LAB 6
Real Time Clock, EEPROM and the I2C Bus

Suggested Reading:
DS1307 Spec sheet
I2C section from Al Williams Book *Microcontroller Projects using the Basic Stamp.*
24C32 Spec sheet

1. Real Time Clock

It's sometimes necessary to have a reliable time keeping method that is accurate over days or longer. For example, it might be desired to log the exact time at which events occur, or to keep an accurate record of the time at which repeated measurements are made. It is sometimes necessary to control motors or relays on an exact schedule. The crude time-keeping capabilities of the Basic Stamp itself are usually too inaccurate for these applications.

A real time clock is simply a time-keeping chip, very similar to those in digital wrist-watches. The oscillator is regulated by a crystal which has a very high accuracy. The quartz crystal most commonly used has a frequency of 32768 Hz. This frequency allows each second to be exactly 2^{15} periods of the oscillation. The Dallas Semiconductor DS1307 is economical and can be easily interfaced to a micro-controller through a standard 2-wire serial interface bus called I2C.

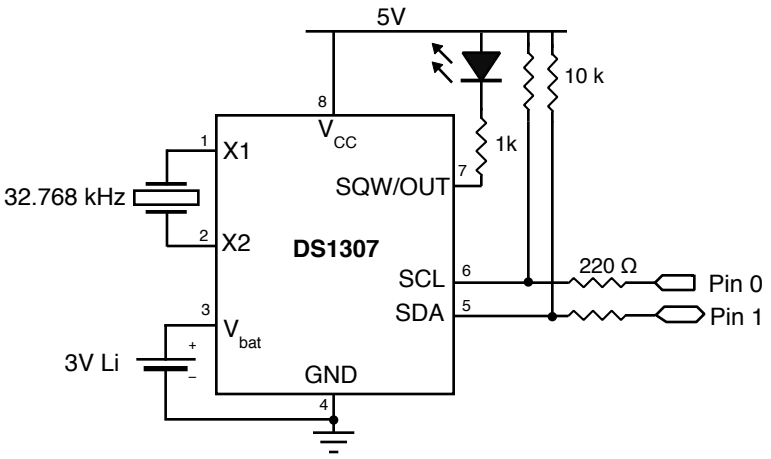

Fig 1: The real time clock has two interface lines SCL, clock, and SDA, data. The 32768 Hz quartz crystal maintains accurate time. The 3 V lithium battery is optional and is necessary if the time setting is to be maintained when the main voltage supply, Vcc, is off. The SQW/OUT can be programmed to output a 1Hz signal and flashe the LED. The LED can be removed after correct operation has ben established.

The DS1307 must be programmed at startup by writing to eight one-byte registers. Seven of these registers contain the date and time in BCD format: seconds, minutes, hours, day of the week, date, month and year. The eighth register control the oscillator output on pin 7. The hour can be either 12-hour mode with and am/pm bit or 24 hour mode.

Subsequent to setting the clock by writing to the internal registers, these registers are continually updated with the current time and date. The time can be read by accessing the values of these registers.

The exact format of the register contents and the protocol for reading and writing them is described in detail in the DS1307 data sheet.

The backup 3V battery is necessary if the clock needs to keep its time when Vcc is off. The LED

is useful to verify that the clock is functioning. It could be removed after the clock's reliability has been established.

The following program, RTCSet.BS2 sets the contents of the registers according the the date and time constants which are defined in the first few lines of the program. The basic I2C subroutines are adapted from Al Williams's I2C routines in chapter 6 of *Microcontroller Projects using the Basic Stamp* .

The I2C bus is based on open-collector signal lines, SCL and SDA. Both lines have a 10kΩ pull up resistor to Vcc. As output, the pins of the Stamp are three-state,. Open-collector operation can be managed by switching the line to "INPUT" when a high level is desired and when A low output is required, the output pin is changed to LOW ouput.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
' RTCSet.BS2  --- set the time and date in DS1307

'-----Date and time constants -- encoded in BCD ------
'  Change these values immediately before downloading

seconds CON $20
minutes CON $0
hours CON $12      ' 12 noon and 24hr mode (use $52 for 12 hr mode)

day CON $05 ' day of the week, 0 = Sunday, etc,  in my system
date  CON $08      ' day of the month
month CON $10      ' Month
year CON $04       ' Year can be set from 2000 to 2099

controlreg CON $10       ' output a 1 Hz square wave to flash led

'---------'

scl CON 0
sda CON 1
sdain VAR IN1
sclin VAR IN0

i2cackbit VAR BIT   ' Ack bit -- should be zero after operations
i2craw VAR BYTE     ' I2C in or out byte
rtcadr VAR WORD      ' rtc RAM address

i VAR BYTE


'  ------------- set up the real time clock registers
GOSUB i2cstart

i2craw = $D0       ' 1307 address in write mode
GOSUB i2cwrite
GOSUB i2cack

i2craw = $00       ' set starting address
GOSUB rtcwrite

i2craw = seconds
GOSUB rtcwrite
```

```
i2craw = minutes
GOSUB rtcwrite


i2craw = hours
GOSUB rtcwrite

i2craw = day
GOSUB rtcwrite

i2craw = date
GOSUB rtcwrite

i2craw = month
GOSUB rtcwrite

i2craw = year
GOSUB rtcwrite

i2craw = controlreg     ' the last register is the contol register which set
output oscillator
GOSUB rtcwrite

GOSUB i2cstop

'    ------------ Read the date and time ---------


      '            ----- reset the register pointer to 0 --------
      GOSUB i2cstart

      i2craw = $D0
      GOSUB rtcwrite    'address the DS1307 in write mode

      i2craw = $00
      GOSUB rtcwrite    ' write the register pointer to 0

      GOSUB i2cstop

      GOSUB i2cstart

      i2craw = $D1      ' Address the DS1307 in read mode
      GOSUB rtcwrite
      DEBUG "RTC registers set to "
      FOR i = 0 TO 6
            GOSUB i2cread
            GOSUB i2csendack
            DEBUG  HEX2 i2craw, " "
      NEXT

      GOSUB i2cread
      GOSUB i2csendnotack
      DEBUG " " , HEX2 i2craw, CR
      GOSUB i2cstop

DEBUG "This program is now stopping." , CR

STOP
```

```
'------------ Begin I2C Code ---------------
'--------- Essential i2c subroutines------------------------

' Set up a start condition
' Could hang if bus stays busy; could remove i2cbusy and the if
' following it if you are the only master
i2cstart:    ' ------------------------------ i2c Start -------------------
  INPUT sda
  INPUT scl
i2cbusy:
  IF (sdain AND sclin)=0 THEN i2cbusy
  LOW sda
  LOW scl
  RETURN

' Set up stop condition
i2cstop:     ' ------------------------------ i2c Stop -------------

  LOW sda
  INPUT scl
  INPUT sda
  RETURN

' Read ack bit
i2cack: ' ------------------------------ i2c Read Ack bit
  INPUT scl
await:
  IF sclin=0 THEN await   ' wait for clock stretch
  i2cackbit=sdain
  LOW scl
  RETURN

' Read ack bit
i2csendack: ' ------------------------------ i2c SEND Ack bit --------------

      LOW sda
      INPUT scl
      LOW scl
      INPUT sda

  RETURN

i2csendnotack: ' ------------------------------ i2c SEND NOT-Ack bit -------
  INPUT scl
  LOW scl
  RETURN

' Write 8 bits to I2C bus
' For multi-master, should perform arbitration
' Should also perform clock sync
i2cwrite: ' ------------------------------ i2c Write 8 bits (i2craw) -------
  SHIFTOUT sda,scl,1,[i2craw\8]
  INPUT sda
  RETURN


' Read 8 bits from I2C bus
```

```
' Should perform clock sync
i2cread: ' ------------------------------ i2c Read 8 bits (i2craw) ---------
  SHIFTIN sda,scl,4,[i2craw\8]
  RETURN
'-------------------------------------------------------------------------
' Write byte to rtc registers
rtcwrite:
  GOSUB i2cwrite
  GOSUB i2cack
  IF i2cackbit=0 THEN rtcwriteok
  ' NACK error
      DEBUG "NACK error"
rtcwriteok:
  RETURN
' End of Essential I2C routines.
```

The next program, RTCRead.BS2, periodically reads the contents of the date and time registers and prints them out on the debug screen. The time between reads is set by the variable "sleepytime" in seconds. Downloading RTCRead.BS2 does not affect the time in the real time clock which runs independently of the micro-controller. The date and time can be immediately read at any time during the sleep cycle by pressing the "RESET" button. The format of the date and time may be inconvenient. You may wish to reformat the output to show the date and time in a more conventional format.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
' RTCREAD.BS2 -- Read the print the date and time registers of DS1307 RTC

sleepytime CON 10 ' Time between update in seconds
'---------'

scl CON 0
sda CON 1
sdain VAR IN1
sclin VAR IN0

i2cackbit VAR BIT  ' Ack bit -- should be zero after operations
i2craw VAR BYTE    ' I2C in or out byte
rtcadr VAR WORD     ' rtc RAM address

i VAR BYTE



'    ------------ Read the date and time at regular intervals ---------
DO

      '            ----- reset the register pointer to 0 --------
      GOSUB i2cstart

      i2craw = $D0
      GOSUB rtcwrite    'address the DS1307 in write mode

      i2craw = $00
      GOSUB rtcwrite    ' write the register pointer to 0

      GOSUB i2cstop
```

```
        GOSUB i2cstart

        i2craw = $D1        ' Address the DS1307 in read mode
        GOSUB rtcwrite

        FOR i = 0 to 5
              GOSUB i2cread
              GOSUB i2csendack
              DEBUG  HEX2 i2craw, " "
        NEXT

        GOSUB i2cread
        GOSUB i2csendnotack
        DEBUG " " , HEX2 i2craw, CR
        GOSUB i2cstop
        SLEEP sleepytime

LOOP



STOP

'------------ Begin I2C Code ---------------
'----------Essential i2c subroutines--
' Same as previous program....
```

2. Logging Data in an EEPROM

Data can be permanently stored in electrically erasable programmable memory (EEPROM). This technology is the same as that used in flash memory. EEPROM can be written a finite number of times, usually more than one million times and read any number of times. The lifetime of the memory is estimated to be more than 200 years.

Many EEPROM chips use the I2C protocol for reading and writing its contents. The 24LC32 chip has 32 k bits of memory, or 4 k bytes. These chips can be read of written in individual bytes or in chunks of 8 bytes (one page) up to 64 bytes at a time. Three address pins allow up to eight EEPROM chips to be bussed on one bus with distinct addresses.

Following are two program to illustrate the usage of the EEPROM. The first, EEPROM1.BS2, is a variation on RTCRead.BS2 which reads the real time clock 64 times and stores the date and time register contents in the memory. The second program reads the EEPROM and prints the values on the DEBUG screen. Note that the first eight bytes are not used in these programs because extensions of these programs which could log an undetermined number of values. It is possible that one would wish keep a pointer to the highest memory location used.

Verify that the EEPROM keeps the data even if power is switched off or even if the EEPROM is removed and later reinserted into the circuit.

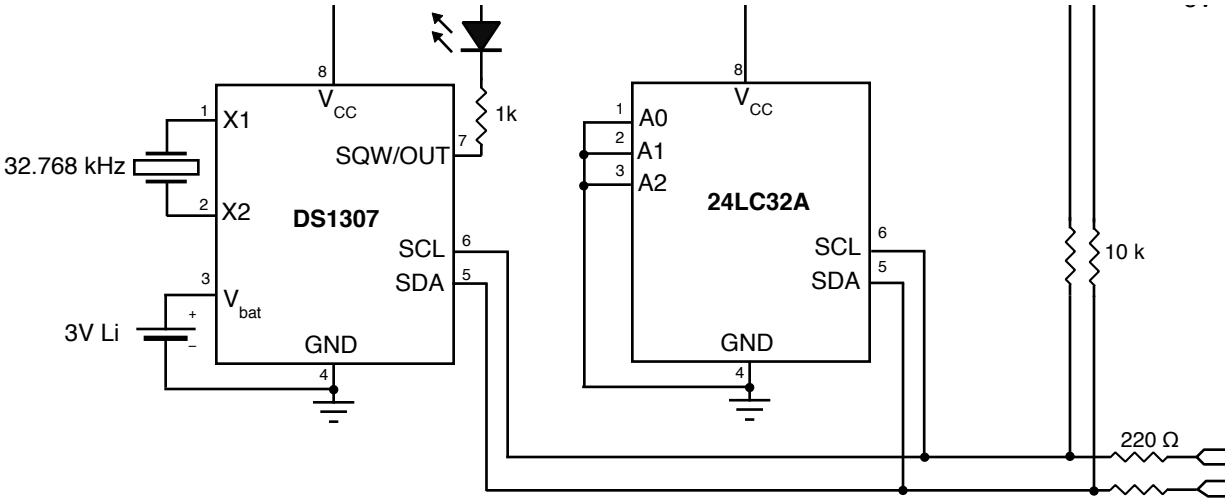

Fig 2: The 24LC23A EEPROM contains 32k bits of storage. The stored data are not lost when the voltage supply is off. It can be interfaced to the micro-controller using the same two lines of the I2C bus in common with the DS1307. Up to 7 additional EEPROM chips can be added if each each is assigned a different address on pins A0,A1 and A2.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
' EEPROM1.bs2  -- write 84 date and time stamps to EEPROM as an example of
'               Storing data in and EEPROM.
sleepytime CON 10
'---------'

scl CON 0
sda CON 1
sdain VAR IN1
sclin VAR IN0

i2cackbit VAR BIT   ' Ack bit -- should be zero after operations
i2craw VAR BYTE     ' I2C in or out byte
rtcadr VAR WORD      ' rtc RAM address
eedata VAR BYTE(8)    ' EEPROM data in or out, array numbered 0..7


eeadr VAR word     'Allow for 12-bit address

i VAR BYTE


'    ------------ Read the date and time at regular intervals ---------
eeadr = 0
DO

      '            ----- reset the register pointer to 0 --------
      GOSUB i2cstart

      i2craw = $D0
      GOSUB rtcwrite     'address the DS1307 in write mode

      i2craw = $00
      GOSUB rtcwrite     ' write the register pointer to 0
```

```
      GOSUB i2cstop

      GOSUB i2cstart

      i2craw = $D1      ' Address the DS1307 in read mode
      GOSUB rtcwrite

      FOR i = 0 TO 5
            GOSUB i2cread
            GOSUB i2csendack
            eedata(i)=i2craw
      NEXT

      GOSUB i2cread
      GOSUB i2csendnotack
      eedata(6)=i2craw
      GOSUB i2cstop
      DEBUG "20",hex2 eedata(6)," ",hex eedata(5), " ", hex eedata(4)," ",
hex eedata(3)," ",hex  eedata(2),":",hex2 eedata(1),":",hex2 eedata(0),CR
      eeadr = eeadr + 8
      GOSUB eewritepage


SLEEP sleepytime

LOOP while eeadr <= 64



STOP

'------------ Begin I2C Code ---------------
'-------Essential i2c subroutines-----------------------
Same as previous program....


eewritepage:
      GOSUB i2cstart
      i2craw = $A0 ' EEPROM device code, write mode
      GOSUB i2cwrite
      GOSUB i2cack
      if i2ackbit = 1 then eepageerr
      debug "MSB of address" ,HEX2 (eeadr >> 8) & $0F, " "
      i2craw = (eeadr >> 8 ) & $0F  ' MS nibble of address
      GOSUB i2cwrite
      GOSUB i2cack
      if i2ackbit = 1 then eepageerr
      DEBUG "LSB of address ",HEX2 eeadr & $FF, CR
      i2craw = eeadr & $FF
      GOSUB i2cwrite
      GOSUB i2cack
      if i2ackbit = 1 then eepageerr

      DEBUG " Finished writing address", CR

      for i = 0 to 6
            DEBUG "Writing ", DEC i, CR
```

```
            i2craw = eedata(i)
            gosub i2cwrite
            GOSUB i2cack
            if i2cackbit = 1 then eepageerr
      next

      GOSUB i2cstop
      GOTO eepageret

eepageerr:
      debug "Error writing to EEPROM ", DEC i, CR
eepageret:
      return
```

Here is the second program which reads back the data and prints it on the DEBUG screen.

```
'{$STAMP BS2}
'{$PBASIC 2.5}
'EEPROMRead.BS2  -- Read back 64 date and time stamps from EEPROM

sleepytime CON 10
'---------'

scl CON 0
sda CON 1
sdain VAR IN1
sclin VAR IN0

i2cackbit VAR BIT  ' Ack bit -- should be zero after operations
i2craw VAR BYTE    ' I2C in or out byte
rtcadr VAR WORD     ' rtc RAM address
eedata VAR BYTE(8)   ' EEPROM data in or out, array numbered 0..7


eeadr VAR word    'Allow for 12-bit address

i VAR BYTE


'    ------------ Read the date and time at regular intervals ---------
eeadr = 0
DO

      eeadr = eeadr + 8
      GOSUB eereadpage
      DEBUG "20",hex2 eedata(6)," ",hex eedata(5), " ", hex eedata(4)," ",
hex eedata(3)," ",hex  eedata(2),":",hex2 eedata(1),":",hex2 eedata(0),CR

LOOP while eeadr <= 64



STOP

'------------ Begin I2C Code ---------------
'-------  Essential I2C subroutines-----------------------

'    Same as previous Program.….
```

```
eewritepage:
      GOSUB i2cstart
      i2craw = $A0 ' EEPROM device code, write mode
      GOSUB i2cwrite
      GOSUB i2cack
      if i2cackbit = 1 then eepageerr

      i2craw = (eeadr >> 8 ) & $0F  ' MS nibble of address
      GOSUB i2cwrite
      GOSUB i2cack
      if i2cackbit = 1 then eepageerr

      i2craw = eeadr & $FF
      GOSUB i2cwrite
      GOSUB i2cack
      if i2cackbit = 1 then eepageerr



      for i = 0 to 6
            DEBUG "Writing ", DEC i, CR
            i2craw = eedata(i)
            gosub i2cwrite
            GOSUB i2cack
            if i2cackbit = 1 then eepageerr
      next

      GOSUB i2cstop
      GOTO eepageret

eereadpage:
      GOSUB i2cstart
      i2craw = $A0 ' EEPROM device code, write mode
      GOSUB i2cwrite
      GOSUB i2cack
      if i2cackbit = 1 then eepageerr

      i2craw = (eeadr >> 8 ) & $0F  ' MS nibble of address
      GOSUB i2cwrite
      GOSUB i2cack
      if i2cackbit = 1 then eepageerr

      i2craw = eeadr & $FF
      GOSUB i2cwrite
      GOSUB i2cack
      if i2cackbit = 1 then eepageerr


      GOSUB i2cstart
      i2craw = $A1      ' EEPROM device code, read mode
      GOSUB i2cwrite
      GOSUB i2cack
      if i2cackbit = 1 then eepageerr

      for i = 0 to 6
            GOSUB i2cread
            gosub i2csendack
            eedata(i)=i2craw
```

```
        next
        GOSUB i2cread
        GOSUB i2csendnotack
        eedata(7)=i2craw
        GOTO eepageret


eepageerr:
        debug "Error writing to EEPROM ", DEC i, CR
eepageret:
        return
```

After you have got these exercises functioning try to make a more interesting application of this basic data logging system. Here are some exceptions:

1. Log the times which a switch is closed. For example, the switch could be operated by a door or window in order to monitor the frequency and times that a room is entered.

2. Extend the exercise of 1 to also keep track of how long the switch is pressed each time. This might be useful in determining how long your roommates keep the fridge door open, or, if connected to your telephone, in maintaining a record of telephone calls.

3. Combine one of the physical measurements of previous labs, such as temperature or light levels, with periodic logging of the value along with date time into the EEPROM.

Your own ideas:
_____
_____
_____

Printed with a Demo of Nisus Writer Express