Column #68, December 2000 by Jon Williams:

There's A New Stamp In Town - Part 1

Wow, 2001 is just around the corner. The first time I saw the movie that made the year famous, 2001 seemed like it was an eternity away…but here it is. Arthur C. Clarke is an incredibly bright man and yet, it seems he over-predicted the ability of computers in the year 2001. Or, perhaps, he under-predicted…I guess it depends on one's point-of-view. No, we don't have HAL this year, what we have is the very well-mannered BASIC Stamp IISX+ (BSP).

The BSP takes all the really good stuff of the BASIC Stamp IISX, makes it better and adds some really great features. Here's an overview:

- Parallel LCD routines for the Hitachi HD44780
- Dallas 1-Wire™ routines
- Philips I2C routines
- Firmware interrupts
- 24-pin (16 I/Os) and 40-pin (32 I/Os) versions
- Double the scratchpad RAM of the BS2-SX (now 127 bytes)
- 20% faster than the BS2-SX (about 12,000 instructions per second)
- Uses less power (about 30%) than the BS2-SX – operates near 40 mA

If you've used the BS2 or BS2-SX for any length of time, you'll recognize that this is a very cool list of features and goes a long way toward extending the Stamp's versatility. This month we'll focus on LCD, 1-Wire™ and I2C routines since they bring us the most bang. Next month we'll cover expanded I/O on the 40-pin version and the use of firmware interrupts.

You know me, I learn by doing and teach by having you do. Let's jump right in.

There is a commonality among the LCD, 1-Wire™ and I2C routines: they have an input/output structure that is identical to SERIN and SEROUT. And even though several Stamp programmers have been successful at implementing LCD (easy) and I2C (not so easy) routines, the code is often slow and bulky. Until the BSP, 1-Wire™ support was not possible without external support and even then it was difficult and clumsy.

LCD Support

The BSP has native support for the popular Hitachi HD4470 LCD controller. The routines that support LCD control are:

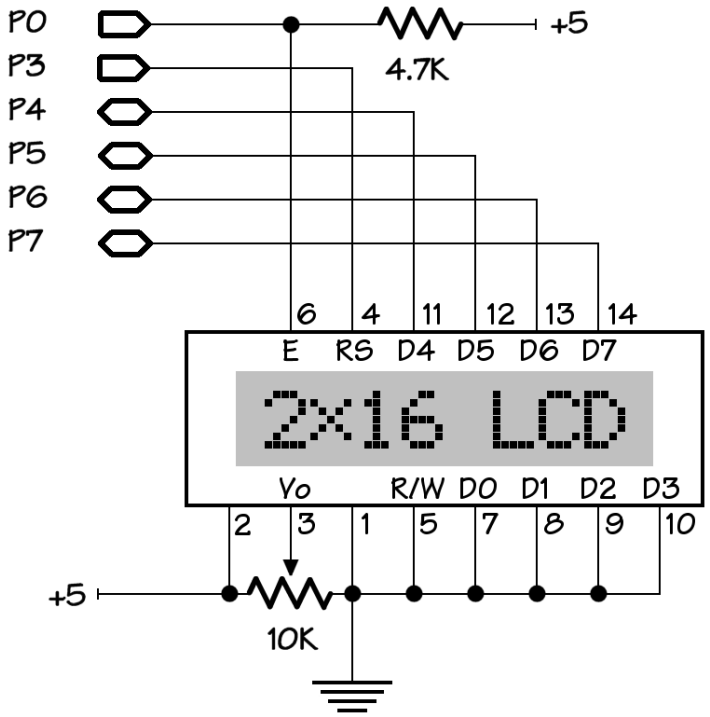
LCDCMD *E-pin*, *command*
LCDOUT *E-pin*, *command*, [*output data*]
LCDIN *E-pin*, *location*, [*input data*]

These routines require that the LCD be configured in 4-bit mode and have specific requirements as to where the connections can be. The syntax of each statement (specifically the control pin for LCD.E) tells the BSP how the LCD is connected.

Connections:

LCD	Option 1	Option 2
LCD.E	BSP.0 or BSP.1	BSP.8 or BSP.9
LCD.R/W	BSP.2	BSP.10
LCD.RS	BSP.3	BSP.11
LCD.DB4	BSP.4	BSP.12
LCD.DB5	BSP.5	BSP.13
LCD.DB6	BSP.6	BSP.14
LCD.DB7	BSP.7	BSP.15

Figure 68.1: BS2p hookup to HD44780 LCD



This table shows that the LCD can be connected to the pins at OutL (0-7) or the pins at OutH (8-15) and specific requirements as to where E, R/W, RS and the data lines need to be connected. Keep in mind that you don't have to use the LCD's R/W line if you are not going to read from its RAM. In this case, you can simply ground the LCD.R/W pin and use the Stamp control pin for other duties. Note that the Parallax documentation suggests that the LCD.E line be pulled down to ground through a 4.7K resistor.

The first of the LCD commands is LCDCMD and is used to send command a control code to the LCD. This command will be used during initialization and for moving the cursor home, clearing the LCD, etc. Here's a few typical commands:

Clear the LCD	$01
Move cursor home	$02
Move cursor left	$10

Move cursor right $14

There are others. Consult the program listings here and the Hitachi documentation for the HD44780.

Writing data to the LCD has been made very easy with LCDOUT. There are two nice things about this new command: you can send the LCD a command byte (i.e., clear the LCD) before the write and the output data is structured just like SEROUT. This means you can use the typical SEROUT modifiers like BIN, HEC, DEC, STR and REP.

Reading information back from the LCD is just as easy with LCDIN. Its syntax is identical to LCDOUT. With LCDIN, the *location* byte needs to point to the starting memory location to read. The following constants are useful in programs that use the LCD commands:

```
DDRam     CON   $80
CGRam CON   $40
```

DDRam is the memory that holds the characters being displayed. CGRam is the 64-byte memory area where custom character patterns are stored. If your program doesn't use this memory for custom characters, it can be used as off-board RAM with LCDOUT and LCDIN. The STR modifier can be used to transfer a block of bytes to or from the LCD.

Dallas 1-Wire™ Support

The Dallas 1-Wire™ bus is a system which has a single bus master and one or more slaves. In this case, the BSP acts as the bus master. Each device on the 1-Wire™ bus has a unique serial number (used for addressing) that is manufactured right into the device.

1-Wire™ devices are supported with two easy-to-use commands:

OWOUT *pin*, *reset*, [*output data*]
OWIN *pin*, *reset*, [*input data*]

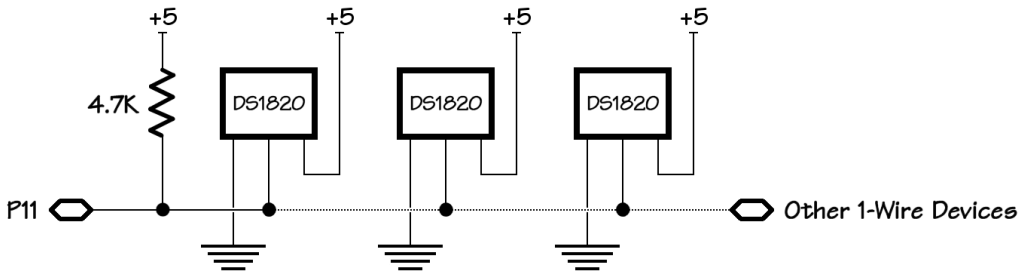
Communication to 1-Wire™ devices can be on any available pin. This pin should be pulled up to Vdd (+5) through a 4.7K resistor. The *reset* parameter has four options:

No reset or presence pulses
Reset and presence pulses only before data initiation
Reset and presence pulses only after data termination

Reset and presence pulses before data initiation and after data termination

Resets shown are for byte input data running at regular speed. Add four for bit input data and add eight for overdrive speed. You should consult the Dallas 1-Wire™ documentation for specifics on the reset option.

Figure 68.2: BS2p hookup to Dallas Semiconductor 1-wire devices



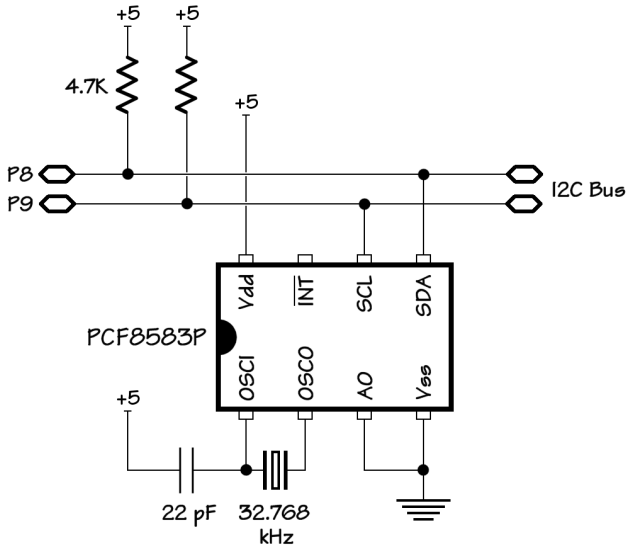
Philips I2C Support

The Philips I2C bus is a 2-wire, bi-directional bus. The two lines are SDA (serial data) and SCL (serial clock). Like the 1-Wire™ line, the SDA and SCL lines must be pulled up to Vdd (+5) through 4.7K resistors.

I2C devices are supported with:

I2COUT *pin*, *slave addr*, *word addr\extra byte*, [*output data*]
I2CIN *pin*, *slave addr*, *word addr\extra byte*, [*input data*]

Figure 68.3: BS2p hookup to Phillips I2C devices



The I2C devices can only be connected to group pins 0 & 1 or 8 & 9.

I2C Bus	Option 1	Option 2
SDA	BSP.0	BSP.8
SCL	BSP.1	BSP.9

The *pin* parameter of each command specifies the SDA pin. The *slave addr* is the I2C device to connect with. The *word addr* is the location with the I2C device to write to or read from. The use of the backslash allows two-byte addressing for those devices that support it.

Demo Programs

Okay, enough chit-chat…let's write a few programs that demonstrate these new features.

Program Listing 70.1 (DS1820.BSP) is a program that reads and displays the temperature from a single Dallas DS1820 1-Wire™ thermometer (do not run this program with more than one device on the 1-Wire™ bus). The output of the DS1820 is the same as the

DS1620. The current temperature is returned in half-degrees Celsius. Negative numbers are returned in two's compliment format.

After defining I/O pins, constants and variables used in the program we initialize the LCD. The Hitachi guidelines for a 4-bit interface are followed. What you'll notice is a PAUSE 0 between each of the LCDCMD statements. This small delay is necessary because of the increased speed of the BSP.

With the LCD initialized a splash screen is displayed with LCDOUT. Notice how easy this command is to use, with the ClrLCD command embedded in the same statement. This is a far cry easier than the old days of using LOOKUP or loops to read strings from DATA statements.

The first active process in the program is reading back the ROM code from the DS1820. We'll need this information later if we want to individually address the DS1820. Each DS1820 contains a 64-bit (eight bytes) ROM code. The first eight bits are the 1-Wire™ family code ($10 for the DS1820), the next 48 bits are a unique serial number. The final eight bits is the CRC of the first 58 bits.

Using OWOUT, the ReadROM command issued to the DS1820. The next line of code uses OWIN to retrieve the data from the DS1820. The STR modifier and \8 specification make it very easy to read all eight ROM bytes into the array called romData.

With the ROM data in memory we use a loop and the HEX2 modifier to display the data as hex bytes on line 2 of the LCD. Be sure to write down this code so you can use it later.

Finally, we get to read the temperature. The DS1820 is a bit easier to use than its cousin, the DS1620, in that we only have to tell it to take a temperature reading; there are no other configuration requirements. In this program, we've only placed one device on the 1-Wire™ bus so we don't need to match ROM contents. This is accomplished by issuing the SkipROM command, followed by the ConvertTemp command (done in one statement).

The DS1820 needs about half a second to take a reading. After the PAUSE 500, the temperature reading is retrieved by sending the ReadScratch (after SkipROM, of course) command. Since we're only interested in the temperature, we'll just grab the first two bytes. There is more information available, including data that will allow higher resolution temperature readings. I'll leave that experimenting to you.

Once we have the raw temperature reading, we drop the he half-degree bit convert to Fahrenheit as well. A couple of LCDOUT commands using SDEC (signed decimal – allows negative numbers) displays the temperature neatly. Easy, huh? Yep, I like this new Stamp.

If you've got more that one DS1820, you can enter (or download) Listing 2 (DS1820-x.BSP) and cycle through all of them. Before you do that though, make sure you plug each of them into your circuit and record the ROM data using Listing 1. You'll need this information for the program to work. In fact, the program in Listing 2 will only work with my three DS1820s. You'll need to update the DATA statements for your sensors.

The bulk of the program is the same as Program Listing 68.1. It differs in reading the temperature data from a specific device. Take a look at the subroutine called GetTemp. The address of the first byte of the ROM code is passed in the variable eeAddr. A simple loop reads the ROM code bytes from EEPROM and stores it in the array called romData.

In this program, the MatchROM command is issued, followed by the romData string. The ReadScratch command is issued in the same manner and the subsequent OWIN retrieves the temperature from the specified device.

Keep in mind that different types of devices can exist on this single-wire network. The I/O and expansion options are nearly limitless. Take a look at the Dallas Semiconductor (www.dalsemi.com) website for 1-Wire™ devices as new ones are being introduced almost every day.

The final listing, PCF8583.BSP, demonstrates the BSP's I2C routines by connecting to a real-time-clock with RAM. In fact, the PCF8583 is built around a 256-byte RAM and several of the registers are automatically updated.

The PCF8583 typically uses a 32.768 kHz crystal, but it can be driven by a 50 Hz signal as well (not very useful here in the US, but still an option). This chip can also be configured as an event counter and features an alarm output pin that can save us the trouble of polling and comparing data.

My middle name is "Simple" (okay, not really, but play along), so that's just how we're going to deal with this program. By letting the device power-up on its own, the time is set to midnight, 24-hour mode and it expects to be driven by a crystal. We'll use a four-button interface to set the minutes, hours and day-of-week. The fourth button allows us to decrement these values.

After defining I/O pins, constants, variables and initializing the LCD, the program displays a short splash screen and then enters the main loop. The loop process is simple: we read the clock and day, update the LCD the scan the buttons. If a button is pressed, the clock is updated and the loop continues.

Since the PCF8583 is register-oriented, we can read and write locations individually or in blocks. The subroutine called GetTimeAndDay retrieves seven bytes from the PCF8583, beginning at address 0. These bytes contain the control, time and date registers. Most PCF8583 registers hold their data in BCD format, so we use the HighNib and LowNib variable modifiers to convert to standard decimal for our program variables. The day-of-week value is store in the highest three bits of the register that holds the current month. The right shift operator (>>) lets us move this value into the variable called day.

Our main time variable is called rawTime. This word-sized variable holds the time in minutes past midnight. When updating the clock, we'll actually update rawTime then pass it to the routine that takes care up sending data to the PCF8583.

The name of the current day is sent to Line 1 of the LCD with the subroutine PrintDay. The day variable is used by a LOOKUP table to get the EE address of the zero-terminated string. This technique is nice because you can update your strings without modifying operational code.

The program loops through the BSP's EEPROM, reading each character in the current day name until a zero is encountered. Zero flags the end of the string. Since some strings are longer than others, the end is clean-up by printing a few spaces.

Printing the time on the LCD is even easier with LCDOUT and the DEC2 modifier. One line of code prints the current time on the right edge of Line 2.

The buttons are scanned and debounced with a loop. The idea behind this code is that a button must start pressed and stay pressed through the entire loop for it to be a valid (debounced) button press. The state of the buttons is returned in a nibble with each bit aliased so we can use it to make the update.

Again, the variable rawTime is our master time variable and any updates with be preformed on it. This variable hold minutes. When the minutes button is pressed, one is added. When the hours button is pressed, 60 is added. The program uses a trick with the modulus operator (//) to keep rawTime in the range of 0 to 1439. Using the modulus operator means that we can subtract from rawTime by adding (1439 for one minute, 1380

for 60 minutes) and then doing the modulus operation. The day value is handled in the same manner.

The subroutine called PutRawClock takes rawTime and coverts it to the proper BCD format used by the PCF8583. The new data is sent to the PCF8583 with I2COUT.

I don't know about you, but I'm really excited about the BASIC Stamp SXII+. More functionality, faster, consumes less current…it's a winner.

Next month we'll talk about the firmware interrupts and additional I/O support on the 40-pin version of the chip. Until then, happy Stamping.

```
' Program Listing 68.1
' Nuts & Volts - December 2000

' ----[ Title ]---------------------------------------------------------
'
' File..... DS1820.BSP
' Purpose... BASIC Stamp SX Plus <--> DS1820 Demo
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 04 NOV 2000
' Updated... 04 NOV 2000


' ----[ Program Description ]-------------------------------------------
'
' This program reads and displays the ROM code and temperature data from a
' DS1820 (1-wire) sensor.
'
' Program requires 2x16 LCD
'   - LCD.E    --> Pin0 (pulled down [to ground] through 4.7K)
'   - LCD.R/W  --> Pin2 (or grounded for write-only operation)
'   - LCD.RS   --> Pin3
'   - LCD.D4   --> Pin4
'   - LCD.D5   --> Pin5
'   - LCD.D6   --> Pin6
'   - LCD.D7   --> Pin7


' ----[ Revision History ]----------------------------------------------
'


' ----[ I/O Definitions ]-----------------------------------------------
'
LCDpin         CON    0                    ' data on pins 4 - 7
DS1820pin      CON    11


' ----[ Constants ]-----------------------------------------------------
'
' LCD control characters
'
NoCmd          CON    $00                  ' just print
ClrLCD         CON     $01                 ' clear the LCD
CrsrHm         CON     $02                 ' cursor home
CrsrLf         CON     $10                 ' cursor left
CrsrRt         CON     $14                 ' move cursor right
DispLf         CON     $18                 ' shift display left
DispRt         CON     $1C                 ' shift displayright
DDRam          CON     $80                 ' Display Data RAM control
```

```
Line1          CON     $80                     ' address of line 1
Line2          CON     $C0                     ' address of line 2

DegSym         CON     223                     ' degrees symbol

' DS1820 control
'
ReadROM        CON     $33                     ' read ID, serial num, CRC
MatchROM       CON     $55                     ' look for specific device
SkipROM        CON     $CC                     ' skip rom (one device)
ConvertTemp    CON     $44                     ' do temperature conversion
ReadScratch    CON     $BE                     ' read DS1820 scratchpad


' ----[ Variables ]------------------------------------------------------
'
idx            VAR     Byte                    ' loop counter
romData        VAR     Byte(8)                 ' ROM data from DS1820
tempIn         VAR     Word                    ' raw temperature
sign           VAR     tempIn.Bit8             ' 1 = negative temperature
tInLow         VAR     tempIn.LowByte
tInHigh        VAR     tempIn.HighByte
tSign          VAR     Bit
tempC          VAR     Word                    ' Celsius
tempF          VAR     Word                    ' Fahrenheit


' ----[ EEPROM Data ]----------------------------------------------------
'


' ----[ Initialization ]-------------------------------------------------
'
LCD_Setup:
  LCDCMD LCDpin,%00110000 : PAUSE 5         ' 8-bit mode
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00100000 : PAUSE 0         ' 4-bit mode
  LCDCMD LCDpin,%00101000 : PAUSE 0         ' 2-line mode
  LCDCMD LCDpin,%00001100 : PAUSE 0         ' no crsr, no blink
  LCDCMD LCDpin,%00000110                   ' inc crsr, no disp shift

' ----[ Main Code ]------------------------------------------------------
'
Main:
  LCDOUT LCDpin,ClrLCD,["BSP <---> DS1820"] ' splash screen
  PAUSE 2000


DisplayROM:
  LCDOUT LCDpin,ClrLCD,["DS1820 ROM:"]
```

```
  OWOUT DS1820pin,1,[ReadROM]                ' send Read ROM command
  OWIN  DS1820pin,2,[STR romData\8]          ' read serial number & CRC
  LCDCMD LCDpin,Line2
  FOR idx = 0 TO 7
    LCDOUT LCDpin,NoCmd,[HEX2 romData(idx)] ' show ID, serial num, CRC
  NEXT

  PAUSE 5000


TempDemo:
  LCDOUT LCDpin,ClrLCD,["CURRENT TEMP:"]

ShowTemp:

  ' * send conversion command
  ' * allow time for conversion
  ' * send read scratch ram command
  ' * grab the data

  OWOUT DS1820pin,1,[SkipROM,ConvertTemp]   ' start conversion
  PAUSE 500                                 ' give time for conversion
  OWOUT DS1820pin,1,[SkipROM,ReadScratch]
  OWIN  DS1820pin,2,[tInLow,tInHigh]        ' read temperature

  tSign = sign                              ' save sign bit
  tempIn = tempIn/2                         ' round to whole degrees
  IF tSign = 0 THEN NoNeg1
  tempIn = tempIn | $FF00                   ' extend sign bits for negs

NoNeg1:
  tempC = tempIn                            ' save Celsius value
  tempIn = tempIn */ $01CC                  ' multiply by 1.8
  IF tSign = 0 THEN NoNeg2                  ' if neg, extend sign bits
  tempIn = tempIn | $FF00

NoNeg2:
  tempF = tempIn+32                         ' finish C -> F conversion

  ' display temps
  '
  LCDOUT LCDpin,Line2,[SDEC tempC, DegSym, " C"]
  LCDOUT LCDpin,NoCmd,[" / ", SDEC tempF, DegSym, " F"]
  LCDOUT LCDpin,NoCmd,[REP " "\6]

  PAUSE 1500
  GOTO ShowTemp                                    ' update temp display
```

```
' Program Listing 68.2
' Nuts & Volts - December 2000

' ----[ Title ]-----------------------------------------------------------
'
' File...... DS1820-x.BSP
' Purpose... BASIC Stamp SX Plus <--> Multiple DS1820s Demo
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 04 NOV 2000
' Updated... 05 NOV 2000


' ----[ Program Description ]---------------------------------------------
'
' This program reads and displays the temperature from multiple DS1820
' (1-wire) sensors.
'
' Program requires 2x16 LCD
'   - LCD.E    --> Pin0 (pulled down [to ground] through 4.7K)
'   - LCD.R/W  --> Pin2 (or grounded for write-only operation)
'   - LCD.RS   --> Pin3
'   - LCD.D4   --> Pin4
'   - LCD.D5   --> Pin5
'   - LCD.D6   --> Pin6
'   - LCD.D7   --> Pin7


' ----[ Revision History ]------------------------------------------------
'


' ----[ I/O Definitions ]-------------------------------------------------
'
LCDpin          CON     0                       ' data on pins 4 - 7
DS1820pin       CON     11


' ----[ Constants ]-------------------------------------------------------
'
' LCD control characters
'
NoCmd           CON     $00                     ' just print
ClrLCD          CON      $01                    ' clear the LCD
CrsrHm          CON      $02                    ' cursor home
CrsrLf          CON      $10                    ' cursor left
CrsrRt          CON      $14                    ' move cursor right
DispLf          CON      $18                    ' shift display left
DispRt          CON      $1C                    ' shift displayright
DDRam           CON      $80                    ' Display Data RAM control
Line1           CON     $80                     ' address of line 1
```

```
Line2          CON     $C0                     ' address of line 2

DegSym         CON     223                     ' degrees symbol

' DS1820 control
'
ReadROM        CON     $33                     ' read ID, serial num, CRC
MatchROM       CON     $55                     ' look for specific device
SkipROM        CON     $CC                     ' skip rom (one device)
ConvertTemp    CON     $44                     ' do temperature conversion
ReadScratch    CON     $BE                     ' read DS1820 scratchpad

NumSensors     CON     3                       ' number of DS1820s

' ----[ Variables ]------------------------------------------------------
'
sensor         VAR     Byte                    ' sensor number to process
idx            VAR     Byte                    ' loop counter
eeAddr         VAR     Byte                    ' ee address of ROM match
romData        VAR     Byte(8)                 ' ROM data to DS1820
tempIn         VAR     Word                    ' raw temperature
sign           VAR     tempIn.Bit8             ' 1 = negative temperature
tInLow         VAR     tempIn.LowByte
tInHigh        VAR     tempIn.HighByte
tSign          VAR     Bit
tempC          VAR     Word                    ' Celsius
tempF          VAR     Word                    ' Fahrenheit


' ----[ EEPROM Data ]-----------------------------------------------------
'
                       ' ROM codes for connected sensors

Temp1          DATA    $10,$F1,$67,$45,$00,$00,$00,$9F
Temp2          DATA    $10,$81,$CF,$45,$00,$00,$00,$0A
Temp3          DATA    $10,$C9,$70,$45,$00,$00,$00,$FE


' ----[ Initialization ]--------------------------------------------------
'
LCD_Setup:
  LCDCMD LCDpin,%00110000 : PAUSE 5         ' 8-bit mode
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00100000 : PAUSE 0         ' 4-bit mode
  LCDCMD LCDpin,%00101000 : PAUSE 0         ' 2-line mode
  LCDCMD LCDpin,%00001100 : PAUSE 0         ' no crsr, no blink
  LCDCMD LCDpin,%00000110                   ' inc crsr, no disp shift


' ----[ Main Code ]-------------------------------------------------------
```

```
'
Main:
  LCDOUT LCDpin,ClrLCD,["BSP <---> DS1820"] ' splash screen
  LCDOUT LCDpin,Line2, ["Multiple Sensors"]
  PAUSE 2000
  LCDCMD LCDpin,ClrLCD


ShowDS1820s:
  FOR sensor = 1 TO NumSensors              ' cycle through sensors
    eeAddr = Temp1+(8*(sensor-1))           ' point to ROM code
    GOSUB GetTemp
    LCDOUT LCDpin,Line1,["Sensor ",("0"+sensor),":"]
    PAUSE 1500
  NEXT

  GOTO ShowDS1820s
  END


' ----[ Subroutines ]-----------------------------------------------------
'

GetTemp:
  FOR idx = 0 TO 7                          ' load ROM pattern
    READ (eeAddr+idx),romData(idx)
  NEXT

  ' * send conversion command
  ' * allow time for conversion
  ' * send read scratch ram command
  ' * grab the data

  OWOUT DS1820pin,1,[MatchROM,STR romData\8,ConvertTemp]
  PAUSE 500
  OWOUT DS1820pin,1,[MatchROM,STR romData\8,ReadScratch]
  OWIN  DS1820pin,2,[tInLow,tInHigh]

  tSign = sign                              ' save sign bit
  tempIn = tempIn/2                         ' round to whole degrees
  IF tSign = 0 THEN NoNeg1
  tempIn = tempIn | $FF00                   ' extend sign bits for negs

NoNeg1:
  tempC = tempIn                            ' save Celsius value
  tempIn = tempIn */ $01CC                  ' multiply by 1.8
  IF tSign = 0 THEN NoNeg2                  ' if neg, extend sign bits
  tempIn = tempIn | $FF00

NoNeg2:
  tempF = tempIn+32                         ' finish C -> F conversion
```

```
ShowTemps:
  LCDOUT LCDpin,Line1+10,[SDEC tempC, DegSym, " C   "]
  LCDOUT LCDpin,Line2+10,[SDEC tempF, DegSym, " F   "]

  RETURN
```

```
' Program Listing 68.3
' Nuts & Volts - December 2000

' ----[ Title ]-------------------------------------------------------------
'
' File...... PCF8583.BSP
' Purpose... BASIC Stamp SX Plus <--> Philips PCF8583 (I2C RTC/RAM)
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 05 NOV 2000
' Updated... 05 NOV 2000


' ----[ Program Description ]-----------------------------------------------
'
' This program demonstrates the I2C routines of the BSP by connecting to
' a Philips PCF8583 RTC/RAM chip.
'
' Program requires 2x16 LCD
'   - LCD.E    --> Pin0 (pulled down [to ground] through 4.7K)
'   - LCD.R/W  --> Pin2 (or grounded for write-only operation)
'   - LCD.RS   --> Pin3
'   - LCD.D4   --> Pin4
'   - LCD.D5   --> Pin5
'   - LCD.D6   --> Pin6
'   - LCD.D7   --> Pin7


' ----[ Revision History ]--------------------------------------------------
'


' ----[ I/O Definitions ]---------------------------------------------------
'
LCDpin         CON    0                    ' data on pins 4 - 7
I2Csda         CON    8                    ' sda pin (scl on 9)
BtnsIn         VAR    InD                  ' button input


' ----[ Constants ]---------------------------------------------------------
'
```

```
' LCD control characters
'
NoCmd          CON     $00                    ' just print
ClrLCD         CON      $01                   ' clear the LCD
CrsrHm         CON      $02                   ' cursor home
CrsrLf         CON      $10                   ' cursor left
CrsrRt         CON      $14                   ' move cursor right
DispLf         CON      $18                   ' shift display left
DispRt         CON      $1C                   ' shift displayright
DDRam          CON      $80                   ' Display Data RAM control
Line1          CON     $80                    ' address of line 1
Line2          CON     $C0                    ' address of line 2

' PCF8583 control
'
RTCwrite       CON     %10100000              ' write address
RTCread        CON     %10100001              ' read address

Yes            CON     1
No             CON     0


' ----[ Variables ]------------------------------------------------------
'
seconds        VAR     Byte
minutes        VAR     Byte
hours          VAR     Byte
day            VAR     Nib                    ' 0 - 6 (day of week)
date           VAR     Byte                   ' 1 - 31
month          VAR     Nib
year           VAR     Nib                    ' 0 - 3 (LeapYear offset)

rawTime        VAR     Word                   ' minutes past midnight

regCtrl        VAR     Byte                   ' [0] control/status
regHuns        VAR     Byte                   ' [1] hundredths (bcd)
regSecs        VAR     Byte                   ' [2] seconds (bcd)
regMins        VAR     Byte                   ' [3] minutes (bcd)
regHrs         VAR     Byte                   ' [4] hours (bcd)
regYrDate      VAR     Byte                   ' [5] year & date (bcd+)
regMoDay       VAR     Byte                   ' [6] day & month (bcd+)

btns           VAR     Nib                    ' button inputs
btnMin         VAR     btns.Bit0              ' update minutes
btnHrs         VAR     btns.Bit1              ' update hours
btnDay         VAR     btns.Bit2              ' update day
btnBack        VAR     btns.Bit3              ' go backward

regAddr        VAR     Byte                   ' register address
regData        VAR     Byte                   ' data to/from register
```

```
eeAddr          VAR     Byte                    ' EE data pointer
char            VAR     Byte                    ' character from EE
idx             VAR     Byte                    ' loop counter

' ----[ EEPROM Data ]-------------------------------------------------
'
Su              DATA    "SUNDAY",0
Mo              DATA    "MONDAY",0
Tu              DATA    "TUESDAY",0
We              DATA    "WEDNESDAY",0
Th              DATA    "THURSDAY",0
Fr              DATA    "FRIDAY",0
Sa              DATA    "SATURDAY",0


' ----[ Initialization ]-----------------------------------------------
'
LCD_Setup:
  LCDCMD LCDpin,%00110000 : PAUSE 5         ' 8-bit mode
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00110000 : PAUSE 0
  LCDCMD LCDpin,%00100000 : PAUSE 0         ' 4-bit mode
  LCDCMD LCDpin,%00101000 : PAUSE 0         ' 2-line mode
  LCDCMD LCDpin,%00001100 : PAUSE 0         ' no crsr, no blink
  LCDCMD LCDpin,%00000110                   ' inc crsr, no disp shift


' ----[ Main Code ]----------------------------------------------------
'
Main:
  LCDOUT LCDpin,ClrLCD,["BSP <--> PCF8583"] ' splash screen
  LCDOUT LCDpin,Line2, ["  (RTC + RAM)"]
  PAUSE 2000
  LCDCMD LCDpin,ClrLCD

Loop:
  GOSUB GetTimeAndDay                       ' get and display clock
  GOSUB PrintDay
  LCDOUT LCDpin,(Line2 + 8),[DEC2 hours,":",DEC2 minutes,":",DEC2 seconds]

  GOSUB GetButtons                          ' any buttons pressed?
  IF btns = %0000 THEN NoUpdate             ' nope
  IF btnBack = Yes THEN GoBack              ' back button pressed?


GoForward:
  rawTime = rawTime + btnMin                ' add one minute
  rawTime = rawTime + (btnHrs * 60)         ' add one hour
  day = (day + btnDay) // 7                 ' next day
  GOTO UpdateClock
```

```
GoBack:
  IF (btns <= %1000) THEN NoUpdate          ' no update button pressed
  rawTime = rawTime + (btnMin * 1439)       ' subtract one minute
  rawTime = rawTime + (btnHrs * 1380)       ' subtract one hour
  day = (day + (btnDay * 6)) // 7           ' previous day


UpdateClock:                                ' send update to RTC
  rawTime = rawTime // 1440                 ' clean-up time mods
  GOSUB PutRawClock                         ' set clock with rawTime
  GOTO NoDelay


NoUpdate:
  PAUSE 100                                 ' pad loop
NoDelay:                                    ' skip pad on updates
  GOTO Loop

  END


' ----[ Subroutines ]------------------------------------------------------
'
PutRegister:
  I2COUT I2Csda,RTCwrite,regAddr,[regData]  ' send data to register
  RETURN


GetRegister:
  I2CIN I2Csda,RTCread,regAddr,[regData]    ' get data from register
  RETURN


PutRawClock:                                ' set with rawTime
  minutes = rawTime // 60
  hours = rawTime / 60

PutClock:
  regSecs = 0
  regMins.HighNib = minutes / 10            ' convert regs to BCD
  regMins.LowNib = minutes // 10
  regHrs.HighNib = hours / 10
  regHrs.LowNib = hours // 10
  regMoDay.HighNib = month / 10
  regMoDay.LowNib = month // 10
  regMoDay = regMoDay + (day << 5)          ' pack weekday in
  I2COUT I2Csda,RTCwrite,2,[STR regSecs\5]  ' write time & day
  RETURN
```

```
GetTimeAndDay:
  I2CIN I2Csda,RTCread,0,[STR regCtrl\7]

  ' convert from BCD

  seconds = (regSecs.HighNib * 10) + regSecs.LowNib
  minutes = (regMins.HighNib * 10) + regMins.LowNib
  hours = (regHrs.HighNib * 10) + regHrs.LowNib
  rawTime = (hours * 60) + minutes
  day = regMoDay >> 5
  RETURN


PrintDay:
  LOOKUP day,[Su,Mo,Tu,We,Th,Fr,Sa],eeAddr  ' point to EE string
  LCDCMD LCDpin,Line1                       ' move to LCD Line1
PrintLoop:
  READ eeAddr,char                          ' read a character
  IF char = 0 THEN PrintDone                ' done?
  LCDOUT LCDpin,NoCmd,[char]                ' print the character
  eeAddr = eeAddr + 1                       ' point to next
  GOTO PrintLoop:                           ' go get it
PrintDone:
  LCDOUT LCDpin,NoCmd,[REP " "\5]           ' spaces for clean-up
  RETURN


GetButtons:
  btns = %1111                              ' enable all button inputs
  FOR idx  = 1 TO 10
    btns = btns & ~BtnsIn                   ' test inputs
    PAUSE 5                                 ' delay between tests
  NEXT
  PAUSE 100                                 ' slow held button(s)
  RETURN
```