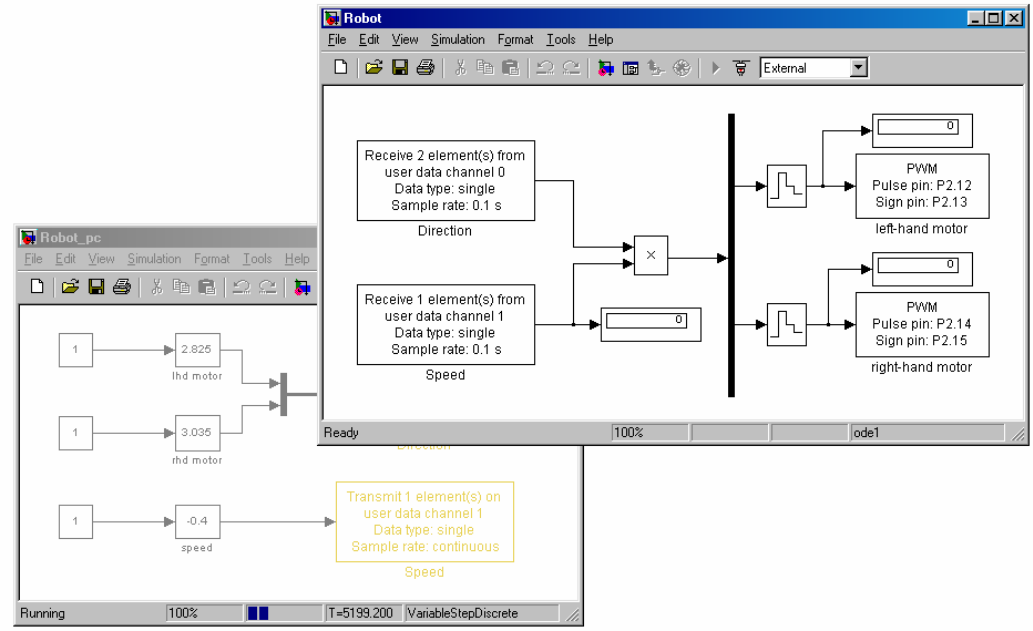
RTMC9S12-Target

A Simulink target for real-time control using Freescale MC9S12DP256B/C microcontrollers



Compiler suite: Metrowerks CodeWarrior

Manual: V 1.12

F. Wörnle, May 2005

Contents

1 Introduction and Background

1.1 Introduction

rtm9S12-Target is a MATLAB/Simulink toolbox for real-time control applications using Freescale MC9S12DP256B/C microcontrollers. The toolbox builds real-time executable code from arbitrary Simulink models (block diagrams). In addition to all normal Simulink blocks, a number of special blocks are provided, giving high-level access to all hardware units of the microcontroller. The controller can thus be programmed without the need for low-level coding in C/C++ or assembler and the usually inevitable testing and debugging of these programs.

In its current version, rtmc9S12-Target provides access to all 16 channels of the two A/D converter units (ATD0, ATD1). There are blocks for reading from and writing to the digital I/O lines, the 4/8 channels of the PWM unit are available (8-bit operation / 16-bit operation) and there are blocks for serially loaded D/A converters, which can be connected to the IIC bus of the chip (e.g. Analog Devices AD5311). Additional hardware units can easily be integrated using user-supplied *s-functions* or Target Language Code script files (tlc).

The target platform (microcontroller) can communicate with a host machine (personal computer, PC) through a serial connection (RS-232, null-modem). This allows for real-time monitoring of process data as well as on-the-fly downloading of control parameters. The host-to-target communication has been based on Simulink's *External Mode* interface, thereby fully integrating rtmc9S12-Target into the Simulink environment. This makes it possible to simulate a model on the host (normal mode) before building real-time executable code to be run on the target platform (external mode). No changes have to be made to the block diagram when switching from normal mode to external mode.

A useful extension of the External Mode interface has been devised to provide a set of upload and download channels for the (optional) exchange of user data telegrams between host and target. This feature allows the microcontroller to access resources which are available on the host, e.g. PC based data-acquisition cards, vision systems, data bases, etc. Using the host as a simple information server, the microcontroller is given access to an extremely wide range of applications.

rtmc9S12-Target is the port of a toolbox for Infineon C167 microcontrollers (www.mecheng.adelaide.edu.au/robotics/WWW_Devs/c167Web/RTC167-Target.htm). This toolbox has been developed on a Phytec single-board computer (phyCORE-167, www.phytec.com), a small unit with an Infineon C167CR-LM micrcontroller, 256 kBytes of external Flash ROM and 256 kBytes of external RAM. The relatively large RAM allow for larger communications buffers as possible on the Freescale MC9S12DP256 (only 12 kByte of internal RAM are available). This contributes to the overall robustness of the host-target communication system. A second advantageous feature of the C167 over the 9S12 is a vastly superior timing unit (5 independently programmable timers) and a priority based interrupt system (64 priority levels – there are none on the 9S12). Nevertheless, rtmc9S12-Target is performing reasonably well in most cases. A large number of sample programs have been included to facilitate assist new users with the first few experiments.

At present, rtmc9S12-Target can be run without any modifications on Wytec MiniDragon+ boards as well as Wytec Dragon-12 boards (www.wytec.com). The executable code is generated from a regular MATLAB/Simulink model using

Metowerks' CodeWarrior (V3.0 or later, www.metrowerks.com). Adaptations to other 9S12 based development boards should be straight forward.

The author believes in the usefulness of *free software*. rtmc9S12-Target is thus released under the terms of the GNU Public License Agreement (GPL). The spirit of *free software* incorporates the users' freedom to run, copy, distribute, study, change and improve the software. Commercial developments based on rtmc9S12-Target are permitted within the limits of the GPL. For further details please visit the GNU website or refer to: www.gnu.org/philosophy/free-sw.html.

1.2 Background

The idea for the development of an easy-to-use real-time target for a microcontroller was born in the course of a number of Control Engineering laboratory sessions, held at Glasgow Caledonian University between 1999 and 2001. Experience showed that students often find it difficult to establish strong links between the theoretic concepts taught in class and practical *real world* applications. The mentioned laboratory sessions were aimed at narrowing the gap between the design stage and the development of an actual product. Working in small groups of 2 to 4 students, the participants were asked to develop a microcontroller based mobile robot which could follow a reflecting track on the ground. Each group was handed a simple frame for a rear wheel driven robot, including a pair of H-bridge motor driver circuits and a module with two light-sensitive photodiodes (Figure 1).

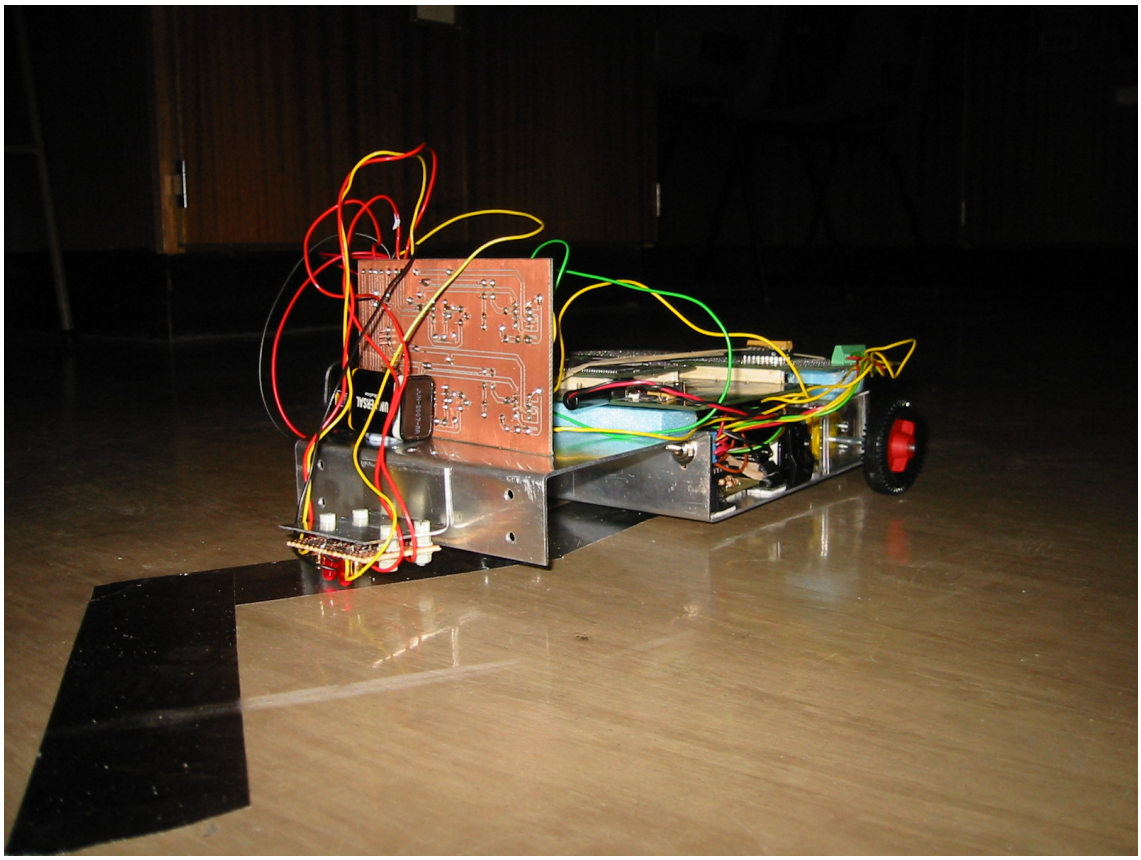


Figure 1 Rear-wheel driven mobile robot (prototype)

The students generally liked this exercise because of its integrative nature, combining many aspects from a variety of areas: control engineering, programming, electronics, communications and project management. However, it soon became apparent that the 5 laboratory sessions (2h per session) were insufficient to bring the exercise to a successful and satisfying conclusion. Judging the situation from a Control Engineering point of view, too much time was “wasted” with tedious programming tasks and the inevitable debugging of the developed programs. To remedy this problem the MATLAB toolbox RTC167-Target was written, giving high-level access to all essential features of the controller using a customised set of Simulink S-Function blocks. This completely eliminated the process of having to convert a control algorithm – often given in form of

a block diagram – into a C-language program for the microcontroller. Using the toolbox, no time is wasted with learning a new software development environment (compiler, debugger) or microcontroller specific details such as hardware registers, memory models, startup files, etc. By reducing this development time, the focus of the exercise can be shifted from low-level development to system-level control engineering. This opens the door to realistic real-world control applications, e.g. robot control using machine vision, process control experiments, intelligent system of multiple autonomous mobile robots, etc.

Like all other Real-Time Workshop (RTW) targets, rtmc9S12-Target generates code from a Simulink model (block diagram). The build process initially turns the block diagram into a series of ANSI-C source code files. These files are then cross-compiled and linked to a single MC9S12 executable. Once downloaded into the FLASH-ROM of the microcontroller, the code can be controlled using the graphical user interface of Simulink. The build process is fully automated and can be customised through the Real-Time Workshop options panel. The generation of timing signals and other status information can be configured. The options panel also allows the setting of the serial communication parameters (host COM port, baudrate).

The present version of rtmc9S12-Target makes use of Metrowerks' CodeWarrior Integrated Development Environment (www.metrowerks.com). The *full version* of this compiler is required to build the executables. Inexpensive licenses are available to educational institutions. rtmc9S12-Target has been developed and tested on MATLAB version 6.5.1 (R13.1).

The remainder of this document presents a detailed description of the installation and use of rtmc9S12-Target. The hardware used for this development was a Wytec Dragon-12 development board (www.wytec.com). Appendix A summarises the key features of this rapid development board. A more compact version of the Dragon-12 is the MiniDragon+ development board (also Wytec). The toolbox has successfully been used on the MiniDragon+ board. Appendix B summarises the key features of the MiniDragon+ board. A number of currently known problems and recommendations for future extensions are listed in Appendix C.

It is hoped that this contribution will be helpful to many users in education, science and research. Any comments and feedback on possible flaws within the code of rtmc9S12-Target are very much appreciated and should be directed to:

Frank Wornle (frank.wornle@mecheng.adelaide.edu.au)

The University of Adelaide
School of Mechanical Engineering

13/04/05

2 The Simulink real-time target 'rtmc9S12-Target'

2.1 Installation

2.1.1 Configuring rtmc9S12-Target

The installation of rtmc9S12-Target is simple: Extract the contents of the archive file *rtmc9S12_CW_R13.zip* to an empty folder of your choice and follow the instructions given below. In this document, the installation folder is referred to as <*rtmc9S12_TARGET_ROOT*>.

Following the successful extraction of the archive file (rtmc9S12_CW_R13.zip), please launch MATLAB and run the m-file 'setup.m' which can be found in the installation folder <*rtmc9S12_TARGET_ROOT*>. This GUI driven MATLAB script ensures that all installation specific path information is suitably adjusted (Figure 2-1). This includes the search path variables to the *compiler installation folder* (Codewarrior) as well as the *work directory* in which the rtmc9S12-Target models can be found. Throughout the remainder of the present document, this latter folder will be referred to as <*WORK_ROOT*>.

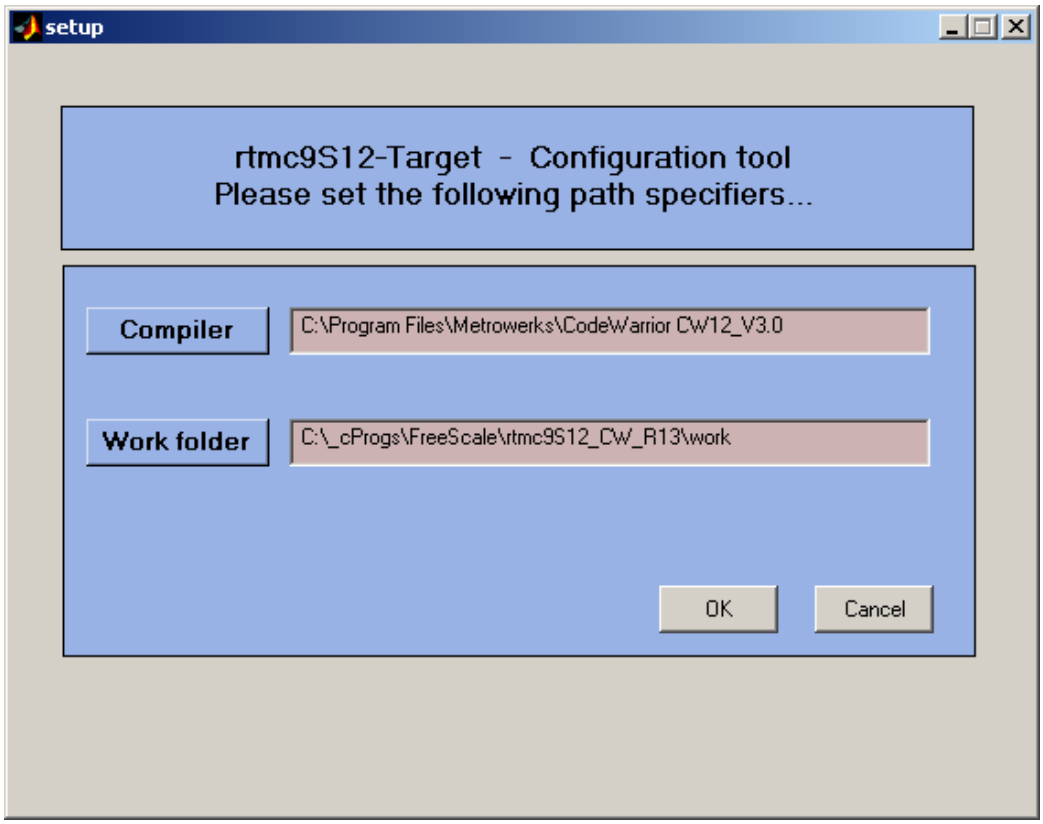


Figure 2-1 rtmc9S12-Target Configuration tool

The path information can directly be entered in the two textboxes. Alternatively, a folder browser can be displayed by clicking the corresponding pushbuttons ('Compiler', or 'Work folder'). Notice that only the names of *folders* are displayed; the names of all other files are suppressed.

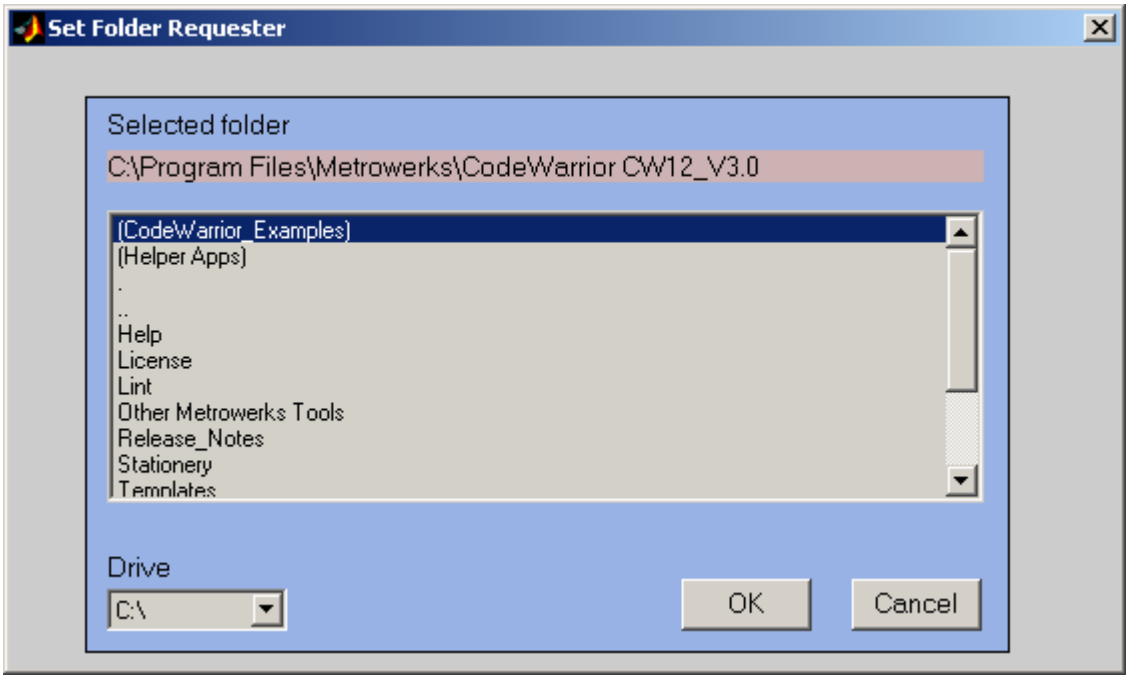


Figure 2-2 Folder browser

The setup script adds the following two folders to the MATLAB search path variable:

<rtmc9S12_TARGET_ROOT>\bin
<rtmc9S12_TARGET_ROOT>\mc

Finally, the setup script copies a number of customized system files to the required folders. The m-file *make_mc9S12.m* is copied to <MATLAB_ROOT>\toolbox\rtw\rtw. Real-Time Workshop calls upon this file during the build process of a model; *make_mc9S12.m* is a slightly modified version of the original MathWorks' script file *make_rtw.m*.

2.1.2 Increasing the CodeWarrior allowance for heap space

The final step of the installation process is to recompile the CodeWarrior library files. The default installation of CodeWarrior has libraries which allow for up to 2 kByte of *heap* memory space. This is not sufficient when using this toolbox, which uses dynamic memory allocation (malloc, calloc) to create and manage the communication buffers of the external mode interface. Future releases may make use of static memory allocation, but for now it is necessary to increase the heap space from 2 kByte to 7 kBytes.

Find the HC12 library folder of the CodeWarrior installation (this is commonly: C:\Program Files\Metrowerks\CodeWarrior CW12_V3.0\lib\HC12c). In the /include sub-folder, find and open the file *libdefs.h* (see Figure 2-3). Scroll down about 3/4 (line 165 in my installation) to locate the following pre-processor macro definition:

```
#define LIBDEF_HEAPSIZE 2000
```

Change this value to 7000 and save the file. Go back to the HC12c folder (up one level from /include).

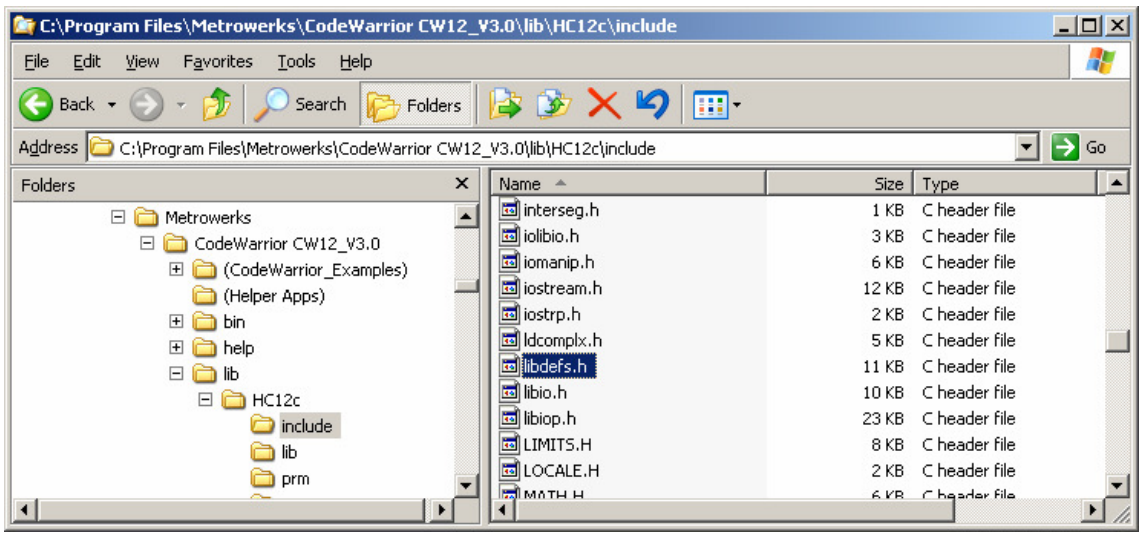


Figure 2-3 Recompiling the CodeWarrior library files for increased heap space

Open the Metrowerks CodeWarrior project file *libhc12.mcp* and recompile (click the 'make' button, see the small circle in Figure 2-4). Exit from CodeWarrior. The toolbox should now be fully operational, ready to be used.

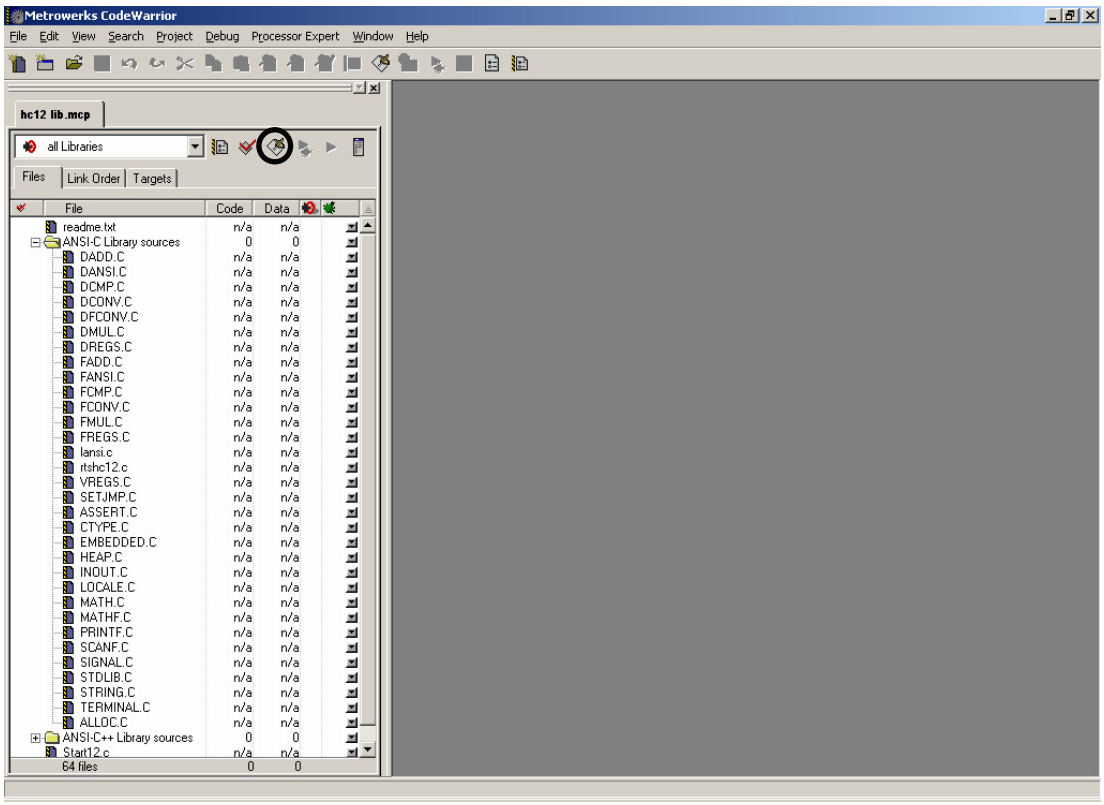


Figure 2-4 Recompiling the CodeWarrior library files for increased heap space

2.2 The rtmc9S12-Target block set

Upon successful installation of rtmc9S12-Target, the Simulink library browser should include a new entry called *Real-time mc9S12 Toolbox*. rtmc9S12-Target contributes 7 new blocks (Figure 2-5): *ADC Input (A/D converter)*, *Digital input*, *Digital output, D/A converter, Pulse-Width Modulation (PWM)*, *Receive user data*, *Send user data*, *FreePortComms_RX* and *FreePortComms_TX*.

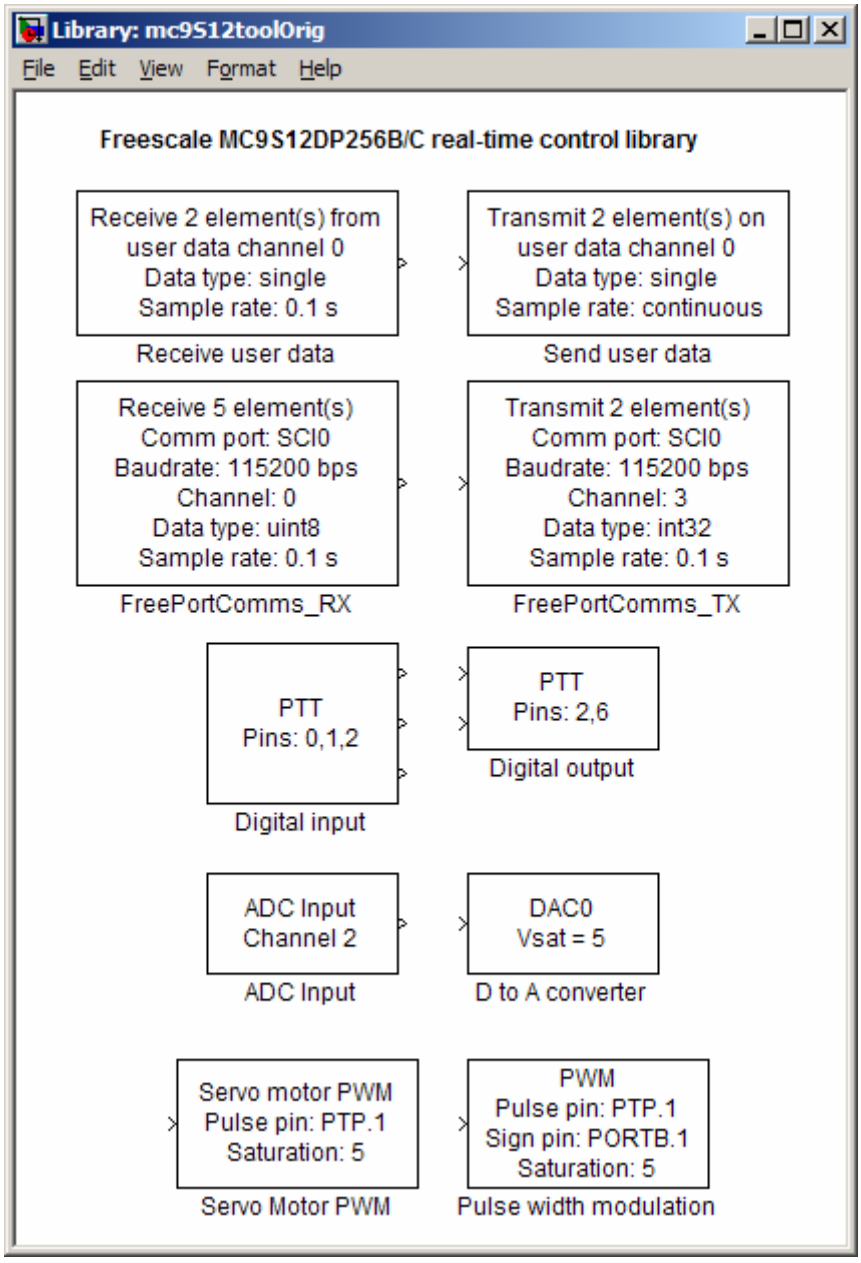


Figure 2-5 The blocks of rtmc9S12-Target

Application examples of all blocks of the toolbox can be found in the working directory <WORK_ROOT>. Wherever a system includes a user communication block (*Receive user data* or *Send user data*), the system comprises of a target model

(<example_name>.mdl) as well as a host model (<example_name>_PC.mdl). See chapter 4 for details about the provided examples.

2.2.1 User communication blocks

An optional user interface between host and target can be implemented with the two blocks *Receive user data* and *Send user data*. These blocks allow the communication between a target model (running on the microcontroller) and a separate host model (running on the PC). The user communication blocks can be used in both the host model as well as the target model. An example of a typical user communication between a host and a target model can be found in chapter 3.

It should be mentioned that the user communication system has been integrated into the *External Mode* communication protocol. User communication block will therefore only work when External Mode interface is enabled (this is usually the case). The user communication system is completely transparent to Simulink: User telegrams are 'smuggled' into the regular data exchange between host and target (defined rather rigidly by the Simulink External Mode interface); the receiving communication partner checks for and removes any incoming user telegrams prior to passing 'regular Simulink data' on to the communication routines of the External Mode interface. The advantage of this approach is that the second Serial Communications Interface (SCI) of the MC9S12 remains unused and can therefore be used otherwise. A disadvantage is that user telegrams reduce the bandwidth which is available for the exchange of log data and/or parameter records between host and target.

The parameters of the user communication blocks are: *Sample time*, *Channel number*, *Number of elements* and *Data type* (Figure 2-6).

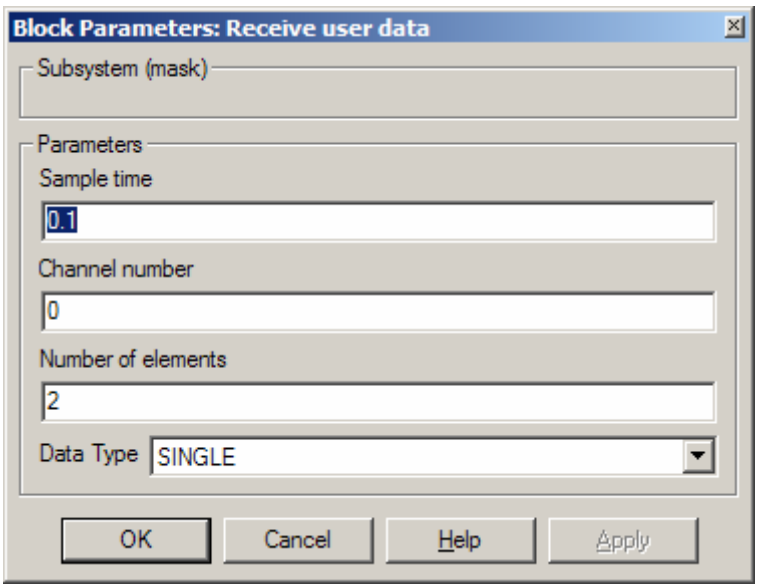


Figure 2-6 Parameters of the block *Receive user data*

The sample time specifies the rate at which the block is updated by the real-time kernel of the target, respectively, by Simulink on the host. Short sample times reduce

the bandwidth which is available for the upload of log data (monitoring of process data) and for other user data channels.

The channel number can range from 0 to 9. It is used to identify corresponding blocks on the host and on the target. To increase the currently available number of 10 user channels the source code macro MAX_UCOM_CHANNELS should be redefined and the affected code sections need to be re-compiled (mex-file *ext_comm_mc9S12.dll*, S-Functions *s0_userTel_rxd* and *s0_userTel_txd*).

The parameter *Number of elements* defines the width of the block input (transmission), respectively, of the block output (reception). Together with the parameter *data type* the number of elements defines the size of the communication buffers. These are currently limited to an overall size of 100 bytes.

2.2.2 FreePort communication blocks

Some applications need simple and reliable communications between several microcontroller platforms or between a microcontroller and host based applications other than MATLAB. The above User communications blocks (see section 2.2.1) are not suitable in this case as they rely on the External Mode interface and therefore only cover communications between MATLAB and the targeted microcontroller.

This problem can be avoided when using *FreePort* communication blocks. The latter allows transmission/reception of short formatted messages via any of the Serial Communication Interfaces of the 9S12 (SCI0 / SCI1) which is not used for *External Mode* communications. The parameters of the FreePort communication blocks are: *Sample time*, *Communication port, Baudrate, Channel number*, *Number of elements* and *Data type*. A tick box allows the transmission of raw data values (Figure 2-7).

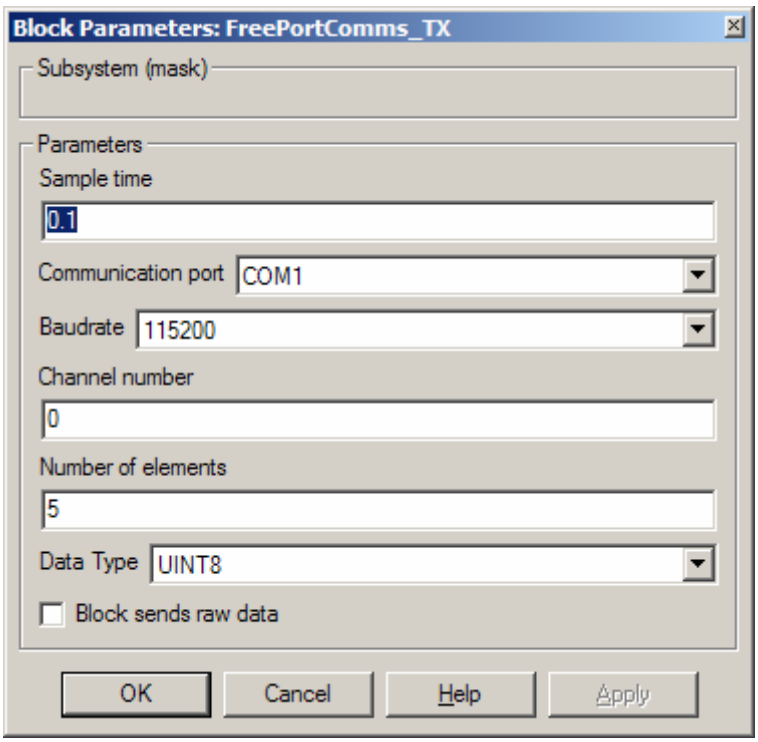


Figure 2-7 Parameters of the block *FreePortComms_TX*

The sample time specifies the rate at which the block is updated by the timing ISR (target sided block) or by Simulink (host sided block). Short sample times reduce the bandwidth which is available for the upload of log data (monitoring of process data) and user data channels.

The channel number can range from 0 to 9. It is used to identify corresponding blocks on the host and on the target. To increase the currently available number of 10 user channels the source code macro MAX_FREEPORT_CHANNELS should be redefined and the affected code sections need to be re-compiled (e.g. the S-Functions *freePortComms_rxd* and *freePortComms_txd*).

The parameter *Number of elements* defines the width of the block input (transmission), respectively, of the block output (reception). Together with the parameter *data type* the number of elements defines the size of the communication buffers. These are currently limited to an overall size of 100 bytes.

The communication port defines whether a block is used as host block (COM1 – COM4) or as target block (SC0 and SCI1). The build process performs a check for and corrects inadequate port settings such as the use of the same SCI port for both External Mode communications as well as FreePort communications. At the writing of this document, baudrates ranging from 300 bps to 115200 bps are supported.

Ticking the box labeled ‘block sends raw data’ causes the underlying S-Function to send unformatted (raw) data telegrams. In this case, only the selected number of data bytes (uint8) is transmitted. No additional overhead such as the channel number, the data type byte, etc. are sent. This is useful when interfacing the microcontroller to intelligent sensors which provide measurements and other information in form of a stream of serial data bytes. Selecting the raw data tick box causes the block mask to change to the reduced format shown in Figure 2-8.

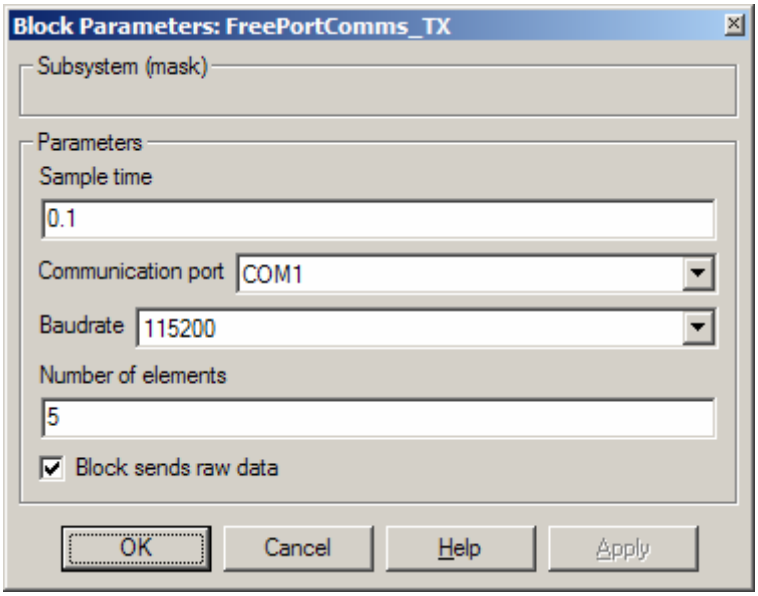


Figure 2-8 Raw data mode – reduced parameter set

2.2.3 A/D Converter unit (ADC)

The A/D converter unit of the MC9S12 can be accessed using the target block *ADC Input*. This block has five parameters: The ADC unit of the microcontroller (ATD0, ATD1), the range of channels to be converted (0 – 7), the resolution of the ADC (8-bit, 10-bit), the output format (raw data, normalized to [0 ... 1], normalized to [0 ... 5]) and the sample time at which the block is to be updated. Figure 2-9 shows the parameter requester of the ADC Input block.

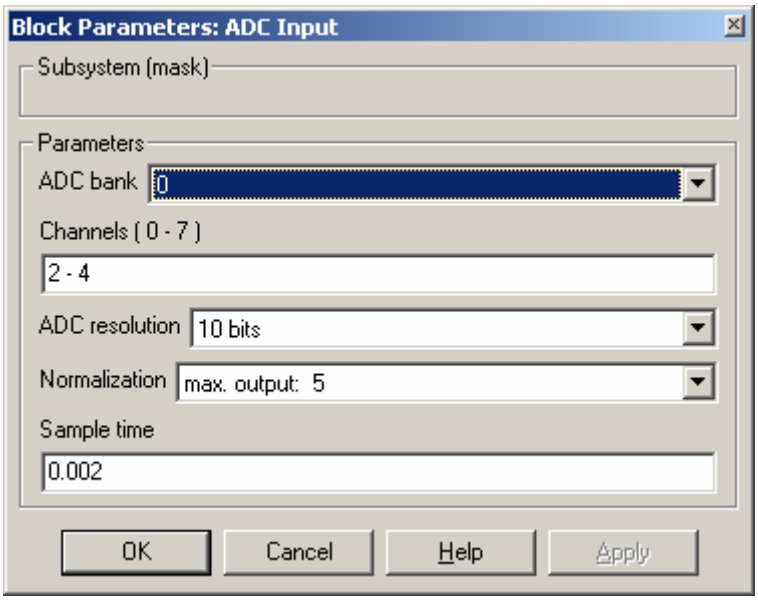


Figure 2-9 Parameters of the block *ADC Input*

2.2.4 Pulse-Width Modulation unit (PWM)

Pulse width modulated (PWM) signals can be generated using the *Pulse-Width Modulation* block. The current version of rtmc9S12-Target offers 4 to 8 PWM channels, depending on the resolution of each individual channel. The unit can be configured for up to 4 16-bit PWM channels or up to 8 8-bit channels. Block inputs can be signed numbers. The sign information is removed and presented on a programmable sign pin. The block mask allows the following parameters to be set (Figure 2-10): *Sample time*, *Resolution, PWM period*, *Pulse pin*, *Sign port, Sign pin, Saturation input level* and an optional *monitoring output*.

The sample time defines the rate at which the block is updated. The PWM period is the period of the created PWM signal. This parameter is limited by the maximum period of the PWM timer (179 seconds).

Pulse pin, Sign port and Sign pin allow the output pins to be chosen on which display the PWM signal should be displayed. The pulse port is fixed (PTP). The sign bit is ‘high’ (+ 5 V) for negative signals and ‘low’ (0 V) for positive inputs.

The Saturation input level defines the magnitude of the block input signal beyond which the PWM output signal has a duty cycle of 100%.

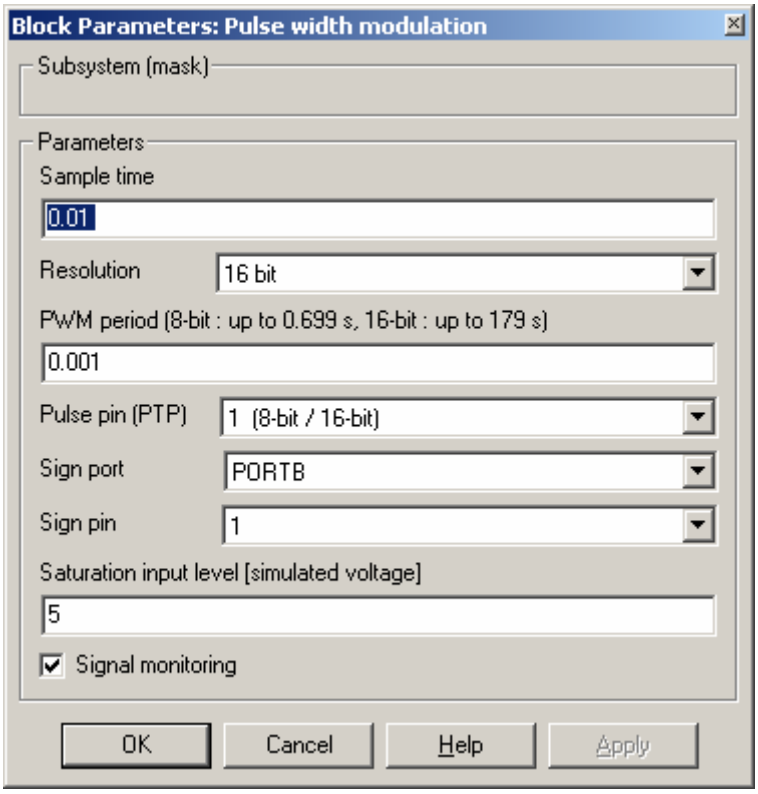


Figure 2-10 Parameters of the block *Pulse width modulation*

An optional block output can be displayed by ticking the *Signal monitoring* check box. This allows the current state of the PWM output signal to be uploaded to the host. Please note that this feature is only useful when the block sample time is much shorter than the PWM period.

2.2.5 Servo Motor Pulse-Width Modulation unit (PWM)

Servo motors are driven by Pulse-Width Modulated (PWM) signals with a very short duty cycle (typically 2% - 12%). These signals can be generated using the *Servo Motor Pulse-Width Modulation* block. The current version of rtmc9S12-Target offers 4 to 8 PWM channels, depending on the resolution of each individual channel. The unit can be configured for up to 4 16-bit PWM channels or up to 8 8-bit channels. Block inputs need to be unsigned numbers. The block mask allows the following parameters to be set (Figure 2-11): *Sample time*, *Resolution, PWM period*, *Pulse pin*, *Minimum Pulse Width, Maximum Pulse Width* and *Saturation input level*.

The sample time defines the rate at which the block is updated. The PWM period is the period of the created PWM signal. This parameter is limited by the maximum period of the PWM timer (179 seconds).

Minimum and maximum pulse widths define the limits of the servo motor. Typical values range from 0.5 ms to 2.5 ms. A duty cycle of 0% will produce pulses with the minimum pulse width; a duty cycle of 100% will generate pulses with the maximum pulse width.

The pulse pin defines on which pin of port PTP the PWM signal should be displayed.

The Saturation input level defines the magnitude of the block input signal beyond which the PWM output signal has a duty cycle of 100%.

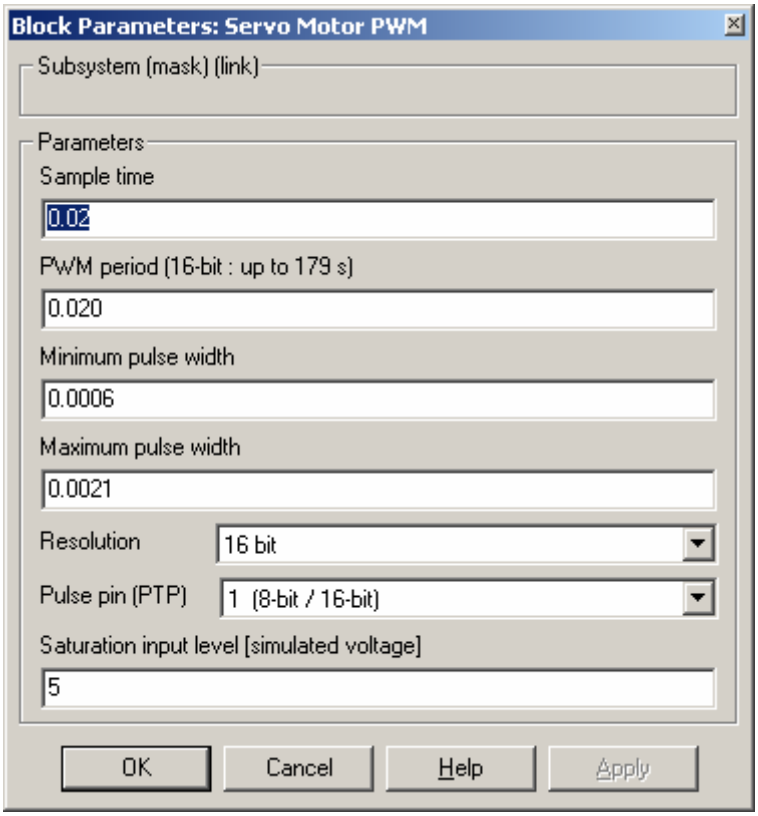


Figure 2-11 Parameters of the block *Servo Motor Pulse-Width Modulation*

An optional block output can be displayed by ticking the *Signal monitoring* check box. This allows the current state of the PWM output signal to be uploaded to the host. Please note that this feature is only useful when the block sample time is much shorter than the PWM period.

2.2.6 Digital input

All available digital inputs of the mc9S12 can be accessed using the block *Digital input*. The block allows specification of *Port, Pin number* as well as the block *sample time* (Figure 2-12). The available ports are PORTA, PORTB, PTH, PTJ, PTM, PTP, PTS and PTT. For each of these ports, one or more pins (0 – 7) can be selected. The mask of block *Digital Input* is adjusted to the number of pins that has been chosen. Example: specifying pins '2 4 5 7' yields a *Digital Input* port with 4 block outputs. The top-most block output corresponds to pin '2' and the bottom-most output is pin '7'.

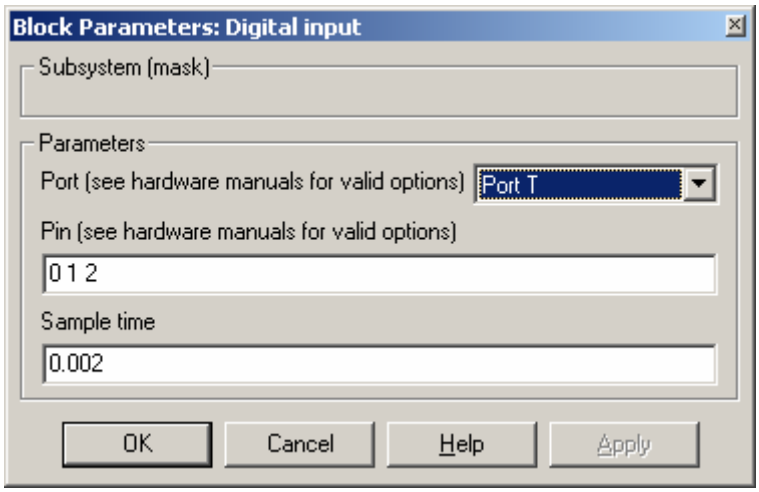


Figure 2-12 Parameters of the block *Digital input*

2.2.7 Digital output

Digital outputs can be programmed using the block *Digital output*. In analogy to the corresponding input block (section 2.2.3) this block also allows specification of *Port, Pin number* and *Sample time* (Figure 2-13). In addition to these block parameters, a lower and an upper block input signal threshold can be specified: *On-threshold (Von)* and *Off-Threshold (Voff)*, respectively. These two levels are used to decide when the block output should become 'high' (above Von) or low (below Voff). Setting Von > Voff allows the programming of a hysteresis.

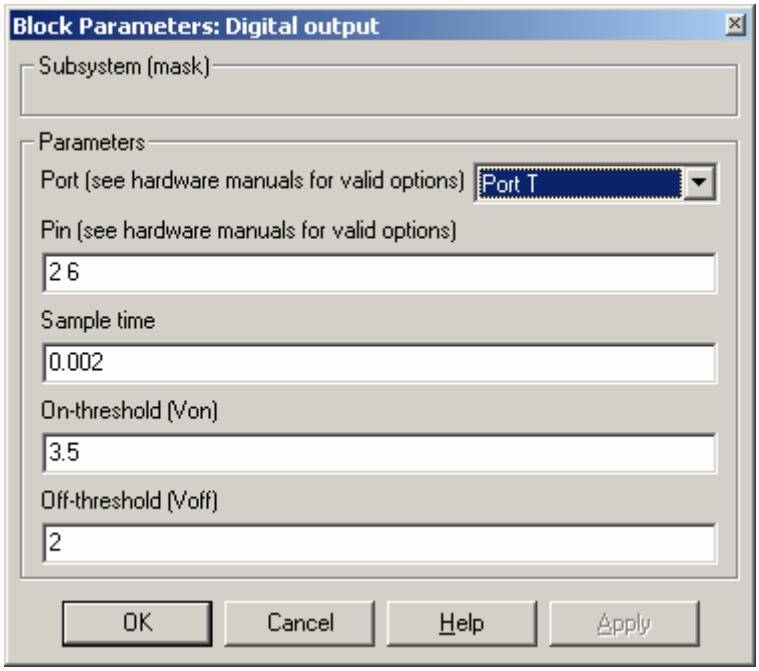


Figure 2-13 Parameters of the block *Digital output*

2.2.8 D/A Converter unit (DAC)

Analogue output signals can be generated using the two external D/A converters (DAC0, DAC1). Two serially loaded Analog Devices D/A converters (AD5311) have been included on the protective circuit board we use in our laboratory. They are loaded through the IIC bus interface of the MC9S12 and output DC voltages between 0 and 5 volts. The block parameters are *Sample time, DAC channel* and *Saturation level* (Figure 2-14). The latter is used to define the correspondence between block input signal level and the full-scale output of the DAC.

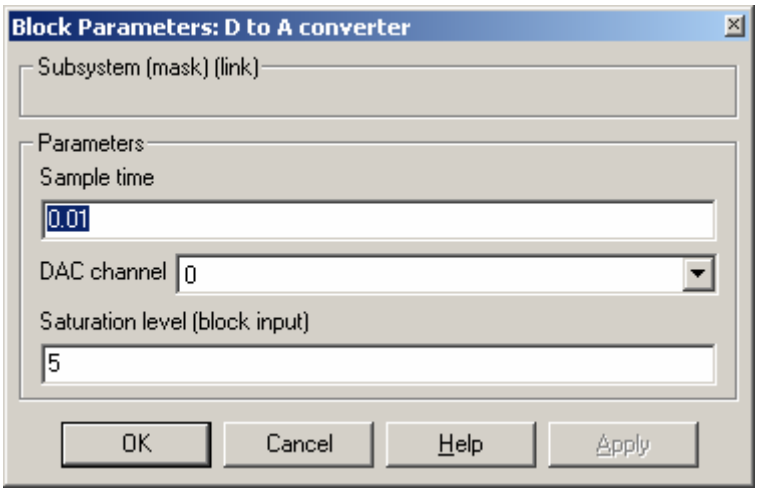


Figure 2-14 Parameters of the block *D to A converter*

2.3 Code generation options

Real-time Workshop (RTW) allows a number of options to be specified during the code generation for a particular target platform. All RTW options can set through the menu *Tools→Real-Time Workshop→Options…* . Figure 2-15 shows the target configuration page of rtmc9S12-Target.

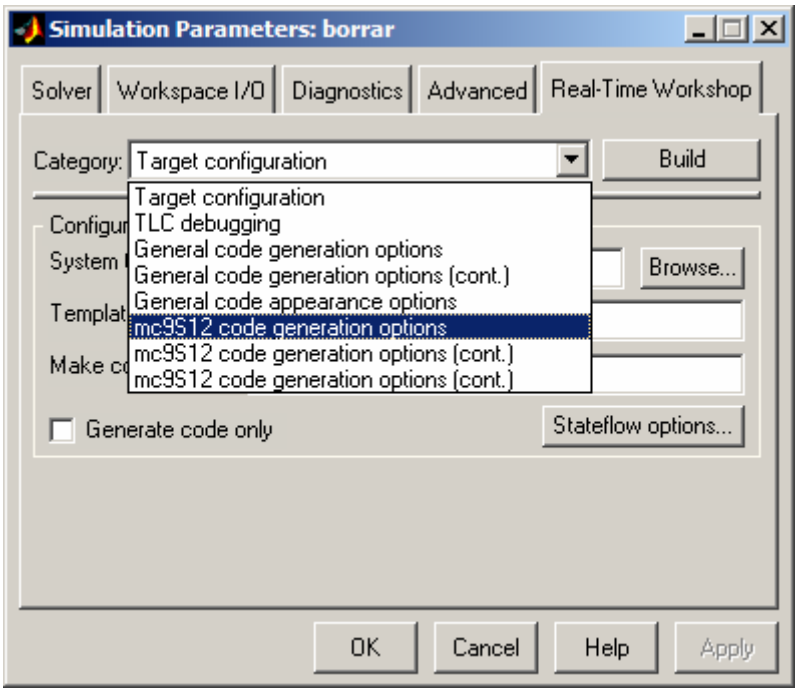


Figure 2-15 Target configuration page of rtmc9S12-Target

Target specific code generation options can be set by choosing the category *mc9S12 code generation options* (Figure 2-16).

This category gives access to the main target configuration options. Choosing a particular target board (currently supported boards are: *Dragon-12* and *MiniDragon+*) presets all remaining options with target specific default values. This accounts for the fact that the Dragon-12, with its two serial port connectors, is commonly programmed through Serial Communication Interface SCI0, whereas the data exchange with MATLAB/Simulink uses SCI1. On the other hand, the MiniDragon+ only has one serial port connector. The default options for this board therefore specify SCI0 for both programming as well as communications. Other options include the host sided communication port (COM1 – COM4) and the serial communication speed (300 bps – 115200 bps). De-selecting the check box *External mode* disables both the user communication between host and target, as well as the upload of log data and the on-the-fly download of process control parameters. Choosing this configuration reduces the code size of the generated run-time executable to a minimum.

Note:

The new *FreePort* communication blocks are independent of the External Mode interface. They can be used to upload/download short message telegrams (currently limited to 100 bytes) from/to the target. When used in parallel to the External Mode communication system, *FreePort* communications may appear somewhat sluggish. This is because these telegrams can only be sent once every sample step. The External Mode interface on the other hand uses the entire available idle time between subsequent sample steps for upload/download. Further details about *FreePort* communication can be found in chapter 2.2.2.

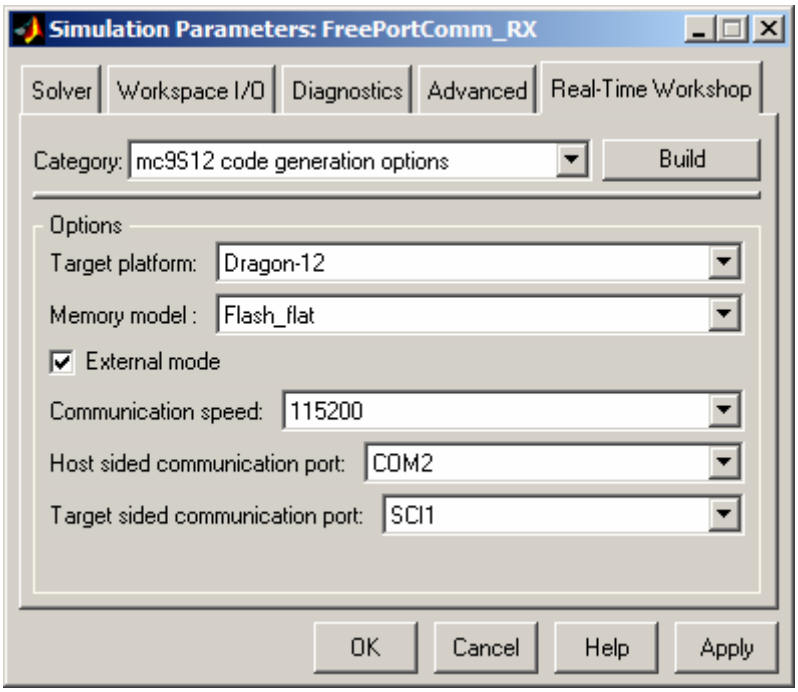


Figure 2-16 Category: mc9S12 code generation options

Option *Memory model* defines the memory layout of the generated runtime code. Two settings are currently available: *Flash_flat* and *Flash_banked*. The Flash_flat memory model assumes a 16-bit linear address space (0 – 64 kByte). RAM is located from 0x1000 – 0x3FFF (12 kByte); ROM exists in two blocks, one from 0x4000 – 0x7FFF (16 kByte), the second from 0xC000 – 0xD7FF (6 kByte). Approximately 7 kByte of the available RAM is assigned to the HEAP and can be allocated dynamically using *malloc* and/or *calloc*. Note that the upper ROM block has been limited to the first 6 kByte. This is only necessary when working with ROM resident system programs in the protected area of Flash. The boards in our laboratory have been installed with a self-test program. For further details about the protected Flash of the 9S12 see:

www.mecheng.adelaide.edu.au/robotics/WWW_Devs/Dragon12/Dragon12.htm.

Model *Flash_banked* produces target executables which use the memory banking mechanism of the 9S12 to expand the address space beyond the 16-bit limit of 64 kByte. The memory window resides at addresses 0x8000 – 0xBFFF. The MC9S12 can be instructed to present any of 16 memory pages (16 kByte each -> 16 x 16 kByte = 256 kByte, this applies to the MC9S12DP_256_B/C) in this memory window. Setting the Memory map parameter to Flash_banked instructs the compiler to build code that makes use of this memory swapping mechanism.

Note:

The memory model *Flash_banked* should only be used if absolutely necessary. It seems that, at the writing of this document, the generated code does not run as reliably as when using the *Flash_flat* memory model. Support for the banked memory model has only just been finished and still requires a lot of testing and fine tuning. Communications between host and target appear to get stuck too easily for reliable operation.

Figure 2-17 shows the second page of target specific options. This page controls a variety of aspects of the generated code as well as the build process. The rebuild of the static libraries of MATLAB can be enforced. Other options control the use of the on-board LCD display of the Dragon-12. The default setting is 'on' for the Dragon-12 and 'off' for the MiniDragon+, as the latter does not feature an LCD display. Option 'RT logging information (debug) on SCI0' should only be used by developers. A number of settings allow the display of target debugging messages on the commonly vacant Serial Communication Interface SCI0 (Dragon-12). Selecting the MiniDragon+ board automatically deselects this option, as the MiniDragon+ commonly uses SCI0 to communicate with MATLAB.

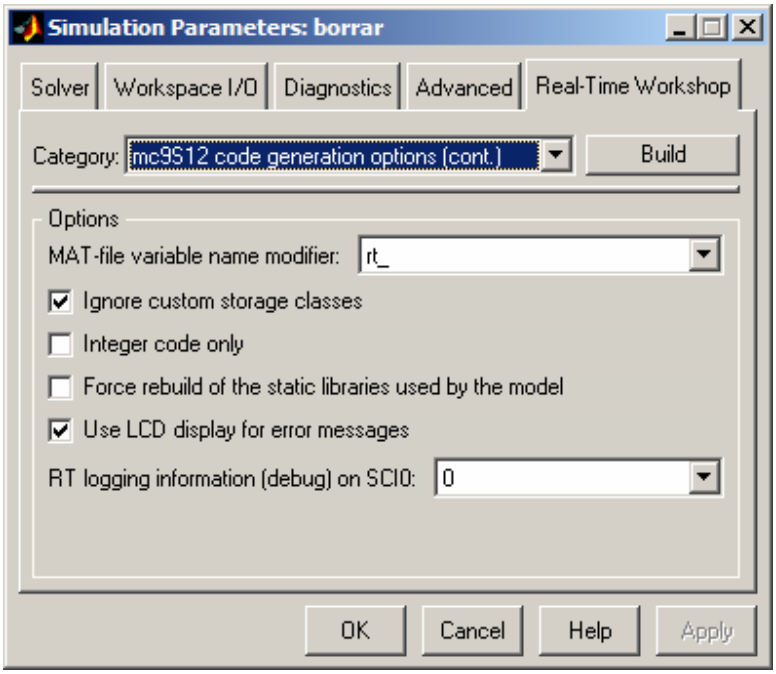


Figure 2-17 Category: mc9S12 code generation options (cont.)

The third options page (Figure 2-18) allows the display of timing signals on a configurable port/pin of the microcontroller. Access to low-level timing information generally facilitates the setting-up of a new control algorithm. The duration of the control cycle is displayed on the chosen I/O pin from where it can be monitored using an oscilloscope. In addition to this so-called *cycle time*, the controller also displays the activity on the serial reception line RxD. This information has proved to come in handy during the debugging of new communication modules. To disable the display of timing signals, deselect the check box *Timing signals*.

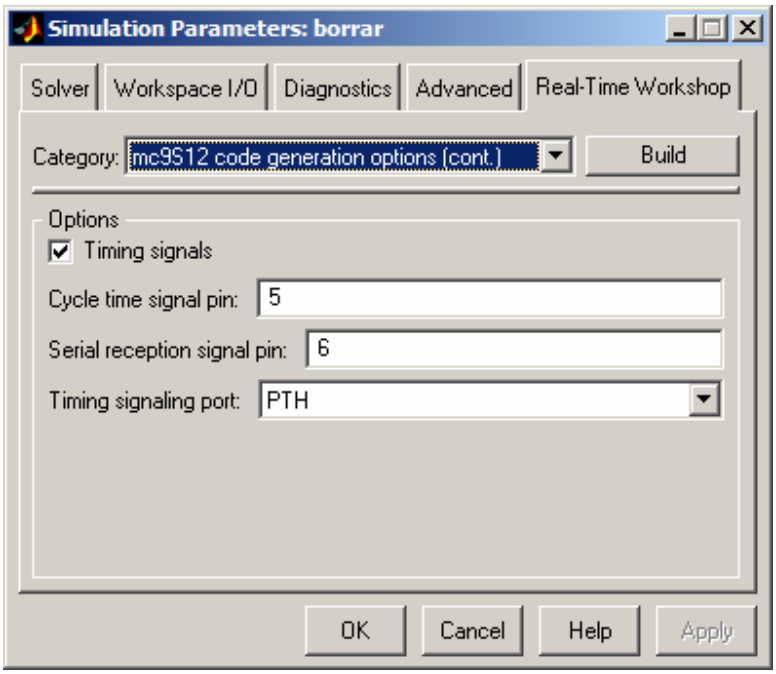


Figure 2-18 Category: mc9S12 code generation options (cont.)

Figure 2-19 shows how the timing signals can be measured using a scope probe. A typical trace of both timing signals as well as the activity on the serial reception line is shown in Figure 2-20.

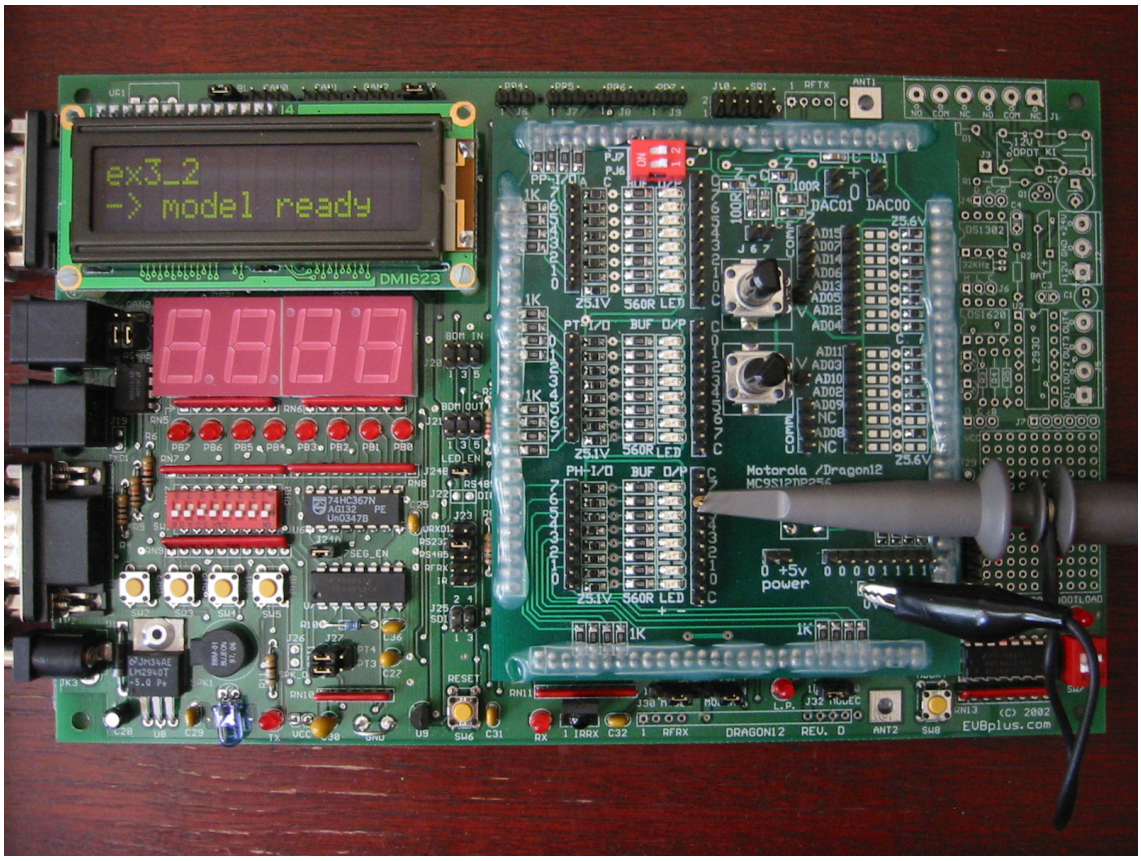


Figure 2-19 Accessing the timing signals on the Dragon-12

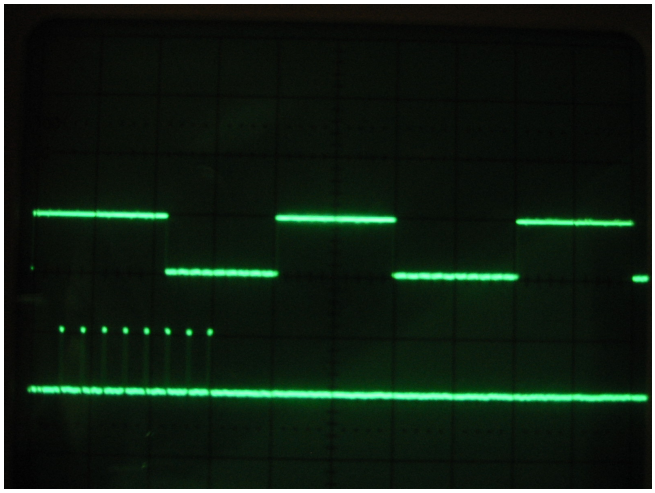


Figure 2-20 Cycle time (top) and serial reception RxD (bottom)

3 Building a simple model – A mini tutorial

The following example gives an outline of the build process of a simple target model with user communication.

Let us begin by creating a new Simulink model. Change to the work folder <WORK_ROOT> and create a new model (*File→New→Model*); save this empty model as *test.mdl*.

Open the library browser. Upon successful installation, you should find a new toolbox entry called *Real-Time mc9S12 Toolbox*. Click on the '+' next to this entry to expand its contents (Figure 3-1).

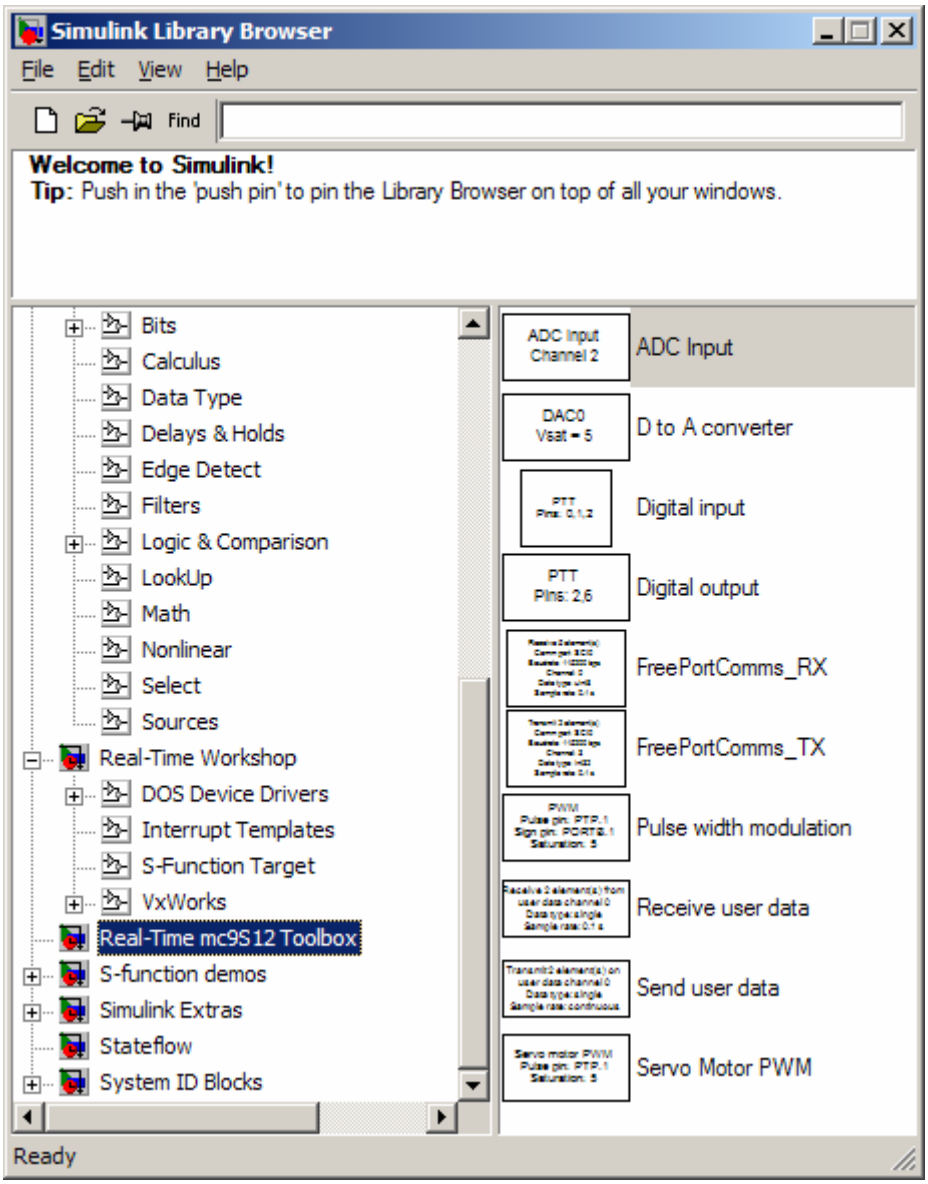


Figure 3-1 The Simulink library browser – Real-Time mc9S12 Toolbox

Drag the block *Receive user data* to the (still empty) model window *test*. From the *sources* category of the main library (Simulink→Sources) select the block *Pulse Generator* and place it in your model. Similarly, from the *sinks* category (Simulink→Sinks) choose a *Scope* block as well as a *Display* block. Link the output of the *Pulse Generator* to the input of the *Scope* block and the output of the *Receive user data block* to the input of the *Display* block. Your model should look as shown in Figure 3-2.

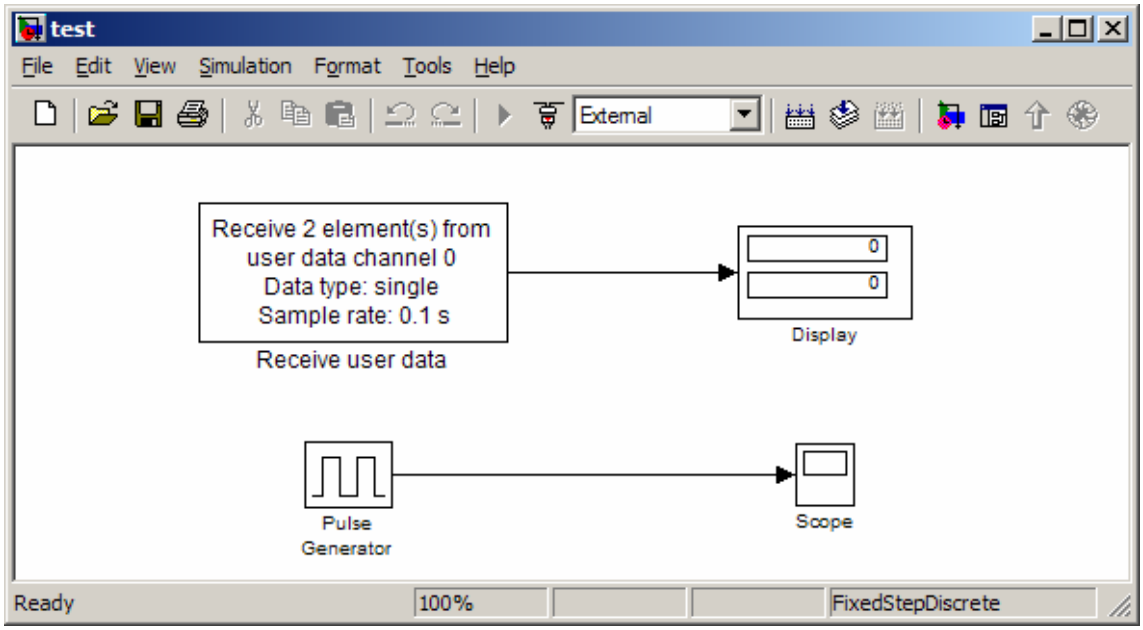


Figure 3-2 A simple example: Real-time pulse train and user communication

Double-click on the *Pulse Generator* block to display its parameters. The default settings are a period of 1 second, a duty cycle of 50% an amplitude of 1 unit and the start-time 0. Change the period to 2 seconds and close the parameter box.

Open the *Receive user data* parameter page by double-clicking the corresponding block icon. The default parameters are a sample time of 0.1 seconds (100 ms), channel number 0 and 2 bytes (UINT8) to be received. Change the sample time to 0.2 seconds, the number of elements to 1 and set the signal data type to SINGLE; close the dialog box.

From the Tools menu select Tools→Real-Time Workshop→Options… . You should be presented with the options dialog box of MATLAB Real-Time Workshop. In the Configuration category click on the push button 'Browse…'. This produces the *System Target File Browser*. Select *rtmc9S12-Target for Metrowerks CodeWarrior* and close the browser dialog box (Figure 3-3).

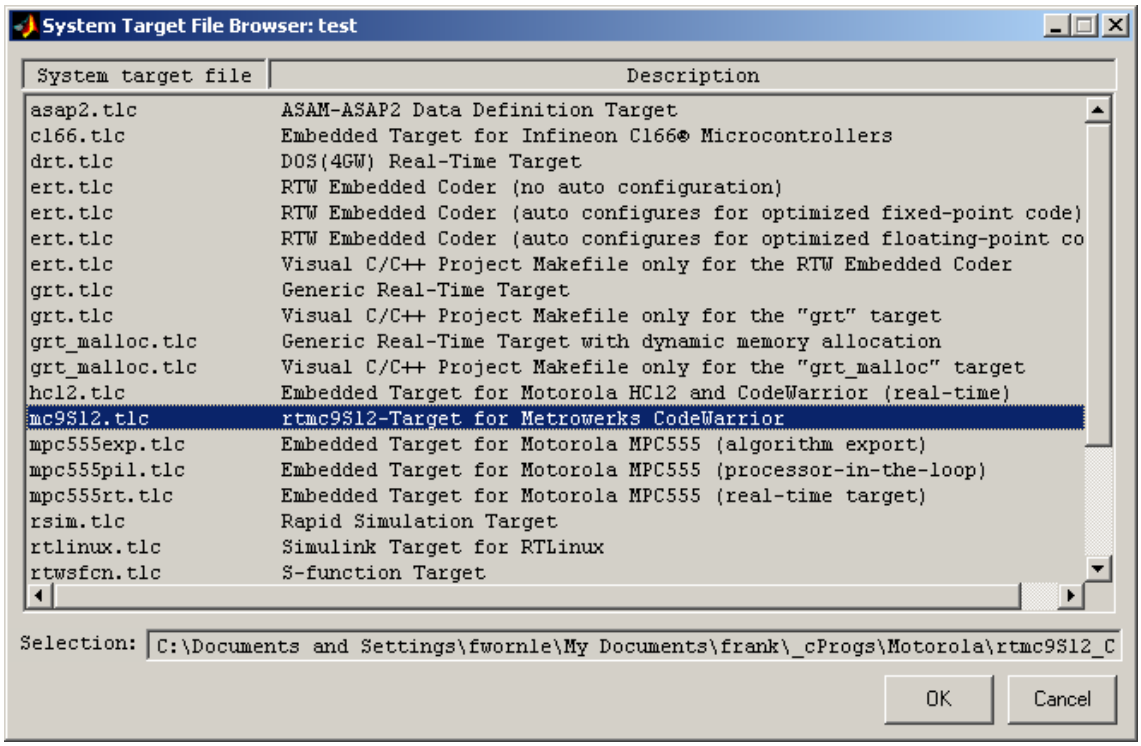


Figure 3-3 System Target File Browser

Please note that the entry for rtmc9S12-Target only appears in the System Target File Browser if MATLAB has been able to locate the file *mc9S12.tlc* on its search path. This should be the case when rtmc9S12-Target is installed using the provided m-file script *setup.m*.

If everything has worked so far, the Options dialog box should look like this (Figure 3-4):

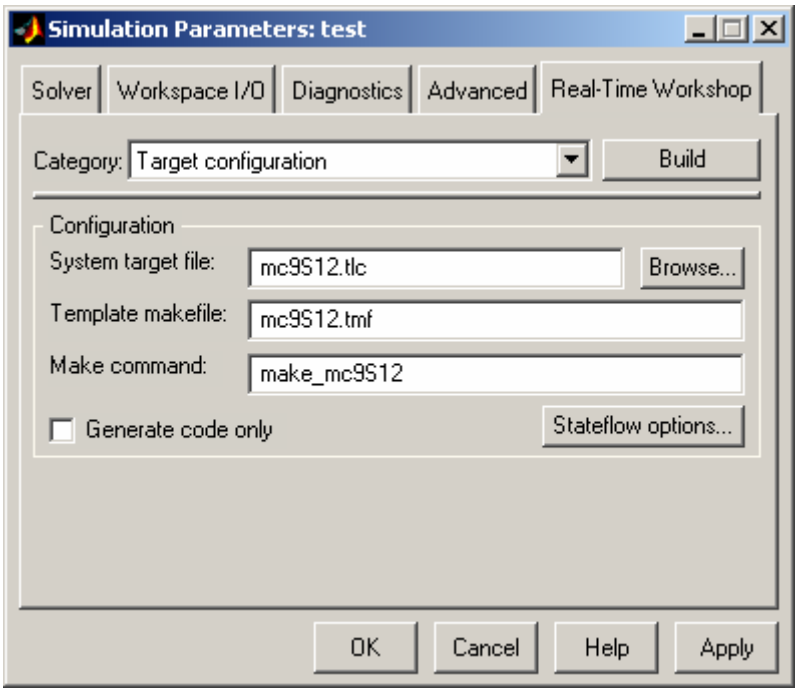


Figure 3-4 Options dialog box – Real-Time Workshop

From the same options dialog box choose the category *mc9S12 code generation options* (cf. Figures 2-12 – 2-14). Ensure that the *External mode* check box is selected and that the communication parameters have been set to 115200 bps and the COM port you are using on your host machine. Deselect the timing signals check box.

Besides these code generation options we also have to choose an appropriate base sample rate for our model. Click on the *Solver* tag of the Options dialog box and change the solver type from *Variable step* to *Fixed step*. Select a step size of 5 ms (0.005). This defines the *base sample rate* at which we intend to run the target program. Care should be taken when choosing this value: Too small a base sample-time might lead to problems as the model has not enough time to execute before the next sample-time hit occurs. Sample-times of less than 1 ms are not recommended.

Change the parameter *Mode* from its default value *Auto* to *Single-Tasking*. This option instructs Real-Time Workshop to build code for a single-tasking environment. Please note that our target model has to perform two jobs, namely the generation of the pulse train (*continuous-time*) and the checking for new user data every 200 ms (*discrete-time*). Real-Time Workshop can service these two jobs from within the base sample rate task (single-tasking) or as two individual tasks (multi-tasking). Please note, however, that the latter approach only works in conjunction with a real-time kernel with a multi-threaded task scheduler. rtmc9S12-Target currently doesn't support this option.

Set the stop time to a reasonably large value, e. g. 999999. This forces the target program to run until we choose to stop it trough the *External Mode Control Panel*. To generate code that truly runs forever, enter a stop time of *inf*. The completed Solver page should appear as shown in Figure 3-5.

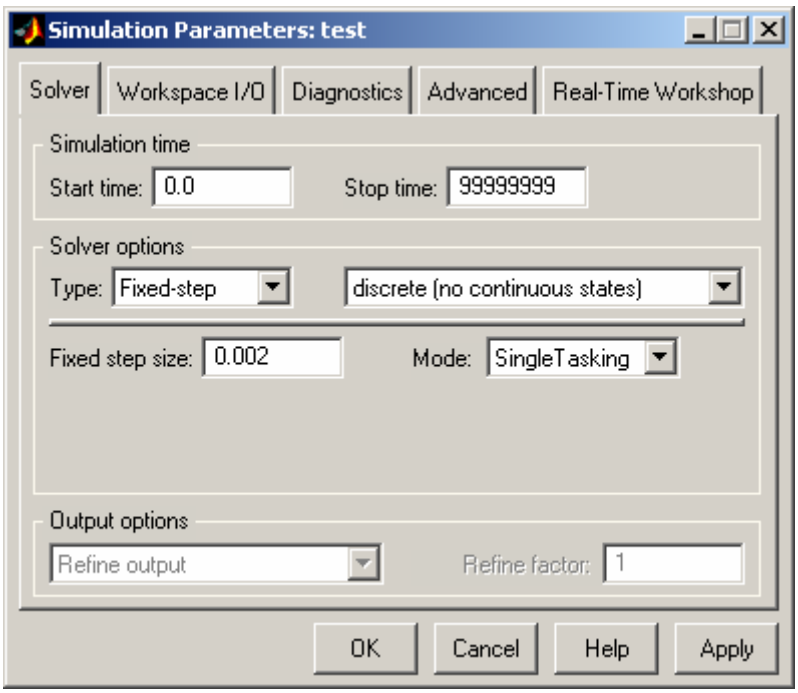


Figure 3-5 Simulation parameters: Fixed step size solver

Close the dialog box using the push button OK and save your model file. When run on the target platform (e. g. the Dragon-12), this simple test program will generate a periodic square wave signal with a 2 seconds period and a duty cycle of 50%. User

telegrams can be downloaded at a maximum rate of 5 Hz (200 ms) and will be displayed on the *Display* block of the block diagram.

However, before we can receive any user telegrams we first have to design a corresponding host model. A host model is only required when the target model includes any of the user communication blocks.

Create a new model and save it as *test_PC.mdl*. From the mc9S12 library we need the user communication block *Send user data*. From the *Sources* category of the Simulink library (Simulink→Sources) select the block *Constant* and place it in your host model. Open the *Math* category of the Simulink library (Simulink→Math) and find the block *Slider gain*. Drag this block to the host block diagram and connect all three blocks as shown in Figure 3-6.

Open the block *Send user data* and choose to all parameters to match those of the reception side, i. e. *channel 0*, *1 element* of type *single*. The sample time can be set to 0 seconds (continuous); this ensures that the host transmits new user data as soon as it decides to update this block. When used on the host, the parameter *Sample time* is relatively meaningless.

To ensure that the host model runs forever, open the simulation parameters menu (Simulation→Simulation Parameters…) and change the Stop time to *inf*.

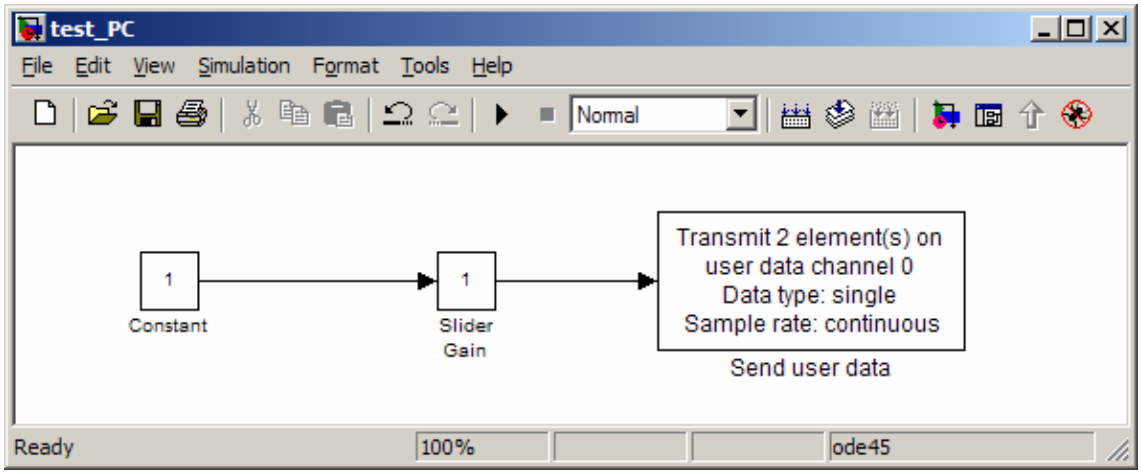


Figure 3-6 Simple test program: Host model

We are now ready to compile the target model and run it on the target hardware. Re-open the target model (*test.mdl*) and select the menus Tools→Real-Time Workshop→Build Model; this initiates the build process of the block diagram. Alternatively, the key-shortcut CTRL-B can be used.

Notice:

When running rtmc9S12-Target for the first time, the toolbox generates a library of all real-time modules supplied by The MathWorks (<MATLAB_ROOT>\rtw\c\libsrc). This may take a few moments. Once compiled, the real-time library is copied to <rtmc9S12_TARGET_ROOT>\rtwlib. Subsequent build processes do not require the library to be rebuilt. At the end of the build process you should be presented with a fully functional CodeWarrior project (Figure 3-7). A number of warnings indicate the

conversion from floating-point numbers to unsigned integers. These can safely be ignored.

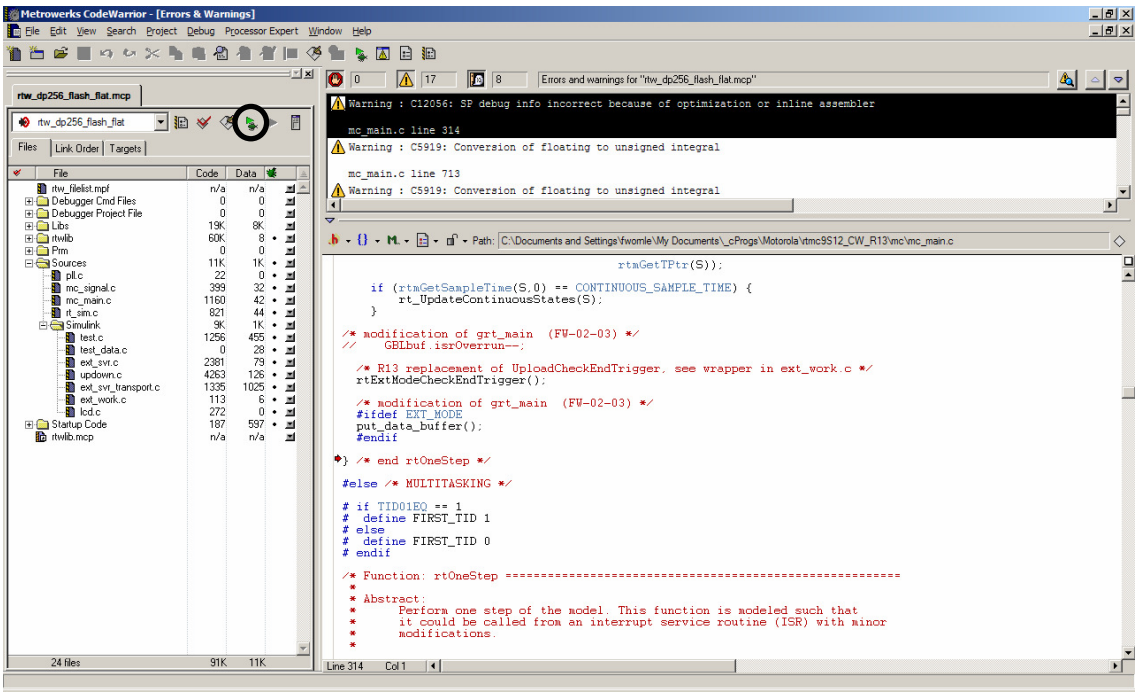


Figure 3-7 MATLAB generates a fully functional CodeWarrior project

Upon successful completion of the build process, the target executable has to be installed in the FLASH-ROM of the target. This can conveniently be done using CodeWarrior. In our laboratory, we use Dragon-12 development boards on which we have installed Motorola's serial monitor (for further details about this monitor program, see: www.mecheng.adelaide.edu.au/robotics/WWW_Devs/Dragon12/Dragon12.htm). This allows CodeWarrior to download the code into the Flash ROM of the 9S12. Simply click the small green arrow (debugger, cf. the small circle in Figure 3-7).

CodeWarrior launches its debugger (Hi-Ware). A small requester appears, indicating the progress of download (Figure 3-8). Once the code has been programmed into Flash ROM, it can be started by clicking on the 'RUN' button (small green arrow, cf. circled button in Figure 3-9).

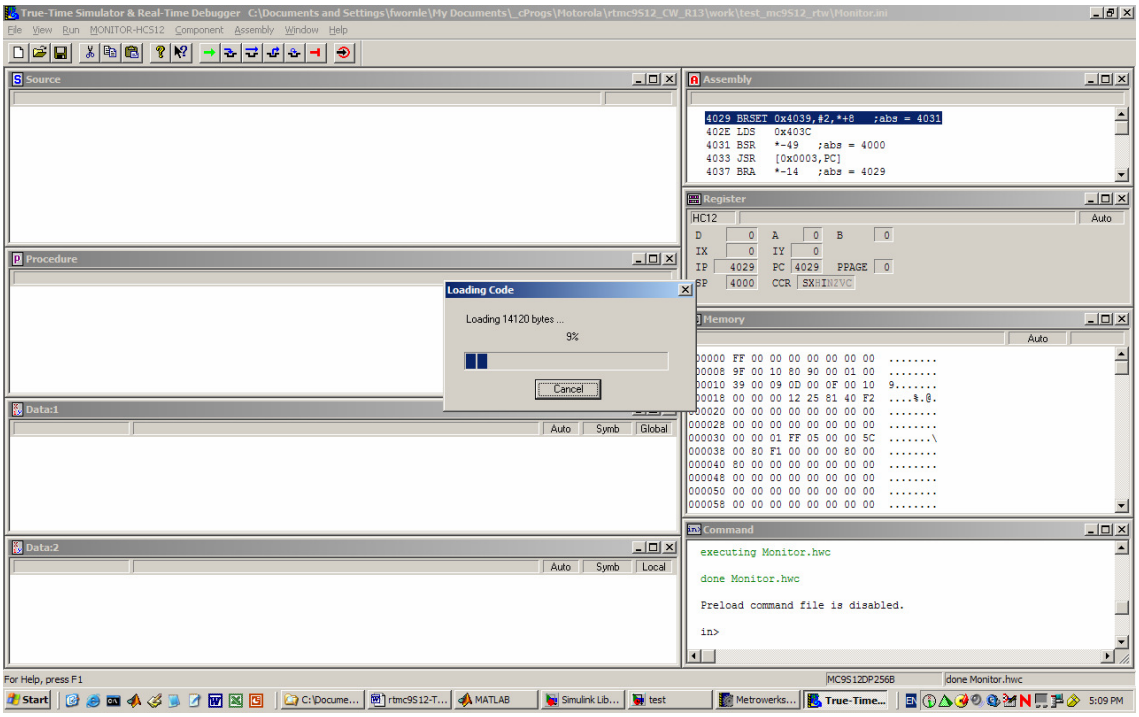


Figure 3-8 Downloading the 9S12 program using the Hi-Ware debugger

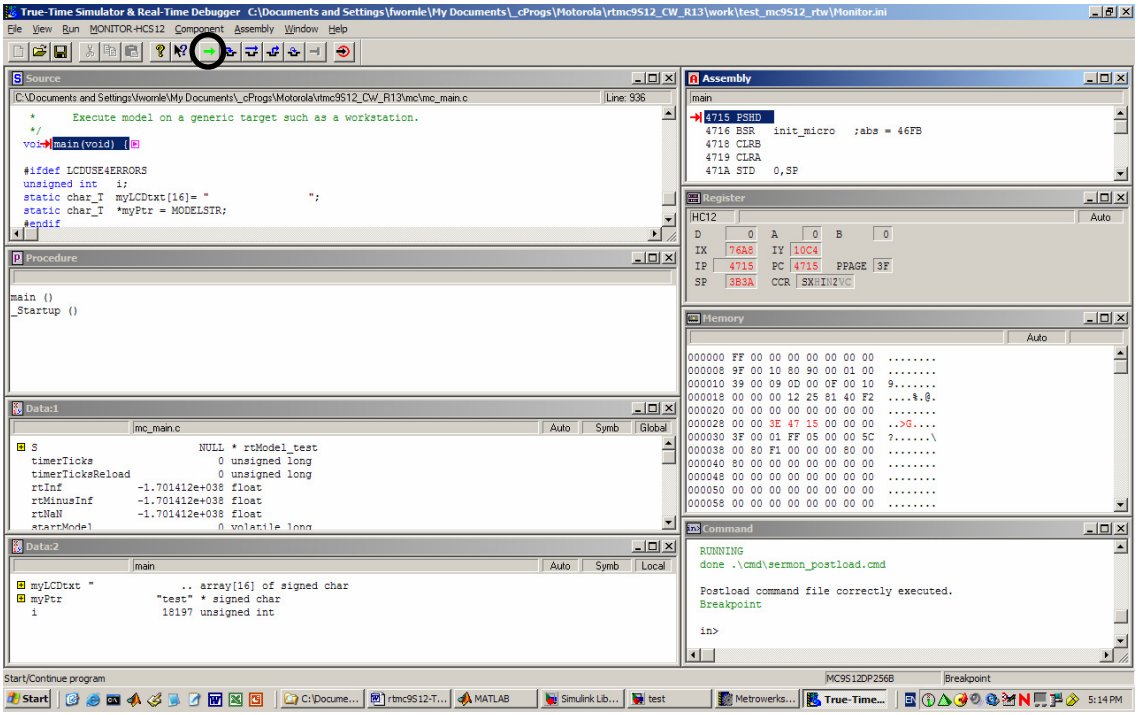


Figure 3-9 Launching the executable target code using the Hi-Ware debugger

Once the target code is running, the Hi-Ware debugger is not needed anymore and can therefore be closed. The communication interface of rtmc9S12-Target is listening on the serial line for commands such as Connect and Disconnect, Start and Stop, etc. These commands can be sent to the target using the *External Control* panel. From the Tools menu choose the item *External Mode Control Panel…* (Figure 3-10).

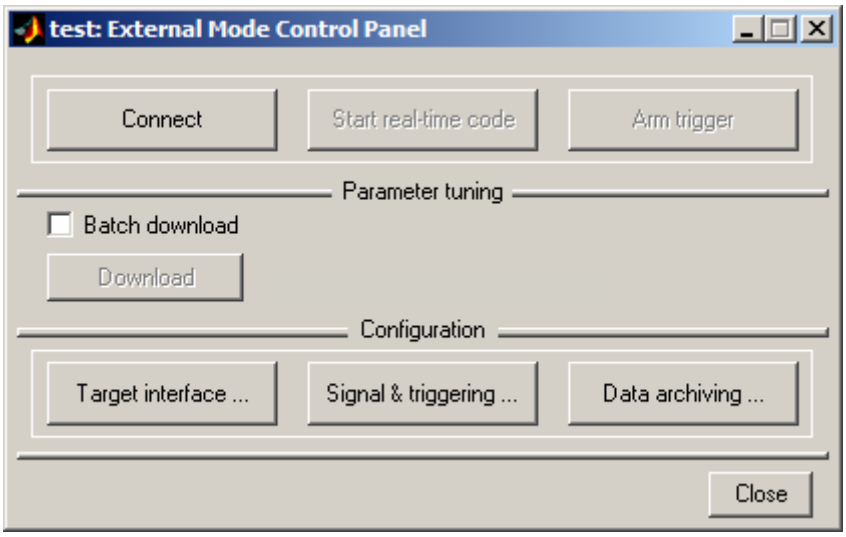


Figure 3-10 External Mode Control Panel

Real-Time Workshop assumes a real-time target to be an autonomous unit to which a user has to *connect* before they can interact with it. Click on the push button *Connect* to initiate the communication between RTW and the target model code. Once connected, the code can be started using the push button *Start real-time code*. Open the Scope block to display the uploaded signal of the Pulse Generator block. A surprising observation is that nothing is displayed. The reason for this is our choice of an extremely large stop time ('9999999') for the target module. Simulink chooses the time axis of all scope blocks to accommodate signals of up to this time value. To change the settings of a scope block the target code execution has to be stopped again.

Click on the push button *Stop real-time code* of the External Mode Control Panel. The real-time code on the target is stopped and RTW disconnects from the target. On the opened Scope window click on the parameters push button (second from the left, Figure 3-11).

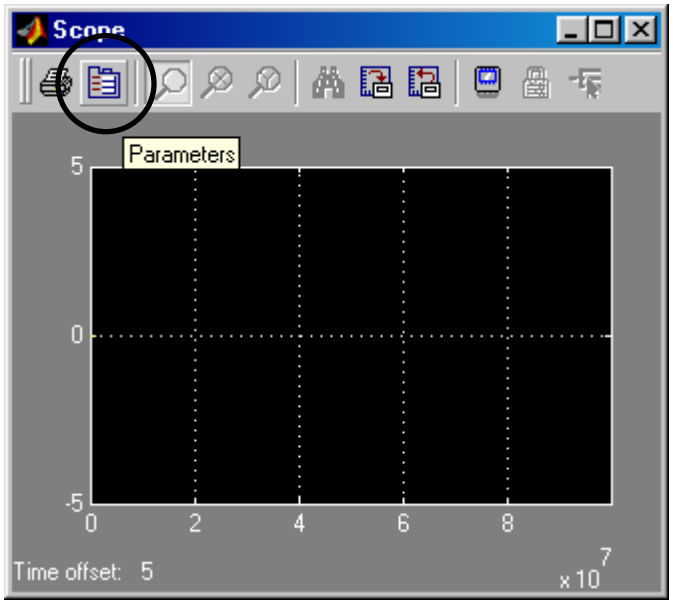


Figure 3-11 Inadequate scope settings

The appearing dialog box allows the scope settings to be customised (Figure 3-12). Replace the automatically chosen *Time range* by a fixed value of, say, 10 seconds. Close the dialog box using the OK push button. The modified scope settings can be saved using the push button *Save current axis settings* (6th button from the left within the toolbar of the scope window).

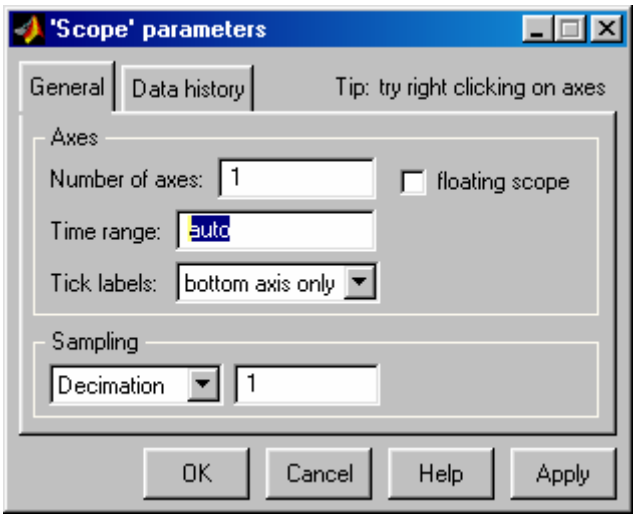


Figure 3-12 Adjusting the scope settings

Re-launch the real-time code (*Connect* then *Start real-time code*). The scope block should now display the trace of the pulse train as produced by the pulse generator. The display block should display a value of 0. Upon starting the host model *test_PC.mdl* the display block should change to the value which is sent by the host. Open the slider gain block and change this value. The target should immediately update to the new value.

An interesting observation is that the trace displayed by the scope block appears to show slight irregularities (Figure 3-13, pulse beginning at 44 seconds). This has to do with the fact that the pulse generator block is running in "continuous time", i. e. at the base rate of the target model (5 ms). The produced amount of log data to be uploaded to the host is unnecessarily high: The minimum width of a log data channel is 36 bytes. The requested 200 uploads per second thus require a total amount of 7200 bytes to be uploaded. The chosen communication speed of 115200 bps yields an average data rate of 115200 bits per seconds / 10 bits per data value, i. e. approximately 11520 data values per second. This shows that the serial interface cannot cope with the amount of data to be uploaded.

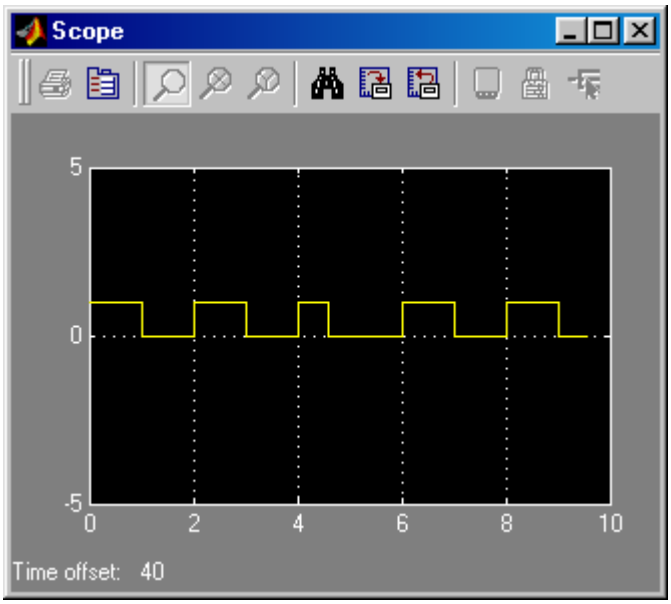


Figure 3-13 Irregularities within the uploaded log data

The volume of log data can be reduced using *Zero-Order Hold (ZOH)* blocks. These blocks achieve a down-sampling from a high rate to a lower rate. From the Simulink library category *Discrete* drag a ZOH into the target model block diagram. Insert this block into the signal path between Pulse Generator and Scope block. Open the ZOH and choose an upload rate of 20 values per seconds (sample time: 0.05 seconds). Figure 3-14 shows the modified target block diagram.

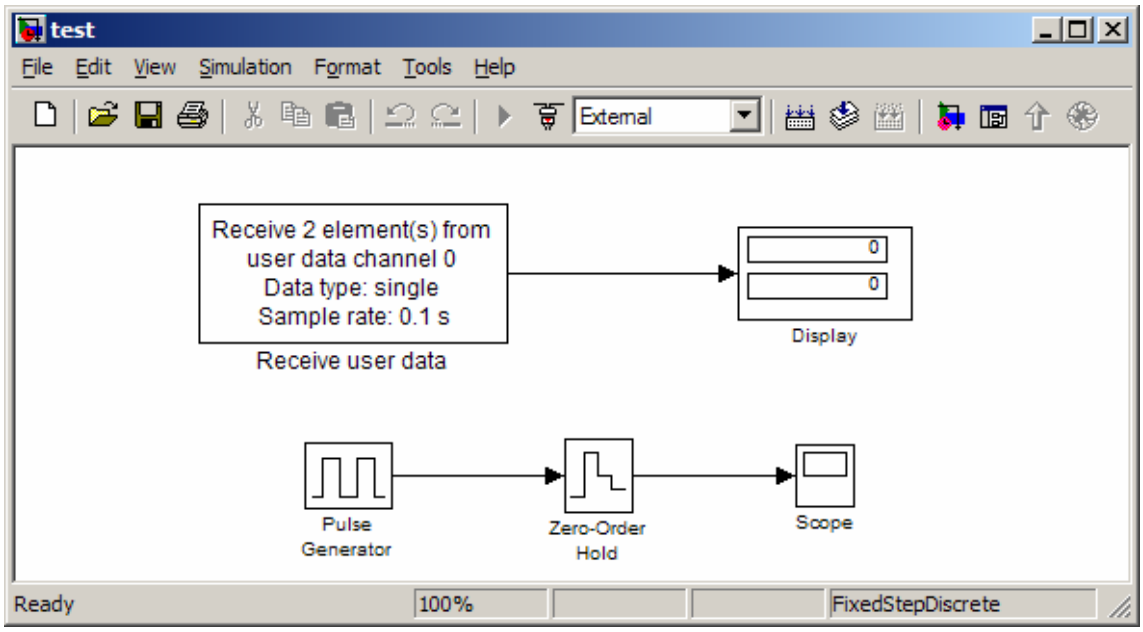


Figure 3-14 Modified target block diagram: Reduced upload rate

Save the block diagram and re-build the target code (CTRL-B). Once downloaded to the target, the executable can be started; the scope block should now display a symmetric pulse train with a period of 2 seconds and a 50% duty cycle (Figure 3-15).

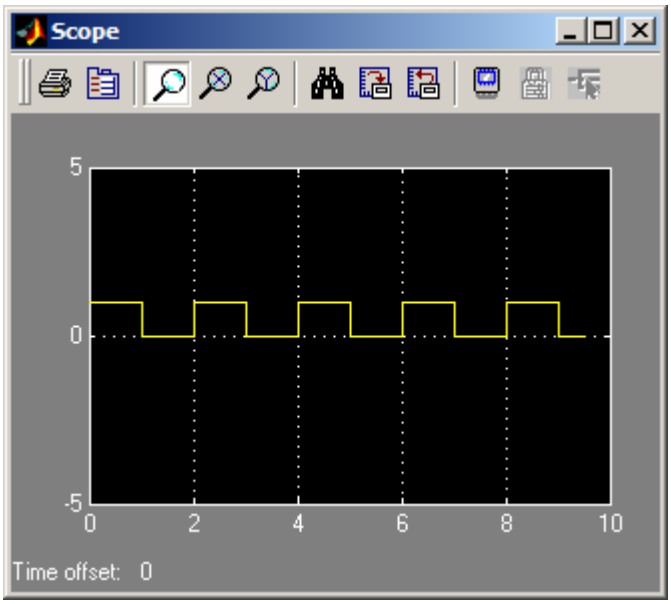


Figure 3-15 Corrected block diagram: Symmetric pulse train

Note that you may have to change the Scope trigger point to match the chosen time axis range. At a sample rate of 0.002 seconds it takes 5000 samples to reach 10 seconds. From the *External Mode Control* panel (Figure 3-10) choose *Signal & Triggering*. Set the Trigger *Duration* to 5000 (Figure 3-16). Close this window and restart the target code. Note that this does not require the code to be rebuilt, as the modified simulation parameters are downloaded upon connecting to the target.

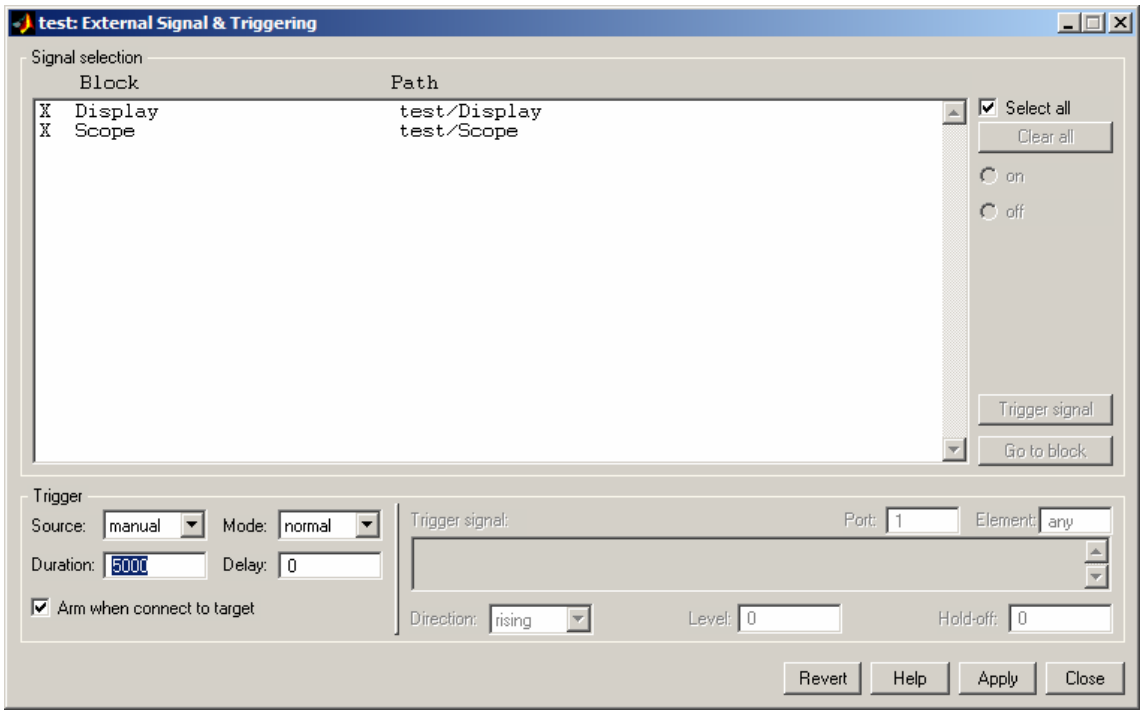


Figure 3-16 Configuring the Upload signal trigger

Most controller parameters can be changed while the controller is running (on-the-fly tuning). As an example, change the amplitude of the Pulse Generator block. The Scope block should immediately reflect the requested change. Figure 3-17 shows a jump from an amplitude value of '1' to the new amplitude '2'.

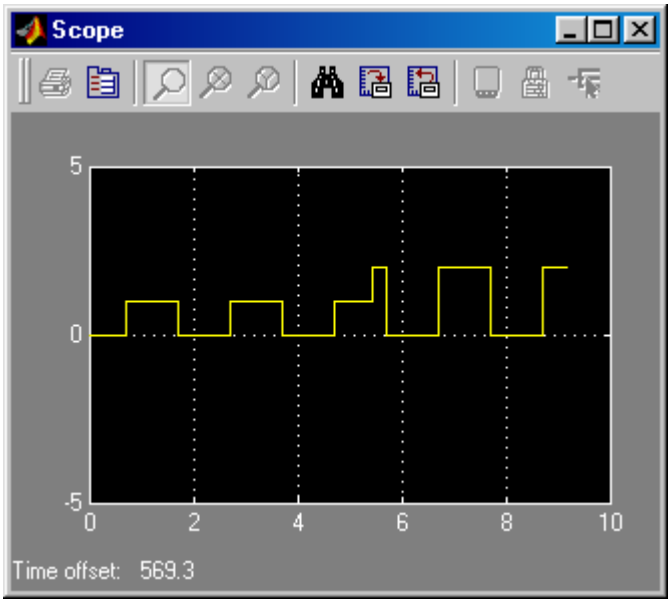


Figure 3-17 On-the-fly changes of model parameters: Doubling the amplitude

A further way to control the data logging process is the push button *Cancel trigger* on the *External Mode Control Panel*. Hitting this button stops all upload of log data; the label of the *Cancel trigger* push button changes to *Arm trigger*. Notice that the controller continues to execute the real-time code and host and target remain connected. The effect of *Cancel trigger* is best compared to that of a *pause* button. To resume data logging click on *Arm trigger* (Figure 3-18).

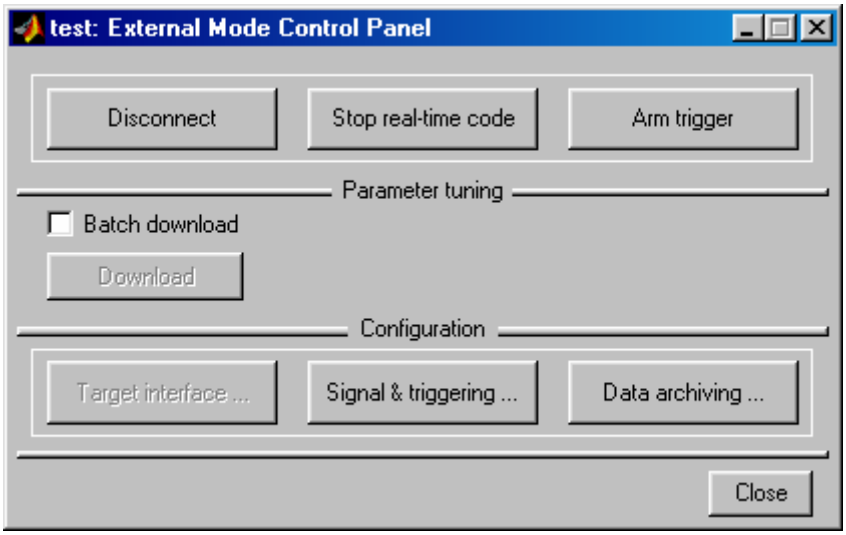


Figure 3-18 Cancel trigger and Arm trigger

Furthermore, it is possible to *disconnect* from the target while keeping it running. This is useful when an experiment runs for a long time and does not require permanent supervision. Once disconnected, the host computer can be switched off and back on without disturbing the currently running program on the microcontroller. To resume control simply *reconnect*.

Remark:

As it can never be guaranteed that the serial link between host and target remains active all the time, a host driven flow control has been implemented. Typical problems occur when a user decides to launch a new application, to move or resize a window, to open a menu, etc. Any of these events can interrupt the regular upload of log data for an unspecified duration. rtmc9S12-Target takes care of this uncertainty by specifically requesting the upload of every single log data telegram. The target thus stops sending log data when the host is busy; regular operation resumes as soon as Simulink continues to operate normally.

4 The example models

A set of sample models has been included to facilitate familiarisation with this toolbox. They can all be found in the work directory <rtmc9S12_TARGET_ROOT>\work.

4.1 AD_9S12.mdl

The model *AD_9S12.mdl* implements a 3-channel A/D Conversion with real-time data logging. A/D channels 2, 3 and 4 are read and converted every 2 ms. Data upload is performed every 20 ms. As can be seen from the status bar (Figure 4-1), the model has been configured to use an ODE solver (ODE1, Euler method). Note that this is not really necessary here, as the model does not feature any continuous states.

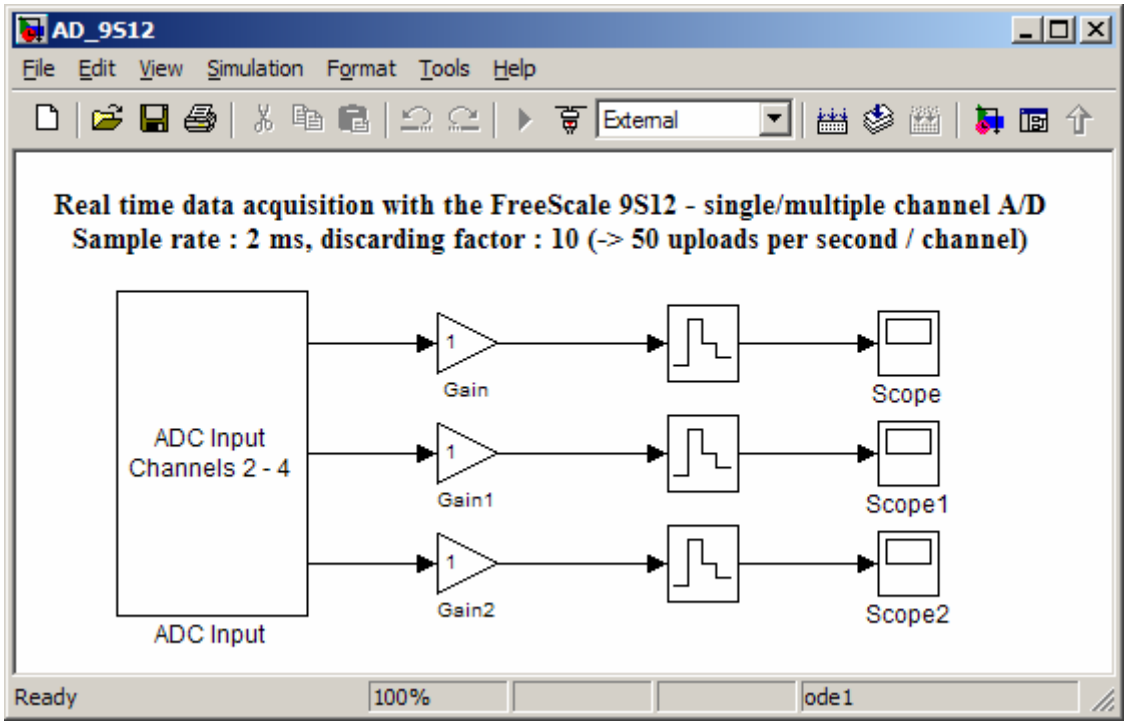


Figure 4-1 Sample model: AD_9S12.mdl

4.2 ADC_DAC.mdl

Model *ADC_DAC.mdl* reads ADC channel 2 (100 times per second) and multiplies the converted value with a sine wave. The output of this operation is then sent to the D/A Converter (DAC, channel 0). A saturation block has been introduced to limit the DAC input values to the valid range (0 … 5). Note that this model does not use any data upload blocks. The *External Mode* could therefore be switched off to reduce the size of the target executable to approximately 1/3 of its current size.

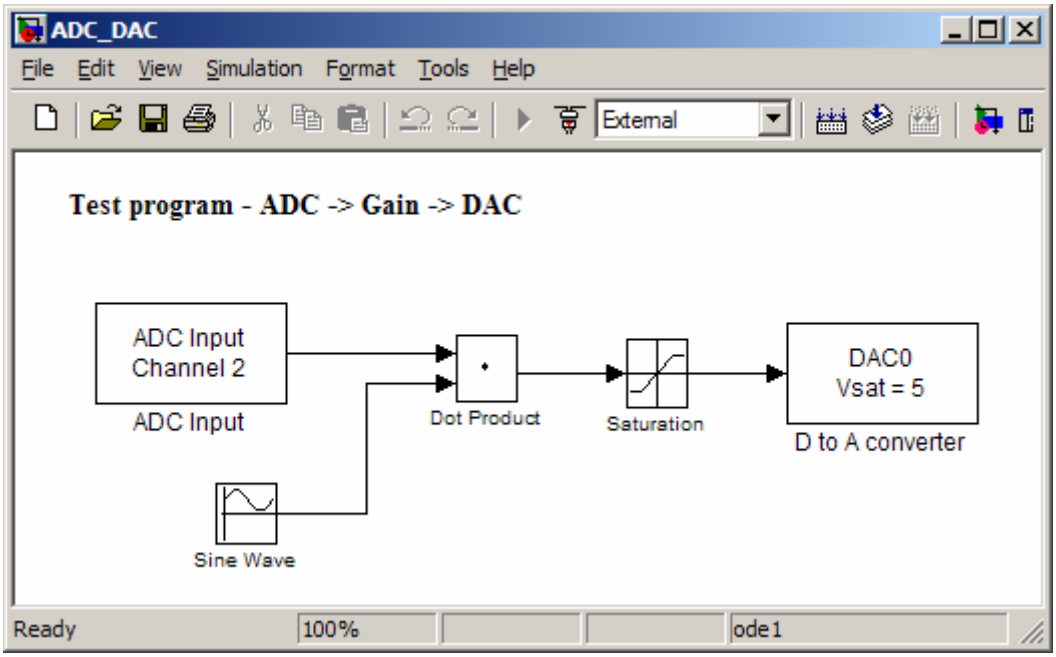


Figure 4-2 Sample model: ADC_DAC.mdl

4.3 DigINPort.mdl

The model *DigINPort.mdl* allows monitoring of the state of digital input lines 0, 1 and 2 of port PTH. The upload of log data is performed every 20 ms.

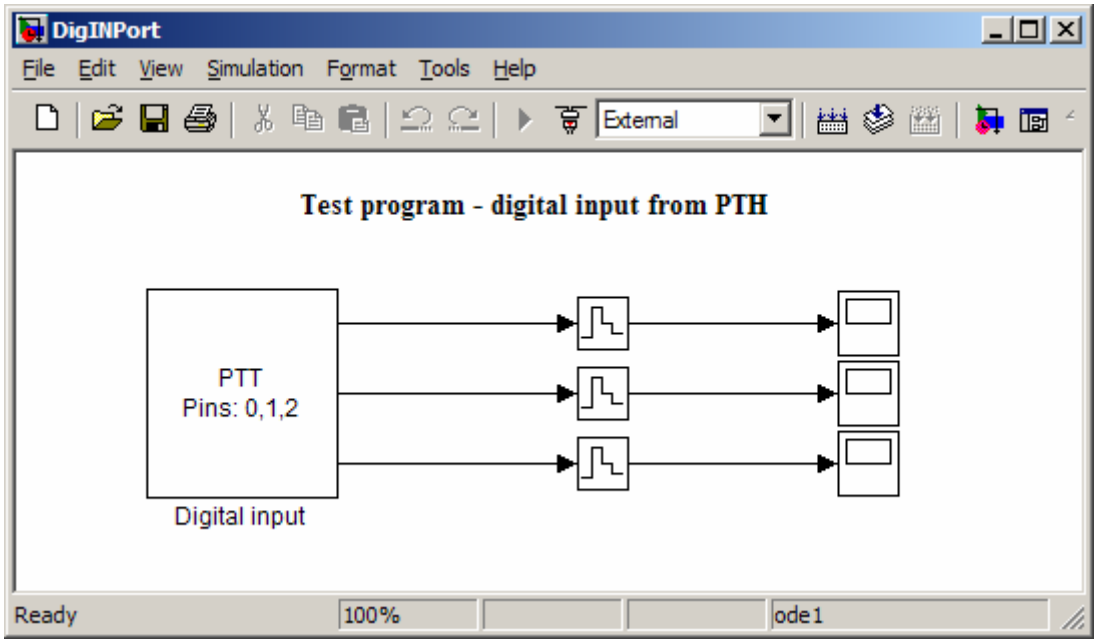


Figure 4-3 Sample model: DigINPort.mdl

4.4 DigOUTPort.mdl

The model *DigOUTPort.mdl* allows the control of digital output lines. Here, pins 2 and 6 of port PTT can be controlled using a slider gain. Gain parameter download is performed 'continuously' (sample time 0), whereas the upload (Display) has been limited to 10 times per second. Notice that the outputs switch 'high' when the slider gain exceeds the On-Threshold (Von = 3.5); the are switched to 'low' when the slider gain is set to values below the Off-Threshold (Voff = 2). Both thresholds can be set via the block parameter dialog box.

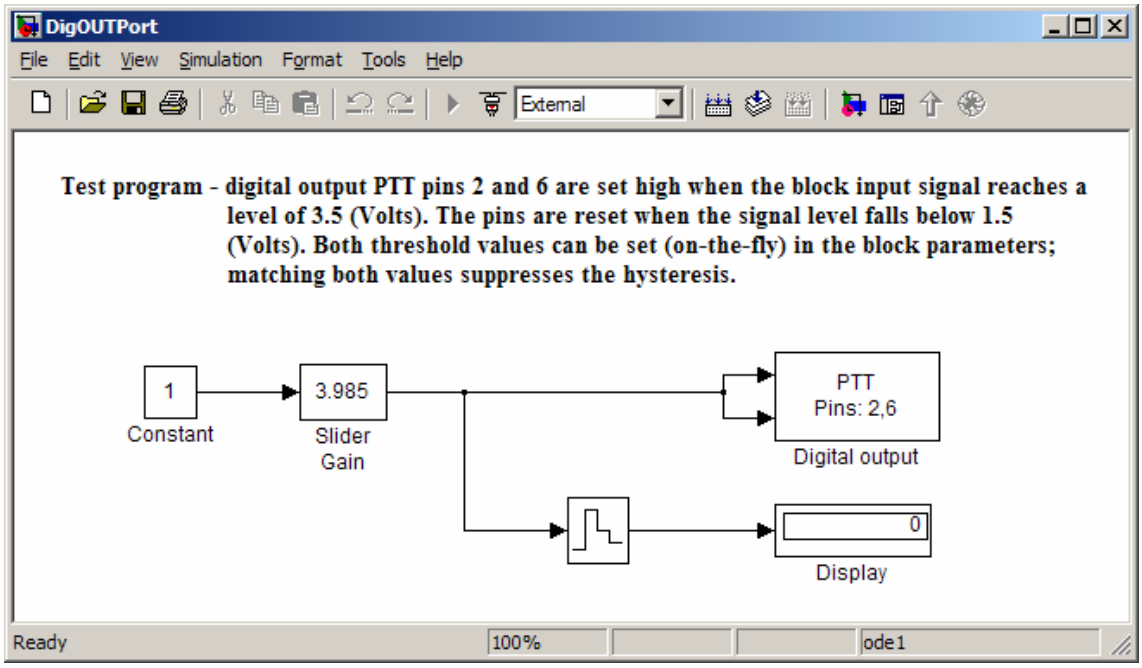


Figure 4-4 Sample model: DigOUTPort.mdl

4.5 Pulse-Width Modulation (PWM)

Model *PWM.mdl* produces a host driven pulse width modulated signal with a period of 100 ms. In conjunction with a suitable amplifier (e.g. a H-bridge amplifier, stepper motor driver), the PWM module can be used to implement a large variety of drive systems (DC motors, servo motors, stepper motors, etc.). Forward / reverse information is derived from the polarity of the block input signal and can be accessed on the configured *sign pin*. The example uses PORTB, pin 1 to display the sign information. Port PTP is connected to the PWM unit of the microcontroller. Each channel has been assigned a pin of port PTP: Channel 0 corresponds to PTP.0, channel 1 is PTP.1 and so on. A cascaded 16-bit PWM channel ties up two 8-bit units. The generated pulse train can then be read from the 'upper' of the two associated pins, i. e. PTP.1 for cascaded unit '0 & 1', PTP.3 for cascaded unit '2 & 3', etc. Block inputs above the Saturation level (here: 5) are clipped and result in a 100% duty cycle. The optional block output has been enabled (Signal monitoring). However, it should be noted that this feature is only useful when the block sample time is much shorter than the PWM period – here we work with a sample time of 10 ms and a pulse period of 100 ms. This means that the resolution of the monitored PWM signal is not too high.

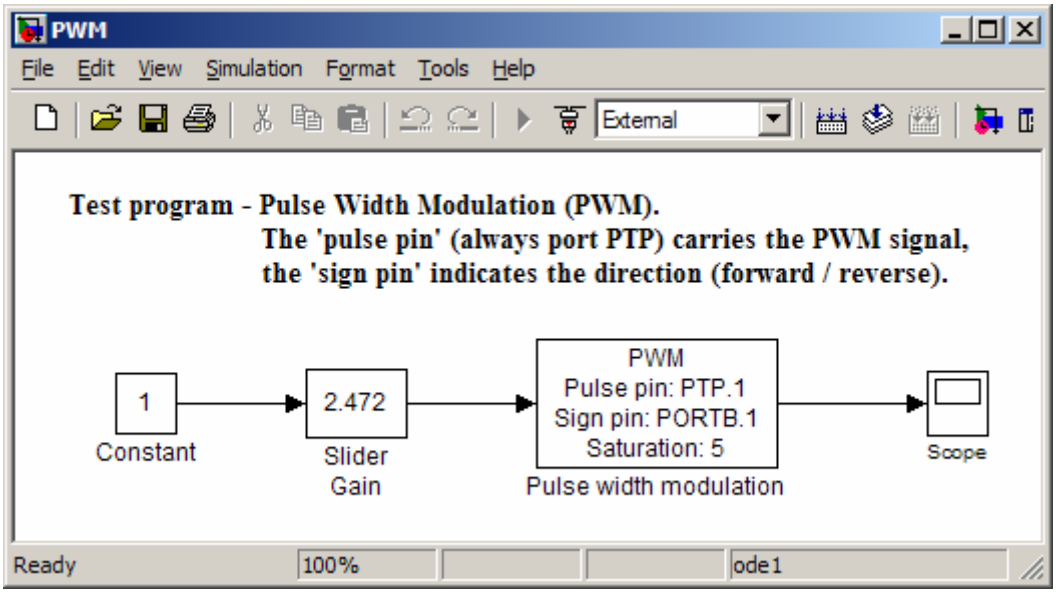


Figure 4-5 Sample model: PWM.mdl

4.6 Pulse-Width Modulation (PWM) with user communications

Model *PWM2.mdl* (Figure 4-6) and the corresponding host model *PWM2_PC.mdl* (Figure 4-7) demonstrate how easy it is to setup a remote-controlled robot control system.

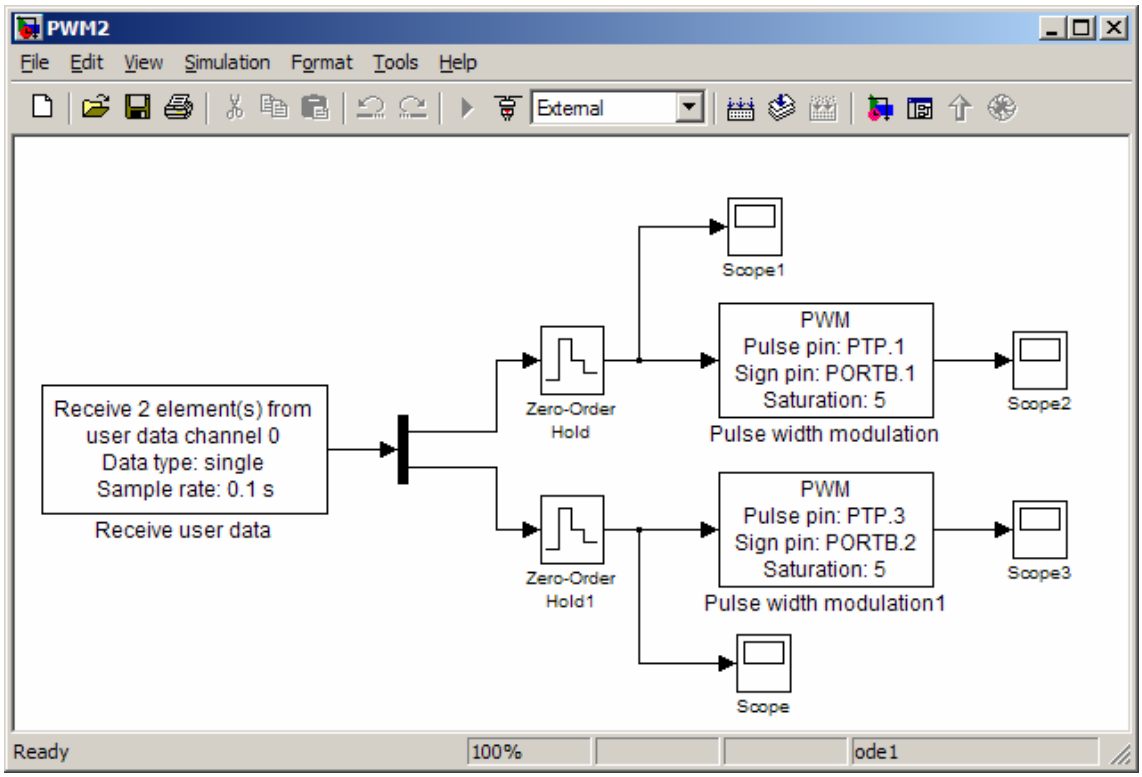


Figure 4-6 Sample model: PWM2.mdl

The target model (PWM2.mdl) has two PWM output channels (16-bit resolution, pulse pins PTP.1 and PTP.3). For demonstration purposes, the PWM period has been set to a rather long interval of 0.5 s. Background monitoring is performed at a rate of 10 times per second. The duty cycle of both channels is controlled by a signal which is sent from the host through a user communication download block ('Receive user data'). The host-sided end of this communication channel is a 'Send user data' block in a second, independent Simulink model (PWM2_PC.mdl). The latter runs on the host in normal simulation mode. Changing the values of either of the slider gains initiates a download through the associated user communication channel. The host sided model can be started / stopped / modified at all times. It is truly independent of the target model.

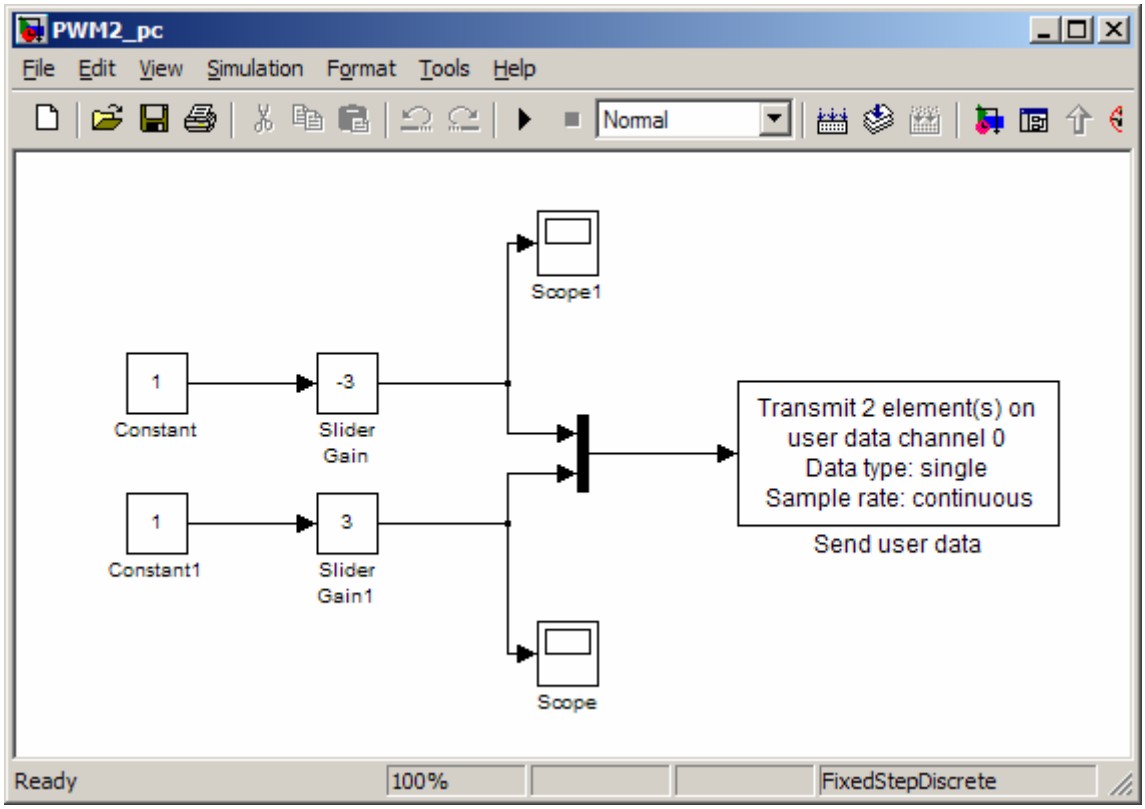


Figure 4-7 Host-end of the user communication: PWM2_pc.mdl

The use of the optional *monitoring output* of the *Pulse width modulation* block is shown in Figure 4-8. The associated Scope blocks visualize the signals generated by the two PWM units. Note that Figure 4-8 also shows the state of sign bit – this feature has not been implemented on the 9S12 (worked fine in R12, but there seemed to be a problem with this in R13).

The monitoring output can be enabled or disabled using a check-box in the block parameter page (Figure 4-9).

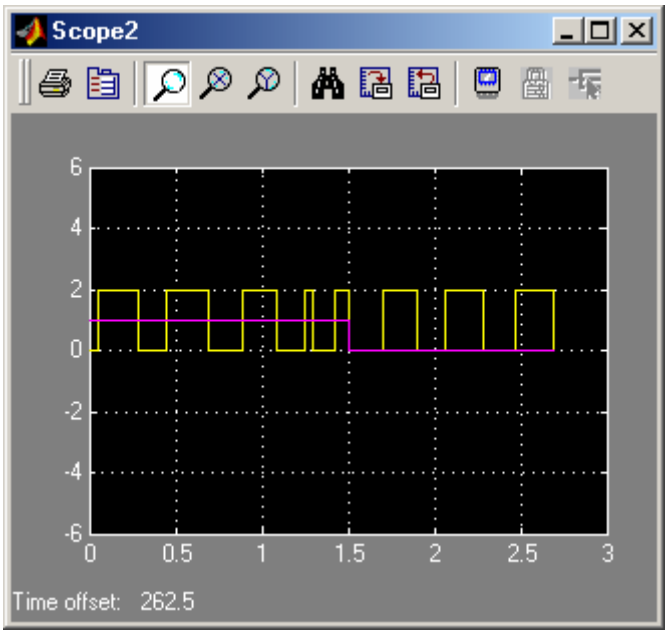


Figure 4-8 Pulse width modulation: monitoring the generated PWM signals

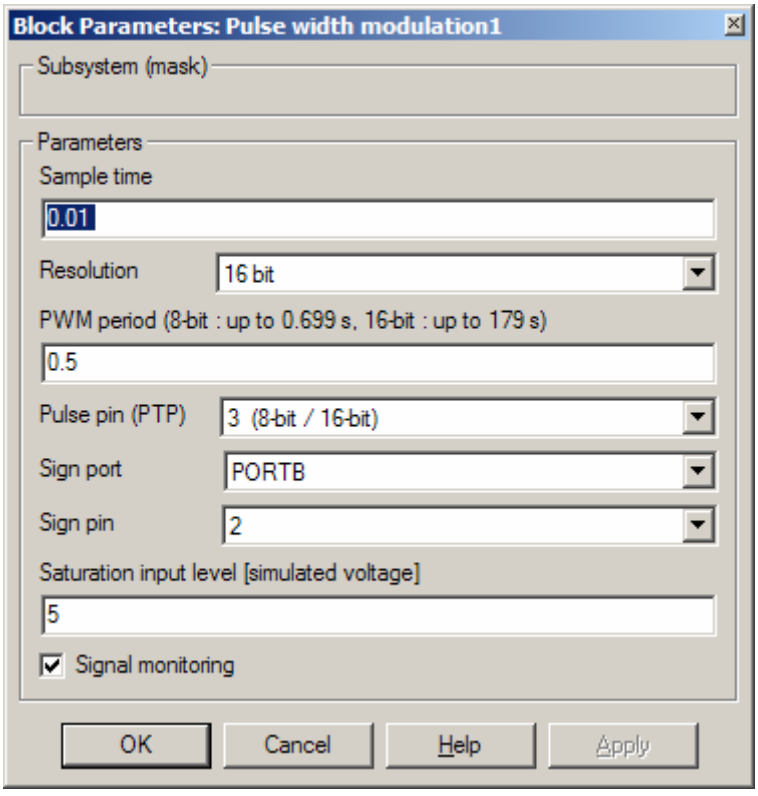


Figure 4-9 Block parameter page: Pulse width modulation

4.7 User communication

The models of this section demonstrate how to implement a user communication between a separate host model (running in *normal* mode) and the target model

(running in *external* mode). All examples thus require a pair of models, the target model *<my_target_name>.mdl* and a corresponding host model, here referred to as *<my_target_name>_pc.mdl*. Host and target model communicate via shared memory pointers which are exchanged using the workspace variable *BufferAdminVar*.

Note: Due to the memory restriction and the basic interrupt handling of the 9S12, the user communication blocks do not work as reliably as on the C167. Communication problems can arise, in particular when a target is asked to do too many things at the same time…

fw-03-05

4.7.1 Download of user telegrams

The target model *USRCom_dwnld.mdl* and the corresponding host model *USRCom_dwnld_pc.mdl* can be used to investigate the capabilities and limits of the automatic download of user data. This target model produces a sine wave of 10 rad/s, which is uploaded to the host as log data using the external communication interface. Two *Receive user data* blocks listen for incoming user data on channel 0 and channel 1. These blocks are updated every 0.1 seconds. New data values are output to the display blocks which can be monitored using the external communication interface. It is thus possible to send customised user telegrams to the target, where they could be interpreted as commands. See the Simulink library *Sub-systems* for ideas (Switch case, If-action sub-system, etc.).

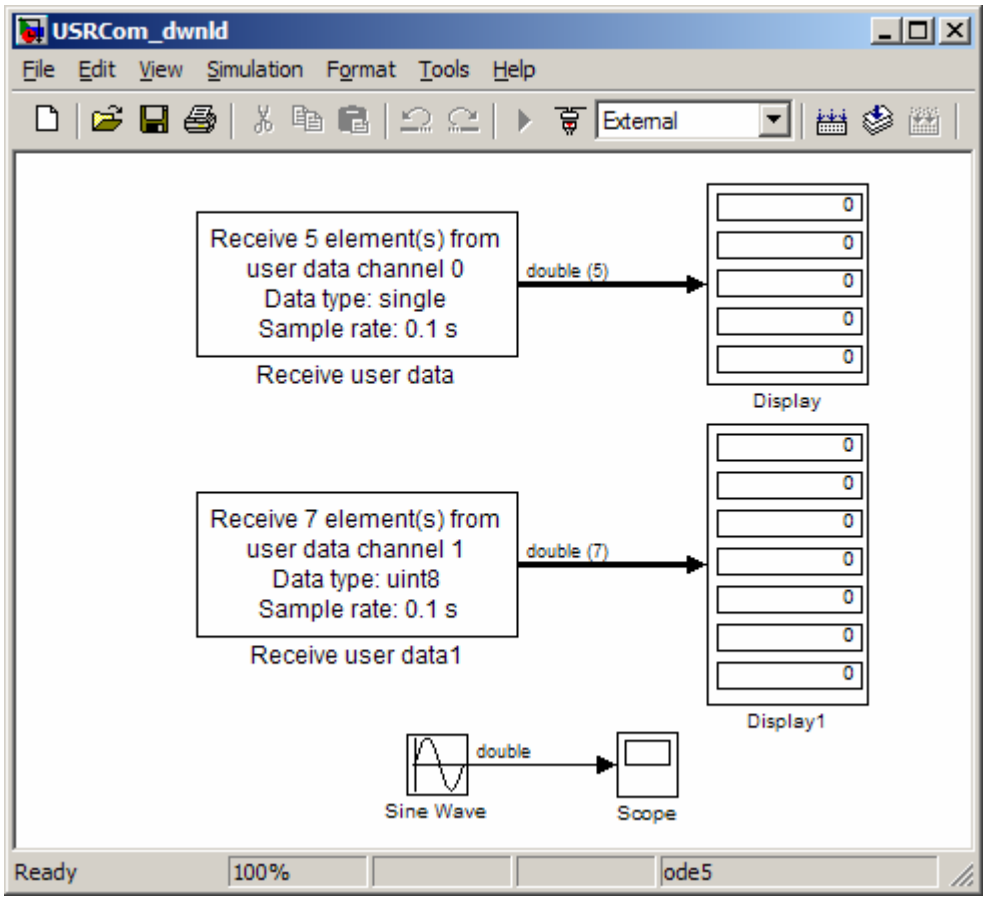


Figure 4-10 Sample model: USRCom_dwnld.mdl (target)

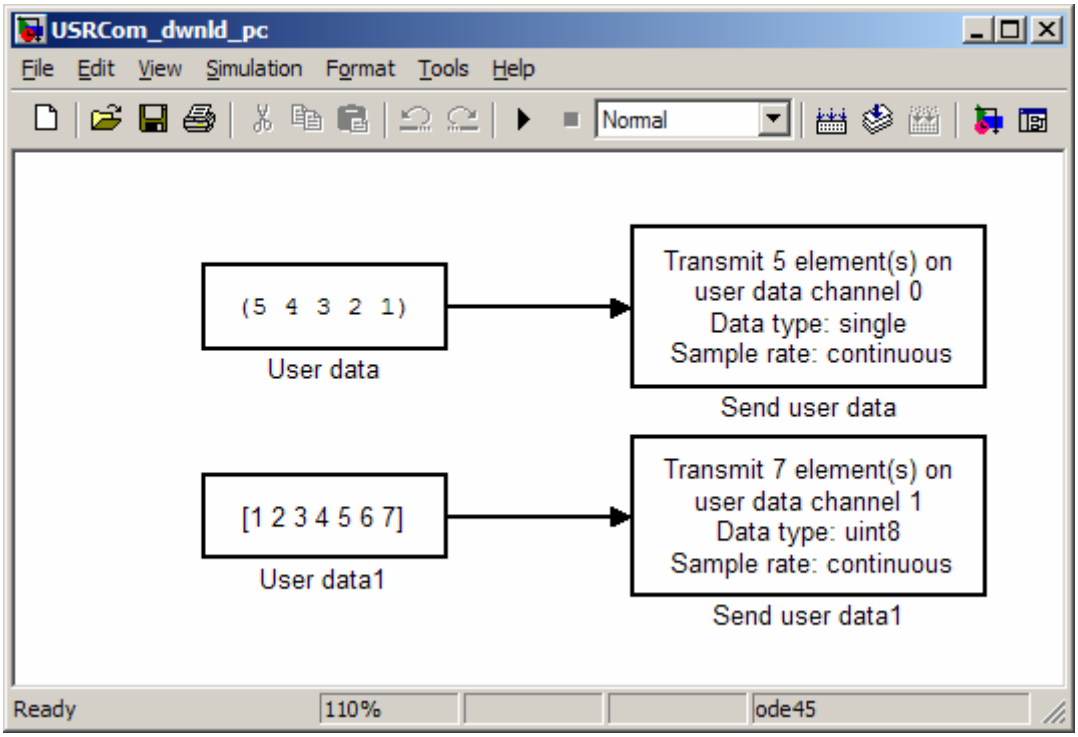


Figure 4-11 Sample model: USRCom_dwnld_pc.mdl (host)

4.7.2 Upload of user telegrams

The target model *USRCom_upld.mdl* and the corresponding host model *USRCom_upld_pc.mdl* can be used to study an upload of externally generated user data. The target model produces a sine wave of 10 rad/s, which is uploaded to the host as log data using the external communication interface. Two *Send user data* blocks are the source of two additional upload channels for user data: Channel 0 uploads 5 elements of data type *single* (float) whereas channel 1 has a data width of 7 unsigned bytes (UINT8). Both blocks are updated every 0.1 seconds. New data sets can be programmed through the preceding *Constant* blocks. RTW treats these changes as on-the-fly parameter downloads. Once the new values are detected at the input of a *Send user data* block, the block initiates the upload of a user telegram. On the host side the model *USRCom_upld_pc.mdl* receives the uploaded user data values and displays them on the associated *Display* blocks.

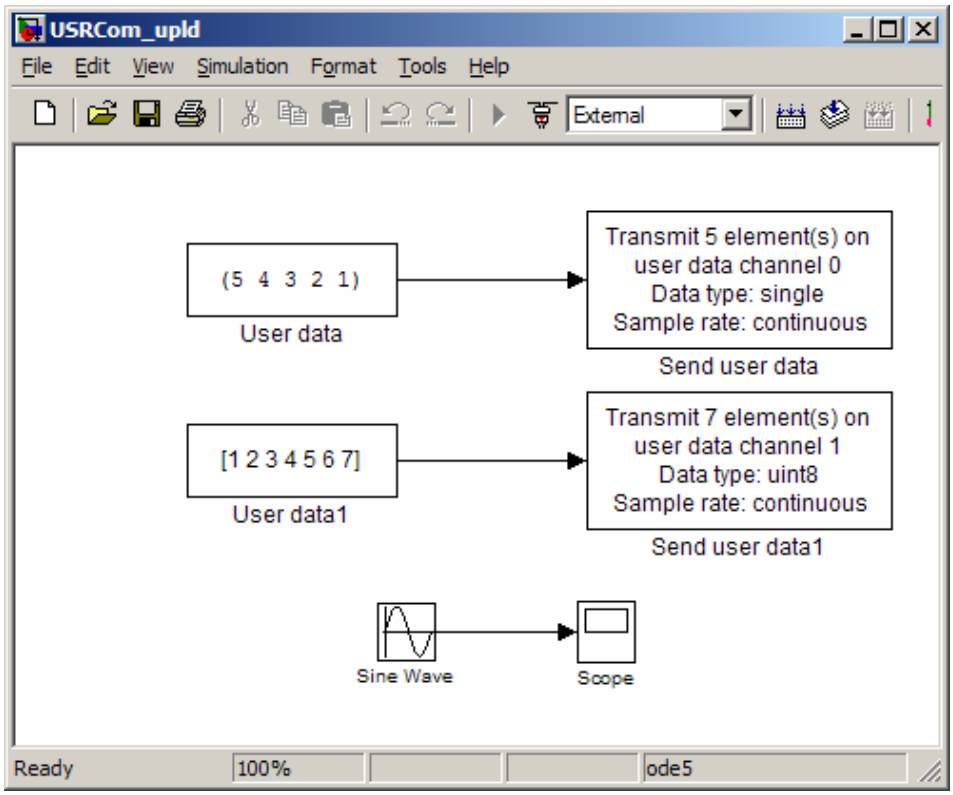


Figure 4-12 Sample model: USRCom_upld.mdl (target)

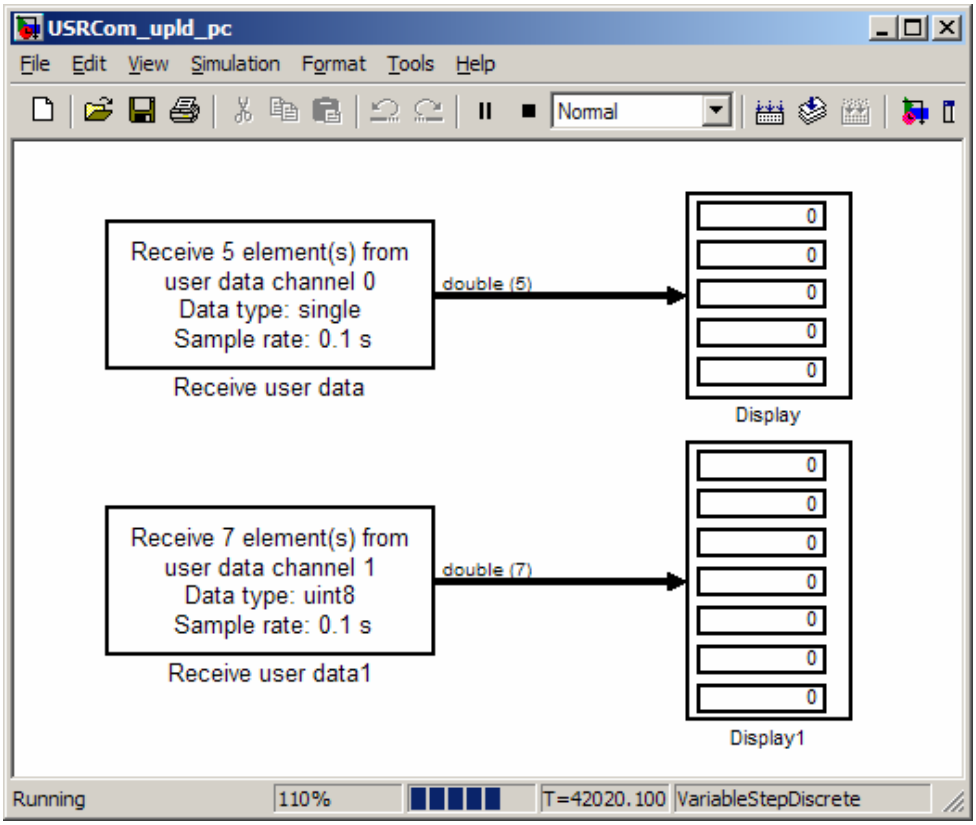


Figure 4-12 Sample model: USRCom_upld_pc.mdl (host)

4.7.3 Upload and download of user telegrams

The target model *USRCom_upndwn.mdl* and the corresponding host model *USRCom_upndwn_pc.mdl* can be used to experiment with the simultaneous upload and download of user data. The target model produces a sine wave of 10 rad/s, which is uploaded to the host as log data using the external communication interface. In addition, the user communication block *Send user data* uploads 5 elements of type *single* (float) via channel 0. This upload is serviced every 0.1 seconds. However, a user telegram is only sent whenever the block input has changed. The second user channel makes use of a *Receive user data* block to await additional user telegrams which may be downloaded from the host. The block specifies a sample time of 0.1 seconds and a channel width of 7 bytes (UINT8).

Note: This does not work very well on the 9S12.

fw-03-05

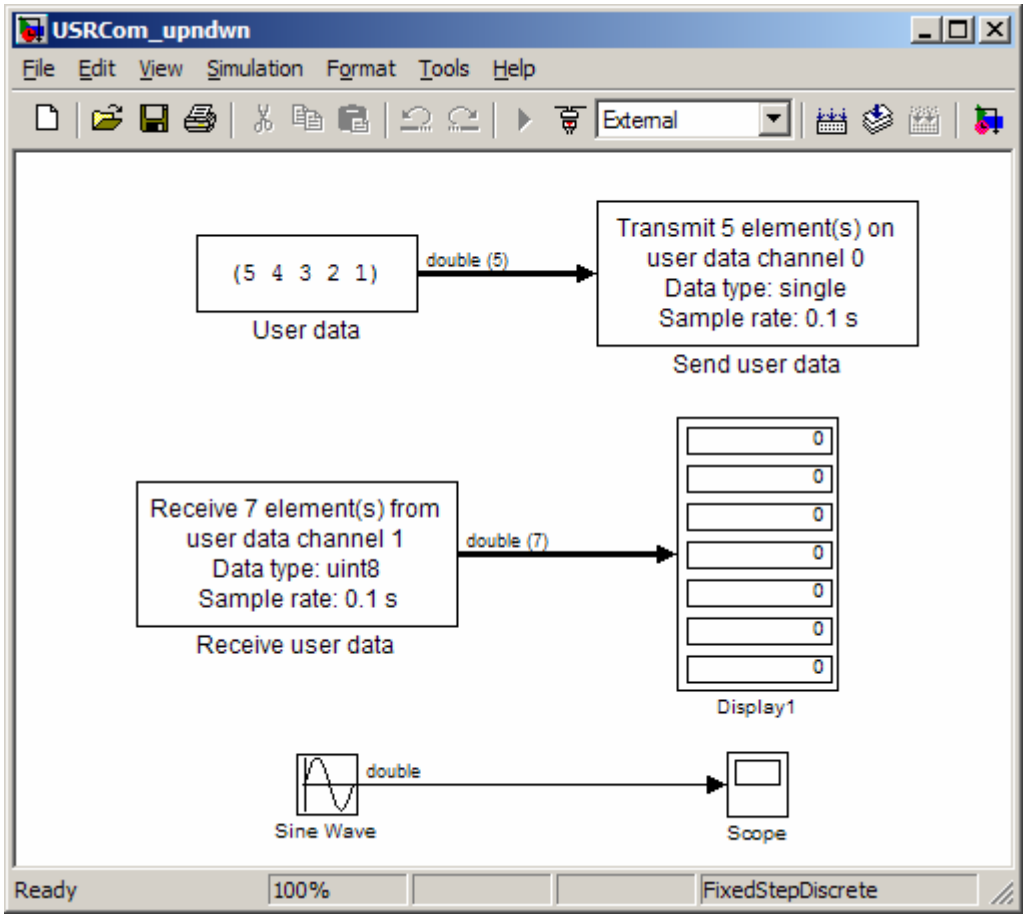


Figure 4-13 Sample model: USRCom_upndwn.mdl (target)

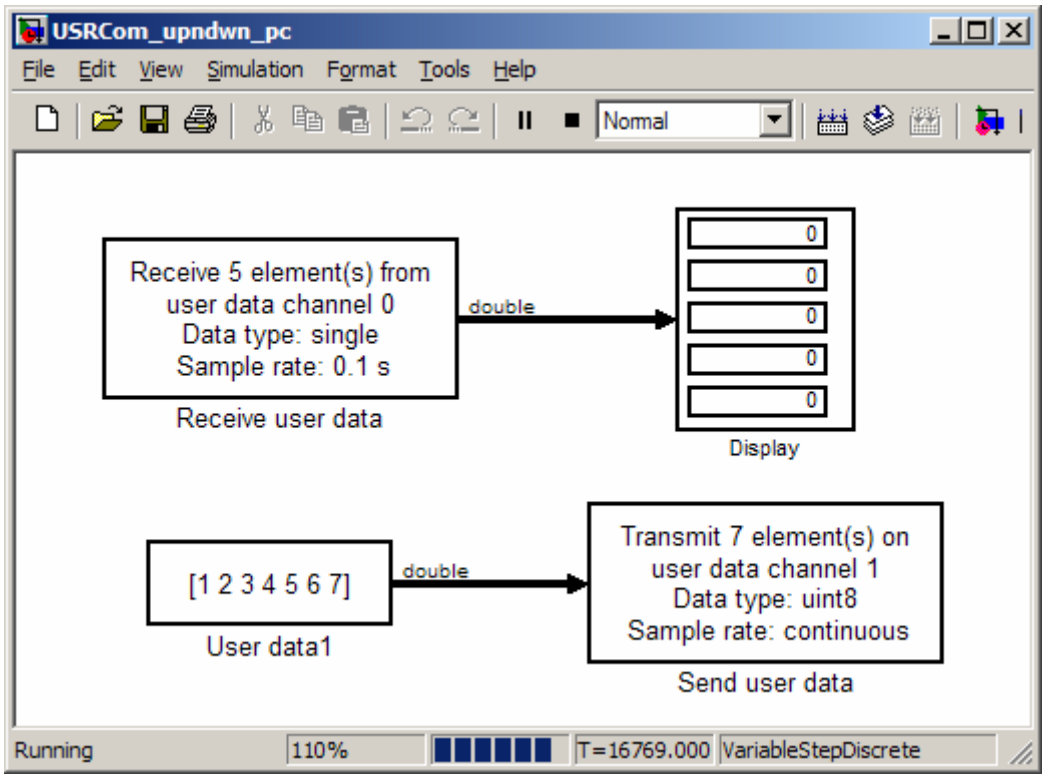


Figure 4-14 Sample model: USRCom_upndwn_pc.mdl (host)

4.7.4 Heavy download of user telegrams

The target model *USRCom_dwnld_heavy.mdl* and the corresponding host model *USRCom_dwnld_heavy_pc.mdl* can be used to investigate the limits of the automatic download of user data. The target model produces a sine wave of 10 rad/s, which is uploaded to the host as log data using the external communication interface. In addition, a *Receive user data* block listen for incoming user data on channel 0 (data channel width: 2 singles, block update rate 10 Hz). However, unlike the example shown in section 4.6.1, this model is virtually bombarded with user telegrams. The host model *USRCom_dwnld_heavy_pc.mdl* permanently updates the block input of the corresponding *Send user data* block. The target thus reads a new value every time the *Receive user data* block is visited by the real-time kernel. By displaying the received user data on a regular scope block, the time delay of the serial download and subsequent upload can be measured.

Note: This does not work very well on the 9S12.

fw-03-05

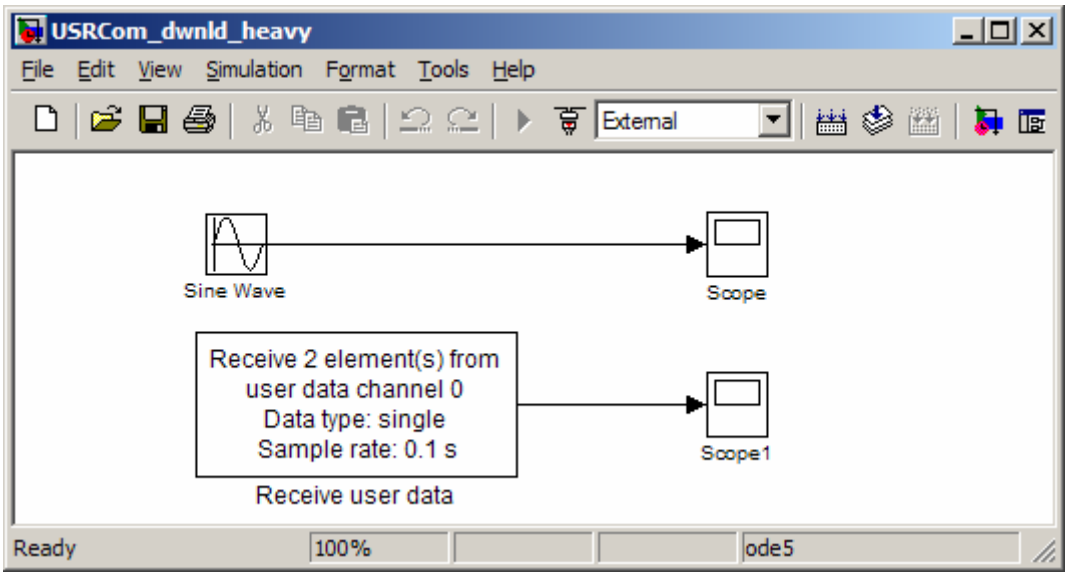


Figure 4-15 Sample model: USRCom_dwnld_heavy.mdl (target)

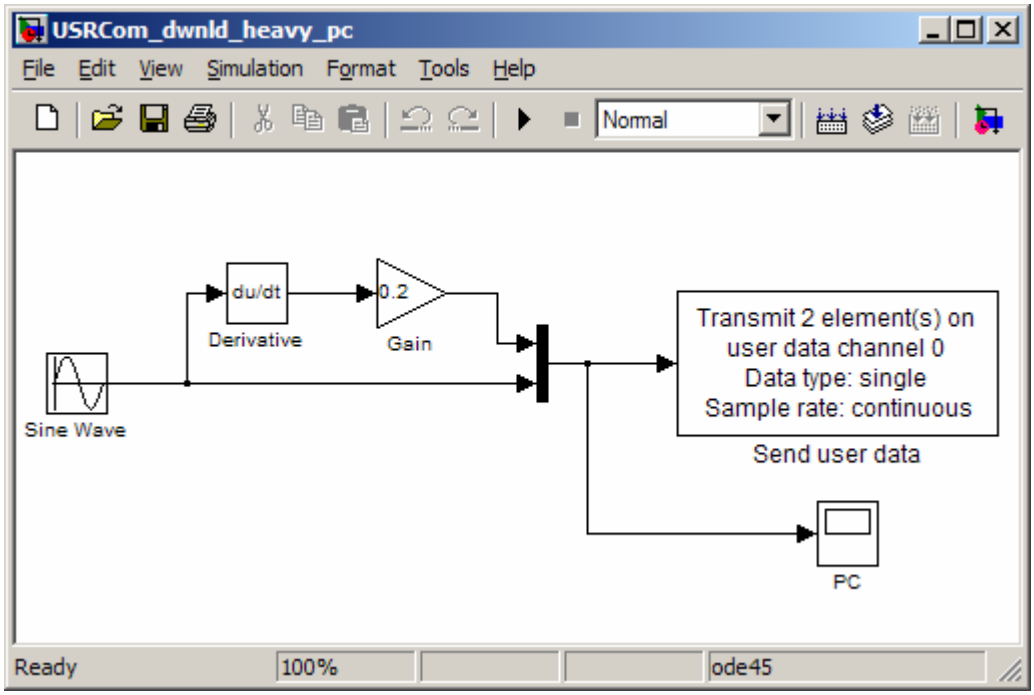


Figure 4-16 Sample model: USRCom_dwnld_heavy_pc.mdl (host)

4.8 FreePort communications

4.8.1 Simple download of data to the target

The target model *FreePortComm_RX_simple.mdl* and the corresponding host model *FreePortComm_TX_simple.mdl* demonstrate the use of the *FreePort* communication blocks for data download from the host to the target. The *FreePort* communication

interface does not rely on the External Mode and can therefore be used with small standalone programs as well as in parallel with optional External Mode communications via the other port. The sample target model receives 5 values on its SCI0 port and feeds these values to an output block for PTH. The latter block has been configured to switch a line high whenever the incoming signal exceeds 3.5. A port line is switched low when the input falls below 2. Port SCI0 is the free communication port on the Dragon-12 when using External Mode. Without External Mode communications, both ports are free. The display shown in Figure 4-17 only shows the incoming data when using External Mode communications. Figure 4-18 is the corresponding host sided block diagram.

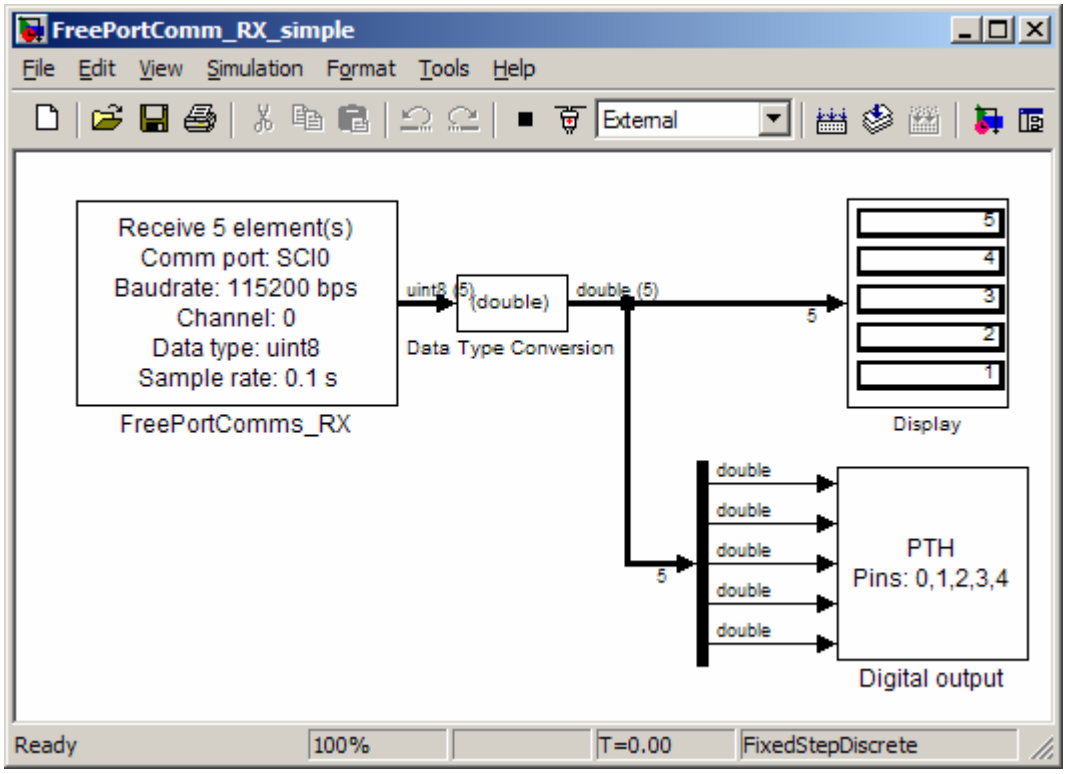


Figure 4-17 Sample model: FreePortComm_RX_simple.mdl (target)

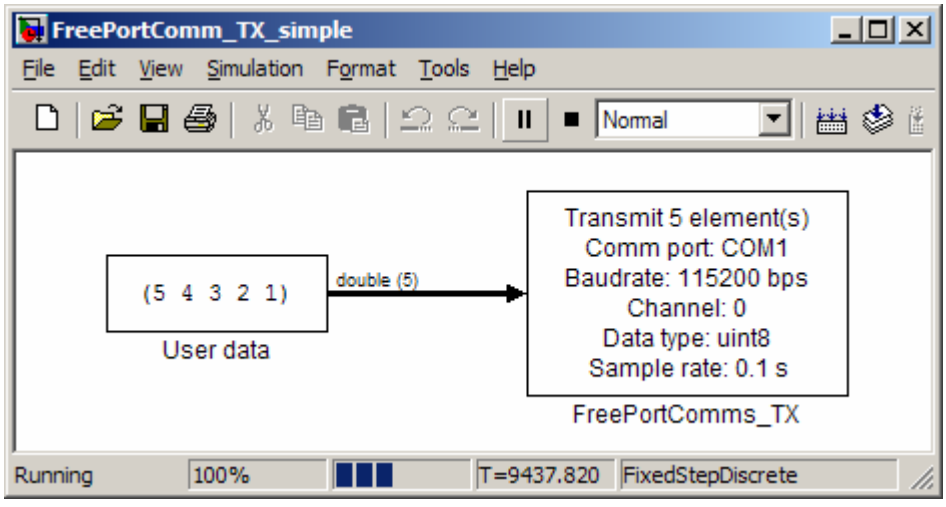


Figure 4-18 Sample model: FreePortComm_TX_simple.mdl (host)

4.8.2 Simple upload of data from the target

The target model *FreePortComm_TX_simple2.mdl* and the corresponding host model *FreePortComm_RX_simple2.mdl* demonstrate the use of the *FreePort* communication blocks for data upload from the target to the host. The target model reads the dip switches connected to port H (PTH) and uploads this information (‘0’ or ‘1’) to the host. The optional (target sided) display block is only serviced when running in External Mode. Figure 4-20 shows the corresponding host model.

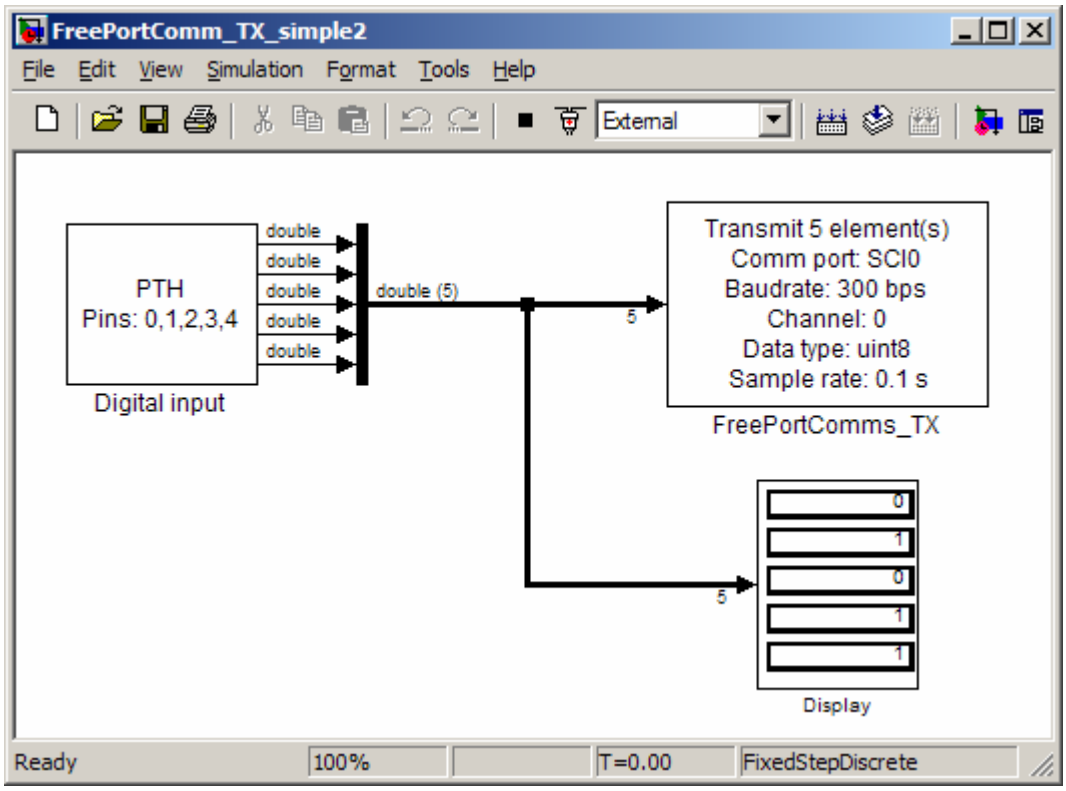


Figure 4-19 Sample model: FreePortComm_RX_simple2.mdl (target)

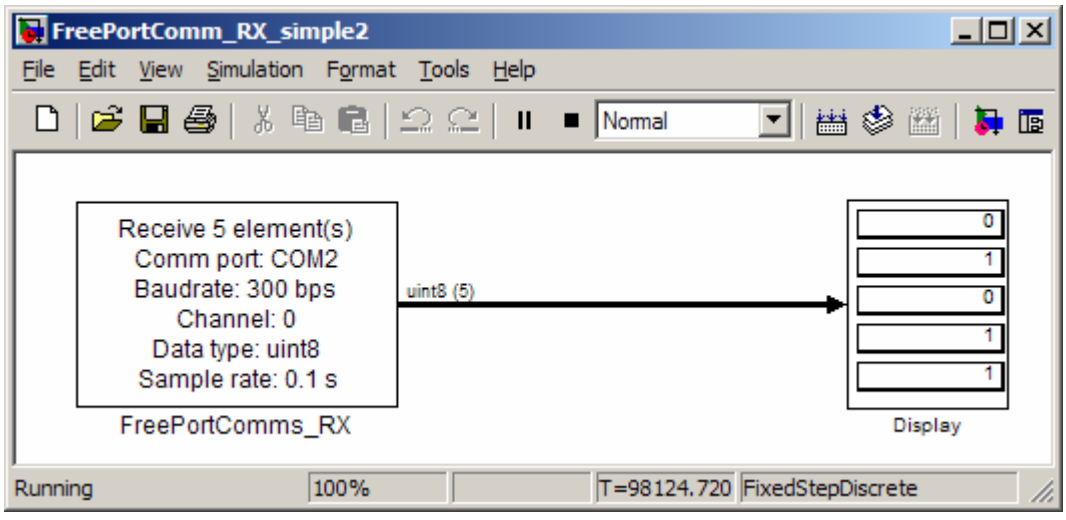


Figure 4-20 Sample model: FreePortComm_TX_simple2.mdl (target)

4.8.3 Simultaneous upload and download of data between host and target

The target model *FreePortComm_RXTX.mdl* and the corresponding host model *FreePortComm_TXRX.mdl* demonstrate the simultaneous upload and download between host and target using the FreePort communication blocks. Figure 4-21 is the target-sided model, whereas Figure 4-22 shows the host-sided equivalent.

Notice that the transmission of the 4-byte data-types *single* and *int32* currently does not work properly. The reason is that the 9S12 stores the individual bytes of floating-point numbers in reverse order as the host (PC). While it is relatively simple to reverse the order of each quadruple of bytes, this has not yet been implemented (no time to do it and we don't need this right now…). Future extensions might fix this bug.

fw-05-05

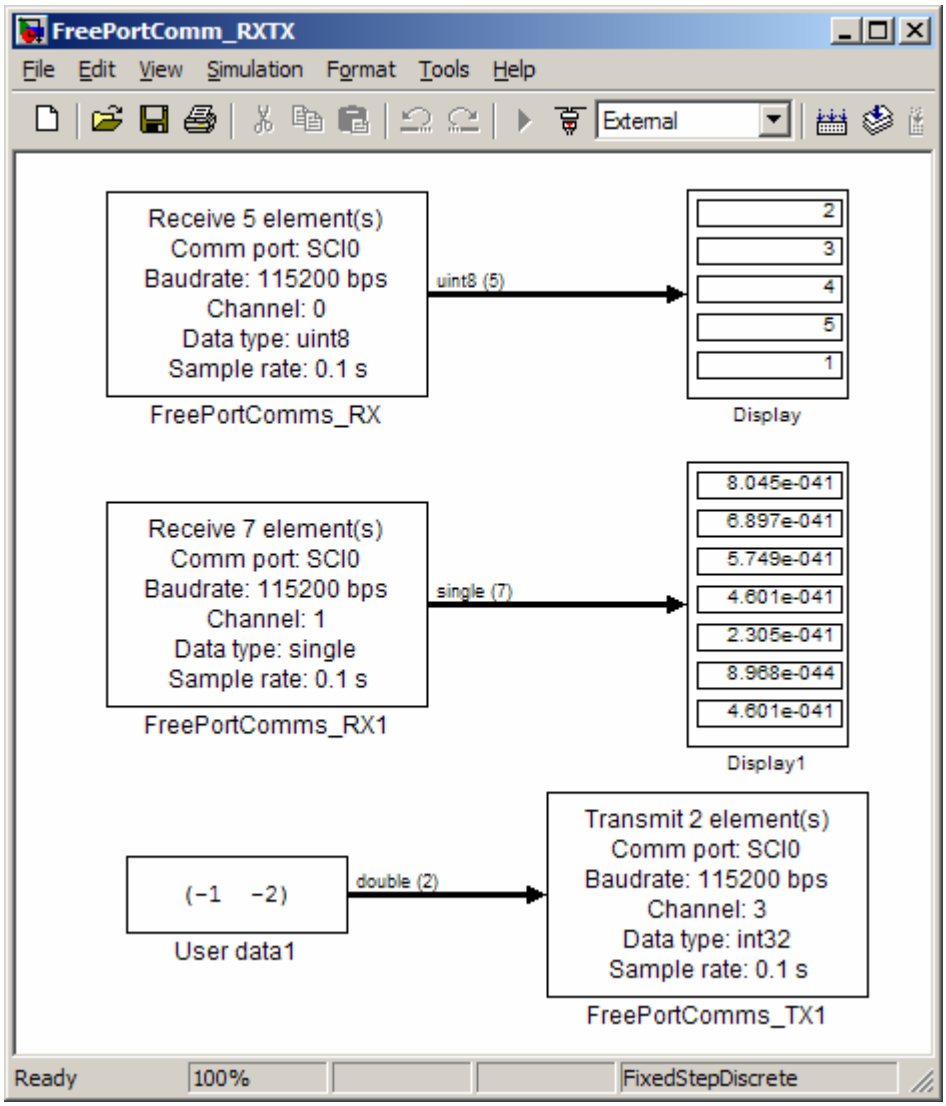


Figure 4-21 Sample model: FreePortComm_RXTX.mdl (target)

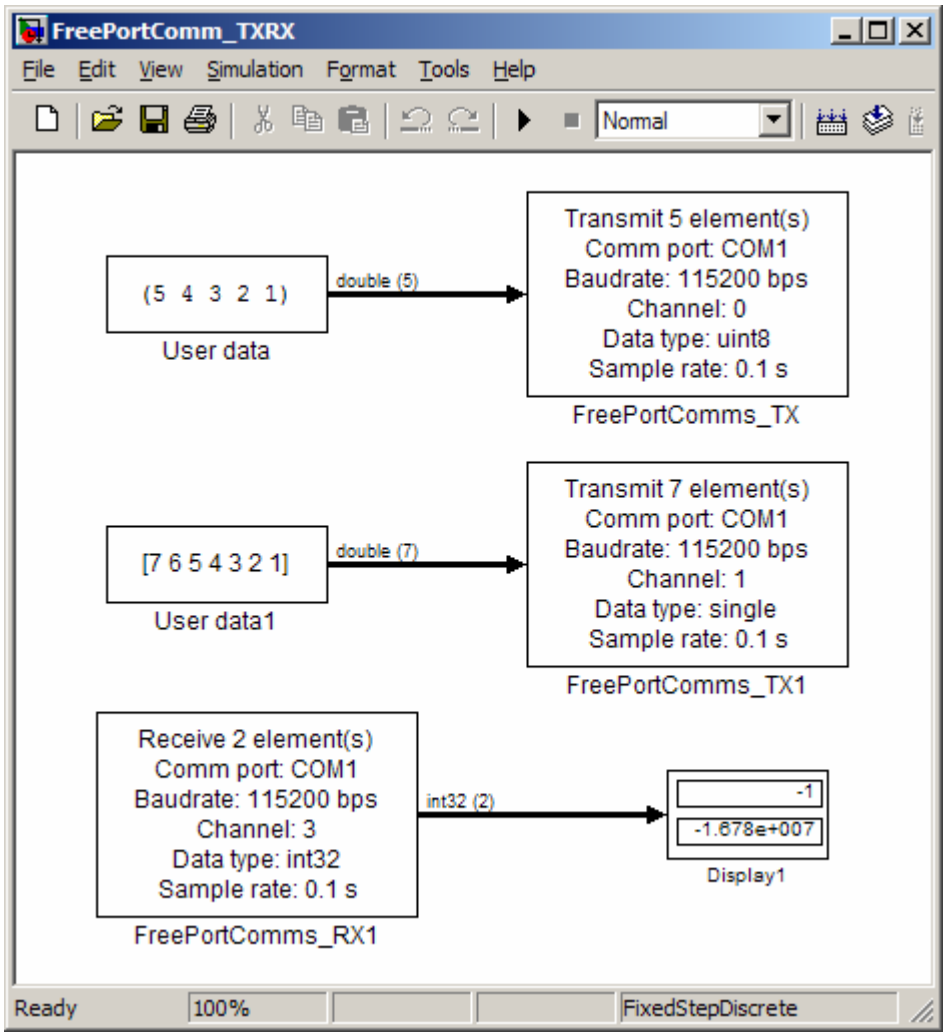


Figure 4-22 Sample model: FreePortComm_TXRX.mdl (host)

4.8.4 Download of unformatted data to the target

The target model *FreePortComm_RX_simple3.mdl* and the corresponding host model *FreePortComm_TX_simple4.mdl* demonstrate the download of raw data from the host to the target using the FreePort communication blocks. Figure 4-23 is the target-sided model, whereas Figure 4-24 shows the host-sided equivalent.

A similar pair of block diagrams can be designed for the upload of raw data from the target (microcontroller) to the host.

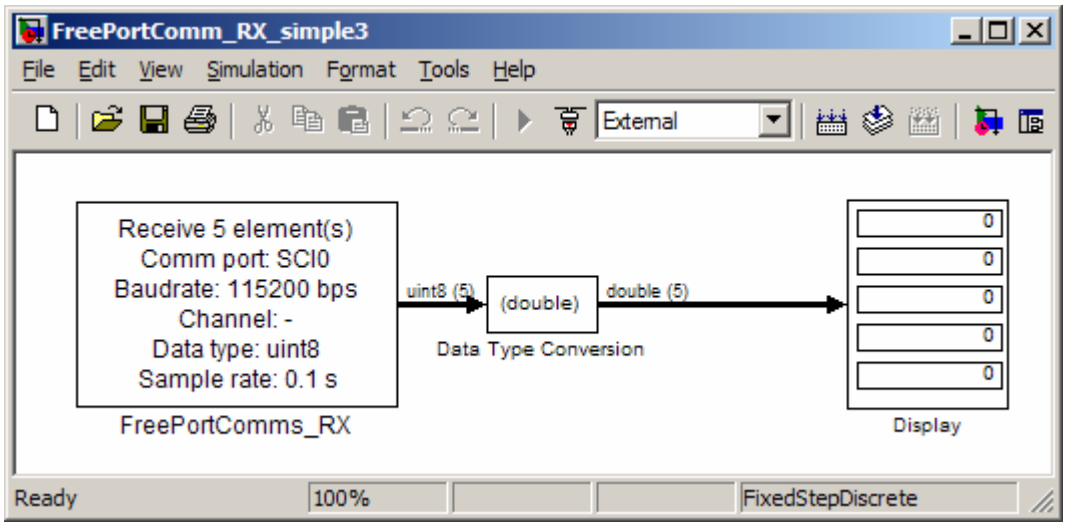


Figure 4-23 Sample model: FreePortComm_RX_simple3.mdl (target)

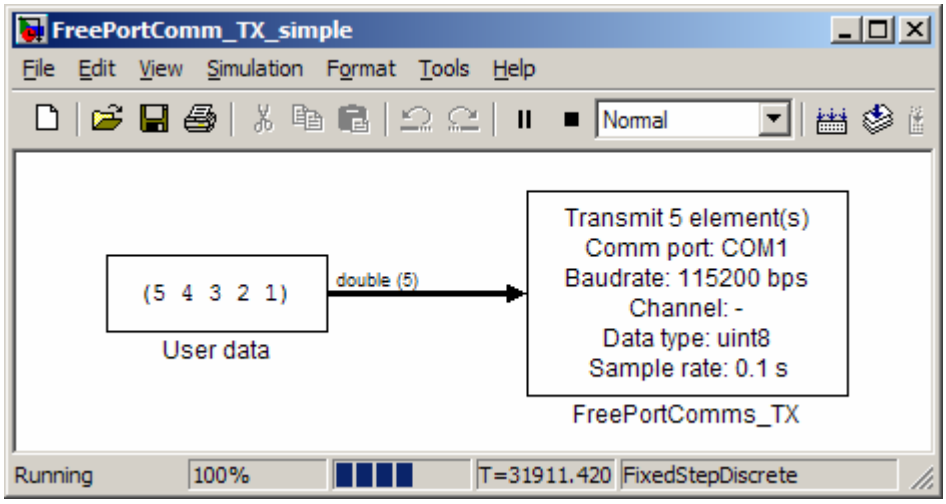


Figure 4-24 Sample model: FreePortComm_TX_simple3.mdl (target)

4.8.5 Upload and download of data via both ports SCI0 and SCI1

The target model *FreePortComm_RXTX_noExt.mdl* and the corresponding host model *FreePortComm_TXRX_noExt.mdl* demonstrate the download of data from the host to the target using SCI0 while simultaneously using SCI1 for data upload. Figure 4-25 is the target-sided model, whereas Figure 4-26 shows the host-sided equivalent. Notice that this requires the *External Mode* interface to be disabled (no background monitoring).

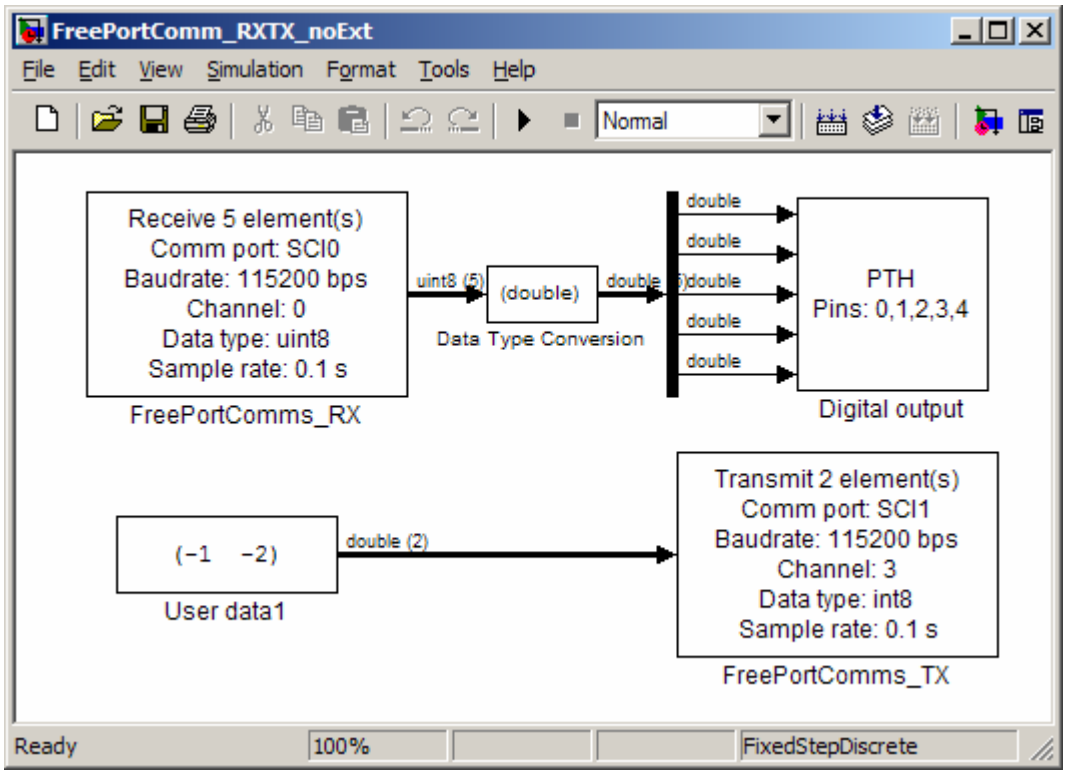


Figure 4-25 Sample model: FreePortComm_RXTX_noExt.mdl (target)

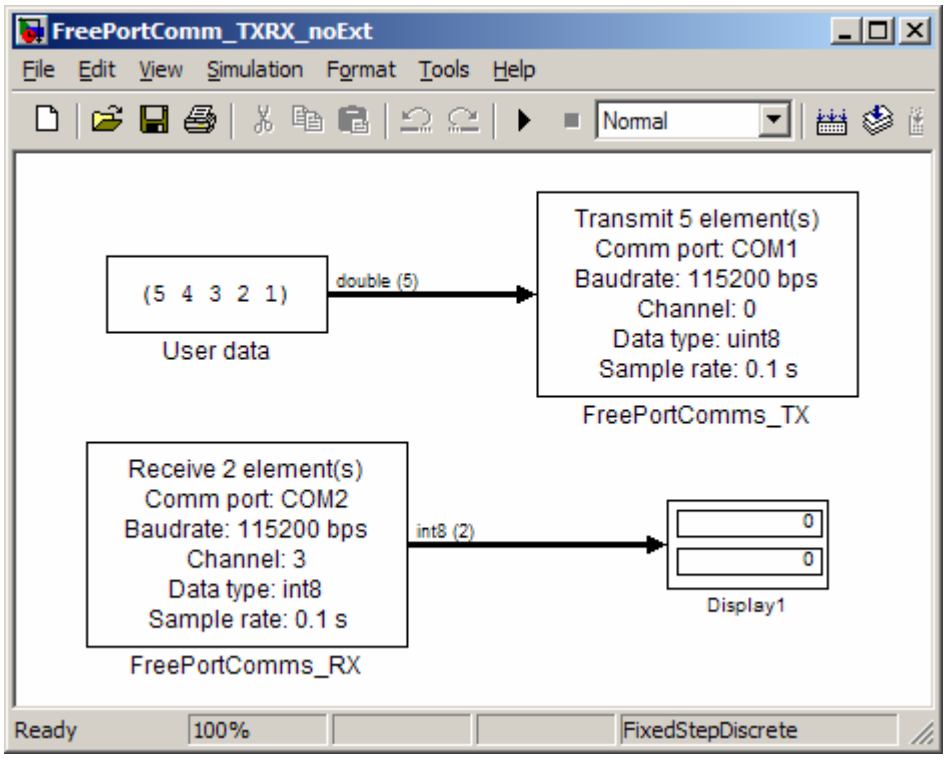


Figure 4-26 Sample model: FreePortComm_TXRX_noExt.mdl (host)

Note:

Using FreePort in parallel to the External Mode interface requires the target to be run out of ROM (switch SW7: on, reset) to prevent the on-chip serial monitor from interfering with the FreePort interface SCI0.

The FreePort communication blocks have been designed to cover a maximum number of situations. They can be used for serial communication between 2 hosts (e.g. COM1 on host A to COM1 on host B), COM1 to COM2 on one and the same host (loop-back operation), between a host and a target or between two different targets (e.g. microcontroller A to microcontroller B).

4.9 Toggle a pin of PORTB

4.9.1 Toggling a pin of PORTB in External Mode

The model *TogPort.mdl* is a simple test model which can be used to toggle an I/O pin of the microcontroller. The model uses a digital square wave block to produce an on/off sequence on PORTB pin 0. On the Dragon-12 development board, this pin is associated with one of the on-board LEDs. The model should therefore make this LED flash with a period of 1 second (0.5 s on, 0.5 s off). Both frequency as well as duty cycle of this pulse sequence can be changed on-the-fly (external mode).

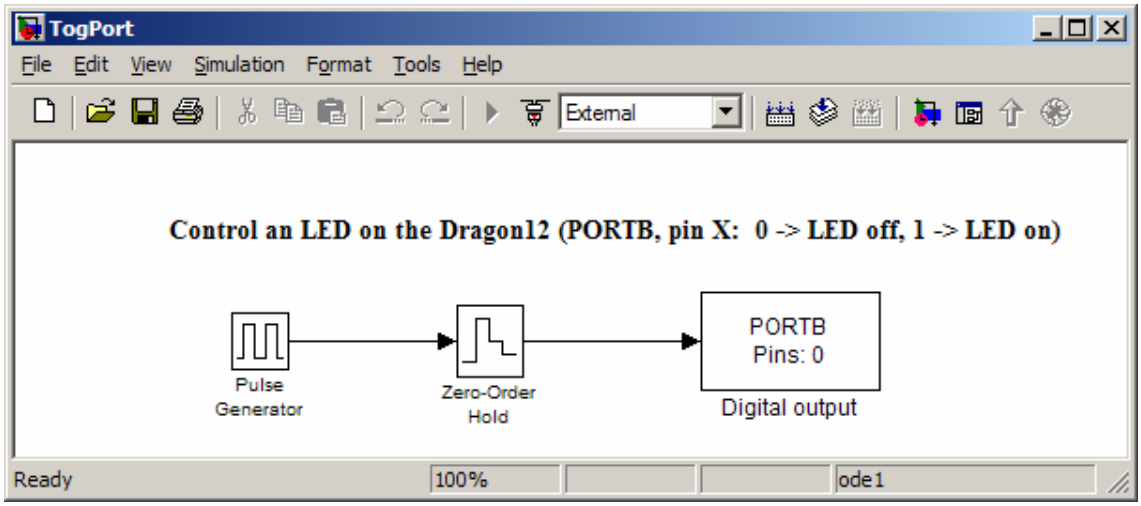


Figure 4-27 Sample model: TogPort.mdl

4.9.1 Toggling a pin of PORTB in standalone mode

The model *TogPortNoCom.mdl* is a simple test model which runs as standalone program, i. e. without the use of the External Mode communication interface. Consequently, the generated executable is much smaller than is the case for all other sample models (approx. 4 kByte instead of the typically 17 - 20 kByte). Following the

successful download of this program and a subsequent reset of the target hardware, the program is automatically launched and runs until the power is switched off.

4.10 Miscellaneous sample models

As a proof of concept a number of the original MathWorks demo files have been built with rtmc9S12-Target and tested on the Dragon-12 development board.

4.10.1 The F14 simulation

The model *f14.mdl* is a direct copy of the original Simulink demonstration file. This example has been included to validate the correct operation of rtmc9S12-Target with models of a slightly elevated complexity. The model runs at a base sample rate of 50 Hz (sample time: 20 ms) with an upload of log data every 100 ms.

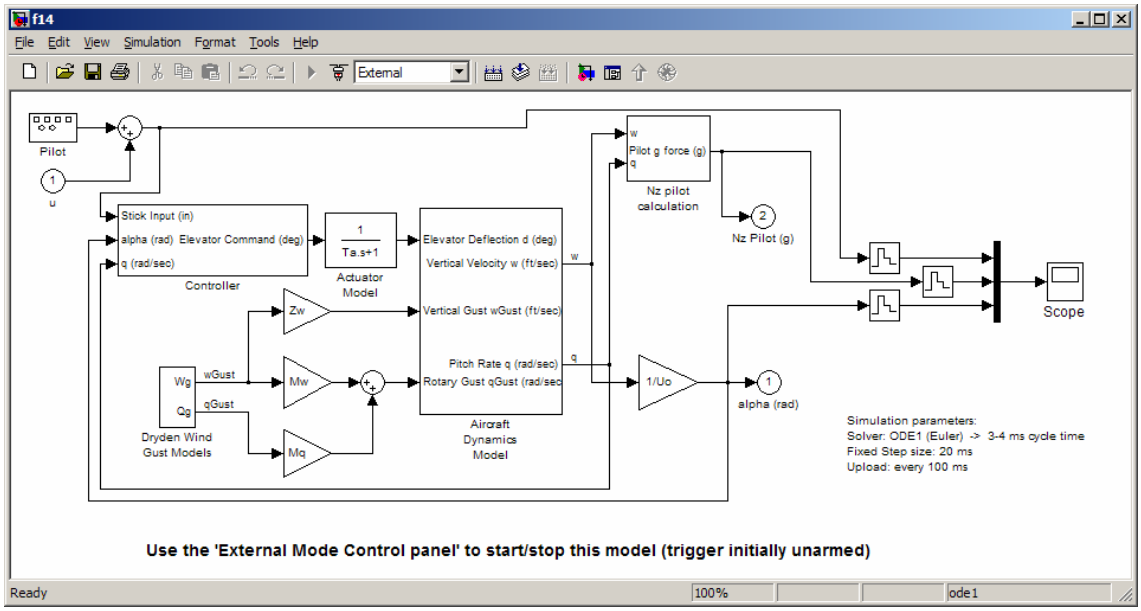


Figure 4-28 Sample model: f14.mdl

4.10.2 A rudimentary robot control

The target model *robot.mdl* together with its corresponding host model *robot_pc.mdl* provide a very basic remote control for a mobile robot. The target model receives direction commands using user data channel 0. The two values are multiplied by a common factor (speed) which is sent to the robot using user channel 1. Many more sophisticated robot control systems can be devised. The pair of models has simply been included to provide a simple starting point for further experiments. In our laboratory, we supply the robot with information about its own position, thus allowing it to react to what it *sees*. Co-ordinates and information about its orientation are supplied

from a suitably programmed video grabber unit and sent to the robot using two separate user channels. Many more applications can be thought of (and put together in minutes), including the communication between various mobile robots. This opens the door to applications using artificial intelligence and the co-ordinated collaboration of otherwise autonomous units.

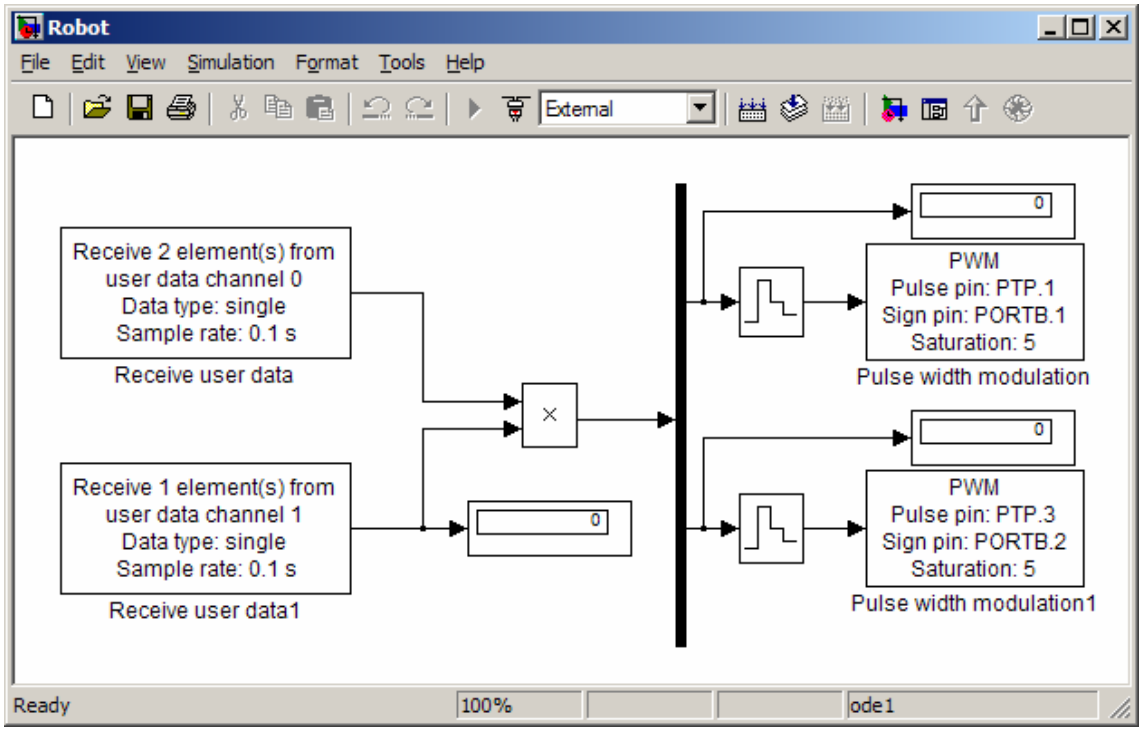


Figure 4-29 Sample model: Robot.mdl (target)

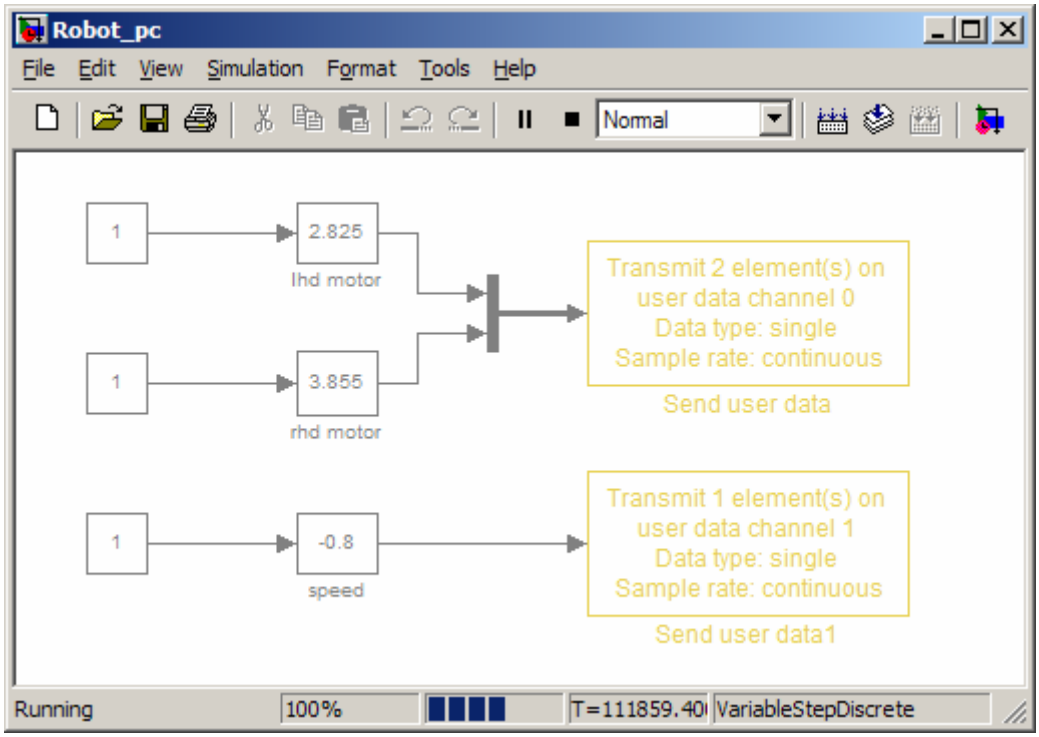


Figure 4-30 Sample model: Robot_pc.mdl (host)

4.10.3 Band-limited white noise generator

The target model *BLNoiseTest.mdl* is a simple test model to ensure the correct operation of the automatically generated real-time libraries. This example generates band-limited white noise in form of pulses which are monitored at a rate of 20 Hz.

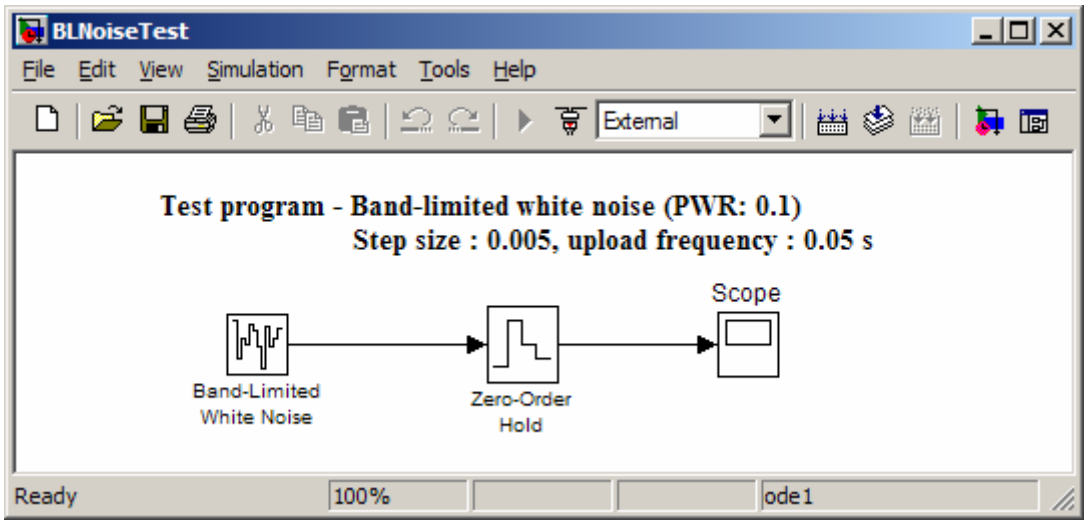


Figure 4-31 Sample model: BLNoiseTest.mdl

4.10.4 Generation of a chirp signal (frequency wobbling)

The target model *ChirpTest.mdl* is a simple test model to ensure the correct operation of the automatically generated real-time libraries. This example generates a sinusoidal oscillation with slowly increasing frequency. The frequency is wobbled from 0.1 Hz to 1 Hz over a sweep time of 60 seconds.

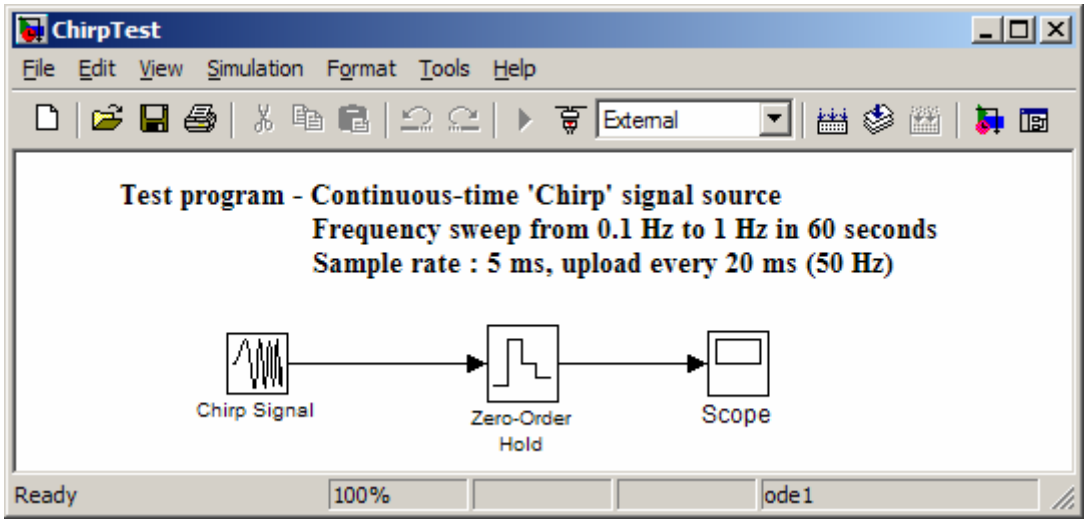


Figure 4-32 Sample model: ChirpTest.mdl

Appendix

Appendix A – The Dragon-12 development board

Under construction.

Appendix B – The MiniDragon+ development board

Under construction.

Appendix C – Things to do…

(1) Further improvement of the serial communication between host and target; additional effort may be spent on re-sending of lost user data telegrams and the re-synchronisation of host and target following a communication error.

(2) Debugging of user communication modules – these are still not reliable enough (compared to the C167 version which exhibits no problems whatsoever).

(3) Development of a block for incremental encoders

(4) Replacing of the currently used single-tasking system with a priority based multi-tasking system.

(5) Testing and improvement of the FreePort communication system.

(6) Enable second serial port when the external mode is not active.

(7) Enabling communication using the CAN bus.

(8) Host-to-host communication using a backbone network (e. g. Ethernet, etc.)