University Of Khartoum

Microcontroller and embedded systems coursework project

Mosque Watch

Prepared by:

- **Mohammed Salah Ibrahim Abdallah – 144074**
- **Maeen Abdelbadea NasrAllah – 144096**
- **Husam Mohammed Abduljabbar – 144024**
- **Amro kamal Mohammed Abbas - 144048**
- **Mohammed Ali Abdallah Ahmed - 144080**

27/9/2017

Supervised by Ustaza

Sara Omer Mahjoob

Functional

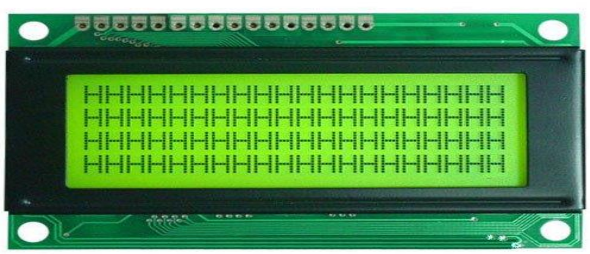
Our project displays the current, Current Date Next prayer’s time on a LCD 20x4 chip and Causes a sound alarm for one minute at prayer time. It Keeps time and date even after power failure using a Real Time Clock (RTC) chip.

The prayer times are set to work for Khartoum, Sudan region.

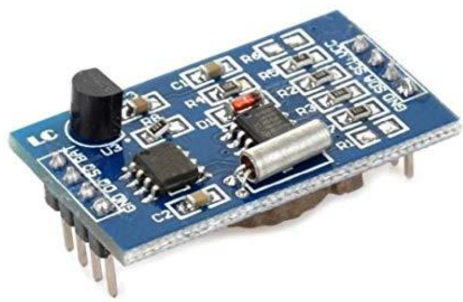
Technical

The hardware components used in this project are:

1. **LCD 20x4**

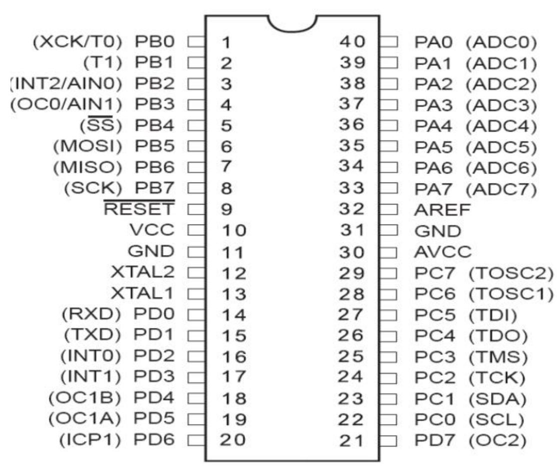


2. **DS1307(Serial Real Time Clock)**



3. **Buzzer.**

4. **AVR ATMEGA32**



5. Potentiometer.
6. Bread board.
7. Crystal (quartz Crystal).

Programming for RTC DS1307

Initially, while using RTC first time, we have to set the clock and calendar values, then RTC always keep updating this clock and calendar values.

We will set the RTC clock and calendar values in 1st step and in 2nd step we will read these value.

Step1: Setting Clock and Calendar to RTC DS1307

- In RTC coding, we require first RTC device address (slave address) through which microcontroller wants to communicate with the DS1307.
- DS1307 RTC device address is 0xD0 (given in data sheet).
- Initialize I2C in ATmega16 /32.
- Start I2C communication with device write address i.e. 0xD0.
- If address is matched we get acknowledgement signal.
- Send the Register address of seconds which is 0x00, then send the value of seconds to write in RTC. RTC address gets auto incremented so next, we only have to send the values of minutes, hours, day, date, month and year.
- And then stop the I2C communication.

Step2: Reading Time and Date value from RTC DS1307

- In the second step we learn how to read the data from the RTC, i.e. second, minute, hours, etc.

- Start the I2C communication with device write address i.e. 0xD0.

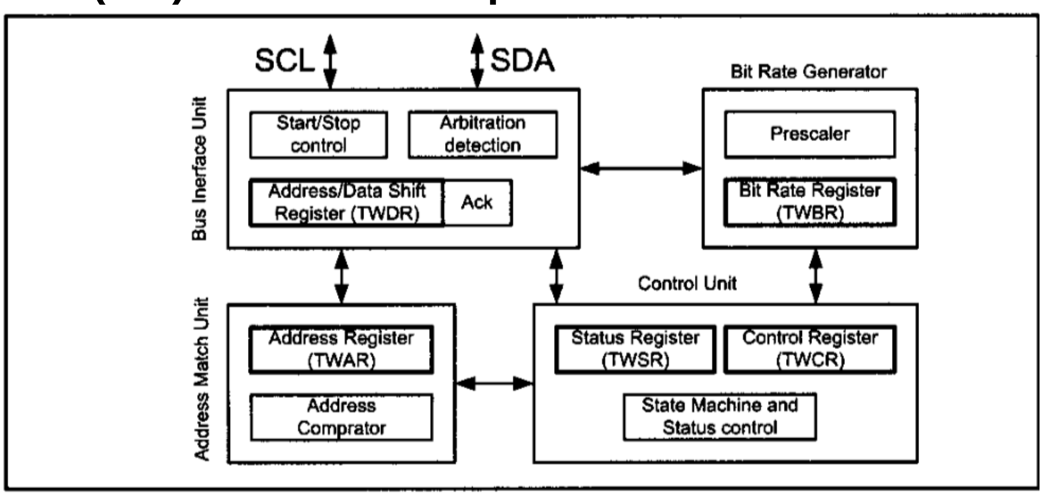
- Then write the register value from where we have to read the data (we read from location 00 i.e. read the second).
- Then repeated start I2C with device read address i.e. 0xD1.
- Now Read the data with acknowledgement from location 00.

- For read the last location always read with the negative acknowledgement, then device will understand this is the last data read from the device.
- For read next location of register address will get auto incremented.

First of all the communication between the AVR microcontroller chip and the RTC “real time clock” is established using the i2c communication protocol .

The communication is exchanged through the data & clock pins in the RTC chip.

TWI (I2C) in the AVR chip:



The RTC has 5 registers used in the control & data exchange:

TWI Control Register (TWCR)

TWCR controls the operation of the TWI. In Figure 18-13 you see each bit of TWCR and a short description of it. Here we will describe some of these bits in more detail.

TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE

Bit 7 – TWINT: TWI Interrupt
This bit is set by hardware when the TWI module has finished its current job. If the TWI and general interrupt are enabled, changing TWINT to one will cause the MCU to jump to the TWI interrupt vector. Clearing this flag starts the operation of the TWI. TWINT must be cleared by software.

Bit 6 – TWEA: TWI Enable Acknowledge
Making this bit HIGH will enable the generation of ACK when needed in slave or receiver mode.

Bit 5 – TWSTA: TWI START condition Bit
Making this bit HIGH will generate a START condition if the bus is free; otherwise, the TWI module waits for the bus to become free and then generates a START condition

Bit 4 – TWSTO: TWI STOP condition bit
In master mode, making this bit HIGH causes the TWI to generate a STOP condition. This bit is cleared by hardware when the STOP condition is transmitted.

Bit 3 – TWWC: TWI Write Collision Flag

Bit 2 – TWEN: TWI Enable
Making this bit HIGH enables the TWI module.

Bit 0 – TWIE: TWI Interrupt Enable
Making this bit HIGH enables the TWI interrupt if the general interrupt is enabled.

TWI Data Register (TWDR)

In Receive mode, the last received byte will be in the TWDR, and in Transmit mode, you should write the next byte into TWDR to be transmitted. As we mentioned before, you can access the TWDR only when the TWIE is set to one otherwise collision happens. This means the Data Register cannot be initialized by the user before the first interrupt occurs.

TWI Address Register (TWAR)

TWAR contains the 7-bit slave address to which the TWI will respond when working as slave. The eighth bit (LSB) of TWAR is TWGCE (TWI General Call Recognition Enable). It controls recognition of general call address (00). If this bit is set to one, receiving of a general call address will cause an interrupt request.

TWI Status Register (TWSR)

As you see in Figure 18-12, five bits of TWSR are dedicated to show the status of the TWI logic and bus. Notice that if you read TWSR, you will read both the status bits and the prescaler value. To check the status bits, you should mask the two LSB bits (prescaler values) to zero. In this book we do not list all of the status codes and their meanings, but we will cover some of more common ones. To see the complete list of status register codes, you should refer to the data sheet of the chip. Next we will see how to use these bits when we want to program the AVR to use the TWI module.

Algorithms

Steps to program the AVR in master operating mode:

1_Initzilization:

To initialize the TWI module to operate in master operating mode, we should do the following steps:

1. Set the TWI module clock frequency by setting the values of the TWBR register and the TWPS bus in the TWSR register.

2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.

To start data transfer in master operating mode, we must transmit a START condition. This is done by setting the TWEN, TWSTA ,and TWINT bit of TWCR to one.

***send data:**

To send data:

1. Copy the data byte to the TWDR.

2. Set the TWEN and TWINT bits of the TWCR register to one to start sending the byte.

3. Poll the TWINT flag in the TWCR register to see whether the byte transmitted completely.

***receive data:**

To receive data:

1. Set TWEN and TWINT bits of the TWCR register to one to start receiving a byte.
2. Poll the TWINT flag in the TWCR register to see whether a byte has been received completely.
3. Copy the received byte from the TWDR to another register to save it.

To stop the transfer we must transmit a STOP condition.

Steps to program the AVR in slave operating mode:

In slave mode we can’t transmit START or STOP conditions. A slave device should listen to the bus and wait to be addressed by a master device or general call.

Initialization:

1. Set the slave address by setting the values for TWAR registers. The upper seven bits of TWAR are the slave address, and the eight bit is TWGCE.
2. Enable the TWI module by setting the TWEN bit in the TWCR register to one.
3. Set the TWEN, TWINT, and TEA bits of TWCR to one to enable the TWI and acknowledge generation.

After initialization the slave should listen to the bus to detect when it is addressed by a master device . Wjen the TWI module detects its own address on the bus , it returns ACK and then sets the TWINT flag in the TWCR register to one.

***send data:**

1. Copy the data byte to the TWDR .

2. Set the TWEN , TEA, TWINT bits of the TWCR register ot one to start sending the byte.

3. Poll the TWINT flag in the TWCR register to see when the byte is completely transmitted.

***receive data:**

After being addressed by a master do the following:

1. Set the TWEN , TEA, TWINT bits of the TWCR register ot one to start sending the byte.

2. Poll the TWINT flag in the TWCR register to see when the byte is received completely.
3. Copy the received byte from the TWDR to another register to save it.

Now from the LCD datasheet we have the instruction table:

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (Max)
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter.	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	–	Sets DD RAM address 0 as address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	1/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 μs
Display On/ Off Control	0	0	0	0	0	0	1	D	C	B	Sets On/Off of entire display (D), cursor On/Off (C), and blink of cursor position character (B).	40 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	–	–	Moves cursor and shifts display without changing DD RAM contents.	40 μs
Function Set	0	0	0	0	1	DL	N	F	–	–	Sets interface data length (DL), number of display lines (L), and character font (F).	40 μs
Set CG RAM Address	0	0	0	1	AGC						Sets CG RAM address. CG RAM data is sent and received after this setting.	40 μs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. DD RAM data is sent and received after this setting.	40 μs
Read Busy Flag & Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40 μs
Write Data CG or DD RAM	1	0	Write Data								Writes data into DD or CG RAM.	40 μs
Read Data CG or DD RAM	1	1	Read Data								Reads data from DD or CG RAM.	40 μs

And we have address ranges for the LCD 20x4 as:

20 × 4 LCD	Line 1:	80	81	82	83	through 93
	Line 2:	C0	C1	C2	C3	through D3
	Line 3:	94	95	96	97	through A7
	Line 4:	D4	D5	D6	D7	through E7

*Note:

That the LCD receives one character at time but the TRC gets the time as multiple characters so we are using an unpacked BCD exchange as shown in the code.

The algorithms for sending commands and data to LCD:

We are using 4-bit mode to save bits, we send the high nibbles first then the lower nibbles ; the data got from the RTC is in BCD formula so we have a conversion function that:

- Get the number In BCD from RTC
- Obtains the higher nibble by anding with 0xf0
- Then adds with 30 to convert it to ascii so the LCD could show it correctly
- Do the same but for the lowest nibble

We have PORTA and PORTB as outputs; specifically pins 4,5,6,7 for the data transfer in PORTA. And 5, 6, and 7 for RS, RW and EN respectively in PORTB

We first initialize the LCD:

We put in the 4-bit mode, the following sequence of commands should be sent to the LCD: 33, 32 and 28 in hex to tell the LCD to go into 4-bit mode

To send commands:

We make pins RS and R/W = 0 and put the command number on the data pins. Then we send a high to low pulse to the EN pin to enable the internal latch of the LCD. After each command we set a delay of 100 microseconds to let the LCD module run command.

To send data to the LCD:

We make the pins RS = 1 and R/W =0. Then we put the data on the data pins and send a high to low pulse to the EN pin to enable the internal latch of the LCD. We set a delay of 100 microseconds to the LCD module write the data on the screen.

To know which the next prayer is:

The function take the current time as parameters, then it has five comparisons between each two prayers to define which is next.

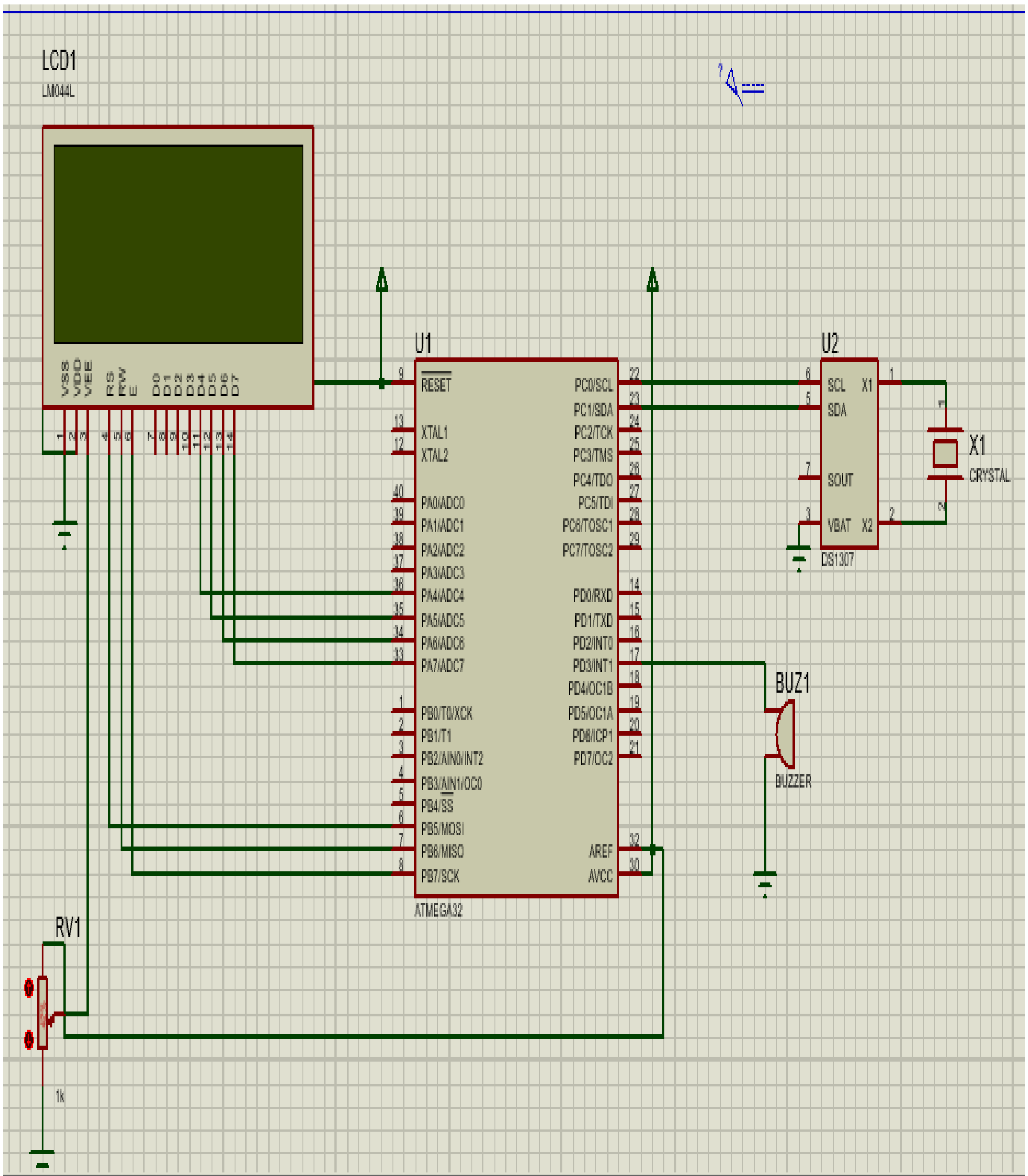
It firstly compares the hours between two hours at a time, if the minutes is less the next prayer is the last if greater it goes to the prayer after the one being compared.

And if the hours is less than the compered then the compared is the next prayer.

The Alarm:

It keeps on comparing the current time with 5 conditional ifs if antmatch it sets the buzzer until the minute is over.

Hardware Design and outline



Code

```
#define F_CPU 1000000UL

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


#define LCD_DataPRT PORTA
#define LCD_DataDDR DDRA
#define LCD_RS 5
#define LCD_RW 6
#define LCD_EN 7
#define LCD_SETPRT PORTB
#define LCD_SETDDRPRT DDRB


/////////////// prayer's time /////////////////////
unsigned char FAJR_H=0x04;
//unsigned char FAJR_HH='0';
//unsigned char FAJR_ML='0';
unsigned char FAJR_M=0x30;

unsigned char DOHR_H=0x12;
//unsigned char DOHR_HL='2';
//unsigned char DOHR_MH='0';
unsigned char DOHR_M=0x00;

unsigned char ASR_H=0x16;
//unsigned char ASR_HH='1';
//unsigned char ASR_ML='0';
unsigned char ASR_M=0x00;

unsigned char MAG_H=0x17;
```

```c
//unsigned char MAG_HL='7';
//unsigned char MAG_MH='0';
unsigned char MAG_M=0x050;

unsigned char ESHA_H=0x19;
//unsigned char ESHA_HL='9';
//unsigned char ESHA_MH='0';
unsigned char ESHA_M=0x00;



void i2c_init (void){
    TWSR = 0x00;
    TWBR = 0x47;
    TWCR = 0x04;
}

void i2c_start()
{
    TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
    while(!(TWCR & (1<<TWINT)));
}

void i2c_write(unsigned char data){
    TWDR = data;
    TWCR = (1<<TWINT)|(1<<TWEN);
    while(!(TWCR & (1<<TWINT)));
}
unsigned char i2c_read (unsigned char ackVal){
    TWCR = (1<<TWINT)|(1<<TWEN)|(ackVal<<TWEA);
    while(!(TWCR & (1<<TWINT)));
    return TWDR;
}
```

```
void i2c_stop (){
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
    int k;
    for (k = 0;k<100;k++);
}

void rtc_init(){
    i2c_init();   //initialize I2C module
    i2c_start();    //transmit START condition
    i2c_write(0xD0);  //address DS1307 for write
    i2c_write(0x07);  //set register pointer to 7
    i2c_write(0x00);   //set value of location 7 to 0
    i2c_stop();   //transmit STOP condition

}

void rtc_setTime(unsigned char h,unsigned char m,unsigned char s){
    i2c_start();    //transmit START condition
    i2c_write(0xD0);  //address DS1307 for write
    i2c_write(0);  //set register pointer to 0
    i2c_write(s);   //set seconds
    i2c_write(m);   //set minutes
    i2c_write(h);   //set hour
    i2c_stop();   //transmit STOP condition
}

void rtc_setDate(unsigned char y,unsigned char m,unsigned char d){
    i2c_start();    //transmit START condition
    i2c_write(0xD0);  //address DS1307 for write
    i2c_write(0x04);  //set register pointer to 0
    i2c_write(d);   //set seconds
    i2c_write(m);   //set minutes
    i2c_write(y);   //set hour
```

```
    i2c_stop();   //transmit STOP condition
}

void rtc_getTime(unsigned char *h,unsigned char *m,unsigned char *s){
    i2c_start();    //transmit START condition
    i2c_write(0xD0);  //address DS1307 for write
    i2c_write(0);  //set register pointer to 0
    i2c_stop();   //transmit STOP condition

    i2c_start();    //transmit START condition
    i2c_write(0xD1);  //address DS1307 for write
    *h = i2c_read(1);   //read second, return ACK
    *m = i2c_read(1);   //read minute, return ACK
    *s = i2c_read(0);   //read hour, return ACK
    i2c_stop();   //transmit STOP condition
}

void rtc_getDate(unsigned char *y,unsigned char *m,unsigned char *d){
    i2c_start();    //transmit START condition
    i2c_write(0xD0);  //address DS1307 for write
    i2c_write(0x04);  //set register pointer to 0
    i2c_stop();   //transmit STOP condition

    i2c_start();    //transmit START condition
    i2c_write(0xD1);  //address DS1307 for write
    *d = i2c_read(1);   //read day, return ACK
    *m = i2c_read(1);   //read month, return ACK
    *y = i2c_read(0);   //set year, return NACK
    i2c_stop();   //transmit STOP condition
}

/*char * get_ASCII(unsigned char data){
    static char r[2] ;
     r[0] = '0' + (data>>4);
```

```
    r[0] = '0' + (data>>4);
    r[1]= '0' + (data & 0xF);

   return r;

   }*/


//////////////////////////////////////lcd

void Send_LCD_Com(unsigned char Command){
        LCD_DataPRT=Command & 0xF0;
        LCD_SETPRT &=~(1<<LCD_RS);            /*RS=0 for command       */
        LCD_SETPRT &=~(1<<LCD_RW);            /*RW=0 for write       */
        LCD_SETPRT |=(1<<LCD_EN);             /*EN=1      */
        _delay_us(1);
        LCD_SETPRT &=~(1<<LCD_EN);             /*EN=0      */
        _delay_us(100);

        LCD_DataPRT=Command<<4;
        LCD_SETPRT |=(1<<LCD_EN);             /*EN=1      */
        _delay_us(1);
        LCD_SETPRT &=~(1<<LCD_EN);             /*EN=0      */
        _delay_us(100);

}

void LCD_Init(){
        LCD_DataDDR=0xFF;
        LCD_SETDDRPRT=0xFF;

        LCD_SETPRT &=~(1<<LCD_EN);
        _delay_us(2000);
        Send_LCD_Com(0x33);
```

```
        Send_LCD_Com(0x32);
        Send_LCD_Com(0x28);
        Send_LCD_Com(0x0C);
        Send_LCD_Com(0x01);
        _delay_us(2000);
        Send_LCD_Com(0x06);
}

void Send_DATA_TO_LCD(unsigned char data){
        LCD_DataPRT=data & 0xF0;

        LCD_SETPRT |=(1<<LCD_RS);          /*RS=0 for command      */
        LCD_SETPRT &=~(1<<LCD_RW);          /*RW=0 for write      */
        LCD_SETPRT |=(1<<LCD_EN);          /*EN=1      */
        _delay_us(1);
        LCD_SETPRT &=~(1<<LCD_EN);           /*EN=0      */
        _delay_us(100);

        LCD_DataPRT=data<<4;
        LCD_SETPRT |=(1<<LCD_EN);          /*EN=1      */
        _delay_us(1);
        LCD_SETPRT &=~(1<<LCD_EN);           /*EN=0      */
        _delay_us(100);

}


void lcd_gotoxy (unsigned char x , unsigned char y)
{
        unsigned char firstCharAdr[] = {0x80, 0xC0, 0x94 , 0xD4};
        Send_LCD_Com(firstCharAdr[y-1] + x - 1);
        _delay_us(100);
}

```

```
void LCD_Print(char* str){
        unsigned char i=0;
        while (str[i]!=0)
        {

                Send_DATA_TO_LCD(str[i]);
                i++;
        }

}
///////////////////////////////main

void show_Time(unsigned char h,unsigned char m,unsigned char s){
        unsigned sh=(s>>4)+'0';
        unsigned sl=(s & 0x0f)+'0';

        unsigned mh=(m>>4)+'0';
        unsigned ml=(m & 0x0f)+'0';

        unsigned hh=(h>>4)+'0';
        unsigned hl=(h & 0x0f)+'0';

        Send_DATA_TO_LCD(hh);
        Send_DATA_TO_LCD(hl);
        Send_DATA_TO_LCD(':');
        Send_DATA_TO_LCD(mh);
        Send_DATA_TO_LCD(ml);
        Send_DATA_TO_LCD(':');
        Send_DATA_TO_LCD(sh);
        Send_DATA_TO_LCD(sl);
}

void show_Date(unsigned char d,unsigned char mon,unsigned char y){
        unsigned dh=(d>>4)+'0';
```

```
        unsigned dl=(d & 0x0f)+'0';

        unsigned monh=(mon>>4)+'0';
        unsigned monl=(mon & 0x0f)+'0';

        unsigned yh=(y>>4)+'0';
        unsigned yl=(y & 0x0f)+'0';


        LCD_Print("20");
        Send_DATA_TO_LCD(dh);
        Send_DATA_TO_LCD(dl);
        Send_DATA_TO_LCD(':');
        Send_DATA_TO_LCD(monh);
        Send_DATA_TO_LCD(monl);
        Send_DATA_TO_LCD(':');
        Send_DATA_TO_LCD(yh);
        Send_DATA_TO_LCD(yl);

}


int next_Prayer_Time(unsigned char h,unsigned char m){
        if(h>=ESHA_H || h<=FAJR_H){
                                        if(h==FAJR_H && m<FAJR_M)  return 1;
                                    else if(h==FAJR_H && m>FAJR_M)  return 2;
                                                        else if(h<FAJR_H || h>ESHA_H) return 1;}

    if(h>=FAJR_H && h<=DOHR_H ){
                                        if(h==DOHR_H && m<DOHR_M) return 2;
                                          else if(h==DOHR_H && m>DOHR_M)  return 3;
                                                                  else if(h<DOHR_H) return 2;}

    if(h>=DOHR_H && h<=ASR_H ) {
```

```
                                        if(h==ASR_H && m<ASR_M) return 3;
                                        else if(h==ASR_H && m>ASR_M ) return 4;
                                                                     else if(h<ASR_H) return 3;}

    if(h>=ASR_H && h<=MAG_H ) {
                                        if(h=MAG_H && m<MAG_M) return 4;
                                         else if(h==MAG_H && m>MAG_M)  return 5;
                                                                          else if(h<MAG_H) return 4;}

    if(h>=MAG_H && h<=ESHA_H){
                                         if(h==ESHA_H && m<ESHA_M) return 5;
                                         if(h==ESHA_H && m>ESHA_M) return 1;
                                                             if(h<ESHA_H) return 5;
                                     }



}

void alaram(unsigned char h,unsigned char m){
        if(h==FAJR_H && m==FAJR_M){
           PORTD |= 0x08;}
        else if(h==DOHR_H && m==DOHR_M) {
           PORTD |= 0x08;}
        else if(h==ASR_H && m==ASR_M) {
           PORTD |= 0x08;}
        else if(h==MAG_H && MAG_M) {
           PORTD |= 0x08;}
        else if(h==ESHA_H && m==ESHA_M) {
           PORTD |= 0x08;}
        else PORTD &= 0xF7;
}
void buzzer_config(){
   DDRD |= 0x08; //PD3 as output
```

```
    PORTD &= 0xF7; // turn off buzzer
    }

int main()
 {
    buzzer_config();

    LCD_Init();

    unsigned char s,m,h;
    unsigned char d,mon,y;

    rtc_init();
    rtc_setTime(0x18,0x59,0x55);
    rtc_setDate(0x18,0x9,0x20);



    while (1){

  rtc_getTime(&s,&m,&h);
  rtc_getDate(&d,&mon,&y);

 alaram(h,m);


lcd_gotoxy(1 , 1);
LCD_Print("TIME ");
show_Time(h,m,s);

lcd_gotoxy(1 , 2);
LCD_Print("DATE ");
show_Date(d,mon,y);
```

```
int npt;
npt=next_Prayer_Time(h,m);
lcd_gotoxy(1,3);
LCD_Print("NEXT PRAYER ");


//lcd_gotoxy(1,4);
switch (npt)
{
        case 1:
        show_Time(FAJR_H,FAJR_M,0x00);
        break;
        case 2:
        show_Time(DOHR_H,DOHR_M,0x00);
        break;
        case 3:
        show_Time(ASR_H,ASR_M,0x00);
        break;
        case 4:
        show_Time(MAG_H,MAG_M,0x00);
        break;
        case 5:
        show_Time(ESHA_H,ESHA_M,0x00);
        break;

}



   }

   return 0;
 }
```