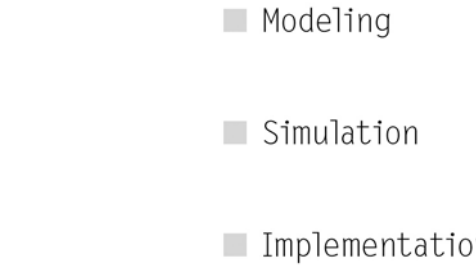
# Embedded Target for OSEK/VDX®

## For Use with Real-Time Workshop®




User's Guide

*Version 1*

**How to Contact The MathWorks:**

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

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

508-647-7000 Phone

508-647-7001 Fax

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

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

*Embedded Target for OSEK/VDX User's Guide*



Printing History: February 2003 Online only Version 1.0 (Release 13+)
June 2004 Online only Version 1.1 (Release 14)

# Contents

## 1 Getting Started



## 2 Configuring the Embedded Target for OSEK/VDX

## 3 Generating Real-Time OSEK/VDX Applications

## 4 Generating Code, Calibration Data, and Reports

## 5 Model Execution



## 6 Block Reference



## Index

1

# Getting Started

| | |
|---|---|
| What Is the Embedded Target for OSEK/VDX? (p. 1-2) | Introduces the product by explaining what it is, explaining what you need to know to use the product, and describing key product features |
| Hardware and Software Requirements (p. 1-5) | Identifies hardware and software required for installing and using the Embedded Target for OSEK/VDX, including development tools (e.g., compilers, debuggers) |
| Installation (p. 1-8) | Discussed installation of required software |
| Getting Help with the Embedded Target for OSEK/VDX (p. 1-9) | Provides a suggested reading path and points you to available online help and demos |

# What Is the Embedded Target for OSEK/VDX?

The Embedded Target for OSEK/VDX® is an add-on product for use with the Real-Time Workshop® Embedded Coder. It provides tools for developing real-time applications for the OSEK/VDX operating system standard of the European automobile industry.

Used with Simulink®, Stateflow®, and the Real-Time Workshop Embedded Coder, the Embedded Target for OSEK/VDX lets you

- Design and model systems and algorithms.
- Compile, download, run, and debug generated code on the target hardware, seamlessly integrating with industry-standard compilers and development tools for OSEK/VDX.

## What You Need to Know to Use This Product

The information provided with this product assumes you are experienced with the following products.

### MathWorks Products

- MATLAB®
- Simulink
- Real-Time Workshop
- Real-Time Workshop Embedded Coder

Minimally, you should read the following from the "Basic Concepts and Tutorials" section of the Real-Time Workshop documentation:

| | |
|---|---|
| "Basic Real-Time Workshop Concepts" | Introduces general concepts and terminology related to Real-Time Workshop |
| "Quick Start Tutorials" | Provides hands-on exercises that demonstrate the Real-Time Workshop 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 particular, you should read

- "Data Structures and Code Modules"
- "Code Generation Options and Optimizations"

### OSEK/VDX Operating System Standard

Familiarity with the OSEK/VDX operating system standard is helpful. Useful documents available through the OSEK/VDX Web site (`http://www.osek-vdx.org/`) include

- *OSEK/VDX Operating System Specification*
- *OSEK Implementation Language (OIL) Specification*

### OSEK/VDX Operating System Implementations

You should be familiar with at least one of the two supported OSEK/VDX operating system implementations and development environments:

| | |
|---|---|
| Tornado for OSEKWorks for PowerPC | Wind River Systems, Inc. |
| ProOSEK | 3SOFT, GmbH |

### Target Boards and Processors

Familiarity with your chosen target board and processor is helpful. Information on the processor register and memory model are useful in configuring your debugger. The Embedded Target for OSEK/VDX has been fully tested with the Phytec phyCORE-MPC555 board, a development board for the Motorola MPC555 processor. This document assumes you are working with the Phytec phyCORE-MPC555 development board. Information about the Motorola MPC555 processor and the Phytec phyCORE-MPC555 board are available at the vendor Web sites.

## Features

Key features of the Embedded Target for OSEK/VDX include

- Target configurations for two major OSEK/VDX operating system implementations and associated development tools.

| | |
|---|---|
| OSEKWorks target | Supports Tornado for OSEKWorks for PowerPC 3.0. |
| ProOSEK target | Supports ProOSEK 3.0r3 |

- Supported Board Support Packages (BSPs):
  - Fully tested with the Phytec PhyCORE-MPC555 development boards. Generates executables for deployment to PhyCORE-MPC555 on-board RAM.
  - Generates code for all BSPs provided with the supported OSEK/VDX implementations.
- Generation of a single- or multirate model or subsystem as an OSEK/VDX executable. You can download and run executables in RAM or FLASH memory.
- Real-Time Workshop task management mechanisms within the OSEK/VDX environment. Maps sample rates in a model to OSEK/VDX tasks.
- Automatic downloading and debugging of code using the SingleStep debugger.
- OSEK/VDX block library that supports basic OSEK APIs.
- ASAP2 file generation
- Single-precision math library (`mathf.h`) support (OSEKWorks only)
- Extensible, open implementation; hook file mechanisms ease customization of code generation options, makefile variables, and rules.

# Hardware and Software Requirements

The following sections identify hardware and software requirements:

- "Host Platform"
- "Hardware Requirements"
- "Required MathWorks Products"
- "OSEK/VDX Software Requirements"

## Host Platform

The Embedded Target for OSEK/VDX supports only the PC as a host platform, running Windows 2000 or Windows XP.

## Hardware Requirements

The MathWorks has tested code generated by the Embedded Target for OSEK/VDX on the Phytec PhyCORE-MPC555 development board, using the Board Support Package (BSP) provided.

The Embedded Target for OSEK/VDX also generates code with other BSPs provided by supported OSEK/VDX implementations. However, the generated code has not been tested on the actual target hardware.

In this document, we assume that you are working with the Phytec phyCORE-MPC555 development board, and we document specific settings and procedures for use with the Phytec phyCORE-MPC555 board, in conjunction with specific cross-development environments. If you use a different development board, you may need to adapt the settings and procedures.

## Required MathWorks Products

The Embedded Target for OSEK/VDX *requires* the following MathWorks products:

- MATLAB 7.0 (Release 14)
- Simulink 6.0 (Release 14)
- Real-Time Workshop 6.0 (Release 14)
- Real-Time Workshop Embedded Coder 4.0 (Release 14)

For more information about these products, see

- The online documentation available through the MATLAB Help browser
- The MathWorks Web site, at `http://www.mathworks.com`; see the "products" section

## OSEK/VDX Software Requirements

The Embedded Target for OSEK/VDX requires one of the following:

- Tornado for OSEKWorks for PowerPC 3.0, from Wind River Systems, Inc.
- ProOSEK 3.0r3 from 3SOFT, GmbH.

### Tornado for OSEKWorks for PowerPC 3.0

OSEKWorks includes the Diab cross-compiler and the SingleStep with vision debugger (Version 7.7.1, for use with VisionPROBE).

The SingleStep debugger is not required for code and executable generation. However, you can use SingleStep to automatically download, run, and debug generated executables. If you prefer, you can use a different debugger or other utility to manually download, execute, and observe the generated application.

The OSEKWorks target supports two versions of SingleStep:

| | |
|---|---|
| SingleStep with Vision, Version 7.7.1 | For use with VisionPROBE. Currently, this version ships with OSEKWorks. Testing at The MathWorks indicates that SingleStep with vision Version 7.7.3 is required for programming FLASH memory. |
| SingleStep On-Chip, Version 7.6.2 | For use with the Phytec PhyCORE-MPC555 board with an on-board Background Debug Mode (BDM) connector, or with an external BDM device, such as the Macraigor Wiggler (WBDM8xx) BDM. |

### ProOSEK 3.0r3 from 3SOFT, GmbH

ProOSEK support options are for the MPC555 processor, with GNU tools. If you want to use the SingleStep debugger, you must obtain it separately. It is not included with ProOSEK.

---

**Note** The GNU tools provided with ProOSEK do not include math library support. Therefore, Simulink blocks that call math library functions are not supported by the ProOSEK target.

---

The SingleStep debugger is not required for code and executable generation. However, you can use SingleStep to automatically download, run, and debug generated executables. If you prefer, you can use a different debugger or other utility to manually download, execute, and observe the generated application.

The ProOSEK target supports two versions of SingleStep:

| | |
|---|---|
| SingleStep with Vision, Version 7.7.1 | For use with VisionPROBE. Testing at The MathWorks indicates that SingleStep with vision, Version 7.7.3 is required for programming FLASH memory. |
| SingleStep On-Chip, Version 7.6.2 | For use with the Phytec PhyCORE-MPC555 board with an on-board Background Debug Mode (BDM) connector, or with an external BDM device, such as the Macraigor Wiggler (WBDM8xx) BDM. |

# Installation

The following sections discuss installation:

- "Installing MathWorks Software"
- "Installing Other Required Software"

## Installing MathWorks Software

To install the Embedded Target for OSEK/VDX and other required MathWorks software:

1. Obtain a License File or Personal License Password from The MathWorks. The License File or Personal License Password identifies the products you can install and use.
2. See the MATLAB installation instructions for your platform.

## Installing Other Required Software

Install software identified in "OSEK/VDX Software Requirements" on page 1-6, as explained in OSEK/VDX product documentation.

# Getting Help with the Embedded Target for OSEK/VDX

The following sections explain how to get help with the Embedded Target for OSEK/VDX:

- "Suggested Reading Path"
- "Online Help"
- "Demos"

## Suggested Reading Path

We suggest the following reading path to get acquainted with the Embedded Target for OSEK/VDX and gain hands-on experience with the features most relevant to your interests:

1. Read "What You Need to Know to Use This Product" on page 1-2 to understand prerequisite knowledge required to use the Embedded Target for OSEK/VDX, and to learn about related documentation you may need to read.

2. Read "Features" on page 1-4 to learn about the general features of the product.

3. Read Chapter 2, "Configuring the Embedded Target for OSEK/VDX" to learn how to set up your development environment and configure the Embedded Target for OSEK/VDX for use with a supported OSEK/VDX implementation (OSEKWorks or ProOSEK).

4. Read Chapter 3, "Generating Real-Time OSEK/VDX Applications" to learn how to generate and deploy OSEK/VDX applications on target hardware. Work through the tutorial that is applicable to your chosen OSEK/VDX implementation.

5. Read Chapter 4, "Generating Code, Calibration Data, and Reports" to learn more about code generation options and other details applicable to your chosen OSEK/VDX implementation.

6. For in-depth information see
   - Chapter 5, "Model Execution" for a description of how generated code executes in the OSEK/VDX environment.

- Chapter 6, “Block Reference” for details on operation of device driver blocks provided in the OSEK/VDX block library.

7 Review “Demos” on page 1-10. Run the demos to gain hands-on experience with the target.

## Online Help

The following online help is available while using the Embedded Coder for OSEK/VDX and related products:

- Online help in the MATLAB Help browser. Click the Embedded Coder for OSEK/VDX product link in the browser’s Contents.
- Online reference help for blocks in the block library. This help is accessible from block context menus that appear when you right-click a specific block or library.

## Demos

The Embedded Coder for OSEK/VDX provides a number of demos to help you get familiar with product features.

To run the demos, use one of the following methods:

- While in the MATLAB Help browser, click on the links in the **Command** column of the following table.
- From the MATLAB **Start** button, click **Start->Simulink->Embedded Target for OSEK/VDX->Demos**.
- Type demo commands from the MATLAB command prompt. For example, `osek_mrate`.

**Table 1-1: Embedded Target for OSEK/VDX Demos**

| Command | Illustrates |
|---|---|
| `osek_apis` | Use of the available OSEK OS API blocks. The OSEK API blocks provide services such as task and alarm activation and buffering. This demo model requires the use of a Phytec phyCORE-MPC555 board. |
| `osek_asap2` | The generation of an ASAP2 file used for calibration. |

**Table 1-1: Embedded Target for OSEK/VDX Demos (Continued)**

| Command | Illustrates |
|---|---|
| `osek_led` | A simple hardware driver that provides visual feedback as the model executes on the target hardware by flashing an LED on and off. This demo model requires the use of a Phytec phyCORE-MPC555 board. |
| `osek_mrate` | How multirate/multitasking models execute under OSEK/VDX. |

2

# Configuring the Embedded Target for OSEK/VDX

| | |
|---|---|
| Setting Up and Verifying Your Installation (p. 2-2) | Gives an overview of the setup process |
| Setting Up Your Target Hardware (p. 2-3) | Identifies port connections and jumper settings required for using the Embedded Target for OSEK/VDX with the Phytec PhyCORE-MPC555 board |
| Setting Target Preferences (p. 2-11) | Explains how to configure environmental settings and preferences associated with the Embedded Target for OSEK/VDX |
| Setting Up Your Installation for the OSEKWorks Target (p. 2-15) | Explains how to configure the Embedded Target for OSEK/VDX for use with OSEKWorks |
| Setting Up Your Installation for the ProOSEK Target (p. 2-17) | Explains how to configure the Embedded Target for OSEK/VDX for use with ProOSEK |
| Setting Up SingleStep (p. 2-19) | Explains how to configure the SingleStep debugger for downloading and debugging generated code to the Phytec PhyCORE-MPC555 board |
| Customization Hooks for the OSEKWorks and ProOSEK Targets (p. 2-24) | Explains how to use hook files to customize target code generation options, makefile variables, and makefile rules |

# Setting Up and Verifying Your Installation

The following sections explain how to configure your development environment for use with the Embedded Target for OSEK/VDX and verify correct operation. Initial configuration steps are explained in

- "Setting Up Your Target Hardware" on page 2-3
- "Setting Target Preferences" on page 2-11

---

**Note** Target preferences properties include information about your local system, such as the location of the OSEK implementation and debugger. Be sure to localize these properties appropriately for you installation.

---

After completing these steps, proceed to the section appropriate to your development environment:

- If you are using OSEKWorks, see "Setting Up Your Installation for the OSEKWorks Target" on page 2-15.
- If you are using ProOSEK, see "Setting Up Your Installation for the ProOSEK Target" on page 2-17.

# Setting Up Your Target Hardware

The following sections assume you are working with the Phytec phyCORE-MPC555 development board. This section gives information on the required connections and jumper settings for that board, and on special test and linker command files provided for the phyCORE-MPC555 board.

After setting up your phyCORE-MPC555 board, you must set environment variables associated with the Embedded Target for OSEK/VDX, as described in "Setting Target Preferences" on page 2-11.

## Physical Connections and Communications Ports

Before you begin working with the Embedded Target for OSEK/VDX, set up your phyCORE-MPC555 board and connect it to your host computer. For hardware setup information, see "Interfacing the phyCORE-MPC555 to a Host PC" in the "Getting Started" chapter of the *phyCORE-MPC555 Quickstart Instructions* manual.

These instructions assume that you have connected the BDM port of your phyCORE-MPC555 board to the parallel port (LPT1) of your host PC. This connection is used for host/target communication when downloading code or debugging via the SingleStep debugger.

The configuration of the host system parallel port depends on both the version of SingleStep and the type of BDM interface that you use. Consult your SingleStep documentation for detailed instructions.

Be sure to configure the parallel port of your host PC correctly for your specific Windows operating system version, as directed by the operating system documentation.

## Jumper Settings

The Embedded Target for OSEK/VDX has been tested by the MathWorks with the Phytec phyCORE-MPC555 board, using the on-board BDM and jumper settings indicated in the tables below:

- Table 2-1 gives jumper settings for use with the on-board BDM interface.
- Table 2-2 gives jumper settings for use with an external Wiggler BDM or with VisionPROBE.

- Table 2-3 gives jumper settings to use when executing code that has been programmed into FLASH memory. (Code execution is initiated either by pressing the **Reset** button or by cycling power on the board.)

Jumpers that are not shown in the tables are not relevant to the Embedded Target for OSEK/VDX.

For jumper locations and pin numbers, see the *phyCORE-MPC555 Quickstart Instructions* manual.

**Table 2-1: PhyCORE-MPC555 Jumper Settings for Use with On-Board BDM**

| Jumper | Setting |
|---|---|
| JP1 | 3+4 |
| JP2 | 1+2 |
| JP5-9 | closed |
| JP17 | 1+2 |

**Table 2-2: PhyCORE-MPC555 Jumper Settings for Use with External Wiggler BDM or VisionPROBE**

| Jumper | Setting |
|---|---|
| JP1 | open |
| JP2 | open |
| JP5-9 | open |
| JP17 | open |

**Table 2-3: PhyCORE-MPC555 Jumper Settings for Execution from On-Chip FLASH Memory at Power-On or Reset**

| Jumper | Setting |
|---|---|
| JP4 | 1+2 |
| JP15 | 1+2 |
| JP5 (solder jumper on daughter card) | 1+2 (default) |

## Special Files Provided for Use with the Phytec phyCORE-MPC555 Board

The following special files are provided for use with the Phytec phyCORE-MPC555 board:

- "Board Support Package for Use with OSEKWorks"
- "Test Executable"

### Board Support Package for Use with OSEKWorks

The Embedded Target for OSEK/VDX provides a Board Support Package that supports use of the Phytec PhyCORE-555 board with the OSEKWorks target. To install this BSP, see "Installing the PhyCORE-555 BSP for OSEKWorks" on page 2-15.

### Test Executable

The Embedded Target for OSEK/VDX provides an executable (`.elf` file) for the demo `osek_led.mdl`. The test file is *`matlabroot`*`/toolbox/rtw/targets/osek/osekdemos/bin/osek_led.elf`. This file was generated from the `osek_led` demo model.

If you are targeting the Phytec phyCORE-MPC555 board, you can use this file with SingleStep or another debugger to verify that your board, cable, and jumper setup are correct. The demo `osek_led` model uses phyCORE-MPC555 specific device driver blocks. When the demo executable is running, the device driver blinks two LEDs on the phyCORE-MPC555 board at different rates.

You can download and run the test executable by following the procedure described in "Downloading the Generated Code to RAM" on page 3-21. In step 2 of that section, use the **Browse** button to locate the `osek_led.elf` file. Then,

follow the remaining steps to download the code and start a debugging session. When the SingleStep session has been started, click the green **Go** arrow to start program execution. Observe the LEDs on the phyCORE-MPC555 board to verify correct operation.

## Configuring the Memory Map for the Phytec phyCORE-MPC555 Board

The Embedded Target for OSEK/VDX provides default linker command files for use in OSEKWorks and ProOSEK implementations. These command files support generation of read-only (code) sections, which can be located either in off-chip RAM or on-chip FLASH. The mapping is as follows:

- From address `0x00000` to `0x40000`: read-only sections (256KB)
- Starting at address `0x3f9800`: read-write sections (26KB) to the on-chip RAM area

This mapping, along with the register setup performed by the initialization (startup) code for generating the Chip Select 1 (CS1) signal, allows you to use the generated executable in either of the following ways:

- "Downloading and Running the Executable in RAM" on page 2-6
- "Downloading and Running the Executable in FLASH" on page 2-7

Other sections that follow discuss

- "Other Memory Mapping Examples" on page 2-7
- "Modifying the Memory Map for the OSEKWorks Target" on page 2-8
- "Modifying the Memory Map for the ProOSEK Target" on page 2-9

### Downloading and Running the Executable in RAM

When you select `Download_and_run` option, the build process invokes the SingleStep debugger, configured with the FLASH Enable (`FLEN`) bit in the Internal Memory Map register (`IMMR`) set to `0`. SingleStep loads the executable (`.elf` file) into off-chip RAM. Thus, by default, SingleStep loads the code image into off-chip RAM and uses the on-chip RAM for read-write sections.

You can also manually download and run code in RAM with the `FLEN` bit set to `0`. For details on how to do this, see

- "Tutorial 3: Downloading the Application to RAM via SingleStep" on page 3-21
- "Tutorial 4: Automated Downloading and Debugging" on page 3-28

### Downloading and Running the Executable in FLASH

If you start and manually configure the SingleStep debugger to connect to the target with the `FLEN` bit of `IMMR` set to `1`, the MPC555 board maps the internal access cycles from `0x0` to `0x2FBFFF` to the on-chip FLASH. Thus, you can connect SingleStep to the target and then use the SingleStep **Flash Programmer** dialog box to load the generated executable (`.bin` file) into the on-chip FLASH. During reset, if you configure the board to set the `FLEN` bit to `1`, the processor executes the code from on-chip FLASH.

For details on running generated code in FLASH, see

- "Tutorial 5: Downloading Generated Code to FLASH" on page 3-31
- Table 2-3, PhyCORE-MPC555 Jumper Settings for Execution from On-Chip FLASH Memory at Power-On or Reset

See also "Hard Reset Configuration Word" in the *MPC555 Users Manual* for information on controlling reset behavior. Also, see the Phytec manuals for information on jumper settings on the Phytec board, which allow board-level control over the Hard Reset Configuration Word.

### Other Memory Mapping Examples

The linker command files provide other memory mapping examples. In the following example, off-chip RAM is equally divided between read-only and read-write sections. This example assumes 256KB of off-chip RAM on the Phytec board. Larger RAM sizes are also acceptable (but would allow use of only the first 256KB of RAM).

```
ram_as_rom: org = 0x2000, len = 0x1e000
ram : org = 0x20000, len = 0x20000
```

This mapping allocates slightly less than 128KB of memory for the program (ROM) and 128KB for RAM. The address space below `0x2000` is reserved for interrupt vectors (refer to the exception memory regions section of linker command file). If your target board has additional memory and you want to make it available for larger executable image sizes, you can edit the linker command file to support the additional memory. For example, to support 1MB

of on-board RAM, and allocate 512KB each to read-only and read-write sections, you would change the mapping as follows.

```
ram_as_rom: org = 0x2000, len = 0x7e000
ram : org = 0x80000, len = 0x80000
```

When you increase the size of the memory map, verify that the phyCORE-MPC555 jumpers are set correctly to ensure proper memory chip selection. Consult Table 5 in the in the *phyCORE-MPC555 Hardware Manual* for proper settings for solder jumpers 18 and 10 on the daughter card module.

Modifying the memory maps or the power-on/reset behavior of the processor requires a detailed understanding of the internal processor register values that control memory mapping and chip select signals, and of how they are wired for your particular board. The register values can be set by

- The Hard Reset Configuration Word
- The debugger when it connects to the target
- The startup code of the executable

Register files for the SingleStep debugger (e.g., `*.cfg`, `*.reg` and `*.wsp`) control initial register values. BSP files (such as the OSEKWorks `procinit.s`) set values when the code is run.

### Modifying the Memory Map for the OSEKWorks Target

To modify the phyCORE-MPC555 memory map for programs generated by the OSEKWorks target,

1. Edit the linker command file, `bsp.lk`. You can find and edit `bsp.lk` in the following location.

   ```
   installdir/TornadoOW_ppc_3.0/target/osekworks/bsp/ppc/phycore555/src
   ```

2. Rebuild the phyCORE-MPC555 BSP by using the following commands from the command prompt window:

   ```
   cd <installdir>TornadoOW_ppc_3.0/target/osekworks/bsp/ppc/phycore555/src
   .\phycore_make.bat clean
   .\phycore_make.bat all
   ```

   Note that if you run the `setup_osekworks_phycorebsp.m` script again, it will overwrite your changes to `bsp.lk`.

## Modifying the Memory Map for the ProOSEK Target

The board configuration specifics required by the ProOSEK environment come from a board directory. The ProOSEK target provides a board directory for the phyCORE-MPC555, which is installed using `setup_proosek_phycorebsp.m`. The board directory for the phyCORE-MPC555 is *`matlabroot`*`/toolbox/rtw/targets/osek/proosek/boards/PHYCORE555`.

This directory contains

| File | Description |
|---|---|
| `os.cmd` | Linker command file. |
| `startup.s` | Source code for the program initialization. |
| `board.h` | Board specific macros. Clock configuration is the most important function defined here. |
| `PHYCORE555.cnf` | Configuration information about the board. |
| `target.txt` | General information about the board. |

These files are copies of files installed with ProOSEK. See your ProOSEK documentation for more information on these files.

To modify the phyCORE-MPC555 memory map for programs generated by the ProOSEK target,

1. At a minimum, edit the linker command file `os.cmd` in the board support directory. The `PHYCORE555.cnf` file also contains memory configuration information, but this file is not used during the code generation, compilation, or automated download and run process.

2. Run the `setup_proosek_phycorebsp` script to recopy the BSP. Respond `y` to all prompts from the script. (See "Installing the PhyCORE-555 BSP for ProOSEK" on page 2-17). The `setup_proosek_phycorebsp` script copies `os.cmd` to the `boards` folder within the ProOSEK installation tree. Where

`installdir` is the ProOSEK root directory, `os.cmd` is located in `installdir/boards/PHYCORE555`.

You can edit this copy of `os.cmd`, but we do not recommend doing so, because if you run the `setup_proosek_phycorebsp.m` script again, it will overwrite `os.cmd`.

# Setting Target Preferences

This section describes environmental settings associated with the Embedded Target for OSEK/VDX. These settings, which persist across MATLAB sessions and different models, are referred to as *target preferences*. Target preferences let you specify properties such as the location of your installed OSEK/VDX implementation and other parameters affecting the generation, building, and downloading of code.

## Target Preference Properties

Table 2-4 summarizes the preference properties, and their defaults, for the Embedded Target for OSEK/VDX.

**Table 2-4: Embedded Target for OSEK/VDX Preferences Summary**

| Preference Name | Description | Default or Example Value |
|---|---|---|
| `Debugger` | Name of debugger used for automatic downloading | `'SingleStep'`<br>Note: An alternate value is `'Custom'`. If you select `'Custom'`, you must implement custom debugger support. (See “Custom Debugger Support” on page 4-11.) |
| `DebuggerEXE` | Name of debugger executable (*must be localized for your installation*) | Default: `'visppc.exe'` for SingleStep with vision<br>Example: `'bdmp58.exe'` (for SingleStep On-Chip 7.6.2) |
| `DebuggerPath` | Path to installed debugger (*must be localized for your installation*) | Default: `d:\Applications\WindRiver\TornadoOW_ppc_3.0\host\sds\7.7.1` |

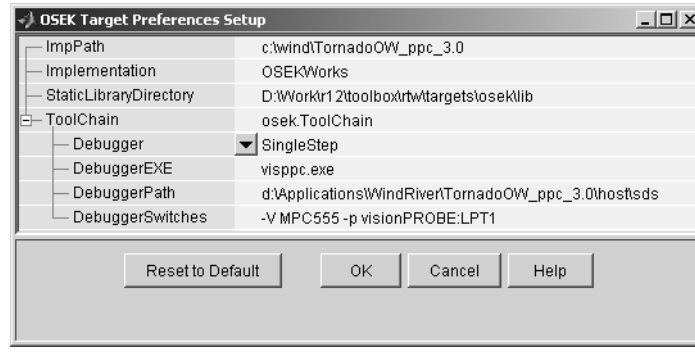
**Table 2-4: Embedded Target for OSEK/VDX Preferences Summary**

| Preference Name | Description | Default or Example Value |
|---|---|---|
| `DebuggerSwitches` | Switches to set debugger options (such as port name and speed) when debugger is invoked for auto-downloading. These options apply to SingleStep. Normally, you should use the defaults, unless you use a port other than `LPT1` for debugger communications. | Default for SingleStep with vision<br>`'-V MPC555 -p visionPROBE:LPT1'`<br>Correct value for SingleStep On-Chip (7.6.2) with BDM port:<br>`-p LPT1:1` |
| `Implementation` | Name of installed OSEK/VDX implementation (`'osekworks'` or `'proosek'`) | `'osekworks'` |
| `ImpPath` | Path to installed OSEK/VDX implementation (*must be localized for your installation*) | Examples:<br>`'c:\wind\TornadoOW_ppc_3.0'` (for an OSEKWorks installation)<br>`'c:\ProOSEK'` (for a ProOSEK installation) |
| `StaticLibraryDirectory` | Directory where static object libraries are built and stored (see "Efficient Use of Persistent Object Libraries" on page 4–10) | Default:<br>*`matlabroot`*`\toolbox\rtw\targets\osek\lib`<br>Examples:<br>*`matlabroot`*`\work`<br>`c:\temp` |

**Note** Do not use the default value for the `ImpPath`, `DebuggerPath`, or `DebuggerEXE` preferences. You must modify these preferences to indicate the locations on your PC (or network) where your OSEK/VDX implementation and debugger are installed. Build errors will result if these preferences are not set correctly.

## Editing Target Preferences

To configure the target preferences, you use the **OSEK Target Preferences Setup** window. This window lets you view, edit, and save the preferences, or reset the preferences to their default (factory) values.




**OSEK Target Preferences Setup Window**

To open the **OSEK Target Preferences Setup** window and edit target preferences:

1. Click on the MATLAB **Start** button. Follow the **Start ->Simulink ->Embedded Target for OSEK/VDX** links.

2. Select **OSEK Target Preferences** from the **Embedded Target for OSEK/VDX** submenu. The **OSEK Target Preferences Setup** window opens.

   Alternatively, you can open this window by typing `osekeditprefs` at the MATLAB prompt.

3 Modify the properties you want to change.

4 Click **OK** to close the window and make your changes persistent.

# Setting Up Your Installation for the OSEKWorks Target

Setting up your installation for the OSEKWorks target is relatively simple:

1. Obtain the required version of OSEKWorks (see "OSEK/VDX Software Requirements" on page 1-6).

2. Install OSEKWorks, following the Wind River Systems documentation. You can install OSEKWorks locally or on your network. Be sure to configure the Windows environment variable `LM_LICENSE_FILE` appropriately for the OSEKWorks license manager.

3. Set the target preferences correctly for your installation (see "Setting Target Preferences" on page 2-11). Make sure that the `ImpPath`, `DebuggerPath`, `DebuggerEXE`, and `DebuggerSwitches` preferences are localized correctly for your installation.

4. If you want to use the Phytec PhyCORE-555 board as your hardware target, run the `setup_osekworks_phycorebsp` script to install the PhyCORE-555 Board Support Package, as described in "Installing the PhyCORE-555 BSP for OSEKWorks" on page 2-15.

5. If you want to use the SingleStep debugger for downloading and debugging code, configure SingleStep as described in "Setting Up SingleStep" on page 2-19. You can use another debugger or download utility to download code manually. Note, however, that the automatic download/debug features of the OSEKWorks target require SingleStep.

## Installing the PhyCORE-555 BSP for OSEKWorks

The OSEKWorks target provides a BSP for the Phytec PhyCORE-555 development board, including full source code and an M-file installer script, `setup_osekworks_phycorebsp.m`. The installer script is located in *`matlabroot`*`\toolbox\rtw\targets\osek\osekworks`. If you want to use the PhyCORE-555 as your hardware target, you must install the PhyCORE-555 BSP. To do this,

1. Set the target preferences correctly for your installation (see "Setting Target Preferences" on page 2-11). Make sure that the `ImpPath` property is set correctly, as the installer script uses `ImpPath` to locate the files.

2 At the MATLAB prompt, type

```
setup_osekworks_phycorebsp
```

3 The installer displays the path to the location where the BSP will be installed, and prompts you to continue as shown below.

```
Ready to create Phytec BSP in directory tree:
'D:\wind\TornadoOW_ppc_3.0\target\osekworks'
Do you want to continue?([y]/n):y
```

4 The installer copies the required files and prompts you to continue as shown below.

```
Successfully copied  files into OSEKWorks tree...
Successfully created 'phycore_make.bat'...
Do you want to build the BSP now?([y]/n):y
```

5 The installer builds the BSP, displaying a number of progress messages. When the BSP is built, the installer displays the following completion message.

```
Finished setup of phycore555.
```

# Setting Up Your Installation for the ProOSEK Target

Setting up your installation for the ProOSEK target is relatively simple:

1. Obtain the required version of ProOSEK (see "OSEK/VDX Software Requirements" on page 1-6).
2. Install ProOSEK, following the 3SOFT documentation. You can install ProOSEK locally or on your network.
3. Set the target preferences correctly for your installation (see "Setting Target Preferences" on page 2-11). Make sure that the `ImpPath`, `DebuggerPath`, `DebuggerEXE`, and `DebuggerSwitches` preferences are localized correctly for your installation.
4. If you want to use the Phytec PhyCORE-555 board as your hardware target, run the `setup_proosek_phycorebsp` script to install the PhyCORE-555 Board Support Package, as described in "Installing the PhyCORE-555 BSP for ProOSEK" on page 2-17.
5. If you want to use the SingleStep debugger for downloading and debugging code, configure SingleStep as described in "Setting Up SingleStep" on page 2-19. You can use another debugger or download utility to download code manually. Note, however, that the automatic download/debug features of the OSEKWorks target require SingleStep.

## Installing the PhyCORE-555 BSP for ProOSEK

The ProOSEK target provides a BSP for the Phytec PhyCORE-555 development board, including an M-file installer script, `setup_proosek_phycorebsp.m`. The installer script is located in *`matlabroot`*`\toolbox\rtw\targets\osek\proosek`.

If you want to use the PhyCORE-555 as your hardware target, you must install the PhyCORE-555 BSP. To do this,

1. Set the target preferences correctly for your installation (see "Setting Target Preferences" on page 2–11). Make sure that the `ImpPath` property is set correctly, as the installer script uses `ImpPath` to locate the files.
2. At the MATLAB prompt, type

```
setup_proosek_phycorebsp
```

3 The installer displays the path to the location where the BSP will be installed, and prompts you to continue as shown below.

```
Ready to copy  into directory tree:
'D:\3Soft\ProOSEK'
Do you want to continue?([y]/n):y
```

4 The installer copies the required files, and displays the following completion message.

```
Successfully copied  files into ProOSEk tree.
Finished setup of phycore555.
```

# Setting Up SingleStep

The SingleStep debugger lets you download, run, and debug code generated by the Embedded Target for OSEK/VDX on a target board. The following sections discuss



After configuring SingleStep, you will also be able to use the debugger directly to debug generated programs.

We assume that you will be using SingleStep in conjunction with a Phytec phyCORE-MPC555 board.

---

**Note** The SingleStep options and user interface screens discussed below are based on SingleStep version 7.6.2 and 7.7.1 and may differ from your installed version of SingleStep, or with future versions of SingleStep. The MathWorks provides the configuration information below only as a convenience. To resolve questions or difficulties with SingleStep, refer to the SingleStep documentation, or contact Wind River Systems.

---

## Installing SingleStep

If you have not already done so, install the SingleStep debugger and confirm its operation with your phyCORE-MPC555 board before proceeding with this section. You should select the `SStep Professional Suite (MPC5xx)` option during installation. If necessary, please consult your SingleStep documentation.

## Configuring SingleStep On-Chip 7.6.2

If you want to program your generated applications into the phyCORE-MPC555 FLASH memory using SingleStep On-Chip 7.6.2, do the following:

1 Obtain the following patch files from Wind River Systems:
   - `pcflash11_29_00.txt`
   - `pcflash11_29_00.zip`
   - `pcflash3_15_01.txt`
   - `pcflash3_15_01.zip`

2 Read the information in `pcflash11_29_00.txt` file.

3 Apply the update in `pcflash11_29_00.zip`.

4 Read the information in `pcflash3_15_01.txt`.

5 Apply the update in `pcflash3_15_01.zip`.

6 Create a shortcut to SingleStep On-Chip 7.6.2 to start SingleStep with the proper options. You will use this shortcut when manually downloading code to RAM or FLASH via SingleStep.

   The SingleStep installer adds a shortcut named `SingleStep On Chip (MPC5xx)` in your system's **Start/Programs/SingleStep 7.6.2** menu.

   a Locate the shortcut file and make a copy of it on your desktop.

   b Rename the copy to `SingleStep On Chip (MPC5xx) for OSEK Target`.

   c Right-click on the `SingleStep On Chip (MPC5xx) for OSEK Target` shortcut file and edit its **Target** property to read as follows:

```
ssteproot\cmd\bdmp58.exe -P -S
matlabroot\toolbox\rtw\targets\osek\osek\@osek_singlestep_tgtaction\phycore-555.w
sp
```

   *ssteproot* is the installed SingleStep directory and *matlabroot* is the MATLAB root directory.

## Configuring SingleStep with Vision

OSEKWorks includes SingleStep with vision (version 7.7.1). This version of SingleStep is intended for use with VisionPROBE hardware. To set up SingleStep with vision and the VisionPROBE for use with the PhyCORE-MPC555 board and the OSEKWorks target, do the following:

1 Configure VisionPROBE nonvolatile RAM. The OSEKWorks target provides a MATLAB command that creates and runs a SingleStep script that

configures the VisionPROBE nonvolatile RAM. The script, `visppcinit.cfg`, is customized to your environment. To use this command, type the following at the MATLAB prompt:

```
osektgtaction('visppcinit');
```

The following message appears:

```
Execute SingleStep as: start
d:\Applications\WindRiver\TornadoOW_ppc_3.0\host\sds\7.7.1\cmd\visppc.exe  -r
d:\work\visppcinit.cfg -S
D:\Work\R12target\toolbox\rtw\targets\osek\osek\@osek_singlestep_tgtaction\visppc
init.wsp
```

SingleStep executes, under control of the script. Note that at this point only the SingleStep command window is visible. The script directs SingleStep to connect to the VisionPROBE. Then, the script executes another VisionSHELL script. This script is located within `matlabroot`, in the subdirectory
`\toolbox\rtw\targets\osek\osek\@osek_singlestep_tgtaction\rtw_phycore555.reg`.

The `rtw_phycore555.reg.` script programs the VisionPROBE nonvolatile RAM.

After the `visppcinit.cfg` script completes, the SingleStep command window opens. When SingleStep has successfully programmed the VisionPROBE, it displays a message instructing you to cycle power on the phyCORE-MPC555 board and manually execute the `reset` command in the SingleStep command window.

2. Cycle power on the phyCORE-MPC555 board.

3. Enter the `reset` command. When the `reset` command returns without error, VisionPROBE programming is complete, and the VisionPROBE is communicating with the PhyCORE-MPC555 board correctly.

4. Exit the SingleStep session.

5. Create a shortcut for SingleStep with vision to start SingleStep with the proper options. You will use this shortcut when manually downloading code to RAM or FLASH via SingleStep with vision.

a Find the SingleStep with vision executable. The executable is in *OSEKWorksroot*\host\sds\7.7.1\cmd\visppc.exe. *OSEKWorksroot* refers to the root OSEKWorks directory.

b Create a shortcut to the executable on your desktop.

c Rename the shortcut to `SingleStep with vision(MPC5xx)for OSEK Target`.

d Right-click on the `SingleStep with vision (MPC5xx) for OSEK Target` shortcut file and edit its `Target` property to read as follows:

```
OSEKWorksroot\host\sds\7.7.1\cmd\visppc.exe -P -S
matlabroot\toolbox\rtw\targets\osek\osek\@osek_singlestep_tgtaction\phycore-555.w
sp
```

6 Install SingleStep 7.7.3 and apply patches for programming FLASH memory.

SingleStep with vision version 7.7.1 does not provide built-in support for programming the phyCORE-MPC555 FLASH memory. However, an extension that supports FLASH programming is included with SingleStep with vision 7.7.3. If you want to program your generated applications into the phyCORE-MPC555 FLASH memory, install SingleStep with vision 7.7.3 and apply this extension.

SingleStep with vision 7.7.3 is available from the Wind River Web site. Version 7.7.3 works with licenses provided for version 7.7.1.

a Locate the following .zip file in the SingleStep subdirectory:

`cmd/Addin_Flash_Drivers/custom/MPC555/c1.0e`

This zip file contains the extension for phyCORE-MPC555 FLASH programming with SingleStep with vision 7.7.3.

b Unpack the .zip file, which includes the documentation and files required to apply the extension to SingleStep. The Embedded Target for OSEK/VDX presumes the .zip file is unpacked in place, and that the `ppcusr.bin` file found there is placed in the SingleStep `cmd` directory.

c Update the `SingleStep with vision (MPC5xx) for OSEK Target` shortcut you created in step 5 to point to the new version of SingleStep.

## Configuring phyCORE-MPC555 Jumpers

Make sure that the jumpers on the phyCORE-MPC555 board are set as described in “Jumper Settings” on page 2–3.

## Configuring SingleStep Parameters

The procedure for configuring SingleStep parameters, downloading code to the target via BDM, and running a debugging session is provided in “Tutorial 3: Downloading the Application to RAM via SingleStep” on page 3-21. Read that section and generate code. Then, configure SingleStep parameters and download code as described.

You can also test your SingleStep installation using a test executable provided with the Embedded Target for OSEK/VDX. See “Special Files Provided for Use with the Phytec phyCORE-MPC555 Board” on page 2-5.

# Customization Hooks for the OSEKWorks and ProOSEK Targets

The Embedded Target for OSEK/VDX provides hook file mechanisms that simplify customization of the system target files (STFs) and template makefiles (TMFs) for both the OSEKWorks and ProOSEK targets. To use these mechanisms, you do not have to edit the distributed STFs or TMFs.

## Adding Custom Code Generation Options

The target-specific code generation options for the OSEKWorks and ProOSEK targets are specified by the `rtwoptions` structures defined in the STFs, `osekworks.tlc` and `proosek.tlc`. The `rtwoptions` structure is described in the Real-Time Workshop documentation.

You can extend these options by creating a hook file named `user_osek_options.tlc` that specifies additional elements for the `rtwoptions` structure. For Real-Time Workshop to find the `user_osek_options.tlc` file, the file must be in your current working directory or on your MATLAB path.

To see how this works, do the following:

1 Create the file `user_osek_options.tlc` such that it includes the following structure definitions:

```
/%
  BEGIN_RTW_OPTIONS

  rtwoption_index = 0;

  rtwoption_index = rtwoption_index + 1;
  rtwoptions(rtwoption_index).prompt        = 'USER OPTIONS';
  rtwoptions(rtwoption_index).type          = 'Category';
  rtwoptions(rtwoption_index).enable        = 'on';
 rtwoptions(rtwoption_index).default   = 1;  % number of items under this category
  rtwoptions(rtwoption_index).popupstrings  = '';
  rtwoptions(rtwoption_index).tlcvariable   = '';
  rtwoptions(rtwoption_index).tooltip       = '';
  rtwoptions(rtwoption_index).callback      = '';
  rtwoptions(rtwoption_index).opencallback  = '';
  rtwoptions(rtwoption_index).closecallback = '';
  rtwoptions(rtwoption_index).makevariable  = '';

  rtwoption_index = rtwoption_index + 1;
  rtwoptions(rtwoption_index).prompt        = 'User load factor';
  rtwoptions(rtwoption_index).type          = 'Edit';
  rtwoptions(rtwoption_index).default       = '20';
  rtwoptions(rtwoption_index).tlcvariable   = 'ulfactor';
  rtwoptions(rtwoption_index).makevariable = 'ULFACTOR';
  rtwoptions(rtwoption_index).tooltip       = ['The user load factor '];
  rtwoptions(rtwoption_index).callback      = '';
  rtwoptions(rtwoption_index).opencallback  = '';
  rtwoptions(rtwoption_index).closecallback = '';

  END_RTW_OPTIONS
 %/
```

2 Check that the file is in your current working directory or is on your MATLAB path.

3 Open the model `osek_led`.

4 Open the **Configuration Parameters** dialog. Note the new **USER OPTIONS** item that appears under **Real-Time Workshop**.

5 Click **USER OPTIONS**. The edit field **User load factor** appears in a right-side pane with a default value of 20. This field provides values for the associated TLC and makefile variables used in the build process.

You can also overload existing options that are defined in the provided STFs (`osekworks.tlc` and `proosek.tlc`). To overload an existing option, use the same `tlcvariable` field that is defined in the STF. An example of overloading an option would be to add a BSP name to the **OSEKWorks Board Support Package** menu. In such a case, you should add to the existing menu, not simply replace it with a single value. In the following example, `myBoard` is added to the list of menu items.

```
rtwoption_index = rtwoption_index + 1;
  rtwoptions(rtwoption_index).prompt        = 'Modified OSEKWorks Board
Support Package (BSP)';
  rtwoptions(rtwoption_index).type          = 'Popup';
  rtwoptions(rtwoption_index).default       = 'phycore555';
  rtwoptions(rtwoption_index).popupstrings  =
'axiomcmd565|axiomcme555|estsbc555|motevb555|motmbx8xx|phycore555|myBoard';
  rtwoptions(rtwoption_index).tlcvariable   = 'bspName';
  rtwoptions(rtwoption_index).makevariable  = 'OSEK_BOARD';
  rtwoptions(rtwoption_index).tooltip       = ['Customized Board Support
Packages'];
  rtwoptions(rtwoption_index).callback      = '';
  rtwoptions(rtwoption_index).opencallback  = '';
  rtwoptions(rtwoption_index).closecallback = '';
```

## Adding Custom Makefile Variables and Rules

The TMFs for the OSEKWorks and ProOSEK targets (`osekworks.tmf` and `proosek.tmf`) provide `include` statements that allow you to specify additional makefile variables and makefile rules.

The `include` statement for variables is

```
-include ..\user_makefile_variables.mk
```

If the `user_makefile_variables.mk` file exists, the variables it defines are added to the generated makefile.The `include` statement for rules is

```
-include ..\user_makefile_rules.mk
```

If the `user_makefile_rules.mk` file exists, the rules it defines are added to the rules section of the generated makefile.

3

# Generating Real-Time OSEK/VDX Applications

# Introduction

This section explains how to use the Embedded Target for OSEK/VDX to generate, download, and run real-time OSEK/VDX applications on a target development board.

A suggested path through these tutorials follows:

1. Read and work through "The Multirate Example Model" on page 3-4 to learn about the demo model that is used in tutorials 1 to 4.

2. Read and work through the tutorial appropriate to the OSEK/VDX implementation you use. In the tutorial, you will generate a target executable using your OSEK/VDX implementation and development environment.
   - If you use OSEKWorks, see "Tutorial 1: Creating an Application with OSEKWorks" on page 3-6.
   - If you use ProOSEK, see "Tutorial 2: Creating an Application with ProOSEK" on page 3-13.

3. Proceed to "Tutorial 3: Downloading the Application to RAM via SingleStep" on page 3-21 after you have generated an executable. This tutorial guides you through a typical manual downloading and debugging session with the SingleStep debugger and a Phytec PhyCORE-MPC555 development board. It will give you some insight into how a generated program executes under OSEK/VDX.

4. Continue with "Tutorial 4: Automated Downloading and Debugging" on page 3-28 to learn how to use the automated downloading and debugging features of the product.

5. The preceding tutorials use a RAM-based application. To learn how to download and run code in FLASH memory, work through the final "Tutorial 5: Downloading Generated Code to FLASH" on page 3-31.

6. The tutorials introduce you to the basic operation of the Embedded Target for OSEK/VDX. After you understand the basic feature set, read "Code Generation Options" on page 4-5 for a complete list of all the options available.

In addition to the Embedded Target for OSEK/VDX, you need the following components:

- A supported OSEK/VDX implementation and development environment (see "Required MathWorks Products" on page 1-5).
- A Phytec PhyCORE-MPC555 development board. The MathWorks has fully tested and qualified the Embedded Target for OSEK/VDX for use with the PhyCORE-MPC555 board and the associated Board Support Package (BSP).
- SingleStep debugger. (See "OSEK/VDX Software Requirements" on page 1-6.) SingleStep is used to download, execute, and observe the generated application.

---

**Note** If you want to use a different development board, debugger, or download utility for this tutorial, adapt the procedures described below, especially "Tutorial 3: Downloading the Application to RAM via SingleStep" on page 3-21. If you plan to use a board other than the PhyCORE-MPC555, be aware that the Embedded Target for OSEK/VDX does support all the BSPs provided by OSEKWorks and ProOSEK. However, the MathWorks has not tested beyond the code generation stage with boards other than the PhyCORE-MPC555.

---

# The Multirate Example Model

All the tutorials in this chapter (except Tutorial 5) use a simple demo model, `osek_mrate`. This demo is provided with the Embedded Target for OSEK/VDX:

1. Open the model. If you are reading this document online in the MATLAB Help browser, you can open the model by clicking this link: `osek_mrate`.

   Alternatively, type the model name at the MATLAB command line.

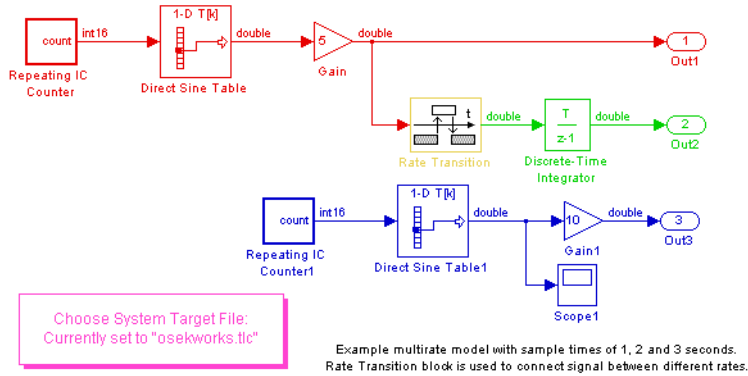
   ```
   osek_mrate
   ```

2. Create a directory, `osek_tut`, that is outside the MATLAB directory structure. Make `osek_tut` your working directory.

3. Save a local copy of the `osek_mrate` model to your working directory. You will work with this copy throughout this exercise.

4. `osek_mrate` is a multirate model with three sample rates. The **Sample time colors** option is enabled for this model. Type **Ctrl+D** to update the diagram and color code the blocks and lines in your model to indicate the sample rates at which the blocks operate.

   Observe that the fastest (red) blocks have a sample rate of 1 Hz (sample time of 1 second); thus the base rate of the model is 1 Hz. The next-fastest blocks (green) have a sample rate of 2 Hz (sample time of 2 seconds). A Rate Transition block is inserted between the 1 Hz Gain block and the 2 Hz Discrete Time Integrator block. The slowest (blue) blocks have a sample rate of 3 Hz (sample time of 3 seconds).




### osek_mrate Model

**5** To learn how to configure the model and generate code for your OSEK/VDX implementation, continue with one of the following tutorials:

- If you use OSEKWorks, continue to “Tutorial 1: Creating an Application with OSEKWorks” on page 3-6.
- If you use ProOSEK, continue to “Tutorial 2: Creating an Application with ProOSEK” on page 3-13.

# Tutorial 1: Creating an Application with OSEKWorks

In this tutorial, you build a real-time multitasking application for OSEKWorks from a simple model. You should already be familiar with Simulink and with the Real-Time Workshop code generation and build process.

In the following sections, you

1. Configure the model.
2. Generate and examine code and build an executable program.
3. Download the executable code to a target board, initiate a debugging session, and set breakpoints and observe the execution of the program.

## Before You Begin

This tutorial requires specific hardware and software (as described in "Introduction" on page 3-2) in addition to the Embedded Target for OSEK/VDX. Be sure that you have

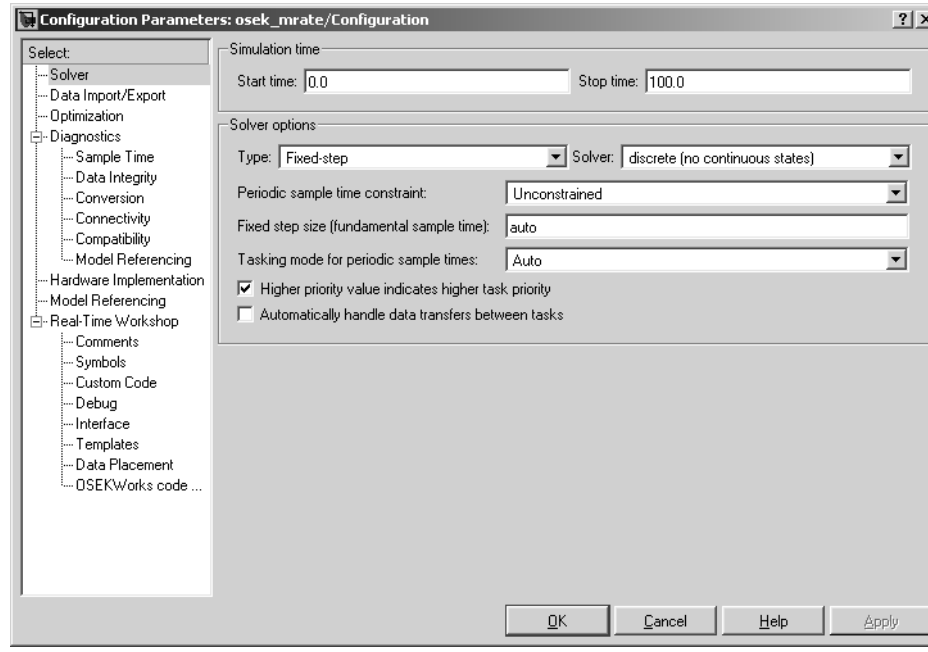
- Set up your development board and connected it to your host PC, as described in "Setting Up Your Target Hardware" on page 2-3.
- Installed OSEKWorks as described in "Setting Up Your Installation for the OSEKWorks Target" on page 2-15.
- Installed SingleStep and configured it as described in "Setting Up SingleStep" on page 2-19.
- Set up your target preferences correctly for OSEKWorks, as described in "Setting Target Preferences" on page 2-11.

## Configuring the Model

1. Open the **Configuration Parameters** dialog and select **Solver**. The **Solver** parameters should be set as follows:
   - **Start time**: `0.0`
   - **Stop time**:`100.0`
   - **Type**: `Fixed-step`
   - **Solver**: `discrete (no continuous states)`

- **Periodic sample time constraint**: `Unconstrained`
- **Fixed step size**: `auto`
- **Tasking mode for periodic sample times**: `Auto`
- **Higher priority value indicates higher task priority**: Check box selected
- **Automatically handle data transfers between tasks**: Check box cleared

The dialog should appear as shown below.

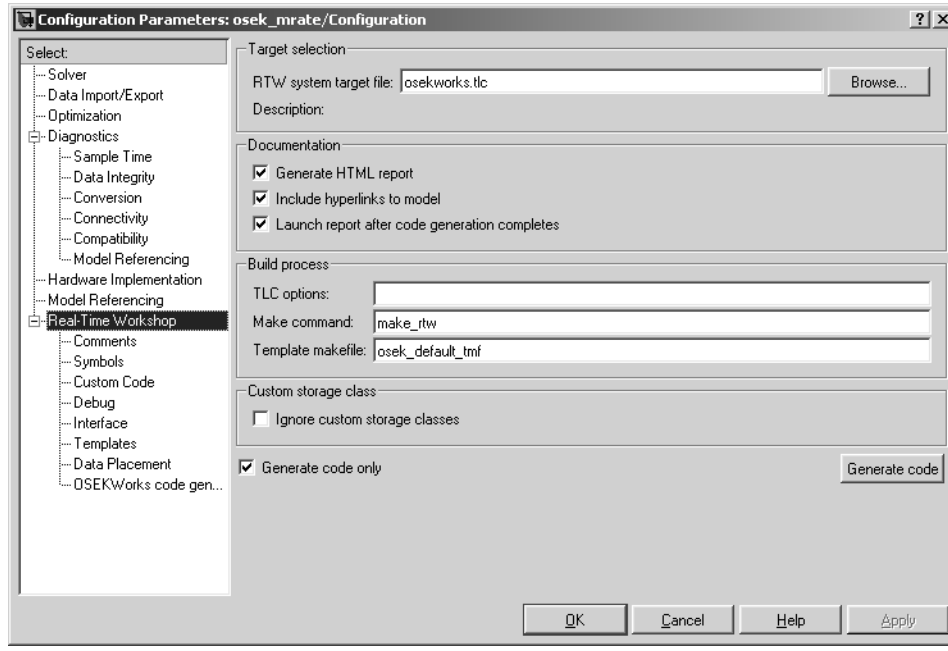



Note the **Tasking mode for periodic sample times** is set to `Auto`. Since the model has three sample rates, this option causes the model to execute in multitasking mode. Code generated from the model causes blocks running at each rate to execute in a separate task.

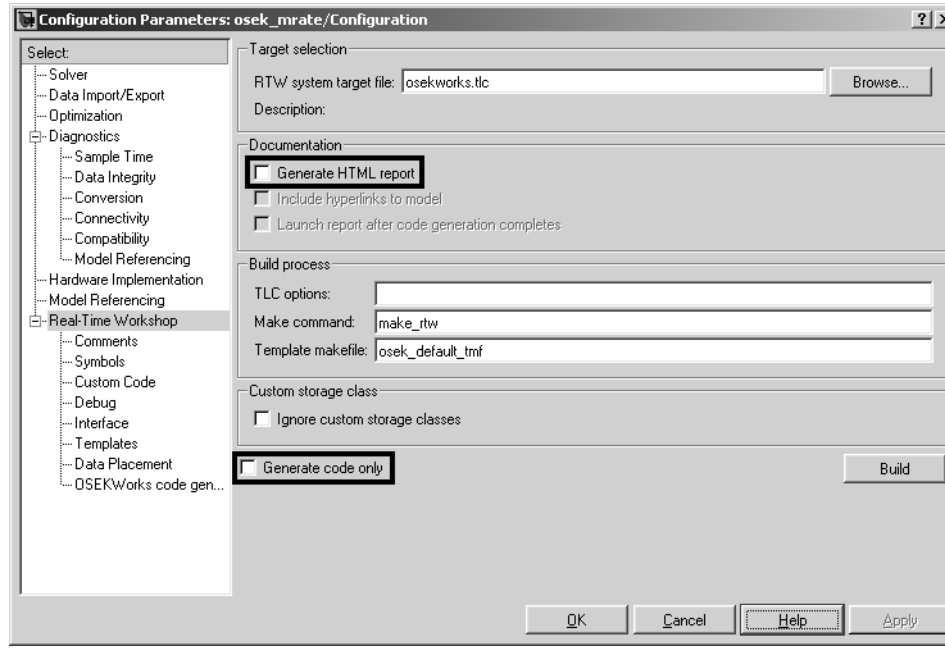
**2** Select **Real-Time Workshop**. The **Real-Time Workshop** pane displays as shown below with the following parameter settings

- **RTW system target file**: `osekworks.tlc`
- **Generate HTML report**: Check box selected

- **Include hyperlinks to model**: Check box selected
- **Launch report after code generator completes**: Check box selected
- **TLC options**: None specified
- **Make command**: `make_rtw`
- **Template makefile**: `osek_default_tmf`
- **Ignore custom storage classes**: Check box cleared
- **Generate code only**: Check box selected

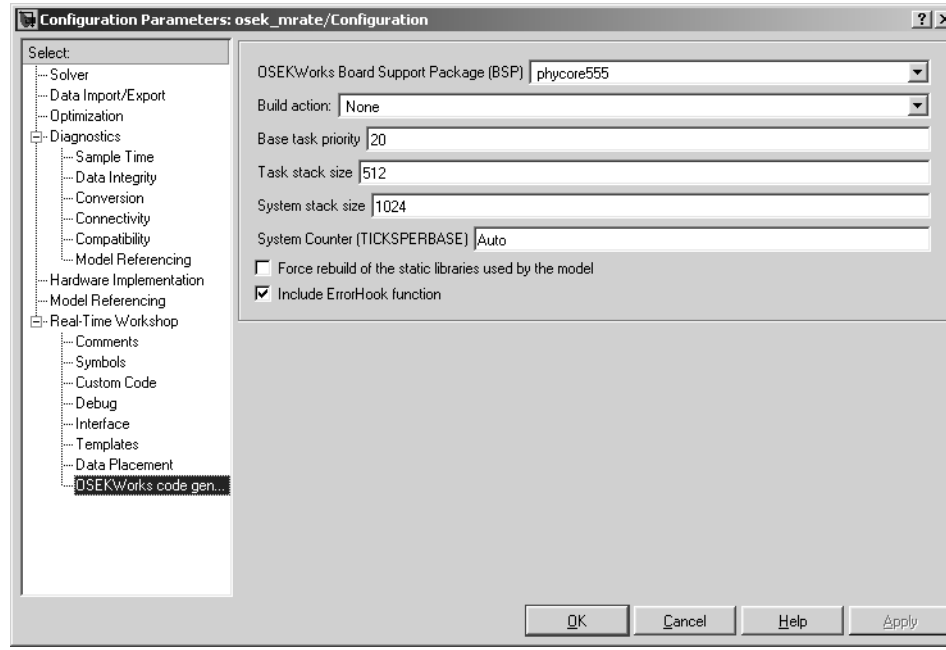



**3** Clear the **Generate HTML report** and **Generate code only** options and click **Apply**. In this tutorial, you do not generate a code generation report, but you do a complete build. The dialog should now appear as shown below. Note that the **Generate** button label changes to **Build**.




**4** Select **OSEKWorks code generation**. The **OSEKWorks code generation** pane appears as follows with the following settings.

- **OSEKWorks Board Support Package (BSP)**: `phycore555`
- **Build action**: `None`
- **Base task priority**: `20`
- **Task stack size**: `512`
- **System stack size**: `1024`
- **System Counter (TICKSPERBASE)**: `Auto`
- **Force rebuild of the static libraries used by the model**: Check box cleared
- **Include ErrorHook function**: Check box selected




The **OSEKWorks Board Support Package (BSP)** menu selection is particularly important for this tutorial. The `phycore555` option is selected, so that the correct hardware-specific support code for the Phytec PhyCORE-MPC555 board is linked into the application.

Also note the **Build action** menu selection is `None`. The **Build** action menu controls whether or not SingleStep is to be invoked at the end of the build process to download and run or debug the generated code. You will manually download and run the generated code, rather than automatically invoke SingleStep, so this option should be set to `None`. See "Tutorial 4: Automated Downloading and Debugging" on page 3-28 for a description of the other **Build action** options.

5 Click **OK** to close the **Configuration Parameters** dialog.

6 Save the model. It is now configured for code generation.

## Building the Application

In this section, you will generate code and build a code module suitable for downloading to the target:

1. Click **Build** on the **Real-Time Workshop** pane to initiate the build process. The build process displays status messages in the MATLAB Command Window.

   On successful completion of the build process, Real-Time Workshop displays the following message.

   ```
   ### Successful completion of Real-Time Workshop build procedure
   for model: osek_mrate
   ```

2. Observe that the build process has created a build directory, `osek_mrate_osekworks`, in your working directory.Use the `dir` command to view the contents of the build directory.

   ```
   dir osek_mrate_osekworks
   ```

   For this model, executable code has been generated in the `phycore555` subdirectory of the build directory. Two executable code files are stored in this directory.

   | | |
   |---|---|
   | `osek_mrate.elf` | Code and symbols, suitable for use with a symbolic debugger such as SingleStep. You will use SingleStep to download this file and execute the code. |
   | `osek_mrate.srec` | Code only (Motorola S-Rec format), without symbols, suitable for execution on the target system. |

   Note that the executables are also copied to the MATLAB working directory (one level above the build directory) for convenience.

   The build process creates a number of other directories and files. For now, you should be concerned with only the executable code that has been generated. See “Build Directories and Files” on page 4-2 for information on the detailed contents of the build directory.

## Downloading and Running the Application

You can now download and execute code on the target hardware. To learn how to do so, proceed to “Tutorial 3: Downloading the Application to RAM via SingleStep” on page 3-21.

# Tutorial 2: Creating an Application with ProOSEK

In this tutorial, you build a real-time multitasking application for ProOSEK from a simple model. You should already be familiar with Simulink and with the Real-Time Workshop code generation and build process.

In the following sections, you

- Configure the model.
- Generate and examine code and build an executable program.
- Download the executable code to a target board, initiate a debugging session, and set breakpoints and observe the execution of the program.

## Before You Begin

This tutorial requires specific hardware and software (as described in "Introduction" on page 3-2) in addition to the Embedded Target for OSEK/VDX. Be sure that you have

- Set up your development board and connected it to your host PC, as described in "Setting Up Your Target Hardware" on page 2-3.
- Installed ProOSEK as described in "Setting Up Your Installation for the ProOSEK Target" on page 2-17.
- Installed SingleStep and configured it as described in "Setting Up SingleStep" on page 2-19.
- Set up your target preferences correctly for ProOSEK, as described in "Setting Target Preferences" on page 2-11

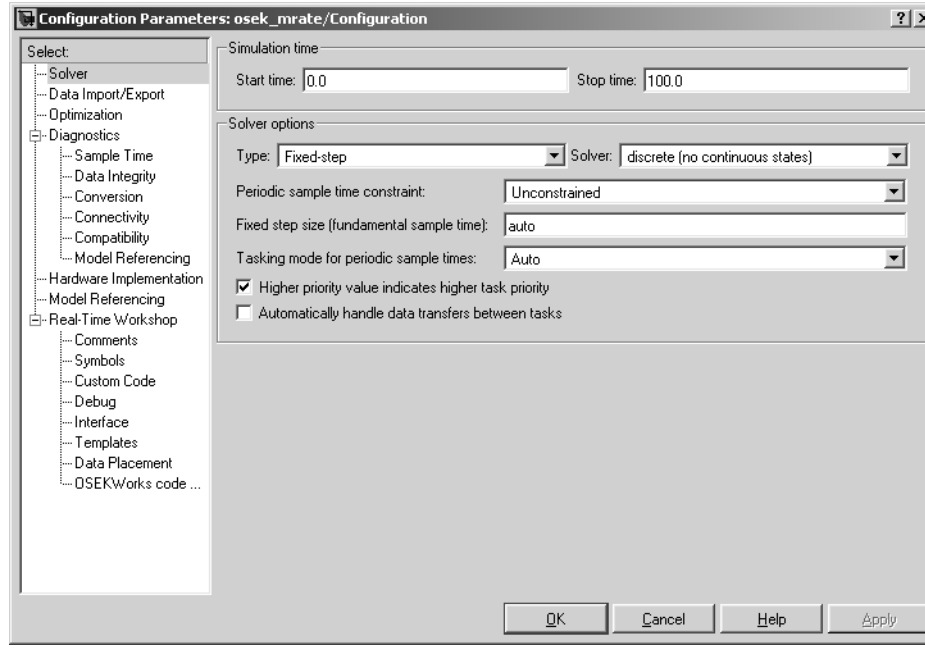
## Configuring the Model

1 Open the **Configurations Parameters** dialog and select the **Solver** pane. The **Solver** parameters should be set as follows:

   - **Start time**: `0.0`
   - **Stop time**: `100.0`
   - **Type**: `Fixed-step`
   - **Solver**: `discrete (no continuous states)`
   - **Periodic sample time constraint**: `Unconstrained`

- **Fixed step size**: `auto`
- **Tasking mode for periodic sample times**: `Auto`
- **Higher priority value indicates higher task priority**: Check box checked
- **Automatically handle data transfers between tasks**: Check box cleared

The dialog should appear as shown below.

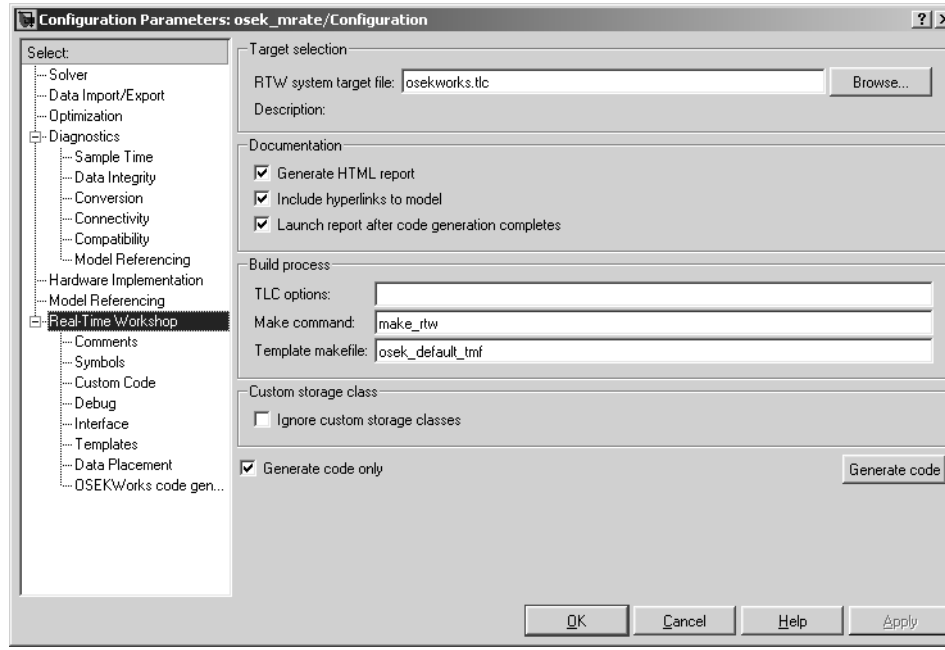



Note the **Tasking mode for periodic sample times** is set to `Auto`. Since the model has three sample rates, this option causes the model to execute in multitasking mode. Code generated from the model causes blocks running at each rate to execute in a separate task.

2 Select **Real-Time Workshop**. The **Real-Time Workshop** pane displays as shown below with the following parameter settings:

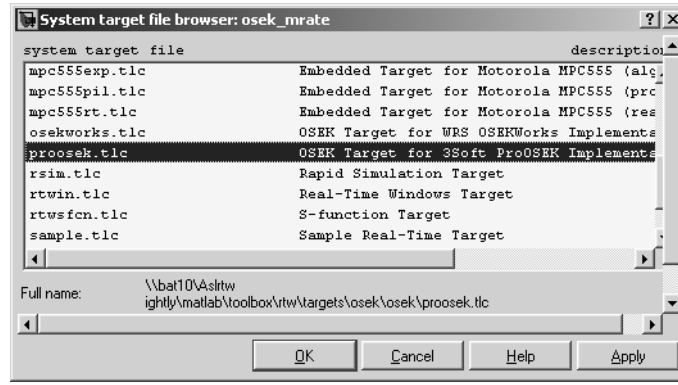
- **RTW system target file**: `osekworks.tlc`
- **Generate HTML report**: Check box selected

- **Include hyperlinks to model**: Check box selected
- **Launch report after code generator completes**: Check box selected
- **TLC options**: None specified
- **Make command**: `make_rtw`
- **Template makefile**: `osek_default_tmf`
- **Ignore custom storage classes**: Check box cleared
- **Generate code only**: Check box selected

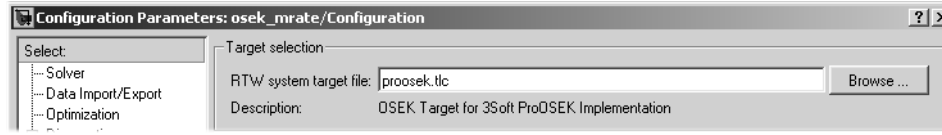



**3** Click **Browse** to open the **System target file browser**.

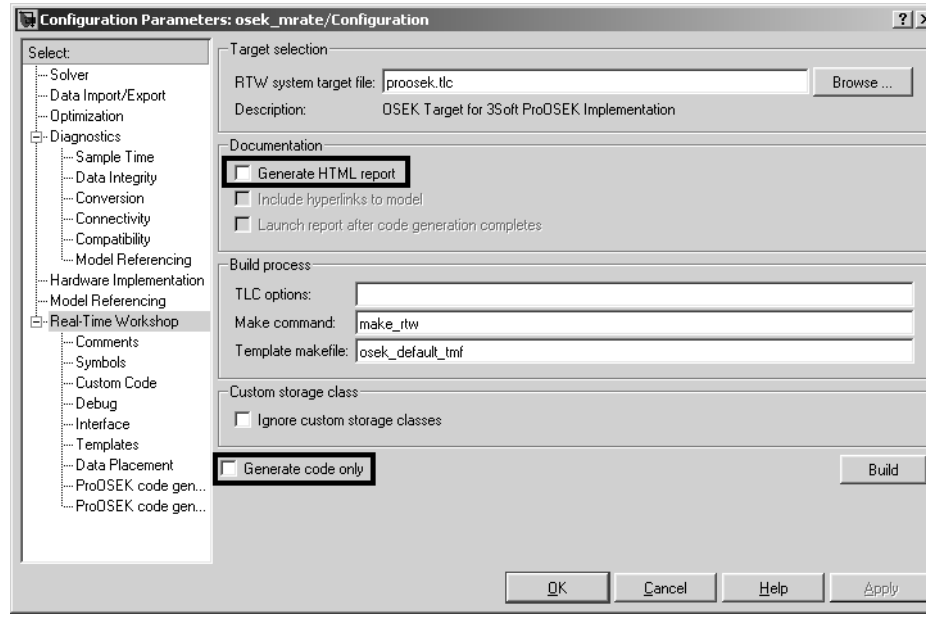
**4** In the browser, select the system target file `proosek.tlc`.




**5** Click **Apply** to register your selection and then click **OK** to close the browser and return to the **Real-Time Workshop** pane. The **Target selection** should now specify `proosek.tlc` as shown below.

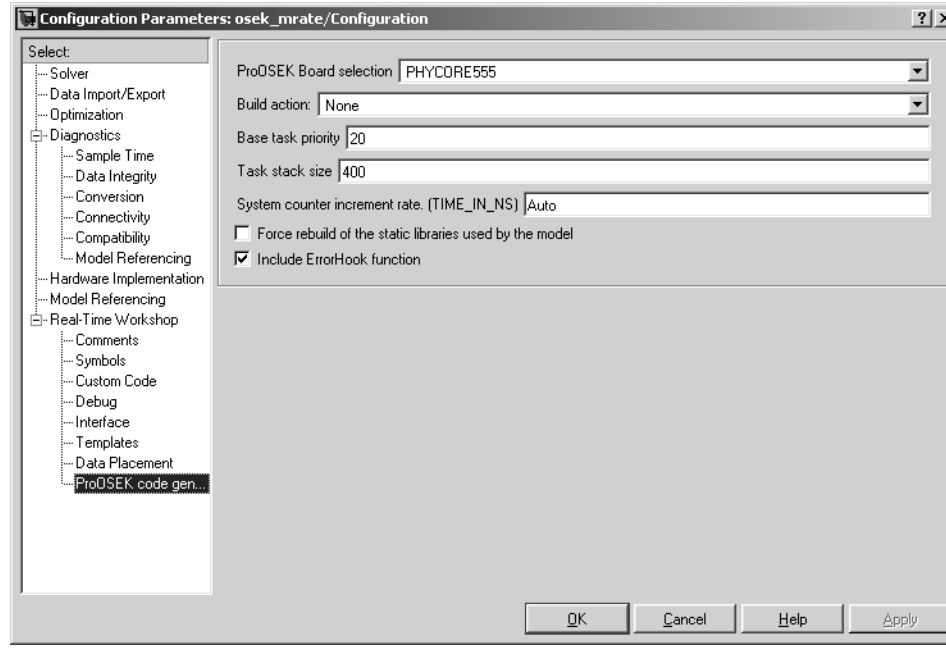



**6** Clear the **Generate HTML report** and **Generate code only** options and click **Apply**. In this tutorial, you do not generate a code generation report, but you do a complete build. The dialog should now appear as shown below. Note that the **Generate** button label changes to **Build**.




*7* Select the **ProOSEK code generation** option. The **ProOSEK code generation** pane appears as follows with the following settings.

- **ProOSEK Board selection**: `PHYCORE555`
- **Build action**: `None`
- **Base task priority**: `20`
- **Task stack size**: `400`
- **System counter increment rate. (TIME_IN_NS)**: `Auto`
- **Force rebuild of the static libraries used by the model**: Check box cleared
- **Include ErrorHook function**: Check box selected




The **ProOSEK Board selection** menu selection is particularly important for this tutorial. The `PHYCORE555` option is selected, so that the correct hardware-specific support code for the Phytec PhyCORE-MPC555 board is linked into the application.

Also note the **Build action** menu selection is `None`. The **Build action** menu controls whether or not SingleStep is to be invoked at the end of the build process to download and run or debug the generated code. You will manually download and run the generated code, rather than automatically invoking SingleStep, so this option should be set to `None`. See "Tutorial 4: Automated Downloading and Debugging" on page 3–28 for a description of the other **Build action** options.

**8** Click **OK** to close the **Configuration Parameters** dialog.

**9** Save the model. It is now configured for code generation.

## Building the Application

In this section, you will generate code and build a code module suitable for downloading to the target:

1. Click the **Build** button on the **Real-Time Workshop** pane to initiate the build process. The build process begins to display status messages in the MATLAB Command Window.

2. On successful completion of the build process, Real-Time Workshop displays the following message.

   ```
   ### Successful completion of Real-Time Workshop build procedure
   for model: osek_mrate
   ```

3. Observe that the build process has created a build directory, `osek_mrate_proosek`, in your working directory. Use the `dir` command to view the contents of the build directory.

   ```
   dir osek_mrate_proosek
   ```

   For this model, executable code has been generated in the `PHYCORE555_obj` subdirectory of the build directory. Two executable code files are stored in this directory:

   | | |
   |---|---|
   | `osek_mrate.elf` | Code and symbols, suitable for use with a symbolic debugger such as SingleStep. You will use SingleStep to download this file and execute the code. |
   | `osek_mrate.srec` | Code only (Motorola S-Rec format), without symbols, suitable for execution on the target system. |

   Note that the executables are also copied to the MATLAB working directory (one level above the build directory) for convenience.

   The build process creates a number of other directories and files. For now, you should be concerned only with the executable code that has been generated. See “Build Directories and Files” on page 4-2 for information on the detailed contents of the build directory.

## Downloading and Running the Application

You can now download and execute code on the target hardware. To learn how to do so, proceed to “Tutorial 3: Downloading the Application to RAM via SingleStep” on page 3-21.

# Tutorial 3: Downloading the Application to RAM via SingleStep

In this section, you download the `osek_mrate.elf` file (generated in the previous tutorial) to RAM on the target system.

It is assumed that the target system is a Phytec PhyCORE-MPC555 board. You will use the SingleStep debugger to download the generated `osek_mrate.elf` file to RAM on the target system, via the BDM port on the target board. You will then initiate a debugging session, set breakpoints, and verify real-time operation of the program.

---

**Note** The SingleStep options and user interface screens discussed below are based on SingleStep versions 7.6.2 and 7.7.x and may differ from your installed version of SingleStep, or with future versions of SingleStep. The MathWorks provides the configuration information below only as a convenience. To resolve questions or difficulties with SingleStep, refer to the SingleStep documentation, or contact Wind River Systems.

---

Make sure you have done the following before you begin:

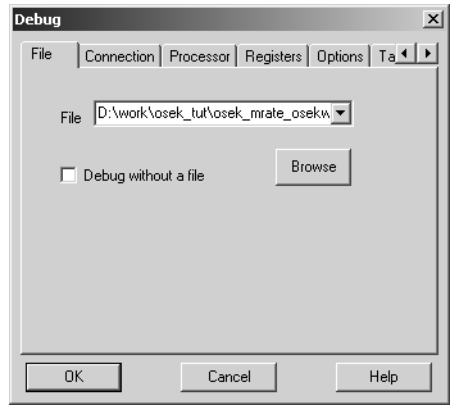
- Configure a shortcut to SingleStep that starts up SingleStep with the correct options (see "Setting Up SingleStep" on page 2-19).
- Connect the BDM port of your development board to parallel port LPT1 of your host PC.
- Make sure that the jumpers on the phyCORE-MPC555 board are set as described in "Jumper Settings" on page 2-3.
- Cycle the power (or perform a hard reset) on your development board to set the board to a known state.

## Downloading the Generated Code to RAM

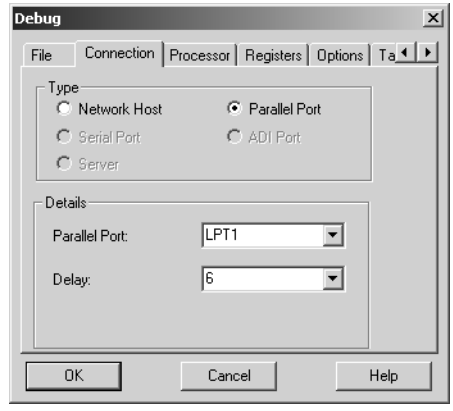
To download the generated `osek_mrate.elf` file to RAM:

1. Start SingleStep using the `SingleStep On Chip (MPC5xx) for OSEK Target` shortcut you created previously (see "Configuring SingleStep On-Chip 7.6.2" on page 2-19).

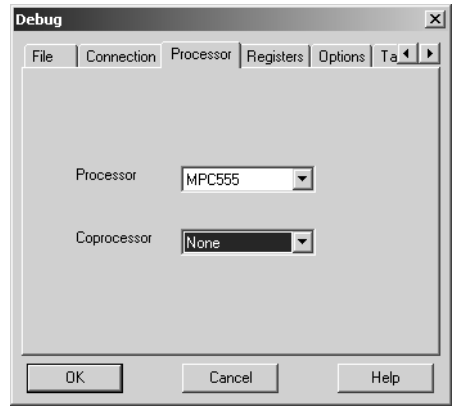
2 The **Debug** dialog opens. Click the **File** tab. Clear the **Debug without a file** check box. Then, use the **Browse** button to locate the `osek_mrate.elf` file



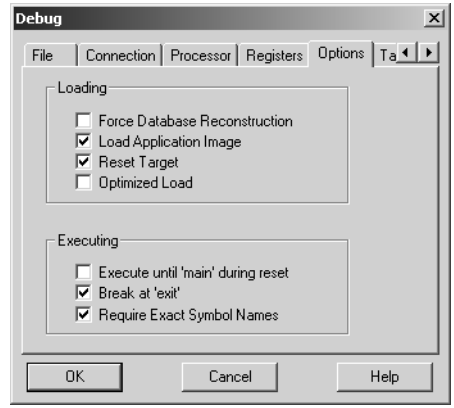

3 Click the **Connection** tab. Choose parallel port or network settings appropriate to the physical connection you will be using between your PC and PhyCORE-MPC555 board. As shown below, connection options are configured for the parallel port `LPT1`.




4 Click the **Processor** tab. Confirm that the `MPC555` is selected in the **Processor** list, as shown.

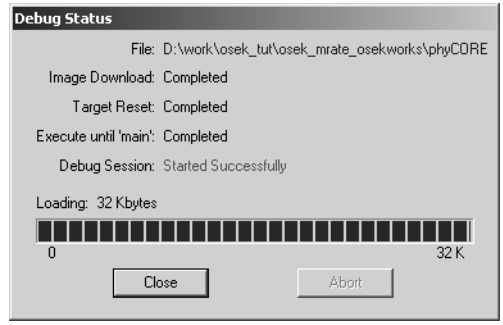



**5** Click the **Options** tab. Make sure that the **Reset Target** and **Load Application Image** check boxes are selected, as shown.




**6** Use the default options for all other tabs.

**7** Click **OK.** SingleStep attempts to connect to the processor, and displays a **Debug Status** window. This figure shows the **Debug Status** window after a successful connection and download.




If you see error messages, you may need to adjust the **Delay** setting on the **Connection** pane (see step 2 above) complete a successful download. If errors persist, consult the SingleStep documentation to troubleshoot the connection, or contact Wind River Systems for technical support.

8 Click **Close** to dismiss the **Debug Status** window.

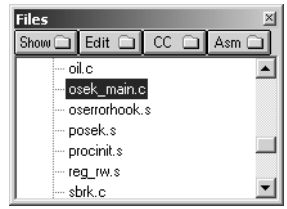
9 At this point, the application code is in RAM, and SingleStep has established a debugging session.

In the next section you will use SingleStep to examine the operation of the generated code as it executes on the target board.

## Observing the Generated Code

In this section, you will display the generated main program in SingleStep, set a breakpoint, and observe the timing of the slowest (3 Hz) sample rate of our model:

1 Locate the arrow labeled **Files** at the bottom of the SingleStep **Debug** window. Click this arrow to display the **Files** selection pane. Select `osek_main.c` from the list of files, as shown below.




**2** Double-click the selected list element. The source code for `osek_main.c` is displayed in the **Debug** window.

**3** The generated code creates several OSEK/VDX tasks. The `init` task in the figure below is activated once during the OSEK/VDX startup phase. The `init` calls `model_initialize` and creates three recurrent OSEK/VDX alarms:

- `baseAlarm`: OSEK/VDX triggers `baseAlarm` at intervals of 1 second (i.e., at the model's base rate).
- `subAlarm_1`: OSEK/VDX triggers `subAlarm_1` at intervals of 2 seconds.
- `subAlarm_2`: OSEK/VDX triggers `subAlarm_2` at intervals of 3 seconds.

```
19
20 TASK(init)
21 {
22   /* Initialize model */
23   osek_mrate_initialize(1);
24
25   /* Base rate will run every           : 1.0 seconds
26      Original rate specified in model   : 1.0 seconds */
27   SetRelAlarm(baseAlarm, 1, 1);
28
29   /* Sub rate will run every            : 2.0 seconds,
30      Original rate specified in model   : 2.0 seconds */
31   SetRelAlarm(subAlarm_1, 1, 2);
32
33   /* Sub rate will run every            : 3.0 seconds,
34      Original rate specified in model   : 3.0 seconds */
35   SetRelAlarm(subAlarm_2, 1, 3);
36
37   TerminateTask();
38 }
39
```

**4** The generated code also creates three OSEK/VDX tasks, as shown below, which are activated by each of the three alarms, and therefore run at the corresponding rate:

- The `baseAlarm` task activates the `baseRate` task. The `baseRate` task calls `model_step`, passing in task identifier (`tid`) 0, indicating that blocks running at the model's base rate should execute.

- The `subAlarm_1` task activates the `subRate_1` task. The `subRate_1` calls `model_step`, passing in `tid` 1, indicating that blocks running at the 2 Hz rate should execute.
- The `subAlarm_2` task activates the `subRate_2` task. The `subRate_2` calls `model_step`, passing in `tid` 1, indicating that blocks running at the 3 Hz rate should execute.

```
39
40 /* Using RTW multitasking execution of model */
41 TASK(baseRate)
42 {
43   /* Set model inputs associated with base rate here */
44   osek_mrate_step(0);
45   /* Get model outputs associated with base rate here */
46
47   TerminateTask();
48 }
49
50 TASK(subRate_1)
51 {
52   /* Set model inputs associated with sub rate here */
53   osek_mrate_step(1);
54   /* Get model outputs associated with sub rate here */
55
56   TerminateTask();
57 }
58
59 TASK(subRate_2)
60 {
61   /* Set model inputs associated with sub rate here */
62   osek_mrate_step(2);
63   /* Get model outputs associated with sub rate here */
64
65   TerminateTask();
66 }
67
```

**5** Set a breakpoint at the beginning of the `subRate_2` task, as shown in the preceding figure. You will now verify that the `subRate_2` task executes at the correct interval (every 3 seconds).

**6** Move the cursor into the main text pane of the **Debug** window. Then press the F5 key (equivalent to the green **Go** arrow) to start program execution. An hourglass cursor is displayed momentarily. Initial activation of the `subRate_2` task occurs almost immediately, and execution stops at the breakpoint.

**7** Subsequent activations of the `subRate_2` task occur every 3 seconds. You can verify this by pressing the F5 key again to resume execution from the breakpoint. Observe that the hourglass cursor is displayed for 3 seconds before execution stops at the breakpoint again. This will occur each time you resume execution.

**8** Select **Exit** from the **File** menu of the SingleStep window to close the debugging session.

In the next tutorial, “Tutorial 4: Automated Downloading and Debugging” on page 3–28, you will work with the automatic downloading and debugging features of the Embedded Target for OSEK/VDX.

# Tutorial 4: Automated Downloading and Debugging

Both the OSEKWorks target and the ProOSEK target let you automatically download generated applications, with the option of initiating a debugging session. You must have the SingleStep debugger to use these features.

The **Build action** menu supports the following options:

| | |
|---|---|
| `Download_and_run` | Invoke SingleStep after the build process to download the executable to target RAM and start execution. |
| `Download_and_debug` | Invoke SingleStep after the build process to download the executable to target RAM and start a debugging session. |
| `None` | SingleStep is not invoked. You must download and run or debug the code manually. |

In this tutorial, you rebuild the `osek_mrate` application used in the previous tutorials, and use the `Download_and_debug` option to download the code and start a debugging session.

Make sure you have done the following before you begin:

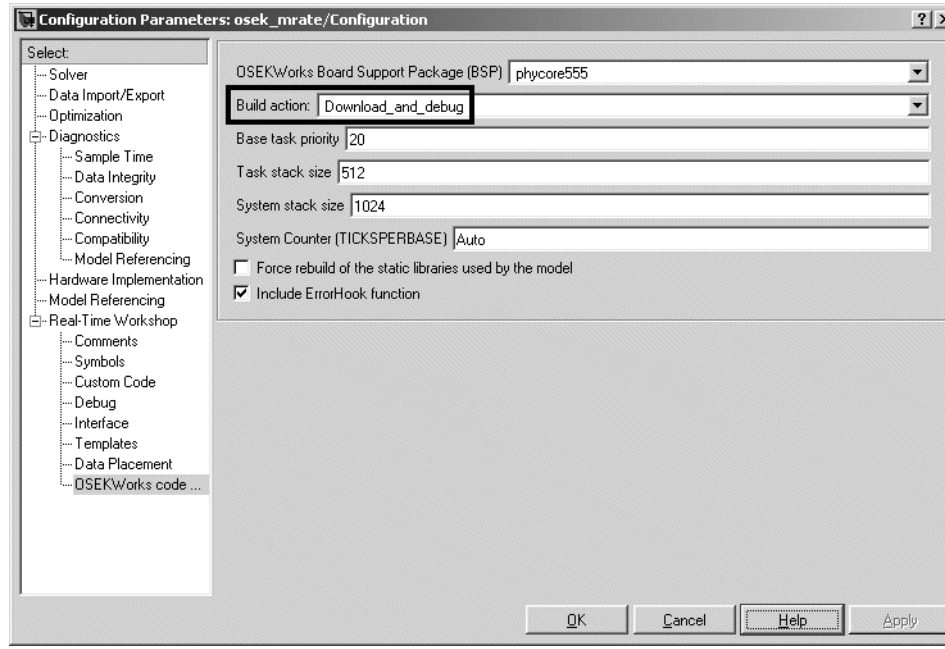
- Set the debugger-related target properties (`Debugger`, `DebuggerEXE`, `DebuggerPath`, `Debugger`, `DebuggerSwitches`) correctly for your SingleStep installation, as described in "Setting Target Preferences" on page 2-11.
- Cycle the power (or perform a hard reset) on your development board to set the board to a known state:

After completing the preceding tasks, do the following.

1. Open the **Real-Time Workshop** pane of the **Configurations Parameters** dialog and select the target-specific options for your implementation (`OSEKWorks code generation options` or `ProOSEK code generation options`).

2. Select the **OSEKWorks code generation** or **ProOSEK code generation** option.

3. Select `Download_and_debug` from the **Build action** menu and click **Apply**. The figure below shows this option selected for the OSEKWorks target




4 Click **Real-Time Workshop** to return to the **Real-Time Workshop** pane.

5 Click **Build** to initiate the build process. The build process begins to display status messages in the MATLAB Command Window.

6 On successful completion of the build process, Real-Time Workshop displays the following messages, indicating that SingleStep has been started up.

```
Execute SingleStep as: start
\\depot\hub\share\apps\WindRiver\SingleStepDebugger\sds762\cmd\bdmp58.exe  -P  -S
D:\Work\r12\toolbox\rtw\targets\osek\osek\@osek_diab_tgtaction\phycore-555.wsp -a
d:\work\osek_tut\osek_mrate.elf -r
d:\work\osek_tut\osek_mrate_osekworks\osek_mrate_ram.scr
### Successful completion of Real-Time Workshop build procedure for model:
osek_mrate
```

7 SingleStep displays an initial splash screen. After a few seconds, the SingleStep **Debug** window is displayed, with the program counter arrow pointing at the first executable instruction.

8 The executable is now downloaded to the target, and ready to execute under control of SingleStep. You can now conduct a SingleStep debugging session, or simply start the program.

9 Select **Exit** from the **File** menu of the SingleStep window to close the debugging session.

You may also want to try setting **Build action** to `Download_and_run`. If so, proceed as follows:

1 Close any existing SingleStep sessions, as multiple SingleStep sessions can conflict with each other.

2 Return to the **OSEKWorks code generation** pane.

3 Select `Download_and_run` for the **Build action** option and click **Apply**.

4 Continue starting at step 4 in the preceding procedure. After completion of the build process, SingleStep starts up and downloads and runs the program on the target, without a breakpoint or any manual intervention.

In the next and final tutorial, "Tutorial 5: Downloading Generated Code to FLASH" on page 3-31, you use a different model to generate code and download it to FLASH rather than RAM.

# Tutorial 5: Downloading Generated Code to FLASH

In this tutorial, you generate code from a different model than that used in the previous tutorials. You will generate, download, and run the generated program in FLASH rather than RAM.

Before you begin, make sure that you have applied any required patches or extensions required for your version of SingleStep, as described in one of the following sections:

- "Configuring SingleStep On-Chip 7.6.2" on page 2-19
- "Configuring SingleStep with Vision" on page 2-20

## The osek_led Demo Model

Since you will be executing the generated code from FLASH, you will not be using an interactive debugging session to observe program execution. The `osek_led` demo model is suitable for this tutorial because it can be started via the Reset button on the target, and because it produces an observable result directly on the target hardware. Set up the model as follows:

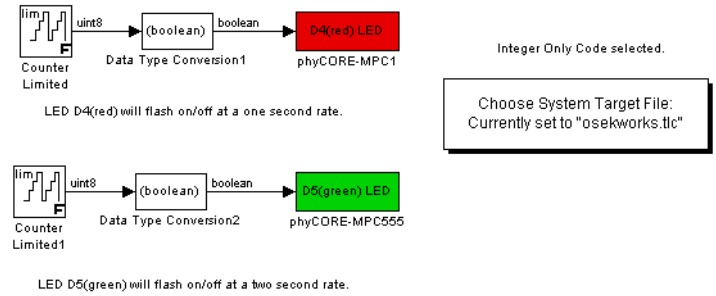
1. Open the model. If you are reading this document online in the MATLAB Help browser, you can open the model by clicking on the link `osek_led`.

   Alternatively, type the model name at the MATLAB command line.

   ```
   osek_led
   ```

2. Save a local copy of the `osek_led` model to your working directory. You will work with this copy throughout this exercise.

3. The `osek_led` model uses two counters and two device driver blocks to toggle two of the LEDs on the PhyCORE-MPC555 board at different rates. The model is shown in this figure.




4 Before building the model, open the **Configuration Parameters** dialog and open the **Real-Time Workshop** pane.

5 Clear the **Generate code only** check box. You may also want to clear **Generate HTML report**.

6 Click **Apply**.

7 Click **Build**. The build process for the Embedded Target for OSEK/VDX creates executables in several formats. These files are created in your working directory:

- `osek_led.bin`: A FLASH executable for use with SingleStep on-Chip (7.6.2)
- `osek_led.srec:` A Motorola S-Rec file used to prepare a FLASH binary for use with SingleStep with vision (7.7.3)
- `osek_led.elf`: Suitable for downloading and execution in RAM

8 Your next step depends on which version of SingleStep you are using:

- SingleStep On-Chip 7.6.2: Proceed to “Downloading Generated Code to FLASH with SingleStep On-Chip 7.6.2” on page 3-33.
- SingleStep with vision: Proceed to “Downloading Generated Code to FLASH with SingleStep with vision” on page 3-39.

# Downloading Generated Code to FLASH with SingleStep On-Chip 7.6.2

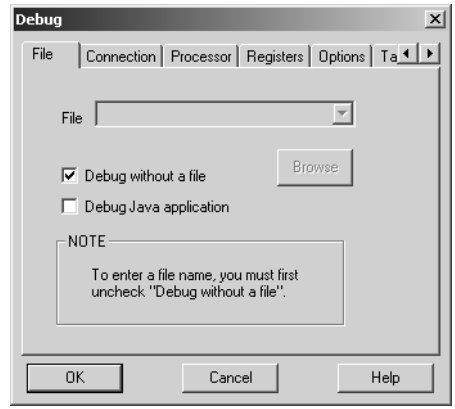
This section describes how to download and debug the generated `osek_led.bin` file to FLASH memory on the target, via SingleStep On-Chip 7.6.2.

## Connect to Target With FLASH Enabled

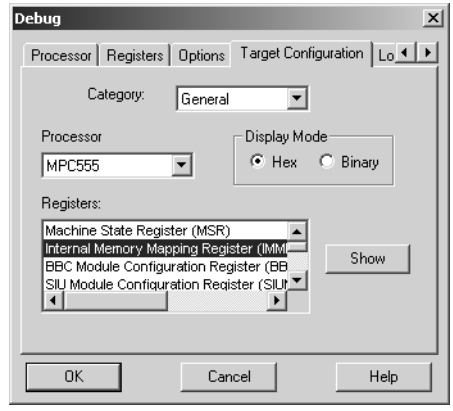
Before programming `osek_led.bin` into FLASH, set the FLASH Enable (`FLEN`) bit on the target. In this section, you use SingleStep for this purpose:

1. Start SingleStep using the `SingleStep On Chip (MPC5xx) for OSEK Target` shortcut you created previously (see "Configuring SingleStep On-Chip 7.6.2" on page 2-19).

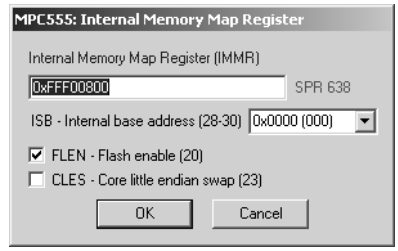
2. The **Debug** dialog opens. Click the **File** tab. Select the **Debug without a file** check box, as shown.




3. Click the **Target Configuration** tab and select `General` from the **Category** menu, as shown below. Then select `Internal Memory Mapping Register` from the **Registers** list, as shown.




**4** Click **Show** to open the **Internal Memory Mapping Register** dialog. Confirm that the **FLEN-flash enable** check box is selected as shown below.



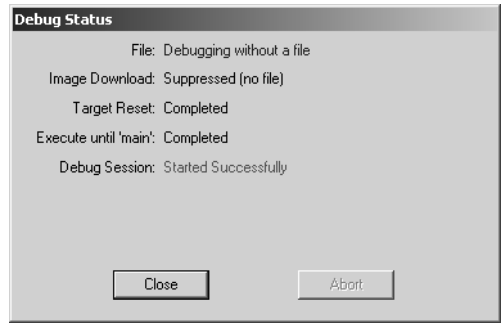

**5** Then click **OK** to close the **Internal Memory Mapping Register** dialog and return to the **Debug** dialog.

---

**Note** Make sure to click **OK**, not **Cancel**, or SingleStep may use settings other than those shown in the dialog.

---

**6** Click **OK**. SingleStep attempts to connect to the processor, and displays a **Debug Status** window. This figure shows the **Debug Status** window after a successful connection.




If you see error messages, consult the SingleStep documentation to troubleshoot the connection, or contact Wind River Systems for technical support.

**7** Click **Close** to dismiss the **Debug Status** window.

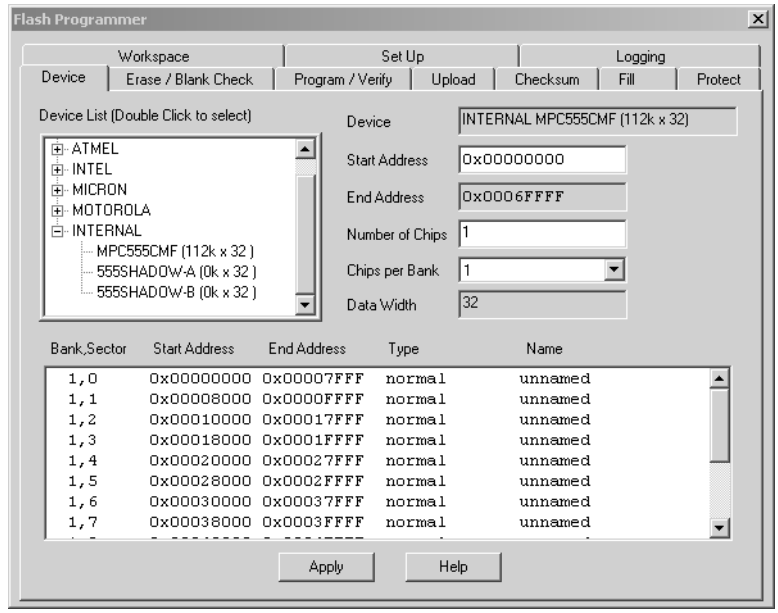
## Download Code and Execute in FLASH Memory

The next step is to download the generated code (`osek_led.bin`) to FLASH memory, using the SingleStep **Flash Programmer** dialog, and start execution on the target board:

**1** Activate the main SingleStep window. If you do not see a **Flash** button in the toolbar, select **Tools** from the **ToolBars** menu.

**2** Click **Flash** on the toolbar. The **Flash Programmer** dialog opens.

**3** Click the **Device** tab. In the **Device List**, double-click `INTERNAL`. Then select and double-click `MPC555CMF`.

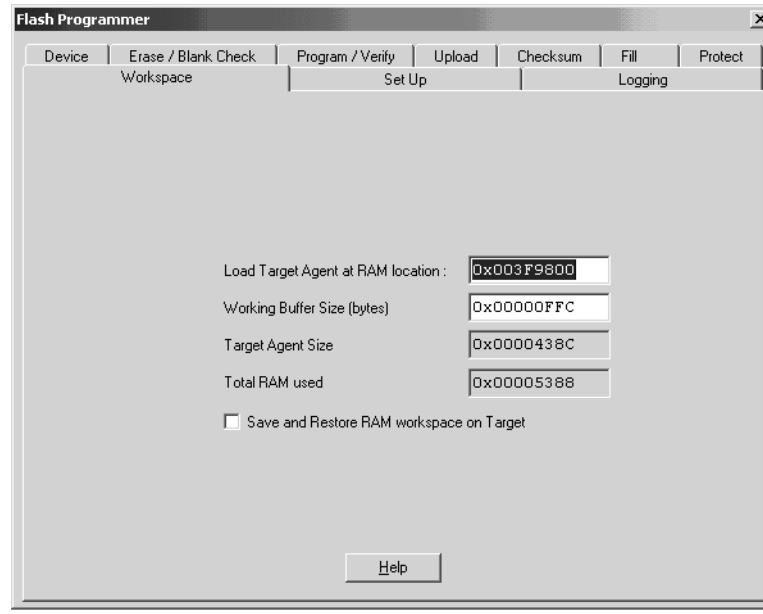



4 Make sure that the **Start Address** field is set to `0x00000000`, as shown.

5 Click **Apply**.

6 Click the **Workspace** tab. Make sure that the following fields are set:

- **Load Target Agent**: `0x003F9800`
- **Working Buffer Size**: `0x00000FFC`




**7** Click the **Program/Verify** tab. Specify the full path to the generated `osek_led.bin` file in the **S-Record or Binary Image File** field. You can do this either by navigating to the file via the **Browse** button, or by entering the path and filename into the field.

**8** The next step is to set the load address of the program in FLASH memory via the **Start** fields in the **Location** panel. The OSEKWorks and ProOSEK targets generate code requiring different load addresses.

If you generated code using the OSEKWorks target, set the **Start** fields as follows:

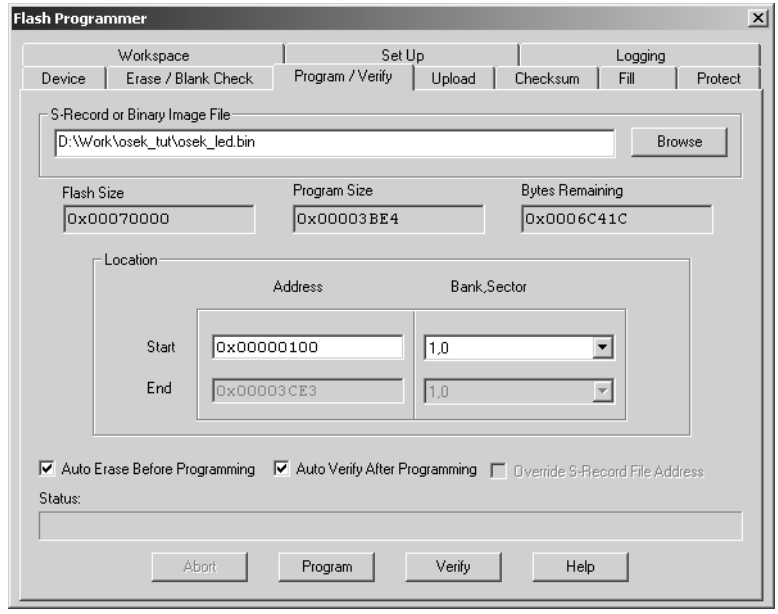
- **Address**: `0x0000100`
- **Bank, Sector**: select `1,0`

If you generated code using the ProOSEK target, set the **Start** fields as follows:

- **Address**: `0x0000000`
- **Bank, Sector**: select `1,0`

**9** Select the **Auto Erase Before Programming** and **Auto Verify After Programming** options.

**10** The **Program/Verify** settings should now appear similar to the following figure.




**11** Click the **Program** button. The code is downloaded. During downloading, a number of progress messages are displayed in the **Status** panel at the bottom of the dialog.

**12** Upon completing the download process, SingleStep displays a message box indicating successful completion. Click **OK** to dismiss the message box. Then, close the **Flash Programmer** dialog. Do not save changes to the **Flash Programmer** when the **Save** dialog appears.

**13** If FLASH programming fails, you should

- Check that all jumpers are set correctly as described in “Jumper Settings” on page 2-3.

- Quit SingleStep and repeat the entire procedure starting at "Connect to Target With FLASH Enabled" on page 3-33. Make sure, in step 5, that you click **OK** (not **Cancel**) when closing the **Internal Memory Mapping Register** dialog. Otherwise, SingleStep may use settings other than those shown in the dialog.
- If errors persist, consult the SingleStep documentation to troubleshoot the connection, or contact Wind River Systems for technical support.

**14** The `osek_led` code has now been programmed into FLASH. To execute the code, press the **Reset** button on the target board. Alternatively, cycle power on the target board.

**15** Observe that LED D4(red) blinks at 1 second intervals, and LED D5(green) blinks every other second.

## Downloading Generated Code to FLASH with SingleStep with vision

This section describes how to convert the generated `osek_led.srec` file into a binary executable (`.bin` file) and download the executable to FLASH memory on the target, via SingleStep with vision.

Before you begin, make sure you have installed the extensions described in "Install SingleStep 7.7.3 and apply patches for programming FLASH memory." on page 2-22.

### Convert S-Rec File to Binary Executable File

When using SingleStep with vision to download code to FLASH, you must convert the `.srec` file generated by the build process into a special binary format. To do this, use the `osektgtaction` utility provided by the Embedded Target for OSEK/VDX.

At the MATLAB command line, use the following syntax

```
osektgtaction('srectoestbin','model.srec')
```

where *`model`* is the name of the `.srec` file you want to convert - in this case, `osek_led.srec`.

The `srectoestbin` argument directs the `osektgtaction` utility to run the SingleStep `convert.exe` utility to produce an intermediate binary file. A

second SingleStep utility, `mpc555fc.exe`, generates a final FLASH-compatible binary, *`model`*`_est.bin`. The final `.bin` file has a default starting address of `0x00000000`.

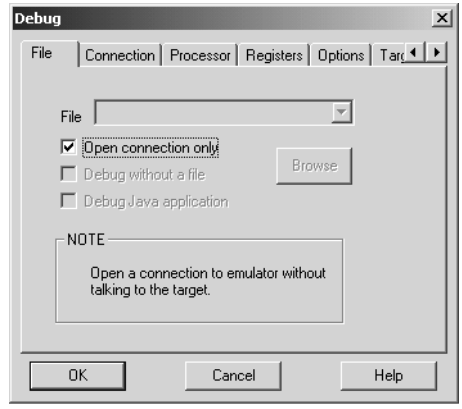
In this case, the final binary file is `osek_led_est.bin`.

### Connect to Target with FLASH Enabled

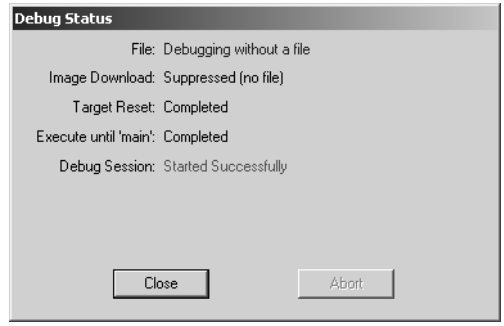
Before programming `osek_led_est.bin` into FLASH, you must set the FLASH Enable (`FLEN`) bit on the target, and make sure that the Software Watchdog Timer is disabled. In this section, you will use SingleStep for this purpose:

1. Start SingleStep using the `SingleStep with vision (MPC5xx) for OSEK Target` shortcut you created previously (see "Configuring SingleStep with Vision" on page 2-20).

2. The **Debug** dialog opens. Click the **File** tab. Select the **Open connection only** check box, as shown.




3. Click **OK**. SingleStep attempts to connect to the processor, and displays a **Debug Status** window. This figure shows the **Debug Status** window after a successful connection.

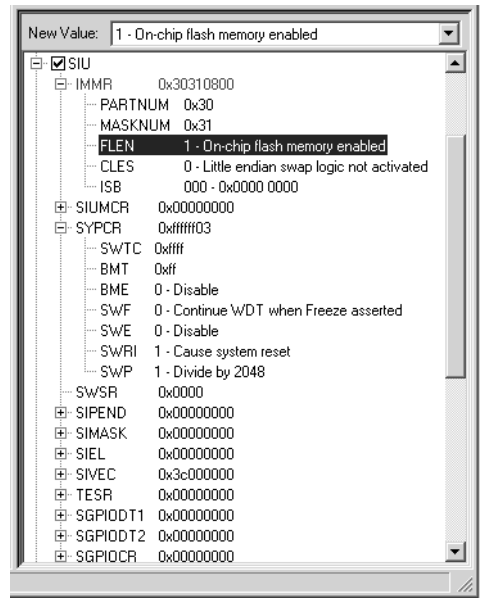



If you see error messages, consult the SingleStep documentation to troubleshoot the connection, or contact Wind River Systems for technical support.

4 Click **Close** to dismiss the **Debug Status** window.

5 A connection to the target now exists. From the **Windows** menu in the SingleStep toolbar, select the **Debug** window.

6 In the **Register** pane of the **Debug** window, scroll and use the `+/-` tree view controls to display the `IMMR:FLEN` and `SYPCR:SWE` fields, as shown in this figure.




**7** In the **Register** pane, set the following fields as shown:

- `SIU:IMMR:FLEN` (Flash Enable) bit to `1`
- `SIU:SYPCR:SWE` (Software Watchdog Enable) bit to `0`

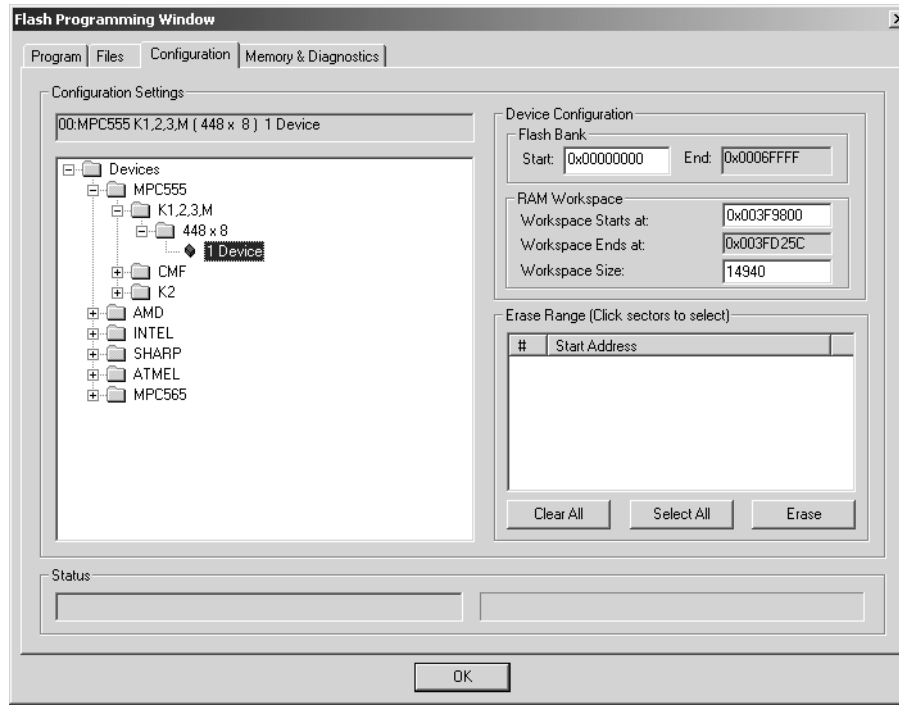
FLASH memory is now enabled. In the next section, you will download and execute the `.bin` file.

## Download Code and Execute in FLASH Memory

The next step is to download the generated code (`osek_led_est.bin`) to FLASH memory, using the SingleStep **Vision Flash Utility**, and start execution on the target board:

**1** Select **Vision Flash Utility** from the **Tools** menu in the SingleStep toolbar.

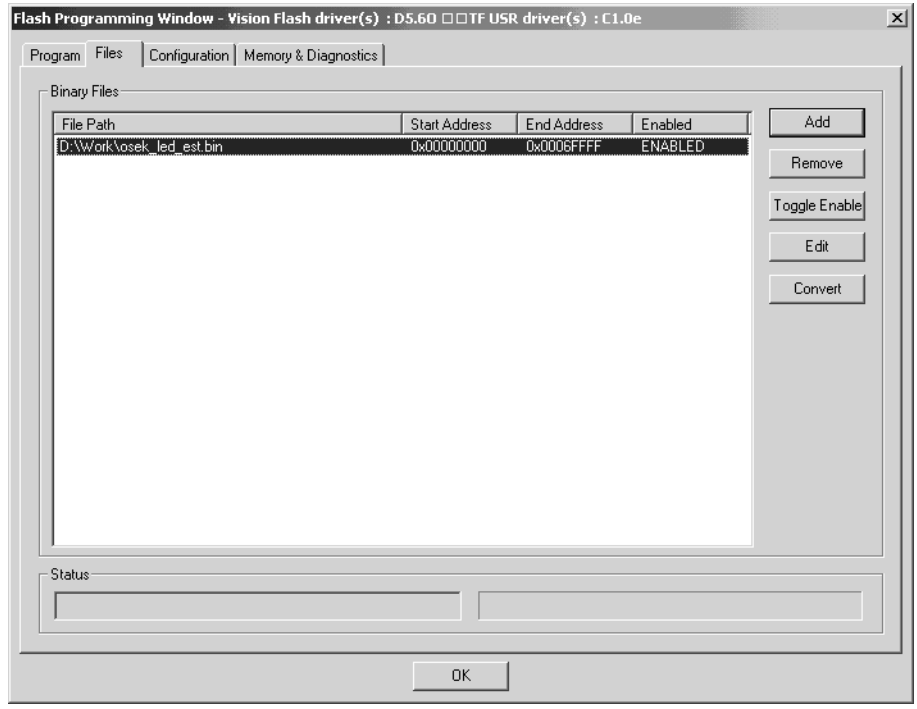
**2** Select the **Configuration** tab in the **Flash Programming** window**.**




**3** In the **Configuration** pane, set `Devices:MPC555:K1,2,3,M:448x8:1 Device`, as shown above.

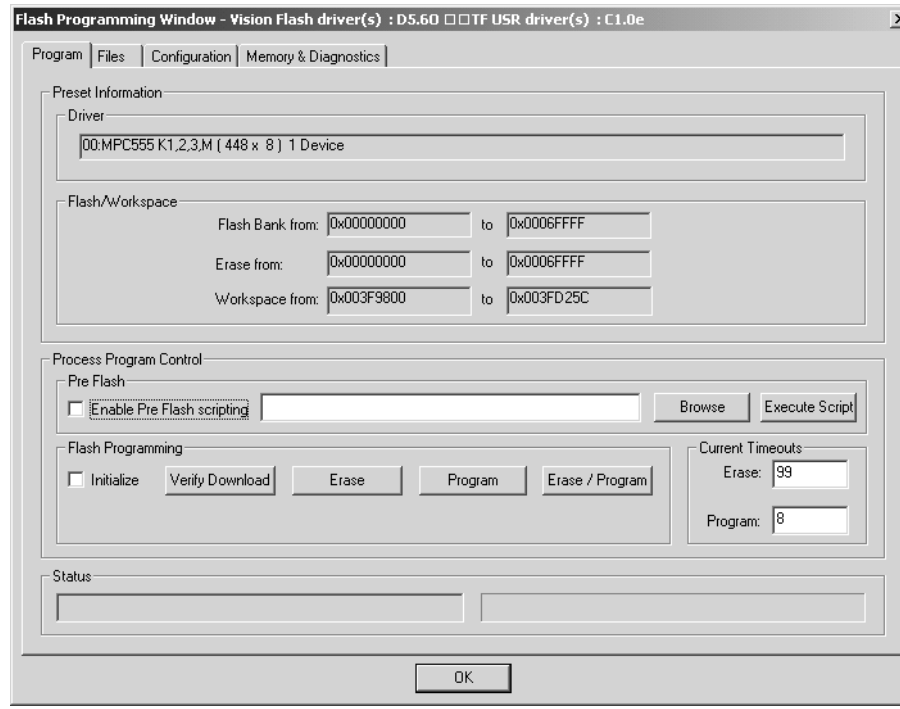
If this selection is not available, the installation of SingleStep 7.7.3 extensions was probably not successful. See “Configuring SingleStep with Vision” on page 2–20 and make sure that the FLASH programming extensions are installed correctly. If problems persist, contact Wind River Systems for technical support.

**4** Select the **Files** tab in the **Flash Programming** window. Use the **Add** button to navigate to the generated `osek_led_est.bin` file in the build directory. After finding the `.bin` file, use the **Open** button to place this file in the **Binary files** list.




5 Select `osek_led_est.bin` file in the **Binary Files** list. Click **Toggle Enable**.

6 Select the **Program** tab in the **Flash Programming Window**.




**7** Click the **Erase** button to be sure the FLASH is erased before programming.

**8** Click the **Program** button to program the binary executable into FLASH.

**9** Close the **Flash Programming Window**.

**10** If FLASH programming fails, you should

- Verify that the jumpers are set correctly as described in “Jumper Settings” on page 2-3.
- Quit SingleStep and repeat the entire procedure starting at “Connect to Target with FLASH Enabled” on page 3-40. Make sure, in step 8, that your settings are applied correctly. Otherwise, SingleStep may use settings other than those shown in the dialog.

- If errors persist, consult the SingleStep documentation to troubleshoot the connection, or contact Wind River Systems for technical support.

**11** The `osek_led` code has now been programmed into FLASH. To execute the code, press **Reset** on the target board. Alternatively, cycle power on the target board.

**12** Observe that LED D4(red) blinks at 1 second intervals, and LED D5(green) blinks every other second.

4

# Generating Code, Calibration Data, and Reports

# Build Directories and Files

The build directory structure and files created by the Embedded Target for OSEK/VDX build process differ slightly from the standard Real-Time Workshop Embedded Coder build directories. Figure 4-1 summarizes the directories and files created during the build process.

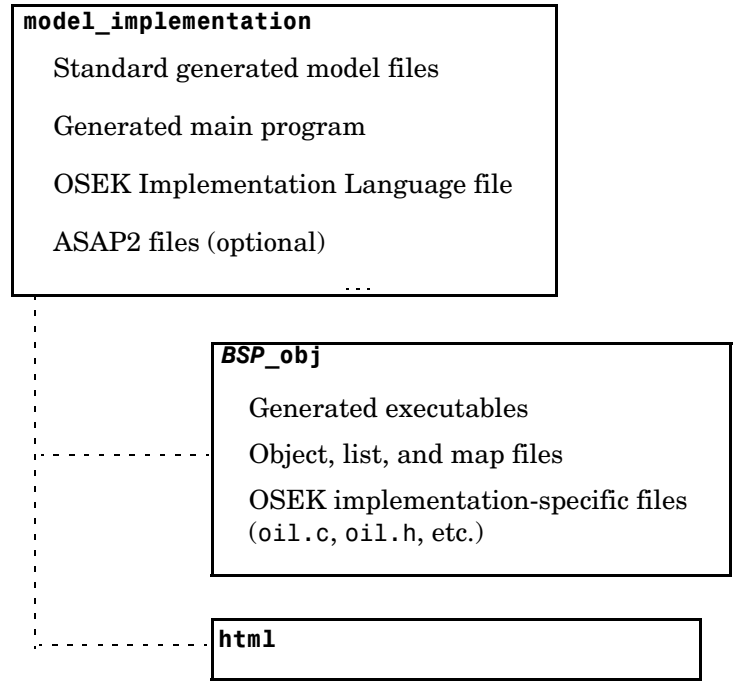



**Figure 4-1: Directories and Files Created by Build Process**

## Build Directory (model_implementation)

The top-level build directory is created in your working directory, using the naming convention `model_implementation`, where `model` is the name of the generating model and `implementation` is name of the selected OSEK/VDX implementation. For example, `mymodel_osekworks` would be the build

directory name where the OSEKWorks target generates code for the OSEKWorks implementation from a model `mymodel`.

The build directory contains

- Standard generated files including source code (*`model`*`.c`, *`model`*`.h`), makefile (*`model`*`.mk`), *`model`*`.bat` file and others as described in the "Data Structures and Code Modules" section of the Real-Time Workshop Embedded Coder documentation.
- OSEK Implementation Language (OIL) file (*`model`*`.oil`) defining OSEK/VDX system objects such as tasks, alarms, and counters.
- Generated main program (`osek_main.c`) that invokes the standard OSEK/VDX kernel startup (`StartOS`) function. The main program also defines model execution tasks and other tasks (such as system initialization) that are activated under control of OSEK/VDX. The format in which tasks are defined is dependent on the OSEK/VDX implementation.
- ASAP2 files are generated if you select the **Generate ASAP2 file** option.

## Output Subdirectory (BSP_obj)

This subdirectory contains the final output of the build process and includes the executable code files as well as other files produced by the build process. Usually, you will need to access only the executable, in order to download it to the target hardware.

Executables are named after the generating model, with a file extension indicating the file format (e.g., *`model`*`.elf` or *`model`*`.srec`). The executable format is determined by your development system.

Note that the *`model`*`.elf`, *`model`*`.srec`, and *`model`*`.map` files are copied to the MATLAB working directory (one level above the build directory). This conforms to Real-Time Workshop Embedded Coder conventions and provides easier access to these files.

The naming convention for the output subdirectory is `BSP_obj`, where `BSP` is the name of the board support package (BSP) selected. (For the OSEKWorks target, the BSP is selected from the **OSEKWorks Board Support Package (BSP)** menu. For the ProOSEK target, the BSP is selected from the **ProOSEK Board selection** menu.) For example, if the `phycore555` BSP is selected, the output directory is named `phycore555_obj`. This convention is adopted because

the final executable contains hardware-specific code linked in from the selected BSP.

This directory also contains C files (`oil.c, oil.h`) derived from the `model.oil` file. Additional artifacts of the build process, such as object and list files and a linker map file, are also located in this directory.

## HTML Report Subdirectory (optional)

This directory is created if the **Generate HTML report** option is selected (see "Code Generation Reports" on page 4-17). It contains the HTML code generation report files.

# Code Generation Options

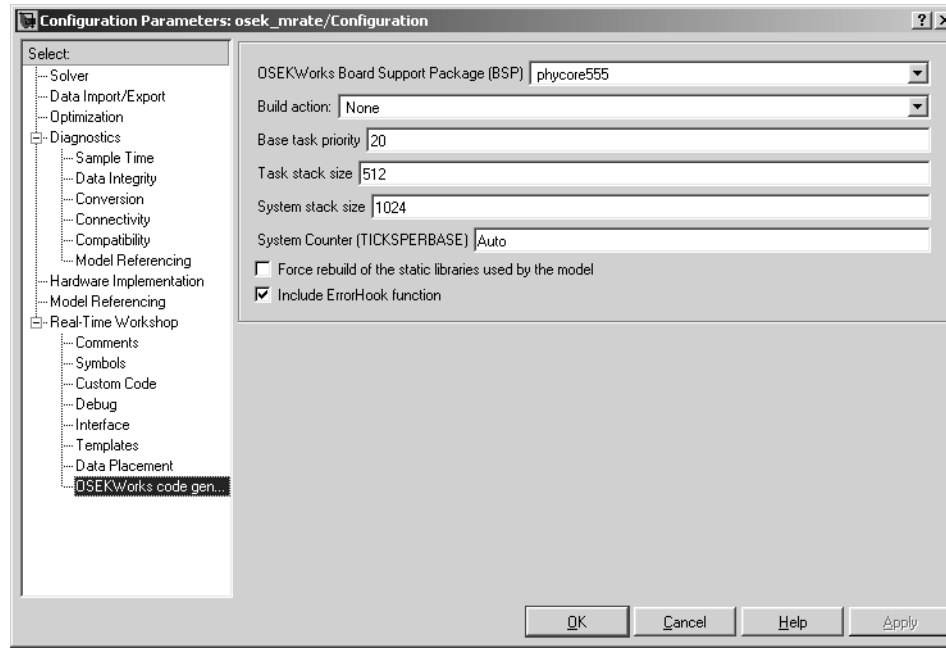
The Embedded Target for OSEK/VDX is an extension of the Real-Time Workshop Embedded Coder embedded real-time (ERT) target configuration. The Embedded Target for OSEK/VDX inherits the code generation options of the ERT target, as well as the general code generation options of the Real-Time Workshop. These options are available from the **Real-Time Workshop** pane of the **Configuration Parameters** dialog; 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 Embedded Target for OSEK/VDX, and are unsupported or restricted in their operation. See “Restrictions on Code Generation Options” on page 4-12 for details.

## Target-Specific Options for OSEKWorks Target

The OSEKWorks Target has two panes of target-specific code generation options that are available from the **Real-Time Workshop** pane of the **Configuration Parameters** dialog. To view or change the settings of these options,

1. Open the **Configuration Parameters** dialog.
2. Select **Real-Time Workshop** to open the **Real-Time Workshop** configuration pane.
3. Click **Browse** and select `osekworks.tlc` from the **System target file browser**. An OSEKworks code generation option appears under **Real-Time Workshop** in the selection list.
4. Click **OSEKWorks code generation**. The **OSEKWorks code generation** pane appears as shown below.




- **OSEKWorks Board Support Package (BSP)**: Specifies a supported BSP for use in code generation. The default is `phycore555`.
- **Build action:** Controls what action, if any, the build process takes after the target executable is created. The options are

| | |
|---|---|
| `Download_and_run` | Invoke SingleStep to download the executable to target RAM and start execution. |
| `Download_and_debug` | Invoke SingleStep to download the executable to target RAM and start a debugging session. |
| `None` (default) | SingleStep is not invoked. You must download and run or debug the code manually. |

It is possible to invoke a debugger other than SingleStep to execute build actions. See “Custom Debugger Support” on page 4-11 for guidelines on how to do this.

- **Base task priority**: Assigns a priority to the base rate (fastest) task in the model. Each OSEK/VDX task is assigned a priority from 0 to 255, with higher numbers signifying higher priority. Subrate tasks are assigned successively lower numbers. The default is `20`.
- **Task stack size**: Allocates a stack size, in bytes, to each task in the model. The default is `512`. (See "Setting System and Task Stack Size" on page 4–10.)
- **System stack size**: Allocates a stack size, in bytes, to the OSEK/VDX kernel. The default is `1024`. (See "Setting System and Task Stack Size" on page 4–10.)
- **System Counter (TICKSPERBASE)**: Specifies the number of ticks per second for the OSEK/VDX system counter. (This parameter sets the `TICKSPERBASE` property of the `SystemTimer` object.)

  You can specify **System Counter** as an integer or as `Auto`. If you specify `Auto`, the Embedded Target for OSEK/VDX determines a minimum value for `TICKSPERBASE`, thus minimizing interrupt and counter overhead. `Auto` is the default value.

  Note that `TICKSPERBASE` also affects the frequency of rescheduling that is attempted on return from Category 2 Interrupt Service Routines (ISRs), because the system counter is a Category 2 ISR.
- **Force rebuild of the static libraries used by the model**: Controls whether object file libraries (such as `rtwlib`) referenced by the model are rebuilt during the build process. By default, this option is cleared, and existing object libraries are not rebuilt. Since rebuilding such libraries may involve compiling a large number of source files, we recommend the default. See "Efficient Use of Persistent Object Libraries" on page 4-10 for further information.
- **Include `ErrorHook` function**: Generates an OSEK `ErrorHook` function. The function originates from the file `osekerrorhook.tlc` in `matlabroot/toolbox/rtw/targets/osek/osek` and is generated in `osek_main.c`. You can change the behavior of the `ErrorHook` function by modifying `osekerrorhook.tlc`. By default, this option is selected.

## Target-Specific Options for ProOSEK Target

The ProOSEK Target has two panes of target-specific code generation options that are available from the **Real-Time Workshop** pane of the **Configuration Parameters** dialog. To view or change the setting of these options,

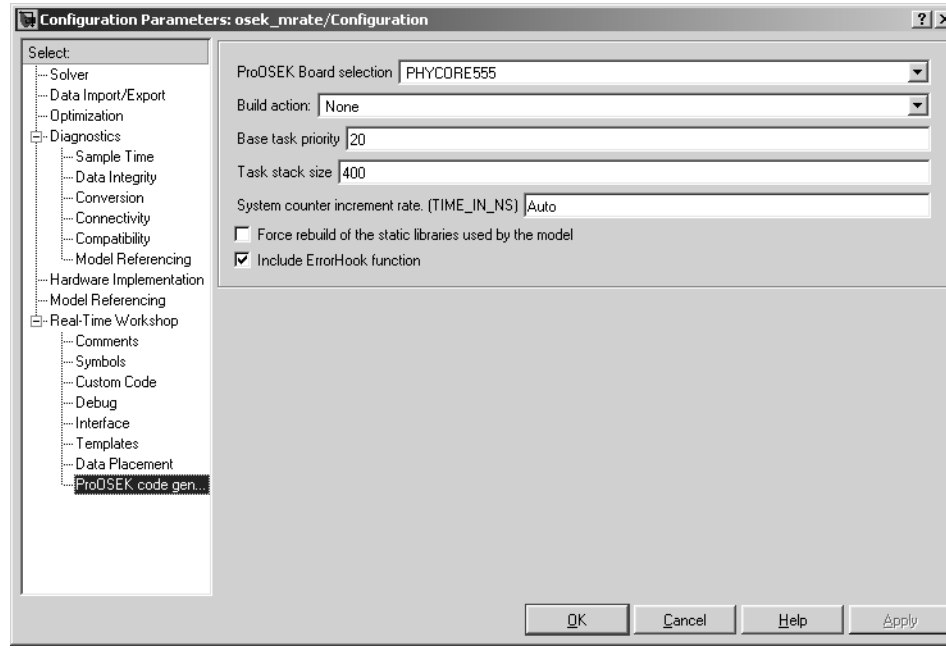
1 Open the **Configuration Parameters** dialog.

2 Select **Real-Time Workshop** to open the **Real-Time Workshop** configuration pane.

3 Click **Browse** and select `proosek.tlc` from the **System Target File Browser**. A **ProOSEK code generation** option appears under **Real-Time Workshop** in the selection list.

4 Click **ProOSEK code generation**. The **ProOSEK code generation** pane appears as shown below.

## Options for ProOSEK code generation options (1)




- **ProOSEK Board selection**: Specifies a supported BSP for use in code generation. The default is `PHYCORE555`.

- **Build action:** Controls what action, if any, the build process takes after the target executable is created. The options are

  | | |
  |---|---|
  | `Download_and_run` | Invoke SingleStep to download the executable to target RAM and start execution. |
  | `Download_and_debug` | Invoke SingleStep to download the executable to target RAM and start a debugging session. |
  | `None` (default) | SingleStep is not invoked. You must download and run or debug the code manually. |

  It is possible to invoke a debugger other than SingleStep to execute build actions. See "Custom Debugger Support" on page 4-11 for guidelines on how to do this.
- **Base task priority**: Assigns a priority to the base rate (fastest) task in the model. Each OSEK/VDX task is assigned a priority from 0 to 255, with higher numbers signifying higher priority. Subrate tasks are assigned successively lower numbers. The default is 20.
- **Task stack size**: Allocates a stack size, in bytes, to each task in the model. The default is 400. (See "Setting System and Task Stack Size" on page 4–10.)
- **System counter increment rate (TIME_IN_NS)**: Specifies the period, in nanoseconds, for the OSEK/VDX system counter.

  You can specify **System counter** as an integer or as `Auto`. If you specify `Auto`, the Embedded Target for OSEK/VDX determines a maximum value for `TIME_IN_NS`, thus minimizing interrupt and counter overhead. The default is `Auto`.
- **Force rebuild of the static libraries used by the model**: Controls whether object file libraries (such as `rtwlib`) referenced by the model are rebuilt during the build process. By default, this option is cleared, and existing object libraries are not rebuilt. Since rebuilding such libraries may involve compiling a large number of source files, the default is recommended. See "Efficient Use of Persistent Object Libraries" on page 4-10 for further information.
- **Include ErrorHook function**: Generates an OSEK `ErrorHook` function. The function originates from the file `osekerrorhook.tlc` in `matlabroot/toolbox/rtw/targets/osek/osek` and is generated in

`osek_main.c`. You can change the behavior of the `ErrorHook` function by modifying `osekerrorhook.tlc`. By default, this option is selected.

## Setting System and Task Stack Size

The stack sizes allocated to the kernel and the application tasks are defined, in the OIL file, by the OSEK implementation. The OSEK implementation may optimize total memory required for all stacks by sharing memory based on application-specific constraints. The stacks are then statically allocated by the OIL configuration process and the build process.

Both the OSEKWorks and ProOSEK targets let you specify the task stack size. In addition, the OSEKWorks target lets you specify the system stack size.

When you set the **System stack size** or **Task stack size** parameters, it is important to consider factors that affect stack usage at run time. The **System stack size** requirements can be affected by

- The nesting and type of interrupts
- The number and type of tasks in the application
- OSEK operating system API calls made at runtime

Similarly, **Task stack size** requirements can be affected by

- Some types of interrupts
- OSEK operating system API calls made at runtime
- Calls to application functions

Consult your OSEK implementation's documentation to understand and determine the worst case system and task stack size requirements. Set the **System stack size** and/or **Task stack size** parameters accordingly.

## Efficient Use of Persistent Object Libraries

Because rebuilding object libraries can involve compilation of large numbers of source code files, it is desirable to minimize the rebuilding of such libraries when possible. The Embedded Target for OSEK/VDX provides flexible mechanisms that let you control how and when object libraries are rebuilt.

The template makefiles provided with the Embedded Target for OSEK/VDX have the ability to create and use persistent object libraries associated with

- The Real-Time Workshop library, `rtwlib (rtw/c/libsrc)`
- Certain blocksets that provide a `rtwmakecfg.m` file that specifies that a related library may be persistent

The template makefiles manage these libraries through make macros and through token expansion within the bounds of the token pair.

```
|>START_PRECOMP_LIBRARIES<|
.
.
.
|>END_PRECOMP_LIBRARIES<|
```

The template makefiles use the `StaticLibraryDirectory` target preference property (see "Setting Target Preferences" on page 2-11). The value of the `StaticLibraryDirectory` property propagates (on a per-model/per-build basis) to the make macro `STATIC_LIBDIR` in the *`model`*`_makevars.mk` file.

When `STATIC_LIBDIR` is not empty, it should contain the path to an existing directory where persistent object libraries are stored. When the compilation of a model refers to a persistent object library, the build process will use a library from this location, or if the library does not exist, will create it there.

The **Force rebuild of the static libraries used by the model** option will cause all such libraries to be rebuilt, even if they already exist.

## Custom Debugger Support

This section provides general guidelines for supporting a debugger other than SingleStep for use in the build process. Implementing custom debugger support requires knowledge of MATLAB object-oriented programming. See "Classes and Objects" in the MATLAB programming documentation if you are unfamiliar with this topic.

The **Build action** options (described in "Code Generation Options" on page 4-5) are supported by MATLAB OOPS classes and by the `Debugger` target preferences (see "Target Preference Properties" on page 2-11).

By default, the `Debugger` target preference is set to `'SingleStep'`. When `'SingleStep'` is selected, the build process invokes methods of the `osek_singlestep_tgtaction` class to execute the required build actions, such as starting SingleStep and downloading a generated executable. The

implementation files for the `osek_singlestep_tgtaction` class and the package containing it located in

```
matlabroot/toolbox/rtw/targets/osek/osek/@osek_singlestep_tgtaction
```

An alternate value for the `Debugger` target preference is `'Custom'`. This value is provided as a mechanism to invoke an alternative debugger or downloading utility. When `'Custom'` is selected, the build process expects that a user-defined package `osek_custom_tgtaction`, containing a class `osek_custom_tgtaction`, exists.

You must implement the `osek_custom_tgtaction` package and class. We suggest that you start by studying the code in the `@osek_singlestep_tgtaction` directory to understand how build actions are supported with SingleStep.

Next, copy the entire `@osek_singlestep_tgtaction` directory to a new directory:

```
matlabroot/toolbox/rtw/targets/osek/osek/@osek_custom_tgtaction
```

In the `@osek_custom_tgtaction` directory, rename `osek_singlestep_tgtaction.m` to `osek_custom_tgtaction.m`. Edit `osek_custom_tgtaction.m`, changing every occurrence of `'osek_singlestep_tgtaction'` to `'osek_custom_tgtaction'`.

At this point, you will have implemented a skeletal `osek_custom_tgtaction` package that will function identically to the `osek_singlestep_tgtaction` package, where the `Debugger` target preference is set to `'Custom'`.

You must now customize the `run` and `debug` methods (implemented respectively in `run.m` and `debug.m`) in your new package directory. We cannot prescribe the exact changes you must make; these modifications depend on the requirements of your debugger or the other tools you want to invoke.

## Restrictions on Code Generation Options

Some ERT code generation options are not supported (or are restricted) by the Embedded Target for OSEK/VDX. The following unsupported options appear grayed out in the dialog displays and you cannot select them:

- **Suppress error status in real-time model data structure**

- **Create Simulink (S-Function) block**
- **MAT-file logging**
- **File customization template**
- **Generate an example main program**

If you select the following **Interface** pane options, the Embedded Target for OSEK/VDX ignores the option or issues an error message during the build process.

| Option | Restriction |
|---|---|
| `ISO_C` specified for **Target floating point math environment** under **Software environment** | For ProOSEK target only, error if selected; build process terminates |
| **Generate reusable code** under **Code interface** | Error if selected; build process terminates |
| **External mode** specified for **Interface** menu under **Data Exchange** on the **Interface** pane | Error if selected; build process terminates |

# Generating ASAP2 Files

ASAP2 is a data definition standard proposed by the Association for Standardization of Automation and Measuring Systems (ASAM). ASAP2 is a standard description you use for data measurement, calibration, and diagnostic systems. The Embedded Target for OSEK/VDX lets you export an ASAP2 file containing information about your model during the code generation process.

Before you begin generating ASAP2 files with the Embedded Target for OSEK/VDX, you should read "Generating ASAP2 Files" in the Real-Time Workshop 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 Embedded Target for OSEK/VDX is similar to that described in the Real-Time Workshop Embedded Coder documentation. However, there are certain differences and limitations. In the following sections, we describe these differences and limitations and how they affect the procedure for generating ASAP2 files.

The `osek_asap2` demo provides an example of the Embedded Target for OSEK/VDX ASAP2 file generation feature.

## Compiler-Specific Post-Processing Requirements

The Embedded Target for OSEK/VDX 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 a compiler-generated MAP file must be created.

After the build process, if the Embedded Target for OSEK/VDX 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. The Embedded Target for OSEK/VDX provides a post-processing mechanism for the compilers supplied with each supported OSEK implementation.

The names of the ASAP2 file and the MAP file derive from the source model. The MAP file is generated in the output subdirectory and copied to the working directory (see "Build Directories and Files" on page 4-2). The ASAP2 file is written to the build directory.

## ASAP2 File Generation Procedure

To generate a model's data definition in ASAP2 format, using the Real-Time Workshop Embedded Coder,

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

2. Define the desired parameters and signals in the model to be `ASAP2.Parameter` and `ASAP2.Signal` objects in the MATLAB workspace.

3. For each ASAP2 data object, configure the `RTWInfo StorageClass` property to be one of the following:

   - `ExportedGlobal`
   - `ImportedExtern`
   - `ImportedExternPointer`

   This configures the data objects so that their corresponding declarations in the generated code are unstructured global storage declarations.

4. Configure the remaining properties as desired for each ASAP2 data object.

5. On the **Optimization** pane of the **Configuration Parameters** dialog, select the **Inline parameters** check.

   Note that you should *not* configure the parameters associated with your data objects as global (tunable) parameters in the **Model Parameter Configuration** dialog box. 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 box to configure other parameters in your model.

6. On the **Real-Time Workshop** pane, click **Browse** to open the **System target file** browser. In the browser, select any `Embedded Real-Time Target` and click **OK**.

7 In the **Interface** field on the **Interface** pane, select `ASAP2`.

8 Select the **Generate code only** check box on the **Real-Time Workshop** pane.

9 Click **Apply**.

10 Click **Generate code**.

For more information on generating ASAP2 files, see “Generating ASAP2 Files” in the Real-Time Workshop documentation.

# Code Generation Reports

The Embedded Target for OSEK/VDX supports an extended version of the Real-Time Workshop Embedded Coder HTML code generation report.

The extended code generation report consists of several sections:

- The **Generated Source Files** section of the Contents pane contains a table of source code files generated from your model. You can view the source code in the MATLAB Help browser. Hyperlinks within the displayed source code let you view the blocks or subsystems from which the code was generated. Click on the hyperlinks to view the relevant blocks or subsystems in a Simulink model window.

  In addition to the standard information and hyperlinks to generated code, the report generated by The Embedded Target for OSEK/VDX includes links to the following generated files:

  - *`model`*`.oil`: OSEK Implementation Language (OIL) file.
  - `oil.c, oil.h`: C definitions and includes derived from the OIL file.
- The **Summary** section lists version and date information, Target Language Compiler (TLC) options used in code generation, and Simulink model settings.
- The **Optimizations** section lists the optimizations used during the build, and also those that are available. If you chose options that generated less than optimal code, they are marked in red. This section can help you select options that will better optimize your code.
- The report also includes information on other code generation options, code dependencies, and links to relevant documentation.
- 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.

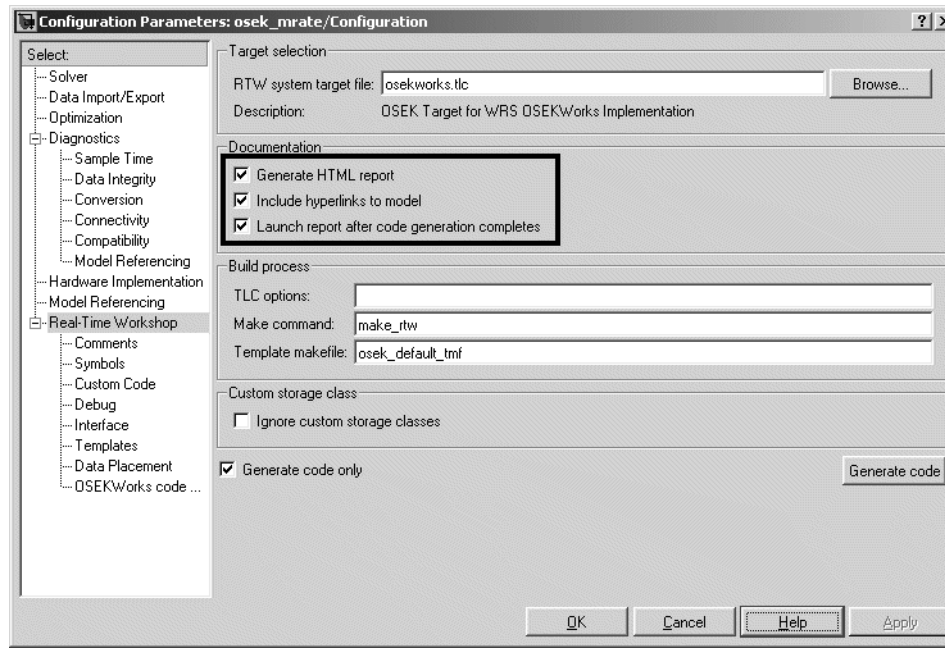
To generate a code generation report and view the profiling report,

1. Open the **Configuration Parameters** dialog and select **Real-Time Workshop**.

2 Under **Documentation**, select **Generate HTML report**. By default, **Include hyperlinks to model** and **Launch report after code generation completes** are also selected, as shown in the display below.

You can deselect either or both these options if desired.

| | |
|---|---|
| **Launch report after code generation completes** | Automatically opens a MATLAB Web browser window and displays the code generation report. If you do not select the option, you can open the code generation report (*model*_codegen_rpt.html or *subsystem*_codegen_rpt.html) manually in a MATLAB Web browser window, or in another Web browser manually. |
| **Include hyperlinks to model** | Includes hyperlinks to blocks in the generating model in the report files. When you view the report files in MATLAB, clicking on these hyperlinks display and highlight the referenced blocks in the model. |




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

Real-Time Workshop writes the code generation report files in the `html` subdirectory of the build directory. The top-level HTML report file is named *`model`*`_codegen_rpt.html` or *`subsystem`*`_codegen_rpt.html`.

---

**Note** You can also view the HTML report files, as well as the generated code files, in the Simulink Model Explorer. See the Real-Time Workshop documentation for details.

---

5

# Model Execution

# Model Execution in the OSEK/VDX Operating Environment

This discussion assumes that you are familiar with the Real-Time Workshop Embedded Coder task management scheme, as described in the "Data Structures and Program Execution" section of the Real-Time Workshop Embedded Coder documentation.

Programs generated by the Embedded Target for OSEK/VDX follow conventions similar to programs generated by Real-Time Workshop Embedded Coder. All the same tasking cases (single-rate/singletasking, multirate/singletasking, and multirate/multitasking) are supported.

However, the Embedded Target for OSEK/VDX maps the generating model's sample rates into corresponding prioritized tasks, executing under the control of OSEK/VDX task management mechanisms. All lower-level timing and task scheduling functions are handled by OSEK/VDX. As described in the following sections, this approach results in some efficiencies that simplify and streamline the generated main program.

# Rate Scheduler Functions

In multirate models, scheduling counters are maintained by a generated rate scheduler function. The rate scheduler function is called by `model_step`, on each base rate time step of the model.

In a multirate model that executes in `SingleTasking` mode, the rate scheduler function is named `rate_scheduler`. In a multirate model that executes in `MultiTasking` mode, the rate scheduler function is named `rate_monotonic_scheduler`. To handle the multitasking case, `rate_monotonic_scheduler` maintains flags that indicate when the Rate Transition blocks in a model need to execute. In models that require an absolute time reference, these flags are also used to update absolute time appropriately.

# Model Rates and OSEK/VDX Tasks

To map multitasking execution of a multirate model to OSEK/VDX, the OSEKWorks target automatically defines, for each rate in the model, an OSEK/VDX Task that runs at the corresponding rate and priority. Each OSEK/VDX Task calls the `model_step` function, passing in an appropriate `tid` argument in accordance with Real-Time Workshop conventions (i.e., the base rate task gets `tid 0`, and subrate tasks get `tids 1..numTasks-1`).

Each OSEK/VDX Task is activated by a corresponding cyclic OSEK/VDX Alarm. OSEK/VDX is responsible for servicing hardware timer interrupts and triggering Alarms at the appropriate rates.

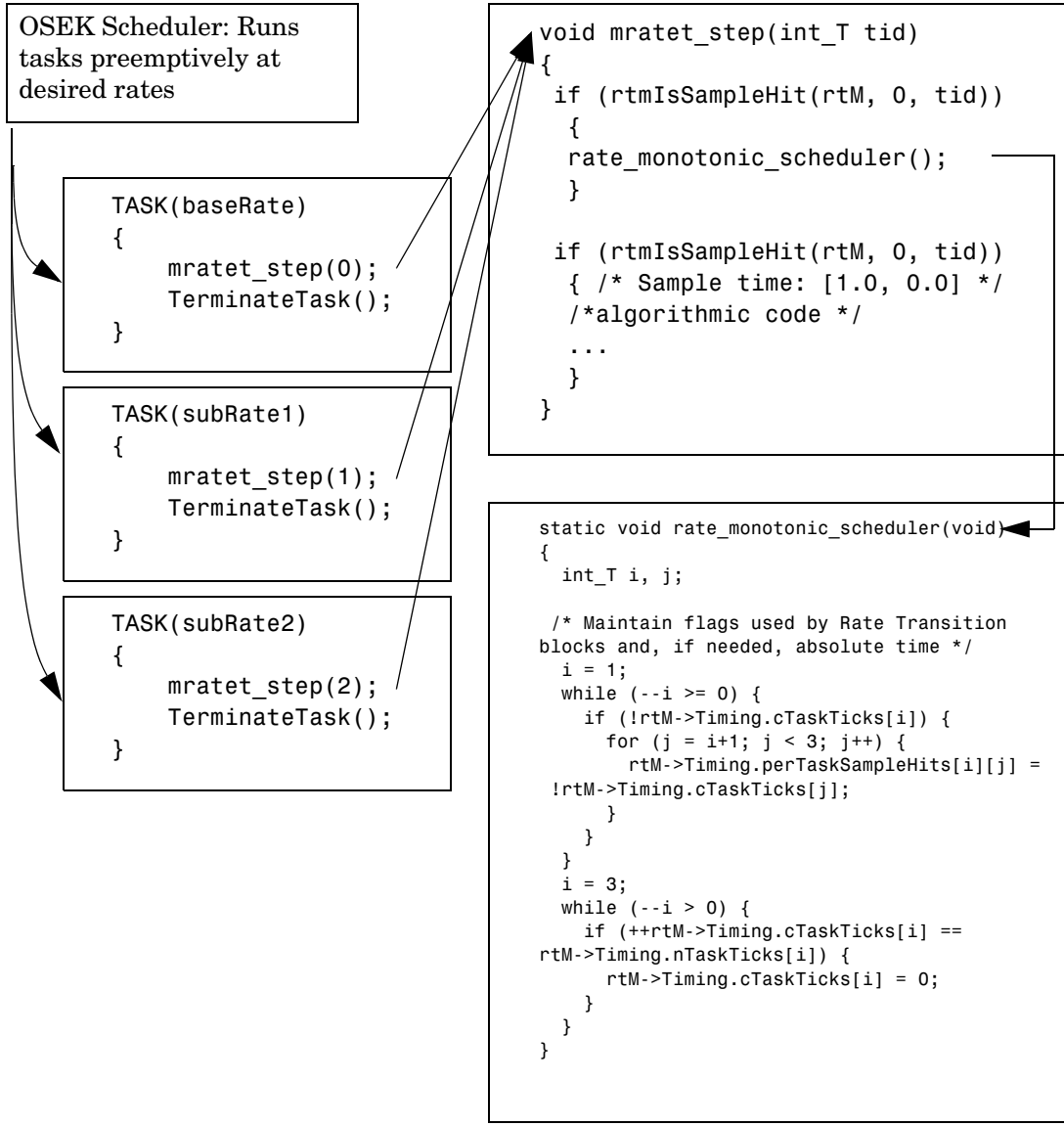
The priority of the base rate (fastest) task is assigned by the user, via the **Base task priority** parameter (see "Code Generation Options" on page 4-5). OSEK/VDX Task priorities decrease from the base rate task to each of the subrate tasks, such that the slowest rate has the lowest priority. Tasks are scheduled preemptively.

This approach allows for several simplifications to be made to the generated main program (`osek_main.c`):

- Each OSEK/VDX Task is specified to have only single activation. Therefore, OSEK/VDX error handling mechanisms can detect timer overrun conditions, so it is not necessary to maintain overrun flags separately.
- The base rate task does not schedule subrate tasks to run; instead OSEK/VDX does the scheduling for the model rates. Therefore, the `rt_OneStep` function is no longer required. Likewise, the event flags array used by `rt_OneStep`, and the `SetEventsForThisBaseStep` function are not needed.

The `rate_monotonic_scheduler` function is still needed to maintain flags for the proper execution of Rate Transition blocks in the model. When the base rate task calls `model_step(0)`, the `rate_monotonic_scheduler` is called to maintain those flags.

The following figure illustrates this model execution approach for the OSEK/VDX target environment.


OSEK Scheduler: Runs tasks preemptively at desired rates
TASK(baseRate)
{
mratet_step(0);
TerminateTask();
}
TASK(subRate1)
{
mratet_step(1);
TerminateTask();
}
TASK(subRate2)
{
mratet_step(2);
TerminateTask();
}
void mratet_step(int_T tid)
{
if (rtmIsSampleHit(rtM, 0, tid))
{
rate_monotonic_scheduler();
}
if (rtmIsSampleHit(rtM, 0, tid))
{ /* Sample time: [1.0, 0.0] */
/*algorithmic code */
...
}
}
static void rate_monotonic_scheduler(void)
{
int_T i, j;
/* Maintain flags used by Rate Transition blocks and, if needed, absolute time */
i = 1;
while (--i >= 0) {
if (!rtM->Timing.cTaskTicks[i]) {
for (j = i+1; j < 3; j++) {
rtM->Timing.perTaskSampleHits[i][j] = !rtM->Timing.cTaskTicks[j];
}
}
}
i = 3;
while (--i > 0) {
if (++rtM->Timing.cTaskTicks[i] == rtM->Timing.nTaskTicks[i]) {
rtM->Timing.cTaskTicks[i] = 0;
}
}
}

## Startup Task for OSEKWorks

Under OSEKWorks conventions, Board Support Packages are configured to start the OSEK kernel directly via the `StartOS()` API, instead of invoking `StartOS()` from the `main()` function of the application code.

Real-Time Workshop expects the application to have at least one `AUTOSTART` task. In the case of generated model code, this is the `init` task. Therefore, the normal entry point (the `main()` function) is never used. The `main()` function is generated within `osek_main.c`, but only to define the symbol `main` and for consistency with other C programs.

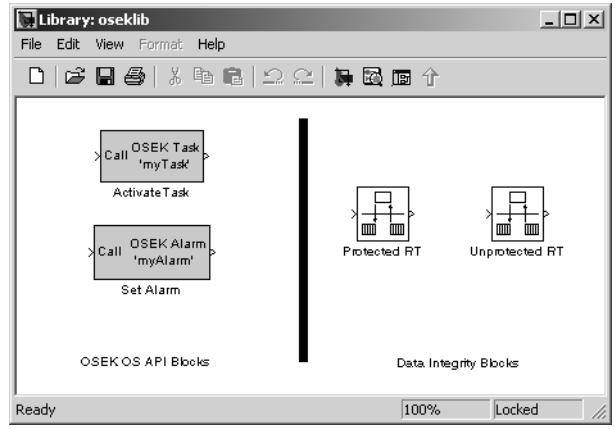
The unused `main()` function for OSEKWorks consumes 36 bytes in the `.text` section of the code.

6

# Block Reference

# Block Library for Embedded Target for OSEK/VDX

The Embedded Target for OSEK/VDX provides a library of blocks (`oseklib.mdl`) that support a subset of the OSEK/VDX API. The following figure shows the Embedded Target for OSEK/VDX block library.




A demonstration model, `osek_apis`, provides a working example of the blocks in the Embedded Target for OSEK/VDX block library.

In addition to the above blocks, an example device driver block for the Phytec PhyCORE-MPC555 board is provided (see “PhyCORE-MPC555 LED” on page 6-11). This block is convenient for providing visual feedback from a generated program. It does not provide OSEK-specific functionality.

The Embedded Target for OSEK/VDX automatically maps model code to OSEK/VDX scheduling mechanisms (see “Model Execution in the OSEK/VDX Operating Environment” on page 5-2). The Embedded Target for OSEK/VDX block library gives you more explicit control over the mapping of the generated model code to OSEK/VDX task management functions. This level of control is useful (for example) when you want to use model-based design and code generation to fit generated code into a larger application on a task-by-task basis.

The blocks in the Embedded Target for OSEK/VDX block library let you specify mappings to OSEK/VDX at a Function-Call Subsystem level. When you use these blocks for code generation, you introduce changes in code execution

behavior. It is important to recognize and understand such effects when using these blocks.

One effect of using the blocks in this library is that the execution of the model code on the target may produce results that do not match simulation behavior. This is because OSEK API blocks (e.g., the Activate Task block) deviate from the rate monotonic scheduling that the automatic mapping performs.

A second effect is related to data integrity. A Simulink signal is said to possess data integrity when the values on the signal are consistent such that all bytes (both within a scalar and across the scalars that form a signal vector), are computed during the same time sample and thus form a meaningful value. Normally, when data is passed between blocks of differing rates in the model, Simulink enforces the use of Rate Transition blocks. This requirement, in conjunction with rate monotonic scheduling, ensures data integrity and also deterministic, repeatable results.

When Embedded Target for OSEK/VDX library blocks are used in a model, however, Simulink does not fully control the relative priorities at which model tasks run. Function-Call Subsystems (generated as OSEK Tasks) can be preempted while they are in the process of reading or writing input or output data. This can cause inconsistencies in the data that is read or written. To avoid this issue and enforce data integrity, you can use the Protected RT and Unprotected RT blocks in the Embedded Target for OSEK/VDX library. Simulink enforces the use of these blocks to ensure data integrity. However, use of these blocks does not guarantee generated code will produce results that exactly match simulation results.

In some applications, data integrity protection is not required, and greater efficiency can be achieved by omitting the data integrity protection code. In such cases, you can use the Unprotected RT block.

For details about a specific block, click **Help** on the **Block Parameters** dialog for the block, or access the block reference page through Help.

# Blocks — Categorical List

The blocks in the Embedded Target for OSEK/VDX library are organized into categories that support different functions. The tables below reflect that organization.

## OSEK Operating System API Blocks

| Block Name | Purpose |
|---|---|
| Activate Task | Generate an OSEK/VDX API `ActivateTask` call |
| Set Alarm | Generate an OSEK/VDX API `SetAbsAlarm` or `SetRelAlarm` call |

## Data Integrity Blocks

**Note** The OSEK Async Rate Transition and Unprotected OSEK Async Rate Transition blocks are now obsolete. They are included only for compatibility with older models. Use the Protected RT and Unprotected RT blocks as a replacement. For details, see Protected RT and Unprotected RT. Also, see rate transition block information in the Real-Time Workshop documentation.

| Block Name | Purpose |
|---|---|
| OSEK Async Rate Transition (Reader) | Ensure data integrity for data signals before crossing an async rate boundary |
| OSEK Async Rate Transition (Writer) | Ensure data integrity for data signals after crossing an async rate boundary |

| Block Name | Purpose |
|---|---|
| Protected RT | Ensures data integrity for data transfers in multirate systems |
| Unprotected RT | Ensures that data transfers within a multirate system are deterministic |

## Example Driver Block

| Block Name | Purpose |
|---|---|
| PhyCORE-MPC555 LED | Demo driver for LEDs D4 and D5 on Phytec PhyCORE-MPC555 board |

# Blocks — Alphabetical List

This section provides detailed block descriptions organized alphabetically by block name.

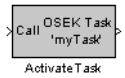
## Purpose

Generate an OSEK/VDX API `ActivateTask` call

## Library

Embedded Target for OSEK/VDX

## Description




The Activate Task block is designed to be connected to the trigger input of a downstream Function-Call Subsystem. It explicitly defines the function-call subsystem as an OSEK/VDX task and activates the Task via a call to the OSEK/VDX API function `ActivateTask`. The `CALL` input to the Activate Task block can be driven by any function-call output, such as the output of a Function-Call Generator block or a Stateflow function-call event.

The generated OSEK task is activated from the model context in which the Activate Task block is placed. The priority of the generated task, relative to the calling Activate Task block's task priority, is important to consider. The caller's priority is typically assigned based on

- The caller's sample rate relative to the model's base sample rate
- The **Configuration Parameters** setting for **Base task priority**. This option is on the **OSEKWorks code generation options** or **ProOSEK code generation options** pane under **Real-Time Workshop**.

If the downstream task priority is set to be lower than the caller's, the `ActivateTask` call puts the task in the ready queue of the OSEK scheduler on the target environment. This differs from simulation, where the function-call subsystem executes immediately upon being triggered and runs to completion before the calling block returns.

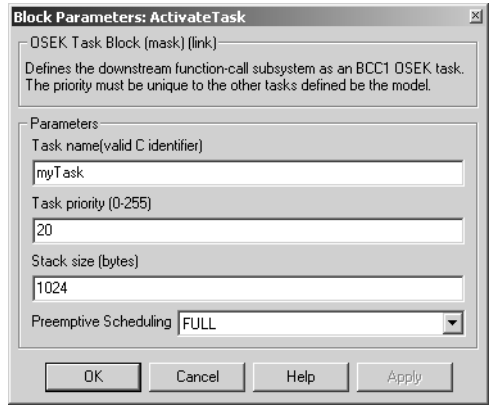
If, however, the priority specified for the downstream function-call subsystem is higher than the task in which the caller resides, the target behavior will match the simulation behavior.

The **Preemptive scheduling** parameter of the Activate Task block lets you specify the scheduling behavior of the task as either full preemptive or nonpreemptive, as allowed by OSEK/VDX. You should also consider this parameter when comparing execution on the target to simulation behavior. The Activate Task Block has the effect of detaching the execution of the Function-Call Subsystem from the caller. As a consequence, data read into and written from the function-call subsystem can be in an inconsistent state. To ensure data integrity, Simulink requires that OSEK Async Rate Transition

(Reader or Writer) blocks are used with all inputs and outputs of for Function-Call Subsystems that are triggered by Activate Task blocks.

When the Activate Task block is driven by a block, its behavior is modified. The Activate Task block does not generate its own call to `ActivateTask`; instead, the alarm generated by the Set Alarm block activates the task. See “Set Alarm” on page 6-15 for additional details.

## Dialog Box




**Task name**

Name of OSEK task in the generated code.

**Task priority**

Priority of the generated OSEK task. The priority must be unique within the model.

**Stack size (bytes)**

Number of bytes allocated for this task’s stack.

**Preemptive scheduling**

- `FULL`: OSEK will allow preemption of the task by the scheduler.
- `NONE`: The Task cannot be preempted by any other task.

## Purpose

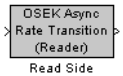
Ensure data integrity for data signals before crossing an async rate boundary

## Library

Embedded Target for OSEK/VDX

## Description

The OSEK Async Rate Transition (Reader) block ensures data integrity for data signals that are entering an asynchronous rate transition boundary.



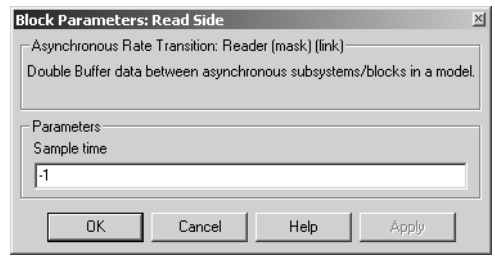

---

**Note** This block is now obsolete. It is included only for compatibility with older models. Use the Rate Transition block in the Simulink Signal Attributes library as a replacement. For details, see "Rate Transition Block Options" in the Real-Time Workshop documentation.

---

The OSEK Async Rate Transition Block (Reader) block is normally paired with an OSEK Async Rate Transition Block (Writer) block. This reader/writer pair uses two buffers and the OSEK/VDX `EnableAllInterrupts` and `DisableAllInterrupts` calls to provide a mechanism that ensures proper, exclusive access to control variables for the double buffers.

## Dialog Box




**Sample time**

Sample time of the block.

# OSEK Async Rate Transition (Writer)

## Purpose

Ensure data integrity for data signals after crossing an async rate boundary

## Library

Embedded Target for OSEK/VDX

## Description

The OSEK Async Rate Transition (Writer) block ensures data integrity for data signals that are exiting an asynchronous rate transition boundary.

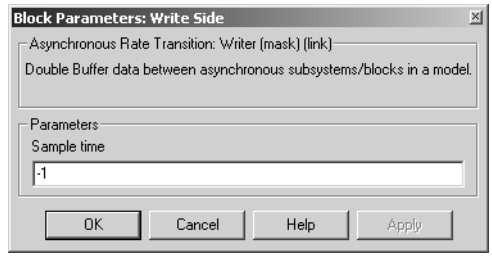

---

**Note** This block is now obsolete. It is included only for compatibility with older models. Use the Rate Transition block in the Simulink Signal Attributes library as a replacement. For details, see "Rate Transition Block Options" in the Real-Time Workshop documentation.

---

The OSEK Async Rate Transition Block (Writer) block is normally paired with an OSEK Async Rate Transition Block (Reader) block. This reader/writer pair uses two buffers and the OSEK/VDX `EnableAllInterrupts` and `DisableAllInterrupts` calls to provide a mechanism that ensures proper, exclusive access to control variables for the double buffers.

## Dialog Box




**Sample time**

Sample time of the block.

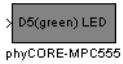
## Purpose

Demo driver for LEDs D4 and D5 on Phytec PhyCORE-MPC555 board

## Library

Embedded Target for OSEK/VDX

## Description




The PhyCORE-MPC555 LED driver block provides a simple means for visual feedback, via an LED, for programs running on the PHYTEC phyCORE-MPC555 board.

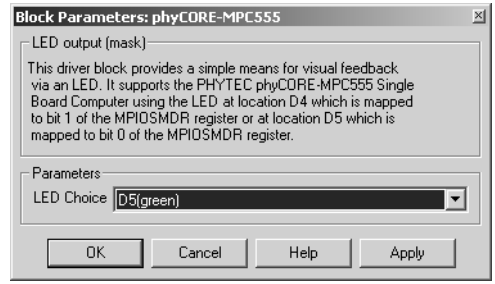
The PhyCORE-MPC555 LED driver block uses the following bits of the MPIOSMDR register:

- Bit 0: mapped to green LED at location D5
- Bit 1: mapped to red LED at location D4

The selected bit is toggled on or off by the input signal.

For an example of the use of this block, see the `osek_led` demo model. This model is described in “The osek_led Demo Model” on page 3-31.

## Dialog Box




### LED Choice

This option lets you select either the green (D5) or red (D4) LED to be driven by the input signal. The color of the block icon changes from green to red, depending upon your selection.

# Protected RT

## Purpose

Ensure data integrity for data transfers in multirate systems

## Library

Embedded Target for OSEK/VDX

## Description


The Protected RT block transfers data from the output of a block operating at one rate to the input of another block operating at a different rate. Block parameters allow you to trade data integrity and deterministic transfers for faster response and lower memory requirements. By default, the Protected RT block ensures data integrity.

---

**Note** See "Data Transfer Problems" in the online Real-Time Workshop documentation for a discussion of data integrity and deterministic data transfer.

---

The block supports the following options:

- Nondeterministic data transfer with minimum latency and assured data integrity but increased memory requirements. This is the default.
- Deterministic transfer of data with data integrity between blocks operating at different speeds at the cost of maximum latency of data transfer.

  To specify this option, select **Ensure data integrity during data transfer** and **Ensure deterministic data transfer**.
- Minimum latency and target size at the cost of nondeterministic data transfer and possible loss of data integrity.

  To specify this option, clear **Ensure data integrity during data transfer** and **Ensure deterministic data transfer**.

See "Sample Rate Transitions" in the online Real-Time Workshop documentation for more information.

---

**Note** The Rate Transition, Zero-Order Hold, and Unit Delay blocks also enable transfer of data between blocks operating at different rates. However, you should use the Protected RT block for this purpose because it is customized for Embedded Target for OSEK/VDX use.

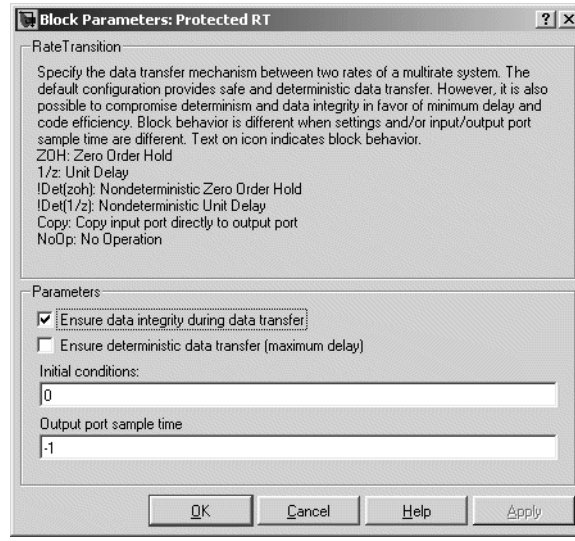
---

## Data Type Support

The Protected RT block accepts signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, refer to "Data Types Supported by Simulink" in the online Simulink documentation.

## Dialog Box




**Ensure data integrity during data transfer**

Specifies that generated code ensure the integrity of data transferred by this block. If you select this option and the transfer is nondeterministic (see **Ensure deterministic data transfer** option below), the generated code uses double-buffering to prevent the fast block from interrupting the data transfer. Otherwise the generated code uses a copy operation to effect the data transfer. The copy operation consumes less memory than double-buffering, but is also interruptible and hence can lead to loss of data during nondeterministic data transfers.

Select this option if you want the generated code to operate with maximum responsiveness (that is, nondeterministically) and assured data integrity. See "Rate Transition Block Options" in the online Real-Time Workshop documentation for more information.

**Ensure deterministic data transfer (maximum delay)**

Specifies that generated code transfer data at the sample rate of the slower block (that is, deterministically). If you do not select this option, which is the default, data transfers occur as soon as new data is available from the source block and the receiving block is ready to receive the data. This avoids the need to delay transfers, thus ensuring that the system operates with maximum responsiveness. However, it also means that transfers can occur unpredictably, which is undesirable in some applications. See "Rate Transition Block Options" in the online Real-Time Workshop documentation for more information.

**Initial conditions**

Applies only to `Slow to fast` transitions. It specifies the initial output at the beginning of a transition when there is not yet any output from the slow block connected to the Protected RT block's input. The default is 0.

**Output port sample time**

Specifies the output rate to which the input rate is converted. The default value (`-1`) specifies that the output rate is inherited from the block to which the Protected RT block's output port is connected. See "Specifying Sample Time" in the online documentation for information on how to specify the output rate.

## Characteristics

| | |
|---|---|
| Direct Feedthrough | No for slow-to-fast transitions that are protected (**Ensure data integrity during data transfer** selected); otherwise, yes. |
| Sample Time | Supports discrete-to-discrete and discrete-to-continuous transitions. |
| Scalar Expansion | Yes, of input. |
| Dimensionalized | Yes |
| Zero Crossing | No |

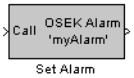
## Purpose

Generate an OSEK/VDX API `SetAbsAlarm` or `SetRelAlarm` call

## Library

Embedded Target for OSEK/VDX

## Description




The Set Alarm block generates either an absolute or relative OSEK alarm via OSEK/VDX API calls, `SetAbsAlarm` or `SetRelAlarm` call.

The Set Alarm block has two modes of operation, controlled by the **Call only at startup** option. When you select this option, the `SetAbsAlarm` or `SetRelAlarm` call is only generated once, in the model initialization function. In this mode:

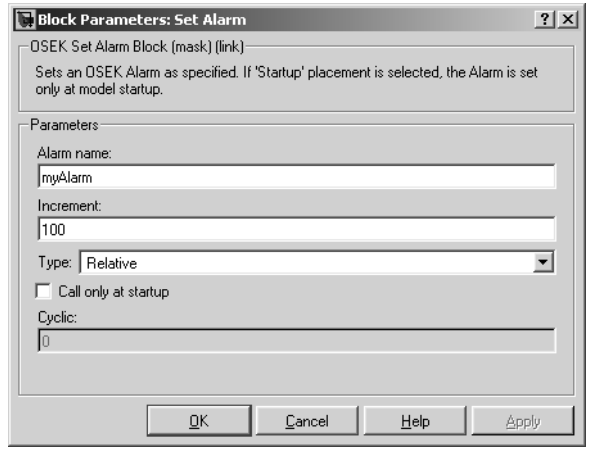
- No signal should be connected to the `CALL` input of the Set Alarm block.
- The **Cyclic** parameter can be set to nonzero values.

When you clear the **Call only at startup** option, the Set Alarm block generates a `SetAbsAlarm` or `SetRelAlarm` call in the model context in which the Set Alarm block is placed. In this mode:

- The `CALL` input to the alarm is valid and can be driven by any function-call source.
- You cannot set the **Cyclic** parameter to nonzero values. This disallows multiple activations of the downstream task.

Note that the Set Alarm block immediately activates the downstream task during simulation.

## Dialog Box




**Alarm name**

The name of the OSEK alarm in the generated code.

**Increment**

A numeric value `n` used as an argument in the generated call to `SetAbsAlarm` or `SetRelAlarm` to specify one of the following:

- For a relative alarm, the number of ticks of the OSEK system counter that must elapse before the task assigned to the alarm is activated
- For an absolute alarm, the value that the OSEK system counter must equal for the task assigned to the alarm to be activated

See the OSEK/VDX documentation for more information.

**Type**

Specifies whether the alarm is relative or absolute.

**Call only at startup**

Specifies that the alarm be activated only during OSEK startup.

**Cyclic**

If **Call only at startup** is enabled, a nonzero value `c` used as an argument in the generated `SetAbsAlarm` or `SetRelAlarm` call to specify cyclic task activation. The alarm activates the task repeatedly every `c` ticks.

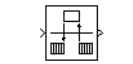
## Purpose

Ensure that data transfers within a multirate system occur with minimum latency and target size

## Library

Embedded Target for OSEK/VDX

## Description




The Unprotected RT block transfers data from the output of a block operating at one rate to the input of another block operating at a different rate while ensuring minimum latency and target size. If necessary, you can select the option **Ensure deterministic data transfer** to ensure deterministic transfers at the cost of increased latency.

---

**Note** See "DataTransfer Problems" and "Sample Rate Transitions" in the online Real-Time Workshop documentation for more information.

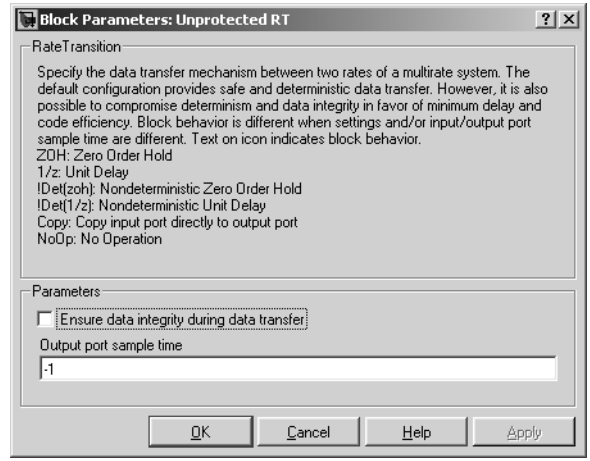
---

## Data Type Support

The Unrotected RT block accepts signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, refer to "Data Types Supported by Simulink" in the online Simulink documentation.

## Dialog Box



**Ensure deterministic data transfer (maximum delay)**

Specifies that generated code transfer data at the sample rate of the slower block (that is, deterministically). If you do not select this option, which is the default, data transfers occur as soon as new data is available from the source block and the receiving block is ready to receive the data. This avoids the need to delay transfers, thus ensuring that the system operates with maximum responsiveness. However, it also means that transfers can occur unpredictably, which is undesirable in some applications. See "Rate Transition Block Options" in the online Real-Time Workshop documentation for more information.

**Output port sample time**

Specifies the output rate to which the input rate is converted. The default value (`-1`) specifies that the output rate is inherited from the block to which the Unprotected RT block's output port is connected. See "Specifying Sample Time" in the online documentation for information on how to specify the output rate.

## Characteristics

| | |
|---|---|
| Direct Feedthrough | Yes |
| Sample Time | Supports discrete-to-discrete and discrete-to-continuous transitions. |
| Scalar Expansion | Yes, of input. |
| Dimensionalized | Yes |
| Zero Crossing | No |

# Index

## D



## E



## F



## G



## H

## P



## R



## S



## T

## U