

Student Name:
Prathosh Tarikere
Satish

Degree Scheme: MEng

Student
Name:19211440

Year: C

Title:Connected
Embedded Svstems

Lecturer: Derek Molly

I hereby declare that the attached submission is all my own work, that it has not previously been submitted for assessment, and that I have not knowingly allowed it to be used by another student. I understand that deceiving or attempting to deceive examiners by passing off the work of another as one's own is not permitted. I also understand that using another's student's work or knowingly allowing another student to use my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings.

Signed: Prathosh T S

Date: 09-03-2020

Note: Coursework examiners are entitled to reject any coursework which does not have a signed copy of this form attached.

For use by examiners only (students should not write below this line)

Comments:

Aims and Objective

Our aims and objective are to interface an embedded system into real-world where DS3231 Real-time clock is used in this assignment to track time, date and other properties. Embedded systems like Raspberry pi will not have a battery-backed clock, so whenever it reboots the clock time is lost so to overcome this we can make use of Realtime battery-backed clock such as RTC DS3231, DS1307. Any of these devices can be connected and controlled using the I2C bus of the Embedded system. In this assignment, DS3231 is chosen as it is easily available and is highly accurate of + or – 63 seconds per year.

Assignment tasks

1. Interfacing the RTC module to Raspberry pi via I2C bus.
2. Testing a connected module using Linux based I2C tool.
3. Reading appropriated registers with C++ code to display properties such as time, date and temperature.
4. Writing into an appropriate register using C++ to set current time date and also two different available alarms.
5. Interrupt enabled alarm and validate it using an LED.
6. Writing and reading appropriate registers for square wave functionality.
7. Adding any novel feature.
8. Integrating RTC using Linux LKM.

Assignment approach

1. The physical connection of the Realtime clock to PI.

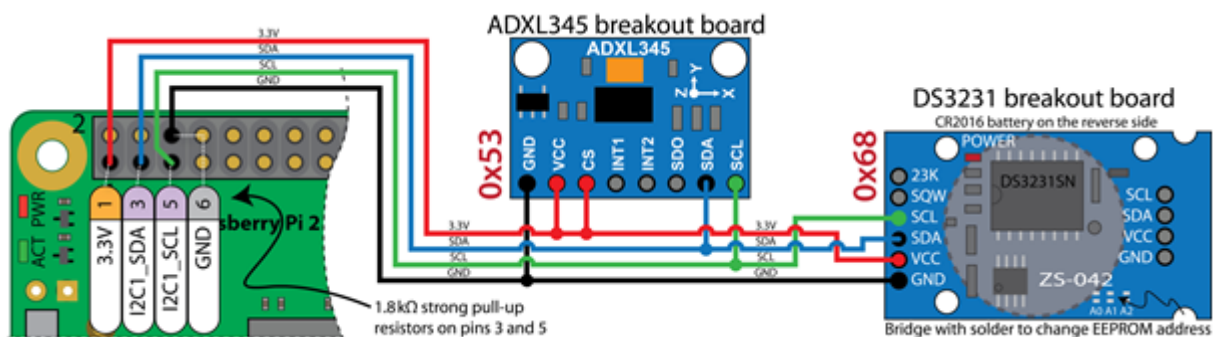
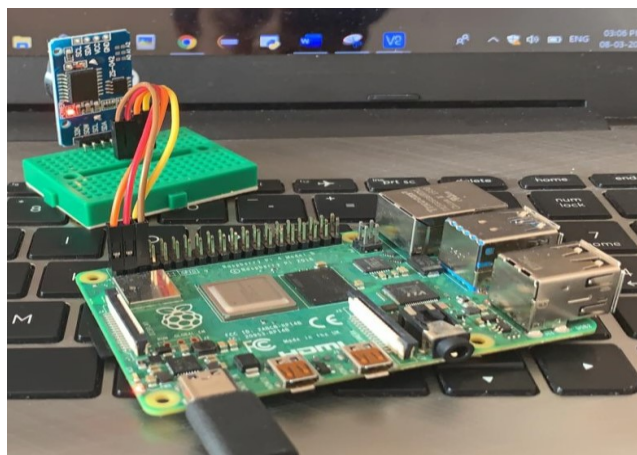


Fig-1

Pin 1 of Raspberry Pi is connected to the Vcc of the module. GPIO 3 is connected to the SDA pin, GPIO 5 is connected to SCL and lastly, GPIO 6 is connected to the GND pin of RTC as shown in above Fig-1



2. Detecting device using Linux I2C tools

After the physical connection, I2C tools are used to verify whether the device is present on the I2C bus or not. Before using the I2C tool, the I2C option must be enabled which can be found in the preferences option of the PI. Command `i2cdetect -y 1` can be used to see the device on the bus, the default address of DS3231 is seen as `0x68`. Tool `i2cdump -y 1 0x68` can be used to check the values of the registers and shown below in Fig 2. `i2cset -y 1 0x68 (register address) (value to be set)` can be used to set values to the specified registers and is shown below in Fig-3. With the help if set tool `i2cset -y 1 0x68 0x01 0x55`, register 01 is set to 55.

```
pi@raspberrypi:~$ i2cdetect -1
Error: Unsupported option "-1"!
Usage: i2cdetect [-y] [-a] [-q|-r] I2CBUS [FIRST LAST]
       i2cdetect -F I2CBUS
       i2cdetect -l
       I2CBUS is an integer or an I2C bus name
       If provided, FIRST and LAST limit the probing range.
pi@raspberrypi:~$ i2cdump -y 1 0x68 b
00:  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f      0123456789abcdef
10: 20 23 00 01 01 01 00 00 81 10 3f 02 20 3d 1c 88      #.???..???? =??
20: 00 19 80 XX XX XX XX XX XX XX XX XX XX XX XX XX      .??XXXXXXXXXXXX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
pi@raspberrypi:~$ i2cdetect -y -r 1
00:  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- 57 -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~$
```

Fig-2

```
pi@raspberrypi: ~/Desktop
File Edit Tabs Help
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
pi@raspberrypi:~/Desktop$ i2cset -y 1 0x68 0x01 0x55
pi@raspberrypi:~/Desktop$ i2cdump -y 1 0x68
No size specified (using byte-data access)
00:  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f      0123456789abcdef
10: 14 55 05 05 21 05 20 15 56 04 21 56 04 45 49 00      ?U??! ?V?IV?EI.
20: 00 18 80 XX XX XX XX XX XX XX XX XX XX XX XX XX XX      .??XXXXXXXXXXXX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX      XXXXXXXXXXXXXXX
pi@raspberrypi:~/Desktop$
```

Fig-3

Tool `i2cget -y 1 0x68 (register number)` is used to see the specific register values. `i2cget -y 1 0x68 0x01` is used to look into register 01 and shown in Fig-4.

```

pi@raspberrypi: ~/Desktop
File Edit Tabs Help
pi@raspberrypi:~/Desktop $ i2cget -y 1 0x68 0x01
0x02
pi@raspberrypi:~/Desktop $ 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: 17 02 00 01 01 01 00 00 81 10 3f 02 20 3d 1c 88    ??..???.?? =??
10: 00 18 80 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 XX XX XX XX    XXXXXXXXXXXXXXX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXX
pi@raspberrypi:~/Desktop $ i2cget -y 1 0x68 0x01
0x02
pi@raspberrypi:~/Desktop $

```

Fig-4

3. Reading time using given C code

Given C program is compiled and run to read the specific seconds, minutes and hours register which are 00, 01 and 02. The reading of time is shown in the below Fig-5.

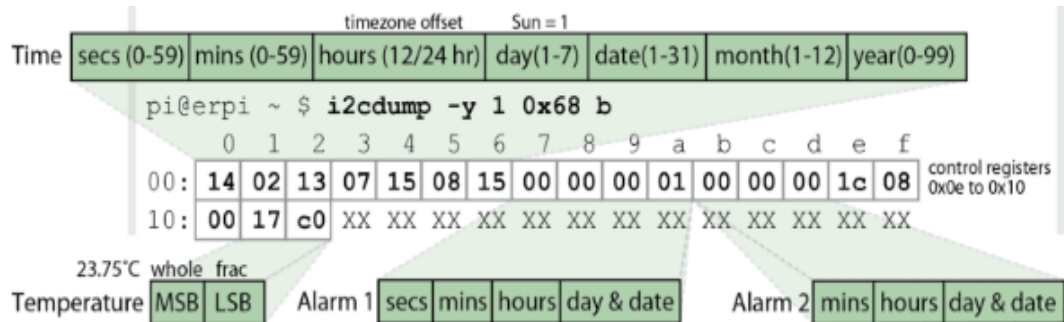
```

pi@raspberrypi: ~/Desktop
File Edit Tabs Help
Closed file
pi@raspberrypi:~/assignment1 $ cd --
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ gcc DS3231.c -o DS3231
DS3231.c: In function 'main':
DS3231.c:22:7: warning: implicit declaration of function 'write'; did you mean 'fwrite'? [-Wimplicit-function-declaration]
    if(write(file, writeBuffer, 1)!=1){
       ^~~~~
    fwrite
DS3231.c:27:7: warning: implicit declaration of function 'read'; did you mean 'fread'? [-Wimplicit-function-declaration]
    if(read(file, buf, BUFFER_SIZE)!=BUFFER_SIZE){
       ^~~~
    fread
DS3231.c:33:4: warning: implicit declaration of function 'close'; did you mean 'pclose'? [-Wimplicit-function-declaration]
    close(file);
    ^~~~~
    pclose
pi@raspberrypi:~/Desktop $ ./DS3231
Starting the DS3231 test application
The RTC time is 05:01:17
pi@raspberrypi:~/Desktop $

```

Fig-5

4. Understanding the role of registers



ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/24	AM/PM 20 Hour	10 Hour	Hour				Hours	1–12 + AM/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/24	AM/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + AM/PM 00–23
0Ah	A1M4	DY/DT	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/24	AM/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + AM/PM 00–23
0Dh	A2M4	DY/DT	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	EOSC	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

- There are a total of 19 registers with different functionality. Each register is of 8 bit in size.
- Clock registers(00h to 02h)
First, register (00h) will have the seconds, (01h) will have minutes and (02h) will have hours information, the hours can be written in the form of 12 hours format or 24 hours format. All three together sums up as clock information. Input to bit 6 of this register decides what hour format is selected. When this bit 6 is high, 12-hour mode is selected and in 12-hour modes bit 5 decides AM/PM mode, by making this bit high, PM mode is selected. In 24 hours format, bit 5 is a 20 hour bit.
- Calendar registers(03h to 06h)
Register (03h) will have day information such as Sunday as 1 and so on, (04h) will have the date and so on, (05h) will have month information as of January as 1 and (06h) will have year information. All four together sums up as calendar information.

- Alarm registers(07h to 0Dh)

The two-alarm configuration is available, one is the time of day and the other is the time of the date. Alarm 1 can be set by writing into registers 07h to 0Ah. Alarm 2 can be set by writing into registers 0Bh to 0Dh. In time-of-day/date alarm registers bit 7 is mask bits and their configuration ends up with different alarm properties and is shown in below Figure-6

DY/DT	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match

DY/DT	ALARM 2 REGISTER MASK BITS (BIT 7)			ALARM RATE
	A2M4	A2M3	A2M2	
X	1	1	1	Alarm once per minute (00 seconds of every minute)
X	1	1	0	Alarm when minutes match
X	1	0	0	Alarm when hours and minutes match
0	0	0	0	Alarm when date, hours, and minutes match
1	0	0	0	Alarm when day, hours, and minutes match

Fig-6

The value in the bit 6 of alarm date/day decides whether the value stored in bit 0 to bit 5 of the register represents days of the week or date of the month if logic 1 is written to DY/DT bit then alarm is triggered when it is matched with the day if it is logic 0 then it will be triggered when it is matched with the date. The alarms can be programmed to activate INT/SQW when the alarm is triggered. A1IE(bit 0), A2IE(bit 1) and INTCN (bit 2) of the control register should be made high before writing into alarm registers. When RTC register values match alarm register settings, the alarm flag A1F or A2F set to high correspondingly, once-per second the match is tested between alarm and time date register.

- Control register

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	EOSC	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR:	0	0	0	1	1	1	0	0

- EOSC (Enable oscillator)

Writing to this bit can control the activity of an oscillator. When Vcc applied always the oscillator will be high regardless of the value in this register. The oscillator will be stopped when bit 7 is made high and Vcc changes to Vbat. When it is made low and Vcc changes to Vbat the oscillators will turn on. This bit helps us to control the oscillator when DS3231 is working on the battery.

- BBSQW (Battery Backed Square Wave Enable)

This bit will be high when Vcc applied regardless of the data. This bit controls the generation of the square wave when DS3231 is working on battery. When this bit is high with bit 2 (INTCN) low, a square wave can be generated. When this bit is low, INT/SQW pin goes to a high impedance state.

- CONV (Convert Temperature)

When this bit is made high, the temperature is converted to digital code and the TCXO algorithm updates the capacitance array to the oscillator. But before making this bit high it is recommended to check BSY bit.

- Rate select bit (RS2 and RS1)

This rate select bit helps us to select the frequency of the square wave. When Vcc is first applied, these two bits will be set to logic 1. The combination of RS2 and RS1 is shown below for different frequency square wave output.

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

- Interrupt control (INTCN)

When this bit low square wave output can be seen at INT/SQW pin of the DS3231 if this bit logic 1 the match between alarm registers and clock with a calendar can be achieved.

- A2IE & A1IE (Alarm Interrupt Enable)

A2IE and A1IE bits should be high along with INTCN = 1 so that the alarm flag can initiate interrupt signals across the INT/SQW pin of DS3231.

5. Writing a C++ class to achieve the assignment task.

- Checking the bus and initiating communication with the slave

```
Realtime :: Realtime (string path, char addr){
    this->path = path;
    this->addr = addr;
}

Realtime::~~Realtime(){
    close(bus);
    cout<< "Closed file"<< endl;
}

void Realtime :: check (){
    if ((bus = open(path.c_str(), O_RDWR)) < 0 ){
        cout << "Failed to open the bus" << endl;
    }
}

void Realtime :: inicom (){
    if(ioctl(bus, I2C_SLAVE, addr) < 0){
        cout << "Failed to connect to the sensor" << endl;
    }
}
```

The path and address of the slave are given through the constructor. In the destructor file is closing. The path gave through the constructor is used in check() function to verify the path and the address is used in inicom() function where communication initiates.

- Writing and Reading function

write1() is the writing function used to write data into the specified register, where number 2 in the if statement represents the number of bytes to be written. The first byte is an address and the second byte is converted from decimal to BCD before writing as clock and calendar registers are in BCD. Buffer() is the reading function used to read data from the registers. Number 19 in the if statement represents num of bytes to be read. A function called w() is used to write direct Hexa values to the control registers.

```

int Realtime :: write1(char a, char b){
    char ip[2];
    ip[0] = a;
    ip[1] = dec2bcd(b);

    if(write(bus, ip, 2)!=2)
    {
        cout << "Writing problem" << endl;
        return 1;
    }
    return 0;
}

void Realtime :: buffer(){
    if(read(bus, reg, 19)!=19){
        cout << "Failed to read in the buffer" << endl;
    }
}

```

- Reading time and date before writing into registers

```

int main() {
    Realtime a = Realtime("/dev/i2c-1", 0x68) ;
    a.check();
    a.inicom();
    a.rst();
    a.buffer();

    cout << "Before setting Time :" << endl;
    cout << a.bcdtodec(reg[0x02]) << ":" << a.bcdtodec(reg[0x01]) << ":" << a.bcdtodec(reg[0x00]) << endl;
    cout << "Before setting Date :" << endl;
    cout << a.bcdtodec(reg[0x03]) << ":" << a.bcdtodec(reg[0x04]) << ":" << a.bcdtodec(reg[0x05]) << ":" << a.bcdtodec(re

```

Reading clock and calendar register before writing into those register.

- Reading Temperature

a.temp() is the function where temperature is reading.

```

void Realtime:: temp(){
    float temperature = reg[0x11] + ((reg[0x12]>>6)*0.25);
    cout << "Temperature :" << temperature << endl;
}

```

- Writing time and date

```

int main() {
    Realtime a = Realtime("/dev/i2c-1", 0x68) ;
    a.check();
    a.inicom();
    a.rst();
    a.buffer();

    cout << "Before setting Time :" << endl;
    cout << a.bcdtodec(reg[0x02]) << ":" << a.bcdtodec(reg[0x01]) << ":" << a.bcdtodec(reg[0x00]) << endl;
    a.t(50, 55, 4);
    a.d(05, 21, 05, 20);
}

```

a.t(ss, mm, hh) is the function where time is writing into seconds, minutes and hours registers.

a.d(day, date, month, year) is the function where writing into the day, date, month and year registers.


```

void Realtime :: t(int s, int m, int h){
    cout << "writing HH : MM : SS " << endl;
    writel(0x00, s);
    writel(0x01, m);
    writel(0x02, h);
    rst();
    buffer();
    cout << bcdtodec(reg[0x2]) << ":" << bcdtodec(reg[0x1]) << ":" << bcdtodec(reg[0x0]) << endl;
}

void Realtime :: d(int d, int da, int m, int y){
    cout << "writing Day : Date : Month : Year " << endl;
    writel(0x03, d);
    writel(0x04, da);
    writel(0x05, m);
    writel(0x06, y);
    day();
    rst();
    buffer();
    cout << bcdtodec(reg[0x4]) << ":" << bcdtodec(reg[0x5]) << ":" << bcdtodec(reg[0x6]) << endl;
}

```

The function Day() is used to show that the register values of 0x03 are days of the week.

```

void Realtime:: day(){
if(bcdtodec(reg[0x3]) == 1)
{cout<< "Sunday :"; }
else if(bcdtodec(reg[0x3]) == 2)
{cout<< "Monday :"; }
else if(bcdtodec(reg[0x3]) == 3)
{cout<< "Tuesday :"; }
else if(bcdtodec(reg[0x3]) == 4)
{cout<< "Wednesday :"; }
else if(bcdtodec(reg[0x3]) == 5)
{cout<< "Thursday :"; }
else if(bcdtodec(reg[0x3]) == 6)
{cout<< "Friday :"; }
else if(bcdtodec(reg[0x3]) == 7)
{cout<< "Saturday :"; }
}

```

- Setting two different alarm

```

void Realtime :: a2(int min, int hr, int day){
    cout << "Setting alarm 2 at" << hr << ":" << min << ":" << "day" << endl;
    w(0x0F, 0x00);
    w(0x0E, 0x07);
    writel(0x08, min);
    writel(0x0C, hr);
    writel(0x0D, day);
    if(bcdtodec(reg[15]) == 02 || bcdtodec(reg[15]) == 01 || bcdtodec(reg[15]) == 03) // checking is any alarm flag
    {
        w(0x0E, 0x07);
    }
}

void Realtime :: a1(int s, int m, int h, int da){
    cout << "Setting alarm 1 at" << h << ":" << m << ":" << s << ":" << "date" << da << endl;
    w(0x0F, 0x00);
    w(0x0E, 0x07);
    writel(0x07, s);
    writel(0x08, m);
    writel(0x09, h);
    writel(0x0A, da);
    if(bcdtodec(reg[15]) == 02 || bcdtodec(reg[15]) == 01 || bcdtodec(reg[15]) == 03) // checking is any alarm flag
    {
        w(0x0E, 0x07);
    }
}

```

Method a2() is used to set alarm 2 and a1() is used to set alarm 1. If the INTCN is made high the interrupt is initiated by alarm flag but at the same time if the square wave is triggered both will be collided to each other, so if condition in both the alarm methods are checking the alarm flags to be high then again INTCN pin is made high for to get interrupt signal across INT/SQWpin.

```

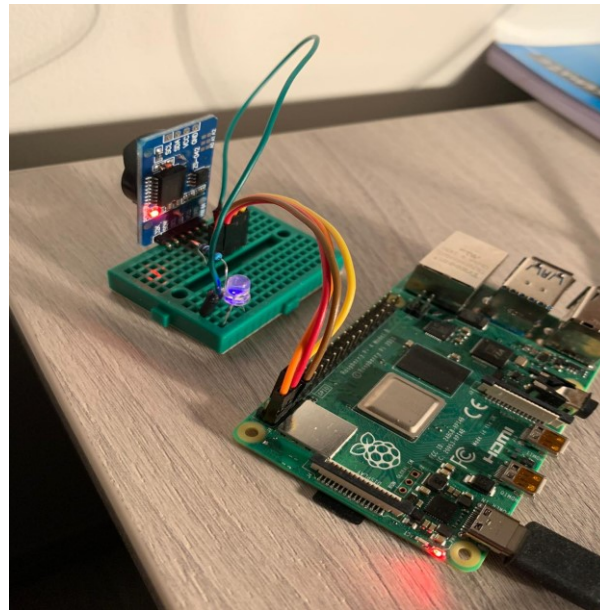
a.a2(56, 04, 45); // only min:hr:day should be given. where 4 in day should not be changed
a.display();

sleep(15);

a.a1(20, 56, 4, 21); // only sec:min:hr:date can be given
a.display();

```

Alarm 2 is set at 56 min, 4 hr and on 5th day that is Thursday. Alarm 1 is set at 20sec, 56 min, 4hr on date 21. LED is connected to INT/SQW pin when alarm time matches with the clock and calendar registers, interrupt is enabled and LED goes off as this pin is pulled up and the link is given to see them working.
<https://drive.google.com/a/mail.dcu.ie/file/d/1TFMYivyioJDPopa1eZ2Chrh4xh5VdkBe/view?usp=drivesdk>



- Square wave functionality

```

void Realtime :: frequency(int a, int b)
{
w(0x0F, 0x00);
w(0x0E, 0x04);

if(a == 1){
    cout << "generating 1Hz for " << b << "sec" << endl;
w(0x0E, 0x00);sleep(b);}
else if(a == 2){
    cout << "generating 1KHz" << b << "sec" << endl;
w(0x0E, 0x09);
sleep(b);
}
else if(a == 3){
    cout << "generating 4KHz" << b << "sec"<< endl;
w(0x0E, 0x10);
sleep(b);
}
else if(a ==4){
    cout << "generating 8KHz" << b << "sec" << endl;
w(0x0E, 0x19);
sleep(b);
}

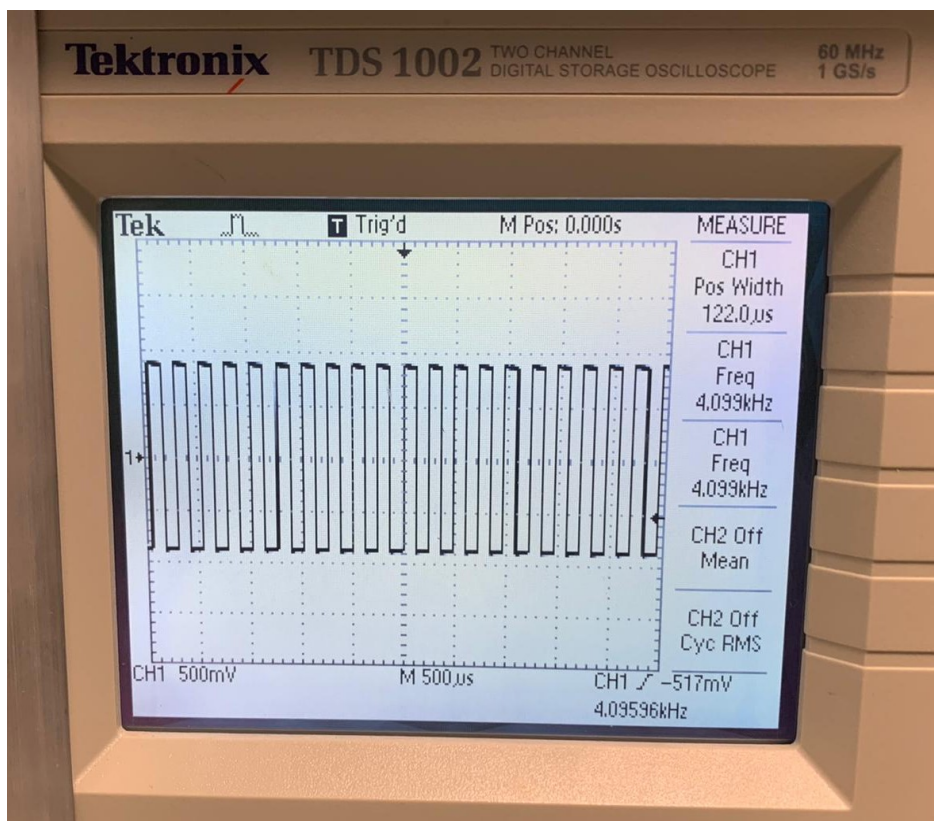
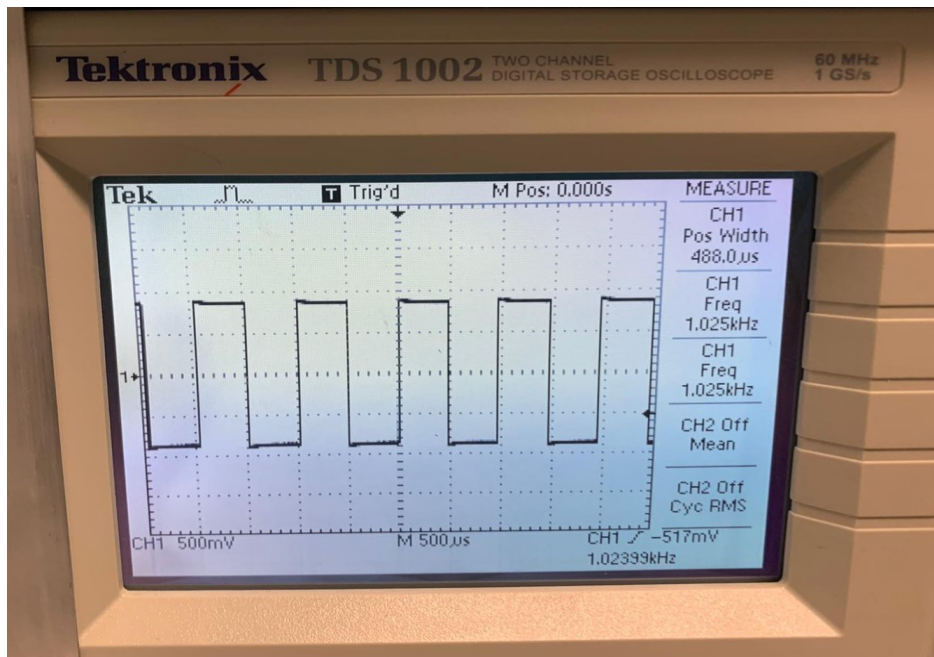
w(0x0E, 0x00);
}

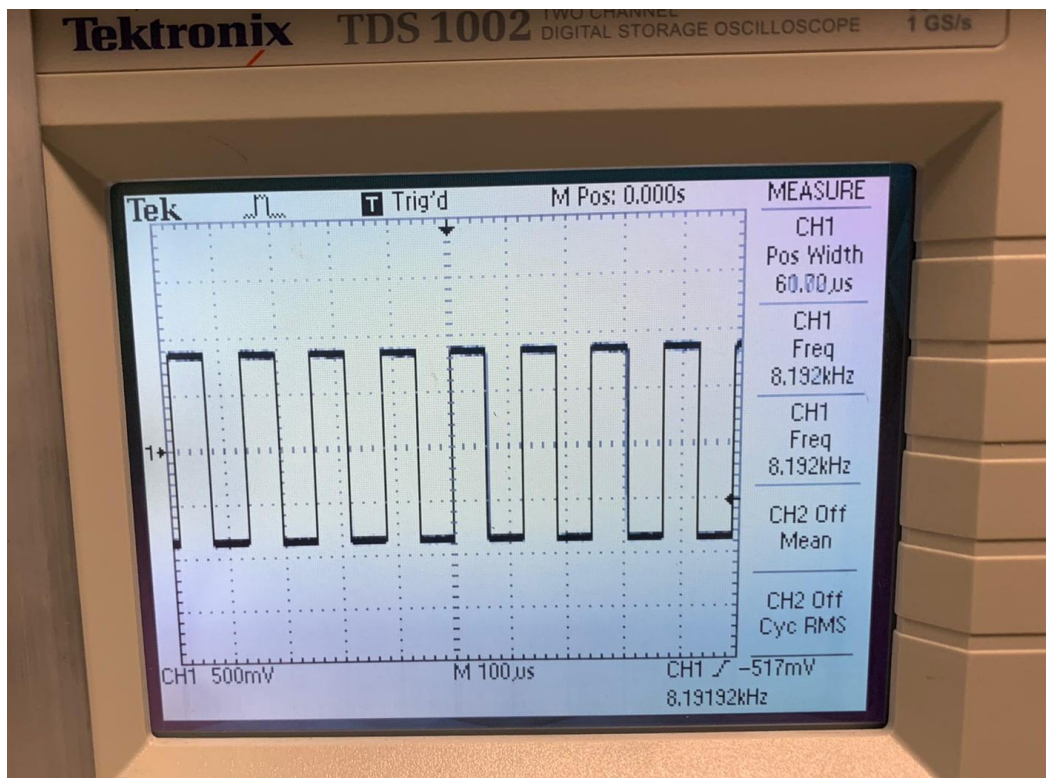
```

Method frequency() is used to select the range of frequency, where parameter a is to select the frequency and b number of seconds wave needed. If a==1 then frequency 1Hz will be selected so on 1KHz, 4KHz, and 8KHz and the link is given to see them working.

<https://drive.google.com/a/mail.dcu.ie/file/d/1TnHE0XnDJKgDgoUwQ6mg2ew2gVwPaEcW/view?usp=drivesdk>

```
a.frequency(1, 5);  
//a.frequency(2, 10);  
//a.frequency(3, 5);  
//a.frequency(4, 5);
```





- Novel feature

When Vcc is removed from DS3231 it can still function with the help of battery supply (Vbat). Bit 7 of the control register is EOSC. Writing to this bit can control the activity of an oscillator. When Vcc applied always the oscillator will be high regardless of the value in this register. The oscillator will be stopped when bit 7 is made high and Vcc changes to Vbat. When it is made low and Vcc changes to Vbat the oscillators will turn on.

```
void Realtime :: EOSC(char c){
    w(0x0E, c);
    cout << " Oscillator will stop only when Vcc changed to Vbat " << endl;
}

a.EOSC(0x9E); // one should use this function only when oscillators need to be stop and
              //works only when VCC change to Vbat

a.BATB(0); // should chnge to vbat
//a.BATB(2);
```










Battery backed square wave enabler method BATB() will allow us to select frequency range when DS3231 is working on Vbat.










```
void Realtime ::BATB(int a){
    cout << " Frequency is seen across INT/SQW only when Vcc changed to vbat" << endl;
    if(a==1)
        w(0x0E, 0x43); // 1Hz
    else if(a==2)
        w(0x0E, 0x4b); // 1 KHz
    else if(a==3)
        w(0x0E, 0x53); // 4 KHz
    else if(a==4)
        w(0x0E, 0x5b); // 8 KHz
    else{
        cout << "No frequency selected" << endl;
    }
}
```


Full output

```
File Edit Tabs Help
pi@raspberrypi:~/assignment1 $ g++ Realtime.cpp -o realtime
pi@raspberrypi:~/assignment1 $ ./realtime
Before setting Time :
5:34:8
Before setting Date :
5:21:5:20
writing HH : MM : SS
4:55:50
writing Day : Date : Month : Year
Thursday :21:5:20
Reading temperature
Temperature :27.5
Setting alarm 2 at4:56:day
If LED turns off at specified alarm time then alarm is triggered
Current time is 4:55:50
Setting alarm 1 at4:56:20:date21
If LED turns off at specified alarm time then alarm is triggered
Current time is 4:56:5
generating 1Hz for 5sec
Oscillator will stop only when Vcc changed to Vbat
Frequency is seen across INT/SQW only when Vcc changed to Vbat
No frequency selected
Closed file
pi@raspberrypi:~/assignment1 $
```

- Gitlab activity details






	Pushed to branch master 19db0e4b - check	
	Prathosh T S @Prathosh Pushed to branch master 207c466e - q	1 week ago
	Prathosh T S @Prathosh Pushed to branch master 9f6708e0 - seconds bit updated	1 week ago
	Prathosh T S @Prathosh Pushed to branch master 17df3ace - writing into register 0x00 (seconds bit) and is setted to 20 sec. B...	1 week ago
	Prathosh T S @Prathosh Pushed to branch master 10f4049e - instead of day 1,2,3.....Monday, tuesday is done and temperature re...	1 week ago
	Prathosh T S @Prathosh Pushed to branch master 3d490176 - upadate cnstructor	1 week ago
	Prathosh T S @Prathosh Pushed to branch master 6aa7433b - with the help of given c program, tried to write C++ to read time	1 week ago
	Prathosh T S @Prathosh Pushed new branch master	1 week ago
	Prathosh T S @Prathosh Created project Prathosh T S / Assignment1	1 week ago

	Prathosh T S @Prathosh ↪ Pushed to branch master #37b69ee - making it user friendly, rather than hard coding it	2 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master 0cd64c31 - square wave working	2 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master fc298d4c - Alarm1 and Alarm 2 working perfectly and verified with LED	3 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master #03bdc12 - :	3 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master 5153608a - structured	3 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master 34b15fb6 - Alarm 1 set and checked using led	3 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master ecd66626 - writing time and date with reading temperature	4 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master cc1abcc1 - some problem is writing, writinf into reg 0x00 is working but has s...	5 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master 194b0e4b - check	5 days ago

Prathosh T S > Assignment1 > Activity

All Push events Merge events Issue events Comments Team



	Prathosh T S @Prathosh ↪ Pushed to branch master 36ac7aee - final user driver for DS3231	4 minutes ago
	Prathosh T S @Prathosh ↪ Pushed to branch master 66473c5a - final	21 hours ago
	Prathosh T S @Prathosh ↪ Pushed to branch master bde8eab3 - added two novel feature (oscillator control and battery backup)	1 day ago
	Prathosh T S @Prathosh ↪ Pushed to branch master 1c755a3a - updated with functions for time, date, alarm1, alarm2 and temperature	2 days ago
	Prathosh T S @Prathosh ↪ Pushed to branch master #37b69ee - making it user friendly, rather than hard coding it	2 days ago

Approach to the whole task

- Firstly I started with writing c++ for reading only time with the help of given C code, then I used constructor to pass path and address instead of hard coding.
- Reading calendar register and updated days as Monday so on instead of a number.
- Writing values into registers was the hardest part, then I was able to write into registers, so I tried writing into the clock and calendar register and was successful.
- Then setting alarm one for a day and another for a date and wiring LED for interrupt, the hardest part was to make LED glow back again after the alarm.
- I structured my program a bit and started working to get a square wave. Initially, my mistake was I was converting to BCD before writing to RS1 and RS2 bit, then I created another function called w() to write direct Hexa number to it.
- Till here I was hard coding everything in main, so structured everything by making function and passing values.
- Added two novel features and LKM.

- LKM

Linux supports the use of RTCs directly within the OS using LKMs. Without any software program, RTC can maintain current time and date with the help of LKM so the first step is to interface the I2C device with compatible LKM. A new device should be added after that if we check `i2cdetect -y 1`, we will see UU instead of 68 which represents that device is under control of drivers.

```

pi@raspberrypi: /sys/c
File Edit Tabs Help
pi@raspberrypi:/sys/class/rtc/rtc0 $ i2cdetect -y 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 $

```

```

pi@raspberrypi: /sys/class/rtc/rtc0
File Edit Tabs Help
hwmon          16384  2  rtc_ds1307,raspberrypi_hwmon
pi@raspberrypi:~ $ sudo sh -c "echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device"
pi@raspberrypi:~ $ dmesg|tail -1
[ 982.140011] i2c i2c-1: new_device: Instantiated device ds1307 at 0x68
pi@raspberrypi:~ $ ls -l /dev/rtc*
lrwxrwxrwx 1 root root    4 Mar  8 01:33 /dev/rtc -> rtc0
crw----- 1 root root 253, 0 Mar  8 01:33 /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
00:23:25
pi@raspberrypi:/sys/class/rtc/rtc0 $ date
Sun  8 Mar 01:40:48 GMT 2020
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -r
2000-01-01 00:24:31.086885+00:00
pi@raspberrypi:/sys/class/rtc/rtc0 $ pi@erpi ~ $ sudo hwclock -w
bash: pi@erpi: command not found
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -w
pi@raspberrypi:/sys/class/rtc/rtc0 $ sudo hwclock -r
2020-03-08 01:42:13.364226+00:00
pi@raspberrypi:/sys/class/rtc/rtc0 $

```

Commands such as `sudo hwclock -r` to read, `sudo hwclock -r` to write and `sudo hwclock -r` to use system clock can be used. We can make reset the time on booting with the help of system service and adding LKM to it. The given commands were not working so I followed those steps suggested by a classmate in the discussion forum.

```
GNU nano 3.2 /etc/rc.local
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
sudo hwclock -s
exit 0
```

After writing a nano, disabled Network Time protocol and after reboot time and the date was according to the RTC module. If we again enable NTP then RTC makes use of this protocol and after boot, it will update time and dates.