



Column #69, January 2001 by Jon Williams:

## There's A New Stamp In Town – Part 2

Well, we made...2001. Yippee!!! You know, there are some calendar-math fanatics out there who insist that this is the year that actually begins the new millennium. I don't know about that, but what I do know is that this is the year of the BASIC Stamp2-SX Plus: the BS2p. All millennium-hype aside, this is the best new Stamp in years.

Last month we introduced the BS2p and some of its really great features: LCD support, Dallas 1-Wire™ support, I2C...big improvements over the BS2sx and BS2e. And thanks to some astute beta testers and the hard work of the Parallax engineering staff (specifically, Chuck Gracey), it's actually been improved even more. Don't worry, the code is frozen now and about the time you're reading this article, the BS2p should be on the streets and making a lot of Stamp programmers very very happy.

Last month we talked about LCDs, 1-Wire™ and I2C. This month we'll talk about the updates, pin polling (firmware interrupts) and how to use the 40-pin BS2p.

Figure 69.1: Button hookups for sample code

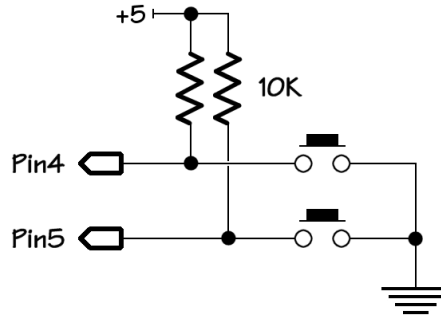
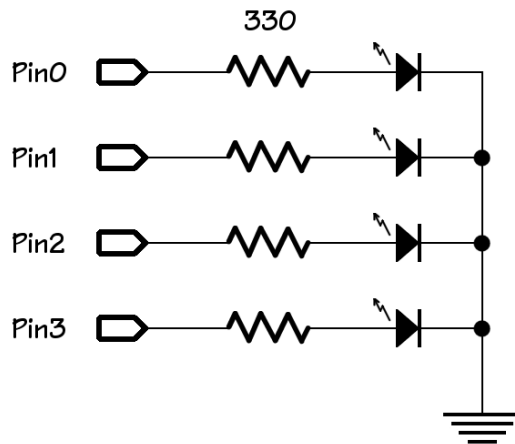


Figure 69.2: LED hookups for sample code



### 1-Wire Update

The Dallas 1-Wire™ routines have been updated to allow bit-level support and high-speed communications. We can use the bit-mode capability with the DS1820 to determine when the conversion is actually done instead of waiting for some default time (500 milliseconds is suggested by Dallas). Jeff Martin at Parallax showed me this trick and it turns out that the DS1280 will actually convert the temperature in about 225 to 250 milliseconds. I know that 250 milliseconds doesn't sound like a big savings when dealing with one sensor, but on a busy 1-Wire network it could be valuable.

Take a look Program Listing 69.1 (an update from last month). Aside from adding some useful constants there is a new routine called WaitForConversion. After the ConvertTemp has been sent, we can use bit mode access to determine when the conversion is done. When we read back a 1, the conversion is complete and we can move on to retrieving the newly measured temperature. The reason we allow 25 milliseconds between checks is to let the DS1820 finish the temperature conversion.

You may have noticed that OWIN and I2CIN do not have a time-out parameter in their syntax. The BS2p defines this internally so it won't "hang" if no input is returned from a device.

### Effective Use Of EEPROM Space

Between last month and this one a new command has been added: STORE. This syntax is simple:

STORE *location*

Location is the program slot (0 to 7) to be used as the target for READ and WRITE. The default location number is the same as the current program and gets changed to the new program slot whenever the RUN command is issued. The neat thing about STORE is that now we can take advantage of unused program slots and treat them like one large, flat EEPROM block.

Program Listing 69.2 is a short demo that shows how to take advantage of STORE and all the available EE space in the BS2p. The routines in this program would be useful in a data logging application. For our demo though, all we need to do is connect to the BS2p.

At the top is a constant called FirstSlot. This sets the first available program slot to be used as general-purpose storage space. The demo assumes that program slot 0 will be used for main code and program slot 1 will be used for subroutines. This leaves program slots 2 through 7 (12,288 bytes) available for storage. Of course, you can adjust this for your own needs.

Now that we know where we can start storing data, we can calculate how much is available. This is important so that we don't overwrite a location or read back the wrong one. Both subroutines check to make sure that we are attempting to write to or read back a valid address.

## Column #69: There's a New Stamp in Town - Part 2

The program slot for our “big EEPROM” is calculated by dividing the required address by 2048 (the EE size for each slot) and adding the FirstSlot constant. The offset into the correct slot is calculated by using the modulus operator (*//*). The STORE command is used to set the correct program slot and WRITE and READ are used as they normally would be.

This feature is significant and we can thank Stamp guru, Tracy Allen, for suggesting it. If the available program slots don't provide enough EEPROM for your application, remember that there are a bunch of I2C EEPROMs that then BS2p can connect to with ease.

### Pin Polling (Firmware Interrupts)

As much as we'd all like it to be, the BS2p is not capable of handling true interrupts – that's hard to do for any interpreter and the compact size of the PBASIC interpreter makes it even tougher. What the BS2p can do, however, is check on pins between PBASIC statements and take a specified action. When setup and enabled, the BS2p perform the following actions on a polled interrupt:

- Nothing
- Set an output pin to specified state
- Run another program
- Wait (pause program) until interrupt condition occurs
- Any combination of 2, 3 and 4

Let me explain again so we're absolutely clear. When setup and enabled, the BS2p will check the state of polled-input pins between each PBASIC instruction. If the specified input condition is met, the “interrupt” state is made true and specified action(s) will be taken.

To define input pins that will be polled, we'll use POLLIN. Here's the syntax:

*POLLIN pin,state*

The pin parameter will always be 0 to 15 and state will be 0 (low) or 1 (high). When activated, the specified pin(s) will be polled between PBASIC instructions.

When the interrupt state is true, polled-output pins can be controlled. Once setup, the control over these pins is automatic and follows the interrupt condition. To define output pins, we'll use POLLOUT.

POLLOUT *pin,state*

The parameters are the same as with POLLIN, except that we're controlling an output. The state will be set on the specified pin when the interrupt condition is true.

With polled-inputs and outputs defined, we'll use POLLMODE to enable them.

POLLMODE *mode*

The mode parameter will be from 0 to 15 with the following definitions:

- Deactivate polling and clear polled-inputs and outputs definition
- Deactivate polling and save polled-inputs and outputs definition
- Activate polling with polled-outputs only
- Activate polling with polled-run action only
- Activate polling with polled-outputs and polled-run action
- Clear polled-inputs configuration
- Clear polled-outputs configuration
- Clear both polled-inputs and polled outputs

Modes 8 through 15 are the same as 0 through 7 except that the interrupt condition is latched.

Here's a really simple little code snippet that will demonstrate polled-inputs and outputs.

```
Setup:
  POLLMODE 0           ' clear and disable polling
  POLLIN 4,0          ' interrupt when pin 4 is low
  POLLOUT 0,1         ' make pin 0 high on interrupt
  POLLMODE 2          ' activate polled-outputs

Loop:
  DEBUG "Waiting for interrupt",CR
  PAUSE 100
  GOTO Loop
```

When you run this code you see a DEBUG screen with the "Waiting..." message happily scrolling by. Now press and hold the button connected to pin 4. The LED connected to pin 0 will light. Now release the button and notice that LED goes out.

## Column #69: There's a New Stamp in Town - Part 2

With the ability to control outputs with polled-inputs, you can create a “background logic gate” that operates while our Stamp program is running. Add this line to the code above and notice that either pin (4 or 5) will light the LED – a background OR gate.

```
POLLIN 5,0           ' interrupt when pin 5 is low
```

The nature of the BS2p lets us control polled-output pins with standard output pins. Try this code:

```
Setup:
  POLLMODE 0           ' clear and disable polling
  POLLIN 0,0          ' interrupt when pin 0 is low
  POLLOUT 1,1         ' make pin 1 high on interrupt
  POLLMODE 2         ' activate polled outputs
Loop:
  TOGGLE 0           ' toggle LED on pin 0
  PAUSE 100
  GOTO Loop
```

Without polling defined, the only thing that would happen is that the LED on pin 0 would blink (due to the TOGGLE command). With polling active though, the LED on pin 1 will blink at the same rate as the LED on pin 0 (in this case in the opposite state).

Keep in mind that the interrupt state is a global condition and if none of the polled-input pins meet their requirements, the interrupt state will be cleared. There will be times when you want to monitor a number of pins and take a specific action based on the inputs. This can be accomplished by latching the interrupt condition and checking to see what pin(s) caused the interrupt state.

Listing 3 is a simple alarm program that latches the interrupt condition. Pins 4 and 5 are setup to create an interrupt when pulled low. When the interrupt condition is true, the Alarm LED on pin 0 is lit. In order to determine which pin(s) caused the interrupt condition, POLLMODE 10 (control outputs and latch interrupt condition) is used.

During the loop, scratchpad location 128 is read to determine any active interrupt pins. Since the interrupts have been latched, the variable *iPins* will show which pins were active during the polling cycle. With this information we can create a simple display. Once we've taken action on the interrupt, we can reset everything by re-issuing the POLLMODE command.

In the examples so far, our program spent time in a loop waiting for an interrupt to happen. With POLLWAIT, the BS2p can suspend program operation until the interrupt

condition is true. When the interrupt condition is true, the program resumes at the next line.

POLLWAIT *period*

POLLWAIT is very similar to NAP in that it puts the BS2p into low-power mode. Based on the period, the BS2p will “wake” from the low-power and scan the define polled-input pins. If any of the conditions are met, the program continues, otherwise the BS2p goes back into low-power mode. The possible values for period are:

- 18 ms
- 36 ms
- 72 ms
- 144 ms
- 288 ms
- 576 ms
- 1152 ms (1.152 seconds)
- 2304 ms (2.304 seconds)
- Wait without low-power mode

Just as with NAP, any outputs will glitch when the BS2p comes out of low-power mode (period from 0 to 7) to do its check. Try this code:

```
HIGH 3           ' turn LED on pin 3 on

POLLIN 5,0       ' scan pin 5 for low input
POLLOUT 2,1      ' light LED on pin 2 during interrupt
POLLMODE 2       ' activate polled-output control

Loop:
  POLLWAIT 0      ' low-power mode for 18 ms
  TOGGLE 1        ' toggle LED on pin 1
  GOTO Loop
```

When the program starts, the LED attached to pin 3 will light. Look carefully. Notice the apparent pulsing of this LED? This happens when the BS2p comes out of low-power mode and all pins are momentarily made inputs. You can see the change by setting the period to a different value. What you'll observe is that as the period gets larger, the “glitching” becomes less frequent, but so does the polling frequency. Holding the button will make the interrupt active through every check and the LED on pin 0 will toggle at a rate determined by the POLLWAIT period.

## Column #69: There's a New Stamp in Town - Part 2

For applications where low-power mode is not required and POLLWAIT is desirable, set the period to 8. This causes the BS2p to wait for the interrupt condition, but doesn't put it into low-power mode.

Finally, we can use POLLRUN to cause the BS2p to jump to another program slot when the interrupt condition is true.

### *POLLRUN program*

The default program slot for POLLRUN is 0. If POLLMODE 3 or 4 is issued and no POLLRUN program was defined, the BS2p will jump to program 0 on the interrupt condition. If polled-outputs are also defined and enabled (POLLMODE 4), the outputs will be set before the new program is run. POLLRUN 3 and 4 have a "one-shot" behavior to prevent the BS2p from appearing to be locked up by continuously jumping to the specified program.

## **Pins, Pins And More Pins**

You asked for them, you got 'em. The BS2p comes in a 40-pin version that gives the Stamp user an extra set of 16 I/O pins. Using the additional pins is somewhat similar to using program slots -- you have to alert the Stamp. All Stamp instructions that require pin numbers stay the same, that is, they expect a pin number from 0 to 15. What we have to do now is tell the Stamp which set of pins we want to use.

There are two ways to select the second set of I/O pins:

```
AUXIO           ' select auxiliary I/O pins
IOTERM 1        ' select auxiliary I/O pins
```

To switch back to the main I/O pins, we can use either of the following:

```
MAINIO          ' select main I/O pins
IOTERM 0        ' select main I/O pins
```

In case you're wondering, polled-pins can be defined on both sets of I/O groups and all of them are active, regardless of which set of I/O pins are in use at the time. You can even determine which of the 32 pins caused the interrupt condition by looking into the scratchpad RAM (see below).

Note that MAINIO, AUXIO and IOTERM are not available on the BS2p-24 since it only has one set of 16 I/O pins.

### Read-Only RAM Locations

As pointed out last month, the BS2p has twice as much scratchpad RAM (127 bytes) as the BS2sx. There are five bytes at the end of the scratchpad that are read-only and provide useful information

Current program (low nibble) and STORE location (high nibble)

Interrupt pin detection, main pins 0 – 7

Interrupt pin detection, main pins 8 – 15

Interrupt pin detection, auxiliary pins 0 – 7

Interrupt pin detection, auxiliary pins 8 – 15

Locations 128 – 131 can be used by your program to determine what pin(s) caused an interrupt condition. An active interrupt pin will be indicated by a 1, regardless of its input state (active-low or active-high). The bits in these locations are active only when the interrupt is active, so you may need to latch the interrupt condition to determine the cause after-the-fact.

### More To Come

To be honest, only a very lucky handful of us have had the opportunity to play with the BS2p, but with it rolling into production that will change very soon. I really think that the BS2p will become my default Stamp. Oh, don't worry, I'm not going to force you to upgrade for future projects; most of my code will be backward-compatible with the BS2. That said, there are a lot of neat 1-Wire™ and I2C parts that I'm interested in experimenting with – and I'm sure you will to.

Be sure to visit the Parallax web site ([www.parallaxinc.com](http://www.parallaxinc.com)) when you get the chance. With all the excitement surrounding the BS2p, they will be creating a separate support page for BS2p projects. Who knows, maybe they'll post one of yours....

Until next time, happy Stamping.

## Column #69: There's a New Stamp in Town - Part 2

```
' Program Listing 69.1
' Nuts & Volts - January 2000

' {$STAMP BS2p}

' ----[ Title ]-----
'
' File..... DS1820b.BSP
' Purpose... BASIC Stamp SX Plus <--> DS1820 Demo
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 04 NOV 2000
' Updated... 05 DEC 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 ]-----
'
' 05 DEC 2000 : Added status check for end-of-conversion

' ----[ I/O Definitions ]-----
'
LCDpin      CON      0          ' data on pins 4 - 7
DS1820pin   CON      9

' ----[ 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
```

Column #69: There's a New Stamp in Town - Part 2

```

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

' 1-Wire Support
'
OW_FERst    CON      %0001    ' Front-End Reset
OW_BERst    CON      %0010    ' Back-End Reset
OW_BitMode  CON      %0100
OW_HighSpd  CON      %1000

ReadROM     CON      $33      ' read ID, serial num, CRC
MatchROM    CON      $55      ' look for specific device
SkipROM     CON      $CC      ' skip ROM (one device)
SearchROM   CON      $F0      ' search

' DS1820 control
'
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

```

## Column #69: There's a New Stamp in Town - Part 2

```
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
  ' * check for conversion complete
  ' * send read scratch ram command
  ' * grab the temperature

  OWOUT DS1820pin,OW_FERst,[SkipROM,ConvertTemp]

WaitForConversion:
  PAUSE 25
  OWIN DS1820pin,OW_BitMode,[tempIn]
  IF tempIn = 0 THEN WaitForConversion

  OWOUT DS1820pin,OW_FERst,[SkipROM,ReadScratch]
  OWIN  DS1820pin,OW_BERst,[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
```

## Column #69: There's a New Stamp in Town - Part 2

```
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]
GOTO ShowTemp                ' update temp display
```

```
' Program Listing 69.2
' Nuts & Volts - January 2000

' {$STAMP BS2p}

' ----[ Title ]-----
'
' File..... STORE.BSP
' Purpose... Uses STORE to create large, flat EE space
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 05 NOV 2000
' Updated... 05 DEC 2000

' ----[ Program Description ]-----
'
' This program demonstrates the use of STORE to create a flat EEPROM space
' from program slots 2 - 7 (12,000 bytes). Program slot 1 is reserved for
' subroutines.

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

' ----[ I/O Definitions ]-----
'

' ----[ Constants ]-----
'
FirstSlot      CON      2                ' start with pgm slot 2
MaxAddr        CON      8 - FirstSlot * 2048 - 1
```

## Column #69: There's a New Stamp in Town - Part 2

```
' ---- [ Variables ]-----
'
eeAddr      VAR      Word      ' flat EE address
eeData      VAR      Byte
slot        VAR      Nib       ' pgm slot for storage
addr        VAR      Word      ' address for storage

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

' ---- [ Initialization ]-----
'
eeAddr = 4090

' ---- [ Main Code ]-----
'
Main:
  DEBUG Home,"BS2p STORE Demonstration",CR
  DEBUG "-----",CR,CR

  DEBUG "Big EE Size: ", DEC MaxAddr + 1, " bytes",CR,CR

  RANDOM eeData                      ' create data
  GOSUB WriteBigEE                    ' write to flat EE

  DEBUG "eeAddr..... ",DEC eeAddr,CR
  DEBUG "slot.....  ",DEC slot,CR
  DEBUG "addr.....  ",DEC addr,CR,CR
  DEBUG "data out... ", DEC eeData,"  ",CR

  GOSUB ReadBigEE                     ' get data from flat EE

  DEBUG "data in.... ", DEC eeData,"  "

  eeAddr = eeAddr + 1 // MaxAddr
  PAUSE 250
  GOTO Main

' ---- [ Subroutines ]-----
'
WriteBigEE:
  IF (eeAddr > MaxAddr) THEN NoWrite ' check for bad eeAddr
  slot = (eeAddr / 2048) + FirstSlot ' calc pgm slot
  addr = eeAddr // 2048              ' calc address in slot
  STORE slot
  WRITE addr,eeData
NoWrite:
  RETURN
```

```

ReadBigEE:
  IF (eeAddr > MaxAddr) THEN NoWrite          ' check for bad eeAddr
  slot = (eeAddr / 2048) + FirstSlot          ' calc pgm slot
  addr = eeAddr // 2048                       ' calc address in slot
  STORE slot
  READ addr,eeData
NoRead:
  RETURN

```

```

' Program Listing 69.3
' Nuts & Volts - January 2000

' {$STAMP BS2p}

' ----[ Title ]-----
'
' File..... ALARM.BSP
' Purpose... Simple alarm system demo using BS2p pin polling
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 05 DEC 2000
' Updated... 05 DEC 2000

' ----[ Program Description ]-----
'
' This program demonstrates BS2p polled-inputs and polled-output control.
' Two pins are monitored and will control an output (alarm) pin. The
' interrupt condition is latched to determine the cause.

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

' ----[ I/O Definitions ]-----
'
AlarmLED      CON      0
FrontDoor     CON      4          ' front door input
BackDoor      CON      5          ' back door input

' ----[ Constants ]-----
'
On             CON      1          ' LED is active high

```

## Column #69: There's a New Stamp in Town - Part 2

```
' ---- [ Variables ]-----  
,  
iPins          VAR      Byte          ' interrupt detect pins  
  
' ---- [ EEPROM Data ]-----  
,  
  
' ---- [ Initialization ]-----  
,  
Setup:  
  POLLMODE 0          ' clear and disable polling  
  POLLIN FrontDoor,0  ' interrupt when inputs low  
  POLLIN BackDoor,0  
  POLLOUT AlarmLED,On ' alarm LED on interrupt  
  POLLMODE 10        ' latch interrupt condition  
  
' ---- [ Main Code ]-----  
,  
Loop:  
  DEBUG Home,"Monitoring... "  
  GET 128,iPins          ' get cause of  
interrupt  
  iPins = iPins >> 4    ' move to lower nibble  
  IF (iPins = 0) THEN Loop ' no alarms  
  
  DEBUG CR,CR          ' alarm screen  
  DEBUG "Alarm(s) Detected",CR  
  DEBUG "-----",CR  
  DEBUG "Front Door: ", DEC iPins.Bit0,CR  
  DEBUG "Back Door:  ", DEC iPins.Bit1,CR  
  
  PAUSE 3000          ' give time for display  
  DEBUG CLS  
  POLLMODE 10        ' reset polling  
  GOTO Loop  
  
' ---- [ Subroutines ]-----  
,
```