LAB 3 - INTERFACE DS1307 REAL TIME CLOCK CHIP AND DISPLAY THE TIME, DAY AND DATE

Aim:

To Interface DS1307 real time clock chip with PIC16F877A and display the time, day and date

Requirements:

i. **Hardware** – PIC16F877A, 20Mhz oscillator, capacitors -33pf, DS1307 –RTC, Oscilloscope, power supplies.

ii. **Software** – Proteus (simulation software), MPLAB IDE

Theory:

Using an RTC IC is the most frequent technique for a microcontroller to keep track of the time or date in the real world. RTC stands for Real Time Clock; this IC keeps track of the current time and date in the real world and sends this information to the microcontroller as needed.

PIC16F877A I2C MODULE - The PIC16F877A Microcontroller consists of a MASTER SYNCHRONOUS SERIAL PORT (MSSP) MODULE. MSSP module is a serial interface, useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I2C)
  - Full Master mode
  - Slave mode (with general address call)

The I2C interface supports the following modes in hardware:

- Master mode
- Multi-Master mode
- Slave mode

Two pins are used for data transfer: The user must configure these pins as inputs or outputs through the TRISC 4&3 bits

- Serial clock (SCL) – RC3/SCK/SCL
- Serial data (SDA) – RC4/SDI/SDA

The MSSP module has six registers for I2C operation. These are:

- MSSP Control Register (SSPCON)
- MSSP Control Register 2 (SSPCON2)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)

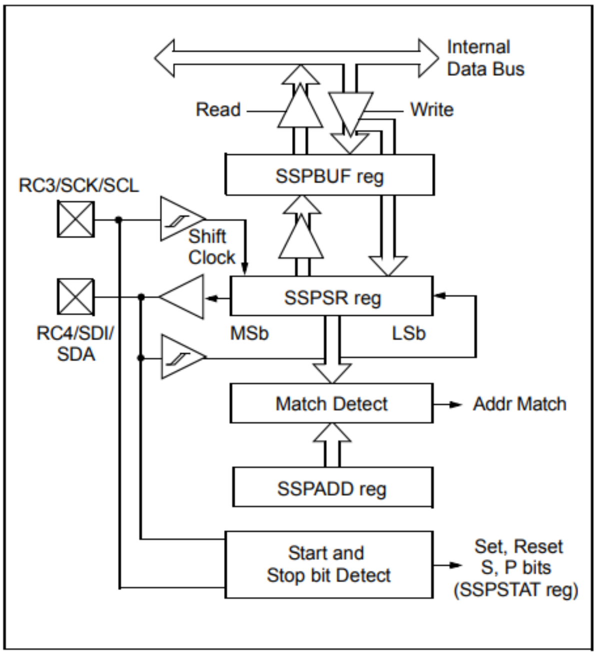
- MSSP Shift Register (SSPSR) – Not directly accessible
- MSSP Address Register (SSPADD)

SSPCON, SSPCON2 and SSPSTAT are the control and status registers in I2C mode operation. The SSPCON and SSPCON2 registers are readable and writable. The lower six bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write.

SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from. SSPADD register holds the slave device address when the SSP is configured in I2C Slave mode. When the SSP is configured in Master mode, the lower seven bits of SSPADD act as the baud rate generator reload value.

In receive operations, SSPSR and SSPBUF together create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set. During transmission, the SSPBUF is not double buffered. A write to SSPBUF will write to both SSPBUF and SSPSR.

MSSP BLOCK DIAGRAM (I^{2}C MODE)



The MSSP module functions are enabled by setting MSSP Enable bit, SSPEN (SSPCON<5>). The SSPCON register allows control of the I2C operation. Four mode selection bits (SSPCON<3:0>) allow one of the following I2C modes to be selected:

- I2C Master mode, clock = OSC/4 (SSPADD + 1)
- I2C Slave mode (7-bit address)
- I2C Slave mode (10-bit address)
- I2C Slave mode (7-bit address) with Start and Stop bit interrupts enabled
- I2C Slave mode (10-bit address) with Start and Stop bit interrupts enabled
- I2C Firmware Controlled Master mode, slave is Idle

SSPSTAT: MSSP STATUS REGISTER (I^{2}C MODE) (ADDRESS 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Slew Rate Control bit

In Master or Slave mode:
`1` = Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
`0` = Slew rate control enabled for high-speed mode (400 kHz)

bit 6 **CKE:** SMBus Select bit

In Master or Slave mode:
`1` = Enable SMBus specific inputs
`0` = Disable SMBus specific inputs

bit 5 **D/$\overline{A}$:** Data/$\overline{\text{Address}}$ bit

In Master mode:
Reserved.

In Slave mode:
`1` = Indicates that the last byte received or transmitted was data
`0` = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit

`1` = Indicates that a Stop bit has been detected last
`0` = Stop bit was not detected last

Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 3 **S:** Start bit

`1` = Indicates that a Start bit has been detected last
`0` = Start bit was not detected last

Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 2 **R/$\overline{W}$:** Read/$\overline{\text{Write}}$ bit information (I^{2}C mode only)

In Slave mode:
`1` = Read
`0` = Write

Note: This bit holds the R/$\overline{W}$ bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit or not $\overline{\text{ACK}}$ bit.

In Master mode:
`1` = Transmit is in progress
`0` = Transmit is not in progress

Note: ORing this bit with SEN, RSEN, PEN, RCEN or ACKEN will indicate if the MSSP is in Idle mode.

bit 1 **UA:** Update Address (10-bit Slave mode only)

`1` = Indicates that the user needs to update the address in the SSPADD register
`0` = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit

In Transmit mode:
`1` = Receive complete, SSPBUF is full
`0` = Receive not complete, SSPBUF is empty

In Receive mode:
`1` = Data Transmit in progress (does not include the $\overline{\text{ACK}}$ and Stop bits), SSPBUF is full
`0` = Data Transmit complete (does not include the $\overline{\text{ACK}}$ and Stop bits), SSPBUF is empty

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

SSPCON1: MSSP CONTROL REGISTER 1 (I^{2}C MODE) (ADDRESS 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit

In Master Transmit mode:

1 = A write to the SSPBUF register was attempted while the I^{2}C conditions were not valid for a transmission to be started. (Must be cleared in software.)

0 = No collision

In Slave Transmit mode:

1 = The SSPBUF register is written while it is still transmitting the previous word. (Must be cleared in software.)

0 = No collision

In Receive mode (Master or Slave modes):

This is a “don’t care” bit.

bit 6 **SSPOV:** Receive Overflow Indicator bit

In Receive mode:

1 = A byte is received while the SSPBUF register is still holding the previous byte. (Must be cleared in software.)

0 = No overflow

In Transmit mode:

This is a “don’t care” bit in Transmit mode.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit

1 = Enables the serial port and configures the SDA and SCL pins as the serial port pins

0 = Disables the serial port and configures these pins as I/O port pins

Note: When enabled, the SDA and SCL pins must be properly configured as input or output.

bit 4 **CKP:** SCK Release Control bit

In Slave mode:

1 = Release clock

0 = Holds clock low (clock stretch). (Used to ensure data setup time.)

In Master mode:

Unused in this mode.

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits

1111 = I^{2}C Slave mode, 10-bit address with Start and Stop bit interrupts enabled

1110 = I^{2}C Slave mode, 7-bit address with Start and Stop bit interrupts enabled

1011 = I^{2}C Firmware Controlled Master mode (Slave Idle)

1000 = I^{2}C Master mode, clock = FOSC/(4 * (SSPADD + 1))

0111 = I^{2}C Slave mode, 10-bit address

0110 = I^{2}C Slave mode, 7-bit address

Note: Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

SSPCON2: MSSP CONTROL REGISTER 2 (I^2C MODE) (ADDRESS 91h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

bit 7 **GCEN:** General Call Enable bit (Slave mode only)
`1` = Enable interrupt when a general call address (0000h) is received in the SSPSR
`0` = General call address disabled

bit 6 **ACKSTAT:** Acknowledge Status bit (Master Transmit mode only)
`1` = Acknowledge was not received from slave
`0` = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (Master Receive mode only)
`1` = Not Acknowledge
`0` = Acknowledge

Note: Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (Master Receive mode only)
`1` = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.
`0` = Acknowledge sequence Idle

bit 3 **RCEN:** Receive Enable bit (Master mode only)
`1` = Enables Receive mode for I^2C
`0` = Receive Idle

bit 2 **PEN:** Stop Condition Enable bit (Master mode only)
`1` = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware.
`0` = Stop condition Idle

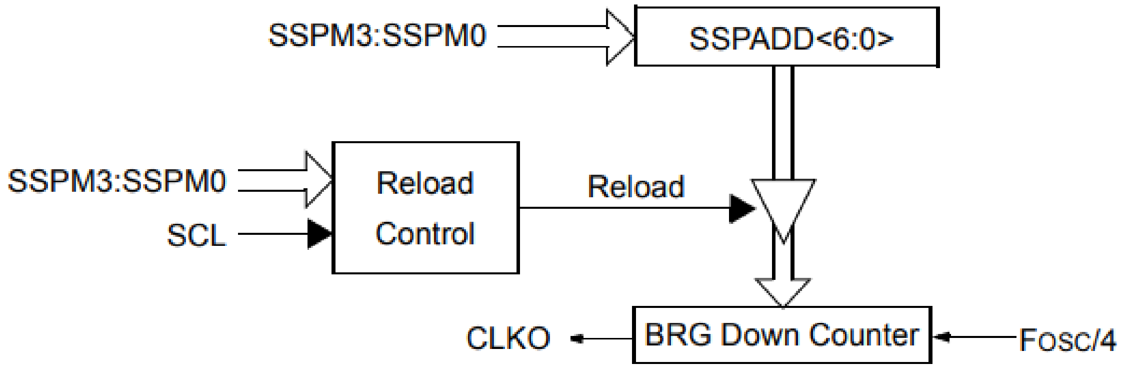
bit 1 **RSEN:** Repeated Start Condition Enabled bit (Master mode only)
`1` = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
`0` = Repeated Start condition Idle

bit 0 **SEN:** Start Condition Enabled/Stretch Enabled bit
<u>In Master mode:</u>
`1` = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware.
`0` = Start condition Idle
<u>In Slave mode:</u>
`1` = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
`0` = Clock stretching is enabled for slave transmit only (PIC16F87X compatibility)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

Note: For bits ACKEN, RCEN, PEN, RSEN, SEN: If the I^2C module is not in the Idle mode, this bit may not be set (no spooling) and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

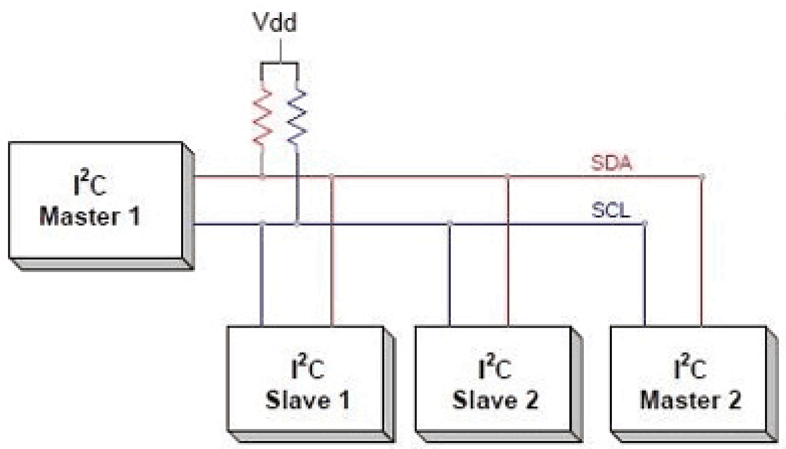
BAUD RATE GENERATOR BLOCK DIAGRAM



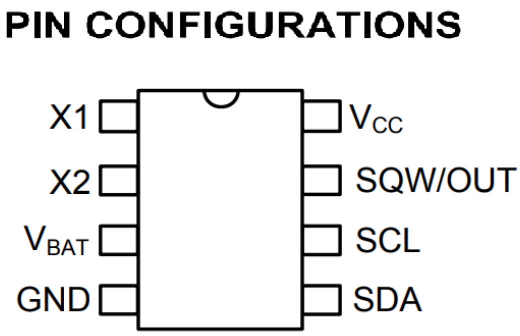
In I2C Master mode, the Baud Rate Generator (BRG) reload value is placed in the lower 7 bits of the SSPADD register. When a write occurs to SSPBUF, the Baud Rate Generator will automatically begin counting. The BRG counts down to 0 and stops until another reload has taken place. The BRG count is decremented twice per instruction cycle (TCY) on the Q2 and Q4 clocks. In I2C Master mode, the BRG is reloaded automatically. Once the given operation is complete (i.e., transmission of the last data bit is followed by ACK), the internal clock will automatically stop counting and the SCL pin will remain in its last state.

Phillips was the first to introduce I2C communication. It contains two wires, as previously stated, and these two wires will be connected across two devices. One device is referred to as the master, while the other is referred to as the slave. Communication between a Master and a Slave should and will always take place. I2C communication has the advantage of allowing several slaves to be connected to a single Master.

The I2C protocol is exclusively used for short-range communication. It is to some extent dependable, as it has a synchronised clock pulse to make it smart. This protocol is primarily used to communicate with sensors and other devices that must transmit data to a master. It comes in helpful when a microcontroller has to communicate with a large number of slave modules using only a few cables. The following Circuit shows the typical, connection of I2C devices, Only the master will be allowed to initiate contact at any given time. Because there are multiple slaves on the bus, the master must use a distinct address for each slave. Only the one with that specific address will respond with the information when addressed, while the others will remain silent.



DS1307 RTC - The DS1307 is a low-power clock/calendar with 56 bytes of battery-backed SRAM. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The DS1307 operates as a slave device on the I2 C bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. The accuracy of the clock is dependent upon the accuracy of the crystal and the accuracy of the match between the capacitive load of the oscillator circuit and the capacitive load for which the crystal was trimmed. Additional error will be added by crystal frequency drift caused by temperature shifts.



X1 & X2	Pins to connect the external 32.768kHz oscillator that provides the clock source to the IC.
V_{BAT}	Backup Supply Input for Any Standard 3V Lithium Cell or Other Energy Source. Battery voltage must be held between the minimum and maximum limits for proper operation. A lithium battery with 48mAh or greater will back up the DS1307 for more than 10 years in the absence of power at +25°C.
GND	Ground
SDA	Serial Data Input/Output. SDA is the data input/output for the I2 C serial interface. The SDA pin is open drain and requires an external pull-up resistor
SCL	SCL Serial Clock Input. SCL is the clock input for the I2 C interface and is used to synchronize data movement on the serial interface.
SQW/OUT	Square Wave/Output Driver. When enabled, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square-wave frequencies (1Hz, 4kHz, 8kHz, 32kHz).
VCC	Primary Power Supply. When voltage is applied within normal limits, the device is fully accessible and data can be written and read

CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. Table shows the RTC registers. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the BCD format.

On first application of power to the device the time and date registers are typically reset to 01/01/00 01 00:00:00 (MM/DD/YY DOW HH:MM: SS)

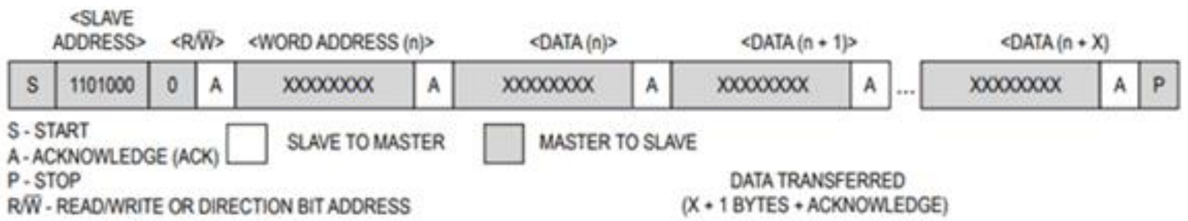
Timekeeper Registers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12 24	10 Hour PM/ AM	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

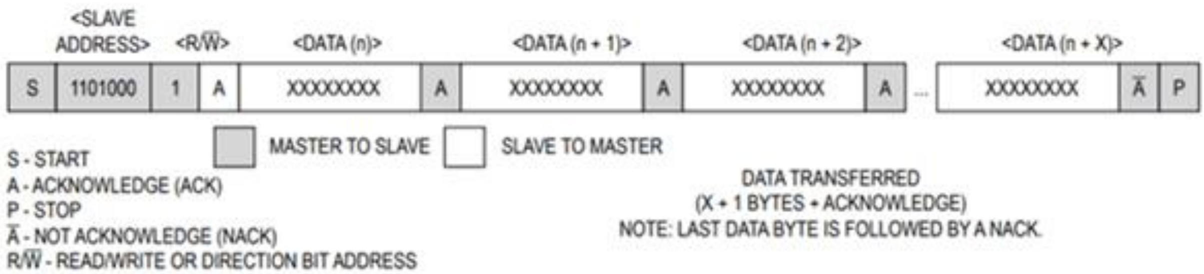
0 = Always reads back as 0.

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hour's register is defined as the 12-hour or 24-hour mode-select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20 to 23 hours). The hour's value must be re-entered whenever the 12/24-hour mode bit is changed

The PIC16F877A is configured as master and RTC is connected as slave. The seven-bit slave address for the DS1307 RTC is 1101000. Therefore, 0xD0 is transmitted to access the RTC in write mode and 0xD1 is transmitted to access the slave in read mode. The least significant bit will describe the operation to be performed whether to write into the RTC or to read from RTC. The below circuit shows the register memory map of the microcontroller to RTC, and RTC to controller in master – slave (vice versa) configuration.



PIC to RTC module Register Select



RTC module to PIC Register Select

PROCEDURE:

Step 1: - Initialize START condition**.**

Step 2: - Set RTC in read mode by transmitting RTC address with

Step 3: - RTC is now sending data from its registers. It must be ensured that data is written to the register whose address is now contained in the pointer. If a user wants to read a specific register, they must first access the RTC in write mode and write the requested register location to a pointer. After a RESTART or a STOP followed by a START, RTC should be addressed in read mode once more.

Step 4: - Each receiving byte should be acknowledged by the master to receive the next byte.

Step 5: -After receiving the last byte master should non-acknowledge the RTC.

Step 6: - Terminate communication with a STOP condition in the bus.

CODE: -

main.c

```
/*
 * File:   main.c
 * Author: Bala
 */


#pragma config FOSC = HS
#pragma config WDTE = OFF
#pragma config PWRTE = ON
#pragma config BOREN = ON
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF

#define _XTAL_FREQ 20000000 //We are running on 20MHz crystal

//Define the LCD pins
#define RS RD2
#define EN RD3
#define D4 RD4
#define D5 RD5
#define D6 RD6
#define D7 RD7

/*Set the current value of date and time below*/
int sec = 00;
int min = 25;
int hour = 12;
int day = 04;
int date = 29;
```

```
int month = 12;
int year = 21;
/*Time and Date Set*/

#include <xc.h>
#include "LCD.h" //Header for using LCD module
#include "PIC_I2C.h" // Header for using I2C protocal
#include "DS1307.h" //Header for using DS3231 RTC module

void day_print(unsigned int);


int main()
{

    TRISD = 0x00; //Make Port D pins as output for LCD interfacing

    Lcd_Start(); // Initialize LCD module

    I2C_Initialize(100); //Initialize I2C Master with 100KHz clock

    Set_Time_Date(); //set time and date on the RTC module

     //Give an intro message on the LCD
     Lcd_Clear();
     Lcd_Set_Cursor(1,1);
     Lcd_Print_String("DS1307RTC Interf");
     Lcd_Set_Cursor(2,7);
     Lcd_Print_String(" - BGGopal ");
     __delay_ms(1500); //display for 1.5sec

while(1)
  {

    Update_Current_Date_Time(); //Read the current date and time
from RTC module

    //Split the into char to display on lcd
     char sec_0 = sec%10;
     char sec_1 = (sec/10);
     char min_0 = min%10;
     char min_1 = min/10;
     char hour_0 = hour%10;
     char hour_1 = hour/10;
     char date_0 = date%10;
     char date_1 = date/10;
     char month_0 = month%10;
     char month_1 = month/10;
     char year_0 = year%10;
     char year_1 = year/10;
```

```
        //Display the Time on the LCD screen
         Lcd_Clear();
         Lcd_Set_Cursor(1,1);
         Lcd_Print_String(" TIME: ");
         Lcd_Print_Char(hour_1+'0');
         Lcd_Print_Char(hour_0+'0');
         Lcd_Print_Char(':');
         Lcd_Print_Char(min_1+'0');
         Lcd_Print_Char(min_0+'0');
         Lcd_Print_Char(':');
         Lcd_Print_Char(sec_1+'0');
         Lcd_Print_Char(sec_0+'0');
         Lcd_Set_Cursor(2,1);
         Lcd_Print_String("Have A Good  Day");
         __delay_ms(500); //refresh for every 0.5 sec

         //Display the Date on the LCD screen
         day_print(day);
         Lcd_Set_Cursor(2,1);
         Lcd_Print_String("DATE: ");
         Lcd_Print_Char(date_1+'0');
         Lcd_Print_Char(date_0+'0');
         Lcd_Print_Char('-');
         Lcd_Print_Char(month_1+'0');
         Lcd_Print_Char(month_0+'0');
         Lcd_Print_Char('-');
         Lcd_Print_Char(year_1+'0');
         Lcd_Print_Char(year_0+'0');
         __delay_ms(500); //refresh for every 0.5 sec

   }

    return 0;
}

void day_print(unsigned int disp)
{
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Print_String("DAY: ");
    switch(disp)
    {
     case 1:Lcd_Print_String("Sunday");
              break;
     case 2:Lcd_Print_String("Monday");
              break;
     case 3:Lcd_Print_String("Tuesday");
              break;
     case 4:Lcd_Print_String("Wednesday");
              break;
     case 5:Lcd_Print_String("Thursday");
```

```
                break;
       case 6:Lcd_Print_String("Friday");
                break;
       case 7:Lcd_Print_String("Saturday");
                break;
    }
}
```

DS1307.h

```
//PIN 18 -> RC3 -> SCL
//PIN 23 -> RC4 ->SDA


/****** Functions for RTC module *******/

int  BCD_2_DEC(int to_convert)
{
   return (to_convert >> 4) * 10 + (to_convert & 0x0F);
}

int DEC_2_BCD (int to_convert)
{
   return ((to_convert / 10) << 4) + (to_convert % 10);
}

void Set_Time_Date()
{
   I2C_Begin();
   I2C_Write(0xD0);
   I2C_Write(0);
   I2C_Write(DEC_2_BCD(sec)); //update sec
   I2C_Write(DEC_2_BCD(min)); //update min
   I2C_Write(DEC_2_BCD(hour)); //update hour
   I2C_Write(1); //ignore updating day
   I2C_Write(DEC_2_BCD(date)); //update date
   I2C_Write(DEC_2_BCD(month)); //update month
   I2C_Write(DEC_2_BCD(year)); //update year
   I2C_End();
}

void Update_Current_Date_Time()
{
   //START to Read
   I2C_Begin();
   I2C_Write(0xD0);
   I2C_Write(0);
   I2C_End();

  //READ
   I2C_Begin();
```

```
   I2C_Write(0xD1);                              // Initialize data
read
   sec = BCD_2_DEC(I2C_Read(1));
   min = BCD_2_DEC(I2C_Read(1));   // Read sec from register
   hour = BCD_2_DEC(I2C_Read(1));
   I2C_Read(1);
   date = BCD_2_DEC(I2C_Read(1));
   month = BCD_2_DEC(I2C_Read(1));
   year = BCD_2_DEC(I2C_Read(1));
   I2C_End();

  //END Reading
    I2C_Begin();
    I2C_Write(0xD1);                              // Initialize data
read
    I2C_Read(1);
    I2C_End();

}
```

PIC_I2C.h

```
/*
 * File: Header file to use I2C with PIC16F877A
 * Author: Bala
 */

//PIN 18 -> RC3 -> SCL
//PIN 23 -> RC4 ->SDA

void I2C_Initialize(const unsigned long feq_K) //Begin IIC as master
{
  TRISC3 = 1;  TRISC4 = 1;  //Set SDA and SCL pins as input pins

  SSPCON  = 0b00101000;    //pg84/234
  SSPCON2 = 0b00000000;    //pg85/234

  SSPADD  =  (_XTAL_FREQ/(4*feq_K*100))-1;  //Setting  Clock  Speed
pg99/234
  SSPSTAT = 0b00000000;    //pg83/234
}

void I2C_Hold()
{
    while (   (SSPCON2 & 0b00011111)    ||    (SSPSTAT & 0b00000100)
) ; //check the bis on registers to make sure the IIC is not in
progress
}

void I2C_Begin()
```

```
{
  I2C_Hold();  //Hold the program is I2C is busy
  SEN = 1;     //Begin IIC pg85/234
}



void I2C_End()
{
  I2C_Hold(); //Hold the program is I2C is busy
  PEN = 1;    //End IIC pg85/234
}

void I2C_Write(unsigned data)
{
  I2C_Hold(); //Hold the program is I2C is busy
  SSPBUF = data;         //pg82/234
}

unsigned short I2C_Read(unsigned short ack)
{
  unsigned short incoming;
  I2C_Hold();
  RCEN = 1;

  I2C_Hold();
  incoming = SSPBUF;      //get the data saved in SSPBUF

  I2C_Hold();
  ACKDT = (ack)?0:1;    //check if ack bit received
  ACKEN = 1;          //pg 85/234

  return incoming;
}
```

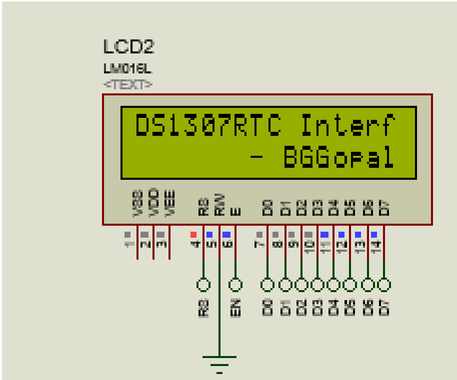
LCD.h

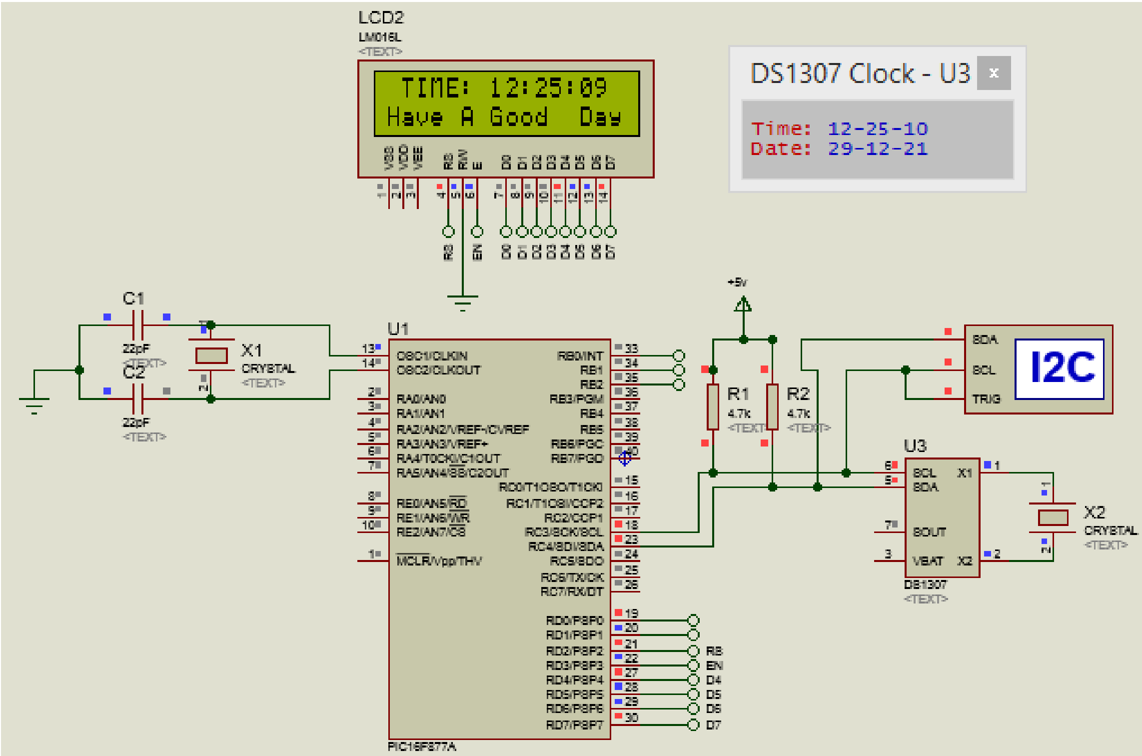
```
/*
 * File: Header file to interface LCD with PIC16F877A
 * Author: Bala
 */

void Lcd_SetBit(char data_bit) //Based on the Hex value Set the Bits
of the Data Lines
{
    if(data_bit& 1)
        D4 = 1;
    else
```

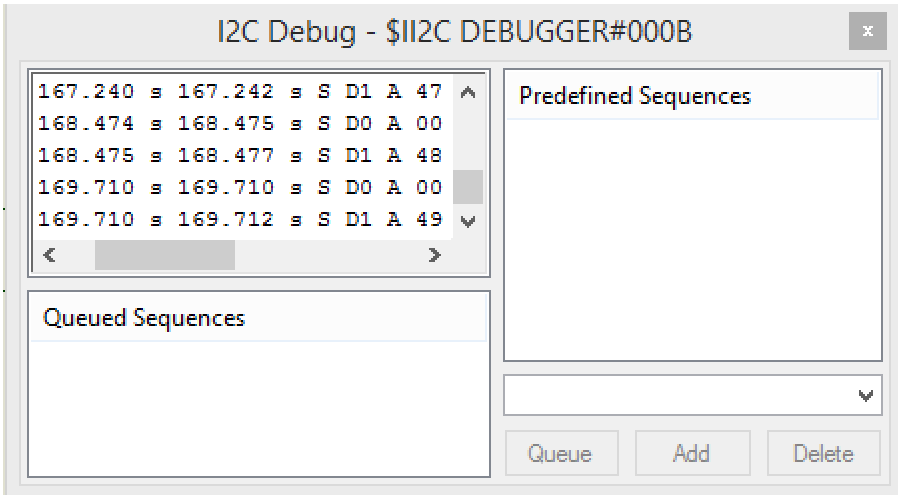
```
        D4 = 0;

    if(data_bit& 2)
        D5 = 1;
    else
        D5 = 0;

    if(data_bit& 4)
        D6 = 1;
    else
        D6 = 0;

    if(data_bit& 8)
        D7 = 1;
    else
        D7 = 0;
}

void Lcd_Cmd(char a)
{
    RS = 0;
    Lcd_SetBit(a); //Incoming Hex value
    EN  = 1;
        __delay_ms(4);
        EN  = 0;
}

Lcd_Clear()
{
    Lcd_Cmd(0); //Clear the LCD
    Lcd_Cmd(1); //Move the curser to first position
}

void Lcd_Set_Cursor(char a, char b)
{
    char temp,z,y;
    if(a== 1)
    {
      temp = 0x80 + b - 1; //80H is used to move the curser
        z = temp>>4; //Lower 8-bits
        y = temp & 0x0F; //Upper 8-bits
        Lcd_Cmd(z); //Set Row
        Lcd_Cmd(y); //Set Column
    }
    else if(a== 2)
    {
        temp = 0xC0 + b - 1;
        z = temp>>4; //Lower 8-bits
```

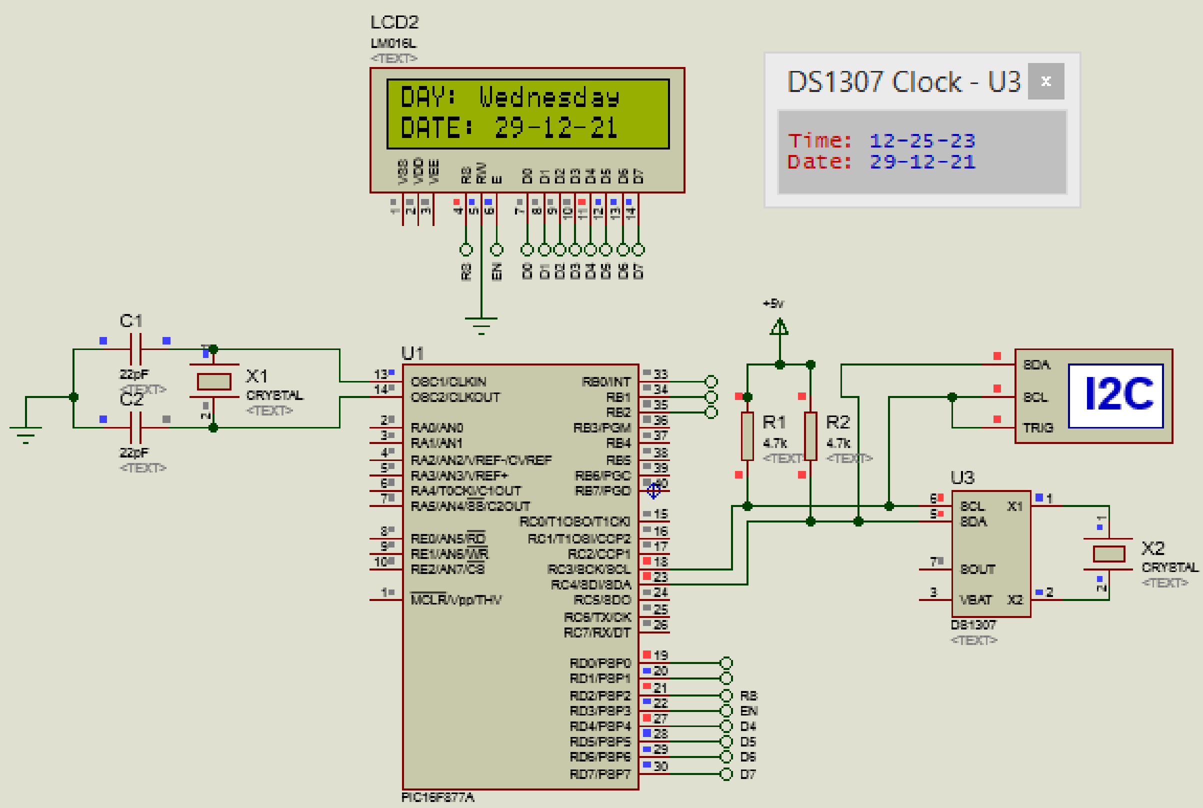
```
        y = temp & 0x0F; //Upper 8-bits
        Lcd_Cmd(z); //Set Row
        Lcd_Cmd(y); //Set Column
    }
}
void Lcd_Start()
{
  Lcd_SetBit(0x00);
  for(int i=1065244; i<=0; i--)  NOP();
  Lcd_Cmd(0x03);
    __delay_ms(5);
  Lcd_Cmd(0x03);
    __delay_ms(11);
  Lcd_Cmd(0x03);
  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and
initializes the LCD
  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and
initializes the LCD
  Lcd_Cmd(0x08); //Select Row 1
  Lcd_Cmd(0x00); //Clear Row 1 Display
  Lcd_Cmd(0x0C); //Select Row 2
  Lcd_Cmd(0x00); //Clear Row 2 Display
  Lcd_Cmd(0x06);
}
void Lcd_Print_Char(char data)  //Send 8-bits through 4-bit mode
{
   char Lower_Nibble,Upper_Nibble;
   Lower_Nibble = data&0x0F;
   Upper_Nibble = data&0xF0;
   RS = 1;             // => RS = 1
   Lcd_SetBit(Upper_Nibble>>4);                 //Send upper half by
shifting by 4
   EN = 1;
   for(int i=2130483; i<=0; i--)  NOP();
   EN = 0;
   Lcd_SetBit(Lower_Nibble); //Send Lower half
   EN = 1;
   for(int i=2130483; i<=0; i--)  NOP();
   EN = 0;
}
void Lcd_Print_String(char *a)
{
    int i;
    for(i=0;a[i]!='\0';i++)
       Lcd_Print_Char(a[i]);  //Split the string using pointers and
call the Char function
}
```

OUTPUT: -


LCD2
LM016L
<TEXT>
DS1307RTC Interf
- BGGopal


LCD2
LM016L
<TEXT>
TIME: 12:25:09
Have A Good Day
DS1307 Clock - U3
Time: 12-25-10
Date: 29-12-21
C1
22pF
C2
22pF
X1
CRYSTAL
U1
PIC16F877A
+5v
R1
4.7k
R2
4.7k
SDA
SCL
TRIG
I2C
U3
DS1307
X2
CRYSTAL


I2C Debug - $II2C DEBUGGER#000B
167.240 s 167.242 s S D1 A 47
168.474 s 168.475 s S D0 A 00
168.475 s 168.477 s S D1 A 48
169.710 s 169.710 s S D0 A 00
169.710 s 169.712 s S D1 A 49
Predefined Sequences
Queued Sequences
Queue
Add
Delete



Result: -

The DS1307 RTC IC was interfaced with PIC16F877A and the time, day and date were displayed connecting a 16x2 LCD.