



Column #54, October 1999 by Lon Glazner:

Faster, Stronger, Better: The BASIC Stamp 2-SX

This article is less an application than it is a primer describing the differences between the BASIC Stamp 2 (BS2) and the BS2-SX. There are some minor hardware differences, a few new commands, and some timing differences in a handful of the older BS2 commands. But, in general, the learning curve to upgrade from the BS2 to the BS2-SX could be best described as a gently sloping incline.

I've been somewhat remiss in not writing a column on the BASIC Stamp 2-SX (BS2-SX) that was released by Parallax quite a few months ago (actually unveiled November 1998). I was really waiting for an application that required the additional speed and program memory available in this new member of the Stamp family. Of course, I tailor my applications in this column so that they fit into a format that can be well described in just a few pages. It dawned on me this month that that process would likely prevent me from defining applications that "required" a BS2-SX.

So, I'm making it a point to describe Parallax's BS2-SX in this article, and use it for applications in the next couple of issues of Nuts & Volts Stamp Applications.

Defining The Design

As I mentioned earlier, this article shouldn't be considered a design in and of itself. But it is the jumping-off point for an application making use of some of the BS2-SX features. Next month, we'll be going into an in-depth application called StampNet. StampNet will cover some of the intricacies of a well-defined RS-485 network. This will include electrical requirements, as well as a Master-Slave communication protocol. The StampNet design will be a generic overview that describes a multi-node network. Many electronic disciplines will be touched on during the StampNet application, and the BS2-SX will be the heart of this system.

What's Different

So what's the big difference between the BS2 and BS2-SX? On first glance I'd have to say the color! Yep, the BS2-SX comes with a new shiny blue paint job. In fact, it is somewhat reminiscent of the midnight-metallic-blue paint job of the '71 Mustang that I had in high school. But, like the Mustang, what really counts is what is under the hood.

But on the electronics front, I would have to say that the most significant difference between the BS2 and the BS2-SX is the Scenix Semiconductor SX28AC that Parallax designed into the BS2-SX. This processor can operate with a much faster oscillator than the previous Microchip PIC16C57 based BS2. With this speed increase of 250% comes an operating current increase of 750%. But we all knew that no improvement comes without a price. Our price is increased power dissipation, and additional operating current.

One of the easiest ways to visualize all of the differences between the BS2 and the BS2-SX is with a side-by-side comparison. So, in Figure 54.1, it is in tabular form. There are three major improvements that warrant discussion with regards to the BS2-SX. The first is the massive increase in memory available for program storage. There are now eight 2K byte blocks available for program storage. This huge increase in program memory does have one drawback. The memory is not contiguous. In other words, these memory blocks can not be crossed with GOTO or GOSUB commands (or straight line coding). Each 2K byte block should be considered a separate program, and is defined as such in the BS2-SX manual.

This is not to say that you can not fully utilize the eight separate blocks of program memory. The new RUN command allows you to switch between the eight different program blocks. This aspect of the BS2-SX is described in more detail in the New Commands and Code Example sections which follow.

A second difference exhibited by the BS2-SX is also related, to a degree, to the program memory block requirements. The BS2-SX has 63 bytes of user-accessible scratch pad RAM (really 64, but see the asterisk below Figure 54.1). This scratch pad RAM is somewhat different, and in addition to, the 32 bytes of variable RAM available in the BS2. As previously stated, each program memory block (0-7) is considered a separate program accessed by the RUN command. You may pass variable data between programs. One method of doing this is to ensure that your variable definitions are defined exactly the same for all programs that are accessed. Another, more eloquent, method of doing this involves the use of the scratch pad RAM.

Figure 54.1: Modifications Exhibited by BS2-SX in BS2 Commands

Command	Change
COUNT	Period variable/constant is now in time units of 0.4ms
DTMFOUT	On-time-Off-time variable/constant is now in time units of 0.4ms
FREQOUT	Duration variable/constant is now in time units of 0.4ms Freq1,2 variable/constant is now in frequency units of 2.5Hz
PULSIN	Result variable is now loaded with time units of 0.8us
PULSOUT	Time variable is now in time units of 0.8us
PWM	Cycles variable/constant is now in time units of 0.4ms
RCTIME	Result variable is now loaded with time units of 0.8us
SERIN	bit period = INT(2,500,000/Baud Rate)-20
SEROUT	bit period = INT(2,500,000/Baud Rate)-20
SHIFTIN	data is clocked in at 42KHz
SHIFTOUT	data is clocked out at 42KHz

Figure 54.2: Baud rate conversion table for BS2-SX

Baud Rate	8N1 inverted	7E1 inverted	8N1	7E1
600	20530	28722	4146	12338
1200	18447	26639	2063	10255
2400	17405	25597	1021	9213
4800	16884	25076	500	8692
9600	16624	24816	240	8432
19200	16494	24686	110	8302
38400	16429	24621	45	8237

Column #54: Faster, Stronger, Better: The BASIC Stamp 2-SX

Using the GET and PUT commands, byte-sized pieces of data may be stored or retrieved from the scratch pad RAM. For instance, if program 0 needs to pass two byte-sized variables to program 6, you would use the PUT command to store each byte and then execute a RUN 6. Program 6 would define the variables that it needs, and then use the GET command to retrieve the data stored in scratch pad RAM.

While on the subject of memory storage, I should also touch on the use of the READ and WRITE commands which are used to store data in unused portions of EEPROM (program memory) in the BS2 and BS2-SX. These commands still function as they did in the BS2. But each of the eight (0-7) possible programs in the BS2-SX can only access EEPROM that is unused in their own 2K byte section of program memory.

Therefore, you could not power up, store data in an EEPROM location, execute a RUN command, and then retrieve the same data from EEPROM while in your second program. The data desired would be in a different 2K byte block of EEPROM, and would be accessible only from the program that originally stored it.

Even if you could pass data from program to program in the BS2-SX, you would still have to take into consideration the limited number of writes available in EEPROM, as well as the potential effect on program execution when accessing program memory EEPROM. The short of it is that these problems are avoided with the introduction of scratch pad RAM.

Finally, the third improvement is speed. This baby is 2.5X faster than its predecessor, the BS2.

New Commands

Previously, I touched on the three new commands that are related to the new program memory allocation in the BS2-SX. Here we'll talk about them in greater detail. As a refresher, the GET and PUT commands are used to retrieve and store data in the scratch pad RAM. The RUN command is used to execute any of the programs residing in the BS2-SX program memory. Program 0 is always accessed on power up or after a reset. Additionally, the scratch pad RAM is always initialized to zero on power up or after a reset. The GET command takes the form ...

```
GET    location,variable
```

where location is the memory location 0-63 that you want to read from. This may be a variable or constant. Scratch pad RAM location 63 is a read only byte which tells the "reader" which program (0-7) is currently active. The variable defined for the GET

command is the variable in which you wish to store the value from the scratch pad RAM. An example of using the GET command to read scratch pad RAM address 5, and store it in a variable named Stuff would be ...

```
Stuff var byte

GET 5,Stuff
DEBUG  "Stuff = ",DEC Stuff,CR
END
```

The PUT command is used to write data to scratch pad RAM. The PUT command takes the form ...

```
PUT    location,value
```

where location is the memory location 0-62 that you want to write to. This may be a variable or constant. The value specified in the PUT command must be a byte-sized value (0-255), and may be a constant or variable. An example of using the PUT command to write the contents of a variable register called Stuff to scratch pad address 6 would be ...

```
Stuff      var    byte

PUT        6,Stuff
END
```

Using a “location” value greater than the specified 0-63 for either the GET or PUT command will cause a rollover (or wrap) internal to the BS2-SX. In other words, a location of 64 will access scratch pad RAM location 0.

```
'Program 0
Condition    var    byte           'Define condition variable
Condition    = 1                'Condition defaults to 1

if IN15 = 1 then No_Change       'P15 state determines Condition
Condition    = 0                'If P15 is low then Condition is 0
No_Change:
  PUT        0,Condition         'Store condition in scratch pad RAM
  RUN        1                  'Execute program 1
  END
```

Column #54: Faster, Stronger, Better: The BASIC Stamp 2-SX

```
'Program 1
Point          var      byte          'Program LED state

      GET          0,Point          'Retrieve Condition variable
      if Point = 0 then LED_off      'Test condition
      HIGH         14              'Light LED
      GOTO         Done
LED_off
      LOW          14              'Extinguish LED
Done:
      RUN          0              'Execute program 0
      END
```

The RUN command is used to switch between any of the eight different program blocks which may reside in the BS2-SX. Program 0 is always accessed on power up. It is important to note that you can only access the beginning of any individual program stored in the BS2-SX program memory. While this precludes program jumps similar to those implemented by GOSUB commands, you can store condition values in scratch pad RAM. Condition values can be used to direct a newly accessed program to specific subroutines within that program. So, with a little house-keeping, the non-contiguous nature of the BS2-SX program memory is not a major roadblock. In Listing 1, the PUT, GET, and RUN commands are used to create a conditional jump.

Old Commands That Have Changed

All of the original 36 commands residing in the BS2 have been incorporated into the BS2-SX. While all of these commands function in the same manner, 11 of the commands do exhibit timing differences due to the change in operating speeds between the BS2 and the speedy BS2-SX. These changes are listed in an abbreviated form in Figure 54.2.

For most BS2 users, these changes will only create minor modifications in old BS2 code when porting it over to the new BS2-SX. Of course, if you've got code running on a BS2, and it's doing the job, there is no reason for a conversion. I listed these changes simply to give someone familiar with the BS2 some insight into differences they might expect when using the BS2-SX for the first time.

I seem to find myself making use of the Stamp's SERIN and SEROUT commands more often than any other instruction. I'm assuming that that may be true of a lot of other Stamp users as well. For that reason, I've included the SERIN/SEROUT baud rate conversions in Figure 54.2; 8N1 is defined as 8 data bits, no parity, and 1 stop bit; 7E1 is similarly defined as 7 data bits, even parity, and 1 stop bit. Inverted data is data defined by a logic high start bit.

Column #54: Faster, Stronger, Better: The BASIC Stamp 2-SX

```
'*****  
'Program 0 - bsx_prg0.bsx  
'{$STAMP BS2SX,C:\Parallax\bsx_prg1.bsx,C:\Parallax\bsx_prg2.bsx}  
'0:bsx_prg0.bsx  
  
Timer          var    byte          'Define constants/variables  
  
    GET          0,Timer          'Get old Timer  
    Timer        = Timer + 10      'Modify Timer  
    DEBUG "Timer = ",DEC Timer,CR  'Display Timer  
    PUT          0,Timer          'Store new Timer  
    RUN          1                'Execute program 1  
    END  
,
```

```
'*****  
'Program 1 - bsx_prg1.bsx  
    Grn_LED con  15                'Define constants/variables  
    Red_LED con  14  
    Timer        var    byte  
  
    GET          0,Timer          'Get Timer  
    LOW          Red_LED          'Turn off LEDs  
    LOW          Grn_LED  
    PAUSE Timer  2                'Pause  
    RUN          2                'Execute program 2  
    END  
,
```

```
'*****  
'Program 2 - bsx_prg2.bsx  
    Grn_LED con  15                'Define constants/variables  
    Red_LED con  14  
    Timer        var    byte  
  
    GET          0,Timer          'Get Timer  
    HIGH         Red_LED          'Turn on LEDs  
    HIGH         Grn_LED  
    PAUSE Timer  0                'Pause  
    RUN          0                'Execute program 0  
    END  
,
```

Code Example

While developing these code fragments, I made use of the Parallax BASIC Stamp Win Interface v1.091. This is a beta version of a Windows 95/98/NT 4.0 interface with many features. This, and older programming software (such as DOS versions), can be downloaded from the Parallax web site at no charge (www.parallaxinc.com).

This beta software did not appear to be too buggy, although the programs I was working with were extremely short. Since this was my first attempt at using the BS2-SX with the Windows interface that Parallax provides, I felt it was best to keep things short and sweet.

My BS2-SX was connected to the PC via a Parallax Board of Education for this software test, but any interface hardware that works with a BS2 should work with a BS2-SX. I wrote three programs which made use of the GET, PUT, and RUN commands to blink a couple of LEDs. The rate of the blinking was based on a variable that was loaded by programs 1 and 2. This variable was modified by program 0, which caused the rate of blinking to cycle from fast to slow.

I couldn't find any well-documented examples of loading multiple programs into the BS2-SX with the Windows Interface software. But from the READ.ME file, and through trial and error, I was able to get all three programs to load whenever I loaded program 0. Over the next few weeks, I'll be contacting Parallax to get the low-down on this process, check for updated documentation, and see if any new information is available. I'll be sure to include whatever information, or corrections, I receive in the next Stamp Applications article.

The trick to linking all of your programs appears to be through the use of the `{$STAMP BS2SX}` directive. This directive takes the form ...

```
{$STAMP BS2SX,file1,file2,...,file7}
```

where each file is the actual path and file name that belongs to your BS2-SX project. My programs were all stored in the C:\Parallax directory. If your files are stored elsewhere, you will want to select the correct path for your directory structure.

You will also wish to set the software up to communicate with a BS2-SX. This can be done through the EDIT/PREFERENCES/EDITOR OPERATION menus. Just select the correct communication port and ensure the BS2-SX is selected. After doing this, you can

use the CTRL+I shortcut to identify the Stamp version you are connected to. If there is a communication problem between your PC and the Stamp, it will show up here.

Each of the three programs are listed together, but I think it is important to point out that each program was actually written in a separate editor window of Parallax's Win Interface software. The software makes it easy to switch between editing windows. I believe that this method of writing software for the BS2-SX will actually help with program organization. This is especially true of Stamp users who have not written code extensively for other microcontrollers (which can get quite large).

In Closing

The two major shortcomings of the BS2 were limited program storage space, and slow instruction execution speed. Both of these shortcomings have been addressed with the release of the BS2-SX. The BS2-SX creeps closer to the performance available in application specific microcontrollers. While the performance of the BS2-SX still does not match that of a custom-designed microcontroller, its ease of use is unmatched. Also, on the plus side, the addition of the faster BS2-SX, with a much larger memory map, has not resulted in a steep learning curve.

In fact, the greatest trouble I had resulted in the minimal documentation available for the Windows interface software. And I'm sure Parallax is working on this aspect of their "design environment."

All in all, the BS2-SX appears to be a fine addition to the BASIC Stamp product line. Next month, we'll go into more depth and learn how to use the multiple program capability of the BS2-SX to develop a versatile communication interface. Hope you can make it!

