block calculates motor speed from two successive *theta* readings of the motor position,  $\theta_{new}$  and  $\theta_{old}$  (the base speed of the motor; and the time between readings). The maximum speed the block can calculate occurs when the difference between two successive samples [ $\text{abs}(\theta_{new} - \theta_{old})$ ] is 1.0—one full motor revolution occurs between theta samples.

Therefore, the value you provide for the Base speed (in revolutions per minute) parameter is the speed, in revolutions per minute, at which your motor position signal reports one full revolution during one sample time. While the motor may spin faster than the base speed, the block cannot calculate the rotation rate correctly in that case. If the motor completes more than one revolution in one sample time, the calculated speed may be wrong. The block does not know that between samples  $\theta_{new}$  and  $\theta_{old}$ , *theta* wrapped from 1 back to 0 and started counting up again.

The time difference between the two theta readings is the sample time. The Speed Measurement block inherits the sample time from the upstream block in your model. You set the sample time in the upstream block and then the Speed Measurement block uses that sample time to calculate the rotation rate of the motor.

## The Sample Time Calculation

Motor speed measurements depend on the sample time you set in the model. Your sample time must be short enough to measure the full speed of the motor.

Two parameters drive your sample time—motor base speed and encoder counts per revolution. To be able to measure the maximum rotation

# Speed Measurement

---

rate, you must take at least one sample for each revolution. For a motor with base speed equal to 1000 rpm, which is 16.67 rps, you need to sample at  $1/16.67$  s, which is 0.06 s/sample. This sample rate of 16.67 samples per second is the maximum sample time (lowest sample rate) that assures you can measure the full speed of the motor.

Using the same sample rate assumption, the minimum speed the block can measure depends on the encoder counts per revolution. At the minimum measurable motor speed, the encoder generates one count per sample period—16.67 counts per second. For an encoder that generates 8000 counts per revolution, this results in being able to measure a speed of  $[(16.67 \text{ counts/s}) * (0.045 \text{ degrees/count})] = 0.752 \text{ degrees per second}$ , or about 45 degrees per minute—one-eighth RPM.

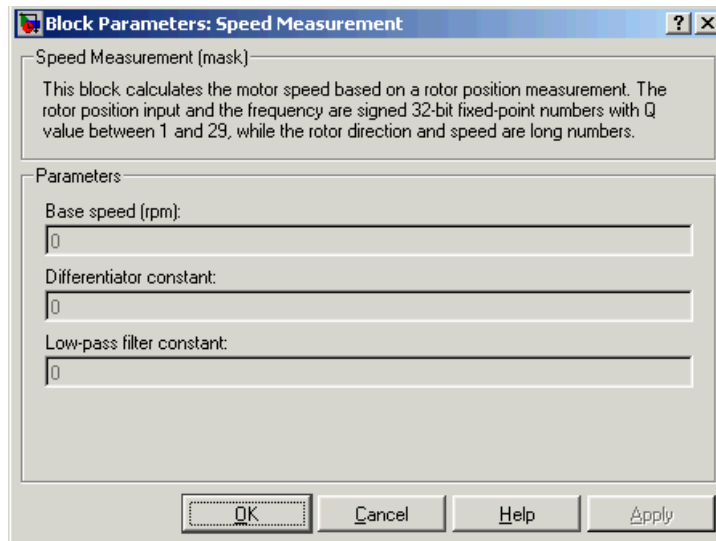
## The Differentiator Constant

The differentiator constant is a scalar value applied to the block output. For example, setting it to 1 produces no effect on the output. Setting the constant to  $1/4$  multiplies the frequency and revolutions per minute outputs by 0.25. This setting can be useful when your motor has multiple pole pairs, and one electrical revolution is not equal to one mechanical revolution. The constant lets you account for the difference between electrical and mechanical rotation rates.

## The Low-Pass Filter Constant

This block includes filtering capability if your position signal is noisy. Setting the filter constant to 0 disables the filter. Setting the filter constant to 1 filters out the entire signal and results in a block output equal to 0. Use a simulation to determine the best filter constant for your system. Your goal is to filter enough to remove the noise on your signal but not so much that the speed measurements cannot react to abrupt speed changes.

## Dialog Box



### Base speed

Maximum speed of the motor to measure in revolutions per minute.

### Differentiator constant

Constant used in the differentiator equation that describes the rotor position.

### Low-pass filter constant

Constant to apply to the lowpass filter. This constant is  $1/(1+T*(2\pi f_c))$ , where  $T$  is the sampling period and  $f_c$  is the cutoff frequency. The  $1/(2\pi f_c)$  term is the lowpass filter time constant. This block uses a lowpass filter to reduce noise generated by the differentiator.

## Example

The following example demonstrates how you configure the Speed Measurement block.

# Speed Measurement

---

## Configuring the Speed Measurement Block to Measure Motor Speed

Use the following process to set up the Speed Measurement block parameters.

- 1 Add the block to your model.
- 2 Open the block dialog box to view the block parameters.
- 3 Set the value for **Base Speed** to the maximum speed to measure, in revolutions per minute.
- 4 Enter values for **Differentiator** and **Low-Pass Filter Constant**.
- 5 Click **OK** to close the dialog box.

## Setting the Sample Time to Measure Motor Speed

Use the following process to set the sample time for measuring the motor speed.

- 1 Open the block dialog box for the block before the Speed Measurement block in your model (the upstream or driving block).
- 2 Set the sample time parameter in the upstream block according to the sample time guidelines described in The Sample Time Calculation.
- 3 Click **OK** to close the dialog box.

## References

For detailed information on the DMC library, refer to *C/F 28xx Digital Motor Control Library*, SPRC080, available at the Texas Instruments Web site.

## See Also

Clarke Transformation, Inverse Park Transformation, Park Transformation, PID Controller, Space Vector Generator

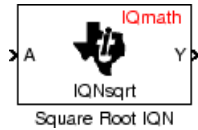
## Purpose

Square root or inverse square root of IQ number

## Library

tiiqmathlib in Target for TI C2000

## Description



This block calculates the square root or inverse square root of an IQ number and returns an IQ number of the same Q format. The block uses table lookup and a Newton-Raphson approximation.

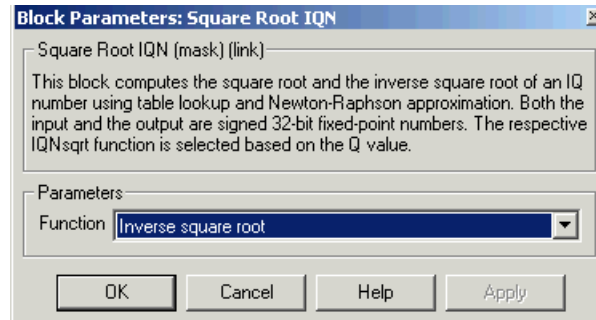
Negative inputs to this block return a value of zero.

---

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

---

## Dialog Box



## Function

Whether to calculate the square root or inverse square root

- Square root (`_sqrt`) — Compute the square root.
- Inverse square root (`_isqrt`) — Compute the inverse square root.

# Square Root IQN

---

## See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Trig Fcn IQN

# Switch External Mode Configuration

---

<b>Purpose</b>	Configure model for external mode or executable building
<b>Library</b>	Target for TI C2000/ C2000 Driver Library/ Utilities
<b>Description</b>	<p>Place the Switch External Mode Configuration block in your model and double-click it to run a convenience function to configure your model for building an executable, or executing your model in external mode. When you double-click the block, a dialog box appears. Choose either <b>Building an executable</b> or <b>External mode</b>, and click <b>OK</b>.</p> <p>When you choose building an executable, messages at the command line inform you the following steps are taken to configure your model:</p> <ol style="list-style-type: none"><li><b>1 Inline parameters</b> are selected (under Optimization in the Configuration Parameters dialog box). This is required for ASAP2 generation</li><li><b>2 Normal</b> simulation mode is selected (in the Simulation menu, and drop-down list in the toolbar).</li><li><b>3</b> ASAP2 is selected as the <b>Interface</b> (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).</li></ol> <p>When you choose external mode, messages at the command line inform you the following steps are taken to configure your model:</p> <ol style="list-style-type: none"><li><b>1 Inline parameters</b> are selected (under Optimization in the Configuration Parameters dialog box). This is required for external mode.</li><li><b>2 External</b> simulation mode is selected (in the Simulation menu, and drop-down list in the toolbar).</li><li><b>3</b> External mode is selected as the <b>Interface</b> (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).</li></ol>

# Switch External Mode Configuration

---

See “Using External Mode” on page 2-9 for instructions for converting a model to use external mode for signal logging and parameter tuning.



## Purpose

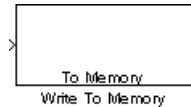
Write data to target memory

## Library

c280xspchiplib or c281xspchiplib in Target for TI C2000

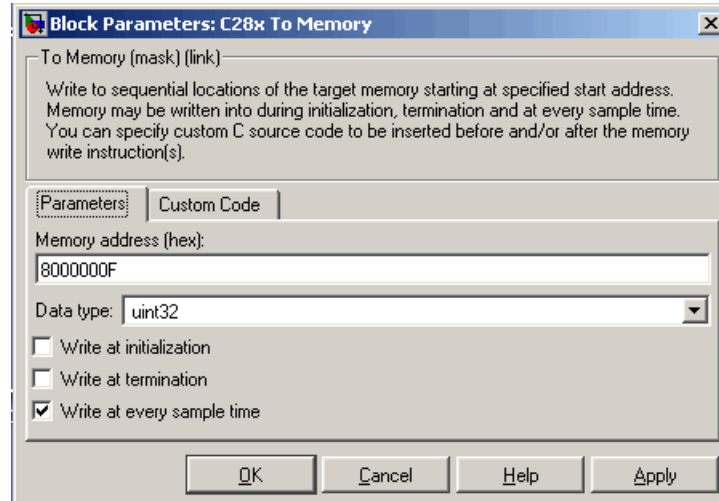
## Description

This block sends data of the specified data type to a particular memory address on the target.



## Dialog Box

### Parameters Pane



### Memory address

Address of the target memory location, in hexadecimal, to which to write data

### Data type

Type of data to be written to the above memory address. Valid data types are double, single, int8, uint8, int16, uint16,

# To Memory

---

int32, and uint32. The data is cast from the selected data type to 16-bit data.

**Write at initialization**

Whether to write the specified **Value** at program start

**Value**

First value of data to be written to memory at program start

**Write at termination**

Whether to write the specified **Value** at program end

**Value**

Last value of data to be written to memory at program termination

**Write at every sample time**

Whether to write data in real time during program execution

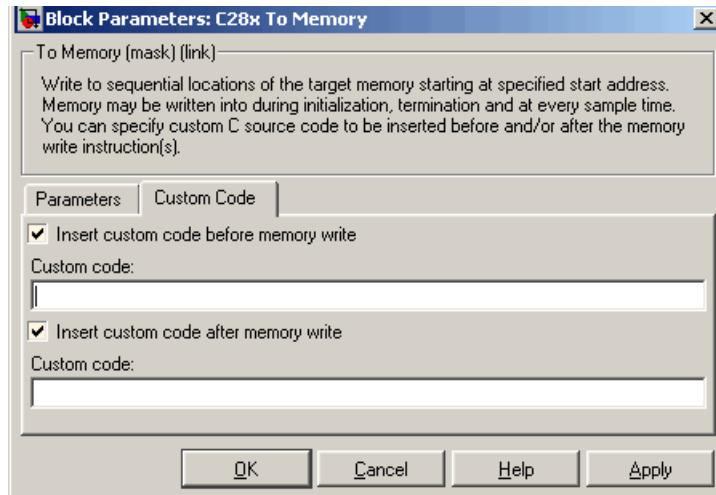
---

**Note** If your To Memory block is set to write to memory at every sample time interval (that is, it has an incoming port) and it receives a vector signal input of N elements, a corresponding memory region starting with the specified **Memory address** is updated at every sample time. If you specify an **Initial** and/or **Termination value**, that value is written to all locations in the same memory region at initialization and/or termination.

If your To Memory block does not write to memory at every sample time (that is, it does not have an incoming port) and you specify an **Initial** and/or **Termination value**, that value is written to a single memory location that corresponds to the specified **Memory address**.

---

## Custom Code Pane



### Insert custom code before memory write

C code to execute before writing to the specified memory address. An example of code that might be inserted here is

```
asm ( " EALLOW " )
```

which enables write access to the device emulation registers on the C2812 DSP.

### Insert custom code after memory write

C code to execute after writing to the specified memory address. An example of code that may be inserted here is

```
asm ( " DIS " )
```

which disables write access to the device emulation registers on the C2812 DSP.

## See Also

From Memory

# To RTDX

---

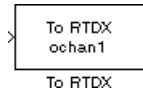
## Purpose

Add RTDX output channel

## Library

rtdxBlocks in Target for TI C2000

## Description



When you generate code from Simulink in Real-Time Workshop with a To RTDX block in your model, code generation inserts the C commands to create an RTDX output channel on the target. Output channels transfer data from the target to the host.

The generated code contains this command:

```
RTDX_enableOutput(&channelname)
```

where channelname is the name you enter in the **channelName** field in the To RTDX dialog box.

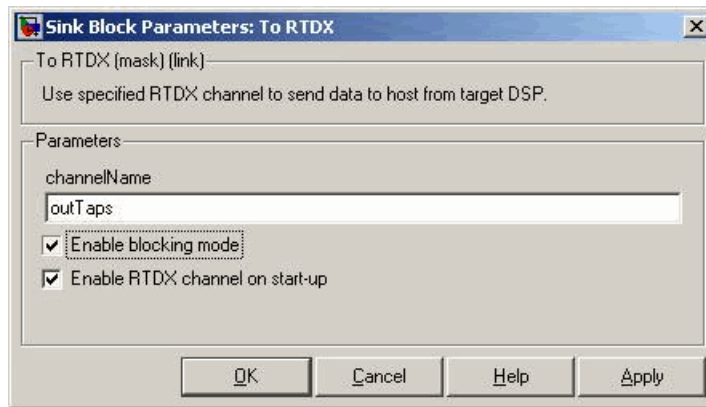
---

**Note** To RTDX blocks work only in code generation and when your model runs on your target. In simulations, this block does not perform any operations.

---

To use RTDX blocks in your model, you must do the following:

- 1 Add one or more To RTDX or From RTDX blocks to your model.
- 2 Download and run your model on your target.
- 3 Enable the RTDX channels from MATLAB or use **Enable RTDX channel on start-up** on the block dialog.
- 4 Use the readmsg and writemsg functions in MATLAB to send and retrieve data from the target over RTDX.

**Dialog  
Box****Channel name**

Name of the output channel to be created by the generated code. The channel name must meet C syntax requirements for length and character content.

**Enable blocking mode**

Enables blocking mode (selected by default). In blocking mode, writing a message is suspended while the RTDX channel is busy, that is, when data is being written in either direction. The code waits at the RTDX\_write call site while the channel is busy. Note that any interrupt of the higher priority will temporarily divert the program execution from this site, but it will eventually come back and wait until the channel stops writing.

When blocking mode is not enabled (when the check box is cleared), writing a message is abandoned if the RTDX channel is busy, and the code proceeds with the current iteration.

**Enable RTDX channel on start-up**

Enables the RTDX channel when you start the channel from MATLAB. With this selected, you do not need to use the enable function in Link for Code Composer Studio Development Tools to prepare your RTDX channels. This option applies only to the

## To RTDX

---

channel you specify in **Channel name**. You do have to open the channel.

### **See Also**

From RTDX

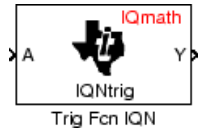
## Purpose

Sine, cosine, or arc tangent of IQ number

## Library

tiiqmathlib in Target for TI C2000

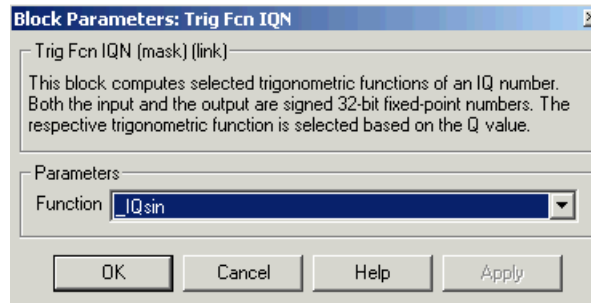
## Description



This block calculates basic trigonometric functions and returns the result as an IQ number. Valid Q values for `_IQsinPU` and `_IQcosPU` are 1 to 30. For all others, valid Q values are from 1 to 29.

**Note** The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “About the IQmath Library” on page 5-2 for more information.

## Dialog Box



## Function

Type of trigonometric function to calculate:

- `_IQsin` — Compute the sine ( $\sin(A)$ ), where A is in radians.
- `_IQsinPU` — Compute the sine per unit ( $\sin(2\pi A)$ ), where A is in per-unit radians.
- `_IQcos` — Compute the cosine ( $\cos(A)$ ), where A is in radians.
- `_IQcosPU` — Compute the cosine per unit ( $\cos(2\pi A)$ ), where A is in per-unit radians.

# Trig Fcn IQN

---

- `_IQatan` — Compute the arc tangent ( $\tan(A)$ ), where A is in radians.

## See Also

Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN



## A

- Absolute IQN block 7-2
- acquisition window
  - ADC blocks
    - ACQ\_PS 3-2
- ADC blocks
  - C281x 7-105
- applications
  - TI C2000 1-2
- Arctangent IQN block 7-3
- ASAP2 files, generating 2-18
- asymmetric vs. symmetric waveforms 7-139
- asynchronous interrupt processing 1-14

## B

- blocks
  - adding to model 1-31
  - CAN Calibration Protocol (C2000) 7-10
  - recommendations 1-20
  - Switch External Mode Configuration 7-247

## C

- C2000 Library
  - SCI Receive
    - Host side 7-226
  - SCI Setup
    - Host side 7-232
  - SCI Transmit
    - Host side 7-235
- c2000lib startup 1-26
- C280x ADC block 7-5
- C280x eCAN Receive block 7-15
- C280x eCAN Transmit block 7-19
- c280x eCAP block 7-23
- C280x ePWM block 7-34
- C280x eQEP block 7-54
- c280x GPIO block 7-70 7-78
- C280x Hardware Interrupt block 7-80

- C280x SCI Receive block 7-85
- C280x SCI Transmit block 7-92
- C280x SPI Receive block 7-99
- C280x SPI Transmit block 7-102
- C280x SW Int Trigger 7-95
- C281x ADC block 7-105
- C281x CAP block 7-110
- C281x eCAN Receive block 7-116
- C281x eCAN Transmit block 7-120
- C281x GPIO Digital Input block 7-124
- C281x GPIO Digital Output block 7-128
- C281x PWM block 7-137
- C281x QEP block 7-148
- C281x SCI Receive block 7-151
- C281x SCI Transmit block 7-158
- C281x SPI Receive block 7-165
- C281x SPI Transmit block 7-168
- C281x SW Int Trigger 7-161
- C281x Timer block 7-171
- CAN Calibration Protocol (C2000) block 7-10
- CAN/eCAN
  - C280x Transmit block 7-19
  - C280xReceive block 7-15
  - C281x Transmit block 7-120
  - C281xReceive block 7-116
  - timing parameters
    - bit rate 2-3
- capture block
  - C281x 7-110
- CCS 1-10
  - See also* Code Composer Studio
- Clarke Transformation block 7-174
- clock speed 1-14
- Code Composer Studio 1-10
- code generation
  - overview 1-33
- code optimization 5-10
- configuration default 1-10
- configuration parameters
  - setting 1-22

- conversion
  - float to IQ number 7-188
  - IQ number to different IQ number 7-208
  - IQ number to float 7-204
- CPU clock speed 1-14
- Custom Board block 7-177

## D

- data type support 1-12
- data types
  - conversion 5-9
- deadband
  - C281x PWM 7-144
- default build configuration 1-10
- device driver blocks
  - CAN Calibration Protocol (C2000) 7-10
- digital motor control. *See* DMC library
- Division IQN block 7-179
- DMC library
  - Clarke Transformation 7-174
  - Inverse Park Transformation 7-202
  - Park Transformation 7-211
  - PID controller 7-214
  - ramp control 7-218
  - ramp generator 7-220
  - Space Vector Generator 7-238
  - Speed Measurement 7-240
- duty ratios 7-238

## E

- enhanced capture channel 7-23
- enhanced quadrature encoder pulse module
  - C280x 7-54
- ePWM blocks
  - C280x 7-34

## F

- fixed-point numbers 5-4

- flash
  - stand alone applications 4-2
- flash memory 1-6
- Float to IQN block 7-188
- floating-point numbers
  - convert to IQ number 7-188
- four-quadrant arctangent 7-3
- Fractional part IQN block 7-189
- Fractional part IQN x int32 block 7-190
- From Memory block 7-191
- From RTDX block 7-193

## G

- GPIO input
  - c280x 7-70
  - C281x 7-124
- GPIO output
  - c280x 7-78
  - C281x 7-128

## H

- hardware 1-4
- high-speed peripheral clock 1-14

## I

- I/O
  - C281x input 7-124
  - C281x output 7-128
- Idle Task block 7-197
- Integer part IQN block 7-200
- Integer part IQN x int32 block 7-201
- interrupt
  - software triggered for C280x 7-95
  - software triggered for C281x 7-161
- Inverse Park Transformation block 7-202
- IQ Math library 5-2
  - Absolute IQN block 7-2
  - Arctangent IQN block 7-3

- building models 5-9
- code optimization 5-10
- common characteristics 5-3
- Division IQN block 7-179
- Float to IQN block 7-188
- Fractional part IQN block 7-189
- Fractional part IQN x int32 block 7-190
- Integer part IQN block 7-200
- Integer part IQN x int32 block 7-201
- IQN to Float block 7-204
- IQN x int32 block 7-205
- IQN x IQN block 7-206
- IQN1 to IQN2 block 7-208
- IQN1 x IQN2 block 7-209
- Magnitude IQN block 7-210
- Q format notation 5-5
- Saturate IQN block 7-224
- Square Root IQN block 7-245
- Trig Fcn IQN block 7-255

**IQ numbers**

- convert from float 7-188
- convert to different IQ 7-208
- convert to float 7-204
- fractional part 7-189
- integer part 7-200
- magnitude 7-210
- multiply 7-206
- multiply by int32 7-205
- multiply by int32 fractional result 7-190
- multiply by int32 integer part 7-201
- square root 7-245
- trigonometric functions 7-255

- IQN to Float block 7-204
- IQN x int32 block 7-205
- IQN x IQN block 7-206
- IQN1 to IQN2 block 7-208
- IQN1 x IQN2 block 7-209

**M**

- Magnitude IQN block 7-210
- math blocks. *See* IQ Math library
- MathWorks software 1-6
- memory management 1-23
- messages
  - F2808 eZdsp 7-16
  - F2812 eZdsp 7-117
- model
  - add blocks 1-31
  - building overview 1-23
  - creation overview 1-19
  - IQmath library 5-9
- multiplication
  - IQN x int32 7-205
  - IQN x int32 fractional part 7-190
  - IQN x int32 integer part 7-201
  - IQN x IQN 7-206
  - IQN1 x IQN2 7-209

**O**

- operating system requirements 1-4
- optimization code 5-10

**P**

- Park Transformation block 7-211
- phase conversion 7-174
- PID controller 7-214
- PWM blocks
  - C281x 7-137

**Q**

- Q format 5-5
- quadrature encoder pulse circuit
  - C28x 7-148

**R**

- ramp control block 7-218
- ramp generator block 7-220
- reference frame conversion
  - inverse Park transformation 7-202
  - Park transformation 7-211
- reset 1-24
- RTDX
  - from 7-193
  - to 7-252

**S**

- sample time
  - F2812 eZdsp 7-17
- Saturate IQN block 7-224
- scheduling 1-13
- SCI Receive
  - Host side 7-226
- SCI Setup
  - Host side 7-232
- SCI Transmit
  - Host side 7-235
- SCI Transmit and Receive blocks
  - Host side
    - Setup 7-232
- serial communications interface
  - C280x receive 7-85
  - C280x transmit 7-92
  - C281x receive 7-151
  - C281x transmit 7-158
- serial peripheral interface
  - C280x receive 7-99

- C280x transmit 7-102
- C281x receive 7-165
- C281x transmit 7-168
- setting up hardware 1-4
- signed fixed-point numbers 5-5
- simulation parameters
  - automatic 1-28
- software requirements 1-6
- Space Vector Generator block 7-238
- Speed Measurement block 7-240
- Square Root IQN block 7-245
- startup c2000lib 1-26
- Switch External Mode Configuration block 7-247

**T**

- target configuration
  - example 6-2 7-184
  - F2808 eZdsp 6-2 7-180
- target model creation 1-19
- Target Preferences blocks
  - Custom Board 7-177
- TI software 1-6
- timing
  - interrupts 1-13
- To Memory block 7-249
- To RTDX block 7-252
- Trig Fcn IQN block 7-255

**W**

- waveforms 7-139