Tutorial (LPC2378) for

RapidiTTy MCU 2.3

TTE Systems

Rapid development of reliable embedded systems

http://www.tte-systems.com

Version

Tutorial (LPC2378) for RapidiTTy MCU v2.3 (September 2009)

Copyright

This document is copyright © TTE Systems Limited 2007-2009. 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.

MIPS® is a registered trademark of MIPS Technologies Inc.

NXP™ is a trademark of NXP Semiconductors

RapidiTTy™, TTE System™, TTE Builder™, TTE Debug™ and TTE Statistics™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 to RapidiTTyTM MCU.....5
1.1 Overview of this tutorial.....5
2 Hardware requirements.....6
3 Example A.....7
3.1 Overview.....7
3.2 Creating a new project.....7
3.3 Adding a “Flashing LED” component.....9
3.4 Adding an ADC component.....12
3.5 Adding a “user component”.....13
3.6 Creating port connections.....16
3.7 Adding a “LCD Display” component.....19
3.8 Obtaining timing analysis.....20
4 Example B.....24
4.1 Overview.....24
4.2 Adding a Switch component.....25
4.3 Modifying an existing user component.....26
4.4 Adding the “Elaps Time” component.....29
5 Where do we go from here?.....32
5.1 RapidiTTy MCU.....32
5.2 Other products in the RapidiTTy family.....32
5.2.1 RapidiTTy FPGA.....33
5.2.2 RapidiTTy x86.....33
5.2.3 RapidiTTy Professional.....34
5.3 Time-triggered systems.....34

1 Introduction to RapidiTTy™ MCU

RapidiTTy™ MCU is a tightly-integrated tool suite which supports the rapid development of code for microcontrollers with ARM7®, Cortex-M3™ and PH03™ processors. In the current version of the tool (v2.3), full support is provided for the LPC-2xxx family of ARM7-based microcontrollers (from NXP), the STM32F10xxx family of Cortex-M3-based microcontrollers (from ST Microelectronics) and the new family of “soft” TTE32-SM3-based microcontrollers (from TTE Systems). Preliminary support is also provided for the LPC-17xx family of Cortex-M3 based microcontrollers (also from NXP).

RapidiTTy™ MCU provides the following key benefits for developers who wish to create, test and maintain reliable and resource-efficient embedded systems:

- Component-based design with the TTE Builder™ engine and an extensive library of source-code: system prototypes can be created in minutes with just a few clicks of a mouse.

- The TTE Statistics™ toolbox, which allows you to obtain important timing information (for example, worst-case execution time and task jitter), based on real measurements from your own code running on your target hardware .

- Static memory analyser that provides useful stack checks and code and data meters for individual functions.

- Full support for an integrated and coherent range of thin, resource-efficient operating systems (all royalty free, all fully integrated with the toolset) .

Like all RapidiTTy™ toolsets, RapidiTTy™ MCU is based on time-triggered (TT) technology[1]. Use of TT technology helps to ensure that even new developers can produce reliable embedded systems, and helps to maximise the efficiency of an experienced development team.

The purpose of this tutorial is to guide you on how a software system can be easily implemented using RapidiTTy MCU. Specifically, we will walk you through two non-trivial case studies to see how this can be achieved whilst demonstrating the features of RapidiTTy MCU.

1 Information about time-triggered systems can be found in the book “Patterns for Time-Triggered Embedded Systems”. This book can be downloaded (without charge) from the TTE Systems website: http://www.tte-systems.com/books/pttes

1.1 Overview of this tutorial

Before attempting to complete the examples described in this document, please refer to the RapidiTTy MCU "Getting Started Guide"[2] for information about installing the tool and a basic guide to the use of the tool.

This tutorial consists of two examples which are intended to help you to understand how to make best use of the RapidiTTy MCU toolset. It is recommended that you try to complete the examples yourself with the tool as you read through this guide.

2 The RapidiTTy MCU "Getting Started Guide" can be obtained from http://www.tte-systems.com/downloads/

2 Hardware requirements

In this "hands-on" tutorial, we will be using some cost-effective evaluation hardware produced by Olimex and Amontec.

The evaluation board we will use is an Olimex LPC2378 STK (see Figure 2-1). This incorporates an NXP LPC2378 32 bit ARM7TDMI-S™t core with 512 KByte Flash, 32 KByte RAM, External memory bus, RTC, 1x 10 bit ADC with 8 channels, 4x UART, 4x CAN, I2C, SPI, 4x 32-bit TIMERS, 7x CCR, 6x PWM, WDT, 5V tolerant I/O, up to 72 MHz operation.

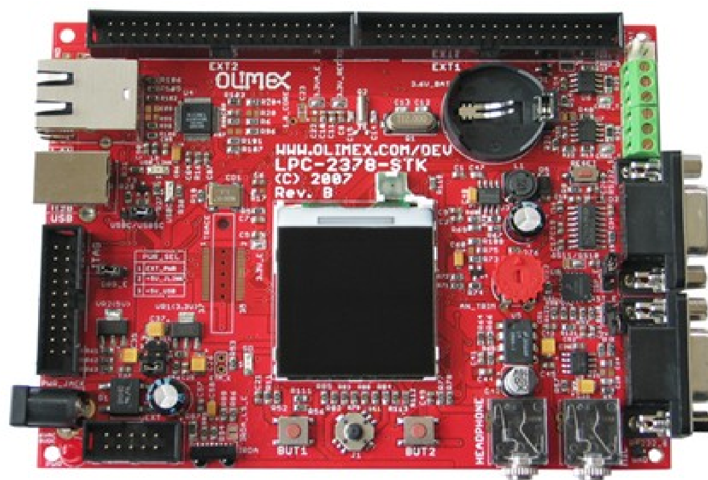

Figure 2-1: An LPC-2378 STK development board from Olimex

The JTAG interface we will use is an Amontec "Tiny" JTAGkey (Figure 2-2). This low-cost device can be self-powered from a USB-HUB (draws < 85mA). It supports a wide range of voltages (2.8V to 5.0 V).

Figure 2-2: JTAGkey "Tiny" device from Amontec

3 Example A

3.1 Overview

The goal in Example A is to use RapidiTTy MCU to develop, test, implement and analyse a simple data acquisition system. Specifically, we will design the embedded system such that it takes readings from an analogue-to-digital converter (ADC) on the LPC2378 processor, translate that value to an appropriate string of characters, and transmits the resulting information to the on-board LCD controller. A high-level representation of this is illustrated in Figure 3-1.

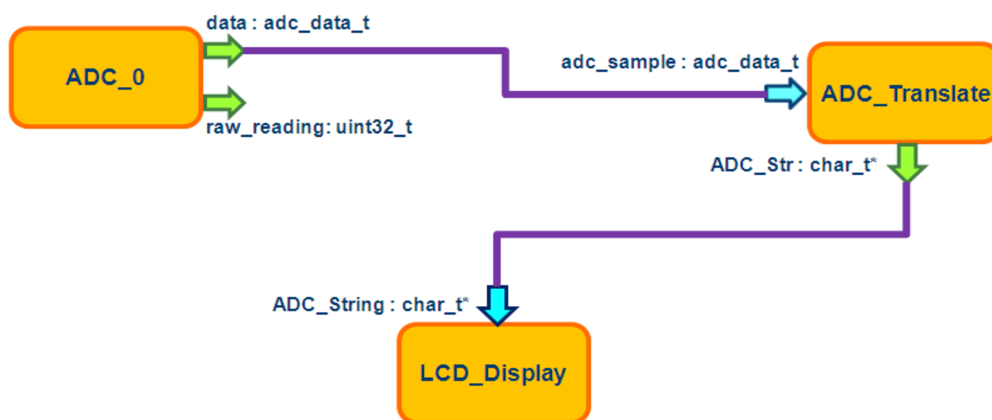

Figure 3-1: High-level component representation of Example A

3.2 Creating a new project

To begin implementing the system described in Section 3.1 , you will first have to create a new project. To do this, please select `File->New-> RapidiTTy Project`. You will be presented with a RapidiTTy "new project" window (see Figure 3-2). In this window, you will be required to enter the project name, for instance "MCU_Tutorial".

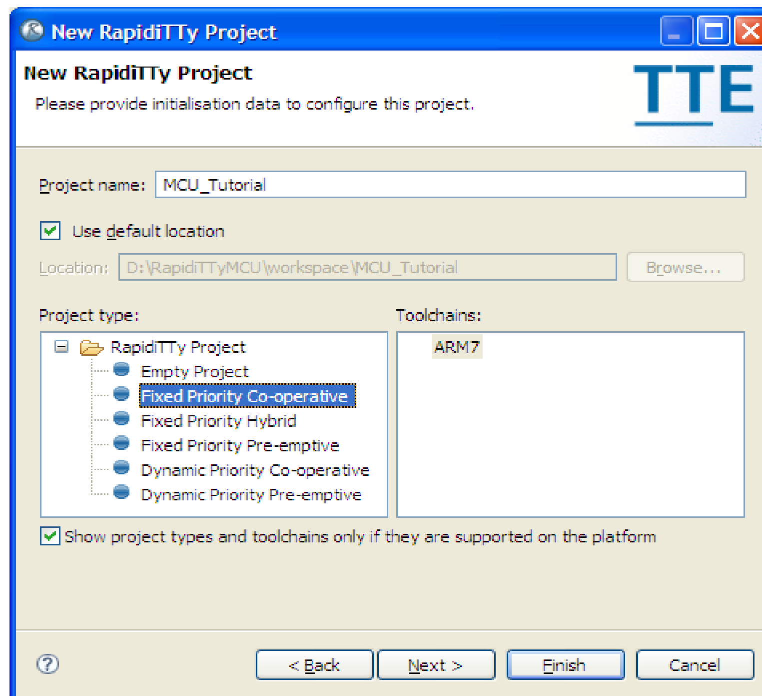

Figure 3-2: The new project dialog

Here, you are given a choice of selecting a range of schedulers[3] supported by RapidiTTy MCU. These schedulers are listed below:

- Fixed priority co-operative
- Fixed priority hybrid
- Fixed priority pre-emptive
- Dynamic priority co-operative
- Dynamic priority pre-emptive

Alternatively, you may choose to start with an "empty project" which only provides you with the initialisation code for the selected device.

> Please note, that in this version, you will not be able to change the scheduler after the project has been created. However, you can always create a new project with a different scheduler, if required.

3 For details of these individual schedulers, please refer to the RapidiTTy MCU "Getting Started Guide" from http://www.tte-systems.com/downloads/

For this tutorial, we will select the fixed-priority co-operative scheduler as our starting point. Clicking "Finish" will generate the project code template and bring you to the project "Overview" page (Figure 3-3).

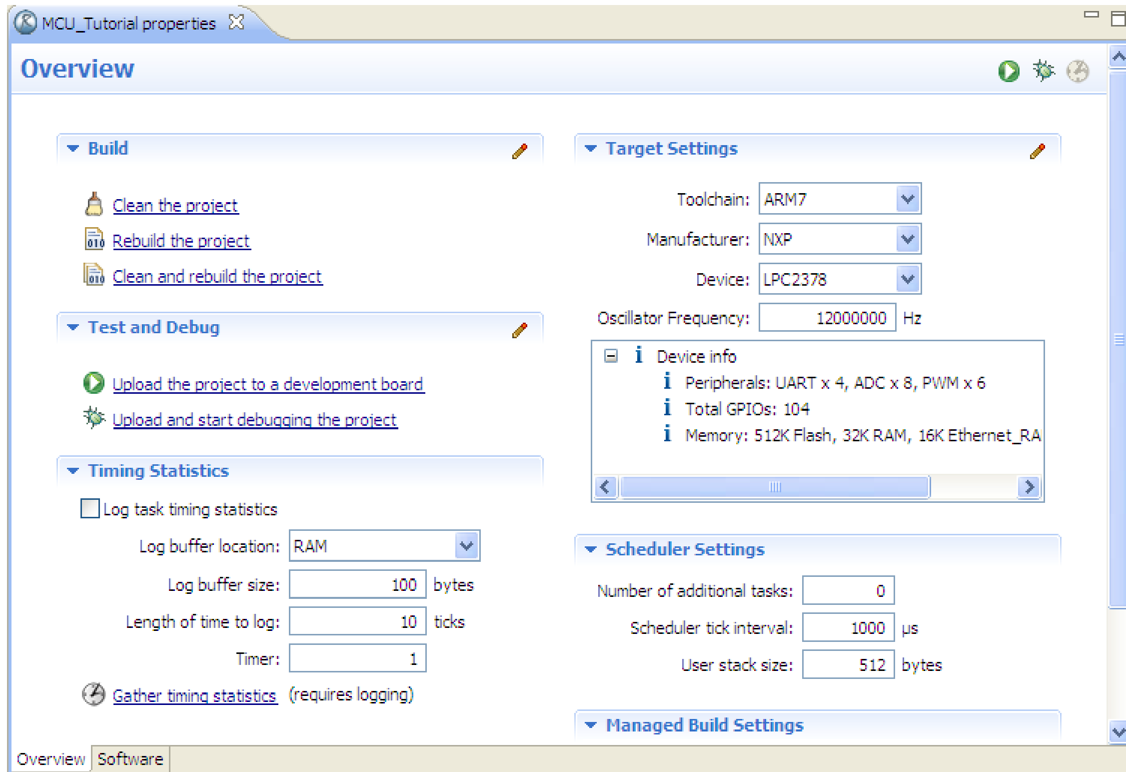

Figure 3-3: The Project Overview page.

3.3 Adding a "Flashing LED" component

Having created a RapidiTTy Project, we are now ready to add components to the system.

As briefly described in Section 1 , the TTE Builder™ engine can be used to implement software components for your embedded application in an efficient and timely fashion.

To add components and tasks using TTE Builder™, please navigate to the "Software Components view (see Figure 3-4). This can be done by clicking the "Software" tab at the bottom left of the "Overview" page.

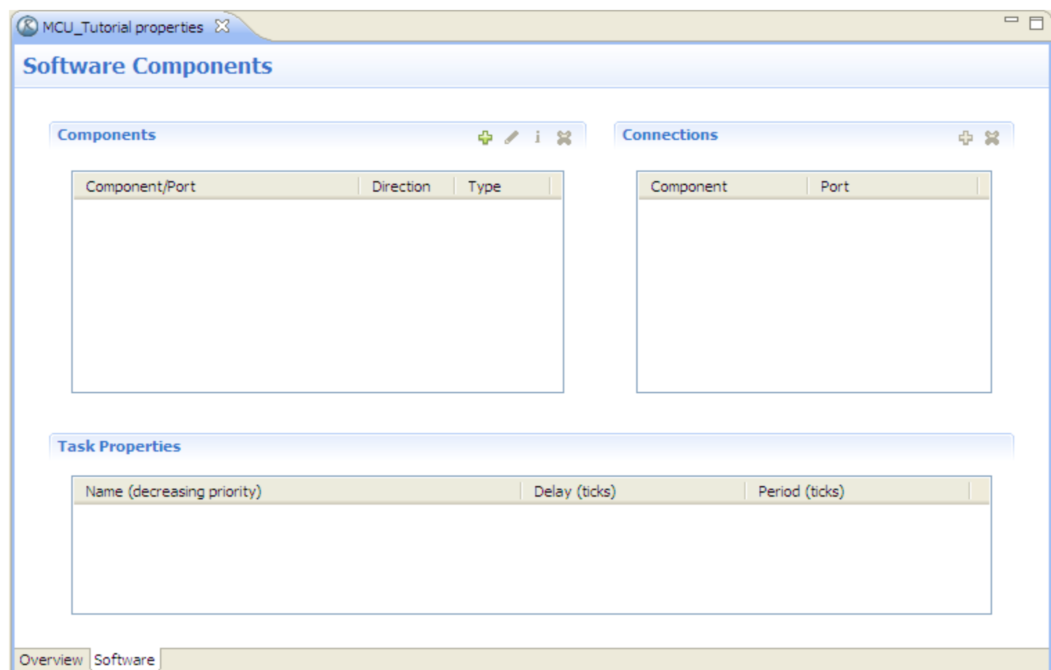

Figure 3-4: The Software Component Page

The first component that we shall add is a "Flashing LED" to indicate that the system is alive. This can be done by clicking on the green "plus" sign in the "Components" section. This will bring up a list of components that can be selected (see Figure 3-5).

Select the "Flashing LED" component and set the LED pin to Port 0.21 (Figure 3-6). This pin is connected to the on-board red LED. The LED is set to toggle periodically; e.g. every 500 ticks (see Figure 3-7).

> Please note, that the tick interval for this current project has been set to 1000µs. The tick interval for the scheduler can be changed in the "Scheduler Settings" in the "Overview" page (see Figure 3-3).

Save, compile, and upload the binary to the target processor (please refer to the "Getting Started Guide" on how to do this). Check to see that the LED is flashing. Then, try modifying the period of the task to 1000 ticks. Save, compile, and upload[4] the updated binary to observe the resultant behaviour on the development board.

4 Occasionally, you may come across problems while uploading your binary using JTAG. Please refer to Section 10 of the "Getting Started Guide" for hints on troubleshooting.

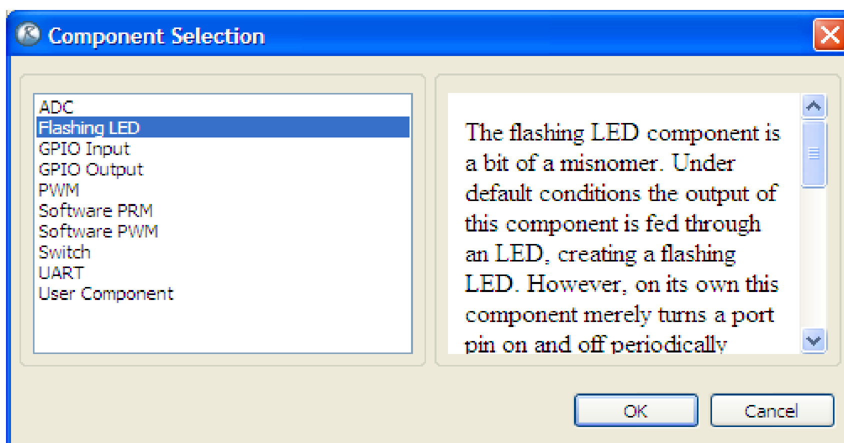

Figure 3-5: Selecting a "Flashing LED" from the list of components

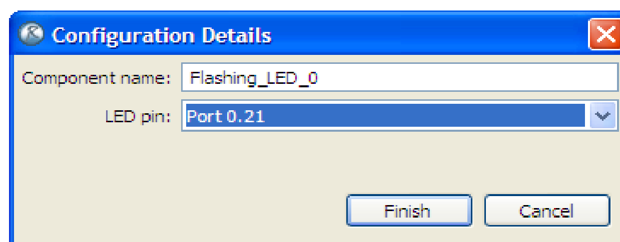

Figure 3-6: Configuring the "Flashing LED" component

Task Properties

Name (decreasing priority)	Delay (ticks)	Period (ticks)
Flashing_LED_0_Toggle	0	500

Figure 3-7: Timing information for the Flashing_LED_0_Toggle task

Having successfully added a "Flashing LED" component, we can now use this system as the basis to begin building the data acquisition system described in Section 3.1 .

3.4 Adding an ADC component

The next step is to add an ADC component to the project. To do this, select the ADC component from the list of available components. Configure the ADC to read from Channel 5, as illustrated in Figure 3-8. This channel is connected to the potentiometer of the development board. Please take a sample every second as shown in Figure 3-9.

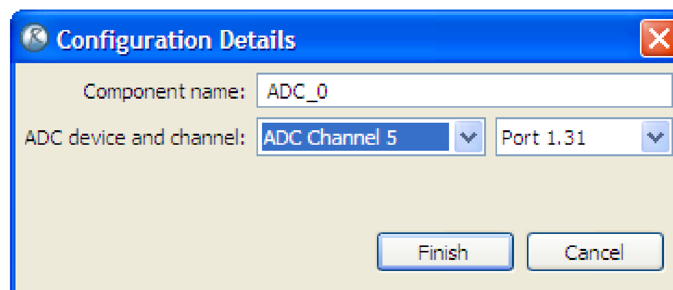

Figure 3-8: Configuring the ADC component

Task Properties

Name (decreasing priority)	Delay (ticks)	Period (ticks)
Flashing_LED_0_Toggle	0	500
ADC_0_Sample	0	1000

Figure 3-9: Updating the ADC_0_Sample task period

After saving and rebuilding your project, you can now use the debugger to check if the ADC values are being read correctly. To launch the debugger, go to the “Overview” page and click on “Upload and start debugging the project”[5]. This will take you to the debug perspective (Figure 3-10). You can add more windows to your debug perspective by selecting `Windows→Show View→Other`.

In the ADC_0.c file, please place a breakpoint on line 41, and click the “Resume” button. The processor should run and pause at line 41. Under the “Variables” panel, observe the “reading” value. This value should change when the potentiometer is changed between subsequent runs.

5 Please refer to Section 6 of the “Getting Started Guide” to obtain detailed description on how to use the debugger and the debug perspective.

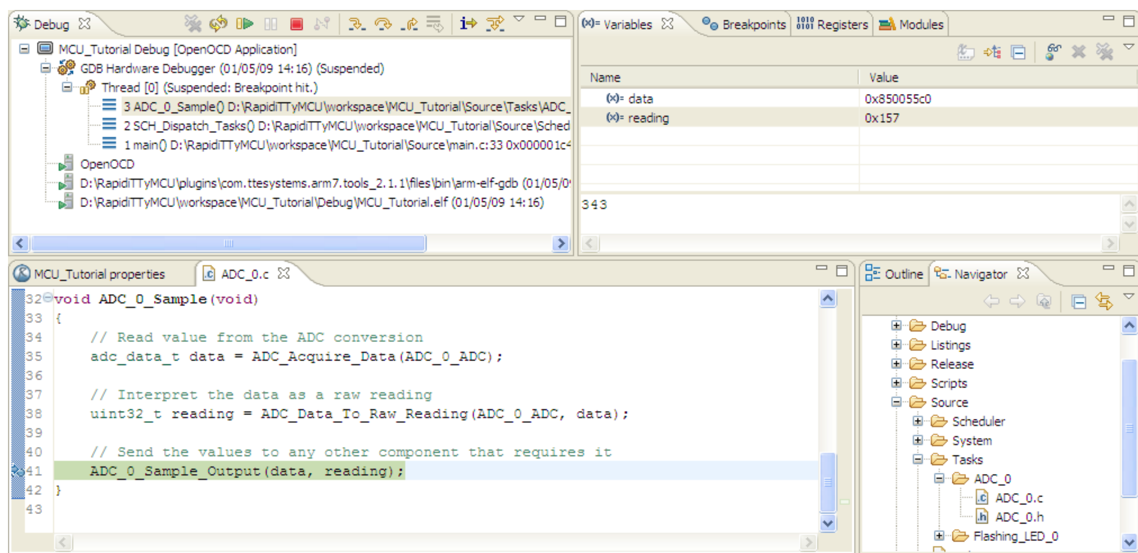

Figure 3-10: Using the debug perspective to check the ADC values

3.5 Adding a "user component"

So far, we have added two software components that are available in RapidiTTy MCU. However, there will be cases when the user may wish to implement a software component specific to a particular application that may not be available in the provided list of components.

RapidiTTy MCU can assist in this process through the "user component". A user-defined component is a software component that can be created and customised by the user. The "user component" will be seamlessly integrated into the RapidiTTy framework and the selected scheduler. This has the effect of saving time and avoiding potentially costly mistakes.

In this example, we will create a "user component" to translate the ADC value to a percentage. To do this, select the "user component" as shown in Figure 3-11.

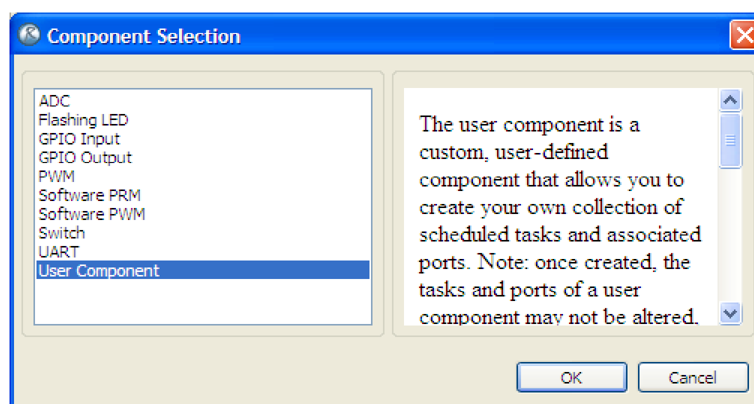

Figure 3-11: Selecting a "user component"

Change the name of the component to "ADC_Translate". Next, add a task called "Update" and set the period of the task to be 1000 ticks (see Figure 3-12).

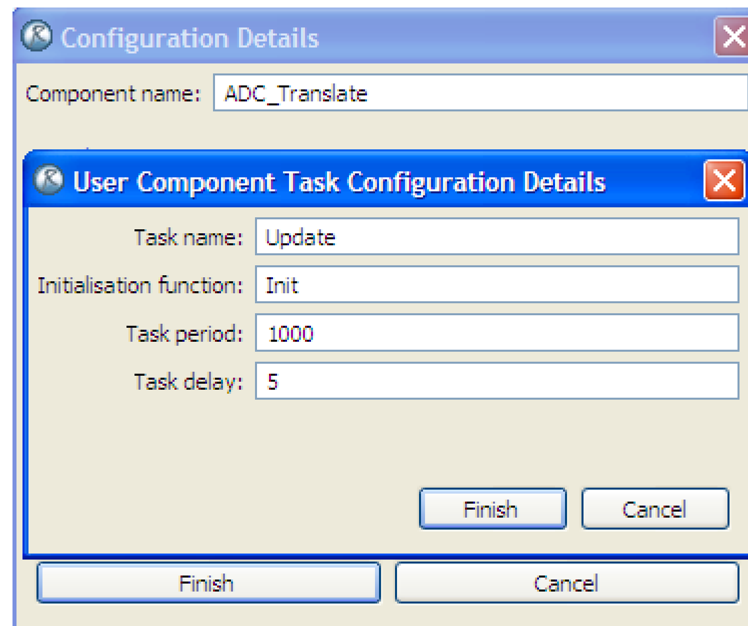

Figure 3-12: Adding a task to the user component

Based on the Figure 3-1, the "ADC_Translate" component has 2 ports:

- An input port called "*adc_sample*" which is of type adc_data_t
- An output port called "*ADC_Str*" which is of type char_t*.

Click on the "Add port" button to add the "*adc_sample*" port, as shown in Figure 3-13. Figure 3-14 shows the process of adding the "*ADC_Str*" output port.

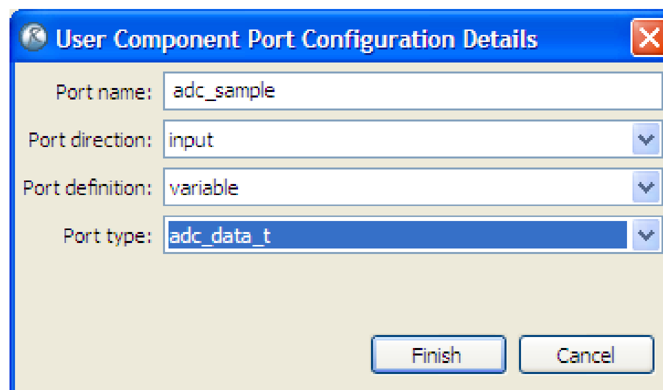

Figure 3-13: Adding an input port called "adc_sample" of type adc_data_t

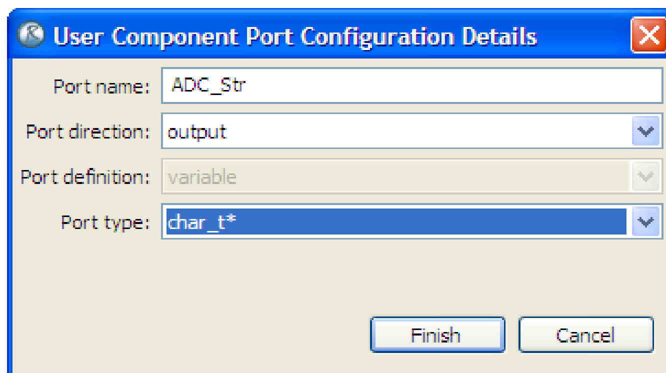

*Figure 3-14: Adding an output port called "ADC_Str" of type char_t**

Please note that by clicking “Finish” in Figure 3-15, you will no longer be able to add or remove tasks and ports. However, you will still be able to import tasks and ports (that have subsequently been written in the source file) into the tool through the “Import” option. This will be illustrated in more detail in Example B (Section 4.3).

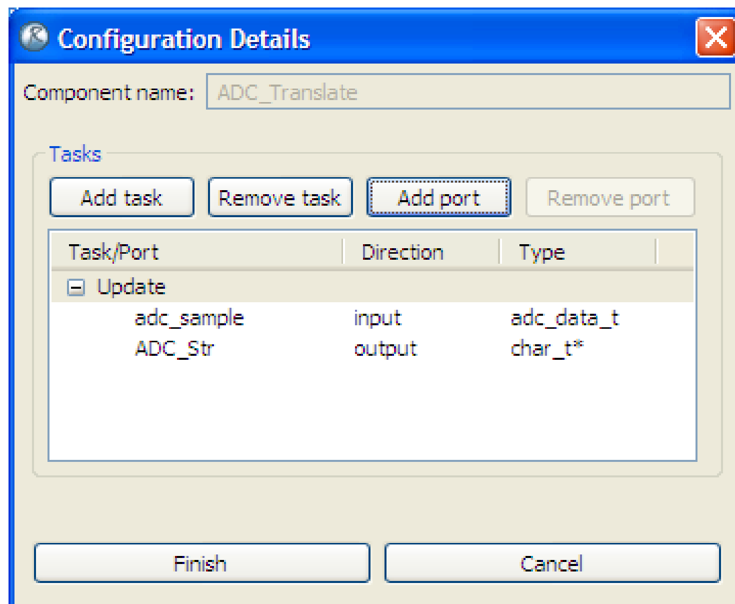

Figure 3-15: The result after having added two ports to the Update task

The result of adding the new “ADC_Translate” component can now be viewed in the “Software Component” page, as seen in Figure 3-16.

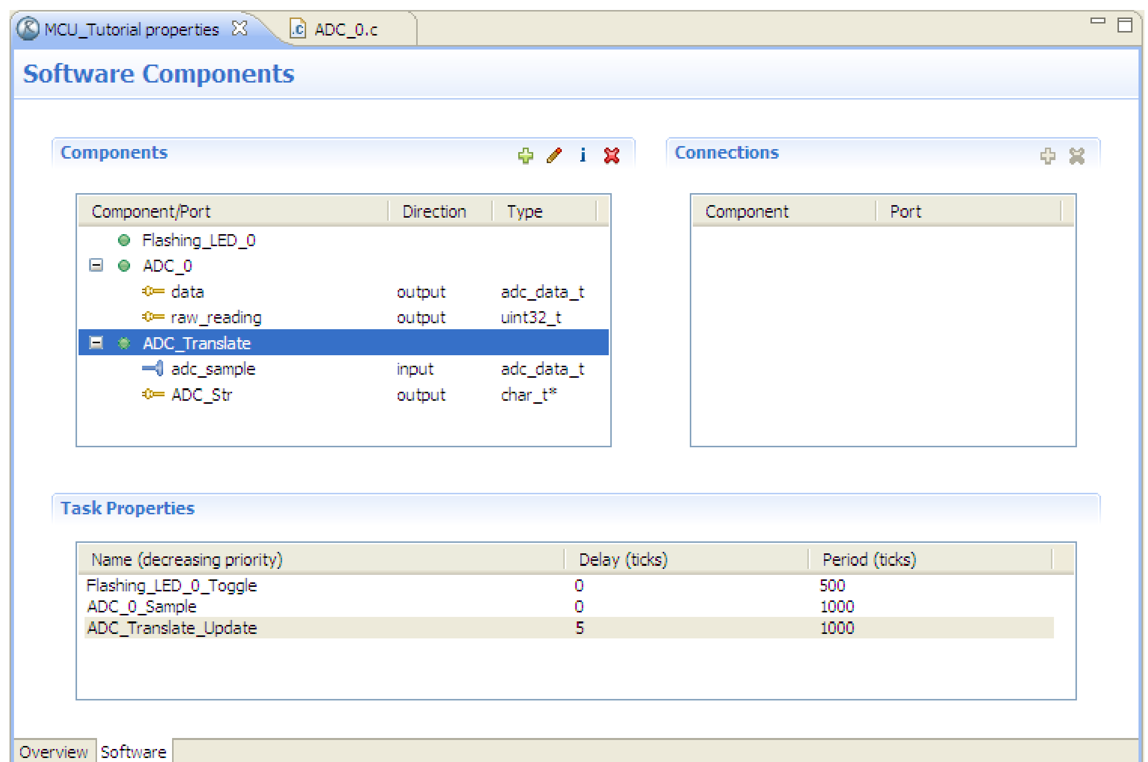

Figure 3-16: Software Components page after adding the ADC_Translate component

3.6 Creating port connections

In most desktop applications, the use of global variables is not encouraged. However, for embedded applications, the use of global variables is a common and acceptable means of passing data between various functions, if used appropriately. In RapidiTTy MCU, the passing of such data is represented through ports and port connections.

Port connections are a way for component to interact with each other. In this particular example, we can now make a port connection between the ADC_0 component and the ADC_Translate component.

To do this, right click on the "*data*" port in the ADC_0 component and select "Create connection". This will bring up a window that displays all the available ports that "*data*" can be connected to. In this case, the only suitable port for connection is the "*adc_sample*" port in the ADC_Translate component. Click on the "*adc_sample*" port and select "OK" to make the connection.

The connections made are displayed in the "Connections" view, as illustrated in Figure 3-17.

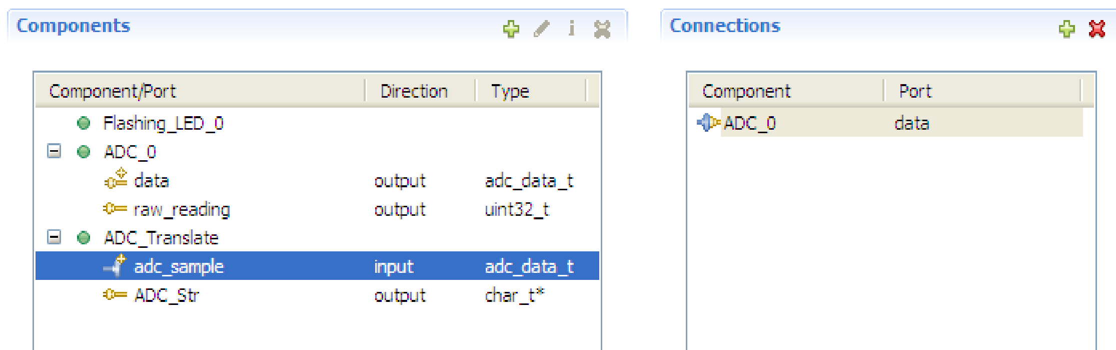

Figure 3-17: "Connections" view shows all the valid port connections made to the selected port or component

To understand how this connection is translated to the source code level, please view the generated ADC_0.h file, as shown in Listing 3.1.

```
TTE_ALWAYS_INLINE
static inline void ADC_0_Sample_Output(TTE_UNUSED adc_data_t data,
                      TTE_UNUSED uint32_t raw_reading)
{
        ADC_Translate_adc_sample = data;
}
```

Listing 3.1: The source code that is generated when a port connection is made

As shown in Listing 3.1, it can be seen that the ADC data from ADC_0 component has now been assigned to the "*adc_sample*" variable of the ADC_Translate component (the "*adc_sample*" variable is defined as an "extern" in the ADC_0.h file). This value can now be used in the ADC Translate component to perform further computations.

You will now need to write come C code in the ADC Translate task to convert the ADC reading into a percentage to be displayed as a string, e.g. "ADC: 60%". To do this, you can use the code sample in Listing 3.2.

```
#include <stdio.h>

/**
 * The ADC_Str output port
 */
char ADC_Str[15];

/**
 * The Update task, executed by the scheduler.
 */
void ADC_Translate_Update(void)
{
        uint32_t adc_data_scaled;

        adc_data_scaled = ADC_Data_To_Scaled_Reading(ADC_0_ADC,
                ADC_Translate_adc_sample, 0, 100);

        const char* adc_data_string = "ADC: %d%%                              ";

        snprintf(ADC_Str, 15, adc_data_string, adc_data_scaled);
        ADC_Str[14] = '\0';

        // Pass the values of the output ports to this function:
        ADC_Translate_Update_Output(ADC_Str);
}
```

Listing 3.2: Sample code required in the ADC_Translate.c file to translate the ADC reading into a string, which is then displayed as a percentage value

Save and compile your project. Once again, you can use the debugger to check if the value "*ADC_Str*" is displayed correctly.

3.7 Adding a "LCD Display" component

Having confirmed in the previous section that the correct string is being produced, an LCD component can now be added to display the string on an 128x128 pixel 12-bit TFT LCD screen.

To do this, another user component called "LCD_Display" will need to be added. An "Update" task is added which is called every 1000 ticks. An input port (called "*ADC_String*") of type char_t* is also required to obtain the string value from the "ADC_Translate" component. Having done this, a connection is made between "*ADC_Str*" port of the "ADC_Translate" component and the "*ADC_String*" port of the "LCD_Display" component. After doing all this, your "Software Component" page should look similar to Figure 3-18.

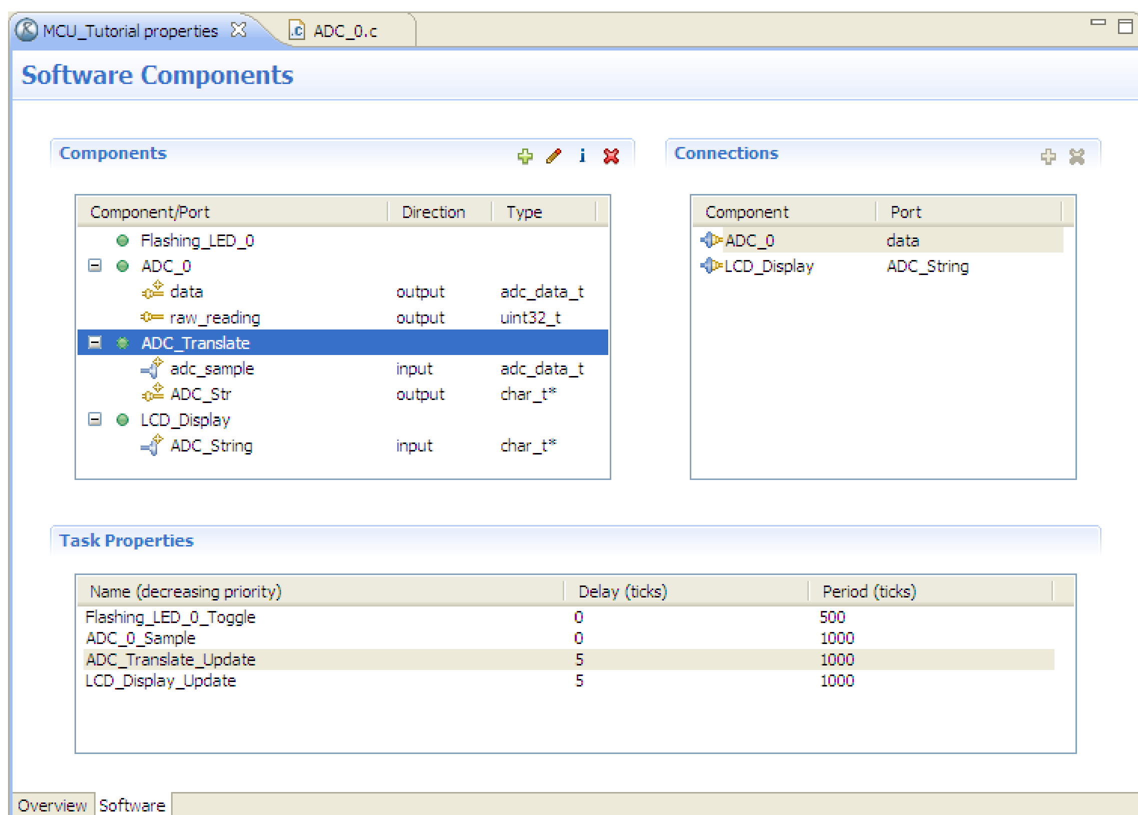

Figure 3-18: Component view after adding an "LCD_Display" component and making the relevant connections

A LCD driver source code is provided to you on the CD (Source_Code\LCD)[6]. Copy these files (lcd_driver.c, lcd_driver.h and font_library.h) and paste them into RapidiTTy under the folder LCD_Display.

6 As an alternative to an LCD screen, the values can also be displayed on hyperterminal (or similar) using the UART for RS-232 communications.

You will now need to write some C code in the “LCD_Display_Update” task to correctly display the ADC reading on the LCD screen. This sample code is provided in Listing 3.3.

Save, compile and upload your code to the processor. You should now be able to see the ADC value displayed on the LCD screen. Try turning the potentiometer to see the ADC value changing on the LCD screen.

Try changing the period of the tasks “ADC_0_Sample”, “ADC_Translate_Update” and “LCD_Display_Update” to 500 ticks. This will improve the response time of the ADC display update.

```
#include <string.h>
#include "lcd_driver.h"
/**
 * Initialises LCD_Display
 */
void LCD_Display_Init(void)
{
        // Initialisation code goes here
        LCD_Init();
}

/**
 * The Update task, executed by the scheduler.
 */
void LCD_Display_Update(void)
{
        char display_string[15] = {"              "};

        strcpy(display_string, LCD_Display_ADC_String);

        LCD_Send_String(display_string, 10, 10, MEDIUM_FONT, BLACK, WHITE);

        // Pass the values of the output ports to this function:
        LCD_Display_Update_Output();
}
```

Listing 3.3: Sample code required to display the the string that represents the ADC value on the LCD screen

3.8 Obtaining timing analysis

When designing reliable embedded systems, it is a requirement that all task timing such as execution times, task period and jitter is known. To do this, the TTE Statistics™ Toolbox in RapidiTTy can be used to obtain timing analysis of the tasks executing on the target hardware[7].

7 Detailed description of timing analysis can be found in Section 7 of the “Getting Started Guide”.

To obtain timing analysis for the data acquisition system that was just created, you will need to click the “Log task timing statistics” box in the ”Overview” page (under the Timing Statistics section). This is illustrated in Figure 3-19.

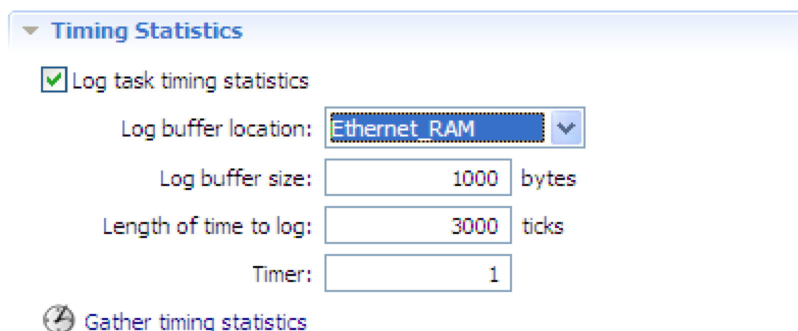

Figure 3-19: Configuration to collect timing analysis for the system

The buffer location specifies where the system will create a log buffer to store the timing values gathered from the system. For most devices, the RAM area can be used. However, for some advanced devices (like the LPC2378), there is additional RAM areas like USB RAM and Ethernet RAM that can be used by the timing analysis, provided that the application is not using it. In this case, we will choose to store the buffer in the Ethernet RAM.

The buffer size determines the size of the buffer to be created in bytes. When this buffer is full, the data is copied over to the host computer to allow the embedded system to continue gathering statistics. In this example, we shall set the buffer size to 1000 bytes.

The length of time indicates how long the timing statistics will need to be gathered for (in ticks). In this case, we want to gather timing information for 3000 ticks (3 seconds since one tick is 1ms).

An on-chip timer is used to obtain the timing analysis. It is important to not that this timer must **not** be the system timer. In this example, Timer 0 is used as the system timer. Therefore, we will use Timer 1 to gather the statistics.

Having configured the Timing Statistics section, save the project and click on “Gather timing statistics”. This action will load the code on the processor and begin gathering the timing data.

After the gathering of timing statistics has completed, a timing graph will be displayed that illustrates the task execution graph and the processor utilisation (see Figure 3-20).

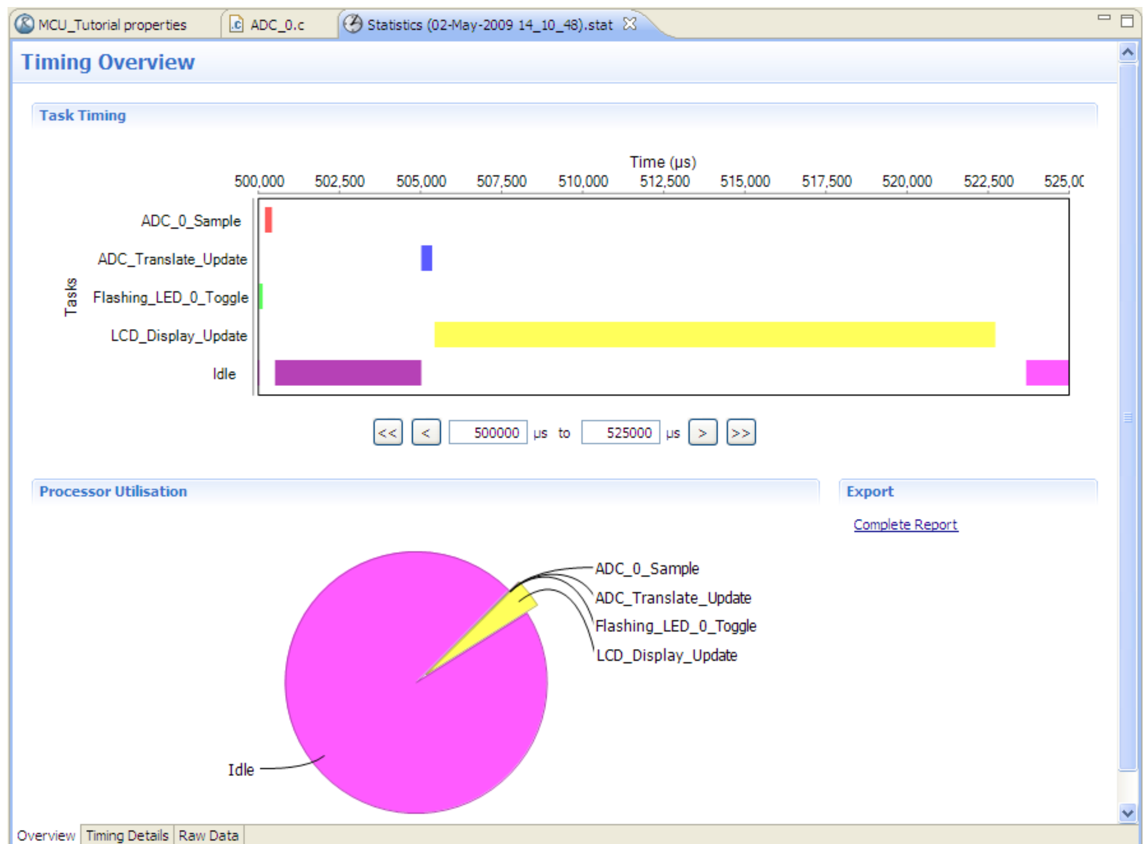

Figure 3-20: Timing Overview illustrating the task execution graph and the processor utilisation

Detailed statistical reading of the individual tasks can be obtained by clicking on the "Timing Details" and "Raw Data" tab located at the bottom left hand side of the Timing Overview page. The "Timing Details and "Raw Data" page is shown in Figure 3-21 and Figure 3-22 respectively.

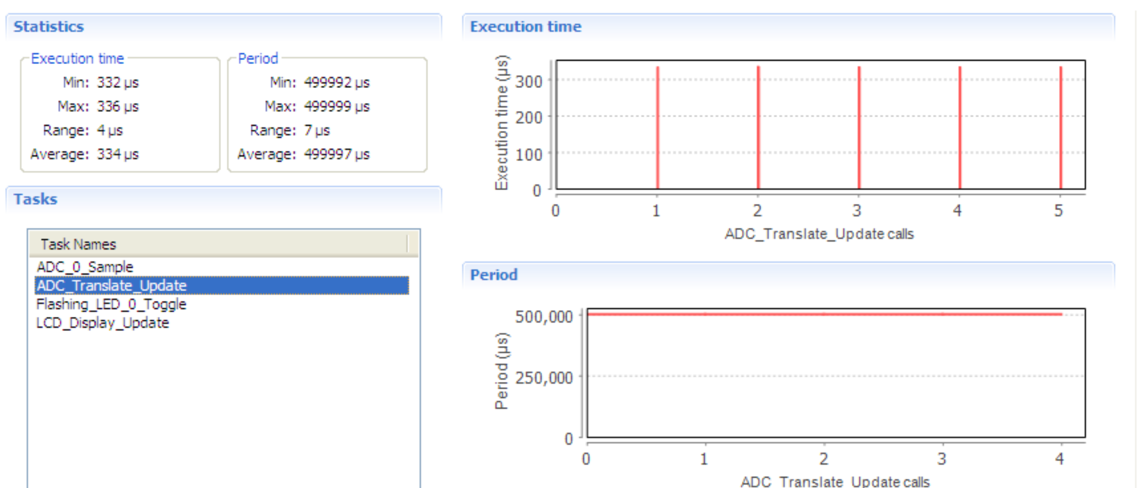

Figure 3-21: Timing details view

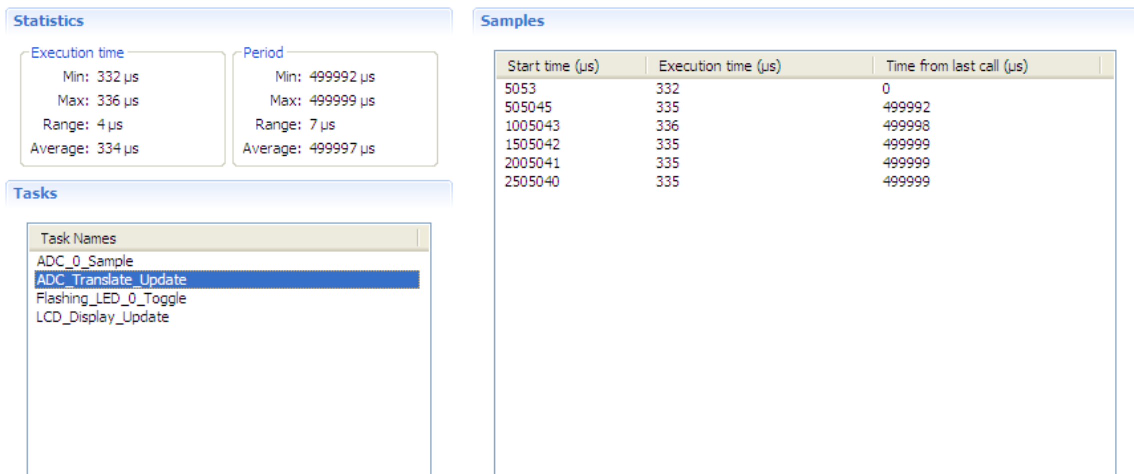

Figure 3-22: Raw timing data view

This concludes the exercise in Example A. Section 4 will discuss the requirements involved for Example B.

4 Example B

4.1 Overview

RapidiTTy MCU allows you to employ the TT Builder™ Toolbox at any time you wish to add further components to a system: this may be when a system is created or as a means of changing the functionality of an existing system during maintenance or upgrade activities. We will make use of these capabilities in Example B.

In Example A, we have created a simple data acquisition system that reads the ADC value from the potentiometer, and displays this value (in percentage) on the LCD screen.

For Example B, we will be building upon the previous system and adding additional functionality to it. For this example, we aim to achieve the following behaviour:

- When Push Button 1 (P0.29) is pressed on the LPC2378 STK board, the current ADC value should be displayed on the LCD screen, as in Example A.
- When Push Button 2 (P0.18) is pressed, the microcontroller should, instead display the information which represents the elapsed time since the microcontroller was reset, e.g. “Time: 5:36”.
- When no buttons are pressed, the LCD screen should simply flash its “heartbeat” LED while displaying a message “Press a button”.
- When both buttons are pressed, Button 1 should have priority.

A high-level component representation of this system is illustrated in Figure 4-1.

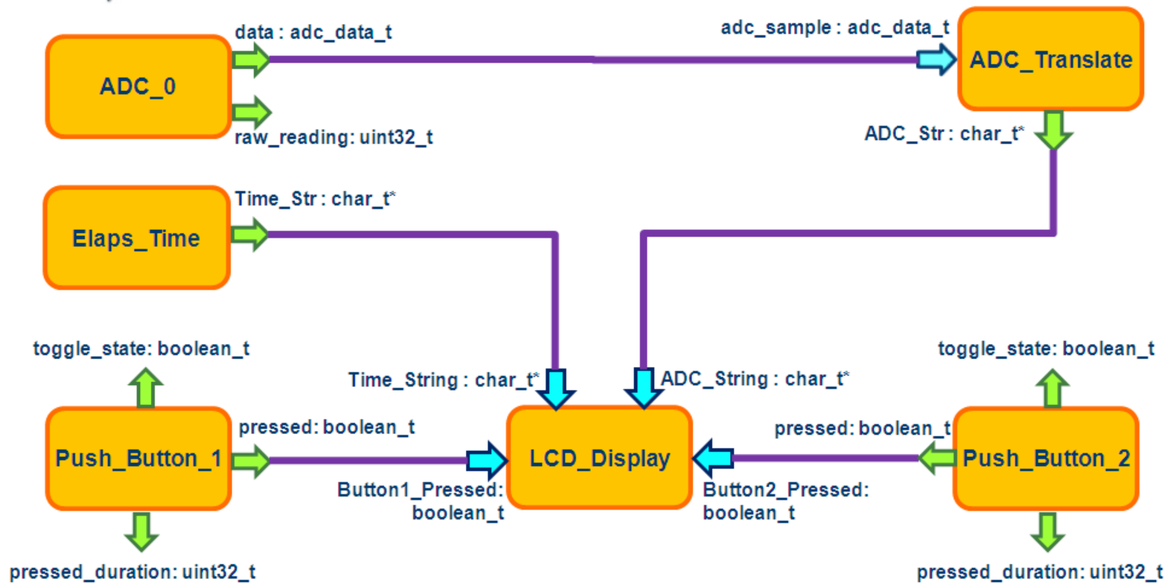

Figure 4-1: High-level component representation for Example B

4.2 Adding a Switch component

In Example B, we shall begin by trying to display the ADC reading only when Button1 is pressed. Otherwise, it will display the message "Press a button".

To do this, we will need to read the status of the push button on the development board. This can be easily done through the TTE Builder™ Toolbox by adding a Switch component (see Figure 4-2).

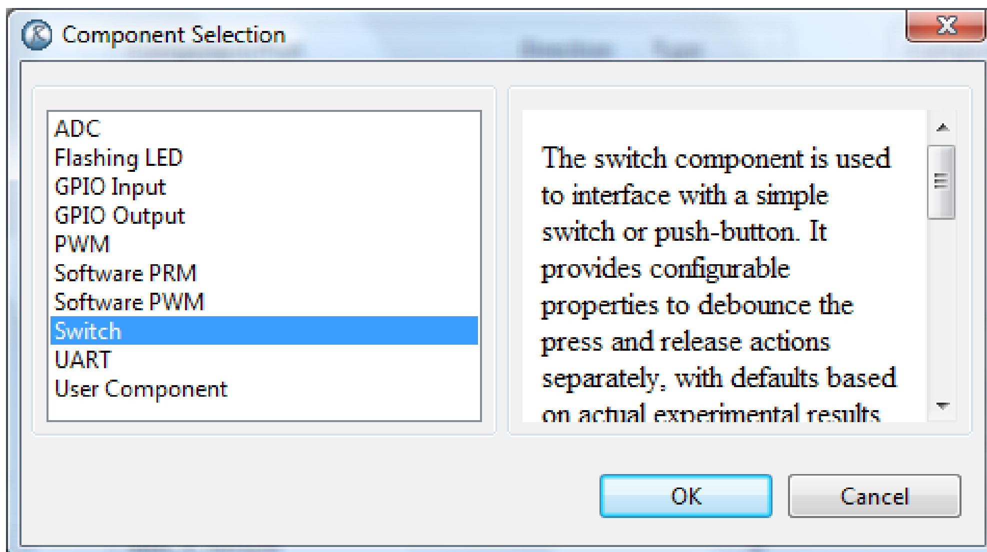

Figure 4-2: Adding a Switch component

Rename the Switch component to "Push_Button_1". Please make sure the switch pin is set to Port 0.29, which is connected to Button1 on the LPC2378 STK. The press and release debounce period can be changed to 50 ticks. Please note that the push button pins go low when the buttons are pressed. Therefore, the high when pressed option is set to FALSE. The configurations for the component is illustrated in Figure 4-3. The switch task is scheduled to run every 50 ticks (you can do this by changing the period in the "Task Properties" section in the Software Components page).

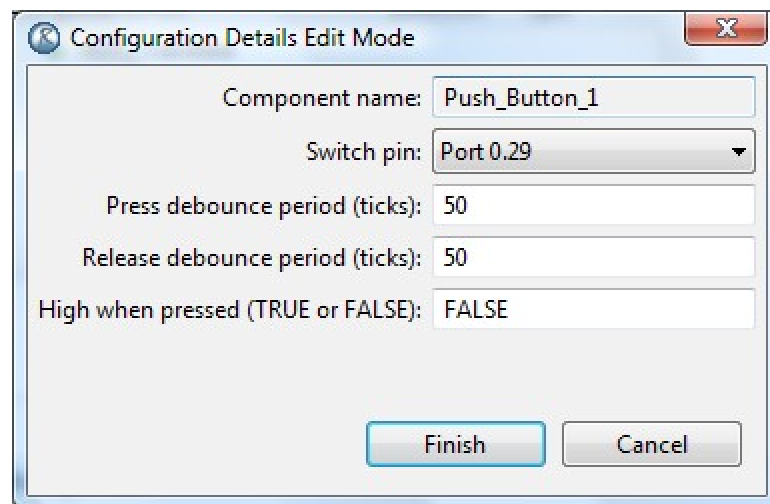

Figure 4-3: Configuration of the Switch component

Having added and configured the "Push_Button_1" component, the similar steps can be used to add "Push_Button_2". The only difference is that this switch pin is connected to Port 0.18.

4.3 Modifying an existing user component

Having added the switch components, the "LCD_Display" component will now need to be modified so that it will display the correct message when the button is pressed. The modifications to the "LCD_Display" component is shown in Listing 4.1.

```
/**
 * The Button1_Pressed input port.
 */
boolean_t LCD_Display_Button1_Pressed;

/**
 * The Update task, executed by the scheduler.
 */
void LCD_Display_Update(void)
{
	char display_string[15] = {"Press a button"};

	if (LCD_Display_Button1_Pressed)
	{
		strcpy(display_string, LCD_Display_ADC_String);
	}

	LCD_Send_String(display_string, 10, 10, MEDIUM_FONT, BLACK, WHITE);

	// Pass the values of the output ports to this function:
	LCD_Display_Update_Output();
}
```

Listing 4.1: Required code modifications for the "LCD_Display_Update" task to read a switch input from Button1

Based on the modifications in Listing 4.1, it can be seen that a new port ("Button1_Pressed") has been manually added into this file. To import this port into RapidiTTy, please double click on the "LCD_Display" component, and click on the "Update" task. Then, click on the "Import port" button to import a port from the source code into the RapidiTTy framework.

The "Import port" button will bring up a window that will allow you to add a port into RapidiTTy (see Figure 4-4). In the "Port name" dropdown box, RapidiTTy will list all the available port names that can be manually added. In this case, the "*Button1_Pressed*" port is the only available port. The port direction is configured to be input.

> Please note, that a port or a task can only be added this way if the name of the port or task is prefixed with the component name. Otherwise, RapidiTTy will be unable to detect that a port has been added.

Having imported the port, a port connection can now be made between "*Button1_Pressed*" port of the "LCD_Display" component and the "*pressed*" port of the "Push_Button_1" component. The final component configuration is illustrated in Figure 4-5.

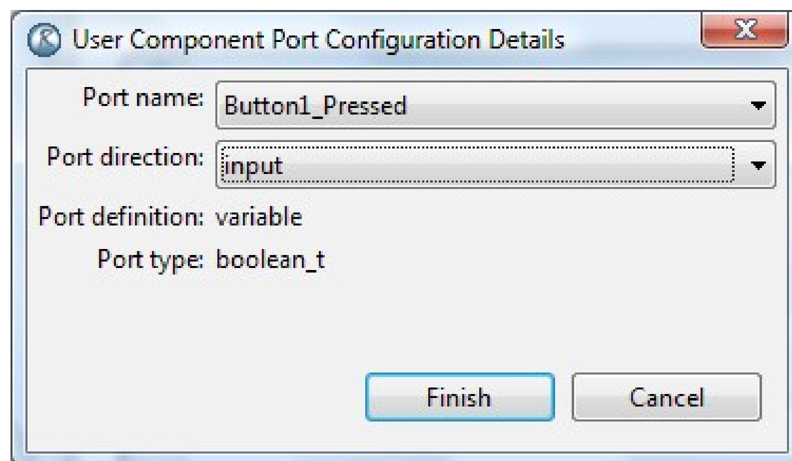

Figure 4-4: Importing the "Button1_Pressed" port into RapidiTTy

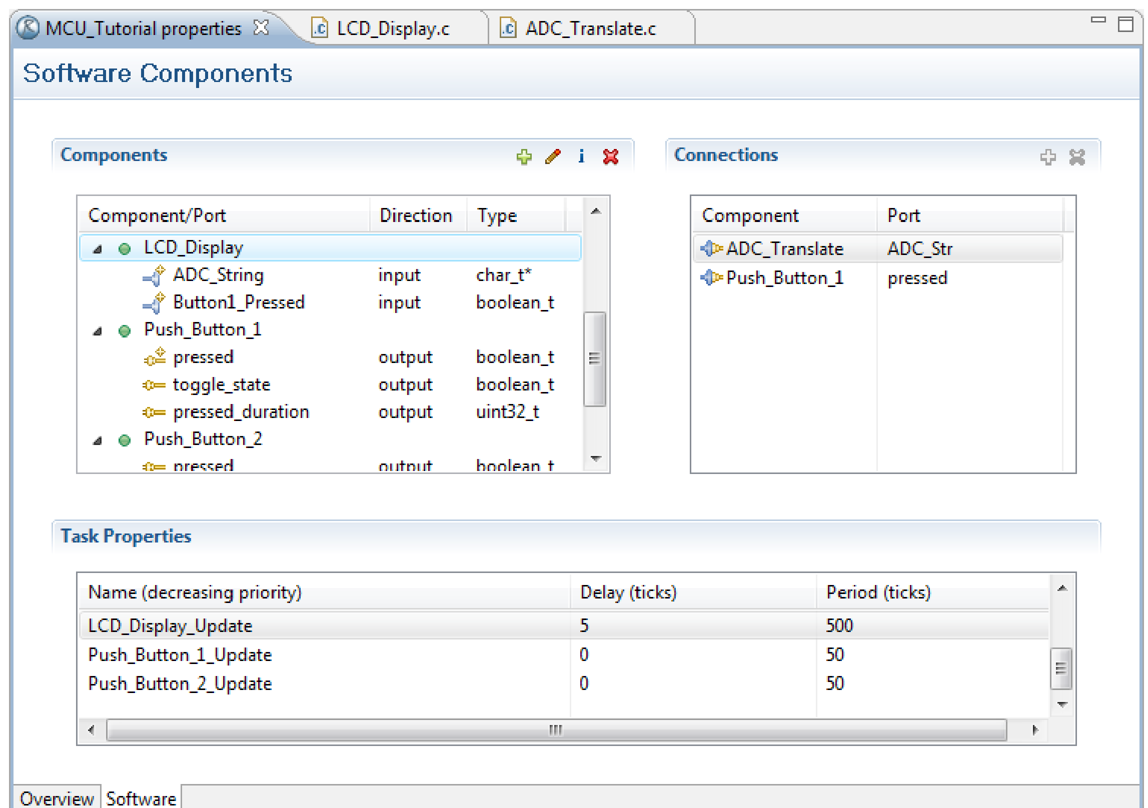

Figure 4-5: The software component view after connecting "Push_Button_1" and the "LCD_Display "component

Save, compile and upload your code onto the processor. You should now only be able to see the ADC reading when Button1 is pressed. Otherwise, a message "Press a button" is displayed on the LCD screen.

4.4 Adding the "Elaps Time" component

We shall now add a new user component called "Elaps_Time" to display the amount of time that has elapsed since the microcontroller was reset. A task called "Update" is added to this component. The Elaps_Time_Update task is configured to be scheduled every 1000 ticks. This component is configured to have an output port called "Time_Str" which is of type char_t*.

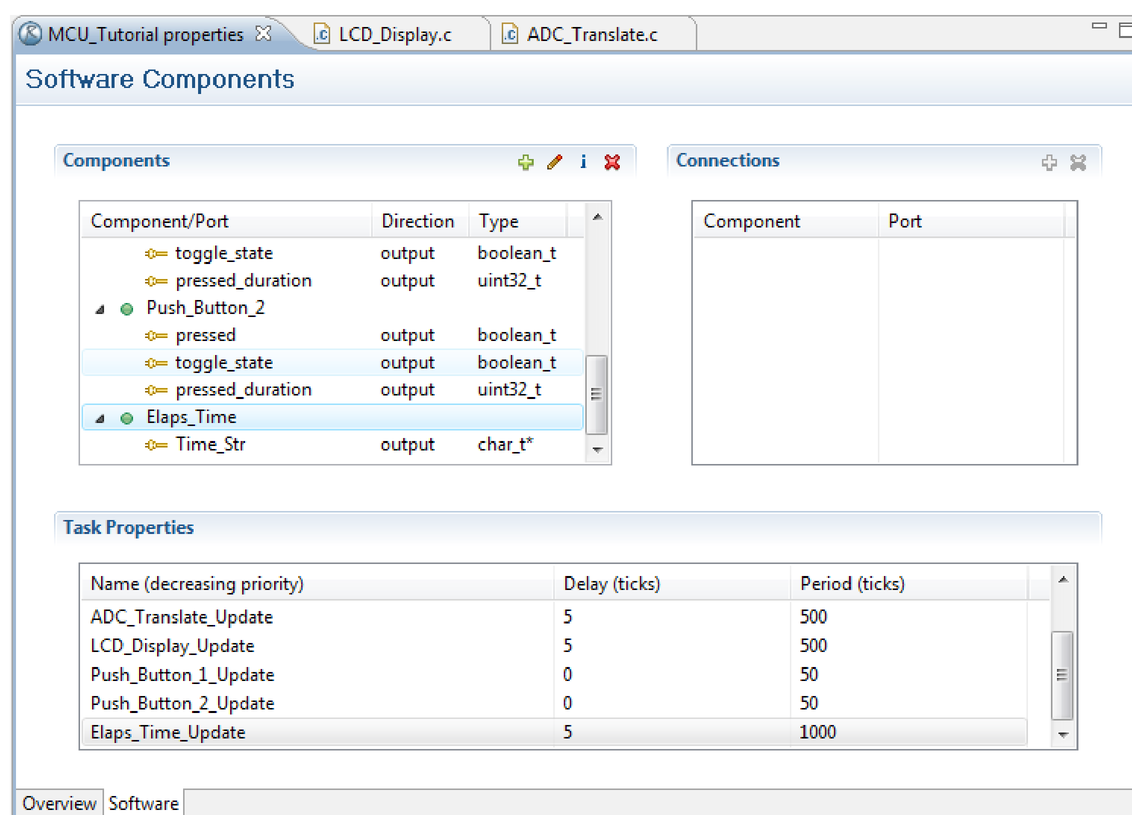

Figure 4-6: The Software Component page after adding the Elaps_Time component

The Elaps_Time_Update task (in Elaps_Time.c) will need to be modified to display the minutes and seconds that have elapsed. Listing 4.2 shows the the required modifications to display the elapsed time.

```
#include <stdio.h>

/**
 * The Time_Str output port
 */
char Time_Str[15];

/**
 * The Update task, executed by the scheduler.
 */
void Elaps_Time_Update(void)
{
        static uint32_t i;
        const uint32_t minutes = ++i / 60;
        const uint32_t seconds = i % 60;
        const char* elapsed_time_string = "Time: %d:%d                          ";

        snprintf(Time_Str, 15, elapsed_time_string, minutes, seconds);
        Time_Str[14] = '\0';

        // Pass the values of the output ports to this function:
        Elaps_Time_Update_Output(Time_Str);
}
```

Listing 4.2: The sample code to display the elapsed time

To display the elapsed time on the LCD screen, the "LCD_Display_Update" task will need to be modified, as shown in Listing 4.3.

The modifications shown in Listing 4.3 shows that two new ports ("*Time_String*" and "*Button2_Pressed*") have been added in the source code. These ports can be manually added into the RapidiTTy frame work, using the "Import port" technique described in Section 4.3 .

Finally, we will connect the remaining unconnected ports. The "*pressed*" port of the "Push_Button_2" component is connected to the "*Button2_Pressed*" port of the "LCD_Display" component. Next, the "*Time_Str*" port of the "Elaps_Time" component is connected to the "*Time_String*" port of the "LCD_Display" component.

The final Software Component page is illustrated in Figure 4-7.

Save, compile and upload you code to check that you have met the requirements set out in Example B.

RapidiTTy MCU

```c
/**
 * The ADC_String input port.
 */
char_t* LCD_Display_ADC_String;

/**
 * The Time_String input port.
 */
char_t* LCD_Display_Time_String;

/**
 * The Button1_Pressed input port.
 */
boolean_t LCD_Display_Button1_Pressed;

/**
 * The Button2_Pressed input port.
 */
boolean_t LCD_Display_Button2_Pressed;

/**
 * The Update task, executed by the scheduler.
 */
void LCD_Display_Update(void)
{
	char display_string[15] = {"Press a button"};

	if (LCD_Display_Button1_Pressed)
	{
		strcpy(display_string, LCD_Display_ADC_String);
	}
	else if (!LCD_Display_Button1_Pressed && LCD_Display_Button2_Pressed )
	{
		strcpy(display_string, LCD_Display_Time_String);
	}

	LCD_Send_String(display_string, 10, 10, MEDIUM_FONT, BLACK, WHITE);

	// Pass the values of the output ports to this function:
	LCD_Display_Update_Output();
}
```

Listing 4.3: Source code modifications to the "LCD_Display_Update" task to display the elapsed time

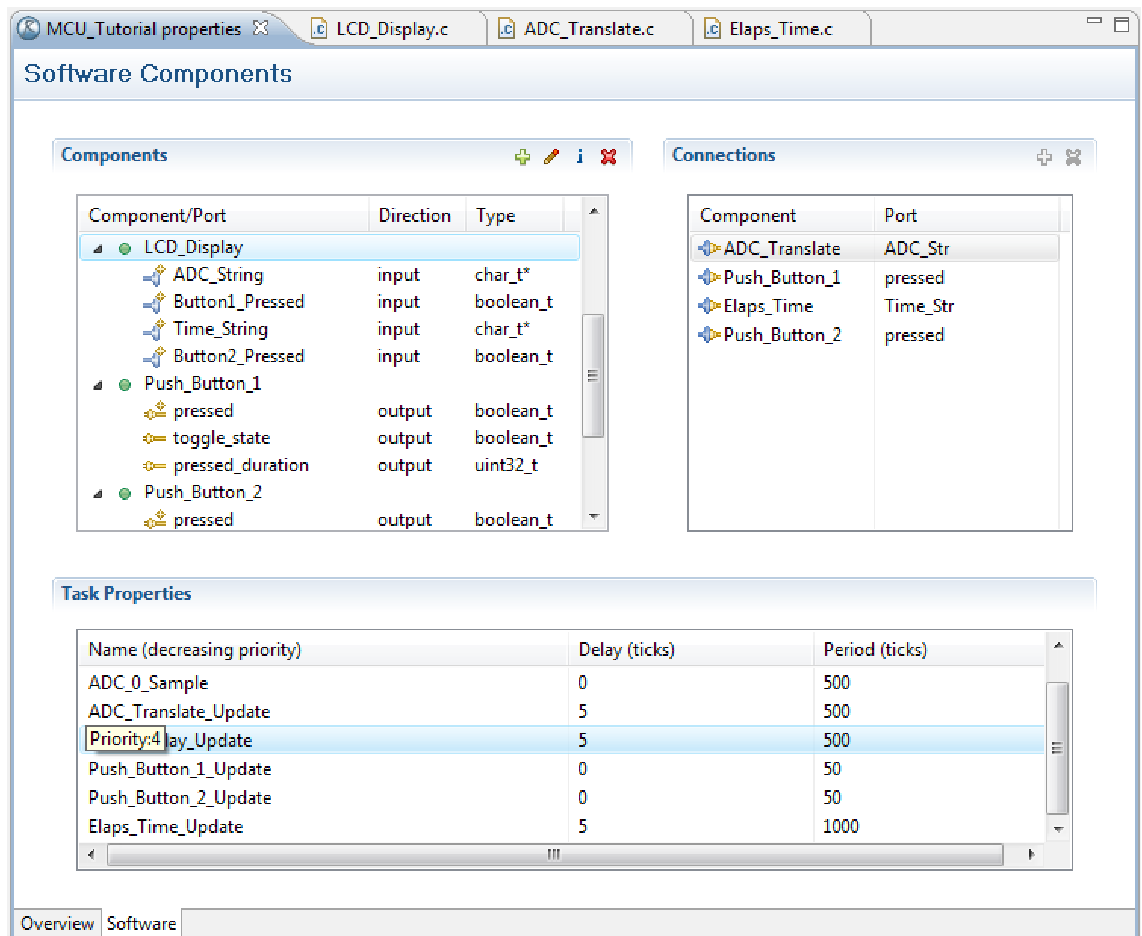

Figure 4-7: The final Software Components page after all the components, ports and port connections have been completed

5 Where do we go from here?

Now that we have completed this tutorial, where can we go from here?

5.1 *RapidiTTy MCU*

For RapidiTTy MCU support, or to ask general questions about embedded systems, you can visit our online discussion forum at http://www.tte-systems.com/forum.

5.2 *Other products in the RapidiTTy family*

RapidiTTy MCU is just one of the products in the RapidiTTy family. The other products are described in this Section.

5.2.1 RapidiTTy FPGA

Where RapidiTTy MCU is designed to work with systems based on commercial-off-the-shelf (CotS) microcontrollers, RapidiTTy FPGA is designed to work with "soft" processor cores running on a Field Programmable Gate Array (FPGA).

RapidiTTy FPGA offers many of the features in RapidiTTy MCU, in addition to a number that are unique. The complete feature list includes:

- Timing analysis for time-triggered systems.
- Full source-code for the Time-Triggered Co-operative Operating System (TTCos).
- Full source-code for the TT3 soft processor core, which consists of:
  - A 32-bit processor core compatible with the MIPS I™ Instruction Set Architecture and capable of running at up to 50 MHz.
  - Predictable timing – one cycle for each of the five pipeline stages.
  - Timer, UART and hardware debugging peripherals.
  - Expansion through a simple peripheral bus.

5.2.2 RapidiTTy x86

RapidiTTy x86 provides the functionality of RapidiTTy MCU, but for higher-end hardware based on the Intel® x86 architecture (including Intel® Atom).

RapidiTTy x86 can be used:

- As a platform for rapid prototyping (designs and code can be moved easily to other members of the RapidiTTy family.
- For deployment of low-volume, high-end applications (taking advantage of the 4Gb+ address space and high CPU performance of embedded PC hardware).

RapidiTTy x86 applications can be booted directly from hard disk, CD-ROM and even USB memory stick. You can also use the integrated simulator to develop and test on the same machine.

5.2.3 RapidiTTy Professional

The RapidiTTy Pro product suite consists of all three development environments: RapidiTTy MCU, RapidiTTy FPGA and RapidiTTy x86 in one package.

5.3 Time-triggered systems

Like all RapidiTTy™ toolsets, RapidiTTy™ MCU is based on time-triggered (TT) technology. Use of TT technology helps to ensure that even new developers can produce reliable embedded systems, and helps to maximise the efficiency of an experienced development team.

Information about time-triggered systems can be found in the book “Patterns for Time-Triggered Embedded Systems”. This book can be downloaded (without charge) from the TTE Systems website: http://www.tte-systems.com/books/pttes