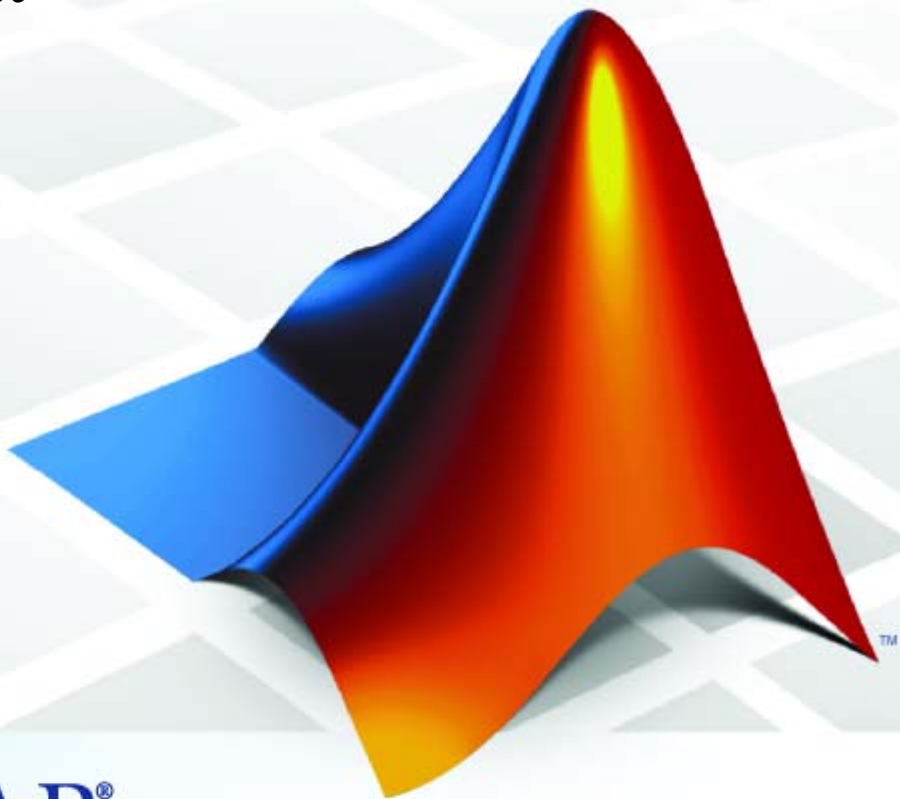


Target Support Package™ 4

User's Guide

For Use with TI's C2000™



MATLAB®
& **SIMULINK®**

How to Contact The MathWorks



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup
www.mathworks.com/contact_TS.html Technical Support



suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com 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 Support Package™ User's Guide

© COPYRIGHT 2003–2009 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 and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be 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 for more information.

Revision History

November 2003	Online only	New for Version 1.0 (Release 13SP1+)
June 2003	Online only	New for Version 1.1 (Release 14)
October 2004	Online only	Revised for Version 1.1.1 (Release 14SP1)
December 2004	Online only	Revised for Version 1.2 (Release 14SP1+)
March 2005	Online only	Revised for Version 1.2.1 (Release 14SP2)
September 2005	Online only	Revised for Version 1.3 (Release 14SP3)
March 2006	Online only	Revised for Version 2.0 (Release 2006a)
September 2006	Online only	Revised for Version 2.1 (Release 2006b)
March 2007	Online only	Revised for Version 2.2 (Release 2007a)
September 2007	Online only	Revised for Version 2.3 (Release 2007b)
March 2008	Online only	Revised for Version 3.0 (Release 2008a)
October 2008	Online only	Revised for Version 3.1 (Release 2008b)
March 2009	Online only	Revised for Version 3.2 (Release 2009a)
September 2009	Online only	Revised for Version 4.0 (Release 2009b)

Getting Started

1

Product Overview	1-2
Introduction	1-2
Product Description	1-2
Setting Up and Configuring	1-3
System Requirements	1-3
Supported Hardware	1-3
Installing and Configuring Software	1-3
Verifying the Configuration	1-4
Code Composer Studio	1-7
Using Code Composer Studio with Target Support Package Software	1-7
Default Project Configuration	1-7
Data Type Support	1-9
Scheduling and Timing	1-10
Overview	1-10
Timer-Based Interrupt Processing	1-10
Asynchronous Interrupt Processing	1-11
Sharing General Purpose Timers between C281x Peripherals	1-16
Example 1	1-17
Example 2	1-21
Overview of Creating Models for Targeting	1-25
Accessing the Target Support Package Block Library	1-25
Online Help	1-26
Blocks with Restrictions	1-26
Setting Simulation Configuration Parameters	1-28
Building Your Model	1-29

Using the c2000lib Blockset	1-31
Introduction	1-31
Hardware Setup	1-31
Starting the c2000lib Library	1-32
Setting Up the Model	1-33
Adding Blocks to the Model	1-36
Generating Code from the Model	1-39

Configuring Timing Parameters for CAN Blocks

2

The CAN Blocks	2-2
Setting Timing Parameters	2-3
Accessing the Timing Parameters	2-3
Determining Timing Parameter Values	2-6
CAN Bit Timing Example	2-7
Parameter Tuning and Signal Logging	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

What Is an Acquisition Window?	3-2
Configuring ADC Parameters for Acquisition Window Width	3-5
Accessing the ADC Parameters	3-5
Examples	3-7

Using the IQmath Library

4

About the IQmath Library	4-2
Introduction	4-2
Common Characteristics	4-3
References	4-3
Fixed-Point Numbers	4-4
Notation	4-4
Signed Fixed-Point Numbers	4-5
Q Format Notation	4-5
Building Models	4-10
Overview	4-10
Converting Data Types	4-10
Using Sources and Sinks	4-11
Choosing Blocks to Optimize Code	4-11

Programming Flash Memory

5

Introduction	5-2
Installing TI Flash APIs	5-3
Configuring the DSP Board Bootloader	5-4
Configuring the Software for Automatic Flash Programming	5-5
Selectively Erase, Program, or Verify Specific Flash Sectors	5-7
Placing Additional Code or Data on Unused Flash Sectors	5-8

Block Reference

6

C280x Chip Support (c280xlib)	6-2
C2802x Chip Support (c2802xlib)	6-4
C281x Chip Support (c281xlib)	6-6
C28x3x Chip Support (c2833xlib)	6-8
C28x DMC (c28xdmclib)	6-10
C28x IQmath (tiiqmathlib)	6-11
CAN Message Handling Blocks (canmsglib)	6-12
Host Communication (hostcommlib)	6-13
Host SCI Blocks (c2000scilib)	6-14
RTDX Instrumentation (rtDXBlocks)	6-15
Target Preferences (c2000tgtpreflib)	6-16

Blocks — Alphabetical List

7

Index

Getting Started

- “Product Overview” on page 1-2
- “Setting Up and Configuring” on page 1-3
- “Code Composer Studio” on page 1-7
- “Data Type Support” on page 1-9
- “Scheduling and Timing” on page 1-10
- “Sharing General Purpose Timers between C281x Peripherals” on page 1-16
- “Overview of Creating Models for Targeting” on page 1-25
- “Using the c2000lib Blockset” on page 1-31

Product Overview

In this section...
“Introduction” on page 1-2
“Product Description” on page 1-2

Introduction

This chapter describes how to use Target Support Package™ software to create and execute applications on Texas Instruments™ C2000™ microcontrollers. To use the targeting software, become familiar with creating Simulink® models and with the basic concepts of using Real-Time Workshop® for automatic code generation. For more information about these concepts, refer to the “Real-Time Workshop” documentation.

Product Description

Use Target Support Package to deploy generated code for real-time execution on embedded microprocessors, microcontrollers, and DSPs. Using Target Support Package, you can integrate peripheral devices and real-time operating systems with the algorithms created using Embedded MATLAB™, Simulink®, and Stateflow®. You can deploy the resulting executable onto embedded hardware for on-target rapid prototyping, real-time performance analysis, and field production.

Setting Up and Configuring

In this section...

“System Requirements” on page 1-3

“Supported Hardware” on page 1-3

“Installing and Configuring Software” on page 1-3

“Verifying the Configuration” on page 1-4

System Requirements

For detailed information about the software and hardware required to use Target Support Package software, refer to the Target Support Package system requirements areas on the MathWorks Web site:

- Requirements for Target Support Package:
www.mathworks.com/products/target-package/requirements.html
- Requirements for use with TI's C2000:
www.mathworks.com/products/target-package/ti-adaptor/

Supported Hardware

For a list of supported hardware, visit
<http://www.mathworks.com/products/target-package/supportedio.html>.

Installing and Configuring Software

Consult the System Requirements for Target Support Package on the MathWorks website. Only use *supported versions* of the software listed under “Third-Party Target Support Package Requirements”. Uninstall unsupported versions *before* installing supported versions. Doing so prevents errors that occur when the Windows Environment Variables points to the unsupported versions.

The System Requirements Web page describes where you can obtain the additional third-party software, and when available, provides links for downloading that software.

Install the software listed in the following order:

- 1** Install the required and optional MathWorks software. (The software license you purchase determines which products are available.)
- 2** Install TI Code Composer Studio™ (CCS).
- 3** Install TI Service Release for CCS.
- 4** Install the TI Code Generation Tools for Windows.
- 5** If you are using a Spectrum Digital board, download and install the matching Spectrum Digital Driver.
- 6** If you are using RTDX for C28x host/target communications, download and install TI DSP/BIOS.
- 7** If you are going to program flash memory with stand-alone code, download the TI Flash API for your target processor.

Configure CCS as follows:

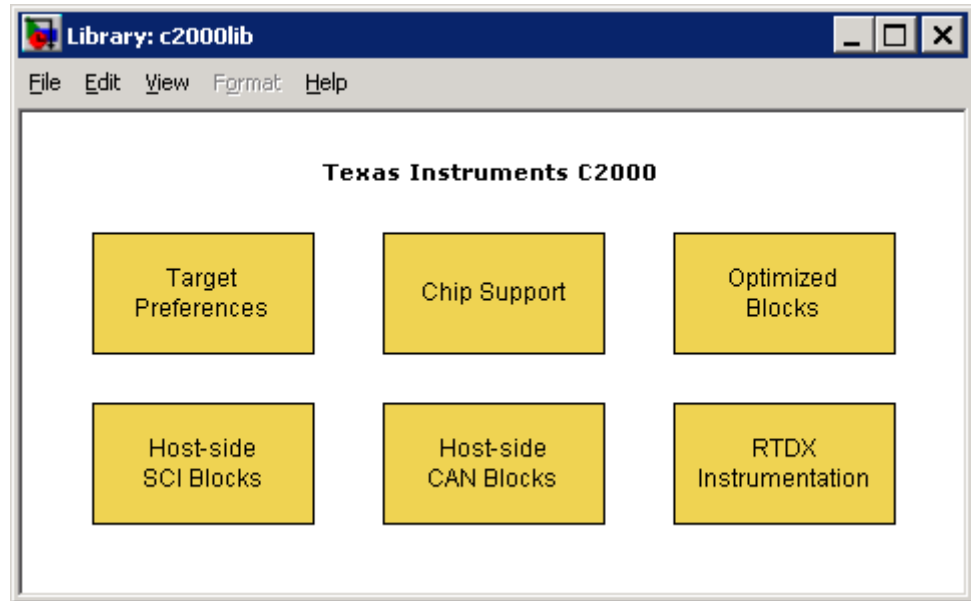
- 1** In CCS, open **Help > About > Component manager > Build tools > TMS320C28XX** and select (check) **C2000 Code Generation Tools**.
- 2** With the Component manager open, open **Target Content(DSP/BIOS) > TMS320C28XX** and select **Texas Instruments DSP/BIOS**.
- 3** Save, exit, and restart CCS.

Verifying the Configuration

To determine whether Target Support Package software is present on your system, enter this command at the MATLAB® prompt:

```
c2000lib
```

MATLAB displays the C2000 block libraries, as shown here:



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

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

```
ccsboardinfo
```

With CCS installed and configured, MATLAB returns a list of the boards that CCS recognizes on your machine like the following example:

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 Support Package software 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, 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).

Code Composer Studio

In this section...

“Using Code Composer Studio with Target Support Package Software” on page 1-7

“Default Project Configuration” on page 1-7

Using Code Composer Studio with Target Support Package Software

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 Support Package software and Real-Time Workshop software, CCS provides an integrated environment that, once installed, requires no coding.

Executing code generated from Real-Time Workshop software on a particular target requires that you tailor the code to the specific hardware target. Target-specific code includes I/O device drivers and interrupt service routines (ISRs). The software must use CCS to compile and link the generated source code in order to load and execute on a TI DSP. To help you to build an executable, Target Support Package software uses Embedded IDE Link™ software 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 Embedded IDE Link software.

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 Support Package software use a custom configuration that provides a third combination of build and optimization settings — **CustomMW**.

Default Build Options in the CustomMW Configuration

The default settings for CustomMW 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 and 32-bit data types, but does not have native 8-bit data types. Simulink models and Target Support Package software support many data types, including 8-bit data types.

If you select `int8` or `uint8` in your model, your simulation runs with 8-bit data, but in the generated code, 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 1-10
“Timer-Based Interrupt Processing” on page 1-10
“Asynchronous Interrupt Processing” on page 1-11

Overview

Normally the code generated by Target Support Package software runs in the context of a timer interrupt. Model blocks run in a periodical fashion clocked by the periodical interrupt whose period is tied to the base sample time of the model.

This execution scheduling model, 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.

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

Timer-Based Interrupt Processing

For code that runs in the context of the timer interrupt, each iteration of the model solver is run after an interrupt has been posted and serviced by an interrupt service routine (ISR). The code generated for the C280x, C281x, and C28x3x 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, F2808, and F28x35) and the CPU clock speed. The CPU clock speed is 100 MHz for the F2808, and 150 MHz for the F2812 and F28335.

If all the blocks in the model inherit their sample time value, and no sample time is explicitly defined, the default value is 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 Support Package software. 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 software 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

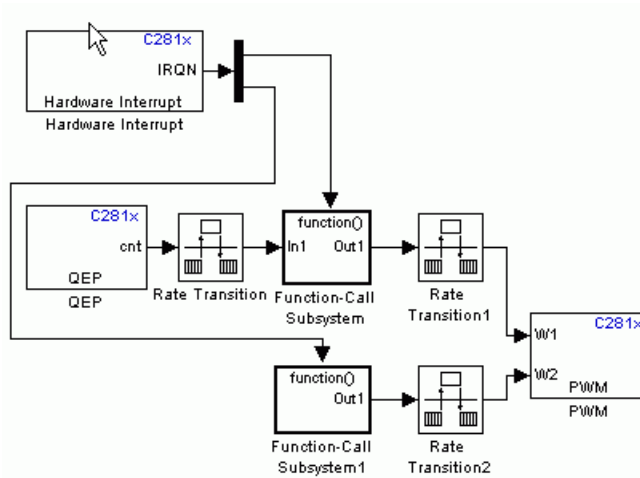
This block enables selected hardware interrupts, generates the corresponding interrupt service routines (ISRs), and connects them to the corresponding interrupt service vector table entries. When you connect the output of the Hardware Interrupt block to the control input of a triggered subsystem (for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

Target Support Package software provides a Hardware Interrupt block for each of the supported processor families.

- Rate Transition blocks

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

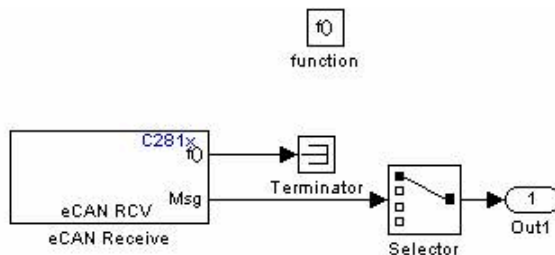
The following diagram illustrates a use case where a Hardware Interrupt block triggers two tasks, connected to other blocks that run periodically in the context of the synchronous scheduler.



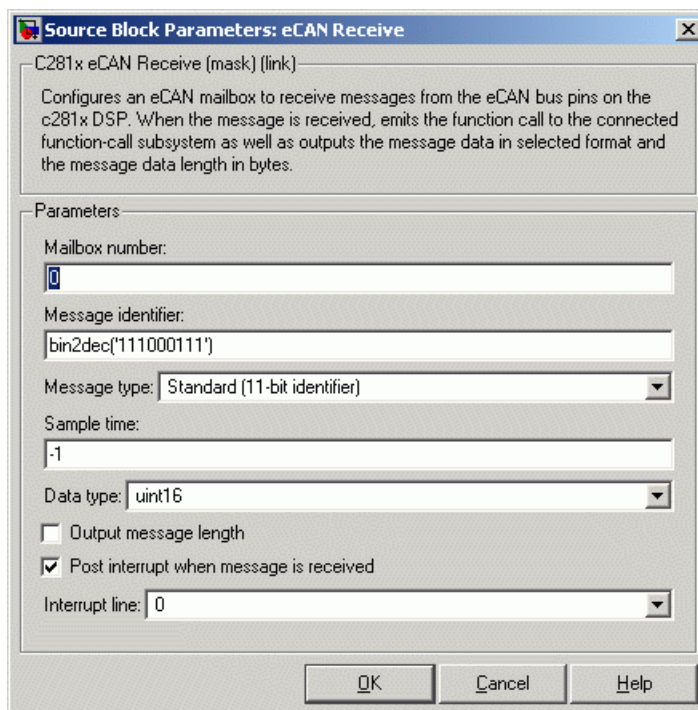
In the preceding figure, the Hardware Interrupt block is set to react on two interrupts. Since only one Hardware Interrupt block is allowed in a model and the output of this block is a vector of length two, you must connect the Hardware Interrupt block to a Demux block to trigger the two function-call subsystems. The function-call 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-call subsystem, as shown in the following figure.

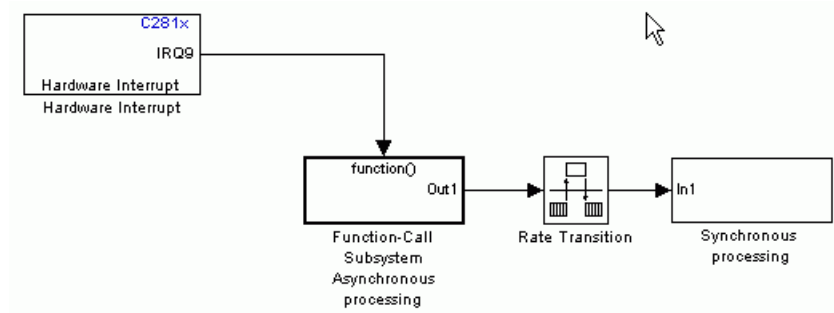


- 2 On the eCAN Receive block dialog, check the box labeled **Post interrupt when message is received**, as shown in the following figure.

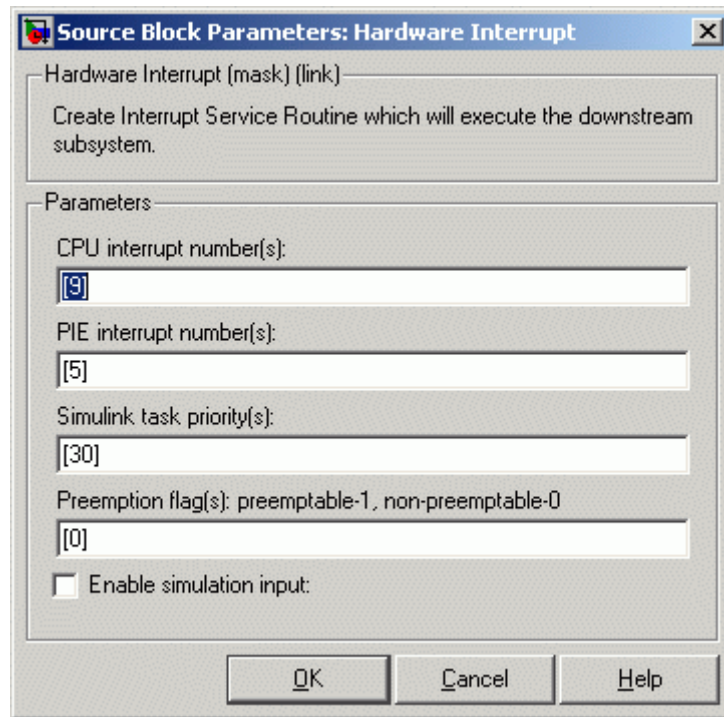


- 3 Set the **Sample Time** of the eCAN Receive block to -1 since the block will be triggered by the ISR, as shown in the preceding figure.

- 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-call subsystem is executed. In this example, the eCAN Receive block is placed in the function-call 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.

Sharing General Purpose Timers between C281x Peripherals

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

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

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

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

GP Timer Use for C281x Peripheral Blocks

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

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

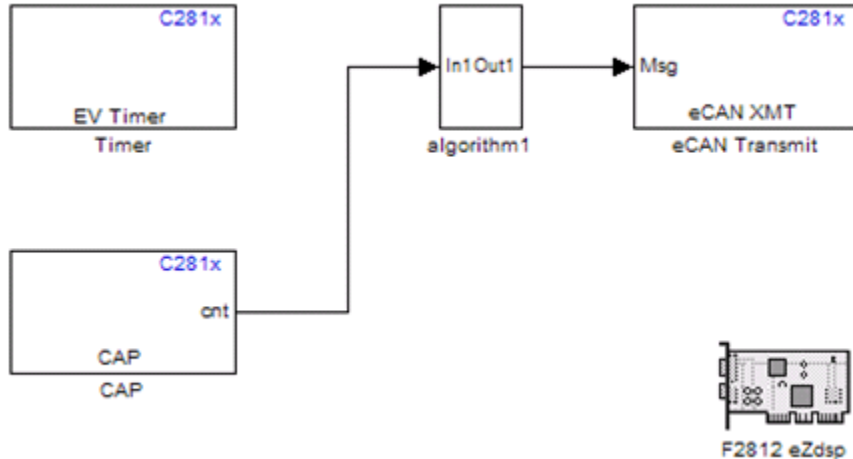
Each GP timer is available to multiple peripherals. For example:

- PWM1-PWM6 and CAP1-CAP3 share GP Timer 1
- PWM7-PWM12 and CAP4-CAP6 share GP Timer 3

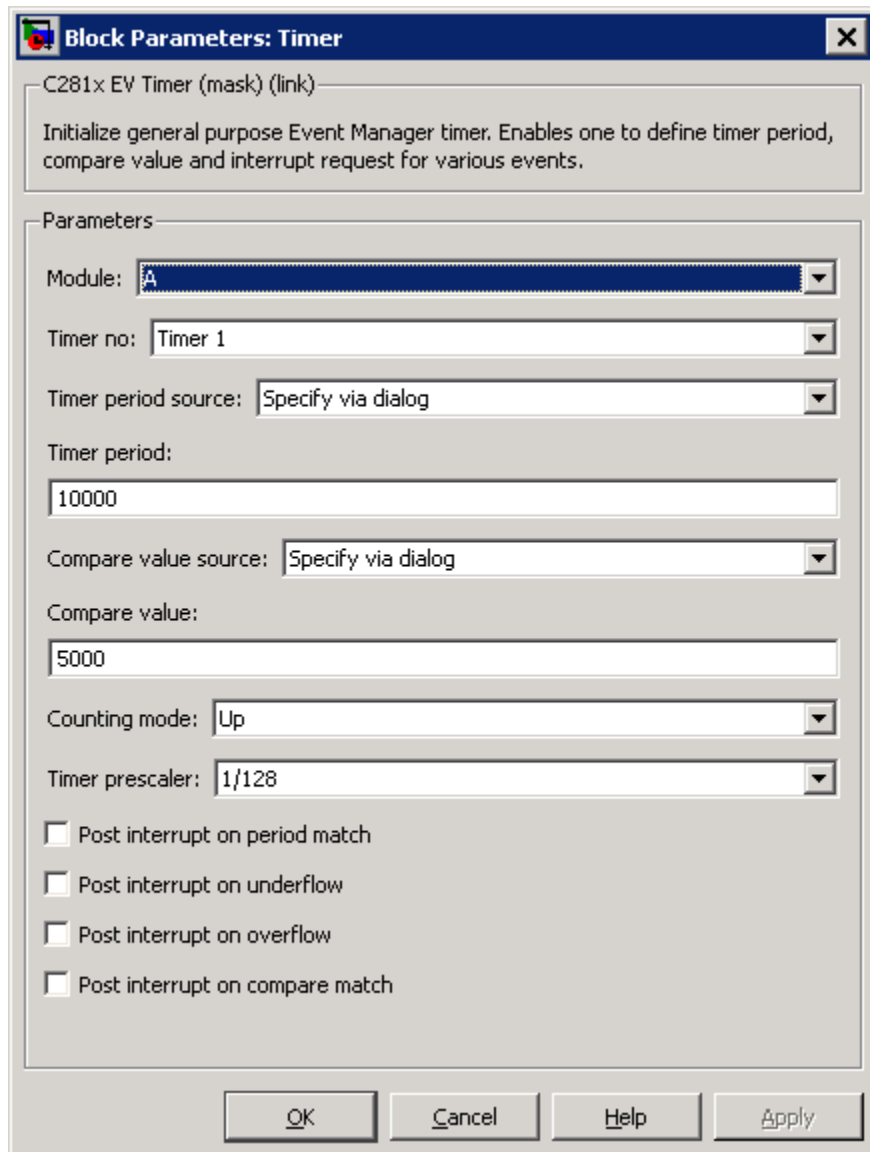
- QEP1-QEP2 and CAP1-CAP3 share GP Timer 2
- QEP3-QEP4 and CAP4-CAP6 share GP Timer 4

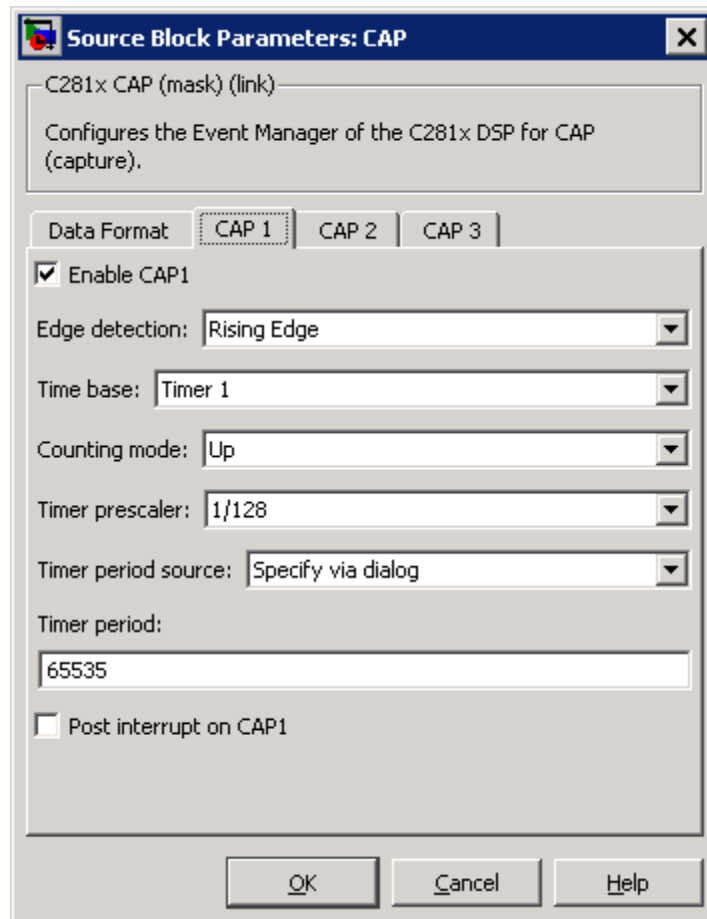
The PWM, QEP, CAP, and Timer blocks each provide independent access to key timer registers. If the blocks in your model share a specific GP timer, ensure that all the timer-related settings are compatible. If the peripheral settings for a shared timer are not compatible, the software generates an error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

Example 1

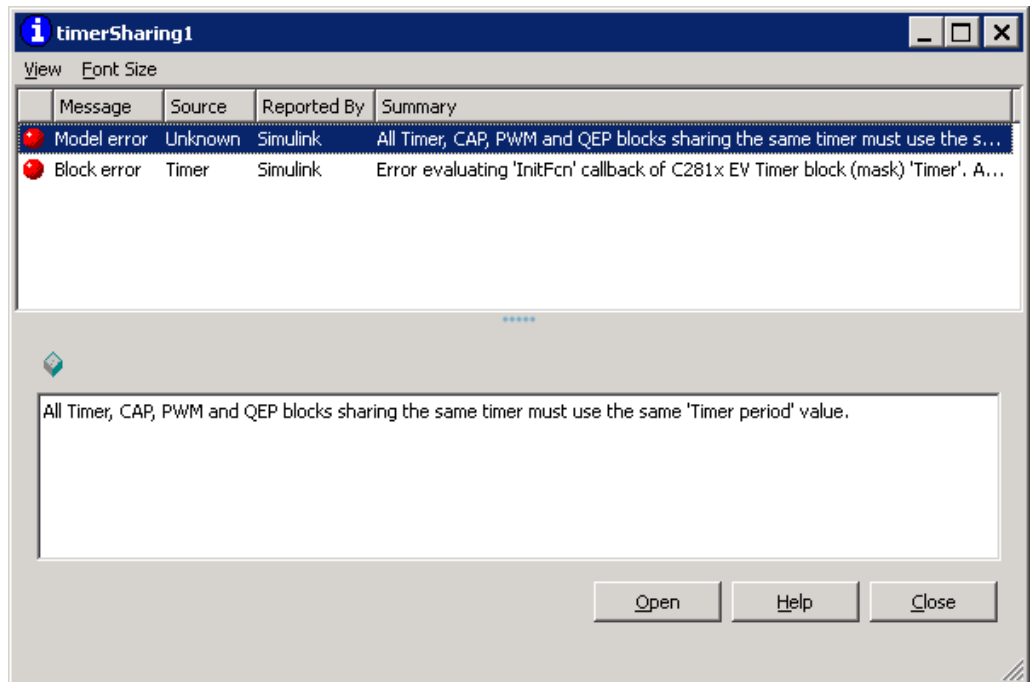


The model contains Timer and CAP blocks that both use Timer 1 (GP Timer 1).



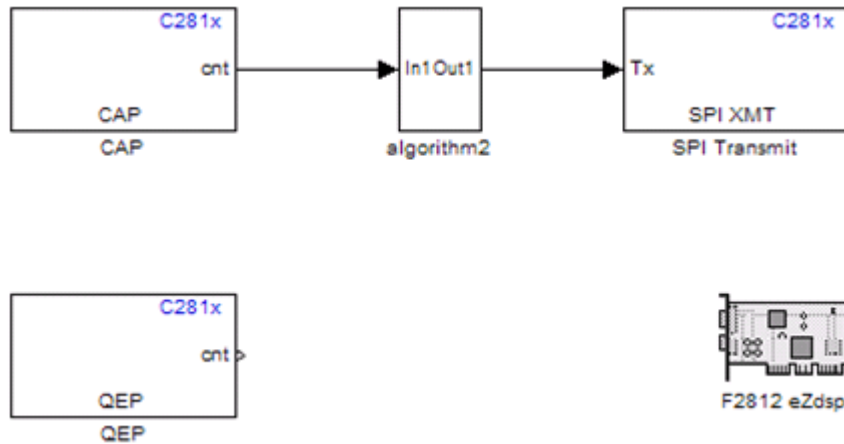


Both blocks have the same values for **Timer prescaler** and **Counting mode**. However, each block has different values for **Timer period**. The value of **Timer period** for Timer 1 is 65535 in the CAP block and 10000 in the Timer block.

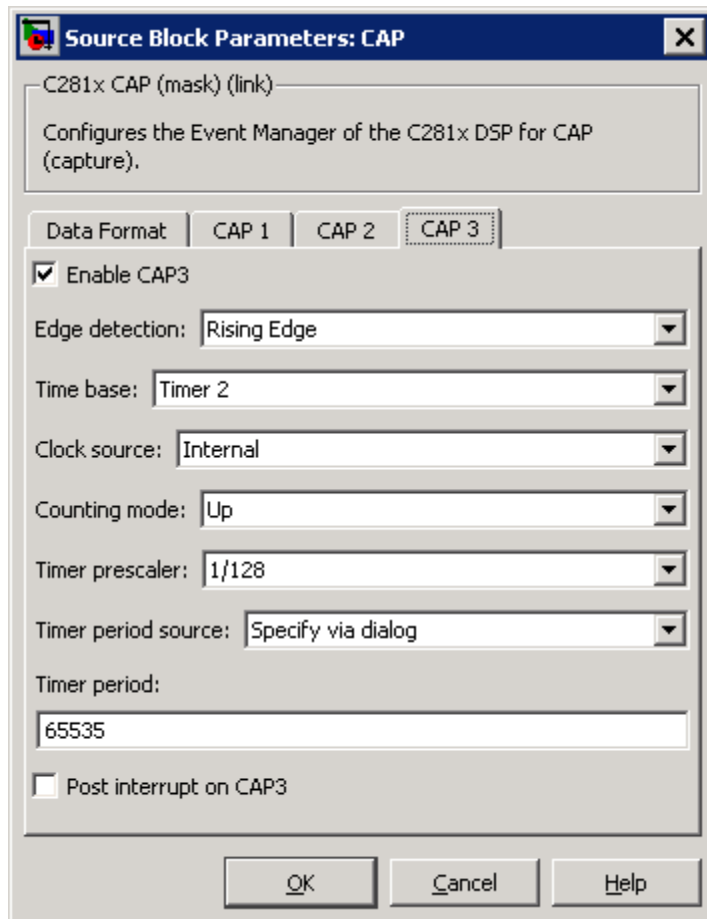


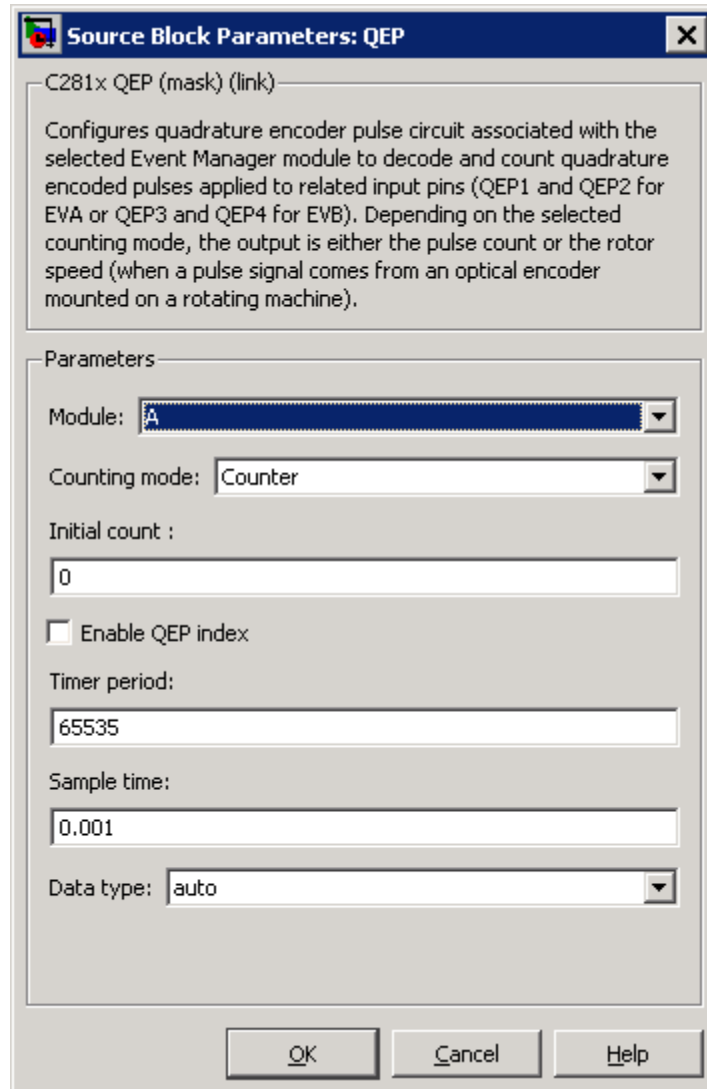
Since both blocks configure the same timer, and settings conflict, the software generates an error when you update the model.

Example 2

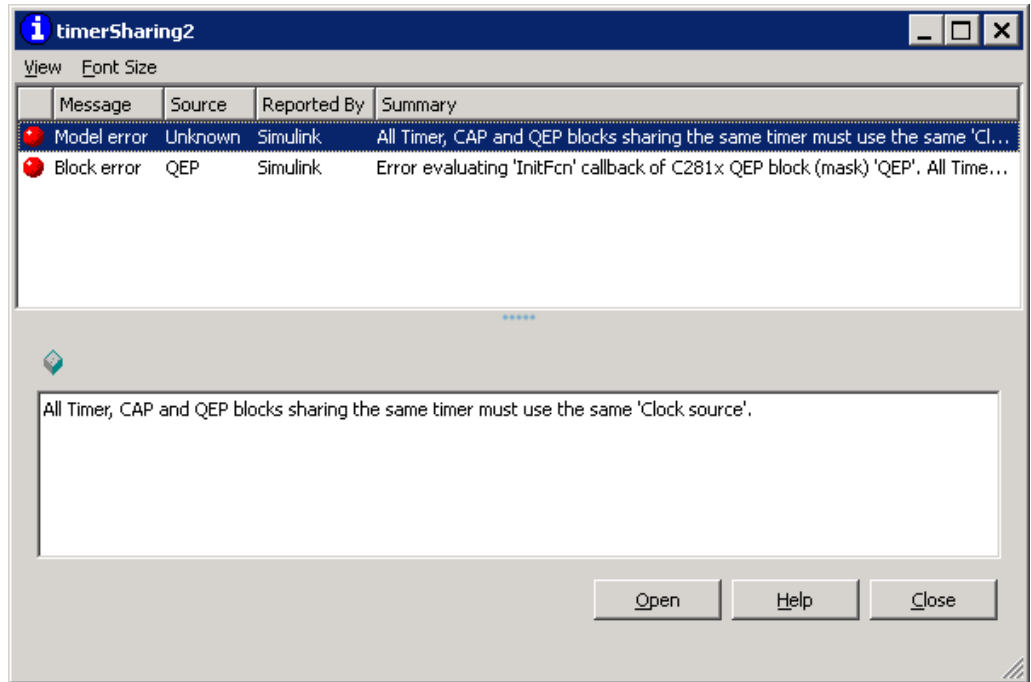


The model contains QEP and CAP blocks that both use Timer 2. In the CAP block, the **Time base** option shows which timer the block uses. In the QEP block, setting **Module** to A configures the block to use QEP1–QEP2. shows that QEP1–QEP2 use Timer 2.





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



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

Overview of Creating Models for Targeting

In this section...

“Accessing the Target Support Package Block Library” on page 1-25

“Online Help” on page 1-26

“Blocks with Restrictions” on page 1-26

“Setting Simulation Configuration Parameters” on page 1-28

“Building Your Model” on page 1-29

Accessing the Target Support Package 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 1-31 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. Select blocks to build your model from the following sources or products:

- The appropriate Target Preferences library block (for setting target and application preferences)
- The appropriate libraries in the `c2000lib` block library (for handling input and output functions for on your target hardware)
- Real-Time Workshop software
- Discrete time blocks from Simulink
- Any other blockset that meets your needs and operates in the discrete time domain

Online Help

To get general help for Target Support Package software, use the help feature in MATLAB. At the command prompt, type

```
help tic2000
```

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

Blocks with Restrictions

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.

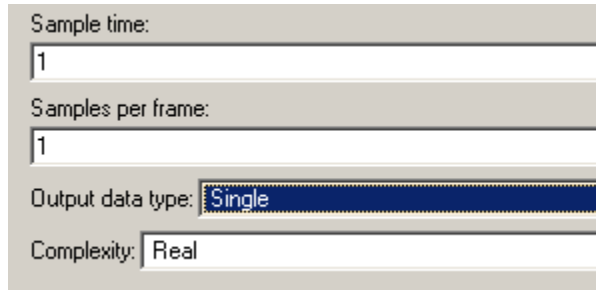
Library	Category	Block Name
Simulink	Sinks	Scope
		To File
		To Workspace
	Sources	From File
		From Workspace

Library	Category	Block Name
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 Output data type supported by the TI C2000 is Single . Be sure to set this parameter correctly in the Block Parameters dialog box. See the following figure.



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 in the **Real-Time Workshop** pane. For Target Support Package software, 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 Support Package software. 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 Support Package software 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 software 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.

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, the TI cross-compiler builds an executable file. After automatically 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, F2808, and F28335 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 1-31
- “Hardware Setup” on page 1-31
- “Starting the c2000lib Library” on page 1-32
- “Setting Up the Model” on page 1-33
- “Adding Blocks to the Model” on page 1-36
- “Generating Code from the Model” on page 1-39

Introduction

This section uses an example to demonstrate how to create a Simulink model that uses Target Support Package 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. 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.

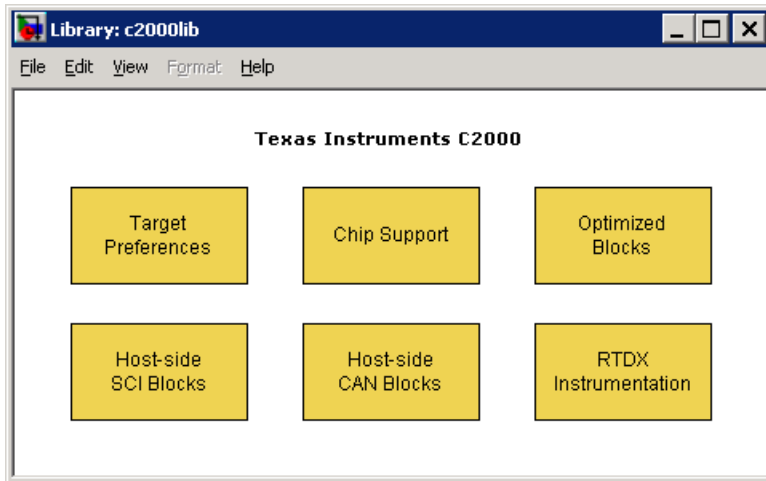
- 3 Connect VREFLO to AGND on the eZdsp F2812. See the section on the Analog Interface in Chapter 2 of the *eZdsp™ F2812 Technical Reference*, available from the Spectrum Digital website at <http://c2000.spectrumdigital.com/ezf2812/>

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 Embedded IDE Link 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 (canmsglib) — Blocks to configure CAN message blocks.
- Host-side SCI Blocks (c2000scilib) — Blocks to configure host-side serial communications interface to send and receive data from serial port

Chip Support

- C281x Chip Support (c281xlib) — Blocks to configure the F2812 eZdsp DSK or on C281x-based custom boards
- C280x Chip Support (c280xlib) — Blocks to configure the F2808 eZdsp DSK or on C280x-based custom boards
- C28x3x Chip Support (c2833xlib) — Blocks to configure the F2833 eZdsp DSK or on C28x3x-based custom boards

Optimized Libraries

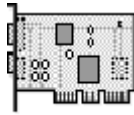
- C28x IQmath Library (tiiqmathlib) — Fixed-point math blocks for use with C28x targets
- C28x DMC Library (c28xdmclib) — Fixed-point math blocks for digital motor control with C28x DSPs

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

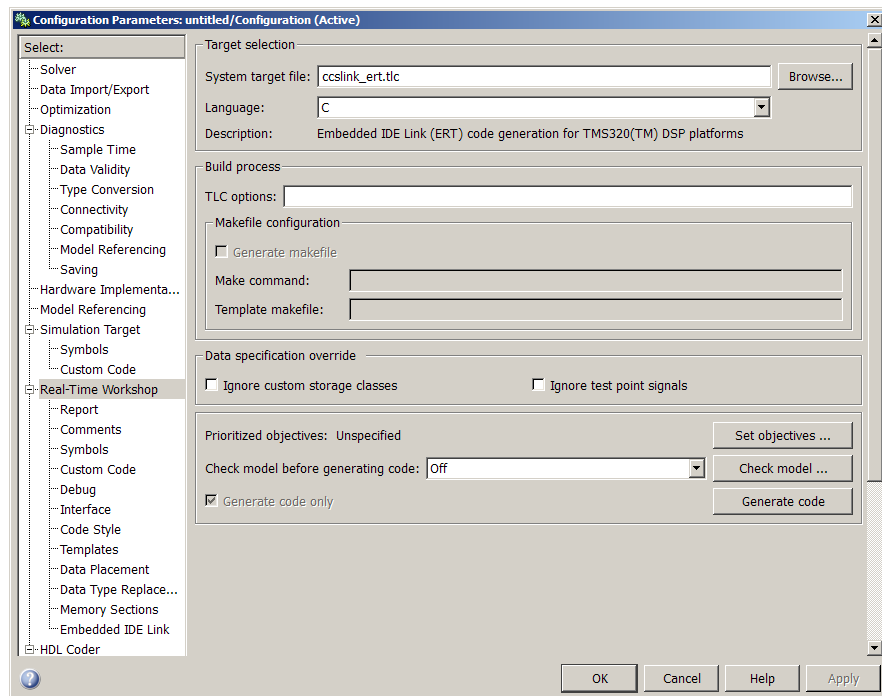
- 4 Click **Yes** to allow automatic setup. The following settings are made, referenced in the table below by their locations in the **Simulation > Configuration Parameters** dialog box:

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.

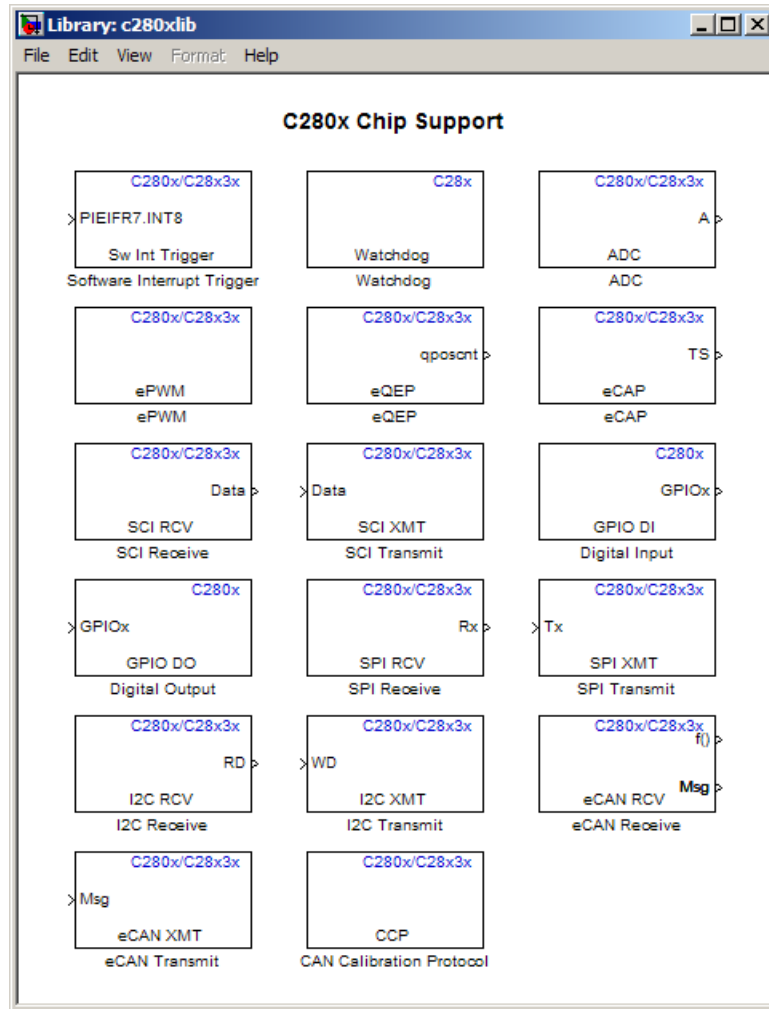


- 7 Use the **Browse** button to locate and select a target configuration file, `ccslink_grt.tlc` or `ccslink_ert.tlc`. When you do this, Real-Time Workshop software chooses the appropriate system target file, and make command.

- 8** Set the configuration parameters by typing **Ctrl-E** and adjust these parameters. For descriptions of these fields, see the Target Preferences/Custom Board reference page and “Setting Simulation Configuration Parameters” on page 1-28 in the section titled “Overview of Creating Models for Targeting” on page 1-25.

Adding Blocks to the Model

- 1** Open or double-click the C281x Chip Support Library, `c281x.lib`.



- 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.

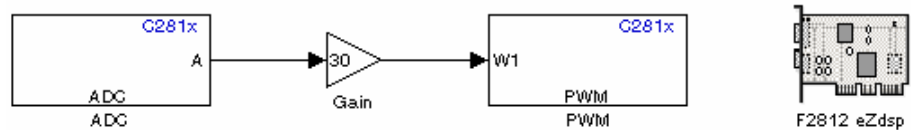
- 3** Drag the C281x PWM block into your model. Double-click the PWM block in the model and set the following parameters. Refer to the C281x PWM reference page for information on these fields.

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

- 4** Enter Simulink 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)
	Integer rounding mode	Floor
Parameter Attributes	Parameter data type mode	Inherit from input

- 5 Connect the ADC block to the Gain block and the Gain block to the PWM block 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 software, 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 folder structure.

Configuring Timing Parameters for CAN Blocks

- “The CAN Blocks” on page 2-2
- “Setting Timing Parameters” on page 2-3
- “Parameter Tuning and Signal Logging” on page 2-9

The CAN Blocks

The bit rate of these four CAN blocks cannot be set directly:

C281x eCAN Receive

C281x eCAN Transmit

C280x/C28x3x eCAN Receive

C280x/C28x3x eCAN Transmit

Setting Timing Parameters

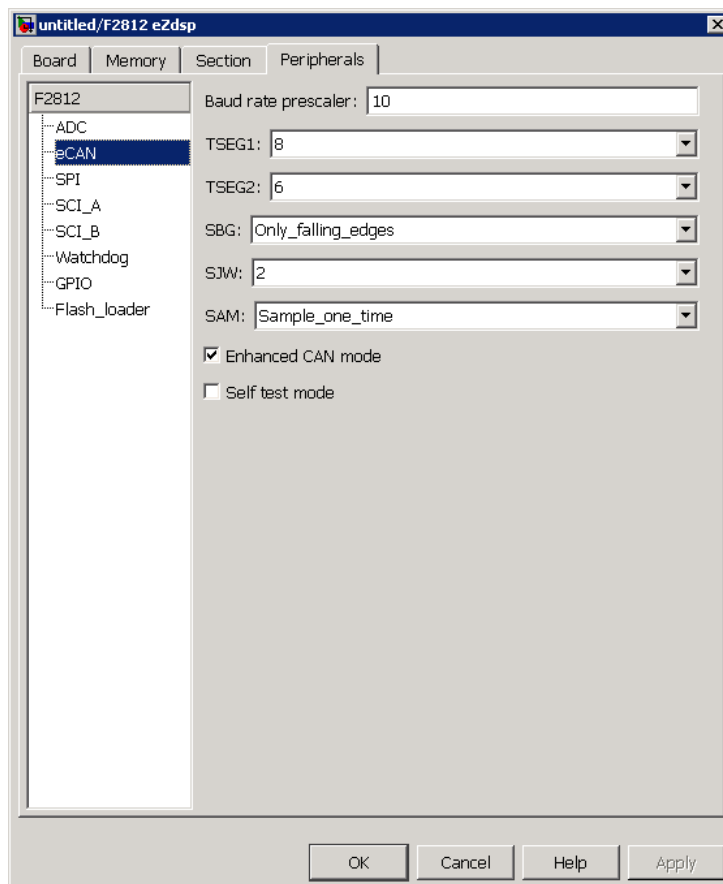
In this section...
“Accessing the Timing Parameters” on page 2-3
“Determining Timing Parameter Values” on page 2-6
“CAN Bit Timing Example” on page 2-7

Accessing the Timing Parameters

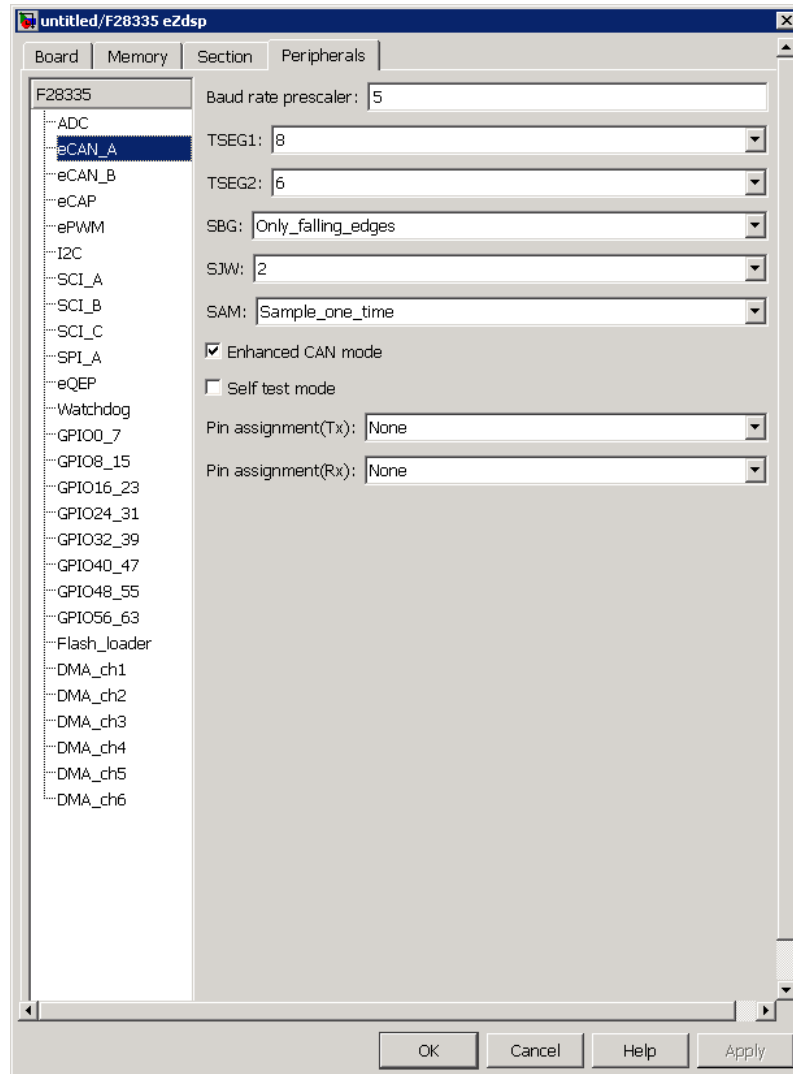
To set the bit rate for “The CAN Blocks”:

- 1 Confirm that your model includes the appropriate Target Preferences block from the C2000 Target Preferences Library.
- 2 Double click the Target Preferences block in your model. This opens the **Target Preferences** dialog box.
- 3 Under the **Peripherals** tab, use the **TSEG1**, **TSEG2**, and **BaudRatePrescaler (BRP)** parameters to set the bit rate.

For example, the **Target Preferences** block for the F2812 eZdsp, this dialog box is shown in the following figure.



The C280x/C28x3x blocks have two independent eCAN modules, 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.

Determining Timing Parameter Values

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

1 Determine the CAN Baudrate specification based on your application.

2 Determine the frequency of the CAN module clock:

- 100 MHz for the F2808. (Same as SYSCLKOUT)
- 150 MHz for the F2812. (Same as SYSCLKOUT)
- 75 MHz for the F2833x. (SYSCLKOUT/2)

3 Estimate the value of the **BaudRatePrescaler (BRP)**.

4 Solve this equation for BitTime:

$$\text{BitTime} = \text{CAN module clock frequency} / (\text{BRP} * \text{Baudrate})$$

5 Solve this equation for Baudrate:

$$\text{Baudrate} = \text{CAN module clock frequency} / (\text{BRP} * \text{BitTime})$$

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

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

7 Use the following rules to determine the values of **TSEG1** and **TSEG2**:

$$\text{TSEG1} \geq \text{TSEG2}$$

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

$$\text{IPT} \leq \text{TSEG1} \leq 16 \text{ TQ}$$

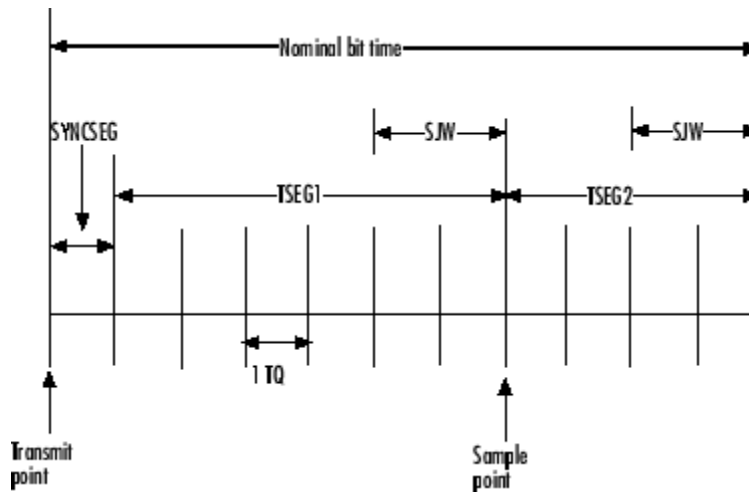
$$\text{IPT} \leq \text{TSEG2} \leq 8 \text{ TQ}$$

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

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

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

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



CAN Bit Timing Example

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

- 1 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 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:

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

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

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

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

4 All chosen values satisfy the criteria, so no further iteration is necessary.

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

Bit Rate	TSEG1	TSEG2	Bit Time	BRP	SJW
.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 typical values of Bit Rate and SYSCLKOUT = 100MHz. This clock frequency is the maximum for the C280x/C28x3x blocks.

Bit Rate	TSEG1	TSEG2	Bit Time	BRP	SJW
.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 Support Package software 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 CAN Calibration Protocol block.

Using External Mode

The Simulink external mode feature 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, download parameter updates, and upload 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 `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 tunable 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:

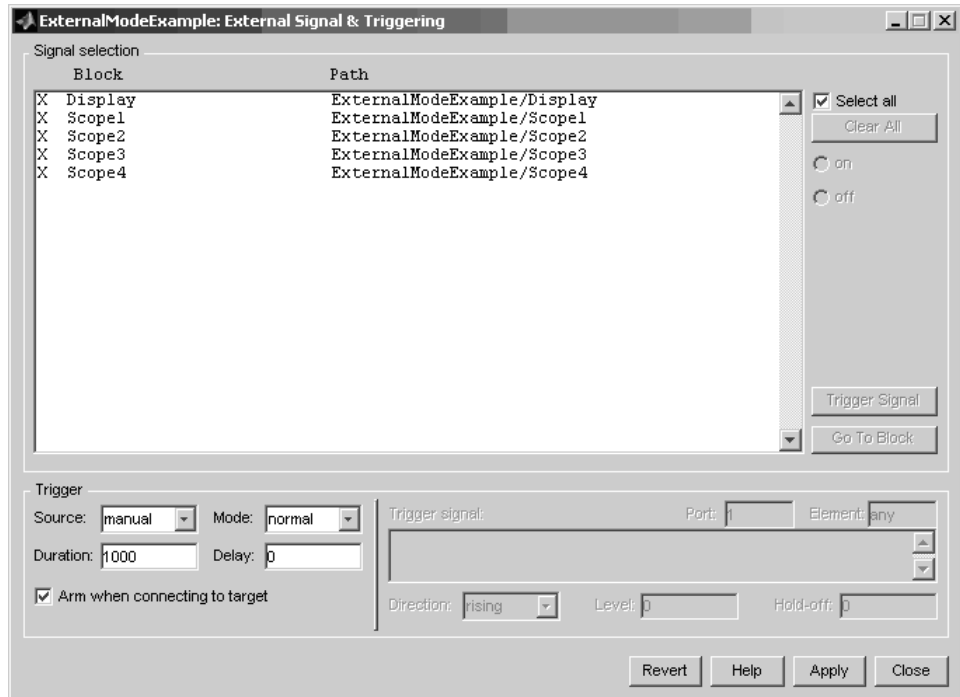
```
stepSize =  
  
Simulink.Parameter (handle)  
    RTWInfo: [1x1 Simulink.ParamRTWInfo]  
    Description: ''  
    DataType: 'uint8'  
    Min: -Inf  
    Max: Inf  
    DocUnits: ''  
    Value: 2  
    Complexity: 'real'  
    Dimensions: [1 1]
```

- 2 Return to your model, and update the model (press **Ctrl+D**) to apply the changed parameter.

Viewing and Storing Signal Data

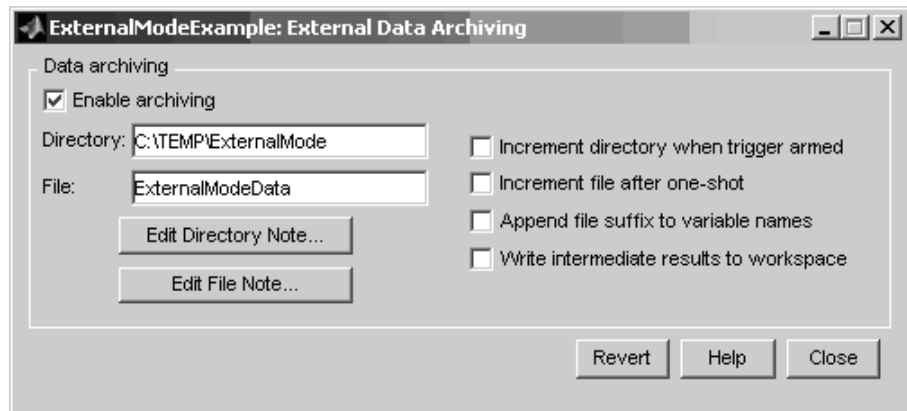
To view the logged signals attach a supported scope type to the signal (see “Limitations” on page 2-17 for supported scope types).

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.

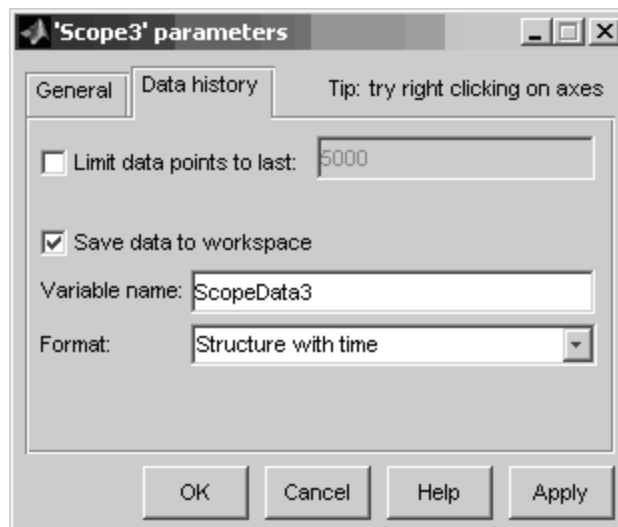


Storing signal data for further analysis. It is possible to store the logged data for further analysis in MATLAB.

- 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 **Folder** 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 folder:

- 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 folder, `.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 sample rates in excess of 10 kHz.

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 Support Package 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 Support Package software lets you export an ASAP2 file containing information about your model during the code generation process.

Before you begin generating ASAP2 files with Target Support Package software, you should read the “Generating an ASAP2 File” 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.
```

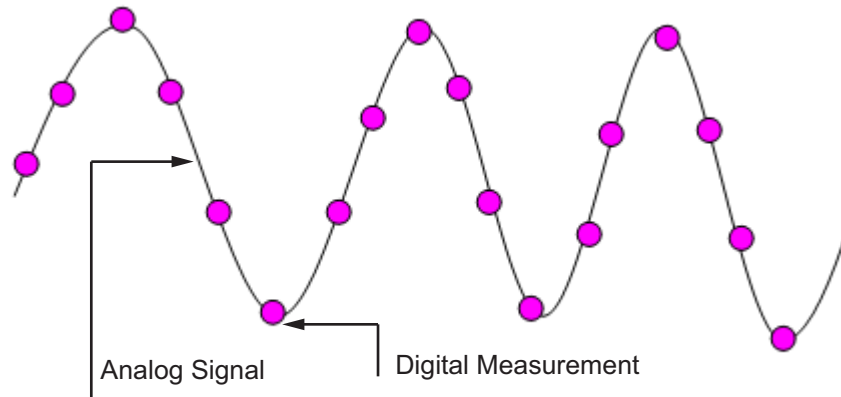
Target Support Package software performs this postprocessing for you. To do this, it first extracts the memory address information from the map file generated during the link process. Secondly, it replaces the placeholders in the ASAP2 file with the actual memory addresses. This postprocessing is performed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

Configuring Acquisition Window Width for ADC Blocks

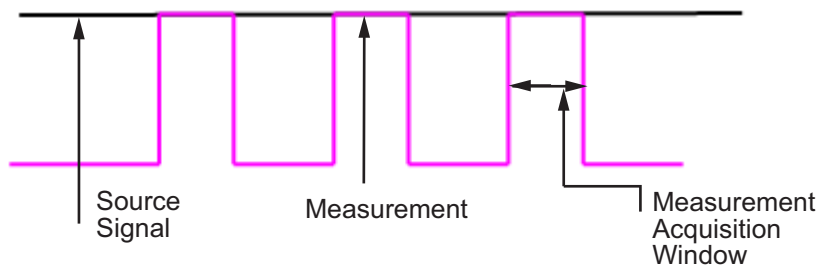
- “What Is an Acquisition Window?” on page 3-2
- “Configuring ADC Parameters for Acquisition Window Width” on page 3-5

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.

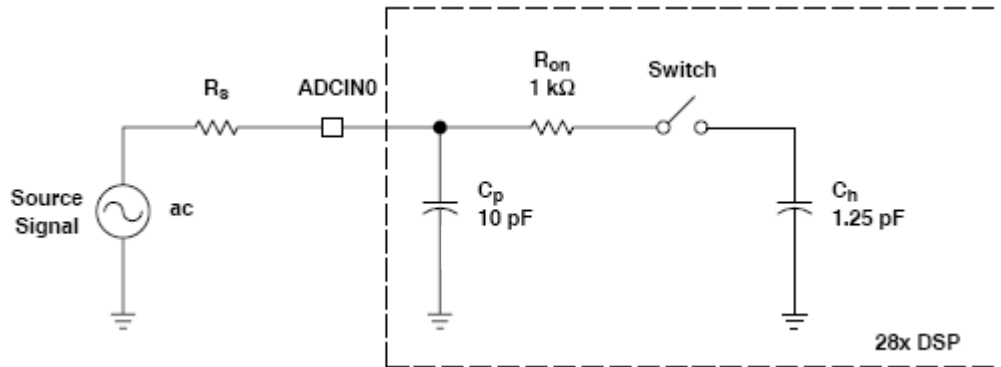


The digital measurement itself is not an instantaneous process, but is a measurement window, where the signal is acquired and measured, as shown below.

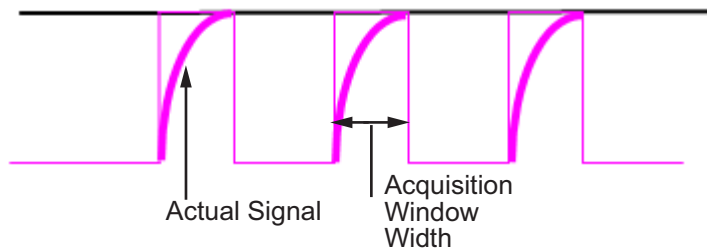


Ideally, as soon as the measurement window is opened, the actual signal coming in would be measured perfectly. In reality the signal does not reach its full magnitude immediately. The measurement process can be modeled by

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

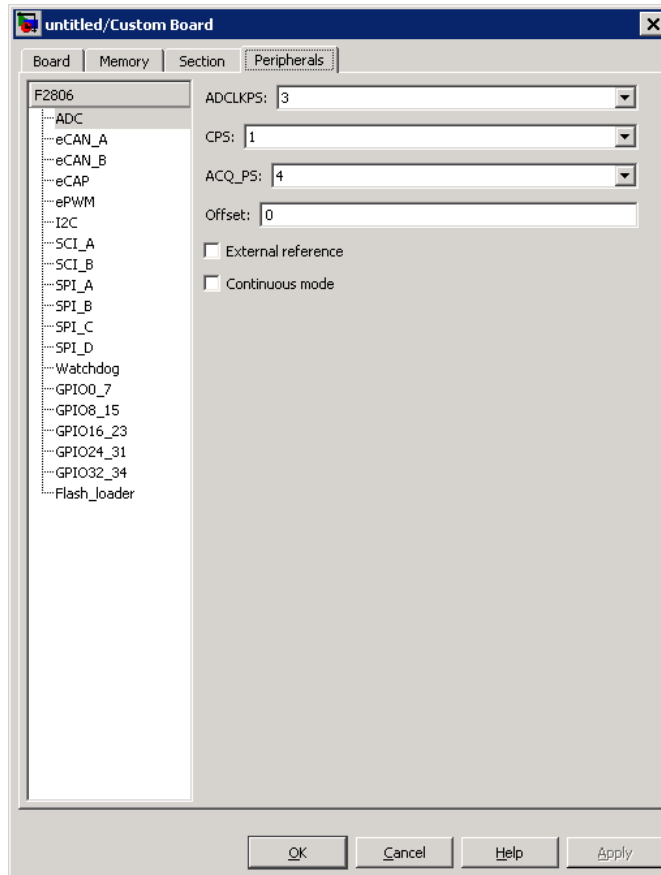
“Examples” on page 3-7

Accessing the ADC Parameters

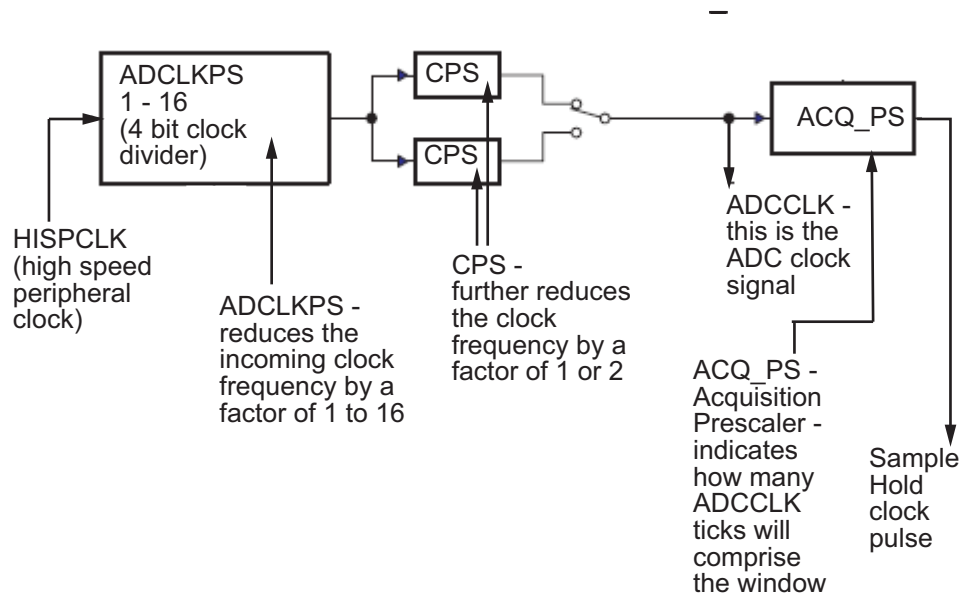
The ADC parameters can be set from the **Peripherals tab** of the Target Preferences block.

- You can set **ACQ_PS** — Acquisition Prescaler — to a value from 0 to 15. To obtain the actual value, increment the setting by 1. This produces an actual range from 1 to 16.
- You can set **ADCLKPS** — AD Clock Prescaler — to a value from 0 to 15. To obtain the actual value, increment the setting by 1. This produces an actual range from 1 to 16.
- You can set **CPS** — Clock Prescaler — to a value from 0 to 1. To obtain the actual value, increment the setting by 1. This produces an actual range from 1 to 2.

3 Configuring Acquisition Window Width for ADC Blocks



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 microseconds.

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 microseconds.

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

Using the IQmath Library

- “About the IQmath Library” on page 4-2
- “Fixed-Point Numbers” on page 4-4
- “Building Models” on page 4-10

About the IQmath Library

In this section...
“Introduction” on page 4-2
“Common Characteristics” on page 4-3
“References” on page 4-3

Introduction

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

Note 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, consult the “Fixed-Point Numbers” on page 4-4 and “Q Format Notation” on page 4-5 topics, as well as the Simulink® Fixed Point™ product documentation, which includes more information on fixed-point data types, scaling, and precision issues.

You can use IQmath Library blocks with some core Simulink blocks and Simulink Fixed Point blocks to run simulations in Simulink models before generating code. Once you develop your model, you can invoke Real-Time Workshop software 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 IQmath Library blocks and blocks from other blocksets, consult “Building Models” on page 4-10.

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.

References

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

Fixed-Point Numbers

In this section...

“Notation” on page 4-4

“Signed Fixed-Point Numbers” on page 4-5

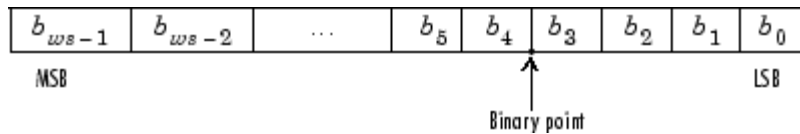
“Q Format Notation” on page 4-5

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.

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

Signed Fixed-Point Numbers

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

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

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

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

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

Q Format Notation

The position of the binary point in a fixed-point number determines how you interpret the scaling of the number. When it performs basic arithmetic such as addition or subtraction, hardware uses the same logic circuits regardless of the value of the scale factor. In essence, the logic circuits 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 software, this data type is expressed as

`sfrac16`

or

`sfix16_En15`

In Filter Design Toolbox™ software, 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 software, this data type is expressed as

`sfix32_En30`

In Filter Design Toolbox software, 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 software, this data type is expressed as

`sfix16_En17`

In Filter Design Toolbox software, 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 software, this data type is expressed as

`sfix16_E2`

In Filter Design Toolbox software, this data type is expressed as

[16 -2]

Building Models

In this section...
“Overview” on page 4-10
“Converting Data Types” on page 4-10
“Using Sources and Sinks” on page 4-11
“Choosing Blocks to Optimize Code” on page 4-11

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, the IQmath Library and Simulink Fixed Point software 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.

Programming Flash Memory

- “Introduction” on page 5-2
- “Installing TI Flash APIs” on page 5-3
- “Configuring the DSP Board Bootloader” on page 5-4
- “Configuring the Software for Automatic Flash Programming” on page 5-5
- “Selectively Erase, Program, or Verify Specific Flash Sectors” on page 5-7
- “Placing Additional Code or Data on Unused Flash Sectors” on page 5-8

Introduction

The Target Support Package software includes a feature for programming Flash memory on the DSP target. You can configure this feature to automatically program Flash memory when you build and execute models for DSP boards. You can also use the Flash programming feature to selectively erase, program, or verify specific sectors of Flash memory.

Note Reprogramming Flash memory thousands of times may deplete its ability to hold data. Consult the manufacturer's documentation for specifications.

Requirements:

- A F2812, F2808, or F28335 eZdsp board
- A working model that includes a target preferences block for “**Stand alone code using Flash Memory**”
- The TI Flash API for your specific target

Installing TI Flash APIs

- 1** Visit the Texas Instruments website and download the TI Flash API installation software for your target:
 - F281x: <http://focus.ti.com/docs/toolsw/folders/print/sprc125.html>
 - F280x: <http://focus.ti.com/docs/toolsw/folders/print/sprc193.html>
 - F2802x: <http://focus.ti.com/docs/toolsw/folders/print/sprc848.html>
 - F2804x: <http://focus.ti.com/docs/toolsw/folders/print/sprc325.html>
 - F2823x: <http://focus.ti.com/docs/toolsw/folders/print/sprc665.html>
 - F2833x: <http://focus.ti.com/docs/toolsw/folders/print/sprc539.html>
- 2** Start the TI Flash API installation software (.exe) contained in the ZIP file.
- 3** During installation, *use the default folder location* for **Location to Save Files**.

Otherwise, each time you create a model, you must configure **Specify API Location**, located under the **Peripherals** tab of the Target Preferences block.
- 4** Complete the installation process.

Configuring the DSP Board Bootloader

Configure the bootloader switch or jumper on the DSP board so that, upon startup, the DSP board executes the program from Flash memory. Consult the manufacturer's hardware documentation to identify the specific switch and settings.

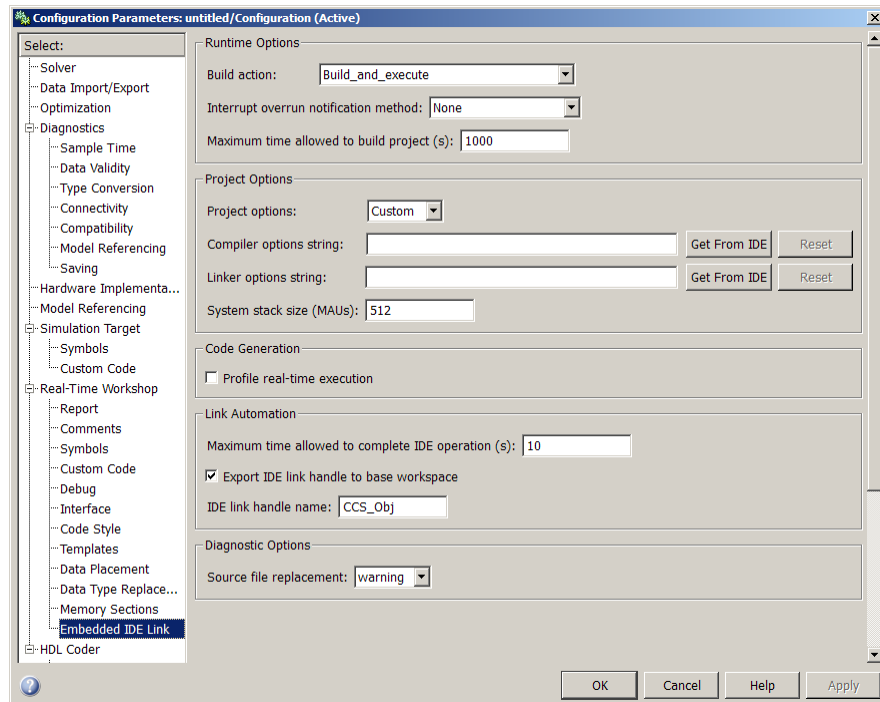
Typically, you can enable the bootloader switch or jumper by moving it from the factory default position (Flash disabled) to the opposite position (enabled). For example:

- On the F2812 eZdsp, change jumper JP7 from the factory default setting.
- On the F2808 eZdsp, change switches 1 and 3 on bank SW1 from the factory default settings.
- On F28335 eZdsp, change switch 3 on bank SW1 from the factory default setting.

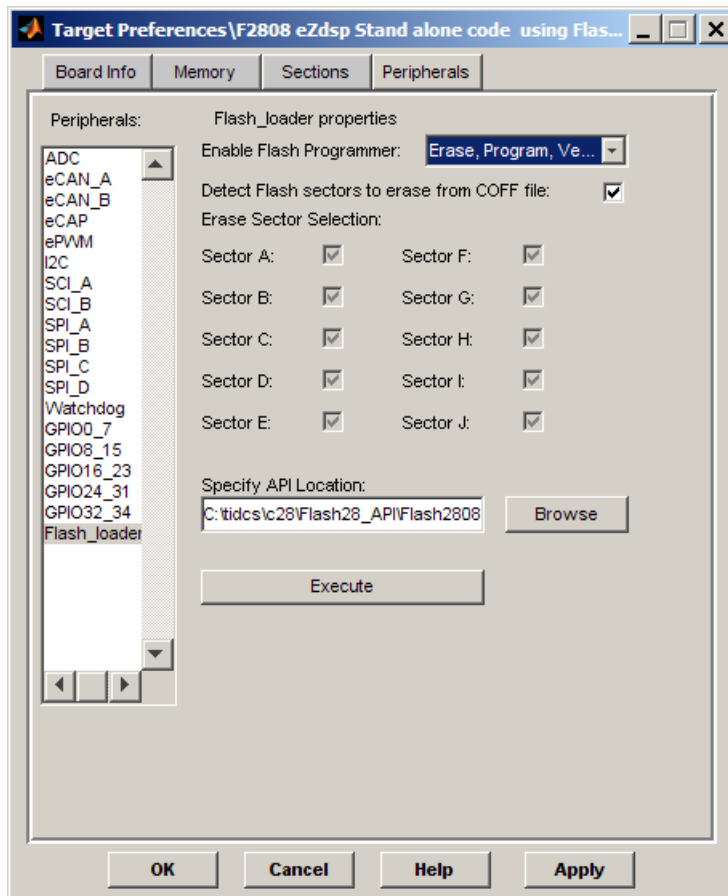
Configuring the Software for Automatic Flash Programming

Configure Target Support Package software to program Flash memory on the target board when you build and execute a model.

- 1 On your keyboard, press Ctrl+E to open the Real-Time Workshop Configuration Parameters dialog box, select *Real Time Workshop* and *Embedded IDE Link*, and confirm **Build Action** is set to **Build_and_execute**.



- 2 Open the target preferences block in your model, select the **Peripherals** tab, and then select **Flash_loader**.
- 3 Set **Enable flash programmer** to **Erase, Program, Verify**.



4 Click **OK** to save and close the new configuration.

When you build the model, the software automatically erases, programs, and verifies Flash memory. When the DSP board restarts, it loads and executes the program from Flash memory.

Selectively Erase, Program, or Verify Specific Flash Sectors

You can manually erase, program, and verify specific sectors of Flash memory:

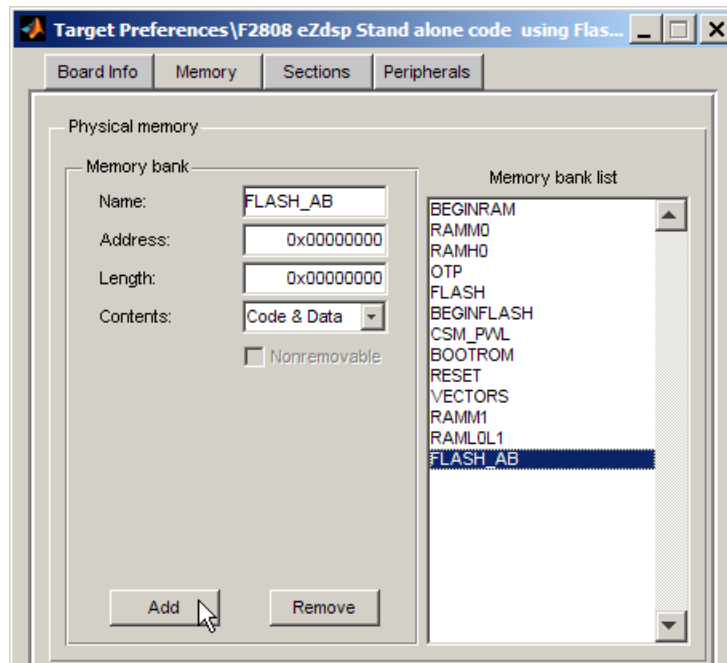
- 1** Open the target preferences block in your model, and select the **Peripherals** tab.
- 2** Select **Flash_loader** from the **Peripherals** list.
- 3** Set **Enable flash programmer** to erase, program, or verify flash.
- 4** (Optional) To protect specific Flash sectors:
 - a** Disable **Detect Flash sectors to erase from COFF file**.
 - b** Deselect the flash sectors you want to protect.
- 5** Click **Execute**. The software performs the action you specified upon the unprotected flash sectors.

Note Erase Flash sectors before programming them.

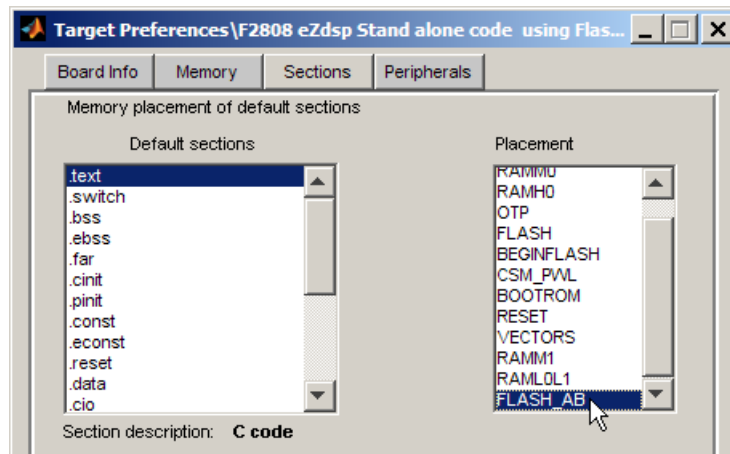
Placing Additional Code or Data on Unused Flash Sectors

To place additional code or data on unused Flash sectors:

- 1 Determine the address and length of the individual Flash sectors. You may need to refer to the manufacturer's specifications.
- 2 Determine the size of the primary C code program and the number of Flash sectors it occupies.
- 3 Determine the size of the additional code or data and the number of Flash sectors it will occupy.
- 4 Under the target preferences **Memory** tab, click **Add** to create two or more new memory banks; one for the primary C code program (e.g., FLASH_AB) and one or more for the additional code or data (e.g., FLASH_CD). The address and length of each memory bank must align with those of the flash sectors.



- 5 Under the **Sections** tab, under **Default sections**, select **.text**. Then, under **Placement**, select the new memory bank (e.g., FLASH_AB) you created for the primary C code program. The next time you program the Flash memory, the software places the **.text** C code file in the new memory bank.



- 6 Similarly, select items from the **Default sections** or **Custom sections list**, and place them in the new memory banks (e.g., FLASH_CD) for the previously unoccupied Flash sectors.

Block Reference

C280x Chip Support (c280xlib) (p. 6-2)	Blocks that support C280x boards
C2802x Chip Support (c2802xlib) (p. 6-4)	Blocks that support C2802x boards
C281x Chip Support (c281xlib) (p. 6-6)	Blocks that support C281x boards
C28x3x Chip Support (c2833xlib) (p. 6-8)	Blocks that support C28x3x boards
C28x DMC (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
CAN Message Handling Blocks (canmsglib) (p. 6-12)	Host CAN blocks
Host Communication (hostcommplib) (p. 6-13)	Blocks for TMS320VC5510 DSP Starter Kit (DSK) (c5510dsk)
Host SCI Blocks (c2000scilib) (p. 6-14)	Host SCI blocks
RTDX Instrumentation (rtDXBlocks) (p. 6-15)	RTDX blocks for C2000 boards
Target Preferences (c2000tgtppreflib) (p. 6-16)	Configure models for code generation and targeting

C280x Chip Support (c280xlib)

C280x/C28x3x ADC	Analog-to-Digital Converter (ADC)
C280x/C28x3x eCAN Receive	Enhanced Control Area Network receive mailbox
C280x/C28x3x eCAN Transmit	Enhanced Control Area Network transmit mailbox
C280x/C28x3x ePWM	Configure Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms
C280x/C28x3x eQEP	Quadrature encoder pulse circuit
C280x/C28x3x/C2802x eCAP	Receive and log capture input pin transitions or configure auxiliary pulse width modulator
C280x/C28x3x/C2802x GPIO Digital Input	Configure general-purpose input pins
C280x/C28x3x/C2802x GPIO Digital Output	Configure general-purpose input/output pins as digital outputs
C280x/C28x3x/C2802x I2C Receive	Configure inter-integrated circuit (I2C) module to receive data from I2C bus
C280x/C28x3x/C2802x I2C Transmit	Configure inter-integrated circuit (I2C) module to transmit data to I2C bus
C280x/C28x3x/C2802x SCI Receive	Receive data on target via serial communications interface (SCI) from host
C280x/C28x3x/C2802x SCI Transmit	Transmit data from target via serial communications interface (SCI) to host
C280x/C28x3x/C2802x Software Interrupt Trigger	Generate software triggered nonmaskable interrupt

C280x/C28x3x/C2802x SPI Receive	Receive data via serial peripheral interface (SPI) on target
C280x/C28x3x/C2802x SPI Transmit	Transmit data via serial peripheral interface (SPI) to host
C28x Watchdog	Configure counter reset source of DSP Watchdog module
CAN Calibration Protocol	Implement CAN Calibration Protocol (CCP) standard
From Memory	Retrieve data from target memory
To Memory	Write data to target memory

C2802x Chip Support (c2802xlib)

C2802x ADC	Configure ADC to sample analog pins and output digital data
C2802x AnalogIO Input	Configure pin, sample time, and data type for analog input
C2802x AnalogIO Output	Configure Analog IO to output analog signals on specific pins
C2802x COMP	Compare two input voltages on comparator pins
C2802x ePWM	Generate Enhanced Pulse Width Modulator (ePWM) waveforms
C280x/C28x3x/C2802x eCAP	Receive and log capture input pin transitions or configure auxiliary pulse width modulator
C280x/C28x3x/C2802x GPIO Digital Input	Configure general-purpose input pins
C280x/C28x3x/C2802x GPIO Digital Output	Configure general-purpose input/output pins as digital outputs
C280x/C28x3x/C2802x I2C Receive	Configure inter-integrated circuit (I2C) module to receive data from I2C bus
C280x/C28x3x/C2802x I2C Transmit	Configure inter-integrated circuit (I2C) module to transmit data to I2C bus
C280x/C28x3x/C2802x SCI Receive	Receive data on target via serial communications interface (SCI) from host
C280x/C28x3x/C2802x SCI Transmit	Transmit data from target via serial communications interface (SCI) to host
C280x/C28x3x/C2802x Software Interrupt Trigger	Generate software triggered nonmaskable interrupt

C280x/C28x3x/C2802x SPI Receive	Receive data via serial peripheral interface (SPI) on target
C280x/C28x3x/C2802x SPI Transmit	Transmit data via serial peripheral interface (SPI) to host
C28x Watchdog	Configure counter reset source of DSP Watchdog module

C281x Chip Support (c281xlib)

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 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 Software Interrupt Trigger	Generate software triggered nonmaskable interrupt
C281x SPI Receive	Receive data via serial peripheral interface on target
C281x SPI Transmit	Transmit data via serial peripheral interface (SPI) to host
C281x Timer	Configure general-purpose timer in Event Manager module
C28x Watchdog	Configure counter reset source of DSP Watchdog module

CAN Calibration Protocol

Implement CAN Calibration Protocol
(CCP) standard

From Memory

Retrieve data from target memory

To Memory

Write data to target memory

C28x3x Chip Support (c2833xlib)

C280x/C28x3x ADC	Analog-to-Digital Converter (ADC)
C280x/C28x3x eCAN Receive	Enhanced Control Area Network receive mailbox
C280x/C28x3x eCAN Transmit	Enhanced Control Area Network transmit mailbox
C280x/C28x3x ePWM	Configure Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms
C280x/C28x3x eQEP	Quadrature encoder pulse circuit
C280x/C28x3x/C2802x eCAP	Receive and log capture input pin transitions or configure auxiliary pulse width modulator
C280x/C28x3x/C2802x GPIO Digital Input	Configure general-purpose input pins
C280x/C28x3x/C2802x GPIO Digital Output	Configure general-purpose input/output pins as digital outputs
C280x/C28x3x/C2802x I2C Receive	Configure inter-integrated circuit (I2C) module to receive data from I2C bus
C280x/C28x3x/C2802x I2C Transmit	Configure inter-integrated circuit (I2C) module to transmit data to I2C bus
C280x/C28x3x/C2802x SCI Receive	Receive data on target via serial communications interface (SCI) from host
C280x/C28x3x/C2802x SCI Transmit	Transmit data from target via serial communications interface (SCI) to host
C280x/C28x3x/C2802x Software Interrupt Trigger	Generate software triggered nonmaskable interrupt

C280x/C28x3x/C2802x SPI Receive	Receive data via serial peripheral interface (SPI) on target
C280x/C28x3x/C2802x SPI Transmit	Transmit data via serial peripheral interface (SPI) to host
C28x Watchdog	Configure counter reset source of DSP Watchdog module
CAN Calibration Protocol	Implement CAN Calibration Protocol (CCP) standard
From Memory	Retrieve data from target memory

C28x DMC (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	Calculate 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

CAN Message Handling Blocks (canmsglib)

CAN Pack

Pack individual signals into CAN message

CAN Unpack

Unpack individual signals from CAN messages

Host Communication (hostcommlib)

Byte Pack	Convert input signals to uint8 vector
Byte Reversal	Reverse order of bytes in input word
Byte Unpack	Unpack UDP uint8 input vector into Simulink data type values
UDP Receive	Receive uint8 vector as UDP message
UDP Send	Send UDP message

Host 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

RTDX Instrumentation (rtdxBlocks)

From RTDX

Add RTDX™ communication channel for target to receive data from host

To RTDX

Add RTDX communication channel to send data from target to host

Target Preferences (c2000tgtplib)

For information about any of the Target Preferences/Custom Board blocks for Texas Instruments' processors, see the following topic:

Target Preferences/Custom Board	Configure model for Texas Instruments processor.
---------------------------------	--

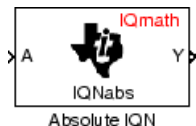
Blocks — Alphabetical List

Absolute IQN

Purpose Absolute value

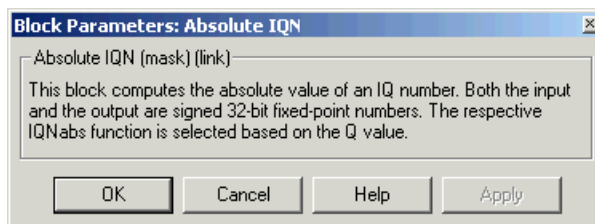
Library “C28x IQmath (tiiqmathlib)” on page 6-11

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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



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

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

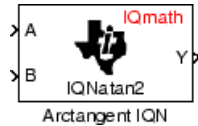
Purpose

Four-quadrant arc tangent

Library

“C28x IQmath (tiiqmathlib)” on page 6-11

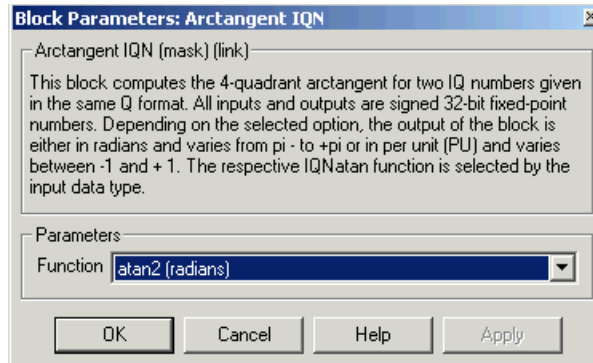
Description



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 Chapter 4, “Using the IQmath Library” 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.

References

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

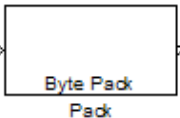
See Also

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

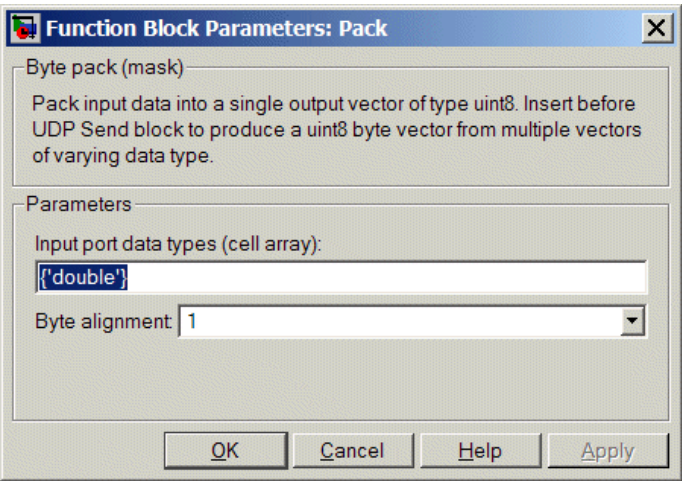
Purpose Convert input signals to uint8 vector

Library Host Communication (hostcomm.lib)

Description Using the input port, the block converts data of one or more data types into a single uint8 vector for output. With the options available, you specify the input data types and the alignment of the data in the output vector. Because UDP messages are in uint8 data format, use this block before a UDP Send block to format the data for transmission using the UDP protocol.



Dialog Box



Input port data types (cell array) Specify the data types for the different signals as part of the parameters. The block supports all Simulink data types except characters. Enter the data types as Simulink types in the cell array, such as 'double' or 'int32'. The order of the data type entries in the cell array must match the order in which the data arrives at the block input. This block determines the signal sizes automatically. The block always has at least one input port and only one output port.

Byte Pack

Byte alignment

This option specifies how to align the data types to form the `uint8` output vector. Select one of the values in bytes from the list.

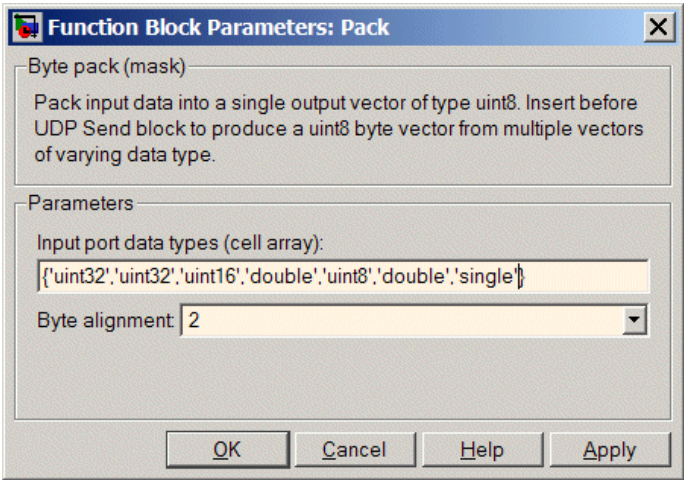
Alignment can occur on 1, 2, 4, or 8-byte boundaries depending on the value you choose. The value defaults to 1. Given the alignment value, each signal data value begins on multiples of the alignment value. The alignment algorithm ensures that each element in the output vector begins on a byte boundary specified by the alignment value. Byte alignment sets the boundaries relative to the starting point of the vector.

Selecting 1 for **Byte alignment** provides the tightest packing, with no holes between any data types for any combination of data types and signals.

Sometimes, you can have multiple data types of varying lengths. In such cases, specifying a 2-byte alignment can produce 1-byte gaps between `uint8` or `int8` values and another data type. In the `pack` implementation, the block copies data to the output data buffer 1 byte at a time. You can specify any of the data alignment options with any of the data types.

Example

Use a cell array to enter input data types in the **Input port data types** parameter. The order of the data types you enter must match the order of the data types at the block input.



In the cell array, you provide the order in which the block expects to receive data—`uint32`, `uint32`, `uint16`, `double`, `uint8`, `double`, and `single`. With this information, the block automatically provides the proper number of input ports.

Byte alignment equal to 2 specifies that each new value begins 2 bytes from the previous data boundary.

The example shows the following data types:

```
{'uint32','uint32','uint16','double','uint8','double','single'}
```

When the signals are scalar values (no matrices or vectors in this example), the first signal value in the vector starts at 0 bytes. Then, the second signal value starts at 2 bytes, and the third at 4 bytes. Next, the fourth signal value follows at 6 bytes, the fifth at 8 bytes, the sixth at 10 bytes, and the seventh at 12 bytes. As the example shows, the packing algorithm leaves a 1-byte gap between the `uint8` data value and the `double` value.

See Also

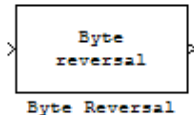
Byte Reversal, Byte Unpack

Byte Reversal

Purpose Reverse order of bytes in input word

Library Host Communication (hostcommlib)

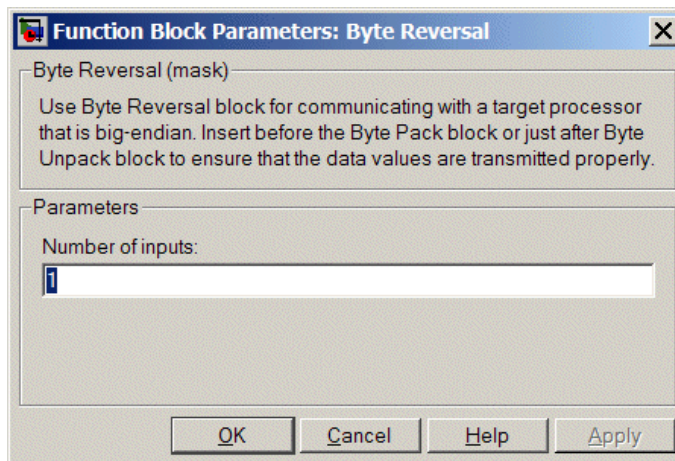
Description



Byte reversal changes the order of the bytes in data you input to the block. Use this block when your process communicates between targets that use different endianness, such as between Intel® processors that are little endian and others that are big endian. Texas Instruments processors are little-endian by default.

To exchange data with a processor that has different endianness, place a Byte Reversal block just before the send block and immediately after the receive block.

Dialog Box



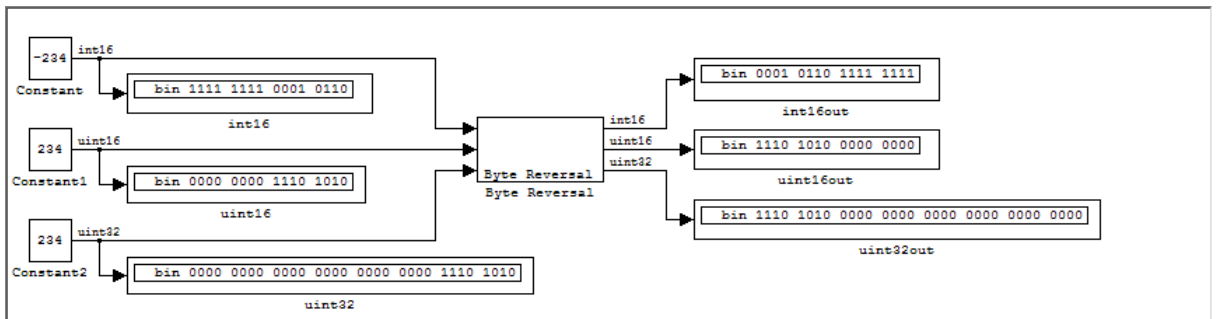
Number of inputs

Specify the number of input ports for the block. The number of input ports adjusts automatically to match value so the number of outputs equals the number of inputs.

When you use more than one input port, each input port maps to the matching output port. Data entering input port 1 leaves through output port 1, and so on.

Reversing the bytes does not change the data type. Input and output retain matching data type.

The following model shows byte reversal in use. In this figure, the input and output ports match for each path.



See Also

Byte Pack, Byte Unpack

Byte Unpack

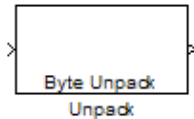
Purpose

Unpack UDP uint8 input vector into Simulink data type values

Library

Host Communication (hostcommlib)

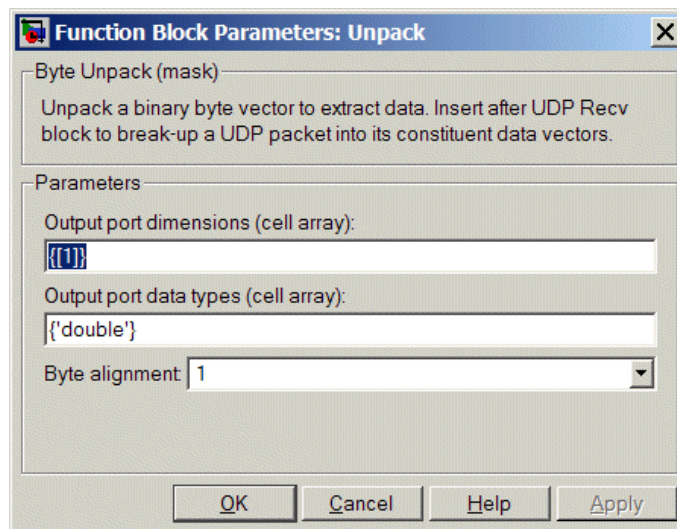
Description



Byte Unpack is the inverse of the Byte Pack block. It takes a UDP message from a UDP receive block as a uint8 vector, and outputs Simulink data types in various sizes depending on the input vector.

The block supports all Simulink data types.

Dialog Box



Output port dimensions (cell array)

Containing a cell array, each element in the array specifies the dimension that the MATLAB size function returns for the corresponding signal. Usually you use the same dimensions as you set for the corresponding Byte Pack block in the model. Entering one value means that the block applies that dimension to all data types.

Output port data types (cell array)

Specify the data types for the different input signals to the Pack block. The block supports all Simulink data types—single, double, int8, uint8, int16, uint16, int32, and uint32, and Boolean. The entry here is the same as the Input port data types parameter in the Byte Pack block in the model. You can enter one data type and the block applies that type to all output ports.

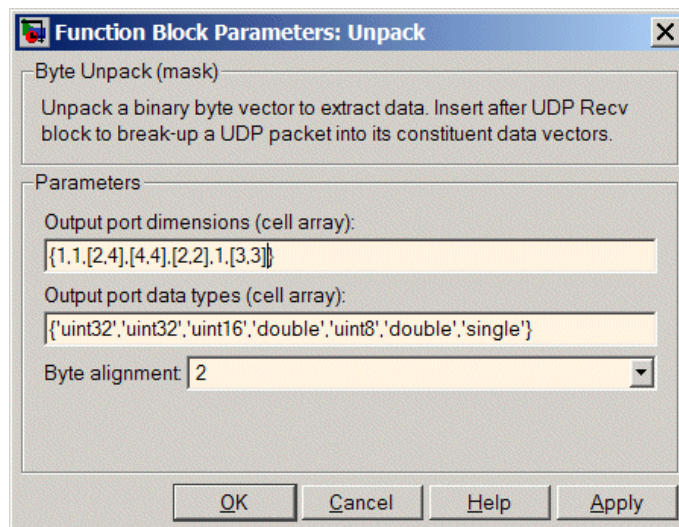
Byte Alignment

This option specifies how to align the data types to form the input uint8 vector. Match this setting with the corresponding Byte Pack block alignment value of 1, 2, 4, or 8 bytes.

Example

This figure shows the Byte Unpack block that corresponds to the example in the Byte Pack example. The **Output port data types (cell array)** entry shown is the same as the **Input port data types (cell array)** entry in the Byte Pack block

```
{'uint32','uint32','uint16','double','uint8','double','single'}.
```



Byte Unpack

In addition, the **Byte alignment** setting matches as well. **Output port dimensions (cell array)** now includes scalar values and matrices to demonstrate entering nonscalar values. The example for the Byte Pack block assumed only scalar inputs.

See Also

Byte Pack, Byte Reversal

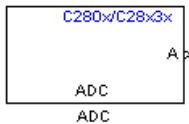
Purpose

Analog-to-Digital Converter (ADC)

Library

“C280x Chip Support (c280xlib)” on page 6-2 and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



The C280x/C28x3x ADC block configures the C280x/C28x3x 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. With the C28x3x, you can configure the ADC to use the processor’s DMA module to move data directly to memory without using the CPU. This frees the CPU to perform other tasks and increases overall system performance.

Output

The output of the C280x/C28x3x ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the C280x/C28x3x ADC is 12-bit converter.

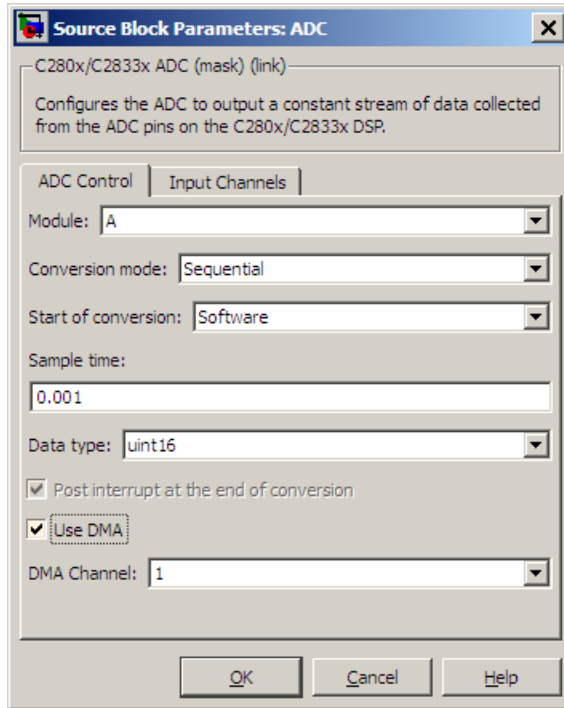
Modes

The C280x/C28x3x 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.

C280x/C28x3x ADC

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 1-10 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 1-11 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 C280x/C28x3x 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. 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 1-11 for a discussion of block placement and other necessary settings.

Use DMA (with C28x3x)

Enable the Direct Memory Access (DMA) to transfer data directly from the ADC to memory, bypassing the CPU and improving overall system performance. This feature is only valid with a C28x3x target.

When enabled, this setting applies the following settings to the channel specified by the **DMA Channel** parameter. *Disable* the corresponding channel in the **Target Preferences block > Peripherals > DMA_ch#**. Modifications to **Target Preferences block > Peripherals > DMA_ch#** do not apply or override the following settings:

- **Enable DMA channel:** Enabled for channel specified by the ADC block **DMA Channel** parameter.
- **Data size:** 16 bit
- **Interrupt source:** If the ADC block **Module** is A or A and B, **Interrupt source** is SEQ1INT. If the ADC block **Module** is B, **Interrupt source** is SEQ2INT.

- **Generate interrupt:** Generate interrupt at end of transfer
- **Size**
 - **Burst:** The value assigned to **Burst** equals the ADC block **Number of conversions** (NOC) multiplied by a value for the ADC block **Conversion mode** (CVM). To summarize, **Burst** = NOC * CVM.

If **Conversion mode** is Sequential, CVM = 1. If **Conversion mode** is Simultaneous, CVM = 2.

For example, **Burst** is 6 when NOC is 3 and CVM is 2.
 - **Transfer:** 1
 - **SRC wrap:** 65536
 - **DST wrap:** 65536
- **Source**
 - **Begin address:** The value of **Begin address** is 0xB00 if the ADC block **Module** is A or A and B. The value of **Begin address** is 0xB08 if the ADC block **Module** is B.
 - **Burst step:** 1
 - **Transfer step:** 0
 - **Wrap step:** 0
- **Destination**
 - **Begin address:** The value of **Begin address** is the ADC buffer address minus the ADC block **Number of conversions**.

If the target is F28232 or F28332, the ADC buffer address is 0xDFFC (57340). For other C28x3x targets, the ADC buffer address is 0xFFFF (65532).

For example, with a F28232 target, the **Begin address** is 0xDFF9 (57337) because the ADC buffer address, 57340 (0xDFFC), minus 3 conversions equals 57337 (0xDFF9).

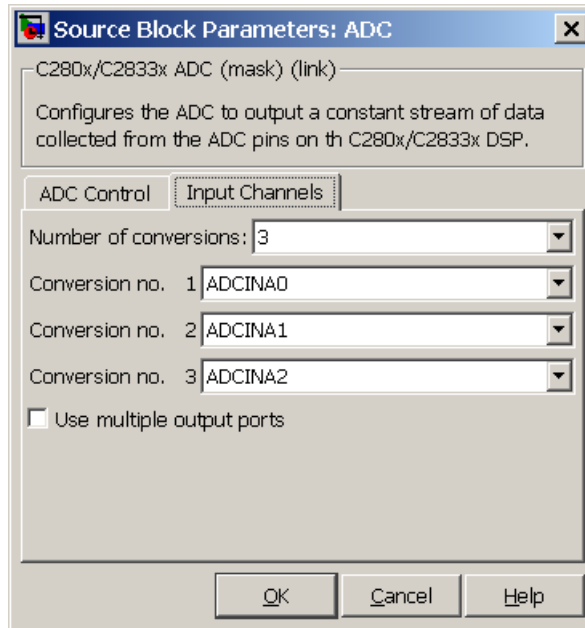
- **Burst step:** 1
- **Transfer step:** 1
- **Wrap step:** 0
- **Mode**
 - **Enable one shot mode:** disabled
 - **Sync enable:** disabled
 - **Enable continuous mode:** enabled
 - **Enable DST sync mode:** disabled
 - **Set channel 1 to highest priority:** disabled
 - **Enable overflow interrupt:** disabled

For more information, consult *TMS320x2833x, 2823x Direct Memory Access (DMA) Module Reference Guide, Literature Number: SPRUFB8A*, available at the Texas Instruments Web site.

DMA Channel

When the **Use DMA** parameter is enabled, select a channel for the DMA module to use for data transfers. To prevent channel conflicts, the same channel number must remain disabled in the F28335 Target Preferences block, otherwise the software will generate an error message.

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

C280x/C28x3x ADC

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/C28x3x ePWM, C280x/C2802x/C2803x/C28x3x Hardware Interrupt, “Configuring Acquisition Window Width for ADC Blocks”

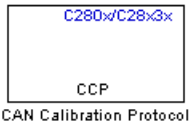
Purpose

Implement CAN Calibration Protocol (CCP) standard

Library

“C280x Chip Support (c280xlib)” on page 6-2, “C281x Chip Support (c281xlib)” on page 6-6, and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



The CAN Calibration Protocol 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 “Compatibility with Calibration Packages” on page 7-26) 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

CAN Calibration Protocol

- Set up signals to be transmitted using CCP DAQ lists.
- Assign signals in your model to a CCP event channel automatically (see “Generating an ASAP2 File”).

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.

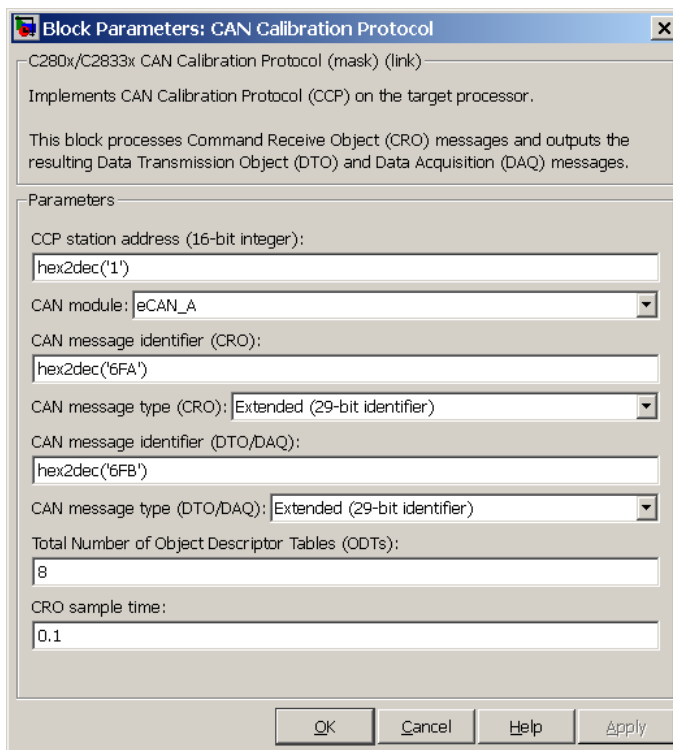
The calibration tool (see “Compatibility with Calibration Packages” on page 7-26) must use CCP commands to assign an event channel and data to the available DAQ lists, and interpret the synchronous response.

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 Support Package software does not currently support event channel prescalers.

Dialog Box



Block Parameters: CAN Calibration Protocol

C280x/C2833x CAN Calibration Protocol (mask) (link)

Implements CAN Calibration Protocol (CCP) on the target processor.

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

Parameters

CCP station address (16-bit integer):
hex2dec('1')

CAN module: eCAN_A

CAN message identifier (CRO):
hex2dec('6FA')

CAN message type (CRO): Extended (29-bit identifier)

CAN message identifier (DTO/DAQ):
hex2dec('6FB')

CAN message type (DTO/DAQ): Extended (29-bit identifier)

Total Number of Object Descriptor Tables (ODTs):
8

CRO sample time:
0.1

OK Cancel Help Apply

CCP 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 module

Choose module `eCAN_A` or `eCAN_B`.

CAN message identifier (CRO)

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

CAN Calibration Protocol

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 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 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
- START_STOP
- START_STOP_ALL
- TEST
- UPLOAD

CAN Calibration Protocol

- WRITE_DAQ

Compatibility with Calibration Packages

The above commands support

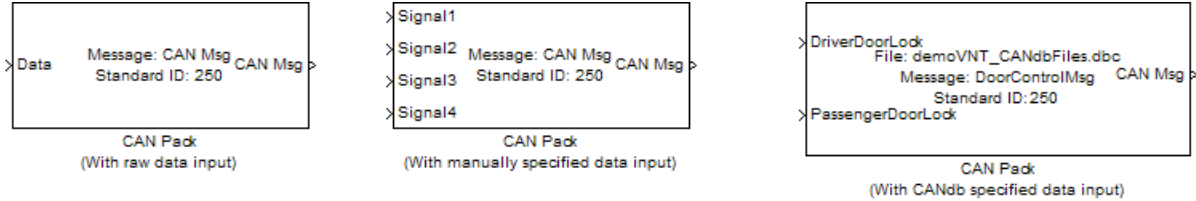
- Synchronous signal monitoring via calibration packages that use DAQ lists
- Asynchronous signal monitoring via calibration packages that poll the target
- Asynchronous parameter tuning via CCP memory programming

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. (Accurate Technologies, Inc., Vision does not support the polling mechanism for signal monitoring).

Purpose Pack individual signals into CAN message

Library CAN Communication

Description



The CAN Pack block loads signal data into a message at specified intervals during the simulation.

Note To use this block, you also need a license for Simulink software.

CAN Pack block has one input port by default. The number of input ports is dynamic and depends on the number of signals you specify for the block. For example, if your block has four signals, it has four input ports.



This block has one output port, CAN Msg. The CAN Pack block takes the specified input parameters and packs the signals into a message.

Other Supported Features

The CAN Pack block supports:

CAN Pack

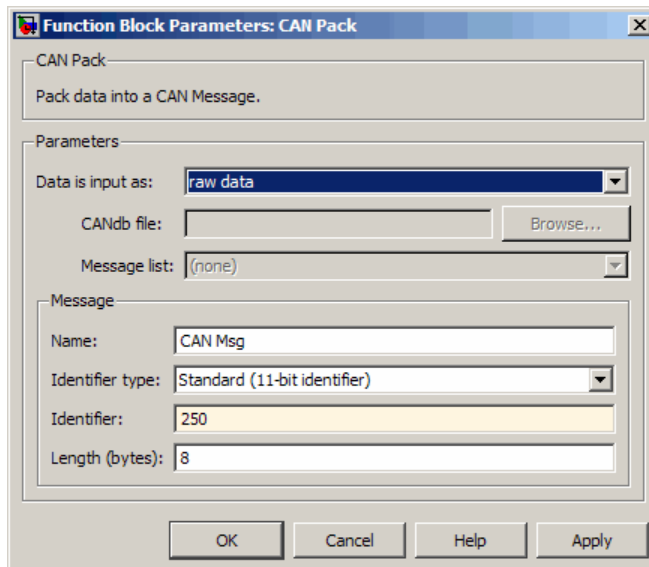
- The use of Simulink® Accelerator™ mode. Using this feature, you can speed up the execution of Simulink models.
- The use of model referencing. Using this feature, your model can include other Simulink models as modular components.
- Code generation using Real-Time Workshop to deploy models to targets.

Note Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32-bits long.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Function Block Parameters dialog box to select your CAN Pack block parameters.



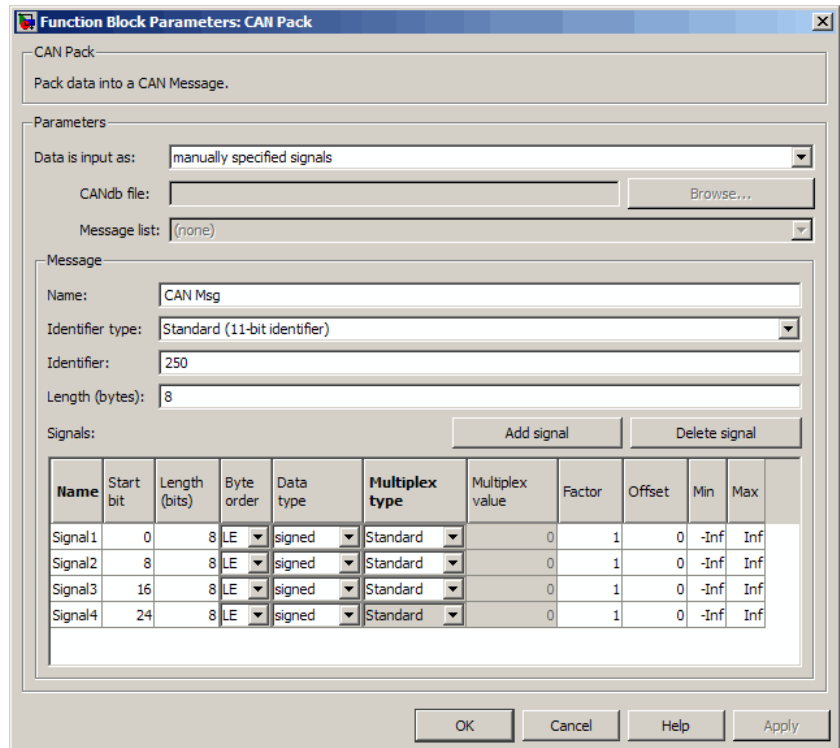
Parameters

Data is input as

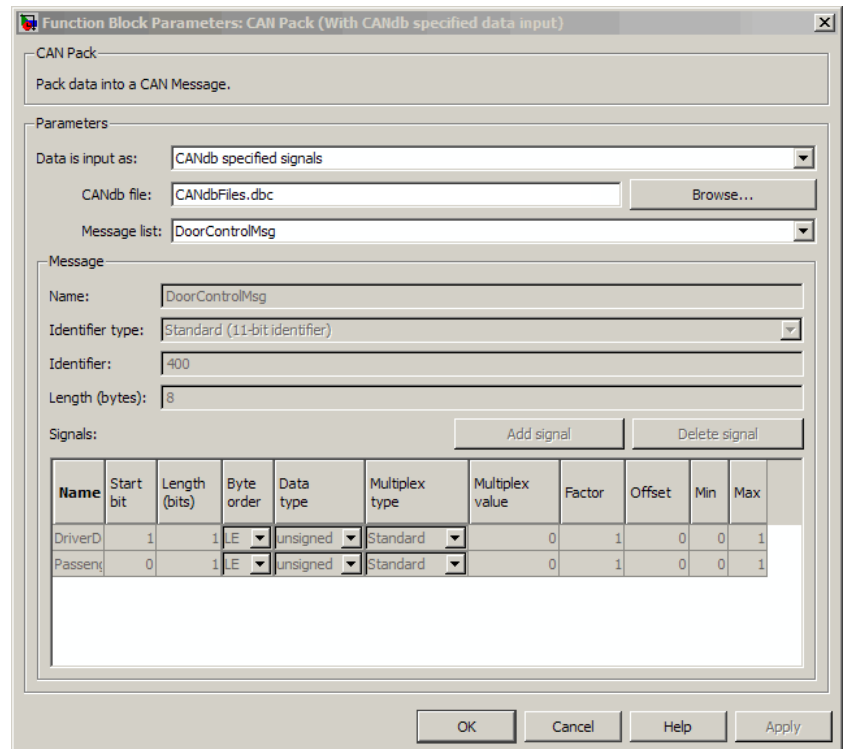
Select your data signal:

- **raw data:** Input data as a uint8 vector array. If you select this option, you only specify the message fields. All other signal parameter fields are unavailable. This option opens only one input port on your block.
- **manually specified signals:** Allows you to specify data signal definitions. If you select this option, use the **Signals** table to create your signals. The number of input ports on your block depends on the number of signals you specify.

CAN Pack



- **CANdb specified signals:** Allows you to specify a CAN database file that contains message and signal definitions. If you select this option, select a CANdb file. The number of input ports on your block depends on the number of signals specified in the CANdb file for the selected message.



CANdb file

This option is available if you specify that your data is input via a CANdb file in the **Data is input as** list. Click **Browse** to find the appropriate CANdb file on your system. The message list specified in the CANdb file populates the **Message** section of the dialog box. The CANdb file also populates the **Signals** table for the selected message.

Message list

This option is available if you specify that your data is input via a CANdb file in the **Data is input as** field and you select a CANdb file in the **CANdb file** field. Select the message to display signal details in the **Signals** table.

Message

Name

Specify a name for your CAN message. The default is `CAN Msg`. This option is available if you choose to input raw data or manually specify signals. This option is unavailable if you choose to use signals from a `CANdb` file.

Identifier type

Specify whether your CAN message identifier is a `Standard` or an `Extended` type. The default is `Standard`. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to input raw data or manually specify signals. For `CANdb` specified signals, the **Identifier type** inherits the type from the database.

Identifier

Specify your CAN message ID. This number must be a positive integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. You can also specify hexadecimal values using the `hex2dec` function. This option is available if you choose to input raw data or manually specify signals.

Length (bytes)

Specify the length of your CAN message from 0 to 8 bytes. If you are using `CANdb` specified signals for your data input, the `CANdb` file defines the length of your message. If not, this field defaults to 8. This option is available if you choose to input raw data or manually specify signals.

Signals Table

This table appears if you choose to specify signals manually or define signals using a `CANdb` file.

If you are using a `CANdb` file, the data in the file populates this table automatically and you cannot edit any fields. To edit signal information, switch to manually specified signals.

If you have selected to specify signals manually, create your signals manually in this table. Each signal you create has the following values:

Name

Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is `Signal [row number]`.

Start bit

Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message data. The start bit must be an integer from 0 through 63.

Length (bits)

Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64.

Byte order

Select either of the following options:

- **LE:** Where the byte order is in little-endian format (Intel). In this format you count bits from the start, which is the least significant bit, to the most significant bit, which has the highest bit index. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.

Bit Number		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data Byte Number	Byte 0	7	6	5	4	3	2	1	0
	Byte 1	15	14	13	12	11	10	9	8
	Byte 2	23	22	21	20	19	18	17	16
	Byte 3	31	30	29	28	27	26	25	24
	Byte 4	39	38	37	36	35	34	33	32
	Byte 5	47	46	45	44	43	42	41	40
	Byte 6	55	54	53	52	51	50	49	48
	Byte 7	63	62	61	60	59	58	57	56

LSB

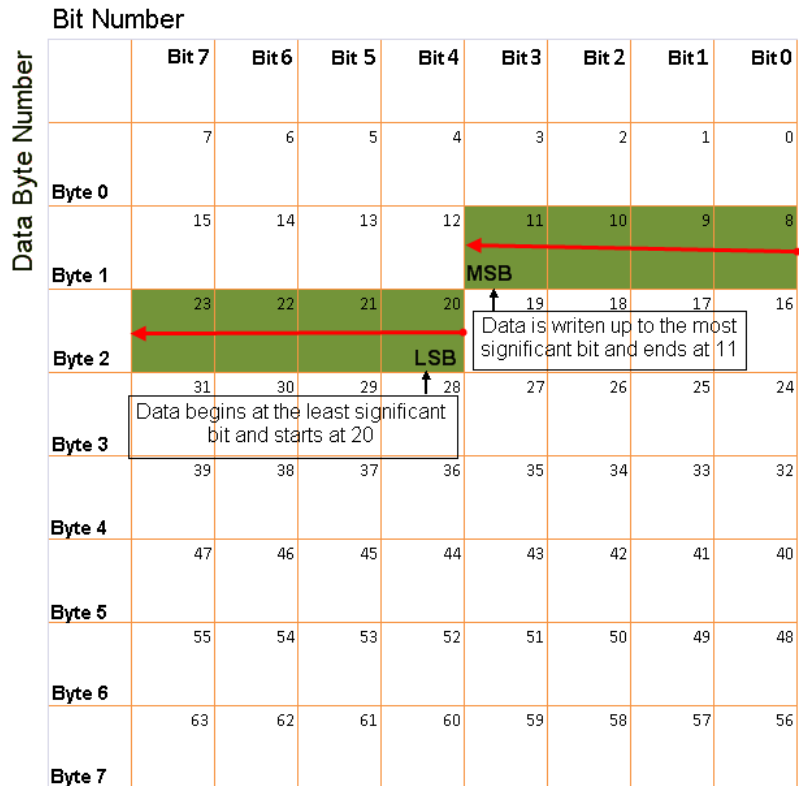
MSB

Data begins at the least significant bit and starts at 20

Data is written up to the most significant bit and ends at 27

Little Endian Byte Order Counted from the Least Significant Bit to the Highest Address

- BE: Where byte order is in big-endian format (Motorola®). In this format you count bits from the start, which is the least significant bit, to the most significant bit. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.



Big Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits.
Choose from:

- signed (default)
- unsigned
- single
- double

Multiplex type

Specify how the block packs the signals into the CAN message at each timestep:

- **Standard:** The signal is always packed at each timestep.
- **Multiplexor:** The Multiplexor signal, or the mode signal is always packed. You can specify only one Multiplexor signal per message.
- **Multiplexed:** The signal is packed if the value of the Multiplexor signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, a message has four signals with the following types and values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	N/A
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	N/A

In this example

- The block packs Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every timestep.
- If the value of Signal-D is 1 at a particular timestep, then the block packs Signal-B along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is 0 at a particular timestep, then the block packs Signal-C along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is not 1 or 0, the block does not pack either of the Multiplexed signals in that timestep.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be Multiplexed. The value you provide here must match the Multiplexor signal value at run time for the block to pack the Multiplexed signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value to apply to convert the physical value (signal value) to the raw value packed in the message. See “Conversion Formula” on page 7-37 to understand how physical values are converted to raw values packed into a message.

Offset

Specify the **Offset** value to apply to convert the physical value (signal value) to the raw value packed in the message. See “Conversion Formula” on page 7-37 to understand how physical values are converted to raw values packed into a message.

Min

Specify the minimum physical value of the signal. The default value is `-inf` (negative infinity). You can specify any number for the minimum value. See “Conversion Formula” on page 7-37 to understand how physical values are converted to raw values packed into a message.

Max

Specify the maximum physical value of the signal. The default value is `inf`. You can specify any number for the maximum value. See “Conversion Formula” on page 7-37 to understand how physical values are converted to raw values packed into a message.

Conversion Formula

The conversion formula is

$$\text{raw_value} = (\text{physical_value} - \text{Offset}) / \text{Factor}$$

CAN Pack

where `physical_value` is the value of the signal after it is saturated using the specified **Min** and **Max** values. `raw_value` is the packed signal value.

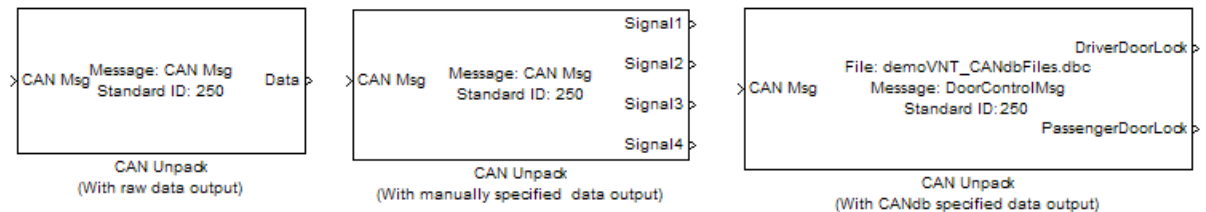
See Also

CAN Unpack

Purpose Unpack individual signals from CAN messages

Library CAN Communication

Description



The CAN Unpack block unpacks a CAN message into signal data using the specified output parameters at every timestep. Data is output as individual signals.

Note To use this block, you also need a license for Simulink software.

The CAN Unpack block has one output port by default. The number of output ports is dynamic and depends on the number of signals you specify for the block to output. For example, if your block has four signals, it has four output ports.



Other Supported Features

The CAN Unpack block supports:

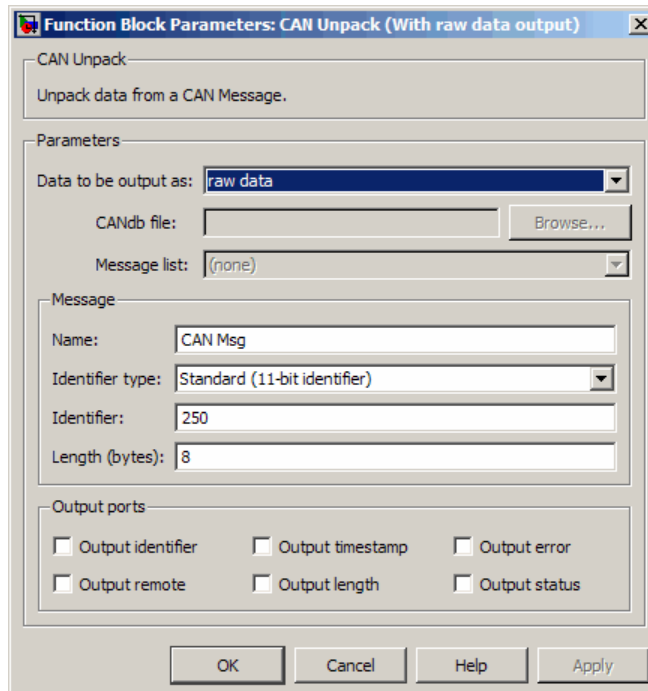
- The use of Simulink Accelerator mode. Using this feature, you can speed up the execution of Simulink models.
- The use of model referencing. Using this feature, your model can include other Simulink models as modular components.
- Code generation using Real-Time Workshop to deploy models to targets.

Note Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32-bits long.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Function Block Parameters dialog box to select your CAN message unpacking parameters.



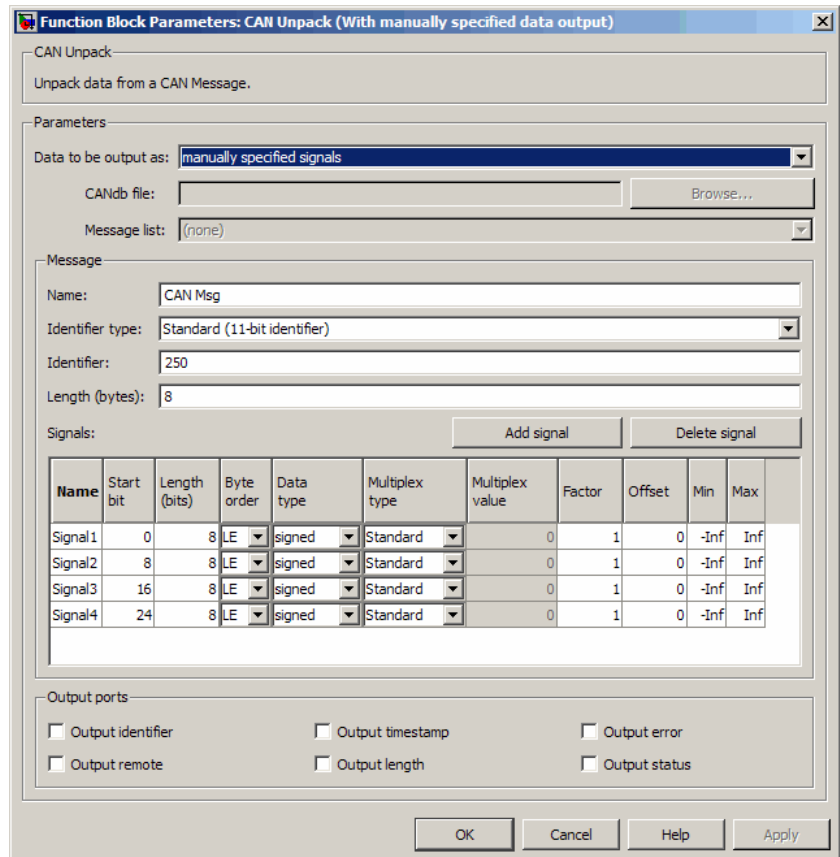
Parameters

Data to be output as

Select your data signal:

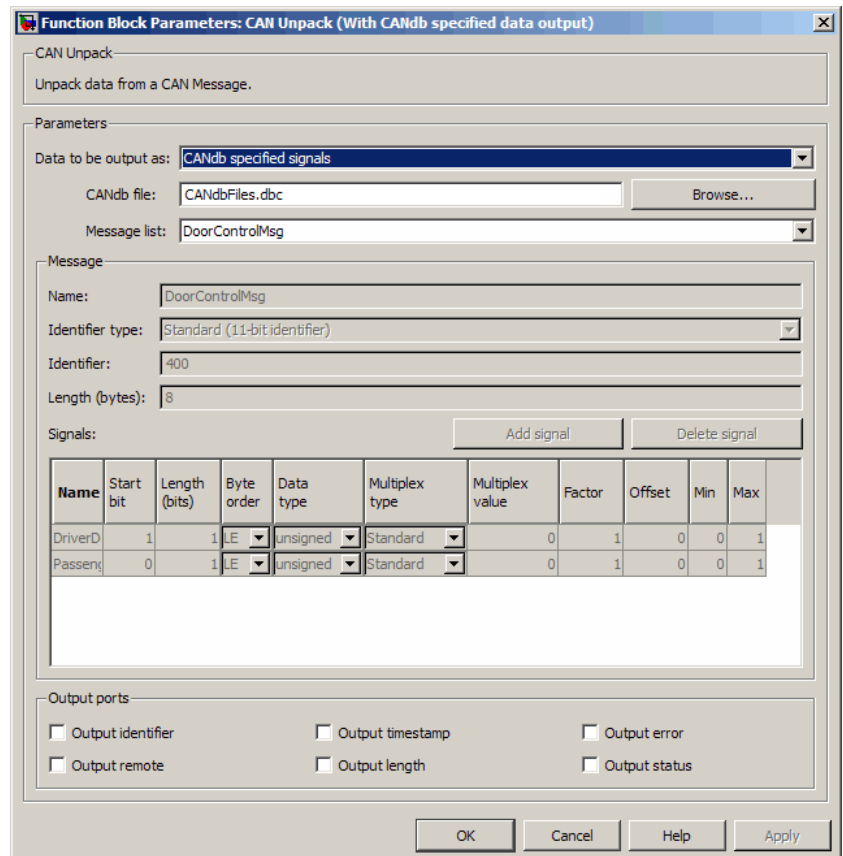
- **raw data:** Output data as a uint8 vector array. If you select this option, you only specify the message fields. All other signal parameter fields are unavailable. This option opens only one output port on your block.
- **manually specified signals:** Allows you to specify data signals. If you select this option, use the Signals table to create your signals message manually.

CAN Unpack



The number of output ports on your block depends on the number of signals you specify. For example, if you specify four signals, your block has four output ports.

- **CANdb specified signals:** Allows you to specify a CAN database file that contains data signals. If you select this option, select a CANdb file.



The number of output ports on your block depends on the number of signals specified in the CANdb file. For example, if the selected message in the CANdb file has four signals, your block has four output ports.

CANdb file

This option is available if you specify that your data is input via a CANdb file in the **Data to be output as** list. Click **Browse** to find the appropriate CANdb file on your system. The messages and signal definitions specified in the CANdb file populate the

Message section of the dialog box. The signals specified in the CANdb file populate **Signals** table.

Message list

This option is available if you specify that your data is to be output as a CANdb file in the **Data to be output as** list and you select a CANdb file in the **CANdb file** field. You can select the message that you want to view. The **Signals** table then displays the details of the selected message.

Message

Name

Specify a name for your CAN message. The default is CAN Msg. This option is available if you choose to output raw data or manually specify signals.

Identifier type

Specify whether your CAN message identifier is a **Standard** or an **Extended** type. The default is **Standard**. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to output raw data or manually specify signals. For CANdb-specified signals, the **Identifier type** inherits the type from the database.

Identifier

Specify your CAN message ID. This number must be a integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. If you specify 1, the block unpacks all messages that match the length specified for the message. You can also specify hexadecimal values using the hex2dec function. This option is available if you choose to output raw data or manually specify signals.

Length (bytes)

Specify the length of your CAN message from 0 to 8 bytes. If you are using **CANdb specified signals** for your output data, the CANdb file defines the length of your message. If not, this field

defaults to 8. This option is available if you choose to output raw data or manually specify signals.

Signals Table

This table appears if you choose to specify signals manually or define signals using a CANdb file.

If you are using a CANdb file, the data in the file populates this table automatically and you cannot edit any fields. To edit signal information, switch to manually specified signals.

If you have selected to specify signals manually, create your signals manually in this table. Each signal you create has the following values:

Name

Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is Signal [row number].

Start bit

Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message. The start bit must be an integer from 0 through 63.

Length (bits)

Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64.

Byte order

Select either of the following options:

- LE: Where the byte order is in little-endian format (Intel). In this format you count bits from the start, which is the least significant bit, to the most significant bit, which has the highest bit index. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.

CAN Unpack

		Bit Number							
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data Byte Number	Byte 0	7	6	5	4	3	2	1	0
	Byte 1	15	14	13	12	11	10	9	8
	Byte 2	23	22	21	20	19	18	17	16
	Byte 3	31	30	29	28	27	26	25	24
	Byte 4	39	38	37	36	35	34	33	32
	Byte 5	47	46	45	44	43	42	41	40
	Byte 6	55	54	53	52	51	50	49	48
	Byte 7	63	62	61	60	59	58	57	56

Little Endian Byte Order Counted from the Least Significant Bit to the Highest Address

- BE: Where the byte order is in big-endian format (Motorola). In this format you count bits from the start, which is the least significant bit, to the most significant bit. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.

		Bit Number							
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data Byte Number	Byte 0	7	6	5	4	3	2	1	0
	Byte 1	15	14	13	12	11	10	9	8
	Byte 2	23	22	21	20	19	18	17	16
	Byte 3	31	30	29	28	27	26	25	24
	Byte 4	39	38	37	36	35	34	33	32
	Byte 5	47	46	45	44	43	42	41	40
	Byte 6	55	54	53	52	51	50	49	48
	Byte 7	63	62	61	60	59	58	57	56

Big Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits.

Choose from:

- signed (default)
- unsigned
- single
- double

Multiplex type

Specify how the block unpacks the signals from the CAN message at each timestep:

- **Standard:** The signal is always unpacked at each timestep.
- **Multiplexor:** The Multiplexor signal, or the mode signal is always unpacked. You can specify only one Multiplexor signal per message.
- **Multiplexed:** The signal is unpacked if the value of the Multiplexor signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, if a message has four signals with the following values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	N/A
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	N/A

In this example

- The block unpacks Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every timestep.
- If the value of Signal-D is 1 at a particular timestep, then the block unpacks Signal-B along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is 0 at a particular timestep, then the block unpacks Signal-C along with Signal-A and Signal-D in that timestep.
- If the value of Signal-D is not 1 or 0, the block does not unpack either of the Multiplexed signals in that timestep.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be Multiplexed. The value you provide here must match the Multiplexor signal value at run time for the block to unpack the Multiplexed signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value applied to convert the unpacked raw value to the physical value (signal value). See “Conversion Formula” on page 7-50 to understand how unpacked raw values are converted to physical values.

Offset

Specify the **Offset** value applied to convert the physical value (signal value) to the unpacked raw value. See “Conversion Formula” on page 7-50 to understand how unpacked raw values are converted to physical values.

Min

Specify the minimum raw value of the signal. The default value is `-inf` (negative infinity). You can specify any number for the minimum value. See “Conversion Formula” on page 7-50 to understand how unpacked raw values are converted to physical values.

Max

Specify the maximum raw value of the signal. The default value is `inf`. You can specify any number for the maximum value. See “Conversion Formula” on page 7-50 to understand how unpacked raw values are converted to physical values.

Output Ports

Selecting an **Output ports** option adds an output port to your block.

Output identifier

Select this option to output a CAN message identifier. The data type of this port is `uint32`.

Output remote

Select this option to output the message remote frame status.
This option adds a new output port to the block. The data type of this port is **uint8**.

Output timestamp

Select this option to output the message time stamp. This option adds a new output port to the block. The data type of this port is **double**.

Output length

Select this option to output the length of the message in bytes.
This option adds a new output port to the block. The data type of this port is **uint8**.

Output error

Select this option to output the message error status. This option adds a new output port to the block. The data type of this port is **uint8**.

Output status

Select this option to output the message received status. The status is 1 if the block receives new message and 0 if it does not.
This option adds a new output port to the block. The data type of this port is **uint8**.

If you do not select any **Output ports** option, the number of output ports on your block depends on the number of signals you specify.

Conversion Formula

The conversion formula is

$$\text{physical_value} = \text{raw_value} * \text{Factor} + \text{Offset}$$

where **raw_value** is the unpacked signal value. **physical_value** is the scaled signal value which is saturated using the specified **Min** and **Max** values.

See Also

CAN Pack

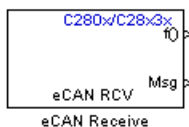
Purpose

Enhanced Control Area Network receive mailbox

Library

“C280x Chip Support (c280xlib)” on page 6-2 and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



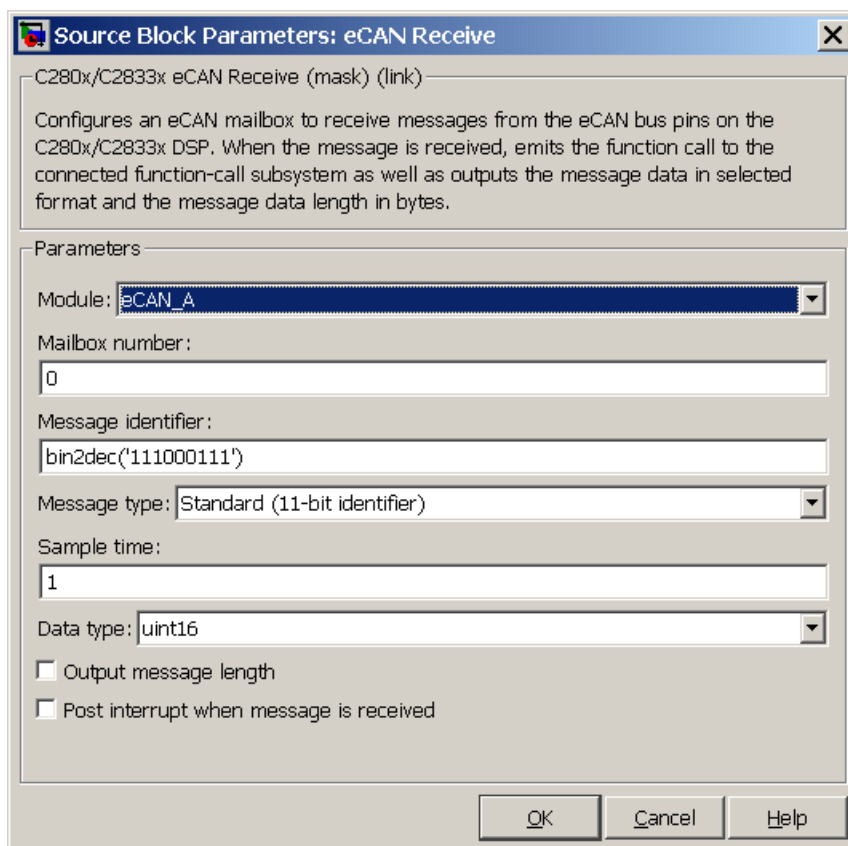
The C280x/C28x3x enhanced Control Area Network (eCAN) Receive block generates source code for receiving eCAN messages through an eCAN mailbox. The eCAN modules on the DSP chip provide serial communication capability and have 32 mailboxes configurable for receive or transmit. The C280x/C28x3x supports 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/C28x3x eCAN Receive

Dialog Box



Module

Determines which of the two eCAN modules is being configured by this instance of the C280x/C28x3x eCAN Receive block. Options are eCAN_A and eCAN_B.

Mailbox number

Sets the value of the mailbox number register (MBNR). For standard CAN controller (SCC) mode, enter a unique number from 0 to 15. For high-end CAN controller (HECC) mode enter a unique number from 0 to 31. In SCC mode, transmissions from

the mailbox with the highest number have the highest priority. In HECC mode, the mailbox number only determines priority if the Transmit priority level (TPL) of two mailboxes is equal.

Message identifier

Sets the value of the message identifier register (MID). The message identifier is 11 bits long for standard frame size or 29 bits long for extended frame size in decimal, binary, or hex format. For the binary and hex formats, use `bin2dec('')` or `hex2dec('')`, respectively, to convert the entry.

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 1-11 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 = 2

C280x/C28x3x eCAN Receive

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

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

References

For detailed information on the eCAN module, see *TMS320x281x, 280x Enhanced Controller Area Network (eCAN) Reference Guide (Rev. D)*,

Literature Number SPRU074D, available at the Texas Instruments Web site.

See Also

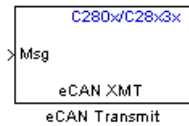
C280x/C28x3x eCAN Transmit, C280x/C2802x/C2803x/C28x3x Hardware Interrupt

C280x/C28x3x eCAN Transmit

Purpose Enhanced Control Area Network transmit mailbox

Library “C280x Chip Support (c280xlib)” on page 6-2 and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



The C280x/C28x3x 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/C28x3x 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 = 2 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,

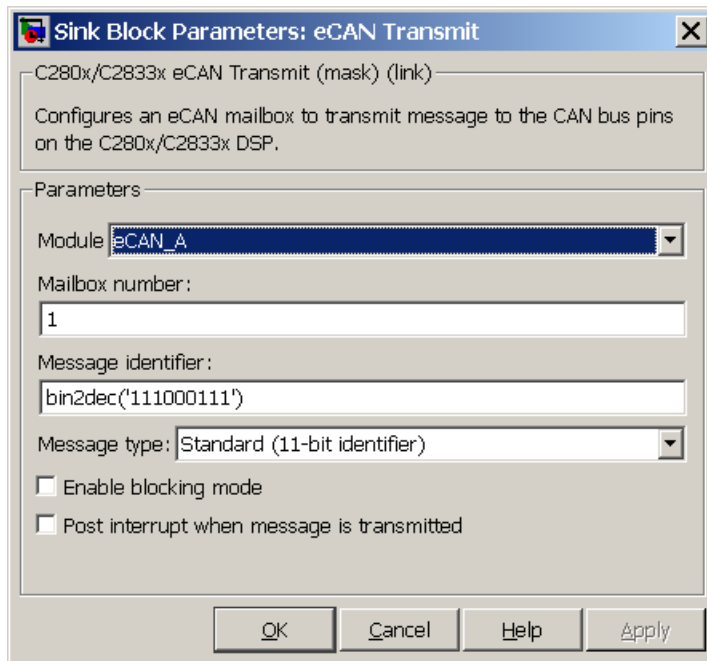
```
inputdata [0] = 0x1234
inputdata [1] = 0x5678
```

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
```

C280x/C28x3x eCAN Transmit

Dialog Box



Module

Determines which of the two eCAN modules is being configured by this instance of the C280x/C28x3x 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.

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

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

See Also

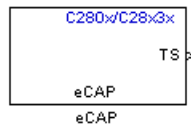
C280x/C28x3x eCAN Receive

C280x/C28x3x/C2802x eCAP

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

Library “C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

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 six C280x/C28x3x 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 five blocks configured for APWM mode with assigned pins eCAP2 through eCAP6. Or six 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.

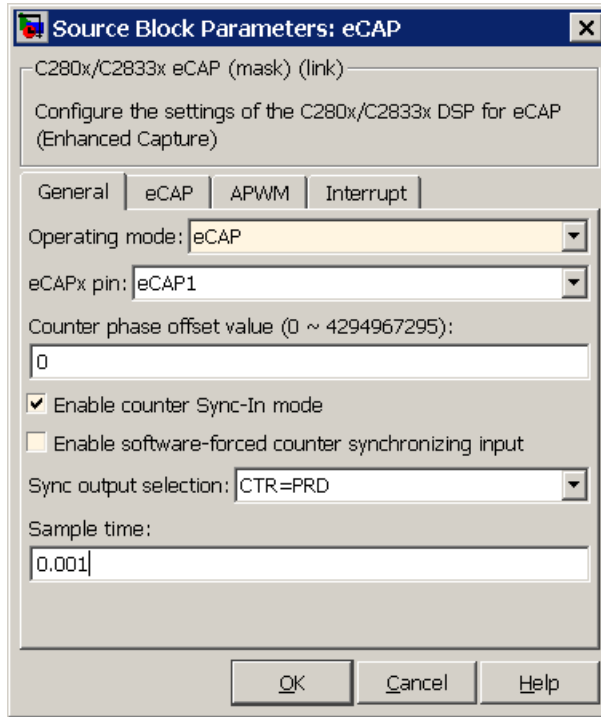
Block Input and Output Ports

The C280x/C28x3x eCAP block has optional input and output ports as shown in the following table.

Port	Description and When the Port is Enabled
Input port SI	Synchronization input for input value from software. Enabled when you select Enable software forced counter synchronizing input 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 Enable reset counter after capture event 1 time-stamp .
Output port CF	This port reports the status of the capture event. Enabled when you select Enable capture event status flag output .
Output port OF	Enabled when you select Enable overflow status flag output .

Note The outputs of this block can be vectorized.

General Pane



Operating mode

When you select eCAP, the block captures and logs pin transitions for each capture unit to a FIFO buffer. When you select APWM, the block generates asymmetric pulse width modulation (APWM) waveforms for driving downstream systems.

eCAPx pin

The capture unit includes the following features:

- One pin for each capture unit. For example, eCAP1, eCAP2, and so on.
- Four maskable interrupt flags, one for each capture unit.

- Ability to specify the transition detection—rising edge, falling edge, or both edges.

Counter phase offset value (0~4294967295)

The value you enter here provides 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.

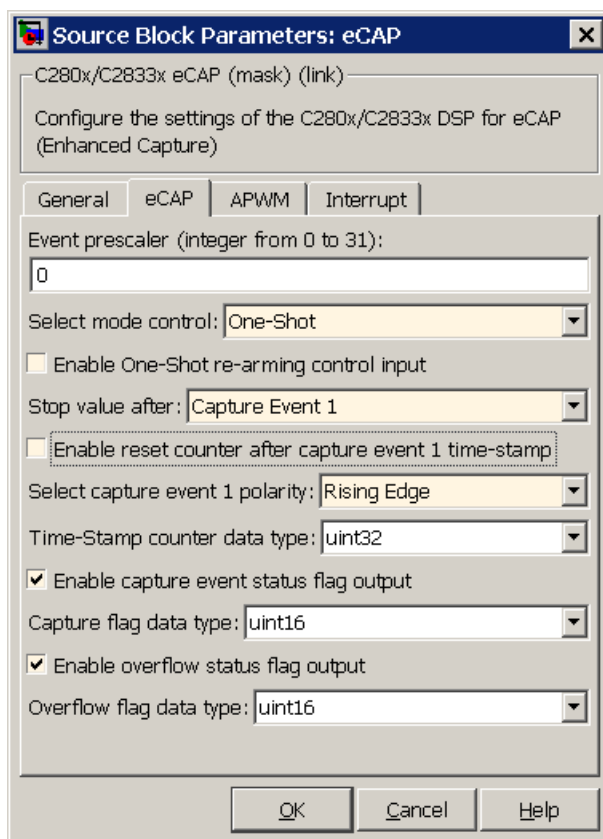
Sample time

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.

C280x/C28x3x/C2802x eCAP



Event prescaler (integer from 0 to 31)

Multiply the input signal, called a pulse train, by this value. Entering a 0 bypasses the input prescaler, leaving the input capture signal unchanged.

Select mode control

Continuous performs continuous timestamp captures using a circular buffer to capture events 1 through 4.

One-Shot disables 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 option 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

Specifies the number of capture events after which to stop the capture.

Enable reset counter after capture event 1 timestamp

Enables a reset after capture event 1 and creates an **Output port TS**. When you select this option, the eCAP process resets the counters after receiving a capture event 1 timestamp.

Select capture event 1 polarity

Start the capture event on a **Rising edge** or **Falling edge**.

Time-Stamp counter data type

Select the data type of the counter. The list includes integer and unsigned 8-, 16-, and 32-bit data types, double, single, and Boolean.

Enable capture event status flag output

Output the capture event status flag on the **Output port CF**. The block outputs a 0 until the event capture. After the event, the flag value is 1.

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.

Enable overflow status flag output

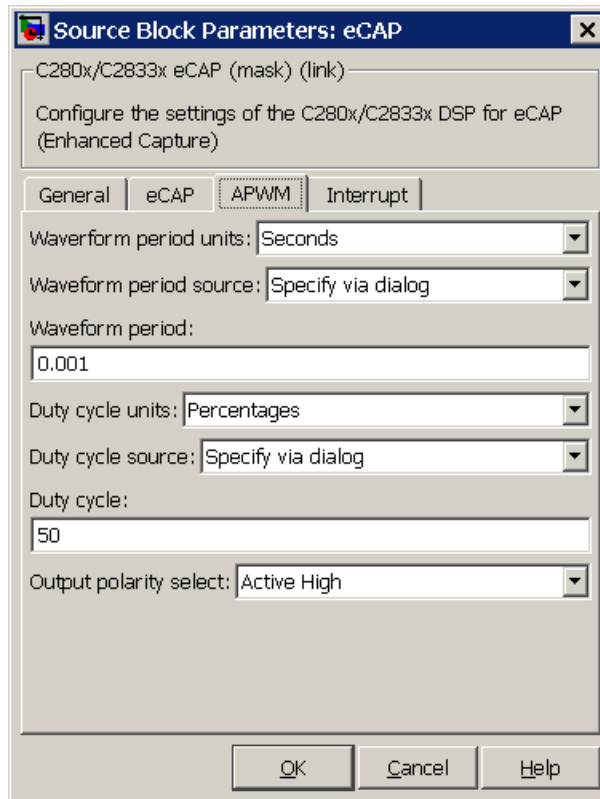
Output the status of the elements of the FIFO buffer on the **Output port OF**. After you select this option, set the data type for the flag in **Overflow flag data type**.

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.

APWM Pane

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



Waveform period units

Set the units for measuring the waveform period. **Clock cycles** uses the high-speed peripheral clock cycles of the DSP chip, or **Seconds**. 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

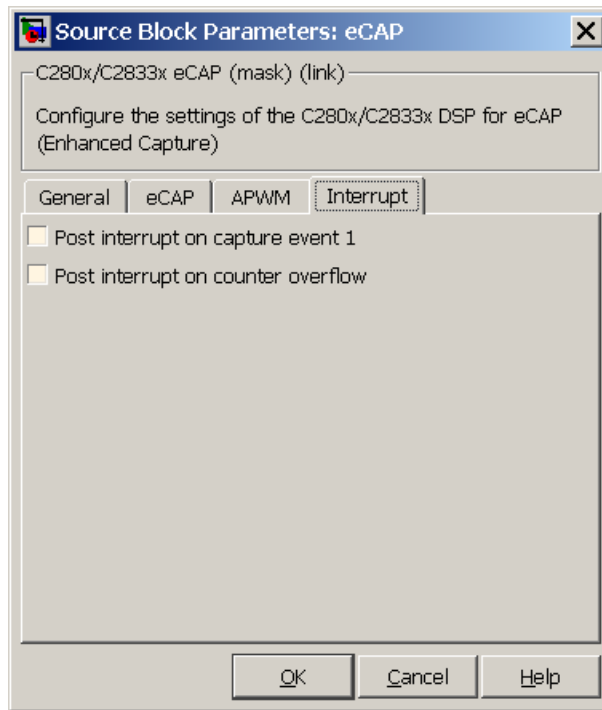
Ratio of the PWM waveform pulse duration to the PWM waveform period expressed in **Duty cycle units**.

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 setting **Operating mode** on the **General** pane to eCAP.



Post interrupt on capture event 1

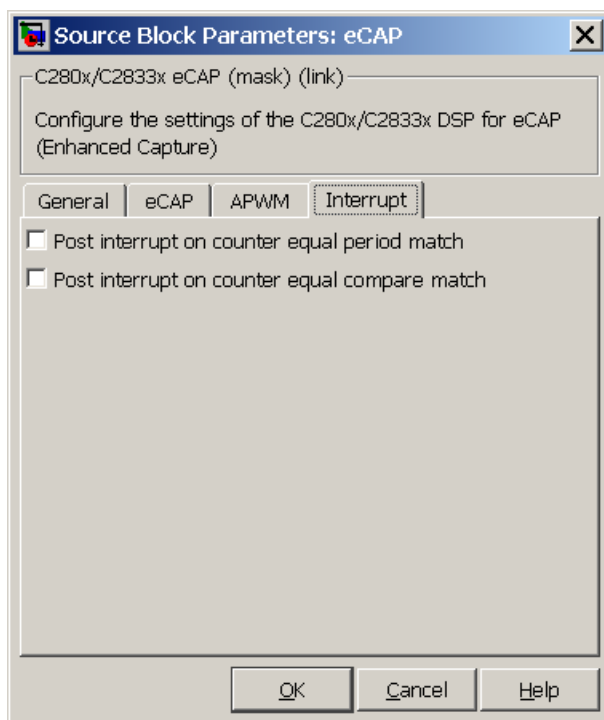
Enables capture event 1 as an interrupt source. You can use the C280x/C2802x/C2803x/C28x3x Hardware Interrupt block to react to this interrupt.

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 setting **Operating mode** on the **General** pane to APWM.

C280x/C28x3x/C2802x eCAP



Post interrupt on counter equal period match

Post an interrupt when the value of the counter is the same as the value of the period register (CTR=PRD).

Post interrupt on counter equal compare match

Post an interrupt when the value of the counter is the same as the value of the compare register (CTR=CMP).

References

For detailed information about interrupt processing, see *TMS320x28xx, 28xxx Enhanced Capture (eCAP) Module Reference Guide*, SPRU807B, available at the Texas Instruments Web site.

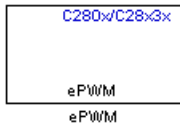
Purpose

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

Library

“C280x Chip Support (c280xlib)” on page 6-2 and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



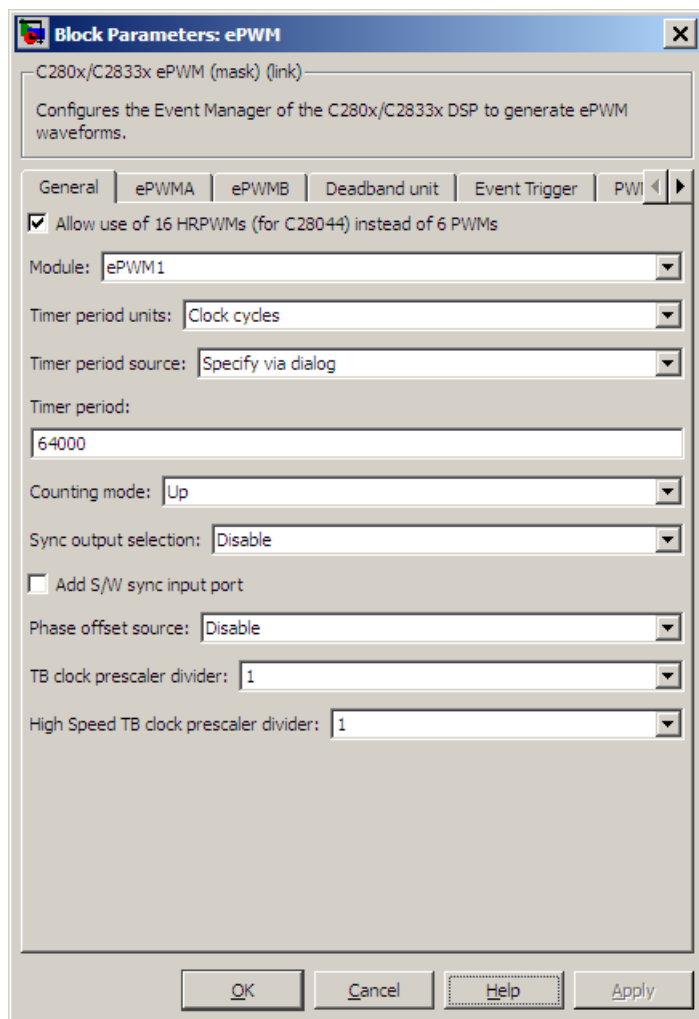
A C280x/C28x3x system contains multiple ePWM modules, each having two PWM outputs. You can use the C280x/C28x3x ePWM block to configure up to six ePWM modules.

When the High-Resolution Pulse Width Modulator (HRPWM) is enabled, the ePWM block uses the Scale Factor Optimizing Software Version 5 library (SFO_TI_Build_V5.lib), which can “dynamically determine the number of MEP steps per SYSCLKOUT period.” For more information, consult *TMS320x28xx, 28xxx High-Resolution Pulse Width Modulator (HRPWM) Reference Guide*, Literature Number SPRU924, available at the Texas Instruments Web site.

C280x/C28x3x ePWM

Dialog Box

General Pane



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

Enable all 16 High-Resolution PWM modules (HRPWM) on the C28044 digital signal controller when the PWM resolution is too low.

For example, the Spectrum Digital eZdsp™ F28044 board has a system clock of 100 MHz (200-kHz switching). At these frequencies, conventional PWM resolution is too low—approximately 9 bits or 10 bits. By comparison, the HRPWM resolution for the same board is 14.8 bits.

All the C280x/C28x3x ePWM blocks in your model become HRPWM blocks. Thus, when you enable this parameter:

- Use the HRPWM parameters under the ePWMA tab to further make additional configuration changes.
- Most of the configuration parameters under the ePWMB tab are unavailable and should be disregarded.
- Your model can contain up to 16 C280x/C28x3x ePWM blocks, provided you configure each one for a separate module. (For example, **Module** is ePWM1, ePWM2, and so on.)

For processors other than the C28044, deselect (disable) **Allow use of 16 HRPWMs (for C28044) instead of 6 PWMs**. To enable HRPWM for other processors, first determine how many HRPWM modules are available. Consult the Texas Instruments documentation for your processor, and then use the HRPWM parameters under the ePWMA tab to enable and configure HRPWM.

For additional information about the C28044 and HRPWM, consult the “References” on page 7-95 section.

Module

Specify which target ePWM module to use.

Timer period units

Specify the units of the **Timer period** or **Timer initial period** as **Clock cycles** (the default) or **Seconds**. If **Timer period units** is set to **Seconds**, the **Timer period** or **Timer initial period**, a double, must be down-converted for the period register, a uint16. For best performance, select **Clock cycles**. Doing so reduces calculations and rounding errors.

Note If you set **Timer period units** to **Seconds**, you must enable support for floating-point numbers. In the model window, select **Simulation > Configuration Parameters**. In the Configuration Parameters dialog box, select **Real-Time Workshop > Interface**. Under **Software Environment**, enable **floating-point numbers**.

Timer period source

Configure the source of the timer period value. Selecting **Specify via dialog** changes the following parameter to **Timer period**. Selecting **Input port** changes the following parameter to **Timer initial period** and creates a timer period input port, **T**, on the block.

Timer period

Set the period of the PWM waveform in clock cycles or in seconds, as determined by the **Timer period units** parameter.

Note The term *clock cycles* refers to the Time-base Clock on the C280x/C28x3x chip. See the discussion of the **TB clock prescaler divider** below for an explanation of how the Time-base Clock speed is calculated.

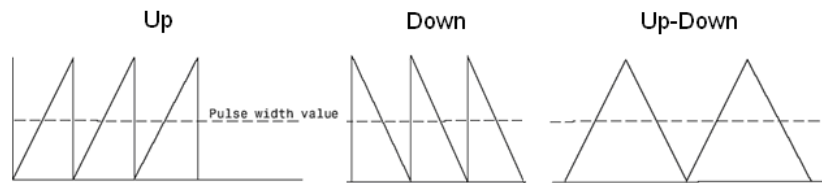
Timer initial period

The period of the waveform from the time the PWM peripheral starts operation until the ePWM input port, **T**, receives a new value for the period. Use **Timer period units** to measure the period in clock cycles or in seconds.

Note The term *clock cycles* refers to the Time-base Clock on the C280x/C28x3x 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

Specify the counting mode in which to operate. C280x/C28x3x 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

This parameter corresponds to the SYNCOSSEL field in the Time-Base Control Register (TBCTL).

Use this parameter to specify the event that generates a Time-base synchronization output signal, EPWMxSYNCO, from the Time-base (TB) submodule.

The available choices are:

- EPWMxSYNCl or SWFSYNc — a Synchronization input pulse or Software forced synchronization pulse, respectively. This option can be used to achieve precise synchronization across multiple ePWM modules by daisy chaining multiple the Time-base (TB) submodules.
- CTR=Zero — Time-base counter equal to zero (TBCTR = 0x0000)
- CTR=CMPB — Time-base counter equal to counter-compare B (TBCTR = CMPB)
- Disable — Disable the EPWMxSYNCO output (the default)

Add S/W sync input port

Create an input port, **SYNc**, for a Time-base synchronization input signal, EPWMxSYNCl. This option can be used to achieve precise synchronization across multiple ePWM modules by daisy-chaining multiple the Time-base (TB) submodules.

Phase offset source

Specify the source of a phase offset to apply to the Time-base synchronization input signal, EPWMxSYNCl from the **SYNc** input port. Selecting **Specify via dialog** creates the **Phase offset value** parameter. Selecting **Input port** creates a phase input port, **PHS**, on the block. Selecting **Disable**, the default value, prevents any phase offsets from being applied to the TB module.

Counting direction after phase synchronization

This parameter appears when **Counting Mode** is set to Up-Down and **Phase offset source** is enabled (set to **Specify via dialog** or **Input port**). Configure the timer to count up from zero, or down to zero, following synchronization. This parameter corresponds to the PHSDIR field of the Time-base Control Register (TBCTL).

Phase offset value

This field appears when you select **Specify via dialog** in **Phase offset source**.

Configure the phase offset (delay) from the arrival of the Time-base synchronization input signal, EPWMxSYNCl, on the **SYNC** input port to moment the Time-base (TB) submodule synchronizes the ePWM module.

Note Enter the **Phase offset value** in TBCLK cycles, from 0 to 65535. Fractional seconds cannot be used.

This parameter corresponds to the Time-Base Phase Register (TBPHS).

TB clock prescaler divider

Use the **TB clock prescaler divider** (CLKDIV) and the **High Speed TB clock prescaler divider** (HSPCLKDIV) to configure the Time-base clock speed (TBCLK) for the ePWM module using the following equation:

$$\text{TBCLK} = \text{SYSCLKOUT}/(\text{HSPCLKDIV} * \text{CLKDIV})$$

For example, the default values of both CLKDIV and HSPCLKDIV are 1, and the default frequency of SYSCLKOUT is 100 MHz, so:

$$\text{TBCLK} = 100 \text{ MHz} = 100 \text{ MHz}/(1 * 1)$$

The choices for the **TB clock prescaler divider** are: 1, 2, 4, 8, 16, 32, 64, and 128.

The **TB clock prescaler divider** parameter corresponds to the CLKDIV field of the Time-base Control Register (TBCTL).

Note The frequency of SYSCLKOUT depends on the oscillator frequency and the configuration of PLL-based clock module. Changing the values of the PLL Control Register (PLLCR) will affect the timing of all ePWM modules.

For more information, consult the “PLL-Based Clock Module” section of the data manual for your specific target (see “References” on page 7-95 below).

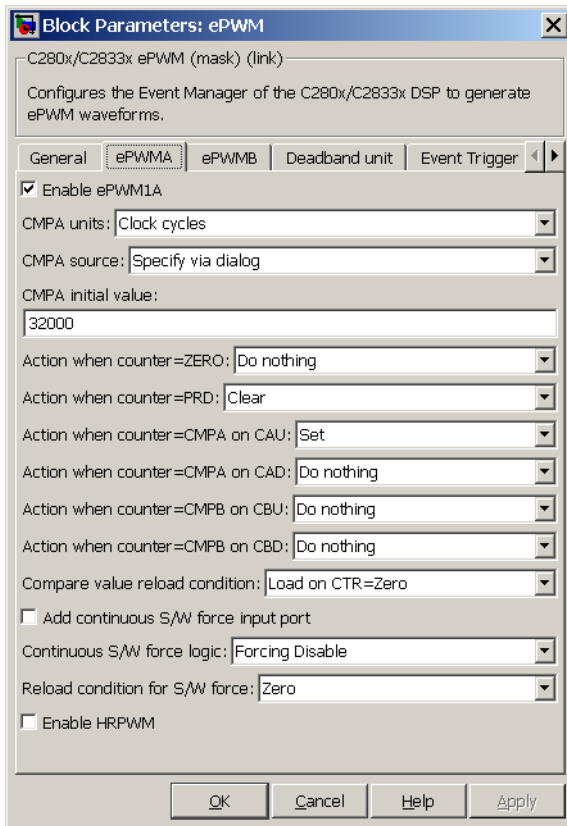
High Speed TB clock prescaler divider

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 to divide by 1, 2, 4, 6, 8, 10, 12, and 14.

This parameter corresponds to the HSPCLKDIV field of the Time-base Control Register (TBCTL).

ePWMA and ePWMB panes

Each ePWM module has two outputs, ePWMA and ePWMB. The **ePWMA output** pane and **ePWMB output** pane include the same settings, although the default values may be different in some cases, as noted.



Block Parameters: ePWM [X]

C280x/C2833x ePWM (mask) (link)

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

General | **ePWMA** | ePWMB | Deadband unit | Event Trigger

Enable ePWM1A

CMPA units: Clock cycles

CMPA source: Specify via dialog

CMPA initial value: 32000

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

Add continuous S/W force input port

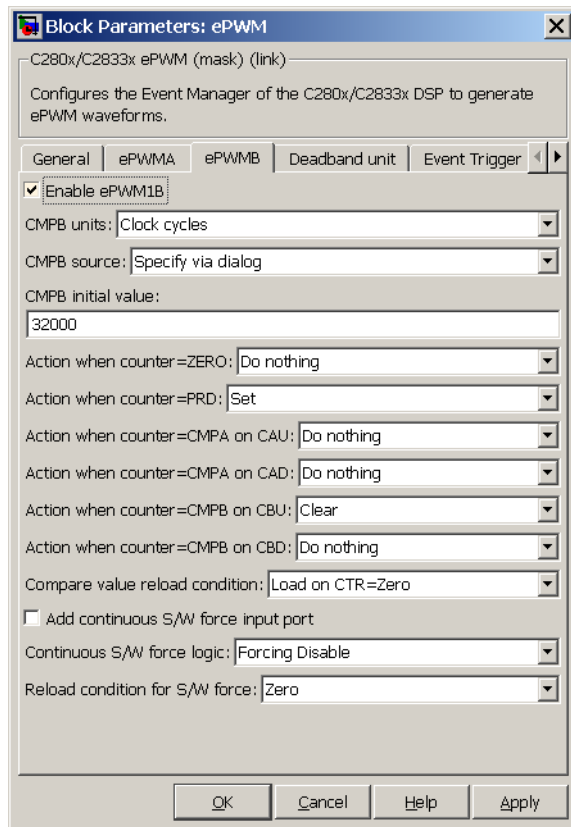
Continuous S/W force logic: Forcing Disable

Reload condition for S/W force: Zero

Enable HRPWM

OK Cancel Help Apply

C280x/C28x3x ePWM



Enable ePWMxA

Enable ePWMxB

Enables the ePWMA and/or ePWMB output signals for the ePWM module identified on the **General** pane. By default, **Enable ePWMxA** is enabled, and **Enable ePWMxB** is disabled.

Note To **Enable ePWMxA** or **Enable ePWMxA**, you must also enable support for floating-point numbers: In the model window, select **Tools > Real Time Workshop > Options**. In the Configuration Parameters dialog box, select **Real-Time Workshop > Interface**. Under **Software Environment**, enable **floating-point numbers**.

CMPA units

CMPB units

Specify the units used by the compare register: Percentages (the default) or Clock cycles.

Notes

- The term *clock cycles* refers to the Time-base Clock on the C280x/C28x3x chip. See the discussion of the **TB clock prescaler divider** below for an explanation of how the Time-base Clock speed is calculated.
 - Using percentages may cause some additional computation time in generated code. The effects may or may not be noticeable in your application.
 - If you set **CMPA units** or **CMPB units** to Percentages, you must also enable support for floating-point numbers: In the model window, select **Simulation > Configuration Parameters**. In the Configuration Parameters dialog box, select **Real-Time Workshop > Interface**. Under **Software Environment**, enable **floating-point numbers**.
-

CMPA source

CMPB source

Specify the source from which the pulse width is to be obtained. If you select **Specify via dialog** (the default), enter a value in the

CMPA value or **CMPB value** field. If you select **Input port**, set the value using an input port, **WA** or **WB**, on the block. If you select **Input port** also set **CMPA initial value** or **CMPB initial value**.

CMPA value

CMPB value

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

CMPA initial value

CMPB initial value

This field appears when you set **CMPA source** or **CMPB source** to **Input port**. Enter the initial pulse width of **CMPA** or **CMPB** the PWM peripheral uses when it starts operation. Subsequent inputs to the **WA** or **WB** ports will change the **CMPA** or **CMPB** pulse width.

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	Do nothing	Do nothing
PRD	Clear	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, consult 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.

Add continuous S/W force input port specifies the source from which the control logic is obtained. This check box is cleared by

C280x/C28x3x ePWM

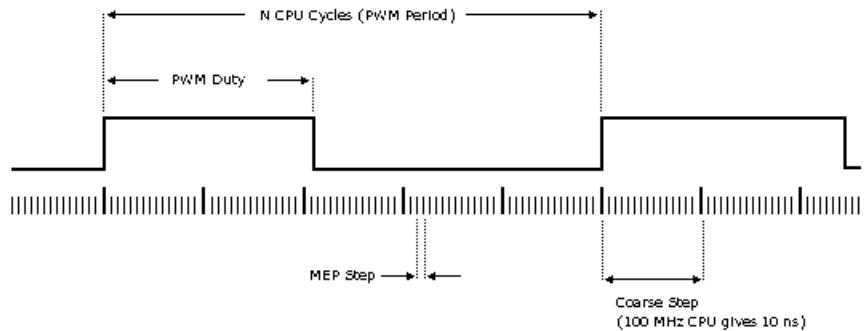
default. Select this check box to obtain the control logic from the input port

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)**™ 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

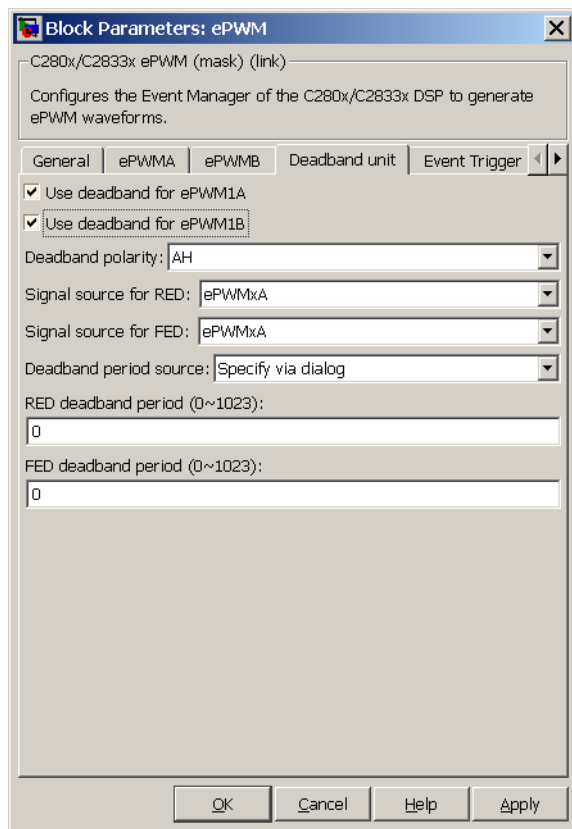
Enable scale factor optimizer software™

Select to enable Scale Factor Optimizing Software Version 5 (SFO_V5) library. 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. Applications that use the HRPWM feature should use the SFO_V5.

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.

C280x/C28x3x ePWM



Use deadband for ePWMxA

Use deadband for ePWMxB

Enables a deadband area of no signal overlap between pairs of ePWM output signals. This check box is cleared by default.

Deadband polarity

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

Deadband period source

Specify 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 you select **Use deadband for ePWMxA** in the **ePWMA output** pane. Enter a value from 0 to 1023 to specify a rising edge delay.

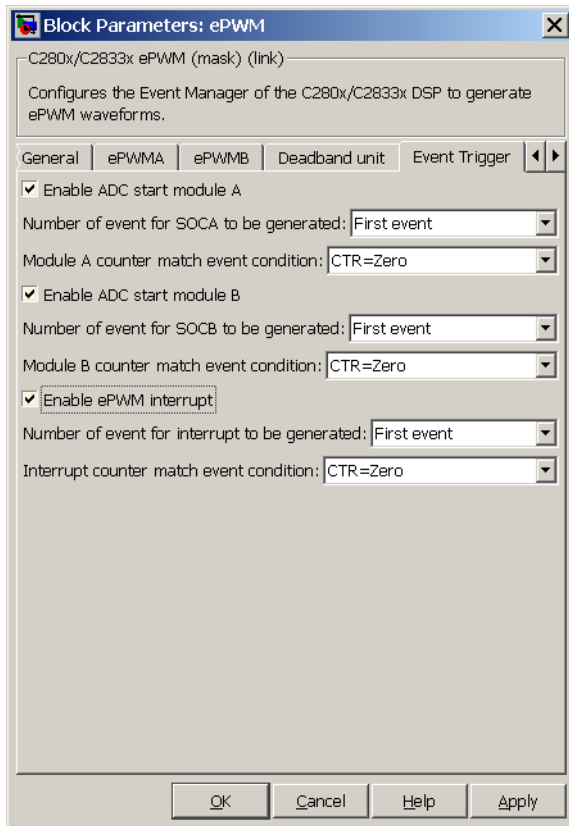
FED deadband period

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

Event Trigger Pane

Configure ADC Start of Conversion (SOC) by one or both of the ePWMA and ePWMB outputs.

C280x/C28x3x ePWM



Enable ADC start module A

When you select this option, ePWM starts the Analog-to-Digital Conversion (ADC) for module A. By default, this check box is cleared (disabled).

Number of event for SOCA to be generated

When you select **Enable ADC start module A**, this field specifies the number of the event that triggers ADC Start of Conversion for Module A (SOCA): **First** event triggers ADC start of conversion with every event (the default), **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

When you select **Enable ADC start module A**, this field specifies the counter match condition that triggers an ADC start of conversion event. The choices are:

CTR=Zero

When the ePWM counter reaches zero (the default).

CTR=PRD

When the ePWM counter reaches the period value.

CTRU=CMPA

When the ePWM counter reaches the compare A value on the way up.

CTRD=CMPA

When the ePWM counter reaches the compare A value on the way down.

CTRU=CMPB

When the ePWM counter reaches the compare B value on the way up.

CTRD=CMPB

When the ePWM counter reaches the compare B value on the way down.

Enable ADC start module B

When you select this option, ePWM starts the Analog-to-Digital Conversion (ADC) for module B. By default, this check box is cleared (disabled).

Number of event for SOCB to be generated

When you select **Enable ADC start module B**, this field specifies the number of the event that triggers ADC start of conversion: First event triggers ADC start of conversion with every event (the default), 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

When you select **Enable ADC start module B**, this field specifies the counter match condition that triggers an ADC start of conversion event. Choices are CTR=Zero (the default), CTR=PRD, CTRU=CMPA, CTRD=CMPA, CTRU=CMPB, and CTRD=CMPB, as defined for **Module A counter match event condition** above.

Enable ePWM interrupt

When you select this option, you can generate interrupts based on different events defined by **Number of event for interrupt to be generated** and **Interrupt counter match event condition**. By default, this check box is cleared.

Number of event for interrupt to be generated

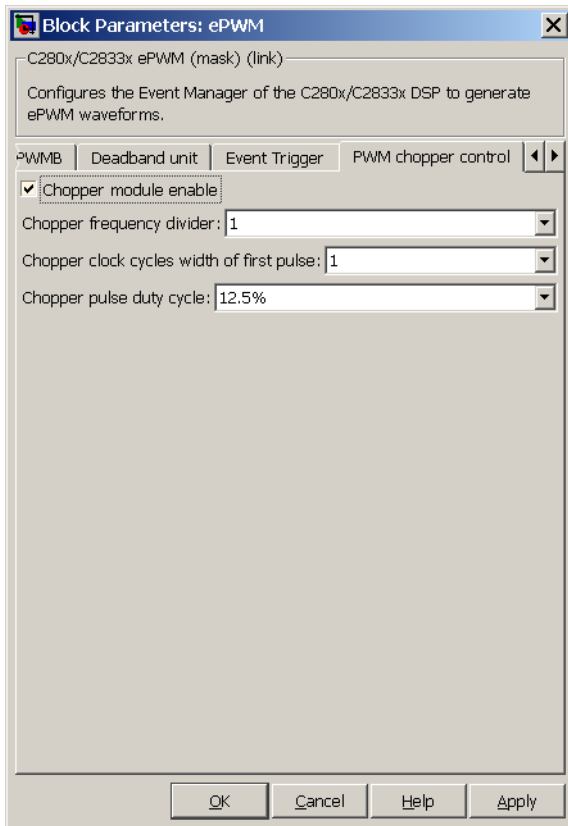
When you select **Enable ePWM interrupt**, this field specifies the number of the event that triggers the ePWM interrupt: First event triggers ePWM interrupt with every event (the default), Second event triggers ePWM interrupt with every second event, and Third event triggers ePWM interrupt with every third event.

Interrupt counter match event condition

When you select **Enable ePWM interrupt**, this field specifies the counter match condition that triggers ePWM interrupt. Choices are CTR=Zero (the default), CTR=PRD, CTRU=CMPA, CTRD=CMPA, CTRU=CMPB, and CTRD=CMPB, as defined for **Module A counter match event condition** above.

PWM Chopper Control Pane

The **PWM chopper control** pane lets you specify parameters for the PWM-Chopper (PC) submodule. The PC submodule uses 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

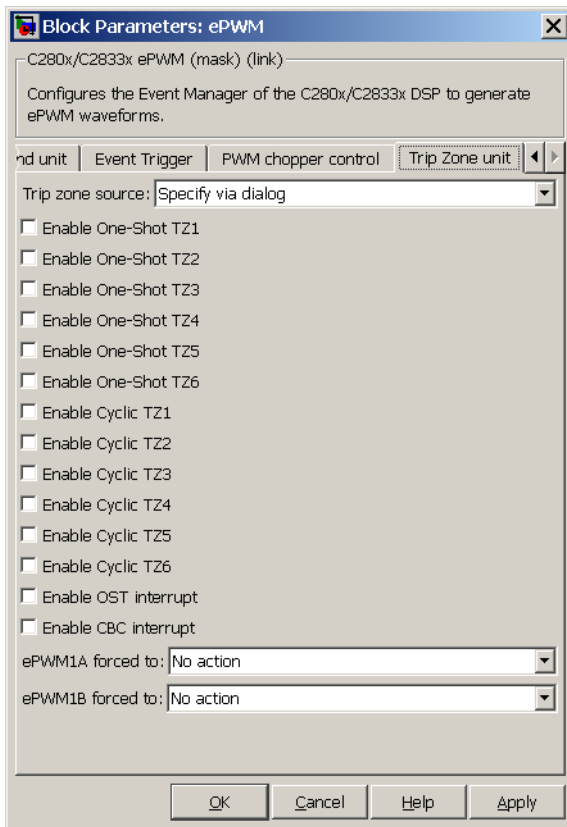
Choose an integer value from 1 to 16 to set the width of the first pulse. Use this feature to provide a high-energy first pulse to ensure hard and fast power switch turn on.

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

Specify the source of the control logic. Choose **Specify via dialog** (the default) to enable specific Trip-zone signals via the block dialog. Choose **Input port** to enable specific Trip-zone signals via a block input port, **TZSEL**.

If you select **Input port**, use the following bit operation to determine the value of the 16-bit integer to send to the **TZSEL**:

$$\text{TZSEL INPUT VALUE} = (\text{OSHT6} * 2^{13} + \text{OSHT5} * 2^{12} + \text{OSHT4} * 2^{11} + \text{OSHT3} * 2^{10} + \text{OSHT2} * 2^9 + \text{OSHT1} * 2^8 + \text{CBC6} * 2^5 + \text{CBC5} * 2^4 + \text{CBC4} * 2^3 + \text{CBC3} * 2^2 + \text{CBC2} * 2^1 + \text{CBC1} * 2^0)$$

For more information, see the "Trip-Zone Submodule Control and Status Registers" section of the *TMS320x28xx, 28xxx Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, Literature Number: SPRU791 on www.ti.com

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).

References

For more information, consult the following references, available at the Texas Instruments Web site:

- *TMS320x28xx, 28xxx Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, literature number SPRU791
- *TMS320x280x, 2801x, 2804x High Resolution Pulse Width Modulator Reference Guide*, literature number SPRU924E
- *Using the ePWM Module for 0% - 100% Duty Cycle Control Application Report*, literature number SPRU791
- *Configuring Source of Multiple ePWM Trip-Zone Events*, literature number SPRAAR4
- *TMS320F2809, TMS320F2808, TMS320F2806 TMS320F2802, TMS320F2801 TMS320C2802, TMS320C2801, and TMS320F2801x DSPs Data Manual*, literature number SPRS230
- *TMS320F28044 Digital Signal Processor Data Manual*, literature number SPRS357
- *TMS320F28335/28334/28332 TMS320F28235/28234/28232 Digital Signal Controllers (DSCs) Data Manual*, literature number SPRS439

See Also

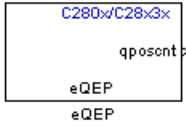
C280x/C28x3x ADC

C280x/C28x3x eQEP

Purpose Quadrature encoder pulse circuit

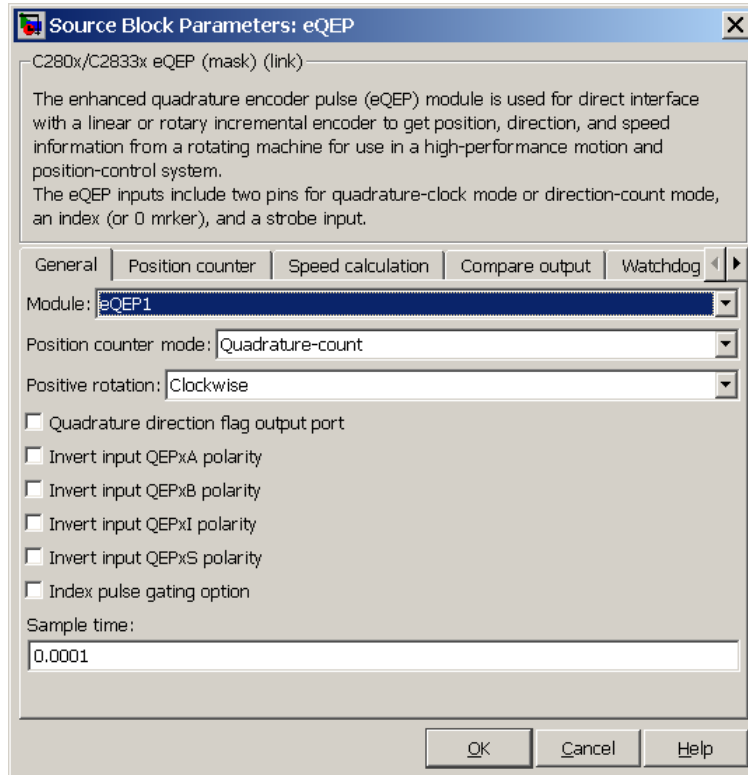
Library “C280x Chip Support (c280xlib)” on page 6-2 and “C28x3x Chip Support (c2833xlib)” on page 6-8

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/C28x3x-based board. Choose eQEP1 (the default) or eQEP2.

Position counter mode

The input signals QEPxA and QEPxB 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 create a port on the block that gives access to the direction flag of the quadrature module.

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.

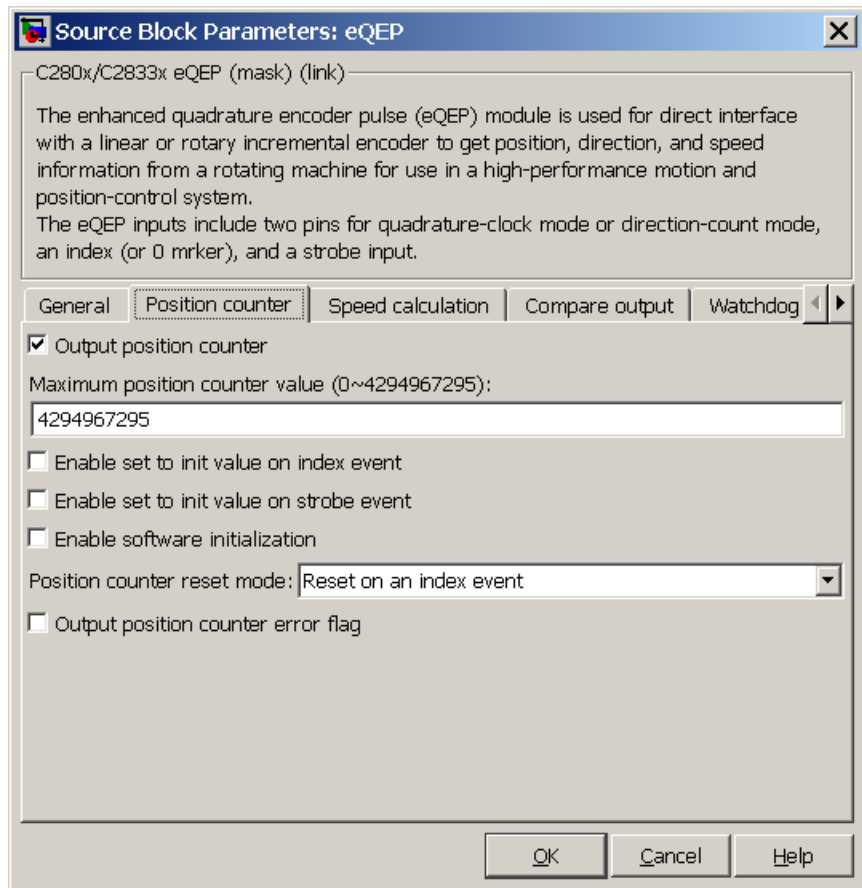
Index pulse gating option

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 value defaults to 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 value defaults to 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

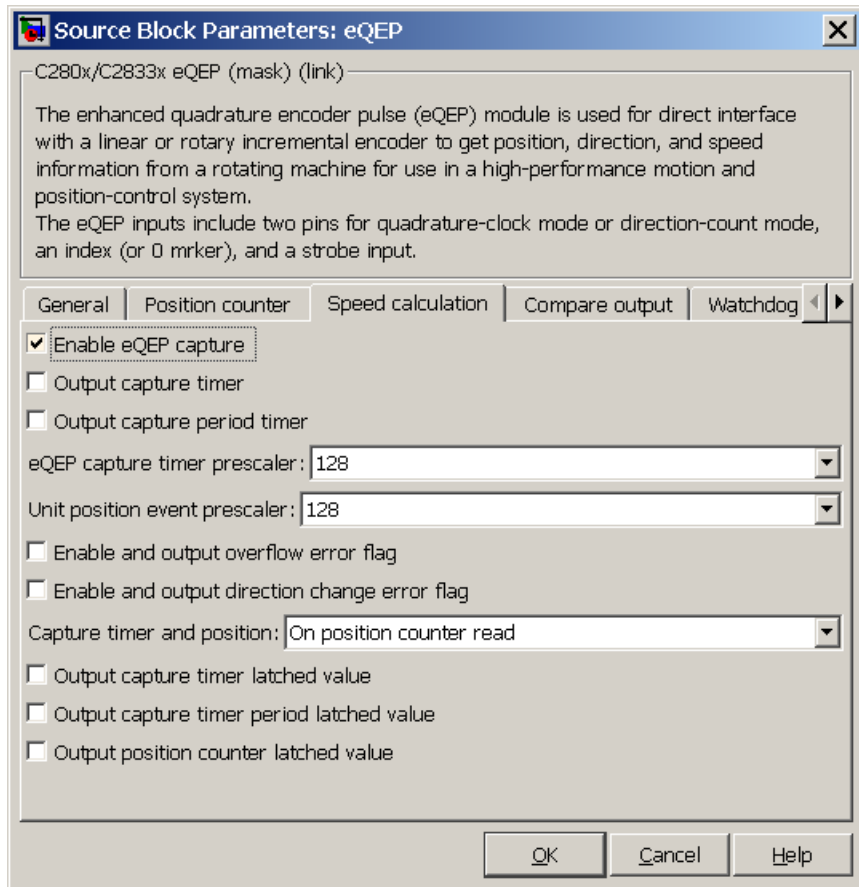
This check box appears only when **Position counter reset mode** is set to Reset on the maximum position or Reset on the first index event. The eQEP strobe input can be configured to latch the position counter (QPOSCNT) into QPOSSLAT on occurrence of a definite event on this pin. Select this check box 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 value defaults to 100000000.

Output capture timer latched value

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

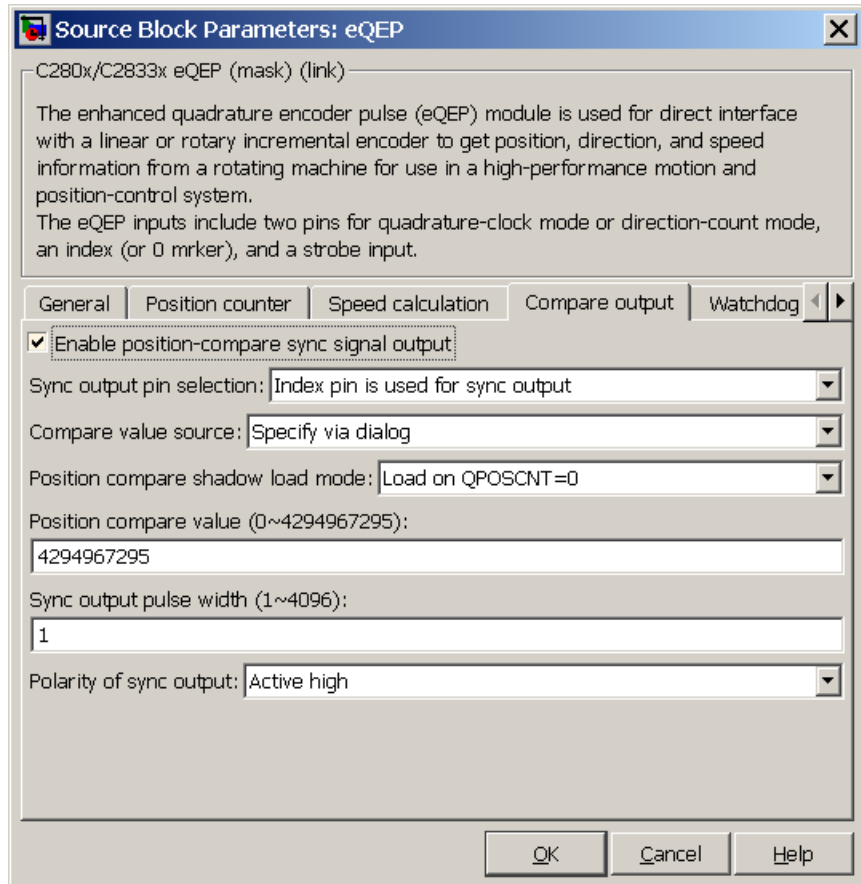
Output capture timer period latched value

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 value defaults to 4294967295. This value is loaded into the position-compare register (QPOSCMP).

Sync output pulse width

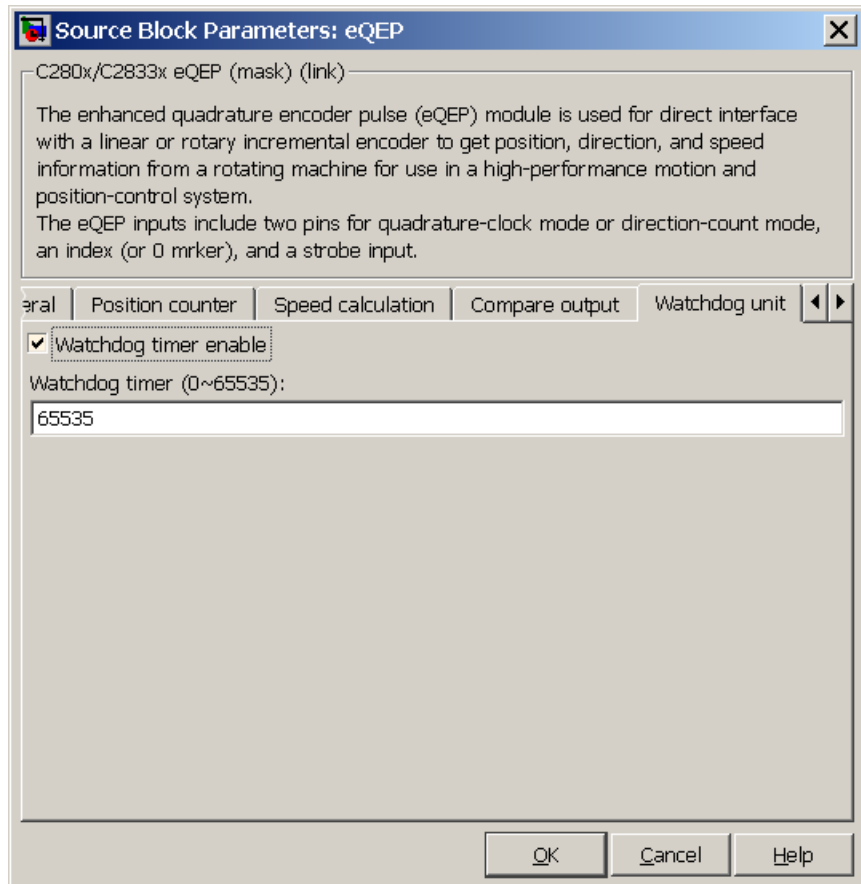
The pulse stretcher logic in the position-compare unit generates a programmable position-compare sync pulse output on the position-compare match.

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



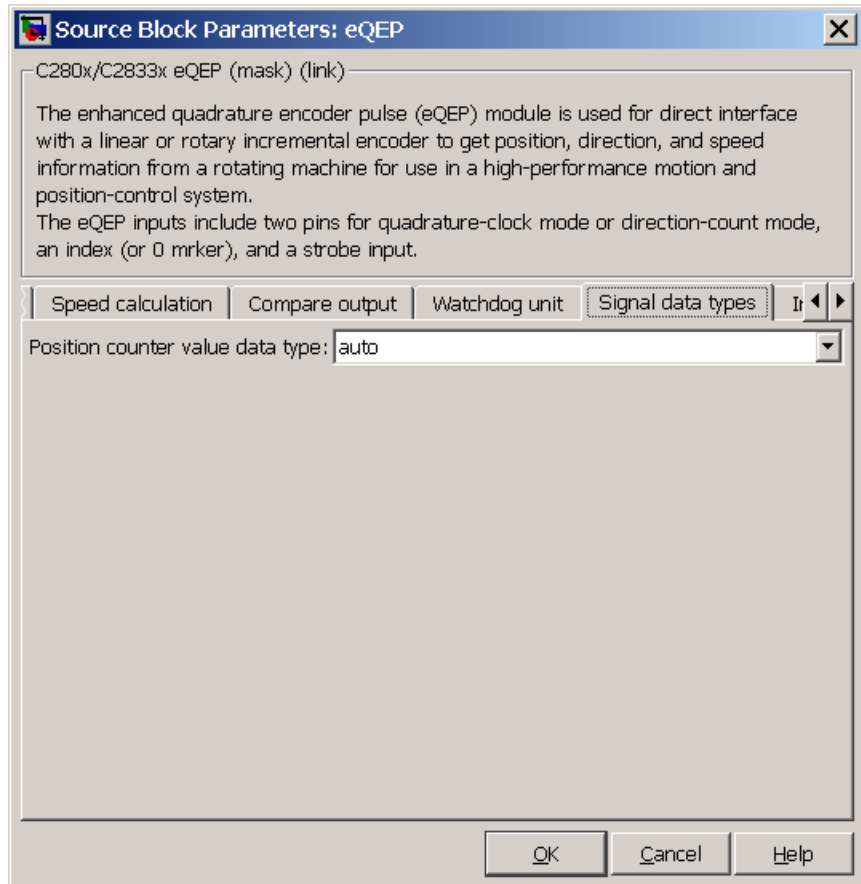
Enable watchdog time out flag via output port

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/C28x3x eQEP dialog box causes a corresponding popup to appear in the **Signal data types** pane.

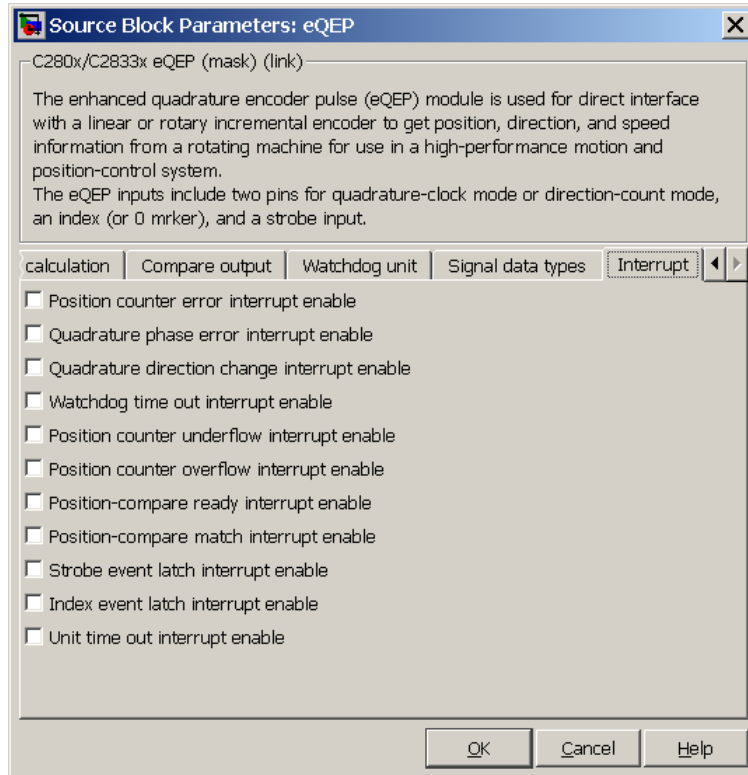
The following table summarizes the options for which you can set the data type in the **Signal data types** pane:

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.

References

For more information on the QEP module, consult the following documents, available at the Texas Instruments Web site:

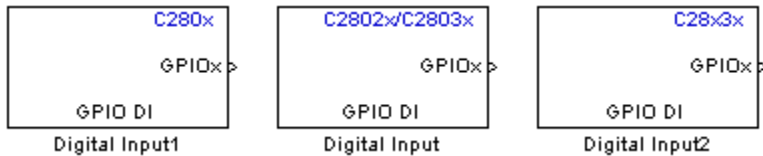
- *TMS320x280x, 2801x, 2804x Enhanced Quadrature Encoder Pulse (eQEP) Module Reference Guide*, Literature Number SPRU790
- *Using the Enhanced Quadrature Encoder Pulse (eQEP) Module in TMS320x280x, 28xxx as a Dedicated Capture Application Report*, Literature Number SPRAAH1

C280x/C28x3x/C2802x GPIO Digital Input

Purpose Configure general-purpose input pins

Library “C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



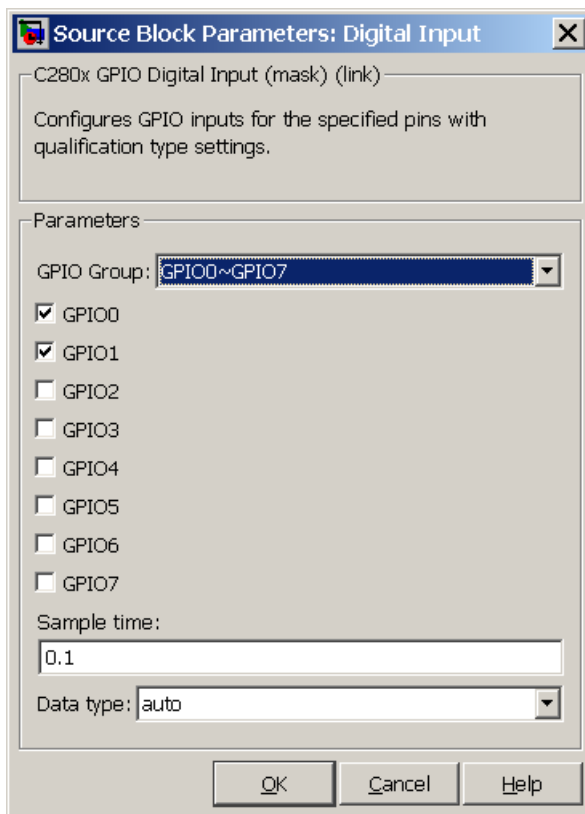
This block configures the general-purpose I/O (GPIO) MUX registers that control the operation of GPIO shared pins for digital input. Each I/O port has one MUX register that selects peripheral operation or digital I/O operation (the default). When a pin is configured for digital input, it becomes unavailable for digital output or peripheral operation. You can configure the **Input qualification type** for individual digital input pins. To do so, use the **Peripheral** tab of the Target Preferences block for your processor type.

Each processor has a different number of available GPIO pins:

- C280x has 35 GPIO pins
- C2802x has 22 GPIO pins, even though **GPIO group** lists 35
- C28x3x has 64 GPIO pins

Note To avoid losing any new settings, click **Apply** before changing the **GPIO Group** parameter.

Dialog Box



The dialog boxes for the C2802x and C28x3x processors are similar to that of the C280x, shown in the preceding figure.

GPIO Group

Select the group of GPIO pins you want to view or configure. For a table of GPIO pins and peripherals, refer to the Texas Instruments documentation for your specific target.

Sample time

Specify the time interval between output samples. To inherit sample time from the upstream block, set this parameter to -1.

C280x/C28x3x/C2802x GPIO Digital Input

For more information, refer to the section on “How to Specify the Sample Time” in the Simulink documentation.

Data type

Specify 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/C28x3x/C2802x GPIO Digital Output

C280x/C28x3x/C2802x GPIO Digital Output

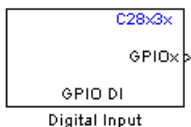
Purpose

Configure general-purpose input/output pins as digital outputs

Library

“C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

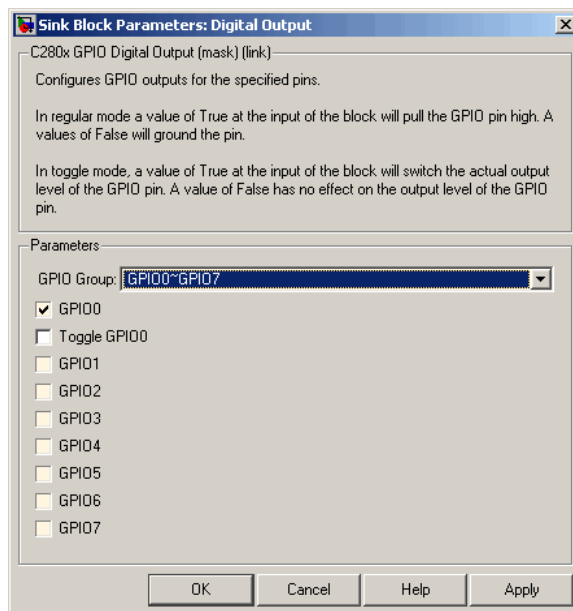
Description



Configure individual general-purpose input/output (GPIO) pins to operate as digital outputs. When a pin is configured for digital output, it cannot operate as a digital input or connect to peripheral I/O signals. When you select a pin for digital output, the user interface presents a **Toggle** option that inverts the output signal on the pin.

Note To avoid losing any new settings, click **Apply** before changing the **GPIO Group** parameter.

Dialog Box



C280x/C28x3x/C2802x GPIO Digital Output

The dialog boxes for the C2802x and C28x3x processors are similar to that of the C280x, shown in the preceding figure.

GPIO Group

Select the group of GPIO pins you want to view or configure.

GPIO pins for output

To configure a GPIO pin for digital output, select the checkbox next to it. Refer to the 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.

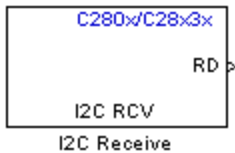
See Also

C280x/C28x3x/C2802x GPIO Digital Input

Purpose Configure inter-integrated circuit (I2C) module to receive data from I2C bus

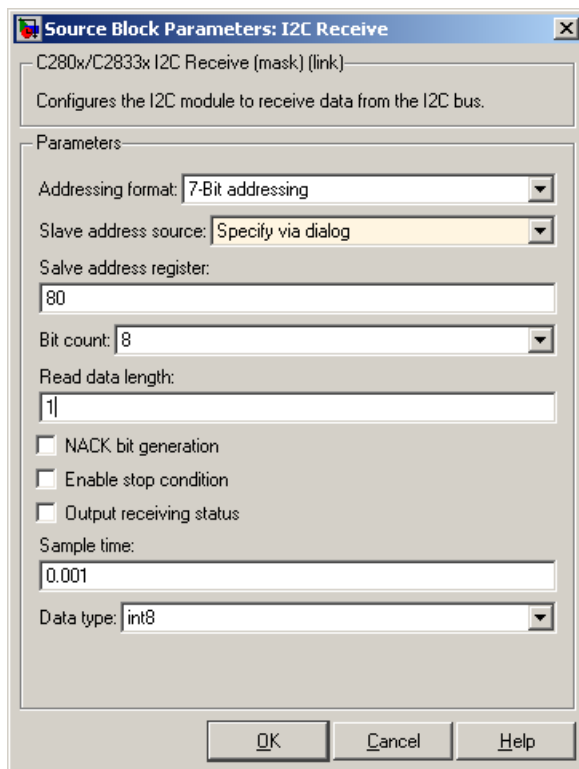
Library “C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description Configure the I2C module to receive data from the two-wire I2C serial bus.



C280x/C28x3x/C2802x I2C Receive

Dialog Box



Addressing format

The I2C receive block supports the **7–Bit addressing**, **10–Bit addressing**, and **Free data format**. The default setting is **7–Bit addressing**.

Slave address source

Select the method for setting the slave address register of the I2C slave. Selecting **Specify via dialog** displays **Slave address register** parameter. Selecting **Input port** enables definition of the address register via the input port. The default setting is **Specify via dialog**.

Slave address register

When you select **Specify via dialog**, enter a value for the **Slave address register**. The default value is **80**. This field takes a decimal value.

Bit Count

Set the bit count to 1 through 8. The default setting is **8**.

Read data length

Set the length of the read data. The default value is **1**.

NACK bit generation

Select this parameter to generate a no-acknowledge bit (NACK) during the I2C acknowledge cycle and ignore new bits from the transmitting I2C node. The default setting is disabled (not selected).

Enable stop condition

Enable the I2C Receive Block in master mode to send a STOP message to the I2C Transmit block while it is in slave mode. The default setting is disabled (not selected).

Output receiving status

Selecting this parameter creates a status output that indicates when the I2C receive block is receiving a message. The default setting is disabled (not selected).

Sample time

Set the sample time for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, and refer to "Asynchronous Interrupt Processing" on page 1-11 for a discussion of block placement and other necessary settings. The default value is **0.001**.

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. You can set this parameter to int8, uint8, int16, uint16, int32, or uint32. The default setting is **int8**.

C280x/C28x3x/C2802x I2C Receive

References

For detailed information on the I2C module, see:

- The *TMS320x28xx, 28xxx Inter-Integrated Circuit (I2C) Module Reference Guide*, Literature Number SPRU721, available at the Texas Instruments Web site, www.ti.com.
- The *Philips Semiconductors Inter-IC bus (I2C-bus) specification version 2.1* is available on the Philips Semiconductors Web site at http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf.

See Also

C280x/C28x3x/C2802x I2C Transmit

C280x/C28x3x/C2802x I2C Transmit

Purpose

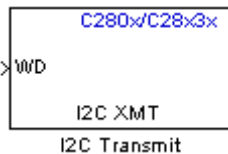
Configure inter-integrated circuit (I2C) module to transmit data to I2C bus

Library

“C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

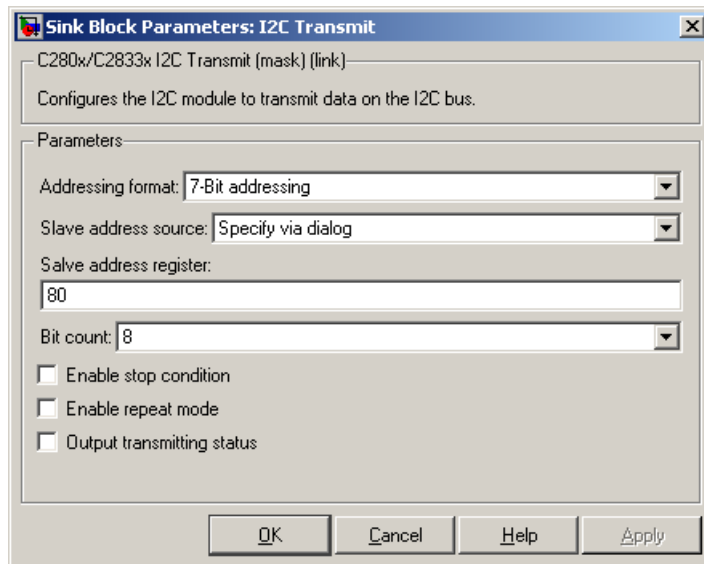
Description

Configure the I2C module to transmit data to the two-wire I2C serial bus. Also configure the



Note You can use this block to configure the I2C settings under the Peripherals tab of the target preference blocks for the Custom, F2808 eZdsp, and F28335 eZdsp boards.

Dialog Box



C280x/C28x3x/C2802x I2C Transmit

Addressing format

The I2C transmit block supports the **7-Bit addressing**, **10-Bit addressing**, and **Free data format**. The default setting is **7-Bit addressing**.

Slave address source

Select the method for setting the slave address register of the I2C slave. Selecting **Specify via dialog** displays **Slave address register** parameter . Selecting **Input port** enables definition of the address register via the input port. The default setting is **Specify via dialog**.

Slave address register

When you select **Specify via dialog**, enter a value for the **Slave address register**. The default value is **80**.

Bit Count

Set the bit count to 1 through 8. The default setting is **8**.

Enable stop condition

Selecting this parameter enables the transmitter to accept a STOP condition from the C280x/C28x3x/C2802x I2C Receive block. The default setting is disabled (not selected).

Enable repeat mode

Selecting this parameter enables repeat mode. The default setting is disabled (not selected).

Output transmitting status

Selecting this parameter creates a status output that indicates when the I2C transmit block is transmitting a message. The default setting is disabled (not selected).

References

For detailed information on the I2C module, see:

- The *TMS320x28xx, 28xxx Inter-Integrated Circuit (I2C) Module Reference Guide*, Literature Number SPRU721, available at the Texas Instruments Web site, www.ti.com.

- The *Philips Semiconductors Inter-IC bus (I2C-bus) specification version 2.1* is available on the Philips Semiconductors Web site at http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf.

See Also

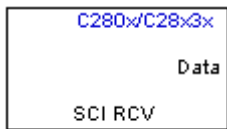
C280x/C28x3x/C2802x I2C Receive

C280x/C28x3x/C2802x SCI Receive

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

Library “C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description The C280x/C28x3x 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/C28x3x DSP target to receive scalar or vector data from the COM port via the C280x/C28x3x target’s COM port.

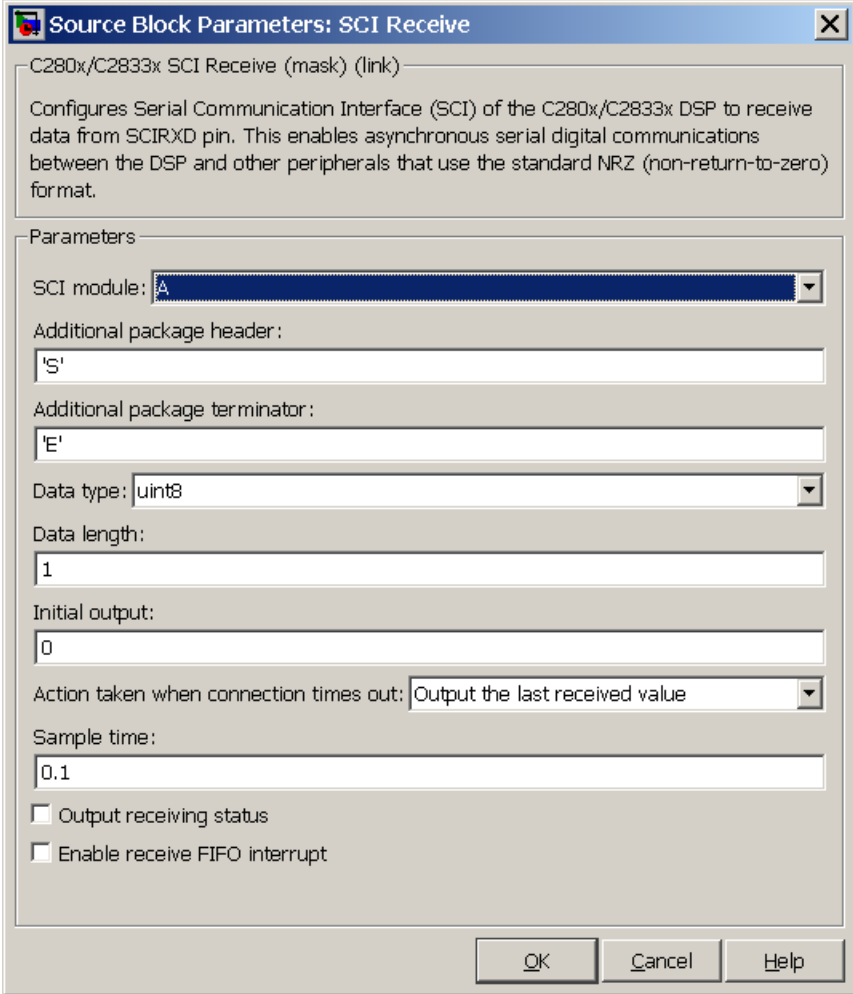


SCI Receive

Note For any given model, you can have only one C280x/C28x3x 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.

Dialog Box



Source Block Parameters: SCI Receive

C280x/C2833x SCI Receive (mask) (link)

Configures Serial Communication Interface (SCI) of the C280x/C2833x DSP to receive data from SCIRXD pin. This enables asynchronous serial digital communications between the DSP and other peripherals that use the standard NRZ (non-return-to-zero) format.

Parameters

SCI module: **A**

Additional package header: 'S'

Additional package terminator: 'E'

Data type: uint8

Data length: 1

Initial output: 0

Action taken when connection times out: Output the last received value

Sample time: 0.1

Output receiving status

Enable receive FIFO interrupt

OK Cancel Help

SCI module

SCI module to be used for communications.

C280x/C28x3x/C2802x SCI Receive

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. To specify a null value (no package header), enter two single quotes alone.

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. To specify a null value (no package terminator), enter two single quotes alone.

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

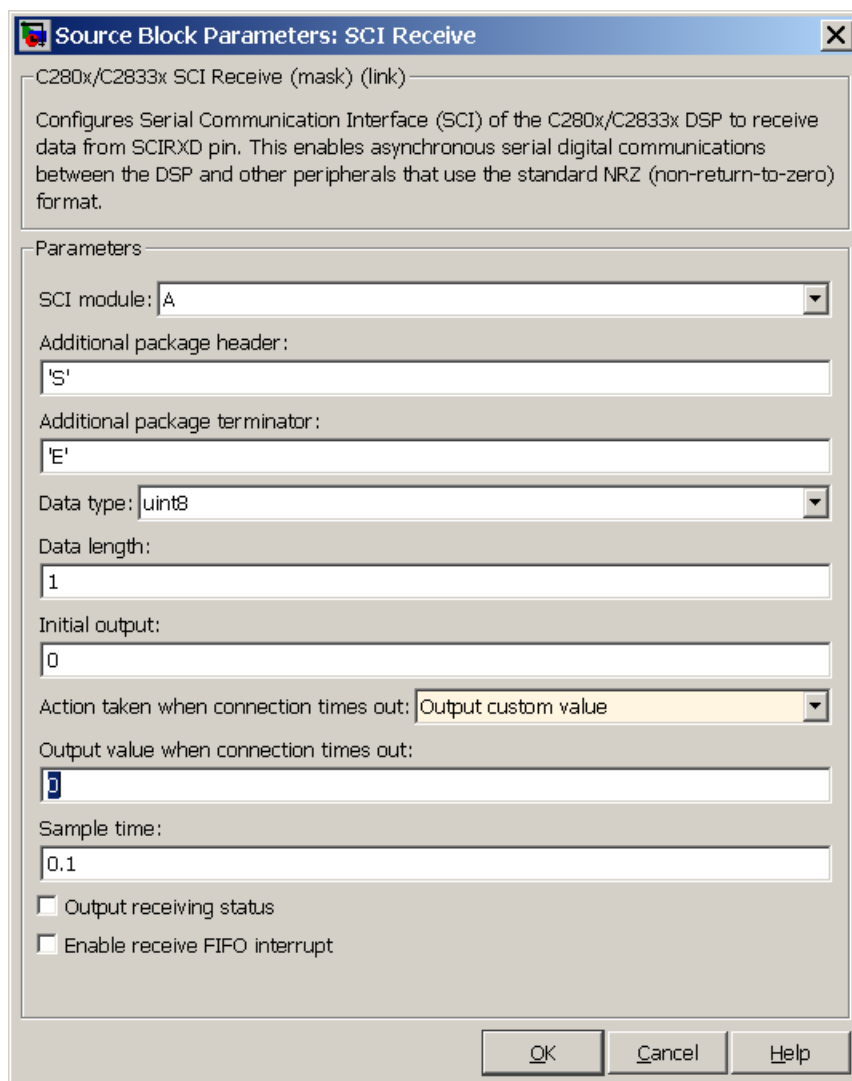
Default value from the C280x/C28x3x SCI Receive block. This value is used, for example, if a connection time-out occurs and the **Action taken when connection timeout** field is set to "Output the last received value", but nothing yet has been received.

Action taken when connection timeout

Specify 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 you select "Output custom value", use the "Output value when connection times out" field to set the custom value.

C280x/C28x3x/C2802x SCI Receive



Source Block Parameters: SCI Receive [X]

C280x/C2833x SCI Receive (mask) (link)

Configures Serial Communication Interface (SCI) of the C280x/C2833x DSP to receive data from SCIRXD pin. This enables asynchronous serial digital communications between the DSP and other peripherals that use the standard NRZ (non-return-to-zero) format.

Parameters

SCI module: A

Additional package header: 'S'

Additional package terminator: 'E'

Data type: uint8

Data length: 1

Initial output: 0

Action taken when connection times out: Output custom value

Output value when connection times out: 0

Sample time: 0.1

Output receiving status

Enable receive FIFO interrupt

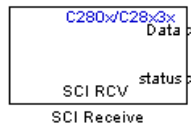
OK Cancel Help

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 1-11 for a discussion of block placement and other necessary settings.

Output receiving status

When this field is checked, the C280x/C28x3x 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).

C280x/C28x3x/C2802x SCI Receive

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

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

See Also

C280x/C28x3x/C2802x SCI Transmit, C280x/C2802x/C2803x/C28x3x Hardware Interrupt

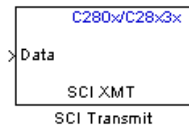
Purpose

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

Library

“C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



The C280x/C28x3x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C280x/C28x3x 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`.

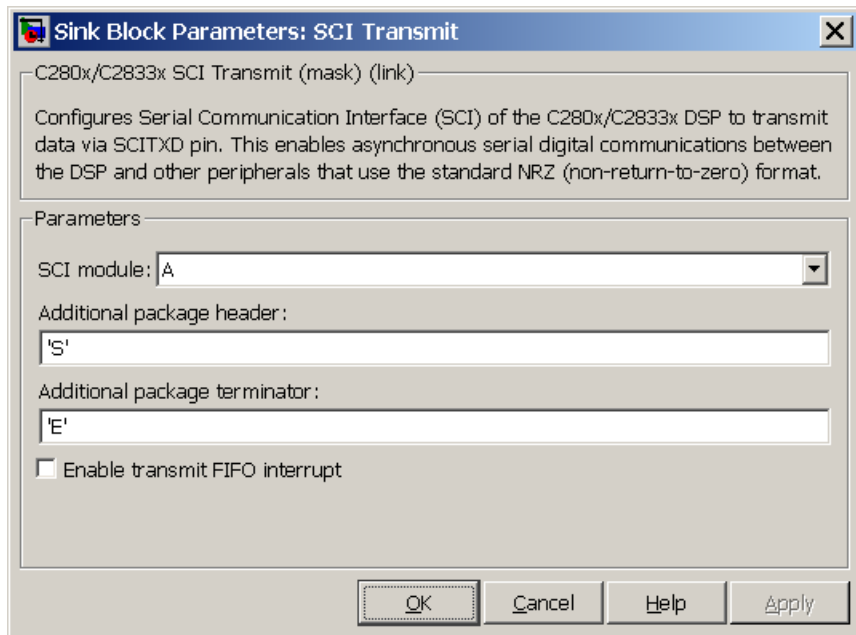
Note For any given model, you can have only one C280x/C28x3x SCI Transmit block per module. There are two modules, A and B, which can be configured through the F2808 eZdsp Target Preferences block.

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.

C280x/C28x3x/C2802x SCI Transmit

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. To specify a null value (no package header), enter two single quotes alone.

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. To specify a null value (no package terminator), enter two single quotes alone.

Enable transmit FIFO interrupt

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

References

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

See Also

C280x/C28x3x/C2802x SCI Receive, C280x/C2802x/C2803x/C28x3x Hardware Interrupt

C280x/C28x3x/C2802x Software Interrupt Trigger

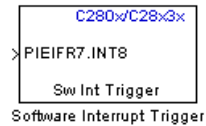
Purpose

Generate software triggered nonmaskable interrupt

Library

“C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

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.

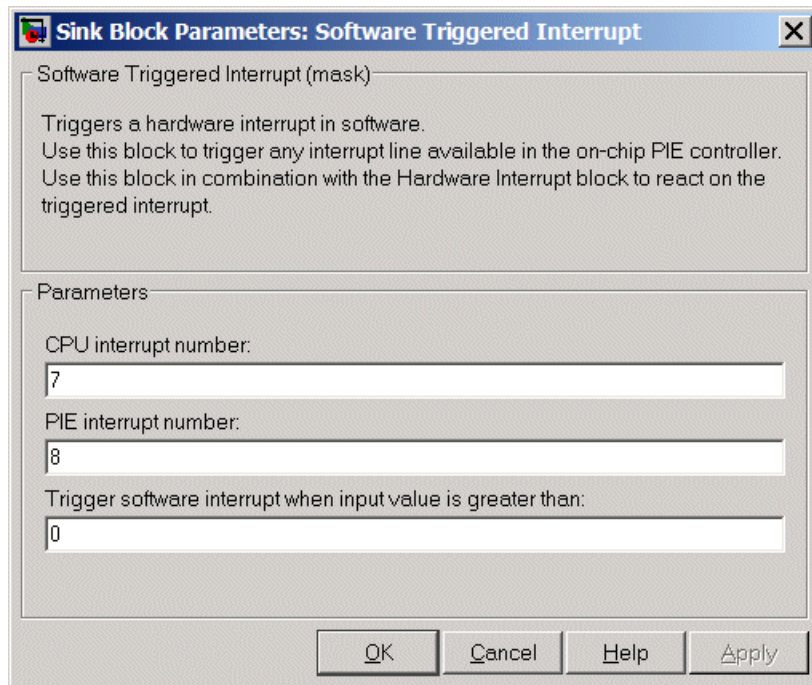
To use this block, add a Hardware Interrupt block to your model to process the software triggered interrupt from this block into an interrupt service routine on the processor. Set the interrupt number in the Hardware Interrupt block to the value you set here in **CPU interrupt number**.

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.

C280x/C28x3x/C2802x Software Interrupt Trigger

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/C28x3x/C2802x Software Interrupt Trigger

References

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

See Also

C280x/C2802x/C2803x/C28x3x Hardware Interrupt

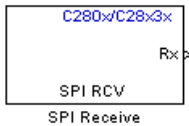
Purpose

Receive data via serial peripheral interface (SPI) on target

Library

“C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



The C280x/C28x3x 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.

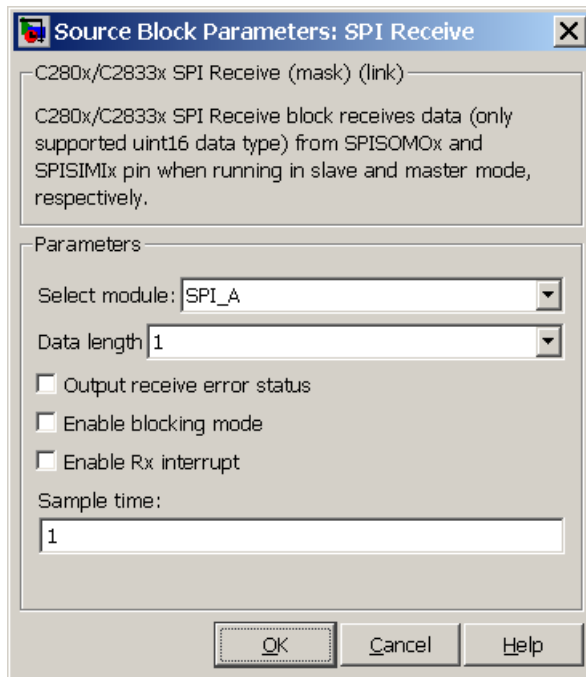
In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which 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/C28x3x 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/C28x3x/C2802x SPI Receive

Dialog Box



Select module

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

Data length

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

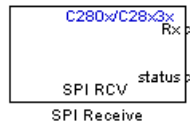
Enable blocking mode

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/C28x3x SPI Receive block adds another output port for the transaction status, and appears as shown in the following figure.

C280x/C28x3x/C2802x SPI Receive



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 1-11 for a discussion of block placement and other necessary settings.

See Also

C280x/C28x3x/C2802x SPI Transmit, C280x/C2802x/C2803x/C28x3x Hardware Interrupt

C280x/C28x3x/C2802x SPI Transmit

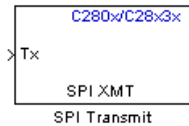
Purpose

Transmit data via serial peripheral interface (SPI) to host

Library

“C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, and “C28x3x Chip Support (c2833xlib)” on page 6-8

Description



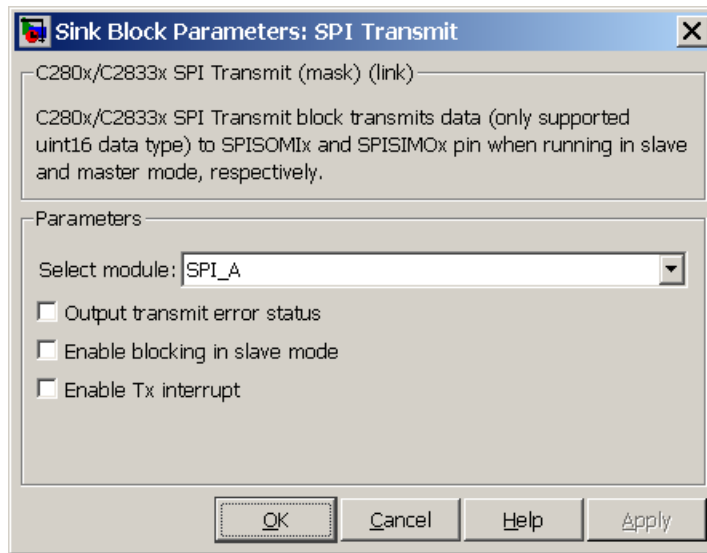
The C280x/C28x3x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives 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/C28x3x 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

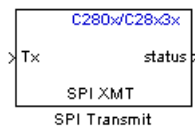


Select module

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

Output transmit error status

When this field is checked, the C280x/C28x3x 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/C28x3x/C2802x 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/C28x3x/C2802x SPI Receive, C280x/C2802x/C2803x/C28x3x Hardware Interrupt

Purpose

Compare two input voltages on comparator pins

Library

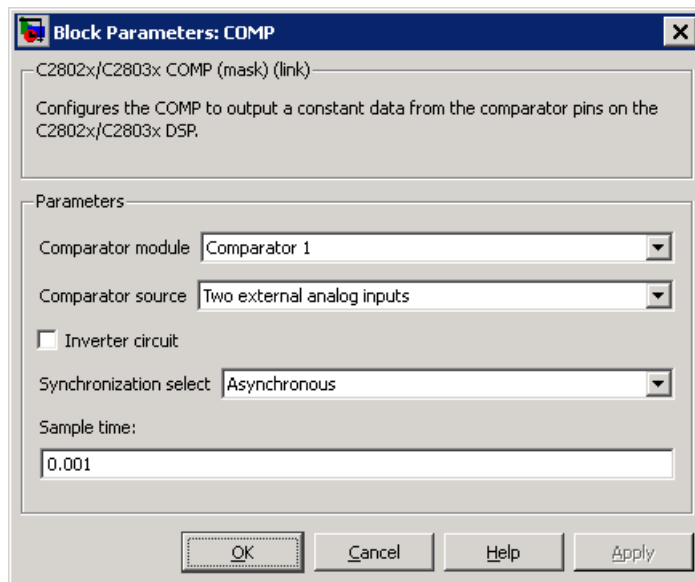
“C2802x Chip Support (c2802xlib)” on page 6-4

Description

Configures the COMP to output a constant data from the comparator pins on the DSP.



Dialog Box



Comparator module

Select which comparator module the block configures. Use only one block per module.

Comparator source

Select Two external analog inputs to compare the voltage of **Input Pin A** with **Input Pin B**. Select One external analog

inputs to compare the voltage of **Input Pin A** with the output of a DAC reference located in the comparator. For more information, see the “DAC Reference” section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator*.

The comparator source outputs 1 if **Input Pin A** has a value greater than **Input Pin B** or the 10-bit DAC reference. Otherwise it outputs 0.

Inverter circuit

Apply a logical NOT to the output of the comparator source. For example, when the comparator source outputs 1, the inverter circuit changes it to 0.

Synchronization select

Select **Asynchronous** to pass the asynchronous version of the comparator output. Select **Synchronous** to pass the synchronous version of the comparator output. Selecting **Synchronous** enables the **Qualification period** option.

Qualification period

Qualify changes in the comparator output before passing them along. The **Passed through** setting passes changes in the comparator value along without qualifying them. The **consecutive clocks** settings pass changes in the comparator value along after receiving the specified number of consecutive samples with the same value. Use this setting to prevent intermittent and spurious changes in the comparator output.

Sample time

Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

References

TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator, Literature Number: SPRUGE5, from the Texas Instruments Web site.

Purpose

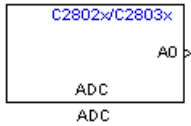
Configure ADC to sample analog pins and output digital data

Library

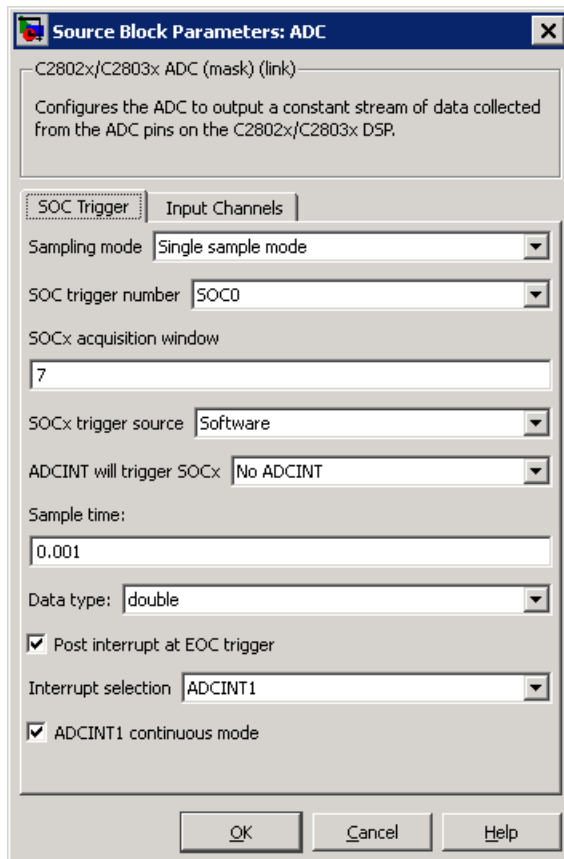
“C2802x Chip Support (c2802xlib)” on page 6-4

Description

Configures the ADC to output a constant stream of data collected from the ADC pins on the DSP.



Dialog Box



Sampling mode

Select **Single sample mode** to sample two signals sequentially. Select **Simultaneous sample mode** to sample the two signals with a minimal delay between the samples.

SOC trigger number

Identify the start-of-conversion trigger by number. In single sampling mode, you can select an individual trigger. In simultaneous sampling mode, you can select triggers in pairs.

SOCx acquisition window

Define the length of the acquisition period, the acquisition window, in sample cycles. The minimal value for this parameter is 7 cycles. For more information, see the “ADC Acquisition (Sample and Hold) Window” section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator Reference Guide*.

SOCx trigger source

Select one of the following input trigger for the start of conversion:

- Software
- CPU Timers 0/1/2 interrupts
- XINT2 SOC
- ePWM1-7 SOCA and SOCB

ADCINT will trigger SOCx

At the end of conversion, use the ADCINT1 or ADCINT2 interrupt to trigger a start of conversion (SOC). This loop creates a continuous sequence of conversions. The default selection, No ADCINT disables this parameter.

Sample time

Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

Data type

Select the data type of the digital output data. You can choose from the options double, single, int8, uint8, int16, uint16, int32, and uint32.

Post interrupt at EOC trigger

Post interrupts when the ADC triggers EOC pulses. When you select this option, the dialog box displays the **Interrupt selection** and **ADCINT# continuous mode** options. For more information, see the “EOC and Interrupt Operation” section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator Reference Guide*.

Interrupt selection

Select which interrupt the ADC posts after triggering an EOC pulse.

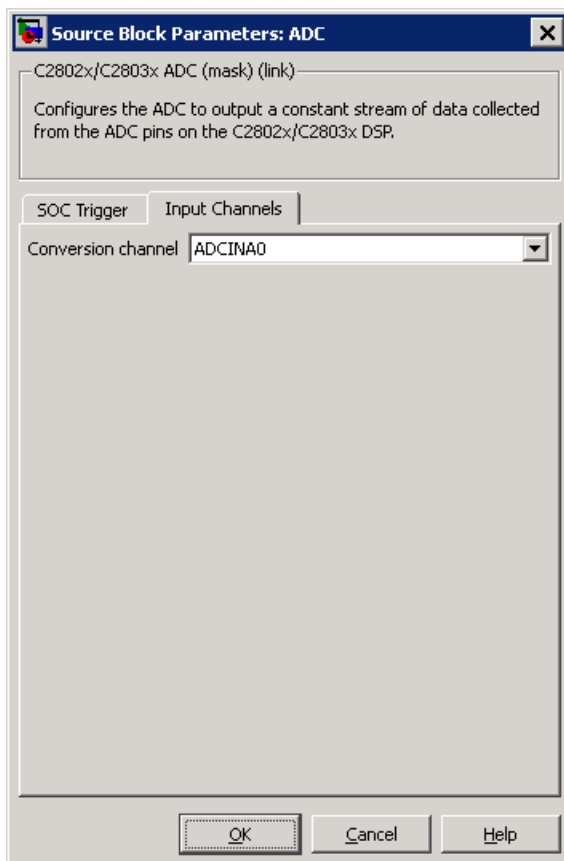
ADCINT1 continuous mode

ADCINT2 continuous mode

When the ADC generates an end of conversion (EOC) signal, generate an ADCINT# interrupt whether the previous interrupt flag has been acknowledged or not.

Input Channels — Conversion channel

Select the input channel to which this ADC conversion applies.



References

TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator, Literature Number: SPRUGE5, from the Texas Instruments Web site.

C2802x AnalogIO Input

Purpose

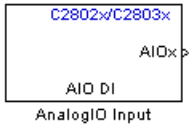
Configure pin, sample time, and data type for analog input

Library

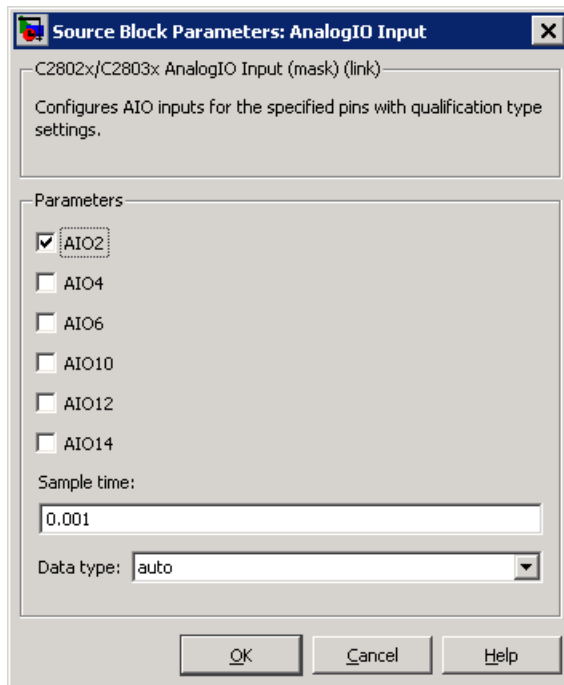
“C2802x Chip Support (c2802xlib)” on page 6-4

Description

Use this block to sample the Analog IO input pins on the C2802x processor for a positive voltage and output the results.



Dialog Box



Parameters (Input pins)

Select the input pins to sample.

Sample time

Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

Data type

Select the data type of the digital output data. If you select auto, the block automatically selects the correct data type for your model. You can also manually select a data type. You can choose from the options double, single, int8, uint8, int16, uint16, int32, and uint32.

See Also

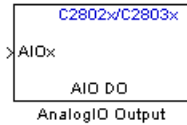
C2802x AnalogIO Output

C2802x AnalogIO Output

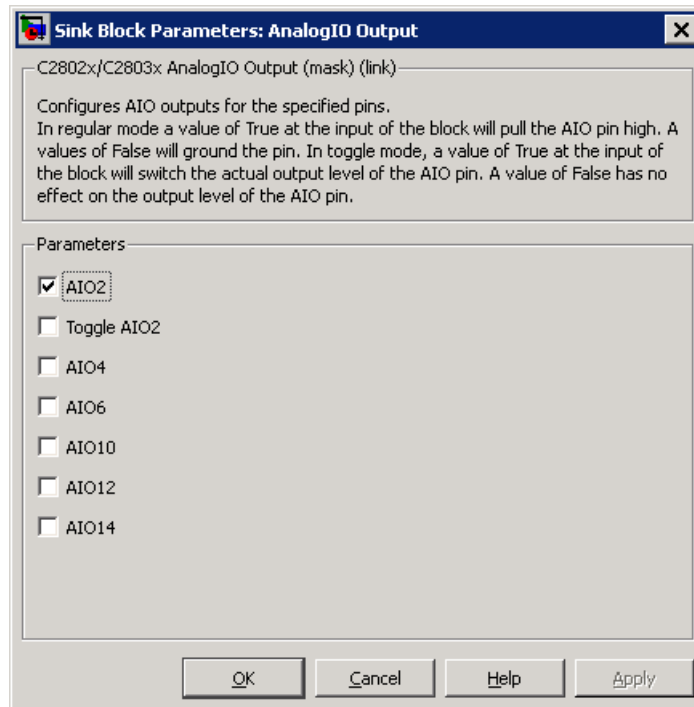
Purpose Configure Analog IO to output analog signals on specific pins

Library “C2802x Chip Support (c2802xlib)” on page 6-4

Description Configures the Analog IO output pins for the specified pins. In regular mode, a value of True at the input of the block pulls the Analog IO pin high. A value of False grounds the pin. In toggle mode, a value of True at the input of the block switches the actual output level of the Analog IO pin. A value of False does not affect on the output level of the Analog IO pin.



Dialog Box



Parameters (Output Pins)

Select the analog output pins that express the value of the digital input on **AIOx**. Selecting **Toggle** inverts the output voltage levels of the pins.

See Also

C2802x AnalogIO Input

C2802x ePWM

Purpose

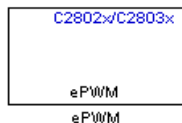
Generate Enhanced Pulse Width Modulator (ePWM) waveforms

Library

“C2802x Chip Support (c2802xlib)” on page 6-4

Description

Configures the Event Manager of the C2802x DSP to generate ePWM waveforms. A C2802x system contains multiple ePWM modules, each having two PWM outputs. You can use the C2802x ePWM block to configure up to four ePWM modules.



Dialog Box

General Pane

The image shows a software dialog box titled "Block Parameters: ePWM". The window title bar includes a close button (X). Below the title bar, there is a subtitle "C2802x/C2803x ePWM (mask) (link)" and a descriptive text: "Configures the Event Manager of the C2802x DSP to generate ePWM waveforms." The dialog has several tabs: "General", "ePWMA", "ePWMB", "Deadband unit", and "Event Trigger". The "General" tab is selected. A dropdown menu for "Module:" is set to "ePWM1". Below this are several configuration fields: "Timer period units:" is set to "Clock cycles"; "Timer period source:" is set to "Specify via dialog"; "Timer period:" is a text field containing "64000"; "Counting mode:" is set to "Up-Down"; "Sync output selection:" is set to "Disable". There are three unchecked checkboxes: "Add S/W sync input port", "Enable DCAEVT1 sync", and "Enable DCBEVT1 sync". Below these are three more dropdown menus: "Phase offset source:" set to "Disable", "TB clock prescaler divider:" set to "1", and "High Speed TB clock prescaler divider:" set to "1". At the bottom, there are four buttons: "OK", "Cancel", "Help", and "Apply".

Block Parameters: ePWM

C2802x/C2803x ePWM (mask) (link)

Configures the Event Manager of the C2802x DSP to generate ePWM waveforms.

General | ePWMA | ePWMB | Deadband unit | Event Trigger

Module: ePWM1

Timer period units: Clock cycles

Timer period source: Specify via dialog

Timer period: 64000

Counting mode: Up-Down

Sync output selection: Disable

Add S/W sync input port

Enable DCAEVT1 sync

Enable DCBEVT1 sync

Phase offset source: Disable

TB clock prescaler divider: 1

High Speed TB clock prescaler divider: 1

Enable swap module A and B

Enable HRPWM (Period)

Enable HRPWM (CMP)

OK Cancel Help Apply

Module

Specify which target ePWM module to use.

Timer period units

Specify the units of the **Timer period** or **Timer initial period** as **Clock cycles** (the default) or **Seconds**. If you set **Timer period units** to **Seconds**, the software must down-convert the **Timer period** or **Timer initial period**, a double, for the period register, a uint16. For best performance, select **Clock cycles**. Doing so reduces calculations and rounding errors.

Note If you set **Timer period units** to **Seconds**, enable support for floating-point numbers. In the model window, select **Simulation > Configuration Parameters**. In the Configuration Parameters dialog box, select **Real-Time Workshop > Interface**. Under **Software Environment**, enable **floating-point numbers**.

Timer period source

Configure the source of the timer period value. Selecting **Specify via dialog** changes the following parameter to **Timer period**. Selecting **Input port** changes the following parameter to **Timer initial period** and creates a timer period input port, **T**, on the block.

Timer period

Set the period of the PWM waveform in clock cycles or in seconds, as determined by the **Timer period units** parameter. When you enable HRMWM, you can enter a high-precision floating point value. The Time-Base Period High Resolution Register (TBPRDHR) stores the high-resolution portion of the timer period value.

Note The term *clock cycles* refers to the Time-base Clock on the C280x/C28x3x chip. See the discussion of the **TB clock prescaler divider** for an explanation of how to calculate the Time-base Clock speed.

Timer initial period

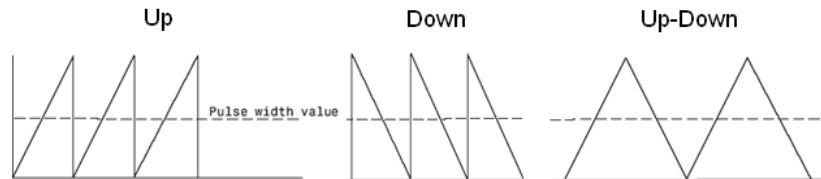
The period of the waveform from the time the PWM peripheral starts operation until the ePWM input port, **T**, receives a new value for the period. Use **Timer period units** to measure the period in clock cycles or in seconds.

Note The term *clock cycles* refers to the Time-base Clock on the C280x/C28x3x chip. See the discussion of the **TB clock prescaler divider** for an explanation of how to calculate the Time-base Clock speed.

Counting mode

Specify the counting mode in which to operate. C280x/C28x3x PWMs can operate in three distinct counting modes: Up, Down, and Up-Down. The Down option is not compatible with HRPWM. To avoid generating an error, do not select Down when you enable **HRPWM (Period)**.

The following illustration shows the waveforms that correspond to these three modes:



Sync output selection

This parameter corresponds to the SYNCOSSEL field in the Time-Base Control Register (TBCTL).

Use this parameter to specify the event that generates a Time-base synchronization output signal, EPWMxSYNCO, from the Time-base (TB) submodule.

The available choices are:

- EPWMxSYNCI or SWFSYNC — a Synchronization input pulse or Software forced synchronization pulse, respectively. You can use this option to achieve precise synchronization across multiple ePWM modules by daisy chaining multiple the Time-base (TB) submodules.
- CTR=Zero — Time-base counter equal to zero (TBCTR = 0x0000)
- CTR=CMPB — Time-base counter equal to counter-compare B (TBCTR = CMPB)
- Disable — Disable the EPWMxSYNCO output (the default)

Add S/W sync input port

Create an input port, **SYNC**, for a Time-base synchronization input signal, EPWMxSYNCI. You can use this option to achieve precise synchronization across multiple ePWM modules by daisy-chaining multiple the Time-base (TB) submodules. This option is not compatible with HRPWM. Enabling HRPWM disables this option.

Enable DCAEVT1 sync

Synchronize the ePWM time base to a DCAEVT1 digital compare event. Use this feature to synchronize this PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**. This option is not compatible with HRPWM. Enabling HRPWM disables this option.

Enable DCBEVT1 sync

Synchronize the ePWM time base to a DCBEVT1 digital compare event. Use this feature to synchronize this PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**. This option is not compatible with HRPWM. Enabling HRPWM disables this option.

Phase offset source

Specify the source of a phase offset to apply to the Time-base synchronization input signal, EPWMxSYNCI from the **SYNC** input port. Selecting **Specify via dialog** creates the **Phase offset value** parameter. Selecting **Input port** creates a phase input port, **PHS**, on the block. Selecting **Disable**, the default value, prevents the software from applying phase offsets to the TB module.

Counting direction after phase synchronization

This parameter appears when you set **Counting Mode** to **Up-Down** and when you enable **Phase offset source** (set to **Specify via dialog** or **Input port**). Configure the timer to count up from zero, or down to zero, following synchronization. This parameter corresponds to the PHSDIR field of the Time-base Control Register (TBCTL).

Phase offset value

This field appears when you select **Specify via dialog** in **Phase offset source**.

Configure the phase offset (delay) between the following events:

- The arrival of the Time-base synchronization input signal (EPWMxSYNCl) on the **SYNC** input port
- The moment the Time-base (TB) submodule synchronizes the ePWM module.

Note Enter the **Phase offset value** in TBCLK cycles, from 0 to 65535. Do not use fractional seconds.

This parameter corresponds to the Time-Base Phase Register (TBPHS).

TB clock prescaler divider

Use the **TB clock prescaler divider** (CLKDIV) and the **High Speed TB clock prescaler divider** (HSPCLKDIV) to configure the ePWM time-base clock speed (TBCLK). Use the following equation:

$$\text{TBCLK} = \text{SYSCLKOUT}/(\text{HSPCLKDIV} * \text{CLKDIV})$$

For example, the default values of both CLKDIV and HSPCLKDIV are 1, and the default frequency of SYSCLKOUT is 100 MHz, so:

$$\text{TBCLK} = 100 \text{ MHz} = 100 \text{ MHz}/(1 * 1)$$

The choices for the **TB clock prescaler divider** are: 1, 2, 4, 8, 16, 32, 64, and 128. Selecting **Enable HRPWM (Period)** forces this option to 1.

The **TB clock prescaler divider** parameter corresponds to the CLKDIV field of the Time-base Control Register (TBCTL).

Note The frequency of SYSCLKOUT depends on the oscillator frequency and the configuration of PLL-based clock module. Changing the values of the PLL Control Register (PLLCR) affects the timing of all ePWM modules.

For more information, consult the “PLL-Based Clock Module” section of the data manual for your specific target (see “References” on page 7-95).

High Speed TB clock prescaler divider

See **TB clock prescaler divider** for an explanation of how to use this value to setting the speed of the Time-base Clock. Choices are to divide by 1, 2, 4, 6, 8, 10, 12, and 14. Selecting **Enable HRPWM (Period)** forces this option to 1.

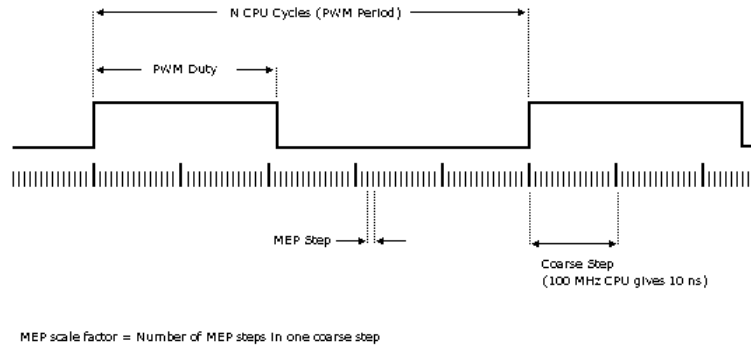
This parameter corresponds to the HSPCLKDIV field of the Time-base Control Register (TBCTL).

Enable swap module A and B

Swap the ePWMA and ePWMB outputs. This option outputs the ePWMA signals on the ePWMB outputs and the ePWMB signals on the ePWMA outputs.

Enable HRPWM (Period)

When the effective resolution for conventionally generated PWM is insufficient, consider using 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)** technology to position edges more finely by dividing each coarse system clock. The accuracy of the subdivision is on the order of 150ps. The following figure shows the relationship between one system clock and edge position in terms of **MEP** steps:



Enable HRPWM mode and control it via the Extension Register for HRPWM Period (TBPRDHR) register. When you enable this parameter, you can enter an 8-bit floating point value in for the **Timer period** parameter. This parameter enables the **Enable HRPWM (CMP)** option, and displays the **HRPWM loading mode**, **HRPWM control mode**, and **HRPWM edge control mode** options. Also configure **HRPWM control mode**.

Selecting Enable HRPWM (Period) forces **TB clock prescaler divider** and **High Speed TB clock prescaler divider** to 1. These settings match the HRPWM time base clock with the SYSCLKOUT frequency.

Enable HRPWM (CMP)

Enable HRPWM mode and control it via the Extension Register for HRPWM Duty (CMPAHR) register. Also configure **HRPWM control mode**.

HRPWM loading mode

Determine when to transfer the value of the CMPAHR shadow to the active register:

- **CTR=ZERO**: Transfer the value when the time base counter equals zero (TBCTR = 0x0000).

- **CTR=PRD:** Transfer the value when the time base counter equals the period (TBCTR = TBPRD).
- **CTR=Zero or CTR=PRD** Transfer the value when either case is true.

This option configures the HRLOAD “Shadow Mode Bit” in the HRPWM Configuration Register (HRCNFG).

HRPWM control mode

Select which register controls the Micro Edge Positioner (MEP) step size. The **HRPWM control mode** option configures the CTLMODE “Control Mode Bits”.

- **Duty control mode** uses the Extension Register for HRPWM Duty (CMPAHR) or the Extension Register for HRPWM Period (TBPRDHR) to control the MEP edge position.
- **Select Phase control mode** to use the Time Base Period High-Resolution Register (TBPRDHR) to control the MEP edge position.

The **HRPWM control mode** option configures the CTLMODE “Control Mode Bits” in the HRPWM Configuration Register (HRCNFG).

HRPWM edge control mode

Swap the ePWMA and ePWMB outputs. This parameter sets the SWAPAB field in the HRPWM Configuration Register (HRCNFG).

Use scale factor optimizer (SFO) software

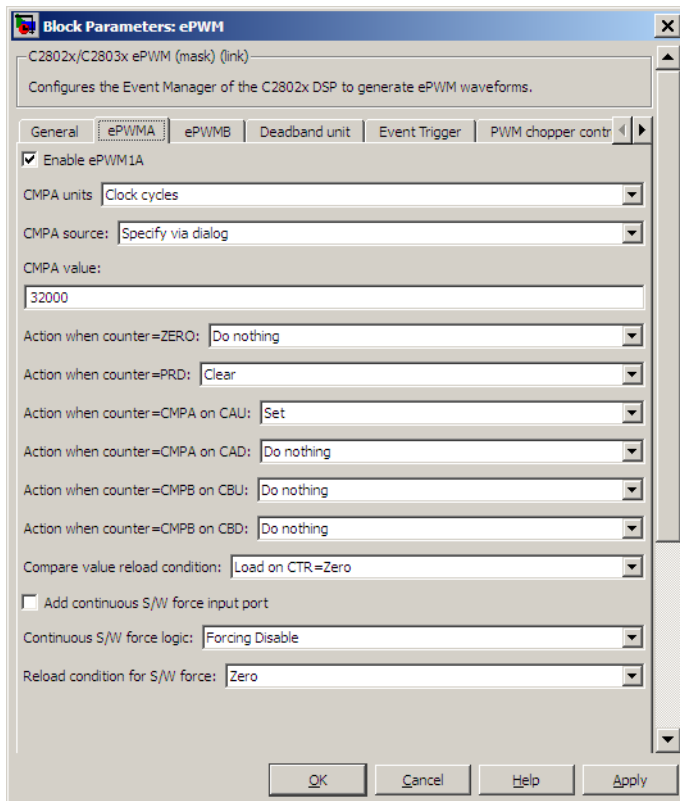
Enable scale factor optimizing (SFO) software with HRPWM. This software dynamically determines the appropriate scaling factor for the Micro Edge Positioner (MEP) step size. The step size varies depending on operating conditions such as temperature and voltage. The SFO software reduces variability due to these conditions. For more information, see the “Scale Factor Optimizing Software (SFO)” section of the *TMS320x2802x, 2803x Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide*, Literature Number: SPRUGE8.

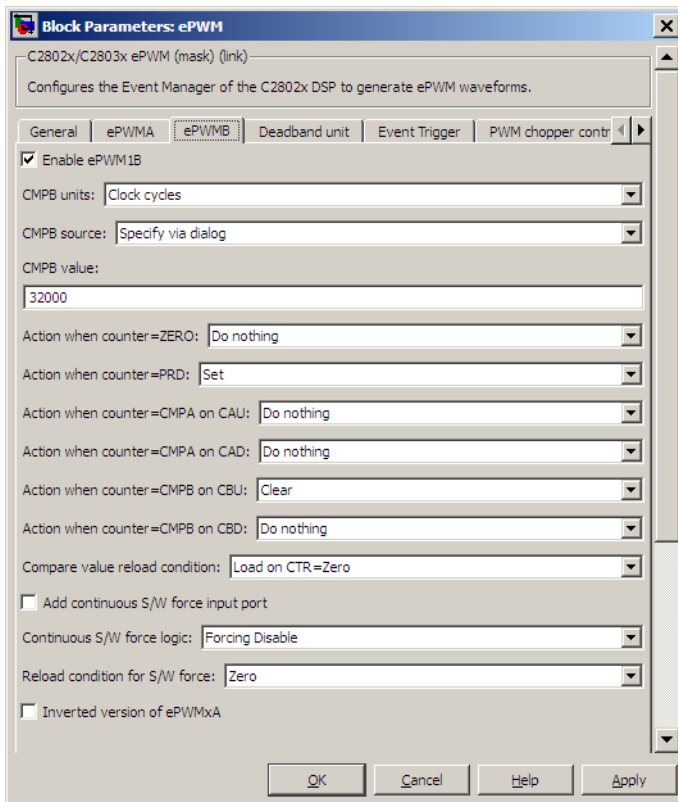
Enable auto convert

Apply the scaling factor calculated by the SFO software to the controlling period or duty cycle. (Use the **HRPWM control mode** to select controlling period or duty cycle.) This parameter sets the AUTOCONV field in the HRPWM Configuration Register (HRCNFG).

ePWMA and ePWMB panes

Each ePWM module has two outputs, ePWMA and ePWMB. The **ePWMA output** pane and **ePWMB output** pane have similar options.





Enable ePWMxA

Enable ePWMxB

Enables the ePWMA and/or ePWMB output signals for the ePWM module identified on the **General** pane. By default, the software enables ePWMxA and disables ePWMxB.

Note To **Enable ePWMxA** or **Enable ePWMxA**, also enable support for floating-point numbers. In the model window, select **Tools > Real Time Workshop > Options**. In the Configuration Parameters dialog box, select **Real-Time Workshop > Interface**. Under **Software Environment**, enable **floating-point numbers**.

CMPA units

CMPB units

Specify the units used by the compare register: Percentages (the default) or Clock cycles.

Notes

- The term *clock cycles* refers to the Time-base Clock on the C280x/C28x3x chip. See **TB clock prescaler divider** for an explanation of how to calculate the Time-base Clock speed.
 - Percentages use additional computation time in generated code and can decrease performance.
 - If you set **CMPA units** or **CMPB units** to Percentages, also enable support for floating-point numbers. In the model window, select **Simulation > Configuration Parameters**. In the Configuration Parameters dialog box, select **Real-Time Workshop > Interface**. Under **Software Environment**, enable **floating-point numbers**.
-

CMPA source

CMPB source

Specify the source of the pulse width value. If you select Specify via dialog (the default), enter a value in the **CMPA value** or **CMPB value** field. If you select Input port, set the value using

an input port, **WA** or **WB**, on the block. If you select Input port also set **CMPA initial value** or **CMPB initial value**.

CMPA value

CMPB value

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

CMPA initial value

CMPB initial value

This field appears when you set **CMPA source** or **CMPB source** to Input port. Enter the initial pulse width of CMPA or CMPB the PWM peripheral uses when it starts operation. Subsequent inputs to the **WA** or **WB** ports change the CMPA or CMPB pulse width.

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

The dialog box displays or hides these parameters, depending on the **Counting mode** selection (up, down, and up/down). 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. This option produces 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	Do nothing	Do nothing
PRD	Clear	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, consult 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 to reload the Action-qualifier S/W Force Register from a shadow register. Choices are Load on CTR=Zero (the default), Load on CTR=PRD, Load on either, and Freeze.

Add continuous S/W force input port specifies the source from which to get the control logic. 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 the type of S/W force logic to apply in the absence of a continuous S/W force input port. 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.

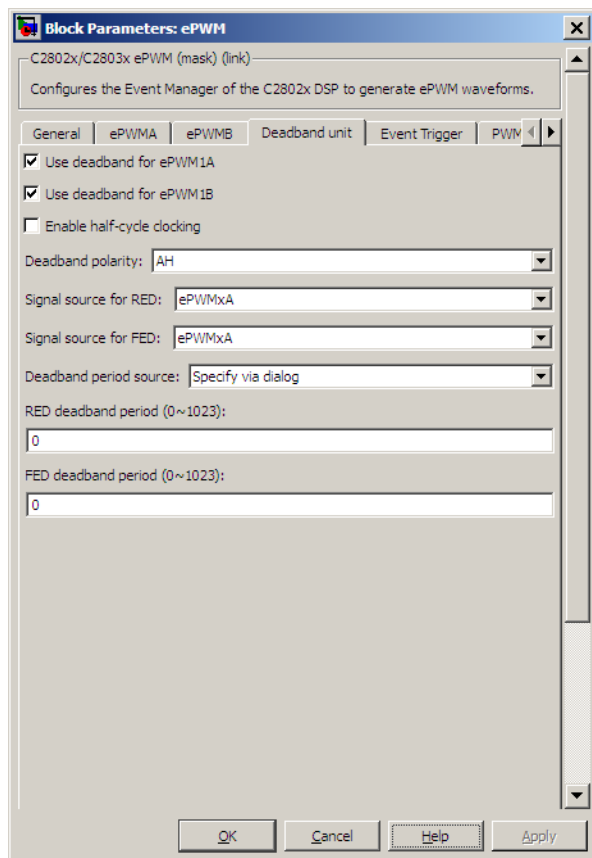
Inverted version of ePWMxA

Only the ePWMB pane displays this option. Invert the ePWMxA signal and output it on the ePWMxB outputs. This parameter sets the SELOUTB field in the HRPWM Configuration Register (HRCNFG).

Deadband Unit Pane

The **Deadband unit** pane lets you specify parameters for the Dead-Band Generator (DB) submodule.

C2802x ePWM



Use deadband for ePWMxA

Use deadband for ePWMxB

Enables a deadband area of no signal overlap between pairs of ePWM output signals. This check box is cleared by default.

Enable half-cycle clocking

To double the deadband resolution, enable half-cycle clocking. This option clocks the deadband counters at $TBCLK*2$. When you

disable this option, the deadband counters use full-cycle clocking (TBCLK*1).

Deadband polarity

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

Deadband period source

Specify the source of the control logic. Choose **Specify via dialog** (the default) to enter explicit values, or **Input port** to use a value from the input port.

RED deadband period

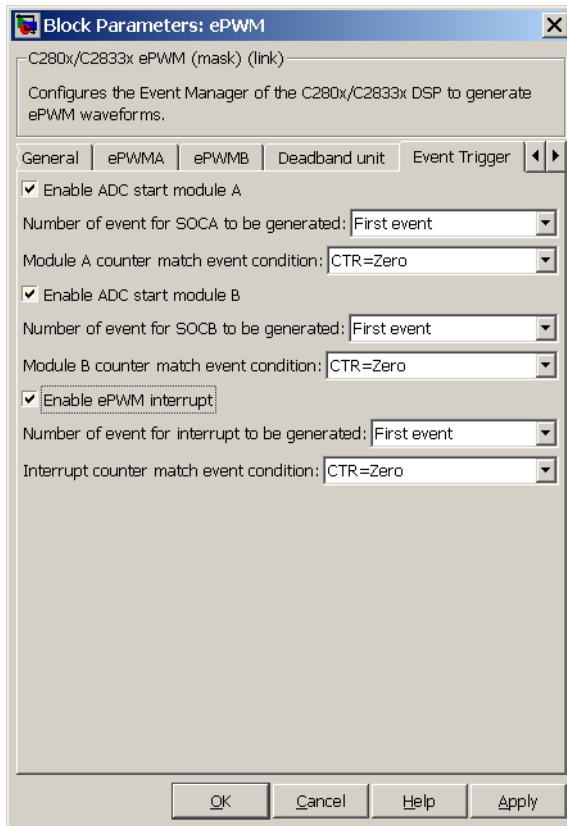
This field appears only when you select **Use deadband for ePWMxA** in the **ePWMA output** pane. Enter a value from 0 to 1023 to specify a rising edge delay.

FED deadband period

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

Event Trigger Pane

Configure ADC Start of Conversion (SOC) by one or both of the ePWMA and ePWMB outputs.



Enable ADC start module A

When you select this option, ePWM starts the Analog-to-Digital Conversion (ADC) for module A. By default, the software clears (disables) this option.

Number of event for SOCA to be generated

When you select **Enable ADC start module A**, this field specifies the number of the event that triggers ADC Start of Conversion for Module A (SOCA): **First** event triggers ADC start of conversion with every event (the default). **Second** event triggers ADC start

of conversion with every second event. Third event triggers ADC start of conversion with every third event.

Module A counter match event condition

When you select **Enable ADC start module A**, this field specifies the counter match condition that triggers an ADC start of conversion (SOC) event. The choices are:

DCAEVT1 soc and DCBEVT1 soc

When the ePWM asserts a DCAEVT1 or DCBEVT1 digital compare event. Use this feature to synchronize this PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**.

CTR=Zero

When the ePWM counter reaches zero (the default).

CTR=PRD

When the ePWM counter reaches the period value.

CTR=Zero or CTR=PRD

When the time base counter equals zero (TBCTR = 0x0000) or when the time base counter equals the period (TBCTR = TBPRD).

CTRU=CMPA

When the ePWM counter reaches the compare A value on the way up.

CTRD=CMPA

When the ePWM counter reaches the compare A value on the way down.

CTRU=CMPB

When the ePWM counter reaches the compare B value on the way up.

CTRD=CMPB

When the ePWM counter reaches the compare B value on the way down.

Enable ADC start module B

When you select this option, ePWM starts the Analog-to-Digital Conversion (ADC) for module B. By default, the software clears (disables) this option.

Number of event for SOCB to be generated

When you select **Enable ADC start module B**, this field specifies the number of the event that triggers ADC start of conversion: First event triggers ADC start of conversion with every event (the default). Second event triggers ADC start of conversion with every second event. Third event triggers ADC start of conversion with every third event.

Module B counter match event condition

When you select **Enable ADC start module B**, this field specifies the counter match condition that triggers an ADC start of conversion event. The choices are the same as for **Module A counter match event condition**.

Enable ePWM interrupt

Select this option to generate interrupts based on different events defined by **Number of event for interrupt to be generated** and **Interrupt counter match event condition**. By default, the software clears (disables) this option.

Number of event for interrupt to be generated

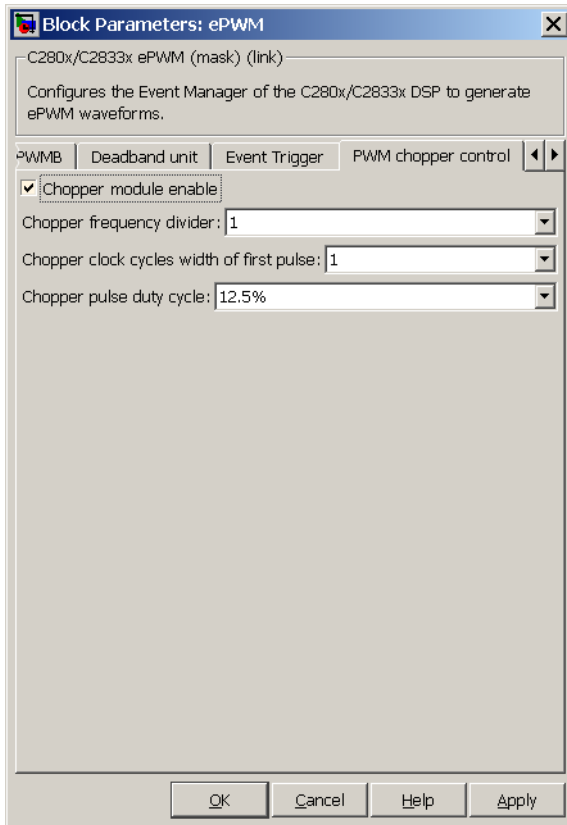
When you select **Enable ePWM interrupt**, this field specifies the number of the event that triggers the ePWM interrupt: First event triggers ePWM interrupt with every event (the default). Second event triggers ePWM interrupt with every second event. Third event triggers ePWM interrupt with every third event.

Interrupt counter match event condition

When you select **Enable ePWM interrupt**, this field specifies the counter match condition that triggers ePWM interrupt. The choices are the same as for **Module A counter match event condition**.

PWM Chopper Control Pane

The **PWM chopper control** pane lets you specify parameters for the PWM-Chopper (PC) submodule. The PC submodule uses 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

Set the prescaler value that determines 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

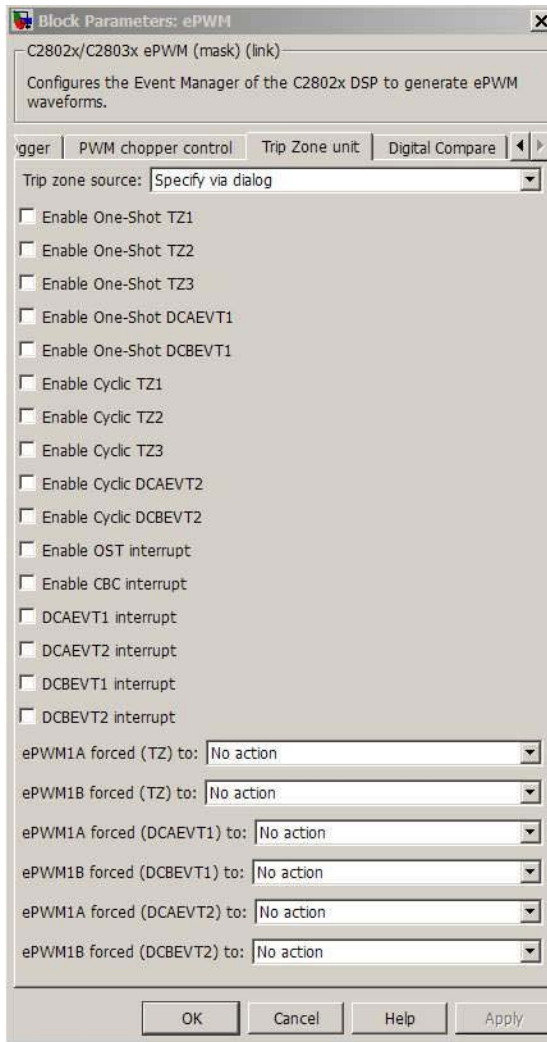
Choose an integer value from 1 to 16 to set the width of the first pulse. Use this feature to provide a high-energy first pulse to ensure hard and fast power switch turn on.

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 receives six TZ signals (TZ1 to TZ6) 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

Specify the source of the control logic. Choose **Specify via dialog** (the default) to enable specific Trip-zone signals via the

block dialog. Choose Input port to enable specific Trip-zone signals via a block input port, **TZSEL**.

If you select Input port, use the following bit operation to determine the value of the 16-bit integer to send to the **TZSEL**:

$$\text{TZSEL INPUT VALUE} = (\text{OSHT6} * 2^{13} + \text{OSHT5} * 2^{12} + \text{OSHT4} * 2^{11} + \text{OSHT3} * 2^{10} + \text{OSHT2} * 2^9 + \text{OSHT1} * 2^8 + \text{CBC6} * 2^5 + \text{CBC5} * 2^4 + \text{CBC4} * 2^3 + \text{CBC3} * 2^2 + \text{CBC2} * 2^1 + \text{CBC1} * 2^0)$$

For more information, see the "Trip-Zone Submodule Control and Status Registers" section of the *TMS320x28xx, 28xxx Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, Literature Number: SPRU791 on www.ti.com

The trip zone source category includes trip-zone and digital compare (DC) signals:

- **Enable One-Shot TZ#**
- **Enable One-Shot DCAEVT1 or DCBEVT1**
- **Enable Cyclic TZ#**
- **Enable Cyclic DCAEVT2 or DCBEVT2**

Enable One-Shot TZ#

Enable a Trip-zone signal in One-Shot Mode. When the trip event happens, the software immediately performs the respective action on the EPWMxA/B output. The condition remains latched until you clear it under software control.

Enable One-Shot DCAEVT1 or DCBEVT1

Select any of these check boxes to enable the corresponding digital compare event in One-Shot Mode. In this mode, when the digital compare event occurs (DCAEVT1 or DCBEVT1), the software immediately performs the respective action on the EPWMxA/B output. The condition remains latched until you clear it under software control.

Enable Cyclic TZ#

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 software immediately performs the respective action on the EPWMxA/B output. In Cycle-by-Cycle Mode, the condition clears automatically when the PWM Counter reaches zero. Therefore, in Cycle-by-Cycle Mode, the trip event clears or resets with every PWM cycle.

Enable Cyclic DCAEVT2 or DCBEVT2

Select any of these check boxes to enable the corresponding digital compare event in Cycle-by-Cycle Mode. In this mode, when the digital compare event occurs (DCAEVT2 or DCBEVT2) , the software immediately performs the respective action on the EPWMxA/B output. In Cycle-by-Cycle Mode, the condition clears automatically when the PWM Counter reaches zero. Therefore, in Cycle-by-Cycle Mode, the trip event clears or resets with every PWM cycle.

Enable OST Interrupt

Generate an interrupt when the one shot (OST) triggering event occurs.

Enable CBC Interrupt

Generate an interrupt when the cyclic or cycle-by-cycle (CBC) triggering event occurs.

DCAEVT1, DCBEVT1, DCAEVT2, or DCBEVT2 Interrupt

Generate an interrupt when the specified triggering digital compare event occurs.

ePWMxA forced to

ePWMxB forced to

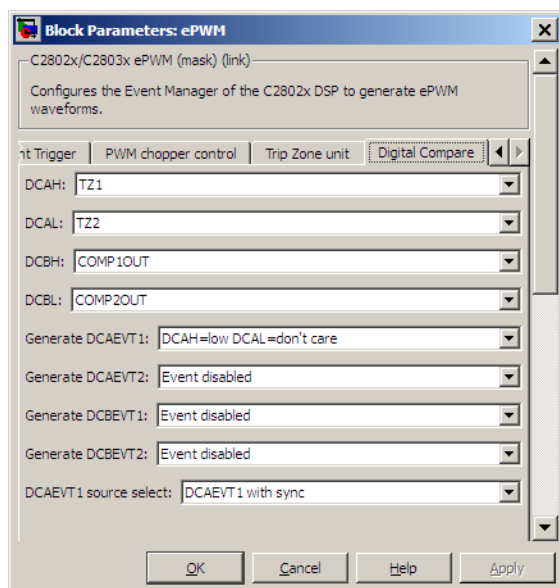
Upon a fault condition, override the ePWMxA and/or ePWMxB output and force it to one of the following: High, Low, or Hi-Z (High Impedance). Select No action, the default value, to disable this feature.

Digital Compare

Use the **Digital Compare** pane to configure the Digital Compare (DC) submodule.

Each digital compare (DC) submodule receives three TZ signals (TZ1 to TZ3) from the GPIO MUX, and three COMP signals from the COMP. These signals indicate fault or trip conditions that are external to the PWM submodule. Use the settings in this pane to output specific DC events in response to those external signals. These DC events feed directly into the Time-base, Trip-zone, and Event-trigger submodules.

For more information, see the “Digital Compare (DC) Submodule” section of the *TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, Literature Number: SPRUGE9.



DCAH, DCBH

If the TZ or COMP event you select occurs, assert a high signal. Qualify this signal using the **Generate DCAEVT#**, **Generate DCBEVT#** options.

DCAL, DCBL

If the TZ or COMP event you select occurs, assert a low signal. Qualify this signal using the **Generate DCAEVT#**, **Generate DCBEVT#** options.

Generate DCAEVT#, Generate DCBEVT#

Qualify the signals that generate DC events, such as DCAEVT# or DCBEVT#. Select the states of **DCAH**, **DCBH**, **DCAL**, and **DCBL** that generate the event. To disable this feature, choose the Event disabled option.

DCAEVT# source select, DCBEVT# source select

This parameter controls two separate aspects of triggering DC events:

- Triggering filtered or unfiltered DC event. (Configures DCACTL[EVT1SRCSEL] or DCACTL[EVT2SRCSEL].)
- Trigger the DC event synchronously or asynchronously. (Configures DCACTL[EVT1FRCSYNCSEL] or DCACTL[EVT2FRCSYNCSEL].)

Filtering

- Options that begin with DCAEVT# or DCBEVT# do not apply filtering to DC events. Qualified signals trigger DC events.
- Options that begin with DCEVTFILT apply filtering to DC events. Qualified signals pass through filtering circuits before triggering DC events. This filtering is not configurable in the ePWM block. For more information, refer to the “Event Filtering” section of the the *TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, Literature Number: SPRUGE9.

Synchronizing

- Options that end with `async` trigger DC events asynchronously. When the qualified or filtered signals exist, the DC submodule triggers the DC event immediately.
- Options that end with `sync` trigger DC events synchronously. Once the qualified or filtered signals exist, the DC submodule triggers the DC event in sync with the TBCLK signal.

References

For more information, consult the following references, available at the Texas Instruments Web site:

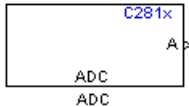
- *TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, literature number SPRUGE9
- *TMS320x2802x, 2803x Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide*, literature number SPRUGE8
- *Using the ePWM Module for 0% - 100% Duty Cycle Control Application Report*, literature number SPRU791
- *Configuring Source of Multiple ePWM Trip-Zone Events*, literature number SPRAAR4
- *TMS320F2809, TMS320F2808, TMS320F2806 TMS320F2802, TMS320F2801 TMS320C2802, TMS320C2801, and TMS320F2801x DSPs Data Manual*, literature number SPRS230

Purpose

Analog-to-digital converter (ADC)

Library

“C281x Chip Support (c281xlib)” on page 6-6

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.

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.

Dialog Box

ADC Control Pane

Source Block Parameters: ADC

C281x ADC (mask) (link)

Configures the ADC to output a constant stream of data collected from the ADC pins on the c281x DSP.

ADC Control | Input Channels

Module: A

Conversion mode: Sequential

Start of conversion: Software

Sample time: 0.001

Data type: uint16

Post interrupt at the end of conversion

OK Cancel Help

Module

Specify 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

Specify the type of signal that triggers the conversion:

- **Software** — Signal from software
- **EVA** — Signal from Event Manager A (only for Module A)
- **EVB** — Signal from Event Manager B (only for Module 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 1-10 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 1-11 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.

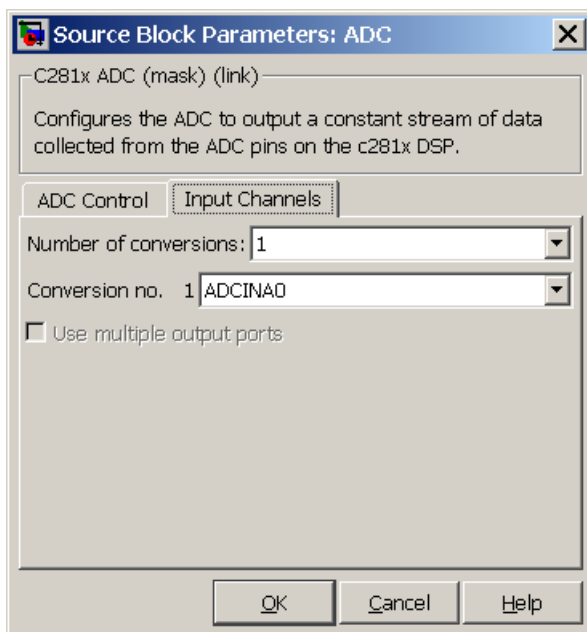
Data type

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. 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

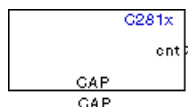
Purpose

Receive and log capture input pin transitions

Library

“C281x Chip Support (c281xlib)” on page 6-6

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. You can set the capture unit pins 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.

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 module stores the value of the selected timer in the two-level deep FIFO stack.

The C281x CAP module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers between C281x Peripherals” on page 1-16.

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

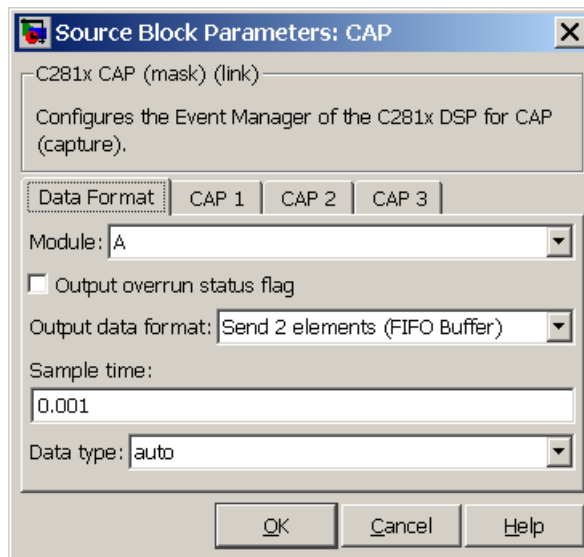
Outputs

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 captures 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:
 - 1** The CAP x FIFO status bits are read and the value is stored in the status flag.
 - 2** The top value of the FIFO is read and stored in the output at index 0.
 - 3** 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:
 - 4** The CAP x FIFO status bits are read and the value is stored in the status flag.
 - 5** 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:

- 6** The CAP x FIFO status bits are read and the value is stored in the status flag.
- 7** 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

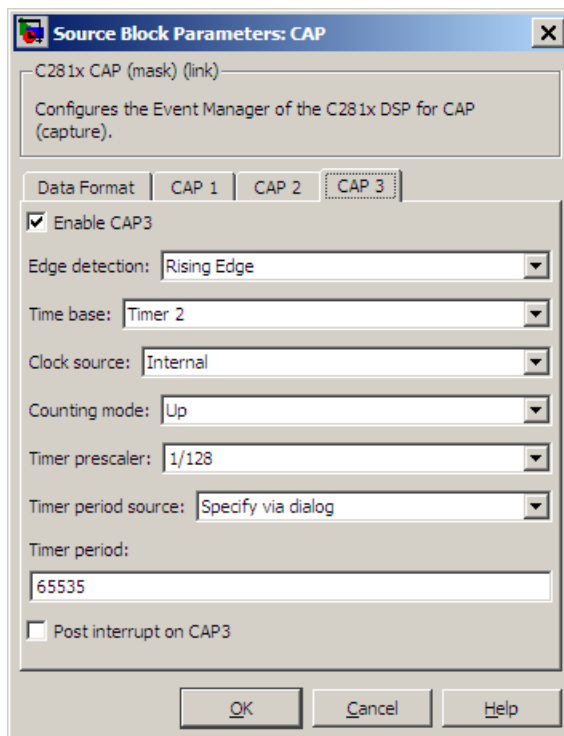
Time between outputs from the FIFO. If new data is not available, the previous data is sent.

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 Panes



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

Select which target board GP timer the CAP uses as a time base. CAPs 1, 2, and 3 can use Timer 1 or Timer 2. CAPs 4, 5, and 6 can use Timer 3 or Timer 4.

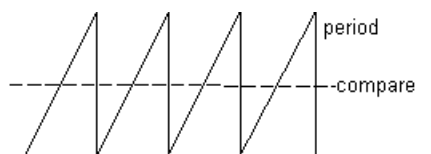
Clock source

This option is available only for the CAP 3 pane. You can select Internal to use the internal time base. Also configure the **Counting mode**, **Timer prescaler**, and **Timer period source** for the internal time base.

Select QEP circuit to generate the input clock from the quadrature encoder pulse (QEP) submodule.

Counting mode

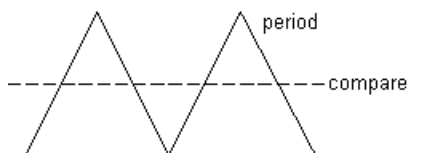
Select Up to generate an asymmetrical waveform output, or Up-down to generate a symmetrical waveform output, as shown in the following illustration.



Mode: Up/Asymmetric



Resulting waveform



Mode: Up-down/Symmetric



Resulting waveform

When you specify the **Counting mode** as Up (asymmetric) the waveform:

- Starts low
- Goes high when the rising period counter value matches the **Compare value**
- Goes low at the end of the period

When you specify the **Counting mode** as Up-down (symmetric) the waveform:

- Starts low
- Goes high when the increasing period counter value matches the **Compare value**

- Goes low when the decreasing period counter value matches the **Compare value**

Counting mode becomes unavailable when you set **Clock source** to QEP circuit.

Timer Prescaler

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 following table shows the rates that result from selecting each option.

Scaling	Resulting Rate (μ 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 These rates assume a 75-MHz input clock.

Timer period source

Select **Specify via dialog** to enable the **Timer period** parameter. Select **Input port** to create a block input, **T1**, that accepts the timer period value.

Timer period

Set the length of the timer period in clock cycles. Enter a value from 0 to 65535. The value defaults to 65535.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$\text{Sysclk}(150\text{MHz}) \rightarrow \text{HISPCLK}(1/2) \rightarrow \text{InputClockPr escaler}(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is $1/0.586\text{MHz}$, which is 1.706 μs .

Post interrupt on CAP

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

See Also

C281x Hardware Interrupt

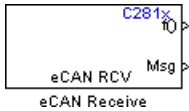
Purpose

Enhanced Control Area Network receive mailbox

Library

“C281x Chip Support (c281xlib)” on page 6-6

Description



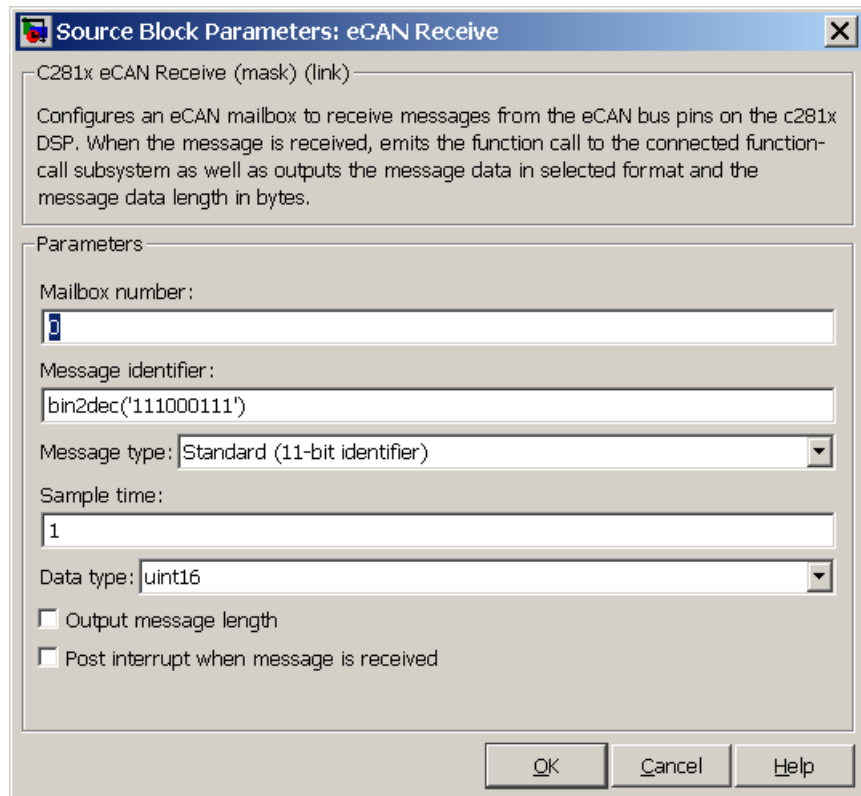
The C281x enhanced Control Area Network (eCAN) Receive block generates source code for receiving eCAN messages through an eCAN mailbox. The eCAN module on the DSP chip provides serial communication capability and has 32 mailboxes configurable for receive or transmit. The C281x supports eCAN data frames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

C281x eCAN Receive

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 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 1-11 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 = 2 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];
```

C281x 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

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

References

For detailed information on the eCAN module, see *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

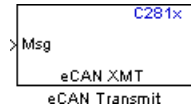
Purpose

Enhanced Control Area Network transmit mailbox

Library

“C281x Chip Support (c281xlib)” on page 6-6

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 = 2 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
```

C281x 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
```

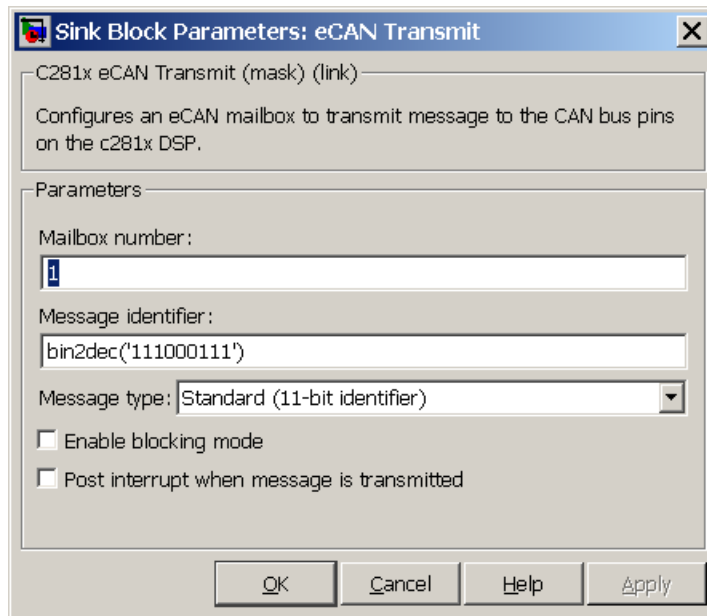
For input of type uint16[2], which is a two-element vector,

```
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



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).

C281x eCAN Transmit

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

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

See Also

C281x eCAN Receive

Purpose

General-purpose I/O pins for digital input

Library

“C281x Chip Support (c281xlib)” on page 6-6

Description

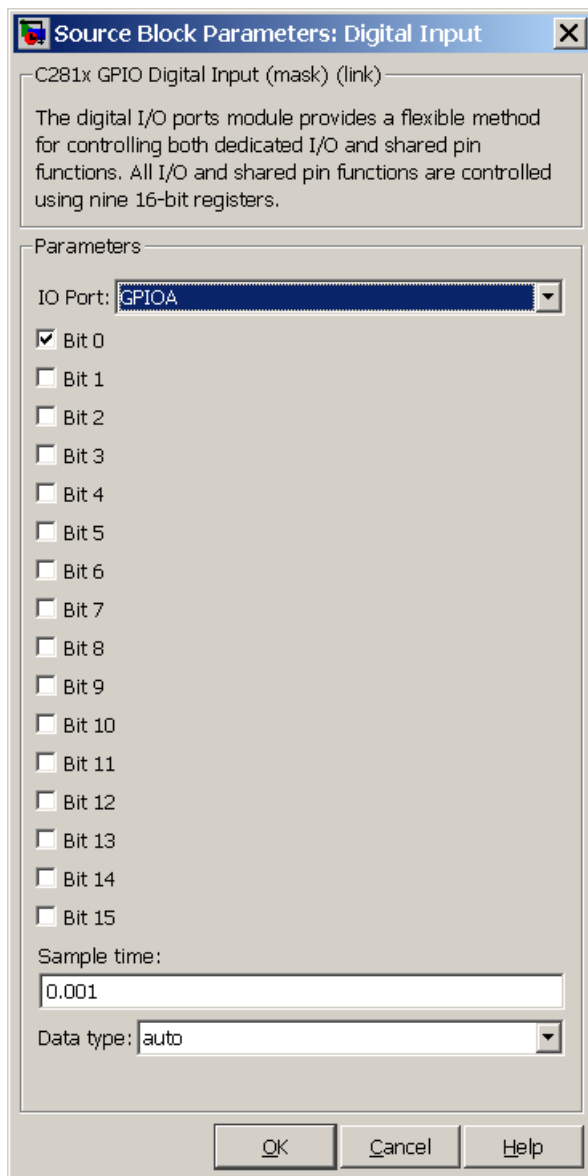


This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital input. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.

Note To avoid losing any new settings, click **Apply** before changing the **IO Port** parameter.

C281x GPIO Digital Input

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. (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
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

C281x GPIO Digital Input

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

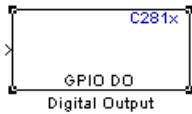
C281x GPIO Digital Output

Purpose

General-purpose I/O pins for digital output

Library

“C281x Chip Support (c281xlib)” on page 6-6

Description

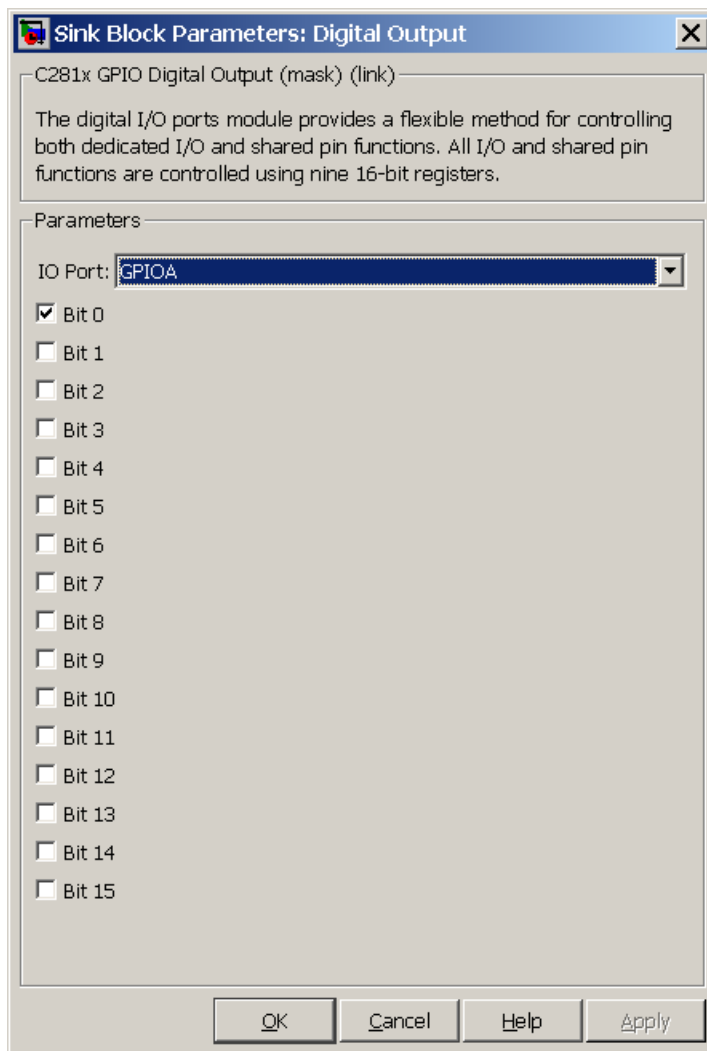
This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital output. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.

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

Note To avoid losing any new settings, click **Apply** before changing the **IO Port** parameter.

C281x GPIO Digital Output

Dialog Box



IO Port

Select the input/output port to use: GPIOA, GPIOB, GPIOPD, GPIOPE, GPIOPF, or GPIOPG and select the I/O Port bits to enable

for digital input. (There is no GPIOPC port on the C281x.) If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Multiple GPIO DO 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
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

C281x GPIO Digital Output

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

See Also

C281x GPIO Digital Input

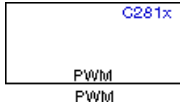
Purpose

Pulse width modulators (PWMs)

Library

“C281x Chip Support (c281xlib)” on page 6-6

Description



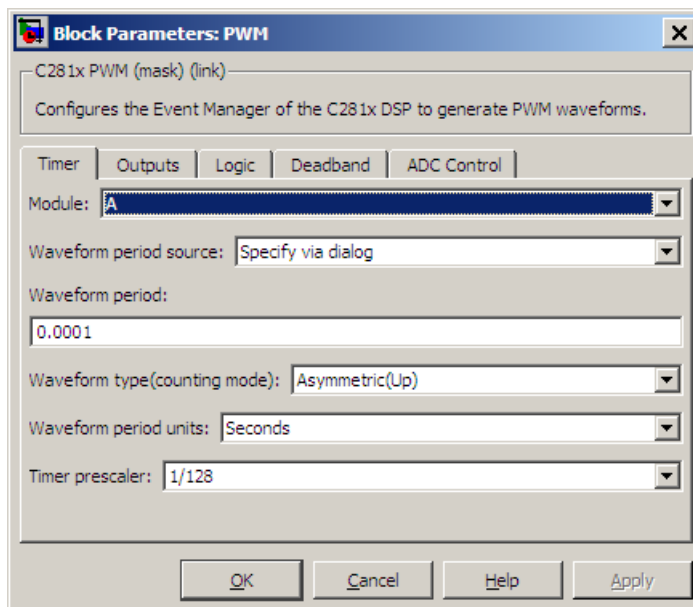
F2812 DSPs include a suite of pulse width modulators (PWMs) used to generate various signals. This block provides options to set the A or B module Event Managers to generate the waveforms you require. The twelve PWMs are configured in six pairs, with three pairs in each module.

The C281x PWM module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers between C281x Peripherals” on page 1-16.

Note All inputs to the C281x PWM block must be scalar values.

Dialog Box

Timer Pane



Module

Specify 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.

Note All inputs to the C281x PWM block must be scalar values.

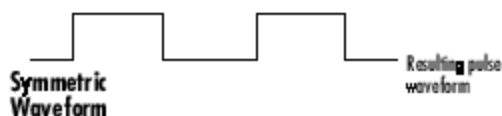
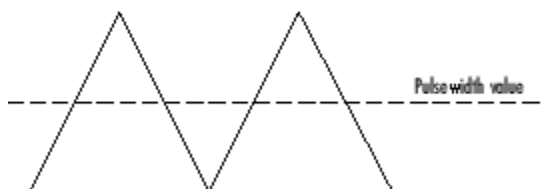
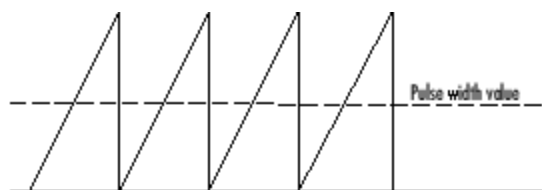
Waveform period

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 (counting mode)

Type of waveform to be generated by the PWM pair. The F2812 PWMs can generate two types of waveforms: **Asymmetric(Up)** and **Symmetric(Up-down)**. The following illustration shows the difference between the two types of waveforms.



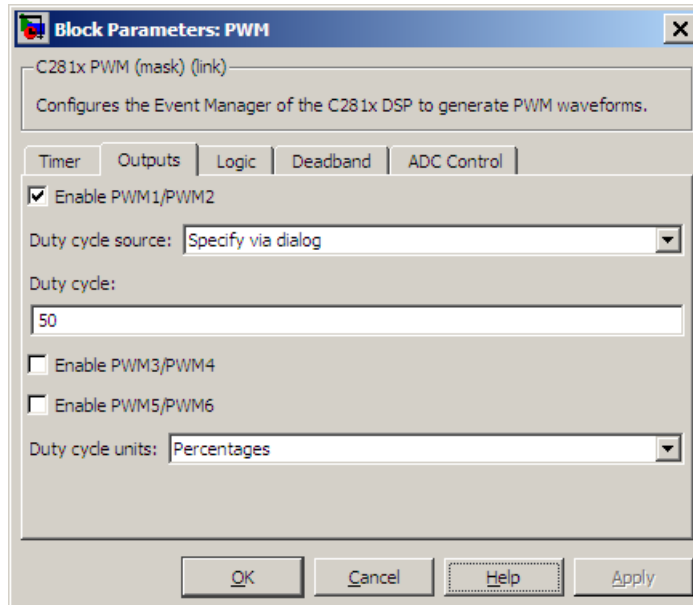
Waveform period units

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**. Changing these units changes the **Waveform period** value and the **Duty cycle** value and **Duty cycle units** selection.

Timer prescaler

Divide the clock input to produce the desired timer counting rate.

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

Select **Specify via dialog** to use the dialog box to enter a **Duty cycle** value for the pair of PWM outputs. Select **Input port** to use the input port, **W#**, to enter a **Duty cycle** value for the pair of PWM outputs.

The input port **W1** corresponds to PWM1/PWM2. **W2** corresponds to PWM3/PWM4. **W3** corresponds to PWM5/6.

C281x PWM

Note All inputs to the C281x PWM block must be scalar values.

Duty cycle

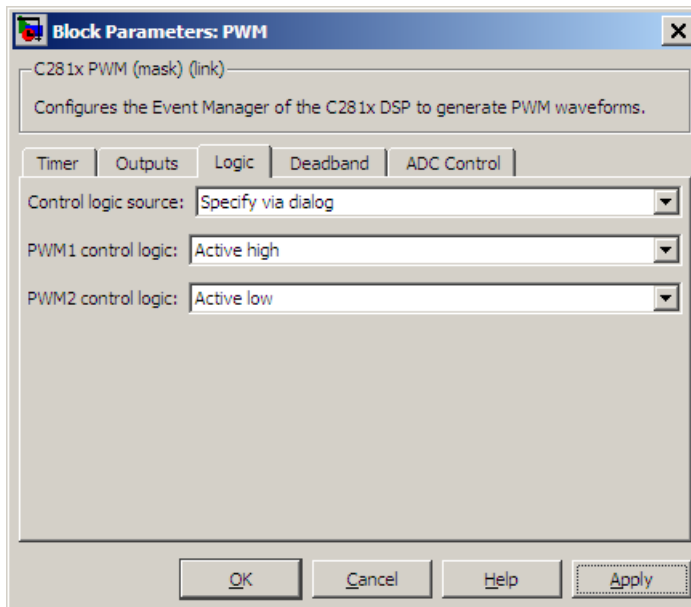
Set the ratio of the PWM waveform pulse duration to the PWM **Waveform period**.

Duty cycle units

Units for the duty cycle. Valid choices are **Clock cycles** and **Percentages**. Changing these units changes the **Duty cycle** value, and the **Waveform period** value and **Waveform period units** selection.

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

Logic Pane



Control logic source

Configure the control logic for all PWMs enabled on the Outputs tab. Valid settings are `Specify via dialog` (default setting) or `Input port`.

`Specify via Dialog` enables **PWM control logic** settings for each PWM output:

- `Forced high` causes the pulse value to be high.
`Active high` causes the pulse value to go from low to high.
`Active low` causes the pulse value to go from high to low.
`Forced low` causes the pulse value to be low.

Input port adds an input port to the PWM block for setting the C2000 ACTRx register. Each PWM uses 2 bits to set the following options:

- 00 Forced Low
- 01 Active Low
- 10 Active High
- 11 Forced High

Bits 11–0 of the 16-bit Compare Action Control Registers for module A control PWM1-6

Bits 11–0 of the 16-bit Compare Action Control Registers for module B control PWM1-6

For example: If a decimal value of 3222 is read at the input port while using PWM module A, the following PWM settings will be honored:

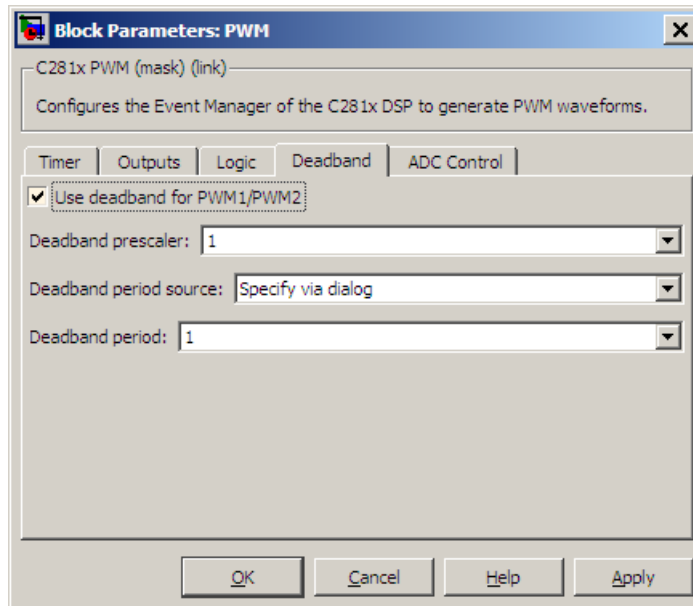
$$3222 = 0C96h = 110010010110b$$

So that:

- PW1: Active High
- PW2: Active Low
- PW3: Active Low
- PW4: Active High
- PW5: Forced Low
- PW6: Forced High

For more information, see the section on Compare Action Control Registers (ACTRA and ACTRB) in the Texas Instruments™ document “TMS320x281x DSP Event Manager (EV) Reference Guide”, literature number SPRU065.

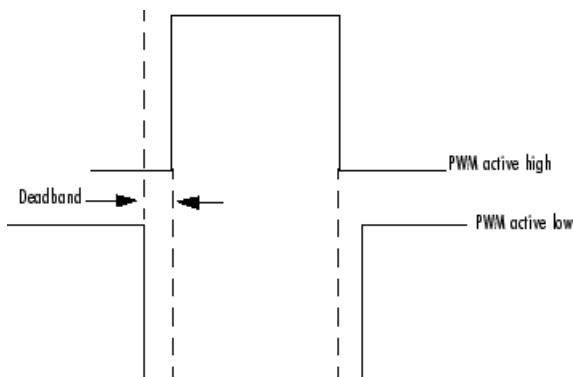
Deadband Pane



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.

C281x PWM



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

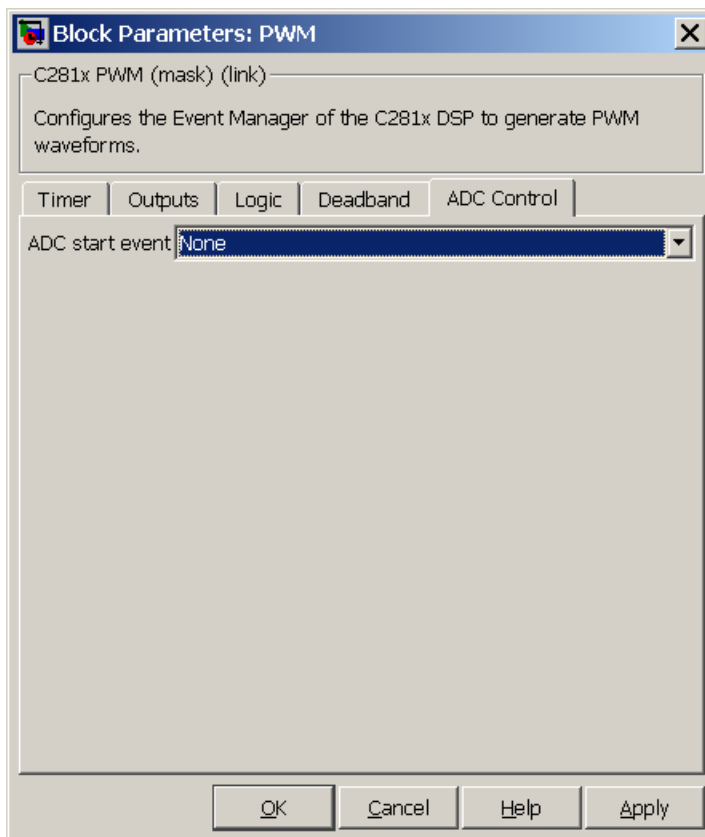
Source from which the deadband period is obtained. Select **Specify via dialog** to enter the values in the **Deadband period** field or select **Input port** to use a value, in clock cycles, from the input port.

Note All inputs to the C281x PWM block must be scalar values.

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 event 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

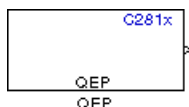
C281x ADC

Purpose

Quadrature encoder pulse circuit

Library

“C281x Chip Support (c281xlib)” on page 6-6

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). The circuit counts both edges of the QEP pulses, so the frequency of the QEP clock is four times the input sequence frequency.

The QEP, in combination with an optical encoder, is 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.

The C281x QEP module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers between C281x Peripherals” on page 1-16.

Dialog Box

Source Block Parameters: QEP

C281x QEP (mask) (link)

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

Parameters

Module: A

Counting mode: RPM

Positive rotation: Clockwise

Initial count : 0

Encoder resolution (pulse/revolution): 1024

Enable QEP index

Enable index qualification mode

Timer period: 65535

Sample time: 0.001

Data type: auto

OK Cancel Help

Module

Specify which QEP pins to use:

- A — Uses QEP1 and QEP2 pins.

- **B** — Uses QEP3 and QEP4 pins.

Counting mode

Specify how to count the QEP pulses:

- **Counter** — Count the pulses based on GP Timer 2 (or GP Timer 4 for EVB).
- **RPM** — Count the rotations 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.

Initial count

Initial value for the counter. The value defaults to 0.

Encoder resolution (pulse/revolution)

Number of QEP pulses per revolution. This field appears only if you select RPM.

Enable QEP index

Reset the QEP counter to zero when the QEP index input on CAP3_QEPI1 transitions from low to high.

Enable index qualification mode

Qualify the QEP index input on CAP3_QEPI1. Ensure that the levels on CAP1_QEP1 and CAP2_QEP2 are high before asserting the index signal as valid.

Timer period

Set the length of the timer period in clock cycles. Enter a value from 0 to 65535. The value defaults to 65535.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$\text{Sysclk}(150\text{MHz}) \rightarrow \text{HISPCLK}(1/2) \rightarrow \text{InputClockPr escaler}(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is $1/0.586\text{MHz}$, which is $1.706\ \mu\text{s}$.

Sample time

Time interval, in seconds, between consecutive reads from the QEP pins.

Data type

Data type of the QEP pin data. The circuit reads the data as 16-bit data and then casts it to the selected data type. Valid data types are auto, double, single, int8, uint8, int16, uint16, int32, uint32 or boolean.

References

For more information on the QEP module, consult the following documents, available at the Texas Instruments Web site:

- *TMS320x280x, 2801x, 2804x Enhanced Quadrature Encoder Pulse (eQEP) Module Reference Guide*, Literature Number SPRU790
- *Using the Enhanced Quadrature Encoder Pulse (eQEP) Module in TMS320x280x, 28xxx as a Dedicated Capture Application Report*, Literature Number SPRAAH1

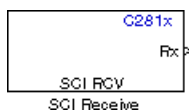
Purpose

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

Library

“C281x Chip Support (c281xlib)” on page 6-6

Description



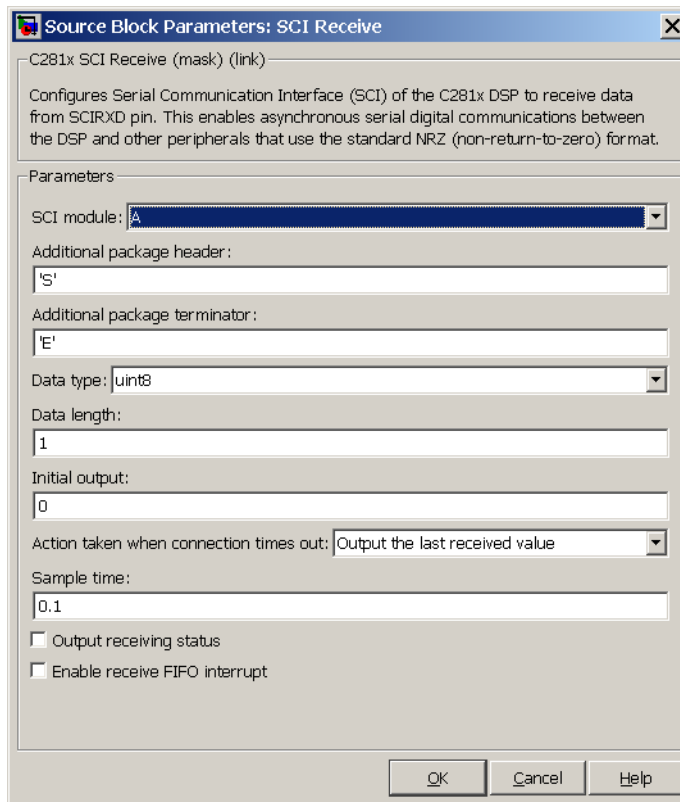
The C281x SCI Receive block supports asynchronous serial digital communications between the target and other asynchronous peripherals in nonreturn-to-zero (NRZ) format. This block configures the C281x DSP target to receive scalar or vector data from the COM port via the C28x target’s COM port.

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



Source Block Parameters: SCI Receive

C281x SCI Receive (mask) (link)

Configures Serial Communication Interface (SCI) of the C281x DSP to receive data from SCIRXD pin. This enables asynchronous serial digital communications between the DSP and other peripherals that use the standard NRZ (non-return-to-zero) format.

Parameters

SCI module: A

Additional package header: 'S'

Additional package terminator: 'E'

Data type: uint8

Data length: 1

Initial output: 0

Action taken when connection times out: Output the last received value

Sample time: 0.1

Output receiving status

Enable receive FIFO interrupt

OK Cancel Help

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 from the input (the data length originally input to the host-side SCI Transmit block).

Initial output

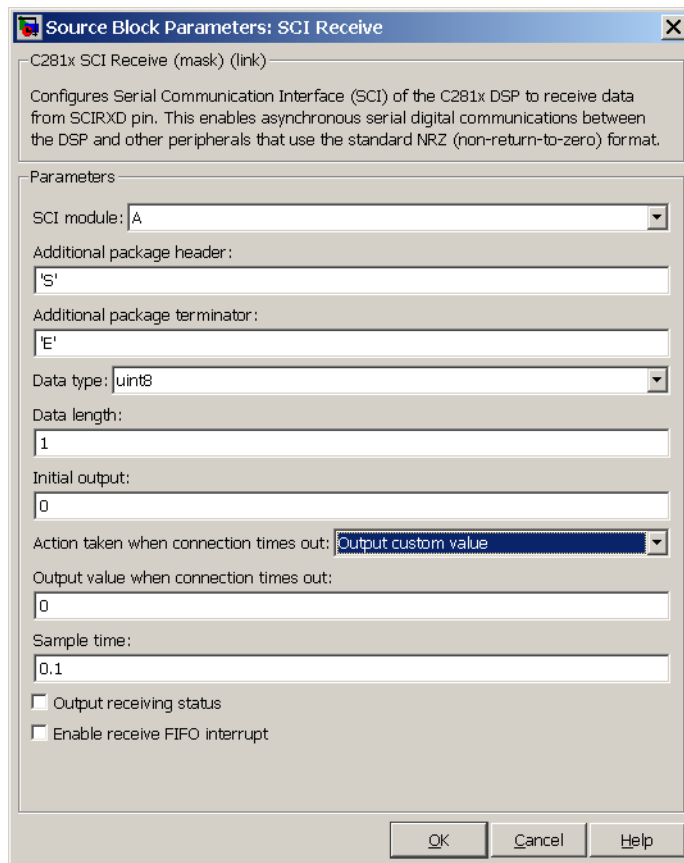
Default value from the C281x SCI Receive block. This value is used, for example, if a connection time-out occurs and the **Action taken when connection timeout** field is set to “Output the last received value”, but nothing yet has been received.

C281x SCI Receive

Action taken when connection timeout

Specify 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 you select "Output custom value", use the "Output value when connection times out" field to set the custom value.



The image shows a dialog box titled "Source Block Parameters: SCI Receive". It contains the following fields and options:

- Parameters section:
 - SCI module: A (dropdown)
 - Additional package header: 'S' (text field)
 - Additional package terminator: 'E' (text field)
 - Data type: uint8 (dropdown)
 - Data length: 1 (text field)
 - Initial output: 0 (text field)
 - Action taken when connection times out: Output custom value (dropdown)
 - Output value when connection times out: 0 (text field)
 - Sample time: 0.1 (text field)
- Checkboxes:
 - Output receiving status
 - Enable receive FIFO interrupt

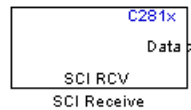
Buttons: OK, Cancel, Help

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 1-11 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).

C281x SCI Receive

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

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

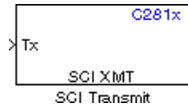
See Also

C281x SCI Transmit, C281x Hardware Interrupt

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

Library “C281x Chip Support (c281xlib)” on page 6-6

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`.

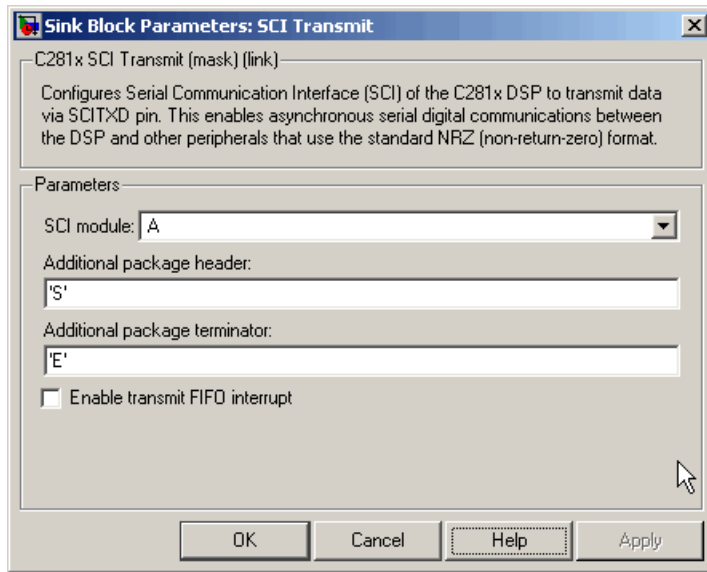
Note For any given model, you can have only one C281x SCI Transmit block per module. There are two modules, A and B, which can be configured through the F2812 eZdsp Target Preferences block.

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.

C281x SCI Transmit

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.

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

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

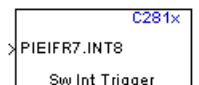
See Also

C281x SCI Receive, C281x Hardware Interrupt

C281x Software Interrupt Trigger

Purpose Generate software triggered nonmaskable interrupt

Library “C281x Chip Support (c281xlib)” on page 6-6



Description Software Interrupt Trigger

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.

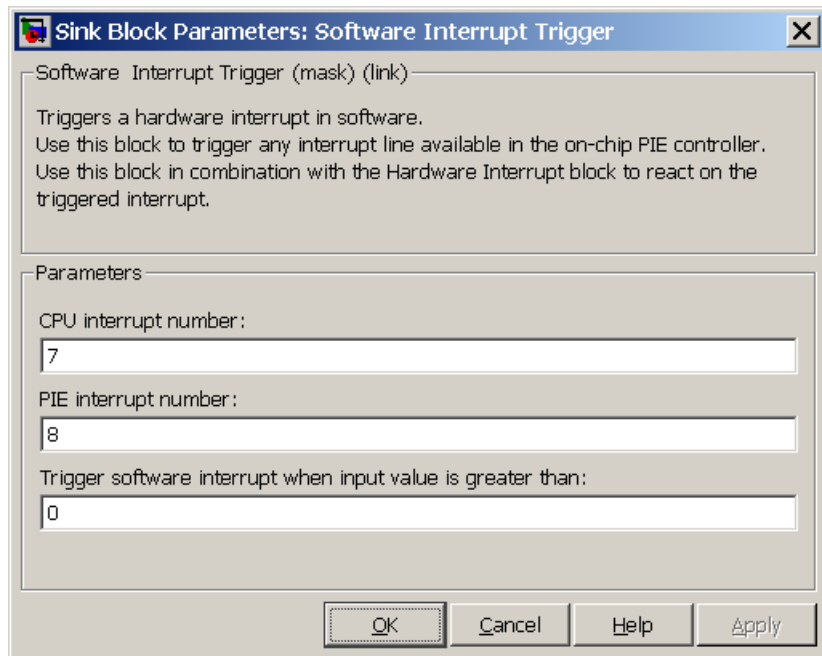
To use this block, add a Hardware Interrupt block to your model to process the software triggered interrupt from this block into an interrupt service routine on the processor. Set the interrupt number in the Hardware Interrupt block to the value you set here in **CPU interrupt number**.

The CPU and PIE interrupt numbers together specify a single interrupt for a single peripheral or peripheral module. The “C281x Peripheral Interrupt Vector Values” table maps CPU and PIE interrupt numbers to these peripheral interrupts.

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

C281x Software Interrupt Trigger

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.

References

For detailed information about interrupt processing, see *TMS320x281x DSP System Control and Interrupts Reference Guide*, SPRU078C, available at the Texas Instruments Web site.

C281x Software Interrupt Trigger

See Also

C281x Hardware Interrupt

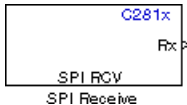
Purpose

Receive data via serial peripheral interface on target

Library

“C281x Chip Support (c281xlib)” on page 6-6

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.

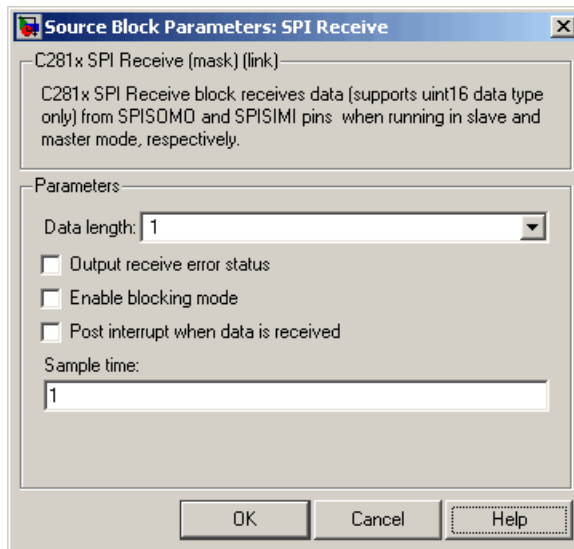
In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which 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

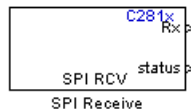
Specify how many uint16s are expected to be received. Select 1 through 16.

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 (Overflow: 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 1-11 for a discussion of block placement and other necessary settings.

See Also

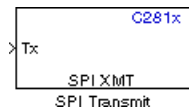
C281x SPI Transmit, C281x Hardware Interrupt

C281x SPI Transmit

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

Library “C281x Chip Support (c281xlib)” on page 6-6

Description



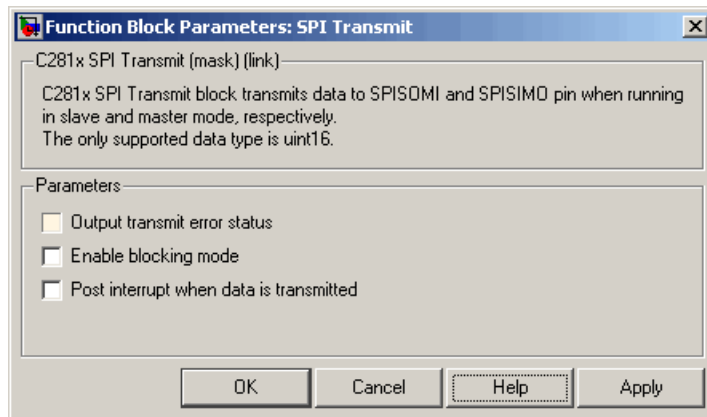
The C281x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. 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.

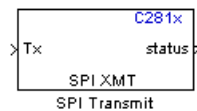
Many SPI-specific settings are in the **DSPBoard** section of the F2812 eZdsp Target Preferences block. You should verify that these settings are correct for your application.

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)

Enable blocking mode

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

C281x SPI Transmit

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 general-purpose timer in Event Manager module

Library

“C281x Chip Support (c281xlib)” on page 6-6

Description



The C281x contains two event-manager (EV) modules. Each module contains two general-purpose (GP) timers. You can use these timers as independent time bases for various applications.

Use the C281x Timer block to set the periodicity of one GP timer and the conditions under which it posts interrupts. Each model can contain up to four C281x Timer blocks.

The C281x Timer module configures GP Timers that other C281 blocks share. For more information and guidance on sharing timers, see “Sharing General Purpose Timers between C281x Peripherals” on page 1-16.

C281x Timer

Dialog Box

C281x EV Timer (mask) (link)

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

Parameters

Module: A

Timer no.: Timer 1

Timer period source: Specify via dialog

Timer period: 10000

Compare value source: Specify via dialog

Compare value: 5000

Counting mode: Up

Timer prescaler: 1/128

Post interrupt on period match

Post interrupt on underflow

Post interrupt on overflow

Post interrupt on compare match

OK Cancel Help Apply

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.**

Clock source

When **Timer no** has a value of **Timer 2** or **Timer 4**, use this parameter to select the clock source for the event timer. You

can choose either Internal or QEP circuit. When you select Internal, you can configure other options such as **Timer period source**, **Counting mode**, and **Timer prescaler**.

Timer period source

Select the source of the event timer period. Use **Specify via dialog** to set the period using **Timer period**. Select **Input port** to create an input, **T**, that accepts the value of the timer period in clock cycles, from 0 to 65535. **Timer period source** becomes unavailable when **Clock source** is set to QEP circuit.

Timer period

Set the length of the timer period in clock cycles. Enter a value from 0 to 65535. The value defaults to 10000.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$\text{Sysclk}(150\text{MHz}) \rightarrow \text{HISPCLK}(1/2) \rightarrow \text{InputClockPr escaler}(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is $1/0.586\text{MHz}$, which is 1.706 μs .

Compare value source

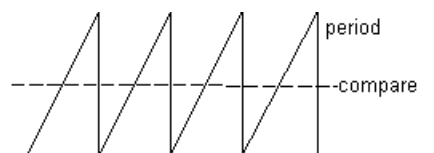
Select the source of the compare value. Use **Specify via dialog** to set the period using the **Compare value** parameter. Select **Input port** to create a block input, **W**, that accepts the value of the compare value, from 0 to 65535.

Compare value

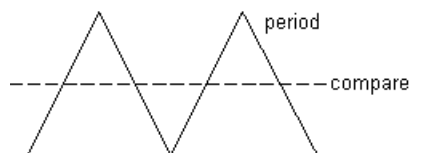
Enter a constant value for comparison to the running timer value for generating interrupts. Enter a value from 0 to 65535. The value defaults to 5000. The timer only generates interrupts if you enable **Post interrupt on compare match**.

Counting mode

Select Up to generate an asymmetrical waveform output, or Up-down to generate a symmetrical waveform output, as shown in the following illustration.



Mode: Up/Asymmetric



Mode: Up-down/Symmetric



When you specify the **Counting mode** as Up (asymmetric) the waveform:

- Starts low
- Goes high when the rising period counter value matches the **Compare value**
- Goes low at the end of the period

When you specify the **Counting mode** as Up-down (symmetric) the waveform:

- Starts low
- Goes high when the increasing period counter value matches the **Compare value**
- Goes low when the decreasing period counter value matches the **Compare value**

Counting mode becomes unavailable when **Clock source** is set to QEP circuit.

Timer prescaler

Divide the clock input to produce the desired timer counting rate.

Timer prescaler becomes unavailable when **Clock source** is set to QEP circuit.

Post interrupt on period match

Generate an interrupt when the value of the timer reaches its maximum value as specified in **Timer period**.

Post interrupt on underflow

Generate an interrupt when the value of the timer cycles back to 0.

Post interrupt on overflow

Generate an interrupt when the value of the timer reaches its maximum, 65535. Also set **Timer period** to 65535 for this parameter to work.

Post interrupt on compare match

Generate an interrupt when the value of the timer equals **Compare value**.

References

TMS320x281x DSP Event Manager (EV) Reference Guide, Literature Number: SPRU065, available from the Texas Instruments Web site.

See Also

C281x Hardware Interrupt, Idle Task

C28x Watchdog

Purpose

Configure counter reset source of DSP Watchdog module

Library

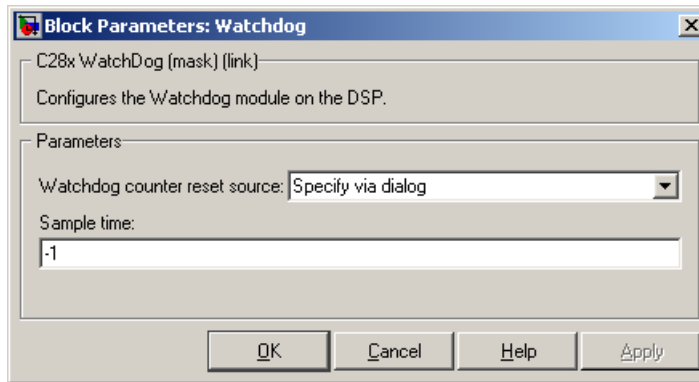
“C280x Chip Support (c280xlib)” on page 6-2, “C2802x Chip Support (c2802xlib)” on page 6-4, “C281x Chip Support (c281xlib)” on page 6-6, “C28x3x Chip Support (c2833xlib)” on page 6-8

Description

This block configures the counter reset source of the Watchdog module on the DSP.



Dialog Box



Watchdog counter reset source

- **Input** — Create an input port on the watchdog block. The input signal resets the counter.
- **Specify via dialog** — Use the value of **Sample time** to reset the watchdog timer.

Sample time

The interval at which the DSP resets the watchdog timer. When you set this value to -1, the model inherits the sample time value of the model. To execute this block asynchronously, set **Sample**

Time to -1, and refer to “Asynchronous Interrupt Processing” on page 1-11 for a discussion of block placement and other necessary settings.

Clarke Transformation

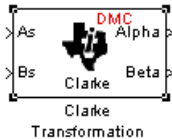
Purpose

Convert balanced three-phase quantities to balanced two-phase quadrature quantities

Library

“C28x DMC (c28xdmclib)” on page 6-10

Description

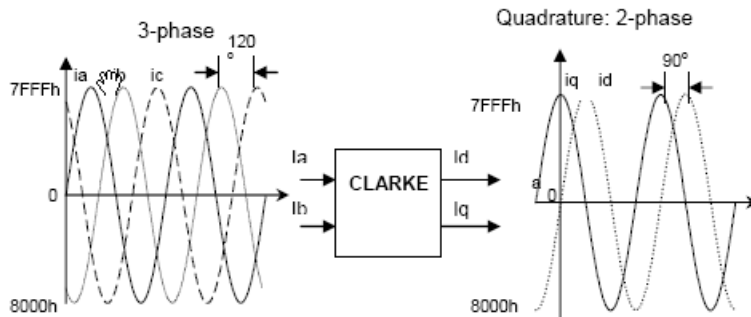


This block converts balanced three-phase quantities into balanced two-phase quadrature quantities. The transformation implements these equations

$$I_d = I_a$$

$$I_q = (2I_b + I_a) / \sqrt{3}$$

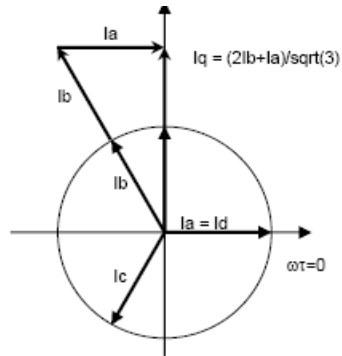
and is illustrated in the following figure.



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:

$$\begin{aligned}
 ia &= I * \sin(\omega t) \\
 ib &= I * \sin(\omega t + 2\pi/3) \\
 ic &= I * \sin(\omega t - 2\pi/3) \\
 id &= I * \sin(\omega t) \\
 iq &= I * \sin(\omega t + \pi/2)
 \end{aligned}$$



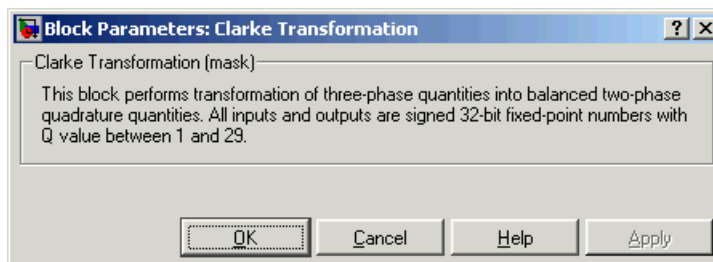
The variables used in the preceding equations and figures correspond to the variables on the block as shown in the following table:

	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 Chapter 4, “Using the IQmath Library” for more information.

Clarke Transformation

Dialog Box



References

For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

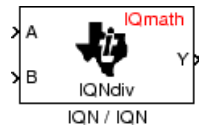
See Also

Inverse Park Transformation, Park Transformation, PID Controller, Space Vector Generator, Speed Measurement

Purpose Divide IQ numbers

Library “C28x IQmath (tiiqmathlib)” on page 6-11

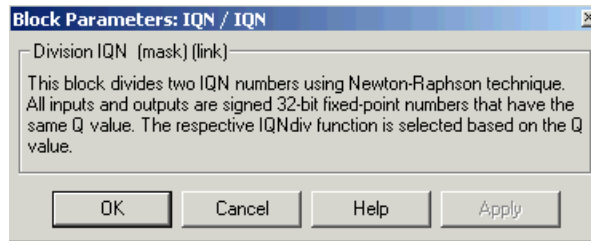
Description



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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

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

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

Float to IQN

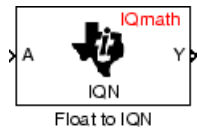
Purpose

Convert floating-point number to IQ number

Library

“C28x IQmath (tiiqmathlib)” on page 6-11

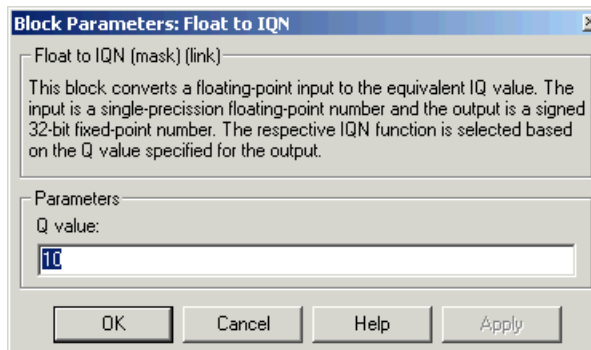
Description



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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



Q value

Q value from 1 to 30 that specifies the precision of the output

References

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

See Also

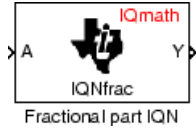
Absolute IQN, Arctangent IQN, Division IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Fractional part IQN

Purpose Fractional part of IQ number

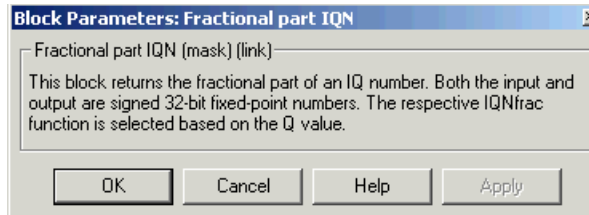
Library “C28x IQmath (tiiqmathlib)” on page 6-11

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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



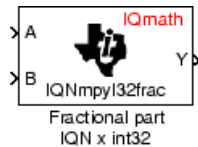
References For detailed information on the IQmath library, see the user’s guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user’s guide is included in the zip file download that also contains the IQmath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

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

Purpose Fractional part of result of multiplying IQ number and long integer

Library “C28x IQmath (tiiqmathlib)” on page 6-11

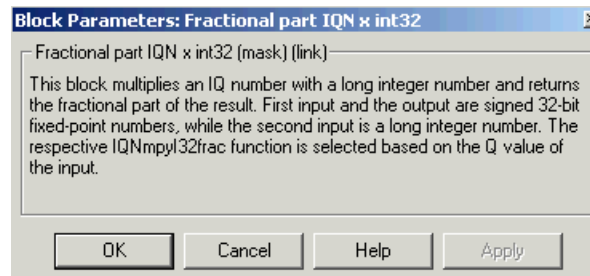
Description



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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

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

See Also

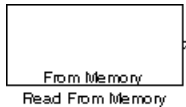
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

From Memory

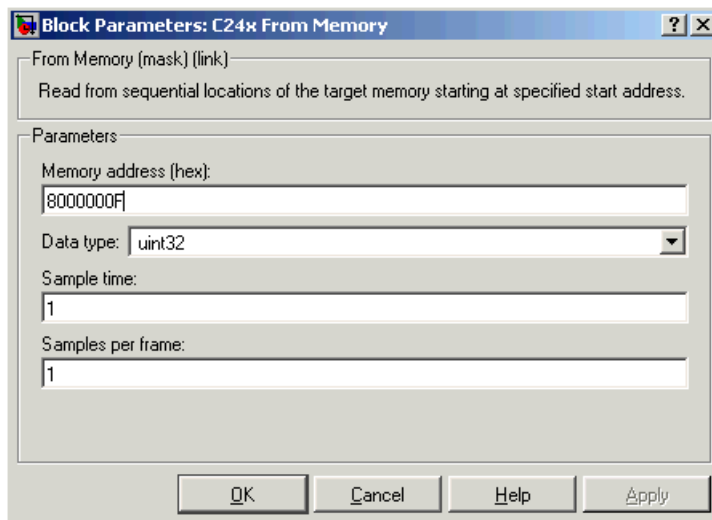
Purpose Retrieve data from target memory

Library “C280x Chip Support (c280xlib)” on page 6-2, “C281x Chip Support (c281xlib)” on page 6-6, and “C28x3x Chip Support (c2833xlib)” on page 6-8 in Target Support Package software

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.

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

From RTDX

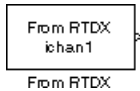
Purpose

Add RTDX communication channel for target to receive data from host

Library

“RTDX Instrumentation (rtdxBlocks)” on page 6-15

Description



Note This block will be removed from the Target Support Package product in an upcoming release. Consider using TCP/IP or UDP blocks instead.

When you generate code from Simulink in Real-Time Workshop software 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:

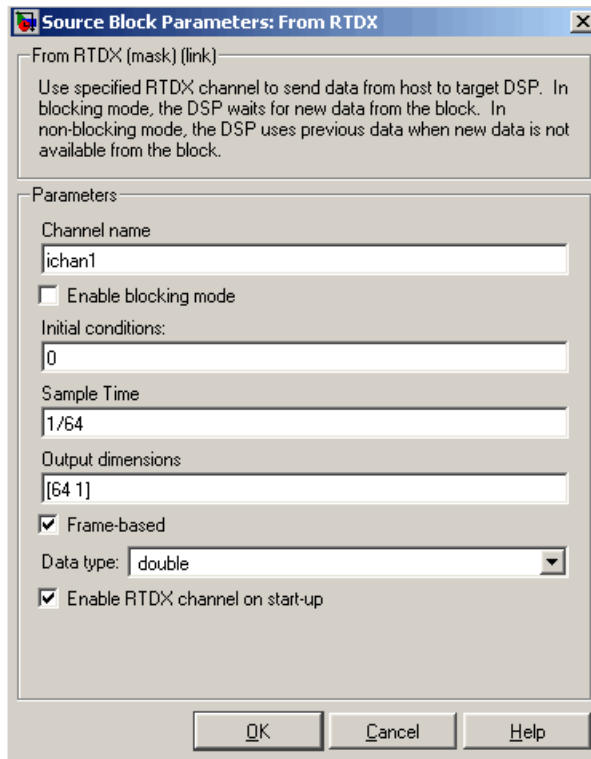
- 1 Add one or more To RTDX or From RTDX blocks to your model.
- 2 Download and run your model on your target.
- 3 Enable the RTDX channels from MATLAB or use **Enable RTDX channel on start-up** on the block dialog.

- 4 Use the `readmsg` and `writemsg` functions on the MATLAB command line to send and retrieve data from the target over RTDX.

To see more details about using RTDX in your model, refer to the Embedded IDE Link User's Guide and the following demos:

- Real-Time Data Exchange (RTDX™) Tutorial
- Sine Wave Generation via RTDX™
- DC Motor Speed Control via 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 software. 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 value defaults to 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. 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 the Embedded IDE Link software 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

`ticcs`, `readmsg`, `To RTDX`, `writemsg`.

References

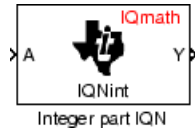
RTDX 2.0 User's Guide, Literature Number: SPRUF7C7, available from the Texas Instruments Web site.

How to Write an RTDX Host Application Using MATLAB, Literature Number: SPRA386, available from the Texas Instruments Web site.

Purpose Integer part of IQ number

Library “C28x IQmath (tiiqmathlib)” on page 6-11

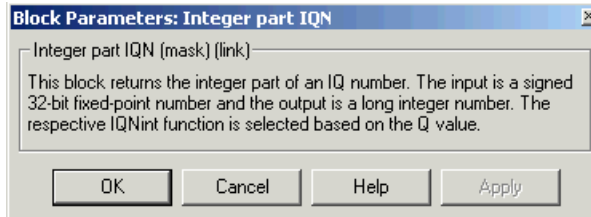
Description



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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

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

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

Integer part IQN x int32

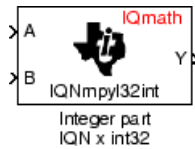
Purpose

Integer part of result of multiplying IQ number and long integer

Library

“C28x IQmath (tiiqmathlib)” on page 6-11

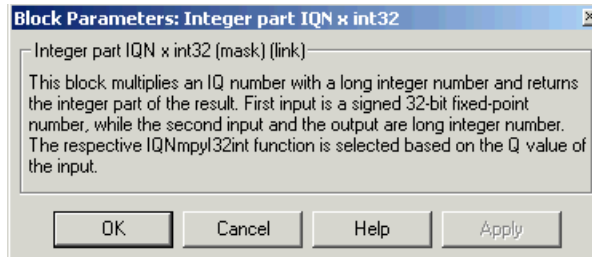
Description



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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

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

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

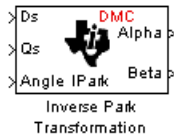
Purpose

Convert rotating reference frame vectors to two-phase stationary reference frame

Library

“C28x DMC (c28xdmclib)” on page 6-10

Description

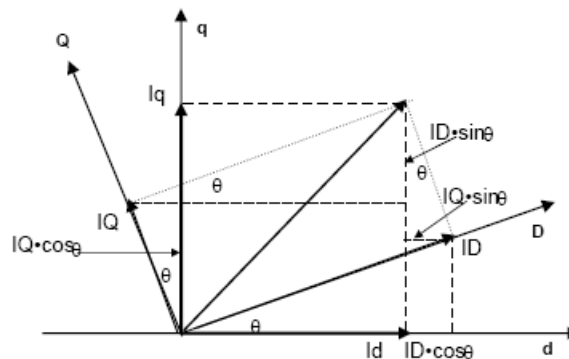


This block converts vectors in an orthogonal rotating reference frame to a two-phase orthogonal stationary reference frame. The transformation implements these equations:

$$I_d = I_D \cdot \cos \theta - I_Q \cdot \sin \theta$$

$$I_q = I_D \cdot \sin \theta + I_Q \cdot \cos \theta$$

and is illustrated in the following figure.



The inputs to this block are the direct axis (Ds) and quadrature axis (Qs) 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.

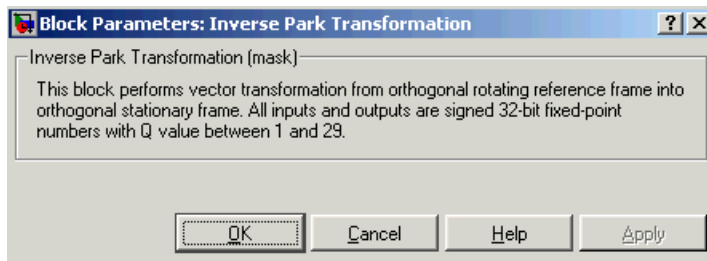
The variables used in the preceding figure and equations correspond to the block variables as shown in the following table:

Inverse Park Transformation

	Equation Variables	Block Variables
Inputs	ID	Ds
	IQ	Qs
	θ	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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

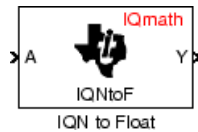
See Also

Clarke Transformation, Park Transformation, PID Controller, Space Vector Generator, Speed Measurement

Purpose Convert IQ number to floating-point number

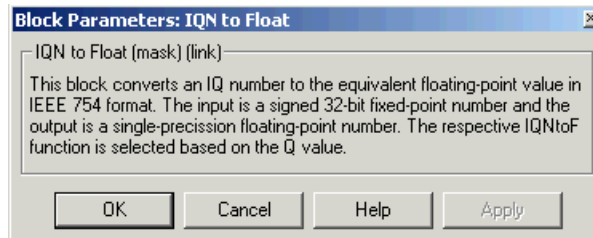
Library “C28x IQmath (tiiqmathlib)” on page 6-11

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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



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

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

IQN x int32

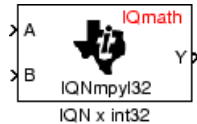
Purpose

Multiply IQ number with long integer

Library

“C28x IQmath (tiiqmathlib)” on page 6-11

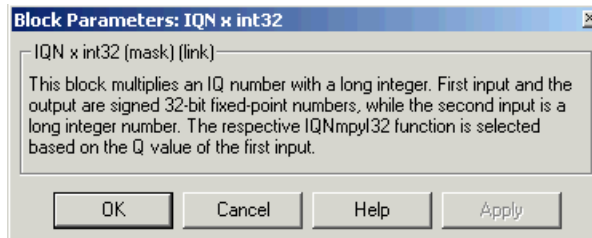
Description



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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

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

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

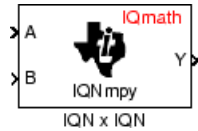
Purpose

Multiply IQ numbers with same Q format

Library

“C28x IQmath (tiiqmathlib)” on page 6-11

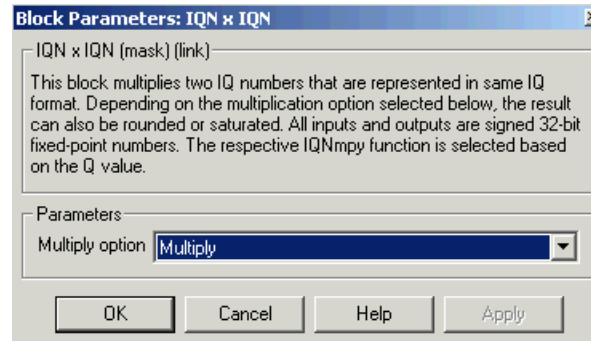
Description



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 Chapter 4, “Using the IQmath Library” 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.

References

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

See Also

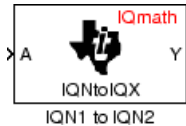
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Purpose Convert IQ number to different Q format

Library “C28x IQmath (tiiqmathlib)” on page 6-11

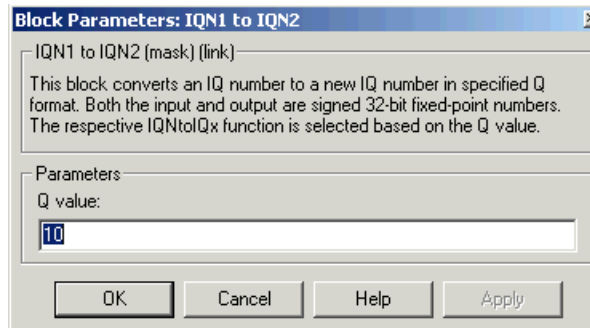
Description

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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



Q value

Q value from 1 to 30 that specifies the precision of the output

References

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

IQN1 to IQN2

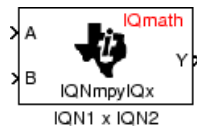
See Also

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 “C28x IQmath (tiiqmathlib)” on page 6-11

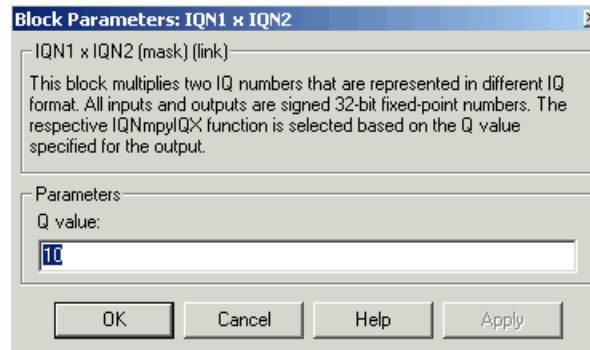
Description



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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



Q value

Q value from 1 to 30 that specifies the precision of the output

References

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

IQN1 x IQN2

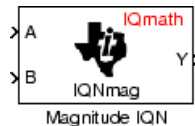
See Also

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

Purpose Magnitude of two orthogonal IQ numbers

Library “C28x IQmath (tiiqmathlib)” on page 6-11

Description This block calculates the magnitude of two IQ numbers using

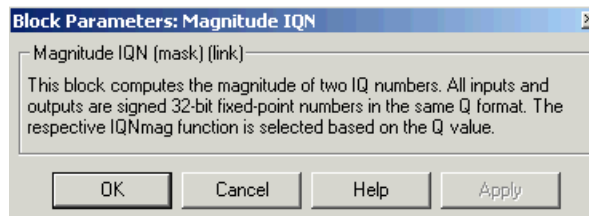


$$\sqrt{a^2 + b^2}$$

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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

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

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

Park Transformation

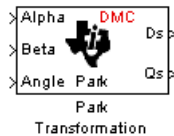
Purpose

Convert two-phase stationary system vectors to rotating system vectors

Library

“C28x DMC (c28xdmclib)” on page 6-10

Description

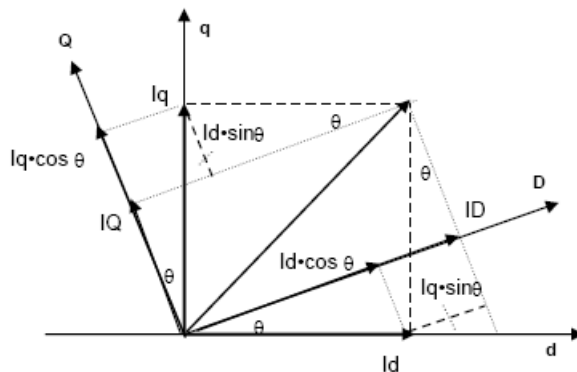


This block converts vectors in balanced two-phase orthogonal stationary systems into an orthogonal rotating reference frame. The transformation implements these equations

$$ID = Id * \cos \theta + Iq * \sin \theta$$

$$IQ = -Id * \sin \theta + Iq * \cos \theta$$

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
	θ	Angle
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.

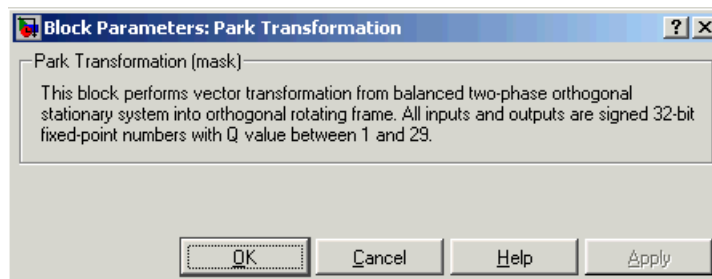
The instantaneous inputs are defined by the following equations:

$$id = I * \sin(\omega t)$$

$$iq = I * \sin(\omega t + \pi / 2)$$

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 Chapter 4, “Using the IQmath Library” for more information.

Dialog Box



References

For detailed information on the DMC library, see *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

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

$$u_i(t) = \int_0^t \left\{ \frac{K_p}{T_i} e(\tau) + K_c (u(\tau) - u_{presat}(\tau)) \right\} d\tau$$

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.

Using backward approximation, the preceding differential equations can be transformed into the following discrete equations.

$$u_p[n] = K_p e[n]$$

$$u_i[n] = u_i[n-1] + K_i K_p e[n] + K_c (u[n-1] - u_{presat}[n-1])$$

$$u_d[n] = K_d K_p (e[n] - e[n-1])$$

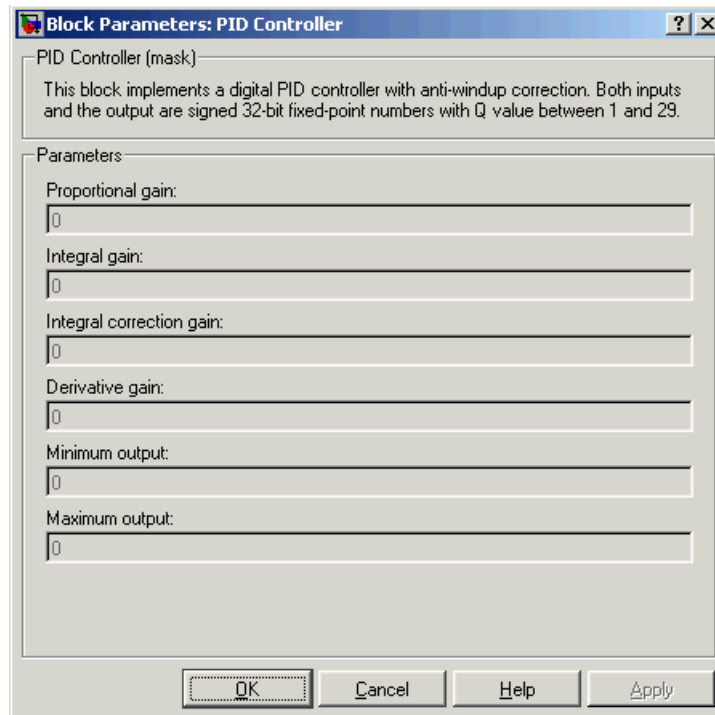
$$u_{presat}[n] = u_p[n] + u_i[n] + u_d[n]$$

$$u[n] = SAT(u_{presat}[n])$$

PID Controller

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 Chapter 4, “Using the IQmath Library” for more information.

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_i) 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

For detailed information on the DMC library, see *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

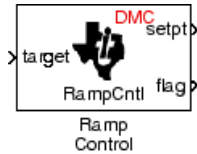
Purpose

Create ramp-up and ramp-down function

Library

“C28x DMC (c28xdmclib)” on page 6-10

Description

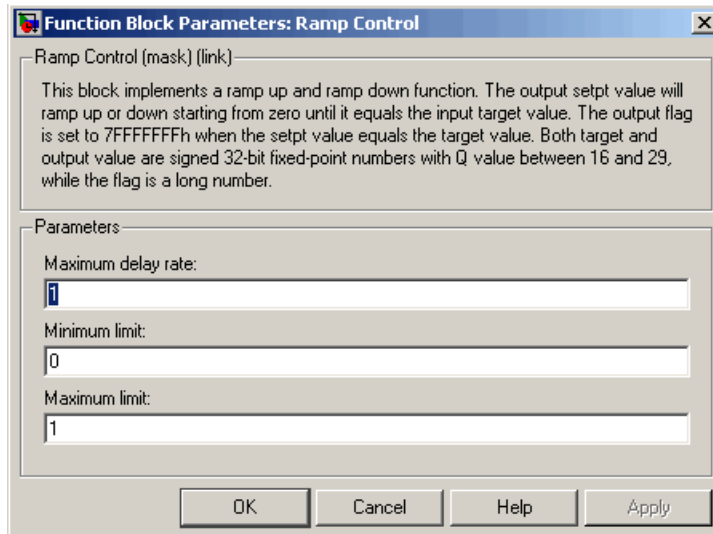


This block implements a ramp-up and ramp-down function. The input is a target value and the outputs are the set point value (`setpt`) and a flag. The `flag` output is set to `7FFFFFFFh` when the output `setpt` value reaches the input target value. The target and `setpt` values are signed 32-bit fixed-point 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

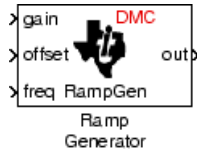
Ramp Generator

Ramp Generator

Purpose Generate ramp output

Library “C28x DMC (c28xdmclib)” on page 6-10

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:

$$\text{“out}(k) = \text{angle}(k) * \text{gain}(k) + \text{offset}(k) \text{”}$$

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:

$$\text{“angle}(k) = \text{angle}(k-1) + \text{freq}(k) * \text{Maximum step angle} \text{”}$$

$$\text{for angle}(k) > 1, \text{angle}(k) = \text{angle}(k) - 1$$

$$\text{for angle}(k) < -1, \text{angle}(k) = \text{angle}(k) + 1 \text{”}$$

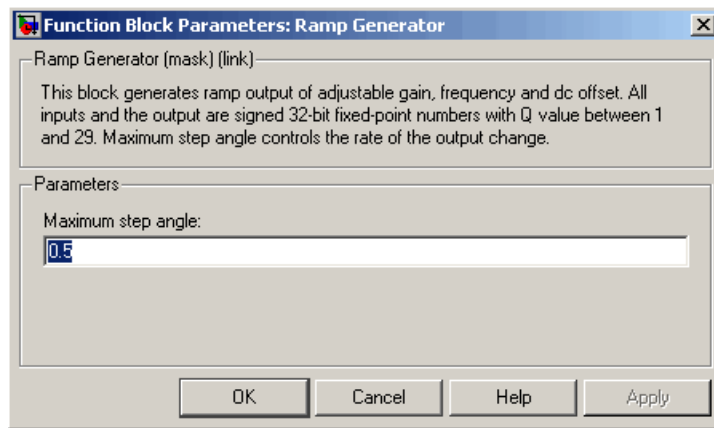
The frequency of the ramp output is controlled by a precision frequency generation algorithm that relies on the modulo nature of the finite length variables. The frequency of the output ramp signal is equal to

$$\text{“}f = (\text{Maximum step angle} * \text{sampling rate}) / 2^m \text{”}$$

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

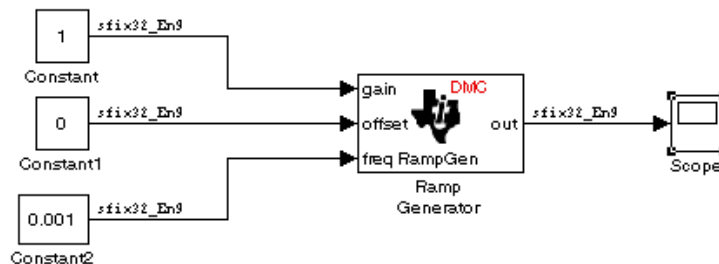


Maximum step angle

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.

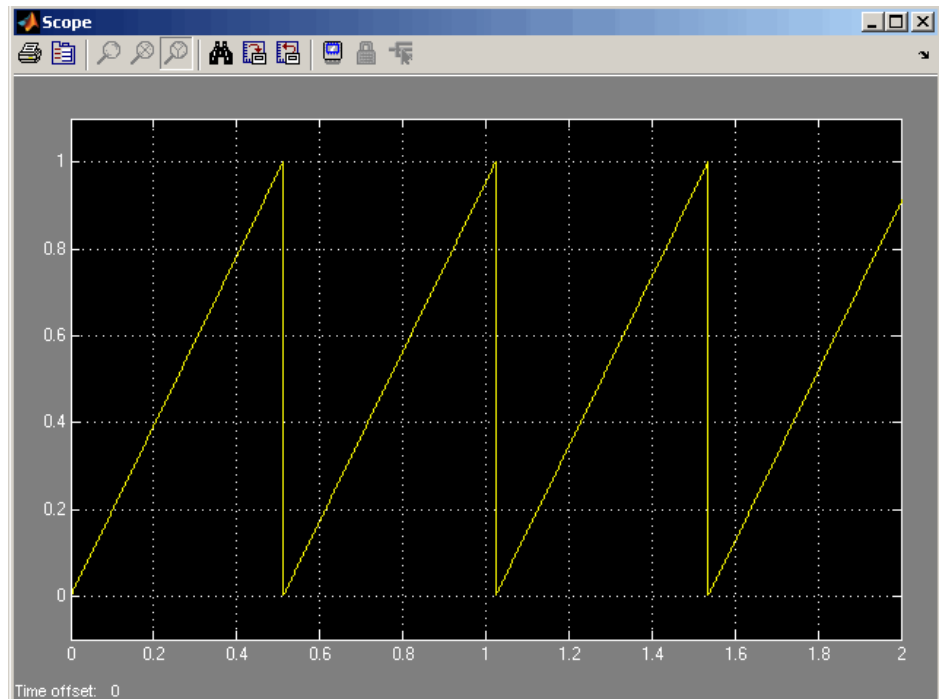


In your model, select **Simulation > Configuration Parameters**. On the **Solver** pane, set **Type** to Fixed-step and **Solver** to Discrete (no continuous states). Set the parameter values for the blocks as shown in the following table.

Ramp Generator

Block	Connects to	Parameter	Value
Constant	Ramp Generator - gain	Constant value	1
		Sample time	0.001
		Output data type	sfix(32)
		Output scalig value	2 ⁻⁹
Constant	Ramp Generator - offset	Constant value	0
		Sample time	inf
		Output data type	sfix(32)
		Output scalig value	2 ⁻⁹
Constant	Ramp Generator - freq	Constant value	0.001
		Sample time	inf
		Output data type	sfix(32)
		Output scalig value	2 ⁻⁹
Ramp Generator	Scope and Floating 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

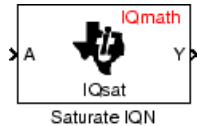
Ramp Control

Saturate IQN

Purpose Saturate IQ number

Library “C28x IQmath (tiiqmathlib)” on page 6-11

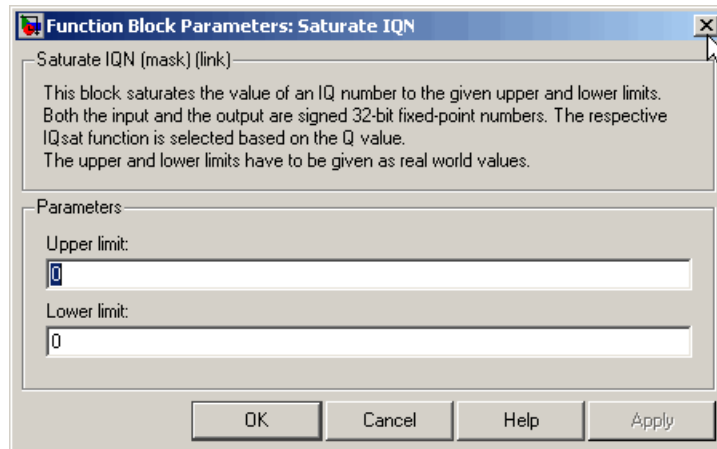
Description



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 Chapter 4, “Using the IQmath Library” 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

References

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

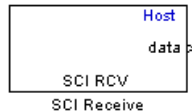
See Also

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 “Host SCI Blocks (c2000scilib)” on page 6-14



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.

Dialog Box

Source Block Parameters: SCI Receive

c2000 Host SCI Receive (mask) [link]

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

Parameters:

Port name: COM 1

Additional package header: S

Additional package terminator: E

Data type: uint8

Data length: 1

Initial output: 0

Action taken when connection times out: Output the last received value

Sample time: -1

Output receiving status

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).

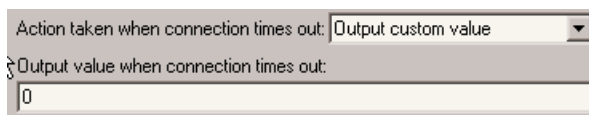
Initial output

Default value from the Receive block. This value is used, for example, if a connection time-out occurs and the **Action taken when connection timeout** field is set to “Output the last received value”, but nothing yet has been received.

Action Taken when connection times out

Specify 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 you select "Output custom value", use the "Output value when connection times out" field to set the custom value.

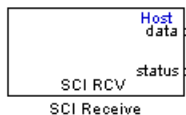


Sample time

Determines how often the SCI Receive block is called (in seconds). When you set this value to -1, the model inherits the sample time value of the model. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-11 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.



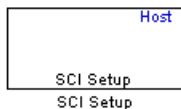
The error status may be one of the following values:

SCI Receive

- 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

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

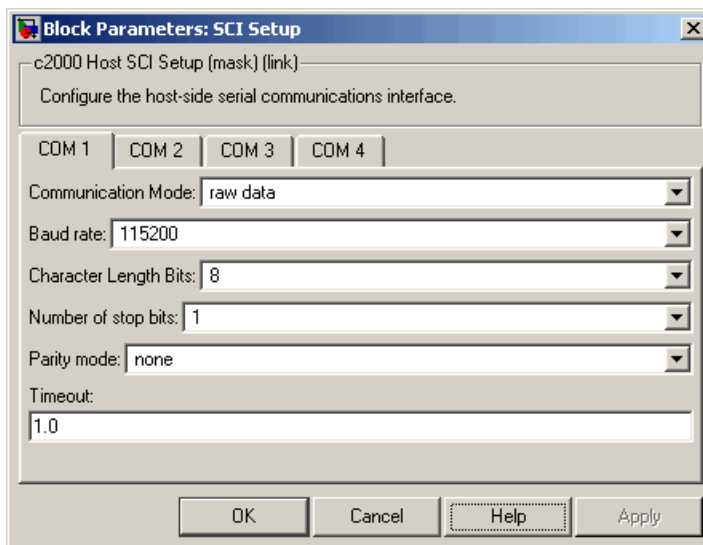
Library “Host SCI Blocks (c2000scilib)” on page 6-14



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.

SCI Transmit

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

Library “Host SCI Blocks (c2000scilib)” on page 6-14



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.

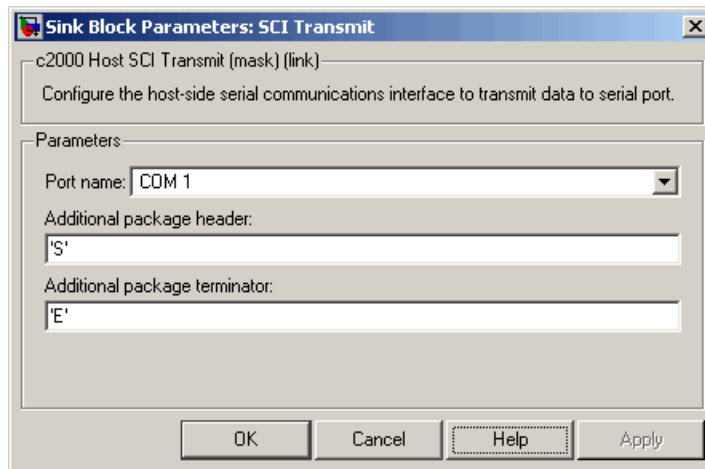
Acceptable data types are `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. The byte size of each data type is as follows:

Data Type	Byte Count
<code>single</code>	4 bytes
<code>int8</code> & <code>uint8</code>	1 byte
<code>int16</code> & <code>uint16</code>	2 bytes
<code>int32</code> & <code>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.

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.

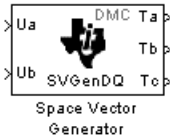
Purpose

Duty ratios for stator reference voltage

Library

“C28x DMC (c28xdmclib)” on page 6-10

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

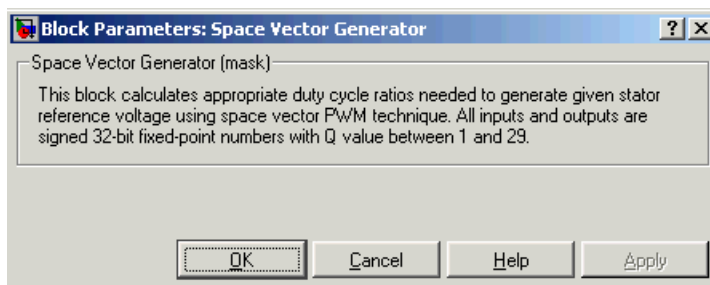
- Alpha component — the reference stator voltage vector on the direct axis stationary reference frame (U_a)
- Beta component — the reference stator voltage vector on the direct axis quadrature reference frame (U_b)

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 Chapter 4, “Using the IQmath Library” for more information.

Space Vector Generator

Dialog Box



References

For detailed information on the DMC library, see *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

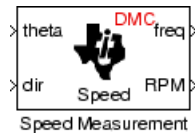
Purpose

Calculate motor speed

Library

“C28x DMC (c28xdmclib)” on page 6-10

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 Chapter 4, “Using the IQmath Library” 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),

Speed Measurement

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, θ_{new} and θ_{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 θ_{new} and θ_{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

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

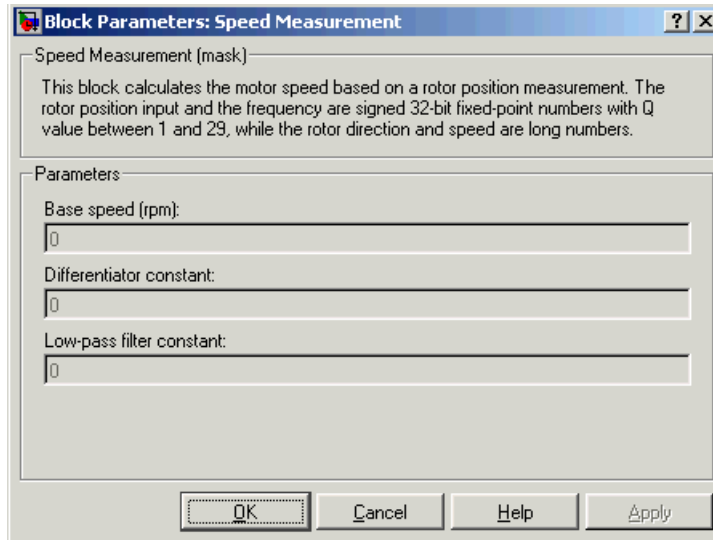
The differentiator constant is a scalar value applied to the block output. For example, setting it to 1 produces no effect on the output. Setting the constant to 1/4 multiplies the frequency and revolutions per minute outputs by 0.25. This setting can be useful when your motor has multiple pole pairs, 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.

Speed Measurement

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.

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, see *C/F 28xx Digital Motor Control Library*, SPRC080, available at the Texas Instruments Web site.

See Also

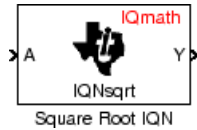
Clarke Transformation, Inverse Park Transformation, Park Transformation, PID Controller, Space Vector Generator

Square Root IQN

Purpose Square root or inverse square root of IQ number

Library “C28x IQmath (tiiqmathlib)” on page 6-11

Description

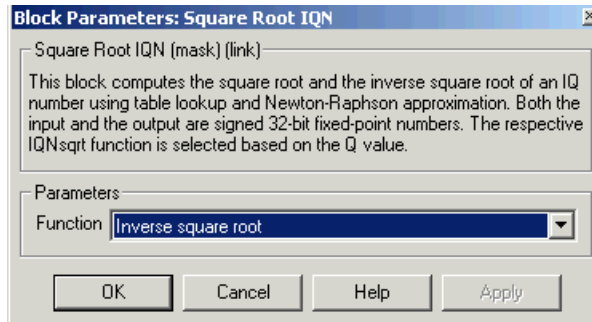


This block calculates the square root or inverse square root of an IQ number and returns an IQ number of the same Q format. The block uses table lookup and a Newton-Raphson approximation.

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 Chapter 4, “Using the IQmath Library” 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.

References

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

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

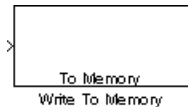
Switch External Mode Configuration

Purpose	Configure model for external mode or executable building
Library	Target Support Package software/ C2000 Driver Library/ Utilities
Description	<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 Building an executable or External mode, and click OK.</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">1 Inline parameters are selected (under Optimization in the Configuration Parameters dialog box). This is required for ASAP2 generation2 Normal simulation mode is selected (in the Simulation menu, and drop-down list in the toolbar).3 ASAP2 is selected as the Interface (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box). <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">1 Inline parameters are selected (under Optimization in the Configuration Parameters dialog box). This is required for external mode.2 External simulation mode is selected (in the Simulation menu, and drop-down list in the toolbar).3 External mode is selected as the Interface (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).

Purpose Write data to target memory

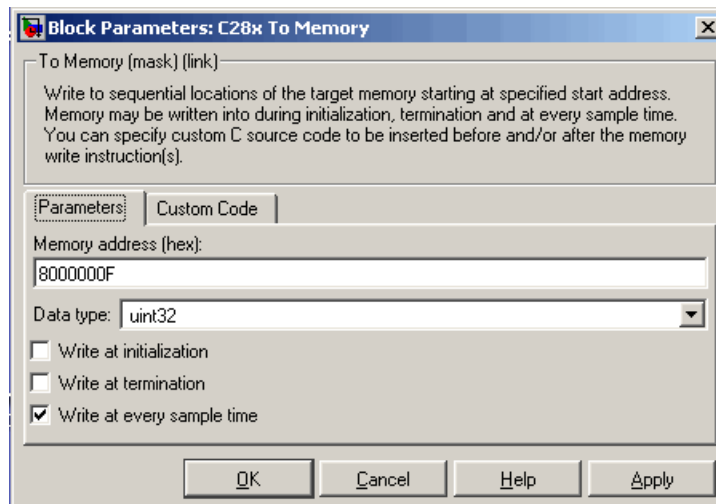
Library “C280x Chip Support (c280xlib)” on page 6-2 or “C281x Chip Support (c281xlib)” on page 6-6

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

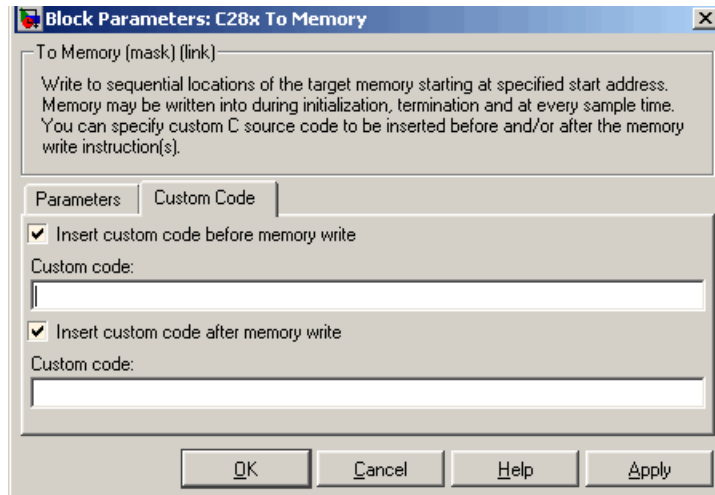
Write at every sample time

Whether to write data in real time during program execution

Note If your Write 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 Write 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 (" EDIS ");
```

which disables write access to the device emulation registers on the C2812 DSP.

See Also

From Memory

To RTDX

Purpose

Add RTDX communication channel to send data from target to host

Library

“RTDX Instrumentation (rtdxBlocks)” on page 6-15

Description



Note This block will be removed from the Target Support Package product in an upcoming release. Consider using TCP/IP or UDP blocks instead.

When you generate code from Simulink in Real-Time Workshop software with a To RTDX block in your model, code generation inserts the C commands to create an RTDX output channel on the target DSP. The output channels transfer data from the target DSP 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:

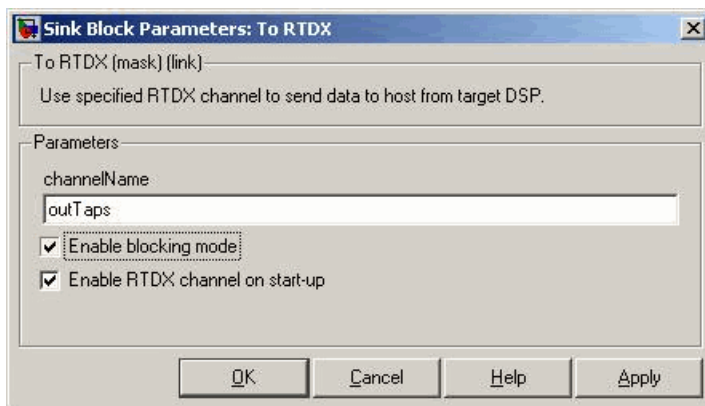
- 1 Add one or more To RTDX or From RTDX blocks to your model.
- 2 Download and run your model on your target.
- 3 Enable the RTDX channels from MATLAB or use **Enable RTDX channel on start-up** on the block dialog.

- 4 Use the `readmsg` and `writemsg` functions on the MATLAB command line to send and retrieve data from the target over RTDX.

To see more details about using RTDX in your model, refer to the Embedded IDE Link User's Guide and the following demos:

- Real-Time Data Exchange (RTDX™) Tutorial
- Sine Wave Generation via RTDX™
- DC Motor Speed Control via 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. Any interrupt of the higher priority will temporary 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 Embedded IDE Link software 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

From RTDX

References

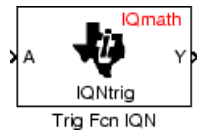
RTDX 2.0 User's Guide, Literature Number: SPRUFC7, available from the Texas Instruments Web site.

How to Write an RTDX Host Application Using MATLAB, Literature Number: SPRA386, available from the Texas Instruments Web site.

Purpose Sine, cosine, or arc tangent of IQ number

Library “C28x IQmath (tiiqmathlib)” on page 6-11

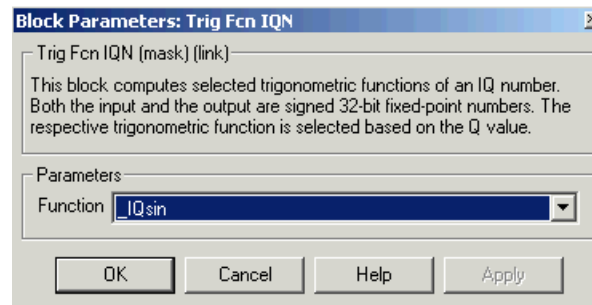
Description



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 Chapter 4, “Using the IQmath Library” 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

References

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

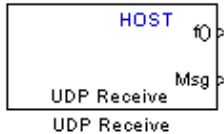
See Also

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

Purpose Receive uint8 vector as UDP message

Library Host Communication (hostcommlib)

Description



A UDP message comes into this block from the transport layer. The block passes the message to the next downstream block. One block output provides the data vector from the message. The second output is a flag that indicates when a new UDP message is available.

Models can contain only one UDP Receive block.

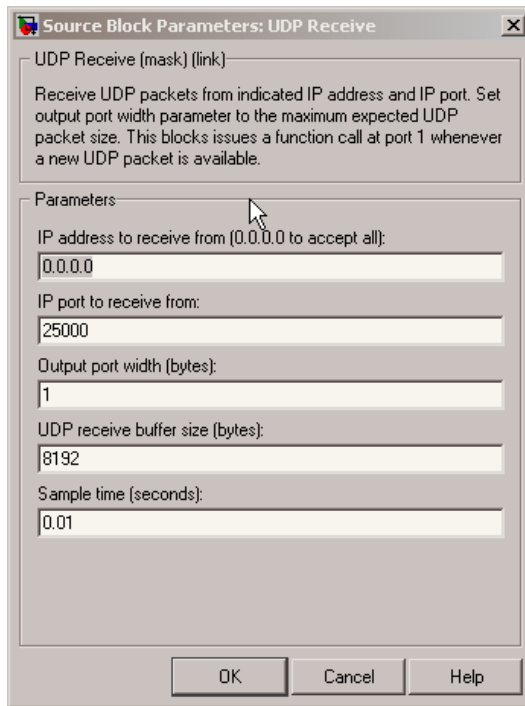
This block issues a function call from the `fcn` port when a new UDP packet becomes available. At the same time, it updates the signal going out of the `msgport` with the contents of the UDP packet. It reads a single UDP packet every sample hit. It does not attempt to receive multiple UDP packets to fill the output vector.

If the UDP packet size is greater than the output port width parameter, the system truncates the UDP messages at the `Msg` port. As a result, the system discards the part of the UDP packet that does not fit into the `Msg` port. The system cannot recover discarded message content.

In some cases, the UDP packet size is smaller than the `Msg` port width. When this condition occurs, the portion of the output vector that does not fit into the specified size processes as invalid data.

UDP Receive

Dialog Box



IP address to receive from (0.0.0.0 to accept all)

Specifies the IP address from which the block accepts messages. Setting the address 0.0.0.0 configures the block to accept messages from any IP address. Setting a specific address, instead of the default value, 0.0.0.0, directs the block to accept messages from the specified address only.

IP port to receive from

Specify the port the block accepts messages from on this machine. The other end of the communication, usually a UDP Send block, sends messages to this port. The value defaults to 25000, but the values range from 1–65535.

Output port width (bytes)

Specifies the width of messages that the block accepts. When you design the transmit end of the UDP communication channel, you decide the message width. Set this option to a value as large or larger than any message you expect to receive.

UDP receive buffer size (bytes)

Specify the size of the buffer to which the system stores UDP messages. The default size is 8192 bytes. Make the buffer large enough to store UDP messages that come in while your process reads a message from the buffer or performs other tasks. Specifying the buffer size prevents the receive buffer from overflowing.

Sample time (seconds)

Use this option to specify when the block polls for new messages. Enter a value that is greater than zero. Setting this option to a large value reduces the likelihood of dropped UDP messages. By default, the sample time is 0.01 s.

See Also

Byte Pack, Byte Reversal, Byte Unpack, UDP Send

UDP Send

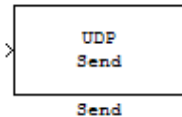
Purpose

Send UDP message

Library

Host Communication (hostcommlib)

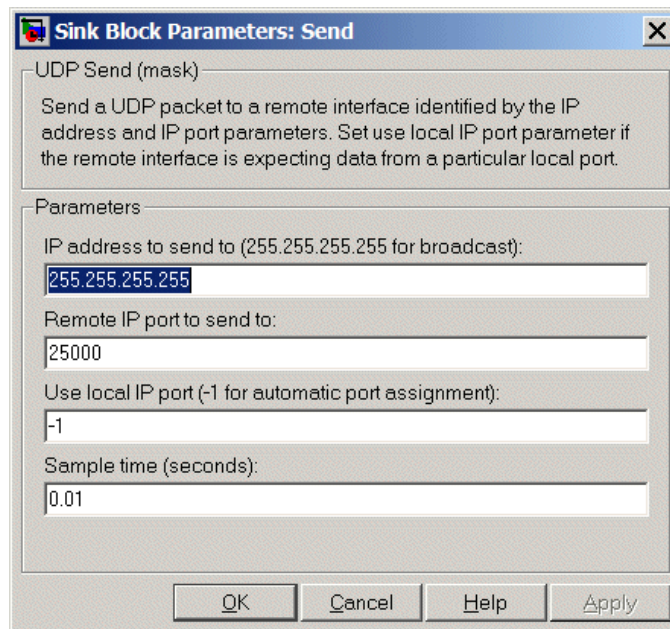
Description



The UDP send block receives a uint8 vector that it sends as a UDP message to the host. Input must be in the form of a uint8 vector with UDP format.

Models can contain only one UDP Send block.

Dialog Box



IP address to send to (255.255.255.255 for broadcast)

Specify the IP address to which the block sends the message.

To broadcast the UDP message, retain the default value, 255.255.255.255.

IP port to send to

Specify the port to which the block sends the message. Port numbers range from 1 to 65535. Configure the network port receiving the UDP messages with the same port number.

Use the following local IP port

Specify the local IP port the block sends the message from. Entering -1 (the default value) for this option allows the network to select automatically the local IP port to use to send the message.

If the address you are sending to expects the message to come from a specific port, enter that port address. If you enter a port number in the UDP Receive block option **IP port to receive from**, enter that port identifier instead of the port address.

Sample time

Sample time tells the block how long to wait before polling for new messages.

See Also

Byte PackByte Reversal, Byte Unpack, UDP Receive

UDP Send

A

- Absolute IQN block 7-2
- acquisition window
 - ADC blocks
 - ACQ_PS 3-2
- ADC blocks
 - C281x 7-185
- Arctangent IQN block 7-3
- ASAP2 files, generating 2-18
- asymmetric vs. symmetric waveforms 7-217
- asynchronous interrupt processing 1-11

B

- blocks
 - adding to model 1-36
 - CAN Calibration Protocol 7-21
 - CAN Pack 7-27
 - CAN Unpack 7-39
 - recommendations 1-26
 - Switch External Mode Configuration 7-318
- Byte Pack block 7-5
- Byte Reversal block 7-8
- Byte Unpack block 7-10

C

- C2000 Library
 - SCI Receive
 - Host-side 7-298
 - SCI Setup
 - Host-side 7-303
 - SCI Transmit
 - Host-side 7-306
- c2000lib startup 1-32
- C2802x ADC 7-147
- C2802x AnalogIO Input 7-152
- C2802x AnalogIO Output 7-154
- C2802x COMP 7-145
- C2802x ePWM 7-156

- C280x/C28x3x ADC block 7-13
- C280x/C28x3x eCAN Receive block 7-51
- C280x/C28x3x eCAN Transmit block 7-56
- C280x/C28x3x ePWM block 7-71
- C280x/C28x3x eQEP block 7-96
- C280x/C28x3x/C2802x eCAP block 7-60
- C280x/C28x3x/C2802x GPIO Digital Input 7-114
- C280x/C28x3x/C2802x GPIO Digital Output 7-117
- C280x/C28x3x/C2802x I2C Receive block 7-119
- C280x/C28x3x/C2802x I2C Transmit block 7-123
- C280x/C28x3x/C2802x SCI Receive block 7-126
- C280x/C28x3x/C2802x SCI Transmit block 7-133
- C280x/C28x3x/C2802x Software Interrupt Trigger 7-136
- C280x/C28x3x/C2802x SPI Receive block 7-139
- C280x/C28x3x/C2802x SPI Transmit block 7-142
- C281x ADC block 7-185
- C281x CAP block 7-190
- C281x eCAN Receive block 7-199
- C281x eCAN Transmit block 7-203
- C281x GPIO Digital Input block 7-207
- C281x GPIO Digital Output block 7-211
- C281x PWM block 7-215
- C281x QEP block 7-227
- C281x SCI Receive block 7-231
- C281x SCI Transmit block 7-237
- C281x Software Interrupt Trigger 7-240
- C281x SPI Receive block 7-243
- C281x SPI Transmit block 7-246
- C281x Timer block 7-249
- C28x3x GPIO Digital Input 7-114
- C28x3x GPIO Digital Output 7-117
- CAN Calibration Protocol block 7-21
- CAN Pack block 7-27
- CAN Unpack block 7-39
- CAN/eCAN
 - C280x/C2833xReceive block 7-51
 - C280x/C28x3x Transmit block 7-56
 - C281x Transmit block 7-203

- C281xReceive block 7-199
 - timing parameters
 - bit rate 2-3
- capture block
 - C281x 7-190
- CCS 1-7
 - See also* Code Composer Studio™
- Clarke Transformation block 7-256
- clock speed 1-11
- Code Composer Studio™ 1-7
- code generation
 - overview 1-39
- code optimization 4-11
- configuration default 1-7
- configuration parameters
 - setting 1-28
- conversion
 - float to IQ number 7-260
 - IQ number to different IQ number 7-279
 - IQ number to float 7-275
- CPU clock speed 1-11

D

- data type support 1-9
- data types
 - conversion 4-10
- deadband
 - C281x PWM 7-223
- default build configuration 1-7
- device driver blocks
 - CAN Calibration Protocol 7-21
- digital motor control. *See* DMC library
- Division IQN block 7-259
- DMC library
 - Clarke Transformation 7-256
 - Inverse Park Transformation 7-273
 - Park Transformation 7-284
 - PID controller 7-286
 - ramp control 7-290

- ramp generator 7-292
- Space Vector Generator 7-309
- Speed Measurement 7-311
- duty ratios 7-309

E

- enhanced capture channel 7-60
- enhanced quadrature encoder pulse module
 - C280x/C2833x 7-96
- ePWM blocks
 - C280x/C2833x 7-71

F

- fixed-point numbers 4-4
- Float to IQN block 7-260
- floating-point numbers
 - convert to IQ number 7-260
- four-quadrant arctangent 7-3
- Fractional part IQN block 7-262
- Fractional part IQN x int32 block 7-263
- From RTDX block 7-266

G

- GPIO Digital Input
 - C280x 7-114
 - C28x3x 7-114
- GPIO Digital Output
 - C280x 7-117
 - C28x3x 7-117
- GPIO input
 - C281x 7-207
- GPIO output
 - C281x 7-211

H

- hardware 1-3
- high-speed peripheral clock 1-11

I

I/O

C281x input 7-207

C281x output 7-211

I2C

C280x/C28x3x Receive 7-119

C280x/C28x3x Transmit 7-123

installing software 1-3

Integer part IQN block 7-271

Integer part IQN x int32 block 7-272

interrupt

software triggered for C280x/C28x3x 7-136

software triggered for C281x 7-240

Inverse Park Transformation block 7-273

IQ Math library 4-2

Absolute IQN block 7-2

Arctangent IQN block 7-3

building models 4-10

code optimization 4-11

common characteristics 4-3

Division IQN block 7-259

Float to IQN block 7-260

Fractional part IQN block 7-262

Fractional part IQN x int32 block 7-263

Integer part IQN block 7-271

Integer part IQN x int32 block 7-272

IQN to Float block 7-275

IQN x int32 block 7-276

IQN x IQN block 7-277

IQN1 to IQN2 block 7-279

IQN1 x IQN2 block 7-281

Magnitude IQN block 7-283

Q format notation 4-5

Saturate IQN block 7-296

Square Root IQN block 7-316

Trig Fcn IQN block 7-325

IQ numbers

convert from float 7-260

convert to different IQ 7-279

convert to float 7-275

fractional part 7-262

integer part 7-271

magnitude 7-283

multiply 7-277

multiply by int32 7-276

multiply by int32 fractional result 7-263

multiply by int32 integer part 7-272

square root 7-316

trigonometric functions 7-325

IQN to Float block 7-275

IQN x int32 block 7-276

IQN x IQN block 7-277

IQN1 to IQN2 block 7-279

IQN1 x IQN2 block 7-281

M

Magnitude IQN block 7-283

math blocks. *See* IQ Math library

memory management 1-28

messages

F2812 eZdsp 7-200

model

add blocks 1-36

building overview 1-29

creation overview 1-25

IQmath library 4-10

multiplication

IQN x int32 7-276

IQN x int32 fractional part 7-263

IQN x int32 integer part 7-272

IQN x IQN 7-277

IQN1 x IQN2 7-281

O

optimization code 4-11

P

Park Transformation block 7-284

phase conversion 7-256

PID controller 7-286

PWM blocks

C281x 7-215

Q

Q format 4-5

quadrature encoder pulse circuit

C28x 7-227

R

ramp control block 7-290

ramp generator block 7-292

Read From Memory block 7-264

reference frame conversion

inverse Park transformation 7-273

Park transformation 7-284

reset 1-29

RTDX

from 7-266

to 7-322

S

sample time

F2812 eZdsp 7-53

Saturate IQN block 7-296

scheduling 1-10

Scheduling

watchdog 7-254

SCI Receive

Host-side 7-298

SCI Setup

Host-side 7-303

SCI Transmit

Host-side 7-306

SCI Transmit and Receive blocks

Host-side

Setup 7-303

serial communications interface

C280x/C28x3x receive 7-126

C280x/C28x3x transmit 7-133

C281x receive 7-231

C281x transmit 7-237

serial peripheral interface

C280x/C28x3x receive 7-139

C280x/C28x3x transmit 7-142

C281x receive 7-243

C281x transmit 7-246

signed fixed-point numbers 4-5

simulation parameters

automatic 1-34

Space Vector Generator block 7-309

Speed Measurement block 7-311

Square Root IQN block 7-316

startup c2000lib 1-3

supported hardware 1-3

Switch External Mode Configuration block 7-318

system requirements 1-3

T

target model creation 1-25

timing

interrupts 1-10

To RTDX block 7-322

Trig Fcn IQN block 7-325

U

UDP Receive block 7-327

UDP Send block 7-330

W

waveforms 7-217

Write To Memory block 7-319