# Target for Freescale™ MPC5xx  2
## User's Guide

**MATLAB**®
**&SIMULINK**®

The MathWorks™

*Accelerating the pace of engineering and science*
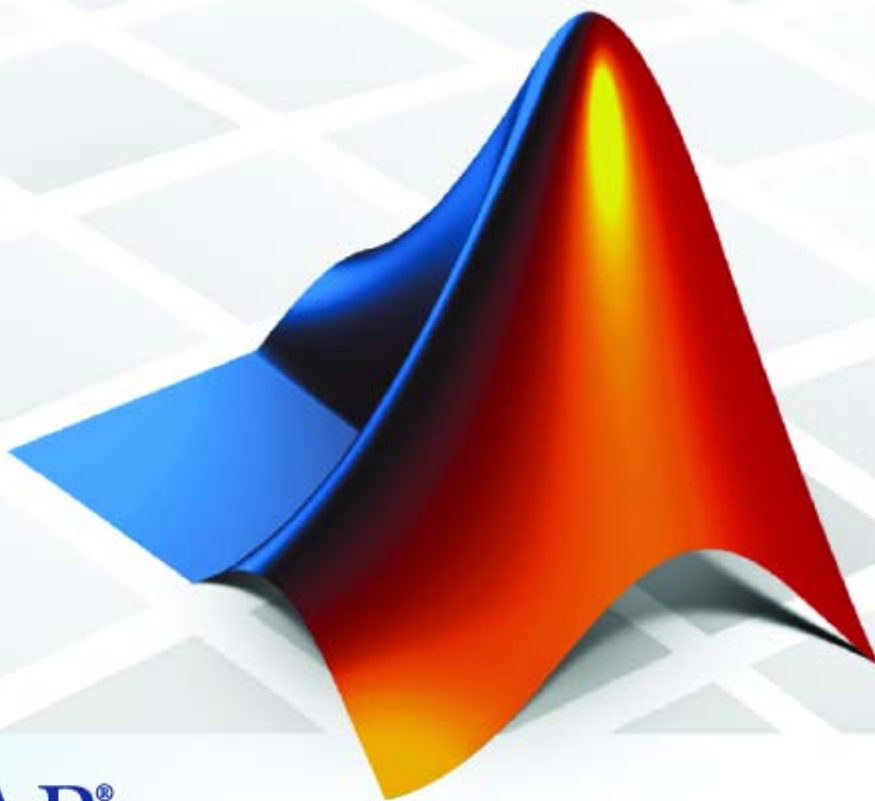
**How to Contact The MathWorks**

| | | |
|---|---|---|
| | www.mathworks.com | Web |
| | comp.soft-sys.matlab | Newsgroup |
| | www.mathworks.com/contact_TS.html | Technical Support |
| @ | suggest@mathworks.com | Product enhancement suggestions |
| | bugs@mathworks.com | Bug reports |
| | doc@mathworks.com | Documentation error reports |
| | service@mathworks.com | Order status, license renewals, passcodes |
| | info@mathworks.com | Sales, pricing, and general information |

☎ 508-647-7000 (Phone)

⌶ 508-647-7001 (Fax)

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

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

*Target for Freescale MPC5xx User's Guide*

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

# Contents

# Generating Stand-Alone Real-Time Applications

**2**

**PIL Cosimulation**

**3**

# Blocks — By Category

## 4

# Blocks — Alphabetical List

## 5

# Configuration Parameters

## 6

# Toolchains and Hardware

# A

# Examples

**B**

# Index

# Getting Started

This section contains the following topics:

# What Is Target for Freescale MPC5xx?

| In this section... |
| --- |
| "Introducing Target for Freescale MPC5xx" on page 1-3 |
| "Feature Summary" on page 1-3 |
| "Applications for Target for Freescale MPC5xx" on page 1-6 |

## Introducing Target for Freescale MPC5xx

Target for Freescale™ MPC5xx is an add-on product for use with the Real-Time Workshop® Embedded Coder. It provides a complete and unified set of tools for developing embedded applications for the Freescale MPC555 and MPC56x processors (MPC561, MPC562, MPC563, MPC564, MPC565 and MPC566). The MPC5xx family of processors are products of Freescale Semiconductor, Inc., formerly a division of Motorola, Inc.

Used in conjunction with Simulink®, Stateflow®, and the Real-Time Workshop Embedded Coder, Target for Freescale MPC5xx lets you

• Design and model your system and algorithms.

• Compile, download, run and debug generated code on the target hardware, seamlessly integrating with industry-standard compilers and development tools for the MPC5xx.

• Use cosimulation and rapid prototyping techniques to evaluate performance and validate results obtained from generated code running on the target hardware.

• Deploy production code on the target hardware.

## Feature Summary

### Production Code Generation

• The Real-Time Workshop Embedded Coder generates production code for use on the target MPC5xx microcontroller.

- The Real-Time Workshop Embedded Coder generates project or makefiles for popular cross-development systems:
  - Wind River Systems Wind River Compiler
  - Freescale CodeWarrior
- Debugger support:
  - Wind River Systems SingleStep debugger
  - Freescale CodeWarrior debugger
- Support for ANSI C (ANSI X3.159-1989) math library for floating-point functions.

## Device Driver Support

- The Target for Freescale MPC5xx Library provides device driver blocks that let your applications access on-chip resources. The I/O blocks support the following features of the MPC555 and MPC56x:
  - Pulse width modulation (PWM) generation via the Modular Input/Output Subsystem (MIOS) PWM unit or the Time Processor Unit 3 (TPU) modules
  - Analog input via the Queued Analog-to-Digital Converter (QADC64)
  - Digital input and output via the MIOS or TPU
  - Digital input via the QADC64
  - Frequency and pulse width measurement via the MIOS Double Action Submodule (MDASM)
  - Transmit or receive Controller Area Network (CAN) messages via the MPC5xx TouCAN modules
  - Driver blocks to support other functions of the TPU modules — Fast Quadrature Decode, New Input Capture/Input Transition Counter, and Programmable Time Accumulator
  - Serial transmit and receive
  - Utility blocks such as a watchdog timer

### Code and Performance Analysis

Web-viewable code generation report includes

- Analysis of RAM/ROM usage and other variables

- Analysis of code generation options used, with optimization suggestions

- Hyperlinks to all generated code files

- Hyperlinks from generated code to source model in Simulink

### Applications Development and Rapid Prototyping

- Generation of real-time, stand-alone code for MPC5xx

- Scheduler and time functions for singlerate or multirate real-time operation

- CAN-based loader for download of generated code to RAM or flash memory

- CAN-based host-target communications for non-real-time retrieval of data on host computer

### Simulation and Cosimulation

- Automatic S-function generation lets you validate your generated code in software-in-the-loop (SIL) simulation.

- Processor-in-the-loop (PIL) cosimulation lets you integrate generated code, running on the target processor, into your simulation.

- SIL and PIL code components are generated by the Real-Time Workshop Embedded Coder. These simulation components are in the same compact and efficient format as the production code generated for final deployment.

### CAN Support

- Transmit or receive CAN messages via the MPC5xx TouCAN modules.

- CAN Drivers (Vector) library provides blocks for transmitting, receiving, configuring, and connecting to Vector-Informatik CAN hardware and drivers. These can be used in simulation to connect to a real CAN bus.

- The CAN Message Blocks library includes blocks for transmitting, receiving, decoding, and formatting CAN messages. It also supports message specification via the Vector-Informatik CANdb standard. CAN is an industry standard protocol used in automotive electronics and many other embedded environments where dispersed components require sharing of information.

### Code Validation and Performance Analysis

**Code Validation.** Since signal data is available to Simulink during each sample interval in a PIL simulation, you can observe signal data on Scope blocks or other Simulink signal viewing blocks. You can also store signal data to MAT-files via To File blocks. To validate the results obtained by the generated code running on the target processor, you can compare these files to results obtained using a normal Simulink plant/controller simulation.

**Determining Code Size.** In control design it is critical to ensure that the size of the generated code does not exceed physical limitations of RAM and ROM. Target for Freescale MPC5xx can automatically produce a code generation report that displays the RAM usage and ROM size of the generated code.

This capability is useful when selecting which code generation optimizations will be used. After determining the size of the required RAM and ROM, you can consider which code generation optimizations to use, and consider modifications to the modeling style.

## Applications for Target for Freescale MPC5xx

Target for Freescale MPC5xx provides targets that support three application scenarios:

- Real-time (RT) execution for production and rapid prototyping
- Processor-in-the-loop (PIL) cosimulation target
- Algorithm export (AE) target

In the sections that follow, we summarize typical applications and the tasks you will need to perform for each; we also provide links to the relevant documentation.

## Real-Time Execution and Rapid Prototyping

The Target for Freescale MPC5xx real-time target enables you to use your controller block diagram in real time to perform embedded control. With this target, you can add I/O blocks for the MPC5xx to your controller subsystem, generate and build code, download to the target, and run the generated C code.

When you first begin using the RT target, see "Tutorial: Creating a New Application" on page 2-5, which demonstrates the following topics through the use of a simple model with a device driver:

- Examining the demo model with a plant model and controller
- Adding the MPC555 Resource Configuration block to your subsystem
- Adding I/O device drivers from the Target for Freescale MPC5xx library
- Selecting the RT target
- Generating code for real time
- Downloading code with
  - A BDM connector
  - CAN
- Running the generated code in real-time

You may also be interested in generating code analysis information from your RT target build. See "HTML Code Analysis (RAM/ROM) Report" on page 3-27 for details.

## Processor-in-the-Loop

The processor-in-the-loop (PIL) target lets you run a cosimulation of a closed-loop Simulink model for the purpose of code validation and analysis. When running a PIL cosimulation, you use a closed-loop model with two major components: a plant model and a controller. The plant model may contain any Simulink blocks including a combination of continuous-time and discrete-time blocks.

To get started with the PIL target, see "Tutorial 1: Building and Running a PIL Cosimulation" on page 3-6. The tutorial covers the following topics:

- Opening the demo model and examining the plant model and controller
- Selecting the PIL target
- Generating the Embedded Real-Time (ERT) S-function and the corresponding library block
- Inserting the S-function back into the closed-loop model
- Automatic downloading of generated code with
  - SingleStep debugger and a Background Debug Mode (BDM) port connector
  - CodeWarrior and a BDM connector
- Running a PIL cosimulation

You may also be interested in generating code analysis information from your PIL target build. See "HTML Code Analysis (RAM/ROM) Report" on page 3-27 for details.

### Algorithm Export

The Target for Freescale MPC5xx algorithm export (AE) target enables you to generate code for your controller subsystem and build the code as a stand-alone executable for use on the MPC5xx. The difference between the AE and the PIL target is that the AE target eliminates all extraneous code (such as serial communications code) used for cosimulation, and also eliminates any real-time interrupts. The AE target therefore generates code only for the basic controller subsystem (e.g. algorithm code). You can then modify or customize this code for your own special purposes.

In contrast, the RT target provides turnkey code including interrupt service routines, driver code, and underlying initialization code for the MPC5xx. Depending upon your particular application, you may find it more valuable to begin with the AE target baseline, and extend this environment for your own use.

The AE target is documented in "Algorithm Export Target" on page 3-25.

Like the PIL and RT targets, the AE target supports generation of code analysis information. See "HTML Code Analysis (RAM/ROM) Report" on page 3-27 for details.

# Additional Blocks on MATLAB Central

Check the MATLAB Central Web site for user- and developer-contributed blocks and demos, such as the MPC555 Motor Control Function Blockset for Release 2006a.

The MPC555 Motor Control Function Blockset is an extensive collection of additional TPU I/O blocks for Target for Freescale MPC5xx. This functionality is particularly useful in the context of motor and powertrain control, including functions for missing and additional tooth detection.

# Prerequisites

This document assumes you are experienced with MATLAB®, Simulink®, Stateflow®, Real-Time Workshop®, and Real-Time Workshop Embedded Coder.

Minimally, you should read the "Getting Started" section of the Real-Time Workshop documentation to understand general concepts and terminology related to Real-Time Workshop, and try some of the demos of the user interface, code generation and build process, and other essential features.

You should also familiarize yourself with the Real-Time Workshop Embedded Coder documentation.

In addition, if you want to understand and use the device driver blocks in the Target for Freescale MPC5xx library, you should have at least a basic understanding of the architecture of the MPC5xx processor you are using. The Freescale *MPC555 Users Guide* (or MPC561-564, or MPC565-566, as appropriate) is required reading. We recommend that you read the introduction to the processor and familiarize yourself with all the major subsystems of the MPC5xx. You can find this document at the following URL: `http://www.freescale.com/`.

# Using This Guide

We suggest the following path to get acquainted with Target for Freescale MPC5xx and gain hands-on experience with the features most relevant to your interests:

- Read Chapter 1, "Getting Started" in its entirety, paying particular attention to "Setting Up and Verifying Your Installation" on page 1-16.

- If you are interested in using the device driver blocks supplied with Target for Freescale MPC5xx and in deploying stand-alone, real-time applications on the MPC5xx, read Chapter 2, "Generating Stand-Alone Real-Time Applications" Work through the "Tutorial: Creating a New Application" on page 2-5.

- If you are interested in processor-in-the-loop (PIL) cosimulation, read Chapter 3, "PIL Cosimulation" to learn about the Target for Freescale MPC5xx PIL target. Work through the "Tutorial 1: Building and Running a PIL Cosimulation" on page 3-6.

- Then, for in-depth information about the device drivers and other blocks supplied with Target for Freescale MPC5xx, see Chapter 4, "Blocks — By Category" It is particularly important to read MPC555 Resource Configuration, as the MPC555 Resource Configuration block is required to use most of the device driver blocks.

See also the Target for Freescale MPC5xx Demos. To browse the demos, open the MPC555 Help and Demos library. You can then double click the **Help for Demos** block to go directly to information and instructions for all demos, or select **Start > Links and Targets > Target for Freescale MPC5xx > Demos**. You can also enter at the command line:

```
demo('Links and Targets','Target for Freescale')
```

These demos are used in the tutorials, where there are detailed explanations.

# Installing Target for Freescale MPC5xx

Your platform-specific MATLAB Installation guide provides all of the information you need to install Target for Freescale MPC5xx.

Prior to installing Target for Freescale MPC5xx, you must obtain a License File or Personal License Password from The MathWorks. The License File or Personal License Password identifies the products you are permitted to install and use.

As the installation process proceeds, it displays a dialog where you can select which products to install.

# Hardware and Software Requirements

| In this section... |
| --- |
| "Operating System Requirements" on page 1-13 |
| "Hardware Requirements" on page 1-13 |
| "Software Requirements" on page 1-14 |

## Operating System Requirements

Target for Freescale MPC5xx is a PC-Windows only product. The product has been tested on Microsoft Windows XP.

You can see the system requirements for MATLAB online at

`http://www.mathworks.com/products/system.shtml/Windows`

## Hardware Requirements

Programs generated by Target for Freescale MPC5xx can run on any Electronic Control Unit (ECU) that is based on the MPC555 or MPC56x (561-6) processor.

In this document, we specify settings and procedures for use with the Phytec phyCORE-MPC555 board, the Phytec MPC565, the Axiom MPC555, and the Axiom MPC564, in conjunction with specific cross-development environments, see "Setting Up Your Target Hardware" on page A-14 for details.

If you use a different development board, you may need to adapt these settings and procedures for your development board.

If you want to use CAN to connect to your target you require Vector-Informatik CAN hardware and drivers. See "CAN Hardware and Drivers" on page A-21.

## Software Requirements

### Required and Related MathWorks Products

Target for Freescale MPC5xx *requires* these MathWorks products:

- MATLAB

- Simulink

- Real-Time Workshop

- Real-Time Workshop Embedded Coder

  - Optional for Real-Time Target

  - Required for Processor-in-the-Loop and Algorithm Export Targets.

  - Required for CCP Data Acquisition (DAQ) List mode of operation.

For more information about any of these products, see either

- The online documentation for that product, if it is installed

- The MathWorks Web site, at `http://www.mathworks.com`; see the Products section

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with Target for Freescale MPC5xx. For required and related products, see: `http://www.mathworks.com/products/target_mpc555/`

### Supported Cross-Development Tools

In addition to the required MathWorks software, a supported cross-development environment is required. Target for Freescale MPC5xx currently supports the cross-development tools listed below; please read carefully the limitations noted:

- Wind River Systems Wind River Compiler version 5.4.0, (formerly known as Diab), and Wind River Systems SingleStep debugger of the following versions:

  - Version 7.7.5 (debug via Wind River visionPROBE) (for MPC5xx)

  - Version 7.6.6 (debug via Macraigor Systems Wiggler, Raven / Blackbird, On-board BDM) (for MPC555 only) (+ Fromelf patch from Wind River Support)

- Freescale CodeWarrior Development Studio, MPC5xx Edition, v8.7

  The full feature set (PIL, RT, and AE targets) is supported for both toolchains.

Before using Target for Freescale MPC5xx with any of the above cross-development tools, please be sure to read and follow the instructions in "Setting Up and Verifying Your Installation" on page 1-16.

# Setting Up and Verifying Your Installation

The next sections describe how to configure your development environment (compiler, debugger, etc.) for use with Target for Freescale MPC5xx and verify correct operation. The initial configuration steps are described in the following sections:

- You must set up your development environment and your target hardware. Information on these settings can be found in the Appendix A, "Toolchains and Hardware":

  - "Setting Up Your Target Hardware" on page A-14

  - "Setting Up Your Toolchain" on page A-3

  ---

  **Note** You MUST check your jumper settings. Incorrect operation or even hardware damage may occur if you do not. See "Jumper Settings" on page A-14.

  ---

- You must configure Target for Freescale MPC5xx to work with your toolchain by specifying the locations of your compiler and debugger. This is described in the section "Setting Target Preferences" on page 1-17.

- We supply a test program to verify your installation. This confirms you have correctly set up your toolchain, target preferences and development board. See "Run Test Program" on page 1-23.

- The next step is to download boot code to the flash memory of your MPC5xx. See "Download Boot Code to Flash Memory" on page 1-24.

  ---

  **Note** You must download the new boot code if you have used a previous release of Target for Freescale MPC5xx with your hardware. See "Download Boot Code to Flash Memory" on page 1-24.

  ---

Once you have completed these steps we suggest you run the tutorials in subsequent sections to get started with Target for Freescale MPC5xx.

# Setting Target Preferences

### In this section...

## Configuring Target for Freescale MPC5xx for Your Cross-Development Toolchain

This section describes how to set target preferences associated with Target for Freescale MPC5xx. These settings persist across MATLAB sessions and different models. Target preferences let you specify the location of your MPC5xx cross-compiler, the communications port to be used for downloading code, and other parameters affecting the generation, building, and downloading of code.

When you set up Target for Freescale MPC5xx, you must make sure you localize the settings to suit your PC and cross-development toolchain. It is important that you set the correct path to your compiler and debugger using the **MPC5xx Target Preferences Setup** dialog box.

Instructions for setting up specific third-party toolchains for use with Target for Freescale MPC5xx are in Appendix A, "Toolchains and Hardware". Make sure you have followed the instructions to set up your toolchain first:

- "Setting Up Your Installation with Wind River Compiler and SingleStep Debugger" on page A-4

  - "Setting Target Preferences for Wind River Compiler and SingleStep" on page A-6. Note especially the settings you must change if you are not using the visionPROBE BDM device. The defaults are set up for the visionPROBE.

- "Setting Up Your Installation with Freescale CodeWarrior" on page A-10

  - "Set Target Preferences for CodeWarrior" on page A-12

You can modify target preference objects via the **MPC5xx Target Preferences Setup** dialog box:

**1** Select **Start > Links and Targets > Target for Freescale MPC5xx > MPC5xx Target Preferences**.

This opens the **MPC5xx Target Preferences Setup** dialog box where you can edit the settings for your cross-development environment. When you first open the dialog the following settings are visible.
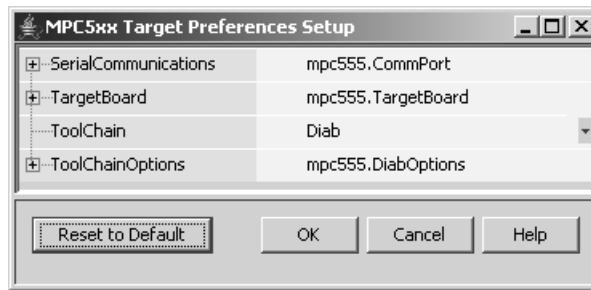


**2** Select Diab or CodeWarrior from the drop-down **Toolchain** menu.

Note the Wind River Compiler was formerly known as Diab. Any appearances of the term *Diab* in the documentation and / or product should be understood to refer to the Wind River Compiler.

**3** Expand **ToolChainOptions** as shown below (by clicking the plus sign) and type the correct path into **CompilerPath**. The following shows Wind River Compiler options. Note that the defaults are set up for the visionPROBE — see the Appendix for settings to use another BDM device, described in "Setting Target Preferences for Wind River Compiler and SingleStep" on page A-6.
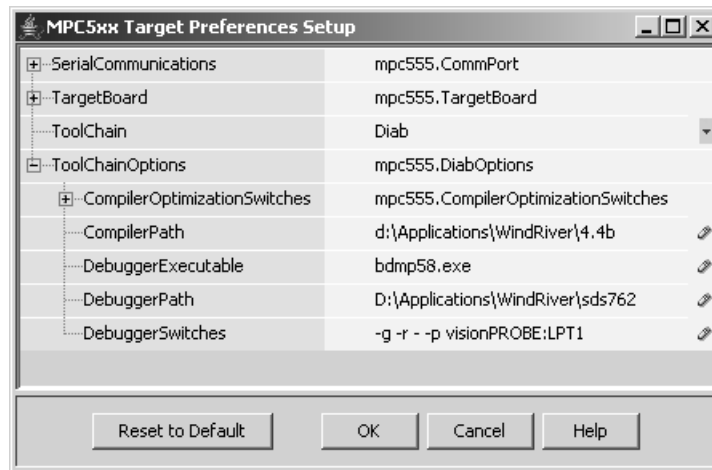
**Note** The drive designated in the compiler and debugger paths must be either an actual hard drive on your PC, or a mapped drive. Do not use a Universal Naming Convention (UNC).

**4** For SingleStep you must also type the correct path into **Debugger Path**. This is not necessary for CodeWarrior as the compiler and debugger are integrated. The example below shows the CodeWarrior preferences.
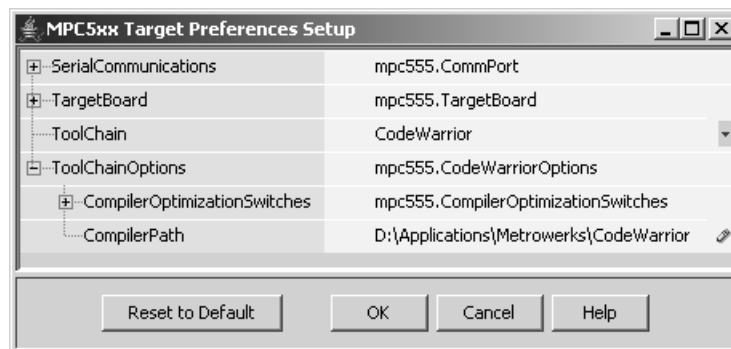


There are other settings in the target preferences you can see by expanding all the options, as shown.
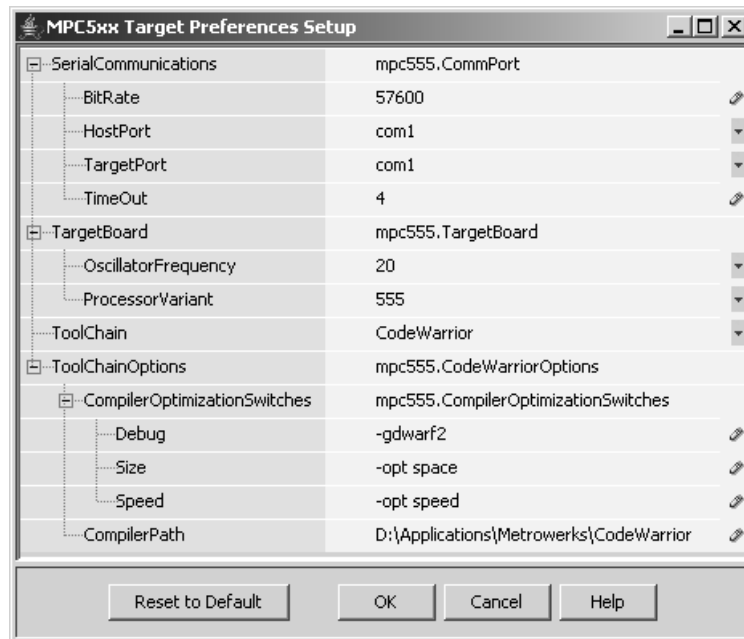
### Serial Communications

These target preferences relate to Processor-in-the-Loop (PIL) cosimulation only.
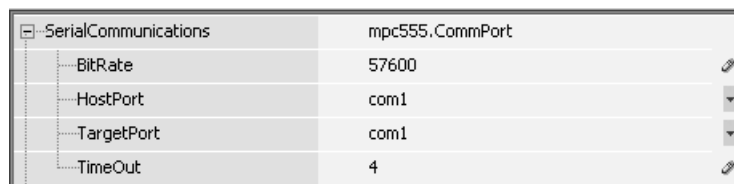


- BitRate — Bit rate (in bps) for host/target communications. The default is 57600.

- HostPort — Host serial port for host/target communications. Select from com1 to com8; the default is com1.

- TargetPort — Target board serial port for host/target communications. Select from com1 to com8; the default is com1.

- `TimeOut` — Time-out value (in seconds) for the serial communications port. The default is 4.

## Target Board

| TargetBoard | mpc555.TargetBoard | |
|---|---|---|
| OscillatorFrequency | 20 | ▾ |
| ProcessorVariant | 555 | ▾ |

- `OscillatorFrequency` — Choose either 20 MHz (the default) or 4 MHz if you are using a 4MHz board.

- `ProcessorVariant` — Here you can select from 555, 561, 562, 563, 564, 565 or 566 to match your target processor. The default is the MPC555.

When you install bootcode after setting target preferences the correct bootcode for your chosen target processor and oscillator frequency will be automatically installed. Note that you also need to make these settings match in your models for the non-default target processor and oscillator frequency. See "Configuration for Nondefault Hardware" on page A-23.

## Compiler Optimization Switches

| ToolChainOptions | mpc555.DiabOptions | |
|---|---|---|
| CompilerOptimizationSwitches | mpc555.CompilerOptimizationSwitches | |
| Debug | -g | 🖉 |
| Size | -XO -Xsize-opt | 🖉 |
| Speed | -XO | 🖉 |
| CompilerPath | d:\Applications\WindRiver\4.4b | 🖉 |
| DebuggerExecutable | bdmp58.exe | 🖉 |
| DebuggerPath | D:\Applications\WindRiver\sds762 | 🖉 |
| DebuggerSwitches | -g -r - -p visionPROBE:LPT1 | 🖉 |
| ToolChainOptions | mpc555.CodeWarriorOptions | |
| CompilerOptimizationSwitches | mpc555.CompilerOptimizationSwitches | |
| Debug | -gdwarf2 | 🖉 |
| Size | -opt space | 🖉 |
| Speed | -opt speed | 🖉 |
| CompilerPath | D:\Applications\Metrowerks\CodeWarrior | 🖉 |

For both toolchains these settings configure optimizations for speed, size, and debug. The settings are compiler specific. These properties can be edited from the **MPC5xx Target Preferences Setup** dialog box or from the **Configuration Parameters** dialog box, described below. The defaults should be adequate for most rapid prototyping purposes.

If you want to alter these settings, consult your compiler documentation for specific optimizations. To edit the settings,

- If you want your changes to apply to many models, edit them within the **MPC5xx Target Preferences Setup** dialog box. Your settings will appear within the **Configuration Parameters** dialog box in the **Compiler optimization switches** field when you select speed, size or debug from the **Optimize compiler for** options in the drop-down menu. You must choose MPC5xx real-time options (1) from the **Real-Time Workshop** tree to reach these settings, as shown in the following example.
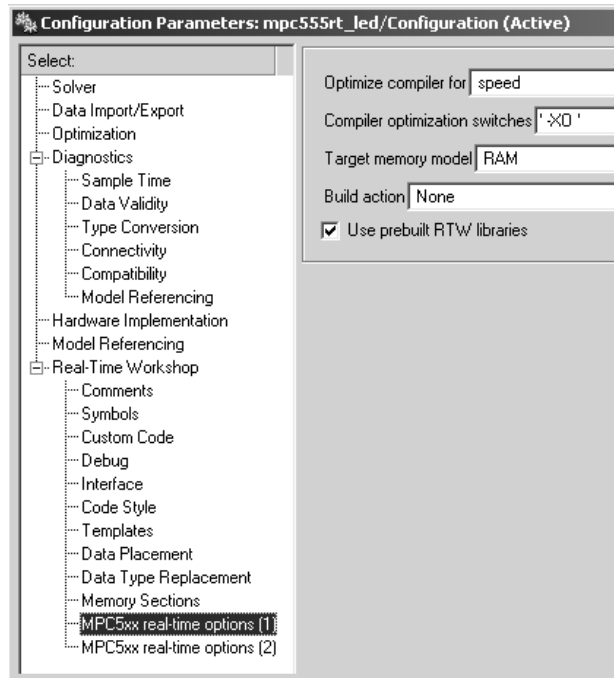
- If you want to customize these settings for a single model, edit them from the **Configuration Parameters** dialog box. **Optimize compiler for** will change to custom and the defaults for these settings will remain unchanged in the **MPC5xx Target Preferences Setup** dialog box. When you edit these settings, you must place single quotation marks at either end of the string. These settings are then applied to model code.

**Use Prebuilt RTW Libraries.** This check box option (selected by default) determines whether prebuilt RTW libraries, compiled with default compiler switches, are linked against during compilation of the generated code. When this option is not selected, the source modules that comprise these libraries will be compiled individually in the model build directory, using the currently selected compiler switches. Using prebuilt RTW libraries saves a considerable amount of time during the build process

### Debugger Switches

This setting is specific to SingleStep. See "Setting Target Preferences for Wind River Compiler and SingleStep" on page A-6.

## Run Test Program

To verify your setup, you can download and run a simple test program on the phyCORE-MPC555 board:

**1** Select **Start > Links and Targets > Target for Freescale MPC5xx > Run Simple MPC555 Test Application**.

**2** To answer the question Do you want to run the application?  Type y at the command line.

If you have not set up your target preferences properly the process will stop and ask you to do this now.

Watch as your toolchain downloads and runs the application on your phyCORE board. Successful execution results in a blinking LED.

You have now verified your installation and are ready to begin working with Target for Freescale MPC5xx.

## Download Boot Code to Flash Memory

The next step is to download the boot code to flash memory, if you have not already done so. Normally, you will only need to program the boot code into flash memory once. After this is done, new application code can be downloaded as often as required without any changes to the boot code.

The first time you program the boot code into the target hardware, you must download it via the BDM port. However, if existing boot code is already programmed into flash memory and must be replaced (for example, with a newer or modified version) it is also possible to download entirely over CAN or serial. If you are upgrading from a previous release of Target for Freescale MPC5xx you must download the new boot code.

If your target does not have bootcode already you can only install new bootcode with a BDM. See the next section "Installing Bootcode via BDM and Serial or CAN" on page 1-24. For existing bootcode, you can use a BDM or CAN; with bootcode from version 1.2 or later you can also download over Serial. See "Installing Bootcode Without a BDM" on page 1-25.

The first time you use Target for Freescale MPC5xx you must use a toolchain to download boot code to the MPC555 flash memory. Once the boot code is loaded into flash memory, you can download code to the processor entirely over serial or the CAN network as described in the tutorials. See "Overview of Memory Organization and the Boot Process" on page 2-21 for more information.

### Installing Bootcode via BDM and Serial or CAN

To install bootcode, follow these steps:

**1** Connect the BDM cable to the target, and a serial or CAN cable. If you do not have a BDM available, see "Installing Bootcode Without a BDM" on page 1-25.

**2** Select **Start > Links and Targets > Target for Freescale MPC5xx > Install MPC5xx Bootcode**.

A dialog appears asking if you are connected to the target via BDM. Read the information on the dialog.

**3** Click **Yes**.

Your toolchain is launched and prepares to download.

The **Download Control Panel** appears.

**4** If you are using CAN (the default) you can proceed to step 5. If you are using serial to connect to the target, click the **Communications Options** tab in the **Download Control Panel** and select Serial from the **Connection type** drop-down menu.

**5** On the **Download** tab, click **Start Download**.

Your development tools execute a command to install the boot code. When the process stops, the messages in the **Download Control Panel** complete, and the **Stop Download** button reverts to **Start Download**. The boot code should now be installed.

### Installing Bootcode Without a BDM

If your target does not have bootcode already you can only install new bootcode with a BDM. For targets with existing bootcode, if you do not have a BDM available you can install bootcode as follows:

• For a target with R14 bootcode, you can install new bootcode using the **Start** menu exactly as described above except step 4 - click **No** when asked if you are connected via BDM. The download should complete successfully over serial or CAN.

• If existing bootcode on the target is version 1.1 (R13+SP1), you can install bootcode without a BDM if you have CAN. Use the **Start** menu bootcode installer as described above and click **No** when asked if connected by BDM. The download should complete successfully over CAN.

• If the existing bootcode is earlier than version 1.1 (if it is R12.1 or R13), you can install bootcode without a BDM if you have CAN. You cannot use the **Download Control Panel**. Instead you must use the upgrade model, can_bootcode_upgrade.mdl.

Once you have successfully downloaded boot code to your target, you have completed your installation and are ready to use all the features of Target for Freescale MPC5xx. If necessary, please consult your toolchain documentation.

We suggest you now turn to Chapter 2, "Generating Stand-Alone Real-Time Applications" to get hands-on experience with using Target for Freescale MPC5xx and your toolchain to generate, download, and execute application code on your phyCORE-MPC555 board. You can then also work through the tutorials in Chapter 3, "PIL Cosimulation" to start using processor-in-the-loop simulation for development via Target for Freescale MPC5xx.

# Start Menu Options

You can use the **Start** menu for the following options:

- **Help** — Opens the Help Browser and displays the product documentation.

- **Environment Setup** — Opens the Help Browser displaying the documentation for setting up and verifying your installation.

- **Demos** — Opens the product demos help with links to open the demos.

- **MPC5xx Driver Library** — Opens the MPC5xx Drivers block library. See "MPC555 Drivers" on page 4-2.

- **CAN Message Blocks** — Opens the CAN Message Blocks library. See "CAN Message Blocks and CAN Drivers" on page 4-7.

- **CAN Drivers (Vector) Library** — Opens the CAN Drivers (Vector) block library. See "CAN Message Blocks and CAN Drivers" on page 4-7.

- **MPC5xx Target Preferences** — Opens the Target Preferences dialog. See "Setting Target Preferences" on page 1-17.

- **Install MPC5xx Bootcode** — Installs the appropriate boot code on your target processor. See "Download Boot Code to Flash Memory" on page 1-24.

- **Run Simple MPC5xx Test Application (via BDM)** — Downloads and runs a simple test application to blink an LED with your hardware. See "Run Test Program" on page 1-23.

- **Inspect the MPC5xx Hardware (via BDM)** — Opens your debugger so you can inspect the hardware.

- **Debug RAM Based Application (via BDM)** — Downloads and then allows you to debug a RAM application in `.elf` format.

- **Debug FLASH Based Application Already in FLASH (via BDM)** — Allows you to debug an application (in `.elf` format) already in FLASH.

- **Download RAM / FLASH Based Application (via CAN / Serial)** — Launches the Download Control Panel, for downloading applications in `.s19` format to your hardware. See "Tutorial: Creating a New Application" on page 2-5, and "Downloading Application Code" on page 2-23.

**1-27**

- **Download FLASH Based Application (via BDM and CAN / Serial)** — Allows you to use a BDM and the Download Control Panel to download an application in .s19 format to FLASH memory. See "Downloading Application Code" on page 2-23.

- **Initialize visionPROBE for Selected Target Board (WindRiver Only)** — If you are using a visionPROBE, you must run this option to initialize the device, after setting target preferences (and again if you change target processor). See "Initialize visionPROBE" on page A-8.

- **Rebuild the MPC5xx Driver Library** — Recompiles the MPC5xx Driver libraries. See "Boot Code Parameters for CAN Download" on page 2-30.

- **Product Page (Web)** — Opens the product page on The MathWorks web site.

# Data Type Support and Scaling for Device Driver Blocks

The following table summarizes the input and output data types supported by the device driver blocks in the Target for Freescale MPC5xx library and the scaling applied to block inputs and outputs.

**I/O Data Types and Scaling for MPC5xx Device Driver Blocks**

| Block | Input Data Type | Input Scaling | Output Data Type | Output Scaling/ Units |
|---|---|---|---|---|
| MIOS Digital In | | | `Boolean` | 0 or 1 only |
| MIOS Digital Out | Any Simulink supported data type | logic 1 if input > 0, logic 0 if input <= 0 | | |
| MIOS Digital Out (MPWMSM) | Any Simulink supported data type | logic 1 if input > 0, logic 0 if input <= 0 | | |
| MIOS Pulse Width Modulation Out | `double` or `single` | 0 to 1 | | |
| MIOS Waveform Measurement | | | `double` or `single` | Seconds |
| QADC Analog In | | | `uint16` or `int16` (defined by **Justification** parameter) | (defined by **Justification** parameter) |
| QADC Digital In | | | `Boolean` | 0 or 1 only |
| TouCAN Receive | | | `CAN_MESSAGE_STANDARD` or `CAN_MESSAGE_EXTENDED` | N/A |

**I/O Data Types and Scaling for MPC5xx Device Driver Blocks (Continued)**

| Block | Input Data Type | Input Scaling | Output Data Type | Output Scaling/ Units |
|---|---|---|---|---|
| TouCAN Transmit | `CAN_MESSAGE_STANDARD` or `CAN_MESSAGE_EXTENDED` | N/A | | |
| TouCAN Warnings | | | `Boolean` | N/A |
| TouCAN Error Count | | | `uint8` | N/A |
| TouCAN Fault Confinement State | | | `uint16` | N/A |
| TPU3 Digital In | | | `Boolean` | 0 or 1 only |
| TPU3 Digital Out | Any Simulink supported data type | Logic 1 if input > 0, logic 0 if input <= 0 | | |
| TPU3 Fast Quadrature Decode | Fast Mode input `Boolean` | | `uint16` | N/A |
| TPU3 New Input Capture/Input Transition Counter | | | `uint16` | N/A |
| TPU3 Programmable Time Accumulator | | | Time Accumulation `uint32` Period Count `uint8` | N/A |

**I/O Data Types and Scaling for MPC5xx Device Driver Blocks (Continued)**

| Block | Input Data Type | Input Scaling | Output Data Type | Output Scaling/ Units |
|---|---|---|---|---|
| TPU3 Pulse Width Modulation Out | Duty cycle input (top if 2 inputs): `double` or `single` | 0 to 1 | | |
| | Pulse period register input — `uint16` | Saturated to be in the range 0 to 32768 | | |
| Serial Transmit | Data: `uint8` (vector or scalar)<br><br>Number of bytes: `uint32` (scalar) | N/A | Number of bytes: `uint32` | 0-16 (for SCI1); 0 or 1 (for SCI2) |
| Serial Receive | Number of bytes: `uint32`<br><br>Reset: `Boolean` | N/A<br><br>0 or 1 | Data: `uint8`<br><br>Actual number of bytes: `uint32`<br><br>Framing and parity error: `Boolean`<br><br>Overrun flag: `Boolean` | N/A<br><br>N/A<br><br><br>0 or 1<br><br>0 or 1 |

**Configuration Class Blocks**

Each sublibrary of the Target for Freescale MPC5xx library contains a *configuration class block* that has an icon similar to the one shown in this figure.



MPC555
Configuration
Class

Configuration class blocks exist only to provide information to other blocks. *Do not copy these objects into a model.*. If you do you see an error dialog box to warn you. This causes build failures.

# Generating Stand-Alone Real-Time Applications

This section includes the following topics:

# Overview

## Generating Real-Time Applications

This section describes how to generate a stand-alone real-time application for the MPC555. The components required to generate stand-alone code are

- The Target for Freescale MPC5xx real-time target

- The MPC555 Resource Configuration object provided in the Target for Freescale MPC5xx library

- I/O driver blocks provided in the Target for Freescale MPC5xx library

- Utilities for downloading generated code to the target hardware

Using these together with your toolchain, you can build a complete application. You do not need to hand-write any C code to integrate the generated code into a final application.

See "Before You Begin" on page 2-6 for information on supported hardware and toolchains.

The tutorial "Tutorial: Creating a New Application" on page 2-5 uses two blocks from the Target for Freescale MPC5xx library. For complete information on the Target for Freescale MPC5xx library blocks, see Chapter 4, "Blocks — By Category".

Before reading this section and using the Target for Freescale MPC5xx library, you should have at least a basic understanding of the architecture of the MPC555. To learn about the MPC555, we suggest that you study the *MPC555 Users Manual*. We recommend that you read the introduction to the processor and familiarize yourself with all the major subsystems of the MPC555. You can find this document at the following URL:

```
http://www.freescale.com/files/microcontrollers/doc/user_guide/MPC555UM.pdf
```

## Deploying Generated Code

You can load a generated program into the MPC555 flash memory for permanent deployment. You can also load your code into external RAM (if available on your development hardware).

Alternatively, you can use the automatic code generation process for rapid prototyping and investigate a range of different design alternatives before making a deployment decision.

Your generated program can run on any Electronic Control Unit (ECU) that is based on the MPC5xx processor. Your application can use any of the supported MPC5xx on-chip I/O devices. We provide driver blocks for the MPC5xx's MIOS, TPU, QADC and TouCAN modules, providing you with drivers for the on-chip analog input, digital I/O, PWM, serial and CAN devices.

See Chapter 4, "Blocks — By Category" for further information on the device driver blocks in the Target for Freescale MPC5xx library.

In addition to on-chip I/O resources, an ECU typically provides additional I/O devices. If you want to access such custom I/O devices, you must write device drivers and integrate them with the automatically generated code. See the following documentation for details:

- Real-Time Workshop User's Guide

- Real-Time Workshop Embedded Coder User's Guide

- Writing S-Functions

Once the application has been programmed into memory on the target system, you may need to monitor signals or tune parameters. Target for Freescale MPC5xx supports signal monitoring and parameter tuning via Simulink external mode or a third party calibration tool. In both cases you must include a CAN Calibration Protocol (CCP) block in your model. The CAN Calibration Protocol block implementation of CCP has been tested against CANape from Vector-Informatik and ATI Vision. See "Parameter Tuning and Signal Logging" on page 2-33 and CAN Calibration Protocol (MPC555) for further information.

# Tutorial: Creating a New Application

| In this section... |
| --- |
| |
| |
| |
| |
| |
| |

## Tutorial Overview

In this tutorial, you build a stand-alone real-time application from a model incorporating blocks from the Target for Freescale MPC5xx library. We assume that you are already familiar with Simulink and with the Real-Time Workshop code generation and build process.

In the following sections, you will

- Configure the model

- Generate code from a subsystem

- Download code by one of the following methods:
  - Download to target RAM via a serial connection, using the **Download Control Panel** utility (provided with Target for Freescale MPC5xx)
  - Download to target RAM via a CAN connection, using the **Download Control Panel** utility
  - Download to target RAM via a BDM connection

- Execute the code on the target

After you complete this tutorial, you may want to learn how to deploy generated code into the MPC555 flash memory. See "Downloading Boot and Application Code" on page 2-19 for that information.

## Before You Begin

This tutorial requires the following specific hardware and software in addition to Target for Freescale MPC5xx:

- Phytec phyCORE-MPC555 development board

  The tutorial model utilizes two LEDs on the phyCORE-MPC555 board. These LEDs are connected to pins `MPIO32B0` and `MPIO32B1` on the MPC555 MIOS digital output pins. If you are using a different development board, you may be able to obtain the same functionality by making similar connections.

- A supported toolchain for compiling and debugging. Currently supported toolchains are

  - Wind River Compiler and SingleStep from Wind River Systems
  - CodeWarrior from Freescale

  See "Setting Up Your Toolchain" on page A-3 for details.

- Hardware to enable downloading:

  - If you want to download generated code to the target board over serial you will need a serial cable to connect your host PC to the target board.
  - If you want to download over BDM you will need a BDM device.
  - If you want to download via CAN, you will need a supported CAN card and drivers from Vector-Informatik. See "CAN Hardware and Drivers" on page A-21.

### Configuring Target for Freescale MPC5xx

- Make sure that your target preferences are set correctly for your development tools. See "Setting Target Preferences" on page 1-17.

- Once your target preferences are set for your toolchain you must download bootcode to the target before you can work through this tutorial. See "Download Boot Code to Flash Memory" on page 1-24.

## The Example Model

In this tutorial we will use a simple example model, mpc555rt_led, from the directory *matlabroot*/toolbox/rtw/targets/mpc555dk/mpc555demos.

This directory is on the default MATLAB path. The path *matlabroot* is the location where MATLAB is installed.

**1** Open the model.

```
mpc555rt_led
```

**2** Save a local copy to your working directory. We will work with this copy throughout this exercise.

mpc555rt_led_demo Model, Root Level on page 2-7 shows the example model at the root level. We will only use this level in simulation.
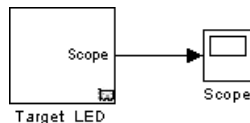


**mpc555rt_led_demo Model, Root Level**

**3** Double-click on the Target_LED subsystem block.

Target_LED Subsystem on page 2-7 shows the Target_LED subsystem, from which we will generate code.
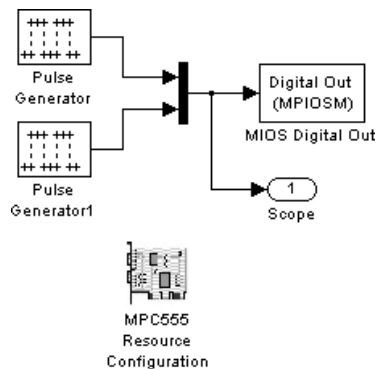


**Target_LED Subsystem**

In the Target_LED subsystem, two square wave signals are multiplexed and routed to the MIOS Digital Out block. The MIOS Digital Out block accepts a vector of numbers representing pins 0-15 on the MIOS 16-bit Parallel Port I/O Submodule (MPIOSM) on the MPC555. As the square wave signals oscillate between 0 and 1, the MIOS Digital Out block writes corresponding logic values to the appropriate pin on the port.

This figure shows the parameters of the MIOS Digital Out block.
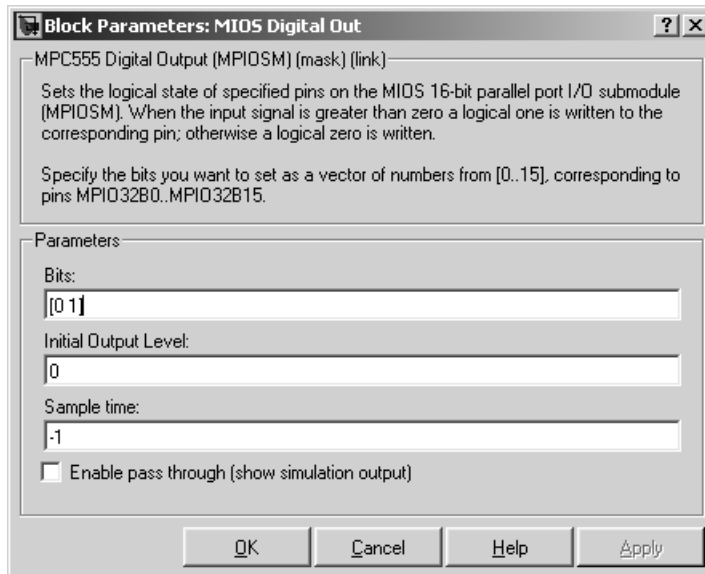


The **Bits** field is set to the vector [0 1]. The block maps this vector to the MPC555 MIOS digital output pins MPIO32B0 and MPIO32B1. When the application runs, it will send a pulse signal to these output pins. On the phyCORE-MPC555 board, these signals are connected to two of the LEDs, which will switch on and off at the frequency set in the respective pulse generator blocks.

In addition to the Pulse Generator, Mux, MIOS Digital Out, and Output blocks, the Target_LED subsystem contains a MPC555 Resource Configuration object. When building a model with driver blocks from the Target for Freescale MPC5xx library, you must always place a MPC555 Resource Configuration

object into the model (or the subsystem from which you want to generate code) first.

The purpose of the MPC555 Resource Configuration object is to provide information to other blocks in the model. Unlike conventional blocks, the MPC555 Resource Configuration object is not connected to other blocks via input or output ports. Instead, driver blocks (such as the MIOS Digital Out block in the example model) query the MPC555 Resource Configuration object for required information.

For example, a driver block may need to find the system clock speed that is configured in the MPC555 Resource Configuration object. The MPC555 has a number of clocked subsystems; to generate correct code, driver blocks need to know the speeds at which these clock busses will run.

The MPC555 Resource Configuration window lets you examine and edit the MPC555 Resource Configuration settings. To open the MPC555 Resource Configuration window, double-click on the MPC555 Resource Configuration icon. This figure shows the **MPC555 Resource Configuration** window for the Target_LED subsystem.
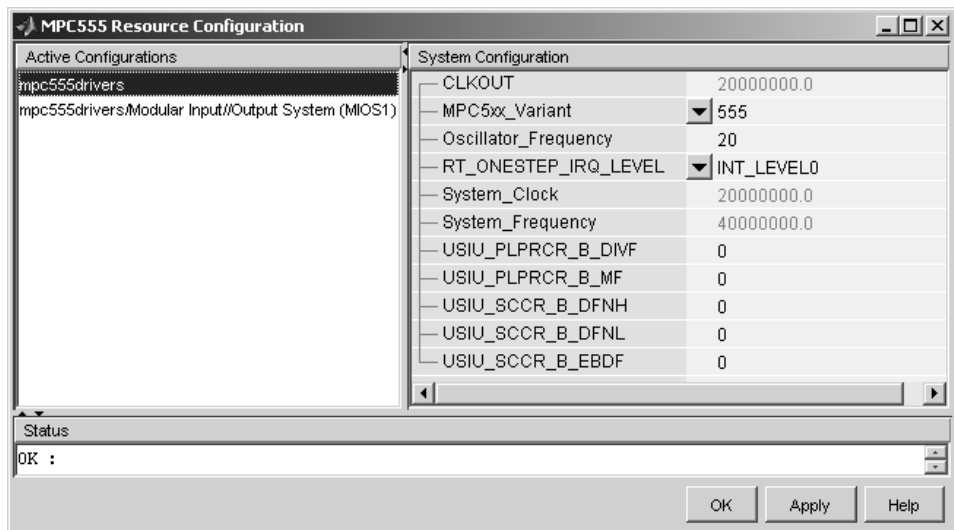
In this tutorial, we will use the default MPC555 Resource Configuration settings. Observe, but do not change, the parameters in the MPC555 Resource Configuration window. To learn more about the MPC555 Resource Configuration object, see MPC555 Resource Configuration.

Close the **MPC555 Resource Configuration** window before proceeding.

The next step in this tutorial is generating code.

## Generating Code

We will now look at settings and then generate application code:

**1** Select **Simulation > Configuration Parameters**. The **Configuration Parameters** dialog opens.

**2** Select **Real-Time Workshop** in the tree, as shown below.
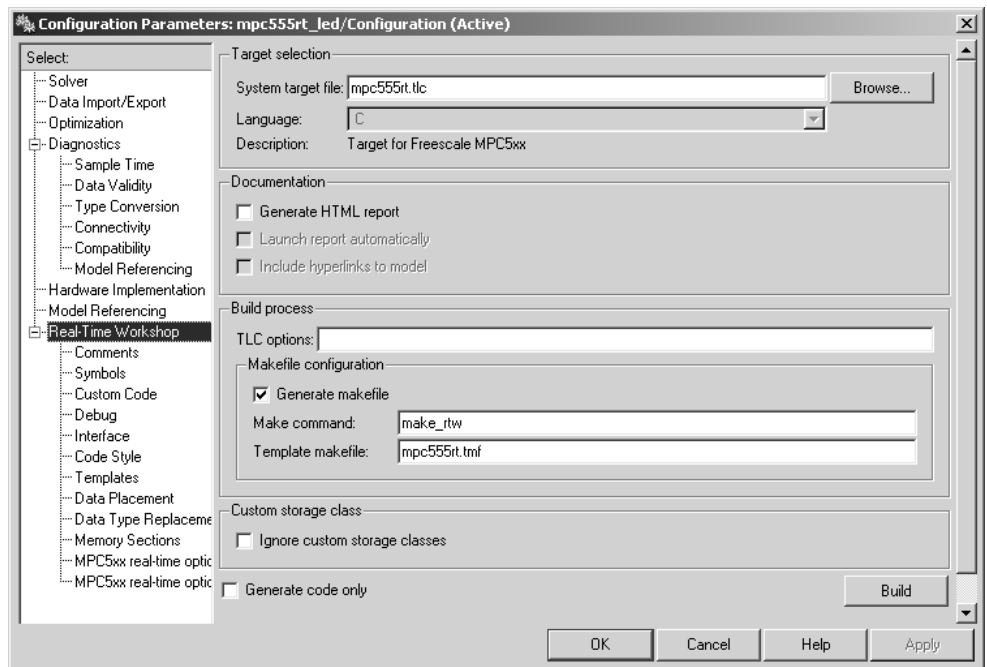
**3** Notice the **RTW system target file** for real-time deployment is
mpc555rt.tlc.

To see how to change from real-time deployment to processor-in-the-loop
or algorithm export, click on the **Browse** button to open the **System
Target File Browser**. In the browser, observe the Target for Freescale
MPC5xx options. Click **Cancel** to keep the default real-time setting and
return to the **Real-Time Workshop** pane.

**4** Select MPC5xx real-time options (1) in the tree. The RAM option should
be selected from the **Target memory model** menu. This option directs
Real-Time Workshop to generate a code file suitable for downloading
and execution in RAM. The files for both RAM and flash are in Motorola
S-record format.

Leave the options set to their defaults. The code generation options should
appear as shown below (though optimization switches settings vary
between toolchains).



Optimize compiler for | speed
Compiler optimization switches | '-XO '
Target memory model | RAM
Build action | None
☑ Use prebuilt RTW libraries

**5** You are now ready to build the application. Do this by right-clicking on the
Target_LED subsystem and selecting **Real-Time Workshop > Build
Subsystem**. Then click the **Build** button in the following dialog.

**6** On successful completion of the build process, two files are created in the
working directory:

**a** Target_LED_ram.s19: This file is for serial or CAN download. It is code
only, without symbols, suitable for execution on the target system.

**b** Target_LED_ram.elf: This file is for BDM download.

If debug is selected in the compiler optimization settings, the elf file will
contain debugging symbols as well as code. These symbols are suitable
for use with a symbolic debugger such as Wind River SingleStep or

Freescale CodeWarrior. The default optimization setting is speed, so no symbols are included. Symbols are only generated for a debug build. See "Compiler Optimization Switches" on page 1-21.

You can download to RAM:

- Via Serial or CAN, using the **Download Control Panel** utility (with Vector-Informatik hardware if you are using CAN), as described in "Downloading the Application to RAM via Serial or CAN" on page 2-12.

- Via the BDM port, as described in "Downloading the Application to RAM via BDM" on page 2-17.

## Downloading the Application to RAM via Serial or CAN

The **Download Control Panel** utility can be used to download application code to MPC555 RAM or to MPC555 flash memory.

In this section, you will use the **Download Control Panel** utility to download the generated Target_LED_ram.s19 file to RAM on the target system. The s19 file is for download over serial or CAN.

Target_LED_ram.elf is for BDM download, as described in the next section, "Downloading the Application to RAM via BDM" on page 2-17. Recall you can perform a debug build to include debugging symbols in the elf file.

Do the following before you begin:

- If you are using serial, make sure you have connected a serial port on your PC to serial port 1 (RS232-1) on the target hardware.

- If you are using CAN, make sure that your Vector-Informatik CAN card and drivers are installed and configured properly. See "CAN Hardware and Drivers" on page A-21. Make sure that the desired CAN port on the PC card is connected to the CAN A port on the target hardware.

- Make sure that you have set up your toolchain as described in "Setting Up Your Toolchain" on page A-3, and downloaded boot code to the flash memory of the MPC555 as described in "Download Boot Code to Flash Memory" on page 1-24.

- Make sure that nothing is connected to the BDM port of your development board.

- Make sure that the jumpers on the phyCORE-MPC555 board are set as described in "Phytec MPC555 Jumper Settings" on page A-15.

- Cycle the power (or perform a hard reset) on your development board, to clear the RAM.

To download the generated Target_LED_ram.s19 file to RAM:

**1** Start the **Download Control Panel** in one of the following ways:

- Use the MATLAB **Start** menu. Select **Start > Links and Targets > Target for Freescale MPC5xx > Download RAM / FLASH-Based Application (via CAN / Serial)**.

- Type embedded_target_download at the MATLAB command prompt.

- You can also open the **Download Control Panel** automatically at the end of the build process. Before you start the build, you can select **Launch Download Control Panel** from the **Build action** options on the MPC5xx real-time options (1) pane of the **Configuration Parameters** dialog. You can see an illustration of this pane in step 4 of "Generating Code" on page 2-10.

**2** After using any of these three options, the **Download Control Panel** dialog opens.
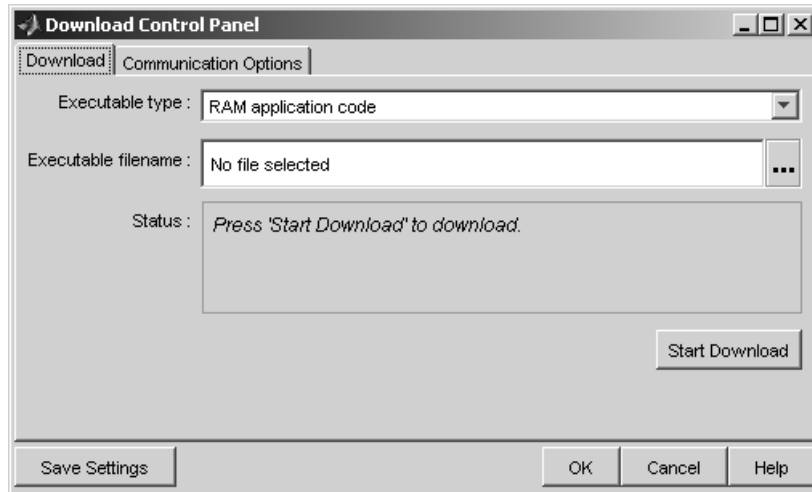
Note `RAM application code` is automatically selected in the **Executable type** menu. You can use exactly the same process to download application code to flash memory by changing this option to `Flash application code`. Note that you need to build a *model*_flash.s19 file in order to use this option, as described in "Downloading Application Code to Flash Memory via Serial or CAN" on page 2-24. For this exercise leave the `RAM` option selected.

**3** Enter the name of the file to be downloaded into the **Filename** field, in this case, `Target_LED_ram.s19`. Alternatively, you can use the browse button (right of the edit box) to navigate to the desired file. The **Download Control Panel** should now appear as shown in this figure.
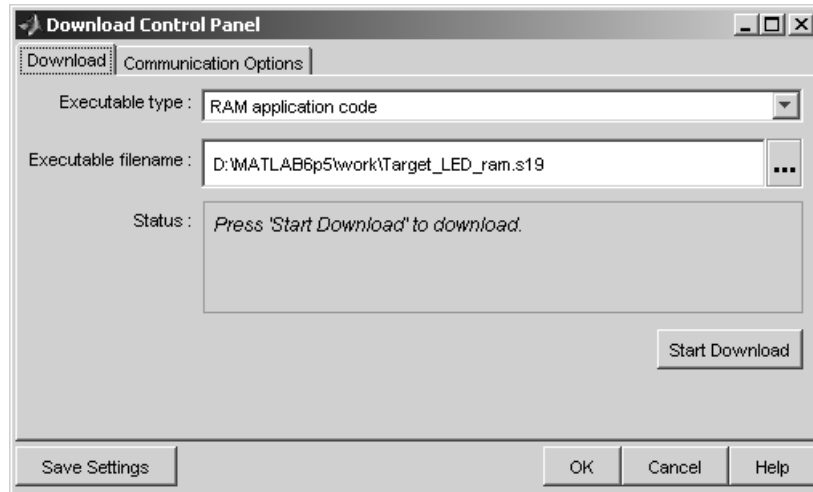
**4** Click on the **Communications Options** tab.

- If you are using serial, select Serial from the **Connection Type** drop-down menu. Select the appropriate host PC connection port from COM1 to COM8. You can save your preferences by clicking the **Save Preferences** button.

- If you are using CAN, select CAN from the **Connection Type** drop-down menu. Click **Configure** to select an appropriate card and port from the **CAN hardware** drop-down menu. You must create a MATLAB application channel to assign to a CAN channel. See "CAN Hardware and Drivers" on page A-21 for instructions. The default settings for the other parameters are appropriate for most cases. You can save your preferences by clicking the **Save Settings** button. The following figure shows the **Communications Options**.
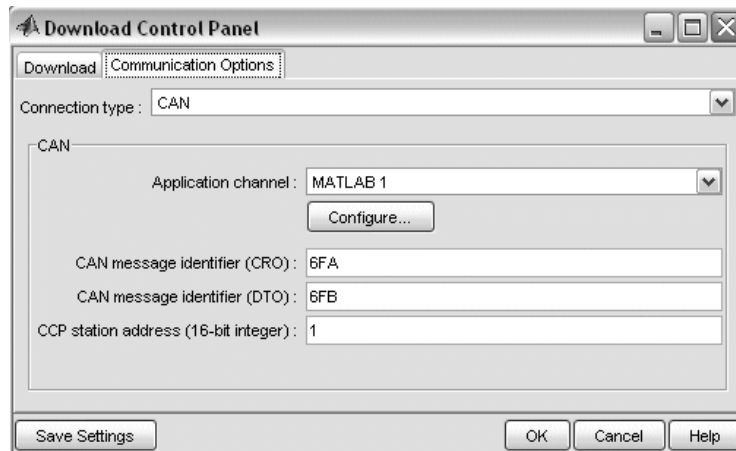
**5** Click the **Download** tab. Then click the **Start Download** button.

When you click **Start**, the **Download Control Panel**'s **Status** box changes to read Press reset or power-cycle your development board to start download.

**6** Press the Reset button on your PhyCORE-MPC555 board (or cycle the power). The **Download Control Panel** changes its **Status** box to inform you that the connection is OK.

Downloading commences, and the **Start** button caption changes to **Stop**.

**7** While downloading proceeds, progress messages are displayed in the **Download Control Panel**. After the download, the **Stop** button caption changes back to **Start**.

If the download does not succeed, reset your development board and return to step 5.

**8** Close the **Download Control Panel** dialog box.

**9** A few seconds after a successful download, the boot code transfers control to the application program. At this point, you should see two LEDs (red and green) blinking on the target board. This indicates that the program is operating correctly.

Note that you can monitor the progress of a CAN download using a program such as CANalyzer. Alternatively, you can use the btest32 utility supplied with the Vector Informatik driver software. You can invoke the btest32 utility from the PC command prompt. The following example runs btest32 with a bit rate of 500000 (500 kbaud):

```
btest32 500000
```

## Downloading the Application to RAM via BDM

You can choose to automatically download to the target over BDM on completion of the build process. Follow these steps to generate, download and execute the Target_LED_ram.elf file in RAM on the target system. Target_LED_ram.elf can contain both code and symbols for use with the debugger if you perform a debug build. You will not perform a debug build in this tutorial, so the file will contain code only.

If you want to download application code to MPC555 flash you can use serial or CAN. The download process is exactly the same as described in "Downloading the Application to RAM via Serial or CAN" on page 2-12, except you change the **Download** option from RAM to Flash. Note that you also need to generate a *model*_flash.s19 file to download to flash memory, as described in "Downloading Application Code to Flash Memory via Serial or CAN" on page 2-24. If you want to download the application to flash memory over BDM manually using your own tools, then the file you need to download is the S-record file *model*_flash.s19.

Do the following before you begin:

- Make sure that you have downloaded boot code to the flash memory of the MPC555. See "Download Boot Code to Flash Memory" on page 1-24.

- Connect the BDM port of your development board to parallel port LPT1 of your host PC (or the port specified for your toolchain if different, see "Setting Up Your Toolchain" on page A-3).

- Make sure that the jumpers on the phyCORE-MPC555 board are set as described in "Phytec MPC555 Jumper Settings" on page A-15.

To generate and download the Target_LED_ram.elf file to RAM over BDM,

**1** Select **Simulation > Configuration Parameters**.

The **Configuration Parameters** dialog appears.
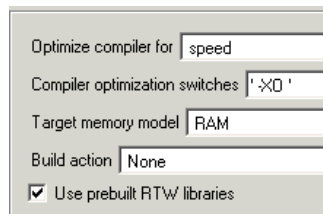
**2** Under **Real-Time Workshop** in the tree, click to select MPC5xx real-time options (1).

**3** Select Run_via_BDM or Debug_via_BDM from the **Build action** drop-down menu.

**4** Ensure the **Target memory model** selected is RAM (not FLASH).

Notice the default **Optimize compiler for** setting is speed. If you change this setting to debug, the generated elf file will contain both code and symbols for use with a symbolic debugger. See "Compiler Optimization Switches" on page 1-21 for more information on these settings. For this tutorial, leave this setting at the default, as shown.

| Optimize compiler for | speed | ▼ |
|---|---|---|
| Compiler optimization switches | '-X0 ' | |
| Target memory model | RAM | ▼ |
| Build action | Run_via_BDM | ▼ |
| ☑ Use prebuilt RTW libraries | | |
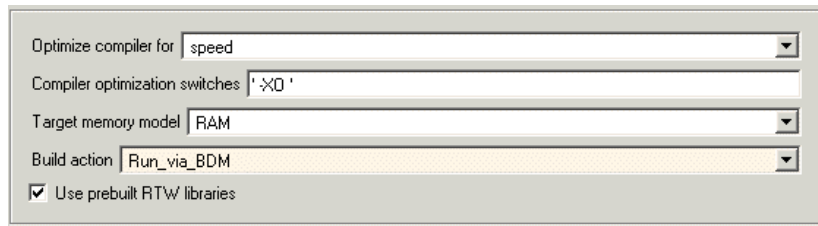
**5** Right click on the Target_LED subsystem and select **Real-Time Workshop –> Build Subsystem**.

You will see progress messages in the MATLAB Command Window as code is generated. Your debugger will be automatically started and will download the code to the target.

Also available is the **Start** menu option **Debug RAM-Based Application (Via BDM)**. Use this option to select a *.elf file and debug over BDM as described above. You can use this option to debug a model you have already built without having to go through the build process again.

# Downloading Boot and Application Code

| **In this section...** |
| --- |
| "RAM vs. Flash Memory" on page 2-19 |
| "Overview of Memory Organization and the Boot Process" on page 2-21 |
| "Downloading Application Code" on page 2-23 |
| "Stand-Alone Download Control Panel Utility" on page 2-27 |
| "Downloading Boot or Application Code via CAN Without Manual CPU Reset" on page 2-28 |
| "Rebuilding the Boot Code and Device Driver Libraries" on page 2-29 |
| "Running Applications with a Debugger" on page 2-31 |

## RAM vs. Flash Memory

Target for Freescale MPC5xx creates a file containing the application executable code that must be programmed onto the MPC555. It can also write a file including symbolic information suitable for use with a debugger. The files are written to your working directory.

The format of the code and symbol files is the same for both RAM and flash memory targets, suitable for downloading into RAM or on-chip flash memory. The naming convention for these files is

- *model*_flash.s19 or *model*_ram.s19 (for serial and CAN download)
- *model*_flash.elf or *model*_ram.elf (for BDM download, can contain debugging symbols).

You can download code to RAM or flash memory via serial or CAN download, or via the MPC555's BDM port.

There are advantages and disadvantages to each memory model.

Loading the application code into RAM is faster than loading it into flash memory. In addition, by using RAM you can avoid using up the programming cycles of the flash memory; this lengthens the usable lifetime of the flash

**2-19**

memory. Running the application from RAM is a good option for initial testing of the application.

---

**Note** The MPC5xx flash memory has a limited lifetime, which is shortened each time the flash memory is programmed. To extend product life, Freescale recommends using flash programming only when necessary.

---

To program applications into RAM, your target hardware must have additional RAM external to the MPC555 on-chip RAM. Target for Freescale MPC5xx does not support downloading of code to MPC5xx on-chip RAM, because the MPC555 has only 26K of on-chip RAM and the MPC565 has 36K.

For final deployment, or to load code onto a test board for use at a test site, you will generally want to program your code into the nonvolatile flash memory. 416K of flash memory is available for application code (992K on the MPC565). Code programmed into flash memory is persistent and restarts when the board is powered on.

To download code to flash memory, you must first load a binary boot code file into the flash memory. Target for Freescale MPC5xx provides the boot code file. You must load the boot code into flash memory in order to run application code. The boot code is always required even for RAM applications.

To understand the download process, it is first necessary to review the memory organization on the MPC555 and the operation of the boot code. This is described in the next section, "Overview of Memory Organization and the Boot Process" on page 2-21.

- If you just want to know how to download application code, you can jump ahead to the section "Downloading Application Code" on page 2-23.

- If you want to know how to download boot code, see the Getting Started section "Download Boot Code to Flash Memory" on page 1-24.

# Overview of Memory Organization and the Boot Process

### Purpose of Flash Memory Boot Code

When reading this section, you may want to refer to the internal memory map of the MPC555 in section 1.3 of the *MPC555 User's Guide*. You can find this document at the following URL:

```
http://www.freescale.com/files/microcontrollers/doc/user_guide/MPC555UM.pdf
```

To run generated code from the RAM or flash memory, you must load the first 32K flash sector with boot code. The primary purpose of the boot code is to load and start application code when the board is powered on or reset. The boot code also acts as a download agent that downloads generated code into RAM or flash memory via CAN or serial.

The boot code manages the exception handling for the MPC555. Applications don't directly handle exceptions but receive them from the boot code. If the boot code is not installed, then applications will not work correctly.

### Memory Organization

The MPC555 has a total of 448K of on-chip flash memory (1024K on the MPC565). This memory is organized into banks of 32K each. The first bank is always used to store the boot code and the remaining 416K is available for application code (992K on the MPC565). When using Target for Freescale MPC5xx, the on-chip flash memory is located at absolute address 0x0000 in the MPC555 address space.

| | |
|---|---|
| Boot Code | 0x0000 |
| | 0x8000 |
| Application Code | |
| | 0x70000 |
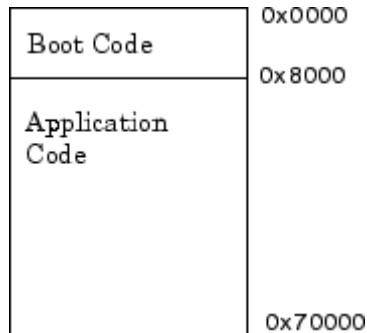
**Organization of Flash Memory**

To run a stand-alone application on the MPC555, it is first necessary to program the boot code into the first bank of flash memory.

## The Boot Process

The boot code is executed following power on or reset (unless a probe is connected to the BDM port). Normally, the boot code performs basic hardware initialization and then branches to the application code. Once the application code is running, there is no way to return to the boot code except by performing a reset.

One of the important functions of the boot code is to serve as agent that allows program code to be downloaded over CAN or serial. There are two methods of initiating a program download over CAN or serial:

- The default method for initiating a program download is to send a special serial or CAN message during a short window of time while the boot code is executing. In the supplied boot code, this window is set to 40ms. If this special message is received during the window while the boot code is executing, a program download sequence commences and a new application can be programmed into RAM or flash memory. See "Downloading Application Code to Flash Memory via Serial or CAN" on page 2-24 for details.

- Alternatively, it is possible to commence a program download over CAN while application code is running on the target. To initiate a download over CAN, you must include a special block in your Simulink model. This block is

the CAN Calibration Protocol block. See "Downloading Boot or Application Code via CAN Without Manual CPU Reset" on page 2-28 for details.

The bootcode download process erases the non-volatile flash memory (including the shadow area) before writing the new bootcode, and the previous configuration word is removed. The bootcode download process does not write a replacement configuration word to the shadow flash. Typically, users of Target for Freescale MPC5xx do not use a Hard Reset Configuration Word that is stored in non-volatile memory (the shadow flash). Instead, the development board is generally assumed to source the configuration word from the data bus.

If you want to use a custom configuration word, you must manually program the shadow flash to an appropriate value for the system. This would only need to be done along with the irregular updates to the bootcode.

## Downloading Application Code

The following sections describe how to download generated image files and run generated code on the target hardware. They also describe how to download to RAM and to flash memory, via either serial, CAN, or the BDM port.

### Downloading the Application Code to RAM

To download application code to RAM, you must generate a code file in Motorola S-Record format, which is suitable for downloading and execution in RAM. To do this, select the RAM option from the **Target memory model** menu in the MPC5xx real-time options (1) category of the **Configuration Parameters** dialog. The build process creates two files in the working directory:

- Files created:
  - *model*_ram.s19: For serial or CAN download. Code only, without symbols, suitable for execution on the target system.
  - *model*_ram.elf: For BDM download. Can also contain symbols if you perform a debug build, suitable for use with a symbolic debugger such as Wind River SingleStep.

- You can download to RAM via serial or CAN, using the **Download Control Panel** utility (with Vector-Informatik CAN hardware if applicable), as described in "Downloading the Application to RAM via Serial or CAN" on page 2-12.

- You can also download to RAM via BDM, as described in "Downloading the Application to RAM via BDM" on page 2-17.

## Downloading the Application Code to Flash Memory

To download application code to flash memory, you must generate a code file which is suitable for downloading and execution in flash memory. To do this, select the FLASH option from the **Target memory model** menu in the MPC5xx real-time options (1) category of the **Configuration Parameters** dialog. The build process creates the file *model*_flash.s19 which contains an image of the executable code, in the working directory.

You can download the file to flash memory via serial or CAN, using the **Download Control Panel** utility (with Vector-Informatik hardware if using CAN), as described in the following section. Note you can also use the **Start** menu option to use a BDM (and serial or CAN) to download application code to flash memory. If you want to download the application to flash memory over BDM manually using your own tools, then the file you need to download is the S-Record file *model*_flash.s19.

## Downloading Application Code to Flash Memory via Serial or CAN

You can use the **Download Control Panel** to download generated application code to the MPC555 flash memory. Note that except for changing the **Download** option from RAM to Flash, the process is the same as downloading to RAM.

Do the following before you begin:

- If you are using serial, make sure you have connected the serial port on your PC to serial port 1 (RS232-1) on the target hardware.

- If you are using CAN, make sure that your Vector-Informatik CAN card and drivers are installed, and are configured properly. See "CAN Hardware

and Drivers" on page A-21. Make sure that the desired CAN port on the PC card is connected to the CAN A port on the target hardware.

- Make sure that you have set up your toolchain and downloaded boot code to the flash memory of the MPC555, as described in "Setting Up and Verifying Your Installation" on page 1-16.

- Make sure that nothing is connected to the BDM port of your development board.

- Make sure that the jumpers on the phyCORE-MPC555 board are set as described in "Phytec MPC555 Jumper Settings" on page A-15.

To download the generated *model*_flash.s19 file to flash:

**1** Open the **Download Control Panel** in one of the following ways:

- Use the MATLAB **Start** menu. Select **Start > Links and Targets > Target for Freescale MPC5xx > Download RAM / FLASH Based Application (via CAN / Serial)**.

- Type embedded_target_download at the MATLAB command prompt.

- You can also open the **Download Control Panel** automatically at the end of the build process. Before you start the build process, you can select **Launch Download Control Panel** from the **Build action** options on the MPC5xx real-time options (1) tab of the Model Explorer. You can see an illustration of this tab in step 4 of "Generating Code" on page 2-10.

After using any of these three options, the **Download Control Panel** window opens.

**2** Select Flash application code from the **Executable type** menu.

**3** Enter the name of the file to be downloaded into the **Executable filename** field. Alternatively, you can use the browse button to navigate to the desired file. Remember the *model*_flash.s19 files are for serial or CAN download to flash. The **Download Control Panel** should now appear as shown in this figure.
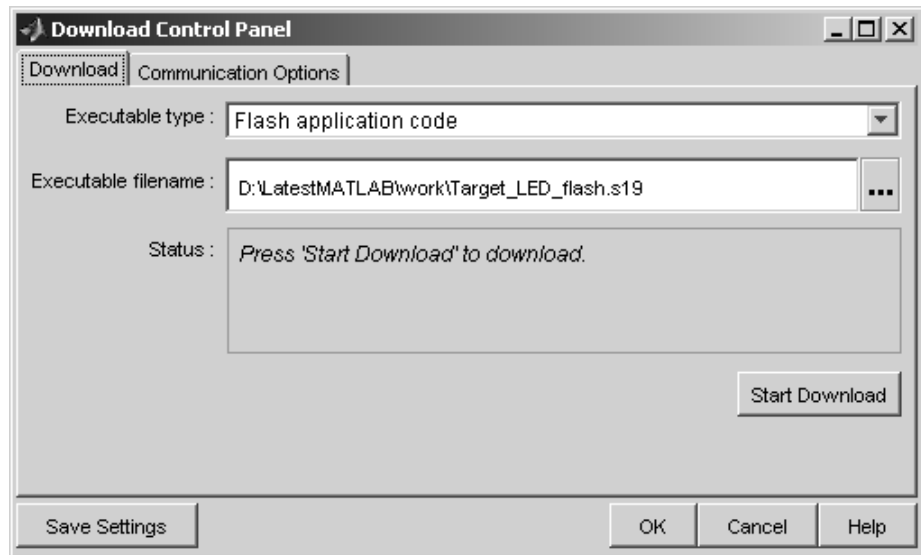
4 Click on the **Communications Options** tab. If you have not saved your preferences already, select Serial or CAN from the **Connection Type** drop-down menu. If necessary, select an appropriate card/port. The default settings for the other parameters are appropriate for the default download process. You can save your preferences by clicking the **Save Preferences** button. The **Communications Options** configured for a Vector-Informatik CAN-AC2-PCI card, channel 1, are shown in the section "Downloading the Application to RAM via Serial or CAN" on page 2-12.

5 The next step is to download code. Click the **Download** tab, and then click the **Start** button.

- If there is an application currently running on the target that contains a CAN Calibration Protocol (CCP) kernel, the download proceeds automatically. For more details see "Downloading Boot or Application Code via CAN Without Manual CPU Reset" on page 2-28.

- If the CCP condition is not met, you must immediately press the reset button on your PhyCORE-MPC555 board after clicking the **Start** button. You will see a message prompt in the **Status** box: Press reset or power-cycle your development board to start download.

When you press the Reset button on your PhyCORE-MPC555 board (or cycle the power), the **Download Control Panel** changes its **Status** box to read CCP Connection OK. Please wait till completion or press Stop to terminate the download.

Downloading commences, and the **Start** button caption changes to **Stop**. While downloading proceeds, progress messages are displayed in the **Download Control Panel**. A successful download ends with an information dialog and the **Stop** button caption changes back to **Start**.

**6** If the download does not succeed, reset the board and return to step 5.

You can monitor the progress of the flash download over CAN by using a program such as CANalyzer. Alternatively, you can use the btest32 utility supplied with the Vector Informatik driver software. You can invoke the btest32 utility from the PC command prompt. The following example runs btest32 with a bit rate of 500000 (500kbaud):

```
btest32 500000
```

**7** Close the **Download Control Panel** window.

Once the download process is complete, the application starts running immediately on the target hardware.

## Stand-Alone Download Control Panel Utility

Note that you can use the **Download Control Panel** utility separately as a stand-alone application from MATLAB. For instructions, run this command in MATLAB:

```
system(['"' matlabroot '\toolbox\rtw\targets\common\tgtcommon\embedded_target_download.bat"-help']);
```

Alternatively, run the batch file found here:

```
matlabroot\toolbox\rtw\targets\common\tgtcommon\embedded_target_download.bat -help
```

where *matlabroot* is the full path to your MATLAB installation directory.

**2-27**

## Downloading Boot or Application Code via CAN Without Manual CPU Reset

The default method for download over CAN requires that the target processor be manually reset in order for the download process to begin. This requirement may be problematic if the target hardware is not physically accessible or if it cannot be individually reset or powered down/up.

It is possible to remove this requirement for manual reset if a suitably prepared application is already running on the target. To do this, include a CAN Calibration Protocol block within the model (see CAN Calibration Protocol (MPC555)).

When the currently running application includes the CAN Calibration Protocol block, the download process begins when you click on the **Start** button of the **Download Control Panel**; it is not necessary to manually reset the target hardware to initiate the download. A reset of the processor is triggered by a CCP Program Prepare message. After the Program Prepare message is received at the target, there will be a short delay until the processor resets and continues the download process by transmitting a response to the Program Prepare message.

The length of the delay will be the watchdog timeout period of the application. By default, for a 20MHz application, this will be approximately 7 seconds; for a 40MHz application, this will be approximately 3 seconds.

It is possible to explicitly set the timeout period of the watchdog timer, by using the Watchdog block in the MPC555 device driver library. See Watchdog .

The **Download Control Panel** is configured to allow a maximum delay of 10 seconds between sending the Program Prepare message and receiving a response from the target. If this delay is exceeded, an error will be reported by the download tool.

When using the CAN Calibration Protocol block, you must specify

- CAN message identifier for Command Receive Objects
- CAN message identifier for Data Transmit Objects
- CAN Calibration Protocol Station Address

Note that the values specified are permitted to differ from the default values for these parameters that are programmed in the boot code. When performing the download procedure using the **Download Control Panel**, you must ensure that the parameters specified on the **Communications Options** tab match those specified in the currently running application.

For an example of how to use the CAN Calibration Protocol block for signal monitoring, parameter tuning and automatic download, see the demo model `mpc555rt_ccp`.

## Rebuilding the Boot Code and Device Driver Libraries

You must rebuild the libraries to enable execution profiling for device driver interrupt service routines. See "Enabling Execution Profiling for Device Driver Interrupt Service Routines" on page 2-56 for instructions in that case.

You cannot change the default boot code parameter values except by modifying and recompiling the boot code. If it is absolutely necessary to do this, you can recompile the boot code as follows:

**1** Select **Start > Links and Targets > Target for Freescale MPC5xx > Rebuild MPC555 Driver Library**.

The **Build Driver Libraries** dialog opens.
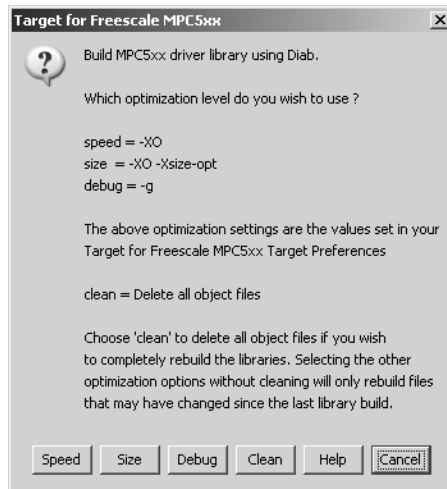
**2** Select the compiler optimization setting you want to use for the build (from `speed`, `size`, `debug`, or `clean`).

- See "Compiler Optimization Switches" on page 1-21 for more information on the `speed`, `size` and `debug` settings, which are compiler-specific. You can edit these settings in the **Target Preferences** dialog.

- The `clean` option deletes all object files. Note that to ensure a rebuild of all files you should run a `clean` build followed by a build using your required optimization setting. Otherwise only files which have changed since last library build will be rebuilt.

Target for Freescale MPC5xx automatically recompiles the code, using your settings in target preferences.

---

**Note** You should not make changes to the boot code without fully understanding the effect of your changes. Note also that the boot code may be changed without notice in future releases of this product.

---

If a required prebuilt library is not found during the build process, then you see a dialog box with instructions to rebuild the missing library. For example, a prebuilt copy of the Signal Processing Library is not installed with the product.

It is preferable to rebuild via the **Start** menu rather than using the commands suggested in the dialog box, because any rebuild done via the dialog is dependent on the options selected in the Real-time Workshop > Interface > Software Environment > Support options, and any library created is based on these settings. You then need to rebuild your model to complete the build process.

### Boot Code Parameters for CAN Download

The boot code parameters for download over CAN determine

- CAN bit rate

- CAN message identifier for Command Receive Objects (CRO)

- CAN message identifier for Data Transmit Objects (DTO)

- CAN Calibration Protocol Station Address
- The duration of the window during which the boot code listens for a download command message

Default Boot Code Parameters on page 2-31 shows the default values for these parameters. These defaults should be suitable for most applications.

**Default Boot Code Parameters**

| Parameter | Default Value |
|---|---|
| CAN bit rate | 500000 |
| CCP station address | 1 |
| CAN message identifier (CRO) | 6FA |
| CAN message identifier (DTO) | 6FB |
| Duration of listening window | 40 ms |

## Running Applications with a Debugger

It is possible to run an application with a debugger. To have full debugging capabilities it is necessary that both the application and device driver libraries are built with debug switches enabled.

To run an application with a debugger it is necessary you must go through the following steps.

**1** In the model **Configuration Parameters** dialog, under the **MPC5xx** options section ensure that **Optimize compiler for** is set to debug.

**2** In the Target Preferences ensure that the debug compiler switches are set appropriately for your configuration; see "Setting Target Preferences" on page 1-17 for examples.

**3** By default the device driver libraries are compiled without debug flags; if you need to be able to debug device driver code as well as model code you must re-build the device driver libraries using the debug option. See "Boot Code Parameters for CAN Download" on page 2-30.

Once you have performed the above steps and built your model, you can run it with the source level debugger by selecting **Links and Targets > Target for Freescale MPC5xx > Debug RAM-Based Application (via BDM)** from the MATLAB **Start** menu.

# Parameter Tuning and Signal Logging

| In this section... |
| --- |
| "Methods for Parameter Tuning and Signal Logging" on page 2-33 |
| "Using External Mode" on page 2-33 |
| "Using a Third Party Calibration Tool" on page 2-42 |
| "Data Acquisition (DAQ) List Configuration" on page 2-45 |

## Methods for Parameter Tuning and Signal Logging

Target for Freescale MPC5xx supports parameter tuning and signal logging either using Simulink external mode or with a third party calibration tool. In both cases the model must include a special block, the CAN Calibration Protocol block (see CAN Calibration Protocol (MPC555)).

## Using External Mode

Simulink external mode enables you to log signals and tune parameters without requiring a calibration tool. This section describes the steps for converting a model to use external mode.

External mode is supported using the CAN Calibration Protocol block and ASAP2 interface. The CAN Calibration Protocol block is used to communicate with the target, downloading parameter updates and uploading signal information. The ASAP2 interface is used to get information about where in the target memory a parameter or signal lives.

---

**Note** You must configure the host-side CAN application channel. See "Configuring the Host Vector CAN Application Channel " on page 2-35.

---

To prepare your model for external mode, follow these steps:

**1** Add a CCP driver block.

**2** Add a Switch External Mode Configuration Block (for ease of use; you can also make changes manually).

**3** Identify signals you want to tune, and associate them with `Simulink.Parameter` objects with `ExportedGlobal` storage class. It is important to set the data type and value of the `Simulink.Parameter` object. See "Using Supported Objects and Data Types" on page 2-35.

**4** Identify signals you want to log, and associate them with `canlib.Signal` objects. It is important to set the data type of the `canlib.Signal`. See "Using Supported Objects and Data Types" on page 2-35.

   For information about visualizing logged signal data, see "Viewing and Storing Signal Data" on page 2-37.

**5** Load the the `Simulink.Parameter` and `canlib.Signal` data objects into the base workspace.

**6** Configure the model for building by double-clicking the Switch External Mode Configuration block. In the dialog box, select **Building an executable**, and click **OK**.

**7** Build the model, and download the executable to the target

**8** After downloading the executable to the target, you can switch the model to external mode by double-clicking the Switch External Mode Configuration Block. In the dialog box that appears, select **External Mode**, and click **OK**.

**9** You can now connect to the target using external mode by clicking the **Connect** button.

**10** If you have set up tunable parameters, you can now tune them. See "Tuning Parameters" on page 2-36.

If you do not want to use the Switch External Mode Configuration block, you can configure for building and then external mode manually. For instructions, see "Manual Configuration For External Mode" on page 2-40.

See the following topics for more information:

• "Configuring the Host Vector CAN Application Channel " on page 2-35

## Configuring the Host Vector CAN Application Channel

External mode expects that the host-side CAN connection is using the `'MATLAB 1'` application channel. To configure the application channel used by the Vector CAN drivers, enter the following at the MATLAB command line:

```
TargetsComms_VectorApplicationChannel.configureApplicationChannels
```

The Vector CAN Configuration tool appears. Use this tool to configure your host-side CAN channel settings.

If you try to connect using an application channel other than `'MATLAB 1'`, then you see the following warning in the command window:

```
Warning:
It was not possible to connect to the target using CCP.
An error occurred when issuing the CONNECT command.
```

## Using Supported Objects and Data Types

Supported objects:

- `Simulink.Parameter` for parameter tuning
- `canlib.Signal` for signal logging

Supported data types:

- uint8, int8
- uint16, int16
- uint32, int32
- single

You need to define data objects for the signals and parameters of interest for ASAP 2 file generation. For ease of use, create an m-file to define the data objects, so that you only have to set up the objects once.

To set up tuneable parameters and signal logging:

**1** Associate the parameters to be tuned with Simulink.Parameter objects with ExportedGlobal storage class. It is important to set the data type and value of the Simulink.Parameter object. See the following m-code for an example of how to create such a Simulink.Parameter object for tuning:

```
stepSize = Simulink.Parameter;
stepSize.DataType = 'uint8';
stepSize.RTWInfo.StorageClass = 'ExportedGlobal';
stepSize.Value = 1;
```

**2** Associate the signals to be logged with canlib.Signal objects. It is important to set the data type of the canlib.Signal. The following m-code example shows how to declare such a canlib.Signal object for logging:

```
counter = canlib.Signal;
counter.DataType = 'uint8';
```

**3** Associate the data objects you have defined in the m-file with parameters or signals in the model. For the previous m-code examples, you could set the **Constant value** in a Source block to stepSize, and set a **Signal name** to counter in the Signal Properties dialog box. Remember that stepSize and counter are data objects defined in the m-code.

### Tuning Parameters

To tune a parameter, follow these steps:

**1** Set *dataobject*.value in the workspace while the model is running in external mode. For example, to tune the parameter stepSize (that is, to change its value) from 1 to 2, enter the following at the command line:

```
stepSize.value = 2
```

You see output similar to the following:

```
stepSize =

Simulink.Parameter (handle)
        RTWInfo: [1x1 Simulink.ParamRTWInfo]
    Description: ''
       DataType: 'uint8'
            Min: -Inf
            Max: Inf
       DocUnits: ''
          Value: 2
     Complexity: 'real'
     Dimensions: [1 1]
```

**2** Return to your model, and update the model (press **Ctrl+D**) to apply the changed parameter.

### Viewing and Storing Signal Data

To view the logged signals attach a supported scope type to the signal (see "Limitations" on page 2-41 for supported scope types).

Select which signals you want to log by using the External Signal & Triggering dialog box. Access the External Mode Control Panel from the Tools menu, and click the **Signal & Triggering** button. By default, all displays appear as selected to be logged, as shown in the following example. Edit these settings if you do not want to log all displays. Individual displays can be selected manually.
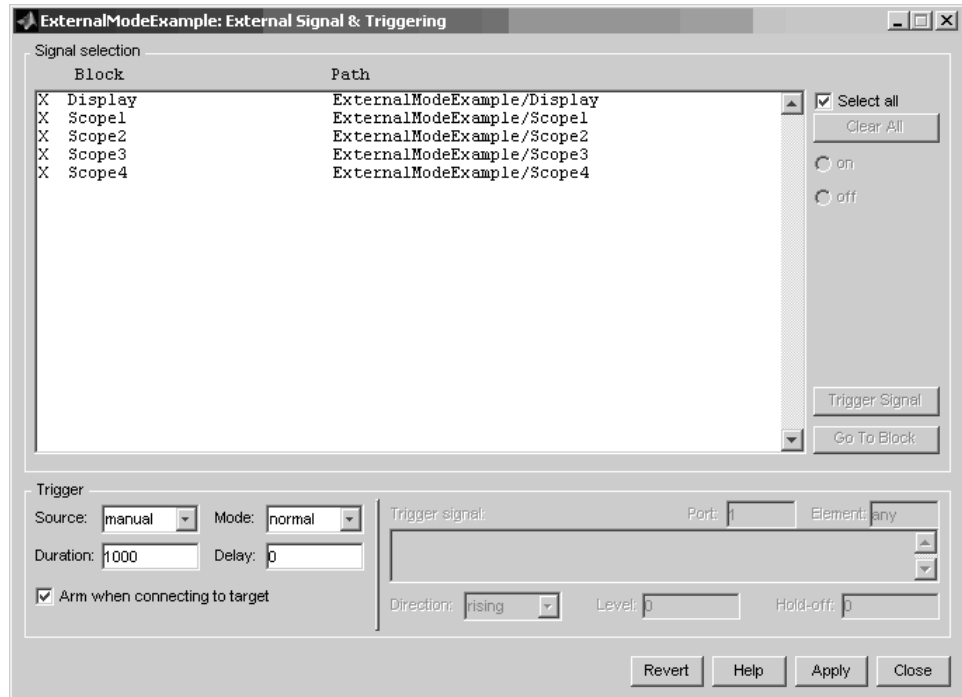
**Storing signal data for further analysis.** It is possible to store the logged data for further analysis in MATLAB.

1 To use the Data Archiving feature of external mode, click **Data Archiving** in the External Mode Control Panel. The External Data Archiving dialog box appears.
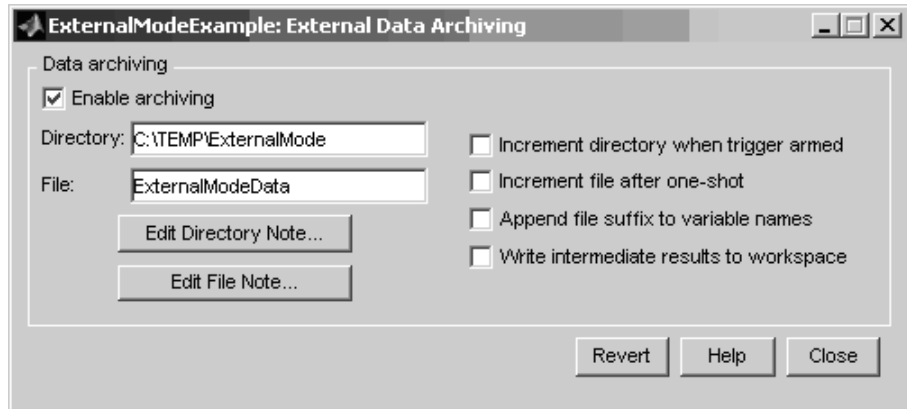
**a** Select the check box **Enable archiving**

**b** Edit the **Directory** and **Filename** and any other desired settings.

**c** Close the dialog box.

**2** Open the Scope parameters, and select the check box **Save data to workspace**.
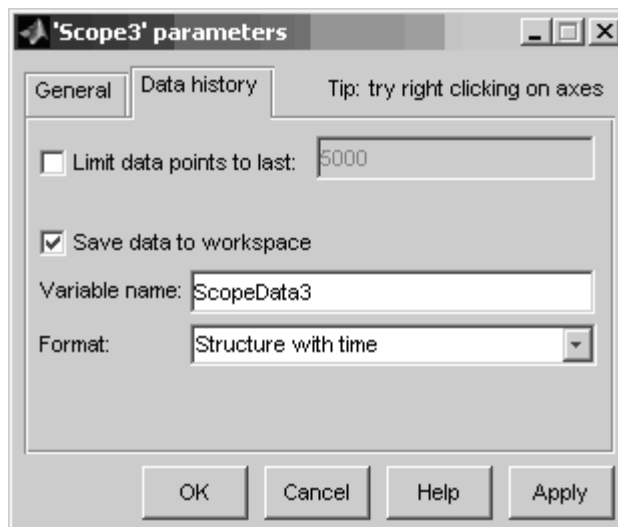
**3** You may want to edit the **Variable name** in the edit box. The data that is displayed on the scope at the end of the external mode session is available in the workspace with this variable name.

The data that was previously displayed in the scope is stored in `.mat` files as previously setup using Data Archiving.

For example, at the end of an external mode session, the following variable and files could be available in the workspace and current directory:

- A variable `ScopeData5` with the data currently displayed on the scope:

```
ScopeData5

ScopeData5 =

        time: [56x1 double]
     signals: [1x1 struct]
   blockName: 'mpc555rt_ccp/Scope1'
```

- In the current directory, .mat files for the three previous **Durations** of scope data:

```
ExternalMode_0.mat
ExternalMode_2.mat
ExternalMode_1.mat
```

## Manual Configuration For External Mode

As an alternative to using the Switch External Mode Configuration block, you can configure models manually for build and execution with external mode.

To configure a model to be built for external mode:

**1** Select **Inline parameters** (under Optimization in the Configuration Parameters dialog box). The **Inline parameters** option is required for ASAP2 generation.

**2** Select **Normal** simulation mode (in either the Simulation menu, or the drop-down list in the toolbar).

**3** Select `ASAP2` as the **Interface** (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).

After you build the model, you can configure it for external mode execution:

**1** Make sure **Inline parameters** are selected (under Optimization in the Configuration Parameters dialog box). The **Inline parameters** option is required for external mode.

**2** Select **External** simulation mode (in either the Simulation menu, or the drop-down list in the toolbar).

**3** Select `External mode` as the **Interface** (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).

### Limitations

Multiple signal sinks (e.g. scopes) are not supported.

Only the following kinds of scopes are supported with External Mode Logging:

- Simulink Scope block
- Simulink Display block
- Viewer type: scope — To use this option, right-click a signal in the model, and select **Create & Connect Viewer > Simulink > Scope**. The other scope types listed there are not supported (e.g., floating scope).

  Before connecting to external mode, you also need to right-click the signal, and select **Signal Properties**. In the dialog box, select the **Test point** check box, and click **OK**.

GRT is supported but only for parameter tuning.

It is not possible to log signals with very fast sample times (e.g., 0.0001) without losing data.

Subsystem builds are not supported for external mode, only top-level builds are supported.

Logging and tuning of nonscalars is not supported. It is possible to log nonscalar signals by breaking the signal down into its scalar components. For an example of how to do this signal deconstruction, see the CCP demo models, which use a Demux and Signal Conversion block with contiguous copy.

Logging and tuning of complex numbers is not supported. It is possible to work with complex numbers by breaking the complex number down into its real and imaginary components. This breakdown can be performed using the following blocks in the Simulink Math Operations library: Complex to Real-Imag, Real-Imag to Complex, Magnitude-Angle to Complex, Complex to Magnitude-Angle.

## Using a Third Party Calibration Tool

Target for Freescale MPC5xx allows an ASAP2 data definition file to be generated during the code generation process. This file can be used by a third party tool to access data from the real-time application while it is executing.

ASAP2 is a data definition standard proposed by the Association for Standardization of Automation and Measuring Systems (ASAM). ASAP2 is a standard description used for data measurement, calibration, and diagnostic systems. The Target for Freescale MPC5xx real-time target lets you export an ASAP2 file containing information about your model during the code generation process.

Before you begin generating ASAP2 files with the Target for Freescale MPC5xx real-time target, you should read the "Generating ASAP2 Files" section of the Real-Time Workshop Embedded Coder documentation. That section describes how to define the signal and parameter information required by the ASAP2 file generation process.

The process of generating an ASAP2 file from your model with the Target for Freescale MPC5xx real-time target is similar to that described in the Real-Time Workshop Embedded Coder documentation.

### How the Process Works

The `mpc555rt_ccp` demo provides an example of the Target for Freescale MPC5xx ASAP2 file generation feature.

Target for Freescale MPC5xx generates an initial ASAP2 file during the code generation process. At this point, the addresses of signals and parameters on the target system are unavailable, since the code has not been compiled and linked. The initial ASAP2 file contains placeholders for the unresolved addresses.

To supply the required memory addresses, the generated code must be compiled and the compiler must generate a MAP file.

After the build process, if the Target for Freescale MPC5xx real-time target detects the presence of the ASAP2 file and a MAP file in the required format, it performs a post-processing phase. During this phase, the MAP file is used to propagate the required address information back into the ASAP2 file.

MAP file formats differ between compilers, so the post processing phase is compiler-specific. Target for Freescale MPC5xx provides the post-processing mechanism for both supported toolchains (Wind River and CodeWarrior).

To use the Target for Freescale MPC5xx ASAP2 file generation feature, you simply need to select the **ASAP2** file option in Real-Time Workshop, as described in the following section "ASAP2 File Generation Procedure" on page 2-43. If it is appropriate to back propagate addresses from the MAP file into the ASAP2 file, then this will also be done automatically. No other steps are necessary to ensure that the generated MAP and ASAP2 files are automatically post processed.

The names of the ASAP2 file and the MAP file derive from the source model. The MAP file is generated in the same directory as the source model. The ASAP2 file is written to the build directory.

### ASAP2 File Generation Procedure

**1** Create the desired model. Use appropriate parameter names and signal labels to refer to ASAP2 CHARACTERISTICS and MEASUREMENTS respectively.

**2** Define the corresponding Simulink.Parameter and Simulink.Signal objects in the MATLAB workspace.

**3** Configure the data objects to generate unstructured global storage declarations in the generated code by assigning one of the following storage classes to the `RTWInfo.StorageClass` property for each object:

- `ExportedGlobal`

- `ImportedExtern`

- `ImportedExternPointer`

`ExportedGlobal` is the default storage class.

**4** Configure the other data object properties for each object. See "Defining ASAP2 Information" in the Real-Time Workshop documentation.

**5** In your model window, select the menu item **Simulation > Configuration Parameters**.

The **Configuration Parameters** dialog box appears.

**6** Select **Optimization** in the tree.

**7** Select the **Inline parameters** option.

Note that you should *not* configure the parameters associated with your data objects in the **Model Parameter Configuration** dialog box (reached via the **Configure** button). If a parameter that resolves to a Simulink data object is configured using the **Model Parameter Configuration** dialog box, the dialog box configuration is ignored. You can, however, use the **Model Parameter Configuration** dialog to configure other parameters in your model.

**8** Under **Real-Time Workshop**, select **Interface** in the tree.

**9** Select the **ASAP2** option from the **Interface** drop-down menu in the `Data exchange` frame, as shown in the following figure.
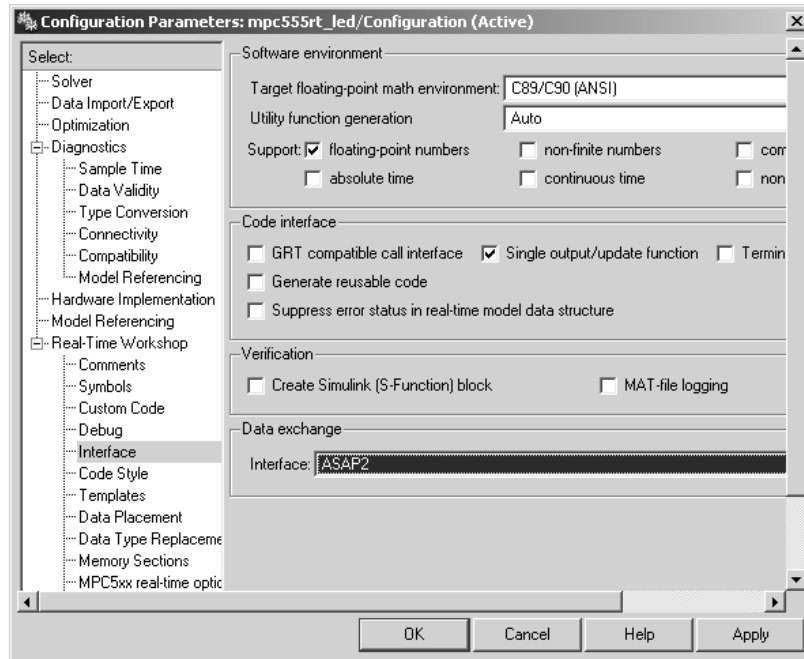
**10** Click **Apply**.

**11** Select **Real-Time Workshop** in the tree, then click **Build**.

The ASAP2 file is generated as part of the build process.

## Data Acquisition (DAQ) List Configuration

Target for Freescale MPC5xx supports the Data Acquisition (DAQ) List feature of the CAN Calibration Protocol (CCP). DAQ lists allow efficient synchronous signal monitoring. The CCP block provided with Target for Freescale MPC5xx supports DAQ lists (see CAN Calibration Protocol (MPC555) for details).

Simulink.Signal objects are used for monitoring a signal in the CCP polling mode of operation. To monitor a signal in a DAQ list, however, you must configure the signal somewhat differently. The differences are as follows:

- Instead of defining a `Simulink.Signal` in the MATLAB workspace (and associated signal in the Simulink model), define a `canlib.Signal` object instead.

- There is no need to set the `RTWInfo.StorageClass` property of the `canlib.Signal` object. By default, the storage class is set to `Custom`.

- You should enter data in the other fields of the `canlib.Signal` object in the same way you would do for a `Simulink.Signal` object.

**Note** In order to use the `canlib.Signal` objects, the model must contain a CAN Calibration Protocol block. See CAN Calibration Protocol (MPC555).

During code generation, Target for Freescale MPC5xx automatically determines how to configure the DAQ lists in the generated code. For each distinct sample rate (of the set of `canlib.Signal` objects assigned by the user) one DAQ list in the model is created. The CCP DAQ List Object Descriptor Tables (ODTs) are shared equally between the created DAQ lists.

The sample rates of the `canlib.Signal` objects are mapped to CCP event channels in an extra file, `DAQ_LIST_EVENT_MAPPINGS`, that is generated in the build directory. This shows how to assign event channels to MEASUREMENT signals in a calibration package.

The event channels periodically transmit events that are used to trigger the sending of DAQ data to the host. By assigning event channels as defined in `DAQ_LIST_EVENT_MAPPINGS`, consistent and efficient transmission of DAQ data is achieved.

It is the responsibility of the calibration tool (see "Compatibility with Calibration Packages" on page 5-9) to assign an event channel and data to the available DAQ lists using CCP commands, and to interpret the synchronous response.

It is the responsibility of the user to make sure the calibration tool is set up correctly and that the event channels assigned to MEASUREMENT signals correspond to those defined in the file `DAQ_LIST_EVENT_MAPPINGS`.

# HTML Code Profile (RAM/ROM) Report

Target for Freescale MPC5xx supports an extended version of the Real-Time Workshop Embedded Coder HTML code generation report.

For instructions, see "HTML Code Analysis (RAM/ROM) Report" on page 3-27. You can generate reports for the real-time target, processor-in-the-loop (PIL) target and algorithm export (AE) target.

# Execution Profiling

| **In this section...** |
| --- |
| "Overview of Execution Profiling" on page 2-48 |
| "The Profiling Command" on page 2-49 |
| "Execution Profiling Definitions" on page 2-51 |
| "Real-Time Workshop Options for Execution Profiling" on page 2-52 |
| "Interpreting the Execution Profiling Graphic" on page 2-54 |
| "Enabling Execution Profiling for Device Driver Interrupt Service Routines" on page 2-56 |

## Overview of Execution Profiling

Target for Freescale MPC5xx provides a set of utilities for recording, uploading and analyzing execution profile data for timer-based tasks and asynchronous Interrupt Service Routines (ISRs). With these utilities, you can

- Generate a graphical display that shows when timer-based tasks and interrupt service routines are activated, preempted, resumed and completed.

- Generate a report with information on

  - Maximum number of overruns for each timer-based task since model execution began

  - Maximum turnaround time for each timer-based task since model execution began

  - Analysis of profiling data for timer-based tasks and asynchronous interrupts over a period of time

You can use the demo model `mpc555rt_multitasking` to see an example. This demo model illustrates both execution profiling and the preemptive multitasking scheduler with configurable overrun handling. For instructions, click the link MPC555 Multitasking Demo.

To perform execution-profiling analysis on a model, you must perform the following steps:

**1** Depending on whether you are using serial or CAN, place a copy of the appropriate execution profiling block in your model (MPC555 Execution Profiling via SCI1 or MPC555 Execution Profiling via CAN A).

**2** Connect a serial or CAN cable between the target processor and your host PC.

**3** Check the box to enable Execution profiling in Real-Time Workshop options. See "Real-Time Workshop Options for Execution Profiling" on page 2-52.

**4** Build, download and run the model.

**5** Initiate execution profiling by running the command `profile_mpc555`. See below for more information on the profiling command.

Two forms of execution profiling are provided:

**1** The worst-case values for task turnaround times and number of task overruns since model execution began are updated whenever a previous worst-case value is exceeded.

**2** A snapshot of task and ISR activity may be recorded over a period of time; the length of this period depends on how much memory is available to log the data.

**Note** You need additional steps if device drivers use interrupt service routines (may include CAN, TPU, and QSPI ). See "Enabling Execution Profiling for Device Driver Interrupt Service Routines" on page 2-56. If this is not done, then no profiling information will be recorded.

## The Profiling Command

Use the profiling command as follows:

```
profile_mpc555(connection)
```

Specify your `connection` as `'can'` or `'serial'`, to collect data via a CAN or serial connection between the target and the host computer. Make sure the model includes the appropriate MPC5xx execution profiling block (CAN or SCI1), to provide an interface between the target-side profiling engine and the host-side computer from which this command is run.

`PROFDATA = profile_mpc555(connection)` collects and displays execution profiling data from a MPC5xx target microcontroller that is running a suitably configured application generated by Target for Freescale MPC5xx. `PROFDATA` contains the execution profiling data in the format documented by `exprofile_unpack`.

The data collected is unpacked then displayed in a summary HTML report and as a MATLAB graphic.

To use the serial connection, the MPC5xx board must be connected via a serial cable to one of the host computer's serial ports. This function defaults to port SCI1 on the MPC5xx and port COM1 on the host computer. If the `'BitRate'` argument is not provided, the default of 57600 baud is used.

```
PROFDATA = PROFILE_mpc555('serial','SerialPort',serialport)
```

sets the serial port to the specified `serialport`, which should be one of COM1, COM2, etc.

Optionally, you can specify the bit rate as follows:

```
PROFDATA = PROFILE_mpc555('serial', 'BitRate', bitrate)
```

This specification sets the `bitrate` for serial connection to the target. `bitrate` must be the same as the bit rate specified for the application that is running on the target.

Alternatively, you can set the bitrate for the serial connection to the target automatically as follows:

```
profdata = profile_mpc555('serial', 'ModelName', modelname)
```

This specification automatically sets the bit rate by analyzing *modelname* and extracting the correct serial connection bit rate setting from the model.

*modelname* should be set to the name of a model which is currently open and running on the target.

To use the CAN connection, you must have suitable CAN hardware installed. If no Application Channel is specified, this function will use the channel `'MATLAB 1'`. The bit rate is a property of the Application Channel; to change the bit rate, you must use a different Application Channel, or change the bit rate by running the Vector Informatik configuration utility. To run this utility, make sure that `vcanconf.exe` is on your System Path, then type `vcanconf` from a Windows command prompt.

You can specify the Application Channel as follows:

```
profdata = profile_mpc555('can', 'CANChannel', canchannel)
```

`canchannel` specifies the Vector Informatik CAN Application Channel, and must be of the form `'MATLAB 1'`, `'MATLAB 2'` etc.

## Execution Profiling Definitions

**Task turnaround time**
> the elapsed time between start and finish of a task. If the task is not preempted then the task turnaround time is equal to the task execution time.

**Task execution time**
> that part of the time between task start and finish when the task is actually running and not preempted by another task. Note that the task execution time cannot be measured directly, but is inferred from the task start and finish time and the intervening periods during which it was preempted by another task. Note that, in performing these calculations, no account is taken of processor time consumed by the scheduler while switching tasks: this means that, in cases where preemption has occurred, the reported task execution times will overestimate the true values.

**Task overruns**
> occur when a timer task does not complete before that same task is next scheduled to run. Depending on how the real-time scheduler is configured, a task overrun may be handled as a real-time failure. Alternatively, a small number of concurrent task overruns may be

allowed in order to accommodate cases where a task occasionally takes longer than normal to complete.
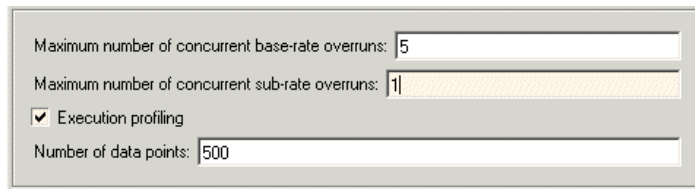
See also "Interpreting the Execution Profiling Graphic" on page 2-54.

### The Execution Profiling Block

See the Block Reference section MPC555 Execution Profiling via SCI1 or MPC555 Execution Profiling via CAN A.

## Real-Time Workshop Options for Execution Profiling

You can see these options on the MPC5xx real-time options (2) section (under **Real-Time Workshop** in the tree) of the **Configuration Parameters** dialog.

```
Maximum number of concurrent base-rate overruns: 5
Maximum number of concurrent sub-rate overruns: 1
☑ Execution profiling
Number of data points: 500
```

**Execution profiling**

If this option is checked then the generated code for the model will be instrumented with function calls at the beginning and end of each task or ISR to be profiled. These function calls read a timer (on MPC555 it is the decrementer timer) and log this reading along with a task identifier.

When code for the model is generated, these functions will update data on the worst-case turnaround time for each timer-based task as well as the worst-case number of concurrent task overruns, whenever a previous worst case value is exceeded. Additionally, when a trigger is provided, data will be logged over a period of time to record all task start and task finish times. The trigger signal can be supplied by the execution profiling blocks. See MPC555 Execution Profiling via SCI1 and MPC555 Execution Profiling via CAN A.

**Number of data points**

When a snapshot of task and ISR activity is logged this data is stored in memory that is statically allocated at build time. Each data point

requires 8 bytes on the MPC555. The larger the number of data points to be stored, the more RAM that must be reserved for this purpose. At the end of a logging run, the data must be uploaded to the host computer for analysis; this is typically achieved by using the execution profiling blocks.

## Real-Time Workshop Overrun Options

These Real-Time Workshop options configure the allowable number of task overruns. You can see these options on the MPC5xx real-time options (2) section (under **Real-Time Workshop** in the tree) of the **Configuration Parameters** dialog.



You can use the options **Maximum number of concurrent base-rate overruns** and **Maximum number of concurrent sub-rate overruns** to configure the behavior of the scheduler when any of the timer based tasks do not complete within their allowed sample time. It is useful to allow task overruns in the case where a task may occasionally take longer than usual to complete (e.g. if extra processing is required when a special event occurs); if the task overrun is only occasional then it is possible for the scheduler to 'catch up' after the extra processing has been completed.

If the maximum number of concurrent overruns for any task is exceeded, this is deemed to be a failure and the real-time application is stopped. This in turn will result in a watchdog timer timeout and the processor will be reset.

As an example, if the base rate is 1 ms and the maximum number of concurrent base-rate overruns is set to 5 then it is possible for the base rate task to run for almost 6 ms before failure occurs. Once the overrun has occurred, it is necessary for subsequent executions of the base rate to complete in less than 1 ms in order that the lost time is recovered.

The occurrence of base-rate overruns does not affect the numerical behavior of the algorithm (although reading/writing external devices will of course be delayed).

If sub-rate overruns are allowed then the transfer of data between different rates (via rate-transition blocks) in the model may be affected; this causes the numerical behavior in real-time to differ from the behavior in simulation. To see an illustration of this effect try running the demo model `mpc555rt_multitasking`. To disallow sub-rate overruns and ensure that this effect does not occur, you should set **Maximum number of concurrent sub-rate overruns** to zero.

**Note** If the option "Maximum number of concurrent sub-rate overruns" is set to a value greater than zero, then the behavior of any Rate-Transition blocks may be affected. Specifically, if the model contains a Rate Transition block where the option "Ensure deterministic data transfer (maximum delay)" is selected, then this setting may not be honored.

## Interpreting the Execution Profiling Graphic

Dark shaded areas show the region where a task is executing. Light shaded areas show the region where a task is preempted by a higher priority task or ISR. Triangles indicate the beginning of a task. An example is shown following.
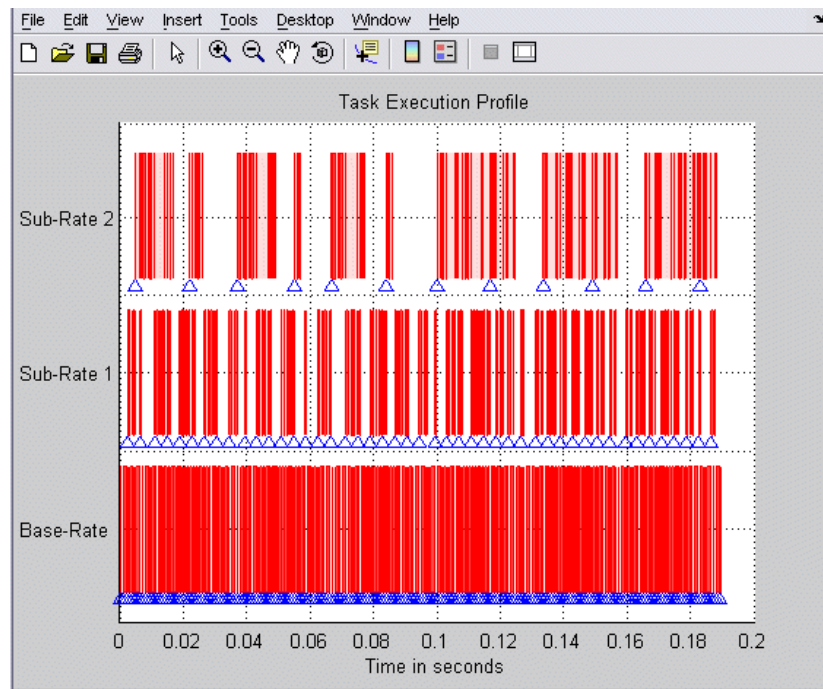
Zoom in to see the details of tasks executing and being preempted, as shown in the following example.
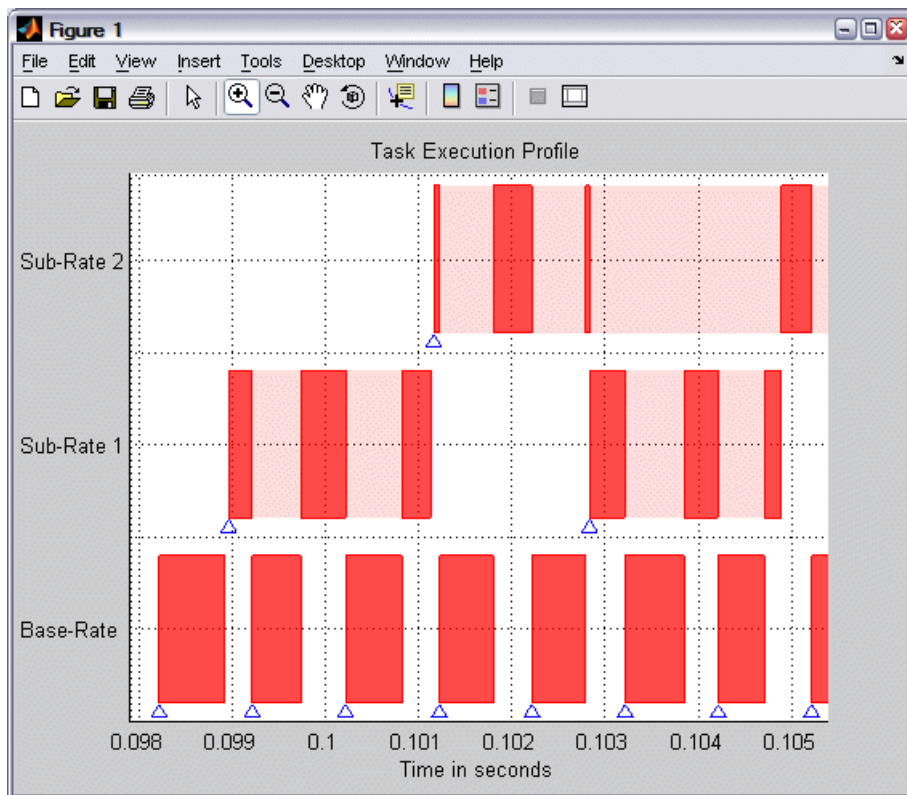
### Enabling Execution Profiling for Device Driver Interrupt Service Routines

By default, execution profiling is not enabled for device driver interrupt service routines. Device drivers that may use interrupt service routines include CAN, TPU and QSPI device drivers.

You can enable execution profiling for device driver interrupt service routines. To do this, you must rebuild the device drivers libraries with a macro PROFILING_ENABLED defined. Follow these steps:

**1** Remove the previously built device driver code using one of the following methods.

**a** Run the command:

```
mpc555_build_drivers('clean')
```

**b** Delete the contents (compiled object code) of the directory

```
matlab\toolbox\rtw\targets\mpc555dk\drivers\src\libsrc\standard\src\bin\COMPILER\XXX
```

where *COMPILER* is one of DIAB or CODE_WARRIOR and *XXX* is the MPC5xx variant you are using.

The second approach will result in a faster rebuild in the next step.

**2** Run the command:

```
mpc555_build_drivers(BUILD_OPTION,'ProfileDeviceDrivers', 'on')
```

Set *BUILD_OPTION* to one of the options 'speed', 'size', or 'debug'.

When rebuilding the driver library using the command mpc555_build_drivers, the compiler and compiler switches used are taken from the currently selected compiler configuration in the MPC5xx Target Preferences.

# Summary of the Real-Time Target

| **In this section...** |
| --- |
| "Code Generation Options" on page 2-58 |
| "Requirements and Restrictions" on page 2-60 |

## Code Generation Options

The real-time target is an extension of the Real-Time Workshop Embedded Coder embedded real-time (ERT) target configuration. The real-time target inherits the code generation options of the ERT target, as well as the general code generation options of the Real-Time Workshop. These options are available under **Real-Time Workshop**, in the tree on the **Configuration Parameters** dialog box; they are documented in the Real-Time Workshop documentation and the Real-Time Workshop Embedded Coder documentation.

Some code generation options of the ERT target are not relevant to the real-time target, and are either unsupported, or restricted in their operation. See "Requirements and Restrictions" on page 2-60 for details.

### Target-Specific Options

The real-time target has several target-specific code generation options. To view or change the setting of these options, select the MPC5xx real-time options(1) section in the **Configuration Parameters** dialog. This figure shows the options at their default settings.



- **Optimize compiler for** — Select speed, size, debug, or custom.

  This option controls compiler optimization switches used during the build process. The exact effect of the optimization switches depends on whether

you are using the Wind River or CodeWarrior compiler. You can optimize for performance by choosing the `speed`, `size`, or `debug` options, or define your own (the `custom` option). You can edit these preferences here in the **Compiler optimization switches** edit box if you want to apply changes to the current model (**Optimize compiler for** will change to `custom`). You can also edit the defaults for these settings in the **Target Preferences** dialog if you want to apply these changes to several models. See "Compiler Optimization Switches" on page 1-21 for more information.

- **Target memory model** Select either `FLASH` or `RAM`.

  If you select the `FLASH` option, files in a format suitable for downloading into the MPC555 on-chip flash memory are written. If you select the `RAM` option, files in a format suitable for downloading into RAM are generated.

  In both cases these two files are generated, with this naming convention:

  - *model*_flash.s19 or *model*_ram.s19 — code only, for CAN download

  - *model*_flash.elf or *model*_ram.elf — for BDM download, containing code and optional debugging symbols if you choose a `debug` build in the **Optimize compiler for** settings

- **Build action**

  - `None` — code generation only.

  - `Launch_Download_Control_Panel` — on completion of code generation the **Download Control Panel** utility is opened.

  - `Run_via_BDM` — on completion of code generation download over BDM connection automatically starts and on completion the code is run.

  - `Debug_via_BDM` — on completion of code generation download over BDM connection automatically starts. When the download is complete the code stops at the first line in debug mode, so you can step through the code.

- **Use prebuilt RTW libraries**

  This check box option (selected by default) determines whether prebuilt RTW libraries, compiled with default compiler switches, are linked against during compilation of the generated code. When this option is not selected, the source modules that comprise these libraries will be compiled individually in the model build directory, using the currently selected compiler switches.

Using prebuilt RTW libraries saves a considerable amount of time during the build process.

# Requirements and Restrictions

### MPC555 Resource Configuration Block Required

To generate code from a model using the Target for Freescale MPC5xx real-time target, an MPC555 Resource Configuration block must be included in the model. The MPC555 Resource Configuration block is required even for models that do not contain any MPC555 device driver blocks.

---

**Note** When using device driver blocks from the Target for Freescale MPC5xx libraries in conjunction with the MPC555 Resource Configuration block, do not disable or break library links on the driver blocks. If library links are disabled or broken, the MPC555 Resource Configuration block will operate incorrectly. See MPC555 Resource Configuration for further information.

---

### Model Reference and Driver Blocks

Referenced sub-models that contain driver blocks (including the MPC555 Resource Configuration block) cause build failures. All driver blocks from Target for Freescale MPC5xx must be placed in the top level model. It is not possible to include driver blocks in any of the referenced sub-models.

### Restricted Code Generation Options

Certain ERT code generation options are not supported by the real-time target. If these options are selected, the real-time target either ignores the option or issues an error message during the build process. Real-Time Target Restricted Code Generation Options on page 2-61 summarizes these restricted options.

**Real-Time Target Restricted Code Generation Options**

| Option | Restriction |
|---|---|
| **MAT-file logging** | Ignored if selected; build process proceeds |
| **Create Simulink (S-function) block** | Error if selected; build process terminates |
| **External mode** | Error if selected; build process terminates |
| **Generate an example main program** | This option should not be selected for the real-time target. The real-time target supplies a target-specific main program, mpc555dk_main.c. Ignored if selected; build proceeds with a warning. |
| **Generate reusable code** | Error if selected; build process terminates |

# Performance Tips

| **In this section...** |
| --- |
| "Run the Model Advisor" on page 2-62 |
| "Increase the System Clock Beyond the Default 20 MHz" on page 2-62 |
| "Use Flash Instead of RAM" on page 2-62 |
| "TouCAN Interrupt Generator Block Performance Tips" on page 2-63 |
| "Optimized Target Function Library" on page 2-63 |

## Run the Model Advisor

Following the suggestions in the Model Advisor report may result in faster on-target execution.

## Increase the System Clock Beyond the Default 20 MHz

The default system clock frequency is 20 MHz. For higher performance, you should consider increasing the system clock frequency up to 40 MHz, which is the maximum for the MPC555 device. Other processor variants may support higher System Clock Frequencies depending on your development board. Please consult your development board documentation for details.

For more information, see:

- System Clock and Related Parameters for information on how to change system clock parameters.

- Switch Target Configuration; this is a utility block you can use to apply some predefined configurations.

## Use Flash Instead of RAM

Configure the model to run from internal Flash (rather than external RAM) because this has faster memory access. See "Downloading Application Code" on page 2-23.

## TouCAN Interrupt Generator Block Performance Tips

When using the TouCAN Interrupt Generator block, you can improve performance as follows:

- Disable **Use floating point** in the TouCAN Interrupt Generator (if possible). This will save significant time during ISR context switches (of which there may be many, depending on the application).

- Minimize the code that runs in the context of the ISR. Try and move as much code out of the ISR (function-call subsystem) as possible to speed up individual ISRs. This should allow an increase in the rate at which CAN messages can be received on that buffer.

## Optimized Target Function Library

If your model contains floating-point mathematical function blocks (e.g., trigonometric functions, log functions), then you should use target optimized function libraries. Select the `Target for Freescale MPC5xx (ISO)` option for the **Target function library** (on the Real-Time Workshop Interface pane of the Configuration Parameters dialog box) to use the CodeWarrior or Diab ISO C function library. This generates calls to the Freescale CodeWarrior or WindRiver Diab ISO/IEC 9899:1999 math library for floating-point functions as appropriate.

When you create new models with the `mpc555rt.tlc`, `mpc555rt_grt.tlc` or `mpc555exp.tlc` **System target file**, the `Target for Freescale MPC5xx (ISO)` is automatically selected for the **Target function library** setting.

# PIL Cosimulation

This section includes the following topics:

| | |
|---|---|
| Overview of PIL Cosimulation (p. 3-3) | Basic concepts you will need to know to use cosimulation effectively in your design process. |
| Tutorial 1: Building and Running a PIL Cosimulation (p. 3-6) | A hands-on, step-by-step introduction to cosimulation with the PIL target, using a plant/controller demonstration model. |
| Tutorial 2: Using the Demo Model in Simulation (p. 3-19) | In addition to building code suitable for cosimulation, the PIL target builds components you can use in closed-loop and software-in-the-loop (SIL) simulations. This tutorial shows you how to use these components. |
| PIL Target Summary (p. 3-20) | Summary of code generation options of the PIL target; restrictions and limitations of the PIL target. |
| Algorithm Export Target (p. 3-25) | The Algorithm Export (AE) target generates only the code that implements the algorithm of your model or subsystem. This is useful for code analysis and interfacing to hand-written or legacy code. |

This section describes the extended HTML code generation report.

Summary of code generation options and restrictions for algorithm export.

# Overview of PIL Cosimulation

| **In this section...** |
| --- |
| "What Is PIL Cosimulation?" on page 3-3 |
| "Why Use Cosimulation?" on page 3-3 |
| "How Cosimulation Works" on page 3-4 |

## What Is PIL Cosimulation?

Target for Freescale MPC5xx supports *processor-in-the-loop* (PIL) *cosimulation*, a technique that is designed to help you evaluate how well a candidate control system operates on the actual target processor selected for the application.

The Target for Freescale MPC5xx (processor-in-the-loop) target is an extended version of the embedded real-time (ERT) target configuration, designed specifically for PIL cosimulation. We refer to this configuration as the *PIL target*.

## Why Use Cosimulation?

PIL cosimulation is particularly useful for simulating, testing and validating a controller algorithm in a system comprising a *plant* and a *controller*. In classic closed-loop simulation, Simulink and Stateflow model such a system as two subsystems and the signals transmitted between them, as shown in this block diagram.
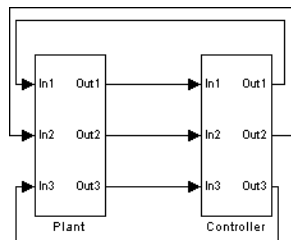


Your starting point in developing a plant/controller system is to model the system as two subsystems in closed-loop simulation. As your design

progresses, you can use Simulink external mode with standard Real-Time Workshop targets (such as GRT or ERT) to help you model the control system separately from the plant.

However, these simulation techniques do not help you to account for restrictions and requirements imposed by the hardware. When you finally reach the stage of deploying controller code on the target hardware, you may need to make extensive adjustments to the controller system. Once these adjustments are made, your deployed system may diverge significantly from the original model. Such discrepancies can create difficulties if you need to return to the original model and change it.

PIL cosimulation addresses these issues by providing an intermediate stage between simulation and deployment. The term *cosimulation* reflects a division of labor in which Simulink models the plant, while code generated from the controller subsystem runs on the actual target hardware. In a PIL cosimulation, the target processor participates fully in the simulation loop — hence the term *processor-in-the-loop*.

## How Cosimulation Works

This figure illustrates how the plant (P) and controller (C) components interact in a PIL cosimulation

# PIL Cosimulation
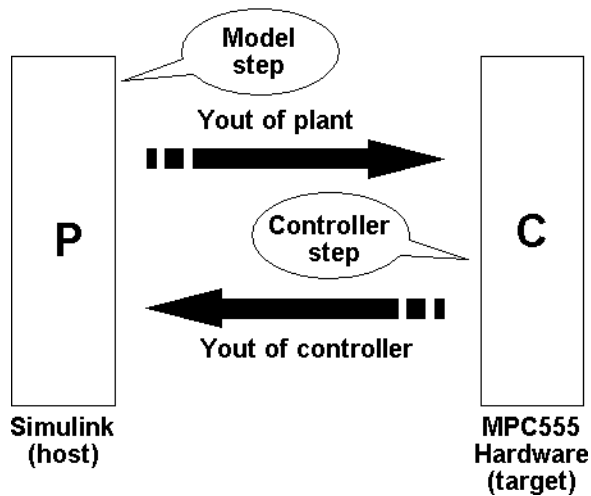


In a PIL cosimulation, Real-Time Workshop Embedded Coder generates efficient code for the control system. This code runs (in simulated time) on a target board using the intended microcontroller. The plant model remains in Simulink without the use of code generation.

During PIL cosimulation, Simulink simulates the plant model for one sample interval and exports the output signals (Yout of the plant) to the target board via a communications link. When the target processor receives signals from the plant model, it executes the controller code for one sample step. The controller returns its output signals (Yout of the controller) computed during this step to Simulink, via the same communications link. At this point one sample cycle of the simulation is complete and the plant model proceeds to the next sample interval. The process repeats and the simulation progresses.

To learn about PIL cosimulation though hands-on experience, see "Tutorial 1: Building and Running a PIL Cosimulation" on page 3-6.

# Tutorial 1: Building and Running a PIL Cosimulation

| **In this section...** |
| --- |
| |
| |
| |
| |
| |
| |
| |

## Before You Begin

In this tutorial, you will use a subsystem in a Simulink model as a component in simulations on your host computer, and also in a PIL cosimulation running on your phyCORE-MPC555 board.

Before working with this tutorial, you should read and follow the procedures in "Setting Up and Verifying Your Installation" on page 1-16. Make sure that the target preferences are set up appropriately for your development system (CodeWarrior or Wind River) as described in "Setting Target Preferences" on page 1-17.

## Hardware Connections

The PIL target requires that you have a serial cable connection. You can also use serial and CAN, or serial with a BDM connection.

Serial cable is required for host/target PIL communications whilst the model is running, and downloads can occur over serial or CAN so the minimal requirement is a single serial cable. BDM is not required but can be used if desired.

We assume that you have made the following connection, as described in the "Interfacing the phyCORE-MPC555 to a Host PC" section of the

*phyCORE-MPC555 Quickstart Instructions* manual: Host PC serial (COM1) port to the RS232-1 (P2) connector on the phyCORE-MPC555 board.

## The Demo Model

We have provided a demo model for your use. The Fault-Tolerant Fuel Control System model, shown in Fault-Tolerant Fuel Control System Model on page 3-7, consists of a plant model with a controller subsystem, the `Fuel Rate Controller` subsystem.
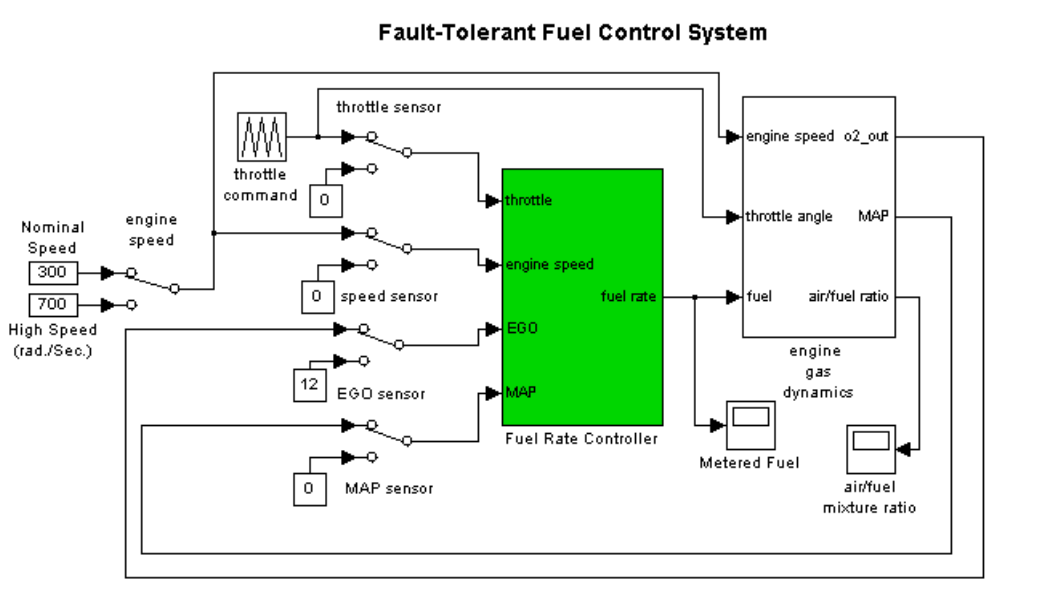


**Fault-Tolerant Fuel Control System Model**

In the following sections, you will use the demo model and the PIL target to generate the following:

- PIL code to run on the target board. The PIL target automatically invokes the appropriate cross-development tools to compile, link, and (optionally) download and run a target executable.
- A library containing

- The original `Fuel Rate Controller` subsystem block for use in simulation.

- An S-function wrapper block, generated by Real-Time Workshop Embedded Coder, that implements the `Fuel Rate Controller` subsystem for use in software-in-the-loop (SIL) simulation.

- A subsystem block that implements the `Fuel Rate Controller` subsystem on the host side during cosimulation. This subsystem communicates with generated PIL code running on the target board.

- A master configurable subsystem block that represents the above three components. You will plug this block into a plant model and select each of the three components in turn for use in a simulation.

This figure shows a library generated by the PIL target.
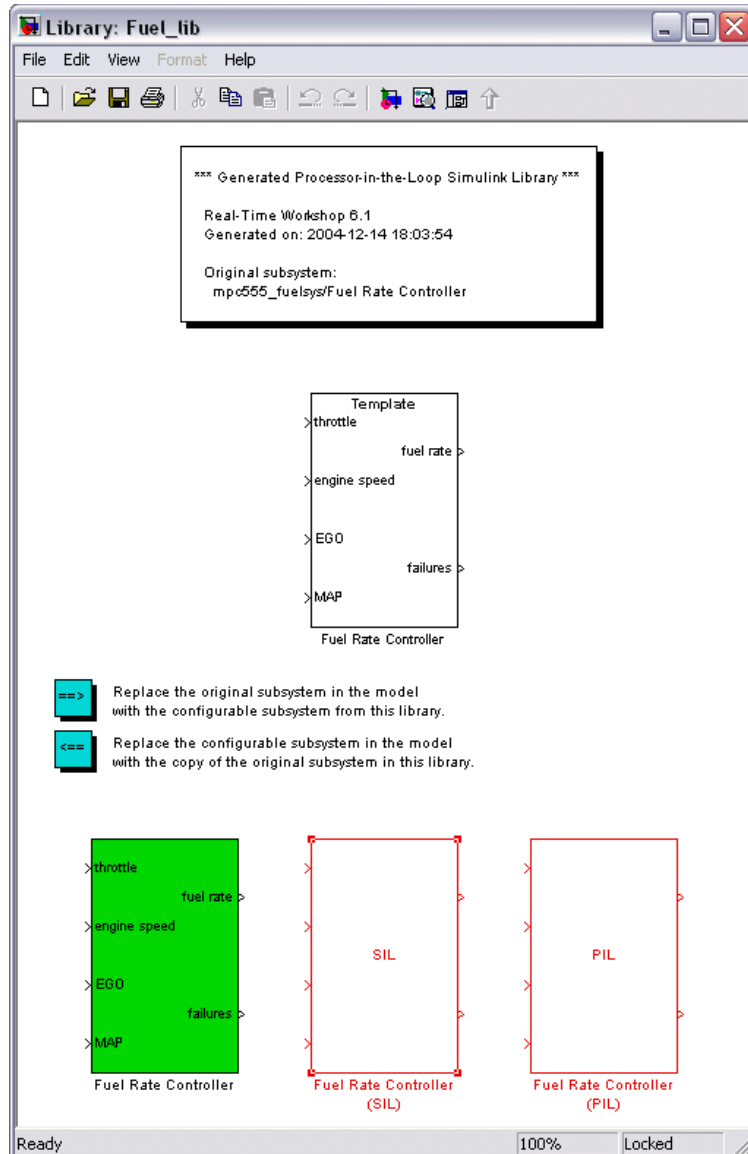
Once you start the build process, there is almost no manual intervention required to build all these components.

After building the components, you will use them in normal simulation, SIL simulation, and PIL cosimulation. You will monitor the results of each simulation via the Scope blocks in the model.

## Setting Up the Model

In this section you will make a local copy of the demo model and configure the model as required by this exercise:

**1** Open the demo model by clicking the link or typing at the command line:

    mpc555_fuelsys

Alternatively you can access the whole MPC555 demo suite by selecting **Start > Demos** and browsing under **Links and Targets**, or **Start > Links and Targets > Target for Freescale MPC5xx > Demos**. The model is located in the directory

    *matlabroot*/toolbox/rtw/targets/mpc555dk/mpc555demos.

The path *matlabroot* should be the location where MATLAB is installed.

**2** Save a copy of the demo model, mpc555_fuelsys.mdl to your working directory.

Next, check that the model is correctly configured for use with Target for Freescale MPC5xx.

**1** Click on the Fuel Rate Controller subsystem, then choose **Configuration Parameters** from the **Simulation** menu. The **Configuration Parameters** dialog opens.

**2** Select **Real-Time Workshop** in the tree.

**3** Observe the RTW system target file setting on the **General** tab. The target configuration should be as shown in this figure.
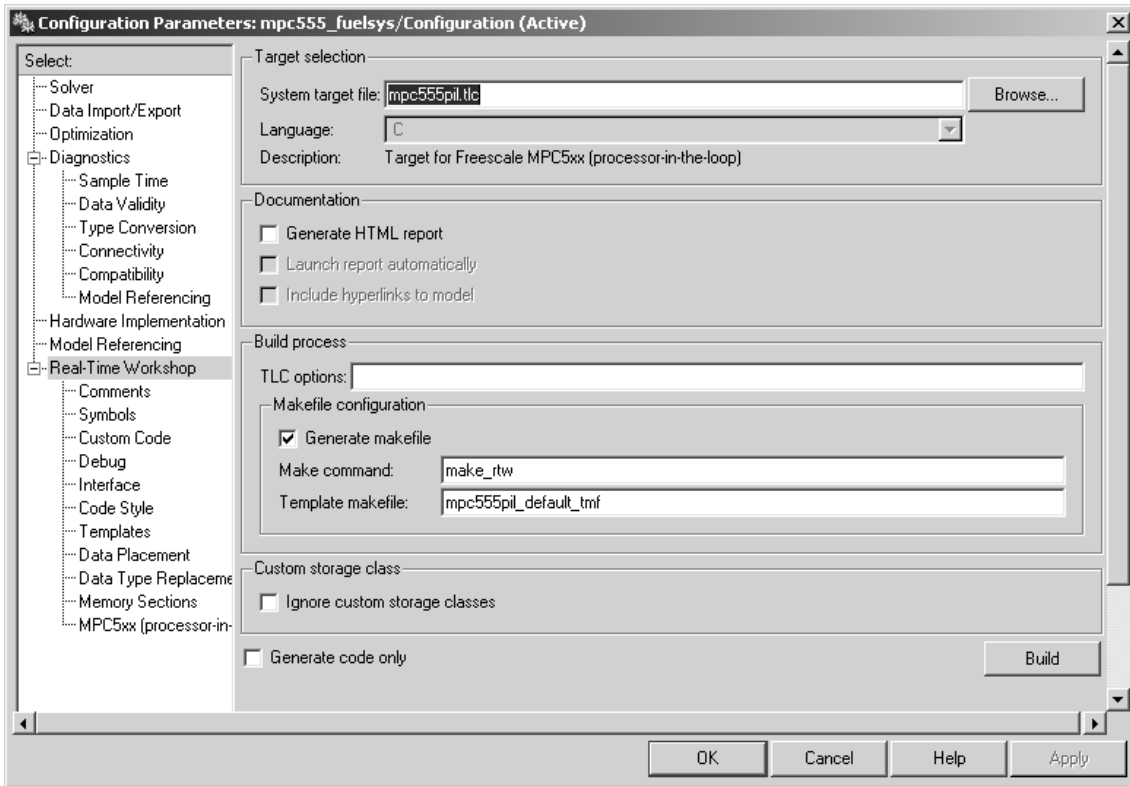
4 To see how to change target configuration settings, click the **Browse** button to open the System Target File Browser, and observe the available **Target for Freescale MPC5xx** targets — algorithm export, processor-in-the-loop, and real-time target. Leave the selected target at mpc555pil.tlc. Click **Cancel** to close the Browser and return to the **Real-Time Workshop** pane.

5 SelectMPC5xx (processor-in-the-loop) options in the tree.
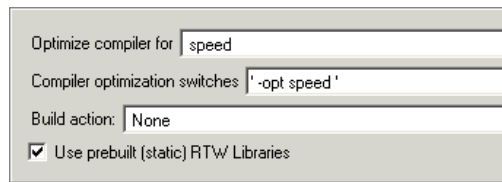
**6** Select Launch_Download_Control_Panel from the **Build action** drop-down menu. This option automatically invokes the appropriate downloading utility.

**7** Click **Apply**. Then close the **Configuration Parameters** dialog box. If needed, save the model to preserve any changes you have made.

## Building PIL and Simulation Components

In this section, you will build a library of simulation, SIL, and PIL components from the Fuel Rate Controller subsystem:

**1** Right-click on the Fuel Rate Controller subsystem. A context menu appears. Select **Build Subsystem** from the **Real-Time Workshop** submenu of the context menu.

**2** The **Build code for Subsystem** window opens. This window displays information about each variable (or data object) that is referenced as a block parameter in the subsystem. The window lets you inline or set the storage class of individual parameters. We will not be concerned with these features in this exercise. Click the **Build** button to continue the code generation and build process.
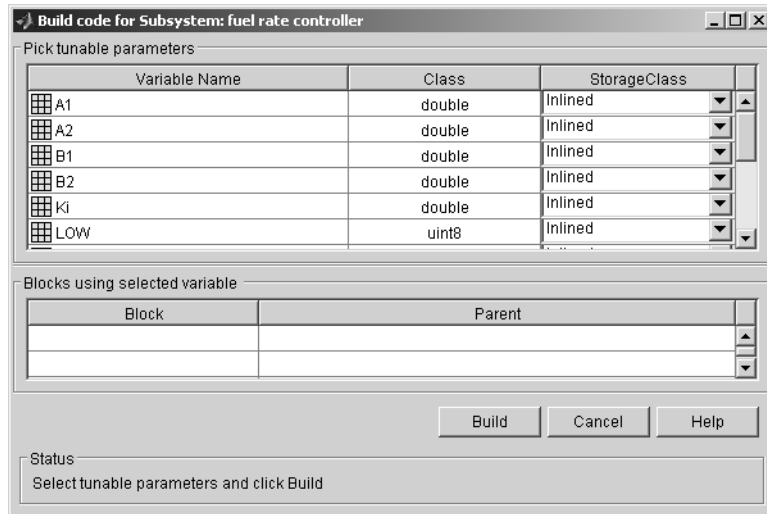
**3** The build process displays status messages in the MATLAB command window. Intermediate Simulink windows are displayed as the build process creates various components.

**4** When the code generation process completes, the PIL target automates the process of compiling, downloading, and executing the generated PIL code that is to run on the target hardware. To accomplish this, the PIL target launches your cross-development system (Wind River or CodeWarrior), compiles and makes the executable, and invokes the **Download Control Panel** to download the code to the target. Click **Start Download** in the **Download Control Panel** to complete the process.

**5** At this point, the generated program is running on the target hardware and waiting for communication to be established with Simulink on the host PC.

**6** The build process has created and opened a library named `Fuel_lib`, as shown in this figure.
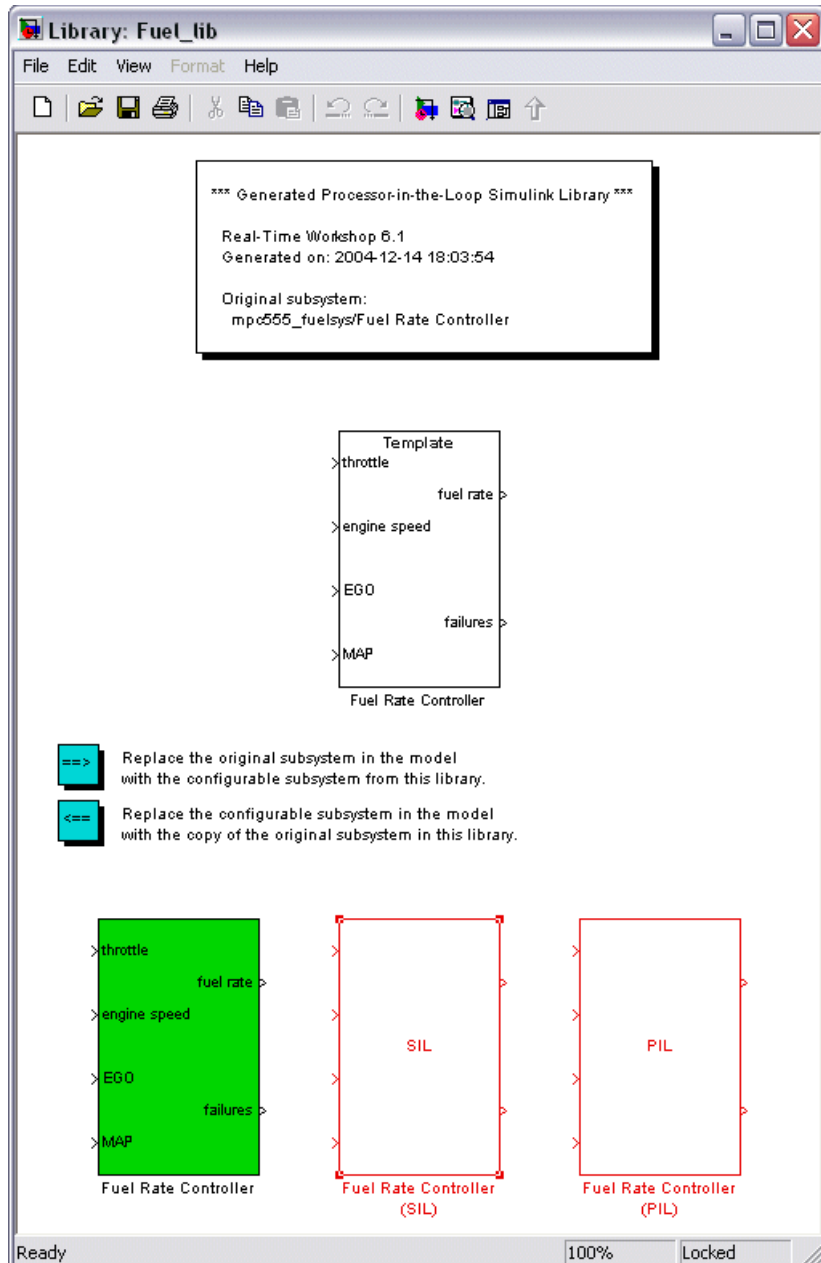
The library contains

- A copy of the original `Fuel Rate Controller` subsystem.
- A Real-Time Workshop Embedded Coder generated S-function, labeled `Fuel Rate Controller (SIL)`.
- A subsystem block that communicates with generated PIL code running on the target board during cosimulation, labeled `Fuel Rate Controller (PIL)`.
- A master configurable subsystem block referencing the other three blocks. The default block choice for this subsystem is the original `Fuel Rate Controller` subsystem.

The configurable subsystem, when plugged into the model, lets you choose which of the three library components will perform the controller functions in the model. We will use the configurable subsystem in the following sections.

The library window also contains the following controls:

- A button that lets you replace the original (generating) subsystem in the model with the generated configurable subsystem.
- A button that lets you do the inverse, i.e., remove the configurable subsystem from the model from the original model and replace it with the original (generating) subsystem from the library.

The library window documents the name of the original model/subsystem from which the library was generated,

## Using the Demo Model In a PIL Cosimulation

In this section, we will plug the configurable subsystem into the demo model, select the PIL component, and use it in a PIL cosimulation:

**1** Click on the `Fuel_lib` library window to activate it. Double-click on the button labeled **Replace the original subsystem in the model with the configurable subsystem from this library**.

**2** The `mpc555pil_fuelsys` model window is now the active window. The original `Fuel Rate Controller` subsystem has been deleted from the model. It has been replaced by the configurable subsystem from the `Fuel_lib` library. The configurable subsystem is automatically connected to the same signals that the original `Fuel Rate Controller` subsystem was connected to.

---

**Note** It is important to be aware that the insertion of the configurable subsystem into the containing model establishes a link between the model, `mpc555pil_fuelsys`, and the library, `Fuel_lib`. The library has information about the model and subsystem from which it was generated. The model, in turn, has information about the library from which the configurable subsystem comes. This linkage is based on the names of the library and the model, and will be broken if either is renamed. To avoid errors, treat the model and library as a single unit, and do not rename either.

---

**3** Save the model.

**4** Right-click on the configurable subsystem in the model. A context menu appears. Select the **Block choice** menu item and observe the block choice submenu. This figure shows the default block choice selection.
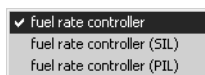


**5** From the **Block choice** submenu of the context menu, select `Fuel Rate Controller (PIL)`.

**6** Open the model's two Scope blocks, if they are not already opened.

**7** Make sure that Simulink is in `Normal` mode. For more information, see the Simulink documentation on Simulation Modes.

**8** You are now ready to run the cosimulation. To start the cosimulation, click the **Start simulation** button in the Simulink toolbar.

The target system now starts executing the controller code. Observe that the output signals computed on the target are displayed on the scopes. The updating of the Scope blocks is slow, relative to a normal simulation, because data is transmitted over the serial line on every model step.

**9** When the simulation completes, the signals displayed on the scopes should appear as shown in Signals Displayed at End of Simulation or Cosimulation on page 3-17.
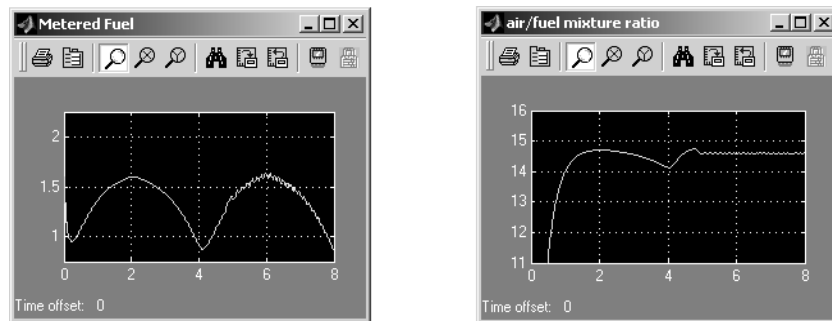


**Signals Displayed at End of Simulation or Cosimulation**

**10** When the cosimulation has completed, or has stopped or paused, the target code enters a wait state until it receives a command to start (or resume) from the host. Restart the cosimulation by clicking the **Start simulation** button again. You can start, stop, restart, pause, or continue a cosimulation exactly as you would a normal simulation. Try each of these operations a few times.

Once your target has been reset, your application will be lost from memory. In this case, you can download the application again by using the **Download Control Panel** from the **Start** menu. Select the `*.s19` file. In this case it will be fuel_ram.s19.

See "Build Process Files and Directories" on page 3-22 for information on the files and directories created by the build process.

## Modifying the Controller Subsystem

Typically during algorithm development you will wish to make modifications to the Controller Subsystem. You can apply your modifications to the Controller Subsystem by changing the original model.

Note that in the `mpc555_fuelsys` demo model the Controller Subsystem is actually a Simulink library block from the `mpc555_fuelsys_project` library, so making modifications may require modification of the library block.

Once you have completed making your modifications to the Controller Subsystem you can go back to step "Building PIL and Simulation Components" in this tutorial to rebuild and download the Controller Subsystem for PIL.

# Tutorial 2: Using the Demo Model in Simulation

| **In this section...** |
| --- |
| "Closed-Loop Simulation" on page 3-19 |
| "SIL Simulation" on page 3-19 |

## Closed-Loop Simulation

In this section, you will continue to use the configurable subsystem in the demo model, using it first in a normal closed-loop simulation and then in a SIL simulation.

**1** Right-click on the configurable subsystem and select `Fuel Rate Controller` from the **Block choice** submenu of the context menu. This selects the controller subsystem that was used in the original model.

**2** Open the Scope blocks and start the simulation. When the simulation completes (simulation time is set to 8 seconds), the signals displayed on the scopes should appear identical to those displayed during the previous cosimulation (see Signals Displayed at End of Simulation or Cosimulation on page 3-17).

## SIL Simulation

**1** Right-click on the configurable subsystem and select `Fuel Rate Controller (SIL)` from the **Block choice** submenu of the context menu.

Selecting this option directs Simulink to call a generated wrapper S-function that implements the controller algorithm in highly efficient Real-Time Workshop Embedded Coder generated code. You can now run a SIL simulation.

**2** Start the simulation. You will notice that the simulation completes much more quickly, due to the efficiency of the generated code. Also, observe that the generated code displays results, on the scopes, that are identical to the previous simulation and cosimulation (see Signals Displayed at End of Simulation or Cosimulation on page 3-17).

# PIL Target Summary

## Code Generation Options

The PIL target is an extension of the Real-Time Workshop Embedded Coder embedded real-time (ERT) target configuration. The PIL target inherits the code generation options of the ERT target, as well as the general code generation options of Real-Time Workshop. These options are available under **Real-Time Workshop**, in the tree on the **Configuration Parameters** dialog box; they are documented in the Real-Time Workshop documentation and the Real-Time Workshop Embedded Coder documentation.

Some code generation options of the ERT target are not relevant to the PIL target, and are either unsupported, or restricted in their operation, by the PIL target. See "Restrictions" on page 3-23 for details.

### Target-Specific Options

The PIL target has four target-specific code generation options: **Optimize compiler for**, **Compiler optimization switches**, **Build action** and **Use prebuilt (static) RTW libraries**. To view or change the setting of these options, select MPC5xx (processor-in-the-loop) options under **Real-Time Workshop** in the tree on the **Configuration Parameters** dialog.

- **Optimize compiler for** — Select speed, size, debug, or custom.

  This option controls compiler optimization switches used during the build process. The exact effect of the optimization switches depends on whether you are using the Wind River or CodeWarrior compiler. You can optimize for performance by choosing the speed, size, or debug options, or define your own (the custom option). You can edit these preferences here in the **Compiler optimization switches** edit box if you want to apply changes to the current model (**Optimize compiler for:** will change to custom). You can also edit the defaults for these settings in the **Target Preferences** dialog if you want to apply these changes to several models. See "Compiler Optimization Switches" on page 1-21 for more information.

- The **Build action** menu has two options that control what action the PIL target takes after completing the code generation process:

  - Launch_Download_Control_Panel: When this option is selected, the PIL target automatically invokes the **Download Control Panel**. When you click **Start Download** the PIL target downloads the generated code to the target board and begins execution of the code.

    Before using this option, make sure that the target preferences (Compiler and Debugger paths) are set correctly.

  - None: When this option is selected, the PIL target does not take any action after code generation completes. To download and run your application, you must do so manually, using your development tools.

  - Run_via_BDM — on completion of code generation download over BDM connection automatically starts and on completion the code is run.

  - Debug_via_BDM — on completion of code generation download over BDM connection automatically starts. When the download is complete the code stops at the first line in debug mode, so you can step through the code.

- **Use prebuilt (static) RTW libraries**

  This check box option (selected by default) saves a considerable amount of time during the build process, as the libraries do not need to be recompiled every time.

### Manual Download

Once a subsystem has been built using the PIL target, it is possible to use the **Download Control Panel** to manually download the generated code to

the target without repeating the entire build process. To do this, use the following procedure:

**1** Select **Start > Links and Targets > Target for Freescale MPC5xx > Launch Download Control Panel**.

**2** Select the required `*.s19` file, and click **Start Download**.

## Build Process Files and Directories

The PIL target creates the following in your working directory:

- A build directory, containing generated source code, object files in their own directory, and a makefile and other control files. The build directory also may contain subdirectories used by Stateflow and by the HTML code generation report generator (see "HTML Code Analysis (RAM/ROM) Report" on page 3-27).

  The naming convention for the build directory is *source*`_mpc555pil`, where *source* is the first word of the generating subsystem or model. For example, the `Fuel Rate Controller` subsystem used in the PIL tutorials generates the build directory `fuel_mpc555pil`.

- The generated library, *source*`_lib.mdl`, and the `.mexw32` components that are bound to the generated PIL and SIL blocks in the library. Note that if you rebuild *source*`_lib.mdl` in the same working directory, a revision number is appended to the *source* string. For example, building from the `Fuel Rate Controller` subsystem used in the PIL tutorials generates `Fuel_lib.mdl`, `fuel1_lib.mdl`, `fuel2_lib.mdl`... `fueln_lib.mdl`.

- Executable PIL code in a format suitable for downloading to the target and execution by your development system (Singlestep or CodeWarrior).

- Project files, debugging symbol files, link maps, and other files specific to your development system (Singlestep or CodeWarrior).

If you do not select the `Launch_Download_Control_Panel` option when you generate code (or if you want to rerun PIL code after it is built), you can use the **Download Control Panel** to manually download and run the generated executable. To do this, see "Manual Download" on page 3-21.

## Restrictions

Please note the following restrictions on the use of the PIL target:

- The PIL target does not support code generation from device driver blocks from the Target for Freescale MPC5xx block libraries. Do not include device driver blocks in your PIL models. See `mpc555_fuelsys_project.mdl` for an example of PIL modeling. This example manages multi-model modeling to deal with RT & PIL operation.

- Do not include To File blocks in your PIL models, they will cause the build to fail.

- Self modifying blocks (such as the Resource Configuration block and other blocks) that modify the PIL subsystem during simulation, may cause an error during simulation of the generated Configurable Subsystem (in original subsystem mode).

  As a workaround it is possible to set the MaskSelfModifiable parameter of the original subsystem in the generated PIL library. To do this select the original subsystem in the generated PIL library with the mouse, and then run the following command in the MATLAB command prompt:

  ```
  set_param (gcb, 'MaskSelfModifiable', 'on')
  ```

  Note that we recommend not placing driver blocks (such as the Resource Configuration block) inside the PIL subsystem.

- If you change the cross-compiler you use with the PIL target (from Wind River to CodeWarrior or vice versa), you should rebuild your PIL models in a clean directory, or delete all files from the models' code generation directories. The PIL build process expects to start with a clean directory, or a directory created in the process of building with the same compiler. Leftover components built by a different compiler cause errors.

- In a plant/controller simulation where the controller is built via the PIL target, the plant model can contain any Simulink blocks, including a combination of continuous-time and discrete-time blocks. However, the controller subsystem must not include any continuous-time blocks. This is because PIL uses the Embedded Coder S-function Generation feature; this feature does not support continuous sample times. However, note that, standard Embedded Coder code generation, as used by the MPC555 RT target, does support continuous-time blocks.

3-23

- The superseded version of the Vector CAN Configuration block should not be placed inside a PIL subsystem. Instead, the model can be updated to use the current Vector CAN Configuration block, which can be placed inside a PIL subsystem.

- Parameters with the following storage requirement are not supported for PIL. If a model contains parameters where the storage class (e.g., custom storage class) of the data objects requires storage in the *model*.c module, then "unresolved external symbol" link errors occur during the processor-in-the-loop (PIL) build process.

- Model directories must be located either an actual hard drive on your PC, or a mapped drive. Do not use a UNC network path. If you run (simulate) a model from a Universal Naming Convention (UNC) network directory (such as \\Server\user\work), errors are produced.

- Vectors are not supported at the PIL boundary.

- Certain ERT code generation options are not supported by the PIL target. If these options are selected, the PIL target either ignores the option or issues an error message during the build process. PIL Target Restricted Code Generation Options on page 3-24 summarizes these restricted options.

**PIL Target Restricted Code Generation Options**

| Option | Restriction |
|--------|-------------|
| **MAT-file logging** | Ignored if selected; build process proceeds |
| **Generate ASAP2 file** | Ignored if selected; build process proceeds |
| **External mode** | Error if selected; build process terminates |
| **Generate an example main program** | This option should not be selected for the PIL target. |
| **Generate reusable code** | Error if selected; build process terminates |
| **Target function library** | C89/C90(ANSI) is the default and is not configurable. |

# Algorithm Export Target

The Target for Freescale MPC5xx Algorithm Export (AE) target is an aid to code analysis and interfacing. The target generates only the code that implements the algorithm of your model or subsystem, without any overhead for PIL host/target communications or other operations extraneous to the model. Such purely algorithmic code is easier to interface to your hand-written or legacy code than code generated by the PIL or RT targets.

Another application of the AE target is to use it to produce a code generation report. Since only model code is included, you can more easily analyze the code generated from your model.

The AE target supports both the CodeWarrior and Wind River cross-compilers, as specified in your target preferences (see "Setting Target Preferences" on page 1-17).

To use the AE target,

**1** Select **Configuration Parameters** from the **Simulation** menu. The **Configuration Parameters** dialog opens.

**2** Select **Real-Time Workshop** in the tree.

**3** In the **Target selection** pane, click on the **Browse** button to open the **System Target File Browser**. In the browser, select `Target for Freescale MPC5xx (algorithm export)` target. Click **OK** to close the browser and return to the **Configuration Parameters** dialog.

**4** Select **Templates** in the tree and make sure **Generate an example main program** is not selected.

**5** Follow the usual procedure for generating code from your model or subsystem.

We recommend using the AE target in conjunction with the Target for Freescale MPC5xx HTML code generation report (see "HTML Code Analysis (RAM/ROM) Report" on page 3-27). If you select the **Generate HTML report** option as described in the next section, you can view a profiling report that includes detailed itemization of RAM and ROM usage for all code and

data sections, and a complete memory map of the generated code. You can also easily examine the generated code via hyperlinks in the code generation report.

# HTML Code Analysis (RAM/ROM) Report

Target for Freescale MPC5xx supports an extended version of the Real-Time Workshop Embedded Coder HTML code generation report. You can generate reports for the real-time target as well as the processor-in-the-loop (PIL) target and algorithm export (AE) target.

The extended code generation report includes an additional section, the **Code profile report** for the generated application. See the Embedded Coder documentation for information on the other report sections.

The code profile report section includes a detailed itemization of RAM and ROM usage for all code and data sections, and a complete memory map of the generated code. The report is generated from the memory map file (MAP file) created during the application build process (compilation and linking). This MAP file is appended to the end of the report to provide the complete details of the memory layout of the application.

An example Code Profile Report is shown below.

To generate a code generation report and view the profiling report,

**1** On the **Real-Time Workshop** options in the **Configuration Parameters** dialog, make sure that the **Generate code only** option is not selected.

The reason for this step is that the Target for Freescale MPC5xx extended code generation report obtains information from MAP files that are created by your cross-compiler during the build process. If the **Generate code only** option is on, these files are not generated, which prevents the generation of the code generation report.

**2** Select **Generate HTML report**, as shown in this figure.
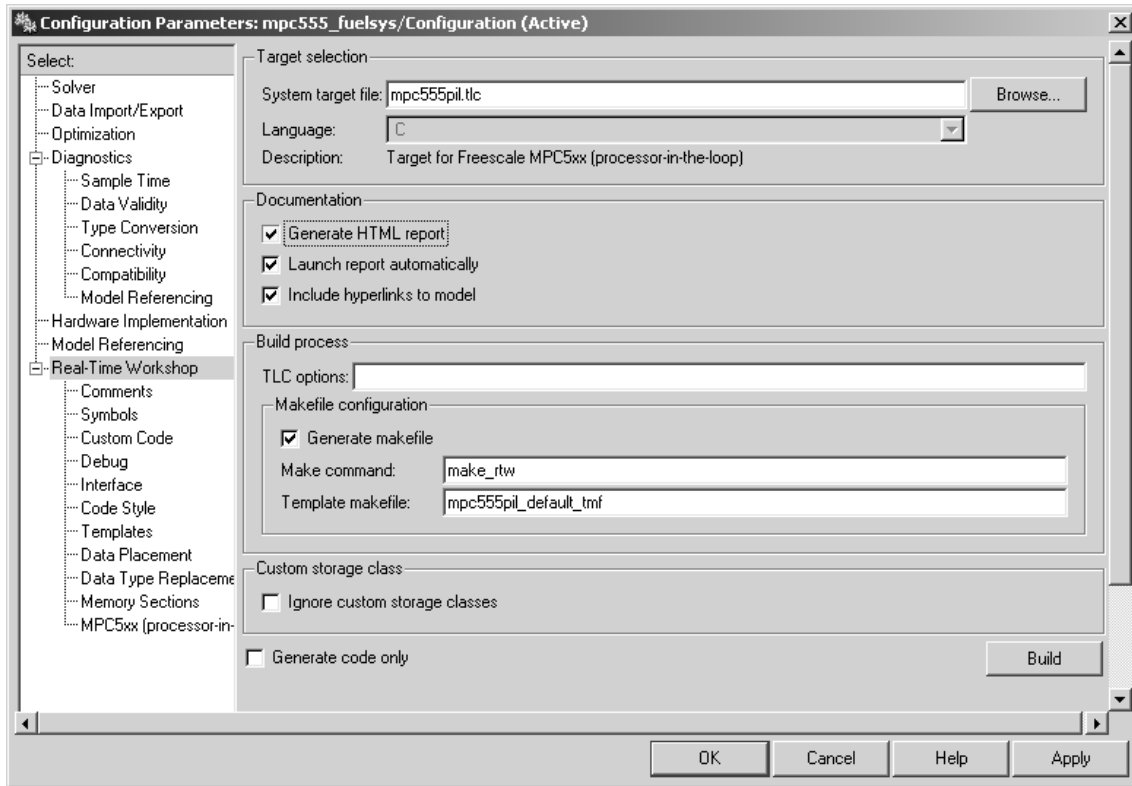
**3** Follow the usual procedure for generating code from your model or subsystem.

**4** Real-Time Workshop writes the code generation report file in the build directory. The file is named *model*_codegen_rpt.html or *subsystem*_codegen_rpt.html.

**5** Real-Time Workshop automatically opens the MATLAB Help browser and displays the code generation report. Alternatively, you can view the code generation report in your Web browser.

**6** To view the profiling report, click on the **Code profile report** link in the **Contents** pane of the report.

A portion of an example code profile report is shown following. The raw memory map (MAP file) is at the bottom of the report.

# Code Profile Report

© The MathWorks, Inc.

**Compiler: CodeWarrior**

- Model Memory Summary
- Entire Memory Summary
- Memory MAP Summary
- Model Memory Detail
- Entire Memory Detail
- Memory MAP
- Original MAP file

## Model Memory Summary

| Memory Regions | Size [in bytes] |
|---|---|
| Read/Write Memory Usage(without rtwlib) | 32 |
| Read Only Memory Usage(without rtwlib) | 1476 |
| Read/Write Memory Usage(with rtwlib) | 32 |
| Read Only Memory Usage(with rtwlib) | 1476 |

## Entire Memory Summary

| Memory Regions and Section Mapping | Size [in bytes] |
|---|---|
| Read/Write Memory Usage (.bss .sbss .sdata .data ) | 708 |
| Read Only Memory Usage (.text .sdata2 .init .application_bios .usr_init .ram_entry ) | 5568 |

## Memory MAP Summary: RAM Target :: Off-chip RAM, On-chip RAM

| Memory Type and Section Mapping | Size [in bytes] |
|---|---|
| OFF-CHIP_RAM<br>(.ram_entry .text .init .usr_init .rodata .ctors .dtors .sdata2 .sbss2 .data .sdata .PPC.EMB.sdata0 .sbss .bss .PPC.EMB.sbss0 .application_bios ) | 6524 |
| ON-CHIP_RAM (.application_bios ) | 120 |

## Model Memory Detail

| Read/Write Memory Usage (with rtwlib) | File | Section | Size [in bytes] |
|---|---|---|---|
| Target_LED_DWork | Target_LED.o | .sbss | 8 |

# Algorithm Export Target Summary

| In this section... |
| --- |
| "Code Generation Options" on page 3-31 |
| "Restrictions" on page 3-31 |

## Code Generation Options

The Algorithm Export (AE) target is an extension of the Real-Time Workshop Embedded Coder embedded real-time (ERT) target configuration. The AE target inherits the code generation options of the ERT target, as well as the general code generation options of Real-Time Workshop. These options are available under **Real-Time Workshop**, in the tree on the **Configuration Parameters** dialog box; they are documented in the Real-Time Workshop documentation and the Real-Time Workshop Embedded Coder documentation.

Some code generation options of the ERT target are not relevant to the AE target, and are either unsupported, or restricted in their operation, by the AE target. See "Restrictions" on page 3-31 below for details.

The only target-specific option for AE target is **Use prebuilt (static) RTW libraries**. This check box option (selected by default) saves a considerable amount of time during the build process, as the libraries do not need to be recompiled every time.

## Restrictions

Certain ERT code generation options are not supported by the AE target. If these options are selected, the AE target either ignores the option or issues an error message during the build process. AE Target Restricted Code Generation Options on page 3-32 summarizes these restricted options.

**AE Target Restricted Code Generation Options**

| Option | Restriction |
|---|---|
| **MAT-file logging** | Ignored if selected; build process proceeds |
| **Create Simulink (S-function) block** | Error if selected; build process terminates |
| **Generate ASAP2 file** | Ignored if selected; build process proceeds |
| **External mode** | Error if selected; build process terminates |

You must not include driver blocks in your model for Algorithm Export. The AE target is designed to generate only the code that implements the algorithm of your model or subsystem, without any overhead for PIL host/target communications or other operations extraneous to the model, so you should not be including driver blocks.

**4**

# Blocks — By Category

# MPC555 Drivers

## Top-Level Blocks

| | |
|---|---|
| MPC555 Resource Configuration | Support device configuration for MPC5xx CPU and MIOS, QADC, and TouCAN submodules |
| Watchdog | In case of application failure, time out and reset processor |

## CAN 2.0B Controller Module (TouCAN)

| CAN Calibration Protocol (MPC555) | Implement CAN Calibration Protocol (CCP) standard |
|---|---|
| TouCAN Error Count | Count transmit and receive errors detected on selected TouCAN modules |
| TouCAN Fault Confinement State | Indicate state of TouCAN module |
| TouCAN Interrupt Generator | Generate asynchronous function-call trigger when CAN interrupt occurs |
| TouCAN Receive | Receive CAN messages from TouCAN module on MPC5xx |
| TouCAN Soft Reset | Reset TouCAN module |
| TouCAN Transmit | Transmit CAN message via TouCAN module on MPC5xx |
| TouCAN Warnings | Flag excessively high transmit or receive error counts on TouCAN modules |

For information about CAN message blocks and CAN drivers, see the "CAN Blockset Reference".

## Enhanced Queued Analog-to-Digital Converter Module-64

| QADCE Analog In | Input driver enables use of Queued Analog-Digital Converter (QADC64) in continuous scan mode on MPC56x (561-6) |
|---|---|
| QADCE Digital In | Input driver enables use of Queued Analog-Digital Converter (QADC64) pins as digital inputs on MPC56x (561-566) |

## Execution Profiling

| | |
|---|---|
| MPC555 Execution Profiling via CAN A | Provide CAN interface to execution profiling engine via CAN channel A |
| MPC555 Execution Profiling via SCI1 | Provide serial interface to execution profiling engine |

## Interrupts

| | |
|---|---|
| Asynchronous Rate Transition | Transfer data between timer-based task and asynchronous task, ensuring data integrity |

## Modular Input/Output System (MIOS1)

| | |
|---|---|
| MIOS Digital In | Input driver for MIOS 16-bit Parallel Port I/O Submodule (MPIOSM) |
| MIOS Digital Out | Output driver for MIOS 16-bit Parallel Port I/O Submodule (MPIOSM) |
| MIOS Digital Out (MPWMSM) | Digital output via the MIOS Pulse Width Modulation Submodule (MPWMSM) |
| MIOS Pulse Width Modulation Out | Output driver for MIOS Pulse Width Modulation Submodule (MPWMSM) |
| MIOS Waveform Measurement | Measure pulse width and pulse period measurement via MIOS Double Action Submodule (MDASM) |

## Queued Analog-to-Digital Converter Module-64

QADC Analog In                          Input driver enables use of Queued
                                        Analog-Digital Converter (QADC64)
                                        in continuous scan mode

QADC Digital In                         Input driver enables use of Queued
                                        Analog-Digital Converter (QADC64)
                                        pins as digital inputs

## Time Processor Unit (TPU3)

TPU3 Digital In                         Configure Time Processor Unit
                                        (TPU3) channel for digital input

TPU3 Digital Out                        Configure Time Processor Unit
                                        (TPU3) channel for digital output

TPU3 Fast Quadrature Decode             Configure pair of TPU3 channels for
                                        Fast Quadrature Decode (FQD)

TPU3 New Input Capture/Input            Configure Time Processor Unit
Transition Counter                      (TPU3) channel for New Input
                                        Capture/Input Transition Counter
                                        (NITC)

TPU3 Programmable Time                  Configure Time Processor Unit
Accumulator                             (TPU3) channel for Programmable
                                        Time Accumulator (PTA)

TPU3 Pulse Width Modulation Out         Configure Time Processor Unit
                                        (TPU3) channel for pulse width
                                        modulation (PWM) output

TPU3 Rectangular Wave                   Configure Time Processor Unit
                                        (TPU3) channel for Rectangular
                                        Wave Output (RECTW)

TPU3 Square Wave                        Configure Time Processor Unit
                                        (TPU3) channel for Square Wave
                                        Output (SQW)

## Serial Communications Interface (SCI)

Serial Receive                 Configure MPC555 for serial receive
                               on either of QSMCM submodules
                               SCI1 or SCI2

Serial Transmit                Configure MPC555 for serial
                               transmit, using one of QSMCM
                               submodules SCI1 or SCI2

## Utilities

Switch External Mode Configuration   Configure model for external mode
                                     or executable building

Switch Target Configuration    Configure model and target
                               preferences to predefined hardware
                               configuration

# CAN Message Blocks and CAN Drivers

For information about CAN message blocks and CAN drivers, see the "CAN Blockset Reference".

# Blocks — Alphabetical List

# Asynchronous Rate Transition

**Purpose**        Transfer data between timer-based task and asynchronous task, ensuring data integrity

**Library**        Target for Freescale MPC5xx/ MPC555 Driver Library/ Interrupts

**Description**    The Asynchronous Rate Transition block is used when reading or writing signals attached to an asynchronous subsystem. An asynchronous subsystem is one which is driven by an interrupt function call trigger. The subsystem is run in the context of an interrupt and not in the context of the model. You must place one of these blocks on each input and output of any subsystem that is triggered asynchronously by an interrupt.

The Asynchronous Rate Transition block copies the signal from input to output while disabling interrupts. This ensures that blocks outside the subsystem that want access to the signal do not get interrupted while reading or writing a signal and end up with corrupt data.

**Dialog Box**
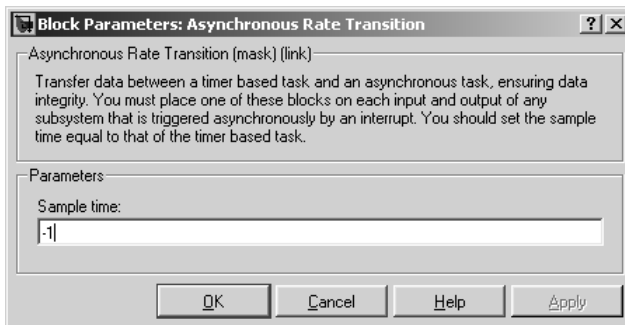
**Sample time**
> You should set the sample time equal to that of the timer based task, as shown in the following example model.
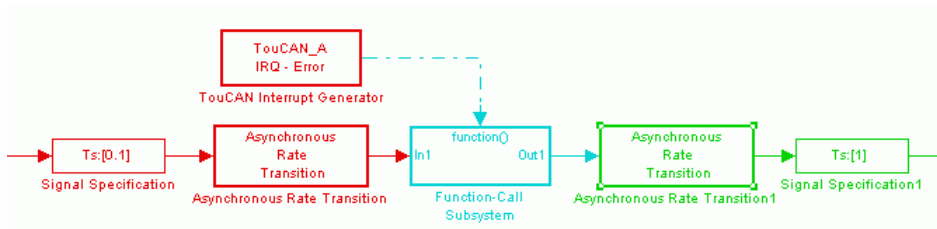
See also the TouCAN Interrupt Generator.

# CAN Calibration Protocol (MPC555)

**Purpose**      Implement CAN Calibration Protocol (CCP) standard

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B
                 Controller Module

**Description**

```
        CCP
CAN Calibration Protocol
```

The CAN Calibration Protocol (MPC555) block provides an
implementation of a subset of the CAN Calibration Protocol (CCP)
Version 2.1. CCP is a protocol for communicating between the target
processor and the host machine over CAN. In particular, a calibration
tool (see "Compatibility with Calibration Packages" on page 5-9)
running on the host can communicate with the target, allowing remote
signal monitoring and parameter tuning.

This block processes Command Receive Object (CRO) messages and
outputs the resulting Data Transmission Object (DTO) and Data
Acquisition (DAQ) messages.

For more information on CCP, refer to *ASAM Standards: ASAM MCD:
MCD 1a* on the Association for Standardization of Automation and
Measuring Systems (ASAM) Web site at `http://www.asam.de`.

You can see an example illustrating how to use the CAN Calibration
Protocol (MPC555) block in the `mpc555rt_ccp` demo.

Note this block is entirely CAN triggered, and so is only designed for the
Real-Time Target (CAN is disabled during PIL and SIL cosimulation.)

### Using the DAQ Output

---

**Note** The CCP Data Acquisition (DAQ) List mode of operation is only
supported with Real-Time Workshop Embedded Coder. If Embedded
Coder is not available then custom storage classes `canlib.signal` are
ignored during code generation: this means that the CCP DAQ Lists
mode of operation cannot be used.

You can use the CCP Polling mode of operation with or without
Real-Time Workshop Embedded Coder.

---

The DAQ output is the output for any CCP DAQ lists that have been set up. You can use the ASAP2 file generation feature of the RT target to

- Set up signals to be transmitted using CCP DAQ lists.
- Assign signals in your model to a CCP event channel automatically (see "Parameter Tuning and Signal Logging" on page 2-33).

Once these signals are set up, event channels then periodically fire events that trigger the transmission of DAQ data to the host. When this occurs, CAN messages with the appropriate CCP/DAQ data appear on the DAQ output, along with an associated function call trigger.

It is the responsibility of the calibration tool (see "Compatibility with Calibration Packages" on page 5-9) to use CCP commands to assign an event channel and data to the available DAQ lists, and to interpret the synchronous response.
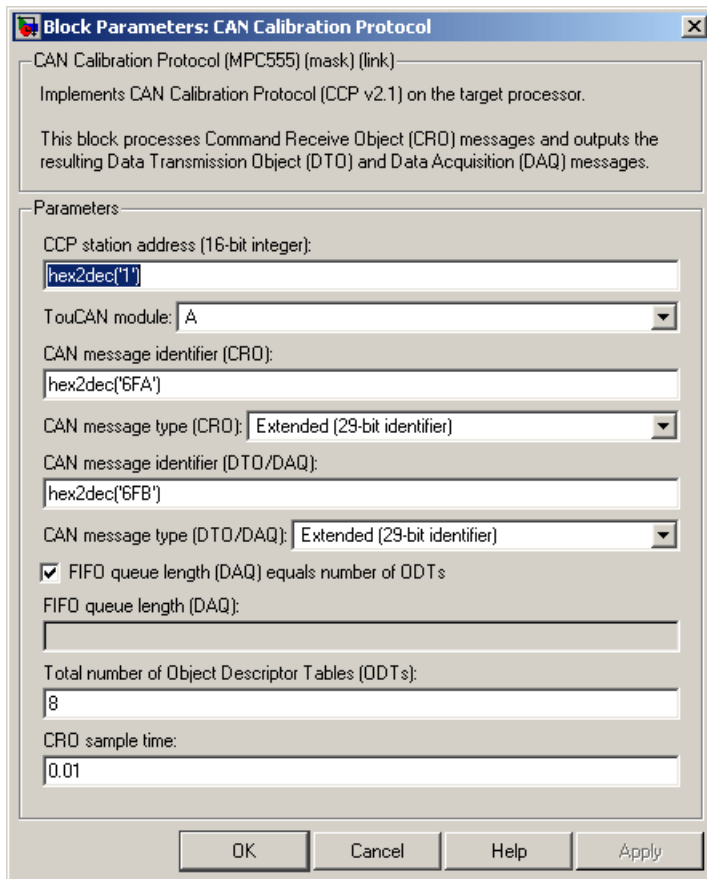
Using DAQ lists for signal monitoring has the following advantages over the polling method:

- There is no need for the host to poll for the data. Network traffic is halved.
- The data is transmitted at the correct update rate for the signal. Therefore there is no unnecessary network traffic generated.
- Data is guaranteed to be consistent. The transmission takes place after the signals have been updated, so there is no risk of interruptions while sampling the signal.

---

**Note** Target for Freescale MPC5xx does not currently support event channel prescalers.

---

# CAN Calibration Protocol (MPC555)

**Dialog Box**



Block Parameters: CAN Calibration Protocol dialog box showing:

CAN Calibration Protocol (MPC555) (mask) (link)

Implements CAN Calibration Protocol (CCP v2.1) on the target processor.

This block processes Command Receive Object (CRO) messages and outputs the resulting Data Transmission Object (DTO) and Data Acquisition (DAQ) messages.

Parameters

CCP station address (16-bit integer):
hex2dec('1')

TouCAN module: A

CAN message identifier (CRO):
hex2dec('6FA')

CAN message type (CRO): Extended (29-bit identifier)

CAN message identifier (DTO/DAQ):
hex2dec('6FB')

CAN message type (DTO/DAQ): Extended (29-bit identifier)

☑ FIFO queue length (DAQ) equals number of ODTs

FIFO queue length (DAQ):

Total number of Object Descriptor Tables (ODTs):
8

CRO sample time:
0.01

OK    Cancel    Help    Apply

**CAN station address (16 bit integer)**
The station address of the target. The station address is interpreted as a `uint16`. It is used to distinguish between different targets. By assigning unique station addresses to targets sharing the same CAN bus, it is possible for a single host to communicate with multiple targets.

**TouCAN module**
Choose A or B.

**CAN message identifier (CRO)**
>    Specify the CAN message identifier for the incoming Command
>    Receive Object (CRO) message you want to process.

**CAN message type (CRO)**
>    The incoming message type. Select either `Standard(11-bit identifier)` or `Extended(29-bit identifier)`.

**CAN message identifier (DTO/DAQ)**
>    The message identifier is the CAN message ID used for Data
>    Transmission Object (DTO) and Data Acquisition (DAQ) message
>    outputs. It is also used for transmitting messages to the host
>    during the software-induced CAN download (soft boot). See
>    "Extended Functionality" on page 5-10.

**CAN message type (DTO/DAQ)**
>    The message type to be transmitted by the DTO and DAQ
>    outputs. Select either `Standard(11-bit identifier)` or
>    `Extended(29-bit identifier)`.

**FIFO queue length (DAQ) equals number of ODTs**
>    Leave this check box selected to automatically set the FIFO queue
>    length equal to the number of Object Descriptor Tables (ODTs)
>    (recommended). Clear the check box to set the length of the FIFO
>    queue manually.

**FIFO queue length (DAQ)**
>    Specify the FIFO queue length manually. This is enabled if you
>    clear the check box to set the queue length automatically.

**Total number of Object Descriptor Tables (ODTs)**
>    The default number of Object Descriptor Tables (ODTs) is 8.
>    These ODTs are shared equally between all available DAQ lists.
>    You can choose a value between 0 and 254, depending on how
>    many signals you want to log simultaneously. You must make
>    sure you allocate at least 1 ODT per DAQ list, or your build will
>    fail. The calibration tool will give an error message if there are too
>    few ODTs for the number of signals you specify for monitoring. Be
>    aware that too many ODTs can make the sample time overrun.

If you choose more than the maximum number of ODTs (254), the build will fail.

A single ODT uses 56 bytes of memory. Using all 254 ODTs would require over 14 KB of memory, a large proportion of the available memory on the target. To conserve memory on the target the default number is low, allowing DAQ list signal monitoring with reduced memory overhead and processing power.

As an example, if you have five different rates in a model, and you are using three rates for DAQ, then this will create three DAQ lists and you must make sure you have at least three ODTs. ODTs are shared equally among DAQ lists, and therefore you will end up with one ODT per DAQ list. With less than three ODTs you get zero ODTs per DAQ list and the behavior is undefined.

Taking this example further, say you have three DAQ lists with one ODT each, and start trying to monitor signals in a calibration tool. If you try to assign too many signals to a particular DAQ list (that is, signals requiring more space than seven bytes (one ODT) in this case), then the calibration tool will report this as an error.

For more information on DAQ lists, see "Data Acquisition (DAQ) List Configuration" on page 2-45.

**CRO sample time**
Sample time at which to check for incoming Command Receive Object (CRO) messages.

## Supported CCP Commands

The following CCP commands are supported by the CAN Calibration Protocol (MPC555) block:

- CONNECT
- DISCONNECT
- DNLOAD

- DNLOAD_6

- EXCHANGE_ID

- GET_CCP_VERSION

- GET_DAQ_SIZE

- GET_S_STATUS

- SET_DAQ_PTR

- SET_MTA

- SET_S_STATUS

- SHORT_UP

- START_STOP

- START_STOP_ALL

- TEST

- UPLOAD

- WRITE_DAQ

## Compatibility with Calibration Packages

The above commands support

- Synchronous signal monitoring via calibration packages that use DAQ lists

- Asynchronous signal monitoring via calibration packages that poll the target

- Asynchronous parameter tuning via CCP memory programming

This CCP implementation has been tested successfully with the Vector-Informatik CANape calibration package running in both DAQ list and polling mode, and with the Accurate Technologies Inc. Vision calibration package running in DAQ list mode. (Note that Accurate

Technologies Inc. Vision does not support the polling mechanism for signal monitoring.)

### Extended Functionality

The CAN Calibration Protocol (MPC555) block also supports the PROGRAM_PREPARE command. This command is an extension of CCP that allows the automatic download of new code into the target memory. This removes the requirement for a manual reset of the processor. On receipt of the PROGRAM_PREPARE command, the target will reboot and begin the CAN download process. This lets you download new application code to RAM or flash memory, or download new boot code to flash memory. See "Downloading Boot or Application Code via CAN Without Manual CPU Reset" on page 2-28.

**Note** The CAN message identifiers of the CCP messages incoming to the target (Command Receive Object (CRO) messages) and the messages outgoing from the target (Data Transmission Object (DTO) or DAQ) are specified in the block mask for the CAN Calibration Protocol (MPC555) block. These message identifiers are used as the CAN identifiers for the download process after a PROGRAM_PREPARE reboot. The type of CAN message used for this PROGRAM_PREPARE download process is always Extended (29-bit identifier).

**Purpose**        Input driver for MIOS 16-bit Parallel Port I/O Submodule (MPIOSM)

**Library**        Target for Freescale MPC5xx/ MPC555 Driver Library/ Modular
                   Input/Output System (MIOS1)

**Description**    The MIOS Digital In block reads the state of selected pins (bits) on the
                   MIOS 16-bit Parallel Port I/O Submodule (MPIOSM) of the MPC555.
                   The **Bits** field specifies a vector of numbers in the range 0..15,
                   corresponding to pins MPIO32B0..MPIO32B15 on the MPIOSM.

```
Digital In
(MPIOSM)
```
MIOS Digital In

The output of the block is a wide vector representing the logic state of
the pins referenced in the **Bits** field. When the signal on a given pin is
a logical 1, the block output element will be equal to 1; otherwise the
block output element will equal zero.

Refer to section 15.13, "MIOS 16-bit Parallel Port I/O Sub module
(MPIOSM)," in the *MPC555 User's Manual* for further information.

---

**Note** You are responsible for ensuring that pin assignments of MIOS
Digital In and MIOS Digital Out blocks in your model do not conflict.
No error checking is performed to detect conditions where the same pin
is referenced by both an input and an output block. If such a condition
occurs, the behavior of the system is undefined.

---

# MIOS Digital In
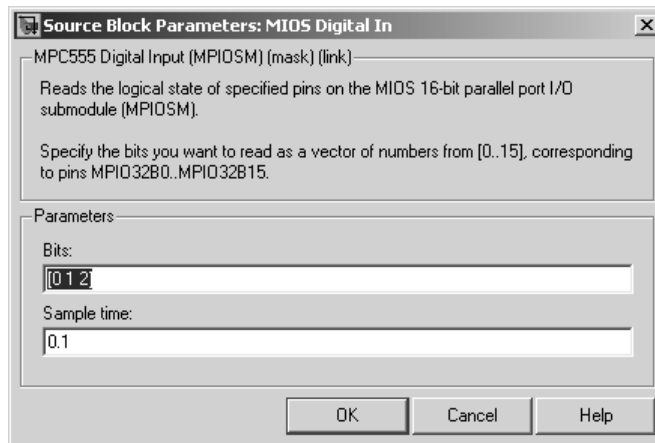
Source Block Parameters: MIOS Digital In

MPC555 Digital Input (MPIOSM) (mask) (link)

Reads the logical state of specified pins on the MIOS 16-bit parallel port I/O submodule (MPIOSM).

Specify the bits you want to read as a vector of numbers from [0..15], corresponding to pins MPIO32B0..MPIO32B15.

Parameters

Bits:

[0 1 2]

Sample time:

0.1

| OK | Cancel | Help |

**Bits**

A vector of numbers in the range 0..15. Each number corresponds to a pin (MPIO32B0..MPIO32B15) on the MPIOSM.

**Sample time**

Sample time of the block.

**Purpose**     Output driver for MIOS 16-bit Parallel Port I/O Submodule (MPIOSM)

**Library**     Target for Freescale MPC5xx/ MPC555 Driver Library/ Modular
Input/Output System (MIOS1)

**Description**     The MIOS Digital Out block sets the state of selected pins (bits) on the
MIOS 16-bit Parallel Port I/O Submodule (MPIOSM) of the MPC555.
The **Bits** field specifies a vector of numbers in the range 0..15,
corresponding to pins MPIO32B0..MPIO32B15 on the MPIOSM.
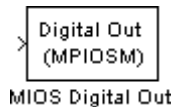
Digital Out
(MPIOSM)

MIOS Digital Out

The input to the block is a wide vector with one signal element per pin.
When the input signal is greater than zero, a logical 1 is written to the
corresponding pin. When the input signal is less than or equal to zero, a
logical zero is written to the corresponding pin.

If you want to write to several digital output pins at the same sample
rate, using a single MIOS Digital Out block with a vector input signal
will result in more efficient code. However, if you want to update
different output pins at different sample rates, you must use a separate
MIOS Digital Out block for each rate.

Refer to section 15.13, "MIOS 16-bit Parallel Port I/O Sub module
(MPIOSM)," in the *MPC555 User's Manual* for further information.

---

**Note** You are responsible for ensuring that pin assignments of MIOS
Digital In and MIOS Digital Out blocks in your model do not conflict.
No error checking is performed to detect conditions where the same pin
is referenced by both an input and an output block. If such a condition
occurs, the behavior of the system is undefined.

---

# MIOS Digital Out

**Dialog Box**

Sink Block Parameters: MIOS Digital Out

─ MPC555 Digital Output (MPIOSM) (mask) (link) ─────────

Sets the logical state of specified pins on the MIOS 16-bit parallel port I/O submodule (MPIOSM). When an element in the input signal is greater than zero a logical one is written to the corresponding pin; otherwise a logical zero is written.

Specify the bits you want to set as a vector of numbers from [0..15], corresponding to pins MPIO32B0..MPIO32B15. The width of this vector must be the same as the width of the input signal.

─ Parameters ─────────────────────────────────

Bits:

[0 1]

Initial output level:

0

Sample time:

-1

[ OK ]   [ Cancel ]   [ Help ]   [ Apply ]

**Bits**

A vector of numbers in the range `0..15`. Each number corresponds to a pin (`MPIO32B0..MPIO32B15`) on the MPIOSM.

**Initial output level**

The value to be placed on the output pins at initialization. This ensures the starting level is always known.

**Sample time**

The sample time of this block.

**Purpose**      Digital output via the MIOS Pulse Width Modulation Submodule
                 (MPWMSM)

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ Modular
                 Input/Output System (MIOS1)
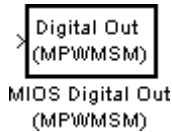
**Description**  The MIOS Digital Out (MPWMSM) block is a device driver that lets
                 you use the MIOS Pulse Width Modulation Submodule (MPWMSM)
                 in *digital output mode*. In digital output mode, the Pulse Width
                 Modulation (PWM) feature of the MPWMSM is turned off. When the
                 input signal is greater than zero, a logical 1 is written to the output pin;
                 otherwise a logical zero is written.

```
  Digital Out
 (MPWMSM)

MIOS Digital Out
 (MPWMSM)
```

Refer to section 15.12, "MIOS Pulse Width Modulation Submodule
(MPWMSM)," in the *MPC555 User's Manual* for further information on
the parameters described below.

**Dialog
Box**
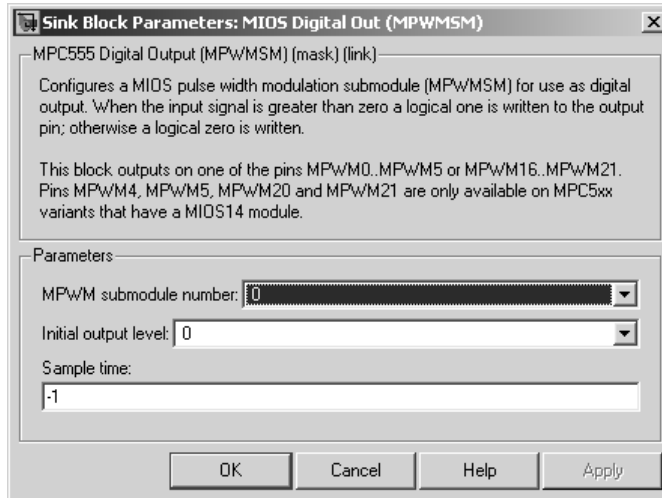
# MIOS Digital Out (MPWMSM)

**MPWM submodule number**
> Select a PWM submodule for output. Note that modules 4, 5, 20 and 21 are for the MPC56x (561-6) only. If you select one of these modules and MPC555 is the processor selected in the Resource Configuration block, then an error will be thrown on updating the model.

**Initial output level**
> The value to be placed on the output pins at initialization. This ensures the starting level is always known.

**Sample time**
> Sample time of the block.

**Invert output polarity**
> Switches the output level for logic one and zero.

**Purpose**          Output driver for MIOS Pulse Width Modulation Submodule
                     (MPWMSM)

**Library**          Target for Freescale MPC5xx/ MPC555 Driver Library/ Modular
                     Input/Output System (MIOS1)
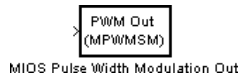
**Description**      The MIOS Pulse Width Modulation Out block is used for Pulse Width
                     Modulation (PWM) output from the MIOS Pulse Width Modulation

```
    PWM Out
   (MPWMSM)
MIOS Pulse Width Modulation Out
```

Submodule (MPWMSM). A PWM signal is a rectangular waveform
whose period is constant but whose duty cycle can be varied, under
control of a modulator signal, between 0% and 100%.

The MIOS Pulse Width Modulation block input signal acts as the
modulator, controlling the duty cycle of the signal on the output pin. The
input signal is multiplied by the period register value, and saturates
if outside 0-1. When the input signal value is 0, the output signal's
duty cycle is 0%. When the input signal value is 1, the output signal's
duty cycle is 100%.

There are two possible methods for calculating the period of the
waveform. You can either control the scaling registers directly, or enter
the desired (ideal) period and the mask will solve for the best values for
the scaling registers.

Refer to section 15.12, "MIOS Pulse Width Modulation Submodule
(MPWMSM)," in the *MPC555 User's Manual* for further information on
the parameters described below.

# MIOS Pulse Width Modulation Out

**Dialog Box**



*Sink Block Parameters: MIOS Pulse Width Modulation Out*

MPC555 Pulse Width Modulation Output (MPWMSM) (mask) (link)

Configures a MIOS pulse width modulation submodule (MPWMSM) to generate a pulse width modulated output signal.

This block outputs on one of the pins MPWM0..MPWM5 or MPWM16..MPWM21. Pins MPWM4, MPWM5, MPWM20 and MPWM21 are only available on MPC5xx variants that have a MIOS14 module.

Parameters

MPWM submodule number: 0

☐ Edit period registers manually

Ideal period (sec):

0.0375

Initial duty cycle (0 <= duty cycle <= 1):

0.5

Clock prescaler field of MPWMSM Status/Control Register:

255

Number of clock ticks per period:

46875

Sample time:

-1

☐ Invert output polarity

☐ Activate transparent mode

☐ Hold output when at debug breakpoint (freeze enable)

OK    Cancel    Help    Apply

**MPWM submodule number**

Select a PWM submodule for output. Note that modules 4, 5, 20 and 21 are for the MPC56x (561-6) only. If you select one of these modules and MPC555 is the processor selected in the Resource Configuration block, then an error will be thrown on updating the model.

**Edit period registers manually**

When this option is selected, the **Clock prescaler field of MPWMSM Status/Control Register** and **Number of clock ticks per period** edit fields are activated. You can then set the PWM period by setting these values.

When this option is not selected, use the **Ideal period (sec)** field to set the PWM period parameters.

**Ideal period (sec)**

Specifies the desired period of the pulse signal. The mask then solves for the clock prescaler and the pulse period.

**Initial duty cycle**

Enter an initial value for the duty cycle (0 <= duty cycle <= 1). This ensures the initial value is always known.

**Clock prescaler field of MPWMSM Status/Control Register**

Divides the counter clock to get the clock signal used to drive the PWM output. Note that the counter clock itself is derived from the MPC555 system clock as configured by the MPC555 Resource Configuration block (see MPC555 Resource Configuration).

**Number of clock ticks per period**

Determines the number of PWM counter ticks per pulse period. Valid values are 1 - 65535.

**Sample time**

Sample time of the block.

**Invert output polarity**

Switches the output level for logic one and zero.

**Activate transparent mode**

Bypasses the register double buffers. When transparent mode is active, a software write to the Next Pulse Width Register is immediately transferred to the Pulse Width Register. When transparent mode is inactive, the updated value only takes effect at the start of the next period.

**Hold output when at debug break point (freeze enable)**
    Stops the PWM counters when a breakpoint is hit during debug
    mode, and holds the current output values.

**Purpose**    Measure pulse width and pulse period measurement via MIOS Double
Action Submodule (MDASM)

**Library**    Target for Freescale MPC5xx/ MPC555 Driver Library/ Modular
Input/Output System (MIOS1)
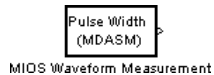
**Description**    Waveform measurement is a feature of the MIOS Double Action

Pulse Width
(MDASM)

MIOS Waveform Measurement

Submodule (MDASM) on the MPC555. The MIOS Waveform
Measurement block currently implements the following features of the
MDASM:

- *Pulse width measurement*: the MIOS Waveform Measurement block
  outputs the time from the leading edge of a pulse to the trailing edge
  of the same pulse.

- *Pulse period measurement*: the MIOS Waveform Measurement block
  outputs the time from the leading edge of a pulse to the next leading
  edge of a pulse.

Note that the minimum and maximum measurable pulse periods and
pulse widths are dependent on the selected clock sources and their
configurations.

You must configure the clock sources via the MPC555 Resource
Configuration object. There are only two clock sources (assigned via the
**Counter bus** parameter) assignable to the 10 MDASM modules. More
than one MDASM can be assigned to a single clock source.

Refer to section 15.11, "MIOS Double Action Submodule (MDASM)
Registers" in the *MPC555 User's Manual* for further information on
the parameters described below.

# MIOS Waveform Measurement

**Dialog Box**
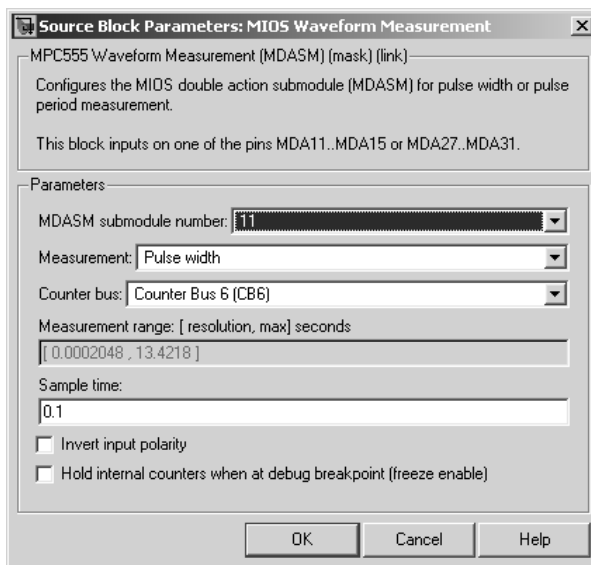


**MDASM submodule number**

Select one of the 10 MIOS Double Action Submodules (MDASM) in the MPC555.

**Measurement**

Select the mode of operation of the block: either pulse width measurement or pulse period measurement.

**Counter bus**

Select one of the two counters that can be used as sources to drive the MDASM module. The counters must be configured via the MPC555 Resource Configuration object. See "MIOS1 Configuration Parameters" on page 5-41.

**Measurement range: [ resolution, max] seconds**

This read only field displays the measurement range of the pulse width or pulse period. The example shown is from the MPC555 real-time I/O demo model `mpc555rt_io`.

**Sample time**

> The period at which Simulink reads the pulse width or period. The measurements are performed in hardware so it is not necessary to set the sample time to suit the expected period of the incoming signal.

**Invert input polarity**

> Changes the sense of the leading edge of the pulse. When **Invert output polarity** is selected, the leading edge is rising. Otherwise, the leading edge is falling.

**Hold internal counters when at debug break point (freeze enable)**

> Stops the clocks of the MDASM module when a breakpoint is hit during debug mode.

# MPC555 Execution Profiling via CAN A

**Purpose**          Provide CAN interface to execution profiling engine via CAN channel A

**Library**          Target for Freescale MPC5xx/ MPC555 Driver Library/ Execution Profiling

**Description**      Provides a CAN interface to the execution profiling engine. On receipt of a start command message, logging of execution profile data is commenced. On completion of a logging run, the recorded data is automatically returned via CAN. You must specify the message identifiers for the start command and the returned data. These identifiers must be compatible with the values used by the host-side part of the execution profiling utility. See also MATLAB command `profile_mpc555`.

`profile_mpc555(connection)` collects and displays execution profiling data from an MPC555 target microcontroller that is running a suitably configured application generated by Target for Freescale MPC5xx. Set `connection` to `'CAN'` in order to collect data via a CAN connection between the target and the host computer. To use the CAN connection, you must have suitable CAN hardware installed on the host computer. This function will test for availability of CanCardX 1 or CanAc2Pci1 and defaults to a bit rate of 500k bits per second. If you need to use a different configuration, you should make a copy of this file (with a different name) and change the configuration data as required. The data collected is unpacked then displayed in a summary HTML report and as MATLAB graphic.

`profdata = profile_mpc555(connection)`

returns the execution profiling data in the format documented by `exprofile_unpack`.

See "The Profiling Command" on page 2-49 for instructions for setting the bit rate automatically or manually.

To configure a model for use with execution profiling, you must perform the following steps:

**1** Make sure the model includes an MPC555 Execution Profiling block that provides an interface between the target-side profiling engine, and the host-side computer from which this command is run.

**2** Make sure the execution profiling option is selected in the Target Specific Options tab of the Real-Time Workshop Options dialog.

For more information see "Execution Profiling" on page 2-48 which includes links to instructions for the example demo mpc555rt_multitasking.mdl.

**Dialog Box**

Block Parameters: MPC555 Execution Profiling via CAN A

MPC555 Execution Profiling via CAN Channel A (mask) (link)

Provides a CAN interface to the execution profiling engine. On receipt of a start command message, logging of execution profile data is commenced. On completion of a logging run, the recorded data is automatically returned via CAN.

You must specify the message identifiers for the start command and the returned data. These identifiers must be compatible with the values used by the host-side part of the execution profiling utility. See also MATLAB command profile_mpc555.

Parameters

Start command CAN message identifier:

hex2dec('1FFFFF00')

Returned data CAN message identifier:

hex2dec('1FFFFF01')

Sample time:

1

| OK | Cancel | Help | Apply |

**Start command CAN message identifier**

Set the identifier of the message to start logging execution profiling data. You should use the default unless you have modified profile_mpc555. This identifier must be compatible with the values used by the host-side part of the execution profiling utility (profile_mpc555).

The utility `profile_mpc555` provides a mechanism for initiating an execution profiling run and for uploading the recorded data to the host machine. To perform this procedure using a CAN connection between host and target, `profile_mpc555` first sends a CAN message that is a command to start an execution profiling run. The CAN identifier for this message must be specified as the same value on the target as on the host. The host-side values are hard-coded in `profile_mpc555`. If you are using an un-modified version of the host side utility, you should use the default value for this CAN message identifier. These are visible to help you avoid using the same identifier for other tasks.

**Returned data CAN message identifier**
Set the message identifier for the returned data. As with the message identifier for the start command, the value specified here must be the same as the hard-coded value in profile_mpc555.

**Sample time**
The sample time of the block. The faster the sample time of the block, the faster data will be uploaded at the end of the execution profiling run. You may want to run this block slower than the fastest rate in the system because the execution profiling itself imposes some loading on the processor. You can minimize this extra loading by not running it at the fastest rate.

**Purpose**      Provide serial interface to execution profiling engine

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ Execution Profiling

**Description**

Execution Profiling
via Serial

MPC555 Execution Profiling
via SCI1
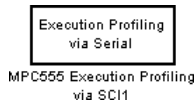
Provides a CAN interface to the execution profiling engine. On receipt of a start command message, logging of execution profile data is commenced. On completion of a logging run, the recorded data is automatically returned via serial. See also MATLAB command `profile_c166`.

`profile_mpc555(connection)` collects and displays execution profiling data from an MPC555 target microcontroller that is running a suitably configured application generated by Target for Freescale MPC5xx. The `connection` may be set to 'serial' in order to collect data via a serial connection between the target and the host computer.

The data collected is unpacked then displayed in a summary HTML report and as MATLAB graphic.

`profdata = profile_mpc555(connection)`

returns the execution profiling data in the format documented by `exprofile_unpack`.

See "The Profiling Command" on page 2-49 for instructions for setting the bit rate automatically or manually.
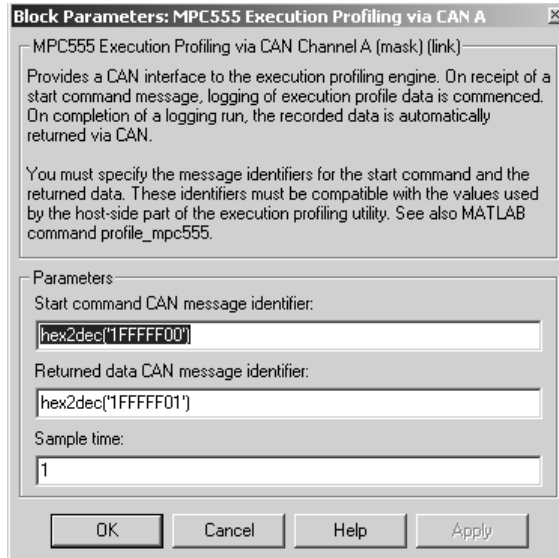
To configure a model for use with execution profiling, you must perform the following steps:

**1** Make sure the model includes an MPC555 Execution Profiling block that provides an interface between the target-side profiling engine, and the host-side computer from which this command is run.

**2** Make sure the execution profiling option is selected in the Target Specific Options tab of the Real-Time Workshop Options dialog.

For more information see "Execution Profiling" on page 2-48 which includes instructions for the example demo `mpc555rt_multitasking.mdl`.
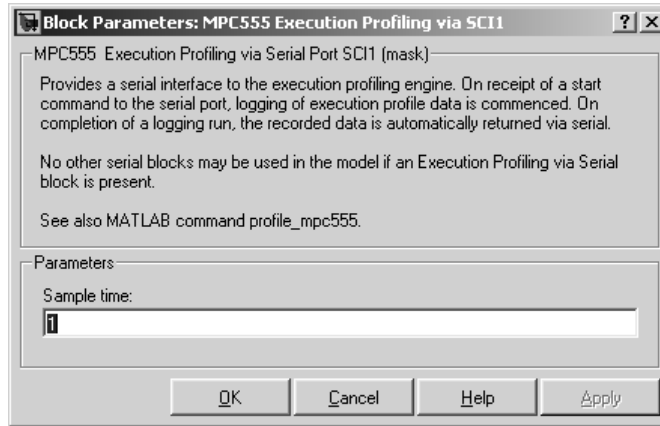
**Dialog Box**



**Sample time**

    The sample time of the block. The faster the sample time of the block, the faster data will be uploaded at the end of the execution profiling run. You may want to run this block slower than the fastest rate in the system because the execution profiling itself imposes some loading on the processor. You can minimize this extra loading by not running it at the fastest rate.

**Purpose**      Support device configuration for MPC5xx CPU and MIOS, QADC, and
TouCAN submodules

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library

**Description**  The MPC555 Resource Configuration block differs in function and
behavior from conventional blocks. Therefore, we refer to this block as
the MPC555 Resource Configuration *object*.

MPC555
Resource
Configuration

The MPC555 Resource Configuration object maintains configuration
settings that apply to the MPC555 CPU and its MIOS, QADC, and
TouCAN subsystems. Although the MPC555 Resource Configuration
object resembles a conventional block in appearance, it is not connected
to other blocks via input or output ports. This is because the purpose of
the MPC555 Resource Configuration object is to provide information to
other blocks in the model. MPC555 device driver blocks register their
presence with the MPC555 Resource Configuration object when they
are added to a model or subsystem; they can then query the MPC555
Resource Configuration object for required information.

To install a MPC555 Resource Configuration object in a model or
subsystem, open the top-level Target for Freescale MPC5xx library and
select the MPC555 Resource Configuration icon. Then drag and drop it
into your model or subsystem, like a conventional block.

Having installed a MPC555 Resource Configuration object into your
model or subsystem, you can then select and edit configuration settings
in the MPC555 Resource Configuration window. See "Using the MPC555
Resource Configuration Window" on page 5-34 for further information.

# MPC555 Resource Configuration

---

**Note** Any model or subsystem using device driver blocks from the Target for Freescale MPC5xx library *must* contain an MPC555 Resource Configuration object. You should place an MPC555 Resource Configuration object at the top level system for which you are going to generate code. If your whole model is going to run on the target processor, put the MPC555 Resource Configuration object at the root level of the model. If you are going to generate code from separate subsystems (to run specific subsystems on the target), place an MPC555 Resource Configuration object at the top level of each subsystem. You should not have more than one MPC555 Resource Configuration object in the same branch of the model hierarchy. Errors will result if these conditions are not met.

---

When the MPC555 Resource Configuration block is placed into a model, it modifies the `preloadfcn` callback of the model. If you wish to add a command to the `preloadfcn` callback of a model that already has an MPC555 Resource Configuration block, do not remove the commands that are already installed.

Instead, copy the installed `preloadfcn` callback and append your commands. Then set the `preloadfcn` to the merged command. If you corrupt the `preloadfcn`, you can retrieve the command from any model that has an MPC555 Resource Configuration block, as the `preloadfcn` will be the same for all models. You can retrieve the `preloadfcn` with the following command:

```
plf = get_param(bdroot,'preloadfcn')
```

### Types of Configurations

A *configuration* is a collection of parameter values affecting the operation of a group of device driver blocks in one of the Target for Freescale MPC5xx libraries, such as the MIOS1, QADC64 or TouCAN libraries. The MPC555 Resource Configuration object currently supports the following types of configurations:

- "System Configuration Parameters" on page 5-36: MPC555 clocks and other CPU-related parameters.

- "QADC64 Configuration Parameters" on page 5-38: parameters related to the Queued Analog-to-Digital Converter module (QADC).

- "QADC64E Configuration Parameters" on page 5-40: parameters related to the QADC for the MPC565.

- "MIOS1 Configuration Parameters" on page 5-41: parameters related to the Modular Input/Output System (MIOS).

- "TouCAN Configuration Parameters" on page 5-43: parameters related to the CAN 2.0B Controller Module (TouCAN).

- "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46: parameters related to the Time Processor Unit module.

- "Serial Communications Interface (SCI) Configuration Parameters" on page 5-50: parameters related to the Serial Communications Interface.

### Active and Inactive Configurations

An *active* configuration is a configuration associated with blocks of the model or subsystem in which the MPC555 Resource Configuration object is installed. There is always an active MPC555 configuration. For any other configuration type (e.g., QADC, MIOS, or TouCAN), there is at most one active configuration. Such configurations are only active when relevant device driver blocks are present in the model or subsystem.

Consider this model, which contains a MPC555 Resource Configuration object but no MPC555 device driver blocks.

# MPC555 Resource Configuration



Pulse
Generator

Out1

MPC555
Resource
Configuration
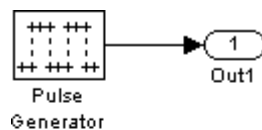
This model has only one active configuration, for the MPC555 itself, as
shown in the MPC555 Resource Configuration window.
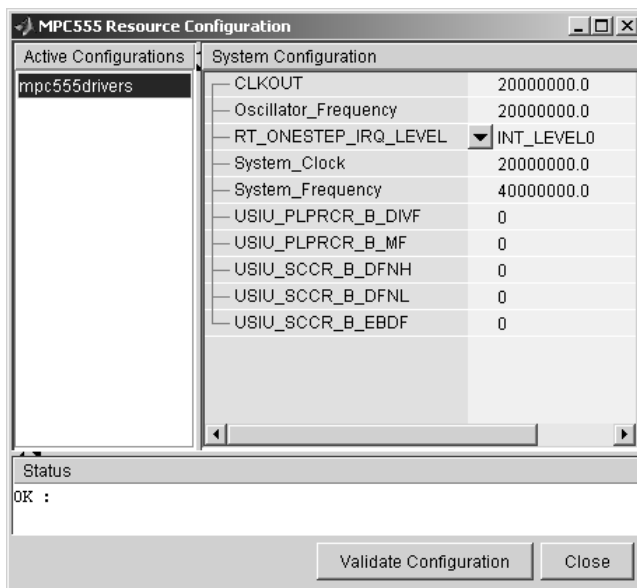


When a device driver block is added to the model, an appropriate
configuration is created and activated. The following figure shows an
MIOS Digital Out block added to the model.

The addition of the MIOS Digital Out block causes an MIOS configuration to be added to the list of active configurations, as shown in this figure.
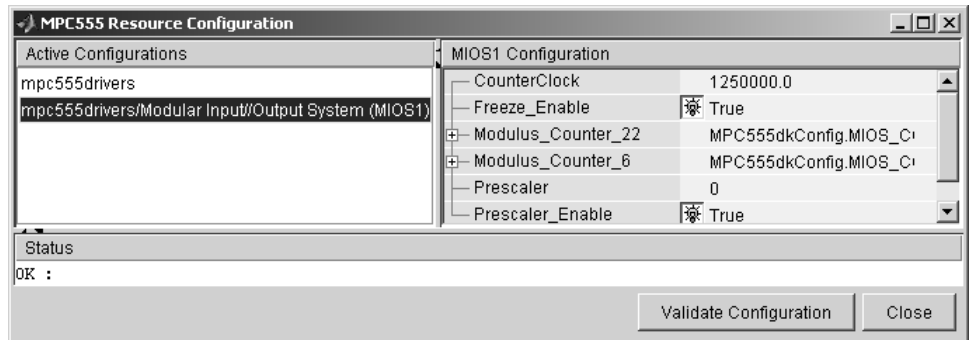


A configuration remains active until all blocks associated with it are removed from the model or subsystem. At that point, the configuration is in an *inactive* state. Inactive configurations are not shown in the MPC555 Resource Configuration window. You can reactivate a configuration by simply adding an appropriate block into the model.

# MPC555 Resource Configuration

---

**Note** When using device driver blocks from the Target for Freescale MPC5xx libraries in conjunction with the MPC555 Resource Configuration block, do not disable or break library links on the driver blocks. If library links are disabled or broken, the MPC555 Resource Configuration block will operate incorrectly.

---

### Using the MPC555 Resource Configuration Window

To open the **MPC555 Resource Configuration** window, install a MPC555 Resource Configuration object in your model or subsystem, and double-click on the MPC555 Resource Configuration icon. The **MPC555 Resource Configuration** window then opens.
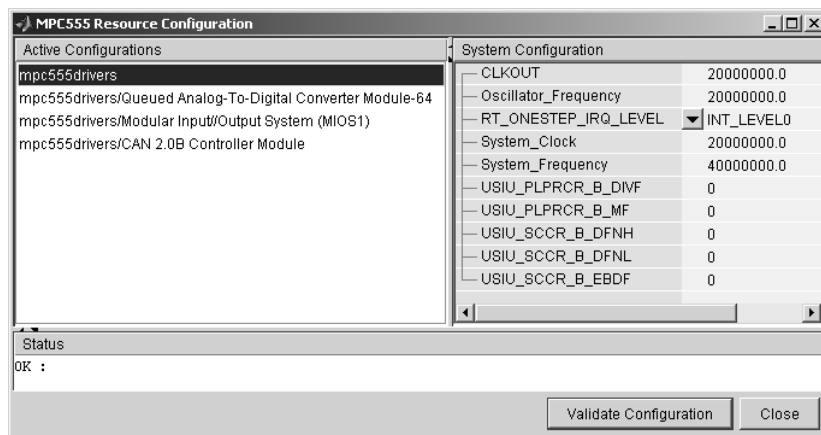


**MPC555 Resource Configuration Window**

This figure shows the MPC555 Resource Configuration window for a model that has active configurations for MPC555, MIOS1, QADC, and TouCAN.

The MPC555 Resource Configuration window consists of the following elements:

- **Active Configurations** panel: This panel displays a list of currently active configurations. To edit a configuration, click on its entry in the list. The parameters for the selected configuration then appear in the **System configuration** panel.

    To link back to the library associated with an active configuration, right-click on its entry in the list. From the pop-up menu that appears, select **Go to library**.

    To see documentation associated with an active configuration, right-click on its entry in the list. From the popup menu that appears, select **Help**.

- **System configuration** panel: This panel lets you edit the parameters of the selected configuration. The parameters of each configuration type are detailed in "MPC555 Resource Configuration Window Parameters" on page 5-36.

    **Note** There is no **Apply** or **Undo** functionality in the **System configuration** panel. All parameter changes are applied immediately.

- **Status** panel: The **Status** panel displays error messages that may arise if resource allocation conflicts are detected in the configuration.

- **Validate Configuration** button: After you edit a configuration, you should always click the **Validate Configuration** button to check for resource allocation conflicts. For example, if both TouCAN modules A and B are assigned to interrupt level IRQ 1, the **Validate Configuration** process will detect the conflict and display a warning in the **Status** panel.

    Note that the **Validate Configuration** button does not validate the entire model; it only checks for resource allocation conflicts related to the selected configuration. To detect problems related to the model as a whole, select **Update diagram** (**Ctrl+D**) from the Simulink Edit menu.

• **Close** button: Dismisses the window.

**MPC555 Resource Configuration Window Parameters**

The sections below describe the parameters for each type of configuration in the MPC555 Resource Configuration window. The default parameter settings are optimal for most purposes. If you want to change the settings, we suggest you read the sections of the *MPC555 User's Manual* referenced below. You can find this document at the following URL:

http://www.freescale.com/files/microcontrollers/doc/user_guide/MPC555UM.pdf

### System Configuration Parameters
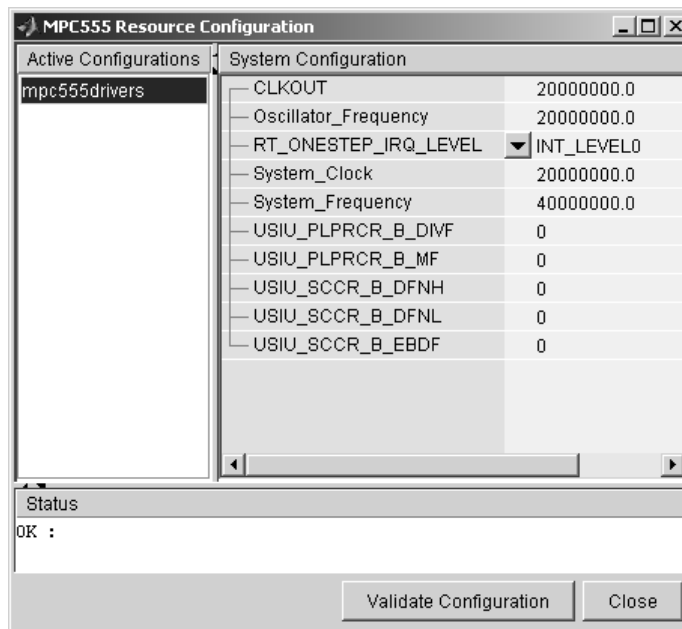


**RT_ONESTEP_IRQ_LEVEL**
> The rt_OneStep function is the basic execution driver of all programs generated by Target for Freescale MPC5xx. rt_OneStep is installed as a timer interrupt service routine; it sequences calls

to the *model*_step function. The **RT_ONESTEP_IRQ_LEVEL** parameter lets you associate rt_OneStep with any of the available IRQ levels (0..7). Do not select Interrupts Disabled, or the model will not work.

See the "Data Structures and Program Execution" section in the Real-Time Workshop Embedded Coder documentation for a detailed description of the rt_OneStep function.

### System Clock and Related Parameters

The parameters Oscillator_Frequency, USIU_PLPRCR_B_DIVF, USIU_PLPRCR_B_MF, USIU_SCCR_B_DFNH, USIU_SCCR_B_DFNL, USIU_SCCR_B_EBDF in the MPC555 group control the speed of the main clocks in the MPC555. Refer to section 8, "Clocks and Power Control," in the MPC555 User's Manual for information on these settings.

Some pre-defined configurations may be applied by inserting the block Switch Target Hardware Configuration into your model. This block is found in the Utilities sublibrary of the MPC555 Driver Library, see Switch Target Configuration. Insert this block in your model, then double-click on the block to choose a configuration from the available list. When one of the pre-defined configurations is selected, the appropriate settings will be applied automatically.

Note Target for Freescale MPC5xx only supports an Oscillator_Frequency of 4 MHz or 20 MHz; the setting of this parameter must correspond to the crystal frequency on your target hardware.

You might want to change these parameters in order to allow a different system clock value to be used; a faster system clock will increase the processing performance, as well as increasing power consumption. With default settings, the default values result in a system clock of 20 MHz for the MPC555. To gain additional processing power it may be desirable to increase the system clock.

For the MPC555, the system clock may be increased up to 40 MHz. The exact settings that are required to achieve a desired system clock value may be calculated using the formulae provided in the MPC555 User Documentation. For example

System clock = `Oscillator_Frequency` * (MF+1)/(DIVF +1)

— where MF is the multiplying factor `USIU_PLPRCR_B_MF` and DIVF is the dividing factor `USIU_PLPRCR_B_DIVF`.

For example, if your hardware uses an external oscillator frequency of 20 MHz (e.g. as used on a phyCORE-MPC555 board), then changing the value of `USIU_PLPRCR_B_MF` from 0 to 1 will increase the system clock from 20 to 40 MHz. For different external oscillator frequencies or different processor variants you should consult the user documentation for your hardware.

### QADC64 Configuration Parameters
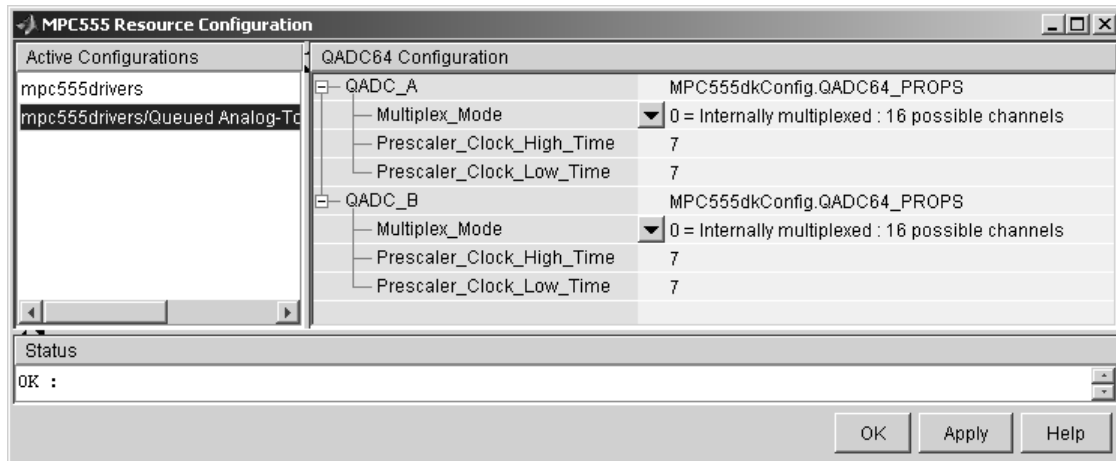


The Queued Analog-To-Digital Converter Module 64 (QADC64) Configuration parameters configure the QADC64 operational mode and supports the blocks in the QADC sublibrary.

The QADC64 performs 10 bit analog to digital conversion on an input signal. Currently the blocks in this library support only the *continuous scan* mode of operation. In continuous scan mode, the QADC64 is set to run, and then continuously acquires data into its result buffer. Input is double buffered, so the model can read the result buffer at any time to get the latest available signal data.

The MPC555 has two QADC modules, QADC_A and QADC_B. You can program these individually. By default each QADC module has 16 input channels. By attaching an external multiplexer to three of the analog input pins, you can increase the number of possible channels to 41. These pins become outputs from the processor and can act as inputs to an analog multiplexer. The **Multiplex Mode** parameter determines whether the QADC64 operates in internally or externally multiplexed mode.

Refer to section 13, "Queued Analog-to-Digital Converter Module-64," in the *MPC555 User's Manual* for detailed information about the QADC64.

In general, you should not need to change any of the settings of the parameters described below from their defaults. The other parameters are advanced settings. Refer to section 13, "Queued Analog-to-Digital Converter Module-64," in the *MPC555 User's Manual* for information on these settings.

**Multiplex Mode**

Configures the QADC64 for internally or externally multiplexed mode by setting the MUX bit. The MUX bit determines the interpretation of the channel numbers and forces the MA[2:0] pins to be outputs. Valid settings are

- `0 = Internally multiplexed : 16 possible channels`

- `1 = Externally multiplexed : 41 possible channels`

**Prescaler Clock High Time**

Prescaler clock high (PSH) time. The default is 7. The PSH field selects the QCLK high time in the prescaler. PSH value plus 1 represents the high time in IMB clocks.

# MPC555 Resource Configuration

**Prescaler Clock Low Time**

Prescaler clock low (PSL) time. The default is 7. The PSL field selects the QCLK low time in the prescaler. PSL value plus 1 represents the low time in IMB clocks.

### QADC64E Configuration Parameters
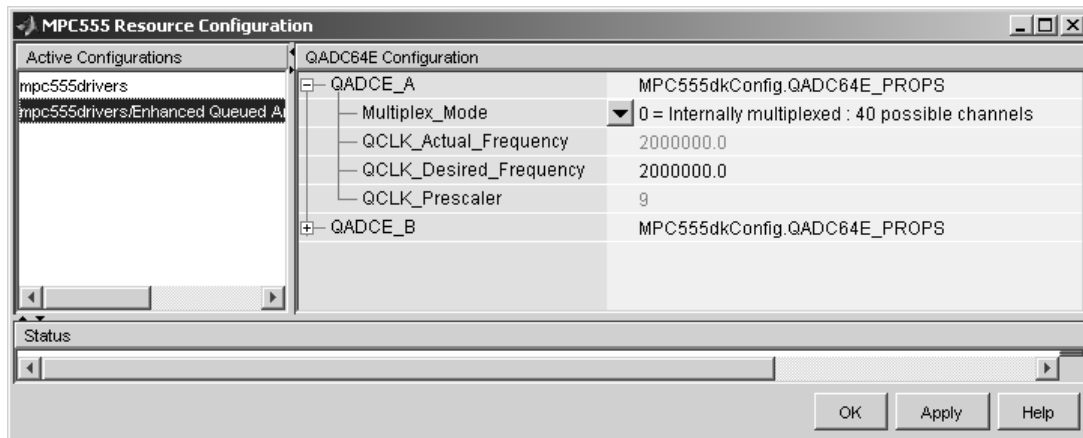


The Enhanced QADC functions are for MPC56x processors – you will see an error message if you try to configure these for an MPC555. Use QADC blocks for an MPC555; for an MPC56x set your target processor accordingly in the Target Preferences and then you can use the QADCE blocks.

The Enhanced Queued Analog-To-Digital Converter Module 64 (QADC64E) Configuration parameters configure the QADC64E operational mode and supports the blocks in the Enhanced QADC sublibrary.

**Multiplex Mode**

Configures the QADC64 for internally or externally multiplexed mode by setting the MUX bit. The MUX bit determines the interpretation of the channel numbers and forces the MA[2:0] pins to be outputs. Valid settings are

- 0 = Internally multiplexed :  40 possible channels

- 1 = Externally multiplexed :  65 possible channels

**QCLK_Desired_Frequency**

Set the Q clock frequency you want here.  The
QCLK_Actual_Frequency field displays the true value
achieved. QCLK_Actual_Frequency and QCLK_Prescalar are
read only fields for information.

### MIOS1 Configuration Parameters
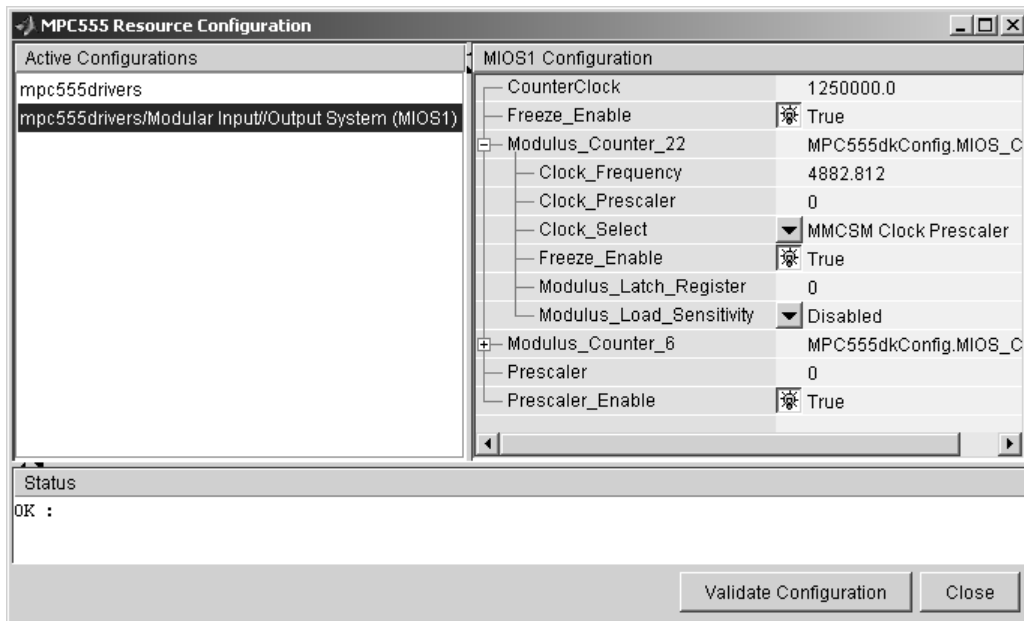


**CounterClock**

The MIOS counter clock is generated by the MIOS counter
prescaler submodule. The MIOS counter clock drives the other
MIOS1 submodules. The value shown for the counter clock is
calculated automatically as the system clock frequency divided
by the prescaler value.

# MPC555 Resource Configuration

**Freeze Enable**

This allows all counters on the MIOS1 to be frozen when the processor is stopped in debug mode. Note that this is in addition to the **Freeze Enable** setting for individual submodules on the MIOS1. To allow the counters on a particular submodule to be stopped, select Freeze enable here, and select **Hold output when at debug break point (freeze enable)** in the block parameters associated with the submodule (e.g., MIOS Pulse Width Modulation block or MIOS Waveform Measurement block).

**Modulus Counter 6 and 22**

These two counters provide reference clocks to submodules such as the MIOS Pulse Width Modulation Submodule and the MIOS Double Action Submodule (Frequency / Period measurement) subsystems. If you change the **Clock Select** to anything other than `MMCSM Clock Prescaler`, the MIOS Pulse Width Modulation and MIOS Waveform Measurement blocks will not work as expected. To change the clock frequency and hence the available resolution of pulse width modulation and waveform measurement, change the **Clock Prescaler** to a value between 0 and 255.

Refer to section 15.10, "MIOS Modulus Counter Submodule (MMCSM)," in the *MPC555 User's Manual* for information on these settings.

## TouCAN Configuration Parameters
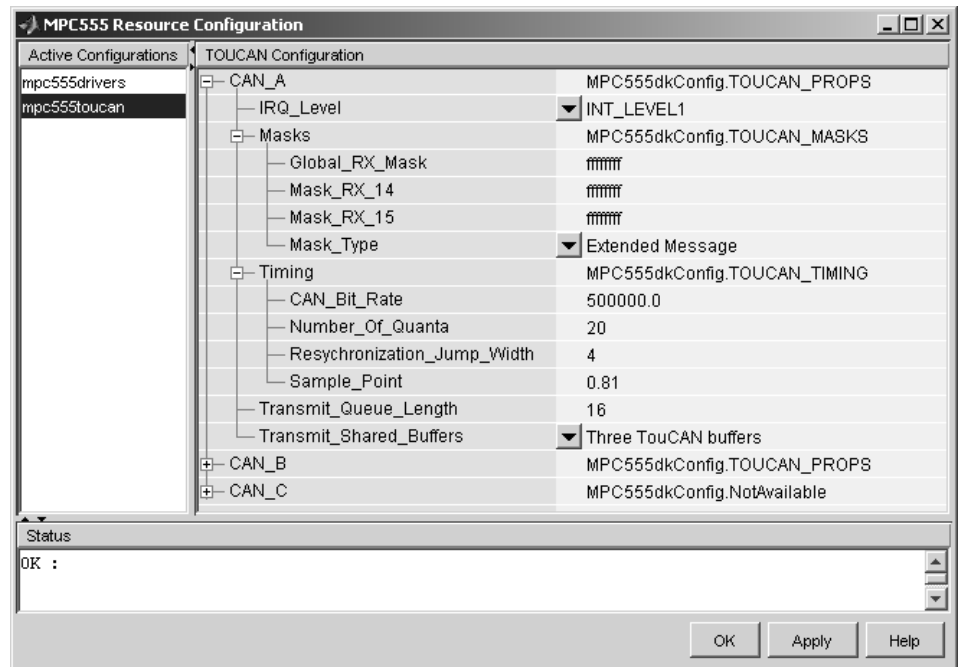


The parameters listed below are the same for TouCAN modules A and B
(and C, for MPC56x). Consult Section 16 of the *MPC555 User's Manual*
before editing the TouCAN configuration parameter defaults.

### IRQ Level

The transmit queue for each TouCAN module requires a processor
interrupt to run. Select an interrupt level (0-31) that is not used
by any other device. Use the **Apply** button to make sure you
do not select an interrupt level that is already in use. Do not
disable interrupts: this will stop the TouCAN Transmit block
from working correctly.

**Mask Configuration Parameters**

**Global RX Mask**

Buffers 0-13 use this mask. Setting a bit to 0 in the mask causes the corresponding bits in the incoming message's identifier to be masked out (i.e., ignored).

0 – Corresponding bit in the incoming message's identifier is "don't care"

1 – Corresponding bit in the incoming message's identifier is checked against the identifier specified in the TouCAN Receive block associated with this buffer.

**Mask RX 14**

Same as **Global RX Mask**, but the mask applies only to buffer 14.

**Mask RX 15**

Same as **Global RX Mask**, but the mask applies only to buffer 15.

**Mask Type**

Specify whether the buffer masks are Standard or Extended frame IDs. If you want to receive Extended Frames in your model, you should set the **Mask Type** to **Extended Message**. The mask type option tells the compiler how to map the bits specified in the mask options to the bits in the hardware. The decision as to whether a message is a Standard or Extended frame is defined on a per message buffer basis.

**Timing Configuration Parameters**

**CAN Bit Rate**

Enter the desired bit rate. The default bit rate is `500000.0`.

**Number of Quanta**

The number of TouCAN clock ticks per message bit.

**Resynchronization Jump Width**

The maximum number of clock ticks that the TouCAN device can resynchronize over when it detects that it is losing message synchronization.

### Sample Point

The point in the message where the TouCAN tries to sample the value of the message bit, between 0 and 1.

Slew Rate

You cannot select the slew rate for the TouCAN modules. By default, the slow slew rate is selected for the TouCAN modules. This results in a slew rate of 50ns for TouCAN C, and 200ns for the other modules.

## Transmission Configuration Parameters

### Transmit Queue Length

Length (number of messages) of the transmit queue. The transmit queue holds messages that are waiting to be transmitted. An increase in performance can be achieved by reducing the queue length. However, if the queue's length is too small it may become full, causing messages to be lost.

### Transmit Shared Buffers

Choose either `Single TouCAN Buffer` or `Three TouCAN Buffers`. This parameter is used in conjunction with all TouCAN Transmit blocks in the model for this TouCAN module that are operating in `Queued transmission with shared buffer` mode. If you select `Single TouCAN Buffer`, then all messages that are queued will be transmitted via a single hardware buffer; in this case, it is possible that a low priority message in the transmit buffer will block higher priority messages that are in the queue. To avoid this problem, use the option `Three TouCAN Buffers`. When three buffers are used, the driver ensures that the message entered into arbitration to be transmitted via the CAN bus is always the highest priority message available; furthermore in this mode the TouCAN module is able to transmit messages continuously by re-loading hardware buffers that become empty while another buffer is active transmitting.

**Time Processor Unit (TPU3) Configuration Parameters**
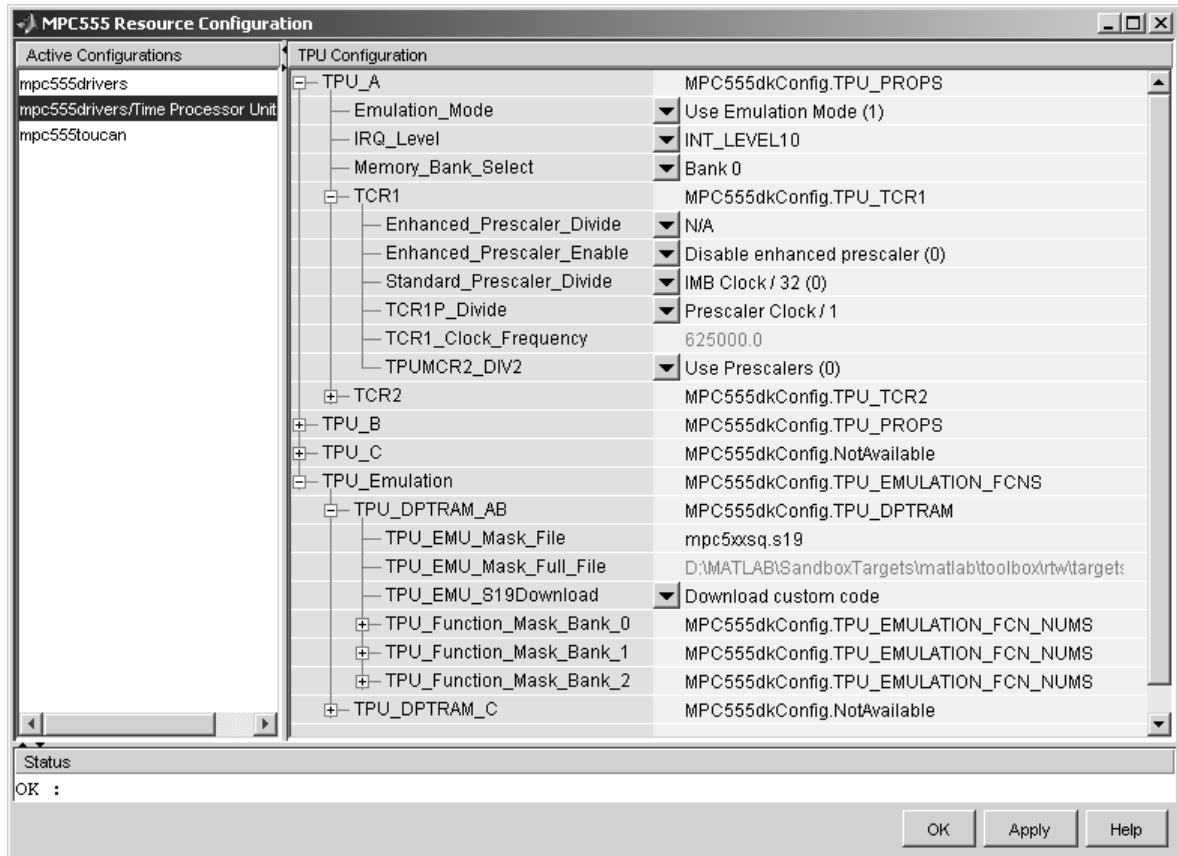


**Emulation_Mode**

The default is to Use ROM TPU Functions (0). Select Use Emulation Mode (1) to use downloaded TPU functions in DPTRAM. Use the parameters under TPU_Emulation to configure downloads for emulation mode. For an example see the demo model mpc555rt_tpu_emu. Note that CCP Program_Prepare

downloads will fail because DPTRAM_AB contains TPU microcode for emulation mode.

**IRQ_Level**

This enables TPU interrupts. The default is disabled. If your model contains any TPU3 Programmable Time Accumulator blocks, you will need to choose an interrupt level.

**Memory_Bank_Select**

Select Bank 0, 1 or 2. If you select an invalid memory bank for the TPU module (e.g. Bank 2 for TPU C) you will see an error message when you click **Apply**. This must match the selection for the parameters TPU_Function_Mask_Bank_0 (also Bank_1, Bank_2). The TCR1 and TCR2 timebases are configurable for TPU Channels A, B and C.

**TCR1**

The parameters under the TCR1 tree allow you full control to specify the clock speed of the TCR1 timebase. Consult Section 17 of the *MPC555 User's Manual* before editing the TPU configuration parameter defaults. The parameters listed below are the same for TPU modules A, B and C.

**Enhanced_Prescaler_Divide**

If you choose to use the Enhanced_Prescaler_Divide, then you can choose to divide the IMB clock down by either 2, 4, 6, 8, ... , 60, 62, 64.

**Enhanced_Prescaler_Enable**

Here you can choose whether you use the Standard Prescaler (set by Standard_Prescaler_Divide) or the Enhanced Prescaler (set by Enhanced_Prescaler_Divide) to derive the Prescaler Clock.

**Standard_Prescaler_Divide**

If you choose to use the Standard_Prescaler_Divide then you can choose to divide the IMB clock down by either 32 or 4.

**TCR1P_Divide**

Whichever type of prescaler you choose (standard or extended), there is a further prescaler that is applied to the clock.

TCR1P_Divide divides the Prescaler Clock by 1, 2, 4, or 8. The resulting clock is the TCR1 timebase.

**TCR1_Clock_Frequency**

Read-only field displaying calculated TCR1 clock frequency.

**TPUMCR2_DIV2**

TPUMCR2_DIV2 (the last setting under the tree) allows you to choose to use a set of prescalers to divide the IMB clock down further (`Use Prescalers (0)`), or to just divide the IMB clock by two (`IMB Clock / 2 (1)`). If you choose the divide by two option then none of the other settings are applicable and are marked `N/A`. Note this is the last setting purely because the parameters are laid out in alphabetical order.

**TCR2**

The parameters under the TCR2 tree for specifying the clock speed of the TCR2 timebase are the same for TPU modules A, B and C. You can configure the TCR2 to use an external clock.

**TCR2P_Divide**

You can choose to divide the TCR2 prescaler clock down by either 1, 2, 4, or 8.

**TCR2_Clock_Frequency**

Read-only field displaying calculated TCR2 clock frequency when using the gated IMB clock. This field displays zero when using an external clock, as it cannot predict an external clock signal.

**TCR2_Counter_Clock_Source**

Select from `Rise transition T2CLK`, `Gated IMB clock`, `Fall transition T2CLK`, or `Rise & fall transition T2CLK`.

The `Gated IMB clock` setting uses the T2CLK pin to gate the internal clock as a source for TCR2 (a logical AND between the input on the T2CLK pin and the IMB clock is performed).

The other settings allow TCR2 to be clocked from the selected edge of an external clock signal applied to the T2CLK pin.

**TCR2_PSCK2**

See the *MPC555 User's Manual* for the effects of setting the TCR2_PSCK2 bit. The default, `Divide by 1`, leaves the **TCR2P_Divide** setting the only prescaler applied to the clock (if using an external clock). If using the gated IMB clock there is always an additional implicit divide by 8.

**TPU_Emulation**

Use these settings to configure downloads for TPU emulation mode.

**TPU_DPTRAM_AB and TPU_DPTRAM_C**

Use the settings under these two parameters to configure emulation mode for TPU modules A and B **(TPU_DPTRAM_AB)** and/or TPU modules C (**TPU_DPTRAM_C**). The parameters listed below are the same for TPU modules A, B and C.

**TPU_EMU_Mask_File**

Enter the name of the S19 file containing the TPU functions to be downloaded. The specified file must be either in the current working directory OR the MATLAB path if an absolute path is not explicitly specified. Note the file name will not be accepted unless **TPU_EMU_S19Download** is set to `Download custom code`. This parameter retains a memory of the last file specified.

The S19 file must be produced from an `.asc` microcode mask file and a TPU microcode assembler. The TPU function names and TPU function numbers are specified in the `.asc` file. Make sure you enter the same TPU function names and numbers in the **TPU_Function_Mask_Bank** parameters.

**TPU_EMU_Mask_Full_File**

Read only field displaying the full path to the download file. Check this to ensure the correct file is shown.

**TPU_EMU_S19Download**

Select `Download custom code` to download to DPTRAM for emulation mode. The default is `No code download`.

**TPU_Function_Mask_Bank_0 (also Bank_1, Bank_2)**

Use the parameters under here to specify which TPU Function Numbers correspond to which TPU functions. For example, typing PTA for TPU_Function_D will specify that the PTA function is configured as TPU function number 13. If you enter a string that is not a valid TPU function name, when you click **Apply** an error message appears in the status field, followed by a list of possible TPU Function Names and their corresponding full function names. Names must be exact including case. The specified TPU function names and numbers must correspond to those specified in the **TPU_EMU_Mask_File**.

### Serial Communications Interface (SCI) Configuration Parameters
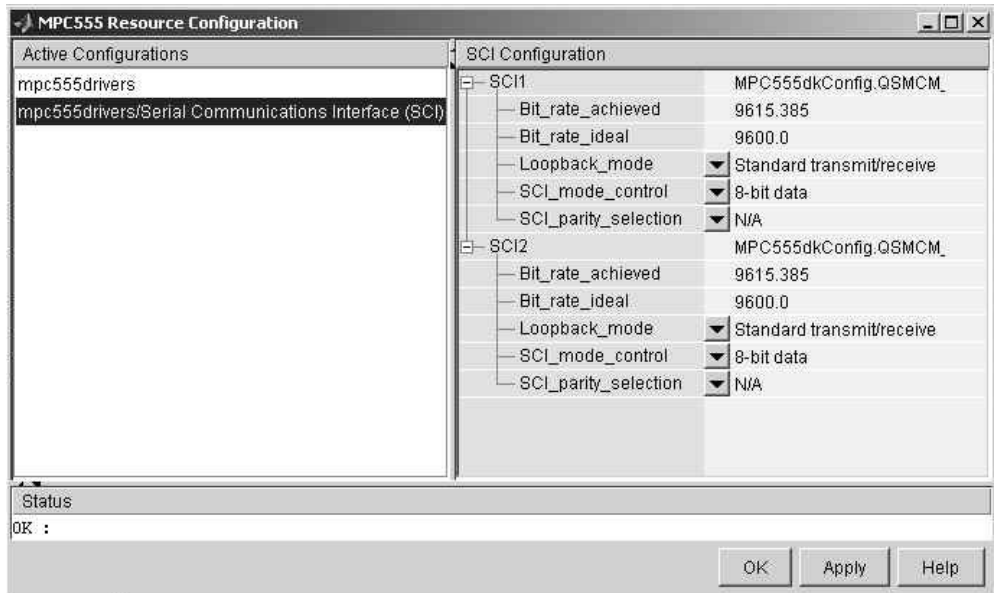
**Bit_rate_achieved**

>   This read-only field shows the achieved serial interface bit rate. In general this value differs slightly from the requested bit rate, but is the closest value that can be achieved by setting allowed values in the MPC555 registers SCC1R0 and SCC2R0 for QSMCM submodules SCI1 and SCI2 respectively.

**Bit_rate_ideal**

>   Enter the desired bit rate for serial communications in this field. Appropriate register settings will be calculated automatically. You can check the actual bit rate in the **Bit_rate_achieved** field.

**Loopback_mode_enable**

>   Select either `Standard transmit/receive` or `Loopback mode enabled`. The loopback mode may be useful for test purposes where the serial interface is required to receive data that it transmitted itself.

**SCI_mode_control**

>   Select the desired combination of word length and parity/no parity.

**Parity_selection**

>   If parity is enabled, you must select `Odd parity` or `Even parity`.

# QADC Analog In

**Purpose**          Input driver enables use of Queued Analog-Digital Converter (QADC64) in continuous scan mode

**Library**          Target for Freescale MPC5xx/ MPC555 Driver Library/ Queued Analog-To-Digital Converter Module-64
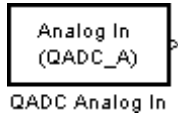
**Description**      The QADC Analog In block sets the QADC64 into continuous scan mode. It then samples the specified channels at the specified rate. In continuous scan mode, the analog-to-digital converter is scanned as fast as possible, at a rate much faster than the sample rate of the model. Using continuous scan mode ensures that your application will obtain the latest signal value.

Analog In
(QADC_A)

QADC Analog In

The MPC555 has two QADC modules, A and B. You can program these individually. You can place only one instance of the QADC Analog In block per module in your model or subsystem.

**Dialog Box**

Source Block Parameters: QADC Analog In

┌─MPC555 Analog Input (QADC64, Continuous-scan) (mask) (link)──────────

Analog input using one of the Queued Analog Digital Converter (QADC64) modules. The module is operated in continuous-scan mode.

Specifying channel numbers:
- In non-multiplexed mode, specify the channels as a vector of numbers from [0..3, 48..59], correponding to pins AN0..AN3 and AN48..AN59.
- In multiplexed mode see the table in the documentation.

┌─Parameters──────────────────────────────────────────────────────

QADC module: A

Channels:
[0 1 2 3 48 49 50 51]

Justification: Right-justified (unsigned)

Sample time:
0.1

OK        Cancel        Help

**QADC module**

> Select module A or B.

**Channels**

> A vector of numbers representing channels to be scanned. See "Channel Number Selection" on page 5-53 below.

**Justification**

> Converted data is read from the 10-bit wide QADC64 result word table into a 16-bit word. Data from the result word table can be accessed in three different formats. The **Justification** menu selects from the following formats:
>
> - Right-justified (unsigned): with zeros in the higher order unused bits.
>
> - Left-justified (signed): with the most significant bit inverted to form a sign bit, and zeros in the unused lower order bits. In this mode, zero is treated as the half scale of the input range.
>
> - Left-justified (unsigned): with zeros in the unused lower order bits.
>
> Refer to section 13.13, in the "Queued Analog-to-Digital Converter Module-64" section of the *MPC555 User's Manual* for further information.

**Sample time**

> Block sample time; determines sample rate at which the port is monitored.

## Channel Number Selection

A channel number in the **Channels** vector selects the input channel number corresponding to the analog input pin to be sampled and converted. The analog input pin channel number assignments and the pin definitions vary, depending on whether the QADC64 is operating in multiplexed or nonmultiplexed mode. The queue scan mechanism makes no distinction between an internally or externally multiplexed analog input.

# QADC Analog In

The following two tables show the mapping between the channel numbers and the hardware pins for the two scanning modes (multiplexed and nonmultiplexed).

For example, in nonmultiplexed mode, to scan all 16 channels of the QADC64 you would specify the following vector in the **Channels** field:

[ 0 1 2 3 48 49 50 51 52 53 54 55 56 57 58 59 ]

**Nonmultiplexed Scan Mode**

| Port Pin Name | Analog Pin Name | Pin Type (I/O) | Channel Number |
|---|---|---|---|
| PQB0 | A_AD0 / AN0 | I | 0 |
| PQB1 | A_AD1 / AN1 | I | 1 |
| PQB2 | A_AD2 / AN2 | I | 2 |
| PQB3 | A_AD3 /AN3 | I | 3 |
| PQB4 | A_AD4 / AN48 | I | 48 |
| PQB5 | A_AD5 / AN49 | I | 49 |
| PQB6 | A_AD6 / AN50 | I | 50 |
| PQB7 | A_AD7 / AN51 | I | 51 |
| PQA0 | A_AD8 / AN52 | I/O | 52 |
| PQA1 | A_AD9 / AN53 | I/O | 53 |

**Nonmultiplexed Scan Mode (Continued)**

| Port Pin Name | Analog Pin Name | Pin Type (I/O) | Channel Number |
|---|---|---|---|
| PQA2 | A_AD10 / AN54 | I/O | 54 |
| PQA3 | A_AD11 / AN55 | I/O | 55 |
| PQA4 | A_AD12 / AN56 | I/O | 56 |
| PQA5 | A_AD13 / AN57 | I/O | 57 |
| PQA6 | A_AD14 / AN58 | I/O | 58 |
| PQA7 | A_AD15 / AN59 | I/O | 59 |

# QADC Analog In

**Multiplexed Scan Mode**

| Port Pin Name | Analog Pin Name | Pin Type (I/O) | Channel Number |
|---|---|---|---|
| PQB0 | A_AD0 / ANw | I | 0–14 even |
| PQB1 | A_AD1 / ANx | I | 1–15 odd |
| PQB2 | A_AD2 / ANy | I | 16–30 even |
| PQB3 | A_AD3 / ANz | I | 17–31 odd |
| PQB4 | A_AD4 / AN48 | I | 48 |
| PQB5 | A_AD5 / AN49 | I | 49 |
| PQB6 | A_AD6 / AN50 | I | 50 |
| PQB7 | A_AD7 / AN51 | I | 51 |
| PQA3 | A_AD11 / AN55 | I/O | 55 |
| PQA4 | A_AD12 / AN56 | I/O | 56 |
| PQA5 | A_AD13 / AN57 | I/O | 57 |
| PQA6 | A_AD14 / AN58 | I/O | 58 |
| PQA7 | A_AD15 / AN59 | I/O | 59 |

**Note** PQA0, PQA1 and PQA2 (corresponding to channels 52–54) are used as output pins (MA0, MA1, and MA2) to drive an external demultiplexer.

**Purpose**      Input driver enables use of Queued Analog-Digital Converter (QADC64) pins as digital inputs

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ Queued Analog-To-Digital Converter Module-64
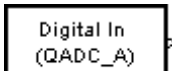
**Description**  The QADC Digital In block allows you to treat the QADC64 pins as digital inputs. Each QADC64 module has two 8-bit ports, A and B. You can use any bit on either port as a digital input.

```
Digital In
(QADC_A)
```

QADC Digital In

**Dialog Box**

Source Block Parameters: QADC Digital In

MPC555 Digital Input (QADC64) (mask) (link)

Digital input using one of the Queued Analog Digital Converter (QADC64) modules.

Specify the bits you want to read as a vector of numbers from [0..7]. Depending on the selected port, the bits entered correspond to pins:
- PQA0..PQA7 (Port A)
- PQB0..PQB7 (Port B)

Parameters

QADC module: A

Port: A

Bits:
[0 1 2 3]

Sample time:
0.1

OK      Cancel      Help

**QADC module**
Select module A or B.

**Port**
Select an 8 bit port (A or B) on the module.

# QADC Digital In
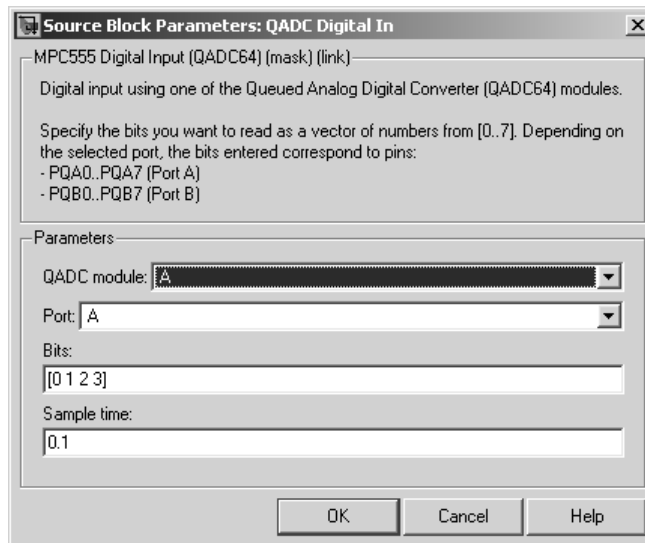
**Bits**

A vector of bits (numbered 0-7) to read. The vector should not be longer than eight elements.

**Sample time**

Block sample time; determines sample rate at which the port is monitored.

## Mapping Bits To Hardware Pins

Use this table to work out how the block ports and bits map to processor pins on the MPC555.

### Relationship of Port/Bit Parameters to Hardware Pins

| Port | Bit | Hardware Pin |
|------|-----|--------------|
| B | 0 | A_AD0 / PQB0 |
| B | 1 | A_AD1 / PQB1 |
| B | 2 | A_AD2 / PQB2 |
| B | 3 | A_AD3 / PQB3 |
| B | 4 | A_AD4 / PQB4 |
| B | 5 | A_AD5 / PQB5 |
| B | 6 | A_AD6 / PQB6 |
| B | 7 | A_AD7 / PQB7 |
| A | 0 | A_AD8 / PQA0 |
| A | 1 | A_AD9 / PQA1 |
| A | 2 | A_AD10 / PQA2 |
| A | 3 | A_AD11 / PQA3 |
| A | 4 | A_AD12 / PQA4 |
| A | 5 | A_AD13 / PQA5 |

**Relationship of Port/Bit Parameters to Hardware Pins (Continued)**

| Port | Bit | Hardware Pin |
|------|-----|--------------|
| A | 6 | A_AD14 / PQA6 |
| A | 7 | A_AD15 / PQA7 |

# QADCE Analog In

**Purpose**       Input driver enables use of Queued Analog-Digital Converter (QADC64) in continuous scan mode on MPC56x (561-6)

**Library**       Target for Freescale MPC5xx/ MPC555 Driver Library/ Enhanced Queued Analog-To-Digital Converter Module-64
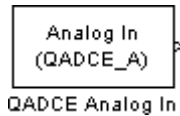
**Description**   The QADCE Analog In block sets the QADC64E into continuous scan mode. It then samples the specified channels at the specified rate. In continuous scan mode, the analog-to-digital converter is scanned as fast as possible, at a rate much faster than the sample rate of the model. Using continuous scan mode ensures that your application will obtain the latest signal value.

```
Analog In
(QADCE_A)
```
QADCE Analog In

The MPC56x has two QADC64E modules, A and B. You can program these individually. You can place only one instance of the QADCE Analog In block per module in your model or subsystem.
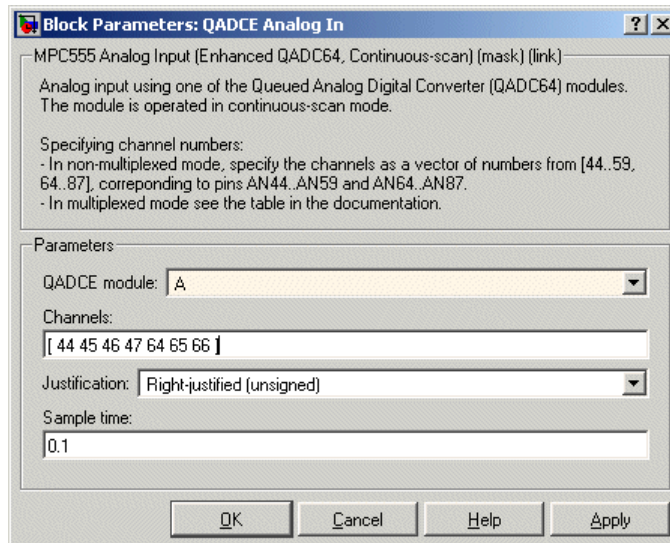
**Dialog Box**

**QADC module**
       Select module A or B.

**Channels**

A vector of numbers representing channels to be scanned. A channel number in the **Channels** vector selects the input channel number corresponding to the analog input pin to be sampled and converted.

The analog input pin channel number assignments and the pin definitions vary, depending on whether the QADC64E is operating in multiplexed or nonmultiplexed mode. The queue scan mechanism makes no distinction between an internally or externally multiplexed analog input.

In nonmultiplexed mode, specify a vector of numbers from [44..59 64..87] corresponding to pins AN44..AN59 and AN64..AN87.

See the table following for the mapping in multiplexed mode between the channel numbers and the hardware pins.

**Justification**

Converted data is read from the 10-bit wide QADC64E result word table into a 16-bit word. Data from the result word table can be accessed in three different formats. The **Justification** menu selects from the following formats:

`Right-justified (unsigned)`: with zeros in the higher order unused bits.

`Left-justified (signed)`: with the most significant bit inverted to form a sign bit, and zeros in the unused lower order bits. In this mode, zero is treated as the half scale of the input range.

`Left-justified (unsigned)`: with zeros in the unused lower order bits.

**Sample time**

Block sample time; determines sample rate at which the port is monitored

# QADCE Analog In

### Mapping Bits To Hardware Pins

Use the following table to work out how the block ports and bits map to processor pins on the MPC565 in multiplexed mode.

In summary

- No multiplexing:

  channels available 44-59 and 64-87

- A only multiplexing:

  channels available 0-31; 48-51; 55-59; 64-87

- B only multiplexing:

  channels available 0-31; 48-59; 64-71; 75-87

- A and B multiplexing:

  channels available 0-31; 48-51; 55-59; 64-71; 75-87

### Multiplexed Scan Mode

| Port Pin Name | Analog Pin Name | Other Functions | Pin Type (I/O) | Channel Number |
|---|---|---|---|---|
| ANw/A_PQB0 | AN00 to AN07 | - | Input | 0 to 7 |
| ANx/A_PQB1 | AN08 to AN15 | - | Input | 8 to 15 |
| ANy/A_PQB2 | AN16 to AN23 | - | Input | 16 to 23 |
| ANz/A_PQB3 | AN24 to AN31 | - | Input | 24 to 31 |
| A_PQB0 | AN44 | ANw | Input/Output | 44 |
| A_PQB1 | AN45 | ANx | Input/Output | 45 |
| A_PQB2 | AN46 | ANy | Input/Output | 46 |
| A_PQB3 | AN47 | ANz | Input/Output | 47 |
| A_PQB4 | AN48 | - | Input/Output | 48 |

**Multiplexed Scan Mode (Continued)**

| Port Pin Name | Analog Pin Name | Other Functions | Pin Type (I/O) | Channel Number |
|---|---|---|---|---|
| A_PQB5 | AN49 | - | Input/Output | 49 |
| A_PQB6 | AN50 | - | Input/Output | 50 |
| A_PQB7 | AN51 | - | Input/Output | 51 |
| A_PQA0 | AN52 | MA0 | Input/Output | 52 |
| A_PQA1 | AN53 | MA1 | Input/Output | 53 |
| A_PQA2 | AN54 | MA2 | Input/Output | 54 |
| A_PQA3 | AN55 | - | Input/Output | 55 |
| A_PQA4 | AN56 | - | Input/Output | 56 |
| A_PQA5 | AN57 | - | Input/Output | 57 |
| A_PQA6 | AN58 | - | Input/Output | 58 |
| A_PQA7 | AN59 | - | Input/Output | 59 |
| B_PQB0 | AN64 | - | AMUX Input | 64 |
| B_PQB1 | AN65 | - | AMUX Input | 65 |
| B_PQB2 | AN66 | - | AMUX Input | 66 |
| B_PQB3 | AN67 | - | AMUX Input | 67 |
| B_PQB4 | AN68 | - | AMUX Input | 68 |
| B_PQB5 | AN69 | - | AMUX Input | 69 |
| B_PQB6 | AN70 | - | AMUX Input | 70 |
| B_PQB7 | AN71 | - | AMUX Input | 71 |
| B_PQA0 | AN72 | MA0 | AMUX Input | 72 |
| B_PQA1 | AN73 | MA1 | AMUX Input | 73 |
| B_PQA2 | AN74 | MA2 | AMUX Input | 74 |

**Multiplexed Scan Mode (Continued)**

| Port Pin Name | Analog Pin Name | Other Functions | Pin Type (I/O) | Channel Number |
|---|---|---|---|---|
| B_PQA3 | AN75 | - | AMUX Input | 75 |
| B_PQA4 | AN76 | - | AMUX Input | 76 |
| A_PQA5 | AN77 | - | AMUX Input | 77 |
| A_PQA6 | AN78 | - | AMUX Input | 78 |
| A_PQA7 | AN79 | - | AMUX Input | 79 |
| - | AN80 | - | - | 80 |
| - | AN81 | - | - | 81 |
| - | AN82 | - | - | 82 |
| - | AN83 | - | - | 83 |
| - | AN84 | - | - | 84 |
| - | AN85 | - | - | 85 |
| - | AN86 | - | - | 86 |
| - | AN87 | - | - | 87 |

In this table, MA0 to MA2 indicates these pins (A_ and B_PQA0-2) are used as output pins to drive an external demultiplexer.

**Purpose**    Input driver enables use of Queued Analog-Digital Converter (QADC64) pins as digital inputs on MPC56x (561-566)

**Library**    Target for Freescale MPC5xx/ MPC555 Driver Library/ Enhanced Queued Analog-To-Digital Converter Module-64

**Description**    The QADCE Digital In block allows you to treat the QADC64E pins as digital inputs. Each QADC64E module has two 8-bit ports, A and B. You can use any bit on either port as a digital input.

Digital In
(QADCE_A)
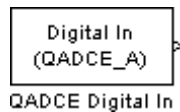
QADCE Digital In

**Dialog Box**
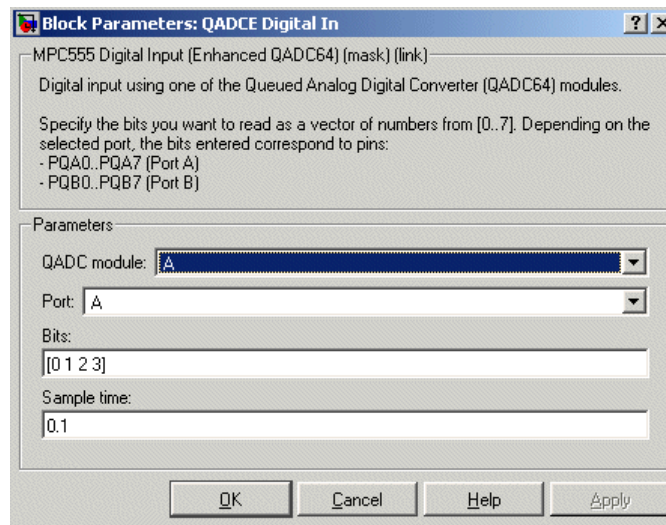


**QADC module**

> Select module A or B.

**Port**

> Select an 8 bit port (A or B) on the module.

# QADCE Digital In

**Bits**

Specify a vector of bits (numbered 0-7) to read. The vector should not be longer than eight elements. Depending on the selected port, the bits entered correspond to pins PQA0 to PQA7 (port A) or PQB0 to PQB7.
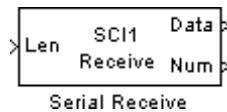
**Sample time**

Block sample time; determines sample rate at which the port is monitored

**Purpose**     Configure MPC555 for serial receive on either of QSMCM submodules
                SCI1 or SCI2

**Library**     Target for Freescale MPC5xx/ MPC555 Driver Library/ Serial
                Communications Interface (SCI)

**Description** The Serial Receive block receives bytes via either of the MPC555
                QSMCM submodules SCI1 or SCI2. It requests either a fixed number of
                bytes to be received, or, by enabling the first input, a variable number
                of bytes can be requested each time this block is called. When the
                block is called, the requested number of bytes are retrieved from a
                hardware buffer provided by the submodule SCI1 or SCI2. On SCI1,
                the total size of the buffer is 16 bytes; note however that the effective
                capacity is reduced due to the hardware behavior and the circular mode
                of buffer operation used by the software driver. You should design your
                application on the basis of 9 bytes for the maximum buffer size for SCI1.
                On SCI2, the size of this buffer is 1 byte.

If the buffer contains fewer bytes than the number requested, these
bytes are pulled from the buffer and made available at the block
output. The number of bytes actually retrieved from the buffer is made
available at the second output. This block will only retrieve bytes that
have already been received and placed in the hardware buffer; it will
never wait for additional data to be received.

To configure the serial interface bit rate and data format, see "Serial
Communications Interface (SCI) Configuration Parameters" on page
5-50.

The device driver used for the Serial Receive block does not require
the use of CPU interrupts.

Block Inputs and Outputs

The first input can be enabled so a variable number of bytes can be
requested each time.

The second input, if enabled, is a reset signal, which must have a
Boolean data type. You must reset the SCI1 module if an overrun error
or framing or parity error occurs. No reset is required for SCI2.

# Serial Receive

The first output (marked `Data`) pulls bytes from the buffer — either the number requested or the number available, whichever is the lower. Note that the number requested is the value of the first input signal if supplied, or the width of the output signal otherwise.

The second output (marked `Num`) is the number of bytes actually retrieved from the buffer. Up to four outputs can be enabled — the third showing framing error and parity error flags, and the fourth showing overrun flags.

See "Data Type Support and Scaling for Device Driver Blocks" on page 1-29 for information on supported input/output data types and scaling of input/output signals.

**Dialog Box**
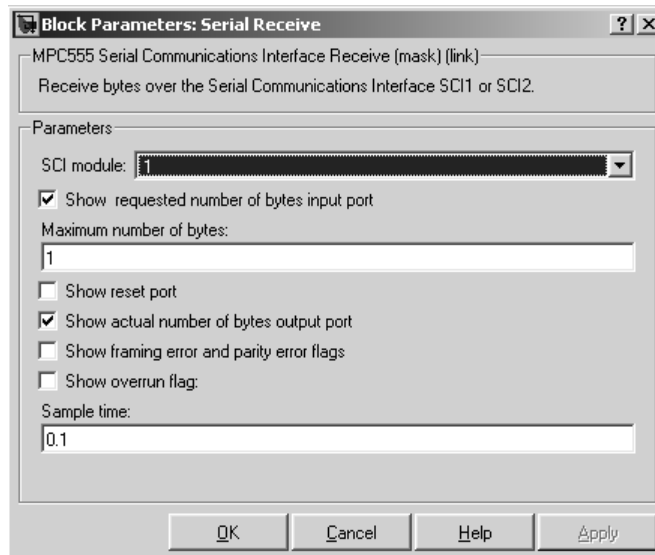


**SCI module**
> Select either 1 or 2 (to choose module SCI1 or SCI2).

**Show requested number of bytes input port**
> Enables an inport (the top one if there are two) where you can input the number of bytes to request.

**Maximum number of bytes**

Maximum number of bytes to receive (this is only visible if the requested number of bytes input port is enabled). This sets an upper limit on the number of bytes that will be read each time the block is called.

**Show reset port**

Enables the reset input (the lower inport).

**Show actual number of bytes output port**

Enables another output that shows the number of bytes actually read from the SCI buffer.

**Show framing error and parity error flags**

Enables another output. This output is zero if no framing or parity error occurred during the current read; it is true (1) otherwise. Note that for SCI1 only, a reset is required once a data overrun has occurred.

**Show overrun flag**

Enables another output. This output is true (1) if a data overrun occurred. Note that for SCI1 only, a reset is required once a data overrun has occurred.

**Sample time**

The time interval between samples. To inherit the sample time, set this parameter to -1. See "Specifying Sample Time" in the Simulink documentation for more information.
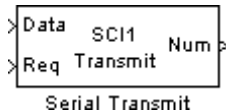
# Serial Transmit

**Purpose**      Configure MPC555 for serial transmit, using one of QSMCM submodules SCI1 or SCI2

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ Serial Communications Interface (SCI)

**Description**  The Serial Transmit block transmits bytes via either of the MPC555 QSMCM submodules SCI1 or SCI2. You can use it either to transmit a fixed number of bytes, or, by enabling the second input, transmit a variable number of bytes each time this block is called. With SCI1, a hardware buffer is used that allows up to 16 bytes to be queued for transmission. With SCI2, the buffer allows only up to one byte to be queued each time the block is called. Once bytes are queued for transmit, they will be sent as fast as possible by the serial interface hardware with no further intervention required by the rest of the application.

If the hardware buffer is not empty when the block is called, i.e., the previous transmission is not yet complete, then no new bytes will be queued for transmit. This condition can be identified from the "actual number of bytes" block output; if no bytes were queued for transmit, this output returns zero.

To configure the serial interface bit rate and data format, see "Serial Communications Interface (SCI) Configuration Parameters" on page 5-50.

The device driver used for the Serial Transmit block does not require the use of CPU interrupts.

Block Inputs and Outputs

The first input contains the data to be transmitted; this input signal may be either a vector or scalar with data type `uint8`. The optional second input must be a scalar and may be used to control the number of bytes transmitted. The number of bytes to transmit should not be greater than the width of the first input signal.

The block output port "actual number of bytes output" gives the number of bytes queued for transmit. If the previous transmission was complete,

this number will be equal to the requested number of bytes to transmit, provided that this was less or equal to 16 in the case of SCI1, or 1 in the case of SCI2. See "Data Type Support and Scaling for Device Driver Blocks" on page 1-29 for information on supported input/output data types and scaling of input/output signals.

## Dialog Box
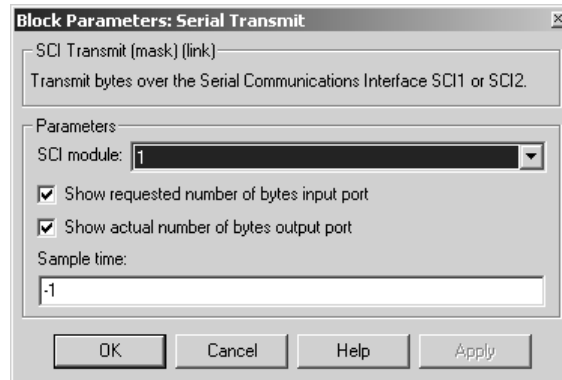


### SCI module

Select either 1 or 2 (to choose module SCI1 or SCI2).

### Show requested number of bytes input port

Enable/disable the input for number of bytes to send. If cleared, the number of bytes sent is just the width of the first inport; if selected, the second input is enabled, which controls the number of bytes to send.

### Show number of bytes output port

Enable/disable the output port for number of bytes actually sent. If selected, this value is available from the first output.

### Sample time

The time interval between samples. To inherit the sample time, leave this parameter at the default -1. See "Specifying Sample Time" in the Simulink documentation for more information.

# Switch External Mode Configuration

**Purpose**      Configure model for external mode or executable building

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ Utilities

**Description**  Place the Switch External Mode Configuration block in your model and double-click it to run a convenience function to configure your model for building an executable or executing your model in external mode. When you double-click the block, a dialog appears. Choose either **Building an executable** or **External mode**, and click **OK**.

When you choose building an executable, messages at the command line inform you the following steps are taken to configure your model:

**1** **Inline parameters** are selected (under Optimization in the Configuration Parameters dialog box). This is required for ASAP2 generation

**2** **Normal** simulation mode is selected (in the Simulation menu, and drop-down list in the toolbar).

**3** ASAP2 is selected as the **Interface** (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).

When you choose external mode, messages at the command line inform you the following steps are taken to configure your model:

**1** **Inline parameters** are selected (under Optimization in the Configuration Parameters dialog box). This is required for external mode.

**2** **External** simulation mode is selected (in the Simulation menu, and drop-down list in the toolbar).

**3** External mode is selected as the **Interface** (under Real-Time Workshop, Interface, in the Data Exchange pane, in the Configuration Parameters dialog box).

# Switch External Mode Configuration

See "Using External Mode" on page 2-33 for instructions for converting a model to use external mode for signal logging and parameter tuning.
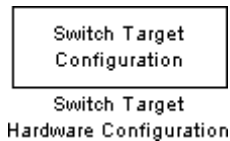
# Switch Target Configuration

**Purpose**     Configure model and target preferences to predefined hardware configuration

**Library**     Target for Freescale MPC5xx/ MPC555 Driver Library/ Utilities

**Description**

    Switch Target
    Configuration

    Switch Target
    Hardware Configuration

Place this block in your model and double click it to run a convenience function that configures your model and Target Preferences to one of a set of predefined configurations. If your setup does not correspond to one of the predefined configurations, you may wish to use the file (mpc555rtswitchconfig.m) as a template for setting up your own customized configurations. The predefined configurations include settings for:

- Phytec phyCORE-MPC555 (system frequency 20 or 40 MHz)
- Phytec phyCORE-MPC565 (system frequency 40 MHz)
- Axiom CME-555 (system frequency 40 MHz)
- Axiom CME-564 (system frequency 40 MHz)

**Purpose**    Count transmit and receive errors detected on selected TouCAN modules

**Library**    Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B
               Controller Module

**Description**  The TouCAN Error Count block maintains and reports a count of errors
                 detected by the selected TouCAN module during receive and transmit.
                 The receive and transmit error counts are output to the RX and TX
                 outputs of the block, respectively.

                 The error counts also drive the TouCAN Warnings block outputs. (See
                 TouCAN Warnings.)

**Dialog
Box**
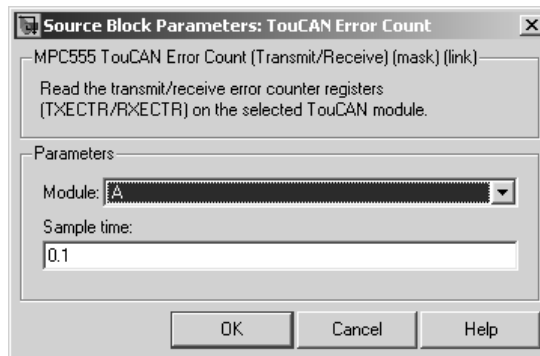


**Module**
    Select TouCAN module A, B or C. Note that the MPC555 only has
    modules A and B. MPC56x (561-6) also have module C. An error
    will be thrown if you select C and your target processor does not
    support this.

**Sample time**
    Sample time of the block.

# TouCAN Fault Confinement State

**Purpose**     Indicate state of TouCAN module

**Library**     Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B
                Controller Module

**Description**     The TouCAN Fault Confinement State block provides an indicator for
                    the state of the selected TouCAN module. The block obtains and outputs
                    a field of two bits from the TouCAN module's Error and Status (ESTAT)
                    register. The possible states are shown in the table below.

```
TouCAN_A
Fault
Confinement State
```
TouCAN Fault Confinement State
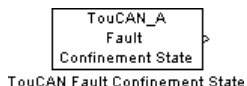
Refer to section 16, "CAN 2.0B Controller Module," in the *MPC555
User's Manual* for further information.

### FCS State Values

| State | Value | Description |
|---|---|---|
| Error Active | 00 | Normal operation |
| Error Passive | 01 | Listening only mode. The device cannot transmit. |
| Bus Off | 1x | The device is not allowed to transmit or receive and is effectively cut off from the bus. |

**Dialog
Box**

Source Block Parameters: TouCAN Fault Confinement ...

─MPC555 TouCAN Fault Confinement State (mask) (link)─
Indicates the fault confinement state of a TouCAN module.

─Parameters─

Module: A

Sample time:

0.1

| OK | Cancel | Help |

**Module**

    Select TouCAN module A, B or C. Note that the MPC555 only has modules A and B. MPC56x (561-6) also have module C. An error will be thrown if you select C and your target processor does not support this.
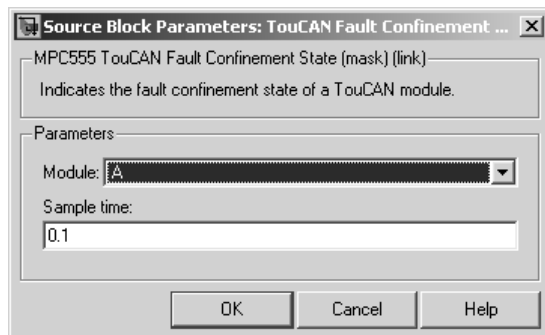
**Sample time**

    Sample time of the block.

# TouCAN Interrupt Generator

**Purpose**     Generate asynchronous function-call trigger when CAN interrupt occurs

**Library**     Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B
Controller Module

**Description**     The TouCAN Interrupt Generator block generates a function-call trigger
within the context of a TouCAN interrupt service routine, which can be
used to asynchronously execute a function-call subsystem in the model.

```
    TouCAN_A
    Interrupt
```
TouCAN Interrupt Generator
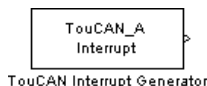
This block may be used to execute a function-call subsystem on
occurrence of Bus Off, Error, Wake, or buffer 0-15 interrupts.

Do not use this block unless you are aware of the dangers of using
asynchronous interrupts in the model. Unpredictable data loss or model
behavior may result unless extreme caution is taken. You must also
place an Asynchronous Rate Transition block on each input and output
of any subsystem that is triggered asynchronously by an interrupt, to
ensure data integrity. See Asynchronous Rate Transition.

For faster interrupts, you can disable floating-point support via the
**Use floating point** option. However, if you disable floating-point
support, do not use blocks that require floating-point operations in the
function-call subsystem. Use of such blocks will cause a floating-point
exception at run-time.

**Dialog Box**
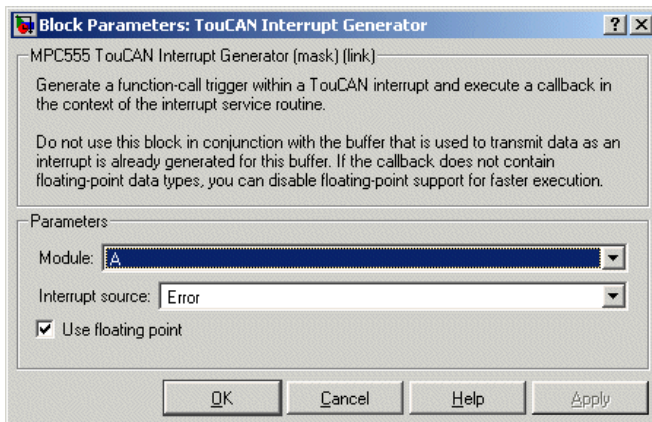


5-78

**Module**

Select TouCAN module A, B or C. Note that the MPC555 only has modules A and B. MPC56x (561-6) also have module C. An error will be thrown if you select C and your target processor does not support this.

**Interrupt source**

Choose the interrupt source (Bus Off, Error, Wake or Buffer 0-15) for your ISR generator.
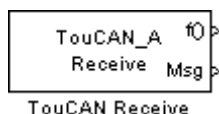
**Use floating point**

Enable or disable floating-point support.

# TouCAN Receive

**Purpose**      Receive CAN messages from TouCAN module on MPC5xx

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B
Controller Module

**Description**      The TouCAN Receive block receives CAN messages from the TouCAN
module.

TouCAN_A    f()
Receive    Msg

TouCAN Receive

The TouCAN Receive block can reserve any of the 16 buffers on the
TouCAN module. Alternatively, you can instruct the TouCAN Receive
block to select a hardware buffer automatically from the available
buffers.

The TouCAN Receive block provides two alternative mechanisms for
notifying downstream blocks that a new message has arrived. The
default behavior is that the block has a Function Call Outport; in
this case the associated trigger is activated whenever a new message
becomes available. The alternative option is more complex and involves
use of a separate TouCAN Interrupt Generator block; the TouCAN
Interrupt Generator block can be used to execute the downstream
function call subsystem within the context of the CAN interrupt service
routine. This alternative option is recommended for advanced users
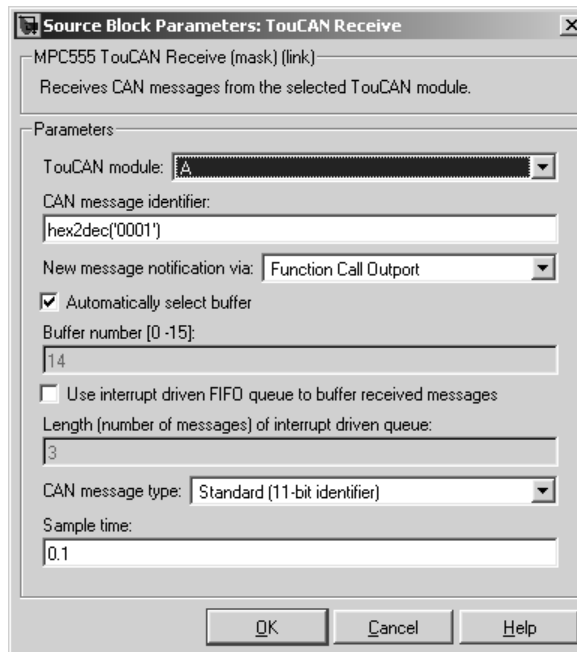only. In most applications it is recommended to use the Function Call
Outport.

With the Function Call Outport mode the TouCAN Receive block polls
its message buffer at a rate determined by the block's sample time.
When the TouCAN Receive block detects that a message has arrived,
the function call trigger is activated.

An additional option for use with the Function Call Output mode is to
use a FIFO queue. In this mode, instead of polling the hardware buffer
directly, the block polls a software FIFO buffer. Each time a message
is received in the hardware buffer for this block an interrupt service
routine automatically transfers the message to the FIFO buffer. On each
block update, the FIFO is cleared by processing the messages in turn; a
separate function call is generated for each message that is extracted
from the FIFO. If it is known that the block sample time is smaller than
the minimum time between messages that the block must receive then

you should use the standard mode of operation where the hardware buffer is polled directly. However, if the messages may be arriving faster than the block is polling the buffer, you should use the FIFO mode.

Tip: if you need to receive several different messages with different identifiers, arriving at irregular intervals, into a single buffer, you can use one of the dedicated receive masks for buffers 14 or 15 along with a CAN Message Filter block, and a TouCAN Receive block operating in FIFO mode. See the Masks parameters in "TouCAN Configuration Parameters" on page 5-43.

**Dialog Box**



Source Block Parameters: TouCAN Receive

MPC555 TouCAN Receive (mask) (link)

Receives CAN messages from the selected TouCAN module.

Parameters

TouCAN module: A

CAN message identifier:

hex2dec('0001')

New message notification via: Function Call Outport

☑ Automatically select buffer

Buffer number [0 -15]:

14

☐ Use interrupt driven FIFO queue to buffer received messages

Length (number of messages) of interrupt driven queue:

3

CAN message type: Standard (11-bit identifier)

Sample time:

0.1

OK    Cancel    Help

**TouCAN module**

Select one of the two TouCAN modules (A or B) on the MPC555. MPC56x (561-6) also have module C. The TouCAN modules can receive messages independently. Note that an error will be thrown if you select C and your target processor does not support this.

The CAN C module shares its pins with the MIOS module (which pins are shared depends on the variant). If you use the CAN C module and MIOS module together, you may experience resource conflicts which you will need to resolve.
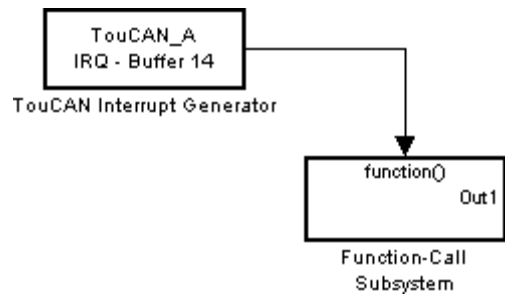
**CAN message identifier**

The identifier of the message you want to receive. Note that if you have set the TouCAN configuration parameters (see "TouCAN Configuration Parameters" on page 5-43) in your model to mask out certain bits (e.g., the message identifier field) you may receive messages with identifiers other than the identifier specified here.

**New message notification via:**

`Function Call Output` — Synchronous notification that a new message has arrived.

`` `TouCAN Interrupt Generator' block `` — If you select this option you must place the TouCAN Receive block in a function-call subsystem that is asynchronously triggered by a TouCAN Interrupt Generator block (as shown below). When you select this option, the function call output is no longer required, and disappears. Make sure you select the same receive buffer within the TouCAN Interrupt Generator and the TouCAN Receive block. When a message is received in the specified buffer the TouCAN Interrupt Generator block generates a function-call trigger (within the context of a TouCAN interrupt service routine), which can be used to asynchronously execute the function-call subsystem containing the TouCAN Receive block. See TouCAN Interrupt Generator for details.

```
        ┌──────────────┐
        │  TouCAN_A    │
        │ IRQ - Buffer 14│─────────────┐
        └──────────────┘              │
        TouCAN Interrupt Generator     │
                                       │
                                       ▼
                              ┌──────────────┐
                              │  function()  │
                              │        Out1 ▷│
                              └──────────────┘
                               Function-Call
                                 Subsystem
```

**Automatically select buffer**

When this option is selected, the TouCAN Receive block automatically selects a receive buffer from the available buffers. We recommend that you use this automatic buffer selection, unless you want to use buffer 14 or 15, which can be masked individually, to receive multiple CAN message identifiers in a single buffer. See the Mask parameters in "TouCAN Configuration Parameters" on page 5-43.

**Buffer number [0..15]**

This field is enabled if the **Automatically select buffer** option is cleared. **Buffer number** specifies the identifier of the receive buffer for this block. We recommend that you select **Automatically select buffer** instead of manually specifying the buffer, unless you want to use buffer 14 or 15, which can be masked, to receive multiple CAN message IDs in a single buffer. See the Mask parameters in "TouCAN Configuration Parameters" on page 5-43.

**Use interrupt driven FIFO queue to buffer received messages**

Use the FIFO mode if the messages may be arriving faster than the block is polling the buffer. Use this option if the messages may be arriving faster than the block is polling the buffer.

**Length (number of messages) of interrupt driven queue**

This field is enabled if you select the interrupt driven queue option, then you can specify a number of messages.

**CAN message type**

The type of message you want to receive. Select either
`Standard(11-bit identifier)` or `Extended(29-bit`
`identifier)`.

**Sample time**

Determines the rate at which to sample the buffer to see if a new
message has arrived. Set to `-1` (inherited) if using this block in
a function-call subsystem triggered by the TouCAN Interrupt
Generator block.

---

**Note** The TouCAN Receive block sample time should be set to
a value that is smaller than the minimum time between CAN
messages that will be received into the corresponding buffer. If
the minimum time between messages may be shorter, use the
FIFO mode (select interrupt driven queue). Otherwise if more
than one message is received into a buffer during a single sample
interval, the older message will be overwritten.

---

**Purpose**       Reset TouCAN module

**Library**       Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B
                  Controller Module
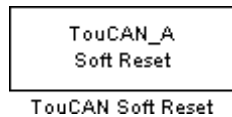
**Description**   When the TouCAN Soft Reset block executes, the TouCAN module
                  resets its internal state. The TouCAN error counters will be reset.

```
TouCAN_A
Soft Reset
```

TouCAN Soft Reset

The Fault Confinement State will be reset to the Error Active state,
provided the TouCAN module has not reached the Bus Off state. See
TouCAN Fault Confinement State.

We recommend that you place this block in a triggered or function-call
subsystem, with a sample time of -1 (inherited).

**Dialog
Box**
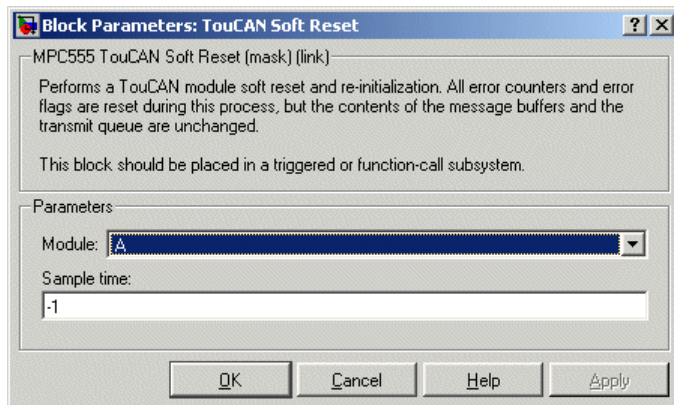


**Module**

Select TouCAN module A, B or C. Note that the MPC555 only has
modules A and B. MPC56x (561-6) also have module C. An error
will be thrown if you select C and your target processor does not
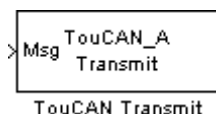support this.

**Sample time**

Sample time of the block.

# TouCAN Transmit

**Purpose**      Transmit CAN message via TouCAN module on MPC5xx

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B
                 Controller Module

**Description**      The TouCAN Transmit block transmits a CAN message onto the CAN
                     bus. The TouCAN Transmit block uses the queue set up by the MPC555
                     Resource Configuration object (see MPC555 Resource Configuration).
                     The block should be connected to CAN Message Packing blocks. Do not
                     ground the block or leave it unconnected. See the demos mpc555rt_io
                     and mpc555rt_candb for an example.

The TouCAN Transmit block provides three different transmission
modes. You should choose which transmission mode to use depending
on the requirements of your application. The properties of each
transmission mode are summarized in the following table.

**Transmit Modes**

|  | Priority Queued Transmission with Shared Buffer | Direct Transmission with Dedicated Buffer | FIFO Queued Transmission with Dedicated Buffer |
|---|---|---|---|
| Uses Interrupts | Yes | No | Yes |
| Configurable queue size | Yes | No | Yes |

**Transmit Modes (Continued)**

| | Priority Queued Transmission with Shared Buffer | Direct Transmission with Dedicated Buffer | FIFO Queued Transmission with Dedicated Buffer |
|---|---|---|---|
| Order of message transmission | Messages transmitted in order of priority; a new message will overwrite any existing message that is in the queue and has the same identifier and type (standard or extended) | Most recent message overwrites any unsent message in the buffer | Messages transmitted in the order that they were placed in the queue |
| Hardware buffers consumed | Either one or three hardware buffers are shared by many CAN Transmit blocks | One hardware buffer required for each CAN Transmit block | One hardware buffer required for each CAN Transmit block |
| CPU time required | Generally more than the other modes; interrupts used but time required to service interrupts is longer because it takes account of message priorities and increases with queue length | Very little; no interrupts used | Little; interrupts used but very simple interrupt service routine |

For applications where the message contains time-sensitive (e.g. real-time sensor readings) information, it is recommended to use one of the `Priority queued transmission with shared buffer` or `Direct transmission with dedicated buffer` modes. For applications where

it is more important that messages are received in the order that they were queued for transmission (e.g. a data logging protocol), it is recommended to use the `FIFO queued transmission with dedicated buffer` mode.

Note that the `Queued transmission with shared buffer` mode can use one or three shared buffers depending upon the setting in the Resource Configuration block. See Transmit Shared Buffers in the TouCAN configuration settings of the MPC555 Resource Configuration object. When three buffers are used, the driver ensures that the message entered into arbitration to be transmitted via the CAN bus is always the highest priority message available; furthermore in this mode the TouCAN module is able to transmit messages continuously by re-loading hardware buffers that become empty while another buffer is active transmitting. The shared buffer approach uses either buffer 0 or buffers 0, 1, and 2, depending on the setting in the Resource Configuration block.

If the Queued transmission with shared buffer mode is configured to use three shared buffers, there is a small possibility that some messages would be transmitted more than once. If you want to prevent this behavior, you should use this mode with a single shared buffer or use a mode other than Queued transmission with shared buffer.

The 'Queued transmission with shared buffer' mode maintains a queue of messages that are loaded into a hardware buffer of the TouCAN module as soon as one is available. Note that if a new message is ready to be sent that is higher priority than messages already in the hardware buffers then the lowest priority message will be moved from the hardware buffer back into the queue. This approach ensures that a high priority message cannot be blocked by one or more lower priority messages that are already in the hardware buffers. Under some circumstances it is possible that a lower priority message will actually be transmitted despite being moved from the hardware buffer back into the software queue; if this happens, the message concerned would be transmitted twice rather than once.

**Dialog Box**
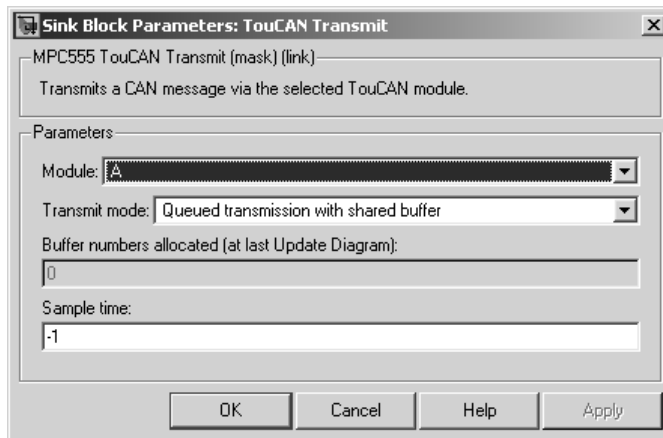


**Module**

Select TouCAN module A, B or C. Note that the MPC555 only has modules A and B. MPC56x (561-6) also have module C. An error will be thrown if you select C and your target processor does not support this.

The CAN C module shares its pins with the MIOS module (which pins are shared depends on the variant). If you use the CAN C module and MIOS module together, you may experience resource conflicts which you will need to resolve.

**Transmit mode**

Select one of the transmit modes described in the table.

**Length (number of messages) of FIFO queue**

If you select the FIFO transmit mode, you can set the number of messages in the FIFO queue here. Note this is only for the FIFO queue and is not the same as the Transmit_Queue_Length Resource Configuration parameter in "TouCAN Configuration Parameters" on page 5-43, which only applies to shared queues.

**Buffer numbers allocated (at last Update Diagram)**

Read only field for information on which buffers are in use.

**Sample time**

Choose -1 to inherit the sample time from the driving blocks. The TouCAN Transmit block does not inherit constant sample times and runs at the base rate of the model if driven by invariant signals.
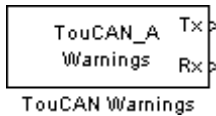
**Purpose**     Flag excessively high transmit or receive error counts on TouCAN modules

**Library**     Target for Freescale MPC5xx/ MPC555 Driver Library/ CAN 2.0B Controller Module
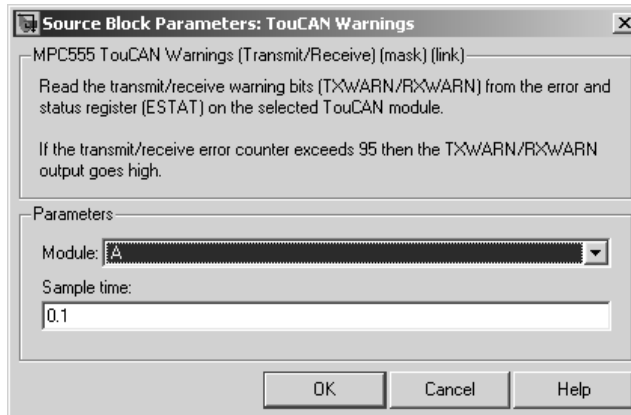
**Description**     The TouCAN Warnings block has two logical outputs, RX and TX. If the transmit error counter is over 95, then the TX output goes high. If the receive error counter is over 95, then the RX output goes high.

Use this block, in conjunction with a TouCAN Error Count block, to monitor error conditions on a selected TouCAN module.

**Dialog Box**



**Module**
> Select TouCAN module A, B or C. Note that the MPC555 only has modules A and B. MPC56x (561-6) also have module C. An error will be thrown if you select C and your target processor does not support this.
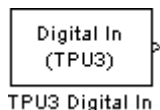
**Sample time**
> Sample time of the block.

# TPU3 Digital In

**Purpose**       Configure Time Processor Unit (TPU3) channel for digital input

**Library**       Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor Unit (TPU3)
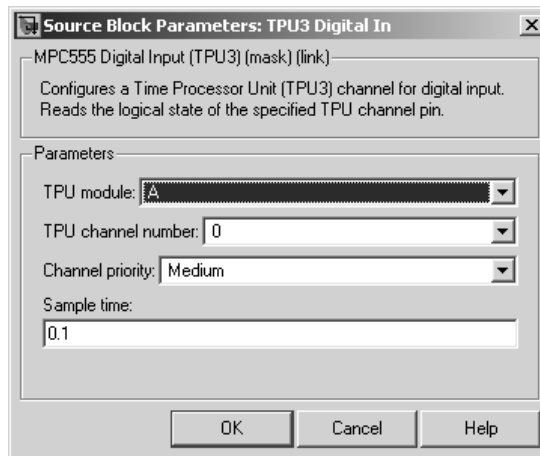
**Description**   The TPU3 Digital In block reads the logical state of the selected pin (channel) on the TPU3 submodules of the MPC555 or MPC56x. You can use this block in the same way as the MIOS Digital In block. You might need to use this block instead of the MIOS Digital In block, for example, if TPU is available but not MIOS. The Channel priority field specifies a number in the range 0..15, corresponding to 16 independent timer channels on each of the modules of the TPU3. The output of the block represents the logic state of the pin referenced in the module and channels fields. When the signal on a given pin is a logical 1, the block output signal will be equal to 1; otherwise the block output element will equal zero.

The TPU has 16 channels on each module A and B (MPC565 and 566 also have module C). You can use each of these channels independently, so for an MPC555 you could use up to 32 of these blocks, specifying different channels, at once.

Refer to Section 17, "Time Processor Unit 3," in the *MPC555 User's Manual* for further information, and the TPU3 Digital I/O Application Programming Note (search for "TPUPN18/D").

For an example showing how to use this block see the `mpc555rt_io` demo.

**Dialog Box**



**TPU module**

Select TPU module A, B or C; each has 16 channels. Note that the MPC555 only has modules A and B. MPC565 and MPC566 also have module C. An error will be thrown if you select C and your target processor does not support this.

**TPU channel number**

Choose 0-15.

**Channel priority**

Choose Low, Medium or High.

The host CPU makes a channel active by assigning it one of the three priorities. You choose the order in which channels are serviced by setting the channel number and assigned priority. The order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest number first).
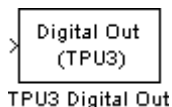
**Sample time**

The default is always 0.1 for input driver blocks, but you will need to change this to suit the frequency of your input signals.

# TPU3 Digital Out

**Purpose**        Configure Time Processor Unit (TPU3) channel for digital output

**Library**        Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor
                   Unit (TPU3)

**Description**    The TPU3 Digital Out block sets the state of the selected pin (channel)
                   on the TPU3 submodule of the MPC555 (or MPC565 or MPC566).

```
  Digital Out
   (TPU3)
```
TPU3 Digital Out

The Channel priority field specifies a number in the range 0..15,
corresponding to the 16 independent channels on each TPU3 module (A,
B or C). You can use each of these channels independently, so you could
use up to 32 of these blocks (48 for an MPC565 or MPC566) specifying
different channels at once.

When the input signal is greater than zero, a logical 1 is written to the
corresponding pin. When the input signal is less than or equal to zero, a
logical zero is written to the corresponding channel.

Refer to Section 17, "Time Processor Unit 3", in the *MPC555 User's
Manual* and the TPU3 Digital I/O Application Programming Note
(search for "TPUPN18/D") for further information about the TPU3.

For an example showing how to use this block see the mpc555rt_io
demo.
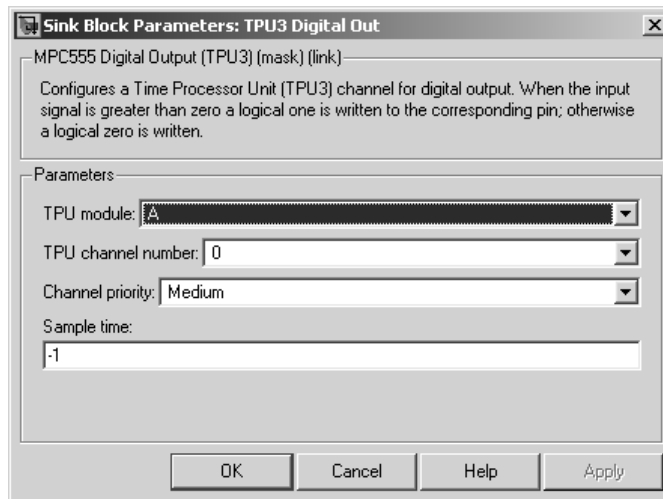
**Dialog Box**



**TPU Module**

Select TPU module A, B or C; each has 16 channels. Note that the MPC555 only has modules A and B. MPC565 and MPC566 also have module C. An error will be thrown if you select C and your target processor does not support this.

**TPU channel number**

Choose 0-15.

**Channel priority**

Choose Low, Medium or High.

The host CPU makes a channel active by assigning it one of the three priorities. You choose the order in which channels are serviced by setting the channel number and assigned priority. The order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest first).

**Sample time**

Default -1: this setting specifies that the block inherits its sample time from the block connected to its input (inheritance) (unless it is in a triggered subsystem). It makes no sense to sample faster

# TPU3 Digital Out

than your input is changing, so normally you should leave this at the default.

TPU Digital Out doesn't use a timebase. The output pin is written to at the rate specified by the block sample time. See "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46 for details on settings for the TCR1 clock. See also the TPU3 Digital In Application Programming Note (search for "TPUPN18/D").
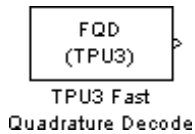
**Purpose**        Configure pair of TPU3 channels for Fast Quadrature Decode (FQD)

**Library**        Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor Unit (TPU3)

**Description**

```
   FQD
  (TPU3)
TPU3 Fast
Quadrature Decode
```

The TPU3 Fast Quadrature Decode block decodes position information from quadrature encoder hardware. The relative phase of a pair of input signals is used to determine direction of movement. The signals are decoded to increment or decrement the position counter (block output). You can derive a speed from the position information. It is particularly useful for decoding position and direction information from a slotted encoder in motion control systems.

In normal mode (the default), the position counter is incremented or decremented for each valid transition on either channel. The counter increments when the primary channel is ahead and decrements when the primary channel lags. A switch in the phase relationship indicates a change of direction.

At certain speeds you may want to switch to fast mode. You can supply an input to tell the block to switch to fast mode under specified conditions. In fast mode only one of the two input signals is read. The position counter increments or decrements by 4 for each rising transition on the primary channel only (instead of once for each transition in each signal). This reduces the TPU processing load; you can also decode at more than four times the maximum count rate of normal mode.

The counter is 16 bit and free flowing (that is, it overflows to 0, and underflows to 0xFFFF). You must take care when calculating speed derived from the counter, as it may be necessary to use two's complement arithmetic. A useful document is the *TPU Fast Quadrature Decode Programming Note* — search for "*TPUPN02/D.*"

It is possible to overload the TPU processor; if you observe unexpected behavior you should consult the TPU documentation. Refer to Section 17, "Time Processor Unit 3," in the *MPC555 User's Manual* for further information.

# TPU3 Fast Quadrature Decode

**Dialog Box**
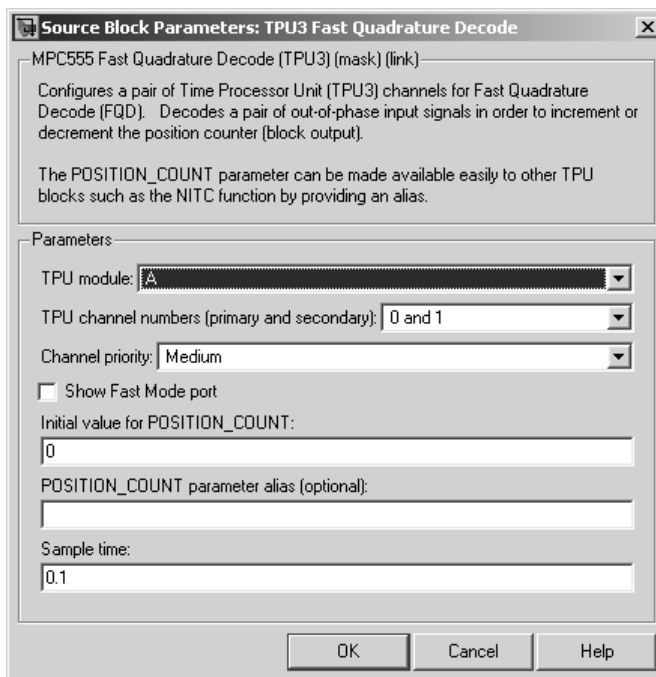


**TPU module**

Select TPU module A, B or C; each has 16 channels. Note that the MPC555 only has modules A and B. MPC565 and MPC566 also have module C. An error will be thrown if you select C and your target processor does not support this.

**TPU channel numbers (primary and secondary)**

Select a pair of consecutive channels from (0 and 1) to (14 and 15). The primary channel is always the lower channel number.

**Channel priority:**

Choose Low, Medium, or High

The order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest number first).

**Show Fast Mode port**

This option is unselected by default. Left unselected, the block always operates in `Normal` mode. If you select this option, an inport appears where you can input a Boolean signal to control the mode of operation (for example, from a Stateflow subsystem): 0 or false = Normal Mode; 1 or true = Fast Mode.

Fast mode conserves TPU activity by only reading one of the two signals. This also allows you to decode at more than four times the maximum count rate of `Normal` mode. This may be appropriate at some speeds where you can assume the behavior of the second sign — instantaneous direction change is assumed to be impossible. The counter is updated in the same direction as when the last transition was serviced in Normal Mode. The position counter is incremented or decremented by 4 for every rising transition read on the primary channel, instead of having to read all four transitions in the two signals.

**Initial value for POSITION_COUNT**

Set an initial value. Range checking is applied (must be 16 bit).

**POSITION_COUNT parameter alias** (optional)

Provide a name that blocks such as the TPU3 New Input Capture/Input Transition Counter can use to refer to the `POSITION_COUNT` Fast Quadrature Decode parameter (see TPU3 New Input Capture/Input Transition Counter). Using a name is clearer than using absolute channel and parameter indices to refer to the position count from another TPU block.

**Sample time**

The default is always `0.1` for input driver blocks, but you will need to change this to suit the frequency of your input signals.

This block uses TCR1 as a timebase, but the functionality of the TPU Fast Quadrature Decode (FQD) function used by the block is not changed by changing the speed of the TCR1 clock. The Position Count output is incremented at a rate entirely controlled by the rising and falling edges of the pair of input waveforms

# TPU3 Fast Quadrature Decode

(and the Fast mode input). See "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46 for more information on the TCR1 timebase settings.

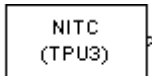# TPU3 New Input Capture/Input Transition Counter

**Purpose**      Configure Time Processor Unit (TPU3) channel for New Input Capture/Input Transition Counter (NITC)

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor Unit (TPU3)

**Description**

NITC
(TPU3)

TPU3 New Input Capture/
Input Transition Counter

The TPU3 New Input Capture/Input Transition Counter block counts transitions on the input pin and/or captures a TCR timebase value or a TPU parameter RAM value after a certain number of transitions. You can select the number of transitions and whether to capture on rising or falling transitions or both.

You can select up to three outputs to display. Each will have a separate outport:

- FINAL_TRANS_TIME shows the captured value each time the maximum number of transitions (MAX_COUNT) is reached

- TRANS_COUNT shows the number of transitions counted (resets each time MAX_COUNT is reached)

- LAST_TRANS_TIME shows the captured value at the most recent transition, updated at every transition (except final transitions). At the final transition LAST_TRANS_TIME shows the captured value at the previous transition.

You can choose whether to capture the TCR1 timebase value each time the MAX_COUNT number of transitions is reached, or you can specify the address of a TPU parameter in RAM to capture at that moment. Note this block always operates in continuous mode, not single-shot — transitions are counted up to MAX_COUNT and then the block resets and continues counting from zero.

We cannot guarantee that the three outputs are read coherently. They are read one after another, and it is possible that while the memory is accessed for one parameter the next to be read may have changed value. This depends on the speed of your input signal. This should not be important for most purposes because only TRANS_COUNT or FINAL_TRANS_TIME will be the outputs of interest.

# TPU3 New Input Capture/Input Transition Counter

As an example, you could use this block in conjunction with the TPU3 Fast Quadrature Decode block for calibration purposes. Quadrature encoders often generate an index signal in addition to the pair of signals whose relative phase contains the position information. You could put this index signal into an NITC input to count pulses in order to calibrate the position of the encoder.

Refer to Section 17, "Time Processor Unit 3," in the *MPC555 User's Manual* for further information. A particularly useful document is the *TPU New Input Capture/Input Transition Capture Programming Note* — search for "*TPUPN08/D.*" Look in the appropriate TPU programming note to look up parameter addresses if you want to capture TPU Parameters instead of TCR1 clock ticks.

As an example of using TPU parameters, if you wanted to use this block to capture the position count from a TPU Fast Quadrature Decode block, you need to set the correct channel number and parameter address. You must set the channel number to the primary FQD channel (FQD blocks use a pair of channels, the first is primary). Each TPU channel can have up to eight parameters (0 through 7), in this case you must choose parameter 1 (`POSITION_COUNT`).

**Dialog Box**



**Source Block Parameters: TPU3 New Input Capture/ Input Transitio...**

MPC555 New Input Capture/Input Transition Counter (TPU3) (mask) (link)

Configures a Time Processor Unit (TPU3) channel for New Input Capture/Input Transition Counter (NITC). Counts individual transitions on the input pin, and allows the capture of a TCR or TPU parameter RAM value after a selectable number of pin transitions.

Parameters

TPU module: A

TPU channel number: 0

Channel priority: Medium

☑ Show FINAL_TRANS_TIME port

☐ Show TRANS_COUNT port

☐ Show LAST_TRANS_TIME port

Detect transition on: Rising Edge

Capture: TCR1 Value

Specify parameter location by: Channel and Parameter Index

TPU channel to capture parameter from: 0

Channel parameter (16-bit) to capture: 0

Parameter alias:

Number of transitions before capture and reset (MAX_COUNT):

1

Sample time:

0.1

OK    Cancel    Help

**TPU module**
　　Select TPU module A, B or C; each has 16 channels. Note that the MPC555 only has modules A and B. MPC565 and MPC566 also have module C. An error will be thrown if you select C and your target processor does not support this.

**TPU channel number**
　　Choose 0-15.

# TPU3 New Input Capture/Input Transition Counter

**Channel priority:**
>    Choose Low, Medium, or High

>    The host CPU makes a channel active by assigning it one of the three priorities. You choose the order in which channels are serviced by setting the channel number and assigned priority. The order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest number first).

**Show FINAL_TRANS_TIME port**
>    Outputs the value captured each time the maximum number of transitions (MAX_COUNT) is reached. This value is only captured when MAX_COUNT is reached.

**Show TRANS_COUNT port**
>    Outputs the number of transitions counted. Resets to zero each time MAX_COUNT is reached.

**Show LAST_TRANS_TIME port**
>    Outputs the captured value at the latest transition. This is updated at every transition except the final one.

**Detect transition on:**
>    Choose from Rising Edge, Falling Edge or Either Edge.

**Capture:**
>    TCR1 Value — captures the value of the TCR1 timebase. See "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46 for information on setting the TCR1 timebase.

>    Parameter RAM Value — captures the value of a TPU parameter in RAM. If you select this option you enable the parameters to choose the TPU channel number and parameter address, or to specify a parameter alias.

**Specify parameter location by**
>    Channel and Parameter Index — if you select this option you enable the two parameters to specify which TPU channel (from 0-15) and which parameter index (out of up to eight parameters per TPU channel) you want.

`Parameter Alias` — If you select this option you enable the **Parameter alias** edit box. For example you can specify a parameter alias for the POSITION_COUNT parameter in the TPU3 Fast Quadrature Decode block. See TPU3 Fast Quadrature Decode.

Note that you cannot set the parameter location unless you have chosen `Parameter RAM Value` for the **Capture** parameter.

**TPU channel to capture parameter from**

Specify which TPU channel (from 0-15) you want. This option is enabled when you choose to specify parameter location by `Channel and Parameter Index`.

**Channel parameter (16-bit) to capture**

Specify which parameter index (out of up to eight parameters per TPU channel) you want. This option is enabled when you choose to specify parameter location by `Channel and Parameter Index`.

**Parameter alias**

This option is enabled when you choose to specify parameter location by `Parameter Alias`. Enter the required alias in the edit box. For example you can specify a parameter alias for the POSITION_COUNT parameter in the TPU3 Fast Quadrature Decode block. See TPU3 Fast Quadrature Decode.

**Number of transitions before capture and reset (MAX_COUNT)**

This must be a 16-bit number specifying how many transitions to count before capturing and then resetting. A zero will be equivalent to 1 (you cannot count zero transitions) and you must not exceed the maximum of a uint16 number. The range of an unsigned 16-bit number is 0-65535 (because $65535 = (2^{16}) - 1$).

Range checking is applied; you will receive a warning if you input an unsuitable number.

**Sample time**

Be sure to set the sample time fast enough not to miss any transitions. This will depend on the frequency of your input signal.

# TPU3 Programmable Time Accumulator

**Purpose**　　　　Configure Time Processor Unit (TPU3) channel for Programmable Time Accumulator (PTA)

**Library**　　　　Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor Unit (TPU3)

**Description**

```
    PTA
   (TPU3)

TPU3 Programmable
 Time Accumulator
```
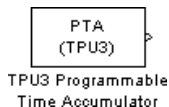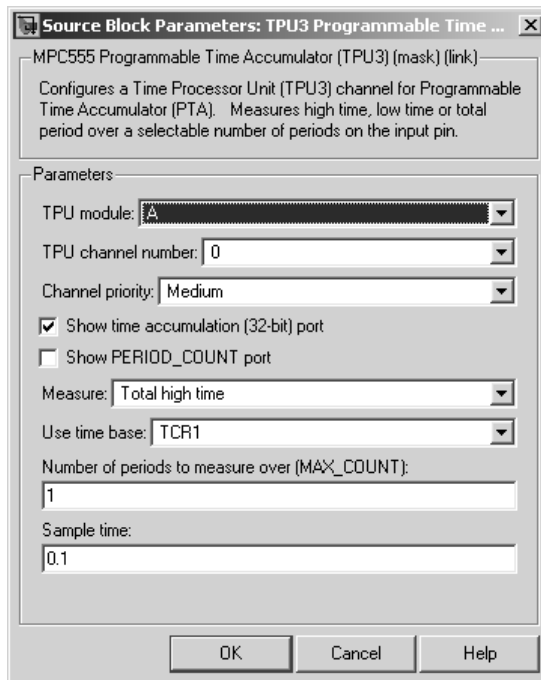
The TPU3 Programmable Time Accumulator block reads an input pin and measures an accumulation of time over a specified number of periods - either high time, low time, or the total time. You can output the accumulated time, the number of periods, or both. You can choose whether to start counting total period on a rising or falling edge.

The accumulated time value will be read at most once between any two model steps. TPU interrupts are used to ensure the 32-bit output is updated only when an accumulation is complete. This ensures that the values of the parameters HW and LW combined to create the 32-bit output are coherent. This block is under MPC555 Resource Configuration object control, and you will receive a warning if you have not enabled TPU interrupts. If your model contains any PTA blocks, you must change the TPU IRQ settings to enable interrupts. See "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46.

Refer to Section 17, "Time Processor Unit 3," in the *MPC555 User's Manual* for further information. A particularly useful document is the *Programmable Time Accumulator TPU Function (PTA) Programming Note* — search for "TPUPN06/D."

**Dialog Box**



### TPU module
Select TPU module A, B or C; each has 16 channels. Note that the MPC555 only has modules A and B. MPC565 and MPC566 also have module C. An error will be thrown if you select C and your target processor does not support this.

### TPU channel number
Choose 0-15

### Channel priority:
Choose Low, Medium, or High

The host CPU makes a channel active by assigning it one of the three priorities. You choose the order in which channels are serviced by setting the channel number and assigned priority. The order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest number first).

**Show time accumulation (32-bit) port**
Outputs the 32-bit time accumulation value (in TCR1 clock ticks) each time MAX_COUNT is reached. Whether the accumulation measures high time, low time or total time depends on the **Measure** setting.

**Show PERIOD_COUNT port**
Outputs the number of periods counted.

**Measure**:
Choose from Total high time, Total low time, Total period (starting on rising edge), Total period (starting on falling edge).

**Use time base**
Select TCR1 or TCR2. You can configure TCR2 to use an external clock. See "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46.

**Number of periods to measure over (MAX_COUNT):**
Set the number of periods to accumulate time over, up to a maximum of 255. The value is read each time MAX_COUNT is reached. Note that MAX_COUNT is 8-bit here (it is 16-bit in the TPU3 New Input Capture/Input Transition Counter block).

**Sample time:**
Make sure you set a sample time fast enough not to miss any periods, depending on the frequency of your input signal.
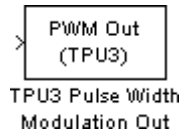
**Purpose**        Configure Time Processor Unit (TPU3) channel for pulse width modulation (PWM) output

**Library**        Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor Unit (TPU3)

**Description**

```
  PWM Out
  (TPU3)
```
TPU3 Pulse Width
Modulation Out

The TPU3 Pulse Width Modulation Out block is used for Pulse Width Modulation (PWM) output from the TPU3 modules. You can use this block in the same way as the MIOS PWM Out block, and with the TPU block you can also vary the period dynamically using a block inport. You can modulate up to 16 of these for each module (A, B or C) using any of the independent TPU channels.

A PWM signal is a rectangular waveform whose period may or may not be constant, and whose duty cycle can be varied, under control of a modulator signal, between 0% and 100%. You can either control the period register directly, or enter the desired (ideal) period and the mask will solve for the best values for the period register. Note for the MIOS Pulse Width Modulation Out block the period is constant, but with the TPU Pulse Width Modulation Out block you can also vary the period of the PWM signal (using the `input port for pulse period` option you can supply the period as an input).

The TPU3 Pulse Width Modulation Out block acts as the modulator, controlling the duty cycle and period of the signal on the output channel. There can be one or two inputs. Input one (top) is always the duty cycle. Here an input signal in the range 0 to 1 generates a PWM output with corresponding duty cycle. Input signals outside this range cause the duty cycle to saturate at 0% or 100%.

You can specify the period register manually in the mask. If you select the option `use input port for pulse period register value`, input two appears. Here you can supply the period as an input, instead of specifying the period in the mask. PWMPER input (either block input or specified as a mask variable) must be 16 bit values in the range 0 <= PWM Period Register Value <= 32768 (0x8000).

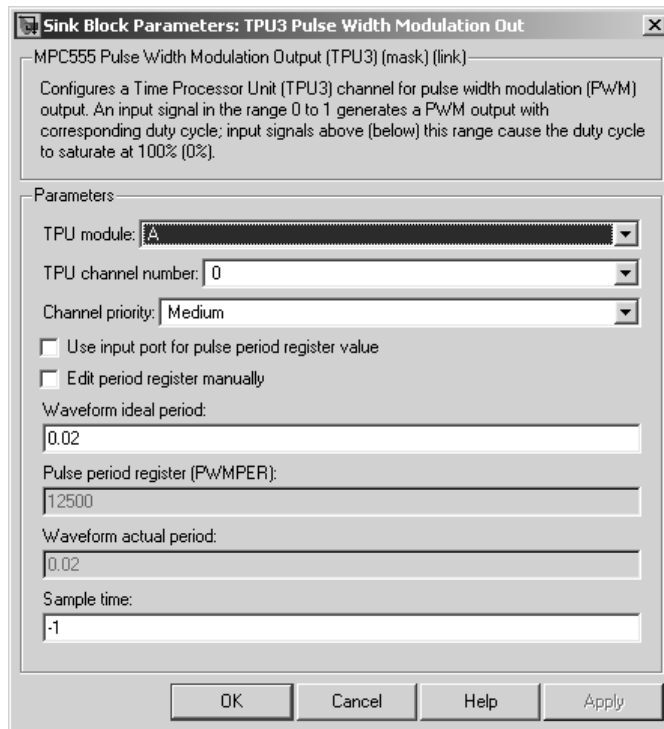# TPU3 Pulse Width Modulation Out

This saturation means that the block will not allow you to enter a value for PWMPER > 0x8000, or a value for `ideal period` that makes the PWMPER register go outside this range.

The TPU Pulse Width Modulation Out block uses TCR1 as a timebase for creating the output waveform. By changing the speed of the TCR1 clock, the range of available PWM periods changes. See "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46 for more information on settings for the TCR1 clock.

Refer to Section 17, "Time Processor Unit 3," in the *MPC555 User's Manual* for further information. See also the relevant TPU3 Application Programming Note (search for "TPUPN17/D").

For an example showing both ways to use this block (specifying the period, and using the PWMPER port to input the period), see the `mpc555rt_io` demo.

**Dialog
Box**



**TPU Module**

Select TPU module A, B or C; each has 16 channels. Note that the
MPC555 only has modules A and B. MPC565 and MPC566 also
have module C. An error will be thrown if you select C and your
target processor does not support this.

**TPU channel number**

Choose 0-15

**Channel priority**

Choose Low, Medium, or High

The host CPU makes a channel active by assigning it one of the
three priorities. You choose the order in which channels are

serviced by setting the channel number and assigned priority. The order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest number first).

**Use input port for pulse period register value**

If you select this box, the parameters relating to setting the period register disappear because they are no longer used.

A new inport appears on the block when you select this option. Here you can input the period register value. Saturation is applied: 0 <= x <= 32768 (0x8000). You can see an example of the block in the demo model mpc555rt_io.

**Edit period register manually**

If you select this check box, you can set the **Pulse period register** parameter.

**Waveform ideal period**

The default is 0.02. You can enter the waveform period you want by typing in this edit box. From this the period register is calculated and appears in the **Pulse period register (PWMPER)** edit box. The actual waveform period is also calculated and displayed, see below.

**Pulse period register (PWMPER)**

The default is 12500. You can enter a value for the period register here ( 0<= x <= 32768 (0x8000)) only if you select **Edit period register manually**. The actual waveform period is calculated and displayed in the actual period field. If **Edit period register manually** is not selected, this edit box is disabled (gray).

**Waveform actual period**

You can never enter anything in this box (so it is always gray) — it is there purely to inform you, and does not affect the model code. You might find this information useful because actual and ideal waveform period are not always the same — the ideal period you enter may not always be possible.

**Sample time**

The default is -1: This setting specifies that the block inherits its sample time from the block connected to its input (inheritance) (unless it is in a triggered subsystem). It makes no sense to sample faster than your input is changing, so normally you leave this at the default.

# TPU3 Rectangular Wave

**Purpose**       Configure Time Processor Unit (TPU3) channel for Rectangular Wave
                  Output (RECTW)

**Library**       Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor
                  Unit (TPU3)
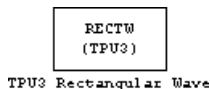
**Description**   This block is provided as an example along with the demo model
                  mpc555rt_tpu_emu. The rectangular wave function is not part of the
                  standard ROM mask of TPU functions but can be downloaded to
                  DPTRAM and used by the TPU in emulation mode.

RECTW
(TPU3)

TPU3 Rectangular Wave

The TPU3 Rectangular Wave block outputs a rectangular wave with a
specified high time and specified wave period. Pulses always begin
with a rising edge, and TCR1 is used as the timebase. You can either
control the high-time and waveform period registers directly, or enter
the desired (ideal) periods and the mask will solve for the best values
for the period registers.

If you select the option **Use input port to vary HIGH_TIME_RECTW
and PERIOD_RECTW**, two inputs appear. You can use these to vary
the high-time and waveform period. The rest of the parameters in the
mask are used as initial values. Input 1 (top) is the high time and
input 2 is the period. Inputs must be 16 bit values in the range $0 <= x
<= 32768$ (0x8000).

The TPU Rectangular Wave block uses TCR1 as a timebase for creating
the output waveform. By changing the speed of the TCR1 clock, the
range of available waveform periods changes. See "Time Processor Unit
(TPU3) Configuration Parameters" on page 5-46 for more information
on settings for the TCR1 clock.

Refer to Section 17, "Time Processor Unit 3," in the *MPC555 User's
Manual* for further information.

**Dialog Box**

Block Parameters: TPU3 Rectangular Wave

MPC555 Rectangular Wave (TPU3) (mask) (link)

Configures a Time Processor Unit (TPU3) channel for Rectangular Wave Output (RECTW)

Outputs a Rectangular Wave with the specified high time, and specified wave period. Pulses always begin with a rising edge, and TCR1 is used as the time base.

When an input port is used to vary HIGH_TIME_RECTW and PERIOD_RECTW, the rest of parameters in the dialog (H,T or HIGH_TIME_RECTW,PERIOD_RECTW) will be used as initial values.

Channel Setup | Waveform Setup

TPU module: A

TPU channel number: 0

Channel priority: Medium

Sample time:

-1

OK | Cancel | Help | Apply

On the **Channel Setup** tab:

**TPU Module**

Select TPU module A, B or C; each has 16 channels. Note that the MPC555 only has modules A and B. MPC565 and MPC566 also have module C. An error will be thrown if you select C and your target processor does not support this.

# TPU3 Rectangular Wave

**TPU channel number**
Choose 0-15

**Channel priority**
Choose Low, Medium, or High

The host CPU makes a channel active by assigning it one of the three priorities. You choose the order in which channels are serviced by setting the channel number and assigned priority. The order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest number first).

**Sample time**
The default is -1. This setting specifies that the block inherits its sample time from the block connected to its input (inheritance) (unless it is in a triggered subsystem). It makes no sense to sample faster than your input is changing, so normally you leave this at the default.

**Block Parameters: TPU3 Rectangular Wave**

MPC555 Rectangular Wave (TPU3) (mask) (link)

Configures a Time Processor Unit (TPU3) channel for Rectangular Wave Output (RECTW)

Outputs a Rectangular Wave with the specified high time, and specified wave period. Pulses always begin with a rising edge, and TCR1 is used as the time base.

When an input port is used to vary HIGH_TIME_RECTW and PERIOD_RECTW, the rest of parameters in the dialog (H,T or HIGH_TIME_RECTW,PERIOD_RECTW) will be used as initial values.

Channel Setup | Waveform Setup

☐ Edit period registers manually

Ideal high-time (H):

0.01

Waveform ideal period (T):

0.02

High-time register (HIGH_TIME_RECTW):

6250

Waveform period register (PERIOD_RECTW):

12500

Actual high-time:

0.01

Waveform actual period:

0.02

☐ Use input port to vary HIGH_TIME_RECTW and PERIOD_RECTW

| OK | Cancel | Help | Apply |

On the **Waveform Setup** tab:

**Edit period registers manually**
If you select this check box, you can manually set the **High-time register** and **Waveform period register** parameters.

**Ideal high-time (H)**
You can enter an ideal high-time period (in seconds). From this the high-time register is calculated and appears in the **High-time**

**register (HIGH_TIME_RECTW)** edit box. The actual waveform period is also calculated and displayed, see below.

**Waveform ideal period (T)**

Enter the waveform period you want by typing in this edit box. From this the waveform period register is calculated and appears in the **Waveform period register (PERIOD_RECTW)** edit box. The actual waveform period is also calculated and displayed, see below.

**High-time register (HIGH_TIME_RECTW)**

You can enter a value for the high-time register here ( 0<= x <= 32768 (0x8000)) only if you select **Edit period registers manually**. The actual high-time period is calculated and displayed in the actual high-time period field.

**Waveform period register (PERIOD_RECTW)**

You can enter a value for the period register here ( 0<= x <= 32768 (0x8000)) only if you select **Edit period registers manually**. The actual waveform period is calculated and displayed in the actual period field.

**Actual high-time**

Information field. You might find this information useful because actual and ideal high-time period are not always the same — the ideal period you enter may not always be possible.

**Waveform actual period**

Information field. You might find this information useful because actual and ideal waveform period are not always the same — the ideal period you enter may not always be possible.

**Use input port to vary HIGH_TIME_RECTW and PERIOD_RECTW**

Select this box to use input ports to control the high-time and waveform period registers. Two input ports appear on the block (the top input is high-time).

**Purpose**      Configure Time Processor Unit (TPU3) channel for Square Wave Output (SQW)

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library/ Time Processor Unit (TPU3)

**Description**

```
   SQW
  (TPU3)
```
TPU3 Square Wave
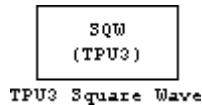
This block is provided as an example along with the demo model mpc555rt_tpu_emu. The square wave function is not part of the standard ROM mask of TPU functions but can be downloaded to DPTRAM and used by the TPU in emulation mode.

The TPU3 Square Wave block outputs a square wave with a specified high time (and corresponding low time). Pulses always begin with a rising edge, and TCR1 is used as the timebase.

You can either control the high-time register directly, or enter the desired (ideal) period and the mask will solve for the best values for the period register.

If you select the option **Use input port to vary HIGH_TIME_SQW**, an input appears. You can use this input to vary the high-time. The rest of the parameters in the mask are used as initial values. The input must be a 16 bit value in the range 0 <= x <= 32768 (0x8000).

The TPU Square Wave block uses TCR1 as a timebase for creating the output waveform. By changing the speed of the TCR1 clock, the range of available waveform periods changes. See "Time Processor Unit (TPU3) Configuration Parameters" on page 5-46 for more information on settings for the TCR1 clock.

Refer to Section 17, "Time Processor Unit 3," in the *MPC555 User's Manual* for further information.

# TPU3 Square Wave

**Dialog Box**



On the **Channel Setup** tab:

**TPU Module**

    Select TPU module A, B or C; each has 16 channels. Note that the MPC555 only has modules A and B. MPC565 and MPC566 also have module C. An error will be thrown if you select C and your target processor does not support this.

**TPU channel number**

    Choose 0-15

**Channel priority**

    Choose Low, Medium, or High

    The host CPU makes a channel active by assigning it one of the three priorities. You choose the order in which channels are serviced by setting the channel number and assigned priority. The

order in which channels are serviced is determined by assigned priority first, followed by channel number (lowest number first).

**Sample time**

The default is -1. This setting specifies that the block inherits its sample time from the block connected to its input (inheritance) (unless it is in a triggered subsystem). It makes no sense to sample faster than your input is changing, so normally you leave this at the default.



On the **Waveform Setup** tab:

Edit high-time register manually

If you select this check box, you can manually set the High-time register (HIGH_TIME_SQW) parameter.

**Ideal high-time (H)**

> You can enter an ideal high-time period (in seconds). From this the high-time register is calculated and appears in the High-time register (HIGH_TIME_SQW) edit box. The actual waveform frequency is also calculated and displayed, see below.

**High-time register (HIGH_TIME_SQW)**

> You can enter a value for the high-time register here ( 0<= x <= 32768 (0x8000)) only if you select **Edit high-time register manually**. The actual high-time period is calculated and displayed in the actual high-time field.

**Actual high-time**

> Information field. You might find this information useful because actual and ideal high-time period are not always the same — the ideal period you enter may not always be possible.

**Use input port to vary HIGH_TIME_SQW**

> Select this box to use an input port to control the high-time register. An input port appears on the block.

**Purpose**      In case of application failure, time out and reset processor

**Library**      Target for Freescale MPC5xx/ MPC555 Driver Library

**Description**   The Watchdog block lets you set the time-out period for the watchdog timer. The watchdog timer is a safety feature that is used to monitor correct behavior of the application. The timer is loaded with an initial value and counts down from this value. If the timer ever reaches zero, a watchdog time-out occurs, forcing a processor reset.

```
Watchdog
 Timer
Watchdog
```

In normal operation, the is serviced at a regular interval (each model step) by the application code; this occurs at a higher frequency than the **Watchdog Timeout** parameter period. Therefore the counter never reaches zero and a processor reset is never triggered.

In the event of a software failure that causes the application to lock up, the watchdog timer will not be serviced. Therefore, it will time out when the counter reaches zero. This in turn causes a processor reset, which restarts the application.

You do not need to include a Watchdog block in your model unless you want to change the **Watchdog Timeout** parameter period to a value other than the default. By default, the watchdog timer is enabled and the time-out period is set to the largest possible value, which is several seconds, depending on system frequency.

Note that the Watchdog block has neither input nor output connections.

# Watchdog

**Block Parameters: watchdog**

─ MPC555 Watchdog (mask) (link) ─────────────

Set the timeout period of the watchdog timer.

Normally the timer is reset every time the fastest sample period of the model executes. If the model gets locked and stops executing then the watchdog will timeout and reset the processor.

─ Parameters ─────────────────────────────

Watchdog Timeout [seconds]

|0.1|

| OK | Cancel | Help | Apply |
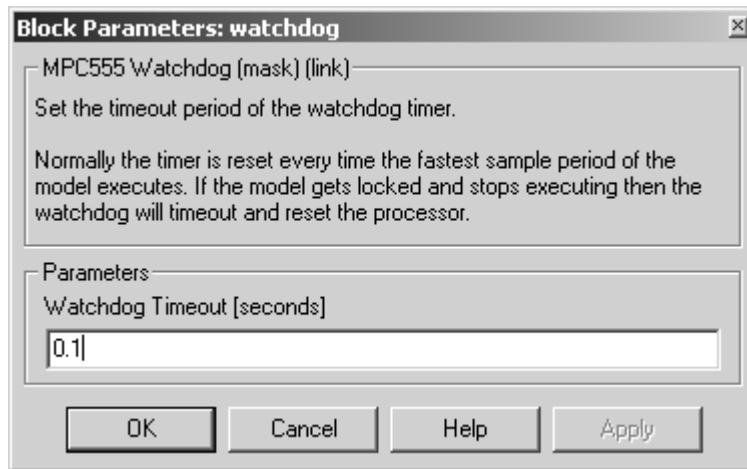
**Watchdog Timeout**

> The **Watchdog Timeout** period must be set to a value that is larger than the fastest sample rate in the system, because this is the rate at which the watchdog timer is serviced. To set the **Watchdog Timeout** period, place a Watchdog block anywhere in the model and open its dialog box.

# 6

# Configuration Parameters

# Real-Time Workshop Pane: ET MPC5xx (Algorithm Export) Options

| **In this section...** |
| --- |
| "ET MPC5xx (Algorithm Export) Options Tab Overview" on page 6-2 |
| "Use prebuilt (static) RTW Libraries" on page 6-4 |

## ET MPC5xx (Algorithm Export) Options Tab Overview
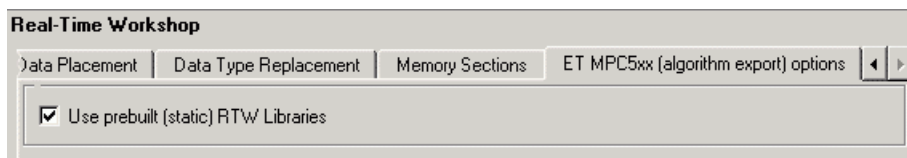


Control recompiling of libraries for faster build times.

### Configuration

This pane appears only if you specify the mpc555exp.tlc system target file.

### Tips

- The Algorithm Export (AE) target generates only the code that implements the algorithm of your model or subsystem. This is useful for code analysis and interfacing to hand-written or legacy code.

- Use the Target for Freescale MPC5xx HTML code generation report to view a profiling report that includes detailed itemization of RAM and ROM usage for all code and data sections, and a complete memory map of the generated code. You can also easily examine the generated code via hyperlinks in the code generation report.

  The **Code profile report** is an additional section in the Real-Time Workshop Embedded Coder HTML Code Analysis (RAM/ROM) Report. To generate the report, on the Real-Time Workshop General pane,

**a** Select **Generate HTML report**.

**b** Make sure **Generate code only** is not selected.

## See Also

- Algorithm Export Target
- HTML Code Analysis (RAM/ROM) Report

## Use prebuilt (static) RTW Libraries

Use prebuilt rtwlib for faster build time.

### Settings

**Default:** On

☑ On

Use prebuilt Real-Time Workshop libraries. This saves a considerable amount of time during the build process, as the libraries do not need to be recompiled every time.

☐ Off

Recompile libraries and do not use prebuilt libraries.

### Command-Line Information

**Parameter:** STATIC_RTWLIB
**Type:** string
**Value:** 'on' | 'off'
**Default:** 'on'

### See Also

Algorithm Export Code Generation Options

# Real-Time Workshop Pane: ET MPC5xx (Processor-in-the-Loop) Options

| In this section... |
| --- |
| "ET MPC5xx (processor-in-the-loop) Tab Overview" on page 6-5 |
| "Optimize compiler for" on page 6-6 |
| "Compiler optimization switches" on page 6-7 |
| "Build action" on page 6-8 |
| "Use prebuilt (static) RTW Libraries" on page 6-9 |

## ET MPC5xx (processor-in-the-loop) Tab Overview
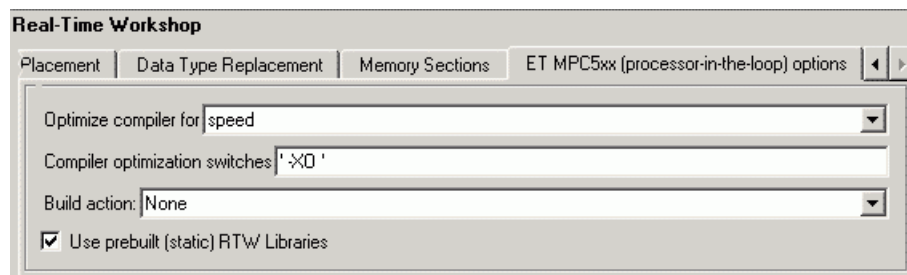


Specify compiler and build action code generation options for processor-in-the-loop.

### Configuration

This pane appears only if you specify the mpc555pil.tlc system target file.

### See Also

- Processor-in-the-Loop Code Generation Options
- Overview of PIL Cosimulation

## Optimize compiler for

Choose whether to optimize C compiler settings for fastest execution speed, smallest code size, debugging, or custom settings.

### Settings

**Default:** speed

speed

   Optimize C compiler settings to minimize execution time.

size

   Optimize C compiler settings to minimize code size.

debug

   Optimize C compiler settings for debugging.

custom

   Define your own optimization switches.

### Tip

The exact effect of the optimization switches depends on whether you are using the Wind River or CodeWarrior compiler. Consult your compiler documentation for specific optimizations.

### Dependency

Setting this parameter changes the **Compiler optimization switches** to the appropriate switches, which depend on the compiler used and may be user-defined.

### Command-Line Information

**Parameter:** MPC555_OPTIMIZATION_SWITCH
**Type:** string
**Value:** 'speed' | 'size' | 'debug' | 'custom'
**Default:** 'speed'

### See Also

Processor-in-the-Loop Code Generation Options

## Compiler optimization switches

Observe or edit compiler optimization switches.

### Settings

**Default:** Depends on the toolchain, and also customizable — there are many possibilities if you modify target preferences.

### Tip

To apply changes to the current model only, you can edit the switches in the Compiler optimization switches edit box (**Optimize compiler for**: changes to custom). If you want to apply these changes to several models, you can edit the defaults for these settings in the Target Preferences dialog box.

### Dependency

Editing this parameter changes the **Optimize compiler for** to custom.

### Command-Line Information

> **Parameter:** MPC555_OPTIMIZATION_FLAGS
> **Type:** string
> **Value:** customizable — there are many possibilities if you modify target preferences or make custom edits
> **Default:** depends on toolchain.

### See Also

Processor-in-the-Loop Code Generation Options

## Build action

Choose action to perform after build process completes.

### Settings

**Default:** None

None
> No action after code generation.

Launch_Download_Control_Panel
> Launch Download Control Panel utility on completion of code generation.

Run_via_BDM
> Download over BDM connection automatically starts on completion of code generation. When the download is complete the code is run.

Debug_via_BDM
> Download over BDM connection automatically starts on completion of code generation. When the download is complete the code stops at the first line in debug mode, so you can step through the code.

### Command-Line Information

**Parameter:** BuildAction
**Type:** string
**Value:** 'None' | 'Launch_Download_Control_Panel' | 'Run_via_BDM' | 'Debug_via_BDM'
**Default:** 'None'

### See Also

Processor-in-the-Loop Code Generation Options

## Use prebuilt (static) RTW Libraries

Use prebuilt rtwlib for faster build time.

### Settings

**Default:** On

☑ On

Use prebuilt Real-Time Workshop libraries. This saves a considerable amount of time during the build process, as the libraries do not need to be recompiled every time.

☐ Off

Recompile libraries and do not use prebuilt libraries.

### Command-Line Information

**Parameter:** STATIC_RTWLIB
**Type:** string
**Value:** 'on' | 'off'
**Default:** 'on'

### See Also

Processor-in-the-Loop Code Generation Options

# Real-Time Workshop Pane: ET MPC5xx Real-Time Options (1)

## ET MPC5xx Real-Time Options (1) Tab Overview
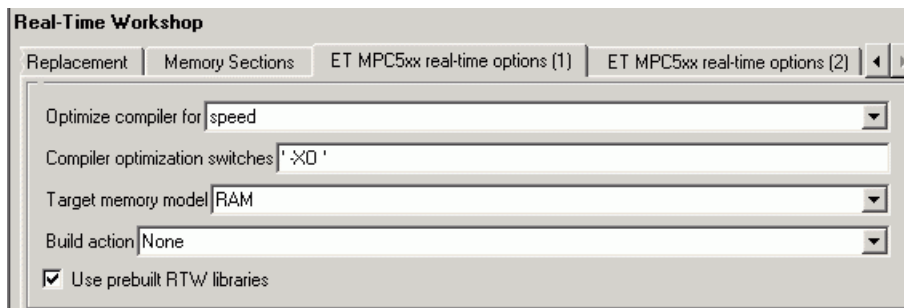


Specify compiler and build action code generation options for real-time standalone execution.

### Configuration

This pane appears only if you specify the mpc555rt.tlc or mpc555rt_grt.tlc system target file.

### See Also

- Real-Time Code Generation Options

- Generating Stand-Alone Real-Time Applications

# Optimize compiler for

Choose whether to optimize C compiler settings for fastest execution speed, smallest code size, debugging, or custom settings.

### Settings
**Default:** speed

speed

    Optimize C compiler settings to minimize execution time.

size

    Optimize C compiler settings to minimize code size.

debug

    Optimize C compiler settings for debugging.

custom

    Define your own optimization switches.

### Tip
The exact effect of the optimization switches depends on whether you are using the Wind River or CodeWarrior compiler. Consult your compiler documentation for specific optimizations.

### Dependency
Setting this parameter changes the **Compiler optimization switches** to the appropriate switches, which depend on the compiler used and may be user-defined.

### Command-Line Information

    **Parameter:** MPC555_OPTIMIZATION_SWITCH
    **Type:** string
    **Value:** 'speed' | 'size' | 'debug' | 'custom'
    **Default:** 'speed'

### See Also

Real-Time Code Generation Options

## Compiler optimization switches

Observe or edit compiler optimization switches.

### Settings

**Default:** Depends on the toolchain, and also customizable — there are many possibilities if you modify target preferences.

### Tip

To apply changes to the current model only, you can edit the switches in the Compiler optimization switches edit box (**Optimize compiler for**: changes to custom). If you want to apply these changes to several models, you can edit the defaults for these settings in the Target Preferences dialog box.

### Dependency

Editing this parameter changes the **Optimize compiler for** to custom.

### Command-Line Information

**Parameter:** MPC555_OPTIMIZATION_FLAGS
**Type:** string
**Value:** customizable — there are many possibilities if you modify target preferences or make custom edits
**Default:** depends on toolchain.

### See Also

Real-Time Code Generation Options

## Target Memory Model

Select either FLASH or RAM.

### Settings

**Default:** RAM

RAM
> Generate files in a format suitable for downloading into external RAM.

FLASH
> Generate files in a format suitable for downloading into the MPC555 on-chip flash memory.

In both cases these two files are generated, with this naming convention:

- *model*_flash.s19 or *model*_ram.s19 — code only, for CAN download

- *model*_flash.elf or *model*_ram.elf — for BDM download, containing code and optional debugging symbols if you choose a debug build in the **Optimize compiler for** settings.

### Tips

- Loading the application code into RAM is faster than loading it into flash memory. In addition, by using RAM you can avoid using up the programming cycles of the flash memory; this lengthens the usable lifetime of the flash memory. Running the application from RAM is a good option for initial testing of the application.

- The MPC5xx flash memory has a limited lifetime, which is shortened each time the flash memory is programmed. To extend product life, Freescale recommends using flash programming only when necessary.

- To program applications into RAM, your target hardware must have additional RAM external to the MPC555 on-chip RAM. Target for Freescale MPC5xx does not support downloading of code to MPC5xx on-chip RAM, because the MPC555 has only 26K of on-chip RAM and the MPC565 has 36K.

- For final deployment, or to load code onto a test board for use at a test site, you will generally want to program your code into the nonvolatile flash memory. 416K of flash memory is available for application code (992K on the MPC565). Code programmed into flash memory is persistent and restarts when the board is powered on.

### Command-Line Information

**Parameter:** TARGET_MEMORY_MODEL
**Type:** string
**Value:** 'RAM' | 'FLASH'
**Default:** 'RAM'

### See Also

- RAM vs. Flash Memory
- Overview of Memory Organization and the Boot Process

# Build action

Choose action to perform after build process completes.

### Settings

**Default:** None

None
> No action after code generation.

Launch_Download_Control_Panel
> Launch Download Control Panel utility on completion of code generation.

Run_via_BDM
> Download over BDM connection automatically starts on completion of code generation. When the download is complete the code is run.

Debug_via_BDM
> Download over BDM connection automatically starts on completion of code generation. When the download is complete the code stops at the first line in debug mode, so you can step through the code.

### Command-Line Information

**Parameter:** BuildAction
**Type:** string
**Value:** 'None' | 'Launch_Download_Control_Panel' | 'Run_via_BDM' | 'Debug_via_BDM'
**Default:** 'None'

### See Also

Real-Time Code Generation Options

## Use prebuilt RTW libraries

Use prebuilt `rtwlib` for faster build time.

### Settings

**Default:** On

☑ On

Prebuilt Real-Time Workshop libraries, compiled with default compiler switches, are linked against during compilation of the generated code. This saves a considerable amount of time during the build process, as the libraries do not need to be recompiled every time.

☐ Off

The source modules that comprise these libraries are compiled individually in the model build directory, using the currently selected compiler switches.

### Command-Line Information

**Parameter:** `STATIC_RTWLIB`
**Type:** string
**Value:** `'on'` | `'off'`
**Default:** `'on'`

### See Also

Real-Time Code Generation Options

# Real-Time Workshop Pane: ET MPC5xx Real-Time Options (2)

| In this section... |
| --- |
| "ET MPC5xx Real-Time Options (2) Tab Overview" on page 6-17 |
| "Maximum number of concurrent base-rate overruns:" on page 6-17 |
| "Maximum number of concurrent sub-rate overruns:" on page 6-18 |
| "Execution profiling" on page 6-20 |
| "Number of data points:" on page 6-20 |

## ET MPC5xx Real-Time Options (2) Tab Overview
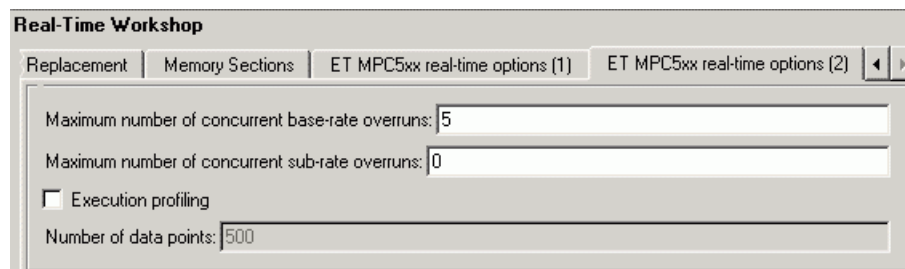


Control execution profiling and scheduling overrun behavior.

### Configuration

This pane appears only if you specify the `mpc555rt.tlc` or `mpc555rt_grt.tlc` system target file.

### See Also

- Real-Time Workshop Options for Execution Profiling
- Execution Profiling

## Maximum number of concurrent base-rate overruns:

Configure allowable base-rate overruns.

### Settings
**Default:** 5

**Minimum:** 0

**Maximum:** No maximum value — it depends on available memory.

### Tips

• Use this option to configure the behavior of the scheduler when timer based tasks do not complete within their allowed sample time.

• It is useful to allow task overruns in the case where a task may occasionally take longer than usual to complete (e.g. if extra processing is required when a special event occurs); if the task overrun is only occasional then it is possible for the scheduler to 'catch up' after the extra processing has been completed.

• If the maximum number of concurrent overruns for any task is exceeded, this is deemed to be a failure and the real-time application is stopped. This in turn will result in a watchdog timer timeout and the processor will be reset.

• The occurrence of base-rate overruns does not affect the numerical behavior of the algorithm (although reading/writing external devices will of course be delayed).

### Command-Line Information

**Parameter:** BaseRateMaxOverrunsValue
**Type:** int
**Value:** 0 | 1 | 2...
**Default:** 5

### See Also
Real-Time Workshop Options for Execution Profiling

## Maximum number of concurrent sub-rate overruns:
Configure allowable sub-rate overruns.

### Settings

**Default:** 0

**Minimum:** 0

**Maximum:** No maximum value — it depends on available memory.

### Tips

- If this option is set to a value greater than zero, then the behavior of any Rate-Transition blocks may be affected. Specifically, if the model contains a Rate Transition block where the option "Ensure deterministic data transfer (maximum delay)" is selected, then this setting may not be honored.

- If sub-rate overruns are allowed then the transfer of data between different rates (via rate-transition blocks) in the model may be affected; this causes the numerical behavior in real-time to differ from the behavior in simulation. To see an illustration of this effect try running the demo model `mpc555rt_multitasking`. To disallow sub-rate overruns and ensure that this effect does not occur, you should set **Maximum number of concurrent sub-rate overruns** to zero.

### Command-Line Information

**Parameter:** `SubRateMaxOverrunsValue`
**Type:** int
**Value:** 0 | 1 | 2...
**Default:** 0

### See Also

Real-Time Workshop Options for Execution Profiling

## Execution profiling

Specify whether to configure code for execution profiling.

### Settings
**Default:** Off

☑ On

Include function calls in the generated code for the model at the beginning and end of each task or asynchronous Interrupt Service Routine (ISR) to be profiled. When you perform an execution profiling run, these function calls read a timer and log this reading, along with a task identifier, for uploading and analyzing.

☐ Off

Do not add function calls for execution profiling.

### Tip
When code for the model is generated, these function calls update data on the worst-case turnaround time for each timer-based task as well as the worst-case number of concurrent task overruns, whenever a previous worst case value is exceeded. Additionally, when a trigger is provided, data can be logged over a period of time to record all task start and task finish times. The trigger signal can be supplied by the execution profiling blocks.

### Dependency
This parameter enables **Number of data points**.

### See Also

- Execution Profiling
- Real-Time Workshop Options for Execution Profiling

## Number of data points:

Specify number of data points to log for execution profiling runs.

### Settings
**Default:** 500

**Minimum:** This depends on the number of tasks. Three is a sensible minimum to get useful information back.

**Maximum:** No maximum value - it depends on available memory.

### Tip
When a snapshot of task and ISR activity is logged this data is stored in memory that is statically allocated at build time. Each data point requires 8 bytes on the MPC555. The larger the number of data points to be stored, the more RAM that must be reserved for this purpose. At the end of a logging run, the data must be uploaded to the host computer for analysis; this is typically achieved by using the execution profiling blocks.

### Dependency
This parameter is enabled by **Execution Profiling**.

### Command-Line Information

**Parameter:** ExecutionProfilingNumSamples
**Type:** int
**Value:** 3 | 4 | 5...
**Default:** 500

### See Also

- Real-Time Workshop Options for Execution Profiling
- Execution Profiling

# A

# Toolchains and Hardware

This section discusses specific settings for different cross-development environments:

# Setting Up Your Toolchain

The currently supported toolchains are WindRiver (Wind River Compiler and SingleStep) and Freescale CodeWarrior. You must first install and configure your toolchain to work with Target for Freescale MPC5xx. The necessary steps are described in the following sections:

- "Setting Up Your Installation with Wind River Compiler and SingleStep Debugger" on page A-4

- "Setting Up Your Installation with Freescale CodeWarrior" on page A-10

**A-3**

# Setting Up Your Installation with Wind River Compiler and SingleStep Debugger

| **In this section...** |
| --- |
| "Required Hardware and Software" on page A-4 |
| "Procedure" on page A-4 |

## Required Hardware and Software

To use Target for Freescale MPC5xx with the Wind River Compiler, you need the following:

- An MPC5xx development board (such as the phyCORE-MPC555 development board, or an Axiom board) and a debugger connector (such as the WindRiver visionPROBE or the BDM Wiggler from Macraigor Systems). Note the phyCORE-MPC555 board comes with built-in debugger connector into which you can directly plug a parallel port connector, in which case you may not require a BDM connector. See "Setting Up Your Target Hardware" on page A-14.

- Wind River Systems Wind River Compiler (version 5.4.0 )

- Wind River Systems SingleStep debugger

  - Version 7.7.5 (debug via visionPROBE BDM connector only)

  - Version 7.6.6 (MPC555 only) (debug via Wiggler, Raven / Blackbird, On-board BDM), plus the Fromelf Utility - Windows, from Wind River Support Online.

    You must download fromelf.exe for the SingleStep debugger 7.6.6, otherwise builds with debug flag -g set will not load, with the following error: "aborting due to failure of ELF reader".

    Note to use these BDM devices you must set up nondefault target preferences, as detailed below.

## Procedure

- "Install Wind River Compiler" on page A-5

- "Install SingleStep Debugger" on page A-5
- "Setting Target Preferences for Wind River Compiler and SingleStep" on page A-6
- "Initialize visionPROBE" on page A-8
- "Configure MPC5xx Jumpers" on page A-8

## Install Wind River Compiler

If you have not already done so, install the Wind River Compiler, following the installation instructions provided by Wind River Systems.

You do not need to set a default processor or other compiler defaults. During the code generation and build process, Target for Freescale MPC5xx will generate a makefile that sets the correct options.

You will need to note the path to the installed compiler in order to configure your target preferences (see "Setting Target Preferences for Wind River Compiler and SingleStep" on page A-6).

## Install SingleStep Debugger

The SingleStep debugger, in conjunction with Target for Freescale MPC5xx, lets you download, run and debug generated code.

Follow the instructions of the SingleStep installer.

To resolve questions or difficulties with Single Step, refer to the SingleStep documentation, or contact Wind River Systems.

You will need to note the path to the installed SingleStep debugger in order to configure your target preferences (see "Setting Target Preferences for Wind River Compiler and SingleStep" on page A-6).

### Setting Target Preferences for Wind River Compiler and SingleStep

After installing your development tools, the next step is to configure your target preferences for the Wind River Compiler and SingleStep debugger. (Please read "Setting Target Preferences" on page 1-17, if you have not yet done so.)

**1** Select **Start > Links and Targets > Target for Freescale MPC5xx > MPC5xx Target Preferences**.

This opens the Target Preferences GUI where you can edit the settings for your cross-development environment.

**2** Select Diab from the **Toolchain** menu (the Wind River Compiler was formerly known as Diab).

**3** Expand the ToolChainOptions by clicking the plus sign, and type the correct path into **CompilerPath**. For example "d:\applications\WindRiver\4.3g".

**4** For SingleStep you must also type the correct path into **DebuggerPath**. For example "d:\applications\sds".

**5** The defaults for **DebuggerSwitches** and **DebuggerExecutable** are set up for use of SingleStep (using a visionPROBE BDM connection). You may need to change LPT1 to whatever port you connect to. Note, once you have set target preferences, you must initialize the device. See "Initialize visionPROBE" on page A-8.

**6** To use any other BDM device than the visionPROBE (such as the Wiggler, Raven/Blackbird or OnBoard BDM with SingleStep), you must change two target preferences from the defaults:

   **a** Change the **DebuggerSwitches** target preference to the following:

   ```
   -g -V mpc555 -r - -p LPT1=1
   ```

   If necessary you can change `LPT1` to whatever port you connect the probe to.

   **b** Change the **DebuggerExecutable** from the default to:

   ```
   bdmp58.exe
   ```



The **DebuggerSwitches** target preference is specific to SingleStep. If you want to change the default debug settings, type

```
help debug
```

at the SingleStep command line to see the options available. For example you can change parallel port here. The default is `-p LPT1=1` which specifies port 1 on your host PC at speed 1. You could change it to `-p LPT2=2` to specify port 2 at speed 2.

Other debugger executables are supplied with SingleStep — if you want to change the defaults to use a different connection device and different debug settings, consult the SingleStep documentation.

Note that the path to the SingleStep debugger, specified in `DebuggerPath` in the Target Preference GUI, is the root directory of your SingleStep

installation, on either an actual hard drive on your PC, or a mapped drive. Do not use a Universal Naming Convention (UNC) path. For most purposes, the other target preferences fields can be left at their defaults. Once you have set these target preferences, the build process will automatically invoke your compiler and debugger when required for downloading code.

### Initialize visionPROBE

Before using the visionPROBE (and after setting target preferences) you must initialize the device using the MATLAB **Start** menu. Select **Start > Links and Targets > Target for Freescale MPC5xx > Initialize visionPROBE for Selected Target Board (WindRiver Only)**.

If you change target processor you will have to initialize again using the same **Start** menu item.

Note that for the visionPROBE you must configure the parallel port BIOS settings as follows:

- ECP mode

- Enabled (as opposed to Auto)

- IRQ and address of the parallel port specified in the BIOS must match that in the visionPROBE comdll.cfg file - edit the cfg file if necessary. Default parallel port I/O address = 0x378; IRQ=7, communicating over PAR1 (LPT1).

- If using Windows 2000, you must also select Control Panel > System Device Manager > ECP Port > Properties > Port Settings and make sure it is set to "Use any interrupt assigned to the port".

### Configure MPC5xx Jumpers

Make sure that the jumpers on the MPC5xx board are set as described in "Jumper Settings" on page A-14. The correct jumper configuration is required when downloading to flash memory. Any other jumper settings may cause downloading to flash memory to fail, or cause other problems when operating with Target for Freescale MPC5xx. For additional information on jumper settings, consult the MPC5xx documentation and the SingleStep manual.

The next step is to verify your installation:

**1** You can download and run the test program supplied. See "Run Test Program" on page 1-23.

**2** You must then follow the instructions to download boot code ("Download Boot Code to Flash Memory" on page 1-24). Once you have completed these steps, you can begin working with Target for Freescale MPC5xx.

# Setting Up Your Installation with Freescale CodeWarrior

| **In this section...** |
| --- |
| "Required Hardware and Software" on page A-10 |
| "Procedure" on page A-10 |

## Required Hardware and Software

To use Target for Freescale MPC5xx with Freescale CodeWarrior, you need the following:

- An MPC5xx development board (such as the phyCORE-MPC555 development board) and a debugger connector (such as the BDM Wiggler from Macraigor Systems). Note the phyCORE-MPC555 board comes with built-in debugger connector which you can plug a parallel port connector into directly, in which case you may not require a BDM connector.

- Freescale CodeWarrior Development Studio, MPC5xx Edition, v8.7 CD (or network access to the Freescale CodeWarrior Development Studio installer).

## Procedure

- "Install Freescale CodeWarrior IDE" on page A-10
- "Configure Freescale CodeWarrior Debugger" on page A-11
- "Set Target Preferences for CodeWarrior" on page A-12
- "Configure MPC5xx Jumpers" on page A-12

### Install Freescale CodeWarrior IDE

The first step is to install the Freescale CodeWarrior IDE:

**1** If you have previously installed an older version of Freescale CodeWarrior for Embedded PowerPC , uninstall it.

**2** Install Freescale CodeWarrior Development Studio, MPC5xx Edition, v8.7 using the setup program provided on your Freescale CodeWarrior CD (or on your network). Run `Setup.exe` and follow the prompts.

**3** Open CodeWarrior IDE. You can use the Windows Start menu (**Start > Programs > CodeWarrior > CodeWarrior IDE**).

**4** Select **Edit > Preferences > Build Settings > Build Before Running**

**5** Select the option **Never** and click **Apply**.

It is vital you set this to avoid errors when building and automatically downloading code with Target for Freescale MPC5xx.

### Configure Freescale CodeWarrior Debugger

The next step is to configure the CodeWarrior debugger to communicate with the MPC5xx board over the parallel port:

**1** From the Freescale CodeWarrior IDE, select the **Edit** menu, and open the I**DE Preferences** dialog box. In the **IDE Preference Panels** pane, click on the plus sign next to **Debugger**.

**2** A list of choices opens below **Debugger**. Select `Remote Connections`. The **Remote Connections** panel is displayed on the right.

**3** If no `MPC555DK Wiggler` configuration exists, create one as follows:

**a** Click the **Add**... button. The **New Connection** configuration dialog box opens. Set the **Name** property to `MPC555DK Wiggler`.

**b** If you are using a Raven or Blackbird BDM device, set the **Debugger** property to `EPPC - MSI BDM Raven`.

**c** If you are using a Wiggler or On-Board BDM, set the **Debugger** property to `EPPC MSI Wiggler`.

**d** Set the **Connection Type** property to `Parallel`.

**e** Set the **Connection Port** property to match the port to which you have connected your MPC5xx board (the default is `LPT1`).

**f** Set the **Speed** property to `1`.

**g** Set the **FPU Buffer Address** property to 0x3f9800.

**A-11**

    **h** Click **OK** and skip to step 5.

**4** If a `MPC555DK Wiggler` exists, click the **Change** button. The `MPC555DK Wiggler` configuration dialog box opens. By default, the **Parallel Port** property is set to `LPT1`. If you have connected your MPC5xx board to a different port, change the **Parallel Port** setting accordingly. Then click **OK** to close the **MPC555DK Wiggler** configuration dialog box.

**5** Click **Apply** and close the **IDE Preferences** dialog box.

### Set Target Preferences for CodeWarrior

The next step is to configure your target preferences for Freescale CodeWarrior. (Please read "Setting Target Preferences" on page 1-17, if you have not yet done so). Follow these steps:

**1** Select **Start > Links and Targets > Target for Freescale MPC5xx > MPC5xx Target Preferences**

    This opens the Target Preferences GUI where you can edit the settings for your cross-development environment.

**2** Select `CodeWarrior` from the **Toolchain** menu.

**3** Expand the `ToolChainOptions` by clicking the plus sign, and type the correct path into **CompilerPath**. Do not use a UNC path, use only local or mapped drives.

Note that when using CodeWarrior, you do not also have to specify the DebuggerPath, as the compiler and debugger are integrated. When required, the build process will automatically invoke the CodeWarrior debugger.

For most purposes, the other target preferences fields can be left at their defaults.

### Configure MPC5xx Jumpers

Make sure that the jumpers on the MPC5xx board are set as described in "Jumper Settings" on page A-14. The correct jumper configuration is required.

The next step is to verify your installation.

**1** You can download and run the test program supplied. See "Run Test Program" on page 1-23.

**2** You must then follow the instructions to download boot code ("Download Boot Code to Flash Memory" on page 1-24). Once you have completed these steps, you can begin working with Target for Freescale MPC5xx.

# Setting Up Your Target Hardware

| **In this section...** |
| --- |
| "Communications Ports" on page A-14 |
| "Jumper Settings" on page A-14 |

This section describes the required connections and jumper settings for the following development boards: Phytec phyCORE-MPC555, the Phytec MPC565 and the Axiom MPC555, MPC564, and MPC566.

If you are using other development boards you may need to see "Configuration for Nondefault Hardware" on page A-23.

## Communications Ports

Before you begin working with Target for Freescale MPC5xx, you should set up your target board and connect it to your host computer. For example, the hardware setup is described in the *phyCORE-MPC555 Quickstart Instructions* manual on the Phytec Spectrum CD. See the "Interfacing the phyCORE-MPC555 to a Host PC" section of the "Getting Started" chapter.

In this document, we assume that you have connected your board to the same serial (COM1) and parallel (LPT1) ports described in the *phyCORE-MPC555 Quickstart Instructions*. Note that you must ensure your computer's LPT parallel port for BDM interface is set to EPP mode and Auto (as opposed to Enabled). This is generally a BIOS level configuration. If you are using a visionPROBE you must configure the parallel port as detailed in "Initialize visionPROBE" on page A-8.

## Jumper Settings

**Note** You MUST check your jumper settings. Do not assume hardware is supplied with jumpers set as documented by the manufacturer. Incorrect operation or even hardware damage may occur if you do not check jumper settings.

Use the following settings for these boards:

### Phytec MPC555 Jumper Settings

Target for Freescale MPC5xx has been tested by the MathWorks with the Phytec phyCORE-MPC555 board, using the jumper settings indicated in the table below. We have tested the PCB 1174.0 board. If you are using a PCB 1174.2 board you may also have to alter settings such as jumper 19. Please see your Phytec development board manual for details.

For jumper locations and pin numbers, see Jumper Layout section of the *Development Board for phyCORE-MPC555 Hardware Manual*, `"L-525E.pdf"` found at

http://www.phytec.de/manuals

The following table summarizes the correct jumper settings to use when your host PC is connected to the on-board BDM port, or via visionPROBE, Wiggler, Raven, or Blackbird devices.

| Jumper | Description | visionPROBE, Raven or Blackbird | Wiggler | On-Board BDM |
|--------|-------------|--------------------------------|---------|--------------|
| JP1 | On-board BDM reset signal connection | Open | as Raven | 3+4 closed |
| JP2 | Power supply for external BDM | Open (unless BDM device requires supply voltage from development board) | 1+2 closed | 1+2 closed |

| Jumper | Description | visionPROBE, Raven or Blackbird | Wiggler | On-Board BDM |
|---|---|---|---|---|
| JP3 | Connect push button to different reset signals | 1+2 (/HRESIN connected to push button) | as Raven | as Raven |
| JP4 | Programming of Internal MPC555 Flash internal memory enabled | Closed | as Raven | as Raven |
| JP5,JP7, JP8,JP9 | Jumpers relating to on-board BDM | Open | as Raven | All closed |
| JP6 | See note below. | Open | as Raven | Closed |
| JP10 | Connect one of the LEDs to supply voltage | Closed | as Raven | as Raven |
| JP11 | Connect 5V supply voltage | Closed | as Raven | as Raven |
| JP12 | Connect 3V3 supply voltage | Closed | as Raven | as Raven |
| JP13 | CAN A bus termination | Closed (apply 120 Ohm termination) | as Raven | as Raven |
| JP14 | CAN B bus termination | Closed (apply 120 Ohm termination) | as Raven | as Raven |
| JP15 | Select boot memory | 1+2 (boot from internal flash memory) | as Raven | as Raven |
| JP16 | Use J5 as source of Hard-Reset-Configuration | Open | as Raven | as Raven |

| Jumper | Description | visionPROBE, Raven or Blackbird | Wiggler | On-Board BDM |
|--------|-------------|---------------------------------|---------|--------------|
| JP17 | Connect /HRESET or /SRESET to external BDM interface logic | 1+2 (/HRESET connected to BDM interface logic) | as Raven | as Raven |
| JP18 | Connect interrupt to push button | Default 1+2 | as Raven | as Raven |
| JP19 | See note below. | | | |

**Note** Jumper 6 must be open unless using an On-Board BDM.

When using the On-Board BDM connection, if you then want to run the target stand-alone (disconnected from the debugger) you must also disconnect (open) jumper 6. This only affects the on-Board BDM, all other configurations *always have jumper 6 open*. Use the On-Board BDM settings in the table if you are using the BDM connection for debugging, but remember you must make this change to run stand-alone:

For debugging: Jumper 6 must be closed (target stops always in debug mode after reset); connect parallel cable to target.

For stand-alone: Jumper 6 must be open (target runs in normal mode after reset); disconnect parallel cable from target.

**Note** The jumper 19 setting may need to be altered if you are using a PCB 1174.2 board. See the Phytec documentation for more information.

### Phytec MPC565 Jumper Settings
These settings are for EXTERNAL BDM device only, NOT On-Board BDM.

Make sure you use the default Phytec documented MPC565 jumper settings and the following additional changes :

**General**:

JP28: 2-3

JP29: Closed

**BDM Related**:

JP32: Open

JP33: Open

JP34: Open

JP35: Open

JP36: Open

JP37: Open

JP38: Open

Additional warning - JP20:

Closing JP20 on the Phytec MPC565 development board connects the MPC565 MDA27 to the ZZ - "Snooze Enable of the burst-RAM". If you wish to use MDA27 on the MPC565 development board then JP20 must be left open.

Please either do not use module 27 with the MIOS Waveform Measurement block or open JP20 on your development board.

*Jumper 33* warning: When using the onboard BDM and the download control panel of Target for Freescale MPC5xx, you might see the download timeout as the target is halted, if Jumper 33 is not correctly set. Perform these debugging steps:

- If you download code for debugging: Connect parallel cable to target and make sure Jumper 33 is closed (target stops always in debug mode after reset).

- If you download code for stand-alone execution: Disconnect the parallel cable from the target and make sure Jumper 33 is open (target runs in normal mode after reset).

- If you cannot download code, but can debug:

  **c** Check if you are using the onboard BDM.

  **d** Check the setting of Jumper 33 and state of parallel cable, as stated above.

  **e** If this does not resolve the issue, check the other jumper settings.

### Axiom MPC555 Jumper Settings
These jumper settings work with an external BDM device.

Make sure you use the default Axiom documented jumper settings and the following additional changes:

| Config Switch | Mode Switch 1 | Mode Switch 2 | Other |
|---|---|---|---|
| 1 : Off | 1 : Off | 1 : On | M-SEL Jumper - Open |
| 2 : On | 2 : Off | 2 : Off | FLSH-SEL Jumper - Open |
| 3 : Off | 3 : Off | 3 : Off | RAM-SEL Jumper - 2 Closed |
| 4 : On | 4 : Off | 4 : Off | MEM-OPT Jumper - 5, 7 |
| 5 : On | 5 : Off | 5 : Off | Closed |
| 6 : On | 6 : Off | 6 : Off | |
| | 7 : Off | 7 : Off | |
| | 8 : Off | 8 : Off | |

### Axiom MPC564EVB Jumper Settings
These settings work with an external BDM device.

Make sure you use the default Axiom documented jumper settings and the following additional changes:

| MAP_SW | CONFIG_SW |
|--------|-----------|
| 1: off | 1: on |
| 2: on | 2: off |
| 3: on | 3: on |
| 4: off | 4: off |
| 5: on | 5: off |
| 6: on | 6: off |
| 7: on | 7: on |
| 8: on | 8: on |

### Axiom MPC566EVB Jumper Settings

Make sure you use the default Axiom documented jumper settings and the following additional changes:

| MAP_SW | CONFIG_SW |
|--------|-----------|
| 5: off | 7: on |
| 8: on | 8: on |

CS0 > Ext_Flash

CS1 > Ext_SRAM

IP bit off (execute from 0x0000_0100 on reset)

Internal chip Flash enabled

Check the oscillator frequency Target Preference is configured correctly to 4MHz for the 566 board (check the board manual).

# CAN Hardware and Drivers

**In this section...**

## Configuring CAN Channels

Similarly to the Vector CAN blocks, the Download Control Panel is based on the Vector CAN Driver Programming Library. The Download Control Panel uses the Application Channel mechanism used by the Vector CAN Blocks.

You can use the **CAN Driver Configuration Tool** from Vector to select a CAN channel (installed CAN hardware or a virtual CAN channel) and set the speed of the connection. You can access this tool by clicking **Configure** on the Communication Options tab of the Download Control Panel. Also this tool is opened automatically when you open the Vector CAN Configuration block if you have installed Vector drivers.

## Creating and Assigning Application Channels

**1** Open the Download Control Panel by selecting **Start > Links and Targets > Target for Freescale MPC5xx > Download RAM / FLASH Based Application (via CAN / Serial)**.

**2** On the **Communications Options** tab, select CAN from the **Connection type** drop-down menu.

**3** Select from the drop-down menu one of the MATLAB application channels (1-10).

Use the Vector CAN Driver Configuration Tool to create and assign the selected MATLAB application channel to the required CAN hardware device or virtual channel as follows.

**4** Click **Configure** on the Download Control Panel to open the Vector CAN Driver Configuration Tool.

**5** Click **App. Settings** in the Vector CAN Driver Configuration Tool.

**6** Click **Add** in the **Application Settings** dialog that appears.

**7** Enter MATLAB in the edit box for the new application name and click **OK**.

**8** Click **Done** to leave the **Application Settings** dialog.

**9** Click to select the CAN hardware device or virtual channel you want to use (for example, Channel 1 of a CAN-AC2-PCI card).

**10** Click **Assign to application** (or right-click on the required channel).

**11** Select MATLAB 1 or MATLAB 2 from the list.

Make sure you select the same MATLAB application channel in the Vector CAN Configuration block. If your model requires more than one application channel take care to assign a different channel to each Vector CAN Configuration block.

See the Vector Help for the CAN Driver Configuration Tool to find out more about how to select the CAN channel, bit rate, synchronization jump width, sample point and number of samples per bit.

Please refer also to the following section of the Vector CAN Blocks documentation in order that you configure your hardware and software drivers correctly:

CAN Blockset Reference Vector CAN Configuration.

# Configuration for Nondefault Hardware

| **In this section...** |
| --- |
| "Hardware Clock Configuration" on page A-23 |
| "Other Configuration Changes for Nondefault Hardware" on page A-25 |

Target for Freescale MPC5xx has been developed and fully tested using the development boards described in "Setting Up Your Target Hardware" on page A-14. We recommend the use of these boards for getting started with Target for Freescale MPC5xx. If you are using different MPC5xx hardware, it may be necessary to perform some additional manual configuration.

The following sections provide information about where to make changes for hardware clock configuration and other hardware-specific configurations.

## Hardware Clock Configuration

Target for Freescale MPC5xx uses the Periodic Interrupt Timer (PIT) to support a range of sample times. Note that the PIT is driven by the crystal frequency. This results in the following possible sample time ranges:

For a crystal frequency of 20Mhz:

- Fastest sample time = 1.28e-5 seconds.
- Slowest sample time = 0.8388 s.

For a crystal frequency of 4 MHz:

- Fastest sample time = 6.4e-5 s.
- Slowest sample time = 4.1942 s.

Note that if you select a sample time slower than the slowest possible for your clock frequency, Simulink issues a warning message.

Also note that the fastest sample time may not be achievable because timer overruns may occur, depending on your model.

Target for Freescale MPC5xx uses the main system oscillator (OSCM) to provide the system clock. The OSCM uses either a 4-MHz or 20-MHz crystal to generate the PLL reference clock. The next section describes how to configure the Target for Freescale MPC5xx real-time target for use on hardware with 4MHz crystal frequency (the default is 20 MHz).

---

**Note** External clock inputs are not supported.

---

### Configuring Target for Freescale MPC5xx for a Crystal Frequency Other Than 20 MHz

The MPC555 can operate with a crystal frequency of either 4 MHz or 20 MHz. By default, Target for Freescale MPC5xx is configured for a crystal frequency of 20 MHz.

You can use the Target Preferences to change to a 4MHz oscillator frequency.

**1** Use the **Start** button to open the Target Preferences: **Start > Links and Targets > Target for Freescale MPC5xx > MPC5xx Target Preferences**.

**2** Use the drop-down menu for OscillatorFrequency to change from 20 (the default) to 4.

**3** Now install the appropriate bootcode for your hardware. Select **Start > Links and Targets > Target for Freescale MPC5xx > Install MPC5xx Bootcode**.

The correct bootcode is installed for the oscillator frequency and processor variant that you have selected in the Target Preferences. See the Target Preferences section "Target Board" on page 1-21.

Note that you must also change the oscillator frequency and processor variant in your models. Use the Resource Configuration block. The oscillator frequency and processor set here must match the Target Preferences, or you will see warnings.

The default value for `Oscillator_Frequency` is 20. If you are using 4MHz hardware, you must change the value for `Oscillator_Frequency` to 4 in every model.

See also System Clock and Related Parameters System Clock and Related Parameters for information on changing the system clock speed, and the block Switch Target Configuration to easily switch between a selection of preset target configurations with different processors and system frequencies.

## Other Configuration Changes for Nondefault Hardware

Depending on your target hardware, it may be necessary to make changes to configure settings such as the size and type of external memory.

If you are downloading using the Freescale CodeWarrior development environment, the relevant hardware configuration settings are contained in *matlabroot*\toolbox\rtw\targets\mpc555dk\mpc555dk\:

```
@codewarrior_tgtaction\mpc5xx_osc20.cfg
@codewarrior_tgtaction\mpc5xx_osc4.cfg
```

If you are downloading using the Wind River Compiler and SingleStep development environment, the configuration settings are contained in *matlabroot*\toolbox\rtw\targets\mpc555dk\mpc555dk\:

```
@diab_tgtaction\mpc5xx_osc20.cfg
@diab_tgtaction\mpc5xx_osc4.cfg
@diab_tgtaction\mpc555.wsp
```

Note that there is now only one SingleStep workspace file for RAM and flash memory.

The necessary changes to these files depend on the hardware that you are using. Depending on your hardware, you may also need to configure switches and jumper settings. Consult the documentation for your development board.

If you are generating stand-alone real-time applications, you may also need to make changes to settings that are contained in the startup code. These are contained in

*matlabroot*\toolbox\rtw\targets\mpc555dk\drivers\src\applications
\bootcode\bootcode_init.s.t

Note that after making any changes to bootcode_init.s.t, you must
recompile the boot code as described in "Rebuilding the Boot Code and Device
Driver Libraries" on page 2-29.

# Integrating External Blocksets

| **In this section...** |
| --- |
| "Introduction" on page A-27 |
| "Example External Blockset Directory Structure and rtwmakecfg.m" on page A-28 |

## Introduction

You can configure a `rtwmakecfg.m` file to seamlessly integrate custom third-party Simulink blocks with Target for Freescale MPC5xx. You must provide the `rtwmakecfg.m` file along with the third party S-function block DLLs and associated files. `rtwmakecfg.m` files are widely used throughout Real-Time Workshop Embedded Coder and they allow you to:

- Specify include paths to add to the list of includes used in the generated makefiles.

- Specify precompiled libraries to add to the list of libraries used in the generated makefiles.

- Specify TLC include paths to be searched for block TLC files during code generation.

For a general explanation of how to use `rtwmakecfg.m` files, please see the section "Customizing and Creating Template Makefiles" in the documentation for Developing Embedded Targets for Real-Time Workshop Embedded Coder.

For a detailed explanation of using the `rtwmakecfg.m` file please consult the section on "Using the rtwmakecfg.m API" in the Real-Time Workshop documentation.

The next section contains a detailed explanatory example for the MPC5xx build process.

These steps are required:

- Add the location of the `rtwmakecfg.m` file to the MATLAB path.

- Make sure this file is located in the same directory as the S-function DLLs.

## Example External Blockset Directory Structure and rtwmakecfg.m

To understand how the `rtwmakecfg.m` file works, imagine a set of S-functions, comprising a Simulink library, provided by an external supplier, and how they can be integrated into the MPC5xx build process.

Example directory structure for an external (plugin) blockset:

```
C:\externalblocks
C:\externalblocks\tlc_c
C:\externalblocks\include
C:\externalblocks\lib
```

Note: Only the root directory `C:\externalblocks` needs to be on the MATLAB path.

`C:\externalblocks` will contain files such as:

- `Rtwmakecfg.m` — rtwmakecfg.m defining MPC5xx Plugins

- `Blocklibrary.mdl` — Simulink block library containing `Sfun_a` and `Sun_b`

- `Sfun_a.mexw32` — S-function member of `Blocklibrary.mdl`

- `Sfun_b.mexw32` — S-function member of `Blocklibrary.mdl`

`C:\externalblocks\tlc_c` will contain files such as:

- `Sfun_a.c` — S-function source for simulation.

- `Sfun_b.c` — S-function source for simulation.

- `Sfun_a.tlc` — S-function TLC for code generation

- `Sfun_b.tlc` — S-function TLC for code generation

Note: `tlc_c` directories in the same directory as the S-function DLLs are automatically added to the TLC include path.

`C:\externalblocks\include` will contain files such as:

- `Blocksetheader.h` — Header file used in the generated code

`C:\externalblocks\lib` will contain files such as:

- Blocksetlibrary_5xx_CODEWARRIOR.a and
  `Blocksetlibrary_5xx_DIAB.a` — Different versions of the
  library are required depending on which toolchain is being used. The
  variable `mpc5xx_tool_chain` (see example `rtwmakecfg.m` below) enables
  different versions of the library to be selected during the build process,
  based on the target toolchain.

An example `rtwmakecfg.m` that will add the `Blocksetheader.h` parent
directory to the list of include paths and Blocksetlibrary_*ToolChain*.a to the
list of libraries follows:

```
% RTWMAKECFG adds include and source directories to rtw make files.
% makeInfo=RTWMAKECFG returns a structured array containing build info.
% Please refer to the rtwmakecfg API section in the Real-Time workshop
% Documentation for details on the format of this structure.

% Get hold of the fullpath to this file, without the filename itself
rootpath = RTW.transformPaths(fileparts(mfilename('fullpath')));

% Get hold of the toolchain token to uniquely indentify libraries
prefs = RTW.TargetPrefs.load('mpc555.prefs');
mpc5xx_tool_chain = upper(prefs.ToolChain);

% External blocks need the following include path added
% Add the header file
makeInfo.includePath = { fullfile(rootpath, 'include') };

% External blocks reference the following precompiled library
% Add the precompiled libraries
makeInfo.linkLibsObjs = { fullfile(rootpath, 'lib',...
['Blocksetlibrary_' mpc5xx_tool_chain '.a']) };
```

**A-29**

# Examples

Use this list to find examples in the documentation.

# Real-Time Target

# Processor-in-the-Loop Target

# Algorithm Export Target

# Index