REAL TIME CLOCK

(R T C)

Project report submitted in partial fulfillment of the requirements

For the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

BY

A.AMALA (07241A0201)

NITHYA RAGHUNATH (07241A0209)

Under the guidance of

Mr. P.M.SARMA

Assistant Professor, Department of EEE

DEPARTMENT OF ELECTRICAL AND ELECTRONICENGINEERING

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING & TECHNOLOGYBACHUPALLY, HYDERABAD-90

2011

Gokaraju Rangaraju Institute of Engineering & Technology

(Affiliated to Jawaharlal Nehru technological University, Hyderabad)

Bachupally, Kukatpally-500090

Hyderabad, Andhra Pradesh

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

C E R T I F I C A T E

This is to certify that the project report entitled **"REAL TIME CLOCK"** that is being submitted by

A.AMALA	**(07241A0201)**
NITHYA RAGHUNATH	**(07241A0209)**

In partial fulfilment for the award of the Degree of **Bachelor of Technology in Electrical and Electronics Engineering** to the Jawaharlal Nehru Technological University is a record of confide work carried out by him under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Mr. P.M.SARMA **Mr. K.SATISH KUMAR**

HOD EEE **EXTERNAL EXAMINER** **ASST. PROF**

GRIET, Hyderabad **GRIET, Hyderabad**

(Internal Guide)

INDEX

1. INTRODUCTION

1.1 THE REAL-TIME CLOCK

A real-time clock (RTC) is a computer clock (most often in the form of an integrated circuit) that keeps track of the current time. Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in almost any electronic device which needs to keep accurate time.

A real time clock (RTC) is a timepiece module having an independent battery for operation and has a backup RAM always supplied with electric power from the battery. Many data processing circuits utilize real-time clocks to provide a real-time clock value indicating, for example, the current day, date and time.

Typically, when the data processing circuit is first activated, the correct day, date and time may need to be set. When the data processing circuit is shut down, power is maintained to the real-time clock by a battery, so that the real-time clock may continue to operate. Personal computers and digital electronics typically include a real time clock that tracks the time of day, day, month, and year. The use of wireless communication systems continues to expand.

In some digital cellular systems, the in-phone real-time clock is synchronized to the precise local time given by the base station to which the phone is communicating with currently. This feature is known as automatic local time adjustment. To implement automated and programmable functions, a real time clock within the wireless device is generally required. The real time clock provides a precise representation of time which can be compared to stored values in registers to provide automated functions. During power interruptions, the real time clock is typically powered from either a dedicated clock battery, such as a small button cell, or from a storage capacitor.

On a computer mother board, a RTC is provided to maintain the data stored in a CMOS on the mother board. Generally, electrical power is provided to a RTC by a battery arranged on the mother board. The RTC is used to update the current time and date within the computer system without any intervention from the system processor. Usually the RTC and other peripheral devices are coupled to the peripheral bus through an interrupt controller. Such a clock has various uses such as time stamping files and inserting dates into documents.

The RTC is used to update a program clock, typically a date/time clock maintained by the operating system (OS) and referred to herein as the OS clock. An operating system may include a system clock to provide a system time for measuring small increments of time. The system clock may update the system clock in response to a periodic interrupt generated by a system timer, or a real time clock event timer.

A Real-Time-Clock (RTC) is, as the name suggests, a clock which keeps track of time in a "real mode." While there are a number of 8051-compatible microcontrollers that have built-in, accurate real-time clocks (especially from Dallas Semiconductor), some simple applications may benefit from a software RTC solution that uses the built-in capabilities of an 8051 microcontroller.

1.2 BENEFITS

The purpose of using real-time clocks has the following benefits:

- Low power consumption (important when running from alternate power)
- Frees the main system for time-critical tasks
- Sometimes more accurate than other methods

1.3 POWER SOURCE

RTCs often have an alternate source of power, so they can continue to keep time while the primary source of power is off or unavailable. This alternate source of power is normally a lithium battery in older systems, but some newer systems use a supercapacitor, because they are rechargeable and can be soldered. The alternate power source can also supply power to battery backed RAM.

1.4 TIMING

Most RTCs use a crystal oscillator, but some use the power line frequency. In many cases the oscillator's frequency is 32.768 kHz. This is the same frequency used in quartz clocks and watches, and for the same reasons, namely that the frequency is exactly 215 cycles per second, which is a convenient rate to use with simple binary counter circuits.

1.5 USAGE

A real-time clock (RTC) is a battery-powered clock that is included in a microchip in a computer motherboard. This microchip is usually separate from the microprocessor and other chips and is often referred to simply as "the CMOS" (complementary metal-oxide semiconductor). A small memory on this microchip stores system description or setup values - including current time values stored by the real-time clock. The time values are for the year, month, date, hours, minutes, and seconds. When the computer is turned on, the Basic Input-Output Operating System (BIOS) that is stored in the computer's read-only memory (ROM) microchip reads the current time from the memory in the chip with the real-time clock.

Many integrated circuit manufacturers make RTCs, including Intersil, Maxim, Philips, Texas Instruments and STMicroelectronics. The RTC was introduced to PC compatibles by the IBM PC/AT in 1984, which used a MC146818 RTC. Later Dallas made compatible RTCs, which was often used in older personal computers, and are easily found on motherboards because of their distinctive black battery cap and silkscreened logo. In newer systems the RTC is integrated into the southbridge chip.

Popular serial clock circuit is numerous now, such as DS1302, DS1307, PCF8485,etc.. The interface of the circuit is simple, cheap, easy to use, widely adopted. The real-time clock circuit DS1302 is that one kind of DALLAS Company has circuit that detailed electric current of tiny stream charges ability, the leading particulars adopts the serial data transmission, can offer the programming charging function for losing power of electric protection, and can close the charging function.

Structure and operating principle of 2 DS1302

DS1302 is a kind of high performance which U.S.A. DALLAS Company introduces, low power consumption, real-time clock circuit with RAM, it can for day, month and year, Sunday, when, go on minute or second time, have leap year to compensate the function, working voltage is 2.5V- 5.5V. Adopt three line interfaces and CPU to carry on synchronous communication, a clock signal or RAM datum of conveying a plurality of byte that and can adopt burst way. There are RAM registers used for preserving the data in temporariness of 31* 8 for one within DS1302. DS1302 is the upgrading products of DS1202, compatible with DS1202, but increased the power duplicate supply pin of the main power source / back, offered and carried on the charged ability of detailed electric current of tiny stream to the power of back at the same time.

This is one kind of RTC that is usually used.

Some microcontrollers have a real-time clock built in, generally only the ones with many other features and peripherals.

The complete operation of the DS1307 RTC and its use in controlling home appliances is described in the other chapters.

2. DESCRIPTION

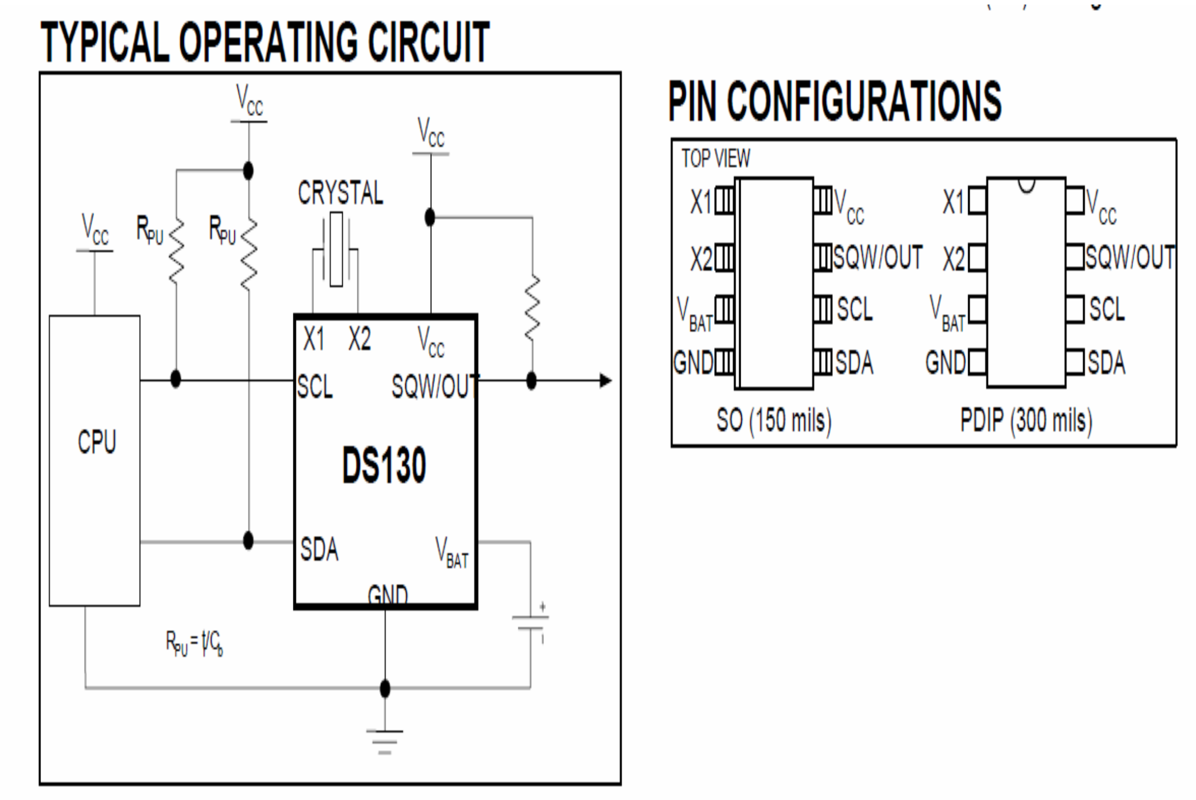
2.1 GENERAL DESCRIPTION

The DS1307 serial real-time clock (RTC) is a low power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM.

Address and data are transferred serially through an I^2C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

2.2 FEATURES

- Real-Time Clock (RTC) Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the week, and Year with Leap-Year Compensation Valid Up to 2100
- 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes
- I^2C Serial Interface
- Programmable Square-Wave Output Signal
- Automatic Power-Fail Detect and Switch Circuitry
- Consumes Less than 500nA in Battery-Backup Mode with Oscillator Running
- Optional Industrial Temperature Range: -40°C to +85°C
- Available in 8-Pin Plastic DIP or SO
- Underwriters Laboratories (UL) Recognized



The typical operating circuit and pin configurations of DS1307 is shown in the figure above

2.3 ABSOLUTE MAXIMUM RATINGS

Voltage Range on Any Pin Relative to Ground ....-0.5V to +7.0V

Operating Temperature Range (Noncondensing)

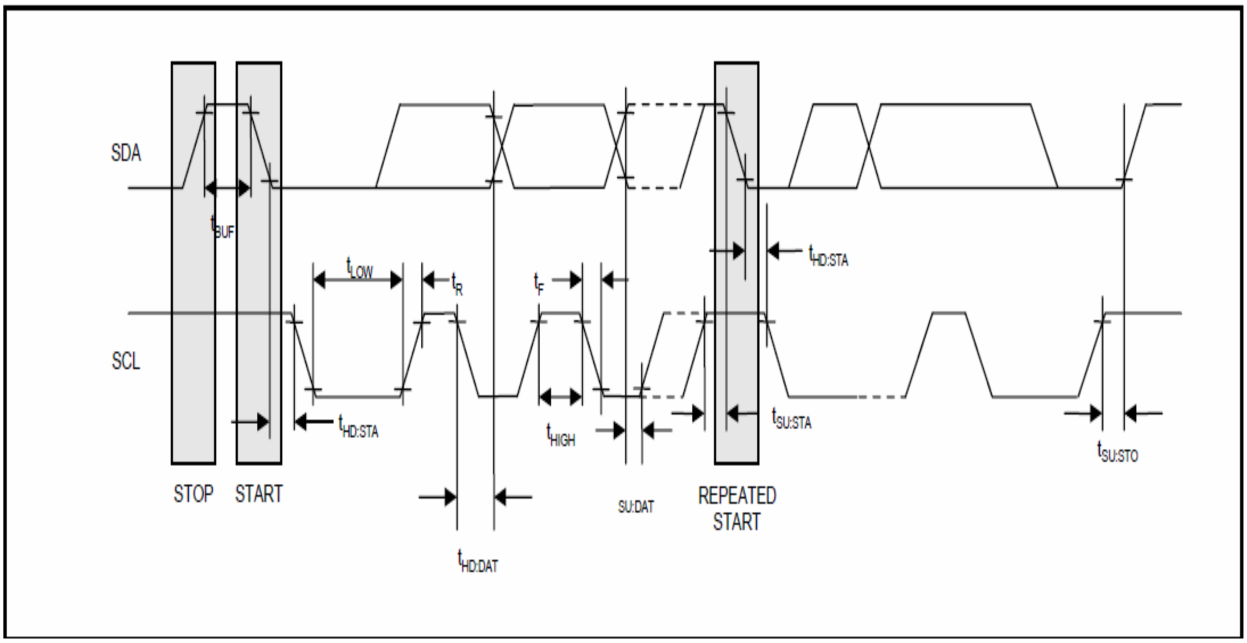
Commercial...................0°C to +70°C

Industrial ....-40°C to +85°C

Storage Temperature Range............. -55°C to +125°C

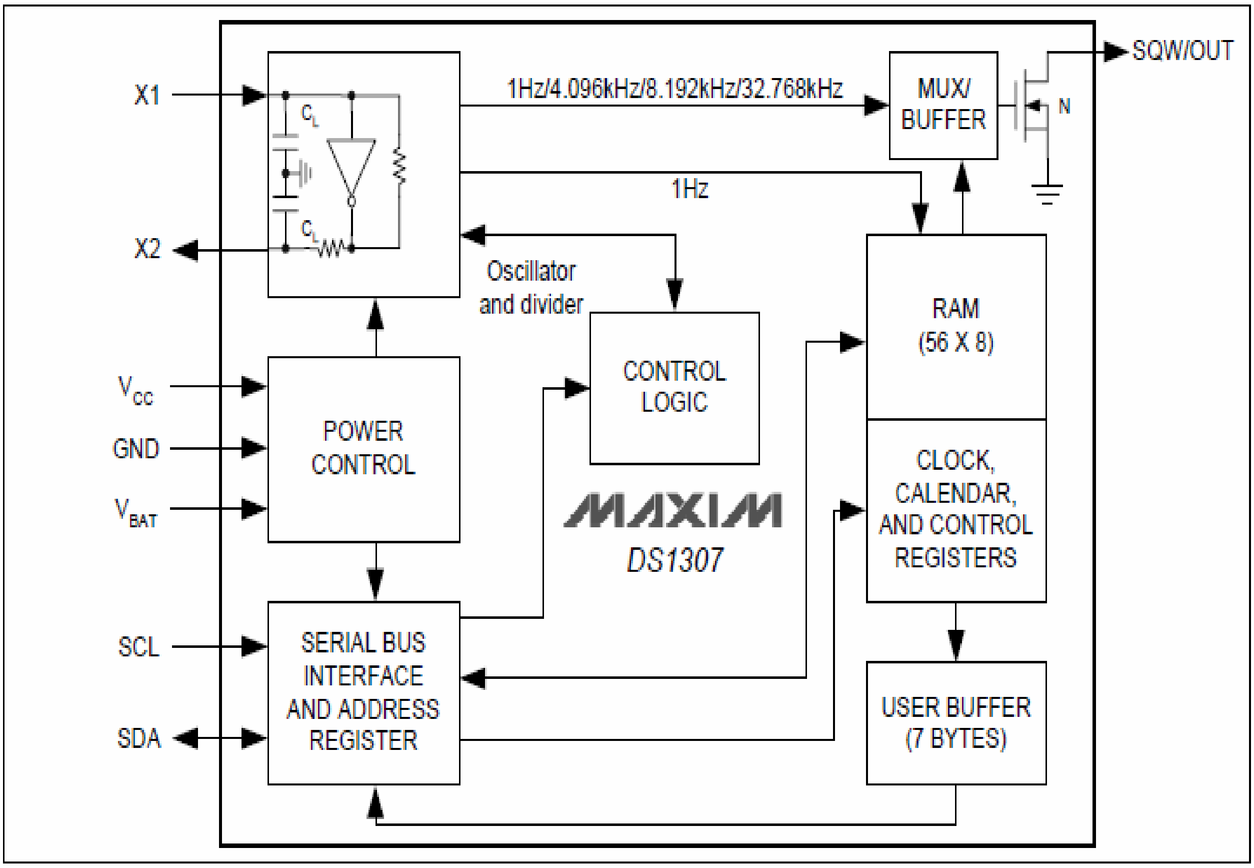
Soldering Temperature (DIP,leads)......................+260°C for 10 seconds

TIMING DIAGRAM



The figure above shows the timing diagram of DS1307

BLOCK DIAGRAM OF DS1307

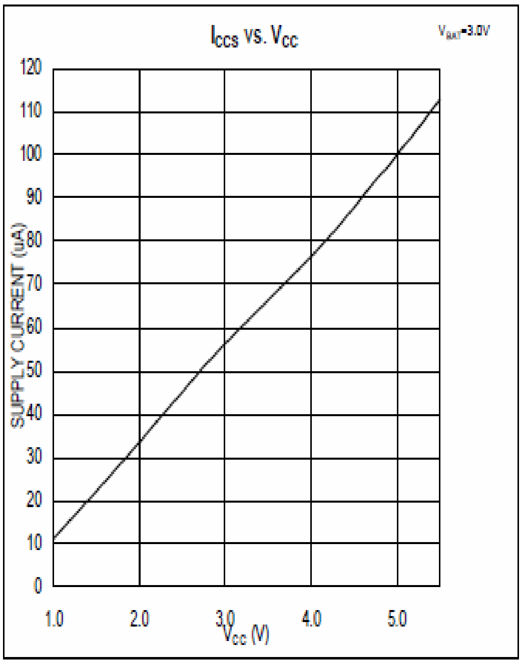


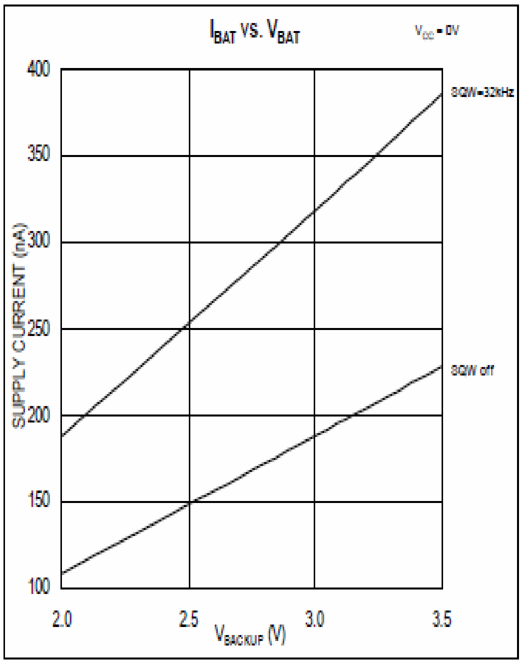
The figure above represents the block diagram of DS1307

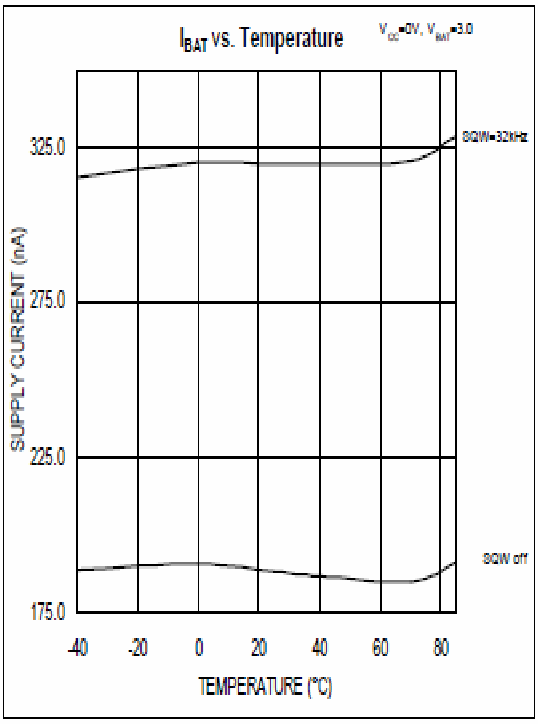
2.4 OPERATING CHARACTERISTICS

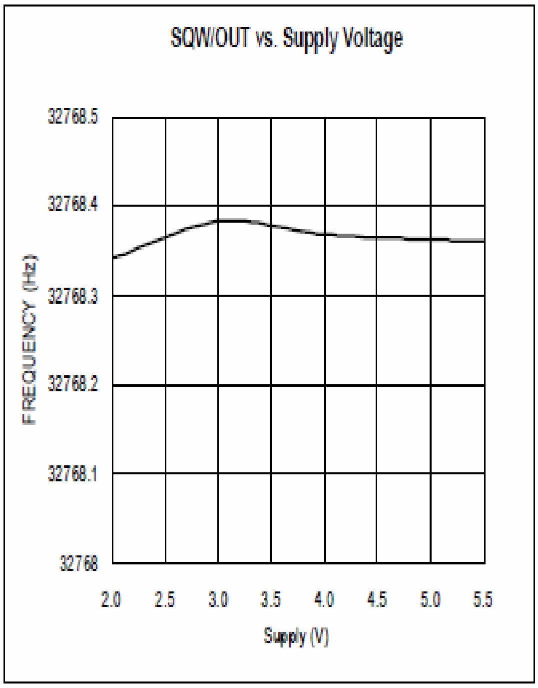
TYPICAL OPERATING CHARACTERISTICS

(V_{CC} = 5.0V, T_A = +25°C, unless otherwise noted.)









2.5 PIN DESCRIPTION

PIN	NAME	FUNCTION
1	X1	Connections for Standard 32.768kHz Quartz Crystal. The internal oscillator circuitry is designed for operation with a crystal having a specified load capacitance (CL) of 12.5pF. X1 is the input to the oscillator and can optionally be connected to an external 32.768kHz oscillator. The output of the internal oscillator, X2, is floated if an external oscillator is connected to X1.
2	X2	The output of the internal oscillator, X2, is floated if an external oscillator is connected to X1.
3	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.Diodes in series between the battery and the V_{BAT} pin may prevent proper operation. If a backup supply is not required, V_{BAT} must be grounded. The nominal power-fail trip point (VPF) voltage at which access to the RTC and user RAM is denied is set by the internal circuitry as 1.25 x VBAT nominal. 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.
4	GND	Ground
5	SDA	Serial Data Input/Output. SDA is the data input/output for the I^2C serial interface. The SDA pin is open drain and requires an external pullup resistor. The pullup voltage can be up to 5.5V regardless of the voltage on V_{CC}.
6	SCL	Serial Clock Input. SCL is the clock input for the I^2C interface and is used to synchronize data movement on the serial interface. The pullup voltage can be up to 5.5V regardless of the voltage on V_{CC}
7	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). The SQW/OUT pin is open drain and requires an external pullup resistor. SQW/OUT operates with either V_{CC} or V_{BAT}

		applied. The pullup voltage can be up to 5.5V regardless of the voltage on V_{CC}. If not used, this pin can be left floating.
8	V_{CC}	Primary Power Supply. When voltage is applied within normal limits, the device is fully accessible and data can be written and read. When a backup supply is connected to the device and V_{CC} is below V_{TP}, read and writes are inhibited. However, the timekeeping function continues unaffected by the lower input voltage.

2.6 DETAILED DESCRIPTION

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 I^2C 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. When V_{CC} falls below 1.25 x V_{BAT}, the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out-of-tolerance system. When V_{CC} falls below V_{BAT}, the device switches into a lowcurrent battery-backup mode. Upon power-up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT} +0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}.

2.7 OSCILLATOR CIRCUIT

The DS1307 uses an external 32.768kHz crystal. The oscillator circuit does not require any external resistors or capacitors to operate. Table 1 specifies several crystal parameters for the external crystal. Figure shows a functional schematic of the oscillator circuit. If using a crystal with the specified characteristics, the startup time is usually less than one second.

2.8 CLOCK ACCURACY

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. External circuit noise coupled into the oscillator circuit may result in the clock running fast.

2.9 CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. Table 2 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. The day-of-week register increments at midnight. Values that correspond to the day of week are user-defined but must be sequential (i.e., if 1 equals Sunday, then 2 equals Monday, and so on.) Illogical time and date entries result in undefined operation. Bit 7 of Register 0 is the clock halt (CH) bit. When this bit is set to 1, the oscillator is disabled. When cleared to 0, the oscillator is enabled. 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). The CH bit in the seconds register will be set to a 1. The clock can be halted whenever the timekeeping functions are not required, which minimizes current (IBATDR).

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours 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 hours value must be re-entered whenever the 12/24-hour mode bit is changed.

When reading or writing the time and date registers, secondary (user) buffers are used to prevent errors when the internal registers update. When reading the time and date registers, the user buffers are synchronized to the internal registers on any I^2C START. The time information is read from these secondary registers while the clock continues to run. This eliminates the need to re-read the registers in case the internal registers update during a read. The divider chain is reset whenever the seconds register is written. Write transfers occur on the I^2C acknowledge from the DS1307. Once the divider chain is reset, to avoid rollover issues, the remaining time and date registers must be written within one second.

Table 2. Timekeeper Registers

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

0 = Always reads back as 0.

2.10 CONTROL REGISTER

The DS1307 control register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

Bit 7: Output Control (OUT). This bit controls the output level of the SQW/OUT pin when the square-wave output is disabled. If SQWE = 0, the logic level on the SQW/OUT pin is 1 if OUT = 1 and is 0 if OUT = 0. On initial application of power to the device, this bit is typically set to a 0.

Bit 4: Square-Wave Enable (SQWE). This bit, when set to logic 1, enables the oscillator output. The frequency of the square-wave output depends upon the value of the RS0 and RS1 bits. With the square-wave output set to 1Hz, the clock registers update on the falling edge of the square wave. On initial application of power to the device, this bit is typically set to a 0.

Bits 1 and 0: Rate Select (RS[1:0]). These bits control the frequency of the square-wave output when the squarewave output has been enabled. The following table lists the square-wave frequencies that can be selected with the RS bits. On initial application of power to the device, these bits are typically set to a 1.

RS1	RS0	SQW/OUT OUTPUT	SQWE	OUT
0	0	1Hz	1	X
0	1	4.096kHz	1	X
1	0	8.192kHz	1	X
1	1	32.768kHz	1	X
X	X	0	0	0
X	X	1	0	1

2.11 I^{2}C DATA BUS

The DS1307 supports the I^{2}C protocol. A device that sends data onto the bus is defined as a transmitter and a device receiving data as a receiver. The device that controls the message is called a master. The devices that are controlled by the master are referred to as slaves. The bus must be controlled by a master device that generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS1307 operates as a slave on the I^{2}C bus. Figures 3, 4, and 5 detail how data is transferred on the I^{2}C bus.

- Data transfer can be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the clock line is HIGH. Changes in the data line while the clock line is high will be interpreted as control signals.

Accordingly, the following bus conditions have been defined:

Bus not busy: Both data and clock lines remain HIGH.

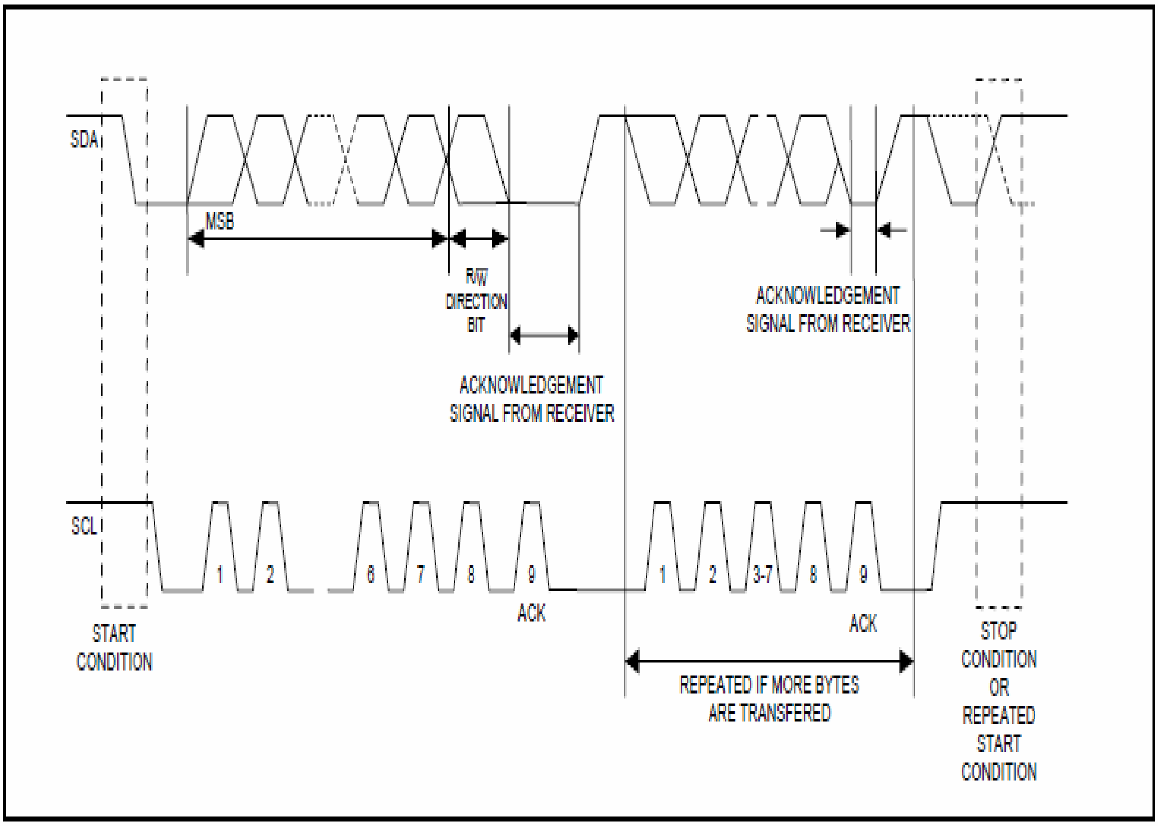
START data transfer: A change in the state of the data line, from HIGH to LOW, while the clock is HIGH, defines a START condition.

STOP data transfer: A change in the state of the data line, from LOW to HIGH, while the clock line is HIGH, defines the STOP condition.

Data valid: The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data. Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between START and STOP conditions is not limited, and is determined by the master device. The information is transferred byte-wise and each receiver acknowledges with a ninth bit. Within the I2C bus specifications a standard mode (100kHz clock rate) and a fast mode (400kHz clock rate) are defined.
The DS1307 operates in the standard mode (100kHz) only.

Acknowledge: Each receiving device, when addressed, is obliged to generate an acknowledge after the reception of each byte. The master device must generate an extra clock pulse which is associated with this acknowledge bit. A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse. Of course, setup and hold times must be taken into account. A master must signal an end of data to the slave by not generating an acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave must leave the data line HIGH to enable the master to generate the STOP condition.

Figure 3. Data Transfer on I^{2}C Serial Bus



Depending upon the state of the R/W bit, two types of data transfer are possible:

1. **Data transfer from a master transmitter to a slave receiver.** The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte. Data is transferred with the most significant bit (MSB) first.

2. **Data transfer from a slave transmitter to a master receiver.** The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. This is followed by the slave transmitting a number of data bytes. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a "not acknowledge" is returned.

The master device generates all the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the bus will not be released. Data is transferred with the most significant bit (MSB) first.

3. OPERATION

The DS1307 can operate in the following two modes:

1. **Slave Receiver Mode (Write Mode):** Serial data and clock are received through SDA and SCL. After each byte is received an acknowledge bit is transmitted. START and STOP conditions are recognized as the beginning and end of a serial transfer. Hardware performs address recognition after reception of the slave address and direction bit (see Figure 4). The slave address byte is the first byte received after the master generates the START condition. The slave address byte contains the 7-bit DS1307 address, which is 1101000, followed by the direction bit (R/W), which for a write is 0. After receiving and decoding the slave address byte, the DS1307 outputs an acknowledge on SDA. After the DS1307 acknowledges the slave address + write bit, the master transmits a word address to the DS1307. This sets the register pointer on the DS1307, with the DS1307 acknowledging the transfer. The master can then transmit zero or more bytes of data with the DS1307 acknowledging each byte received. The register pointer automatically increments after each data byte are written. The master will generate a STOP condition to terminate the data write.

2. **Slave Transmitter Mode (Read Mode):** The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will indicate that the transfer direction is reversed. The DS1307 transmits serial data on SDA while the serial clock is input on SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer (see Figure 5). The slave address byte is the first byte received after the START condition is generated by the master. The slave address byte contains the 7-bit DS1307 address, which is 1101000, followed by the direction bit (R/W), which is 1 for a read. After receiving and decoding the slave address the DS1307 outputs an acknowledge on SDA. The DS1307 then begins to transmit data starting with the register address pointed to by the register pointer. If the register pointer is not written to before the initiation of a read mode the first address that is read is the last one stored in the register pointer. The register pointer automatically increments after each byte are read. The DS1307 must receive a Not Acknowledge to end a read.

Figure 4. Data Write—Slave Receiver Mode

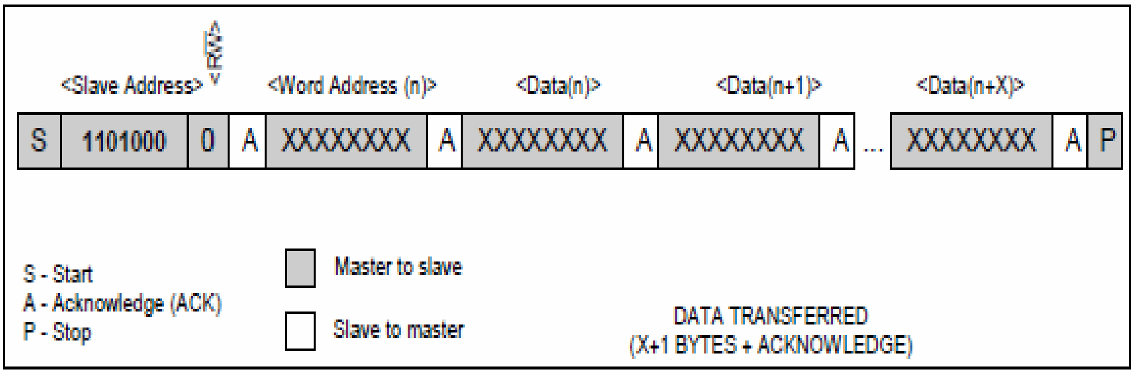


Figure 5. Data Read—Slave Transmitter Mode

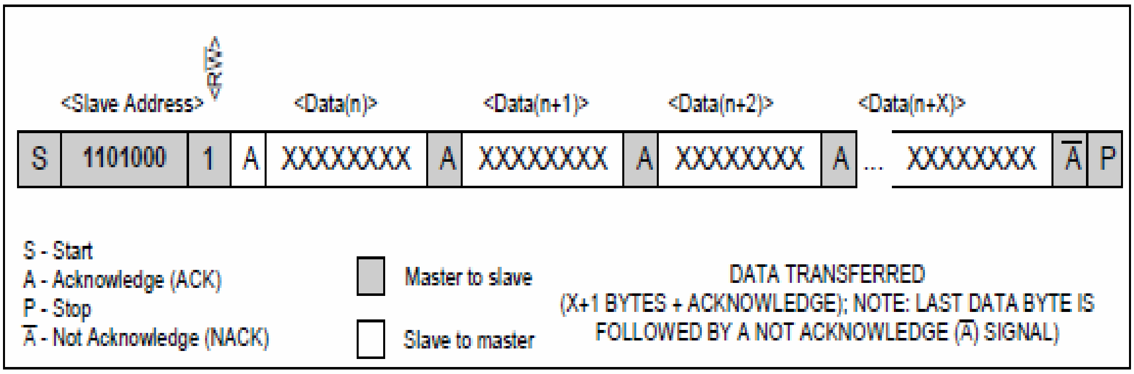
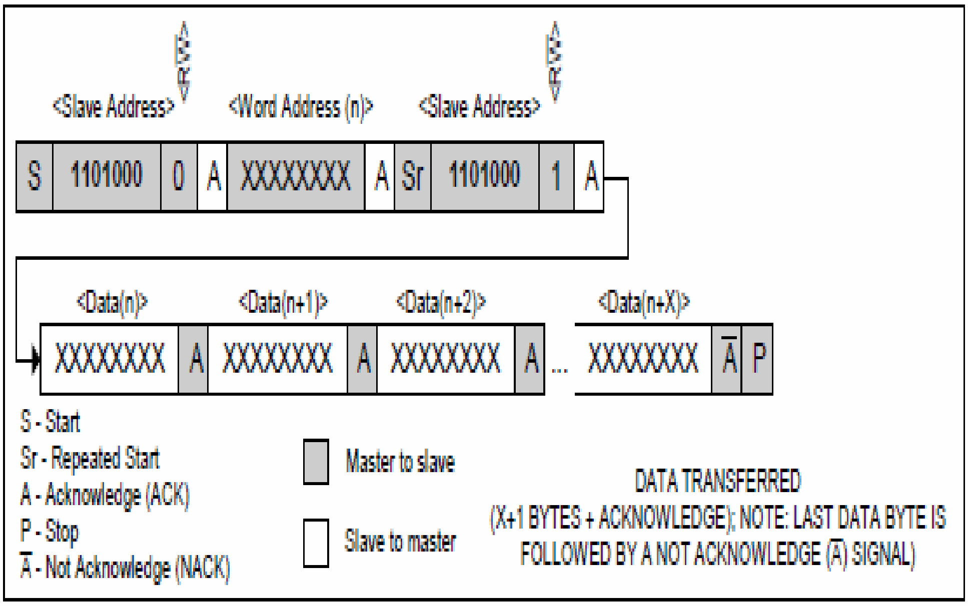


Figure 6. Data Read (Write Pointer, Then Read)—Slave Receive and Transmit

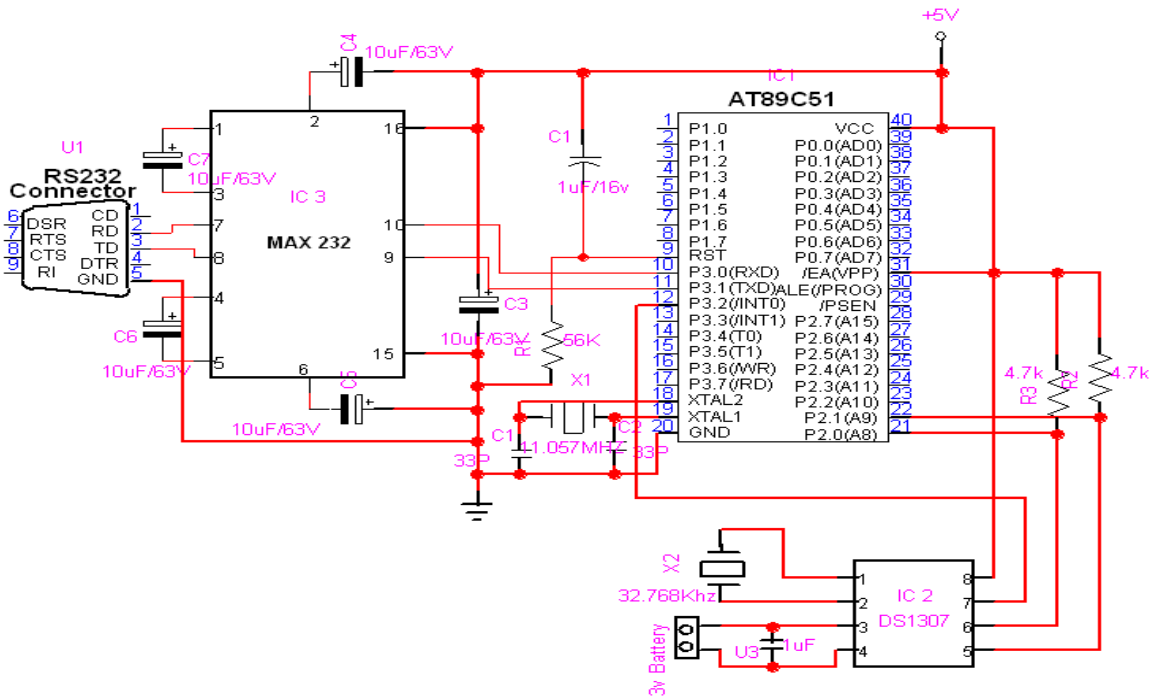


4. REAL-TIME CLOCK INTERFACING (DS1307) WITH AT89C51

4.1 DESCRIPTION

This application note describes the general hardware configuration and basic software communication examples for the Dallas I2C serial-interface Real-Time Clocks (RTC).The devices covered are the BCD-format I2C clocks: DS1307, DS1337, DS1338, DS1339 and DS1340.The DS1375 could also be supported, if circuit modifications were made to provide digital clock signal (32,768Hz, 8,192Hz, 60H, or 50Hz) to the CLK input pin.The microcontroller used for this example is the DS2250, and example software is written in C.

A schematic of the circuit is shown in Figure 1. The schematic shows connections for a DS1340. The other RTCs may require modifications. The DS1337, for example, replaces the battery back up input with an additional interrupt output. The low voltage versions of the RTCs would require replacing the DS2250/DS5000 with a suitable low-voltage microcontroller. Figure 2 shows the software listing. The #define directive is used to conditionally compile the code for the proper device. The example shown is for the DS1307. The #define statement for the DS1307 should be replaced with the correct device before compiling the code




```
/************************************************************************/
/* DEMO1307.c                                              */
/************************************************************************/
#include <at89x52.h>          /* Prototypes for I/O functions */
#include <stdio.h>
/****************************** Defines ****************************/
#define    ACK      0
#define    NACK     1
#define    ADDRTC   0xd0    /* I2C slave address */
#define    DS1307   /* compile directive, modify as required */

/************************* bit definitions *************************/
sbit scl = P2^0;         /* I2C pin definitions */
sbit sda = P2^1;
sbit sqw = P3^2;         /* pin function depends upon device */

void    I2C_start();
void    I2C_stop();
void    I2C_write(unsigned char d);
uchar   I2C_read(uchar);
void    readbyte();
void    writebyte();
void    initialize();
```

```
void        disp_clk_regs(uchar);
void        burstramwrite(uchar);
void        burstramread();
void        alrm_int();
void        alrm_read();
void        tc_setup();

/* global variables */
uchar       sec, min, hr, dy, dt, mn, yr;
void I2C_start()        /* ----------------------------------------------*/
{
       sda = 1;  scl = 1;        /* Initiate start condition */
       sda = 0;
}
void I2C_stop()                /* ----------------------------------------*/
{
       sda = 0; sda = 0; sda = 0; sda = 0;     /* Initiate stop condition*/
       scl = 1; scl = 1; sda = 1;
}
void I2C_write(uchar d)                /* ----------------------------- */
{
uchar i;

       scl = 0;
       for (i = 1; i <= 8; i++)
       {
              sda = (d >> 7);
              scl = 1;
              d = d << 1;        /* increase scl high time */
              scl = 0;
       }
       sda = 1;        /* Release the sda line */
       scl = 0;
       scl = 1;
       if(sda) printf("Ack bit missing  %02X",(unsigned int)d);
       scl = 0;
}
uchar I2C_read(uchar b)        /* ----------------------------------- */
{
uchar d, i;
```

```
        sda = 1;            /* Let go of sda line */
        scl = 0;
        for (i = 1; i <= 8; i++)        /* read the msb first */
        {
                scl = 1;
                d = d << 1;
                d = d | (unsigned char)sda;
                scl = 0;
        }
        sda = b;          /* Hold sda low for acknowledge */

        scl = 0;
        scl = 1;
        if(b == NACK)        sda = 1;   /* sda = 1 if next cycle is reset */
        scl = 0;
        sda = 1;          /* Release the sda line */
        return d;
}
void readbyte() /* -- read one byte of data from the specific address -- */
{
uchar Add;
        printf("ADDRESS: ");        /* Get Address */
        scanf("%bx", &Add);
        I2C_start();
        I2C_write(ADDRTC);
        I2C_write(Add);
        I2C_start();
        I2C_write(ADDRTC | 1);
        printf("%2bx", I2C_read(NACK) );
        I2C_stop();
}
void writebyte()/* -- write one byte of data to the specified address -- */
{
uchar Add;
uchar Data;
        printf("Address: ");        /* Get Address */
        scanf("%bx", &Add);
        printf("DATA: ");
        scanf("%bx", &Data);        /* and data */
```

```
    I2C_start();
    I2C_write(ADDRTC);
    I2C_write(Add);
    I2C_write(Data);
    I2C_stop();
}
void initialize() /*initialize the time and date using entries from stdin*/
/* Note: NO error checking is done on the user entries! */
{
uchar       yr, mn, dt, dy, hr, min, sec, day;

    I2C_start();            /* The following Enables the Oscillator */
    I2C_write(ADDRTC);      /* address the part to write */
    I2C_write(0x00);        /* position the address pointer to 0 */
    I2C_write(0x00);  /* write 0 to the seconds register,clear the CH bit */
    I2C_stop();
    printf("Enter the year (0-99): ");
    scanf("%bx", &yr);
    printf("Enter the month (1-12): ");
    scanf("%bx", &mn);
    printf("Enter the date (1-31): ");
    scanf("%bx", &dt);
    printf("Enter the day (1-7): ");
    scanf("%bx", &dy);
    printf("Enter the hour (1-23): ");
    scanf("%bx", &hr);
    hr = hr & 0x3f;         /* force clock to 24 hour mode */
    printf("Enter the minute (0-59): ");
    scanf("%bx", &min);
    printf("Enter the second (0-59): ");
    scanf("%bx", &sec);

    I2C_start();
    I2C_write(ADDRTC);      /* write slave address + write */
    I2C_write(0x00);        /* write register address, 1st clock register */
    I2C_write(sec);
    I2C_write(min);
    I2C_write(hr);
    I2C_write(dy);
    I2C_write(dt);
```

```
	I2C_write(mn);
	I2C_write(yr);
#if defined DS1307 || defined DS1338
{
	I2C_write(0x10);	/* enable sqwe, 1Hz output */
}
#elif defined DS1337 || defined DS1339
{
	I2C_start();
	I2C_write(ADDRTC);	/* write slave address + write */
	I2C_write(0x0e);	/* write register address, control register */
	I2C_write(0x20);	/* enable osc, bbsqi */
	I2C_write(0);		/* clear OSF, alarm flags */
				/* could enable trickle charger here */
}
#elif defined DS1340
	I2C_write(0x10);	/* enable sqwe, 1Hz output */
	I2C_start();		/* address pointer wraps at 7, so point to flag register */
	I2C_write(ADDRTC);	/* write slave address + write */
	I2C_write(0x09);	/* write register address, control register */
	I2C_write(0);		/* clear OSF */
}
#endif

	I2C_stop();

}
void	disp_clk_regs(uchar prv_sec)	/* ---------------------------------------- */
{
uchar Sec, Min, Hrs, Dte, Mon, Day, Yr, mil, pm;

	printf("Yr Mn Dt Dy Hr:Mn:Sc");

	while(!RI)	/* Read & Display Clock Registers */
	{
		I2C_start();
		I2C_write(ADDRTC);	/* write slave address + write */
		I2C_write(0x00);	/* write register address, 1st clock register */
		I2C_start();
		I2C_write(ADDRTC | 1);	/* write slave address + read */
```

```
            Sec = I2C_read(ACK);        /* starts w/last address stored in register pointer */
            Min = I2C_read(ACK);
            Hrs = I2C_read(ACK);
            Day = I2C_read(ACK);
            Dte = I2C_read(ACK);
            Mon = I2C_read(ACK);
            Yr  = I2C_read(NACK);
            I2C_stop();
            if(Hrs & 0x40)
                  mil = 0;
            else
                  mil = 1;
            if(Sec != prv_sec)       /* display every time seconds change */
            {
                  if(mil)
                  {
                        printf("%02bX/%02bX/%02bX %2bX", Yr, Mon, Dte, Day);
                        printf(" %02bX:%02bX:%02bX", Hrs, Min, Sec);
                  }
                  else
                  {
                        if(Hrs & 0x20)
                              pm = 'A';
                        else
                              pm = 'P';
                        Hrs &= 0x1f;        /* strip mode and am/pm bits */
                        printf("%02bx/%02bx/%02bx %02bx", Yr, (Mon & 0x1f), Dte, Day);
                        printf(" %02bx:%02bx:%02bx %cM", Hrs, Min, Sec, pm);
                  }
            }
            if(prv_sec == 0xfe)       return;
            prv_sec = Sec;
      }
      RI = 0;  /* Swallow keypress before exiting */
}
void burstramwrite(uchar Data)        /* -------- fill RAM with data -------- */
{
uchar j;

   I2C_start();
```

```
    I2C_write(ADDRTC);      /* write slave address + write */
    I2C_write(0x08);        /* write register address, 1st RAM location */
    for (j = 0; j < 56; j++)        /* write until the pointer wraps around */
    {
        I2C_write(Data);
    }
    I2C_stop();
}
void burstramread()             /* ----------------------------------------- */
{
uchar j;

    I2C_start();
    I2C_write(ADDRTC);      /* write slave address + write */
    I2C_write(8);       /* write register address, 1st RAM location -1*/
    I2C_start();
    I2C_write(ADDRTC | 1);      /* write slave address + read */

    for (j = 0; j < 56; j++)
    {
        if(!(j % 16)) printf("%02bX ", j);
        printf("%02bX ", I2C_read(ACK) );
    }
    I2C_read(NACK);
    I2C_stop();
}
void    alrm_int()      /* ----- initialize alarm registers ------ */
{
uchar   M, Sec, Min, Hr, DyDt;

    printf("1-Alarm each second     2-Alarm match=sec 3-Alarm match=sec+min");
    printf("4-Alarm match=sec+min+hr 5-Alarm match=sec+min+hr+date");
    printf("6-Alarm match=sec+min+hr+day        Enter selection: ");

    M = _getkey();      /* Note-No error checking is done on entries! */

    switch(M)
    {
        case '1':   M = 0xf;    break;
        case '2':   M = 0xe;    break;
```

```
    case '3':    M = 0xc;    break;
    case '4':    M = 8;      break;
    case '5':    M = 0;      break;
    case '6':    M = 0x40;   break;
}
if(M & 0x40)
{
    printf("Enter the day (1-7): ");
    scanf("%bx", &DyDt);
}
else
{
    printf("Enter the date (1-31): ");
    scanf("%bx", &DyDt);
}
printf("Enter the hour (1-23): ");
scanf("%bx", &Hr);
printf("Enter the minute (0-59): ");
scanf("%bx", &Min);
printf("Enter the second (0-59): ");
scanf("%bx", &Sec);

if( (M & 1) )    Sec |= 0x80;
if( ((M >> 1) & 1) )    Min |= 0x80;
if( ((M >> 2) & 1) )    Hr |= 0x80;
if( ((M >> 3) & 1) )    DyDt |= 0x80;
if(M & 0x40)    DyDt |= 0x40;

I2C_start();
I2C_write(ADDRTC);    /* write slave address + write */
I2C_write(7);         /* write register address */
I2C_write(Sec);
I2C_write(Min);
I2C_write(Hr);
I2C_write(DyDt);
I2C_start();
I2C_write(ADDRTC);    /* write slave address + write */
I2C_write(0x0e);      /* write register address */
I2C_write(5);         /* enable interrupts, alarm 1 */
I2C_stop();
```

```
}
void    alrm_read()    /* ----- read alarm registers ------ */
{
uchar    Sec, Min, Hr, DyDt;

    I2C_start();
    I2C_write(ADDRTC);    /* write slave address + write */
    I2C_write(7);         /* write register address */
    I2C_start();
    I2C_write(ADDRTC | 1);    /* write slave address + read */
    Sec = I2C_read(ACK);
    Min = I2C_read(ACK);
    Hr = I2C_read(ACK);
    DyDt = I2C_read(NACK);

    printf("Alarm 1: %02bx %02bx %02bx %02bx", Sec, Min, Hr, DyDt);
}
void    tc_setup()    /* ---- trickle charger set up routine ---- */
{
uchar    M, val;

#if defined DS1339
    #define    TC 0x10    /* address for DS1339 trickle charge register */
#else
    #define    TC 0x08    /* address for DS1340 trickle charge register */
#endif

    printf("Enable Trickle Charger (Y/N)? ");
    M = _getkey();

    if(M == 'Y' || M == 'y')
    {
        printf("1-250 ohm res2-2K res=sec3-4K res");

        M = _getkey();    /* Note-No error checking is done on entries! */

        switch(M)
        {
            case '1':    val = 1;    break;
            case '2':    val = 2;    break;
```

```
            case '3':     val = 3;     break;
        }
        printf("1-no diode2-1 diode");

        M = _getkey();     /* Note-No error checking is done on entries! */

        switch(M)
        {
            case '1':     val += 4;     break;
            case '2':     val += 8;     break;
        }
        I2C_start();
        I2C_write(ADDRTC);          /* write slave address + write */
        I2C_write(TC);          /* write register address */
        I2C_write(val | 0xa0);     /* enable trickle charger per user input */
        I2C_stop();
    }
    else
    {
        I2C_start();
        I2C_write(ADDRTC);          /* write slave address + write */
        I2C_write(TC);          /* write register address */
        I2C_write(0);               /* disable trickle charger */
        I2C_stop();
    }
    I2C_start();
    I2C_write(ADDRTC);          /* write slave address + write */
    I2C_write(TC);          /* write register address */
    I2C_start();
    I2C_write(ADDRTC | 1);     /* write slave address + read */
    printf("Trickle Charger: %02bx", I2C_read(NACK) );
}
main     (void)     /* ----------------------------------------------------- */
{
uchar     M, M1;

    sqw = 1;     /* set up for read, weak pull-up */

    while (1)
    {
```

```
#if defined DS1307
        printf("DEMO1307 build %s", __DATE__);
#elif defined DS1337
        printf("DEMO1337 build %s", __DATE__);
#elif defined DS1338
        printf("DEMO1338 build %s", __DATE__);
#elif defined DS1339
        printf("DEMO1339 build %s", __DATE__);
#elif defined DS1340
        printf("DEMO1340 build %s", __DATE__);
#endif
        printf("CI Init RTC    CR Read Clock");
        printf("BR Byte Read   BW Write Byte");

#if defined DS1337 || defined DS1339        /* only print if part has alarms */
        printf("AI Alarm 1 Int AR Alarm Read");
#endif
#if defined DS1340 || defined DS1339        /* parts that have trickle charger */
        printf("Tc Trickle charger");
#endif

#if defined DS1307 || defined DS1338        /* only print if part has RAM */
        printf("RR RAM Read    RW RAM Write");
#endif


        printf("Enter Menu Selection:");

        M = _getkey();

        switch(M)
        {
            case 'A':
            case 'a':
            printf("Init or Read: ");
            M1 = _getkey();

            switch(M1)
            {
                case 'I':
                case 'i':       alrm_int();     break;
```

```
        case 'R':
        case 'r':       alrm_read();       break;
}
break;

case 'B':
case 'b':
printf("Read or Write: ");
M1 = _getkey();
switch(M1)
{
        case 'R':
        case 'r':       readbyte();       break;

        case 'W':
        case 'w':       writebyte();       break;
}
break;

case 'C':
case 'c':
printf("\rEnter Clock Routine to run:C");
M1 = _getkey();

switch(M1)
{
        case 'I':
        case 'i':       initialize();       break;

        case 'R':
        case 'r':       disp_clk_regs(0x99);       break;
}
break;

case 'R':
case 'r':
printf("\rEnter Ram Routine to run:R");
M1 = _getkey();
```

```
            switch(M1)
            {
                case 'R':
                case 'r':       burstramread();       break;

                case 'W':
                case 'w':       printf("Enter the data to write: ");
                                scanf("%bx", &M1);
                                burstramwrite(M1);       break;
            }
            break;

            case 'T':
            case 't':       tc_setup();       break;
        }
    }
```

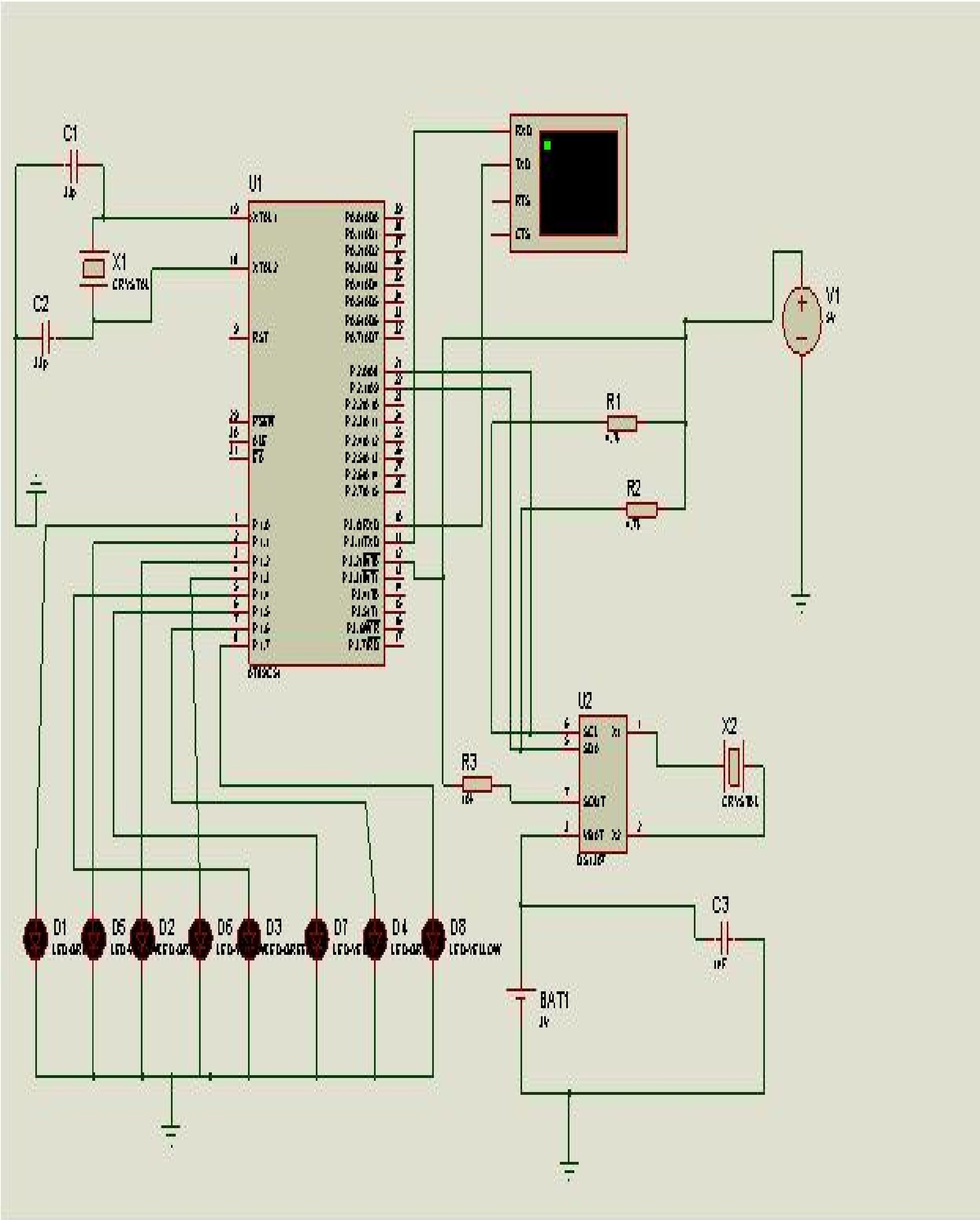
4.2 RTC PROGRAM

```
#include<reg51.h>

#include<stdio.h>

#include<ds1307.h>

void delay(int);

unsigned char  RTC_ARR[7];  // Buffer for second,minute,.....,year

unsigned char p;

void main(void)

{

TMOD=0x20;

SCON=0x52;

TH1=0xFD;

TR1=1;



               //-----------------------------------
```

```
// Setup time and enable oscillator
//-----------------------------------
ReadRTC(&RTC_ARR[0]);
RTC_ARR[0] = RTC_ARR[0] & 0x7F;        // enable oscillator (bit 7=0)
RTC_ARR[1] = 0x50;       // minute = 59
RTC_ARR[2] = 0x12;       // hour = 05 ,24-hour mode(bit 6=0)
RTC_ARR[3] = 0x03;       // Day = 1 or sunday
RTC_ARR[4] = 0x10;       // Date = 30
RTC_ARR[5] = 0x08;       // month = August
RTC_ARR[6] = 0x10;       // year = 05 or 2005
WriteRTC(&RTC_ARR[0]);           // Set RTC
//-----------------------------------
while(1)
{           if(RTC_ARR[1]==0x51)
   P1=0xaa;
            if(RTC_ARR[1]==0x52)
            P1=0x55;
            ReadRTC(&RTC_ARR[0]);
            putchar(0x0C);  // clear Hyper terminal
            printf("calender \n");
            printf("Day  : %s\r\n",Int2Day(RTC_ARR[3]));
            printf("Time : %02bX:%02bX:%02bX\r\n",RTC_ARR[2],RTC_ARR[1],RTC_ARR[0]);
            printf("Date                                    :                                    %02bX-%s-
20%02bX",RTC_ARR[4],Int2Month(RTC_ARR[5]),RTC_ARR[6]);
            //
            delay(10000);      // delay about 1 second
```

```
        }
}
void delay(int time)
{
int i,j;
for(i=0;i<time;i++);
for(j=0;j<5000;j++);
}
```

4.3 CIRCUIT


C1
C2
X1
CRYSTAL
U1
RST
V1
R1
R2
R3
U2
X2
CRYSTAL
C3
BAT1
D1
D5
D2
D6
D3
D7
D4
D8
LED-YELLOW

5. APPLICATIONS OF RTC

- A real time clock (RTC) circuit may be used to maintain rough GPS time while the rest of the GPS circuitry is off. The global positioning system (GPS) is a collection of twenty-four earth-orbiting satellites. Each of the GPS satellites travels in a precise orbit about 11,000 miles above the earth's surface. A GPS receiver locks onto at least three of the satellites to determine its precise location.

- Data recorders for collecting and managing vehicle operational data or messages are known in the automotive and heavy duty truck industries. Data analyzers having such capability are thus able to link the vehicle operational data to vehicle operating conditions existing during data recordation such as a particular driver or group of drivers, weather conditions, geographic locations, road grade conditions encountered, and other vehicle operating conditions. Real time clocks are incorporated in electronic data recorders to "time-stamp" the vehicle operational data or messages as they are recorded.

- Real time clock is also used in managed network elements for time stamping event reports sourced from the respective element. Such reports may, for example, be the start or finish times of a traffic connection

  Clock circuits are an essential component of modern computer systems

- A typical computer system includes a processor subsystem of one or more microprocessor, a memory subsystem, one or more chipsets provided to support different types of host processors for different platforms such as desktops, personal computers (PC), servers, workstations and mobile platforms, and to provide an interface with a plurality of input/output (I/O) devices. Chipsets may integrate a large amount of I/O bus interface circuitry and other circuitry onto only a few chips. These chipsets may implement the I/O bus interface circuitry, timer, real-time clock (RTC), direct memory access (DMA) controller, and other additional functionality such as integrated power and thermal management with quick resume capabilities and random seed number generation for security applications such as cryptography, digital signatures, and protected communication protocols. Clock circuits generate a regular series of pulses based on a piezoelectric crystal, which governs the frequency of the pulses. The clock signal that is generated is used to synchronize the operation of the other components and circuits in the system. The boot up operation of a computer system is reliant upon the establishment of a reliable system clock. Establishing this reliable signal adds additional time to the boot up process for a system.

- A typical personal computer includes two time keeping systems: a hardware real time clock, and a software virtual clock maintained by an operating system. The hardware real time clock typically includes a battery backup source of electrical power, and continuously maintains an estimate of the current date and time. The software virtual clock is typically synchronized to the RTC during PC power up and initialization.

  On a computer mother board, a RTC is provided to maintain the data stored in a CMOS on the mother board. Generally, electrical power is provided to a RTC by a battery arranged on the mother board. The RTC is used to update the current time and date within the computer system without any intervention from the system processor. Usually the RTC and other peripheral devices are coupled to the peripheral bus through an interrupt controller. Such a clock has various uses such as time stamping files and inserting dates into documents.

- The RTC is used to update a program clock, typically a date/time clock maintained by the operating system (OS) and referred to herein as the OS clock. An operating system may include a system clock to provide a system time for measuring small increments of time. The system clock may update the system clock in response to a periodic interrupt generated by a system timer, or a real time clock event timer.

- A personal digital assistant (PDA) is a pen-based computer system where the primary method for inputting data includes a pen or stylus. Many of the functions of a PDA require an accurate real-time clock to provide the calendar function, the clock display, and various timer functions of a PDA require an accurate real-time clock.

- An electronic apparatus generally has a RTC (real time clock) installed in the circuit board thereof for recording system time. A real time clock is one of the requisite hardware devices of an electronic device. A standard RTC includes an oscillating circuit coupled to a digital counter.

- A typical real time clock can be programmed to produce or generate an RTC alarm event at a designated time. The RTC alarm event may cause, for example, the assertion of an interrupt signal to the microprocessor

- A real time clock is typically implemented in specific hardware with a dedicated crystal oscillator to insure accuracy and a battery backup power supply to insure preservation of timekeeping data during an interruption of the primary power supply. This is especially important with personal computers which are frequently powered down. It is common to back up a real-time clock (RTC) with a battery, such as a long-life lithium battery, so that when a main power supply for the RTC fails, or has been disconnected, the RTC will continue to indicate the correct time and provide correct time intervals.

6.CONCLUSION

Real time clocks can be much more accurate than other time-keeping alternatives, they allow the main system to perform important tasks, and they do not consume much power. Electronic devices can even increase their functionality by using real-time clocks. Certain electronic devices can rely on real-time clocks when comparing the times of previous functions. If the functions have taken place within a designated period of time, device functions can be reduced drastically.

Hence real time clocks using microcontrollers are used extensively in controlling home appliances