Appendix C: Transmitting IR Remote Signals with the BASIC Stamp 2

The activities in this text focused on programming the BASIC Stamp to receive and decode IR remote signals. If you are also interested in encoding and transmitting the same signals an IR remote would send, this appendix/activity provides a circuit and example programs that can actually be used in place of the universal remote to send SONY TV control commands to the Boe-Bot.

Other Protocols

Several IR remote protocols can be transmitted by the BASIC Stamp 2. Typically, if the 38 kHz transmit times and the delays between transmit times are greater than or equal to 0.6 ms, it's possible to write a PBASIC program to make the BASIC Stamp 2 do the job. The challenge is typically writing code to control the transmit and delay times. This code has to be executed during the delays, and it contributes to the delays as well. If the code takes longer to do the calculations than the protocol allows for the delays, it may be necessary to use a different microcontroller (see Better Tools and Other Protocols, below).

There is usually more than one way to write code that calculates the 38 kHz transmit times, and it can make a big difference in the delays between transmits. PULSOUT commands to unconnected pins can further tune the delay times when they are critical to the protocol. This kind of tuning involves monitoring the signals the BASIC Stamp sends with an oscilloscope. (See Understanding Signals, which has a section on analyzing the SONY protocol with the Parallax USB Oscilloscope. Both Understanding Signals and the Parallax USB Oscilloscope are available at www.parallax.com)

Better Tools for Other Protocols

Although several different protocols can be mimicked by the BASIC Stamp 2 with the help of the 555 timer circuit introduced in this activity, it really isn't the best tool for the job. It's kind of like using a hammer and a nail to make a hole. It would be better to just use a drill. Microcontrollers like the SX and Propeller are much better suited to precisely timing signals, and they do not need an external 555 timer circuit. They are also reasonable next steps after you have become comfortable with BASIC Stamp programs and circuits. Programs for the Propeller and SX microcontrollers typically monitor an internal timer and turn the 38 kHz signals on/off at the required time intervals.

For more information about the SX and Propeller microcontrollers, go to www.parallax.com.

Pulse Controlled 38 kHz Transmitter Parts

Next to the circuit from the previous appendix, the parts for the circuit in this appendix are arguably the second least expensive form of wireless communication between BASIC Stamps. Some of the parts listed here are not included in the Boe-Bot or IR Remote for

the Boe-Bot kits. However, the missing parts are inexpensive, easy to obtain, and they can also be ordered from Parallax. The Parallax part numbers for the extra parts are included in the parts list below.

(1) Capacitor - 0.01 µF
(1) Resistor - 470 Ω (Yellow, Violet, Brown)
(2) Resistors - 220 Ω (Red, Red, Brown)
(1) IR LED
(6) Jumper wires

(1) Potentiometer - 10 kΩ
    Parallax Part#: 152-01031
(1) NE555N timer IC
    Parallax Part#: 604-00009

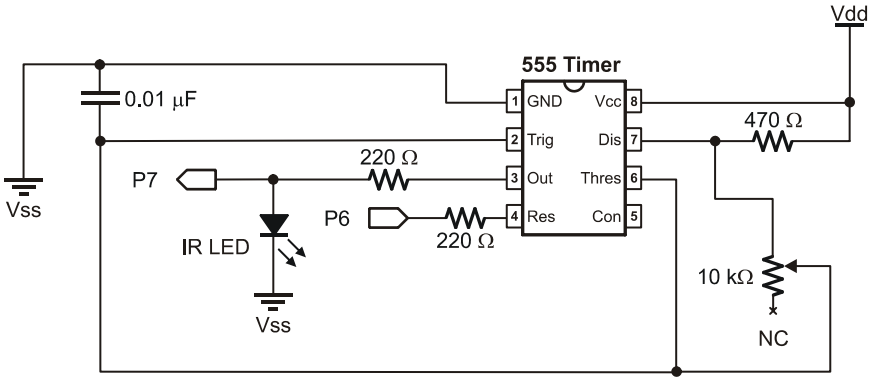
Pulse Controlled 38 kHz Transmitter Circuit

Figure C-1 shows the IR transmit circuit. The 10 kΩ potentiometer in this circuit has to be adjusted so that it transmits 38 kHz. The BASIC Stamp can be programmed to monitor and report the frequency transmitted by the 555 timer's Out pin with P7. When the correct transmit frequency is established, the BASIC Stamp can then be programmed to control the durations the 555 timer transmits 38 kHz by sending `PULSOUT` signals to the 555 timer's Res pin with P6.

- √ If you are using the 10 kΩ potentiometer with the Parallax Part#: 152-01031, pinch each of its legs with a needle-nose pliers to remove the kinks and straighten them before inserting the part into the breadboard.
- √ Build the circuit shown in Figure C-1 on a second board, which you will use to beam IR remote messages to your board with a receiver circuit (or a Boe-Bot with IR object detection circuits).

To learn more about designing 555 circuits for transmitting various frequencies and duty cycles, download the *Basic Analog and Digital* PDF from www.parallax.com, and read the first five pages of Chapter #6.

Figure C-1



Tuning and Testing the 38 kHz Transmit Circuit

Test555Frquency.bs2 sets P6 high to enable the 555 timer, at which point its Out pin will start sending high/low signals. P7 is connected to the 555 timer's Out pin, and the command **`COUNT 7, 100, cycles`** stores the number of times it sends the high/low signal in 100 ms. The command **`cycles = cycles * 10`** gives adjusts the value to the number of cycles per second, which is then displayed with a **`DEBUG`** command. The potentiometer can then be adjusted for a target 555 timer output frequency of 38 kHz.

- √ Enter, save, and run Test555Frequency.bs2
- √ Adjust the potentiometer knob with a screwdriver until the Debug Terminal reports a frequency in the 37 to 39 kHz range (your target frequency is 38.0 kHz).

Example Program: Test555Frquency.bs2

```
' IR Remote for the Boe-Bot - Test555Frequency.bs2
' Displays 555 timer frequency for potentiometer adjustment and tuning.

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

cycles VAR Word

DO

  HIGH 6
  COUNT 7, 100, cycles
```

```
  LOW 6
  cycles = cycles * 10
  DEBUG HOME, "Frequency = ", DEC5 cycles, " Hz"

LOOP
```

Transmitting the SONY IR Remote Signal

Since high/low signals to the 555 timer's Res pin turn the 555 timer's signal on/off, you can now use the **PULSOUT** command to send 38 kHz signals for precise amounts of time. For example, to send a 2.4 ms start pulse, you can use the command **PULSOUT 6, 1, 1200**. To send a 1.2 ms binary-1 pulse, you can use the command **PULSOUT 6, 1, 600**, and to send a 0.6 ms binary-0 pulse, the command **PULSOUT 6, 1, 300** does the job.

TransmitIrRemoteButtons.bs2 has a subroutine that automates transmitting IR remote button information. Simply set the remoteCode variable to the button or value you want to transmit, then call the program's **Send_Ir_Remote_Code** subroutine like this.

```
remoteCode = ChUp
GOSUB Send_Ir_Remote_Code
```

In practice, it's best to send, between 5 to 10 copies of the same code, since that's what happens when you press and release a button on the remote. So, your code might actually look like this.

```
FOR counter = 1 to 5
  remoteCode = ChUp
  GOSUB Send_Ir_Remote_Code
NEXT
```

In Chapter 1, you calculated the number of times per second an IR remote repeats its message. Depending on the remote and the code that was sent, it was probably in the neighborhood of 40 to 50 ms per message. This can be considered the period (T) of the message cycle. The frequency (f) of messages is f = 1/T, and that's the number of messages sent per second the remote sends if you hold your finger on one of the remote's buttons for one second.

$$f_{messages} = 1 \div 0.045 \text{ s} = 22.2.. \text{ Hz}$$

TransmitIrRemoteButtons.bs2 emulates the signal coming from a remote for the sequence of key presses listed below.

- CH+ for 2 seconds
- VOL+ for 1 second
- VOL- for 1 second
- CH- for 2 seconds
- Power for 0.45 seconds
- 3 for 0.45 seconds

Example Program: TransmitRemoteButtons.bs2

The goal with this program is to send commands to a Boe-Bot that is running IrMultiBot.bs2 from Chapter 3, Activity #2. Then, when you run TransmitRemoteButtons.bs2 while pointing the IR LED at the Boe-Bot's IR receiver while maintaining line of sight and close proximity, it will make the Boe-Bot do the following.

- Forward for 2 seconds
- Rotate right for 1 second
- Rotate left for 1 second
- Backward for 2 seconds
- Power for 1/2 a second followed by 3 for 1/2 a second switches the Boe-Bot to mode-3, object following.

If you have just a BASIC Stamp board with the receiver circuit (but not a Boe-Bot), download IrRemoteButtons.bs2 to your receiver board instead of IrMultiBot.bs2. Then, use the Debug Terminal to display the messages the receiver board receives.

√ Connect your board (or a second Boe-Bot) with the 555 timer circuit to the programming cable.
√ Enter, save, and run TransmitIrRemoteButtons.bs2.
√ Disconnect your transmit board from the programming cable.
√ Connect your receiver board to the serial cable.

If your receiver board is on a Boe-Bot chassis:

√ Open IrMultiBot.bs2 with the BASIC Stamp Editor, download it to your Boe-Bot and disconnect the serial cable.
√ Make sure the 3-position switch on the Boe-Bot that will receive messages is set to 2.

- √ Press/release the reset button on the Boe-Bot that will receive messages.
- √ Establish line of sight and close proximity between the top of the transmitting IR LED and the face of the IR receiver.
- √ Press and release the Reset button on your transmitter board.
- √ The Boe-Bot should perform the maneuvers and mode changes discussed earlier. As it turns, you will probably need to adjust the position of your transmitter board to maintain line of sight.

If your receiver board is not on a Boe-Bot chassis:

- √ Open IrRemoteButtons.bs2 with the BASIC Stamp Editor, download it to your receiver board, and leave the serial cable connected.
- √ Establish line of sight and close proximity between the top of the transmitting IR LED and the face of the IR receiver.
- √ Press and release the Reset button on your transmitter board.
- √ The Debug Terminal should display the various codes for the amounts of time listed earlier, except for the last code, 3, which will persist until you press/release the transmitter board's reset button again.

```
' -----[ Title ]-------------------------------------------------------------
' IR Remote for the Boe-Bot - TransmitIrRemoteButtons.bs2
' Transmit SONY TV protocol IR remote codes to a Boe-Bot using a second
' Board of Education and BASIC Stamp with a 555 timer circuit to supply
' the 38 kHz carrier signal to an IR LED.  The BASIC Stamp's P6 I/O pin
' is connected to the 555 timer's Res pin, which shuts off the signal
' with a low signal and turns it back on with a high signal.

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

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

' Button press and number of times each message should be re-sent.

Buttons DATA ChUp, VolUp, VolDn, ChDn, Power,   3, 255
Reps    DATA   44,    22,    22,   44,    10,  10, 0

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

tPin            PIN     6                     ' Transmit pin
fPin            PIN     7                     ' Frequency sense pin

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

' SONY TV IR remote constants for non-keypad buttons
```

```
Enter           CON     11                      ' Enter
ChUp            CON     16                      ' Channel +
ChDn            CON     17                      ' Channel -
VolUp           CON     18                      ' Volume +
VolDn           CON     19                      ' Volume -
Power           CON     21                      ' Power on/off

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

' SONY TV IR remote variables

remoteCode      VAR     Word                    ' Stores remote code
cycles          VAR     Word                    ' Stores 555 timer frequency

' EEPROM and message repetition counting variables

index           VAR     Byte                    ' EEPROM index
messageCnt      VAR     Byte                    ' Number of IR message reps
counter         VAR     Byte                    ' General purpose counter

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

  HIGH 6                                        ' Enable 555 timer
  COUNT 7, 1000, cycles                         ' Measure frequency
  LOW 6                                         ' Disable 555 timer
  DEBUG HOME, "Frequency = ", DEC5 cycles,     ' Display frequency
        " Hz", CR,   "Transmitting..."

' -----[ Main Routine ]-----------------------------------------------------

DO                                              ' Main loop
  READ Buttons + index, remoteCode              ' EEPROM button -> remoteCode
  READ Reps + index, messageCnt                 ' EEPROM reps -> messageCnt
  IF remoteCode = 255 THEN EXIT                 ' remoteCode = 255? Exit loop
  FOR counter = 1 TO messageCnt                 ' Repeat messageCnt times
    GOSUB Send_Ir_Remote_Code                   ' Send current remoteCode
  NEXT
  index = index + 1                             ' Increment EEPROM index
LOOP                                            ' Repeat main loop

DEBUG CR, "Done!"                               ' Desplay "Done!" message

END                                             ' BASIC Stamp -> Sleep mode

' -----[ Subroutine - Send_Ir_Remote_Code ]--------------------------------

' Sends pulses to 555 timer that causes the IR LED connected to the Out
' pin to send 38 kHz signals of about the same durations as an IR
' remote set to send SONY TV signals.
```

```
Send_Ir_Remote_Code:

  ' Change button values to remote codes.

  IF (remoteCode = 0) THEN remoteCode = 10
  IF (remoteCode <= 10) THEN remoteCode = remoteCode - 1



  remoteCode.BIT7 = 1                         ' Set bit-7 = 1 for TV

  LOW tPin                                    ' 555 signal off
  PAUSE 23                                    ' Rest between messages
  PULSOUT tPin, 1200                          ' Start pulse

  PULSOUT tPin, remoteCode.BIT0 * 300 + 300   ' Bit-0 pulse
  PULSOUT tPin, remoteCode.BIT1 * 300 + 300   ' Bit-1 pulse
  PULSOUT tPin, remoteCode.BIT2 * 300 + 300   ' Bit-2 pulse
  PULSOUT tPin, remoteCode.BIT3 * 300 + 300   ' Etc...
  PULSOUT tPin, remoteCode.BIT4 * 300 + 300
  PULSOUT tPin, remoteCode.BIT5 * 300 + 300
  PULSOUT tPin, remoteCode.BIT6 * 300 + 300
  PULSOUT tPin, remoteCode.BIT7 * 300 + 300
  PULSOUT tPin, remoteCode.BIT8 * 300 + 300
  PULSOUT tPin, remoteCode.BIT9 * 300 + 300
  PULSOUT tPin, remoteCode.BIT10 * 300 + 300
  PULSOUT tPIn, remoteCode.BIT11 * 300 + 300

  RETURN
```

How TransmitIrRemoteButtons.bs2 Works

The **Send_Ir_Remote_Code** subroutine in TransmitIrRemoteButtons.bs2 is shown below. It sets tPin (P6) low to turn off the 555 timer's signal. The command **remoteCode.BIT7 = 1** sets bit-7 in the remote code variable to 1, which is what it always is for SONY TV messages. Then, it pauses for 23 ms. This, plus the processing time for the **READ** commands causes a typical message to repeat every 45 ms. After that, **PULSOUT tPIn, 1200** sends the start pulse. Next, a series of **PULSOUT** commands calculates and sends the pulses of the correct durations for binary-1 and 0. For example, if **remoteCode.BIT3** is 1, the command **PULSOUT tPin**, **remoteCode.BIT3 * 300 + 300** sends a **PULSOUT** of 600, which is 1.2 ms, which is a binary 1 pulse. However, if **remoteCode.BIT3** is 0, then the **PULSOUT** command only sends a pulse of 300, which lasts 0.6 ms, which transmits a binary-0 pulse.

```
Send_Ir_Remote_Code:
  ' Change button values to remote codes.
  IF (remoteCode = 0) THEN remoteCode = 10
  IF (remoteCode <= 10) THEN remoteCode = remoteCode - 1

  remoteCode.BIT7 = 1                          ' Set bit-7 = 1 for TV

  LOW tPin                                     ' 555 signal off
  PAUSE 26                                     ' Rest between messages
  PULSOUT tPin, 1200                           ' Start pulse

  PULSOUT tPin, remoteCode.BIT0 * 300 + 300  ' Bit-0 pulse
  PULSOUT tPin, remoteCode.BIT1 * 300 + 300  ' Bit-1 pulse
  PULSOUT tPin, remoteCode.BIT2 * 300 + 300  ' Bit-2 pulse
  PULSOUT tPin, remoteCode.BIT3 * 300 + 300  ' Etc...
    .
    .
    .
  RETURN
```

Each **PULSOUT** command in the **Send_Ir_Remote_Code** subroutine has an argument that multiplies one of the bits in the remoteCode variable by 300 before adding 300. Without parenthesis, the order of operation is left to right. If **remoteCode.BIT2** = 0 then the result of **remoteCode.BIT2 * 300 + 300** is 0 * 300 + 300 = 300. **If remoteCode.BIT3** is 1, then the result of **remoteCode.BIT3** is 1 * 300 + 300 = 300 + 300 = 600. More generally, the result of these expressions in each **PULSOUT** command's duration argument is 600 if the given bit is 1, or 300 if it's 0, which causes the **PULSOUT** command to turn the 555 timer's 38 kHz signal on, either for a duration of 1.2 ms or 0.6 ms.

Even though the same results can be obtained by the **FOR...NEXT** loop below, it won't work in the program because of signal timing considerations. Unlike the **remoteCode.BITX * 300 + 300** calculation, which takes around 500 µs, the code block below takes well over a millisecond. While a SONY TV set might or might not be forgiving enough to decode the signal, the example programs from this text that receive and decode IR messages are not that forgiving.

```
FOR index = 0 TO 11
  IF remoteCode.L0WBIT(index) = 1 THEN
    duration = 600
  ELSE
    duration = 300
  ENDIF
  PULSOUT tPin, duration
NEXT
```

The reason the way the code is written makes such a big difference in IR remote transmission is because the calculations are performed between **PULSOUT** commands. The **FOR...NEXT** loop shown above, which takes more than a millisecond, throws the timing off so much, that this text's example programs can no longer decode the messages. On the other hand, the calculation **remoteCode.BITX * 300 + 300** takes about 500 µs, which is really close to the 600 µs rests between pulses shown in Figure 1-4 on page 6. It works, both with the Boe-Bot and SONY TVs.