

---

# USBee Toolbuilder User's Guide

Version 4.0

For the USBee Digital Test Pod

---

CWAV  
[www.cwav.com](http://www.cwav.com)  
[support@cwav.com](mailto:support@cwav.com)





### Table of Contents

- 1 OVERVIEW ..... 3**
- 1.1 THE USBEE SYSTEM..... 3
- 2 THE USBEE TOOLBUILDER VISUAL BASIC PROJECT..... 3**
- 2.1 USBEE TOOLBUILDER PROJECT CONTENTS..... 3
  - 2.1.1 Contents of the USBee Toolbuilder Visual Basic Program..... 3
- 2.2 MAKING YOUR OWN USBEE TOOLBUILDER PROJECT ..... 4
- 3 INSIDE THE USBEE SOURCE CODE..... 4**
- 3.1 INITIALIZING THE POD..... 4
  - 3.1.1 *ConnectToUSBeePod* ..... 4
    - 3.1.1.1 Calling Convention..... 4
    - 3.1.1.2 Example ..... 4
  - 3.1.2 *InitializePod*..... 5
    - 3.1.2.1 Calling Convention..... 5
    - 3.1.2.2 Example ..... 5
- 3.2 SETTING USBEE OUTPUT SIGNALS ..... 5
  - 3.2.1 *SetSignalStates* ..... 5
    - 3.2.1.1 Calling Convention..... 5
    - 3.2.1.2 Example ..... 6
    - 3.2.1.3 Timing ..... 6
    - 3.2.1.4 Bandwidth ..... 7
  - 3.2.2 *SetSignalStatesMultiple* ..... 7
    - 3.2.2.1 Calling Convention..... 7
    - 3.2.2.2 Example ..... 8
    - 3.2.2.3 Timing ..... 8
    - 3.2.2.4 Bandwidth ..... 10
  - 3.2.3 *GenerateData*..... 10
    - 3.2.3.1 Calling Convention..... 11
    - 3.2.3.2 Example ..... 11
    - 3.2.3.3 Timing ..... 11
    - 3.2.3.4 Bandwidth ..... 14
- 3.3 READING THE USBEE INPUT SIGNALS..... 14
  - 3.3.1 *GetSignalStates* ..... 14
    - 3.3.1.1 Calling Convention..... 14
    - 3.3.1.2 Example ..... 14
    - 3.3.1.3 Bandwidth ..... 14



## **1 Overview**

The USBee is a complete digital test bench in one compact and easy to use pod. Combined with the FREE, yet powerful, Windows® software the USBee performs the following functions:

- Data Logger
- Remote Controller
- Logic Analyzer
- Signal/Pattern Generator
- Frequency Counter
- Pulse Width Modulator

The USBee Digital Test Pod is also configurable and controllable using your own Visual Basic source code and the USBee Tool Builder project. This allows you to interface and control your own unique type of data bus or hardware. This document details how to use the USBee Toolbuilder software to make your own tools.

### **1.1 The USBee System**

The USBee System consists of the USBee Digital Test Pod connected to your Windows® 98 or Windows® 2000 PC through the USB cable, and to your digital circuit using the multicolored test leads and clips. Once connected and installed, the USBee can then be controlled using either the USBee Windows Software or your own USBee Toolbuilder software.

The USBee System is also expandable by simply adding more USBee pods for more channels and combined features.

## **2 The USBee Toolbuilder Visual Basic Project**

### **2.1 USBee Toolbuilder Project Contents**

#### **2.1.1 Contents of the USBee Toolbuilder Visual Basic Program**

USBEETOOLBUILDER.VBP	Visual Basic Project File
USBIF.BAS	Visual Basic USB interface routines
MAINFORM.FRM	Visual Basic Example Form
YOURTOOLMODULE.BAS	Example Visual Basic USBee Tool code

The USBee Toolbuilder also depends on the following files for proper operation. These files are installed when the USBee Digital Testbench CD is installed.

- 1) USBINF.DLL in the Windows/System directory
- 2) USBEE.BIX in the applications directory where your "USBeetool.exe" is running from
- 3) USBEE.INF in the Windows/INF directory
- 4) USBEE.SYS in the Windows/System32/Drivers directory
- 5) MSVBVM60.DLL in the Windows/System directory



## 2.2 Making your own USBee Toolbuilder Project

Follow these steps to create your own USBee Tool using the USBee Toolbuilder software.

1. First start by installing the USBee Digital Test Bench CD onto the computer you will be using to run your USBee Tool. This will install all of the necessary projects and support files needed.
2. Copy the Program Files\USBee\USBTool directory and contents to a new directory (for example \NewTool)
3. Start Visual Basic and Open the USBeeToolbuilder project in the new directory.
4. Make all of your necessary changes using the example tools and this document as a reference.
5. Run and debug your new tool using the Visual Basic interface.

## 3 Inside the USBee Source Code

### 3.1 Initializing the Pod

The USBee pod must be initialized to configure the pod to communicate with the host application before any pod operations can take place. There are 2 routines that can initialize the USBee pod: InitializePod and ConnectToUSBeePod.

#### 3.1.1 ConnectToUSBeePod

ConnectToUSBeePod will initialize a single USBee pod. If more than one USBee pod is present, a message box will appear that will allow the user to choose which pod gets initialized.

##### 3.1.1.1 Calling Convention

```
Function ConnectToUSBeePod() As Integer
```

↳ Return Value: Pod number that was initialized if successful or 0 if no pod found

##### 3.1.1.2 Example

```
Dim PodIDFound as Integer  
' Initialize my USBee pod  
PodIDFound = ConnectToUSBeePod  
If PodIDFound = 0 Then MsgBox "No Pod found"
```



### 3.1.2 InitializePod

InitializePod will initialize the USBee pod when the ID number of the USBee pod is known. This routine can be called more than once with different ID numbers to initialize multiple pods. No interaction from the user is needed in the case of multiple pods.

#### 3.1.2.1 Calling Convention

```
Function InitializePod(PodNumber As Integer) As Boolean
```

- ↳ PodNumber is the Pod ID number located on the back of the USBee pod
- ↳ Return Value: True = Success, False = Failure

#### 3.1.2.2 Example

```
` Initialize my USBee pod (ID number 324)  
If InitializePod( 324 ) = False Then MsgBox "Initialization Failed"
```

## 3.2 Setting USBee Output Signals

Each of the 8 USBee pod signals (0 through 7) can be either an Input or an Output. There are 3 routines that set the USBee signals to the desired I/O state as well as drive the outputs to know values. Each routine has unique features to provide optimized performance. These routines are SetSignalStates, SetSignalStatesMultiple, and GenerateData.

These three routines take a parameter called StateValue. The StateValue parameter sets the Input/Output state of each of the signals. Each bit in the byte represents each individual signal. Bit 0 corresponds to Signal 0 and Bit 7 corresponds to Signal 7 on the USBee Pod. A bit value of 0 means the signal is an Input. A bit value of 1 means the signal is an Output. If a USBee pod signal is set to be an Input, the associated signal is not driven.

Once the signals are set to the desired values, the CLK line will pulse to indicate stable data is present. The polarity of the CLK line defaults to Active High (3.3V), but can be changed by the ClockActiveState in the GenerateData routine. The CLK line toggles from inactive to active to inactive. The timing of the CLK and Signals vary depending on the routine called and are shown in the routine sections below.

### 3.2.1 SetSignalStates

The SetSignalStates routine sets the Input/Output state of each of the signals as well as the value driven on the output signals.

#### 3.2.1.1 Calling Convention

```
Sub SetSignalStates(PodNumber As Integer, StateValue As Byte, ByteValue  
As Byte)
```



- PodNumber is the Pod ID number located on the back of the USBee pod
- StateValue is the Input/Output state of each of the 8 USBee signals (0 through 7). A signal is an Input if the corresponding bit is a 0. A signal is an Output if the corresponding bit is a 1.
- ByteValue sets the levels driven on the output signals. When set as an output, a signal is driven high (3.3V) if the corresponding bit is a 1. A signal is driven low (0V) if the corresponding bit is a 0. If a signal is set to be an Input in the StateValue parameter, the associated signal is not driven.

### 3.2.1.2 Example

```
\ Talk to USBee Pod number 123
\ Set all 8 USBee signals to be outputs
\ And set Signal 0 and Signal 4 to 3.3V, all others to 0V
SetSignalStates 123, 255, 17

\ Talk to USBee Pod number 123
\ Set USBee Signal 0 to be Input, all others to be outputs
\ And set Signal 1 and Signal 7 to 3.3V, all others to 0V
SetSignalStates 123, 254, 130
```

### 3.2.1.3 Timing

The following diagram shows the timing of the Signals and CLK lines for a single call to SetSignalStates.

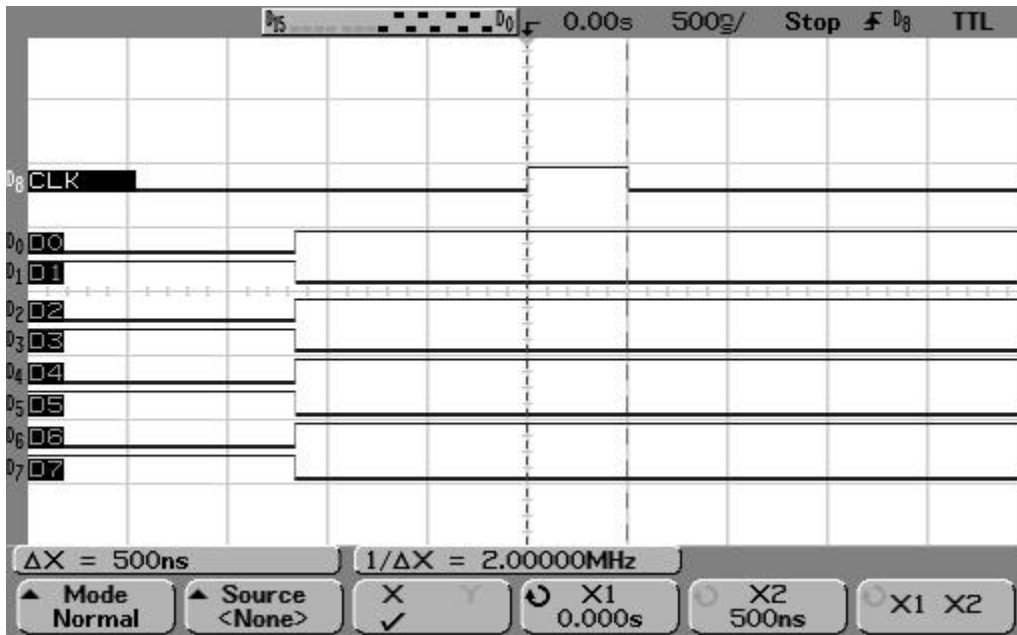


Figure 1. SetSignalStates Timing

```
SetSignalStates 123,&HFF,&H55
```

The following diagram shows the timing between consecutive calls to SetSignalStates. The actual timing between calls varies greatly depending on the PC clock speed, Windows loading,



and number of USB peripherals attached to your computer. The following timing is based on a 400MHz PIII processor running Windows 98 with no other USB devices attached.

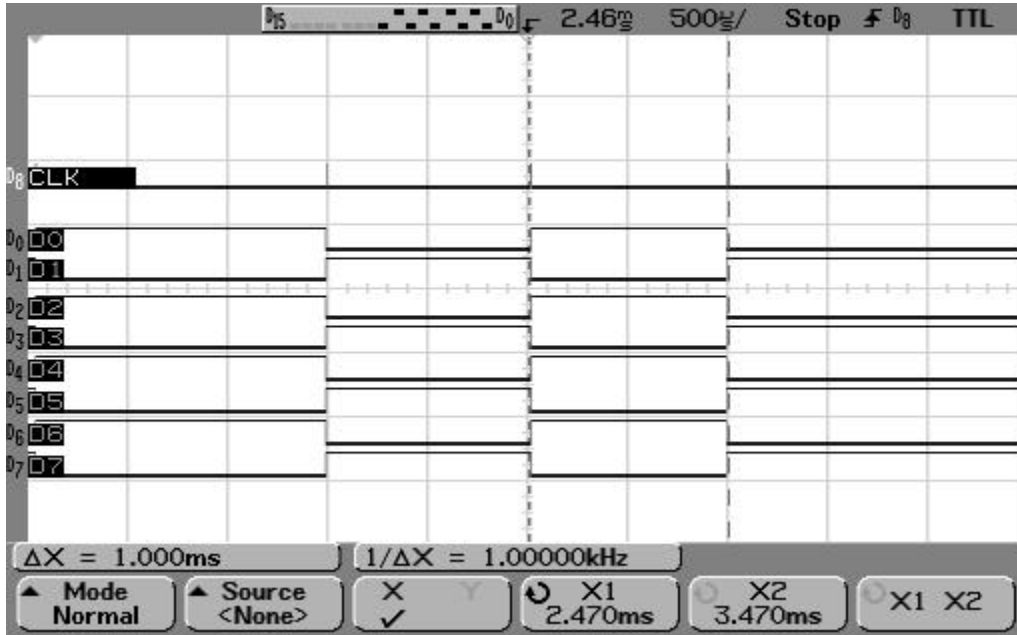


Figure 2. Intercall Timing

```
SetSignalStates 123,&HFF,&HAA
SetSignalStates 123,&HFF,&H55
SetSignalStates 123,&HFF,&HAA
```

### 3.2.1.4 Bandwidth

Using the PC described above, the SetSignalState routine can change the Signals at a rate of 1000 times per second.

## 3.2.2 SetSignalStatesMultiple

The SetSignalStatesMultiple routine sets the Input/Output state of each of the signals and drives the output signals with the series of levels defined by the bytes in an array. Up to 4000 bytes can be written in a single call to this routine.

### 3.2.2.1 Calling Convention

```
Sub SetSignalStatesMultiple(PodNumber As Integer, StateValue As Byte,
Length as Integer, ByteValue() As Byte)
```

- ↖ PodNumber is the Pod ID number located on the back of the USBee pod
- ↖ StateValue is the Input/Output state of each of the 8 USBee signals (0 through 7). A signal is an Input if the corresponding bit is a 0. A signal is an Output if the corresponding bit is a 1.
- ↖ Length is the number of bytes in the array ByteValue() that will be shifted out the USBee pod. The maximum length is 4000.



ByteValue() is the array that holds the series of bytes that represent the levels driven on the output signals. When set as an output, a signal is driven high (3.3V) if the corresponding bit is a 1. A signal is driven low (0V) if the corresponding bit is a 0. If a signal is set to be an Input in the StateValue parameter, the associated signal is not driven. The CLK line toggles as described above for each output byte (Length times).

### 3.2.2.2 Example

```
` Declare the array that will hold the output data
Dim OutData(4000) as Byte

` We want to clock out 1400 consecutive bytes that increment
For X = 0 to 1399
    OutData( X ) = X And 255
Next X

` Talk to USBee Pod number 123
` Set all 8 USBee signals to be outputs
` And send the data in the array one byte at a time while toggling CLK
SetSignalStatesMultiple 123, 255, 1400, OutData
```

### 3.2.2.3 Timing

The following diagram shows the timing of the Signals and CLK lines for a single byte within a call to SetSignalStatesMultiple.

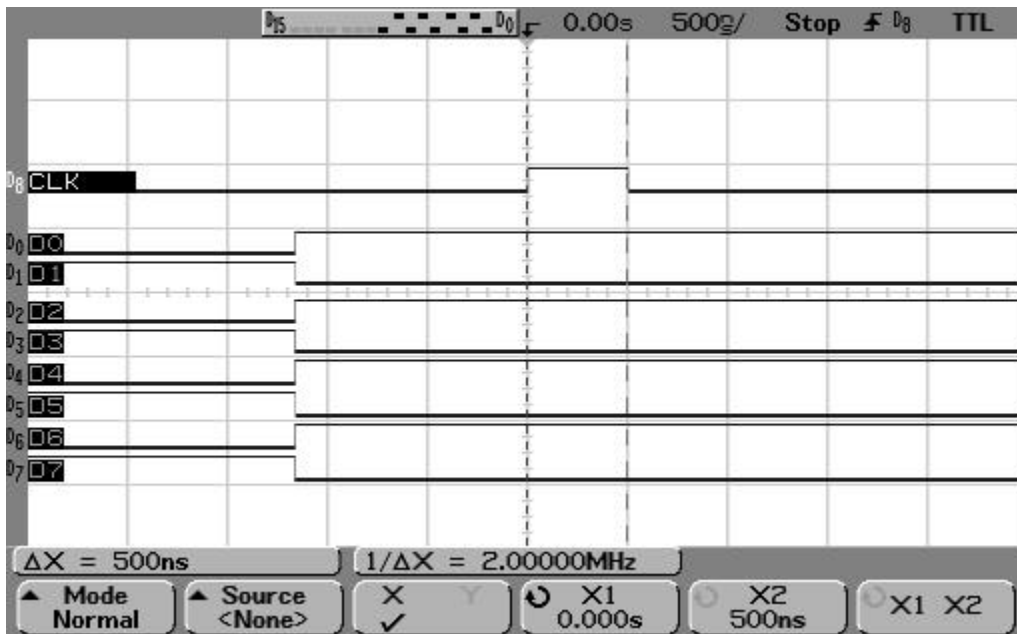
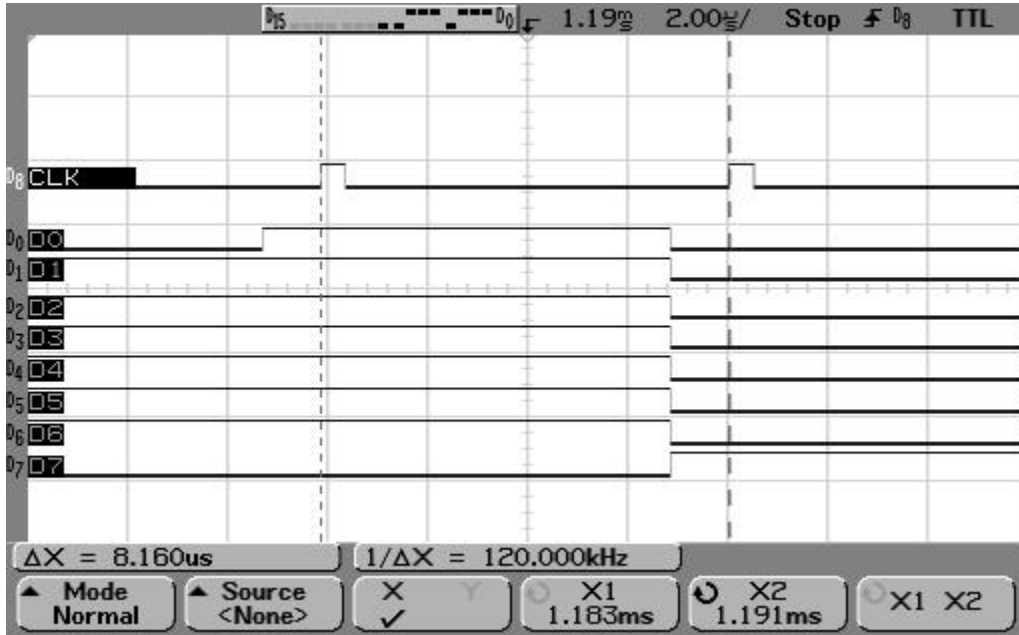


Figure 3. SetSignalStatesMultiple Timing

```
SetSignalStatesMultiple 123,&HFF,1400,OutData
```



The following diagram shows the timing of the Signals and CLK lines for two consecutive bytes within a single call to SetSignalStatesMultiple.



**Figure 4. SetSignalStatesMultiple Intra-byte timing**

`SetSignalStatesMultiple 123,&HFF,1400,OutData`

The following diagram shows the timing between consecutive calls to SetSignalStates. The actual timing between calls varies greatly depending on the PC clock speed, Windows loading, and number of USB peripherals attached to your computer. The following timing is based on a 400MHz PIII processor running Windows 98 with no other USB devices attached.

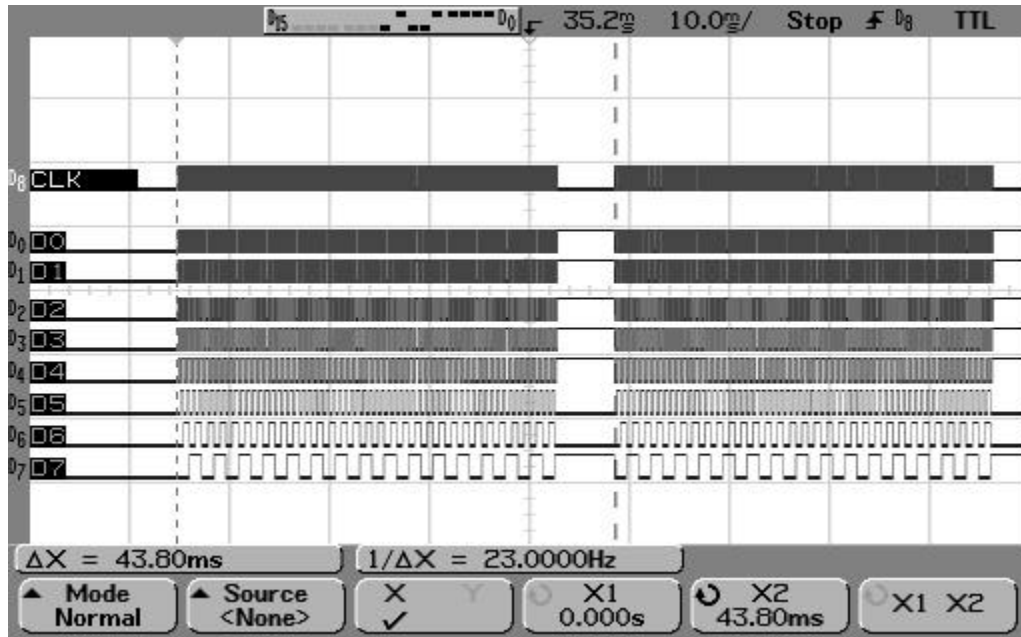


Figure 5. Intercall Timing

```
SetSignalStatesMultiple 123,&HFF,4000,OutData  
SetSignalStatesMultiple 123,&HFF,4000,OutData
```

### 3.2.2.4 Bandwidth

Using the PC described above, the SetSignalStateMultiple routine can change the Signals at a rate of about 91,000 times per second (4000 bytes per 43.8ms).

### 3.2.3 GenerateData

The GenerateData routine sets the Input/Output state of each of the signals and drives the output signals with the series of levels defined by the bytes in an array. It also fills an array with samples of the 8 Signal lines after CLKing out each byte. Up to 60 bytes can be written in a single call to this routine.

With each call to GenerateData, the USBee Pod

- 1) Sets the CLK line to the Inactive State
- 2) Sets the Input/Output mode for each signal
- 3) Starts at the first byte of the array (ByteValue()) and
- 4) Sets the output Signals to the value in the current byte of the array (ByteValue())
- 5) Toggles CLK to the Active state and then back to the Inactive state
- 6) Samples the 8 Signal levels and places the result in the current byte of the input array (InValue())
- 7) Increments to the next bytes in both the ByteValue and InValue arrays
- 8) Repeat from #4 until we have transferred Length bytes



### 3.2.3.1 Calling Convention

Sub GenerateData(PodNumber As Integer, StateValue As Byte, ClockActiveState As Byte, Length As Byte, ByteValue() As Byte, ByRef InValue() As Byte)

- ↖ PodNumber is the Pod ID number located on the back of the USBee pod.
- ↖ StateValue is the Input/Output state of each of the 8 USBee signals (0 through 7). A signal is an Input if the corresponding bit is a 0. A signal is an Output if the corresponding bit is a 1.
- ↖ Length is the number of bytes in the array ByteValue() that will be shifted out the USBee pod. The maximum length is 60.
- ↖ ClockActiveState is the active clock (CLK) state (0 active low (0V), 1 active high (3.3V)). The CLK line goes from inactive to active and back to inactive after each output byte is stable on the USBee Signals.
- ↖ ByteValue() is the array that holds the series of bytes that represent the levels driven on the output signals. When set as an output, a signal is driven high (3.3V) if the corresponding bit is a 1. A signal is driven low (0V) if the corresponding bit is a 0. If a signal is set to be an Input in the StateValue parameter, the associated signal is not driven. The CLK line toggles as described above for each output byte (Length times).
- ↖ InValue() is the array that holds the series of bytes that represent the levels sampled on the USBee signals after each CLK cycle. All Signals are sampled, even those set as outputs. A bit is read as a 1 if the corresponding Signal is a logic high (> 2.4V). A bit is read as a 0 if the corresponding Signal is a logic low (<0.8V).

### 3.2.3.2 Example

```
` Declare the array that will hold the output data
Dim OutData(60) as Byte

` We want to clock out 40 consecutive bytes that increment and
` sample the lines after each byte
For X = 0 to 39
    OutData( X ) = X And 255
Next X

` Talk to USBee Pod number 123
` Set the upper 4 signals (4-7) of the USBee to be outputs,
` the others (0-3) as Inputs
` Set the CLK line to be active low
` And send the data in the array one byte at a time, then toggling
` CLK, then sampling Signals
GenerateData 123, &HF0, 0 ,40, OutData, InData

` We can now read the levels of the Signal's after each byte
` was clocked out by looking at the InData array
X = InData(8)      ` Holds the levels of all 8 signals after the
                  ` OutData(8) byte was clocked out
```

### 3.2.3.3 Timing

The following diagram shows the timing of the Signals and CLK lines for a single cycle within a call to GenerateData. The InValue() sample is taken 500ns after the CLK line is set inactive for all bytes.

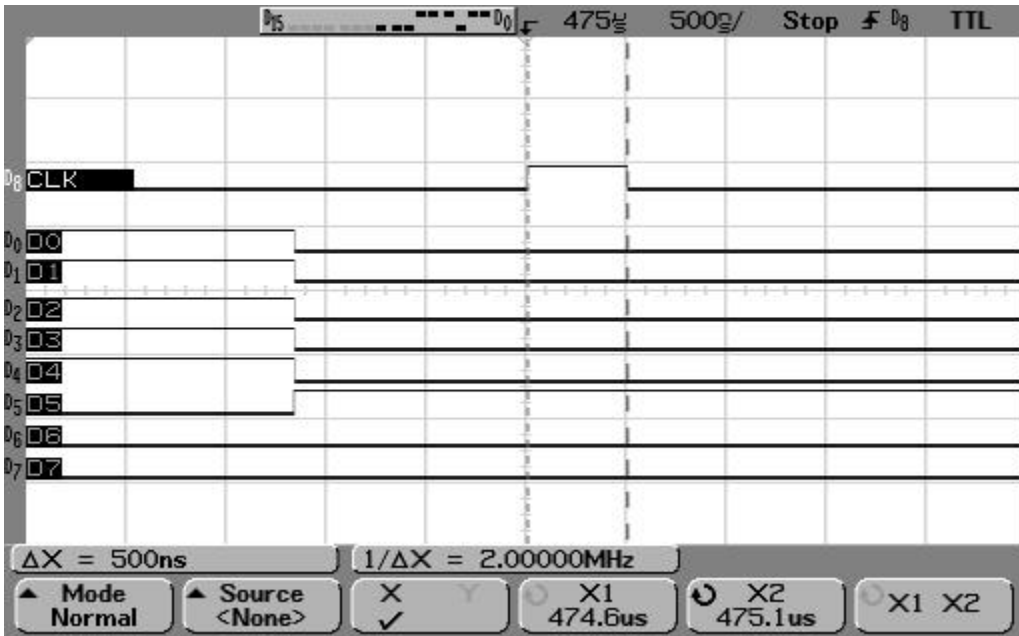


Figure 6. GenerateData Timing (active high CLK)

GenerateData 123, &HFF, 1, 40, OutData, InData

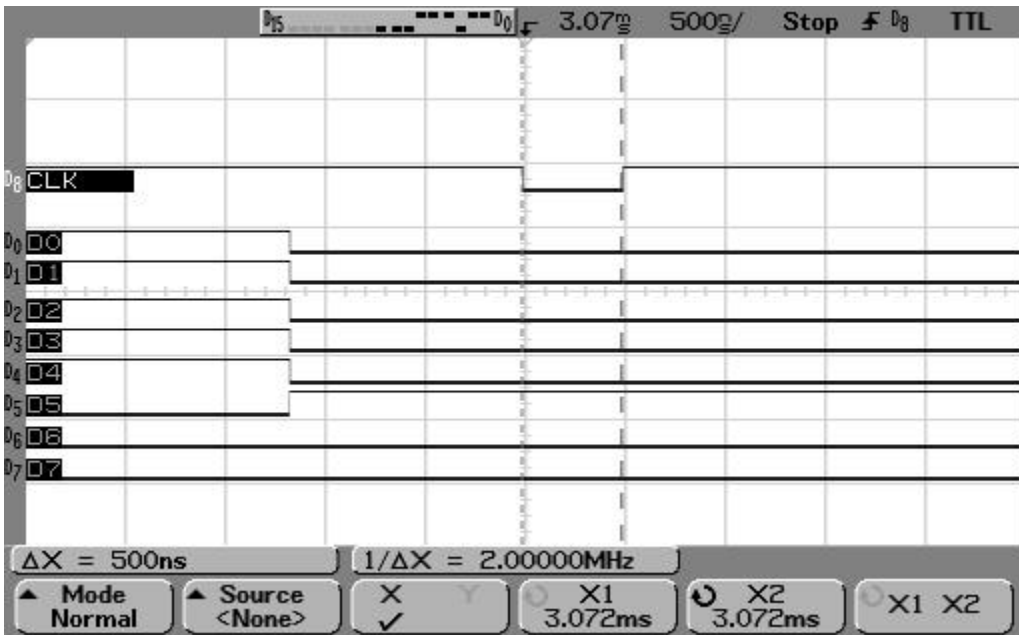


Figure 7. GenerateData Timing (active low CLK)

GenerateData 123, &HFF, 0, 40, OutData, InData

The following diagram shows the timing of the Signals and CLK lines for two consecutive bytes within a single call to GenerateData.

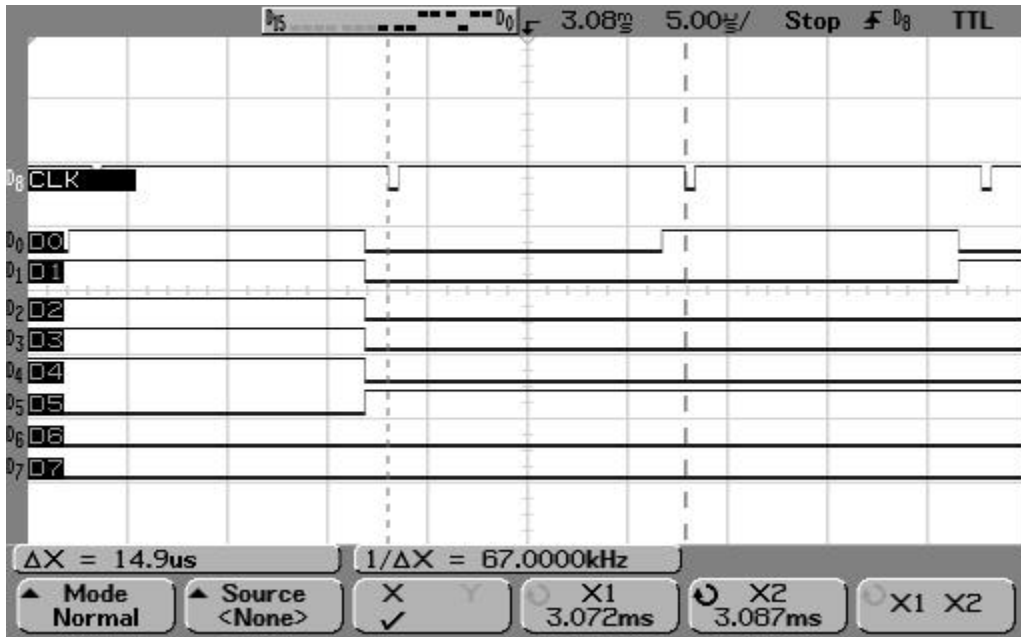


Figure 8. GenerateData Intra-byte Timing

GenerateData 123, &HFF, 0, 40, OutData, InData

The following diagram shows the timing between consecutive calls to GenerateData. The actual timing between calls varies greatly depending on the PC clock speed, Windows loading, and number of USB peripherals attached to your computer. The following timing is based on a 400MHz PIII processor running Windows 98 with no other USB devices attached.

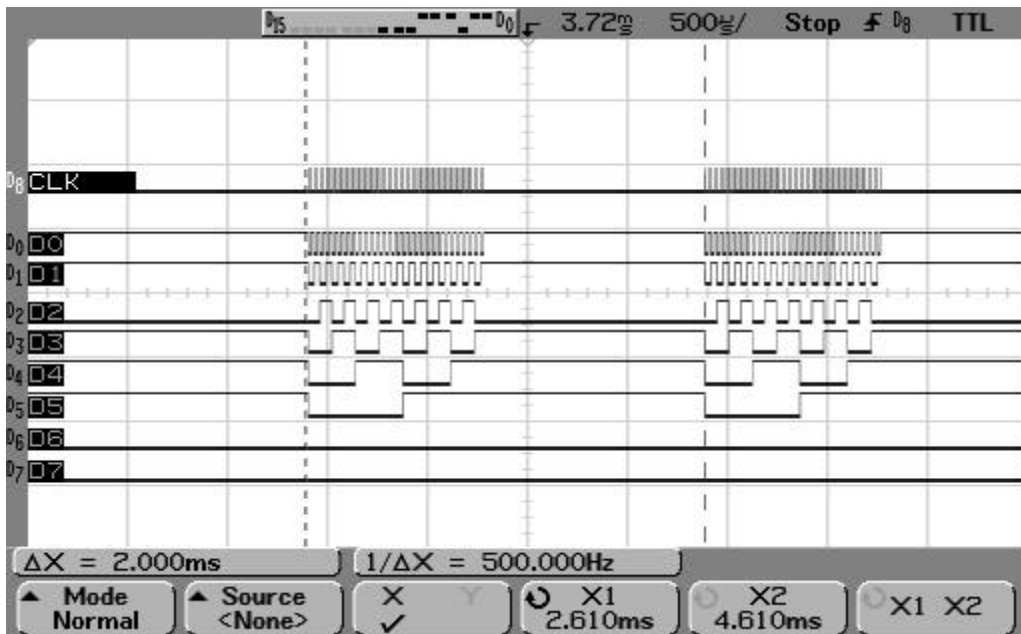


Figure 9. GenerateData Intercall Timing

GenerateData 123, &HFF, 1, 60, OutData, InData  
GenerateData 123, &HFF, 1, 60, OutData, InData



### 3.2.3.4 Bandwidth

Using the PC described above, the GenerateData routine can change the Signal outputs and sample the inputs at a rate of 30,000 times per second (60 bytes per 2ms).

## 3.3 Reading the USBee Input Signals

The USBee Signals can be read by using the GenerateData routine as described in the previous section, or by calling the GetSignalStates routine. Both of these routines read the digital levels on the 8 USBee Signals.

Although each of the 8 USBee pod signals (0 through 7) can be either an Input or an Output, the values read are the true level on the Signal at the time. To set the Input/Output mode of the Signals, call one of the Output routines (GenerateData, SetSignalState, and SetSignalStateMultiple) and specify the desired StateValue.

These routines take a parameter called StateValue. The StateValue parameter sets the Input/Output state of each of the signals. Each bit in the byte represents each individual signal. Bit 0 corresponds to Signal 0 and Bit 7 corresponds to Signal 7 on the USBee Pod. A bit value of 0 means the signal is an Input. A bit value of 1 means the signal is an Output

### 3.3.1 GetSignalStates

The GetSignalStates routine does not toggle the CLK line. If you want to toggle the CLK line on Signal reads, use the GenerateData routine.

#### 3.3.1.1 Calling Convention

```
Function GetSignalStates(PodNumber As Integer) As Byte
```

- ↗ PodNumber is the Pod ID number located on the back of the USBee pod.
- ↗ Return Value is the digital level of all 8 USBee pod Signals (bit 0 is signal 0, bit 7 is signal 7)

#### 3.3.1.2 Example

```
` Talk to USBee Pod number 123  
` Get the current state of all 8 USBee digital signals  
X = GetSignalStates( 123 )
```

#### 3.3.1.3 Bandwidth

The actual timing between calls varies greatly depending on the PC clock speed, Windows loading, and number of USB peripherals attached to your computer. Using a 400MHz PIII processor running Windows 98 with no other USB devices attached can achieve one sample every 1ms. Using this PC configuration, the GetSignalStates routine can sample the inputs at a rate of 1000 times per second.