USBee ZX Toolbuilders User's Guide

Version 1.0

For the USBee ZX Digital Test Pod

CWAV
www.cwav.com
support@cwav.com

Table of Contents

1 OVERVIEW 3

2 THE USBEE ZX POD 4

3 INSTALLING THE USBEE ZX TOOLBUILDER 5

3.1 USBEE ZX TOOLBUILDER PROJECT CONTENTS 5

4 INSIDE THE USBEE ZX C SOURCE CODE 5

4.1 BIT BANG-MODES 6
4.1.1 Initializing the Pod 6
4.1.2 Setting the USBee ZX Output Signals 7
4.1.3 Reading the USBee ZX Input Signals 7
4.2 EMBEDDING LOGIC ANALYZER FUNCTION 7
4.2.1 InitializeZXLAPod 7
4.2.2 SpeedTest 8
4.2.3 MakeBuffer 8
4.2.4 DeleteBuffer 8
4.2.5 StartCapture 9
4.2.6 CaptureStatus 9
4.2.7 StopCapture 10
4.2.8 LoggedData 10
4.3 EMBEDDING SIGNAL GENERATOR FUNCTION 10
4.3.1 InitializeZXSGPod 10
4.3.2 SetData 11
4.3.3 StartGenerate 11
4.3.4 GenerateStatus 11
4.4 EXAMPLE CODE 12
4.4.1 File USBeeZXApp.c 12
4.4.2 Performance Analysis of the "Bit-Bang" portion of the above 14

5 INSIDE THE USBEE ZX VISUAL BASIC SOURCE CODE 18

5.1 FUNCTION DEFINITIONS 18
5.2 EXAMPLE CODE 18
5.3 VISUAL BASIC SAMPLE CODE 19
5.3.1 Performance Analysis 21

1 Overview

The USBee ZX Digital Test Pod System consists of the USBee ZX Digital Test Pod connected to a Windows® 98, 2000 or XP PC High Speed USB 2.0 port 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 ZX Windows Software or your own USBee ZX Toolbuilder software.

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

The USBee ZX Digital Test Pod is ideal for students or designer's that need to get up and running with High Speed USB immediately. With a mini-B USB connector on one end and signal pin headers on the other, this simple pod will instantly USB 2.0 High-Speed enable your design. Then using the source code libraries, drivers and DLL's that are included here you can write your own Visual Basic or C PC application to control and monitor the signal pins on the pod.

Because the USBee ZX board runs at 480Mbps and the pod requires over 250mA to operate at the higher speeds, the USBee ZX pod operates at an elevated temperature. This high temperature is normal.

The USBee ZX has headers that are the interface to your circuits. The signals on these headers represent an 8 bit data bus, a Read/Write#/TRG signal and a clock line. Using the libraries and source code provided you can do byte-wide reads and writes to these signals. The USBee ZX acts as the master, driving the Read/Write#/TRG signals and Clock lines to your circuit.

There are four modes of data transfers that you can use depending on your system needs. You can use the Logic Analyzer or Signal Generator capabilities of the pod, or one of two "bit-bang" modes. The first offers complete flexibility of the 8 signal lines, while the other gives you very high transfer rates.

In the first mode, Bi-Directional Mode, each of the 8 data signals can be independently setup as inputs or outputs. When sending data to the pod, only the lines that are specified as outputs will be driven. When reading data from the pod, all 8 signals lines will return the actual value on the signal (whether it is an input or an output)

In the second mode, High-Speed Mode, all of the 8 data signal lines are setup in the same direction (as inputs or outputs) at the same time. When sending data to the pod, all signals become outputs. When reading data from the pod, all signals become inputs.

Also in High Speed mode, you can specify the CLK rate. Available CLK rates are 24MHz, 12MHz, 6MHz, 3MHz, and 1MHz. For slower rates you can use the bi-directional mode

In each of the modes you can specify the polarity of the CLK line. You can set the CLK line to change data on the falling edge and sample on the rising edge, or visa versa.

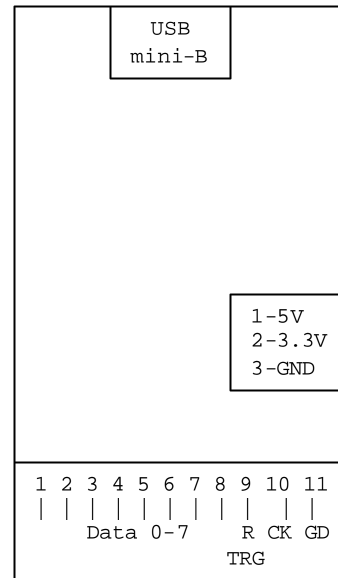
The routines used to read and write the data to the pod are the same for both modes. You call the SetMode function to specify the mode you want to use. All subsequent calls for data transfers will then use that mode of transfer.

The following table shows the possible transfer rates for the various modes. This assumes that your USB 2.0 host controller can achieve these rates. USB 2.0 Host controllers can vary greatly.

Mode	Transfer Type	Burst Rate	Sustained Average Rate

Bi-Directional	Data Write (SetSignals)	300k Bytes/sec	~300k Bytes/sec
Bi-Directional	Data Read (GetSignals)	175k Bytes/sec	~175k Bytes/sec
High-Speed	Data Write (SetSignals)	24M Bytes/sec	~20M Bytes/sec
High-Speed	Data Read (GetSignals)	16M Bytes/sec	~13M Bytes/sec

2 The USBee ZX Pod



11 pin Header Pinout:
Pin 1 Data In/Out Bit 0
Pin 2 Data In/Out Bit 1
Pin 3 Data In/Out Bit 2
Pin 4 Data In/Out Bit 3
Pin 5 Data In/Out Bit 4
Pin 6 Data In/Out Bit 5
Pin 7 Data In/Out Bit 6
Pin 8 Data In/Out Bit 7
Pin 9 Read/Write# Output (bit-bang mode) or TRG (Signal Generator Mode) (R/W#/TRG)
Pin 10 Clock Output (CLK)
Pin 11 Ground

3 pin Header Pinout:
Pin 1 +5V Output
Pin 2 +3.3V Output
Pin 3 Ground

Each of the calls to the USBee ZX interface libraries operate on a byte wide transfer. For each byte that is sent out the signal pins or read into the signal pins, the R/W#/TRG line is set and the

CLK line toggles to indicate the occurrence of a new sample. Each of the bits in the byte transferred maps to the corresponding signal on the ZX pod. For example, if you send out a byte 0x80 to the pod, first the Read/Write# line will be driven low, then the signal on Pin 8 will go high and the others (pin 1-7) will go low. Once the data is on the pins, the Clock line is toggled to indicate that the new data is present.

3 Installing the USBee ZX Toolbuilder

Do not plug in the USBee ZX pod until after you install the software.

The USBee ZX Toolbuilder software is includes as part of the installation for all of the other USBee ZX tools and can be downloaded from www.cwav.com. Run the setup.exe install program in the download. The install program will install the following files and drivers into their correct location on your system.

3.1 USBee ZX Toolbuilder Project Contents

Contents of the USBee ZX Toolbuilder Visual C Program
(contained in the \Program Files\USBee ZX\USBeeZXToolbuilder\HostInC directory after the install).

USBeeZXTB.dsp	Visual C Project File
USBeeZXTB.dsw	Visual C Workspace File
USBeeZXApp.cpp	Visual C program
Usbzxla.lib	USBee ZX Interface library file

Contents of the USBee ZX Toolbuilder Visual Basic Program
(contained in the \Program Files\USBee ZX\USBeeZXToolbuilder\HostInVB directory after the install).

USBeeZXTB.vbp	Visual Basic Project File
USBeeZXTB.vbw	Visual Basic Workspace File
ZX.bas	Visual Basic program including DLL declarations
USBeeZXTB.frm	Visual Basic main form

The USBee ZX Toolbuilder also depends on the following files for proper operation. These files will be installed in the following directories prior to plugging in the USBee ZX pod to USB.

1) USBzxla.DLL in the Windows/System32 directory
2) USBEEZX.INF in the Windows/INF directory
3) USBEEZX.SYS in the Windows/System32/Drivers directory

Once the above files are in the directories, plugging in the USBee ZX pod into a high speed USB port will show a "New Hardware Found" message and the drivers will be loaded.

4 Inside the USBee ZX C Source Code

4.1 Bit Bang-Modes

4.1.1 Initializing the Pod

4.1.1.1 InitializeZXLAPod

This routine initializes the Pod number PodNumber. This routine must be called before calling any other USBee ZX functions.

Calling Convention

```
int InitializeZXLAPod(unsigned int PodNumber);
```

where `PodNumber` is the Pod ID of the pod used

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

4.1.1.2 SetMode

This routine sets the operating mode for the Pod number PodNumber. This routine must be called before calling the SetSignals or GetSignals functions.

Calling Convention

```
int SetMode (int Mode);
```

➢ Mode is the type of transfers that you will be doing and includes a number of bit fields.

- Bit 0 – High Speed or Bi-Directional mode
  - Bit 0 = 0 specifies independent Bi-Directional transfer mode. In this mode, each of the 8 data signals can be independently setup as inputs or outputs. When sending data to the pod, only the lines that are specified as outputs will be driven. When reading data from the pod, all 8 signals lines will return the actual value on the signal (whether it is an input or an output).
  - Bit 0 = 1 specifies high speed all-input or all-output transfer mode. In this mode, all of the 8 data signal lines are setup in the same direction (as inputs or outputs). When sending data to the pod, all signals become outputs. When reading data from the pod, all signals become inputs.
- Bit 1 – CLK mode
  - Bit 1 = 0 specifies that data changes on the Rising edge and data is sampled on the Falling edge of CLK.
  - Bit 1 = 1 specifies that data changes on the Falling edge and data is sampled on the Rising edge of CLK.
- Bits 4,3,2 – High Speed CLK rate (don't care in bi-directional mode)
  - Bits 4,3,2 = 0,0,0 CLK=24MHz
  - Bits 4,3,2 = 0,0,1 CLK=12MHz
  - Bits 4,3,2 = 0,1,0 CLK=6MHz
  - Bits 4,3,2 = 0,1,1 CLK=3MHz
  - Bits 4,3,2 = 1,0,0 CLK=1MHz

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

4.1.2 Setting the USBee ZX Output Signals

Calling Convention

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

- State is not used for High-Speed Mode. In Bi-Directional mode, 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 32767.
- 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. In Bi-Directional mode, if a signal is set to be an Input in the State parameter, the associated signal is not driven. The Read/Write#/TRG line is set low prior to data available, and the CLK line toggles for each output byte (Length times).
- Return Value:
  1 = Successful
  0 = Failure

4.1.3 Reading the USBee ZX Input Signals

Calling Convention

```
int GetSignals   (unsigned char State,
                  unsigned int length,
                  unsigned char *Bytes)
```

- State is not used for High-Speed Mode. In Bi-Directional mode, 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 read from the USBee pod. The maximum length is 32767.
- Bytes() is the array that will hold the series of bytes that represent the levels read on the input signals. The Read/Write# line is set high prior to data available, and the CLK line toggles for each input byte (Length times).
- Return Value is the digital level of all 8 USBee pod Signals (bit 0 is signal 0, bit 7 is signal 7)

4.2 Embedding Logic Analyzer Function

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

4.2.1 InitializeZXLAPod

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

Calling Convention

```
int InitializeZXLAPod( unsigned int PodNumber );
```

where `PodNumber` is the Pod ID of the pod used

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

4.2.2 SpeedTest

This routine tests the connection to the initialized ZX pod to determine the maximum sample rate for the current PC configuration.

Calling Convention

```
int SpeedTest( void )
```

Return Value:
247 = 24MHz
167 = 16MHz
127 = 12MHz
87 = 8MHz
67 = 6MHz
47 = 4MHz
37 = 3MHz
27 = 2MHz
17 = 1MHz

4.2.3 MakeBuffer

This routine creates the sample buffer that will be used to store the acquired samples.

Calling Convention

```
unsigned char *MakeBuffer( unsigned long Size )
```

where `Size` is the number of samples (Bytes) to allocate.

Return Value:
0 = Failed to allocate the buffer
other = pointer to allocated buffer

4.2.4 DeleteBuffer

This routine releases the sample buffer that was used to store the acquired samples.

Calling Convention

```
unsigned char *DeleteBuffer(unsigned char *buffer)
```

where `buffer` is the pointer to the allocated buffer.

Return Value:
0 = Failed to deallocate the buffer
other = Success

4.2.5 StartCapture

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

Calling Convention

```
int StartCapture(   unsigned int SampleRate,
                    unsigned int ClockMode,
                    unsigned char *Triggers,
                    signed int TriggerNumber,
                    unsigned char *buffer,
                    unsigned long length,
                    unsigned long poststore);
```

- SampleRate is as follows:
  - 247 = 24MHz
  - 167 = 16MHz
  - 127 = 12MHz
  - 87 = 8MHz
  - 67 = 6MHz
  - 47 = 4MHz
  - 37 = 3MHz
  - 27 = 2MHz
  - 17 = 1MHz
- ClockMode: 2 = Internal Clocking Mode (olnly available option at this time)
- Triggers: array of Mask/Value byte pairs. Mask is a bit mask that indicates which bit signals to observe. 1 in a bit position means to observe that signal, 0 means to ignore it. Value is the actual value of the bits to compare against. If a bit is not used in the Mask, make sure that the correcsponding bit is a 0 in Value.
- TriggerNumber: the number of pairs of Mask/Value in the above Triggers Array.
- Buffer: pointer to the sample buffer to store the acquired data into. This buffer must be created using the MakeBuffer routine.
- Length: The total number of samples to acquire. This value must be a multiple of 65536.
- Poststore: The total number of bytes to store after the trigger event happens. If the trigger happens early, the samples are stored until the buffer is full.

Return Value:
0 = Failed
1 = Success

4.2.6 CaptureStatus

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

Calling Convention

```
int CaptureStatus(      char *breaks,
                        char *running,
                        char *triggered,
                        long *start,
                        long *end,
                        long *trigger,
                        char *full )
```

- Break: The number of breaks that have occurred in the data sampling since the start of the acquisition. This value is zero (0) if the acquisition has been continuous. If the value is 1 or greater, there was a break in the capture for some reason. If breaks occur repeatedly, your

PC is not capable of the sample rate you've chosen and a lower sample rate is needed to achieve continuous sampling.
- Running: 1 = Acquisition is still running, 0 = Acquisition has completed
- Triggered: 1 = Trigger has occurred, 0 = still waiting for the trigger
- Start: Sample Number of the start of the buffer. 0 unless there is an error.
- End: The sample number of the last sample.
- Trigger: The sample number at the point of trigger.
- Full: The percentage of the buffer that is currently filled. Ranges from 0 to 100.

Return Value:
Number of breaks in the sampling

4.2.7 StopCapture

This routine terminates a pending capture.

Calling Convention

```
int StopCapture(void)
```

Return Value:
1 = Capture Stopped
0 = Stop Failed

4.2.8 LoggedData

This routine returns the value of a particular sample.

Calling Convention

```
long LoggedData( unsigned long index )
```

- Index: sample number to return

Return Value:
Value of the given sample

4.3 Embedding Signal Generator Function

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

4.3.1 InitializeZXSGPod

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

Calling Convention

```
int InitializeZXSGPod(unsigned int PodNumber);
```

where `PodNumber` is the Pod ID of the pod used

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

4.3.2 SetData

This routine sets the value of a given 8-bit sample to the value specified..

Calling Convention

```
long SetData(     unsigned long index,
                  unsigned char value);
```

- Index: sample number to change
- Value: byte value to store in that sample

Return Value:
0 = Set failed
1 = Set successful

4.3.3 StartGenerate

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

Calling Convention

```
int StartGenerate(  unsigned int SampleRate,
                    unsigned char triggermode,
                    unsigned char *buffer,
                    unsigned long length);
```

- SampleRate is as follows:
  - 247 = 24MHz
  - 167 = 16MHz
  - 127 = 12MHz
  - 87 = 8MHz
  - 67 = 6MHz
  - 47 = 4MHz
  - 37 = 3MHz
  - 27 = 2MHz
  - 17 = 1MHz

- TriggerMode: Indicates the value on the external TRG signal that must occur before the waveforms are generated. 0 = Don't Care, 1 = rising edge, 2 = falling edge, 3 = high level, 4 = low level
- Buffer: pointer to the sample that holds the data to generate. This buffer must be created using the MakeBuffer routine.
- Length: The total number of samples to generate. This value must be a multiple of 65536.

Return Value:
0 = Failed
1 = Success

4.3.4 GenerateStatus

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

Calling Convention

```
int GenerateStatus(     char *breaks,
                        char *running,
```

```
char *triggered,
char *complete );
```

- Breaks: The number of breaks that have occurred in the data generating since the start of the generation. This value is zero (0) if the sample timing has been continuous. If the value is 1 or greater, there was a break in the generation for some reason. If breaks occur repeatedly, your PC is not capable of the sample rate you've chosen and a lower sample rate is needed to achieve continuous sample timing.
- Running: 1 = Generation is still running, 0 = Generation has completed
- Triggered: 1 = Trigger has occurred, 0 = still waiting for the trigger
- Complete: The percentage of the buffer that has been generated. Ranges from 0 to 100.

Return Value:
0 = Status Failed
1 = Status Sucessful

4.4 Example Code

4.4.1 File USBeeZXApp.c

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

#include "stdio.h"
#include "conio.h"
#include "windows.h"


#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

// ZX DLL Routine Declarations

// Basic I/O Routines
CWAV_IMPORT int CWAV_API SetSignals (unsigned char State, unsigned int length, unsigned char *Bytes);
CWAV_IMPORT int CWAV_API GetSignals (unsigned char State, unsigned int length, unsigned char *Bytes);
CWAV_IMPORT int CWAV_API SetMode (int Mode);

// SetMode definitions
#define FAST_ONEWAY_DATA                1
#define SLOW_TWOWAY_DATA                0

#define DATA_CHANGES_ON_RISING_EDGE                         2
#define DATA_CHANGES_ON_FALLING_EDGE              0
#define DATA_IS_SAMPLED_ON_RISING_EDGE            0
#define DATA_IS_SAMPLED_ON_FALLING_EDGE           2

#define _24MHz               (0 << 2)
#define _12MHz               (1 << 2)
#define _6MHz                (2 << 2)
#define _3MHz                (3 << 2)
#define _1MHz                (4 << 2)

// Buffer Routines
CWAV_IMPORT unsigned char * CWAV_API MakeBuffer( unsigned long Size );
CWAV_IMPORT int CWAV_API DeleteBuffer( unsigned char *buffer );
CWAV_IMPORT long CWAV_API SetData( unsigned long index, unsigned char value);

// Logic Analyzer Declarations
CWAV_IMPORT int CWAV_API InitializeZXLAPod(unsigned int PodNumber);
CWAV_IMPORT int CWAV_API StartCapture(unsigned int SampleRate, unsigned int ClockMode, unsigned char *Triggers, signed
int TriggerNumber, unsigned char *buffer, unsigned long length, unsigned long poststore);
CWAV_IMPORT int CWAV_API StopCapture(void);
CWAV_IMPORT int CWAV_API CaptureStatus( char *breaks, char *running, char *triggered, long *start, long *end, long
*trigger, char *full );
CWAV_IMPORT int CWAV_API SpeedTest( void );


// Signal Generator Declarations
CWAV_IMPORT int CWAV_API InitializeZXSGPod(unsigned int PodNumber);
CWAV_IMPORT int CWAV_API GenerateStatus( char *breaks, char *running, char *triggered, char *complete );
CWAV_IMPORT int CWAV_API StopGenerate( void );
CWAV_IMPORT int CWAV_API StartGenerate(unsigned int SampleRate, unsigned char triggermode, unsigned char *buffer,
unsigned long length);
// StartGenerate External Trigger Settings
```

```
#define DONT_CARE_TRIGGER 0
#define RISING_EDGE_TRIGGER 1
#define FALLING_EDGE_TRIGGER 2
#define HIGH_LEVEL_TRIGGER 3
#define LOW_LEVEL_TRIGGER 4

CWAV_IMPORT long CWAV_API SetData( unsigned long index, unsigned char value);


int main(int argc, char* argv[])
{
          unsigned char DataInBuffer[65536];
          unsigned char DataOutBuffer[65536];
          int ReturnVal;
          unsigned long x;
          unsigned char ch;
          unsigned int PodNumber;

          PodNumber = 123;              // *******  Change this to your pod number **********

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

          //***********************************
          // Pod Initializations Functions - must call InitializeZXLAPod before using any functions
          //***********************************

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


          //***********************************
          // Basic I/O Functions
          //***********************************


          printf("Setting the Mode to fast mode\n");
          ReturnVal = SetMode(FAST_ONEWAY_DATA | DATA_CHANGES_ON_RISING_EDGE | _24MHz );
          if (ReturnVal != 1) {
                    printf("Failure setting the mode\n");
                    getch();
                    return 0;
          }

          // Make some data to send out the pod signals
          for(x=0;x<65536;x++) DataOutBuffer[x]= (char)x;

          printf("Sending 163,835 bytes out the pod\n");
          for (x = 0; x < 5; x++)
          {
                    DataOutBuffer[0]=ch++;
                    SetSignals (0xFF /* Don't Care */, 32767, DataOutBuffer);
          }

          printf("Reading 163,835 bytes from the pod signals\n");
          for (x = 0; x < 5; x++)
          {
                    GetSignals (0x00 /* Don't Care */, 32767, DataInBuffer);
          }

          printf("Setting the Mode to bi-directional mode\n");
          ReturnVal = SetMode(SLOW_TWOWAY_DATA | DATA_IS_SAMPLED_ON_RISING_EDGE );
          if (ReturnVal != 1) {
                    printf("Failure setting the mode\n");
                    getch();
                    return 0;
          }

          printf("Sending 32767 bytes out the pod\n");

          DataOutBuffer[0]=ch++;
          SetSignals (0xFF, 32767, DataOutBuffer);

          printf("Reading 32767 bytes from the pod signals\n");

          GetSignals (0x00, 32767, DataInBuffer);


          //***********************************
          // Logic Analyzer Functions
          //***********************************

          printf("\nSample USBee ZX Logic Analyzer Toolbuilder application in C\n");

          printf("Start Capturing Data from Pod\n");

          unsigned char Rate = 247;                                                       // Sample Rate = 24Msps
          unsigned char ClockMode = 2;                                                    // Internal Timing
          unsigned char Triggers[4];
          Triggers[0] = 0;                                                                // Trigger Mask = Don't Care
          Triggers[1] = 0;                                                                // Trigger Value
```

```
        char NumberOfTriggers = 1;
        long SampleBufferLength = 16 * 65536;                              // 1Meg Sample Buffer
        unsigned char *SampleBuffer = MakeBuffer(SampleBufferLength);
        long PostStore = SampleBufferLength;

        ReturnVal = StartCapture(Rate, ClockMode, Triggers, NumberOfTriggers, SampleBuffer, SampleBufferLength,
                              PostStore);

        if (ReturnVal != 1) {
                  printf("Failure Starting Capture\n");
                  getch();
                  return 0;
        }

        printf("Waiting for data to be captured...");

        char Breaks;
        char Running;
        char Triggered;
        long Start;
        long End;
        long Trigger;
        char Full;

        do {

                  Sleep(500);         // This is required to put pauses between the status requests, otherwise the
                                      // CaptureStatus  will eat into the USB bandwidth.

                  ReturnVal = CaptureStatus(&Breaks, &Running, &Triggered, &Start, &End, &Trigger, &Full);
                  printf(".");
                  if (Running && (Breaks != 0)) {
                            printf("LA Sample Rate too high\n");
                            break;
                  }

        } while (Running && (Breaks == 0));
        printf("\n");

        StopCapture();

        //***********************************
        // Signal Generator Functions
        //***********************************

        printf("Sample USBee ZX Signal Generator Application in C\n");

        // Make some data
        for (long y = 0; y < SampleBufferLength; y++)
                  SampleBuffer[y] = y & 255;

        ReturnVal = StartGenerate (247, DONT_CARE_TRIGGER, SampleBuffer, SampleBufferLength);

        printf("Completed:  000");

        Running = 1;

        while (Running)
        {
                  GenerateStatus( &Breaks, &Running, &Triggered, &Full );
                  Sleep(400);

                  printf("\b\b\b\b\b%3d %1d", Full, Triggered);

                  if (Breaks) break;
        }
        printf("Breaks= %d\n", Breaks);
        printf("Running= %d\n", Running);
        printf("Triggered= %d\n", Triggered);
        printf("Complete= %d\n", Full);

        printf("Stopped\n");
        StopGenerate();

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

        getch();

        return 0;
}
```

4.4.2 Performance Analysis of the "Bit-Bang" portion of the above

The following logic analyzer capture shows the timing of the execution of the first part of the above example (The SetSignals and Get Signals section). The Clock line (CLK) is the strobe for each of the bytes transferred and the Data line (DATA) represents the data on each of the 8 pod signal lines. The R/W# indicates if it is a read or a write.

As you can see, the entire program takes about 325msec to execute. In this time we perform:

- Initializing the Pod
- Setting the Mode to High Speed mode
- Sending 163,835 bytes out the pod using High Speed mode
- Reading 163,835 bytes from the pod signals using High-Speed mode
- Setting the Mode to bi-directional mode
- Sending 32767 bytes out the pod using bi-directional mode
- Reading 32767 bytes from the pod signals using bi-directional mode

As a comparison between the modes, all transfers in high speed mode (all 327,670 bytes) occur before the first cursor on the logic analyzer trace below. The Bi-Directional write (32767 bytes) occur between the cursors, and the bi-direction reads occur after the second cursor.

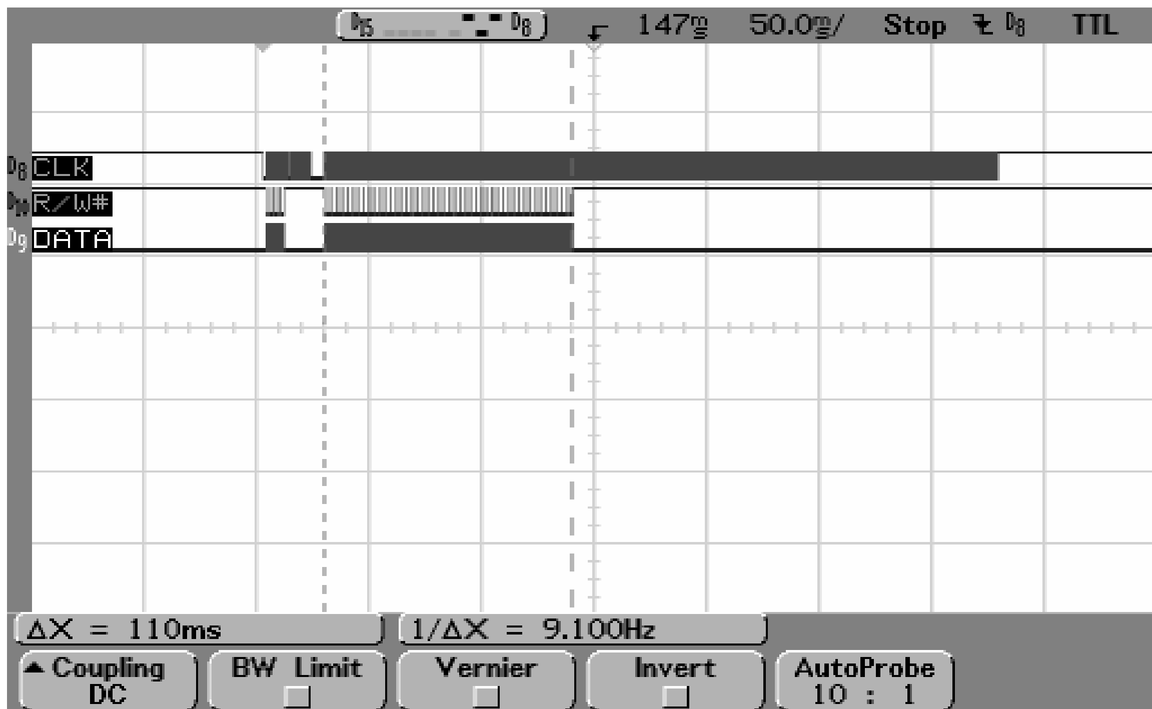


Figure 1. All USBee ZX transfers for USBeeZXApp.c

We will now break down the transfers above so that you can see the exact timing of the signals in the various modes.

The following trace shows the High-Speed Writes (163,835 bytes) followed by Reads (163,835 bytes). We first send out 5 blocks of 32767 bytes which take about 8.2msec. Then we follow with reads of 5 blocks of 32767 bytes which take about 11.6msec.

For writes, each of the blocks of 32768 bytes is bursted at 24Mbytes/sec. The Reads are slower at about 16MHz. The time between bursts is the time it takes for the PC to queue up the next USB transfer. This time may vary depending on your processor speed. Even with these delays in the data stream, the overall bandwidth achieved for writes here is over 20Mbytes/sec and 13.4bytes/sec for reads.

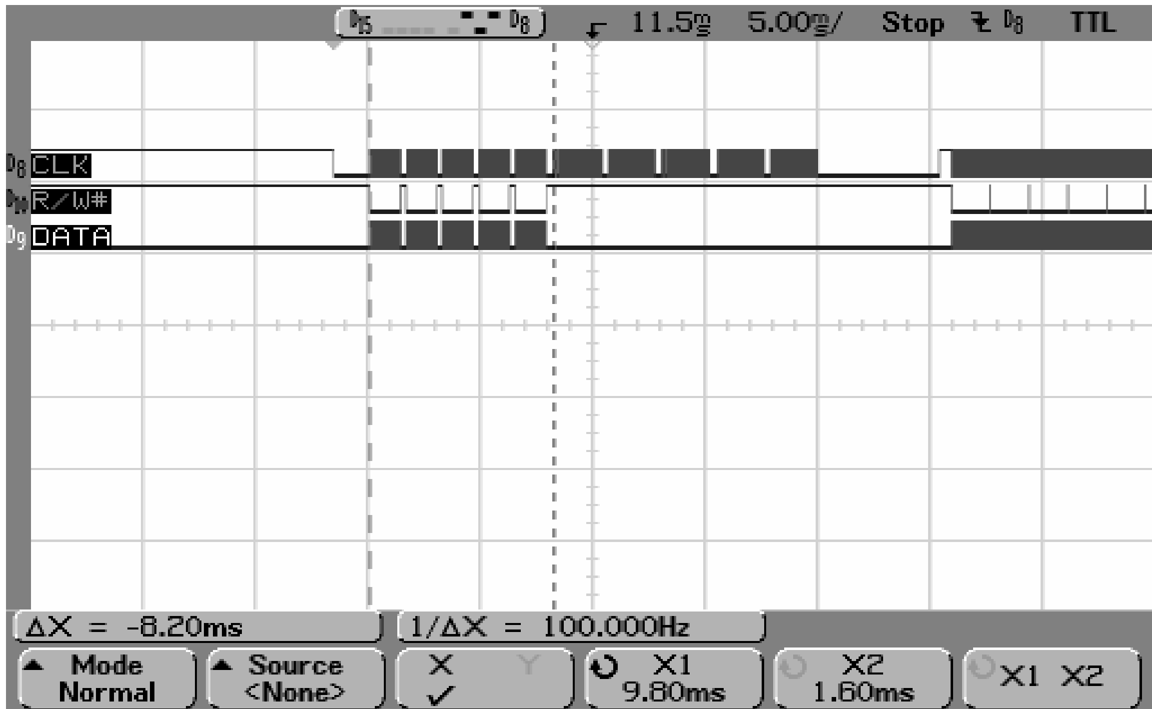


Figure 2. High-Speed Mode – 5 SetSignals followed by 5 GetSignals (each 32767 bytes)

The following trace shows the signal byte transfer timing using the High-Speed mode. You can see that the Clock period is 24MHz and each data line changes close to the rising edge of the clock.

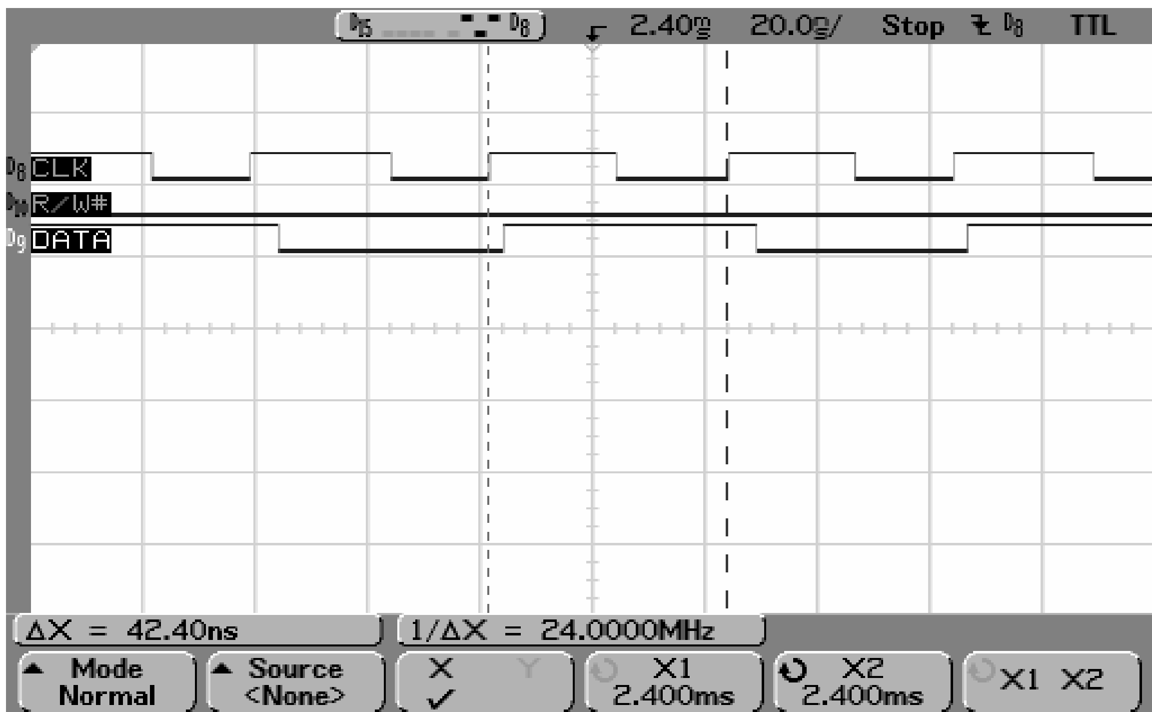


Figure 3. Data and Clock timing in High-Speed mode (SetSignal)

The following traces show the low level timing for the Bi-Directional Mode SetSignal and GetSignal calls.

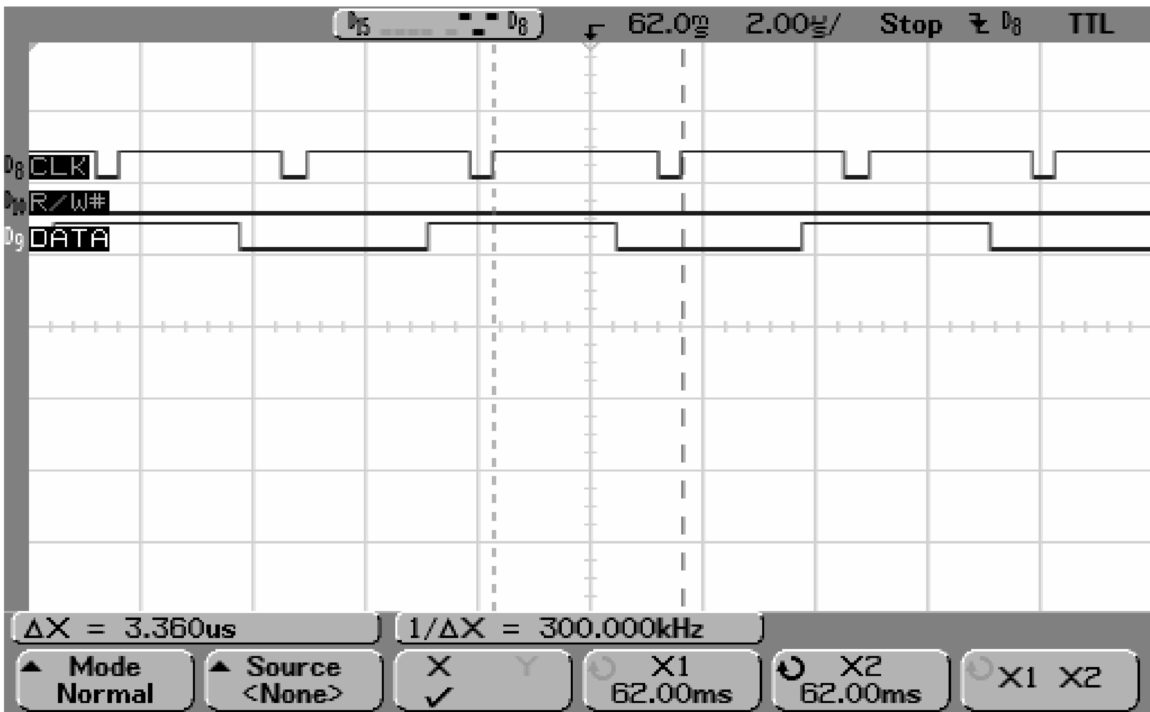


Figure 4. Bi-Directional mode SetSignal byte timing

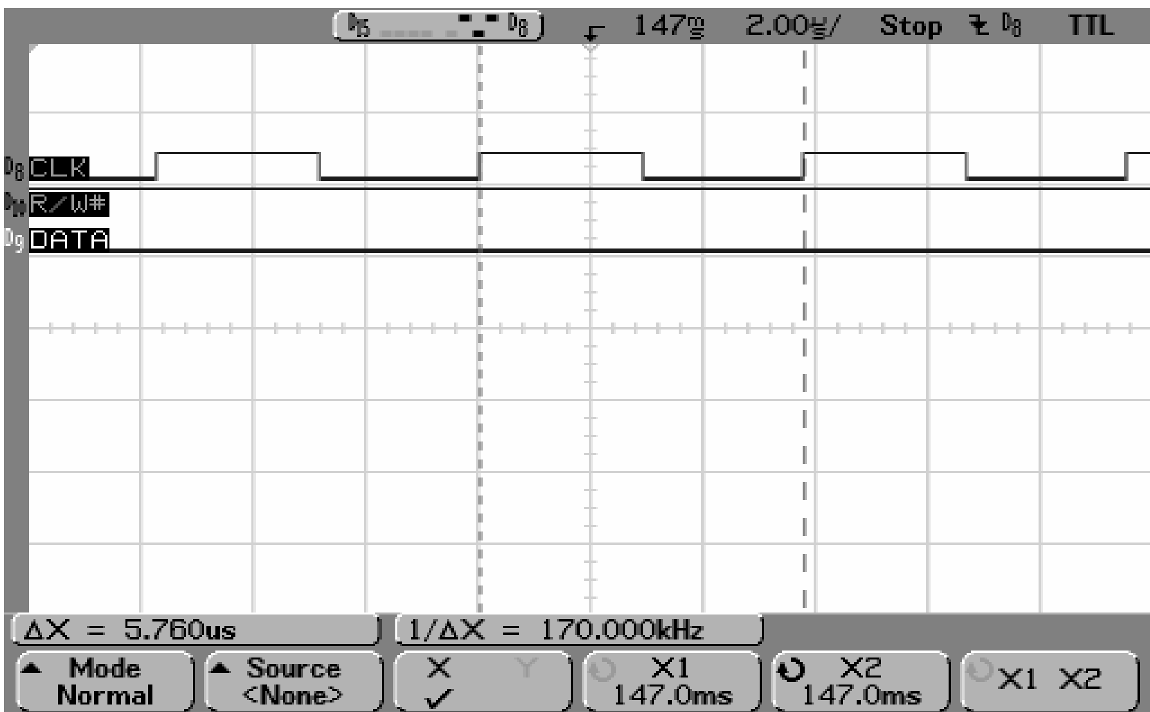


Figure 5. Bi-Directional mode GetSignal byte timing

The data is sampled in the middle of the low clock period.

All of the above traces can have the opposite polarity for the CLK line by setting the appropriate bit in the SetMode parameter.

5 Inside the USBee ZX Visual Basic Source Code

5.1 Function definitions

The following are the API declarations that control the USBee ZX pod. These are the exact same calls that the C source code calls above. Refer to the above sections for a detailed description of the parameters.

5.2 Example Code

```
Declare Function InitializeZXLAPod Lib "usbzxla" Alias _
"?InitializeZXLAPod@@YGHI@Z" (ByVal PodID As Long) As Long

Declare Function StartCapture Lib "usbzxla" Alias _
"?StartCapture@@YGHIIPAEH0KK@Z" (ByVal SampleRate As Long, _
ByVal ClockMode As Long, ByRef Triggers As Byte, _
ByVal TriggerNumber As Long,ByVal Buffer As Long, _
ByVal Length As Long, ByVal PostStore As Long) As Long

Declare Function StopCapture Lib "usbzxla" Alias _
"?StopCapture@@YGHXZ" () As Long

Declare Function CaptureStatus Lib "usbzxla" Alias _
"?CaptureStatus@@YGHPAD00PAJ110@Z" (ByRef Breaks As Byte, _
ByRef Running As Byte, ByRef Triggered As Byte, _
ByRef StartP As Long, ByRef EndP As Long, ByRef Trigger As Long, _
ByRef Full As Byte) As Long

Declare Function SpeedTest Lib "usbzxla" Alias _
"?SpeedTest@@YGHXZ" () As Long

Declare Function MakeBuffer Lib "usbzxla" Alias _
"?MakeBuffer@@YGPAEK@Z" (ByVal Size As Long) As Long

Declare Function DeleteBuffer Lib "usbzxla" Alias _
"?DeleteBuffer@@YGHPAE@Z" (ByVal Buffer As Long) As Long

Declare Function LoggedData Lib "usbzxla" Alias _
"?LoggedData@@YGJK@Z" (ByVal Index As Long) As Long

Declare Function InitializeZXSGPod Lib "usbzxla" Alias _
"?InitializeZXLAPod@@YGHI@Z" (ByVal PodID As Long) As Long

Declare Function GenerateStatus Lib "usbzxla" Alias _
"?GenerateStatus@@YGHPAD000@Z" (ByRef Breaks As Byte, _
ByRef Running As Byte, ByRef Triggered As Byte, _
ByRef Complete As Byte) As Long
```

```
Declare Function StopGenerate Lib "usbzxla" Alias _
"?StopGenerate@@YGHXZ" () As Long

Declare Function StartGenerate Lib "usbzxla" Alias _
"?StartGenerate@@YGHIEPAEK@Z" (ByVal SampleRate As Long, _
ByVal TriggerMode As Byte, ByVal Buffer As Long, _
ByVal Length As Long) As Long

Declare Function SetData Lib "usbzxla" Alias _
"?SetData@@YGJKE@Z" (ByVal Index As Long, ByVal Value As Byte) As Long

Declare Function SetBlockData Lib "usbzxla" Alias _
"?SetBlockData@@YGJKKEE@Z" (ByVal TStart As Long, _
ByVal TEnd As Long, ByVal Bit As Byte, ByVal Value As Byte) As Long

Declare Function SetSignals Lib "usbzxla" Alias _
"?SetSignals@@YGHEIPAE@Z" (ByVal State As Byte, ByVal Length As Long, _
ByRef Buffer As Byte) As Long

Declare Function GetSignals Lib "usbzxla" Alias _
"?GetSignals@@YGHEIPAE@Z" (ByVal State As Byte, ByVal Length As Long, _
ByRef Buffer As Byte) As Long

Declare Function SetMode Lib "usbzxla" Alias "?SetMode@@YGHH@Z" _
(ByVal mode As Byte) As Long

Declare Function SetData Lib "usbzxla" Alias _
"?SetData@@YGJKE@Z" (ByVal Index As Long, ByVal Value As Byte) As Long

Declare Function SetBlockData Lib "usbzxla" Alias _
"?SetBlockData@@YGJKKEE@Z" (ByVal TStart As Long, ByVal TEnd As Long,

Declare Function LoggedData Lib "usbzxla" Alias _
"?LoggedData@@YGJK@Z" (ByVal Index As Long) As Long
```

5.3 Visual Basic Sample Code

```
Declare Function InitializeZXLAPod Lib "usbzxla" Alias "?InitializeZXLAPod@@YGHI@Z" (ByVal PodID As Long) As Long
Declare Function StartCapture Lib "usbzxla" Alias "?StartCapture@@YGHIIPAEH0KK@Z" (ByVal SampleRate As Long, ByVal
ClockMode As Long, ByRef Triggers As Byte, ByVal TriggerNumber As Long, ByVal Buffer As Long, ByVal Length As Long,
ByVal PostStore As Long) As Long
Declare Function StopCapture Lib "usbzxla" Alias "?StopCapture@@YGHXZ" () As Long
Declare Function CaptureStatus Lib "usbzxla" Alias "?CaptureStatus@@YGHPAD00PAJ110@Z" (ByRef Breaks As Byte, ByRef
Running As Byte, ByRef Triggered As Byte, ByRef StartP As Long, ByRef EndP As Long, ByRef Trigger As Long, ByRef Full
As Byte) As Long
Declare Function SpeedTest Lib "usbzxla" Alias "?SpeedTest@@YGHXZ" () As Long
Declare Function MakeBuffer Lib "usbzxla" Alias "?MakeBuffer@@YGPAEK@Z" (ByVal Size As Long) As Long
Declare Function DeleteBuffer Lib "usbzxla" Alias "?DeleteBuffer@@YGHPAE@Z" (ByVal Buffer As Long) As Long
Declare Function LoggedData Lib "usbzxla" Alias "?LoggedData@@YGJK@Z" (ByVal Index As Long) As Long
Declare Function InitializeZXSGPod Lib "usbzxla" Alias "?InitializeZXLAPod@@YGHI@Z" (ByVal PodID As Long) As Long
Declare Function GenerateStatus Lib "usbzxla" Alias "?GenerateStatus@@YGHPAD000@Z" (ByRef Breaks As Byte, ByRef Running
As Byte, ByRef Triggered As Byte, ByRef Complete As Byte) As Long
Declare Function StopGenerate Lib "usbzxla" Alias "?StopGenerate@@YGHXZ" () As Long
Declare Function StartGenerate Lib "usbzxla" Alias "?StartGenerate@@YGHIEPAEK@Z" (ByVal SampleRate As Long, ByVal
TriggerMode As Byte, ByVal Buffer As Long, ByVal Length As Long) As Long
Declare Function SetData Lib "usbzxla" Alias "?SetData@@YGJKE@Z" (ByVal Index As Long, ByVal Value As Byte) As Long
Declare Function SetSignals Lib "usbzxla" Alias "?SetSignals@@YGHEIPAE@Z" (ByVal State As Byte, ByVal Length As Long,
ByRef Buffer As Byte) As Long
Declare Function GetSignals Lib "usbzxla" Alias "?GetSignals@@YGHEIPAE@Z" (ByVal State As Byte, ByVal Length As Long,
ByRef Buffer As Byte) As Long
Declare Function SetMode Lib "usbzxla" Alias "?SetMode@@YGHH@Z" (ByVal mode As Byte) As Long


' GetSignal and SetSignal Mode definitions
Global Const FAST_ONEWAY_DATA = 1
Global Const SLOW_TWOWAY_DATA = 0
```

```
Global Const DATA_CHANGES_ON_RISING_EDGE = 2
Global Const DATA_CHANGES_ON_FALLING_EDGE = 0
Global Const DATA_IS_SAMPLED_ON_RISING_EDGE = 0
Global Const DATA_IS_SAMPLED_ON_FALLING_EDGE = 2

Global Const C_24MHz = (0 * 4)
Global Const C_12MHz = (1 * 4)
Global Const C_6MHz = (2 * 4)
Global Const C_3MHz = (3 * 4)
Global Const C_1MHz = (4 * 4)

Global TriggerMode As Integer
Global Rate As Byte
Global NumberOfSamples As Long
Global BigBuffer As Long
Global TriggerSetting(256) As Byte           ' Holds the screen trigger states
Global TriggerStates As Long
Global ReturnVal As Long
Global TRunning As Byte     ' Are we currently capturing data?
Global TBreaks As Byte      ' Were there breaks in the data? (only sample if still running)
Global TTriggered As Byte   ' Are we triggered yet?
Global TStartP As Long      ' What is the starting pointer
Global TEndP As Long        ' What is the ending pointer
Global TTrigger As Long     ' The trigger position
Global TFull As Byte        ' How full are we (0 to 100%)
Sub DoIt()


    Dim DataInBuffer(65536) As Byte
    Dim DataOutBuffer(65536) As Byte
    Dim ReturnVal As Integer
    Dim X As Long
    Dim ch As Byte
    Dim NumberOfSamples As Long
    Dim BigBuffer As Long
    Dim PodNumber As Integer

    Mainform.OutText.Cls

    ' USBee ZX Sample Application

    PodNumber = 123     ' *******  Change this to your pod number **********

    '***********************************
    ' Pod Initializations Functions - must call InitializeZXLAPod before using any functions
    '***********************************

    ReturnVal = InitializeZXLAPod(PodNumber)
    If ReturnVal = 0 Then
        MsgBox "Pod Not Found"
        End
    End If


    '***********************************
    ' Basic I/O Functions
    '***********************************

    Mainform.OutText.Print "Sample USBee ZX Toolbuilder application in Visual Basic"

    ' Fill the output buffer with some fake data
    For X = 0 To 65535
        DataOutBuffer(X) = X And 255
    Next X

    Mainform.OutText.Print "Setting the Mode to fast mode"
    ReturnVal = SetMode(FAST_ONEWAY_DATA Or DATA_CHANGES_ON_RISING_EDGE Or C_24MHz)
    If (ReturnVal <> 1) Then
        Mainform.OutText.Print "Failure setting the mode" & ReturnVal
        Exit Sub
    End If

    Mainform.OutText.Print "Sending 163,835 bytes out the pod"
    For X = 0 To 5
        DataOutBuffer(0) = ch
        ch = ch + 1
        ReturnVal = SetSignals(255, 32767, DataOutBuffer(0))
    Next X

    Mainform.OutText.Print "Reading 163,835 bytes from the pod signals"
    For X = 0 To 5
        ReturnVal = GetSignals(0, 32767, DataInBuffer(0))
    Next X

    Mainform.OutText.Print "Setting the Mode to bi-directional mode"
    ReturnVal = SetMode(SLOW_TWOWAY_DATA Or DATA_IS_SAMPLED_ON_RISING_EDGE)
    If (ReturnVal <> 1) Then
        Mainform.OutText.Print "Failure setting the mode"
        Exit Sub
    End If

    Mainform.OutText.Print "Sending 32767 bytes out the pod"

    DataOutBuffer(0) = ch
    ch = ch + 1
```

```
    ReturnVal = SetSignals(255, 32767, DataOutBuffer(0))

    Mainform.OutText.Print "Reading 32767 bytes from the pod signals"
    ReturnVal = GetSignals(0, 32767, DataInBuffer(0))

End Sub
```

5.3.1 Performance Analysis

This Visual Basic example gets the same performance as that described in section 4.4.2.