Column #119 March 2005 by Jon Williams:

Ping – I See You

I used to work for a man named Bob who insisted – and quite frequently – that most of us needed to be exposed to the same piece of information five to seven times before that information could be absorbed. I didn't always agree with Bob's philosophy, but just in case he's right I thought we'd work through the mysteries of conditional compilation again. Conditional compilation is worth mastering; it lets us write one program that will work on nearly any BASIC Stamp module.

Maybe I'm just taking things for granted. Being on "the inside" and close to the development of the BASIC Stamp Editor IDE, I completely understand conditional compilation and how to take advantage of it. Apparently, however, I haven't done a very good job getting the good word out – I keep getting a lot of question on this subject. So, I'm going to try again.

Let's start from the beginning. Why should we even bother with conditional compilation? Well, it depends, really. If we're going to write a program that will NEVER (yeah, right...) need to run on another BS2-family module then we don't need to bother. But ... what if we want to share our cool program with a friend who uses a different module? And what if we wrote our program for the BS2 and our friend is using a BS2sx? Most programs will run without change, but the use of certain PBASIC keywords will require the code to be updated

to run properly on the BS2sx. By using conditional compilation up front we can save ourselves and others trouble later.

Ping... Ping...

Before we getting into the gritty details, let's have a little bit of fun first with a simple program that actually uses conditional compilation. Sonic range-finding modules are very popular with robotics builders and experimenters, and Parallax has recently created a new module called Ping))) that makes sonar range-finding pretty easy. Honestly, I really like the Ping))) sensor; it requires only one IO pin, it works with any BASIC Stamp module, and is very low cost.

As you can see by Figure 119.1, the connection is a no-brainer – connect power (+5 volts), ground (Vss), and a signal line to a free BASIC Stamp pin. With the Ping))) module, the IO pin serves as both the trigger output and the echo input.

Figure 119.1: Ping))) Module to BASIC Stamp Connection

Figure 119.2: Ping))) Timing Diagram

Initially, the trigger pin is made an output and a short pulse (5 to 10 µS) is used to trigger the Ping))) (we'll use PULSOUT to generate the trigger). The next step is what allows it to be used with any BASIC Stamp. The Ping))) module delays the trigger to the sonic transmitter element for 500 microseconds. This allows the BASIC Stamp to load the next instruction (PULSIN) and be ready for the return echo. Once the echo pulse in measured, a bit of math is used to convert the pulse width to distance.

Let's look at the subroutine which handles the Ping))) sensor:

```
Get_Sonar:
  Ping = 0
  PULSOUT Ping, Trigger
  PULSIN  Ping, 1, rawDist
  rawDist = rawDist */ Scale
  rawDist = rawDist / 2
  RETURN
```

The code starts by making the output bit of the trigger pin 0. The reason for this is that PULSOUT makes the trigger pin an output, then toggles its state, delays, then toggles that pin back to the original state. Since the Ping))) module is looking for a low-high-low pulse to trigger the measurement, presetting the pin to 0 makes this happen.

After the trigger is sent PULSIN is used to measure the width of the echo pulse. As I stated earlier, the 500 microsecond delay in the Ping))) allows PULSIN to get loaded and ready. There is no danger of PULSIN timing out as even the BS2p (fastest BASIC Stamp module) won't time out for about 49 milliseconds. And for you clever readers that are wondering what happens if we forget to make the signal pin an input after the trigger pulse ... no worries, there is protection on the Ping))) sensor so that no harm is done if both sides are trying to drive the signal line.

And now we have to get back to that pesky conditional compilation stuff. Remember that the various BASIC Stamp modules run at different speeds, and with some instructions the speed differences give us different resolution. Let's look at the units returned by PULSIN:

BS2, BS2e:	2.00 µs
BS2sx, BS2p:	0.80 µs
BS2pe:	1.88 µs

And now let's see how conditional compilation lets us handle the differences in the various modules:

```
#SELECT $STAMP
  #CASE BS2, BS2E
    Scale       CON     $200
  #CASE BS2SX, BS2P
    Scale       CON     $0CD
  #CASE BS2PE
    Scale       CON     $1E1
#ENDSELECT
```

The instructions prefaced with # are used in the conditional compilation process. These instructions actually get processed before our program is tokenized. This allows constant values and even bits of code if we choose, to be included in the program based on the BASIC Stamp module in use. So, using the code above, if a stock BS2 module is installed, the constant called Scale will have the value $200. If we unplug the BS2 and swap in a BS2p, when we program the module Scale will have the value $0CD.

Let's get back to the program – we'll cover more conditional compilation later. The raw value from PULSIN is converted to units of one microsecond with this line of code:

```
  rawDist = rawDist */ Scale
```

We're forced to use the */ (star-slash) operator to account for the fractional units when using the BS2sx, BS2p, or BS2pe. For review, */ works like multiplication but in units of 1/256. To determine the various values for Scale, we multiply the PULSIN units by 256 and take the [rounded] integer result. Things work out like this:

BS2, BS2e	INT(2.00 x 256) = 512 ($200)
BS2sx, BS2p	INT(0.80 x 256) = 205 ($0CD)
BS2pe	INT(1.88 x 256) = 481 ($1E1)

I prefer to use hex notation for values that are used with */ as the upper byte represents the whole portion of the value, and the lower by the fractional portion (in units of 1/256).

Okay, the pulse is measured and converted to microseconds. Before returning to the caller we'll divide the raw value by two. Why? Well, the pulse we've just measured actually accounts for the distance to and from the target – actually twice as wide as we need it, hence the division.

Now to convert to distance: At sea level and room temperature we assume that sound travels at about 1130 feet per second; by multiplying by 12 we get 13560 inches per second. By taking the reciprocal we find that it takes about 73.746 microseconds for sound to travel one inch. For those that prefer the metric system we can convert 13560 inches to 34442 centimeters, and a timing value of 29.034 microseconds to travel one centimeter.

For our conversion code we'll use the other fraction math operator, **. This is similar to */, except that it uses units of 1/65536 (which means in the 16-bit values used by the BASIC Stamp we can use it to multiply by fractional values of less than one). In our program we can convert 73.746 microseconds to a constant value like this:

$1 / 73.746 \rightarrow \text{INT}(0.01356 \times 65536) = 889$ ($379)

With that we can look at the rest of the program:

```
Reset:
  DEBUG CLS,
        "Parallax Ping Sonar", CR,
        "-------------------", CR,
        CR,
        "Time (uS).....     ", CR,
        "Inches........     ", CR,
        "Centimeters...     "
Main:
  DO
    GOSUB Get_Sonar
    inches = rawDist ** RawToIn
    cm = rawDist ** RawToCm
    DEBUG CRSRXY, 15, 3,
          DEC rawDist, CLREOL
    DEBUG CRSRXY, 15, 4,
          DEC inches, CLREOL
    DEBUG CRSRXY, 15, 5,
          DEC cm, CLREOL
```

```
    PAUSE 100
  LOOP
  END
```

The Reset section simply sets up the text portion of the Debug Terminal window, and in Main we measure the distance, do the conversions, and display the results. Figure 119.3 shows the output of the program.

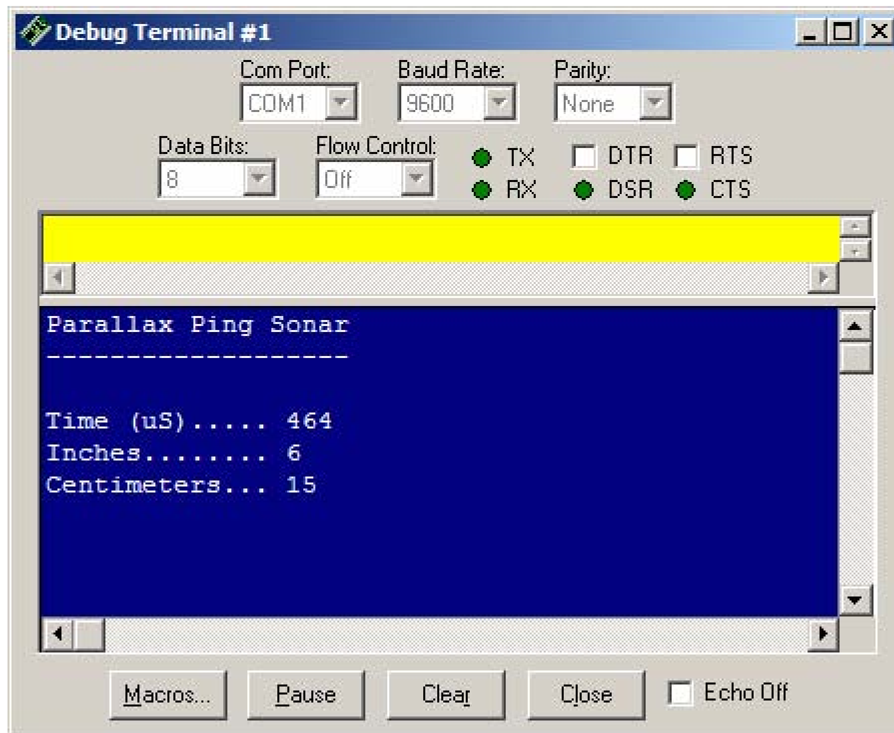

Figure 119.3: Ping))) Debug Terminal Output

Stamping Under Any Condition

Okay, time to get back to conditional compilation. While most PBASIC instructions do not require parameter changes when moving from one module to another, there are a few that do:

COUNT	Units for Duration of COUNT window
DTMFOUT	Units for OnTime
FREQOUT	Units for Duration, Freq1, and Freq2
PULSIN	Units for Variable (measured pulse)
PULSOUT	Units for Duration
PWM	Units for Duration
RCTIME	Units for Variable (measured RC delay)
SERIN	Units in Timeout, value of Baudmode
SEROUT	Units in Pace and Timeout, value of Baudmode

The most common issue encountered by BASIC Stamp users when moving from module to module is with SERIN and SEROUT. So common are these instructions that I have built the following section into my default programming template:

```
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T1200       CON     813
    T2400       CON     396
    T4800       CON     188
    T9600       CON     84
    T19K2       CON     32
    TMidi       CON     12
    T38K4       CON     6
  #CASE BS2SX, BS2P
    T1200       CON     2063
    T2400       CON     1021
    T4800       CON     500
    T9600       CON     240
    T19K2       CON     110
    TMidi       CON     60
    T38K4       CON     45
#ENDSELECT

SevenBit        CON     $2000
Inverted        CON     $4000
Open            CON     $8000

Baud            CON     T9600
```

If SERIN and SEROUT aren't used by a given program there is no harm done – and it's far handier to have constants predefined than to have to look them up. And this gives me the opportunity to bring up another programming tip. I frequently get code that looks like this:

```
  SEROUT 15, 16468, [DEC temperature]
```

... which is followed by the complaint, "Jon, this used to work with my BS2, but now it doesn't work with my BS2sx."

By now I'm sure the reason is obvious: by changing from the BS2 to a BS2sx we are forced to update the baudmode parameter of SEROUT. The problem can be averted by using the conditional section above and changing the Baud definition as follows:

```
Baud            CON     Inverted + T9600
```

And while we're cleaning up the code to make it easier to maintain, let's give a definition to P15 so that we know the serial output is going to a serial LCD:

```
Lcd             PIN     15
```

And now the corrected code becomes:

```
  SEROUT Lcd, Baud, [DEC temperature]
```

Where else might conditional compilation come in handy? How about program debugging? There is an instruction called #DEFINE that can help in this regard. For example:

```
#DEFINE DebugOn = 1
```

While developing and troubleshooting an application we can do this:

```
#IF DebugOn #THEN
  DEBUG "Value = ", DEC value, CR
#ENDIF
```

... in as many places in the program as we need.

Once the program is fully tested and working as desired, changing the DebugOn definition to zero will prevent the DEBUG statements in the #IF-#THEN section(s) from executing. It's important to understand that conditional definitions are either defined (not zero) or not. In our example above we could in fact remove the #DEFINE DebugOn line without harm to the

program. When the compiler encounters a conditional block (like #IF-#THEN) with an undefined symbol, the section is skipped. I don't recommend this, however, as it can lead to confusion if someone else reads code from which we've removed conditional symbol definitions. It is best to disable the conditional symbol by redefining it as zero.

Another good use of conditional definitions is variable conservation. In our sonar program, for example, practical use would usually not require both English and Metric units. We could do this:

```
#DEFINE MetricUnits = 1

and...

#IF MetricUnits #THEN
  distance = rawDist ** RawToCm
#ELSE
  distance = rawDist ** RawToIn
#ENDIF
```

Finally, what about features that exist in the newer BASIC Stamp modules that do not exist in the older – LCD control, for example? Well, we can deal with that too.

There was a project we did some time back that involved the Parallax LCD Terminal AppMod that took advantage of conditional compilation. A program can check for the availability of built-in LCD commands like this:

#DEFINE LcdReady = ($STAMP >= BS2P)

We can now put this definition to use in the following manner:

```
LCD_Command:
  #IF LcdReady #THEN
    LCDCMD E, char
    RETURN
  #ELSE
    LOW RS
    GOTO LCD_Write
  #ENDIF

LCD_Write:
  #IF LcdReady #THEN
    LCDOUT E, 0, [char]
```

```
#ELSE
  LcdBusOut = char.HIGHNIB
  PULSOUT E, 3
  LcdBusOut = char.LOWNIB
  PULSOUT E, 3
  HIGH RS
#ENDIF
RETURN
```

Yes, it takes a little bit of planning and extra work to implement conditional compilation, but in the end I think you'll find it fairly simple to do, and a big time-saver when it comes to moving code from one BASIC Stamp module to another.

Installing a Template

Earlier I mentioned my default template and its use of serial baudmode values. I've included a copy of my template in the project files, and let me share a tip that may not be obvious. You can have the BASIC Stamp IDE load this template each time you select File / New (or the New icon from the toolbar).

Start by copying the template (template.bs2) to a convenient location. The open the Preferences dialog (Edit / Preferences), select the Files & Directories tab, and then click on the Browse button located net to the New File Template field. In the Open dialog, navigate to the location where you copied the template file, select it, and then click on Open. Lock in the setting by clicking OK at the bottom of the Preferences dialog. I find the template helps keep my programs organized and I'm sure it will work for you too – if it doesn't quite, modify it until it does!

Until next time, then, Happy Stamping.

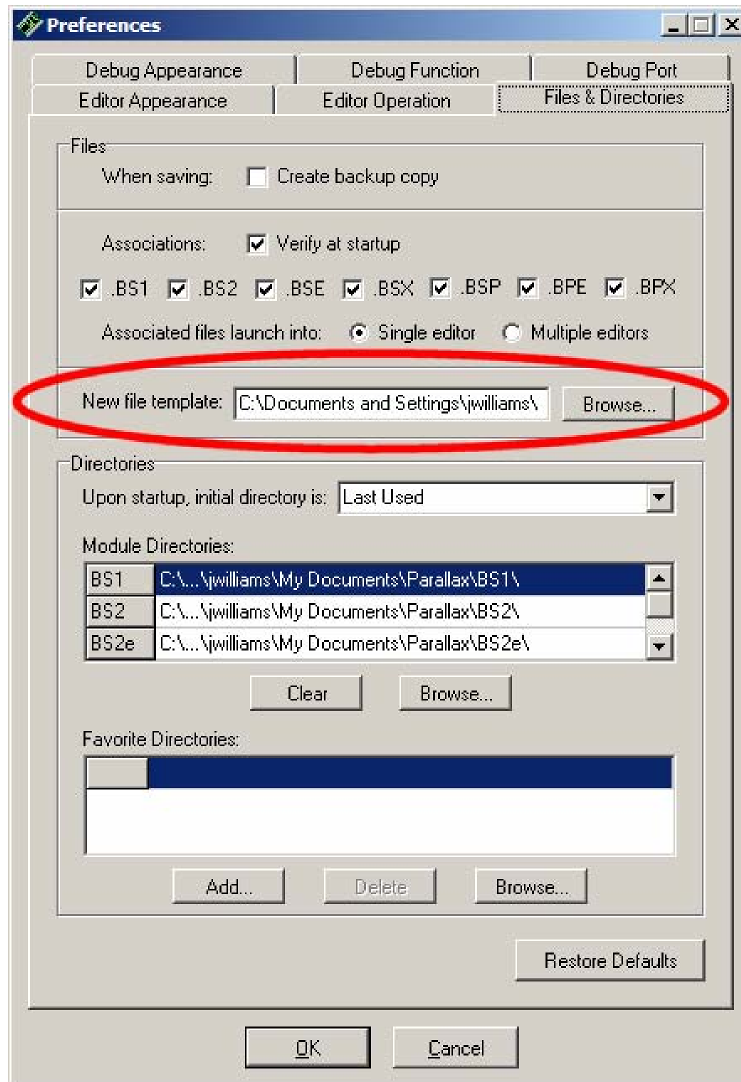

Figure 119.4: Configure the BASIC Stamp Windows Editor for a Template

```
' =========================================================================
'
'   File....... Ping_Demo.BS1
'   Purpose.... Demo Code for Parallax Ping))) Sonar Sensor
'   Author..... Jon Williams -- Parallax, Inc.
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 11 JAN 2005
'
'   {$STAMP BS1}
'   {$PBASIC 1.0}
'
' =========================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program demonstrates the use of the Parallax Ping Sonar sensor and
' converting the raw measurement to English (inches) and Metric (cm) units.
'
' Sonar Math:
'
' At sea level sound travels through air at 1130 feet per second.  This
' equates to 1 inch in 73.746 uS, or 1 cm in 29.034 uS.
'
' Since the Ping sensor measures the time required for the sound wave to
' travel from the sensor and back.  The result -- after conversion to
' microseconds for the BASIC Stamp module in use -- is divided by two to
' remove the return portion of the echo pulse.  The final raw result is
' the duration from the front of the sensor to the target in microseconds.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

SYMBOL  Ping            = 7


' -----[ Constants ]-------------------------------------------------------

SYMBOL  Trigger         = 1                     ' 10 uS trigger pulse
SYMBOL  Scale           = 10                    ' raw x 10.00 = uS

SYMBOL  RawToIn         = $0379                 ' 1 / 73.746
SYMBOL  RawToCm         = $08D1                 ' 1 / 29.034


' -----[ Variables ]-------------------------------------------------------
```

```
SYMBOL  rawDist         = W1                    ' raw measurement
SYMBOL  inches          = W2
SYMBOL  cm              = W3


' -----[ EEPROM Data ]----------------------------------------------------


' -----[ Initialization ]-------------------------------------------------

Reset:


' -----[ Program Code ]---------------------------------------------------

Main:
  GOSUB Get_Sonar                               ' get sensor value
  inches = rawDist ** RawToIn                   ' convert to inches
  cm = rawDist ** RawToCm                       ' convert to centimeters

  DEBUG CLS                                     ' report
  DEBUG "Time (uS)..... ", #rawDist, "    ", CR
  DEBUG "Inches........ ", #inches, "    ", CR
  DEBUG "Centimeters... ", #cm, "    "

  PAUSE 1000
  GOTO Main

  END


' -----[ Subroutines ]----------------------------------------------------

' This subroutine triggers the Ping sonar sensor and measures
' the echo pulse.  The raw value from the sensor is converted to
' microseconds based on the BASIC Stamp module in use.  This value is
' divided by two to remove the return trip -- the result value is
' the distance from the sensor to the target in microseconds.

Get_Sonar:
  LOW Ping                                      ' make trigger 0-1-0
  PULSOUT Ping, Trigger                         ' activate sensor
  PULSIN  Ping, 1, rawDist                      ' measure echo pulse
  rawDist = rawDist * Scale                     ' convert to uS
  rawDist = rawDist / 2                         ' remove return trip
  RETURN
```

```
' =========================================================================
'
'   File....... Ping_Demo.BS2
'   Purpose.... Demo Code for Parallax Ping))) Sonar Sensor
'   Author..... Jon Williams -- Parallax, Inc.
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 11 JAN 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =========================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program demonstrates the use of the Parallax Ping Sonar sensor and
' converting the raw measurement to English (inches) and Metric (cm) units.
'
' Sonar Math:
'
' At sea level sound travels through air at 1130 feet per second.  This
' equates to 1 inch in 73.746 uS, or 1 cm in 29.034 uS).
'
' Since the Ping sensor measures the time required for the sound wave to
' travel from the sensor and back.  The result -- after conversion to
' microseconds for the BASIC Stamp module in use -- is divided by two to
' remove the return portion of the echo pulse.  The final raw result is
' the duration from the front of the sensor to the target in microseconds.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

Ping            PIN     15


' -----[ Constants ]-------------------------------------------------------

#SELECT $STAMP
  #CASE BS2, BS2E
    Trigger     CON     5                       ' trigger pulse = 10 uS
    Scale       CON     $200                    ' raw x 2.00 = uS
  #CASE BS2SX, BS2P
    Trigger     CON     13
    Scale       CON     $0CD                    ' raw x 0.80 = uS
  #CASE BS2PE
    Trigger     CON     5
```

```
   Scale       CON     $1E1                    ' raw x 1.88 = uS
#ENDSELECT

RawToIn         CON     $0379                   ' 1 / 73.746
RawToCm         CON     $08D1                   ' 1 / 29.034


' -----[ Variables ]-------------------------------------------------------

rawDist         VAR     Word                    ' raw measurement
inches          VAR     Word
cm              VAR     Word


' -----[ EEPROM Data ]-----------------------------------------------------


' -----[ Initialization ]--------------------------------------------------

Reset:
  DEBUG CLS,                                    ' setup report screen
        "Parallax Ping Sonar", CR,
        "-------------------", CR,
        CR,
        "Time (uS).....     ", CR,
        "Inches........     ", CR,
        "Centimeters...     "


' -----[ Program Code ]----------------------------------------------------

Main:
  DO
    GOSUB Get_Sonar                             ' get sensor value
    inches = rawDist ** RawToIn                 ' convert to inches
    cm = rawDist ** RawToCm                     ' convert to centimeters

    DEBUG CRSRXY, 15, 3,                        ' update report screen
          DEC rawDist, CLREOL
    DEBUG CRSRXY, 15, 4,
          DEC inches, CLREOL
    DEBUG CRSRXY, 15, 5,
          DEC cm, CLREOL

    PAUSE 100
  LOOP
  END


' -----[ Subroutines ]-----------------------------------------------------
```

```
' This subroutine triggers the Ping sonar sensor and measures
' the echo pulse.  The raw value from the sensor is converted to
' microseconds based on the BASIC Stamp module in use.  This value is
' divided by two to remove the return trip -- the result value is
' the distance from the sensor to the target in microseconds.

Get_Sonar:
  Ping = 0                                        ' make trigger 0-1-0
  PULSOUT Ping, Trigger                           ' activate sensor
  PULSIN  Ping, 1, rawDist                        ' measure echo pulse
  rawDist = rawDist */ Scale                      ' convert to uS
  rawDist = rawDist / 2                           ' remove return trip
  RETURN
```