PBASIC Language Basics

Dhinesh Sasidaran

PBASIC INTRODUCTION

PBASIC stands for Parallax BASIC which is a variant of BASIC. This special language has familiar BASIC instructions such as FOR..NEXT, IF..THEN and GOTO along with some useful extra instructions that are specially for input and output (I/O). Programs can be written using the STAMP programming software and downloaded on a serial port to the BASIC Stamp.

YOUR FIRST STAMP PROGRAM

You can use the DEBUG command to print values to the debugging terminal managed by the STAMP programming software. The command allows you to view variables or track the flow of execution through your program. Run the parallax software and write the following program in the STAMP editor window:

```
'{$STAMP BS2}
DEBUG "My first STAMP Program!"
END
```

Run your program. A window should pop up and you should see the debugging message on the screen.

What appears is what was sent from the BASIC Stamp, through the programming cable to the PC. The first line of the program is a special comment (or directive) which indicates which version of the Stamp is in use.

Writing your program

Write your program using the BASIC Stamp Windows Editor as shown in Figure 1.

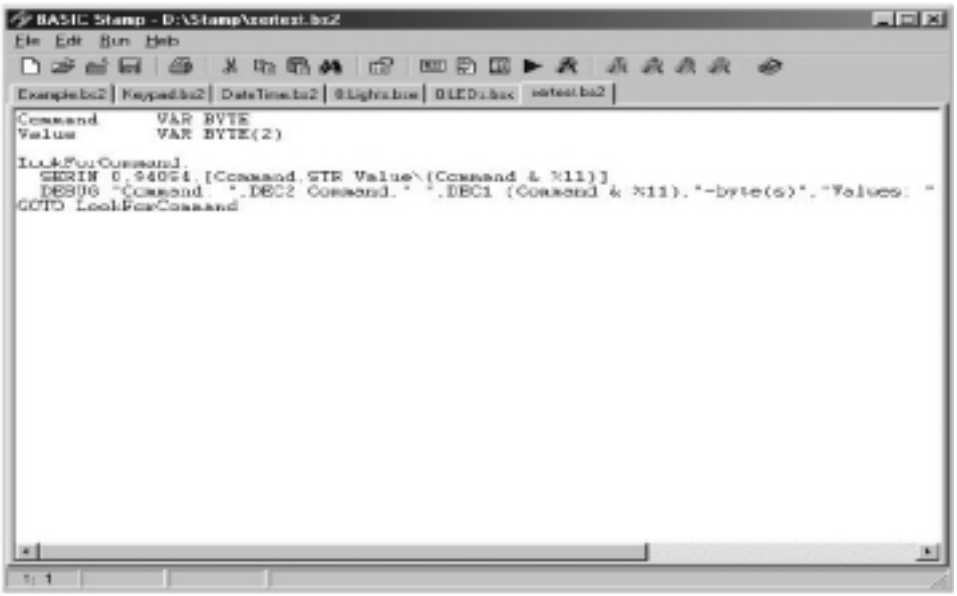


FIGURE 1. BASIC Stamp Windows Editor

After entering your program using the Stamp editor, select Run --> Run (or by pressing Ctrl-R). This tokenizes your code and downloads it into the Stamp.

MEMORY MAP AND I/O

The BASIC Stamp has 32 bytes of variable RAM space arranged as shown in Figure 2.

INS/OUTS	
INH/OUTH	INL/OUTL

DIRS		W6	
DIRH	DIRL	B13	B12
W0		W7	
B1	B0	B15	B14
W1		W8	
B3	B2	B17	B16
W2		W9	
B5	B4	B19	B18
W3		W10	
B7	B6	B21	B20
W4		W11	
B9	B8	B23	B22
W5		W12	
B11	B10	B25	B24

FIGURE 2. Stamp memory map

IDENTIFIERS

Identifiers are names that a programmer makes up when writing a program. In PBASIC, an identifier must begin with a letter and can be composed of a combination of letters, digits and the underscore character (_) but cannot be the same as PBASIC keywords (reserved words) or labels. PBASIC is not case sensitive and therefore both upper case and lower case letters can be used in an identifier.

KEYWORDS (Reserved Words)

Appendix B of the STAMP manual provides a list of PBASIC keywords. You can download the manual from :

(www.parallaxinc.com/downloads/download_documentation.htm)

GENERAL PROGRAM FORMAT

1. Labels

   You can place labels in your program to identify a particular spot that your program has to jump to. Labels are identified by colons (:) proceeding the label name.

2. Variables

   The VAR keyword is used to specify a variable along with the size of the variable. The VAR keyword causes the Stamp II programming software to reserve a register based on the size specified. Before you use a variable in your program, you will need to declare it. PBASIC however does have predefined variables that you can use without first declaring them in your program. These are general-purpose variables with defined names.

   W0 through W12

   B0 through B25 (where B0 is the lower byte of W0 and B1 is the high byte of W0)

   However, it is recommended that you avoid using fixed variables in most situations and let the PBASIC arrange the variables into the registers as it sees fit to make optimal use of memory.

   For example:

   counter VAR byte

   This command selects one of the byte-sized registers to be assigned to the variable named counter. The sizes of the variable can vary from WORD, BYTE, NIB (nibble) or BIT.

```
Dog      VAR   BIT    'Value can be 0 or 1
Cat      VAR   NIB    'Value can be 0 to 15
Dolphin  VAR   BYTE   'Value can be 0 to 255
Whale    VAR   WORD   'Value can be 0 to 65535
```

Variables should be assigned by determining the largest value that will ever be stored in it. The smallest size with respect to that should be chosen for the variable.

Aliases for variables can also be created using the VAR command. For example:

```
counter     VAR   BYTE
countodd    VAR   counter
```

In the lines above, *countodd* is an alias to the variable counter. Anything stored in counter shows up in *countodd* and vice versa. Both names will refer to the same physical address.

You can also use the alias as a window into a portion of another variable. This is done using “modifiers”. For example:

```
Whale   VAR   WORD                ‘A 16-bit variable
Dolphin VAR   Whale.HIGHBYTE      ‘Highest 8 bits of Whale
Shark   VAR   Whale.LOWBYTE       ‘Lowest 8 bits of Whale
```

The following table lists the modifiers and the definition of their use with variables.

Modifiers	**Definition**
LOWBYTE	Low byte of a word
HIGHBYTE	high byte of a word
BYTE0	byte 0 (low byte) of a word
LOWNIB	low nibble of word or byte
HIGHNIB	high nibble of word or byte
NIB0	nib 0 of a word or byte
NIB1	nib 1 of a word or byte
NIB2	nib 2 of a word
NIB3	nib 3 of a word
LOWBIT	low bit of a word, byte, or nibble
HIGHBIT	high bit of a word, byte, or nibble
BIT0	bit 0 of a word, byte, or nibble
BIT1	bit 1 of a word, byte, or nibble
BIT2	bit 2 of a word, byte, or nibble
BIT3	bit 3 of a word, byte, or nibble
BIT4 ... BIT7	bits 4 through 7 of a word or byte
BIT8 ... BIT15	bits 8 through 15 of a word

3. Array variables

   You can also declare arrays using the VAR command. An array has multiple items of the same type. The following command is used to declare an array with a list of 3 bytes.

```
myarray    VAR    byte(3)
```

This command creates a 3 byte-sized element array. With the abovementioned declaration, the following assignments could be made:

```
myarray(0)  =  1
myarray(1)  =  10
myarray(2)  =  100
```

Using an array without the specified index will cause the software to respond with/to the 1st element in the array.

4. Input/Output direction

   The direction of input and output pins on the Stamp can be controlled by setting the appropriate bits in the DIRS register: a '1' indicates an output pin and a '0' indicates an input pin. For example:

   ```
   dirs = $F000
   ```

   this is equivalent to setting I/O pins 15:12 as output pins while pins 11:0 are designated as input pins.

   Separate registers are provided for input (INS) and output (OUTS) and can be assigned separately (IN0 or OUT1).

   The OUTS register remembers what is written into it even if some of the bits are not outputs to begin with at that time. When the DIRS register changes the directions of those bits to outputs, the output pin will use the value that exists in the OUTS register.

5. Math expressions

   Math operations are performed from left to right. There are no operator precedence rules except when it comes to UNARY and BINARY operators. Unary operators are given precedence in math calculations. For example:

   ```
   10 - SQR 16
   ```

   The BASIC Stamp first takes the square root of 16 and then subtracts it from 10.

   In the case of binary operators, the expression:

   ```
   5 + 3 * 2
   ```

   will yeild the result 16 and not 11. Therefore, for proper calculation, the parentheses character can be placed in the expression.

   ```
   5 + (3 * 2) = 11
   ```

   Note: Only 8 levels of parentheses are allowed in your math expressions.

   Math expressions can also be used when dealing with input and output pins. For example:

   ```
   B1  =  10
   INPUT B1+1          'Make pin 11 an input pin
   ```

6. Constants

   Constants can be specified using the CON keyword. For example:

   ```
   delay      CON    1000
   ```

   Constants can also be defined in terms of another constant but must be kept fairly simple. For example

```
Whale    CON  20
Dolphin  CON  Whale*2-1
```

NUMBERS

PBASIC allows you to use several numbering systems. By default, it assumes that numbers are in decimal, but you can identify binary and hexadecimal numbers with a prefix.

```
99      decimal
%1010   binary
$FE     hex
```

BASIC Stamp performs integer match (whole numbers only) and drops any fractional portion from the results of the computation. The size of the variables can be a bit (0-1), nibble (0-15), byte (0-255) or word (0-65535) respectively.

For negative numbers, two's complement math can be used for representation.

Negative Numbers

When the Stamp performs calculations, if the value is greater than the size of the variable, then it just chops off the extra bits from left to right to form the appropriate size. For example:

When adding $64 to $FFDF (DEC 100 and -33), the result would be $10043 but the Stamp chops the extra bit(s) off, leaving the result to be $0043 = 67.

Adding 2 large numbers

Sometimes there may be a need to add or subtract large numbers depending on the application. In order to do this, we can concatenate (not physically) several registers together. For example, suppose we wish to add 2, 32 bit numbers together:

Add: W1:W2 + W3:W4

W1 = $2
W2 = $FFFF
W3 = 0
W4 = 1

```
W1:W2      0002 FFFF
W3:W4      0000 0001
           ---------
W1:W2      0003 0000
           ---------
```

overflow results in this word (W2) being less than the original W2

Add 1 to W1 (or W3) before adding words W1 and W3 together.

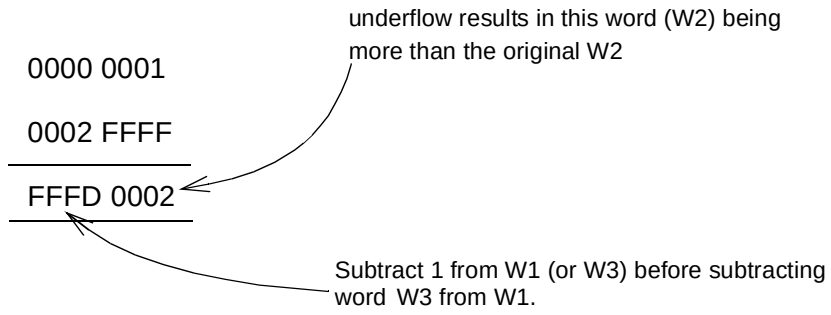
Program Example 1:

```
W1 = $2
W2 = $FFFF
W3 = 0
W4 = 1


W5 = W2                         'temporary storage for W2
W2 = W2 + W4
IF W2 >= W5 then nocy           'if W2  < (old) W2
W1 = W1 + 1

nocy:
W1 = W1 + W3
```

Subtracting 2 large numbers

Subtract: W1:W2 - W3:W4

```
W1 = 0
W2 = 1
W3 = $2
W4 = $FFFF
```



Program Example 2:

```
W1 = 0
W2 = 1
W3 = $2
W4 = $FFFF


W5 = W2                         'temporary storage for W2
W2 = W2 - W4
IF W2 <= W5 then nobor          'if W2  > (old) W2
```

W1 = W1 - 1

nobor:
W1 = W1 - W3

Multiplying 2 large numbers

Multiplying 2 16 bit numbers requires a 32 bit result. The '*' and '**' operators can be used for this purpose.

* --> returns value of bottom 16 bits

** --> returns value of top 16 bits

For example:

Multiplication of $FFFF and $FFFF produces the result $FFFE0001

W1 = $FFFF

W2 = $FFFF

W3 = W1 * W2

W4 = W1 ** W2

Non-integer numbers

The BASIC Stamp can only handle whole integer numbers. Therefore, methods of conversion to integer numbers is necessary depending on your exact needs. For the expression:

F = 1.8 * C + 32

can be re-written as

F = 18 * C + 320

This conversion however means that the actual result 1/10th of the result obtained from the calculation.

One way to deal with fractional numbers is to use the ' */ ' operator. It has the effect of multiplying a value by a whole number and a fraction. This operator places the whole number portion in the upper byte, multiplies the fractional part by 256 and places the result in the lower byte.

For example, if multiplying value with 1.8, using the ' */ ' would mean 0.8 * 256 = 204.8 = 205 and therefore:

Upper byte : 01

Lower byte : CD

so 1.8 can be represented as $01CD. Therefore, some care will need to be taken when dealing with non-integer numbers.

Now, F = 1.8 * C + 32 can be re-written as F = C */ $01CD + 32