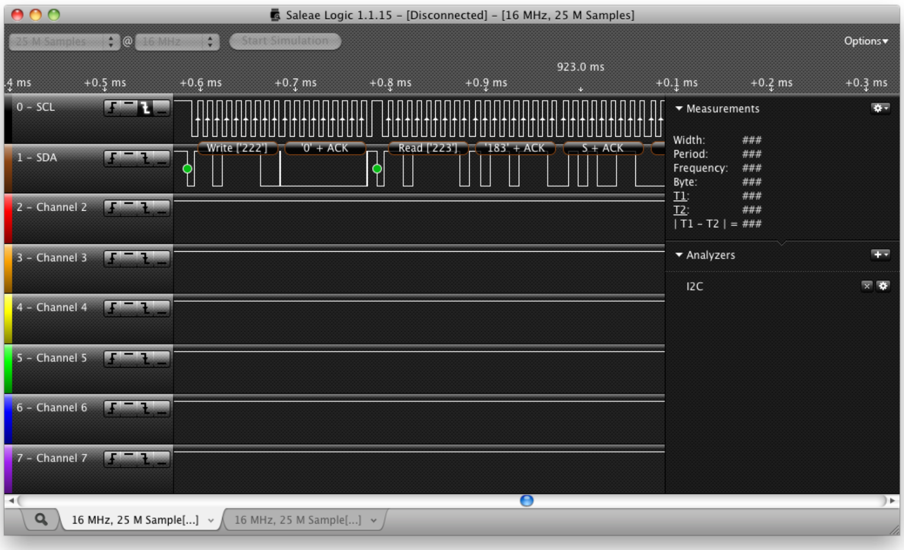
C7

Serial Communications

This exercise is an introduction to serial communications using the `Il Matto` development board. The focus of the exercise is to learn to use the three most common serial communication standards (UART, SPI and I^{2}C) to interface to a number of common peripherals. You will discover how to create a simple debug interface for your embedded platform using the UART interface as well as interfacing a digital potentiometer using the SPI and a real-time clock using I^{2}C.



Schedule

Preparation Time	:	3 hours
Lab Time	:	3 hours

Items provided

Tools	:	
Components	:	MCP4261 5kΩ Dual Digital Potentiometer IC (SPI), MCP7940M Real-time Clock IC (I^2C), 32.768kHz watch crystal, 2×10pF capacitors, 2×2.2kΩ resistors, 2×100nF capacitors. [*Components to be retained by the student after the exercise.*]
Equipment	:	Saleae Logic Analyser [6], Oscilloscope [7]
Software	:	`avr-gcc`, `Eclipse`

Items to bring

Il Matto

c232hm cable

Identity card

Laboratory logbook

Toolkit

Version: November 9, 2012

Steve R. Gunn
Electronics and Computer Science
University of Southampton

Before entering the laboratory you should read through this document and complete the preparatory tasks detailed in section 2.

Academic Integrity – *If you wish you may undertake the preparation jointly with other students. If you do so it is important that you acknowledge this fact in your logbook. Similarly, you will probably want to use sources from the internet to help answer some of the questions. Again, record any sources in your logbook.*

You will undertake the exercise working with your laboratory partner. During the exercise you should use your logbook to record your observations, which you can refer to in the future – perhaps to write a formal report on the exercise, or to remind you about the procedures. As such it should be legible, and observations should be clearly referenced to the appropriate part of the exercise. As a guide the ✎ symbol has been used to indicate a mandatory entry in your logbook. However, you should always record additional observations whenever something unexpected occurs, or when you discover something of interest.

For each Task you should create a new directory so that you have a working version of every program at the end of the lab. Remember to place comments in your code.

You will be marked using the standard laboratory marking scheme; at the beginning of the exercise one of the laboratory demonstrators will mark your preparatory work and at the end of the exercise you will be marked on your progress, understanding and logbook.

Notation

This document uses the following conventions:

✎	An entry should be made in your logbook
`command input`	Command to be entered at the command line

1 Introduction

This lab introduces you to serial communications on an embedded platform. It covers the three most common microcontroller serial communication standards: UART, SPI and I^{2}C.

1.1 Outcomes

At the end of the exercise you should be able to:

- Communicate with a microcontroller and a PC host using the UART interface.
- Control a digital potentiometer from the microcontroller using the SPI.
- Communicate with a real-time clock using the I^{2}C interface.
- Use a logic analyser to observe and debug serial communications.

2 Preparation

Study and familiarise yourself with the provided `C` source code files. It is not necessary to understand all implementation details, but it is important that you understand the interface and the task undertaken by each function. Answer the following questions with reference to the `Il Matto` board running at 3.3V:

1. What does the acronym UART stand for? ✎
2. What baud rates are supported by the UART for serial communication? ✎
3. When an 8-bit ASCII character is transmitted over the UART how many bits are sent, and what does each one represent? ✎
4. How many devices can be connected to the UART simultaneously? ✎
5. What does the acronym SPI stand for? ✎
6. What clock rates are supported by the SPI for serial communication? ✎
7. When an 8-bit ASCII character is transmitted over the SPI how many bits are sent, and what does each one represent? ✎
8. How many devices can be connected to the SPI simultaneously? ✎
9. What does the acronym I^{2}C stand for? ✎
10. What clock rates are supported by the I^{2}C for serial communication? ✎
11. When an 8-bit ASCII character is transmitted over the I^{2}C how many bits are sent, and what does each one represent? ✎
12. How many devices can be connected to the I^{2}C simultaneously? ✎

3 Laboratory Work

Open the Eclipse application on the computer you are using and when asked for the workspace you wish to use, make sure the checkbox for *use as default* is not ticked and type `H:\ELEC1201\Labs\C7`.

Remember to create a new C project (File -> New -> New project.. -> C/C++ -> C Project), for each task of the exercise and name it with the appropriate part of the exercise for easy future reference, e.g. `C7_3.1`. This time you should select "AVR Cross Target Application" -> "Empty Project" as "Project type" and add in a new C source file.

3.1 Using the UART as a debugging aid for embedded programming

You have already used the C232HM cable [2] which uses the FT232H IC [1] to enable `Il Matto` to communicate with a PC host in the lab X2. Here you will discover how to use the UART to enable you to display output and error messages as your embedded program executes, which can be very helpful in diagnosing problems or logging statistics.

The ATMEGA644P contains two UART interfaces which are both located on port `D`. You will be using the first UART, but you could equally use the second UART if the first was already being used by your application. You will need to connect three wires from the C232HM cable (TX, RX and GND) to make the connection to the first interface on port `D`. You can use single strand hook-up wire to make the connection. Make sure to swap over the TX and RX lines from the cable to the board, so that the cable TX line goes to the RX line on the board and vice-versa. Do not forget the ground connection.

Note that the C232HM cable will provide enough power over the TX *line to power the microcontroller, preventing `S1` being able to fully power cycle and reset the device. In order to restart the program it may be possible to reset the device by pressing `S2` and exiting the boot loader by executing `avrdude -c usbasp -p m644p`. Alternatively, connect the* TX *line after the microcontroller has been powered-up.*

You have been provided with a file `debug.h` which provides all of the necessary initialisation code to redirect all of the standard input/output to the first UART. To enable the UART for debugging, include this file and call the function `init_debug_uart0()` at the beginning of the function `main()`.

Take the first program that you wrote in lab C6 and modify it so that it initialises the debug UART and uses `printf()` to also write the current number on the display to the debug UART. It is not required to have the seven-segment display connected in this exercise, where the purpose is to demonstrate the ability to observe internal variables in executing embedded programs.

Start up a terminal on the host PC. On a Linux or Mac machine you can use the command

```
screen /dev/tty.usbserial??????? 9600
```

On a Windows machine you can use `hyperterminal` or `PuTTY` to make the connection to the correct COM port (usually the highest number in the list - you can check in device manager by unplugging the C232HM cable and seeing which COM port disappears). Select 9600 for the baud rate, no flow control, no parity and one stop bit.

Confirm that you observe the correct output on the terminal.

Connect up the Saleae Logic and verify that you can observe the data being transmitted to the PC host. Remember to set the protocol to Async Serial.

Modify your program to issue an `fprintf(stderr,"Count overflow\n\r");` message when the count is reset from 9 to 0 to test that you are able to capture messages sent to `stderr`.

Confirm that you are able to get your microcontroller to read in a number from `stdin` by waiting for a key to be pressed before the program resumes counting up from 0.

3.2 Controlling a digital potentiometer using the SPI

In this section you will learn how to control a digital potentiometer using the SPI. You will configure the potentiometer to behave like a simple digital to analogue converter by connecting its ends across the supply rails and observing the voltage on the wiper.

Connect up the MCP4261 digital potentiometer IC [4] on your breadboard. /CS should be connected to ground as you only have a single device on the SPI. Connect VDD to VCC and VSS to GND. Connect the SPI signals which can be found on port B to SCK, SDI and SDO. Connect /SHDN to VCC. Connect P0A and P1A to VCC and connect P0B and P1B to GND. Connect a 100nF decoupling capacitor across the IC as close as possible to VDD and VSS. Connect P0W and P1W to channels 1 and 2 of the oscilloscope respectively using $\times 10$ probes.

You have been provided with two files `spi.h` and `spi.c` which will configure the SPI in Master mode and provide functions for transmitting and receiving data. You have also been provided with two additional files `digitalPot.h` and `digitalPot.c`. Unfortunately, the file `digitalPot.c` has not been implemented and it is your task to fill in the implementation to provide functions to set the potentiometer to a given value, as well as providing implementations for incrementing or decrementing its value.

Write a program that sets each potentiometer to zero and then increments the value 255 times before reseting the value to zero and repeating. Confirm that you can observe a sawtooth waveform on each channel of the oscilloscope and calculate the maximum frequency at which the potentiometer can be incremented.

Connect up the Saleae Logic and verify that you can observe the data being transmitted between the microcontroller and the digital potentiometer. The connections can be made to the ISP connector on the board since these bring out the SPI connections from port B. Remember to set the protocol to SPI. What is the delay between the transmitted and received byte?

3.3 Communicating with a real-time clock using I²C

In this section you will learn how to communicate with a real-time clock/calendar (RTCC) using I²C.

Connect up the MCP7940M RTCC IC [5] on your breadboard according to figure 1. Connect the I²C signals which can be found on port C to SDA and SCL.

> *A common problem with I²C failing to work is failing to install pull-up resistors on* **both** *SDA and SCL lines. Here, the internal pull-ups on the AVR are marginally too weak and for reliable operation it is best to use external values* $< 5k\Omega$.

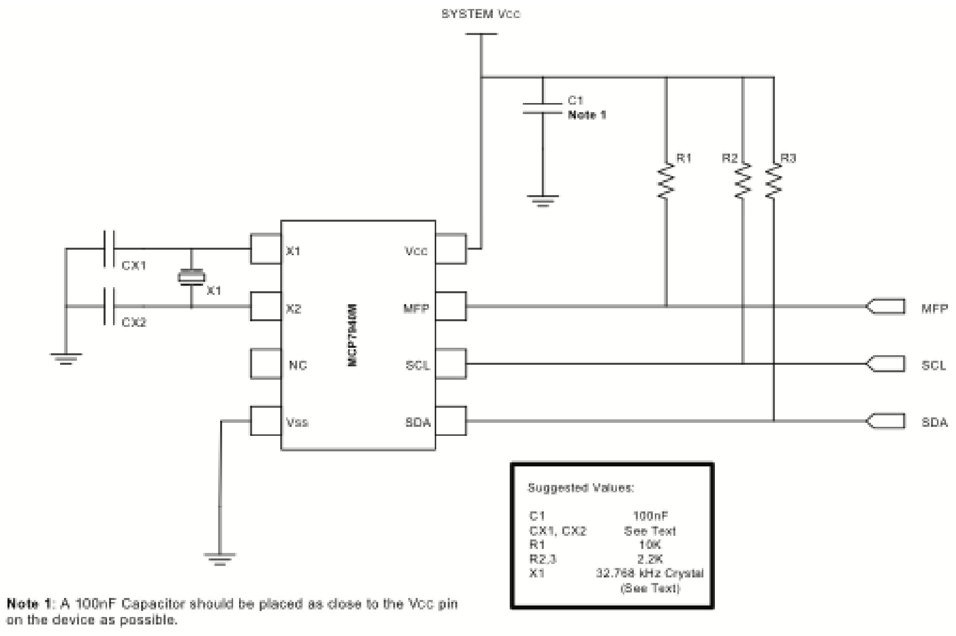


FIGURE 1: RTCC connections

You have been provided with two files `i2c.h` and `i2c.c` which will configure the I²C in Master mode and provide functions for transmitting and receiving data. You have also been provided with two additional files `RTCC.h` and `RTCC.c`. Unfortunately, the file `RTCC.c` has not been implemented and it is your task to fill in the implementation to provide functions to initialise and start the RTCC, set it to a given time, and read the current time from it. You will need to start the clock by setting the MSB in the first memory address of the RTCC.

Write a program that sets the current time to 12:34:56 and starts the clock. The program should then go into a forever loop. Each time around the loop it should wait five seconds before requesting the current time from the RTCC. Use the debug UART technique that you learnt in section 3.1 to display the received time on a PC terminal.

Connect up the Saleae Logic and verify that you can observe the data being transmitted between the microcontroller and the RTCC. Remember to set the protocol to I²C.

4 Optional Additional Work

4.1 Oscilloscope Clock

Use the two digital potentiometers to draw the current time from the RTCC on the oscilloscope in X-Y mode. You are supplied with a vector graphics package consisting of three `C` source code files: `vectorFont.h`, `vectorGraphics.h` and `vectorGraphics.c`. This package can draw lines and uses the Hershey [3] vector font to draw characters.

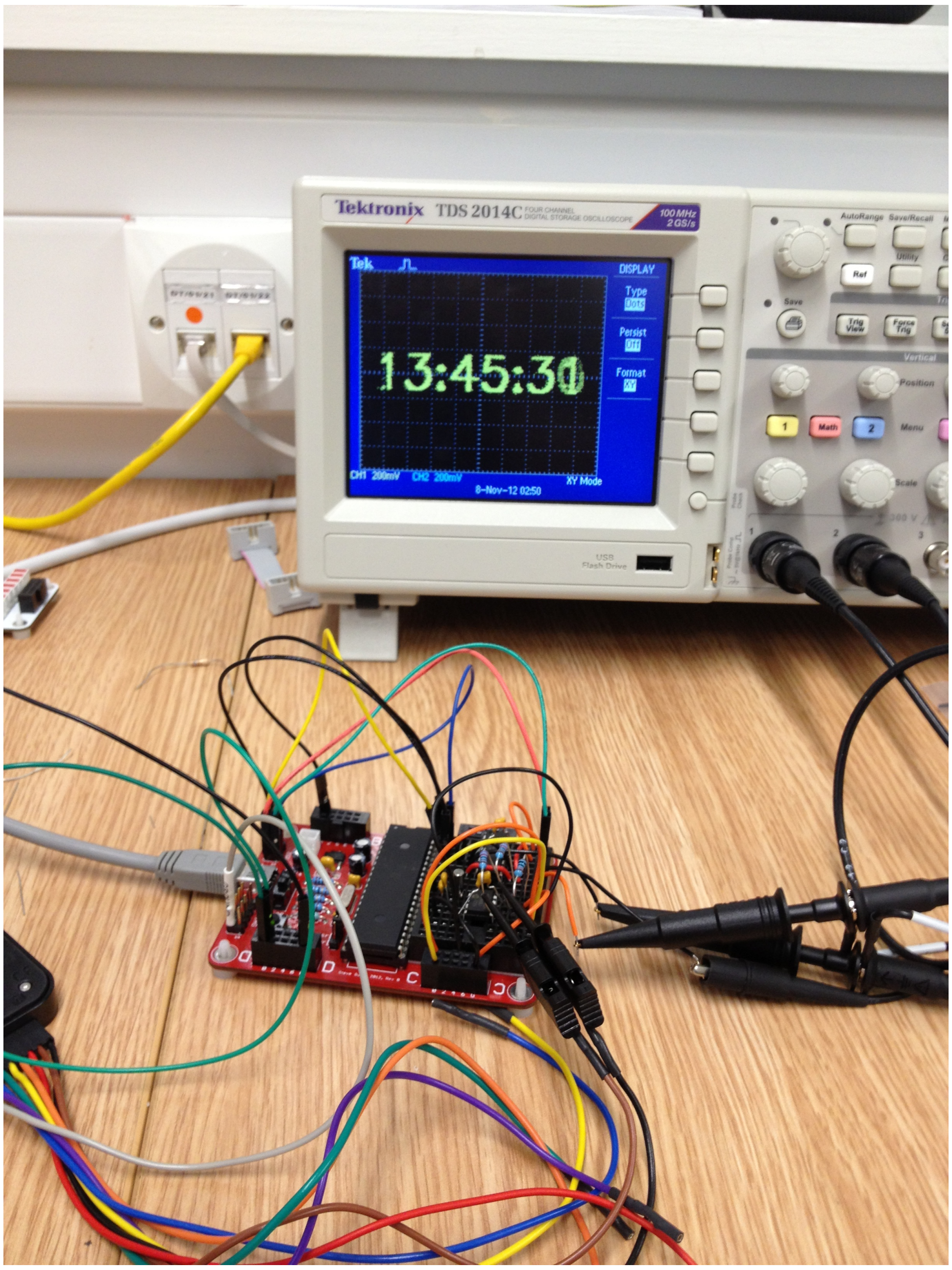


FIGURE 2: Oscilloscope Clock

References

[1] FTDI Chip. FT232H Single Channel Hi-Speed USB to Multipurpose UART/FIFO IC. Datasheet FT_000288, Future Technology Devices International Ltd, 2012. URL http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232H.pdf.

[2] FTDI Chip. C232HM USB Hi-Speed to MPSSE Cable Datasheet. Datasheet FT_000401, Future Technology Devices International Ltd, 2012. URL http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_C232HM_MPSSE_CABLE.PDF.

[3] A. V. Hershey. Hershey Vector Font, 1997. URL http://paulbourke.net/dataformats/hershey/.

[4] Microchip. Dual SPI Digital Potentiometer Datasheet, 2008. URL http://www.microchip.com/S/?q=DS22059.

[5] Microchip. I2C Real-Time Clock/Calendar Datasheet, 2012. URL http://www.microchip.com/S/?q=DS22292.

[6] Saleae. Logic Analyser (Logic). User's Guide 1.1.15, 2012. URL https://secure.ecs.soton.ac.uk/notes/ellabs/reference/equipment/std-bench/Logic%20Analyser-%20Saleae%20User%20Guide.pdf.

[7] Tektronix. Digital Storage Oscilloscope (TDS1000C). User Manual Rev. A, 2006. URL https://secure.ecs.soton.ac.uk/notes/ellabs/reference/equipment/std-bench/Tektronix_TDS1000C_2000C_User_Manual.pdf.