Embedded Target for the TI TMS320C2000™ DSP Platform

For Use with Simulink®

- Modeling
- Simulation
- Implementation

User's Guide

Version 1

The MathWorks

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. 3 Apple Hill Drive Natick, MA 01760-2098	Mail

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

Printing History: November 2003 Online only New for Version 1.0 (Release 13SP1+)

Contents

1 Getting Started

2 Using the IQmath Library

3 Block Reference

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?

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`. 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`.

Texas Instruments Software

In addition to the required software from The MathWorks, Embedded Target for the TI TMS320C2000 DSP Platform requires that you install the Texas Instruments development tools and software listed in the following table. Installing Code Composer Studio IDE Version 2.12 or 2.2 for the C28x series installs the software shown.

Required TI Software for Targeting Your TI C2000 Hardware

Installed Product	Additional Information
Assembler	Creates object code (`.obj`) for C2000 boards from assembly code
Compiler	Compiles C code from the blocks in Simulink models into object code (`.obj`). As a byproduct of the compilation process, you get assembly code (`.asm`) as well.

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

Installed Product	Additional Information
Linker	Combines various input files, such as object files and libraries
Code Composer Studio	Texas Instruments integrated development environment (IDE) that provides code debugging and development tools
TI C2000 miscellaneous utilities	Various tools for developing applications for the C2000 digital signal processor family
Code Composer Setup Utility	Program you use to configure your CCS installation by selecting your target boards or simulator

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

Verifying the Configuration

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

```
c2000lib
```

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

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

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

Note For up-to-date information about system requirements, refer to the system requirements page, available in the products area at the MathWorks Web site (`http://www.mathworks.com`).

To verify that CCS is installed on your machine, enter

```
ccsboardinfo
```

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

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

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

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

Note For any model to work in the targeting environment, you must select the discrete-time solver in the **Solver** 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:

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

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

Maximum Sample Times

TimerClockPrescaler Setting	C24x Maximum Sample Time (seconds)	C28x Maximum Sample Time (seconds)
1	0.0016	0.0004
2	0.0032	N/A
4	0.0065	N/A
8	0.0131	N/A
16	0.0262	N/A
32	0.0524	N/A

Maximum Sample Times (Continued)

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

Overview of Creating Models for Targeting

After you have installed the supported development board, start MATLAB. At the MATLAB command prompt, type

```
c2000lib
```

This opens the `c2000lib` Simulink blockset that includes libraries containing blocks predefined for C2000 input and output devices. As needed, add the blocks to your model. See "Using the c2000lib Blockset" on page 1-14 for an example of how to use this library.

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

- Appropriate Target Preferences library block, to set preferences for your target and application
- From the appropriate libraries in the `c2000lib` block library, to handle input and output functions for your target hardware
- From Real-Time Workshop
- From 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.

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

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 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.

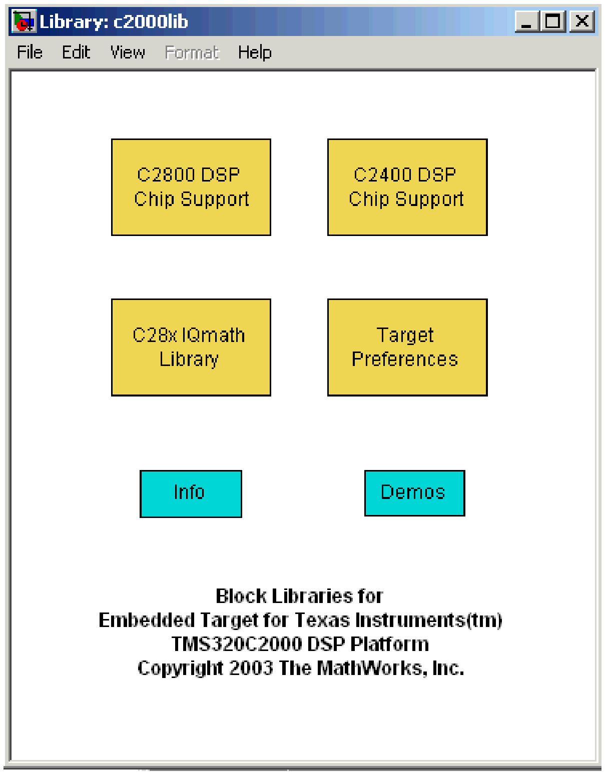
2. Connect the output of PWM1 on the eZdsp F2812 to the analog input of the oscilloscope.

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.



Block Libraries for
Embedded Target for Texas Instruments(tm)
TMS320C2000 DSP Platform
Copyright 2003 The MathWorks, Inc.

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 (`tiiqmathlib`) — 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.

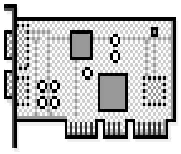
Setting Up the Model

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

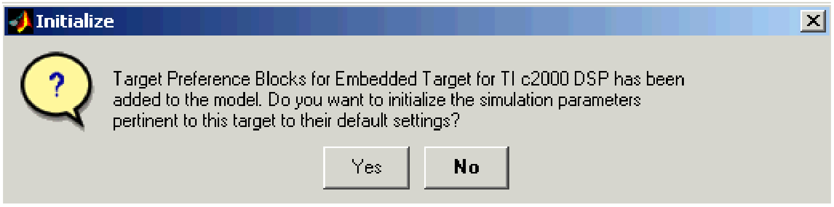
1 Select **New** from the **File** menu to create a new Simulink model.

2 Double-click the Target Preferences library in c2000lib to open it.

3 Drag the F2812 eZdsp block into your new model.


F2812 eZdsp

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


Initialize

Target Preference Blocks for Embedded Target for TI c2000 DSP has been added to the model. Do you want to initialize the simulation parameters pertinent to this target to their default settings?

Click **Yes** to allow automatic setup. The following simulation parameters are set:

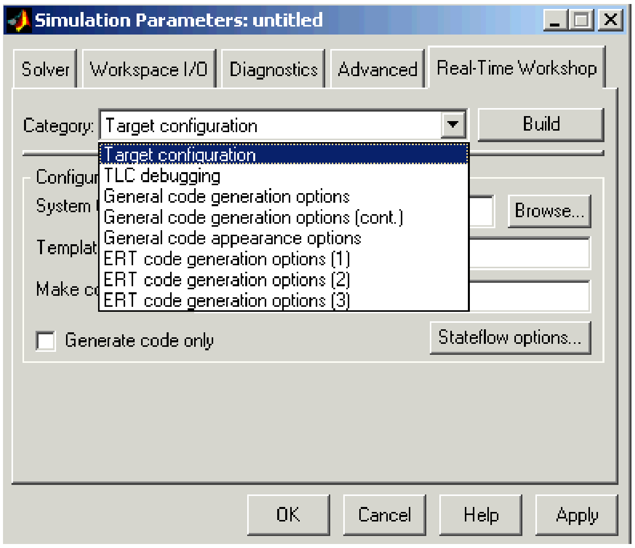
Tab Page	**Field**	**Setting**
Solver	**Stop time**	`inf`
Solver	**Type**	`Fixed-step discrete`
Workspace I/O	**Save to workspace - Time**	`off`
Workspace I/O	**Save to workspace - Output**	`off`
Advanced	**Production hardware characteristics**	`Microprocessor`
Advanced	**Number of bits for C 'char'**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

The default **Target configuration - System target file** is `ti_c2000.grt.tlc`, because you need to purchase and install the optional Real-Time Workshop Embedded Coder to use the `ti_c2000_ert.tlc`.

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

Select **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.


Simulation Parameters: untitled
Solver
Workspace I/O
Diagnostics
Advanced
Real-Time Workshop
Category:
Target configuration
Build
Target configuration
TLC debugging
General code generation options
General code generation options (cont.)
General code appearance options
ERT code generation options (1)
ERT code generation options (2)
ERT code generation options (3)
Configur
System
Browse...
Templat
Make c
Generate code only
Stateflow options...
OK
Cancel
Help
Apply

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 `model.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.

Build Options

Subfield	Field	Setting
Compiler Options	**CompilerVerbosity**	`Verbose`
	KeepASMFiles	`False`
	OptimizationLevel	`Function(-o2)`
	SymbolicDebugging	`Yes`

Build Options (Continued)

Subfield	Field	Setting
Linker Options	**CreateMAPFile**	`True`
	KeepOBJFiles	`True`
	LinkerCMDFile	`Full_memory_map`
RunTime Options	**BuildAction**	`Build_and_execute`
	OverrunAction	`Continue`

CCSLink Options

Field	Setting
CCSHandleName	`CCS_Obj`
ExportCCSHandle	`True`

CodeGeneration Options

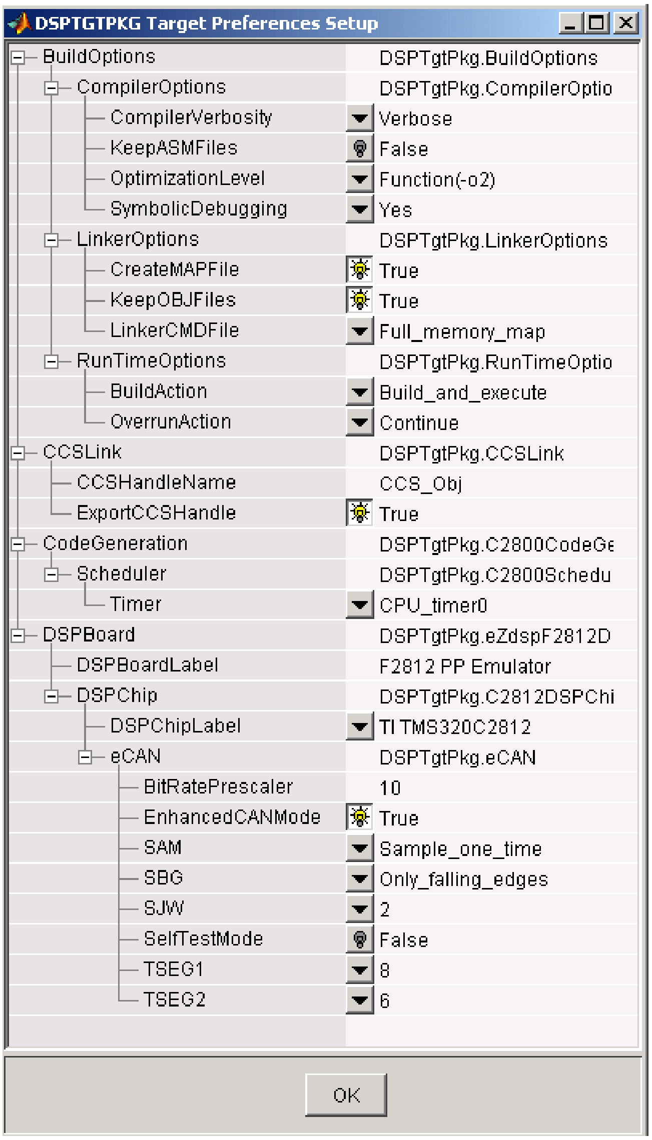
Subfield	Field	Setting
Scheduler	**Timer**	`CPU_timer0`
	TimerClockPrescaler	`1`

DSPBoard Options

Subfield	Field	Setting
DSP Board Label	**DSPBoardLabel**	`F2812 PP Emulator`
DSP Chip	**DSPChipLabel**	`TI TMS320C2812`

DSPBoard Options (Continued)

Subfield	Field	Setting
eCAN	**BitRatePrescaler**	`10`
	EnhancedCANMode	`True`
	SAM	`Sample_one_time`
	SBJ	`Only_falling_edges`
	SJW	`2`
	SelfTestMode	`False`
	TSEG1	`8`
	TSEG2	`6`

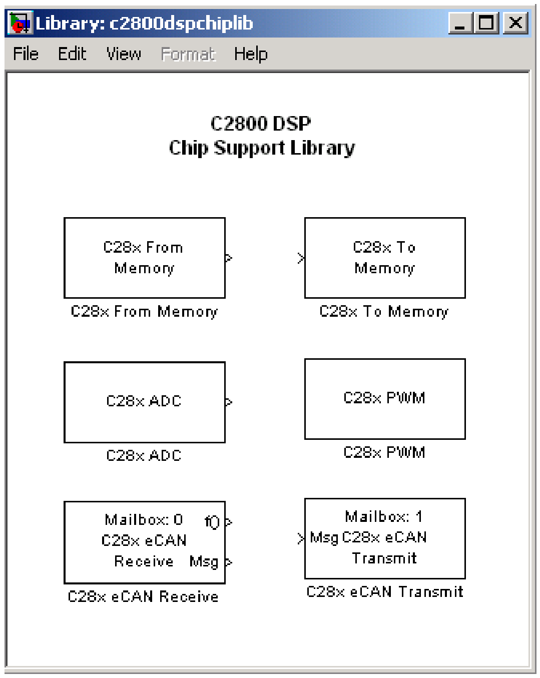


DSPTGTPKG Target Preferences Setup

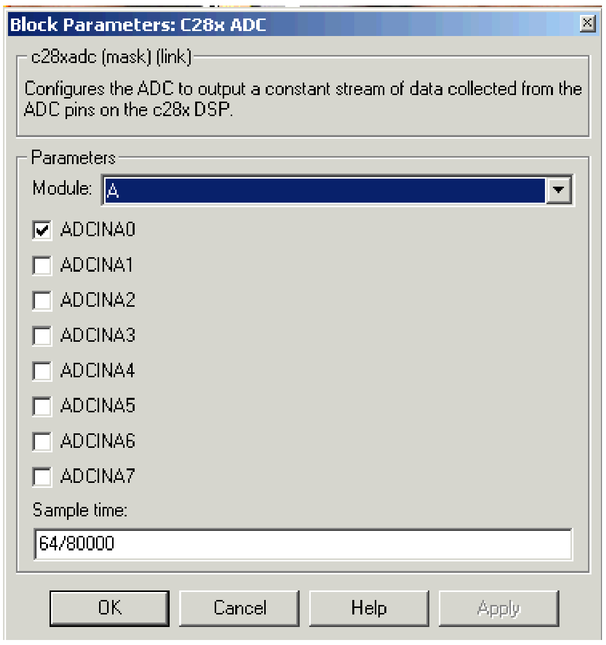
BuildOptions	DSPTgtPkg.BuildOptions
CompilerOptions	DSPTgtPkg.CompilerOptio
CompilerVerbosity	Verbose
KeepASMFiles	False
OptimizationLevel	Function(-o2)
SymbolicDebugging	Yes
LinkerOptions	DSPTgtPkg.LinkerOptions
CreateMAPFile	True
KeepOBJFiles	True
LinkerCMDFile	Full_memory_map
RunTimeOptions	DSPTgtPkg.RunTimeOptio
BuildAction	Build_and_execute
OverrunAction	Continue
CCSLink	DSPTgtPkg.CCSLink
CCSHandleName	CCS_Obj
ExportCCSHandle	True
CodeGeneration	DSPTgtPkg.C2800CodeGe
Scheduler	DSPTgtPkg.C2800Schedu
Timer	CPU_timer0
DSPBoard	DSPTgtPkg.eZdspF2812D
DSPBoardLabel	F2812 PP Emulator
DSPChip	DSPTgtPkg.C2812DSPChi
DSPChipLabel	TI TMS320C2812
eCAN	DSPTgtPkg.eCAN
BitRatePrescaler	10
EnhancedCANMode	True
SAM	Sample_one_time
SBG	Only_falling_edges
SJW	2
SelfTestMode	False
TSEG1	8
TSEG2	6

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.


Block Parameters: C28x ADC

c28xadc (mask) (link)

Configures the ADC to output a constant stream of data collected from the ADC pins on the c28x DSP.

Parameters

Module: A

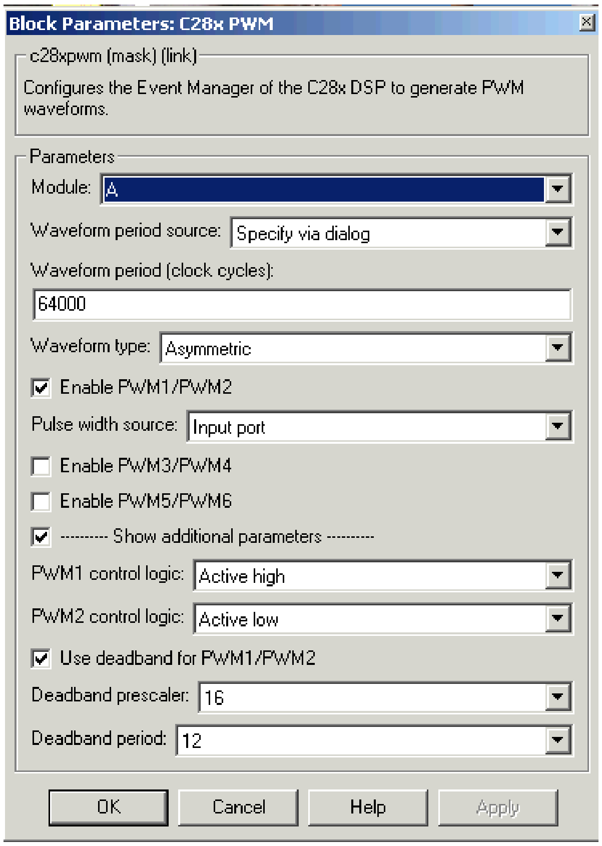
- [✓] ADCINA0
- [ ] ADCINA1
- [ ] ADCINA2
- [ ] ADCINA3
- [ ] ADCINA4
- [ ] ADCINA5
- [ ] ADCINA6
- [ ] ADCINA7

Sample time: 64/80000

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

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

Field	Parameter
PWM1 control logic	`Active high`
PWM2 control logic	`Active low`
Use deadband for PWM1/PWM2	selected
Deadband prescaler	`16`
Deadband period	`12`



Block Parameters: C28x PWM

c28xpwm (mask) (link)

Configures the Event Manager of the C28x DSP to generate PWM waveforms.

Parameters

Module: A

Waveform period source: Specify via dialog

Waveform period (clock cycles):

64000

Waveform type: Asymmetric

[✓] Enable PWM1/PWM2

Pulse width source: Input port

[ ] Enable PWM3/PWM4

[ ] Enable PWM5/PWM6

[✓] --------- Show additional parameters ---------

PWM1 control logic: Active high

PWM2 control logic: Active low

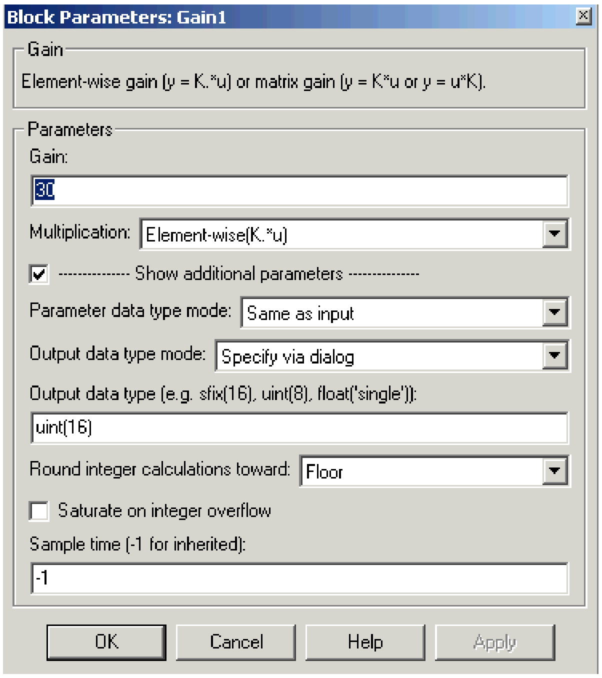
[✓] Use deadband for PWM1/PWM2

Deadband prescaler: 16

Deadband period: 12

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

Field	**Parameter**
Gain	`30`
Multiplication	`Element-wise(K.*u)`
Show additional parameters	selected
Parameter data type mode	`Same as input`
Output data type mode	`Specify via dialog`
Output data type	`uint(16)`
Round integer calculations toward	`Floor`
Sample time	`-1`



Block Parameters: Gain1

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Parameters

Gain:

30

Multiplication: Element-wise(K.*u)

[✓] -------------- Show additional parameters --------------

Parameter data type mode: Same as input

Output data type mode: Specify via dialog

Output data type (e.g. sfix(16), uint(8), float('single')):

uint(16)

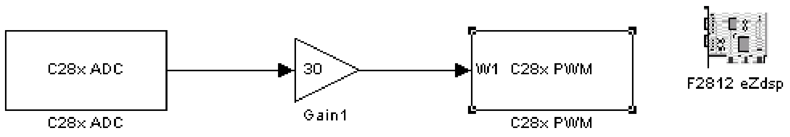
Round integer calculations toward: Floor

[ ] Saturate on integer overflow

Sample time (-1 for inherited):

-1

5 Connect the ADC block to the Gain block and the Gain block to the PWM block as shown.



Generating Code from the Model

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

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

The code building process consists of these tasks:

1. Real-Time Workshop invokes the function `make_rtw` to start the Real-Time Workshop build procedure for a block diagram. `make_rtw` invokes the Target Language Compiler to generate the code and then invokes the language-specific make procedure.

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

Creating Code Composer Studio Projects Without Loading

To create projects in CCS without loading files to your target, follow these steps:

1. In the **Real-Time Workshop** pane in the **Simulation Parameters** dialog, select `ti_c2000.tlc` as the system target file.

2. Select `Create_CCS_Project` for the **BuildAction** in the Target Preferences block. Note that the `Build` and `Build_and_execute` options create CCS projects as well.

3. Set the other Target Preferences options, including those for `CCSLink`. On the **Real-Time Workshop** pane of the **Simulation Parameters** dialog, click **Build** to build your new CCS project.

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

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

Using the IQmath Library

About the IQmath Library

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.

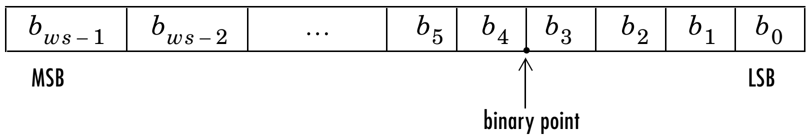
To learn more about characteristics particular to each block in the library, refer to the "Block Reference" pages.

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.



where

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

Signed Fixed-Point Numbers

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

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

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

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

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

Q Format Notation

The position of the binary point in a fixed-point number determines how you interpret the scaling of the number. When it performs basic arithmetic such as addition or subtraction, hardware uses the same logic circuits regardless of the value of the scale factor. In essence, the logic circuits have no knowledge of a binary point. They perform signed or unsigned integer arithmetic — as if the binary point is to the right of b_0. Therefore, you determine the binary point.

In the IQmath Library, the position of the binary point in the signed, fixed-point data types is expressed in and designated by Q format notation. This fixed-point notation takes the form

$Qm.n$

where

- Q designates that the number is in Q format notation — the Texas Instruments representation for signed fixed-point numbers.
- m is the number of bits used to designate the two's complement integer portion of the number.
- n is the number of bits used to designate the two's complement fractional portion of the number, or the number of bits to the right of the binary point.

In Q format, the most significant bit is always designated as the sign bit. Representing a signed fixed-point data type in Q format always requires m+n+1 bits to account for the sign.

Example — Q.15

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

```
Q0.15
```

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

```
Q.15
```

In 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:

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

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 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 — Q17.-2

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

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

Therefore this number is expressed as

```
Q17.-2
```

In 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

C2400 DSP Chip Support Library (c2400dspchiplib)

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

C2800 DSP Chip Support Library (c2800dspchiplib)

Block	Description
C28x ADC	Configure analog to digital converters (ADC)
C28x eCAN Receive	Configure enhanced Control Area Network receive mailbox
C28x eCAN Transmit	Configure enhanced Control Area Network transmit mailbox
C28x PWM	Configure one or more pairs of pulse wave modulators (PWMs)

Block	Description
C28x From Memory	Retrieve data from specific memory location on the target
C28x To Memory	Write data to the specific memory location on the target

Target Preferences Library (c2000tgtpreflib)

Block	Description
F2812 eZdsp	Preferences for F2812 eZdsp™ DSK targets
LF2407 eZdsp	Preferences for LF2407 eZdsp™ DSK targets

C28x IQmath Library (tiiqmathlib)

Block	Description
Absolute IQN	Calculate absolute value
Arctangent IQN	Calculate four-quadrant arc tangent
Division IQN	Divide two IQ numbers
Float to IQN	Convert a floating-point number to an IQ number
Fractional part IQN	Return the fractional part of an IQ number
Fractional part IQN x int32	Return the fractional part of the result of multiplying an IQ number and a long integer
Integer part IQN	Return the integer part of an IQ number
Integer part IQN x int32	Return the integer part of the result of multiplying an IQ number and a long integer

Block	Description
IQN to Float	Convert an IQ number to a floating-point number
IQN x int32	Multiply an IQ number and a long integer
IQN x IQN	Multiply two IQ numbers with the same Q format
IQN1 to IQN2	Convert an IQ number to a different Q format
IQN1 x IQN2	Multiply two IQ numbers with different Q formats
Magnitude IQN	Calculate the magnitude of two orthogonal IQ numbers
Saturate IQN	Saturate an IQ number
Square Root IQN	Calculate the square root or inverse square root of an IQ number
Trig Fcn IQN	Calculate the sine, cosine, or tangent of an IQ number

Blocks — Alphabetical List

Purpose

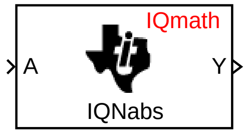
Calculate the absolute value of an IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

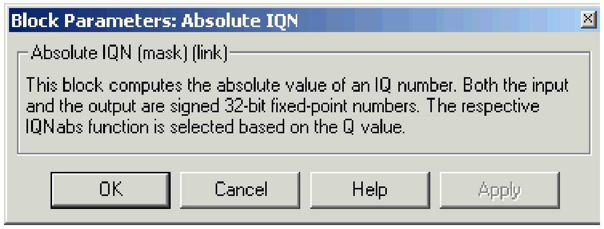
Description

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



Absolute IQN

Dialog Box



Absolute IQN (mask) (link)

This block computes the absolute value of an IQ number. Both the input and the output are signed 32-bit fixed-point numbers. The respective IQNabs function is selected based on the Q value.

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

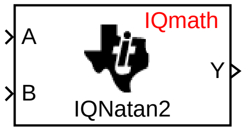
Purpose

Calculate the four-quadrant arc tangent

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description



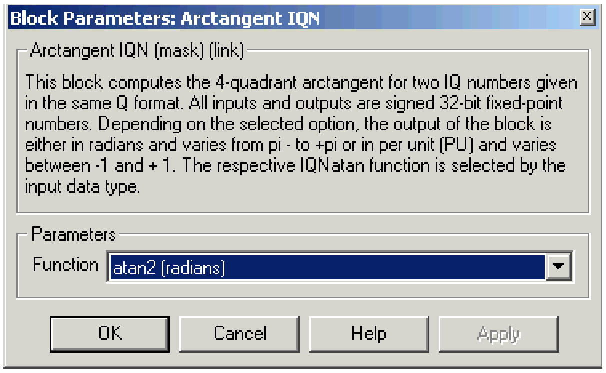
Arctangent IQN

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



Arctangent IQN (mask) (link)

This block computes the 4-quadrant arctangent for two IQ numbers given in the same Q format. All inputs and outputs are signed 32-bit fixed-point numbers. Depending on the selected option, the output of the block is either in radians and varies from pi - to +pi or in per unit (PU) and varies between -1 and + 1. The respective IQNatan function is selected by the input data type.

Parameters

Function: atan2 (radians)

See Also

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

C24x ADC

Purpose

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

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

C24x ADC

The C24x ADC block configures the C24x ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. 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 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

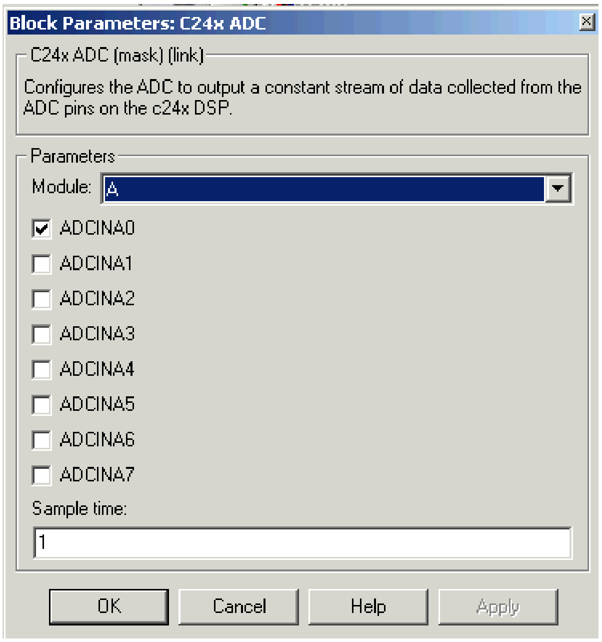
- **A** — Displays the ADC channels in module A (ADCINA0 through ADCINA7)
- **B** — Displays the ADC channels in module B (ADCINB0 through ADCINB7)
- **A and B** — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7)

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.

Dialog Box



Block Parameters: C24x ADC

C24x ADC (mask) (link)

Configures the ADC to output a constant stream of data collected from the ADC pins on the c24x DSP.

Parameters

Module: A

[✓] ADCINA0

[ ] ADCINA1

[ ] ADCINA2

[ ] ADCINA3

[ ] ADCINA4

[ ] ADCINA5

[ ] ADCINA6

[ ] ADCINA7

Sample time: 1

See Also

C24x PWM

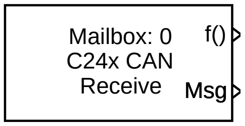
Purpose

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



C24x CAN Receive

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

The C24x CAN Receive block has up to two and, optionally, three output ports.

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

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

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

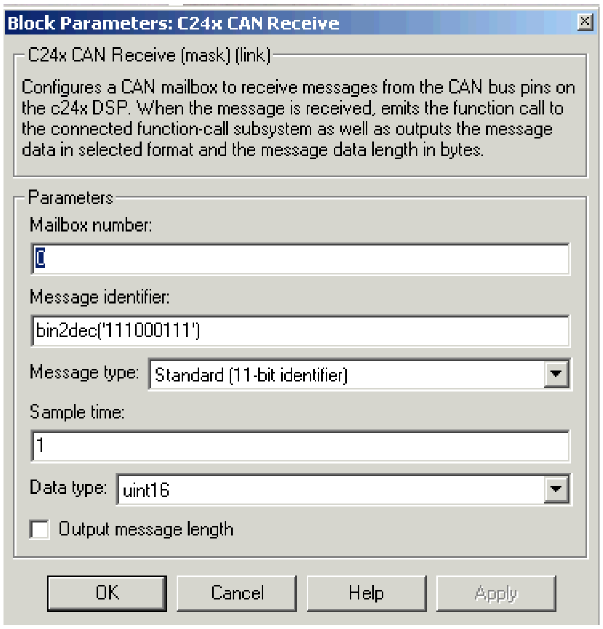
Message type — Select `Standard (11-bit identifier)` or `Extended (29-bit identifier)`.

Sample time — Frequency with which the mailbox is polled to determine if a new message has been received. A new message causes a function call to be emitted from the mailbox.

Data type — Type of 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



Block Parameters: C24x CAN Receive

C24x CAN Receive (mask) (link)

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

Parameters

Mailbox number: 0

Message identifier: bin2dec('111000111')

Message type: Standard (11-bit identifier)

Sample time: 1

Data type: uint16

☐ Output message length

See Also

C24x CAN Transmit

C24x CAN Transmit

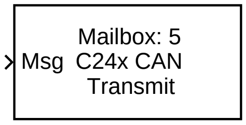
Purpose

Configure a CAN mailbox to transmit messages to the CAN pins

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description



C24x CAN Transmit

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

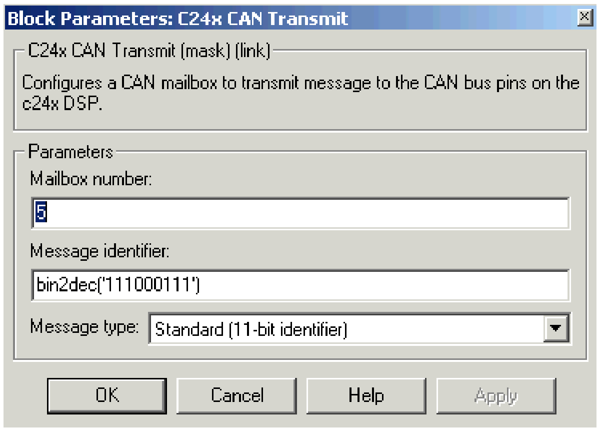
Detailed information on the CAN module is in the *TMS320LF/LC240xA DSP Controller Reference Guide — System and Peripherals*, Literature Number SPRU357B, available at the Texas Instruments website.

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)`.

Dialog Box



Block Parameters: C24x CAN Transmit

C24x CAN Transmit (mask) (link)

Configures a CAN mailbox to transmit message to the CAN bus pins on the c24x DSP.

Parameters

Mailbox number: 5

Message identifier: bin2dec('111000111')

Message type: Standard (11-bit identifier)

See Also

C24x CAN Receive

C24x From Memory

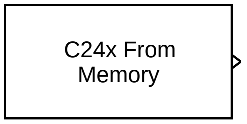
Purpose

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



C24x From Memory

This block retrieves data of the specified data type from a particular memory address on the target.

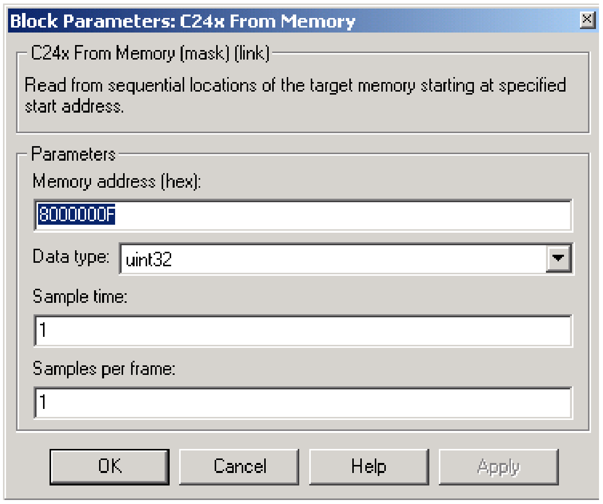
Memory address — Address of the target memory location, in hexadecimal, from which to read data

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



C24x From Memory (mask) (link)

Read from sequential locations of the target memory starting at specified start address.

Parameters

Memory address (hex): 8000000F

Data type: uint32

Sample time: 1

Samples per frame: 1

See Also

C24x To Memory

C24x PWM

Purpose

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

Library

`c2400dspchiplib` in Embedded Target for TI C2000 DSP

Description

C24x PWM

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.

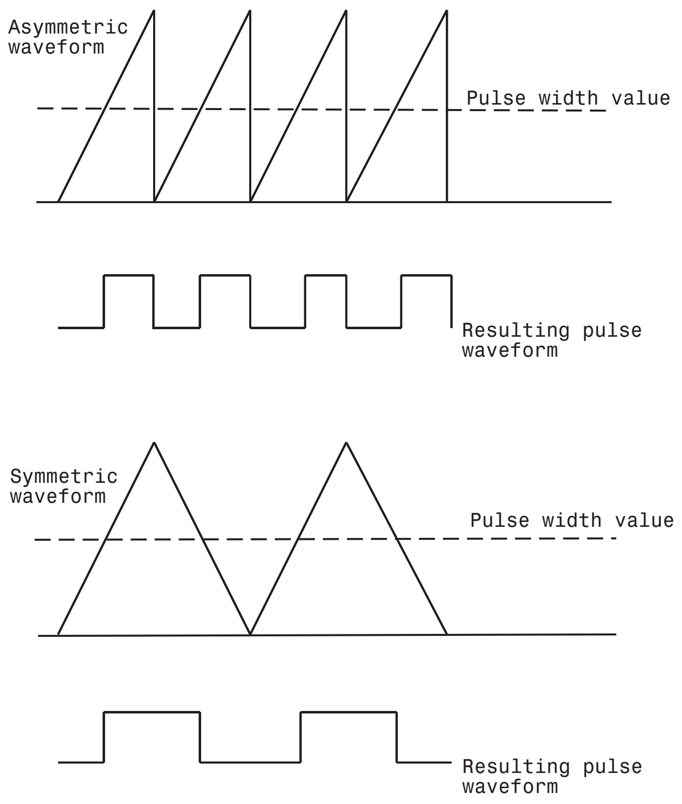
Waveform period source — Source from which the waveform period value is obtained. Select **Specify via dialog** to enter the value in **Waveform period** or select **Input port** to use a value from the input port.

Waveform period — Period of the timer used to generate PWM waveform measured in clock cycles. The relationship betwen 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

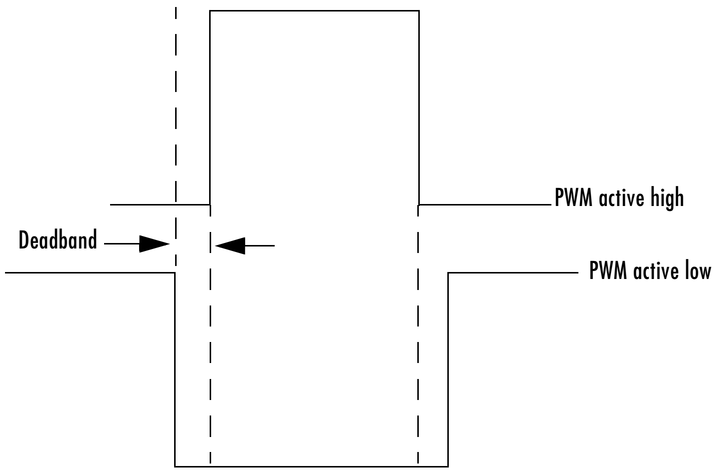
Pulse width source — Source from which the pulse width is obtained. Select **Specify via dialog** to enter the value in **Pulse width** or select **Input port** to use a value from the input port.

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.

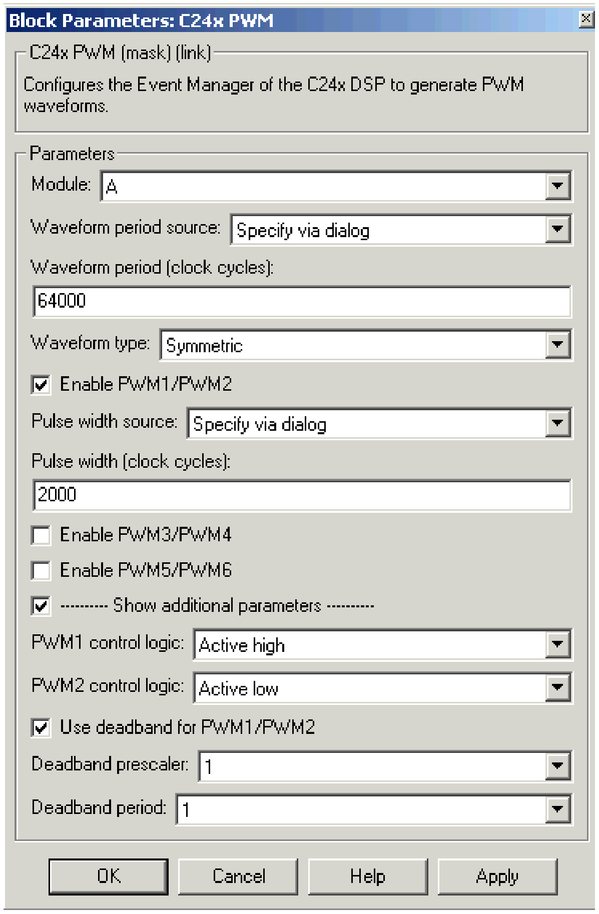
Use deadband for PWM#/PWM# — Enables a deadband area of no signal overlap at the beginning of particular PWM pair signals



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


Block Parameters: C24x PWM

C24x PWM (mask) (link)

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

Parameters

Module: A

Waveform period source: Specify via dialog

Waveform period (clock cycles):

64000

Waveform type: Symmetric

[✓] Enable PWM1/PWM2

Pulse width source: Specify via dialog

Pulse width (clock cycles):

2000

[ ] Enable PWM3/PWM4

[ ] Enable PWM5/PWM6

[✓] --------- Show additional parameters ---------

PWM1 control logic: Active high

PWM2 control logic: Active low

[✓] Use deadband for PWM1/PWM2

Deadband prescaler: 1

Deadband period: 1

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.

C24x To Memory

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

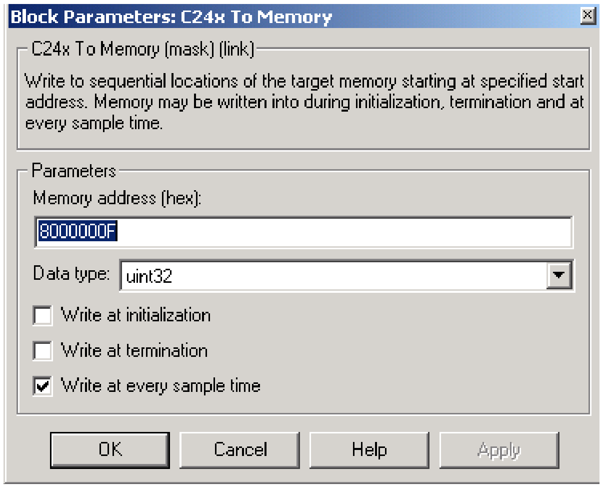
Write at every sample time — Whether to write data in real time during program execution

---

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**.

Dialog Box



Block Parameters: C24x To Memory

C24x To Memory (mask) (link)

Write to sequential locations of the target memory starting at specified start address. Memory may be written into during initialization, termination and at every sample time.

Parameters

Memory address (hex): 8000000F

Data type: uint32

- [ ] Write at initialization
- [ ] Write at termination
- [x] Write at every sample time

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

C28x ADC

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 reprensenting 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 blcok 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

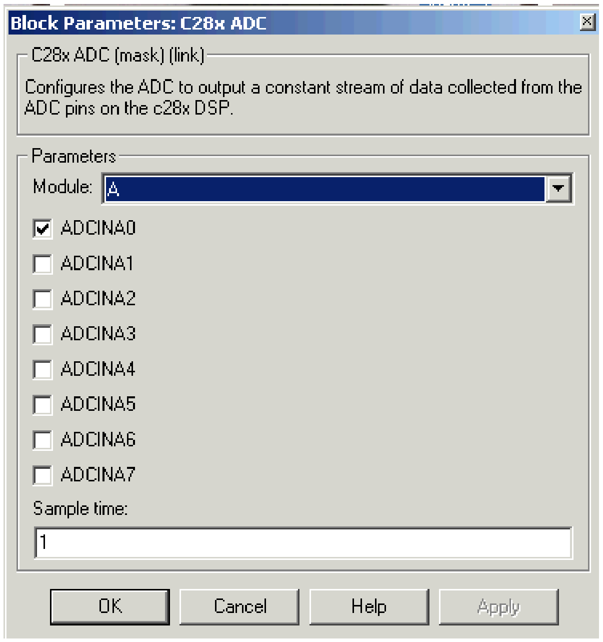
- **A** — Displays the ADC channels in module A (ADCINA0 through ADCINA7)
- **B** — Displays the ADC channels in module B (ADCINB0 through ADCINB7)
- **A and B** — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7)

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



Block Parameters: C28x ADC

C28x ADC (mask) (link)

Configures the ADC to output a constant stream of data collected from the ADC pins on the c28x DSP.

Parameters

Module: A

- [✓] ADCINA0
- [ ] ADCINA1
- [ ] ADCINA2
- [ ] ADCINA3
- [ ] ADCINA4
- [ ] ADCINA5
- [ ] ADCINA6
- [ ] ADCINA7

Sample time: 1

See Also

C28x PWM

C28x eCAN Receive

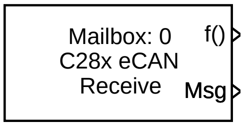
Purpose

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



C28x eCAN Receive

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.

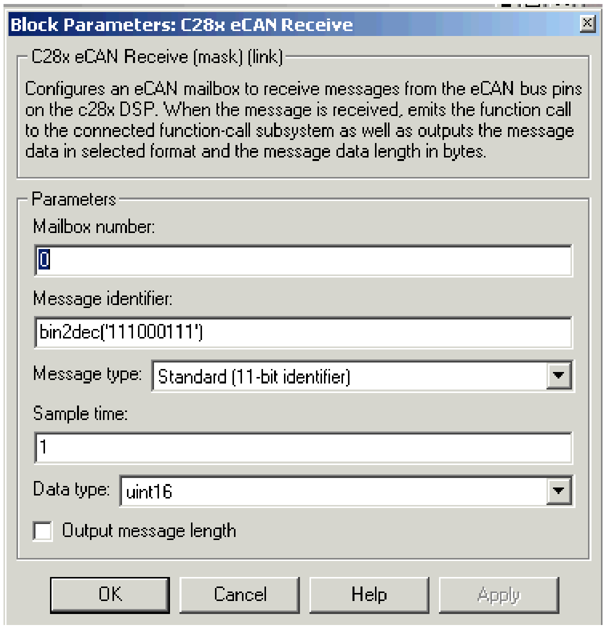
Message type — Select `Standard (11-bit identifier)` or `Extended (29-bit identifier).`

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 `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



C28x eCAN Receive (mask) (link)

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

Parameters

Mailbox number: 0

Message identifier: bin2dec('111000111')

Message type: Standard (11-bit identifier)

Sample time: 1

Data type: uint16

☐ Output message length

See Also

C28x eCAN Transmit

C28x eCAN Transmit

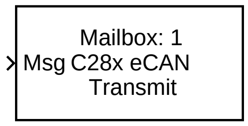
Purpose

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



C28x eCAN Transmit

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.

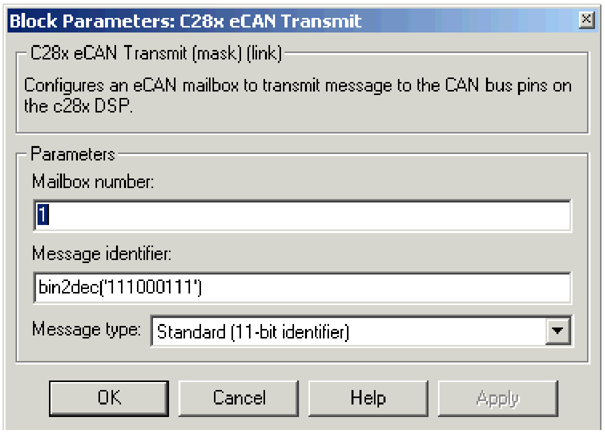
Detailed information on the eCAN module is in the *TMS320F28x DSP Enhanced Control Area Network (eCAN) Reference Guide*, Literature Number SPRU074A, available at the Texas Instruments Web site.

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

C28x eCAN Transmit (mask) (link)

Configures an eCAN mailbox to transmit message to the CAN bus pins on the c28x DSP.

Parameters

Mailbox number:

1

Message identifier:

bin2dec('111000111')

Message type: Standard (11-bit identifier)

See Also C28x eCAN Receive

C28x From Memory

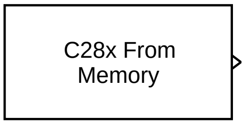
Purpose

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



C28x From Memory

This block retrieves data of the specified data type from a particular memory address on the target.

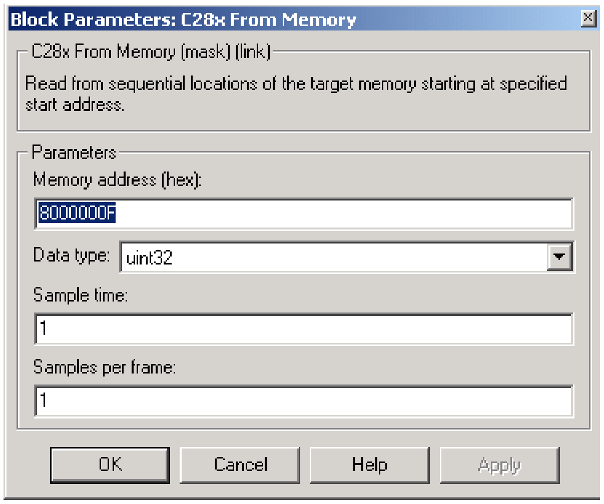
Memory address — Address of the target memory location, in hexadecimal, from which to read data

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

C28x From Memory (mask) (link)

Read from sequential locations of the target memory starting at specified start address.

Parameters

Memory address (hex): 8000000F

Data type: uint32

Sample time: 1

Samples per frame: 1

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

C28x PWM

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.

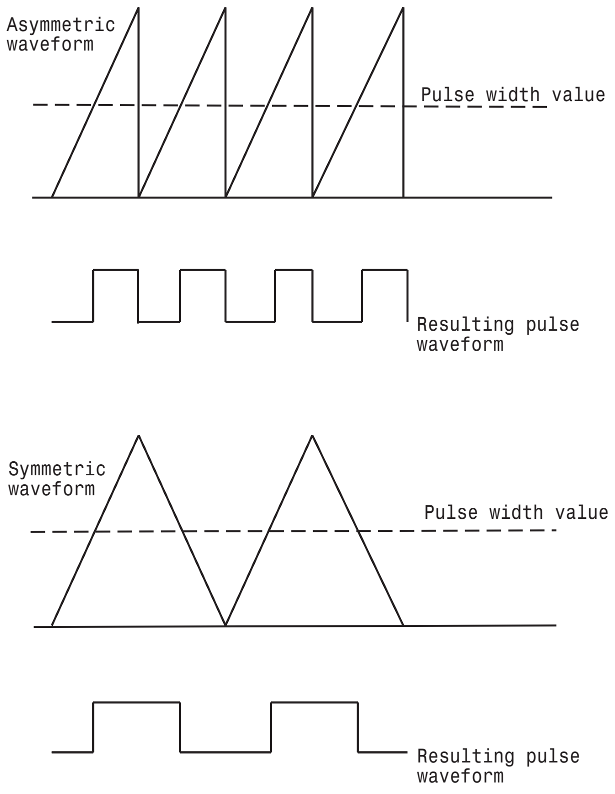
Waveform period source — Source from which the waveform period value is obtained. Select **Specify via dialog** to enter the value in **Waveform period** or select **Input port** to use a value from the input port.

Waveform period — Period of the timer used to generate the PWM waveform measured in clock cycles. The relationship betwen 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

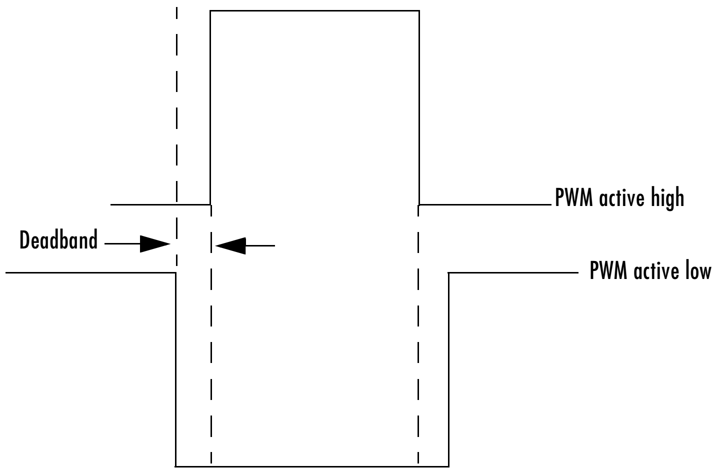
Pulse width source — Source from which the pulse width is obtained. Select **Specify via dialog** to enter the value in **Pulse width** or select **Input port** to use a value from the input port.

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.

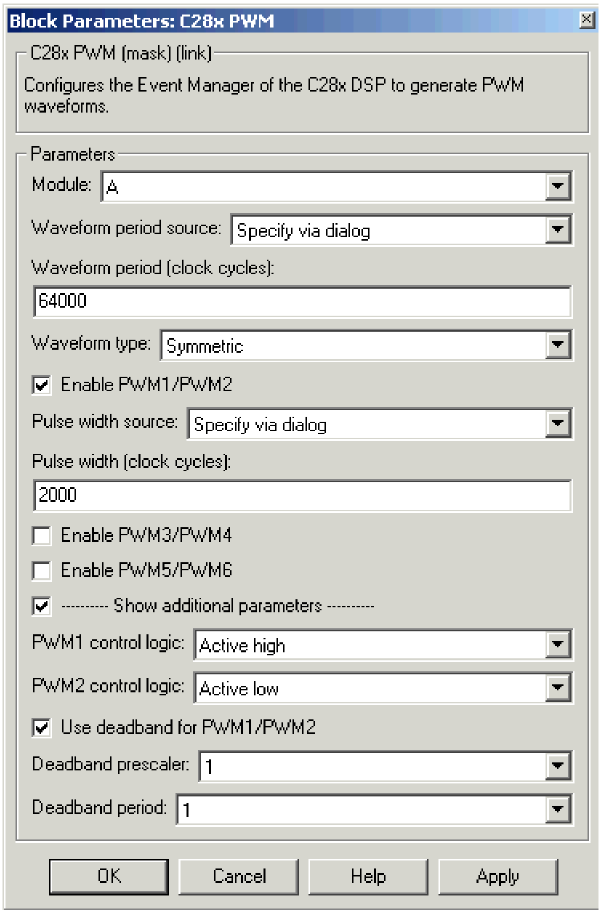
Use deadband for PWM#/PWM# — Enables a deadband area of no signal overlap at the beginning of particular PWM pair signals



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.

Dialog Box



Block Parameters: C28x PWM

C28x PWM (mask) (link)

Configures the Event Manager of the C28x DSP to generate PWM waveforms.

Parameters

Module: A

Waveform period source: Specify via dialog

Waveform period (clock cycles): 64000

Waveform type: Symmetric

[✓] Enable PWM1/PWM2

Pulse width source: Specify via dialog

Pulse width (clock cycles): 2000

[ ] Enable PWM3/PWM4

[ ] Enable PWM5/PWM6

[✓] --------- Show additional parameters ---------

PWM1 control logic: Active high

PWM2 control logic: Active low

[✓] Use deadband for PWM1/PWM2

Deadband prescaler: 1

Deadband period: 1

See Also

C28x ADC

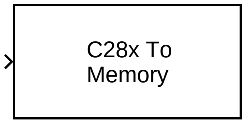
Purpose

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



C28x To Memory

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

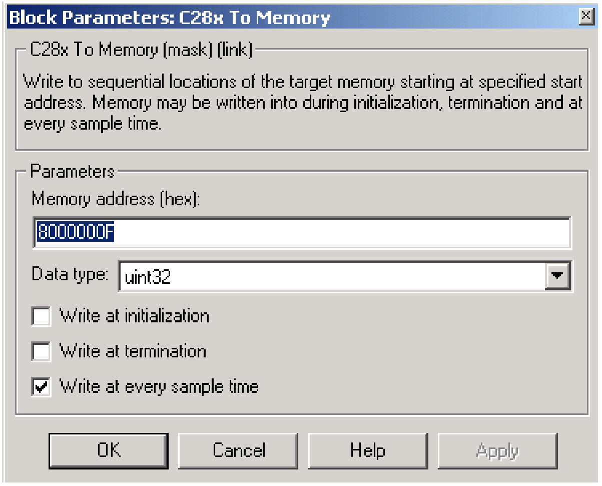
---

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



Block Parameters: C28x To Memory

C28x To Memory (mask) (link)

Write to sequential locations of the target memory starting at specified start address. Memory may be written into during initialization, termination and at every sample time.

Parameters

Memory address (hex): 8000000F

Data type: uint32

- [ ] Write at initialization
- [ ] Write at termination
- [x] Write at every sample time

See Also

C28x From Memory

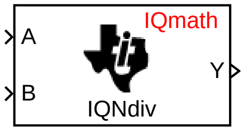
Purpose

Divide two IQ numbers

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

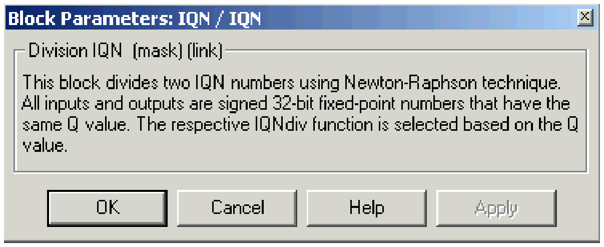
Description



IQN / IQN

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

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 IQNdiv function is selected based on the Q value.

See Also

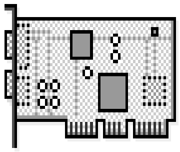
Absolute IQN, Arctangent IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

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.


F2812 eZdsp

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` — Drects 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

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

    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.

    EnhancedCANMode — Whether to use the CAN module in extended mode, which provides additional mailboxes and time stamping. The default is `True`. Setting this parameter to `False` enables only standard mode.

SAM — Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of TQ/2. A majority decision is made from the three points.

SBG — Sets the message resynchronization triggering. Options are `Only_falling_edges` and `Both_falling_and_rising_edges`.

SJW — Sets the synchronization jump width, which determines how many units of TQ a bit is allowed to be shortened or lengthened when resynchronizing.

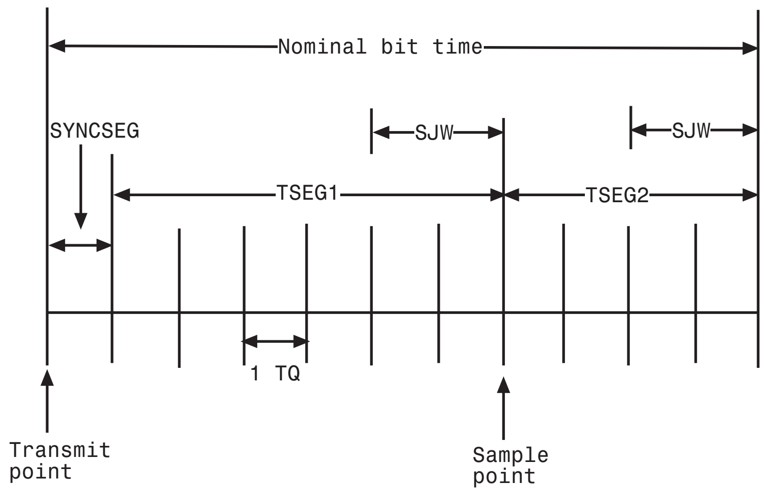
SelfTestMode — If `True`, sets the eCAN module to loopback mode, where a “dummy” acknowledge message is sent back without needing an acknowledge bit. The default is `False`.

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

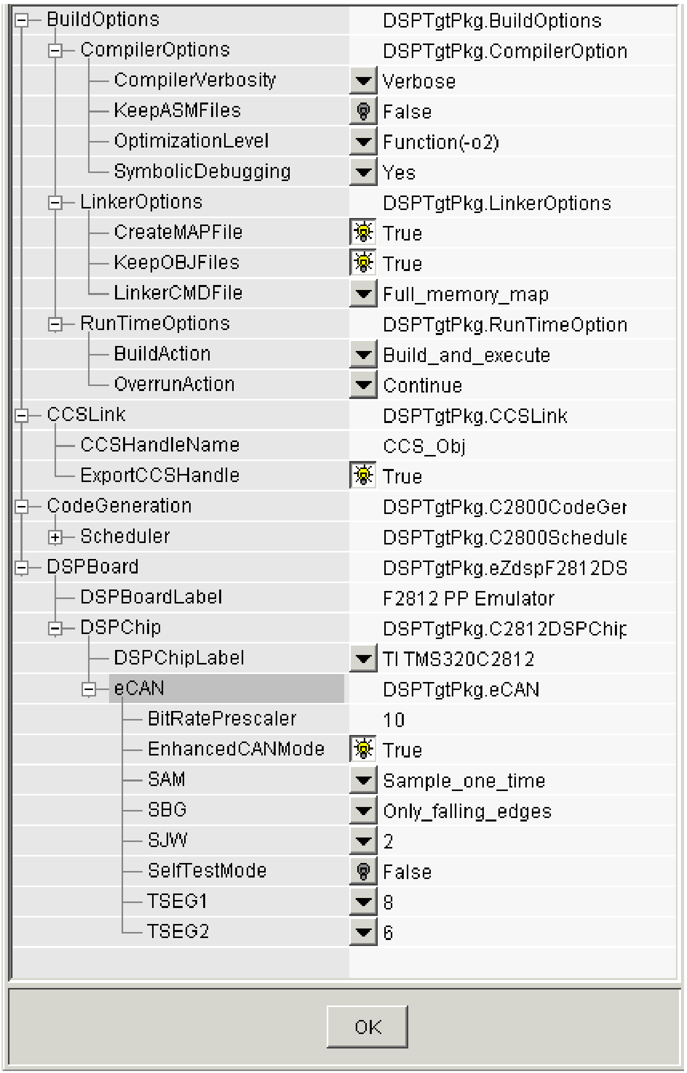
TSEG2 — Sets the value of time segment 2 (PHASE_SEG2), which, with TSEG1 and BRP, determines the length of a bit on the eCAN bus. TSEG2 must be less than or equal to TSEG1 and greater than or equal to IPT. Valid values for TSEG2 are from 1 through 8.

The eCAN bit timing is shown in the following illustration.

CAN Bit Timing



Dialog Box



BuildOptions	DSPTgtPkg.BuildOptions
CompilerOptions	DSPTgtPkg.CompilerOption
CompilerVerbosity	Verbose
KeepASMFiles	False
OptimizationLevel	Function(-o2)
SymbolicDebugging	Yes
LinkerOptions	DSPTgtPkg.LinkerOptions
CreateMAPFile	True
KeepOBJFiles	True
LinkerCMDFile	Full_memory_map
RunTimeOptions	DSPTgtPkg.RunTimeOption
BuildAction	Build_and_execute
OverrunAction	Continue
CCSLink	DSPTgtPkg.CCSLink
CCSHandleName	CCS_Obj
ExportCCSHandle	True
CodeGeneration	DSPTgtPkg.C2800CodeGer
Scheduler	DSPTgtPkg.C2800Schedule
DSPBoard	DSPTgtPkg.eZdspF2812DS
DSPBoardLabel	F2812 PP Emulator
DSPChip	DSPTgtPkg.C2812DSPChip
DSPChipLabel	TI TMS320C2812
eCAN	DSPTgtPkg.eCAN
BitRatePrescaler	10
EnhancedCANMode	True
SAM	Sample_one_time
SBG	Only_falling_edges
SJW	2
SelfTestMode	False
TSEG1	8
TSEG2	6

See Also

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

Float to IQN

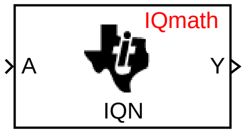
Purpose

Convert a floating-point number to an IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

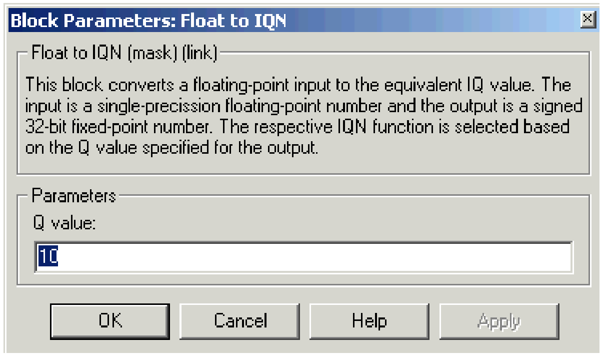


Float to IQN

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



Float to IQN (mask) (link)

This block converts a floating-point input to the equivalent IQ value. The input is a single-precission floating-point number and the output is a signed 32-bit fixed-point number. The respective IQN function is selected based on the Q value specified for the output.

Parameters

Q value:

10

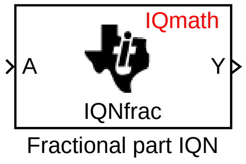
See Also

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

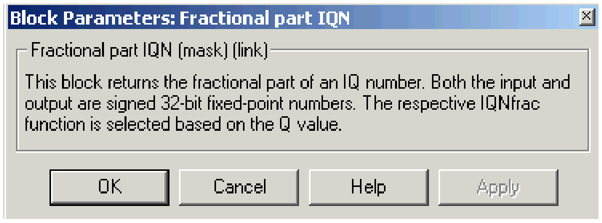
Purpose Return the fractional portion of an IQ number

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

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



Dialog Box


Block Parameters: Fractional part IQN

Fractional part IQN (mask) (link)

This block returns the fractional part of an IQ number. Both the input and output are signed 32-bit fixed-point numbers. The respective IQNfrac function is selected based on the Q value.

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

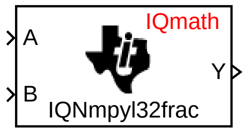
Purpose

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

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

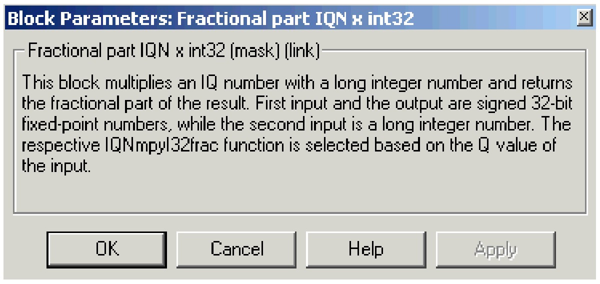
Description



Fractional part
IQN x int32

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

Dialog Box



Fractional part IQN x int32 (mask) (link)

This block multiplies an IQ number with a long integer number and returns the fractional part of the result. First input and the output are signed 32-bit fixed-point numbers, while the second input is a long integer number. The respective IQNmpyl32frac function is selected based on the Q value of the input.

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

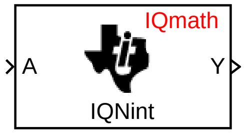
Return the integer portion of an IQ number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

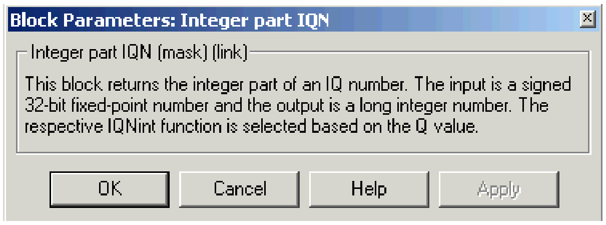
Description

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



Integer part IQN

Dialog Box


Block Parameters: Integer part IQN
Integer part IQN (mask) (link)
OK
Cancel
Help
Apply

This block returns the integer part of an IQ number. The input is a signed 32-bit fixed-point number and the output is a long integer number. The respective IQNint function is selected based on the Q value.

See Also

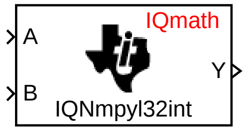
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN x int32, IQN to Float, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

Integer part IQN x int32

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

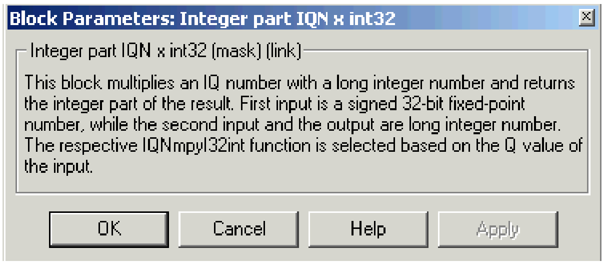
Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

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



Integer part
IQN x int32

Dialog Box


Block Parameters: Integer part IQN x int32
Integer part IQN x int32 (mask) (link)

This block multiplies an IQ number with a long integer number and returns the integer part of the result. First input is a signed 32-bit fixed-point number, while the second input and the output are long integer number. The respective IQNmpyI32int function is selected based on the Q value of the input.

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

Purpose

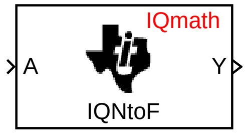
Convert an IQ number to a floating-point number

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

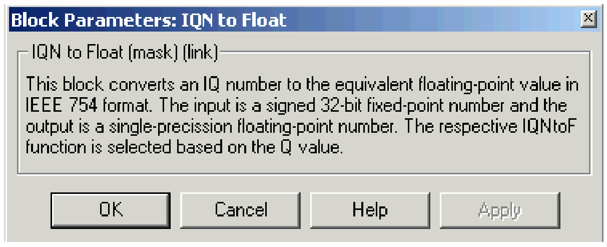
Description

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



IQN to Float

Dialog Box



IQN to Float (mask) (link)

This block converts an IQ number to the equivalent floating-point value in IEEE 754 format. The input is a signed 32-bit fixed-point number and the output is a single-precission floating-point number. The respective IQNtoF function is selected based on the Q value.

See Also

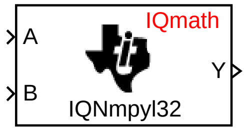
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN x int32, IQN x IQN, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

IQN x int32

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

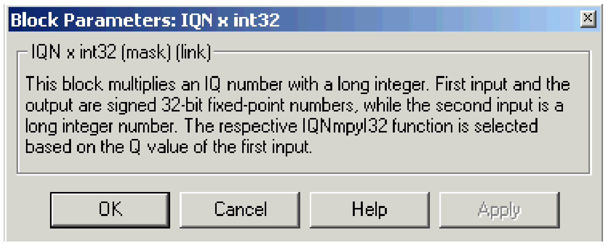
Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

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



IQN x int32

Dialog Box



This block multiplies an IQ number with a long integer. First input and the output are signed 32-bit fixed-point numbers, while the second input is a long integer number. The respective IQNmpyI32 function is selected based on the Q value of the first input.

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

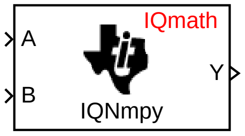
Purpose

Multiply two IQ numbers

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description



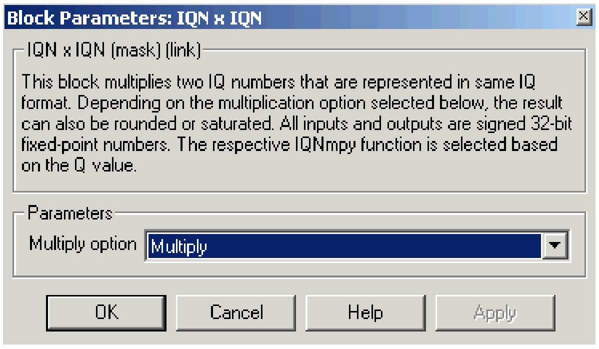
IQN x IQN

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



Block Parameters: IQN x IQN

IQN x IQN (mask) (link)

This block multiplies two IQ numbers that are represented in same IQ format. Depending on the multiplication option selected below, the result can also be rounded or saturated. All inputs and outputs are signed 32-bit fixed-point numbers. The respective IQNmpy function is selected based on the Q value.

Parameters

Multiply option: Multiply

See Also

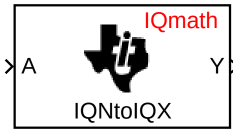
Absolute IQN, Arctangent IQN, Division IQN, Float to IQN, Fractional part IQN, Fractional part IQN x int32, Integer part IQN, Integer part IQN x int32, IQN to Float, IQN x int32, IQN1 to IQN2, IQN1 x IQN2, Magnitude IQN, Saturate IQN, Square Root IQN, Trig Fcn IQN

IQN1 to IQN2

Purpose Convert an IQ number to a different Q format

Library `tiiqmathlib` in Embedded Target for TI C2000 DSP

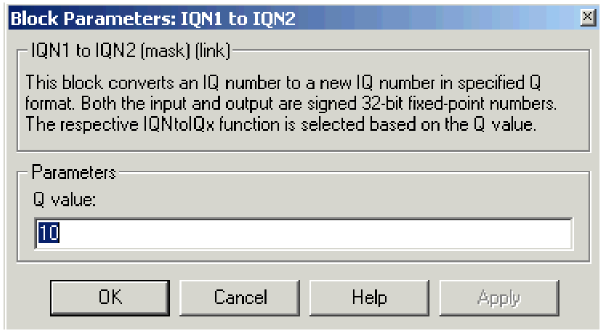
Description This block converts an IQ number in a particular Q format to a different Q format.



IQN1 to IQN2

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

Dialog Box


Block Parameters: IQN1 to IQN2

IQN1 to IQN2 (mask) (link)

This block converts an IQ number to a new IQ number in specified Q format. Both the input and output are signed 32-bit fixed-point numbers. The respective IQNtoIQx function is selected based on the Q value.

Parameters

Q value: 10

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

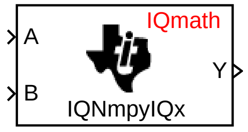
Purpose

Multiply two IQ numbers that are in different Q formats

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

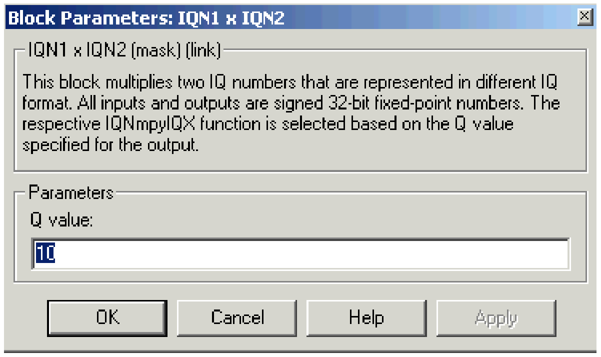


IQN1 x IQN2

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

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

Dialog Box



Block Parameters: IQN1 x IQN2

IQN1 x IQN2 (mask) (link)

This block multiplies two IQ numbers that are represented in different IQ format. All inputs and outputs are signed 32-bit fixed-point numbers. The respective IQNmpyIQX function is selected based on the Q value specified for the output.

Parameters

Q value: 10

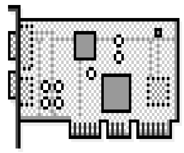
See Also

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.


LF2407 eZdsp

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:

  ```
  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` — Drects 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** — 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

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

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

    PROP_SEG — Time used to compensate for the physical delays in the network

    PHASE_SEG1 — Phase used to compensate for positive edge phase error

    PHASE_SEG2 — Phase used to compensate for negative edge phase error

    The settable parameters are

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

    SAM — Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of TQ/2. A majority decision is made from the three points.

SBG — Sets the message resynchronization triggering. Options are `Only_falling_edges` and `Both_falling_and_rising_edges`.

SJW — Sets the synchronization jump width, which determines how many units of TQ a bit is allowed to be shortened or lengthened when resynchronizing.

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

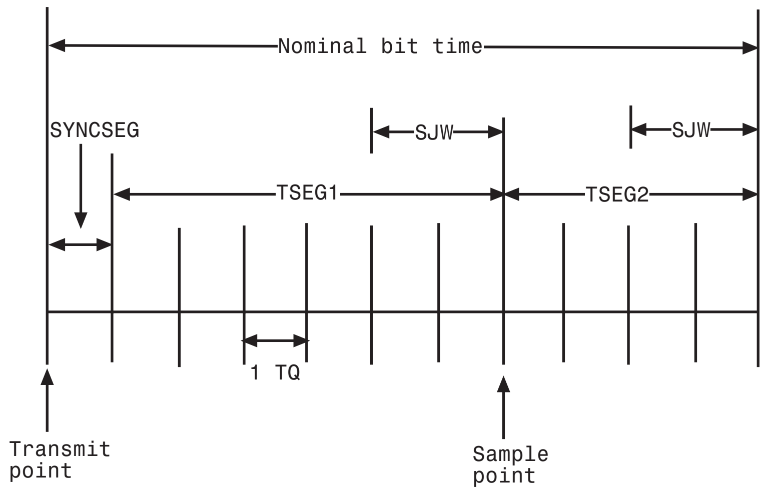
TSEG1 — Sets the value of time segment 1, which, with TSEG2 and BRP, determines the length of a bit on the CAN bus. TSEG1 must be greater than TSEG2 and the Information Processing Time (IPT). The IPT is the time needed to process one bit and corresponds to two TQ units. TSEG1 = PROP_SEG + PHASE_SEG1.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

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

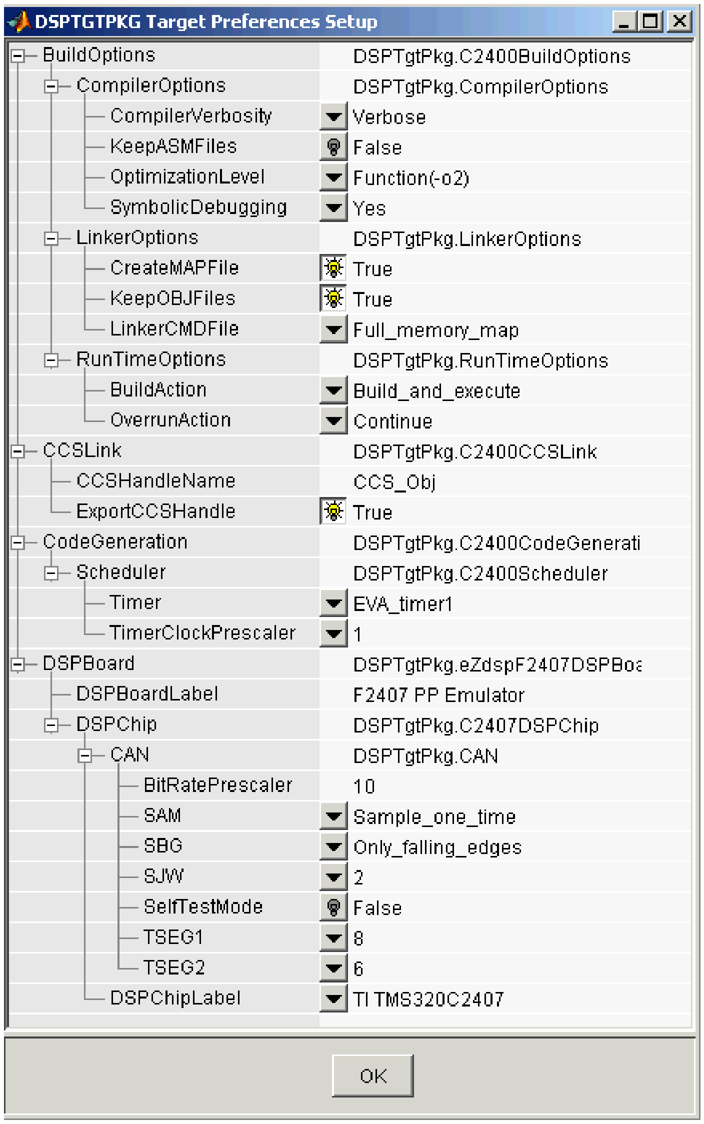
- **DSP Chip Label** — DSP chip model. Select the DSP chip installed on your target. The chip model is fixed for the LF2407 eZdsp. If you change the chip model, an error will be generated in code generation.

The CAN bit timing is shown in the following illustration.

CAN Bit Timing



Dialog Box



BuildOptions	DSPTgtPkg.C2400BuildOptions
CompilerOptions	DSPTgtPkg.CompilerOptions
CompilerVerbosity	Verbose
KeepASMFiles	False
OptimizationLevel	Function(-o2)
SymbolicDebugging	Yes
LinkerOptions	DSPTgtPkg.LinkerOptions
CreateMAPFile	True
KeepOBJFiles	True
LinkerCMDFile	Full_memory_map
RunTimeOptions	DSPTgtPkg.RunTimeOptions
BuildAction	Build_and_execute
OverrunAction	Continue
CCSLink	DSPTgtPkg.C2400CCSLink
CCSHandleName	CCS_Obj
ExportCCSHandle	True
CodeGeneration	DSPTgtPkg.C2400CodeGenerati
Scheduler	DSPTgtPkg.C2400Scheduler
Timer	EVA_timer1
TimerClockPrescaler	1
DSPBoard	DSPTgtPkg.eZdspF2407DSPBoa
DSPBoardLabel	F2407 PP Emulator
DSPChip	DSPTgtPkg.C2407DSPChip
CAN	DSPTgtPkg.CAN
BitRatePrescaler	10
SAM	Sample_one_time
SBG	Only_falling_edges
SJW	2
SelfTestMode	False
TSEG1	8
TSEG2	6
DSPChipLabel	TI TMS320C2407

See Also

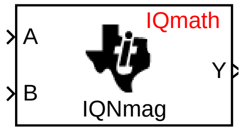
C24x ADC, C24x CAN Receive, C24x CAN Transmit, C24x PWM

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

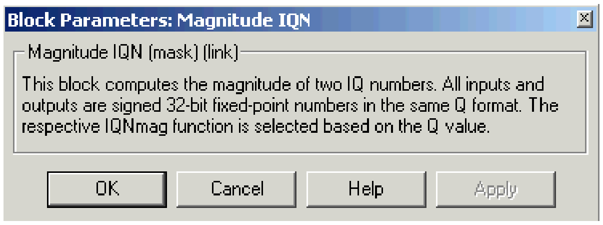


Magnitude IQN

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

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

Dialog Box



Magnitude IQN (mask) (link)

This block computes the magnitude of two IQ numbers. All inputs and outputs are signed 32-bit fixed-point numbers in the same Q format. The respective IQNmag function is selected based on the Q value.

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

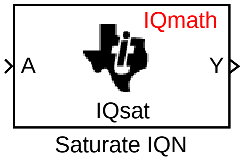
Purpose

Saturate an IQ value to specified limits

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description

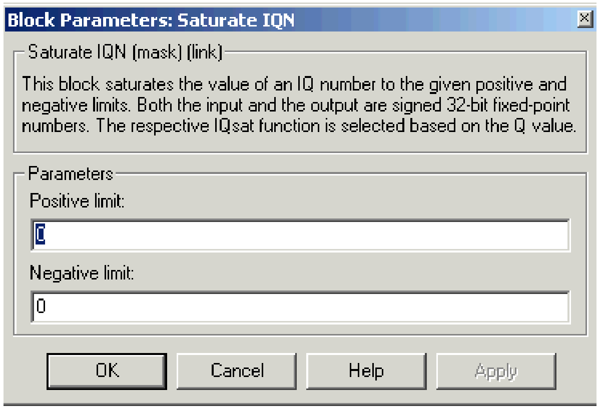


This block saturates an input IQ number to the specified positive and negative limits. The returned value is an IQ number of the same Q value as the input.

Positive Limit — Maximum positive value to which to saturate

Negative Limit — Minimum negative value to which to saturate

Dialog Box



Saturate IQN (mask) (link)

This block saturates the value of an IQ number to the given positive and negative limits. Both the input and the output are signed 32-bit fixed-point numbers. The respective IQsat function is selected based on the Q value.

Parameters

Positive limit:

0

Negative limit:

0

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

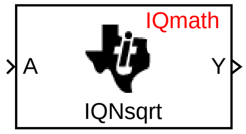
Purpose

Calculate the square root or inverse square root

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description



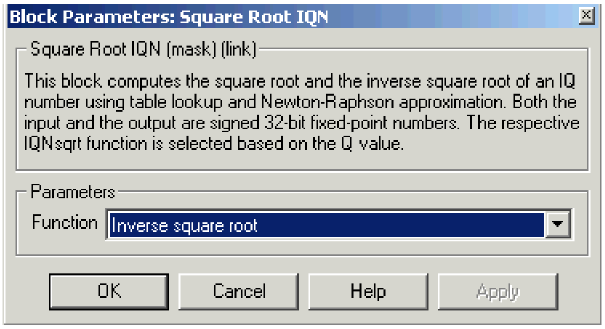
Square Root IQN

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

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



Square Root IQN (mask) (link)

This block computes the square root and the inverse square root of an IQ number using table lookup and Newton-Raphson approximation. Both the input and the output are signed 32-bit fixed-point numbers. The respective IQNsqrt function is selected based on the Q value.

Parameters

Function: Inverse square root

See Also

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

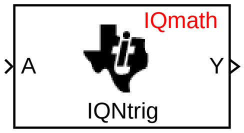
Purpose

Calculate the sine, cosine, or arc tangent

Library

`tiiqmathlib` in Embedded Target for TI C2000 DSP

Description



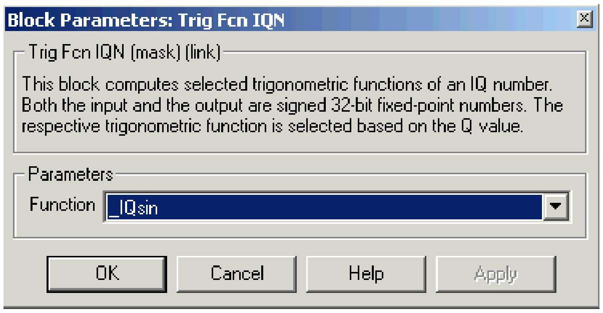
Trig Fcn IQN

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

Function option — Type of trigonometric function to calculate.

- **_IQsin** — Compute the sine `(sin(A))`, where A is in radians.
- **_IQsinPU** — Compute the sine per unit `(sin(2*pi*A))`, where A is in per-unit radians.
- **_IQcos** — Compute the cosine `(cos(A))`, where A is in radians.
- **_IQcosPU** — Compute the cosine per unit `(cos(2*pi*A))`, where A is in per-unit radians.
- **_IQatan** — Compute the arc tangent `(tan(A))`, where A is in radians.

Dialog Box


Block Parameters: Trig Fcn IQN

Trig Fcn IQN (mask) (link)

This block computes selected trigonometric functions of an IQ number. Both the input and the output are signed 32-bit fixed-point numbers. The respective trigonometric function is selected based on the Q value.

Parameters

Function: _IQsin

See Also

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

Index