Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	-
Core Size	-
Speed	-
Connectivity	-
Peripherals	-
Number of I/O	-
Program Memory Size	-
Program Memory Type	-
EEPROM Size	-
RAM Size	-
Voltage - Supply (Vcc/Vdd)	-
Data Converters	-
Oscillator Type	-
Operating Temperature	-
Mounting Type	-
Package / Case	-
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/parallax/pbasic48w-pe

Preface

Thank you for purchasing a Parallax BASIC Stamp® microcontroller module. We have done our best to produce several full-featured, easy to use development systems for BASIC Stamp microcontrollers. Depending on the Starter Kit you purchased, your BASIC Stamp model, development board and other contents will vary.

This manual is written for the latest available BASIC Stamp modules and software as of February 2005. As the product-line evolves, new information may become available. It is always recommended to visit the Parallax web site, www.parallax.com, for the latest information.

This manual is intended to be a complete reference manual to the architecture and command structure of the various BASIC Stamp models. This manual is not meant to teach BASIC programming or electrical design; though a person can learn a lot by paying close attention to the details in this book.

If you have never programmed in the BASIC language or are unfamiliar with electronics, it would be best to locate one or more of the books listed on the following page for assistance. All are available, either to order or to download, from www.parallax.com.

Books available in Adobe's PDF format are published for free download on the Parallax web site or on the CD-ROM which ships with our different Starter Kits. Books available in print may be purchased directly from Parallax or other distributors.

In addition, there are hundreds of great examples available on the Parallax CD and web site (www.parallax.com). Also, Nuts & Volts Magazine (www.nutsvolts.com / 1-800-783-4624) is a national electronic hobbyist's magazine that features monthly articles featuring BASIC Stamp applications. This is an excellent resource for beginners and experts alike!

any BS2 model source code. A $PBASIC directive is required to use version 2.5, which is compatible with all BS2 models.

PBASIC 2.5 has enhanced syntax options for several commands, as well as some additional commands not available in PBASIC 2.0. Table 3.4 shows the number of PBASIC commands that are available in each version of the PBASIC language, on each BASIC Stamp model. Details about the syntax differences among the three versions of PBASIC are denoted by icons in the margins of Chapters 4 and 5; also refer to Table 5.1 on page 124 and individual command syntax descriptions.

Table 3.4: Number of Available Commands for each BASIC Stamp Model with each version of the PBASIC language .

	BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px
PBASIC 1.0	32	-	-	-	-	-	-
PBASIC 2.0	-	37	40	40	56	56	58
PBASIC 2.5	-	42	45	45	61	61	63

A categorical listing of all PBASIC commands is included at the beginning of Chapter 5, followed by detailed descriptions of each command in alphabetical order.

Note that the syntax-highlighting feature of the BASIC Stamp Editor will also adjust to the language version indicated by the $PBASIC directive. The best way to select the $PBASIC directive is to use the toolbar icons, as was shown in Figure 3.9. Like the $STAMP directive, you must use care if you choose to type it in by hand. The syntax is:

```
' {$PBASIC 1.0}     'Default when a BASIC Stamp 1 module is detected
' {$PBASIC 2.0}     'Default when any BASIC Stamp 2 module is detected
' {$PBASIC 2.5}     'Required for PBASIC 2.5 command set & enhanced syntax
```

If you try to run a program that contains command syntax specific to PBASIC 2.5 without including the corresponding compiler directive, you will probably get an error message. In this case, insert a $PBASIC 2.5 directive and try running the program again.

THE $PORT DIRECTIVE.

The optional $PORT directive allows you to indicate a specific PC communications port through which to download a program to a BASIC Stamp module. The syntax is as follows:

```
' {$PORT COM#}
```

where # is a valid port number. When any PBASIC program containing this directive is downloaded, all other ports will be ignored. This directive is especially convenient when using two of the same BASIC Stamp models

THE FIXED TAB POSITIONS LIST.

The Fixed Tab Positions list is used to provide a list of desired fixed tab positions (used with Fixed Tabs or Fixed plus Smart Tabs options). The list can be a single number, or a list of comma separated numbers in ascending order. The allowable range is 2 to 512 and the list size is virtually unlimited. When multiple values are entered, the difference between the last two values will be used to set tab positions beyond the last position. For example, in the default list, the last two positions are 9 and 11; resulting in further tab positions of 13, 15, 17, etc. (multiples of 2 after the last specified position). Since source code is usually indented by multiples of two (2) spaces, the default list of 3, 5, 7, 9 and 11 is recommended.

DEFAULT COM PORT.

The Default Com Port setting allows you to specify which COM port to download through. If you specify a specific port here, the Identification window will report that it is "ignoring" other known ports. This can be selectively overridden by placing a $PORT directive in the program. If this setting is left on "AUTO", the default, the editor will open and scan all known ports for the correct BASIC Stamp. The button to the right, labeled '...', opens the a window allowing the known port list to be edited. Modifying the known port list only affects which ports the BASIC Stamp Editor tries to use; it does not affect which serial ports are installed on your computer. It is recommended that you delete all known modem ports and any problematic ports from this list.

For an explanation of the Default Project Download Modes, see Table 3.7 on page 70. This is part of a discussion on BASIC Stamp Projects in the Advanced Compilation Techniques beginning on page 68, below.

Selecting the "Ignore BS1 Modules unless downloading BS1 source code" checkbox optimizes identification speed by attempting to locate BS1 modules only if you are downloading BASIC Stamp 1 code. This feature can also be activated via the Identification or Download window.

THE FILES AND DIRECTORIES TAB.

Under the Files and Directories tab (Figure 3.23), you can set preferences for saving and accessing files, as well as automatically creating backup copies.

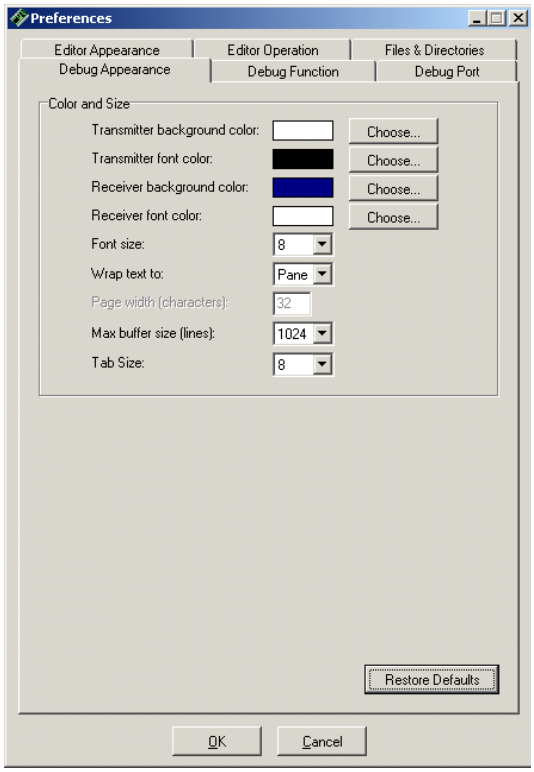


Figure 3.24: The Debug Appearance Tab under Edit → Preferences.

TEXT WRAPPING IN THE DEBUG TERMINAL.

The "Wrap Text to" field gives two options, Pane and Page. Wrapping to Pane is the default, and causes text to wrap at the right edge of the Receiver pane, reflecting the current visible size that the user happens to have set for the Debug Terminal's window. Wrapping to Page, however, causes text to wrap at a specific line width, regardless of the user's current Debug Terminal window size. The "Page width (characters)" field is enabled when wrap mode is set to Page. The default page width is 32, characters and the range is 32 to 128. Note: wrapping to page can be handy to maintain formatting of formatted tabular information, but could lead to information being displayed off the edge of the Receive pane if the Debug Terminal is sized too small.

MAXIMUM BUFFER SIZE.

The maximum Receive pane buffer size is defined in terms of lines. It can be set to any power of two between 256 and 8192; 1024 is the default. Data received by the Debug Terminal is maintained in this buffer for display on

```
result          VAR     Word

result = -99                            ' Put -99 into result
                                        ' ...(2's complement format)
DEBUG SDEC ? result                     ' Display as a signed #
DEBUG SDEC ? ABS result                 ' Display as a signed #
```

COSINE: COS

All 2

The Cosine operator (COS) returns the two's complement, 16-bit cosine of an angle specified as an 8-bit "binary radian" (0 to 255) angle. COS is the same as SIN in all respects, except that the cosine function returns the x distance instead of the y distance. See "Sine: SIN", below, for a code example and more information.

DECODER: DCD

All 2

The Decoder operator (DCD) is a 2^n-power decoder of a four-bit value. DCD accepts a value from 0 to 15, and returns a 16-bit number with the bit, described by value, set to 1. For example:

```
result    VAR   Word

result  =  DCD  12                    ' Set bit 12
DEBUG  BIN16 ?  result                ' Display result (%0001000000000000)
```

INVERSE: ~

The Inverse operator (~) complements (inverts) the bits of a number. Each bit that contains a 1 is changed to 0 and each bit containing 0 is changed to 1. This process is also known as a "bitwise NOT" and "ones complement". For example:

```
result    VAR   Byte

result = %11110001                    ' Store bits in byte result.
DEBUG  BIN8  ?  result                ' Display in binary (%11110001)
result = ~ result                     ' Complement result
DEBUG  BIN8  ?  Result                ' Display in binary (%00001110)
```

NEGATIVE: -

1

The Negative operator (-) negates a 16-bit number (converts to its twos complement).

```
SYMBOL  result          = W1

result = -99                            ' Put -99 into result
                                        ' ...(2's complement format)
result = result + 100                   ' Add 100 to it
DEBUG result                            ' Display result (1)
```

-- or --

DEBUG

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

1 All 2

DEBUG ***OutputData*** **{ , *OutputData* }**

Function

Display information on the PC screen within the BASIC Stamp Editor's Debug Terminal. This command can be used to display text or numbers in various formats on the PC screen in order to follow program flow (called debugging) or as part of the functionality of the BASIC Stamp application.

1

NOTE: Expressions are not allowed as arguments on the BS1. The only constant allowed for the BS1 DEBUG command is a text string.

- ***OutputData*** is a variable/constant/expression (0 – 65535) that specifies the information to output. Valid data can be ASCII characters (text strings and control characters), decimal numbers (0 - 65535), hexadecimal numbers ($0000 - $FFFF) or binary numbers (up to %1111111111111111). Data can be modified with special formatters as explained below.

Quick Facts

Table 5.9: DEBUG Quick Facts.

	BS1	BS2, BS2e, BS2sx BS2p, BS2pe	BS2px
Serial Protocol	Asynchronous 4800, N, 8, 1 True polarity Custom packetized format	Asynchronous 9600, N, 8, 1 Inverted polarity Raw data	Asynchronous 19200, N, 8, 1 Inverted polarity Raw data
Related Commands	None	SEROUT and DEBUGIN	

Explanation

DEBUG provides a convenient way for your BASIC Stamp to send messages to the PC screen while running. The name "debug" suggests its most popular use; debugging programs by showing you the value of a variable or expression, or by indicating what portion of a program is currently executing. DEBUG is also a great way to rehearse programming techniques. Throughout this manual, we use DEBUG to give you immediate feedback on the effects of instructions. The following example demonstrates using the DEBUG command to send the text string message "Hello World!".

```
DEBUG  "Hello, World!"
```

After you download this one-line program, the BASIC Stamp Editor will open a Debug Terminal on your PC screen and wait for a response from

but typing the name of the variables in quotes (for the display) can get a little tedious. A special formatter, the question mark (?), can save you a lot of time. The code below does exactly the same thing (with less typing):

```
x       VAR     Byte
y       VAR     Byte

x = 100
y = 250
DEBUG DEC ? x                           ' Show decimal value of x
DEBUG DEC ? y                           ' Show decimal value of y
```

The display would look something like this:

```
x = 100
y = 250
```

The ? formatter always displays data in the form "symbol = value" (followed by a carriage return). In addition, it defaults to displaying in decimal, so we really only needed to type: DEBUG ? x for the above code. You can, of course, use any of the three number systems. For example: DEBUG HEX ? x or DEBUG BIN ? y.

It's important to note that the "symbol" it displays is taken directly from what appears to the right of the ?. If you were to use an expression, for example: DEBUG ? x*10/2+3 in the above code, the display would show: "x*10/2+3 = 503".

A special formatter, ASC, is also available for use only with the ? formatter to display ASCII characters, as in: DEBUG ASC ? x.

DISPLAYING FIXED-WIDTH NUMBERS.

What if you need to display a table of data; multiple rows and columns? The Signed/Unsigned code (above) approaches this but, if you notice, the columns don't line up. The number formatters (DEC, HEX and BIN) have some useful variations to make the display fixed-width (see Table 5.12). Up to 5 digits can be displayed for decimal numbers. To fix the value to a specific number of decimal digits, you can use DEC1, DEC2, DEC3, DEC4 or DEC5. For example:

```
x       VAR     Byte

x = 165
DEBUG DEC5 x                            ' Show decimal value of x in 5 digits
```

Table 5.17: DEBUGIN Conversion Formatters.

Conversion Formatter	Type of Number	Numeric Characters Accepted	Notes
DEC{1..5}	Decimal, optionally limited to 1 – 5 digits	0 through 9	1
SDEC{1..5}	Signed decimal, optionally limited to 1 – 5 digits	-, 0 through 9	1,2
HEX{1..4}	Hexadecimal, optionally limited to 1 – 4 digits	0 through 9, A through F	1,3,5
SHEX{1..4}	Signed hexadecimal, optionally limited to 1 – 4 digits	-, 0 through 9, A through F	1,2,3
IHEX{1..4}	Indicated hexadecimal, optionally limited to 1 – 4 digits	$, 0 through 9, A through F	1,3,4
ISHEX{1..4}	Signed, indicated hexadecimal, optionally limited to 1 – 4 digits	-, $, 0 through 9, A through F	1,2,3,4
BIN{1..16}	Binary, optionally limited to 1 – 16 digits	0, 1	1
SBIN{1..16}	Signed binary, optionally limited to 1 – 16 digits	-, 0, 1	1,2
IBIN{1..16}	Indicated binary, optionally limited to 1 – 16 digits	%, 0, 1	1,4
ISBIN{1..16}	Signed, indicated binary, optionally limited to 1 – 16 digits	-, %, 0, 1	1,2,4
NUM	Generic numeric input (decimal, hexadecimal or binary); hexadecimal or binary number must be indicated	$, %, 0 through 9, A through F	1, 3, 4
SNUM	Similar to NUM with value treated as signed with range -32768 to +32767	-, $, %, 0 through 9, A through F	1,2,3,4

1 All numeric conversions will continue to accept new data until receiving either the specified number of digits (ex: three digits for DEC3) or a non-numeric character.

2 To be recognized as part of a number, the minus sign (-) must immediately precede a numeric character. The minus sign character occurring in non-numeric text is ignored and any character (including a space) between a minus and a number causes the minus to be ignored.

3 The hexadecimal formatters are not case-sensitive; “a” through “f” means the same as “A” through “F”.

4 Indicated hexadecimal and binary formatters ignore all characters, even valid numerics, until they receive the appropriate prefix ($ for hexadecimal, % for binary). The indicated formatters can differentiate between text and hexadecimal (ex: ABC would be interpreted by HEX as a number but IHEX would ignore it unless expressed as $ABC). Likewise, the binary version can distinguish the decimal number 10 from the binary number %10. A prefix occurring in non-numeric text is ignored, and any character (including a space) between a prefix and a number causes the prefix to be ignored. Indicated, signed formatters require that the minus sign come before the prefix, as in -$1B45.

5 The HEX modifier can be used for Decimal to BCD Conversion. See “Hex to BCD Conversion” on page 97.

For examples of all conversion formatters and how they process incoming data, see Appendix C.

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


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


Setup:
  sample = 11500

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

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

All 2

Demo Program (IF-THEN-ELSE.bs2)

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

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

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

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


Setup:
  sample = 11500

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

LOOKUP

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

1 **LOOKUP *Index*, (*Value0, Value1, …ValueN*), *Variable***

All 2 **LOOKUP *Index*, [*Value0, Value1, …ValueN*], *Variable***

Function

Find the value at location *Index* and store it in *Variable*. If *Index* exceeds the highest index value of the items in the list, *Variable* is left unaffected.

1 NOTE: Expressions are not allowed as arguments on the BS1.

- ***Index*** is a variable/constant/expression (0 – 255) indicating the list item to retrieve.
- ***Values*** are variables/constants/expressions (0 – 65535).
- ***Variable*** is a variable that will be set to the value at the *Index* location. If *Index* exceeds the highest location number, *Variable* is left unaffected.

Quick Facts

Table 5.55: LOOKUP Quick Facts.

	BS1 and all BS2 Models
Limit of *Value* Entries	256
Starting Index Number	0
If index exceeds the highest location…	Variable is left unaffected
Related Command	LOOKDOWN

Explanation

LOOKUP retrieves an item from a list based on the item's position, *Index,* in the list. For example:

1
```
SYMBOL  index   = B2
SYMBOL  result  = B3

index = 3
result = 255

LOOKUP index, (26, 177, 13, 1, 0, 17, 99), result
DEBUG  "Item ", #index, "is: ", #result
```

-- or --

POLLIN – BASIC Stamp Command Reference

RANDOM

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

RANDOM ***Variable***

Function

Generate a pseudo-random number.

- ***Variable*** is a variable (usually a word) whose bits will be scrambled to produce a random number. *Variable* acts as RANDOM's input and its result output. Each pass through RANDOM stores the next number, in the pseudorandom sequence, in *Variable*.

Explanation

RANDOM generates pseudo-random numbers ranging from 0 to 65535. They're called "pseudo-random" because they appear random, but are generated by a logic operation that uses the initial value in *Variable* to "tap" into a sequence of 65535 essentially random numbers. If the same initial value, called the "seed", is always used, then the same sequence of numbers is generated. The following example demonstrates this:

```
SYMBOL  result           = W1

Main:
  result = 11000
  RANDOM result
  DEBUG result
  GOTO Main
```

-- or --

```
result          VAR     Word

Main:
  result = 11000
  RANDOM result
  DEBUG DEC ? result
  GOTO Main
```

In this example, the same number would appear on the screen over and over again. This is because the same seed value was used each time; specifically, the first line of the loop sets *result* to 11,000. The RANDOM command really needs a different seed value each time. Moving the "result =" line out of the loop will solve this problem, as in:

SELECT…CASE – BASIC Stamp Command Reference

```
  #CASE BS2SX, BS2P
    T1200       CON     2063
    T2400       CON     1021
    T9600       CON     240
    T19K2       CON     110
    T38K4       CON     45
  #CASE BS2PX
    T1200       CON     3313
    T2400       CON     1646
    T9600       CON     396
    T19K2       CON     188
    T38K4       CON     84
#ENDSELECT

Inverted        CON     $4000
Open            CON     $8000
Baud            CON     T38K4 + Inverted

letter          VAR     Byte


Main:
  DO
    SERIN SI\FC, Baud, [letter]         ' receive one byte
    DEBUG letter                        ' display on screen
    PAUSE 1000                          ' wait one second
  LOOP                                  ' repeat forever
  END
```

STOP – BASIC Stamp Command Reference

TOGGLE

BS1	BS2	BS2e	BS2sx	BS2p	BS2pe	BS2px

1 All 2

TOGGLE ***Pin***

Function

Invert the state of an output pin.

- ***Pin*** is a variable/constant/expression (0 – 15) that specifies which I/O pin to switch logic state. This pin will be placed into output mode.

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

Quick Facts

Table 5.122: TOGGLE Quick Facts.

	BS1	All BS2 Models
Affected Register	PINS	OUTS
Related Commands	HIGH and LOW	

Explanation

TOGGLE sets a pin to output mode and inverts the output state of the pin, changing 0 to 1 and 1 to 0.

In some situations TOGGLE may appear to have no effect on a pin's state. For example, suppose pin 2 is in input mode and pulled to +5V by a 10k resistor. Then the following code executes:

1
```
DIR2 = 0                                ' make P2 an input
PIN2 = 0                                ' make P2 output driver low
DEBUG PIN2                              ' show P2 state (1 due to pull-up)
TOGGLE 2                                ' toggle P2
DEBUG PIN2                              ' show P2 state (1 again)
```

- or -

All 2
```
DIR2 = 0                                ' make P2 an input
OUT2 = 0                                ' make P2 output driver low
DEBUG ? IN2                             ' show P2 state (1 due to pull-up)
TOGGLE 2                                ' toggle P2
DEBUG ? IN2                             ' show P2 state (1 again)
```

The state of pin 2 doesn't change; it's high (due to the resistor) before TOGGLE, and it's high (due to the pin being output high) afterward. The point is that TOGGLE works on the OUTS register (PINS on the BS1), which may not match the pin's state when the pin is initially an input. To

guarantee that the state actually changes, regardless of the initial input or output mode, do this:

```
PIN2 = PIN2                             ' make output driver match input
TOGGLE 2                                ' then toggle
```

- or -

```
OUT2 = IN2                              ' make output driver match input
TOGGLE 2                                ' then toggle
```

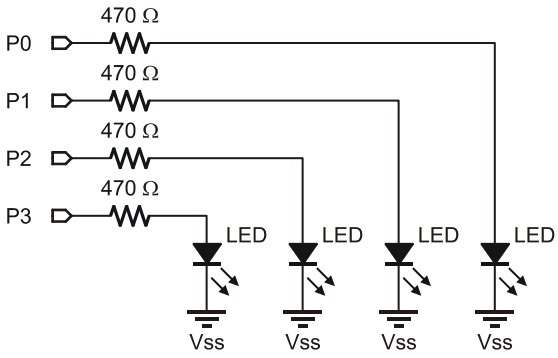


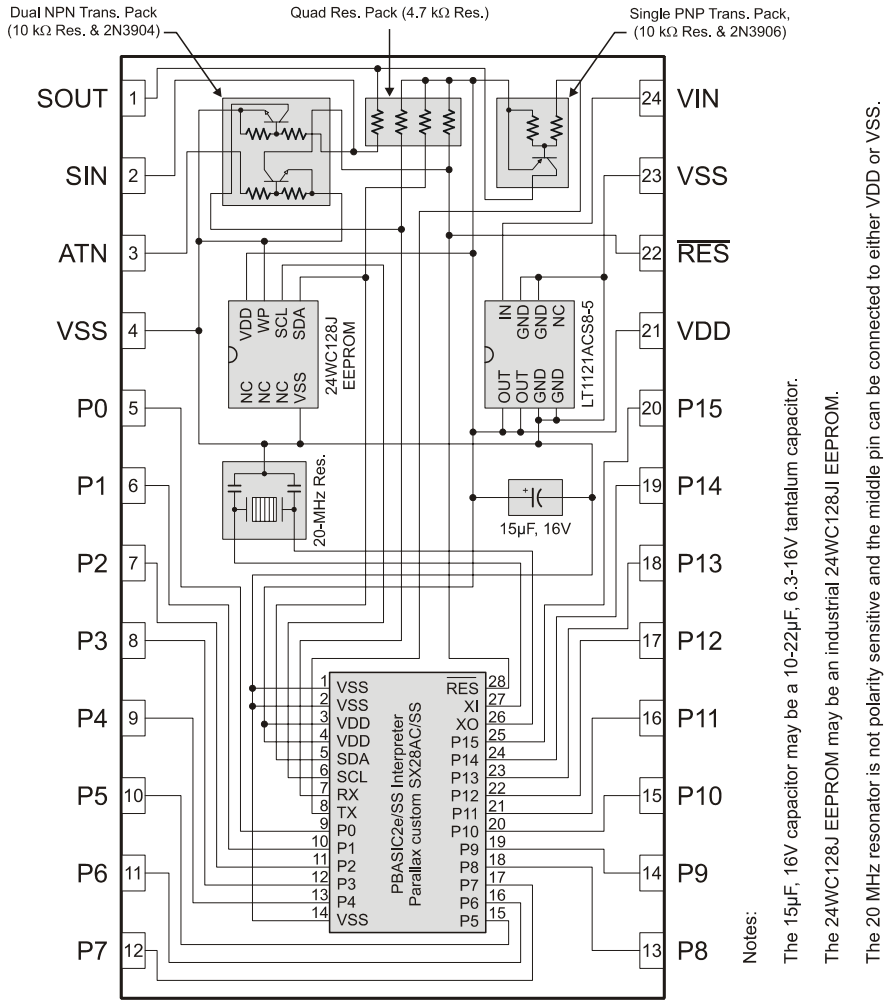
Figure 5.47: Example LED Circuit for TOGGLE Demo Programs.

Demo Program (TOGGLE.bs1)

```
' TOGGLE.bs1
' Connect LEDs to pins 0 through 3 as shown in the TOGGLE command descrip-
' tion in the manual and run this program. The TOGGLE command will treat
' you to a light show. You may also run the demo without LEDs. The Debug
' window will show you the states of pins 0 through 3.

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

SYMBOL  thePin          = B0            ' pin 0 - 3

Setup:
  DIRS = %1111                          ' make LEDs output, low

Main:
  FOR thePin = 0 TO 3                   ' loop through pins
    TOGGLE thePin                       ' toggle current pin
    DEBUG CLS, %PINS                    ' show on Debug
    PAUSE 100                           ' short delay
  NEXT
  GOTO Main                             ' repeat forever
  END
```

TOGGLE – BASIC Stamp Command Reference

BASIC Stamp 2e Schematic (Rev B)



Notes:

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

The 24WC128J EEPROM may be an industrial 24WC128JI EEPROM.

The 20 MHz resonator is not polarity sensitive and the middle pin can be connected to either VDD or VSS.

— S —