

Embedded Target for the TI TMS320C2000™ DSP Platform For Use with Real-Time Workshop®

- Modeling
- Simulation
- Implementation

How to Contact The MathWorks:



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup



support@mathworks.com 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. Mail
3 Apple Hill Drive
Natick, MA 01760-2098

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

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

© COPYRIGHT 2003 - 2004 by The MathWorks, Inc.

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

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

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.

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

Printing History: November 2003 Online only New for Version 1.0 (Release 13SP1+)
June 2004 Online only Revised for Version 1.1 (Release 14)

Getting Started

1

| | |
|---|------|
| What Is the Embedded Target for the TI TMS320C2000 DSP Platform? | 1-2 |
| Suitable Applications | 1-2 |
| Setting Up and Configuring | 1-3 |
| Platform Requirements — Hardware and Operating System .. | 1-3 |
| Supported Hardware for Targets | 1-3 |
| Software Requirements | 1-4 |
| Verifying the Configuration | 1-5 |
| Embedded Target for TI C2000 and Code Composer Studio | 1-7 |
| Default Project Configuration | 1-7 |
| Scheduling and Timing | 1-8 |
| Overview of Creating Models for Targeting | 1-10 |
| Online Help | 1-10 |
| Notes About Selecting Blocks for Your Models | 1-11 |
| Setting Simulation Configuration Parameters | 1-12 |
| Building Your Model | 1-12 |
| Using the c2000lib Blockset | 1-14 |
| Hardware Setup | 1-14 |
| Starting the c2000lib Library | 1-14 |
| Setting Up the Model | 1-16 |
| Adding Blocks to the Model | 1-23 |
| Generating Code from the Model | 1-27 |
| Creating Code Composer Studio Projects Without Loading .. | 1-27 |

Using the IQmath Library

2

| | |
|--|------------|
| About the IQmath Library | 2-2 |
| Common Characteristics | 2-2 |
| Fixed-Point Numbers | 2-3 |
| Signed Fixed-Point Numbers | 2-3 |
| Q Format Notation | 2-4 |
| Building Models | 2-7 |
| Converting Data Types | 2-7 |
| Using Sources and Sinks | 2-7 |
| Choosing Blocks to Optimize Code | 2-7 |

Block Reference

3

| | |
|---|------------|
| Blocks — Categorical List | 3-2 |
| C2400 DSP Chip Support Library (c2400dspchilib) | 3-2 |
| C2800 DSP Chip Support Library (c2800dspchilib) | 3-2 |
| Target Preferences Library (c2000tgtpreflib) | 3-3 |
| C28x Digital Motor Control Library (c28xdmclib) | 3-3 |
| C28x IQmath Library (tiiqmathlib) | 3-4 |
| Blocks — Alphabetical List | 3-5 |

Index

Getting Started

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

| | |
|---|--|
| What Is the Embedded Target for the TI TMS320C2000 DSP Platform? (p. 1-2) | Introduces the Embedded Target for TI C2000 DSP and describes some of its features and supported hardware |
| Setting Up and Configuring (p. 1-3) | Describes the software and hardware required to use the Embedded Target for the TI TMS320C2000 DSP Platform and how to set them up |
| Embedded Target for TI C2000 and Code Composer Studio (p. 1-7) | Information about Code Composer Studio |
| Scheduling and Timing (p. 1-8) | Information about C2000 scheduling |
| Overview of Creating Models for Targeting (p. 1-10) | Summary of steps required to create models for your target |
| Using the c2000lib Blockset (p. 1-14) | Example of creating a model and targeting hardware |

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

The Embedded Target for the TI TMS320C2000™ DSP Platform integrates Simulink® and MATLAB® with Texas Instruments eXpressDSP™ tools. You can use this product to develop and validate digital signal processing and control designs from concept through code. The Embedded Target for the TI TMS320C2000 DSP Platform uses C code generated by Real-Time Workshop® and your TI development tools to generate a C language real-time implementation of your Simulink model. Real-Time Workshop builds a Code Composer Studio® project from the C code. You can compile, link, download, and execute the generated code on an eZdsp™ DSP board from Spectrum Digital.

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

- 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:

- 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 NT 4.0 Server or Workstation, 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:

- TMS320F2812 eZdsp DSK — the F2812eZdsp DSP Starter Kit
- TMS320LF2407 eZdsp DSK — the LF2407eZdsp DSP Starter Kit

The above DSP Starter Kits (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 You do not need to change any jumpers from their factory defaults on either the LF2407 or F2812 target board.

Software Requirements

MathWorks Software

For up-to-date information about other MathWorks software you need 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 2.12 or 2.2 for the C28x 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 byproduct of the compilation process, you get assembly code (.asm) as well. |

Required TI Software for Targeting Your TI C2000 Hardware (Continued)

| Installed Product | Additional Information |
|----------------------------------|---|
| 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 |

In addition to the TI software, you need one or more TMS320F2812 eZdsp DSP Starter Kits or TMS320LF2407 eZdsp DSP Starter Kits from Spectrum Digital.

Verifying the Configuration

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

```
c2000lib
```

When you enter this command, MATLAB displays the C2000 block library containing the following libraries that comprise the C2000 library:

- C2800 DSP Core Support
- C2400 DSP Core Support
- Target Preferences
- C28x IQmath library
- C28x DMC library

If you do not see the listed libraries, or MATLAB does not recognize the command, you need to install 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 up-to-date 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

```
ccsboardinfo
```

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

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

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

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

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

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. Once 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.

Scheduling and Timing

A timer interrupt is used to run generated code in real time on the C2000 DSP. 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 C28x 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:

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

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 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 and for the F2812 it is 150 MHz.

Maximum Sample Times

| TimerClockPrescaler Setting | C24x Maximum Sample Time (seconds) | C28x Maximum Sample Time (seconds) |
|------------------------------------|---|---|
| 1 | 0.0016 | 28.63 |
| 2 | 0.0032 | N/A |
| 4 | 0.0065 | N/A |
| 8 | 0.0131 | N/A |
| 16 | 0.0262 | N/A |
| 32 | 0.0524 | N/A |

Maximum Sample Times (Continued)

| TimerClockPrescaler Setting | C24x Maximum Sample Time (seconds) | C28x Maximum Sample Time (seconds) |
|--|---|---|
| 64 | 0.1048 | N/A |
| 128 | 0.2097 | N/A |

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 1-14 for an example of how to use this library.

Create your real-time model for your application the 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 get a list of 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.

Notes About Selecting Blocks for 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.

| Block Name/Category | Library |
|---------------------------------|--------------------------------------|
| Scope | Simulink, Signal Processing Blockset |
| To Workspace | Simulink |
| From Workspace | Simulink |
| Spectrum Scope | Signal Processing Blockset |
| To File | Simulink |
| From File | Simulink |
| Triggered to Workspace | Signal Processing Blockset |
| Signal To Workspace | Signal Processing Blockset |
| Signal From Workspace | Signal Processing Blockset |
| Triggered Signal From Workspace | Signal Processing Blockset |
| To Wave Device | Signal Processing Blockset |
| From Wave Dvice | Signal Processing Blockset |
| To Wave File | Signal Processing Blockset |
| From Wave File | Signal Processing Blockset |

Setting Simulation Configuration Parameters

To set the simulation parameters manually, with your model open, select **Configuration Parameters** from the **Simulation** option. From this dialog, select **Real-Time Workshop**. You must specify the appropriate version of the system target file and template makefile. For the Embedded Target for the TI TMS320C2000™ DSP Platform, in the **Real-Time Workshop** pane of the dialog, specify

```
ti_C2000_grt.tlc
```

or, optionally, select

```
ti_C2000_ert.tlc
```

to select the correct target file or click **Browse** and select from the list of targets. The associated template filename is automatically filled in.

A Generic Real-Time (GRT) target 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 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.

You must also specify discrete time by selecting `Fixed-step` and `discrete (no continuous states)` from the **Solver** panel of the **Configuration Parameters** dialog.

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 is open when you drag the Target Preferences block into the model.

Building Your Model

With this configuration, you can generate a real-time executable and download it 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 Preference Block in your model match exactly.

During the same build operation, block parameter dialog 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 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

Connect the hardware as follows:

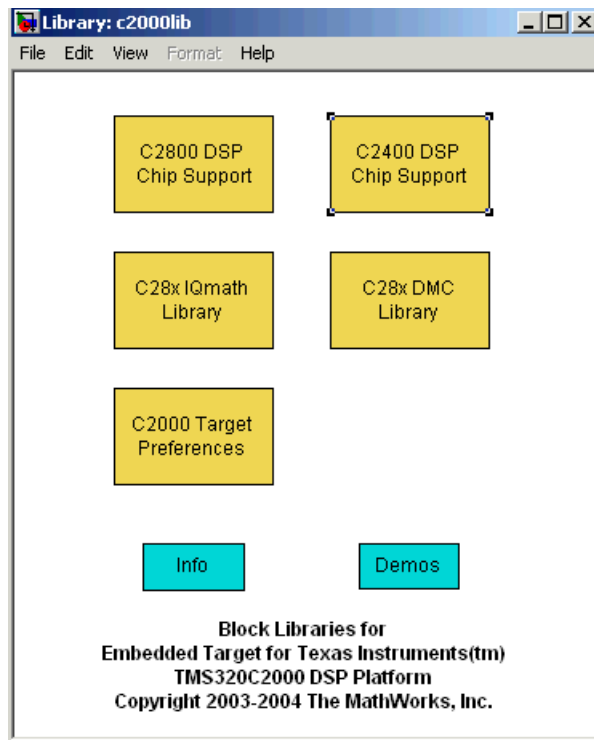
- 1** Connect the function generator output to the ADC input ADCINA0 on the eZdsp F2812.
- 2** Connect the output of PWM1 on the eZdsp F2812 to the analog input of the oscilloscope.

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

- C2800 DSP Core Support (c2800dsp1ib) — Blocks to configure the codec on the F2812 eZdsp DSK or on the F2812 DSP
- C2400 DSP Core Support (c2400dsp1ib) — Blocks to configure the codec on the LF2407 eZdsp DSK or on the LF2407 DSP
- C28x IQMath Library (tiiqmath1ib) — Fixed-point math blocks for use with C28x targets
- C28x DMC Library (c28xdmc1ib) — Fixed-point math blocks for digital motor control with C28x DSPs
- Target Preferences (c2000tgtpref1ib) — Blocks to specify target preferences and options. You do not connect this block to any other block in your model.

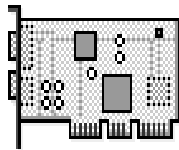
- Info block — Online help
- Demos block — Demos window

For more information on each block, refer to its reference page.

Setting Up the Model

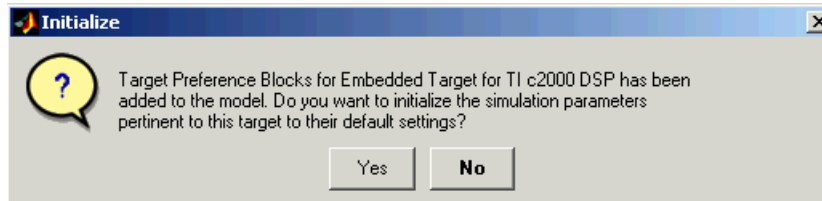
Preliminary tasks for setting up a new model include adding a Target Preferences block, setting or verifying Target Preferences, and setting the simulation parameters.

- 1 Select **New** from the **File** menu to create a new Simulink model.
- 2 Double-click the Target Preferences library in c2000lib to open it.
- 3 Drag the F2812 eZdsp block into your new model.



F2812 eZdsp

The following dialog appears, asking if you want preferences to be set automatically.



Click **Yes** to allow automatic setup. The following **Simulation -> Configuration Parameters** are set:

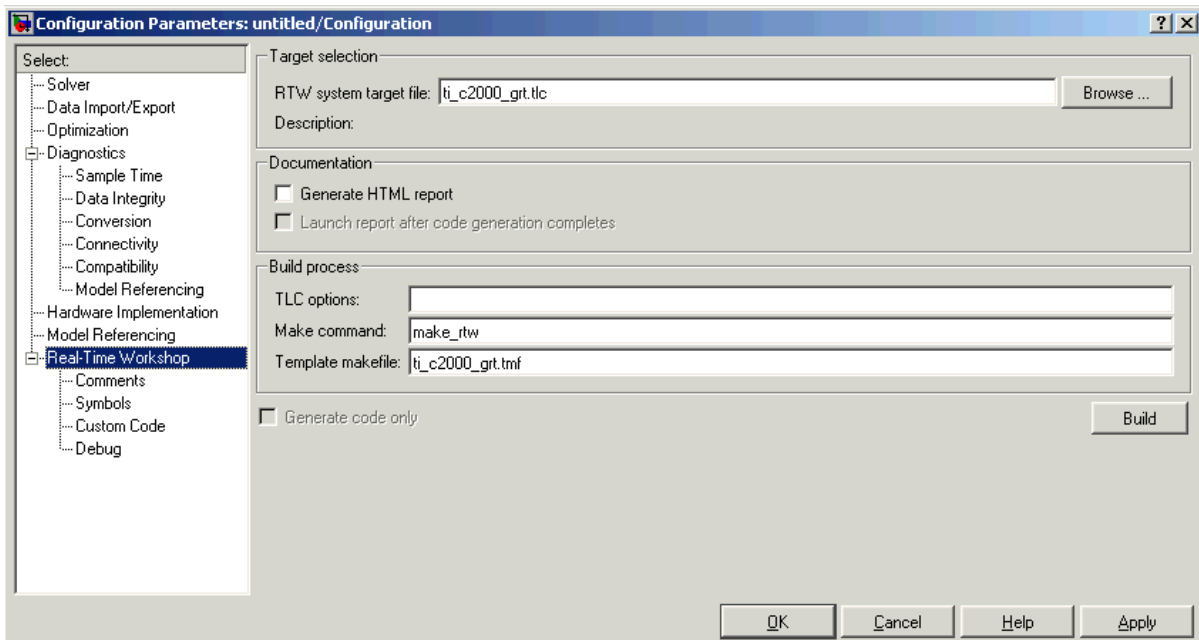
| Panel | Field | Setting |
|-------------------------|--|---------------------|
| Solver | Stop time | inf |
| Solver | Type | Fixed-step discrete |
| Data Import/Export | Save to workspace - Time | off |
| Data Import/Export | Save to workspace - Output | off |
| Hardware Implementation | Device type | 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 |

The default **Target configuration - System target file** is `ti_c2000_grt.tlc` because you need to purchase and install the optional Real-Time Workshop Embedded Coder to use the `ti_c2000_ert.tlc`.

Note One Target Preference 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.

Select **Configuration Parameters** from the **Simulation** menu to verify and set the simulation parameters for this model. Parameters you set in this dialog 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.

Use the **Real-Time Workshop** pane of the **Configuration Parameters** dialog 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.



- **RTW system target file.** Clicking **Browse** opens the **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` as the **Make command**. On **Configuration** in the Target configuration category, enter `make_rtw` for the **Make command**.
- **Template makefile.** Set the **Template makefile** option to `ti_c2000_grt.tmf` or `ti_c2000_ert.tmf` when you build your application for the C2000 target. If the template makefile shown in the option is not the one for the selected **System target file**, click **Browse** to open the list of available system target files and select the correct file from the list. Real-Time Workshop then selects the appropriate template makefile.
- **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, in the **Target Preference BuildOptions — RunTimeOptions** for **BuildAction**, select `Generate code only`.

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

- 4** Set the Target Preferences by double-clicking on the F2812 eZdsp block and adjust these parameters. The default values are also shown in the figure below. 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

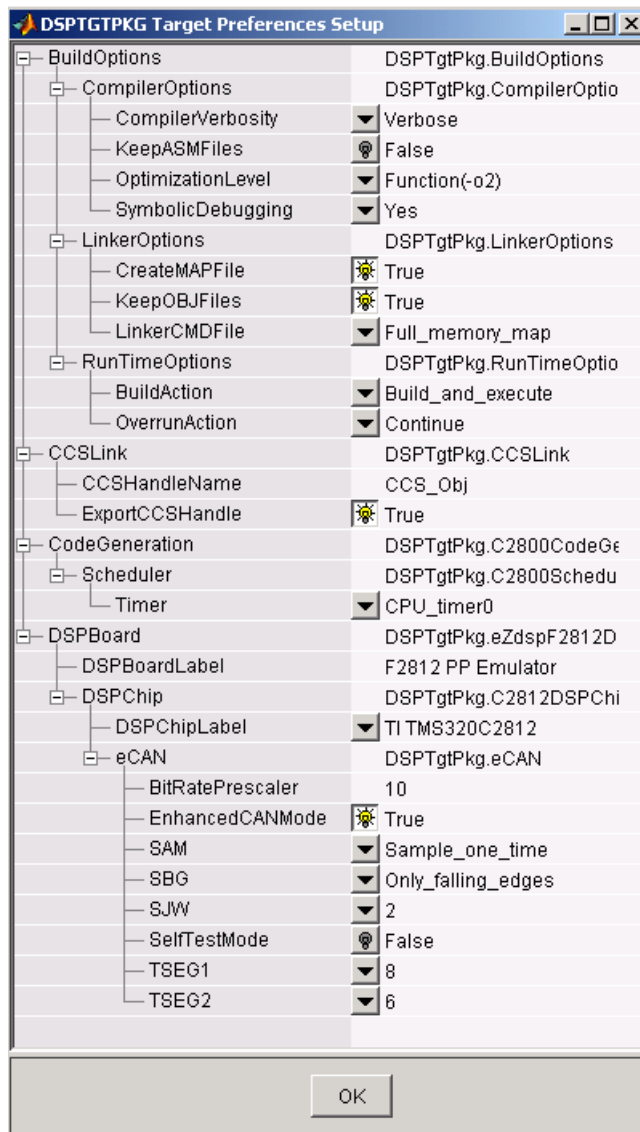
| Subfield | Field | Setting |
|-----------------|----------------------------|----------------|
| Scheduler | Timer | CPU_timer0 |
| | TimerClockPrescaler | 1 |

DSPBoard Options

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

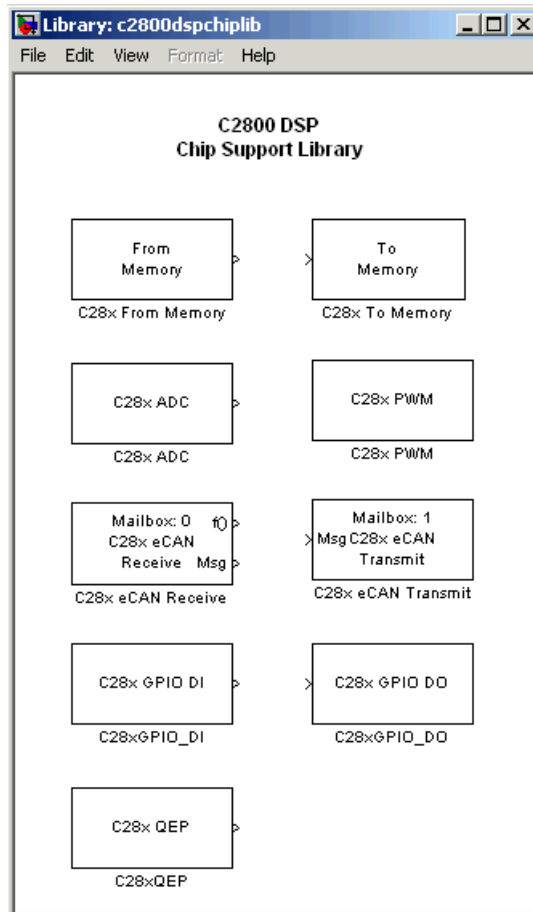
Note If the board label in your Code Composer Studio setup is different than the default DSP Board Label shown in the Target Preferences block, you can change the default setting. This would assure that whenever you drag a Target Preferences block into a new model, the DSP Board Label of your model will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

Open the C2000 Target Preferences library. Double-click on the appropriate Target Preferences block. Click on **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.

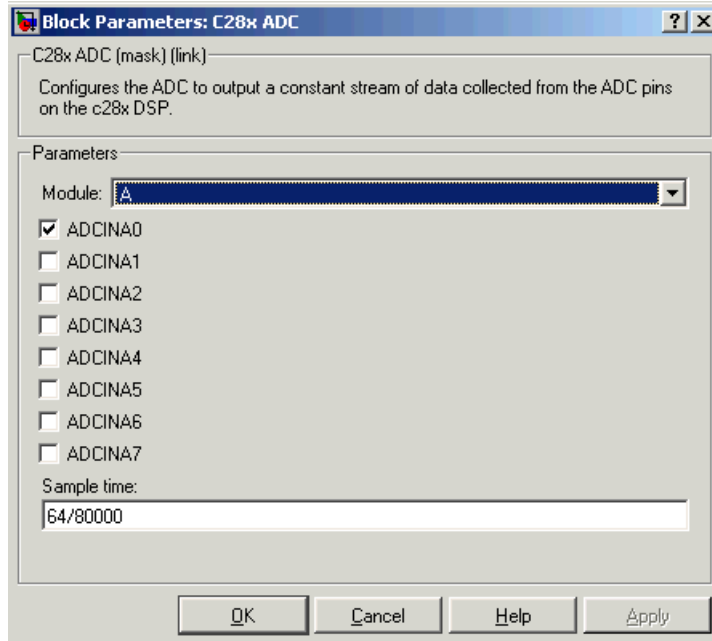


Adding Blocks to the Model

- 1 Double-click the C2800 DSP Chip Support Library to open it.



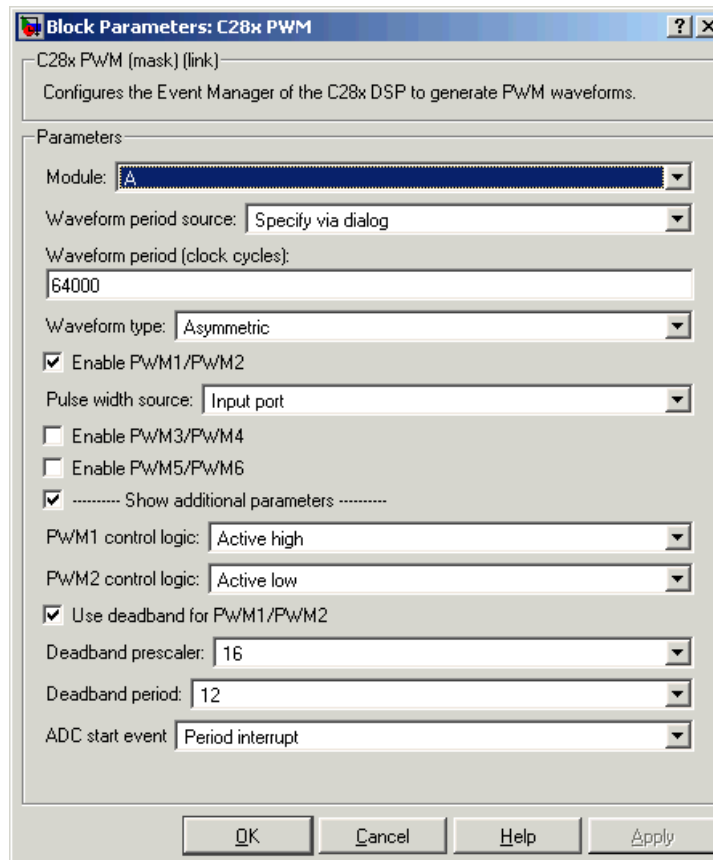
- 2 Drag the C28x ADC block into your model. Double-click the ADC block in the model and set the **Module** to A, select only **ADCINA0**, and enter a **Sample time** of 64/80000. Refer to the C28x ADC reference page for information on these fields.



- 3 Drag the C28x PWM block into your model. Double-click the PWM block in the model and set the following parameters. Refer to the C28x PWM reference page for information on these fields.

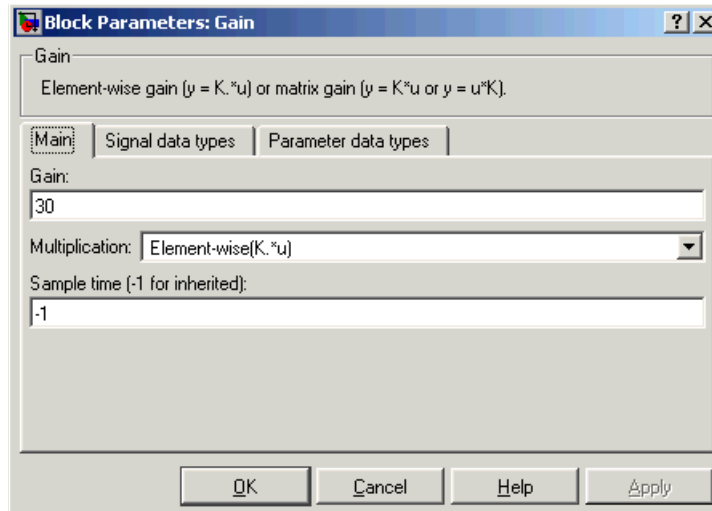
| Field | Parameter |
|----------------------------|--------------------|
| Module | A |
| Waveform period source | Specify via dialog |
| Waveform period | 64000 |
| Waveform type | Asymmetric |
| Enable PWM1/PWM2 | selected |
| Pulse width source | Input port |
| Show additional parameters | selected |

| Field | Parameter |
|-----------------------------------|------------------|
| PWM1 control logic | Active high |
| PWM2 control logic | Active low |
| Use deadband for PWM1/PWM2 | selected |
| Deadband prescaler | 16 |
| Deadband period | 12 |
| ADC start event | Period interrupt |

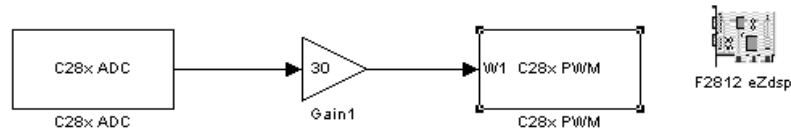


- 4 Type Simulink at the MATLAB command line to start the **Simulink Library browser**. Drag a Gain block from the **Math Operations** library into your model. Double-click on the Gain block in the model and set the following parameters.

| Field | Parameter |
|-----------------------------------|---------------------|
| Gain | 30 |
| Multiplication | Element-wise (K.*u) |
| Sample time | -1 |
| Output data type mode | Specify via dialog |
| Output data type | uint(16) |
| Round integer calculations toward | Floor |
| 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 your 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 using the Build all button on the toolbar of your model, or by using the keyboard shortcut, **Ctrl+B**, while your model is open and in focus.

The code building process consists of these tasks:

- 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, `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, follow these steps:

- 1 In the **Real-Time Workshop** pane in the **Simulation Parameters** dialog, 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, 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.

Using the IQmath Library

About the IQmath Library (p. 2-2)

Introduces the IQmath Library

Fixed-Point Numbers (p. 2-3)

Representation of fixed-point numbers in the IQmath Library

Building Models (p. 2-7)

Issues to consider when you build models with the IQmath Library

About the IQmath Library

The blocks in the C28x IQmath Library correspond to functions in the Texas Instruments C28x IQmath Library assembly-code library, which target the TI C2800 family of digital signal processors. You can use these blocks to run simulations by building models in Simulink 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 C2000 DSP. During code generation, each IQmath Library block in your model is mapped to its corresponding TI IQmath Library assembly-code routine to create target-optimized code.

The IQmath Library blocks generally input and output fixed-point data types. The block reference pages discuss the data types accepted and produced by each block in the library. “Fixed-Point Numbers” on page 2-3 gives a brief overview of using fixed-point data types in Simulink. For a discussion of this topic, including issues with scaling and precision when performing fixed-point operations, refer to the Simulink Fixed Point documentation.

You can use IQmath Library blocks with certain core Simulink blocks, as well as with certain blocks from Simulink Fixed Point. To learn more about creating models that include both IQmath Library blocks and blocks from other blocksets, refer to “Building Models” on page 2-7.

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.

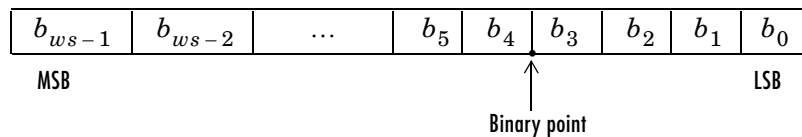
To learn more about characteristics particular to each block in the library, refer to the “Block Reference” pages.

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 four.

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

$$Q_{m.n}$$

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.

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:

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

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:

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

Therefore, this number is expressed as

$$Q-2.17$$

In Simulink Fixed Point, this data type is expressed as

```
sfixed16_En17
```

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

```
[16 17]
```

Example – Q17.-2

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

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

Therefore, this number is expressed as

$$Q17.-2$$

In Simulink Fixed Point, this data type is expressed as

```
sfixed16_E2
```

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

```
[16 -2]
```

Building Models

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

Converting Data Types

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

Block Reference

Blocks — Categorical List (p. 3-2)

Provides tables that list each block in the Embedded Target for C2000 DSP by library

Blocks — Alphabetical List (p. 3-5)

Lists each block in the Embedded Target for C2000 DSP in alphabetical order

Blocks — Categorical List

C2400 DSP Chip Support Library (c2400dspchiplib)

| | |
|--------------------------|---|
| C24x ADC | Configure analog-to-digital converters (ADC) |
| C24x CAN Receive | Configure an enhanced Control Area Network receive mailbox |
| C24x CAN Transmit | Configure an enhanced Control Area Network transmit mailbox |
| C24x GPIO Digital Input | Configure the general-purpose I/O pins for digital input |
| C24x GPIO Digital Output | Configure the general-purpose I/O pins for digital output |
| C24x PWM | Configure one or more pairs of pulse wave modulators (PWMs) |
| C24x QEP | Configure the quadrature encoder pulse circuit |
| From Memory | Retrieve data from a specific memory location on the target |
| To Memory | Write data to a specific memory location on the target |

C2800 DSP Chip Support Library (c2800dspchiplib)

| | |
|-------------------------|---|
| C28x ADC | Configure analog-to-digital converters (ADC) |
| C28x eCAN Receive | Configure an enhanced Control Area Network receive mailbox |
| C28x eCAN Transmit | Configure an enhanced Control Area Network transmit mailbox |
| C28x GPIO Digital Input | Configure the general-purpose I/O pins for digital input |

| | |
|--------------------------|---|
| C28x GPIO Digital Output | Configure the general-purpose I/O pins for digital output |
| C28x PWM | Configure one or more pairs of pulse wave modulators (PWMs) |
| C28x QEP | Configure the quadrature encoder pulse circuit |
| From Memory | Retrieve data from a specific memory location on the target |
| To Memory | Write data to a specific memory location on the target |

Target Preferences Library (c2000tgtplib)

| | |
|--------------|--|
| F2812 eZdsp | Preferences for F2812 eZdsp DSK targets |
| LF2407 eZdsp | Preferences for LF2407 eZdsp DSK targets |

C28x Digital Motor Control Library (c28xdmclib)

| | |
|-----------------------------|---|
| Clarke Transformation | Convert balanced three phase quantities into balanced two phase quadrature quantities |
| Inverse Park Transformation | Convert rotating reference frame vectors to a two-phase stationary reference frame |
| Park Transformation | Convert two-phase stationary system vectors to rotating system vectors |
| PID Controller | Create a digital PID controller |
| Space Vector Generator | Calculate duty ratios to generate stator reference voltage |
| Speed Measurement | Calculate motor speed |

C28x IQmath Library (tiiqmathlib)

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

Blocks — Alphabetical List

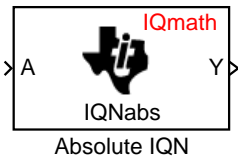
This section contains block reference pages listed alphabetically.

Absolute IQN

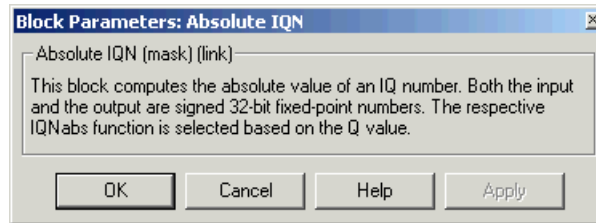
Purpose Calculate the absolute value of an IQ number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

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



Dialog Box

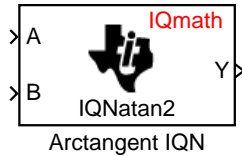


See Also Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Purpose Calculate the four-quadrant arc tangent

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

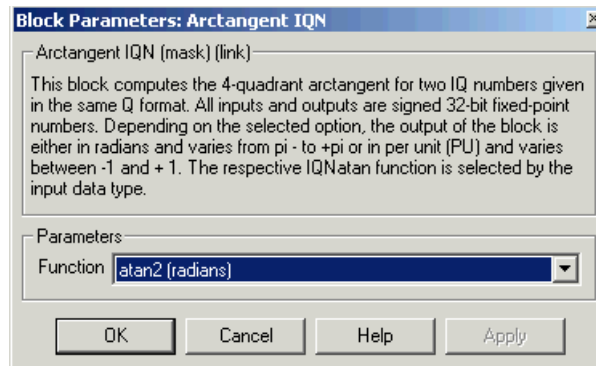
Description This block computes the four-quadrant arc tangent of the IQ number inputs and produces IQ number output.



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, $\text{atan2PU}(B,A) = \text{atan2}(B,A) / 2\pi$. Otherwise, $\text{atan2PU}(B,A) = \text{atan2}(B,A) / 2\pi + 1$. The output is in per-unit radians with values from 0 to 2π radians.

Dialog Box



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

C24x ADC

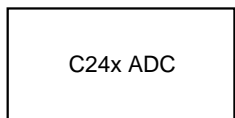
Purpose

Generate code to configure the C24x analog-to-digital converter

Library

c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description



The C24x ADC block configures the C24x ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. The ADC block outputs digital values representing the analog input signal and stores the converted values in the result register of your digital signal processor. You use this block to capture and digitize analog signals from external sources such as signal generators, frequency generators, or audio devices.

Triggering

The C24x ADC trigger mode depends on the internal setting of the Source Start-of-Conversion (SOC) signal. The ADC is usually triggered by software at the sample time intervals specified in the ADC block — this is unsynchronized mode.

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 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-sequencer 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 operation in dual-sequencer and cascaded-sequencer modes. In dual-sequencer 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-sequencer mode, both Module A and Module B are used for a single ADC block.

Module — Specifies which DSP module to use:

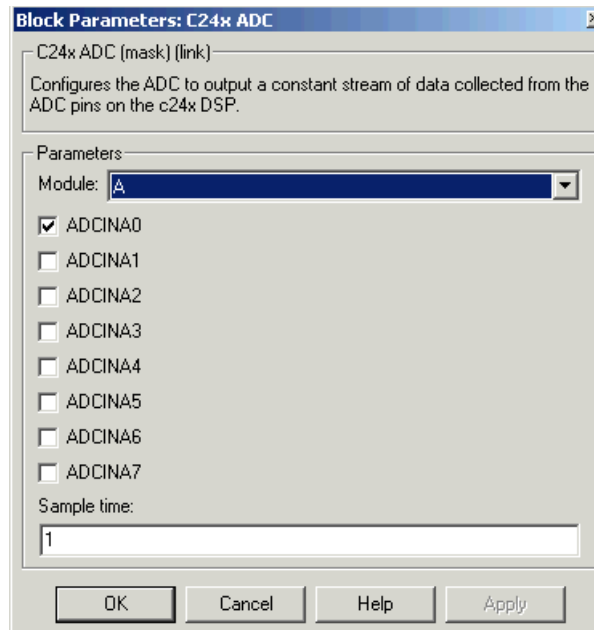
- A — Displays the ADC channels in module A (ADCINA0 through ADCINA7)
- B — Displays the ADC channels in module B (ADCINB0 through ADCINB7)
- A and B — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7).

Use the check boxes to select the desired ADC channels.

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 1-8 for additional 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.

Dialog Box



See Also

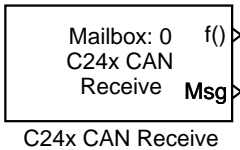
C24x PWM

C24x CAN Receive

Purpose Configure a CAN mailbox to receive messages from the CAN pins and output received messages at specified sample intervals

Library c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description 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. The length of the vector is always 8 bytes.
- The third output port is optional and appears only if **Output message length** is selected.

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.

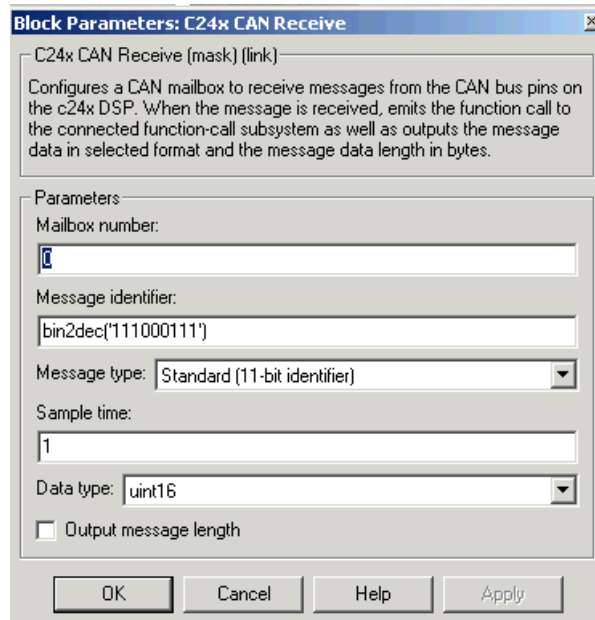
Message type — Select Standard (11-bit identifier) or Extended (29-bit identifier).

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 8-byte data vector. Valid values are `uint16` or `uint32`.

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.

Dialog Box



See Also

C24x CAN Transmit

References

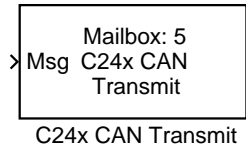
Detailed information on the CAN module is in the *TMS320LF/LC240xA DSP Controller Reference Guide — System and Peripherals*, Literature Number SPRU357B, available at the Texas Instruments Web site.

C24x CAN Transmit

Purpose Configure a CAN mailbox to transmit messages to the CAN pins

Library c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description



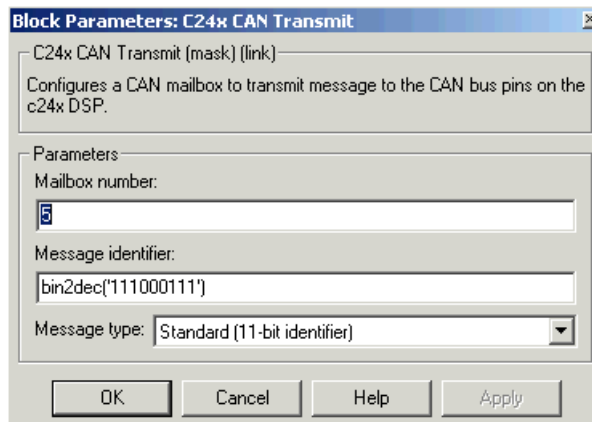
The C24x Control Area Network (CAN) Transmit block generates source code for transmitting CAN messages through a CAN mailbox. The CAN module on the DSP chip provides serial communication capability and has six mailboxes — two for receive, two for transmit, and two configurable for receive or transmit.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

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).

Dialog Box



See Also C24x CAN Receive

References

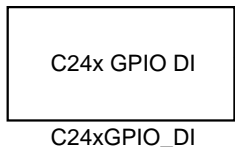
Detailed information on the CAN module is in the *TMS320LF/LC240xA DSP Controller Reference Guide — System and Peripherals*, Literature Number SPRU357B, available at the Texas Instruments website.

C24x GPIO Digital Input

Purpose Configure the shared general-purpose input/output pin registers

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.



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. Unselected bits are available for peripheral functionality. Note that 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.

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.

The following tables show the shared pins.

IO MUX Output Control Register A

| Bit | Peripheral Name | GPIO Name |
|-----|-----------------|-----------|
| 3 | QEP1 | IOPA3 |
| 4 | QEP2 | IOPA4 |
| 6 | PWM1 | IOPA6 |
| 7 | PWM2 | IOPA7 |
| 8 | PWM3 | IOPB0 |

IO MUX Output Control Register A (Continued)

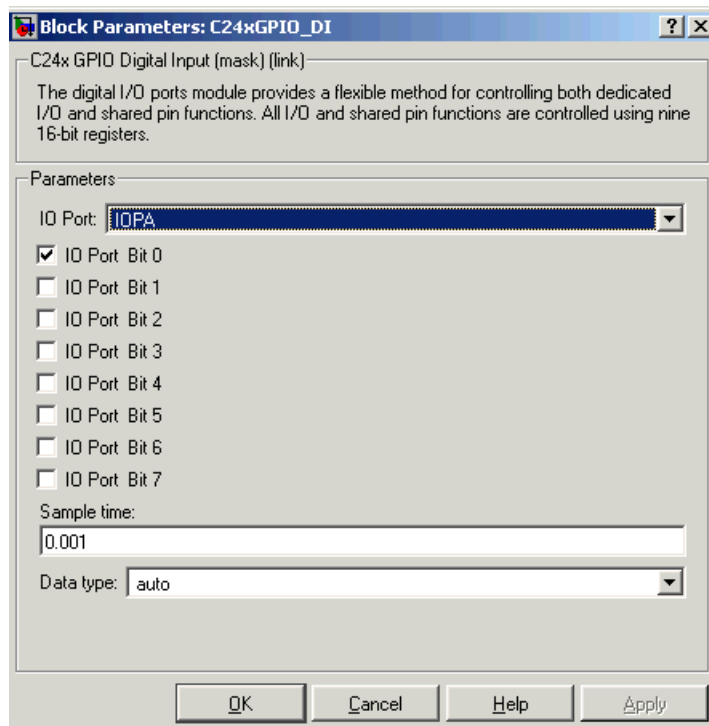
| Bit | Peripheral Name | GPIO Name |
|-----|-----------------|-----------|
| 9 | PWM4 | IOPB1 |
| 10 | PWM5 | IOPB2 |
| 11 | PWM6 | IOPB3 |

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 | IOPE7 |
| 8 | QEP4 | IOPF0 |

C24x GPIO Digital Input

Dialog Box

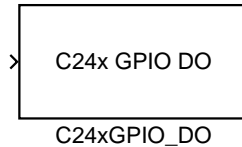


See Also C24x GPIO Digital Output

Purpose Configure the shared general-purpose input/output pin registers

Library c2400dspchip1lib in Embedded Target for TI C2000 DSP

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



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. 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.

IO MUX Output Control Register A

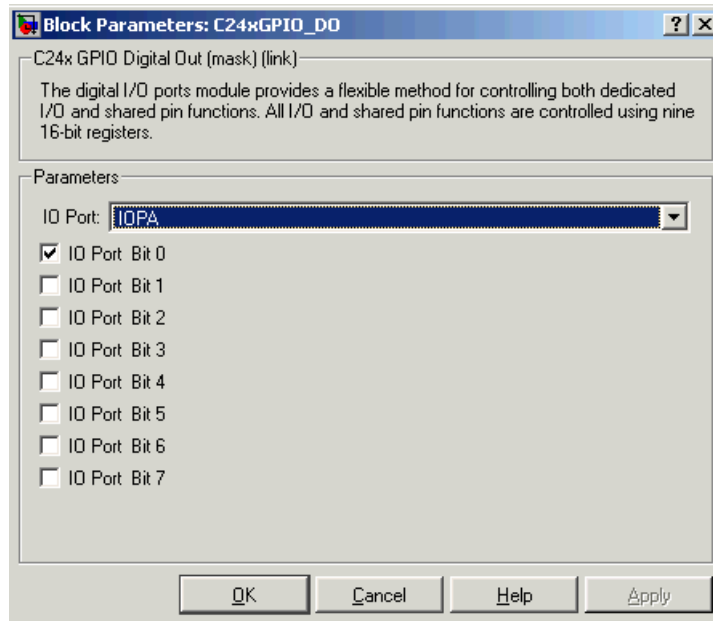
| Bit | Peripheral Name | GPIO Name |
|-----|-----------------|-----------|
| 3 | QEP1 | IOPA3 |
| 4 | QEP2 | IOPA4 |
| 6 | PWM1 | IOPA6 |
| 7 | PWM2 | IOPA7 |
| 8 | PWM3 | IOPB0 |
| 9 | PWM4 | IOPB1 |
| 10 | PWM5 | IOPB2 |
| 11 | PWM6 | IOPB3 |

C24x GPIO Digital Output

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 | IOPE7 |
| 8 | QEP4 | IOPF0 |

Dialog Box



See Also

C24x GPIO Digital Input

C24x PWM

Purpose Generate code that configures the Event Manager (EV) modules to generate PWM waveforms

Library c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description 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.

C24x PWM

C24x PWM

Timer Panel

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).

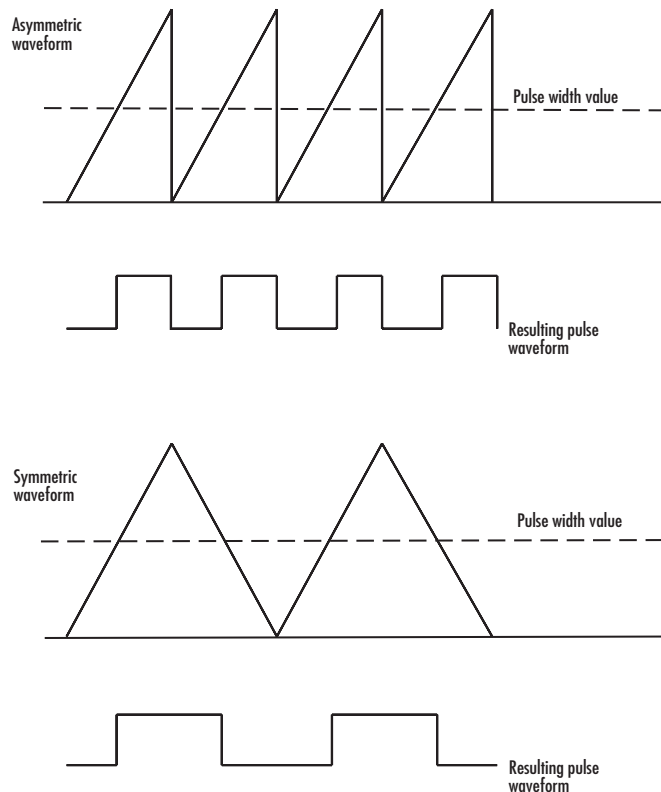
Note PWMs in module A use Event Manager A, Timer 1, and PWMs in module B use Event Manager B, Timer 3. You should make sure that the **TimerClock** selected in the Scheduling section of the LF2407 eZdsp Target Preferences block does not conflict with the timers used for the PWMs.

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 timer used to generate PWM waveform measured in clock cycles. The relationship between the timer period and the waveform period depends on the **Waveform type**.

Note Clock cycles refers to the system CPU clock on the LF2407 chip. This clock is 40 MHz.

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



Output 1/Output 2/Output 3 Panels

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

Pulse width source — Source from which the pulse width is obtained. Select Specify via dialog to enter the value in **Pulse width** or select Input port to use a value from the input port.

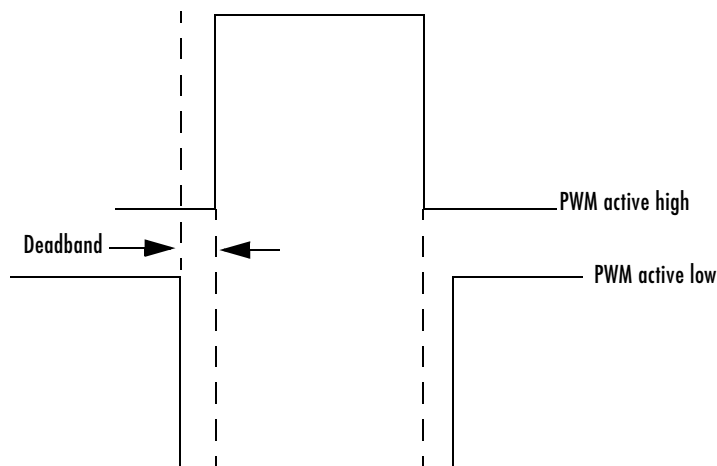
C24x PWM

Pulse width — Width of the pulse in clock cycles. The default is for the first PWM in a pair to be triggered active high and for the second PWM to be triggered active low.

PWM# control logic — Control logic trigger for the PWM. 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.

Deadband Panel

Use deadband for PWM#/PWM# — Enables a deadband area of no signal overlap at the beginning of particular PWM pair signals.



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

ADC Control Panel

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.

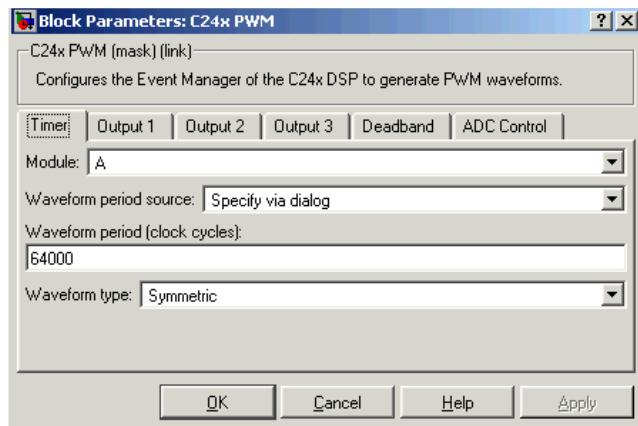
Note If you select **Period interrupt** and specify a sampling time less than the specified **(Waveform period)/(CPU Clock speed)**, zero-order hold interpolation will occur. (For example, if you enter 64000 as the waveform period, the period for the ADC register is $64000/40 \text{ MHz} = .0016$. If 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.

C24x PWM

Dialog Box

Only the Timer panel is shown in the figure below. Press the desired tab to display other panels.



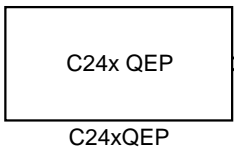
See Also

C24x ADC

Purpose Configure the quadrature encoder pulse circuit

Library c2400dspchip1lib in Embedded Target for TI C2000 DSP

Description



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

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

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

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:

- CountBase — Count the pulses based on the board’s GP Timer 2 (or GP Timer 4 for EVB).
- RPMBase — Count the machine’s revolutions per minute.

Positive rotation — Defines whether to use Clockwise or Counter clockwise as the direction to use as positive rotation. This field appears only if you select RPMBase above.

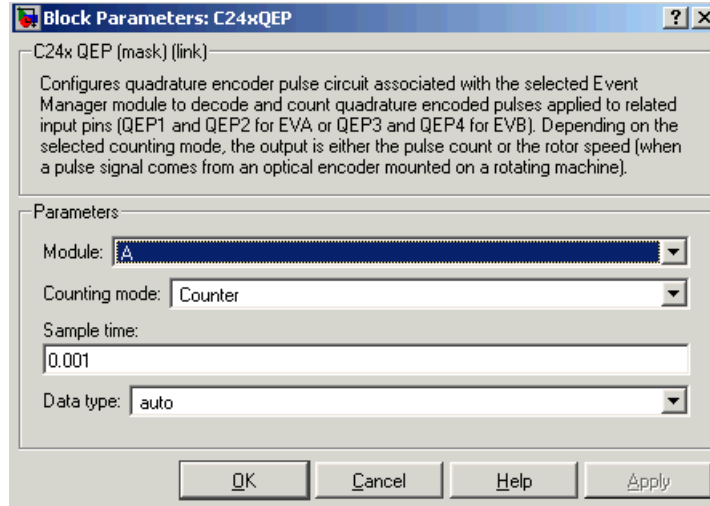
Encoder resolution — Number of QEP pulses per revolution. This field appears only if you select RPMBase above.

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

C24x QEP

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.

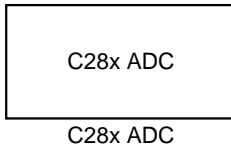
Dialog Box



Purpose Generate code to configure the ADC to output data streams

Library c2800dspchip1lib in Embedded Target for TI C2000 DSP

Description



The C28x ADC block configures the C28x 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 C28x 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 PWM waveforms generated by the same EV unit via the **ADC Start Event** signal setting. The **ADC Start Event** is set in the C28x 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-sequencer mode (see below).

Output

The output of the C28x ADC is a vector of uint16 values. The output values are in the range 0 to 4095 because the C28x ADC is 12-bit converter.

Modes

The C28x ADC block supports ADC operation in dual-sequencer and cascaded-sequencer modes. In dual-sequencer 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-sequencer mode, both Module A and Module B are used for a single ADC block.

Module — Specifies which DSP module to use:

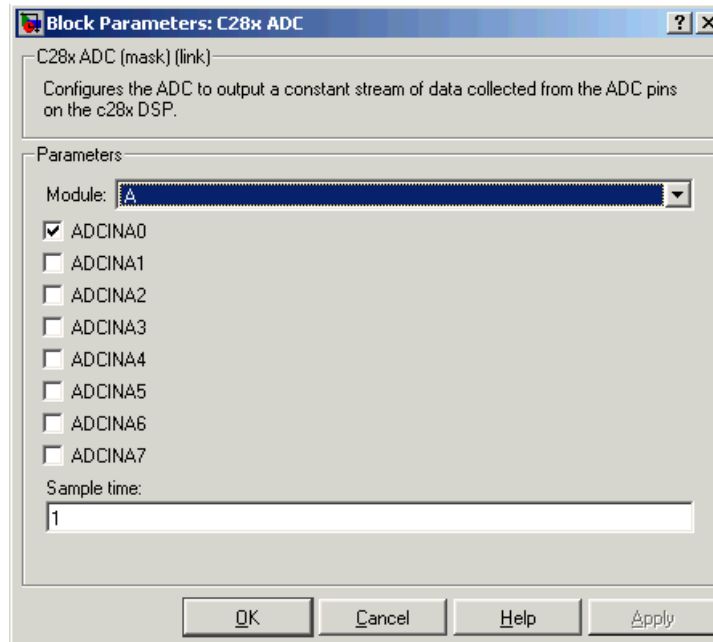
- A — Displays the ADC channels in module A (ADCINA0 through ADCINA7).
- B — Displays the ADC channels in module B (ADCINB0 through ADCINB7).
- A and B — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7)

Use the check boxes to select the desired ADC channels.

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 1-8 for additional information on timing.

To set different sample times for different groups of ADC channels, you must add separate C28x ADC blocks to your model and set the desired sample times for each block.

Dialog Box



See Also

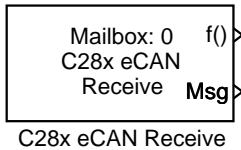
C28x PWM

C28x eCAN Receive

Purpose Configure an eCAN mailbox to receive messages from the eCAN pins and output received messages at specified sample intervals

Library c2800dspchip1lib in Embedded Target for TI C2000 DSP

Description The C28x 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 C28x 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.

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.

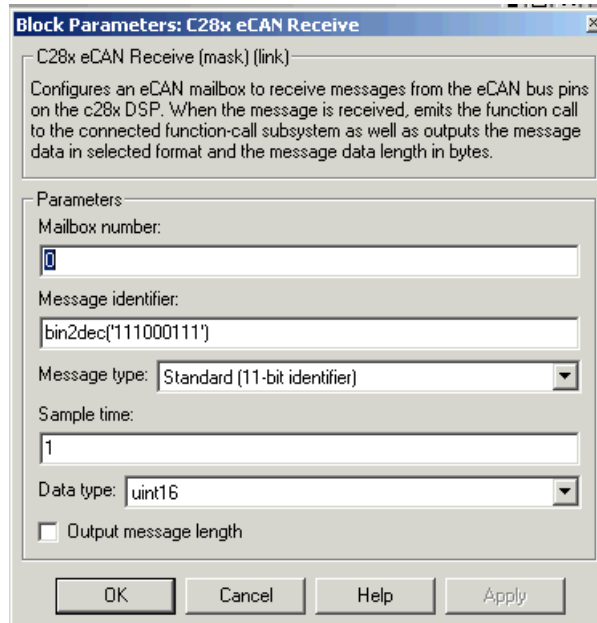
Message type — Select Standard (11-bit identifier) or Extended (29-bit identifier).

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 the data in the 8-byte data vector. Valid values are `uint16` or `uint32`.

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.

Dialog Box



See Also

C28x eCAN Transmit

References

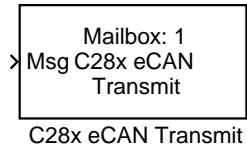
Detailed information on the eCAN module is in the *TMS320F28x DSP Enhanced Control Area Network (eCAN) Reference Guide*, Literature Number SPRU074A, available at the Texas Instruments Web site.

C28x eCAN Transmit

Purpose Configure an eCAN mailbox to transmit a message to the board's CAN bus pins

Library c2800dspchip1lib in Embedded Target for TI C2000 DSP

Description



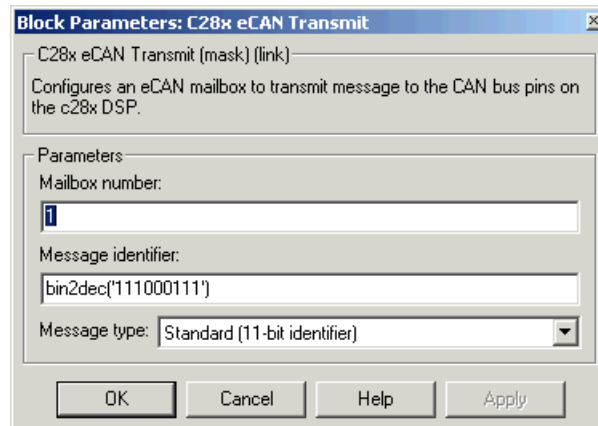
The C84x enhanced Control Area Network (eCAN) Transmit block generates source code for transmitting eCAN messages through an eCAN mailbox. The eCAN module on the DSP chip provides serial communication capability and has 32 mailboxes configurable for receive or transmit. The C28x supports eCAN data frames in standard or extended format.

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).

Dialog Box



See Also C28x eCAN Receive

References

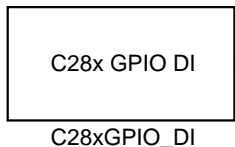
Detailed information on the eCAN module is in the *TMS320F28x DSP Enhanced Control Area Network (eCAN) Reference Guide*, Literature Number SPRU074A, available at the Texas Instruments Web site.

C28x GPIO Digital Input

Purpose Configure the shared general-purpose input/output pin registers

Library c2800dspchip1lib 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.



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. Unselected bits are available for peripheral functionality. Note that 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.

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.

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 |

GPIO A MUX (Continued)

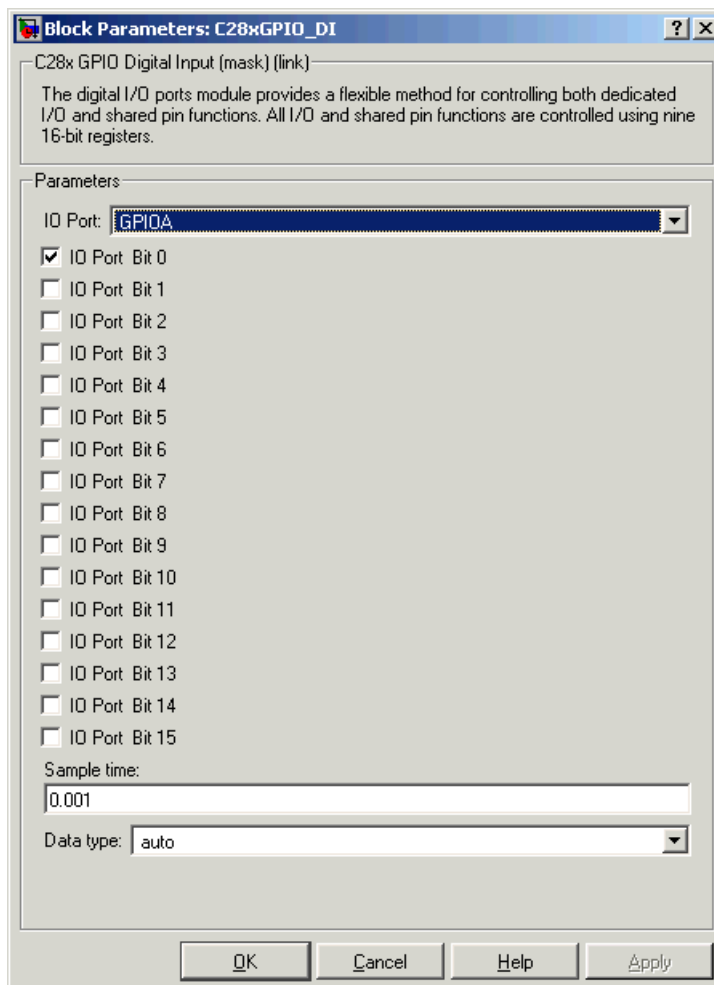
| Bit | Peripheral Name (bit = 1) | GPIO Name (bit = 0) |
|-----|------------------------------|------------------------|
| 4 | PWM5 | GPIOA4 |
| 5 | PWM6 | GPIOA5 |
| 8 | QEP1 | GPIOA8 |
| 9 | QEP2 | GPIOA9 |

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 | GPIOB8 |
| 9 | QEP4 | GPIOB9 |

C28x GPIO Digital Input

Dialog Box

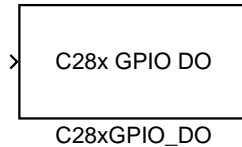


See Also C28x GPIO Digital Output

Purpose Configure the shared general-purpose input/output pin registers

Library c2800dspchip1lib 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.



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 output. 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

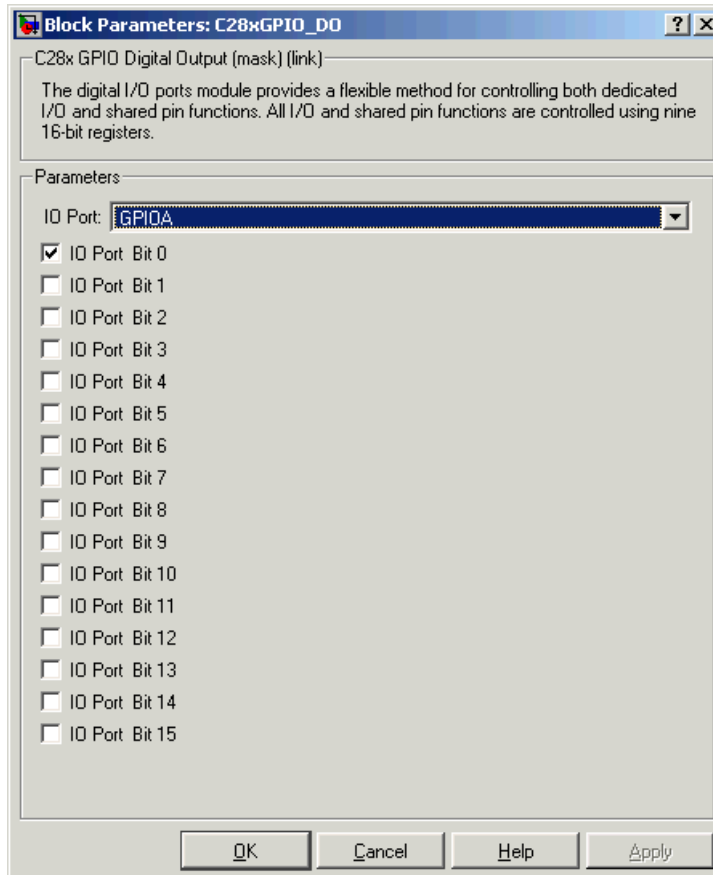
| Bit | Peripheral Name (bit = 1) | GPIO Name (bit = 0) |
|-----|------------------------------|------------------------|
| 0 | PWM1 | GPIOA0 |
| 1 | PWM2 | GPIOA1 |
| 2 | PWM3 | GPIOA2 |
| 3 | PWM4 | GPIOA3 |
| 4 | PWM5 | GPIOA4 |
| 5 | PWM6 | GPIOA5 |
| 8 | QEP1 | GPIOA8 |
| 9 | QEP2 | GPIOA9 |

C28x GPIO Digital Output

GPIO B MUX

| Bit | Peripheral Name (bit = 1) | GPIO Name (bit = 0) |
|-----|------------------------------|------------------------|
| 0 | PWM7 | GPIOB0 |
| 1 | PWM8 | GPIOB1 |
| 2 | PWM9 | GPIOB2 |
| 3 | PWM10 | GPIOB3 |
| 4 | PWM11 | GPIOB4 |
| 5 | PWM12 | GPIOB5 |
| 8 | QEP3 | GPIOB8 |
| 9 | QEP4 | GPIOB9 |

Dialog Box



See Also

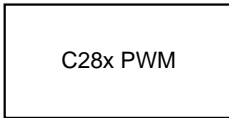
C28x GPIO Digital Input

C28x PWM

Purpose Generate code that configures the Event Manager (EV) modules to generate PWM waveforms

Library c2800dspchip1lib in Embedded Target for TI C2000 DSP

Description F2812 DSPs include a suite 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.



C28x PWM

Timer Panel

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).

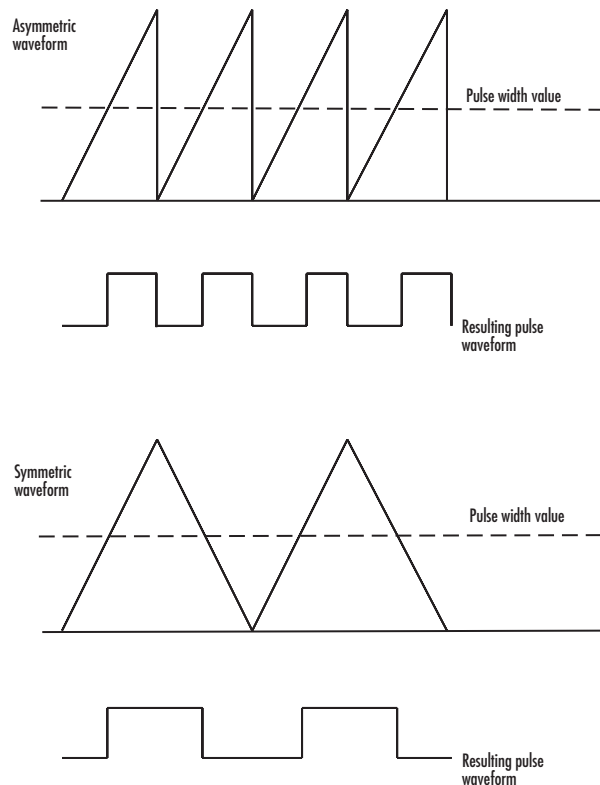
Note PWMs in module A use Event Manager A, Timer 1, and PWMs in module B use Event Manager B, Timer 3. You should make sure that the **TimerClock** selected in the Scheduling section of the F2812 eZdsp Target Preferences block does not conflict with the timers used for the PWMs.

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 timer used to generate the PWM waveform measured in clock cycles. The relationship between the timer period and the waveform period depends on the **Waveform type**.

Note Clock cycles refers to the system CPU clock on the F2812 chip. This clock is 150 MHz.

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.



Output 1/Output 2/Output 3 Panels

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

Pulse width source — Source from which the pulse width is obtained. Select Specify via dialog to enter the value in **Pulse width** or select Input port to use a value from the input port.

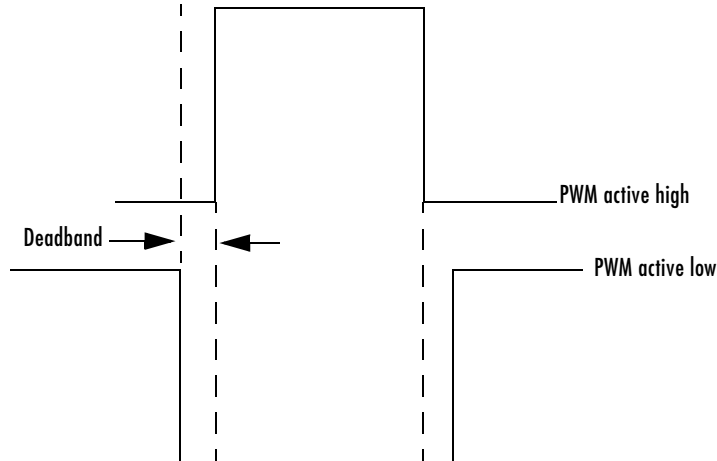
C28x PWM

Pulse width — Width of the pulse in clock cycles. The default is for the first PWM in a pair to be triggered active high and for the second PWM to be triggered active low.

PWM# control logic — Control logic trigger for the PWM. 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.

Deadband Panel

Use deadband for PWM#/PWM# — Enables a deadband area of no signal overlap at the beginning of particular PWM pair signals.



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

ADC Control Panel

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.

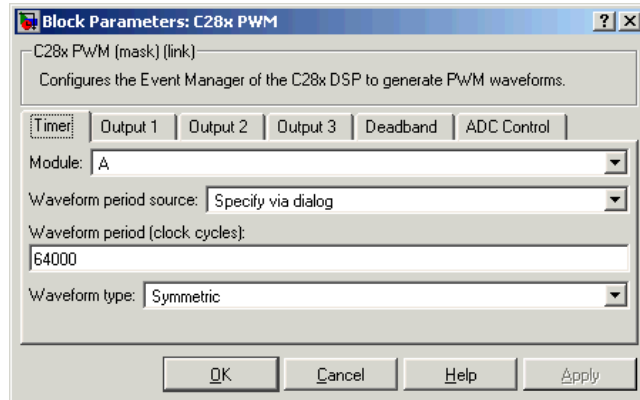
Note If you select **Period interrupt** and specify a sampling time less than the specified **(Waveform period)/(CPU Clock speed)**, zero-order hold interpolation will occur. (For example, if you enter 64000 as the waveform period, the period for the ADC register is $64000/150 \text{ MHz} = 4.26673\text{-}004$. If you enter a **Sample time** in the C28x 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.

C28x PWM

Dialog Box

Only the Timer panel is shown in the figure below. Press the desired tab to display other panels.



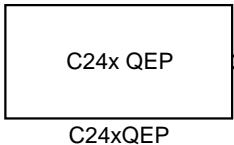
See Also

C28x ADC

Purpose Configure the quadrature encoder pulse circuit

Library c2800dspchip1lib 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.

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:

- CountBase — Count the pulses based on the board’s GP Timer 2 (or GP Timer 4 for EVB).
- RPMBase — Count the machine’s revolutions per minute.

Positive rotation — Defines whether to use Clockwise or Counter **clockwise** as the direction to use as positive rotation. This field appears only if you select RPMBase above.

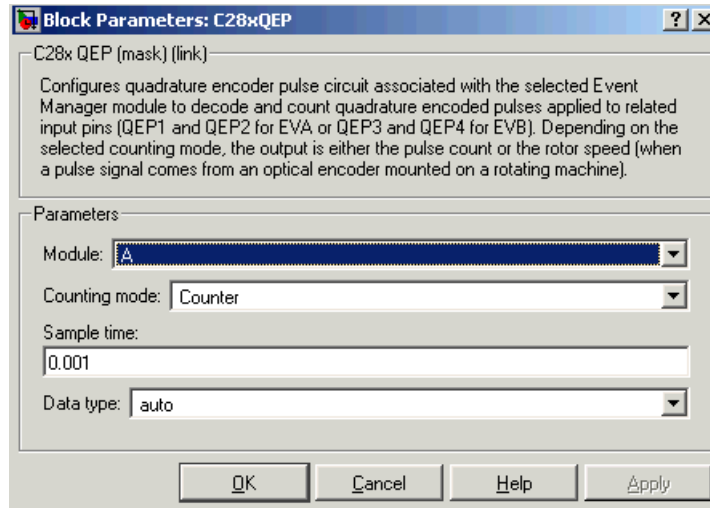
Encoder resolution — Number of QEP pulses per revolution. This field appears only if you select RPMBase above.

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

C28x QEP

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.

Dialog Box

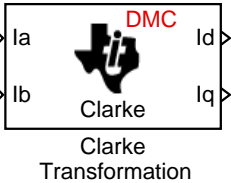


Clarke Transformation

Purpose Transform three-phase quantities into two-phase quadrature quantities

Library c28xdmc1.lib in Embedded Target for TI C2000 DSP

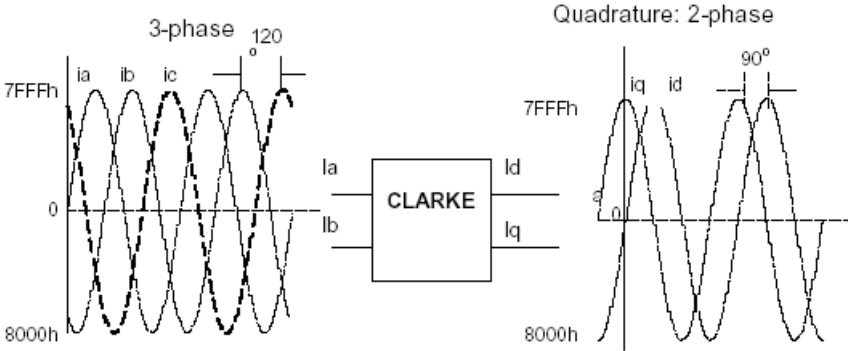
Description This block converts balanced three-phase quantities into balanced two-phase quadrature quantities. The transformation implements these equations



$$I_d = I_a$$

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

and is illustrated in the following figure.



The inputs to this block are the phase a (I_a) and phase b (I_b) components of the balanced three-phase quantities and the outputs are the direct axis (I_d) component and the quadrature axis (I_q) of the transformed signal.

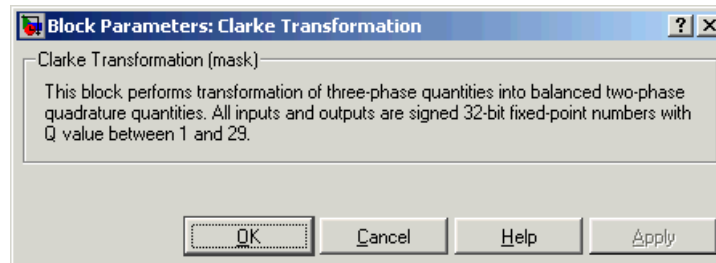
The instantaneous outputs are defined by the following equations:

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

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

Clarke Transformation

Dialog Box



See Also

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

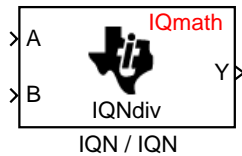
References

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.

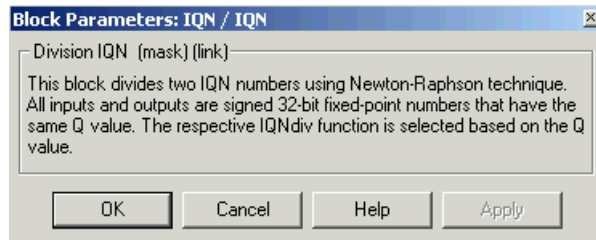
Purpose Divide two IQ numbers

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block divides two numbers that use the same Q format, using the Newton-Raphson technique. The resulting quotient uses the same Q format at the inputs.



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

F2812 eZdsp

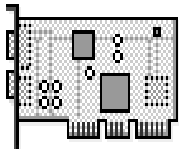
Purpose

Set the build, link, and board code generation preferences for F2812 eZdsp DSK targets

Library

c2000tgtpreflib in Embedded Target for TI C2000 DSP

Description



F2812 eZdsp

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

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

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 `true` — .asm files are kept in your current directory. If you choose not to keep the .asm files, set this option to `false`.
- **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` — 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 **File Name** field. This option allows you to 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.

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. `Internal_memory_map` represents the most efficient memory use.

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**
 - **Timer** — CPU timer to use for scheduling.

DSPBoard

- **DSPBoardLabel** — Name of the installed DSP board. Click in the edit box to change the label.

Note The board label here must match exactly the label (name) of the board entered in your Code Composer Studio setup.

- **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.
- **eCAN** — Parameters that affect the extended control area network (eCAN) module. Most of these parameters affect the eCAN 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

$$TQ = \frac{1}{SYSCLK} \cdot (BitRatePrescalar + 1)$$

where *SYSCLK* is the CAN module system clock frequency, and the *BitRatePrescalar* 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 settable parameters are

BitRatePrescaler — 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.

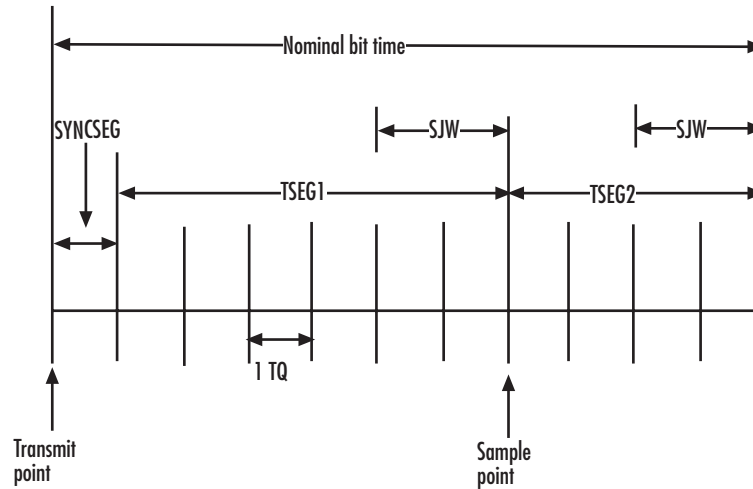
SelfTestMode — If `True`, sets the eCAN module to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit. The default is `False`.

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`

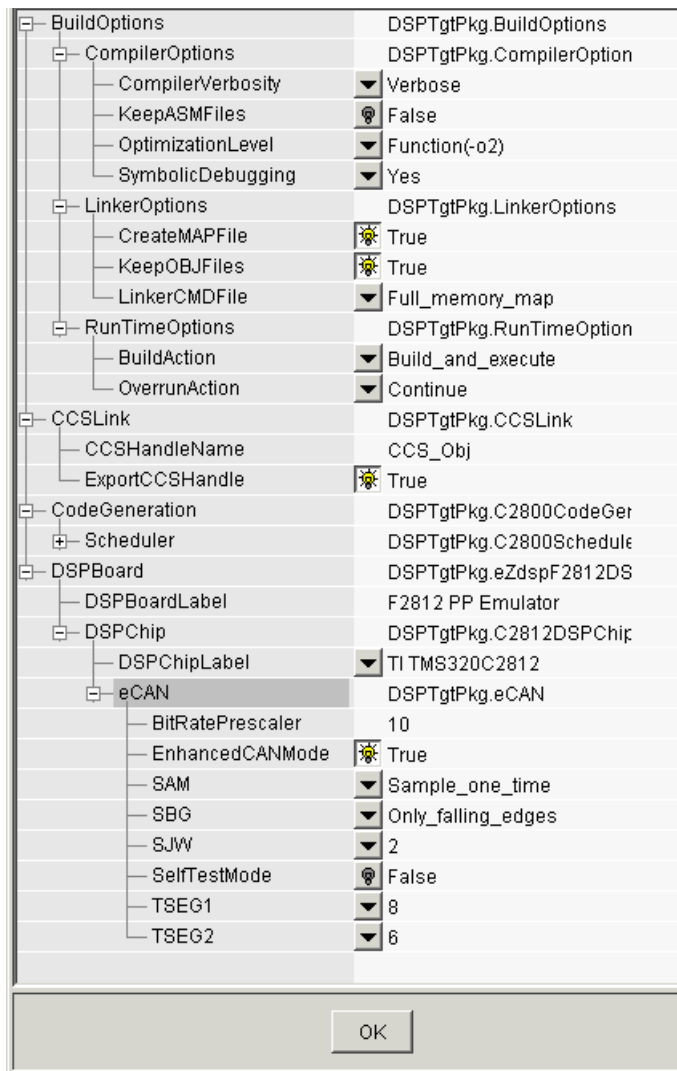
must be less than or equal to TSEG1 and greater than or equal to IPT.
Valid values for TSEG2 are from 1 through 8.

The eCAN bit timing is shown in the following illustration.



CAN Bit Timing

Dialog Box



See Also

C28x ADC, C28x eCAN Receive, C28x eCAN Transmit, C28x PWM

Float to IQN

Purpose Convert a floating-point number to an IQ number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

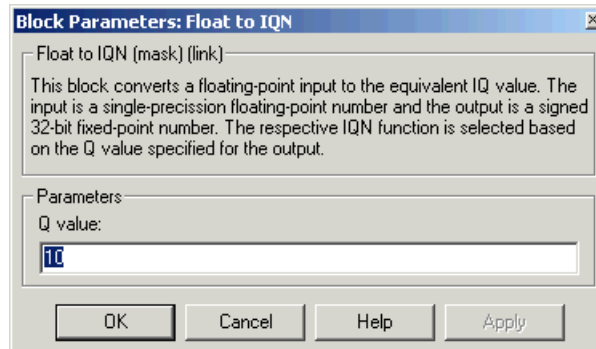
Description This block converts a floating-point number to an IQ number. The Q value of the output is specified in the following field:



Float to IQN

Q value — Q value from 1 to 30 that specifies the precision of the output

Dialog Box

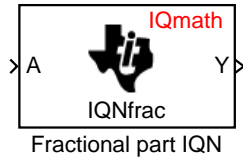


See Also Absolute IQN, Arctangent IQN, Division IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

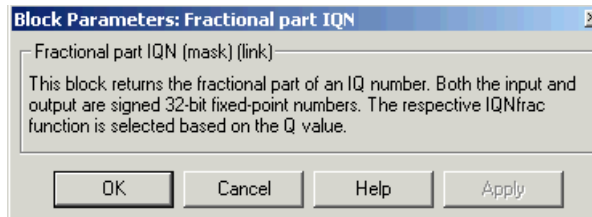
Purpose Return the fractional portion of an IQ number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block returns the fractional portion of an IQ number. The returned value is an IQ number in the same IQ format.



Dialog Box



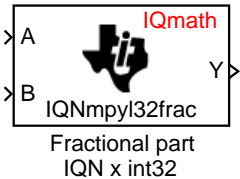
See Also Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Fractional part IQN x int32

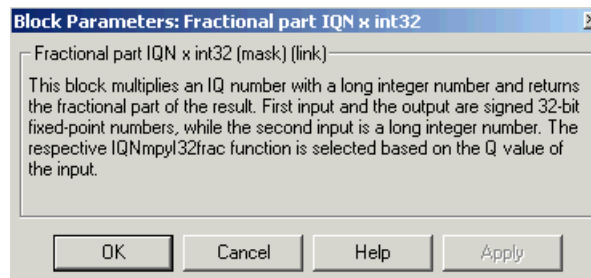
Purpose Multiply an IQ number with a long integer and return the fractional part of the result

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block multiplies an IQ input and a long integer input and returns the fractional portion of the resulting IQ number.



Dialog Box

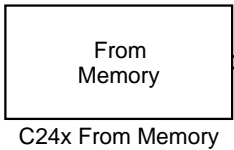


See Also Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Purpose Generate code that reads data from any valid memory address on the board, internal or external

Library c2400spchiplib or c2800spchiplib 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 icon and dialog box shown here are for the C24x, the same block and dialog box apply to the C28x.

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.

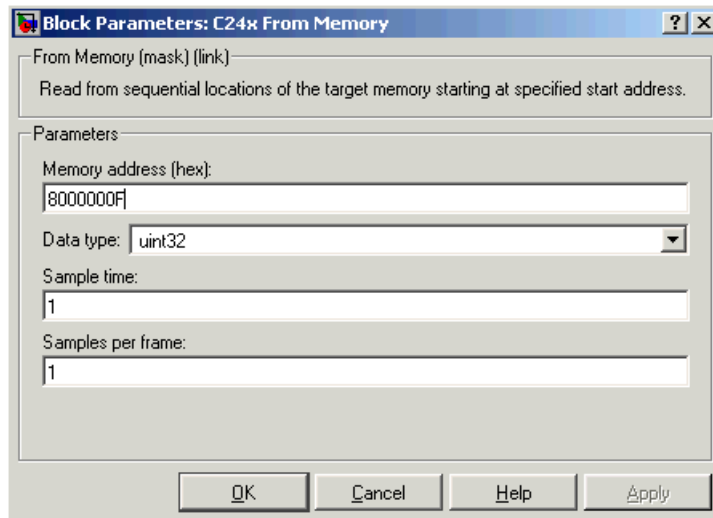
Data type — Data type of the data to obtain from the above memory address. The data is read as 16-bit data and then cast to the selected data type. Valid data types are double, single, int8, uint8, int16, uint16, int32, and uint32.

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.

From Memory

Dialog Box



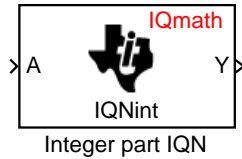
See Also

To Memory

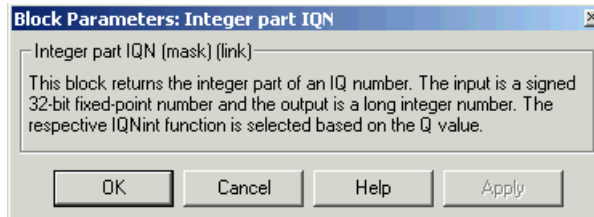
Purpose Return the integer portion of an IQ number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block returns the integer portion of an IQ number. The returned value is a long integer.



Dialog Box



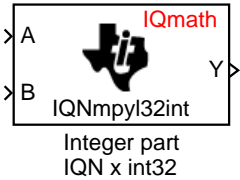
See Also Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Integer part IQN x int32

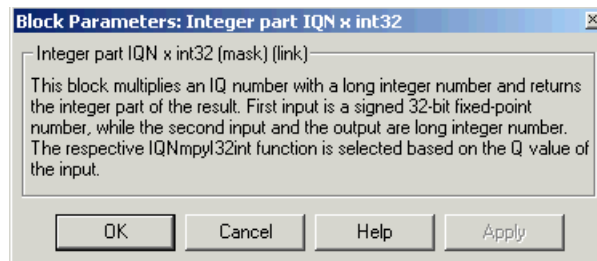
Purpose Multiply an IQ number with a long integer and return the integer part of the result

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block multiplies an IQ input and a long integer input and returns the integer portion of the resulting IQ number as a long integer.



Dialog Box



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

Inverse Park Transformation

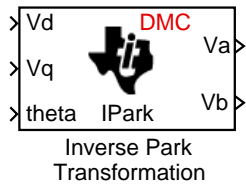
Purpose

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

Library

c28xdmclib in Embedded Target for TI C2000 DSP

Description

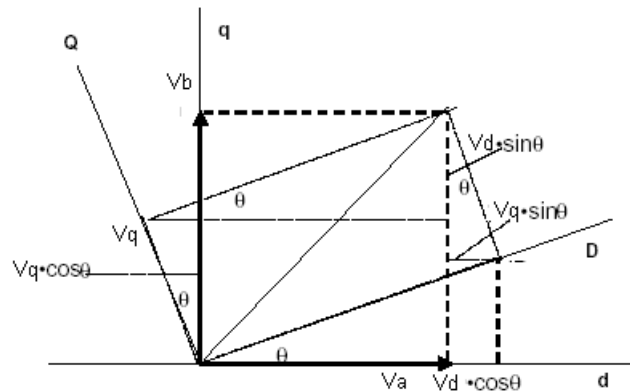


This block converts vectors in an orthogonal rotating reference frame to a two-phase orthogonal stationary reference frame. The transformation implements these equations

$$V_a = V_d \cos \theta - V_q \sin \theta$$

$$V_b = V_d \sin \theta + V_q \cos \theta$$

and is illustrated in the following figure.

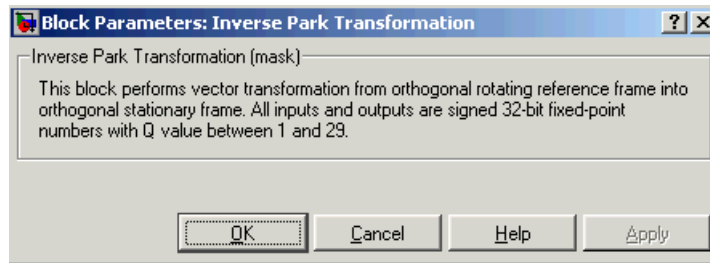


The inputs to this block are the direct axis (V_d) and quadrature axis (V_q) components of the transformed signal in the rotating frame and the phase angle (θ) between the stationary and rotating frames.

The outputs are the direct axis (V_a) and the quadrature axis (V_b) components of the transformed signal.

Inverse Park Transformation

Dialog Box



See Also

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

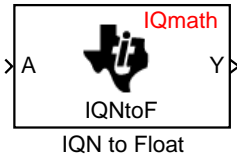
References

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.

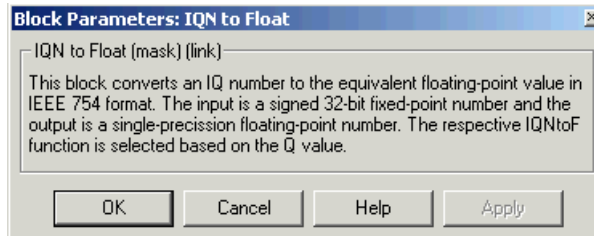
Purpose Convert an IQ number to a floating-point number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block converts an IQ input to an equivalent floating-point number. The output is a single floating-point number.



Dialog Box



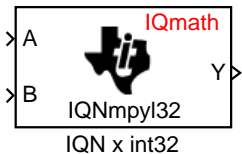
See Also Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

IQN x int32

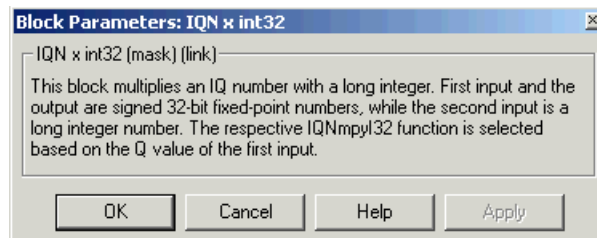
Purpose Multiply an IQ number with a long integer and return an IQ number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block multiplies an IQ input and a long integer input and produces an IQ output of the same Q value as the IQ input.



Dialog Box

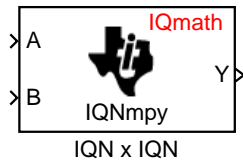


See Also Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Purpose Multiply two IQ numbers

Library tiiqmathlib in Embedded Target for TI C2000 DSP

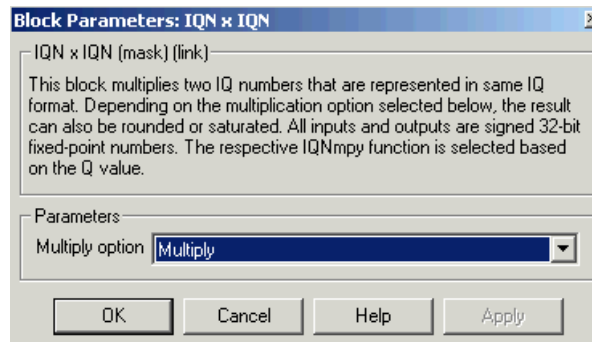
Description This block multiplies two IQ numbers. Optionally, it can also round and saturate the result.



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.

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, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

IQN1 to IQN2

Purpose Convert an IQ number to a different Q format

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

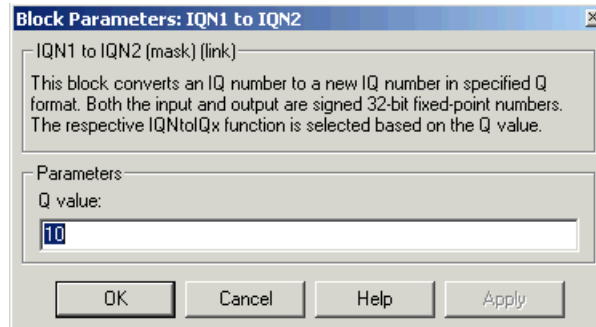
Description This block converts an IQ number in a particular Q format to a different Q format.



IQN1 to IQN2

Q value — Q value from 1 to 30 that specifies the precision of the output

Dialog Box

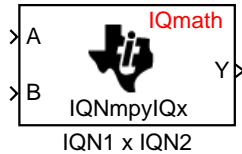


See Also Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Purpose Multiply two IQ numbers that are in different Q formats

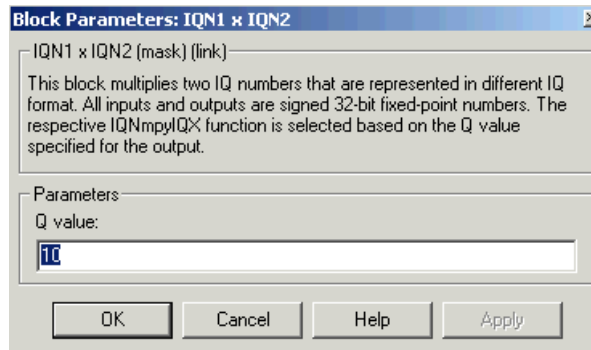
Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block multiplies two IQ numbers when the numbers are represented in different Q formats. The format of the result is specified in the following field:



Q value — Q value from 1 to 30 that specifies the precision of the output

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, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

LF2407 eZdsp

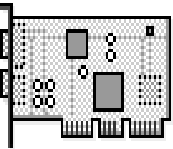
Purpose

Set the build, link, and board code generation preferences for LF2407 eZdsp DSK targets

Library

c2000tgtpreflib in Embedded Target for TI C2000 DSP

Description



LF2407 eZdsp

Options on the block mask let you set features of code generation for your Spectrum Digital LF2407 eZdsp target. Adding this block to your Simulink model provides access to building, linking, compiling, and targeting settings you need to configure the code that Real-Time Workshop generates.

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

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 true — .asm files are kept in your current directory. If you choose not to keep the .asm files, set this option to false.
- **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` — This option is not supported. 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 **File Name** field.

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. `Internal_memory_map` represents the most efficient memory use.

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**
 - **Timer** — Event manager (EV) timer to use for scheduling.
 - **TimerClockPrescaler** — Clock divider factor by which to prescale the selected timer to produce the desired model rate. The system clock for the TMS320LF2407 DSP is 40 MHz.

DSPBoard

- **DSPBoardLabel** — Name of the installed DSP board. Click in the edit box to change the label.

Note The board label here must match exactly the label (name) of the board entered in your Code Composer Studio setup.

- **DSPChip**

- **CAN** — Parameters that affect the control area network (CAN) module. Most of these parameters affect the 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

$$TQ = \frac{1}{SYSCLK} \cdot (BitRatePrescaler + 1)$$

where *SYSCLK* is the CAN module system clock frequency, and the *BitRatePrescaler* 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 settable parameters are

BitRatePrescaler — 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.

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.

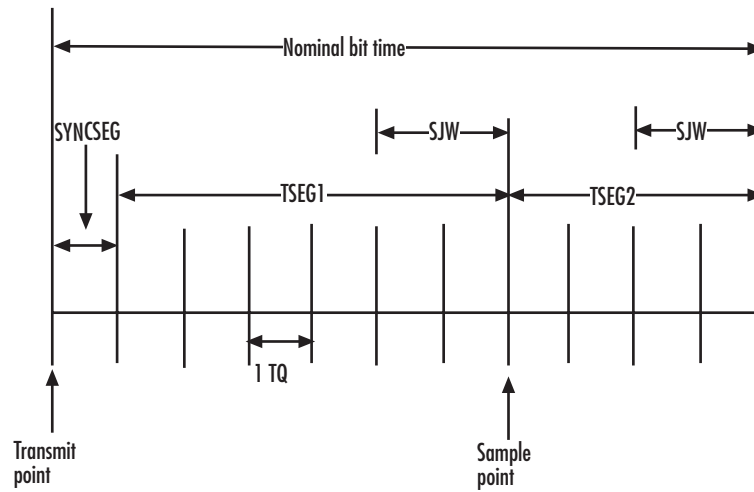
SelfTestMode — If `True`, sets the CAN module to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit.

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. $TSEG1 = PROP_SEG + PHASE_SEG1$. 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. **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.

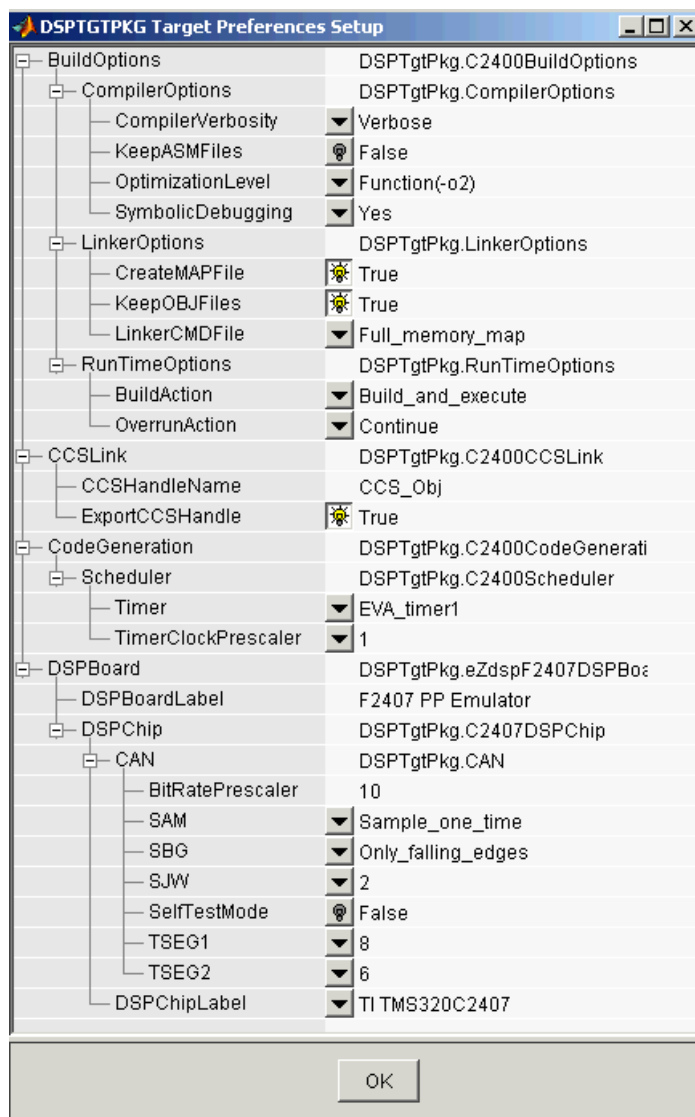
- **DSP Chip Label** — DSP chip model. Select the DSP chip installed on your target. The chip model is fixed for the LF2407 eZdsp. If you change the chip model, an error will be generated in code generation.

The CAN bit timing is shown in the following illustration.



CAN Bit Timing

Dialog Box



See Also

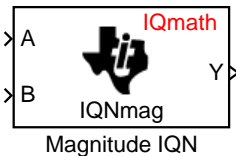
C24x ADC, C24x CAN Receive, C24x CAN Transmit, C24x PWM

Magnitude IQN

Purpose Calculate the magnitude of two IQ numbers

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

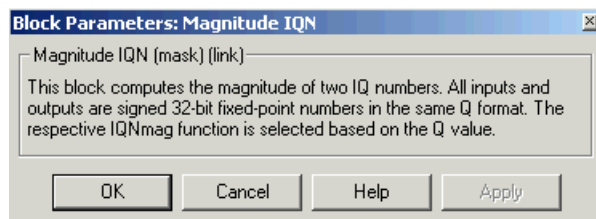
Description This block calculates the magnitude of two IQ numbers using



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

The output is an IQ number in the same Q format as the input.

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

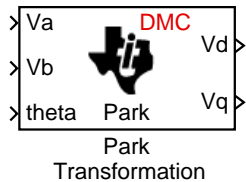
Purpose

Convert two-phase stationary system vectors to rotating system vectors

Library

c28xdmclib in Embedded Target for TI C2000 DSP

Description

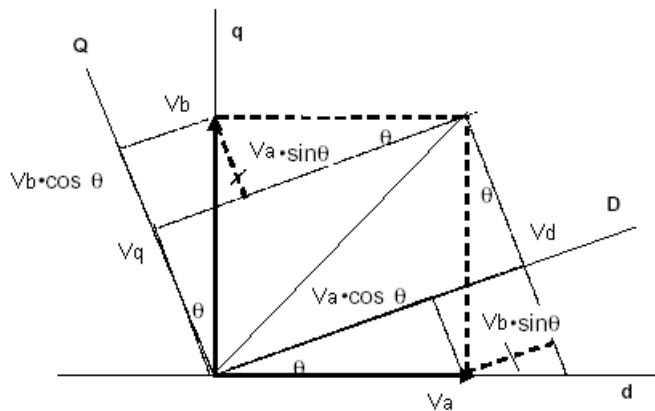


This block converts vectors in balanced two-phase orthogonal stationary systems into an orthogonal rotating reference frame. The transformation implements these equations

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

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

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 (θ) 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.

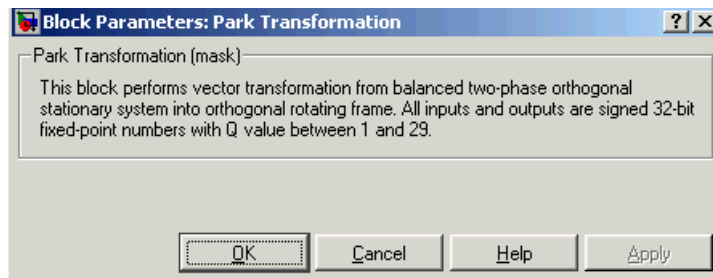
The instantaneous inputs are defined by the following equations.

$$id = I \sin(\omega t)$$

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

Park Transformation

Dialog Box



See Also

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

References

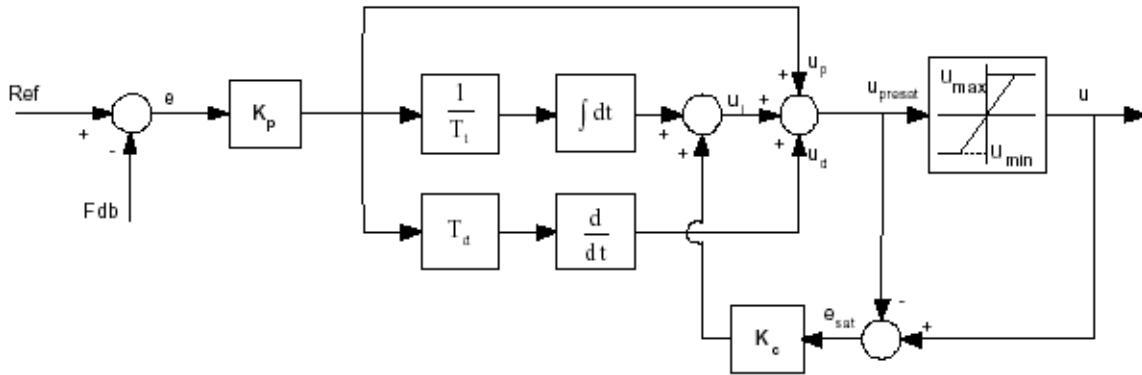
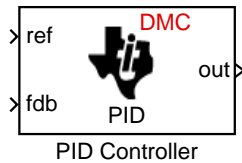
Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

Purpose Create a digital PID controller

Library c28xdmc1.lib in Embedded Target for TI C2000 DSP

Description

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



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

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

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

The proportional term is

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

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

PID Controller

The integral term with saturation correction is

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

where K_c is the integral correction gain of the 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.

Set the following values in the block 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

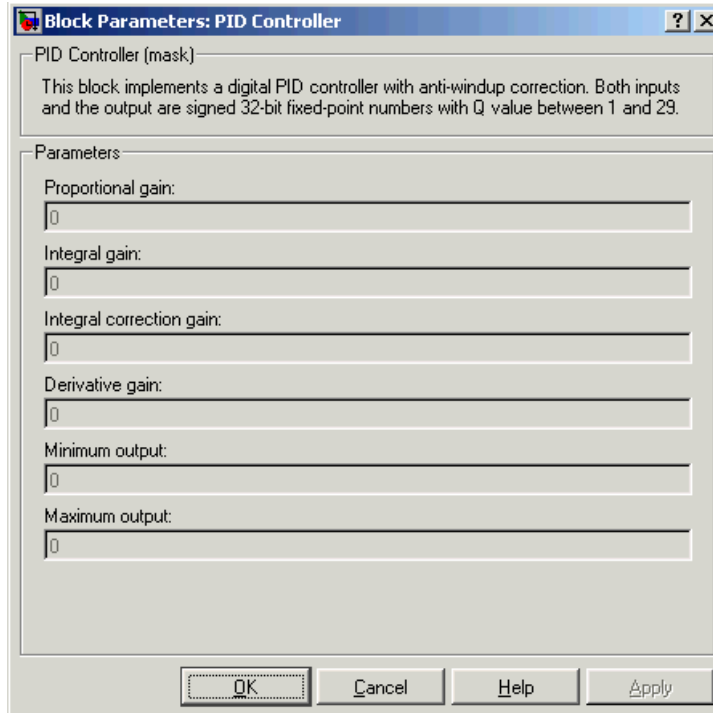
Integral correction gain — Amount of correction gain (K_c) to apply to the integration equation

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

Dialog Box



See Also

Clarke Transformation, Inverse Park Transformation, Park Transformation, Space Vector Generator, Speed Measurement

References

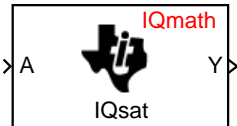
Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

Saturate IQN

Purpose Saturate an IQ value to specified limits

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block saturates an input IQ number to the specified positive and negative limits. The returned value is an IQ number of the same Q value as the input.

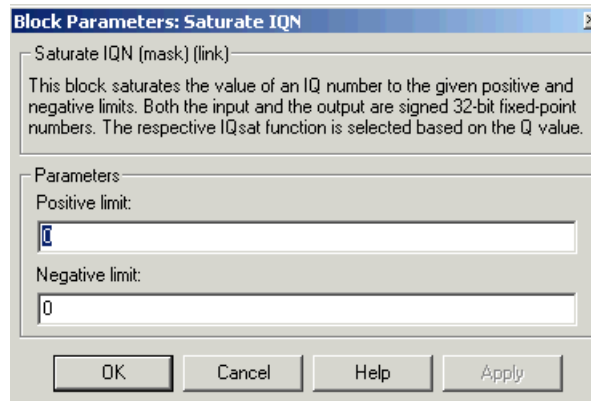


Saturate IQN

Positive Limit — Maximum positive value to which to saturate

Negative Limit — Minimum negative value to which to saturate

Dialog Box



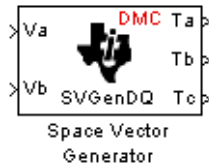
See Also

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, Magnitude IQN, Square Root IQN, Trig Fcn IQN

Purpose Calculate duty ratios to generate stator reference voltage

Library c28xdmc1.lib in Embedded Target for TI C2000 DSP

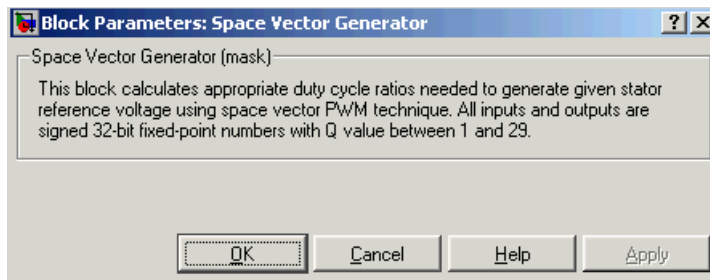
Description



This block calculates appropriate duty ratios needed to generate a given stator reference voltage using space vector PWM technique. Space vector pulse width modulation is a switching sequence of the upper three power devices of a three-phase voltage source inverter and is used in applications such as AC induction and permanent magnet synchronous motor drives. The switching scheme results in three pseudo-sinusoidal currents in the stator phases. This technique approximates a given stator reference voltage vector by combining the switching pattern corresponding to the basic space vectors.

The inputs to this block are the alpha component, which is the reference stator voltage vector on the direct axis stationary reference frame (V_a), and the beta component, which is the reference stator voltage vector on the direct axis quadrature reference frame (V_b). The alpha and beta components are transformed via the inverse Clarke equation and projected into reference phase voltages. These voltages are represented in the outputs as the duty ratios of the PWM1 (T_a), PWM3 (T_b), and PWM5 (T_c).

Dialog Box



See Also Clarke Transformation, Inverse Park Transformation, Park Transformation, PID Controller, Speed Measurement

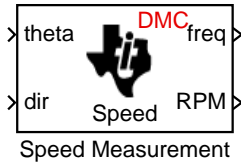
References Detailed information on the DMC library is in the *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

Speed Measurement

Purpose Calculate motor speed

Library c28xdmclib in Embedded Target for TI C2000 DSP

Description



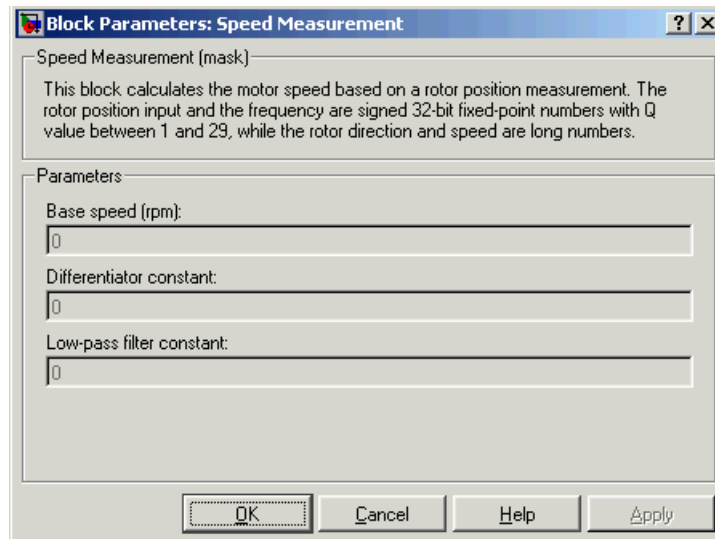
This block calculates the motor speed based on the rotor position when the direction information is available. The inputs are the electrical angle (θ) and the direction of rotation (dir) from the QEP encoder. The outputs are the speed in per-unit frequency ($freq$) and the speed in revolutions per minute (rpm).

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.

Dialog Box



See Also

Clarke Transformation, Inverse Park Transformation, Park Transformation, PID Controller, Space Vector Generator

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.

Square Root IQN

Purpose Calculate the square root or inverse square root

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

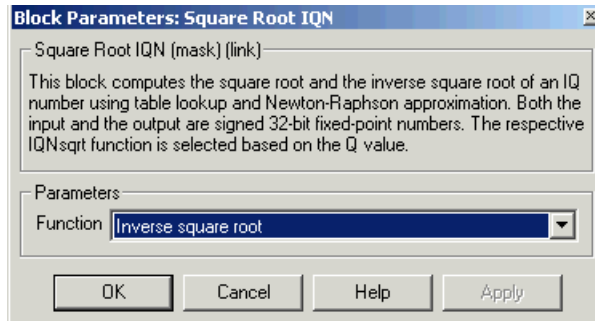


This block calculates the square root or inverse square root of an IQ number and returns an IQ number of the same Q format. The block uses table lookup and a Newton-Raphson approximation.

Function — Whether to calculate the square root or inverse square root

- Square root (`_sqr`) — Compute the square root.
- Inverse square root (`_isqr`) — Compute the inverse square root.

Dialog Box



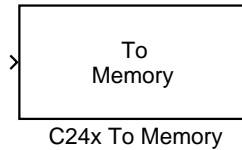
See Also

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, Magnitude IQN, Saturate IQN, Trig Fcn IQN

Purpose Generate code that writes data to any valid memory address on the board, internal or external

Library c2400dspchip1lib 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 icon and dialog box shown here are for the C24x, the same block and dialog box apply to the C28x.

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

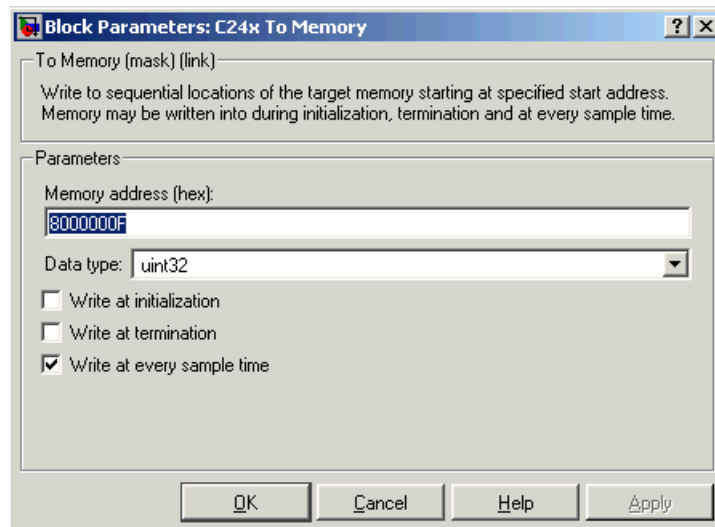
Write at every sample time — Whether to write data in real time during program execution

To Memory

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**.

Dialog Box



See Also

From Memory

Purpose Calculate the sine, cosine, or arc tangent

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

Description This block calculates basic trigonometric functions and returns the result as an IQ number. Valid Q values for `_IQsin` and `_IQcos` are 1 to 29. For all others, valid Q values are 1 to 30.

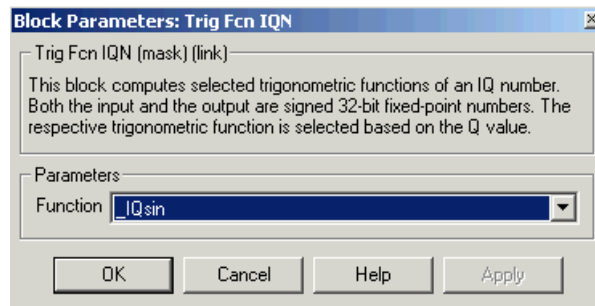


Trig Fcn IQN

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.

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, Magnitude IQN, Saturate IQN, Square Root IQN

Trig Fcn IQN

A

- Absolute IQN block 3-6
- ADC blocks
 - C24x 3-8
 - C28x 3-27
- analog-to-digital converter
 - See* ADC blocks
- applications
 - TI C2000 1-2
- Arctangent IQN block 3-7
- asymmetric vs. symmetric waveforms 3-41

B

- blocks
 - adding to model 1-23
 - recommendations 1-11
- build model 1-12
- build options 3-50

C

- c2000lib startup 1-14
- C24x ADC block 3-8
- C24x CAN Receive block 3-10
- C24x CAN Transmit block 3-12
- C24x GPIO Digital Input block 3-14
- C24x PWM block 3-20
- C24x QEP block 3-25, 3-45
- C24xGPIO Digital Output block 3-17
- C28x ADC block 3-27
- C28x eCAN Receive block 3-30
- C28x eCAN Transmit block 3-32
- C28x GPIO DIgital Input block 3-34
- C28x GPIO Digital Output block 3-37
- C28x PWM block 3-40

- CAN/eCAN Receive blocks
 - C24x 3-10
 - C28x 3-30
- CAN/eCAN timing 3-54
- CAN/eCAN Transmit blocks
 - C24x 3-12
 - C28x 3-32
- CCS

- link options 3-53
 - See also* Code Composer Studio

- Clarke Transformation block 3-47
- clock speed 1-8
- Code Composer Studio 1-7
 - projects 1-27
- code generation
 - options 3-53
 - overview 1-27
- code optimization 2-7
- compiler options 3-50
- configuration default 1-7
- control area network
 - See* CAN/eCAN
- control logic 3-22
- conversion
 - float to IQ number 3-58
 - IQ number to different IQ number 3-70
 - IQ number to float 3-67
- CPU clock speed 1-8

D

- data types
 - conversion 2-7
- deadband
 - C24x PWM 3-42
 - C28x PWM 3-22

default build configuration 1-7

digital motor control

See DMC library

Division IQN block 3-49

DMC library

Clarke Transformation 3-47

Inverse Park Transformation 3-65

Park Transformation 3-81

PID controller 3-83

Space Vector Generator 3-87

Speed Measurement 3-88

DSP board

target preferences options 3-53

duty ratios 3-87

E

event manager timer 3-20

F

F2812 eZdsp block 3-50

fixed-point numbers 2-3

Float to IQN block 3-58

floating-point numbers

convert to IQ number 3-58

four-quadrant arctangent 3-7

Fractional part IQN block 3-59

Fractional part IQN x int32 block 3-60

From Memory block 3-61

G

GPIO input 3-14, 3-34

GPIO output 3-17, 3-37

H

hardware 1-3

I

I/O 3-14, 3-17, 3-34, 3-37

Integer part IQN block 3-63

Integer part IQN x int32 block 3-64

Inverse Park Transformation block 3-65

IQ Math library 2-2

Absolute IQN block 3-6

Arctangent IQN block 3-7

building models 2-7

code optimization 2-7

common characteristics 2-2

Division IQN block 3-49

Float to IQN block 3-58

Fractional part IQN block 3-59

Fractional part IQN x int32 block 3-60

Integer part IQN block 3-63

Integer part IQN x int32 block 3-64

IQN to Float block 3-67

IQN x int32 block 3-68

IQN x IQN block 3-69

IQN1 to IQN2 block 3-70

IQN1 x IQN2 block 3-71

Magnitude IQN block 3-80

Q format notation 2-4

Saturate IQN block 3-86

Square Root IQN block 3-90

Trig Fcn IQN block 3-93

IQ numbers

convert from float 3-58

convert to different IQ 3-70

convert to float 3-67

fractional part 3-59

integer part 3-63

- magnitude 3-80
 - multiply by int32 fractional result 3-60
 - multiply 3-69
 - multiply by int32 3-68
 - multiply by int32 integer part 3-64
 - square root 3-90
 - trigonometric functions 3-93
 - IQN to Float block 3-67
 - IQN x int32 block 3-68
 - IQN x IQN block 3-69
 - IQN1 to IQN2 block 3-70
 - IQN1 x IQN2 block 3-71
- L**
- LF2407 eZdsp block 3-72
 - linker options 3-51
- M**
- Magnitude IQN block 3-80
 - mailbox 3-10
 - math blocks
 - See also* IQ Math library
 - MathWorks software 1-4
 - messages
 - F2812 eZdsp 3-30
 - LF2401 eZdsp 3-10
 - model
 - add blocks 1-23
 - building overview 1-12
 - creation overview 1-10
 - IQmath library 2-7
 - multiplication
 - IQN x int32 3-68
 - IQN x int32 fractional part 3-60
 - IQN x int32 integer part 3-64
 - IQN x IQN 3-69
 - IQN1 x IQN2 3-71
- O**
- operating system requirements 1-3
 - optimization code 2-7
- P**
- Park Transformation block 3-81
 - phase conversion 3-47
 - PID controller 3-83
 - prescaler 3-22
 - projects
 - CCS 1-27
 - pulse wave modulators
 - See* PWM blocks
 - pulse width 3-21
 - PWM blocks
 - C24x 3-20
 - C28x 3-40
 - control logic 3-22
 - deadband 3-22
- Q**
- Q format 2-4
 - quadrature encoder pulse circuit 3-25, 3-45
- R**
- Real Time Workshop build options
 - F2812 eZdsp 3-74
 - LF2407 eZdsp 3-52
 - receive 3-10
 - reference frame conversion 3-65, 3-81

reset 1-13
runtime options 3-52

S

sample time
 F2812 eZdsp 3-30
 LF2407 eZdsp 3-10
 maximum 1-8
Saturate IQN block 3-86
scheduling 1-8
setting up hardware 1-3
signed fixed-point numbers 2-3
simulation parameters 1-12
 automatic 1-17
software requirements 1-4
Space Vector Generator block 3-87
Speed Measurement block 3-88
Square Root IQN block 3-90
startup c2000lib 1-14

T

target configuration
 example 3-50
 LF2407 eZdsp 3-72
target model creation 1-10
target preferences
 compiler options 3-50
 DSP board options 3-53
 linker options 3-51
Target Preferences blocks
 F2812 eZdsp 3-50
 LF2407 eZdsp 3-72
TI software 1-4
timing 1-8
 CAN/eCAN 3-54

To Memory block 3-91
transmit 3-12
Trig Fcn IQN block 3-93

W

waveforms 3-41