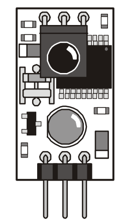
PARALLAX INC.

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

General: info@parallaxinc.com
Technical: stamptech@parallaxinc.com
Web Site: www.parallaxinc.com
Educational: www.stampsinclass.com

IR Buddy Demo Kit (#28016)
Infrared Control and Communications



Introduction

The IR Buddy is an intelligent peripheral that allows the BASIC Stamp to transmit and receive infrared control codes, and communicate (send/receive data packets) with other BASIC Stamps – all through a single I/O pin and without wires between devices. The IR Buddy has onboard IR transmit and receive hardware, as well as a communications controller that buffers incoming signals and manages data exchange with the BASIC Stamp.

What kind of things can be done with the IR Buddy? While the possibilities are many, here's a small list of ideas that can be realized with an IR Buddy and the Parallax BASIC Stamp:

- Device controller using a standard television/VCR remote control
- Fool-proof beam interrupt detection for alarm systems
- Wireless data exchange between BASIC Stamps and IR master-slave control

Packing List

Verify that your IR Buddy Demo Kit is complete in accordance with the list below:

- (2) IR Buddy Control/Communications modules (#550-28016)
- (5) Red LEDs (#350-00006)
- (5) 470 ohm resistors (#150-04710)
- Jumper wires (#800-00016)
- Documentation

Note: IR Buddy demonstration software files may be downloaded from www.parallax.com.

Features

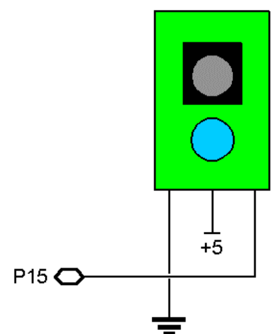
- Simple, single-wire connection [bi-directional] with BASIC Stamp
- 3-pin male header for connection to breadboards and standard 0.1" sockets
- Auto-baud detection (2400, 4800, 9600) for Stamp-to-IR Buddy communications
- Transmit RC-5 remote control codes; with programmable repeat and modulation frequency
- Receive RC-5 control codes; buffer up to four separate key press events
- Send and receive buffered 8-byte data packets between BASIC Stamps through wireless IR link
- Loopback test checks Stamp-to-IR Buddy communications connection
- Low current operation: 2 mA quiescent, 20 mA when transmitting
- Sleep mode reduces current to 14 uA

Connections

Connecting the IR Buddy to the BASIC Stamp is through a single I/O pin. The BASIC Stamp will use **SEROUT** to send commands and requests to the IR Buddy, and **SERIN** to accept data. The IR Buddy also requires a regulated five-volt power supply (Vdd) and connection to ground (Vss).

Since the IR Buddy uses a single bi-directional serial line, communications must use one of the open baud modes. The open baud modes leave the I/O pin in a high-impedance (input) state when finished with the **SEROUT** command, preventing a possible conflict with data arriving from the IR Buddy.

Figure 1. IR Buddy Connections



How It Works

At its core, the IR Buddy is designed to transmit and receive Philips RC-5 control codes. The RC-5 protocol was designed by Philips for use in consumer electronics and is very robust. The IR Buddy capitalizes on the strength of this protocol and, with specialized internal software, uses a similar technique to allow the BASIC Stamp to transmit and receive eight-byte data packets that can be used for any purpose a particular application requires.

An onboard microcontroller manages communications with the BASIC Stamp and controls the infrared transmit and receive hardware.

IR Buddy Commands

$72 Receive RC-5 Key Codes

Use: `SEROUT` ***`pin, baud,`*** `[$72,` ***`holdoff`***`]`

pin	variable or constant value: 0 to 15
baud	variable or constant value for 2400, 4800 or 9600 baud; open-mode
holdoff	variable or constant value: 0 to 255 ms; delays serial output to BASIC Stamp

This command directs the IR Buddy to receive and buffer up to four RC-5 key codes from a standard consumer electronics remote control or companion IR Buddy. After issuing the receive command, the BASIC Stamp should prepare to accept the codes into a variable array. The period specified in the holdoff parameter gives the BASIC Stamp time to get ready for the incoming data.

In a typical application, the command to receive RC-5 codes would be immediately followed with a **SERIN** function to accept the data.

Example:

```
SEROUT pin, baud, [$72, 10]
SERIN  pin, baud, [STR buffer\8\255]
```

The first line above sends the RC-5 receive command and specifies a 10 millisecond holdoff period. The second line will accept the data into an eight-byte array called buffer. The \8 parameter tells **SERIN** to accept eight bytes of data. The trailing \255 parameter tells **SERIN** to terminate if the value 255 is encountered in the input stream.

This syntax allows the BASIC Stamp to retrieve all the available key codes from the IR Buddy, regardless of the actual number buffered (up to four codes; two bytes per code). The IR Buddy uses the value 255 to identify an unused position in the buffer. The end of the buffered data is signified with the value 254. For example, if the IR Buddy buffered key codes for numbers "1" and "2" from a typical television remote, the IR Buddy buffer would contain the following data:

0	**1**	**0**	**2**	**254**	**255**	**255**	**255**

The first two bytes are the system and command codes for "1," the second two bytes are the system and command codes for "2." The fifth byte, 254, signifies the end of data in the buffer. The rest of the buffer is padded with the value 255. Note that (with the syntax specified above) **SERIN** will terminate upon reading the first 255.

The IR Buddy uses one additional special value: 253. This value tells the BASIC Stamp that the last key code has been repeated without an intermediate release. The value 253 will appear in the buffer when a remote key is pressed and held.

$74 Transmit RC-5 Key Code

Use: `SEROUT` ***`pin, baud,`*** `[$74,` ***`repeats, modulation, system, command`***`]`

pin	variable or constant value: 0 to 15
baud	variable or constant value for 2400, 4800 or 9600 baud; open-mode
repeats	variable or constant value: 0 to 255; number of repeats of this key code
modulation	variable or constant value: 30, 38 or 56; IR modulation frequency in kilohertz
system	variable or constant value: 0 to 31; system code
command	variable or constant value: 0 to 63; command value

This command directs the IR Buddy to transmit any one of the 2048 unique key codes using the Philips RC-5 protocol. Each key code consists of a five-bit system code and a six-bit command. This function will always send the key code once at the specified modulation frequency (30, 38 or 56 kHz), followed by the specified number of repeats.

Once the IR Buddy begins transmitting the key code, it will pull the serial line low to indicate its busy state. This line can be monitored by the BASIC Stamp to prevent resetting the IR Buddy or otherwise disrupting a transmission in progress.

Example:

```
TX_Code:
  SEROUT pin, baud, [$74, 0, 38, system, command]
  PAUSE 5

TX_Wait:
  IF (Ins.LowBit(pin) = 0) THEN TX_Wait
```

This example will cause the IR Buddy to send a single key code (specified in system and command) at the modulation frequency of 38 kilohertz. The **SEROUT** command is followed by a short **PAUSE** to give the IR Buddy time to pull the serial line low to indicate the its state. The next line of code will monitor the serial line, waiting for it to go high before continuing with the rest of the program.

$44 Transmit 8-Byte Data Packet

Use: `SEROUT` ***pin, baud,*** `[$44,` ***modulation, byte0, byte1, byte2, byte3, byte4, byte5, byte6, byte7***`]`

pin	variable or constant value: 0 to 15
baud	variable or constant value for 2400, 4800 or 9600 baud; open-mode
modulation	variable or constant value: 30, 38 or 56; IR modulation frequency in kilohertz
bytes	variable or constant values: 0 to 255; 8-byte packet

This command directs the IR Buddy to transmit an eight-byte data packet to a companion IR Buddy using the specified modulation frequency. Note that the transmission scheme is designed for reliability under harsh conditions and can take approximately 25 milliseconds per byte to transmit.

As above, the BASIC Stamp can monitor the serial line to determine the end of the transmission process.

Example:

```
TX_Packet:
  SEROUT pin, baud, [$44, 38, STR buffer\8]
  PAUSE 5

TX_Wait:
  IF (Ins.LowBit(pin) = 0) THEN TX_Wait
```

This example transmits the eight-byte array called buffer at a modulation frequency of 38 kilohertz to a companion IR Buddy and BASIC Stamp.

$64 Receive/Transfer 8-Byte Data Packet

Use: `SEROUT` ***pin, baud,*** `[$64]`

pin	variable or constant value: 0 to 15
baud	variable or constant value for 2400, 4800 or 9600 baud; open-mode

This command directs the IR Buddy to receive and buffer an 8-byte data packet. It is also used to signal the transfer of any data that has been buffered. This command is followed by a **SERIN** function.

Example:

```
SEROUT pin, baud, [$64]
SERIN  pin, baud, 1000, TO_Error, [STR buffer\8]
```

The first line above puts the IR Buddy in 8-byte receive/transfer mode and will initiate a transfer of any buffered data. Since the transfer will not take place until eight bytes have been received, the **SERIN** command that follows should use a timeout period and label to allow the BASIC Stamp program to proceed while waiting for data.

When data is available, it will be transferred to the BASIC Stamp in the eight-byte array called buffer.

$4C Loopback Test Mode

Use: `SEROUT` ***`pin, baud,`*** `[$4C,` ***`holdoff`***`]`

pin	variable or constant value: 0 to 15
baud	variable or constant value for 2400, 4800 or 9600 baud; open-mode
holdoff	variable or constant value: 0 to 255 ms; RX/TX delay and data byte

This function tests the serial connection between the BASIC Stamp and the IR Buddy. It is particularly useful when the connection is over a long wire run, especially in an electrically-noisy environment. The Loopback test can be used to establish the highest error-free baud rate between the BASIC Stamp and the IR Buddy.

The BASIC Stamp should expect four bytes returned at the transmission baud rate after a delay specified in holdoff. The first two bytes are the holdoff value, the third should be 254, the fourth 255.

Example:

```
SEROUT pin, baud, [$4C, holdoff]
SERIN  pin, baud, 300, TO_Error, [STR buffer\4]
```

If, for example, the value 127 is sent in the holdoff parameter, the BASIC Stamp should expect the following packet to arrive after approximately 127 milliseconds:

127   127   254   255

Resetting The IR Buddy

Before using the IR Buddy or when switching modes, it is important to reset its controller. This process is also useful to clear any buffered data. To reset the IR Buddy, take the serial line low for at least five milliseconds, then place it in an input mode for at least 50 milliseconds to allow the reset operation of complete.

Example:

```
LOW pin
PAUSE 5
INPUT pin
PAUSE 50
```

IR Buddy Application: RC-5 Reception and Display

This application demonstrates the reception and decoding of RC-5 key codes from a (Philips or compatible) consumer electronics remote control. It is useful for mapping key values from the remote for use in control projects.

The program puts the IR Buddy into RC-5 receive mode then retrieves buffered key codes for display. Note that the program checks for and indicates End-of-Buffer and Repeated Key values. A **PAUSE** value of 1000 milliseconds between IR Buddy access requests simulates normal program activity and will demonstrate the IR Buddy's ability to buffer multiple key code data.

Connect the IR Buddy to the BASIC Stamp as shown if Figure 1.

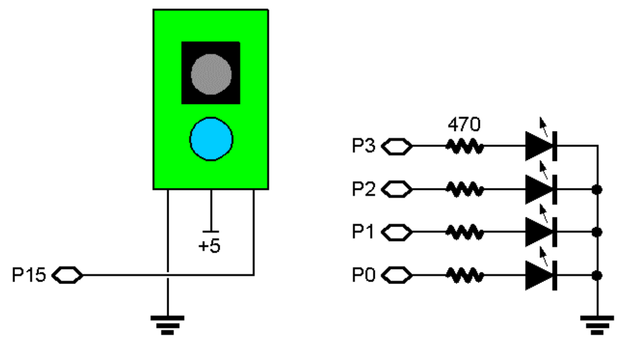
```
' -----[ Title ]-----------------------------------------------------------
'
' IRB RC-5 Monitor.BS2
' {$STAMP BS2}


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

IRbSIO          CON     15                      ' IR Buddy serial I/O


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

IRbRc5Rx        CON     $72                     ' RC-5 protocol RX

IRb96           CON      84 + $8000             ' 9600 baud, open
IRb48           CON     188 + $8000             ' 4800 baud, open
IRb24           CON     396 + $8000             ' 2400 baud, open
IRbBaud         CON     IRb96

KeyRpt          CON     253                     ' repeated key
BufEnd          CON     254                     ' end of buffer

CrsrXY          CON      2                      ' DEBUG position command
ClrEOL          CON     11                      ' Clear DEBUG line to right


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

buffer          VAR     Byte(8)                 ' RC-5 RX buffer
idx             VAR     Byte                    ' loop counter


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

Setup:
  GOSUB IR_Buddy_Reset

  PAUSE 250                                     ' let DEBUG window open
  DEBUG CLS
  DEBUG "IR Buddy RC-5 RX Monitor", CR
  DEBUG "------------------------", CR
  DEBUG CR
  FOR idx = 0 TO 3
```

```
    DEBUG "System.... ", CR
    DEBUG "Command... ", CR
  NEXT


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

Main:
  SEROUT IRbSIO, IRbBaud, [IRbRc5Rx, 10]        ' start RC-5 RX
  SERIN  IRbSIO, IRbBaud, [STR buffer\8\255]    ' get data

Show_Buffer:
  FOR idx = 0 TO 7
    DEBUG CrsrXY, 11, (idx + 3)                 ' move to display line
    DEBUG DEC buffer(idx)                       ' display buffer value
    IF (buffer(idx) = BufEnd) THEN End_Of_Buffer
    IF (buffer(idx) = KeyRpt) THEN Repeated_Key
    DEBUG ClrEOL                                ' clear old message
  NEXT
  GOTO Loop_Pad

End_Of_Buffer:
  DEBUG " (End of Buffer)"
  GOTO Clear_Old_Data

Repeated_Key:
  DEBUG " (Repeated Key)"

Clear_Old_Data:
  idx = idx + 1                                 ' point to next line
  IF (idx > 7) THEN Loop_Pad                    ' done?
  DEBUG CrsrXY, 11, (idx + 3), ClrEOL           ' no; move to line & clear it
  GOTO Clear_Old_Data

Loop_Pad:
  PAUSE 1000                                    ' simulate program activity
  GOTO Main

  END


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

' Reset the IR Buddy.  This code is useful for clearing data from the RX
' buffer and prepping to switch modes.  Timing specific; do not change.

IR_Buddy_Reset:
  LOW IRbSIO                                    ' signal reset
  PAUSE 5
  INPUT IRbSIO                                  ' release reset signal
  PAUSE 50                                      ' allow time for reset actions
  RETURN
```

IR Buddy Application: RC-5 Reception / Device Control

This application demonstrates the use of the IR Buddy and BASIC Stamp as a device controller. Signals are accepted from a (Philips or compatible) consumer electronics remote control and converted to appropriate channel control commands. For the demonstration program, LEDs are used to indicate the state of each control output.

The program will control four devices (outputs) using numeric buttons **[1]** though **[4]** on the remote. The **[Mute]** button is used to turn all outputs off. The system code for the program has been set to zero, the typical value used for television control.



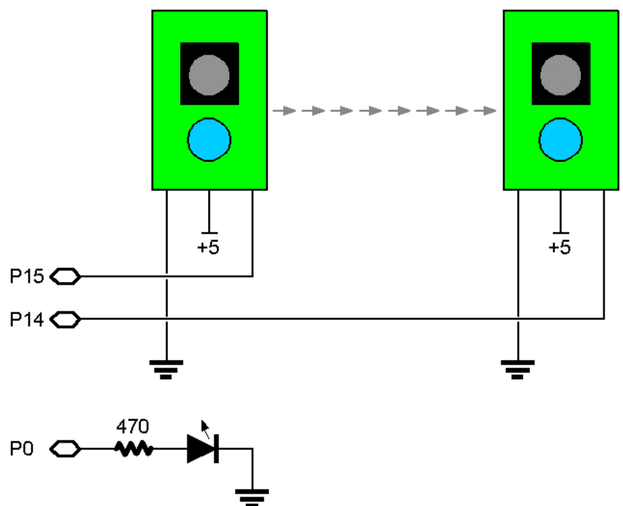

```
' -----[ Title ]-----------------------------------------------------------------
'
' IRB RC-5 Control.BS2
' {$STAMP BS2}


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

IRbSIO          CON     15                      ' IR Buddy serial I/O

Ports           VAR     OutA                    ' LED / device control pins
Port1           CON     0
Port2           CON     1
Port3           CON     2
Port4           CON     3


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

IRbRc5Rx        CON     $72                     ' RC5 protocol RX

IRb96           CON      84 + $8000             ' 9600 baud, open
IRb48           CON     188 + $8000             ' 4800 baud, open
IRb24           CON     396 + $8000             ' 2400 baud, open
IRbBaud         CON     IRb96

BufEnd          CON     254                     ' end of buffer
System          CON     0                       ' system code for this Stamp

On              CON      1
Off             CON      0
AllOff          CON     13                      ' "Mute" key on Philips remote
```

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

buffer          VAR     Byte(8)                     ' RC-5 RX buffer
idx             VAR     Nib                         ' loop counter
sysCode         VAR     Byte                        ' received system code
cmdCode         VAR     Byte                        ' received command code


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

Setup:
  Ports = Off                                       ' all outputs off
  DirA = %1111                                      ' all ports are outputs

  GOSUB IR_Buddy_Reset


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

Main:
  SEROUT IRbSIO, IRbBaud, [IRbRc5Rx, 10]            ' start RC-5 RX
  SERIN  IRbSIO, IRbBaud, [STR buffer\8\255]        ' get data

Process_Commands:
  FOR idx = 0 TO 6 STEP 2
    sysCode = buffer(idx)                           ' extract system code
    IF (sysCode = BufEnd) THEN Loop_Pad             ' reached end of buffer
    IF (sysCode <> System) THEN Skip_Key            ' check for valid system code
      cmdCode = buffer(idx + 1)                     ' extract command

Check_All_Off:
    IF (cmdCode <> AllOff) THEN Check_Toggle
      Ports = Off                                   ' all outputs off

Check_Toggle:
    IF (cmdCode = 0) OR (cmdCode > 4) THEN Skip_Key
      TOGGLE (cmdCode - 1)

Skip_Key:
  NEXT

Loop_Pad:
  PAUSE 500                                         ' give IR Buddy time to work
  GOTO Main

  END


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

' Reset the IR Buddy.  This code is useful for clearing data from the RX
' buffer and prepping to switch modes.  Timing specific; do not change.

IR_Buddy_Reset:
  LOW IRbSIO                                        ' signal reset
  PAUSE 5
  INPUT IRbSIO                                      ' release reset signal
  PAUSE 50                                          ' allow time for reset actions
  RETURN
```

IR Buddy Application: Intelligent Beam-Break Detection

This application uses two IR Buddies and a BASIC Stamp to form an intelligent IR beam-break detector as might be used in an alarm system. The first IR Buddy sends a coded message to the second. If the second does not receive the message properly, an error is generated. If enough errors accumulate, an alarm output in enabled. Performance can be improved by shielding the IR Buddies from extraneous IR sources.

The use of the error accumulator allows the system to work in and be fine-tuned for IR-noisy environments. The use of random coded message prevents defeat – even from a duplicate circuit because the extreme difficulty of synchronizing the counterfeit circuit with transmit IR Buddy.




```
' -----[ Title ]-----------------------------------------------------------------
'
' IRB Beam Break.BS2
' {$STAMP BS2}


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

IRbTX           CON     15                      ' transmitter IRB
IRbRX           CON     14                      ' receiver IRB

AlarmLED        CON     0


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

IRbRc5Tx        CON     $74                     ' RC-5 protocol TX
IRbRc5Rx        CON     $72                     ' RC-5 protocol RX

IRbMod          CON     38                      ' modulation freq: 30, 38 or 56

IRb96           CON      84 + $8000             ' 9600 baud, open
IRb48           CON     188 + $8000             ' 4800 baud, open
IRb24           CON     396 + $8000             ' 2400 baud, open
IRbBaud         CON     IRb96
```

```
Busy            CON     0                       ' IRB is transmitting
ErrorLevel      CON     5                       ' max errors before alarm

CrsrXY          CON      2                      ' DEBUG position command
ClrEOL          CON     11                      ' Clear DEBUG line to right


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

randVal         VAR     Word                    ' pseudo-random value
sysOut          VAR     Byte                    ' system code for RC-5 TX
cmdOut          VAR     Byte                    ' command code for RC-5 TX
sysIn           VAR     Byte                    ' system code for RC-5 RX
cmdIn           VAR     Byte                    ' command code for RC-5 RX
errors          VAR     Nib                     ' error count


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

Setup:
  GOSUB IR_Buddy_Reset                          ' reset TX and RX side


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

Main:
  RANDOM randVal                                ' create pseudo-random value
  sysOut = randVal.HighByte & %00011111         ' extract system value
  cmdOut = randVal.LowByte & %00111111          ' extract command value

TX_Code:
  SEROUT IRbTX, IRbBaud, [IRbRc5Tx, 0, IRbMod, sysOut, cmdOut]
  PAUSE 5                                       ' let IRB grab SIO line

TX_Wait:
  IF (Ins.LowBit(IRbTX) = Busy) THEN TX_Wait    ' wait for TX to end

RX_Code:
  SEROUT IRbRX, IRbBaud, [IRbRc5Rx, 10]         ' get codes from other side
  SERIN  IRbRX, IRbBaud, [STR sysIn\2\254]      ' expecting just two bytes

Display:
  DEBUG Home                                    ' display status
  DEBUG "Out", TAB, DEC3 sysOut, TAB, DEC3 cmdOut, CR
  DEBUG "In", TAB, DEC3 sysIn, TAB, DEC3 cmdIn, CR, CR
  DEBUG "Errors: ", DEC errors, ClrEOL

Check_Codes:
  IF (sysIn <> sysOut) THEN Codes_Bad           ' check system code
  IF (cmdIn <> cmdOut) THEN Codes_Bad           ' check command code

Codes_Okay:
  errors = 0                                    ' clear errors
  LOW AlarmLED                                  ' alarm off
  GOTO Main

Codes_Bad:
  errors = errors + 1                           ' update error count
  IF (errors < ErrorLevel) THEN Main            ' continue if error count okay
```

```
Alarm_On:
  HIGH AlarmLED                                 ' alarm on
  PAUSE 1000

  ' other alarm code here

  GOTO Main
  END


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

' Reset the IR Buddy.  This code is useful for clearing data from the RX
' buffer.  Timing specific; do not change.

IR_Buddy_Reset:
  LOW IRbTX                                     ' signal reset
  LOW IRbRX
  PAUSE 5
  INPUT IRbTX                                   ' release reset signal
  INPUT IRbRX
  PAUSE 50                                      ' allow time for reset actions
  RETURN
```

Note: The transmit pattern from the IR Buddy is not like a laser, it is conical. Beam spread and interference can be reduced by placing the IR Buddies in cylindrical enclosures that are aligned with open ends facing each other from either side of the path to be monitored.

IR Buddy Application: Loopback Testing

In some applications, the IR Buddy may be separated from the BASIC Stamp through a long serial connection. The following code can be used to test the serial connection to the IR Buddy to determine the fastest error-free serial baud rate.

The program tests each baud rate (starting at 2400 baud) for all possible data values on the line. The transmission speed and buffer return will be displayed as the program runs. If a connection or reception error occurs, the program will halt and display an appropriate message.

```
' -----[ Title ]-----------------------------------------------------------
'
' IRB Loopback.BS2
' {$STAMP BS2}


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

IRbSIO          CON     15                      ' IR Buddy serial I/O


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

IRbLoopback     CON     $4C                     ' loopback test

IRbMod          CON     38                      ' modulation freq: 30, 38 or 56

IRb96           CON      84 + $8000             ' 9600 baud, open
IRb48           CON     188 + $8000             ' 4800 baud, open
IRb24           CON     396 + $8000             ' 2400 baud, open

CrsrXY          CON      2                      ' DEBUG position command
ClrEOL          CON     11                      ' Clear DEBUG line to right


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

testNum         VAR     Nib                     ' test number (fore each baud)
testVal         VAR     Byte                    ' test value
buffer          VAR     Byte(4)                 ' receive buffer
idx             VAR     Nib                     ' loop counter

testBaud        VAR     Word


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

Setup:
  GOSUB IR_Buddy_Reset

  PAUSE 250                                     ' let DEBUG window open
  DEBUG CLS
  DEBUG "IR Buddy Loopback Test", CR
  DEBUG "----------------------", CR
  DEBUG CR
  DEBUG "Baud: ", CR
  DEBUG CR
```

```
  DEBUG "Data: "


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

Main:
  FOR testNum = 0 TO 2
    LOOKUP testNum, [2400, 4800, 9600], testBaud
    DEBUG CrsrXY, 6, 3, DEC testBaud            ' display test baud rate

    LOOKUP testNum, [IRb24, IRb24, IRb24], testBaud

    FOR testVal = 0 TO 255                      ' loop through holdoff values
      SEROUT IRbSIO, testBaud, [IRbLoopback, testVal]
      SERIN  IRbSIO, testBaud, 300, TO_Error, [STR buffer\4]

      FOR idx = 0 TO 3                          ' display rx buffer
        DEBUG CrsrXY, 6, (idx + 5)
        DEBUG DEC buffer(idx), ClrEOL, CR
      NEXT

      IF (buffer(0) <> testVal) THEN Packet_Error
      IF (buffer(1) <> testVal) THEN Packet_Error
      IF (buffer(2) <> 254) THEN Packet_Error
      IF (buffer(3) <> 255) THEN Packet_Error
    NEXT ' testVal
  NEXT ' testNum

Test_Complete:
  DEBUG CR, "Test Complete - PASS"
  END

TO_Error:
  DEBUG CR, "Timeout Error - check connection"
  END

Packet_Error:
  DEBUG CR, "Packet Error"
  END


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

' Reset the IR Buddy.  This code is useful for clearing data from the RX
' buffer and prepping to switch modes.  Timing specific; do not change.

IR_Buddy_Reset:
  LOW IRbSIO                                    ' signal reset
  PAUSE 5
  INPUT IRbSIO                                  ' release reset signal
  PAUSE 50                                      ' allow time for reset actions
  RETURN
```

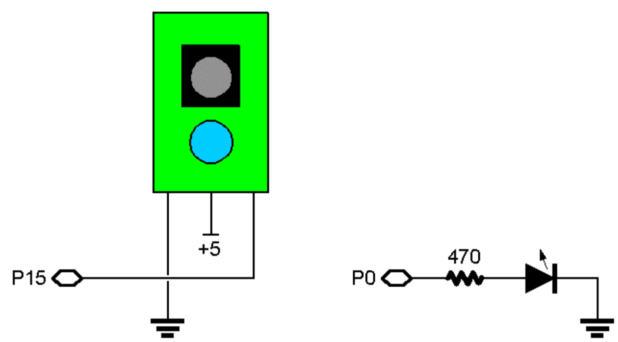
IR Buddy Application: Master Controller

This application demonstrates the use of data communications between BASIC Stamps using the IR Buddy's 8-byte data packet mode. This program, the master controller, sends an 8-byte packet to a slave controller. The slave will validate the packet and if the transmission was successful will act on it and respond accordingly. If the packet was not received properly, the slave will respond with an error message and the master will retransmit the the packet.

In this application the 8-byte packet is structured accordingly:

0	Header	
1	Command	(i.e., Light LEDs on slave)
2	Data byte 1	(i.e., LED pattern for slave LEDs)
3	Data byte 2	
4	Data byte 3	
5	Data byte 4	
6	Data byte 5	
7	Checksum	

The use of a pre-defined header byte and a simple checksum algorithm allows the slave to validate the packet before acting on the command. The software validation, combined with the IR Buddy's robust modulation scheme ensures reliable transfers under the worst of conditions.




```
' -----[ Title ]-----------------------------------------------------------
'
' IRB Master.BS2
' {$STAMP BS2}


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

IRbSIO          CON     15                      ' IR Buddy serial I/O
TxLED           CON      0                      ' transmitter on indicator


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

IRbDataTx       CON     $44                     ' 8-byte data transmit
IRbDataRx       CON     $64                     ' 8-byte data receive

IRbMod          CON     38                      ' modulation freq: 30, 38 or 56
```

```
IRb96           CON      84 + $8000             ' 9600 baud, open
IRb48           CON     188 + $8000             ' 4800 baud, open
IRb24           CON     396 + $8000             ' 2400 baud, open
IRbBaud         CON     IRb96

CrsrXY          CON      2                      ' DEBUG position command
ClrEOL          CON     11                      ' Clear DEBUG line to right

STX             CON     $02
ACK             CON     $06
NAK             CON     $15

LightLeds       CON     $C0                     ' commands for slave
LedsOff         CON     $C1

Busy            CON     0                       ' IR Buddy is transmitting


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

buffer          VAR     Byte                    ' tx-rx buffer (8 bytes)
cmd             VAR     Byte
data1           VAR     Byte
data2           VAR     Byte
data3           VAR     Byte
data4           VAR     Byte
data5           VAR     Byte
chkSum          VAR     Byte

header          VAR     buffer                  ' tx header
ackByte         VAR     buffer                  ' rx status
lastCmd         VAR     Byte                    ' last command sent
rxChkSum        VAR     Byte                    ' comparison checksum

counter         VAR     Nib                     ' counter for slave display
idx             VAR     Nib                     ' loop counter


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

Setup:
  PAUSE 250                                     ' let DEBUG window open
  DEBUG CLS
  DEBUG "IR Buddy Master-Slave Demo", CR
  DEBUG "--------------------------", CR
  DEBUG CR
  DEBUG "TX: ", CR
  DEBUG "RX: ", CR
  DEBUG CR
  DEBUG "Status: "


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

Build_Packet:
  GOSUB Clear_Buffer
  header = STX                                  ' build TX packet
  cmd = LightLeds
  data1 = counter
  GOSUB Make_CheckSum
```

```
  GOSUB Show_TX_Packet
  lastCmd = cmd                                 ' save for RX check

TX_Packet:
  GOSUB IR_Buddy_Reset
  HIGH TxLED
  SEROUT IRbSIO, IRbBaud, [IRbDataTx, IRbMod, STR buffer\8]
  PAUSE 5                                       ' let IRB grab SIO line

TX_Wait:
  IF (Ins.LowBit(IRbSIO) = Busy) THEN TX_Wait
  LOW TxLED

RX_Packet:
  GOSUB IR_Buddy_Reset
  SEROUT IRbSIO, IRbBaud, [IRbDataRx]           ' prep for 8-byte RX
  SERIN  IRbSIO, IRbBaud, 1000, TO_Error, [STR buffer\8]
  GOSUB Show_RX_Packet                          ' display received bytes
  DEBUG CrsrXY, 8, 6                            ' prep for status report

Check_RX_Packet:
  IF (header <> ACK) THEN NAK_Error             ' check packet bytes
  IF (cmd <> lastCmd) THEN Packet_Error
  rxChkSum = chkSum                             ' save rx checksum
  GOSUB Make_CheckSum                           ' calc checksum of rx packet
  IF (rxChkSum <> chkSum) THEN Packet_Error     ' compare checksum values

Good_Packet:
  DEBUG "Good Packet", ClrEOL
  counter = (counter + 1) & $0F                 ' update counter
  PAUSE 500
  GOTO Build_Packet                             ' build & send new packet

NAK_Error:
  DEBUG "Slave returned NAK", ClrEOL
  GOTO Build_Packet                             ' rebuild & resend

Packet_Error:
  DEBUG "Packet error", ClrEOL
  GOTO Build_Packet

TO_Error:
  DEBUG CrsrXY, 8, 6
  DEBUG "Timeout error", ClrEOL
  PAUSE 250                                     ' give slave time to reset
  GOTO Build_Packet

  END


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

' Reset the IR Buddy.  This code is useful for clearing data from the RX
' buffer and prepping to switch modes.  Timing specific; do not change.

IR_Buddy_Reset:
  LOW IRbSIO                                    ' signal reset
  PAUSE 5
  INPUT IRbSIO                                  ' release reset signal
  PAUSE 50                                      ' allow time for reset actions
```

```
  RETURN


Clear_Buffer
  FOR idx = 0 TO 7
    buffer(idx) = 0
  NEXT
  RETURN


Make_CheckSum:                                  ' checksum of bytes 0 to 6
  chkSum = 0
  FOR idx = 0 TO 6
    chkSum = chkSum + buffer(idx)
  NEXT
  RETURN


Show_TX_Packet:
  DEBUG CrsrXY, 4, 4, ClrEOL                    ' clear last RX message
  DEBUG CrsrXY, 4, 3, ClrEOL                    ' clear last TX message
  GOTO Show_Packet_Data

Show_RX_Packet:
  DEBUG CrsrXY, 4, 4, ClrEOL                    ' clear last RX message

Show_Packet_Data:                               ' display packet bytes
  FOR idx = 0 TO 7
    DEBUG HEX2 buffer(idx), " "
  NEXT
  RETURN
```

Note the use of timeout value and label when the master is expecting a packet from the slave controller. This will clear the reset the IR Buddy in the event of a blocked transmission path and clear any partial data from its buffer, allowing the master and slave to resynchronize with each other.

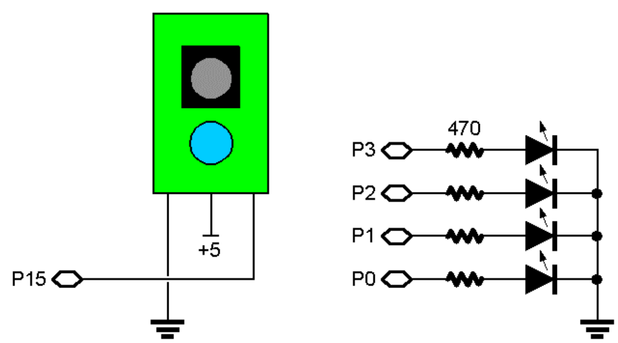
IR Buddy Application: Slave Controller

The application receives the an 8-byte command and data packet from the master controller and, if the packet is valid, will act on the command. For this demonstration, the command is to light LEDs connected to pins 0 – 3 of the BASIC Stamp. The pattern to place on the LEDs is transmitted in byte 2 of the packet.

In order to prevent possible problems, the slave performs several checks on the packet before acting on the command. The first check is to validate the header byte. If the header is proper, the slave will run the checksum routine used by the master on the packet and compare it with the value sent. If the checksums match, the slave assumes the packet is good and will continue with the command.

If the header or checksums are not valid, the slave responds by placing a NAK byte in the header and sending the packet back to the master. This will alert the master that the packet arrived with errors and cause the master to resend the last command/data packet.

If the packet is good, the slave will act on the command then return the packet with an ACK in the header and any requested data to the master along with a new packet checksum. This will let the master know that the last command was received and acted upon.




```
' -----[ Title ]-----------------------------------------------------------
'
' IRB Slave.BS2
' {$STAMP BS2}


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

IRbSIO          CON     15                      ' IR Buddy serial I/O
LEDs            VAR     OutA                    ' LED control outputs


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

IRbDataTx       CON     $44                     ' 8-byte data transmit
IRbDataRx       CON     $64                     ' 8-byte data receive

IRbMod          CON     38                      ' modulation freq: 30, 38 or 56

IRb96           CON      84 + $8000             ' 9600 baud, open
IRb48           CON     188 + $8000             ' 4800 baud, open
IRb24           CON     396 + $8000             ' 2400 baud, open
```

```
IRbBaud         CON     IRb96

CrsrXY          CON      2                        ' DEBUG position command
ClrEOL          CON     11                        ' Clear DEBUG line to right

STX             CON     $02
ACK             CON     $06
NAK             CON     $15

LightLeds       CON     $C0                       ' commands for slave
LedsOff         CON     $C1

Busy            CON     0                         ' IR Buddy is transmitting


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

buffer          VAR     Byte                      ' rx-tx buffer (8 bytes)
cmd             VAR     Byte
data1           VAR     Byte
data2           VAR     Byte
data3           VAR     Byte
data4           VAR     Byte
data5           VAR     Byte
chkSum          VAR     Byte

header          VAR     buffer                    ' rx packet
ackByte         VAR     buffer                    ' ack/nak byte
rxChkSum        VAR     Byte                      ' comparison checksum

idx             VAR     Nib                       ' loop counter


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

Setup:
  LEDs = %0000                                    ' LEDs off
  DirA = %1111                                    ' make LED pins outputs


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

Main:
  GOSUB IR_Buddy_Reset
  SEROUT IRbSIO, IRbBaud, [IRbDataRx]             ' prep for 8-byte RX

RX_Packet:
  SERIN IRbSIO, IRbBaud, 2000, TO_Error, [STR buffer\8]

Check_RX_Packet:
  IF (header <> STX) THEN Packet_Error
  rxChkSum = chkSum                               ' save rx checksum
  GOSUB Make_CheckSum                             ' calc checksum of rx packet
  IF (rxChkSum <> chkSum) THEN Packet_Error       ' compare checksum values

Process_Command:
  IF (cmd <> LightLEDs) THEN Packet_Error         ' is command valid?
  LEDs = data1                                    ' yes, move data to LEDs
```

```
Good_Packet:                                    ' responsd to good packet
  header = ACK

  ' change data fields if required by Master

  GOSUB Make_CheckSum
  GOTO TX_Packet

Packet_Error:                                   ' respond to bad packet
  header = NAK

TX_Packet:
  GOSUB IR_Buddy_Reset
  SEROUT IRbSIO, IRbBaud, [IRbDataTx, IRbMod, STR buffer\8]
  PAUSE 5                                       ' let IRB grab SIO line

TX_Wait:
  IF (Ins.LowBit(IRbSIO) = Busy) THEN TX_Wait
  GOTO Main

TO_Error:

  ' put code here that handles timeout error

  GOTO Main                                     ' reset, look for new packet

  END

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

' Reset the IR Buddy.  This code is useful for clearing data from the RX
' buffer and prepping to switch modes.  Timing specific; do not change.

IR_Buddy_Reset:
  LOW IRbSIO                                    ' signal reset
  PAUSE 5
  INPUT IRbSIO                                  ' release reset signal
  PAUSE 50                                      ' allow time for reset actions
  RETURN


Make_CheckSum:                                  ' checksum of bytes 0 to 6
  chkSum = 0
  FOR idx = 0 TO 6
    chkSum = chkSum + buffer(idx)
  NEXT
  RETURN
```

Note the use of timeout value and label when the slave is expecting a packet from the master controller. This will clear the reset the IR Buddy in the event of a blocked transmission path and clear any partial data from its buffer, allowing the master and slave to resynchronize with each other.

Also note that the slave does not respond unless addressed by the master. This prevents synchronization problems and allows the code to be used for multi-slave applications. In multi-slave applications, part of the data structure would be the slave address.