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

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

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

Revision History

November 2003	Online only	New for Version 1.0 (Release 13SP1+)
June 2003	Online only	New for Version 1.1 (Release 14)
October 2004	Online only	Revised for Version 1.1.1 (Release 14SP1)
December 2004	Online only	Revised for Version 1.2 (Release 14SP1+)
March 2005	Online only	Revised for Version 1.2.1 (Release 14SP2)
September 2005	Online only	Revised for Version 1.3 (Release 14SP3)
March 2006	Online only	Revised for Version 2.0 (Release 2006a)

Contents

1 Getting Started

2 Using the IQmath Library

3 Blocks — By Category

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?

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 "Running Code from Flash Memory" on page 1-5.

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
  - TMS320F2806
  - TMS320F2808
  - 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 the F2812, you need to change the jumper settings from their factory defaults. For more information on this, see the discussion of the jumper settings for Boot Mode in the *eZdsp™ F2812 Technical Reference*, available from the Spectrum Digital Web site at `http://c2000.spectrumdigital.com/ezf2812/`.

For the F2808, you need to change DIP switch settings. For more information on this, see the *eZdsp™ F2808 USB Technical Reference*, available from the Spectrum Digital Web site at `http://c2000.spectrumdigital.com/ezf2808/`.

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 Board             Proc Processor  Processor
Num   Name              Num  Name       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:

$$Base\ Rate\ Sample\ Time = \frac{Timer\ Period}{\left(\frac{(Timer\ Clock\ Speed)}{TimerClockPrescaler}\right)}$$

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):

HISCLK = SYSCLKOUT / (high-speed peripheral prescaler)

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

For example, on the F2812, the HISCLK rate becomes

HISCLK = 150 MHz / 2 = 75 MHz

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 connecst 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.

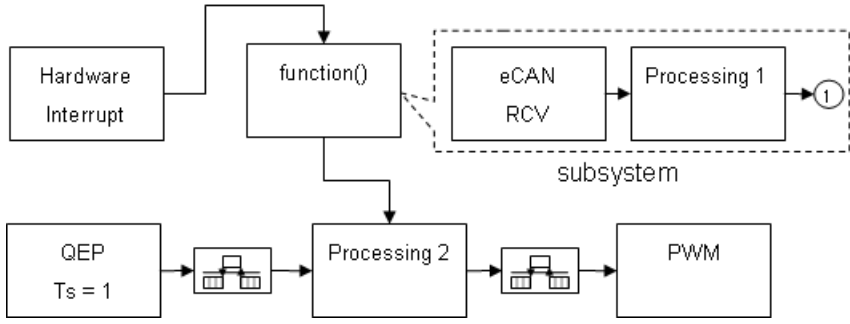
  Embedded Target for TI C2000 provides a Hardware Interrupt block for each of the supported processor families: C280x Hardware Interrupt and C281x Hardware Interrupt.

- 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 triggers a task that also runs periodically in the context of the timer interrupt. The Hardware Interrupt block installs an ISR for a given hardware interrupt and relates the ISR to the appropriate function calls that trigger further processing. The subsystem contains the blocks that are executed asynchronously, only within the context of the hardware interrupt.

The periodic tasks, represented in the bottom half of the diagram, are clocked by a dedicated timer. The blocks are executed from within the context of a corresponding ISR, which is generated by default. The timer period is tied to the model base rate. Rate Transition blocks are used to connect the block labelled **Processing 2**, which is triggered both synchronously and asynchronously, to the model.



If there were no interaction between the synchronous and asynchronous parts of the model, the Rate Transition blocks would not be necessary.

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 1-20 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 Table of Contents in Help, select **Embedded Target for the TI C2000 DSP**.

Blocks to Avoid Using in Your Models

Many blocks in the blocksets communicate with your MATLAB workspace. These blocks also 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. The next table presents the blocks you should *not* use in your target models.

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

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 runtime `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:

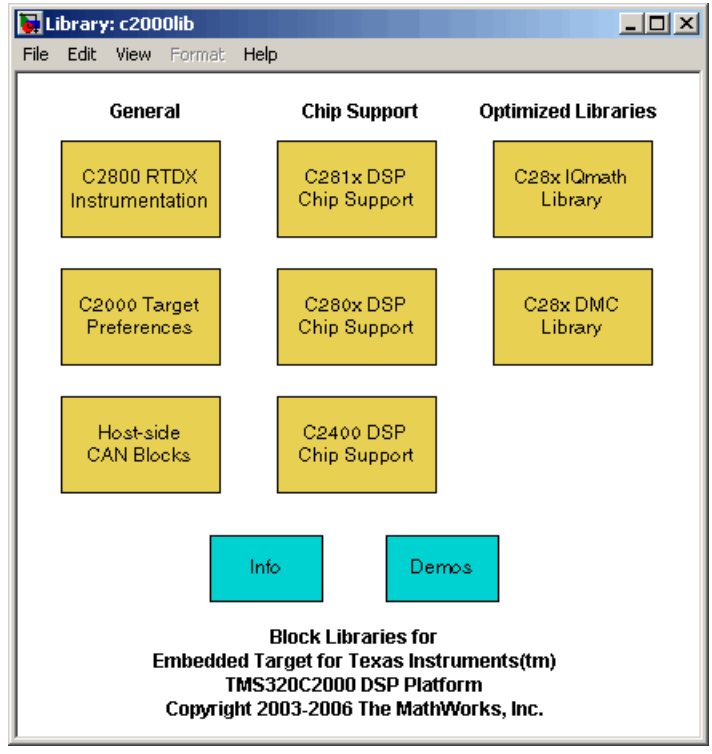
1. Connect the function generator output to the ADC input ADCINA0 on the eZdsp F2812.
2. Connect the output of PWM1 on the eZdsp F2812 to the analog input of the oscilloscope.
3. Connect VREFLO to AGND on the eZdsp F2812. See the section on the Analog Interface in Chapter 2 of the *eZdsp™ F2812 Technical 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

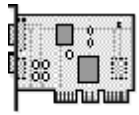
- Info block — Online help
- Demos block — Demos window

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

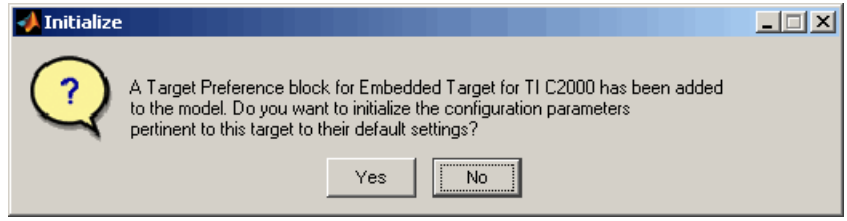
Setting Up the Model

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.



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:

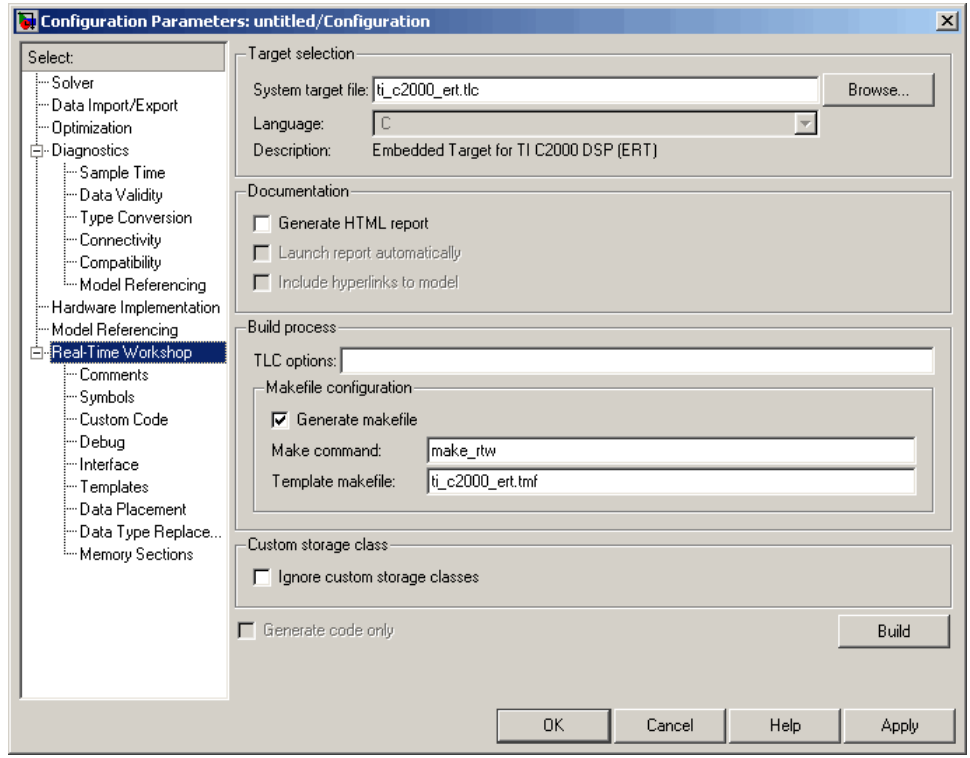
Pane	Field	Setting
Solver	**Stop time**	`inf`
Solver	**Type**	`Fixed-step`
Data Import/Export	**Save to workspace - Time**	Off (cleared)
Data Import/Export	**Save to workspace - Output**	Off (cleared)
Hardware Implementation	**Device type**	`TI C2000`
Real-Time Workshop	**Target configuration - System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

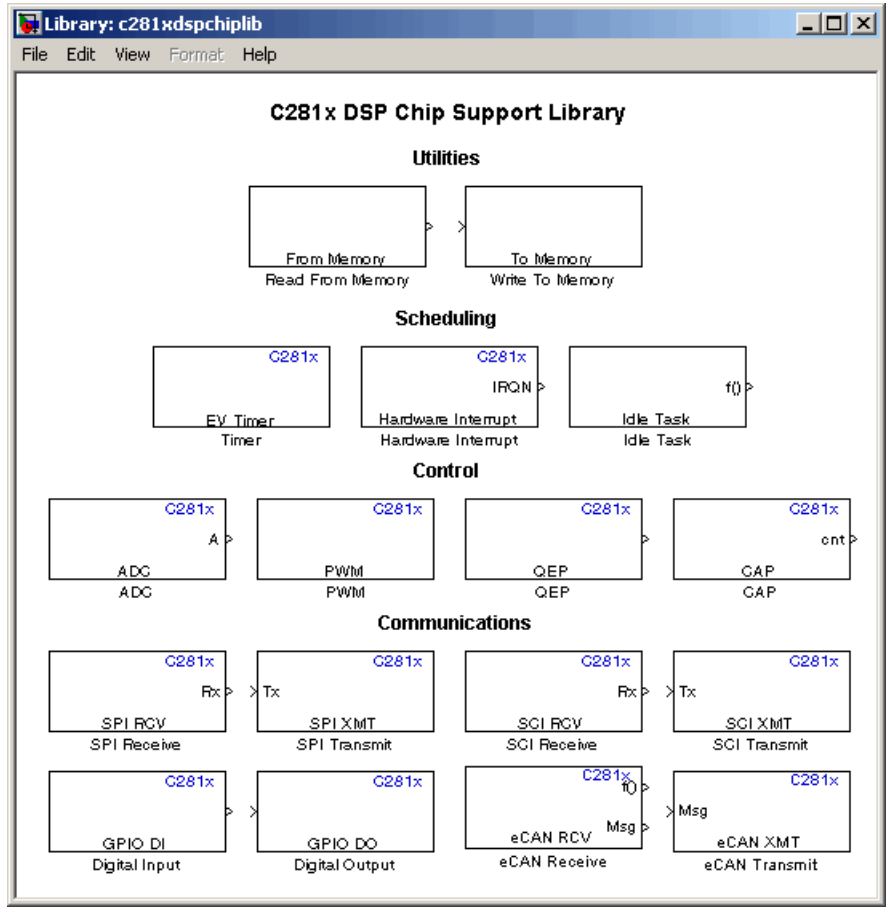
Subfield	Field	Setting
DSP Board Label	**DSPBoardLabel**	`F2812 PP Emulator` (see Note below)
DSP Chip	**DSPChipLabel**	`TI TMS320C2812`
eCAN	**BaudRatePrescaler**	`10`
	EnhancedCANMode	`True`
	SAM	`Sample_one_time`
	SBJ	`Only_falling_edges`
	SJW	`2`
	SelfTestMode	`False`
	TSEG1	`8`
	TSEG2	`6`

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.

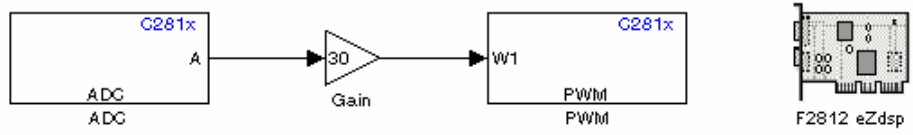
3 Drag the C281x PWM block into your model. Double-click the PWM block in the model and set the following parameters. Refer to the C281x PWM reference page for information on these fields.

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 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.

2

Using 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 2-4. 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 2-5

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 2-9.

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.

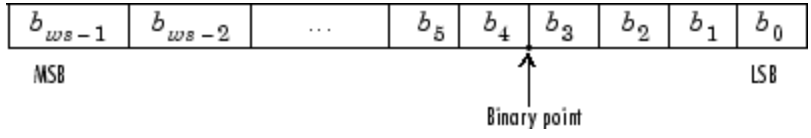
To learn more about characteristics particular to each block in the library, see “C28x IQmath Library (tiiqmathlib)” on page 3-10 for links to the individual block reference pages.

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 Q.15 numbers yields a product that is a signed 32-bit data type with n = 30 bits to the right of the binary point. One bit is the designated sign bit, thereby forcing m to be 1:

m+n+1 = 1+30+1 = 32 bits total

Therefore, this number is expressed as

```
Q1.30
```

In Simulink Fixed Point, this data type is expressed as

```
sfix32_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 bits total

Therefore, this number is expressed as

```
Q-2.17
```

In Simulink Fixed Point, this data type is expressed as

```
sfix16_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

```
sfix16_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 Library (c2000tgtpreflib) (p. 3-2)	Target preference blocks for C2000 boards
Host-side CAN Blocks (c2000canlib) (p. 3-3)	Host-side CAN blocks
C2000 RTDX Instrumentation Library (rtdxBlocks) (p. 3-4)	RTDX blocks for C2000 boards
C2400 DSP Chip Support Library (c2400dspchiplib) (p. 3-5)	Blocks that support C24x boards
C280x DSP Chip Support Library (c280xdspchiplib) (p. 3-6)	Blocks that support C280x boards
C281x DSP Chip Support Library (c281xdspchiplib) (p. 3-7)	Blocks that support C281x boards
C28x Digital Motor Control Library (c28xdmclib) (p. 3-9)	Blocks that represent the functionality of the TI C28x DMC Library
C28x IQmath Library (tiiqmathlib) (p. 3-10)	Blocks that represent the functionality of the TI IQmath Library

C2000 Target Preferences Library (c2000tgtpreflib)

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)

Refer to the CAN Blockset documentation for information on these blocks.

Vector CAN Configuration	Configure a CAN channel (either hardware or virtual) for use with Vector-Informatik drivers
Vector CAN Receive	Read CAN frames from a Vector CAN channel
Vector CAN Transmit	Transmit CAN frames on a Vector CAN channel
CAN Message Packing	Map Simulink signals to CAN messages.
CAN Message Packing (CANdb)	Pack Simulink signals into CAN messages defined by CANdb
CAN Message Filter	Dispatch message processing based on message ID
CAN Message Unpacking	Inspect and unpack the individual fields in a CAN message
CAN Message Unpacking (CANdb)	Decompose a CAN frame into its constituent signals

C2000 RTDX Instrumentation Library (rtdxBlocks)

From RTDX	Add RTDX input channel
To RTDX	Add RTDX output channel

C2400 DSP Chip Support Library (c2400dspchiplib)

C24x ADC	Analog-to-digital converter (ADC)
C24x CAN Receive	Enhanced Control Area Network receive mailbox
C24x CAN Transmit	Enhanced Control Area Network 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 wave modulators (PWMs)
C24x QEP	Quadrature encoder pulse circuit
C24x SCI Receive	Receive data on the target via serial communications interface (SCI) from the host
C24x SCI Transmit	Transmit data on target via serial communications interface (SCI) from host
C24x SPI Receive	Receive data via the serial peripheral interface (SPI) on target
C24x SPI Transmit	Transmit data via the serial peripheral interface (SPI) to host
From Memory	Retrieve data from target memory
To Memory	Write data to target memory

C280x DSP Chip Support Library (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	Configures the C280x Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms.
C280x eQEP	Quadrature encoder pulse circuit
C280x Hardware Interrupt	Create an Interrupt Service Routine to handle hardware interrupts
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 Library (c281xdspchiplib)

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 an 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 on target via serial communications interface (SCI) from host
C281x SPI Receive	Receive data via the serial peripheral interface (SPI) on the target
C281x SPI Transmit	Transmit data via the serial peripheral interface (SPI) to the 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 Library (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 a 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 Library (tiiqmathlib)

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 an 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

4

Blocks — Alphabetical List

Absolute IQN

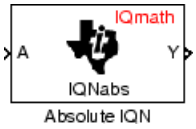
Purpose

Absolute value

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

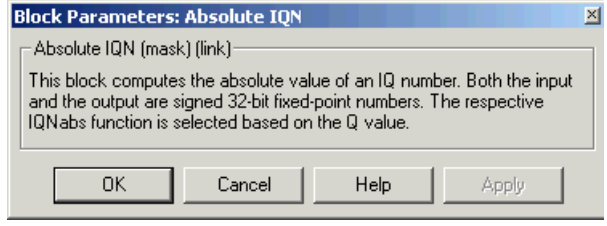


Absolute IQN

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 2-2 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

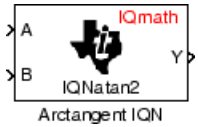
Purpose

Four-quadrant arc tangent

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

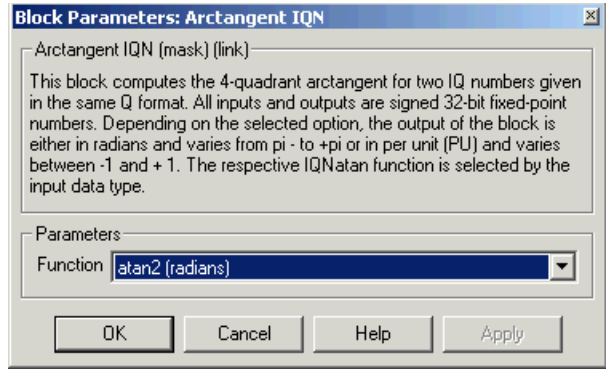


Arctangent IQN

This 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 2-2 for more information.

Dialog Box



Function

Type of arc tangent to calculate, either

- `atan2` — Compute the four-quadrant arc tangent with output in radians with values between -pi and +pi.
- `atan2PU` — Compute the four-quadrant arc tangent per unit. If `atan2(B,A)` is greater than or equal to zero, `atan2PU(B,A) = atan2(B,A)/2*pi`. Otherwise, `atan2PU(B,A)`

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

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

C24x ADC

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.

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 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

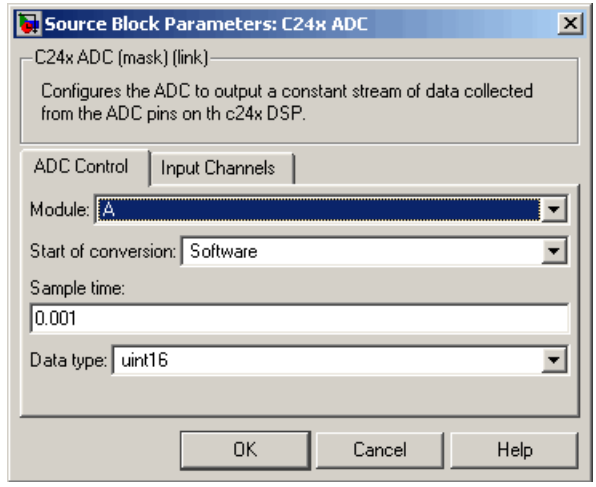
The output of the C24x ADC is a vector of `uint16` values. The output values are in the range 0 to 1023 because the C24x ADC is a 10-bit converter.

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

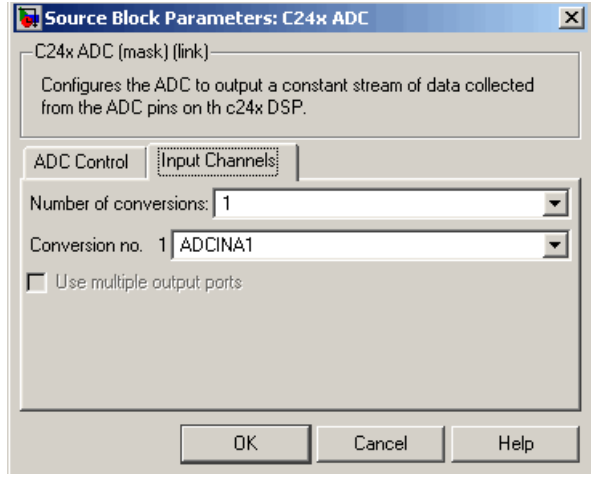
Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. See "Scheduling and Timing" on page 1-11 for more information on timing.

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 receive mailbox

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

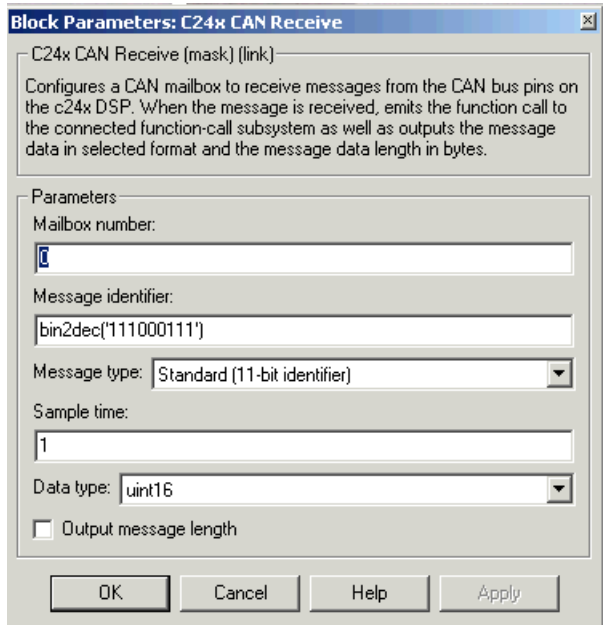
C24x CAN Receive

The C24x Control Area Network (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 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



Mailbox number

Unique number between 0 and 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.

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.

Data type

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

For `uint16` data,

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

For `uint32` data,

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

For example, if the received message has two bytes:

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

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

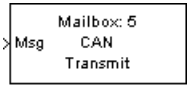
Purpose

Enhanced Control Area Network transmit mailbox

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description



C24x CAN Transmit

The C24x Control Area Network (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 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 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
```

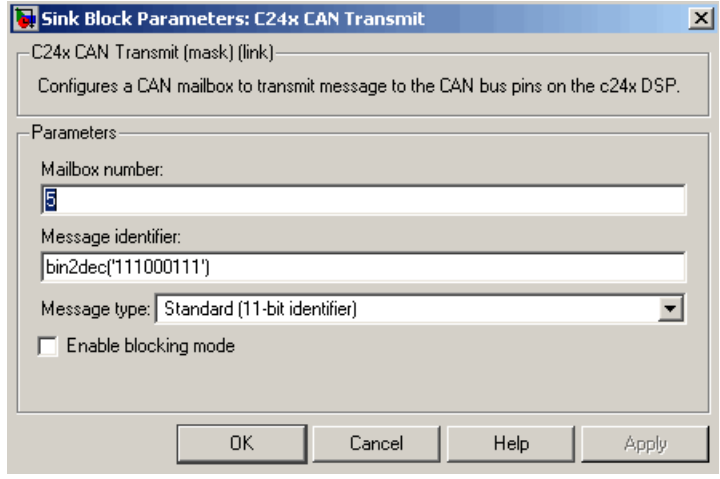
For input of type `uint16[2]`, which is a two-element vector,

```
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
```

Dialog Box



Mailbox number

Unique number between 0 and 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.

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

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

C24x CAP

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

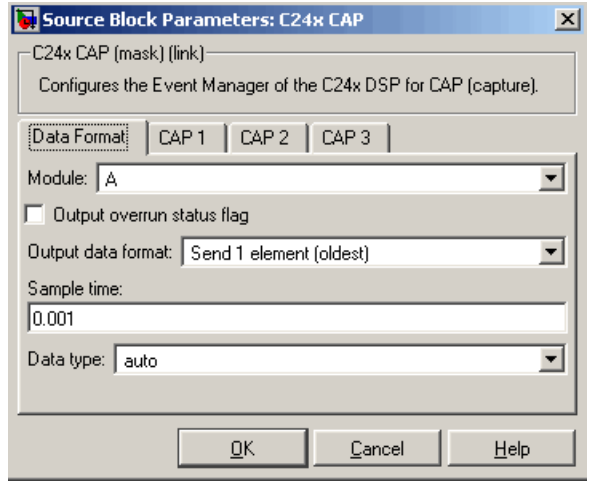
This block has up to two outputs: a `cnt` (count) output and an optional, FIFO status `flag` output. The `cnt` output increments each time a transition of the selected type occurs. The status flag outputs are

- `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.

Send data format

The type of data to output:

- `Send 2 elements (FIFO Buffer)` — Sends the latest two values. The output is updated when there are two elements in the FIFO, which is indicated by bit 13 or 11 or 9 being sent (CAP x FIFO). If the CAP is polled when fewer than two elements are captures, old values are repeated. The CAP registers are read as follows:

  a The CAP x FIFO status bits are read and the value is stored in the status flag.

  b The top value of the FIFO is read and stored in the output at index 0.

  c The new top value of the FIFO (the previously stored bottom stack value) is read and stored in the output at index 1.

- `Send 1 element (oldest)` — Sends the older of the two most recent values. The output is updated when there is at least one element in the FIFO, which is indicated by any of the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:

  a The CAP x FIFO status bits are read and the value is stored in the status flag.

  b The top value of the FIFO is read and stored in the output.

- `Send 1 element (latest)` — Sends the most recent value. The output is updated when there is at least one element in the FIFO, which is indicated by any of the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:

  a The CAP x FIFO status bits are read and the value is stored in the status flag.

  b If there are two entries in the FIFO, the bottom value is read and stored in the output. If there is only one entry in the FIFO, the top value is read and stored in the output.

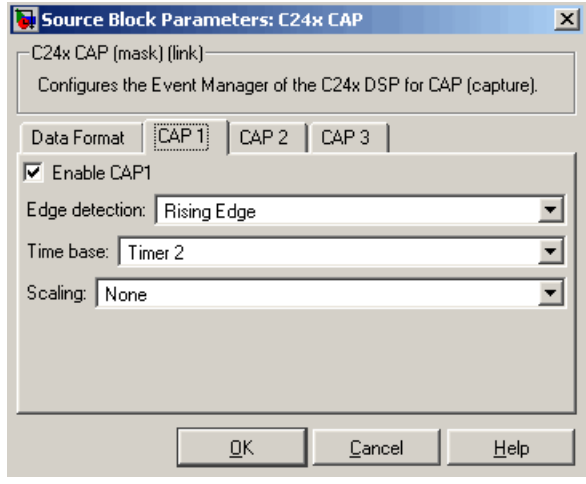
Sample time

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

Data type

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

C24x GPIO Digital Input

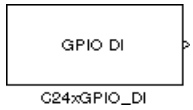
Purpose

General-purpose I/O pins for digital input

Library

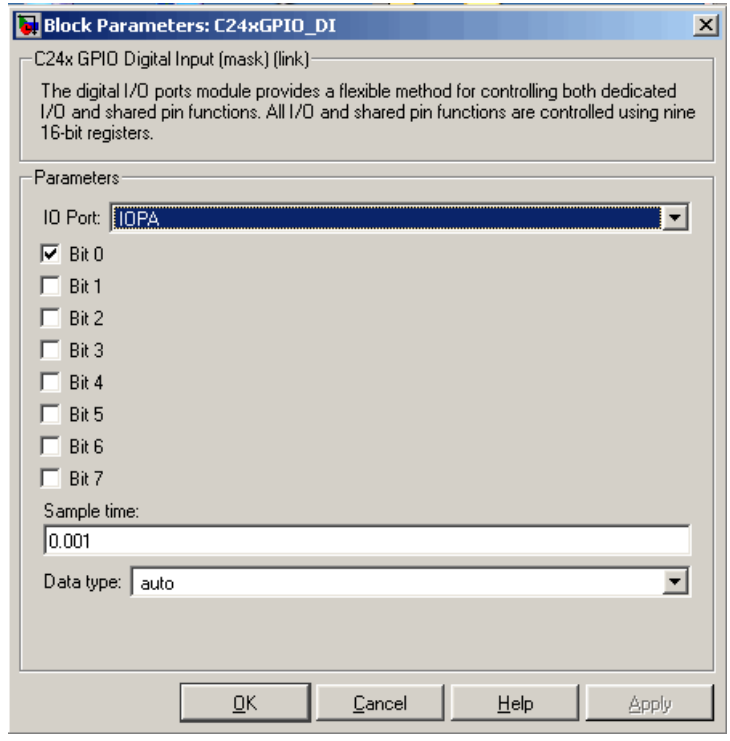
`c2400dspchiplib` in Embedded Target for TI C2000 DSP

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 you select multiple bits, vector input is expected. Unselected bits are available for peripheral functionality. Note that multiple 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

Purpose

General-purpose I/O pins for digital output

Library

`c2400dspchiplib` 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.

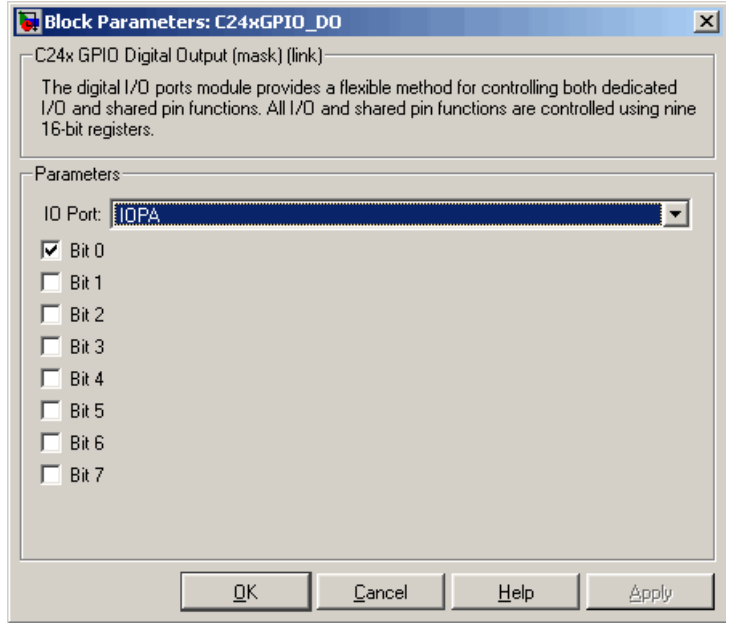
IO MUX Output Control Register A

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

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. Unselected bits are available for peripheral functionality. Note that multiple GPIO DO blocks cannot share the same I/O port. Only one bit is available for `IOPD`.

See Also

C24x GPIO Digital Input

C24x PWM

Purpose

Pulse wave modulators (PWMs)

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

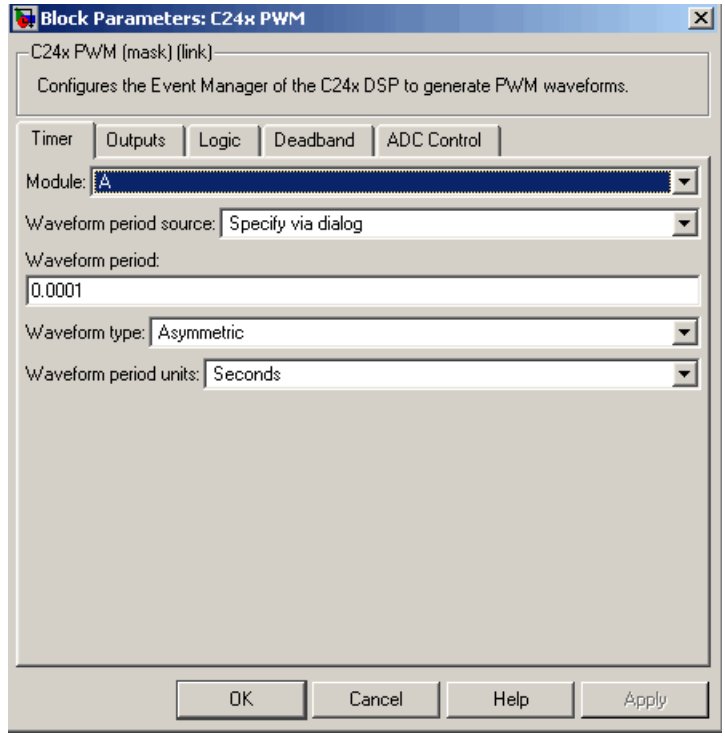
C24x PWM

LF2407 DSPs include a set of pulse width modulators (PWM) 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 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.

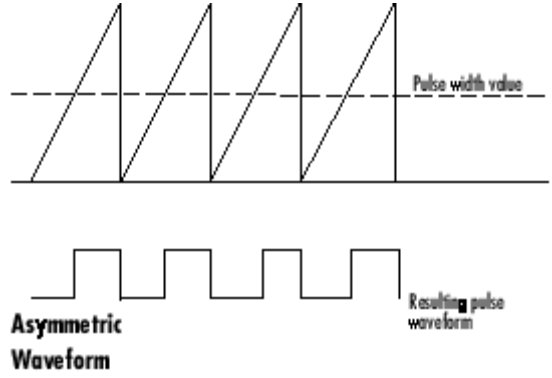
Waveform period

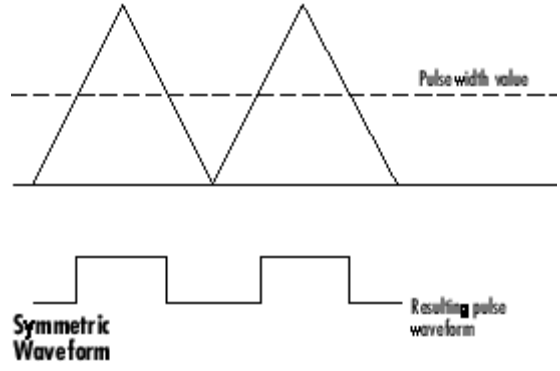
Period of the PWM waveform measured in clock cycles or in seconds, as specified in the **Waveform period units**.

Note "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 illustration below shows the difference between the two types of waveforms.

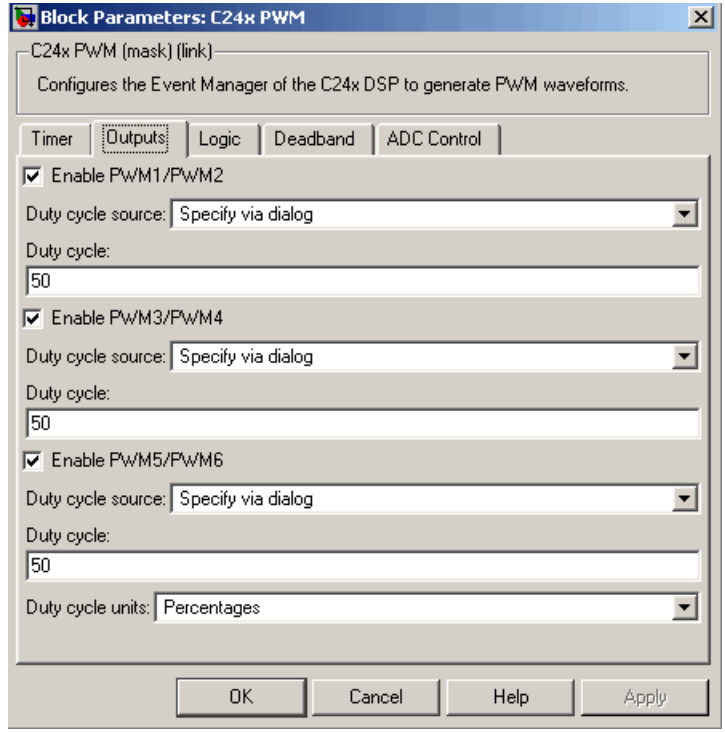




Waveform period units

Units in which to measure the waveform period. Options are `clock cycles`, which refer to the 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



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.

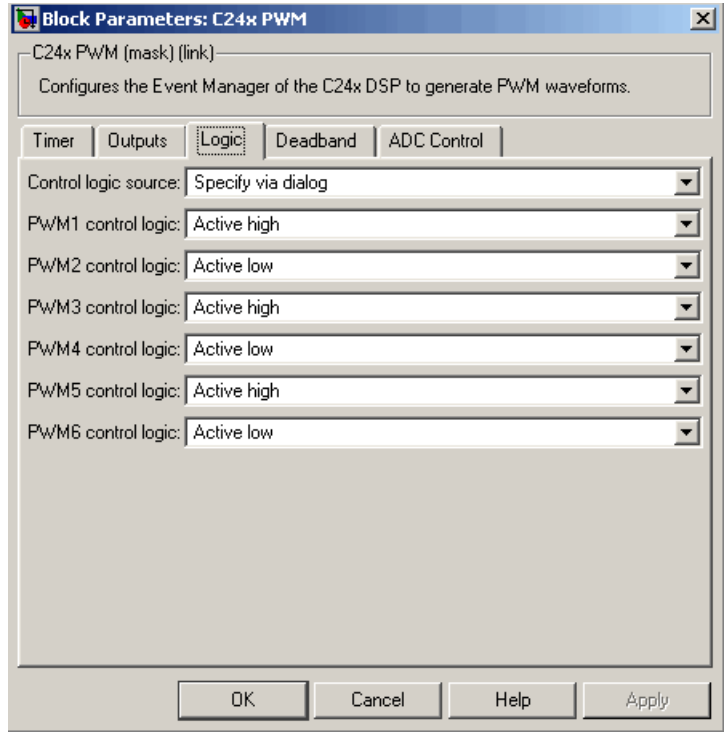
Duty cycle

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



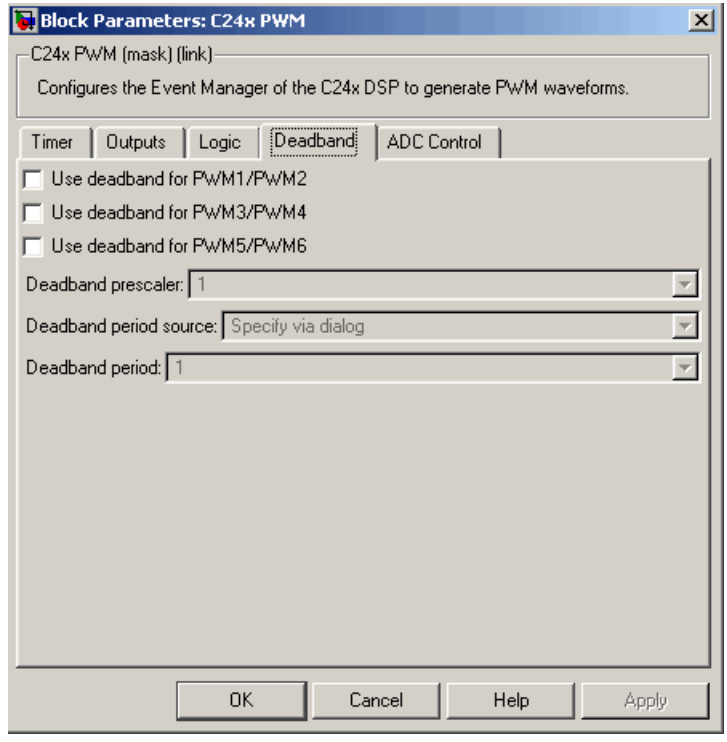
Control logic source

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

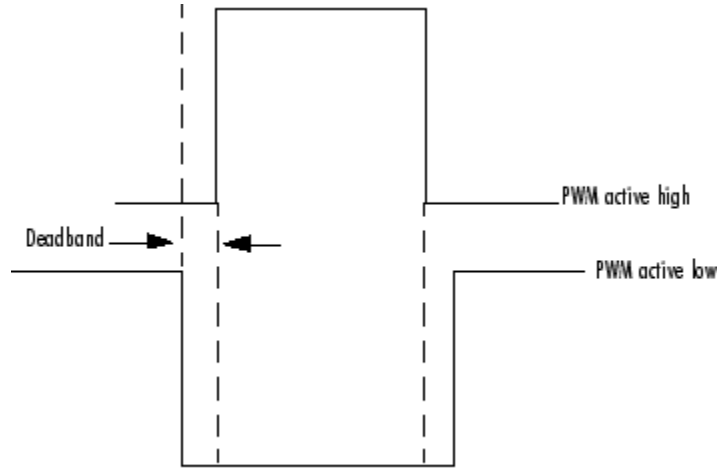
Control logic trigger for the PWM. `Forced high` causes the pulse value to be high. `Active high` causes the pulse value to go from low to high. `Active low` causes the pulse value to go from high to low. `Forced low` causes the signal to be low.

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 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`.

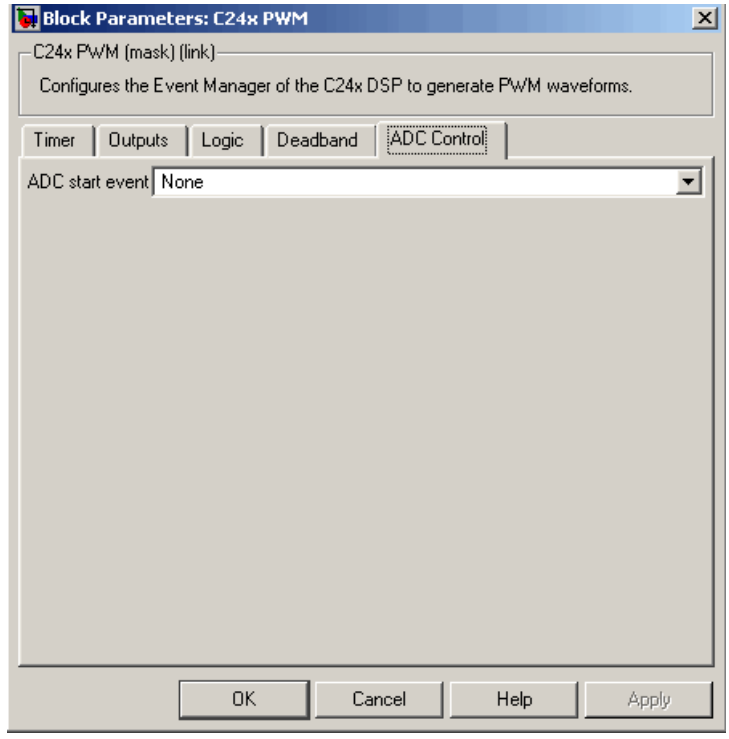
Deadband period source

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).

- `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 MHz = 0.0016. If you enter a **Sample time** in the C24x ADC dialog 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 **Pulse width** above determines the value in the register.

See Also

C24x ADC

C24x QEP

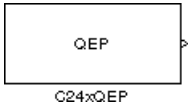
Purpose

Quadrature encoder pulse circuit

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description



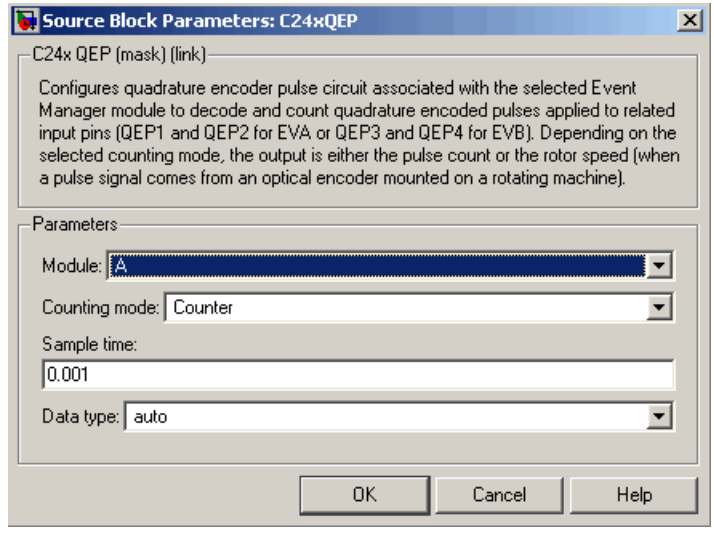
C24xQEP

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`.

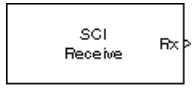
Purpose

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

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description



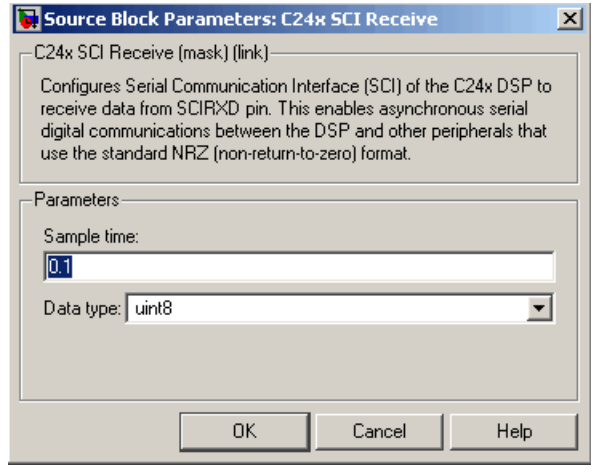
C24x SCI Receive

The C24x SCI Receive block supports asynchronous serial digital communications between the target and other asynchronous peripherals in non-return-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

Purpose

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

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

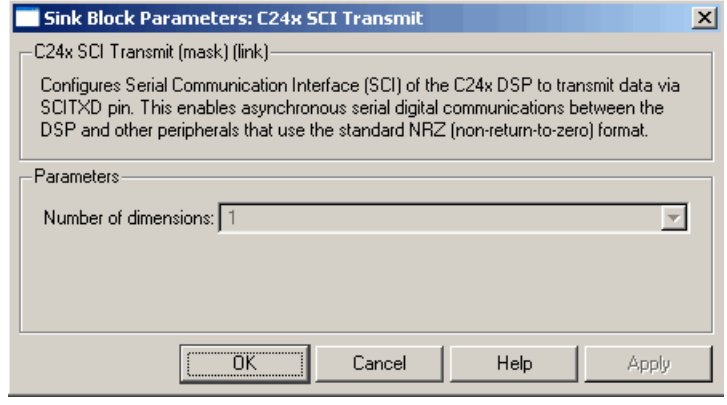
C24x SCI Transmit

The C24x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C24x target's COM ports in non-return-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

Purpose

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

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

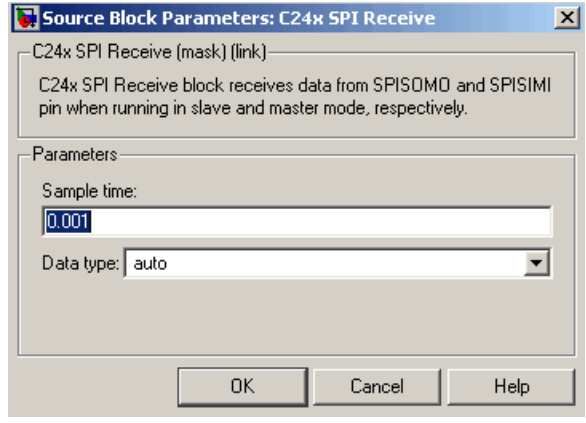
C24x SPI Receive

The C24x 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 SPISIM0 pin transmits data and the SPISOM1 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

Purpose

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

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

C24x SPI Transmit

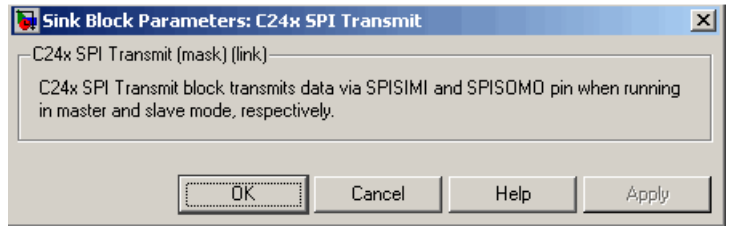
The C24x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIM0 pin transmits data and the SPISOM1 pin receives data. 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

C280x ADC

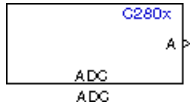
Purpose

Analog-to-digital converter (ADC)

Library

`c280xdspchiplib` 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

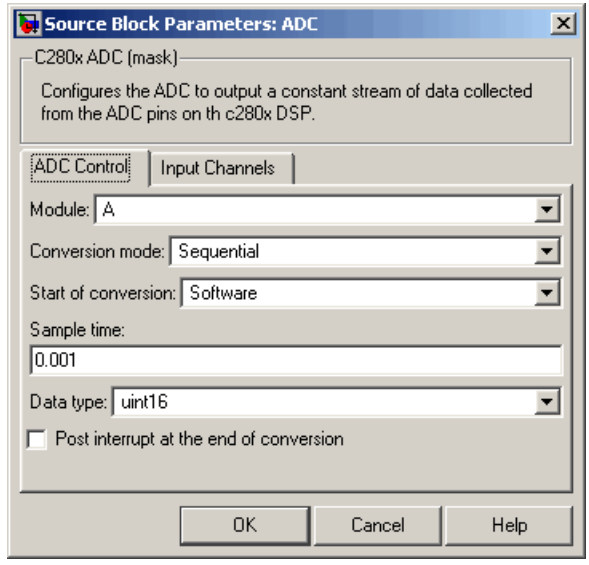
The output of the C280x ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the C280x ADC is 12-bit converter.

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 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.

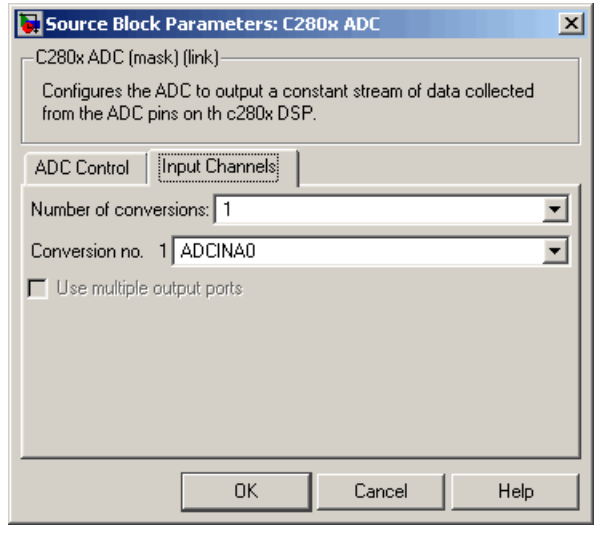
Data type

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.

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

Purpose

Enhanced Control Area Network receive mailbox

Library

`c280xdspchiplib` in Embedded Target for TI C2000 DSP

Description

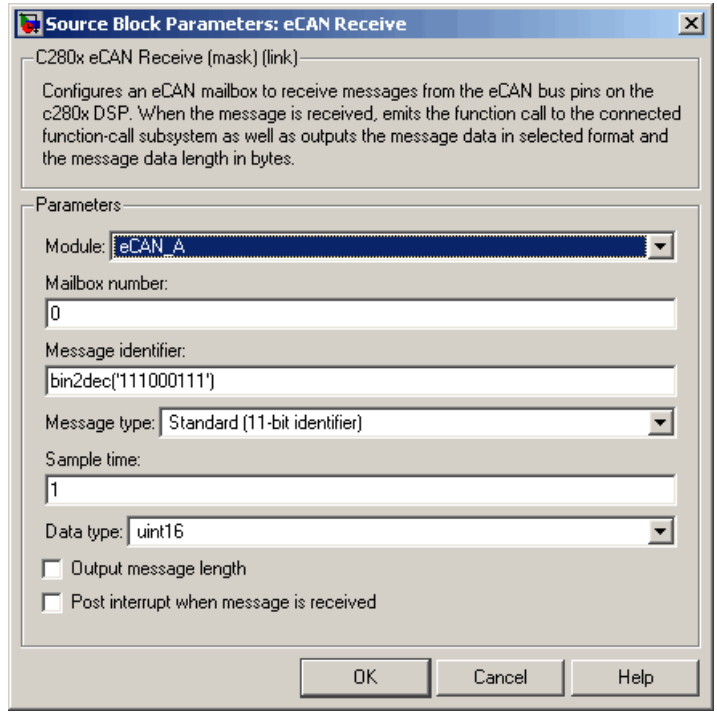
eCAN Receive

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 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

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.

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

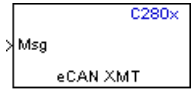
Purpose

Enhanced Control Area Network transmit mailbox

Library

`c280xdspchiplib` in Embedded Target for TI C2000 DSP

Description



eCAN Transmit

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
```

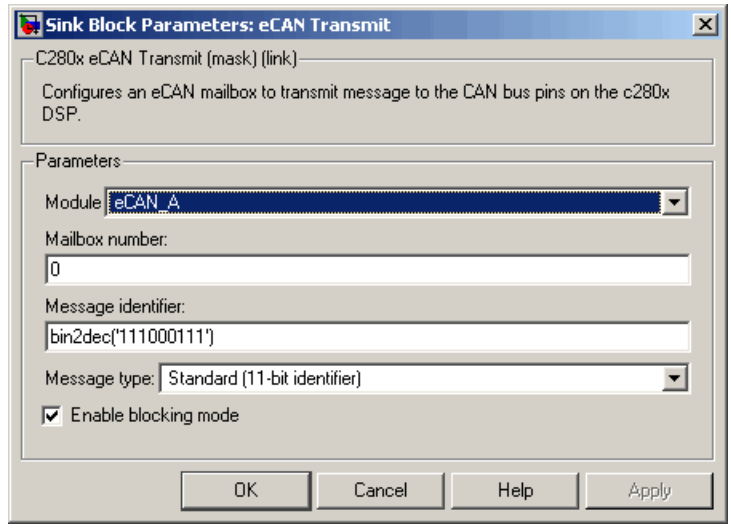
For input of type `uint16[2]`, which is a two-element vector,

```
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
```

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.

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

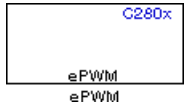
Purpose

Configures the C280x Event Manager to generate Enhanced Pulse Width Modulator (ePWM) waveforms.

Library

`c280xdspchiplib` in Embedded Target for TI C2000 DSP

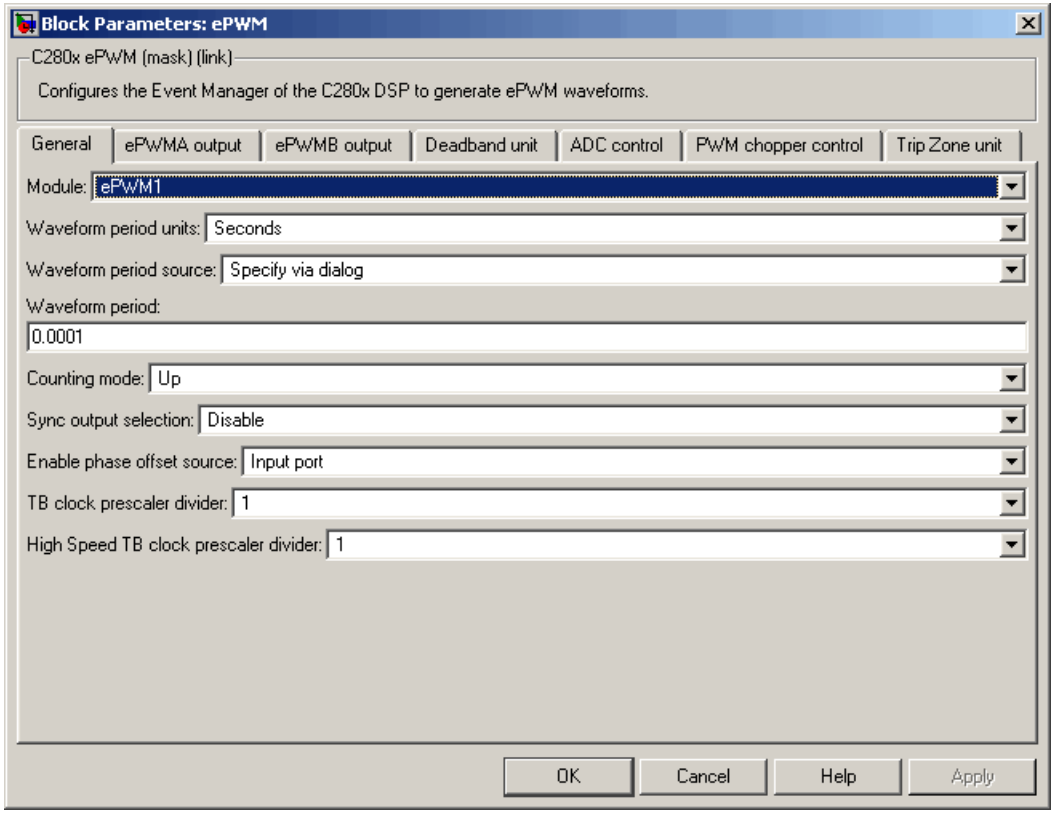
Description



ePWM

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.

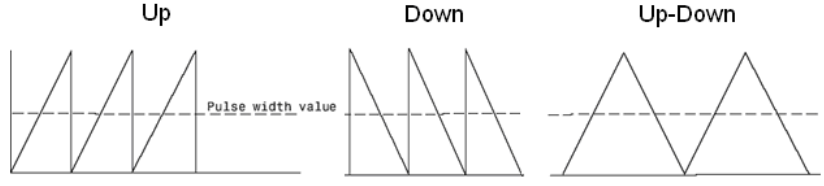
Waveform period

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

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

Counting mode

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



Sync output selection

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

Enable S/W sync input port

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

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

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

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

TBCLK = SYSCLKOUT/(HSPCLKDIV * CLKDIV)

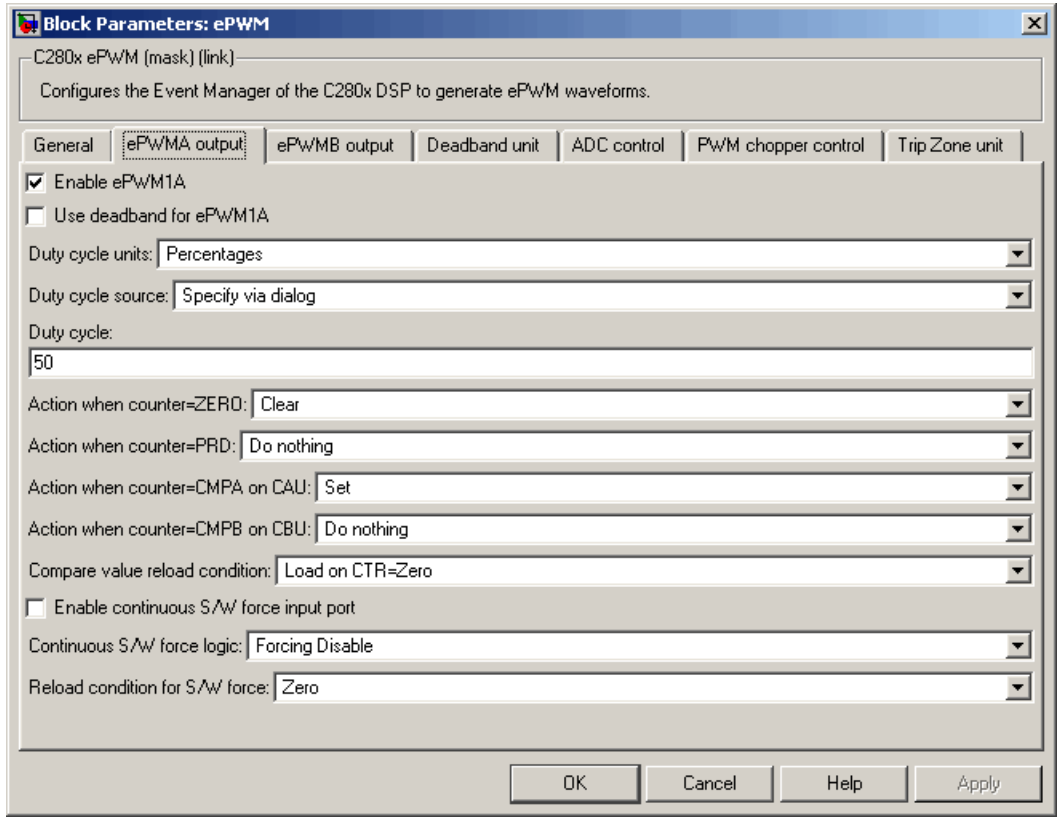
Because the default values for both the **High Speed TB clock prescaler divider** and the **High Speed TB clock prescaler divider** are both `1`, the default value of the Time-base Clock is equal to the system clock speed of 100 MHz

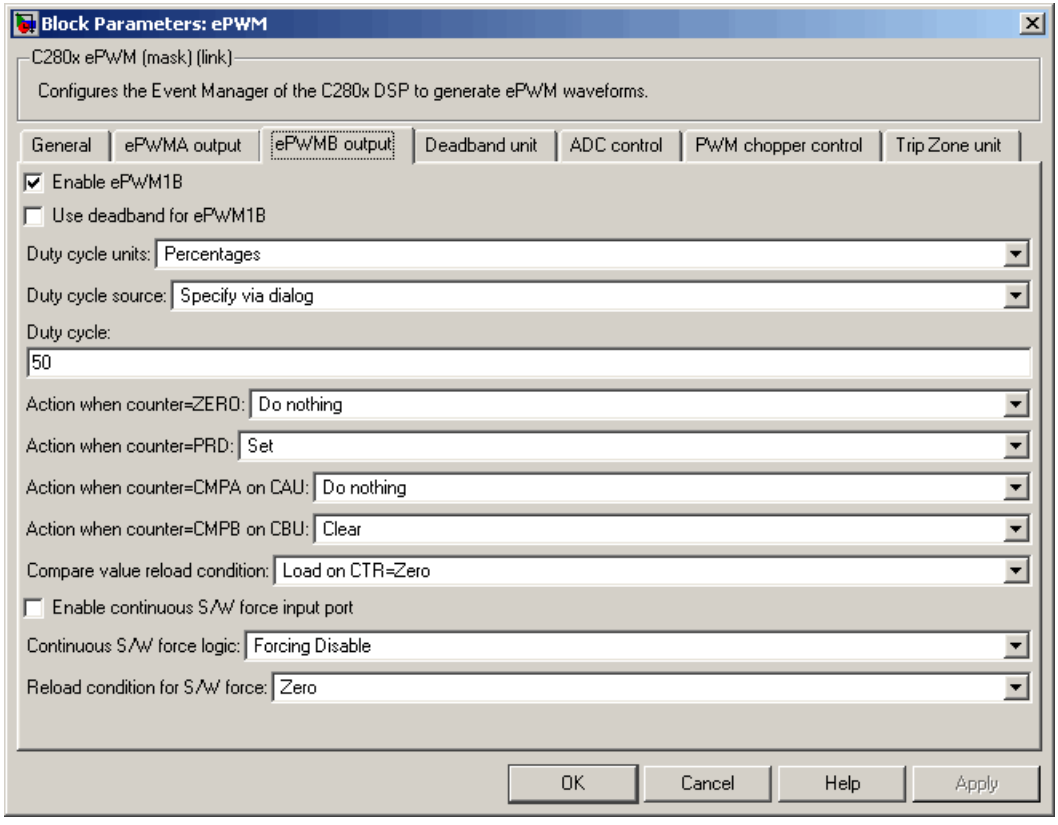
Choices are `1`, `2`, `4`, `8`, `16`, `32`, `64`, and `128`.

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

ePWMA 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.


Block Parameters: ePWM
C280x ePWM (mask) (link)
Configures the Event Manager of the C280x DSP to generate ePWM waveforms.
General
ePWMA output
ePWMB output
Deadband unit
ADC control
PWM chopper control
Trip Zone unit
Enable ePWM1A
Use deadband for ePWM1A
Duty cycle units:
Percentages
Duty cycle source:
Specify via dialog
Duty cycle:
50
Action when counter=ZERO:
Clear
Action when counter=PRD:
Do nothing
Action when counter=CMPA on CAU:
Set
Action when counter=CMPB on CBU:
Do nothing
Compare value reload condition:
Load on CTR=Zero
Enable continuous S/W force input port
Continuous S/W force logic:
Forcing Disable
Reload condition for S/W force:
Zero
OK
Cancel
Help
Apply



Enable ePWMxA
Enable ePWMxB

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

Use deadband for ePWMxA
Use deadband for ePWMxB

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

Duty cycle units

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

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

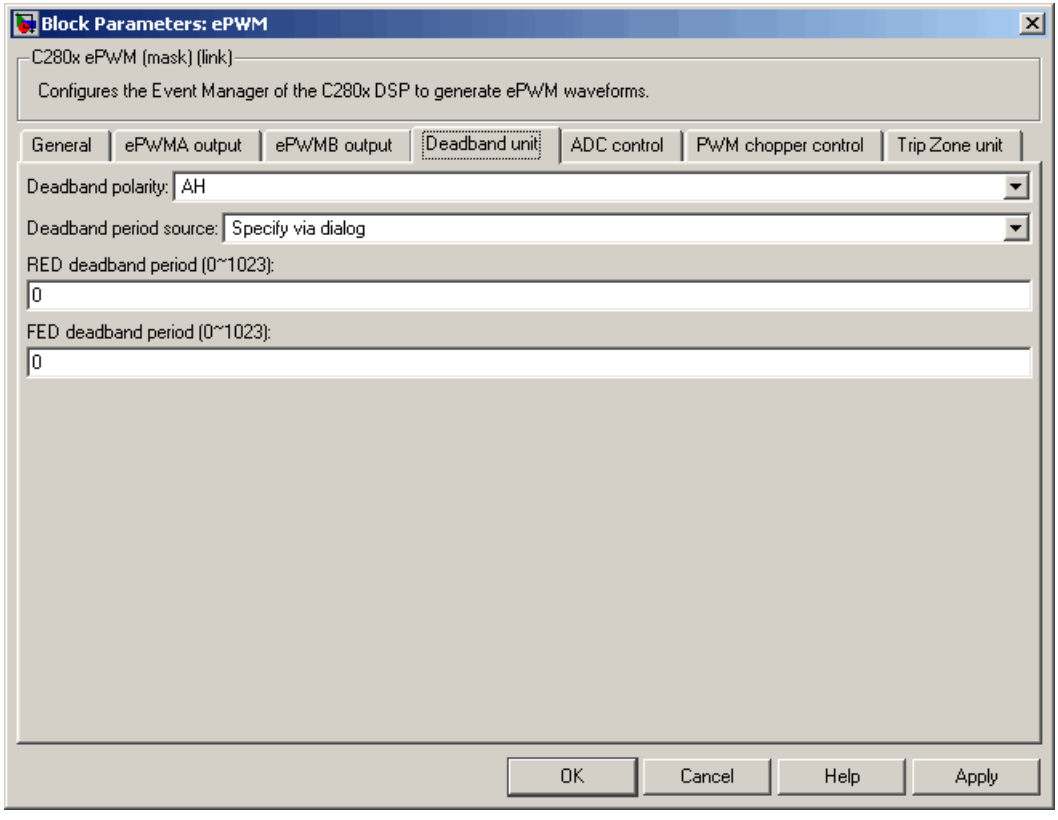
by default. Select this check box to obtain the control logic from the input port

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

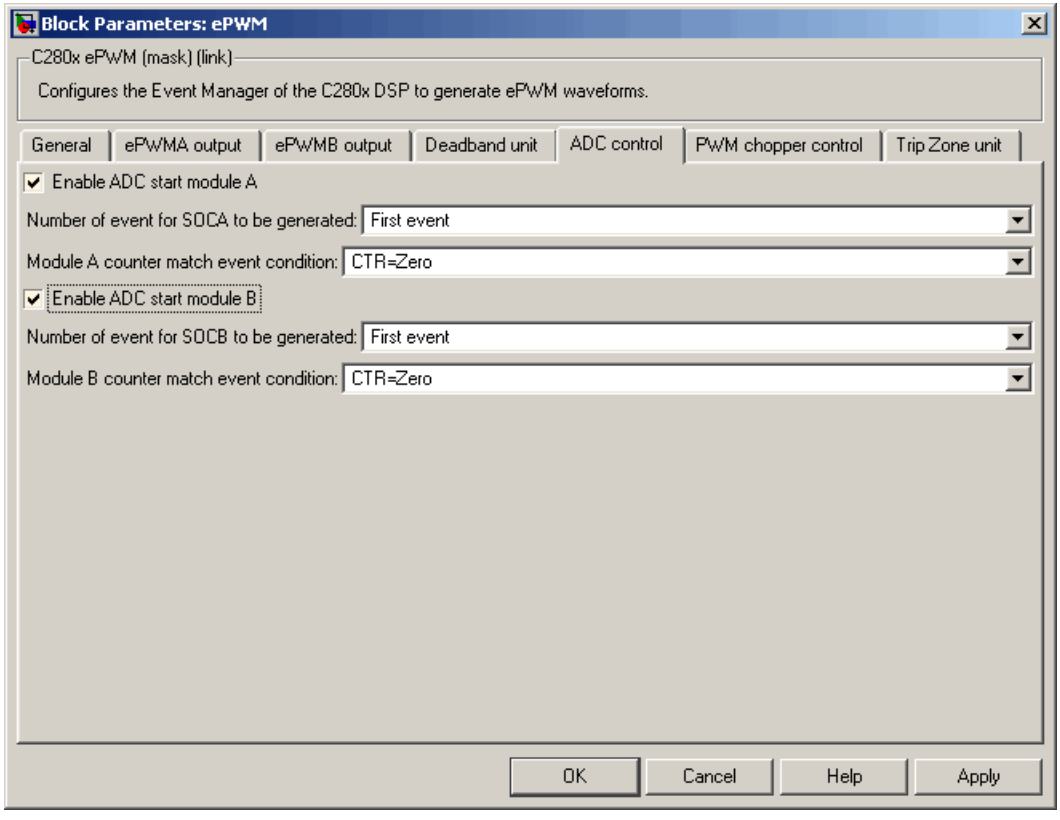
This field appears only when **Use deadband for ePWMxA** is selected in the **ePWMA output** pane. Enter a value from 0 to 1023 to specify a rising edge delay.

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

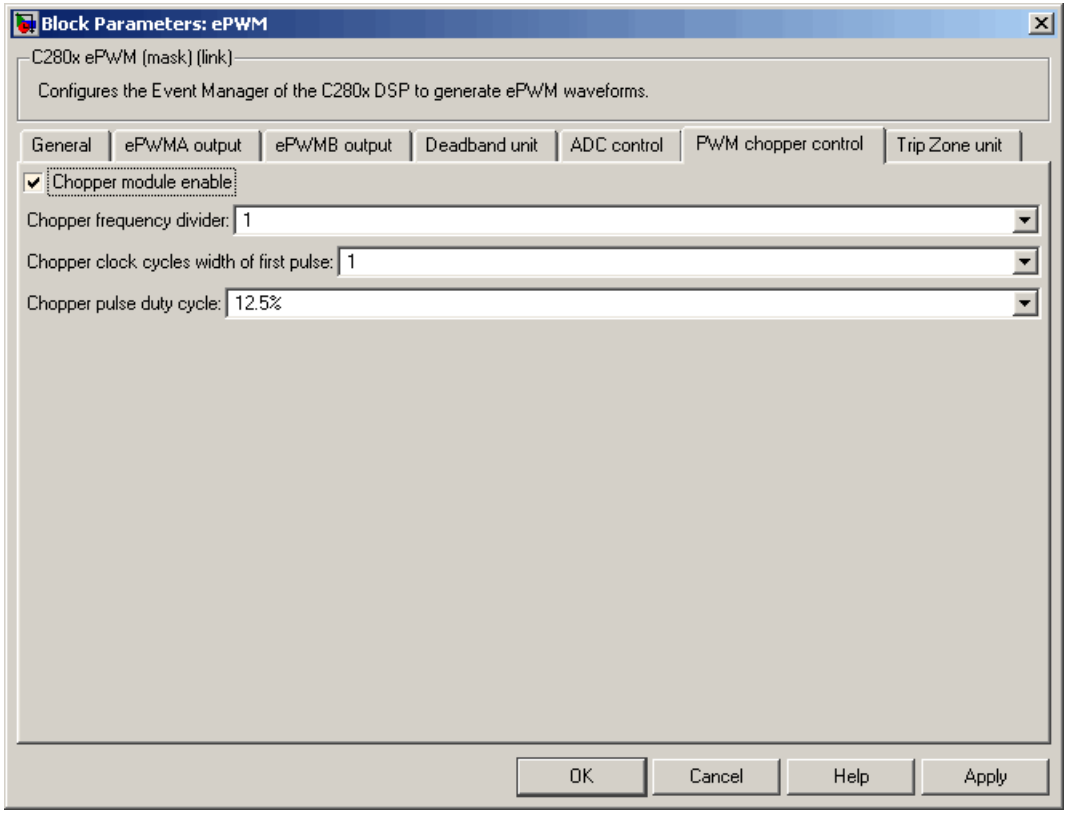
This field appears only when you select the **Enable ADC start module B** check box. Specify how often you want ADC start of conversion to be triggered. `First event` triggers ADC start of conversion with every event, `Second event` triggers ADC start of conversion with every second event, and `Third 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 to8.

Chopper clock cycles width of first pulse

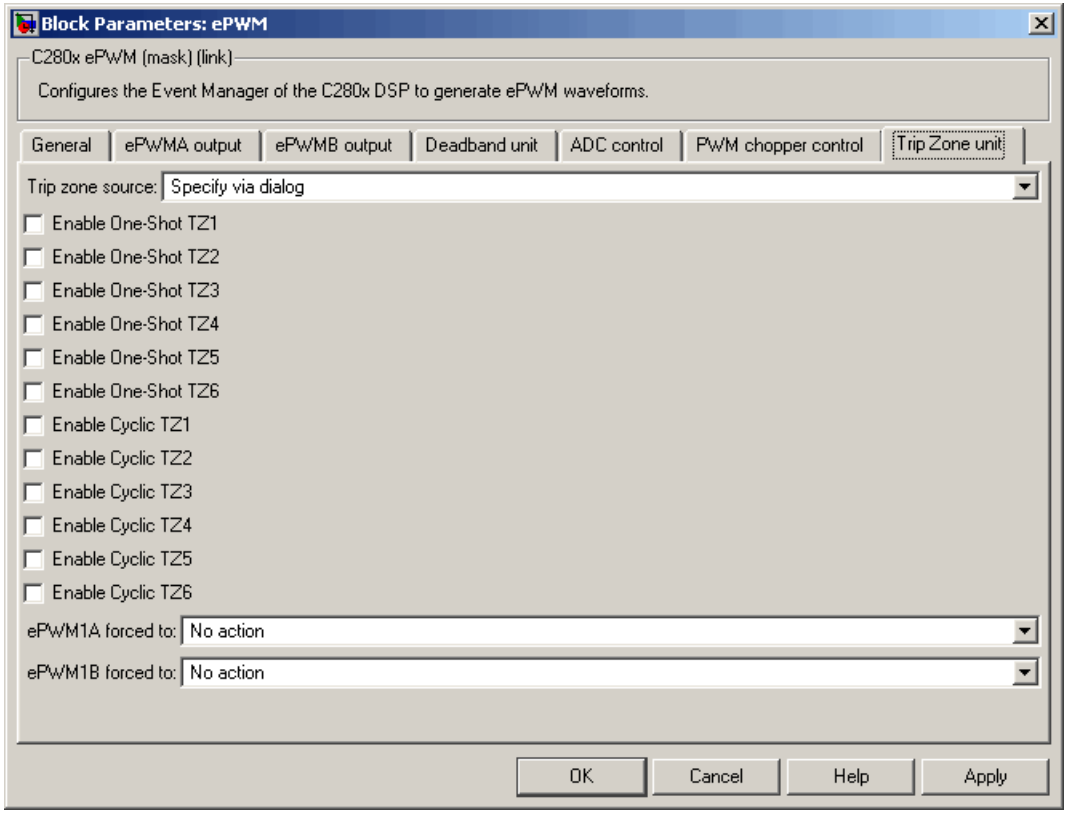
Choose an integer value from 1 to 16 to set the width of the first pulse. Use this feature to provide a high-energy first pulse to ensure hard and fast power switch turnon.

Chopper pulse duty cycle

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

Trip Zone unit pane

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



Trip zone source

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

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

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

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

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

ePWMxA forced to
ePWMxB forced to

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

See Also

C280x ADC

C280x eQEP

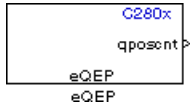
Purpose

Quadrature encoder pulse circuit

Library

`c280xdspchiplib` in Embedded Target for TI C2000 DSP

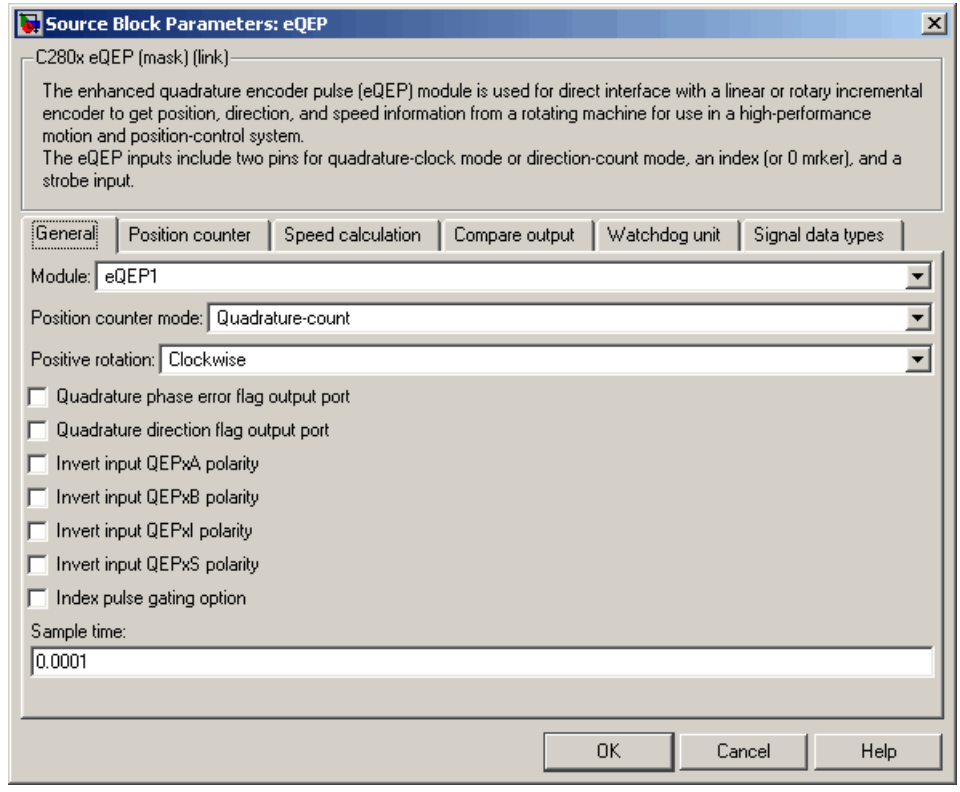
Description



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.

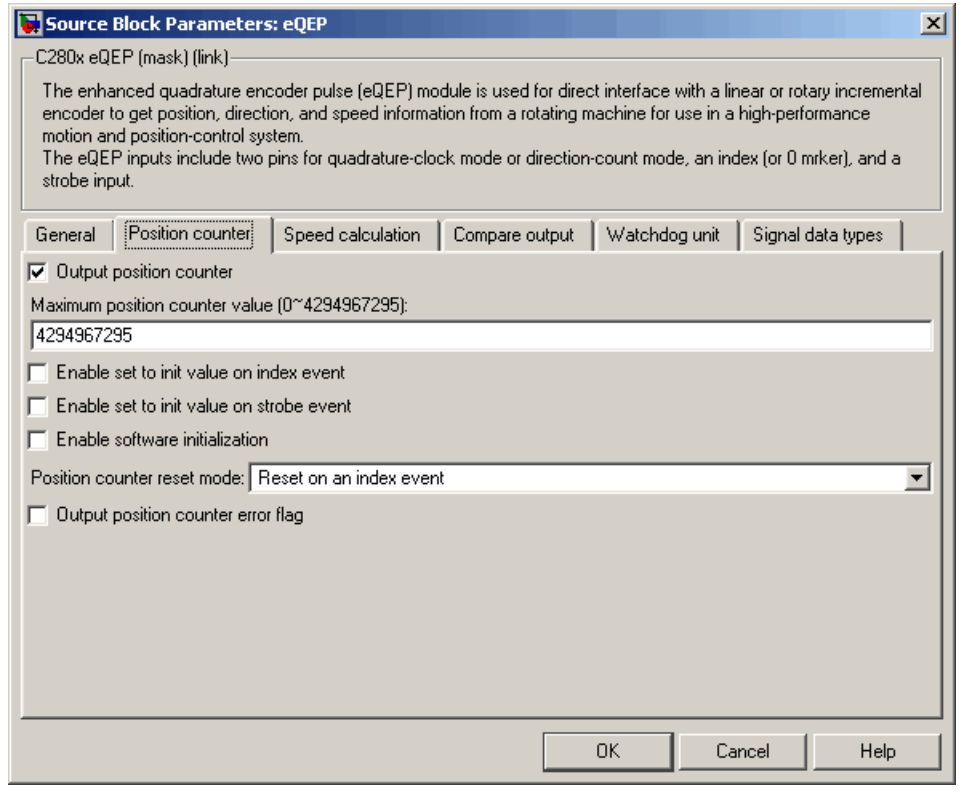
Index pulse gating option

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.

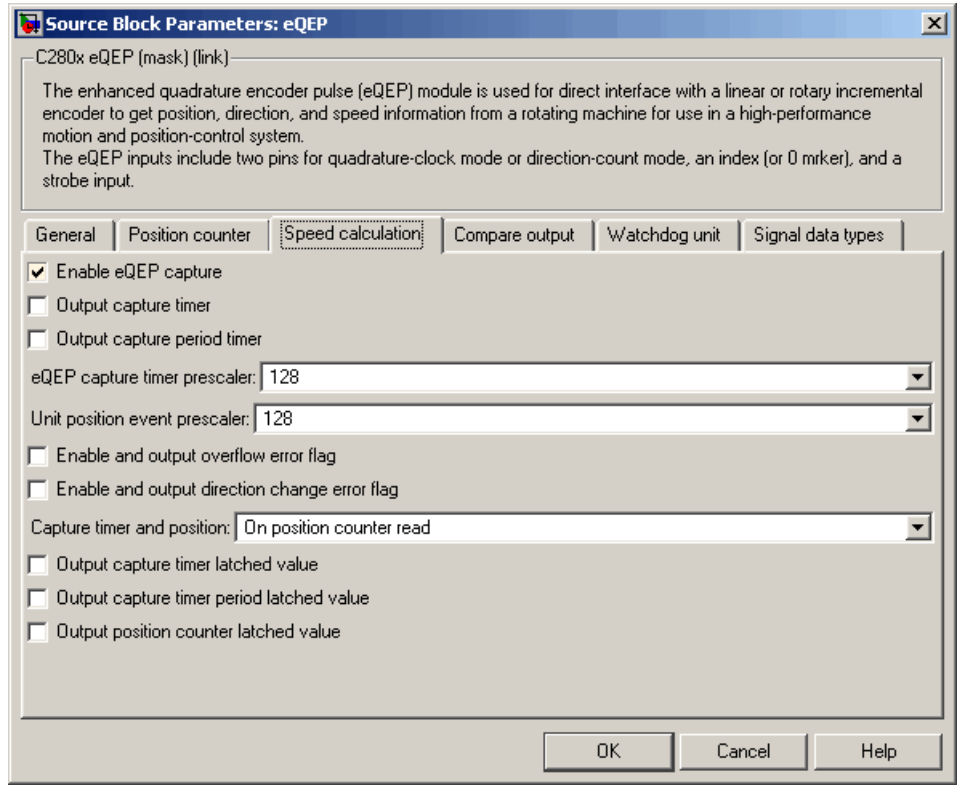
Output latch position counter on index event
This check box appears only when **Position counter reset mode** is set to `Reset on the maximum position` or `Reset on the first index event`. The eQEP index input can be configured to latch the position counter (QPOSCNT) into QPOSILAT on occurrence of a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

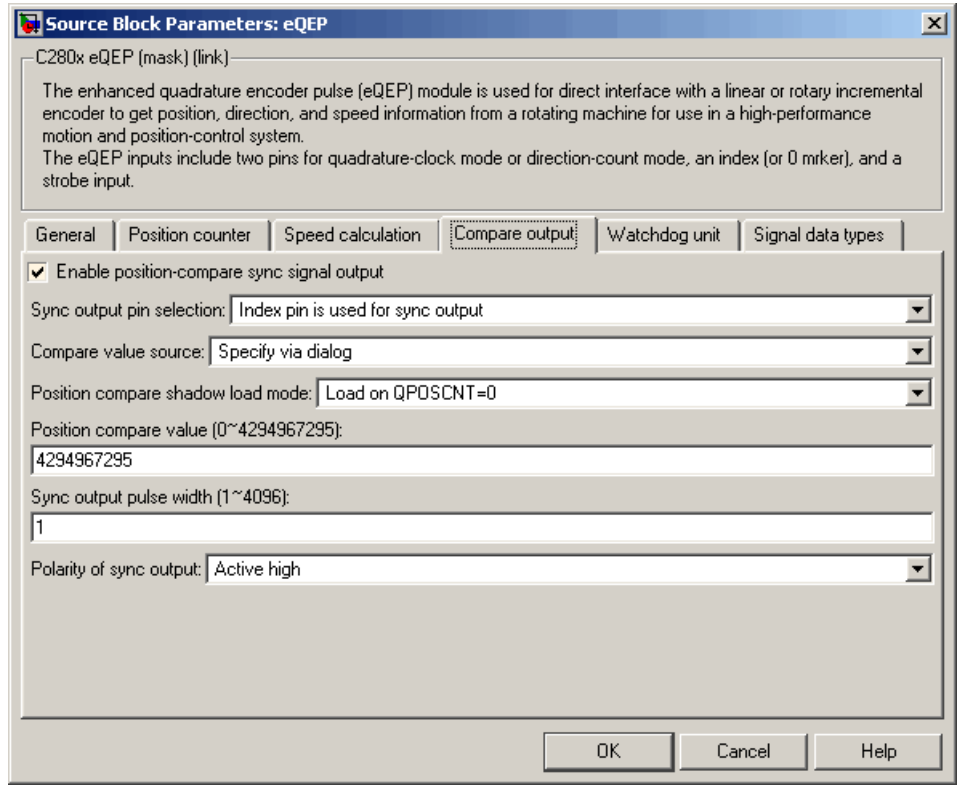
Output capture timer period latched value

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

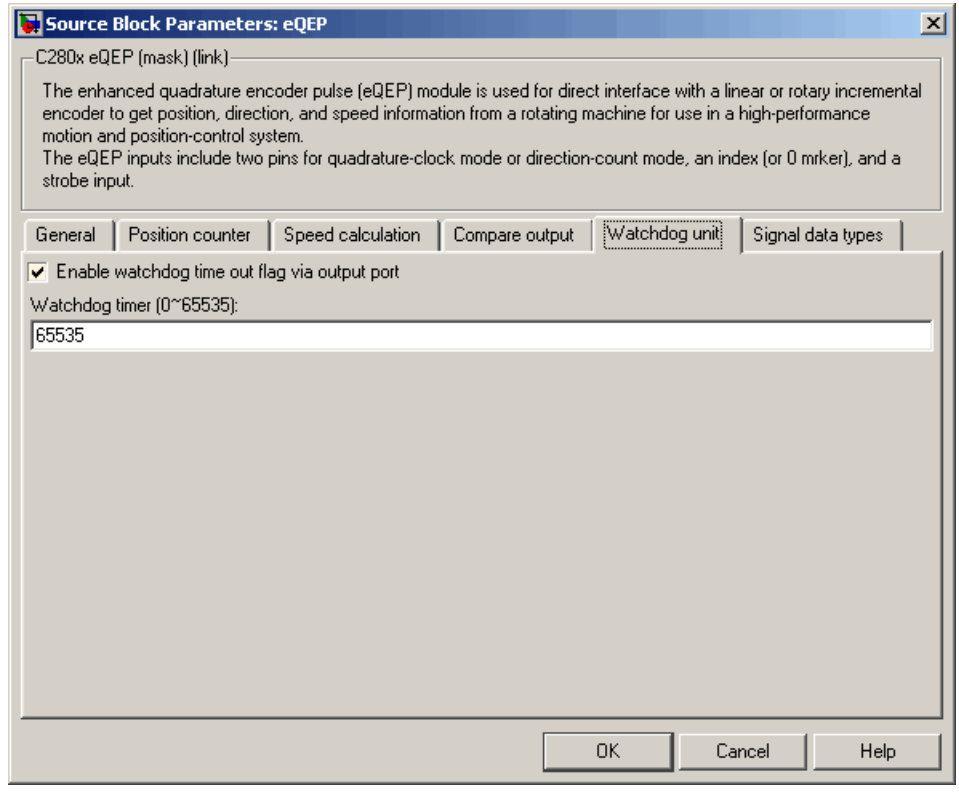
The pulse stretcher logic in the position-compare unit generates a programmable position-compare sync pulse output on the position-compare match.

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



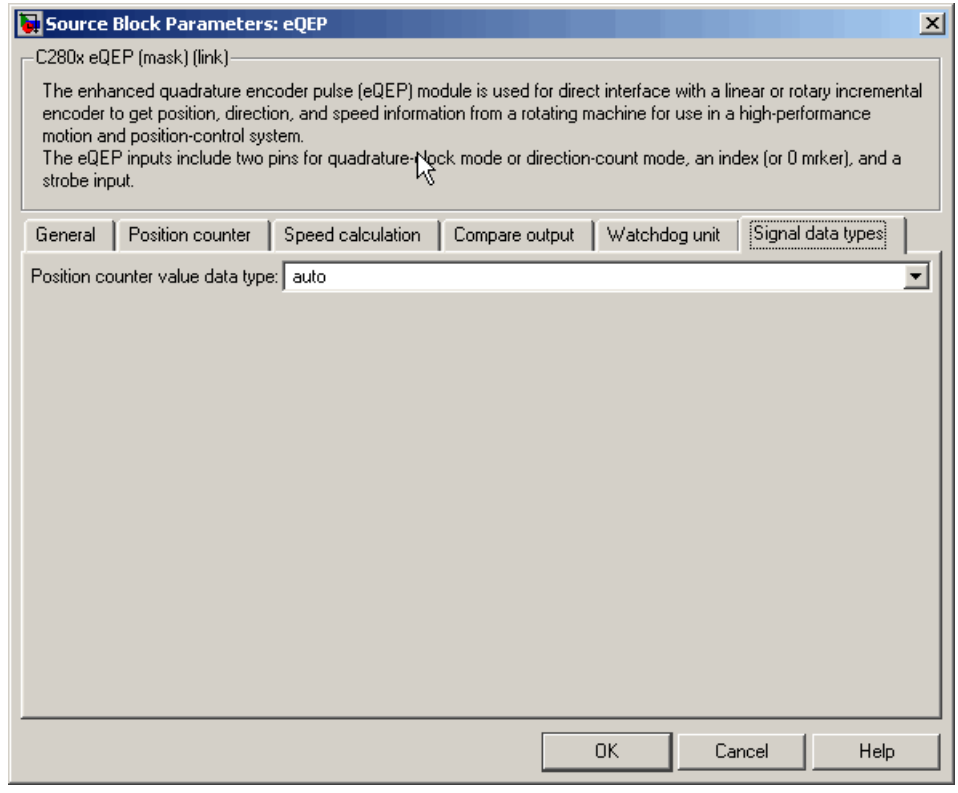
Enable watchdog time out flag via output port

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
	Output position counter latched value
Watchdog unit	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`.

C280x Hardware Interrupt

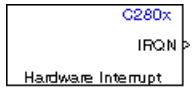
Purpose

Create an Interrupt Service Routine to handle hardware interrupts

Library

`c280xdspchiplib` in Embedded Target for TI C2000 DSP

Description



Hardware Interrupt

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

Vectorized Output

The output of this block includes a set of four vectors of equal length. One interrupt is represented by four elements, one from the same position in each of these vectors.

Each of the four text boxes in the dialog box for this block represents one of these vectors. The four vectors contain

- 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.

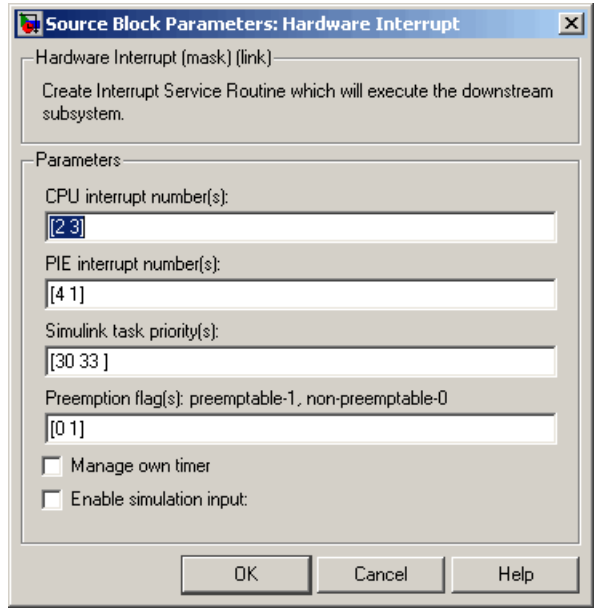
C280x Peripheral Interrupt Vector Values

	Row numbers = CPU values / Column numbers = PIE values							
	8	7	6	5	4	3	2	1
1	WAKEINT (LPM/WD)	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

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 4-96 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 4-96 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 4-94 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 4-94 for an explanation of preemption flags.

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 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

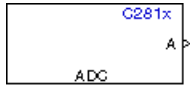
Purpose

Analog-to-digital converter (ADC)

Library

`c281xdspchiplib` 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. The ADC is usually triggered by software at the sample time intervals specified in the ADC block — this is unsynchronized mode.

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

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

Output

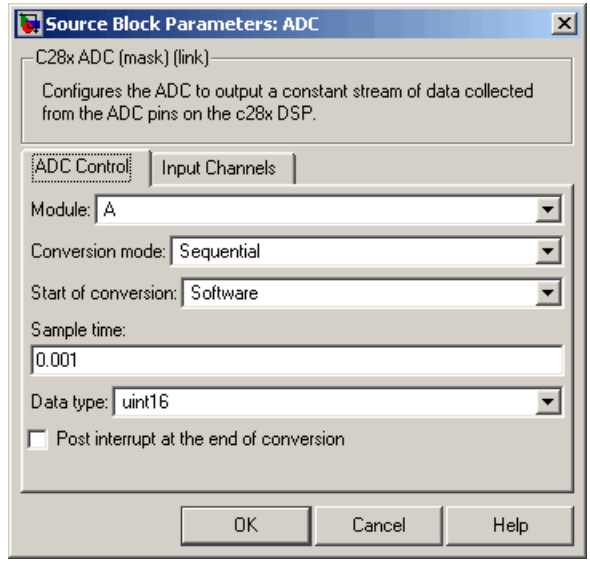
The output of the C281x ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the C281x ADC is 12-bit converter.

Modes

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

Dialog Box

ADC Control pane



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 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.

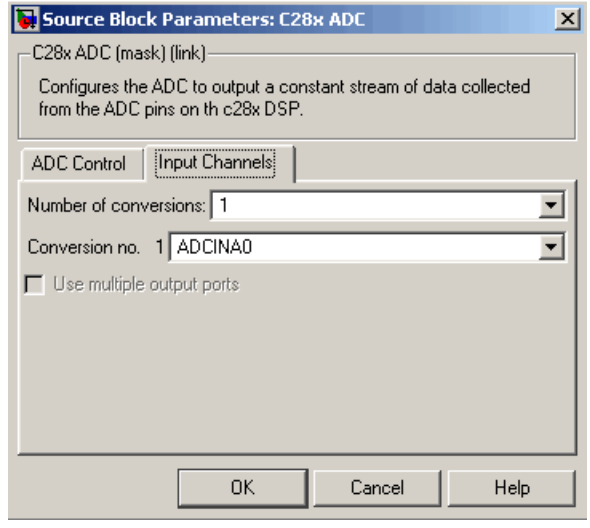
Data type

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

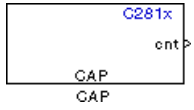
Purpose

Receive and log capture input pin transitions

Library

`c281xdspchiplib` 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

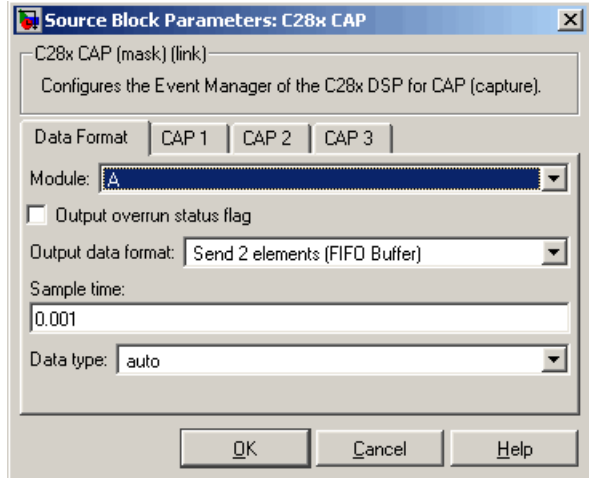
This block has up to two outputs: a `cnt` (count) output and an optional, FIFO status `flag` output. The `cnt` output increments each time a transition of the selected type occurs. The status flag outputs are

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

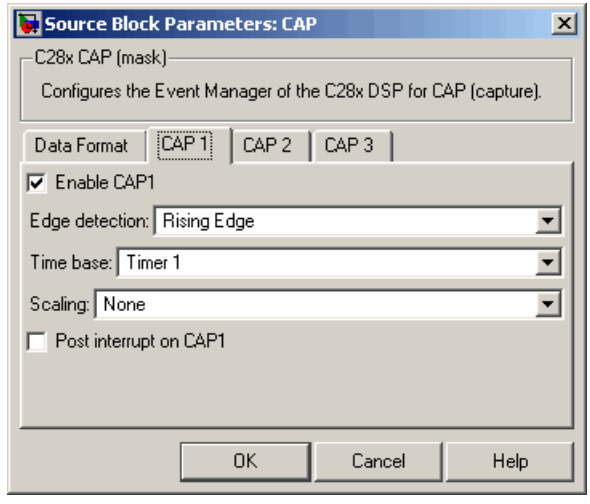
Sample time

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

Data type

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

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 (µs)
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

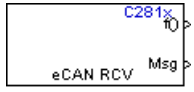
Purpose

Enhanced Control Area Network receive mailbox

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description



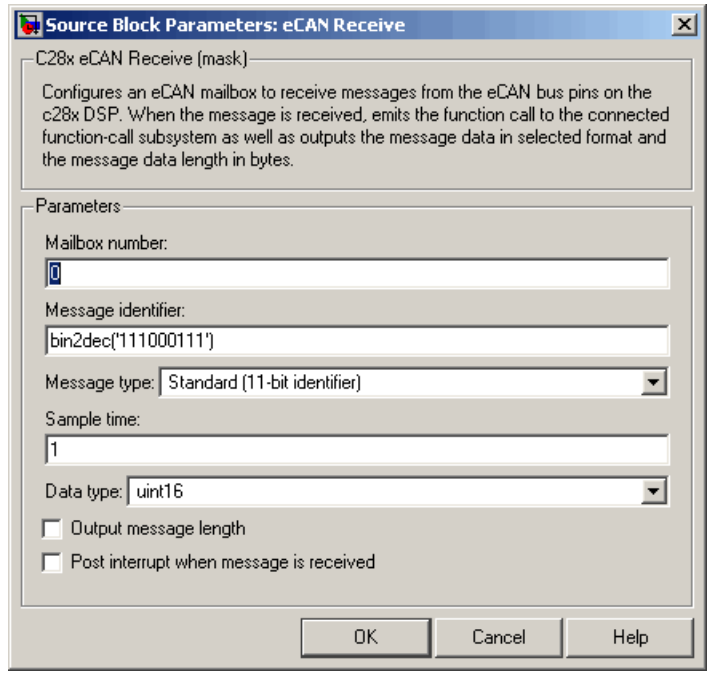
eCAN Receive

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

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

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

Dialog Box



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.

Data type

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

For `uint16` data,

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

For `uint32` data,

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

For example, if the received message has two bytes:

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

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

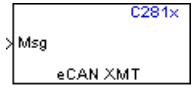
Purpose

Enhanced Control Area Network transmit mailbox

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description



eCAN Transmit

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:

```
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
```

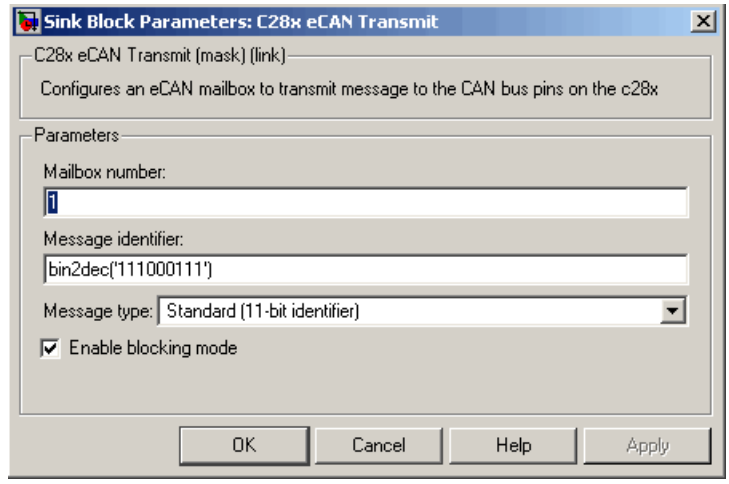
For input of type `uint16[2]`, which is a two-element vector,

```
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.

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

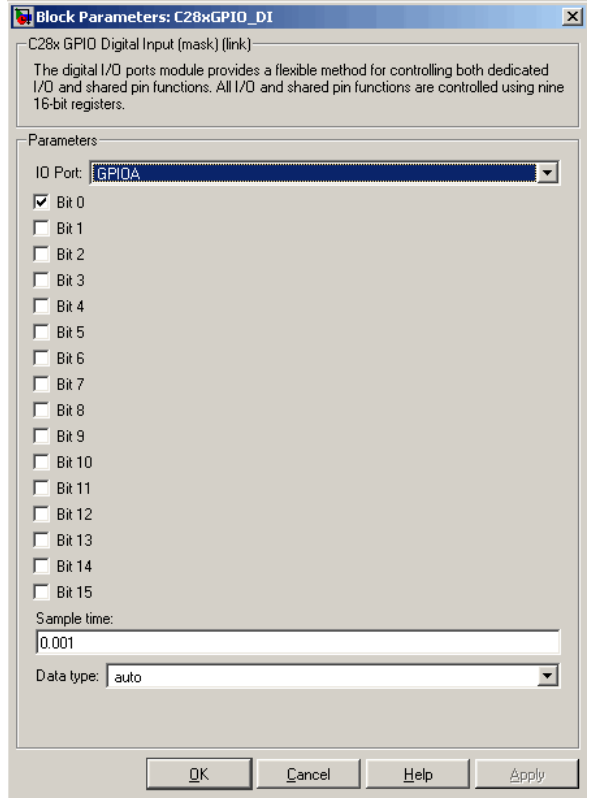
Library

`c281xdspchiplib` 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: `GPIOPA`, `GPIOPB`, `GPIOPD`, `GPIOPE`, `GPIOPF`, or `GPIOPG` and select the I/O Port bits to enable for digital input. (Note that there is no GPIOPC 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

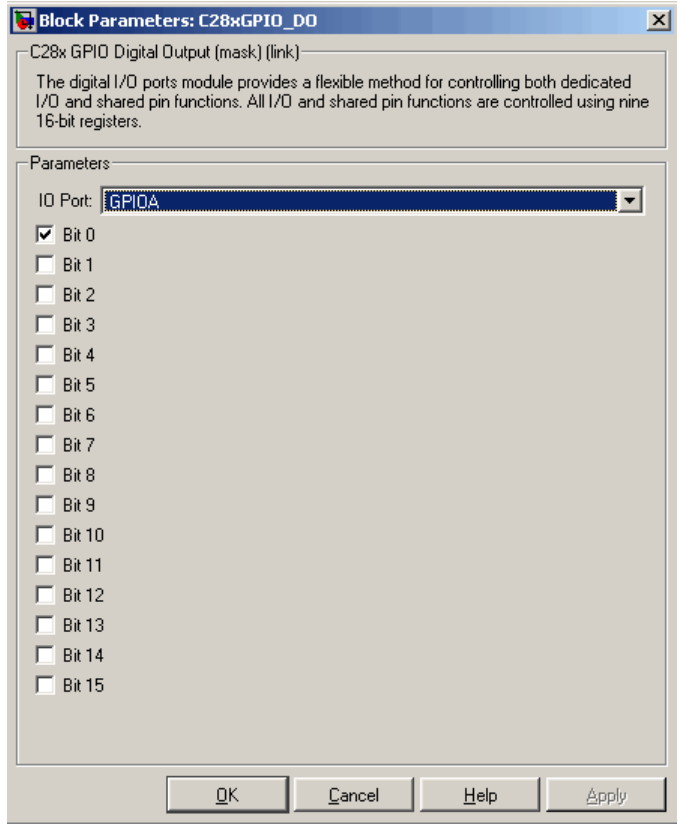
Library

`c281xdspchiplib` 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: `GPIOPA`, `GPIOPB`, `GPIOPD`, `GPIOPE`, `GPIOPF`, or `GPIOPG` and select the I/O Port bits to enable for digital input. (Note that there is no GPIOPC 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

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

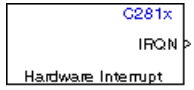
Purpose

Create an Interrupt Service Routine to handle hardware interrupts

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description



Hardware Interrupt

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 QEP
- C281x SCI Receive
- C281x SCI Transmit
- C281x SPI Receive

Vectorized Output

The output of this block includes a set of four vectors of equal length. One interrupt is represented by four elements, one from the same position in each of these vectors.

Each of the four text boxes in the dialog box for this block represents one of these vectors. The four vectors contain

- 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.

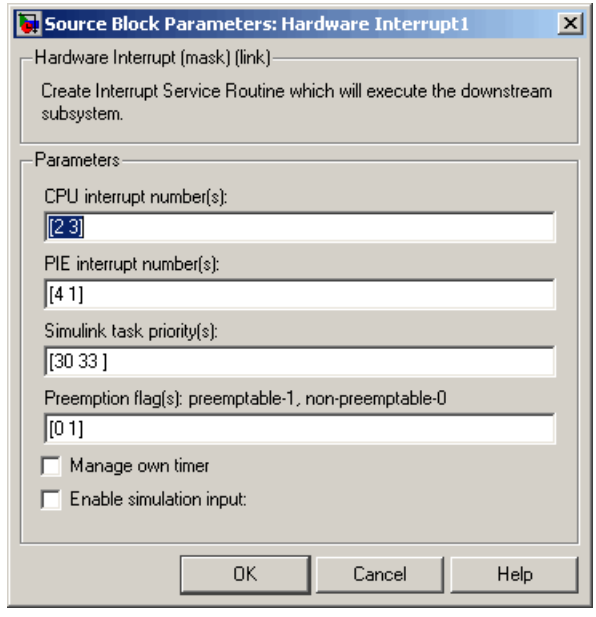
C281x Peripheral Interrupt Vector Values

	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

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 4-127 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 4-127 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 4-125 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 4-125 for an explanation of preemption flags.

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 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

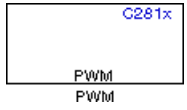
Purpose

Pulse wave modulators (PWMs)

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description

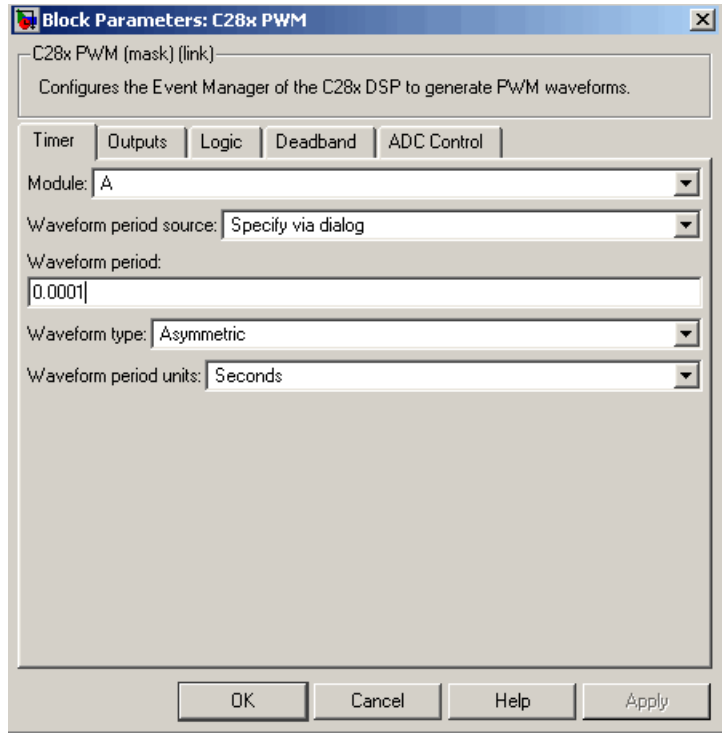


F2812 DSPs include a suite of pulse width modulators (PWMs) used to generate various signals. This block provides options to set the A or B module Event Managers to generate the waveforms you require. The twelve PWMs are configured in six pairs, with three pairs in each module.

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.

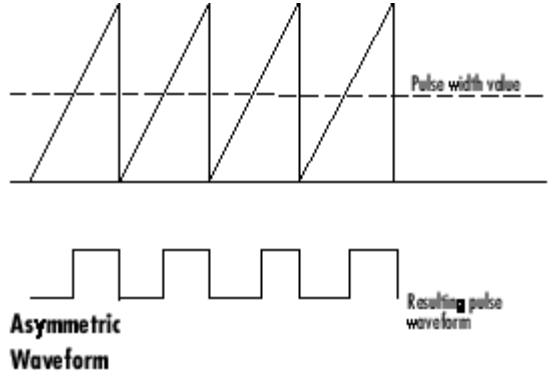
Waveform period

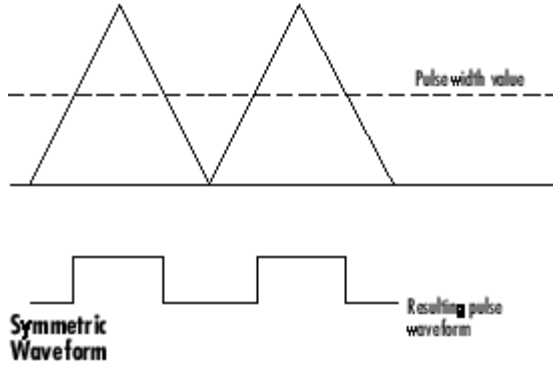
Period of the PWM waveform measured in clock cycles or in seconds, as specified in the **Waveform period units**.

Note "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.

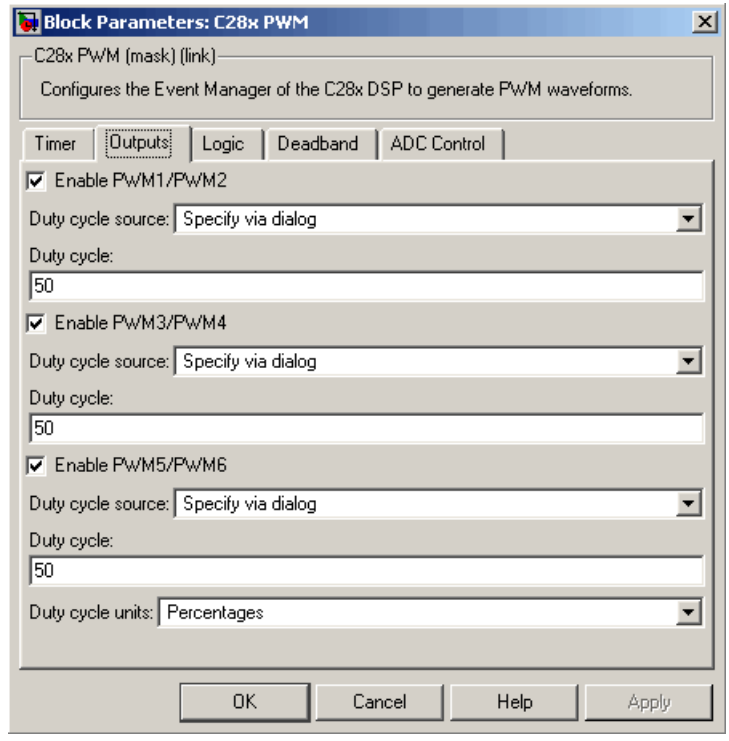




Waveform period units

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.

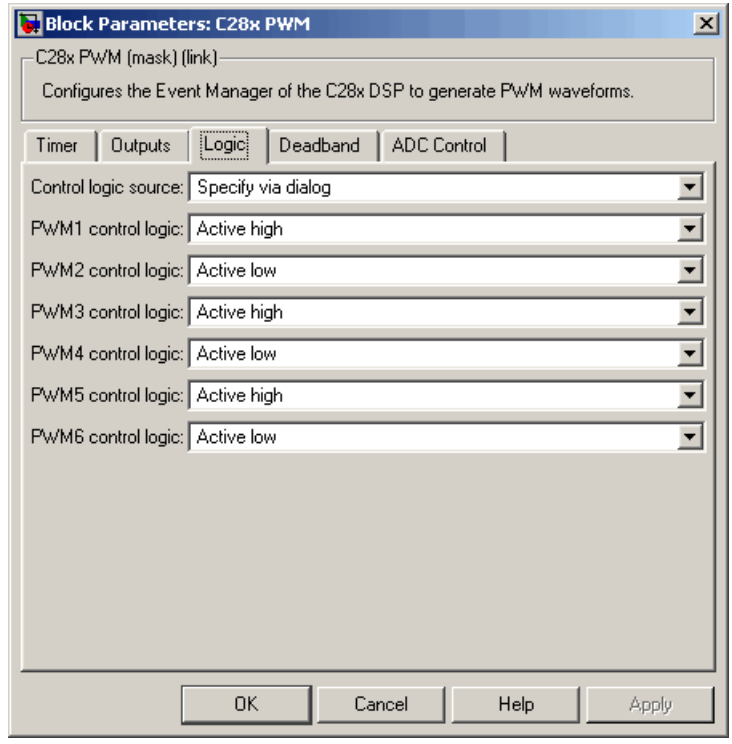
Duty cycle

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



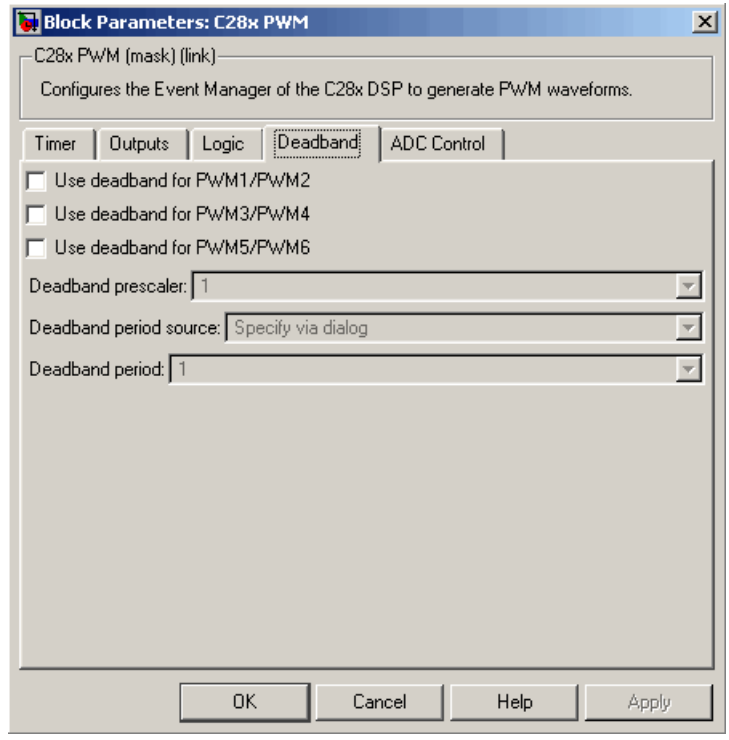
Control logic source

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

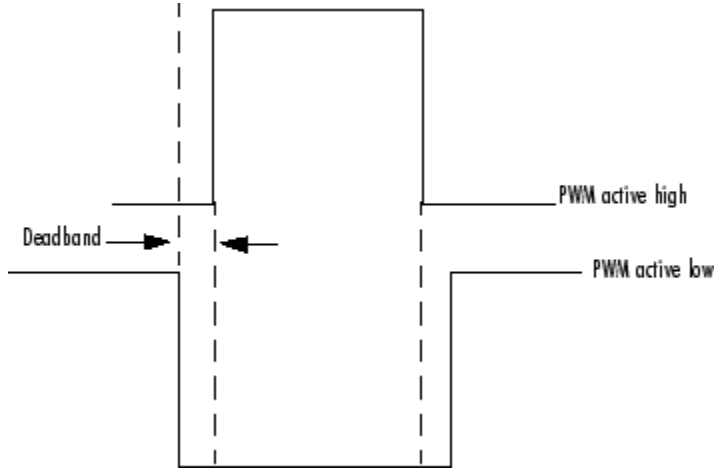
Control logic trigger for the PWM. `Forced high` causes the pulse value to be high. `Active high` causes the pulse value to go from low to high and `Active low` causes the pulse value to go from high to low. `Forced low` causes the pulse value to be low.

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.

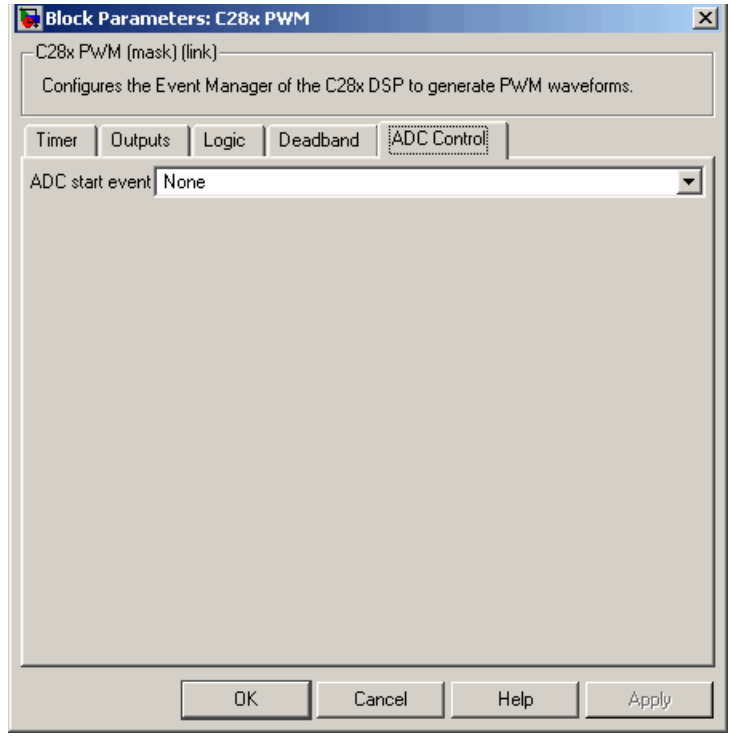
Deadband period source

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 MHz = 8.5333e-004. If you enter a **Sample time** in the C281x ADC dialog 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 **Pulse width** above determines the value in the register.

See Also

C281x ADC

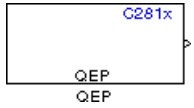
Purpose

Quadrature encoder pulse circuit

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description

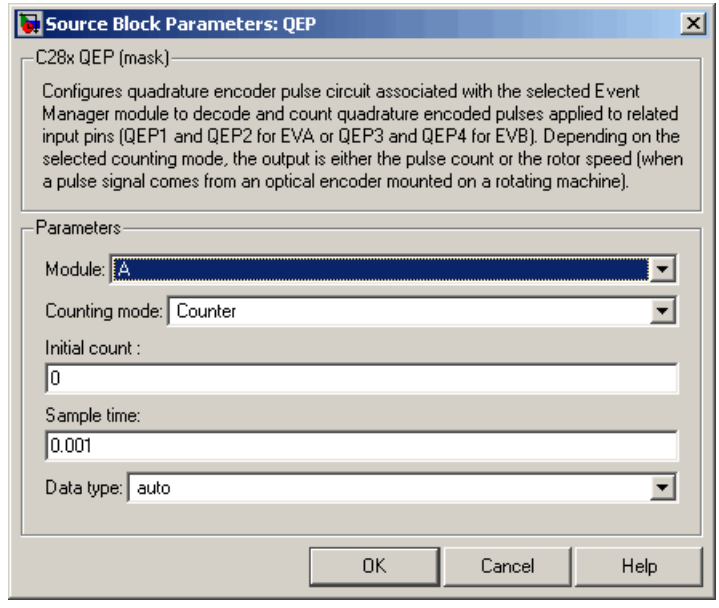


Each F2812 Event Manager has three capture units, which can log transitions on its capture unit pins. Event manager A (EVA) uses capture units 1, 2, and 3. Event manager B (EVB) uses capture units 4, 5, and 6.

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`.

C281x SCI Receive

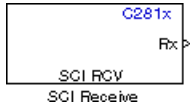
Purpose

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

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description



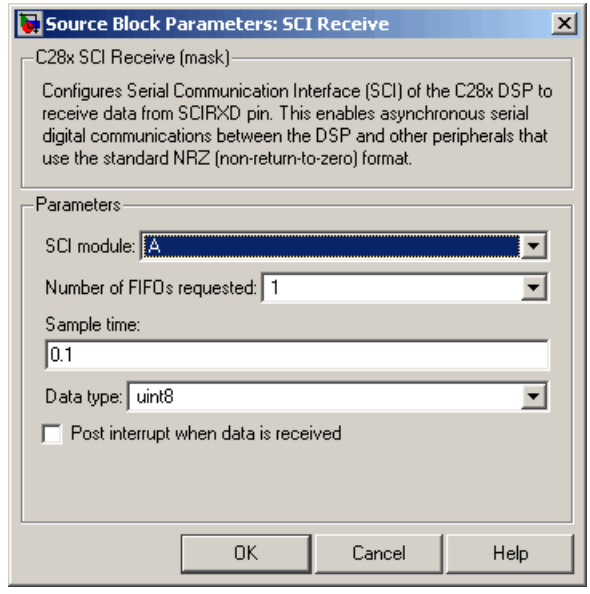
SCI Receive

The C281x SCI Receive block supports asynchronous serial digital communications between the target and other asynchronous peripherals in non-return-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 You can have only one C281x SCI Receive block in a single model.

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.

Number of FIFOs requested
Number of elements to be read from the hardware FIFO.

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`.

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

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

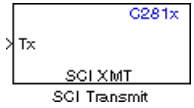
Purpose

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

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description



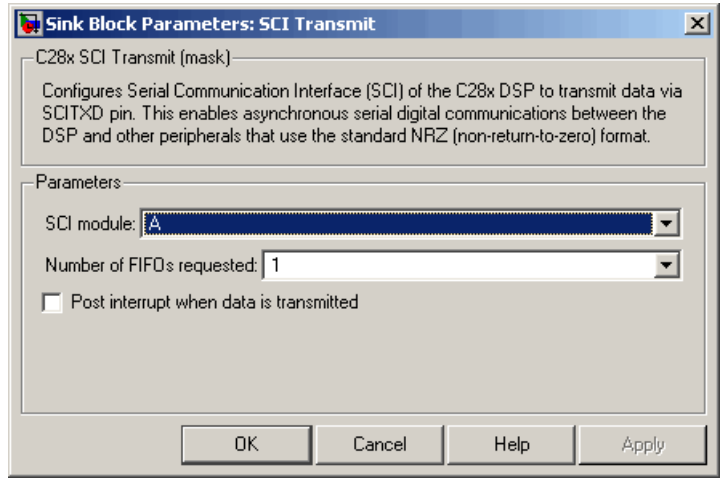
SCI Transmit

The C281x SCI Transmit block transmits scalar or vector data in `int8` or `uint8` format from the C281x target's COM ports in non-return-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 C281x SCI Transmit block in a single model.

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.

Number of FIFOs requested
Number of elements to be transmitted from the hardware FIFO.

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

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

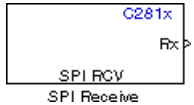
Purpose

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

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description



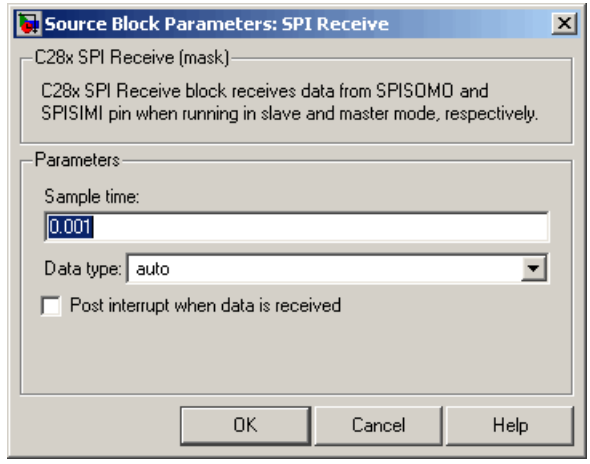
SPI Receive

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 SPISIM0 pin transmits data and the SPISOM1 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 C281x SPI Receive block in a single model.

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



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`, and `uint32`.

Post interrupt when data is received

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

See Also

C281x SPI Transmit, C281x Hardware Interrupt

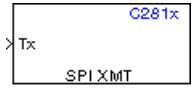
Purpose

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

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

Description



SPI Transmit

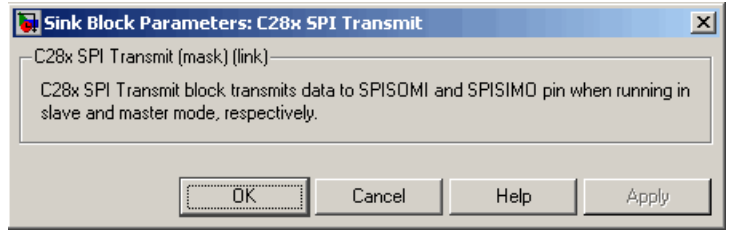
The C281x SPI Transmit supports synchronous, serial peripheral input/output port communications between the DSP controller and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIM0 pin transmits data and the SPISOM1 pin receives data. 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 C281x SPI Transmit block in a single model.

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



See Also

C281x SPI Receive

C281x Timer

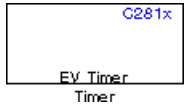
Purpose

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

Library

`c281xdspchiplib` in Embedded Target for TI C2000 DSP

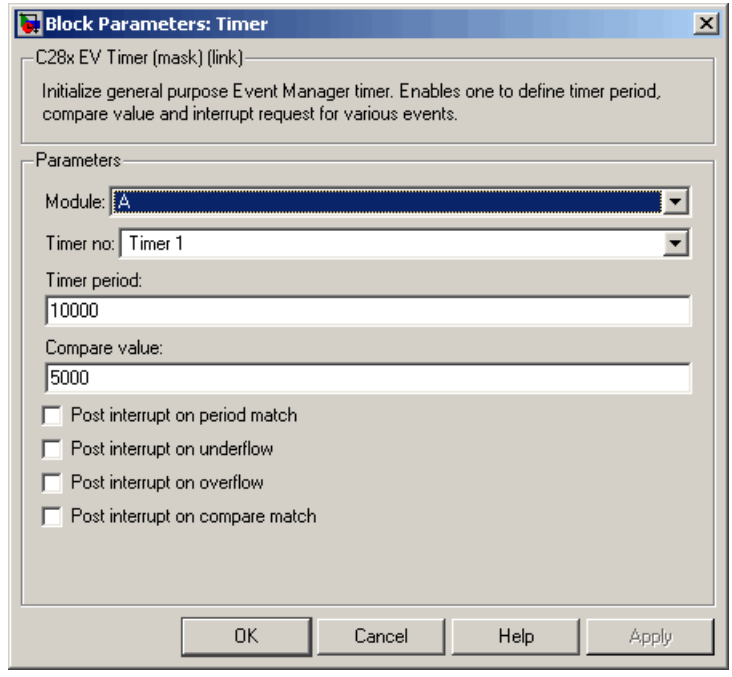
Description



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`.

Note "Clock cycles" refers to the high-speed peripheral clock on the C281x chip. This clock is 75 MHz by default because the high-speed peripheral clock prescaler is set to 2 (150 MHz/2).

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

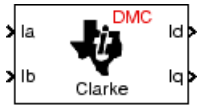
Purpose

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

Library

`c28xdmclib` in Embedded Target for TI C2000 DSP

Description



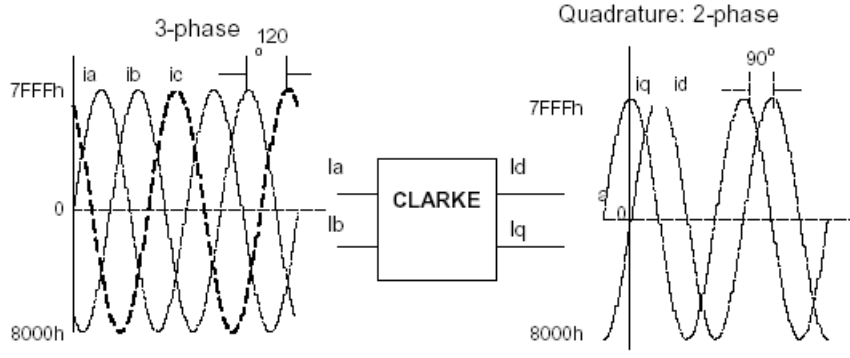
Clarke Transformation

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

"*Id = Ia*"

$$Iq = (2Ib + Ia)/\sqrt{3}$$

and is illustrated in the following figure.



The inputs to this block are the phase a (`Ia`) and phase b (`Ib`) components of the balanced three-phase quantities and the outputs are the direct axis (`Id`) component and the quadrature axis (`Iq`) of the transformed signal.

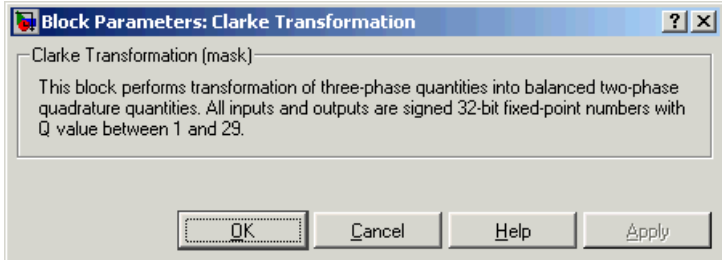
The instantaneous outputs are defined by the following equations:

"*id = I sin(wt)*"

"*iq = I sin(wt + π/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 2-2 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

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

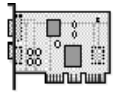
Purpose

Target preferences for custom C280x board

Library

`c2000tgtpreflib` in Embedded Target for TI C2000 DSP

Description

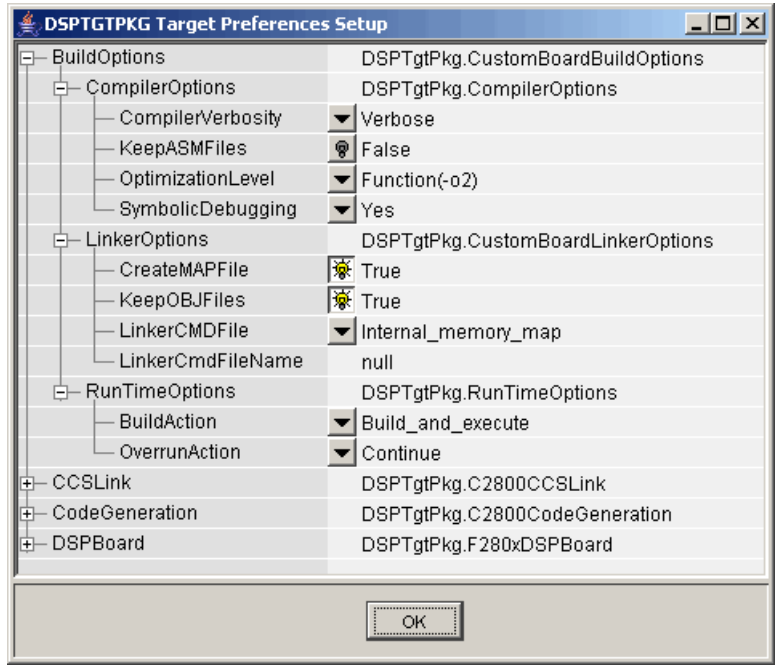


Options on the block mask let you set features of code generation for your custom board based on a C2801, C2806, or C2808 chip. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

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.

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 that if you use `Internal_memory_map`, specifying a **LinkerCmdFileName** has no effect.

- `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.

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 is the default.

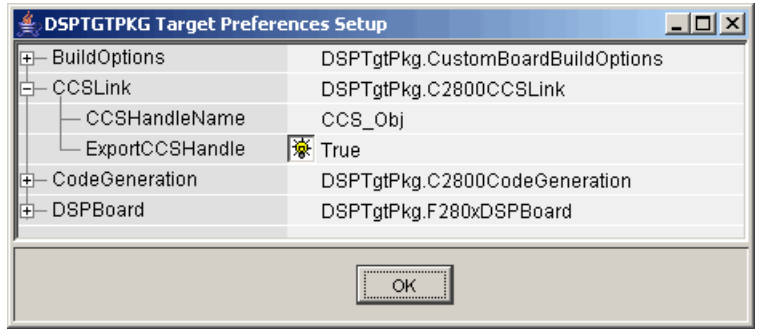
Note When you build and execute a model on your target, Real-Time Workshop resets the target automatically. You do not need to reset the board before building models.

OverrunAction

Defines the action to take when an interrupt overrun occurs:

- `Continue` — Ignore overruns encountered while running the model. This is the default.
- `Halt` — Stop program execution.

CCSLink



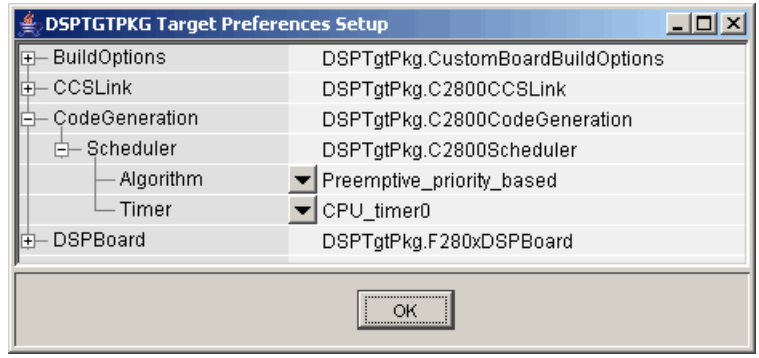
CCSHandleName

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 is set to `True` (the default), then after you build your model, you will see the CCS object in your MATLAB workspace browser with the name you provided and class type `ccsdsp`.

CodeGeneration — Scheduler



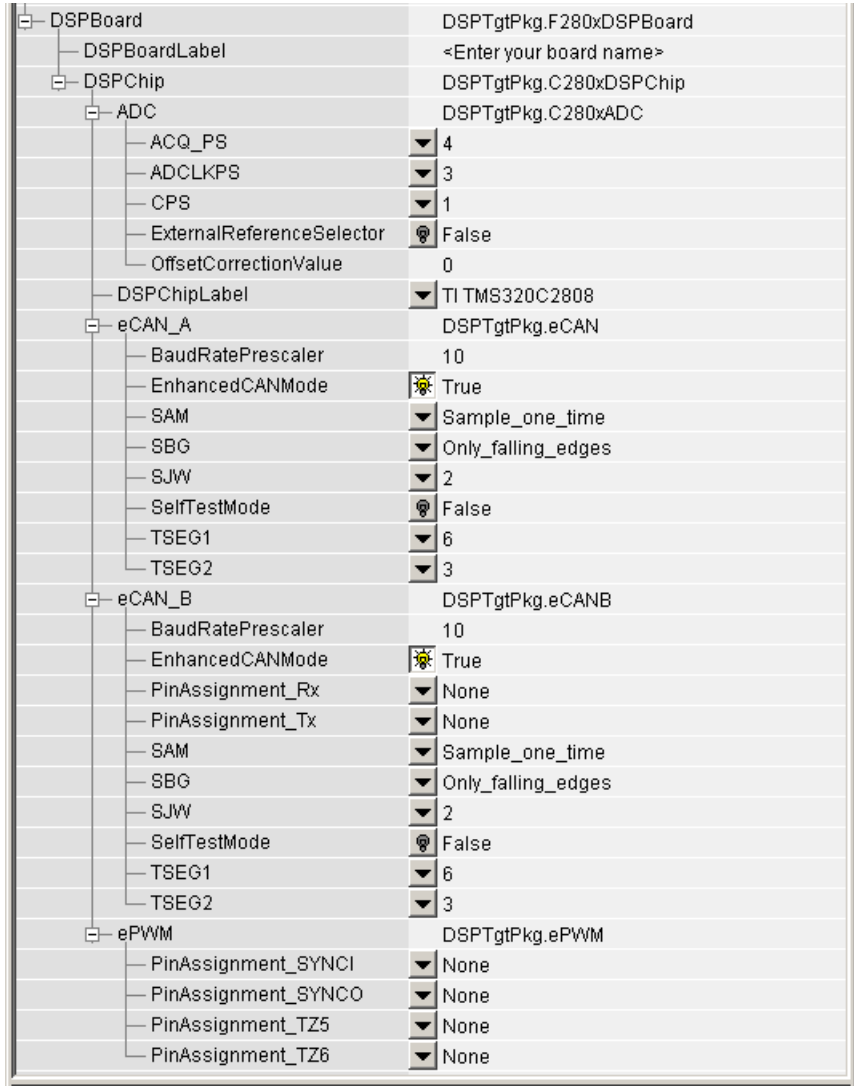
Algorithm

Algorithm to use for scheduling. The algorithm options are

- `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.
- `Free_running` — This scheduler does not use any interrupts. Tasks run in priority-based order and the execution of each task depends only on how fast the task can run on the given processor. This algorithm does not support preemption or multitasking. (Selecting `MultiTasking` as the Tasking mode in **Configuration Parameters-Solver** is not allowed for this scheduling.) Overruns do not occur with this type of scheduling, so any value in **BuildOptions — RuntimeOptions** `OverrunAction` is ignored.

Timer
CPU timer to use for scheduling.

DSPBoard


DSPBoard
DSPTgtPkg.F280xDSPBoard
DSPBoardLabel
<Enter your board name>
DSPChip
DSPTgtPkg.C280xDSPChip
ADC
DSPTgtPkg.C280xADC
ACQ_PS
4
ADCLKPS
3
CPS
1
ExternalReferenceSelector
False
OffsetCorrectionValue
0
DSPChipLabel
TI TMS320C2808
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_SYNCI
None
PinAssignment_SYNCO
None
PinAssignment_TZ5
None
PinAssignment_TZ6
None

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

DSPChipLabel

DSP chip model. All supported C280x chips are listed. Select the DSP chip installed on your target. The selected value defaults to `TI TMS320C2808`. Other available options are `TI TMS320C2801` and `TI TMS320C2806`.

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. The following are the settable 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/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 will be 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`.

eCAN_A

The settable parameters 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 the **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. **TSEG1** must be greater than **TSEG2** and the Information Processing Time (IPT). The IPT is the time needed to process one bit and corresponds to two TQ units. `TSEG1 = PROP_SEG + PHASE_SEG1`. Valid values for **TSEG1** are from `1` through `16`.

TSEG2

Sets the value of time segment 2 (PHASE_SEG2), which, with **TSEG1** and the **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. **TSEG2** must be less than or equal to **TSEG1** and greater than or equal to IPT. Valid values for **TSEG2** are from `1` through `8`.

eCAN_B

The settable parameters 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

Custom C281x Board

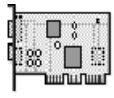
Purpose

Target preferences for custom C281x board

Library

`c2000tgtpreflib` in Embedded Target for TI C2000 DSP

Description

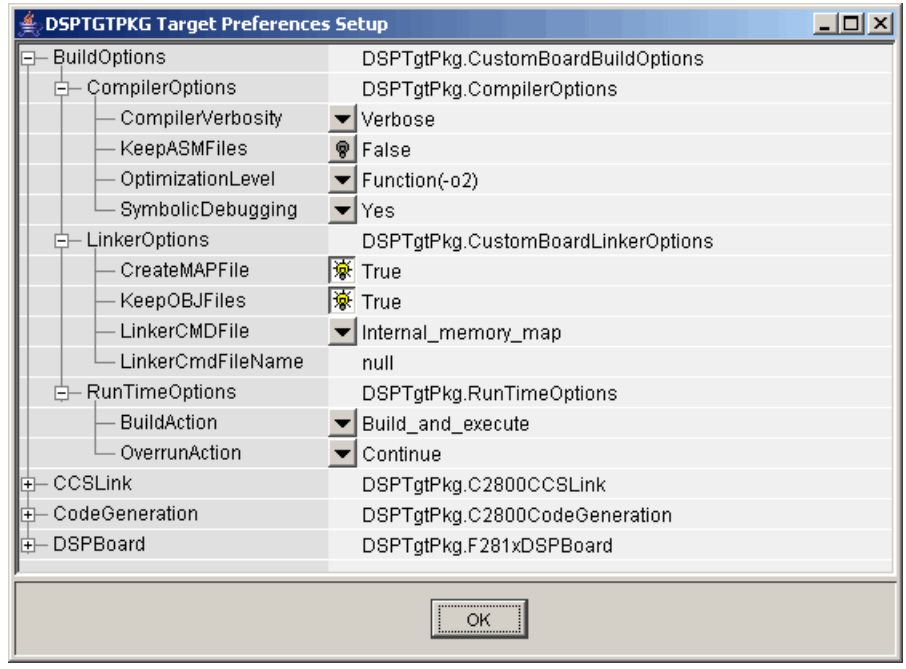


Options on the block mask let you set features of code generation for your custom board based on a C2810, F2810, C2811, F2811, R2811, C2812, F2812, or R2812 chip. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the 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.

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'
  ```

  Note that if you use `Internal_memory_map`, specifying a **LinkerCmdFileName** has no effect.

- `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.

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 is the default.

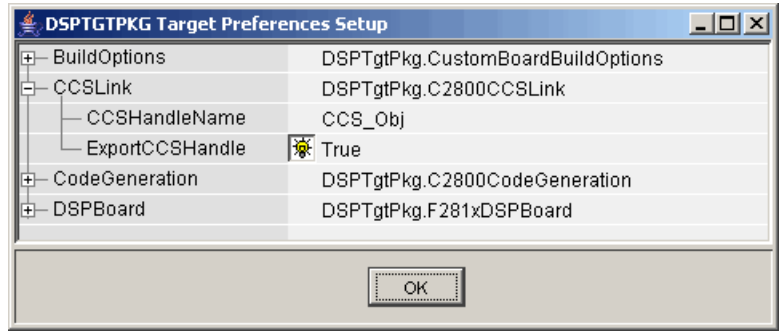
Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

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



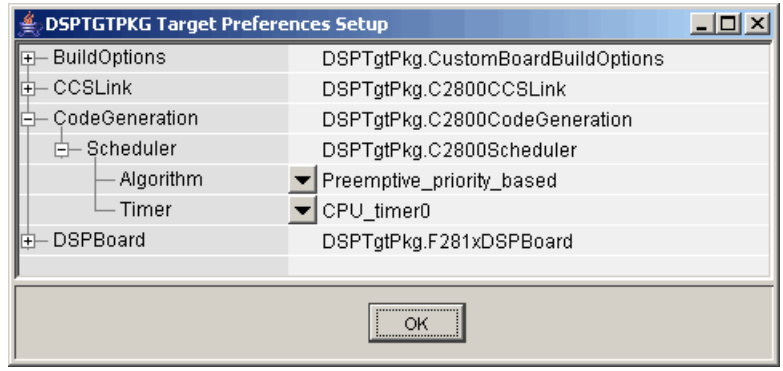
CCSHandleName

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 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 `ccsdsp`.

CodeGeneration — Scheduler



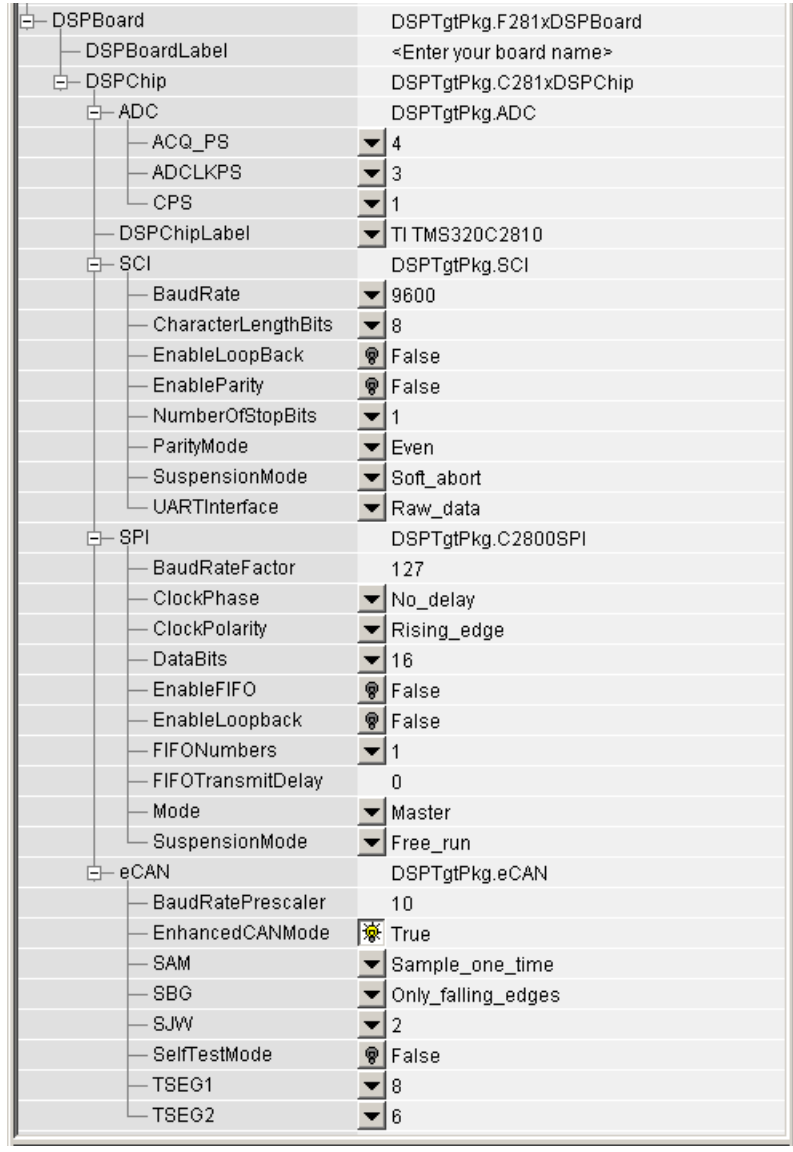
Algorithm

Algorithm to use for scheduling. The algorithm options are

- `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.
- `Free_running` — This scheduler does not use any interrupts. Tasks run in priority-based order and the execution of each task depends only on how fast the task can run on the given processor. This algorithm does not support preemption or multitasking. (Selecting `MultiTasking` as the Tasking mode in **Configuration Parameters-Solver** is not allowed for this scheduling.) Overruns do not occur with this type of scheduling, so any value in **BuildOptions — RuntimeOptions** `OverrunAction` is ignored.

Timer
CPU timer to use for scheduling.

DSPBoard


DSPBoard
DSPTgtPkg.F281xDSPBoard
DSPBoardLabel
<Enter your board name>
DSPChip
DSPTgtPkg.C281xDSPChip
ADC
DSPTgtPkg.ADC
ACQ_PS
4
ADCLKPS
3
CPS
1
DSPChipLabel
TI TMS320C2810
SCI
DSPTgtPkg.SCI
BaudRate
9600
CharacterLengthBits
8
EnableLoopBack
False
EnableParity
False
NumberOfStopBits
1
ParityMode
Even
SuspensionMode
Soft_abort
UARTInterface
Raw_data
SPI
DSPTgtPkg.C2800SPI
BaudRateFactor
127
ClockPhase
No_delay
ClockPolarity
Rising_edge
DataBits
16
EnableFIFO
False
EnableLoopback
False
FIFONumbers
1
FIFOTransmitDelay
0
Mode
Master
SuspensionMode
Free_run
eCAN
DSPTgtPkg.eCAN
BaudRatePrescaler
10
EnhancedCANMode
True
SAM
Sample_one_time
SBG
Only_falling_edges
SJW
2
SelfTestMode
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 the label (name) of the board entered in your Code Composer Studio setup.

DSPBoard — DSPChip

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
```

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. The following are the settable 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/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.

SCI

Parameters that affect the serial communications interfaces (SCIs) on the target. The settable parameters are

BaudRate

Baud rate for transmitting and receiving data. Choices are `115200`, `57600`, `38400`, `19200`, `9600` (the default), `4800`, `2400`, `1200`, `300`, and `100`.

CharacterLengthBits

Length in bits from `1` to `8` of each transmitted/received character.

EnableLoopBack

Select `True` to enable the loopback function for self-test and diagnostic purposes only. When this is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and it 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/receive data.

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.

UARTInterface

Protocol to use when sending or receiving UART mode data. Although available protocols are `Raw_data` and `To/From_host_block`, only `Raw_data` is supported. `Raw_data` sends or receives all data in its raw format, one byte at a time. Since the C281x SCI module has a 16-byte FIFO buffer, the C281x SCI Receive and Transmit blocks can receive and transmit scalar or vector data.

`To/From_host_block` is not supported currently and is provided only for use in demos. It uses the serial communication interface to communicate with host-side SCI blocks. It attempts to read and interpret a specified number of elements via a `for` loop using internal protocol.

SPI

Parameters that affect the serial peripheral interface (SPI) on the target. The settable parameters are

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

Baud Rate = CPU Rate / (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.

EnableFIFO
Select `True` to enable the FIFO buffers in the SPI module.

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 is enabled, a C281x DSP's SIMO/SOMI lines are connected internally.

FIFONumbers
Enter the number of FIFO buffers to enable. You can specify `1` to `16` buffers.

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 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/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.

eCAN Bit Timing

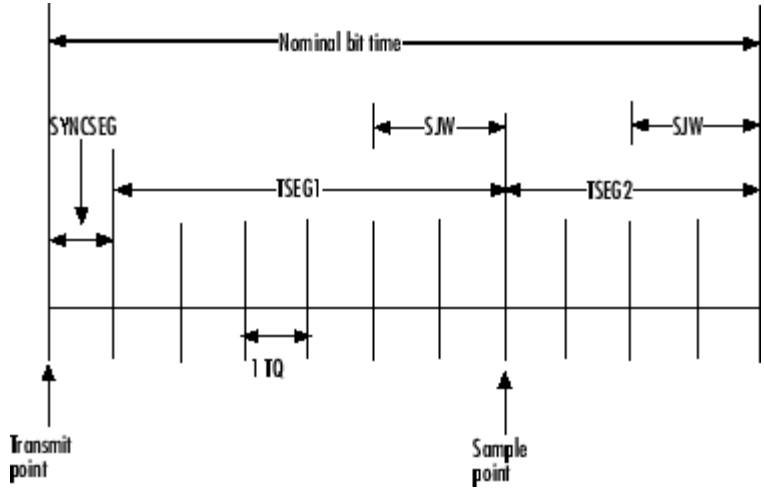
The eCAN protocol divides the nominal bit time into the following four segments:

- SYNCSEG — Time used to synchronize the nodes on the bus. It is always one time quantum (TQ), which is defined as

  $$TQ = \frac{BaudRatePrescaler}{SYSCLK}$$

  where *SYSCLK* is the CAN module system clock frequency, and the **BaudRatePrescaler** is defined below.
- PROP_SEG — Time used to compensate for physical delays in the network
- PHASE_SEG1 — Phase used to compensate for positive edge phase error
- PHASE_SEG2 — Phase used to compensate for negative edge phase error

The eCAN bit timing is shown in the following illustration.



Calculating Baud Rate

The length of a bit in the CAN module is determined by **TSEG1**, **TSEG2**, and **BaudRatePrescaler** parameters. The baud rate is

$$BaudRate = \frac{SYSCLK}{\mathbf{BaudRatePrescaler} \times BitTime}$$

where *BitTime* = *TSEG1*+*TSEG2*+1

The following table shows the corresponding baud rates (for a 150-Mhz clock as on the F2812 DSP) for the indicated parameter settings.

TSEG1	TSEG2	BaudRate Prescaler	SJW	SBG	Baud Rate
8	6	20	2	0	0.5 Mbit/s
8	6	10	2	0	1 Mbit/s
8	6	5	2	0	2 Mbit/s

For additional details, refer to the *280x Enhanced Controller Area Network (eCAN) Reference Guide*, Literature Number SPRU074C, on the Texas Instruments Web site.

The settable eCAN parameters are

BaudRatePrescaler
Value by which to scale the bit rate. Valid values are from `1` to `256`. As noted in the equation above, this value determines the value of TQ.

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. **TSEG1** must be greater than **TSEG2** and the Information Processing Time (IPT). The IPT is the time needed to process one bit and corresponds to two TQ units. `TSEG1 = PROP_SEG + PHASE_SEG1`. Valid values for **TSEG1** are from `1` through `16`.

TSEG2

Sets the value of time segment 2 (PHASE_SEG2), which, with **TSEG1** and **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. **TSEG2** must be less than or equal to **TSEG1** and greater than or equal to IPT. Valid values for **TSEG2** are from `1` through `8`.

See Also

C281x ADC, C281x eCAN Receive, C281x eCAN Transmit, C281x PWM

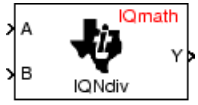
Purpose

Divide two IQ numbers

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

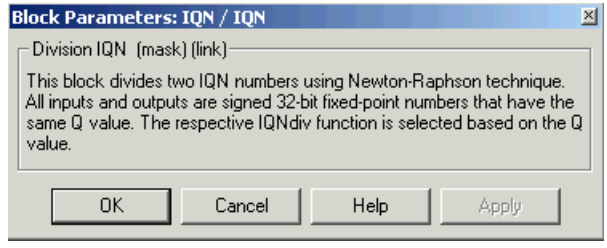


IQN / IQN

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 2-2 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

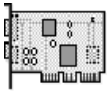
Purpose

F2808 eZdsp DSK target preferences

Library

`c2000tgtpreflib` in Embedded Target for TI C2000 DSP

Description

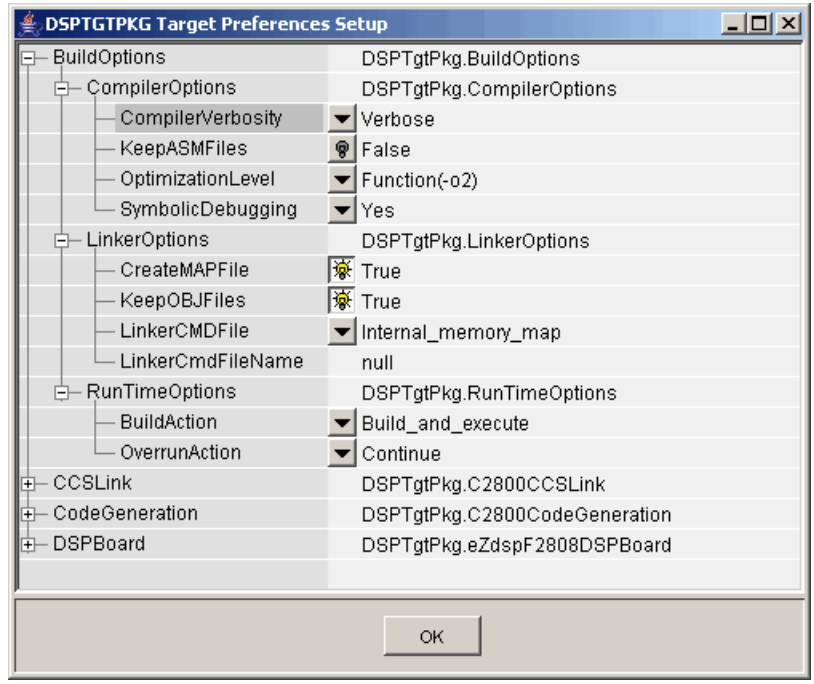


Options on the block mask let you set features of code generation for your Spectrum Digital F2808 eZdsp target. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

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 `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 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`. Note that 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.

- `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. Note that if you use `Internal_memory_map` or `Full_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. Note that when you are using flash memory, you

cannot set the **BuildAction** preference in the **BuildOptions — RunTimeOptions** section to `Build_and_execute`.

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 is the default.

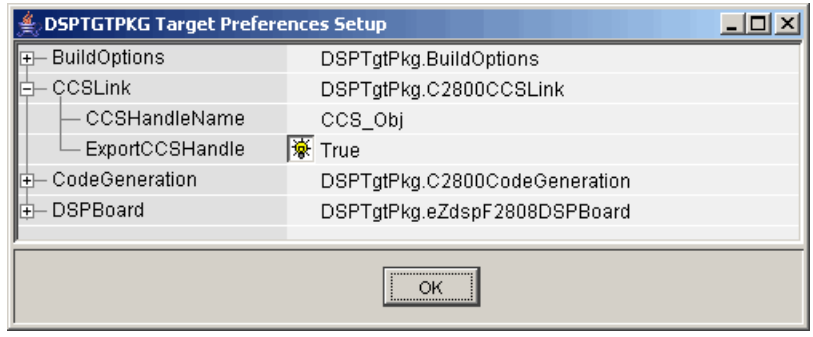
Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

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

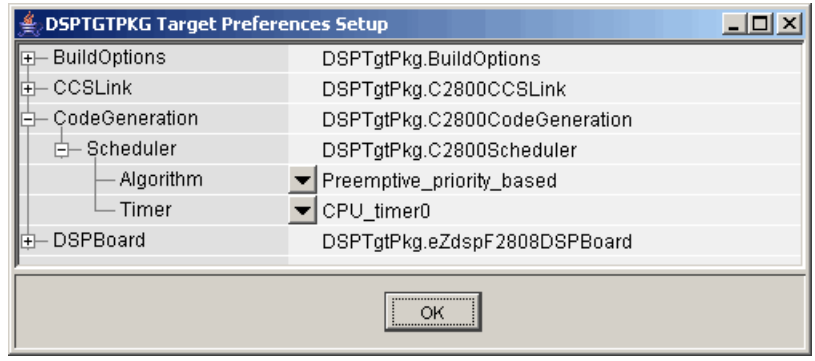
Name of the CCS handle. Click in the edit box to change the name. When you use Real-Time Workshop to build a model for a C2000 target, Embedded Target for TI C2000 DSP makes a link between MATLAB and CCS. If you have used the link portion of the Embedded Target for TI C2000 DSP, you are familiar with 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 is set to `True` (the default), then after you build your model, you will 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. The algorithm options are

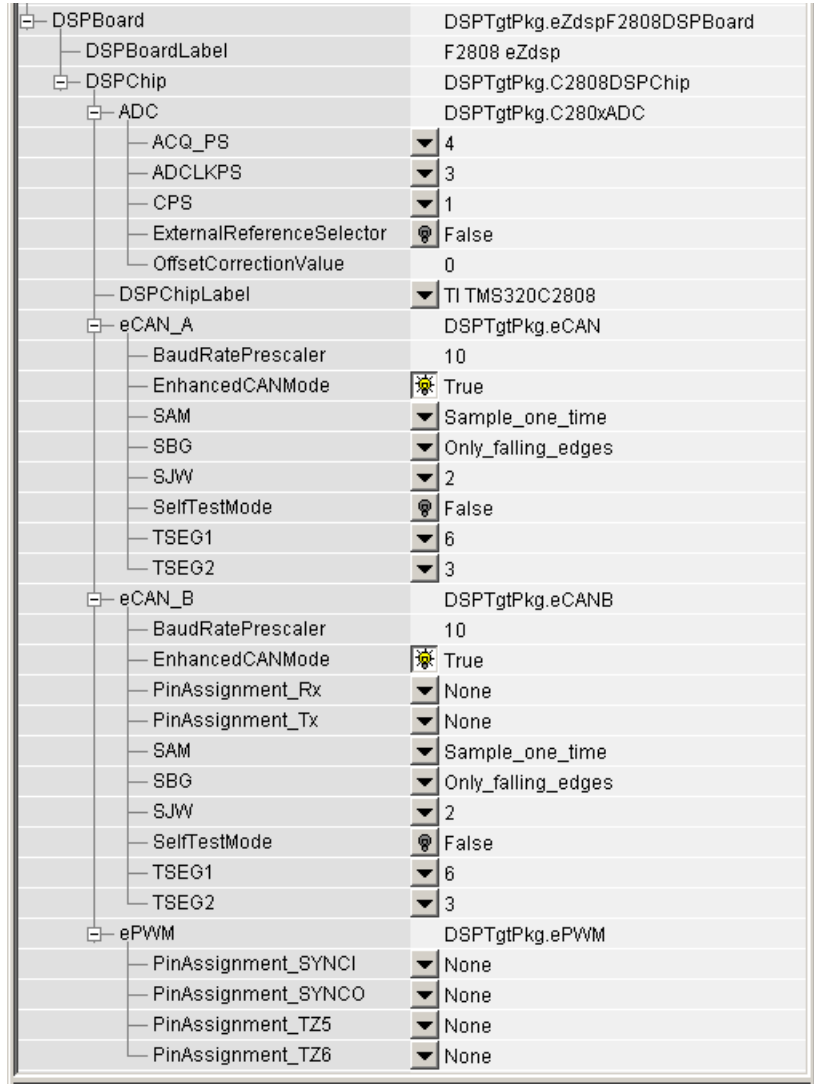
- `Preemptive_priority_based` — This scheduler runs based on the timer interrupt. The timer period is set based on the base rate sample time you specify for your model. This algorithm supports multirate systems in multitasking mode with priority-based preemption. The task for the fastest group (the base 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.

- `Free_running` — This scheduler does not use any interrupts. Tasks run in priority-based order and the execution of each task depends only on how fast the task can run on the given processor. This algorithm does not support preemption or multitasking. (Selecting `MultiTasking` as the Tasking mode in **Configuration Parameters-Solver** is not allowed for this scheduling.) Overruns do not occur with this type of scheduling, so any value in **BuildOptions — RuntimeOptions** `OverrunAction` is ignored.

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 TMS320C2808
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_SYNCI
None
PinAssignment_SYNCO
None
PinAssignment_TZ5
None
PinAssignment_TZ6
None

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

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 will be 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. The following are the settable 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/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 will be added or subtracted before the results are available in the ADC result registers. Timing for results is not affected. The default value is `0`.

eCAN_A

The settable parameters 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. **TSEG1** must be greater than **TSEG2** and the Information Processing Time (IPT). The IPT is the time needed to process one bit and corresponds to two TQ units. `TSEG1 = PROP_SEG + PHASE_SEG1`. Valid values for **TSEG1** are from `1` through `16`.

TSEG2

Sets the value of time segment 2 (PHASE_SEG2), which, with **TSEG1** and **BaudRatePrescaler**, determines the length of a bit on the eCAN bus. **TSEG2** must be less than or equal to **TSEG1** and greater than or equal to IPT. Valid values for **TSEG2** are from `1` through `8`.

eCAN_B

The settable parameters 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

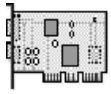
Purpose

F2812 eZdsp DSK target preferences

Library

`c2000tgtpreflib` in Embedded Target for TI C2000 DSP

Description

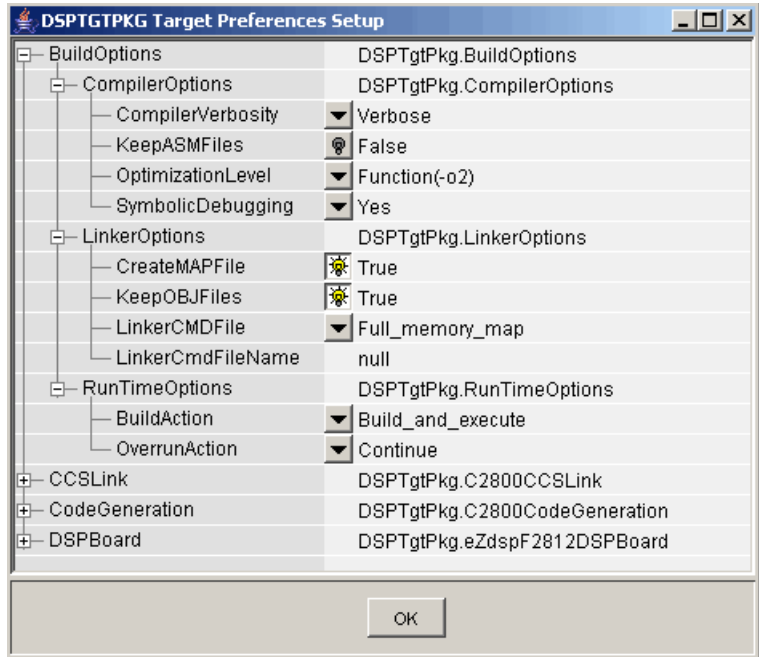


Options on the block mask let you set features of code generation for your Spectrum Digital F2812 eZdsp target. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

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.
- `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. Note that 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`. Note that 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 is the default.

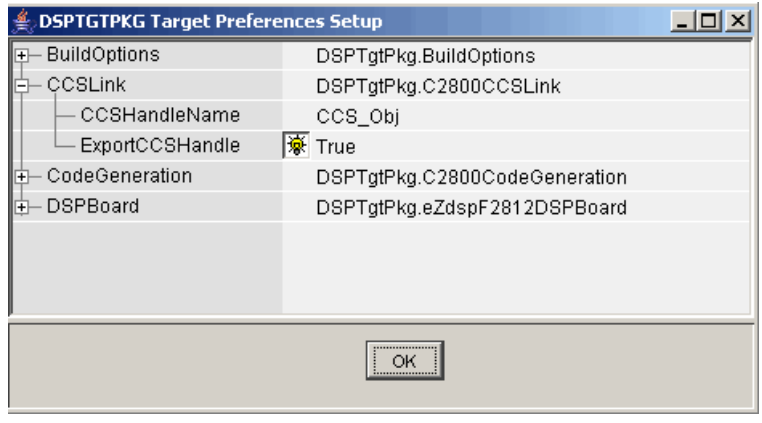
Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

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

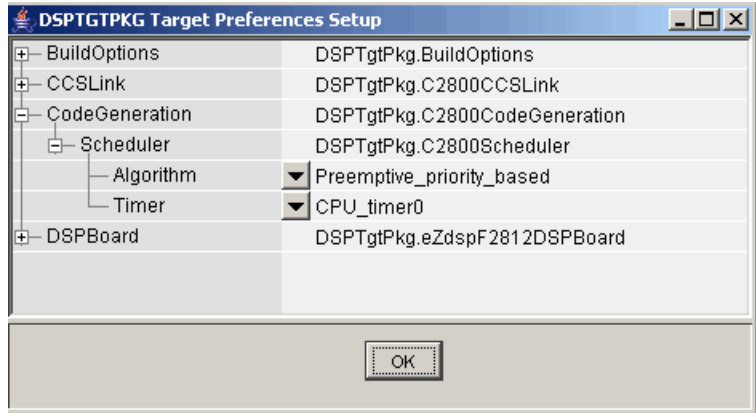
Name of the CCS handle. Click in the edit box to change the name. When you use Real-Time Workshop to build a model for a C2000 target, Embedded Target for TI C2000 DSP makes a link between MATLAB and CCS. If you have used the link portion of the Embedded Target for TI C2000 DSP, you are familiar with 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 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 `ccsdsp`.

CodeGeneration — Scheduler



Algorithm

Algorithm to use for scheduling. The algorithm options are

- `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

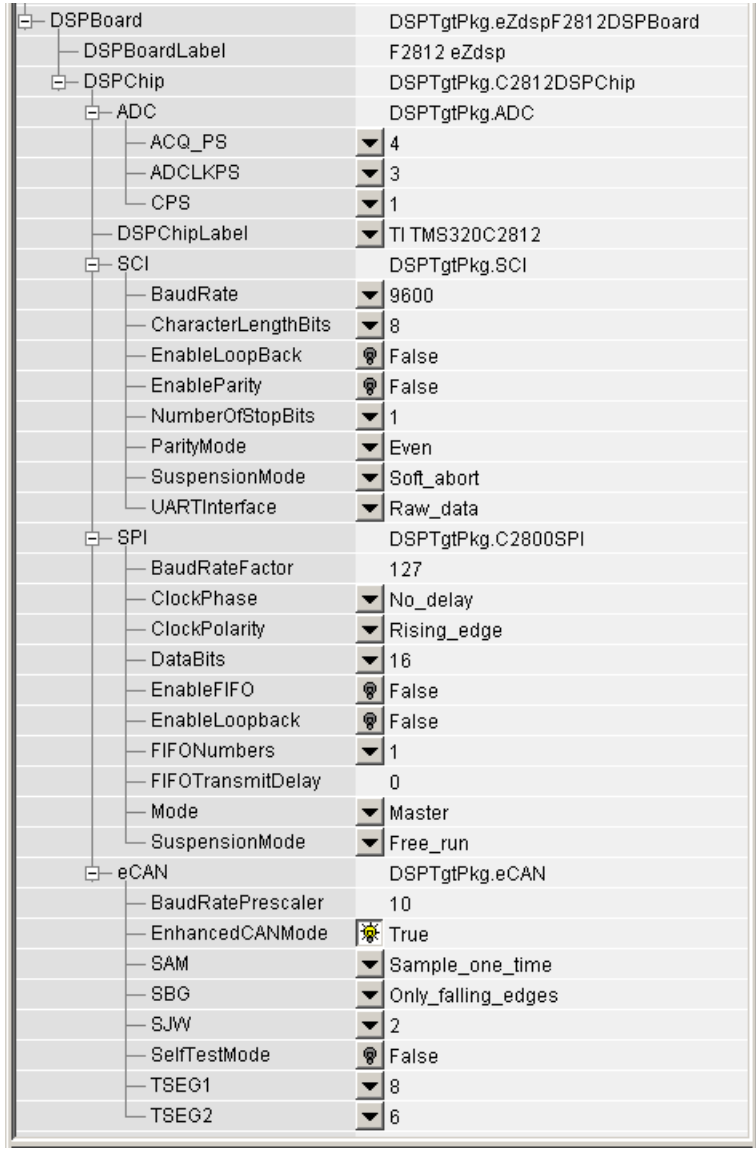
to slower tasks. For more information, see the Models with Multiple Sample Rates section of the Real Time Workshop documentation.

- `Free_running` — This scheduler does not use any interrupts. Tasks run in priority-based order and the execution of each task depends only on how fast the task can run on the given processor. This algorithm does not support preemption or multitasking. (Selecting `MultiTasking` as the Tasking mode in **Configuration Parameters-Solver** is not allowed for this scheduling.) Overruns do not occur with this type of scheduling, so any value in **BuildOptions — RuntimeOptions** `OverrunAction` is ignored.

Timer

CPU timer to use for scheduling.

DSPBoard


DSPBoard
DSPTgtPkg.eZdspF2812DSPBoard
DSPBoardLabel
F2812 eZdsp
DSPChip
DSPTgtPkg.C2812DSPChip
ADC
DSPTgtPkg.ADC
ACQ_PS
4
ADCLKPS
3
CPS
1
DSPChipLabel
TI TMS320C2812
SCI
DSPTgtPkg.SCI
BaudRate
9600
CharacterLengthBits
8
EnableLoopBack
False
EnableParity
False
NumberOfStopBits
1
ParityMode
Even
SuspensionMode
Soft_abort
UARTInterface
Raw_data
SPI
DSPTgtPkg.C2800SPI
BaudRateFactor
127
ClockPhase
No_delay
ClockPolarity
Rising_edge
DataBits
16
EnableFIFO
False
EnableLoopback
False
FIFONumbers
1
FIFOTransmitDelay
0
Mode
Master
SuspensionMode
Free_run
eCAN
DSPTgtPkg.eCAN
BaudRatePrescaler
10
EnhancedCANMode
True
SAM
Sample_one_time
SBG
Only_falling_edges
SJW
2
SelfTestMode
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 will be generated in code generation.

SCI
Parameters that affect the serial communications interfaces (SCI) on the target. The settable parameters are:

BaudRate
Baud rate for transmitting and receiving data.

CharacterLengthBits
Length in bits from `1` to `8` of each transmitted/received character.

EnableLoopBack
Select `True` to enable the loopback function for self-test and diagnostic purposes only. When this is enabled, a C28x DSP's Tx pin is internally connected to its Rx pin and it 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/receive data.

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.

UARTInterface
Protocol to use when sending or receiving UART mode data. Although available protocols are `Raw_data` and `To/From_host_block`, only `Raw_data` is supported. `Raw_data` sends or receives all data in its raw format, one byte at a time. Since the C28x SCI module has a 16-byte FIFO buffer, both the C28x SCI Receive and Transmit blocks can receive/transmit scalar or vector data.

`To/From_host_block` is not supported currently and is provided only for use in demos. It uses the serial communication interface to communicate with host-side SCI blocks. It attempts to read and interpret a specified number of elements via a `for` loop using internal protocol.

SPI

Parameters that affect the serial peripheral interfaces (SPI) on the target. The settable parameters are:

BaudRateFactor
Factor to customize the baud rate, where the CPU rate is the target's working frequency and

Baud Rate = CPU Rate / (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/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.

EnableFIFO
Select `True` to enable the FIFO buffers in the SPI module.

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 is enabled, a C28x DSP's SIMO/SOMI lines are connected internally.

FIFONumbers
Enter the number of FIFO buffers to enable. You can specify `1` to `16` buffers.

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 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/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.

eCAN Bit Timing

The eCAN protocol divides the nominal bit time into four segments, which are reflected in the settable parameters below. The four segments are

- SYNCSEG — Time used to synchronize the nodes on the bus. It is always one time quantum (TQ), which is defined as

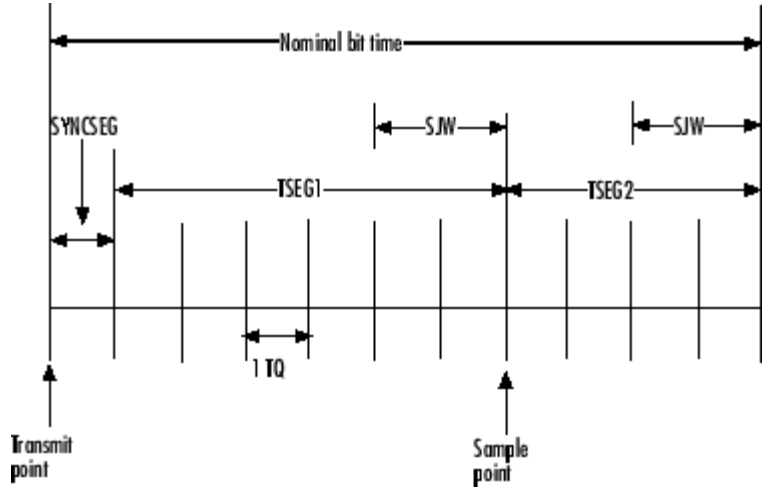
$$TQ = \frac{BaudRatePrescaler}{SYSCLK}$$

where *SYSCLK* is the CAN module system clock frequency, and the **BaudRatePrescaler** is defined below.

- PROP_SEG — Time used to compensate for physical delays in the network
- PHASE_SEG1 — Phase used to compensate for positive edge phase error

- PHASE_SEG2 — Phase used to compensate for negative edge phase error

The eCAN bit timing is shown in the following illustration.



Calculating Baud Rate

The length of a bit in the CAN module is determined by **TSEG1**, **TSEG2**, and **BaudRatePrescaler** parameters. The baud rate is

$$BaudRate = \frac{SYSCLK}{\boldsymbol{BaudRatePrescaler} \times BitTime}$$

where

"*BitTime* = *TSEG1*+*TSEG2*+1"

The following table shows the corresponding baud rates (for a 150-Mhz clock as on the F2812 DSP) for the indicated parameter settings.

TSEG1	TSEG2	BaudRate Prescaler	SJW	SBG	Baud Rate
8	6	20	2	0	0.5 Mbit/s
8	6	10	2	0	1 Mbit/s
8	6	5	2	0	2 Mbit/s

For additional details, refer to the *280x Enhanced Controller Area Network (eCAN) Reference Guide*, Literature Number SPRU074C, on the Texas Instruments Web site.

The settable eCAN parameters are:

BaudRatePrescaler
Value by which to scale the bit rate. Valid values are from 1 to 256. As noted in the equation above, this value determines the value of TQ.

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 BRP, determines the length of a bit on the eCAN bus. **TSEG1** must be greater than **TSEG2** and the Information Processing Time (IPT). The IPT is the time needed to process one bit and corresponds to two TQ units. `TSEG1 = PROP_SEG + PHASE_SEG1`. Valid values for **TSEG1** are from `1` through `16`.

TSEG2

Sets the value of time segment 2 (PHASE_SEG2), which, with **TSEG1** and BRP, determines the length of a bit on the eCAN bus. TSEG2 must be less than or equal to **TSEG1** and greater than or equal to IPT. Valid values for **TSEG2** are from `1` through `8`.

See Also

C281x ADC, C281x eCAN Receive, C281x eCAN Transmit, C281x PWM

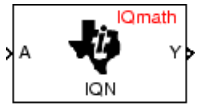
Purpose

Convert floating-point number to IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

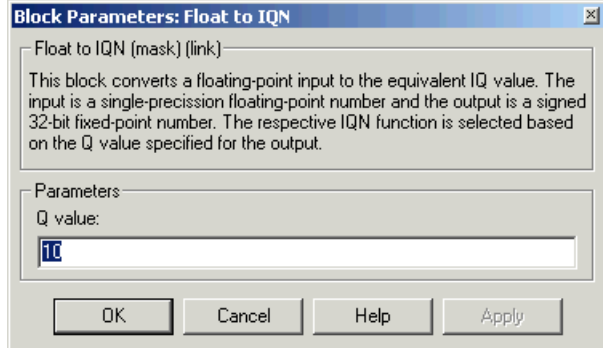


Float to IQN

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 2-2 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

Fractional part IQN

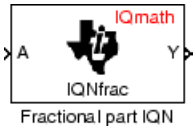
Purpose

Fractional part of IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

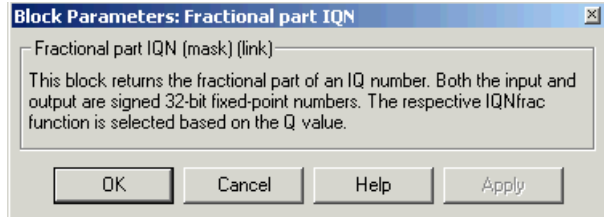


Fractional part IQN

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 2-2 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

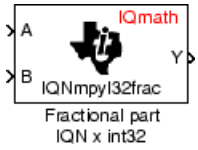
Purpose

Fractional part of result of multiplying IQ number and long integer

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

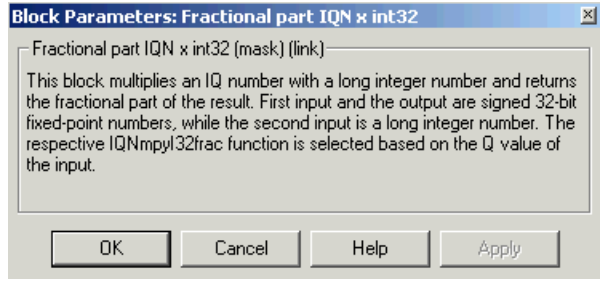


Fractional part
IQN x int32

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 2-2 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

From Memory

Purpose

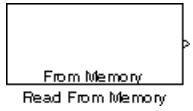
Retrieve data from target memory

Library

`c2400spchiplib` or `c280xspchiplib` or `c281xspchiplib` in Embedded Target for TI C2000 DSP

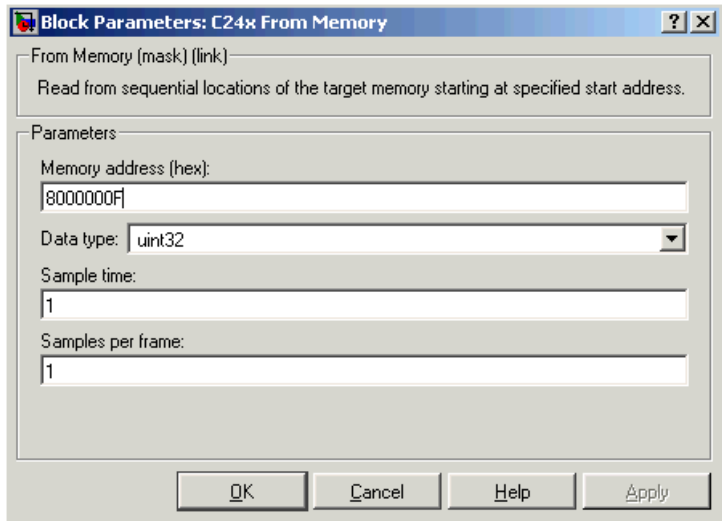
Description

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.

Note To ensure the correct operation of this block, you must specify exactly the desired memory location. Refer to your Linker CMD file for available memory locations.

Data type
Data type of the data to obtain from the above memory address. The data is read as 16-bit data and then cast to the selected data type. Valid data types are `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`.

Sample time
Time interval, in seconds, between consecutive reads from the specified memory location.

Samples per frame
Number of elements of the specified data type to be read from the memory region starting at the given address.

See Also

To Memory

From RTDX

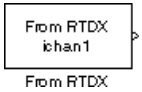
Purpose

Add RTDX 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)
```

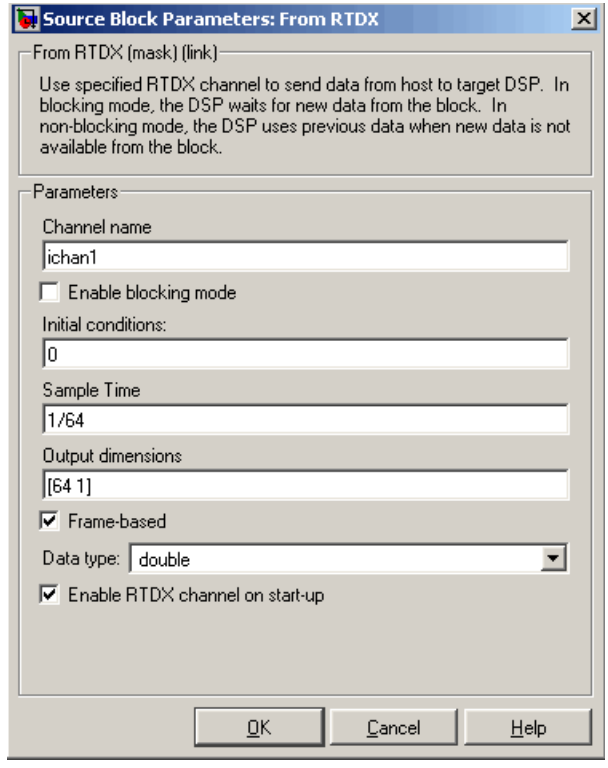
where `channelname` is the name you enter in **Channel name.**

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

`ccsdsp`, `readmsg`, To RTDX, `writemsg`.

Idle Task

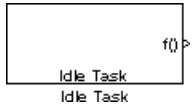
Purpose

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 subsytem 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

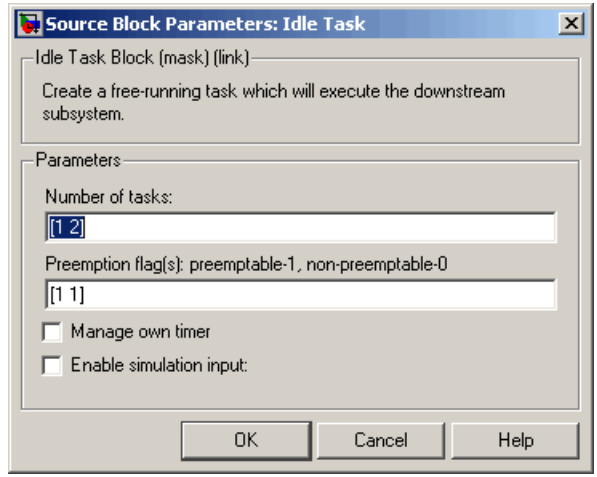
The output of this block includes a set of two vectors, the **Number of tasks** and the corresponding **Preemption flag(s)**. The **Preemption flag(s)** vector must be the same length as the **Number of tasks** vector unless it has only one element.

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.

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 exectuted 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 downstreasm 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 values 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

Purpose

Integer part of IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

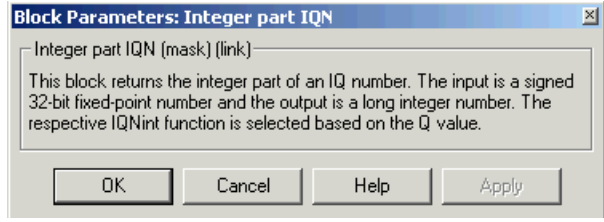
Description

Integer part IQN

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 2-2 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

Integer part IQN x int32

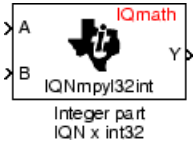
Purpose

Integer part of result of multiplying IQ number and long integer

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

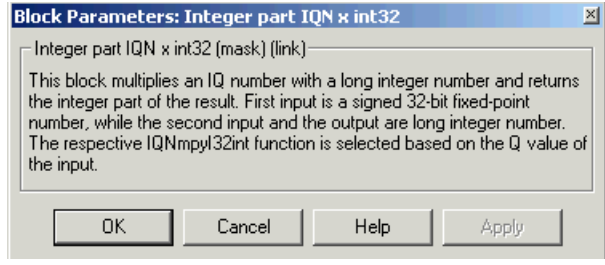


Integer part
IQN x int32

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 2-2 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

Purpose

Convert rotating reference frame vectors to two-phase stationary reference frame

Library

`c28xdmclib` in Embedded Target for TI C2000 DSP

Description

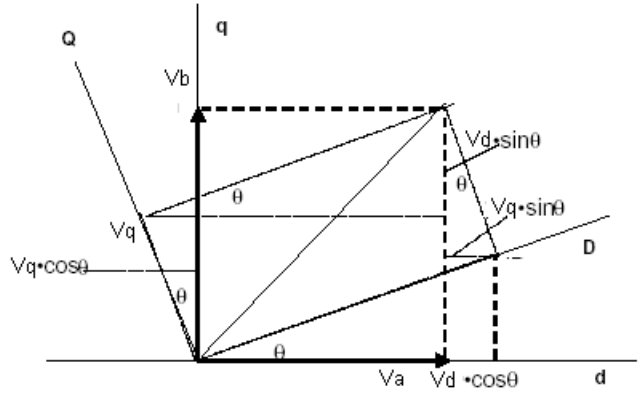
Inverse Park Transformation

This block converts vectors in an orthogonal rotating reference frame to a two-phase orthogonal stationary reference frame. The transformation implements these equations

$$Va = Vd\cos\theta - Vq\sin\theta$$

$$Vb = Vd\sin\theta + Vq\cos\theta$$

and is illustrated in the following figure.

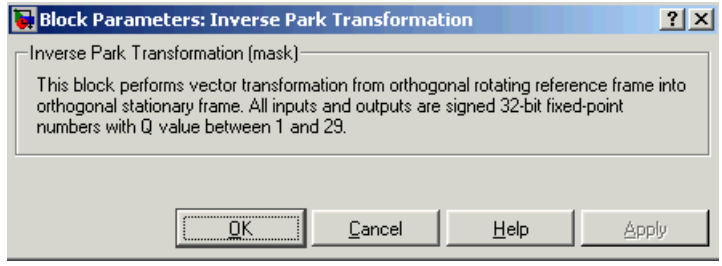


The inputs to this block are the direct axis (`Vd`) and quadrature axis (`Vq`) components of the transformed signal in the rotating frame and the phase angle (`theta`) between the stationary and rotating frames.

The outputs are the direct axis (`Va`) and the quadrature axis (`Vb`) components of the transformed signal.

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 2-2 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

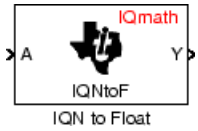
Purpose

Convert IQ number to floating-point number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

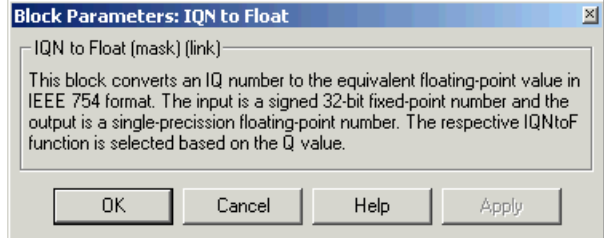


IQN to Float

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 2-2 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

IQN x int32

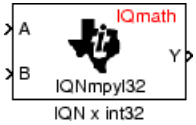
Purpose

Multiply IQ number with long integer

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

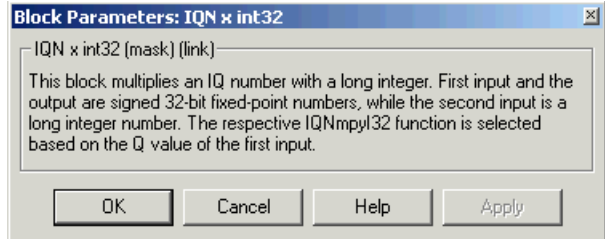


IQN x int32

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 2-2 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

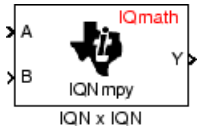
Purpose

Multiply two IQ numbers with same Q format

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

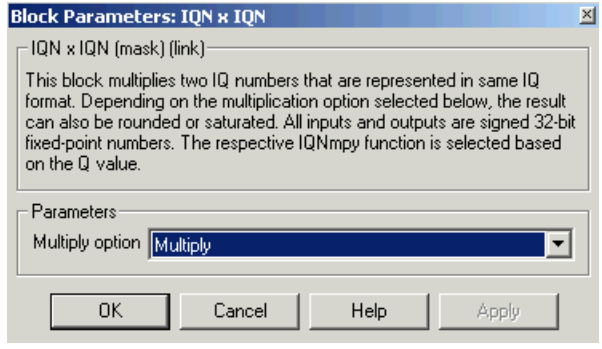


IQN x IQN

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 2-2 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

Purpose

Convert IQ number to different Q format

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

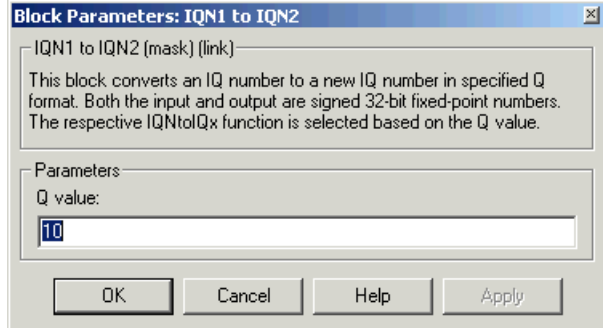
Description

IQN1 to IQN2

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 2-2 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

IQN1 x IQN2

Purpose

Multiply two IQ numbers with different Q formats

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

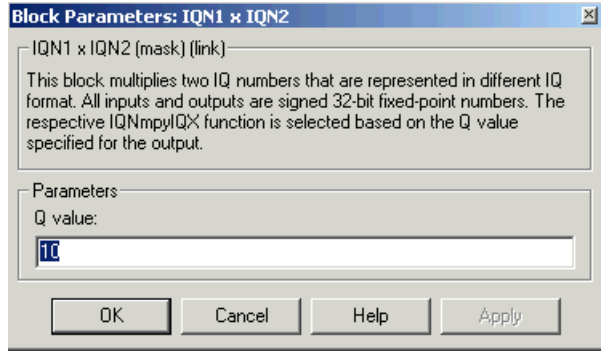
Description

IQN1 x IQN2

This block multiples 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 2-2 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

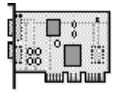
Purpose

LF2407 eZdsp DSK target preferences

Library

`c2000tgtpreflib` in Embedded Target for TI C2000 DSP

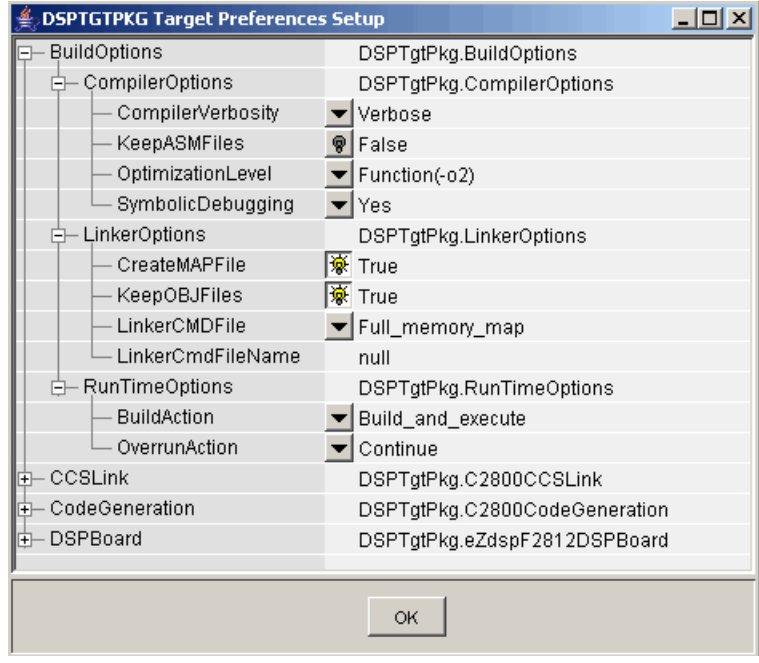
Description



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 will fit in the internal chip memory can be used. If your program is too large, a linker error will occur. 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. Note that 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`. Note that 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 is the default.

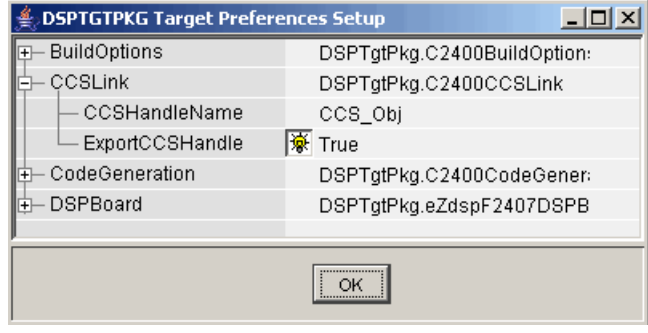
Note When you build and execute a model on your target, the Real-Time Workshop build process resets the target automatically. You do not need to reset the board before building models.

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



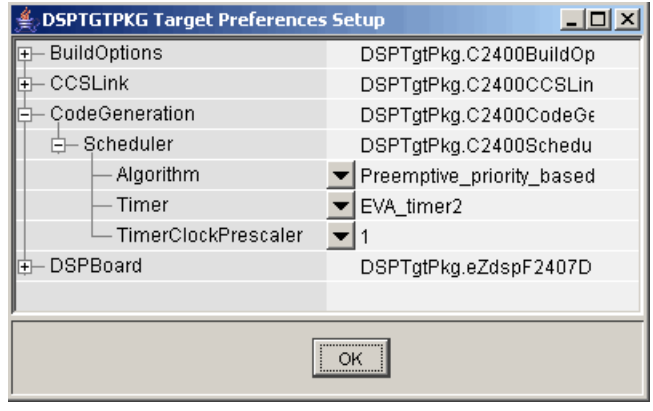
CCSHandleName

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 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 `ccsdsp`.

CodeGeneration — Scheduler



Algorithm

Algorithm to use for scheduling. The algorithm options are

- `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 the Models with Multiple Sample Rates section of the Real Time Workshop documentation.
- `Free_running` — This scheduler does not use any interrupts. Tasks run in priority-based order and the execution of each task depends only on how fast the task can run on the given processor. This algorithm does not support preemption or multitasking. (Selecting `MultiTasking` as the Tasking mode in **Configuration Parameters-Solver** is not allowed for this scheduling.) Overruns do not occur with this type of scheduling, so any value in **BuildOptions – RuntimeOptions** – `OverrunAction` is ignored.

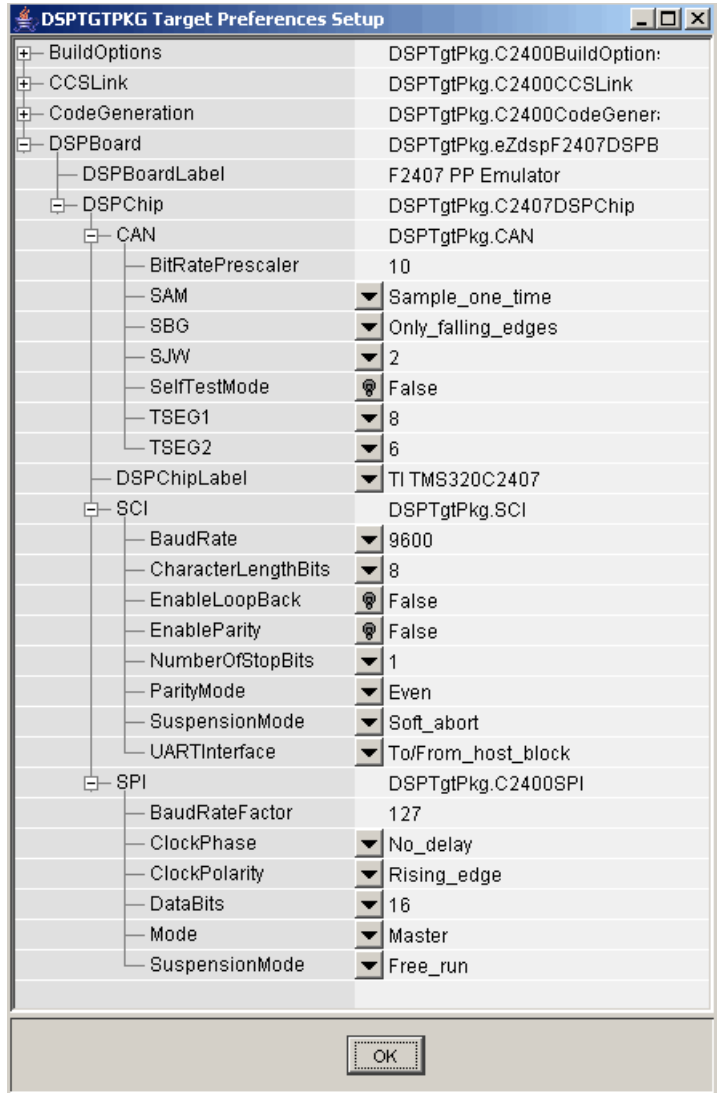
Timer

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


DSPTGTPKG Target Preferences Setup
BuildOptions
DSPTgtPkg.C2400BuildOption:
CCSLink
DSPTgtPkg.C2400CCSLink
CodeGeneration
DSPTgtPkg.C2400CodeGener:
DSPBoard
DSPTgtPkg.eZdspF2407DSPB
DSPBoardLabel
F2407 PP Emulator
DSPChip
DSPTgtPkg.C2407DSPChip
CAN
DSPTgtPkg.CAN
BitRatePrescaler
10
SAM
Sample_one_time
SBG
Only_falling_edges
SJW
2
SelfTestMode
False
TSEG1
8
TSEG2
6
DSPChipLabel
TI TMS320C2407
SCI
DSPTgtPkg.SCI
BaudRate
9600
CharacterLengthBits
8
EnableLoopBack
False
EnableParity
False
NumberOfStopBits
1
ParityMode
Even
SuspensionMode
Soft_abort
UARTInterface
To/From_host_block
SPI
DSPTgtPkg.C2400SPI
BaudRateFactor
127
ClockPhase
No_delay
ClockPolarity
Rising_edge
DataBits
16
Mode
Master
SuspensionMode
Free_run
OK

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.

CAN Bit Timing

The CAN protocol divides the nominal bit time into four segments, which are reflected in the settable parameters below. The four segments are

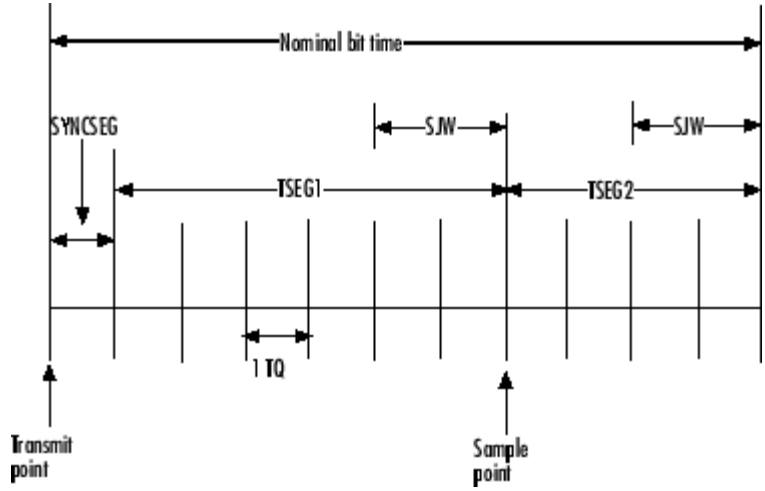
- SYNCSEG — Time used to synchronize the nodes on the bus. It is always one time quantum (TQ), which is defined as

$$TQ = \frac{BaudRatePrescaler}{SYSCLK}$$

where *SYSCLK* is the CAN module system clock frequency, and the **BaudRatePrescaler** is defined below.

- PROP_SEG — Time used to compensate for the physical delays in the network
- PHASE_SEG1 — Phase used to compensate for positive edge phase error
- PHASE_SEG2 — Phase used to compensate for negative edge phase error

The CAN bit timing is shown in the following illustration.



Calculating Baud Rate
The length of a bit in the CAN module is determined by **TSEG1**, **TSEG2**, and **BaudRatePrescaler** parameters. The baud rate is

$$BaudRate = \frac{SYSCLK}{\boldsymbol{BaudRatePrescaler} \times BitTime}$$

where

BitTime = **TSEG1**+**TSEG2**+1

The following table shows the corresponding baud rates (for a 40-Mhz clock as on the F2407 DSP) for the indicated parameter settings.

TSEG1	TSEG2	BaudRate Prescaler	SJW	SBG	Baud Rate
4	3	10	2	0	0.5 Mbit/s
5	4	4	2	0	1 Mbit/s
6	3	2	2	0	2 Mbit/s

For additional details, refer to the *TMS320LF/LC240xA DSP Controllers Reference Guide - Systems and Peripherals*, Literature Number SPRU357B, on the Texas Instruments Web site.

The settable CAN parameters are:

BaudRatePrescaler
Value by which to scale the baud rate. Valid values are from 1 to 256. As noted in the equation above, this value determines the value of TQ.

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 CAN module 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. TSEG1 must be greater than TSEG2 and the Information Processing Time (IPT). The IPT is the time needed to process one bit and corresponds to two TQ units. TSEG1 = PROP_SEG + PHASE_SEG1. See above for definitions of PROP_SEG and PHASE_SEG1. Valid values for TSEG1 are from 1 through 16.

TSEG2

Sets the value of time segment 2 (PHASE_SEG2), which, with TSEG1 and BRP, determines the length of a bit on the CAN bus. See above for definitions of PHASE_SEG2. TSEG2 must be less than or equal to TSEG1 and greater than or equal to IPT. 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 will be generated in code generation.

SCI

Parameters that affect the serial communications interfaces (SCI) on the target.

The settable parameters are:

BaudRate

Baud rate for transmitting and receiving data.

CharacterLengthBits
Length in bits from `1` to `8` of each transmitted/received character.

EnableLoopBack
Select `True` to enable the loopback function for self-test and diagnostic purposes only. When this is enabled, a C24x DSP's Tx pin is internally connected to its Rx pin and it 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/receive data.

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.

UARTInterface
Protocol to use when sending or receiving UART mode data. Although available protocols are `Raw_data` and `To/From_host_block`, only `Raw_data` is supported. `Raw_data` sends or receives all data in its raw format, one byte at a time.

`To/From_host_block` is not supported currently and is provided only for use in demos. It uses the serial communication interface to communicate with host-side SCI blocks. It attempts to read and interpret a specified number of elements via a `for` loop using internal protocol.

SPI

Parameters that affect the serial peripheral interfaces (SPI) on the target.

The settable parameters are:

BaudRateFactor

Factor to customize the baud rate, where the CPU rate is the target's working frequency and

$$Baud\ Rate = CPU\ Rate\ /\ (Baud\ Rate\ Factor + 1)$$

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/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. Note that 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/transmit sequence is complete. `Free run` continues running regardless of the breakpoint.

See Also

C24x ADC, C24x CAN Receive, C24x CAN Transmit, C24x PWM

Magnitude IQN

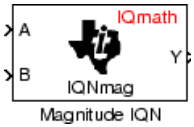
Purpose

Magnitude of two orthogonal IQ numbers

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description



Magnitude IQN

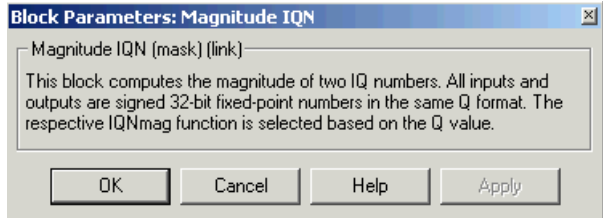
This block calculates the magnitude of two IQ numbers using

$$\sqrt{a^2 + b^2}$$

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 2-2 for more information.

Dialog Box



See Also

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

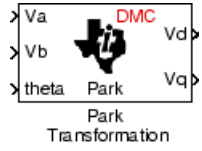
Purpose

Convert two-phase stationary system vectors to rotating system vectors

Library

`c28xdmclib` in Embedded Target for TI C2000 DSP

Description

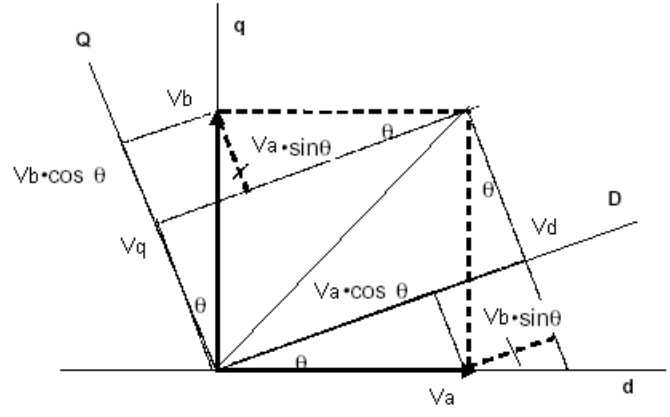


This block converts vectors in balanced two-phase orthogonal stationary systems into an orthogonal rotating reference frame. The transformation implements these equations

$$Vd = Va\cos\theta + Vb\sin\theta$$

$$Vq = -Va\sin\theta + Vb\cos\theta$$

and is illustrated in the following figure.



The inputs to this block are the direct axis (`Va`) and the quadrature axis (`Vb`) components of the transformed signal and the phase angle (`theta`) between the stationary and rotating frames.

The outputs are the direct axis (`Vd`) and quadrature axis (`Vq`) components of the transformed signal in the rotating frame.

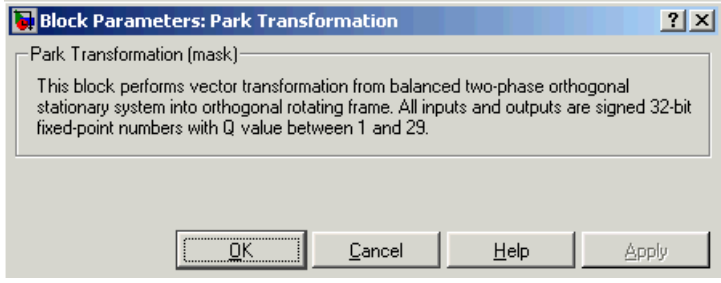
The instantaneous inputs are defined by the following equations.

“*id = I sin(wt)*

iq = I sin(wt + π/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 2-2 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, PID Controller, Space Vector Generator, Speed Measurement

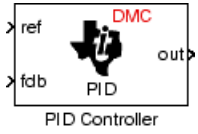
Purpose

Digital PID controller

Library

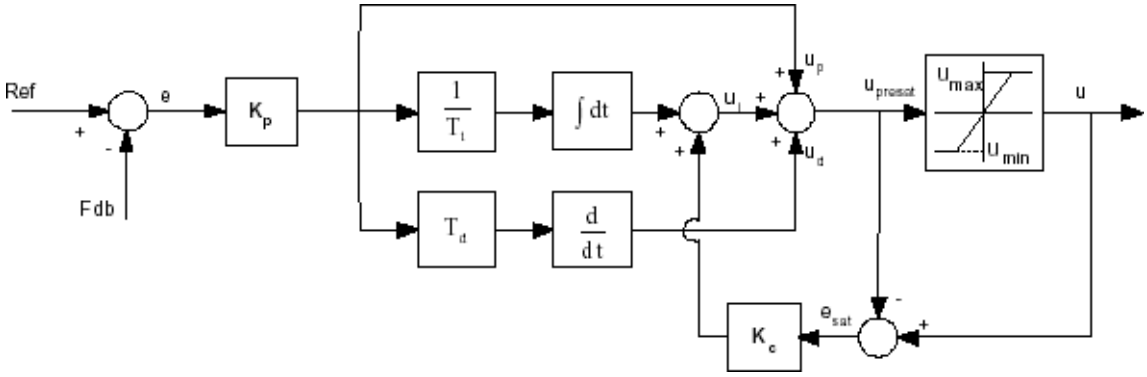
`c28xdmclib` in Embedded Target for TI C2000 DSP

Description



PID Controller

This block implements a 32-bit digital PID controller with antiwindup correction. The inputs are a reference input (`ref`) and a feedback input (`fdb`) and the output (`out`) is the saturated PID output. The following diagram shows a PID controller with antiwindup.



The differential equation describing the PID controller before saturation that is implemented in this block is

"$u_{presat}(t) = u_p(t) + u_i(t) + u_d(t)$"

where u_{presat} is the PID output before saturation, u_p is the proportional term, u_i is the integral term with saturation correction, and u_d is the derivative term.

The proportional term is

"$u_p(t) = K_p e(t)$"

where K_p is the proportional gain of the PID controller and $e(t)$ is the error between the reference and feedback inputs.

The integral term with saturation correction is

$$u_i(t) = \frac{K_p}{T_i}\int_0^t e(\varsigma)d\varsigma + K_c(u(t)-u_{presat}(t))$$

where K_c is the integral correction gain of the PID controller.

The derivative term is

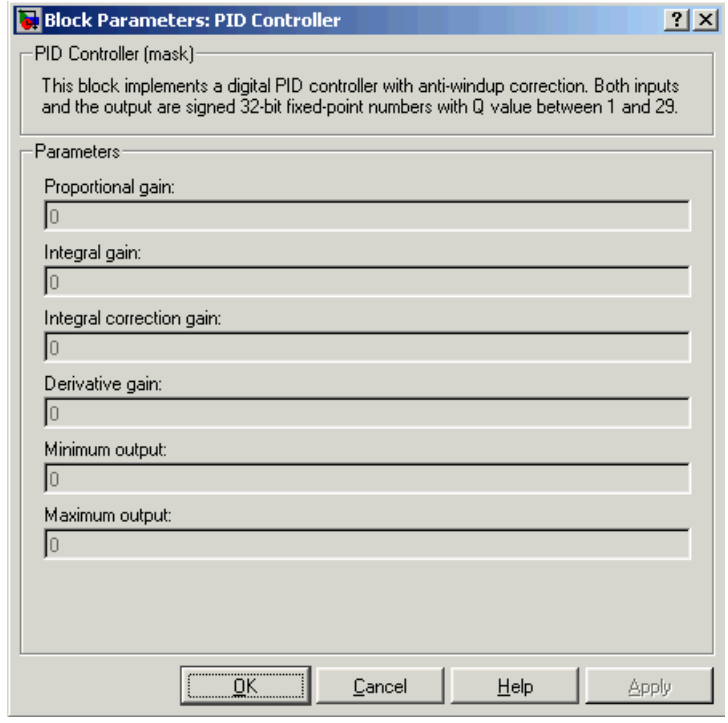
$$u_d(t) = K_p T_d \frac{de(t)}{dt}$$

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 2-2 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

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

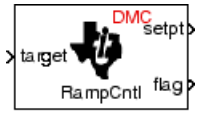
Purpose

Create a ramp-up and ramp-down function

Library

`c28xdmclib` in Embedded Target for TI C2000 DSP

Description



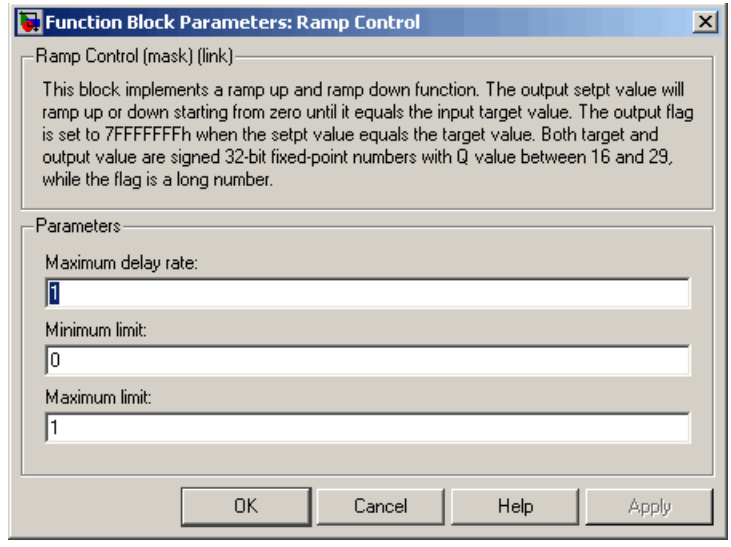
Ramp Control

This block implements a ramp-up and ramp-down function. The input is a `target` value and the outputs are the set point value (`setpt`) and a `flag`. The `flag` output is set to `7FFFFFFFh` when the output `setpt` value reaches the input `target` value. The `target` and `setpt` values are signed 32-bit fixed-point numbers with Q values between 16 and 29. The flag is a long number.

The `target` value is compared with the `setpt` value. If they are not equal, the output `setpt` is adjusted up or down by a fixed step size (0.0000305).

If the fixed step size is relatively large compared to the `target` value, the output may oscillate around the `target` value.

Dialog Box



Maximum delay rate

Value that is multiplied by the sampling loop time period to determine the time delay for each ramp step. Valid values are integers greater than 0.

Minimum limit

Minimum allowable ramp value. If the input falls below this value, it will be saturated to this minimum. The smallest value you can enter is the minimum value that can be represented in fixed-point data format by the input and output blocks to which this Ramp Control block is connected in your model. If you enter a value below this minimum, an error occurs at the start of code generation or simulation. For example, if your input is in Q29 format, its minimum value is -4.

Maximum limit

Maximum allowable ramp value. If the input goes above this value, it will be reduced to this maximum. The largest value you can enter is the maximum value that can be represented in fixed-point data format by the input and output blocks to which this Ramp Control block is connected in your model. If you enter a value above this maximum, an error occurs at the start of code generation or simulation. For example, if your input is in Q29 format, its maximum value is 3.9999....

See Also

Ramp Generator

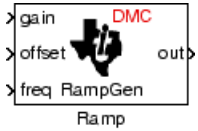
Purpose

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:

"`out`(k) = angle(k) * `gain`(k) + `offset`(k) "

For `out`(k) > 1, `out`(k) = `out`(k) - 1. For `out`(k) < -1, `out`(k) = `out`(k) + 1.

Angle(k) is defined as follows:

"angle(k) = angle(k-1) + `freq`(k) * **Maximum step angle**

for angle(k) > 1, angle(k) = angle(k) - 1

for angle(k) < -1, angle(k) = angle(k) + 1"

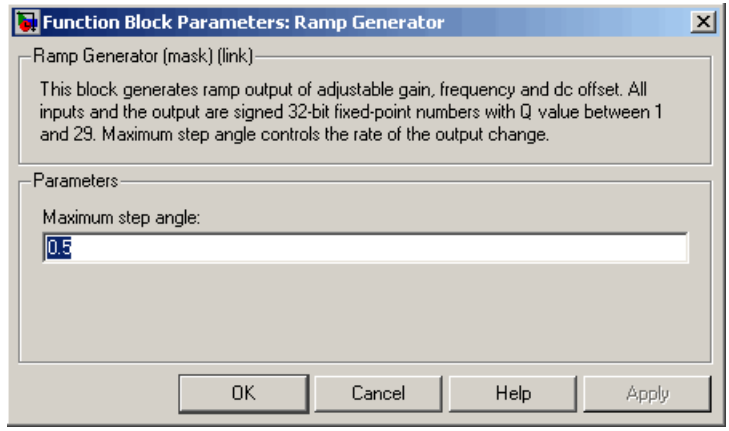
The frequency of the ramp output is controlled by a precision frequency generation algorithm that relies on the modulo nature of the finite length variables. The frequency of the output ramp signal is equal to

"f = (**Maximum step angle** * sampling rate) / 2^m "

where m represents the fractional length of the data type of the inputs.

All math operations are carried out in fixed-point arithmetic, where the fixed-point fractional length is determined by the block's inputs.

Dialog Box

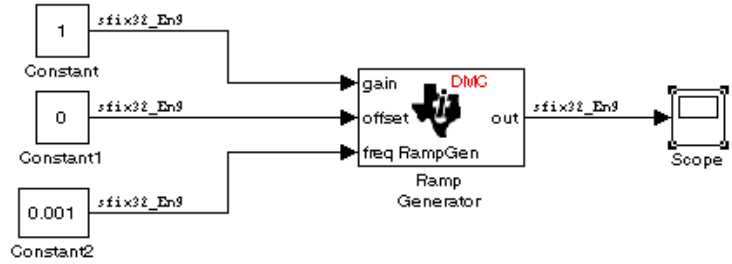


Maximum step angle

The maximum step size, which determines the rate of change of the output (i.e., the minimum period of the ramp signal).

Examples

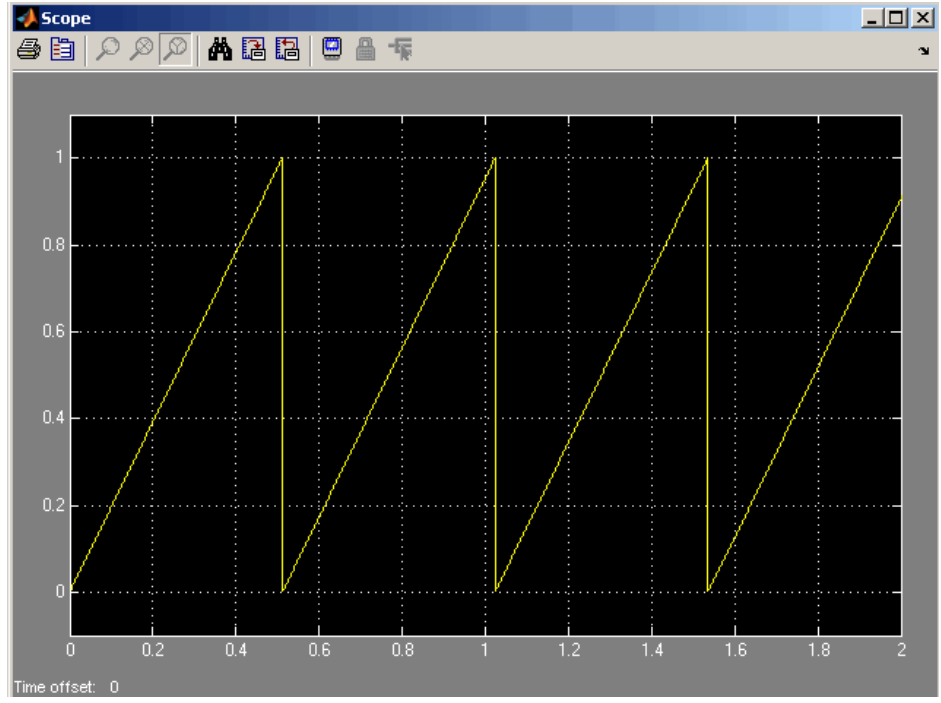
The following model demonstrates the Ramp Generator block. The Constant and Scope blocks are available in Simulink Commonly Used Blocks.



In your model, select **Simulation > Configuration Parameters**. On the **Solver** pane, set **Type** to `Fixed-step` and **Solver** to `discrete (no continuous states)`. Set the parameter values for the blocks as shown in the following table.

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).



The expected frequency of the output is

"f = (maximum step angle * sampling rate) / 2^m

f = (1 * 1000) / 2^9 = 1.9531 Hz "

The expected period is then

"$T = 1/f$ = 0.5120 s "

which is what the above Scope output shows.

See Also

Ramp Control

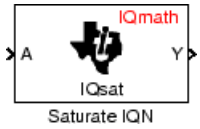
Purpose

Saturate an IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

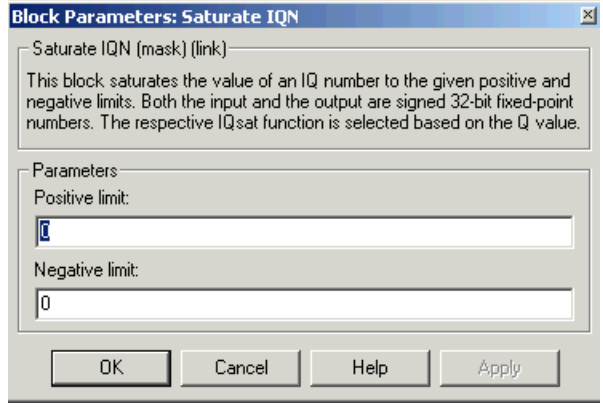


Saturate IQN

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 2-2 for more information.

Dialog Box



Positive Limit

Maximum positive value to which to saturate

Negative Limit

Minimum negative value to which to saturate

See Also

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

Purpose

Duty ratios for stator reference voltage

Library

`c28xdmclib` in Embedded Target for TI C2000 DSP

Description

Space Vector Generator

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.

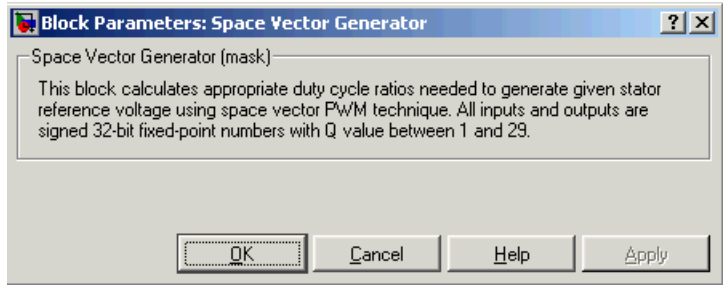
The inputs to this block are

- Alpha component — the reference stator voltage vector on the direct axis stationary reference frame (`Ua`)
- Beta component — the reference stator voltage vector on the direct axis quadrature reference frame (`Ub`)

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 (`Ta`), PWM3 (`Tb`), and PWM5 (`Tc`).

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 2-2 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

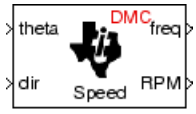
Purpose

Motor speed

Library

`c28xdmclib` in Embedded Target for TI C2000 DSP

Description

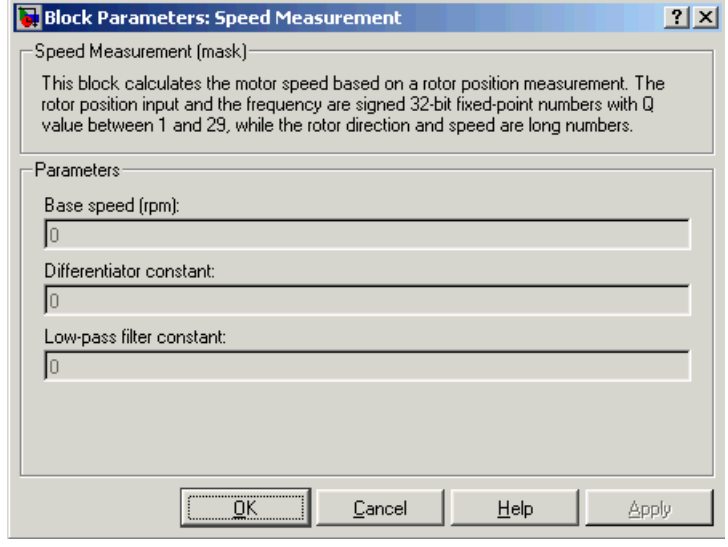


Speed Measurement

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 2-2 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

Purpose

Square root or inverse square root of IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

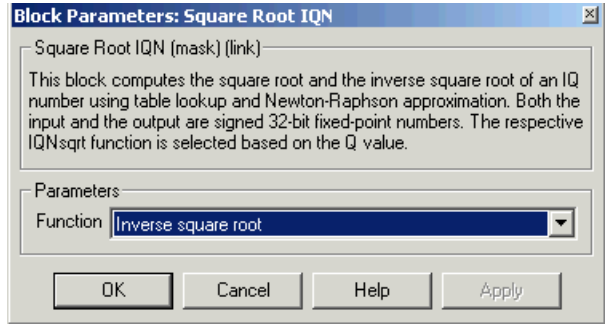
Square Root IQN

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 2-2 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

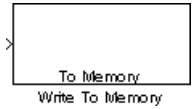
Purpose

Write data to target memory

Library

`c2400spchiplib` or `c280xspchiplib` or `c281xspchiplib` in Embedded Target for TI C2000 DSP

Description

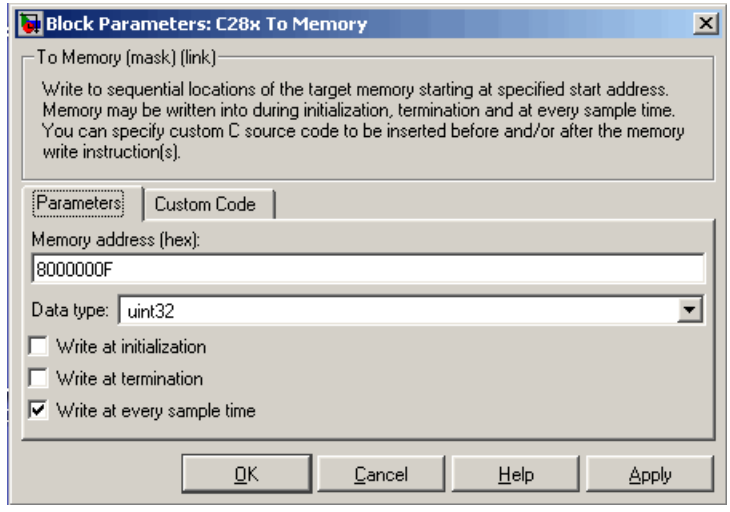


This block sends data of the specified data type to a particular memory address on the target.

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

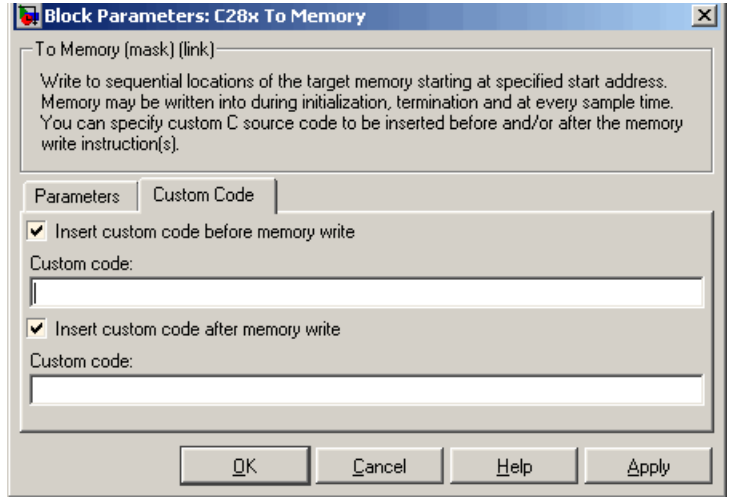
Value
Last value of data to be written to memory at program termination

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

To RTDX

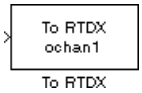
Purpose

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)
```

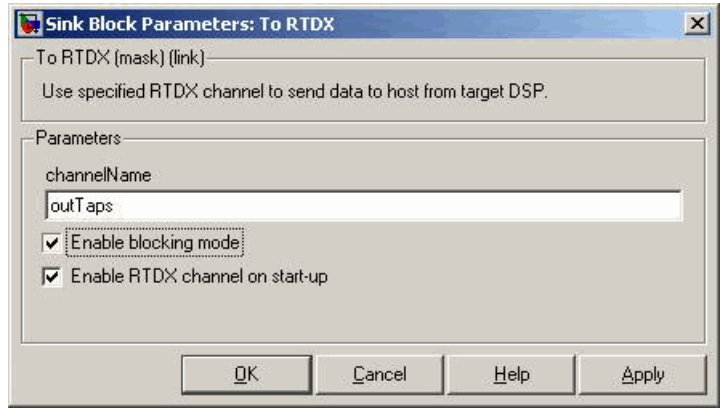
where `channelname` is the name you enter in the **channelName** field in the **To RTDX** dialog box.

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 temporary divert the program execution from this site, but it will eventually come back and wait until the channel stops writing.

When blocking mode is not enabled (when the check box is cleared), writing a message is abandoned if the RTDX channel is busy, and the code proceeds with the current iteration.

Enable RTDX channel on start-up

Enables the RTDX channel when you start the channel from MATLAB. With this selected, you do not need to use the enable function in 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

Purpose

Sine, cosine, or arc tangent of IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

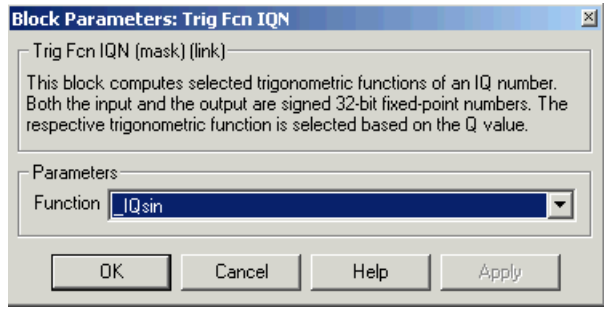
Description

Trig Fcn IQN

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 2-2 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

Index

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

W