

PCBUDDY INTERFACE

Installation and Programming Manual

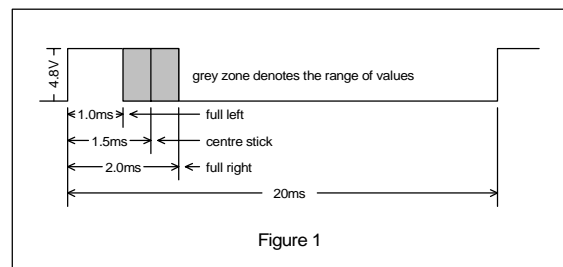
Product Overview

Introduction

The PCBuddy Interface is designed to be used with standard Radio Control equipment to enable remote robotic vehicles to be controlled using a computer program. At the transmitting end the equipment configuration is a PC with the PCBuddy Interface plugged into a COM port, linked to an R/C transmitter via its training socket, or 'buddy-box interface'. At the receiving end a standard R/C receiver may be connected to servos, electronic motor speed controllers or any of the many R/C auxiliary-function accessories available.

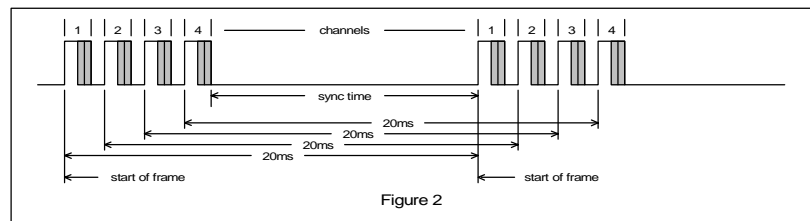
Servo Control Frames

A typical R/C servo will provide, at its output arm, a mechanical movement of approximately ± 45 degrees about a nominal centre position. The precise position within this range is determined by the width of the input pulse sent to it by the receiver and the range over which the input pulse varies is typically 1ms to 2ms. These limit values provide the extremes of movement and a pulse width of 1.5ms provides the nominal centre position. In order to assure best performance the servo requires that the control pulses are sent periodically and, although servos are usually fairly tolerant of the exact rate, around 50 times per second, or every 20ms, is normal. Figure 1 shows a typical servo control frame waveform.



Multi-Channel Frame

In order to control many servos using a single radio link the servo control frames of multiple servos are combined to form a Multi-Channel frame as shown in Figure 2. This is the basic signal sent by the transmitter to the receiver, which then



separates out the individual servo control frames and distributes them to each servo. After the final channel has been transmitted and before the start of the next frame there is a period containing no information, known as the sync time. The receiver uses this to establish the frame boundaries thus ensuring that the channel information is distributed correctly to each servo.

Downloading Servo Position Data

The PCBuddy Interface also makes use of the sync time. Having sent the final channel terminating pulse to the transmitter it signals to the PC that it is ready to receive a channel information update, if one is available. If the PC has a new data set it transmits it and the new channel data received is presented to the transmitter in the next frame. Figure 3, overleaf, shows a single 8-channel frame including a data update. If no update is ready then the servo control pulse information from the previous frame is repeated.

Trace Annotation

The upper trace, denoted R1, shows the PCBuddy Interface output signal and depicts a full set of 8 channels.

The cursor bars bracket Channel 8 with the dotted cursor aligned with the start of the final channel-terminating pulse.

The middle trace, denoted 2, shows the CTS control signal sent to the PC by the PCBuddy Interface indicating that it is ready to receive new data.

The lower trace, denoted 1, shows the activity on the RS232 transmit line from the PC with the individual data bytes clearly identifiable.

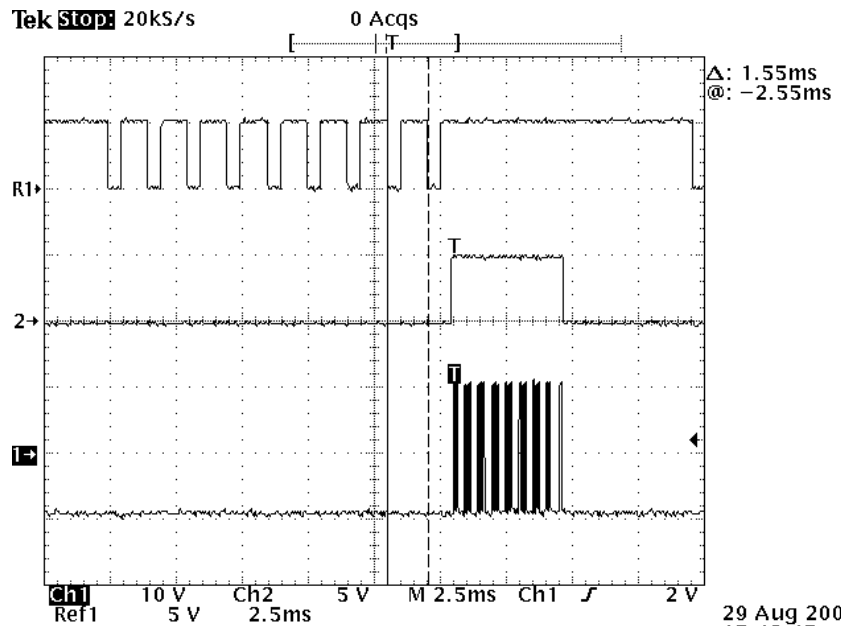


Figure 3

Modulating the Transmitted Carrier

The signal accepted by the transmitter, via its training socket, from the PCBuddy Interface is used to modulate the transmitted carrier. Depending on the design of a particular transmitter this signal may be of the form shown in Figure 2 or, possibly, an inverted version. The PCBuddy Interface may be factory configured to provide either form thus ensuring its use with the widest range of transmitters.

Using the Demonstration Facility

Introduction

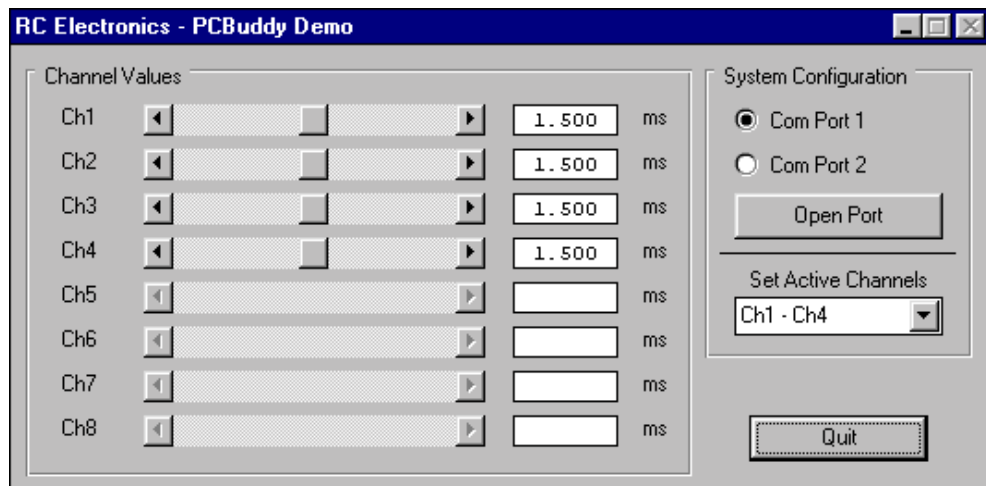
In order to verify the unit's operation, and to provide a possible starting point for further program development, a small PC application complete with the source code is included as part of the product package. A screen-shot of the application window is shown in Figure 4 below.

Screen shot of the Demonstration Application window showing the default settings at start-up.

The first four channels are set as active and their values set to the mid-position. Channels 5 to 8 are inactive.

The default COM Port is set to COM1.

Figure 4



The following suggested sequence is designed to minimise the risk of unwanted transmissions and thus the risk of interference to others. *Note that these instructions presume that you have obtained the self-extracting archive file, 'PCBUDDY.EXE', from the 'downloads' page of the RC Electronics web-site and placed it in a suitable folder:*

<http://www.rc-electronics.co.uk/downloads>

Connecting to the PC

Ensuring that all equipment is powered down connect the PCBuddy Interface to the PC using a free COM port, noting its identifying number, but do not make a connection to the transmitter yet.

Installing and Running the Demonstration Software

Power up the PC and when it has booted run the self-extracting archive file, 'PCBUDDY.EXE', using either the Windows Start Menu and selecting the 'Run' menu item or by double-clicking on the filename from within Windows Explorer. Either method will install the demonstration application onto your PC, which may then be invoked from either the Program Menu or from within Windows Explorer. When the demonstration application has loaded it should appear as shown in Figure 4 on the previous page.

Selecting and Opening a COM Port

Select the previously noted COM port number using the appropriate radio button and click the <Open Port> button. After a moment or two the button caption should change to 'Close Port' indicating that the COM port is open and allocated to the demonstration application. If a 'beep' is heard from the PC speaker and the button caption fails to change this indicates that an error occurred during the application's attempt to obtain access to the COM port resource. The most common cause of this failure is that the port is already in use by another program. *Note that the COM port, under control of the demonstration application software, powers the PCBuddy Interface and, therefore, a COM port Open failure will leave the PCBuddy Interface in an indeterminate state.*

Selecting the Number of Active Channels

The PCBuddy system will provide control of up to 8 R/C channels and any channel set as 'active' will have a corresponding 'slot' in the Multi-Channel frame. In order to ensure correct operation of the receiver frame synchronisation function it is advised that only the appropriate number of channels for the R/C equipment in use be enabled. This may be accomplished using the Set Active Channels drop-down selection box.

Adjusting the Channel Values

For each active channel the demonstration application provides a 'slider control' that may be used to set the corresponding servo control pulse width over the range 1ms to 2ms with the current setting indicated in the display window adjacent to each slider. The servo control pulse width may be quickly set to any value, or changed in 'real time', using the slider's 'scroll' function. Alternatively it may be changed in 4us or 100us steps by clicking on either of the end arrows or in the 'slider channel' respectively. Any channels set as inactive have their slider controls disabled.

Connecting to the Transmitter

The PCBuddy Interface should have been supplied with a training socket connector compatible with the specified transmitter. Power up the transmitter, insert the connector and configure the transmitter to use the training socket if appropriate. Connecting up a suitable receiver system incorporating a few servos should permit the control of servo position using the demonstration application.

Important Note.

When you wish the computer to have control of the R/C system you must activate the transmitter's spring-loaded training switch. With this switch in the normal position the transmitter sticks are active and any signal present at the training socket is ignored.

Description of Demonstration Code

Introduction

The source code for the demonstration application was written in Microsoft Visual Basic 3.0 and is included as part of the product package. It is intended to provide an understanding of the operation of the PCBuddy system and also as the basis of possible further program development by the user. This section describes various elements of the program using original source code fragments.

Initialising the Application (Form:Load)

The application configuration begins by setting all 8 channel sliders to their central values of 1.5ms and then goes on to add the various channel selection options into the drop-down list box before setting the default active channels to Ch1 to Ch4.

```
For Count% = 0 To 7                                'Set all channel values to the mid
  Let Form01.hsbChannelValue(Count%).Value = 125    'position ie 1.5ms.
Next Count%

Form01.cboSetActiveChannels.AddItem "Ch1 Only"      'Set up options in the Active
Form01.cboSetActiveChannels.AddItem "Ch1 - Ch2"    'Channels selection and set to
Form01.cboSetActiveChannels.AddItem "Ch1 - Ch3"    'default value of Ch1 to Ch4.
Form01.cboSetActiveChannels.AddItem "Ch1 - Ch4"
Form01.cboSetActiveChannels.AddItem "Ch1 - Ch5"
Form01.cboSetActiveChannels.AddItem "Ch1 - Ch6"
Form01.cboSetActiveChannels.AddItem "Ch1 - Ch7"
Form01.cboSetActiveChannels.AddItem "Ch1 - Ch8"
Let Form01.cboSetActiveChannels.ListIndex = 3
```

Configuring and Opening the PC COM Port

When the user clicks the <Open Port> button the appropriate port number is set along with the transmission parameters. The communications handshaking mode is set to 'RTS/CTS' which means that the PC hardware performs this function, thus relieving the software of the task, and an Open Port attempt is made. *Note that the use of 19200 baud is mandatory in order to maintain the frame rate of 22ms when all 8 channels are in use.*

```
If Form01.optCommPortSelect(0).Value = True Then   'Depending on the users choice first
  Let Form01.Comml.CommPort = 1                    'set the COM Port, 1 or 2.
Else
  Let Form01.Comml.CommPort = 2
End If
Let Form01.Comml.Settings = "19200,N,8,1"         'Then set the transmission params,
Let Form01.Comml.Handshaking = 2                  'and handshaking method to 'RTS/CTS'
Let Form01.Comml.PortOpen = True                  'and attempt to open the port.
```

On the assumption that the Open Port attempt is successful then the PCBuddy Interface hardware is powered up by the RTS signal and various settings changed to reflect the port state.

```
Let Form01.Comml.RTSEnable = True                  'If Port Open = OK then power up the
Let Form01.cmdCommPortAction.Caption = "Close Port" 'interface h/w, alter the caption
Let Form01.cmdQuit.Enabled = False                'on the button, disable Quit button,
Let PortIsOpen% = True                             'and set the Port State variable.
```

The receive buffer is then emptied of any stray characters and a set of servo control data is sent to the PCBuddy Interface thus updating the servo positions.

```
Let Form01.Comml.InBufferCount = 0                'Because the slider values may have
For Count% = 0 To (NumberOfActiveChannels% - 1)  'changed while Port was closed an
  Let Form01.Comml.Output = ChannelValueArray(Count%) 'update is performed to ensure the
Next Count%                                       'h/w reflects the current values.
Exit Sub                                          'Note the purge of the RX buffer.
```

This section of the code provides an error handler to cater for problems occurring during the Open Port attempt. It simply sets the various controls to the appropriate condition and alerts the user.

```
CommErrorHandler:                                     'Here if error occurred opening
Let Form01.cmdCommPortAction.Caption = "Open Port"   'COM Port. Ensure that the button
Let Form01.cmdQuit.Enabled = True                   'caption, the Quit button and the
Let PortIsOpen% = False                             'Port State variable are correctly
Beep: Beep                                           'set. Sound to alert the user.
Exit Sub
```

Configuring Active/Inactive Channels

This loop goes through each of the 8 channels determining whether they are active or not based on the integer variable 'NumberOfActiveChannels'. For each active channel the slider is enabled and its value display updated. If the channel is inactive its slider is disabled and its value is cleared.

```
Do
  If Count% < NumberOfActiveChannels% Then           'This loop runs 8 times and
    Let Form01.hsbChannelValue(Count%).Enabled = True 'for Active Channels enables
    Call hsbChannelValue_Scroll(Count%)              'the corresponding slider and
  Else                                               'updates the displayed value.
    Let Form01.hsbChannelValue(Count%).Enabled = False 'For Inactive Channels the
    Let Form01.lblChannelValue(Count%).Caption = ""   'slider bar is disabled and
  End If                                           'the displayed value cleared.
  Let Count% = (Count% + 1)
Loop Until Count% = 8
```

Displaying and Storing Channels Values

This line of code takes the current value of the appropriate channel slider, ranging from 0-250 and corresponding to 1ms (1000us) to 2ms (2000us), formats it and displays it in the corresponding display box.

```
Let Form01.lblChannelValue(Index%).Caption = _
  Format$(((Form01.hsbChannelValue(Index%).Value * 4) + 1000) / 1000), "0.000")
```

The value property of a slider is contained in a variable of signed integer type (the smallest numeric entity in Visual Basic 3.0 and requiring 16bits) but, because the slider controls are configured to produce numbers between 0 and 250, they can be contained in an 8bit entity. The PC serial port deals with 8bit quantities, ie bytes, and it is convenient to make use of a single character string variable to 'simulate' an unsigned 8bit numeric value. This line converts the 8bit numeric value into its character equivalent and stores it in the appropriate location in the channel value array.

```
Let ChannelValueArray(Index%) = Chr$(Form01.hsbChannelValue(Index%).Value)
```

Writing the Channel Values to the Interface

This code first verifies that the COM port is in use by this application. If it is, it goes on to check if a transmission is already in progress and, if so, waits for it to end. It then purges both the TX and RX buffers to ensure they contain no unwanted data before loading the latest set of channel data into the TX buffer. *Note that in the demonstration application only data for the active channels is actually sent to the PCBuddy Interface and, if required by user software, the alternative method for setting channels as inactive may be used (see Writing your own Code below).*

```
If PortIsOpen% = True Then                           'If Port is Open then check for a
  While Form01.Comml.CTSHolding = True: Wend         'transmission in progress and wait
  Let Form01.Comml.OutBufferCount = 0                'if so. Purge the Transmit buffer
  Let Form01.Comml.InBufferCount = 0                 'and Receive buffer.
  For Count% = 0 To (NumberOfActiveChannels% - 1)   'Load the new set of values
    Let Form01.Comml.Output = ChannelValueArray(Count%) 'for the Active Channels only
  Next Count%                                         'into the Transmit buffer.
End If
```

Writing your own Code

Introduction

This section is intended to provide a few hints and tips for people intending to write their own software applications for use with the PCBuddy Interface. It describes the operation of various key aspects of the PCBuddy system and provides, where appropriate, guidance on the limitations and any techniques that may be used to overcome them.

Communication Protocol Set-up

As has been shown in the preceding sections the frame rate is set at 22ms, which is a little slower than typical R/C systems. In order to achieve even this, with all 8 channels in use, it is necessary that the communications rate is set at 19200 baud with the minimum of 'overhead' in the form of parity and stop bits.

After sending the data for the final channel to the transmitter the PCBuddy Interface signals to the PC that it is ready to receive a data update by asserting the CTS signal. The PC must start to transmit the data packet within 1.25ms of this assertion in order for the interface to recognise it and failure to achieve this will result in the interface assuming that there is no new data available. It is, therefore, best to leave the detection of the CTS signal to the PC hardware, rather than application software, and so the COM port must be configured to use 'hardware handshaking', also sometimes known as RTS/CTS handshaking.

Powering the Interface Hardware from the PC

Because the PCBuddy Interface hardware requires only a small amount of power, and in order to remove the need for an external PSU, it is powered from one of the RS232 control lines, namely RTS. It is therefore imperative that any user application code set the RTS line to its active state when initialising the COM port and before any data is sent to the PCBuddy Interface. If, for any reason, it is desired to remove power from the interface while connected to the PC the RTS line may be set to its inactive state.

Reading the RX Buffer

Within the PCBuddy Interface hardware the RS232 TX and RX signals are linked which has the effect of every byte transmitted by the PC also being received. This has been done deliberately and, although the demonstration application does not make use of the facility, permits the application software to implement a form of 'interface detection' if required. However, even if it is intended not to make use of this feature, it should be noted that it is imperative that the application software purges the RX buffer periodically to prevent it over-filling. Failure to do this will result in the RTS line being de-asserted automatically by the PC hardware thus removing power from the interface. If this occurs then the Port must be closed and reopened again to reapply power to the PCBuddy Interface.

Programming the Servo Positions

For each active channel the PCBuddy Interface hardware produces a servo control pulse between 1ms and 2ms long. The precise length of each pulse is set by the value, with limits of 0 and 250, downloaded to the interface in respect of each channel with the formula for calculating the output pulse width based on the value downloaded as follows:-

$((\text{Channel Value} \times 4) + 1000)$ us. Dividing this by 1000 provides the result in ms.

Setting Inactive Channels

There are two methods that may be used to set unused channels as inactive. The first is to make use of a timeout mechanism within the PCBuddy Interface software and only transmit a limited number of values. Starting from channel 1 the number of bytes received will determine the number of consecutive active channels and, if desired, this may be changed on a frame-by-frame basis.

The second method is to always transmit 8 data values but set the value of any unused channel to FFh or 255. The interface hardware will read in all 8 data values but recognise the special case of FFh as representing an inactive channel. As with the first method it is not possible to place an inactive channel between active channels but the channel state, active or inactive, may be changed at will.

Writing Channel Values to the TX Buffer

The PCBuddy Interface hardware will accept a maximum of one new set of data values every 22ms. When the sliders are scrolled every slight movement produces a change in its value, which would normally be placed in the TX buffer ready for transmission. On a long scroll it is therefore possible to fill up the TX buffer with a large amount of data, which can only be 'digested' by the interface at the prescribed rate leading to the possibility of slow servo response times. The demonstration application overcomes this problem by overwriting any existing data contained within the TX buffer provided it is not currently being transmitted, ie it represents a complete set. For an application updating the servo positions less frequently than the frame rate of 22ms this aspect is of no concern.

PC Response Times to Data Request

There are two communications timeout mechanisms implemented within the PCBuddy Interface software. The first, with a length of 1.25ms, is started immediately after the assertion of the CTS signal informing the PC that the interface is ready to accept a new set of data. If there is data available the PC must start its transmission within this time or the interface will assume that there is no new data. The second timeout, which has a duration of 200us, is used to detect the premature end of transmission in the event of less than 8 channels of data being sent. The PC must ensure that following the completion of a byte transmission that the next one starts within this period otherwise the interface will assume that there is no further data and set all subsequent channels as inactive. Using the PC hardware to control the transmission process, as indicated by the demonstration application, should guarantee this.