

Embedded Target for the TI TMS320C2000™ DSP Platform

For Use with Real-Time Workshop®

- Modeling
- Simulation
- Implementation

User's Guide

Version 2



How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Embedded Target for the TI TMS320C2000™ DSP Platform User's Guide

© COPYRIGHT 2003–2006 by The MathWorks, Inc.

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

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

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks, and SimBiology, SimEvents, and SimHydraulics are trademarks of The MathWorks, Inc.

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

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

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

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

Getting Started

1

What Is the Embedded Target for the TI TMS320C2000	
DSP Platform?	1-2
Suitable Applications	1-2
Setting Up and Configuring	1-3
Platform Requirements — Hardware and Operating System	1-3
Supported Hardware for Targets	1-3
Software Requirements	1-5
Verifying the Configuration	1-6
Embedded Target for TI C2000 and Code Composer Studio	1-9
Default Project Configuration	1-9
Data Type Support	1-10
Scheduling and Timing	1-11
Timer-Based Interrupt Processing	1-11
Asynchronous Interrupt Processing	1-13
Overview of Creating Models for Targeting	1-18
Online Help	1-18
Blocks with Restrictions	1-19
S-Function Builder Blocks	1-21
Setting Simulation Configuration Parameters	1-21
Building Your Model	1-22
Using the c2000lib Blockset	1-24
Hardware Setup	1-24
Starting the c2000lib Library	1-24
Setting Up the Model	1-26
Adding Blocks to the Model	1-32

Generating Code from the Model	1-35
Creating Code Composer Studio Projects Without Loading	1-36

Configuring Timing Parameters for CAN Blocks

2

Blocks Where the Bit Rate Cannot Be Set Directly	2-2
Setting Timing Parameters	2-3
Equations for Bit Rate Calculation	2-6
CAN Bit Timing Examples	2-7

Configuring Acquisition Window Width for ADC Blocks

3

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

Creating Stand-Alone Applications by Saving Code into Flash Memory

4

The Need for Stand-Alone Applications	4-2
Generating Code for Flash Memory	4-3
Running Code from Flash Memory	4-5

Using the IQmath Library

5

About the IQmath Library	5-2
Common Characteristics	5-2
Fixed-Point Numbers	5-4
Signed Fixed-Point Numbers	5-4
Q Format Notation	5-5
Building Models	5-9
Converting Data Types	5-9
Using Sources and Sinks	5-9
Choosing Blocks to Optimize Code	5-9

Blocks — By Category

6

C2000 Target Preferences (c2000tgtpreflib)	6-2
Host-Side CAN Blocks (c2000canlib)	6-3
Host-Side SCI Blocks (c2000scilib)	6-4
C2000 RTDX Instrumentation (rtDXBlocks)	6-5
C2400 DSP Chip Support (c2400dspchiplib)	6-6
C280x DSP Chip Support (c280xdspchiplib)	6-7
C281x DSP Chip Support (c281xdspchiplib)	6-8
C28x Digital Motor Control (c28xdmelib)	6-10
C28x IQmath (tiiqmathlib)	6-11

Blocks — Alphabetical List

7

Index

Getting Started

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

What Is the Embedded Target for the TI TMS320C2000 DSP Platform? (p. 1-2)	Introduces the Embedded Target for TI C2000 DSP and describes some of its features and supported hardware
Setting Up and Configuring (p. 1-3)	Describes the software and hardware required to use the Embedded Target for the TI TMS320C2000™ DSP Platform and how to set them up
Embedded Target for TI C2000 and Code Composer Studio (p. 1-9)	Provides information about Code Composer Studio™
Data Type Support (p. 1-10)	Compares the data types supported by Simulink and the TI C2000 DSP chips
Scheduling and Timing (p. 1-11)	Provides information about TI C2000 scheduling
Overview of Creating Models for Targeting (p. 1-18)	Summarizes the steps required to create models for your target
Using the c2000lib Blockset (p. 1-24)	Provides an example of creating a model and targeting hardware

What Is the Embedded Target for the TI TMS320C2000 DSP Platform?

The Embedded Target for the TI TMS320C2000 DSP Platform integrates Simulink and MATLAB with Texas Instruments eXpressDSP™ tools. You can use this product to develop and validate digital signal processing and control designs from concept through code.

The Embedded Target for the TI TMS320C2000 DSP Platform uses C code generated by Real-Time Workshop® and your TI development tools to generate a C language real-time implementation of your Simulink model. Real-Time Workshop builds a Code Composer Studio™ project from the C code.

You can compile, link, download, and execute the generated code on an LF2407, F2808, or F2812 eZdsp™ DSP board from Spectrum Digital, Inc. or on a custom board based on a TI C280x or C281x chip.

Suitable Applications

The Embedded Target for the TI TMS320C2000 DSP Platform enables you to develop digital signal processing and control applications. Some important characteristics of the applications that you can develop are

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

Setting Up and Configuring

Platform Requirements – Hardware and Operating System

To run the Embedded Target for the TI TMS320C2000 DSP Platform, your host PC must meet the following hardware configuration requirements:

- Intel Pentium or Intel Pentium processor-compatible PC
- 64 MB RAM (128 MB recommended)
- 20 MB hard disk space available after installing MATLAB
- Color monitor
- One parallel printer port or one USB port to connect your target board to your PC
- CD-ROM drive
- Windows 2000 or Windows XP

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

Supported Hardware for Targets

The Embedded Target for TI C2000 DSP supports the following boards:

- DSP Starter Kits (DSKs) from Spectrum Digital, Inc.
 - TMS320F2812 eZdsp DSK — The F2812eZdsp DSP Starter Kit
 - TMS320F2808 eZdsp DSK — The F2808eZdsp DSP Starter Kit
 - TMS320LF2407 eZdsp DSK — The LF2407eZdsp DSP Starter Kit

The above DSKs help developers evaluate digital signal processing applications for the Texas Instruments DSP chips. You can create, test, and deploy your processing software and algorithms on the target processor without the difficulties inherent in starting with the digital signal processor itself and building the support hardware to test the application on the

processor. Instead, the development board provides the input hardware, output hardware, timing circuitry, memory, and power for the digital signal processor. Texas Instruments provides the software tools, such as the C compiler, linker, assembler, and integrated development environment, for PC users to develop, download, and test their algorithms and applications on the processor.

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

Note To generate code, and download the code to your target board, you do not need to change any jumpers from their factory defaults on either the LF2407 or F2812 target board.

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

Note In factory default condition, both the LF2407 and F2812 target boards are set to operate in microcontroller mode. The Embedded Target for the TI TMS320C2000 DSP Platform does not support microprocessor mode.

- Custom boards based on any of the following Texas Instruments C2000 Digital Signal Controllers:
 - TMS320F2801
 - TMS320F2802
 - TMS320F2806
 - TMS320F2808
 - TMS320F2809
 - TMS320C2810
 - TMS320F2810

- TMS320C2811
- TMS320F2811
- TMS320R2811
- TMS320C2812
- TMS320F2812
- TMS320R2812

Running Code from Flash Memory

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

Software Requirements

MathWorks Software

For information about other MathWorks software required to use the Embedded Target for the TI TMS320C2000 DSP Platform, refer to the MathWorks Web site — <http://www.mathworks.com>. Check the Products area for the Embedded Target for the TI TMS320C2000 DSP Platform.

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

Texas Instruments Software

In addition to the required software from The MathWorks, Embedded Target for the TI TMS320C2000 DSP Platform requires that you install the Texas Instruments development tools and software listed in the following table. Installing Code Composer Studio IDE Version 3.1 for the C2000 series installs the software shown.

Required TI Software for Targeting Your TI C2000 Hardware

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

Verifying the Configuration

To determine whether the Embedded Target for the TI TMS320C2000 DSP Platform is installed on your system, enter this command at the MATLAB prompt:

```
c2000lib
```

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

- C2800 RTDX Instrumentation
- C2000 Target Preferences
- Host-side CAN Blocks
- C281x DSP Chip Support
- C280x DSP Chip Support
- C2400 DSP Chip Support
- C28x IQMath Library
- C28x DMC Library
- Info block
- Demos block

If you do not see the listed libraries, or MATLAB does not recognize the command, you need to install the Embedded Target for the TI TMS320C2000 DSP Platform. Without the software, you cannot use Simulink and Real-Time Workshop to develop applications targeted to the TI boards.

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

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

```
ccsboardinfo
```

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

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

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

As a final test, launch CCS to ensure that it starts up successfully. For the Embedded Target for the TI TMS320C2000 DSP Platform to operate with CCS, the CCS IDE must be able to run on its own.

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

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

Embedded Target for TI C2000 and Code Composer Studio

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

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

Default Project Configuration

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

Default Build Options in the `custom_MW` Configuration

The default settings for `custom_MW` are the same as the Release project configuration in CCS, except for the compiler options. `custom_MW` uses `Function(-o2)` for the compiler optimization level.

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

The TI C2000 DSP chips support 16-bit data types and do not have native 8-bit data types. Simulink and the Embedded Target for TI C2000 support many data types, including 8-bit data types.

If you select `int8` or `uint8` in your model, your simulation will run 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, if you want the overflow behavior of the simulation and generated code to match for a Simulink Add block in your model, select **Saturate on integer overflow** in that block.

Scheduling and Timing

Normally the code generated by the Embedded Target for TI C2000 runs out of the context of a timer interrupt. Model blocks run in a periodical fashion clocked by the periodical interrupt whose period is tied to the base sample time of the model.

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

For C280x and C281x-based boards, Embedded Target for TI C2000 lets you model systems that include asynchronous hardware interrupt processing in addition to the tasks that are left to be handled in the context of the timer interrupt.

Timer-Based Interrupt Processing

For code that runs in the context of the timer interrupt, each iteration of the model solver is run after an interrupt has been posted and serviced by an interrupt service routine (ISR). The code generated for the C280x or C281x uses CPU_timer0. The code generated for the C24x uses an Event Manager (EV) timer, which you can select.

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:

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

The minimum achievable base rate sample time depends on the model complexity. The maximum value depends on the maximum timer period value ($2^{32}-1$ for the F2812 and F2808 or $2^{16}-1$ for the LF2407), the CPU clock speed and, for the LF2407, the **TimerClockPrescaler** setting in the appropriate Target Preferences block. The CPU clock speed for the LF2407 is 40 MHz, for the F2808 it is 100 MHz, and for the F2812 it is 150 MHz.

Maximum Sample Times

TimerClockPrescaler Setting	C24x Maximum Sample Time(s)	C280x Maximum Sample Time(s)	C281x Maximum Sample Time(s)
1	0.0016	42.94	28.63
2	0.0032	N/A	N/A
4	0.0065	N/A	N/A
8	0.0131	N/A	N/A
16	0.0262	N/A	N/A
32	0.0524	N/A	N/A
64	0.1048	N/A	N/A
128	0.2097	N/A	N/A

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

High-Speed Peripheral Clock

The Event Managers and their general-purpose timers, which drive PWM waveform generation use the high-speed peripheral clock (HISCLK). By default, this clock is always selected in the Embedded Target for TI C2000. This clock is derived from the system clock (SYSCLKOUT):

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

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

For example, on the F2812, the HISCLK rate becomes

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

Asynchronous Interrupt Processing

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

- Hardware Interrupt block

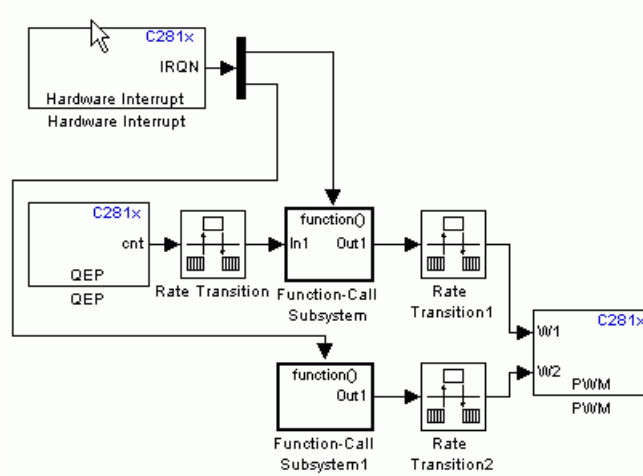
This block enables selected hardware interrupts, generates the corresponding interrupt service routines (ISRs), and connects them to the corresponding interrupt service vector table entries. When you connect the output of the Hardware Interrupt block to the control input of a triggered subsystem (for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

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

- Rate Transition blocks

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

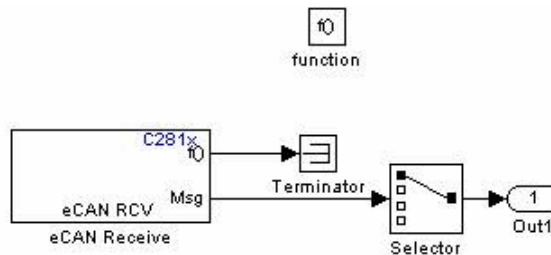
The following diagram illustrates a use case where a Hardware Interrupt block triggers two tasks, connected to other blocks that run periodically in the context of the synchronous scheduler.



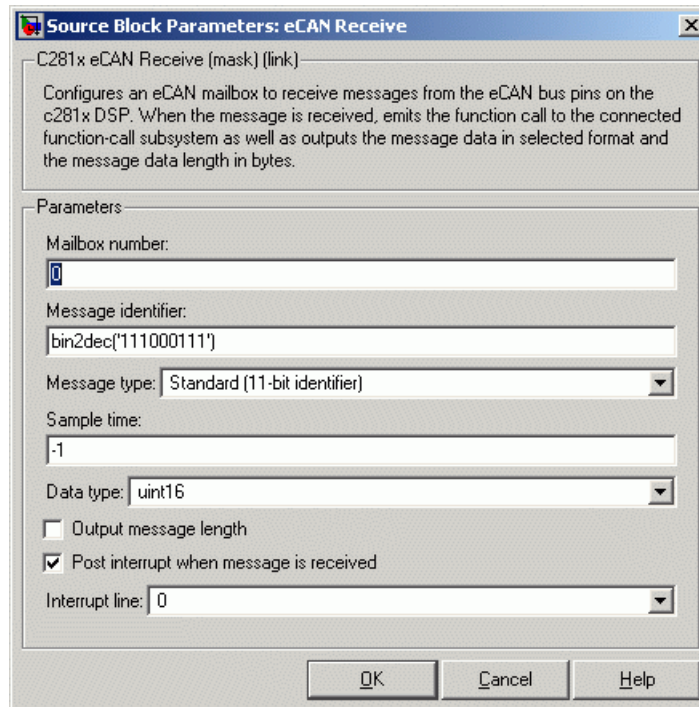
In the preceding figure, the Hardware Interrupt block is set to react on two interrupts. Since only one Hardware Interrupt block is allowed in a model and the output of this block is a vector of length two, you must connect the Hardware Interrupt block to a Demux block to trigger the two function-called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

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

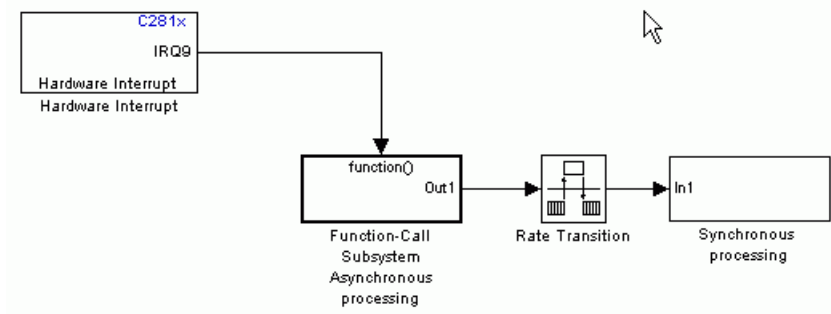
- 1 Place the eCAN Receive block in a function-called subsystem, as shown in the following figure.



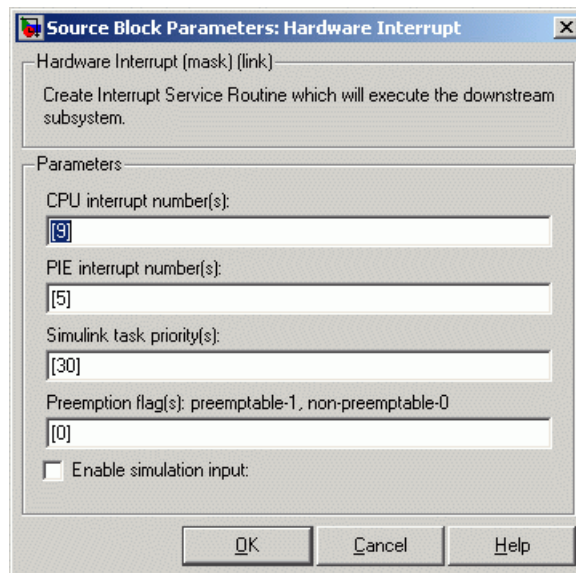
- 2 On the eCAN Receive block dialog, check the box labeled **Post interrupt when message is received**, as shown in the following figure.



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



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



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

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

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

Overview of Creating Models for Targeting

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 1-24 for an example of how to use this library.

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

- Appropriate Target Preferences library block, to set preferences for your target and application
- From the appropriate libraries in the c2000lib block library, to handle input and output functions for your target hardware
- From Real-Time Workshop
- From Simulink Fixed Point
- Discrete time blocks from Simulink
- From any other blockset that meets your needs and operates in the discrete time domain

Online Help

To get general help for using the Embedded Target for the TI TMS320C2000 DSP Platform, use the help feature in MATLAB. At the command prompt, type

```
help tic2000
```

to list the functions and block libraries included in the Embedded Target for the TI TMS320C2000 DSP Platform. Or select **Help > Full Product Family Help** from the menu bar in the MATLAB desktop. When you see the Contents in Help, select **Embedded Target for the TI C2000 DSP**.

Blocks with Restrictions

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

Blocks to Avoid Using in Your Models

The blocks listed in the table below generate code, but they do not work on the target as they do on your desktop—in general, they slow your signal processing application without adding instrumentation value. For this reason, The MathWorks recommends that you *avoid* using certain blocks, such as the Scope block and some source and sink blocks, in SIMULINK models that you use on Embedded Target 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	Platform-Specific I/O	From Wave Device
		From Wave File
		To Wave Device
		To Wave File
	Signal Operations	Triggered Signal From Workspace
	Signal Processing Sinks	Signal To Workspace
		Spectrum Scope
		Triggered to Workspace
	Signal Processing Sources	Signal From Workspace

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 TIC2000 Embedded Targets is Single. Be sure to set this parameter correctly in the Block Parameters dialog box. See the following figure.

The image shows a configuration dialog box for an S-Function Builder. It contains four fields: 'Sample time' with a value of '1', 'Samples per frame' with a value of '1', 'Output data type' with a dropdown menu set to 'Single', and 'Complexity' with a value of 'Real'.

S-Function Builder Blocks

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

Setting Simulation Configuration Parameters

When you drag a Target Preferences block into your model, you are given the option to set basic simulation parameters automatically. (Note that this option does not appear if the Configuration Parameters dialog box is open when you drag the Target Preferences block into the model.)

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

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

- You must specify discrete time by selecting **Fixed-step and discrete** (no continuous states) in the **Solver** pane of the Configuration Parameters dialog box.
- You must also specify the appropriate version of the system target file and template makefile in the **Real-Time Workshop** pane. For the Embedded Target for the TI TMS320C2000 DSP Platform, specify one of the following system target files, or click **Browse** and select from the list of targets.

```
ti_C2000_grt.tlc
ti_C2000_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 the Embedded Target for the TI TMS320C2000 DSP Platform. These correspond to the two system target files mentioned above.

A Generic Real-Time (GRT) target (such as `ti_C2000_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 `ti_C2000_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 the Embedded Target for the TI TMS320C2000 DSP Platform 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 **Build** on the **Real-Time Workshop** pane. Real-Time Workshop automatically generates C code and inserts the I/O device drivers as specified by the hardware blocks in your block diagram, if any. These device drivers are inserted in the generated C code as inlined S-functions. For information about inlining S-functions, refer to your target language compiler documentation. For a complete discussion of S-functions, refer to your documentation about writing S-functions.

Note To build, load, and run code successfully on your target board, MATLAB must be able to locate that board in your system configuration. Make sure that the **Board Name** in your Code Composer Studio setup and the **DSPBoardLabel** in the Target Preferences block in your model match.

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 Target Preferences block, your makefile invokes the TI cross-compiler to build an executable file that is automatically downloaded via the parallel port to your target. After downloading the executable file to the target, the build process runs the file on the board's DSP.

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

F2812 eZdsp and F2808 eZdsp Reset Sequence

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

LF2407 eZdsp Reset Sequence

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

Using the c2000lib Blockset

This section uses an example to demonstrate how to create a Simulink model that uses the Embedded Target for TI C2000 DSP blocks to target your board. The example creates a model that performs PWM duty cycle control via pulse width change. It uses the C2812 ADC block to sample an analog voltage and the C2812 PWM block to generate a pulse waveform. The analog voltage controls the duty cycle of the PWM and you can observe the duty cycle change on the oscilloscope. This model is also provided in the Demos library. Note that the model in the Demos library also includes a model simulation.

Hardware Setup

The following hardware is needed for this example:

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

To connect the hardware:

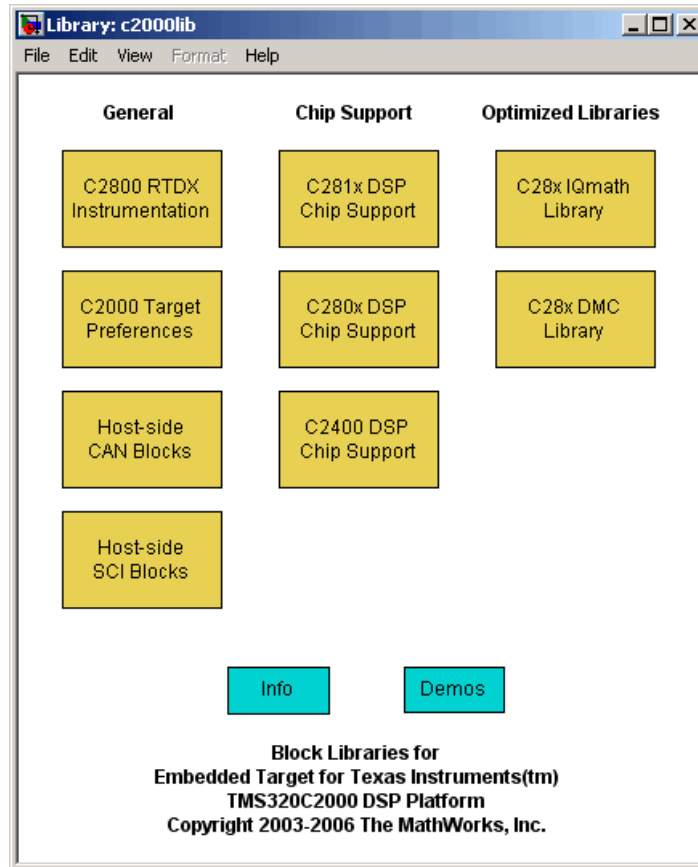
- 1** Connect the function generator output to the ADC input ADCINA0 on the eZdsp F2812.
- 2** Connect the output of PWM1 on the eZdsp F2812 to the analog input of the oscilloscope.
- 3** Connect VREFLO to AGND on the eZdsp F2812. See the section on the Analog Interface in Chapter 2 of the *eZdsp™ F2812 Technical Reference*, available from the Spectrum Digital Web site at <http://c2000.spectrumdigital.com/ezf2812/>

Starting the c2000lib Library

At the MATLAB prompt, type

```
c2000lib
```

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



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

General

- C2800 RTDX Instrumentation (`rtdxBlocks`) — Blocks for adding RTDX communications channels to Simulink models. See the tutorial in the Link for Code Composer Studio Development Tools documentation for an example of using these blocks.

- C2000 Target Preferences (c2000tgtpreflib) — Blocks to specify target preferences and options. You do not connect this block to any other block in your model.
- Host-side CAN Blocks (c2000canlib) — Blocks to configure CAN message blocks and Vector CAN driver blocks

Chip Support

- C281x DSP Chip Support (c281xdspchiplib) — Blocks to configure the codec on the F2812 eZdsp DSK or on C281x-based custom boards
- C280x DSP Chip Support (c280xdspchiplib) — Blocks to configure the codec on the F2808 eZdsp DSK or on C280x-based custom boards
- C2400 DSP Chip Support (c2400dspchiplib) — Blocks to configure the codec on the LF2407 eZdsp DSK or on the LF2407 DSP

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

Other Blocks

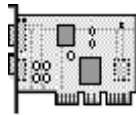
- Info block — Online help
- Demos block — Demos window

For more information on the blocks in each library, refer to their reference pages.

Setting Up the Model

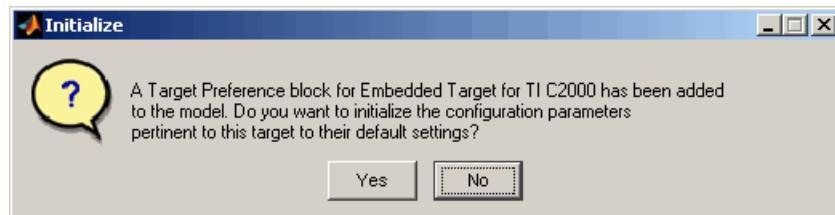
Preliminary tasks for setting up a new model include adding a Target Preferences block, setting or verifying Target Preferences, and setting the simulation parameters.

- 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

The following query asks if you want preferences to be set automatically.



- 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:

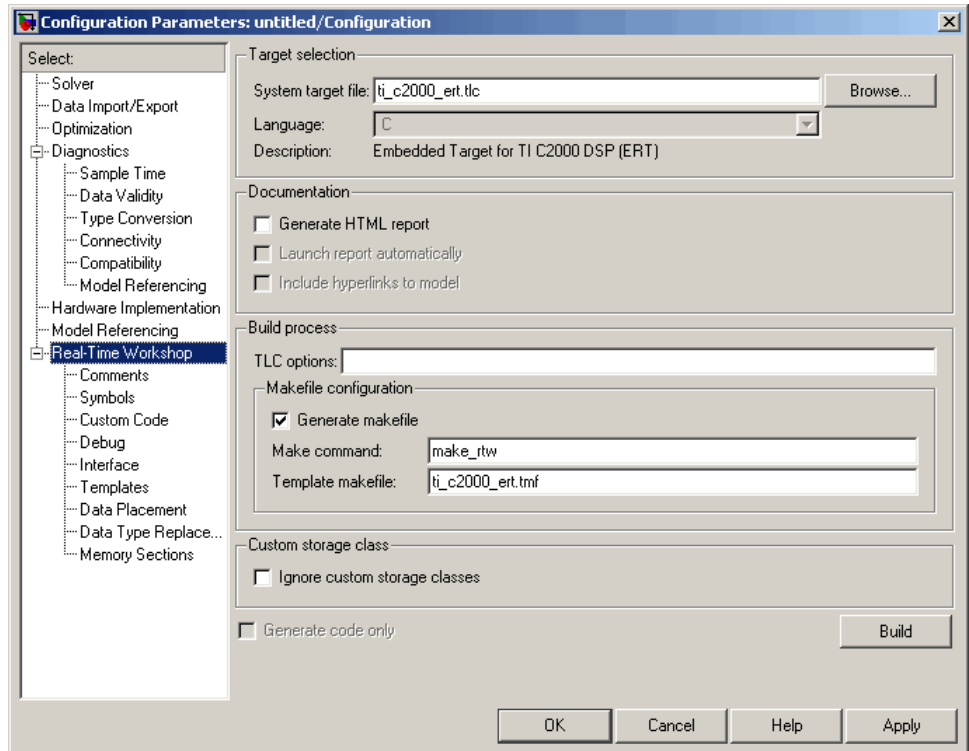
Pane	Field	Setting
Solver	Stop time	inf
Solver	Type	Fixed-step
Data Import/Export	Save to workspace - Time	Off (cleared)
Data Import/Export	Save to workspace - Output	Off (cleared)
Hardware Implementation	Device type	TI C2000

Pane	Field	Setting
Real-Time Workshop	Target configuration - System target file	ti_c2000_grt.tlc
Real-Time Workshop	Target configuration - Template makefile	ti_c2000_grt.tmf

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

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

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



- **System target file.** Clicking **Browse** opens the **System target file browser** where you select `ti_c2000_grt.tlc` or `ti_c2000_ert.tlc`. When you select your target configuration, Real-Time Workshop chooses the appropriate system target file, template makefile, and make command. You can also enter the target configuration filename, and Real-Time Workshop will fill in the **Template makefile** and **Make command** selections.
- **Make command.** When you generate code from your digital signal processing application, use the standard command `make_rtw`. Enter `make_rtw` for the **Make command**.
- **Template makefile.** When you select the **System target file**, Real-Time Workshop automatically selects the appropriate template makefile: `ti_c2000_grt.tmf` or `ti_c2000_ert.tmf`.

- **Generate code only.** This option does not apply to targeting with the Embedded Target for TI C2000 DSP. To generate source code without building and executing the code on your target, open the Target Preferences block in your model and select **Generate code only** as the **BuildAction (BuildOptions > RunTimeOptions > BuildAction)**.

For all other Real-Time Workshop options, leave the default values for this example.

- 7 Set the Target Preferences by double-clicking the F2812 eZdsp block and adjust these parameters. For descriptions of these fields, see the F2812 eZdsp reference page.

Build Options

Subfield	Field	Setting
Compiler Options	CompilerVerbosity	Verbose
	KeepASMFiles	False
	OptimizationLevel	Function(-o2)
	SymbolicDebugging	Yes
Linker Options	CreateMAPFile	True
	KeepOBJFiles	True
	LinkerCMDFile	Full_memory_map
RunTime Options	BuildAction	Build_and_execute
	OverrunAction	Continue

CCSLink Options

Field	Setting
CCSHandleName	CCS_Obj
ExportCCSHandle	True

CodeGeneration Options

Subfield	Field	Setting
Scheduler	Algorithm	Preemptive_priority_based
	Timer	CPU_timer0

DSPBoard Options

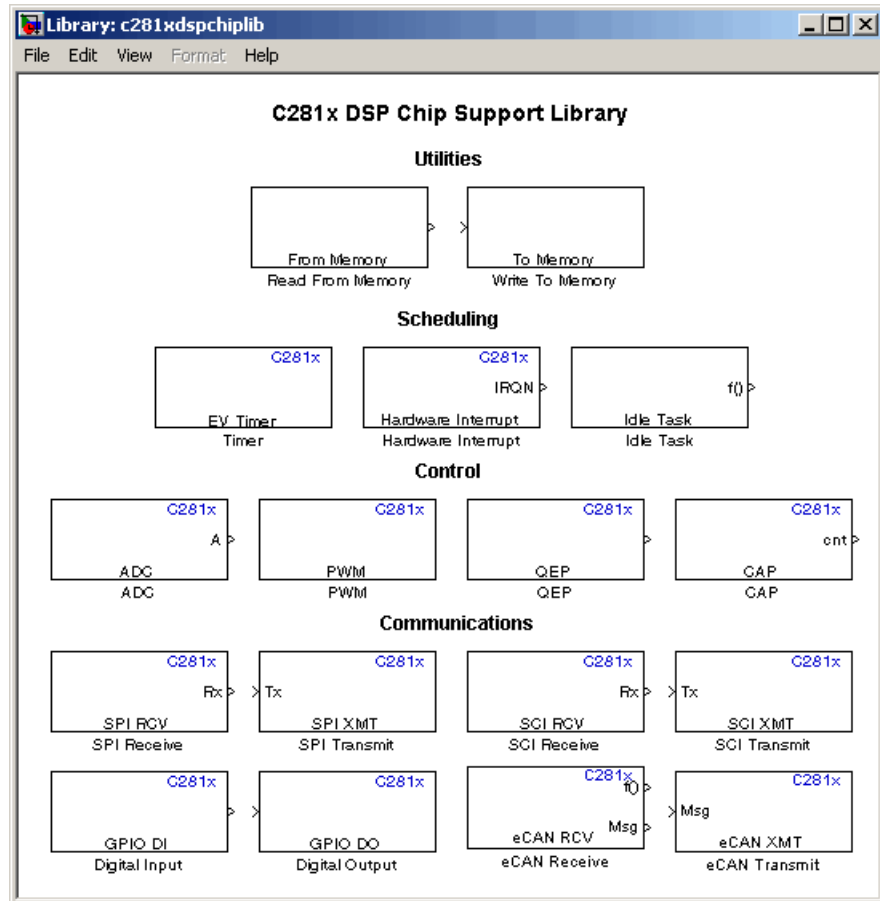
Subfield	Field	Setting
DSP Board Label	DSPBoardLabel	F2812 PP Emulator (see Note below)
DSP Chip	DSPChipLabel	TI TMS320C2812
eCAN	BaudRatePrescaler	10
	EnhancedCANMode	True
	SAM	Sample_one_time
	SBJ	Only_falling_edges
	SJW	2
	SelfTestMode	False
	TSEG1	8
	TSEG2	6

Note If the board label in your Code Composer Studio setup differs from the default DSP Board Label shown in the Target Preferences block, you can change the default setting. This would ensure that whenever you drag a Target Preferences block into a new model, the DSP Board Label of your model will match the label in your Code Composer Studio setup.

Open the C2000 Target Preferences library. Double-click the appropriate Target Preferences block. Click **DSP Board** and change the text in the DSP Board Label right column to the desired string. Click **OK** to close the Target Preferences block and then close the library to save your change.

Adding Blocks to the Model

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



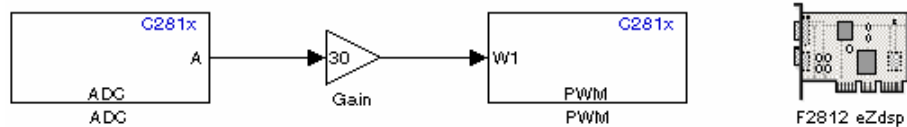
- 2 Drag the C281x ADC block into your model. Double-click the ADC block in the model and set **Sample time** to 64/80000. Use the default values for all other fields. Refer to the C281x ADC reference page for information on these fields.
- 3 Drag the C281x PWM block into your model. Double-click the PWM block in the model and set the following parameters. Refer to the C281x PWM reference page for information on these fields.

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** Type 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 Data Types	Output data type mode	Specify via dialog
	Output data type	uint(16)
	Round integer calculations toward	Floor
Parameter Data Types	Parameter data type mode	Same as input

- 5 Connect the ADC block to the Gain block and the Gain block to the PWM block as shown:



Generating Code from the Model

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

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

The code building process consists of these steps:

- 1 Real-Time Workshop invokes the function `make_rtw` to start the Real-Time Workshop build procedure for a block diagram. `make_rtw` invokes the

Target Language Compiler to generate the code and then invokes the language-specific make procedure.

- 2 `gmake` builds the file `modelName.out`. Depending on the build options you select in the Simulation Parameters dialog box, `gmake` can initiate the sequence that downloads and executes the model on your TI target board.

Creating Code Composer Studio Projects Without Loading

To create projects in CCS without loading files to your target:

- 1 In the **Real-Time Workshop** pane in the Simulation Parameters dialog box, select `ti_c2000.tlc` as the system target file.
- 2 Select `Create_CCS_Project` for the **BuildAction** in the Target Preferences block. Note that the `Build` and `Build_and_execute` options create CCS projects as well.
- 3 Set the other Target Preferences options, including those for **CCSLink**. On the **Real-Time Workshop** pane of the Simulation Parameters dialog box, click **Build** to build your new CCS project.

Real-Time Workshop and the Embedded Target for TI C2000 DSP generate all the files for your project in CCS and create a new project in the IDE. Your new project is named for the model you built.

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

Configuring Timing Parameters for CAN Blocks

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

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

Setting Timing Parameters (p. 2-3)

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

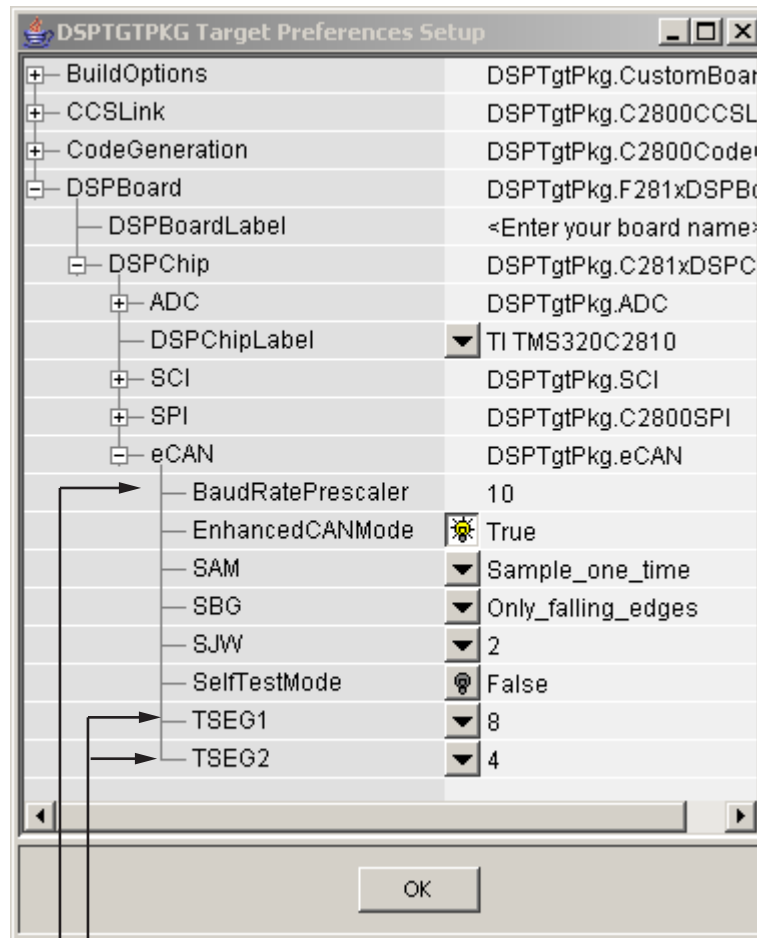
Blocks Where the Bit Rate Cannot Be Set Directly

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

- C281x eCAN Receive
- C281x eCAN Transmit
- C280x eCAN Receive
- C280x eCAN Transmit
- C24x CAN Receive
- C24x CAN Transmit

Setting Timing Parameters

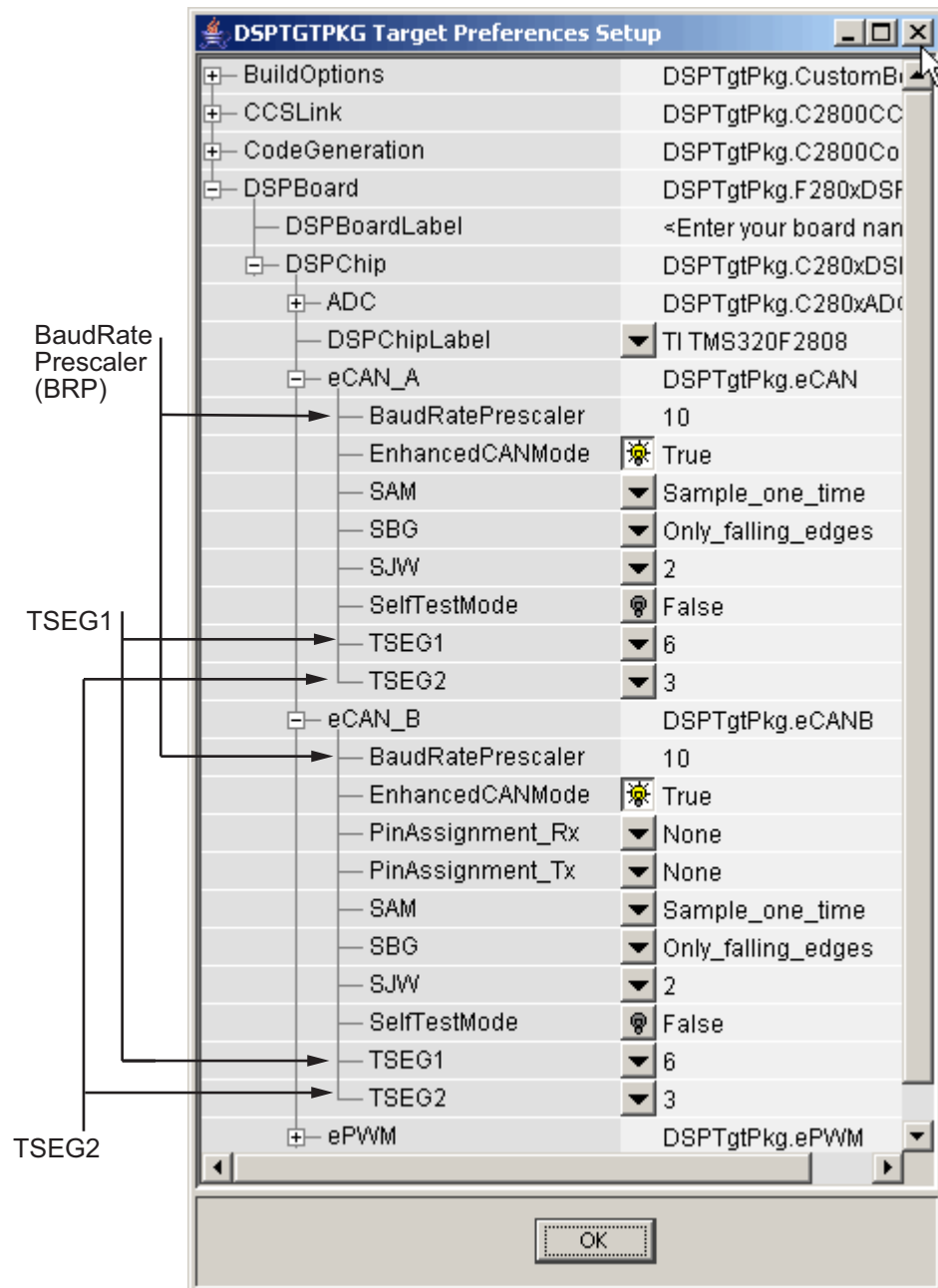
In “Blocks Where the Bit Rate Cannot Be Set Directly” you must use the following parameters: **TSEG1**, **TSEG2**, and **BaudRatePrescaler (BRP)** to set the required bit rate. These parameters are configured from the **Target Preference Setup** dialog box used for your specific model. For example, for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



TSEG1 and TSEG2

BaudRatePrescaler (BRP)

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



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

Equations for Bit Rate Calculation

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

- 1 Review the known entities:

Bit Rate

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

SYSCLOCKOUT

This is the CAN module system clock frequency.

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

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

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

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

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

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

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

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

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

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

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

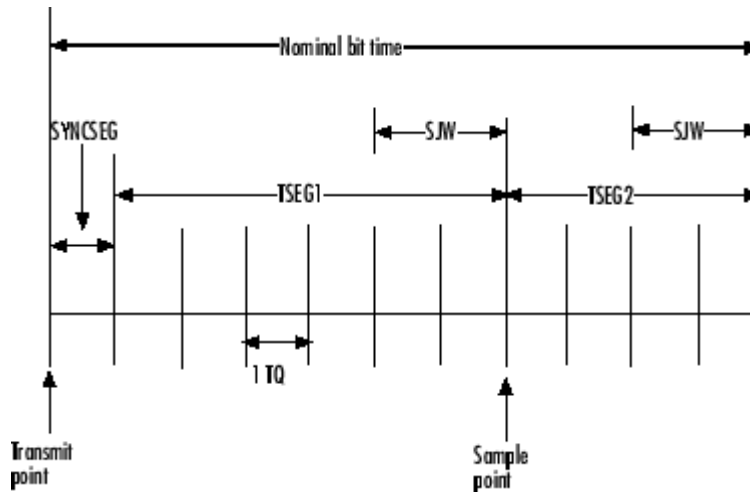
$$\text{IPT} \leq \textbf{TSEG2} \leq 8 \text{ TQ}$$

$$1 TQ \leq S\text{JW} \leq \min(4 TQ, T\text{SEG}2)$$

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

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

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



CAN Bit Timing Examples

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

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

$$\text{BitTime} = \text{SYSCLKOUT} / (\text{BRP} * \text{Bitrate})$$

$$\text{BitTime} = 150 / (10 * 1) = 15TQ$$

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

$$\begin{aligned} \text{BitTime} &= \text{TSEG1} + \text{TSEG2} + 1 \\ 15\text{TQ} &= 8\text{TQ} + 6\text{TQ} + 1 \end{aligned}$$

- 3** Finally, check the selected values against the rules:

$$\begin{aligned} \text{IPT} &= 3/\text{BRP} = 3/10 = .3 \\ \text{IPT} &\leq \text{TSEG1} \leq 16 \text{ TQ True! } .3 \leq 8\text{TQ} \leq 16\text{TQ} \\ \text{IPT} &\leq \text{TSEG2} \leq 8 \text{ TQ True! } .3 \leq 6\text{TQ} \leq 8\text{TQ} \\ 1\text{TQ} &\leq \text{SJW} \leq \min(4\text{TQ}, \text{TSEG2}) \text{ which means that SJW can be set to either 2, 3, or 4} \end{aligned}$$

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

The following table provides common timing parameter settings for three typical values of Bit Rate and SYSCLKOUT = 40MHz. This clock frequency is the maximum for the C24x blocks.

Bit Rate	TSEG1	TSEG2	Bit Time	BRP	SJW
.5 Mbit/s	4	3	8	10	2
1 Mbit/s	5	4	10	4	2
2 Mbit/s	6	3	10	2	2

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

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 3 typical values of Bit Rate and SYSCLKOUT = 100MHz. This clock frequency is the maximum for the C280x 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

Configuring Acquisition Window Width for ADC Blocks

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

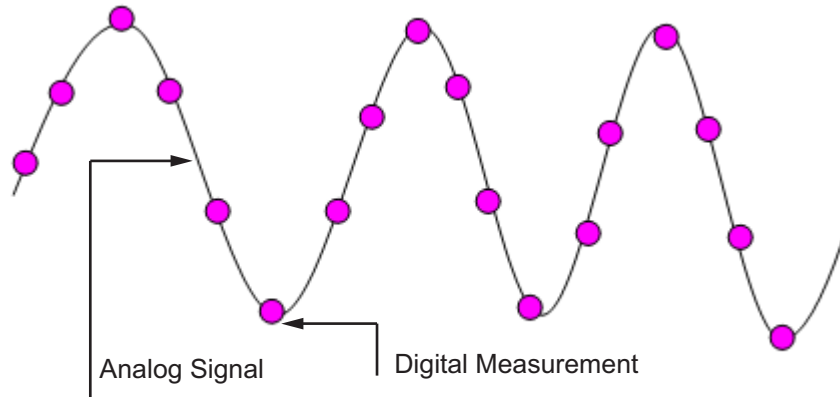
Explains the concept of the acquisition window and how it affects data validity

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

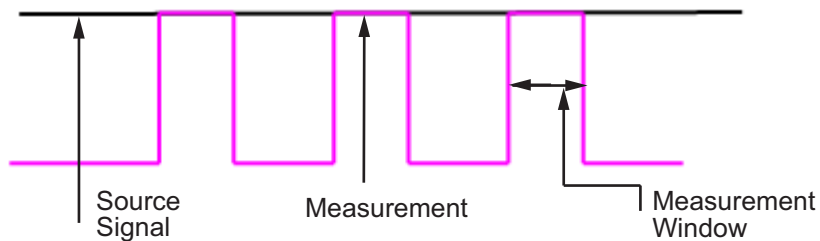
Describes how to set ADC parameters to obtain the proper acquisition window width

What Is an Acquisition Window?

The purpose of ADC blocks is to 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 rather in a series of discrete measurements, close enough together to approximate the source signal with the required accuracy, as shown in the following figure:

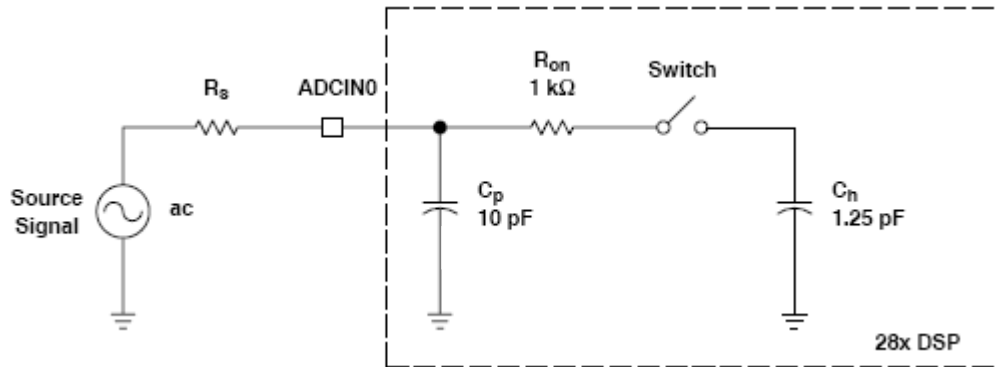


The digital measurement itself is not an instantaneous process, but is rather a measurement window, where the signal is acquired and then measured, as shown below:

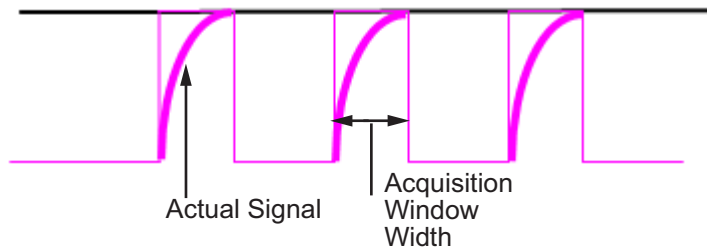


In an ideal environment, as soon as the measurement window is opened, the actual signal coming in would be measured perfectly. However, in actuality, the signal does not reach its full magnitude immediately. In the real world

environment, the measurement 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. However, because of the nature of the circuit, characterized by its capacitance, the signal received is not in a form of a step function as shown by the ideal measurement, but rather 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:



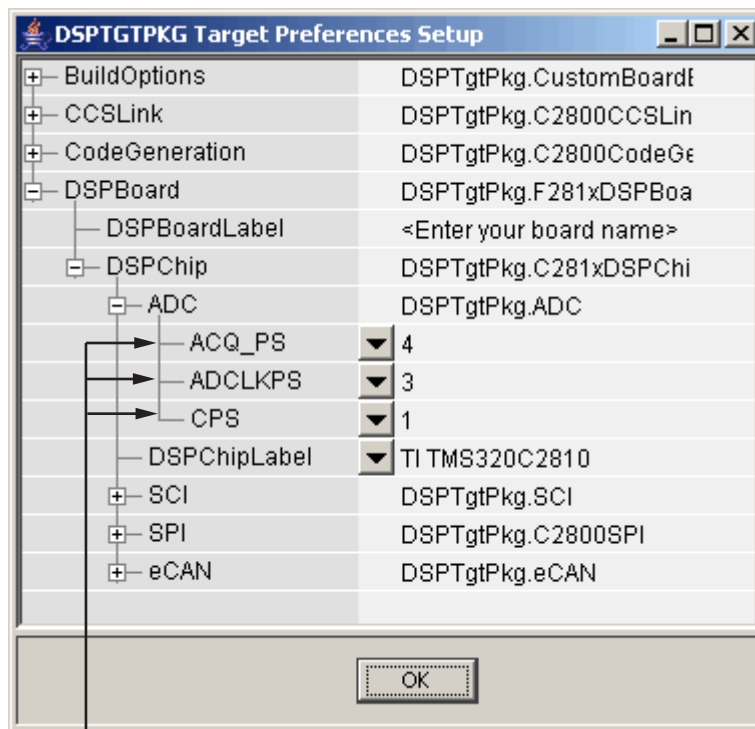
Because the signal acquisition is not instantaneous, it is very important to set a wide enough acquisition window to allow the signal to ramp up to full strength before the measurement is taken. If the window is too narrow, the measurement is taken before the signal has reached its full magnitude, resulting in erroneous data. If the window is too wide, there is a chance of the source signal itself changing, and the sampling being too infrequent to reflect the actual value, resulting in erroneous data. Therefore, setting the

correct width of the acquisition window is crucial in obtaining correct data. You are responsible for calculating 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

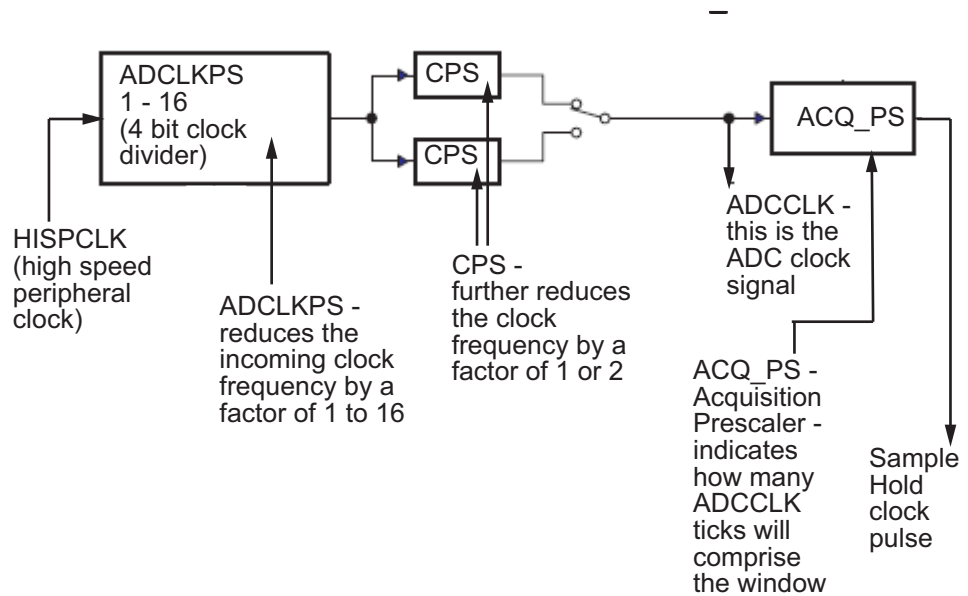
The ADC parameters can be set from the **Target Preferences Setup** dialog of the Custom C280x Board, or the Custom C281x Board, or the F2808 eZdsp. These parameters are:

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



The three parameters affecting Acquisition Window Width: ACQ_PS, ADCLKPS, and CPS

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 nano seconds.

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 nano seconds.

Creating Stand-Alone Applications by Saving Code into Flash Memory

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

Explains the need and use for storing code in Flash memory

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

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

Running Code from Flash Memory (p. 4-5)

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

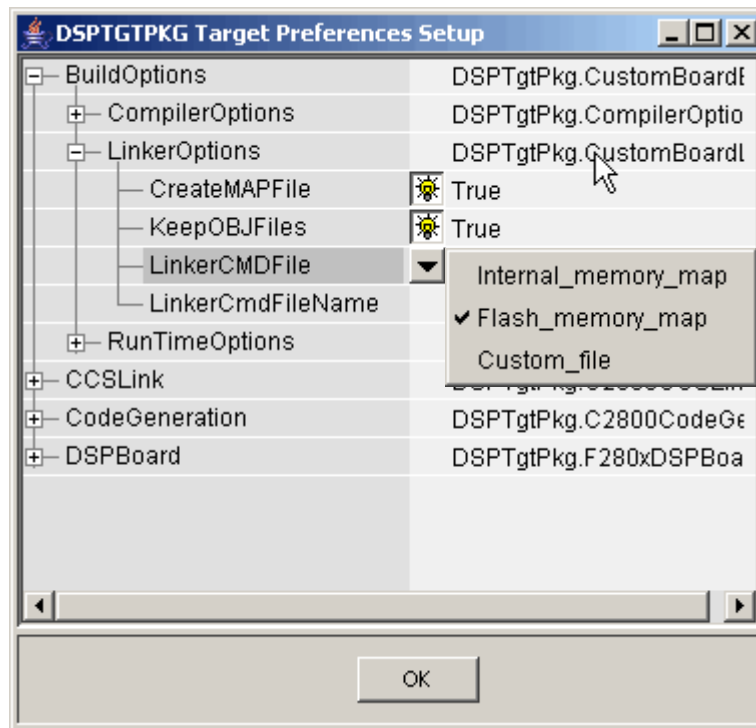
The Need for Stand-Alone Applications

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

Generating Code for Flash Memory

To store generated code in the internal Flash memory of the C28xx DSPs specific parameters need to be set. You also need a Flash Programmer. The following process guides you through the necessary steps:

- 1 On the Target Preferences block, open the **Target Preferences Setup** dialog box. From **BuildOptions > LinkerOptions > LinkerCMDFile** and select **Flash_memory_map**, as shown in the following figure:



Selecting this option points the code generation to the on-chip Flash memory.

- 2 Programming the on-chip Flash for TI C28xx DSPs requires a Flash Programmer. The two most commonly used options are the TI's Flash Programmer plug-in to CCS or the Spectrum Digital™ SDFlash. Refer

to the specific vendor's documentation for more information, and then download and install a Flash Programmer of your choice.

- 3** Build and generate code in CCS. Then, launch the Flash Programmer to erase, program, and verify the Flash. Your chip now contains the code in its Flash memory.

Running Code from Flash Memory

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

Using the IQmath Library

About the IQmath Library (p. 5-2)

Introduces the IQmath Library

Fixed-Point Numbers (p. 5-4)

Representation of fixed-point numbers in the IQmath Library

Building Models (p. 5-9)

Issues to consider when you build models with the IQmath Library

About the IQmath Library

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

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

The IQmath Library blocks generally input and output fixed-point data types and use numbers in Q format. The C28x IQmath Library block reference pages discuss the data types accepted and produced by each block in the library. For more information on fixed-point numbers and Q format, see

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

You can use these blocks with some core Simulink blocks and Simulink Fixed Point blocks to run simulations in Simulink models before generating code. Once you develop your model, you can invoke Real-Time Workshop to generate equivalent code that is optimized to run on a TI C28x DSP. During code generation, a call is made to the IQmath Library for each IQmath Library block in your model to create target-optimized code. To learn more about creating models that include both IQmath Library blocks and blocks from other blocksets, refer to “Building Models” on page 5-9.

Common Characteristics

The following characteristics are common to all IQmath Library blocks:

- Sample times are inherited from driving blocks.

- Blocks are single rate.
- Parameters are not tunable.
- All blocks support discrete sample times.

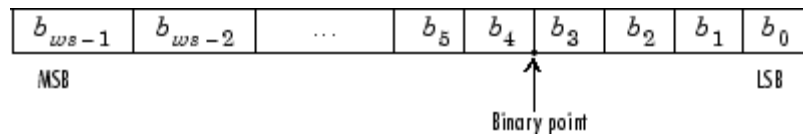
To learn more about characteristics particular to each block in the library, see “C28x IQmath (tiiqmathlib)” on page 6-11 for links to the individual block reference pages.

Fixed-Point Numbers

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

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

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



where

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

Signed Fixed-Point Numbers

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

- Sign/magnitude
- One's complement

- Two's complement

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

Negation using signed two's complement representation consists of a bit inversion (translation into one's complement) 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, which use 24 bits, the resolution of the corresponding 32-bit number cannot be attained. The 24-bit number approximates its value by truncating the lower end. For example:

```
32-bit integer 11110000 11001100 10101010 00001111
Single-precision float +1.1110000 11001100 10101010 x 231
Corresponding value 11110000 11001100 10101010 00000000
```

Example – Q.15

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

Q0.15

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

Q.15

In Simulink Fixed Point, this data type is expressed as

sfrac16

or

sfix16_En15

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

[16 15]

Example – Q1.30

Multiplying two Q₁₅ numbers yields a product that is a signed 32-bit data type with $n = 30$ bits to the right of the binary point. One bit is the designated sign bit, thereby forcing m to be 1:

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

Therefore, this number is expressed as

Q1.30

In Simulink Fixed Point, this data type is expressed as

`sfixed32_En30`

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

[32 30]

Example – Q-2.17

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

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

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

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

Therefore, this number is expressed as

Q-2.17

In Simulink Fixed Point, this data type is expressed as

`sfixed16_En17`

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

[16 17]

Example – Q17.-2

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

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

Therefore, this number is expressed as

Q17. -2

In Simulink Fixed Point, this data type is expressed as

`sfixed16_E2`

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

[16 -2]

Building Models

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

Converting Data Types

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

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

Some Signal Processing Blockset blocks and core Simulink blocks also accept fixed-point data types. Make the appropriate settings in these blocks' parameters when you connect them to an IQmath Library block.

Using Sources and Sinks

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

Choosing Blocks to Optimize Code

In some cases, blocks that perform similar functions appear in more than one blockset. For example, both the IQmath Library and Simulink Fixed Point have a Multiply block. When you are building a model to run on C2000 DSP, choosing the block from the IQmath Library always yields better

optimized code. You can use a similar block from another library if it gives you functionality that the IQmath Library block does not support, but you will generate code that is less optimized.

Blocks — By Category

C2000 Target Preferences (c2000tgtpreflib) (p. 6-2)	Target preference blocks for C2000 boards
Host-Side CAN Blocks (c2000canlib) (p. 6-3)	Host-Side CAN blocks
Host-Side SCI Blocks (c2000scilib) (p. 6-4)	Host-Side SCI blocks
C2000 RTDX Instrumentation (rtdxBlocks) (p. 6-5)	RTDX blocks for C2000 boards
C2400 DSP Chip Support (c2400dspchilib) (p. 6-6)	Blocks that support C24x boards
C280x DSP Chip Support (c280xdspchilib) (p. 6-7)	Blocks that support C280x boards
C281x DSP Chip Support (c281xdspchilib) (p. 6-8)	Blocks that support C281x boards
C28x Digital Motor Control (c28xdmclib) (p. 6-10)	Blocks that represent the functionality of the TI C28x DMC Library
C28x IQmath (tiiqmathlib) (p. 6-11)	Blocks that represent the functionality of the TI IQmath Library

C2000 Target Preferences (c2000tgtplib)

Custom C280x Board

Target preferences for custom C280x board

Custom C281x Board

Target preferences for custom C281x board

F2808 eZdsp

F2808 eZdsp DSK target preferences

F2812 eZdsp

F2812 eZdsp DSK target preferences

LF2407 eZdsp

LF2407 eZdsp DSK target preferences

Host-Side CAN Blocks (c2000canlib)

See the “CAN Blockset Reference” for information on these blocks.

Host-Side SCI Blocks (c2000scilib)

SCI Receive

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

SCI Setup

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

SCI Transmit

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

C2000 RTDX Instrumentation (rtdxBlocks)

From RTDX

Add RTDX input channel

To RTDX

Add RTDX output channel

C2400 DSP Chip Support (c2400dspchiplib)

C24x ADC	Analog-to-digital converter (ADC)
C24x CAN Receive	Enhanced Control Area Network (CAN) receive mailbox
C24x CAN Transmit	Enhanced Control Area Network (CAN) transmit mailbox
C24x CAP	Receive and log capture input pin transitions
C24x GPIO Digital Input	General-purpose I/O pins for digital input
C24x GPIO Digital Output	General-purpose I/O pins for digital output
C24x PWM	Pulse width modulators (PWMs)
C24x QEP	Quadrature encoder pulse circuit
C24x SCI Receive	Receive data on target via serial communications interface (SCI) from host
C24x SCI Transmit	Transmit data on target via serial communications interface (SCI) from host
C24x SPI Receive	Receive data via serial peripheral interface (SPI) on target
C24x SPI Transmit	Transmit data via serial peripheral interface (SPI) to host
From Memory	Retrieve data from target memory
To Memory	Write data to target memory

C280x DSP Chip Support (c280xdspchiplib)

C280x ADC	Analog-to-digital converter (ADC)
C280x eCAN Receive	Enhanced Control Area Network receive mailbox
C280x eCAN Transmit	Enhanced Control Area Network transmit mailbox
C280x ePWM	Configure C280x Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms
C280x eQEP	Quadrature encoder pulse circuit
C280x Hardware Interrupt	Create Interrupt Service Routine to handle hardware interrupts
C280x SCI Receive	Receive data on target via serial communications interface (SCI) from host
C280x SCI Transmit	Transmit data from target via serial communications interface (SCI) to host
C280x SPI Receive	Receive data via the serial peripheral interface (SPI) on the target
C280x SPI Transmit	Transmit data via the serial peripheral interface (SPI) to the host
From Memory	Retrieve data from target memory
Idle Task	Create free-running task that executes downstream subsystem
To Memory	Write data to target memory

C281x DSP Chip Support (c281xdspchilib)

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

Idle Task

Create free-running task that
executes downstream subsystem

To Memory

Write data to target memory

C28x Digital Motor Control (c28xdmclib)

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

C28x IQmath (tiqmathlib)

Absolute IQN	Absolute value
Arctangent IQN	Four-quadrant arc tangent
Division IQN	Divide two 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 two IQ numbers with same Q format
IQN1 to IQN2	Convert IQ number to different Q format
IQN1 x IQN2	Multiply two IQ numbers with different Q formats
Magnitude IQN	Magnitude of two orthogonal IQ numbers
Saturate IQN	Saturate IQ number
Square Root IQN	Square root or inverse square root of IQ number
Trig Fcn IQN	Sine, cosine, or arc tangent of IQ number

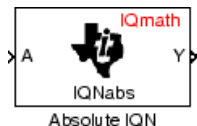
Blocks — Alphabetical List

Absolute IQN

Purpose Absolute value

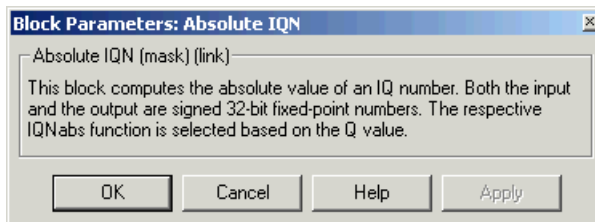
Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

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



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

Dialog Box



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

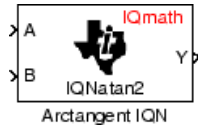
Purpose

Four-quadrant arc tangent

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

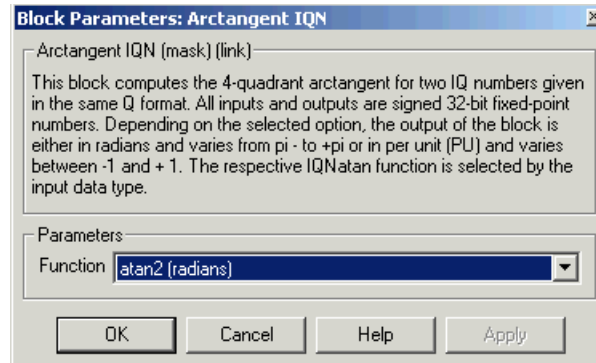
Description



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

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

Dialog Box



Function

Type of arc tangent to calculate:

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

Arctangent IQN

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

See Also

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

Purpose

Analog-to-digital converter (ADC)

Library

c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description

The C24x ADC block configures the C24x 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 C24x ADC trigger mode depends on the internal setting of the source start-of-conversion (SOC) signal. The ADC is usually triggered by software at the sample time intervals specified in the ADC block — this is unsynchronized mode. For more information on configuring the specific parameters for this mode, see “Configuring Acquisition Window Width for ADC Blocks”.

In synchronized mode, the Event Manager (EV) 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. **ADC Start Event** is set in the C24x 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

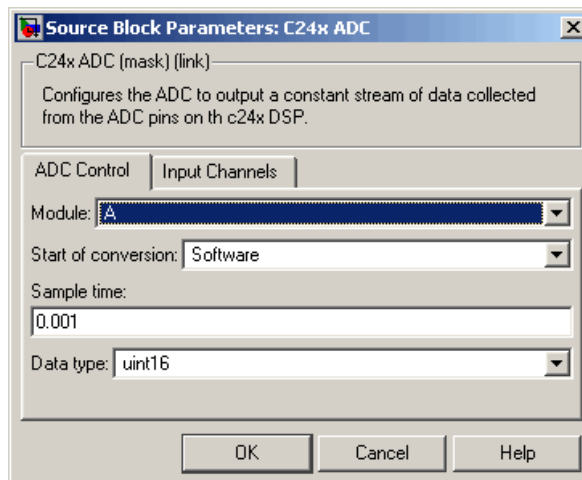
The output of the C24x ADC is a vector of uint16 values. The output values are in the range 0 to 1023 because the C24x ADC is a 10-bit converter.

Modes

The C24x ADC block supports ADC sequential operation in dual and cascaded modes. In dual mode, either module A or module B can be used for the ADC block, and two ADC blocks are allowed in the model. In cascaded mode, both module A and module B are used for a single ADC block.

Dialog Box

ADC Control pane



Module

Specifies which DSP module to use:

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

Start of conversion

Type of signal that triggers sequential conversions to begin:

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

Sample time

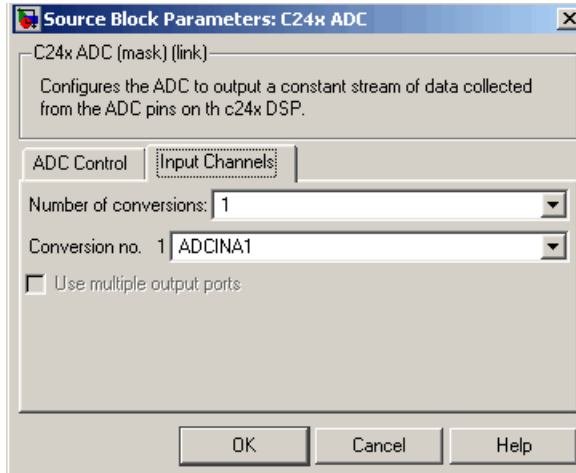
Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. See “Scheduling and Timing” on page 1-11 for more information on timing.

To set different sample times for different groups of ADC channels, you must add separate C24x 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.

Input Channels pane



Number of conversions

Number of analog-to-digital conversions to perform in a single sampling sequence.

Conversion no.

Specific ADC channel to associate with each conversion number.

In simultaneous mode, a pair of ADC channels is associated with each conversion. 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.

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.

Note The Discrete Filter block in Simulink accepts only mono input. To connect a C24x ADC block to this block, you must output a single channel or connect only one of the ADC's output ports to a Discrete Filter block.

See Also C24x PWM

C24x CAN Receive

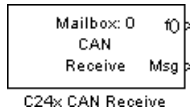
Purpose

Enhanced Control Area Network (CAN) receive mailbox

Library

c2400dspchip1ib in Embedded Target for TI C2000 DSP

Description



The C24x CAN Receive block generates source code for receiving CAN messages through a CAN mailbox. The CAN module on the DSP chip provides serial communication capability and has six mailboxes — two for receive, two for transmit, and two configurable for receive or transmit. The C24x chip supports CAN data frames in standard or extended format.

The C24x CAN 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. (See **Data type** below for information.)
- The third output port is optional and appears only if **Output message length** is selected.

Dialog Box

Block Parameters: C24x CAN Receive

C24x CAN Receive (mask) (link)

Configures a CAN mailbox to receive messages from the CAN bus pins on the c24x DSP. When the message is received, emits the function call to the connected function-call subsystem as well as outputs the message data in selected format and the message data length in bytes.

Parameters

Mailbox number:

Message identifier:

Message type:

Sample time:

Data type:

Output message length

OK Cancel Help Apply

Mailbox number

Unique number from 0 to 5 that refers to a mailbox area in RAM. Mailboxes 0 and 1 are receive mailboxes, 2 and 3 are configurable for receive or transmit, and 4 and 5 are transmit mailboxes. In standard data frame 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.

C24x CAN Receive

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.

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 rightaligned in the output. Only uint16 (vector length = 4 elements) or uint32 (vector length = 8 elements) data are allowed. The data are unpacked as follows using the data buffer, which is 8 bytes.

For uint16 data,

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

For uint32 data,

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

For example, if the received message has two bytes

```
data_buffer[0] = 0x21
```

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

then 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.

References

Detailed information on the CAN module is in the *TMS320LF/LC240xA DSP Controller Reference Guide — System and Peripherals*, Literature Number SPRU357B, available at the Texas Instruments Web site.

See Also

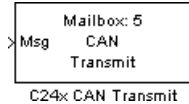
C24x CAN Transmit

C24x CAN Transmit

Purpose Enhanced Control Area Network (CAN) transmit mailbox

Library c2400dspchip1ib in Embedded Target for TI C2000 DSP

Description



The C24x CAN Transmit block generates source code for transmitting CAN messages through a CAN mailbox. The CAN module on the DSP chip provides serial communication capability and has six mailboxes — two for receive, two for transmit, and two configurable for receive or transmit. The C24x chip supports CAN data frames in standard or extended format.

Data Vectors

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

For input of type `uint32`,

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

the data buffer is

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

For input of type `uint16`,

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

the data buffer is


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

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

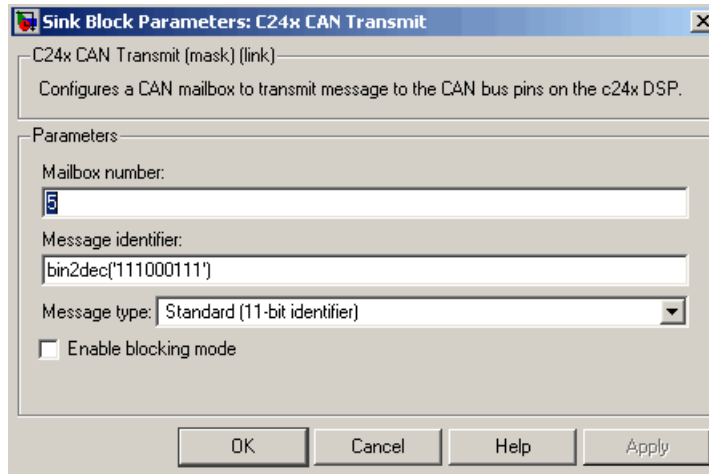
```
inputdata [0] = 0x1234
inputdata [1] = 0x5678
```

the data buffer is

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

C24x CAN Transmit

Dialog Box



Mailbox number

Unique number from 0 to 5 that refers to a mailbox area in RAM. Mailboxes 0 and 1 are receive mailboxes, 2 and 3 are configurable for receive or transmit, and 4 and 5 are transmit mailboxes. In standard data frame mode, the mailbox number determines priority.

Message identifier

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

Message type

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

Enable blocking mode

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

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

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

References

Detailed information on the CAN module is in the *TMS320LF/LC240xA DSP Controller Reference Guide — System and Peripherals*, Literature Number SPRU357B, available at the Texas Instruments Web site.

See Also

C24x CAN Receive

C24x CAP

Purpose

Receive and log capture input pin transitions

Library

c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description



The C24x CAP block sets parameters for the capture units (CAPs) of the Event Manager (EV) module. The capture units log transitions detected on the capture unit pins by recording the times of these transitions into a two-level deep FIFO stack. The capture unit pins can be set to detect rising edge, falling edge, either type of transition, or no transition.

The C24x chip has six capture units — three associated with each EV module. Capture units 1, 2, and 3 are associated with EVA and capture units 4, 5, and 6 are associated with EVB. Each capture unit is associated with a capture input pin.

Note You can have up to two C24x CAP blocks in any one model — one block for each EV module.

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

Outputs

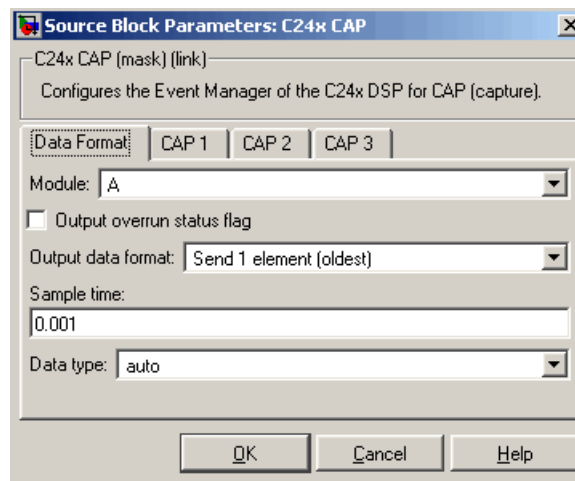
This block has up to two outputs: a cnt (count) output and an optional, FIFO status flag output. The cnt output increments each time a transition of the selected type occurs. The status flag outputs are

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

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

Dialog Box

Data Format pane



Module

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.

Output data format

The type of data to output:

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

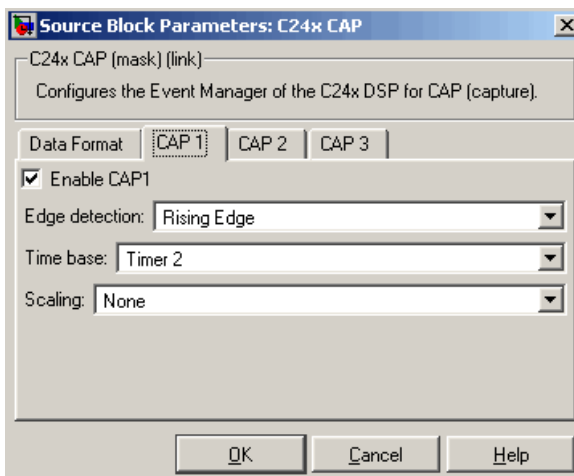
Sample time

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

Data type

Data type of the output data. Available options are auto, double, single, int8, uint8, int16, uint16, int32, uint32, and boolean. Selecting auto defaults to double.

CAP# pane



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

- **CAP1** controls CAP 1 or CAP 4, for EV module A or B, respectively.
- **CAP2** controls CAP 2 or CAP 5, for EV module A or B, respectively.
- **CAP3** controls CAP 3 or CAP 6, for EV module A or B, respectively.

Enable CAP#

Select to use the specified capture unit pin.

Edge detection

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

Time base

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

Note CAP 1 and CAP 2 must use the same GP timer. CAP 4 and CAP 5 must use the same GP timer.

Scaling

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

Scaling	Resulting Rate (μ s)
none	0.025
1/2	0.05
1/4	0.1
1/8	0.2
1/16	0.4
1/32	0.8
1/64	1.6
1/128	3.2

Note The above rates assume a 40-MHz input clock.

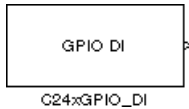
Purpose

General-purpose I/O pins for digital input

Library

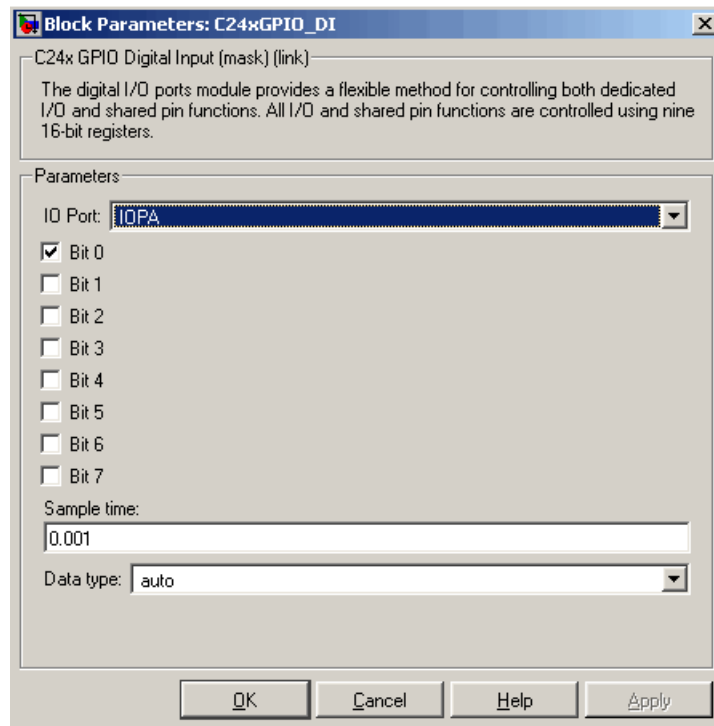
c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description



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

Dialog Box



IO Port

Select the input/output port to use: IOPA, IOPB, IOPC, IOPD, IOPE, or IOPF and select the I/O port bits to enable for digital input. If

C24x GPIO Digital Input

you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Note that multiple C24x GPIO DI blocks cannot share the same I/O port. Only one bit is available for IOPD.

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.

IO MUX Output Control Register A

Bit	Peripheral Name	GPIO Name
3	QEP1/CAP1	IOPA3
4	QEP2/CAP2	IOPA4
5	CAP3	IOPA5
6	PWM1	IOPA6
7	PWM2	IOPA7
8	PWM3	IOPB0
9	PWM4	IOPB1
10	PWM5	IOPB2
11	PWM6	IOPB3

IO MUX Output Control Register C

Bit	Peripheral Name	GPIO Name
1	PWM7	IOPE1
2	PWM8	IOPE2
3	PWM9	IOPE3
4	PWM10	IOPE4
5	PWM11	IOPE5
6	PWM12	IOPE6
7	QEP3/CAP4	IOPE7
8	QEP4/CAP5	IOPF0
9	CAP6	IOPF1

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.

See Also

C24x GPIO Digital Output

C24x GPIO Digital Output

Purpose General-purpose I/O pins for digital output

Library c2400dspchip1ib in Embedded Target for TI C2000 DSP

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 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.

IO MUX Output Control Register A

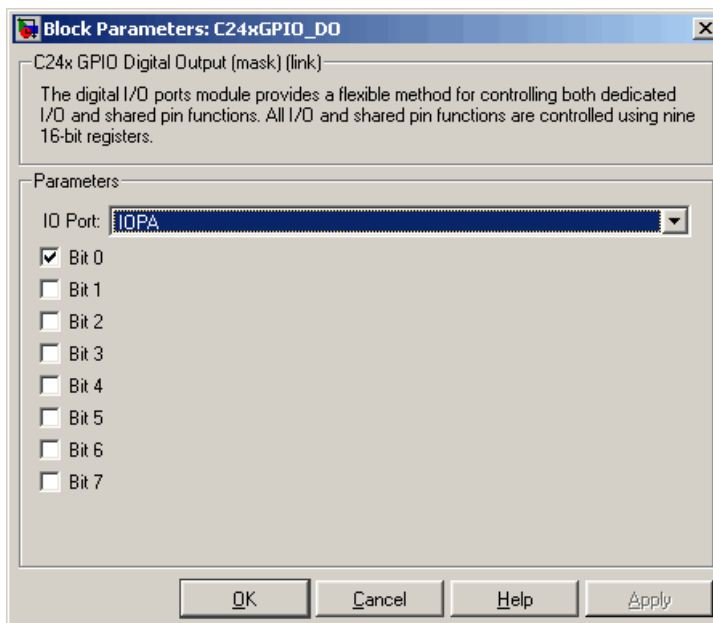
Bit	Peripheral Name	GPIO Name
3	QEP1/CAP1	IOPA3
4	QEP2/CAP2	IOPA4
5	CAP3	IOPA5
6	PWM1	IOPA6
7	PWM2	IOPA7
8	PWM3	IOPB0
9	PWM4	IOPB1
10	PWM5	IOPB2
11	PWM6	IOPB3

IO MUX Output Control Register C

Bit	Peripheral Name	GPIO Name
1	PWM7	IOPE1
2	PWM8	IOPE2
3	PWM9	IOPE3
4	PWM10	IOPE4
5	PWM11	IOPE5
6	PWM12	IOPE6
7	QEP3/CAP4	IOPE7
8	QEP4/CAP5	IOPF0
9	CAP6	IOPF1

C24x GPIO Digital Output

Dialog Box



IO Port

Select the input/output port to use: IOPA, IOPB, IOPC, IOPD, IOPE, or IOPF and select the bits to enable for digital output. If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Note that multiple C24x GPIO DO blocks cannot share the same I/O port. Only one bit is available for IOPD.

See Also

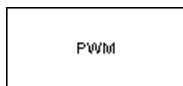
C24x GPIO Digital Input

Purpose

Pulse width modulators (PWMs)

Library

c2400dspchip1ib in Embedded Target for TI C2000 DSP

Description

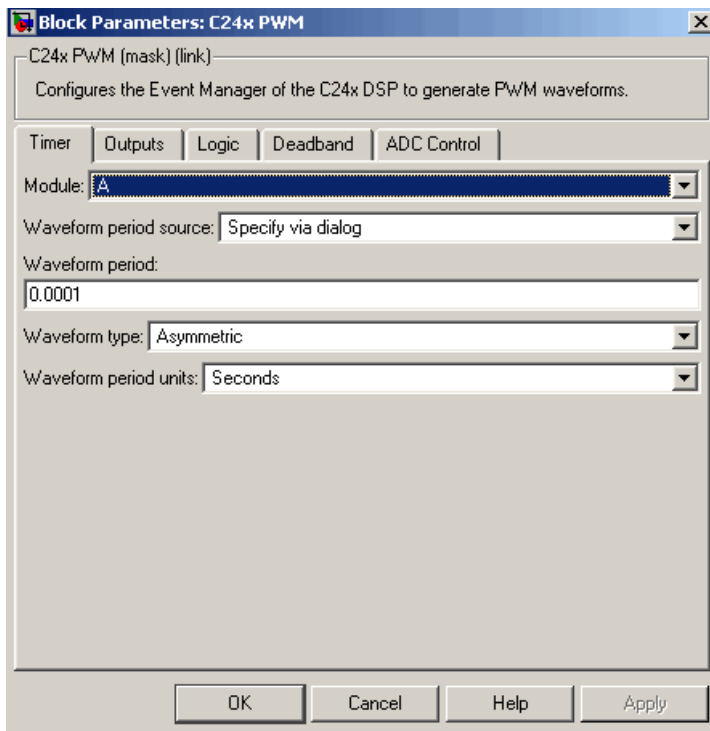
C24x PWM

LF2407 DSPs include a set of pulse width modulators 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 12 PWMs are configured in six pairs, with three pairs in each module.

Note All inputs to the C24x PWM block must be scalar values.

Dialog Box

Timer pane



Module

Specifies which target PWM pairs to use:

- A — Enables the PWMs in module A (PWM1/PWM2, PWM3/PWM4, and PWM5/PWM6).
- B — Enables 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. You should make sure that the **TimerClock** selected in the Scheduling section of the LF2407 eZdsp Target Preferences block does not conflict with the timers used for the PWMs.

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

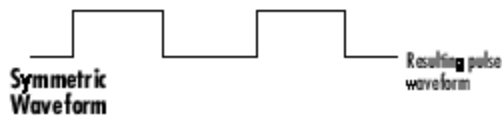
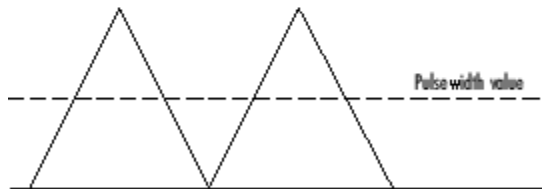
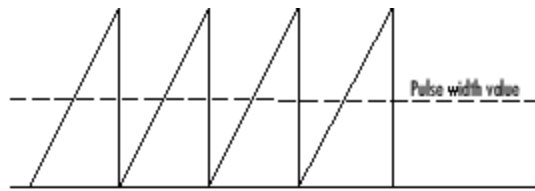
Period of the PWM waveform measured in clock cycles or in seconds, as specified in the **Waveform period units**.

Note The term *clock cycles* refers to the peripheral clock on the LF2407 chip. This clock is 40 MHz by default because the timer prescaler is set to 1.

Waveform type

Type of waveform to be generated by the PWM pair. The LF2407 PWMs can generate two types of waveforms: **Asymmetric** and **Symmetric**. The following illustration shows the difference between the two types of waveforms.

C24x PWM



Waveform period units

Units in which to measure the waveform period. Options are Clock cycles, which refer to the peripheral clock on the LF2407 chip (40 MHz), or Seconds. Note that changing these units changes the **Waveform period** value and the **Duty cycle** value and **Duty cycle units** selection.

Outputs pane

Block Parameters: C24x PWM

C24x PWM (mask) (link)

Configures the Event Manager of the C24x DSP to generate PWM waveforms.

Timer **Outputs** Logic Deadband ADC Control

Enable PWM1/PWM2

Duty cycle source: Specify via dialog

Duty cycle: 50

Enable PWM3/PWM4

Duty cycle source: Specify via dialog

Duty cycle: 50

Enable PWM5/PWM6

Duty cycle source: Specify via dialog

Duty cycle: 50

Duty cycle units: Percentages

OK Cancel Help Apply

Enable PWM#/PWM#

Select to activate the PWM pair(s).

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, in seconds, from the input port.

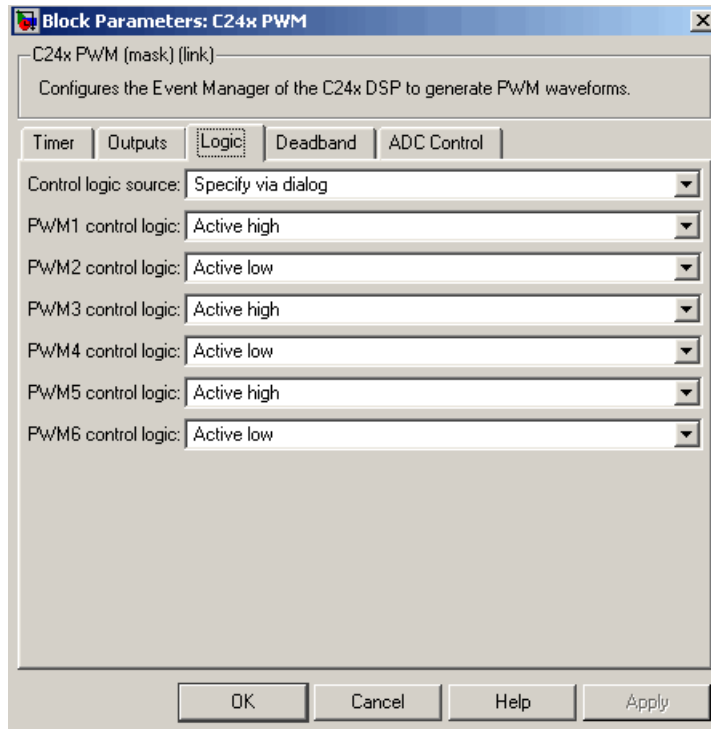
Duty cycle

PWM waveform pulse duration expressed in **Duty cycle units**.

Duty cycle units

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

Logic pane



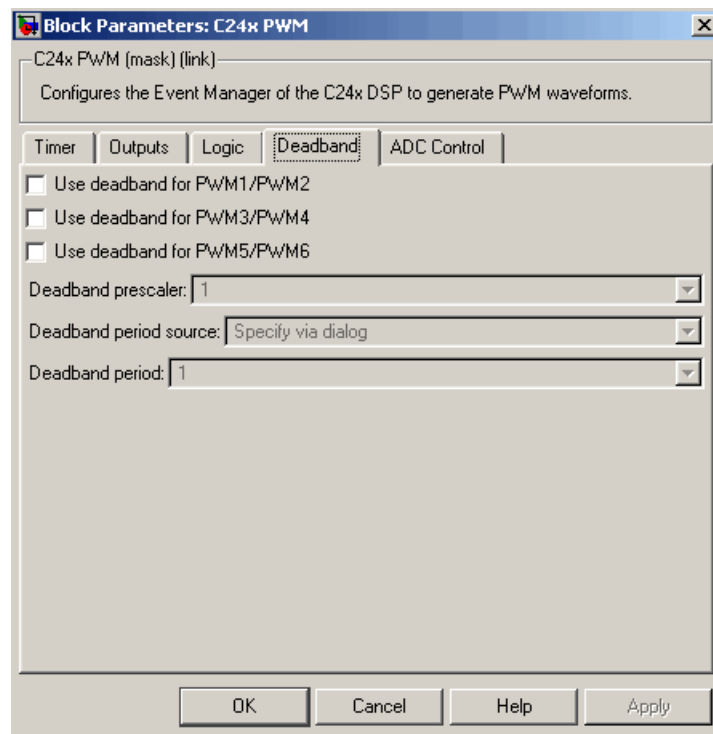
Control logic source

Source from which the control logic is obtained for all PWMs. Select **Specify via dialog** to enter the values in the **PWM# control logic** fields or select **Input port** to use values from the input port.

PWM# control logic

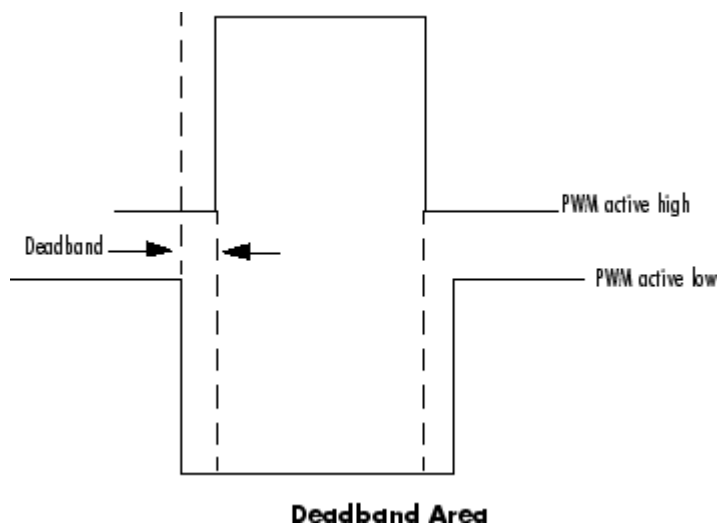
Control logic trigger for the PWM. Forced high causes the pulse value to be high. Active high causes the pulse value to go from low to high. Active low causes the pulse value to go from high to low. Forced low causes the signal to be low.

Deadband pane



Use deadband for PWM#/PWM#

Enables a deadband area of no signal overlap at the beginning of particular PWM pair signals. The following figure shows the deadband area.



Deadband prescaler

Number of clock cycles, which when multiplied by the deadband period, determines the size of the deadband. Selectable values are 1, 2, 4, 8, 16, and 32.

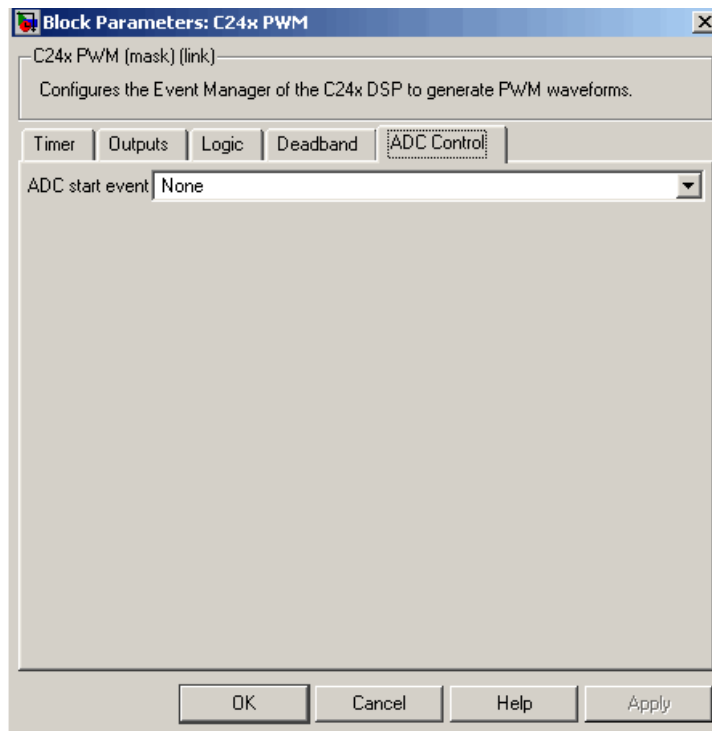
Deadband period source

Source from which to obtain the deadband period. Select **Specify** via **dialog** to enter the value in **Deadband period** or select **Input port** to use a value, in clock cycles, from an external source.

Deadband period

Value that, when multiplied by the deadband prescaler, determines the size of the deadband. Selectable values are from 1 to 15 clock cycles.

ADC Control pane



ADC start event

Controls whether this PWM and ADC associated with the same EV module are synchronized. Select None for no synchronization or select an interrupt to generate the source start-of-conversion (SOC) signal for the associated ADC.

- None — The ADC and PWM are not synchronized. The EV does not generate an SOC signal and the ADC is triggered by software (that is, the analog-to-digital conversion occurs when the ADC block is executed in the software).

C24x PWM

- 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 FFFFh.
- Period interrupt — The EV generates an SOC signal for the ADC associated with the same EV module when the value of the 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)/(CPU clock speed)**, zero-order hold interpolation will occur. For example, if you enter 64000 as the waveform period, the period for the ADC register is $64000/40 \text{ MHz} = 0.0016$. If you enter a **Sample time** in the C24x 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

C24x ADC

Purpose

Quadrature encoder pulse circuit

Library

c2400dspchip1lib in Embedded Target for TI C2000 DSP

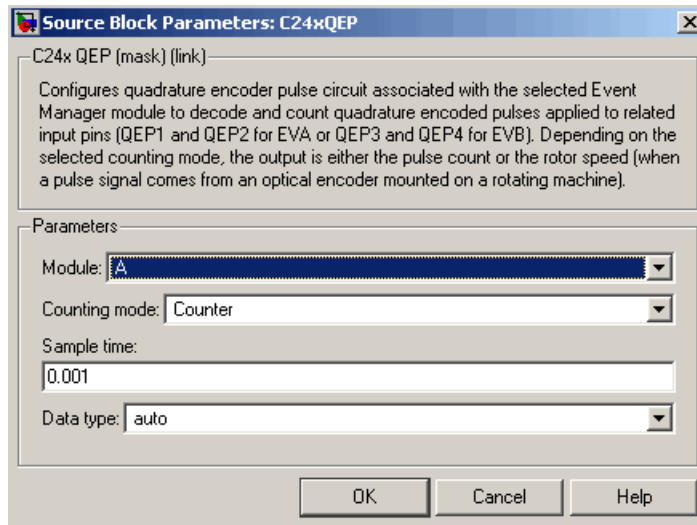
Description

Each L2407 Event Manager has three capture units, which can log transitions on its capture unit pins. Event Manager A (EVA) uses capture units 1, 2, and 3. Event Manager B (EVB) uses capture units 4, 5, and 6.

The quadrature encoder pulse (QEP) circuit decodes and counts quadrature encoded input pulses on these capture unit pins. QEP pulses are two sequences of pulses with varying frequency and a fixed phase shift of 90 degrees (or one-quarter of a period). Both edges of the QEP pulses are counted so the frequency of the QEP clock is four times the input sequence frequency.

The QEP, in combination with an optical encoder, is particularly useful for obtaining speed and position information from a rotating machine. Logic in the QEP circuit determines the direction of rotation by which sequence is leading. For module A, if the QEP1 sequence leads, the general-purpose (GP) timer counts up and if the QEP2 sequence leads, the timer counts down. The pulse count and frequency determine the angular position and speed.

Dialog Box



Module

Specifies which QEP pins to use:

- A — Uses QEP1 and QEP2 pins.
- B — Uses QEP3 and QEP4 pins.

Counting mode

Specifies how to count the QEP pulses:

- Counter — Count the pulses based on the board's GP Timer 2 (or GP Timer 4 for EVB).
- RPM — Count the machine's revolutions per minute.

Positive rotation

Defines whether to use Clockwise or Counterclockwise as the direction to use as positive rotation. This field appears only if you select RPM above.

Encoder resolution

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

Sample time

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

Data type

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

C24x SCI Receive

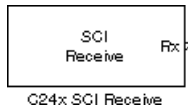
Purpose

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

Library

c2400dspchip1ib in Embedded Target for TI C2000 DSP

Description

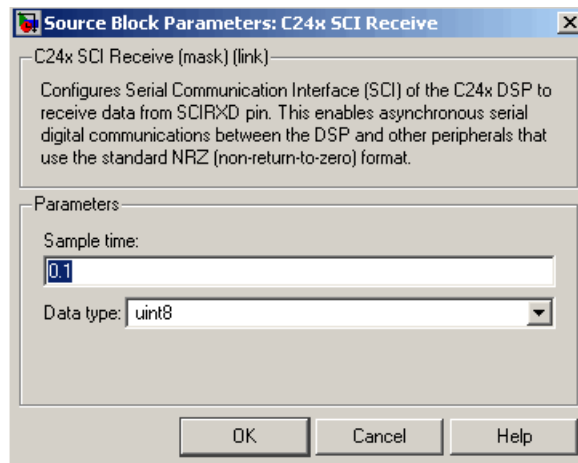


The C24x SCI Receive block supports asynchronous serial digital communications between the target and other asynchronous peripherals in nonreturn-to-zero (NRZ) format. This block configures the C24x DSP target to receive scalar or vector data from the COM port via the C24x target's COM port.

Note You can have only one C24x SCI Receive block in a single model.

Many SCI-specific settings are in the **DSPBoard** section of the LF2407 eZdsp target preferences block. You should verify that these settings are correct for your application.

Dialog Box



Note If you open this block from the SCI-Based Host-Target Communication demo, you will see an additional parameter used only in that demo.

Sample time

Sample time, T_s , for the block's input sampling.

Data type

Data type of the output data. Available options are int8 and uint8.

See Also

C24x SCI Transmit

C24x SCI Transmit

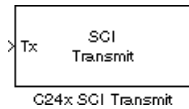
Purpose

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

Library

c2400dspchip1ib in Embedded Target for TI C2000 DSP

Description

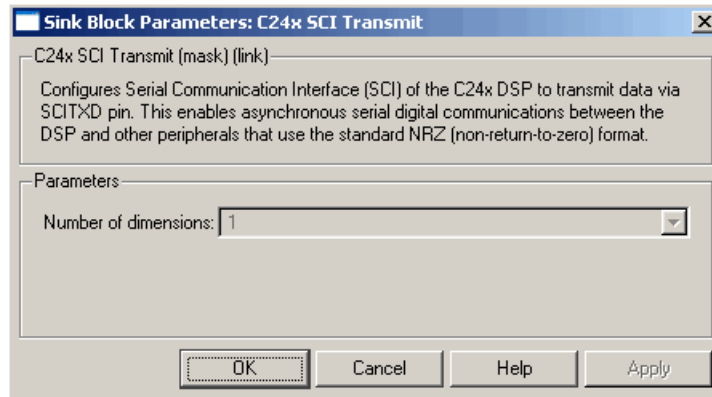


The C24x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C24x target's COM ports in nonreturn-to-zero (NRZ) format. You can specify how many of the six target COM ports to use. The sampling rate and data type are inherited from the input port. If no data type is specified, the default data type is `uint8`.

Note You can have only one C24x SCI Transmit block in a single model.

Many SCI-specific settings are in the **DSPBoard** section of the LF2407 eZdsp target preferences block. You should verify that these settings are correct for your application.

Dialog Box



Note The parameter shown in this block is active only for demos, i.e., if you open the block from the SCI-Based Host-Target Communication demo.

See Also

C24x SCI Receive

C24x SPI Receive

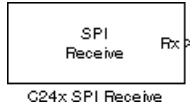
Purpose

Receive data via serial peripheral interface (SPI) on target

Library

c2400dspchip1ib in Embedded Target for TI C2000 DSP

Description

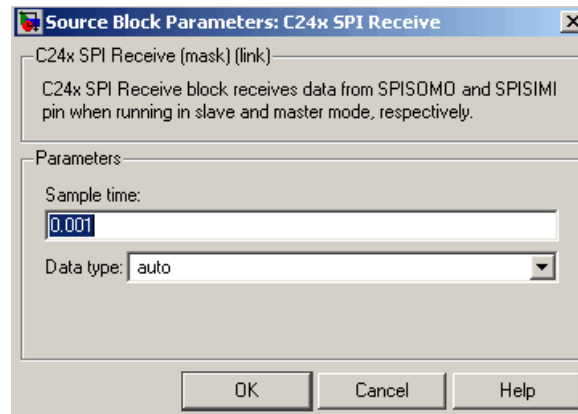


The C24x SPI Receive block supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. 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.

Note You can have only one C24x SPI Receive block in a single model.

Many SPI-specific settings are in the **DSPBoard** section of the LF2407 eZdsp target preferences block. You should verify that these settings are correct for your application.

Dialog Box



Sample time

Sample time, T_s , for the block's input sampling.

Data type

Data type of the output data. Available options are auto, double, single, int8, uint8, int16, uint16, int32, uint32, and boolean.

See Also

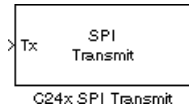
C24x SPI Transmit

C24x SPI Transmit

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

Library c2400dspchip1ib in Embedded Target for TI C2000 DSP

Description



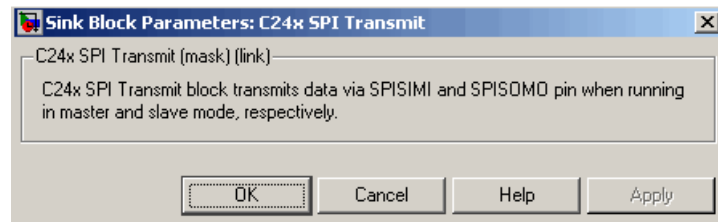
The C24x SPI Transmit block supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives 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 and data type are inherited from the input port. If no data type is specified, the default data type is uint16.

Note You can have only one C24x SPI Transmit block in a single model.

Many SPI-specific settings are in the **DSPBoard** section of the LF2407 eZdsp target preferences block. You should verify that these settings are correct for your application.

Dialog Box



See Also C24x SPI Receive

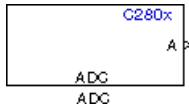
Purpose

Analog-to-digital converter (ADC)

Library

c280xdspchip1lib in Embedded Target for TI C2000 DSP

Description



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

Output

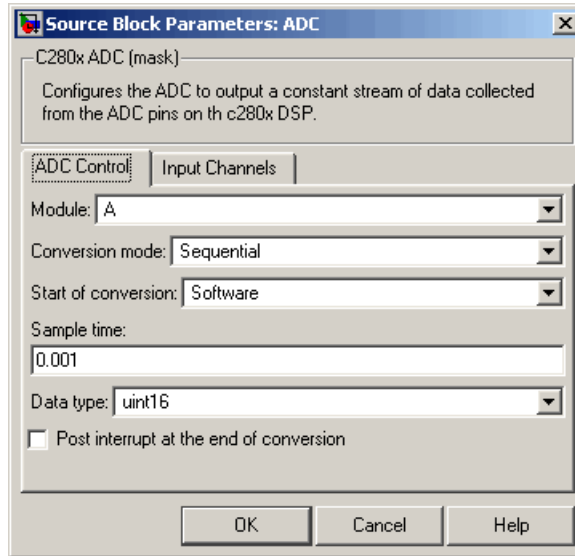
The output of the C280x ADC is a vector of uint16 values. The output values are in the range 0 to 4095 because the C280x ADC is 12-bit converter.

Modes

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

Dialog Box

ADC Control pane



Module

Specifies which DSP module to use:

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

Conversion mode

Type of sampling to use for the signals:

- Sequential — Samples the selected channels sequentially.

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

Start of conversion

Type of signal that triggers conversions to begin:

- Software — Signal from software. Conversion values are updated at each sample time.
- ePWMxA / ePWMxB / ePWMxA_ePWMxB — Start of conversion is controlled by user-defined PWM events.
- XINT2_ADCSOC — Start of conversion is controlled by the XINT2_ADCSOC external signal pin.

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

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

Sample time

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. See “Scheduling and Timing” on page 1-11 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 1-13 for a discussion of block placement and other necessary settings.

C280x ADC

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

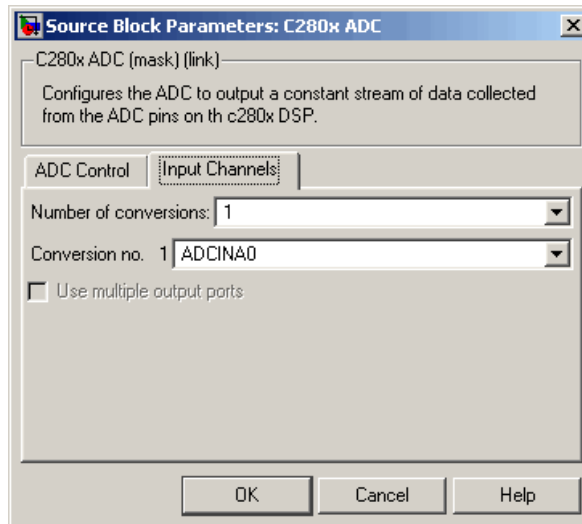
Data type

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

Post interrupt at the end of conversion

Select this check box to post an asynchronous interrupt at the end of each conversion. Note that the interrupt is always posted at the end of conversion. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-13 for a discussion of block placement and other necessary settings.

Input Channels pane



Number of conversions

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

Conversion no.

Specific ADC channel to associate with each conversion number.

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

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

Use multiple output ports

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

See Also

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

C280x eCAN Receive

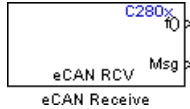
Purpose

Enhanced Control Area Network receive mailbox

Library

c280xdspchip1ib in Embedded Target for TI C2000 DSP

Description

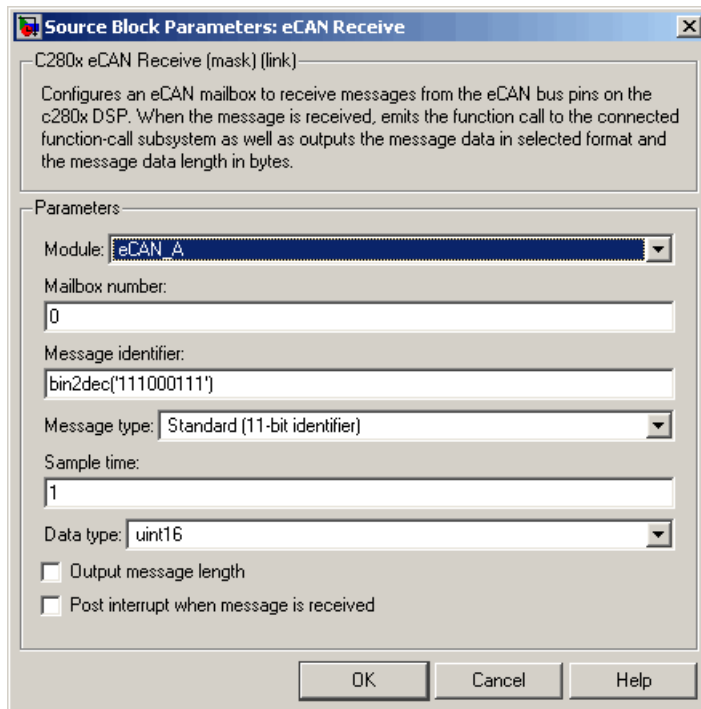


The C280x enhanced Control Area Network (eCAN) Receive block generates source code for receiving eCAN messages through an eCAN mailbox. The eCAN modules on the DSP chip provide serial communication capability and have 32 mailboxes configurable for receive or transmit. The C280x supports eCAN data frames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

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

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

Dialog Box



Module

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

Mailbox number

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

Message identifier

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

C280x eCAN Receive

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 1-13 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 rightaligned in the output. Only uint16 (vector length = 4 elements) or uint32 (vector length = 8 elements) data are allowed. The data are unpacked as follows using the data buffer, which is 8 bytes.

For uint16 data,

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

For uint32 data,

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

For example, if the received message has two bytes,

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

then the uint16 output would be:

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

Output message length

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

Post interrupt when message is received

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

References

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

See Also

C280x eCAN Transmit, C280x Hardware Interrupt

C280x eCAN Transmit

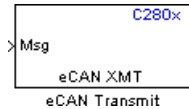
Purpose

Enhanced Control Area Network transmit mailbox

Library

c280xdspchiplib in Embedded Target for TI C2000 DSP

Description



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

Data Vectors

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

For input of type `uint32`,

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

the data buffer is:

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

For input of type `uint16`,

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

the data buffer is:

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

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

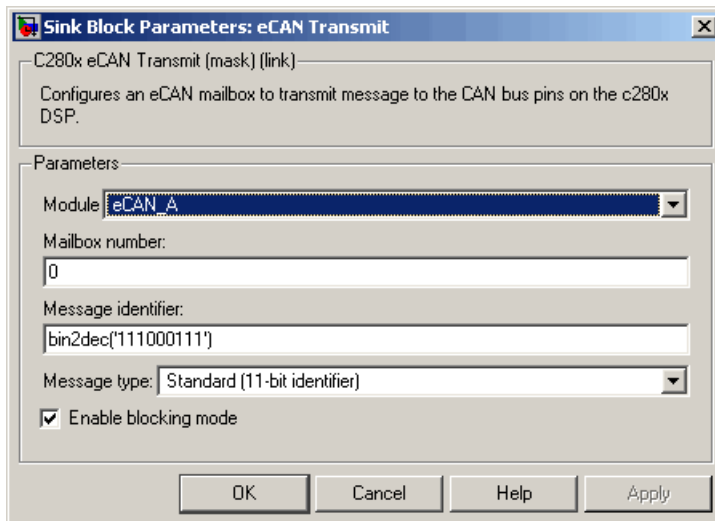
```
inputdata [0] = 0x1234
inputdata [1] = 0x5678
```

the data buffer is:

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

C280x eCAN Transmit

Dialog Box



Module

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

Mailbox number

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

Message identifier

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

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.

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

References

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

See Also

C280x eCAN Receive

C280x ePWM

Purpose

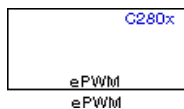
Configure C280x Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms

Library

c280xdspchip1lib in Embedded Target for TI C2000 DSP

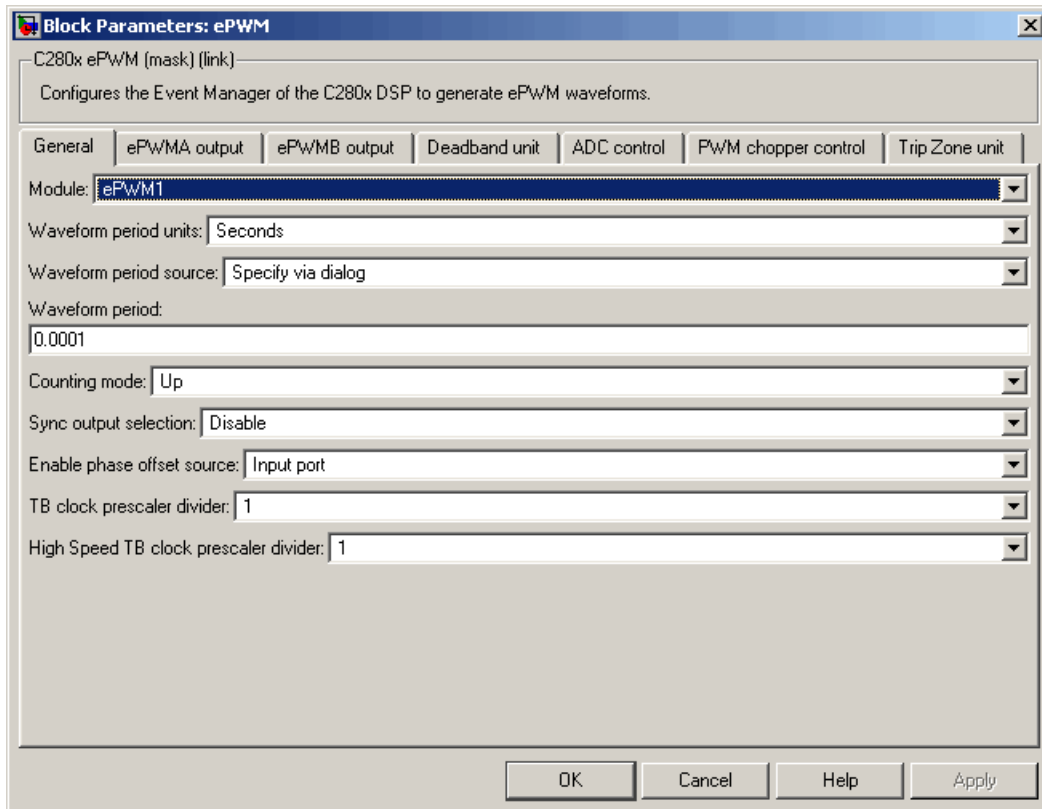
Description

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



Dialog Box

General pane



Module

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

Waveform period units

Specifies the units in which the **Waveform period** is expressed. Choose Seconds (the default) or Clock cycles.

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.

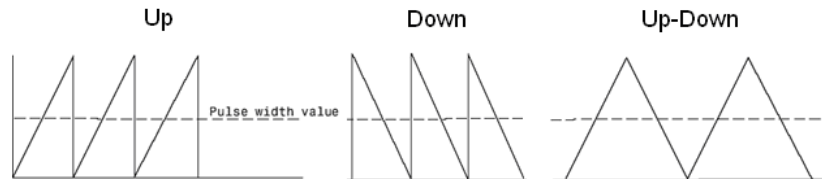
Waveform period

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

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

Counting mode

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



Sync output selection

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

Enable S/W sync input port

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

Enable phase offset source

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

Phase offset value

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

TB clock prescaler divider

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

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

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

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

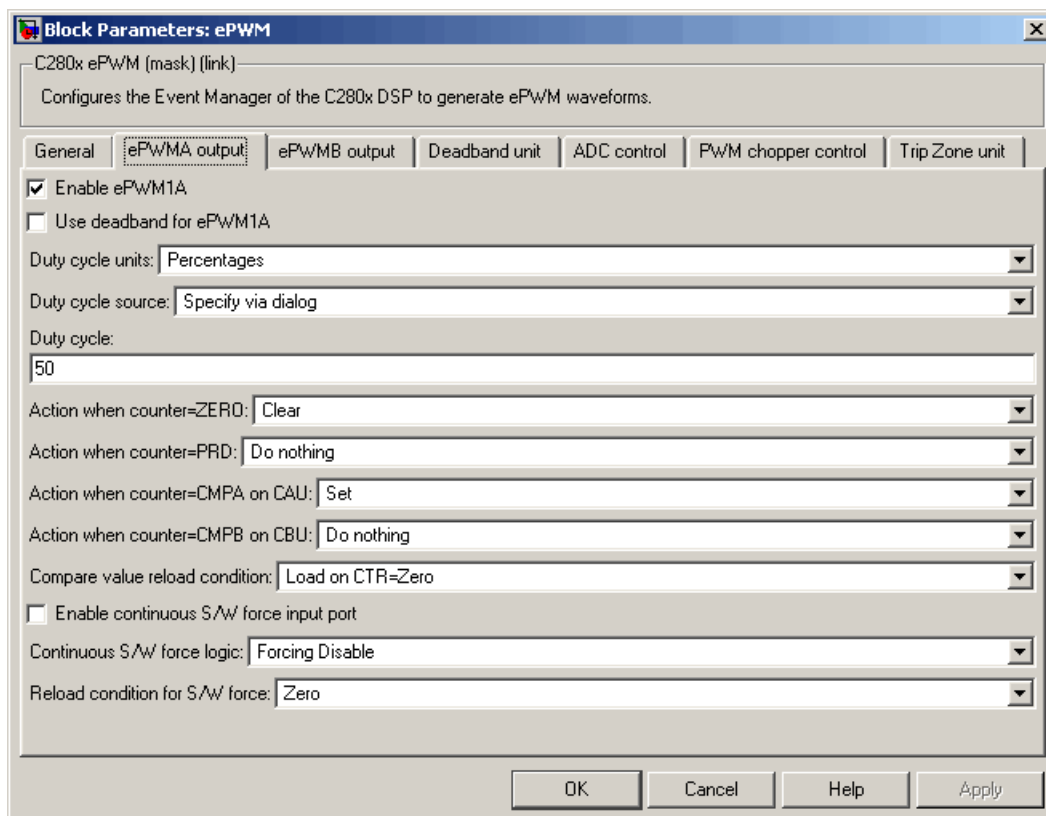
High Speed TB clock prescaler divider

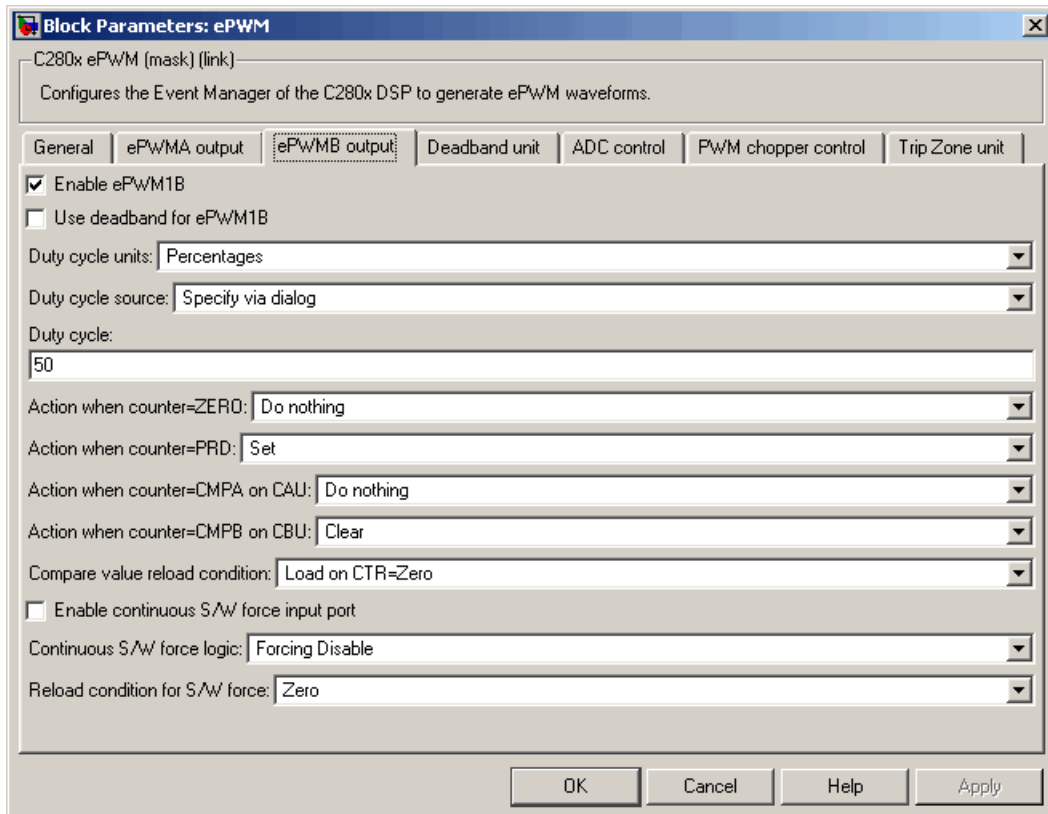
See the discussion of the **TB clock prescaler divider** above for an explanation of this value's role in setting the speed of the Time-base Clock. Choices are 1, 2, 4, 6, 8, 10, 12, and 14.

C280x ePWM

ePWMA output and ePWMB output panes

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





Enable ePWMxA Enable ePWMxB

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

Use deadband for ePWMxA

Use deadband for ePWMxB

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

Duty cycle units

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

Duty cycle source

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

Duty cycle

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

Action when counter=ZERO

Action when counter=PRD

Action when counter=CMPA on CAU

Action when counter=CMPA on CAD

Action when counter=CMPB on CBU

Action when counter=CMPB on CBD

These settings, along with the other remaining settings in the **ePWMA output** and **ePWMB output** panes, determine the behavior of the Action Qualifier (AQ) submodule. Based on these settings, the AQ module decides which events are converted into various action types, thereby producing the required switched waveforms of the ePWMxA and ePWMxB output signals.

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

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

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

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

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

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

Compare value reload condition determines if and when the Action-qualifier S/W Force Register is reloaded from a shadow register. Choices are Load on CTR=Zero (the default), Load on CTR=PRD, Load on either, and Freeze.

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

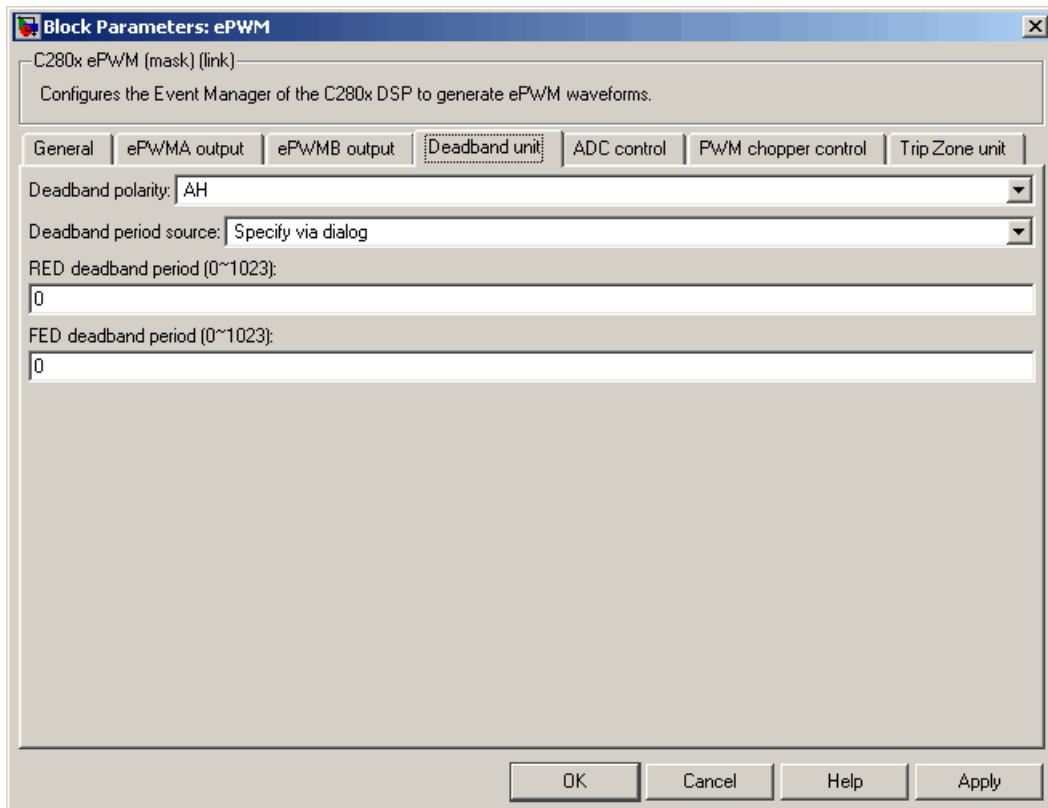
by default. Select this check box to obtain the control logic from the input port

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

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

Deadband unit pane

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



Deadband polarity

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

Deadband period source

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

RED deadband period

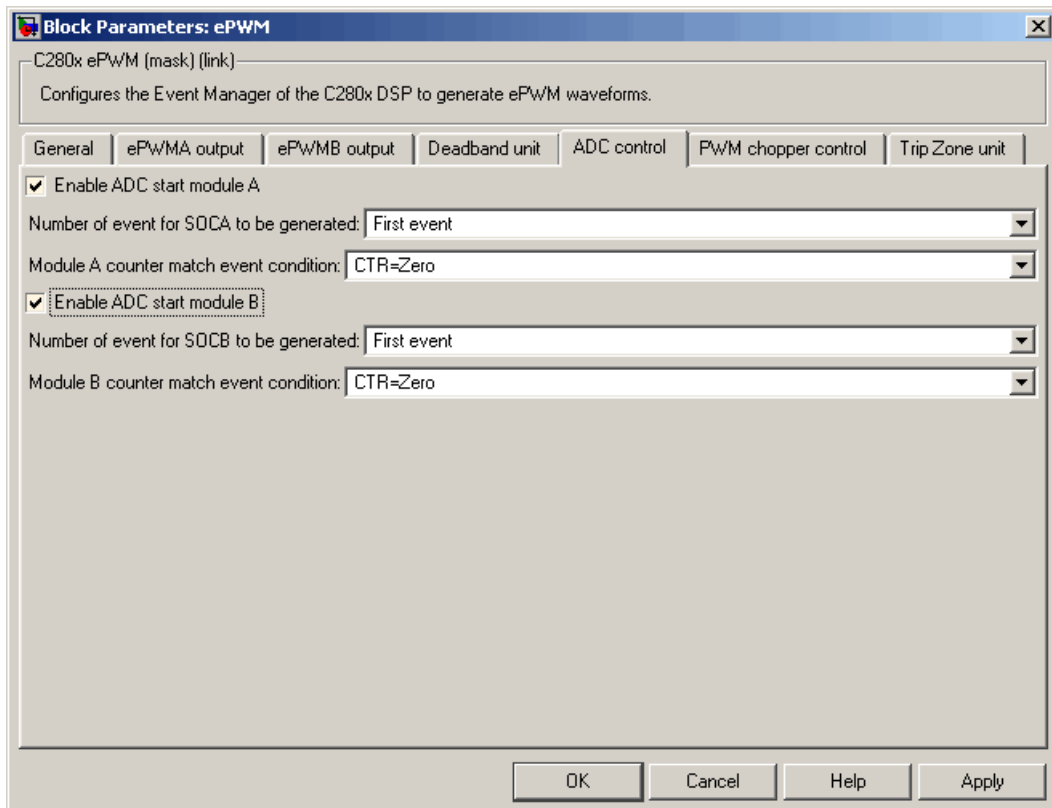
This field appears only when **Use deadband for ePWMxA** is selected in the **ePWMA output** pane. Enter a value from 0 to 1023 to specify a rising edge delay.

FED deadband period

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

ADC control pane

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



Enable ADC start module A

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

Number of event for SOCA to be generated

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

Module A counter match event condition

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

Enable ADC start module B

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

Number of event for SOCB to be generated

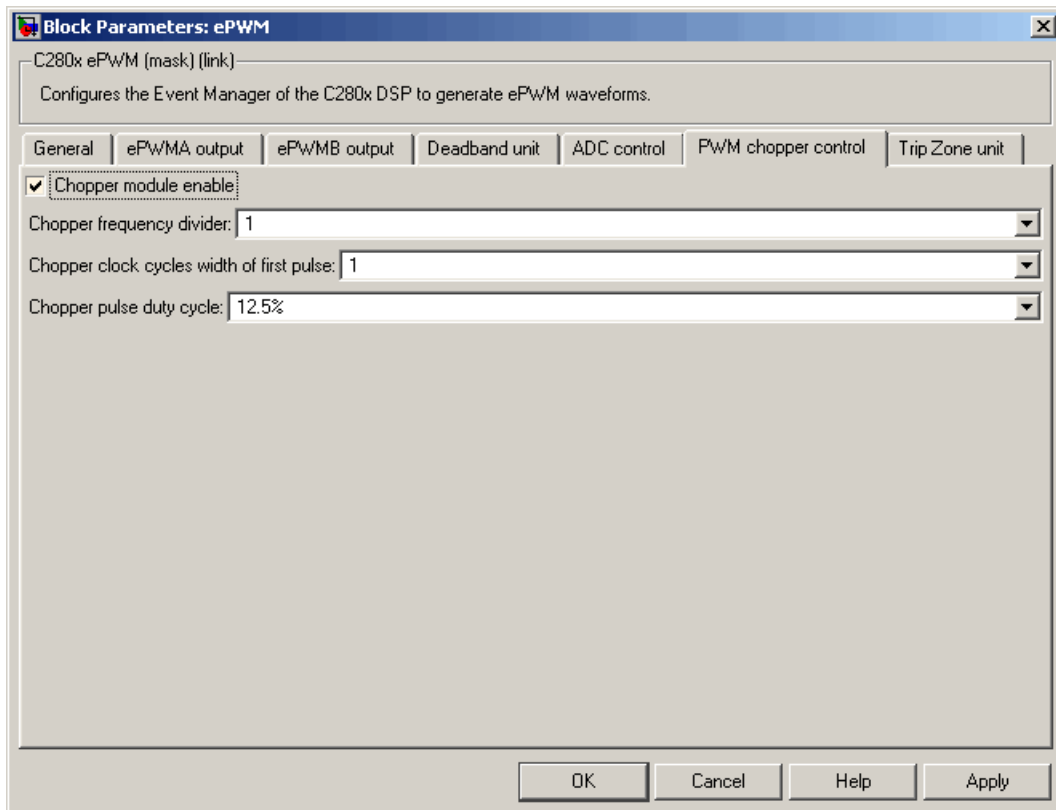
This field appears only when you select the **Enable ADC start module B** check box. Specify how often you want ADC start of conversion to be triggered. First event triggers ADC start of conversion with every event, Second event triggers ADC start of conversion with every second event, and Third event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

Module B counter match event condition

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

PWM chopper control pane

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



Chopper module enable

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

Chopper frequency divider

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

Chopper clock cycles width of first pulse

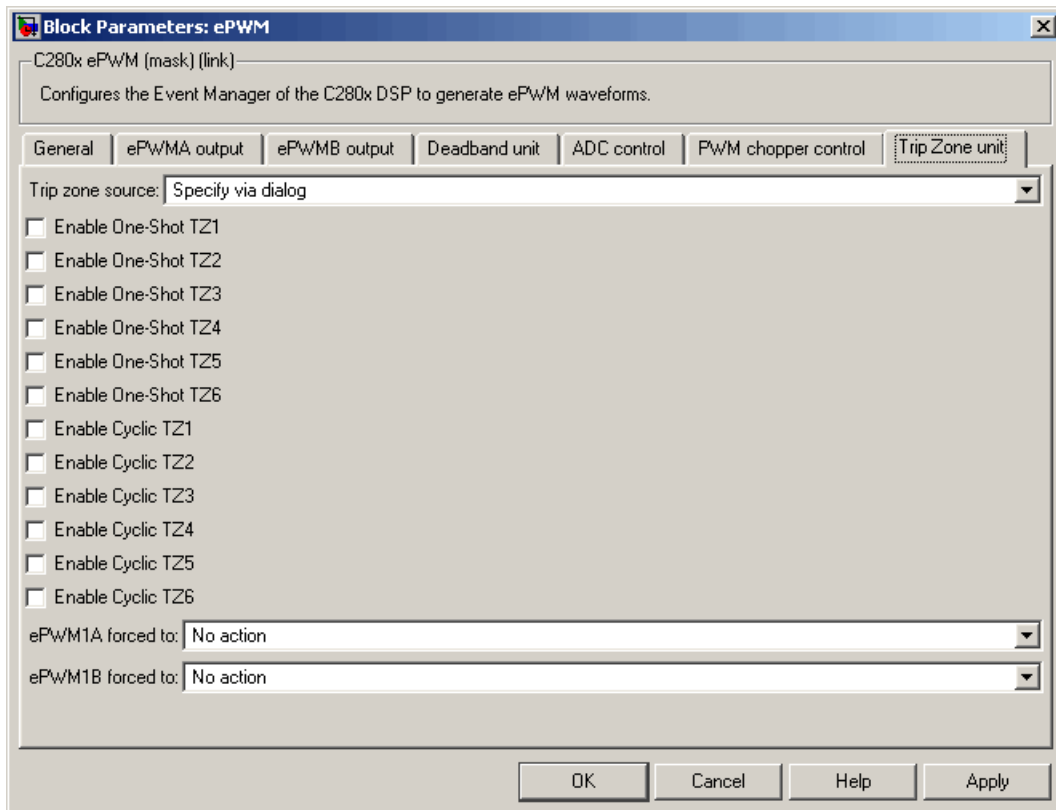
Choose an integer value from 1 to 16 to set the width of the first pulse. Use this feature to provide a high-energy first pulse to ensure hard and fast power switch turn on.

Chopper pulse duty cycle

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

Trip Zone unit pane

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



Trip zone source

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

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

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

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

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

ePWMxA forced to**ePWMxB forced to**

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

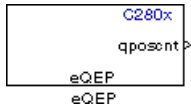
See Also

C280x ADC

Purpose Quadrature encoder pulse circuit

Library c280xdspchip1lib in Embedded Target for TI C2000 DSP

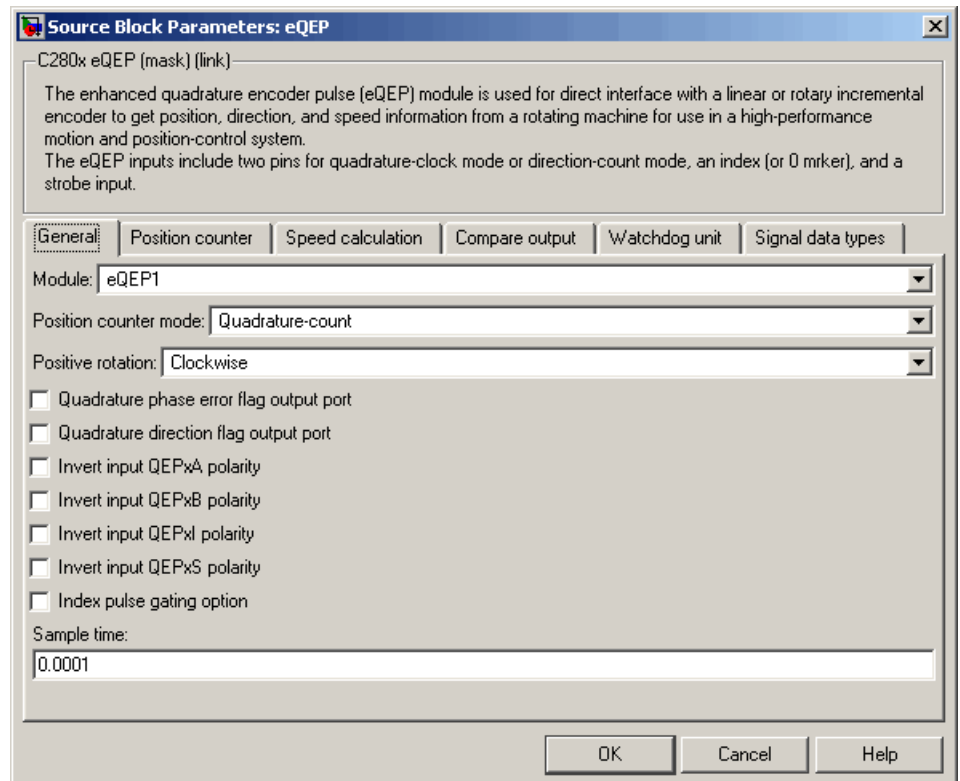
Description



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

Dialog Box

General pane



Module

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

Position counter mode

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

Positive rotation

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

External clock rate

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

Quadrature phase error flag output port

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

Quadrature direction flag output port

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

Invert input QEPxA polarity

Invert input QEPxB polarity

Invert input QEPxI polarity

Invert input QEPxS polarity

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

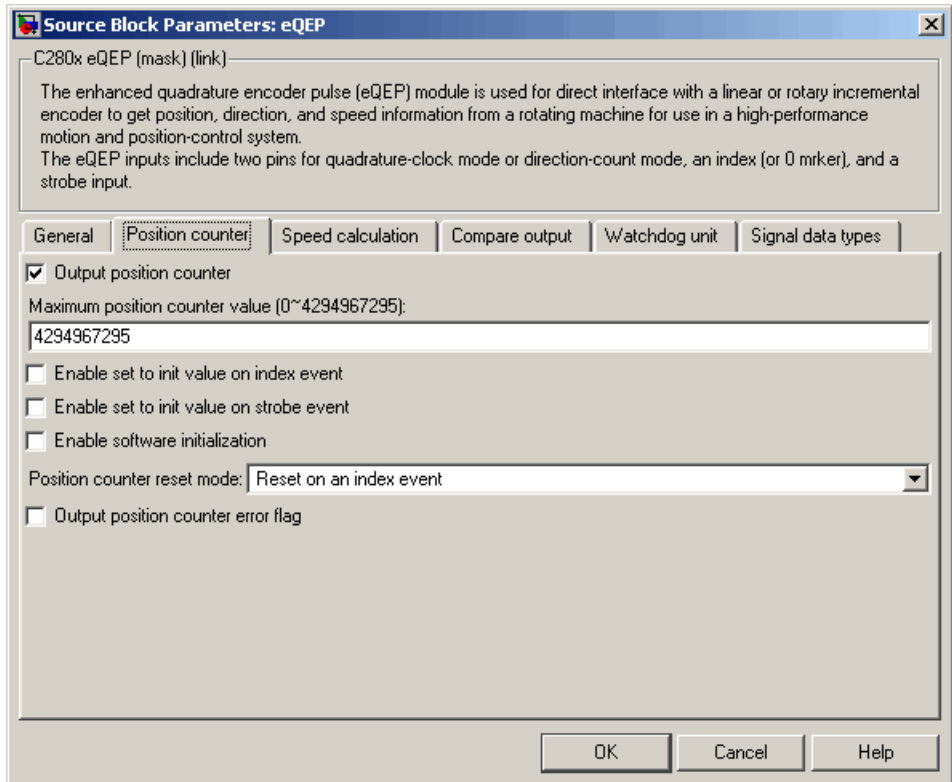
Index pulse gating option

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

Sample time

Enter the sample time in seconds.

Position counter pane



Output position counter

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

Maximum position counter value

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

Enable set to init value on index event

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

Set to init value on index event

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

Enable set to init value on strobe event

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

Set to init value on strobe event

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

Enable software initialization

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

Software initialization source

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

Initialization value

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

Position counter reset mode

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

Reset on the first index event, or Reset on a time unit event.

Output position counter error flag

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

Output latch position counter on index event

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

Index event latch of position counter

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

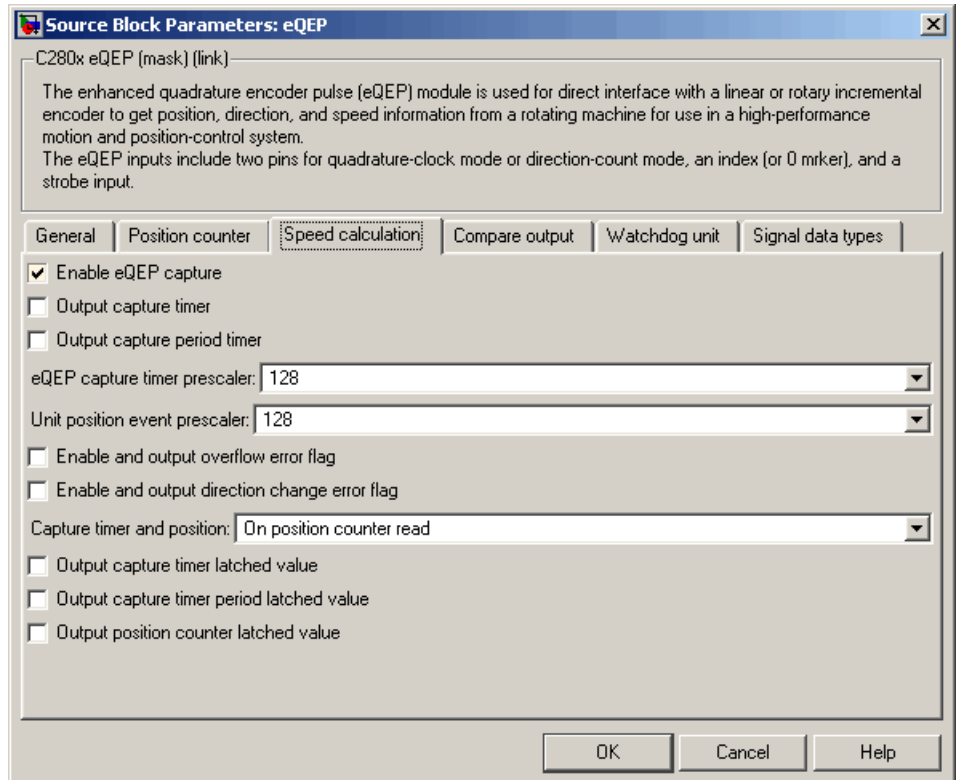
Output latch position counter on strobe event

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

Strobe event of latched position counter

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

Speed calculation pane



Enable QEP capture

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

Output capture timer

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

Output capture period timer

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

eQEP capture timer prescaler

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

Unit position event prescaler

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

Enable and output overflow error flag

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

Enable and output direction change error flag

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

Capture timer and position

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

Unit timer period

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

Output capture timer latched value

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

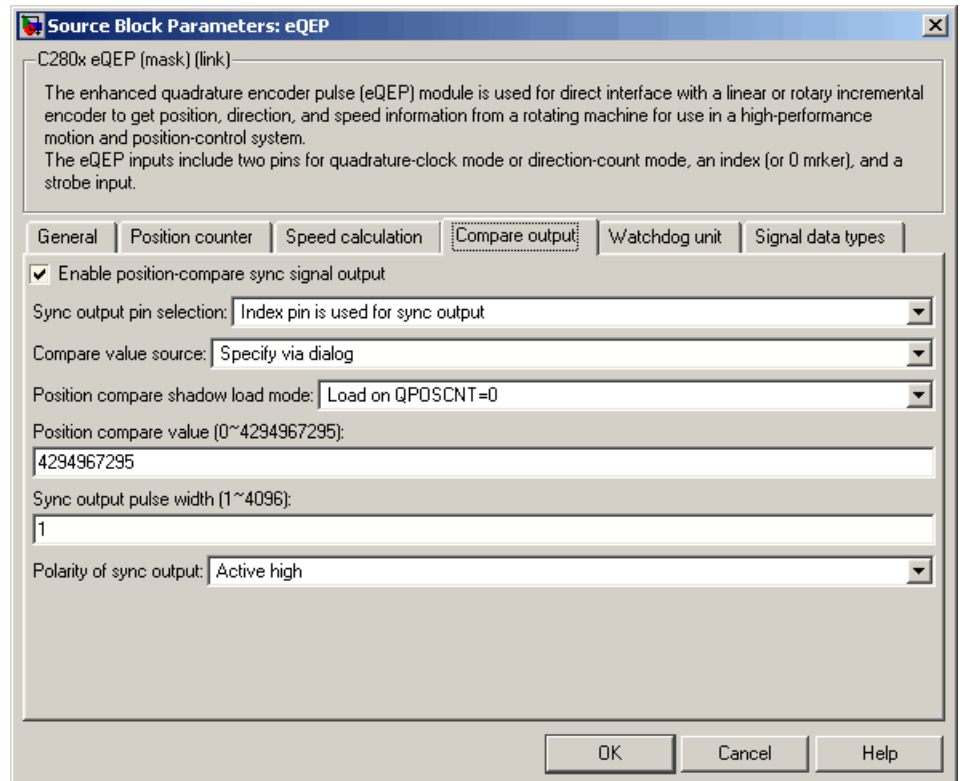
Output capture timer period latched value

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

Output position counter latched value

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

Compare output pane



Enable position-compare sync signal output

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

Sync output pin selection

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

Compare value source

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

Position compare shadow load mode

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

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

Position compare value

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

Sync output pulse width

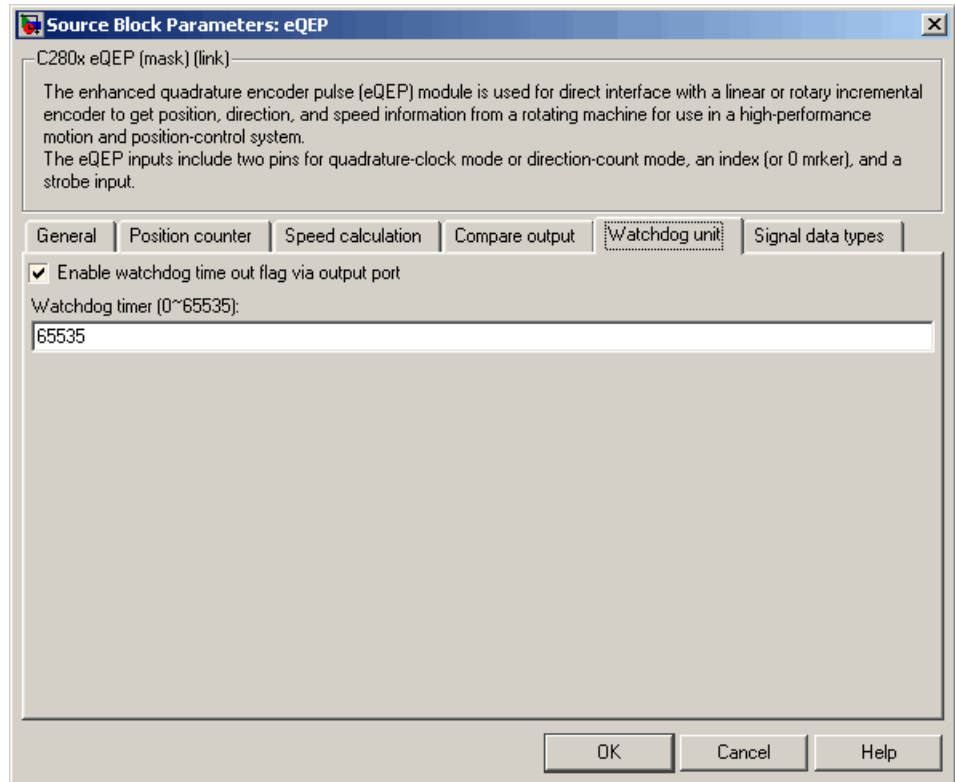
The pulse stretcher logic in the position-compare unit generates a programmable position-compare sync pulse output on the position-compare match.

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

Polarity of sync output

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

Watchdog unit pane



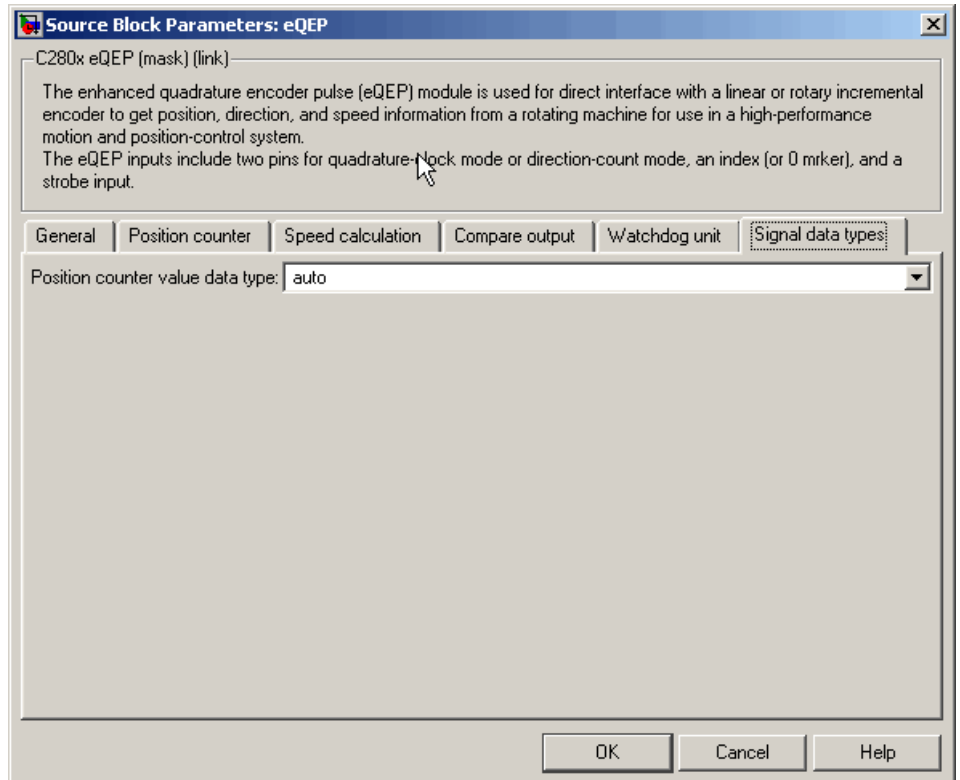
Enable watchdog time out flag via output port

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

Watchdog timer

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

Signal data types pane



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

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

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

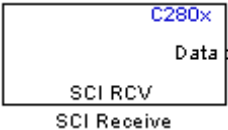
The fields that appear on the **Signal data types** pane are named similarly to these options. For example, **Position counter value data type** on the **Signal data types** pane corresponds to the **Output position counter** option on the **Position counter** pane.

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

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

Library c280xdspchip1lib in Embedded Target for TI C2000 DSP

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

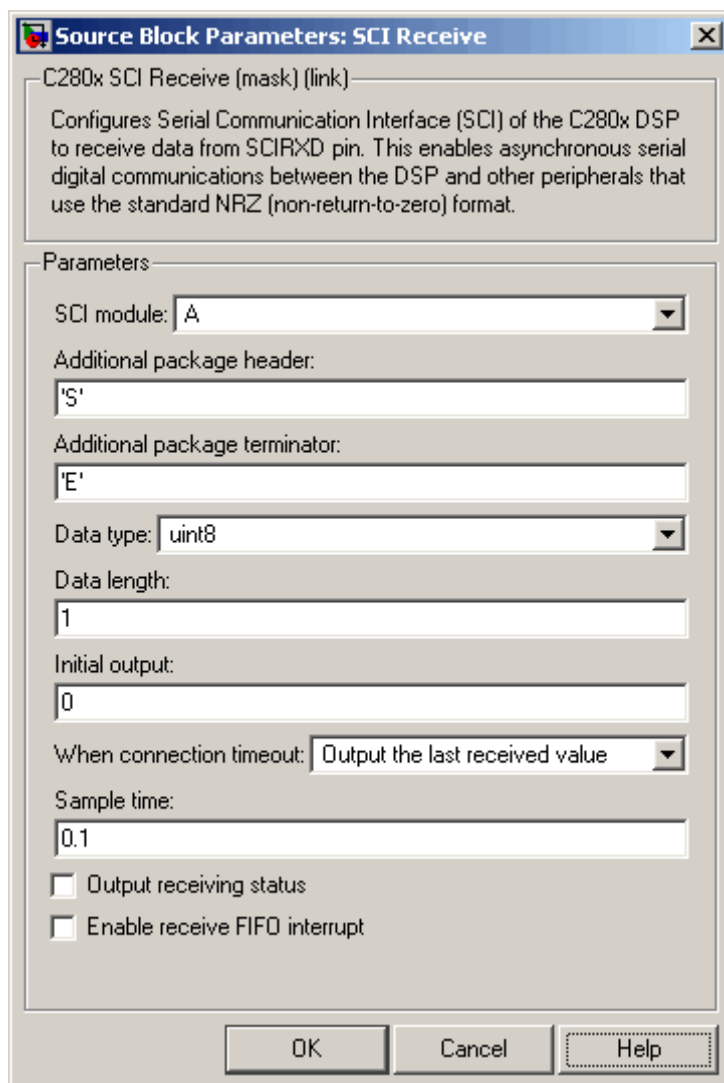


Note For any given model, you can have only one C280x SCI Receive block per module. There are two modules, A and B, which can be configured through the F2808 eZdsp target preferences block.

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

C280x SCI Receive

Dialog Box



The dialog box is titled "Source Block Parameters: SCI Receive" and contains the following fields and options:

- SCI module:** A dropdown menu with "A" selected.
- Additional package header:** A text input field containing "S".
- Additional package terminator:** A text input field containing "E".
- Data type:** A dropdown menu with "uint8" selected.
- Data length:** A text input field containing "1".
- Initial output:** A text input field containing "0".
- When connection timeout:** A dropdown menu with "Output the last received value" selected.
- Sample time:** A text input field containing "0.1".
- Output receiving status
- Enable receive FIFO interrupt

At the bottom of the dialog box are three buttons: "OK", "Cancel", and "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.

Data type

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

Data length

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

Initial output

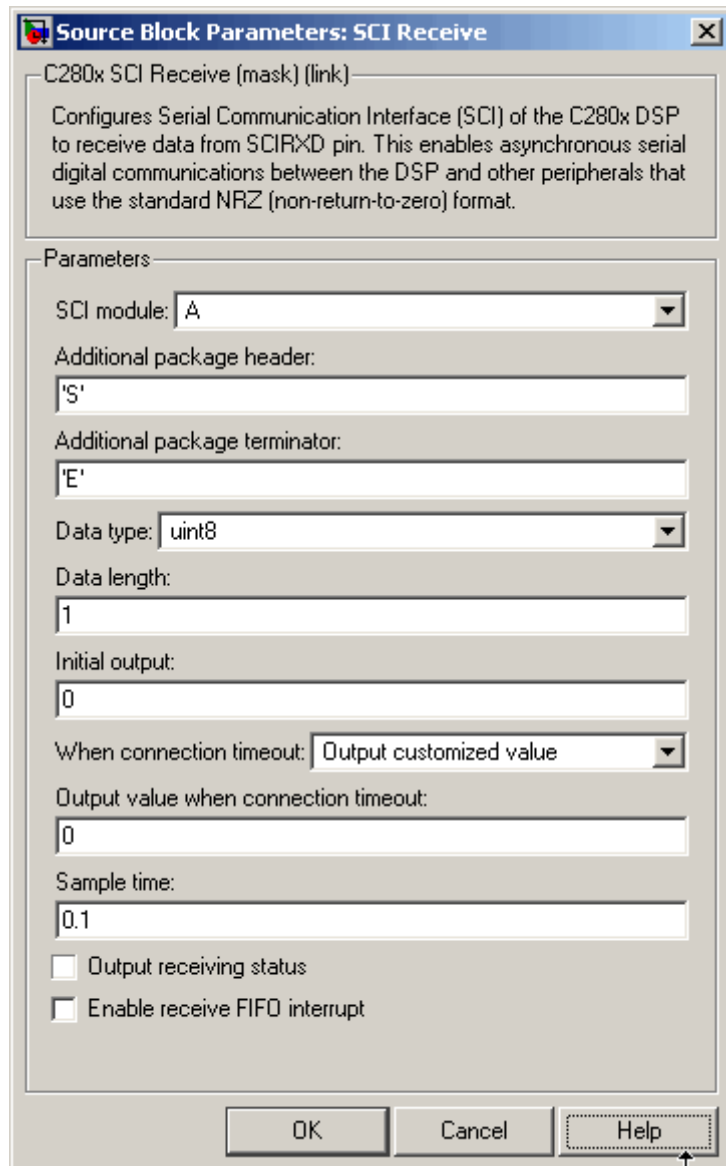
Default value from the c280x SCI Receive block. This value is used, for example, if a connection time-out occurs and the **When**

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

When connection timeout

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

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



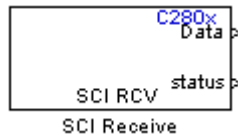
C280x SCI Receive

Sample time

Sample time, T_s , for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-13 for a discussion of block placement and other necessary settings.

Output receiving status

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



The error status may be one of the following values:

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

Enable receive FIFO interrupt

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

References

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

See Also

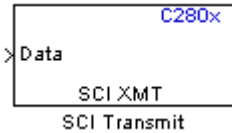
C280x SCI Transmit, C280x Hardware Interrupt

C280x SCI Transmit

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

Library c280xdspchip1lib in Embedded Target for TI C2000 DSP

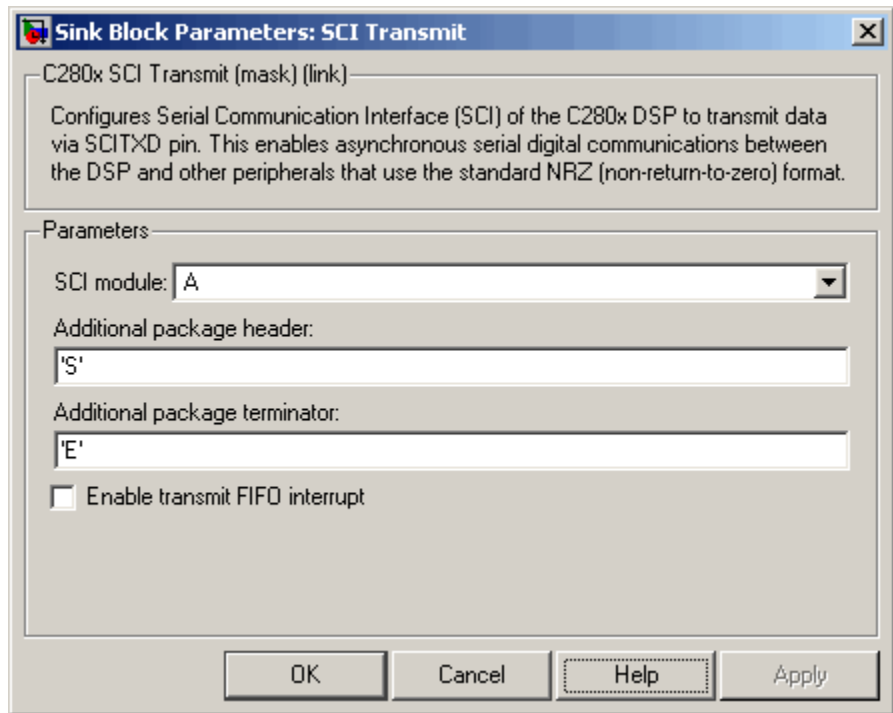
Description The C280x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C280x target's COM ports in nonreturn-to-zero (NRZ) format. You can specify how many of the six target COM ports to use. The sampling rate and data type are inherited from the input port. The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



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

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

Dialog Box



SCI module

SCI module to be used for communications.

Additional package header

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

C280x SCI Transmit

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

Additional package terminator

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

Enable transmit FIFO interrupt

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

References

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

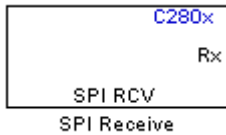
See Also

C280x SCI Receive, C280x Hardware Interrupt

Purpose Receive data via the serial peripheral interface (SPI) on the target

Library c280xdspchip1lib in Embedded Target for TI C2000 DSP

Description



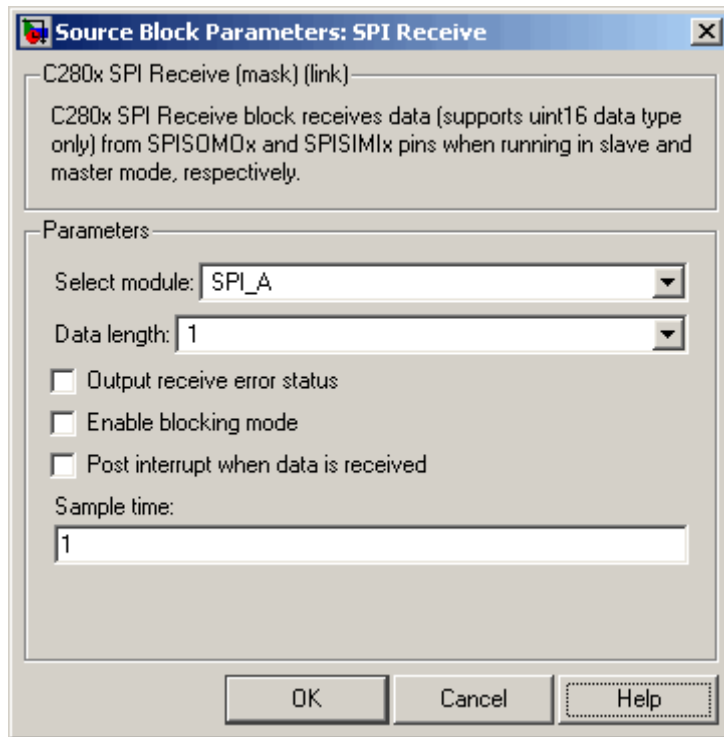
The C280x SPI Receive supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data.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

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

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

C280x SPI Receive

Dialog Box



Select module

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

Data length

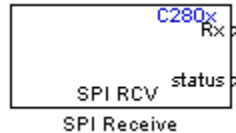
Specifies how many uint16s are expected to be received. Select 1 through 16.

Enable blocking mode

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

Output receive error status

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



Error status may be one of the following values:

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

Post interrupt when data is received

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

Sample time

Sample time, T_s , for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt when message is received** box, and refer to "Asynchronous Interrupt Processing" on page 1-13 for a discussion of block placement and other necessary settings.

See Also

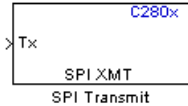
C280x SPI Transmit, C280x Hardware Interrupt

C280x SPI Transmit

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

Library c280xdspchip1ib in Embedded Target for TI C2000 DSP

Description



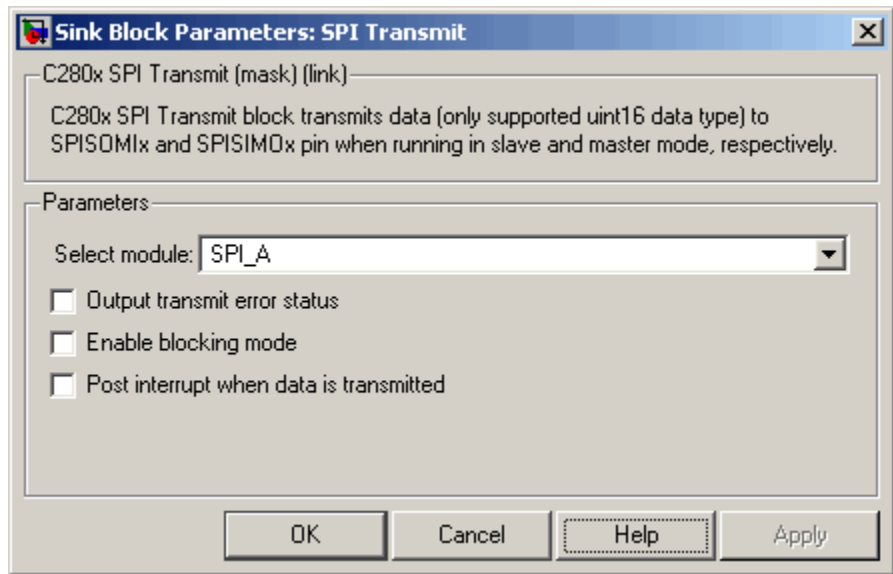
The C280x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data.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

The sampling rate and data type are inherited from the input port. If no data type is specified, the default data type is uint16.

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

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

Dialog Box

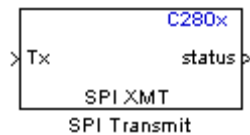


Select module

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

Output transmit error status

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



Error status may be one of the following values:

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

C280x SPI Transmit

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

Enable blocking mode

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

Post interrupt when data is transmitted

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

See Also

C280x SPI Receive, C280x Hardware Interrupt

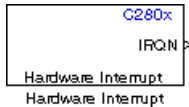
Purpose

Create Interrupt Service Routine to handle hardware interrupts

Library

c280xdspchip1lib in Embedded Target for TI C2000 DSP

Description



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

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

- C280x ADC
- C280x eCAN Receive
- C280x SCI Receive
- C280x SCI Transmit
- C280x SPI Receive
- C280x SPI Transmit

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

Vectorized Output

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

Each interrupt is described by:

C280x Hardware Interrupt

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

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

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

C280x Peripheral Interrupt Vector Values

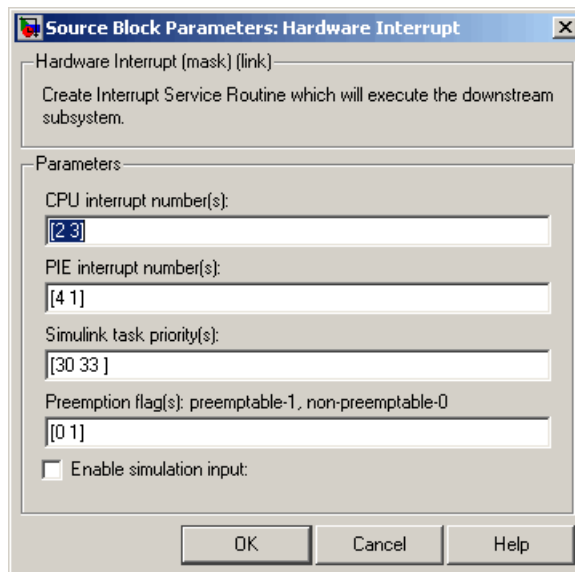
Row numbers = CPU values / Column numbers = PIE values								
	8	7	6	5	4	3	2	1
1	WAKEINT (LPM/WVD)	TINT0 (TIMER 0)	ADCINT (ADC)	XINT2	XINT1	Reserved	SEQ2INT (ADC)	SEQ1INT (ADC)
2	Reserved	Reserved	EPWM6_TZINT (ePWM6)	EPWM5_TZINT (ePWM5)	EPWM4_TZINT (ePWM4)	EPWM3_TZINT (ePWM3)	EPWM2_TZINT (ePWM2)	EPWM1_TZINT (ePWM1)
3	Reserved	Reserved	EPWM6_INT (ePWM6)	EPWM5_INT (ePWM5)	EPWM4_INT (ePWM4)	EPWM3_INT (ePWM3)	EPWM2_INT (ePWM2)	EPWM1_INT (ePWM1)
4	Reserved	Reserved	Reserved	Reserved	ECAP4_INT (eCAP4)	ECAP3_INT (eCAP3)	ECAP2_INT (eCAP2)	ECAP1_INT (eCAP1)
5	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	EQEP2_INT (eQEP2)	EQEP1_INT (eQEP1)
6	SPITXINTD (SPI-D)	SPIRXINTD (SPI-D)	SPITXINTC (SPI-C)	SPIRXINTC (SPI-C)	SPITXINTB (SPI-B)	SPIRXINTB (SPI-B)	SPITXINTA (SPI-A)	SPIRXINTA (SPI-A)
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
8	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	I2CINT1A (I2C-A)	I2CINT2A (I2C-A)
9	ECAN1INTB (CAN-B)	ECAN0INTB (CAN-B)	ECAN1INTA (CAN-A)	ECAN0INTA (CAN-A)	SCITXINTB (SCI-B)	SCIRXINTB (SCI-B)	SCITXINTA (SCI-A)	SCIRXINTA (SCI-A)
10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

C280x Hardware Interrupt

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

The preemption flag determines whether a given interrupt is preemptable or not. Preemption overrides prioritization, such that a preemptable task of higher priority can be preempted by a non-preemptable task of lower priority.

Dialog Box



CPU interrupt number(s)

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

See the table of C280x Peripheral Interrupt Vector Values on page 7-111 for a mapping of CPU interrupt number to interrupt names.

PIE interrupt number(s)

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

See the table of C280x Peripheral Interrupt Vector Values on page 7-111 for a mapping of CPU interrupt number to interrupt names.

Simulink task priority(s)

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

See the discussion of this block's "Vectorized Output" on page 7-109 for an explanation of task priorities.

Preemption flag(s)

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

See the discussion of this block's "Vectorized Output" on page 7-109 for an explanation of preemption flags.

Enable simulation input

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

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

References

Detailed information interrupt processing is in the *TMS320x280x DSP System Control and Interrupts Reference Guide*, Literature Number SPRU712B, available at the Texas Instruments Web site.

See Also

Idle Task

C281x ADC

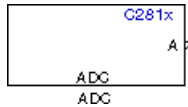
Purpose

Analog-to-digital converter (ADC)

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description



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

Triggering

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

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

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

Output

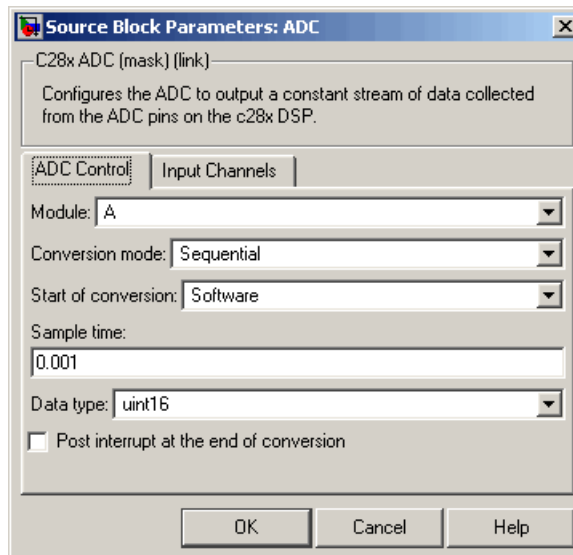
The output of the C281x ADC is a vector of uint16 values. The output values are in the range 0 to 4095 because the C281x ADC is 12-bit converter.

Dialog Box

Modes

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

ADC Control pane



Module

Specifies which DSP module to use:

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

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

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

Conversion mode

Type of sampling to use for the signals:

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

Start of conversion

Type of signal that triggers conversions to begin:

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

Sample time

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. See “Scheduling and Timing” on page 1-11 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 1-13 for a discussion of block placement and other necessary settings.

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

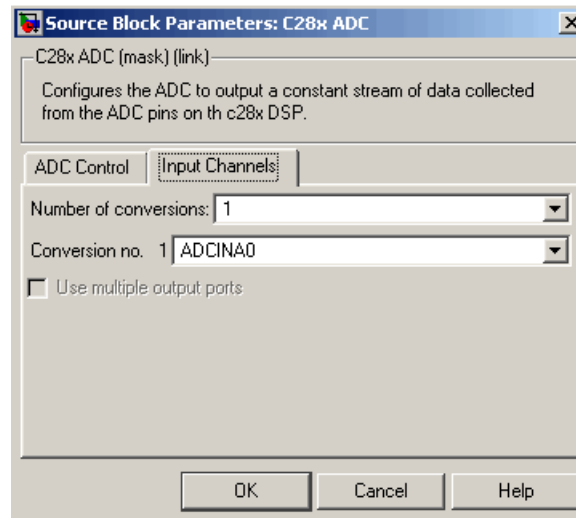
Data type

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

Post interrupt at the end of conversion

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

Input Channels pane



Number of conversions

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

Conversion no.

Specific ADC channel to associate with each conversion number.

In oversampling mode, a signal at a given ADC channel can be sampled multiple times during a single conversion sequence. To oversample, specify the same channel for more than one conversion. Converted samples are output as a single vector.

Use multiple output ports

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

See Also

C281x PWM, C281x Hardware Interrupt

Purpose Receive and log capture input pin transitions

Library c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description



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

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

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

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

Outputs

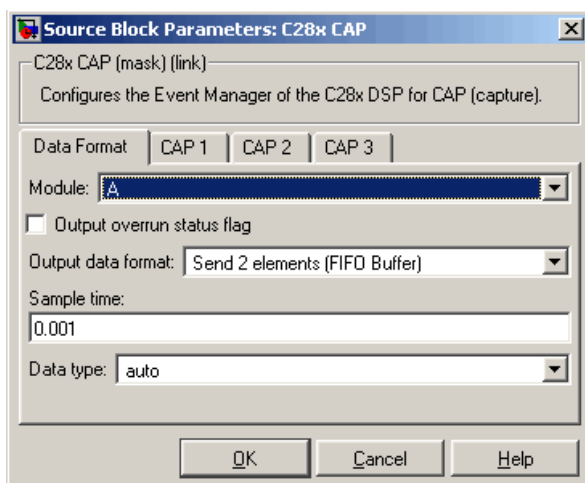
This block has up to two outputs: a cnt (count) output and an optional, FIFO status flag output. The cnt output increments each time a transition of the selected type occurs. The status flag outputs are

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

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

Dialog Box

Data Format pane



Module

Select the Event Manager (EV) module to use:

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

Output overrun status flag

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

Send data format

The type of data to output:

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

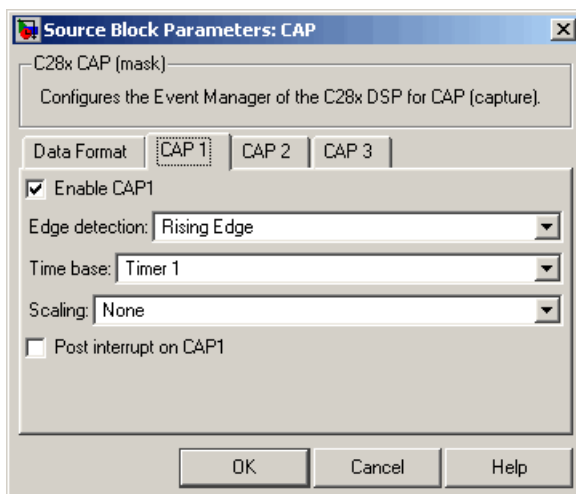
Sample time

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

Data type

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

CAP# pane



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

- **CAP1** controls CAP 1 or CAP 4, for EV module A or B, respectively.
- **CAP2** controls CAP 2 or CAP 5, for EV module A or B, respectively.
- **CAP3** controls CAP 3 or CAP 6, for EV module A or B, respectively.

Enable CAP#

Select to use the specified capture unit pin.

Edge Detection

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

Time Base

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

Scaling

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

Scaling	Resulting Rate (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 The above rates assume a 75 MHz input clock.

Post interrupt on CAP#

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

See Also

C281x Hardware Interrupt

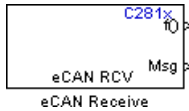
Purpose

Enhanced Control Area Network receive mailbox

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

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

Source Block Parameters: eCAN Receive

C28x eCAN Receive (mask)

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

Parameters

Mailbox number:

Message identifier:

Message type:

Sample time:

Data type:

Output message length

Post interrupt when message is received

OK Cancel Help

Mailbox number

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

Message identifier

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

Message type

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

Sample time

Frequency with which the mailbox is polled to determine if a new message has been received. A new message causes a function call to be emitted from the mailbox. If you want to update the message output only when a new message arrives, then the block needs to be executed asynchronously. To execute this block asynchronously, set **Sample Time** to -1, check the **Post interrupt when message is received** box, and refer to “Asynchronous Interrupt Processing” on page 1-13 for a discussion of block placement and other necessary settings.

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

Data type

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

For uint16 data,

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

For uint32 data,

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

C281x eCAN Receive

For example, if the received message has two bytes:

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

then the uint16 output would be:

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

Output message length

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

Post interrupt when message is received

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

References

Detailed information on the eCAN module is in the *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

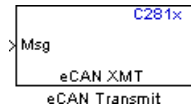
Purpose

Enhanced Control Area Network transmit mailbox

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

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.

Data Vectors

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

For input of type `uint32`,

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

the data buffer is:

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

For input of type `uint16`,

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

the data buffer is:

C281x eCAN Transmit

```
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
```

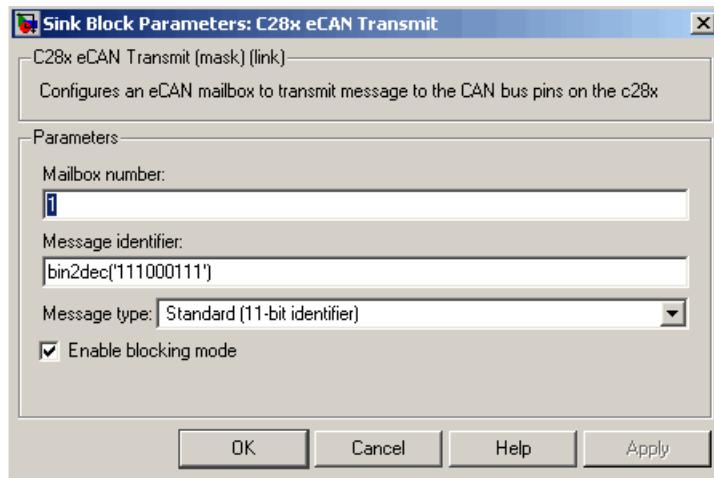
For input of type `uint16[2]`, which is a two-element vector,

```
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).

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.

C281x eCAN Transmit

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

References

Detailed information on the eCAN module is in the *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

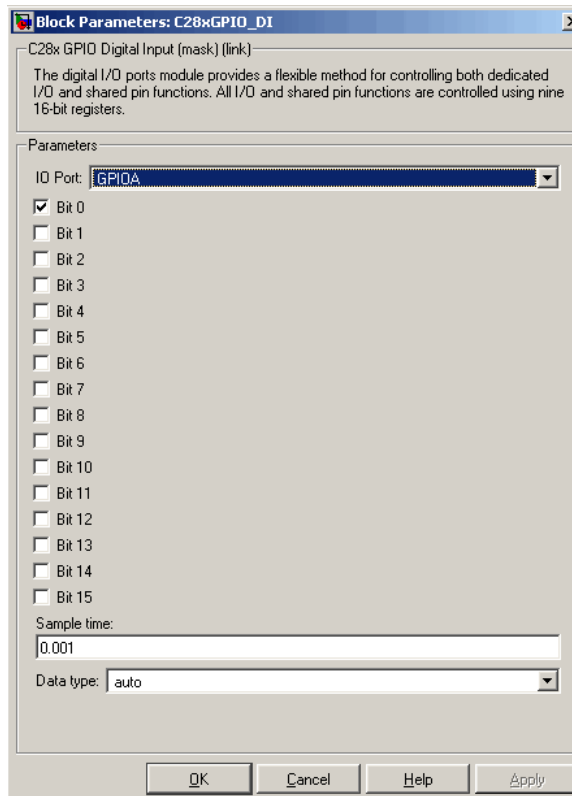
c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description



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

Dialog Box



C281x GPIO Digital Input

IO Port

Select the input/output port to use: GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, or GPIOG and select the I/O Port bits to enable for digital input. (Note that there is no GPIOC port on the C281x.) If you select multiple bits, vector input is expected. Unselected 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

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.

See Also

C281x GPIO Digital Output

C281x GPIO Digital Output

Purpose General-purpose I/O pins for digital output

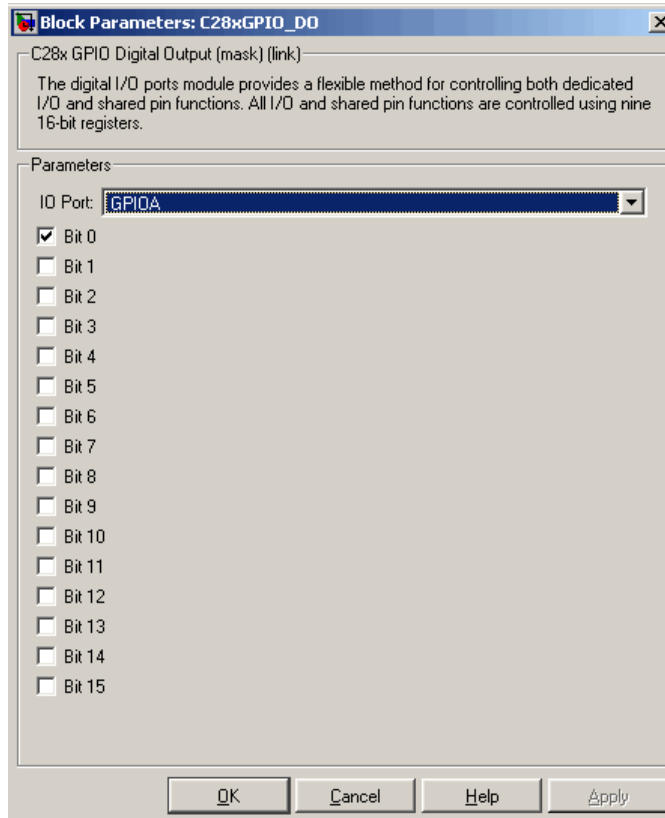
Library c281xdspchip1ib in Embedded Target for TI C2000 DSP

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.

Dialog Box



IO Port

Select the input/output port to use: GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, or GPIOG and select the I/O Port bits to enable for digital input. (Note that there is no GPIOC port on the C281x.) If you select multiple bits, vector input is expected. Unselected bits are available for peripheral functionality. Note that 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

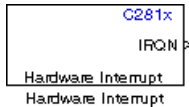
Purpose

Create Interrupt Service Routine to handle hardware interrupts

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description



For many systems, an execution scheduling model based on a timer interrupt is not sufficient to ensure a real-time response to external events. The C281x Hardware Interrupt block addresses this problem by allowing for the asynchronous processing of interrupts triggered by events managed by other blocks in the C281x DSP Chip Support Library.

The C281x blocks that can generate an interrupt for asynchronous processing are

- C281x ADC
- C281x CAP
- C281x eCAN Receive
- C281x Timer
- C281x SCI Receive
- C281x SCI Transmit
- C281x SPI Receive
- C281x SPI Transmit

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

Vectorized Output

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

C281x Hardware Interrupt

Each interrupt is described by:

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

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

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

C281x Peripheral Interrupt Vector Values

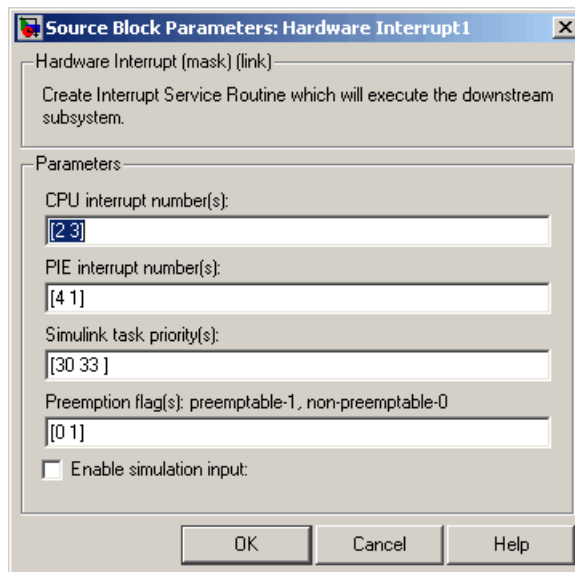
Row numbers = CPU values / Column numbers = PIE values								
	8	7	6	5	4	3	2	1
1	WAKEINT (LPM/WD)	TINT0 (TIMER 0)	ADCINT (ADC)	XINT2	XINT1	Reserved	PDPINTB (EV-B)	PDPINTA (EV-A)
2	Reserved	T1OFINT (EV-A)	T1UFINT (EV-A)	T1CINT (EV-A)	T1PINT (EV-A)	CMP3INT (EV-A)	CMP2INT (EV-A)	CMP1INT (EV-A)
3	Reserved	CAPINT3 (EV-A)	CAPINT2 (EV-A)	CAPINT1 (EV-A)	T2OFINT (EV-A)	T2UFINT (EV-A)	T2CINT (EV-A)	T2PINT (EV-A)
4	Reserved	T3OFINT (EV-B)	T3UFINT (EV-B)	T3CINT (EV-B)	T3PINT (EV-B)	CMP6INT (EV-B)	CMP5INT (EV-B)	CMP4INT (EV-B)
5	Reserved	CAPINT6 (EV-B)	CAPINT5 (EV-B)	CAPINT4 (EV-B)	T4OFINT (EV-B)	T4UFINT (EV-B)	T4CINT (EV-B)	T4PINT (EV-B)
6	Reserved	Reserved	MXINT (McBSP)	MRINT (McBSP)	Reserved	Reserved	SPITXINTA (SPI)	SPIRXINTA (SPI)
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
8	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
9	Reserved	Reserved	ECAN1INT (CAN)	ECAN0INT (CAN)	SCITXINTB (SCI-B)	SCIRXINTB (SCI-B)	SCITXINTA (SCI-A)	SCIRXINTA (SCI-A)
10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

C281x Hardware Interrupt

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

The preemption flag determines whether a given interrupt is preemptable or not. Preemption overrides prioritization, such that a preemptable task of higher priority can be preempted by a non-preemptable task of lower priority.

Dialog Box



CPU interrupt number(s)

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

See the table of C281x Peripheral Interrupt Vector Values on page 7-141 for a mapping of CPU interrupt number to interrupt names.

PIE interrupt number(s)

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

See the table of C281x Peripheral Interrupt Vector Values on page 7-141 for a mapping of CPU interrupt number to interrupt names.

Simulink task priority(s)

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

See the discussion of this block's "Vectorized Output" on page 7-139 for an explanation of task priorities.

Preemption flag(s)

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

See the discussion of this block's "Vectorized Output" on page 7-139 for an explanation of preemption flags.

Enable simulation input

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

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

References

Detailed information interrupt processing is in the *TMS320x281x DSP System Control and Interrupts Reference Guide*, Literature Number SPRU078C, available at the Texas Instruments Web site.

See Also

C281x Timer, Idle Task

C281x PWM

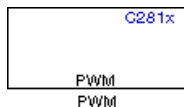
Purpose

Pulse wave modulators (PWMs)

Library

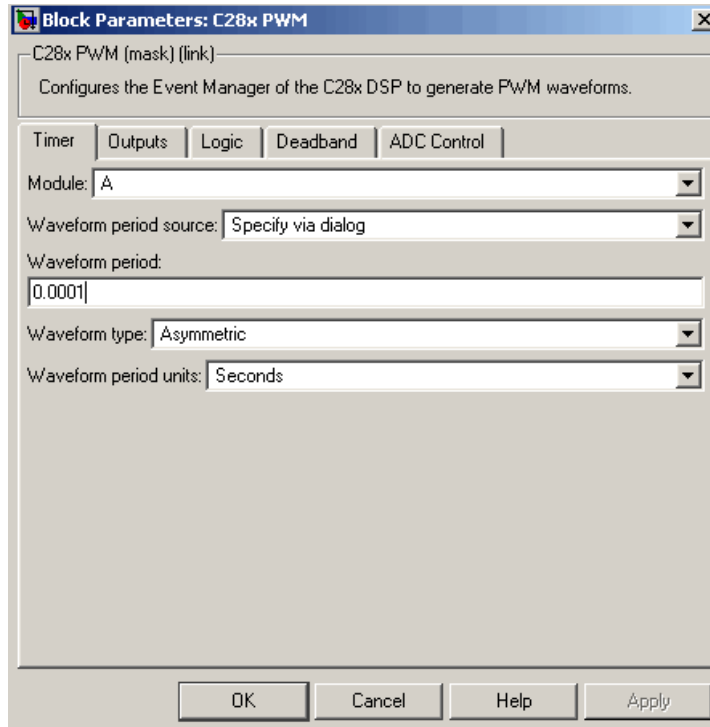
c281xdspchip1ib in Embedded Target for TI C2000 DSP

Description



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

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

**Dialog
Box****Timer pane****Module**

Specifies which target PWM pairs to use:

- A — Displays the PWMs in module A (PWM1/PWM2, PWM3/PWM4, and PWM5/PWM6).
- B — Displays the PWMs in module B (PWM7/PWM8, PWM9/PWM10, and PWM11/PWM12).

Note PWMs in module A use Event Manager A, Timer 1, and PWMs in module B use Event Manager B, Timer 3.

Waveform period source

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

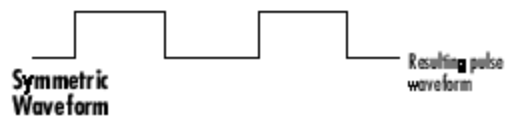
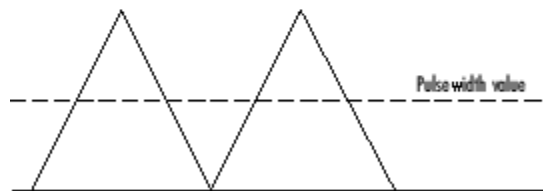
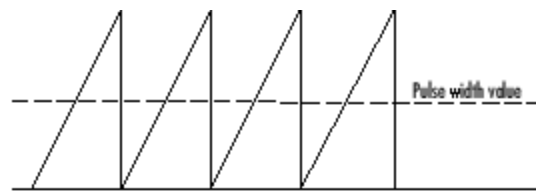
Waveform period

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

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

Waveform type

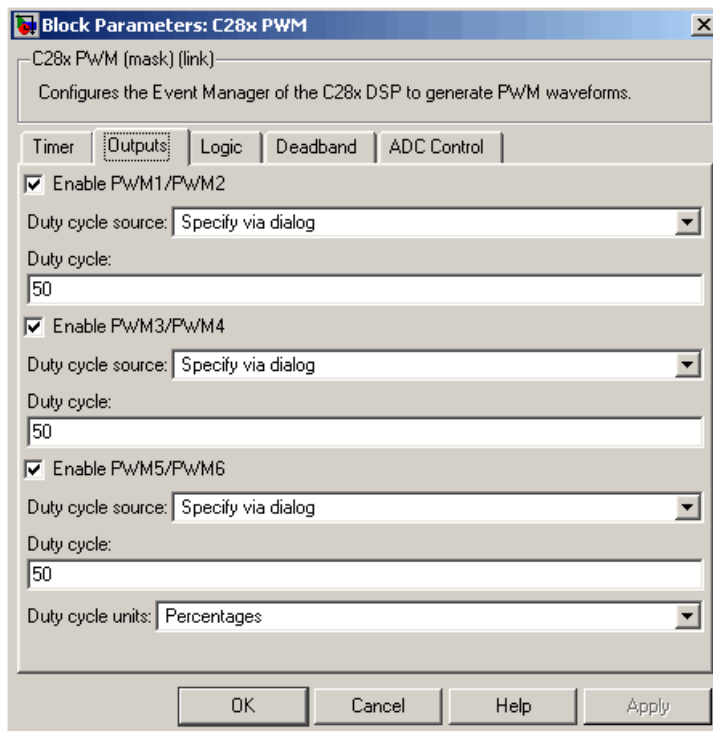
Type of waveform to be generated by the PWM pair. The F2812 PWMs can generate two types of waveforms: `Asymmetric` and `Symmetric`. The following illustration shows the difference between the two types of waveforms.



Waveform period units

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

Outputs pane



Enable PWM#/PWM#

Check to activate the PWM pair. PWM1/PWM2 are activated via the Output 1 pane, PWM3/PWM4 are on Output 2, and PWM5/PWM6 are on Output 3.

Duty cycle source

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

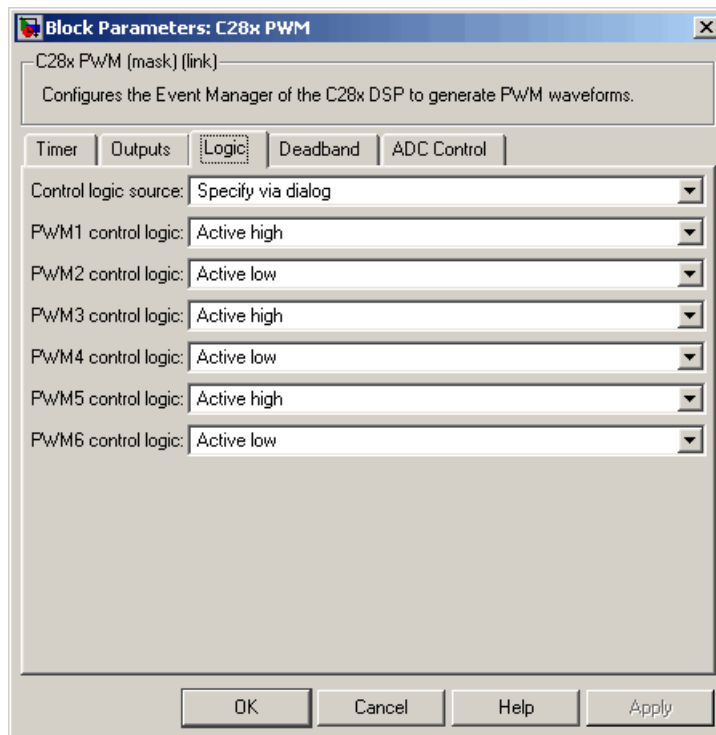
Duty cycle

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

Duty cycle units

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

Logic pane



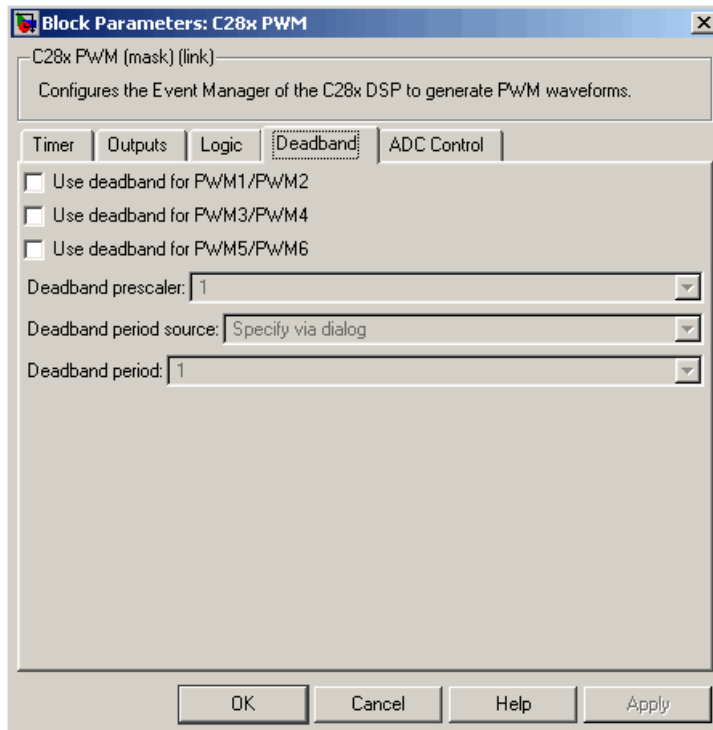
Control logic source

Source from which the control logic is obtained for all PWMs. Select Specify via dialog to enter the values in the **PWM# control logic** fields or select Input port to use values from the input port.

PWM# control logic

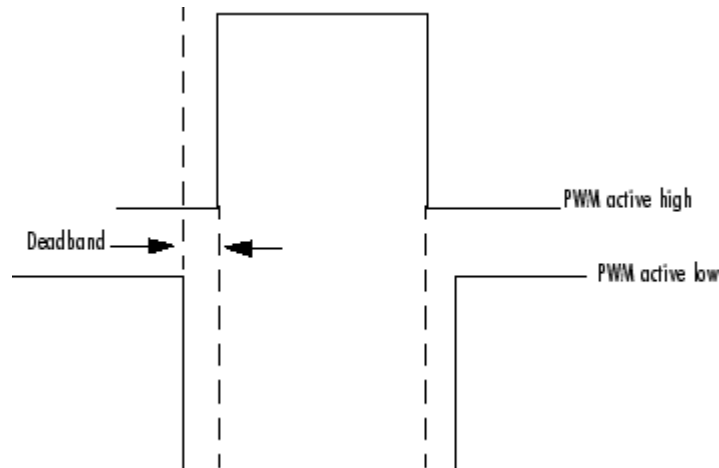
Control logic trigger for the PWM. Forced high causes the pulse value to be high. Active high causes the pulse value to go from low to high and Active low causes the pulse value to go from high to low. Forced low causes the pulse value to be low.

Deadband pane



Use deadband for PWM#/PWM#

Enables a deadband area of no signal overlap at the beginning of particular PWM pair signals. The following figure shows the deadband area.



Deadband prescaler

Number of clock cycles, which, when multiplied by the Deadband period, determines the size of the deadband. Selectable values are 1, 2, 4, 8, 16, and 32.

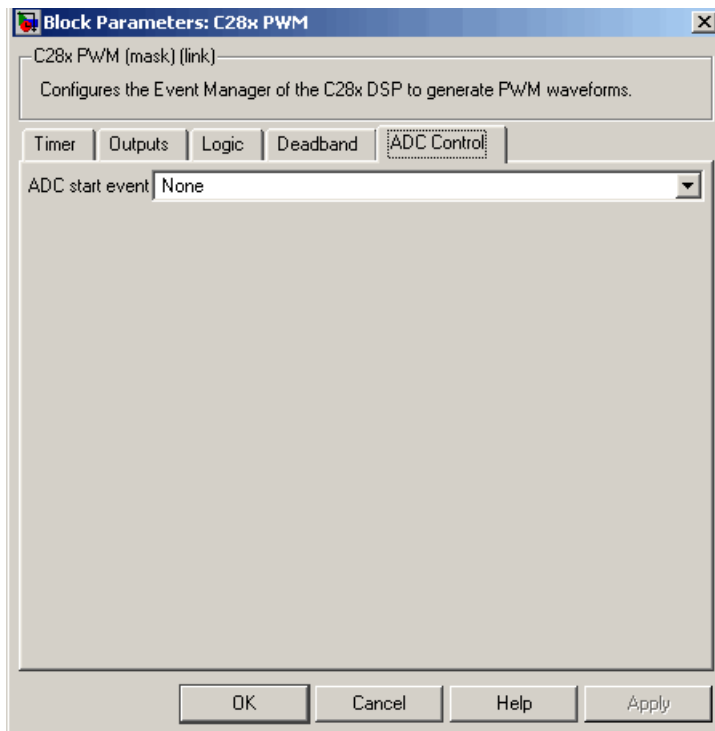
Deadband period source

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

Deadband period

Value that, when multiplied by the Deadband prescaler, determines the size of the deadband. Selectable values are from 1 to 15.

ADC Control pane



ADC start event

Controls whether this PWM and ADC associated with the same EV module are synchronized. Select None for no synchronization or select an interrupt to generate the source start-of-conversion (SOC) signal for the associated ADC.

- None — The ADC and PWM are not synchronized. The EV does not generate an SOC signal and the ADC is triggered by software (that is, the A/D conversion occurs when the ADC block is executed in the software).

- Underflow interrupt — The EV generates an SOC signal for the ADC associated with the same EV module when the board's General Purpose (GP) timer counter reaches a hexadecimal value of FFFFh.
- Period interrupt — The EV generates an SOC signal for the ADC associated with the same EV module when the value in GP timer matches the value in the period register. The value set in **Waveform period** above determines the value in the register.

Note If you select Period interrupt and specify a sampling time less than the specified (**Waveform period**)/(Event timer clock speed), zero-order hold interpolation will occur. (For example, if you enter 64000 as the waveform period, the period for the timer is $64000/75 \text{ MHz} = 8.5333\text{e-}004$. If you enter a **Sample time** in the C281x ADC dialog box that is less than this result, it will cause zero-order hold interpolation.)

- Compare interrupt — The EV generates an SOC signal for the ADC associated with the same EV module when the value in the GP timer matches the value in the compare register. The value set in **Duty cycle** above determines the value in the register.

See Also

C281x ADC

C281x QEP

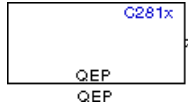
Purpose

Quadrature encoder pulse circuit

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description

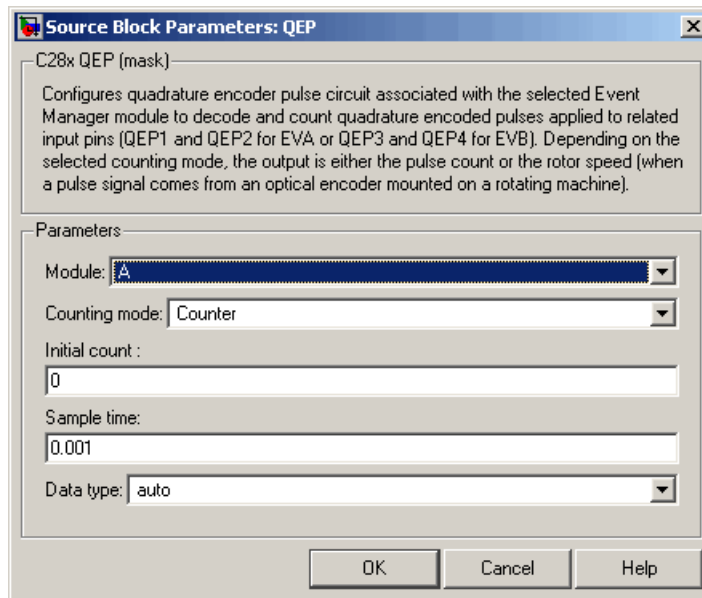


Each F2812 Event Manager has three capture units, which can log transitions on its capture unit pins. Event Manager A (EVA) uses capture units 1, 2, and 3. Event Manager B (EVB) uses capture units 4, 5, and 6.

The quadrature encoder pulse (QEP) circuit decodes and counts quadrature encoded input pulses on these capture unit pins. QEP pulses are two sequences of pulses with varying frequency and a fixed phase shift of 90 degrees (or one-quarter of a period). Both edges of the QEP pulses are counted so the frequency of the QEP clock is four times the input sequence frequency.

The QEP, in combination with an optical encoder, is particularly useful for obtaining speed and position information from a rotating machine. Logic in the QEP circuit determines the direction of rotation by which sequence is leading. For module A, if the QEP1 sequence leads, the general-purpose (GP) Timer counts up and if the QEP2 sequence leads, the timer counts down. The pulse count and frequency determine the angular position and speed.

Dialog Box



Module

Specifies which QEP pins to use:

- A — Uses QEP1 and QEP2 pins.
- B — Uses QEP3 and QEP4 pins.

Counting mode

Specifies how to count the QEP pulses:

- Counter — Count the pulses based on the board's GP Timer 2 (or GP Timer 4 for EVB).
- RPM — Count the machine's revolutions per minute.

Positive rotation

Defines whether to use Clockwise or Counterclockwise as the direction to use as positive rotation. This field appears only if you select RPM above.

Encoder resolution

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

Initial count

Initial value for the counter. The default is 0.

Sample time

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

Data type

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

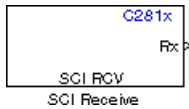
Purpose

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

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description



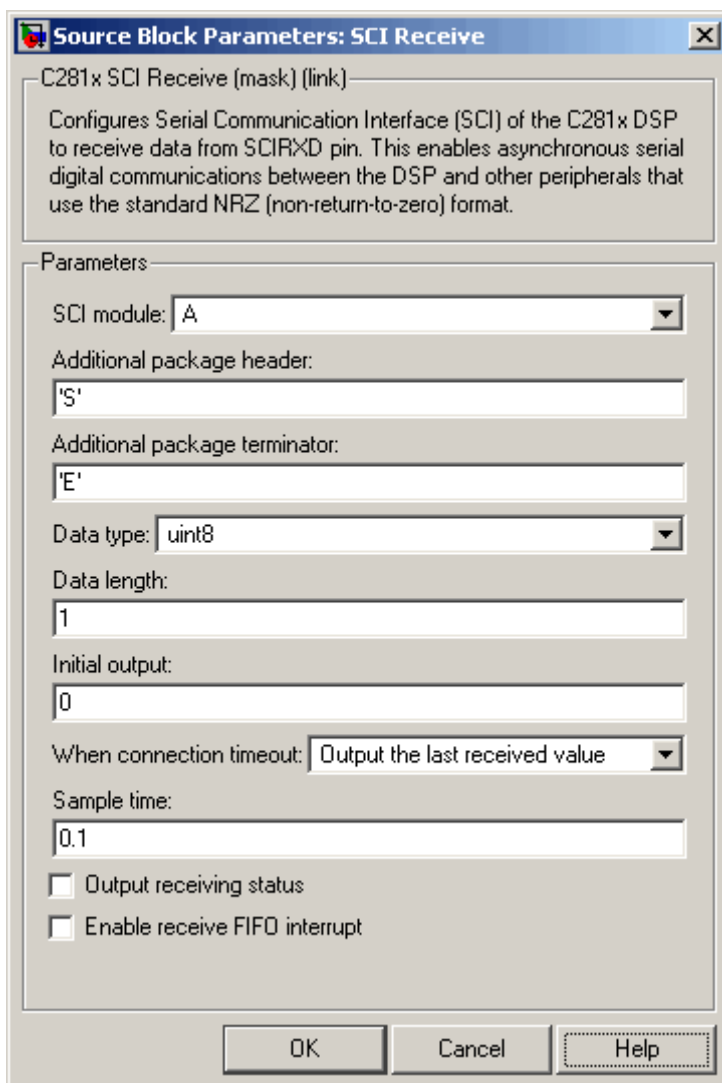
The C281x SCI Receive block supports asynchronous serial digital communications between the target and other asynchronous peripherals in nonreturn-to-zero (NRZ) format. This block configures the C281x DSP target to receive scalar or vector data from the COM port via the C28x target's COM port.

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

Many SCI-specific settings are in the **DSPBoard** section of the F2812 eZdsp target preferences block. You should verify that these settings are correct for your application.

C281x SCI Receive

Dialog Box



SCI module

SCI module to be used for communications.

Additional package header

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

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

Additional package terminator

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

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

Data type

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

Data length

How many of **Data type** the block will receive (not bytes). Anything more than 1 is a vector. The data length is inherited

C281x SCI Receive

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

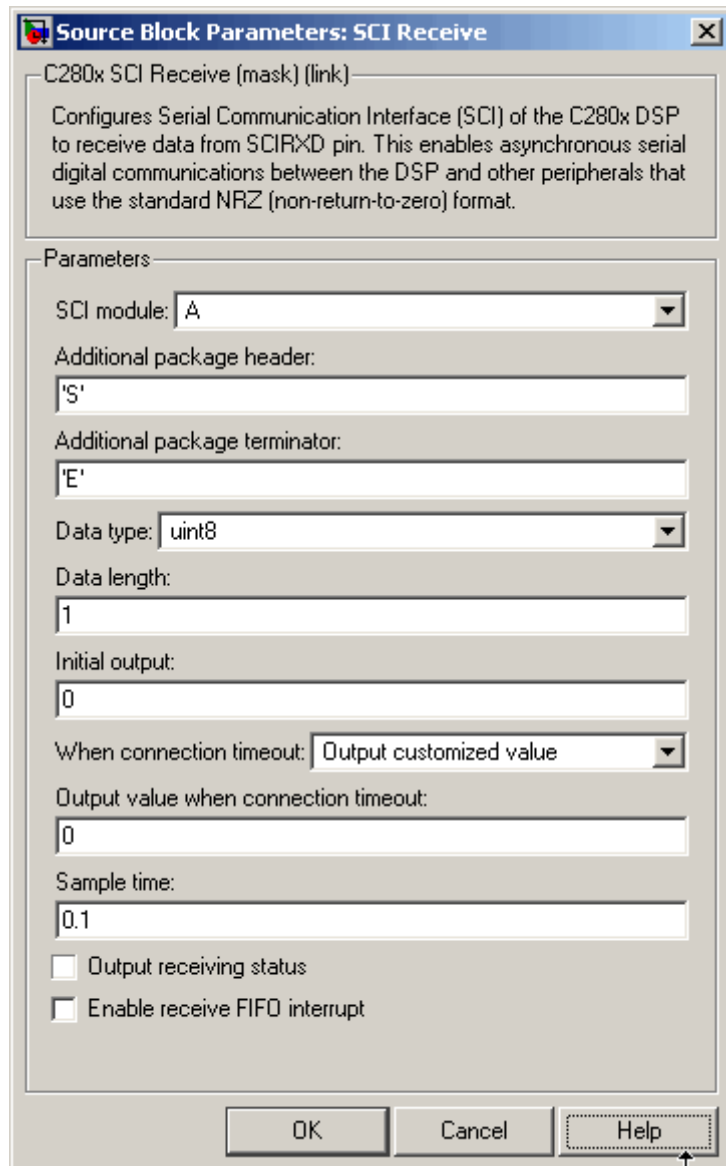
Initial output

Default value from the c281x SCI Receive block. This value is used, for example, if a connection time-out occurs and the **When connection timeout** field is set to “Output the last received value”, but nothing yet has been received.

When connection timeout

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

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



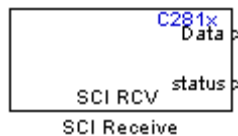
C281x SCI Receive

Sample time

Sample time, T_s , for the block's input sampling. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-13 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).

References

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

See Also

C281x SCI Transmit, C281x Hardware Interrupt

C281x SCI Transmit

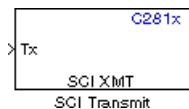
Purpose

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

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description

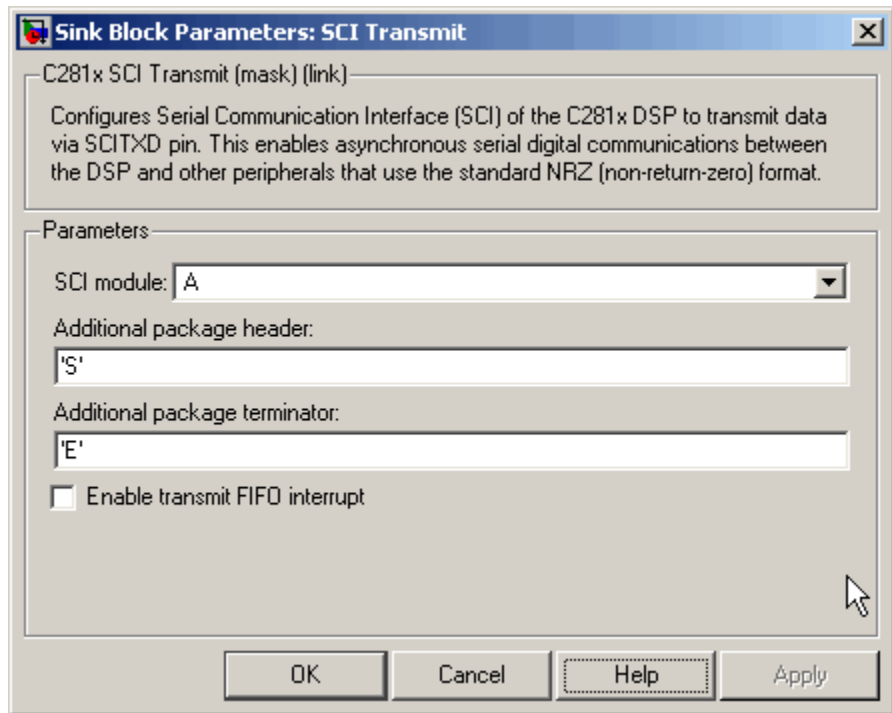


The C281x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C281x target's COM ports in nonreturn-to-zero (NRZ) format. You can specify how many of the six target COM ports to use. The sampling rate and data type are inherited from the input port. The data type of 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.

Dialog Box



SCI module

SCI module to be used for communications.

Additional package header

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

C281x SCI Transmit

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

Additional package terminator

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

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

Enable transmit FIFO interrupt

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

References

Detailed information on the SCI module is in the *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

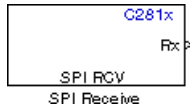
Purpose

Receive data via serial peripheral interface (SPI) on target

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description



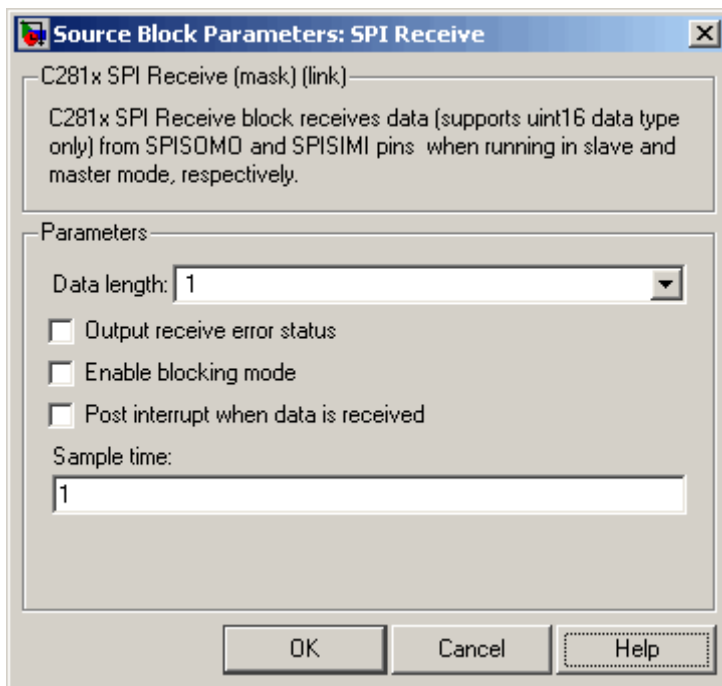
The C281x SPI Receive supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. 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.

Note 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.

Many SPI-specific settings are in the **DSPBoard** section of the F2812 eZdsp target preferences block. You should verify that these settings are correct for your application.

C281x SPI Receive

Dialog Box



Data length

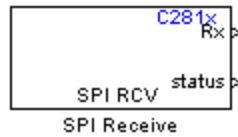
Specifies how many uint16s are expected to be received. Select 1 through 16.

Enable blocking mode

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

Output receive error status

When this field is checked, the c281x SPI Receive block adds another output port for the transaction status, and appears as shown in the following figure.



Error status may be one of the following values:

- 0: No errors
- 1: Data loss occurred (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 1-13 for a discussion of block placement and other necessary settings.

See Also

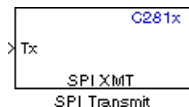
C281x SPI Transmit, C281x Hardware Interrupt

C281x SPI Transmit

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

Library c281xdspchip1lib in Embedded Target for TI C2000 DSP

Description



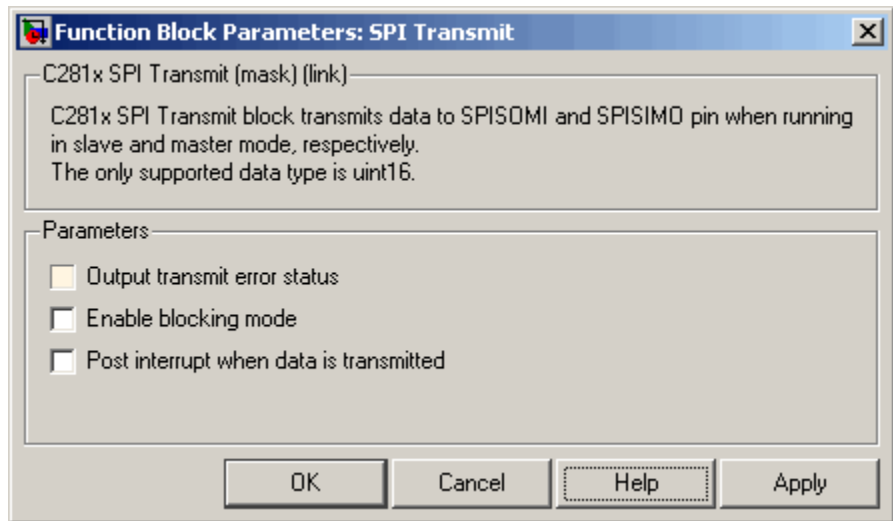
The C281x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data and the SPISOMI pin receives data. 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 and data type are inherited from the input port. If no data type is specified, the default data type is uint16.

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

Many SPI-specific settings are in the **DSPBoard** section of the F2812 eZdsp target preferences block. You should verify that these settings are correct for your application.

Dialog Box



Output transmit error status

When this field is checked, the c281x SPI Transmit block adds another output port for the transaction status, and appears as shown in the following figure.



Error status may be one of the following values:

- 0: No errors
- 1: A time-out occurred while the block was transmitting data
- 2: There is an error in the transmitted data (for example, header or terminator don't match, length of data expected is too big or too small)

C281x SPI Transmit

Enable blocking mode

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

Post interrupt when data is transmitted

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

See Also

C281x SPI Receive

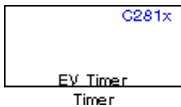
Purpose

Configure up to four general-purpose, stand alone Event Manager timers

Library

c281xdspchip1lib in Embedded Target for TI C2000 DSP

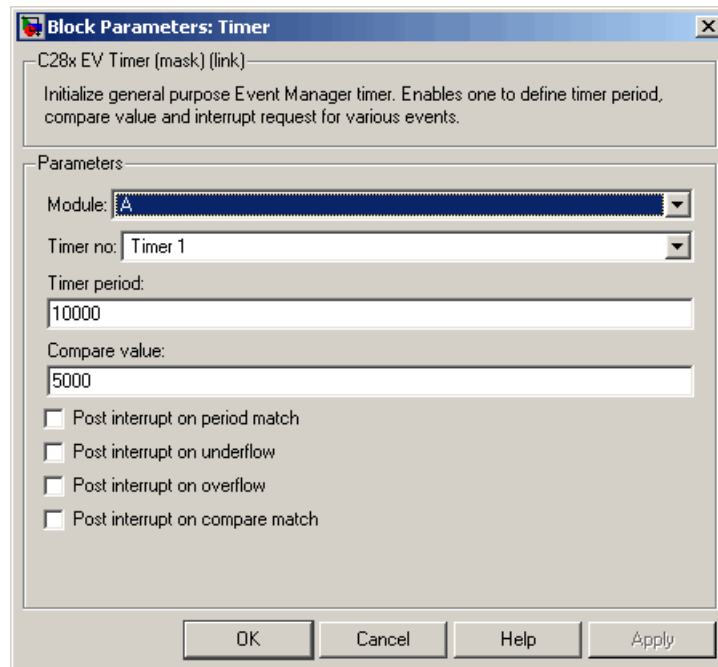
Description



The C281x event-manager (EV) modules include general-purpose (GP) timers. There are two general-purpose (GP) timers in each module. These timers can be used as independent time bases in various applications.

The C281x Timer block lets you set the periodicity of the general-purpose timers, and configure them to post interrupts under specified conditions.

Dialog Box



Module

Timer no

Select which of four possible timers to configure. Setting **Module** to A lets you select Timer 1 or Timer 2 in **Timer no**. Setting **Module** to B lets you select Timer 3 or Timer 4 in **Timer no**.

Timer period

Set the length of the timer period in clock cycles. Enter a value from 0 to 65535. The default is 10000.

You can easily calculate how many clock cycles to set for the timer period if you know the length of a clock cycle. The calculation for the length of one clock cycle is as follows:

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

where the System clock frequency of 150MHz is divided by the high speed clock prescaler of 2, and then divided by the timer control input clock prescaler, which is 128. The resulting frequency is .586MHz. Thus, one clock cycle is 1/.586MHz, which is 1.706 μ s.

Compare value

Enter a constant value to be used for comparison to the running timer value for the purpose of generating interrupts. Enter a value from 0 to 65535. The default is 5000. Note that interrupts will be generated only if **Post interrupt on compare match** is selected.

Post interrupt on period match

Select this check box to generate an interrupt whenever the value of the timer reaches its maximum value as specified in **Timer period**.

Post interrupt on underflow

Select this check box to generate an interrupt whenever the value of the timer cycles back to 0.

Post interrupt on overflow

Select this check box to generate an interrupt whenever the value of the timer reaches its maximum possible value of 65535. Note

that unless **Timer period** is set to 65535, this interrupt will never be generated even if this check box is selected.

Post interrupt on compare match

Select this check box to generate an interrupt whenever the value of the timer equals **Compare value**.

See Also

C281x Hardware Interrupt, Idle Task

Clarke Transformation

Purpose

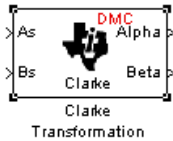
Convert balanced three-phase quantities to balanced two-phase quadrature quantities

Library

c28xdmclib in Embedded Target for TI C2000 DSP

Description

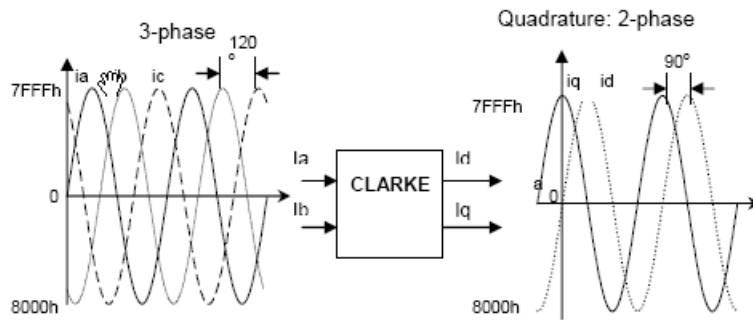
This block converts balanced three-phase quantities into balanced two-phase quadrature quantities. The transformation implements these equations



$$I_d = I_a$$

$$I_q = (2I_b + I_a) / \sqrt{3}$$

and is illustrated in the following figure.



The inputs to this block are the phase a (As) and phase b (Bs) components of the balanced three-phase quantities and the outputs are the direct axis (Alpha) component and the quadrature axis (Beta) of the transformed signal.

The instantaneous outputs are defined by the following equations and are shown in the following figure:

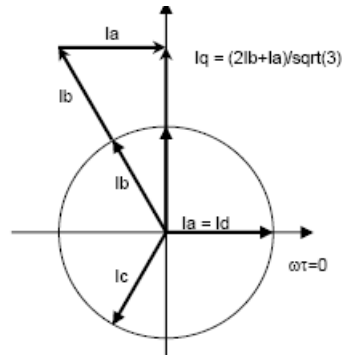
$$i_a = I * \sin(\omega t)$$

$$i_b = I * \sin(\omega t + 2\pi/3)$$

$$i_c = I * \sin(\omega t - 2\pi/3)$$

$$i_d = I * \sin(\omega t)$$

$$i_q = I * \sin(\omega t + \pi/2)$$



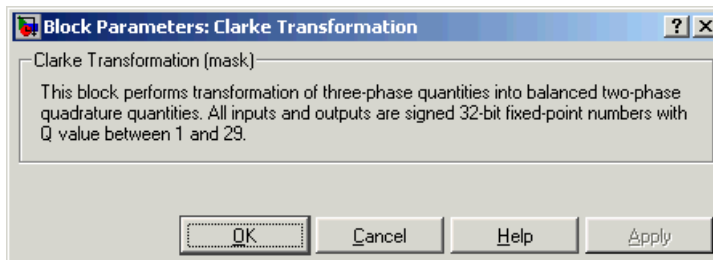
The variables used in the preceding equations and figures correspond to the variables on the block as shown in the following table:

	Equation Variables	Block Variables
Inputs	ia	As
	ib	Bs
Outputs	id	Alpha
	iq	Beta

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

Clarke Transformation

Dialog Box



References

Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

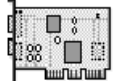
Inverse Park Transformation, Park Transformation, PID Controller, Space Vector Generator, Speed Measurement

Purpose

Target preferences for custom C280x board

Library

c2000tgtpref1lib in Embedded Target for TI C2000 DSP

Description

Custom C280x
Board

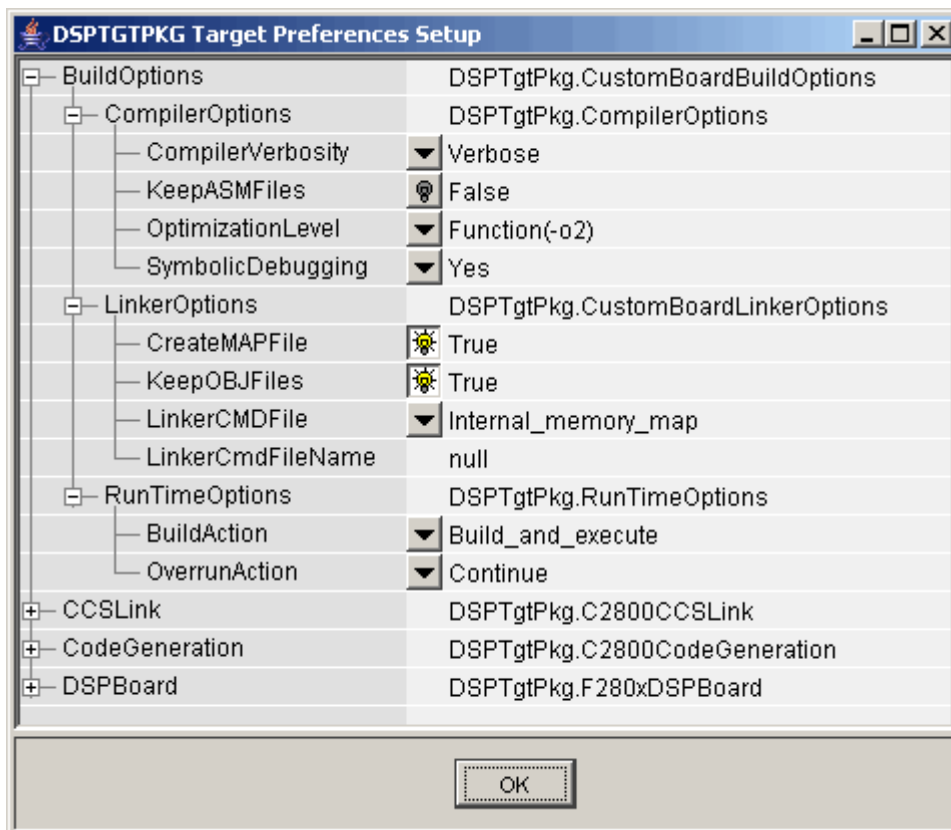
Options on the block mask let you set features of code generation for your custom board based on a F2801, F2802, F2806, F2808, or F2809 chip. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

Note This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model.

Custom C280x Board

Dialog Box

BuildOptions



BuildOptions – CompilerOptions

Compiler Verbosity

Amount of information the compiler returns while it runs. Options are

- Verbose — Returns all compiler messages.
- Quiet — Suppresses compiler progress messages.

- `Super_quiet` — Suppresses all compiler messages.

KeepASMFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your assembly language (.asm) files after creation. The default is `False`—.asm files are not kept in your current directory. If you choose to keep the .asm files, set this option to `True`.

OptimizationLevel

Degree of optimization provided by the TI optimizing compiler to apply to files in your project. For details about the compiler options, refer to your CCS documentation. When you create new projects, the Embedded Target for TI C2000 DSP sets the optimization to `Function(-o2)`.

SymbolicDebugging

Whether to generate symbolic debugging directives that the C source-level debugger uses and whether to enable assembly source debugging. By default, this option is `Yes`—symbolic debugging is enabled.

BuildOptions – LinkerOptions

CreateMAPFile

Whether the linker produces a map of the input and output sections, including null areas, and places the listing in a file in your current directory with the name `modelName.map`. The default is `True`—the listing is produced.

KeepOBJFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your object (.obj) files after creation. The linker uses object files to generate a single executable common object file format (COFF) file that you run on the target DSP. The object files are saved to your current project directory. Saving your .obj files can speed up the compile process by not having to recompile files that you have not changed. The default is `True`—the .obj files are retained.

LinkerCMDFile

Type of linker command file to use when the linker runs. Linker command files contain linker or hex conversion utility options and the names of input files to the linker or hex conversion utility.

Linker command file types are

- `Internal_memory_map` — Uses the small memory model on the target, which requires that all sections of the code and data fit into the internal memory available only on the C280x chip (minus the flash memory).

If you select `Internal_memory_map`, but your data or program requires far calls, the TI compiler returns an error message in the CCS IDE indicating that your data does not fit in internal memory or that your code does not fit in internal memory. The error message looks like one of the following:

```
error: can't allocate '.far'  
error: can't allocate '.text'
```

Note When you use `Internal_memory_map`, specifying a **LinkerCmdFileName** has no effect.

- `Flash_memory_map` — Uses flash memory, in which case your data can use the full storage capacity of the available flash memory. When you are using flash memory, you cannot set the **BuildAction** preference in the **BuildOptions — RunTimeOptions** section to `Build_and_execute`. For more information on using Flash memory, see “Creating Stand-Alone Applications by Saving Code into Flash Memory”.
- `Custom_file` — Uses the file in the **LinkerCmdFileName** field. This option lets you target custom boards. You must specify the full path of the file. The software does not verify that the commands in this file are correct.

BuildOptions – RunTimeOptions

BuildAction

Action taken by Real-Time Workshop when you click **Build** or press **Ctrl+B** in the Simulation Parameters dialog box. The order in which the actions are presented is significant — each listed action does what the previous action in the list does, and adds new features of its own:

- **Generate_code_only** — Directs Real-Time Workshop to generate C code only from the model. It does not use the TI software tools, such as the compiler and linker, and you do not need to have CCS installed. Also, MATLAB does not create the handle to CCS that results from the other options.

The build process for a model also generates the files `modelName.c`, `modelName.cmd`, `modelName.bld`, and many others. It puts the files in a build directory named `modelName_C2000_rtw` in your MATLAB working directory. This file set contains many of the same files that Real-Time Workshop generates to populate a CCS project when you choose **Create_CCS_Project** for the build action.

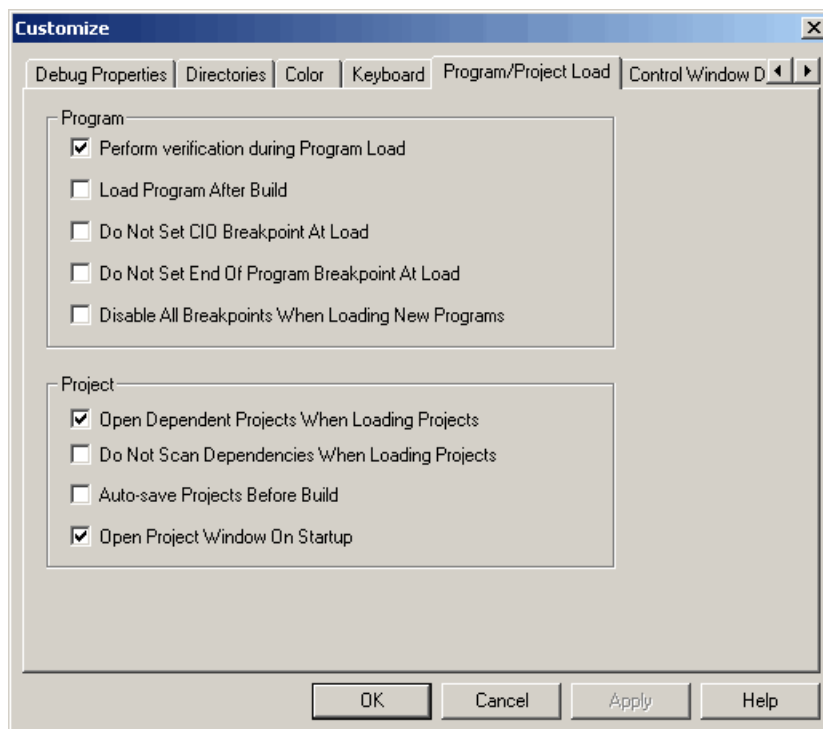
- **Create_CCS_Project** — Directs Real-Time Workshop to start CCS and populate a new project with the files from the build process. Selecting this setting enables the CCS board number option so you can select which installed board to target. This option offers a convenient way to build projects in CCS.
- **Build** — Builds the executable COFF file, but does not download the file to the target.
- **Build_and_execute** — Directs Real-Time Workshop to download and run your generated code as an executable on your target. This option is the default.

Setting this option requires corresponding options to be set in the CCS:

- From the CCS window, select **Option > Customize...** .

Custom C280x Board

- b** Select the **Program/Project Load** tab on the **Customize** dialog box, as shown in the following figure.



- c** Uncheck the **Load Program After Build** option.

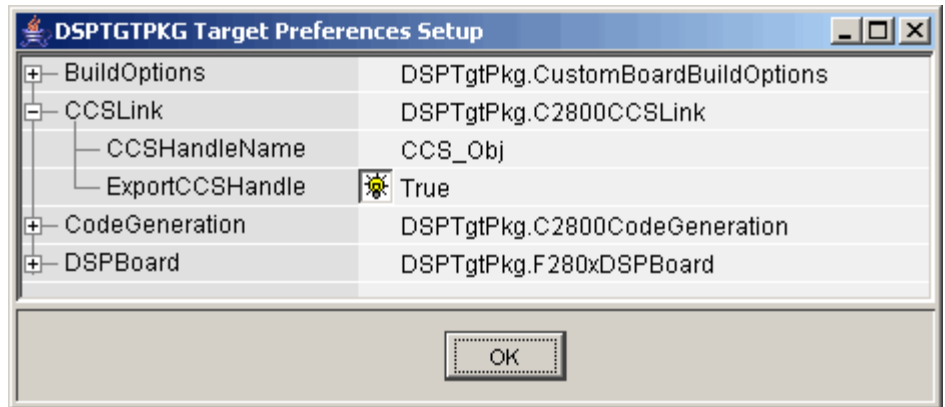
Note When you build and execute a model on your target, Real-Time Workshop resets the target automatically. You do not need to reset the board before building models.

OverrunAction

Defines the action to take when an interrupt overrun occurs:

- Continue — Ignore overruns encountered while running the model. This option is the default.
- Halt — Stop program execution.

CCSLink



CCSHandleName

Name of the CCS handle. Click in the edit box to change the name. When you use Real-Time Workshop to build a model for a C2000 target, Embedded Target for TI C2000 DSP makes a link between MATLAB and CCS. If you have used the link portion of the Embedded Target for TI C2000 DSP, you are familiar with function `ccdsp`, which creates links between the IDE and MATLAB. This option refers to the same link, called `cc` in the function reference pages. Although MATLAB to CCS is a link, it is actually a handle to an object that contains information about the object, such as the target board and processor it accesses.

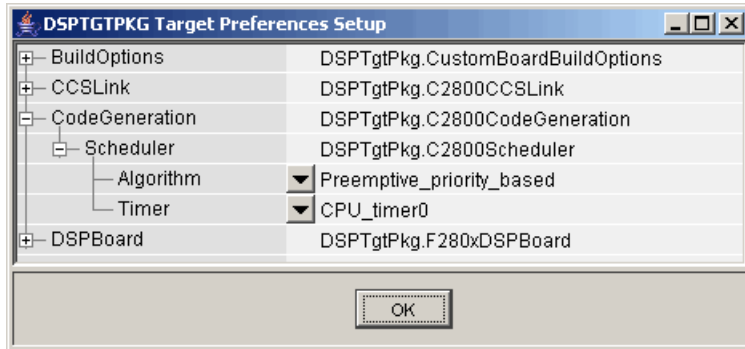
ExportCCSHandle

Whether to export the CCS handle to your MATLAB workspace, giving it the name you assigned in **CCSHandleName**. If this option is set to True (the default), then after you build your model,

Custom C280x Board

you see the CCS object in your MATLAB workspace browser with the name you provided and class type `ccdsp`.

CodeGeneration – Scheduler



Algorithm

Algorithm to use for scheduling. Currently, the only algorithm option is `Preemptive_priority_based`. This scheduler runs based on the timer interrupt. The timer period is set based on the base rate sample time you specify for your model. This algorithm supports multirate systems in multitasking mode with priority-based preemption. The task for the fastest group (the base rate task) runs first and other tasks run in the order determined by their sample rates from faster tasks to slower tasks. For more information, see “Models with Multiple Sample Rates” in the “Real-Time Workshop” documentation.

Timer

CPU timer to use for scheduling.

DSPBoard

+ BuildOptions	DSPTgtPkg.CustomBoardBuildOptions
+ CCSLink	DSPTgtPkg.C2800CCSLink
+ CodeGeneration	DSPTgtPkg.C2800CodeGeneration
- DSPBoard	DSPTgtPkg.F280xDSPBoard
- DSPBoardLabel	<Enter your board name>
- DSPChip	DSPTgtPkg.C280xDSPChip
+ ADC	DSPTgtPkg.C280xADC
- DSPChipLabel	▼ TI TMS320F2808
+ SCI_A	DSPTgtPkg.C2800SCI
+ SCI_B	DSPTgtPkg.C2808SCIB
+ SPI_A	DSPTgtPkg.C2800SPI
+ SPI_B	DSPTgtPkg.C2808SPI
+ SPI_C	DSPTgtPkg.C2800SPI
+ SPI_D	DSPTgtPkg.C2800SPI
+ eCAN_A	DSPTgtPkg.eCAN
+ eCAN_B	DSPTgtPkg.eCANB
+ ePWM	DSPTgtPkg.ePWM

DSPBoardLabel

Name of the installed DSP board. Click in the edit box to change the label.

Note The board label displayed in the dialog box must match the label (name) of the board entered in your Code Composer Studio setup.

Custom C280x Board

DSPBoard – DSPChip

[-] DSPChip	DSPTgtPkg.C280xDSPChip
[-] ADC	DSPTgtPkg.C280xADC
ACQ_PS	▼ 4
ADCLKPS	▼ 3
CPS	▼ 1
ExternalReferenceSelector	<input type="checkbox"/> False
OffsetCorrectionValue	0
DSPChipLabel	▼ TI TMS320F2808
[-] SCI_A	DSPTgtPkg.C2800SCI
BaudRate	▼ 115200
BlockingMode	<input type="checkbox"/> False
CharacterLengthBits	▼ 8
CommunicationMode	▼ Raw_data
DataByteOrder	▼ Little_Endian
DataSwapWidth	▼ 8_bits
EnableLoopBack	<input type="checkbox"/> False
NumberOfStopBits	▼ 1
ParityMode	▼ None
SuspensionMode	▼ Free_run
[-] SCI_B	DSPTgtPkg.C2808SCIB
BaudRate	▼ 115200
BlockingMode	<input type="checkbox"/> False
CharacterLengthBits	▼ 8
CommunicationMode	▼ Raw_data
DataByteOrder	▼ Little_Endian
DataSwapWidth	▼ 8_bits
EnableLoopBack	<input type="checkbox"/> False
NumberOfStopBits	▼ 1
ParityMode	▼ None
PinAssignment_Rx	▼ GPIO11
PinAssignment_Tx	▼ GPIO9
SuspensionMode	▼ Free_run
[-] SPI_A	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	▼ No_delay
ClockPolarity	▼ Rising_edge
DataBits	▼ 16
EnableLoopback	<input type="checkbox"/> False
FIFOEnable	<input type="checkbox"/> False
FIFOInterruptLevel_Rx	▼ 16
FIFOInterruptLevel_Tx	▼ 0
FIFOTransmitDelay	0
Mode	▼ Master
SuspensionMode	▼ Free_run

ADC

The internal timing of the ADC module is controlled by the high-speed peripheral clock (HSPCLK). The ADC operating clock speed is derived in several prescaler stages from the HSPCLK speed. For more information on configuring these specific prescalers, see “Configuring Acquisition Window Width for ADC Blocks”. You can set the following parameters for the ADC module:

ACQ_PS

This value does not actually have a direct effect on the ADC module’s core clock speed. It serves to determine the width of the sampling or acquisition period. The higher the value, the wider the sampling period. The default value is 4.

ADCLKPS

The HSPCLK speed is divided by this 4-bit value as the first step in deriving the ADC module’s core clock speed. The default value is 3.

CPS

After the HSPCLK speed is divided by the **ADCLKPS** value, the result is further divided by 2 if the **CPS** parameter is set to 1, which is the default.

ExternalReferenceSelector

By default, an internally generated bandgap voltage reference is selected to supply the ADC logic. However, depending on application requirements, the ADC logic may be supplied by an external voltage reference. Choose True to use an external voltage reference.

OffsetCorrectionValue

The 280x ADC supports offset correction via a 9-bit value that is added or subtracted before the results are available in the ADC result registers. Timing for results is not affected. The default for this field is 0.

DSPChipLabel

DSP chip model. All supported C280x chips are listed. Select the DSP chip installed on your target. The selected value defaults to TI TMS320F2808. Other available options are TI TMS320F2801, TI TMS320F2802, TI TMS320F2806, and TI TMS320F2809.

SCI_A

The serial communications interface parameters you can set for module A are

BaudRate

Baud rate for transmitting and receiving data. Select from 115200, 57600, 38400, 19200, 9600 (the default), 4800, 2400, 1200, 300, and 100.

BlockingMode

If this option is set to True, the system is going to wait until data is available to read (when data length is reached).

If this option is set to False, the system will check FIFO periodically to see if there is data to read. If there is data, the system reads and outputs the contents. If not, it outputs the last value and continues.

CharacterLengthBits

Length in bits of each transmitted/received character, set to 8 bits.

CommunicationMode

Select raw data or protocol mode. 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 completed, sends a checksum.

Advantages to using protocol mode include

- Avoids deadlock
- 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.

DataByteOrder

Select Little Endian or Big Endian.

DataSwapWidth

Select 8-bits or 16-bits.

EnableLoopBack

Select True to enable the loopback function for self-test and diagnostic purposes only. When this parameter is enabled, a C28x DSP's Tx pin is internally connected to Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

NumberOfStopBits

Select whether to use 1 or 2 stop bits.

ParityMode

Type of parity to use. Available selections are None, Odd parity, or Even parity. None disables parity. Odd sets the parity bit to one if you have an odd number of ones in your bytes (e.g., 00110010). Even sets the parity bit to one if you have an even number of ones in your bytes (e.g., 00110011).

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard_abort, Soft_abort, and Free_run. Hard_abort stops the program immediately. Soft_abort stops when the current receive/transmit sequence is complete. Free_run continues running regardless of the breakpoint.

SCI_B

The serial communications interface parameters you can set are

BaudRate

Baud rate for transmitting and receiving data. Select from 115200, 57600, 38400, 19200, 9600 (the default), 4800, 2400, 1200, 300, and 100.

BlockingMode

If this option is set to True, system is going to wait until data is available to read (when data length is reached). If this option is set to False, system will check FIFO periodically to see if there is any data to read. If there is data, it reads and outputs the contents. If there is no data, it outputs the last value and continues.

CharacterLengthBits

Length in bits of each transmitted/received character, set to 8 bits.

CommunicationMode

Select raw data or protocol mode. 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 transmission is completed, it sends a checksum.

Advantages to using protocol mode include

- Avoids deadlock
- 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.

DataByteOrder

Select Little Endian or Big Endian.

DataSwapWidth

Select 8-bits or 16-bits.

EnableLoopBack

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

NumberOfStopBits

Select whether to use 1 or 2 stop bits.

ParityMode

Type of parity to use. Available selections are None, Odd parity, or Even parity. None disables parity. Odd sets the parity bit to one if you have an odd number of ones in your bytes (e.g., 00110010) . Even sets the parity bit to one if you have an even number of ones in your bytes (e.g., 00110011).

PinAssignment_Rx

Assigns the SCI receive something to a GPIO pin. Choices are None (default), GPIO11, GPIO15, GPIO19, or GPIO23.

PinAssignment_Tx

Assigns the SCI transmit something to a GPIO pin. Choices are None (default), GPIO9, GPIO14, GPIO18, or GPIO22.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard_abort, Soft_abort, and Free_run. Hard_abort stops the program immediately. Soft_abort stops when the current receive/transmit sequence is complete. Free_run continues running regardless of the breakpoint.

SPI_A

The serial peripheral interface parameters you can set for the A module are:

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$\text{Baud Rate} = \text{CPU Rate} / (\text{Baud Rate Factor} + 1)$$

ClockPhase

Select No_delay or Delay_half_cycle.

ClockPolarity

Select Rising_edge or Falling_edge.

DataBits

Length in bits from 1 to 16 of each transmitted or received character. For example, if you select 8, the maximum data that can be transmitted using SPI is 2^{8-1} . If you send data greater than this value, the buffer overflows.

EnableLoopback

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

FIFOEnable

Set true or false.

FIFOInterruptLevel_Rx

Set level for receive FIFO interrupt. Select 0 through 16.

FIFOInterruptLevel_Tx

Set level for transmit FIFO interrupt. Select 0 through 16.

FIFOTransmitDelay

Enter FIFO transmit delay (in target clock cycles) to pause between data transmissions.

Mode

Set to Master or Slave.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are `Hard_abort`, `Soft_abort`, and `Free_run`. `Hard_abort` stops the program immediately. `Soft_abort` stops when the current receive or transmit sequence is complete. `Free_run` continues running regardless of the breakpoint.

[-] SPI_B	DSPTgtPkg.C2808SPI
BaudRateFactor	127
ClockPhase	▼ No_delay
ClockPolarity	▼ Rising_edge
DataBits	▼ 16
EnableLoopback	⚙ False
FIFOEnable	⚙ False
FIFOInterruptLevel_Rx	▼ 16
FIFOInterruptLevel_Tx	▼ 0
FIFOTransmitDelay	0
Mode	▼ Master
PinAssignment_CLK	▼ None
PinAssignment_SIMO	▼ None
PinAssignment_SOMI	▼ None
PinAssignment_STE	▼ None
SuspensionMode	▼ Free_run
[-] SPI_C	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	▼ No_delay
ClockPolarity	▼ Rising_edge
DataBits	▼ 16
EnableLoopback	⚙ False
FIFOEnable	⚙ False
FIFOInterruptLevel_Rx	▼ 16
FIFOInterruptLevel_Tx	▼ 0
FIFOTransmitDelay	0
Mode	▼ Master
SuspensionMode	▼ Free_run
[-] SPI_D	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	▼ No_delay
ClockPolarity	▼ Rising_edge
DataBits	▼ 16
EnableLoopback	⚙ False
FIFOEnable	⚙ False
FIFOInterruptLevel_Rx	▼ 16
FIFOInterruptLevel_Tx	▼ 0
FIFOTransmitDelay	0
Mode	▼ Master
SuspensionMode	▼ Free_run

Custom C280x Board

SPI_B

The serial peripheral interface parameters you can set for the B module are

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$\text{Baud Rate} = \text{CPU Rate} / (\text{Baud Rate Factor} + 1)$$

ClockPhase

Select No_delay or Delay_half_cycle.

ClockPolarity

Select Rising_edge or Falling_edge.

DataBits

Length in bits from 1 to 16 of each transmitted or received character. For example, if you select 8, the maximum data that can be transmitted using SPI is 2^{8-1} . If you send data greater than this value, the buffer overflows.

EnableLoopback

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

FIFOEnable

Set true or false.

FIFOInterruptLevel_Rx

Set level for receive FIFO interrupt. Select 0 through 16.

FIFOInterruptLevel_Tx

Set level for transmit FIFO interrupt. Select 0 through 16.

FIFOTransmitDelay

Enter FIFO transmit delay (in seconds).

Mode

Set to Master or Slave.

PinAssignment_CLK

Assigns the SPI something (CLK) to a GPIO pin. Choices are None (default), GPIO14, or GPIO26.

PinAssignment_SIMO

Assigns the SPI something (SIMO) to a GPIO pin. Choices are None (default), GPIO12, or GPIO24.

PinAssignment_SOMI

Assigns the SPI something (SOMI) to a GPIO pin. Choices are None (default), GPIO13, or GPIO25.

PinAssignment_STE

Assigns the SPI something (STE) to a GPIO pin. Choices are None (default), GPIO15, or GPIO27.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard_abort, Soft_abort, and Free_run. Hard_abort stops the program immediately. Soft_abort stops when the current receive/transmit sequence is complete. Free_run continues running regardless of the breakpoint.





SPI_C

The parameters you can set for the SPI_C module include all the parameters for the SPI_A module.

SPI_D

The parameters you can set for the SPI_D module include all the parameters for the SPI_A module.

Custom C280x Board

[-] eCAN_A	DSPTgtPkg.eCAN
BaudRatePrescaler	10
EnhancedCANMode	 True
SAM	▼ Sample_one_time
SBG	▼ Only_falling_edges
SJW	▼ 2
SelfTestMode	 False
TSEG1	▼ 6
TSEG2	▼ 3
[-] eCAN_B	DSPTgtPkg.eCANB
BaudRatePrescaler	10
EnhancedCANMode	 True
PinAssignment_Rx	▼ None
PinAssignment_Tx	▼ None
SAM	▼ Sample_one_time
SBG	▼ Only_falling_edges
SJW	▼ 2
SelfTestMode	 False
TSEG1	▼ 6
TSEG2	▼ 3
[-] ePWM	DSPTgtPkg.ePWM
PinAssignment_SYNCl	▼ None
PinAssignment_SYNCO	▼ None
PinAssignment_TZ5	▼ None
PinAssignment_TZ6	▼ None

eCAN_A

Most of these parameters control the timing of the board. For more information see “Configuring Timing Parameters for CAN Blocks”. The parameters you can set are:

BaudRatePrescaler

Value by which to scale the bit rate. Valid values are from 1 to 256.

EnhancedCANMode

Whether to use the CAN module in extended mode, which provides additional mailboxes and time stamping. The

default is True. Setting this parameter to False enables only standard mode.

SAM

Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of $TQ/2$. A majority decision is made from the three points.

SBG

Sets the message resynchronization triggering. Options are `Only_falling_edges` and `Both_falling_and_rising_edges`.

SJW

Sets the synchronization jump width, which determines how many units of TQ a bit is allowed to be shortened or lengthened when resynchronizing.

SelfTestMode

If this option is set to True, the eCAN module goes to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit. The default is False.

TSEG1

Sets the value of time segment 1, which, with **TSEG2** and the **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG1** are from 1 through 16.

TSEG2

Sets the value of time segment 2, which, with **TSEG1** and the **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG2** are from 1 through 8.

Custom C280x Board

eCAN_B

The parameters you can set for the eCAN_B module include all the parameters for the eCAN_A module.

ePWM

Assigns ePWM signals to GPIO pins, if required.

PinAssignment_SYNCI

Assigns the ePWM external sync pulse input (SYNCI) to a GPIO pin. Choices are None (the default), GPIO6, and GPIO32.

PinAssignment_SYNCO

Assigns the ePWM external sync pulse output (SYNCO) to a GPIO pin. Choices are None (the default), GPIO6, and GPIO33.

PinAssignment_TZ5

Assigns the trip-zone input 5 (TZ5) to a GPIO pin. Choices are None (the default), GPIO16, and GPIO28.

PinAssignment_TZ6

Assigns the trip-zone input 6 (TZ6) to a GPIO pin. Choices are None (the default), GPIO17, and GPIO29.

See Also

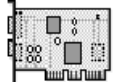
C280x ADC, C280x eCAN Receive, C280x eCAN Transmit, C280x ePWM

Purpose

Target preferences for custom C281x board

Library

c2000tgtpref1lib in Embedded Target for TI C2000 DSP

Description

Custom C281x
Board

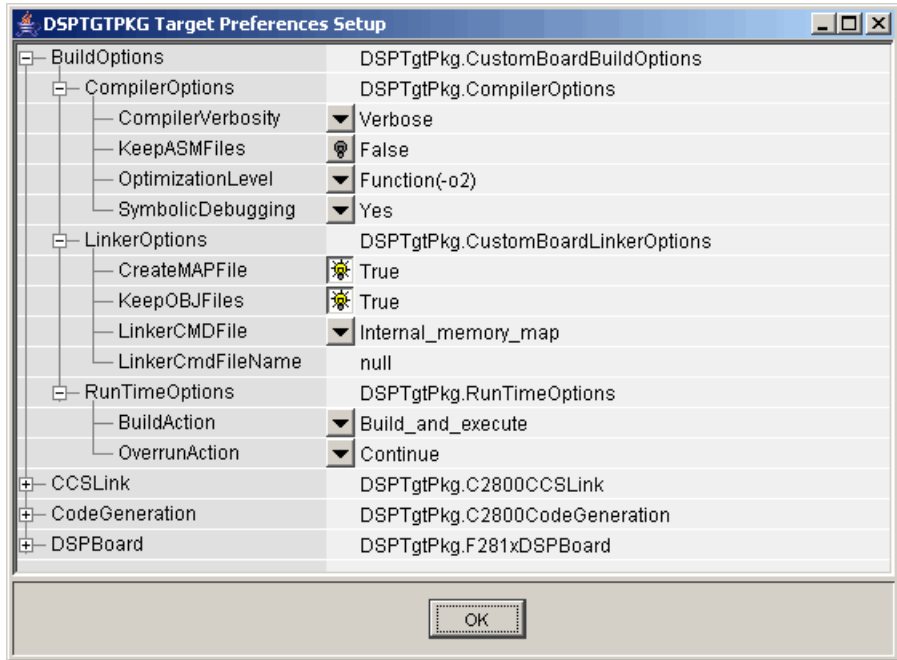
Options on the block mask let you set features of code generation for your custom board based on a C2810, F2810, C2811, F2811, R2811, C2812, F2812, or R2812 chip. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

Note This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model.

Custom C281x Board

Dialog Box

BuildOptions



BuildOptions – CompilerOptions

Compiler Verbosity

Amount of information the compiler returns while it runs. Options are

- Verbose — Returns all compiler messages.
- Quiet — Suppresses compiler progress messages.
- Super_quiet — Suppresses all compiler messages.

KeepASMFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your assembly language (.asm) files after creation. The default is false—.asm files are not kept in your

current directory. If you choose to keep the .asm files, set this option to true.

OptimizationLevel

Degree of optimization provided by the TI optimizing compiler to apply to files in your project. For details about the compiler options, refer to your CCS documentation. When you create new projects, the Embedded Target for TI C2000 DSP sets the optimization to `Function(-o2)`.

SymbolicDebugging

Whether to generate symbolic debugging directives that the C source-level debugger uses and whether to enable assembly source debugging. By default, this option is Yes—symbolic debugging is enabled.

BuildOptions – LinkerOptions

CreateMAPFile

Whether the linker produces a map of the input and output sections, including null areas, and places the listing in a file in your current directory with the name `modelName.map`. The default is True—the listing is produced.

KeepOBJFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your object (.obj) files after creation. The linker uses object files to generate a single executable common object file format (COFF) file that you run on the target DSP. The object files are saved to your current project directory. Saving your .obj files can speed up the compile process by not having to recompile files that you have not changed. The default is True—the .obj files are retained.

LinkerCMDFile

Type of linker command file to use when the linker runs. Linker command files contain linker or hex conversion utility options and the names of input files to the linker or hex conversion utility. Linker command file types are

- `Internal_memory_map` — Uses the small memory model on the target, which requires that all sections of the code and data fit into the internal memory available only on the C281x chip (not including the flash memory).

If you select `Internal_memory_map`, but your data or program requires far calls, the TI compiler returns an error message in the CCS IDE indicating that your data does not fit in internal memory or that your code does not fit in internal memory. The error message looks like one of the following:

```
error: can't allocate '.far'  
error: can't allocate '.text'
```

If you use `Internal_memory_map`, specifying a **LinkerCmdFileName** has no effect.

- `Flash_memory_map` — Uses flash memory, in which case your data can use the full storage capacity of the available flash memory. When you are using flash memory, you cannot set the **BuildAction** preference in the **BuildOptions — RunTimeOptions** section to `Build_and_execute`. For more information on using Flash memory, see “Creating Stand-Alone Applications by Saving Code into Flash Memory”.
- `Custom_file` — Uses the file in the **LinkerCmdFileName** field. This option lets you target custom boards. You must specify the full path of the file. The software does not verify that the commands in this file are correct.

BuildOptions — RunTimeOptions

BuildAction

Action taken by Real-Time Workshop when you click **Build** or press **Ctrl+B** in the Simulation Parameters dialog box. The actions are cumulative—each listed action adds features to the previous action on the list and includes all the previous features:

- `Generate_code_only` — Directs Real-Time Workshop to generate C code only from the model. It does not use the TI

software tools, such as the compiler and linker, and you do not need to have CCS installed. Also, MATLAB does not create the handle to CCS that results from the other options.

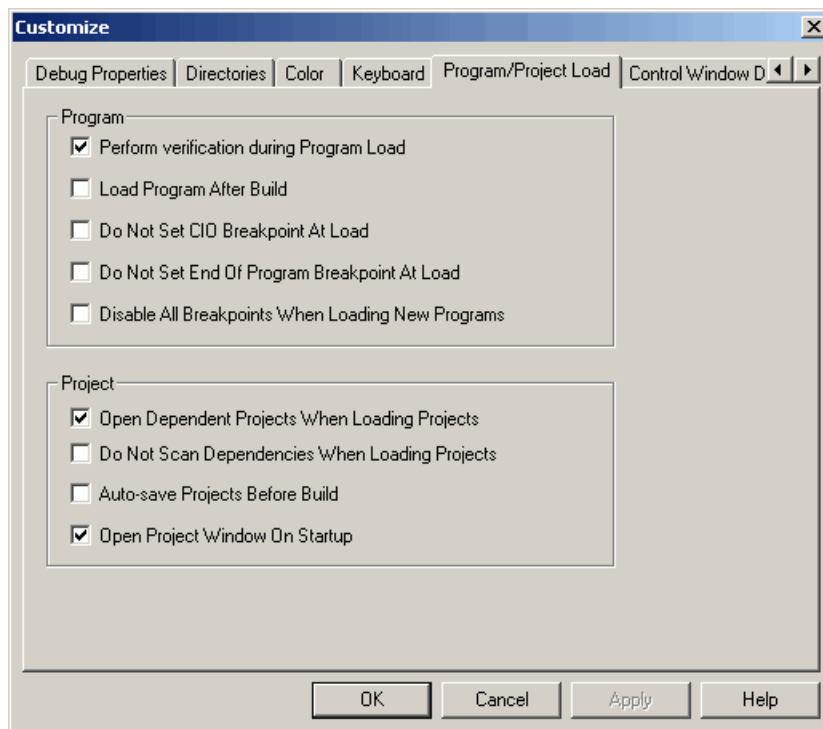
The build process for a model also generates the files `modelName.c`, `modelName.cmd`, `modelName.bld`, and many others. It puts the files in a build directory named `modelName_C2000_rtw` in your MATLAB working directory. This file set contains many of the same files that Real-Time Workshop generates to populate a CCS project when you choose `Create_CCS_Project` for the build action.

- `Create_CCS_Project` — Directs Real-Time Workshop to start CCS and populate a new project with the files from the build process. Selecting this setting enables the CCS board number option so you can select which installed board to target. This option offers a convenient way to build projects in CCS.
- `Build` — Builds the executable COFF file, but does not download the file to the target.
- `Build_and_execute` — Directs Real-Time Workshop to download and run your generated code as an executable on your target. This option is the default.

Setting this option requires corresponding options to be set in the CCS:

- a** From the CCS window, select **Option > Customize...**
- b** Select the **Program/Project Load** tab on the **Customize** dialog box, as shown in the following figure.

Custom C281x Board



c Uncheck the **Load Program After Build** option.

Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

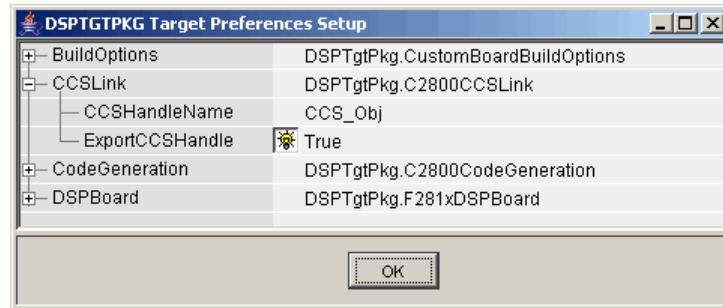
OverrunAction

Defines the action to take when an interrupt overrun occurs:

- **Continue** — Ignore overruns encountered while running the model. This option is the default.

- Halt — Stop program execution.

CCSLink



CCSHandleName

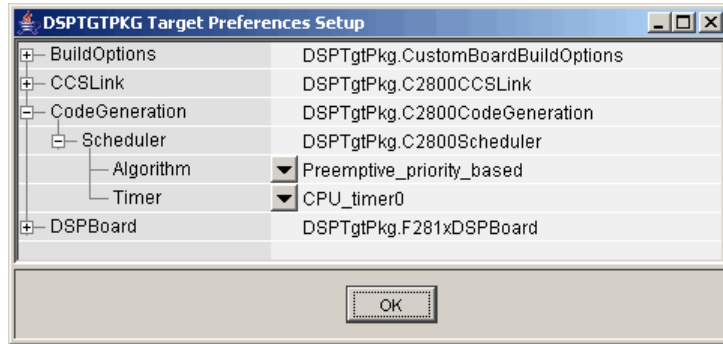
Name of the CCS handle. Click in the edit box to change the name. When you use Real-Time Workshop to build a model for a C2000 target, Embedded Target for TI C2000 DSP makes a link between MATLAB and CCS. If you have used the link portion of the Embedded Target for TI C2000 DSP, you are familiar with function `ccdsp`, which creates links between the IDE and MATLAB. This option refers to the same link, called `cc` in the function reference pages. Although MATLAB to CCS is a link, it is actually a handle to an object that contains information about the object, such as the target board and processor it accesses.

ExportCCSHandle

Whether to export the CCS handle to your MATLAB workspace, giving it the name you assigned in **CCSHandleName**. If this parameter is set to True, after you build your model, you will see the CCS object in your MATLAB workspace browser with the name you provided and class type `ccdsp`.

Custom C281x Board

CodeGeneration – Scheduler



Algorithm

Algorithm to use for scheduling. Currently, the only algorithm option is `Preemptive_priority_based`. This scheduler runs based on the timer interrupt. The timer period is set based on the base rate sample time you specify for your model. This algorithm supports multirate systems in multitasking mode with priority-based preemption. The task for the fastest group (the base rate task) runs first and other tasks run in the order determined by their sample rates from faster tasks to slower tasks. For more information, see “Models with Multiple Sample Rates” in the “Real-Time Workshop” documentation.

Timer

CPU timer to use for scheduling.

DSPBoard

[-] DSPBoard	DSPTgtPkg.F281xDSPBoard
DSPBoardLabel	<Enter your board name>
[-] DSPChip	DSPTgtPkg.C281xDSPChip
[-] ADC	DSPTgtPkg.ADC
DSPChipLabel	TI TMS320C2810
[-] SCL_A	DSPTgtPkg.C2800SCI
BaudRate	115200
BlockingMode	<input type="checkbox"/> False
CharacterLengthBits	8
CommunicationMode	Raw_data
DataByteOrder	Little_Endian
DataSwapWidth	8_bits
EnableLoopBack	<input type="checkbox"/> False
NumberOfStopBits	1
ParityMode	None
SuspensionMode	Free_run
[-] SCL_B	DSPTgtPkg.C2800SCI
BaudRate	115200
BlockingMode	<input type="checkbox"/> False
CharacterLengthBits	8
CommunicationMode	Raw_data
DataByteOrder	Little_Endian
DataSwapWidth	8_bits
EnableLoopBack	<input type="checkbox"/> False
NumberOfStopBits	1
ParityMode	None
SuspensionMode	Free_run
[-] SPI	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	No_delay
ClockPolarity	Rising_edge
DataBits	16
EnableLoopback	<input type="checkbox"/> False
FIFOEnable	<input type="checkbox"/> False
FIFOInterruptLevel_Rx	16
FIFOInterruptLevel_Tx	0
FIFOTransmitDelay	0
Mode	Master
SuspensionMode	Free_run
[-] eCAN	DSPTgtPkg.eCAN
BaudRatePrescaler	10
EnhancedCANMode	<input checked="" type="checkbox"/> True
SAM	Sample_one_time
SBG	Only_falling_edges
SJW	2
SelfTestMode	<input type="checkbox"/> False
TSEG1	8
TSEG2	6

Custom C281x Board

DSPBoardLabel

Name of the installed DSP board. Click in the edit box to change the label.

Note The board label here must match the label (name) of the board entered in your Code Composer Studio setup.

DSPBoard – DSPChip

ADC

The internal timing of the ADC module is controlled by the high-speed peripheral clock (HSPCLK). The ADC operating clock speed is derived in several prescaler stages from the HSPCLK speed. For more information on configuring these prescalers, see “Configuring Acquisition Window Width for ADC Blocks”. You can set the following parameters for the ADC clock prescaler:

ACQ_PS

This value does not actually have a direct effect on the ADC module’s core clock speed. It serves to determine the width of the sampling or acquisition period. The higher the value, the wider the sampling period. The default value is 4.

ADCLKPS

The HSPCLK speed is divided by this 4-bit value as the first step in deriving the ADC module’s core clock speed. The default value is 3.

CPS

After the HSPCLK speed is divided by the **ADCLKPS** value, the result is further divided by 2 if the **CPS** parameter is set to 1, which is the default value.

DSPChipLabel

DSP chip model. All supported C281x chips are listed. Select the DSP chip installed on your target. The selected value defaults to TI TMS320C2810. Other available options are

- TI TMS320F2810
- TI TMS320C2811
- TI TMS320F2811
- TI TMS320R2811
- TI TMS320C2812
- TI TMS320F2812
- TI TMS320R2812

SCI_A

Parameters that affect the serial communications interface (SCI) for module A on the target. You can set the following parameters:

BaudRate

Baud rate for transmitting and receiving data. Select from 115200 (the default), 57600, 38400, 19200, 9600, 4800, 2400, 1200, 300, and 100.

BlockingMode

If this option is set to True, the system waits until data is available to read (when data length is reached). If this option is set to False, system checks FIFO periodically (in polling mode) to see if there is any data to read. If data is present, it reads and outputs the contents. If no data is present, it outputs the last value and continues.

CharacterLengthBits

Length in bits of each transmitted/received character, set to 8 bits.

CommunicationMode

Select raw data or protocol mode. 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 that the receiving side could miss data, but if the data is noncritical, then using raw data mode avoids 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 transaction is completed, it sends a checksum.

Advantages to using protocol mode include

- Avoids deadlock
- 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.

DataByteOrder

Select Little Endian or Big Endian.

DataSwapWidth

Select 8-bits or 16-bits.

EnableLoopBack

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a

C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

NumberOfStopBits

Select whether to use 1 or 2 stop bits.

ParityMode

Type of parity to use. Available selections are None, Odd parity, or Even parity. None disables parity. Odd sets the parity bit to one if you have an odd number of ones in your bytes (e.g., 00110010) . Even sets the parity bit to one if you have an even number of ones in your bytes (e.g., 00110011).

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard_abort, Soft_abort, and Free_run. Hard_abort stops the program immediately. Soft_abort stops when the current receive/transmit sequence is complete. Free_run continues running regardless of the breakpoint.

SCI_B

The parameters you can set for the SCI_B module include all the parameters for the SCI_A module.

SPI

Parameters that affect the serial peripheral interface (SPI) on the target. You can set the following parameters:

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$\text{Baud Rate} = \text{CPU Rate} / (\text{Baud Rate Factor} + 1)$$

ClockPhase

Whether the data is output immediately (`No_delay`) or delayed by a half clock cycle (`Delay_half_cycle`) with respect to the rising edge.

ClockPolarity

Whether the data is output at the `Rising_edge` or `Falling_edge` of the system clock.

DataBits

Length in bits from 1 to 16 of each transmitted or received character. For example, if you select 8, the maximum data that can be transmitted using SPI is 2^{8-1} . If you send data greater than this value, the buffer overflows.

EnableLoopback

Select `True` to enable the loopback function for self-test and diagnostic purposes only. The SPI must be in master mode to use loopback. When this function is enabled, a C281x DSP's SIMO/SOMI lines are connected internally.

FIFOEnable

Select `True` to enable the FIFO buffers in the SPI module.

FIFOInterruptLevel_Rx

Set level for receive FIFO interrupt. Select 0 through 16.

FIFOInterruptLevel_Tx

Set level for transmit FIFO interrupt. Select 0 through 16.

FIFOTransmitDelay

Amount of time in target clock cycles to pause between data transmissions.

Mode

Whether to run the SPI module in `Master` or `Slave` mode. Master mode initiates the transmission. Slave mode is triggered by another master SPI and is synchronized to the clock used by the master SPI. This option cannot be changed at runtime.

SuspensionMode

Suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are `Hard_abort`, `Soft_abort`, and `Free_run`. `Hard_abort` stops the program immediately. `Soft_abort` stops when the current receive or transmit sequence is complete. `Free_run` continues running regardless of the breakpoint.

eCAN

Parameters that affect the extended Control Area Network (eCAN) module. Most of these parameters affect the eCAN bit timing. For more information see “Configuring Timing Parameters for CAN Blocks”.

The eCAN parameters you can set are:

BaudRatePrescaler

Value by which to scale the bit rate. Valid values are from 1 to 256.

EnhancedCANMode

Whether to use the CAN module in extended mode, which provides additional mailboxes and time stamping. The default is `True`. Setting this parameter to `False` enables only standard mode.

SAM

Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of $TQ/2$. A majority decision is made from the three points.

SBG

Sets the message resynchronization triggering. Options are `Only_falling_edges` and `Both_falling_and_rising_edges`.

SJW

Sets the synchronization jump width, which determines how many units of TQ a bit is allowed to be shortened or lengthened when resynchronizing.

SelfTestMode

If `True`, sets the eCAN module to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit. The default is `False`.

TSEG1

Sets the value of time segment 1, which, with **TSEG2** and **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG1** are from 1 through 16.

TSEG2

Sets the value of time segment 2, which, with **TSEG1** and **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG2** are from 1 through 8.

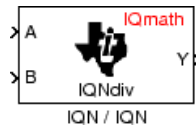
See Also

C281x ADC, C281x eCAN Receive, C281x eCAN Transmit, C281x PWM

Purpose Divide two IQ numbers

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

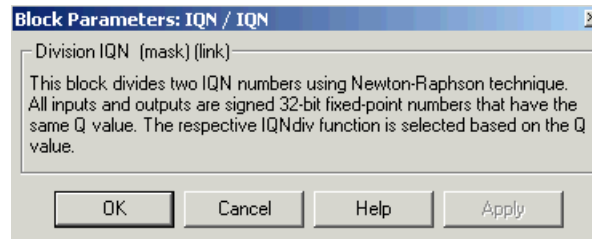
Description



This block divides two numbers that use the same Q format, using the Newton-Raphson technique. The resulting quotient uses the same Q format at the inputs.

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

Dialog Box



See Also

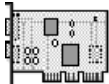
Absolute IQN, Arctangent IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

F2808 eZdsp

Purpose F2808 eZdsp DSK target preferences

Library c2000tgtpref1lib in Embedded Target for TI C2000 DSP

Description



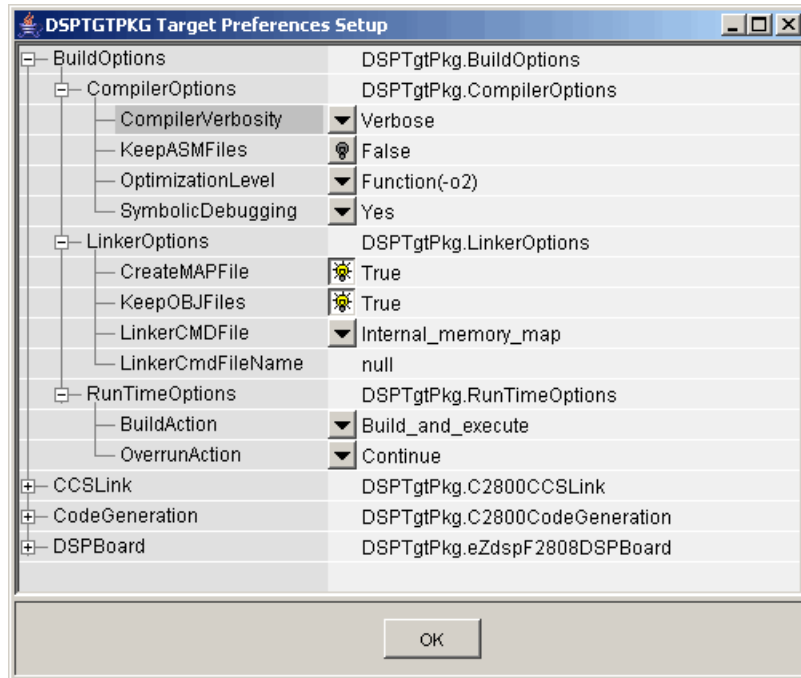
F2808 eZdsp

Options on the block mask let you set features of code generation for your Spectrum Digital F2808 eZdsp target. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

Note This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model.

Dialog Box

BuildOptions



BuildOptions – CompilerOptions

Compiler Verbosity

Amount of information the compiler returns while it runs. Options are

- Verbose — Returns all compiler messages.
- Quiet — Suppresses compiler progress messages.
- Super_quiet — Suppresses all compiler messages.

KeepASMFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your assembly language (.asm) files after

creation. The default is `False`—.asm files are not kept in your current directory. If you choose to keep the .asm files, set this option to `True`.

OptimizationLevel

Degree of optimization provided by the TI optimizing compiler to apply to files in your project. For details about the compiler options, refer to your CCS documentation. When you create new projects, the Embedded Target for TI C2000 DSP sets the optimization to `Function(-o2)`.

SymbolicDebugging

Whether to generate symbolic debugging directives that the C source-level debugger uses and whether to enable assembly source debugging. By default, this option is `Yes`—symbolic debugging is enabled.

BuildOptions – LinkerOptions

CreateMAPFile

Whether the linker produces a map of the input and output sections, including null areas, and places the listing in a file in your current directory with the name `modename.map`. The default is `True`—the listing is produced.

KeepOBJFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your object (.obj) files after creation. The linker uses object files to generate a single executable common object file format (COFF) file that you run on the target DSP. The object files are saved to your current project directory. Saving your .obj files can speed up the compile process by not having to recompile files that you have not changed. The default is `True`—the .obj files are retained.

LinkerCMDFile

Type of linker command file to use when the linker runs. Linker command files contain linker or hex conversion utility options and

the names of input files to the linker or hex conversion utility. Linker command file types are

- `Internal_memory_map` — Uses the small memory model on the target, which requires that all sections of the code and data fit into the memory available only on the F2808 DSP chip (minus the flash memory).

When you select the `Internal_memory_map` option, the Embedded Target for TI C2000 DSP specifies that only the available internal memory on the F2808 is used.

If you select `Internal_memory_map`, but your data or program requires far calls, the TI compiler returns an error message in the CCS IDE indicating that your data does not fit in internal memory or that your code does not fit in internal memory. The error message looks like one of the following:

```
error: can't allocate '.far'  
error: can't allocate '.text'
```

To eliminate these errors, select `Full_memory_map`. Your program might run slower than if you use the internal map option.

- `Full_memory_map` — Uses the large memory model on the target, which does not restrict the size of the code and data sections to DSP memory only. Your data can use the storage space up to the limits of the board.
- `Flash_memory_map` — Uses flash memory, in which case your data can use the full storage capacity of the available flash memory. When you are using flash memory, you cannot set the **BuildAction** preference in the **BuildOptions** — **RunTimeOptions** section to `Build_and_execute`. For more information on using Flash memory, see “Creating Stand-Alone Applications by Saving Code into Flash Memory”.
- `Custom_file` — Uses the file in the **LinkerCmdFileName** field. This option lets you target custom boards. You must specify the full path of the file. The software does not verify

that the commands in this file are correct. Note that if you use `Internal_memory_map` or `Full_memory_map`, specifying a `LinkerCmdFileName` has no effect.

BuildOptions – RunTimeOptions

BuildAction

Action taken by Real-Time Workshop when you click **Build** or press **Ctrl+B** in the Simulation Parameters dialog box. The order in which the actions are presented is significant — each listed action does what the previous action in the list does, and adds new features of its own:

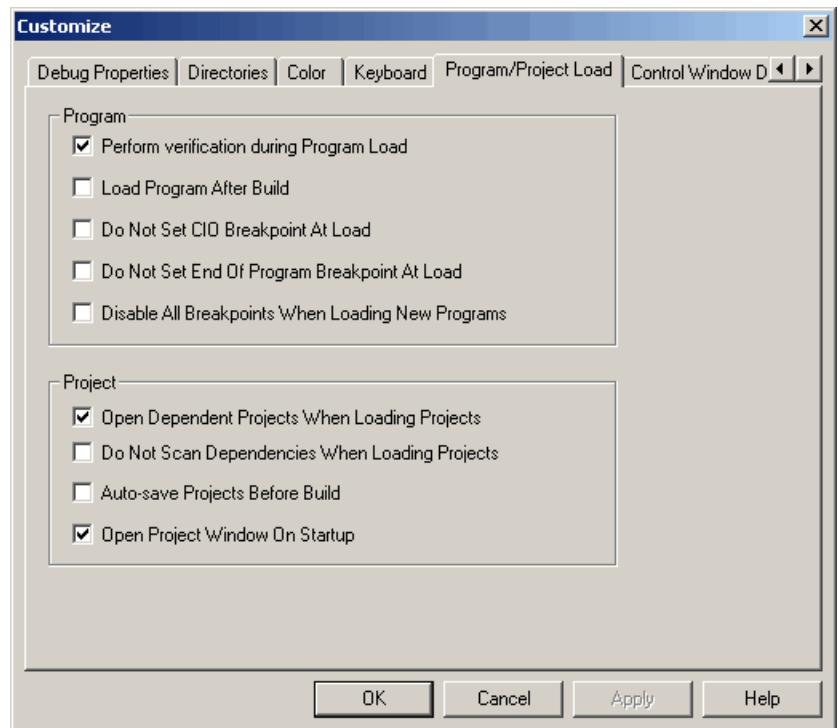
- `Generate_code_only` — Directs Real-Time Workshop to generate C code only from the model. It does not use the TI software tools, such as the compiler and linker, and you do not need to have CCS installed. Also, MATLAB does not create the handle to CCS that results from the other options.

The build process for a model also generates the files `modelName.c`, `modelName.cmd`, `modelName.bld`, and many others. It puts the files in a build directory named `modelName_C2000_rtw` in your MATLAB working directory. This file set contains many of the same files that Real-Time Workshop generates to populate a CCS project when you choose `Create_CCS_Project` for the build action.

- `Create_CCS_Project` — Directs Real-Time Workshop to start CCS and populate a new project with the files from the build process. Selecting this setting enables the CCS board number option so you can select which installed board to target. This option offers a convenient way to build projects in CCS.
- `Build` — Builds the executable COFF file, but does not download the file to the target.
- `Build_and_execute` — Directs Real-Time Workshop to download and run your generated code as an executable on your target. This option is the default.

Setting this option requires corresponding options to be set in the CCS:

- a From the CCS window, select **Option > Customize...**
- b Select the **Program/Project Load** tab on the **Customize** dialog box, as shown in the following figure



- c Uncheck the **Load Program After Build** option.

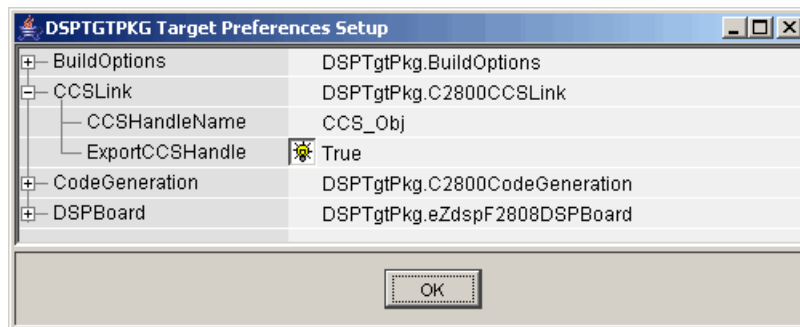
Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

OverrunAction

Defines the action to take when an interrupt overrun occurs:

- Continue — Ignore overruns encountered while running the model. This is the default.
- Halt — Stop program execution.

CCSLink



CCSHandleName

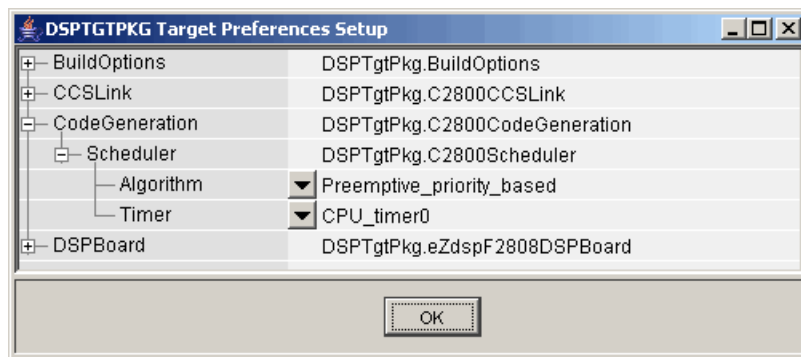
Name of the CCS handle. Click in the edit box to change the name. When you use Real-Time Workshop to build a model for a C2000 target, Embedded Target for TI C2000 DSP makes a link between MATLAB and CCS. If you have used the link portion of the Embedded Target for TI C2000 DSP, you are familiar with function `ccdsp`, which creates links between the IDE and MATLAB. This option refers to the same link, called `cc` in the function reference pages. Although MATLAB to CCS is a link, it

is actually a handle to an object that contains information about the object, such as the target board and processor it accesses.

ExportCCSHandle

Whether to export the CCS handle to your MATLAB workspace, giving it the name you assigned in **CCSHandleName**. If this parameter is set to True (the default), then after you build your model, you see the CCS object in your MATLAB workspace browser with the name you provided and class type `ccsdsp`.

CodeGeneration – Scheduler



Algorithm

Algorithm to use for scheduling. Currently, the only algorithm option is `Preemptive_priority_based`. This scheduler runs based on the timer interrupt. The timer period is set based on the base rate sample time you specify for your model. This algorithm supports multirate systems in multitasking mode with priority-based preemption. The task for the fastest group (the base rate task) runs first and other tasks run in the order determined by their sample rates from faster tasks to slower tasks. For more information, see “Models with Multiple Sample Rates” in the “Real-Time Workshop” documentation.

Timer

CPU timer to use for scheduling.

DSPBoard

[-] DSPBoard	DSPTgtPkg.eZdspF2808DSPBoard
DSPBoardLabel	F2808 eZdsp
[-] DSPChip	DSPTgtPkg.C2808DSPChip
[-] ADC	DSPTgtPkg.C280xADC
ACQ_PS	4
ADCLKPS	3
CPS	1
ExternalReferenceSelector	False
OffsetCorrectionValue	0
DSPChipLabel	TI TMS320F2808
[-] SCL_A	DSPTgtPkg.C2800SCI
BaudRate	115200
BlockingMode	False
CharacterLengthBits	8
CommunicationMode	Raw_data
DataByteOrder	Little_Endian
DataSwapWidth	8_bits
EnableLoopBack	False
NumberOfStopBits	1
ParityMode	None
SuspensionMode	Free_run
[-] SCL_B	DSPTgtPkg.C2808SCIB
BaudRate	115200
BlockingMode	False
CharacterLengthBits	8
CommunicationMode	Raw_data
DataByteOrder	Little_Endian
DataSwapWidth	8_bits
EnableLoopBack	False
NumberOfStopBits	1
ParityMode	None
PinAssignment_Rx	GPIO11
PinAssignment_Tx	GPIO9
SuspensionMode	Free_run
[-] SPL_A	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	No_delay
ClockPolarity	Rising_edge
DataBits	16
EnableLoopback	False
FIFOEnable	False
FIFOInterruptLevel_Rx	16
FIFOInterruptLevel_Tx	0
FIFOTransmitDelay	0
Mode	Master
SuspensionMode	Free_run

DSPBoardLabel

Name of the installed DSP board. Click in the edit box to change the label.

Note The board label here must match the label (name) of the board entered in your Code Composer Studio setup.

DSPBoard – DSPChip

ADC

The internal timing of the ADC module is controlled by the high-speed peripheral clock (HSPCLK). The ADC operating clock speed is derived in several prescaler stages from the HSPCLK speed. For more information on configuring these scalers, see “Configuring Acquisition Window Width for ADC Blocks”. You can set the following parameters for the ADC clock prescaler:

ACQ_PS

This value does not actually have a direct effect on the ADC module’s core clock speed. It serves to determine the width of the sampling or acquisition period. The higher the value, the wider the sampling period. The default value is 4.

ADCLKPS

The HSPCLK speed is divided by this 4-bit value as the first step in deriving the ADC module’s core clock speed. The default value is 3.

CPS

After the HSPCLK speed is divided by the **ADCLKPS** value, the result will be further divided by 2 if the **CPS** parameter is set to 1, which is the default.

ExternalReferenceSelector

By default, an internally generated bandgap voltage reference is selected to supply the ADC logic. However,

depending on application requirements, the ADC logic may be supplied by an external voltage reference. Choose True to use an external voltage reference.

OffsetCorrectionValue

The 280x ADC supports offset correction via a 9-bit value that is added or subtracted before the results are available in the ADC result registers. Timing for results is not affected. The default value is 0.

DSPChipLabel

DSP chip model. All supported C2000 chips are listed. For the F2808 eZdsp board, the selected value defaults to TI TMS320C2808. If you select a different chip model, an error is generated in code generation.

SCI_A

The serial communications interface parameters you can set for module A. These parameters are:

BaudRate

Baud rate for transmitting and receiving data. Select from 115200 (the default), 57600, 38400, 19200, 9600, 4800, 2400, 1200, 300, and 110.

BlockingMode

If this option is set to True, system waits until data is available to read (when data length is reached). If this option is set to False, system checks FIFO periodically (in polling mode) to see if there is any data to read. If data is present, it reads and outputs the contents. If no data is present, it outputs the last value and continues.

CharacterLengthBits

Length in bits of each transmitted or received character, set to 8 bits.

CommunicationMode

Select raw data or protocol mode. 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

- Avoids deadlock
- 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.

DataByteOrder

Select Little Endian or Big Endian.

DataSwapWidth

Select 8-bits or 16-bits.

EnableLoopBack

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

NumberOfStopBits

Select whether to use 1 or 2 stop bits.

ParityMode

Type of parity to use. Available selections are Odd parity or Even parity. **Enable Parity** must be set to True to use the selected **ParityMode**.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard_abort, Soft_abort, and Free_run. Hard_abort stops the program immediately. Soft_abort stops when the current receive/transmit sequence is complete. Free_run continues running regardless of the breakpoint.

SCI_B

The serial communications interface parameters you can set for module B. These parameters are:

BaudRate

Baud rate for transmitting and receiving data. Select from 115200(the default), 57600, 38400, 19200, 9600, 4800, 2400, 1200, 300, and 110.

BlockingMode

If this option is set to True, the system waits until data is available to read (when data length is reached). If this option is set to False, system will check FIFO periodically (in polling mode) to see if there is any data to read. If data is present, it reads and outputs the contents. If no data is present, it outputs the last value and continues.

CharacterLengthBits

Length in bits of each transmitted/received character, set to 8 bits.

CommunicationMode

Select raw data or protocol mode. 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

- Avoids deadlock
- 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.

DataByteOrder

Select Little Endian or Big Endian.

DataSwapWidth

Select 8-bits or 16-bits.

EnableLoopBack

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

NumberOfStopBits

Select whether to use 1 or 2 stop bits.

ParityMode

Type of parity to use. Available selections are None, Odd parity, or Even parity. None disables parity. Odd sets the parity bit to one if you have an odd number of ones in your bytes (e.g., 00110010) . Even sets the parity bit to one if you have an even number of ones in your bytes (e.g., 00110011).

PinAssignment_Rx

Assigns the SCI receive something to a GPIO pin. Choices are None (default), GPIO11, GPIO15, GPIO19, or GPIO23.

PinAssignment_Tx

Assigns the SCI transmit something to a GPIO pin. Choices are None (default), GPIO9, GPIO14, GPIO18, or GPIO22.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are `Hard_abort`, `Soft_abort`, and `Free_run`. `Hard_abort` stops the program immediately. `Soft_abort` stops when the current receive or transmit sequence is complete. `Free_run` continues running regardless of the breakpoint.

SPI_A

The serial peripheral interface parameters you can set for the A module. These parameters are:

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$\text{Baud Rate} = \text{CPU Rate} / (\text{Baud Rate Factor} + 1)$$

ClockPhase

Select `No_delay` or `Delay_half_cycle`.

ClockPolarity

Select `Rising_edge` or `Falling_edge`.

DataBits

Length in bits from 1 to 16 of each transmitted/received character. For example, if you select 8, the maximum data that can be transmitted using SPI is 2^{8-1} . If you send data greater than this value, the buffer overflows.

EnableLoopback

Select `True` to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and

can transmit data from its output port to its input port to check the integrity of the transmission.

FIFOEnable

Set true or false.

FIFOInterruptLevel_Rx

Set level for receive FIFO interrupt. Select 0 through 16.

FIFOInterruptLevel_Tx

Set level for transmit FIFO interrupt. Select 0 through 16.

FIFOTransmitDelay

Enter FIFO transmit delay (in target clock cycles) to pause between data transmissions.

Mode

Set to Master or Slave.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are `Hard_abort`, `Soft_abort`, and `Free_run`. `Hard_abort` stops the program immediately. `Soft_abort` stops when the current receive or transmit sequence is complete. `Free_run` continues running regardless of the breakpoint.

[-] SPI_B	DSPTgtPkg.C2808SPI
BaudRateFactor	127
ClockPhase	▼ No_delay
ClockPolarity	▼ Rising_edge
DataBits	▼ 16
EnableLoopback	⊗ False
FIFOEnable	⊗ False
FIFOInterruptLevel_Rx	▼ 16
FIFOInterruptLevel_Tx	▼ 0
FIFOTransmitDelay	0
Mode	▼ Master
PinAssignment_CLK	▼ None
PinAssignment_SIMO	▼ None
PinAssignment_SOMI	▼ None
PinAssignment_STE	▼ None
SuspensionMode	▼ Free_run
[-] SPI_C	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	▼ No_delay
ClockPolarity	▼ Rising_edge
DataBits	▼ 16
EnableLoopback	⊗ False
FIFOEnable	⊗ False
FIFOInterruptLevel_Rx	▼ 16
FIFOInterruptLevel_Tx	▼ 0
FIFOTransmitDelay	0
Mode	▼ Master
SuspensionMode	▼ Free_run
[-] SPI_D	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	▼ No_delay
ClockPolarity	▼ Rising_edge
DataBits	▼ 16
EnableLoopback	⊗ False
FIFOEnable	⊗ False
FIFOInterruptLevel_Rx	▼ 16
FIFOInterruptLevel_Tx	▼ 0
FIFOTransmitDelay	0
Mode	▼ Master
SuspensionMode	▼ Free_run

SPI_B

The serial peripheral interface parameters you can set for the B module. These parameters are:

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$\text{Baud Rate} = \text{CPU Rate} / (\text{Baud Rate Factor} + 1)$$

ClockPhase

Select No_delay or Delay_half_cycle.

ClockPolarity

Select Rising_edge or Falling_edge.

DataBits

Length in bits from 1 to 16 of each transmitted or received character. For example, if you select 8, the maximum data that can be transmitted using SPI is 2^{8-1} . If you send data greater than this value, the buffer overflows.

EnableLoopback

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

FIFOEnable

Set true or false.

FIFOInterruptLevel_Rx

Set level for receive FIFO interrupt. Select 0 through 16.

FIFOInterruptLevel_Tx

Set level for transmit FIFO interrupt. Select 0 through 16.

FIFOTransmitDelay

Enter FIFO transmit delay (in seconds).

Mode

Set to Master or Slave.

PinAssignment_CLK

Assigns the SPI something (CLK) to a GPIO pin. Choices are None (default), GPIO14, or GPIO26.

PinAssignment_SIMO

Assigns the SPI something (SIMO) to a GPIO pin. Choices are None (default), GPIO12, or GPIO24.

PinAssignment_SOMI

Assigns the SPI something (SOMI) to a GPIO pin. Choices are None (default), GPIO13, or GPIO25.

PinAssignment_STE

Assigns the SPI something (STE) to a GPIO pin. Choices are None (default), GPIO15, or GPIO27.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard_abort, Soft_abort, and Free_run. Hard_abort stops the program immediately. Soft_abort stops when the current receive or transmit sequence is complete. Free_run continues running regardless of the breakpoint.

SPI_C

The parameters you can set for the SPI_C module include all the parameters for the SPI_A module.

SPI_D

The parameters you can set for the SPI_D module include all the parameters for the SPI_A module.

[-] eCAN_A	DSPTgtPkg.eCAN
BaudRatePrescaler	10
EnhancedCANMode	<input checked="" type="checkbox"/> True
SAM	▼ Sample_one_time
SBG	▼ Only_falling_edges
SJW	▼ 2
SelfTestMode	<input type="checkbox"/> False
TSEG1	▼ 6
TSEG2	▼ 3
[-] eCAN_B	DSPTgtPkg.eCANB
BaudRatePrescaler	10
EnhancedCANMode	<input checked="" type="checkbox"/> True
PinAssignment_Rx	▼ None
PinAssignment_Tx	▼ None
SAM	▼ Sample_one_time
SBG	▼ Only_falling_edges
SJW	▼ 2
SelfTestMode	<input type="checkbox"/> False
TSEG1	▼ 6
TSEG2	▼ 3
[-] ePWM	DSPTgtPkg.ePWM
PinAssignment_SYNCI	▼ None
PinAssignment_SYNCO	▼ None
PinAssignment_TZ5	▼ None
PinAssignment_TZ6	▼ None

eCAN_A

For more help on setting the timing parameters for the eCAN modules see: “Configuring Timing Parameters for CAN Blocks”. You can set the following parameters for the eCAN module:

BaudRatePrescaler

Value by which to scale the bit rate. Valid values are from 1 to 256.

EnhancedCANMode

Whether to use the CAN module in extended mode, which provides additional mailboxes and time stamping. The

default is True. Setting this parameter to False enables only standard mode.

SAM

Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of $TQ/2$. A majority decision is made from the three points.

SBG

Sets the message resynchronization triggering. Options are `Only_falling_edges` and `Both_falling_and_rising_edges`.

SJW

Sets the synchronization jump width, which determines how many units of TQ a bit is allowed to be shortened or lengthened when resynchronizing.

SelfTestMode

If this parameter is set to True, the eCAN module goes to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit. The default is False.

TSEG1

Sets the value of time segment 1, which, with **TSEG2** and **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG1** are from 1 through 16.

TSEG2

Sets the value of time segment 2, which, with **TSEG1** and **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG2** are from 1 through 8.

eCAN_B

The parameters you can set for the eCAN_B module include all the parameters for the eCAN_A module plus the following parameters which only apply when you are using the eCAN_B module:

PinAssignment_Rx

Assigns the CAN receive pin to use with the eCAN_B module. Possible values are GPIO10, GPIO13, GPIO17, and GPIO21.

PinAssignment_Tx

Assigns the CAN transmit pin to use with the eCAN_B module. Possible values are GPIO8, GPIO12, GPIO16, and GPIO20.

ePWM

Assigns ePWM signals to GPIO pins, if required.

PinAssignment_SYNCI

Assigns the ePWM external sync pulse input (SYNCI) to a GPIO pin. Choices are None (the default), GPIO6, and GPIO32.

PinAssignment_SYNCO

Assigns the ePWM external sync pulse output (SYNCO) to a GPIO pin. Choices are None (the default), GPIO6, and GPIO33.

PinAssignment_TZ5

Assigns the trip-zone input 5 (TZ5) to a GPIO pin. Choices are None (the default), GPIO16, and GPIO28.

PinAssignment_TZ6

Assigns the trip-zone input 6 (TZ6) to a GPIO pin. Choices are None (the default), GPIO17, and GPIO29.

See Also

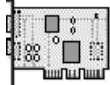
C280x ADC, C280x eCAN Receive, C280x eCAN Transmit, C280x ePWM, C280x eQEP, C280x Hardware Interrupt, Idle Task

Purpose

F2812 eZdsp DSK target preferences

Library

c2000tgtpref1lib in Embedded Target for TI C2000 DSP

Description

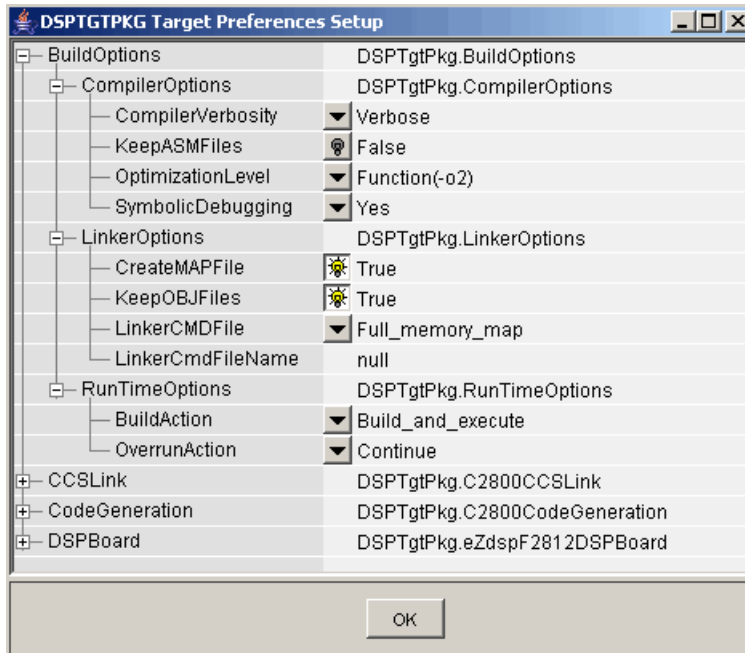
F2812 eZdsp

Options on the block mask let you set features of code generation for your Spectrum Digital F2812 eZdsp target. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

Note This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model.

Dialog Box

BuildOptions



BuildOptions – CompilerOptions

Compiler Verbosity

Amount of information the compiler returns while it runs. Options are

- Verbose — Returns all compiler messages.
- Quiet — Suppresses compiler progress messages.
- Super_quiet — Suppresses all compiler messages.

KeepASMFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your assembly language (.asm) files after creation. The default is false—.asm files are not kept in your

current directory. If you choose to keep the .asm files, set this option to true.

OptimizationLevel

Degree of optimization provided by the TI optimizing compiler to apply to files in your project. For details about the compiler options, refer to your CCS documentation. When you create new projects, the Embedded Target for TI C2000 DSP sets the optimization to `Function(-o2)`.

SymbolicDebugging

Whether to generate symbolic debugging directives that the C source-level debugger uses and whether to enable assembly source debugging. By default, this option is `Yes`—symbolic debugging is enabled.

BuildOptions – LinkerOptions

CreateMAPFile

Whether the linker produces a map of the input and output sections, including null areas, and places the listing in a file in your current directory with the name `modelName.map`. The default is `True` — the listing is produced.

KeepOBJFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your object (.obj) files after creation. The linker uses object files to generate a single executable common object file format (COFF) file that you run on the target DSP. The object files are saved to your current project directory. Saving your .obj files can speed up the compile process by not having to recompile files that you have not changed. The default is `True`—the .obj files are retained.

LinkerCMDFile

Type of linker command file to use when the linker runs. Linker command files contain linker or hex conversion utility options and the names of input files to the linker or hex conversion utility. Linker command file types are

- `Internal_memory_map` — Uses the small memory model on the target, which requires that all sections of the code and data fit into the memory available only on the F2812 DSP chip (minus the flash memory).
- `Full_memory_map` — Uses the large memory model on the target, which does not restrict the size of the code and data sections to DSP memory only. Your data can use the storage space up to the limits of the board.
- `Flash_memory_map` — Uses flash memory, in which case your data can use the full storage capacity of the available flash memory. When you are using flash memory, you cannot set the **BuildAction** preference in the **BuildOptions — RunTimeOptions** section to `Build_and_execute`. For more information on using Flash memory, see “Creating Stand-Alone Applications by Saving Code into Flash Memory”.
- `Custom_file` — Uses the file in the **LinkerCmdFileName** field. This option lets you target custom boards. You must specify the full path of the file. The software does not verify that the commands in this file are correct. If you use `Internal_memory_map` or `Full_memory_map`, specifying a `Custom_file` has no effect.

When you select the `Internal_memory_map` option, the Embedded Target for TI C2000 DSP specifies that only the available internal memory on the F2812 is used.

If you select `Internal_memory_map`, but your data or program requires far calls, the TI compiler returns an error message like the following in the CCS IDE:

```
error: can't allocate '.far'  
or  
error: can't allocate '.text'
```


indicating that your data does not fit in internal memory or that your code does not fit in internal memory. To eliminate these errors, select `Full_memory_map`. Your program might run more slowly than if you use the internal map option.

BuildOptions – RunTimeOptions

BuildAction

Action taken by Real-Time Workshop when you click **Build** or press **Ctrl+B** in the Simulation Parameters dialog box. The actions are cumulative—each listed action adds features to the previous action on the list and includes all the previous features:

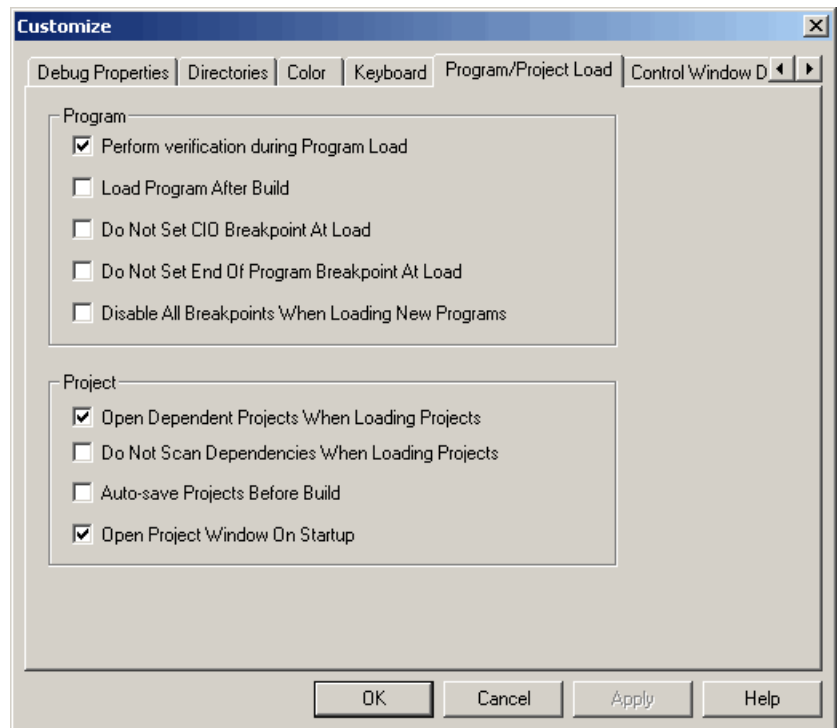
- `Generate_code_only` — Directs Real-Time Workshop to generate C code only from the model. It does not use the TI software tools, such as the compiler and linker, and you do not need to have CCS installed. Also, MATLAB does not create the handle to CCS that results from the other options.

The build process for a model also generates the files `modelName.c`, `modelName.cmd`, `modelName.bld`, and many others. It puts the files in a build directory named `modelName_C2000_rtw` in your MATLAB working directory. This file set contains many of the same files that Real-Time Workshop generates to populate a CCS project when you choose `Create_CCS_Project` for the build action.

- `Create_CCS_Project` — Directs Real-Time Workshop to start CCS and populate a new project with the files from the build process. Selecting this setting enables the CCS board number option so you can select which installed board to target. This option offers a convenient way to build projects in CCS.
- `Build` — Builds the executable COFF file, but does not download the file to the target.
- `Build_and_execute` — Directs Real-Time Workshop to download and run your generated code as an executable on your target. This option is the default.

Setting this option requires corresponding options to be set in the CCS:

- a From the CCS window, select **Option > Customize...**
- b Select the **Program/Project Load** tab on the **Customize** dialog box as shown in the following figure:



- c Uncheck the **Load Program After Build** option.

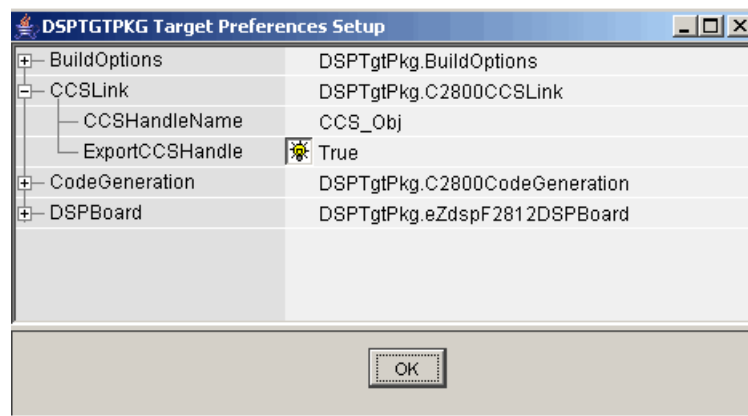
Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

OverrunAction

Defines the action to take when an interrupt overrun occurs.

- Continue — Ignore overruns encountered while running the model. This is the default.
- Halt — Stop program execution.

CCSLink



CCSHandleName

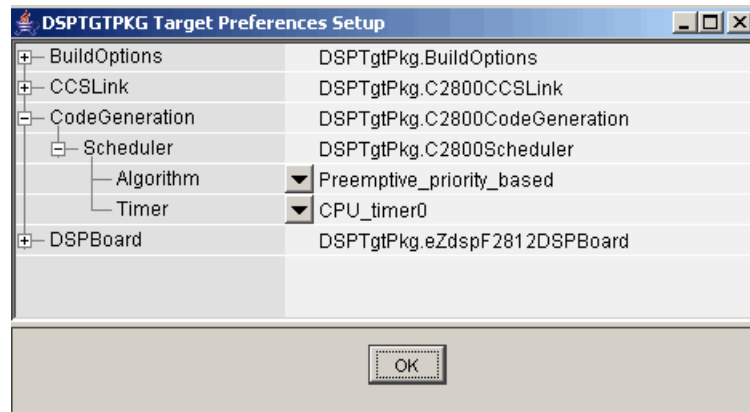
Name of the CCS handle. Click in the edit box to change the name. When you use Real-Time Workshop to build a model for a C2000 target, Embedded Target for TI C2000 DSP makes a link between MATLAB and CCS. If you have used the link portion of the Embedded Target for TI C2000 DSP, you are familiar with function `ccsdsp`, which creates links between the IDE and

MATLAB. This option refers to the same link, called `cc` in the function reference pages. Although MATLAB to CCS is a link, it is actually a handle to an object that contains information about the object, such as the target board and processor it accesses.

ExportCCSHandle

Whether to export the CCS handle to your MATLAB workspace, giving it the name you assigned in **CCSHandleName**. If this parameter is set to true, after you build your model, you see the CCS object in your MATLAB workspace browser with the name you provided and class type `ccsdsp`.

CodeGeneration – Scheduler



Algorithm

Algorithm to use for scheduling. Currently, the only algorithm option is `Preemptive_priority_based`. This scheduler runs based on the timer interrupt. The timer period is set based on the base rate sample time you specify for your model. This algorithm supports multirate systems in multitasking mode with priority-based preemption. The task for the fastest group (the base rate task) runs first and other tasks run in the order determined by their sample rates from faster tasks to slower

tasks. For more information, see “Models with Multiple Sample Rates” in the “Real-Time Workshop” documentation.

Timer

CPU timer to use for scheduling.

F2812 eZdsp

DSPBoard

[-] DSPBoard	DSPTgtPkg.eZdspF2812DSPBoard
DSPBoardLabel	F2812 eZdsp
[-] DSPChip	DSPTgtPkg.C2812DSPChip
[-] ADC	DSPTgtPkg.ADC
ACQ_PS	4
ADCLKPS	3
CPS	1
DSPChipLabel	TI TMS320C2812
[-] SCL_A	DSPTgtPkg.C2800SCI
BaudRate	115200
BlockingMode	<input type="checkbox"/> False
CharacterLengthBits	8
CommunicationMode	Raw_data
DataByteOrder	Little_Endian
DataSwapWidth	8_bits
EnableLoopBack	<input type="checkbox"/> False
NumberOfStopBits	1
ParityMode	None
SuspensionMode	Free_run
[-] SCL_B	DSPTgtPkg.C2800SCI
BaudRate	115200
BlockingMode	<input type="checkbox"/> False
CharacterLengthBits	8
CommunicationMode	Raw_data
DataByteOrder	Little_Endian
DataSwapWidth	8_bits
EnableLoopBack	<input type="checkbox"/> False
NumberOfStopBits	1
ParityMode	None
SuspensionMode	Free_run
[-] SPI	DSPTgtPkg.C2800SPI
BaudRateFactor	127
ClockPhase	No_delay
ClockPolarity	Rising_edge
DataBits	16
EnableLoopback	<input type="checkbox"/> False
FIFOEnable	<input type="checkbox"/> False
FIFOInterruptLevel_Rx	16
FIFOInterruptLevel_Tx	0
FIFOTransmitDelay	0
Mode	Master
SuspensionMode	Free_run
[-] eCAN	DSPTgtPkg.eCAN
BaudRatePrescaler	10
EnhancedCANMode	<input checked="" type="checkbox"/> True
SAM	Sample_one_time
SBG	Only_falling_edges
SJW	2
SelfTestMode	<input type="checkbox"/> False
TSEG1	8
TSEG2	6

DSPBoardLabel

Name of the installed DSP board. Click in the edit box to change the label.

Note The board label here must match exactly the label (name) of the board entered in your Code Composer Studio setup.

DSPBoard – DSPChip

DSPChipLabel

DSP chip model. Select the DSP chip installed on your target. The chip model is fixed for the F2812 eZdsp. If you change the chip model, an error is generated in code generation.

ADC

The internal timing of the ADC module is controlled by the high-speed peripheral clock (HSPCLK). The ADC operating clock speed is derived in several prescaler stages from the HSPCLK speed. For more information on configuring these specific scalers, see “Configuring Acquisition Window Width for ADC Blocks”. You can set the following parameters for the ADC clock prescaler:

ACQ_PS

This value does not actually have a direct effect on the ADC module’s core clock speed. It serves to determine the width of the sampling or acquisition period. The higher the value, the wider the sampling period. The default value is 4.

ADCLKPS

The HSPCLK speed is divided by this 4-bit value as the first step in deriving the ADC module’s core clock speed. The default value is 3.

CPS

After the HSPCLK speed is divided by the **ADCLKPS** value, the result is further divided by 2 if the **CPS** parameter is set to 1, which is the default.

ExternalReferenceSelector

By default, an internally generated bandgap voltage reference is selected to supply the ADC logic. However, depending on application requirements, the ADC logic may be supplied by an external voltage reference. Choose True to use an external voltage reference.

OffsetCorrectionValue

The 280x ADC supports offset correction via a 9-bit value that is added or subtracted before the results are available in the ADC result registers. Timing for results is not affected. The default for this field is 0.

SCI_A

Parameters that affect the serial communications interface (SCI) for module A on the target. The parameters you can set are:

BaudRate

Baud rate for transmitting and receiving data. Choices are 115200 (the default), 57600, 38400, 19200, 9600, 4800, 2400, 1200, 300, and 110.

BlockingMode

If this option is set to True, the system waits until data is available to read (when data length is reached). If this option is set to False, the system checks FIFO periodically (in polling mode) to see if there is any data to read. If data is present, it reads and outputs the contents. If no data is present, it outputs the last value and continues.

CharacterLengthBits

Length in bits of each transmitted or received character, set to 8 bits.

CommunicationMode

Select raw data or protocol mode. 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

- Avoids deadlock
- 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.

DataByteOrder

Select Little Endian or Big Endian.

DataSwapWidth

Select 8-bits or 16-bits.

EnableLoopBack

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

NumberOfStopBits

Select whether to use 1 or 2 stop bits.

ParityMode

Type of parity to use. Available selections are None, Odd parity, or Even parity. None disables parity. Odd sets the parity bit to one if you have an odd number of ones in your bytes (e.g., 00110010) . Even sets the parity bit to one if you have an even number of ones in your bytes (e.g., 00110011).

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard_abort, Soft_abort, and Free_run. Hard_abort stops the program immediately. Soft_abort stops when the current receive/transmit sequence is complete. Free_run continues running regardless of the breakpoint.

SCI_B

The parameters you can set for the SCI_B module include all the parameters for the SCI_A module.

SPI

Parameters that affect the serial peripheral interfaces (SPI) on the target. The parameters you can set are

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$\text{Baud Rate} = \text{CPU Rate} / (\text{Baud Rate Factor} + 1)$$

ClockPhase

Whether the data is output immediately (`No_delay`) or delayed by a half clock cycle (`Delay_half_cycle`) with respect to the rising edge.

ClockPolarity

Whether the data is output at the `Rising_edge` or `Falling_edge` of the system clock.

DataBits

Length in bits from 1 to 16 of each transmitted or received character. For example, if you select 8, the maximum data that can be transmitted using SPI is 2^{8-1} . If you send data greater than this value, the buffer overflows.

EnableLoopBack

Select `True` to enable the loopback function for self-test and diagnostic purposes only. The SPI must be in master mode to use loopback. When this function is enabled, a C28x DSP's SIMO/SOMI lines are connected internally.

FIFOEnable

Select `True` to enable the FIFO buffers in the SPI module.

FIFOInterruptLevel_Rx

Set level for receive FIFO interrupt. Select 0 through 16.

FIFOInterruptLevel_Tx

Set level for transmit FIFO interrupt. Select 0 through 16.

FIFOTransmitDelay

Amount of time in target clock cycles to pause between data transmissions.

Mode

Whether to run the SPI module in Master or Slave mode. Master mode initiates the transmission. Slave mode is triggered by another master SPI and is synchronized to the clock used by the master SPI. Note that this option cannot be changed at runtime.

SuspensionMode

Suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard abort, Soft abort, and Free run. Hard abort stops the program immediately. Soft abort stops when the current receive or transmit sequence is complete. Free run continues running regardless of the breakpoint.

eCAN

Parameters that affect the extended control area network (eCAN) module. Most of these parameters affect the eCAN bit timing. For more information on setting the CAN bit timing parameters, see “Configuring Timing Parameters for CAN Blocks”

You can set the following CAN parameters:

BaudRatePrescaler

Value by which to scale the bit rate. Valid values are from 1 to 256.

EnhancedCANMode

Whether to use the CAN module in extended mode, which provides additional mailboxes and time stamping. The default is True. Setting this parameter to False enables only standard mode.

SAM

Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples

once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of $TQ/2$. A majority decision is made from the three points.

SBG

Sets the message resynchronization triggering. Options are `Only_falling_edges` and `Both_falling_and_rising_edges`.

SJW

Sets the synchronization jump width, which determines how many units of TQ a bit is allowed to be shortened or lengthened when resynchronizing.

SelfTestMode

If this option is set to `True`, the eCAN module goes to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit. The default is `False`.

TSEG1

Sets the value of time segment 1, which, with **TSEG2** and **BRP**, determines the length of a bit on the eCAN bus. Valid values for **TSEG1** are from 1 through 16.

TSEG2

Sets the value of time segment 2, which, with **TSEG1** and **BRP**, determines the length of a bit on the eCAN bus. Valid values for **TSEG2** are from 1 through 8.

See Also

C281x ADC, C281x eCAN Receive, C281x eCAN Transmit, C281x PWM

Float to IQN

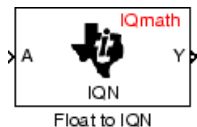
Purpose

Convert floating-point number to IQ number

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

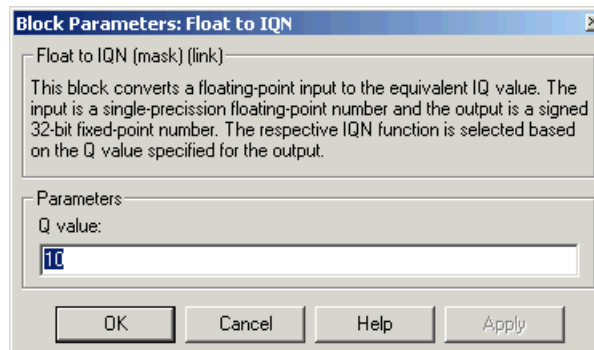
Description



This block converts a floating-point number to an IQ number. The Q value of the output is specified in the dialog.

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

Dialog Box



Q value

Q value from 1 to 30 that specifies the precision of the output

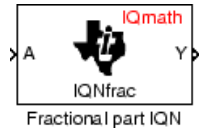
See Also

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

Purpose Fractional part of IQ number

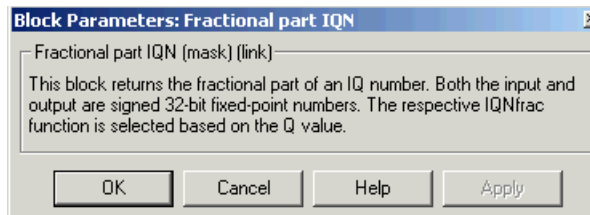
Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block returns the fractional portion of an IQ number. The returned value is an IQ number in the same IQ format.



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

Dialog Box



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

Fractional part IQN x int32

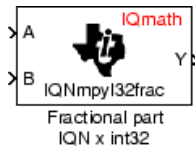
Purpose

Fractional part of result of multiplying IQ number and long integer

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

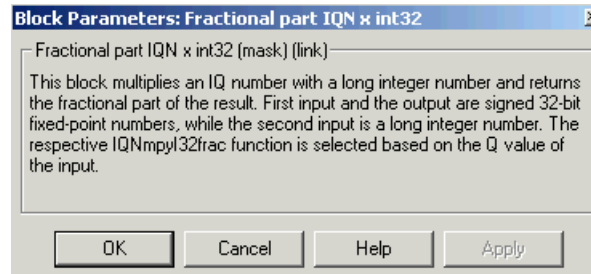
Description



This block multiplies an IQ input and a long integer input and returns the fractional portion of the resulting IQ number.

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

Dialog Box



See Also

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

Purpose

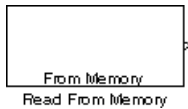
Retrieve data from target memory

Library

c2400spchiplib or c280xspchiplib or c281xspchiplib in Embedded Target for TI C2000 DSP

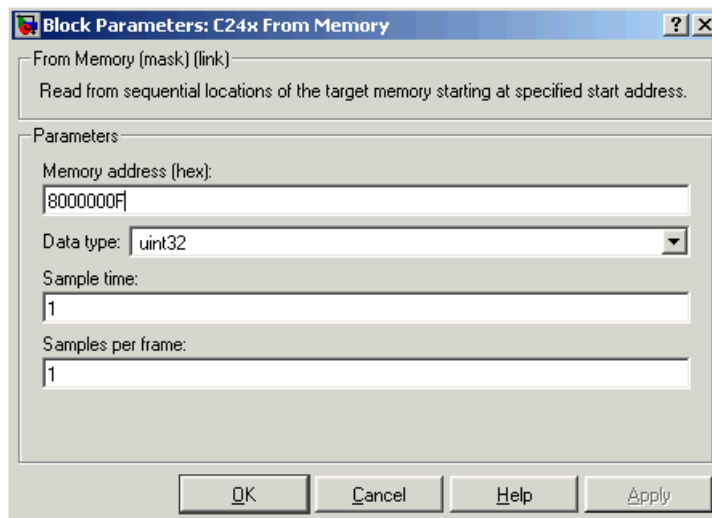
Description

This block retrieves data of the specified data type from a particular memory address on the target.



Note Although the block dialog box shown here is for the C24x, the same block and dialog box apply to the C280x and the C281x.

Dialog Box



Memory address

Address of the target memory location, in hexadecimal, from which to read data.

From Memory

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

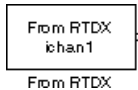
Purpose

Add RTDX input channel

Library

rtdxBlocks in Embedded Target for TI C2000 DSP

Description



When you generate code from Simulink in Real-Time Workshop with a From RTDX block in your model, code generation inserts the C commands to create an RTDX input channel on the target. Input channels transfer data from the host to the target.

The generated code contains this command:

```
RTDX_enableInput(&channelname)
```

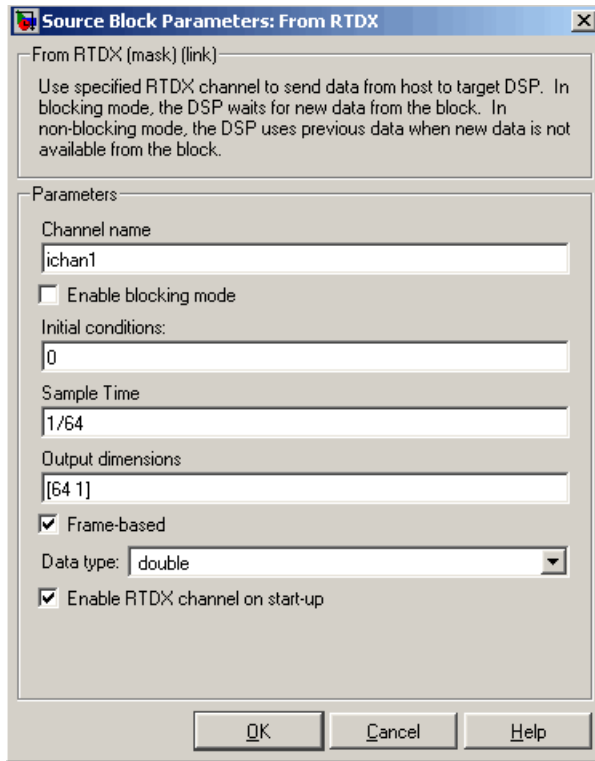
where channelname is the name you enter in **Channel name**.

Note From RTDX blocks work only in code generation and when your model runs on your target. In simulations, this block does not perform any operations, except generating an output matching your specified initial conditions.

To use RTDX blocks in your model, you must do the following:

- 1** Add one or more To RTDX or From RTDX blocks to your model.
- 2** Download and run your model on your target.
- 3** Enable the RTDX channels from MATLAB or use **Enable RTDX channel on start-up** on the block dialog.
- 4** Use the readmsg and writemsg functions in MATLAB to send and retrieve data from the target over RTDX.

Dialog Box



Channel name

Name of the input channel to be created by the generated code. The channel name must meet C syntax requirements for length and character content.

Enable blocking mode

Blocking mode instructs the target processor to pause processing until new data is available from the From RTDX block. If you enable blocking and new data is not available when the processor needs it, your process stops. In nonblocking mode, the processor uses old data from the block when new data is not available.

Nonblocking operation is the default and is recommended for most operations.

Initial conditions

Data the processor reads from RTDX for the first read. If blocking mode is not enabled, you must have an entry for this option. Leaving the option blank causes an error in Real-Time Workshop. Valid values are 0, null ([]), or a scalar. The default value is 0.

0 or null ([]) outputs a zero to the processor. A scalar generates one output sample with the value of the scalar. If **Output dimensions** specifies an array, every element in the array has the same scalar or zero value. A null array ([]) outputs a zero for every sample.

Sample time

Time between samples of the signal. The default is 1 second. This produces a sample rate of one sample per second (1/**Sample time**).

Output dimensions

Dimensions of a matrix for the output signal from the block. The first value is the number of rows and the second is the number of columns. For example, the default setting [1 64] represents a 1-by-64 matrix of output values. Enter a 1-by-2 vector for the dimensions.

Frame-based

Sets a flag at the block output that directs downstream blocks to use frame-based processing on the data from this block. In frame-based processing, the samples in a frame are processed simultaneously. In sample-based processing, samples are processed one at a time. Frame-based processing can increase the speed of your application running on your target. Note that throughput remains the same in samples per second processed. Frame-based operation is the default.

Data type

Type of data coming from the block. Select one of the following types:

- **Double** — Double-precision floating-point values. This is the default. Values range from -1 to 1.
- **Single** — Single-precision floating-point values ranging from -1 to 1.
- **Uint8** — 8-bit unsigned integers. Output values range from 0 to 255.
- **Int16** — 16-bit signed integers. With the sign, the values range from -32768 to 32767.
- **Int32** — 32-bit signed integers. Values range from -2^{31} to $(2^{31} - 1)$.

Enable RTDX channel on start-up

Enables the RTDX channel when you start the channel from MATLAB. With this selected, you do not need to use the enable function in the Link for Code Composer Studio Development Tools to prepare your RTDX channels. This option applies only to the channel you specify in **Channel name**. You do have to open the channel.

See Also

ccdsp, readmsg, To RTDX, writemsg.

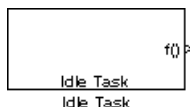
Purpose

Create free-running task that executes downstream subsystem

Library

c280xspchiplib or c281xspchiplib in Embedded Target for TI C2000 DSP

Description



The Idle Task block, and the subsystem to which it is connected, specify one or more functions to execute as background tasks. By definition, all tasks executed through the Idle Task block are of the lowest priority, lower than that of the base rate task.

Vectorized Output

The output of this block includes a set of two vectors, the **Number of tasks** and the corresponding **Preemption flag(s)**. The **Preemption flag(s)** vector must be the same length as the **Number of tasks** vector unless it has only one element.

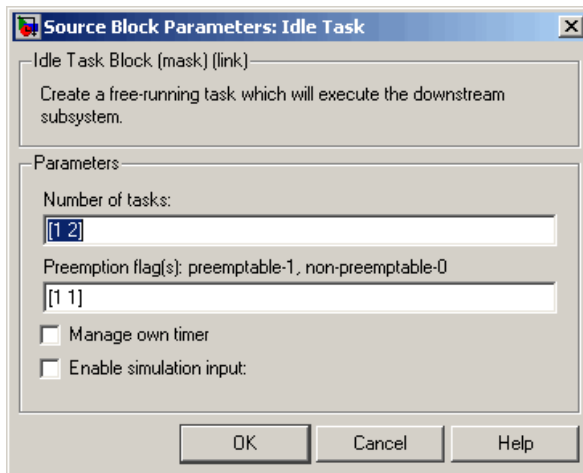
If the **Preemption flag(s)** vector does have one element, then that value applies to all functions in the downstream subsystem.

If the **Preemption flag(s)** vector has the same number of elements as the **Number of tasks** vector, then each task's preemption flag value is the value of the corresponding element in the **Preemption flag(s)** vector.

The preemption flag determines whether a given interrupt is preemptable or not. Preemption overrides prioritization, such that a preemptable task of higher priority can be preempted by a non-preemptable task of lower priority.

Idle Task

Dialog Box



Number of tasks

The values you enter determine the order in which the functions in the downstream subsystem are to be executed, while the number of values you enter corresponds to the number of functions in the downstream subsystem.

Enter a vector containing the same number of elements as the number of functions in the downstream subsystem. This vector can contain no more than 16 elements, and the values must be from 0 to 15 inclusive.

The value of the first element in the vector determines the order in which the first function in the subsystem will be executed, and so on.

For example, if you enter `[2, 3, 1]` in this field, you are indicating that there are three functions to be executed, and that the third function will be executed first, the first function will be executed second, and the second function will be executed third.

When all functions have been executed, the Idle Task block cycles back and repeats the execution of the functions in the same order.

Preemption flag(s)

The preemption flag determines whether a given interrupt is preemptable or not. Preemption overrides prioritization, so if you flag one of these functions as non-preemptable, its execution will not be suspended by another task even though the functions in the downstream subsystem all have the lowest priority by definition.

Enter either a vector of one element, in which case that preemption flag applies to all functions to be executed in the downstream subsystem, or a vector containing the same number of elements as the **Number of tasks** vector, in which case each preemption flag value applies to the task number in the corresponding position within its vector. All preemption flag values must be either 0 (non-preemptable) or 1 (preemptable).

Manage own timer

Some Simulink blocks need to keep track of time in order to function properly. Select this check box if your model contains such a block in the downstream subsystem.

Enable simulation input

Select this check box to make it possible to test asynchronous interrupt processing in the context of your Simulink model.

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

See Also

C280x Hardware Interrupt, C281x Hardware Interrupt

Integer part IQN

Purpose

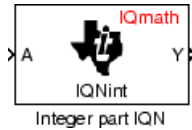
Integer part of IQ number

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

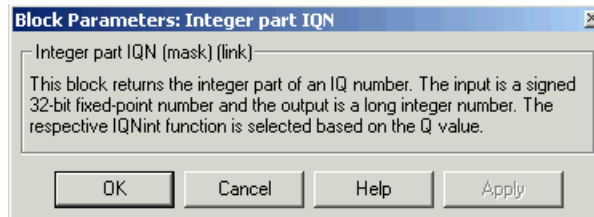
Description

This block returns the integer portion of an IQ number. The returned value is a long integer.



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

Dialog Box



See Also

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

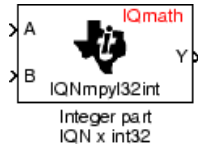
Purpose

Integer part of result of multiplying IQ number and long integer

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

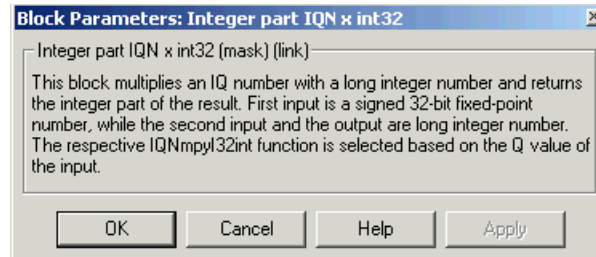
Description



This block multiplies an IQ input and a long integer input and returns the integer portion of the resulting IQ number as a long integer.

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

Dialog Box



See Also

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

Inverse Park Transformation

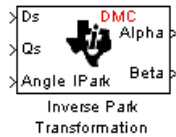
Purpose

Convert rotating reference frame vectors to two-phase stationary reference frame

Library

c28xdmclib in Embedded Target for TI C2000 DSP

Description

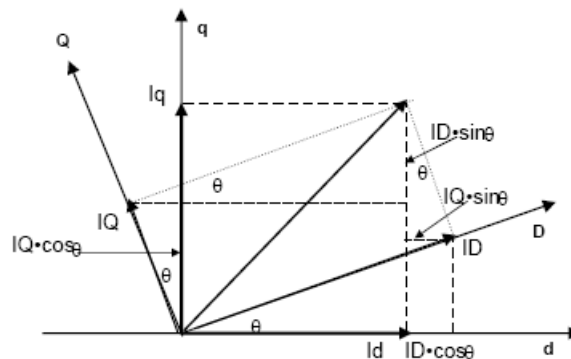


This block converts vectors in an orthogonal rotating reference frame to a two-phase orthogonal stationary reference frame. The transformation implements these equations:

$$I_d = I_D * \cos \theta - I_Q * \sin \theta$$

$$I_q = I_D * \sin \theta + I_Q * \cos \theta$$

and is illustrated in the following figure.



The inputs to this block are the direct axis (D_s) and quadrature axis (Q_s) components of the transformed signal in the rotating frame and the phase angle ($Angle$) between the stationary and rotating frames.

The outputs are the direct axis ($Alpha$) and the quadrature axis ($Beta$) components of the transformed signal.

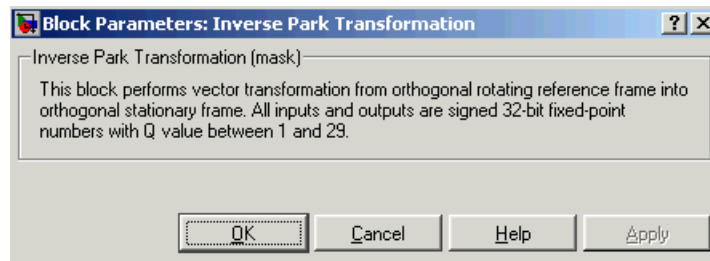
The variables used in the preceding figure and equations correspond to the block variables as shown in the following table:

Inverse Park Transformation

	Equation Variables	Block Variables
Inputs	ID	Ds
	IQ	Qs
	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 “About the IQmath Library” on page 5-2 for more information.

Dialog Box



References

Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

Clarke Transformation, Park Transformation, PID Controller, Space Vector Generator, Speed Measurement

IQN to Float

Purpose

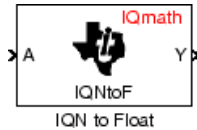
Convert IQ number to floating-point number

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

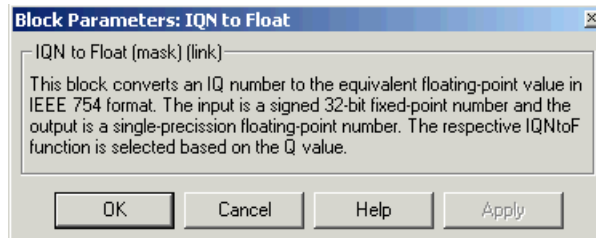
Description

This block converts an IQ input to an equivalent floating-point number. The output is a single floating-point number.



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

Dialog Box



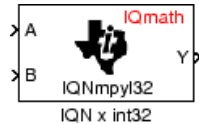
See Also

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

Purpose Multiply IQ number with long integer

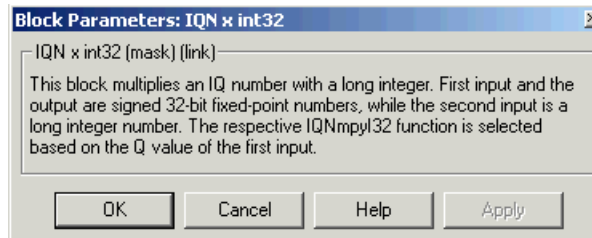
Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block multiplies an IQ input and a long integer input and produces an IQ output of the same Q value as the IQ input.



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

Dialog Box



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

IQN x IQN

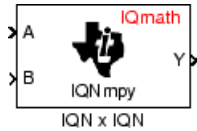
Purpose

Multiply two IQ numbers with same Q format

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

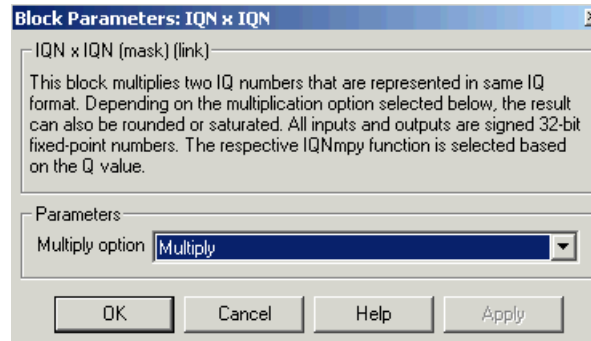
Description



This block multiplies two IQ numbers. Optionally, it can also round and saturate the result.

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

Dialog Box



Multiply option

Type of multiplication to perform:

- **Multiply** — Multiply the numbers.
- **Multiply with Rounding** — Multiply the numbers and round the result.
- **Multiply with Rounding and Saturation** — Multiply the numbers and round and saturate the result to the maximum value.

See Also

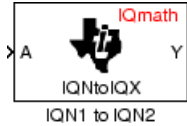
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

IQN1 to IQN2

Purpose Convert IQ number to different Q format

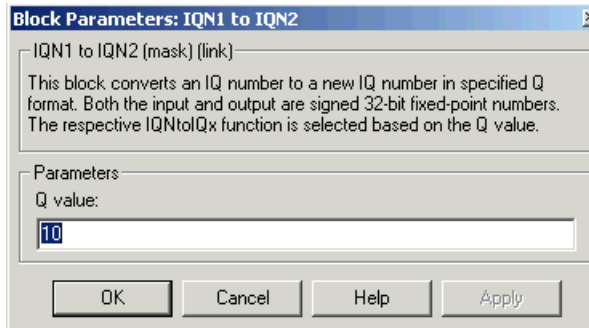
Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block converts an IQ number in a particular Q format to a different Q format.



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

Dialog Box



Q value

Q value from 1 to 30 that specifies the precision of the output

See Also

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

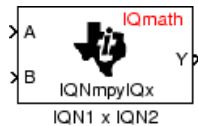
Purpose

Multiply two IQ numbers with different Q formats

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

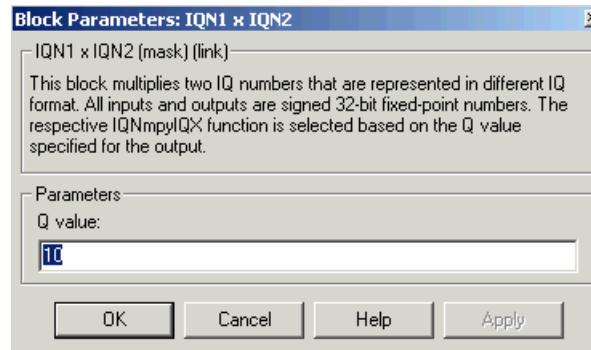
Description



This block multiplies two IQ numbers when the numbers are represented in different Q formats. The format of the result is specified in the dialog box.

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

Dialog Box



Q value

Q value from 1 to 30 that specifies the precision of the output

See Also

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

LF2407 eZdsp

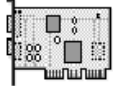
Purpose

LF2407 eZdsp DSK target preferences

Library

c2000tgtpref1lib in Embedded Target for TI C2000 DSP

Description

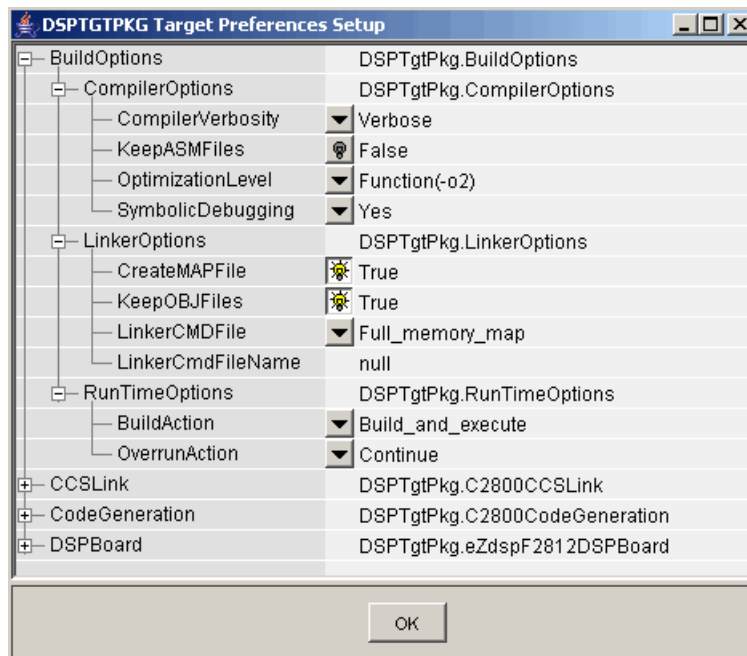


LF2407 eZdsp

Options on the block mask let you set features of code generation for your Spectrum Digital LF2407 eZdsp target. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

Note This block must be in your model at the top level and not in a subsystem. It does not connect to any other blocks, but stands alone to set the target preferences for the model.

Dialog Box



BuildOptions – CompilerOptions

Compiler Verbosity

Amount of information the compiler returns while it runs. Options are

- `Verbose` — Returns all compiler messages.
- `Quiet` — Suppresses compiler progress messages.
- `Super_quiet` — Suppresses all compiler messages.

KeepASMFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your assembly language (`.asm`) files after creation. The default is `false`—`.asm` files are not kept in your current directory. If you choose to keep the `.asm` files, set this option to `true`.

OptimizationLevel

Degree of optimization provided by the TI optimizing compiler to apply to files in your project. For details about the compiler options, refer to your CCS documentation. When you create new projects, the Embedded Target for TI C2000 DSP sets the optimization to `Function(-o2)`.

SymbolicDebugging

Whether to generate symbolic debugging directives that the C source-level debugger uses and whether to enable assembly source debugging. By default, this option is `Yes`—symbolic debugging is enabled.

BuildOptions – LinkerOptions

CreateMAPFile

Whether the linker produces a map of the input and output sections, including null areas, and places the listing in a file in your current directory with the name `modelName.map`. The default is `True`—the listing is produced.

KeepOBJFiles

Whether Real-Time Workshop and the Embedded Target for TI C2000 DSP save your object (.obj) files after creation. The linker uses object (.obj extension) files to generate a single executable common object file format (COFF) file that you run on the target DSP. The object files are saved to your current project directory. Saving your .obj files can speed up the compile process by not having to recompile files that you have not changed. The default is True—the .obj files are retained.

LinkerCMDFile

Type of linker command file to use when the linker runs. Linker command files contain linker or hex conversion utility options and the names of input files to the linker or hex conversion utility. Linker command file types are

- `Internal_memory_map` — Although this option is supported, only very small programs that fit in the internal chip memory can be used. If your program is too large, a linker error occurs. In general, you should use `Full_memory_map` or `Custom_file`.
- `Full_memory_map` — Uses the large memory model on the target, which does not restrict the size of the code and data sections to DSP memory only. Your data can use the storage space up to the limits of the board.
- `Custom_file` — Uses the file in the **LinkerCmdFileName** field. This option lets you target custom boards. You must specify the full path of the file. Note that the software does not verify that the commands in this file are correct. If you use `Internal_memory_map` or `Full_memory_map`, specifying a `Custom_file` has no effect.

When you select the `Internal_memory_map` option, the Embedded Target for TI C2000 DSP specifies that only the available internal memory on the LF2407 is used.

If you select `Internal_memory_map`, but your data or program requires far calls, the TI compiler returns an error message like the following in the CCS IDE:

```
error: can't allocate '.far'
or
error: can't allocate '.text'
```

indicating that your data does not fit in internal memory or your code or program do not fit in internal memory. To eliminate these errors, select `Full_memory_map`. Your program might run more slowly than if you use the internal map option.

BuildOptions – RunTimeOptions

BuildAction

Action taken by Real-Time Workshop when you click **Build** or press **Ctrl+B** in the Simulation Parameters dialog box. The actions are cumulative—each listed action adds features to the previous action on the list and includes all the previous features:

- `Generate_code_only` — Directs Real-Time Workshop to generate C code only from the model. It does not use the TI software tools, such as the compiler and linker, and you do not need to have CCS installed. Also, MATLAB does not create the handle to CCS that results from the other options.

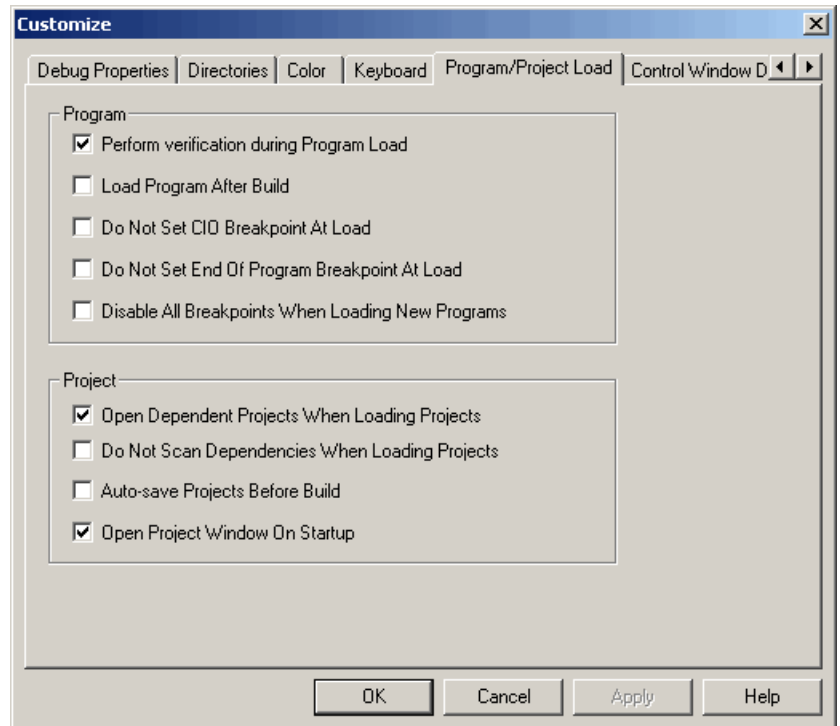
The build process for a model also generates the files `modelName.c`, `modelName.cmd`, `modelName.bld`, and many others. It puts the files in a build directory named `modelName_C2000_rtw` in your MATLAB working directory. This file set contains many of the same files that Real-Time Workshop generates to populate a CCS project when you choose `Create_CCS_Project` for the build action.

- `Create_CCS_Project` — Directs Real-Time Workshop to start CCS and populate a new project with the files from the build process. Selecting this setting enables the CCS board number option so you can select which installed board to target. This option offers a convenient way to build projects in CCS.
- `Build` — Builds the executable COFF file, but does not download the file to the target.

- **Build_and_execute** — Directs Real-Time Workshop to download and run your generated code as an executable on your target. This option is the default.

Setting this option requires corresponding options to be set in the CCS:

- From the CCS window, select **Option > Customize...****
- Select the **Program/Project Load** tab on the **Customize** dialog box as shown in the following figure:**



- Uncheck the **Load Program After Build** option.**

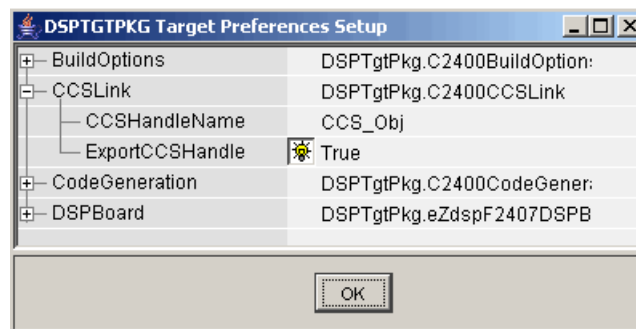
Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

OverrunAction

Defines the action to take when an interrupt overrun occurs.

- Continue — Ignore overruns encountered while running the model. This option is the default.
- Halt — Stop program execution.

CCSLink



CCSHandleName

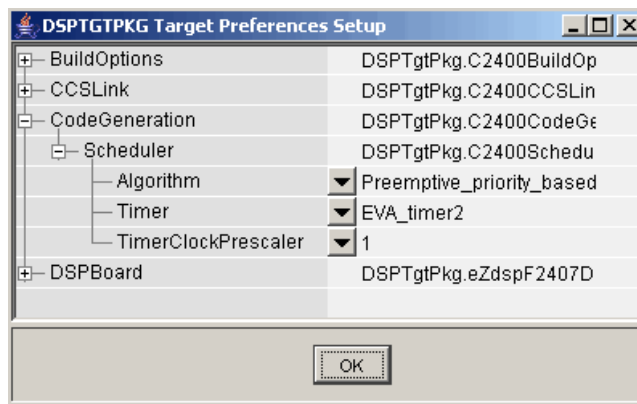
Name of the CCS handle. Click in the edit box to change the name. When you use Real-Time Workshop to build a model for a C2000 target, Embedded Target for TI C2000 DSP makes a link between MATLAB and CCS. If you have used the link portion of the Embedded Target for TI C2000 DSP, you are familiar with function `ccsdsp`, which creates links between the IDE and MATLAB. This option refers to the same link, called `cc` in the function reference pages. Although MATLAB to CCS is a link, it

is actually a handle to an object that contains information about the object, such as the target board and processor it accesses.

ExportCCSHandle

Whether to export the CCS handle to your MATLAB workspace, giving it the name you assigned in **CCSHandleName**. If this parameter is set to True, after you build your model, you see the CCS object in your MATLAB workspace browser with the name you provided and class type `ccsdsp`.

CodeGeneration – Scheduler



Algorithm

Algorithm to use for scheduling. Currently, the only algorithm option is `Preemptive_priority_based`. This scheduler runs based on the timer interrupt. The timer period is set based on the base rate sample time you specify for your model. This algorithm supports multirate systems in multitasking mode with priority-based preemption. The task for the fastest group (the base rate task) runs first and other tasks run in the order determined by their sample rates from faster tasks to slower tasks. For more information, see “Models with Multiple Sample Rates” in the “Real-Time Workshop” documentation.

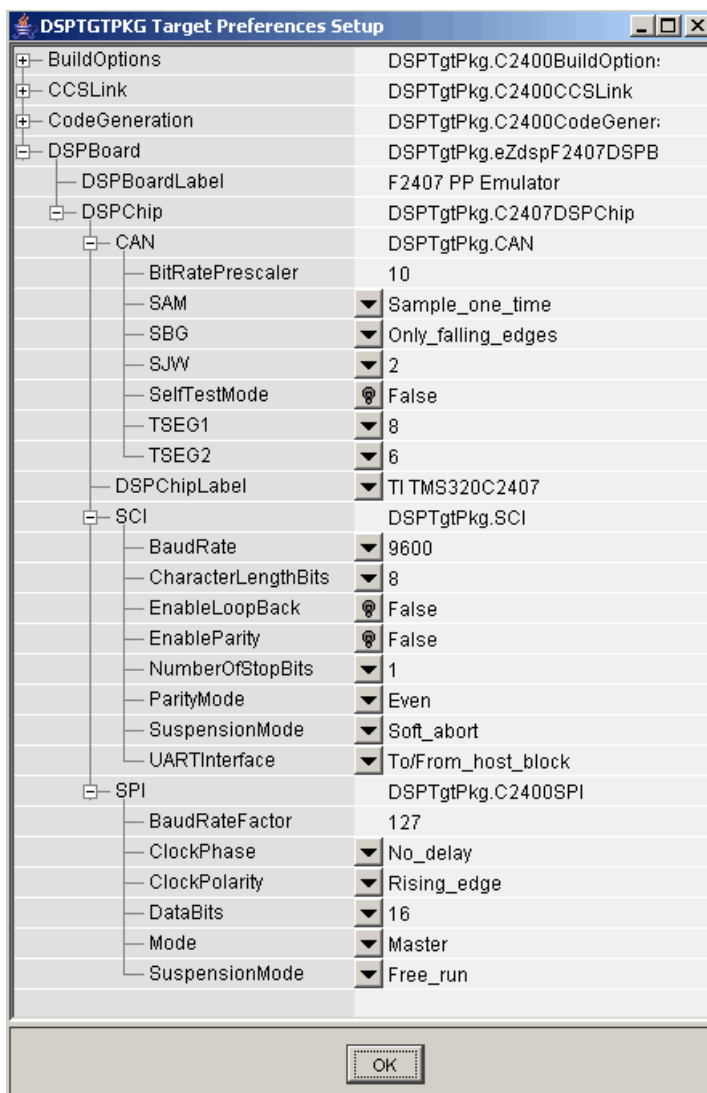
Timer

Event Manager (EV) timer to use for scheduling.

TimerClockPrescaler

Clock divider factor by which to prescale the selected timer to produce the desired model rate. The system clock for the TMS320LF2407 DSP is 40 MHz.

DSPBoard



DSPBoardLabel

Name of the installed DSP board. Click in the edit box to change the label.

Note The board label here must match exactly the label (name) of the board entered in your Code Composer Studio setup.

DSPBoard – DSPChip

CAN

Parameters that affect the control area network (CAN) module. Most of these parameters affect the CAN bit timing. For more information see “Configuring Timing Parameters for CAN Blocks”

The CAN parameters you can set are:

BaudRatePrescaler

Value by which to scale the baud rate. Valid values are from 1 to 256.

SAM

Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of $TQ/2$. A majority decision is made from the three points.

SBG

Sets the message resynchronization triggering. Options are `Only_falling_edges` and `Both_falling_and_rising_edges`.

SJW

Sets the synchronization jump width, which determines how many units of TQ a bit is allowed to be shortened or lengthened when resynchronizing.

SelfTestMode

If this parameter is set to True, the CAN module goes to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit.

TSEG1

Sets the value of time segment 1, which, with TSEG2 and BRP, determines the length of a bit on the CAN bus. Valid values for TSEG1 are from 1 through 16.

TSEG2

Sets the value of time segment 2, which, with TSEG1 and BRP, determines the length of a bit on the CAN bus. Valid values for TSEG2 are from 1 through 8.

DSP Chip Label

DSP chip model. Select the DSP chip installed on your target. The chip model is fixed for the LF2407 eZdsp. If you change the chip model, an error is generated during code generation.

SCI

Parameters that affect the serial communications interfaces (SCI) on the target.

The SCI parameters you can set are:

BaudRate

Baud rate for transmitting and receiving data. Choices are 115200 (the default), 57600, 38400, 19200, 9600, 4800, 2400, 1200, 300, and 110.

CharacterLengthBits

Length in bits of each transmitted or received character, set to 8 bits.

EnableLoopBack

Select True to enable the loopback function for self-test and diagnostic purposes only. When this function is enabled, a C24x DSP's Tx pin is internally connected to its Rx pin and can transmit data from its output port to its input port to check the integrity of the transmission.

EnableParity

Select True to enable parity checking on the transmit or receive data.

NumberOfStopBits

Select whether to use 1 or 2 stop bits.

ParityMode

Type of parity to use. Available selections are None, Odd parity, or Even parity. None disables parity. Odd sets the parity bit to one if you have an odd number of ones in your bytes (e.g., 00110010). Even sets the parity bit to one if you have an even number of ones in your bytes (e.g., 00110011).

EnableParity must be set to True to use the selected **ParityMode**.

SuspensionMode

Type of suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are Hard abort, Soft abort, and Free run. Hard abort stops the program immediately. Soft abort stops when the current receive or transmit sequence is complete. Free run continues running regardless of the breakpoint.

SPI

Parameters that affect the serial peripheral interfaces (SPI) on the target.

You can set the following parameters:

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$\text{Baud Rate} = \text{CPU Rate} / (\text{Baud Rate Factor} + 1)$$

ClockPhase

Whether the data is output immediately (`No_delay`) or delayed by a half clock cycle (`Delay_half_cycle`) with respect to the rising edge.

ClockPolarity

Whether the data is output at the `Rising_edge` or `Falling_edge` of the system clock.

DataBits

Length in bits from 1 to 16 of each transmitted or received character. For example, if you select 8, the maximum data that can be transmitted using SPI is 2^{8-1} . If you send data greater than this value, the buffer overflows.

Mode

Whether to run the SPI module in Master or Slave mode. Master mode initiates the transmission. Slave mode is triggered by another master SPI and is synchronized to the clock used by the master SPI. This option cannot be changed at run time.

SuspensionMode

Suspension to use when debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. Available options are `Hard abort`, `Soft abort`, and `Free run`. `Hard abort` stops the program immediately. `Soft abort` stops when the current receive or transmit sequence is complete. `Free run` continues running regardless of the breakpoint.

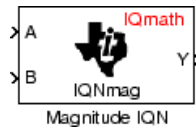
See Also

C24x ADC, C24x CAN Receive, C24x CAN Transmit, C24x PWM

Purpose Magnitude of two orthogonal IQ numbers

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block calculates the magnitude of two IQ numbers using

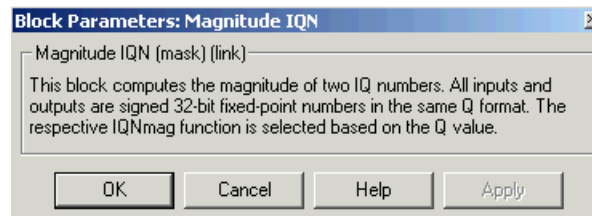


$$\sqrt{a^2 + b^2}$$

The output is an IQ number in the same Q format as the input.

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

Dialog Box



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

Park Transformation

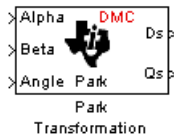
Purpose

Convert two-phase stationary system vectors to rotating system vectors

Library

c28xdmclib in Embedded Target for TI C2000 DSP

Description

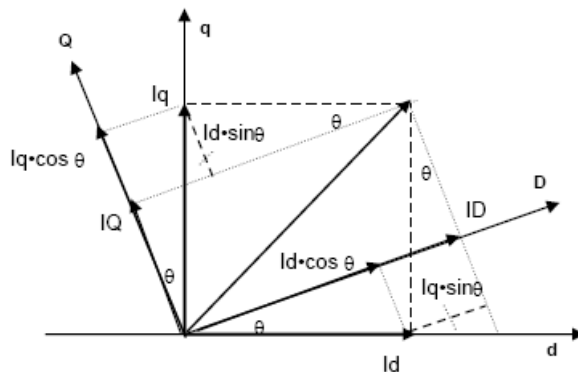


This block converts vectors in balanced two-phase orthogonal stationary systems into an orthogonal rotating reference frame. The transformation implements these equations

$$ID = Id * \cos \theta + Iq * \sin \theta$$

$$IQ = -Id * \sin \theta + Iq * \cos \theta$$

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

	Equation Variables	Block Variables
Outputs	ID	Ds
	IQ	Qs

The inputs to this block are the direct axis (Alpha) and the quadrature axis (Beta) components of the transformed signal and the phase angle (Angle) between the stationary and rotating frames.

The outputs are the direct axis (Ds) and quadrature axis (Qs) components of the transformed signal in the rotating frame.

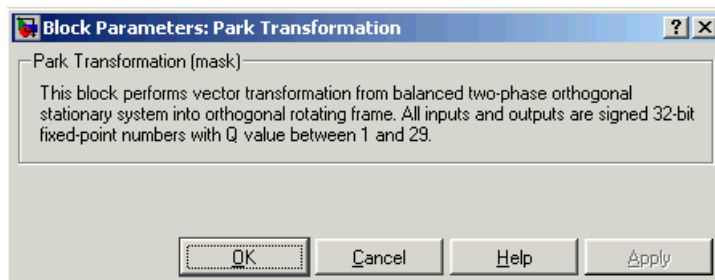
The instantaneous inputs are defined by the following equations:

$$id = I * \sin(\omega t)$$

$$iq = I * \sin(\omega t + \pi/2)$$

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

Dialog Box



Park Transformation

References

Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

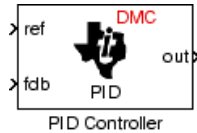
See Also

Clarke Transformation, Inverse Park Transformation, PID Controller, Space Vector Generator, Speed Measurement

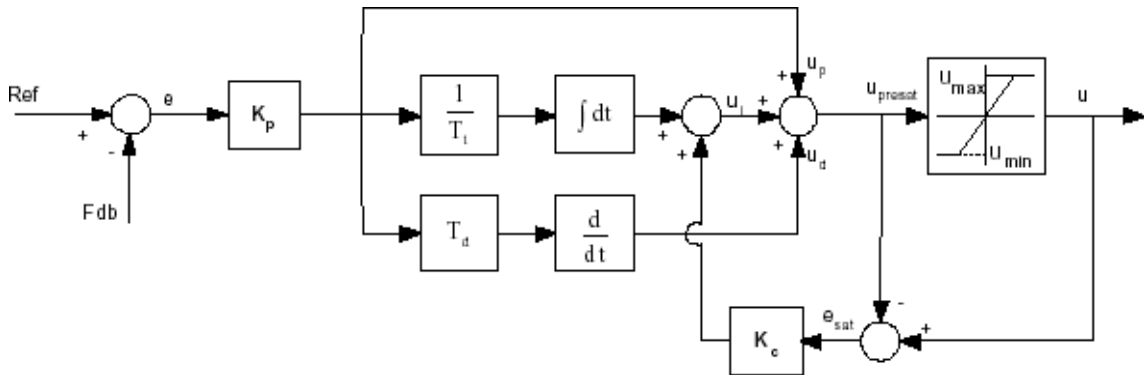
Purpose Digital PID controller

Library c28xdmclib in Embedded Target for TI C2000 DSP

Description



This block implements a 32-bit digital PID controller with antiwindup correction. The inputs are a reference input (ref) and a feedback input (fdb) and the output (out) is the saturated PID output. The following diagram shows a PID controller with antiwindup.



The differential equation describing the PID controller before saturation that is implemented in this block is

$$u_{presat}(t) = u_p(t) + u_i(t) + u_d(t)$$

where u_{presat} is the PID output before saturation, u_p is the proportional term, u_i is the integral term with saturation correction, and u_d is the derivative term.

The proportional term is

$$u_p(t) = K_p e(t)$$

where K_p is the proportional gain of the PID controller and $e(t)$ is the error between the reference and feedback inputs.

PID Controller

The integral term with saturation correction is

$$u_i(t) = \frac{K_p}{T_i} \int_0^t e(\zeta) d\zeta + K_c(u(t) - u_{presat}(t))$$

where K_c is the integral correction gain of the PID controller.

The derivative term is

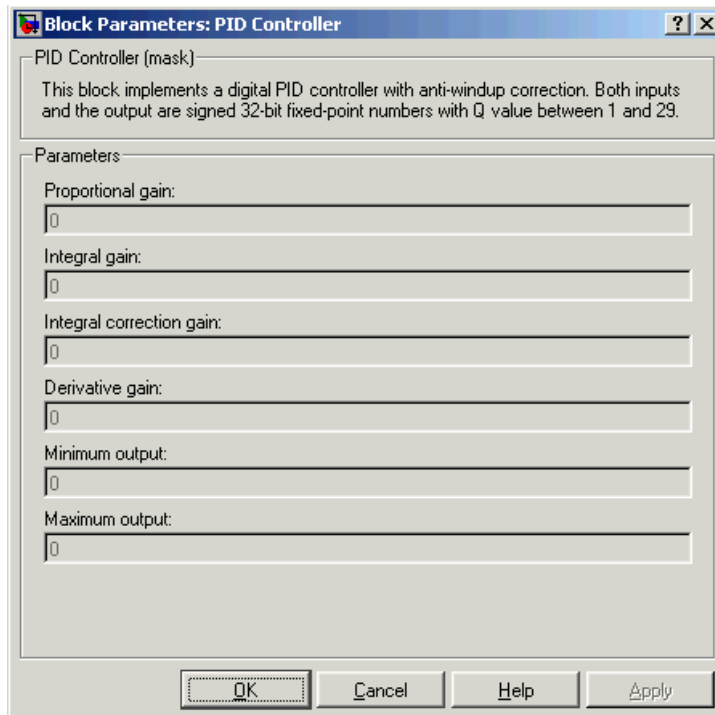
$$u_d(t) = K_p T_d \frac{de(t)}{dt}$$

where T_d is the derivative time of the PID controller. In discrete terms, the derivative gain is defined as $K_d = T_d/T$, and the integral gain is defined as $K_i = T/T_i$, where T is the sampling period and T_i is the integral time of the PID controller.

The above differential equations are transformed into a difference equations by backward approximation.

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

Dialog Box



Proportional gain

Amount of proportional gain (K_p) to apply to the PID

Integral gain

Amount of gain (K_i) to apply to the integration equation

Integral correction gain

Amount of correction gain (K_c) to apply to the integration equation

Derivative gain

Amount of gain (K_d) to apply to the derivative equation.

Minimum output

Minimum allowable value of the PID output

PID Controller

Maximum output

Maximum allowable value of the PID output

References

Detailed information on the DMC library is in the *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

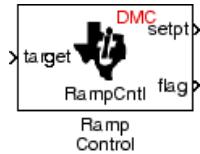
Purpose

Create ramp-up and ramp-down function

Library

c28xdmclib in Embedded Target for TI C2000 DSP

Description

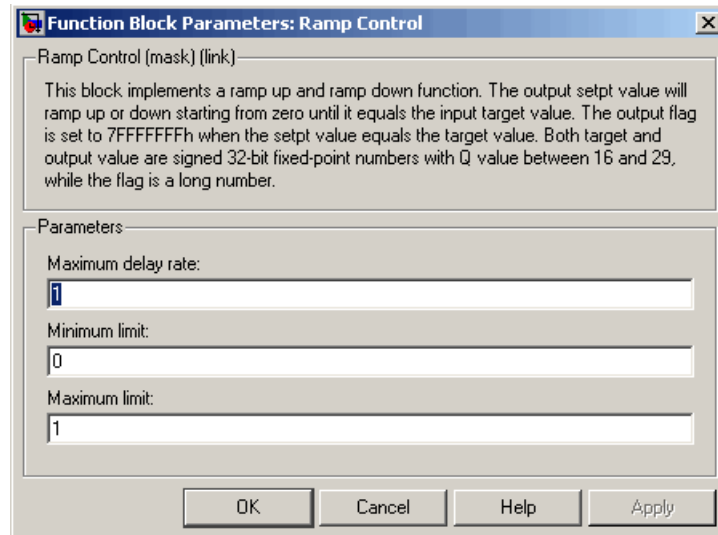


This block implements a ramp-up and ramp-down function. The input is a target value and the outputs are the set point value (setpt) and a flag. The flag output is set to 7FFFFFFFh when the output setpt value reaches the input target value. The target and setpt values are signed 32-bit fixed-point numbers 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



Ramp Control

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

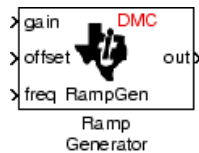
Purpose

Generate ramp output

Library

c28xdmclib in Embedded Target for TI C2000 DSP

Description



This block generates ramp output (out) from the slope of the ramp signal (gain), DC offset in the ramp signal (offset), and frequency of the ramp signal (freq) inputs. All of the inputs and output are 32-bit fixed-point numbers with Q values between 1 and 29.

Algorithm

The block's output (out) at the sampling instant k is governed by the following algorithm:

$$\text{out}(k) = \text{angle}(k) * \text{gain}(k) + \text{offset}(k) "$$

For $\text{out}(k) > 1$, $\text{out}(k) = \text{out}(k) - 1$. For $\text{out}(k) < -1$, $\text{out}(k) = \text{out}(k) + 1$.

Angle(k) is defined as follows:

$$\text{angle}(k) = \text{angle}(k-1) + \text{freq}(k) * \text{Maximum step angle}$$

$$\text{for } \text{angle}(k) > 1, \text{angle}(k) = \text{angle}(k) - 1$$

$$\text{for } \text{angle}(k) < -1, \text{angle}(k) = \text{angle}(k) + 1"$$

The frequency of the ramp output is controlled by a precision frequency generation algorithm that relies on the modulo nature of the finite length variables. The frequency of the output ramp signal is equal to

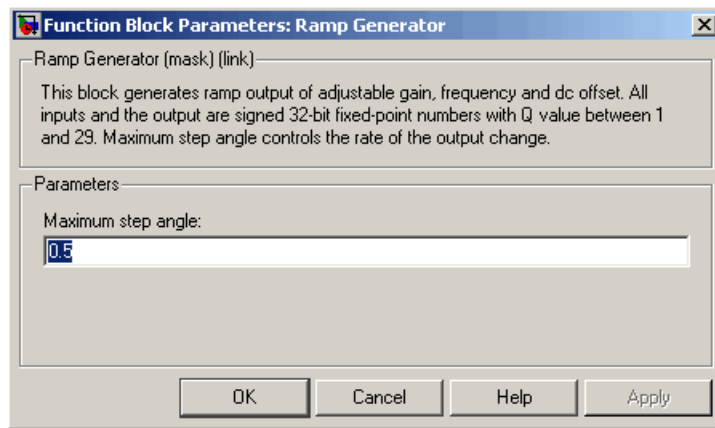
$$f = (\text{Maximum step angle} * \text{sampling rate}) / 2^m "$$

where m represents the fractional length of the data type of the inputs.

All math operations are carried out in fixed-point arithmetic, where the fixed-point fractional length is determined by the block's inputs.

Ramp Generator

Dialog Box

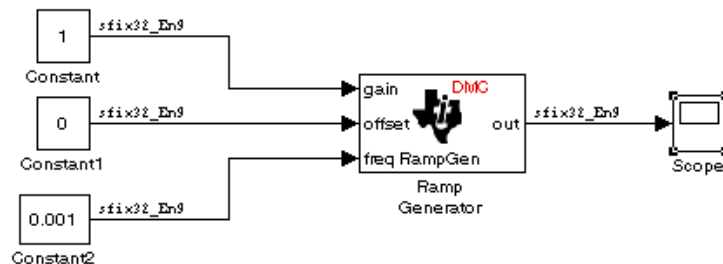


Maximum step angle

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.



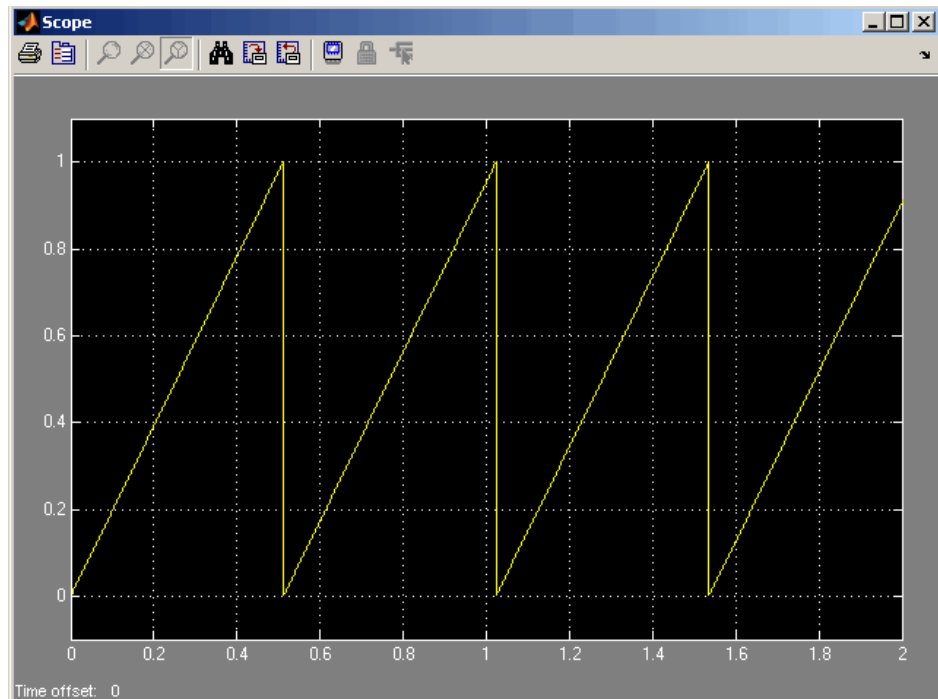
In your model, select **Simulation > Configuration Parameters**. On the **Solver** pane, set **Type** to Fixed-step and **Solver** to discrete (no continuous states). Set the parameter values for the blocks as shown in the following table.

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 (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).

Ramp Generator



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

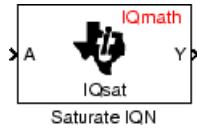
Purpose

Saturate IQ number

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

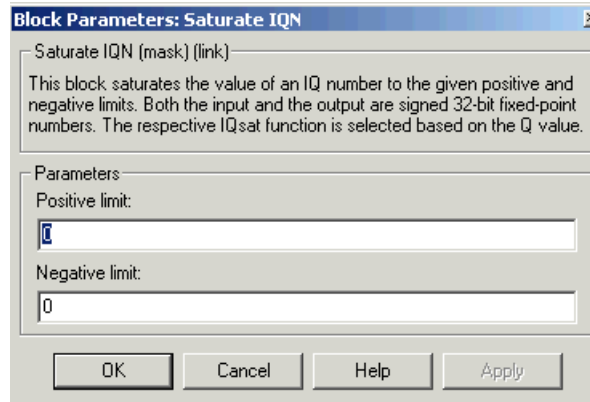
Description



This block saturates an input IQ number to the specified positive and negative limits. The returned value is an IQ number of the same Q value as the input.

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

Dialog Box



Positive Limit

Maximum positive value to which to saturate

Negative Limit

Minimum negative value to which to saturate

Saturate IQN

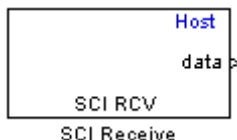
See Also

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

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

Library c2000scilib in Embedded Target for TI C2000 DSP

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 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 listed in the following table:

Data Type	Byte Count
single	4 bytes
int8 & uint8	1 byte
int16 & uint16	2 bytes
int32 & 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 two bytes left over. Even though you could fit two int8s or one uint16 in the remaining space, you may not, as you cannot mix data types in the same package.

SCI Receive

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

When connection timeout: 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).

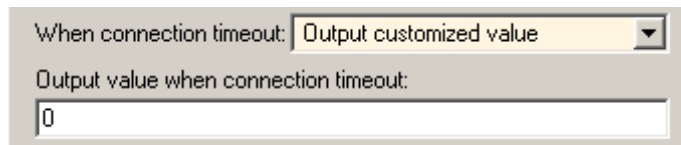
Initial output

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

When connection timeout

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

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

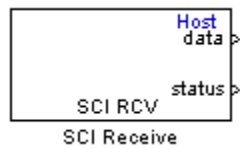


Sample time

Determines how often the SCI Receive block is called (in seconds). A value of -1 indicates the time is inherited from the model parameters. To execute this block asynchronously, set **Sample Time** to -1, and refer to “Asynchronous Interrupt Processing” on page 1-13 for a discussion of block placement and other necessary settings.

Output receiving status

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



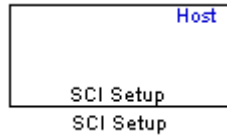
The error status may be one of the following values:

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

SCI Setup

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

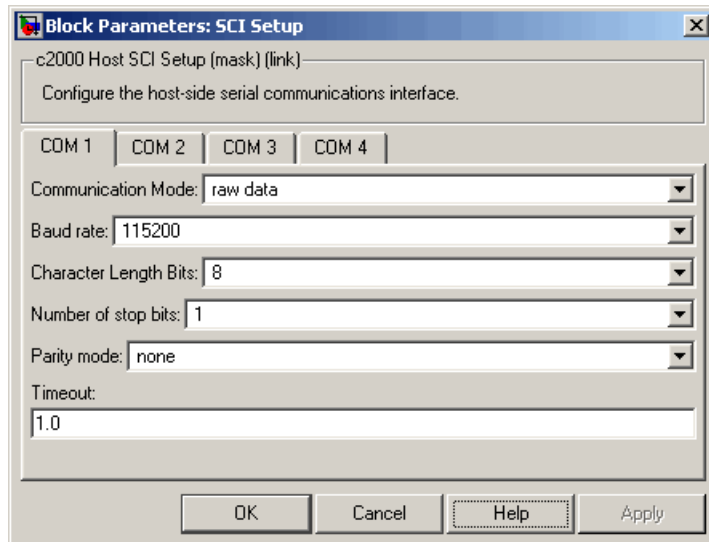
Library c2000scilib in Embedded Target for TI C2000 DSP



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.

Character Length Bits

Choose from 5, 6, 7, or 8.

Number of stop bits

Select 1 or 2.

Parity mode

Select none, odd, or even.

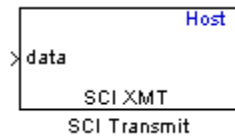
Timeout

Enter any value greater than or equal to 0, in seconds. When the COM port involved is using protocol mode, this value indicates how long the transmitting side waits for an acknowledgement from the receiving side or how long the receiving side waits for data. The system displays a warning message if the time-out is exceeded, every n number of seconds, n being the value in **Timeout**.

Note Simulink actually suspends processing for the length of the time-out, and you will not be able to perform any Simulink action. If the time-out is set for a long period of time, it may appear that Simulink has frozen.

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

Library c2000scilib in Embedded Target for TI C2000 DSP



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:

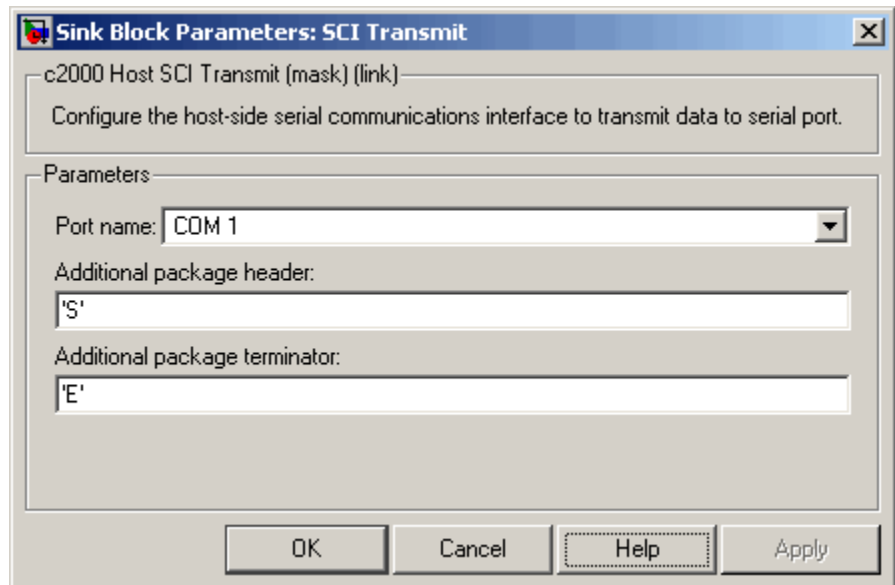
Data Type	Byte Count
single	4 bytes
int8 & uint8	1 byte
int16 & uint16	2 bytes
int32 & uint32	4 bytes

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 only 7 uint16s. If your data is of type int32, there is room in the data package for only 3 int32s, with two bytes left over. Even though you could fit two int8s or one uint16 in the remaining space, you may not, as you cannot mix data types in the same package.

SCI Transmit

The number of data types that can fit into a data package determine the data length (see **Data length** in the Dialog Box description). In the example just given, the 14 for data type int8 and the 7 for data type uint16 are the data lengths for each data package, respectively. When the data length exceeds 16 bytes, unexpected behavior, including run time errors, may result.

Dialog Box



Port name

You may configure up to four COM ports (COM1 through COM4) for up to four host-side SCI Transmit blocks.

Additional package header

This field specifies the data located at the front of the transmitted data package, which is not part of the data being transmitted, and generally indicates start of data. The additional package header must be an ASCII value. You may use any string or number (0–255). You must put single quotes around strings entered in

this field, but the quotes are not sent nor are they included in the total byte count.

Note Any additional packager header or terminator must match the additional package header or terminator specified in the target SCI receive block.

Additional package terminator

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

Space Vector Generator

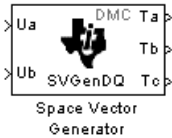
Purpose

Duty ratios for stator reference voltage

Library

c28xdmclib in Embedded Target for TI C2000 DSP

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 pseudo-sinusoidal currents in the stator phases. This technique approximates a given stator reference voltage vector by combining the switching pattern corresponding to the basic space vectors.

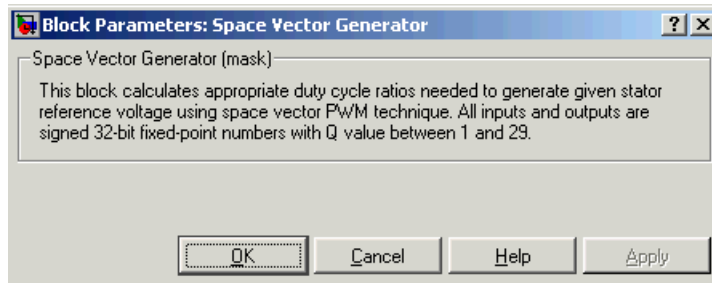
The inputs to this block are

- Alpha component — the reference stator voltage vector on the direct axis stationary reference frame (U_a)
- Beta component — the reference stator voltage vector on the direct axis quadrature reference frame (U_b)

The alpha and beta components are transformed via the inverse Clarke equation and projected into reference phase voltages. These voltages are represented in the outputs as the duty ratios of the PWM1 (T_a), PWM3 (T_b), and PWM5 (T_c).

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

Dialog Box



References

Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

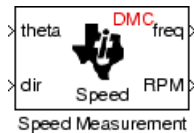
Clarke Transformation, Inverse Park Transformation, Park Transformation, PID Controller, Speed Measurement

Speed Measurement

Purpose Motor speed

Library c28xdmclib in Embedded Target for TI C2000 DSP

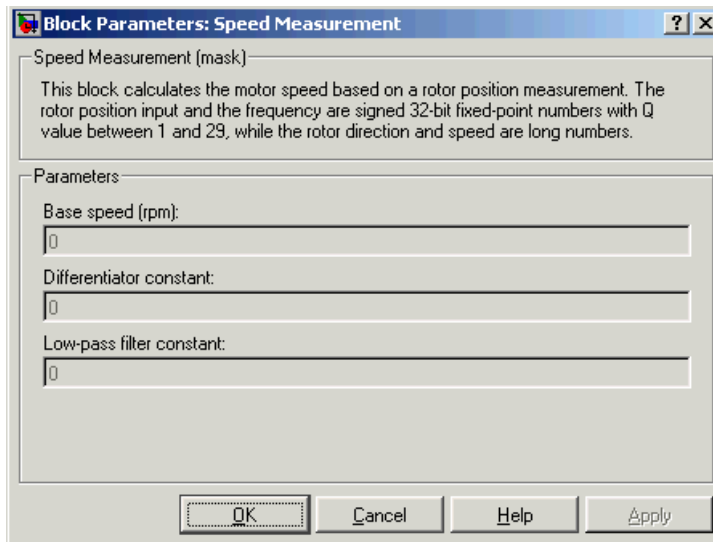
Description



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 QEP encoder. The outputs are the speed in per-unit frequency (*freq*) and the speed in revolutions per minute (*rpm*).

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

Dialog Box



Base speed

Nominal speed of the machine in rpm.

Differentiator constant

Constant used in the differentiator equation that describes the rotor position.

Low-pass filter constant

Constant to apply to the low-pass 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 low-pass filter time constant. A low-pass filter is used in this block to reduce amplifying noise generated by the differentiator.

References

Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

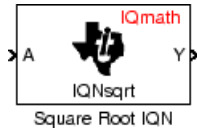
Clarke Transformation, Inverse Park Transformation, Park Transformation, PID Controller, Space Vector Generator

Square Root IQN

Purpose Square root or inverse square root of IQ number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

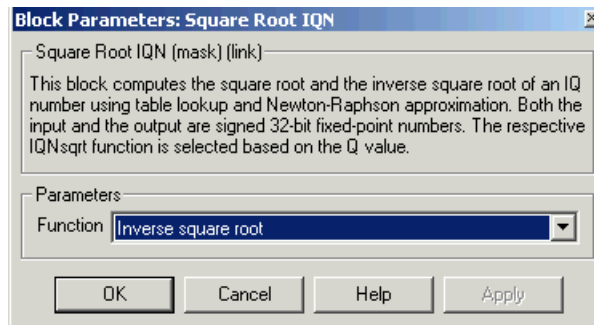


This block calculates the square root or inverse square root of an IQ number and returns an IQ number of the same Q format. The block uses table lookup and a Newton-Raphson approximation.

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

Note Negative inputs to this block return a value of zero.

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.

See Also

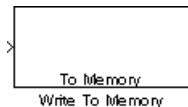
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Trig Fcn IQN

To Memory

Purpose Write data to target memory

Library c2400spchiplib or c280xspchiplib or c281xspchiplib in Embedded Target for TI C2000 DSP

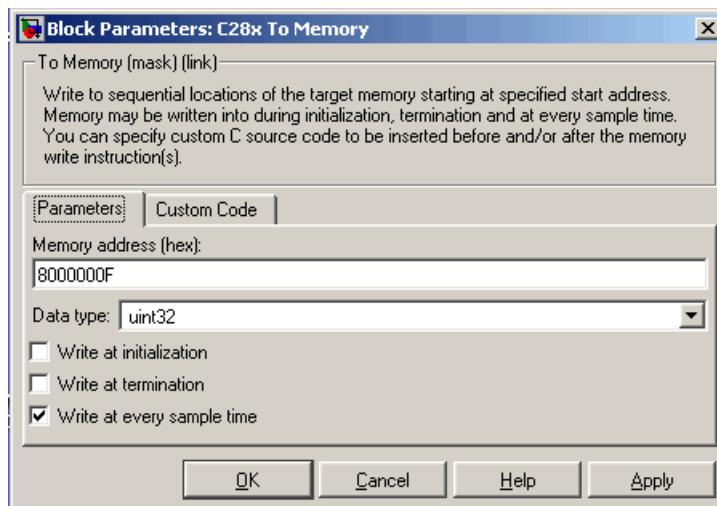
Description This block sends data of the specified data type to a particular memory address on the target.



Note Although the block dialog box shown here is for the C24x, the same block and dialog box apply to the C280x and the C281x.

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, int32, and uint32. The data is cast from the selected data type to 16-bit data.

Write at initialization

Whether to write the specified **Value** at program start

Value

First value of data to be written to memory at program start

Write at termination

Whether to write the specified **Value** at program end

Value

Last value of data to be written to memory at program termination

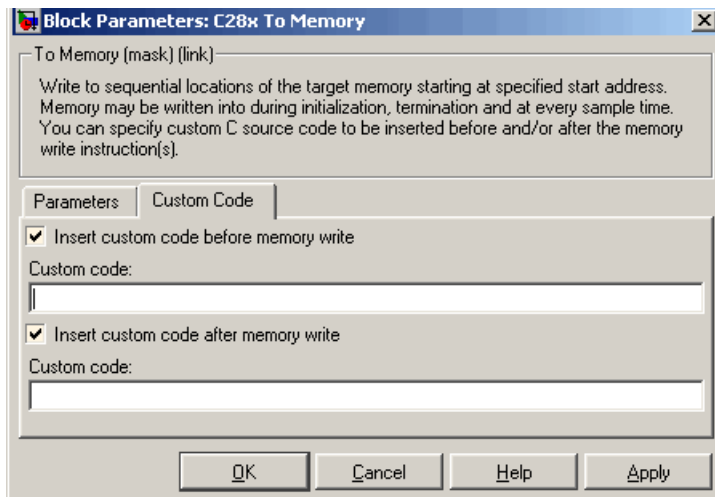
Write at every sample time

Whether to write data in real time during program execution

Note If your To Memory block is set to write to memory at every sample time interval (that is, it has an incoming port) and it receives a vector signal input of N elements, a corresponding memory region starting with the specified **Memory address** is updated at every sample time. If you specify an **Initial** and/or **Termination value**, that value is written to all locations in the same memory region at initialization and/or termination.

If your To Memory block does not write to memory at every sample time (that is, it does not have an incoming port) and you specify an **Initial** and/or **Termination value**, that value is written to a single memory location that corresponds to the specified **Memory address**.

Custom Code pane



Insert custom code before memory write

C-code to execute before writing to the specified memory address. An example of code that may be inserted here is

```
asm ( " EALLOW " )
```

which enables write access to the device emulation registers on the C2812 DSP.

Insert custom code after memory write

C-code to execute after writing to the specified memory address. An example of code that may be inserted here is

```
asm ( " DIS " )
```

which disables write access to the device emulation registers on the C2812 DSP.

See Also

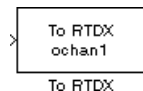
From Memory

Purpose

Add RTDX output channel

Library

rtDXBlocks in Embedded Target for TI C2000 DSP

Description

When you generate code from Simulink in Real-Time Workshop with a To RTDX block in your model, code generation inserts the C commands to create an RTDX output channel on the target. Output channels transfer data from the target to the host.

The generated code contains this command:

```
RTDX_enableOutput(&channelName)
```

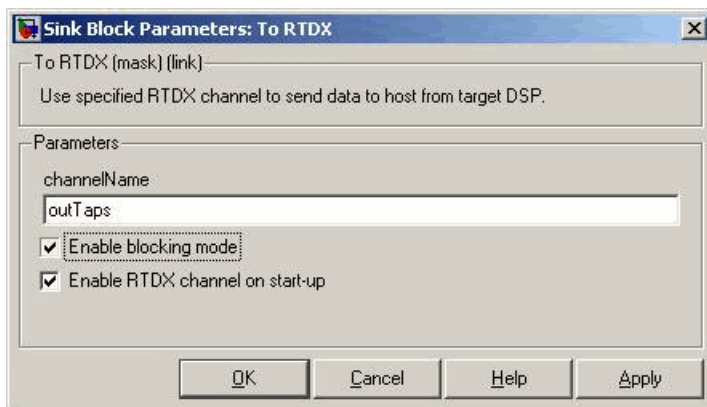
where channelName is the name you enter in the **channelName** field in the To RTDX dialog box.

Note To RTDX blocks work only in code generation and when your model runs on your target. In simulations, this block does not perform any operations.

To use RTDX blocks in your model, you must do the following:

- 1** Add one or more To RTDX or From RTDX blocks to your model.
- 2** Download and run your model on your target.
- 3** Enable the RTDX channels from MATLAB or use **Enable RTDX channel on start-up** on the block dialog.
- 4** Use the readmsg and writemsg functions in MATLAB to send and retrieve data from the target over RTDX.

Dialog Box



Channel name

Name of the output channel to be created by the generated code. The channel name must meet C syntax requirements for length and character content.

Enable blocking mode

Enables blocking mode (selected by default). In blocking mode, writing a message is suspended while the RTDX channel is busy, that is, when data is being written in either direction. The code waits at the RTDX_write call site while the channel is busy. Note that any interrupt of the higher priority will temporarily divert the program execution from this site, but it will eventually come back and wait until the channel stops writing.

When blocking mode is not enabled (when the check box is cleared), writing a message is abandoned if the RTDX channel is busy, and the code proceeds with the current iteration.

Enable RTDX channel on start-up

Enables the RTDX channel when you start the channel from MATLAB. With this selected, you do not need to use the enable function in the Link for Code Composer Studio Development Tools to prepare your RTDX channels. This option applies only to the

channel you specify in **Channel name**. You do have to open the channel.

See Also From RTDX

Trig Fcn IQN

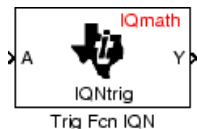
Purpose

Sine, cosine, or arc tangent of IQ number

Library

tiiqmathlib in Embedded Target for TI C2000 DSP

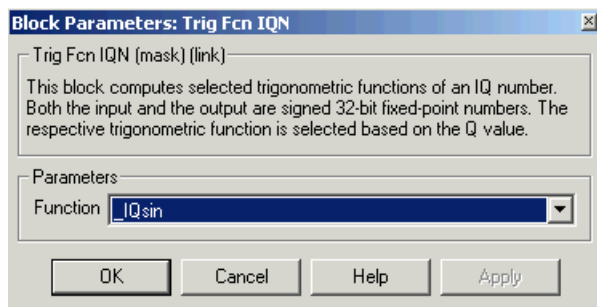
Description



This block calculates basic trigonometric functions and returns the result as an IQ number. Valid Q values for `_IQsinPU` and `_IQcosPU` are 1 to 30. For all others, valid Q values are from 1 to 29.

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

Dialog Box



Function

Type of trigonometric function to calculate:

- `_IQsin` — Compute the sine ($\sin(A)$), where A is in radians.
- `_IQsinPU` — Compute the sine per unit ($\sin(2\pi A)$), where A is in per-unit radians.
- `_IQcos` — Compute the cosine ($\cos(A)$), where A is in radians.
- `_IQcosPU` — Compute the cosine per unit ($\cos(2\pi A)$), where A is in per-unit radians.

- `_IQatan` — Compute the arc tangent ($\tan(A)$), where A is in radians.

See Also

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

A

- Absolute IQN block 7-2
- acquisition window
 - ADC blocks
 - ACQ_PS 3-2 3-5
- ADC blocks
 - C24x 7-5
 - C281x 7-114
- analog-to-digital converter, *see* ADC blocks
- applications
 - TI C2000 1-2
- Arctangent IQN block 7-3
- asymmetric vs. symmetric waveforms 7-146
- asynchronous interrupt processing 1-13

B

- blocking mode
 - C24x 7-16
- blocks
 - adding to model 1-32
 - recommendations 1-19
- build options 7-180 7-204 7-221 7-244

C

- C2000 Library
 - SCI Receive
 - Host side 7-311
 - SCI Setup
 - Host side 7-316
 - SCI Transmit
 - Host side 7-319
- c2000lib startup 1-24
- C24x ADC block 7-5
- C24x CAN Receive block 7-10
- C24x CAN Transmit block 7-14 7-18
- C24x CAP block 6-6 7-18
- C24x GPIO Digital Input block 7-23
- C24x GPIO Digital Output block 7-26

- C24x PWM block 7-29
- C24x QEP block 7-39
- C24x SCI Receive block 7-42
- C24x SCI Transmit block 7-44
- C24x SPI Receive block 7-46
- C24x SPI Transmit block 7-48
- C280x ADC block 7-49
- C280x eCAN Receive block 7-54
- C280x eCAN Transmit block 7-58
- C280x ePWM block 7-62
- C280x eQEP block 7-79
- C280x Hardware Interrupt block 7-109
- C280x SCI Receive block 7-93
- C280x SCI Transmit block 7-100
- C280x SPI Receive block 7-103
- C280x SPI Transmit block 7-106
- C281x ADC block 7-114
- C281x CAP block 7-119
- C281x eCAN Receive block 7-125
- C281x eCAN Transmit block 7-129
- C281x GPIO Digital Input block 7-133
- C281x GPIO Digital Output block 7-136
- C281x PWM block 7-144
- C281x QEP block 7-154
- C281x SCI Receive block 7-157
- C281x SCI Transmit block 7-164
- C281x SPI Receive block 7-167
- C281x SPI Transmit block 7-170
- C281x Timer block 7-173
- CAN/eCAN
 - C24x Receive block 7-10
 - C24x Transmit block 7-14 7-18
 - C280x Transmit block 7-58
 - C280xReceive block 7-54
 - C281x Transmit block 7-129
 - C281xReceive block 7-125
 - timing 7-217 7-258
 - timing parameters
 - bit rate 2-2 to 2-3
- capture block

- C24x 6-6 7-18
- C281x 7-119
- CCS 1-9
 - link options 7-185 7-209 7-226 7-249
 - See also* Code Composer Studio
- Clarke Transformation block 7-176
- clock speed 1-11
- Code Composer Studio 1-9
 - projects 1-36
- code generation
 - options 7-186 7-210 7-227 7-250
 - overview 1-35
- code optimization 5-9
- compiler options 7-180 7-204 7-221 7-244
- configuration default 1-9
- control area network, *see* CAN/eCAN
- control logic 7-35
- conversion
 - float to IQ number 7-260
 - IQ number to different IQ number 7-280
 - IQ number to float 7-276
- CPU clock speed 1-11
- Custom C280x Board block 7-179
- Custom C281x Board block 7-203

D

- data type support 1-10
- data types
 - conversion 5-9
- deadband
 - C281x PWM 7-151
 - C28x PWM 7-35
- default build configuration 1-9
- digital motor control, *see* DMC library
- Division IQN block 7-219 7-305
- DMC library
 - Clarke Transformation 7-176
 - Inverse Park Transformation 7-274
 - Park Transformation 7-296

- PID controller 7-299
- ramp control 7-303
- ramp generator 7-305
- Space Vector Generator 7-322
- Speed Measurement 7-324

DSP board

- target preferences options 7-187 7-211 7-228 7-252

- duty ratios 7-322

E

- enhanced quadrature encoder pulse module
 - C280x 7-79
- ePWM blocks
 - C280x 7-62
- Event Manager timer 7-31

F

- F2808 eZdsp block 7-220
- F2812 eZdsp block 7-243
- fixed-point numbers 5-4
- flash
 - stand alone applications 4-2 to 4-3 4-5
- flash memory 1-5
- Float to IQN block 7-260
- floating-point numbers
 - convert to IQ number 7-260
- four-quadrant arctangent 7-3
- Fractional part IQN block 7-261
- Fractional part IQN x int32 block 7-262
- From Memory block 7-263
- From RTDX block 7-265

G

- GPIO input
 - C24x 7-23
 - C281x 7-133
- GPIO output

C24x 7-26
C281x 7-136

H

hardware 1-3
high-speed peripheral clock 1-12

I

I/O

C24x input 7-23
C24x output 7-26
C281x input 7-133
C281x output 7-136

Idle Task block 7-269
Integer part IQN block 7-272
Integer part IQN x int32 block 7-273
Inverse Park Transformation block 7-274
IQ Math library 5-2
 Absolute IQN block 7-2
 Arctangent IQN block 7-3
 building models 5-9
 code optimization 5-9
 common characteristics 5-2
 Division IQN block 7-219
 Float to IQN block 7-260
 Fractional part IQN block 7-261
 Fractional part IQN x int32 block 7-262
 Integer part IQN block 7-272
 Integer part IQN x int32 block 7-273
 IQN to Float block 7-276
 IQN x int32 block 7-277
 IQN x IQN block 7-278
 IQN1 to IQN2 block 7-280
 IQN1 x IQN2 block 7-281
 Magnitude IQN block 7-295
 Q format notation 5-5
 Saturate IQN block 7-309
 Square Root IQN block 7-326

Trig Fcn IQN block 7-334

IQ numbers

convert from float 7-260
convert to different IQ 7-280
convert to float 7-276
fractional part 7-261
integer part 7-272
magnitude 7-295
multiply 7-278
multiply by int32 7-277
multiply by int32 fractional result 7-262
multiply by int32 integer part 7-273
square root 7-326
trigonometric functions 7-334

IQN to Float block 7-276

IQN x int32 block 7-277

IQN x IQN block 7-278

IQN1 to IQN2 block 7-280

IQN1 x IQN2 block 7-281

L

LF2407 eZdsp block 7-282
linker options 7-181 7-205 7-222 7-245

M

Magnitude IQN block 7-295
mailbox 7-11
math blocks, *see* IQ Math library
MathWorks software 1-5
memory management 1-22
messages
 F2808 eZdsp 7-55
 F2812 eZdsp 7-126
 LF2401 eZdsp 7-11
model
 add blocks 1-32
 building overview 1-22
 creation overview 1-18

- IQmath library 5-9
- multiplication
 - IQN x int32 7-277
 - IQN x int32 fractional part 7-262
 - IQN x int32 integer part 7-273
 - IQN x IQN 7-278
 - IQN1 x IQN2 7-281

O

- operating system requirements 1-3
- optimization code 5-9

P

- Park Transformation block 7-296
- phase conversion 7-176
- PID controller 7-299
- prescaler 7-36
- projects
 - CCS 1-36
- pulse width modulators, *see* PWM blocks
- PWM blocks
 - C24x 7-29
 - C281x 7-144
 - control logic 7-35
 - deadband 7-35

Q

- Q format 5-5
- quadrature encoder pulse circuit
 - C24x 7-39
 - C28x 7-154

R

- ramp control block 7-303
- ramp generator block 7-305
- Real Time Workshop build options
 - Custom C280x Board 7-183

- F2808 eZdsp 7-224
- F2812 eZdsp 7-285
- LF2407 eZdsp 7-206 7-247
- receive 7-10
- reference frame conversion
 - inverse Park transformation 7-274
 - Park transformation 7-296
- reset 1-23
- RTDX
 - from 7-265
 - to 7-331
- runtime options 7-183 7-206 7-224 7-247

S

- sample time
 - F2812 eZdsp 7-56 7-127
 - LF2407 eZdsp 7-12
 - maximum 1-12
- Saturate IQN block 7-309
- scheduling 1-11
- SCI Receive
 - Host side 7-311
- SCI Setup
 - Host side 7-316
- SCI Transmit
 - Host side 7-319
- SCI Transmit and Receive blocks
 - Host side
 - Setup 7-316
- serial communications interface
 - C24x receive 7-42
 - C24x transmit 7-44
 - C280x receive 7-93
 - C280x transmit 7-100
 - C281x receive 7-157
 - C281x transmit 7-164
- serial peripheral interface
 - C24x receive 7-46
 - C24x transmit 7-48

- C280x receive 7-103
- C280x transmit 7-106
- C281x receive 7-167
- C281x transmit 7-170
- setting up hardware 1-3
- signed fixed-point numbers 5-4
- simulation parameters
 - automatic 1-27
 - setting 1-21
- software requirements 1-5
- Space Vector Generator block 7-322
- Speed Measurement block 7-324
- Square Root IQN block 7-326
- startup c2000lib 1-24

T

- target configuration
 - example 6-2 7-243
 - F2808 eZdsp 6-2 7-220
 - LF2407 eZdsp 7-282
- target model creation 1-18
- target preferences

- compiler options 7-180 7-204 7-221 7-244
- DSP board options 7-187 7-211 7-228
 - 7-252
- linker options 7-181 7-205 7-222 7-245
- Target Preferences blocks
 - Custom C280x Board 7-179
 - Custom C281x Board 7-203
 - F2808 eZdsp 7-220
 - F2812 eZdsp 7-243
 - LF2407 eZdsp 7-282
- TI software 1-5
- timing
 - CAN/eCAN 7-217 7-258
 - interrupts 1-11
- To Memory block 7-328
- To RTDX block 7-331
- transmit 7-14
- Trig Fcn IQN block 7-334

W

- waveforms 7-146