For Macintosh Programmers & Developers
MacTech MAGAZINE

MICROCONTROLLER PROGRAMMING

By Tom Djajadiningrat & Marcelle Stienstra
Mads Clausen Institute for Product Innovation
University of Southern Denmark

Programming the BasicStamp

Using MacBS2

About the authors...
Tom Djajadiningrat and Marcelle Stienstra work at the Mads Clausen Institute for Product Innovation, a research and teaching facility of the University of Southern Denmark. When not gossiping about their colleagues on ICQ, they claim to teach interaction design to students on the IT Product Design masters course. You can reach them on j.p.djajadiningrat@mci.sdu.dk and marcelle@mci.sdu.dk.

ABSTRACT

This article explains how to program a BasicStamp microcontroller using a Macintosh. It takes an absolute beginner's perspective and talks you through the whole process including how to hook up a programmer board, how to write some programs in the BasisStamp language PBASIC, and how to do some simple input and output such as reading switches and controlling LEDs.

INTRODUCING THE BASICSTAMP

The BasicStamp by Parallax is probably the most popular microcontroller amongst electronics hobbyists. One of the reasons for this is that it can be programmed in a simplified, customized form of BASIC called PBasic, unlike other microcontrollers such as Microchip PICs or Atmel AVRs which usually need to be programmed in C(++) or assembler. A second reason is that the development tools are free. Once you have bought a programmer board and a BasicStamp microcontroller you do not need to make any further investments in compilers or integrated development environments. Finally, the documentation and website that Parallax provide are comprehensive and written in plain language with many examples ranging from beginner's level to some pretty complex stuff. In any case, Parallax seem to consciously avoid much of the electronic engineering jargon that you find in the documentation of other microcontroller manufacturers.

You may wonder: "If this BasicStamp thing is so marvelous why isn't every electronic product powered by one?". Of course, there are drawbacks to BasicStamps. The first is speed, as a BasicStamp is basically a PIC microcontroller with a BASIC interpreter stuck on top which causes a performance penalty. The second is control. The PBASIC that the BasicStamp uses hides much complexity but also moves you one level further away from the hardware, reducing the amount of control you have over it. Last but not least, the BasicStamp is expensive, ranging from about 50 to 90 US Dollars. Compared to PICs where 10 US Dollars buys you a top model, that is expensive.

With these characteristics it is hardly surprising that the BasicStamp is mainly used in situations in which ease and speed of development are more important than cutting edge performance or high volume production with low unit cost. This is usually the case with hobbyist work but also with some one-off prototypes and sometimes even with small production runs.

BASICSTAMP ON MAC

So what has all of this got to do with the Macintosh? For a long time, there were no Macintosh development tools for the BasicStamp, requiring you to use VirtualPC if you wanted to use a Mac [1]. Recently, however, that situation changed when Parallax introduced a PBasic Tokenizer library for OSX and Murat M. Konar introduced MacBS2, an OSX native programming environment for the BasicStamp [5]. This is what you need to get stamping on the Mac:

Hardware

- Macintosh running OSX.2 or later
- A Keyspan USB to serial adapter [2]. (costs: 40 USD for a USA-19, 60 USD for a USA-28x)

  You can either use a USA-28x (Figure 1) with two traditional Macintosh style Mini-Din8 connectors or a USA-19 (Figure 2) with a single PC style DB9 connector. The advantages of the USA-19 are that it is the least expensive Keyspan USB-serial adapter and that you don't have to modify and solder a serial cable as you can simply use a standard DB9 male-DB9 female cable. The advantages of the USA-28x are that you get two serial ports and that you may regain connectivity with legacy Macintosh serial peripherals. In a minute we will give you an explanation on how to make a suitable serial cable should you wish to use a USA-28x.

Figure 1: Keyspan USA-28x USB-serial adapter with two Mac-style Mini-Din8 connectors

Figure 2: Keyspan USA-19 USB-serial adapter with a single PC-style DB9 connector

- A BasicStamp 2 Variant (costs: 50-90 US Dollars)

There are a number of BS2 microcontrollers which differ in speed, number of I/O pins, supported calls and obviously price [3]. We chose a BS2p24 which is the fastest Stamp currently available (Figure 3). You cannot use the simpler, older and cheaper BS1, or the JAVA-based Javelin Stamp, simply because Parallax does not provide Mac-compatible tokenizer libraries for them (a tokenizer library is software that converts PBASIC source code to the "program tokens" that can be downloaded to BASIC Stamp modules). Therefore, it is not possible for MacBS2 to support these microcontrollers.

Figure 3: A BasicStamp. This particular one is a BS2p24

- A BasicStamp programming board (25-200 US Dollars)

Parallax makes a whole range of programming boards which differ in the number and type of connectors that are provided [4]. Some are very minimalistic, offering only a serial connector and a socket for the BasicStamp itself, others add such things as LCD connectors and breadboards (Figure 4). We chose one called the BS2p24 Demo Board because it has a small breadboard integrated on it which facilitates prototyping without soldering and connectors for some more advanced features of the BS2p24 such as I2C and iButton (if you don't know what these terms mean, don't worry about it).

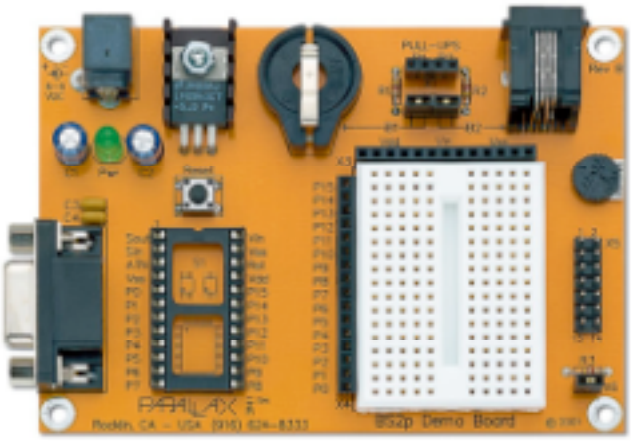

Figure 4: A BasicStamp programming board. In this case, a BS2p24 Demo Board

- A Power Adapter or a 9V battery (costs: 10 US Dollars)

  Depending on which programmer board you use you will need either a power adapter (6-9V) or a 9V battery.

Software

- Keyspan Drivers

  Download from http://www.keyspan.com/downloads/macosx/

  Make sure you have drivers that match your version of OSX. We use 1.8 which matches OSX.3 'Panther'.

- MacBS2, Mac Tokenizer and BasicStamp Manual

  Download from http://www.muratnkonar.com/MacBS2/

  What is very elegant about MacBS2, is that it automatically downloads and installs the Parallax tokenizer and BasicStamp Manual when you need them.

Total costs

If you do the maths, you will find that the stuff required for the BasicStamp will set you back between 115 and 360 US Dollars. The setup we are using cost us approximately 250 US Dollars.

CREATING A SERIAL CABLE TO CONNECT A BASICSTAMP PROGRAMMER BOARD TO A KEYSPAN USA-28X

(If you use a Keyspan USA-19 High Speed USB-serial adapter with a DB9 connector, you can safely skip this section: just buy a 'straight thru' male to female DB9 cable)

The information provided here is a more elaborate version for the electronically impaired of what you find in MacBS2 under the menu **Help**, item **Programming Wiring Diagram**.

Required components:

Mini-Din8 to Mini-Din8 Macintosh serial cable

DB9 male connector

The DB9 connector you can probably find at your local electronics store. The Mini-Din8 cable is not so current anymore so you may need to hunt around a bit. If all else fails, you can always buy one from www.maccables.com.

Instructions

To use a Keyspan USA-28x you need to make a cable with a male DB9 connector on one side and a male Mini-Din8 connector on the other. The easiest way to make one of these is to start with a Macintosh style Mini-Din8 to Mini-Din8 cable and cut it through the middle so that you end up with two cables, each of which have one end with a Mini-Din8 connector and one bare end with exposed wires. This way you don't have to solder the Mini-Din8 plug which has the Apple elegance and is therefore hell to solder and just need to make the end with the PC-style DB9 plug, which is lumpy, coarse and therefore easy to solder. The easiest way to get the cable right is:

- First, figure out which of the wires on the bare end of the cable are linked to the relevant pins of the Mini-Din8 plug. Figure 5 shows a head-on photo of the Mini-Din8 plug. Figure 6 shows the same view diagrammatically with the relevant pins indicated. Using a multimeter set to measure resistance, hold one probe on the pins of the Mini-Din8 plug and one on the bare wires. You are looking for the wires connected to the pins *gnd*, *sin*, *sout* and *attention*. Note down the colours of the wires. Table I shows which colours matched which pins in our case, though your cable may use different colours so tread carefully.

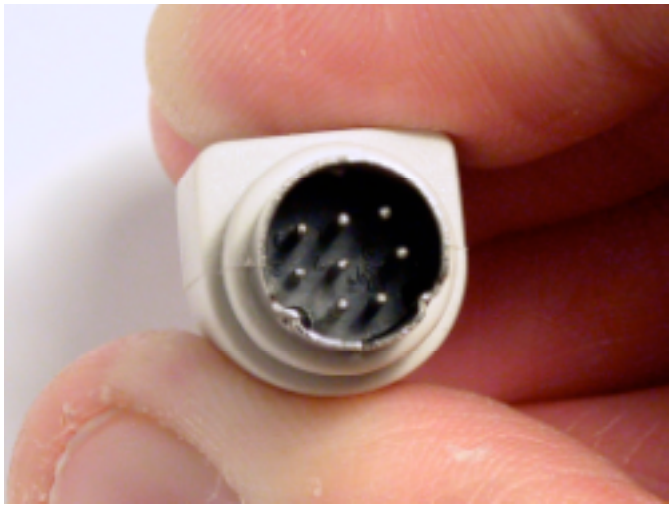

Figure 5: The Mini-Din8 plug head-on

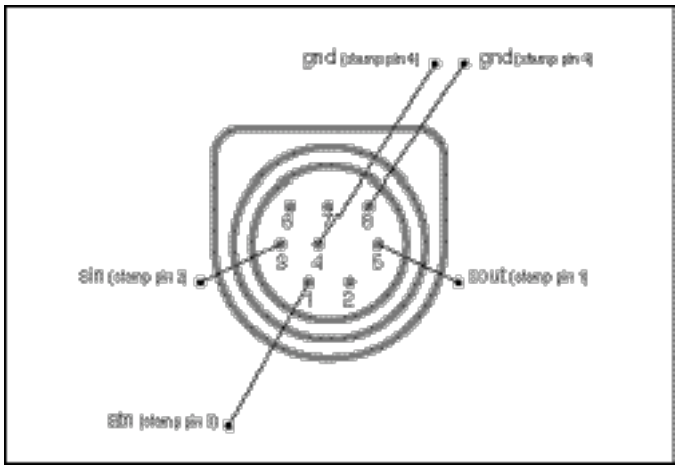

Figure 6: The Mini-Din8 plug head-on with the relevant pins indicated

mini-Din8 pin#	serial cable strand colour	DB9 pin#	BasicStamp pin name	BasicStamp pin#
5	orange	2	sout	1
3	green	3	sin	2
1	red	4	atn	3
4	yellow	5	gnd	4
6	blue	5	gnd	4

Table 1: a summary of the connections

- Secondly, we figure out which pins of the DB9 male plug match the relevant pins. Plug the loose DB9 male connector into the DB9 female connector on your BasicStamp board so that we can measure directly on the soldering side of the male DB 9 plug (Figure 7 and 8). With a multimeter, work out which pins of the DB9 plug are connected to pin 1 (sout), pin 2 (sin), and pin 4 (gnd) of the BasicStamp. Pin 3 (atn) is a bit of strange case as it is not possible to measure continuity between pin 3 on the BasicStamp and any pin on the DB9 plug. Still, even though continuity cannot be measured, pin 3 of the Stamp is connected to a pin on the DB9 plug in between gnd and sin. Looking at the soldering side of the DB9 plug, the connections are shown diagrammatically in Figure 9. You can refer back to Table 1 if you get stuck.

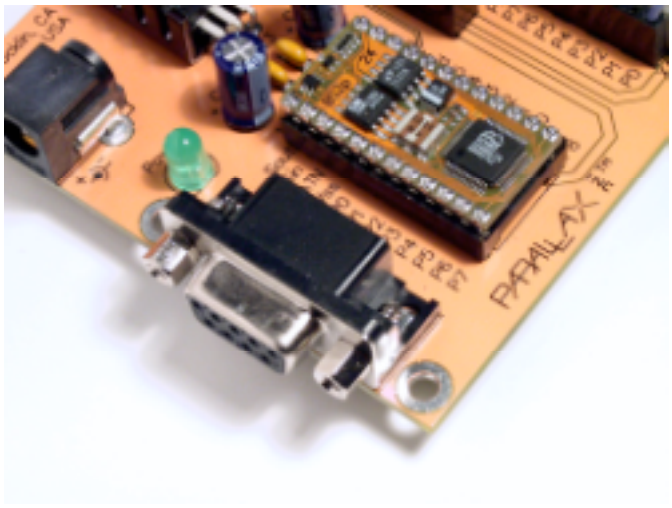

Figure 7: Your BasicStamp programming board has a female DB9 connector...

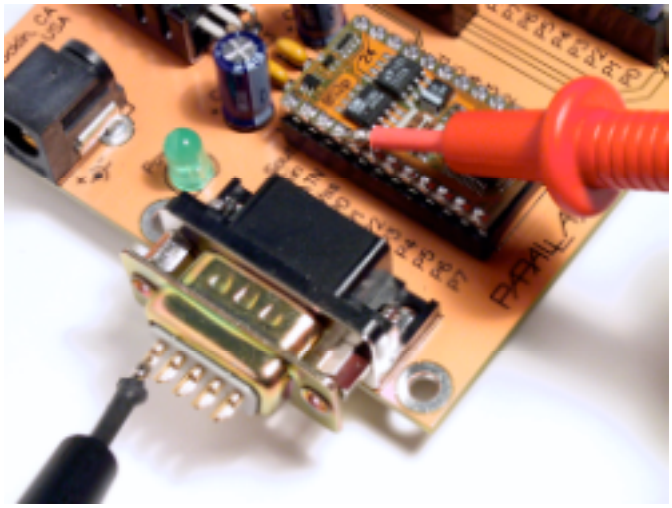

Figure 8: ...in which you can plug a DB9 male connector so that you can easily measure directly on the soldering side.

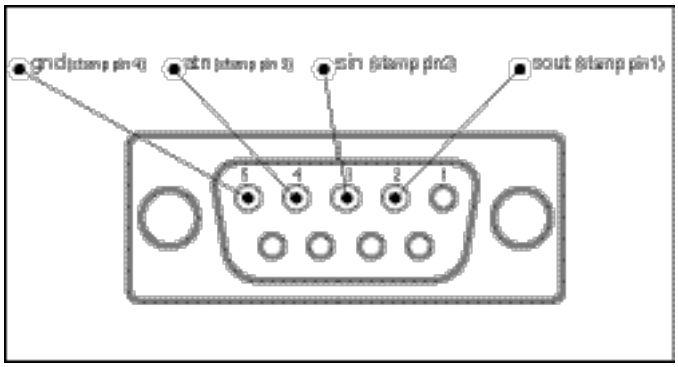

Figure 9: A diagram of the rear, soldering side of the DB9 plug.

- All that is left now, is to solder the DB9 plug. Before you put on the housing of the DB9 plug do a final check all the way from those MiniDin-8 pins are really connected to the right DB9 pins. The finished soldering job is shown in Figure 10.

Figure 10: The finished soldering job on the DB9 plug

CHECKING THE CONNECTION

- Hook up your Keyspan adapter to the USB port of your Mac.
- Do a sanity check: does the Mac acknowledge the Keyspan adapter? You can check by running the Keyspan Serial Assistant which should have ended up on your harddrive during installation of the Keyspan drivers, probably in your Applications folder. What you should see is either the dialog box in Figure 11 or Figure 12 depending on which model Keyspan adapter you use.

Figure 11: The Keyspan Serial Assistant acknowledging a USB-28x with two serial ports

Figure 12: The Keyspan Serial Assistant acknowledging a USB-19 with one serial port

- Plug your BasicStamp into your programmer board.
- Connect the power supply or the 9V battery to the programmer board. The power LED on the board should light up.
- Now connect your Keyspan adapter to the programmer board with your serial cable.
- Start up MacBS2. In the pop-up menu in the windows menubar you should see either one serial port for a Keyspan USA-28x (Figure 13) or two for a Keyspan USA-19 (Figure 14).

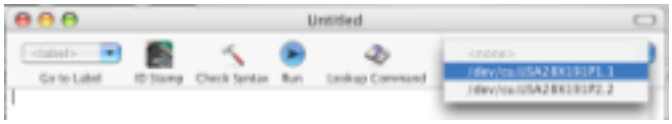

Figure 13: The port popup menu in MacBS2 for a Keyspan USA-28x

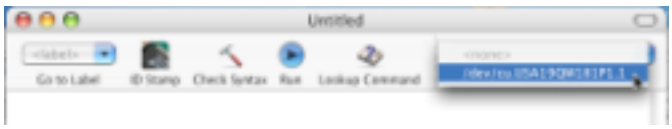

Figure 14: The port popup menu in MacBS2 for a Keyspan USA-19

- Click the **ID Stamp** button in the toolbar. The status bar at the bottom of the MacBS2 window says something like: "Found Basic Stamp 2p, firmware version = 1.2". Of course, the details may vary according to the type of Stamp and firmware you are using.

No joy? Check your physical connections, check that the programmer board is powered, make sure that you have correctly inserted the BS2 microcontroller and verify that you have the latest Keyspan drivers.

EIGHT STEPS TO BECOMING A PBASIC GURU

Time to do some Stamping! The easiest way to become familiar with the code-upload-debug cycle is to simply work your way through some small PBasic programs. Here is a set of eight, which will familiarize you with printing to the console window, variables, conditionals, loops and subroutines. Whether you save each program and open a new MacBS2 window for the next one and or simply modify the previous one is up to you.

1. Debug

At the start of every PBasic program, you need to tell MacBS2 which version of the BS2 you are programming. The easiest way to do this is to click the **Check Syntax** button. This brings up a dialog box with a popup menu, allowing you to choose between different models BS2 (Figure 15). Pressing the **Fix** button causes MacBS2 to insert a Stamp directive at the top of your file. We use a BS2p24 Stamp and therefore choose a BS2p from the popup menu causing MacBS2 to insert the code:

```
'{$STAMP BS2p}
```

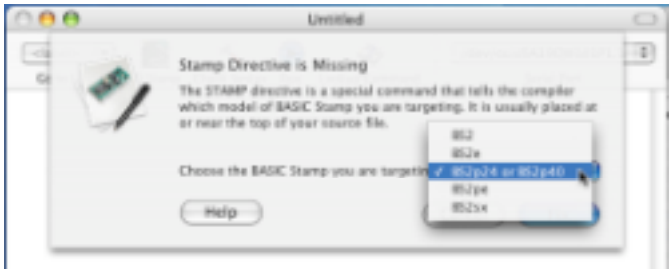

Figure 15: Choosing the type of BasicStamp

In all other cases, the ' (apostrophe) starts a comment, causing MacBS2 to ignore the rest of the line. Strangely enough, the stamp directive is essential and still starts with an apostrophe. Basically, it is a kind of include statement, telling MacBS2 which kind of stamp we are addressing. Complete the program as per Listing 1.

Listing 1: Debug

Debug

```
'{$STAMP BS2p}

'Our first PBasic program
DEBUG "Hello World!"
```

Now, press the check syntax button (or command-K) and then the run button (or command-R). The text "Hello World!" should appear once in the debugger pane. You may have noticed that some of the code is in lower and some is in uppercase. This does not actually matter: PBasic code is not case sensitive. It does make it a little easier to decipher the code when you write PBasic reserved words in capitals and everything else in lowercase (Unlike the PC version, MacBS2 does not automatically convert PBasic reserved words to uppercase).

If you have a reset switch on your BasicStamp programming board, pressing it causes the program to run from the beginning. So in this case, everytime you press the reset button, it writes "Hello World!" to the debugger pane.

This may not be the most convincing of BasicStamp programs: how can we be sure that the Mac is not simply printing the string directly to the debugger pane? Well, unplug the serial cable from the BasicStamp programming board, press the reset button on your board to run your program from the beginning, and you'll see that the printing to the debugger pane does not happen. Plugging the serial cable back in restores the programs functionality. So, the program does actually run on the Stamp and sends the string over the serial connection to the Mac.

2. Variable

Open a new MacBS2 window and type in the code in Listing 2.

Listing 2: Variable

Variable

```
'{$STAMP BS2p}

' variable declaration
gNumber  VAR BYTE
```

```
' variable initialization
gNumber = 42

'print gNumber to the debugger pane
DEBUG DEC gNumber, CR
DEBUG DEC ? gNumber
```

Press the **check syntax** button, then the **run** button. We start again with the stamp directive. What follows is a variable declaration: first the name of the variable (gNumber), then the reserved word VAR and finally the amount of memory that we want the Stamp to set aside for the variable, in this case a byte. This means that the variable can range from 0-255 in decimal, which can therefore easily store the number 42. All variables in PBasic are global. Just to remind you, we start the variable name with the letter g. We then initialize the variable gNumber with the value 42. Finally, we print the variable `gNumber` to the debugger pane in two ways. The first simply prints the value of the variable in decimal resulting in `42` and the carriage return forces the entry point to the next line. The second uses the `DEC ?` notation which makes the Stamp precede the value of the variable in decimal by `variableName =` and automatically follows it with a carriage return, resulting in:

```
gNumber = 42
```

3. Goto

Ah, the evil goto. In PBasic you need to create a label ending with a colon before you can use it as an argument in a goto statement. Listing 3 shows a simple code example.

Listing 3: Goto

Goto

```
'{$STAMP BS2p}

loop:

  DEBUG "Hello World!", CR

  GOTO loop
```

Press the syntax button, then the run button. This program simply causes the Stamp to write "Hello World!" to the debug pane endlessly.

4. For/next

A for/next statement in PBasic looks like Listing 4.

Listing 4: For/next

For/next

```
'{$STAMP BS2p}

' declare a variable and reserve a nibble of memory
gCounter VAR NIB

for gCounter = 1 to 3
  debug DEC ? gCounter
next
```

Press the **check syntax** button, then the **run** button. This prints the variable gCounter to the debug pane three times. Note that the amount of memory reserved for gCounter is a nibble (NIB) or half a byte. Yes, that is how thrifty you need to be with

memory on a microcontroller: you need gCounter to count up to only three. A nibble ranges from 0-15 which is more than sufficient, so why use a full byte?

5. Out and pause

Finally, we are ready to connect some electronic components and make something happen in 'the real world'. What you need is an LED and a 330Ohm resistor. The schematic of the circuit is shown in Figure 16. Figure 17 shows how to connect this on the breadboard of a BS2p24 Demo Board. The LED is connected to pin 3.

Type in the code in Listing 5.

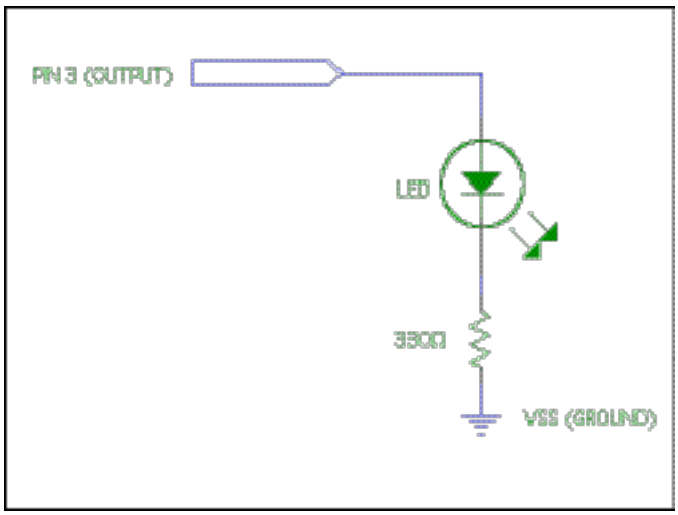

Figure 16: An LED connected on pin 3

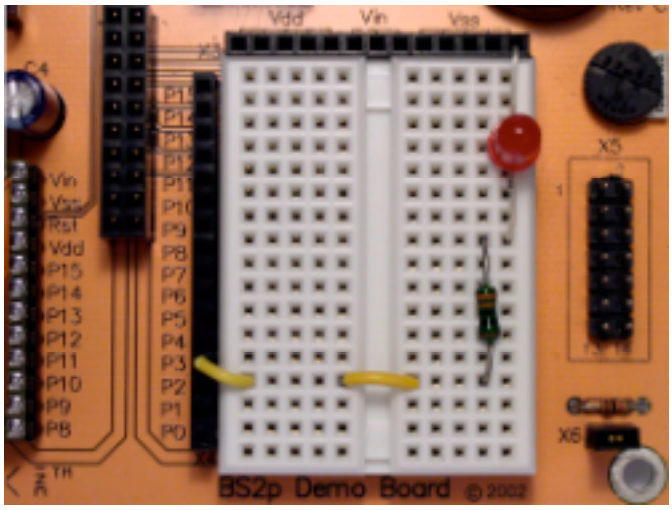

Figure 17: What things look like on the breadboard

Listing 5: Out and pause

Out and pause

```
'{$STAMP BS2p}

OUTPUT 3
```

```
loop:

  OUT3 = 1
  DEBUG "LED on", CR
  PAUSE 500

  OUT3 = 0
  DEBUG "LED off", CR
  PAUSE 1000
  GOTO loop
```

The new commands in here are OUTPUT, OUT and PAUSE. OUTPUT 3 makes pin 3 act as a digital output which can either be 0 or 1. OUT3 actually sets pin 3 to 0 or 1. PAUSE takes an argument between 0 and 65535 causing the BasicStamp to wait for that many microseconds. So the PAUSE 500 causes the Stamp to delay for 0.5s whilst PAUSE 1000 causes a delay of a full second.

Check the syntax and run the program. The LED should blink in a pattern of on for 0.5 second and off for 1 second.

No joy? Remember that LEDs are directional: current only flows from the anode (+) to the cathode (-). The cathode is usually indicated by a flat side on the LED's housing or by a shorter lead. In this case we have chosen to connect the cathode to ground and the anode to the resistor.

Confused about resistor values? There is a very handy web-based resistor calculator available [6].

6. Button

You are now ready to try your hands at some interactive stuff: press a button and an LED will light up. Let's try using a pin as an input. Add the circuit as in the schematic in Figure 18 to what you already built in Figure 17. Our breadboard ended up looking like Figure 19. Type in the code in Listing 6.

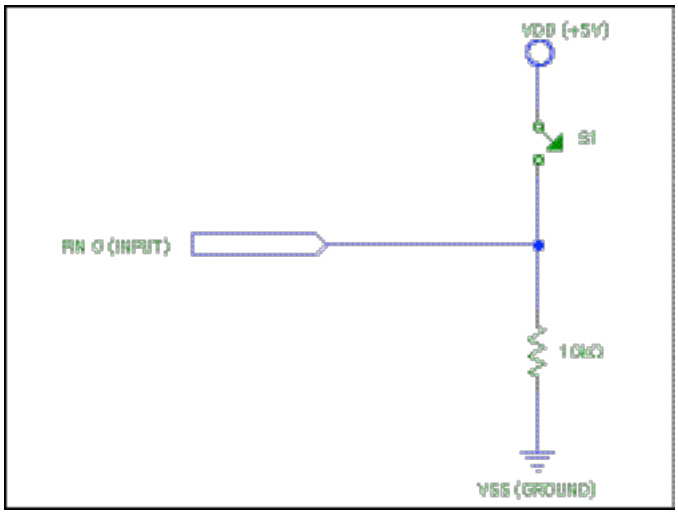

Figure 18: The active-high button circuitry

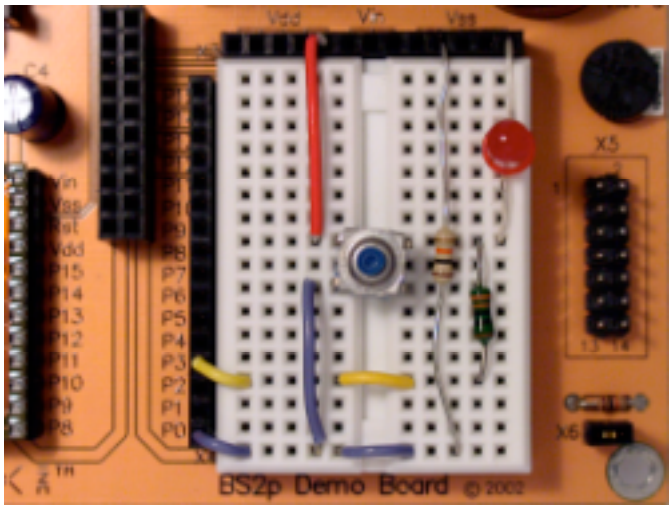

Figure 19: The LED output and button input circuitry together

Listing 6: Button

Button

```
'{$STAMP BS2p}

gWorkSpace   VAR BYTE

'make pin 3 an output
OUTPUT 3


loop:

   'BUTTON Pin, DownState, Delay, Rate, Workspace,
   ' TargetState, Address
   BUTTON 0, 1, 0, 0, gWorkSpace, 1, press
   DEBUG "no, not pressed", CR
   OUT3=0
   goto loop


press:

   DEBUG "yes, pressed", CR
   OUT3=1
   GOTO loop
```

The only new PBasic call in this listing is BUTTON. BUTTON is a conditional depending on the binary state of an input. It thus forms a kind of convenient mix of monitoring an input and an IF/THEN statement. It always needs to be in a loop and comes with no less than seven parameters.

The first, Pin, indicates which pin of the Stamp we are looking at. The second, Downstate tells the Stamp for which logic state we are looking for at the pin. Delay and Rate are best explained together. The Button command makes it possible to autorepeat, similar to what happens when you keep a key of your Mac pressed. The Delay parameter indicates how long it takes before autorepeat starts whilst the Rate parameter determines the intervals between autorepeats. Both parameters are expressed in the number of cycles

of the loop containing BUTTON. Perhaps a more important use of Delay is the debounce feature, which prevents the BUTTON statement being triggered multiple times due to scraping contacts of mechanical switches This is useful when you are looking for a single button press. To activate debounce, you use Delay = 255. To deactivate debounce, you use Delay = 0. In this example, we use neither debounce nor autorepeat.

Workspace is a byte variable for use by BUTTON. Basically, if you declare it once and never use it for anything but the BUTTON command, everything will be ok. The sixth parameter, TargetState, determines on which state of the button we want the branch to occur. 0 = not pressed and 1 = pressed. Beware of the difference between the DownState and TargetState parameters. Things can get confusing as the behaviour also depends on whether the switch circuitry is active high or active low and on whether you use a push-to-make or push-to-break switch.

Here we have chosen what we think is the easiest situation to get your head round: we use a push-to-make switch with active high circuitry, which therefore causes a logic high on the input pin when pressed. This means that we use DownState = 1. Also, we find it helps readability of the code to branch when the button is pressed and to continue when the button is not pressed and for that we need TargetState = 1.

Ofcourse, other combinations of circuitry, switches, DownState and TargetState can work fine too, just make sure you think things through carefully to avoid strange behaviour.

7. If/Then

Type in the code in Listing 7.

Listing 7: If/Then

If/Then

```
'{$STAMP BS2p}

gWorkSpace     VAR BYTE
gButtonCounter VAR NIB

gButtonCounter = 0

loop:

  BUTTON 0,1,255,0,gWorkSpace,1,Press

noPress:

  DEBUG "no, not pressed. ", DEC ? gButtonCounter

  GOTO loop


press:

gButtonCounter = gButtonCounter + 1
DEBUG "yes, pressed. ", DEC ? gButtonCounter

if gButtonCounter = 3 then lastMessage
GOTO loop

lastMessage:
```

```
DEBUG "Button pressed 3 times"
```

The program counts button presses and ends when the user has pressed the button three times. Note that this time we use the BUTTON command with Delay = 255, meaning that debouncing is on and autorepeat off. This is ideal for detecting a single button press.

8. Gosub

One more step and you will have reached your PBasic Guruship. This example shows how to use subroutines in PBasic. Type in the code in Listing 8.

Listing 8: Gosub

Gosub

```
'{$STAMP BS2p}

gCounter VAR NIB


FOR gCounter = 1 to 3

   ' calling a subroutine
   GOSUB debugCounterValue

NEXT

STOP


' example of a subroutine
debugCounterValue:

   DEBUG "start subroutine", CR
   DEBUG DEC ? gCounter
   DEBUG "end subroutine", CR, CR

   RETURN
```

A subroutine needs to start with a label and end with a RETURN statement. GOSUB causes a jump to a label of that name. The code in the subroutine is then executed and the RETURN statement causes the Stamp to jump back to the line following the GOSUB.

Note the use of the STOP command after the main program. It prevents us running into the subroutine accidentally after the main loop has executed three times.

CONCLUSION

So there you have it. You can now program a BasicStamp on your Mac and do some simple input and output. Try to address some other pins using multiple LEDs or buttons. The BasicStamp Manual is very friendly, so once you start playing with a Stamp you'll quickly find yourself exploring new commands and syntax. The Parallax website has links to some great application notes from Nuts & Volts magazine in which you can find info on hooking up various sensors, servos, stepper motors etc [7]. Enjoy!

REFERENCES

1. If for some incomprehensible reason you insist on using Windows-based BasicStamp development tools under VirtualPC emulation, here is how:

http://www.robotstore.com/download/Stamps_on_iMac_2.13.pdf

2. Keyspan make USB-serial adapters:

   http://www.keyspan.com

3. Here you find a full list of all BasicStamp models:

   http://www.parallax.com/html_pages/products/basicstamps/basic_stamps.asp

4. Here you find a list of all the programmer boards available:

   http://www.parallax.com/html_pages/products/boards/programming_boards.asp

5. This is the home of the MacBS2 software:

   http://www.muratnkonar.com/MacBS2/

6. There is a handy JavaScript-based calculator floating around the net for working out resistor values from colour codes. Here is one site that has it:

   http://www.dannyg.com/examples/res2/resistor.htm

7. Here you find the Nuts & Volts application notes, ranging from very basic to some pretty hairy stuff:

   http://www.parallax.com/html_pages/downloads/nvcolumns/Nuts_Volts_Downloads.asp