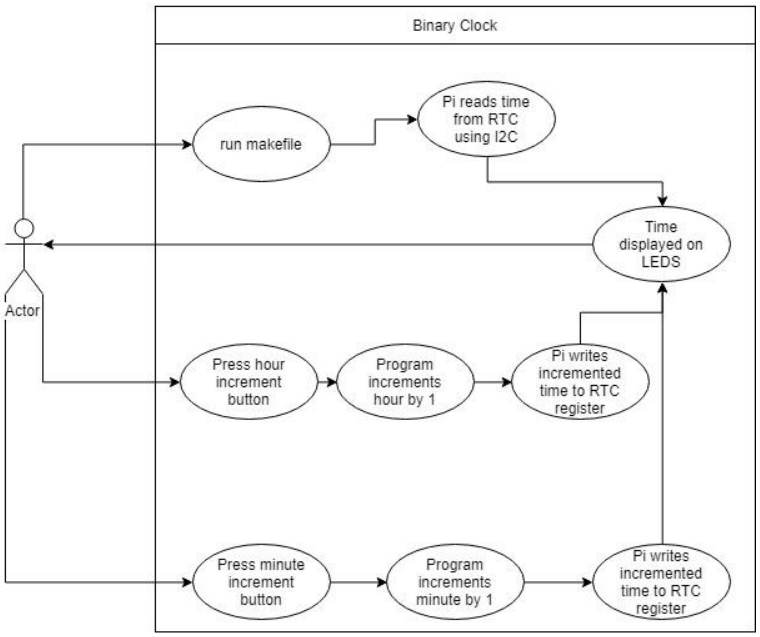
EEE3096S: Practical 3 Report
JFFMAH001 NDKNIC001

Introduction

For this practical we were required to interface with the RTC using I2C to display real time on a binary clock using LEDS. The hour and minutes could be incremented using buttons and interrupts, hence two buttons were used in the circuit. The first button was used to fetch the hours value from RTC, increase the hour by 1 and then the pi would write this new time back to the RTC. The time was then displayed on the LEDS. The second button was used to fetch the minutes value from RTC, increase the minutes value by 1 and then write the new time back to the RTC. This time was then displayed on the LEDS.

UML Diagram of System



I2C Communication using WiringPi

(a) Initialisation

Initialising the RTC is done by setting up the RTC using the method wiringPiI2CSetup(address) with the address of the RTC being 0x6f.

(b) Send data

Writing DATA to the RTC was done using the method wiringPiI2CWriteReg8 (int fd, int reg, int data) where “fd” is the instance of the RTC, “reg” is the register and “data” is the data to be sent. The registers used in this practical were the seconds, minutes and hour which had addresses 0x00, 0x01 and 0x02 respectively.

(c) Receive Data

Receiving data from the RTC is the same as writing to it, however, the master becomes the RTC. The Pi sends a read command and receives the data by calling wiringPiI2CReadReg8 (int fd, int reg). The registers are the same as those used to send the data.

Example timing diagrams of how the data goes between the two devices is shown below. To initialize the oscillator, the value 0x80 is sent the seconds register.

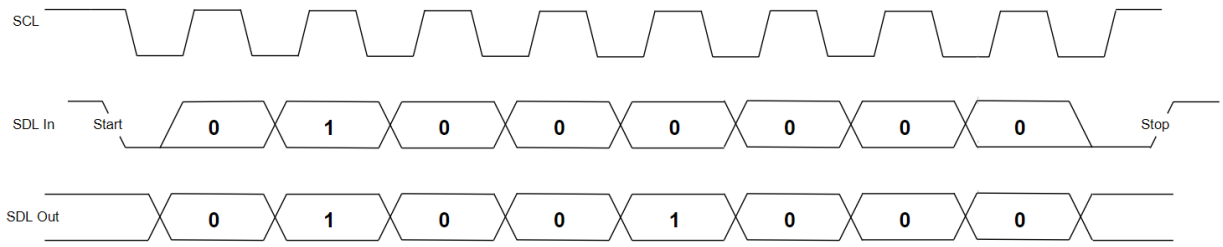


Fig 1. Data written to the RTC seconds register and data read

The timing diagram shows the 8 bit data that is sent to the seconds register and the data that is read.

Interrupts and Debouncing

Interrupts

An interrupt is a signal that prompts the operating system to stop work on one process and start work on another process [1]. Once the new process is completed, the previous one will continue. For this practical, interrupts were used when the push buttons were pressed to increment hours and minutes. The benefit of interrupts is that it allows the operating system to function smoothly when performing multiple processes, such as incrementing the hours/minutes while getting the time from the RTC.

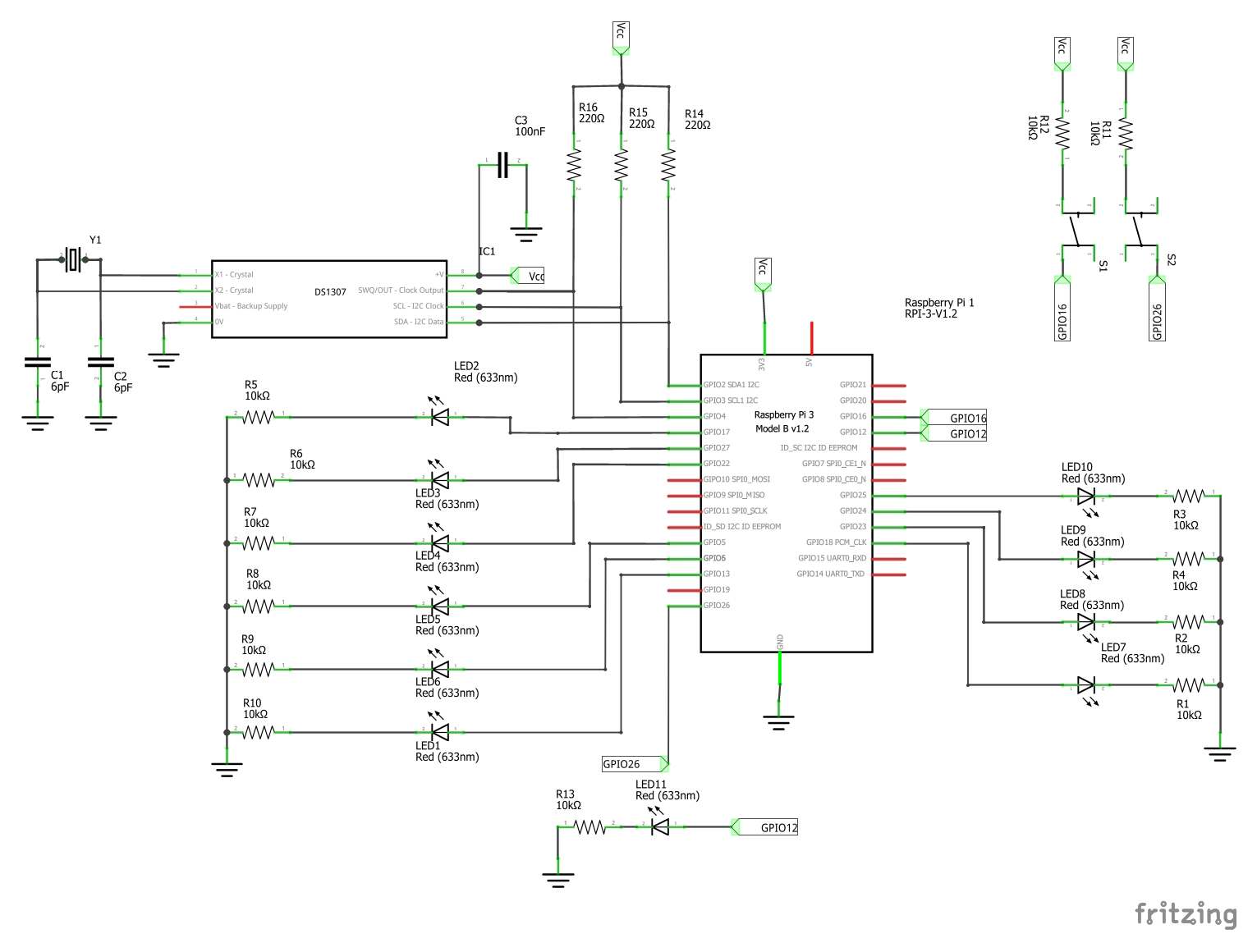
Debouncing

Bouncing in the case of this practical is when the contacts in the buttons being used don't make contact cleanly and causes multiple signals to generate when the contacts within the button open or close [2]. For this practical we've used software debouncing. Debouncing stops the signal from "bouncing up and down" – essentially it holds the signal steady. This is useful as it ensures that only a single signal will be acted upon when pressing the buttons.

References

[1] https://www.techopedia.com/definition/3373/interrupt-computing

[2] https://whatis.techtarget.com/definition/debouncing


Vcc
R16
220Ω
R15
220Ω
R14
220Ω
C3
100nF
Y1
IC1
DS1307
X1 - Crystal
X2 - Crystal
Vbat - Backup Supply
0V
+V
SWQ/OUT - Clock Output
SCL - I2C Clock
SDA - I2C Data
Vcc
C1
6pF
C2
6pF
R12
10kΩ
R11
10kΩ
S1
S2
GPIO16
GPIO26
Raspberry Pi 1
RPI-3-V1.2
3V3
5V
GND
GPIO2 SDA1 I2C
GPIO3 SCL1 I2C
GPIO4
GPIO17
GPIO27
GPIO22
GPIO10 SPI0_MOSI
GPIO9 SPI0_MISO
GPIO11 SPI0_SCLK
ID_SD I2C ID EEPROM
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
Raspberry Pi 3
Model B v1.2
GPIO21
GPIO20
GPIO16
GPIO12
ID_SC I2C ID EEPROM
GPIO7 SPI0_CE1_N
GPIO8 SPI0_CE0_N
GPIO25
GPIO24
GPIO23
GPIO18 PCM_CLK
GPIO15 UART0_RXD
GPIO14 UART0_TXD
GPIO16
GPIO12
GPIO26
R5
10kΩ
R6
10kΩ
R7
10kΩ
R8
10kΩ
R9
10kΩ
R10
10kΩ
LED2
Red (633nm)
LED3
Red (633nm)
LED4
Red (633nm)
LED5
Red (633nm)
LED6
Red (633nm)
LED1
Red (633nm)
LED10
Red (633nm)
LED9
Red (633nm)
LED8
Red (633nm)
LED7
Red (633nm)
R3
10kΩ
R4
10kΩ
R2
10kΩ
R1
10kΩ
R13
10kΩ
LED11
Red (633nm)
fritzing