

Getting Started with

RapidiTTy  FPGA

RapidiTTy FPGA 2.0

TTE Systems

Rapid development of reliable embedded systems

<http://www.tte-systems.com>

Version

Getting Started with RapiDiTTy FPGA v2.0 (November 2009)

Copyright

This document is copyright © TTE Systems Limited 2007-2008. All rights reserved.

Trademarks

ARM™ and Cortex™ are registered trademarks of ARM Limited.

Cygwin™ is a registered trademark of Red Hat, Inc.

Eclipse™ and Built on Eclipse™ are trademarks of the Eclipse Foundation, Inc.

GNU™ is a registered trademark of the Free Software Foundation.

Linux™ is a registered trademark of Linus Torvalds.

NXP™ is a trademark of NXP Semiconductors

RapiDiTTy® and TTE® are registered trademarks of TTE Systems Ltd.

TTE Builder™ and TTE Debug™ are trademarks of TTE Systems Ltd.

Sun®, Java™ and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc.

Windows® is a registered trademark of Microsoft Corporation.

Xilinx®, ISE™, Spartan™, Virtex™ and WebPACK™ are trademarks or registered trademarks of Xilinx, Inc.

All other trademarks are acknowledged.

Table of Contents

1	Introduction.....	5
2	Installation and hardware requirements.....	5
2.1	Installation.....	6
2.1.1	Installing and configuring Xilinx ISE.....	6
2.1.2	Installing and configuring Digilent Adept Suite.....	6
3	Working with projects.....	8
3.1	Workspaces.....	8
3.2	Project creation.....	9
3.3	Project properties.....	12
4	Testing code on the simulator.....	15
5	Adding new hardware peripherals.....	17
5.1	Adding a new peripheral.....	17
5.2	Adding code to use the new peripheral.....	17
5.3	Simulating the design.....	19
6	Gathering timing statistics.....	20
7	Porting a project to work on hardware.....	21

1 Introduction

RapidiTTY FPGA is a complete environment designed to help software developers exploit the full potential of “soft” processor cores and field-programmable gate arrays. Based on the highly regarded Eclipse framework, it offers the following key benefits:

- Designed for *software* developers who have worked previously with low-level microcontrollers or embedded PCs.
- Includes the popular PH 03 soft processor and tools to configure this core: with just a few clicks of a mouse you can create a processor which matches the *precise* needs of your application.
- No need to learn VHDL (but full VHDL source for the PH 03 processor is provided for situations where you want low-level control).
- Provides extensive debug support (such as breakpoints and single-stepping), fully integrated with the IDE.
- Includes a complete simulation environment, to help speed up your development process. Full debug support is provided in the simulator too.
- Provides exceptional support for developing time-triggered¹ systems.
- Includes source code for two complete low-level operating systems as well as numerous example projects.
- Minimises the effort involved in precise timing analysis (including worst-case execution time measurements).

In this tutorial, we will see how these features work in RapidiTTY FPGA by using them to develop a simple embedded system. First, however, RapidiTTY FPGA must be installed and configured to work with the hardware.

2 Installation and hardware requirements

RapidiTTY FPGA includes an automated installer that should take care of all the necessary software installation tasks. In addition, we must connect and configure a development board for the target embedded hardware platform.

¹ More information about time-triggered systems can be found online at the TTE Systems website: <http://www.tte-systems.com/books/rdres>.



In this tutorial, we will target the popular Xilinx Spartan family, which are mid-range FPGA devices. More specifically, we will be using the Spartan-3 starter board from Digilent.²

2.1 Installation

While RapidITTy FPGA is very easy to install (simply run the installation executable), we rely on several third party applications in order to support their specific hardware.

2.1.1 Installing and configuring Xilinx ISE

To compile the bundled soft core for Xilinx hardware, RapidITTy FPGA requires the use of the Xilinx ISE design tools. A free (and unrestricted) version, the Xilinx ISE WebPACK, is available for download from the Xilinx website.³ RapidITTy FPGA is fully compatible with this version, as it is with any version from 9.1 and above.

Before using RapidITTy FPGA for SoC development, it may be necessary to run Xilinx ISE at least once to ensure that all the necessary configuration steps have taken place. Once the application has successfully opened, the configuration will have completed and it can be closed again. It is only necessary to do this once, after the tool has been installed.

2.1.2 Installing and configuring Digilent Adept Suite

In order to upload compiled software and hardware to Digilent FPGA development boards, RapidITTy FPGA requires the use of the Digilent Adept Suite of tools and drivers. These are available from Digilent's website; please refer to Digilent's documentation for installation details.

The Digilent Adept Suite contains a utility called "ExPort", which controls the configuration and programming of Digilent FPGA boards through a JTAG connection. In order for RapidITTy FPGA to program the board, the correct configuration must be setup in ExPort. The connection box in ExPort is shown in Figure 2-1, below.

Note that this configuration must be updated whenever a different development board (or JTAG connection) is used. This process also requires administrator-level access, or you may not be able to save any changes that are made.

² Available online from the Digilent website: <http://www.digilentinc.com/>.

³ Available from: <http://www.xilinx.com/ise/>. WebPACK does not support the XC4VSX35 Virtex-4 device.

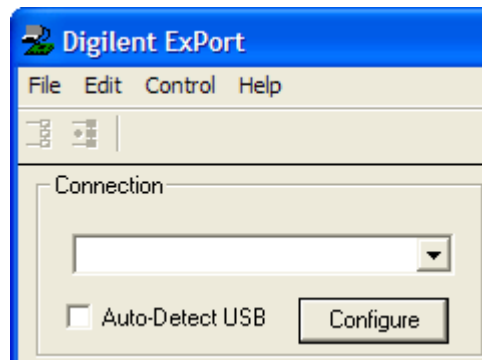


Figure 2-1: The connection box in Digilent ExPort.

To configure RapiDiTTY FPGA for the current board, first ensure the device is plugged in and drivers are installed, then start ExPort and uncheck “Auto-Detect USB” in the connection box (shown in Figure 2-1). Select the “Configure” button and then “Add module”. Select the tab for your chosen connection type and pick the device from the list (required for USB connections). Saving the changes and then pressing “Initialize Chain” will complete the configuration, providing the view shown in Figure 2-2.

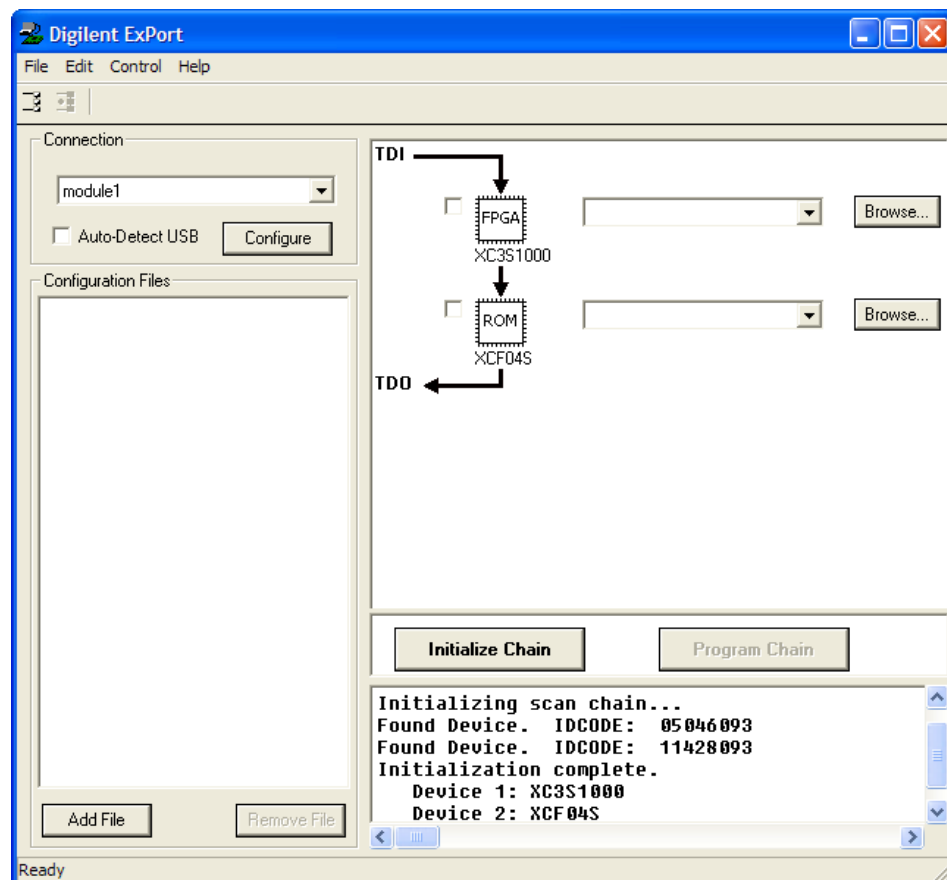


Figure 2-2: The configured device in Digilent ExPort.

At this point the ExPort configuration is complete and RapidITy FPGA should be able to program the device correctly. A list of potential problems with this process, along with their solutions, is shown in Figure 2-3.

Problem	Solution
<p>“Dpcutil.dll is already in use” Message given when attempting to run the Digilent ExPort application</p>	<p>Ensure that RapidITy FPGA is closed before attempting to run Digilent ExPort</p>
<p>“Unable to save changes” Message given when attempting to save a new configuration in Digilent ExPort</p>	<p>Ensure that you run Digilent ExPort from an account with administrator-level access privileges</p>
<p>“Please make sure that the right development board is powered up and is connected to the computer with the right cable” Message given when attempting to upload from RapidITy FPGA</p>	<p>Assuming the development board is plugged in and powered up, the FPGA targets may require further configuration with Digilent ExPort. Also ensure that the correct “Debug Device” is set in the Debug or Run launch configuration dialogs in RapidITy FPGA</p>

Figure 2-3: Troubleshooting the Digilent ExPort configuration.

3 Working with projects

Suppose we want to develop an embedded application utilising an unusual or highly customised combination of hardware peripherals. In this tutorial, we will consider an embedded system that outputs the elapsed time since it was switched on to another device over a standard RS-232 serial link.

We will then build on this system to additionally display the elapsed time on a standard VGA computer monitor.

3.1 Workspaces

The first time we start RapidITy FPGA, we are presented with the workspace selection dialog shown in Figure 3-1, below.

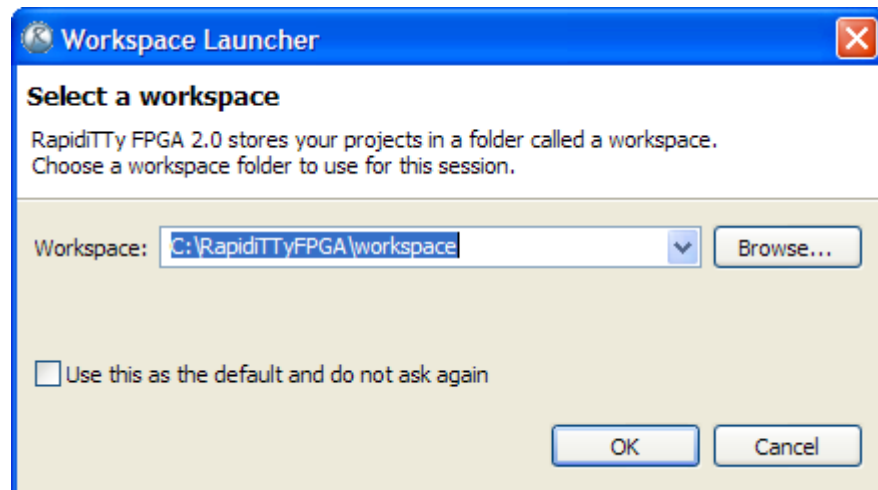


Figure 3-1: The workspace selection dialog.

For this tutorial, we will be using "C:\RapidITyFPGA\workspace".

3.2 Project creation

Create new projects by selecting "New → RapidITy Project" on the File menu.

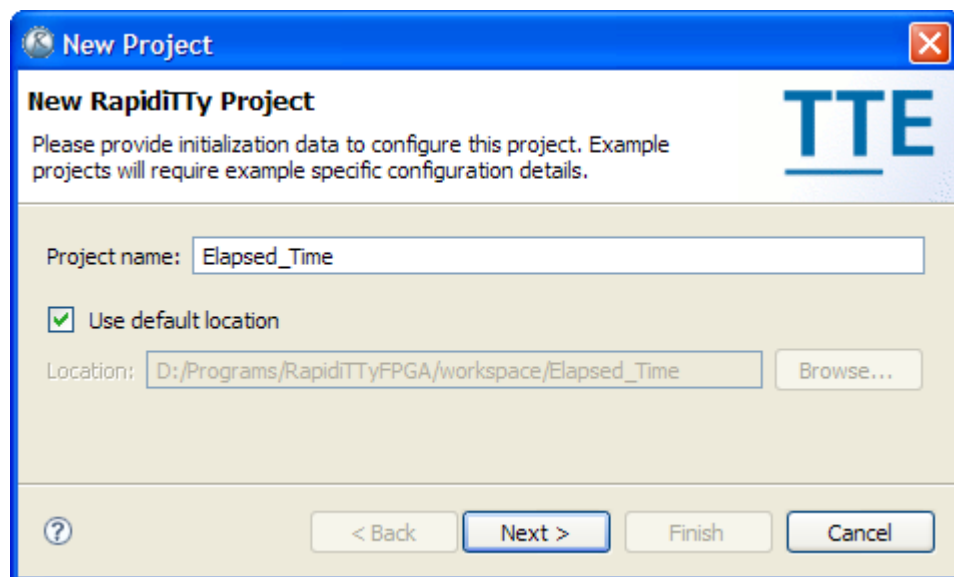


Figure 3-2: The new project dialog.

As shown in Figure 3-2, we have chosen to name the project "Elapsed_Time". This name will also be the default name of the compiled binary, should it be required outside of RapidITy FPGA. After selecting a name, we are asked to select the development board to target, as shown in Figure 3-3.

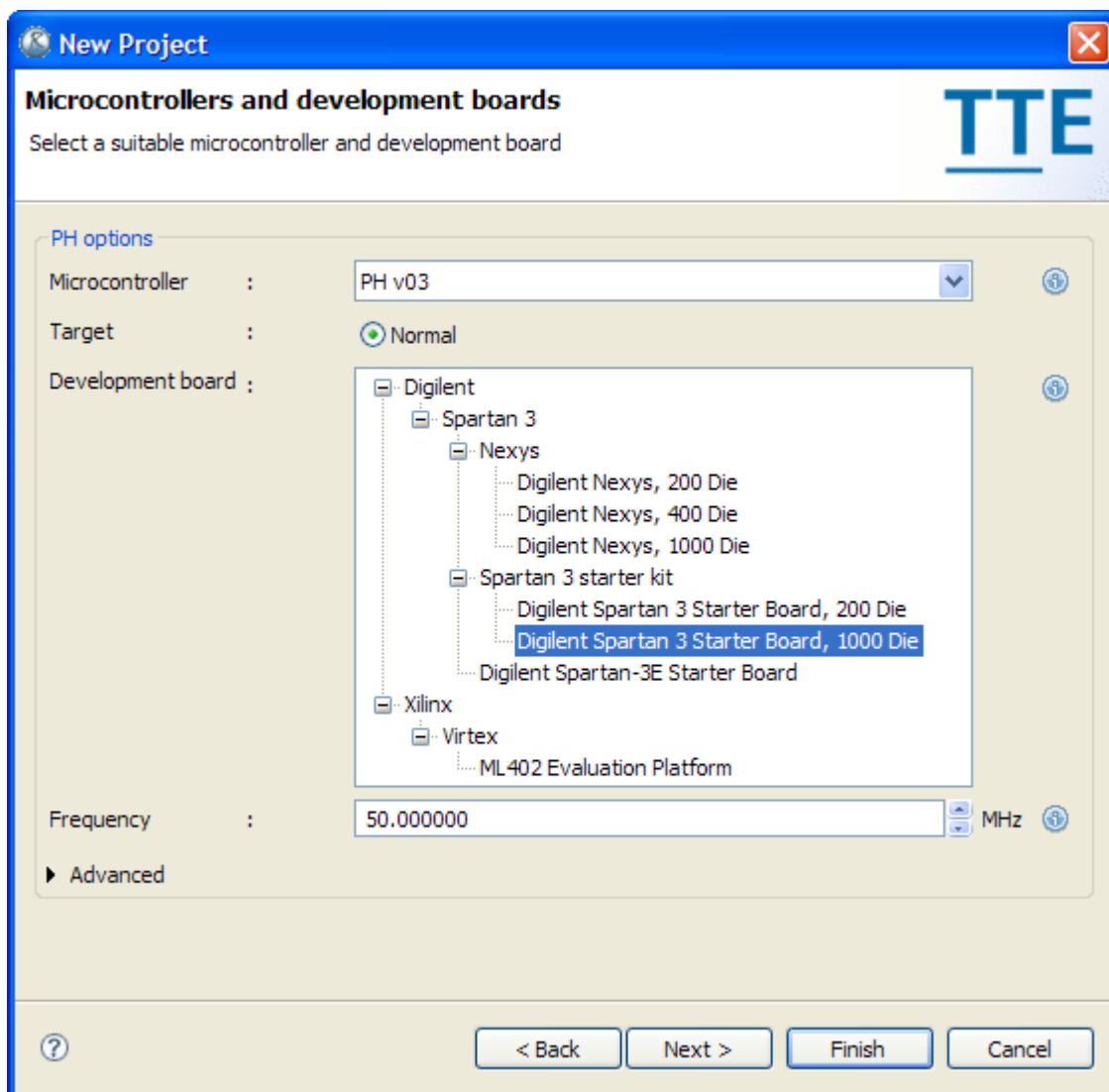


Figure 3-3: Selecting the development board.

The dialog shown in Figure 3-3 prompts for the choice of microcontroller, which is the “soft” processor core that will be configured for the FPGA development board chosen in the same dialog. There is one such soft core that ships with RapidiTTY FPGA: the PH processor. This is a full 32-bit processor core that is compatible with the MIPS I Instruction Set Architecture (but without the patented instructions).

The following dialog, shown in Figure 3-4, allows us to choose code and data sizes as well as the initial peripherals that should be included in the design.

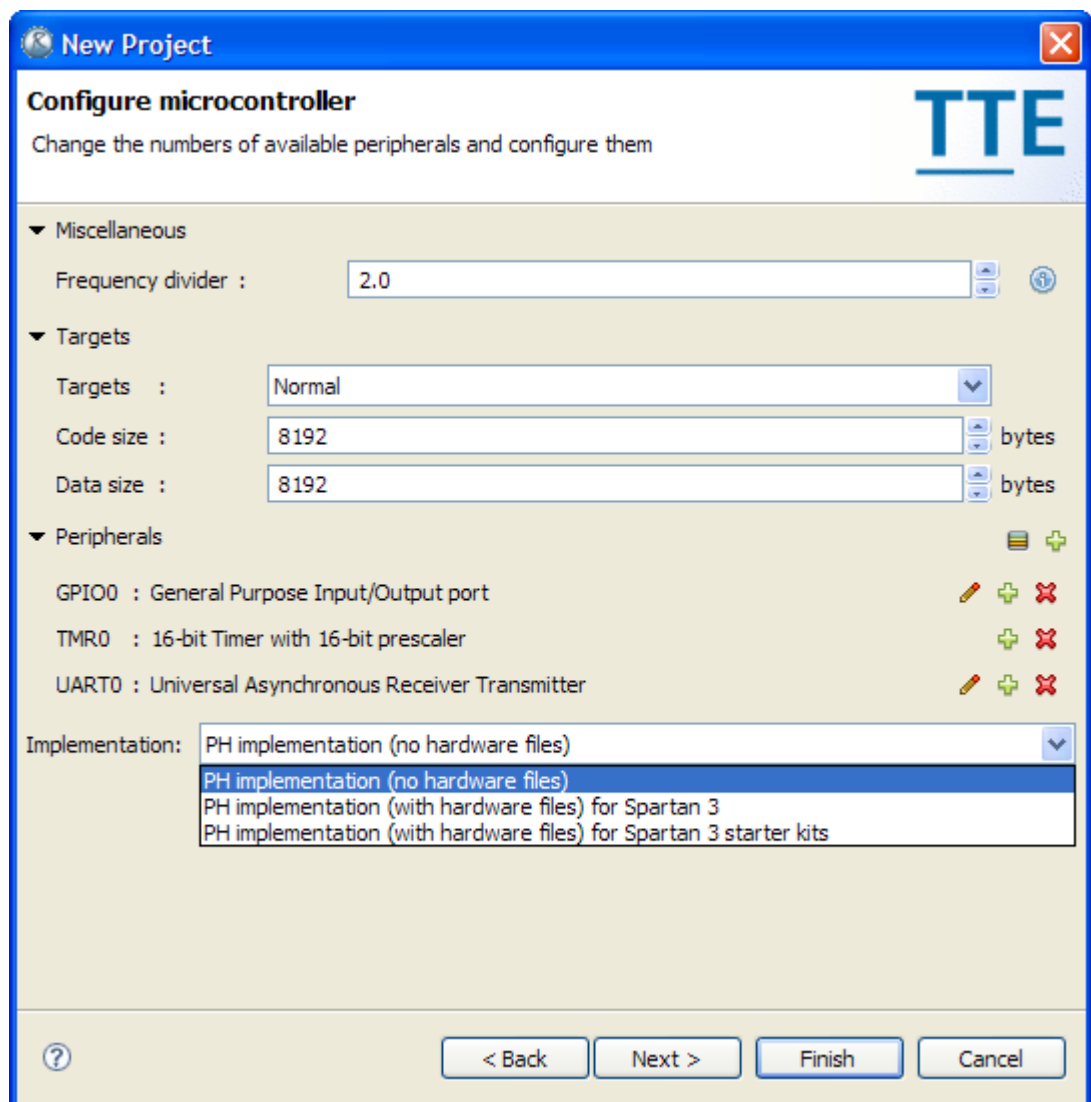


Figure 3-4: Configuring the microcontroller.

The configuration dialog in Figure 3-4 also shows the selection of the implementation method for the design. In this example, these are as follows:

- **No hardware** – produces a project with no hardware files, suitable for simulation.
- **Spartan 3** – produces a project suitable for generic Spartan 3 hardware.
- **Spartan 3 starter kits** – produces a project suitable for the starter kit development boards, such as those provided by Digilent.

Finally, the project creation wizard will prompt for you to optionally select an example to generate for the design, as shown in Figure 3-5.

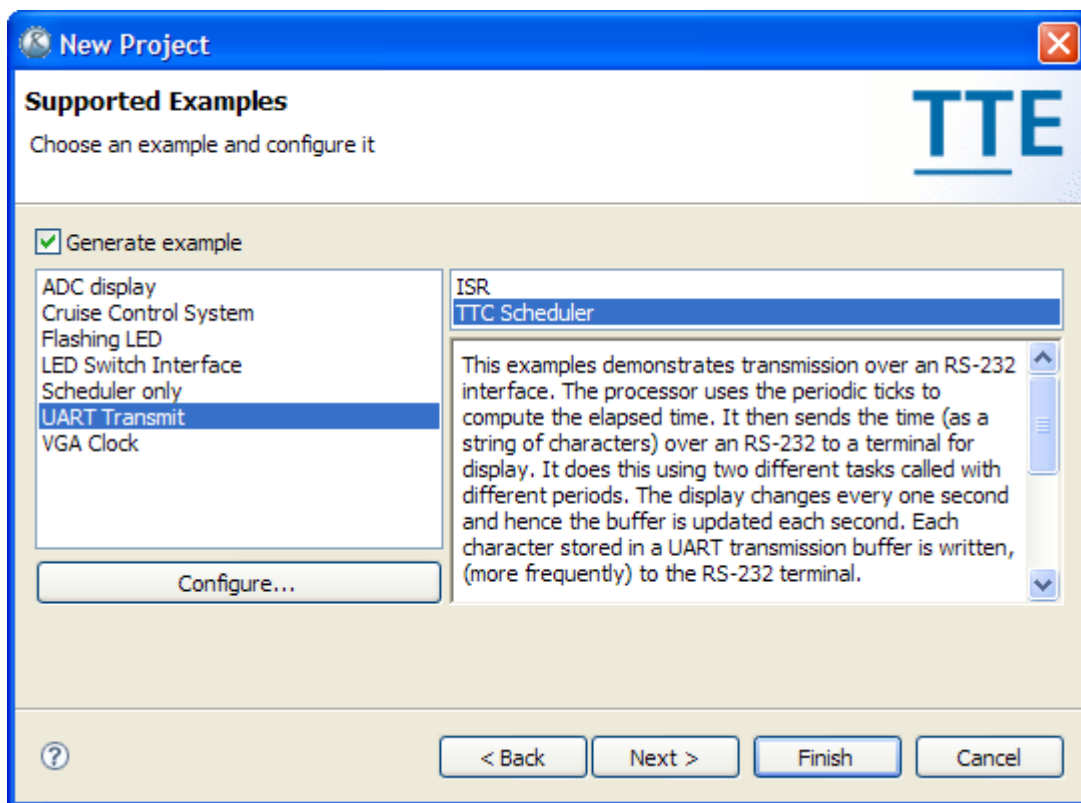


Figure 3-5: Generating an example for the design.

For this tutorial we will start with a “UART Transmit” project, with all the hardware settings and peripherals left at their defaults and using an implementation with no hardware files. The development board used here is the “Digilent Spartan 3 starter board, 1000 die”, but any board of similar specification could be used.

3.3 Project properties

Once the project has been created, the project properties page is displayed. This is where settings can be altered, and the project can be built as shown in Figure 3-6.

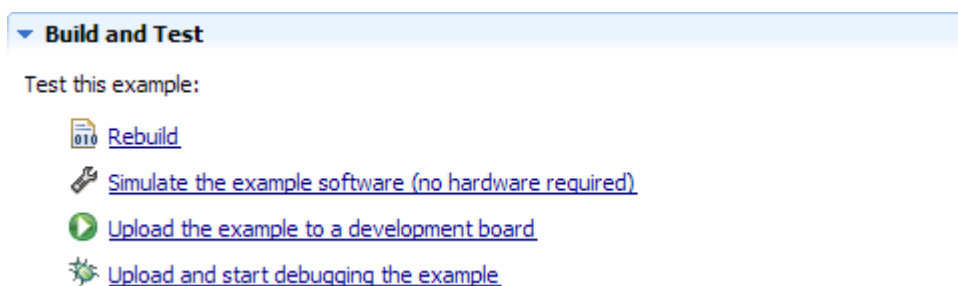


Figure 3-6: The build and test section of the project properties.

The project properties also allows setting of options related to the example we have just generated. These are shown in Figure 3-7.

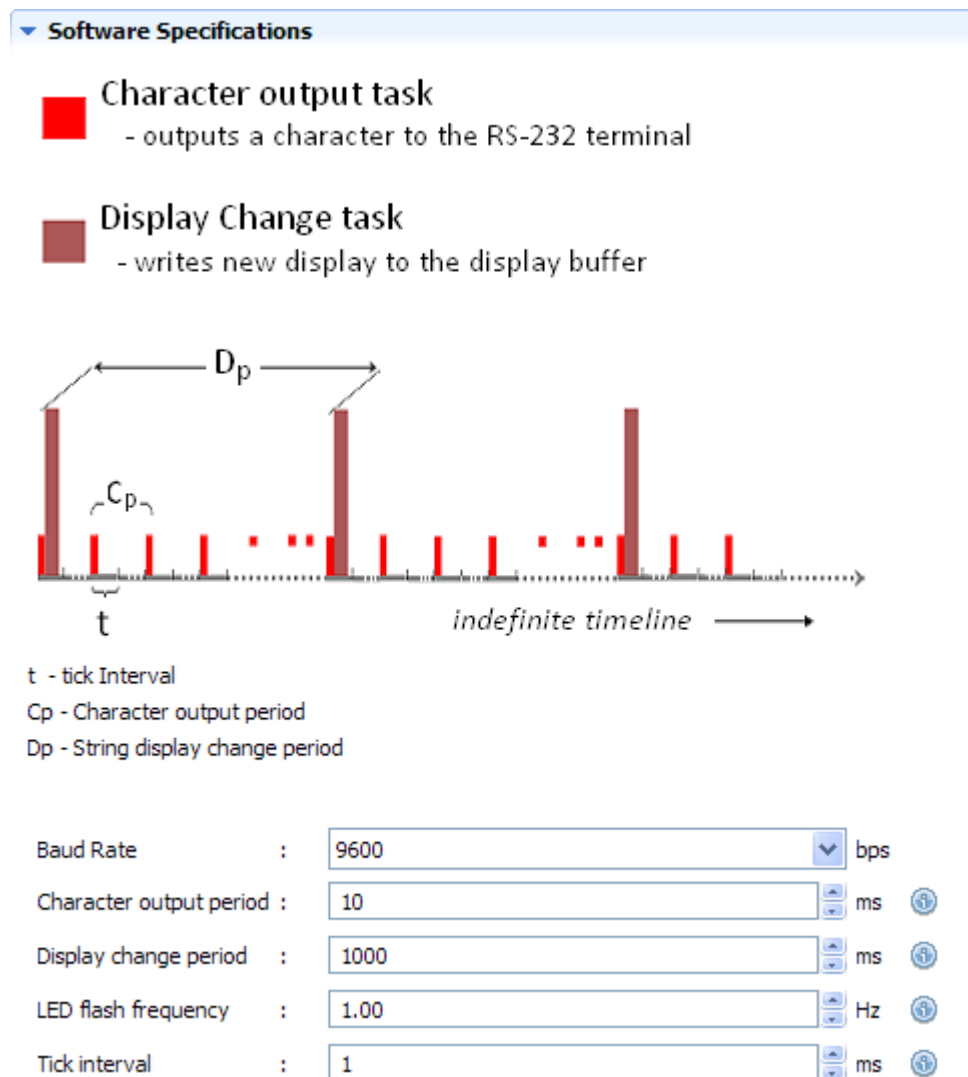


Figure 3-7: Specifications and settings for the “UART Transmit” example.

In this case, the “UART Transmit” example has the following configuration settings:

- **Baud rate** – the rate at which individual bits are sent over the serial link.
- **Character output period** – how frequently characters are sent from the buffer.
- **Display change period** – how frequently the buffer is updated with new data.
- **LED flash frequency** – a flashing LED is employed as an easy demonstration that the hardware is functioning and configured correctly.
- **Tick interval** – a “time-triggered” scheduler runs all tasks with this fixed period.

The hardware specifications can also be changed from the project properties, as shown in Figure 3-8.

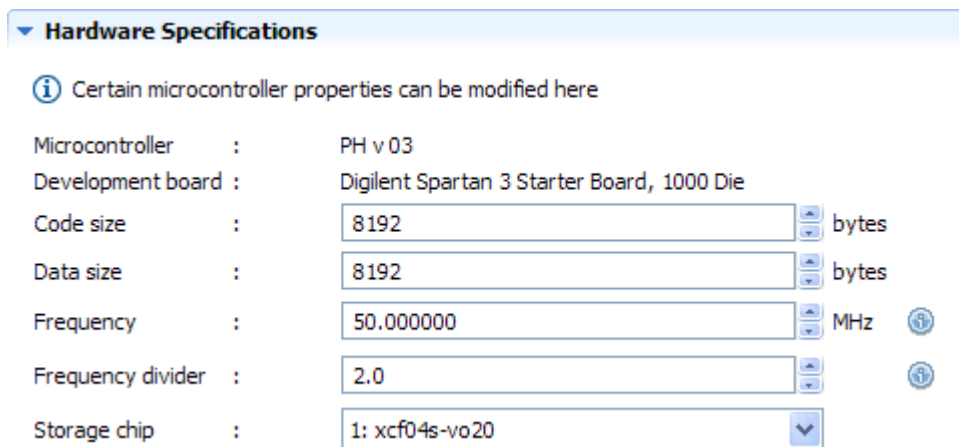


Figure 3-8: Altering hardware specifications in the project properties.

These are essentially the same settings as were selected in the new project wizard. Note that changes made to the project properties page must be saved before they are committed.

Finally, the project properties page allows us to add and remove peripherals, as shown in Figure 3-9.

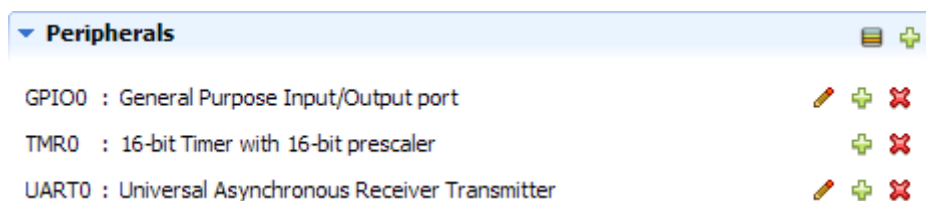


Figure 3-9: Adding and removing peripherals in the project properties.

New peripherals can be added using the green “plus” button at the top left of the section, or by duplicating existing peripherals (with their individual plus buttons). Likewise, the red “cross” buttons allow individual peripherals to be removed and the “pencil” buttons allow us to edit their settings.

We will be using this functionality later on (in Section 5 on page 17) to add a VGA controller to the design.

4 Testing code on the simulator

Without hardware files, the only way we can test the design is to use the built-in simulator. To start a simulation, simply select the simulation link from the build and test section of the project properties, as shown in Figure 3-6. We are then presented with the debugging perspective, as shown in Figure 4-1.

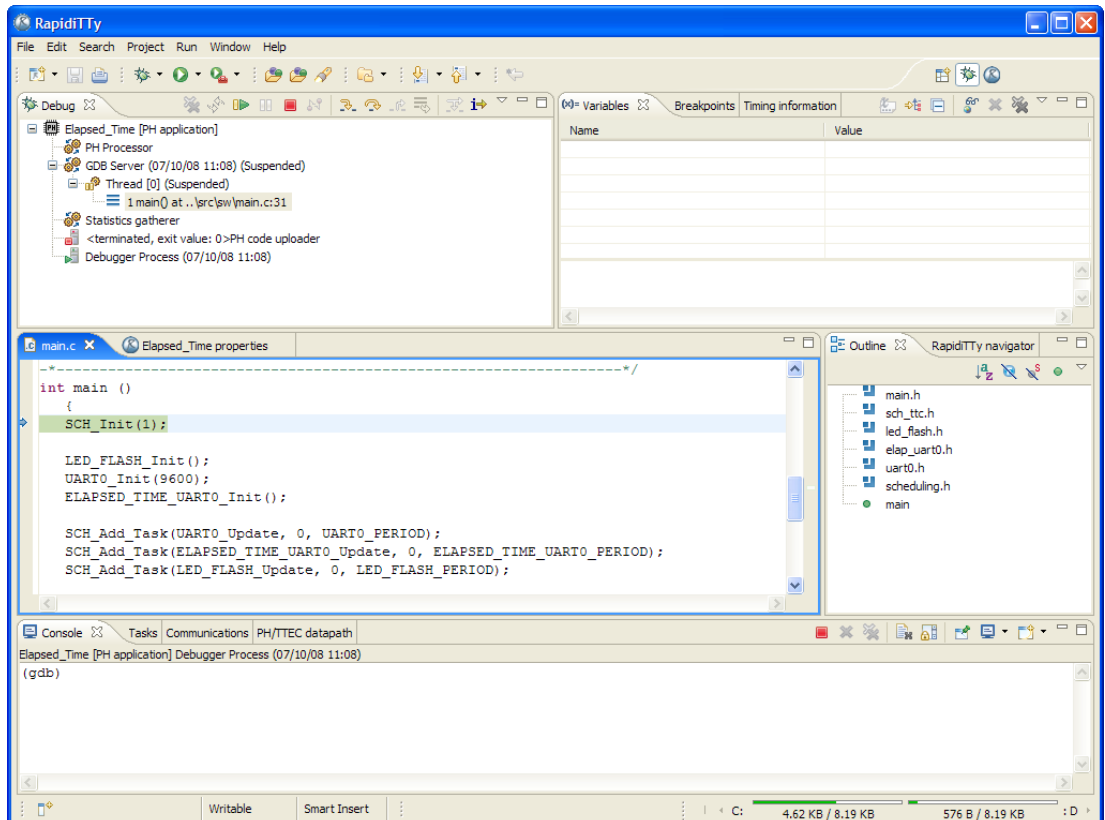


Figure 4-1: Debugging a project, stopped at the entry point.

Once in the debugging perspective, RapidITTy FPGA will halt execution at the first line of the `main` function. We can step through the code line-by-line, manage breakpoints or allow execution to resume unhindered from here using the controls in the views at the top of the window (labelled “Debug”, “Variables” and “Breakpoints”).

Because the example we are testing is designed to output a value over the serial port, it would be nice if we could observe this while debugging. To do so, we must first configure the UART peripheral in the simulator. While in the debug perspective, bring up the peripherals view by selecting “Window → Show View → Peripherals”. This view is shown in Figure 4-2.

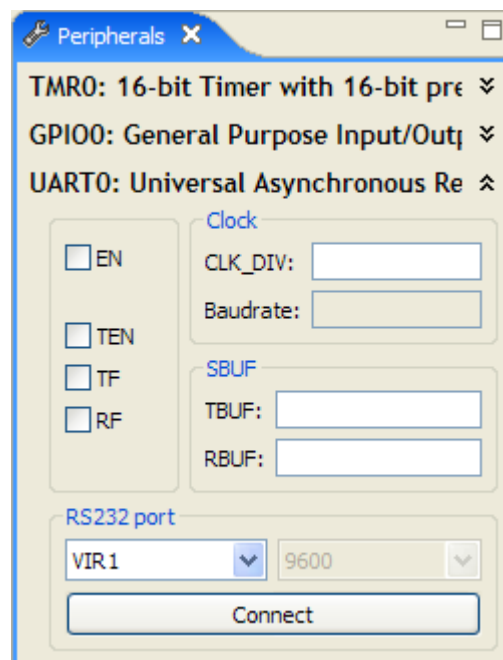


Figure 4-2: The peripherals view – configuring a UART.

By default, the UART will be configured as the virtual RS232 port named “VIR1”, but will not be connected to it. To remedy this, simply press the “Connect” button. We can view the output from the communications view, located at the bottom of the debug perspective and shown in Figure 4-3.

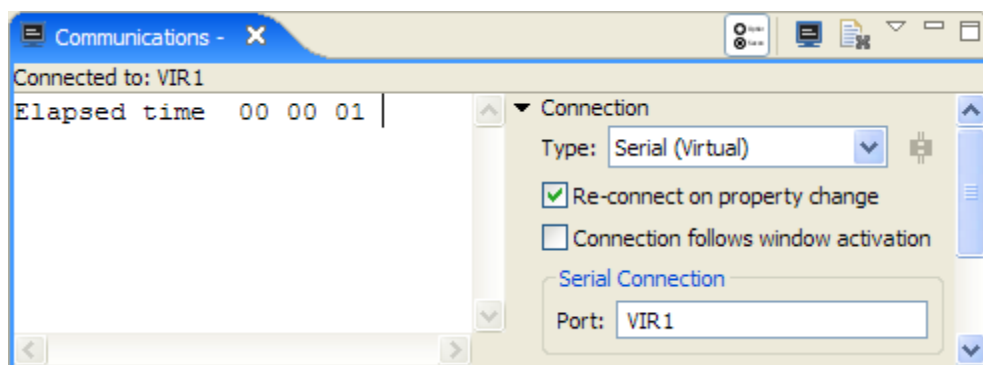


Figure 4-3: The communications view – connected to “VIR1” and showing output.

When the communications view has been set up as shown in Figure 4-3, we should see the output from the application (if not, ensure the application is running by pressing the “continue” button in the “Debug” view). In this case it is a string, updated once per second, containing the elapsed time since the device was last powered up or reset.

At this point, it is clear that the design (both hardware and software) is functioning correctly.

5 Adding new hardware peripherals

Now that we have the basic example up and running, it is time to extend the design by adding a new hardware peripheral and the software to use it.

5.1 Adding a new peripheral

Going back to the peripherals section of the project properties page (preferably by first returning to the RapidITy perspective with the button in the top-right of the window), we can use the green “plus” button to add a new peripheral as shown in Figure 5-1.

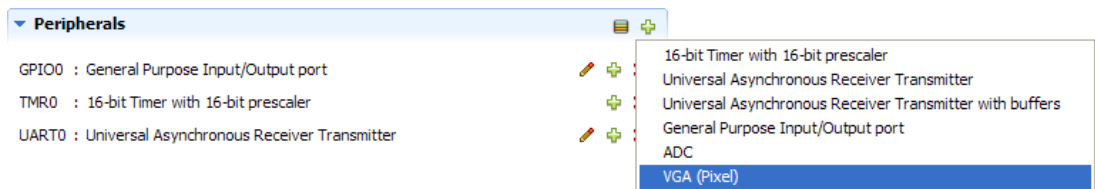


Figure 5-1: Adding a VGA controller peripheral to the system.

Adding a VGA controller, as shown in Figure 5-1, will automatically add the source-code for the drivers necessary to operate it. When the drivers are added and the project is rebuilt (which may happen automatically, depending on your settings), we can see the effect of the additional code in the meters at the bottom-right of the window, shown in Figure 5-2.

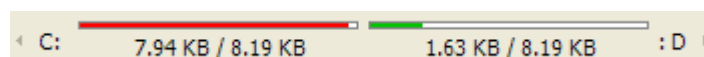


Figure 5-2: Memory meters, showing that we are almost out of code space!

The VGA driver includes some relatively large arrays used for drawing characters, so there has been a significant increase in the code size. If necessary, we can increase the amount of code memory available in the project properties.

5.2 Adding code to use the new peripheral

In order to display the elapsed time on our new VGA peripheral, we need to add some source-code to “`elap_uart0.c`”. First, at the top of the file (below the existing “`#include`” directives) we need the code shown in Figure 5-3. This provides access to the VGA drivers, and sets up some constants to centre our output string in the display.

```
#include "vgap0.h"

// Coordinates to centre the string in the display
#define TIME_STRING_CHARS 22
#define STRING_WIDTH      TIME_STRING_CHARS * VGAP0_CHAR_WIDTH
#define STRING_HEIGHT     VGAP0_CHAR_HEIGHT
#define STRING_X          (VGAP0_WIDTH - STRING_WIDTH)/2
#define STRING_Y          (VGAP0_HEIGHT - STRING_HEIGHT)/2
```

Figure 5-3: Code to be added to “eLap_uart0.c” – sets up some constants.

Next, we must alter the “ELAPSED_TIME_UART0_Init” function to also initialise the VGA display with starting values. We also switch from white-on-black text to a black-on-white, in order to allow easier viewing and printing in this manual. This source-code is shown in Figure 5-4.

```
void ELAPSED_TIME_UART0_Init(void)
{
    char time_str[] = "Elapsed time";

    UART0_Write_String_To_Buffer(time_str);

    VGAP0_Init();

    // Switch to black on white, for screenshot readability in the manual
    VGAP0_Foreground = VGAP0_CLR_BLACK;
    VGAP0_Background = VGAP0_CLR_WHITE;
    VGAP0_Fill_Rectangle(0, 0, VGAP0_WIDTH, VGAP0_HEIGHT);

    VGAP0_Draw_String(STRING_X, STRING_Y, time_str, FALSE);
}
```

Figure 5-4: Code to initialise the VGA peripheral – additions are in bold.

Finally, we need to output the new elapsed time to the display in the update function (“ELAPSED_TIME_UART0_Update” in the example). This function already formats the necessary string (as the “time_str” variable), so we can simply output it at the end of the function, as shown in Figure 5-5.

```
// Added to the end of the ELAPSED_TIME_UART0_Update function.
VGAP0_Draw_String(STRING_X, STRING_Y, time_str, FALSE);
```

Figure 5-5: Code to update the display.

5.3 Simulating the design

To simulate the design we must configure the VGA peripheral, as shown in Figure 5-6.

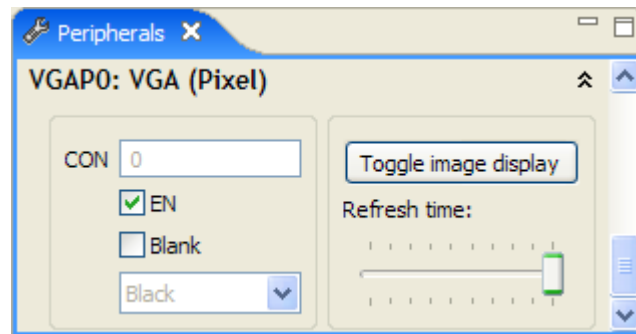


Figure 5-6: Configuring the VGA peripheral.

Selecting “Toggle image display” in the peripherals view of the simulator will bring up a window with the output of the VGA (effectively simulating a monitor). Start the application running with the “continue” button in the “Debug” view; this should result in the output shown in Figure 5-7.



Figure 5-7: Output of the simulated VGA peripheral.

6 Gathering timing statistics

At this point, we have a complete system that we have tested and found to be functioning correctly. However, we might be curious about the timing of the tasks in the system – especially as we are outputting a time-based value and may need to demonstrate its accuracy.

This can be achieved through the statistics gathering functionality of the RapidITy FPGA debugger, either with the application running on the simulator or directly in hardware. Simply open the debug launch configuration dialog, but selecting the down-arrow next to the debug button on the toolbar and clicking on “Debug...”. The resulting dialog is shown in Figure 6-1, with the statistics tab selected.

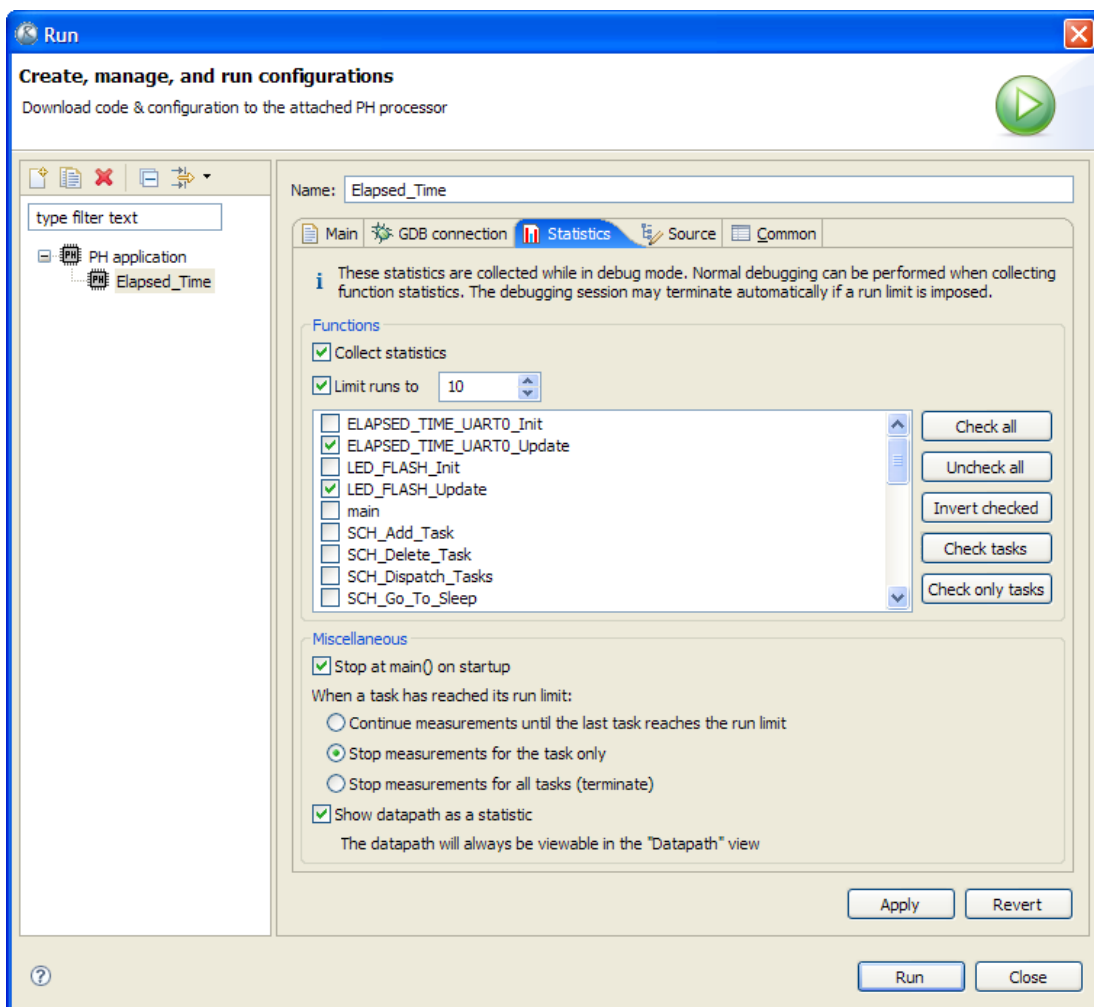
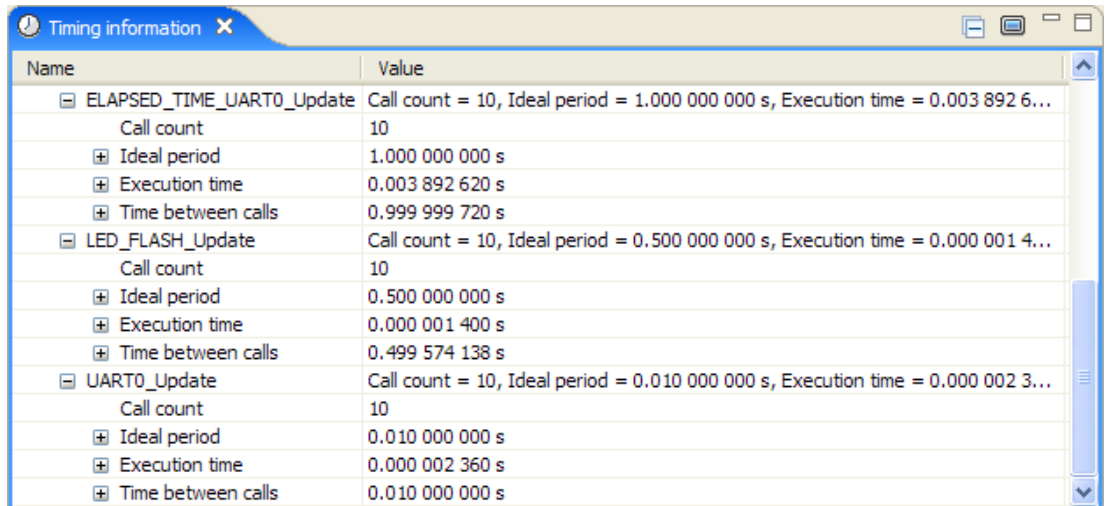


Figure 6-1: The statistics settings, with some tasks selected.

In the statistics settings, shown in Figure 6-1, we have selected to monitor three tasks: `ELAPSED_TIME_UART0_Update`, `LED_FLASH_Update` and `UART0_Update`. Limiting the number of runs to monitor is a useful way of reducing the amount of

time required to complete the statistics gathering; in this case we have chosen to restrict RapidITy FPGA to monitoring just ten runs of each task. Simulating with these settings in place will actually gather the statistics, as shown in Figure 6-2.



Name	Value
[-] ELAPSED_TIME_UART0_Update	Call count = 10, Ideal period = 1.000 000 000 s, Execution time = 0.003 892 6...
Call count	10
[+] Ideal period	1.000 000 000 s
[+] Execution time	0.003 892 620 s
[+] Time between calls	0.999 999 720 s
[-] LED_FLASH_Update	Call count = 10, Ideal period = 0.500 000 000 s, Execution time = 0.000 001 4...
Call count	10
[+] Ideal period	0.500 000 000 s
[+] Execution time	0.000 001 400 s
[+] Time between calls	0.499 574 138 s
[-] UART0_Update	Call count = 10, Ideal period = 0.010 000 000 s, Execution time = 0.000 002 3...
Call count	10
[+] Ideal period	0.010 000 000 s
[+] Execution time	0.000 002 360 s
[+] Time between calls	0.010 000 000 s

Figure 6-2: The timing view after gathering statistics for three tasks.

When complete, RapidITy FPGA will also generate a series of detailed graphs with all of the data presented in a more easily readable form. We can export the statistics to a variety of formats, including data in Excel or CSV formats and reports in PDF, HTML or RTF formats.

From the data shown in Figure 6-2, we can see that all three tasks have execution times well below their periods (so there appears to be no real risk of overrunning) and the display update task has an average period around 280ns short of one second (so the elapsed time will be reasonably accurate).

7 Porting a project to work on hardware

Now that we have a complete system that has been tested and to some extent verified on the simulator, it would be nice to have it uploaded and running on actual FPGA hardware.

In order to do this we must create a new project, this time with hardware files included. Figure 7-1 shows the new project wizard's configuration page, with all the necessary peripherals selected and an increase in the available code memory (to allow for some expansion later on).

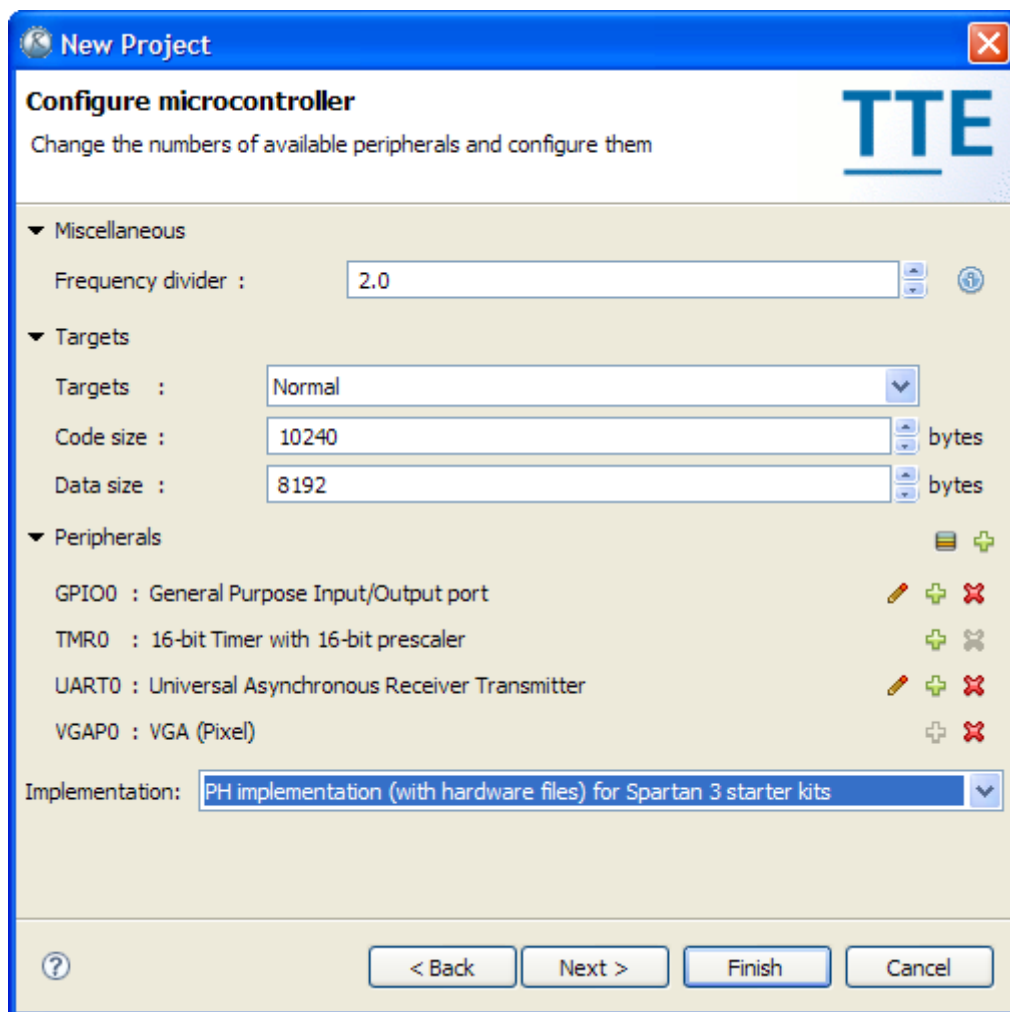


Figure 7-1: New project configuration, this time with hardware files included.

This time we have our own source-code from the previous project, so we only want to generate code for the peripherals and the startup script. To achieve this, we can deselect the “Generate example” check box on the following page of the wizard.

When the project is created RapidiTTy FPGA will probably attempt to build it for you (depending on your settings). We can expect this operation to fail, because we have no “main” function and only source-code for the drivers. This is acceptable at this phase, as we will be adding the additional source-code from the previous project.

Hardware projects take a long time to build, as the hardware configuration must be generated from the VHDL source files – for this reason, we recommend turning off “build automatically” from the “Project” menu. Once this is done, we can copy the source-files from the old project. Only the files containing the application entry-point, the scheduler and the tasks are required – for this example, only the files shown in Figure 7-2.

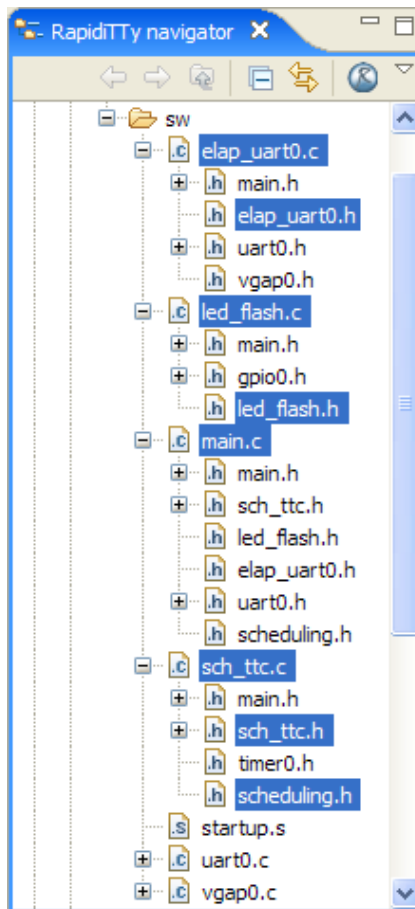


Figure 7-2: Source files selected and ready to drag-and-drop into the new project.

Simply drag-and-drop the source-files to the new project (hold control to copy rather than move them). Once built, we can see the output with the communications view, as shown in Figure 7-3.

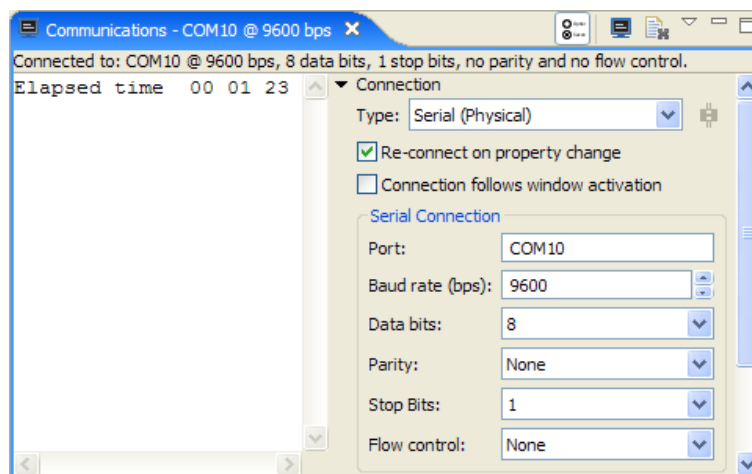


Figure 7-3: Actual output from the FPGA hardware.