Welcome to E-XFL.COM

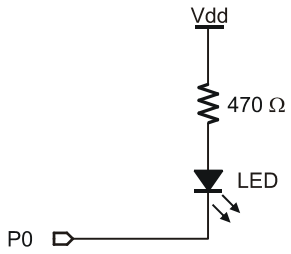
Understanding Embedded - Microcontroller, Microprocessor, FPGA Modules

Embedded - Microcontroller, Microprocessor, and FPGA Modules are fundamental components in modern electronic systems, offering a wide range of functionalities and capabilities. Microcontrollers are compact integrated circuits designed to execute specific control tasks within an embedded system. They typically include a processor, memory, and input/output peripherals on a single chip. Microprocessors, on the other hand, are more powerful processing units used in complex computing tasks, often requiring external memory and peripherals. FPGAs (Field Programmable Gate Arrays) are highly flexible devices that can be configured by the user to perform specific logic functions, making them invaluable in applications requiring customization and adaptability.

Applications of Embedded - Microcontroller,

Details	
Product Status	Active
Module/Board Type	MCU Core
Core Processor	SX48AC
Co-Processor	-
Speed	8MHz
Flash Size	32KB EEPROM
RAM Size	38B
Connector Type	-
Size / Dimension	1.2" x 0.6" (30mm x 15mm)
Operating Temperature	0°C ~ 70°C
Purchase URL	https://www.e-xfl.com/product-detail/parallax/bs2pe

Figure 5.18: Example LED Circuit.



1 All 2

NOTE: This example program can be used with the BS1 and all BS2 models by changing the $STAMP directive accordingly.

Demo Program (NAP.bs2)

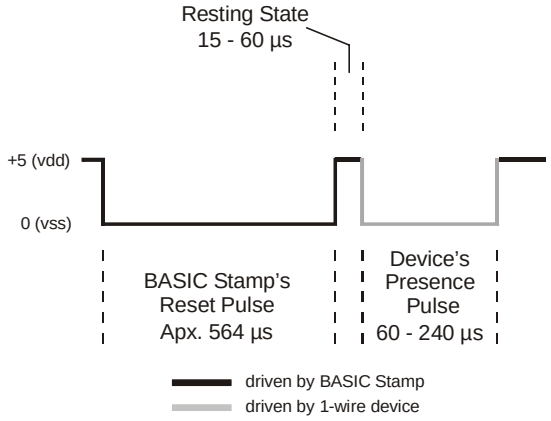
```
' NAP.bs2
' The program below lights an LED by placing a low on pin 0. This completes
' the circuit from +5V, through the LED and resistor, to ground. During the
' NAP interval, the LED stays lit, but blinks off for a fraction of a sec.
' This blink is caused by the NAP wakeup mechanism during wakeup, all pins
' briefly slip into input mode, effectively disconnecting them from loads.

' {$STAMP BS2}


Setup:
  LOW 0                        ' turn LED on

Snooze:
  NAP 4                        ' nap for 288 ms
  GOTO Snooze
  END
```

THE 1-WIRE PROTOCOL FORMAT.

The 1-Wire protocol has a well-defined standard for transaction sequences. Every transaction sequence consists of four parts: 1) Initialization, 2) ROM Function Command, 3) Memory Function Command, and 4) Transaction/Data. Additionally, the ROM Function Command and Memory Function Command are always 8 bits wide (1 byte in size) and is sent least-significant-bit (LSB) first.

The Initialization part consists of a reset pulse (generated by the master) and will be followed by a presence pulse (generated by all slave devices). Figure 5.20 details the reset pulse generated by the BASIC Stamp and a typical presence pulse generated by a 1-wire slave, in response.

Figure 5.20: OWIN Reset and Presence Pulse.



This reset pulse is controlled by the lowest two bits of the Mode argument in the OWIN command. It can be made to appear before the ROM Function Command (ex: Mode= 1), after the Transaction/Data portion (ex: Mode= 2), before and after the entire transaction (ex: Mode= 3) or not at all (ex: Mode= 0). See the section on Mode, above, for more information.

Following the Initialization part is the ROM Function Command. The ROM Function Command is used to address the desired 1-Wire device. Table 5.67 shows common ROM Function Commands. If only a single 1-Wire device is connected, the Skip ROM command may be used to address it. If more than one 1-Wire device is attached, the BASIC Stamp will ultimately have to address them individually using the Match ROM command.

NOTE: This example program can be used with the BS2p, BS2pe, and BS2px by changing the $STAMP directive accordingly.

Demo Program (OWIN_OWOUT.bsp)

```
' OWIN_OWOUT.bsp
' This program demonstrates interfacing to a Dallas Semiconductor DS1822
' 1-Wire Digital Thermometer chip using the BS2p's 1-Wire commands. Connect
' the BS2p, BS2pe or BS2px to the DS1822 as shown in the diagram in the
' OWIN or OWOUT command description. This program uses a simplified
' approach that ignores the fractional portion of the temperature.

' {$STAMP BS2p}
' {$PBASIC 2.5}

DQ          PIN    0          ' 1-Wire buss pin

RdROM       CON    $33        ' read serial number
MatchROM    CON    $55        ' match SN -- for multiple devices
SkipROM     CON    $CC        ' ignore SN -- use for one device
CvrtTmp     CON    $44        ' start temperature conversion
RdSP        CON    $BE        ' read DS1822 scratch pad

tempIn      VAR    Word       ' raw temperature
sign        VAR    tempIn.BIT11   ' 1 = negative temperature
tLo         VAR    tempIn.BYTE0
tHi         VAR    tempIn.BYTE1
tSign       VAR    Bit        ' saved sign bit
tempC       VAR    Word       ' final Celsius temp
tempF       VAR    Word       ' final Fahrenheit temp


Main:
 DO
  GOSUB Get_Temperature          ' read temperature from DS1822
  DEBUG HOME,                    ' display
      "DS1822", CR,
      "------", CR,
      SDEC tempC, " C ", CR,
      SDEC tempF, " F "
  PAUSE 1000
 LOOP
 END

Get_Temperature:
 OWOUT DQ, 1, [SkipROM, CvrtTmp]     ' send convert temperatrue command
 DO                                  ' wait on conversion
  PAUSE 25                           ' small loop pad
  OWIN DQ, 4, [tempIn]               ' check status (bit transfer)
 LOOP UNTIL (tempIn)                 ' 1 when complete
 OWOUT DQ, 1, [SkipROM, RdSP]        ' read DS1822 scratch pad
 OWIN  DQ, 2, [tLo, tHi]             ' get raw temp data
 tSign = sign                        ' save sign bit
 tempC = tempIn >> 4                 ' round to whole degrees
 tempC.BYTE1 = $FF * tSign           ' correct twos complement bits
```

(greater than the Maximum Pulse Width shown in Table 5.83) PULSIN "times out" and store 0 in Variable. This operation keeps your program from locking-up should the desired pulse never occur.

HOW THE RESULT IS REPORTED.

Regardless of the size of Variable, PULSIN internally uses a 16-bit timer. Unless the pulse widths are known to be short enough to fit in an 8-bit result, it is recommended to use a word-sized variable. Not doing so may result in strange and misleading results as the BASIC Stamp will only store the lower 8-bits into a byte variable.

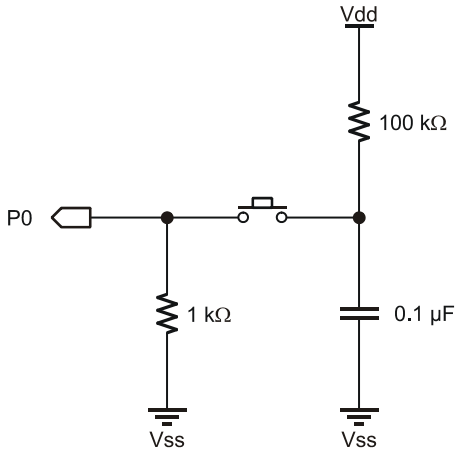


Figure 5.28: R/C Pulse Generator.

Demo Program (PULSIN.bs1)

```
' PULSIN.bs1
' This program uses PULSIN to measure a pulse generated by discharging a
' 0.1 uF capacitor through a 1k resistor. Pressing the switch generates
' the pulse, which should ideally be approximately 120 us (12 PULSIN units
' of 10 us) long. Variations in component values may produce results that
' are up to 10 units off from this value. For more information on
' calculating resistor-capacitor timing, see the RCTIME command.

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

SYMBOL  Pulse        = 7          ' pulse input pin

SYMBOL  time         = W1         ' pulse width (10 uS units)
```

it would see the RETURN again (although it didn't GOSUB to that routine this time) and because there wasn't a previous place to return to, the BASIC Stamp will start the entire program over again. This would cause an endless loop. The important thing to remember here is to always make sure your program doesn't allow itself to "fall into" a subroutine.

Demo Program (RETURN.bs2)

1 All 2

NOTE: This example program can be used with the BS1 and all BS2 models by changing the $STAMP directive accordingly.

```
' RETURN.BS2
' This program demonstrates a potential bug caused by allowing a program to
' "fall into" a subroutine.  The program was intended to indicate that it
' is "Starting...", then "Executing Subroutine,", then "Returned..." from
' the subroutine and stop.  Since we left out the END command (indicated in
' the comments), the program then falls into the subroutine, displays
' "Executing..." again and then RETURNs to the start of the program and
' runs continuously in an endless loop.

' {$STAMP BS2}


Reset:
  DEBUG "Starting Program", CR          ' show start-up

Main:
  PAUSE 1000
  GOSUB Demo_Sub                        ' call the subroutine
  PAUSE 1000
  DEBUG "Returned from Subroutine", CR  ' show that we're back
  PAUSE 1000
                                        ' <-- Forgot to put END here

Demo_Sub:
  DEBUG "  Executing Subroutine", CR    ' show subroutine activity
  RETURN
```