Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	-
Core Size	-
Speed	-
Connectivity	-
Peripherals	-
Number of I/O	-
Program Memory Size	-
Program Memory Type	-
EEPROM Size	-
RAM Size	-
Voltage - Supply (Vcc/Vdd)	-
Data Converters	-
Oscillator Type	-
Operating Temperature	-
Mounting Type	-
Package / Case	-
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/parallax/pbasic2e-ss

Stamp 2 OEM is available in either an assembled form or a kit form. These three packages are functionally equivalent.

In addition to the dual-inline and OEM packages, there are prototyping boards available that feature a surface mounted BS2. Please check www.parallax.com → Products → Development Boards for product descriptions.

Table 1.2: BASIC Stamp 2 Pin Descriptions.

Pin	Name	Description
1	SOUT	Serial Out: connects to PC serial port RX pin (DB9 pin 2 / DB25 pin 3) for programming.
2	SIN	Serial In: connects to PC serial port TX pin (DB9 pin 3 / DB25 pin 2) for programming.
3	ATN	Attention: connects to PC serial port DTR pin (DB9 pin 4 / DB25 pin 20) for programming.
4	VSS	System ground: (same as pin 23) connects to PC serial port GND pin (DB9 pin 5 / DB25 pin 7) for programming.
5-20	P0-P15	General-purpose I/O pins: each can sink 25 mA and source 20 mA. However, the total of all pins should not exceed 50 mA (sink) and 40 mA (source) if using the internal 5-volt regulator. The total per 8-pin groups (P0 – P7 or P8 – 15) should not exceed 50 mA (sink) and 40 mA (source) if using an external 5-volt regulator.
21	VDD	5-volt DC input/output: if an unregulated voltage is applied to the VIN pin, then this pin will output 5 volts. If no voltage is applied to the VIN pin, then a regulated voltage between 4.5V and 5.5V should be applied to this pin.
22	RES	Reset input/output: goes low when power supply is less than approximately 4.2 volts, causing the BASIC Stamp to reset. Can be driven low to force a reset. This pin is internally pulled high and may be left disconnected if not needed. Do not drive high.
23	VSS	System ground: (same as pin 4) connects to power supply's ground (GND) terminal.
24	VIN	Unregulated power in: accepts 5.5 - 15 VDC (6-40 VDC on BS2-IC Rev. e, f, and g), which is then internally regulated to 5 volts. Must be left unconnected if 5 volts is applied to the VDD (+5V) pin.

See the "BASIC Stamp Programming Connections" section on page 27 for more information on the required programming connections between the PC and the BASIC Stamp.

editor will have its own tab at the top of the page labeled with the name of the file, as seen in Figure 3.2. The full file path of the currently displayed source code appears in the title bar. Source code that has never been saved to disk will default to "Untitled#"; where # is an automatically generated number. A user can switch between source code files by simply pointing and clicking on a file's tab or by pressing Ctrl+Tab or Ctrl+Shift+Tab while the main edit pane is active.

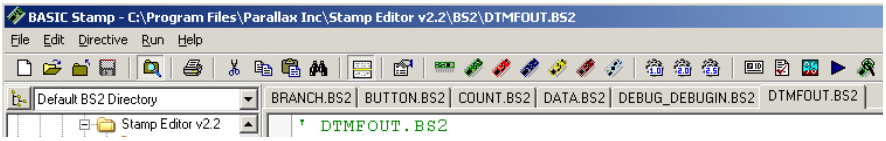


Figure 3.2: Example Editor Tabs. Shown with 6 separate files open; Title Bar shows current code's file path.

The status of the active source code is indicated in the status bar below the main edit pane and integrated explorer panel. The status bar contains information such as cursor position, file save status, download status and syntax error/download messages. The example in Figure 3.3 indicates that the source code tokenized successfully.

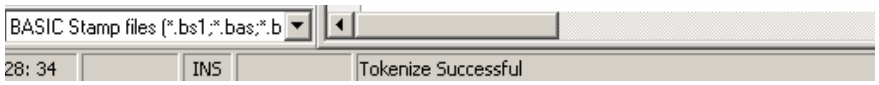


Figure 3.3: Status Bar beneath the Main Edit Pane.

SPLIT WINDOW VIEW.

Each editor pane can be individually split into two views of the same source code. This can be done via the Split button on the toolbar, pressing Ctrl-L, or clicking and dragging the top or bottom border of the editor pane with the mouse.

Once split, the top and bottom edit controls allow viewing of different areas of the same source code; this can be handy when needing to keep variable declarations or a particular routine in view while modifying a related section of code elsewhere. Note that the Split button and Ctrl+L shortcut act like a toggle function, splitting or un-splitting the edit pane.

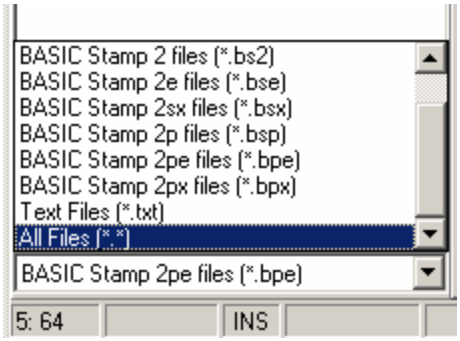


Figure 3.8: The Filter List found at the bottom of the Integrated Explorer Panel.

The BASIC Stamp Editor automatically associates BASIC Stamp source code file types (.bs1, .bs2, .bse, .bsx, .bsp, .bpe, and .bpx) with itself. This feature can be configured through automatic prompts or through the Preferences → Files & Directories tab. Also, when using any Explorer-shell for file browsing, right-clicking on a BASIC Stamp source code file provides you with an Open With Stamp Editor option.

The integrated explorer panel can be resized via the vertical splitter bar that separates it and the edit pane. The Directory list and File list can be resized via the horizontal splitter bar that separates them. The integrated explorer can also be hidden or shown via the Explorer toolbar button, by pressing Ctrl+E, or by resizing it to zero width using the vertical splitter bar.

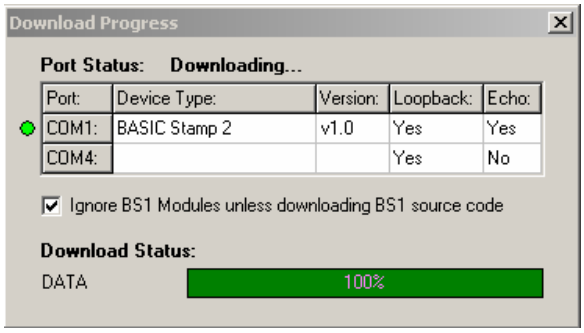
Table 3.3 lists keyboard shortcuts for several file functions.

File Functions	
Shortcut Key	**Function**
Ctrl+E	Show/hide explorer panel
Ctrl+L	Show/hide split view in edit pane
Ctrl+O	Open a source code file into edit pane
Ctrl+Shift+O	Open a source code file from a recent directory into edit pane
Ctrl+S	Save current source code file to its current location on disk
Ctrl+Shift+S	Save current source code file to a recent directory on disk
Ctrl+P	Print current source code
Ctrl+Tab	Switch to next open file page
Ctrl+Shift+Tab	Switch to previous open file page

Table 3.3: Keyboard Shortcuts for File Functions.

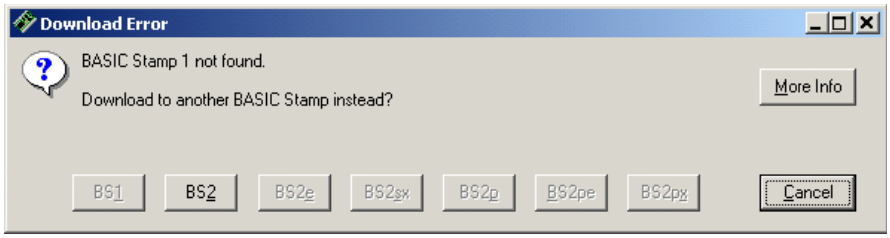
to the BASIC Stamp module (assuming the code is correct and the BASIC Stamp is properly connected). The Download Progress window looks similar to the Identify window with the exception of the additional Download Status progress bar, and the indicator LED by the port transmitting the data.

Figure 3.11: The Download Progress Window.



If any errors occur, such as communication failure or inability to detect a BASIC Stamp module, you will be prompted appropriately. One possible error occurs when the BASIC Stamp your PBASIC program is targeting does not appear to be connected to the PC (see Figure 3.12). This may be caused, for example, by opening up a BASIC Stamp 1 program (usually has a .bas or .bs1 extension) and trying to download it to a BASIC Stamp 2 module, instead.

Figure 3.12: A Download Error message.



When this happens, you'll be prompted to correct the situation, quickly done by clicking on the BS2 button (if you really intended to download to the BS2 in the first place). Keep in mind that programs written for one BASIC Stamp model may not function properly on a different BASIC Stamp model. Click on the More Info button for more detail. NOTE: If you select the BS2 button, as in this example, the editor will modify the

customers using BASIC Stamp-based products, you can release firmware updates to them in this manner.

Object code can be saved as a separate .obj file (downloadable with the StampLoader.exe program) or as a single executable (integrated with the StampLoader.exe inside of it). The single executable method provides a simpler way to pass your firmware update on to your customers.

Any syntactically correct PBASIC source code can be used with the Generate Object Code feature; this includes BS1 and BS2 code as well as BS2e, BS2sx, BS2p, BS2pe, and BS2px code that is either a single file or a multi-file project. Note: The original DOS-based software for the BS1 included a directive called BSAVE; when used it would cause the software to generate an object file. In the BASIC Stamp Windows Editor, the Generate Object Code feature replaces and enhances the BSAVE feature; the reserved word BSAVE is still accepted in BS1 source code, but is simply ignored. Old BS1 object code saved via the BSAVE option is not compatible with the StampLoader.exe program so you must regenerate the object file using the BASIC Stamp Windows Editor.

If you don't have the StampLoader.exe program, it can be automatically generated for you by selecting the second output file option, "Object Code and Stamp Loader", in the Generate Object Code window. Additionally, firmware, product, company and related info can be embedded in the object code or single executable file for your customers to view before downloading.

```
  IOTERM port                       ' Switch to main or aux I/Os
                                    ' -- depending on port
  TOGGLE 3                          ' Toggle state of I/O pin 3
                                    ' -- on main and aux, alternately
  port = ~port                      ' Invert port
  PAUSE 1000                        ' 1 second delay
LOOP
END
```

CONFIGPIN – BASIC Stamp Command Reference

reps becomes 0 (bytes will rollover after 255 just like words will rollover after 65535). The result, 0, is compared against the range (0 – 255) and it is found to be within the range, so the FOR…NEXT loop continues.

It's important to realize that on all the BS2 models, the test is against the entire range, not just the *EndValue*. The code below is a slight modification of the previous example (the *StartValue* is 10 instead of 0) and will not loop endlessly.

NOTE: On the BS1, the loop will continue until *Counter* has gone past *EndValue*. The rollover error will still occur if the BS1 cannot determine if *Counter* went past *EndValue*.

All 2

```
reps    VAR     Byte                    ' counter for the loop

FOR reps = 10 TO 300                    ' each loop add 1
  DEBUG DEC ? reps                      ' show reps in Debug window
NEXT
```

reps still rolls over to 0, as before, however, this time it is outside the range of 10 to 255. The loop stops, leaving *reps* at 0. Note that this code is still in error since *reps* will never reach 300 until it is declared as a Word.

Demo Program (FOR-NEXT.bs1)

1

```
' FOR-NEXT.bs1
' This example uses a FOR...NEXT loop to churn out a series of sequential
' squares (numbers 1, 2, 3, 4... raised to the second power) by using a
' variable to set the FOR...NEXT StepValue, and incrementing StepValue
' within the loop.  Sir Isaac Newton is generally credited with the
' discovery of this technique.

' {$STAMP BS1}
' {$PBASIC 1.0}

SYMBOL  square          = B2            ' FOR/NEXT counter
SYMBOL  stepSize        = B3            ' step size increases by 2 each loop


Setup:
  stepSize = 1
  square = 1

Main:
  FOR square = 1 TO 250 STEP stepSize   ' show squares up to 250
    DEBUG square                        ' display on screen
    stepSize = stepSize + 2             ' add 2 to stepSize
  NEXT                                  ' loop until square > 250
  END
```

IF…THEN – BASIC Stamp Command Reference

Quick Facts

	BS1	All BS2 Models
Comparison Operators	=, <>, >, <, >=, <=	=, <>, >, <, >=, <=
Conditional Logic Operators	AND, OR	NOT, AND, OR, XOR
Format of *Condition*	*Variable Comparison Value*; where *Value* is a variable or constant	*Value1 Comparison Value2*; where *Value1* and *Value2* can by any of variable, constant or expression
Parentheses	Not Allowed	Allowed
Max nested IF…THENs	n/a	16
Max ELSEIFs per IF	n/a	16
Max ELSEs per IF	n/a	1
Related Command	None	SELECT…CASE

Table 5.38: IF...THEN Quick Facts.

Explanation

IF...THEN is PBASIC's decision maker that allows one block of code or another to run based on the value (True or False) of a condition. The condition that IF...THEN tests is written as a mixture of comparison and logic operators. The available comparison operators are:

Comparison Operator Symbol	Definition
=	Equal
<>	Not Equal
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To

Table 5.39: IF...THEN Comparison Operators.

Comparisons are always written in the form: Value1 Comparison Value2. The values to be compared can be any combination of variables (any size), constants, or expressions.

NOTE: On the BS1, expressions are not allowed as arguments. Also, the Value1 (to the left of comparison) must be a variable.

The following example is an IF…THEN command with a simple condition:

1 All 2

A SIMPLE FORM OF IF…THEN

```
IF value < 4000 THEN Main
```

This code will compare the value of *value* to the number 4000. If *value* is less than 4000, the condition is true and the program will jump (implied

```
' {$STAMP BS2}
' {$PBASIC 2.0}


sample          VAR     Word            ' Random number to be tested
samps           VAR     Nib             ' Number of samples taken
temp            VAR     Nib             ' Temporary workspace


Setup:
  sample = 11500

Mult3:
  RANDOM sample                         ' Put a random number into sample
  temp = sample // 3
  IF temp <> 0 THEN Mult3               ' Not multiple of 3? -- try again
    DEBUG DEC5 sample, " divides by 3", CR
    samps = samps + 1                   ' Count multiples of 3
    IF samps = 10 THEN Done             ' Quit with 10 samples
  GOTO Mult3                            ' keep checking

Done:
  DEBUG CR, "All done."
  END
```

All 2

Demo Program (IF-THEN-ELSE.bs2)

NOTE: This example program can be used with all BS2 models by changing the $STAMP directive accordingly.

```
' IF-THEN-ELSE.bs2
' The program below generates a series of 16-bit random numbers and tests
' each to determine whether they're evenly divisible by 3. If a number is
' evenly divisible by 3, then it is printed, otherwise, the program
' generates another random number. The program counts how many numbers it
' prints, and quits when this number reaches 10.

' {$STAMP BS2}
' {$PBASIC 2.5}                         ' version 2.5 required

sample          VAR     Word            ' Random number to be tested
hits            VAR     Nib             ' Number of hits
misses          VAR     Word            ' Number of misses


Setup:
  sample = 11500

Main:
  DO
    RANDOM sample                       ' Put a random number into sample
    IF ((sample // 3) = 0) THEN         ' divisible by 3?
      DEBUG DEC5 sample,                ' - yes, print value and message
            " is divisible by 3", CR
```

on the BS1) will appear on the pin. The demo program shows how this works.

Demo Program (INPUT.bs1)


```
' INPUT.bs1
' This program demonstrates how the input/output direction of a pin is
' determined by the corresponding bit of DIRS. It also shows that the
' state of the pin itself (as reflected by the corresponding bit of PINS)
' is determined by the outside world when the pin is an input, and by the
' corresponding bit of OUTS when it's an output. To set up the demo,
' connect a 10k resistor from +5V to P7 on the BASIC Stamp. The resistor
' to +5V puts a high (1) on the pin when it's an input. The BASIC Stamp
' can override this state by writing a low (0) to bit 7 of OUTS and
' changing the pin to output.

' {$STAMP BS1}
' {$PBASIC 1.0}

Main:
  INPUT 7                               ' Make P7 an input
  DEBUG "State of P7: ", #PIN7, CR

  PIN7 = 0                              ' Write 0 to output latch
  DEBUG "After 0 written to OUT7: "
  DEBUG #PIN7, CR

  OUTPUT 7                              ' Make P7 an output
  DEBUG "After P7 changed to output: "
  DEBUG #PIN7
```

Demo Program (INPUT.bs2)

NOTE: This example program can be used with all BS2 models by changing the $STAMP directive accordingly.

```
' INPUT.bs2
' This program demonstrates how the input/output direction of a pin is
' determined by the corresponding bit of DIRS. It also shows that the
' state of the pin itself (as reflected by the corresponding bit of INS)
' is determined by the outside world when the pin is an input, and by the
' corresponding bit of OUTS when it's an output. To set up the demo,
' connect a 10k resistor from +5V to P7 on the BASIC Stamp. The resistor
' to +5V puts a high (1) on the pin when it's an input. The BASIC Stamp
' can override this state by writing a low (0) to bit 7 of OUTS and
' changing the pin to output.

' {$STAMP BS2}
' {$PBASIC 2.5}

Main:
  INPUT 7                               ' Make P7 an input
  DEBUG "State of P7: ",
```

MAINIO

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

MAINIO

Function

Switch from control of auxiliary I/O pins to main I/O pins (on the BS2p40 only).

Quick Facts

Table 5.58: MAINIO Quick Facts.

	BS2p, BS2pe, and BS2px
I/O pin IDs	0 – 15 (just like auxiliary I/O, but after MAINIO command, all references affect physical pins 5 – 20).
Special Notes	The 24-pin BS2p, BS2pe, and BS2px accept this command, however, only the BS2p40 gives access to the auxiliary I/O pins.
Related Commands	AUXIO and IOTERM

Explanation

The BS2p, BS2pe and BS2px are available as 24-pin modules that are pin compatible with the BS2, BS2e and BS2sx. Also available is a 40-pin module called the BS2p40, with an additional 16 I/O pins (for a total of 32). The BS2p40's extra, or auxiliary, I/O pins can be accessed in the same manner as the main I/O pins (by using the IDs 0 to 15) but only after issuing AUXIO or IOTERM commands. The MAINIO command causes the BASIC Stamp to affect the main I/O pins (the default) instead of the auxiliary I/O pins in all further code until the AUXIO or IOTERM command is reached, or the BASIC Stamp is reset or power-cycled.

A SIMPLE MAINIO EXAMPLE.

The following example illustrates this:

```
AUXIO                       ' switch to auxiliary pins
HIGH 0                      ' make X0 high
MAINIO                      ' switch to main pins
LOW 0                       ' make P0 low
```

The first line of the above example will tell the BASIC Stamp to affect the auxiliary I/O pins in the commands following it. Line 2, sets I/O pin 0 of the auxiliary I/O pins (physical pin 21) high. Afterward, the MAINIO command tells the BASIC Stamp that all commands following it should affect the main I/O pins. The last command, LOW, will set I/O pin 0 of the main I/O pins (physical pin 5) low.

OUTPUT

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

1 All 2

OUTPUT *Pin*

Function

Make the specified pin an output.

- ***Pin*** is a variable/constant/expression (0 – 15) that specifies which I/O pin to set to output mode.

1 NOTE: Expressions are not allowed as arguments on the BS1. The range of the *Pin* argument on the BS1 is 0 – 7.

Quick Facts

Table 5.62: OUTPUT Quick Facts.

	BS1 and all BS2 Models
Related Commands	INPUT and REVERSE

Explanation

There are several ways to make a pin an output. Commands that rely on output pins, like PULSOUT and SEROUT, automatically change the specified pin to output. Writing 1s to particular bits of the variable DIRS makes the corresponding pins outputs. And then there's the OUTPUT command.

When a pin is an output, your program can change its state by writing to the corresponding bit in the OUTS variable (PINS on the BS1). For example:

All 2

```
OUTPUT 4
OUT4 = 1
```

EFFECTS OF SETTING AN INPUT PIN TO AN OUTPUT.

When your program changes a pin from input to output, whatever state happens to be in the corresponding bit of OUTS (PINS on the BS1) sets the initial state of the pin. To simultaneously make a pin an output and set its state use the HIGH and LOW commands.

1

Demo Program (INPUT_OUTPUT.bs1)

```
' INPUT_OUTPUT.bs1
' This program demonstrates how the input/output direction of a pin is
' determined by the corresponding bit of DIRS. It also shows that the
' state of the pin itself (as reflected by the corresponding bit of PINS)
' is determined by the outside world when the pin is an input, and by the
' corresponding bit of PINS when it's an output. To set up the demo,
' connect a 10k resistor from +5V to P7 on the BASIC Stamp. The resistor
```

OWIN

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

OWIN ***Pin, Mode,*** **[** ***InputData*** **]**

Function

Receive data from a device using the 1-Wire protocol.

- ***Pin*** is a variable/constant/expression (0 – 15) that specifies which I/O pin to use. 1-Wire devices require only one I/O pin (called DQ) to communicate. This I/O pin will be toggled between output and input mode during the OWIN command and will be set to input mode by the end of the OWIN command.
- ***Mode*** is a variable/constant/expression (0 – 15) indicating the mode of data transfer. The *Mode* argument controls placement of reset pulses (and detection of presence pulses) as well as byte vs. bit input and normal vs. high speed. See explanation below.
- ***InputData*** is a list of variables and modifiers that tells OWIN what to do with incoming data. OWIN can store data in a variable or array, interpret numeric text (decimal, binary, or hex) and store the corresponding value in a variable, wait for a fixed or variable sequence of bytes, or ignore a specified number of bytes. These actions can be combined in any order in the *InputData* list.

Quick Facts

Table 5.63: OWIN Quick Facts.

	BS2p, BS2pe, and BS2px
Receive Rate	Approximately 20 kbits/sec (low speed, not including reset pulse)
Special Notes	The DQ pin (specified by *Pin*) must have a 4.7 KΩ pull-up resistor. The BS2pe is not capable of high-speed transfers.
Related Commands	OWOUT

Explanation

The 1-Wire protocol is a form of asynchronous serial communication developed by Dallas Semiconductor. It only requires one I/O pin and that pin can be shared between multiple 1-Wire devices. The OWIN command allows the BASIC Stamp to receive data from a 1-wire device.

A SIMPLE OWIN EXAMPLE.

The following is an example of the OWIN command:

```
result  VAR     Byte

OWIN 0, 1, [result]
```

```
bitOne  VAR     Bit
bitTwo  VAR     Bit

bitOne = 0
bitTwo = 1
OWOUT 0, 5, [bitOne, bitTwo]
```

In the code above, we chose the value "5" for *Mode*. This sets Bit transfer and Front-End Reset modes. Also, we could have chosen to make the *bitOne* and *bitTwo* variables each a byte in size, but the BASIC Stamp would still only use the their lowest bit (BIT0) as the value to transmit in the OWOUT command (due to the *Mode* we chose).

SENDING AND FORMATTING DATA.

The OWOUT command's *OutputData* argument is similar to the DEBUG and SEROUT command's *OutputData* argument. This means data can be sent as literal text, ASCII character values, repetitive values, decimal, hexadecimal and binary translations and string data as in the examples below. (Assume a 1-wire device is used and that it transmits the string, "Value: 3A:101" every time it receives a Front-End Reset pulse).

```
value   VAR     Byte
value = 65

OWOUT 0, 1, [value]                     ' send "A"
OWOUT 0, 1, [REP value\5]               ' send "AAAAA"
OWOUT 0, 1, [DEC value]                 ' send "6" and "5"
OWOUT 0, 1, [HEX value]                 ' send "4" and "1"
OWOUT 0, 1, [BIN value]                 ' send "1000001"
```

Table 5.70 and Table 5.71 list all the special formatters and conversion formatters available to the OWOUT command. See the DEBUG and SEROUT commands for additional information and examples of their use.

Table 5.70: OWOUT Special Formatters.

Special Formatter	Action
?	Displays "symbol = x' + carriage return; where x is a number. Default format is decimal, but may be combined with conversion formatters (ex: BIN ? x to display "x = binary_number").
ASC ?	Displays "symbol = 'x'" + carriage return; where x is an ASCII character.
STR *ByteArray* {\L}	Send character string from an array. The optional \L argument can be used to limit the output to L characters, otherwise, characters will be sent up to the first byte equal to 0 or the end of RAM space is reached.
REP *Byte* \L	Send a string consisting of *Byte* repeated L times (ex: REP "X"\10 sends "XXXXXXXXXX").

user program; giving the appearance that it is polling "in the background". This feature should not be confused with the concept of interrupts, as the BASIC Stamp *does not support true interrupts*.

A SIMPLE POLLIN EXAMPLE.

The following is an example of the POLLIN command:

```
POLLIN 0, 0
POLLMODE 2
```

The POLLIN command in the above code will cause the BASIC Stamp to set I/O pin 0 to an input mode and get ready to poll it for a low (0) state. The BASIC Stamp will not actually start polling until it is set to the appropriate mode, however. The second line, POLLMODE, initiates the polling process (see the POLLMODE description for more information). From then on, as the BASIC Stamp executes the rest of the program, it will check for a low level (logic 0) on I/O pin 0, in-between instructions.

SETTING ONE OF THE POSSIBLE ACTIONS: POLLED-OUTPUTS

In the code above, no obvious action will be noticed since we didn't tell the BASIC Stamp what to do when it detects a change on the I/O pin. One possible action the BASIC Stamp can be instructed to take is to change the state of an output, called a polled-output. Take a look at the next example:

```
POLLIN 0, 0
POLLOUT 1, 1
POLLMODE 2

Main:
  DEBUG "Looping...", CR
  GOTO Main
```

In this example, in addition to an endless loop, we've added another polling command called POLLOUT (see the POLLOUT description for more information). Our POLLOUT command tells the BASIC Stamp to set I/O pin 1 to an output mode and set it high (1) when it detects the desired poll state. The poll state is the low (0) level on I/O pin 0 that POLLIN told it to look for. If the polled-input pin is high, it will set polled-output pin 0 to low (0), instead.

Once the program reaches the endless loop, at *Main*, it will continuously print "Looping…" on the PC screen. In between reading the DEBUG command and the GOTO command (and vice versa) it will check polled-input pin 0 and set polled-output pin 1 accordingly. In this case, when I/O pin 0 is set low, the BASIC Stamp will set I/O pin 1 high. When I/O

SEROUT

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

1 **SEROUT *Tpin*, *Baudmode*, ({#} *OutputData*)**

All 2 **SEROUT *Tpin* { *Fpin* }, *Baudmode*, { *Pace*, } { *Timeout*, *Tlabel*, } [*OutputData*]**

Function

Transmit asynchronous serial data (e.g., RS-232 data).

1 NOTE: Expressions are not allowed as arguments on the BS1. The range of the Rp*in* argument on the BS1 is 0 – 7.

- ***Tpin*** is a variable/constant/expression (0 – 16) that specifies the I/O pin through which the serial data will be transmitted. This pin will be set to output mode. On all BS2 models, if Tpin is set to 16, the BASIC Stamp uses the dedicated serial-output pin (SOUT, physical pin 1), which is normally used by the Stamp Editor during the download process.
- ***Fpin*** is an optional variable/constant/expression (0 – 15) that specifies the I/O pin to monitor for flow control status. This pin will be set to input mode. NOTE: Fpin must be specified to use the optional *Timeout* and *Tlabel* arguments in the SEROUT command.
- ***Baudmode*** is variable/constant/expression (0 – 7 on the BS1, 0 – 65535 on all BS2 models) that specifies serial timing and configuration.
- ***Pace*** is an optional variable/constant/expression (0 – 65535) that determines the length of the pause between transmitted bytes. NOTE: Pace cannot be used simultaneously with *Timeout* and *Fpin*.
- ***Timeout*** is an optional variable/constant/expression (0 – 65535) that tells SEROUT how long to wait for *Fpin* permission to send. If permission does not arrive in time, the program will jump to the address specified by *Tlabel*. NOTE: Fpin must be specified to use the optional *Timeout* and *Tlabel* arguments in the SEROUT command.
- ***Tlabel*** is an optional label that must be provided along with *Timeout*. Tlabel indicates where the program should go in the event that permission to send data is not granted within the period specified by *Timeout*.

1 NOTE: The BS1's *OutputData* argument can only be a list of variables and the optional decimal modifier (#).

- ***OutputData*** is list of variables, constants, expressions and formatters that tells SEROUT how to format outgoing data. SEROUT can transmit individual or repeating bytes, convert values into decimal,

```
' msb first so that the msb appears on pin QH and the lsb on QA. Changing
' MSBFIRST to LSBFIRST causes the data to appear backwards on the outputs.

Main:
  DO
    SHIFTOUT Dpin, Clk, MSBFIRST, [counter]     ' send the bits
    PULSOUT Latch, 1                            ' transfer to outputs
    PAUSE 100                                   ' Wait 0.1 seconds
    counter = counter + 1                       ' increment counter
  LOOP
  END
```

Demo Program (STORE1.bsp)

NOTE: This example program can be used with the BS2p, BS2pe, and BS2px by changing the $STAMP directive accordingly.

```
' STORE1.bsp

' {$STAMP BS2p}
' {$PBASIC 2.5}

idx             VAR     Word            ' index
value           VAR     Byte

LocalData       DATA    @0, 6, 7, 8, 9, 10


Main:
  GOSUB Show_Slot_Info                  ' show slot info/data
  PAUSE 2000
  STORE 0                               ' point READ/WRITE to Slot 0
  GOSUB Show_Slot_Info
  PAUSE 2000
  RUN 2                                 ' run program in Slot 2
  END

Show_Slot_Info:
  GET 127, value
  DEBUG CR, "Pgm Slot: ", DEC value.NIB0,
        CR, "R/W Slot: ", DEC value.NIB1,
        CR, CR

  FOR idx = 0 TO 4
    READ idx, value
    DEBUG "Location: ", DEC idx, TAB,
          "Value: ", DEC3 value, CR
  NEXT
  RETURN
```

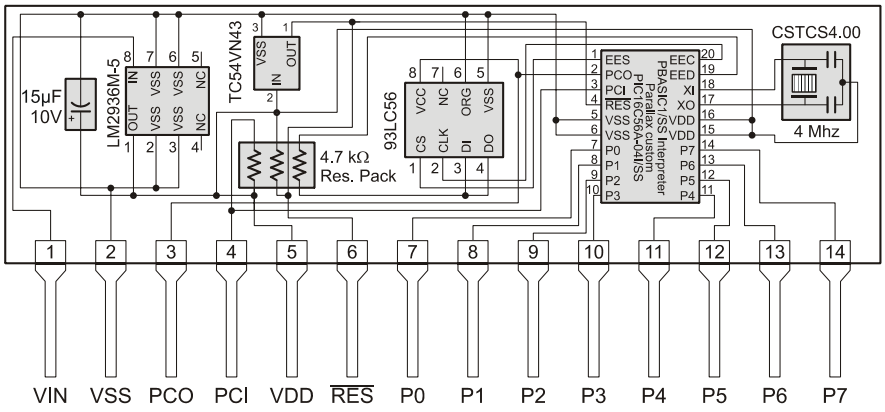
Demo Program (STORE2.bsp)

NOTE: This example program can be used with the BS2p, BS2pe, and BS2px by changing the $STAMP directive accordingly.

```
' STORE2.bsp

' {$STAMP BS2p}
' {$PBASIC 2.5}

idx             VAR     Word            ' index
value           VAR     Byte

LocalData       DATA    @0, 11, 12, 13, 14, 15


Main:
  GOSUB Show_Slot_Info                  ' show slot info/data
  PAUSE 2000
  STORE 0                               ' point READ/WRITE to Slot 0
```

Appendix D: BASIC Stamp Schematics

BASIC Stamp 1 Schematic (Rev B)



Notes:

The 15µF, 10V capacitor may be a 10-22µF, 6.3-16V tantalum capacitor.

The 93LC56 EEPROM may be a 93LC56A, 93LC66 or 93LC66A.

The 4 MHz resonator is not polarity sensitive and the middle pin can be connected to either VDD or VSS.

The PBASIC/SS Interpreter chip may be a commercial PIC16C56A-04/SS or an industrial PIC16C56A-04I/SS