Embedded Target for OSEK/VDX®

For Use with Real-Time Workshop®

- Modeling
- Simulation
- Implementation

User's Guide

Version 1

The MathWorks

Embedded Target for OSEK/VDX User's Guide

Printing History: February 2003 Online only Version 1.0 (Release 13+)

Contents

Preface

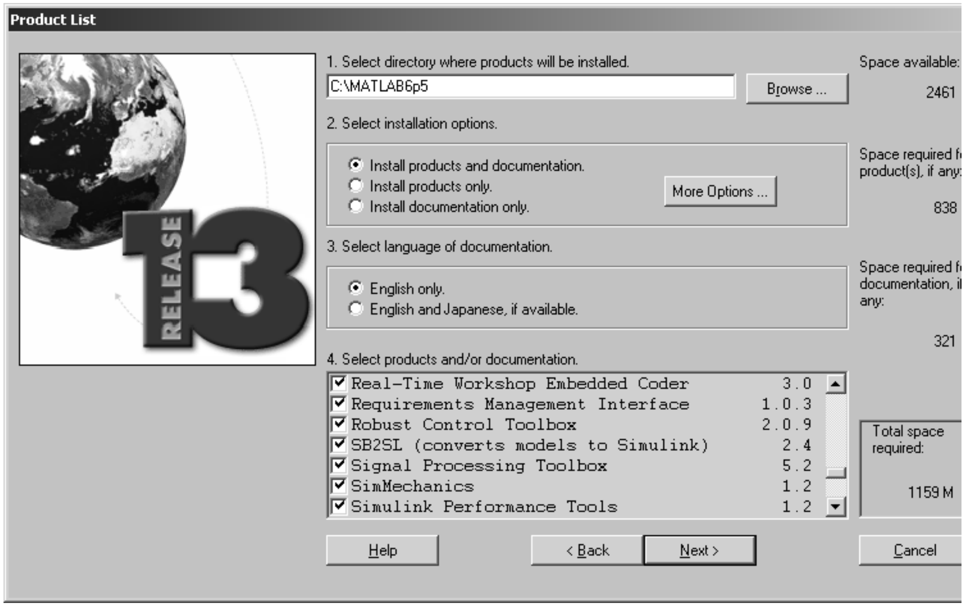
This section includes the following topics:

Installing the Embedded Target for OSEK/VDX

Your platform-specific MATLAB Installation guide provides all of the information you need to install the Embedded Target for OSEK/VDX.

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

As the installation process proceeds, it displays a dialog similar to the one below, letting you indicate which products to install.



1. Select directory where products will be installed.
C:\MATLAB6p5

2. Select installation options.
- [✓ Install products and documentation.]
- Install products only.
- Install documentation only.

3. Select language of documentation.
- [✓ English only.]
- English and Japanese, if available.

4. Select products and/or documentation.

```
✓ Real-Time Workshop Embedded Coder          3.0
✓ Requirements Management Interface        1.0.3
✓ Robust Control Toolbox                   2.0.9
✓ SB2SL (converts models to Simulink)        2.4
✓ Signal Processing Toolbox                  5.2
✓ SimMechanics                               1.2
✓ Simulink Performance Tools                 1.2
```

Space available: 2461

Space required f product(s), if any: 838

Space required f documentation, i any: 321

Total space required: 1159 M

After the installation process completes, proceed to “Product Overview” to learn about other software required for use with the Embedded Target for OSEK/VDX. Then read “Configuring the Embedded Target for OSEK/VDX” and follow the configuration process described.

Version 1.0 Release Notes

Known Software and Documentation Problems

This section describes problems and limitations that have been identified in the current release.

Compiler Optimizations

In some very rare instances, due to compiler defects, compiler optimizations applied to Embedded Target for OSEK/VDX generated code may cause the executable program to produce incorrect results, even though the code itself is correct.

To work around such problems, first refer to your compiler's documentation for information on how to lower the optimization level of the compiler or turn off optimizations. Then, having found the optimization switches required, you can edit the options directly into the template makefile for your OSEK implementation (`osekworks.tmf` or `proosek.tmf`).

Make Error When Compiling Large Numbers of Files

A spurious make utility error occasionally occurs when executing a large number of target rules, such as building object libraries that contain over a hundred object files. You can work around this problem as follows:

1. Deselect the **Force Rebuild** option in the OSEKWorks or ProOSEK code generation options category of the Real-Time Workshop pane.
2. Execute the make process several times, noting that the same error does not recur in subsequent builds. You can do this either by clicking the **Build** button in the Real-Time Workshop pane, or by executing the *`model`*`.bat` file manually.

Set Alarm Block Limitation

In the current release, the output of the Set Alarm block can be connected only to an Activate Task block. The Set Alarm block activates the Task associated with the Activate Task block.

Using This Guide

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

- Read “What You Need to Know to Use This Product” on page 1-2 to understand the prerequisite knowledge required to use the Embedded Target for OSEK/VDX, and to learn about related documentation you may need to read.
- Read “Introduction to the Embedded Target for OSEK/VDX” on page 1-4 to learn about the general features of the product.
- 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).
- Read Chapter 3, “Generating Real-Time OSEK/VDX Applications” to get started with generating deploying OSEK/VDX applications on target hardware. Work through the tutorial that is applicable to your chosen OSEK/VDX implementation.
- 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.
- Then, 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.
- See also “Embedded Target for OSEK/VDX Demos” below.

Embedded Target for OSEK/VDX Demos

We have provided a number of demos to help you become familiar with features of the Embedded Target for OSEK/VDX.

If you are reading this document online in the MATLAB Help browser, you can run the demos by clicking on the links in the **Command** column of the following table.

Alternatively, you can access the demo suite from the Launch Pad, or by typing commands from the MATLAB command prompt, as in this example:

```
osek_mrate
```

Embedded Target for OSEK/VDX Demos

Command	Demo Topic
`osek_mrate`	Illustrates how multirate/multitasking models execute under OSEK/VDX.
`osek_led`	Illustrates 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 the Phytec phyCORE-MPC555 board.
`osek_apis`	Illustrates 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 the Phytec phyCORE-MPC555 board.
`osek_asap2`	Illustrates the generation of an ASAP2 file used for calibration.

Related Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the Embedded Target for OSEK/VDX. They are listed in the table below.

The Embedded Target for OSEK/VDX *requires* these products:

- MATLAB® 6.5.1
- Simulink® 5.1
- Real-Time Workshop® 5.1
- Real-Time Workshop Embedded Coder 3.1

For more information about any of these products, see either

- The online documentation for that product, if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at `http://www.mathworks.com`; see the "products" section

Note The toolboxes listed below all include functions that extend the capabilities of MATLAB. The blocksets all include blocks that extend the capabilities of Simulink.

Product	**Description**
Fixed-Point Blockset	Design and simulate fixed-point systems
MATLAB	The Language of Technical Computing
Real-Time Workshop	Generate C code from Simulink models
Real-Time Workshop Embedded Coder	Generate production code for embedded systems
Simulink	Design and simulate continuous- and discrete-time systems

Product	Description
Stateflow®	Design and simulate event-driven systems
Stateflow Coder	Generate C code from Stateflow charts

Typographical Conventions

This manual uses some or all of these conventions.

Item	Convention	Example
Example code	`Monospace` font	To assign the value 5 to `A`, enter `A = 5`
Function names, syntax, filenames, directory/folder names, and user input	`Monospace` font	The `cos` function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Buttons and keys	**Boldface** with book title caps	Press the **Enter** key.
Literal strings (in syntax descriptions in reference chapters)	**`Monospace bold`** for literals	`f = freqspace(n,'`**`whole`**`')`
Mathematical expressions	*Italics* for variables Standard text font for functions, operators, and constants	This vector represents the polynomial $p = x^2 + 2x + 3$.

MATLAB output	`Monospace font`	MATLAB responds with
		`A =`
		`    5`

Menu and dialog box titles	**Boldface** with book title caps	Choose the **File Options** menu.
New terms and for emphasis	*Italics*	An *array* is an ordered collection of information.
Omitted input arguments	(...) ellipsis denotes all of the input/output arguments from preceding syntaxes.	`[c,ia,ib] = union(...)`
String variables (from a finite list)	*`Monospace italics`*	`sysc = d2c(sysd,'`*`method`*`')`

Product Overview

This section contains the following topics:

What You Need to Know to Use This Product

This document assumes you are experienced with MATLAB, Simulink, Real-Time Workshop, and the 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.” This section introduces general concepts and terminology related to Real Time Workshop.
- “Quick Start Tutorials.” This section provides several 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 the following sections:

- “Data Structures and Code Modules”
- “Code Generation Options and Optimizations”

Familiarity with OSEK/VDX is also 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*

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

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

Familiarity with your chosen target board and processor are also 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. In this document, we assume that 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:

- The *Motorola MPC555 Users Guide* is available at the following URL: `http://e-www.motorola.com/webapp/sps/library/prod_lib.jsp`.
- Information on the Phytec PhyCORE-MPC555 board is available at the Phytec Web site: `http://www.phytec.com/pc_devbd_555.html`.

Introduction to 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 a unified set of tools for developing real-time applications for OSEK/VDX.

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

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

Feature Summary

- Supports two major OSEK/VDX implementations and associated development tools. The Embedded Target for OSEK/VDX provides two target configurations:
  - **OSEK Target for WRS OSEKWorks Implementation**: Supports Tornado for OSEKWorks for PowerPC 3.0. In this document, we refer to this target as the *OSEKWorks target*.
  - **OSEK Target for 3Soft ProOSEK Implementation**: Supports ProOSEK 3.0r3. In this document, we refer to this target as the *ProOSEK target*.
- 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.
- Supports generation of a single- or multirate model or subsystem as an OSEK/VDX executable. Executables can be downloaded and run in either RAM or FLASH memory.
- Supports Real-Time Workshop task management mechanisms within the OSEK/VDX environment by mapping sample rates in the model to OSEK/VDX tasks.

- Both target configurations support automatic downloading and debugging of code via the SingleStep^{TM} debugger.
- OSEK/VDX block library supports basic OSEK APIs
- Supports ASAP2 file generation
- Single-precision math library (`mathf.h`) support (OSEKWorks only)
- Extensible, open implementation; hook file mechanisms ease customization of code generation options and makefile variables and rules.

Configuring the Embedded Target for OSEK/VDX

This section contains the following topics:

Hardware and Software Requirements

Host Platform

The Embedded Target for OSEK/VDX supports only the PC as host platform, running Windows NT, 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 the 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 these settings and procedures.

Software Requirements

See “Related Products” in the Preface for information on MathWorks products required to use Embedded Target for OSEK/VDX. In addition to these products, a supported OSEK/VDX implementation and development environment are necessary. This section gives the requirements for the currently supported OSEK/VDX environments and related tools.

Software Requirements for the OSEKWorks Target

The OSEKWorks target requires Tornado for OSEKWorks for PowerPC 3.0, from Wind River Systems, Inc. OSEKWorks includes the Diab cross-compiler and the SingleStep with vision debugger (Version 7.7.1, for use with VisionPROBE). However, note that testing at The MathWorks indicates that SingleStep with vision Version 7.7.3 is required for programming FLASH memory.

The SingleStep debugger is used for automatic downloading, running, and debugging of generated executables. SingleStep is not required for the generation of code and executables, however. 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). This version is currently shipped with OSEKWorks.)
- SingleStep On-Chip version 7.6.2, for use with the Phytec PhyCORE-MPC555 board with on-board Background Debug Mode (BDM) connector, or with an external BDM device such as the Macraigor Wiggler (WBDM8xx) BDM.

Software Requirements for the ProOSEK Target

The ProOSEK target requires ProOSEK 3.0r3 from 3SOFT, GmbH. The target support options are for the MPC555 processor, with GNU tools.

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 used for automatic downloading, running, and debugging of generated executables. SingleStep is not included with ProOSEK, so you must obtain it separately. SingleStep is not required for the generation of code and executables, however. 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). However, note that 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 on-board BDM, or with an external BDM such as the Macraigor Wiggler (WBDM8xx) BDM.

Setting Up and Verifying Your Installation

The next sections describe how to configure your development environment for use with the Embedded Target for OSEK/VDX and verify correct operation. The initial configuration steps are described in the following sections:

- "Setting Up Your Target Hardware" on page 2–5
- "Setting Target Preferences" on page 2–12. The 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 your 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 2–16.
- If you are using ProOSEK, see "Setting Up Your Installation for the ProOSEK Target" on page 2–18.

Setting Up Your Target Hardware

In this document, we assume that you are working with the Phytec phyCORE-MPC555 development board. This section gives information on the required connections and jumper settings for the 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 2–12.

Physical Connections and Communications Ports

Before you begin working with the Embedded Target for OSEK/VDX, you should set up your phyCORE-MPC555 board and connect it to your host computer. The hardware setup is described in the *phyCORE-MPC555 Quickstart Instructions* manual on your Phytec Spectrum CD. See the "Interfacing the phyCORE-MPC555 to a Host PC" section of the "Getting Started" chapter.

In this document, we assume that you have connected 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. Please 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 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

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 2-16.

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 3-18. In step 1 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 provided for the OSEKWorks and ProOSEK implementations. These command files support generation of read-only (code) sections that 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:

Download and Run in RAM. When the `Download_and_run` option is selected, 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, the code image is loaded into off-chip RAM and the on-chip RAM is used for read-write sections.

You can also manually download and run code in RAM with the `FLEN` bit set to `0`.

See “Tutorial 3: Downloading the Application to RAM via SingleStep” on page 3-18 and “Tutorial 4: Automated Downloading and Debugging” on page 3-25 for details.

Download and Run in FLASH. If the SingleStep debugger is started and manually configured to connect to the target with the `FLEN` bit of `IMMR` set to `1`, internal access cycles from `0x0` to `0x2FBFFF` are mapped to the on-chip FLASH by the MPC555. 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 the board is configured to set the `FLEN` bit to `1`, the processor will execute the code from on-chip FLASH.

For details on running generated code in FLASH, see

- "Tutorial 5: Downloading Generated Code to FLASH" on page 3-28
- Table 2-3, PhyCORE-MPC555 Jumper Settings for Execution from On-Chip FLASH Memory at Power-On or Reset

See also the section "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

There are also other memory mapping examples provided in the linker command files. In the following example below, 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. It also acceptable for larger RAM sizes (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, be sure to 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.

To modify 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, you must

- Edit the linker command file, `bsp.lk`.
- Rebuild the phyCORE-MPC555 BSP.

You can find and edit `bsp.lk` in the following location:

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

After you edit this copy of `bsp.lk`, use the following commands from the PC command prompt window to rebuild the BSP.

```
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 that 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

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

After you make changes to `os.cmd`, you must 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 2-18). 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
`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: `'c:\wind\TornadoOW_ppc_3.0'` (for an OSEKWorks installation) `'c:\ProOSEK'` (for a ProOSEK installation)

`StaticLibraryDirectory`	Directory where static object libraries are to be built and stored (see "Efficient Use of Persistent Object Libraries" on page 4–10)	Default: *`matlabroot`*`\toolbox\rtw\targets\osek\lib`
		Examples:
		`matlabroot``\work`
		`c:\temp`

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

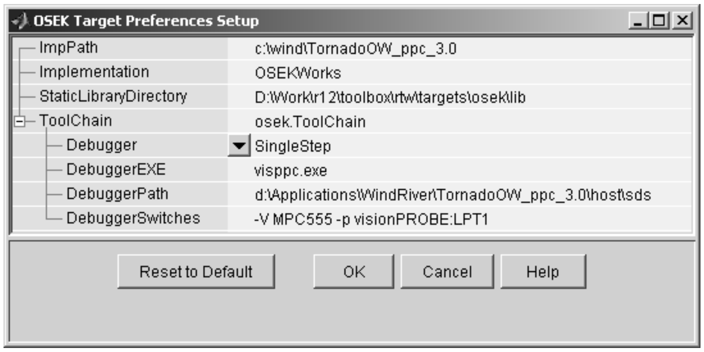
Preference Name	Description	Default or Example Value
`Debugger`	Name of debugger used for automatic downloading	`'SingleStep'` Note: An alternate value is `'Custom'`. If you select `'Custom'`, you must implement custom debugger support. (See "Custom Debugger Support" on page 4-11)
`DebuggerEXE`	Name of debugger executable (*must be localized for your installation*)	Default: `'visppc.exe'` (for SingleStep with vision) 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`

`DebuggerSwitches`	Switches to set debugger options (such as port name and speed) when debugger is invoked for auto-downloading. In this release, 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
		
		`'-V MPC555 -p visionPROBE:LPT1'`
		
		Correct value for SingleStep On-Chip (7.6.2) with BDM port:
		
		`-p LPT1:1`

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



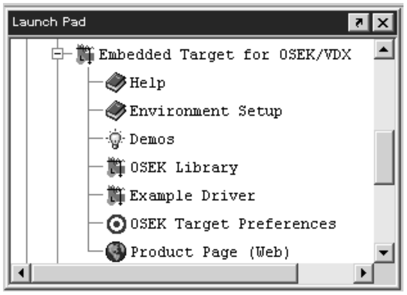
ImpPath	c:\wind\TornadoOW_ppc_3.0
Implementation	OSEKWorks
StaticLibraryDirectory	D:\Work\r12\toolbox\rtw\targets\osek\lib
ToolChain	osek.ToolChain
Debugger	SingleStep
DebuggerEXE	visppc.exe
DebuggerPath	d:\Applications\WindRiver\TornadoOW_ppc_3.0\host\sds
DebuggerSwitches	-V MPC555 -p visionPROBE:LPT1

Target Preferences Setup Window

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

1 Select **Launch Pad** from the **View** menu in the MATLAB desktop. In the **Launch Pad**, click on the plus sign to the left of the **Simulink** icon.

2 In the expanded list under the **Simulink** icon, locate the **Embedded Target for OSEK/VDX** icon**.** Click on the plus sign to the left of the **Embedded Target for OSEK/VDX** icon. The **OSEK Target Preferences** icon is now visible, as shown.



3 Double-click on the **OSEK Target Preferences** icon. The **Target Preferences Setup** window opens.

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

4 Modify the properties you want to change.

5 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:

- Obtain the required version of OSEKWorks (see “Software Requirements” on page 2–2).
- 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.
- Set the target preferences correctly for your installation (see “Setting Target Preferences” on page 2–12). Make sure that the `ImpPath`, `DebuggerPath`, `DebuggerEXE`, and `DebuggerSwitches` preferences are localized correctly for your installation.
- 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 2-16.
- If you want to use the SingleStep debugger for downloading and debugging code, configure SingleStep as described in “Setting Up SingleStep” on page 2–20. 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 2–12). 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 for continuation 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 for continuation 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 following completion message is displayed:

```
Finished setup of phycore555.
```

Setting Up Your Installation for the ProOSEK Target

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

- Obtain the required version of ProOSEK (see “Software Requirements” on page 2–2).
- Install ProOSEK, following the 3SOFT documentation. You can install ProOSEK locally or on your network.
- Set the target preferences correctly for your installation (see “Setting Target Preferences” on page 2–12). Make sure that the `ImpPath`, `DebuggerPath`, `DebuggerEXE`, and `DebuggerSwitches` preferences are localized correctly for your installation.
- 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 2-18.
- If you want to use the SingleStep debugger for downloading and debugging code, configure SingleStep as described in “Setting Up SingleStep” on page 2–20. 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 2–12). 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 for continuation 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 the following completion message is displayed:

```
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 next sections describe how to install and configure SingleStep for this purpose. 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.

SingleStep Installation

If you have not already done so, you should 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

The following sections explain how to configure the SingleStep debugger. (The configuration for SingleStep On-Chip (7.6.2) differs slightly from that for SingleStep with vision.) After configuring SingleStep, you will also be able to use the debugger directly to debug generated programs.

Configuring SingleStep On-Chip (7.6.2)

Apply Patches for Programming FLASH memory. If you want to program your generated applications into the phyCORE-MPC555 FLASH memory using SingleStep On-Chip (7.6.2), you must obtain the following files from Wind River Systems and apply the updates they contain:

- `pcflash11_29_00.zip` and `pcflash11_29_00.txt`

  Apply this update first, after reading the information in the `pcflash11_29_00.txt` file.
- `pcflash3_15_01.zip` and `pcflash3_15_01.txt`

  Apply this update second, after reading the information in the `pcflash3_15_01.txt` file.

Creat Shortcut to SingleStep On-Chip. If you are using SingleStep On-Chip (7.6.2), we recommend that you configure a shortcut to SingleStep as follows in order to start SingleStep with the proper options.

1 If you are using SingleStep On-Chip (7.6.2), the SingleStep installer will have installed a shortcut named `SingleStep On Chip (MPC5xx)` in your system's **Start/Programs/SingleStep 7.6.2** menu. Locate this shortcut file and make a copy of it on your desktop. Rename the copy to `SingleStep On Chip (MPC5xx) for OSEK Target`.

2 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
```

where `ssteproot` is the installed SingleStep directory and `matlabroot` is the MATLAB root directory.

You will use this `SingleStep On Chip (MPC5xx) for OSEK Target` shortcut when manually downloading code to RAM or FLASH via SingleStep.

Configuring SingleStep with vision

OSEKWorks includes SingleStep with vision (version 7.7.1). This version of SingleStep is intended for use with VisionPROBE hardware. This section describes configuration steps you must follow to set up SingleStep with vision and the VisionPROBE for use with the PhyCORE-MPC555 board and with the OSEKWorks target.

Configure VisionPROBE Nonvolatile RAM. The OSEKWorks target provides MATLAB command that creates and runs a SingleStep script that configures the VisionPROBE nonvolatile RAM. The script, `visppcinit.cfg`, is customized to your environment.

Use this command as follows:

1 To execute the command, type the following at the MATLAB prompt:

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

2 A message similar to the following is displayed.

```
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
```

3 SingleStep executes, under control of the `visppcinit.cfg` 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 a further 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 actually programs the VisionPROBE nonvolatile RAM.

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

5 Cycle power and type the `reset` command as instructed. When the `reset` command returns without error, VisionPROBE programming is complete, and the VisionPROBE is communicating with the PhyCORE-MPC555 board correctly.

6 Exit the SingleStep session.

7 You should now proceed to the next section and create a shortcut for SingleStep.

Configuring a Shortcut to SingleStep with vision. The OSEKWorks installer does not install a shortcut for SingleStep with vision. If you are using SingleStep with vision, we recommend that you configure a shortcut to SingleStep as follows in order to start SingleStep with the proper options.

To configure the shortcut, follow the instructions below. Note that below, `OSEKWorksroot` refers to the root OSEKWorks directory:

1 Locate the SingleStep with vision executable, and create a shortcut to it on your desktop. The location of the executable is `OSEKWorksroot\host\sds\7.7.1\cmd\visppc.exe`.

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

3 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
   ```

   You will use this `SingleStep with vision (MPC5xx) for OSEK Target` shortcut when manually downloading code to RAM or FLASH via SingleStep with vision.

Install SinglStep 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, you should 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.

The extension for phyCORE-MPC555 FLASH programming with SingleStep with vision 7.7.3 is provided as a `.zip` file in the SingleStep subdirectory:

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

Unzip this file to obtain the documentation and files required to apply the extension to SingleStep. The Embedded Target for OSEK/VDX presumes this `.zip` file will be unpacked in place, and that the `ppcusr.bin` file found there will be place in the SingleStep `cmd` directory.

After setting up the SingleStep with vision 7.7.3 environment, update the `SingleStep with vision (MPC5xx) for OSEK Target` shortcut you created previously (see “Configuring a Shortcut to SingleStep with vision” on page 2–22) to point to the new version of SingleStep.

Configure phyCORE-MPC555 Jumpers

Make sure that the jumpers on the phyCORE-MPC555 board are set as described in “Jumper Settings” on page 2–5.

Configure SingleStep Parameters

The procedure for configuring SingleStep parameters, downloading code to the target via BDM, and running a debugging session is given in “Tutorial 3: Downloading the Application to RAM via SingleStep” on page 3-18.We recommend that you read that section and generate code, and 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 2–7.

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 "RTW_OPTIONS Section" discussion in the "Targeting Real-Time Systems" chapter of the Real-Time Workshop documentation.

You can extend these options by creating a hook file, `user_osek_options.tlc`. This file must be in the Target Language Compiler path. The `user_osek_options.tlc` file should specify additional elements of the `rtwoptions` structure.

The following example adds a `USER OPTIONS` item to the **Category** menu of the Real-Time Workshop pane. When `USER OPTIONS` is selected, the Real-Time Workshop pane displays an edit filed labeled **User load factor**. This field provides values for the associated TLC and makefile variables, which are used in the build process.

```
/%
  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
 %/
```

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** pop-up 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.

Generating Real-Time OSEK/VDX Applications

This section includes the following topics:

Introduction

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

We suggest the following path through these tutorials:

1. Read and work through "The Multi-Rate Example Model" on page 3-4 to learn about the demo model that is used in tutorials 1-4.
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 3-12.
3. Proceed to "Tutorial 3: Downloading the Application to RAM via SingleStep" on page 3-18 after you have generated an executable. This tutorial walks 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 3-25 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 3-28
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 will need the following components:

- A supported OSEK/VDX implementation and development environment (see "Software Requirements" on page 2–2).
- 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 "Software Requirements" on page 2–2.) 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, you will need to adapt the procedures described below, especially "Tutorial 3: Downloading the Application to RAM via SingleStep" on page 3-18 below. 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 Multi-Rate 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 on this link: `osek_mrate`.

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

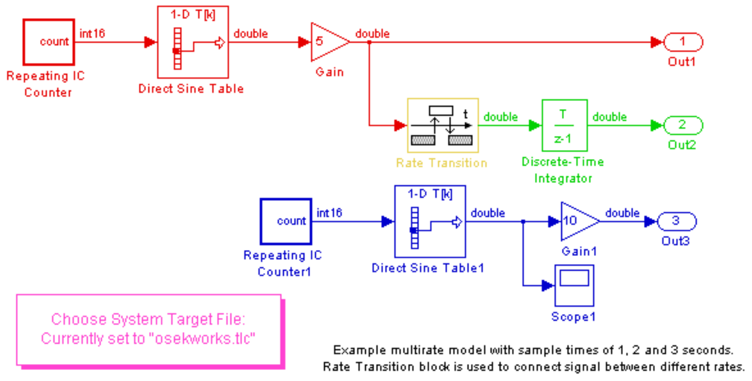
   ```
   osek_mrate
   ```

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. We will work with this copy throughout this exercise.

4 `osek_mrate` is a multi-rate 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 3-12.

Tutorial 1: Creating an Application with OSEKWorks

In this tutorial, we will build a real-time multitasking application for OSEKWorks from a simple model. We assume that you are already familiar with Simulink and with the Real-Time Workshop code generation and build process.

In the following sections, we will

- 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.

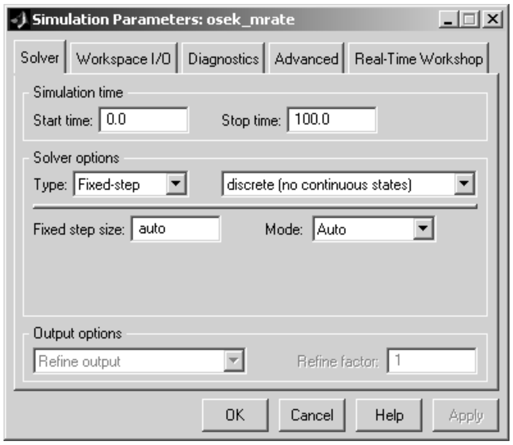
Before You Begin

This tutorial requires specific hardware and software (as described in "Introduction" on page 3-2) in addition to the Embedded Target for OSEK/VDX. Please 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 2–5.
- Installed OSEKWorks as described in "Setting Up Your Installation for the OSEKWorks Target" on page 2–16.
- Installed SingleStep and configured it as described in "Setting Up SingleStep" on page 2–20.
- Set up your target preferences correctly for OSEKWorks, as described in "Setting Target Preferences" on page 2–12

Configuring the Model

1 Open the **Simulation parameters** dialog box and select the Solver pane. Make sure that the Solver parameters are set as shown in the figure below.



Simulation time

Start time: 0.0 Stop time: 100.0

Solver options

Type: Fixed-step discrete (no continuous states)

Fixed step size: auto Mode: Auto

Output options

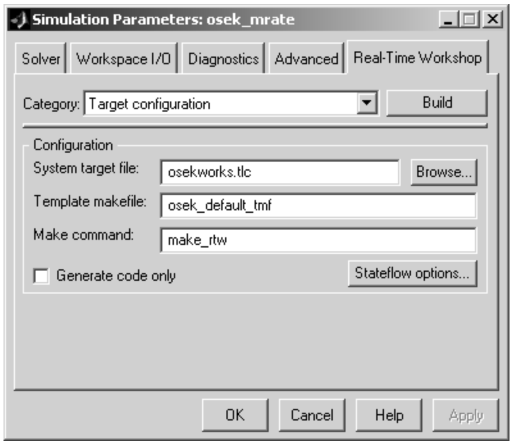
Refine output Refine factor: 1

Observe that the solver **Mode** is set to `Auto`. Since the model has three sample rates, this option will cause the model to execute in `MultiTasking` mode. When code is generated from the model, blocks running at each rate will execute in a separate task.

2 Select the Real-Time Workshop pane. Select `Target configuration` from the **Category** menu.

3 Click on the **Browse** button to open the **System Target File Browser**. In the browser, select `OSEK Target for WRS OSEKWorks Implementation`. Then click **OK** to close the browser and return to the Real-Time Workshop pane.

The `Target configuration` settings should now be as shown in the figure below.



Category: Target configuration

Configuration

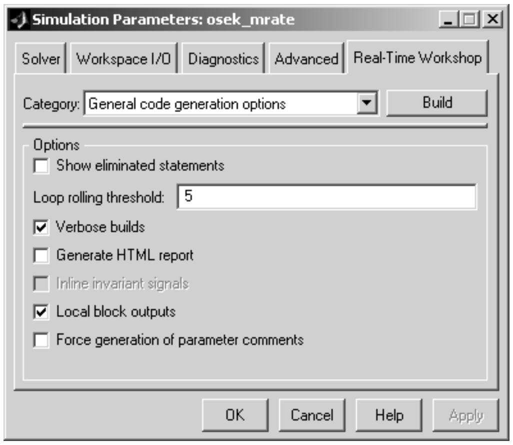
System target file: osekworks.tlc

Template makefile: osek_default_tmf

Make command: make_rtw

☐ Generate code only

4 Select `General code generation options` from the **Category** menu. Make sure the **Generate HTML report** option is off. In this tutorial, we will not generate a code generation report.



Category: General code generation options

Options

☐ Show eliminated statements

Loop rolling threshold: 5

☑ Verbose builds

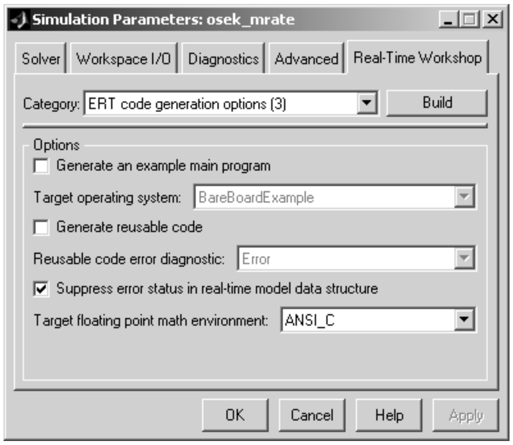
☐ Generate HTML report

☐ Inline invariant signals

☑ Local block outputs

☐ Force generation of parameter comments

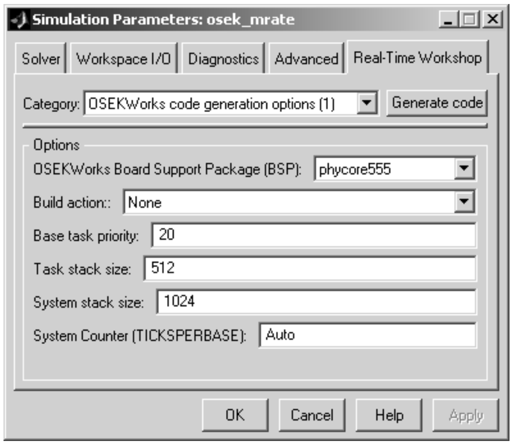
5 Select `ERT code generation options (3)` from the **Category** menu. Make sure that the **Generate an example main program** option is off. This option is not supported, because the Embedded Target for OSEK/VDX does not use the standard main program module generated by Real-Time Workshop Embedded Coder. Instead, the Embedded Target for OSEK/VDX generates its own main program, `osek_main.c`.



Category: ERT code generation options (3)

Options
- [ ] Generate an example main program
- Target operating system: BareBoardExample
- [ ] Generate reusable code
- Reusable code error diagnostic: Error
- [✓] Suppress error status in real-time model data structure
- Target floating point math environment: ANSI_C

6 Select `OSEKWorks code generation options (1)` from the **Category** menu. Make sure that all options are set to their defaults, as shown in this figure.



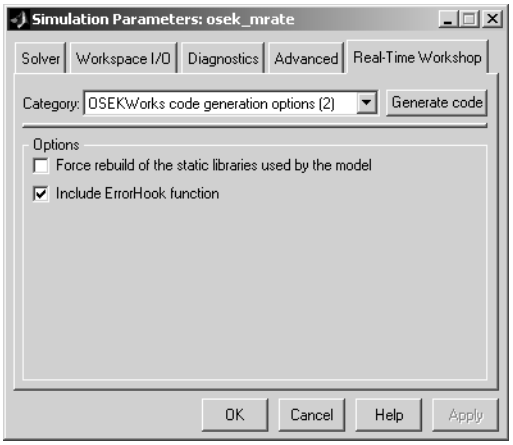
Category: OSEKWorks code generation options (1)

Options
- 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

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 (`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. We 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 3–25 for a description of the other **Build action** options.

7 Select `OSEKWorks code generation options (2)` from the **Category** menu. Make sure that all options are set to their defaults, as shown in this figure.



Category: OSEKWorks code generation options (2)

Options

☐ Force rebuild of the static libraries used by the model

☑ Include ErrorHook function

8 The model is now configured for code generation. Save the model.

Building the Application

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

1 Click the **Build** button 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, the following message is displayed:

```
### 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_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. We 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, we are only concerned 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 3-18.

Tutorial 2: Creating an Application with ProOSEK

In this tutorial, we will build a real-time multitasking application for ProOSEK from a simple model. We assume that you are already familiar with Simulink and with the Real-Time Workshop code generation and build process.

In the following sections, we will

- 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.

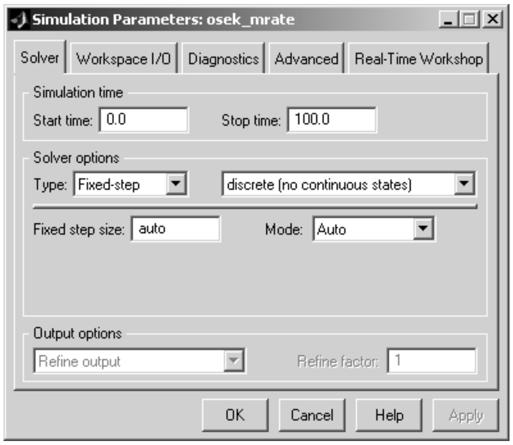
Before You Begin

This tutorial requires specific hardware and software (as described in "Introduction" on page 3-2) in addition to the Embedded Target for OSEK/VDX. Please 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 2–5.
- Installed ProOSEK as described in "Setting Up Your Installation for the ProOSEK Target" on page 2–18.
- Installed SingleStep and configured it as described in "Setting Up SingleStep" on page 2–20.
- Set up your target preferences correctly for ProOSEK, as described in "Setting Target Preferences" on page 2–12

Configuring the Model

1 Open the **Simulation parameters** dialog box and select the Solver pane. Make sure that the Solver parameters are set as shown in the figure below.



Simulation time

Start time: 0.0 Stop time: 100.0

Solver options

Type: Fixed-step discrete (no continuous states)

Fixed step size: auto Mode: Auto

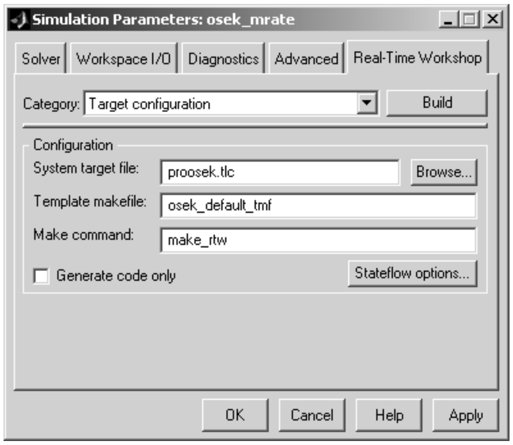
Output options

Refine output Refine factor: 1

Observe that the solver **Mode** is set to `Auto`. Since the model has three sample rates, this option will cause the model to execute in `MultiTasking` mode. When code is generated from the model, blocks running at each rate will execute in a separate task.

2 Select the Real-Time Workshop pane. Select `Target configuration` from the **Category** menu.

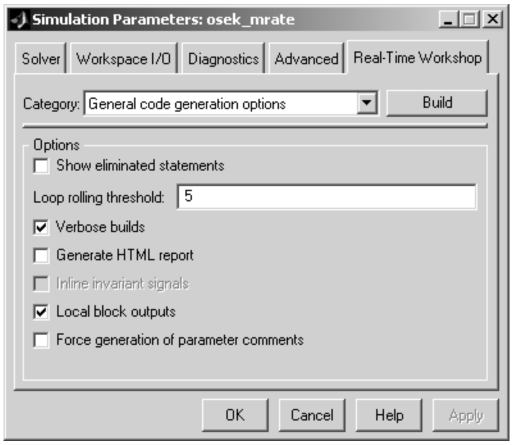
3 Click on the **Browse** button to open the **System Target File Browser**. In the browser, select `OSEK Target for 3soft ProOSEK Implementation`. Then click **OK** to close the browser and return to the Real-Time Workshop pane.The `Target configuration` settings should now be as shown in the figure below.


Simulation Parameters: osek_mrate

Category: Target configuration

Configuration
System target file: proosek.tlc
Template makefile: osek_default_tmf
Make command: make_rtw
[ ] Generate code only

4 Select `General code generation options` from the **Category** menu. Make sure the **Generate HTML report** option is off. In this tutorial, we will not generate a code generation report.

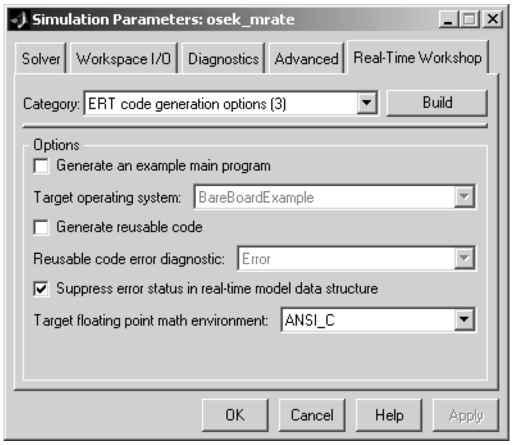

Simulation Parameters: osek_mrate

Category: General code generation options

Options
[ ] Show eliminated statements
Loop rolling threshold: 5
[✓] Verbose builds
[ ] Generate HTML report
[ ] Inline invariant signals
[✓] Local block outputs
[ ] Force generation of parameter comments

5 Select `ERT code generation options (3)` from the **Category** menu. Make sure that the **Generate an example main program** option is off. This option is not supported, because the Embedded Target for OSEK/VDX does not use

the standard main program module provided by Real-Time Workshop Embedded Coder. Instead, the Embedded Target for OSEK/VDX generates its own main program, `osek_main.c`.



Simulation Parameters: osek_mrate

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

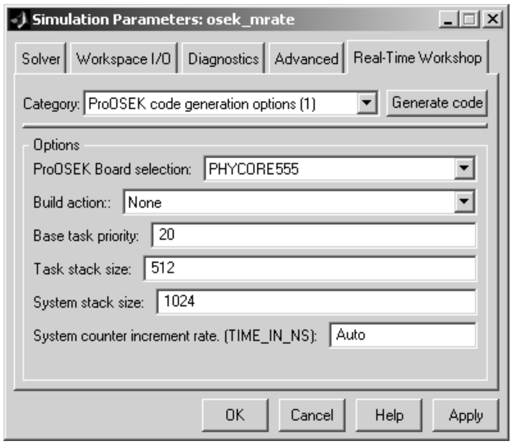
Category: ERT code generation options (3) — Build

Options

- [ ] Generate an example main program
- Target operating system: BareBoardExample
- [ ] Generate reusable code
- Reusable code error diagnostic: Error
- [✓] Suppress error status in real-time model data structure
- Target floating point math environment: ANSI_C

OK | Cancel | Help | Apply

6 Select `ProOSEK code generation options (1)` from the **Category** menu. Make sure that all options are set to their defaults, as shown in this figure.



Simulation Parameters: osek_mrate

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Category: ProOSEK code generation options (1) — Generate code

Options

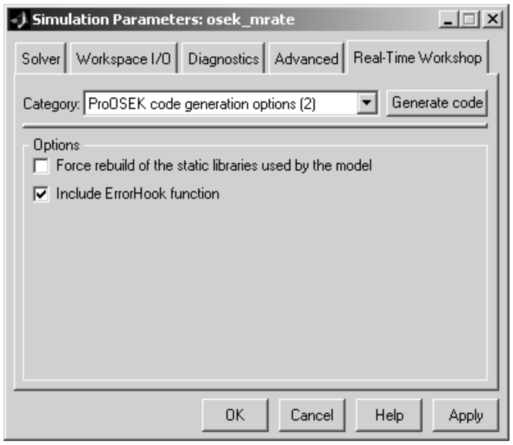
- ProOSEK Board selection: PHYCORE555
- Build action:: None
- Base task priority: 20
- Task stack size: 512
- System stack size: 1024
- System counter increment rate. (TIME_IN_NS): Auto

OK | Cancel | Help | Apply

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 (`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. We 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 3–25 for a description of the other **Build action** options.

7 Select `ProOSEK code generation options (2)` from the **Category** menu. Make sure that all options are set to their defaults, as shown in this figure



Simulation Parameters: osek_mrate

Category: ProOSEK code generation options (2)

Options

☐ Force rebuild of the static libraries used by the model

☑ Include ErrorHook function

8 The model is now configured for code generation. Save the model.

Building the Application

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

1 Click the **Build** button 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, the following message is displayed:

```
### 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. We 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, we are only concerned 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 3-18.

Tutorial 3: Downloading the Application to RAM via SingleStep

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

We assume that the target system is a Phytec PhyCORE-MPC555 board. We 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. We 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 version 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:

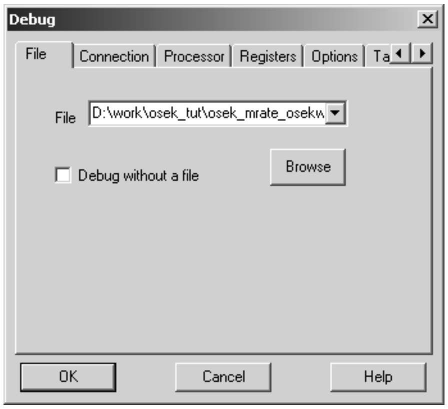
- Configure a shortcut to SingleStep that starts up SingleStep with the correct options (see "Setting Up SingleStep" on page 2-20).
- Connect the BDM port of your development board to parallel port LPT1 of your host PC.
- Make sure that the jumpers on the phyCORE-MPC555 board are set as described in "Jumper Settings" on page 2-5.
- 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 2-20).

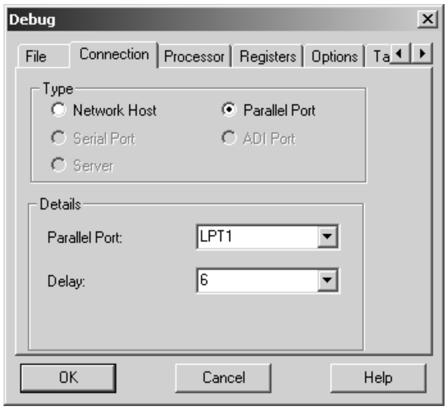
2 The **Debug** dialog opens. Click on the **File** tab. Clear the **Debug without a file** option. Then, use the **Browse** button to locate the `osek_mrate.elf` file.



File: D:\work\osek_tut\osek_mrate_osekw

☐ Debug without a file

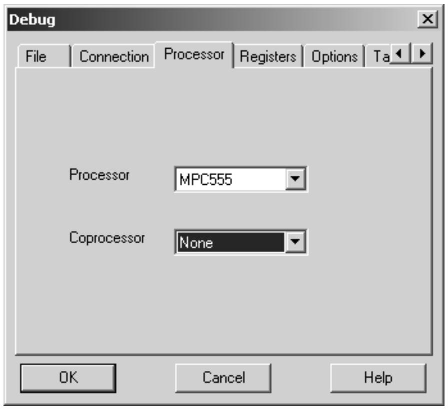
3 Click the **Connection** tab. Choose parallel port or network settings appropriate to the physical connection you will be using between your PC and PhyCORE-MPC555 board. In the picture below, connection options are configured for the parallel port `LPT1`.



Type
○ Network Host ● Parallel Port
○ Serial Port ○ ADI Port
○ Server

Details
Parallel Port: LPT1
Delay: 6

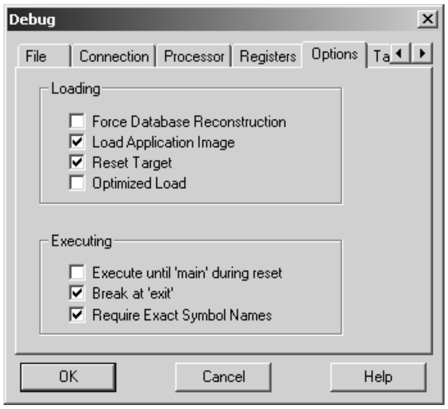
4 Click on the **Processor** tab. Confirm that the `MPC555` is selected in the **Processor** list, as shown.



Processor: MPC555

Coprocessor: None

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

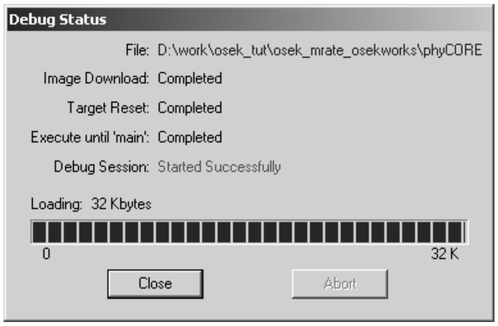


Loading
- [ ] Force Database Reconstruction
- [x] Load Application Image
- [x] Reset Target
- [ ] Optimized Load

Executing
- [ ] Execute until 'main' during reset
- [x] Break at 'exit'
- [x] Require Exact Symbol Names

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 picture shows the **Debug Status** window after a successful connection and download.


Debug Status

File: D:\work\osek_tut\osek_mrate_osekworks\phyCORE
Image Download: Completed
Target Reset: Completed
Execute until 'main': Completed
Debug Session: Started Successfully

Loading: 32 Kbytes

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.

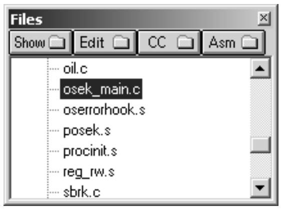
9 At this point, the application code is in RAM, and SingleStep has established a debugging session.

In the next section we 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, we 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 labelled **Files** at the bottom of the SingleStep **Debug** window. Click on this arrow to display the **Files** selection pane. Select `osek_main.c` from the list of files, as shown below.



2 Double-click on 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 (see Figure 3-1) is activated once during the OSEK/VDX startup phase. `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.

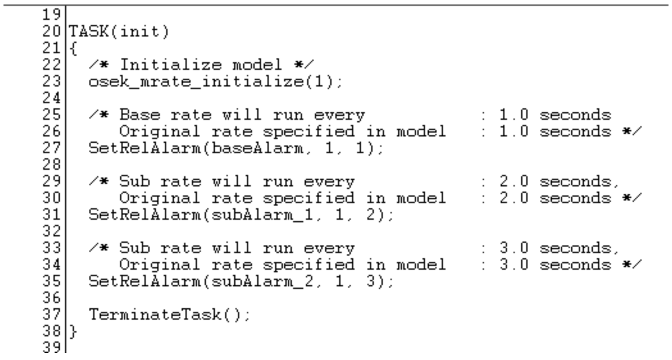
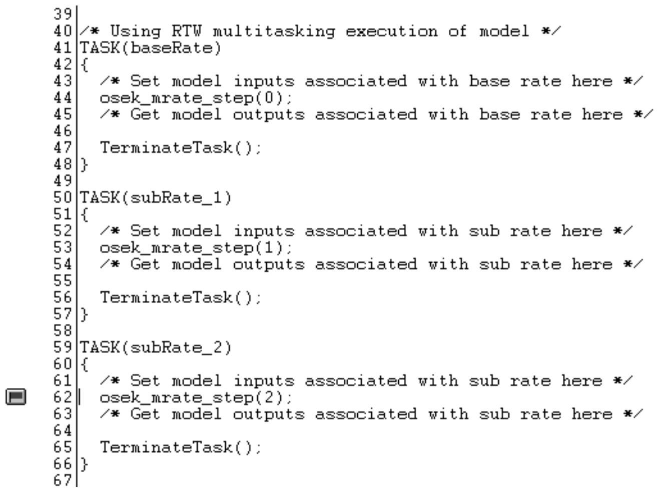

```

TASK(init)
{
  /* Initialize model */
  osek_mrate_initialize(1);

  /* Base rate will run every           : 1.0 seconds
     Original rate specified in model   : 1.0 seconds */
  SetRelAlarm(baseAlarm, 1, 1);

  /* Sub rate will run every            : 2.0 seconds,
     Original rate specified in model   : 2.0 seconds */
  SetRelAlarm(subAlarm_1, 1, 2);

  /* Sub rate will run every            : 3.0 seconds,
     Original rate specified in model   : 3.0 seconds */
  SetRelAlarm(subAlarm_2, 1, 3);

  TerminateTask();
}

```

Figure 3-1: init Task Code as Listed in SingleStep

4 The generated code also creates three OSEK/VDX tasks (see Figure 3-2) that are activated by each of the three alarms, and therefore run at the corresponding rate:

- The `baseRate` task is activated by `baseAlarm`. `baseRate` calls `model_step`, passing in task identifier (`tid`) 0, indicating that blocks running at the model's base rate should execute.
- `subRate_1` task: This task is activated by `subAlarm_1`. `subRate_1` calls `model_step`, passing in `tid` 1, indicating that blocks running at the 2 Hz rate should execute.
- `subRate_2` task: This task is activated by `subAlarm_2`. `subRate_2` calls `model_step`, passing in `tid` 1, indicating that blocks running at the 3 Hz rate should execute.



```
/* Using RTW multitasking execution of model */
TASK(baseRate)
{
  /* Set model inputs associated with base rate here */
  osek_mrate_step(0);
  /* Get model outputs associated with base rate here */

  TerminateTask();
}

TASK(subRate_1)
{
  /* Set model inputs associated with sub rate here */
  osek_mrate_step(1);
  /* Get model outputs associated with sub rate here */

  TerminateTask();
}

TASK(subRate_2)
{
  /* Set model inputs associated with sub rate here */
  osek_mrate_step(2);
  /* Get model outputs associated with sub rate here */

  TerminateTask();
}

```

Figure 3-2: osek_main TASK Definitions, with Breakpoint at subRate_2 Task

5 Set a breakpoint at the beginning of the `subRate_2` task, as shown in Figure 3-2. We 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 will occur every 3 seconds. You can verify this as follows: Press 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 3–25, we 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 optional initiation of a debugging session. The SingleStep debugger is required to use these features.

The **Build action** pull-down 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.

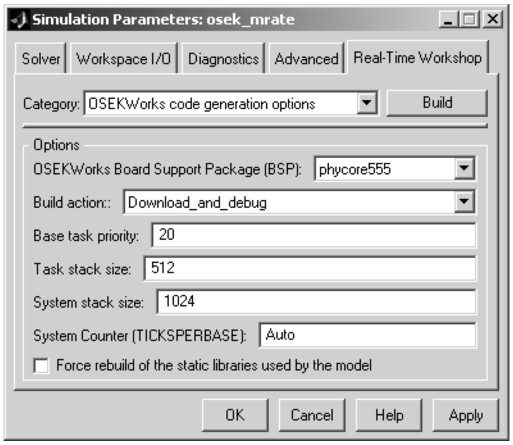
In this tutorial, we will rebuild the `osek_mrate` application used in the previous tutorials, and use the `Download_and_debug` option to download the code and start a debugging session.

Make sure you have done the following before you begin:

- Make sure you have set the debugger-related target properties (`Debugger`, `DebuggerEXE`, `DebuggerPath, Debugger`, `DebuggerSwitches`) correctly for your SingleStep installation, as described in “Setting Target Preferences” on page 2-12.
- Cycle the power (or perform a hard reset) on your development board to set the board to a known state.

1 Return to the Real-Time Workshop pane of the **Simulation parameters** dialog box and select the target-specific options for your implementation (either `OSEKWorks code generation options` or `ProOSEK code generation options`) from the **Category** menu.

2 Select `Download_and_debug` from the **Build action** pull-down menu. The figure below shows this option selected for the OSEKWorks target.


Simulation Parameters: osek_mrate

Category: OSEKWorks code generation options

Options

OSEKWorks Board Support Package (BSP): phycore555

Build action:: Download_and_debug

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

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

4 On successful completion of the build process, messages similar to the following are displayed, 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
```

5 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.

6 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.

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

8 You may also want to try the `Download_and_run` option. If so, make sure you have closed any existing SingleStep sessions, as multiple SingleStep sessions can conflict with each other. Then return to the **Build action** pull-down menu, select `Download_and_run`, and click the **Build** button. 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 3-28, we will 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, we will generate code from a different model than that used in the previous tutorials. We 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 2–20
- “Configuring SingleStep with vision” on page 2–21

The osek_led Demo Model

Since we will be executing the generated code from FLASH, we 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:

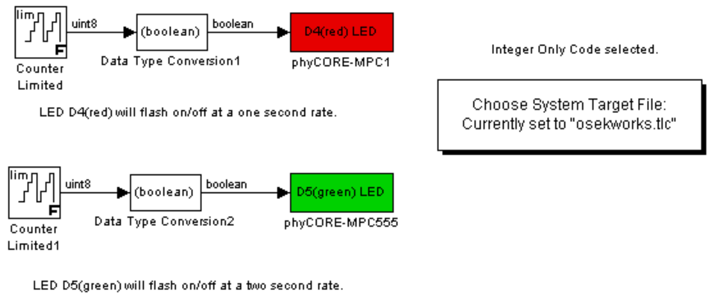
1 Open the model. If you are reading this document online in the MATLAB Help browser, you can open the model by clicking on this link: `osek_led`.

   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. We 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.


uint8
(boolean)
boolean
D4(red) LED
Counter Limited
Data Type Conversion1
phyCORE-MPC1
LED D4(red) will flash on/off at a one second rate.
Integer Only Code selected.
Choose System Target File: Currently set to "osekworks.tlc"
uint8
(boolean)
boolean
D5(green) LED
Counter Limited1
Data Type Conversion2
phyCORE-MPC555
LED D5(green) will flash on/off at a two second rate.

4 Before building the model, open the **Simulation parameters** dialog box and open the Real-Time Workshop pane. Deselect the **Generate code only** option in the `Target configuration` category. You may also want to deselect the **Generate HTML report** option in the `General code generation options` category.

5 Click **Apply**.

6 Click the **Build** button. 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

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

- SingleStep On-Chip (v. 7.6.2): Proceed to “Downloading Generated Code to FLASH with SingleStep On-Chip (v. 7.6.2)” on page 3-30.
- SingleStep with vision: Proceed to “Downloading Generated Code to FLASH with SingleStep with vision” on page 3-36.

Downloading Generated Code to FLASH with SingleStep On-Chip (v. 7.6.2)

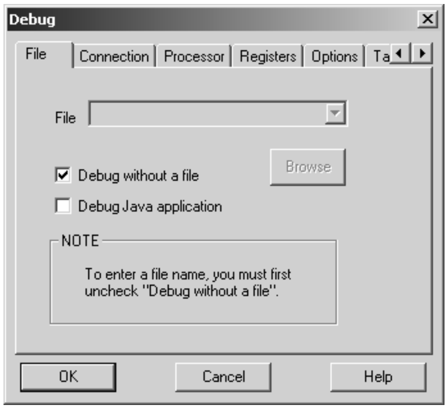
This section describes how to download and debug the generated `osek_led.bin` file to FLASH memory on the target, via SingleStep On-Chip (v. 7.6.2).

Connect to Target With FLASH Enabled

Before programming `osek_led.bin` into FLASH, it is necessary to set the FLASH Enable (`FLEN`) bit on the target. In this section, we will 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 2-20).

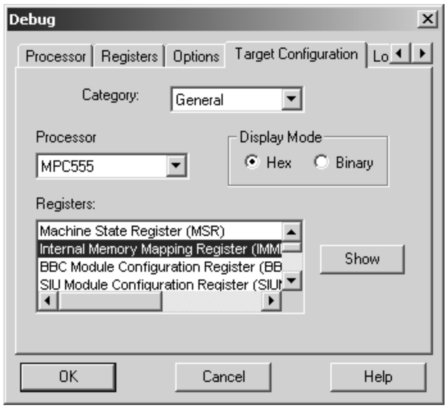
2 The **Debug** dialog box opens. Click on the **File** tab. Select the **Debug without a file** check box, as shown.



File:
[✓] Debug without a file
[ ] Debug Java application

NOTE
To enter a file name, you must first uncheck "Debug without a file".

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



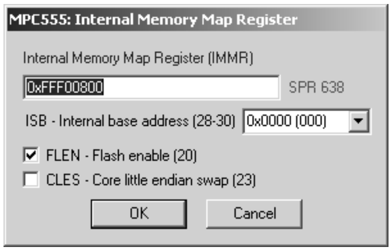
Category: General

Processor: MPC555

Display Mode: [✓ Hex] Binary

Registers:
Machine State Register (MSR)
Internal Memory Mapping Register (IMM
BBC Module Configuration Register (BB
SIU Module Configuration Register (SIU

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



Internal Memory Map Register (IMMR): 0xFFF00800 SPR 638

ISB - Internal base address (28-30): 0x0000 (000)

☑ FLEN - Flash enable (20)

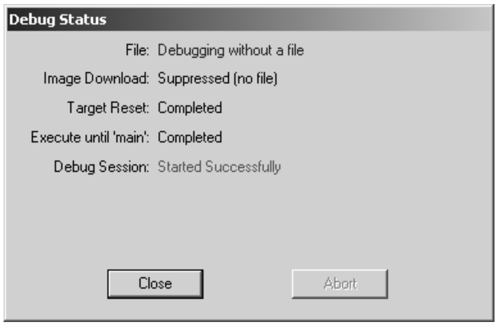
☐ CLES - Core little endian swap (23)

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

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

---

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.



Debug Status

File: Debugging without a file
Image Download: Suppressed (no file)
Target Reset: Completed
Execute until 'main': Completed
Debug Session: Started Successfully

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.

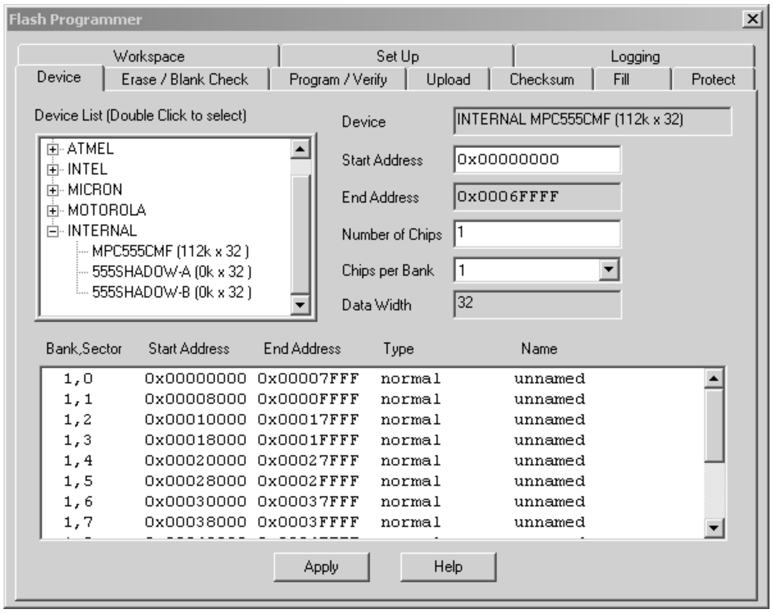
Download Code and Execute in FLASH Memory

The next step is to download the generated code (`osek_led.bin`) to FLASH memory, using the SingleStep **Flash Programmer** dialog box, 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 on the **Flash** button on the toolbar. The **Flash Programmer** dialog box opens.

3 Click on the **Device** tab. In the **Device List**, double-click on `INTERNAL.` Then select `MPC555CMF` (as shown in this figure), and double-click on this list entry.



Device List (Double Click to select)

- ATMEL
- INTEL
- MICRON
- MOTOROLA
- INTERNAL
  - MPC555CMF (112k x 32)
  - 555SHADOW-A (0k x 32)
  - 555SHADOW-B (0k x 32)

Device	INTERNAL MPC555CMF (112k x 32)
Start Address	0x00000000
End Address	0x0006FFFF
Number of Chips	1
Chips per Bank	1
Data Width	32

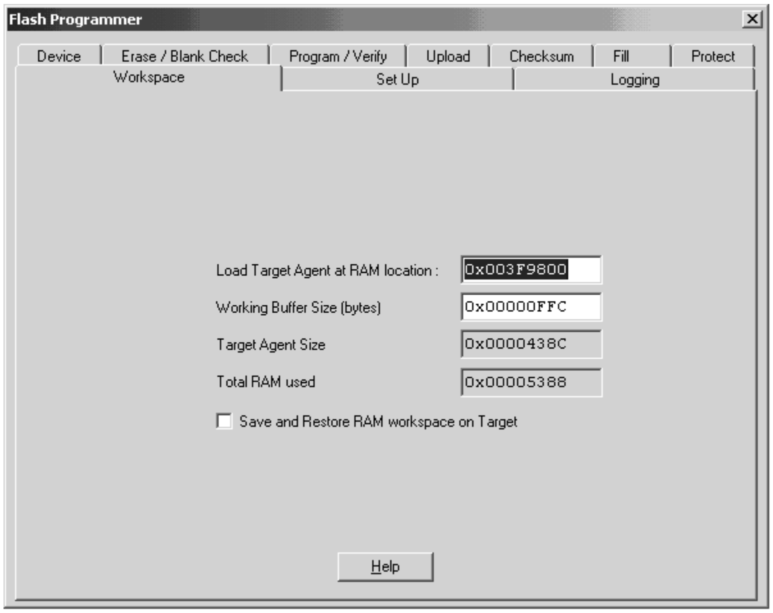
Bank,Sector	Start Address	End Address	Type	Name
1,0	0x00000000	0x00007FFF	normal	unnamed
1,1	0x00008000	0x0000FFFF	normal	unnamed
1,2	0x00010000	0x00017FFF	normal	unnamed
1,3	0x00018000	0x0001FFFF	normal	unnamed
1,4	0x00020000	0x00027FFF	normal	unnamed
1,5	0x00028000	0x0002FFFF	normal	unnamed
1,6	0x00030000	0x00037FFF	normal	unnamed
1,7	0x00038000	0x0003FFFF	normal	unnamed

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

5 Click on the **Apply** button.

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

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



Load Target Agent at RAM location : 0x003F9800

Working Buffer Size (bytes) 0x00000FFC

Target Agent Size 0x0000438C

Total RAM used 0x00005388

☐ Save and Restore RAM workspace on Target

7 Click on 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 file name 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`

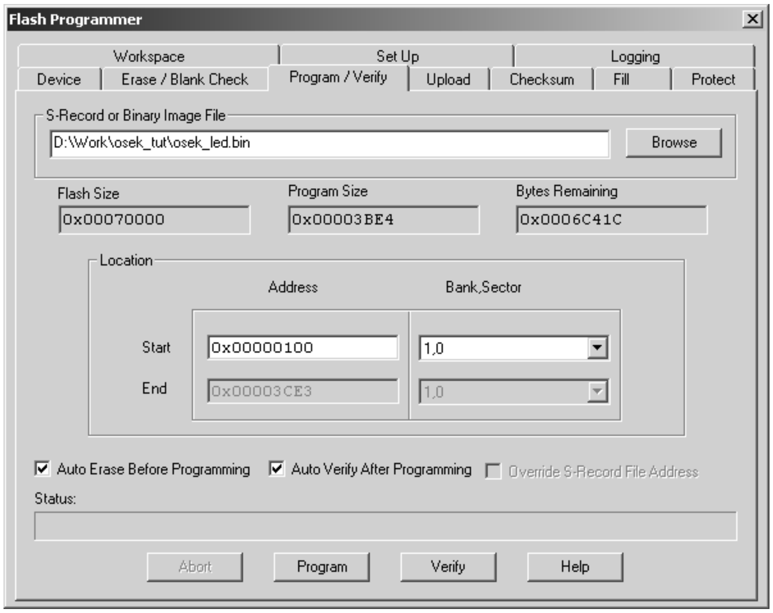
If you generated code using the ProOSEK target set the **Start** fields as follows:

- **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.



S-Record or Binary Image File: D:\Work\osek_tut\osek_led.bin

Flash Size: 0x00070000
Program Size: 0x00003BE4
Bytes Remaining: 0x0006C41C

Location

	Address	Bank,Sector
Start	0x00000100	1,0
End	0x00003CE3	1,0

[✓] Auto Erase Before Programming [✓] Auto Verify After Programming [ ] Override S-Record File Address

Status:

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 box.

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 box. 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 2-5.
- Quit SingleStep and repeat the entire procedure starting at "Connect to Target With FLASH Enabled" on page 3-30. Make sure, in step 5, that you click **OK** (not **Cancel**) when closing the **Internal Memory Mapping Register** dialog box. Otherwise, SingleStep may use settings other than those shown in the dialog box.
- 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 one 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 SinglStep 7.7.3 and Apply Patches for Programming FLASH memory" on page 2–23.

Convert S-Rec File to Binary Executable File

When using SingleStep with vision to download code to FLASH, it is necessary to 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, type

```
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`.

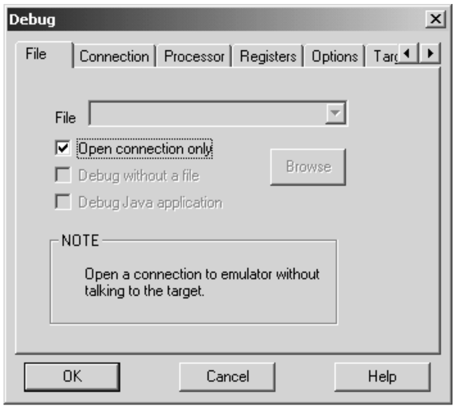
In this case, the final binary file is `osek_led_est.bin`.

Connect to Target With FLASH Enabled

Before programming `osek_led_est.bin` into FLASH, it is necessary to set the FLASH Enable (`FLEN`) bit on the target, and make sure that the Software Watchdog Timer is disabled. In this section, we 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 2-21).

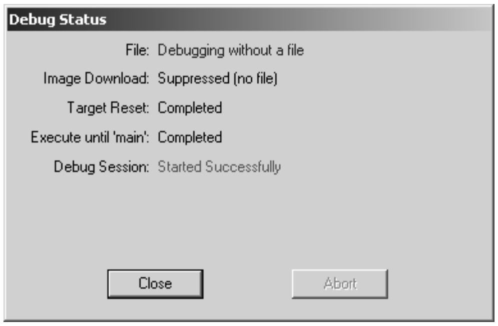
2 The **Debug** dialog box opens. Click on the **File** tab. Select the **Open connection only** check box, as shown.



[✓] Open connection only
[ ] Debug without a file
[ ] Debug Java application

NOTE
Open a connection to emulator without talking to the target.

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.



Debug Status

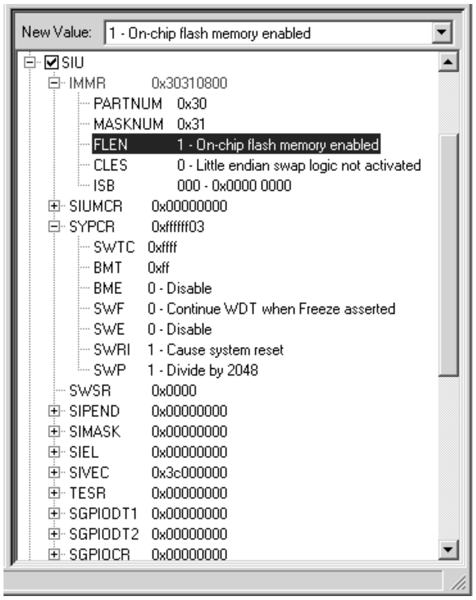
File: Debugging without a file
Image Download: Suppressed (no file)
Target Reset: Completed
Execute until 'main': Completed
Debug Session: Started Successfully

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** pull-down menu in the SingleStep toolbar, select the **Debug** window.

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



```
New Value: 1 - On-chip flash memory enabled
SIU
  IMMR        0x30310800
    PARTNUM   0x30
    MASKNUM   0x31
    FLEN      1 - On-chip flash memory enabled
    CLES      0 - Little endian swap logic not activated
    ISB       000 - 0x0000 0000
  SIUMCR      0x00000000
  SYPCR       0xffffff03
    SWTC  0xffff
    BMT   0xff
    BME   0 - Disable
    SWF   0 - Continue WDT when Freeze asserted
    SWE   0 - Disable
    SWRI  1 - Cause system reset
    SWP   1 - Divide by 2048
  SWSR        0x0000
  SIPEND      0x00000000
  SIMASK      0x00000000
  SIEL        0x00000000
  SIVEC       0x3c000000
  TESR        0x00000000
  SGPIODT1    0x00000000
  SGPIODT2    0x00000000
  SGPIOCR     0x00000000
```

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`

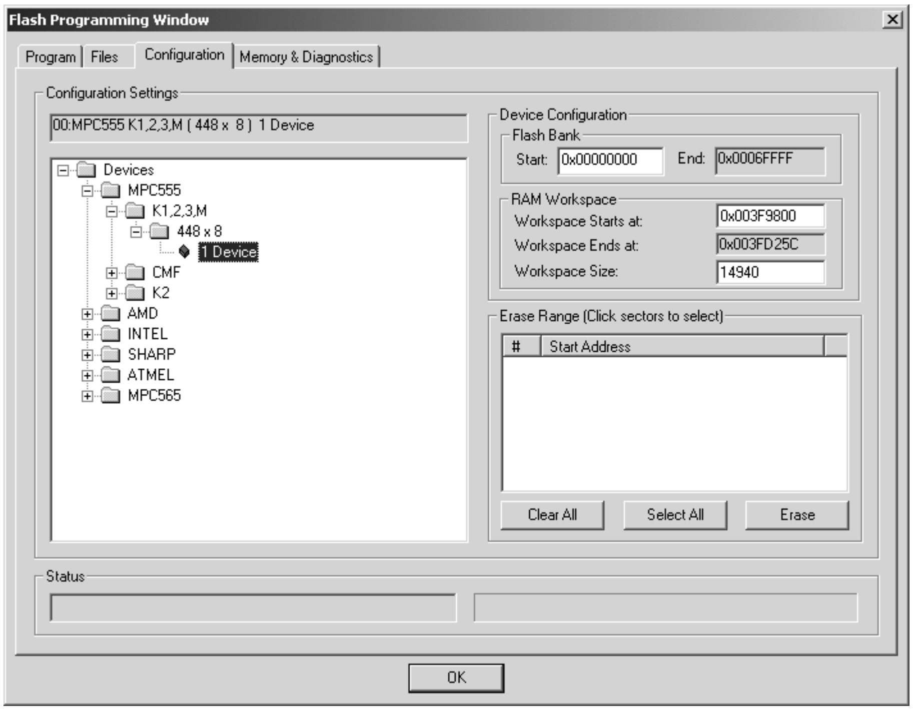
FLASH Memory is now enabled. In the next section, you will download and execute the `.bin` file.

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 pull-down menu in the SingleStep toolbar.

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


Flash Programming Window

Program | Files | Configuration | Memory & Diagnostics

Configuration Settings

00:MPC555 K1,2,3,M (448 x 8) 1 Device

```
Devices
  MPC555
    K1,2,3,M
      448 x 8
        1 Device
    CMF
    K2
  AMD
  INTEL
  SHARP
  ATMEL
  MPC565
```

Device Configuration

Flash Bank

Start: 0x00000000 End: 0x0006FFFF

RAM Workspace

Workspace Starts at: 0x003F9800

Workspace Ends at: 0x003FD25C

Workspace Size: 14940

Erase Range (Click sectors to select)

#	Start Address

Clear All | Select All | Erase

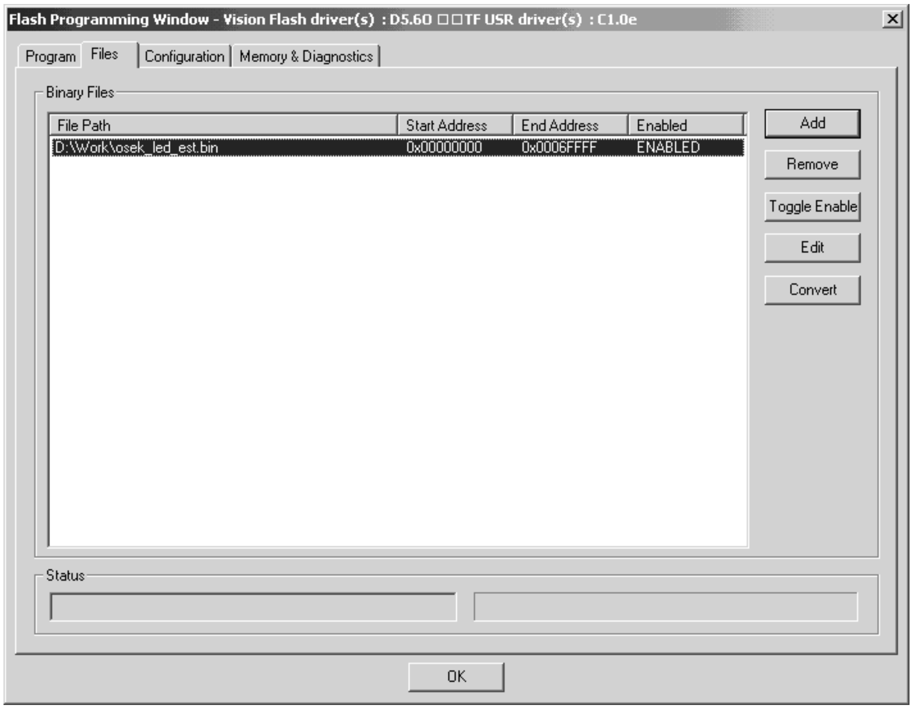
Status

OK

3 In the **Configuration** pane, and 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 performed successfully. See “Configuring SingleStep with vision” on page 2–21 and make sure that the FLASH programming extensions are installed correctly. If problems persist, contact Wind River Systems for technical support.

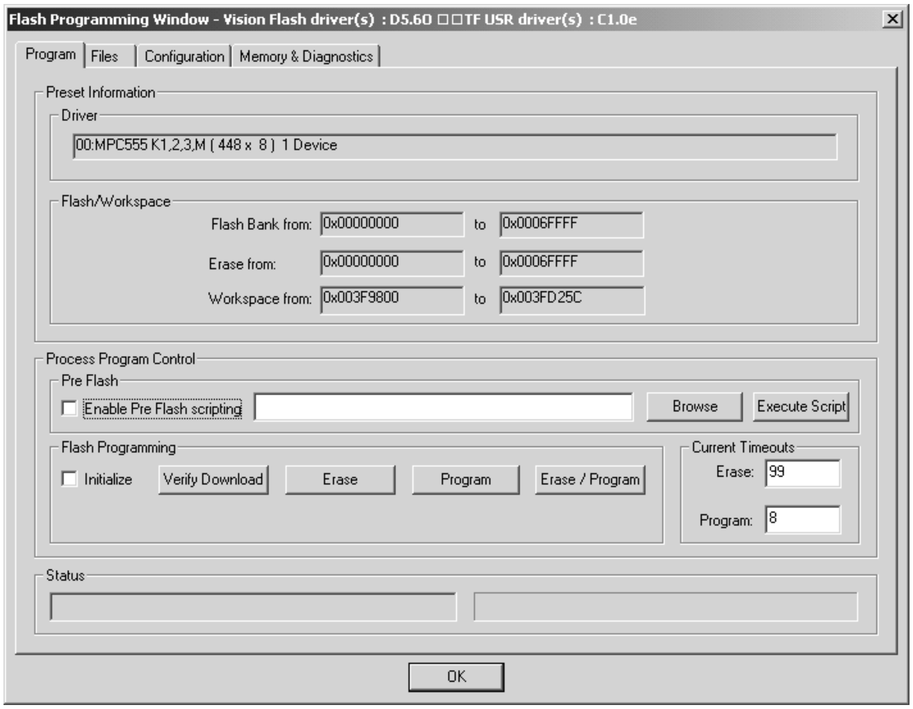
4 Select the **Files** tab in the **Flash Programming** window. Use the **Add** button to navigate to the generated `osek_led_est.bin` file in the build directory. After finding the `.bin` file, use the **Open** button to place this file in the **Binary files** list.



File Path	Start Address	End Address	Enabled
D:\Work\osek_led_est.bin	0x00000000	0x0006FFFF	ENABLED

5 Select `osek_led_est.bin` file in the **Binary files** list. Click on the **Toggle Enable** button.

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


Flash Programming Window - Vision Flash driver(s) : D5.60 □□TF USR driver(s) : C1.0e

Preset Information

Driver: 00:MPC555 K1,2,3,M (448 x 8) 1 Device

Flash/Workspace

Flash Bank from: 0x00000000 to 0x0006FFFF

Erase from: 0x00000000 to 0x0006FFFF

Workspace from: 0x003F9800 to 0x003FD25C

Process Program Control

Pre Flash: ☐ Enable Pre Flash scripting

Flash Programming: ☐ Initialize

Current Timeouts

Erase: 99

Program: 8

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 Programmer** dialog box.

10 If FLASH programming fails, you should

- Check that all jumpers are set correctly as described in “Jumper Settings” on page 2-5.
- Quit SingleStep and repeat the entire procedure starting at “Connect to Target With FLASH Enabled” on page 3-37. Make sure, in step 8, that your settings are applied correctly. Otherwise, SingleStep may use settings other than those shown in the dialog box.

- 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 the Reset button on the target board. Alternatively, cycle power on the target board.

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

Generating Code, Calibration Data, and Reports

This section includes the following topics:

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.

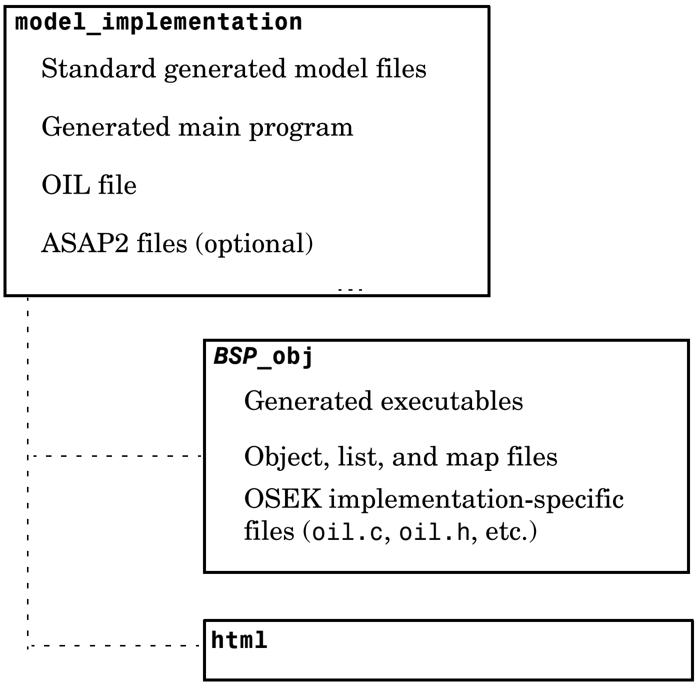

model_implementation
Standard generated model files
Generated main program
OIL file
ASAP2 files (optional)
...
BSP_obj
Generated executables
Object, list, and map files
OSEK implementation-specific files (oil.c, oil.h, etc.)
html

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`). The main program 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 (optional; generated if the **Generate ASAP2 file** option is selected.

Output Subdirectory (BSP_obj)

This subdirectory contains the final output of the build process: the executable code file(s), as well as a number of 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 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.

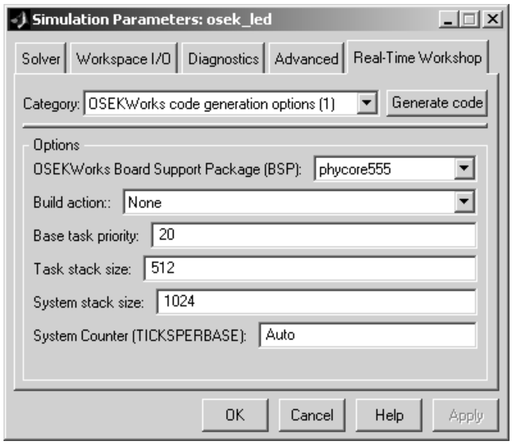
Code Generation Options

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 via the **Category** menu of the Real-Time Workshop pane of the **Simulation Parameters** dialog box; they are documented in the Real-Time Workshop documentation and the Real-Time Workshop Embedded Coder documentation.

Some code generation options of the ERT target are not relevant to the Embedded Target for OSEK/VDX, and are either 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 several target-specific code generation options. To view or change the setting of these options, select `OSEKWorks code generation options (1)` or `OSEKWorks code generation options (2)`from the **Category** menu of the Real-Time Workshop pane of the **Simulation Parameters** dialog box. The following figures show the options at their default settings.



Category: OSEKWorks code generation options (1)

OSEKWorks Board Support Package (BSP): phycore555

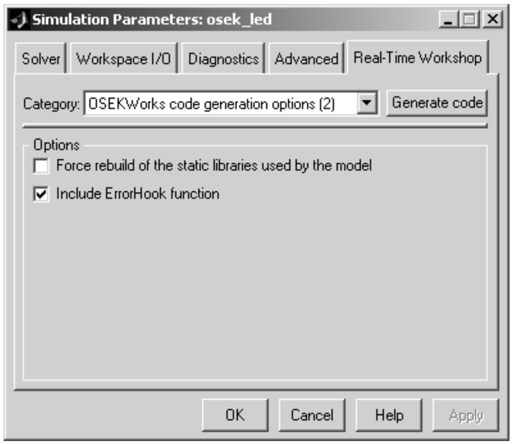
Build action:: None

Base task priority: 20

Task stack size: 512

System stack size: 1024

System Counter (TICKSPERBASE): Auto



Category: OSEKWorks code generation options (2)

Options

☐ Force rebuild of the static libraries used by the model

☑ Include ErrorHook function

The options are

- **OSEKWorks Board Support Package (BSP)**: This pull-down menu selects one of the supported BSPs for use in code generation. The default is `phycore555`.
- **Build action:** This pull-down menu controls what action, if any, the build process takes after the target executable has been 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`: SingleStep is not invoked. You must download and run or debug the code manually.

  The default **Build action** is `None`.

  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**: Each OSEK/VDX task is assigned a priority from 0 to 255, with higher numbers signifying higher priority. The **Base task priority** is the priority assigned to the base rate (fastest) task in the model. Subrate tasks are assigned successively lower numbers.

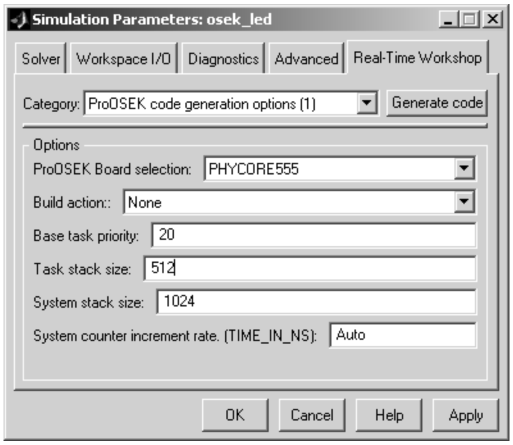
- **Task stack size**: Stack size, in bytes, allocated to each task in the model. (See "Setting System and Task Stack Size" on page 4–10.)
- **System stack size**: Stack size, in bytes, allocated to the OSEK/VDX kernel. (See "Setting System and Task Stack Size" on page 4–10.)
- **System counter (TICKSPERBASE)**: 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 either an integer or as `Auto`. If `Auto` is specified, 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**: This option controls whether or not object file libraries (such as `rtwlib`) referenced by the model are rebuilt during the build process. By default, this option is deselected, and existing object libraries are not rebuilt. Since rebuilding such libraries may involve compiling a large number of source files, we recommend the use of the default. See "Efficient Use of Persistent Object Libraries" on page 4-10 for further information.
- **Include ErrorHook function**: If this option is selected, a default OSEK `ErrorHook` function is generated. The `ErrorHook` function is generated in the `osek_main.c` file. By default, **Include ErrorHook function** is selected.

Target-Specific Options for ProOSEK Target

The ProOSEK Target has several target-specific code generation options. To view or change the setting of these options, select `ProOSEK code generation options (1)` or `ProOSEK code generation options (2)`from the **Category** menu of the Real-Time Workshop pane of the **Simulation Parameters** dialog box. The following figures show the options at their default settings.


Simulation Parameters: osek_led

Category: ProOSEK code generation options (1)

Options

ProOSEK Board selection: PHYCORE555

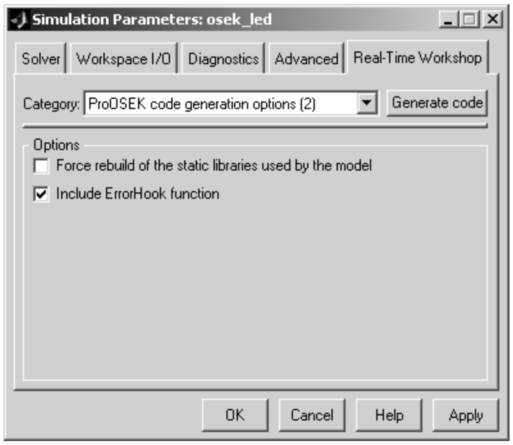
Build action:: None

Base task priority: 20

Task stack size: 512

System stack size: 1024

System counter increment rate. (TIME_IN_NS): Auto


Simulation Parameters: osek_led

Category: ProOSEK code generation options (2)

Options

- [ ] Force rebuild of the static libraries used by the model
- [✓] Include ErrorHook function

The options are

- **ProOSEK Board selection**: This pull-down menu selects one of the supported Boards for use in code generation. The default is `PHYCORE555`.
- **Build action:** This pull-down menu controls what action, if any, the build process takes after the target executable has been 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`: SingleStep is not invoked. You must download and run or debug the code manually.

  The default **Build action** is `None`.

  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**: Each OSEK/VDX task is assigned a priority from 0 to 255, with higher numbers signifying higher priority. The **Base task priority** is the priority assigned to the base rate (fastest) task in the model. Subrate tasks are assigned successively lower numbers.
- **Task stack size**: Stack size, in bytes, allocated to each task in the model. (See "Setting System and Task Stack Size" on page 4–10.)
- **System counter increment rate (TIME_IN_NS)**: The period, in nanoseconds, for the OSEK/VDX system counter.

  You can specify **System counter** as either an integer or as `Auto`. If `Auto` is specified, the Embedded Target for OSEK/VDX determines a maximum value for `TIME_IN_NS`, thus minimizing interrupt and counter overhead. `Auto` is the default value.
- **Force rebuild of the static libraries used by the model**: This option controls whether or not object file libraries (such as `rtwlib`) referenced by the model are rebuilt during the build process. By default, this option is deselected, and existing object libraries are not rebuilt. Since rebuilding such libraries may involve compiling a large number of source files, we recommend the use of the default. See "Efficient Use of Persistent Object Libraries" on page 4-10 for further information.

- **Include ErrorHook function**: If this option is selected, a default OSEK `ErrorHook` function is generated. The `ErrorHook` function is generated in the `osek_main.c` file. By default, **Include ErrorHook function** 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 OSEWorks 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 OS API calls made at run time

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

- Some types of interrupts
- OSEK OS API calls made at run time
- 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 that the build process rebuilding such libraries where 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 2–12). 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 the “MATLAB Classes and Objects” in the MATLAB 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 2-12).

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 occurance 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

Certain ERT code generation options are not supported (or are restricted) by the Embedded Target for OSEK/VDX. If these options are selected, the Embedded Target for OSEK/VDX either ignores the option or issues an error message during the build process. Table 4-1 summarizes these restricted options.

Table 4-1: Embedded Target for OSEK/VDX Restricted Code Generation Options

Option	Restriction
MAT-file logging	Ignored if selected; build process terminates
Create Simulink (S-Function) block	Error if selected; build process terminates
External mode	Error if selected; build process terminates
Generate an example main program	Error if selected; build process terminates. This option should not be selected for the Embedded Target for OSEK/VDX. The Embedded Target for OSEK/VDX generates a target-specific main program, `osek_main.c`.
Generate reusable code	Error if selected; build process terminates
Suppress error status in real-time model data structure	Selected by default. Warning generated if not selected.
Target floating point math environment	For ProOSEK target only, the `ISO_C` option is not supported, and an error occurs if `ISO_C` is selected, followed by build process termination.

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 the "Generating ASAP2 Files" section of the Real-Time Workshop Embedded Coder documentation. That section describes how to define the signal and parameter information required by the ASAP2 file generation process.

The process of generating an ASAP2 file from your model with 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

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

2 Define the corresponding `ASAP2.Parameter` and `ASAP2.Signal` objects in the MATLAB workspace.

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

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

   `ExportedGlobal` is the default storage class.

4 Configure the other data object properties such as `LongID_ASAP2`, `PhysicalMin_ASAP2`, etc.

5 In the Advanced pane of the **Simulation Parameters** dialog box, select the **Inline parameters** option.

   Note that you should *not* configure the parameters associated with your ASAP2 data objects in the **Model Parameter Configuration** dialog box. The **Model Parameter Configuration** dialog box does not support the configuration of ASAP2 parameters. You can, however, use the **Model Parameter Configuration** dialog box to configure other parameters in your model.

6 In the Real-Time Workshop pane of the **Simulation Parameters** dialog, select `ERT code generation options(2)` from the **Category** menu. Then select the **Generate ASAP2 file** option.

7 Click **Apply**.

8 Click **Build** (or **Generate code**).

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

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, 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.

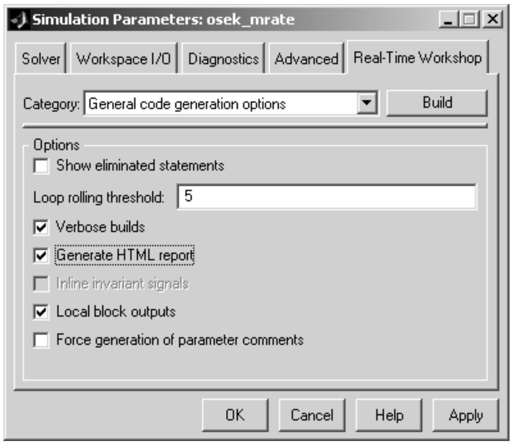
To generate a code generation report and view the profiling report:

1. Select the Real-Time Workshop tab of the **Simulation Parameters** dialog box. Select `Target configuration` from the **Category** menu. Make sure that the **Generate code only** option is not selected.

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

2 Select `General code generation options` from the **Category** menu.

3 Select **Generate HTML report**, as shown in this picture.


Simulation Parameters: osek_mrate

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Category: General code generation options

Options
- [ ] Show eliminated statements
- Loop rolling threshold: 5
- [x] Verbose builds
- [x] Generate HTML report
- [ ] Inline invariant signals
- [x] Local block outputs
- [ ] Force generation of parameter comments

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

5 The Real-Time Workshop writes the code generation report file in the `html` subdirectory.

6 The Real-Time Workshop automatically opens the MATLAB Help browser and displays the code generation report.

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

Alternatively, you can view the code generation report in your Web browser.

Model Execution

This section includes the following topic:

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, multi-rate/singletasking, and multi-rate/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

Recall that in multi-rate 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 multi-rate model that executes in `SingleTasking` mode, the rate scheduler function is named `rate_scheduler`. In a multi-rate 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 multi-rate 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 sub-rate tasks gets `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.

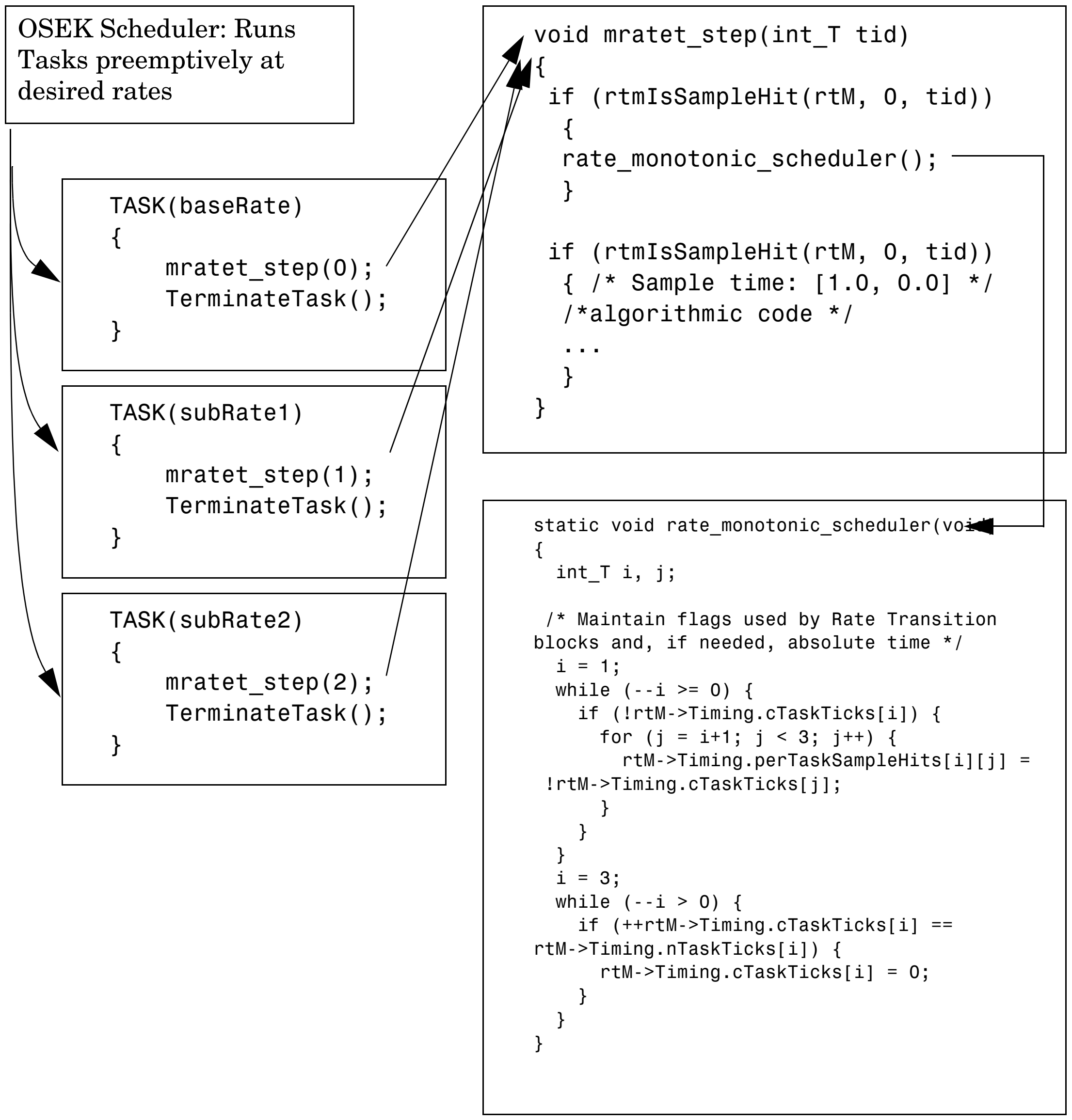
The priority of the base rate (fastest) task is assigned by the user, via the **Base task priority** parameter (see "Code Generation Options" on page 4–5). OSEK/VDX Task priorities decrease from the base rate task to each of the sub-rate tasks, such that the slowest rate has the lowest priority. Tasks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



```
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;
    }
  }
}
```

Model Execution in OSEK/VDX Target Environment

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.

It is expected that the application will 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.

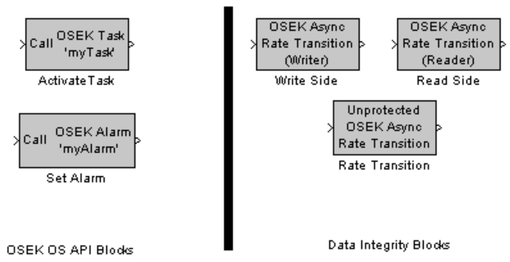
The unused `main()` function for OSEKWorks consumes 36 bytes in the `.text` section of the code.

Block Reference

This section contains the following topics:

The Embedded Target for OSEK/VDX Block Library

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 all 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 6-12). This block is convenient for providing visual feedback from a generated program. It does not provide any 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 these blocks are used for code generation, certain changes in code execution behavior are introduced. 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, the Embedded Target for OSEK/VDX provides a set of blocks (e.g., the OSEK Async Rate Transition blocks) that ensure data integrity. Simulink enforces the use of these blocks. This ensures data integrity, but 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 Async Rate Transition block.

The following sections provide complete information on each block in the Embedded Target for OSEK/VDX block library, in a structured format. Refer to these pages when you need details about a specific block. Click **Help** on the **Block Parameters** dialog for the block, or access the block reference page through Help.

Using Block Reference Pages

Block reference pages are listed in alphabetical order by the block name. Each entry contains the following information:

- **Purpose** — describes why you use the block or function.
- **Library** — identifies the block library where you find the block.

- **Description** — describes what the block does.
- **Dialog Box** — shows the **Block Parameters** dialog and describes the parameters and options contained in the dialog. Each parameter or option appears with the appropriate choices and effects.
- **Examples** — optional section that provides demonstration models to highlight block features.

In addition, block reference pages provide pictures of the Simulink model icon for the blocks.

Blocks Organized by Category

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

Embedded Target for OSEK/VDX Library Blocks

OSEK OS 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

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
Unprotected OSEK Async Rate Transition	Bypass data integrity protection for a signal

Example Driver Block

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

Alphabetical List of Blocks

Activate Task

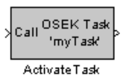
Purpose

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 **Base task priority** specified in the `OSEKWorks code generation options` or `ProOSEK code generation options` parameters

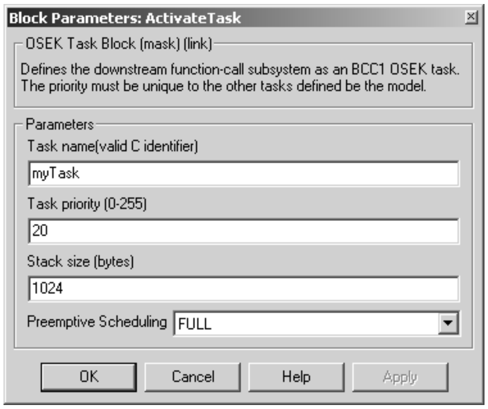
If the downstream Task priority is set to be lower than the caller's, the `ActivateTask` call puts the Task in the ready queue of the OSEK scheduler on the target environment. This differs from simulation, where the Function-Call Subsystem executes immediately upon being triggered and runs to completion 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 non-preemptive, as allowed by OSEK/VDX. This parameter should also be considered 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 Set Alarm 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 6-13 for additional details.

Dialog Box



Defines the downstream function-call subsystem as an BCC1 OSEK task.
The priority must be unique to the other tasks defined be the model.

Parameters

Task name(valid C identifier): myTask

Task priority (0-255): 20

Stack size (bytes): 1024

Preemptive Scheduling: FULL

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.
- `NON`: The Task cannot be preempted by any other Task.

OSEK Async Rate Transition (Reader)

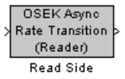
Purpose

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

Library

Embedded Target for OSEK/VDX

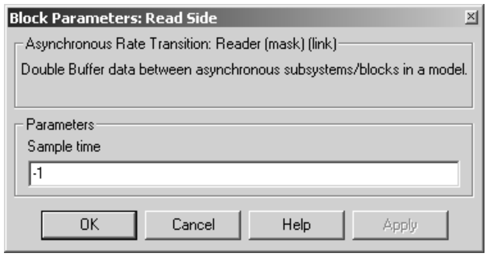
Description



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

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



Block Parameters: Read Side

Asynchronous Rate Transition: Reader (mask) (link)

Double Buffer data between asynchronous subsystems/blocks in a model.

Parameters

Sample time: -1

Sample time

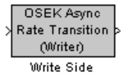
Sample time of the block.

OSEK Async Rate Transition (Writer)

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

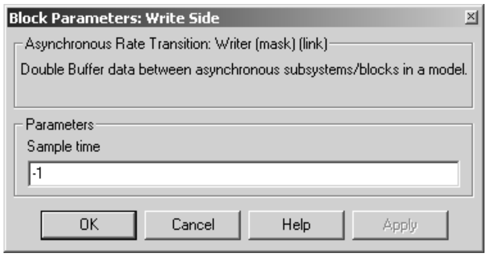
Library Embedded Target for OSEK/VDX

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



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



Block Parameters: Write Side

Asynchronous Rate Transition: Writer (mask) (link)

Double Buffer data between asynchronous subsystems/blocks in a model.

Parameters

Sample time

-1

Sample time

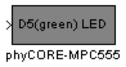
Sample time of the block.

PhyCORE-MPC555 LED

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.

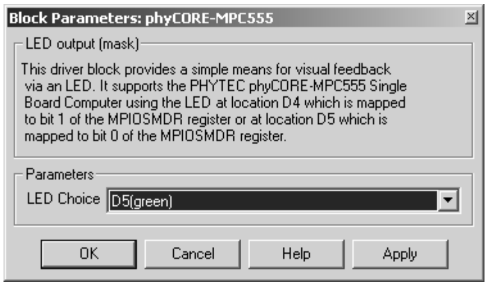
The PhyCORE-MPC555 LED driver block uses the following bits of the MPIOSMDR register:

- 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 3-28.

Dialog Box



Block Parameters: phyCORE-MPC555

LED output (mask)

This driver block provides a simple means for visual feedback via an LED. It supports the PHYTEC phyCORE-MPC555 Single Board Computer using the LED at location D4 which is mapped to bit 1 of the MPIOSMDR register or at location D5 which is mapped to bit 0 of the MPIOSMDR register.

Parameters

LED Choice: D5(green)

LED choice

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

Purpose

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

Library

Embedded Target for OSEK/VDX

Description

Call
OSEK Alarm
'myAlarm'

Set Alarm

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 this option is selected, the `SetAbsAlarm` or `SetRelAlarm` call is only generated once, in the model initialization function. In this mode:

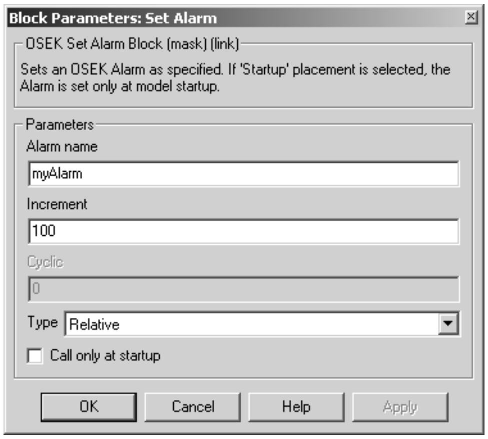
- No signal should be connected to the `CALL` input of the Set Alarm block.
- The **Cyclic** parameter can be set to nonzero values.

When the **Call only at startup** option is deselected, 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.
- The **Cyclic** parameter can not be set to nonzero values. this disallows multiple activations of the downstream Task.

Note that, in the current release, the Set Alarm block immediately activates the downstream Task during simulation.

Dialog Box


Block Parameters: Set Alarm

OSEK Set Alarm Block (mask) (link)

Sets an OSEK Alarm as specified. If 'Startup' placement is selected, the Alarm is set only at model startup.

Parameters

Alarm name: myAlarm

Increment: 100

Cyclic: 0

Type: Relative

☐ Call only at startup

Alarm name

Name of the OSEK Alarm in the generated code.

Increment

The **Increment** parameter value is used directly as an argument to the generated `SetAbsAlarm` or `SetRelAlarm` call. Where `n` is the **Increment** value:

- For relative alarms, the Task assigned to the Alarm is activated after `n` ticks of the OSEK system counter have elapsed.
- For absolute alarms, the Task assigned to the Alarm is activated when the OSEK system counter reaches a value of `n`.

See the OSEK/VDX documentation for further information.

Cyclic

The **Cyclic** parameter value is used directly as an argument to the generated `SetAbsAlarm` or `SetRelAlarm` call.

If **Cyclic** is set to a nonzero value `c`, the Alarm will repeatedly activate the assigned Task every `c` ticks.

If **Call only at startup** is deselected, **Cyclic** is disabled (grayed out).

Type

This menu selects whether the alarm is relative or absolute.

Call only at startup

When this option is selected, the Alarm is activated only during OSEK startup.

Unprotected OSEK Async Rate Transition

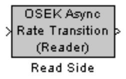
Purpose

Bypass data integrity protection for a signal

Library

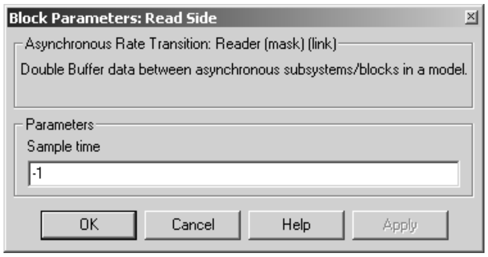
Embedded Target for OSEK/VDX

Description



The Unprotected OSEK Async Rate Transition block can be used when no data integrity protection is desired for signals passing to and from a Function-Call Subsystem that is defined as an OSEK Task. The block informs Simulink to allow the data connection.

Dialog Box


Block Parameters: Read Side

Asynchronous Rate Transition: Reader (mask) (link)

Double Buffer data between asynchronous subsystems/blocks in a model.

Parameters

Sample time

-1

Sample time

Sample time of the block.

Index