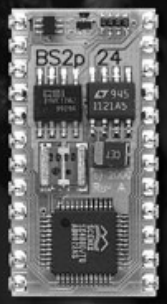
BASIC Stamp 2p

Commands, Features and Projects



by Claus Kühnel and Klaus Zahnert
for Parallax, Inc. Press

Warranty

Parallax warrants its products and printed documentation against defects in materials and workmanship for a period of 90 days. If you discover a defect, Parallax will, at its option, repair, replace, or refund the purchase price. Simply call for a Return Merchandise Authorization (RMA) number, write the number on the outside of the box and send it back to Parallax. Please include your name, telephone number, shipping address, and a description of the problem. We will return your product, or its replacement, using the same shipping method used to ship the product to Parallax.

14-Day Money Back Guarantee

If, within 14 days of having received this book, you find that it does not suit your needs, you may return it for a full refund. Parallax will refund the purchase price of the product, excluding shipping / handling costs. This does not apply if the book has been altered or damaged.

Copyrights and Trademarks

Disclaimer of Liability

Parallax, Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs or recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your microcontroller application, no matter how life threatening it may be.

Internet Access

We maintain Internet systems for your use. These may be used to obtain software, communicate with members of Parallax, and communicate with other customers. Access information is shown below:

Web: http://www.parallax.com

Internet BASIC Stamp Discussion List

Parallax maintains an e-mail discussion list for people interested in BASIC Stamps. The “basicstamps” list server includes engineers, hobbyists, and enthusiasts. The list works like this: lots of people subscribe to the list, and then all questions and answers to the list are distributed to all subscribers. It’s a fun, fast, and free way to discuss BASIC Stamp programming issues and get answers to technical questions. This list generates about 40 messages per day and has 2,300 subscribers. Subscribe at www.yahoogroups.com under the group name “basicstamps”.

Preface

The Parallax BASIC Stamp 2p extends the family of the well-known BASIC Stamp microcontrollers with a wider feature set for an increased number of applications. The BASIC Stamp 2p has an extended instruction set suitable for more complex projects, making BASIC Stamp 2 projects easier as well. The BS2p supports the Philips I^2C protocol and the Dallas Semiconductor 1-Wire bus along with the direct control of Hitachi-compatible HD44780 alpha-numeric LCDs.

For a long time Parallax customers requested an interrupt capability. The implementation of a powerful polling feature is an answer to this request. Enhancements of additional memory and I/O were also added to let the BS2p to do "double duty" as a datalogger.

In this book we will explain the entire BASIC Stamp 2 family and provide numerous code snippets for using these Parallax microcontrollers.

This book is an English translation of our German book "BASIC Stamp 2 – Neue Eigenschaften – neue Projekte" (ISBN 3-907857-02-X) published in 2002 with numerous updates for the American market.

As manufacturer of the BASIC Stamp, Parallax offers significant information and application support on its web site or in printed form. Do not miss a visit on Parallax's web site [www.parallax.com]. This level of support has differentiated Parallax for years and is certainly a key to the product line's success.

All code examples used in this book are available for download from http://www.parallax.com/bs2pbook.

The authors thank Parallax and especially Ken Gracey for the support of this book. Though Parallax edited the book in detail, the possibility that German language or mannerisms appear in the book remains. If you find such errors please report them to Parallax using e-mail (info@parallax.com).

Klaus Zahnert and Claus Kühnel

Table of Contents

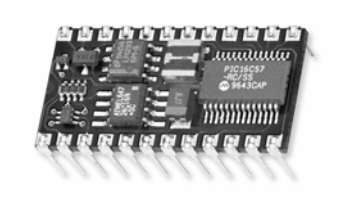
1 BASIC Stamp – an Overview

For years Parallax's BASIC Stamp microcontrollers have been well-known for their ease of use, comfortable programming language and easy debugging using a PC.

BASIC Stamps are not just for engineers (one could also say they are not just for hobbyists, too). Everybody interested in measurements, control and human interaction with electronic circuits will find ease of entry and continuous success with these devices. It's amazing what can be done with a small feature set.

Parallax's educational program "Stamps in Class" introduces new interested parties to this subject with well-designed free tutorials and ready-to-implement class curriculum.

The BASIC Stamp 2 family consists of several variations. Though an overview of technical specifications of all models is helpful to see the differences, Parallax often leads newcomers to the BASIC Stamp 2 and BASIC Stamp 2p due to extensive application support.

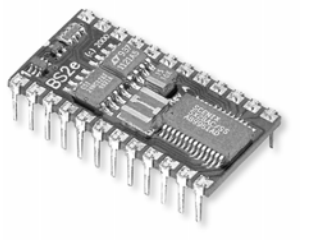


BS2

Microcontroller	PIC16C57 SMD
Clock	20 MHz
EEPROM	2K Bytes
Length of program	500 Lines PBASIC Code
RAM (Variable)	6 I/O, 26 Variable
Input/Outputs	16
Output current(Source/Sink)	20 mA / 25 mA
Current consumption	7 mA Run, 50 µA Sleep
PC Interface	Serial Port
Package	24-Pin DIP Module (green PCB)
Dimensions	30 mm L x 16 mm W x 9 mm H

BS2sx

Microcontroller	Ubicom SX28AC/SS
Clock	50 MHz
EEPROM	8 x 2K Bytes
Length of program	4000 Lines PBASIC Code
RAM (Variable)	6 I/O, 26 Variable plus 63 Byte Scratch Pad RAM
Input/Outputs	16
Output current(Source/Sink)	30 mA/30 mA
Current consumption	60 mA Run, 200 µA Sleep
PC Interface	Serial Port
Package	24-Pin DIP Module (blue PCB)
Dimensions	30 mm L x 16 mm W x 9 mm H



BS2e

Microcontroller	Ubicom SX28AC/SS
Clock	20 MHz
EEPROM	8 x 2K Bytes
Length of program	4000 Lines PBASIC Code
RAM (Variable)	6 I/O, 26 Variable plus 63 Byte Scratch Pad RAM
Input/Outputs	16
Output current(Source/Sink)	30 mA/30 mA
Current consumption	20 mA Run, 200 µA Sleep
PC Interface	Serial Port
Package	24-Pin DIP Module (red PCB)
Dimensions	30 mm L x 16 mm W x 9 mm H

BS2p

Microcontroller	Ubicom SX48AC
Clock	20 MHz Turbo
EEPROM	8 x 2K Bytes
Length of program	4000 Lines PBASIC Code
RAM (Variable)	12 I/O, 26 Variable
Input/Outputs	32 Bytes (6 for I/O and 26 for Variables) plus 32 Byte Scratch Pad RAM
Output current(Source/Sink)	30 mA / 30 mA
Current consumption	40 mA Run, 60 µA Sleep
PC Interface	Serial Port
Package	24-Pin or 40-Pin DIP Module (gold PCB)
Dimensions	24-Pin: 30 L x 16 W x 9 H (mm) 40-Pin: 53 L x 16 W x 9 H (mm)

1.1 What is a BASIC Stamp?

Anyone experienced with a BASIC Stamp can already able to answer this question. Skip to the next subchapter if you do not need this information.

Users with no BASIC Stamp experience will find the next few pages to be useful background information.

The BASIC Stamp is a single board computer containing the microcontroller, an EEPROM, a voltage regulator and reset circuitry. As Figure 1 shows, 24 I/O pins are available for peripherals.

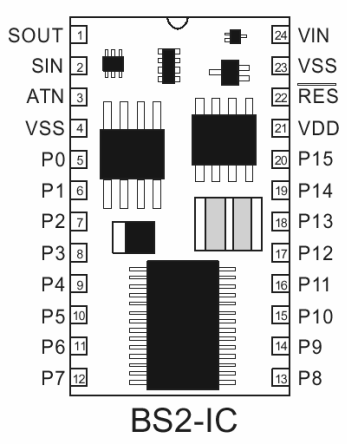


Figure 1 BS2-IC

The pinout listed in the following table shows all I/O resources in detail. In the left column the notation BS2x-24 stands for all 24-pin BS2 modules according to Figure 1, while the second column applies to the 40-pin BS2p only.

Pin BS2x-24[1]	*Pin BS2p-40*	*Name*	*Function*
1	1	SOUT	Serial output (to RxD of the PC COM Port)
2	2	SIN	Serial input (from TxD of the PC COM Port)
3	3	ATN	Attention (to DTR of the PC COM Port)
4	4	VSS	Ground (to GND of the PC COM Port)
5-20	5-20	P0-P15	Digital I/O pins
	21-36	X0-X15	Digital I/O pins
21	37	VDD	+ 5 V DC I/O (stabilized)
22	38	RES	Reset I/O
23	39	VSS	Ground
24	40	VIN	+ 5,5 - 12 V DC input (not stabilized)

Table 1. The "x" refers to a model number of BASIC Stamp in the BS2 series.

Connecting 5 to +12 VDC to V_{IN} and the internal voltage regulator provides a stabilized voltage of +5 VDC at V_{DD}.

If you want to power the BASIC Stamp with a reliable voltage of +5 VDC then connect it to V_{DD} directly. The V_{IN} pin could remain disconnected in this case.

With the reset pin it is quite similar. An internally generated Reset (power-down reset) pulls the RES pin low during the reset phase. An external pulling of RES to low can also force a reset. Both pins (V_{DD} and RES) have I/O characteristics.

The BASIC Stamp Module must be supplied with one voltage only (possibly a battery) and will run a program after program immediately after downloading.

This sounds very easy and it is! Figure 2 shows the complete programming infrastructure required.

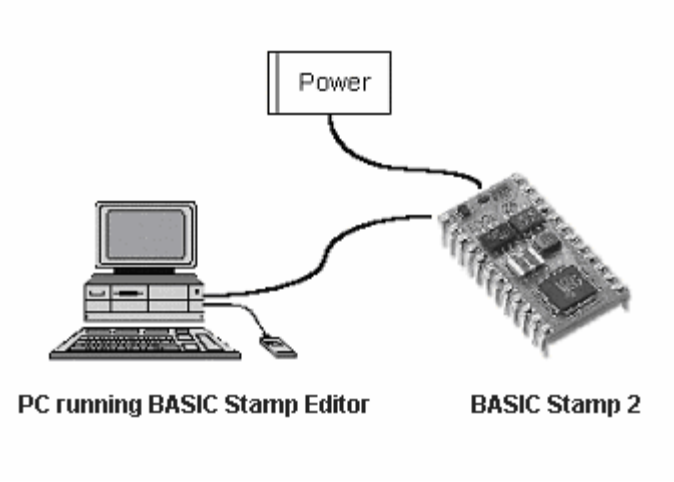


Figure 2 BASIC Stamp Development Environment

If you want to save some money, you can make the required download cable connections from a standard serial cable with the BASIC Stamp module plugged into a breadboard. Figure 3 shows the programming connection. The BASIC Stamp Editor is available as a Windows or DOS program from Parallax's web site for free. At the time of this publication Parallax is about to release a link for Linux, Macintosh and Palm operating systems that would enable developers to design their own IDE and download environments for different operating systems.

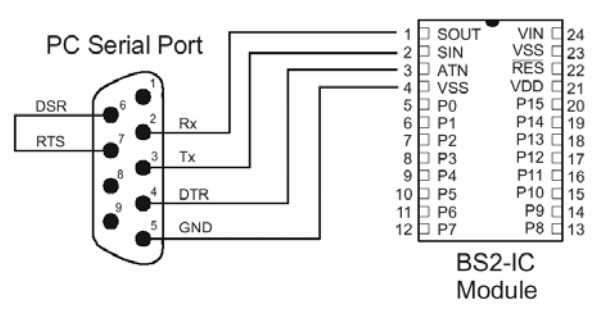


Figure 3 BS2 Download Connection

Let's have a look at how an application could be coded to run on the BASIC Stamp.

The BASIC Stamp Editor must first be installed on the development PC. Parallax offers a detailed "Quick Start Guide" in the BASIC Stamp Manual.

Parallax offers a DOS Editor for each type of BS2 and a single Windows Editor for all types of BS2s.

Figure 4 shows the BASIC Stamp Editor StampW. This programming software is very easy to use.

Figure 4 BASIC Stamp Editor StampW

If the PBASIC source is free of any errors, then it can be compiled and downloaded to the BASIC Stamp.

The downloaded tokens will be saved in the BASIC Stamp's external EEPROM. The BASIC Stamp's microcontroller contains the PBASIC firmware called a "token interpreter". This token interpreter is responsible for running the downloaded tokens and represents Parallax's core intellectual property. The download procedure is the same for all types of BS2s and should not considered in detail here but is discussed in some detail in other resources found on the web (see Brian Forbes' book "Inside the BASIC Stamp 2").

A list of all available PBASIC commands follows in one of the next chapters.

1.2 StampW Editor

Using StampW you can develop programs for all types of BS2s in a Windows environment. StampW can be used from an intuitive standpoint like other Windows application programs. Therefore, this book describes only the more specific features.

To tell the compiler which type of BS2 is described in the PBASIC some innovations were introduced:

STAMP Directive

Different file extensions

Default Stamp Mode (set over the menu **Edit>Preferences**)

The STAMP directive must be used at the beginning of the program. For example, this directive looks for a BS2p:

```
'{$STAMP BS2p}
' Directive shows that this is a BS2p program
```

If you don't insert the directive the BASIC Stamp Windows Editor will add it for you by presenting a pull-down menu and asking you to select a BASIC Stamp.

For the other members of the BS2 family the directives are as follows:

```
'{$STAMP BS2}           ' valid for BS2
'{$STAMP BS2sx}         ' valid for BS2sx
'{$STAMP BS2e}          ' valid for BS2e
```

For the file extensions the following is valid:

filename.bs2 characterizes a source file for BS2
filename.bsx characterizes a source file for BS2sx
filename.bse characterizes a source file for BS2e
filename.bsp characterizes a source file for BS2p

Using the **Edit>Preferences** menu you can set the Default Stamp Mode and different directories for saving the source programs. Figure 5 shows the setup possibilities in the menu **Edit>Preferences>Editor Operation**.

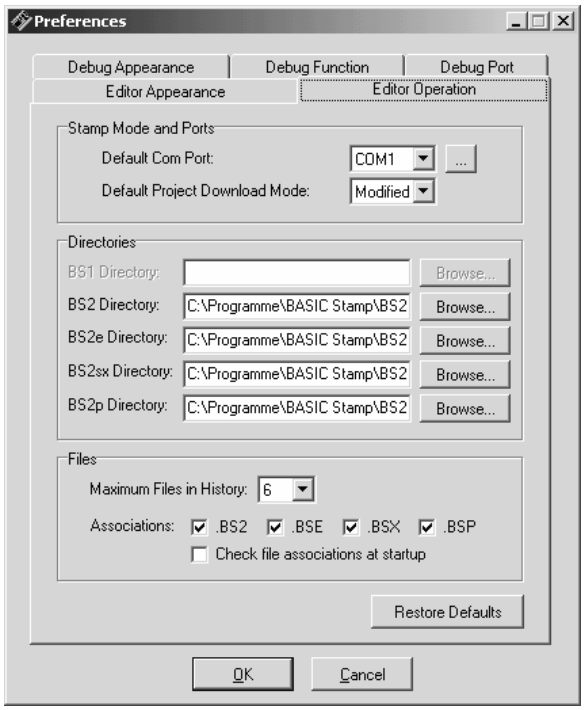


Figure 5 Directory Setup

To test the communication with a connected BASIC Stamp, you can use the Identify Function in the **Run>Identify** menu. Figure 6 shows the response from a BS2p after you've sent the Identify command.

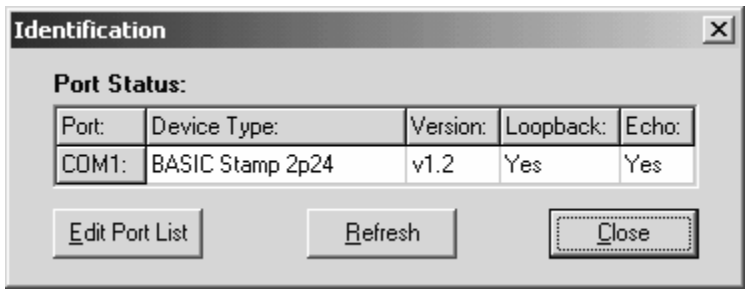


Figure 6 Answer to Identify

Before download you can check the syntax with **Run>Check Syntax**. Syntax errors found will be marked – but undefined commands as "nonsense", for example, stay unrecognized.

The use of the BASIC Stamp resources can be inspected via **Run>Memory Map**. Figure 7 shows a memory map from this pull-down menu.

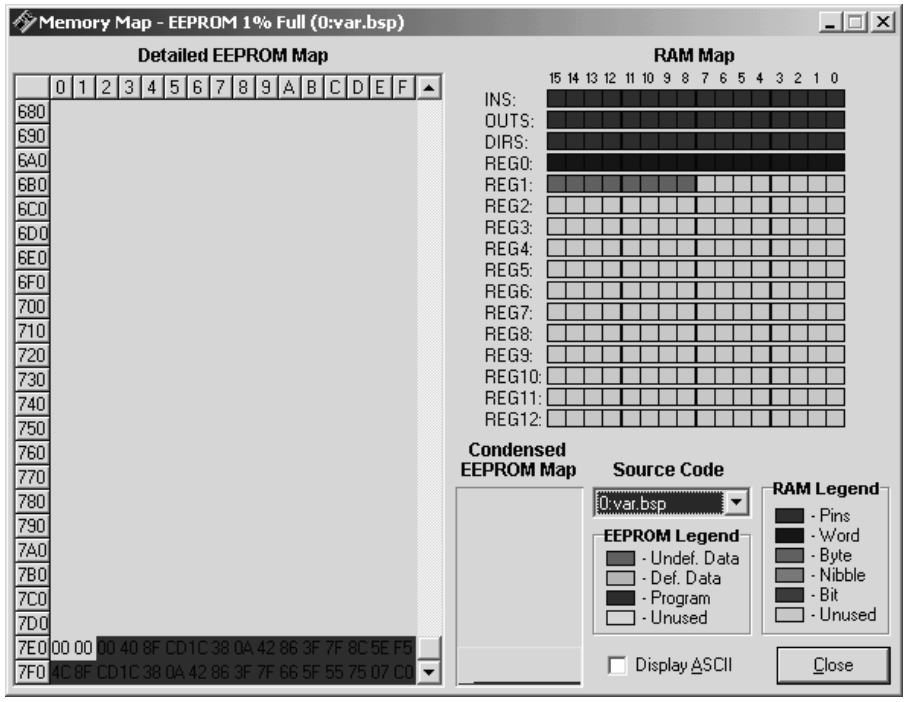


Figure 7 Memory Map

In the left half of the memory map you'll see the 2 KByte EEPROM. For BS2p and BS2sx this is only one slot of program memory. On the right side all 32 bytes of RAM are listed. The Scratch Pad RAM is not shown.

To the usage of EEPROM and RAM by the BASIC Stamp is detailed in the following examples.

StampW offers some new features with the Debug Window. Using the **Run>Debug** menu you can open the Debug Window. Output from the BASIC Stamp's Debug command will be redirected automatically to the Debug Window. But this isn't all it can do.

The programming port can be used for bi-directional communication between a Debug Window and a BS2. The following example demonstrates bi-directional communication on a BASIC Stamp Activity Board:

```
'{$STAMP BS2p}

serstring var byte(3)

loop:
     DEBUG CLS,"D: Waiting for 3 chars from Debug Window..."
     ' SERIN 16, 84, [STR serstring\3]                          ' for BS2
     SERIN 16, 240, [STR serstring\3]                           ' for BS2p
    DEBUG CR,"D: String = ",STR  serstring, CR, CR
    ' SEROUT 16, 84, ["Reflected characters: ",STR serstring\3]' BS2
    SEROUT 16, 240, ["Reflected characters: ",STR serstring\3] ' BS2p

L1:
     IF IN11 = 0 THEN L1
     ' press red key on activity board to proceed
goto loop
```

After the start of the program the DEBUG command sends the string “D: Waiting for 3 chars from Debug Window…” and the SERIN statement waits for exactly three characters. Operating with I/O “Pin 16”, SERIN is redirected to the PC’s serial port and the BASIC Stamp’s DEBUG window. The programmer interface communicates with the Debug Window with the DEBUG command.

After receiving these three characters the second DEBUG command sends the string "`D: String = ___`". The characters received replace the underlined part of the string. After the DEBUG command is executed the BS2p sends the received characters with SEROUT command to the DEBUG Window.

At the end of the loop I/O Pin 11 is checked (if you’re using the BASIC Stamp Activity Board this is the red button) and the program will decide whether or not the whole procedure should be repeated.

Sometimes long instruction lines in our program examples will be broken with what appears to be a carriage return. Please pay attention that in the first column where a label begins a routine. Exceptions are the definitions at the beginning of the program. If you see a character in the first column in the body of the program then this instruction line was too long to print. During compilation such a broken line would be found and signalizes an error. Note that all program examples are available from download from www.parallaxinc.com/bs2pbook.

Figure 8 shows the communication between the BS2p and the Debug Window for the test program.

The Debug Window has two important aspects. The large area in the lower half of the window shows all output of DEBUG and SEROUT 16,..., while the white box above serves as an input line for those characters expected from SERIN.

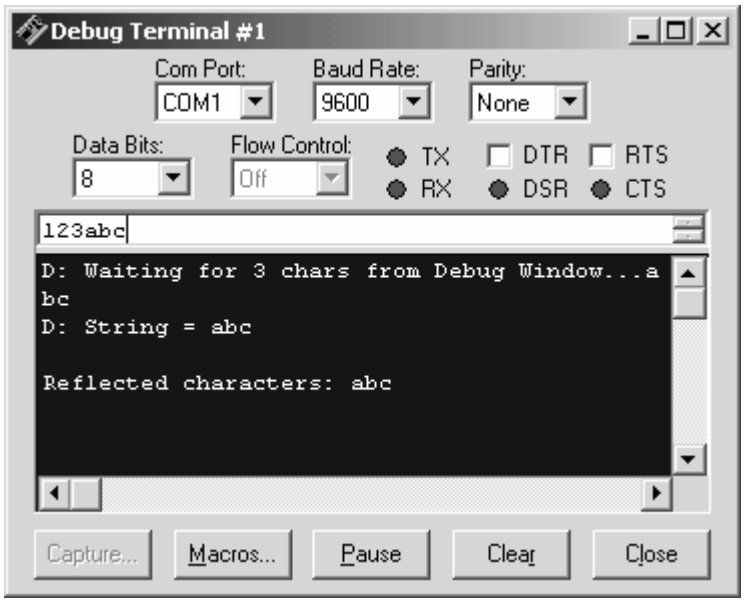


Figure 8 Debug Window

Defining macros can help to simplify the repeated key-pressing during debugging.

Figure 9 shows two macros we defined called *String1* and *String2*. If one presses Ctrl+Shft+A during debugging then the characters "xyz" will be sent to the BASIC Stamp.

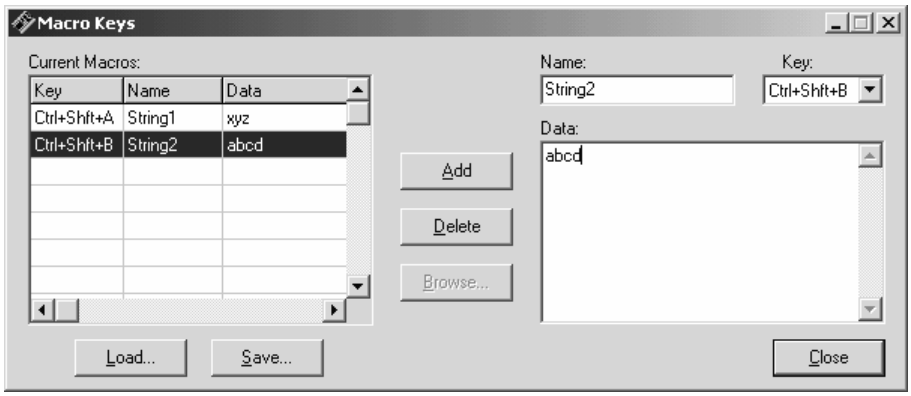


Figure 9 Defining Macros

You can save these definitions in macro files with the extension MCR for further usage.

1.3 BS2 Memory

In the ROM of a BASIC Stamp the token interpreter is saved. The program is saved as tokens in an external EEPROM located on the BS2-IC. Rarely changed information such as configuration data, for example, can be saved in this EEPROM too.

Data is stored in the BASIC Stamp's RAM. After restarting the BASIC Stamp the entire contents of RAM is cleared.

1.3.1 Program Memory

Aside from the original BS2 all other BS2 types (e, SX, p) support a program memory of eight times 2K bytes. With the newer BS2 styles you can define projects consisting of up to eight different programs. Each sub-program can be saved in one program slot of 2 K bytes.

You can save additional program modules, mathematical routines, drivers, text prompts for the user dialogue, calibration data or setup information in these additional program slots.

In this case you may utilize a Stamp directive option Listing all the files to be considered. This Stamp directive must be placed in slot #0 at first.

The examples explain a project with program modules in three different program slots.

SHIFTOUT BS2 BS2e BS2sx BS2p

`SHIFTOUT dpin#,cpin#,mode,[var{\bits},...]`

Sends synchronous serial data

It means:

dpin#	I/O pin for serial output (0-15 = I/O pin)
cpin#	I/O pin as clock output (0–15 = I/O pin)
mode	Mode of operation (0-3)
var	Variable with data to sent
bits	Number of bits. Default is eight.

Example:

```
DataP  con 0                    'Data pin to 74HC595
Clock  con 1                    'Shift clock to '595
Latch  con 2                    'Moves data from shift
                                'regiser to output latch.
counter var byte                'Counter for demo program.

again:    shiftout DataP,Clock,MSBFIRST,[counter]
          pulsout Latch,1       'Transfer to outputs
          pause 50                    'Wait 50ms
          counter = counter+1   'Increment counter.
          goto Again            'Do it again.
```

Remark:

The SHIFTOUT command sends the content of the variable `counter` to a shift register. The circuit used in this example (a 74HC595) needs the MSBFIRST mode.

SLEEP **BS2 BS2e BS2sx BS2p**

```
SLEEP seconds
```

Switches the BS2 for the given time to Sleep Mode

It means:

seconds Duration of the Sleep Mode in seconds (0...65535)

Example:

```
time     con 5  'sleep time about 5 s

         high 8
         sleep time
         low 8
         end
```

Remark:

Switches into Low-Power Mode (Sleep Mode). The I/O pins stay active. Every 2.3 s the outputs go for about 18 ms to tri-state.

	BS2	*BS2e*	*BS2sx*	*BS2p*
Current consumption during run time	8 mA	25 mA	60 mA	40 mA
Current consumption during sleep	40 µA	60 µA	60 µA	60 µA

STOP **BS2 BS2e BS2sx BS2p**

`STOP`

Stops the program

Example:

```
tone     var byte

start:   tone = 2000
         freqout 11, 1000, tone
         stop                          'stops the program
         goto start                    'never executed
```

Remark:

The program stops. In opposition to the command `end` the BS2 does not switch to the low-power mode. All outputs stay active.

STORE BS2p

`STORE Programslot#`

Specifies a program slot for READ and WRITE

It means:

Program slot# Number of the program slot (0...7)

Example:

```
'{$STAMP BS2p, store1}

lang var nib
char var nib
value    var byte
addr var word

         value = 7
         lang = 1

         lookup lang, [0,1], value
         store value
         debug ? value, cr

         addr = text2
start:   read addr, value
         if value = 0 then exit
         debug value
         addr = addr + 1
         goto start
exit:    end

Text1 DATA "Textzeile 1 in Deutsch",0
Text2 DATA "Textzeile 2 in Deutsch",0
Text3 DATA "Textzeile 3 in Deutsch",0
```

Remark:

The variable `value` controls the access to the text strings saved in different program slots of the EEPROM. This way you can implement a multi-language dialog very easy.

For example, here the German text strings are in program slot# 0 and the English text strings are in program slot#1.

The source of the program STORE1.BSP reads as:

```
Text1 DATA "Text line 1 in English",0
Text2 DATA "Text line 2 in English",0
Text3 DATA "Text line 3 in English",0
```

TOGGLE **BS2 BS2e BS2sx BS2p**

```
TOGGLE pin#
```

Switches an I/O pin to output and inverts is state

It means:

pin# I/O pin as output

Example:

```
         low 8
loop:    pause 200
         toggle 8
         goto loop
```

Remark:

The state of I/O pin 8 changes every 200 ms.

WRITE **BS2 BS2e BS2sx BS2p**

`WRITE location,variable`

Writes a byte into EEPROM

It means:

location Memory location in EEPROM (0-2047)

variable Variable with byte to save

Example:

```
addr      con 0
value     var byte

          value = $AA
          write addr, value
          value = 0
          read  addr,value
          debug hex2 value, cr

          value = $55
          write addr, value
          value = 0
          read  addr,value
          debug hex2 value
```

Remark:

The number of write cycles for EEPROM is limited. For the BS2 a minimum 10 million write cycles are allowed. The other types of BS2 allow 100,000 write cycles.

For BS2p the STORE command selects a program slot for READ and WRITE.

XOUT **BS2 BS2e BS2sx BS2p**

```
XOUT mpin#,zpin#,[house\keyorcommand{\cycles …}]
```

Sends a X-10 Powerline Control command to a PL513 or TW523 Powerline Interface Module (60 Hz)

It means:

mpin#	I/O pin for modulation control
zpin#	I/O pin for zero-cross detection
house	House Code (0-15 = “A” - “P”)
key or command	Key Number (0-15 = “1” - “16”) or command cycles

Example:

```
zpin     con  0              'Zero-crossing-detect
mpin     con  1              'Modulation-control pin
houseA   con 0               'House code: 0=A, 1=B . . .
Unit1    con 0               'Unit code: 0=1, 1=2 . . .
Unit2    con 1               'Unit code: 1=2

xout mpin,zPin,[houseA\Unit1]  'Talk to Unit1.
xout mPin,zPin,[houseA\uniton] 'Turn ON.
pause 1000                   'Wait a second.
xout mPin,zPin,[houseA\unitoff]'Turn OFF.
xout mPin,zPin,[houseA\Unit2]  'Talk to Unit2.
xout mPin,zPin,[houseA\unitoff]'Turn OFF.
xout mPin,zPin,[houseA\dim\10] 'Dim unit.
```

Remark:

```
X-10 Commands Value  || Powerline
==================== || Interface    BS2
UNITON         %10010 ||    Pin       Pin
UNITOFF        %11010 || ==================
UNITSOFF       %11100 ||     1        zPin
LIGHTSON       %10100 ||     2        GND
DIM            %11110 ||     3        GND
BRIGHT         %10110 ||     4        mPin
```

2.2 Comments about the Instruction Set

The BS2 commands explained in the last chapter have varying complexity. While the explanation of some commands need less then one page, the explanation of other commands need several pages.

Due to the compact presentation all explanations in the last chapter were short. In this chapter we will give some tips about using some of the more complex commands.

Separate chapters contain extensive explanations to specific commands or command groups like 1-Wire and LCD commands. Therefore they are not listed here.

2.2.1 SERIN and SEROUT

A big strength of all BASIC Stamps are the features of serial data exchange. Practically, each of the 16 I/O pins can be used as a serial input or output.

Questions about serial transmission rates need careful consideration. If you ignore the given possibilities you might be in for a difficult debugging session.

Tracy Allen published a very detailed analysis of the SERIN and SEROUT timing [www.emesystems.com/BS2rs232.htm]. Read this informative web site for more tips.

Note that the serial communication of all BASIC Stamps does not operate in interrupt mode. That means that only a data exchange occurs during execution of the SERIN and SEROUT commands.

This usually doesn't represent a problem for sending data.

Receiving data is more complex. The SERIN command must be executed to get data from a given I/O pin. You have to know when a transmitter will send data to the BASIC Stamp. If nothing is sent the SERIN command waits and blocks the program from further execution. Let us look behind the scene for a proper serial data exchange.

2.2.1.1 SEROUT

The SEROUT command is provided with some parameters briefly explained in chapter 2.1. Here the command is shown with its possible parameters:

```
SEROUT tpin#{\fpin#}, baudmode, {pace,} {timeout,tlabel,} [outputdata]
```

Not all parameter combinations are allowed in command SEROUT.

The simplest serial output operation you will find in many application programs makes use of the used I/O pin, the baud rate and the data to be sent.

```
'{$STAMP BS2}

TxD       con 16             'TxD at SOUT
baud      con 16468          'N9600 for BS2

loop:     serout TxD, baud, ["Hello world", cr]
          pause 1000
          goto loop
```

The short program example above send the string “Hello world“ and a CR in an endless loop. The pause of one second is for a better visualization only and has no importance for the communication itself.

If you comment the PAUSE 1000 command then it is ignored. Inspecting the I/O pin with an oscilloscope you can see short delays between the repeated SEROUT commands yet.

The data stream is not continuous because the SEROUT command must be interpreted at first before the characters can be sent. Beside this there are gaps between every single character.

The next program example shows how the parameter *pace* lengthens the gap.

```
'{$STAMP BS2}

TxD       con 16             'TxD at SOUT
baud      con 16468          'N9600 for BS2
pace      con 500            'Pace = 500 ms

loop:     serout TxD, baud, pace, ["Hello world", cr]
          pause 1000
          goto loop
```

Have a look to the Debug Window of StampW and you can see a delayed serial output of the string “Hello world”.

For the synchronization of the data exchange between receiver and transmitter the BS2 offers the possibility of flow control with an additional I/O pin for handshaking.

The next program example uses I/O pin 15 for flow control.

```
'{$STAMP BS2}

TxD       con 16             'TxD at SOUT
Fpin      con 15             'Fpin is I/O pin 15
baud      con 16468          'N9600 for BS2

loop:     serout TxD, baud, [cls]
          serout TxD\Fpin, baud, ["Hello world", cr]
          goto loop
```

Opposite of the first SEROUT command the second SEROUT command sends data only when I/O pin 15 goes high to signalize that it's "ready to receive".

The parameter *baud* of 16468 switches the BS2 to inverted data transmission mode at 9600 baud (8N1). A receiver must signalize that it is "ready to receive" with high on it's I/O pin responsible for flow control.

If the receiver does not signalize "ready for receive" \then the program example above stops with the second SEROUT command. Use timeouts to avoid such situations.

In our next program example we wait a time of 100 ms for the "Ready to receive". If no "ready to receive" is detected during this time the program branches to a timeout handler beginning with *tlabel*. In our program example the function of this handler is the output "Timeout occurred." in the Debug Window.

```
'{$STAMP BS2}

TxD       con 16         'TxD at SOUT
Fpin      con 15         'Fpin is I/O pin 15
baud      con 16468      'N9600 for BS2
tout      con 100        'Timeout = 100 ms

loop:     serout TxD\Fpin, baud, tout, tlabel, ["Hello world",cr]
          pause 1000
          goto loop

tlabel:   debug "Timeout occurred.", cr
          goto loop
```

Note that a timeout and a delayed transmission of serial data using *pace* can not be combined.

2.2.1.2 SERIN

The SERIN command too is provided with some parameters briefly explained in chapter 2.1. Here the command is shown with its possible parameters:

```
SERIN rpin#{\fpin#}, baudmode, {plabel,} {timeout,tlabel,} [inputdata]
```

We start our short program examples with the simplest input – receiving one single character.

```
'{$STAMP BS2}

RxD       con 16             'RxD at SIN
baud      con 16468          'N9600 for BS2

char var byte

loop:     serin RxD, baud, [char]
          debug cr, "D: ", char
          goto loop
```

The byte variable `char` saves the character received. To simplify matters we use serial input I/O pin 16. We have to note that in this case all received characters will be sent back to the transmitter as an echo.

To distinguish between output from DEBUG and SEROUT we placed ahead the characters “D:” in the DEBUG command.

Sending characters from the PC keyboard to the BASIC Stamp is no problem due to the delays in typing. If the characters come with the specified baud rate (without a gap between the characters) from a measuring device, GPS or something similar then characters can be lost and the communication can hang.

To synchronize the receiver with a transmitter we use flow control. In the next program example we use I/O pin 15 for flow control. The rest of the program is unchanged.

```
'{$STAMP BS2}

RxD       con 16             'RxD at SIN
Fpin      con 15             'Fpin is I/O pin 15
baud      con 16468          'N9600 for BS2

char var byte
```

```
loop:     serin RxD\Fpin, baud, [char]
          debug cr, "D: ", char
          goto loop
```

The parameter *baud* of 16468 switches the BS2 into inverted transmission mode with 9600 baud (8N1) again. In this case the BS2 signalizes it's "ready to receive" by showing a high signal on the flow control pin. If the SERIN command is finished then the flow control pin goes low to avoid sending more characters. If the expected data isn't received the program would stop at the SERIN command. Here a timeout would help avoid this situation.

The timeout in the next program example can be used independent of flow control. If there is no data in the `tout` time then the program detects a timeout and branches to the timeout handler `NoData`.

```
'{$STAMP BS2}

RxD       con 16         'RxD at SIN
Fpin      con 15         'Fpin is I/O pin 15
baud      con 16468      'N9600 for BS2
tout      con 1000       'Timeout = 1000 ms

char var byte

loop:     serin RxD\Fpin, baud, tout, NoData, [char]
          debug cr, "D: ", char
          goto loop

NoData:   debug cr, "No Data received - Timeout."
          pause 1000
          goto loop
```

Avoid the values tout = 0 and tout = 1. In these cases the BS2 has a timeout.

For more security of data exchange you can use the parity check (7E1). As shown in the next program example, a parity check works independent of a timeout.

Because the DEBUG command only works with 9600 Baud 8N1 we have to use other I/O pins for serial communication at 9600 Baud 7E1.

```
'{$STAMP BS2}

RxD       con 1              'RxD is I/O pin 1
Fpin      con 0              'Fpin is I/O pin 0
baud      con 24660          'N9600-7E1 for BS2
tout      con 1000           'Timeout = 1000 ms
```

```
char var byte

loop:     serin RxD\Fpin, baud, BadData, tout, NoData, [char]
          debug cr, "D: ", char
          goto loop

NoData:   debug cr, "No Data received - Timeout."
          pause 1000
          goto loop

BadData:
          debug cr, "Bad Data - Parity Error."
          pause 1000
          goto loop
```

Receiving single characters is quite rare. Normally we will receive strings and evaluate them in the a PBASIC program. With the many formatters the BS2 gives very good support for these tasks.

The BS2 can use a string as an argument in the SERIN command. We must first declare this string as a byte. After receiving this byte an array is filled with the received characters.

```
'{$STAMP BS2}

RxD       con 16              'RxD at SIN
baud      con 16468           'N9600 for BS2
tout      con 1000            'Timeout = 1000 ms

SerString var byte(11)        'Make a 11-byte array
SerString(11) = 0             'Character 0 is the String Terminator

loop:     serin RxD, baud, [str SerString\10]
          debug str SerString ' Display the string.
          goto loop
```

In our program example we declared an array of 11 bytes. The last bytes were set to 0 during initialization. The SERIN command receives 10 characters and saves them to the byte array.

On the 11th character we encounter a 0-terminated string in the byte array. The DEBUG command expects a 0-terminated string.

A 0 string terminator is often used in strings with a variable length. If you wait for strings with a variable length as input data then SERIN can stop receiving after a defined number of characters or when a special character was detected.

In the next program example a 0-terminated string is expected. Receiving stops after 10 characters or a 0. Use the DEBUG command to visualize this example.

```
'{$STAMP BS2}

RxD       con 16              'RxD at SIN
baud      con 16468           'N9600 for BS2
tout      con 1000            'Timeout = 1000 ms

SerString var byte(11)        'Make a 11-byte array
SerString(11) = 0             'Character 0 is the String Terminator

loop:     serin RxD, baud, [str SerString\10\"0"]
          debug str SerString ' Display the string.
          goto loop
```

The serial data format of a multimeter could look like this:

VDC = 12345 mV<CR> FRQ = 12345 Hz<CR>

If you need the result of the DC voltage measuring only then you can filter these value very simple. The following program example shows the usage of the formatters *wait* and *skip*.

```
'{$STAMP BS2}

RxD       con 16              'RxD at SIN
baud      con 16468           'N9600 for BS2

value     var word

loop:     serin RxD, baud, [wait ("VDC ="), skip 2, dec value]
          debug "Voltage = ", dec value
          goto loop
```

The parameter *wait ("VDC =")* checks the input data stream for the characters “VDC =”. *Skip 2* passes the next two characters – spaces in this example. The characters before CR are changed to a decimal number.

Use the formatter *waitstr* to compare a saved string with the received data. In the next program example a string consisting of four characters is sampled for comparison.

This pattern is compared with the next set of characters received. If the compare finds the strings the same fits then SERIN is finished. The loop repeats after the message “Matching Pattern found.”

```
'{$STAMP BS2}

RxD       con 16             'RxD at SIN
baud      con 16468          'N9600 for BS2

SerString var byte(5)        ' Make a 5-byte array.
SerString(5) = 0             ' Put 0 in last byte.

loop:     serin RxD, baud, [str SerString\4]    'Get the string
          debug "Waiting for: ", str SerString, cr
          serin RxD, baud, [waitstr SerString]  'Wait for a match
          debug "Matching Pattern found.", CR
          goto loop
```

The quickest way to receive data is given in the format `serin RxD, baud, [str SerString\L]`. With this simple syntax the BS2 can receive data with 9600 Baud without gaps. If there are further arguments as *wait* or *skip* then the interpretation needs more resources and the BS2 could loose characters.

For the BS2p there is an enhanced possibility for a temporary saving of the received data using scratch pad RAM.

```
'{$STAMP BS2p}

RxD       con 16             'RxD via SIN
baud      con 16624          'N9600 for BS2p
N         con 16             'Buffer length

char      var byte(N)        'Byte array
addr      var byte           'Address pointer

loop:     serin RxD, baud, [wait("ABC"), spstr N]
          for addr = 0 to 15
               get addr,char(addr)
          next
          debug str char\16, cr
          goto loop
```

The SERIN command saves the characters after recognizing the “ABC”. Then, the next 16 characters are saved to the BS2p’s scratch pad RAM.

To display the received characters with DEBUG the string is first saved in a byte-array and then displayed in the Debug Window.

This way you can save complex strings up to 126 bytes into the scratch pad RAM for further processing or parsing of the individual characters.

You will find such an example in chapter 6.3 for processing GPS data.

2.2.2 RUN

2.2.2.1 Several programs in the controller

Sometimes electronic devices support different functions activated by a query of a switch after power-on. Most of us know these configuration switches as DIP devices mounted on a printed circuit board accessible to the user.

The enhanced EEPROM of the BS2x can contain different applications in different program slots. A DIP switch is connected to the BS2x and after the program starts this switch is queried. Based on this query the program branches to a different program slot.

In the next program example the keys of the BASIC Stamp Activity Board will replace the configuration switch idea by serving as a program selector. The basics of this branching are explained in the next program example.

```
switch   var nib
switch = ins.nib.2 & $03
'debug bin2 switch, cr

BRANCH (switch), [PRG0, PRG1, PRG2, PRG3]
PRG1:    RUN 1  'redirect to page 1
PRG2:    RUN 2  'redirect to page 2
PRG3:    RUN 3  'redirect to page 3
PRG0:           'stay on page 0
...
end
```

The command `switch = ins.NIB2 & $03` queries the I/O pins 11-8 (keys on the BASIC Stamp Activity Board) and masks two bits so only I/O pin 8 (blue key) and I/O pin 9 (black key) can be active.

You can test the branching with the follow programs on the BASIC Stamp Activity Board. To show the links to a project the appropriate Windows Editor screenshots are included.

We have explained the details in chapter 1.3.1. Figure 18 shows the main program in program slot #0. Figure 19 shows one of the executable applications in program slot#1. Here the whole application consists of one DEBUG command only. But this does not matter.

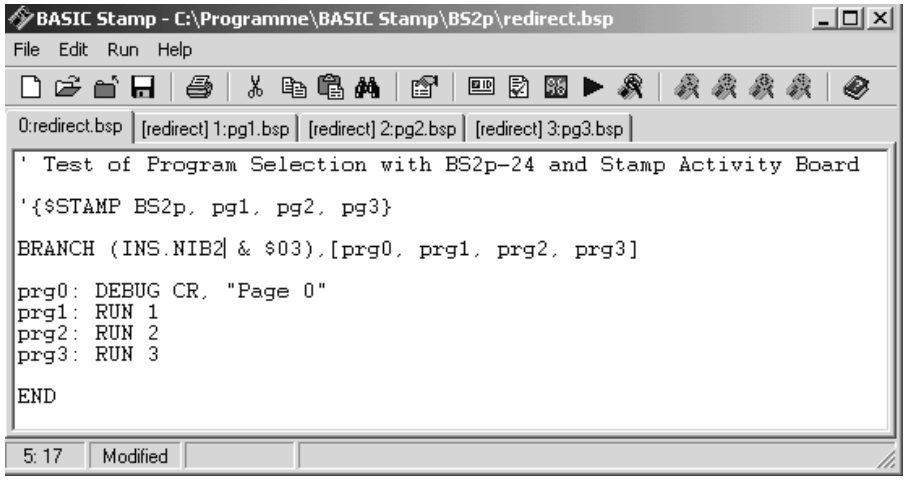


Figure 18 Program REDIRECT.BSP Program Slot #0

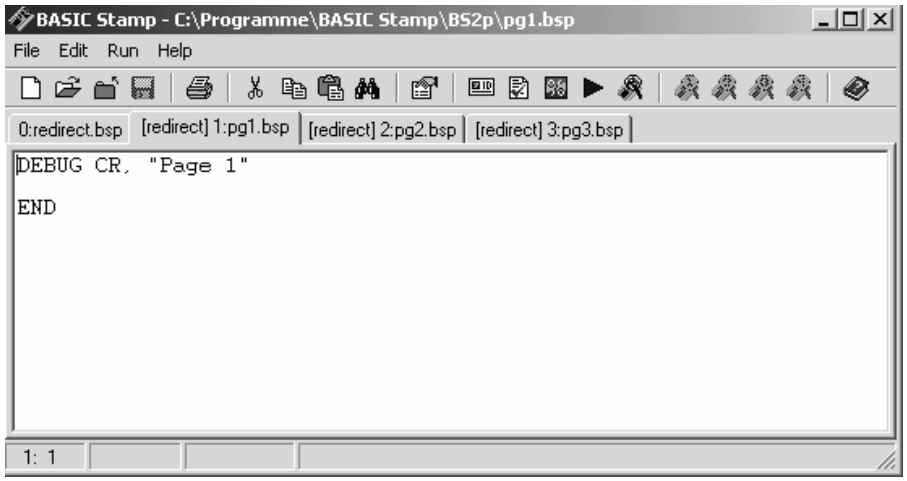


Figure 19 Program REDIRECT.BSP Program Slot #1 (PG1.BSP)

2.2.2.2 Subroutines in different program slots

Some programs are enhanced by using several program slots for program fragments or subroutines.

The goal is to ensure that after processing of such a subroutine that the program jumps to the place it was after the subroutine call back.

Figure 20 shows with dotted lines how to switch between the program slots for the common subroutine technique. The dotted lines show what the RUN command is able to carry out.

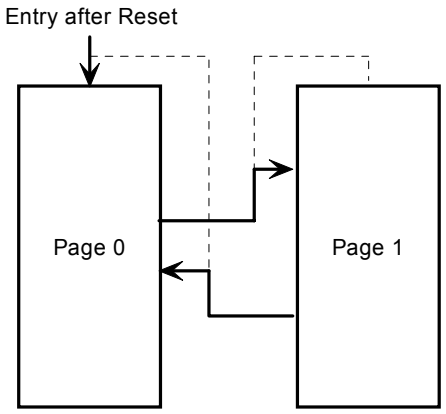


Figure 20 Switch between program slots

A small trick helps us here. At the beginning of each program slot we'll place a branch table. An example shows the principle.

Listing 1 shows the main program in program slot #0. This program slot is active after Reset. Due to the default initialization of the variable `reentry` in slot #0 the main program does not branch and begins at the label `start` to proceed.

As Listing 2 shows, the called subroutines `add` and `mult` are placed in program slot #1.

```
'{$STAMP BS2p, slot1}

'The main program calls subroutines (tasks) in another
'program slot. The point for reentry must be defined
'before the subroutine call.

reentry var nib 'reentry points are _1 and _2
task var nib

x         var byte
y         var byte
z         var word

          branch reentry, [start, _1, _2]

start:    x = 170
          y = 85

          'access task 0 on program slot 1
          reentry = 1 : task = 0 : run 1
_1:       debug dec5 z, cr

          'access task 1 on program slot 1
          reentry = 2 : task = 1 : run 1
_2:       debug dec5 z, cr
          end
```

Listing 1 Main Program in Program Slot #0

```
reentry   var nib       'used by page0
task      var nib       'used by page0

x         var byte
y         var byte
z         var word

          branch task, [add, mult]

add: z = x + y          'operate task 0
          run 0         'reentry in page0

mult:      z = x * y    'operate task 1
          run 0         'reentry in page0
```

Listing 2 Subroutines in Program Slot #1

Use some precaution when calling these subroutines after the initialization of the variables x and y.

The subroutine `add` in program slot #1 is declared as task 0 in the `branch` command, while the subroutine `mult` is declared as task 1. Accordingly the variable `task` must be set in program slot #0 before the call of the respective subroutine.

In the same way we have to organize the return with the help of the variable `reentry`. The line following the subroutine call must be marked with a Reentry Label. After returning to program slot #0 the `branch` command branches to the referred label.

To avoid access conflicts on the variables, both program slots begin with the variable declarations. This ensures the declaration of the variables is global.

2.2.3 Switching the I/O Blocks with the BS2p

In the next program (Listing 3) you can test the functionality of the I/O Block Switching using the BASIC Stamp Activity Board and a BS2p-24.

```
' I/O Test for BASIC Stamp Activity Board with BS2p-24

'{$STAMP BS2p}

         low 11        ' Red key LED on
         gosub brk     ' To leave breakpoint press the blue key
         high 11       ' Red key LED off
         ' auxio
         low 10        ' Black key LED on
         gosub brk     ' To leave breakpoint press the blue key
         high 10       ' Black key LED off
         mainio
         end

brk:     debug cr, "Press the blue key to continue..."
brk1:    if in8=1 then brk1
         pause 100
brk2:    if in8=0 then brk2
         return
```

Listing 3 Switching the I/O Blocks

In this source code example the command `auxio` is commented by the leading character ' and is without any effect.

After the program starts the LED connected to I/O pin 11 (located above the red key on the BASIC Stamp Activity Board) is switched on. Next the first breakpoint is reached.

Leave this breakpoint by a making I/O pin 8 low (or by pressing the blue key on the Stamp Activity Boards).

In the next step the LED connected to I/O pin 11 switches off and the LED connected to I/O pin 10 switches on. Handle the next breakpoint as before and the LED connected to I/O pin 10 switches off, too.

Now activate the I/O Block Switch by removing the comment character ' before `auxio` and download again.

You will see that the LED connected to I/O pin 10 does not work. The I/O operation points to the AUXIO block not available in a BS2p-24. Therefore this I/O operation has no effect in a BS2p-24.

2.2.4 Interrupting by Polling the BS2p

The BASIC Stamp 2 has no hardware interrupt mechanism. For reactions to external events this is sometimes a disadvantage.

Sure, there was the possibility to check any I/O pin periodically for its logical condition, but a short response to a certain logic state makes this option difficult.

The BS2p has a polling mechanism for time-critical applications. This polling mechanism eliminates the disadvantage mentioned above associated with sporadically check I/O states. But, do not confuse this polling with real interrupts. Table 4 lists an overview for the Polling Commands.

Polling Command	*Effect*
POLLIN Pin, State	Configuration of checked inputs
POLLOUT Pin, State	Defines the output activity after a polled-input event occurred
POLLMODE *Mode*	Defines the Polling Mode
POLLRUN *Slot*	Specifies a program running after a polled-input event occurred
POLLWAIT *Period*	Switches the BS2p for a certain time into sleep state and polls the polling inputs afterwards

Table 4 Overview of BS2p Polling Commands

We'll describe this new polling mechanism next with small and clear program examples in significant detail.

Use the POLLIN command to declare the I/O pin to poll and define the state of the polled-input event. The output activity depends on the commands POLLOUT, POLLMODE and POLLRUN and is triggered when the polled-input event occurred.

Several I/O pins can be declared for polling. The polled-input event occurs if one of the polling conditions comes true.

The BS2p firmware checks the I/O pins defined by the command POLLIN at the end of each instruction and before reading the next PBASIC instruction – practically as a background task in that it is done between lines of code. This mechanism allows a significantly faster response to a polled-input event than a programmed query of I/O pins by PBASIC instructions.

The POLLOUT command declares an I/O pin and it's state after a polled-input event occurs.

The POLLMODE command defines the polling mode according to Table 5.

Mode	*Effect*
0	Polling disabled, Clears the POLLIN and POLLOUT configuration
1	Polling disabled, Saves the POLLIN and POLLOUT configuration
2	Polling enabled with POLLOUT and POLLWAIT function
3	Polling enabled only with POLLRUN function
4	Polling enabled with all Polling functions
5	Clears the POLLIN configuration
6	Clears the POLLOUT configuration
7	Clears the POLLIN and the POLLOUT configuration
8-15	Identically to 0-7, but POLLOUT states are saved

Table 5 POLLMODE Parameter

With this information we can have a look to the first program example.

```
'{$STAMP BS2p}

InitPolling:
```

```
     POLLIN 8,0          ' If this polled input is Lo
     POLLIN 9,0          ' or this input is Lo
     POLLOUT 11,0        ' then the polled output is Lo
     POLLMODE  2         ' Pollout only

Loop:                    ' This in the "main" program
     DEBUG "."
     goto LOOP
```

I/O pins 8 and 9 will be polled. The polled-input event occurs when one or both polled inputs are low.

Using the BASIC Stamp Activity Board this means pressing the blue or the black key.

When the polled-input event occurs I/O pin 11 switches low and the LED switches on.

The defined polling mode allows only the `POLLOUT` and the `POLLWAIT` function. `POLLWAIT` is not used here.

After initializing the polling the program runs an endless loop and displays it's activity with the `DEBUG` command.

In our program example the output defined by `POLLOUT 11, 0` reacts to the polled-input event directly.

As long as one polling input detects low the polling output is also low, the polling output switches to high immediately. The Latch Option of the poll mode can freeze this state. This means after the polled-input event the polling output switches to low as before, but stays low independent what happens on the polling inputs.

In the next program example the poll mode was changed from 2 to 10.

```
'{$STAMP BS2p}

InitPolling:
     POLLIN 8,0     ' If this polled input is Lo
     POLLIN 9,0     ' or this input is Lo
     POLLOUT 11,0   ' then the polled output is Lo
     POLLMODE 10

Loop:               ' This in the "main" program
     DEBUG "."
     goto LOOP
```

With this latch function we have the ability to monitor an I/O pin and to react to the polled-input event later. The polled-input event can be processed after a continuous routine.

Again, we modify our program example.

```
'{$STAMP BS2p}

i var NIB

InitPolling:
     POLLIN 8,0    ' If this polled input is Lo
     POLLIN 9,0    ' or this input is Lo
     POLLOUT 11,0  ' then the polled output is Lo
     POLLMODE 10   ' Enable latched POLLOUT only

Loop:              ' This in the "main" program
     ' Print always 10 dots without any interruption
     for i= 0 to 9
          DEBUG "." : pause 100
     next
     DEBUG CR
     ' on a latched polling event goto the handler
     IF IN11=0 THEN Event
     goto LOOP

Event:                 ' handler for latched polling event
     DEBUG CR, "Latched Polling detected.", CR
     POLLMODE 10       ' restore of polling initialization
     goto LOOP
```

In the main loop `DEBUG` outputs a package of 10 dots before querying I/O pin 11. If this I/O pin were set to low from a polled-input event occurring during `DEBUG` outputs then the program branches to the label `Event` and outputs “Latched Polling detected.” The following `POLLMODE 10` command re-initializes the polling and I/O pin 11 switches to high again.

The command `POLLRUN` specifies a program that runs after an polled-input event occurred. This program can be saved in any program slot. The slot number is parameter of `POLLRUN`.

We add the command `POLLRUN 1` to the last program example and change the poll mode to 4 to enable all polling activities.

```
'{$STAMP BS2p, pollrun}

InitPolling:
     POLLIN 8,0    ' If this polled input is Lo
     POLLIN 9,0    ' or this input is Lo
     POLLOUT 11,0  ' then the polled output is Lo
     POLLRUN 1     ' and program in Slot #1 runs.
     POLLMODE 4    ' All polling functions enabled

Loop:              ' This in the "main" program
```

```
    DEBUG "."
    goto LOOP
```

As defined, the program running after the polled-input event occurred must be placed in program slot #1.

```
' POLLRUN Activity

loop:
     debug "+"
     goto loop
```

After the program downloads it starts by initializing the polling and running in an endless loop. In this endless loop the DEBUG command outputs one dot after another.

When the polled-input event occurred I/O pin 11 switched to low and the program in program slot #1 begins. You can watch the Pollrun activity by a repeated output of the character "+" by the DEBUG command.

Finally, let's have a look to the POLLWAIT command. POLLWAIT queries (a minimum of) one polling input periodically while the BS2p switches between these queries to its power-down mode with about 60 µA current consumption. The periodic query corresponds to the NAP command and its parameters are listed in Table 6.

Period	*Polling Cycle*
0	10 ms
1	36 ms
2	72 ms
3	144 ms
4	288 ms
5	576 ms
6	1,152 s
7	2,304 s
8	No Power-Down (< 160 µs)

Table 6 POLLWAIT Parameter

Let's modify our program example for some additional experimentation.

```
    char = Line2                        ' address second line
    gosub LCDcmd
    for index = 16 to 31
      read index, char                  ' get char from EEPROM
      gosub WrLCD                       ' write it
    next

    char = Line3                        ' address third line
    gosub LCDcmd
    for index = 32 to 47
      read index, char                  ' get char from EEPROM
      gosub WrLCD                       ' write it
    next

    char = Line4                        ' address forth line
    gosub LCDcmd
    for index = 48 to 63
      read index, char                  ' get char from EEPROM
      gosub WrLCD                       ' write it
    next
    pause 1000                          ' wait 2 seconds

    char = ClrLCD                       ' clear the LCD
    gosub LCDcmd

    pause 500

    goto Start                          ' do it all over

' -----[ Subroutines ]----------------------------------------
'
' Send command to the LCD
'
' Load char with command value, then call
'
'   Clear the LCD.............. $01, %00000001
'   Home the cursor............ $02, %00000010
'   Display control............ (see below)
'   Entry mode................. (see below)
'   Cursor left................ $10, %00010000
'   Cursor right............... $14, %00010100
'   Scroll display left........ $18, %00011000
'   Scroll display right...... $1C, %00011100
'   Set CG RAM address........      %01aaaaaa (Character Generator)
'   Set DD RAM address........      %1aaaaaaa (Display Data)
'
' Display control byte:
'
'   % 0 0 0 0 1 D C B
'               | | -- blink character ander cursor (1=blink)
'               | ---- cursor on/off (1=on)
'               ------ display on/off (1=on)
'
' Entry mode byte:
'
'   % 0 0 0 0 0 1 X S
'                 | --- shift display (S=1), left (X=1), right (X=0)
'                 ---- cursor move: right (X=1), left (X=0)
'
```

```
LCDcmd:
     low RS
     gosub WrLCD
     high RS
     return

' Write ASCII char to LCD
'
WrLCD:
     outl = outl & %00001000            ' RS = 1, data bus clear
     outp = char & %11110000            ' mask the high nibble
     outl = outl | outp                 ' output the nibble
     PULSOUT E, 10                      ' strobe the Enable line
     outl = outl & %00001000
     outp = char & %00001111            ' get low nibble
     outp = outp << 4
     outl = outl | outp
     PULSOUT E, 10
     return
```

Listing 14 Text Output to an LCD (LCD.BS2)

3.3.3 Serial Control of an LCD Module

If the minimum six I/O lines are not available for LCD control you can use a serially controlled LCD module.

Scott Edwards Electronics [www.seetron.com] offers a Serial Backpack and completely equipped LCD modules. Figure 54 shows a printed circuit board able to connect different LCDs. To connect a BASIC Stamp there is a serial data line and +5 V and GND.

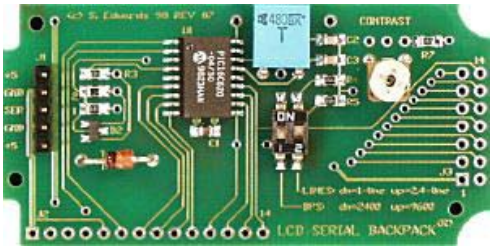


Figure 54 LCD Serial Backpack

The Serial Backpack converts the serially received data into parallel transmitted LCD commands.

The RS line differs between commands and data. On the serial side a command is marked by a leading code &FE. If you want to send the command Clear Display to the LCD the Serial Backpack must receive the sequence <&FE> <&01> first.

See Listing 15 for the details of serial LCD control. There are no special features.

```
' -----[ Title ]-----------------------------------------------
'
' File...... SERIALLCD1.BS2
' Purpose... Stamp -> Serial LCD
' Author.... Claus Kuhnel
' Started... 01 Sept 2001
' Updated...

' -----[ Program Description ]--------------------------------
'
' This program demonstrates the various standard features of an LCD
' display that uses the Hitatchi HD44780 controller.
' The LCD used to test this program was 20x4 Serial LCD from Seetron.
'
' -----[ Revision Hisory ]-----------------------------------
'
' 09-01-01 : Version 1.0
'
' -----[ Directives ]-----------------------------------------
'
'{$STAMP BS2}               'specifies a BS2
'
' -----[ Constants ]------------------------------------------
'
TxD       con  1             ' Serial Data to LCD
N9600     con $4054          ' Baudmode-9600 bps inverted
N2400     con $418c          ' Baudmode-2400 bps inverted
I         con $FE            ' Instruction prefix value

' LCD control characters
'
ClrLCD    con $01            ' clear the LCD
CrsrHm    con $02            ' move cursor to home position
CrsrLf    con $10            ' move cursor left
CrsrRt    con $14            ' move cursor right
DispLf    con $18            ' shift displayed chars left
DispRt    con  $1C           ' shift displayed chars right

Line1     con  $80           ' addr line #1 ¦ 80H
Line2     con $C0            ' addr line #2 ¦ 80H
Line3     con $94            ' addr line #3 ¦ 80H
Line4     con $D4            ' addr line #4 ¦ 80H

' -----[ Variables ]-----------------------------------------
'
char    var   byte                    ' char sent to LCD
index   var   byte                    ' loop counter

' -----[ Initialization ]------------------------------------
'
data "THE BASIC STAMP!"               ' Preload EEPROM
```

```
data "A PIC16C57 knows"                 ' Preload EEPROM
data "P B A S I C from"                 ' Preload EEPROM
data "Parallax Inc."                    ' Preload EEPROM

' Initialize the Serial LCD (HD44780 controller & Serial Backpack)
'
LCDini:
     ' Now clear the screen in case there's text left from a previous
     ' run of the program. Note that there's a 1-second pause prior to
     ' sending any data to the Backpack. This gives the Backpack plenty
     ' of time to initialize the LCD after power up.
     low TxD                            ' Make the serial output low
     pause 1000                         ' Let the LCD wake-up

' -----[ Main Code ]----------------------------------------
'
start:
     serout TxD,n2400,[I,ClrLCD]        ' Clear the LCD screen
     serout TxD,n2400,[I,Line1+2]

     for index = 0 to 15
       read index, char                 ' Get char from EEPROM
       serout TxD,n2400,[char]          ' and print it.
     next

     serout TxD,n2400,[I,Line2+2]       ' Move to line 2
     for index = 16 to 31
       read index, char                 ' Get char from EEPROM
       serout TxD,n2400,[char]          ' and print it.
     next

     serout TxD,n2400,[I,Line3+2]       ' Move to line 3
     for index = 32 to 47
       read index, char                 ' Get char from EEPROM
       serout TxD,n2400,[char]          ' and print it.
     next

     serout TxD,n2400,[I,Line4+3]       ' Move to line 4
     for index = 48 to 60
       read index, char                 ' Get char from EEPROM
       serout TxD,n2400,[char]          ' and print it.
     next
     pause 2000                         ' Wait 2 seconds

     goto Start                         ' Do it all over
```

Listing 15 Serial LCD Control (SERIALLCD1.BS2)

3.4 Interface to the PC Keyboard

When interfacing to unique circuits we often look into using Al Williams' different products [www.al-williams.com/awce]. Al Williams designed several PAK Co-Processors. Table 12 shows the actual products:

Type	*Application*
PAK-I	Mathematic Co-Processor
PAK-II	Enhanced Mathematic Co-Processor
PAK-III	Port-Expander 8 I/O Lines
PAK-IV	Port-Expander 16 I/O Lines
PAK-V	PWM Co-Processor
PAK-VI	Keyboard Co-Processor
PAK-VII	Pulse-In Co-Processor
PAK-VIII	Pulse-Out Co-Processor
PAK-IX	Combines PAK-II with 8 digital I/O Lines and 5-Channel 10-Bit Analog-to-Digital Converter

Table 12 PAK-Co-Processors of AWC

We'll use the PAK-VI in this application example to connect a PC keyboard to a BS2.

Keyboards are tossed aside by computer users so they are offered cheap by second hand shops or even for free in the corner of your office. They are able to produced alphanumeric messages and control codes. It is useful to connect these keyboards to BASIC Stamps.

PAK-VI is a PICmicro specially programmed for that purpose. The keyboard signals are changed to serial data for working with the BASIC Stamp. The difficulties of trying to directly interface with the keyboard are avoided. This co-processor is also available for interfacing to a serial mouse instead of a keyboard, but we aren't using it in this application.

Let's take a short look at the bi-directional signals of a keyboard. It's normal use is connected to a PC with its cable. See to the books describing the PC connection in detail if you're interested in more details.

The synchronous serial data transfer is controlled by a clock signal from PC. In our case the PAK-VI generates this clock instead of the PC. Synchronous to the clock the data is transmitted bit by bit. The eight bits of a byte are enhanced with a frame of pulses. So we essentially get eleven bits for one byte.

This data transmission is hidden to the user with a PAK-VI. The BASIC Stamp is free of the trouble of handling the keyboard data transmission protocol on it's own.

The BASIC Stamp is connected to the PAK-VI with two-way RS232 in an asynchronous connection. With the SERIN and SEROUT commands the BASIC Stamp has control over the keyboard. In the following program example an additional line (I/O pin 15 of the BASIC Stamp) is used to reset the PAK-VI.

Sending commands to the keyboard and Pak-VI uses special control words. For both we'll use standard presettings. For PAK-VI these presettings come with the humorous name "Cook-Mode".

The keyboard is basically fixed to scan code 3. In this mode each key generates a so-called "make-code". Each key is marked with the number of its position. The task for the connected device is to align a specific character or control to this numbered event. Once a key is pressed there is no repeat function in this mode.

Figure 55 shows the connection diagram to the PS2 keyboard.

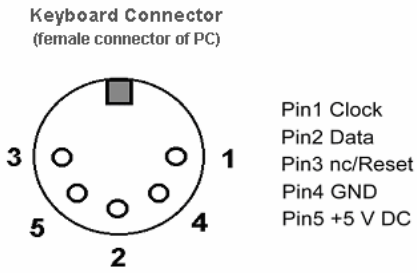


Figure 55 Connection Diagram to a PS/2 Keyboard

Table 13 shows the pin declarations of the PAK-VI co-processor.

Pin	*Name*	*I/O Type*	*Description*
1	RX	Input	TTL-level RS232 input
2	TX	Output	TTL-level RS232 output
17	Enable	Input	If this pin is not connected or high, the PAK transmits code it receives from the keyboard; the PAK always responds to commands.
18	Enable2	Input	If this pin is low, the PAK transmits code it receives from the keyboard; the PAK always responds to commands.
13	DAvil	Output	High when data is available.
4	RESET	Input	Hardware resets the PAK when low. Must be high for normal operation
3, 5	VSS	Power	GND (ground both pins)
14	VDD	Power	+ 5 V DC
15	RES1	Clock	connects to resonator
16	RES2	Clock	connects to resonator
11	DATA	I/O	Keyboard data line
12	CLOCK	I/O	Keyboard clock line
6…10	N/C	N/C	Not used

Table 13 Pin connections PAK-VI

Figure 56 shows the connection of a PC keyboard via keyboard co-processor PAK-VI to a BS2.

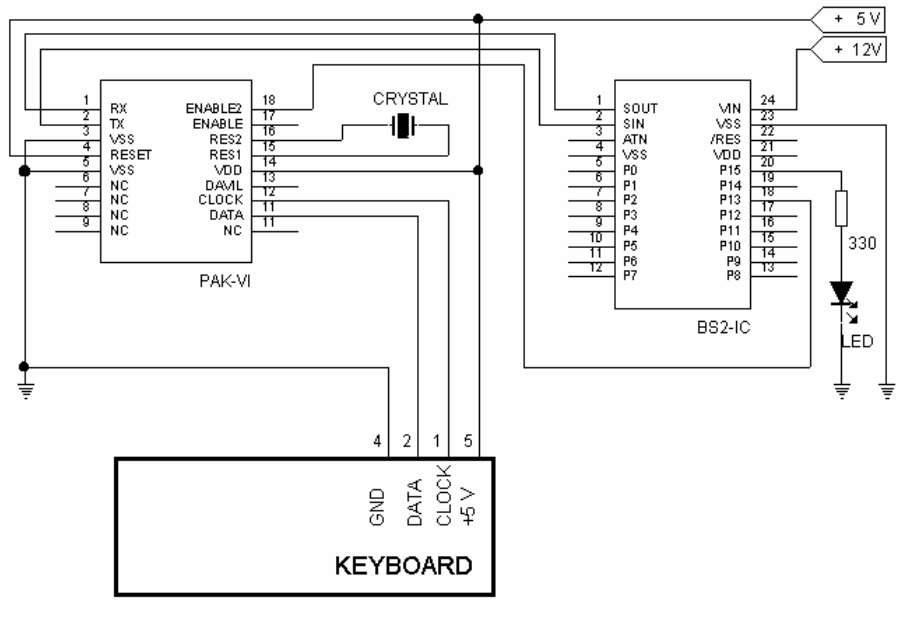


Figure 56 PC Keyboard with PAK-VI to BS2

One notable fact is that when interfacing via PAK-VI the keyboard appears to the BS2 as a serial asynchronous RS232 coupled device. Because we have no data transmission from BS2 to the keyboard in our example you will find no SEROUT command in the example program.

The connection from the pin SOUT of the BS2 to the PAK-VI input RX can be cancelled. This connection is shown here to prepare the reader for possible modifications.

As a simple example we control BS2 outputs with signals from the PC keyboard with an LED connected to I/O pin15. The flashing period of this LED is controlled by a keyed cipher. The keyed cipher is shown on the Debug Window.

Listing 16 shows the program used for the BS2.

```
' -----[ Title ]-----------------------------------------------
'
' File......  PAK_NUM2.BS2
' Purpose...  Using PAK VI for recognizing codes from
'             keyboard to control the period of a flashing LED
' Author....  Klaus Zahnert
' Started...  06/16/01
' Updated...
'
' -----[ Program Description ]--------------------------------
'
' BASIC Stamp 2 is connected to PC's keyboard by using the PAK VI
' keyboard-controller to make input for codes 0...9. Timing-value
' for blinking the LED is aligned to key pressed codes with schedules.
' this program runs for demonstration PAK VI with polling from BS2.
'
' -----[ Revision Hisory ]-------------------------------
'
'
' -----[ Constants ]-------------------------------------------
'
LED       con 15             'pin to drive LED
'baudval  con 16624          'BS2SX/9600/8/n/1 direct con.
baudval   con 16468          'BS2  /9600/8/n/1 direct con.
tout      con 100            'wait for SERIN-response (ms)
datinpin  con 12
fpin      con 13
'
' -----[ Variables ]-------------------------------------------
'
border    var word           'generated with precalc.schedule
n         var word           'Loopindex
datinbyt  var byte
datoldbyt var byte
code var byte
'
' -----[ Initialization ]--------------------------------------
'
'{$STAMP BS2}

high LED
low  fpin                    'PAK 6 enable 2
border    =   10             'Startvalue

' -----[ Main Code ]-------------------------------------------
'
start:
     For n = 1 to border        'toggle LED with period(keypr)
     serin datinpin,baudval,tout,goon,[datinbyt]
goon:     If datinbyt <> datoldbyt then calc 'need new period?
     next

     toggle LED
     goto start

calc:
     lookdown datinbyt,[6,103,51,102,25,101,50,100,12,99],code
    'makes code :    0,  1, 2,  3, 4,  5, 6,  7, 8  9

     lookup   code  ,[1,2,4,7,12,22,42,75,135,255],border
```

```
'makes value for bord1 from schedules position, given by code
debug "Ziffer = ", dec code, TAB,TAB,"Zeitkonst.= ",dec border,cr

 'store datinbyt in datoldbyt to compare again
datoldbyt = datinbyt
goto start
```

Listing 16 Keyboard input using PAK-VI (PAK_NUM2.BS2)

In this program example the restriction is that we're only using ciphers as a reference. By enhancing the scan code you can receive more inputs from the keyboard. The limit is only determined by the memory space of the BASIC Stamp.

The program permanently polls the state of the PAK-VI output register. If any key was pressed the output value changes. Recognizing that, the polling loop is left and the new output is used for identifying the pressed key.

In the program example the main code of the codes (from keyboard documentation) is stored in a lookdown table. The returned value of the LOOKDOWN command is the codes value. Comparing this value with one in the table is the same task as the PC's keyboard controller does but with more elegance for the whole set of characters.

The DEBUG command sends the value of recognized key to the Debug Window.

In this simple example it is not practical the flashing period of a LED it is not practical to set the flashing time immediately from the keyed values 0…9. For more visible effects we use a LOOKUP table to align blinking values in steady increments to the variable *Border*. This value is used as an index for a counting loop.

After finishing the counting by ending the loop, the LED output is toggled.

3.5 Port Enhancement with Shift Registers

With 16 I/O pins the BASIC Stamps (BS2, BS2sx, BS2e and BS2p-24) are armed with almost enough resources to connect different peripheral components. With BS2p-40 the number of I/O pins is doubled to 32. But, for enhancing the number of I/O pins often there is an inexpensive solution with separate integrated circuits.

Output registers store values from serial input with a few I/O lines. The bits are shifted from one register to the next while input occurs. After that, the message is available on parallel outputs.

This kind of port enhancement is useful in cases where the loss of speed in data transmission can be tolerated by the connected peripheral components. This is often the reality with

common examples using keyboards and displays. It is often possible to design mechanical control systems with such hardware.

There are different types of serial protocols. With one line at a minimum you can have functional RS232 communication.

Synchronous serial protocols need a clock line in addition to the data line. A sample was already given in Chapter 3.1.1 with the I²C coupled port enhancement using the PCF8574A. This example developed a parallel printer control with only two I/O lines.

The cheapest solution is to use shift registers from the standard line of TTL circuits.

In the following example the D-Flip-Flop 74HCT174 is used. The 74HCT174 contains six single D-Flip-Flops with a common clock line. Each of the six Flip-Flops has an input and two outputs with inverse levels. A D-Flip- Flop moves the state of the input line to outputs at the positive edge of the clock line.

To build a shift register, the flip-flops must be cascaded in chain. For a 74HCT174 the data output Q(n) must be connected to the data input D(n+1) by external wiring. With the positive edge of the clock the state of the former Flip-Flop is moved to the following Flip-Flop. Using one 74HCT174 we gain six digital outputs for two I/O pins of the BASIC Stamp.

There is no strict advantage with this small number of wired outputs. But in the following example it is easy to recognize how it works as we demonstrate using a chain of 74HCT174s. By cascading the 74HCT174s we get n*6 outputs by using only 2 BASIC Stamp I/O pins.

Figure 57 shows the three 74HCT174 cascaded to get a 18-Bit shift register. The transition from one Flop-Flop to the next is drawn by arrows in a symbolic manner.

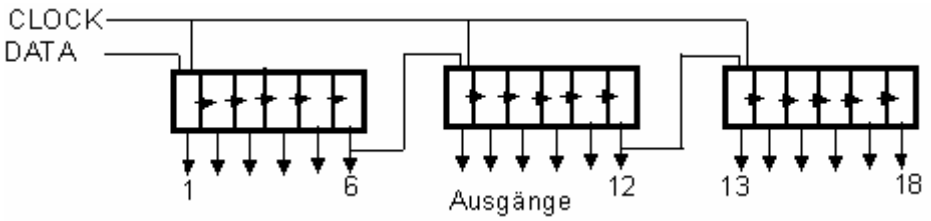


Figure 57 Cascading three 74HCT174

Figure 58 shows the circuit diagram for the port enhancement using one 74HCT174.

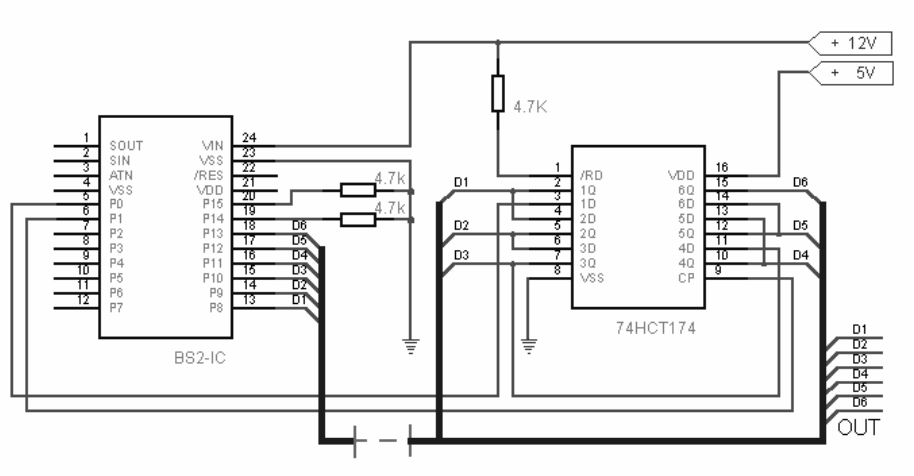


Figure 58 Port Enhancement with 74HCT174

The BASIC Stamp 2's P0 is used for data output and I/O pin 1 is for the clock. The /RD input of 74HCT174 is connected high using a pull-up resistor of 4.7 kOhm. In this configuration the shift register is setup for shifting in data serially. Pay careful attention to the wiring connection between the six flip-flops to get a properly functioning shift register.

The six outputs of the shift register, shown on the right site of the diagram on the bus are available for your project. But why reconnect the six output lines to the BASIC Stamp? This seems to negate the additional outputs we've created!

This connection, denoted by a dashed line, is only for demonstration. By connecting the parallel output to the BASIC Stamp, we can re-read it to see the result in the Debug Window. While running the program we can verify the equality of transmitted and received message.

Listing 17 shows the source of the program SHIFTREG.BS2.

```
'{$STAMP BS2}

' -----[ Title ]-----------------------------------------------
'
' File......  SHIFTREG.BS2
' Purpose...  Port Enhancement with serial connected
'             shift regiser
' Author....  Klaus Zahnert
' Started...  06.06.01
' Updated...

' -----[ Program Description ]-------------------------------
'
' Two output lines for clock and data to drive a connected
' shiftregiser are used for port enhancing up to 6 output-lines.

'For demonstration these outputs are inputs of BS2-IC. So the
'serial transmitted states of that 6 lines are shown on debug-
'window to see the same contents.

' -----[ Revision Hisory ]--------------------------------
'
' -----[ Constants ]-------------------------------------------
'
clkpin   con 1
datapin con 0

' -----[ Variables ]-------------------------------------------
'
outbyt   var byte

' -----[ Initialization ]--------------------------------------
'
     DIRH = $00              'I/O Port.highbyte for input
     low clkpin
     low datapin

' -----[ Main Code ]-------------------------------------------
'
start:
     For outbyt = 0 to 31
          shiftout datapin, clkpin, msbfirst, [outbyt\6]
          debug dec2 outbyt,tab,tab,bin8 outbyt,tab,bin8 INH ,cr
          pause 500
     next
     end
```

Listing 17 Port Enhancement with Shift Register (SHIFTREG.BS2)

In a loop the SHIFTOUT command sends the values 0 to 31 to the shift register. The DEBUG command can display the re-read value to compare with the value sent.

4 BASIC Stamps on the Net

TCP/IP is the standard for a platform-independent data exchange of different components via intranet or internet. A device connected with TCP/IP to the internet can accessed from any point in the internet. The infrastructure needed for this type of networking uses Ethernet networks, telephone lines, or even wireless. The device that should be integrated to a network only needs a TCP/IP stack.

Depending on the application there are very different solutions to make this work.

The resources required for implementing a TCP/IP stack are not available in small microcontrollers like the BASIC Stamp. A simple way out is to use the PC as a gateway.

4.1 MondoMini Webserver

The MondoMini Webserver (www.mondomini.com) is a gateway installed on a PC and can connect any microcontroller with the internet (Figure 59).

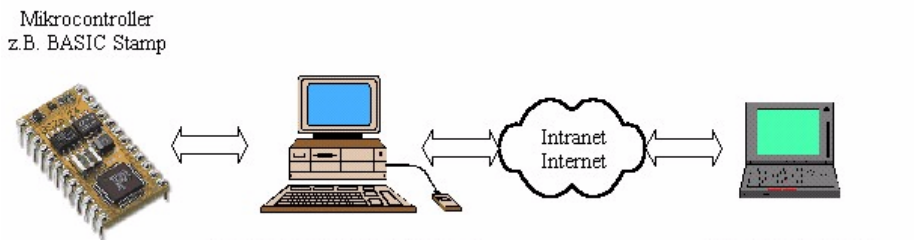


Figure 59 MondoMini Webserver as a Gateway

MondoMini Webserver has the following features:

Windows 98/NT/2000 compatibility

Usage of the PC to display web sites which were updated by the microcontroller

Update of websites via FTP

Transmission of control data to the microcontroller

Display of messages generated by the microcontrollers on a web site

Sending of an E-Mail triggered by event of the microcontroller application

The MondoMini Webserver runs on a PC using Windows 98/NT/2000. Simple commands over the serial interface build the communication between the microcontroller and the MondoMini Webserver. The MondoMini Webserver includes the received data into HTML pages accessible by a Web browser.

4.2 BASIC Stamp connected to the MondoMini Webserver

In the next program examples we can use any BS2 connected to the MondoMini Webserver. We do not need the special features of the BS2p for this application.

The hardware base for the following examples is the circuit diagram shown in Figure 60. If you use the BASIC Stamp Activity Board then you already have the complete circuitry.

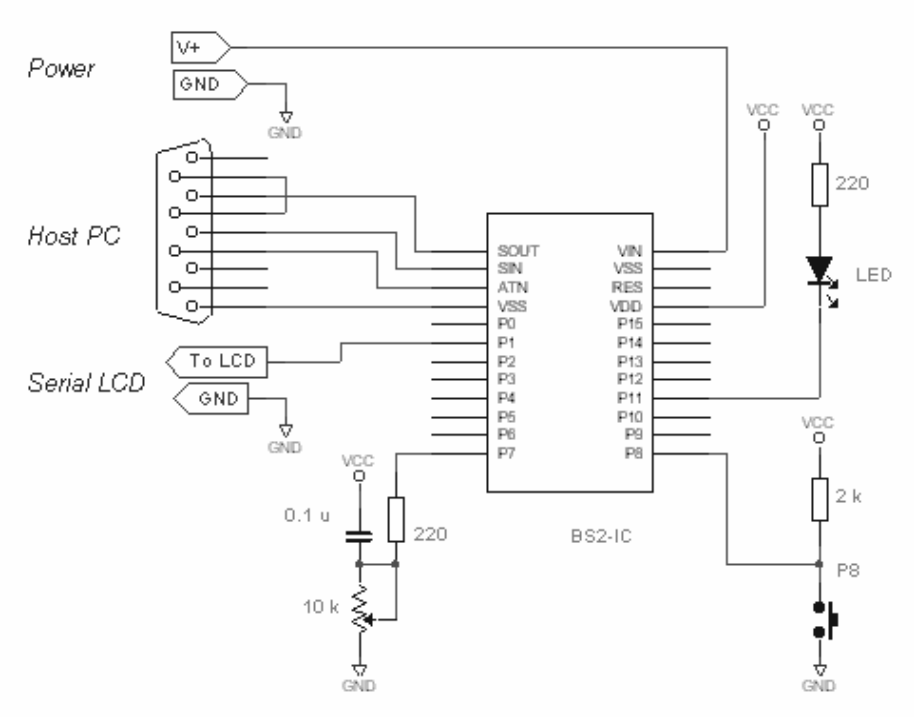


Figure 60 BS2 with MondoMini Webserver

To run the program examples the BS2 must be connected serial to the PC according to Figure 60 and the MondoMini Webserver must be running on the Host PC.

Because the MondoMini Webserver is connected to the programming interface of the BS2 we have to use the following commands for the communication between MondoMini Webserver and the BS2:

BS2 transmits:

```
SEROUT 16,84,1,[...]
```

BS2 receives:

```
        SERIN 16,84,1000,nocommand,[STR string\7\";"]
```

4.2.1 Sending E-Mails

Sending an E-Mail can be initiated at a selected time.

In our first program example the BS2 sends an E-Mail after the pushbutton on P8 is pressed.

In an endless loop we'll check the pushbutton on P8. A blinking LED signalizes that the program is running.

After detecting that the pushbutton is pressed we query the potentiometer connected to P7. You can put any hardware on this pin for any procedure or data acquisition. We display the result on a serial connected LCD for verification before the E-Mail is built and sent. Listing 18 shows the program responsible for sending E-Mails.

```
' This program demonstrates sending email from BASIC Stamp 2
' to one email address.
' BASIC Stamp Activity Board was used as target hardware.
' After pressing a key (P8) the pot meter connected to P7 is
' read and the pot meter value is sent afterwards.
' The serial link is connected to a PC's COM Port running
' MondoMini webserver.

' Created: 28.02.2001 Claus Kuhnel

LCD      con 1                          ' Serial LCD on P1
LED      con 10
POTIN    con 7                          ' Potentiometer on P7
N2400    con $418c                      ' Baudrate for serial LCD
I        con 254                        ' Instruction prefix value.
CLR      con 1                          ' LCD clear-screen instruction

adc  var word                           ' Word variable for ADC

     pause 1000
     SEROUT LCD,n2400,[I,CLR]           ' Clear the LCD screen.
     pause 1
     SEROUT LCD,n2400,[I,128]
     SEROUT LCD,n2400,["Value:"]        ' Print message.

start:
     high LED: pause 200                ' Blink LED
     low LED :pause 10
     IF In8 <> 0 THEN pass      ' Pass if key is not pressed
     gosub readpot              ' Read pot meter value
     SEROUT LCD,n2400,[I,135]   ' Print message on LCD
     SEROUT LCD,N2400,[DEC2 adc.lowbyte]
```

```
     gosub sendmail
pass:    goto start

readpot:
     high potin
     pause 1
     RCTIME POTIN, 1, adc
     adc=adc/2
     adc=adc.nib2
     return

sendmail:
     SEROUT 16,84+$4000,1,["EM=info@ckuehnel.ch;"]
     SEROUT 16,84+$4000,1,["This is an email alert generated by\n;"]
     SEROUT 16,84+$4000,1,["BASIC Stamp and MondoMini Webserver.\n;"]
     SEROUT 16,84+$4000,1,["P8 key was pressed on BS Activity Board.\n;"]
     SEROUT 16,84+$4000,1,["Read Pot value is ",dec adc,".;"]
     SEROUT 16,84+$4000,1,["EM;"]
     return
```

Listing 18 Sending E-Mails (EMAIL.BS2)

Building the E-Mail is quite simple. The tag `EM=...` marks the E-Mail address of the receiver. The text to be sent follows this address and the E-Mail is closed by the tag `EM`.

Now you can click the “Events” tab on MondoMini Webserver to show all activities of the MondoMini Webserver. Figure 61 shows this screen.

Figure 61 Webserver Protocol

At 12:44:19 an e-Mail was sent triggered by the pushbutton P8 on the BASIC Stamp Activity Board. Compare the protocol and program Listing 18 to verify.

The E-Mail received is shown below. Figure 62 shows the received E-Mail in the mail program Eudora Light. The appearance of this e-Mail depends of the e-Mail client you are using, of course. The content would be the same.

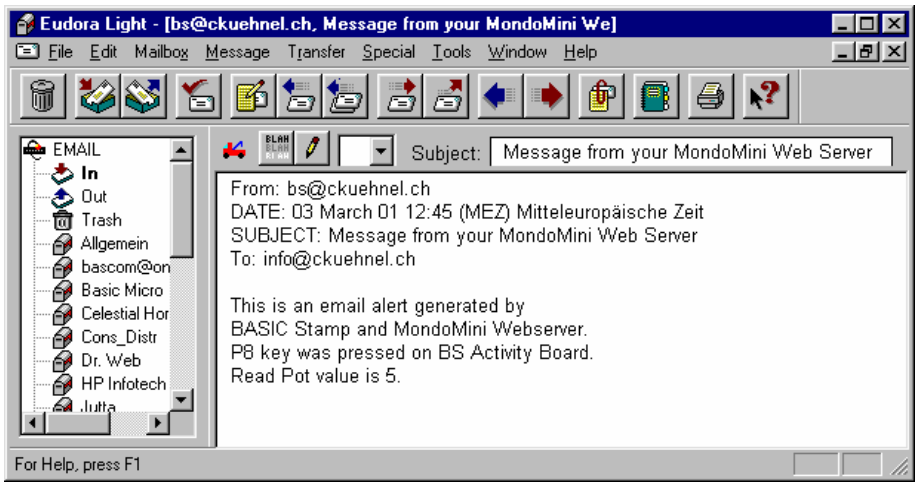


Figure 62 Received E-Mail

4.2.2 Query of Variables

The e-Mail in the last chapter showed the value of the potentiometer (5) as a variable inserted in the text of the message.

If one wants to query a variable in the BS2 using the web browser then MondoMini must know this variable's name. The BS2 application program would transmit the content of the variable to MondoMini so it can be queried by the web browser. The following program example explains how the BS2 transfers a variable to the MondoMini Webserver.

Listing 19 contains an endless loop which queries the potentiometer and transmits the value to the LCD and to MondoMini Webserver. The command `SEROUT 16,84+$4000,1,["P1=",DEC adc,";"]` sends the variable `adc` to MondoMini. The web server can identify this variable by the tag `P1`. The blinking LED flashes to demonstrate the program is operating.

```
' This program demonstrates sending a variable from BS2
' to MondoMini web server.
' Your web browser can read this value and display.
' BASIC Stamp Activity Board was used as target hardware.
' After pressing a key (P8) the pot meter connected to P7
' is read and the pot meter value is sent afterwards.
' The serial link is connected to a PC's COM Port running
' MondoMini webserver.
```

```
' Created: 28.02.2001 Claus Kuhnel

LCD       con 1                         ' Serial LCD on P1
LED       con 10
POTIN     con 7                         ' Potentiometer on P7
N2400     con $418c                     ' Baudrate for serial LCD
I         con 254                       ' Instruction prefix value.
CLR       con 1                         ' LCD clear-screen instruction

adc       var word                      ' Word variable for ADC

     pause 1000
     SEROUT LCD,n2400,[I,CLR]           ' Clear the LCD screen.
     pause 1
     SEROUT LCD,n2400,[I,128]
     SEROUT LCD,n2400,["Value:"]        ' Print message.

start:
     high LED: pause 500:               ' Blink LED
     low LED :pause 10
     gosub readpot
     SEROUT LCD,n2400,[I,135]
     SEROUT LCD,N2400,[DEC2 adc.lowbyte] ' Print value on LCD
     SEROUT 16,84+$4000,1,["P1=",DEC adc,";"]
     goto start

readpot:
     high potin
     pause 1
     RCTIME POTIN, 1, adc
     adc=adc/2
     adc=adc.nib2
     return
```

Listing 19 Sending a Variable (PUTVAR.BS2)

For querying a variable using a web browser you need to install an HTML program. We need no special features here and can use any text editor to write this HTML program. Figure 63 shows the presentation of this file in the Internet Explorer before we have a look to the HTML text itself.

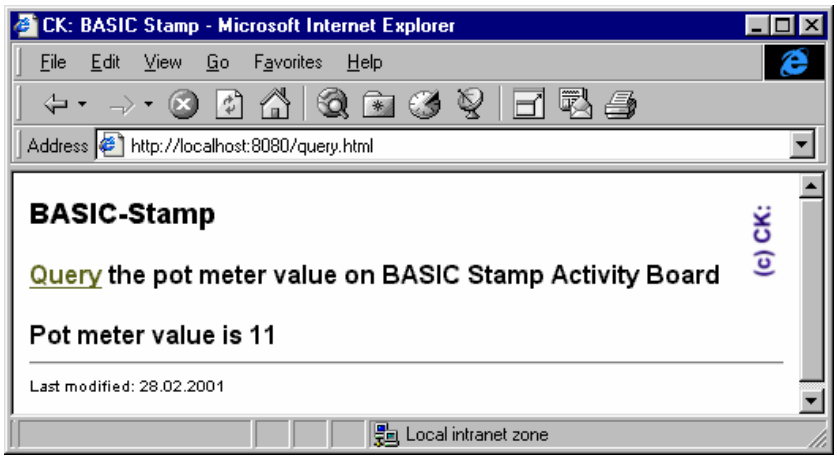


Figure 63 Query a Variable (QUERY.HTML)

A click to the hyperlink "Query" starts a query to the MondoMini Webserver and it transmits the subject value. Listing 20 shows the HTML text of the page shown in Figure 63.

```
<HTML>
<HEAD>
   <TITLE>CK: BASIC Stamp</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<H3><FONT SIZE="+3" FACE="Arial"><B><IMG SRC="ck.gif" WIDTH=20 HEIGHT=55
ALIGN=right></B></FONT><FONT FACE="Arial">BASIC-Stamp</FONT></H3>

<P><A HREF="query.html"><FONT FACE="Arial"><B>Query</B></FONT></A>
<FONT FACE="Arial"><B>the pot meter value on BASIC Stamp Activity Board</B></FONT></P>

<P><FONT FACE="Arial"><B>Pot meter value is 'P1</B></FONT>

<HR>

<FONT SIZE="-2" FACE="Arial">Last modified: 28.02.2001</FONT></P>
</BODY>
</HTML>
```

Listing 20 Query a Variable (QUERY.HTML)

By clicking the links the program QUERY.HTML refreshes the web page. The variable P1, linked to the variable `adc` in the BS2 application program will be displayed by the Web browser.

4.2.3 Changing of Variables

If you want to initialize or modify a variable in the BS2 application program using the web browser you must know whether the webserver has access to the client. In our case the BS2 is the client (Client-Server-Model).

The BS2 application program has to ask the Webserver if there are new commands or data for it. Listing 21 shows an example BS2 program. The new commands here are marked in bold.

```
' This program demonstrates sending a variable
' from MondoMini web server to BASIC Stamp 2.
' A click in your web browser sets a flag on BS2.
' BASIC Stamp Activity Board was used as target hardware.
' The serial link is connected to a PC's COM Port running
' Mondo Mini webserver.

' Created: 28.02.2001 Claus Kuhnel

string   var byte(8)
flag     var bit

LCD      con 1                        ' Serial LCD on P1
LED      con 10
POTIN    con 7                        ' Potentiometer on P7
N2400    con $418c                    ' Baudrate for serial LCD
I        con 254                      ' Instruction prefix value.
CLR      con 1                        ' LCD clear-screen instruction


     DIRS =%0000111100000000
     OUTC =%1111

     pause 1000
     SEROUT LCD,n2400,[I,CLR]         ' Clear the LCD screen.
     pause 1
     SEROUT LCD,n2400,[I,128]
     SEROUT LCD,n2400,["Value:"]      ' Print message.

     'Tell MondoMini to clear all commands queued up
     SEROUT 16,84+$4000,1,["CC;"]

start:
     high LED: pause 500:             ' Blink LED
     low LED :pause 10


     'Query the MondoMini for a command.
     SEROUT 16,84+$4000,1,["QC;"]
```

```
     'Wait 1000 ms for a command from MondoMini
     SERIN 16,84+$4000,1000,nocommand,[STR string\7\";"]

     'Test for "P1=1"
     IF (string(0)<>"P" OR string(1)<>"1" OR string(3)<>"1") THEN nextcommand
          flag = 1 : OUT8 = flag
          SEROUT LCD,n2400,[I,135]
          SEROUT LCD,N2400,[BIN1 flag] ' Print value on LCD
          SEROUT 16,84+$4000,1,["P1=", BIN1 flag, ";"]

nextcommand:
     'Test for "P1=0"
     IF (string(0)<>"P" OR string(1)<>"1" OR string(3)<>"0") THEN nextcommand1
          flag = 0 : OUT8 = flag
          SEROUT LCD,n2400,[I,135]
          SEROUT LCD,N2400,[BIN1 flag] ' Print value on LCD
          SEROUT 16,84+$4000,1,["P1=", BIN1 flag, ";"]

nextcommand1:

nocommand:
     goto start
```

Listing 21 Receiving a Command (GETVAR.BS2)

Before running the endless loop the program clears everything in MondoMini Webserver by the command "CC". Inside the endless loop the program queries the MondoMini Webserver periodically for received commands.

The command `SERIN 16,84+$4000,1000,nocommand,[STR string\7\";"]` reads a command with maximum of 7 characters or until the character ";" in the variable `string` is encountered. If no command is received from MondoMini Webserver the BS2 goes into timeout and runs the loop again.

If a command is received then we have to decode it. Valid are the commands P1=0 and P1=1 only; all other commands are ignored. Depending on the flag variable the BS2 application program controls several displays (LED, LCD).

For setting or resetting this flag from the web browser an appropriate HTML program was installed on the Webserver. Figure 64 shows the web side in the Internet Explorer belonging to it.

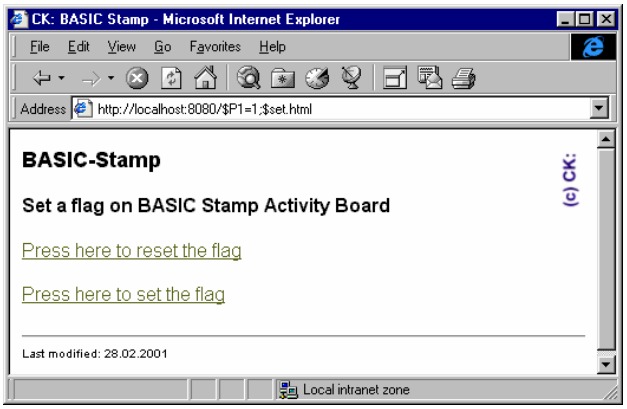


Figure 64 Setting a Flag (SET.HTML)

A click on one of the hyperlinks sends a variable to MondoMini Webserver where it is saved for queries by the BS2. Listing 22 shows the HTML text of that web side shown in Figure 64. Both hyperlinks were marked bold later.

```
<HTML>
<HEAD>
   <TITLE>CK: BASIC Stamp</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<H3><FONT SIZE="+3" FACE="Arial"><B><IMG SRC="ck.gif" WIDTH=20 HEIGHT=55
ALIGN=right></B></FONT><FONT FACE="Arial">BASIC-Stamp</FONT></H3>

<P><FONT FACE="Arial"><B>Set a flag on BASIC Stamp Activity
Board</B></FONT></P>

<P><A HREF="$P1=0;$set.html"><FONT FACE="Arial">Press here to reset the
flag</FONT></A></P>

<P><A HREF="$P1=1;$set.html"><FONT FACE="Arial">Press here to set the
flag</FONT></A></P>

<HR>

<FONT SIZE="-2" FACE="Arial">Last modified:
28.02.2001</FONT></P>
</BODY>
</HTML>
```

Listing 22 Setting a Flag (SET.HTML)

4.2.4 BASIC Stamp Monitoring System

Based on the explanations we've provided we can build a monitoring system with the following features:

- Query of a measuring value
- Alarm after exceeding one of the defined limits by E-Mail
- Periodic query of a measured value and limit by a Web browser
- Signalization of the Alarm on a web site displayed using a web browser
- Setting the limit via a web page

The measuring procedure is simulated by a query of the potentiometer as before. Listing 23 shows the BS2 application program. All commands important for the communication with the MondoMini Webserver were marked bold later again.

```
' This program demonstrates setting a limit in a BS2
' application program by your web browser.
' BASIC Stamp Activity Board with a serial LCD was
' used as target hardware.
' The BS2 application reads periodically the pot meter value,
' displays them on LCD and sends an email when the value is
' over the limit.
' The serial link is connected to a PC's COM Port running
' Mondo Mini webserver.

' Created: 28.02.2001 Claus Kuhnel


LCD      con 1                          ' Serial LCD on P1
LED      con 10
POTIN    con 7                          ' Potentiometer on P7
N2400    con $418c                      ' Baudrate for serial LCD
I        con 254                        ' Instruction prefix value.
CLR      con 1                          ' LCD clear-screen instruction

string   var byte(8)                    ' Command string
limit    var byte                       ' Byte variable for limit
adc      var word                       ' Word variable for ADC
number   var word                       ' Word variable for number conversion
ii       var nib                        ' Index
emailsentvar bit                        ' Flag


     DIRS =%0000111100000000
     OUTC = %1111

     limit = 15                         ' Initialize limit with maximum

     'Initialize the LCD
```