Stamp™ Weekend Application Kit

599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office/Tech Support: (916) 624-8333
Fax: (916) 624-8003

General E-mail: info@parallaxinc.com
Technical E-mail: stamptech@parallaxinc.com
Web Site: www.parallaxinc.com
Educational Resources: www.stampsinclass.com

Contents

Introduction

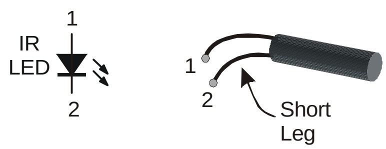
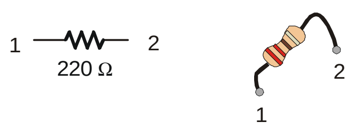
This document is for use with the infrared (IR) LED and 40 kHz detector added to on-line orders above $25 placed on www.parallaxinc.com from noon on April 26 through April 29, 2001. You can use the parts in this kit for object detection and line of sight communication. This document has six examples to get you started along with references to other Parallax documentation with more information (all available for free download).

Applications

The infrared (IR) LED in this kit is similar to what you would find in any handheld remote for your TV, cable box, VCR, etc. The IR detector inside each of these appliances is also very similar to the detector in your kit. These two components can be used together to perform a variety of functions. Just a few examples are:

- **Object detection** – determining when a person or object either reflects broadcasted IR or breaks an IR beam.
- **Proximity detection** – broadcasting IR that when reflected indicates the presence of an object/person within a certain proximity. You can also use some techniques to get a distance measurement with high enough resolution for some personal robotics applications.
- **Communication** - taking instructions from some handheld remotes and for communication between BASIC Stamps.

Parts

Quantity	Parallax Stock Code	Schematic Symbol, Part Drawing and Description
1	350-00017	 **Shrink wrapped IR LED**
1	350-00014	 **40 kHz IR Detector**
1	150-02210	 **220 W resistor**

Circuit

Figure 1.1 shows how to connect the parts. In this example, we'll use BASIC Stamp I/O pin P7 to send signals out the IR LED and P8 to read the 40 kHz IR detector's output.

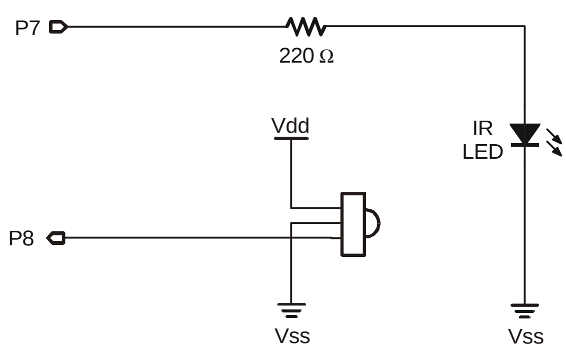


Figure 1.1: IR LED and detector schematic.

The IR detector is designed to send a low signal when it sees IR flashing on and off at about 40 kHz with a 50% duty cycle, meaning that the on/of times are equal.

Although the FREQOUT command only goes up to 32,768 Hz, if you leave off any filtering, you can use arguments above 32,768 to specify the frequency of a harmonic signal. In other words, when you use a command like FREQOUT 7,1,37500 without the RC filter shown in the *BASIC Stamp Manual*, a harmonic signal is broadcast at roughly 37500. The 40 kHz detectors do a good job picking up this signal. The unfiltered FREQOUT signal also causes the IR detector's output to rebound more slowly than when it sees 40 kHz at 50% duty. The result is that you can send the FREQOUT signal out 1 I/O pin, stop sending the signal, and still have enough time to check the detector's (slowly rebounding) output.

For more information on how the "FREQOUT trick" works, take a look at the beginning of Chapter 5 in the Parallax *Robotics! v1.4 Student Workbook*. It's available for free download from the www.stampsinclass.com → Downloads → Educational Curriculum page. This chapter also discusses the IR wavelengths these components use in case you want to try to find substitute parts locally.

Breaking the Beam and Testing the Circuit

The round bump on the face of the IR detector is a lens it uses to collect light. In the absence of IR flashing on and off at around 40 kHz, the IR detector sends a 5 V high signal. When IR flashing on and off at around 40 kHz enters this lens, the circuitry inside the IR detector sets its output to 0 V (active-low).

By pointing the IR LED's output at the IR detector's input (see Figure 1.2), we can test to make sure the system is working by writing a program that tells you if there is an obstacle, such as a piece of paper or your hand between the IR LED and the detector. If there is nothing to break the beam, the IR detector sends a low signal. If there is something breaking the beam, the IR detector does not see the modulated IR and sends a high signal.

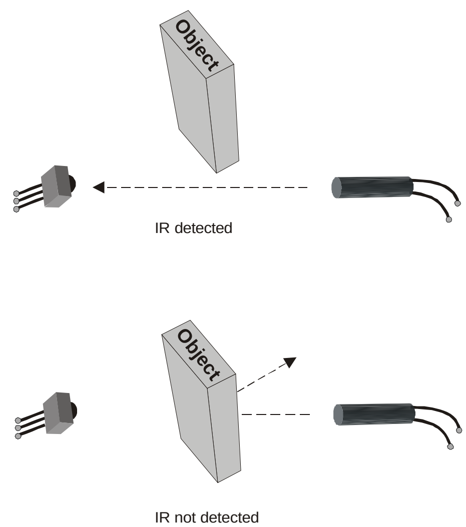


Figure 1.2: Test setup for breaking the beam.

Try this program. The debug terminal should tell you whether or not the beam is broken. If it tells you the beam is broken regardless of whether or not there is something blocking the IR, it means you have a problem with either your wiring or setup. Double check Figure 1.1 and 1.2, and try again.

```
' Program Listing 1.1 - Testing the IR Beam.bs2

IR_detect  var  bit
low 7

loop:

  pause 50
  freqout 7, 1, 38500
  IR_detect = in8
  if IR_detect = 0 then unbroken

    ' Make sure to add six spaces to the debug statement
    ' below.  That way both debug statements will have the
    ' same number of spaces for a better display.

    debug home, "Beam is broken; object detected.      "
    goto loop

  unbroken:
    debug home, "Beam is unbroken; object not detected."
    goto loop
```

This code uses the FREQOUT command to broadcast a harmonic at about 37.5 kHz, which turns out to be the best harmonic frequency for detection using the "FREQOUT trick".

Proximity Detection

You can also point the IR-LED away from the detector (see Figure 1.3). When an object is close enough (and not dark black, which absorbs IR quite well), it reflects the IR signal back to the detector.

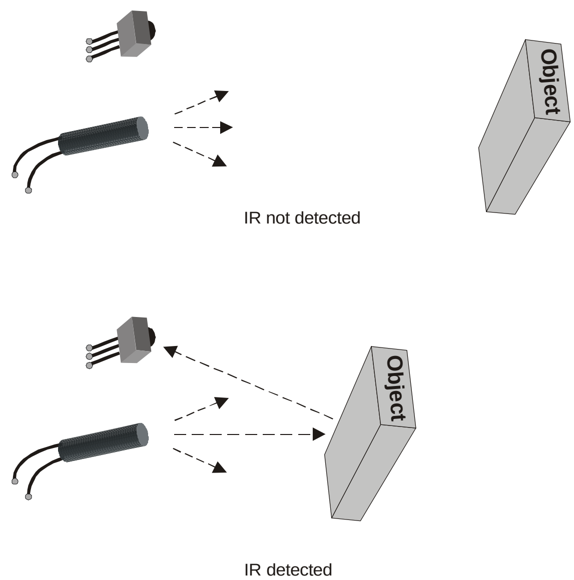


Figure 1.3: Test setup for proximity detection

Program listing 1.2 is almost the same as Program Listing 1.1. The only difference is that Program Listing 1.2 considers the presence of IR as indicating the presence of an object, while Program Listing 1.1 sees it in reverse.

```
' Program Listing 1.2 – Proximity Detection.bs2

IR_detect  var  bit
low 7

loop:

  pause 50
  freqout 7, 1, 38500
  IR_detect = in8
```

```
if IR_detect = 0 then not_detected

  debug home, "Output is high, no object detected."
  goto loop

not_detected:

  ' This time, add a space to the end of this debug
  ' command.  This will keep both strings the same
  ' length and make the display better.

  debug home, "Output is low, object is detected. "
  goto loop
```

Distance Detection

Program listing 1.3 uses a frequency argument of 37500 for maximum sensitivity. Then it progressively detunes the detector by using frequencies that make the detector less sensitive using these frequency arguments:

```
37500,38250,39500,40500,41500
```

Figure 1.4 shows how this detuning can be used to give you a rough idea of an object's distance from the detector.

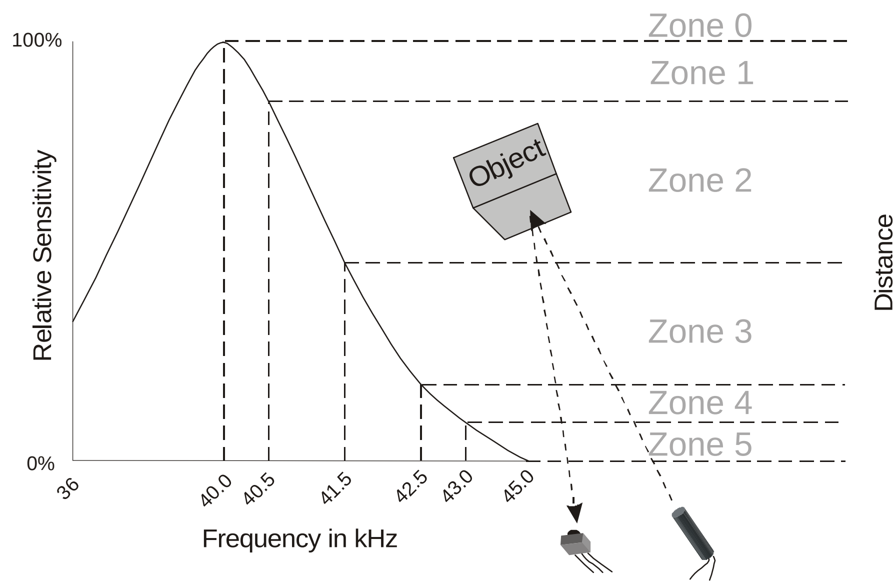


Figure 4: Test setup for distance detection.

The key idea here is that as the object gets closer, the reflected IR is brighter. In other words, the signal is stronger. To cause the IR detector's 40 kHz filter not to see the object, the frequency that's broadcast has to be further from the filter's center frequency. It's pretty easy to write a PBASIC program that tests at each frequency and tracks when the object was no longer detected.

Run Program Listing 1.3 and try placing your hand, or a piece of paper at various distances from the detector. You should be able to detect at least zones 0 through 4; zone 5 may or may not show up.

```
' Program Listing 1.3 – Distance Detection.bs2

  counter    var   nib
  IR_outputs var   byte
  IR_freq    var   word

  output 7

  main:

    IR_outputs = 0
```

```
  ' Load sensor outputs into l_IR_outputs and r_IR_outputs
  ' using a for...next loop,and a lookup table, and bit
  ' addressing.

  for counter = 0 to 4

    lookup counter,[37500,38250,39500,40500,41500], IR_freq

    freqout 7,1, IR_freq
    IR_outputs.lowbit(counter) = ~in8

  next

  ' Display l_IR_outputs and r_IR_outputs in binary and ncd
  ' format.

  debug home, "Readings from IR detector", cr
  debug "Binary IR_outputs: ", bin5 IR_outputs, cr
  debug "Object is in zone: ",dec5 ncd(IR_outputs)

goto main
```

Reading Signals from a Universal IR Remote

You can configure most universal remotes to use SONY® Corporation's IR communication protocol to send messages to the BASIC Stamp. To try this you'll need to dig up the documentation for one of your universal remotes or buy one. They are available for around $10 at most electronics retail outlets (such as Radio Shack®) and also at general purpose retailers (such as K-Mart®).

Once you've got the documentation for your universal remote, check the instructions for how to configure it for a particular TV. There's usually a programming button that you have to press and hold until an LED on the remote blinks a couple of times. Then, you enter the a code from a list of manufacturers (also in the universal remote's documentation), and the LED will blink twice again indicating that the code was accepted. Follow the instructions in the universal remote's documentation to make the universal remote send signals to a SONY® television.

Try Program Listing 1.4 to see if the BASIC Stamp is receiving the expected messages. While running Program Listing 1.4, Press and hold the 5 key on the remote. Check and make sure your display is similar to Figure 1.5, which shows what the debug terminal should display. The pulse duration values might slightly different from yours, but the decimal and binary numbers at the bottom of the display should match exactly.

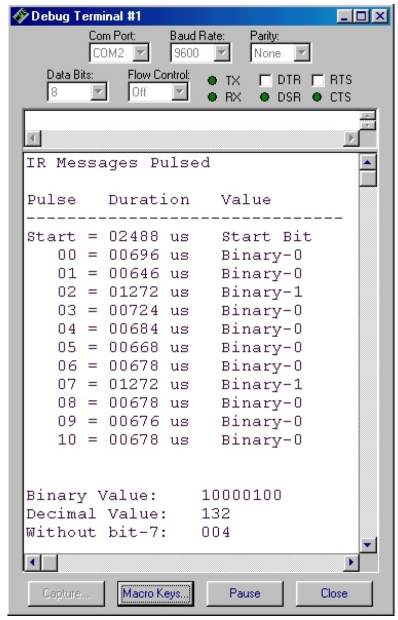


Figure 5: Debug terminal when you press and hold the 5 key while running Program Listing 1.4.

```
'-------------------------------------------------------------
' Program Listing 1.4 – Display IR Remote.bs2
'-------------------------------------------------------------
declarations:

IR_det_pin              con     8
pause_time              con     20
active_low              con     0

IR_detect               var     in8

IR_pulse                var     word(12)
counter                 var     nib
type                    var     nib
pulse_delay_time        con     2


debounce_time           con     20
IR_message              var     byte
active_high             con     1
```

```
'------------------------------------------------------------
initialize:                                    ' Boot Routine

  debug cls

'------------------------------------------------------------
main:                                          ' Main Routine

   if IR_detect = 1 then main:
   gosub display_heading
   gosub find_and_display_start_pulse
   gosub process_IR_pulses
   gosub display_IR_pulse_values
   gosub convert_to_binary_number_display

goto main

'------------------------------------------------------------
display_heading:                               ' Subroutine

  debug home
  debug "IR Messages Pulsed ", cr, cr
  debug "Pulse   Duration   Value", cr
  debug "-------------------------------", cr

return

'------------------------------------------------------------
find_and_display_start_pulse:                     ' Subroutine

  for counter = 0 to 15
    pulsin IR_det_pin,active_low,IR_pulse(0)
    if IR_pulse(0) > 900 then display_start_bit
  next

  goto exit_find_and_display_start

  display_start_bit:
    debug "Start"
    debug " = ", dec5 IR_pulse(0) * 2, " us "
    debug  "  Start Bit", cr

  exit_find_and_display_start:

return

'------------------------------------------------------------
process_IR_pulses:                             ' Subroutine

  check_for_stop_bit:
     pulsin IR_det_pin,active_high,IR_pulse(0)
     if IR_pulse(0) > 1400 and IR_pulse(0) <> 0 then continue
     goto check_for_stop_bit
```

```
     continue:

     pulsin IR_det_pin,active_low,IR_pulse(0)
     pulsin IR_det_pin,active_low,IR_pulse(1)
     pulsin IR_det_pin,active_low,IR_pulse(2)
     pulsin IR_det_pin,active_low,IR_pulse(3)
     pulsin IR_det_pin,active_low,IR_pulse(4)
     pulsin IR_det_pin,active_low,IR_pulse(5)
     pulsin IR_det_pin,active_low,IR_pulse(6)
     pulsin IR_det_pin,active_low,IR_pulse(7)
     pulsin IR_det_pin,active_low,IR_pulse(8)
     pulsin IR_det_pin,active_low,IR_pulse(9)
     pulsin IR_det_pin,active_low,IR_pulse(10)
     pulsin IR_det_pin,active_low,IR_pulse(11)

return

'---------------------------------------------------------------
display_IR_pulse_values:                     ' Subroutine

  for  counter = 0 to 10

    debug "   ", dec2 counter
    debug " = ", dec5 IR_pulse(counter) * 2, " us "

    branch IR_pulse(counter)>>9,[zero,one]

      zero:  debug  "  Binary-0", cr: goto loop_again
      one:   debug  "  Binary-1", cr: goto loop_again

    loop_again:

  next

return

'---------------------------------------------------------------
convert_to_binary_number_display:            ' Subroutine

  for  counter = 0 to 10

    lookdown IR_pulse(counter), < [400,800],
IR_message.lowbit(counter)

  next

  debug cr,cr,"Binary Value:    ", bin8 IR_message, cr
  debug "Decimal Value:   ", dec3 IR_message, cr
  debug "Without bit-7: "
  debug "  ",dec3 IR_message & %01111111,cr

return
```

The IR detector's output when pressing the 5-key looks about like this on an oscilloscope.

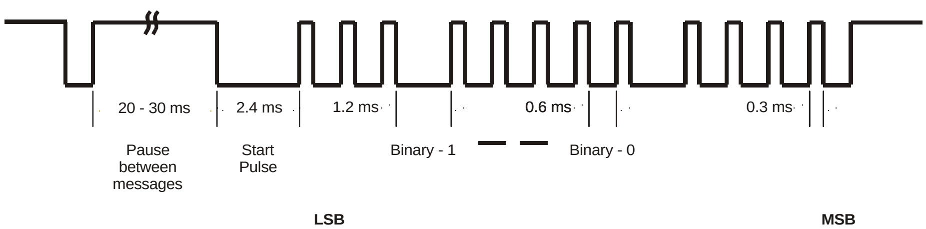


Figure 6: IR Output.

Regardless of which key is pressed, the message repeats itself every 20 to 30 ms. The start pulse lasts 2.4 ms, which is followed by binary pulses. A binary-1 lasts 1.2 ms, a binary-0 lasts 0.6 ms, and the time between pulses is usually somewhere between 0.25 and 0.8 ms depending on the brand and vintage of the remote.

The BASIC Stamp 2p is fast enough to decide what value it received between pulses, so the programming is considerably more concise (similar to Program Listing 1.7). To keep it general, this program is set up for the BASIC Stamp 2.

To determine if an IR signal is being transmitted, an if…then statement just before the main: routine checks over and over again to see if the IR detector's output is active-low. If yes, then it executes the rest of the program, else, it just keeps checking. When an IR broadcast is detected, Program Listing 1.4 next searches for the start pulse and displays it. Then, the data bits are captured and displayed. The process_IR_pulses subroutine looks for the lengthy pause between data pulses (a positive pulse). If you think about that high time between the last data pulse in the previous message and the start pulse in the current message as a long positive pulse, you'll see the reasoning for the PULSIN and IF…THEN reasoning at work for finding the first data pulse. It will either return a number larger than 2.8 ms or a data overrun value of 0. As soon as this happens, you're in the middle of the start pulse. So, the next negative pulse will be signal the start of the first data pulse.

Since this program was designed to work with the BASIC Stamp 2 or better, each PULSIN command is consecutive to prevent any data bits from being lost due to processing time.

Entering Multiple Numbers

You can use Program Listing 1.5 to display sequences of numbers entered into the universal remote. This program is only using the first seven bits instead of all eleven bits the remote sends. This saves memory, and you can still use most of the commonly used keys on the remote. The program waits for you to press the “Enter” key on the remote before displaying the value. You can use the previous example program (Program Listing 1.4) to figure out what the code is for each key and expand this program’s functionality.

```
'--------------------------------------------------------------

' Program Listing 1.5 – Entering Multiple Numbers with IR.bs2

'--------------------------------------------------------------
declarations:

IR_det_pin       con         8
IR_detect        var         in8

pause_time       con         20
active_low       con         0
active_high      con         1
debounce_time    con         200

enter            con         11
power            con         21

IR_pulse         var         word(7)
counter          var         nib
type             var         nib
entered_value    var         word
pulse_delay_time  con        2
IR_message       var         byte


'--------------------------------------------------------------
initialize:                                      ' Boot Routine

  IR_message = 0
  entered_value = 0

  debug cls
  debug "Enter a number between 0 and 65535.", cr
  debug "Press the enter key on your remote", cr
```

```
  debug "to see the number you entered.", cr, cr
  debug "Press Power (TV on/off) to start over", cr, cr

'---------------------------------------------------------------
main:                                         ' Main Routine

   if IR_detect = 1 then main:
   get_IR_data:
     gosub process_IR_pulses
     if IR_message = power then initialize
     if IR_message > enter then user_pressed_wrong_key
     if IR_message = enter then user_pressed_enter

     ' Correct for the fact that the 1-key is 0, the 2-key
     ' is 1,..., the 0 key is 9.

     lookup IR_message,[1,2,3,4,5,6,7,8,9,0],IR_message
     entered_value = entered_value * 10 + IR_message
     pause debounce_time
     debug "You pressed ", dec1 IR_message, cr
     goto main

   user_pressed_wrong_key:
     debug "Press one of the numeric keys", cr
     debug "or press enter. ", cr
     goto main

   user_pressed_enter:
    debug "The number entered was: ", dec5 entered_value, cr, cr
     entered_value = 0

goto main

'---------------------------------------------------------------
process_IR_pulses:                            ' Subroutine

  check_for_stop_bit:
    pulsin IR_det_pin,active_high,IR_pulse(0)
    if IR_pulse(0) > 1400 and IR_pulse(0) <> 0 then continue
    goto check_for_stop_bit

    continue:

      pulsin IR_det_pin,active_low,IR_pulse(0)
      pulsin IR_det_pin,active_low,IR_pulse(1)
      pulsin IR_det_pin,active_low,IR_pulse(2)
      pulsin IR_det_pin,active_low,IR_pulse(3)
```

```
      pulsin IR_det_pin,active_low,IR_pulse(4)
      pulsin IR_det_pin,active_low,IR_pulse(5)
      pulsin IR_det_pin,active_low,IR_pulse(6)

      for  counter = 0 to 6
lookdown IR_pulse(counter), < [400,800], IR_message.lowbit(counter)
      next

return
```

Stamp to Stamp Communication

For kicks, let's create our own Stamp-2-Stamp protocol. Here are some specifications I just made up:

- A start bit is 1 ms
- A binary-0 is 2 ms
- A binary-1 is 3 ms
- A stop bit is 4 ms
- The delay between pulses is 2 ms + any loop processing overhead
- 8 data bits are transmitted

This activity is designed for use with two BASIC Stamp 2 modules. Connect your IR LED and IR detector circuits to separate BASIC Stamp 2 modules as shown in Figure 1.7. Download Program Listing 1.6 into the transmitting BS2, and disconnected it from the programming cable. Next, connect the receiving BS2 to the programming cable and download Program Listing 1.7 into it. Leave it connected to the programming cable for debugging. Press and release the reset button on the transmitting BS2 while pointing its IR LED at the receiving BS2's IR detector. The receiving BS2 will display the number the transmitting BS2 sent (25 in this case).

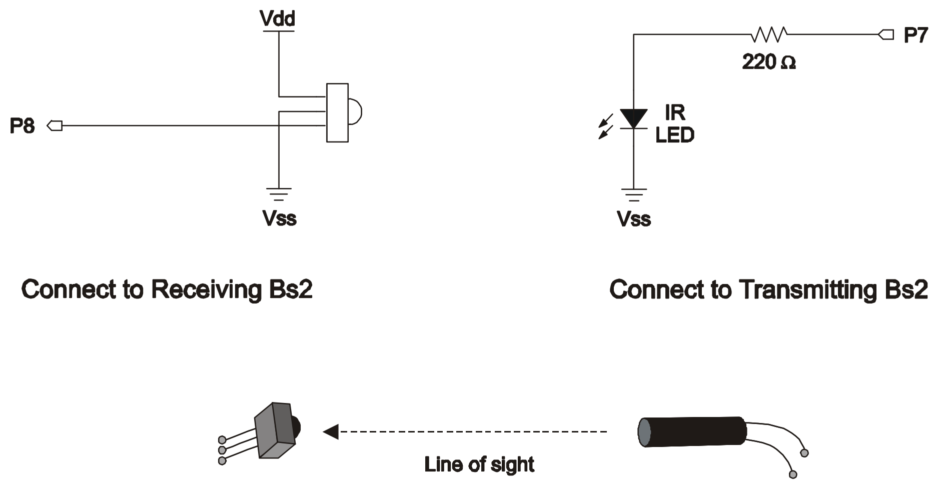


Figure 7: IR Output.

```
'-----------------------------------------------------------------

File Name:  Program Listing 1.6 - Stamp-2-Stamp IR_tx.bs2

'-----------------------------------------------------------------

IR_LED_pin       con    7
IR_freq       con    37500

start_bit     con    1
stop_bit      con    4
bin_0         con    2
bin_1         con    3
between_pulses   con    2

counter       var       nib
IR_message       var       byte
duration      var       nib

IR_message = 25

freqout IR_LED_pin,start_bit,IR_freq
pause between_pulses

for counter = 0 to 7

  duration = 2 + IR_message.lowbit(counter)
  freqout IR_LED_pin,duration,IR_freq
  pause between_pulses
```

```
next
freqout IR_LED_pin,stop_bit,IR_freq

' ****** Reset Stamp to run again *******
stop
```

```
'--------------------------------------------------------------
File Name:  Program Listing 7 - Stamp-2-Stamp IR_rx.bs2
'--------------------------------------------------------------

IR_detect_pin con    8
IR_signal     var    in8

start_bit     con    500
bin_0         con    1100
bin_1         con    1600
stop_bit      con    1900

active_low       con    0

counter       var       nib
IR_pulse      var       word
duration      var       nib
IR_message       var    byte

loop:

  if IR_signal = 1 then loop

  process_message:
    pulsin IR_detect_pin,active_low,IR_pulse
    if IR_pulse > stop_bit then display_message
    lookdown IR_pulse, < [bin_0,bin_1],
IR_message.lowbit(counter)
    counter = counter + 1
    goto process_message

  display_message:
    debug "Message received,", cr
    debug "it's ", dec3 IR_message, cr
    IR_message = 0
    counter = 0

goto loop
```

Extending the IR Range and Capabilities

You can use a transistor and a 100 Ω to better than double the transmitting power of the IR LED in your kit. A 555 timer can also be used to enable BASIC Stamp to use the SERIN and SEROUT commands for IR data exchange. The circuit is featured in BASIC Stamp 1 Application Note 11 – Infrared Communication. The BASIC Stamp 1 application notes are available for free download from the www.parallaxinc.com → Downloads page.