**Column #76, August 2001 by Jon Williams:**

# Control From the Couch

*Yep ... I'm a real man, alright. I live in the great state of Texas, I drink milk right out of the carton, I leave the toilet seat up and, of course ... I have five remotes to run the electronics in my entertainment center.*

IR (infrared) remotes have become as commonplace as pagers and cellular phones – they're everywhere and there is no escape. I actually have a small TV/VCR unit that has many functions that WON'T work without the remote. I know some of you (youngsters) are thinking, "Yeah ... so what, dude?" Well, there are more than a few of us that remember having to cross the room to adjust the volume or change the channel. Yes, the television stone age.

Since I've got all these remotes and one more – as they say here in the South – "ain't no big thang..." I thought I'd play with decoding IR commands with a BASIC Stamp so that I could control more than the entertainment center from my couch. Now let me admit right up front that the code I'm presenting here is based on the work of one of my Parallax colleagues, Andy Lindsay. Andy is one of those incredibly enthusiastic guys who is like a bulldog when solving a problem and his enthusiasm is infectious – he's like the Pied Piper of hardcore Stamp programmers. Andy's done a lot of work with IR decoding and has created some really neat projects that use his techniques.

A few months ago, Andy showed me how easy it is to decode the Sony IR protocol with a standard Stamp 2. Easy, but consumes a fair chunk of variable space to do the decoding. What I thought I'd try to do is use the speed of the BS2sx and BS2p to do more detailed decoding while using fewer variables – a precious resource for the Stamp. Thankfully, it worked and I'm here to show you how. Our purpose, then, is to build a framework for IR remote control applications. What you control is up to you (I'll point to a couple examples on the web to give you ideas).

**Understanding The Sony IR Protocol (SIRCS)**

The Sony IR Control System protocol, SIRCS, is serial, but not like the serial signals we're accustomed to receiving with **SERIN**. The typical serial signal begins with a start bit then (usually) has eight data bits and one or two stop bits evenly spaced in the packet. The level of the bit determines its value.

The Sony protocol is pulse coded; the *width* of a bit determines its value. The start bit is 2.4 mS wide, a zero bit is 0.6 mS wide and a one bit is 1.2 mS wide. Every bit is followed by a rest period of 0.6 mS. There are 12 bits in the packet: the upper five for the device code, the lower seven for the unique command (Note: Internet resources indicate that there are also 15- and 20-bit versions of the protocol that are beginning to appear in high-end television and video equipment). When a key is held down, packets are repeated with a 20 to 30 mS break between them.

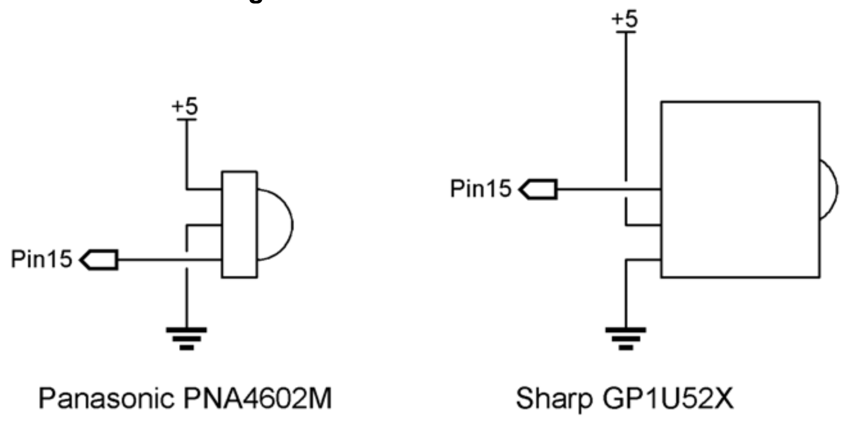
It should be clear by now that we can't use **SERIN** for this, so how are we going to read the Sony IR code?

**Decode ... Decode ...**

The idea is easy and so is the process. We're going to monitor the output of an IR detector and measure the width of output pulses. Lucky for us, the Stamp's **PULSIN** function is specifically designed for this purpose. Bit by bit, we'll build a packet. Once we find a start bit, we know that the next twelve bits are the meat of the packet and we can grab them. The BS2sx and BS2p are fast enough to decode the value of the last bit before the next one arrives (the BS2 isn't). This allows us to use just one word-sized variable to do pulse measurement (the BS2 requires a separate variable for each bit).

Hardware for IR detection is very simple: just connect an IR detector module to Vdd, Ground and an available Stamp pin. I tested these programs with a 40 kHz detector (Sharp GP1U52X) from Radio Shack (#276-137) and the 38 kHz detector (Panasonic PNA4602M) that is available from Parallax (#350-00014). The Sony specification is for

Figure 76.1: IR Detector Circuits



40 kHz modulation but I found that the 38 kHz detector worked just as well. Figure 76.1 shows the connections.

**The Code ... The Code ...**

Okay, then, let's make it work. The code in Program Listing 76.1 (IR_SCAN12.BSX) is a general-purpose Sony IR scanner. This program will monitor the IR detector output and display a code as it is received. The absence of a key is indicated by the constant value $FFF. This program has a repeat timer/counter so we can deal with a key that is being held down.

Take a look at the constants section first. You'll see the declarations StartWidth, Bit0Width and Bit1Width. These may look a little funky considering the specifications we just talked about, so let me explain. The **PULSIN** function on the BS2sx measures the width of a pulse in 0.8 uS units. This means that we have to multiply the **PULSIN** result by 0.8 to convert it to microseconds. That's what we're doing here – just the other way around: we divide microseconds by 0.8 to get our expected result value. 2400 uS (2.4 ms) divided by 0.8 is 3000. So why is the start bit width for the BS2sx set to 2700?

In my experiments (with supplemental file IR_ANALYZE.BSX), I've found that every remote I tested outputs bits wider than the specification, but I didn't want to risk missing a start bit on a remote that may be tighter. So what I did is scaled back the start bit width by 10% (3000 x 0.9 = 2700). This width is far wider than the "1" bit spec (1500) so there is no danger of false triggering. Note that the **PULSIN** period for the BS2p is 0.75 uS. This accounts for the slight difference in constant values.

There's another important constant value, BitTest, that is actually calculated from the width of a zero bit. As we get into the heart of the code, you'll see that what we're actually going to do is look for ones. If a bit isn't one, it must be zero. Our test width is 150% of a zero bit and yet, still shorter than a "1." Let's go look at the Scan_IR subroutine to see how this works.

The routine starts by assuming a key isn't pressed and setting the irCode variable to $FFF (constant value NoKey). Then we wait for a start bit by using **PULSIN**. The output of the IR detector is active low, hence the IsLow (value = 0) declaration in the **PULSIN** function. The next line will cause the routine to terminate if no bit arrives before **PULSIN** times out (52 mS for the BS2sx). This line probably looks a little funny; using **BRANCH** with only one address. It's the same as

```
    IF irStart = 0 THEN IR_Exit
```

but works a little faster. Speed is important in this routine.

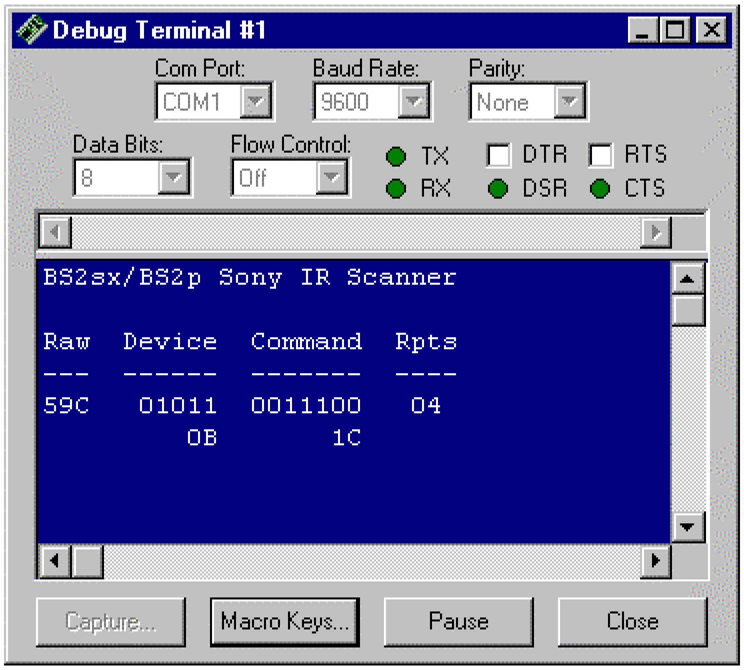
If we do receive a pulse it is checked by dividing it by the constant value, StartWidth. If the bit is a zero or one, dividing its width by the start bit width will return zero (remember that the Stamp uses integer math and division returns whole numbers) and the **BRANCH** command will force the program to look for another bit. When we do receive a start bit, the division will return one and we'll fall through the **BRANCH** and start collecting our data bits.

The next section works similarly: measure a bit and calculate its value. We're using the value BitTest as our divisor here and BitTest is 150% of a zero bit. When we do receive a one bit (which is wider than BitTest), the division will return one, otherwise it returns zero. When you look at the (redundant) code that receives the bits, you may be tempted to put it into a loop and save a bit of typing. Don't ... it won't work. I know because I tried every trick in the book and a few that aren't. The timing overhead required to deal with a loop and variable indexing is just too slow and prevents decoding the packet properly.

Now that we have a decoded packet, let's go back to the main section of code and see what's happening.

The first thing we do is check to see if the key we just received is the same as the last one. If not, we'll clear the repeat timer and show the key value. The **DEBUG** output section will show the key value as three HEX digits and separated into its 5-bit device code and 7-bit command code (the HEX values for these numbers are displayed on the next line).

**Figure 76.2: Debug Terminal Screenshot**



```
BS2sx/BS2p Sony IR Scanner

Raw   Device   Command   Rpts
---   ------   -------   ----
59C    01011   0011100    04
          0B        1C
```

When a key is held down, the program goes to Key_Timer before the display. If a valid key has been pressed, this section increments the key timer variable. What this does is let us control the key repeat rate. The actual rate is a function of the overall program loop timing multiplied by the KeyDelay constant value (which must always be greater than 0). In practice, we'd check to see that the keyRepeats value is zero before dealing with the key. A zero value means the key was just pressed or it has gone through the key repeat timing delay.

Figure 76.2 shows the output from the program when the VCR Fast Forward button is pressed. If you press a TV remote button, you'll get a device code of $01. The remote for my Sony video camera outputs a device code of $19 for VCR functions and $14 for camera functions

## Going Remote

If you don't have a Sony (or compatible) remote, it's not a problem. Just pop into your local discount store and get one of the generic multifunction models. It'll cost somewhere between $5 and $10. You need to get a remote that lets you manually program it (by entering a manufacturer's device code). In our techno-phobic world of "I can't program my VCR..." many remotes simply scan an internal table (while you're pointing it at the target device) until the device turns on or off. This won't work for us. I bought a Magnavox multi-function remote at Wal*Mart for $9. It let me set the TV and VCR buttons for Sony products.

Interestingly, the VCR buttons (Play, Rewind, Fast Forward, etc.) still work when the remote is in TV mode. The device code of $0B indicates the VCR device. The device code is useful for keys that are common to both, like the channel changer and numeric keys. We can take advantage of the unique device code in our own projects.

## Take A Number, Buddy

The first few IR control programs I wrote simply used the channel up and down buttons to change a program variable. Then I saw one of Andy's IR controlled BOE-Bots. Andy could tell the BOE-Bot – through the IR remote – how far to move. He entered the movement value using the remote's numeric keys. This was way too cool to ignore.

Program Listing 76.3 (IR_NUMBER.BSX) is my generic version of the numeric input, updated for the BS2sx and BS2p. Since this program only cares about numeric keys, we can ignore the device code in the packet and scale our input variables down to bytes. This saves a bit of variable space.

When the program runs it asks you to press digits (up to some maximum) and then [Enter]. This program forces a key release by making the repeat rate very large. When the key timer value is something other than zero, the key is not processed. This code takes advantage of us having been conditioned by the operation of other remotes. If you do hold the key, it will eventually repeat. Human nature will cause you to release it and press again to repeat the digit.

Okay, let's analyze the heart of the program by starting with a valid number key. Since the key is not [Vol-] and not [Enter], the program will make its way to this line:

**IrCode = irCode + 1 // 10**

The purpose of this line is to "fix" the code alignment of the numeric keys. The "1" key has a code value of zero while the "0" key has a code value of nine. Adding one and taking the modulus (remainder of division) of ten takes care of correcting the alignment. The value in irCode now matches the key that was pressed.

The key is displayed on screen and the user's value is updated. Since the user value is a decimal number, we shift the old digits left by multiplying by 10. Our new key is added after the shift to complete the update.

If we make an entry error, we can correct it by pressing the [Vol-] key. This key is used because it's typically a left-arrow key on the remote, just like the backspace key on a computer keyboard. When this key is pressed the entry digit is erased by moving the screen cursor left with a backspace (8), then printing a space to remove the old digit, then printing another backspace to return the cursor to the correct spot. We also have to update the usrValue variable. This is a simple matter of dividing by 10 to get rid of the "ones" digit.

You may have notice that I set the MaxDigits value to four. This code doesn't do any validity testing, so allowing a 5-digit value could result in errors. To see for yourself, change MaxDigits to five, then enter the number 99,999. The entry area will show "99999," but the result in usrValue will be 34,463. The reason for this is that the maximum value of a 16-bit (word) variable is 65,535 so the usrValue gets truncated.

If you're interested in numeric input while still maintaining 12-bit code for device identification, download the supplemental file IR_NUMBER12.BSX.

## It's Up To You Now

So what do you want to control? The sky is the limit. At the Embedded Systems Conference in April, Parallax had several demos that used IR control. I wrote the code for our neon sign (each letter was an individual neon tube and controller by a Stamp pin) and for a model train speed controller. You can find the code for these projects on the Parallax web site at this link:

www.parallaxinc.com/html_files/resources/esc2001.htm

Just keep in mind that code was written a few months ago and I've updated the BS2sx/BS2p IR input routine. If you're a BS2/BS2e user and are chomping at the bit (so to speak) to use an IR remote with your project, download the file "IR LED & 40 KHZ DETECTOR.PDF" from Parallax. This document was written by Andy and is full of great IR stuff for the BS2. You can find find it on the Parallax web site.

Happy Stamping – from across the room or otherwise.

```
' -----[ Title ]-----------------------------------------------------------
' Program Listing 76.1
' File...... IR_SCAN12.BSX
' Purpose... IR Remote Scanner / Reporter
' Author.... Jon Williams (based on work by Andy Lindsay)
' E-mail.... jwilliams@parallaxinc.com
' Started... 23 MAR 2001
' Updated... 06 JUL 2001

' { $STAMP BS2sx }


' -----[ Program Description ]---------------------------------------------
'
' This program monitors an IR detector module and decodes the 12-bit Sony
' IR protocol (SIRCS).  When a key is detected, it's 12-bit code is
' displayed on the DEBUG screen and separated into device and command
' codes.
'
' No key pressed is indicated by code $FFF.
'
' Change the KeyDelay value to change the auto-repeat response.  The
' larger this value, the longer the delay repeats of the same key.


' -----[ Revision History ]------------------------------------------------
'
' 23 MAR 2001 : Original program developed for IR testing with BS2p
' 04 JUL 2001 : Improved IR scan routine to 12 bits
' 06 JUL 2001 : Improved display to show device and command codes


' -----[ I/O Definitions ]-------------------------------------------------
'
IR_pin  CON     15


' -----[ Constants ]-------------------------------------------------------
'
IsLow          CON     0
IsHigh         CON     1

NoKey          CON     $FFF           ' no IR key
KeyDelay       CON     5              ' loops for "new" key ( >0 )

StartWidth     CON     2700           ' width of IR start bit (BS2sx)
Bit0Width      CON     750            ' width of IR zero bit (BS2sx)
Bit1Width      CON     1500           ' widht of IR one bit (BS2sx)

'StartWidth    CON     2880           ' width of IR start bit (BS2p)
'Bit0Width     CON     800            ' width of IR zero bit (BS2p)
```

```
'Bit1Width      CON     1600            ' width of IR one bit (BS2p)

BitTest         CON     Bit0Width * 3 / 2 ' test width -- look for 1's

LF              CON     10              ' linefeed character


' -----[ IR Codes ]------------------------------------------------------
'
' Generic Sony IR remote codes (not a complete list)
'
IR_1            CON     $080
IR_2            CON     $081
IR_3            CON     $082
IR_4            CON     $083
IR_5            CON     $084
IR_6            CON     $085
IR_7            CON     $086
IR_8            CON     $087
IR_9            CON     $088
IR_0            CON     $089
IR_Enter        CON     $08B

IR_ChUp         CON     $090
IR_ChDn         CON     $091
IR_VolUp        CON     $092
IR_VolDn        CON     $093
IR_Mute         CON     $094
IR_Power        CON     $095


' -----[ Variables ]-----------------------------------------------------
'
irCode          VAR     Word            ' returned code
lastCode        VAR     Word            ' last returned code
irStart         VAR     Word            ' width or IR start bit
irBit           VAR     irStart         ' width of IR bit
keyRpts         VAR     Byte            ' repeats of current key

device          VAR     Byte            ' upper 5 bits of irCode
command         VAR     Byte            ' lower 7 bits of irCode


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


' -----[ Initialization ]------------------------------------------------
'
Intialize:
  PAUSE 500
  DEBUG "BS2sx/BS2p Sony IR Scanner", CR, CR
  DEBUG "Raw  Device  Command  Rpts", CR
  DEBUG "---  ------  -------  ----", CR


' -----[ Main Code ]----------------------------------------------------
'
Main:
  GOSUB Scan_IR                               ' check for IR key
  IF (irCode = lastCode) THEN Key_Timer       ' key is being held
  keyRpts = 0                                 ' not held, reset timer
  GOTO Show_Key

Key_Timer:
  IF (irCode = NoKey) THEN Show_Key           ' no key, skip timer
  keyRpts = keyRpts + 1       // KeyDelay     ' update the repeats timer

Show_Key:
  lastCode = irCode                           ' save last key

  device = irCode >> 7                        ' extract device code
  command = irCode & $7F                      ' extract command

  DEBUG Home, LF, LF, LF, LF
  DEBUG HEX3 irCode,"   "
  DEBUG BIN5 device,"  ", BIN7 command, "   "
  DEBUG DEC2 keyRpts, CR
  DEBUG "         ", HEX2 device, "       ", HEX2 command

Loop_Pad:
  PAUSE 50                                    ' pad loop timing
  GOTO Main


' -----[ Subroutines ]--------------------------------------------------
-
'
' Receive and decode Sony IR command
'
Scan_IR:
  irCode = NoKey                              ' flag value

Wait_For_Start:                               ' wait for start bit
  PULSIN IR_pin,IsLow,irStart
  BRANCH irStart,[IR_Exit]                    ' exit if no key down
  BRANCH irStart/StartWidth,[Wait_For_Start]

  ' This code MUST stay inline
  ' -- will NOT work in a loop

  PULSIN IR_pin,IsLow,irBit                   ' decode 12 bits
  irCode.Bit0 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit1 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit2 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit3 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit4 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit5 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit6 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit7 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit8 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit9 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit10 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit11 = irBit/BitTest

IR_Exit:
  RETURN
```

```
' -----[ Title ]-----------------------------------------------------------
' Program Listing 76.2
' File...... IR_NUMBER.BSX
' Purpose... Input Number from Sony IR remote
' Author.... Jon Williams (based on work by Andy Lindsay)
' E-mail.... jwilliams@parallaxinc.com
' Started... 04 JUL 2001
' Updated... 06 JUL 2001

' { $STAMP BS2sx }


' -----[ Program Description ]---------------------------------------------
'
' This program accepts numeric input from a Sony IR remote. This program
' uses only 7 bits (of 12 in the Sony protocol) for the IR code.
'
' Digits are entered from remote keypad. The Volume-down [Vol-] key acts
' like a backspace key to correct mistakes. Pressing [Enter] accepts the
' value.


' -----[ Revision History ]------------------------------------------------
'
' 05 JUL 2001 : Version 1 tested and working
' 06 JUL 2001 : Added backspace editing


' -----[ I/O Definitions ]-------------------------------------------------
'
IR_pin          CON     15


' -----[ Constants ]-------------------------------------------------------
'
IsLow           CON     0
IsHigh          CON     1

NoKey           CON     $7F             ' no IR key
KeyDelay        CON     50              ' loops for "new" key ( >0 )

StartWidth      CON     2700            ' width of IR start bit (BS2sx)
Bit0Width       CON     750             ' width of IR zero bit (BS2sx)
Bit1Width       CON     1500            ' width of IR one bit (BS2sx)

'StartWidth     CON     2880            ' width of IR start bit (BS2p)
'Bit0Width      CON     800             ' width of IR zero bit (BS2p)
'Bit1Width      CON     1600            ' widht of IR one bit (BS2p)

BitTest         CON     Bit0Width * 3 / 2 ' test width -- look for 1's
```

```
BS              CON     8               ' backspace character
MaxDigits       CON     4               ' width of input field


' -----[ IR Codes ]------------------------------------------------------
'
' Generic Sony IR remote codes (7-bit; not a complete list)
'
IR_1            CON     $00
IR_2            CON     $01
IR_3            CON     $02
IR_4            CON     $03
IR_5            CON     $04
IR_6            CON     $05
IR_7            CON     $06
IR_8            CON     $07
IR_9            CON     $08
IR_0            CON     $09
IR_Enter        CON     $0B

IR_ChUp         CON     $10
IR_ChDn         CON     $11
IR_VolUp        CON     $12
IR_VolDn        CON     $13
IR_Mute         CON     $14
IR_Power        CON     $15


' -----[ Variables ]-----------------------------------------------------
'
irCode          VAR     Byte            ' returned code
lastCode        VAR     Byte            ' last returned code
irStart         VAR     Word            ' width or IR start bit
irBit           VAR     irStart         ' width of IR bit
keyRpts         VAR     Byte            ' repeats of current key

numDigits       VAR     Nib             ' digits entered
usrValue        VAR     Word            ' entered value


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


' -----[ Initialization ]------------------------------------------------
'
Initialize:
  numDigits = 0                         ' reset digits entered
  usrValue = 0                          ' clear old value

  PAUSE 500
  DEBUG CLS, "Press digits (up to ",DEC MaxDigits,"), then [Enter]: "


' -----[ Main Code ]-------------------------------------------------------
'
Main:
  GOSUB Scan_IR                               ' check for IR key
  IF (irCode = lastCode) THEN Key_Timer       ' key is being held
  keyRpts = 0                                 ' not held, reset timer
  GOTO Check_Key

Key_Timer:
  IF (irCode = NoKey) THEN Check_Key          ' no key, skip timer
  keyRpts = keyRpts + 1       // KeyDelay     ' update the repeats timer

Check_Key:
  lastCode = irCode                           ' save last key
  IF (irCode = NoKey) THEN Main               ' no key, go get one
  IF (keyRpts > 0) THEN Main                  ' in repeat delay

Check_BS:
  IF (irCode <> IR_VolDn) THEN Check_Digit
  IF (numDigits = 0) THEN Main                ' nothing to clear
  DEBUG BS," ",BS                             ' clear screen digit
  usrValue = usrValue / 10                    ' update user value
  numDigits = numDigits - 1                   ' update digit count
  GOTO Loop_Pad

Check_Digit:
  IF (irCode = IR_Enter) THEN Show_Value
  IF (numDigits = MaxDigits) THEN Main        ' no space for another
  IF (irCode > 9) THEN Main                   ' not a digit

  irCode = irCode + 1 // 10                   ' correct digit value
  DEBUG DEC1 irCode                           ' show digit on screen
  usrValue = usrValue * 10 + irCode           ' update user value
  numDigits = numDigits + 1                   ' update digit count

Loop_Pad:
  PAUSE 100                                   ' pad loop timing
  GOTO Main

Show_Value:
  IF (numDigits > 0) THEN Has_Value           ' check for actual entry
  DEBUG CR, CR, "No value entered."
  PAUSE 1500
  GOTO Initialize

Has_Value:
  DEBUG CR, CR, "Your value was ", DEC usrValue
  PAUSE 2500
  GOTO Initialize


' -----[ Subroutines ]----------------------------------------------------
'
' Receive and decode Sony IR command
' -- downsized to 7 bits
'
Scan_IR:
  irCode = NoKey                              ' flag value

Wait_For_Start:                               ' wait for start bit
  PULSIN IR_pin,IsLow,irStart
  BRANCH irStart,[IR_Exit]                    ' exit if no key down
  BRANCH irStart/StartWidth,[Wait_For_Start]

  ' This code MUST stay inline
  ' -- will NOT work in a loop

  PULSIN IR_pin,IsLow,irBit                   ' decode 7 bits (command)
  irCode.Bit0 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit1 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit2 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit3 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit4 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit5 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit6 = irBit/BitTest
  PULSIN IR_pin,IsLow,irBit
  irCode.Bit7 = 0

IR_Exit:
  RETURN
```