

Robot Car: An Embedded Target Example

Modeling
|

Simulation
|

Implementation
|

How to Contact The MathWorks:



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



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



| | |
|--------------|-------|
| 508-647-7000 | Phone |
|--------------|-------|



| | |
|--------------|-----|
| 508-647-7001 | Fax |
|--------------|-----|



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

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

Robot Car: An Embedded Target Example

© COPYRIGHT 2002 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

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

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and Target Language Compiler is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: July 2002 Online only Release 13

Robot Car: an Embedded Target Example

| | |
|---|------|
| Overview | 1-2 |
| Prerequisites | 1-2 |
| Hardware and Software Requirements | 1-3 |
| | |
| Operating the Target | 1-6 |
| Target Directories | 1-6 |
| Verifying Installation | 1-7 |
| A Test Drive with the Demonstration Model | 1-11 |
| | |
| The Demonstration Model | 1-15 |
| Design of the robot3 Model | 1-15 |
| The robot3 Model in Simulation | 1-18 |
| The Generated Program | 1-19 |
| | |
| Components of the ECRobot Target | 1-21 |
| Main Program | 1-21 |
| Device Drivers | 1-23 |
| System Target File | 1-27 |
| Template Makefile | 1-31 |
| | |
| Exercise: Creating a Device Driver | 1-35 |
| LCD Driver Specification | 1-35 |
| Suggestions for Implementing Your LCD Driver | 1-38 |
| Solution | 1-40 |
| | |
| Further Reading | 1-42 |

Overview

We developed the ECRobot (Embedded Coder Robot) target as a simple example of a custom target based on the Real-Time Workshop® Embedded Coder.

Programs generated by the ECRobot target run on the Robot Command System (RCX™)¹ module of the LEGO® MINDSTORMS™ Robotics Invention System 2.0™. This platform affords an inexpensive and simple way to study concepts and techniques essential to developing a custom embedded target, and to develop, run and observe generated programs.

This document, and the files included with the target, illustrate typical approaches to problems encountered in custom target development, including:

- Interfacing an Embedded Coder generated program to an external real-time operating system (RTOS) or kernel.
- Implementing device drivers, via wrapper S-functions, for use in simulation and inlined code generation.
- Customizing a system target file by adding code generation options and adding the target to the System Target File Browser.
- Customizing a template makefile to use a target specific cross compiler and download generated code to the target hardware.

Prerequisites

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

If you have not already done so, you should become familiar with Real-Time Workshop code generation procedures by reading “Getting Started: Basic Concepts and Tutorials” and doing the examples in the “Quick Start Tutorials” in Chapter 1 of the *Real-Time Workshop User’s Guide*.

A minimal conceptual understanding of masked S-functions, C MEX-file S-functions, and wrapper S-functions is also helpful in understanding how device drivers are implemented for the ECRobot target.

1. MINDSTORMS, RCX, Robotics Invention System 2.0, and LEGO are registered trademarks of The LEGO Group.

These topics are covered in:

- “Writing S-Functions in C” and “Writing S-Functions for Real-Time Workshop” in the *Writing S-Functions* manual.
- “Using Masks to Customize Blocks” in *Using Simulink*.
- “Getting Started” and “Inlining S-Functions” in the *Target Language Compiler Reference Manual*.

See also “Further Reading” on page 1-43.

Hardware and Software Requirements

Host Platform

The ECRobot target uses a Windows-based cross-compiler and other utilities; it is therefore hosted on Windows 2000 or Windows XP. A UNIX configuration is not planned.

Your PC must be equipped with a USB port to download programs to the MINDSTORMS RCX module via the IR Transmitter.

Hardware Requirements

Programs generated by the ECRobot target run on the Robot Command System (RCX) module of the MINDSTORMS Robotics Invention System (version 2.0 or higher) manufactured by LEGO Systems, Inc.

For information on the MINDSTORMS Robotics Invention System, see the LEGO MINDSTORMS Web site at <http://mindstorms.lego.com>.

The demonstration model (robot3.mdl) discussed in this document requires construction of the “Acrobot” car project described in the MINDSTORMS Robotics Invention System documentation. Note that the basic “Acrobot” car incorporates a single front-mounted touch sensor. This is sufficient to test and exercise the device drivers discussed in this document. The robot3 demonstration model actually contains two touch sensor device driver blocks; you have the option to add an additional sensor device to the car.

Note This document assumes that you are familiar with the process of downloading programs to the MINDSTORMS RCX module via the IR Transmitter, as described in the MINDSTORMS Robotics Invention System documentation. Before proceeding you should verify that your IR Transmitter is properly connected to the USB port on the host computer, and that the host and the RCX can communicate properly.

Software Requirements

This section summarizes the software required to build and run programs generated by the ECRobot example target. Note that in addition to components provided by The MathWorks, the ECRobot target requires a number of third-party utilities.

ECRobot Target Files. If you have licensed and installed Real-Time Workshop Embedded Coder 3.0 the ECRobot target files are installed automatically in `matlabroot/toolbox/rtw/targets/ECRobot`.

Otherwise, you can also find a copy on the MathWorks web site:

<http://www.mathworks.com/products/rtwembedded/ECRobot.zip>

Unzip this file into `matlabroot/toolbox/rtw/targets/ECRobot`.

gmake Utility. The gmake utility (shipped with the Real-Time Workshop) is required to build programs generated by the ECRobot example target.

Third-party Utilities. The ECRobot target requires a number of third-party utilities. These utilities are not included with Real-Time Workshop Embedded Coder, but are available free of charge on various web sites. The required third-party utilities are:

- legOS operating system kernel: legOS is a possible alternative to the standard MINDSTORMS kernel.

Note that legOS is not a product of The LEGO Group and is not endorsed by the LEGO Group.

legOS provides a C API that supports development of MINDSTORMS applications in C. The *Introduction to the legOS Kernel* guide (see “Further

Reading” on page 1–43) describes the RCX hardware and the architecture of legOS, including the legOS API.

LegOS is an open source project that is available free of charge from the following web site: <http://legOS.sourceforge.net>.

See “Operating the Target” on page 1-6 for information on downloading the legOS kernel to the RCX module.

- GCC cross-compiler for Hitachi microcontroller: ECRobot programs execute on a Hitachi H8/3292 microcontroller in the RCX. Makefiles generated from the ECRobot target template makefile invoke a GCC cross-compiler for this microcontroller. See “Operating the Target” on page 1-6 for information on installing the compiler.
- Downloading utilities: both the legOS kernel and the programs generated by the ECRobot target are downloaded to the RCX module via an infrared (IR) device that interfaces to a serial port on the PC. The download utilities provided are:
 - `firmed13`: downloads legOS kernel to the RCX module.
 - `d11`: (not to be confused with a `.d11` file on Windows) downloads user programs to the RCX module. You can use this utility to download programs manually, or use the **Download program to robot** feature to automatically download programs after building them.

Please see the file

`matlabroot/toolbox/rtw/targets/ECRobot/documentation/readme.html` for information on the third-party utilities and how to obtain and install them. We supply an installer script that unzips and set up these utilities, after you have downloaded them to your PC.

Operating the Target

This section is in three parts:

- “Target Directories” describes the ECRobot target directory structure.
- “Verifying Installation” describes how to verify that the correct ECRobot target files and directories are installed, and how to download the legOS kernel and a small test program to the RCX module.
- “A Test Drive with the Demonstration Model” describes how to generate code from a demonstration model, download it to the RCX module, and run it on the target hardware.

Note Before working with the ECRobot target, please read *matlabroot/toolbox/rtw/targets/ECRobot/documentation/readme.html* and follow the instructions for obtaining and installing the required third-party utilities and setting up the MATLAB path.

Target Directories

The root directory for the ECRobot target is *matlabroot/toolbox/rtw/targets/ECRobot*. Verify that the following target directories are present:

- *ECRobot/common*
This folder contains setup and installation scripts.
- *ECRobot/demo_models*
This folder contains the robot3 demonstration model.
- *ECRobot/documentation*
This folder contains documentation related to the ECRobot target, including this document.
- *ECRobot/drivers*
This folder contains device driver source code and related files.

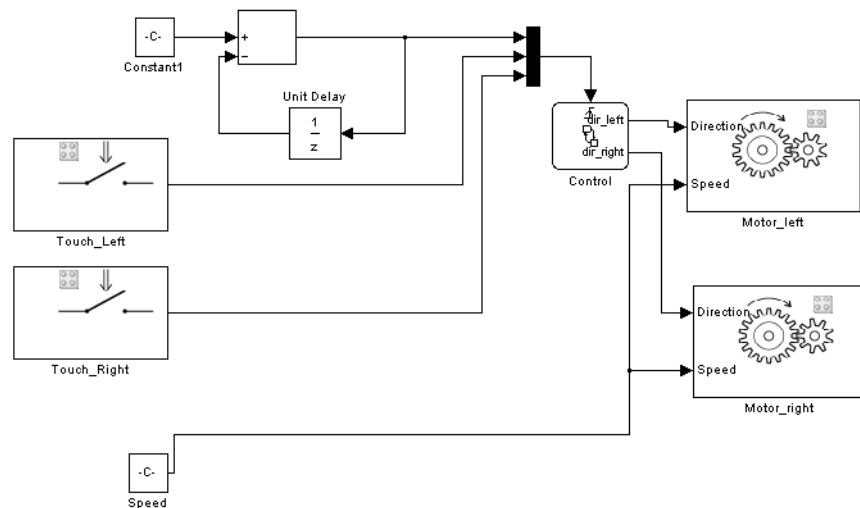
- ECRobot/ECRobot
This folder contains the ECRobot system target file, template makefile, main program module, and other target files.
- ECRobot/ECRobotLib
This folder contains a Simulink library of device driver blocks used in the demonstration model.
- ECRobot/solutions
This folder contains a solution to the LCD driver exercise discussed in “Exercise: Creating a Device Driver” on page 1-36.

Verifying Installation

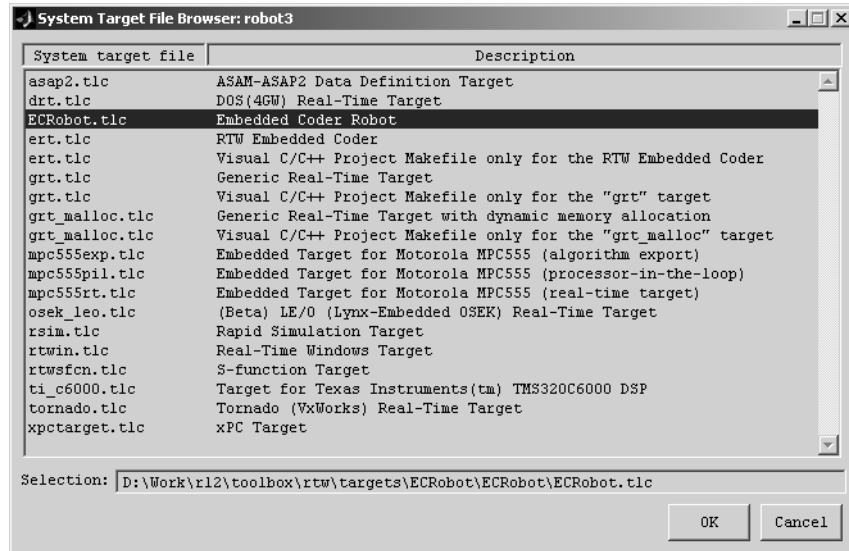
Verify that the ECRobot target is visible in the Real-Time Workshop as follows:

- 1 Open the ECRobot target demonstration model, `robot3.mdl`, by typing
`robot3`

at the MATLAB prompt. The model is shown in the following picture.

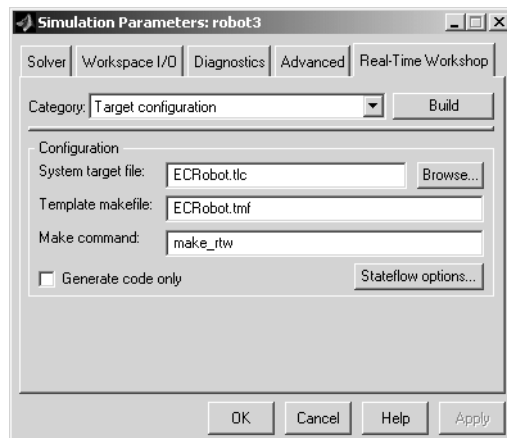


- 2 Open the **Simulation Parameters** dialog, select the Real-Time Workshop page, and click on the **Browse** button. **Embedded Coder Robot** should appear in the System Target File Browser, and should be selected, as shown in the following picture.



If **Embedded Coder Robot** is not the selected target, select it.

- 3 Click **OK** to close the System Target File Browser. Verify that the Real-Time Workshop page displays the target configuration shown in this picture.



- 4 Close the robot3 model.

Programs generated by the ECRobot target require that the legOS kernel be installed on the RCX module, replacing the standard RCX firmware.

Note Note that you need only install the legOS kernel the *first* time you use the ECRobot target. The kernel will then reside in RCX memory (unless you deliberately delete it or replace it with other firmware, or unless battery power on the RCX fails).

In the following discussion, *legos_root* refers to the directory where legOS is installed.

To install and test the legOS kernel:

- 1 Make sure that IR Transmitter is connected to the USB port of your computer and that the IR Transmitter and RCX module are properly aligned.

- 2 Power up the RCX module if necessary, and press the PRGRM and ON-OFF buttons simultaneously. This initializes the RAM on the RCX module; the initialization includes deleting any kernel that may already be present.
- 3 To download the legOS kernel to the RCX module, type the following command at the MATLAB prompt.

```
!legos_root\util\firmd13 legos_root\boot\legOS.srec
```

The legOS firmd13 utility begins downloading code to the RCX. If the transmission is working properly, you should see progress messages similar to the following in the MATLAB window.

```
Transferring "Fast Download Image" to RCX...
```

```
0%  
100%
```

```
Transferring "C:\ERobotUtils\legos\boot\legOS.srec" to RCX...
```

```
0%  
2%  
...  
100%
```

As the download proceeds, the LCD panel on the RCX displays rapidly changing numbers. A successful download completes with the 100% message.

If error messages are displayed, retry the above command. You may need to use the “slow download” option (-s) of the firmd13 utility, as follows:

```
!legos_root\util\firmd13 -s legos_root\boot\legOS.srec
```

Note The IR transmission process is sensitive to ambient light, misalignment of the RCX and IR tower, weak batteries, and other problems. Any of these factors may cause errors on downloading firmware and/or user programs. If errors persist, you may need to turn off lights near the IR tower

and RCX, realign the IR tower and RCX, replace the batteries in the IR tower and/or RCX, or use the “slow” download option (see above). Note that the LEGO MINDSTORMS Web site has extensive troubleshooting information; link to <http://mindstorms.lego.com/> and visit the Tech Support section.

- 4 To verify that legOS is properly installed on the RCX, download the helloworld test program by typing the following command:

```
!legos_root\util\dll legos_root\demo\helloworld.lx
```

The legOS dll utility downloads a small program to the RCX. The utility does not display any message upon completion. When MATLAB returns to the prompt, press the RUN button on the RCX. The string “Hello World” is displayed on the LCD panel.

The ECRobot target is now installed and ready for use with legOS. In the next section, we will build, download, and run a generated program on the target hardware.

A Test Drive with the Demonstration Model

In this section you will:

- Generate a program from the robot3 demonstration model, manually download it to the RCX, and run it on the RCX.
- Modify a parameter of the model, rebuild it, and automatically download it using the **Download program to robot** option of the ECRobot target.

To generate, download, and run a program:

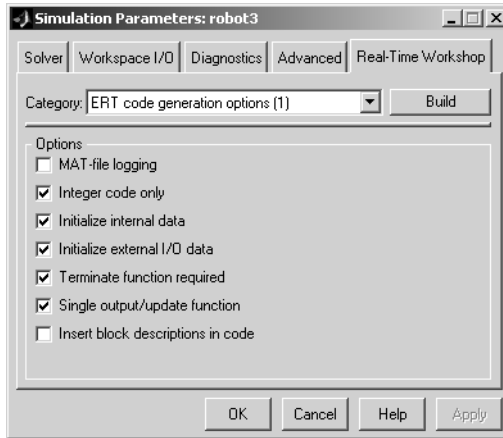
- 1 Create a directory called ECRobotTest. Make it your working directory. Type

```
mkdir ECRobotTest
cd ECRobotTest
```

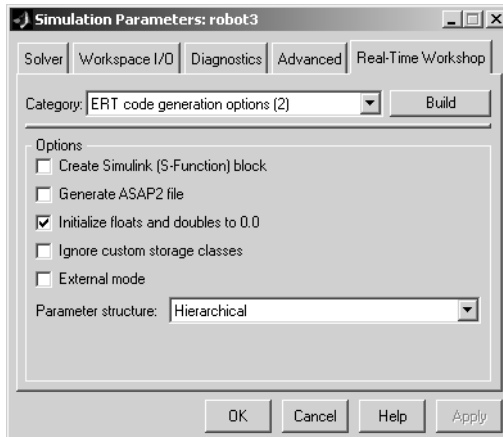
at the MATLAB prompt.

- 2 Copy the ECRobot target demonstration model, robot3.mdl, to your working directory. Open the model.

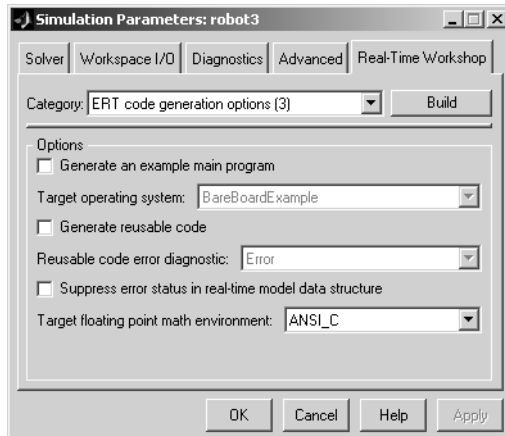
- 3 The ECRobot target inherits code generation options from the ERT target. It is necessary to make sure that options not relevant to the ECRobot target are turned off. Open the **Simulation Parameters** dialog and select **ERT code generation options (1)** from the **Category** menu. Make sure that the options are set as shown in this picture.



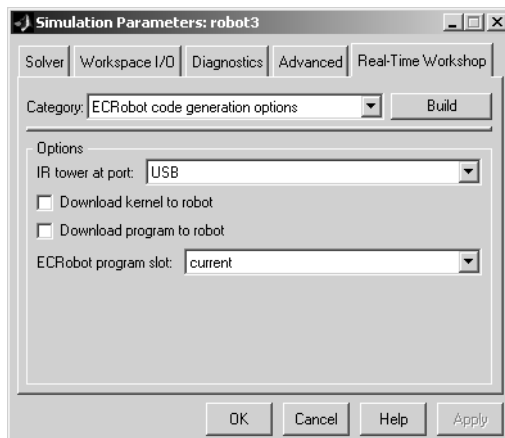
- 4 Select **ERT code generation options (2)** from the **Category** menu. Make sure that the options are set as shown in this picture.



- 5 Select **ERT code generation options (3)** from the **Category** menu. Make sure that the options are set as shown in this picture.



- 6 Select **ECRobot code generation options** and make sure that the **Download program to robot** and **Download kernel to robot** options are cleared, as in this picture. In this part of the exercise we will manually download the generated program.



- 7 Click **Apply** if necessary. Then click the **Build** button.

- 8 The Real-Time Workshop generates code and compiles it using the GCC cross-compiler for the Hitachi microcontroller, `h8300-hms-gcc`. When the build completes, the following message is displayed:

```
### Successful completion of Real-Time Workshop build procedure for model:  
robot3
```

- 9 Programs built by the ECRobot target have the filename extension `.lx`. The generated `robot3.lx` program is written to your working directory.

Like other targets, the ECRobot target creates a build directory (in this case `robot3_ECRobot_rtw`) in the working directory.

- 10 To manually download `robot3.lx` to the RCX module, use the `legOS dll` utility. Type the following command.

```
!legos_root\util\dll robot3.lx
```

(`legos_root` refers to the directory where `legOS` is installed.)

Press the **RUN** button on the RCX. The left and right motors on the “Acrobot” car should begin running. When the touch sensor is activated, the wheels should reverse direction for a brief period of time.

Now change the motor speed of the model, rebuild `robot3.lx`, and automatically download it to the RCX. To do this:

- 1 The motor speed is set by a Constant block, **Speed**. Open the **Block Parameters** dialog for the Speed block and change the Constant value field to `uint8(25)`. Click **Apply**. Then click **OK**.
- 2 Open the **Simulation Parameters** dialog and select **ECrobot code generation options** from the **Category** menu. Select the **Download program to robot** option.

The **Download kernel to robot** option should be off. This option is a convenience that lets you download the LEGOS kernel to the RCX during the build process. This is not usually necessary.

- 3 The **IR tower at port** option selects a port for communication with the IR tower. Make sure this option is set to USB.
- 4 The **ECRobot program slot** option selects which of the program slots (0-4) on the RCX is to receive the downloaded code. Make sure this option is set

to current. The program will be downloaded to the slot that is currently selected on the RCX front panel.

- 5** Click **Apply**. Then click the **Build** button.

This time, after completing code generation and compilation, the build process invokes the d11 utility. The robot3.1x code is downloaded without manual intervention.

- 6** When downloading completes, run the program on the RCX and observe that the motor speed is considerably slower.

The Demonstration Model

This section discusses the demonstration model, robot3.mdl (see Figure 1-1). The discussion is in three parts:

- “Design of the robot3 Model” is an overview of the model’s inputs, outputs, and control logic.
- “The robot3 Model in Simulation” discusses how the model functions in Simulink.
- “The Generated Program” discusses the generated code, and how it interacts with the target environment.

Design of the robot3 Model

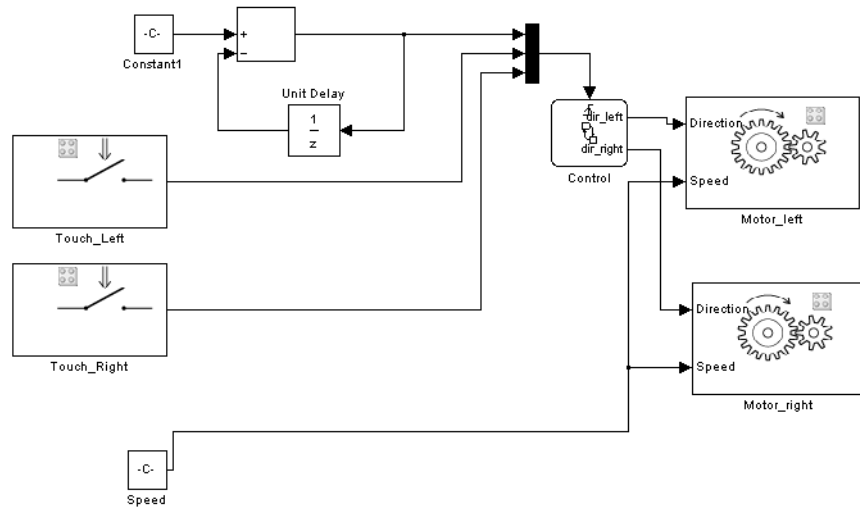


Figure 1-1: robot3 Demonstration Model

Software Timer

A feedback loop, consisting of the Constant1, Sum, and Unit Delay blocks, generates a “square wave” that toggles from 1 to 0 on each model step. This signal functions as a software timer for the Control block, which counts rising edges as events (see “Control Logic” on page 1-18).

Note that in this example, the software timer is not a true real-time clock. Its function is better understood as an event generator for the Control block.

This method of event timing was chosen due to limitations of the LegOS kernel. In a true real-time system, model functions would be timed by a hardware timer interrupt service routine (ISR). This functionality is fully supported by the Real-Time Workshop Embedded Coder. See the *Real-Time Workshop Embedded Coder User's Guide* for further information.

Inputs

The inputs to the model come from two device driver blocks, Touch_Right and Touch_Left. Each of these blocks calls legOS to obtain an 8-bit value from ADCs attached to sensors on the target hardware. The expected values are 1 (sensor activated) or 0 (sensor not activated).

The Touch_Right and Touch_Left blocks read sensor values on every time step of the model. These values function as events within the Control block. The sensor values are multiplexed with the software timer signal described above.

See “Device Drivers” on page 1-24 for a detailed description of the touch sensor input device driver.

Outputs

The outputs from the model come from two device driver blocks, Motor_right and Motor_left. Each of these blocks calls legOS to set the direction and speed of two motors on the target hardware.

The speed value is an unsigned 8-bit constant. The unsigned 8-bit directional value is either 1 (forward) or 2 (backward).

The Motor_right and Motor_left blocks output speed and direction values on each time step of the model, regardless of whether these values have changed.

See “Device Drivers” on page 1-24 for a detailed description of the motor device driver.

Control Logic

The logical engine of the model is the Control block, a Stateflow chart. The chart is shown in Figure 1-2.

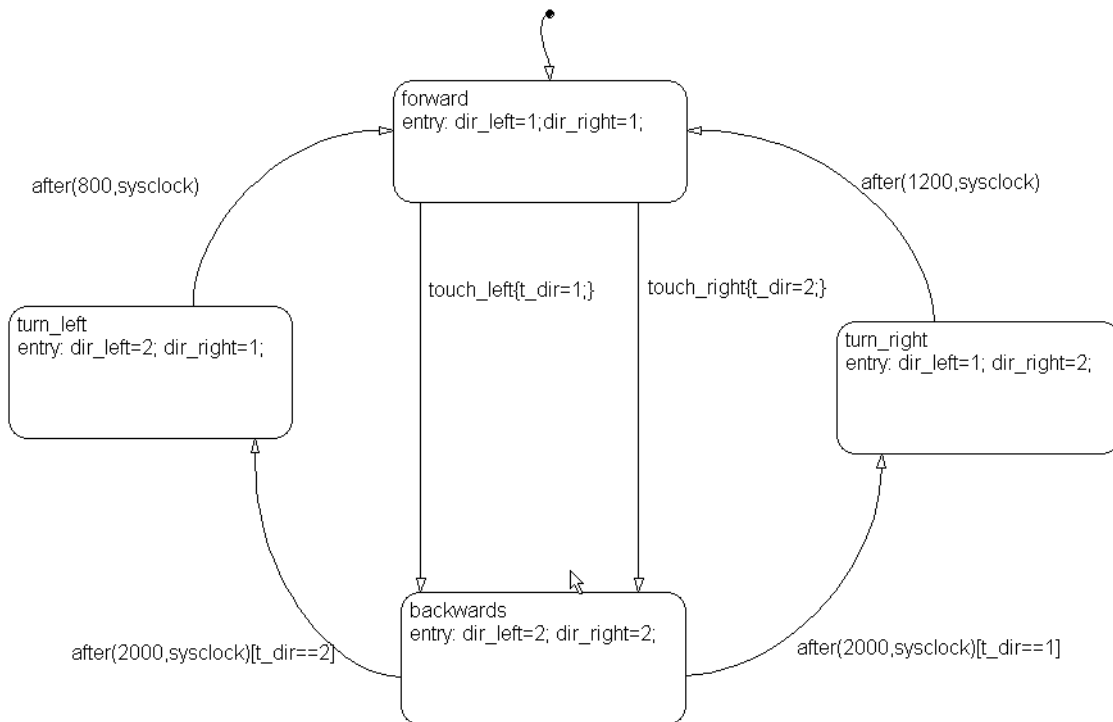


Figure 1-2: Control Chart

The purpose of the Control block is to determine directional control values to be transmitted to the Motor_right and Motor_left device driver blocks. The Control block defines 4 states: forward, backwards, turn_left, and turn_right. When the Control block transitions to a given state, it sets its dir_left and dir_right outputs to an appropriate pair of control values.

The Control block transitions from one state to another in response to the following trigger events:

- `touch_left`: occurs on a rising edge of the signal from the `Touch_Left` sensor (sensor is activated). In response to this event, if the current state is forward, the Control block goes to the backwards state, and the value of the local variable `t_dir` is set to 1.
- `touch_right`: occurs on a rising edge of the signal from the `Touch_Right` sensor (sensor is activated). In response to this event, if the current state is forward, the Control block goes to the backwards state, and the value of the local variable `t_dir` is set to 2.
- `sysclock`: occurs on the rising edge of the software timer generated by the feedback loop at the root level.

The Control block counts `sysclock` events, and transitions from one state to another after a certain (arbitrary) number of `sysclock` events. For example, if the model remains in the `turn_right` state for 1200 `sysclock` events, the Control block goes to the forward state. This logic ensures that the car does not become stuck travelling in a certain direction for too long.

Note that transitions from backwards to `turn_left` or `turn_right` are conditional upon the variable `t_dir` as well as upon elapsed clock counts.

The robot3 Model in Simulation

In simulation, the model performs little computation, since its I/O driver blocks function simply as placeholders, and do not interact with any physical hardware. Each driver block performs the following during simulation:

- Validates its block parameters.
- Types a message to the MATLAB window.
- Computes a zero output value, or no output at all.

For example, the Touch Sensor blocks output a zero value, as shown in its `mdlOutputs` function (see “Touch Sensor Driver” on page 1-26).

The Generated Program

Figure 1-3 shows the structure of an ECRobot target generated program when loaded and running on the RCX.

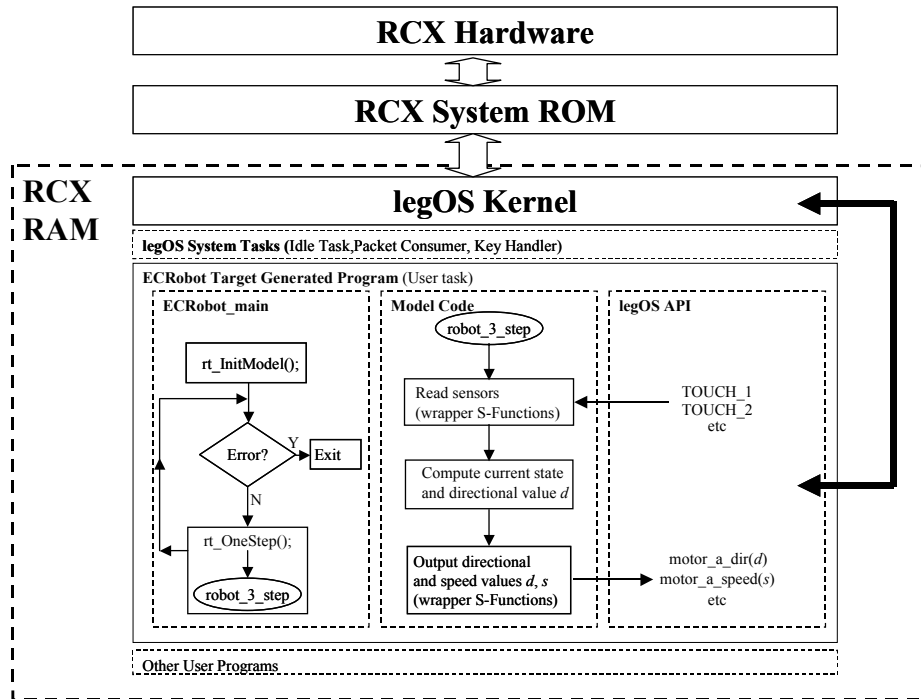


Figure 1-3: Generated Program (robot3) Running in RCX Environment

The RCX RAM is located at 0x8000 to 0xFFFF within the RCX address space. The legOS kernel resides in the lowest part of RAM. The remaining RAM is allocated to:

- legOS system tasks. When legOS is booted, it starts the systems tasks, which run as long as legOS is running. An example of a system task is the *packet consumer*, which handles IR communication with a host computer and downloads and locates user programs in memory.
- Reserved locations for memory-mapped registers, I/O ports, and interrupt vectors.

- User programs. User programs are relocatable code modules, whose address is determined when they are loaded by the packet handler. Figure 1-3 shows the ECRobot target example program, `robot3`, residing in RAM as the first user program, above the legOS system tasks.

The generated `robot3` program is a single-rate, single-tasking program that runs as a task under legOS. The components of the program, laid out schematically in Figure 1-3, are:

- Main program (`ECRobot_main`): The main program performs a simple initialization routine and then runs in a tight loop, continually calling the generated model step function until terminated. See “Main Program” on page 1-22 for a detailed description.
- Model code: the generated `robot3_step` function (see `robot3.c`) implements the top-level logic of the model. Calls to the legOS API perform input of sensor values and output of motor control values. These calls are generated inline by wrapper S-functions (see “Device Drivers” on page 1-24).
- legOS API: legOS provides a library of low-level functions that let user programs access and control motors, sensors, and other hardware. These functions are imported into the generated program via `#include` directives in `ECRobot_main.c` and in the generated code.

Components of the ECRobot Target

The ECRobot target was implemented largely by modifying code and control files provided with the Real-Time Workshop Embedded Coder. These files fall into two categories:

- *Run-time interface.* The run-time interface consists of code that supervises and supports execution of generated model code. In the ECRobot target, the run-time interface includes the main program, inlined device drivers, and header files that provide glue to the legOS kernel.
- *Control files.* The code generation and build process are controlled by a customized system target file and template makefile. The custom control files run a target-specific cross-compiler and (optionally) download the generated program to the target system.

The following sections discuss how Embedded Coder components were modified to create the ECRobot target. The source code files discussed are located in:

- ECRobot\ECRobot: main program, system target file, and template makefile.
- ECRobot\drivers: device drivers.

Main Program

The main program for the ECRobot target is `ECRobot_main.c`. The main program was adapted from `ert_main.c`, the template Embedded Coder main program.

This section discusses the operation of the main program and the target-specific adaptations that were made to `ert_main.c`. Please refer to `ECRobot_main.c` while reading this section. You may also find it helpful to run a file comparison between `ECRobot_main.c` and `matlabroot/rtw/c/ert/ert_main.c`.

Hardware Initialization

Target-specific hardware initialization is performed by the `rt_InitModel` function, which runs when the generated program starts up. `rt_InitModel` calls the generated `model_initialize` function in `model_reg.h`. `model_initialize` contains all hardware initialization calls generated from device driver blocks (See “Device Drivers” on page 1-24).

The following code fragment illustrates *model_initialize* code generated from the device drivers in the robot3 demonstration model.

```

/* S-Function Block: <Root>/Touch_Left */
ds_passive(&SENSOR_1);

/* S-Function Block: <Root>/Touch_Right */
ds_passive(&SENSOR_3);
.
.
.
/* S-Function Block: <Root>/Motor_left */
motor_a_dir(off);
motor_a_speed(0);

/* S-Function Block: <Root>/Motor_right */
motor_c_dir(off);
motor_c_speed(0);

```

main() Function

The `main()` function simply calls the initialization function and then enters a loop in which it continually checks for an error condition and calls `rt_OneStep`. `rt_OneStep` in turn calls the generated *model_step* function. The program normally terminates when the user presses the Run button on the RCX. An abnormal termination (due to an `rt_OneStep` error) is handled either by returning 0, or by calling `rt_TermModel` (if the **Terminate function required** code generation option is selected). `rt_TermModel` returns the final error status to the caller.

Note that programs built with the ECRobot target do not qualify as true real-time programs because model execution is not stepped by a real-time clock. Instead, the generated program executes as a free-running task under `legOS`, and can be preempted by other tasks in the system. The timing of model steps is governed by a software loop, rather than by attaching `rt_OneStep` to a timer interrupt.

Modifications of the Main Program

Modifications of the main program were relatively minor. Modifications include:

- The `main()` function takes no arguments, since the target system has no command line mechanism for passing in arguments.
- Calls to standard I/O functions such as `printf` have been removed, since they are not supported by the target system.
- Some legOS header files are included.

Device Drivers

The ECRobot target includes device driver S-function blocks and C MEX-file components for use in simulation, as well as corresponding TLC files for generation of inlined code. For each driver, the files supplied are:

- `driver.c`: C implementation of the driver S-function for simulation.
- `driver.dll`: MEX-file component.
- `driver_img.tif`: icon for the block.
- `driver.tlc`: TLC implementation of the block for inlined code generation.

The ECRobot target device driver blocks are available in a library, `ECRobot\ECRobotLib\ECRobot.mdl`.

All the device drivers use the *wrapper* S-function technique. That is, the TLC implementations of the blocks generate only:

- Glue (such as `#include` directives) necessary to make external functions (such as legOS library functions) visible.
- The minimal code necessary to call the external functions.

Motor Driver

The Motor block is an output driver. Its function is to set the direction and speed of one of three motors (A, B, or C) on the target hardware. The direction and speed values are obtained from the block input signals and passed directly to legOS.

Please refer to `lmotor.c` and `lmotor.tlc` when reading the discussion below.

Simulation (C MEX-file) Implementation (lmotor.c). The Motor block is a masked S-function that has two parameters:

- Port: Output port (A - C maps to 1 . . 3) associated with the motor. The value is obtained from a popup menu.

- Sample time: Value is obtained from an edit field.

During simulation, the Motor block performs little computation. Most of the standard functions (such as `MdlStart`, `MdlOutputs`, etc.) are stub routines.

The functions of the simulation implementation of the Motor block are:

- Validation of block parameters.
The `mdlInitializeSizes` function checks for the expected number of parameters and call `mdlCheckParameters` to range-check the port number. Note that `mdlInitializeSizes` declares all block parameters to be non-tunable. Declaring the tunability of all parameters is a recommended programming practice.
- The `mdlInitializeConditions` function types a message to the MATLAB window during simulation.
- The `mdlRTW` function writes the port number parameter to the block's record in the `model.rtw` file during code generation.

Note that the `mdlRTW` function is responsible for converting the port number obtained from the popup (a double) to an unsigned 8-bit integer. This is required because the Hitachi H8/3292 microcontroller in the RCX unit supports only integer operations. To enforce this restriction, the ECRobot target sets the **Integer code only (must be checked)** option on by default. (see “Code Generation Options” on page 1-30).

TLC Implementation (`lmotor.tlc`). The TLC implementation of the Motor block is responsible for generating inlined code that passes the speed and direction input values to legOS. There are three TLC functions in `lmotor.c`:

- `BlockInstanceSetup`: TLC provides the `BlockInstanceSetup` function to perform work that needs to be performed for every instance of a given block that is encountered in the `model.rtw` file.

The `BlockInstanceSetup` function for the Motor block checks that each Motor block has a unique port assignment. `BlockInstanceSetup` raises a code generation error if more than one Motor block has the same port number. This could occur, for example, if the model contained more than three Motor blocks.

The `BlockInstanceSetup` function also generates the following directive, which is written to `model_common.h`.

```
#include "dmotor.h"
```

`dmotor.h` is a glue file that defines the public interface to the legOS motor control functions that are required by the Motor driver.

Note that this directive is generated the first time a Motor block occurs in the `model.rtw` file. An alternative method would be to implement a `BlockTypeSetup` function for the Motor block and generate the `#include` directive there.

- **Start:** The `Start` function for the Motor block generates code that initializes the associated motor to an “off” state. This code is written to the `model_initialize` function in `model_reg.h`.

The generated code consists of calls to legOS functions, as in the following example, which initializes motor A.

```
/* S-Function Block: <Root>/Motor_left */
motor_a_dir(off);
motor_a_speed(0);
```

- **Outputs:** The `Outputs` function for the Motor block, like the `Start` function, generates calls to the same legOS functions used in the `Start` function. However, the function arguments are obtained from the block’s `Speed` and `Direction` input signals.

References to the signals are generated via the TLC function `LibBlockInputSignal`. See “TLC Function Library Reference” in the *Target Language Compiler Reference Manual* for information about `LibBlockInputSignal`.

The code is written to the `model_step` function in `model.c`. The following code fragment was generated from the demonstration model, `robot3.mdl`, in which `Speed` is a constant and `Direction` is stored in the global `BlockIO` structure (`robot3_B`).

```
/* S-Function Block: <Root>/Motor_left */
motor_a_dir(robot3_B.dir_left);
motor_a_speed((200U));
```

Touch Sensor Driver

The Touch Sensor block is an input driver. Its function is to read the value of one of three analog/digital converters (ADCs) attached to sensors on the target

hardware. The input value is obtained from legOS and passed directly to the block output without any processing.

Please refer to `touch.c` and `touch.tlc` when reading the discussion below.

Simulation (C MEX-file) Implementation (`touch.c`). The simulation implementation of the Touch Sensor driver is a masked S-function that is almost identical to the Motor block. The Touch Sensor block parameters are:

- Port: Input port (1..3) associated with the sensor. The value is obtained from a pop-up menu.
- Sample time: Value is obtained from an edit field.

Since the Touch Sensor block is an input driver, it must produce a value during simulation. This is the most significant difference between the Touch Sensor block and the Motor block. The `mdlOutputs` function, shown below, generates a zero output value.

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    uint8_T *y = (uint8_T *)ssGetOutputPortSignal(S, 0);
    *y = (uint8_T)0;
}
```

Like the Motor block, the Touch Sensor block validates block inputs and types a message to the MATLAB window during simulation. The Touch Sensor block also implements a `mdlRTW` function that writes an unsigned 8-bit port number to the block's record in the `model.rtw` file during code generation.

TLC Implementation (`touch.tlc`). The TLC implementation of the Touch Sensor block generates inlined code that obtains sensor values from legOS, using macros and function calls provided for that purpose. There are three TLC functions in `touch.c`:

- `BlockInstanceSetup`: The `BlockInstanceSetup` function for the Touch Sensor is almost identical to that of the Motor block. The `BlockInstanceSetup` function checks that each Touch Sensor block has a unique port assignment.

The `BlockInstanceSetup` function also generates the directive to include a header file that defines the public interface to the legOS input sensor functions.

```
#include "dsensor.h"
```

This directive is written to `model_common.h`.

- **Start:** The Start function for the Touch Sensor block generates code that informs legOS that the associated sensor functions in a “passive” mode - that is, the touch sensor does not need to receive power from the RCX in order to be read. This code is written to the `model_initialize` function in `model_reg.h`.

The generated code consists of calls to a legOS function, as in the following example. Note that the sensor is referenced by a macro defined in `dsensor.h`.

```
/* S-Function Block: <Root>/Touch_Left */  
ds_passive(&SENSOR_1);
```

- **Outputs:** The Outputs function for the Touch Sensor block generates a single line of code that addresses the sensor via a legOS macro, and assigns the sensor value to the block’s output signal.

A reference to the output signal is generated via the TLC function `LibBlockOutputSignal`. See “TLC Function Library Reference” in the *Target Language Compiler Reference Manual* for information about `LibBlockOutputSignal`.

The code is written to the `model_step` function in `model.c`. The following code fragment was generated from the demonstration model, `robot3.mdl`, in which the sensor value is stored in the global `BlockIO` structure (`robot3_B`).

```
/* S-Function Block: <Root>/Touch_Left */  
robot3_B.Touch_Left = (uint8_T)TOUCH_1;
```

System Target File

The system target file for the ECRobot target is `ECRobot.tlc`. It is a customized version of `ert.tlc`, the system target file for the Real-Time Workshop Embedded Coder.

Specifying the Target Configuration

The comment lines at the head of the system target file specify the target string to be displayed in the System Target File Browser, as well as the template

makefile and make command to be used in the build process. Since the ECRobot target does not support external mode, no external mode communications MEX-file is specified.

```
%% SYSTLC: Embedded Coder Robot TMF: ECRobot.tmf MAKE: make_rtw \  
%% EXTMODE: no_ext_comm
```

When the ECRobot target is added to the Real-Time Workshop configuration, the System Target File Browser appears as in Figure 1-4 and the Target Configuration section of the Real-Time Workshop page appears as shown in Figure 1-5.

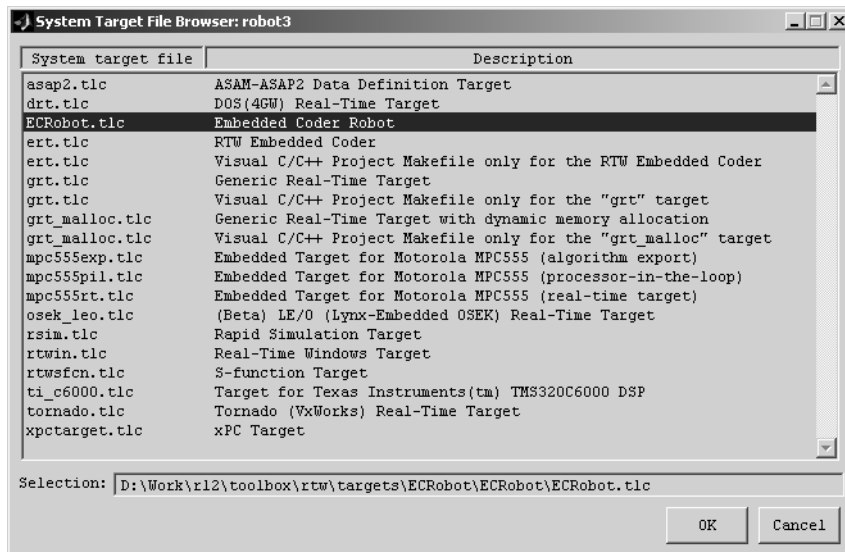


Figure 1-4: ECRobot Target Selected in System Target File Browser

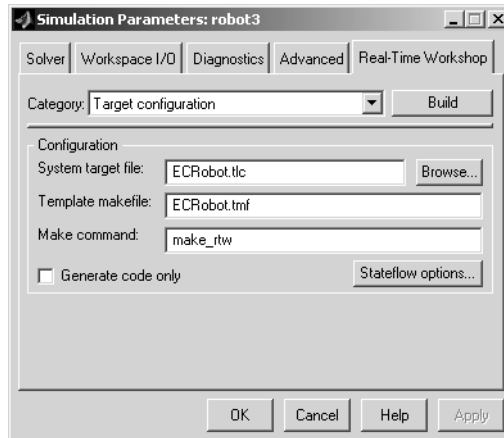


Figure 1-5: ECRobot Target Configuration

Code Generation Options

rtwoptions. The `rtwoptions` array defines variables and associated user interface elements to be displayed in the Real-Time Workshop page. The elements of the `rtwoptions` array specify the code generation options displayed in the **ECrobot code generation options** category (see Figure 1-6).

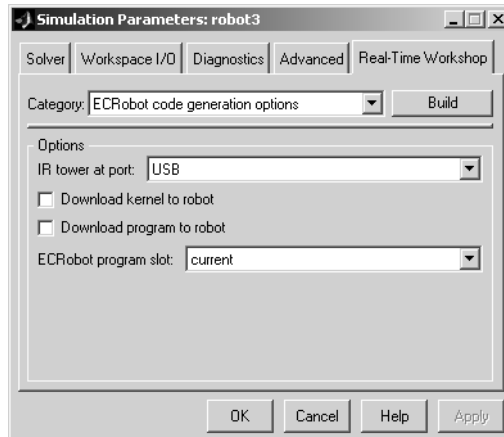


Figure 1-6: ECRobot code generation options

These include the **Download program to robot** option. When this option is selected, the generated makefile invokes the `dll` utility to download the generated code to the RCX unit. The details of this option are described in “Template Makefile” on page 1-32. The following code specifies the **Download program to robot** option.

```
rtwoption_index = rtwoption_index + 1;
rtwoptions(rtwoption_index).prompt = 'Download program to robot';
rtwoptions(rtwoption_index).type = 'Checkbox';
rtwoptions(rtwoption_index).default = 'on';
rtwoptions(rtwoption_index).tlcvariable = 'DOWNLOAD_PROGRAM';
rtwoptions(rtwoption_index).makevariable = 'DOWNLOAD_PROGRAM';
rtwoptions(rtwoption_index).tooltip = ['Download the model executable to Robot
(3 tries: fast,slow,debug mode)'];
```

Refer to the `rtwoptions` array to study how the **Download kernel to robot, IR tower at port** and **ECRobot program slot** options are set.

Inheriting ERT options. The ECRobot target inherits the full set of `rtwoptions` code generation options from the ERT system target file and appends the ECRobot-specific `rtwoptions` to the ERT options, as shown in the following code.

```
% Inherit ERT options

file = fullfile(matlabroot, 'rtw', 'c', 'ert', 'ert.tlc');
propsObj = tlc.rtwoptions(file);
props = propsObj.getOptions;

rtwoptions = propsObj.combineCategories(props,rtwoptions);
```

rtwgensettings. The `rtwgensettings` section of the system target file specifies that build directories that store code generated by the ECRobot target will be named `model_ECRobot_rtw`, as follows.

```
rtwgensettings.BuildDirSuffix = '_ECRobot_rtw';
```

Other Code Generation Options. Some of the inherited ERT code generation options are not meaningful in the context of the ECRobot target. Therefore, the target checks several options and displays warning messages if they are set incorrectly. For example, the **Integer code only** option should always be on, since the Hitachi H8/3292 microcontroller in the RCX unit supports only

integer operations. The following code checks the **Integer code only** option, and if it is set incorrectly, turns it on and displays a message.

```
%if "%<PurelyIntegerCode>" == "0"  
    %assign PurelyIntegerCode = 1  
    %assign results = FEVAL("warndlg","ECRobot target support integer code only.  
    The option has been reset to true.")  
%endif
```

See `matlabroot/toolbox/rtw/targets/ECRobot/ECRobot/ecrobot_settings.tlc` for other option checks.

Template Makefile

This discussion assumes that you know how the `make` command works and how the `make` utility processes makefiles. You should also understand makefile build rules. For information on these topics, please refer to the documentation provided with the `make` utility. There are also several good books on the `make` utility.

This discussion also assumes that you have read the “Template Makefiles” section in the “Targeting Real-Time Systems” chapter of the *Real-Time Workshop User’s Guide*, which contains information on the general structure of template makefiles.

Custom target-specific template makefiles are usually developed by modifying one of the template makefiles distributed with the Real-Time Workshop. The template makefile for the ECRobot target is `ECRobot.tmf`. It is a customized version of `ert_unix.tmf`, one of the template makefiles supplied with the Real-Time Workshop Embedded Coder.

The most important customizations are:

- The build process invokes `gmake`, a `make` utility that is installed with the Real-Time Workshop.
- The generated makefile runs the GCC cross-compiler for the Hitachi h8300 microcontroller and specifies command line options for that compiler.
- The generated code file is named `model.lx`. Optionally, the generated code is downloaded to the RCX hardware via the `d11` utility.

Coverage of every detail of `ECRobot.tmf` is beyond the scope of this document. If you want to inspect every change that was made to the original template makefile, we suggest that you use a file differencing utility (such as the GNU

Emacs Compare tools) to view differences between `ECRobot\ECRobot\ECRobot.tmf` and `matlabroot\rtw\c\ert\ert_unix.tmf`.

In the following sections, we will highlight the target-specific features of each section of the template makefile. Each heading below corresponds to a heading in the template makefile.

Macros Read by the Build Procedure

- The `MAKECMD` macro directs the build process to use the `matlabroot\rtw\bin\win32\gmake` utility. Make sure this path is correctly localized for your installation.
- The `HOST` macro specifies that the build platform is the PC.
- The `DOWNLOAD_SUCCESS` macro specifies a string to display when code is downloaded to the RCX module.
- The `SYS_TARGET_FILE` macro specifies `ECRobot.tlc`.

Tokens Expanded by `make_rtw`

- `|>DOWNLOAD<|`: The system target file scans the **Download program to robot** option and expands this token to either 0 or 1. If 1, the generated makefile attempts to download the `model.lx` code to the target system.
- Note that the **Create Simulink (S-Function) block** option is not supported by the ECRobot target. Therefore, the `|>GENERATE_ERT_S_FUNCTION<|` token and all code that depends on it were removed from `ECRobot.tmf`.

Tool Specifications

Make sure the following paths are correctly localized for your installation. These paths specify directories containing utilities and include files required by the make process:

- `ECROBOT_ROOT` specifies the path to the ECRobot target directory.
- `LEGOS_DIR` specifies the path to the legOS directory.
- `CYGNUS_ROOT` specifies the path to the directory where the GCC cross-compiler for the Hitachi microcontroller is installed.

Include Path

- The standard RTW Embedded Coder directory (*matlabroot/rtw/c/ert*) has been removed from the `MATLAB_INCLUDES` path.
- The `INCLUDES` path incorporates include directories within the `ECROBOT_ROOT`, `LEGOS_DIR`, and `CYGNUS_ROOT` paths, enabling the build process to access required include files.

C Flags

- The option `-DNO_FLOATS` has been added to the compiler options in the `CPP_REQ_DEFINES` list. This option ensures that an error will be raised if any non-integer code is encountered during the build.
- The standard RTW function libraries (*rtwlib_int_ert.a* or *rtwlib_ert.a*) are replaced by *librtw_ECRobot.a*. This library is constructed by a Perl script invoked in the Rules section, and linked into the generated program with other object files.

Source Files

Two target-specific source files are specified in the source file list:

- The standard *rt_nofinite.c* has been eliminated.
- *ECRobot_main.c* replaces *ert_main.c*.

The `PROGRAM` macro defines the naming convention (*model.lx*) for the generated code file.

Rules

This section generates the portion of the makefile that builds the Real-Time Workshop library (*librtw_ECRobot.a*) and the executable (*model.lx*) and invokes the `dll` utility to download the executable to the target.

To build *librtw_ECRobot.a*, the makefile invokes a Perl script, *mklib_ECRobot.pl*. The path to the Perl utility in the `MATLAB` directory structure is defined by the following macro.

```
PERL = $(MATLAB_ROOT)/sys/perl/win32/bin/perl
```

Dependencies

For standard targets, the Real-Time Workshop generates a `rtw_proj.tmw` file that specifies include paths and source files that are required to rebuild object files. The ECRobot target generates a dependency file (`.depend`), in the build directory, for this purpose. Rebuilding only occurs when the name of the current Real-Time Workshop project changes.

Miscellaneous Rules to purge, clean and lint

The `purge`, `clean`, and `lint` commands have been eliminated from this section, as the GCC cross-compiler for the Hitachi microcontroller has no lint utility.

Exercise: Creating a Device Driver

In this section, you create a device driver for the ECRobot target, following specifications given below. The driver displays information on the LCD panel of the RCX.

In this exercise, we provide a driver specification and some suggestions, leaving it up to you to produce a solution creatively. It is expected that you may need to refer to other Real-Time Workshop and Simulink documentation (see “Further Reading” on page 1-43).

After creating and testing your driver, you can compare it to solution files provided in ECRobot\solutions. We urge you not to refer to the solution files until you have completed the exercise on your own.

LCD Driver Specification

Overview

The purpose of the LCD driver is to provide a simple diagnostic that monitors the directional input signal to a motor and displays a string representing the directional value.

Figure 1-7 shows a typical application of the driver. Here the LCD driver block receives the `dir_left` signal from the Control block in the robot3 demonstration model.

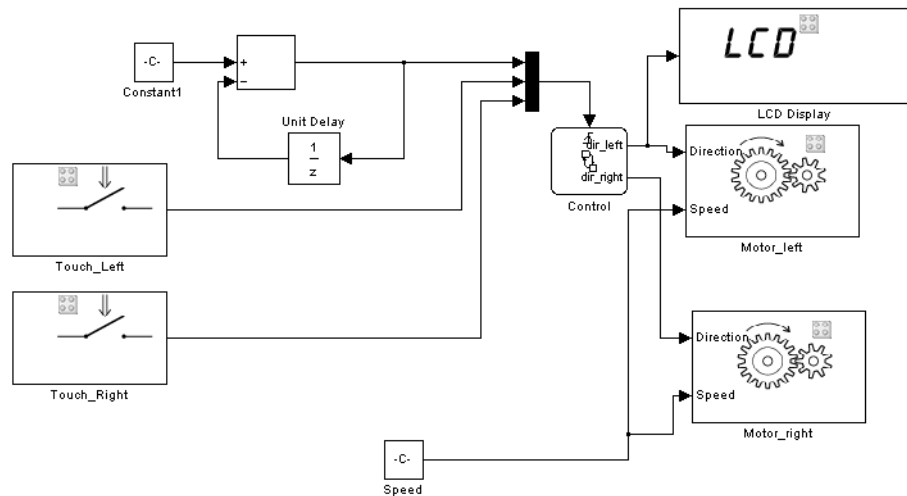


Figure 1-7: LCD Driver Connected to Control Block Output Signal

In the generated application, the driver code selects a string (such as FWD or REV) based on the value of the input signal. If the value is outside the expected range, the driver indicates an error message, (such as ERR) on the LCD.

Driver Files

Your LCD driver implementation will include the following files:

- `lcd.c`: C implementation of the driver S-function for simulation.
- `lcd.dll`: MEX-file component, built from `lcd.c`.
- `lcd.tlc`: TLC implementation of the block for inlined code generation.

When you implement the driver, you should place these files in the ECRobot target directory. In addition, the file `lcd_img.tif` is provided in the `ECRobot\solutions` directory for use as a block icon (as shown in Figure 1-7) if desired.

Simulation Implementation

The LCD block is a masked S-function block, with one parameter, **Sample time**. The mask is shown in Figure 1-8.

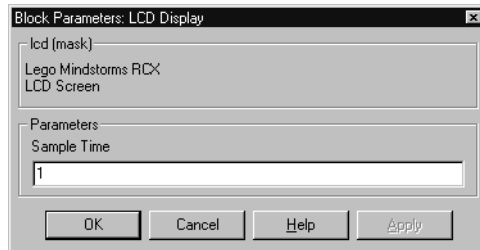


Figure 1-8: Mask Dialog for LCD Block

The S-function name is `lcd`. The LCD block should:

- Act like a ground block during simulation, producing no output.
- Validate that the **Sample time** parameter is scalar and display an error message if a nonscalar parameter is entered.
- When the simulation runs, display a simple message to the MATLAB window indicating that the LCD block acts like a ground block during simulation.

Target Language Compiler Implementation

Like the other ECRobot drivers, the Target Language Compiler (TLC) implementation of the LCD driver is a wrapper, calling legOS functions to drive the LCD.

Includes. The legOS functions and definitions required for the driver are in the header files `legos_root/include/conio.h` and `legos_root/include/dmotor.h`. The generated code must include these files. (Note that if the generating model contains a Motor block, `dmotor.h` will already be included).

The *Introduction to the legOS Kernel* guide (see “Further Reading” on page 1–43) describes the legOS LCD functions.

Validation. There is only one LCD display on the target hardware. Therefore during code generation, the TLC implementation of the driver must raise an error if more than one LCD block exists in the model.

Initialization. The driver should clear the LCD, using the legOS function `cls()`. See the declaration in `legos_root/legos/include/conio.h`.

Input/Output. The driver has a single input. The expected values of the input signal correspond to the possible states of a motor. Input signal values should be stored in a local variable of type `MotorDirection`. `MotorDirection` is a legOS enumerated data type representing all possible motor directional states.

```
typedef enum {
    off = 0, //!< freewheel
    fwd = 1, //!< forward
    rev = 2, //!< reverse
    brake = 3, //!< hold current position
} MotorDirection;
```

`MotorDirection` is defined in `legos_root/include/dmotor.h`.

The driver does not generate an output signal. The driver's output function should obtain the LCD block's input signal value, store it in a local variable, and select (by your choice of logic) an appropriate message string to display on the LCD screen.

In addition to indicating all expected values, the driver should handle illegal values by displaying an error string.

The message strings must be standard (null-terminated) C character strings of maximum length 9 bytes.

To display the string, call the legOS function `cputs()`. See the declaration in `legos_root/include/conio.h`.

Suggestions for Implementing Your LCD Driver

- Make sure that you have correctly installed the ECRobot target and tested your installation as described in "Operating the Target" on page 1-6.
- As a starting point, we suggest that you copy and rename the Touch Sensor block files (`touch.c` and `touch.tlc`) to `lcd.c` and `lcd.tlc` as a starting point for your driver implementation.

- Create and test your C implementation of the driver S-function for simulation first:
 - Create a masked S-function block, using the Touch Sensor block as a model. The mask should look like Figure 1-8.
 - Adapt the parameter validation code to handle only one parameter, which should be declared nontunable.
 - Adapt the message that is displayed during simulation
 - Implement `MdlOutputs` as a stub function.
 - A `mdlRTW` function is not required.
 - Build the MEX-file `lcd.dll` using the `mex` command, and make sure that your S-function block is correctly bound to `lcd.dll`.

If you are not sure how to do this, see “Building the MEX-File and the Driver Block” in the “Targeting Real-Time Systems” chapter of the *Real-Time Workshop User’s Guide*.
 - Test the block by adding it to the `robot3` demonstration model as shown in Figure 1-7. Test your parameter validation code. Then run the model, and make sure your block displays the expected message in the MATLAB window.
- When your simulation implementation is working correctly, create your TLC implementation:
 - Using `touch.tlc` as a template, you need only adapt existing functions to the specification given.
 - For extra credit, use a `BlockTypeSetup` function to generate the required `#include` directive, instead of `BlockInstanceSetup`.
 - Make sure that the `%implements` directive specifies the block type correctly.
 - Make sure to include directives that generate block comments identifying the `lcd` block.
 - Before building the program, turn on the **Generate HTML report** option in the **General code generation options** category of the Real-Time Workshop page. This facilitates browsing the generated code.
 - Build a program from the demonstration model. The HTML code generation report automatically opens in the MATLAB Help browser. The code for your driver, like the code generated by the touch sensor driver, is

written to the generated `robot3_common.h`, `robot3_reg.h`, and `robot3.c` files. Use the links in the report to inspect these files and verify that the expected code was generated.

- Assuming you had the **Download program to robot** option on during the build, and that there were no communication or other errors, you can now run your generated program. If you did not use this feature, download the `robot3.lx` file and run the program. By manually activating the touch sensor(s), you can change the motor direction. Observe that the correct messages are displayed on the LCD when the direction changes.

Solution

Simulation Implementation

Refer to `ECRobot\solutions\lcd.c`.

To make the LCD block act like a ground block during simulation, `MdlOutputs` is implemented as a stub.

The function `mdlCheckParameters`, called from `mdlInitializeSizes`, validates that the **Sample time** parameter is scalar.

```
static void mdlCheckParameters(SimStruct *S)
{
    /*Check that parameters are scalar*/
    if ( mxGetNumberOfElements(SAMPLE_TIME_PRM(S)) > 1)

        {
            ssSetErrorStatus(S, "The LCD block parameters must be
scalar.");
            return;
        }
}
```

`mdlInitializeConditions` displays a message about the block to the MATLAB window when the block initializes itself.

Target Language Compiler Implementation

Refer to `ECRobot\solutions\lcd.tlc`.

The Target Language Compiler (TLC) implementation contains three simple functions:

- The `BlockInstanceSetup` function generates an include statement for the legOS API file `conio.h`. `BlockInstanceSetup` also checks for multiple instances of the LCD block in the `model.rtw` file.
- The `Start` function generates only a comment and a single line of code, which calls the `legOS cls()` function.
- The `Outputs` function stores the input signal value in a local variable of type `MotorDirection` and uses a switch statement to select a string for display; it then passes the string to the `legOS cputs()` function. The code generated by the `Outputs` function is shown in the following code excerpt from the `model_step()` function.

```
/* S-Function Block: <Root>/LCD Display */
{
    MotorDirection dir = robot3_B.dir_left;
    char * msg = NULL;
    switch(dir)
    {
        case off:
            msg = "OFF";
            break;
        case fwd:
            msg = "FWD";
            break;
        case rev:
            msg = "REV";
            break;
        case brake:
            msg = "BRAKE";
            break;
        default:
            msg = "ERROR";                /* should never happen */
            break;
    }
    cputs(msg);
}
```

Further Reading

To supplement this document, we recommend that you read the following:

- The *Real-Time Workshop Embedded Coder User's Guide*, especially the “Data Structures and Code Modules” and “Program Execution” sections. These describe the functioning of the main program and execution engine of the Real-Time Workshop Embedded Coder. Also see “Generating a Code Generation Report” if you are not familiar with the **Generate HTML report** option.

- “Customizing the Build Process” and “Tutorial: Creating a Custom Target Configuration” in the “Targeting Real-Time Systems” chapter of the *Real-Time Workshop User's Guide*.

These sections discuss the detailed structure of system target files and template makefiles.

- “Using Masks to Customize Blocks” in *Using Simulink* describes how to create a mask for an S-function.
- “Writing S-Functions for Real-Time Workshop” in the *Writing S-Functions* manual.

This chapter includes information on writing wrapper S-functions. Wrapper S-functions are used to generate code for the device drivers of the ECRobot target.

- The *Target Language Compiler Reference Manual* documents all functions of the Target Language Compiler and contains further information on wrapper and fully inlined S-functions.
- The *Introduction to the legOS Kernel* guide describes the RCX hardware and the architecture of legOS, including the legOS API. The guide is available in PDF format at the following URL:

<http://legos.sourceforge.net/docs/kerneldoc.pdf>.

- The following books contain information about LEGO MINDSTORMS and legOS:

- *The Unofficial Guide to LEGO MINDSTORMS Robots*

Jonathan B. Knudsen

O'Reilly 1999

ISBN 1-56592-692-7

- *Extreme MINDSTORMS*
Dave Baum et al.
Apress 2000
ISBN 1-893115-84-4