Embedded Target for the TI TMS320C2000™ DSP Platform

For Use with Simulink®

User’s Guide
Version 1
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 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 or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as “commercial” computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government’s use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government’s minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

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+)
Contents

1

Getting Started

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 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 ................................ 1-24
  Generating Code from the Model ............................ 1-28
  Creating Code Composer Studio Projects Without Loading 1-29
Using the IQmath Library

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

Blocks — By Library ............................................. 3-2
  C2400 DSP Chip Support Library (c2400dspchiplib) .. 3-2
  C2800 DSP Chip Support Library (c2800dspchiplib) .. 3-2
  Target Preferences Library (c2000tgtpreflib) ........ 3-3
  C28x IQmath Library (tiiqmathlib) ..................... 3-3

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. The 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 that have any of the following characteristics:

- Fixed-point arithmetic
- Single rate
- Multirate
- Multistage
- 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

Spectrum Digital DSP Starter Kits (DSKs) help developers create 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](http://www.mathworks.com). Check the Product area for the Embedded Target for the TI TMS320C2000 DSP Platform.

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

<table>
<thead>
<tr>
<th>Installed Product</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td>Creates object code (.obj) for C2000 boards from assembly code</td>
</tr>
<tr>
<td>Compiler</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

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.
To verify that 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.

<table>
<thead>
<tr>
<th>Board Num</th>
<th>Board Name</th>
<th>Processor Num</th>
<th>Processor Name</th>
<th>Processor Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F2812 Simulator</td>
<td>0</td>
<td>CPU</td>
<td>TMS320C28xx</td>
</tr>
<tr>
<td>0</td>
<td>F2812 PP Emulator</td>
<td>0</td>
<td>CPU_1</td>
<td>TMS320C28xx</td>
</tr>
</tbody>
</table>

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 **Simulation 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 provide 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 MATLAB 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 MATLAB 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:

\[
\text{Base Rate Sample Time} = \frac{\text{Timer Period}}{(\text{CPU Clock Speed}) / \text{TimerClockPrescaler}}
\]

The minimum achievable base rate sample time depends on the model complexity. The maximum value depends on the maximum timer period value (\(2^{32} - 1\) for the F2812 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

<table>
<thead>
<tr>
<th>TimerClockPrescaler Setting</th>
<th>C24x Maximum Sample Time (seconds)</th>
<th>C28x Maximum Sample Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0016</td>
<td>0.0004</td>
</tr>
<tr>
<td>2</td>
<td>0.0032</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>0.0065</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>0.0131</td>
<td>N/A</td>
</tr>
<tr>
<td>16</td>
<td>0.0262</td>
<td>N/A</td>
</tr>
<tr>
<td>32</td>
<td>0.0524</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Maximum Sample Times (Continued)

<table>
<thead>
<tr>
<th>TimerClockPrescaler Setting</th>
<th>C24x Maximum Sample Time (seconds)</th>
<th>C28x Maximum Sample Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.1048</td>
<td>N/A</td>
</tr>
<tr>
<td>128</td>
<td>0.2097</td>
<td>N/A</td>
</tr>
</tbody>
</table>
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 Fixed-Point Blockset
- 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 TMS320C2000 DSP Platform.
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.

<table>
<thead>
<tr>
<th>Block Name/Category</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Simulink, DSP Blockset</td>
</tr>
<tr>
<td>To Workspace</td>
<td>Simulink</td>
</tr>
<tr>
<td>From Workspace</td>
<td>Simulink</td>
</tr>
<tr>
<td>Spectrum Scope</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>To File</td>
<td>Simulink</td>
</tr>
<tr>
<td>From File</td>
<td>Simulink</td>
</tr>
<tr>
<td>Triggered to Workspace</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>Signal To Workspace</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>Signal From Workspace</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>Triggered Signal From Workspace</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>To Wave Device</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>From Wave Device</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>To Wave File</td>
<td>DSP Blockset</td>
</tr>
<tr>
<td>From Wave File</td>
<td>DSP Blockset</td>
</tr>
</tbody>
</table>
Setting Simulation Parameters

To set the simulation parameters manually, with your model open, select `Simulation Parameters` from the `Simulink` option. From this dialog, click `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 file name 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 `Simulation Parameters` dialog. After you select `Fixed-step`, select `SingleTasking` as the `Mode` parameter. TI C2000 targets do not support multitasking.

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 `Simulation 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 to your TI development board by clicking `Build` on the `Real-Time Workshop` pane. Real-Time Workshop automatically generates C code and inserts the I/O device drivers as specified by the hardware blocks in your block diagram, if any. These device drivers are inserted in the generated C code as inlined S-functions. For information about inlining S-functions, refer to your target
language compiler documentation. For a complete discussion of S-functions, refer to your documentation about writing S-functions.

**Note** To build, load, and run code successfully on your target board, MATLAB must be able to locate that board in your system configuration. Make sure that the **Board Name** in your Code Composer Studio setup and the **DSPBoardLabel** in the Target 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.
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 (**c2800dsplib**) — Blocks to configure the codec on the F2812 eZdsp DSK or on the F2812 DSP
- C2400 DSP Core Support (**c2400dsplib**) — Blocks to configure the codec on the LF2407 eZdsp DSK or on the LF2407 DSP
- Target Preferences (**c2000tgtpreflib**) — Blocks to specify target preferences and options. You do not connect this block to any other block in your model.
- C28x IQMath Library (**ttiqmathlib**) — Math blocks for use with C2000 targets
- 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.

The following dialog appears, asking if you want preferences to be set automatically.
Click Yes to allow automatic setup. The following simulation parameters are set:

<table>
<thead>
<tr>
<th>Tab Page</th>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solver</td>
<td>Stop time</td>
<td>inf</td>
</tr>
<tr>
<td>Solver</td>
<td>Type</td>
<td>Fixed-step discrete</td>
</tr>
<tr>
<td>Workspace I/O</td>
<td>Save to workspace - Time</td>
<td>off</td>
</tr>
<tr>
<td>Workspace I/O</td>
<td>Save to workspace - Output</td>
<td>off</td>
</tr>
<tr>
<td>Advanced</td>
<td>Production hardware characteristics</td>
<td>Microprocessor</td>
</tr>
<tr>
<td>Advanced</td>
<td>Number of bits for C 'char'</td>
<td>16</td>
</tr>
<tr>
<td>Advanced</td>
<td>Number of bits for C 'short'</td>
<td>16</td>
</tr>
<tr>
<td>Advanced</td>
<td>Number of bits for C 'int'</td>
<td>16</td>
</tr>
<tr>
<td>Advanced</td>
<td>Number of bits for C 'long'</td>
<td>32</td>
</tr>
<tr>
<td>Real-Time Workshop</td>
<td>Target configuration - System target file</td>
<td>ti_c2000_grt.tlc</td>
</tr>
<tr>
<td>Real-Time Workshop</td>
<td>Target configuration - Template makefile</td>
<td>ti_c2000_grt.tmf</td>
</tr>
</tbody>
</table>

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 **Simulation 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 **Simulation Parameters** dialog.

Use the **Real-Time Workshop** pane of the **Simulation 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.

The **Real-Time Workshop** categories are

- **Target configuration** — Real-Time Workshop general configuration options
- **TLC debugging** — Real-Time Workshop general debugging options
- **General code generation options** — Real-Time Workshop general code generation options
- **ERT code generation options** — Target-specific run-time options
The ERT options apply to the embedded targets and appear when you select an ERT TI C2000 target, such as ti_c2000_ert.tlc, as the System target file. The ERT file requires Real-Time Workshop Embedded Coder.

**Target configuration** — Use the selections in this category to specify your target.

- **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.

- **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.

- **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.

- **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.

**TLC Debugging** — Real-Time Workshop uses the Target Language Compiler (TLC) to generate C code from your model1.rtw file. The TLC debugger helps you identify programming errors in your TLC code. Options in this pane are specific to TLC debugging. For details about using the options in TLC debugging, refer to the section “About TLC Debugger” in your Real-Time Workshop documentation.

For this example, do not select any of the TLC Debugging options.
General code generation options — Real-Time Workshop uses the general code generation options during the build process to tailor the generated code to your needs. For more information about using these General code generation options, refer to your Real-Time Workshop documentation.

For this example, use the default settings of the General code generation options.

ERT code generation options — Real Time Workshop uses the ERT code generation options to customize your embedded real-time code generation. See the Real-Time Workshop Embedded Coder documentation for information.

For this example, use the default settings of the ERT code generation options. Verify that the following options, which apply to the C2000 targets, are set appropriately:

- Integer code only — Check this box to ensure that no floating-point data is used, because the C2000 targets do not support floating-point data. Using floating-point data on a C2000 target generates an error.
- Generate an example main program — This box does not affect the Embedded Target for C2000 product because it generates its own main program.
- Target floating-point math environment — Verify that this box is not checked. It does not apply to the Embedded Target for TI C2000 DSP.

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.

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler Options</td>
<td>CompilerVerbosity</td>
<td>Verbose</td>
</tr>
<tr>
<td></td>
<td>KeepASMFiles</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>OptimizationLevel</td>
<td>Function(-o2)</td>
</tr>
<tr>
<td></td>
<td>SymbolicDebugging</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Build Options (Continued)

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linker Options</td>
<td>CreateMAPFile</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>KeepOBJFiles</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>LinkerCMDFile</td>
<td>Full_memory_map</td>
</tr>
<tr>
<td>RunTime Options</td>
<td>BuildAction</td>
<td>Build_and_execute</td>
</tr>
<tr>
<td></td>
<td>OverrunAction</td>
<td>Continue</td>
</tr>
</tbody>
</table>

### CCSLink Options

<table>
<thead>
<tr>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCSHandleName</td>
<td>CCS_Obj</td>
</tr>
<tr>
<td>ExportCCSHandle</td>
<td>True</td>
</tr>
</tbody>
</table>

### CodeGeneration Options

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduler</td>
<td>Timer</td>
<td>CPU_timer0</td>
</tr>
<tr>
<td></td>
<td>TimerClockPrescaler</td>
<td>1</td>
</tr>
</tbody>
</table>

### DSPBoard Options

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP Board Label</td>
<td>DSPBoardLabel</td>
<td>F2812_PP_Emulator</td>
</tr>
<tr>
<td>DSP Chip</td>
<td>DSPChipLabel</td>
<td>TI_TMS320C2812</td>
</tr>
</tbody>
</table>
## DSPBoard Options (Continued)

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCAN</td>
<td>BitRatePrescaler</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>EnhancedCANMode</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>SAM</td>
<td>Sample_one_time</td>
</tr>
<tr>
<td></td>
<td>SBJ</td>
<td>Only_falling_edges</td>
</tr>
<tr>
<td></td>
<td>SJW</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SelfTestMode</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>TSEG1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>TSEG2</td>
<td>6</td>
</tr>
</tbody>
</table>
Using the c2000lib Blockset

![DSPTargetkg Target Preferences Setup](image)

### BuildOptions
- **DSPTgPkg.BuildOptions**

### CompilerOptions
- **DSPTgPkg.CompilerOptions**
  - **-CompileVerbose**
  - **-KeepASMFiles**
  - **-OptimizationLevel**
  - **-SymbolicDebugging**

### LinkerOptions
- **DSPTgPkg.LinkerOptions**
  - **-CreateMAPFile**
  - **-KeepOBJFiles**
  - **-LinkerCMDFile**

### RunTimeOptions
- **DSPTgPkg.RuntimeOptions**
  - **-BuildAction**
  - **-OverrunAction**

### CCELink
- **DSPTgPkg.CCELink**
  - **-CCSHandleName**
  - **-ExportCCSHandle**

### CodeGeneration
- **DSPTgPkg.C2000CodeGen**
  - **-Scheduling**

### DSPBoard
- **DSPTgPkg.DSPBoard**
  - **-DSPBoardLabel**
  - **-DSPChip**
    - **-DSPChipLabel**

### eCAN
- **DSPTgPkg.eCAN**
  - **-BLIRatePrescaler**
  - **-EnhancedCANMode**
  - **-SAM**
  - **-SBG**
  - **-SJW**
  - **-SoftTestMode**
  - **-TSEG1**
  - **-TSEG2**

**OK**
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.
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>A</td>
</tr>
<tr>
<td>Waveform period source</td>
<td>Specify via dialog</td>
</tr>
<tr>
<td>Waveform period</td>
<td>64000</td>
</tr>
<tr>
<td>Waveform type</td>
<td>Asymmetric</td>
</tr>
<tr>
<td>Enable PWM1/PWM2</td>
<td>selected</td>
</tr>
<tr>
<td>Pulse width source</td>
<td>Input port</td>
</tr>
<tr>
<td>Show additional parameters</td>
<td>selected</td>
</tr>
<tr>
<td>Field</td>
<td>Parameter</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>PWM1 control logic</td>
<td>Active high</td>
</tr>
<tr>
<td>PWM2 control logic</td>
<td>Active low</td>
</tr>
<tr>
<td>Use deadband for PWM1/PWM2</td>
<td>selected</td>
</tr>
<tr>
<td>Deadband prescaler</td>
<td>16</td>
</tr>
<tr>
<td>Deadband period</td>
<td>12</td>
</tr>
</tbody>
</table>

### Block Parameters C28x PWM

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
</tr>
<tr>
<td>Waveform period source</td>
</tr>
<tr>
<td>Waveform period [clock cycles]</td>
</tr>
<tr>
<td>Waveform type</td>
</tr>
<tr>
<td>Enable PWM1/PWM2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Enable PWM3/PWM4</td>
</tr>
<tr>
<td>Enable PWM5/PWM6</td>
</tr>
<tr>
<td>Show additional parameters</td>
</tr>
<tr>
<td>PWM1 control logic</td>
</tr>
<tr>
<td>PWM2 control logic</td>
</tr>
<tr>
<td>Use deadband for PWM1/PWM2</td>
</tr>
<tr>
<td>Deadband prescaler</td>
</tr>
<tr>
<td>Deadband period</td>
</tr>
</tbody>
</table>

---

1-26
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain</td>
<td>30</td>
</tr>
<tr>
<td>Multiplication</td>
<td>Element-wise(K.*u)</td>
</tr>
<tr>
<td>Show additional parameters</td>
<td>selected</td>
</tr>
<tr>
<td>Parameter data type mode</td>
<td>Same as input</td>
</tr>
<tr>
<td>Output data type mode</td>
<td>Specify via dialog</td>
</tr>
<tr>
<td>Output data type</td>
<td>uint(16)</td>
</tr>
<tr>
<td>Round integer calculations</td>
<td>Floor</td>
</tr>
<tr>
<td>Sample time</td>
<td>-1</td>
</tr>
</tbody>
</table>
5 Connect the ADC block to the Gain block and the Gain block to the PWM block as shown.

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.

1-28
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.
1 Getting Started
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 thorough discussion of this topic, including issues with scaling and precision when performing fixed-point operations, refer to your Fixed-Point Blockset documentation.

You can use IQmath Library blocks with certain core Simulink blocks, as well as with certain blocks from the Fixed-Point Blockset. 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.

\[
\begin{array}{cccccccc}
  b_{ws-1} & b_{ws-2} & \cdots & b_3 & b_4 & b_3 & b_2 & b_1 & b_0 \\
\end{array}
\]

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:

\[ \begin{align*}
000101 & \rightarrow 111010 \text{ (bit inversion)} \rightarrow 111011 \text{ (binary addition of a 1 to the LSB)}
\end{align*} \]

**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

\[ 00.15 \]
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 the Fixed-Point Blockset, this data type is expressed as

sfrac16

or

sfix16_En15

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

[16 15]

**Example — Q1.30**

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

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

Therefore this number is expressed as

Q1.30

In the Fixed-Point Blockset, 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 the Fixed-Point Blockset, this data type is expressed as

\[
sfix16\_En17
\]

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

\[
[16 \ 17]
\]

**Example — \( Q^{17.-2} \)**

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

\[
Q^{17.-2}
\]

In the Fixed-Point Blockset, this data type is expressed as

\[
sfix16\_E2
\]

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

\[
[16 \ -2]
\]
Building Models

You can use IQmath Library blocks in models along with certain core Simulink, Fixed-Point Blockset, 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 Fixed-Point Blockset blocks, you often need to set the data type and scaling in the block parameters of the Fixed-Point Blockset block to match the data type of the IQmath Library block. Many Fixed-Point Blockset 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 Fixed-Point Blockset block to match a connected IQmath Library block.

Some DSP 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 Fixed-Point Blockset 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 the Fixed-Point Blockset 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 — By Library (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 — By Library

#### C2400 DSP Chip Support Library (c2400dsphiplib)

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C24x ADC</td>
<td>Configure analog to digital converters (ADC)</td>
</tr>
<tr>
<td>C24x CAN Receive</td>
<td>Configure enhanced Control Area Network receive mailbox</td>
</tr>
<tr>
<td>C24x CAN Transmit</td>
<td>Configure enhanced Control Area Network transmit mailbox</td>
</tr>
<tr>
<td>C24x PWM</td>
<td>Configure one or more pairs of pulse wave modulators (PWMs)</td>
</tr>
<tr>
<td>C24x From Memory</td>
<td>Retrieve data from specific memory location on the target</td>
</tr>
<tr>
<td>C24x To Memory</td>
<td>Write data to the specific memory location on the target</td>
</tr>
</tbody>
</table>

#### C2800 DSP Chip Support Library (c2800dsphiplib)

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C28x ADC</td>
<td>Configure analog to digital converters (ADC)</td>
</tr>
<tr>
<td>C28x eCAN Receive</td>
<td>Configure enhanced Control Area Network receive mailbox</td>
</tr>
<tr>
<td>C28x eCAN Transmit</td>
<td>Configure enhanced Control Area Network transmit mailbox</td>
</tr>
<tr>
<td>C28x PWM</td>
<td>Configure one or more pairs of pulse wave modulators (PWMs)</td>
</tr>
</tbody>
</table>
### Blocks — By Library

<table>
<thead>
<tr>
<th>Block Description</th>
<th>Target Preferences Library (c2000tgtpreflib)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C28x From Memory</td>
<td>Preferences for F2812 eZdsp™ DSK targets</td>
</tr>
<tr>
<td>C28x To Memory</td>
<td>Preferences for LF2407 eZdsp™ DSK targets</td>
</tr>
</tbody>
</table>

### C28x IQmath Library (tiiqmathlib)

<table>
<thead>
<tr>
<th>Block Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute IQN</td>
<td>Calculate absolute value</td>
</tr>
<tr>
<td>Arctangent IQN</td>
<td>Calculate four-quadrant arc tangent</td>
</tr>
<tr>
<td>Division IQN</td>
<td>Divide two IQ numbers</td>
</tr>
<tr>
<td>Float to IQN</td>
<td>Convert a floating-point number to an IQ number</td>
</tr>
<tr>
<td>Fractional part IQN</td>
<td>Return the fractional part of an IQ number</td>
</tr>
<tr>
<td>Fractional part IQN x int32</td>
<td>Return the fractional part of the result of multiplying an IQ number and a long integer</td>
</tr>
<tr>
<td>Integer part IQN</td>
<td>Return the integer part of an IQ number</td>
</tr>
<tr>
<td>Integer part IQN x int32</td>
<td>Return the integer part of the result of multiplying an IQ number and a long integer</td>
</tr>
<tr>
<td>Block</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>IQN to Float</td>
<td>Convert an IQ number to a floating-point number</td>
</tr>
<tr>
<td>IQN x int32</td>
<td>Multiply an IQ number and a long integer</td>
</tr>
<tr>
<td>IQN x IQN</td>
<td>Multiply two IQ numbers with the same Q format</td>
</tr>
<tr>
<td>IQN1 to IQN2</td>
<td>Convert an IQ number to a different Q format</td>
</tr>
<tr>
<td>IQN1 x IQN2</td>
<td>Multiply two IQ numbers with different Q formats</td>
</tr>
<tr>
<td>Magnitude IQN</td>
<td>Calculate the magnitude of two orthogonal IQ numbers</td>
</tr>
<tr>
<td>Saturate IQN</td>
<td>Saturate an IQ number</td>
</tr>
<tr>
<td>Square Root IQN</td>
<td>Calculate the square root or inverse square root of an IQ number</td>
</tr>
<tr>
<td>Trig Fcn IQN</td>
<td>Calculate the sine, cosine, or tangent of an IQ number</td>
</tr>
</tbody>
</table>
## Blocks — Alphabetical List

<table>
<thead>
<tr>
<th>Block</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute IQN</td>
<td>3-7</td>
</tr>
<tr>
<td>Arctangent IQN</td>
<td>3-8</td>
</tr>
<tr>
<td>C24x ADC</td>
<td>3-9</td>
</tr>
<tr>
<td>C24x CAN Receive</td>
<td>3-11</td>
</tr>
<tr>
<td>C24x CAN Transmit</td>
<td>3-13</td>
</tr>
<tr>
<td>C24x From Memory</td>
<td>3-15</td>
</tr>
<tr>
<td>C24x PWM</td>
<td>3-16</td>
</tr>
<tr>
<td>C24x To Memory</td>
<td>3-20</td>
</tr>
<tr>
<td>C28x ADC</td>
<td>3-22</td>
</tr>
<tr>
<td>C28x eCAN Receive</td>
<td>3-24</td>
</tr>
<tr>
<td>C28x eCAN Transmit</td>
<td>3-26</td>
</tr>
<tr>
<td>C28x From Memory</td>
<td>3-28</td>
</tr>
<tr>
<td>C28x PWM</td>
<td>3-29</td>
</tr>
<tr>
<td>C28x To Memory</td>
<td>3-33</td>
</tr>
<tr>
<td>Division IQN</td>
<td>3-35</td>
</tr>
<tr>
<td>F2812 eZdsp</td>
<td>3-36</td>
</tr>
<tr>
<td>Float to IQN</td>
<td>3-44</td>
</tr>
<tr>
<td>Fractional part IQN</td>
<td>3-45</td>
</tr>
<tr>
<td>Fractional part IQN x int32</td>
<td>3-46</td>
</tr>
<tr>
<td>Integer part IQN</td>
<td>3-47</td>
</tr>
<tr>
<td>Integer part IQN x int32</td>
<td>3-48</td>
</tr>
<tr>
<td>IQN to Float</td>
<td>3-49</td>
</tr>
<tr>
<td>IQN x int32</td>
<td>3-50</td>
</tr>
<tr>
<td>IQN x IQN</td>
<td>3-51</td>
</tr>
<tr>
<td>IQN1 to IQN2</td>
<td>3-52</td>
</tr>
<tr>
<td>IQN1 x IQN2</td>
<td>3-53</td>
</tr>
<tr>
<td>LF2407 eZdsp</td>
<td>3-54</td>
</tr>
<tr>
<td>Magnitude IQN</td>
<td>3-62</td>
</tr>
<tr>
<td>Saturate IQN</td>
<td>3-63</td>
</tr>
<tr>
<td>Square Root IQN</td>
<td>3-64</td>
</tr>
<tr>
<td>Trig Fcn IQN</td>
<td>3-65</td>
</tr>
</tbody>
</table>
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
Arctangent 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 option — Type of arc tangent to calculate, either
- atan2 — Compute the four-quadrant arc tangent with output in radians with values between -pi and +pi.
- atan2PU — Compute the four-quadrant arc tangent per unit. If atan2(B,A) is greater than or equal to zero, atan2PU(B,A) = atan2(B,A)/2*pi. Otherwise, atan2PU(B,A) = atan2(B,A)/2*pi+1. The output is in per-unit radians with values from 0 to 2pi radians.

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
Purpose

Generate code to configure the C24x analog-to-digital converter
c2400dspchiplib 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. It 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.

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.

The C24x ADC block supports ADC operation in dual-sequence 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.
C24x ADC

Dialog Box

See Also

C24x PWM
**Purpose**

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

**Library**

c2400dspchiplib 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.

- 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.
- 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.
- 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).
C24x CAN Receive

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 `unit32`.

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
C24x CAN Transmit

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

Library

c2400dspchiplib 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.

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.

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).
C24x CAN Transmit

Dialog Box

![Dialog Box](image)

See Also

C24x CAN Receive
Purpose
Generate code that retrieves data from any valid memory address on the board, internal or external.

Library
C2400dspchiplib 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.

- **Memory address** — Address of the target memory location, in hexadecimal, from which to read data.
- **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.

Dialog Box

![Block Parameters: C24x From Memory](image)

See Also
C24x To Memory
C24x PWM

Purpose
Generate code that configures the event manager (EV) modules to generate PWM waveforms

Library
Library c2400dspchiplib in Embedded Target for TI C2000 DSP

Description
LF2407 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.

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 the 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.

Enable PWM#/PWM# — Check to select the PWM pairs to activate

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.

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. You can change the PWM control logic by selecting Show additional parameters.
Show additional parameters — Check to display the dialog box with additional PWM parameters

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.

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.
Dialog Box

See Also

C24x ADC
**C24x To Memory**

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

**Library**
c2400dspchiplib 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.

- **Memory address** — Address of the target memory location, in hexadecimal, to which to write data
- **Data type** — Type of data to be written to the above memory address. Valid data types are double, single, int8, uint8, int16, uint16, int32, and uint32. The data is cast from the selected data type to 16-bit data.
- **Write at initialization** — Whether to write the specified **Value** at program start
  - **Value** — First value of data to be written to memory at program start
- **Write at termination** — Whether to write the specified **Value** at program end
  - **Value** — Last value of data to be written to memory at program termination
- **Write at every sample time** — Whether to write data in real time during program execution

**Note**
If your C24x 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 C24x 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**.
C24x To Memory

Dialog Box

![Dialog Box Image]

See Also

C24x From Memory
**C28x ADC**

**Purpose**
Generate code to configure the ADC to output data streams

**Library**
c2800dspchiplib 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. It 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.

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.

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

C28x ADC

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**
c2800dspchiplib 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.

- 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.
- 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.
- Third output port is optional and appears only if **Output message length** is selected.

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.

**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
**C28x eCAN Transmit**

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

**Library**
c2800dspchiplib 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.

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.

**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**

![Block Parameters: C28x eCAN Transmit](image)

- **Mailbox number:**
  - Configures an eCAN mailbox to transmit message to the CAN bus pins on the C28x DSP.
  - Example: Mailbox number: 1

- **Message identifier:**
  - Identifier of length 11 bits for standard frame size or length 29 bits for extended frame size in decimal, binary, or hex.
  - Example: `bin2dec('11100111')` or `hex2dec('11')`

- **Message type:**
  - Select Standard (11-bit identifier) or Extended (29-bit identifier).
See Also

C28x eCAN Receive
C28x From Memory

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

Library
c2800dspchiplib 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.

- **Memory address** — Address of the target memory location, in hexadecimal, from which to read data
- **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 datatype to be read from the memory region starting at the given address

Dialog Box

![Block Parameters: C28x From Memory](image)

See Also
C28x To Memory
### Purpose
Generate code that configures the event manager (EV) modules to generate PWM waveforms.

### Library
`c2800dspchiplib` 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.

**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 the 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.

**Use PWM#/PWM#** — Check to select the PWM pairs to activate

**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.

**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. You can change the PWM control logic by selecting **Show additional parameters**.
Show additional parameters — Check to display the dialog box with additional PWM parameters

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.

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.
C28x PWM

Dialog Box

See Also

C28x ADC
C28x To Memory

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

**Library**
c2800dspchiplib 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.

- **Memory address** — Address of the target memory location, in hexadecimal, to which to write data.
- **Data type** — Type of data to be written to the above memory address. Valid data types are `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`. The data is cast from the selected data type to 16-bit data.
- **Write at initialization** — Whether to write the specified `Value` at program start.
- **Value** — First value of data to be written to memory at program start.
- **Write at termination** — Whether to write the specified `Value` at program end.
- **Value** — Last value of data to be written to memory at program termination.
- **Write at every sample time** — Whether to write data in real time during program execution.

**Note**
If your C28x 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 C28x 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**.
C28x To Memory

Dialog Box

![Dialog Box Image]

See Also

C28x From Memory
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
Block Parameters: IQN / IQN
- Division IQN (mask) link
  This block divides two IQN numbers using Newton-Raphson technique. All inputs and outputs are signed 32-bit fixed point numbers that have the same Q value. The respective IQN division function is selected based on the Q value.

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
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
- **Keep ASM Files** — 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.
- **Optimization Level** — 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)`.
- **Symbolic Debugging** — 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

- **Create MAP File** — 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.

- **Keep OBJ Files** — 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.

- **Linker CMD File** — 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.

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**

- **Build Action** — 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.

- **Overrun Action** — 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**

- **CCS Handle Name** — 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.

- **Export CCS Handle** — Whether to export the CCS handle to your MATLAB workspace, giving it the name you assigned in **CCS Handle Name**. 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**

- **DSP Board Label** — 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.
DSP Chip

- **DSP Chip Label** — 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

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

    where SYSCLK is the CAN module system clock frequency, and the \textit{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.

- **EnhancedCANMode** — Whether to use the CAN module in extended mode, which provides additional mailboxes and time stamping. The default is \texttt{True}. Setting this parameter to \texttt{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.
F2812 eZdsp

CAN Bit Timing

Nominal bit time

SYNCSEG

TSEG1

TSEG2

1 TQ

Transmit point

Sample point
### Dialog Box

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildOptions</td>
<td>DSPTgiPkg.BuildOptions</td>
</tr>
<tr>
<td>CompilerOptions</td>
<td>DSPTgiPkg.CompilerOption</td>
</tr>
<tr>
<td>CompilerVerbose</td>
<td>Verbose</td>
</tr>
<tr>
<td>KeepASMFiles</td>
<td>False</td>
</tr>
<tr>
<td>OptimizationLevel</td>
<td>Function (O2)</td>
</tr>
<tr>
<td>SymbolsDebugging</td>
<td>Yes</td>
</tr>
<tr>
<td>LinkerOptions</td>
<td>DSPTgiPkg.LinkerOptions</td>
</tr>
<tr>
<td>CreateMAPFile</td>
<td>True</td>
</tr>
<tr>
<td>KeepOBJFiles</td>
<td>True</td>
</tr>
<tr>
<td>LinkGCMDFile</td>
<td>Full_memory.rag</td>
</tr>
<tr>
<td>RunTimeOptions</td>
<td>DSPTgiPkg.RuntimeOption</td>
</tr>
<tr>
<td>BUILDAction</td>
<td>Build_and_execute</td>
</tr>
<tr>
<td>OVERrunAction</td>
<td>Contin</td>
</tr>
<tr>
<td>CCSLink</td>
<td>DSPTgiPkg.CCSLink</td>
</tr>
<tr>
<td>CCBHandleName</td>
<td>CCB_Obj</td>
</tr>
<tr>
<td>ExportCCSHandle</td>
<td>True</td>
</tr>
<tr>
<td>CodeGeneration</td>
<td>DSPTgiPkg.C2800CodeG01</td>
</tr>
<tr>
<td>Scheduler</td>
<td>DSPTgiPkg.C2800Schedule</td>
</tr>
<tr>
<td>DSPBoard</td>
<td>DSPTgiPkg.eZdspF2812DB</td>
</tr>
<tr>
<td>DSPBoardLabel</td>
<td>F2812 PP Emulator</td>
</tr>
<tr>
<td>DSPChip</td>
<td>DSPTgiPkg.TMS320C2812</td>
</tr>
<tr>
<td>DSPChipLabel</td>
<td>TI TMS320C2812</td>
</tr>
<tr>
<td>SCAN</td>
<td>DSPTgiPkg.eCAN</td>
</tr>
<tr>
<td>BitRatePrescaler</td>
<td>10</td>
</tr>
<tr>
<td>EnhancedCANMode</td>
<td>True</td>
</tr>
<tr>
<td>SAM</td>
<td>Sample_ora_time</td>
</tr>
<tr>
<td>SBG</td>
<td>Only_falling_eigen</td>
</tr>
<tr>
<td>BJW</td>
<td>2</td>
</tr>
<tr>
<td>SelfTestMode</td>
<td>False</td>
</tr>
<tr>
<td>TSEG1</td>
<td>8</td>
</tr>
<tr>
<td>TSEG2</td>
<td>6</td>
</tr>
</tbody>
</table>

### 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:

- **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

3-44
**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**

![Block Parameters: Fractional part IQN](image)

**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
Return the integer portion of an IQ number

Library
tiiiqmathlib 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**
tiiiqmathlib 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
IQN to Float

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
**Purpose**

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

**Library**

tiiiqmathlib 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
**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.

**New Q** — 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
ttiQmathlib in Embedded Target for TI C2000 DSP

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

Output 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
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

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

- **Keep ASM Files** — 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.

- **Optimization Level** — 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)`.

- **Symbolic Debugging** — 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

- **Create MAP File** — 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.

- **Keep OBJ Files** — 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.

- **Linker CMD File** — 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 LF2407 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.

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:

```plaintext
error: can t allocate '.far'
```

or

```plaintext
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

- **Build Action** — 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.

- **Overrun Action** — Defines the action to take when an interrupt overrun occurs.

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

**CCSLink**

- **CCS Handle Name** — 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.

- **Export CCS Handle** — Whether to export the CCS handle to your MATLAB workspace, giving it the name you assigned in **CCS Handle Name**. 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`.

**CodeGen**

- **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**

- **DSP Board Label** — 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

Nominal bit time

SYNCSEG

SJW

TSEG1

TSEG2

Transmit point

Sample point

1 TQ
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
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.

- **Positive Limit** — Maximum positive value to which to saturate
- **Negative Limit** — Minimum negative value to which to saturate

**Dialog Box**

![Block Parameters: Saturate IQN]

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

Purpose

Calculate the square root or inverse square root

Library

tiiiqmathlib 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 option — Whether to calculate the square root or inverse square root

- **Square root (_sqrt)** — Compute the square root
- **Inverse square root (_isqrt)** — Compute the inverse square root

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**

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 \texttt{IQsin} and \texttt{IQcos} are 1 to 29. For all others, valid Q values are 1 to 30.

**Function option** — Type of trigonometric function to calculate.

- \texttt{IQsin} — Compute the sine \((\sin(A))\), where \(A\) is in radians.
- \texttt{IQsinPU} — Compute the sine per unit \((\sin(2\pi A))\), where \(A\) is in per-unit radians.
- \texttt{IQcos} — Compute the cosine \((\cos(A))\), where \(A\) is in radians.
- \texttt{IQcosPU} — Compute the cosine per unit \((\cos(2\pi A))\), where \(A\) is in per-unit radians.
- \texttt{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 3-7
ADC blocks
  C24x 3-9
  C28x 3-22
analog-to-digital converter
  See ADC blocks
applications
  TI C2000 1-2
Arctangent IQN block 3-8
asymmetric vs. symmetric waveforms 3-29

B
blocks
  adding to model 1-24
  recommendations 1-11
build model 1-12
build options 3-36

C
c2000lib startup 1-14
C24x ADC 3-9
C24x CAN Receive block 3-11
C24x CAN Transmit block 3-13
C24x PWM block 3-16
C28x ADC block 3-22
C28x eCAN Receive block 3-24
C28x eCAN Transmit block 3-26
C28x PWM block 3-29
CAN/eCAN
  timing 3-40
CAN/eCAN Receive blocks
  C24x 3-11
  C28x 3-24
CAN/eCAN Transmit blocks
  C24x 3-13
  C28x 3-26
CCS
  link options 3-39
  See also Code Composer Studio
clock speed 1-8
Code Composer Studio 1-7
  projects 1-29
code generation
  ERT and GRT 1-20
  options 3-39
  overview 1-28
code optimization 2-7
compiler options 3-36
configuration default 1-7
can area network
  See CAN/eCAN
can logic 3-18
can
  conversion
    float to IQ number 3-44
    IQ number to different IQ number 3-52
    IQ number to float 3-49
  CPU clock speed 1-8
  default build configuration 1-7
  target preferences options 3-39

Index-1
Index

E
ERT code generation 1-20
event manager timer 3-16

F
F2812 eZdsp block 3-36
fixed-point numbers 2-3
Float to IQN block 3-44
floating-point numbers
convert to IQ number 3-44
four-quadrant arctangent 3-8
Fractional part IQN block 3-45
Fractional part IQN x int32 block 3-46
From Memory block
  C24x 3-15
  C28x 3-28

G
general code generation 1-20
GRT code generation 1-20

H
hardware 1-3

I
Integer part IQN block 3-47
Integer part IQN x int32 block 3-48
IQ Math library 2-2
  Absolute IQN block 3-7
  Arctangent IQN block 3-8
  building models 2-7
  code optimization 2-7
  common characteristics 2-2
Division IQN block 3-35
Float to IQN block 3-44
Fractional part IQN block 3-45
Fractional part IQN x int32 block 3-46
Integer part IQN block 3-47
Integer part IQN x int32 block 3-48
IQN to Float block 3-49
IQN x int32 block 3-50
IQN x IQN block 3-51
IQN1 to IQN2 block 3-52
IQN1 x IQN2 block 3-53
Magnitude IQN block 3-62
Q format notation 2-4
Saturate IQN block 3-63
Square Root IQN block 3-64
Trig Fcn IQN block 3-65
IQ numbers
  convert from float 3-44
  convert to different IQ 3-52
  convert to float 3-49
  fractional part 3-45
  integer part 3-47
  magnitude 3-62
  multiply by int32 fractional result 3-46
  multiply 3-51
  multiply by int32 3-50
  multiply by int32 integer part 3-48
  square root 3-64
  trigonometric functions 3-65
IQN to Float block 3-49
IQN x int32 block 3-50
IQN x IQN block 3-51
IQN1 to IQN2 block 3-52
IQN1 x IQN2 block 3-53
L
LF2407 eZdsp block 3-54
linker options 3-37

M
Magnitude IQN block 3-62
mailbox 3-11
math blocks
  See also IQ Math library
MathWorks software 1-4
memory reading
  C24x 3-15
  C28x 3-28
messages
  F2812 eZdsp 3-24
  LF2401 eZdsp 3-11
model
  add blocks 1-24
  building overview 1-12
  creation overview 1-10
  IQmath library 2-7
multiplication
  IQN x int32 3-50
  IQN x int32 fractional part 3-46
  IQN x int32 integer part 3-48
  IQN x IQN 3-51
  IQN1 x IQN2 3-53

O
operating system requirements 1-3
optimization code 2-7

P
prescaler 3-18

projects
  CCS 1-29
pulse wave modulators
  See PWM blocks
pulse width 3-17
PWM blocks
  C24x 3-16
  C28x 3-29
  control logic 3-18
  deadband 3-18

Q
Q format 2-4

R
receive 3-11
RTW build options
  F2812 eZdsp 3-56
  LF2407 eZdsp 3-38
RTW categories 1-18
runtime options 3-38

S
sample time
  F2812 eZdsp 3-25
  LF2407 eZdsp 3-12
  maximum 1-8
Saturate IQN block 3-63
scheduling 1-8
set up hardware 1-3
signed fixed-point numbers 2-3
simulation parameters 1-12
  automatic 1-17
  RTW categories 1-18
software requirements 1-4
Square Root IQN block 3-64
startup c2000lib 1-14

T

target configuration
  example 3-36
  F2812 eZdsp 1-19
  LF2407 eZdsp 3-54
target model creation 1-10
target preferences
  compiler options 3-36
  DSP board options 3-39
  linker options 3-37
Target preferences blocks
  F2812 eZdsp 3-36
  LF2407 eZdsp 3-54
TI software 1-4
timing 1-8
  CAN/eCAN 3-40
TLC debugging 1-19
To Memory blocks
  See C24x To Memory, C28x To Memory
transmit 3-13
Trig Fcn IQN block 3-65

W
waveforms 3-29