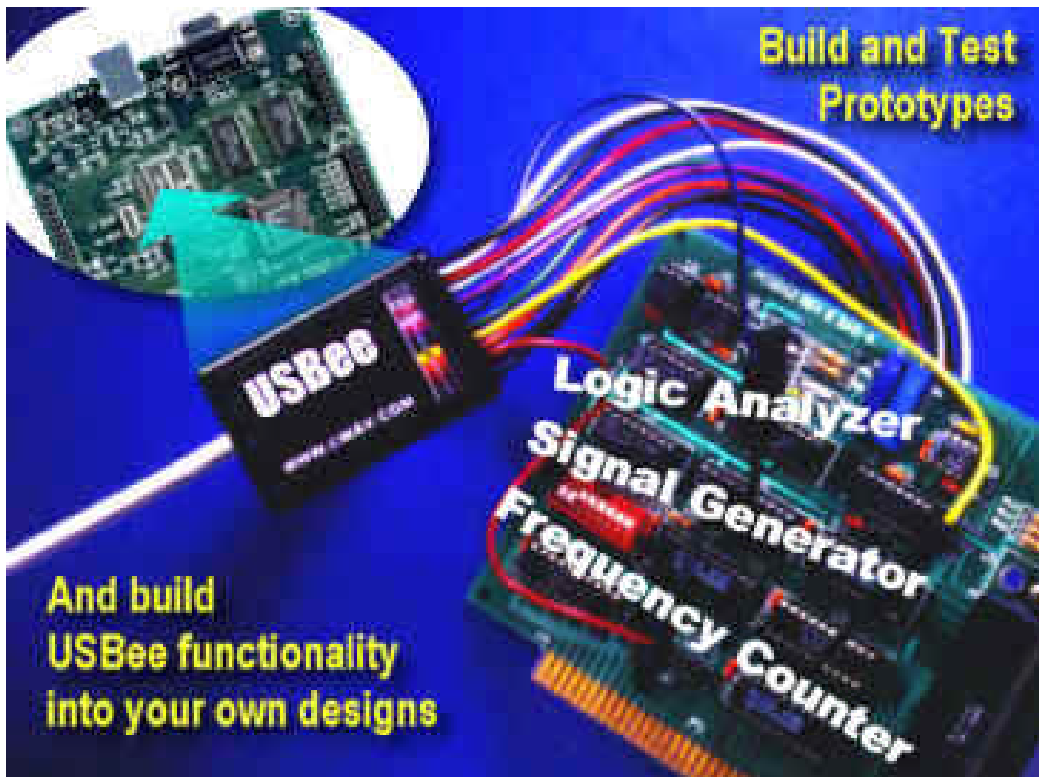

USBee LX Toolbuilder User's Guide

Version 1.4

For the USBee LX Digital Test Pod



CWAV
www.cwav.com
support@cwav.com





Table of Contents

USBEE LX TOOLBUILDER USER'S GUIDE.....1

VERSION 1.4.....1

FOR THE USBEE LX DIGITAL TEST POD.....1

1 OVERVIEW4

1.1 THE USBEE LX SYSTEM.....4

2 THE USBEE LX TOOLBUILDER VISUAL BASIC PROJECT.....4

2.1 USBEE LX TOOLBUILDER PROJECT CONTENTS.....4

2.1.1 *Contents of the USBee LX Toolbuilder Visual Basic Program.....4*

2.2 MAKING YOUR OWN USBEE LX TOOLBUILDER PROJECT.....5

3 INSIDE THE USBEE LX SOURCE CODE5

3.1 INITIALIZING THE POD.....5

3.1.1 *ConnectToUSBeePod.....5*

3.1.1.1 Calling Convention.....5

3.1.1.2 Example6

3.1.2 *InitializePod.....6*

3.1.2.1 Calling Convention.....6

3.1.2.2 Example6

3.2 SETTING USBEE OUTPUT SIGNALS.....6

3.2.1 *SetSignalStates.....7*

3.2.1.1 Calling Convention.....7

3.2.1.2 Example7

3.2.1.3 Timing.....7

3.2.1.4 Bandwidth9

3.2.2 *SetSignalStatesMultiple.....9*

3.2.2.1 Calling Convention.....9

3.2.2.2 Example9

3.2.2.3 Timing.....9

3.2.2.4 Bandwidth11

3.2.3 *GenerateData.....11*

3.2.3.1 Calling Convention.....12

3.2.3.2 Example12

3.2.3.3 Timing.....12

3.2.3.4 Bandwidth15

3.3 READING THE USBEE INPUT SIGNALS.....15

3.3.1 *GetSignalStates.....15*

3.3.1.1 Calling Convention.....15

3.3.1.2 Example15

3.3.1.3 Bandwidth15

3.4 EMBEDDING LOGIC ANALYZER FUNCTION.....16

3.4.1 *InitializeLADLL.....16*

3.4.2 *InitializeLAPod.....16*

3.4.3 *StartCapture.....16*

3.4.4 *CaptureStatus.....17*

3.4.5 *GetCaptureData.....17*

3.4.6 *Logic Analyzer Code Example.....18*

3.5 EMBEDDING SIGNAL GENERATOR FUNCTION18

3.5.1 *InitializeSGDLL.....18*



- 3.5.2 *InitializeSGPod*.....19
- 3.5.3 *StartGenerate*.....19
- 3.5.4 *GenerateStatus*.....20
- 3.5.5 *Signal Generator Example*.....20
- 4 THE USBEE LX TOOLBUILDER VISUAL C PROJECT20**
 - 4.1 USBEE LX TOOLBUILDER PROJECT CONTENTS..... 20
 - 4.1.1 *Contents of the USBee LX Toolbuilder Visual C Program*.....21
 - 4.2 MAKING YOUR OWN USBEE LX TOOLBUILDER C PROJECT 21
- 5 INSIDE THE USBEE LX C SOURCE CODE21**
 - 5.1 INITIALIZING THE POD..... 21
 - 5.1.1 *InitializeLXDLL*.....21
 - 5.1.2 *InitializeLXPod*.....22
 - 5.2 SETTING THE USBEE LX OUTPUT SIGNALS..... 22
 - 5.3 READING THE USBEE LX INPUT SIGNALS..... 22
 - 5.4 EXAMPLE CODE..... 23
 - 5.4.1 *File USBeeLXApp.c*23
 - 5.4.2 *File usbeelx.h*.....24
 - 5.4.3 *Screen Shot when executed*.....24
 - 5.4.4 *Actual Timing of the above sample code*.....25



1 Overview

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

- Logic Analyzer
- Signal/Pattern Generator
- Frequency Counter

The USBee LX Digital Test Pod is also configurable and controllable using your own Visual Basic source code and the USBee LX 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 LX Toolbuilder software to make your own tools.

1.1 The USBee LX System

The USBee LX System consists of the USBee LX Digital Test Pod connected to your Windows® 98, 2000 or XP 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 LX Windows Software or your own USBee LX Toolbuilder software.

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

2 The USBee LX Toolbuilder Visual Basic Project

2.1 USBee LX Toolbuilder Project Contents

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

- 1) USBINF.DLL in the Windows/System directory
- 2) USBEELA.DLL in the Windows/System directory
- 3) USBEESG.DLL in the Windows/System directory
- 4) USBEELX.BIX in the BIX directory under the applications directory where your "USBeetool.exe" is running from
- 5) USBEE.INF in the Windows/INF directory
- 6) USBEE.SYS in the Windows/System32/Drivers directory
- 7) MSVBVM60.DLL in the Windows/System directory
- 8) Lxyz.BIX files in the BIX directory under the applications directory where your "USBeetool.exe" is running from



- 9) SGxyz.BIX files in the BIX directory under the applications directory where your "USBeetool.exe" is running from

2.2 Making your own USBee LX Toolbuilder Project

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

1. First start by installing the USBee LX 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\USBeeLXToolBuilder directory and contents to a new directory (for example \NewTool)
3. Make a subdirectory under your new project directory named \bix (for example \NewTool\bix)
4. Copy the contents of Program Files\USBee\bix to the new bix directory. These files are needed by the Toolbuilder libraries to run.
5. Start Visual Basic and Open the USBee LX Toolbuilder project in the new directory.
6. Make all of your necessary changes using the example tools and this document as a reference.
7. Run and debug your new tool using the Visual Basic interface.

3 Inside the USBee LX 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.

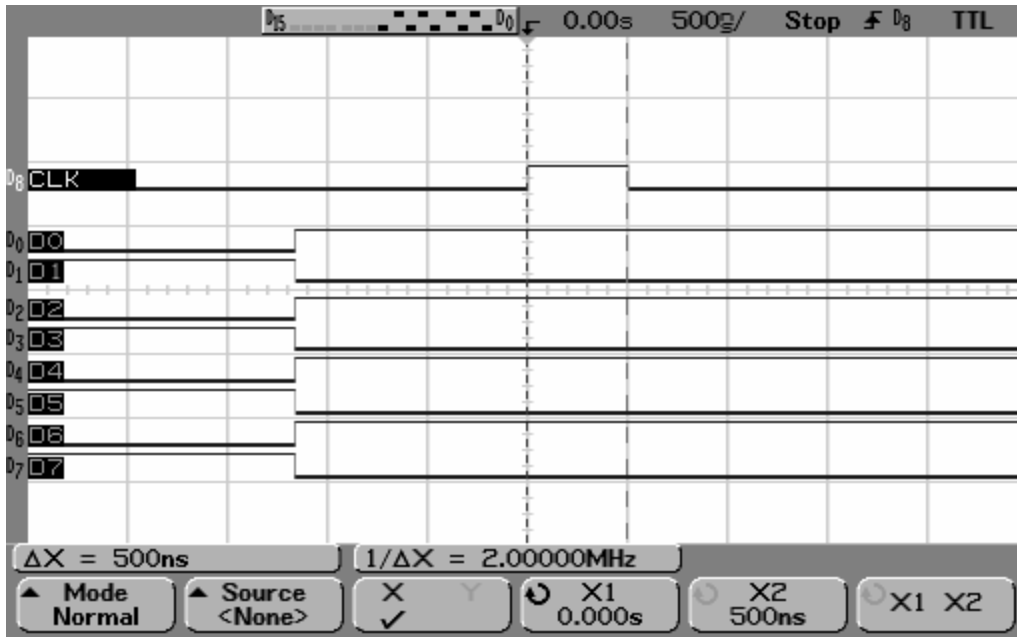


Figure 1. SetSignalStates Timing

```
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.

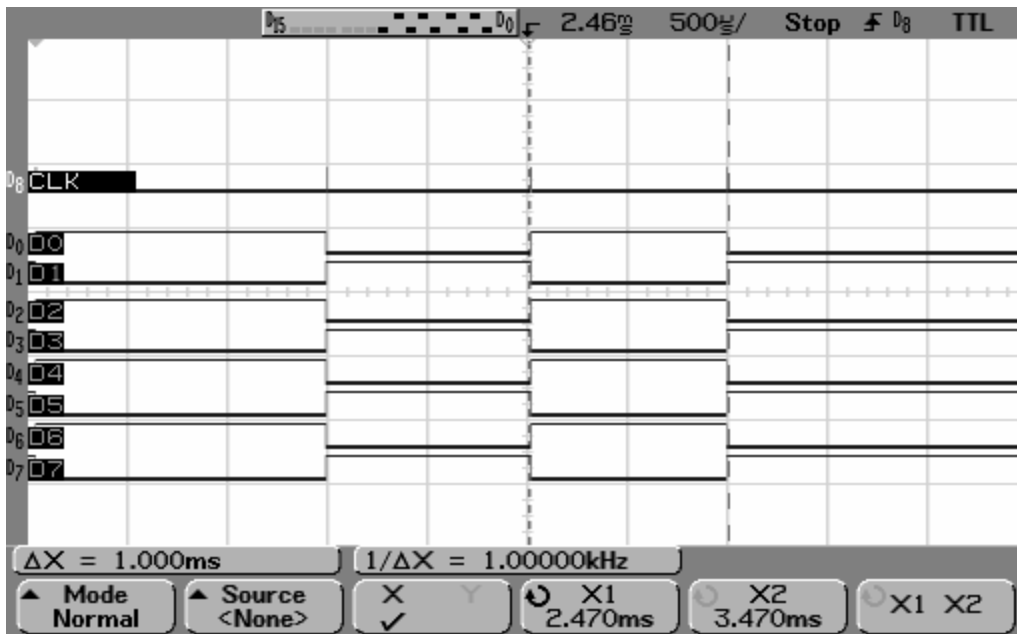


Figure 2. Intercall Timing

```
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.

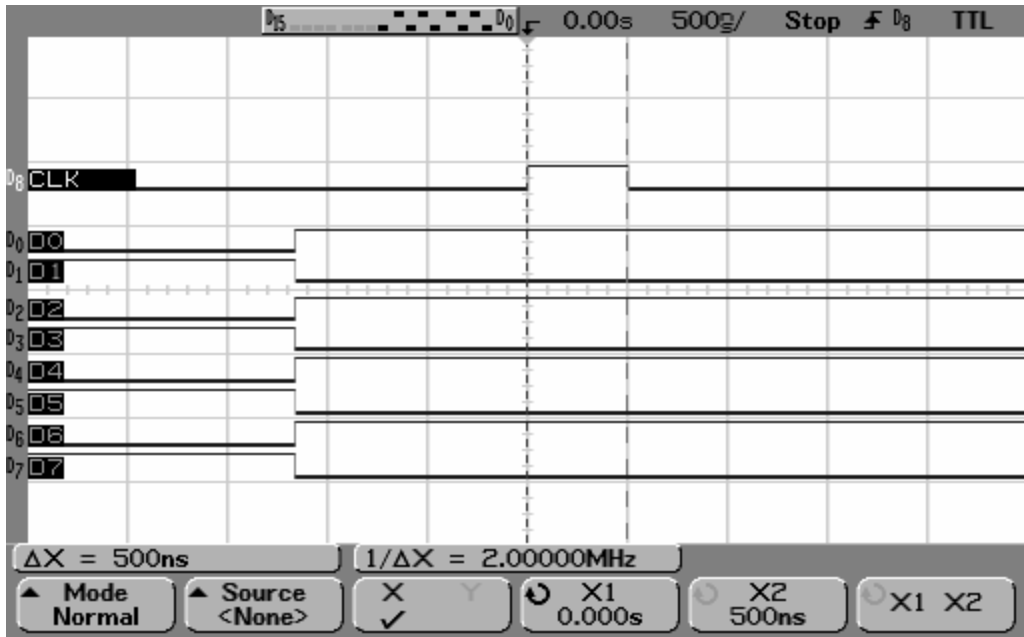


Figure 3. SetSignalStatesMultiple Timing

```
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.

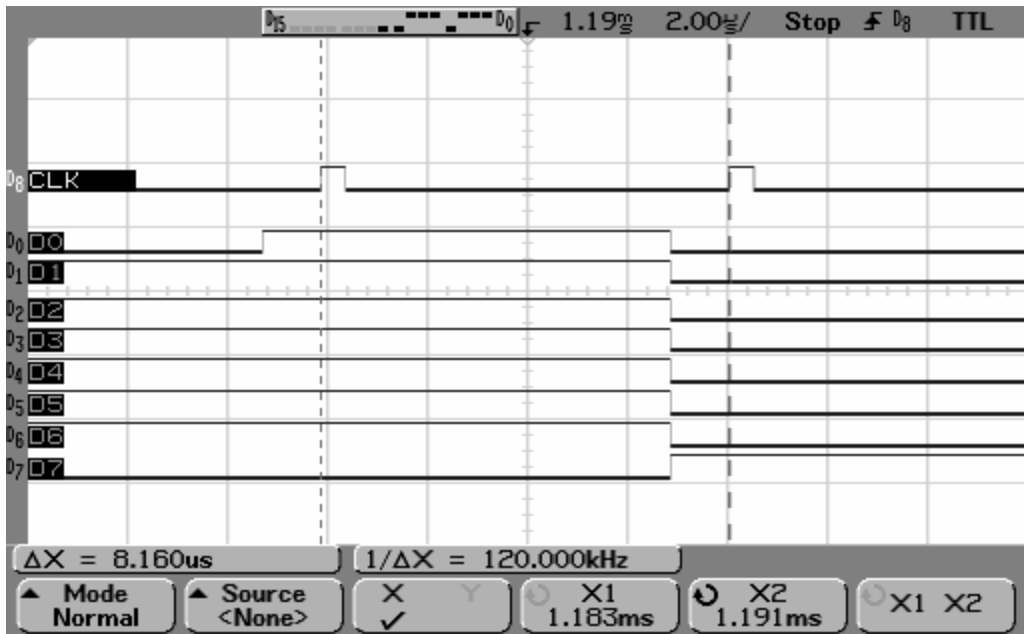


Figure 4. SetSignalStatesMultiple Intra-byte timing

```
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.

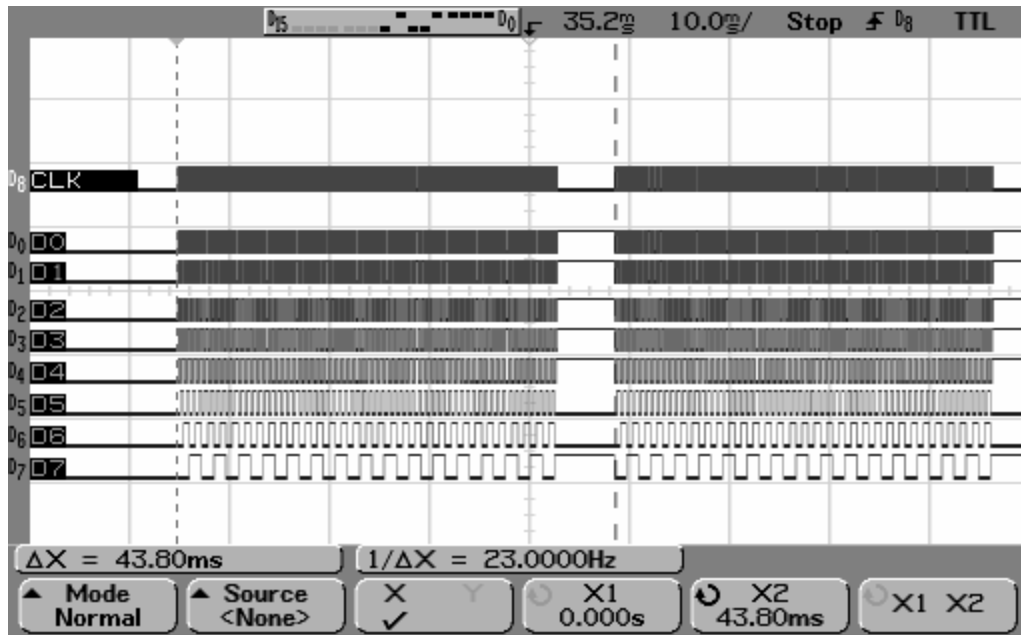


Figure 5. Intercall Timing

```
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.

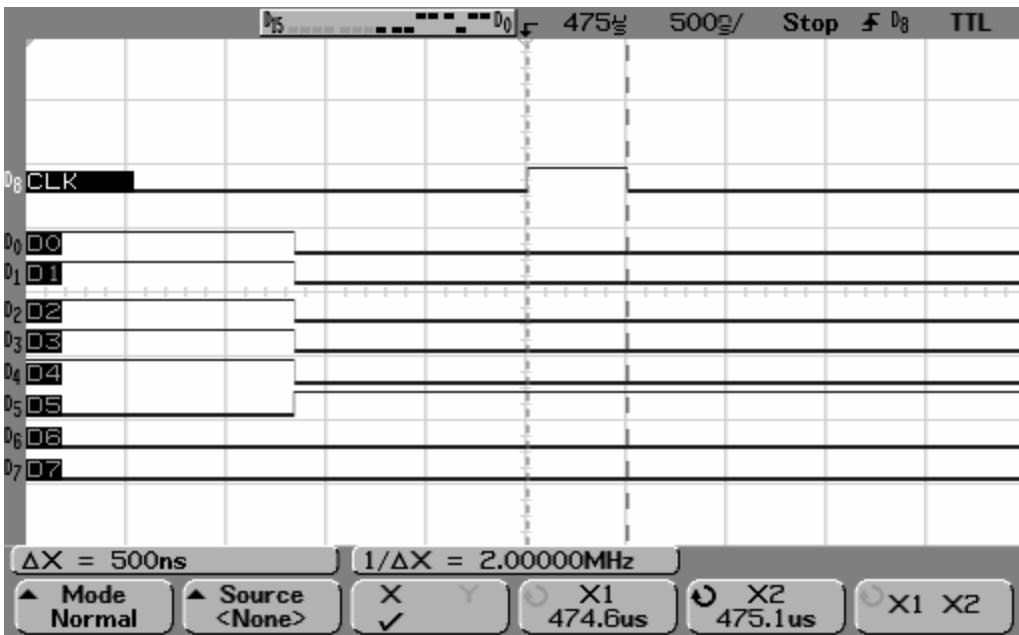


Figure 6. GenerateData Timing (active high CLK)

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

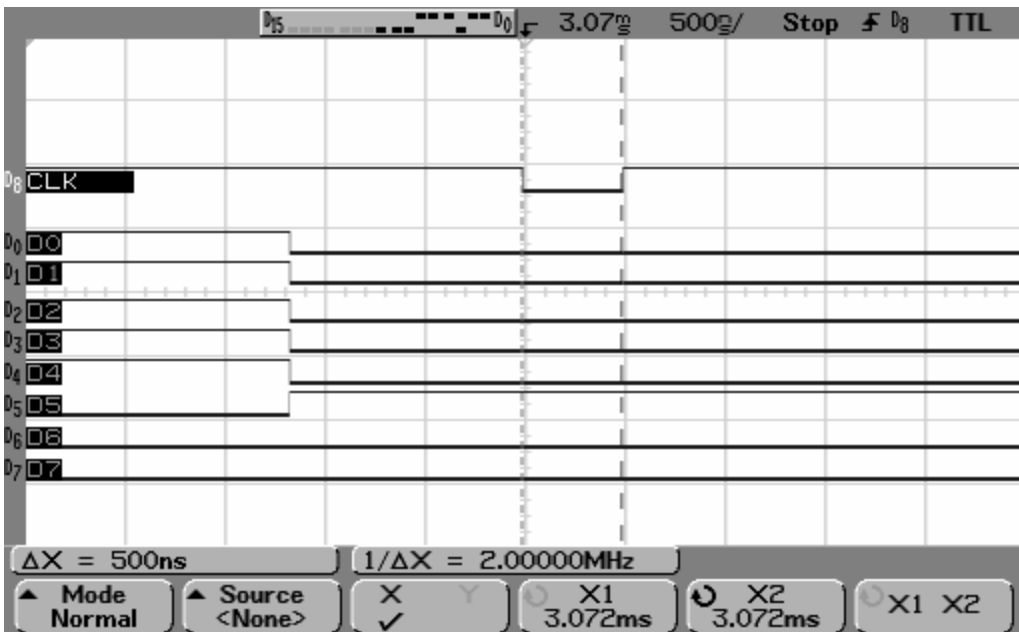


Figure 7. GenerateData Timing (active low CLK)

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.

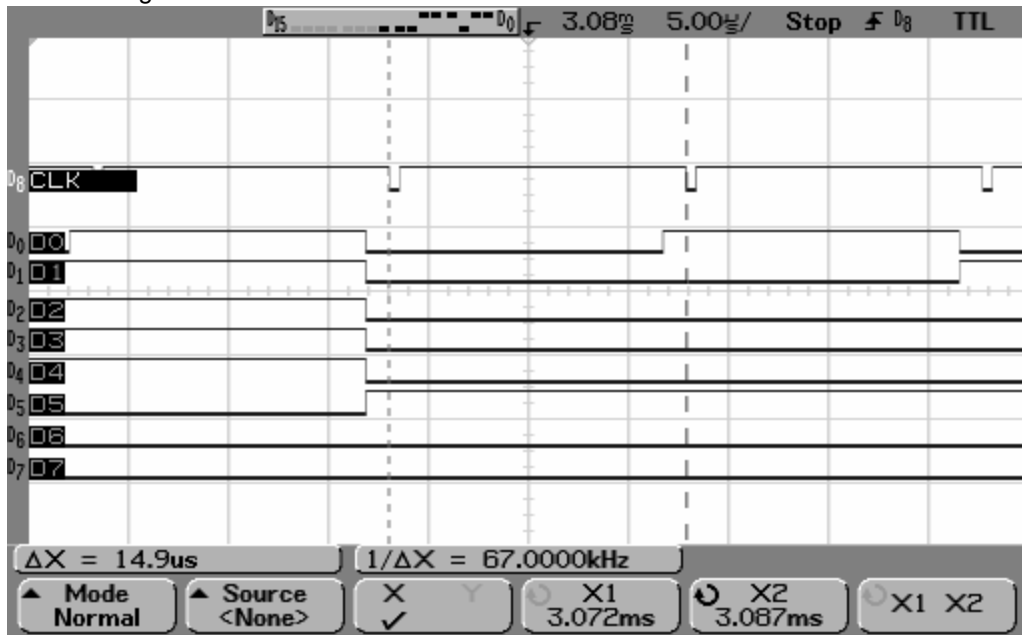


Figure 8. GenerateData Intra-byte Timing

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.

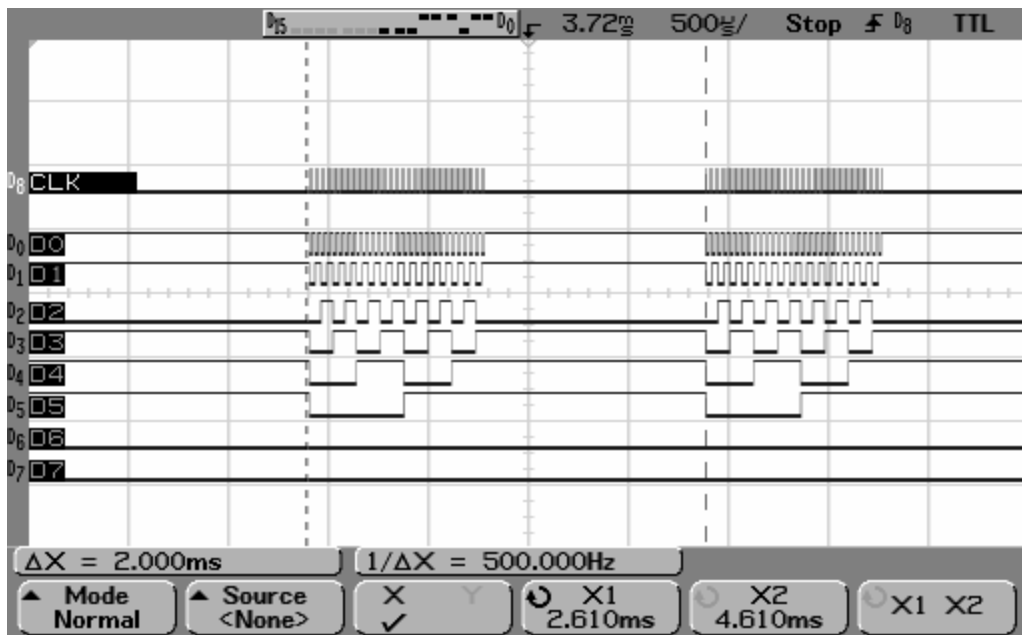


Figure 9. GenerateData Intercall Timing

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.



3.4 Embedding Logic Analyzer Function

The following API describes the routines that control the Logic Analyzer functionality of the USBee LX Digital Test Pod.

3.4.1 InitializeLADLL

This function initializes the DLL and assigns the working directory to the AppPath variable. It must be called once before any of the Logic Analyzer functions are called.

Calling Convention

```
int InitializeLADLL(char *AppPath);
```

where AppPath is a string that contains the path location where the BIX files can be found.

Return Value:
1 - always

3.4.2 InitializeLAPod

This routine initializes the Pod number PodNumber as a Logic Analyzer function.

Calling Convention

```
int InitializeLAPod( unsigned int PodNumber );
```

where PodNumber is the Pod ID of the pod used

Return Value:
0 = Pod Not Found
1 = Pod Initialized

3.4.3 StartCapture

This routine starts the pod capturing data at the specified trigger and sample rates.

Calling Convention

```
int StartCapture (unsigned int PodNumber,  
                 int SampleRate,  
                 char ClockMode,  
                 char TriggerMode);
```

where PodNumber is the Pod ID of the pod used,
and the following table shows the other parameters.



SampleRate:	ClockMode	TriggerMode	Return Value:
Don't care for State Mode: In Timing Mode: 24MHz (index 27) 12MHz (index 17) 6MHz (index 66) 2MHz (index 26) 1MHz (index 16) 500kHz (index 55) 200kHz (index 25) 100kHz (index 15) 50kHz (index 54) 20kHz (index 24) 10kHz (index 14) 5kHz (index 53) 2kHz (index 23) 1kHz (index 13)	0 – External Clocking (State Mode) sample near Rising Edge 1 – External Clocking (State Mode) sample near Falling Edge 2 – Internal Clocking (Timing Mode) sample near Rising Edge 3 – Internal Clocking (Timing Mode) sample near Falling edge	Trigger the capture when the Trigger input signal is: 0 – don't care 1 - rising edge 2 - falling edge 3 - high 4 - low	0 – Capture Start Failed 1 – Capture Start Successful 123 – Pod ID not found 234 – Download Settings Failed 345 – Firmware Download Failed

3.4.4 CaptureStatus

This routine checks the status of the data capture in progress.

Calling Convention

```
int CaptureStatus(unsigned int PodNumber )
```

where PodNumber is the Pod ID of the pod used

Return Value:

- 0 = Pod Not Found
- 1 = Pod Still Capturing Data
- 2 = Pod Finished Capturing Data

3.4.5 GetCaptureData

This routine returns the data that was captured by the Pod. This always returns 6144 bytes.

Calling Convention

```
int GetCaptureData (unsigned int PodNumber,
                   char *DataBuf )
```

where



PodNumber is the Pod ID of the pod used
DataBuf[6144] is where the captured data will be placed

Return Value:
0 = Pod Not Found
8 = Pod Data Collected successfully

3.4.6 Logic Analyzer Code Example

```
Sub ExampleTool6()  
Dim ReturnVal As Long  
Dim Buffer(6144) As Byte  
  
    Debug.Print "USBee LX Logic Analyzer Function"  
  
    ReturnVal = InitializeLADLL(App.Path)      ' Startup the DLL  
    Debug.Print "InitializeLADLL: " & ReturnVal  
  
    ReturnVal = InitializeLAPod(123)         ' Configure the Pod Number 1  
    Debug.Print "InitializeLAPod: " & ReturnVal  
  
    ReturnVal = StartCapture(123, 27, 2, 0)  
    Debug.Print "StartCapture: " & ReturnVal  
  
    ReturnVal = CaptureStatus(123)  
    Debug.Print "CaptureStatus: " & ReturnVal  
  
    ReturnVal = GetCaptureData(123, Buffer(0))  
    ' Send first byte ByRef to pass array  
    Debug.Print "GetCaptureData: " & ReturnVal  
  
    ' Buffer() now holds the captured data (if the status was "done")  
  
End Sub
```

3.5 Embedding Signal Generator Function

The following API describes the routines that control the Signal Generator functionality of the USBee LX Digital Test Pod.

3.5.1 InitializeSGDLL

This function initializes the DLL and assigns the working directory to the AppPath variable. It must be called once before any of the Sig Gen functions are called.

Calling Convention

```
int InitializeSGDLL(char *AppPath);
```

where AppPath is a string that contains the path location where the BIX files can be found.



Return Value:
1 - always

3.5.2 InitializeSGPod

This routine initializes the Pod number PodNumber as a Signal Generator function.

Calling Convention

```
int InitializeSGPod( unsigned int PodNumber );  
where PodNumber is the Pod ID of the pod used
```

Return Value:
0 = Pod Not Found
1 = Pod Initialized

3.5.3 StartGenerate

This routine starts the pod generating data with the specified trigger, sample rates, and data.

Calling Convention

```
int StartGenerate(unsigned int PodNumber,  
char *Buffer,  
int Count,  
int SampleRate,  
char ClockMode,  
char TriggerMode,  
char RepeatFlag);
```

where PodNumber is the Pod ID of the pod used,
and

Count:	SampleRate:	ClockMode:	TriggerMode	RepeatFlag	Return Value:
Number of samples to generate in pattern 1 to 6144	24MHz (index 27)	0 – External Clocking change sample near Rising Edge	Trigger the pattern generation when the Trigger input signal is: 0 – don't care 1 -rising edge 2 –falling edge 3 –high 4 –low	0 - No repeat	0 – Generate Start Failed
	12MHz (index 17)				
	6MHz (index 66)			1 – External Clocking change sample near Falling Edge	1 - Repeat the pattern
	2MHz (index 26)				
	1MHz (index 16)	2 – Internal Clocking change sample near Rising Edge			
	500kHz (index 55)				
	200kHz (index 25)				
	100kHz (index 15)	3 – Internal Clocking change sample near Falling edge			
	50kHz (index 54)				
	20kHz (index 24)				
	10kHz (index 14)				
	5kHz (index 53)				
	2kHz (index 23)				
1kHz (index 13)					



3.5.4 GenerateStatus

This routine checks the status of the data generation in progress.

Calling Convention

```
int GenerateStatus(unsigned int PodNumber )
```

where PodNumber is the Pod ID of the pod used

Return Value:

0 = Pod Not Found

1 = Pod Still Generating Data

2 = Pod Finished Generating Data

3.5.5 Signal Generator Example

```
Sub ExampleTool7()  
Dim ReturnVal As Long  
Dim Buffer(6144) As Byte  
  
Debug.Print "USBee LX Signal Generator Function"  
  
For x = 0 To 6143      ' Make some data to generate  
    Buffer(x) = x And 255  
Next x  
  
ReturnVal = InitializeSGDLL(App.Path)      ' Startup the DLL  
Debug.Print "InitializeSGDLL: " & ReturnVal  
  
ReturnVal = InitializeSGPod(123)      ' Configure the Pod Number 123  
Debug.Print "InitializeSGPod: " & ReturnVal  
  
If ReturnVal <> 1 Then MsgBox "The USBee is not available"  
  
ReturnVal = StartGenerate(123, Buffer(0), 6144, 27, 2, 0, 0)  
Debug.Print "StartGenerate: " & ReturnVal  
  
ReturnVal = GenerateStatus(123)  
Debug.Print "GenerateStatus: " & ReturnVal  
  
End Sub
```

4 The USBee LX Toolbuilder Visual C Project

4.1 USBee LX Toolbuilder Project Contents



4.1.1 Contents of the USBee LX Toolbuilder Visual C Program

USBeeLXApp directory

Usbeelx.h	Header file that declares the DLL procedures
Usbeelx.lib	Library file that calls the DLL
USBeeLXApp.c	Visual C example source code
USBeeLXApp.dsp	Visual C project file
USBeeLXApp.dsw	Visual C project file

USBeeLXApp\Debug directory

USBeeLXApp.exe	Compiled executable of above code
----------------	-----------------------------------

USBeeLXApp\Debug\Bix directory

Usbeelx.bix	Downloaded firmware for the pod to execute the function This directory MUST be underneath the directory that is specified in the call the InitializeLXDLL.
-------------	--

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

1. USBEE.INF in the Windows/INF directory
2. USBEE.SYS in the Windows/System32/Drivers directory
3. MSVBVM60.DLL in the Windows/System directory

4.2 Making your own USBee LX Toolbuilder C Project

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

1. First start by installing the USBee LX 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\USBeeLXApp directory and contents to a new directory (for example \NewTool)
3. Start Visual Studio 6.0 (C++) and Open the USBee LX Toolbuilder project in the new directory.
4. Make all of your necessary changes using the example tools and this document as a reference.
5. Run your new application that was built in the Debug directory..

5 Inside the USBee LX C Source Code

5.1 Initializing the Pod

5.1.1 InitializeLXDLL



This function initializes the DLL and assigns the working directory to the AppPath variable. It must be called once before any of the LX functions are called.

Calling Convention

```
int InitializeLXDLL(char *AppPath);
```

where AppPath is a string that contains the path location where the BIX files can be found.

Return Value:

1 - always

5.1.2 InitializeLXPod

This routine initializes the Pod number PodNumber as a LX function.

Calling Convention

```
int InitializeLXPod( unsigned int PodNumber );
```

where PodNumber is the Pod ID of the pod used

Return Value:

0 = Pod Not Found

1 = Pod Initialized

5.2 Setting the USBee LX Output Signals

Calling Convention

```
int SetSignals (unsigned int PodNumber,  
               unsigned char State, unsigned int length,  
               unsigned char *Bytes)
```

- ↪ PodNumber is the Pod ID number located on the back of the USBee pod
- ↪ State 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 Bytes() that will be shifted out the USBee pod. The maximum length is 4000.
- ↪ Bytes() 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 State parameter, the associated signal is not driven. The CLK line toggles for each output byte (Length times).
- ↪ Return Value:
 - 1 = Successful
 - 0 = Failure

5.3 Reading the USBee LX Input Signals

```
int GetSignals (unsigned int PodNumber)
```

- ↪ 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)



5.4 Example Code

5.4.1 File USBeeLXApp.c

```
// USBeeLXApp.c : Defines the entry point for the console application.
//

#include "stdio.h"
#include "conio.h"
#include "usbeelx.h"

int main(int argc, char* argv[])
{
    unsigned char Data[4];
    int ReturnVal;

    printf("Sample USBee LX Toolbuilder application in C\n");

    printf("Initializing the DLL\n");
    ReturnVal = InitializeLXDDL(".");
    if (ReturnVal != 1) {
        printf("Failure Initializing the DLL\n");
        getch();
        return 0;
    }

    printf("Initializing the Pod\n");
    ReturnVal = InitializeLXPod(123);
    if (ReturnVal != 1) {
        printf("Failure Initializing the Pod\n");
        getch();
        return 0;
    }

    Data[0] = 0x00;
    Data[1] = 0xAA;
    Data[2] = 0x55;
    Data[3] = 0xFF;

    printf("Setting the USBee LX Output Signals\n");
    ReturnVal = SetSignals (123, 0xff, 4, Data);
    if (ReturnVal != 1) {
        printf("Failure Setting the USBee LX\n");
        getch();
        return 0;
    }

    printf("Reading the USBee LX Signal levels\n");
    ReturnVal = GetSignals (123);
    if (ReturnVal == -1) {
        printf("Failure Reading the USBee LX: %d\n", ReturnVal);
        getch();
        return 0;
    }
    printf("The USBee LX signals value: %d\n", ReturnVal);

    printf("Hit any key to continue...\n");

    getch();

    return 0;
}
```



5.4.2 File *usbeelx.h*

```
#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

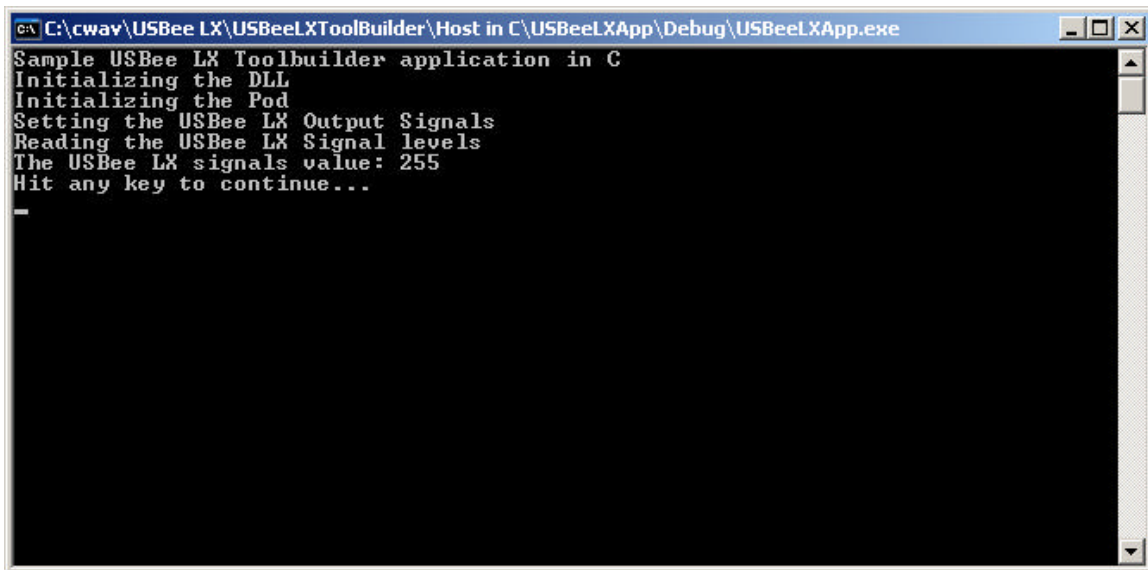
CWAV_IMPORT int CWAV_API InitializeLXDLL(char *buffer);

CWAV_IMPORT int CWAV_API InitializeLXPod(int PodNumber);

CWAV_IMPORT int CWAV_API SetSignals (unsigned int PodNumber, unsigned char State,
unsigned int length, unsigned char *Bytes);

CWAV_IMPORT int CWAV_API GetSignals (unsigned int PodNumber);
```

5.4.3 Screen Shot when executed





5.4.4 Actual Timing of the above sample code

The screen below shows the resulting waveform of the USBee LX signals when the above program is run. As you can see in the trace, the call to SetSignals produces a sample every 4.12us for a rate of 240kps.

