1 Prac 3 - I2C and PWM

1.1 Overview

Before connecting to the internet for the first time, you may have noticed that the time on your Pi is a little strange. This is because the Pi doesn't have what is called an RTC (real time clock). Instead, it relies NTPD (Network Time Protocol Daemon) to fetch, set and store the date and time. However, this might be problematic as you may not always have an internet connection, and if your Pi doesn't have the correct localisation options, you may end up with the wrong time due to timezone settings. It's possible to add an RTC to the Raspberry Pi to hold the system time correctly, but for this practical we're simply going to interface with the RTC using I2C, and set the time using buttons and interrupts.

1.1.1 Design overview

You will be creating a modified version of a binary clock. For your convenience, an image displaying the expected operation is shown below in Figure 1.

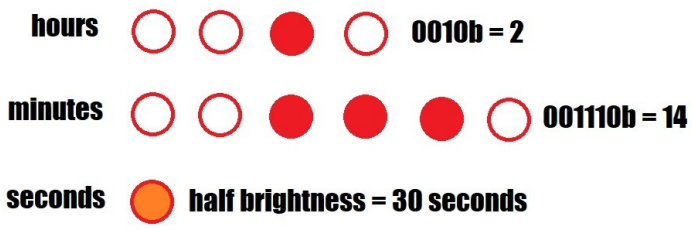

Figure 1: A Modified Binary Clock Showing 2:14 (AM or PM)

Two buttons should be connected, using interrupts and debouncing, which do the following:

1. Button 1 - Fetches the hours value from the RTC, increases it by 1, and writes it back to the RTC.
2. Button 2 - Fetches the minutes value from the RTC, increases it by 1, and writes it back to the RTC.

You cannot use any time libraries in your script, i.e. make sure you are using the I2C communication protocol between the RTC and Pi in your script.

1.2 Pre-prac requirements

This section covers what you will need to know before starting the practical.

1.2.1 Knowledge Areas

- I2C
- PWM
- Knowledge about BASH as learnt in previous pracs
- Read the data sheet for the MCP7940M [1]
- You should acquaint yourself with the Wiring Pi documentation, available here.

1.2.2 Pre-prac Submissions

A circuit diagram for the practical. You will need to upload a schematic detailing how you are going to set up the RTC and connect it to the Raspberry Pi. It should also show how the buttons and LEDs are connected to the Pi. Software used for drawing the circuit is up to you. [2]

1.3 Outcomes

You will learn about the following aspects:

- I2C
- Real Time Clocks
- Wiring Pi
- Starting a script on boot on the Raspberry Pi

1.4 Deliverables

At the end of this practical, you must:

- Demonstrate your working implementation to a tutor
- Submit your code on Vula alongside a short write-up (no more than three - 3 - pages) detailing how you completed the practical. It is strongly recommended you do use git, but a GitHub link is not required for practical submission. See the write up format below for further guidelines. Your code and write up should be contained in a compressed folder. Your write up should be a PDF. It is not required to complete the write up using LaTeX.

[1] The Pi has onboard resistors for I2C, so you don't need to use those.

[2] You should be aware that Wiring Pi has 3 different potential modes for pin usage. Be aware of that during config and your wiring.

1.5 Hardware Required

- Configured Raspberry Pi
- RPi Power Source
- Ethernet Cable
- A breadboard
- 2 x push buttons
- 11 x LEDs (total)
- 11 x Resistors (total)
- Dupont Wires
- RTC
- Capacitors (kit)
- Crystal

1.6 Walkthrough

1. If you didn't in Prac 0, start by enabling I2C in raspi-config
2. Do a git pull in the prac source folder to fetch the Prac 3 content
3. Build the circuit you designed in the pre-prac
4. Run `$ gpio i2cdetect` to see if you can see the RTC (0x6F)
5. Open `BinClock.c` and write the code required. Some function templates are made available to give you a guide, but you may be required to write more functions as you see fit. Function definitions are placed in `BinClock.h`
    - Be sure to take care with regards to which pin numbering is used. You can run `$ gpio readall` to check pinouts and current assignments
6. As a fun task, you can get the clock to run on startup using `rc.local`. You can read through this guide.[3]

1.7 Some Hints

1. Read the Docs. This includes the datasheet for the RTC (which you will absolutely have to do), as well as the documentation of Wiring Pi.
2. You will need to debounce your button presses. You can use hardware debouncing, but this circuit is a bit complex, so to save space we recommend software debouncing.
3. The source code provided to you has a lot of implementation in it already. Ensure you read and understand it before embarking out on the practical.
4. Wiring Pi doesn't have a cleanup function like RPi.GPIO (Python). It's suggest you write your own that's caught and executed upon a keyboard interrupt.
5. You may notice a function called `toggleTime`. This function pulls the time from the Raspberry Pi, and writes it to the LEDs. This is a nice option for "reloading" the system time to RTC.

[3] Admittedly, this isn't a very efficient or effective way to keep track of time. There are services that will keep track of time for us in the Pi OS, rather than us polling the RTC every second. We could even write our own code better - letting the time on the Raspberry Pi "freewheel" and having an interrupt fetch the time from the RTC every few minutes (or hours, or days or just on boot). At this point though, it becomes more sensible to simply use the hwclock service.

1.8 Write up Format

Your write up should consist of the following headings:

1. Introduction
2. A UML use-case diagram of the system
3. I2C communication using Wiring Pi
   (a) Initialisation
   (b) Send data
   (c) Receive Data

   Include example timing diagrams in your descriptions
4. A short paragraph on interrupts and debouncing, and why they're important in Embedded Systems (one advantage of each is enough)
5. The circuit diagram from the pre-prac, with any changes you've made updated in the circuit diagram