ASSIGNMENT – 1

CONNECTED EMBEDDED SYSTEMS (EE513)

STUDENT NAME: AKSHAT SINGH

STUDENT ID:19210931

Aims and Objectives

Interfacing Embedded Systems to the Real World. Raspberry Pi has no real-time clock and it doesn't keep record of time after it is switched off for a significant amount of time. Since Raspberry pi is used for IOT applications such as servers, GPS, Utility power meters, Telematics. It sometimes needs to send or receive signals and to incorporate these devices into a network for which time is important. Only condition when there won't be any glitch/conflict is same time.

In this Assignment I am using the following.
Equipment: Raspberry Pi 3B+, RTC-DS3231, Hook-up wires, Mini Breadboard, LED, Resistors.
Programming language: C++
Operating system: Linux

Real-Time Clocks (RTCs) - DS3231

The DS3231 is a low-cost, extremely accurate I2C real-time clock (RTC) with an integrated temperature- compensated crystal oscillator (TCXO) and crystal.

It holds details about the seconds, minutes, hours, day, date, month, and year.

The clock works with an AM / PM indicator, either in the 24-hour or 12-hour configuration.

There are two programmable time-of-day alarms and one programmable square-wave output.

Address and data are serially transferred via a bidirectional bus with I2C.

PHYSICAL CONNECTION OF THE RTC MODULE TO THE I2C BUS [5%]

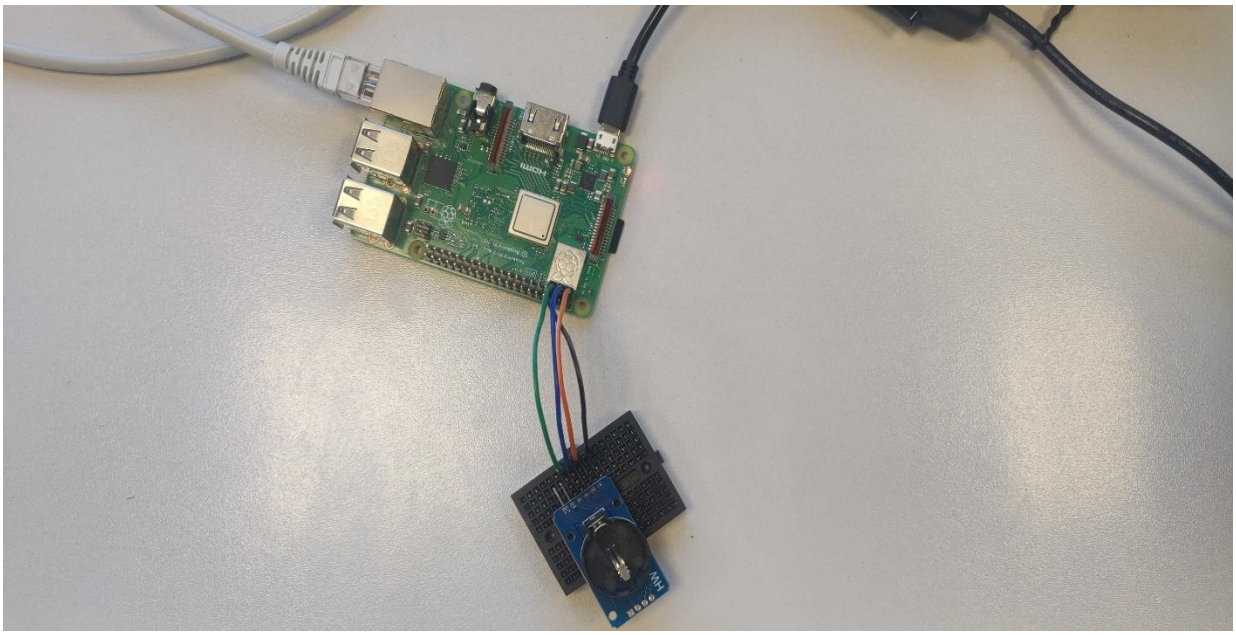
The RTC-DS3231 supports i2c serial interface and it can be connected to Raspberry pi with 4 wires. The connections are VCC (power), GND (ground), SDA (Serial Data Pin(I2Cinterface)) and SCL (Serial Clock Pin (I2C interface)). The connection was done as mentioned below where pin 1,3,5,6 are of Raspberry Pi3B+ and image has been provided for the successful connection.

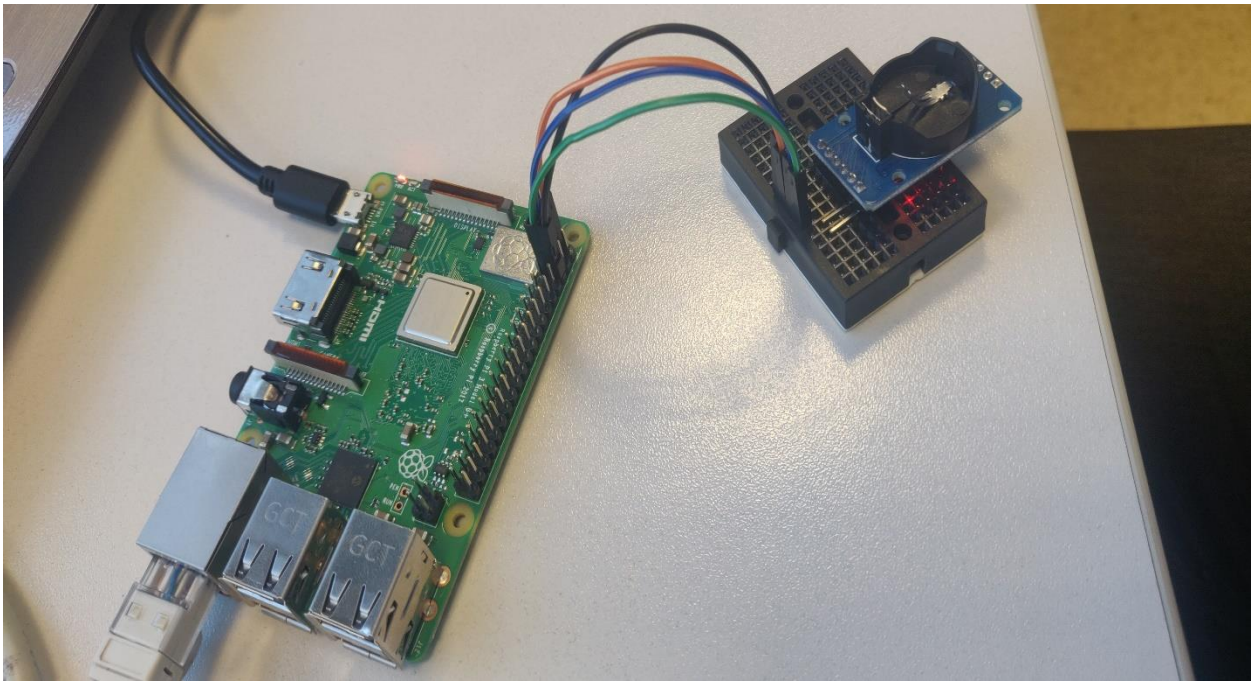
Pin 1: 3.3V connected to VCC of RTC

Pin 3: 12CI_SDA connected to SDA of RTC

Pin 5: 12CI_SCL connected to SCL of RTC

Pin 6: GND connected to GND of RTC



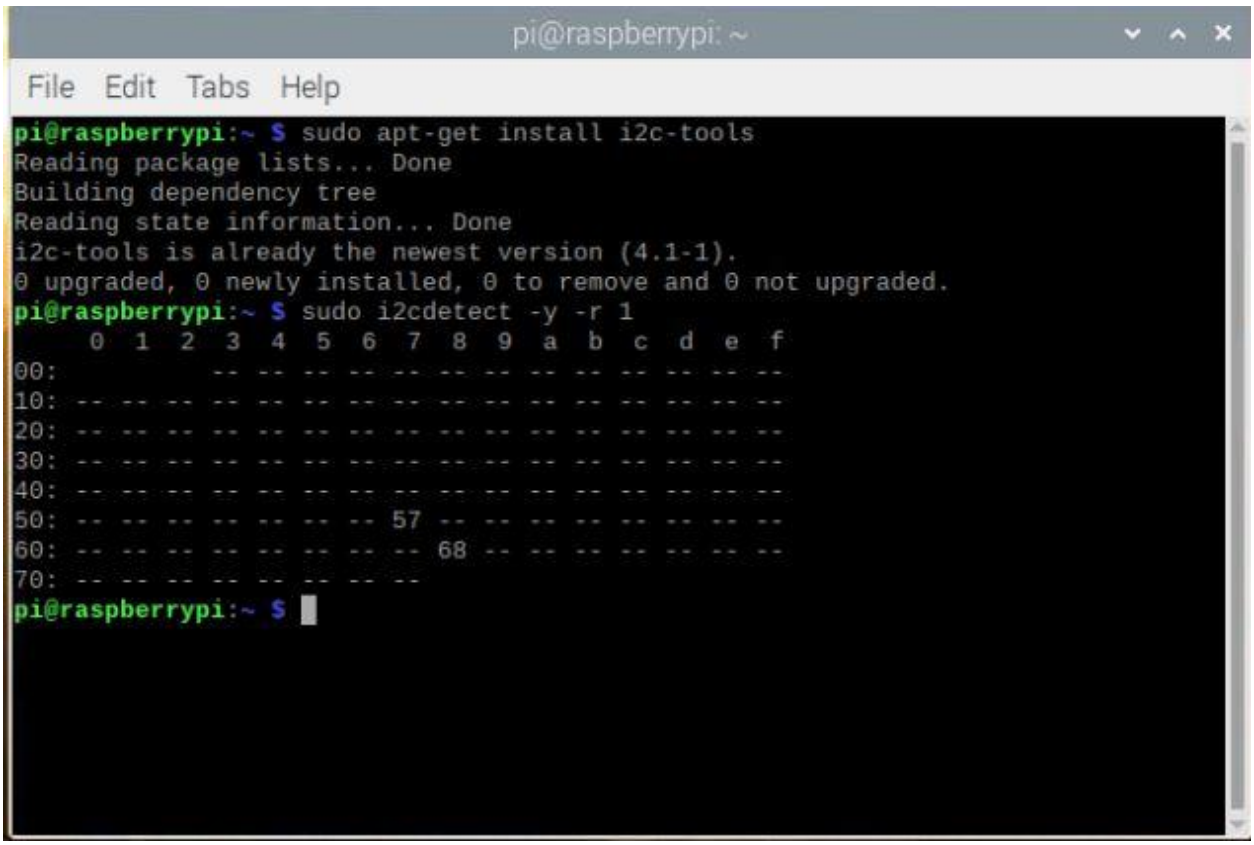

GCT
GCT

DISCOVERY AND TESTING OF THE MODULE USING LINUX-BASED I2C-TOOLS [10%]

As the RTC-DS3231 is connected to Raspberry Pi, Linus based i2c commands were used to check if the RTC is well connected and functioning fine.

These commands were executed in the terminal:

i2cdetect -y -r 1: It explains about the i2c bus with the address of the RTC connected to it. where '1' indicates that the device is connected to the i2c bus 1. When we execute it, we can see that the number 0x68 which is address of the RTC. Hence RTC is successfully connected.



i2cdump -y 1 0x68: It displays the values in registers at the address 0x68. Here we can say that we are able to read the registers of the RTC. These details can be used later to read and set the values.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- 57 -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $ sudo i2cdump -y 1 0x68
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 34 00 01 01 01 01 00 00 00 00 00 00 00 21 1c 88    4.????.......!??
10: 00 18 00 XX XX XX XX XX XX XX XX XX XX XX XX XX    .?.XXXXXXXXXXXX
20: XX XX XX XX XX XX XX XX XX XX XX XX 00 00 00 00    XXXXXXXXXXXX....
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
pi@raspberrypi:~ $
```

i2cdump -y 1 0x68 b: It displays the values stored in registers at the address 0x68.

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
pi@raspberrypi:~ $ sudo i2cdump -y 1 0x68 b
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 08 04 01 01 01 01 00 00 00 00 00 00 00 21 1c 88    ??????.......!??
10: 00 18 40 XX XX XX XX XX XX XX XX XX XX XX XX XX    .?@XXXXXXXXXXXX
20: XX XX XX XX XX XX XX XX XX XX XX XX 00 00 00 00    XXXXXXXXXXXX....
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
pi@raspberrypi:~ $
```

NOTE: RTC-DS3231 has deactivated its oscillator and set the time and most of its data to 0. With the assignment support material, a C file was given to read the time. Just a minor changes were required changing i2c-2 to i2c-1. I was able to read the time and played around to understand the registers better. CH bit in buffer was default set to '1' and this disabled the oscillator. Then I enabled oscillator using i2c command: i2cset -y 1 0x68 0x00 0x00 and set entire buffer to 0 which forced the oscillator to start. From here I started working with C++ class.

READ AND DISPLAY THE CURRENT RTC MODULE TIME AND DATE. [10%]

Basic read/write functions of C++ are used to open the file to read the data and store it in the buffer. It declares an integer 'file' and this opens the i2c bus 1 with the code as: "file=open("/dev/i2c-1", O_RDWR)". This code is using buffer of 15 so I kept the size as 19 which is slightly more than the one being used by the code. I have seen in the data sheet of the DS3231, the time is stored in buffers 00h, 01h, and 02h as seconds, minutes, hours respectively. Note that the read data is in BCD format, and must be converted to decimal before printing. According to the datasheet we know the following address.

00h	Seconds
01h	Minutes
02h	Hours
03h	Day
04h	Date
05h	Month
06h	Year
07h	Alarm 1 Seconds

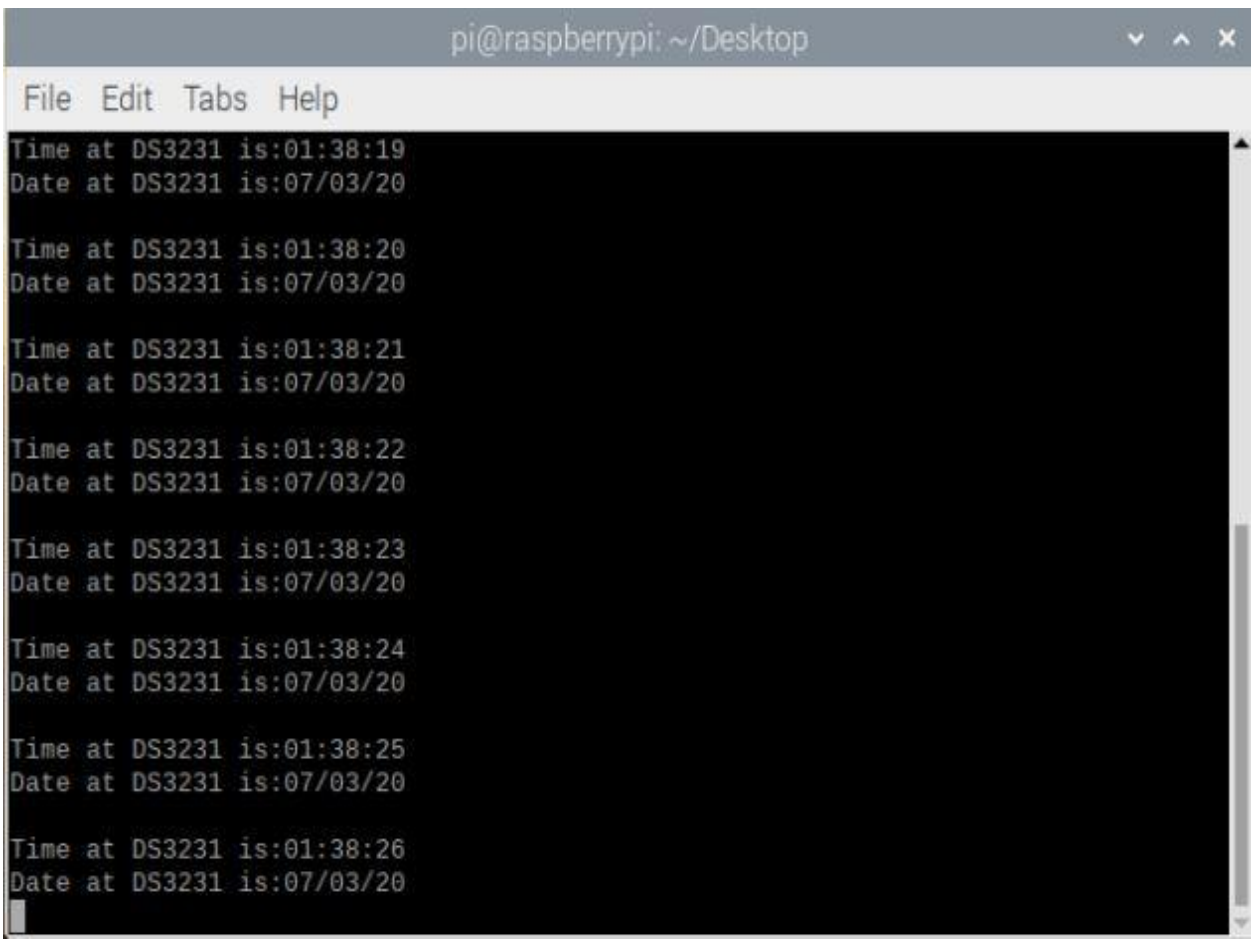
```
#define BUFFER_SIZE 19
#define RTCaddress      0x68
#define Seconds         0x00
#define Minutes         0x01
#define Hours           0x02
#define Date            0x04
#define Month           0x05
#define Year            0x06
```

```
//Read the Registers
int RTCDS3231::ReadRegisters() {
        file=open("/dev/i2c-1", O_RDWR);
        ioctl(file, I2C_SLAVE, 0x68);
        char writeBuffer[1] = {0x00};
        if(write(file, writeBuffer, 1)!=1){
        perror("Failed to reset the
address\n");
        }
        read(file, buffer, BUFFER_SIZE);
        close(file);
        return 0;
}
```

THIS BLOCK IS TO PRINT THE TIME AND DATE

```
cout<<endl;
cout << "Time at DS3231 is:" <<setw(2)<<setfill('0')<<this->bcdToDec(buffer[2])
<<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[1])<<":"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[0])<<endl;


cout << "Date at DS3231 is:" <<setw(2)<<setfill('0')<<this->bcdToDec(buffer[4])
<<"/"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[5])<<"/"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[6])<<endl;
```

Here is the output as expected since the RTC was active and then switched off to for a certain period to read the existing time. And I observed that the time **didn't change** after I switched it off.

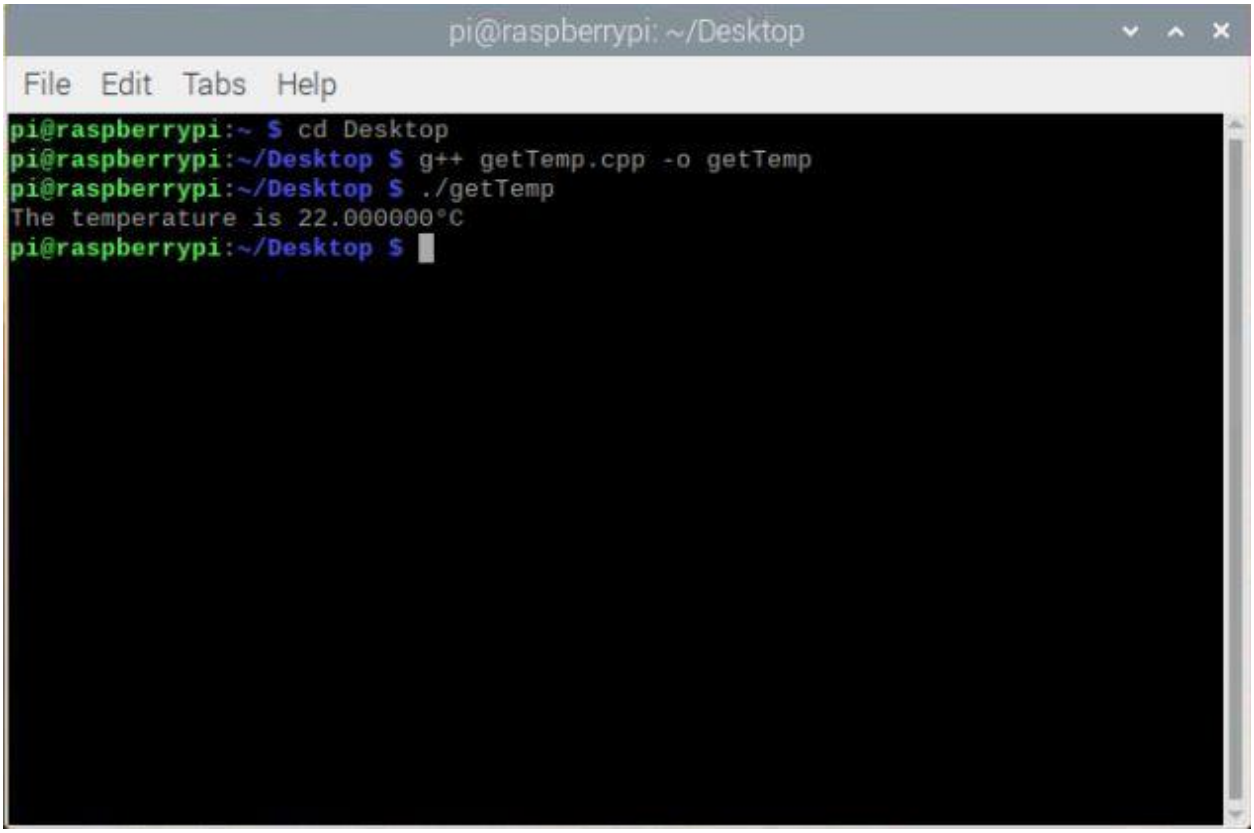


READ AND DISPLAY THE CURRENT TEMPERATURE [5%]

According to the datasheet we know that the temperature registers are 11h and 12h. The integer portion is at address 0x11 and the fractional portion is at 0x12. The temperature gets set to default 0 degree when the power reset is done. The temperature registers are updated every 64 seconds automatically and the output accuracy according to the datasheet is +-3 degrees. These registers are read only type of registers. Here is the code snippet for the conversion of the temperature readings at the mentioned address.

```
float temperature = buf[0x11] + ((buf[0x12]>>6)*0.25);

printf("The temperature is %f°C\n", temperature);

close(file);

return 0;
```

Here is the screenshot of the output received after the execution of the getTemp.cpp file.

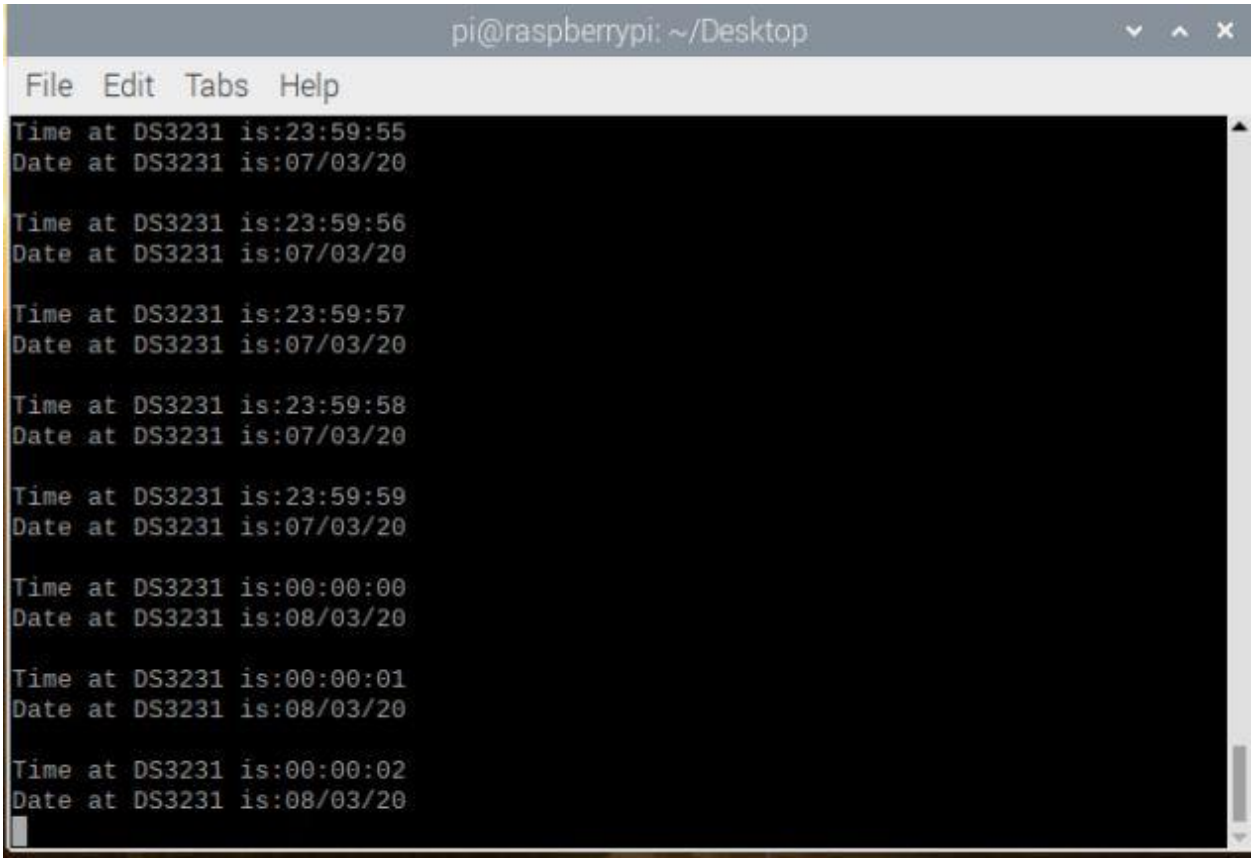


SET THE CURRENT TIME AND DATE ON THE RTC MODULE [15%]

We can explicitly set the time and date of DS3231 using write function of C++. The function "write(file, buf,2);" writes the data stored in buf[1] to the buf address[1]. To do this I made functions by just keeping Set in front for the readability of the code. For example I made SetSeconds() a function to set the seconds. In int main() I created object as Obj and using the Obj I called all the functions and passed the arguments to set the time and date explicitly. In this code I gave time as 23:59:00 and date as 20/03/07. The best thing here I observed was the date is getting changed automatically to 20/03/08.

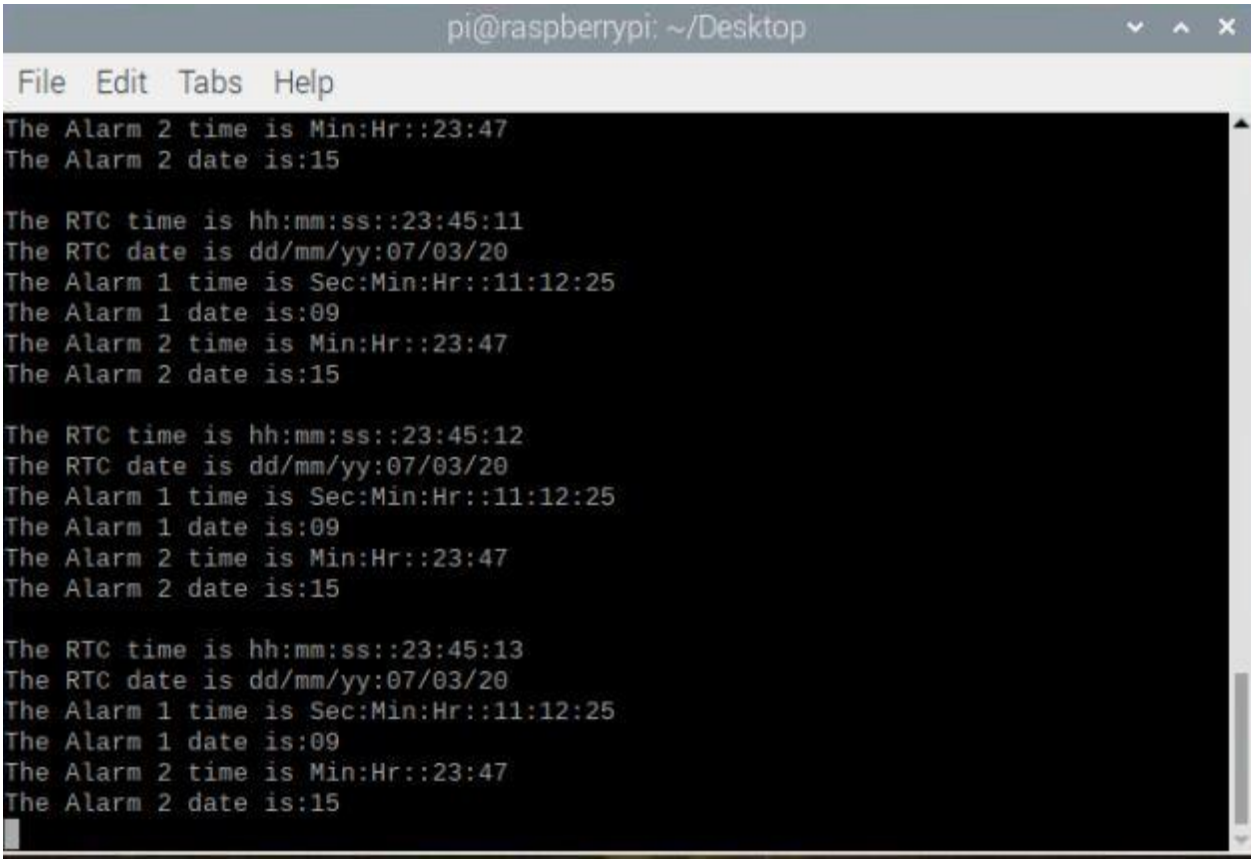
```
int main() {

        RTCDS3231 Obj;
        Obj.DetectRTC();
        Obj.rtcon(Seconds,0x00);
        Obj.SetMinutes(Minutes,0x59);
        Obj.SetHours(Hours,0x23);
        Obj.SetDate(Date,0x07);
        Obj.SetMonth(Month,0x03);
        Obj.SetYear(Year,0x20);

        //Using while loop to print the time continuously.
        while(1){
        Obj.ReadRegisters();
        Obj.Displaytime();
        sleep(1);
        printf("\r");
        }
}
```

Here is the screenshot of the output which I got after running the file set_date_time.cpp.



SET AND READ THE TWO ALARMS (PLEASE NOTE THAT THERE ARE DIFFERENT TYPES OF ALARMS, E.G., TIME OF DAY, TIME ON DATE ETC.) SET AN RTC INTERRUPT SIGNAL DUE TO AN ALARM CONDITION AND EVALUATE THAT IT WORKS CORRECTLY USING PHYSICAL WIRING [30%]

For RTC time, format is hh:min:ss using registers 00h, 01h and 02h. For AM/PM format, I have used bit 5 of register 02h. Alarm 1 rate is set using A1M1, A1M2, A1M3, A1M4 (MSBs of 07h, 08h, 09h, 0Ah registers). Alarm 2 rate is set using A2M2, A2M3, A2M4 (MSBs of registers 0Bh, 0Ch, 0Dh). Alarm interrupts are enabled using register 0Eh (control) A1IE and A2IE bits. Alarm flags are used to confirm alarm trigger using INTCN of control register and A1F, A2F of status (0Fh) register. For example, Alarm 1 rate is set to 1110 which means it wil trigger alarm 1 whenever alarm register seconds match with timer seconds and alarm 2 rate is set to 110 which means alarm 2 will trigger whenever alarm 2 minutes register will match timer minutes.



WRAP THE SQUARE-WAVE GENERATION FUNCTIONALITY OF THE RTC MODULE [5%]

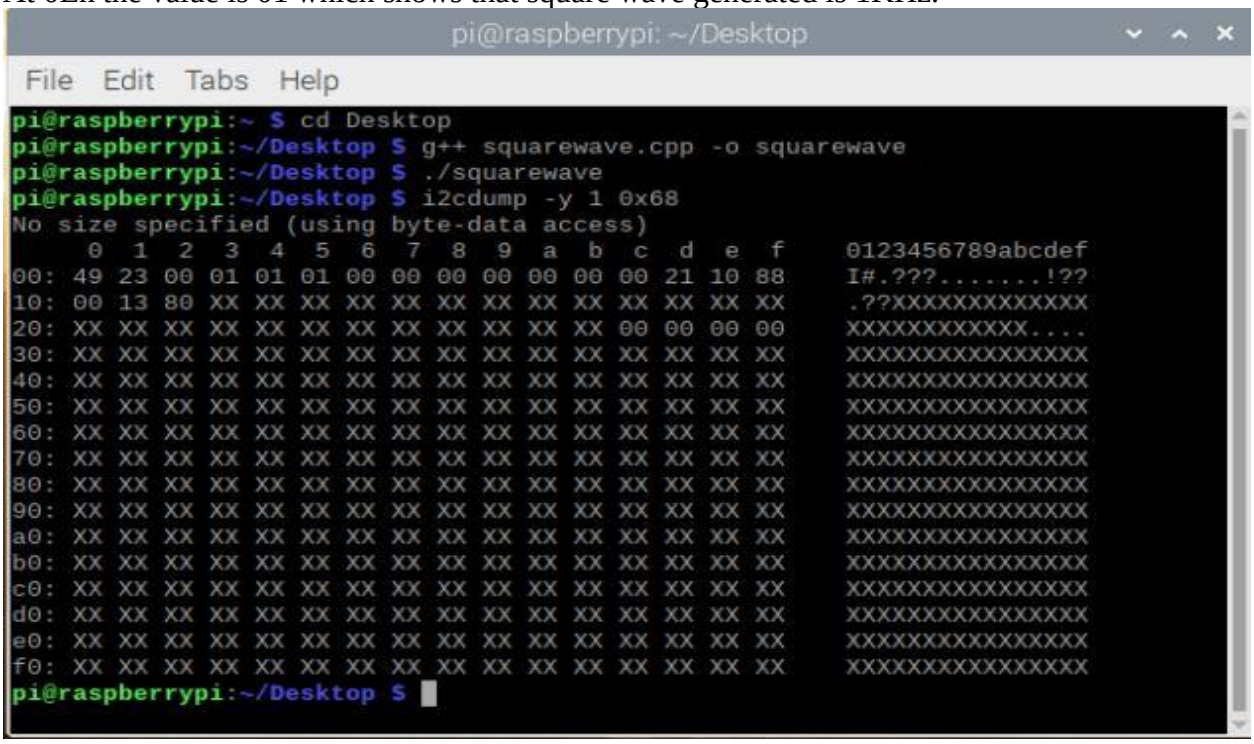
RS1 and RS2 bits control the frequency of the square-wave output when the square wave has been enabled. The following table was taken from the datasheet of the DS3231. Using this table, we can put the desired logic to generate the square wave.

SQUARE-WAVE OUTPUT FREQUENCY

RS2	RS1	SQUARE-WAVE OUTPUT FREQUENCY
0	0	1Hz
0	1	1.024kHz
1	0	4.096kHz
1	1	8.192kHz

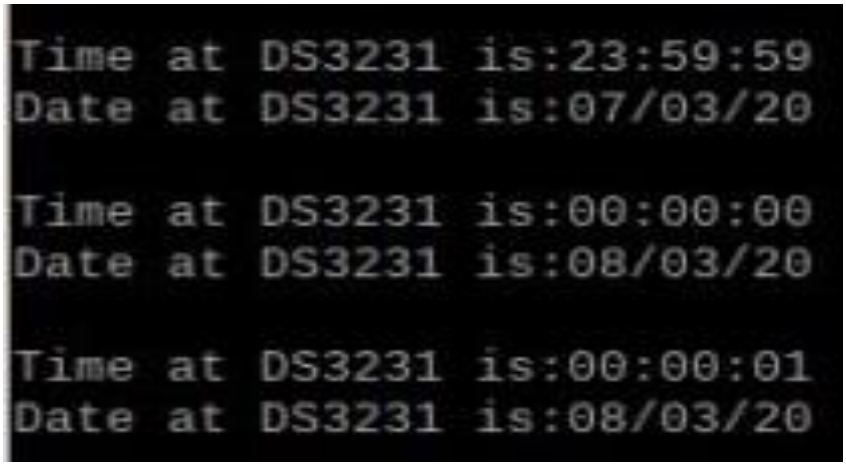
I have enabled the square wave by setting the bit SQWE in bit of address 0Eh to high. To confirm if the square wave is generating properly the pin SQW of RTC-DS3231 can be connected to oscilloscope. Here I am generating 1KHz square wave by keeping RS1 and RS2 as 0.The value of i2cdump at 0Eh has the value '01' which translates to '00010000' which means the bit SQWE is 1 which gives the SQWE pin a 1Khz square wave.

At 0Eh the value is 01 which shows that square wave generated is 1KHz.



ADD NOVEL FUNCTIONALITY OF YOUR OWN DESIGN TO YOUR CLASS [10%]

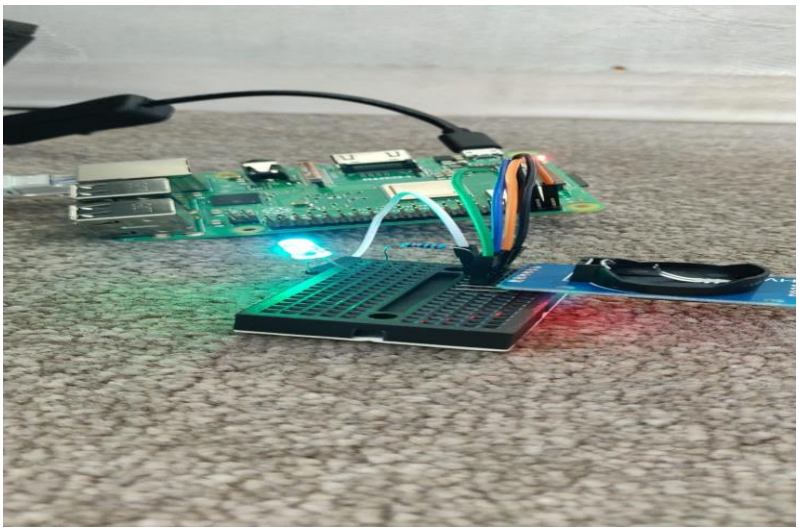
1. First different functionality is the 24 hour format I used which was to set date and time and the thing I observed was the date gets changes automatically after the 24th hour.



2. I used the cout with set width and set fill to print the outputs at places.

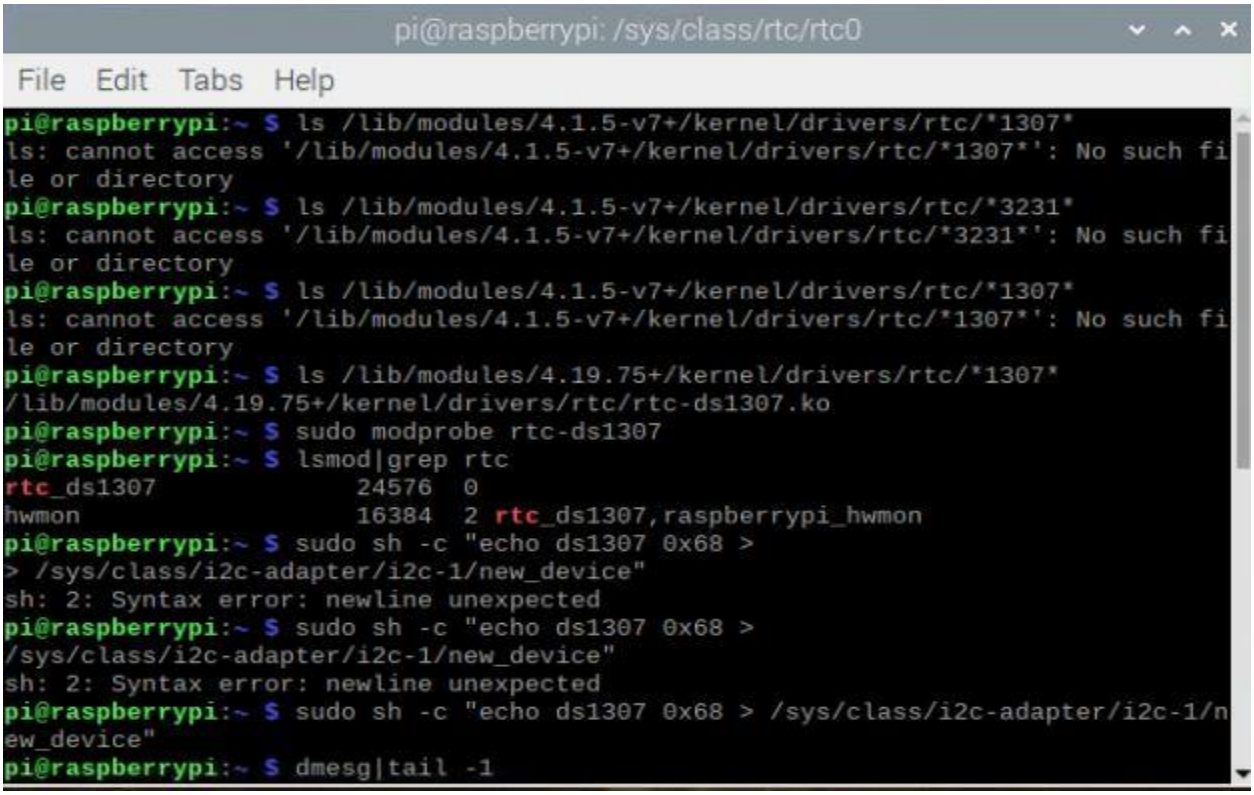
```
cout << "Time at DS3231 is:" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[2]) <<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[1])<<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[0])<<endl;
```

3. I was trying to Use LED blink function with the alarm not working properly otherwise alarm with LED would have been a good new feature in this. The function I was trying to make.



USE A PRE-WRITTEN STANDARD LINUX RTC LKM FOR I NTEGRATING THE RTC (BUT NOT USING YOUR C++ CODE) WITH THE LINUX OS. (SEE THE SUPPORTING MATERIALS BELOW.) [10%]

The pre provided Linux code was executed in the terminal to integrate the RTC-DS3231. Here the version was different so I changed it to the kernel version I have as you can see the below image has can't access executed three times and then I specified the right kernel and I was able to communicate.



I entered $ sudo modprobe rtc-ds1307 to load the RTC.
lsmod grep to show up RTC.
`dmesg|tail -1` to Instantiate the RTC.
See the screenshot below i2cdetect now shows UU for the RTC address instead of 68 which means the address is in use by the driver.

```
pi@raspberrypi:/sys/class/rtc/rtc0 $ i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- 57 -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- UU -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:/sys/class/rtc/rtc0 $
```

The hwclock utility can be used to read (-r) time from or write (-w) time to the RTC device. It can also use the RTC to set (-s) the system clock.

```
pi@raspberrypi: /sys/class/rtc/rtc0
File  Edit  Tabs  Help
sh: 2: Syntax error: newline unexpected
pi@raspberrypi:~ $ sudo sh -c "echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/n
ew_device"
pi@raspberrypi:~ $ dmesg|tail -1
[ 6599.657773] i2c i2c-1: new_device: Instantiated device ds1307 at 0x68
pi@raspberrypi:~ $ ls -l /dev/rtc*
lrwxrwxrwx 1 root root      4 Mar  7 11:41 /dev/rtc -> rtc0
crw------- 1 root root 253, 0 Mar  7 11:41 /dev/rtc0
pi@raspberrypi:~ $ cd /sys/class/rtc/rtc0/
pi@raspberrypi:/sys/class/rtc/rtc0 $ ls
date  device   max_user_freq  power        subsystem  uevent
dev   hctosys  name           since_epoch  time
pi@raspberrypi:/sys/class/rtc/rtc0 $ cat time
01:52:08
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo sh -c "echo 0x68 > delete_device"
sh: 1: cannot create delete_device: Permission denied
pi@raspberrypi:/sys/class/rtc/rtc0 $ date
Sat 07 Mar 2020 11:44:36 AM GMT
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -r
2020-03-07 11:45:09.315496+00:00
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -w
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -r
2020-03-07 11:45:36.899982+00:00
pi@raspberrypi:/sys/class/rtc/rtc0 $
```

```
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -r
2020-03-07 11:45:09.315496+00:00
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -w
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -r
2020-03-07 11:45:36.899982+00:00
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock --set --date="2000-01-01 00:00
:00"
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -r
2000-01-01 00:00:13.360657+00:00
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -s
pi@raspberrypi:/sys/class/rtc/rtc0 $ date
Sat 01 Jan 2000 12:00:59 AM GMT
pi@raspberrypi:/sys/class/rtc/rtc0 $
```

COMMIT HISTORY: I was working with the file RTCDS3231.cpp file throughout the assignment but when I could not write code for setting up alarm, I changed to different .cpp files. Please bare me for doing that however I didn't delete the original file I was working with. Individually executed the files and added the screengrabs in the report.

09 Mar, 2020 8 commits

finally done!!!! AKSHAT SINGH authored just now	d3768fcf
Update RTCDS3231.cpp AKSHAT SINGH authored 9 hours ago	2216bf80
Update RTCDS3231.cpp AKSHAT SINGH authored 9 hours ago	2092771d
Update squarewave.cpp AKSHAT SINGH authored 9 hours ago	9d4ac154
Update set_date_time.cpp AKSHAT SINGH authored 9 hours ago	c9439614
Update getTemp.cpp AKSHAT SINGH authored 9 hours ago	2f49858d
Update current_date_time.cpp AKSHAT SINGH authored 9 hours ago	6e12b65f
Read temperature of RTC-DS3231 AKSHAT SINGH authored 12 hours ago	0ba7a019

08 Mar, 2020 7 commits

Add new file AKSHAT SINGH authored 1 day ago	55798a29
Update set_date_time.cpp AKSHAT SINGH authored 1 day ago	29173e11
Update current_date_time.cpp AKSHAT SINGH authored 1 day ago	993a3d1d
Set new time and date AKSHAT SINGH authored 1 day ago	af2659fa
Delete Current_date_time AKSHAT SINGH authored 1 day ago	9676d3a7
Read current date and time AKSHAT SINGH authored 1 day ago	01643a5a
Read current date and time AKSHAT SINGH authored 1 day ago	a7a1efad

07 Mar, 2020 1 commit

Update RTCDS3231.cpp AKSHAT SINGH authored 2 days ago	9a9395d5

06 Mar, 2020 2 commits

Update RTCDS3231.cpp AKSHAT SINGH authored 3 days ago	0e2f1c7f
Update RTCDS3231.cpp AKSHAT SINGH authored 3 days ago	84e47959

04 Mar, 2020 2 commits

Update RTCDS3231.cpp AKSHAT SINGH authored 5 days ago	1008315e
Update RTCDS3231.cpp AKSHAT SINGH authored 5 days ago	23deec9f

03 Mar, 2020 4 commits

Update RTCDS3231.cpp AKSHAT SINGH authored 5 days ago	decd0b17
Add new file AKSHAT SINGH authored 6 days ago	0bd58eff
Update DS3231.c AKSHAT SINGH authored 6 days ago	c8039837
Add new file AKSHAT SINGH authored 6 days ago	fb4b1c76

17 Feb, 2020 1 commit

Initial commit AKSHAT SINGH authored 3 weeks ago	37aa5d1c

SOURCE CODE:

current_date_time.cpp

```
//Include all the necessary libraries.
#include<stdio.h>
```

```
#include<unistd.h>
#include<math.h>
#include<iostream>
#include<fcntl.h>
#include<sys/ioctl.h>
#include<linux/i2c.h>
#include<linux/i2c-dev.h>
#include <fstream>
#include<string.h>
#include<iomanip>
using namespace std;

//Using address of the clock we want to read time and date
#define BUFFER_SIZE 19
#define RTCaddress      0x68
#define Seconds          0x00
#define Minutes          0x01
#define Hours            0x02
#define Date             0x04
#define Month          0x05
#define Year           0x06

//Creating Base class RTCDS3231
class RTCDS3231{
  private:
        int file;
        char buf[2];                      //Carries the data and address to write
        char buffer[BUFFER_SIZE];  //Carries the data stored in RTCDS3231

//Declaration of all functions used
  public:
       RTCDS3231(){};
       int bcdToDec(unsigned char  b);
       void DetectRTC();
       int Displaytime();
       int ReadRegisters();
       int rtcon(unsigned int address1, unsigned char value1);
       ~RTCDS3231(){};
     };
//Convert BCD to decimal
int RTCDS3231::bcdToDec(unsigned char  b) {
         int dec=(b/16)*10 + (b%16);
         return dec;
}
//Open RTCDS3231
void RTCDS3231::DetectRTC() {
```

```
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	char writeBuffer[1] = {0x00};
	if(write(file, writeBuffer, 1)!=1){
	perror("Failed to reset the read address\n");
	}
	close(file);
}
//Switch on the oscillator by setting CH bit to 0 which can be used to set Seconds
int RTCDS3231::rtcon(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Read the Registers
int RTCDS3231::ReadRegisters() {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	char writeBuffer[1] = {0x00};
	if(write(file, writeBuffer, 1)!=1){
	perror("Failed to reset the read address\n");
	}
	read(file, buffer, BUFFER_SIZE);
	close(file);
	return 0;
}
//Display Time and Date of RTCDS3231 and Alarm
int RTCDS3231::Displaytime() {
	int sec=this->bcdToDec(buffer[0]);
	int min=this->bcdToDec(buffer[1]);
	int hou=this->bcdToDec(buffer[2]);
	int date=this->bcdToDec(buffer[4]);
	int month=this->bcdToDec(buffer[5]);
	int year=this->bcdToDec(buffer[6]);

	cout<<endl;
	cout << "Time at DS3231 is:" <<setw(2)<<setfill('0')<<this->bcdToDec(buffer[2])
<<":"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[1])<<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[0])<<endl;
	cout << "Date at DS3231 is:" <<setw(2)<<setfill('0')<<this->bcdToDec(buffer[4])
<<"/"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[5])<<"/"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[6])<<endl;
```

```
	return 0;
}
//Call functions in order and pass the respective arguments
int main() {

	RTCDS3231 Obj;
	Obj.rtcon(Seconds,0x00);
	//Using while loop to print the time continuously.
	while(1){
	Obj.ReadRegisters();
	Obj.Displaytime();
	sleep(1);
	printf("\r");
	}
}
```

getTemp.cpp

```
//Include all the necessary libraries.
#include<stdio.h>
#include<unistd.h>
#include<math.h>
#include<iostream>
#include<fcntl.h>
#include<sys/ioctl.h>
#include<linux/i2c.h>
#include<linux/i2c-dev.h>
#include <fstream>
#include<string.h>
#include<iomanip>
using namespace std;

//Using address of the clock we want to read time and date
#define BUFFER_SIZE 19
#define RTCaddress  0x68
//Convert BCD to decimal
// the time is in the registers in encoded decimal form
int bcdToDec(char b) { return (b/16)*10 + (b%16); }
int main(){
int file;
if((file=open("/dev/i2c-1", O_RDWR)) < 0){
perror("failed to open the bus\n");
return 1;
}
if(ioctl(file, I2C_SLAVE, 0x68) < 0){
perror("Failed to connect to the sensor\n");
```

```
return 1;
}
char writeBuffer[1] = {0x00};
if(write(file, writeBuffer, 1)!=1){
perror("Failed to reset the read address\n");
return 1;
}
char buf[BUFFER_SIZE];
if(read(file, buf, BUFFER_SIZE)!=BUFFER_SIZE){
perror("Failed to read in the buffer\n");
return 1;
}
// note that 0x11 = 17 decimal and 0x12 = 18 decimal
float temperature = buf[0x11] + ((buf[0x12]>>6)*0.25);
printf("The temperature is %f°C\n", temperature);
close(file);
return 0;
}
```

set_date_time.cpp

```
//Include all the necessary libraries.
#include<stdio.h>
#include<unistd.h>
#include<math.h>
#include<iostream>
#include<fcntl.h>
#include<sys/ioctl.h>
#include<linux/i2c.h>
#include<linux/i2c-dev.h>
#include <fstream>
#include<string.h>
#include<iomanip>

using namespace std;
//Using address of the clock we want to read time and date
#define BUFFER_SIZE 19
#define RTCaddress      0x68
#define Seconds         0x00
#define Minutes         0x01
#define Hours           0x02
#define Date            0x04
#define Month         0x05
#define Year          0x06

//Creating Base class RTCDS3231
```

```
class RTCDS3231{
  private:
          int file;
          char buf[2];                        //Carries the data and address to write
          char buffer[BUFFER_SIZE];  //Carries the data stored in RTCDS3231

//Declaration of all functions used
  public:
        RTCDS3231(){};
        int bcdToDec(unsigned char  b);
        void DetectRTC();
        int Displaytime();
        int ReadRegisters();
        int rtcon(unsigned int address1, unsigned char value1);
        int SetMinutes(unsigned int address1, unsigned char value1);
        int SetHours(unsigned int address1, unsigned char value1);
        int SetDay(unsigned int address1, unsigned char value1);
        int SetDate(unsigned int address1, unsigned char value1);
        int SetMonth(unsigned int address1, unsigned char value1);
        int SetYear(unsigned int address1, unsigned char value1);
        ~RTCDS3231(){};
      };
//Convert BCD to decimal
int RTCDS3231::bcdToDec(unsigned char  b) {
           int dec=(b/16)*10 + (b%16);
           return dec;
}
//Open RTCDS3231
void RTCDS3231::DetectRTC() {
           file=open("/dev/i2c-1", O_RDWR);
           ioctl(file, I2C_SLAVE, 0x68);
           char writeBuffer[1] = {0x00};
           if(write(file, writeBuffer, 1)!=1){
           perror("Failed to reset the read address\n");
           }
           close(file);
}

//Switch on the oscillator by setting CH bit to 0 which can be used to set Seconds
int RTCDS3231::rtcon(unsigned int address1, unsigned char value1) {
           file=open("/dev/i2c-1", O_RDWR);
           ioctl(file, I2C_SLAVE, 0x68);
           buf[0]= address1;
           buf[1]= value1;
           write(file, buf,2 );
           close(file);
```

```
	return 0;
}

//Set Minutes
int RTCDS3231::SetMinutes(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}

//Set Hours
int RTCDS3231::SetHours(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}

//Set Date
int RTCDS3231::SetDate(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Set Month
int RTCDS3231::SetMonth(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Set Year
```

```
int RTCDS3231::SetYear(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Read the Registers
int RTCDS3231::ReadRegisters() {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	char writeBuffer[1] = {0x00};
	if(write(file, writeBuffer, 1)!=1){
	perror("Failed to reset the read address\n");
	}
	read(file, buffer, BUFFER_SIZE);
	close(file);
	return 0;
}
//Display Time and Date of RTCDS3231 and Alarm
int RTCDS3231::Displaytime() {
	int sec=this->bcdToDec(buffer[0]);
	int min=this->bcdToDec(buffer[1]);
	int hou=this->bcdToDec(buffer[2]);
	int date=this->bcdToDec(buffer[4]);
	int month=this->bcdToDec(buffer[5]);
	int year=this->bcdToDec(buffer[6]);

	cout<<endl;
	cout << "Time at DS3231 is:" <<setw(2)<<setfill('0')<<this->bcdToDec(buffer[2])
<<":"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[1])<<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[0])<<endl;
	cout << "Date at DS3231 is:" <<setw(2)<<setfill('0')<<this->bcdToDec(buffer[4])
<<"/"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[5])<<"/"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[6])<<endl;
	return 0;
}

//Call functions in order and pass the respective arguments
int main() {

	RTCDS3231 Obj;
	Obj.DetectRTC();
	Obj.rtcon(Seconds,0x00);
```

```
        Obj.SetMinutes(Minutes,0x59);
        Obj.SetHours(Hours,0x23);
        Obj.SetDate(Date,0x07);
        Obj.SetMonth(Month,0x03);
        Obj.SetYear(Year,0x20);

        //Using while loop to print the time continuously.
        while(1){
        Obj.ReadRegisters();
        Obj.Displaytime();
        sleep(1);
        printf("\r");
        }
}
```

Alarm.cpp

```
//Include all the necessary libraries.
#include<stdio.h>
#include<unistd.h>
#include<math.h>
#include<iostream>
#include<fcntl.h>
#include<sys/ioctl.h>
#include<linux/i2c.h>
#include<linux/i2c-dev.h>
#include <fstream>
#include<string.h>
#include<iomanip>
extern"C"{
        #include <wiringPi.h>
        }


using namespace std;


//Using address of the clock we want to read time and date
#define BUFFER_SIZE 19
#define RTCaddress      0x68
#define Seconds         0x00
#define Minutes         0x01
#define Hours           0x02
#define Date            0x04
#define Month       0x05
#define Year        0x06
```

```
#define AlarmSeconds          0x07
#define AlarmMinutes          0x08
#define AlarmHours   0x09
#define AlarmDate        0x0A
#define AlarmMinutes2        0x0B
#define AlarmHours2          0x0C
#define AlarmDate2   0x0D
#define Control              0x0E
#define Status               0x0F

//Creating Base class RTCDS3231
class RTCDS3231{
  private:
        int file;
        char buf[2];                      //Carries the data and address to write
        char buffer[BUFFER_SIZE];  //Carries the data stored in RTCDS3231


  //Declaration of all functions used
  public:
       RTCDS3231(){};
       int bcdToDec(unsigned char  b);
       void DetectRTC();
       int Displaytime();
       int ReadRegisters();
       int AlarmCondition();
       int rtcon(unsigned int address1, unsigned char value1);
       int SetMinutes(unsigned int address1, unsigned char value1);
       int SetHours(unsigned int address1, unsigned char value1);
       int SetDay(unsigned int address1, unsigned char value1);
       int SetDate(unsigned int address1, unsigned char value1);
       int SetMonth(unsigned int address1, unsigned char value1);
       int SetYear(unsigned int address1, unsigned char value1);
       int SetAlarmSeconds(unsigned int address1, unsigned char value1);
       int SetAlarmMinutes(unsigned int address1, unsigned char value1);
       int SetAlarmHours(unsigned int address1, unsigned char value1);
       int SetAlarmDate(unsigned int address1, unsigned char value1);
       int SetAlarmMinutes2(unsigned int address1, unsigned char value1);
       int SetAlarmHours2(unsigned int address1, unsigned char value1);
       int SetAlarmDate2(unsigned int address1, unsigned char value1);

       ~RTCDS3231(){};
     };
//Convert BCD to decimal
int RTCDS3231::bcdToDec(unsigned char  b) {
          int dec=(b/16)*10 + (b%16);
```

```
	return dec;
}

//Open RTCDS3231
void RTCDS3231::DetectRTC() {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	char writeBuffer[1] = {0x00};
	if(write(file, writeBuffer, 1)!=1){
	perror("Failed to reset the read address\n");
	}
	close(file);
}

//Switch on the oscillator by setting CH bit to 0 which can be used to set Seconds
int RTCDS3231::rtcon(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}

//Set Minutes
int RTCDS3231::SetMinutes(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}

//Set Hours
int RTCDS3231::SetHours(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
```

```
//Set Date
int RTCDS3231::SetDate(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Set Month
int RTCDS3231::SetMonth(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Set Year
int RTCDS3231::SetYear(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Read the Registers
int RTCDS3231::ReadRegisters() {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	char writeBuffer[1] = {0x00};
	if(write(file, writeBuffer, 1)!=1){
	perror("Failed to reset the read address\n");
	}
	read(file, buffer, BUFFER_SIZE);
	close(file);
	return 0;
}
//set the alarm 1 seconds
int RTCDS3231::SetAlarmSeconds(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
```

```
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//set the alarm 1 minutes
int RTCDS3231::SetAlarmMinutes(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//set the alarm 1 hours
int RTCDS3231::SetAlarmHours(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//set the alarm 1 date
int RTCDS3231::SetAlarmDate(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//set the alarm 2 minutes
int RTCDS3231::SetAlarmMinutes2(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
```

```
}
//set the alarm 2 hours
int RTCDS3231::SetAlarmHours2(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//set the alarm 2 date
int RTCDS3231::SetAlarmDate2(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}

//Display Time and Date of RTCDS3231 and Alarm
int RTCDS3231::Displaytime() {
	int sec=this->bcdToDec(buffer[0]);
	int min=this->bcdToDec(buffer[1]);
	int hou=this->bcdToDec(buffer[2]);
	int date=this->bcdToDec(buffer[4]);
	int month=this->bcdToDec(buffer[5]);
	int year=this->bcdToDec(buffer[6]);

	int alsec=this->bcdToDec(buffer[7]);
	int almin=this->bcdToDec(buffer[8]);
	int alhou=this->bcdToDec(buffer[9]);
	int aldate=this->bcdToDec(buffer[10]);

	int almin2=this->bcdToDec(buffer[11]);
	int alhou2=this->bcdToDec(buffer[12]);
	int aldate2=this->bcdToDec(buffer[13]);

	int ctrl=this->bcdToDec(buffer[14]);
	int status=this->bcdToDec(buffer[15]);
	cout<<endl;
	cout << "The RTC time is hh:mm:ss::" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[2]) <<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[1])<<":"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[0])<<endl;
```

```
	cout << "The RTC date is dd/mm/yy:" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[4]) <<"/"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[5]&0x7F)<<"/"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[6])<<endl;
	cout << "The Alarm 1 time is Sec:Min:Hr::" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[9]&0x7F) <<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[8]&0x7F)<<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[7]&0x7F)<<endl;
	cout << "The Alarm 1 date is:" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[10]&0x3F)<<endl;
	cout << "The Alarm 2 time is Min:Hr::" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[12]&0x7F)<<":"<<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[11])<<endl;
	cout << "The Alarm 2 date is:" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[13]&0x3F)<<endl;
	//cout << "The Control and status are:" <<setw(2)<<setfill('0')<<this-
>bcdToDec(buffer[15]) <<"/"<<setw(2)<<setfill('0')<<this->bcdToDec(buffer[14])<<endl;
	return 0;
}

//Call functions in order and pass the respective arguments
int main() {

	RTCDS3231 Obj;
	Obj.DetectRTC();
	Obj.rtcon(Seconds,0x00);
	Obj.SetMinutes(Minutes,0x45);
	Obj.SetHours(Hours,0x23);
	Obj.SetDate(Date,0x07);
	Obj.SetMonth(Month,0x03);
	Obj.SetYear(Year,0x20);
	Obj.SetAlarmSeconds(AlarmSeconds,0xa5);
	Obj.SetAlarmMinutes(AlarmMinutes,0x92);
	Obj.SetAlarmHours(AlarmHours,0x91);
	Obj.SetAlarmMinutes(Control,0x07); // control

	Obj.SetAlarmDate(AlarmDate,0x09);
	Obj.SetAlarmMinutes2(AlarmMinutes2,0x47);
	Obj.SetAlarmHours2(AlarmHours2,0x23);
	Obj.SetAlarmDate2(AlarmDate2,0x15);

	//Using while loop to print the time continuously.
	while(1){
	Obj.ReadRegisters();
	Obj.Displaytime();
	sleep(1);
	printf("\r");
```

```
        }
}
```

Squarewave.cpp

```
//Include all the necessary libraries.
#include<stdio.h>
#include<unistd.h>
#include<math.h>
#include<iostream>
#include<fcntl.h>
#include<sys/ioctl.h>
#include<linux/i2c.h>
#include<linux/i2c-dev.h>
#include <fstream>
#include<string.h>
#include<iomanip>

using namespace std;
//Using address of the clock we want to read time and date
#define BUFFER_SIZE 19
#define RTCaddress      0x68
#define Control             0x0E
//Creating Base class RTCDS3231
class RTCDS3231{
  private:
        int file;
        char buf[2];                        //Carries the data and address to write
        char buffer[BUFFER_SIZE];  //Carries the data stored in RTCDS3231
 //Declaration of all functions used
  public:
       RTCDS3231(){};
       int bcdToDec(unsigned char  b);
       void DetectRTC();
       int ReadRegisters();
       int SetSquareWave(unsigned int address1, unsigned char value1);

       ~RTCDS3231(){};
     };
//Convert BCD to decimal
int RTCDS3231::bcdToDec(unsigned char  b) {
        int dec=(b/16)*10 + (b%16);
        return dec;
}

//Open RTCDS3231
```

```
void RTCDS3231::DetectRTC() {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	char writeBuffer[1] = {0x00};
	if(write(file, writeBuffer, 1)!=1){
	perror("Failed to reset the read address\n");
	}
	close(file);
}
int RTCDS3231::SetSquareWave(unsigned int address1, unsigned char value1) {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	buf[0]= address1;
	buf[1]= value1;
	write(file, buf,2 );
	close(file);
	return 0;
}
//Read the Registers
int RTCDS3231::ReadRegisters() {
	file=open("/dev/i2c-1", O_RDWR);
	ioctl(file, I2C_SLAVE, 0x68);
	char writeBuffer[1] = {0x00};
	if(write(file, writeBuffer, 1)!=1){
	perror("Failed to reset the read address\n");
	}
	read(file, buffer, BUFFER_SIZE);
	close(file);
	return 0;
}
int main() {

	RTCDS3231 Obj;
	Obj.DetectRTC();
	Obj.SetSquareWave(Control,0x10);
	Obj.ReadRegisters();
}
```