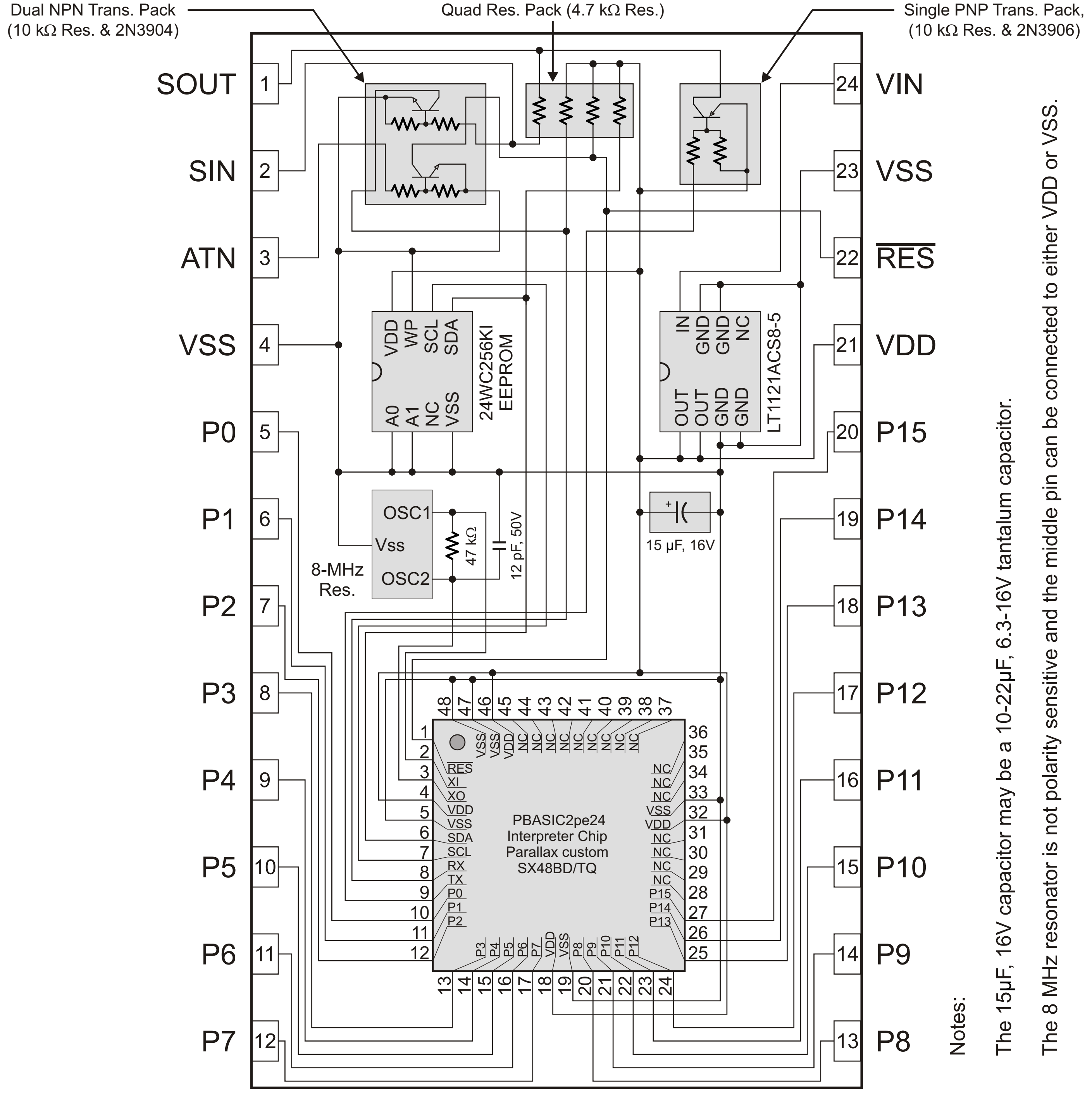
PARALLAX INC

599 Menlo Drive, Suite 100
Rocklin, CA 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com
Educational: www.stampsinclass.com

BASIC Stamp® BS2pe-IC Module Schematic Rev. B

Updated 10-25-04 © Parallax, Inc.


Dual NPN Trans. Pack (10 kΩ Res. & 2N3904)
Quad Res. Pack (4.7 kΩ Res.)
Single PNP Trans. Pack, (10 kΩ Res. & 2N3906)
SOUT 1
SIN 2
ATN 3
VSS 4
P0 5
P1 6
P2 7
P3 8
P4 9
P5 10
P6 11
P7 12
VIN 24
VSS 23
RES 22
VDD 21
P15 20
P14 19
P13 18
P12 17
P11 16
P10 15
P9 14
P8 13
VDD WP SCL SDA
A0 A1 NC VSS
24WC256KI EEPROM
IN GND GND NC
OUT OUT GND GND
LT1121ACS8-5
OSC1
Vss
OSC2
8-MHz Res.
47 kΩ
12 pF, 50V
15 µF, 16V
RES XI XO VDD VSS SDA SCL RX TX P0 P1 P2
VSS VSS VDD NC NC NC NC NC NC NC NC NC
P3 P4 P5 P6 P7 VDD VSS P8 P9 P10 P11 P12
NC NC NC VSS VDD NC NC NC NC P15 P14 P13
PBASIC2pe24 Interpreter Chip Parallax custom SX48BD/TQ

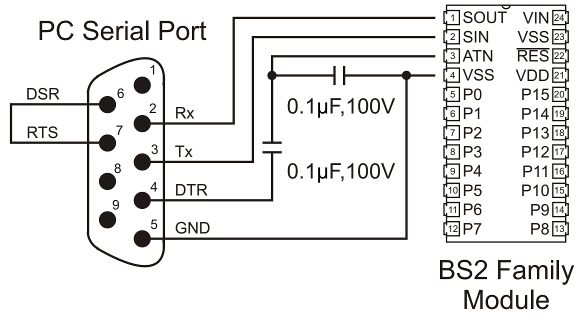
Notes:

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

The 8 MHz resonator is not polarity sensitive and the middle pin can be connected to either VDD or VSS.

Connect DSR and RTS for automatic port detection.

PC to BS2 Family Connection


PC Serial Port
DSR
RTS
Rx
Tx
DTR
GND
0.1µF,100V
0.1µF,100V
SOUT SIN ATN VSS P0 P1 P2 P3 P4 P5 P6 P7
VIN VSS RES VDD P15 P14 P13 P12 P11 P10 P9 P8
BS2 Family Module

Stamp Specifications (revised 04/05)

Released Products	Rev.Dx / BS1-IC	BS2-IC	BS2e-IC	BS2sx-IC
Package	PCB w/Proto / 14-pin SIP	24-pin DIP	24-pin DIP	24-pin DIP
Package Size (L x W x H)	2.5" x 1.5" x .5" / 1.4" x .6" x .1"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"
Environment *	0º - 70º C (32º - 158º F) **	0º - 70º C (32º - 158º F) **	0º - 70º C (32º - 158º F)	0º - 70º C (32º - 158º F)
Microcontroller	Microchip PIC16C56a	Microchip PIC16C57c	Ubicom SX28AC	Ubicom SX28AC
Processor Speed	4 MHz	20 MHz	20 MHz	50 MHz
Program Execution Speed	~2,000 instructions/sec.	~4,000 instructions/sec.	~4,000 instructions/sec.	~10,000 instructions/sec.
RAM Size	16 Bytes (2 I/O, 14 Variable)	32 Bytes (6 I/O, 26 Variable)	32 Bytes (6 I/O, 26 Variable)	32 Bytes (6 I/O, 26 Variable)
Scratch Pad RAM	N/A	N/A	64 Bytes	64 Bytes
EEPROM (Program) Size	256 Bytes, ~80 instructions	2K Bytes, ~500 instructions	8 x 2K Bytes, ~4,000 inst.	8 x 2K Bytes, ~4,000 inst.
Number of I/O pins	8	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial
Voltage Requirements	5 - 15 vdc	5 - 15 vdc	5 - 12 vdc	5 - 12 vdc
Current Draw @ 5V	1 mA Run / 25 µA Sleep	3 mA Run / 50 µA Sleep	25 mA Run / 200 µA Sleep	60 mA Run / 500 µA Sleep
Source / Sink Current per I/O	20 mA / 25 mA	20 mA / 25 mA	30 mA / 30 mA	30 mA / 30 mA
Source / Sink Current per unit	40 mA / 50 mA	40 mA / 50 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins
PBASIC Commands***	32	42	45	45
PC Programming Interface	Serial (w/BS1 Serial Adapter)	Serial (9600 baud)	Serial (9600 baud)	Serial (9600 baud)
Windows Text Editor	Stampw.exe (v2.1 and up)	Stampw.exe (v1.04 and up)	Stampw.exe (v1.096 and up)	Stampw.exe (v1.091 and up)

Released Products	BS2p24-IC	BS2p40-IC	BS2pe-IC	BS2px-IC	Javelin Stamp
Package	24-pin DIP	40-pin DIP	24-pin DIP	24-pin DIP	24-pin DIP
Package Size (L x W x H)	1.2" x 0.6" x 0.4"	2.1" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"	1.24" x 0.60" x 0.45"
Environment *	0º - 70º C (32º - 158º F)	0º - 70º C (32º - 158º F)	0º - 70º C (32º - 158º F)	0º - 70º C (32º - 158º F)	0º - 70º C (32º - 158º F)
Microcontroller	Ubicom SX48AC	Ubicom SX48AC	Ubicom SX48AC	Ubicom SX48AC	Ubicom SX48AC
Processor Speed	20 MHz Turbo	20 MHz Turbo	8 MHz Turbo	32 MHz Turbo	25 MHz Turbo
Program Execution Speed	~12,000 instructions/sec.	~12,000 instructions/sec.	~6000/sec.	~19,000 instructions/sec.	~8,500 instructions/sec.
RAM Size	38 Bytes (12 I/O, 26 Variable)	38 Bytes (12 I/O, 26 Variable)	38 Bytes (12 I/O, 26 Variable)	38 Bytes (12 I/O, 26 Variable)	32768 Bytes
Scratch Pad RAM	128 Bytes	128 Bytes	128 Bytes	128 Bytes	N/A
EEPROM (Program) Size	8 x 2K Bytes, ~4,000 inst.	8 x 2K Bytes, ~4,000 inst.	16 x 2K Bytes (16 K for source)	8 x 2K Bytes, ~4,000 inst.	32768 Bytes
Number of I/O pins	16 + 2 Dedicated Serial	32 + 2 Dedicated Serial	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial	16
Voltage Requirements	5 - 12 vdc	5 - 12 vdc	5 - 12 vdc	5 - 12 vdc	5 - 24 vdc
Current Draw @ 5V	40 mA Run / 350 µA Sleep	40 mA Run / 350 µA Sleep	15 mA Run / 36 µA Sleep	55 mA Run / 450 µA Sleep	80 mA Run / No Sleep
Source / Sink Current per I/O	30 mA / 30 mA	30 mA / 30 mA	30 mA / 30 mA	30 mA / 30 mA	30 mA / 30 mA
Source / Sink Current per unit	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins
PBASIC Commands***	61	61	61	63	0 (Java)
PC Programming Interface	Serial (9600 baud)	Serial (9600 baud)	Serial (9600 baud)	Serial (19200 baud)	Serial (28800 baud)
Windows Text Editor	Stampw.exe (v1.1 and up)	Stampw.exe (v1.1 and up)	Stampw.exe (v1.33 and up)	Stampw.exe (v2.2 and up)	Javelin Stamp IDE

* 70% Non-Condensing Humidity

** Industrial Models Available, -40º - 85º C (-40º - 185º F). Contact Parallax Sales for information.

*** Using PBASIC 2.5 for BS2-type models.

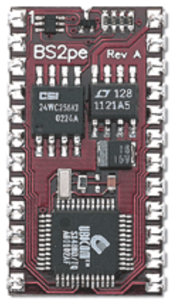
BS2pe Data Logger Application Note

Parts used:

- BS2pe (#BS2PE)
- BS2p24 demo board (#45183) (suggested but not required)
- Parallel LCD (#603-00006)
- DS1822 (#604-00013)
- Push button (#400-00001)
- 10k ohm resistor (#150-01030)
- 4.7k ohm resistor (#150-04720)

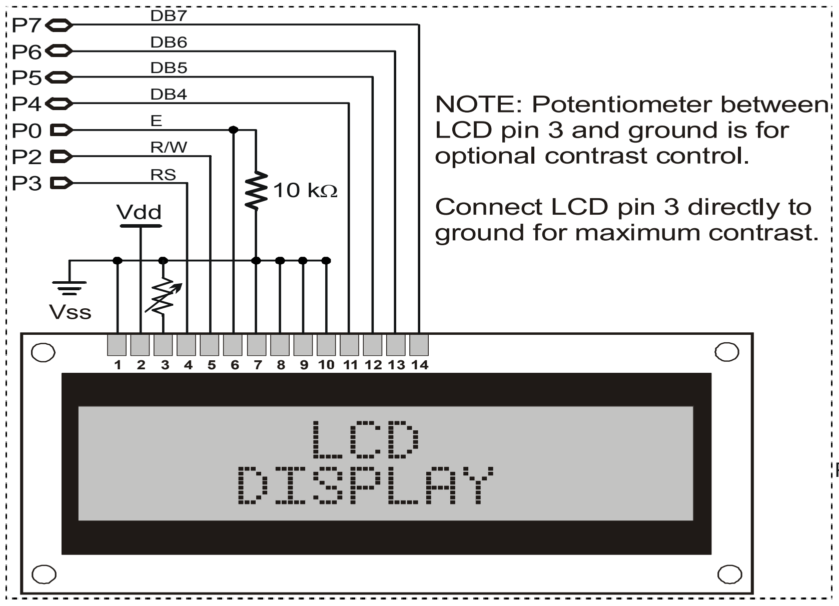
General Information

The following code and setup will implement the BS2pe features of data logging, Parallel LCD interface, and expanded EEprom.



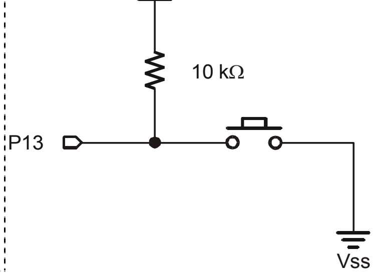
Control from a BASIC Stamp

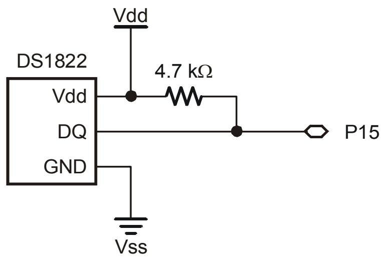
For easy wiring of the LCD, Parallax would suggest the BS2p24 demo board, but it is not required. The wiring for the LCD does require numerous I/O pins as seen below. If you use the demo board then you will not need to wire the LCD section below.



NOTE: Potentiometer between LCD pin 3 and ground is for optional contrast control.

Connect LCD pin 3 directly to ground for maximum contrast.





```
' Temp_lcd_data_demo.bs2
' Purpose... Reads and displays information from a Dallas DS1822,and stores data across memory
blocks.
' {$STAMP BS2pe}
' {$PBASIC 2.5}

' -----[ Declarations ]----------------------------------------------------
OWpin           CON     15
LCDpin              CON       0
NoCmd           CON     $00                     ' No command in LCDOUT
ClrLCD          CON     $01                     ' clear the LCD
CrsrHm          CON     $02                     ' move cursor to home position
CrsLf           CON     $10                      ' move cursor left
CrsRt           CON     $14                      ' move cursor right
DispLf          CON     $18                     ' shift displayed chars left
DispRt          CON     $1C                     ' shift displayed chars right
DDRam           CON     $80                     ' Display Data RAM control
CGRam           CON     $40                     ' Custom character RAM
Line1           CON     $80                     ' DDRAM address of line 1
Line2           CON     $C0                     ' DDRAM address of line 2
UNit_off        CON     $08
Unit_on             CON     $0d

OW_FERst        CON     %0001                   ' Front-End Reset
OW_BERst        CON     %0010                   ' Back-End Reset
OW_BitMode      CON     %0100
OW_HighSpd      CON     %1000

ReadROM         CON     $33                     ' read ID, serial num, CRC
MatchROM        CON     $55                     ' look for specific device
SkipROM         CON     $CC                     ' skip rom (one device)
SearchROM       CON     $F0                     ' search


CnvrtTemp       CON     $44                     ' do temperature conversion
RdScratch       CON     $BE                     ' read scratchpad

NoDevice        CON     %11                     ' no device present
DS1822          CON     $22                     ' device code
DegSym          CON     176


DevCheck        VAR     Nib                     ' device check return ocde
Idx             VAR     Byte                    ' loop counter
RomData         VAR     Byte(8)                 ' ROM data from DS1820
TempIn          VAR     Word                    ' raw temperature
Sign            VAR     tempIn.Bit11            ' 1 = negative temperature
TLo             VAR     tempIn.LowByte
THi             VAR     tempIn.HighByte
TSign           VAR     Bit
TempF           VAR     Word                    ' Fahrenheit
TempC           VAR     Word                    ' Celsius
Run_bit   VAR     Word
Bank                VAR       Byte                          'bank address
Address   VAR       Word                        'address
Readings  VAR     Word                          'Readings
Temp                VAR     Word                          'Work space varablie


' -----[ Initialization ]--------------------------------------------------
'
Initialize:



          PAUSE 500                                     ' let the LCD settle
          LCDCMD LCDpin,%00110000 : PAUSE 5             ' 8-bit mode
          LCDCMD LCDpin,%00110000 : PAUSE 0
          LCDCMD LCDpin,%00110000 : PAUSE 0
          LCDCMD LCDpin,%00100000 : PAUSE 0             ' 4-bit mode
          LCDCMD LCDpin,%00001100 : PAUSE 0             ' no crsr, no blink
          LCDCMD LCDpin,%00000110 : PAUSE 0                     ' inc crsr, no disp shift
          LCDCMD LCDpin,%00101000 : PAUSE 0                     ' 2 Line 5x8 font
          LCDCMD LCDpin,ClrLCD
pause 1000



' -----[ Main Routine ]--------------------------------------------------
          Bank = 1
Main:


          IF run_bit = 1 then Display_Temperatures
GOSUB Device_Check                          ' look for device
          IF (devCheck <> NoDevice) THEN Get_ROM




No_Device_Found:
          LCDOUT LCDpin,Line1,["   No DS1822"] : PAUSE 5
          LCDOUT LCDpin,Line2,["    Present"]
          PAUSE  1500
          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["Insert Ds1822"] : PAUSE 5
          LCDOUT LCDpin,Line2,["and reset stamp"]
          PAUSE 1500
          LCDCMD LCDpin,ClrLCD  : PAUSE 0
goto main

Get_ROM:
          OWOUT OWpin,OW_FERst,[ReadROM]                ' send Read ROM command
          OWIN  OWpin,OW_BERst,[STR romData\8]          ' read serial number & CRC
          IF (romData(0) = DS1822) THEN Show_Data
          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["Installed device"]
          PAUSE 5
          LCDOUT LCDpin,Line2,["is not DS1822"]    : PAUSE 5
          LCDOUT LCDpin,NoCmd,["Code = ",HEX2 romData(0)]
          PAUSE 1500
goto main

Show_Data:

          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["DS1822 Data"]
          PAUSE 1500
          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["Serial Number : "]    : PAUSE 5
          FOR idx = 6 TO 1
          LCDOUT LCDpin,Line2,[HEX2 romData(idx)]
NEXT
          PAUSE 1500
          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["Checksum : ",HEX2 romData(7)]
          PAUSE 2000



Display_Temperatures:

          run_bit = 1

GOSUB Get_Temp

          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["Temp ","F:", SDEC tempF]     : PAUSE 0

skip_over:

          readings = tempF
          IF in13 = 0 THEN full
GOSUB record

GOTO Main


' -----[ Subroutines ]---------------------------------------------------
'
' This subroutine checks to see if any 1-Wire devices are present on the
' bus.  It does NOT search for ROM codes
'
Device_Check:

          devCheck = 0
          OWOUT OWpin,OW_FERst,[SearchROM]              ' reset and start search
          OWIN  OWpin,OW_BitMode,[devCheck.Bit1,devCheck.Bit0]
RETURN
```

```
Get_Temp:
          FOR temp = 1 TO 250
          OWOUT OWpin,OW_FERst,[SkipROM,CnvrtTemp]      ' send conversion command
          PAUSE 1                                       ' give it some time

NEXT

          OWOUT OWpin,OW_FERst,[SkipROM,RdScratch]      ' go get the temperature
          OWIN  OWpin,OW_BERst,[tLo,tHi]
          tSign = sign                                  ' save sign bit
          tempC = tempIn
          tempC = tempC >> 4                            ' round to whole degrees

          IF (tSign = 0) THEN NoNegC
          tempC = tempC | $FF00                         ' extend sign bits for negs

NoNegC:
           tempF = tempC */ $01CD                        ' multiply by 1.8
           IF tSign = 0 THEN NoNegF                      ' if neg, extend sign bits
           tempF = tempF | $FF00

NoNegF:
            tempF = tempF + 32                            ' finish C -> F conversion
RETURN

Full:

          FOR bank = 1 TO 15
          FOR address = 0 TO 2048 step 2
          STORE bank
          READ address ,readings.lowbyte
          READ address + 1,readings.highbyte

          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["Reading: ",dec address/2]   : PAUSE 0
          LCDOUT LCDpin,Line2,["Bank:",dec bank,"Temp ","F:", SDEC tempF]     : PAUSE 0
          PAUSE 1000 ' lessen this pause to increase rate of display
          NEXT
          NEXT

Sleeping:
          SLEEP 3
          LCDCMD LCDpin,unit_off  : PAUSE 0
          IF IN12 = 1 THEN sleeping
          LCDCMD LCDpin,unit_on  : PAUSE 0
GOTO sleeping



Record:
          PAUSE 1000 ‘Adjust to vary the rate of storage
          If address < 2040 THEN no_up
          bank = bank +1
          address = 0
No_up:

          'DEBUG 0,cr,"record: ",? bank,? address,? readings," Low ",dec readings.lowbyte,"  High
",dec readings.highbyte
          STORE bank
          WRITE address ,readings.lowbyte
          WRITE address + 1,readings.highbyte
          address = address + 2
          IF bank = 15 AND address = 2040 THEN filled
          return
filled:

          LCDCMD LCDpin,ClrLCD  : PAUSE 0
          LCDOUT LCDpin,Line1,["Full"]    : PAUSE 5
STOP
```

BASIC STAMP FAQ

(FREQUENTLY ASKED QUESTIONS AND ANSWERS)

What is the BASIC Stamp?

The BASIC Stamp is a microcontroller developed by Parallax, Inc. which is easily programmed using a form of the BASIC programming language. It is called a "Stamp" simply because it is close to the size of an average postage stamp.

Does the BASIC Stamp lose its program when I remove the battery or power supply?

No, your PBASIC code is stored inside a serial EEPROM on-board the BASIC Stamp. EEPROMs provide non-volatile storage; they retain memory even without power. The EEPROM used in the BASIC Stamp 1 and 2 is guaranteed to function properly for 40 years and for 10,000,000 write cycles per memory location. The EEPROM used in the BASIC Stamp 2e and 2sx is guaranteed for 1,000,000 write cycles per memory location.

Does the BASIC Stamp come in different versions?

Yes. Currently there are four functional versions of the BASIC Stamp and seven physical versions. The BASIC Stamp line consists of the BASIC Stamp I, the BASIC Stamp II, the BASIC Stamp IIe and the BASIC Stamp IIsx . The BASIC Stamp I is available in three package types (physical versions) as shown in figures 1a, 1b and 1c below. The BASIC Stamp 1 Rev. D is a through-hole, socketed package (with prototyping area). The BS1-IC is a 14-pin SIP with surface-mounted components. The OEMBS1 is a 14-pin SIP in a larger board layout meant for OEM use (for recreating the entire BASIC Stamp 1 circuitry). It's important to note the three versions are functionally equivalent (with the exception of the lack of a Reset pin on the BASIC Stamp 1 Rev. D) but are just in physically different packages. The BASIC Stamp II , BASIC Stamp IIe and BASIC Stamp IIsx are available in a 24-pin DIP, with surface-mounted components (see Figures 2a, 3 and 4. They are called the BS2-IC, BS2E-IC and BS2SX-IC, respectively. The BS2-IC is also available in an OEM version (Figure 2b) that is a 20-pin SIP package.

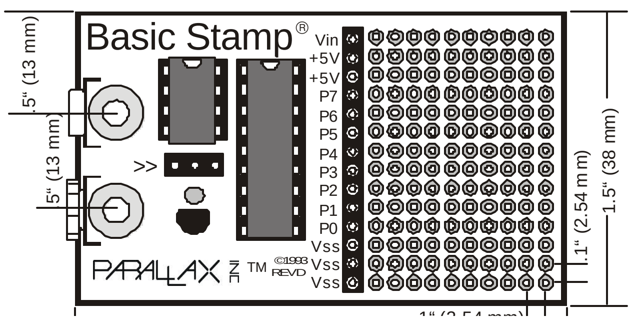


Figure 1a –BASIC Stamp 1 Rev. D

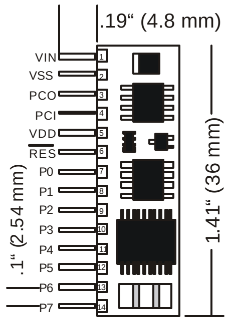


Figure 1b –BS1-IC

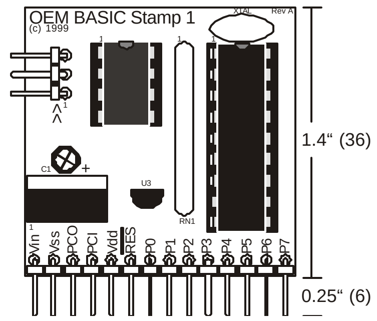


Figure 1c –OEMBS1

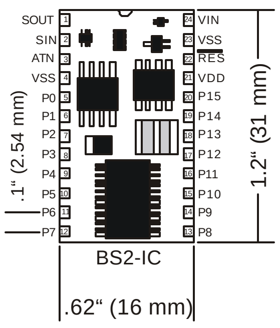


Figure 2a – BS2-IC

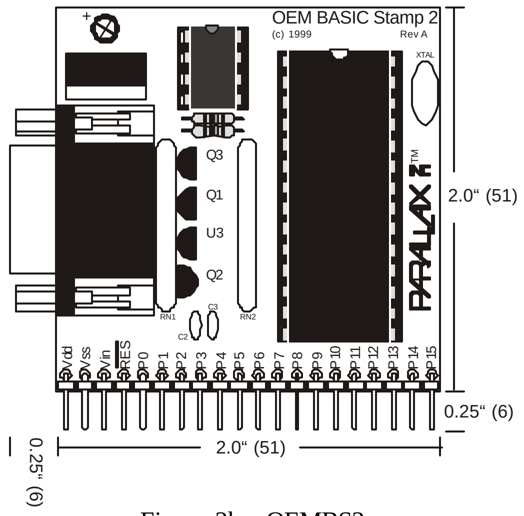


Figure 2b – OEMBS2

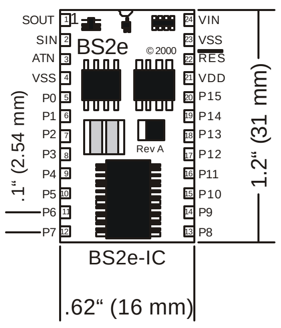


Figure 3 – BS2e-IC

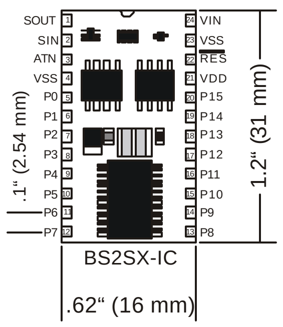


Figure 4 – BS2sx-IC

What is the difference between the BASIC Stamp Rev. D (sometimes just called "BASIC

The original version of the BASIC Stamp I was simply called, "BASIC Stamp". That name was sufficient until the arrival of the new physical package type, as well as the second model in the Stamp line, the BASIC Stamp II. At that point, the original version was split into two package types, the Rev. D and the BS1-IC; both of which are considered to be the "BASIC Stamp I". The "BASIC Stamp II" is known as the BS2-IC, the "BASIC Stamp IIe" is known as the BS2E-IC, and the "BASIC Stamp IIsx" is known as the BS2SX-IC. Today, the term "BASIC Stamp" is too generic and really could be applied to any one of the units in the product line. Long-time Parallax customers, however, still frequently refer to the BASIC Stamp I as the BASIC Stamp, thus, in many cases, there is no difference between the term BASIC Stamp and the BASIC Stamp I. To keep things clear, we prefer the names: "BASIC Stamp I", "BASIC Stamp II", "BASIC Stamp IIe", and "BASIC Stamp IIsx", or more specifically, -IC" , "BS2-IC" , "BS2E-IC" and "BS2SX-IC" and try to use the term "BASIC Stamp" to refer to the entire line of microcontrollers.

How big is the BASIC Stamp?

The BASIC Stamp rev. D measures 2.5" (65 mm) L x 1.52" (39 mm) W x 0.4" (10 mm) D.
The BS1-IC measures 1.4" (35 mm) L x 0.58" (15 mm) W x 0.13" (3 mm) D.
The BS2-IC measures 1.2" (30 mm) L x 0.62" (16 mm) W x 0.35" (9 mm) D.
The BS2E-IC measures 1.2" (30 mm) L x 0.62" (16 mm) W x 0.35" (9 mm) D.
The BS2SX-IC measures 1.2" (30 mm) L x 0.62" (16 mm) W x 0.35" (9 mm) D.

How big are the BASIC Stamp I and BASIC Stamp II Carrier Boards?

The BASIC Stamp I Carrier Board measures 2.5" (65 mm) L x 1.5" (38 mm) W x 0.5" (13 mm) D
The BASIC Stamp II Carrier Board measures 2.8" (71 mm) L x 3.1" (79 mm) W x 0.6" (15 mm) D.

How do I power the BASIC Stamp?

The BASIC Stamp runs on 5 to 15 volts DC. All BASIC Stamps feature an on-board 5-volt regulator which will convert an input 6 to 15 volts (on the VIN pin) down to the 5 volts that it's components require. If your power supply is 6 to 15 volts, you should connect it directly to the VIN and GND pins or to the battery clips on the development board. The VIN and GND pins are pin number 1 and 2 on the BASIC Stamp rev. D or BS1-IC or pin 24 and 23 on the BS2-IC, BS2E-IC and BS2SX-IC. If your power supply delivers a regulated 5 volts, you should connect it directly to the +5V and GND pins (14 and 2 on the BASIC Stamp rev. D, 5 and 2 on the BS1-IC, and 21 and 23 on the BS2-IC or BS2SX-IC). NOTE: When using battery or wall-pack power supplies, it is recommended to limit the voltage to 9 volts on the Stamp 1 and Stamp 2, and 7.5 volts on the Stamp 2e and Stamp 2sx.

How much current does the BASIC Stamp consume?

The BASIC Stamp I consumes 2 mA in running mode and 20 µA in sleep mode, not including any circuitry on the I/O pins. The BS2-IC consumes 8 mA in running mode and 100 µA in sleep mode, not including any circuitry on the I/O pins. The BS2e-IC consumes 20 mA in running mode and 100 µA in sleep mode, not including any circuitry on the I/O pins. The BS2SX-IC consumes 65 mA in running mode and 200 µA in sleep mode, not including any circuitry on the I/O pins. A full comparison of the Stamps can be found at the end of this document.

Is the BASIC Stamp sensitive to static electricity?

While many electronic devices, including BASIC Stamps, can be damaged by static electricity, the BASIC Stamp is generally less sensitive to static. We do, however, recommend taking all the usual precautions when handling BASIC Stamps in static prone environments.

Are there any production possibilities with the BASIC Stamp?

Yes. We offer the major components of the BASIC Stamp circuit (the interpreter, the EEPROM and the resonator) separately at a discounted price for tight integration into your products. For initial development purposes, request the OEMBS1 or OEMBS2 product. These are “OEM” versions of the modules with an easy to follow circuit layout and all the components provided in through-hole format. The “OEM” versions provide a great “base” unit, which you can build yourself, to test the circuit and help in your own designs.

What items do I need to get started with the BASIC Stamp?

The items you need are: 1) the programming software, 2) the programming cable, 3) the manual, 4) the BASIC Stamp module itself, and optionally 5) the appropriate development board. If you are interested in using a particular version of the BASIC Stamp, it is best to purchase the BASIC Stamp I, BASIC Stamp II, BASIC STAMP IIe or BASIC Stamp IIsx Starter Kits. These kits include all five of the items listed above at a lower cost than if you were to purchase each of the components separately. If you choose not to purchase a starter kit, you can retrieve all the documentation and software from our web site and simply purchase the BASIC Stamp module you desire.

How can I make my own BASIC Stamp circuit?

Parallax sells the BASIC Stamp’s interpreter chip in three different package type: DIP, SOIC and SSOP. Any of these can be used to create your own BASIC Stamp circuit for purposes of deeply embedding it within an application. For initial development purposes, request the OEMBS1 or OEMBS2 product. These are “OEM” versions of the modules with an easy to follow circuit layout and all the components provided in through-hole format. The “OEM” versions provide a great “base” unit, which you can build yourself, to test the circuit and help in your own designs. Please call Parallax Technical Support to receive information and pointers on other package types (particularly SSOP) as the pin configurations of the interpreter chips are different and are not properly shown in the 1995/96 Microchip documentation.

What kind of microchips are used in the BASIC Stamp?

The BASIC Stamp I uses a PIC16C56 from Microchip Technology Inc. The BASIC Stamp II uses a PIC16C57 Microchip Technology Inc. The BASIC Stamps IIe and IIsx use an SX28AC from Scenix.

What kind of operating environment does the BASIC Stamp require?

The BASIC Stamp modules will work in 0° to 70° C temperatures with up to 70%, non-condensing humidity. While the modules may continue to function outside these ranges, it is not guaranteed or recommended. Parallax also sells the BASIC Stamp I and BASIC Stamp II in industrial (-40° to 85° C) and extended (-40° to 125° C) ranges. (At the time of this writing, this option is not available for the BS2E-IC or BS2SX-IC) Additionally, it is best to keep the BASIC Stamps away from, or shielded from, any nearby RF interference as this may impact the accuracy of its I/O functions.

What are the main differences between the BASIC Stamps?

The BASIC Stamp I has 8 I/O pins, room for 80 to 100 lines of code, executes approximately 2000 instructions per second and requires a parallel interface for programming. The BASIC Stamp II has 16 I/O pins, 2 dedicated serial port pins (1 input, 1 output), room for 500 to 600 lines of code, executes approximately 4000 instructions per second and requires a serial interface for programming. The BASIC Stamp IIsx has 16 I/O pins, 2 dedicated serial port pins (1 input, 1 output), room for 4000 lines of code, executes approximately 10000 instructions per second and requires a serial interface for programming For a more detailed comparison, refer to the language differences page and review the BASIC Stamp I to BASIC Stamp II Conversion document on our web site: http://www.parallaxinc.com/stamps/langdiff. There is also a comparison chart at the end of this document.

Is the BASIC Stamp II better than the BASIC Stamp I?

Yes and no, depending on your application. The BASIC Stamp II has twice the number of I/O pins, twice the execution speed, 5 times the memory (code) space, a serial port and 5 times the resolution on time sensitive commands. In many cases, the BASIC Stamp II fits applications better than the BASIC Stamp I. In some cases, however, this is not true. For example, you may have a need for only a couple of inputs and outputs with which you need to perform relatively simple, non speed-critical tasks. This is a perfect application for the BASIC Stamp I; the BASIC Stamp II would be overkill if used in this way. A slightly different example would be if you have a variable pulse width signal you need to monitor. The BASIC Stamp I can detect and measure a pulse as little as 10 µS wide and as long as 0.65535 seconds wide. The BASIC Stamp II has 5 times the resolution, so it can measure a pulse as little as 2 µS wide, however, only as long as 0.13107 seconds wide. While the better resolution may be helpful, the smaller maximum pulse width measuring capability may prove disadvantageous to your application.

Is the BASIC Stamp IIsx better than the BASIC Stamp II?

Yes and no, depending on your application. The BASIC Stamp IIsx executes instructions 2.5 times as fast, has 8 times the memory (code) space and 2.5 times the resolution on time sensitive commands. In some cases, the BASIC Stamp IIsx fits applications better than the BASIC Stamp II. For example, you may have a need for additional program space that doesn't exist in the Stamp II. Another example would be if you have a variable pulse width signal you need to monitor. The BASIC Stamp II can detect and measure a pulse as little as 2 µS wide and as long as 0.13107 seconds wide. The BASIC Stamp IIsx has 2.5 times the resolution, so it can measure a pulse as little as 800 nS wide, however, only as long as 0.05243 seconds wide. While the better resolution may be helpful, the smaller maximum pulse width measuring capability may prove disadvantageous to your application. A full comparison of the Stamps can be found at the end of this document.

How many I/O pins does the BASIC Stamp have?

The BASIC Stamp I has 8 I/O pins while the BASIC Stamp II, BASIC STAMP IIe and BASIC Stamp IIsx has 16 I/O pins plus 2 special serial port pins (1 input, 1 output).

What can I use the I/O pins for?

The BASIC Stamp's I/O pins are perfectly suited for digital input and output with TTL/CMOS level (0 to 5 volt) signals. However, you can use some special commands and techniques to input and output limited analog signals. For example, the RCTIME and PWM commands can be used to read a variable resistance or output a variable voltage from 0 to 5 volts. The I/O pins can not be used to read analog voltages by themselves, however, but this can be done by interfacing the I/O pins to an A to D converter chip.

Can the I/O pins be used to control relays, solenoids and other similar devices?

Yes, however, due to the demanding current and voltage requirements of some of these components, driver circuitry will need to be used to properly isolate the I/O pins from harmful effects. For examples of this, refer to BASIC Stamp I Application Note #6 and BASIC Stamp Article #6, "Silicon Steroids for Stamps".

Can I control LEDs with the I/O pins?

Yes. Simply use a 470 ohm resister in series with the LED to limit the current draw through the I/O pin (see "How much current can the I/O pins handle?"). Also, keep in mind that most LEDs require a lot of current in relation to what the BASIC Stamp can provide. If you attach and power 3 or more LEDs at one time from the BASIC Stamp's I/O pins, you are likely to see flaky and unpredictable results caused by voltage sag, I/O pin damage and/or hardware resets. Driver circuitry or low current LEDs will need to be used if you require such an application.

How much current can the I/O pins handle?

On the BASIC Stamp I and BASIC Stamp II, each I/O pin is capable of sourcing 20 mA and sinking 25 mA. The total across all I/O pins should not exceed 40 mA source or 50 mA sink at any given time, however. One exception to this rule exists with the BS2-IC. If the BS2-IC is powered by an external 5-volt regulator capable of delivering at least 100 mA, the total across each group of 8 I/O pins (0 - 7 and 8 - 15) can reach 40 mA source or 50 mA sink providing a total of 80 mA source or 100 mA sink overall. The BASIC Stamp IIsx's I/O pins can source and sink 30 mA, and the total across all I/O pins should not exceed 150 mA.

What is the input impedance of the BASIC Stamp's I/O pins?

Approximately 1 Meg Ohm.

How fast does the BASIC Stamp execute its PBASIC code?

On average, the BASIC Stamp I executes 2000 instructions per second, the BASIC Stamp II executes 4000 instructions per second, the BASIC Stamp IIe executes apx 3800 instructions per second and the BASIC Stamp IIsx executes 10000 instructions per second. This is only an estimate as the true execution speed depends upon many factors including, the particular set of instructions used and the type of arguments provided to the instructions. Some instructions execute at a much faster rate while others, especially those that have to deal with external signal measurement or specific protocol speeds, will execute much slower.

What is the VIN pin used for?

The VIN (Voltage Input) pin is used to power the BASIC Stamp from a 6 to 15 volt source. The VIN pin is the positive connection while the VSS pin is the negative, or ground, connection. When powered from the VIN pin the BASIC Stamp regulates the voltage down and outputs +5 volts on the VDD pin. For optimal operation, it is recommended to keep the voltage below 12 volts for the BASIC Stamp I and II, IIe, and around 7.5 volts for the BASIC Stamp IIsx.

How does the VDD pin work?

The VDD pin (+5V) outputs 5 volts when the BASIC Stamp is powered by a 6 to 15 volt source using the VIN and VSS (ground) pins. The VDD pin can be used to power other circuitry if the overall current consumption is within the capabilities of the BASIC Stamp's regulator. The VDD pin can also be used to power the BASIC Stamp from an external 5-volt regulator. In this case, the VDD pin is the +5 volt connection and the VSS (ground) pin is the negative connection of the regulator.

How does the reset pin (RES) work?

The reset pin is internally controlled by the BASIC Stamp's brownout detector. It is normally high (+5V), which allows the BASIC Stamp to run its program normally, and is pulled low when the power supply voltage drops below 4 volts (safely putting the BASIC Stamp to sleep). This pin can be monitored to detect when a reset condition occurs, or, you can pull the line to ground to force a reset. After the reset pin is allowed to rise to +5 volts, the BASIC Stamp wakes up and starts executing its program from the first line of code. Do not drive this pin high, it should be left electrically disconnected (floating) when you want the BASIC Stamp to run normally. This pin has an inverse relationship to the ATN (Attention) pin on the BASIC Stamp II, BASIC STAMP IIe and BASIC Stamp IIsx.

How does the ATN (Attention) pin work (BASIC Stamp II, BASIC STAMP IIe and BASIC Stamp IIsx)?

The ATN pin is used strictly for programming the BASIC Stamp II, BASIC STAMP IIe, or BASIC Stamp IIsx, and should not be connected to anything other than the DTR pin of the programming connector and/or filtering capacitors. The ATN pin is an input and has an inverse relationship with the reset (RES) pin. When the ATN pin is left unconnected, or is driven low, it causes the RES pin to go high (allowing the BASIC Stamp II, IIe or IIsx to function properly). When the ATN pin is driven high it causes the RES pin to go low (performing a hardware reset on the BASIC Stamp II, IIe or IIsx). The BASIC Stamp editor software toggles this pin from low to high and then low again at the start of the programming process to perform a hardware reset on the module and begin the bi-directional programming communication. It is important to be aware of a particular artifact of the DTR pin on a standard serial port: upon port selection by standard software (terminal packages, development tools, etc.), the DTR pin is driven high by default. If the same connections for programming the BASIC Stamp II, IIe or IIsx are used to communicate with it via a serial port, the default state of the DTR pin will cause the BASIC Stamp to be held in permanent reset and its PBASIC program will not execute. (This is the case on the Stamp II Carrier Board). This problem can be fixed in a number of ways: 1) disconnect the DTR to ATN pin connection when trying to communicate serially to the BASIC Stamp, or 2) explicitly set the DTR pin to low after selecting the port if the terminal package or development tool allows this, or 3) modify the programming connection to include a 0.1 Fu capacitor in series with the DTR to ATN pin connection and a 0.1 Fu capacitor from ATN to ground (VSS). The last two options will allow you to use the same connection for both programming and serial communications. (The Super Carrier Board, BASIC Stamp Activity Board and Board of Education use the last method).

How much current can the BASIC Stamp's on-board regulator provide?

The built in 5-volt regulator on the BASIC Stamp I and BASIC Stamp II can supply 50 mA of current if powered by a 12-volt source. The BASIC Stamp IIsx's regulator can supply 150 mA if powered by a 7.5-volt source. The BASIC Stamp I consumes 2 mA in running mode and the BASIC Stamp II consumes 7 mA. This leaves 48 mA and 43 mA respectively for use with other circuitry via the VDD pin and the I/O pins. The BASIC Stamp IIsx consumes 65 mA, leaving 85 mA for use with other circuitry via the VDD pin and the I/O pins.

Can the BASIC Stamp generate a sine wave?

Only the BASIC Stamp II, BASIC STAMP IIe and BASIC Stamp IIsx can generate sine waves. The BASIC Stamp II uses the FREQOUT command to generate from 0 Hz to 32,767 Hz. The BASIC Stamp IIsx uses the FREQOUT command to generate from 0 Hz to 81,917 Hz. Note that the output from the FREQOUT command appears to be static when viewed on an oscilloscope. This is due to the nature of the underlying algorithm that generates the frequency, however, when averaged out over time, the result is a sine wave. You may achieve a cleaner signal by attaching the I/O pin to an appropriate RC filter circuit.

Can the BASIC Stamp generate a sine wave on two pins at once?

No. The BASIC Stamp is a single-task device and thus will execute only one FREQOUT command at a time.

Can the BASIC Stamp generate more than one sine wave on a pin at the same time?

Yes, the BASIC Stamp II, IIe and IIsx's FREQOUT command can generate two frequencies at the same time, on the same pin. This can be used to create harmonics, which greatly improve the sound output.

Can the BASIC Stamp generate square waves?

Yes, however, it is limited. The SOUND command in the BASIC Stamp I generates square waves which vary from 94.8 Hz to 10,550 Hz (non-linearly). The BASIC Stamp II, IIe and IIsx, however, does not include any functions to generate an accurately timed square wave, however, it can be done manually as long as fast timing is not an issue. This can be done with multiple lines of code that simply set a pin high or low and pause for the appropriate duration in-between transitions. The timing will vary somewhat depending upon your loop execution speeds.

Does the BASIC Stamp include a real time clock function?

No. If you need to keep track of time, especially dates and time, it is best to interface a real time clock to the BASIC Stamp. Many are available from various manufacturers. The real time clocks with a serial interface are usually the best choice, as they require fewer I/O pins from the BASIC Stamp to function properly. Parallax sells the DS1302 Application Kit to address this issue.

Does the BASIC Stamp have a built-in timer?

Yes, however, it is used for timing functions within the BASIC Stamp interpreter and there is no direct access to it through any PBASIC commands. Refer to BASIC Stamp I Application Note #20, "An Accurate Timebase" for one possible solution to timing problems.

Does the BASIC Stamp support interrupts?

No, the interpreter chip used in the BASIC Stamp does not support interrupts. In many cases, a fast polling routine may be used to accomplish the same effect, however, depending on the number and size of the tasks involved in some applications, this will not be fast enough and the BASIC Stamp may not be a plausible solution.

Does the BASIC Stamp II, BASIC STAMP IIe or BASIC Stamp IIsx have the same bsave/bsload feature available in the BASIC Stamp I?

No. It is intended to provide this feature in the future.

How do I program the BASIC Stamp?

You can write your PBASIC program using the BASIC Stamp I, II, IIe or IIsx editor on a standard IBM compatible PC. Both DOS and Windows versions are available for the BASIC Stamp II, IIe and IIsx. After you write the code for your application, you simply connect the BASIC Stamp to the computer's parallel port (BASIC Stamp I) or serial port (BASIC Stamp II, IIe and IIsx), provide power to the BASIC Stamp and press ALT-R (DOS version) or CTRL-R (Windows version) within the appropriate BASIC Stamp editor to download your program into the BASIC Stamp's EEPROM. As soon as the program has been downloaded successfully, it begins executing its new program from the first line of code.

How do I connect the BASIC Stamp to my computer for programming?

The BASIC Stamp I (rev. D or BS1-IC) requires a three-wire connection to any available parallel port on your PC. The BASIC Stamp II, IIe and IIsx (BS2-IC and BS2SX-IC) require a standard, straight-through, serial cable connection to a 9-pin or 25-pin serial port on your PC. Only 6 wires are used for programming the BASIC Stamp II, IIe or IIsx, 4 of which are absolutely required. You can not use a null-modem cable to program the BASIC Stamp II, IIe or IIsx. Refer to the schematics on our web site for the details of the cable connections, http://www.parallaxinc.com/stamps/stmpsche.htm.

How does my program get stored in the BASIC Stamp?

Your PBASIC code is tokenized (compressed) and stored within the Stamp's EEPROM memory. This memory is non-volatile, meaning it retains its program even without power. Every time the BASIC Stamp receives power, it starts running its PBASIC code starting with the first executable line. This EEPROM can be rewritten immediately, without any lengthy erase cycle or procedure, by simply re-downloading the code from the BASIC Stamp editor again. Each location in the EEPROM is guaranteed for 10,000,000 write cycles before it wears out. The code is tokenized before downloading. Source code elements like comments and constant and variable names are not stored in the BASIC Stamp, thus you may feel free to use as many comments and descriptive symbol names in your code as you like without worrying about increasing your code size.

How do I erase the BASIC Stamp's program space?

An erase cycle is performed at the beginning of every programming process (in the BASIC Stamp I) or during the programming process (in the BASIC Stamp II, IIe and IIsx), thus, by downloading from the BASIC Stamp editor the necessary areas of the EEPROM are erased and then reprogrammed with the current PBASIC program in the editor. The BASIC Stamp II, IIe and IIsx only erase data in 16-byte increments. It is possible to completely erase the EEPROM by downloading the following line of code (in a program by itself):

```
DATA 0(2048)
```

How do I reprogram the BASIC Stamp?

Simply re-connect it to the computer, run the BASIC Stamp editor and press ALT-R (DOS software) or CTRL-R (Windows software).

How big of a program can I store in the BASIC Stamp?

The BASIC Stamp I has 256 bytes of program storage; enough for 80 to 100 lines of PBASIC1 code. The BASIC Stamp II has 2048 bytes of program storage; enough for 500 to 600 lines of PBASIC2 code. The BASIC Stamps IIe and IIsx each have 16,384 bytes of program storage separated into 8 pages of 2048 bytes. Each page (or code section) can hold 500 to 600 lines of PBASIC2e, or PBASIC2sx code, respectively, for an overall total of approx. 4000 instructions.

Can I expand the program memory?

No, the program memory is not expandable as the interpreter chip expects the memory to be a specific, fixed size.

Can I expand the data memory?

Yes, you may interface EEPROMs, or other memory devices, the BASIC Stamp's I/O pins to gain more data storage area. You will have to include the appropriate code within your PBASIC program to interface to the particular device you choose.

How difficult is it to program the BASIC Stamp?

When compared to other programmable microcontrollers, the BASIC Stamps are perhaps the easiest to use because of their simple, but powerful, language structure and straightforward method of downloading and debugging. If you have some experience programming in BASIC, C or Pascal, you should find the learning process with the BASIC Stamp to be a simple one. If, however, you have never had any programming experience, you may have to spend some extra time studying the examples and application notes before you feel comfortable with the BASIC Stamp.

Can I program the BASIC Stamp in Visual BASIC or QBASIC?

No. The BASIC Stamp is a microcontroller, not a miniature PC. It does not have a graphical user interface, a hard drive or a lot of RAM. The BASIC Stamp must only be programmed with PBASIC, which has been specifically designed to exploit all of the BASIC Stamp's capabilities.

What is PBASIC?

PBASIC (Parallax BASIC) is a hybrid form of the BASIC programming language in which many people are familiar. Currently there are three versions of PBASIC: PBASIC1 for the BASIC Stamp I, PBASIC2 for the BASIC Stamp II and PBASIC2sx for the BASIC Stamp IIsx. Each version is specifically tailored to take advantage of the inherent features of the hardware it runs on. PBASIC is called a hybrid because, while it contains some simplified forms of standard BASIC commands, it also has special commands to efficiently control the I/O pins.

Can I imbed assembly language routines in my PBASIC code?

No. The BASIC Stamp is a PBASIC interpreter only and cannot except assembly language routines.

How much space does each PBASIC command take?

Each command takes a variable amount of space within the BASIC Stamp's EEPROM. It varies due to complexities of the command itself and the types of arguments you supply each command. Generally, each command takes 2 to 4 bytes of memory space, however, commands like SERIN, SEROUT, LOOKUP and LOOKDOWN, which accept a variable length list of arguments, may take tens of bytes or more.

How do I know how much space my PBASIC program consumes?

On the BASIC Stamp I, enter the following code at the start of your PBASIC1 program:

```
READ 255,B0
DEBUG #B0
```

Upon running the program, a number will display in the debug window of the editor. Use the following equation to determine how many bytes are used by your PBASIC1 code: 255 - # - 6; where # is the number displayed on the debug window. Note, the "- 6" in the equation results from the fact that the above two lines of code take 6 bytes of program space, thus without those two lines, your program takes 6 fewer bytes of space.

On the BASIC Stamp II, IIe or IIsx, from within the editor and with your PBASIC program on the screen, press ALT-M (DOS software) or CTRL-M (Windows software) to view the memory map screens. In the DOS software, the first screen shows your RAM space, the second screen shows an expanded view of the program (EEPROM) space and the third screen shows a detailed view of the program space. Press the spacebar to alternate between the three screens. In the Windows software, all EEPROM and RAM is shown on one window.

Why doesn't the Memory Map feature show me data I have already stored in the BASIC Stamp II, IIe or IIsx's EEPROM?

The memory map feature (ALT-M, DOS software or CTRL-M, Windows software) shows only what the memory space will look like immediately after you program the Stamp II, IIe or IIsx. The editor determines this by examining your PBASIC source code within the editor, not the PBASIC code in the Stamp. It does not read the BASIC Stamp's EEPROM. It will only display data in the EEPROM that you've explicitly defined with DATA commands; no data which is operated upon by READ and WRITE commands can be shown since those are run-time, rather than compile-time, functions.

How do I debug my PBASIC programs?

You may use two features of the BASIC Stamp editor to debug your PBASIC program: 1) Syntax checking, and 2) the DEBUG command. Syntax checking alerts you to any syntactical errors and is automatically performed on your code the moment you try to download to the BASIC Stamp. Any syntax errors will cause the download process to abort and the editor will display an error message with a highlight cursor pointing to the source of the error in your code. Logical errors are sometimes more difficult to find but may be more easily discovered by using the DEBUG command. Debug operates similar to the PRINT command in the BASIC language and can be used to print the current status of specific variables within your PBASIC program as it is executed within the BASIC stamp. If your PBASIC code includes a DEBUG command, the editor opens up a special window at the end of the download process to display the results for you. In the Windows editor, the Debug window (called the Debug Terminal) can be left open, while you edit your code.

How do I print out my PBASIC programs?

The BASIC Stamp DOS editors do not have a print option built-in, however, the source file created by the editor is simply a DOS text file. This means it can be printed with the standard PRINT or TYPE commands as in:

```
PRINT source.bas

    -or-

TYPE  source.bas > LPT1
```

where source.bas is the name of your PBASIC program. The Windows software features built-in printing support.

Can I read-out, or upload, the PBASIC program which is already stored in the BASIC Stamp?

No. For security reasons, this feature was not made available. It is possible to read the tokenized form of the PBASIC code out of the BASIC Stamp's EEPROM, but there is no easy way to "reassemble" it into usable source code.

Since the BASIC Stamp II, IIe and IIsx use the serial port for programming, can I use a simple terminal program to download my PBASIC code?

No. The download process requires a very fast, bi-directional communication between the BASIC Stamp and PC to program and verify the contents of the BASIC Stamp's memory.

Will a program and circuitry designed for the BASIC Stamp I work with the BASIC Stamp II, IIe or IIsx?

Yes, however, some changes, usually in the PBASIC code only, will have to be made for it to run properly or to take advantage of some of the new features of the BASIC Stamp II, IIe or IIsx. For help in performing these modifications or for insight into how the PBASIC1 language differs from PBASIC2, review Appendix C in the BASIC Stamp Manual.

Can I program the BASIC Stamp II, IIe or IIsx with the BASIC Stamp I editor (stamp.exe)?

No. The BASIC Stamp II, IIe and IIsx requires the BASIC Stamp II, IIe and IIsx editor (stamp2.exe, stamp2e.exe or stamp2sx.exe in DOS, stampw.exe in Windows) due to the enhancements in the PBASIC2 language and the serial, rather than parallel, programming interface. Similarly, you cannot program the BASIC Stamp I with the BASIC Stamp II, IIe or IIsx editors.

How do I make my PBASIC program start over or continue running forever?

All BASIC programs will simply end if one or more of their executable paths don't lead them to a continuous loop. In the simplest case, a program can be made to continue endlessly by adding a label (a unique name followed by a colon ':') to the start of code and adding a GOTO statement at the end of code directing logical execution back to that label. Without this, the BASIC Stamp will run the program only once and then will remain unresponsive until the power is cycled or a reset condition is created.

How do I set an I/O pin to input or output mode?

There are many ways to set the I/O pin directions. The most readable and intuitive method is to use the INPUT and OUTPUT commands. For example:

```
INPUT 5
OUTPUT 6
```

will set I/O pin 5 to an input and I/O pin 6 to an output. By default, upon reset or startup, all I/O pins are inputs, thus only the second line of code above is really necessary.

Another method for setting the pin directions is to use the predefined variable DIRS or any one of its derivatives. The following code is functionally equivalent to the first example:

```
DIR5 = 0
DIR6 = 1
```

DIR0 through DIR15 (or DIR7 on the BASIC Stamp I) are bit variables which control the direction of the corresponding I/O pin. DIRS is a 16-bit variable (or 8-bit variable on the BASIC Stamp I) in which each bit represents the direction of the corresponding I/O pin. A ‘1’ bit means output and a ‘0’ bit means input. The advantage of this method of declaring pin directions is that multiple pin directions can be defined at once:

```
DIRS = %00101110
```

The above line of code defines I/O pins 7, 6, 4 and 0 as inputs and 5, 3, 2 and 1 as outputs. Review the BASIC Stamp manual for more examples of this.

How do I make an I/O pin output a high or a low?

As with setting I/O pin directions, there are many ways to set the I/O pin states. The most readable and intuitive method is to use the HIGH and LOW commands. For example:

```
HIGH 5
LOW 6
```

will first set I/O pin 5 to output mode and then set it to a high (+5V) state. The second line will first set I/O pin 6 to output mode and then set it to a low (0V) state. Notice that these commands automatically set the pin direction for you; it's not necessary to manually set the pins to output when using the HIGH and LOW commands. Keep in mind that if a pin is set to input, its logical state, high or low, remains the same internally within the BASIC Stamp but will no longer affect the voltage on the I/O pin.

Another method for setting the pin state is to use the predefined variable OUTS (or PINS in the BASIC Stamp I) or any one of its derivatives. The following code is functionally equivalent to the first example:

```
DIR5 = 1
DIR6 = 1
OUT5 = 1
OUT6 = 0
```

Notice here that we had to explicitly define the I/O pins as outputs first. If we had left the pins as inputs, the OUT5 and OUT6 lines would have changed the state of the pins internally, but those changes would not appear externally from the BASIC Stamp. OUT0 through OUT15 (or PIN0 through PIN7 on the BASIC Stamp I) are bit variables which control the logical state of the corresponding I/O pin. OUTS is a 16-bit variable (PINS is an 8-bit variable on the BASIC Stamp I) in which each bit represents the state of the corresponding I/O pin. A '1' bit means high and a '0' bit means low. The advantage of this method of declaring pin states is that multiple pin states can be defined at once:

```
DIRS = %00101110
OUTS = %00001010
```

The above line of code defines I/O pins 7, 6, 4 and 0 as inputs and 5, 3, 2 and 1 as outputs. And sets I/O pins 7, 6, 5, 4, 2 and 0 to logical 0's and I/O pins 3 and 1 to logical 1's. Only pins 5, 3, 2 and 1 will output the voltage set in the OUTS variable because they are the only pins defined as outputs by the DIRS variable. Review the BASIC Stamp manual for more examples of this.

How do I input or output data in parallel on the I/O pins?

All BASIC Stamps have predefined variables that refer to the I/O pins. On the BASIC Stamp I, the PINS variable refers to all eight I/O pins. Thus code such as:

```
SYMBOL  Snapshot = B0
Snapshot = PINS
```

would copy the current state of all I/O pins (an 8-bit byte) into the variable Snapshot. Similarly:

```
DIRS  = %11111111  'Set all eight pins as outputs
PINS = 150
```

would set the eight output pins to the binary form of 150, or %10010110.

On the BASIC Stamp II, IIe and IIsx, the PINS variable was replaced with the INS and OUTS variables. Both INS and OUTS refer to all sixteen pins, while their derivatives, INL, INH, OUTL and OUTH refer to the low and high bytes (groups of 8 pins). For example, the following code will input 8-bits in parallel from the lower 8 pins:

```
Snapshot      VAR      BYTE
Snapshot = INL
```

The following code will output 8-bits in parallel from the upper 8 pins:

```
DIRS = %1111111100000000 'Set upper 8 pins to outputs
OUTH = 255
```

Why do the BASIC Stamp's output pins toggle briefly when I use the SLEEP, NAP or END commands or allow my program to end naturally?

Inside the BASIC Stamp's interpreter chip is a watchdog timer whose main purpose is to reset the interpreter chip if, for some reason, it should stop functioning properly. The SLEEP and NAP commands also utilize the watchdog timer to periodically (every 2.3 seconds to be exact) "wake-up" the BASIC Stamp from its low-power mode. Upon reset, the I/O pins are set to inputs for approximately 18 mS before returning to their previous directions and states. If you have an output pin set to a logical 1 state (+5V) and you use the SLEEP command, every 2.3 seconds during sleep mode that I/O pin will switch to an input for 18 mS causing a momentary signal loss. This "power glitch" is easily viewable with an LED and a 470 ohm resister tied to an I/O pin and switched on just before entering sleep mode. In many cases this problem can be remedied by tying a pull-up or pull-down resistor to the I/O pin in question to provide a constant source of power should the I/O pin change directions. Allowing a PBASIC program to end naturally, or using the END command, will exhibit the same "power glitch" behavior because the interpreter chip enters a low-power state. The BASIC Stamp II, IIe and IIsx includes the STOP command which acts just like END, except the output pins remain driven.

The LET command is not available on the BASIC Stamp II, IIe and IIsx. Does this mean the BASIC Stamp II, IIe and IIsx cannot perform mathematical operations?

No. The LET command on the BASIC Stamp I was optional; i.e.: you could leave it out of the variable assignment statement or equation and it would work normally. To save some space in the interpreter code of the BASIC Stamp II, IIe and IIsx, the LET command was removed completely, however, the mathematical ability was not removed and, in fact, it is greatly improved on the BASIC Stamp II, IIe and IIsx.

Can the BASIC Stamp store data, such as temperature readings, for review or use at a later time?

Yes. The READ and WRITE commands allow the BASIC Stamp to manipulate memory locations in its EEPROM. Any space in the built-in EEPROM that is not occupied by PBASIC code may be used for data storage and is just as non-volatile as the PBASIC code itself. You may also attach external EEPROMs of various sizes to gain an even greater data storage area. The READ and WRITE commands will not function on external EEPROMs so you must write the appropriate code to interface with the device.

How can I store a word value into the internal EEPROM?

The internal EEPROM of the BASIC Stamp (used for both program and data storage) is organized and accessed in units of bytes only. Since a word value is actually two bytes it requires two consecutive memory locations to store it in EEPROM. You must break the word value into its lower and upper byte parts and use two WRITE commands to store the value, or two READ commands to retrieve it.

The following code demonstrates this for the BASIC Stamp I:

```
W0 = 1250
WRITE 0, B0
WRITE 1, B1
```

The following code demonstrates this for the BASIC Stamp II:

```
Temp VAR WORD

Temp = 1250
WRITE 0, Temp.LOWBYTE
WRITE 1, Temp.HIGHBYTE
```

How does the fpin (flow control pin) work in the SERIN and SEROUT commands on the BASIC Stamp II, IIe and IIsx?

The optional fpin argument in the SERIN and SEROUT commands allows the use of an I/O pin as a hardware flow control line between two BASIC Stamp II, IIe and IIsx's at baud rates of up to 19.2 kBps. The flow control pin is always controlled by the receiving device (SERIN) and monitored by the sending device (SEROUT). The logical state of this pin and its meaning depend on the data mode, true or inverted, as specified within the baudmode parameter. If true data mode is selected, the receiving BASIC Stamp II, IIe or IIsx holds the fpin low to indicate it is ready to receive or high to indicate it is not ready to receive. If inverted data mode is selected, the receiving BASIC Stamp II, IIe or IIsx holds the fpin high to indicate it is ready to receive or low to indicate it is not ready to receive.

Assuming true data mode: the fpin is set to output high the moment the SERIN command is first encountered and remains high until the SERIN command is complete at which point the fpin is set to low and remains there until the SERIN command is executed again. The transmitting BASIC Stamp II, IIe or IIsx's SEROUT command monitors this line and halts transmission whenever a low on the fpin is detected.

How are arithmetic expressions evaluated within the BASIC Stamp?

All expressions are evaluated using 16-bit, integer arithmetic. Even if byte and bit variables are used in expressions they are first expanded to 16-bits and then the expression is evaluated.

In the BASIC Stamp I, mathematical expressions are evaluated strictly from left to right. No order-of-precedence is utilized, parentheses are not allowed and expressions must be on a line by themselves; i.e.: they can not be used as an argument for a command. For example, the expression: W0 = 10 - 2 * 3 results in the value 24 being stored into W0 (not the value 4 as you might think). To evaluate this expression properly, specify it as: W0 = 0 - 2 * 3 + 10.

In the BASIC Stamp II, IIe and IIsx, mathematical expressions are evaluated from left to right, as in the BASIC Stamp I. No order-of-precedence is utilized, however, unlike the BASIC Stamp I, parentheses are allowed and expressions may be used as arguments within commands. For example, the expression: Answer = 10 - 2 * 3 results in the value 24 being stored into Answer, however: Answer = 10 - (2 * 3) results in the value 4.

Does the BASIC Stamp handle signed numbers and arithmetic?

Yes. The BASIC Stamp uses twos-compliment notation for signed numbers. This means that the expression: 0 - 10 + 5 will result in -5 if viewed as a signed number, however, most instructions see the number as a positive value, in this case 65531 (the twos-compliment value for -5). All mathematical operators, except division, will work properly with signed numbers in the BASIC Stamp and signed numbers can be formatted for output properly using the SDEC, SHEX and SBIN formatters within DEBUG and SEROUT statements on the BASIC Stamp II, IIe and IIsx. Be careful how you use signed numbers elsewhere. For example, if the value -5 is stored in a variable called Temp, and you use the following statement:

```
IF Temp < 0 THEN Loop
```

it will evaluate to false and will not branch to Loop because -5 is actually 65531 in twos-compliment form and thus 65531 is not less than 0.

How can I define an alias to an I/O pin or another variable?

In the BASIC Stamp I you could specify the following:

```
SYMBOL Counter      =  B0
SYMBOL Index        =  B0
SYMBOL LED          =  PIN0
```

to designate the symbol Index as an alias to the Counter variable and the symbol LED as an alias to I/O pin 0. Since Counter and Index use the same register, B0, anytime Counter is changed, Index will change in the same way. The symbol LED will always contain either a 1 or a 0 depending on the logical state of I/O pin 0.

In the BASIC Stamp II, IIe or IIsx you could specify the following:

```
Counter    VAR   BYTE
Index      VAR   Counter
LED        VAR   IN0
```

to designate the symbol Index as an alias to the Counter variable and the symbol LED as an alias to I/O pin 0. Since Index uses the same memory space as Counter, anytime Counter is changed, Index will change in the same way. The symbol LED will always contain either a 1 or a 0 depending on the logical state of I/O pin 0.

How do I reference a specific bit within a byte or word variable?

On the BASIC Stamp I there is only one general purpose word register, W0, and two general purpose byte registers, B0 and B1, which are bit addressable. The predefined symbols Bit0 through Bit15 refer to the corresponding bits within W0 as well as B0 and B1 since those two byte variables correspond to the lower and upper bytes of W0 respectively. Thus Bit0 is the lowest bit of both W0 and B0 while Bit8 is the lowest bit of B1 (and the 9th bit of W0).

On the BASIC Stamp II, IIe and IIsx all the variables are bit addressable, as well as nibble addressable. You can reference a specific part of a variable by specifying the variable name followed by a dot '.' and then the name of the portion you're interested in. For example, bit 2 of a word variable called Temp can be referenced with the notation: Temp.BIT2. Additionally, the second nibble of Temp can be referenced with: Temp.NIB1. The valid modifiers are: BIT0 through BIT15, LOWBIT, HIGHBIT, NIB0 through NIB3, LOWNIB, HIGHNIB, BYTE0 through BYTE3, LOWBYTE and HIGHBYTE.

How do I define a string variable?

A string variable is simply an array of bytes. Only the BASIC Stamp II, IIe and IIsx allows the explicit definition of arrays. If, for example, you want to define a string variable called Text to hold 6 characters, use the following code:

```
Text  VAR     BYTE(6)
```

The elements of an array can be accessed by specifying the array name followed by an open parenthesis, an index value and a close parenthesis. The first element is always index 0. For example, the second element, or character in this case, can be referenced with: Text(1) and the last element with: Text(5). Byte arrays are useful for receiving a string of characters via the SERIN command using the STR modifier. The BASIC Stamp II, IIe and IIsx allows you to define arrays of bits, nibbles and words as well.

How do I define aliases to specific bytes within a word array on the BASIC Stamp II, IIe and IIsx?

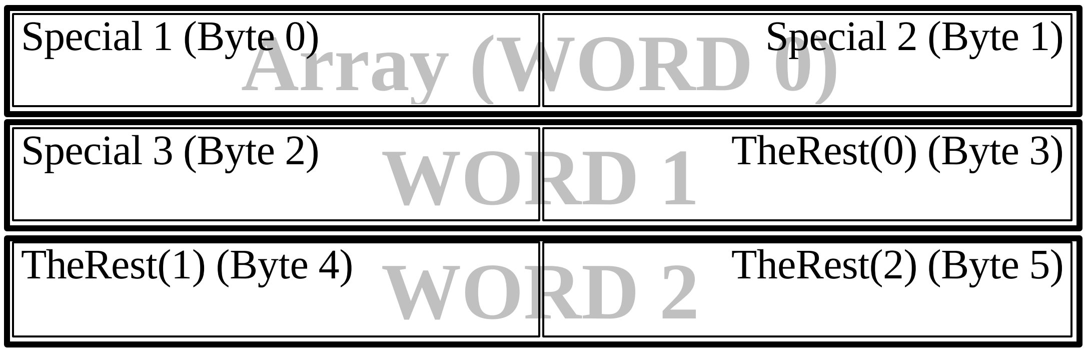
The BASIC Stamp II, IIe and IIsx uses a very relaxed and flexible indexing scheme when it comes to arrays of RAM space. When trying to define aliases in this way it is best to view the entire RAM space as one array of word, byte, nibble and bit registers. The BASIC Stamp II, IIe and IIsx organizes variables in the RAM space from largest entities (word declarations) to smallest entities (bit declarations), regardless of the order you define them, to be most efficient. The relaxed nature of indexing allows a user to cross array boundaries at will, or by accident, quite easily. For example, the following code defines two distinct byte variables:

```
Result          VAR    BYTE
Temp            VAR    BYTE
```

Normally you would reference the Result and Temp variables by name, as in: Result = 10. Even though this variable is not an array, you can still access it as such. The code: Result(0) = 10 is equivalent to the previous example. The most interesting, and powerful, feature is that you can index beyond the boundaries of this byte variable, as in Result(1) = 20. This statement actually modifies the byte of RAM immediately following the Result variable. The next byte happens to have been defined as Temp, in this case, thus the variable Temp will now equal 20. By carefully defining variables we can take advantage of this flexible feature to define byte sized aliases to specific locations within a word array. Suppose we have the need for an array of three words in which the first three bytes have a special meaning; we'd like to be able to easily manipulate those bytes as well as access the three distinct word elements.

```
Array                  VAR  WORD
Special1               VAR  Array.LOWBYTE
Special2               VAR  Array.HIGHBYTE
Special3               VAR  BYTE
TheRest                VAR  BYTE(3)
```

Remember: the BASIC Stamp II, IIe and IIsx will always organize RAM space from biggest elements to smallest. In the example above we defined a single word variable called array (this becomes assigned to the very first word in RAM space, word 0). The next two definitions create aliases to the first and second bytes of Array (no additional RAM space is allocated here since we're pointing at previously allocated space, byte 0 and byte 1). Next we define a single byte variable (this becomes assigned to the first byte of RAM immediately following the Array variable, byte 2). Finally, we define an array of three bytes (which are assigned to the next three bytes of RAM immediately following the previous byte declaration, byte 3, byte 4 and byte 5). Visually, our RAM space now appears like this:



Now, Array(0), Array(1) and Array(2) refer to the three words of our array and Special1, Special2 and Special3 refer to the first three bytes within our three byte array.

Stamp Specifications (revised 6/00)

Released Products	Rev.D / BS1-IC	BS2-IC	BS2e-IC	BS2SX-IC
Package	PCB w/Proto / 14-pin SIP	24-pin DIP	24-pin DIP	24-pin DIP
Package Size (L x W x H)	2.5" x 1.5" x .5" / 1.4" x .6" x .1"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"
Environment	0º - 70º C* (32º - 158º F) **	0º - 70º C* (32º - 158º F) **	0º - 70º C* (32º - 158º F) **	0º - 70º C* (32º - 158º F) **
Microcontroller	Microchip PIC16C56	Microchip PIC16C57	Scenix SX28AC	Scenix SX28AC
Processor Speed	4 MHz	20 MHz	20 MHz	50 MHz
Program Execution Speed	~2,000 instructions/sec.	~4,000 instructions/sec.	~4,000 instructions/sec.	~10,000 instructions/sec.
RAM Size	16 Bytes (2 I/O, 14 Variable)	32 Bytes (6 I/O, 26 Variable)	32 Bytes (6 I/O, 26 Variable)	32 Bytes (6 I/O, 26 Variable)
Scratch Pad RAM	N/A	N/A	64 Bytes	64 Bytes
EEPROM (Program) Size	256 Bytes, ~80 instructions	2K Bytes, ~500 instructions	8 x 2K Bytes, ~4,000 inst.	8 x 2K Bytes, ~4,000 inst.
Number of I/O pins	8	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial
Voltage Requirements	5 - 15 vdc	5 - 15 vdc	5 - 12 vdc	5 - 12 vdc
Current Draw @ 5V	2mA Run / 20µA Sleep	8 mA Run / 100 µA Sleep	20 mA Run / 100 µA Sleep	60 mA Run / 200 µA Sleep
Source / Sink Current per I/O	20 mA / 25 mA	20 mA / 25 mA	30 mA / 30 mA	30 mA / 30 mA
Source / Sink Current per unit	40 mA / 50 mA	40 mA / 50 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins
PBASIC Commands	32	36	39	39
PC Programming Interface	Parallel Port	Serial Port (9600 baud)	Serial Port (9600 baud)	Serial Port (9600 baud)
DOS Text Editor	STAMP.EXE	STAMP2.EXE	STAMP2E.EXE	STAMP2SX.EXE
Windows Text Editor	N/A	Stampw.exe (v1.04 and up)	Stampw.exe (v1.096 and up)	Stampw.exe (v1.091 and up)

* Industrial Models Available - Environment is -40º - 85º C (-40º - 185º F)** ** 70% Non-Condensing Humidity

BASIC Stamp I and Stamp II Conversions

BASIC Stamp I and Stamp II Conversions

BASIC Stamp I and Stamp II Conversions

Appendix C

INTRODUCTION

The BASIC Stamp I and BASIC Stamp II have many differences in both hardware and software. While it is trivial to recognize the differences in the Stamp hardware, the modifications to the PBASIC command structure are intricate and not always obvious. This appendix describes the Stamp I and Stamp II PBASIC differences in a detailed manner to aid in the conversion of programs between the two modules. This document may also serve to give a better understanding of how certain features of the two versions can be helpful in problem solving.

TYPOGRAPHICAL CONVENTIONS

This Appendix will use a number of symbols to deliver the needed information in a clear and concise manner. Unless otherwise noted the following symbols will have consistent meanings throughout this document.

TOPIC HEADING

Each discussion of a topic or PBASIC command will begin with a topic heading such as the one above.

MODULE HEADING

When separate discussion of a Stamp I or Stamp II module is necessary it will begin with a module heading such as this one.

-
-

Inside the module section bulleted items will precede information on the properties of various arguments for the indicated command.

CONVERSION:

When conversion between the two versions of PBASIC are necessary, each set of steps will begin under the conversion heading as shown above. This header will always begin with the word "Conversion" and will indicate in which direction the conversion is taking place; i.e. from BS1 to BS2 or from BS2 to BS1.

1. First do this...
2. Next do this...

The most important steps in conversion will be listed in a numeric sequence within the conversion section. The order of the numbered steps may be important in some situations and unimportant in others; it is best to follow the order as closely as possible.

Tips which are not vital to the conversion are listed within the conversion section and are preceded by bullets as shown above. These tips include additional information on valid argument types, properties of the command, etc. and may be used for further optimization of the code if desired.

As an example, using the above conventions, a typical section within this document will look like this:

SAMPLE COMMAND

BASIC Stamp I

Command syntax line shown here

- Argument one is…
- Argument two is…

BASIC Stamp II

Command syntax line shown here

- Argument one is...
- Argument two is...

Conversion: BS1 ⇨ BS2

1. First do this...
2. Next do this...

- You might like to know this...
- You might want to try this...

CONVERSION: BS1 ⇦ BS2

1. First do this...
2. Next do this...

- You might like to know this...
- You might want to try this...

The following symbols appear within command syntax listings or within the text describing them.

UPPER CASE	All command names will be shown is upper case lettering within the command syntax line. Argument names will be in upper case lettering outside of the command syntax line.
lower case	All arguments within the command syntax line will be in lower case lettering.
( )	Parentheses may appear inside a command syntax line and indicate that an actual parenthesis character is required at that location.
[ ]	Brackets may appear inside a command syntax line and indicate that an actual bracket character is required at that location.
[|]	Brackets with an internal separator may appear in the text following a command syntax line and indicate that one, and only one, of the items between the separators may be specified.
{ }	Wavy brackets may appear inside a command syntax line and indicate that the items they surround are optional and may be left out of the command. The wavy bracket characters themselves should not be used within the command, however.

#..#	Double periods between numbers indicate that a contiguous range of numbers are allowed for the given argument. Wherever a range of numbers are shown it usually indicates the valid range which a command expects to see. If a number is given which is outside of this range the Stamp will only use the lowest bits of the value which correspond to the indicated range. For example, if the range 0..7 is required (a 3 bit value) and the number 12 is provided, the Stamp will only use the lowest 3 bits which would correspond to a value of 4.

HOW TO USE THIS APPENDIX

This appendix should be used as a reference for converting specific commands, or other PBASIC entities, from one version of the Stamp to another. While this document will help to convert most of the programs available for the Stamp I and Stamp II, some programs may require logic changes to achieve correct results. The required logic changes are beyond the scope of this document.

In an effort to lessen the time spent in performing a code conversion the following routine should be followed in the order listed for each program.

1. Review the entire code briefly to familiarize yourself with how it functions and the types of commands and expressions which are used.
2. Consult the RAM SPACE AND REGISTER ALLOCATION section in this manual and go through the entire program carefully converting symbols, variables and expressions to the proper format.
3. Go through the code instruction by instruction, consulting the appropriate section in this document, and convert each one to the appropriate form.
4. Make any necessary circuit changes as required by the new stamp code.

COMMAND AND DIRECTIVE DIFFERENCES

Many enhancements to the Stamp I command structure were made in the Stamp II. Commands have also been added, replaced or removed. The following table shows the differences between the two modules.

BASIC Stamp I	BASIC Stamp II	Comments
BRANCH	BRANCH	Syntax Modifications
BSAVE		Removed
BUTTON	BUTTON	
	COUNT	New Command
DEBUG	DEBUG	Enhanced
EEPROM	DATA	Enhanced
	DTMFOUT	New Command
END	END	
(Expressions)	(Expressions)	Enhanced
FOR...NEXT	FOR...NEXT	Enhanced
GOSUB	GOSUB	Enhanced
GOTO	GOTO	
HIGH	HIGH	
IF...THEN	IF...THEN	Enhanced
INPUT	INPUT	
LET	(Expression)	Enhanced
LOOKDOWN	LOOKDOWN	Enhanced
LOOKUP	LOOKUP	Syntax Modifications
LOW	LOW	
NAP	NAP	
OUTPUT	OUTPUT	
PAUSE	PAUSE	
POT	RCTIME	Enhanced
PULSIN	PULSIN	Enhanced
PULSOUT	PULSOUT	Enhanced
PWM	PWM	Enhanced
RANDOM	RANDOM	
READ	READ	
(Register Allocation)	(Register Allocation)	Enhanced
REVERSE	REVERSE	
SERIN	SERIN	Enhanced
SEROUT	SEROUT	Enhanced
	SHIFTIN	New Command
	SHIFTOUT	New Command
SLEEP	SLEEP	
SOUND	FREQOUT	Enhanced
	STOP	New Command
TOGGLE	TOGGLE	
WRITE	WRITE	
	XOUT	New Command

BASIC Stamp I and Stamp II Conversions

RAM SPACE AND REGISTER ALLOCATION

BASIC Stamp I

The RAM space in the BASIC Stamp I consists of eight 16-bit words. Each word has a unique, predefined name as shown in the table below. Each word consists of two 8-bit bytes which have unique, predefined names. Additionally the first two words, PORT and W0, can be accessed as individual bits.

The first word, named PORT, is reserved to allow access and control over the 8 I/O pins on the Stamp I. This word consists of two bytes, PINS and DIRS, which represent the status and the data direction of the pins.

The other seven words are general purpose registers for use by the PBASIC program. They may be used via their direct name or by assigning symbols as aliases to specific registers.

To assign a symbol to a specific register, use the following format:

SYMBOL symbolname = registername

Example: SYMBOL LoopCounter = W0

- SYMBOLNAME is a series of characters (letters, numbers and underscores but not starting with a number) up to 32 characters in length.
- REGISTERNAME is a valid bit, byte or word register name as shown in the table below.

You may assign a symbol to a constant value by using a similar format:

SYMBOL symbolname = constantvalue

Example: SYMBOL MaxLoops = 100

- SYMBOLNAME is a series of characters (letters, numbers and underscores but not starting with a number) up to 32 characters in length.

- CONSTANTVALUE is a valid number in decimal, hexidecimal, binary or ascii.

Stamp I I/O and Variable Space			
Word Name	**Byte Name**	**Bit Names**	**Special Notes**
PORT	PINS DIRS	PIN0 - PIN7 DIR0 - DIR7	I/O pins; bit addressable. I/O pin direction control; bit addressable.
W0	B0 B1	BIT0 - BIT7 BIT8 - BIT15	Bit addressable. Bit addressable.
W1	B2 B3		
W2	B4 B5		
W3	B6 B7		
W4	B8 B9		
W5	B10 B11		
W6	B12 B13		Used by GOSUB instruction. Used by GOSUB instruction.

BASIC Stamp II

The RAM space of the BASIC Stamp II consists of sixteen words of 16 bits each. Each word and each byte within the word has a unique, predefined name similar to the Stamp I and shown in the table below.

The first three words, named INS, OUTS and DIRS, are reserved to allow access and control over the 16 I/O pins on the Stamp II. These reserved words represent the input states, output states and directions of the pins respectively and are the Stamp II version of the single control word, PORT, on the Stamp I. In comparison to the Stamp I, the control registers' size has been doubled and the I/O register PINS has been split into two words, INS and OUTS, for flexibility. Each word consists of a predefined name for its byte, nibble and bit parts.

The other thirteen words are general purpose registers for use by the PBASIC program. There are two methods of referencing these registers within the Stamp II as follows:

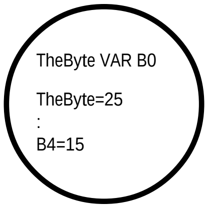
1. They may be referenced via their direct name or by defining symbols as aliases.

   - OR -

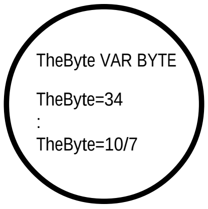
2. They may be referenced by defining variables of specific types (byte, word, etc.). The software will automatically assign variables to registers in an efficient manner.

The first method is used in the Stamp I, and supported in the Stamp II, as a means of directly allocating register space. The second method was introduced with the Stamp II as a means of indirectly allocating register space and is the *recommended method*.

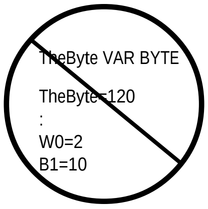
It is important to note that defining variables of specific types in the Stamp II is not directly equivalent to assigning symbols to registers in the Stamp I. Defining variables of specific types on the Stamp II allows the software to efficiently and automatically organize variable space within the general purpose registers while assigning symbols to registers allows you to organize variable space yourself. While both methods of register allocation are legal in the Stamp II, care should be taken to implement only one method of register use within each program. Each PBASIC program should either reference all registers by their predefined names (or symbols assigned to them) or reference all registers by defining variables of specific types and let the software do the organization for you. If you use both methods within the same program, it is likely that variables will *overlap* and your program will behave erratically. The following diagram may serve to clarify the use of the register allocation methods within a single Stamp II program:



Using only method 1 within a program is safe.



Using only method 2 within a program is safe.



Using both methods within a program leads to erratic execution.

To define a variable of a specific type, use the following format.

```
variablename  VAR  [ type{(arraysize)} | previousvariable{.modifier{.modifier...}} ]
```

Example:

```
LoopCounter     VAR  WORD                  'defines LoopCounter as a word.
LoopCounter2    VAR  BYTE(2)               'defines LoopCounter2 as an array of
                                           'two bytes.
FirstBit        VAR  LoopCounter.LOWBIT    'defines FirstBit as the lowest bit
                                           'within the variable LoopCounter.
```

- VARIABLENAME is a series of characters (letters, numbers and underscores but not starting with a number) up to 32 characters in length.
- TYPE is a valid variable type of BIT, NIB, BYTE or WORD.
- ARRAYSIZE is an optional constant value, in parentheses, specifying the number of elements of TYPE to define for the variable VARIABLENAME.
- PREVIOUSVARIABLE is the name of a previously defined variable. This can be used to assign alias names to the same variable space.
- MODIFIER is an optional offset, preceded by a period '.', which indicates which part of a previously defined variable to set VARIABLENAME to. Valid modifiers are: LOWBYTE, HIGHBYTE, BYTE0..1, LOWNIB, HIGHNIB, NIB0..3, LOWBIT, HIGHBIT and BIT0..15.

You may define a constant by using a similar format:

```
constantname  CON  constantexpression
```

Example:

```
MaxLoops     CON  100            'defines MaxLoops as a constant
                                 'equivalent to the number 100.
MaxLoops2    CON  50 * 4 / 2     'also defines MaxLoops as a
                                 'constant equivalent to the number 100.
```

- CONSTANTNAME is a series of characters (letters, numbers and underscores but not starting with a number) up to 32 characters in length.
- CONSTANTEXPRESSION is a numerical expression in decimal, hexidecimal, binary or ascii using only numbers and the +, -, *, /, &, |, ^, << or >> operators. NOTE: Parentheses are not allowed and expressions are always computed using 16-bits.

Stamp II I/O and Variable Space

Word Name	Byte Name	Nibble Names	Bit Names	Special Notes
INS	INL INH	INA, INB, INC, IND	IN0 - IN7, IN8 - IN15	Input pins; word, byte, nibble and bit addressable.
OUTS	OUTL OUTH	OUTA, OUTB, OUTC, OUTD	OUT0 - OUT7, OUT8 - OUT15	Output pins; word, byte, nibble and bit addressable.
DIRS	DIRL DIRH	DIRA, DIRB, DIRC, DIRD	DIR0 - DIR7, DIR8 - DIR15	I/O pin direction control; word, byte, nibble and bit addressable.
W0	B0 B1			General Purpose; word, byte, nibble and bit addressable.
W1	B2 B3			General Purpose; word, byte, nibble and bit addressable.
W2	B4 B5			General Purpose; word, byte, nibble and bit addressable.
W3	B6 B7			General Purpose; word, byte, nibble and bit addressable.
W4	B8 B9			General Purpose; word, byte, nibble and bit addressable.
W5	B10 B11			General Purpose; word, byte, nibble and bit addressable.
W6	B12 B13			General Purpose; word, byte, nibble and bit addressable.
W7	B14 B15			General Purpose; word, byte, nibble and bit addressable.
W8	B16 B17			General Purpose; word, byte, nibble and bit addressable.
W9	B18 B19			General Purpose; word, byte, nibble and bit addressable.
W10	B20 B21			General Purpose; word, byte, nibble and bit addressable.
W11	B22 B23			General Purpose; word, byte, nibble and bit addressable.
W12	B24 B25			General Purpose; word, byte, nibble and bit addressable.

Symbol Conversion: BS1 ⇨ BS2

1. Remove the 'SYMBOL' directive from variable or constant declarations.
2. On all variable declarations, replace the predefined register name, to the right of the '=', with the corresponding variable type or register name according to the following table:

BS1 to BS2 Register Allocation Conversion	
Stamp I Register Name	**Stamp II Variable Type / Register Name**
PORT	NO EQUIVALENT*
PINS or PIN0..PIN7	INS / OUTS or IN0..IN7 / OUT0..OUT7**
DIRS or DIR0..DIR7	DIRS or DIR0..DIR7
W0..W6	WORD
B0..B13	BYTE
BIT0..BIT15	BIT

* The PORT control register has been split into three registers, INS, OUTS and DIRS, on the Stamp II. There is no predefined name representing all registers as a group as in the Stamp I. Additional symbol and / or program structure and logic changes are necessary to access all three registers properly.

** The Stamp I PINS register has been split into two registers, INS and OUTS, in the Stamp II. Each register now has a specific task, input or output, rather than a dual task, both input and output, as in the Stamp I. If the Stamp I program used the symbol assigned to PINS for both input and output, an additional symbol is necessary to access both functions. This may also require further changes in program structure and logic.

1. On all variable declarations, replace the equal sign, '=', with 'VAR'.
2. On all constant declarations, replace the equal sign, '=', with 'CON'.

Variable or Constant Conversion: BS1 ⇦ BS2

1. Insert the 'SYMBOL' directive before the variable's name or constant's name in the declaration.
2. On all variable declarations, replace the variable type or register name, to the right of the '=', with the corresponding, predefined register name according to the following table:

BS2 to BS1 Register Allocation Conversion	
Stamp II Variable Type / Register Name	**Stamp I Register Name**
INS	PINS
OUTS	PINS
DIRS	DIRS
WORD	W0..W6
BYTE	B0..B13
NIB	B0..B13*
BIT	BIT0..BIT15**

* There are no registers on the Stamp I which are nibble addressable. The best possible solution is to place one or two nibble variables within a byte register and modify the code accordingly.

** The only general purpose registers on the Stamp I which are bit addressable are B0 and B1. BIT0..BIT7 correspond to the bits within B0 and BIT8..BIT15 correspond to the bits within B1. If you have a set of bit registers in the Stamp II program, you should reserve B0 and B1 for this bit usage; i.e.: do not assign any other symbols to B0 or B1.

3. On all variable and constant declarations, replace the variable or constant directive, ‘VAR’ or ‘CON’, with an equal sign, ‘=’.

Assignment Conversion: BS1 ⇦ BS2

1. Remove the ‘LET’ command if it is specified.
2. If PINS or PIN0..PIN7 appears to the left, or to the left and right, of the equal sign, ‘=’, replace PINS with OUTS and PIN0..PIN7 with OUT0..OUT7.
3. If PINS or PIN0..PIN7 appears to the right of the equal sign, ‘=’, replace PINS with INS and PIN0..PIN7 with IN0..IN7.
4. If PORT appears in an assignment, determine which byte (PINS or DIRS) is affected and replace PORT with the corresponding Stamp II symbol (INS, OUTS or DIRS). If both bytes are affected, separate assignment statements may be needed to accomplish the equivalent effect in the Stamp II.

BRANCH

BASIC STAMP I

BRANCH index,(label0, label1,... labeln)

- INDEX is a constant or a bit, byte or word variable.
- LABEL0..LABELN are valid labels to jump to according to the value of INDEX.

BASIC STAMP II

BRANCH index,[label0, label1,... labeln]

- INDEX is a constant, expression or a bit, nibble, byte or word variable.
- LABEL0..LABELN are valid labels to jump to according to the value of INDEX.

CONVERSION: BS1 ⇨ BS2

1. Change open and close parentheses, "(" and ")", to open and close brackets, "[" and "]".

Example:

```
BS1:        BRANCH  B0, ( Loop1, Loop2, Finish )

BS2:        BRANCH  BranchIdx, [ Loop1, Loop2, Finish ]
```

CONVERSION: BS1 ⇦ BS2

1. Change open and close brackets, "[" and "]", to open and close parentheses, "(" and ")".

Example:

```
BS2:        BRANCH BranchIdx, [ Loop1, Loop2, Finish ]

BS1:        BRANCH  B0, ( Loop1, Loop2, Finish )
```

BASIC Stamp I and Stamp II Conversions

BSAVE

BASIC STAMP I

BSAVE

- This is a compiler directive which causes the Stamp I software to create a file containing the tokenized form or the associated source code.

BASIC STAMP II

NO EQUIVELANT COMMAND

CONVERSION:

No conversion possible.

BUTTON

BASIC STAMP I

BUTTON pin, downstate, delay, rate, workspace, targetstate, label

- PIN is a constant or a bit, byte or word variable in the range 0..7.
- DOWNSTATE is a constant or a bit, byte or word variable in the range 0..1.
- DELAY is a constant or a bit, byte or word variable in the range 0..255.
- RATE is a constant or a bit, byte or word variable in the range 0..255.
- WORKSPACE is a byte or word variable.
- TARGETSTATE is a constant or a bit, byte or word variable in the range 0..1.
- LABEL is a valid label to jump to in the event of a button press.

BASIC STAMP II

BUTTON pin, downstate, delay, rate, workspace, targetstate, label

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- DOWNSTATE is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- DELAY is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- RATE is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- WORKSPACE is a byte or word variable.
- TARGETSTATE is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.

- LABEL is a valid label to jump to in the event of a button press.

CONVERSION: BS1 ⇨ BS2

1. PIN may be a constant or a bit, nibble, byte or word variable in the range 0..15.
2. Any or all arguments other than LABEL may be nibble variables for efficiency.

Example:

```
BS1:        BUTTON  0, 1, 255, 0, B0, 1, ButtonWasPressed

BS2:        BUTTON  0, 1, 255, 0, WkspcByte, 1, ButtonWasPressed
```

CONVERSION: BS1 ⇦ BS2

1. PIN must be a constant or a bit, byte or word variable in the range 0..7.
2. No arguments may be nibble variables.

Example:

```
BS2:        BUTTON  12, 1, 255, 0, WkspcByte, 1, ButtonWasPressed

BS1:        BUTTON  7, 1, 255, 0, B0, 1, ButtonWasPressed
```

COUNT

BASIC Stamp I

NO EQUIVELANT COMMAND

BASIC Stamp II

COUNT pin, period, result

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- PERIOD is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.
- RESULT is a bit, nibble, byte or word variable.

Conversion:

No conversion possible.

BASIC Stamp I and Stamp II Conversions

DEBUG

BASIC Stamp I

DEBUG outputdata{,outputdata...}

- OUTPUTDATA is a text string, bit, byte or word variable (no constants allowed).
- If no formatters are specified DEBUG defaults to "variablename = value" + carriage return.

FORMATTERS:
(The following formatting characters may precede the variable name)

#	displays value in decimal followed by a space.
$	displays "variablename = $value " + carriage return; where value is in hexidecimal.
%	displays "variablename = %value " + carriage return; where value is in binary.
@	displays "variablename = 'character' " + carriage return; where character is an ascii character.

SPECIAL SYMBOLS:
(The following symbols can be included in the output data)

CLS	causes the debug window to be cleared.
CR	causes a carriage return in the debug window.

BASIC Stamp II

DEBUG outputdata{,outputdata...}

- OUTPUTDATA is a text string, constant or a bit, nibble, byte or word variable.
- If no formatters are specified DEBUG defaults to ascii character display without spaces or carriage returns following the value.

FORMATTERS:
(The following formatting tokens may precede the data elements as indicated below)

ASC?	value	Displays “variablename = ‘character’ ” + carriage return; where character is an ascii character.
STR	bytearray	Displays values as an ascii string until a value of 0 is reached.
STR	bytearray\n	Displays values as an ascii string for n bytes.
REP	value\n	Displays value n times.
DEC{1..5}	value	Displays value in decimal, optionally limited or padded for 1 to 5 digits.
SDEC{1..5}	value	Displays value in *signed* decimal, optionally limited or padded for 1 to 5 digits. Value must not be less than a word variable.
HEX{1..4}	value	Displays value in hexidecimal, optionally limited or padded for 1 to 4 digits.
SHEX{1..4}	value	Displays value in *signed* hexidecimal, optionally limited or padded for 1 to 4 digits. Value must not be less than a word variable.
IHEX{1..4}	value	Displays value in hexidecimal preceded by a “$” and optionally limited or padded for 1 to 4 digits.
ISHEX{1..4}	value	Displays value in *signed* hexidecimal preceded by a “$” and optionally limited or padded for 1 to 4 digits. Value must not be less than a word variable.
BIN{1..16}	value	Displays value in binary, optionally limited or padded for 1 to 16 digits.
SBIN{1..16}	value	Displays value in *signed* binary, optionally limited or padded for 1 to 16 digits. Value must not be less than a word variable.

IBIN{1..16}	value	Displays value in binary preceded by a "%" and optionally limited or padded for 1 to 16 digits.
ISBIN{1..16}	value	Displays value in *signed* binary preceded by a "%" and optionally limited or padded for 1 to 16 digits. Value must not be less than a word variable.

SPECIAL SYMBOLS:
(The following symbols can be included in the output data)

BELL	Causes the computer to beep.
BKSP	Causes the cursor to backup one space.
CLS	Causes the debug window to be cleared.
CR	Causes a carriage return to occur in debug window.
HOME	Causes the cursor in the debug window to return to home position.
TAB	Causes the cursor to move to next tab position.

CONVERSION: BS1 ⇨ BS2

1. Replace all '#' formatters with 'DEC'.
2. Replace all '$' formatters with 'HEX?'.
3. Replace all '%' formatters with 'BIN?'.
4. Replace all '@' formatters with 'ASC?'.
5. If variable has no formatters preceding it, add the 'DEC?' formatter before variable.

- Signs, type indicators, strings and digit limitation formatting options are available for more flexibility.

Example:

```
BS1:     DEBUG #B0, $B1, %B2

BS2:     DEBUG DEC AByte, HEX? AWord, BIN? ANibble
```

Conversion: BS1 ⇦ BS2

1. Remove any 'DEC?' formatters preceding variables.
2. Replace all 'DEC' formatters with '#'.
3. Replace all 'HEX?' formatters with '$'.
4. Replace all 'BIN?' formatters with '%'.
5. Replace all 'ASC?' formatters with '@'.
6. Delete any '?' formatting characters.
7. Signs, type indicators, strings and digit limitation formatters are not available in the Stamp I. Manual formatting will have to be done (possibly multiple DEBUG statements) to accomplish the same formatting.

Example:

```
BS2:        DEBUG DEC AByte, HEX? AWord, BIN? ANibble, CR

BS1:        DEBUG #B0, $B1, %B2, CR
```

DATA

BASIC STAMP I

EEPROM {location,}(data{,data...})

- LOCATION is in the range 0..255.
- DATA is a constant in the range 0..255. No variables are allowed.

BASIC STAMP II

{pointer} DATA {@location,} {WORD} {data}{(size)} {, { WORD} {data}{(size)}...}

- POINTER is an optional undefined constant name or a bit, nibble, byte or word variable which is assigned the value of the first memory location in which data is written.
- @LOCATION is an optional constant, expression or a bit, nibble, byte or word variable which designates the first memory location in which data is to be written.
- WORD is an optional switch which causes DATA to be stored as two separate bytes in memory.
- DATA is an optional constant or expression to be written to memory.
- SIZE is an optional constant or expression which designates the number of bytes of defined or undefined data to write / reserve in memory. If DATA is not specified then undefined data space is reserved and if DATA is specified then SIZE bytes of data equal to DATA are written to memory.

CONVERSION: BS1 ⇨ BS2

1. Replace the EEPROM directive with the DATA directive.
2. If LOCATION is specified, insert an at sign, ‘@’, immediately before it.
3. Remove the open and close parentheses, ‘(‘ and ‘)’.

- The POINTER constant and WORD and (SIZE) directives may be used for added flexibility.

Example:

```
BS1:        EEPROM 100, (255, 128, 64, 92)

BS2:        DATA @100, 255, 128, 64, 92
```

CONVERSION: BS1 ⇦ BS2

1. If a POINTER constant is specified, remove it and set it equal to the value of the first location using a Stamp I assign statement.
2. Replace the DATA directive with the EEPROM directive.
3. If LOCATION is specified, remove the at sign, '@', immediately before it.
4. If the WORD directive is given, remove it and convert the data element immediately following it, if one exists, into two bytes of low-byte, high-byte format. If no data element exists immediately following the WORD directive, (the (SIZE) directive must exist) insert zero data element pairs, '0, 0,' for the number of elements given in (SIZE).
5. Add an open parenthesis, '(', just before the first data element and a close parenthesis, ')', after the last data element.
6. If the (SIZE) directive is given, remove it and copy the preceding data element, if available, into the number of SIZE data elements. If data was not given, insert SIZE data elements of zero, '0', separated by commas.

Example:

```
BS2:        MyDataPtr  DATA @100, 255, 128(2), 64, WORD 920, (10)

BS1:        SYMBOL MyDataPtr  =  100
            EEPROM MyDataPtr, (255, 128, 128, 64, 152, 3, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0)
```

BASIC Stamp I and Stamp II Conversions

DTMFOUT

BASIC STAMP I

NO EQUILEVANT COMMAND

BASIC STAMP II

DTMFOUT pin, {ontime, offtime,}[key{,key...}]

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- ONTIME and OFFTIME are constants, expressions or bit, nibble, byte or word variables in the range 0..65535.
- KEY is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

CONVERSION:

No conversion possible.

EEPROM (See DATA)

BASIC Stamp I and Stamp II Conversions

END

BASIC STAMP I

END

- 20uA reduced current (no loads).

BASIC STAMP II

END

- 50uA reduced current (no loads).

CONVERSION:

No conversion necessary.

Appendix C

EXPRESSIONS

BASIC STAMP I

{-} value ?? value {?? value...}

- Stamp I expressions are only allowed within an assignment statement. See the LET command for more detail.
- VALUE is a constant or a bit, byte or word variable.
- ?? is +,-,*,**,/,//,MIN,MAX,&,1,^,&/,|/,^/.

BASIC STAMP II

{?} value ?? value {?? {?} value}

- Stamp II expressions are allowed in place of almost any argument in any command as well as within an assignment statement.
- ? is SQR, ABS, ~, -, DCD, NCD, COS, SIN.
- VALUE is a constant or a bit, nibble, byte or word variable.
- ?? is +,-,*,**,*/,/,//,MIN,MAX,&,|,^,DIG,<<,>>,REV.
- Parentheses may be used to modify the order of expression evaluation.

CONVERSION: BS1 ⇨ BS2

1. Remove the LET command. This is not allowed in the Stamp II.

- VARIABLE and VALUE may be nibble variables for efficiency.
- The optional unary operator {-} may now also include SQR, ABS, ~, DCD, NCD, COS and SIN.
- The binary operators can now include */, DIG, <<, >> and REV.

Example:

```
BS1:        LET  b0 = -10 + 16

BS2:        Result = -10 + 16
```

CONVERSION: BS1 ⇦ BS2

1. Remove any unary operator other than minus (-) and modify the equation as appropriate, if possible.
2. The binary operator can not be */, DIG, <<, >> or REV.
3. VARIABLE and VALUE must not be a nibble variable.

Example:

```
BS2:        Result =  ~%0001 + 16

BS1:        b0 = %1110 + 16
```

FOR...NEXT

BASIC STAMP I

FOR variable = start TO end {STEP {-} stepval}...NEXT {variable}

- Up to 8 nested FOR...NEXT loops are allowed.
- VARIABLE is a bit, byte or word variable.
- START is a constant or a bit, byte or word variable.
- END is a constant or a bit, byte or word variable.
- STEPVAL is a constant or a bit, byte or word variable.
- VARIABLE (after NEXT) must be the same as VARIABLE (after FOR).

BASIC STAMP II

FOR variable = start TO end {STEP stepval}...NEXT

- Up to 16 nested FOR...NEXT loops are allowed.
- VARIABLE is a bit, nibble, byte or word variable.
- START is a constant, expression or a bit, nibble, byte or word variable.
- END is a constant, expression or a bit, nibble, byte or word variable.
- STEPVAL is an optional constant, expression or a bit, nibble, byte or word variable and must be positive.

CONVERSION: BS1 ⇨ BS2

1. Remove the minus sign "-" from the step value if given. The Stamp II dynamically determines the direction at run-time depending on the order of START and END. This allows for great flexibility in programming.

2. Remove the VARIABLE name after the NEXT statement if given. The variable is always assumed to be from the most recent FOR statement and is not allowed in the Stamp II.

- VARIABLE, START, END and STEPVAL may be a nibble variable for efficiency.
- Up to 16 nested FOR...NEXT statements may be used.

Example:

```
BS1:        FOR  B0 = 10 TO 1 STEP -1
  {code inside loop}
  NEXT B0

BS2:        FOR  LoopCount = 10 TO 1 STEP 1
  {code inside loop}
  NEXT
```

CONVERSION: BS1 ⇦ BS2

1. VARIABLE, START, END and STEPVAL must not be a nibble.
2. If negative stepping is to be done, a negative STEPVAL must be specified.
3. Must have no more than 8 nested FOR...NEXT loops.

Example:

```
BS2:        FOR  LoopCount = 100 TO 10 STEP 2
  {code inside loop}
  NEXT

BS1:        FOR  B0 = 100 TO 10 STEP -2
  {code inside loop}
  NEXT
```

FREQOUT

BASIC STAMP I

SOUND pin, (note, duration {,note, duration...})

- PIN is a constant or a bit, byte or word variable in the range of 0..7.
- NOTE is a constant or a bit, byte or word variable in the range of 0..255 representing frequencies in the range 94.8 Hz to 10,550 Hz.
- DURATION is a constant or a bit, byte or word variable in the range of 1..255 specifying the duration in 12 ms units.

BASIC STAMP II

FREQOUT pin, milliseconds, freq1 {,freq2}

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range of 0..15.
- MILLISECONDS is a constant, expression or a bit, nibble, byte or word variable.
- FREQ1 and FREQ2 are constant, expression or bit, nibble, byte or word variables in the range 0..32767 representing the corresponding frequencies. FREQ2 may be used to output 2 sine waves on the same pin simultaneously.

CONVERSION: BS1 ⇨ BS2

1. Change command name ‘SOUND’ to ‘FREQOUT’.
2. Remove the parentheses, ‘(‘ and ‘)’.
3. Swap the orientation of DURATION with NOTE and multiply DURATION by 12.
4. (MILLISECONDS = DURATION * 12).
5. Calculate FREQ1 using the formula: FREQ1 = 1 / (95 x 10^{-6} + ((127 - NOTE) * 83 x 10^{-6}).
5. Place successive NOTE and DURATION pairs into separate FREQOUT commands.

- PIN may be in the range 0..15.

Example:

```
BS1:        SOUND  1, (92, 128, 75, 25)

BS2:        FREQOUT  1, 1536, 333
            FREQOUT  1, 300, 226
```

CONVERSION: BS1 ⇦ BS2

1. Change command name ‘FREQOUT’ to ‘SOUND’.
2. PIN must be in the range 0..7.
3. Insert an open parenthesis just before the MILLISECONDS argument.
4. Swap the orientation of MILLISECONDS with FREQ1 and divide MILLISECONDS by 12. (DURATION = MILLISECONDS / 12).
5. Calculate NOTE using the formula: NOTE = $127 - ((1 / FREQ1) - 95 \times 10^{-6}) / 83 \times 10^{-6.}$
6. Successive FREQOUT commands may be combined into one SOUND command by separating NOTE and DURATION pairs with commas.
7. Insert a close parenthesis, ‘)’, after the last DURATION argument.

- Notes can not be mixed as in the Stamp II

Example:

```
BS2:        FREQOUT  15, 2000, 400
            FREQOUT  15, 500, 600

BS1:        SOUND  7, (98, 167, 108, 42)
```

GOSUB

BASIC STAMP I

GOSUB label

- Up to 16 GOSUBs allowed per program.
- Up to 4 nested GOSUBs allowed.
- Word W6 is modified with every occurrence of GOSUB.

BASIC STAMP II

GOSUB label

- Up to 255 GOSUBs allowed per program.
- Up to 4 nested GOSUBs allowed.

CONVERSION: BS1 ⇨ BS2

- Up to 255 GOSUBs can be used in the program.
- No general purpose variables are modified with the occurrence of GOSUB.

CONVERSION: BS1 ⇦ BS2

1. Only 16 GOSUBs can be used in the program.

- Word W6 is modified with every occurrence of GOSUB.

BASIC Stamp I and Stamp II Conversions

GOTO

BASIC Stamp I

GOTO label

BASIC Stamp II

GOTO label

Conversion:

No conversion necessary.

HIGH

BASIC STAMP I

HIGH pin

- PIN is a constant, expression or a bit, byte or word variable in the range 0..7.

BASIC STAMP II

HIGH pin

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

CONVERSION: BS1 ⇨ BS2

- PIN may be a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

CONVERSION: BS1 ⇦ BS2

- PIN must be a constant or a bit, byte or word variable in the range 0..7.

Example:

```
BS2:        HIGH  15

BS1:        HIGH  7
```

IF...THEN

BASIC STAMP I

IF variable ?? value {AND/OR variable ?? value...} THEN label

- VARIABLE is a bit, byte or word variable. No constants are allowed.
- ?? is =, <>, >, <, >=, <=.
- VALUE is a constant or a bit, byte, or word variable.
- LABEL is a location to branch to if the result is true.

BASIC STAMP II

IF conditionalexpression THEN label

- CONDITIONALEXPRESSION is any valid Boolean expression using the =, <>, >, <, >=, <=, conditional operators and the AND, OR, NOT, and XOR logical operators.
- LABEL is a location to branch to if the result is true.

CONVERSION: BS1 ⇨ BS2

1. If VARIABLE is PINS or PIN0..PIN7 then replace in with INS or IN0..IN7.

CONVERSION: BS1 ⇦ BS2

1. If the INS or OUTS symbol is specified to the left of the conditional operator, replace it with PINS.
2. If the logical operator NOT is specified, remove it and switch the conditional operator to negative logic.
3. If one of the values is an expression, you must perform the calculation in a dummy variable outside of the IF...THEN statement.

Example:

```
BS2:     IF NOT FirstValue > LastValue * (2 + NextValue) THEN Loop

BS1:     Temp = 2 + NextValue * LastValue
         IF FirstValue <= Temp THEN Loop
```

INPUT

BASIC STAMP I

INPUT pin

- PIN is a constant, expression or a bit, byte or word variable in the range 0..7.

BASIC STAMP II

INPUT pin

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

CONVERSION: BS1 ⇨ BS2

- PIN may be a nibble variable in the range 0..15.

CONVERSION: BS1 ⇦ BS2

- PIN must not be a nibble variable and must be in the range 0..7 only.

Example:

```
BS2:        INPUT  15

BS1:        INPUT  7
```

LET

BASIC Stamp I

{LET} variable = {-} value ?? value {?? value...}

- VARIABLE is a bit, byte or word variable.
- VALUE is a constant or a bit, byte or word variable.
- ?? is +,-,*,**,/,//,MIN,MAX,&,1,^,&/,|/,^/.

BASIC Stamp II

variable = {?} value ?? value {?? {?} value}

- VARIABLE is a bit, nibble, byte or word variable.
- ? is SQR, ABS, ~, -, DCD, NCD, COS, SIN.
- VALUE is a constant or a bit, nibble, byte or word variable.
- ?? is +,-,*,**,*/,/,//,MIN,MAX,&,|,^,DIG,<<,>>,REV.
- Parentheses may be used to modify the order of expression evaluation.

Conversion: BS1 ⇨ BS2

1. Remove the LET command. This is not allowed in the Stamp II.

- VARIABLE and VALUE may be nibble variables for efficiency.
- The optional unary operator {-} may now also include SQR, ABS, ~, DCD, NCD, COS and SIN.
- The binary operators can now include */, DIG, <<, >> and REV.

Example:

```
BS1:        LET  b0 = -10 + 16

BS2:        Result = -10 + 16
```

Conversion: BS1 ⇦ BS2

1. Remove any unary operator other than minus (-) and modify the equation as appropriate, if possible.

2. The binary operator can not be */, DIG, <<, >> or REV.
3. VARIABLE and VALUE must not be a nibble variable.

Example:

```
BS2:        Result =  ~%0001 + 16

BS1:        b0 = %1110 + 16
```

LOOKDOWN

BASIC Stamp I

LOOKDOWN value, (value0, value1,... valueN), variable

- VALUE is a constant or a bit, byte or word variable.
- VALUE0, VALUE1, etc. are constants or a bit, byte or word variables.
- VARIABLE is a bit, byte or word variable.

BASIC Stamp II

LOOKDOWN value, {??,} [value0, value1,... valueN], variable

- VALUE is a constant, expression or a bit, nibble, byte or word variable.
- ?? is =, <>, >, <, <=, =>. (= is the default).
- VALUE0, VALUE1, etc. are constants, expressions or bit, nibble, byte or word variables.
- VARIABLE is a bit, nibble, byte or word variable.

Conversion: BS1 ⇨ BS2

1. Change all parentheses, “(“ and “)”, to brackets, “[“ and “]”

- Any or all arguments may be nibble variables for efficiency.
- The optional ?? operator may be included for flexibility.

Example:

```
BS1:        LOOKDOWN  b0, (“A”, “B”, “C”, “D”), b1

BS2:        LOOKDOWN  ByteValue, [“A”, “B”, “C”, “D”], Result
```

Conversion: BS1 ⇦ BS2

1. Change all brackets, “[“ and “]”, to parentheses, “(“ and “)”.
2. Remove the “??,” argument if it exists and modify the list if possible. “=” is assumed in the Stamp I.

- None of the arguments may nibble variables.

Example:

```
BS2:        LOOKDOWN  ByteValue, [1, 2, 3, 4], Result

BS1:        LOOKDOWN  b0, (1, 2, 3, 4), b1
```

LOOKUP

BASIC STAMP I

LOOKUP index, (value0, value1,... valueN), variable

- INDEX is a constant or a bit, byte or word variable.
- VALUE0, VALUE1, etc. are constants or a bit, byte or word variables.
- VARIABLE is a bit, byte or word variable.

BASIC STAMP II

LOOKUP index, [value0, value1,... valueN], variable

- INDEX is a constant, expression or a bit, nibble, byte or word variable.
- VALUE0, VALUE1, etc. are constants, expressions or bit, nibble, byte or word variables.
- VARIABLE is a bit, nibble, byte or word variable.

CONVERSION: BS1 ⇨ BS2

1. Change all parentheses, "(" and ")", to brackets, "[" and "]"

- Any or all arguments may be nibble variables for efficiency.

Example:

```
BS1:        LOOKUP  b0, (1, 2, 3, 4), b1

BS2:        LOOKUP  ByteValue, [1, 2, 3, 4], Result
```

CONVERSION: BS1 ⇦ BS2

1. Change all brackets, "[" and "]", to parentheses, "(" and ")"

- None of the arguments may nibble variables.

Example:

```
BS2:        LOOKUP  ByteValue, [1, 2, 3, 4], Result

BS1:        LOOKUP  b0, (1, 2, 3, 4), b1
```

LOW

BASIC Stamp I

LOW pin

- PIN is a constant or a bit, byte or word variable in the range 0..7.

BASIC Stamp II

LOW pin

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

Conversion: BS1 ⇨ BS2

- PIN may be a constant or a bit, nibble, byte or word variable in the range 0..15.

Conversion: BS1 ⇦ BS2

PIN must be a constant or a bit, byte or word variable in the range 0..7.

Example:

```
BS2:        LOW  15

BS1:        LOW  7
```

NAP

BASIC STAMP I

NAP period

- PERIOD is a constant or a bit, byte or word variable in the range 0..7 representing 18ms intervals.
- Current is reduced to 20uA (assuming no loads).

BASIC STAMP II

NAP period

- PERIOD is a constant, expression or a bit, nibble, byte or word variable in the range 0..7 representing 18ms intervals.
- Current is reduced to 50uA (assuming no loads).

CONVERSION:

No conversion necessary.

OUTPUT

BASIC STAMP I

OUTPUT pin

- PIN is a constant or a bit, byte or word variable in the range 0..7.

BASIC STAMP II

OUTPUT pin

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

CONVERSION: BS1 ⇨ BS2

- PIN may be a constant or a bit, nibble, byte or word variable in the range 0..15.

CONVERSION: BS1 ⇦ BS2

1. PIN must be a constant or a bit, byte or word variable in the range 0..7.

Example:

```
BS2:      OUTPUT  15

BS1:      INPUT  7
```

PAUSE

BASIC Stamp I

PAUSE milliseconds

- MILLISECONDS is a constant or a bit, byte or word variable in the range 0..65535.

BASIC Stamp II

PAUSE milliseconds

- MILLISECONDS is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.

Conversion:

No conversion necessary.

POT (See RCTIME)

PULSIN

BASIC Stamp I

PULSIN pin, state, variable

- PIN is a constant, expression or a bit, byte or word variable in the range 0..7.
- STATE is a constant, expression or a bit, byte or word variable in the range 0..1.
- VARIABLE is a bit, byte or word variable.
- Measurements are in 10uS intervals and the instruction will time out in 0.65535 seconds.

BASIC Stamp II

PULSIN pin, state, variable

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- STATE is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- VARIABLE is a bit, nibble, byte or word variable.
- Measurements are in 2uS intervals and the instruction will time out in 0.13107 seconds.

Conversion: BS1 ⇨ BS2

- Any or all arguments may be a nibble variable for efficiency.
- PIN may be in the range 0..15.
- Returned value is 5 times less than in the Stamp I counterpart.

CONVERSION: BS1 ⇦ BS2

- None of the arguments may be a nibble variable.
- PIN must be in the range 0..7.
- Returned value is 5 times more than in the Stamp I counterpart.

Example:

```
BS2:        PULSIN  15, 1, Result

BS1:        PULSIN  7, 1, W0
```

PULSOUT

BASIC Stamp I

PULSOUT pin, time

- PIN is a constant or a bit, byte or word variable in the range 0..7.
- TIME is a constant or a bit, byte or word variable in the range 0..65535 representing the pulse width in 10uS units.

BASIC Stamp II

PULSOUT pin, period

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- PERIOD is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 representing the pulse width in 2uS units.

Conversion: BS1 ⇨ BS2

1. PERIOD = TIME * 5.

- PIN may be a nibble variable in the range 0..15.

Example:

```
BS1:        PULSOUT  1, 10

BS2:        PULSOUT  1, 50
```

Conversion: BS1 ⇦ BS2

1. TIME = PERIOD / 5.

- PIN must be in the range 0..7 and must not be a nibble variable.

Example:

```
BS2:        PULSOUT  15, 25

BS1:        PULSOUT  7, 5
```

PWM

BASIC STAMP I

PWM pin, duty, cycles

- PIN is a constant or a bit, byte or word variable in the range 0..7.
- DUTY is a constant or a bit, byte or word variable in the range 0..255.
- CYCLES is a constant or a bit, byte or word variable in the range 0..255 representing the number of 5ms cycles to output.

BASIC STAMP II

PWM pin, duty, cycles

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- DUTY is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- CYCLES is a constant, expression or a bit, nibble, byte or word variable in the range 0..255 representing the number of 1ms cycles to output.

CONVERSION: BS1 ⇨ BS2

1. CYCLES = CYCLES * 5.

- PIN may be a nibble variable in the range 0..15.

Example:

```
BS1:        PWM  0, 5, 1

BS2:        PWM  0, 5, 5
```

CONVERSION: BS1 ⇦ BS2

1. CYCLES = CYCLES / 5.

- PIN must be in the range 0..7 and must not be a nibble variable.

Example:

```
BS2:        PWM  15, 5, 20

BS1:        PWM  7, 5, 4
```

RANDOM

BASIC STAMP I

RANDOM variable

- VARIABLE is a byte or word variable in the range 0..65535.

BASIC STAMP II

RANDOM variable

- VARIABLE is a byte or word variable in the range 0..65535.

CONVERSION: BS1 ⇨ BS2

- The numbers generated for any given input will not be the same on the Stamp II as in the Stamp I.

CONVERSION: BS1 ⇦ BS2

- The numbers generated for any given input will not be the same on the Stamp I as in the Stamp II.

RCTIME

BASIC Stamp I

POT pin, scale, bytevariable

- PIN is a constant or a bit, byte or word variable in the range 0..7.
- SCALE is a constant or a bit, byte or word variable in the range 0..255.
- BYTEVARIABLE is a byte variable.

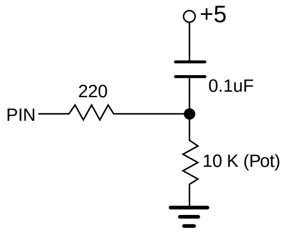
BASIC Stamp II

RCTIME pin, state, variable

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- STATE is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- VARIABLE is a bit, nibble, byte or word variable.

Conversion: BS1 ⇨ BS2

1. Modify the circuit connected to PIN to look similar to the following diagram. (Note, your values for the resistors and capacitor may be different).



2. Insert two lines before the POT command as follows:

   HIGH pin ; where PIN is the same PIN in the POT command.

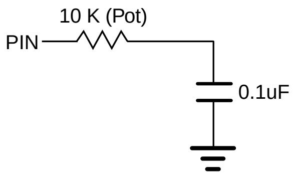
PAUSE delay ; where DELAY is an appropriate time in milliseconds to allow the capacitor to; fully discharge. You may have to try different DELAY values to find an optimal; value.

3. Change the command's name from 'POT' to 'RCTIME'.
4. Replace the SCALE argument with a STATE argument; our example requires a 1.

- PIN may be a nibble variable in the range 0..15.

CONVERSION: BS1 ⇦ BS2

1. Modify the circuit connected to PIN to look similar to the following diagram. (Note, your values for the resistor and capacitor may be different).



2. Delete the code before the RCTIME command which discharges the capacitor. This code usually consists of two lines as follows:

HIGH pin ; where PIN is the same PIN in the RCTIME command.

PAUSE delay ; where DELAY is an appropriate time in milliseconds to allow the capacitor to; fully discharge.

3. Change the command's name from 'RCTIME' to 'POT'.
4. Use the ALT-P key combination to determine the appropriate scale factor for the POT you are using as described in the BASIC Stamp I manual.

5. Replace the STATE argument with a SCALE argument.
6. Make VARIABLE a byte variable.

- PIN must be in the range 0..7 and must not be a nibble variable.

READ

BASIC STAMP I

READ location, variable

- LOCATION is a constant or a bit, byte or word variable in the range 0..255.
- VARIABLE is a bit, byte or word variable.

BASIC STAMP II

READ location, variable

- LOCATION is a constant, expression or a bit, nibble, byte or word variable in the range 0..2047.
- VARIABLE is a bit, nibble, byte or word variable.

CONVERSION: BS1 ⇨ BS2

- LOCATION and VARIABLE may be a nibble variable for efficiency.
- LOCATION may be in the range 0..2047.

CONVERSION: BS1 ⇦ BS2

- LOCATION and VARIABLE must not be a nibble variable.
- LOCATION must be in the range 0..255.

REVERSE

BASIC Stamp I

REVERSE pin

- PIN is a constant or a bit, byte or word variable in the range 0..7.

BASIC Stamp II

REVERSE pin

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

Conversion: BS1 ⇨ BS2

- PIN may be a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

Conversion: BS1 ⇦ BS2

- PIN must be a constant or a bit, byte or word variable in the range 0..7.

Example:

```
BS2:        REVERSE 15

BS1:        REVERSE 7
```

SERIN

BASIC STAMP I

SERIN pin, baudmode {,(qualifier {,qualifier...}) } {,{#} variable...}

- PIN is a constant or a bit, byte or word variable in the range 0..7.
- BAUDMODE is a constant or a bit, byte or word variable in the range 0..7 or a symbol with the following format: [T | N][2400 | 1200 | 600 | 300].
- QUALIFIERs are optional constants or a bit, byte or word variables which must be received in the designated order for execution to continue.
- VARIABLE is a bit, byte or word variable.
- # will convert ascii numbers to a binary equivalent.

BASIC STAMP II

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

- RPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..16.
- FPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- BAUDMODE is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.
- PLABEL is a label to jump to in case of a parity error.
- TIMEOUT is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 representing the number of milliseconds to wait for an incoming message.
- TLABEL is a label to jump to in case of a timeout.
- INPUTDATA is a set of constants, expressions and variable names separated by commas and optionally proceeded by the formatters available in the DEBUG command, except the ASC and REP

formatters. Additionally, the following formatters are available:

STR	bytearray\L{\E}	input a string into bytearray of length L with optional end-character of E. (0's will fill remaining bytes).
SKIP	L	input and ignore L bytes.
WAITSTR	bytearray{\L}	Wait for bytearray string (of L length, or terminated by 0 if parameter is not specified and is 6 bytes maximum).
WAIT	(value {,value...})	Wait for up to a six-byte sequence.

CONVERSION: BS1 ⇨ BS2

1. BAUDMODE is a constant or a bit, nibble, byte or word variable equal to the bit period of the baud rate plus three control bits which specify 8-bit / 7-bit, True / Inverted and Driven / Open output. The following table lists the Stamp I baudmodes and the corresponding Stamp II baudmode:

SERIN Baudmode Conversion

Stamp I Baudmode		Stamp II Baudmode
0	T2400	396
1	T1200	813
2	T600	1646
3	T300	3313
4	N2400	396 + $4000
5	N1200	813 + $4000
6	N600	1646 + $4000
7	N300	3313 + $4000

2. INPUTDATA includes QUALIFIERS and VARIABLES and must

be encased in brackets, "[" and "]". If QUALIFIERS are present, insert the modifier "WAIT" immediately before the open parenthesis before the first QUALIFIER.

3. Replace any optional "#" formatters with the equivalent "DEC" formatter.

- RPIN = PIN and may be in the range 0..16
- BAUDMODE may be any bit period in between 300 baud and 50000 baud and can be calculated using the following formula: INT(1,000,000 / Baud Rate) - 20.
- The optional formatter may include any formatter listed for INPUTDATA above.

Example:

```
BS1:        SERIN  0, 1, ("ABCD"), #B0, B1

BS2:        SERIN  0, 813, [WAIT("ABCD"), DEC FirstByte, SecondByte]
```

CONVERSION: BS1 ⇦ BS2

1. PIN = RPIN and must be in the range 0..7.
2. Remove the FPIN argument "\fpin" if it is specified. No flow control pin is available on the Stamp I.
3. BAUDMODE is a constant or a symbol or a bit, byte or word variable representing one of the predefined baudmodes. Refer to the BAUDMODE Conversion table above for Stamp II baudmodes and their corresponding Stamp I baudmodes. While the Stamp II baudmode is quite flexible, the Stamp I can only emulate specific baud rates.
4. Remove the PLABEL argument if it is specified. No parity error checking is done on the Stamp I.
5. Remove the TIMEOUT and TLABEL arguments if they are specified. No timeout function is available on the Stamp I; the program will halt at the SERIN instruction until satisfactory data arrives.
6. Remove the brackets, "[" and "]".

7. If QUALIFIERS are specified within a WAIT modifier, remove the word “WAIT”.

8. IF QUALIFIERS are specified within a WAITSTR modifier, replace the word “WAITSTR” with an open parenthesis, “(“. Convert the bytearray into a constant text or number sequence separated by commas if necessary (remove the length specifier “\L” if one exists) and insert a close parenthesis, “)”, immediately afterward.

9. If a variable is preceded with a DEC formatter, replace the word “DEC” with “#”.

10. Any formatter other than DEC and WAIT or WAITSTR has no direct equivalent in the Stamp I and must be removed. Additional variables or parsing routines will have to be used to achieve the same results in the Stamp I as with the Stamp II.

Example:

```
BS2:       SERIN  15, 813, 1000, TimedOut, [WAIT(“ABCD”), DEC
           FirstByte, SecondByte]

BS1:       SERIN  7, 1, (“ABCD”), #B0, B1
```

SEROUT

BASIC STAMP I

SEROUT pin, baudmode, ({#} data {, {#} data...})

- PIN is a constant or a bit, byte or word variable in the range 0..7.
- BAUDMODE is a constant or a bit, byte or word variable in the range 0..15 or a symbol with the following format: {O}[T | N][2400 | 1200 | 600 | 300].
- DATA is a constant or a bit, byte or word variable.
- # will convert binary numbers to ascii text equivalents up to 5 digits in length.

BASIC STAMP II

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

- TPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..16.
- FPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- BAUDMODE is a constant, expression or a bit, nibble, byte or word variable in the range 0..60657.
- PACE is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 specifying a time (in milliseconds) to delay between transmitted bytes. This value can only be specified if the FPIN is not specified.
- TIMEOUT is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 representing the number of milliseconds to wait for the signal to transmit the message. This value can only be specified if the FPIN is specified.
- TLABEL is a label to jump to in case of a timeout. This can only be specified if the FPIN is specified.

- OUTPUTDATA is a set of constants, expressions and variable names separated by commas and optionally proceeded by the formatters available in the DEBUG command.

CONVERSION: BS1 ⇨ BS2

1. BAUDMODE is a constant or a bit, nibble, byte or word variable equal to the bit period of the baud rate plus three control bits which specify 8-bit / 7-bit, True / Inverted and Driven / Open output. The following table lists the Stamp I baudmodes and the corresponding Stamp II baudmode:

SEROUT Baudmode Conversion

Stamp I Baudmode		Stamp II Baudmode
0	T2400	396
1	T1200	813
2	T600	1646
3	T300	3313
4	N2400	396 + $4000
5	N1200	813 + $4000
6	N600	1646 + $4000
7	N300	3313 + $4000
8	OT2400	396 + $8000
9	OT1200	813 + $8000
10	OT600	1646 + $8000
11	OT300	3313 + $8000
12	ON2400	396 + $C000
13	ON1200	813 + $C000
14	ON600	1646 + $C000
15	ON300	3313 + $C000

1. Replace the parentheses, “(“ and “)”, with brackets, “[“ and “]”.
2. Replace any optional “#” formatters with the equivalent “DEC” formatter.

- TPIN = PIN and may be in the range 0..16.
- BAUDMODE may be any bit period in between 300 baud and 50000 baud and can be calculated using the following formula: INT(1,000,000 / Baud Rate) - 20.
- The optional formatter may include any valid formatter for the DEBUG command.

Example:

```
BS1:        SEROUT  3, T2400, ("Start", #B0, B1)

BS2:        SEROUT  3, 396, ["Start", DEC FirstByte, SecondByte]
```

CONVERSION: BS1 ⇦ BS2

1. PIN = TPIN and must be in the range 0..7.
2. Remove the FPIN argument "\fpin" if it is specified. No flow control pin is available on the Stamp I.
3. BAUDMODE is a constant or a symbol or a bit, byte or word variable representing one of the predefined baudmodes. Refer to the BAUDMODE Conversion table above for Stamp II baudmodes and their corresponding Stamp I baudmodes. While the Stamp II baudmode is quite flexible, the Stamp I can only emulate specific baud rates.
4. Remove the PACE argument if it is specified. No pace value is allowed on the Stamp I.
5. Remove the TIMEOUT and TLABEL arguments if they are specified. No timeout function is available on the Stamp I; the program will transmit data regardless of the status of the receiver.
6. Replace the brackets, "[" and "]", with parentheses, "(" and ")".
7. If a variable is preceded with a DEC formatter, replace the word "DEC" with "#".
8. Any formatter other than DEC has no direct equivalent in the Stamp I and must be removed. Additional variables or constants will have to be used to achieve the same results in the Stamp I as with the Stamp II.

Example:

BS2: SEROUT 15, 3313, 1000, TimedOut, [“Start”, DEC FirstByte, SecondByte]

BS1: SEROUT 7, T300, (“Start”, #B0, B1)

SHIFTIN

BASIC STAMP I

NO EQUIVELANT COMMAND

BASIC STAMP II

SHIFTIN dpin, cpin, mode, [result{\bits} { ,result{\bits}... }]

- DPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the data pin.
- CPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the clock pin.
- MODE is a constant, symbol, expression or a bit, nibble, byte or word variable in the range 0..4 specifying the bit order and clock mode. 0 or MSBPRE = msb first, pre-clock, 1 or LSBPRE = lsb first, pre-clock, 2 or MSBPOST = msb first, post-clock, 3 or LSBPOST = lsb first, post-clock.
- RESULT is a bit, nibble, byte or word variable where the received data is stored.
- BITS is a constant, expression or a bit, nibble, byte or word variable in the range 1..16 specifying the number of bits to receive in RESULT. The default is 8.

CONVERSION: BS1 ⇨ BS2

- Code such as the following:

```
SYMBOL  Value = B0           'Result of shifted data
SYMBOL  Count = B1           'Counter variable
SYMBOL  CLK = 0              'Clock pin is pin 0
SYMBOL  DATA = PIN1          'Data pin is pin 1

DIRS  =  %00000001           'Set Clock pin as output and Data pin as input

FOR  Count = 1 TO 8
   PULSOUT CLK,1             'Preclock the data
   Value = Value * 2 + DATA  'Shift result left and grab next data bit
NEXT Count
```

May be converted to the following code:

```
Value       VAR     BYTE          'Result of shifted data
CLK         CON     0             'Clock pin is pin 0
DATA        CON     1             'Data pin is pin 1

DIRS  =  %0000000000000001        'Set Clock pin as output and Data pin
as input

SHIFTIN  DATA, CLK, MSBPRE, [Value\8]
```

CONVERSION: BS1 ⇦ BS2

• Code such as the following:

```
Value       VAR     BYTE          'Result of shifted data

DIRS  =  %0000000000000001        'Clock pin is 0 and Data pin is 1

SHIFTIN  1, 0, LSBPOST, [Value\8]
```

May be converted to the following code:

```
SYMBOL  Value = B0                'Result of shifted data
SYMBOL  Count = B1                'Counter variable

DIRS  =  %00000001                'Clock pin is 0 and Data pin is 1

FOR  Count = 1 TO 8
   Value = DATA * 256 + Value / 2 'Shift grab next data bit and shift right
   PULSOUT CLK,1                  'Postclock the data
NEXT Count
```

SHIFTOUT

BASIC STAMP I

NO EQUIVELANT COMMAND

BASIC STAMP II

SHIFTOUT dpin, cpin, mode, [data{\bits} {, data{\bits}... }]

- DPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the data pin.
- CPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the clock pin.
- MODE is a constant, symbol, expression or a bit, nibble, byte or word variable in the range 0..1 specifying the bit order. 0 or LSBFIRST = lsb first, 1 or MSBFIRST = msb first.
- DATA is a constant, expression or a bit, nibble, byte or word variable containing the data to send out.
- BITS is a constant, expression or a bit, nibble, byte or word variable in the range 1..16 specifying the number of bits of DATA to send. The default is 8.

CONVERSION: BS1 ⇨ BS2

Code such as the following:

```
SYMBOL  Count = B1            'Counter variable
SYMBOL  CLK = 0               'Clock pin is pin 0
SYMBOL  DATA = PIN1           'Data pin is pin 1

DIRS  =  %00000011            'Set Clock and Data pins as outputs

   B0 = 125                   'Value to be shifted out

FOR  Count = 1 TO 8
   DATA = BIT7                'Send out MSB of B0
   PULSOUT CLK,1              'Clock the data
   B0 = B0 * 2                'Shift the value left; note that this causes us
                              'to lose the value
NEXT Count                    'when we're done shifting
```

May be converted to the following code:

```
Value       VAR     BYTE                            'Value to be shifted out
CLK         CON     0                               'Clock pin is pin 0
DATA        CON     1                               'Data pin is pin 1

DIRS  =  %0000000000000011                          'Set Clock and Data pins as
                                                    'outputs

Value = 125

SHIFTOUT  DATA, CLK, MSBFIRST, [Value\8]            'Note that value is still intact
                                                    'after were done shifting
```

CONVERSION: BS1 ⇦ BS2

Code such as the following:

```
Value       VAR     BYTE                            'Value to be shifted out
CLK         CON     0                               'Clock pin is pin 0
DATA        CON     1                               'Data pin is pin 1

DIRS  =  %0000000000000011                          'Set Clock and Data pins as
                                                    'outputs

Value = 220

SHIFTOUT  DATA, CLK, LSBFIRST, [Value\8]            'Note that value is still intact
                                                    'after were done shifting
```

May be converted to the following code:

```
SYMBOL  Count = B1                                  'Counter variable
SYMBOL  CLK = 0                                     'Clock pin is pin 0
SYMBOL  DATA = PIN1                                 'Data pin is pin 1

DIRS  =  %00000011                                  'Set Clock and Data pins as
                                                    'outputs

            B0 = 220                                'Value to be shifted out

FOR  Count = 1 TO 8
            DATA = BIT0                             'Send out LSB of B0
            PULSOUT CLK,1                           'Clock the data
            B0 = B0 / 2                             'Shift the value left; note that
                                                    'the value is lost after were
                                                    'done
NEXT Count                                          'shifting
```

SLEEP

BASIC STAMP I

SLEEP seconds

- SECONDS is a constant or a bit, byte or word variable in the range 1..65535 specifying the number of seconds to sleep.

BASIC STAMP II

SLEEP seconds

- SECONDS is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 specifying the number of seconds to sleep.

CONVERSION:

No conversion necessary.

SOUND (See FREQOUT)

STOP

BASIC Stamp I

NO EQUIVELANT COMMAND

BASIC Stamp II

STOP

- Execution is frozen, such as with the END command, however, low-power mode is not entered and the I/O pins never go into high impedance mode.

Conversion: BS1 ⇨ BS2

Code such as the following:

```
StopExecution:          GOTO StopExecution
```

May be converted to the following code:

```
StopExecution:          STOP
```

Conversion: BS1 ⇦ BS2

Code such as the following:

```
Quit:          STOP
```

May be converted to the following code:

```
Quit:          GOTO Quit
```

TOGGLE

BASIC STAMP I

TOGGLE pin

- PIN is a constant or a bit, byte or word variable in the range 0..7.

BASIC STAMP II

TOGGLE pin

- PIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

CONVERSION: BS1 ⇨ BS2

- PIN may be a nibble variable and may be in the range 0..15.

CONVERSION: BS1 ⇦ BS2

- PIN must not be a nibble variable and must be in the range 0..7.

Example:

```
BS2:        TOGGLE  15

BS1:        TOGGLE  7
```

WRITE

BASIC STAMP I

WRITE location, data

- LOCATION is a constant or a bit, byte or word variable in the range 0..255.
- DATA is a constant or a bit, byte or word variable.

BASIC STAMP II

WRITE location, data

- LOCATION is a constant, expression or a bit, nibble, byte or word variable in the range 0..2047.
- DATA is a constant, expression or a bit, nibble, byte or word variable.

CONVERSION: BS1 ⇨ BS2

- LOCATION and DATA may be a nibble variable for efficiency.
- LOCATION may be in the range 0..2047.

CONVERSION: BS1 ⇦ BS2

- LOCATION and DATA must not be a nibble variable.
- LOCATION must be in the range 0..255.

XOUT

BASIC STAMP I

NO EQUIVELANT COMMAND

BASIC STAMP II

**XOUT mpin, zpin, [house\keyorcommand{\cycles}
{, house\keyorcommand{\cycles}... }]**

- MPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the modulation pin.
- ZPIN is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the zero-crossing pin.
- HOUSE is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the house code A..P respectively.
- KEYORCOMMAND is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying keys 1..16 respectively or is one of the commands in the following table:

X-10 Commands	
X-10 Command (symbol)	**Value**
UNITON	%10010
UNITOFF	%11010
UNITSOFF	%11100
LIGHTSON	%10100
DIM	%11110
BRIGHT	%10110

- CYCLES is a constant, expression or a bit, nibble, byte or word variable in the range 2..65535 specifying the number of cycles to send. (Default is 2).

CONVERSION:

No conversion possible.