

RPI+ Simulink coder

`jacques.gangloff@unistra.fr`

This tutorial will explain the essential steps required to target a raspberry pi (RPI) hardware with the Matlab Simulink Coder toolbox. At the end of this tutorial, you will have a new functional target in your target list. With a single click, Matlab will automatically convert your Simulink model into a Linux-compatible c-code project, upload and compile it on the ARM11 platform of the RPI, launch it and connect to it with Simulink's external mode. You will be able to monitor the signals in real-time with scopes or change the parameters values in blocks on the fly. The code on the RPI will run in soft real-time using standard POSIX calls. According to benchmarks, you can expect jitters below 500 μ s.

RPIt Simulink coder

jacques.gangloff@unistra.fr

Introduction

Simulink Coder, formerly known as “Real-time Workshop”, is a Matlab/Simulink toolbox providing automatic code generation features from a bloc-diagram description of the system. This feature is called “Model-based” programming. You describe the system in Simulink, and Simulink coder translates this description in a C or C++ project that can be compiled and run on a target computer. The process of translating the code can be heavily customized in order to be tailored to a specific target. In this tutorial, we will learn how to customize this translation for the Raspberry pi (RPI), a cheap ARM11 platform that is perfectly suited for educational purposes.

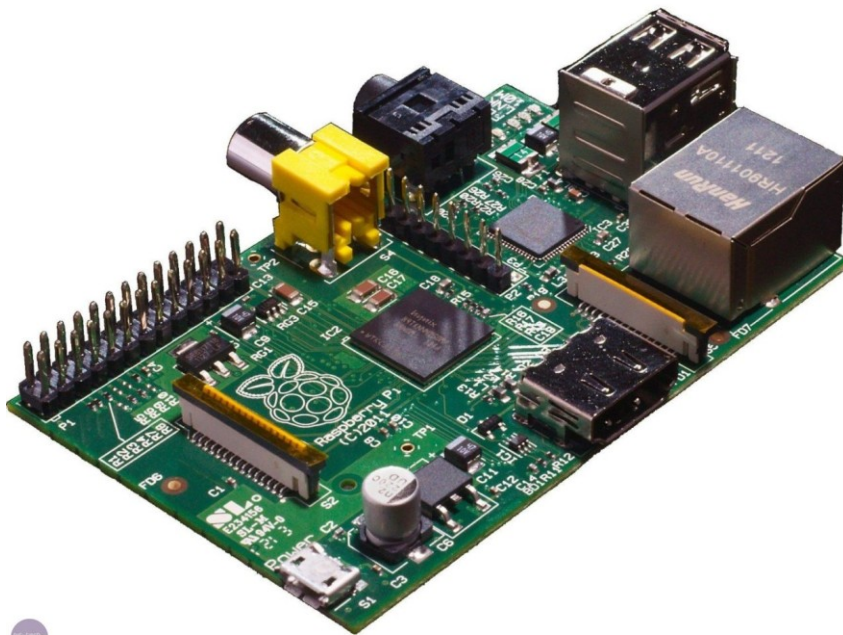


Figure 1 - The Raspberry pi

Quickstart

...

1. Extract the package into a directory of your choice which path contains no spaces.
2. Run the “setup.m” script.
3. Go to the section “Testing the RPI target” at the end of this document.
4. If something goes wrong, read the whole documentation

Jacques Gangloff

Configuration of the target

Prerequisites

This tutorial has been made as generic as possible. However, the process of creating a target is subject to variations with the various releases of Matlab. This tutorial has been tested on a R2010b version of Matlab running Windows XP 32bits. It should work with little modifications on other Windows hosts and on subsequent releases of Matlab. Note that in release 2010b, the “Simulink coder” toolbox is still called “Real-Time Workshop”. For Linux hosts, some more important modifications should be expected. The Simulink coder and the embedded coder toolboxes are required. Matlab coder toolbox, though very useful, is optional. Since the compilation is performed directly on the RPI, there is no need to install a toolchain. The RPI is powerful enough to provide decent compilation time. Furthermore, since you compile directly on the target, all the desired libraries that you could need for your custom blocks are already at hand or installable with the “apt-get” utility.

Creation of a new target

When opening a Simulink model parameter configuration dialog box under the “Real-Time Workshop” tab, there is a field entitled “System target file”:

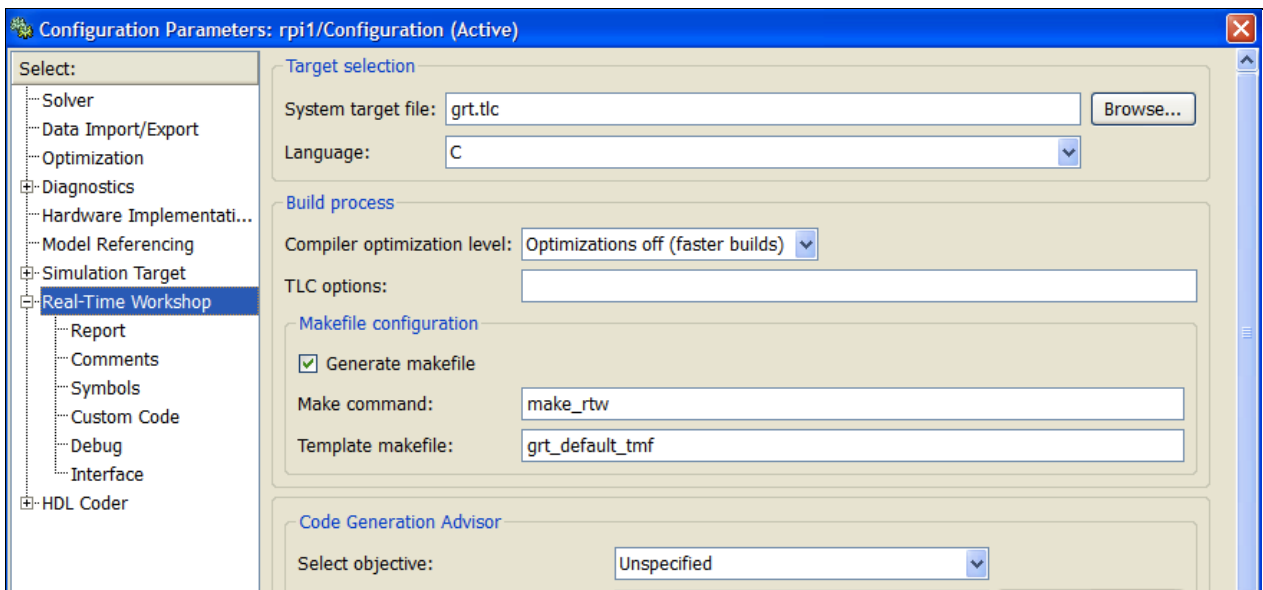


Figure 2 - Configuration of a system target

The default system target file is “grt.tcl” which means “Generic Real-Time”. The “tcl” suffix stands for “Target Language Compiler”. It is a MathWorks proprietary language that is used to define the translation process from Simulink model description to C or C++ code. In this section we will create a custom “.tcl” file for the RPI target and some peripheral files that this file uses.

1. Create a new “rpit” folder under the existing “rpit” folder:
“E:\MATLAB\Targets\rpit\rpit”

2. Add this last path to Matlab's known paths from the "File->SetPath" menu item (see Figure 3).
3. In folder "MATLAB\R2010b\rtw\c\ert" copy the file "ert.tlc" and paste it in the folder "rpit\rpit". Rename it "ert_rpi.tlc".
4. Edit the file "ert_rpi.tlc" with your preferred text editor (you can use Matlab's one). Change its header according to Figure 4. The information in the header will be used by Simulink when browsing for a target file.
5. Then remove almost everything at the bottom of the file to leave only what is shown in Figure 5.
6. A Template MakeFile has also been defined in "ert_rpi.tlc" in its header. It is called "ert_rpi.tmf". It defines the way the customized Makefile should be defined. As a starting point, we will use a template provided by MathWorks and then customize it to cross-compile for the RPI. Copy the file "ert_unix.tmf" from "MATLAB\R2010b\rtw\c\ert" and paste it into "rpit\rpit". Rename it as "ert_rpi.tmf". Edit it and look at its syntax. We will modify it later. If you want to learn the syntax of ".tmf" file, you can refer to Matlab's embedded help. But it's not mandatory for this tutorial.
7. A callback file has been defined in "ert_rpi.tlc". It is called "rpi_callback_handler" and defined by a Matlab function. It will be called when the user selects or modifies the customized target we are creating. Its purpose is to automate some critical parameter settings with command line instructions. Create a file "rpi_callback_handler.m" in "rpit\rpit" with only this line: "function rpi_callback_handler(hDlg, hSrc)".
8. Now we should be able to test our new target. Create an empty Simulink model. Open the parameter configuration dialog box. When browsing for a system target file, you should now see a new entry entitled "ert_rpi" as shown in Figure 6.

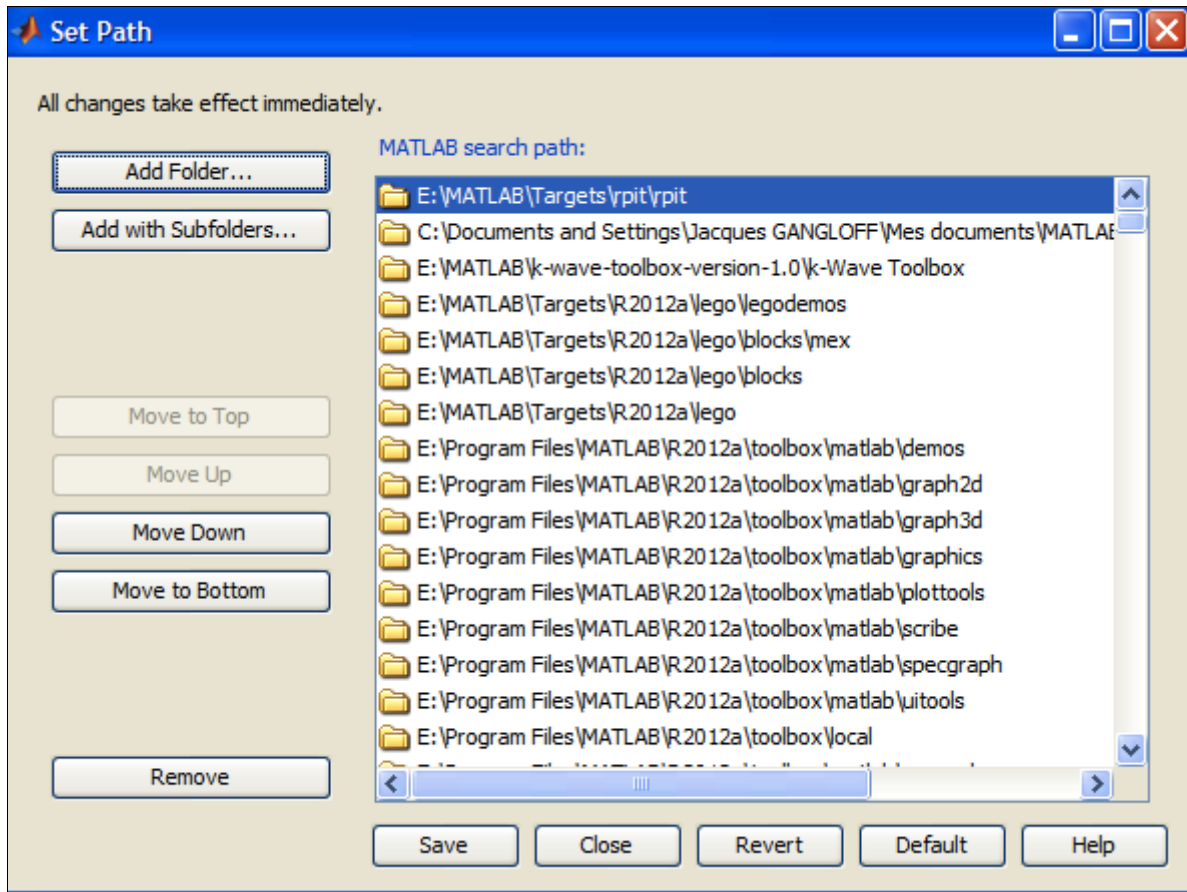


Figure 3 - Defining a new path

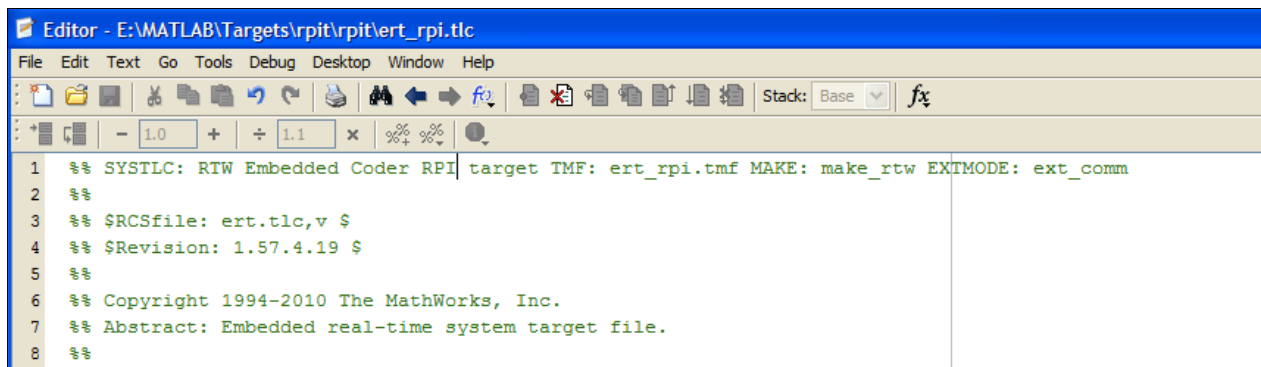


Figure 4 - Customize main TLC file header

```

24 %assign SuppressMultiTaskScheduler = TLC_TRUE
25 %endif
26
27 %include "codegenentry.tlc"
28
29 %% Configure code generation settings
30 /%
31 BEGIN_RTW_OPTIONS
32
33 %-----%
34 % Configure RTW code generation settings %
35 %-----%
36
37 rtwgensettings.DerivedFrom      = 'ert.tlc';
38 rtwgensettings.BuildDirSuffix  = '_ert_rtw';
39 rtwgensettings.Version         = '1';
40 rtwgensettings.SelectCallback  = ['rpi_callback_handler(hDlg, hSrc)'];
41 rtwgensettings.ActivateCallback = ['rpi_callback_handler(hDlg, hSrc)'];
42
43 END_RTW_OPTIONS
44 %/
45

```

Figure 5 - Customizing code generation settings

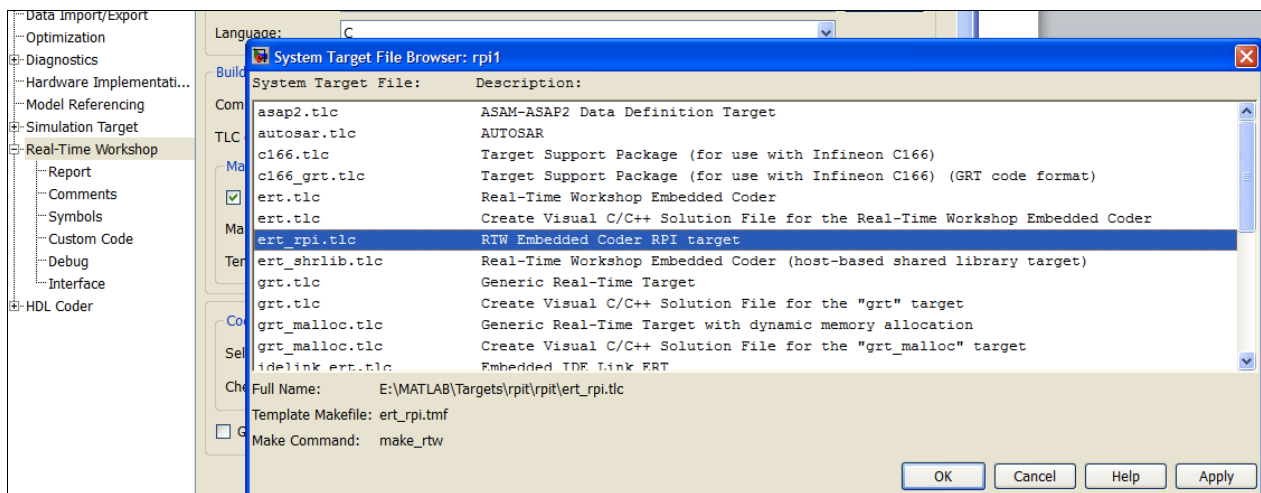


Figure 6 - Testing the new target

Enabling external mode

External mode is a very nice feature enabling bidirectional communication between the host and the target while the system is running. From host to target, it allows for parameters modification into the blocks on the fly and so, for example, testing in live the effect of gain modifications on a running system. In the other direction, from target to host, it allows various signals to be monitored using almost all the blocks that are in the "Sinks" sub-library.

This was a real struggle to troubleshoot. Indeed, this trick was buried deep inside the documentation. Simulink can be taught to show some optional tabs in the configuration dialog box. By default, the tab “Real-Time Workshop->Interface” is not shown. To enable it, create a file entitled “sl_customization.m” in the “rpit\rpit” directory with the following code inside:

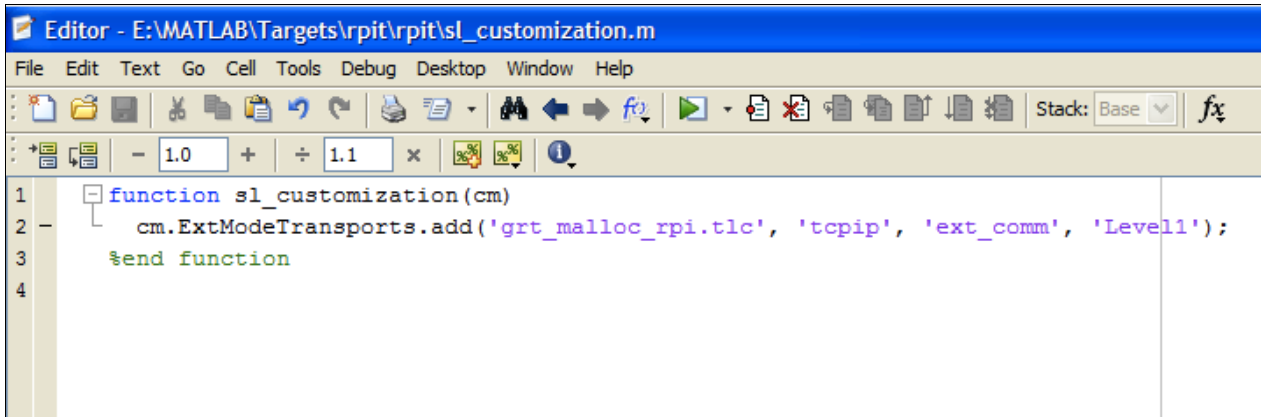


Figure 7 - Enabling external mode

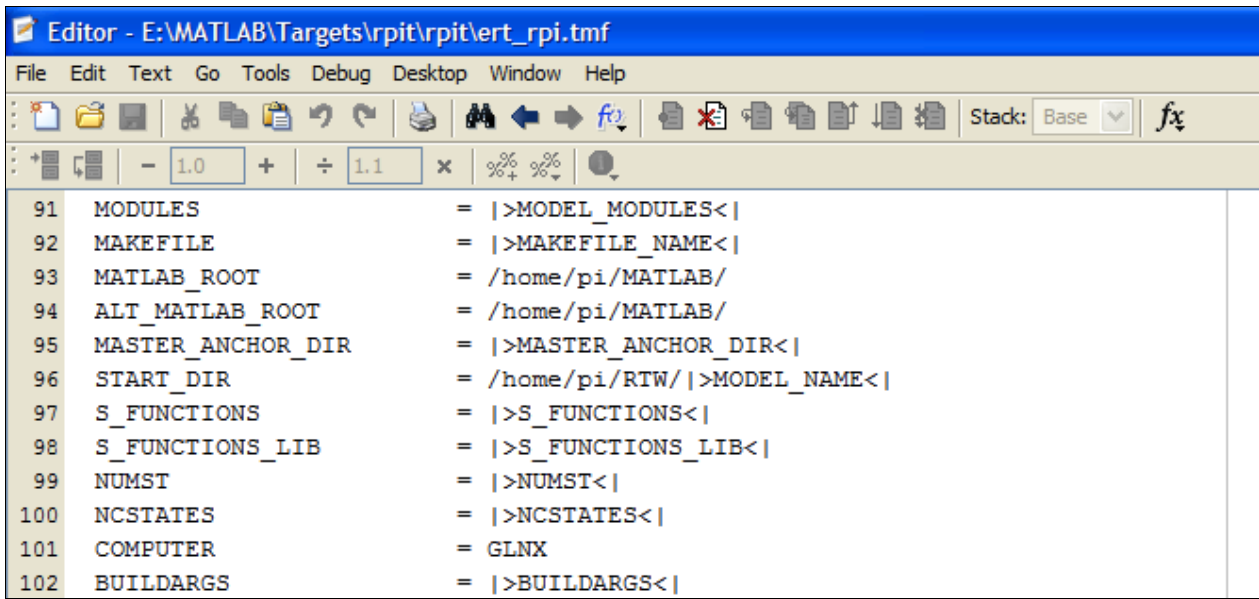
Restart Matlab. Now, when selecting your target, you will be able to see the “Interface” tab in the parameter configuration dialog box with “tcpip” as an optional transport layer.

Customizing the Makefile generation

In this section we modify the way Simulink compiles the target executable. With the template we imported from the Matlab tree, compilation is configured to use the default compiler that you select with the command “mex -setup”. By default, the executable is intended to run on the host, not on a distant target. With little modifications in “ert_rpi.tmf” we can alter the Makefile generation in order to generate a Makefile that can run on the RPI after been automatically uploaded.

1. Change the header of “ert_rpi.tmf” according to Figure 8. Keep in mind that the Makefile that will be generated from this TMF file will run on the RPI. So, it is coherent to have a “make” command located at “/usr/bin/make” on the RPI filesystem.
2. In “ert_rpi.tmf”, at the top of section “Tool specifications”, replace the first line that includes a “.mk.” file according to Figure 9. This will define the main macros needed for compilation.
3. Modify some macros in order to point to right paths in the RPI filesystem according to Figure 10.
4. This step is optional. If you need to further customize your Makefile and need to know the root directory of your target files installation, you can follow this step. In the file “rpi_callback_handler.m” add the line “set_param(bdroot, 'PostCodeGenCommand', 'postGenFunc(buildInfo)');”. This will tell the Makefile generation process to call a function “PostCodeGenCommand” beforehand. So we need to define this function. It will guess the installation directory and store it into a custom “TMF” variable called “|>RPI_MW_ROOT<|”. Create a new file in the current directory called “postGenFunc.m”. Fill it according to the script given in Figure 11. Edit the file “ert_rpi.tmf”. Just below the definition list of TMF constants add the line “RPI_MW_ROOT = |>RPI_MW_ROOT<|”.

Figure 8 - Header of the TMF file
Figure 9 - TMF file customization: set the main macros

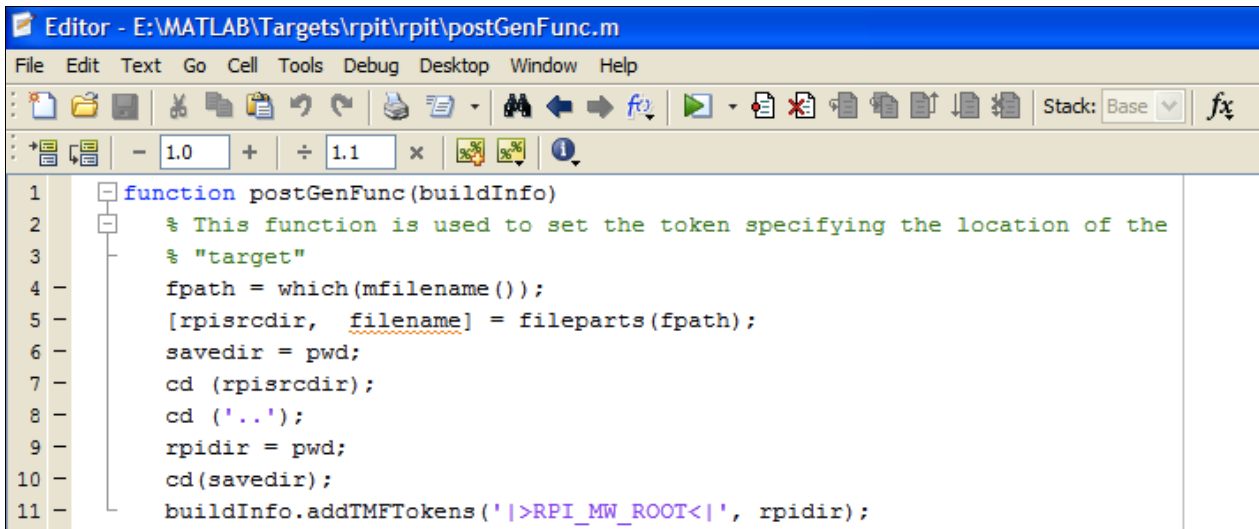


```

Editor - E:\MATLAB\Targets\rpit\rpit\ert_rpi.tmf
File Edit Text Go Tools Debug Desktop Window Help
+ - 1.0 + 1.1 x % % %
91 MODULES = |>MODEL_MODULES<|
92 MAKEFILE = |>MAKEFILE_NAME<|
93 MATLAB_ROOT = /home/pi/MATLAB/
94 ALT_MATLAB_ROOT = /home/pi/MATLAB/
95 MASTER_ANCHOR_DIR = |>MASTER_ANCHOR_DIR<|
96 START_DIR = /home/pi/RTW/|>MODEL_NAME<|
97 S_FUNCTIONS = |>S_FUNCTIONS<|
98 S_FUNCTIONS_LIB = |>S_FUNCTIONS_LIB<|
99 NUMST = |>NUMST<|
100 NCSTATES = |>NCSTATES<|
101 COMPUTER = GLNX
102 BUILDARGS = |>BUILDARGS<|

```

Figure 10 - Customizing paths



```

Editor - E:\MATLAB\Targets\rpit\rpit\postGenFunc.m
File Edit Text Go Cell Tools Debug Desktop Window Help
+ - 1.0 + 1.1 x % % %
1 function postGenFunc(buildInfo)
2 % This function is used to set the token specifying the location of the
3 % "target"
4 fpath = which(mfilename());
5 [rpisrcdir, filename] = fileparts(fpath);
6 savedir = pwd;
7 cd (rpisrcdir);
8 cd ('..');
9 rpidir = pwd;
10 cd(savedir);
11 buildInfo.addTMFTokens('|>RPI_MW_ROOT<|', rpidir);

```

Figure 11 - Root directory determination

Define a STF_make_rtw_hook.m file¹

To create a custom STF_make_rtw_hook hook file for your build procedure, copy and edit the example ert_make_rtw_hook.m file (located in the matlabroot/toolbox/rtw/targets/ecoder folder) as follows:

1. Copy ert_make_rtw_hook.m to a folder in the MATLAB path, and rename it in accordance with the naming conventions described in File and Function Naming Conventions. For example, to use it with the GRT target grt.tlc, rename it to grt_make_rtw_hook.m.
2. Rename the ert_make_rtw_hook function within the file to match the filename.
3. Implement the hooks that you require by adding code to the appropriate case statements within the switch hookMethod statement.

¹ Parts of this section are extracted from Matlab help.

So our hook file will be called `"ert_rpi_make_rtw_hook"`. This file is very important since it can be used to automate the whole process of compilation, uploading and starting of the model. There is a switch in this file with many cases, depending on the state of the compilation process. The case that interests us is `"case 'after_make'"`. It is called after the file generation has occurred (in our case, Matlab does not launch the make command since it is not recognized in the Windows tree, so it only generates the code). We will not go into all the details of this script but only describe the main steps.

We use two very interesting Windows tool that can be called from the command line: `"pscp"` and `"plink"`. The first is the Windows equivalent of `"scp"`. It is used to upload files to the RPI. The `"-i"` flag tells the command where the private key is located so the connection can be passwordless. This passwordless connection should be configured by running the `"setup.m"` script. Reading this script may be very instructive. The `"plink"` program is like UNIX `"ssh"` command. It is used here to run shell commands distantly on the RPI.

Here are the main steps that are performed after file generation :

1. Check if the RPI responds.
2. Kill any `"rpi"` running programs. Indeed the target real-time executable will always be called `"rpi"`.
3. Synchronize the clocks: in order to have the make process to correctly detect file changes, clocks have to be synchronized. Indeed, the RPI don't have its time saved. So, if there is no access to a NTP server, its time can be totally wrong.
4. Upload all the files to a `"RTW"` directory on the RPI.
5. We need to modify some text in the generated Makefile. Indeed, rules are automatically generated by Matlab. Especially, when the model contains a S-Function, the associated code is located in the Windows tree. In this case, the TMF creates a rule that points to this tree. Of course, it is inexistent on the RPI. In fact, the `"setup.m"` script copies all the S-Function sources contained in the `"blocks"` directory into a `"~/MATLAB/blocks"` directory on the RPI. Note that the `"~/MATLAB/"` directory on the RPI contains also some important folders from MATLAB's installation directory that are needed for the compilation. This is also installed automatically the `"setup.m"` script. Since the Makefile must be altered, its timestamp will change, so we need to backup it beforehand, run the find a replace script based on UNIX `"sed"` command and restore it afterwards. Note that the whole process is run on the RPI.
6. Compilation is launched on the RPI. The first time, it may take a while since all S-function sources must be compiled. But the second time, only a few files are compiled and the process should be over in less than 15 seconds.
7. The executable is renamed `"rpi"`.
8. The executable is launched with the options `"-w -tf Inf"` which means it should start in a wait state and the run indefinitely.
9. Then, Simulink is taught to connect to the target and start in external mode. After that, the model running on the RPI can be monitored in real-time using the scope or other sinks and some parameters can be modified using the blocks standard dialog box.

Customizing the “main” function generation

By default, the model runs at top speed, its rate depending on the hardware it runs on. Since we intend to test real-time processes that may interact with external hardware, it is critical to be accurately clocked. It is possible to run a hard real-time OS on the RPI like Xenomai. It has been proved that jitters less than 20us can be achieved in this case. Here, for simplicity sake, we will only use soft real-time functions of the standard Linux kernel provided in the Raspbian distribution (contraction of raspberry and Debian). Benchmarks show that the worst case jitter is around 500us with standard deviation around 80us.

According to Matlab’s documentation:

matlabroot/toolbox/rtw/targets/ecoder/example_file_process.tlc: An example custom file processing (CFP) template, which you should use as the starting point for creating your own CFP templates. Guidelines and examples for creating a CFP template are provided in Generating Source and Header Files with a Custom File Processing (CFP) Template.

Search for the string “Generating Source and Header Files with a Custom File Processing (CFP) Template” in Matlab’s integrated documentation to have a good tutorial on this point. We will here describe the main steps.

To support generation of main program modules, two TLC files are provided:

- “bareboard_srmain.tlc”: TLC code to generate an example single-rate main program module for a bareboard target environment. Code is generated by a single TLC function, “FcnSingleTaskingMain”.
- “bareboard_mrmain.tlc”: TLC code to generate a multirate main program module for a bareboard target environment. Code is generated by a single TLC function, “FcnMultiTaskingMain”.

Only these last two file will be customized for the RPI. CFP should be configured in Simulink’s parameters configuration dialog box as shown in Figure 12.

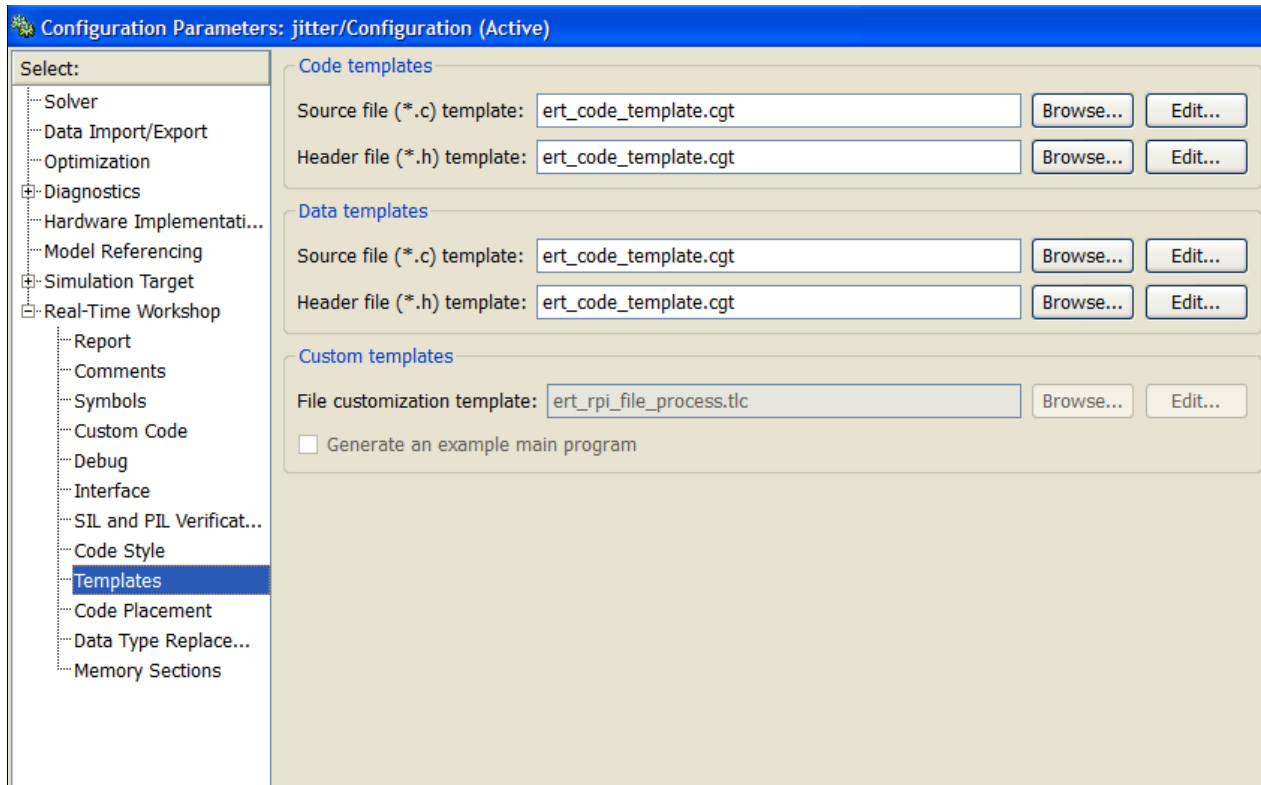


Figure 12 - File customization template

1. "example_file_process.tlc" should be renamed "ert_rpi_file_process.tlc". Also "bareboard_srmain.tlc" is renamed "rpi_srmain.tlc" and "bareboard_mrmain.tlc" is renamed "rpi_mrmain.tlc".
2. Edit "ert_rpi_file_process.tlc" and modify it according to Figure 13. Don't forget to uncomment the "%assign ERTCustomFileTest = TLC_TRUE" line.
3. Edit "rpi_srmain.tlc" and "rpi_mrmain.tlc" and modify them. The first is invoked when your model has only one sampling rate whereas the latter is invoked when you have blocks with different sampling time. Here the modifications are too lengthy to be detailed. Just look into these files to understand the process. You could make change to these files and look at the result into the generated "ert_main.c" file into the model "modelname_ert_rtw" subdirectory. Basically, two functions are added: "rpi_make_periodic" and "rpi_wait_period". The first initializes a timer at the fastest model sampling rate obtained through the "%assign period = LibGetClockTickStepSize(0)" TLC function. It uses the Linux "timerfd_create" function. The second added function is called at every time step and is used to wait for the next clock tic. It uses a read on the created "timer_fd" file descriptor.

```

9  %% Note: This file can contain any valid TLC code, which Real-Time Workshop
10 %% executes just prior to writing the generated source files to disk.
11 %% Using this template "hook" file, you are able to augment the generated
12 %% source code and create additional files.
13 %%
14 %% Copyright 1994-2006 The MathWorks, Inc.
15 %%
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 %selectfile NULL_FILE
18
19 %% Uncomment this TLC line to execute the example
20 %% || ||
21 %% || ||
22 %% \ / \ /
23 %assign ERTCustomFileTest = TLC_TRUE
24
25 %if EXISTS("ERTCustomFileTest") && ERTCustomFileTest == TLC_TRUE
26
27     %% Need to set the template compliance flag before you can use the API
28     %<LibSetCodeTemplateComplianceLevel(2)>
29
30     %% Create a simple main. Files are located in MATLAB/rtw/c/tlc/mw.
31
32     %if LibIsSingleRateModel() || LibIsSingleTasking()
33         %include "rpi_srmain.tlc"
34         %<FcnSingleTaskingMain()>
35     %else
36         %include "rpi_mrmain.tlc"
37         %<FcnMultiTaskingMain()>
38     %endif
39
40 %endif
41

```

Figure 13 - Modifying the main CFP file

Configure parameters automatically

When you select a target in the RTW configuration pane, it is possible to configure a callback to be called when pressing the “Apply” or “OK” button. These callbacks were defined at the end of the file “ert_rpi.tlc”. The callback function is called “rpi_callback_handler” and it is stored in the file “rpi_callback_handler.m”. When you look into this file, you see two kinds of invocations:

- “slConfigUISetVal”: sets a value of the model configuration.
- “slConfigUISetEnabled”: enables or disables (grey it out) a configuration item.

Read the comments in this file to know what has been configured. Note that inline parameters has been disabled to allow for real-time modification of blocks parameters during external mode.

Creating custom blocks

The main interest of having a model running on a distant hardware is to interact with peripherals (sensors, actuators) in real-time. These peripherals are platform dependent, so custom blocks have to be defined to interact with them within Simulink. The way to do this is to use S-Functions.

1. Create a “blocks” directory directly in your target root folder and add it to Matlab’s known paths.
2. In Simulink, open a new model and drag a block called “S-Function Builder” in it.
3. Double-click in this block to open its interface.
4. To define its sampling rate and make it a parameter, select “Discrete” sampling mode and enter a parameter name in “Sampling time value”. Let say “rpi_Ts”.
5. In the “Data properties” tab and the “Parameters” subtab, add a new parameter and call it “My_sampling_time”. At the top of the dialog box, in the “S-function parameters” area, you now have a line dedicated to this parameter. Enter a variable name in this field that will be used later in a mask dialog box to define the sampling time of the whole masked block (Figure 14).
6. Enter a name for your S-Function.
7. You can add as many other inputs or parameters to your S-function, depending on your needs.
8. In the “Outputs” pane, enter the custom code of the S-function. Note that Matlab needs a local executable of this code to work (especially to guess the block sampling time). This executable is created with the “mex” command in the “setup.m” script. This fake executable will only output 0 as shown in the code of Figure 15. The macro “MATLAB_MEX_FILE” is false in the case of a mex file and so only the code below “#else” will be compiled on the RPI platform.
9. In the “Build info” pane (Figure 16), you should select “Generate wrapper TLC”. Optionally, you can select “Start” and/or “Terminate” by pushing the “Additional methods” button if you need a specific constructor and/or destructor function to be created in the generated TLC file.
10. The S-function clock cannot be masked alone. A simple workaround is to embed it into a subsystem as shown in Figure 17 and then create a mask on the subsystem.
11. After having the mask created, you can edit it as shown in Figure 18. Be sure to enter at least one parameter, the one defining the sampling time.
12. After that, by double-clicking the mask block containing the S-function, a dialog box pops up asking you to enter a value for the defined parameter as shown in Figure 19.
13. You may embed your custom block into a library and make this library known by Simulink’s model browser. To do that, simply create a new library, copy the block into it, save it into the “blocks” directory and add a “slblocks.m” file into this directory according to Figure 20.

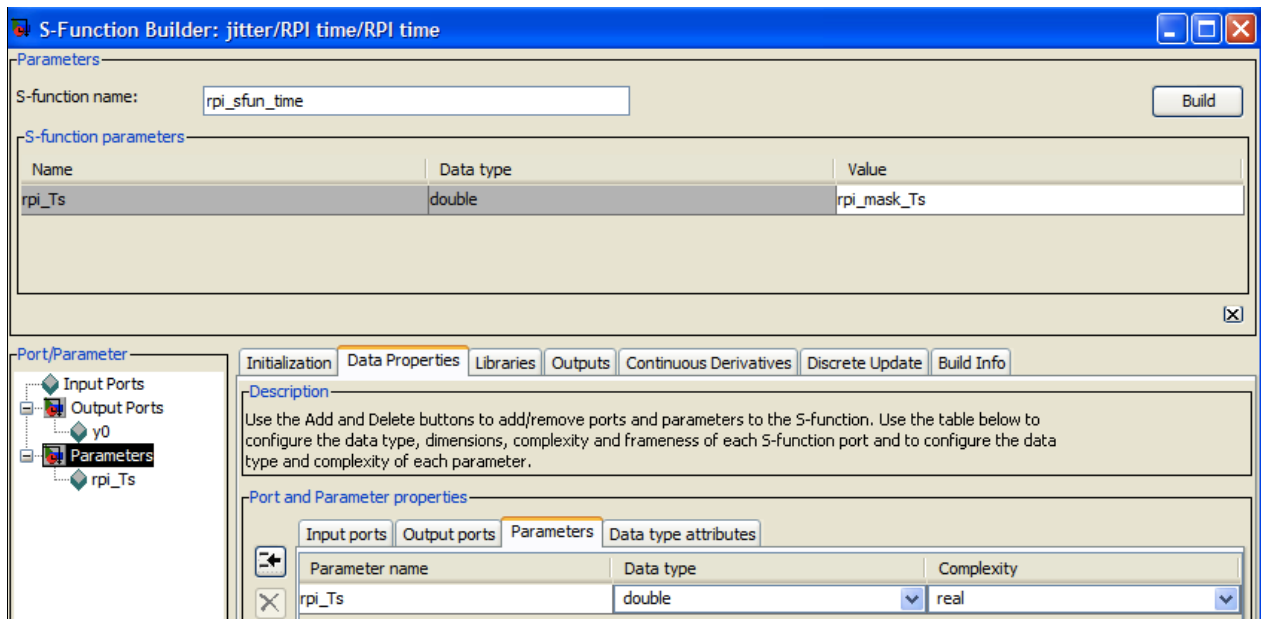


Figure 14 - Defining the sampling time parameter

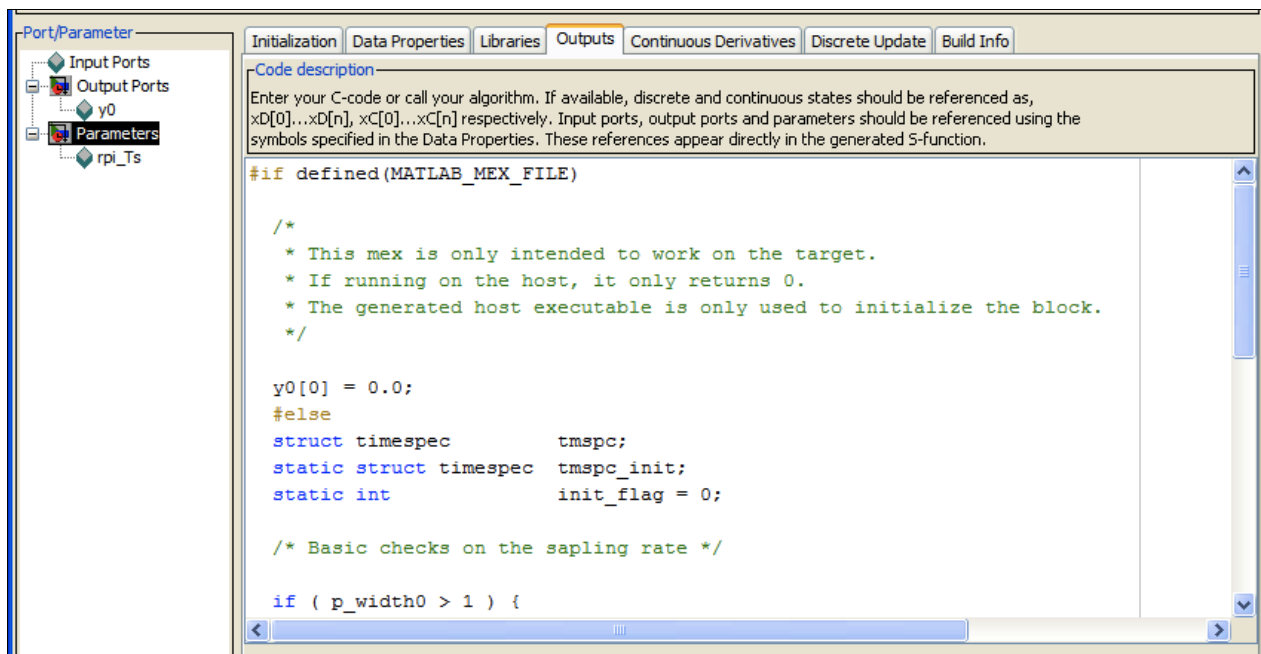


Figure 15 - Coding your S-function

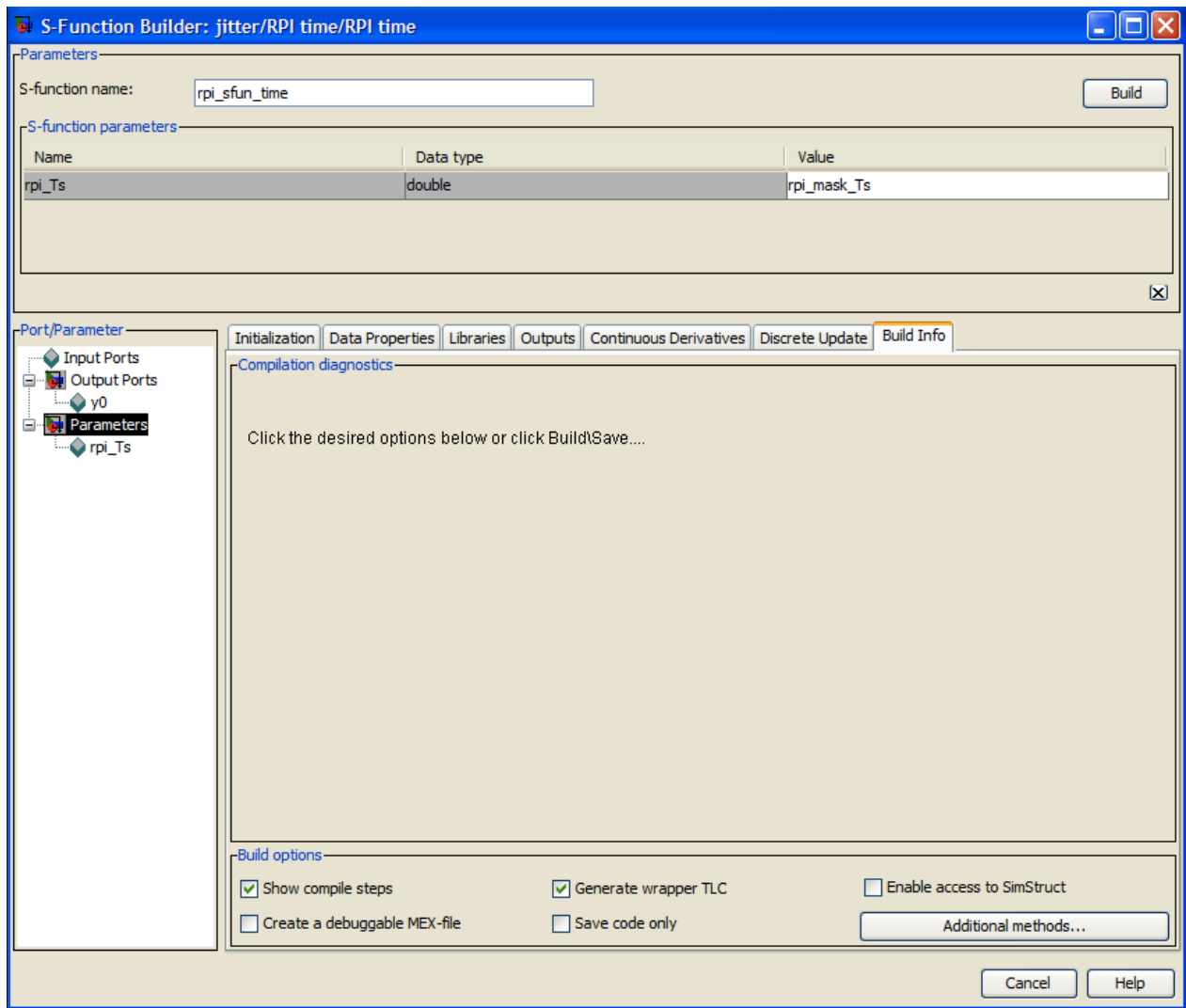


Figure 16 - S-function build options

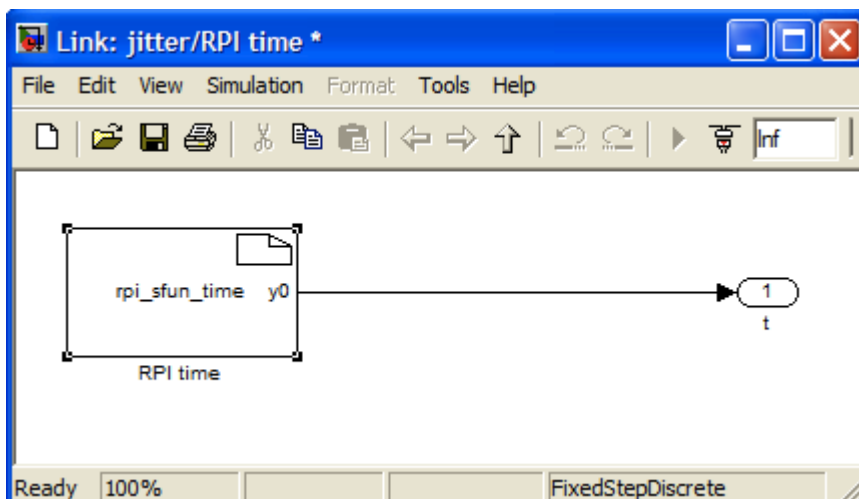


Figure 17 - Insert S-function into a subsystem

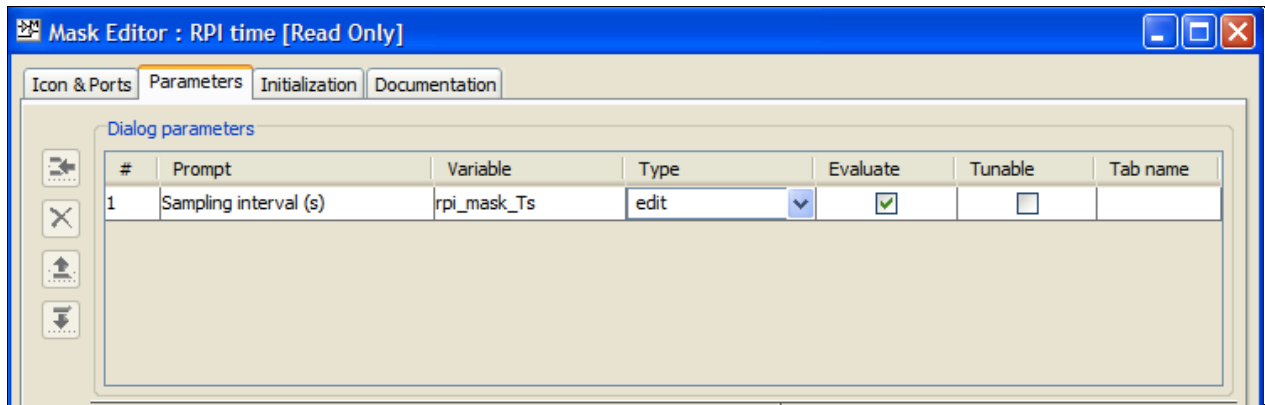


Figure 18 - Edit S-function mask

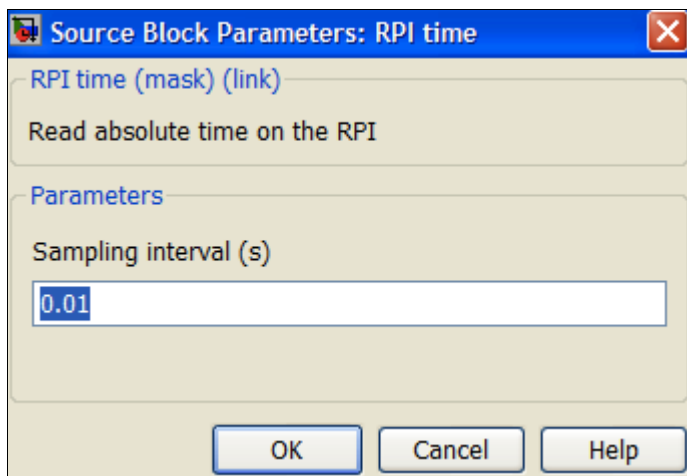
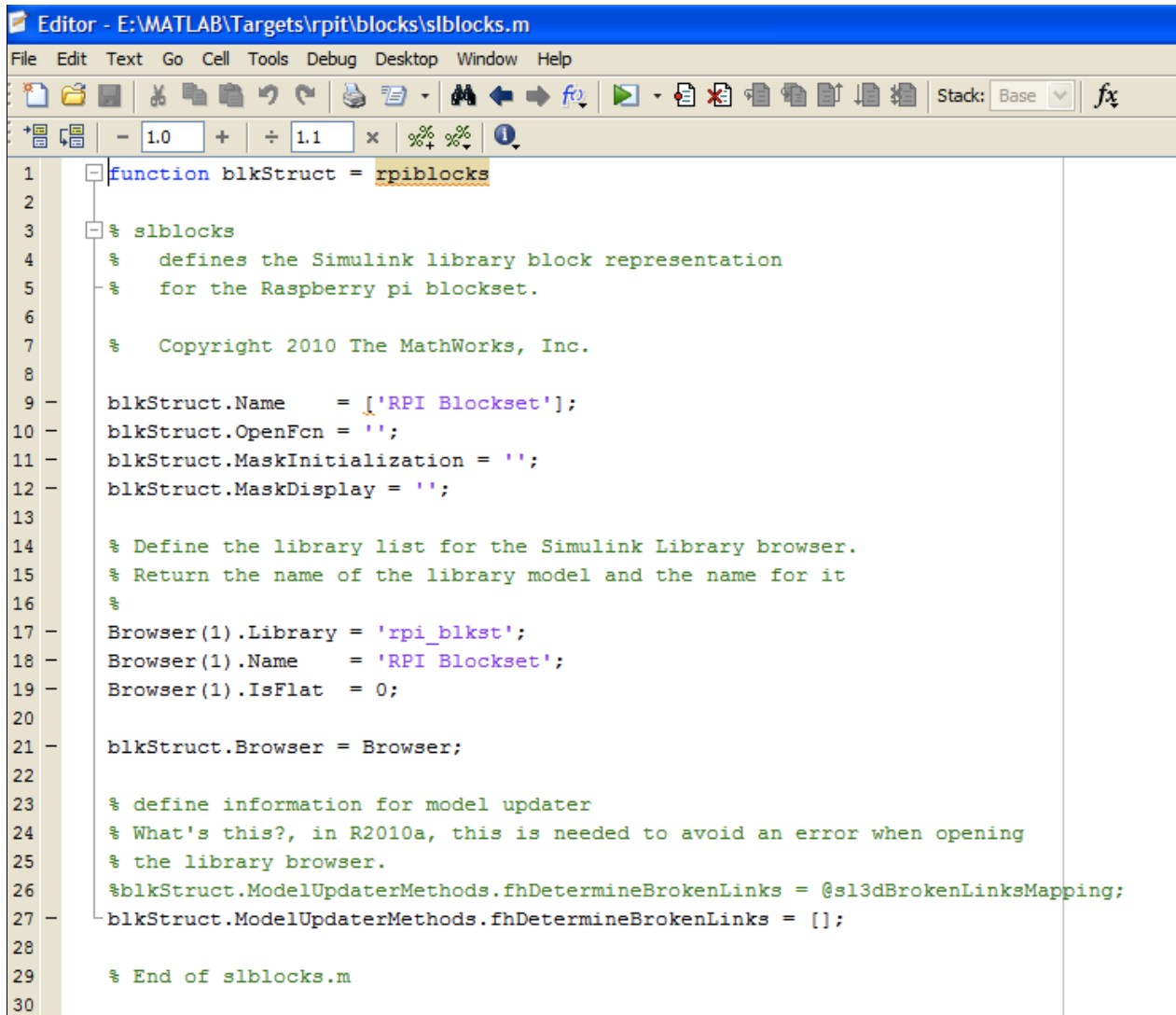


Figure 19 - S-function custom configuration dialog



```

1 function blkStruct = rpiblocks
2
3 % slblocks
4 % defines the Simulink library block representation
5 % for the Raspberry pi blockset.
6
7 % Copyright 2010 The MathWorks, Inc.
8
9 blkStruct.Name = ['RPI Blockset'];
10 blkStruct.OpenFcn = '';
11 blkStruct.MaskInitialization = '';
12 blkStruct.MaskDisplay = '';
13
14 % Define the library list for the Simulink Library browser.
15 % Return the name of the library model and the name for it
16 %
17 Browser(1).Library = 'rpi_blkst';
18 Browser(1).Name = 'RPI Blockset';
19 Browser(1).IsFlat = 0;
20
21 blkStruct.Browser = Browser;
22
23 % define information for model updater
24 % What's this?, in R2010a, this is needed to avoid an error when opening
25 % the library browser.
26 blkStruct.ModelUpdaterMethods.fhDetermineBrokenLinks = @sl3dBrokenLinksMapping;
27 blkStruct.ModelUpdaterMethods.fhDetermineBrokenLinks = [];
28
29 % End of slblocks.m
30

```

Figure 20 - Creating a custom block library

Testing the RPI target

Now it's time to test the whole process.

1. Go into the "demo" directory and open the "jitter" model (Figure 21).
2. Open "Simulation->Configuration Parameters..." and go to the "Real-Time Workshop->Interface" pane. In the "MEX-file arguments" field, enter the IP address of your Raspberry pi as shown in Figure 22. Click "OK" button. Note that the single quotes around the IP address are mandatory.
3. Check that the current Matlab directory is the one containing "jitter.mdl".
4. Run the process with the shortcut "[CTRL]-B" in the "jitter" Simulink model window.
5. You should end up with messages in Matlab's command window instructing about the state of the process (Figure 23).
6. Double-click on the scope and you should have an idea of your RPI jitter. Indeed, the digital clock count the number of time steps and multiply it by the sampling time. It is the theoretical time. Whereas the RPI time S-function block reads directly the absolute time on the RPI hardware timer. The difference is the jitter. In Figure 24, the peak-to-peak difference is around 200us. There

is an offset due to the time needed to initialize the absolute time offset at the beginning of the simulation.

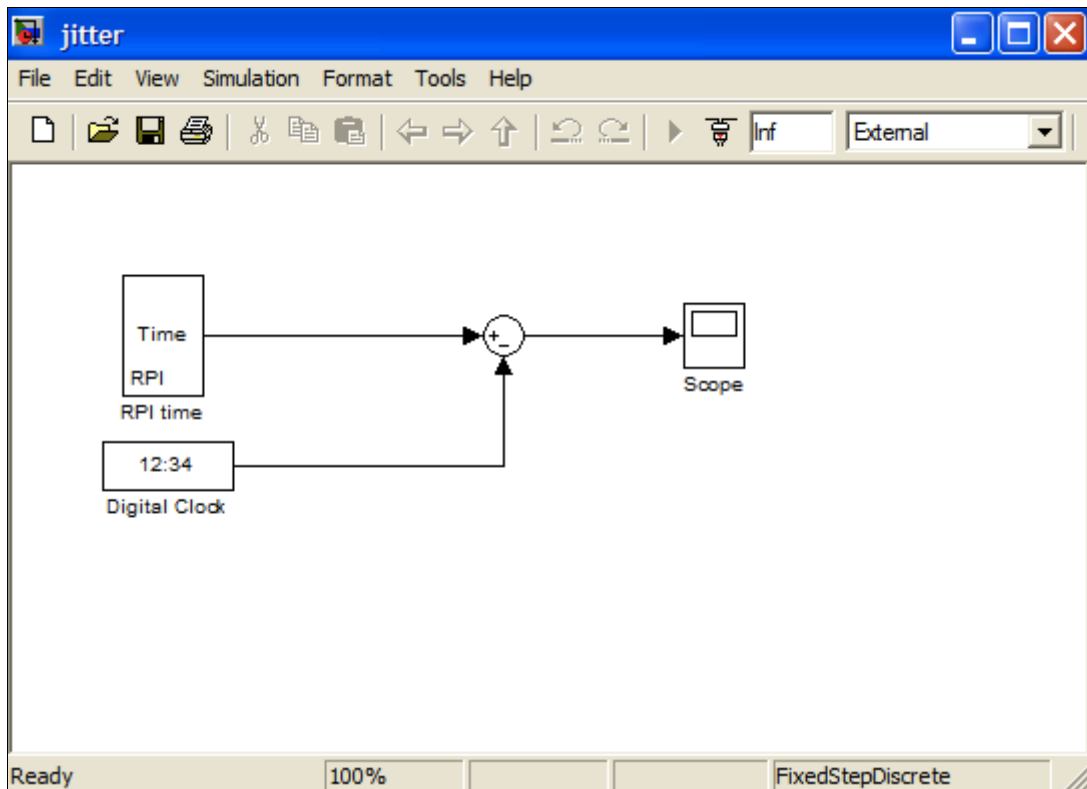


Figure 21 - The jitter demo model

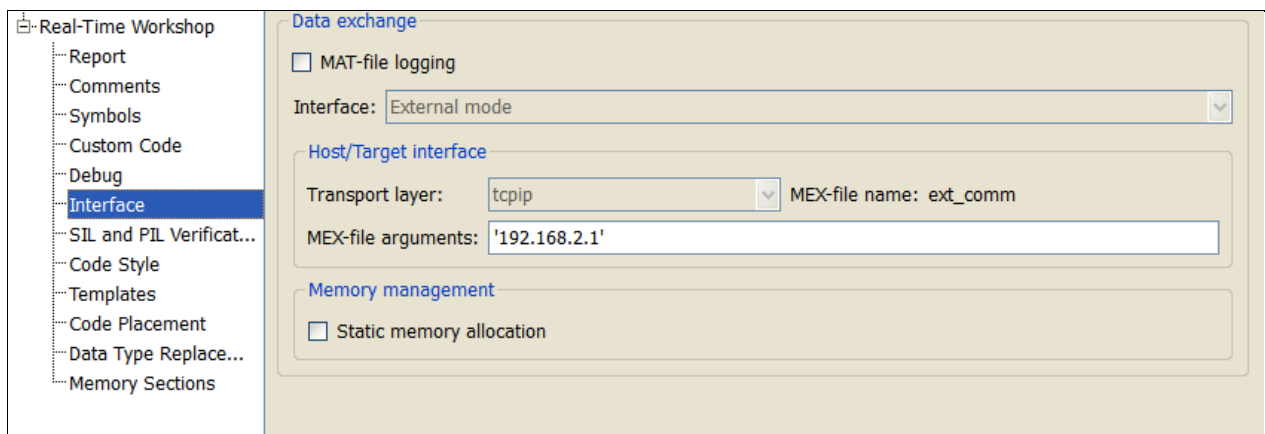


Figure 22 - Defining the RPI IP address

```

Command Window
### TLC code generation complete.
### Generating TLC interface API.
.
### Creating data type transition file jitter_dt.h
.### Evaluating PostCodeGenCommand specified in the model
.
### Processing Template Makefile: E:\MATLAB\Targets\rpit\rpit\ert_rpi.tmf
### Wrapping unrecognized make command (angle brackets added)
### <\usr\bin\make>
### in default batch file
### jitter.mk which is generated from E:\MATLAB\Targets\rpit\rpit\ert_rpi.tmf is up to date
### Make will not be invoked - template makefile is for a different host
### Kill pending rtw process and clean RTW directory.
### Uploading jitter to the Raspberry PI.
### Compiling jitter on the Raspberry PI (may take awhile).
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -c -std=gnu99 -pedantic -O0 -ffloat-store -fPIC -DEXT_MODE -DGLNX -DMODEL=jitter -DNUMST=1 -DNCSTATES=0 -DUNIX -DMAT_FILE=0 -DINI
gcc -lpthread -lrt -o ../jitter -lm rpi_sfun_time_wrapper.o jitter_data.o jitter.o ext_svr.o updown.o ext_work.o rtiostream_interfac
### Created executable: ../jitter

### Compilation successful. Starting model.
### Real-time code successfully started on the RPI.
### Simulink started and running in external mode.
### Successful completion of Real-Time Workshop build procedure for model: jitter
fx >>

```

Figure 23 - Target compilation process

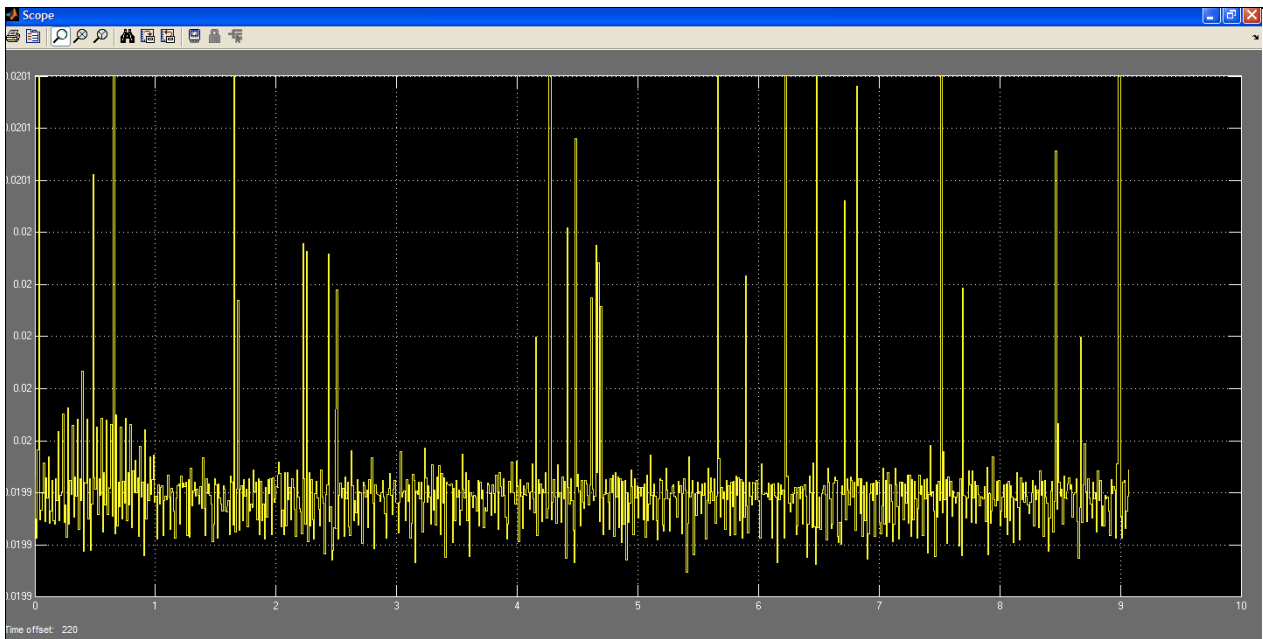


Figure 24 - Jitter of a RPI

Contact

I'm professor in robotics at the University of Strasbourg. You can contact me by mail at this address: jacques.gangloff@unistra.fr. I would be delighted if you contact me to submit bugs or improvements or even additional blocks for the RPI. Unfortunately, my schedule doesn't allow me to respond quickly to more prosaic questions like "It does not work! Why?"