USBee DX Test Pod Users Manual

CWAV
www.usbee.com

USBee DX Test Pod
Users Manual

CWAV
www.usbee.com
(951) 693-3065
support@usbee.com

USBee DX License Agreement

The following License Agreement is a legal agreement between you (either an individual or entity), the end user, and CWAV. You have received the USBee Package, which consists of the USBee Pod, USBee Software and Documentation. If you do not agree to the terms of the agreement, return the unopened USBee Pod and the accompanying items to CWAV for a full refund. Contact support@usbee.com for the return address.

By opening and using the USBee Pod, you agree to be bound by the terms of this Agreement.

Grant of License
CWAV provides royalty-free Software, both in the USBee Package and on-line at www.usbee.com, for use with the USBee Pod and grants you license to use this Software under the following conditions: a) You may use the USBee Software only in conjunction with the USBee Pod, or in demonstration mode with no USBee Pod connected, b) You may not use this Software in conjunction with any pod providing similar functionality made by other than CWAV, and c) You may not sell, rent, transfer or lease the Software to another party.

Copyright
No part of the USBee Package (including but not limited to manuals, labels, USBee Pod, or accompanying diskettes) may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of CWAV, with the sole exception of making backup copies of the diskettes for restoration purposes. You may not reverse engineer, decompile, disassemble, merge or alter the USBee Software or USBee Pod in any way.

Limited Warranty
The USBee Package and related contents are provided “as is” without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with the sole exception of manufacturing failures in the USBee Pod or diskettes. CWAV warrants the USBee Pod and physical diskettes to be free from defects in materials and workmanship for a period of 12 (twelve) months from the purchase date. If during this period a defect in the above should occur, the defective item may be returned to the place of purchase for a replacement. After this period a nominal fee will be charged for replacement parts. You may, however, return the entire USBee Package within 30 days from the date of purchase for any reason for a full refund as long as the contents are in the same condition as when shipped to you. Damaged or incomplete USBee Packages will not be refunded.

The information in the Software and Documentation is subject to change without notice and, except for the warranty, does not represent a commitment on the part of CWAV. CWAV cannot be held liable for any mistakes in these items and reserves the right to make changes to the product in order to make improvements at any time.

IN NO EVENT WILL CWAV BE LIABLE TO YOU FOR DAMAGES, DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL, INCLUDING DAMAGES FOR ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE SUCH USBEE POD, SOFTWARE AND DOCUMENTATION, EVEN IF CWAV HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR FOR ANY CLAIM BY ANY OTHER PARTY. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. IN NO EVENT WILL CWAV’S LIABILITY FOR DAMAGES TO YOU OR ANY OTHER PERSON EVER EXCEED THE AMOUNT OF THE PURCHASE PRICE PAID BY YOU TO CWAV TO ACQUIRE THE USBEE, REGARDLESS OF THE FORM OF THE CLAIM.

Term
This license agreement is effective until terminated. You may terminate it at any time by returning the USBee Package (together with the USBee Pod, Software and Documentation) to CWAV. It will also terminate upon conditions set forth elsewhere in this agreement or if you fail to comply with any term or condition of this agreement. You agree that upon such termination you will return the USBee Package, together with the USBee Pod, Software and Documentation, to CWAV.

Table of Contents

1 Introducing the USBee DX Pod

The USBee DX Test Pod is a large sample buffer PC and USB based programmable multifunction digital storage 2-channel oscilloscope, 16-channel logic analyzer and digital signal generator in a single compact and easy to use device. It is the ideal bench tool for engineers, hobbyists and students

Connecting to your PC, the USBee DX Test Pod uses the power and speed of the USB 2.0 bus to capture and control analog and digital information from your own hardware designs. The USBee DX takes advantage of already existing PC resources by streaming data over the High-Speed USB 2.0 bus to and from the PC. This allows the PC to perform all of the triggering and data storing and makes possible an affordable USBee DX, while pushing the sample storage capabilities orders of magnitudes beyond that of traditional dedicated oscilloscopes, logic analyzers or signal generators. The USBee DX Test Pod can utilize available PC memory as the sample buffer, allowing selectable sample depths from one to many hundreds of millions of samples.

The USBee DX Test Pod can capture and generate samples up to a maximum of 24 million samples per second depending on the PC configuration. The USBee DX Auto-Calibration feature automatically reduces the sample rate to ensure accurate and reliable timing, even on systems with slower processor and USB bus speeds. The USBee DX Test Pod perfectly merged features and functions to provide exactly

the performance needed for hardware and microprocessor designs such as BASIC Stamp and PIC systems to ensure an affordable and compact unit.

The USBee DX Test Pod does not need an external power supply. The USB bus supplies the power to the pod, so your PC will be supplying the power. The Pod does, however, require a self powered hub (not bus powered) if a hub is used between the PC and Pod.

WARNING

As with all electronic equipment where you are working with live voltages, it is possible to hurt yourself or damage equipment if not used properly. Although we have designed the USBee DX pod for normal operating conditions, you can cause serious harm to humans and equipment by using the pod in conditions for which it is not specified.

Specifically:

- ALWAYS connect at least one GND line to your circuits ground
- NEVER connect the digital signal lines (0 thru 7, TRG and CLK) to any voltage other than between 0 to 5 Volts
- NEVER connect the analog signal lines (CH1 and CH2) to any voltage other than between -10 and +10 Volts
- The USBee DX actively drives Pod signals 0 through F in some applications. Make sure that these pod test leads are either unconnected or connected to signals that are not also driving. Connecting these signals to other active signals can cause damage to you, your circuit under test or the USBee DX test pod, for which CWAV is not responsible.
- Plug in the USBee DX Pod into a powered PC BEFORE connecting the leads to your design.

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

1.1 PC System Requirements

The USBee DX Test Pod requires the following minimum PC features:

- Windows® 2000, XP or Vista operating system
- Pentium or higher processor
- One USB2.0 High Speed enabled port. It will not run on USB 1.1 Full Speed ports.
- 32MBytes of RAM
- 125MBytes of Hard disk space
- Internet Access (for software updates and technical support)

1.2 Each Package Includes

The USBee DX contains the following in each package:

- USBee DX Universal Serial Bus Pod
- Set of 24 multicolored test leads and high performance miniature test clips
- Getting Started Guide
- USB Cable (A to Mini-B)
- USBee DX Test Pod CD-ROM

1.3 Hardware Specifications

Connection to PC	USB 2.0 High Speed (required)
Power	via USB cable
Test Leads	24 9" leads with 0.025" square sockets
USB Cable Length	6 Feet
Dimensions	2.25" x 1.5" x 0.75"
Minigrip Test Clips	24

The maximum sample rate for any mode depends on your PC hardware CPU speed and USB 2.0 bus utilization. For the fastest possible sample rates, follow these simple steps:

- Disconnect all other USB devices not needed from the PC
- Do not run other applications while capturing or generating samples.

The maximum sample buffer size also depends on your PC available RAM at the time the applications are started.

1.4 Software Installation

Each USBee DX pod is shipped with an installation CD that contains the USBee DX software and manuals. You can also download the software from the software from our web site at www.usbee.com. Either way, you must install the software on each PC you want to use the USBee DX on before you plug in the device.

To install the software:

- Download the USBee DX Software from http://www.usbee.com/download.htm and unzip into a new directory. Or insert the USBee DX CD in your CD drive. Unzip the downloaded file into a new directory.
- From the "Start|Run" Windows® menu, run the SETUP.EXE.
- Follow the instructions on the screen to install the USBee DX software on your hard drive. This may take several minutes.
- Now, plug a USB A to USB Mini-B cable in the USBee DX and the other end into a free USB 2.0 High Speed port on your computer.
- You will see a dialog box indicating that it found new hardware and is installing the software for it. Follow the on screen directions to finish the driver install.
- You will see another dialog box indicating that it found new hardware and is installing the software for it. Follow the on screen directions to finish the driver install.
- The USBee DX Software is now installed.
- Run any of the applications by going to the Start | Program Files | USBee DX Test Pod and choosing the application you want to run.

1.5 Calibration

Since electronic components vary values slightly over time and temperature, the USBee DX Pod requires calibration periodically to maintain accuracy. The USBee DX has been calibrated during manufacturing and should maintain accuracy for a long time, but in case you want to recalibrate the device, follow these steps. The calibration values are stored inside the USBee DX pod. Without calibration the measurements of the oscilloscope may not be accurate as the pod ages.

To calibrate your USBee DX Pod you will need the following equipment:

- External Voltage Source (between 5V and 9V)
- High Precision Multimeter

When you are ready to calibrate the USBee DX Pod, plug in the pod and run the Oscilloscope and Logic Analyzer application. Then go to

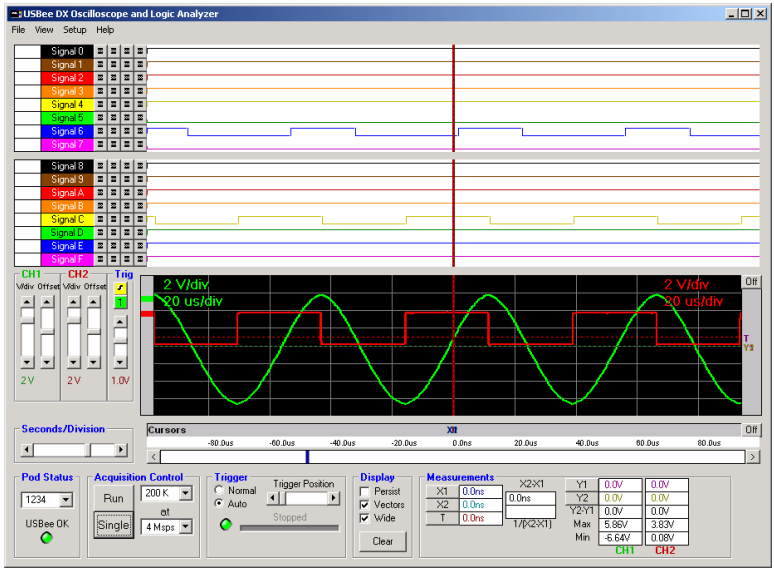
the menu item Setup | Calibrate. You will be asked to confirm that you really want to do the calibration. If so, press Yes, otherwise press No. Then follow these steps:

- Connect the CH1 and CH2 signals to the GND signal using the test leads and press OK. A measurement will be taken.
- Connect the GND signal to the ground and the CH1 and CH2 signals to the positive connection of the External Voltage Source using the test leads and press OK. A measurement will be taken.
- With the Multimeter, measure the actual voltage between the GND signal and the CH1 signal and enter this value in the dialog box.
- The calibration is now complete. The calibration values have been saved inside the pod.

The analog measurements of your USBee DX pod are only as accurate as the voltages supplied and measured during calibration.

2 Logic Analyzer and Oscilloscope (MSO)

This section details the operation of the Logic Analyzer and Oscilloscope application that comes with the USBee DX, also known as a Mixed Signal Oscilloscope, or MSO. Below you see the application screen after startup.



The USBee DX Mixed Signal Oscilloscope functions as a standard Digital Storage Oscilloscope combined with a Digital Logic Analyzer, which is a tool used to measure and display analog and digital signals in a graphical format. It displays what the analog and digital input signals do over time. The digital and analog samples are taken at the same time and can be used to debug mixed signal systems.

2.1 Quick Start

In order to quickly get up and running using this application, here is a step by step list of the things you need to do to view a mixed signal (analog and digital) waveform trace.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.

- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the CH1 pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire. Connect the other end of the wire to your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to your signal of choice.
- Connect any of the digital inputs 0 thru F on the USBee DX pod to one of the signal wires using the small socket on the end of the wire. Connect the other end of the wire to your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to your signal of choice.
- Run the Oscilloscope and Logic Analyzer Application.
- Press the Run button. This will capture and display the current activity on all of the signals.
- You can then scroll the display, either by using the slider bars, or by clicking and dragging on the waveform itself. You can also change the knobs to zoom the waveform.
- You can make simple measurements by using the Cursors area (gray bars under and along side the waves). Click the left mouse button to place one cursor and click the right mouse button to place the second. The resulting measurements are then displayed in the Measurements section of the display.

2.2 Mixed Signal Oscilloscope/Logic Analyzer Specifications

Analog Channels	2
Maximum Analog Sample Rate [1]	24 Msps
Analog Bandwidth	40 MHz
Input Impedance	1M Ohm/30 pF
Analog Input Voltage Range	-10V to +10V
Analog Sensitivity	78mV
Analog Resolution	256 steps
Channel Buffer Depth [2]	>200k Samples
Volts per Division Settings	100mV to 5V in 6 steps
Time per Division Settings	100ns to 2s in 23 steps
Trigger Modes	Auto, Normal, Analog and Digital Triggers
Analog Trigger Voltage	Between -10V and +10V
Cursors	2 Time and 2 Voltage
Voltage Display Offset	Up to maximum inputs
Time Display Offset	Up to available buffer depth
Trigger Position Setting	10% to 90%
Measurements	Min, Max
Digital Channels	16
Maximum Digital Sample Rate [1]	24 Msps
Internal Clocking	Yes
External Clocking	Yes – through Parallel Decoder
Digital Trigger Levels	4
Digital Trigger Qualifiers	Rising Edge, Falling Edge, High,Low
Trigger Prestore	Yes
Trigger Poststore	Yes
Sample Clock Output	Yes
Maximum Digital Input Voltage	+5.5V

Digital Input Low Level	< 0.8V
Digital Input High Level	> 2.0V

[1] Maximum sample rate depends on your PC hardware CPU speed, USB 2.0 bus utilization and number of channels selected.

For the fastest possible sample rates, follow these simple steps:

- Disconnect all other USB devices not needed from the PC
- Do not run other applications while capturing or generating samples.

[2] Maximum buffer size depends on your PC available RAM at the time the application is started. Each sample requires 4 bytes of RAM (16 bits for the 16 digital lines and 8 bits each for the 2 analog channels)

2.3 Features

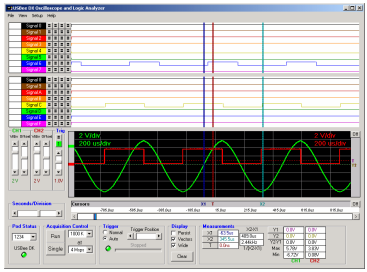
2.3.1 Setup Configuration

The MSO can capture 16 channels of digital and 2 channels of analog at the same time. All of the captured data is streamed over the USB bus to your PC to be stored in the RAM of the PC. In order to optimize the sample bandwidth you can choose to see only the channels of interest to you.

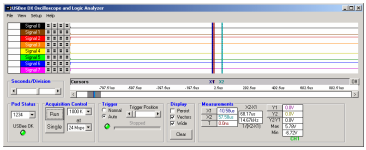
The configurations available are as follows:

Analog Channels	Digital Channels	Max Sample Rate
0	8	24 Msps
0	16	12 Msps
1	0	24 Msps
1	8	12 Msps
1	16	6 Msps
2	0	12 Msps
2	8	6 Msps
2	16	6 Msps

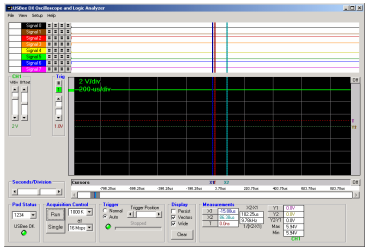
To select a configuration, click **Setup** on the menu and select the configuration of your choice. Below are examples of the application in various modes.



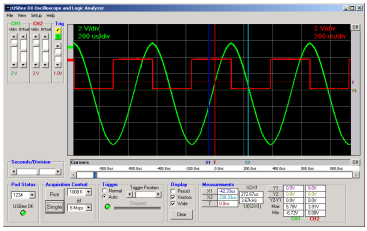
16 Digital–2 Analog Channels



8 Digital–0 Analog Channels



8 Digital–1 Analog Channels



0 Digital–2 Analog Channels

2.3.2 Signal Names

To change the names shown for a signal, click on the signal name and enter a new name.

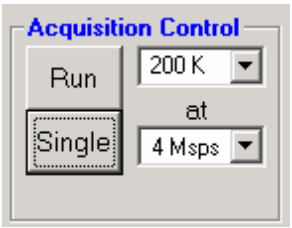
2.3.3 Pod Status

The MSO display shows a current USBee DX **Pod Status** by a red or green LED. When a USBee DX is connected to the computer, the Green LED shows and the list box shows the available **Pod ID List** for all of the USBee DX's that are connected. You can choose which one you want to use. The others will be unaffected. If a USBee DX is not connected, the LED will glow red and indicate that there is no pod attached.

If you run the software with no pod attached, it will run in demonstration mode and simulate data so that you can still see how the software functions.

2.3.4 Acquisition Control

The MSO captures the behavior of the digital and analog signals and displays them as “traces” in the waveform window. The Acquisition Control section of the display lets you choose how the traces are captured. Below is the Acquisition Control section of the display.



When the MSO is first started, no acquisition is taking place. You need to press one of the acquisition buttons to capture data.

The Run button is the **Run / Stop** control. This Run mode performs an infinite series of traces, one after the other. This lets you see frequent updates of what the actual signals are doing in real time. If you would like to stop the updating, just press the Stop button and the updating will stop. This run mode is great for signals that repeat over time.

The **Single** button captures a single trace and stops. This mode is good for detailed analysis of a single event, rather than one that occurs repeatedly.

The **Buffer Size** lets you select the size of the Sample Buffer that is used. For each trace, the buffer is completely filled, and then the waveform is displayed. You can choose buffers that will capture the information that you want to see, but remember that the larger the buffer, the longer it will take to fill.

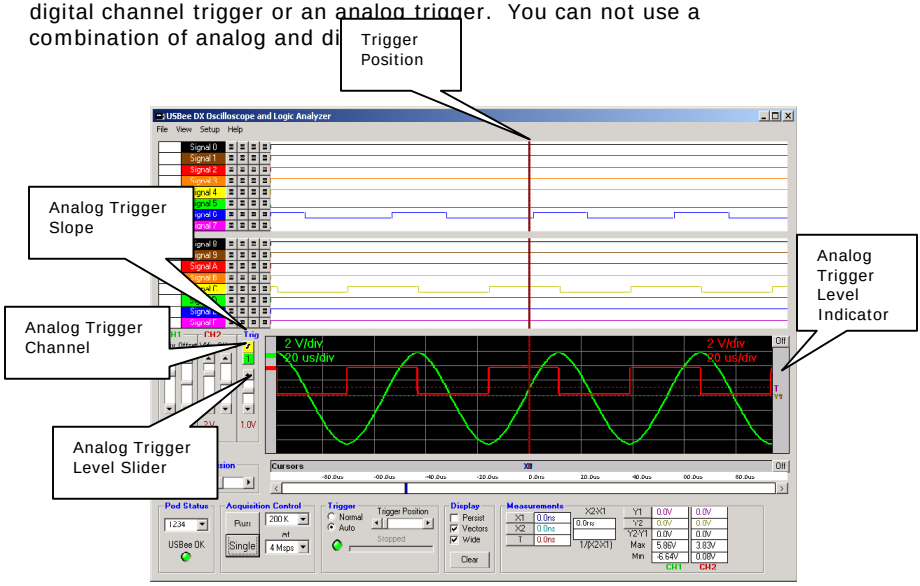
You can also choose the **Sample Rate** that you want samples taken. You can choose from 1Msps (samples per second) to up to 24 Msps. The actual maximum sample rate depends on your PC configuration and the number of channels that you are using. See the table below for maximum sample rates for a given channel setting.

Analog Channels	Digital Channels	Max Sample Rate
0	8	24 Msps
0	16	12 Msps
1	0	24 Msps
1	8	12 Msps
1	16	6 Msps
2	0	12 Msps
2	8	6 Msps
2	16	6 Msps

2.3.5 Trigger Control

The Mixed Signal Oscilloscope uses a Trigger mechanism to allow you to capture just the data that you want to see. You can use either a

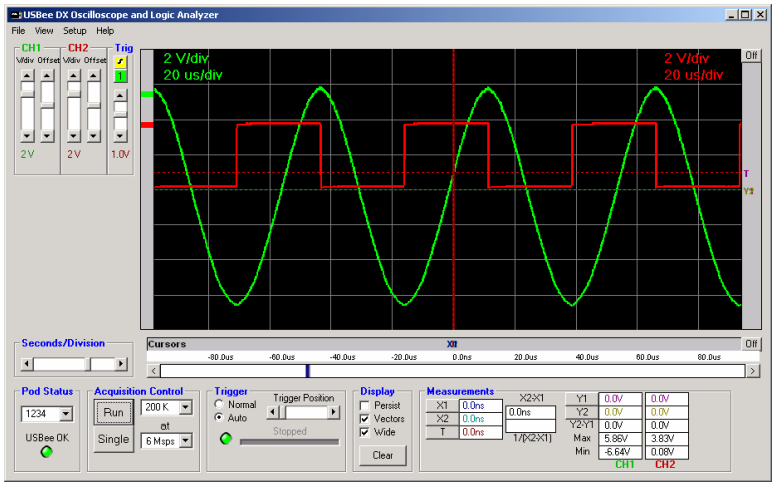
digital channel trigger or an analog trigger. You can not use a combination of analog and di



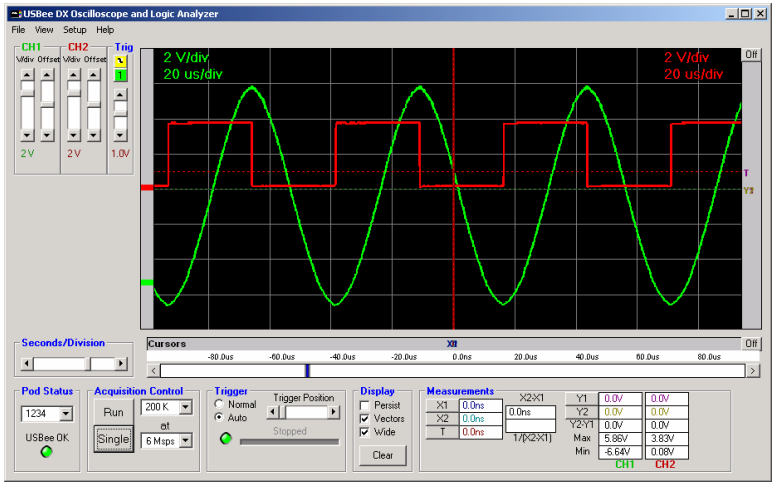
For an **Analog trigger**, you can specify the trigger voltage level (-10V to +10V) by using the slider on the left hand side of the analog waveform display. A red line that indicates the trigger level will momentarily be shown as you scroll this level. A small T will also be shown on the right hand side of the screen (in the cursors bar) that shows where this level is set to.

For an analog trigger, the trigger position is where the waveform crossed the **Trigger Voltage** level that you have set at the specified slope. To move the trigger voltage level, just move the slider on the left of the waveform. To change the slope, press the Analog Trigger Slope button.

You can also specify if you want the MSO to trigger on a **Rising or Falling Edge**. The following figures show a trace captured on each of the edges.



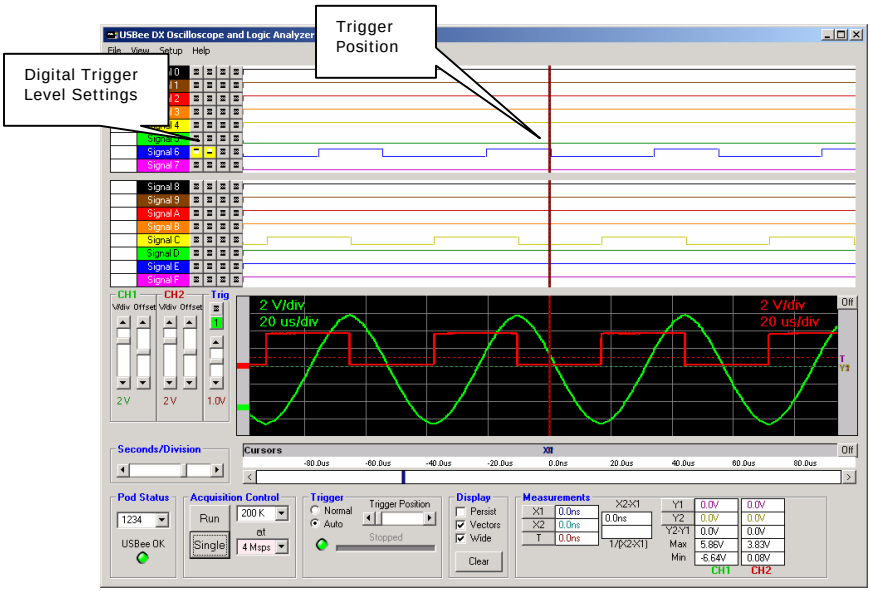
Analog Trigger Slope = Rising Edge



Analog Trigger Slope = Falling Edge

The Trigger position is placed where the actual signal crosses the trigger voltage with the proper slope. The USBee DX allows for huge sample buffers, which means that you can capture much more data than can be shown on a single screen. Therefore you can scroll the waveform back and forth on the display to see what happened before or after the trigger.

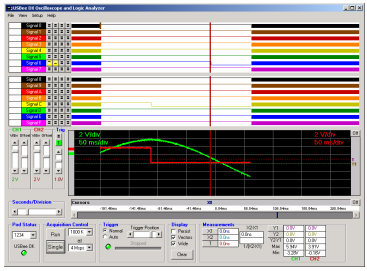
For a **Digital trigger**, you can specify the digital states for any of the 16 signals that must be present on the digital lines before it will trigger. Below shows the trigger settings (to the right of the Signal labels). This example shows that we want to trigger on a falling edge of Signal 6, which is represented by a high level followed by a low level. To change the level of any of the trigger settings, just click the level button to change from don't care to high to low.



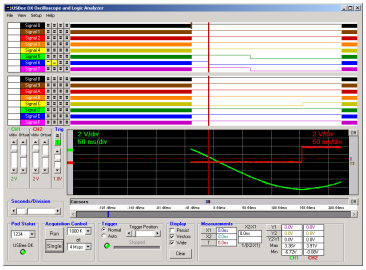
The digital trigger condition is made up of up to 4 sequential states of any of the 16 signals. Each state for a single signal can be high, low or don't care. This allows you to trigger on rising edges, falling edges, edges during another signals constant level, or one edge followed by another edge.

The waveforms are shown with a trigger position which represents where the trigger occurred. This sample point is marked on the waveform display with a Vertical red dotted line and a "T" in the horizontal cursors bar.

You can use the **Trigger Position** setting to specify how much of the data that is in the sample buffer comes before the actual trigger position. If you place the Trigger Position all the way to the left, most of the samples taken will be after the trigger sample. If you place Trigger Position all the way to the right, most of the samples taken will be before the Trigger sample. This control lets you see what actually happened way before or way after the trigger occurred.



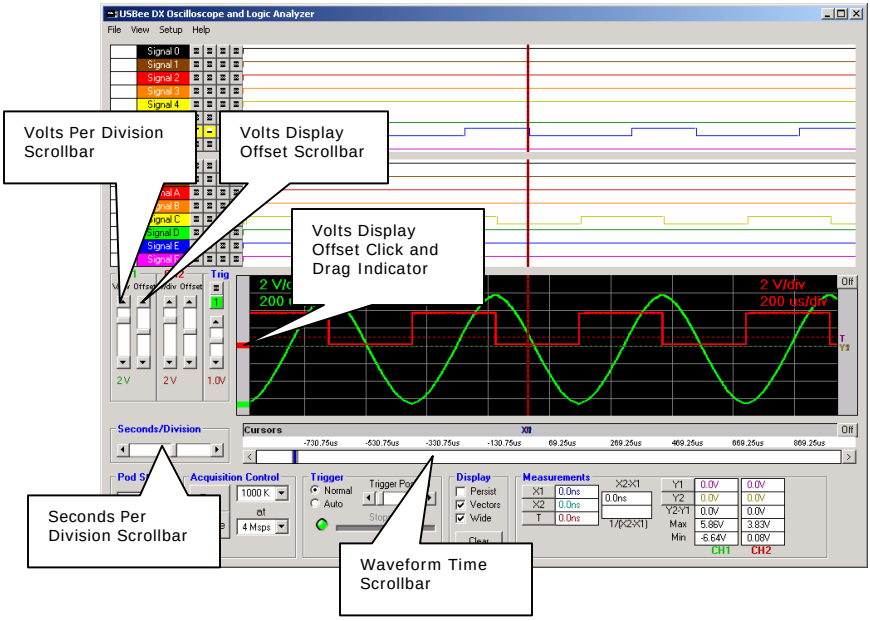
Trigger Position to the Right



Trigger Position to the Left

2.3.6 Waveform Display and Zoom Settings

The Waveform display area is where the measured signal information is shown. It is displayed with time increasing from left to right and voltage increasing from bottom to top. The screen is divided into **Divisions** to help in measuring the waveforms.



The position of the waveform defaults to show the actual trigger position in the center of the screen after a capture. However, you can move the display to see what happened before or after the trigger position.

To **Scroll the Waveforms in Time** left and right, you can use the scroll bar at the bottom of the waveform display (right above all of the

controls), or you can simply click and drag the waveform itself with the left mouse button.

To **Scroll the Analog Waveform in Voltage** up and down, you can use the scroll bar at the left of the waveform display (one for each channel), or you can simply click and drag the waveform itself by using the colored bar to the immediate left of the actual waveform.

To change the number of **Seconds per Division** use the scrollbar at the bottom left of the waveforms. To change the number of **Volts per Division** for an analog channel, use the scrollbars at the left of the analog waveforms. You can also zoom in and out in time by clicking on the waveform. To zoom in, click the left mouse on the waveform window. To zoom out in time, click the right mouse button on the waveform window.

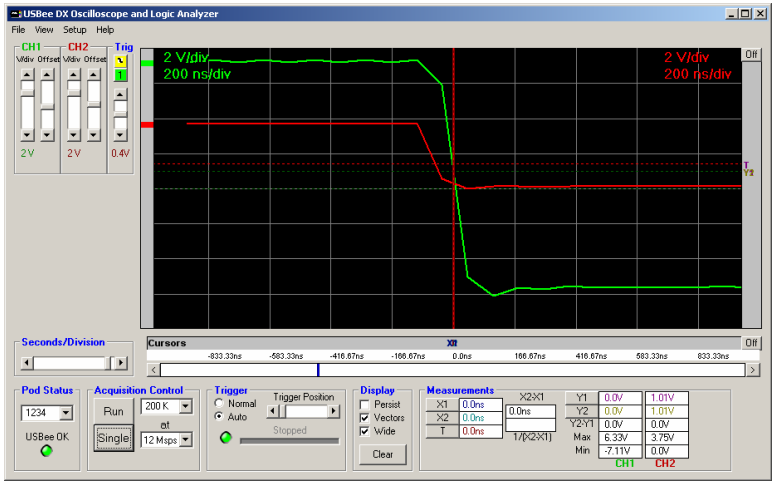
The Display section of the screen shows three selections that affect the way the waveform is displayed.

The **Wide** setting shows the wave using a wider pixel setting. This makes the wave easier to see.

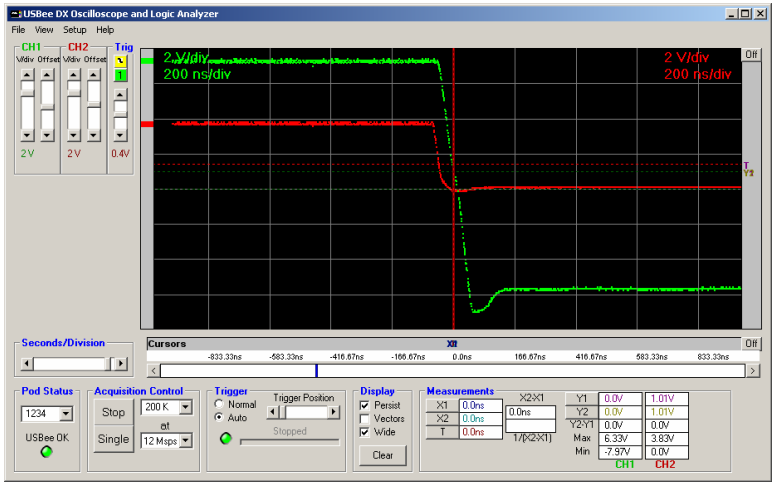
The **Vectors** setting draws the waveform as a line between adjacent samples. With this mode turned off, the samples are shown simply as dots on the display at the sample position.

The **Persist** mode does not clear the display and writes one trace on top of the other trace.

The benefits of these display modes can be seen when you are measuring fast signals and want to get more resolution out of the oscilloscope than the maximum sample rate allows. See the below traces to see the difference. Each trace is taken of the same signal, but the right one shows much more wave detail over a short time of display updates.



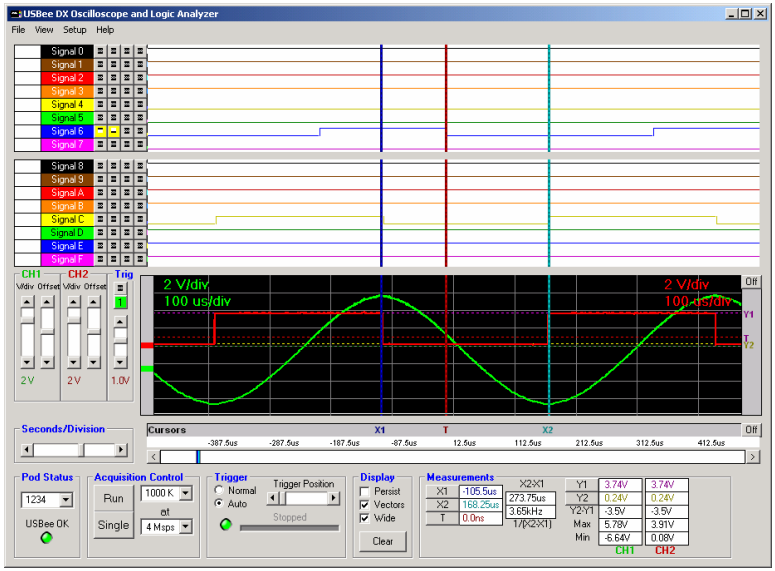
Persist = OFF, Vectors = ON, Wide = ON



Persist = ON, Vectors = OFF, Wide = ON

2.3.7 Measurements and Cursors

The main reason for using an oscilloscope or logic analyzer is to measure the various parts of a waveform. The USBee DX uses cursors to help in these measurements.



The **X1 and X2 Cursors** are placed on any horizontal sample time. This lets you measure the time at a specific location or the time between the two cursors. To place the X cursors, move the mouse to the gray box just below the waveform. When you move the mouse in this window, you will see a temporary line that indicates where the cursors will be placed. Place the X1 cursor by left clicking the mouse at the current location. Place the X2 cursor by right clicking the mouse at the current location.

The **Y1 and Y2 Cursors** are placed on any vertical voltage level. This lets you measure the voltage at a specific location or the difference in voltage between the two cursors. To place the Y cursors, move the mouse to the gray box just to the right of the scroll bar to the right of the waveform. When you move the mouse in this window, you will see a temporary line that indicates where the cursors will be placed. Place the Y1 cursor by left clicking the mouse at the current location. Place the Y2 cursor by right clicking the mouse at the current location.

In the Measurement window, you will see the various measurements made off of these cursors.

- **X1 Position** – time at the X1 cursor relative to the trigger position
- **X2 Position** – time at the X2 cursor relative to the trigger position
- **X2-X1** – time difference between X1 and X2 cursors
- **1/(X2-X1)** – the frequency or the period between X1 and X2 cursors
- **Y1 Position** – voltage at the Y1 cursor relative to Ground for both CH1 and CH2
- **Y2 Position** – voltage at the Y2 cursor relative to Ground for both CH1 and CH2
- **Y2-Y1** – voltage difference between Y1 and Y2 cursors for both CH1 and CH2

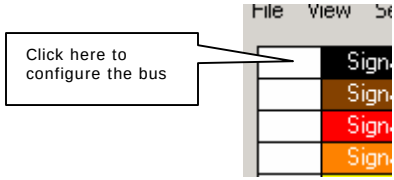
There are also a set of automatic measurements that are made on the analog waveform for each trace. These are calculated without the use of the cursors. These are:

- **Max** – the maximum voltage of all samples in the current trace for both CH1 and CH2
- **Min** – the minimum voltage of all samples in the current trace for both CH1 and CH2

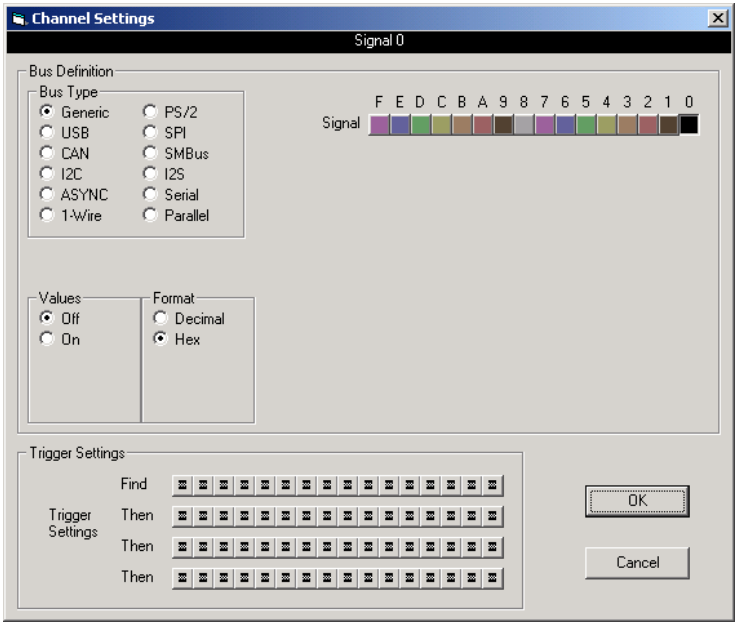
2.3.8 Bus Decoding

The USBee DX Logic Analyzer and Oscilloscope has a power embedded bus decoder feature that allows you to quickly analyze the contents of embedded communications captured by the pod.

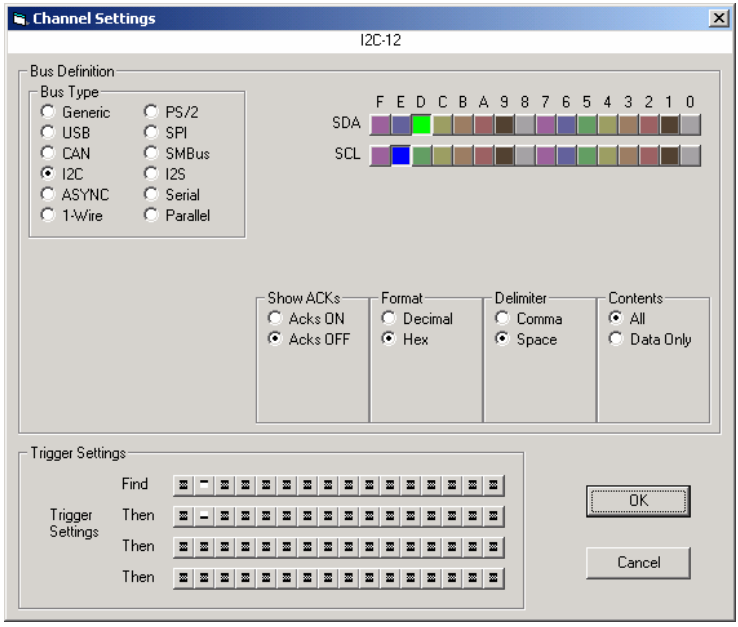
2.3.8.1 Bus Setup



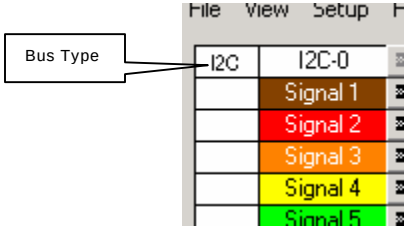
To setup a single line on the waveform display as a bus, click on the white box to the left of the signal name. The Channel Settings dialog box will appear as below.



Select which bus you would like displayed on this line using the Bus Type radio buttons, select the required channels for the given bus type, and click OK. Below is an example of a setup for an I2C bus.



Once set, you see the bus identifier to the left of the signal name on the main screen.

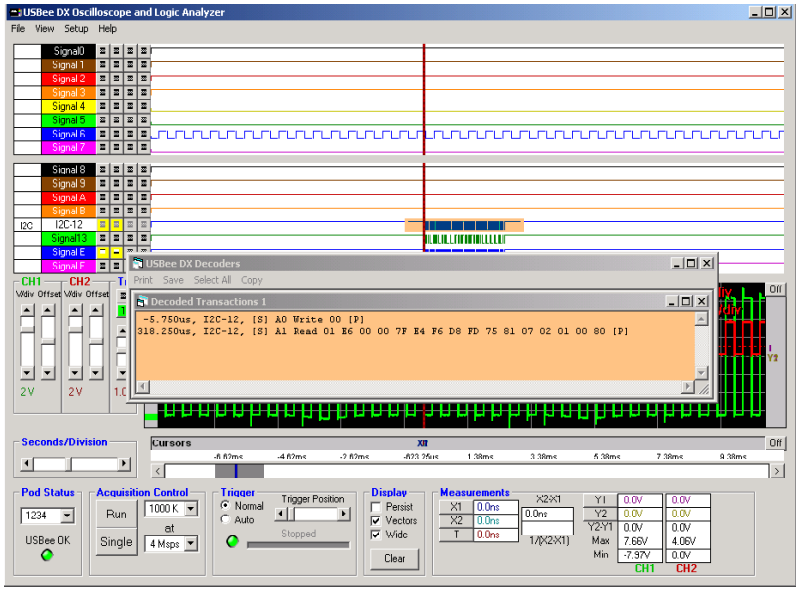


Each bus is renamed with the bus type followed by a number. This allows you to have many of the same types of busses, yet uniquely identify them in decoder listings.

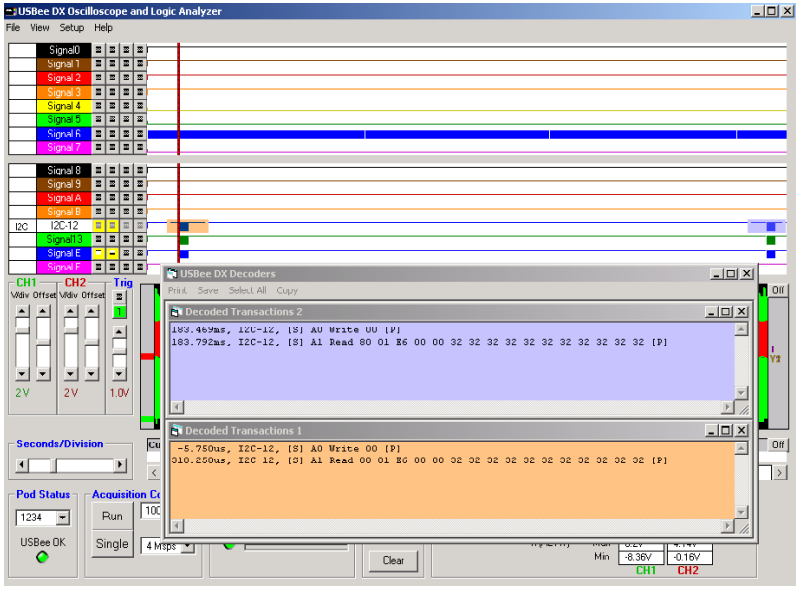
2.3.8.2 Decoding Bus Traffic – Click and Drag

Once a bus is defined you can capture data as usual. You can then scroll and zoom to find the area of interest on that bus.

To decode a portion of the bus traffic, simply **Right-Click and Drag** across the waveform you want to decode. When you let go of the mouse button, the selected section of traffic will be decoded into the decoder window as shown below.



You can then scroll and zoom to see a different portion of the capture and decode a different section of bus traffic in the same way. You can decode up to 4 different sections and each section will display in its own window with matching color highlights.



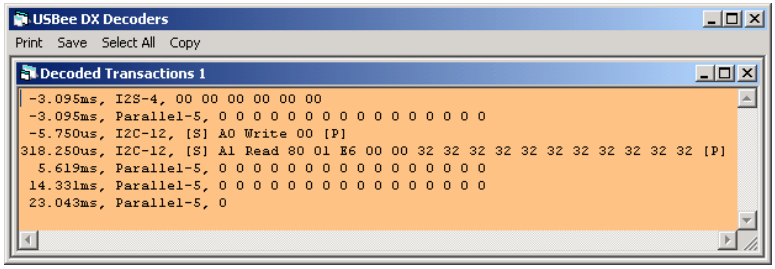
When you click on the text portion of the decode window, the main waveform screen will move to make sure that the decoded section for that window is displayed.

Once the decoded text window contains the data you want to see, you have the option to use the menus to print that data, save it to a text file, or select it and copy it to the clipboard for importing to other programs such as Excel.

2.3.8.3 Decoding Bus Traffic – Multiple Busses

You can also decode multiple busses at the same time and get the traffic displayed in chronological order from the different busses.

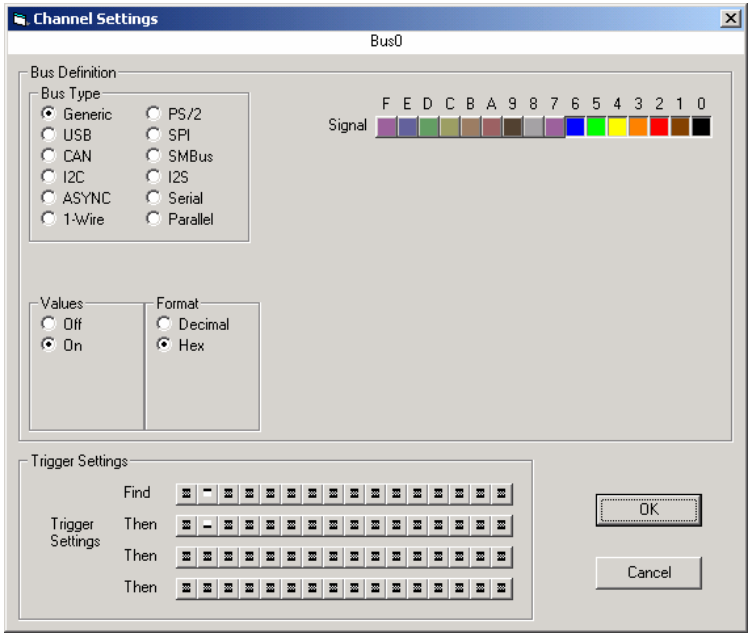
First place the X1 and X2 cursors around the section of time you want decoded. Then choose the menu item View | Decode Busses Between Cursors. The decoder will then decode all busses defined, extract the data for each bus and interlace all data so that each transaction is listed chronologically.



2.3.8.4 Generic Bus Setup

Although not decoded in the decoder windows, you can combine multiple DX signals into a single line on the waveform display using the Generic Bus setting.

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



The resulting waveform shows the signals 0 through 6 on a single line of the display and shows the value on the waveform for those signals.

2.3.8.5 CAN Bus Setup

The CAN Bus Decoder takes the captured data from a CAN bus (11 or 29-bit identifier supported), formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

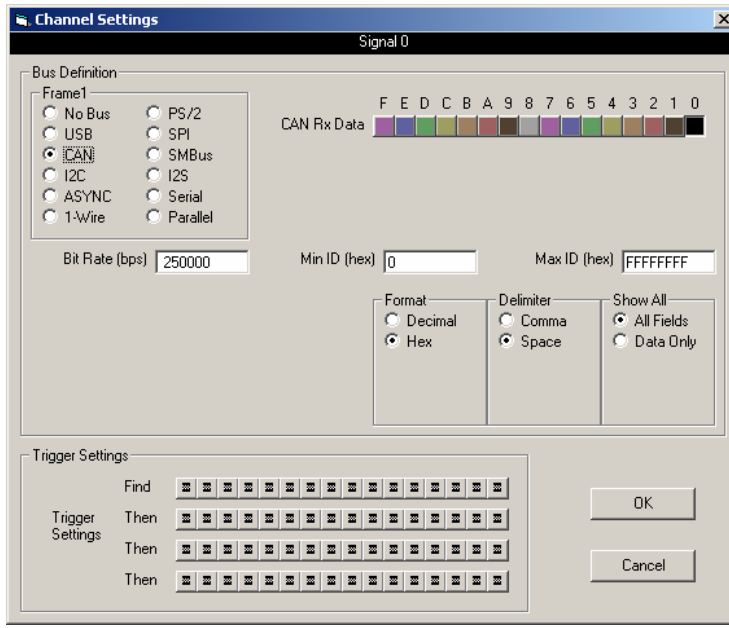
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The CAN Bus Decoder connects to the digital side of your CAN bus transceiver and only needs to listen to the receiving side of the transceiver (such as the RxD pin on the Microchip MCP2551 CAN bus transceiver chip). Use signal 0 as the RxD data line and connect the GND line to the digital ground of your system. Connect these signals to the CAN bus transceiver IC using the test clips provided.

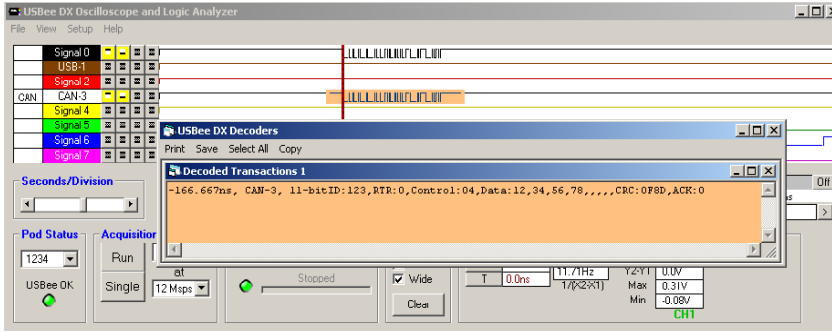
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the CAN data signal, what speed the bus is operating at, what filter value for the ID you want (if any), and what output format you want the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.6 USB Bus Setup

The USB Bus Decoder decodes Low and Full Speed USB. It does NOT decode High Speed USB. To decode Full Speed USB, the sample rate must be 24Msps, meaning you must sample with just 8 digital channels only. To decode Low Speed USB, you can sample as low as 3Msps.

Hardware Setup

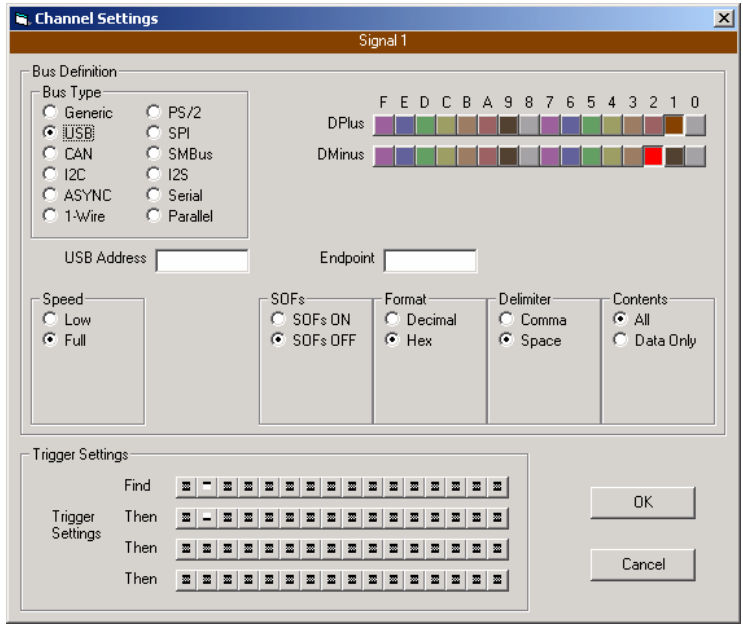
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

Connect two of the DX digital signals to the D+ and D- of your embedded USB bus, preferably at the IC of the USB device or the connector that the USB cable plugs into.

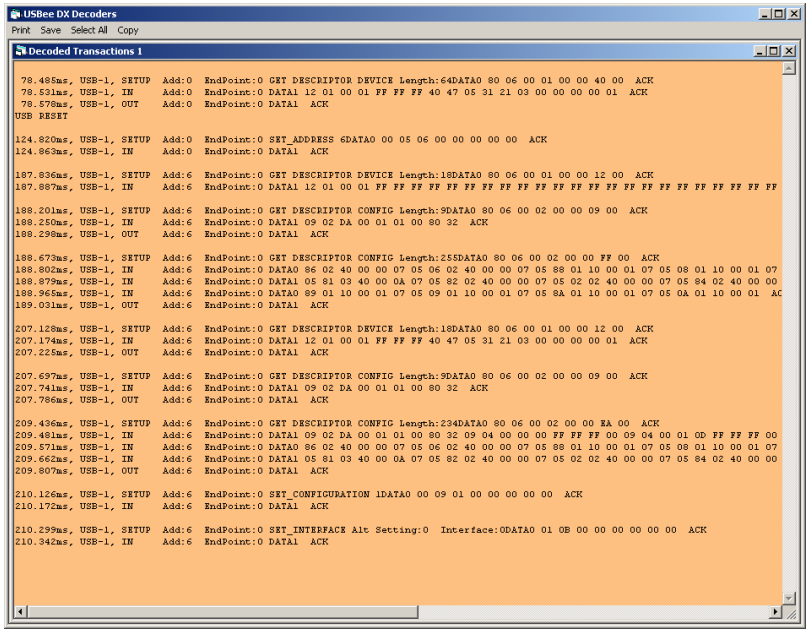
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the DPlus and DMinus signals, what speed the bus is operating at, if you want Start of Frames (SOF's) displayed, and what output format you want the traffic. You can also specify a specific USB Address or Endpoint you want to see. All other transactions will be filtered out. Leave the fields blank to see all transactions.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.7 I2C Bus Setup

The I2C Bus Decoder takes the captured data from a I2C bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

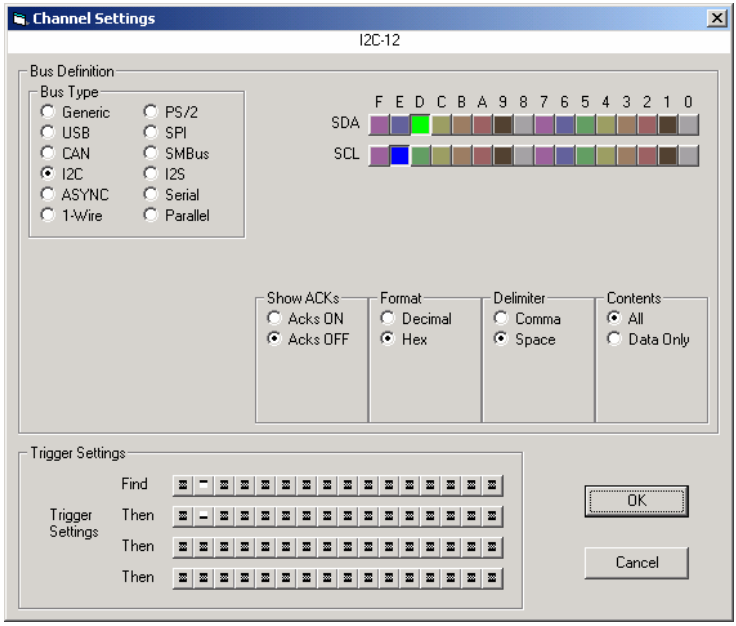
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The I^2C Bus Decoder connects to the SDA and SCL lines of the I^2C bus. Use one signal as the SDA data line and one signal as the SCL clock line. Also connect the GND line to the digital ground of your system. Connect these signals to the I^2C bus using the test clips provided.

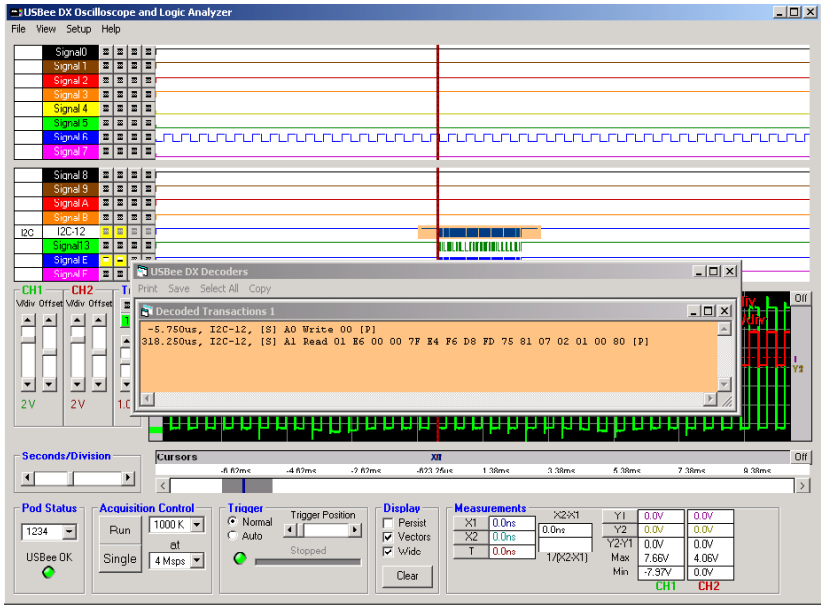
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the SDA and SCL signals, what portions of the transaction packet you want to see, and what output format you want the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.8 Async Bus Setup

The Async Bus Decoder takes the captured data from an asynchronous bus (UART), formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

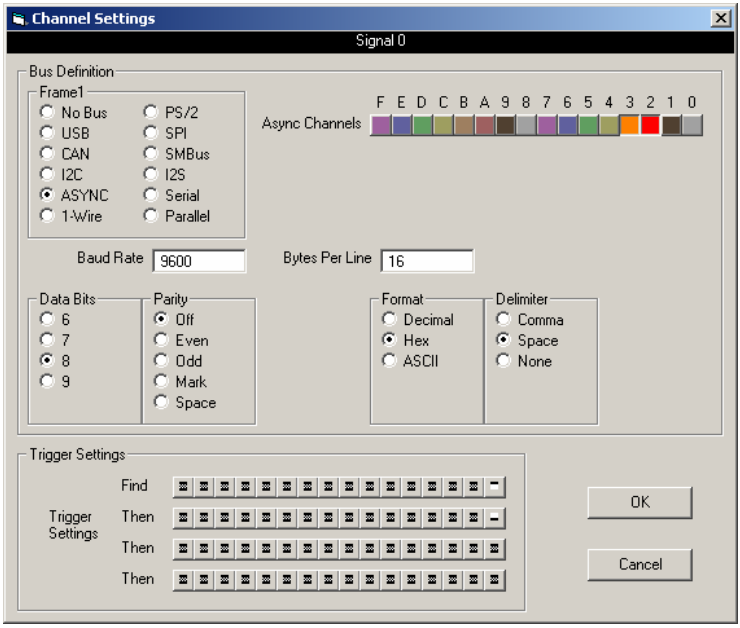
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The Async Bus Data Extractor uses one or more of the 16 digital signal lines (0 thru F) and the GND (ground) line. Connect any of the 16 signal lines to an Async data bus. Connect the GND line to the digital ground of your system.

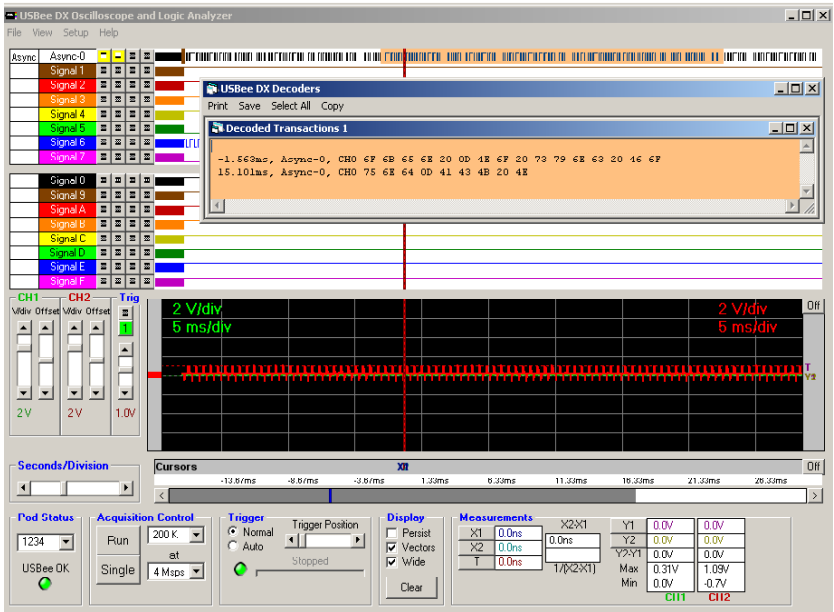
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the channels you want to observe. Each channel can be attached to a different async channel. Also enter the baud rate (from 1 to 24000000), how many bytes per line you want output, the number of data and parity bits, and what output format you want the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.9 Parallel Bus Setup

The Parallel Bus Decoder takes the captured data from a parallel bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste. The Parallel Bus decoder is also a way to capture the data using an external clock.

Hardware Setup

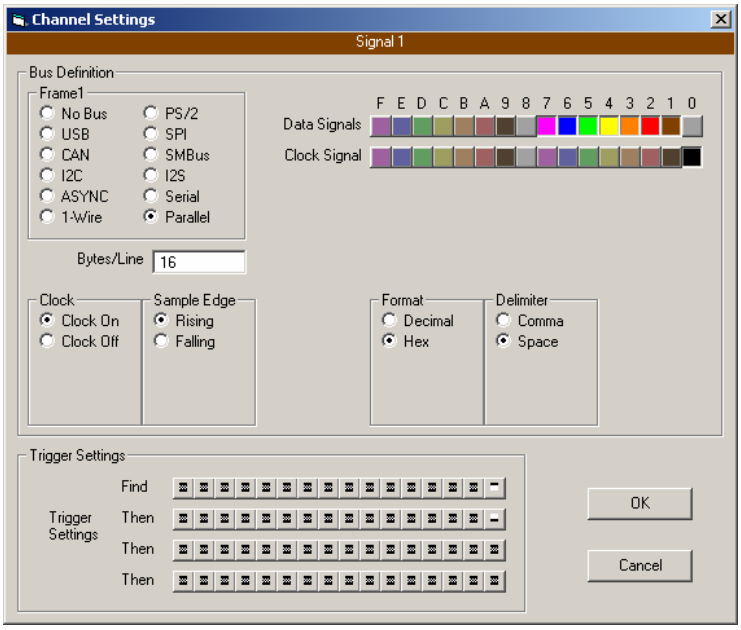
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The Parallel Bus Data Extractor uses the 16 digital signal lines (0 thru F), the GND (ground) line. Connect the GND line to the digital ground of your system.

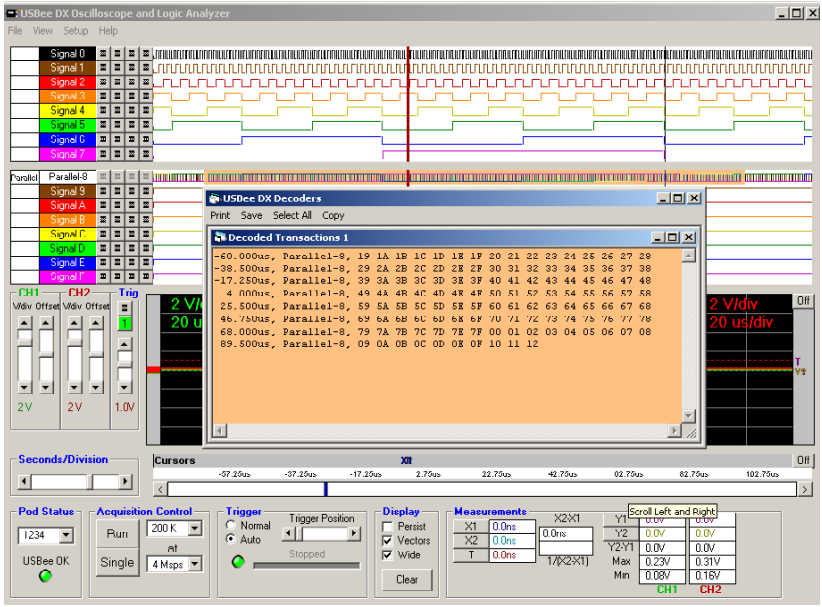
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the channels you want to include in the parallel data bus. You can also use any one of the 16 digital signals as an external clock. Choose if you want to use the external clock signal, the external clock edge polarity, how many bytes per line you want output, and what output format you want the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.10 1-Wire Bus Setup

The 1-Wire Bus Decoder takes the captured data from a 1-Wire bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

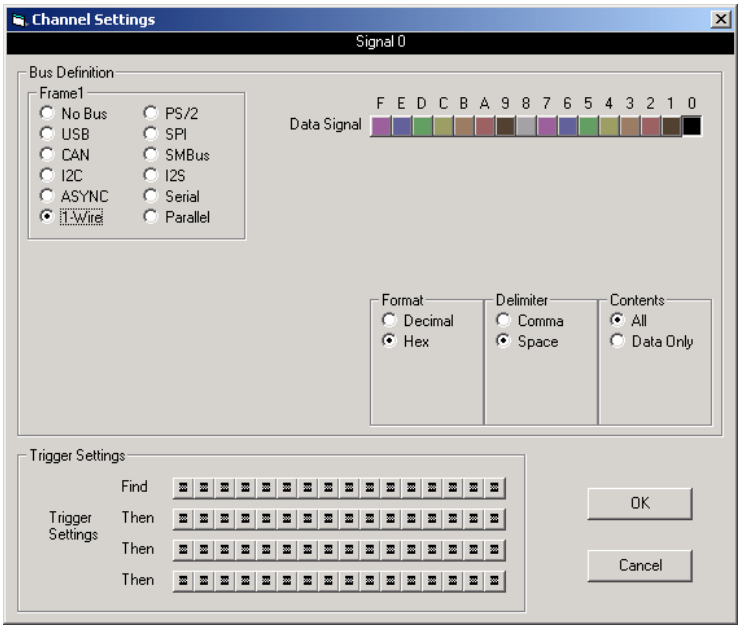
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The 1-Wire Bus Data Extractor uses any one of the 16 digital signal lines (0 thru F), the GND (ground) line. Connect the GND line to the digital ground of your system.

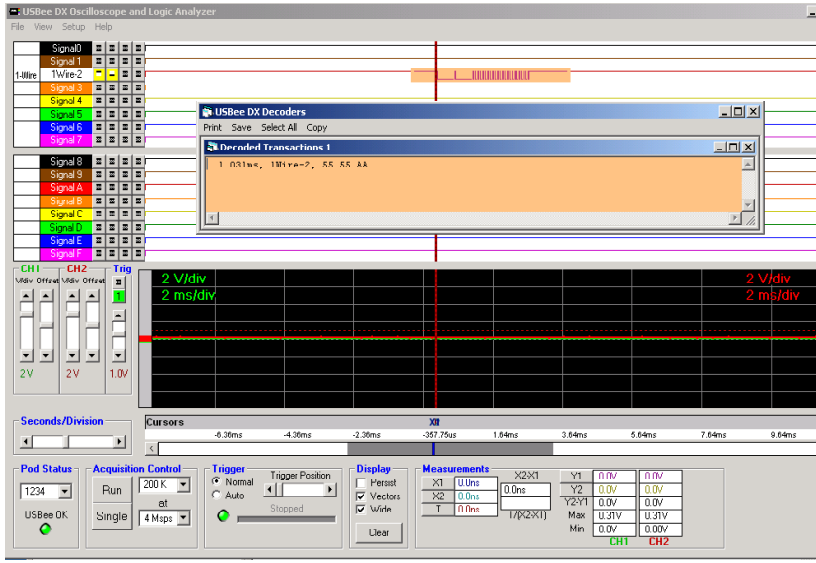
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signal running your 1-Wire protocol. Choose if you want to see just the data or all information on the bus and what output format you want the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.11 SPI Bus Setup

The SPI Bus Decoder takes the captured data from an SPI bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

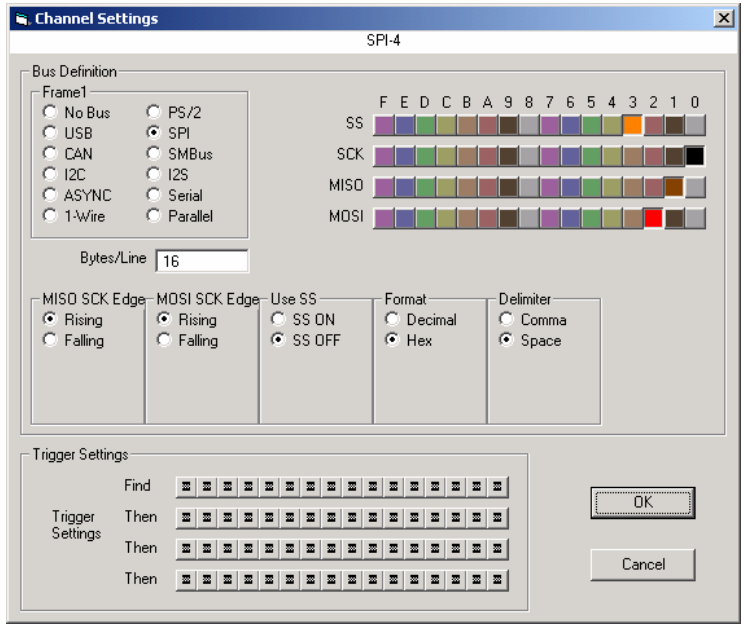
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The SPI Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the SS (slave select), SCK (clock), MISO (data in), MOSI (data out), and the GND (ground) line. Connect the SS, SCK, MISO, and MOSI to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

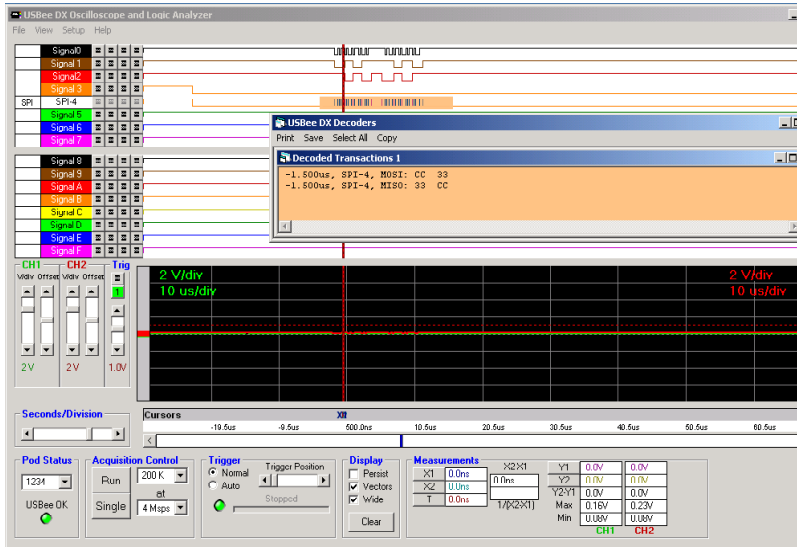
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the SPI protocol. Also set the appropriate sampling edges for both data lines and if you would like to use the SS (slave select) signal. If you turn off the SS, all clocks are considered valid data bits starting at the first clock detected. Also choose what output format you want the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.12 SM Bus Bus Setup

The SM Bus Decoder takes the captured data from an SM bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

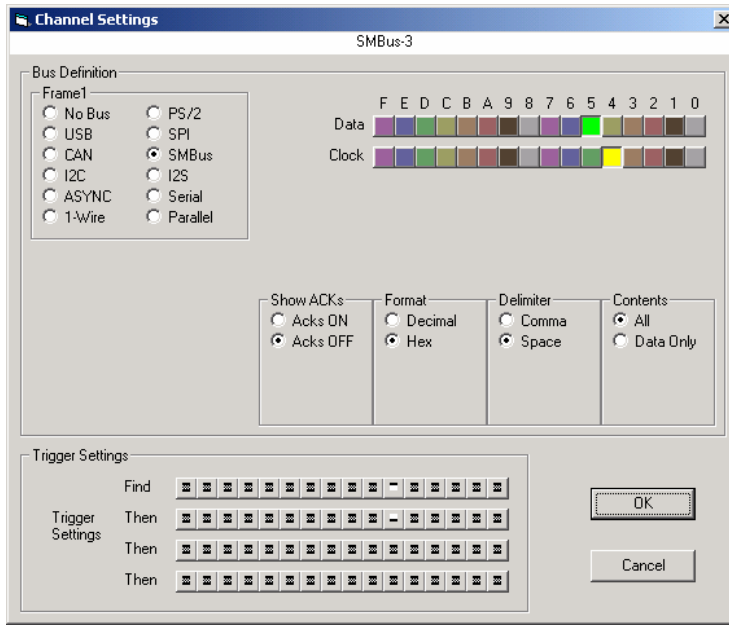
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The SM Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the SM Clock and SM Data, and the GND (ground) line. Connect the SM Clock and SM Data to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

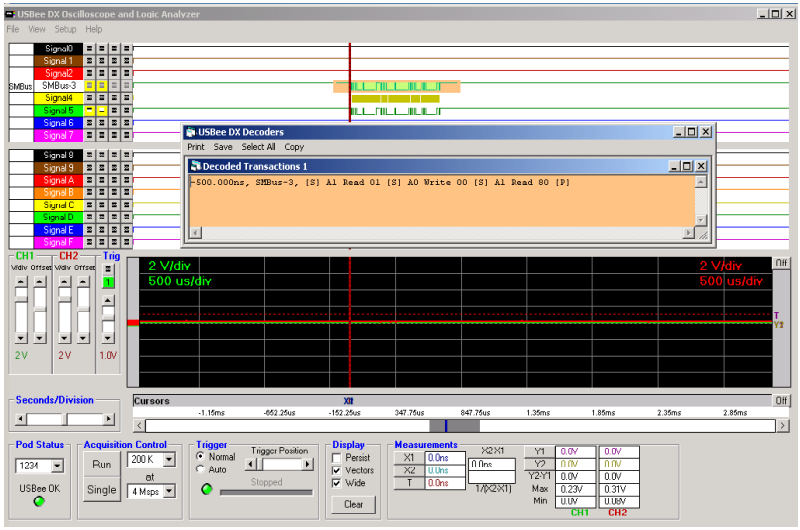
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the SM Bus protocol. Also choose what output format you want the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.13 Serial Bus Setup

The Serial Bus Decoder takes the captured data from a Serial bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste. The serial data can be from any clocked serial bus and can be aligned using a hardware signal or an embedded sync word.

Hardware Setup

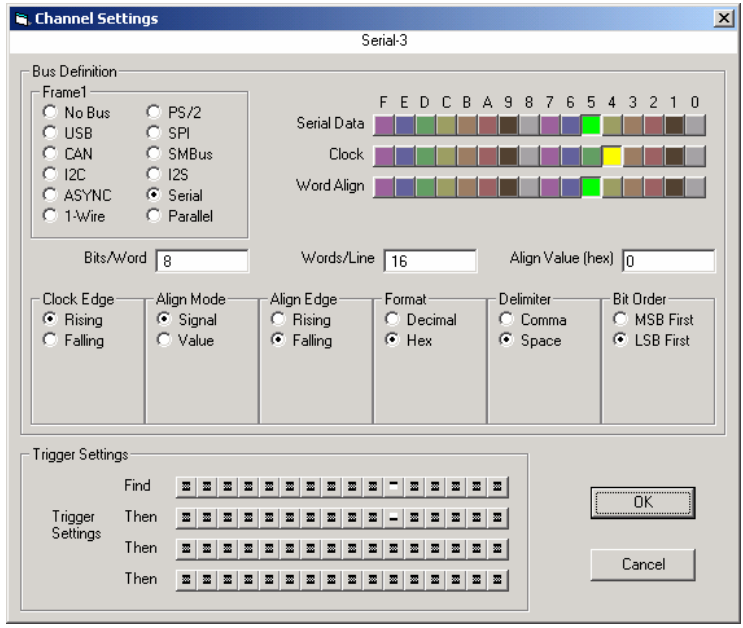
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The Serial Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the Clock, Data and optional Word Align signal, and the GND (ground) line. Connect the Clock, Data and Word Align to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

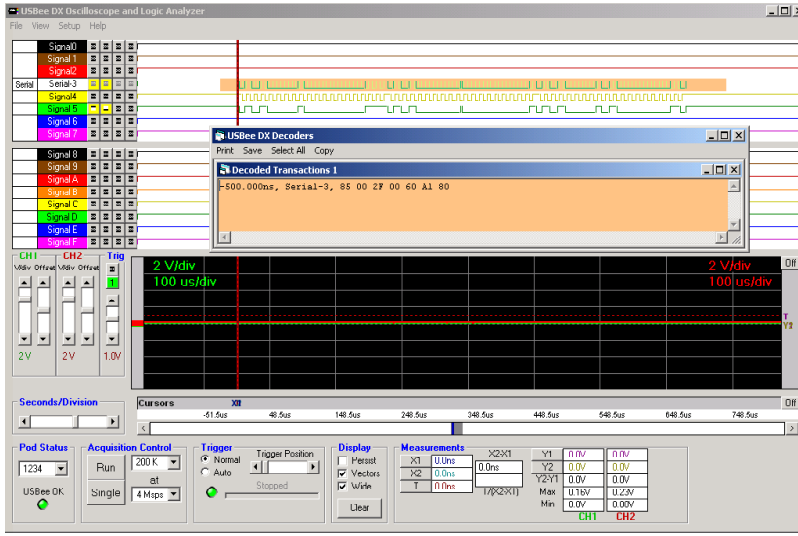
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the Serial Bus protocol. Select whether you have an external word align signal (Align Mode = Signal) or if your serial data has an embedded sync word in the data stream (Align Mode = Value). The Bits/Word is the size of the Sync word as well as the output word size. Choose the bit ordering as well as the output format of the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.14 I2S Bus Setup

The I2S Bus Decoder takes the captured data from an I2S bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

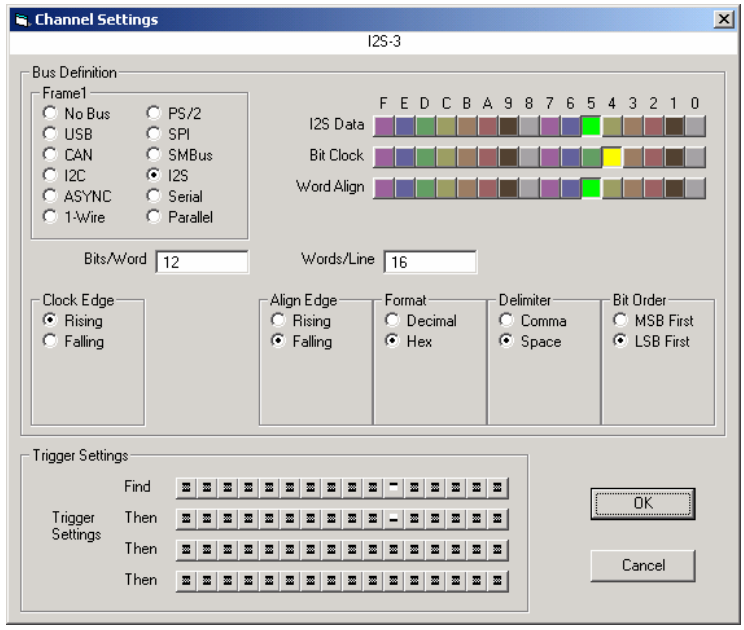
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The I2S Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the Clock, Data and Word Align signal, and the GND (ground) line. Connect the Clock, Data and Word Align to your digital bus using the test leads and clips. Connect the GND line to the digital ground of your system.

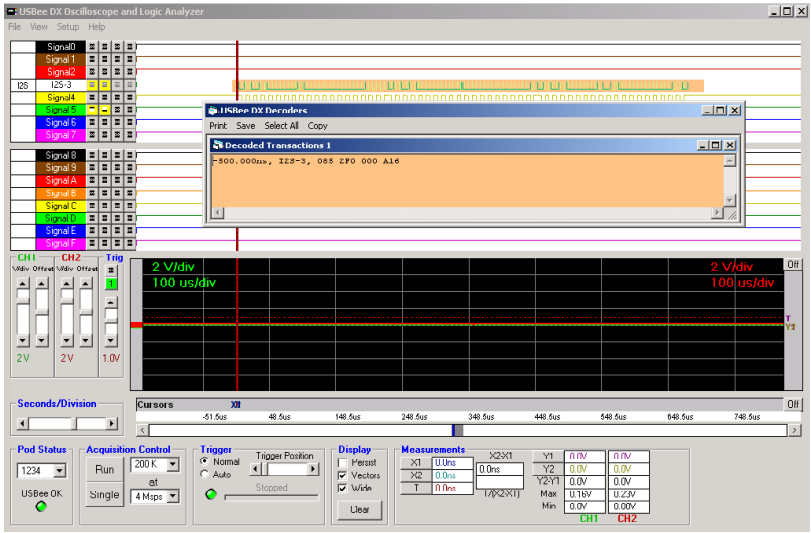
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the I2S Bus protocol. Select the start edge for the external word align signal, the Bits/Word and the Clock sampling edge. Choose the bit ordering as well as the output format of the traffic.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.



2.3.8.15 PS/2 Bus Setup

The PS/2 Bus Decoder takes the captured data from an PS/2 bus, formats it and allows you to save the data to disk or export it to another application using Cut and Paste.

Hardware Setup

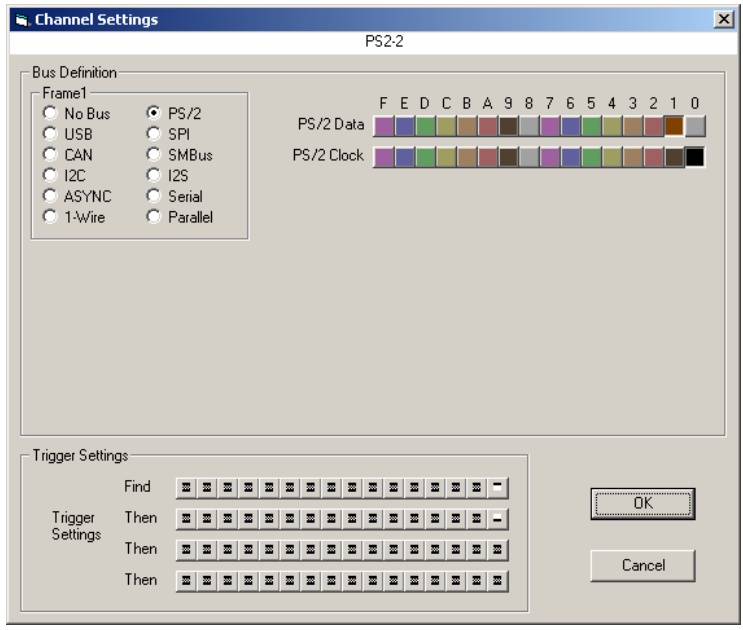
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee DX Test Pod digital inputs are strictly 0-5V levels. Any voltage outside this range on the signals will damage the pod and may damage your hardware. If your system uses different voltage levels, you must buffer the signals externally to the USBee DX Test Pod before connecting the signals to the unit.

The PS/2 Bus Decoder uses any one of the 16 digital signal lines (0 thru F) for the Clock and Data signals, and the GND (ground) line. Connect the Clock and Data to your PS/2 bus using the test leads and clips. Connect the GND line to the digital ground of your system.

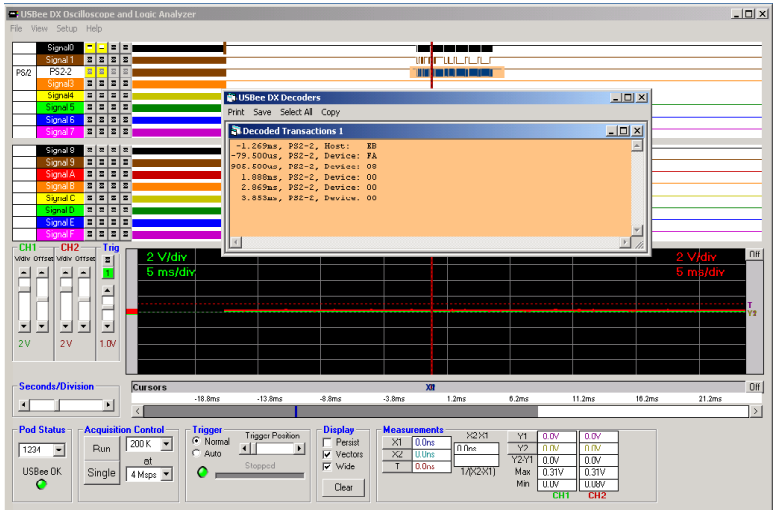
Software Setup

Activate the below Channel Settings Dialog by clicking the white box on the left of the signal names on the main application screen.



On the above dialog box, select the signals you plan to use for the PS/2 Bus protocol.

Then when you click and drag (with the right mouse button) on the waveform screen on that waveform, the bus traffic will be decoded as in the following screen.


USBee DX Oscilloscope and Logic Analyzer
File View Setup Help
USBee DX Decoders
Print Save Select All Copy
Decoded Transactions 1
-1.269ms, PS2-2, Host: KB
-79.500us, PS2-2, Device: FA
905.600us, PS2-2, Device: 00
1.886ms, PS2-2, Device: 00
2.869ms, PS2-2, Device: 00
3.853ms, PS2-2, Device: 00
2 V/div
5 ms/div
Seconds/Division
Cursors
Pod Status
USBee OK
Acquisition Control
Run
Single
Trigger
Normal
Auto
Stopped
Display
Persist
Vectors
Wide
Clear
Measurements

2.3.9 File Save, Open and Export

Using the File menu functions, you can save, open or export the current set of configuration and trace sample data.

Choose the menu item File | Save As to save the current configuration and sample data to a binary ULD file.

To load a previously saved waveform and display it, choose File | Open and specify the filename to load. This waveform will then be displayed as it was saved.

2.3.9.1 Output File Format

The following is the format of the saved files for the Logic Analyzer/ Oscilloscope application.

```
 String, "USBee DX Data File  " + Format(Date, "LONG DATE")
 String, "WaveHighlighted", WaveHighlighted

For x = 0 To 15
    String, "BusType" & Str(x), BusType(x)
    String, "Bus" & Str(x), Bus(x)
    String, "ShowVal" & Str(x), ShowVal(x)
    String, "HexVal" & Str(x), HexVal(x)
    String, "Delimiter" & Str(x), Delimiter(x)
    String, "ShowAll" & Str(x), ShowAll(x)
    String, "BytesPerLine" & Str(x), BytesPerLine(x)
    String, "Channels" & Str(x), Channels(x)
    String, "ClockSignal" & Str(x), ClockSignal(x)
    String, "UseClock" & Str(x), UseClock(x)
    String, "ClockEdge" & Str(x), ClockEdge(x)

    String, "SerialChannel" & Str(x), SerialChannel(x)
    String, "AlignValue" & Str(x), AlignValue(x)
    String, "AlignEdge" & Str(x), AlignEdge(x)
    String, "AlignChannel" & Str(x), AlignChannel(x)
    String, "UseAlignChannel" & Str(x), UseAlignChannel(x)
    String, "ClockChannel" & Str(x), ClockChannel(x)
    String, "BitsPerValue" & Str(x), BitsPerValue(x)
    String, "msbfirst" & Str(x), msbfirst(x)
    String, "DPlusSignal" & Str(x), DPlusSignal(x)
    String, "DMinusSignal" & Str(x), DMinusSignal(x)
    String, "USBSpeed" & Str(x), USBSpeed(x)
    String, "USBAddr" & Str(x), USBAddr(x)
    String, "USBEndpoint" & Str(x), USBEndpoint(x)
    String, "SOF" & Str(x), SOF(x)
    String, "SDASignal" & Str(x), SDASignal(x)
    String, "SCLSignal" & Str(x), SCLSignal(x)

    String, "ShowAck" & Str(x), ShowAck(x)
    String, "SSsignal" & Str(x), SSsignal(x)
    String, "SCKsignal" & Str(x), SCKsignal(x)
    String, "MOSISignal" & Str(x), MOSISignal(x)
    String, "MISOSignal" & Str(x), MISOSignal(x)
    String, "MISOEdge" & Str(x), MISOEdge(x)
    String, "MOSIEdge" & Str(x), MOSIEdge(x)
    String, "SSOn" & Str(x), SSOn(x)
    String, "CanSignal" & Str(x), CanSignal(x)
    String, "BitRate" & Str(x), BitRate(x)
    String, "MinID" & Str(x), MinID(x)
    String, "MaxID" & Str(x), MaxID(x)
    String, "OneWireSignal" & Str(x), OneWireSignal(x)

    String, "I2SWordSelectSignal" & Str(x), I2SWordSelectSignal(x)
    String, "I2SClkSignal" & Str(x), I2SClkSignal(x)
    String, "I2SDataSignal" & Str(x), I2SDataSignal(x)
    String, "ClkSignal" & Str(x), ClkSignal(x)
    String, "DataSignal" & Str(x), DataSignal(x)
    String, "AsyncSignal" & Str(x), AsyncSignal(x)
    String, "BaudRate" & Str(x), BaudRate(x)
    String, "DataBits" & Str(x), DataBits(x)
    String, "Parity" & Str(x), Parity(x)
    String, "ASCII" & Str(x), ASCII(x)
    String, "PS2DataSignal" & Str(x), PS2DataSignal(x)
```

```
    String, "PS2ClockSignal" & Str(x), PS2ClockSignal(x)
Next x

String, "TCenterSample", TCenterSample
String, "Infinite", Infinite
String, "TimelineMode", TimelineMode
String, "OffsetValue", OffsetValue
String, "OffsetValue", OffsetValue
String, "TimePerDiv", TimePerDiv
String, "MaxNumberOfSamples", MaxNumberOfSamples
String, "ActualNumberOfSamples", ActualNumberOfSamples
String, "TimeFlag", TimeFlag
String, "Rate", Rate
String, "MaxRate", MaxRate
String, "Captured", Captured
String, "TRIGValidSetting", TRIGValidSetting
String, "CLKEdgeSetting", CLKEdgeSetting
String, "TriggerOffset", TriggerOffset
String, "KnobValue2", KnobValue2
String, "NumberOfSections", NumberOfSections
String, "ScopeVoltsPerDiv", ScopeVoltsPerDiv
String, "TCenterSample", TCenterSample
String, "ScreenMax", ScreenMax
String, "ScreenMin", ScreenMin
String, "Initialized", Initialized
String, "NumberOfSamples", NumberOfSamples

For x = 0 To 255
    String, "TBuffer" & Str(x), TBuffer(x)
Next x

For x = 0 To 15
    For Y = 0 To 3
        String, "TriggerSetting" & Str(x) & "-" & Str(Y), TriggerSetting(Y, x)
        String, "Trigg" & Str(x) & "-" & Str(Y), Trigg(Y, x)
    Next Y
Next x

String, "TriggerStates", TriggerStates
String, "ScaleP", ScaleP
String, "TOCursor", TOCursor
String, "TCurrentCursor", TCurrentCursor
String, "TXCursor", TXCursor
String, "TY1Cursor", TY1Cursor
String, "TY2Cursor", TY2Cursor
String, "TScale", TScale
String, "TSubScale", TSubScale
String, "TStartingSample", TStartingSample
String, "TCenterSample", TCenterSample
String, "CalibrationSlope", CalibrationSlope
String, "Scope1GroundCalibrationLevel", Scope1GroundCalibrationLevel
String, "Scope1DisplayCenterVolts", Scope1DisplayCenterVolts
String, "Scope1TriggerLevel", Scope1TriggerLevel
String, "Scope1TriggerSlope", Scope1TriggerSlope
String, "VoltsPerPixel", VoltsPerPixel
String, "NumberOfDiv", NumberOfDiv
String, "AnalogWaveIndex", AnalogWaveIndex
String, "DigitalHighOn", DigitalHighOn
String, "DigitalLowOn", DigitalLowOn
String, "AnalogHighOn", AnalogHighOn
String, "AnalogLowOn", AnalogLowOn

For x = 0 To 16
    String, "SigColor" & Str(x), SigColor(x)
    String, "SigBackColor" & Str(x), SigBackColor(x)
    String, "SigForeColor" & Str(x), SigForeColor(x)
Next x

String, "AnalogColor", AnalogColor
String, "XCursorsOn", XCursorsOn
String, "YCursorsOn", YCursorsOn

For x = 0 To 15
    For Y = 0 To 15
        String, "SignalsInWave" & Str(x) & "-" & Str(Y), SignalsInWave(Y, x)
    Next Y
Next x

String, "GlobalCalValue", GlobalCalValue

For x = 0 To 15
    String, "SignalLabel" & Str(x), Form1.SignalLabel(x).Caption
Next x

String, "A0D8", Form1.A0D8.Checked
String, "A0D16", Form1.A0D16.Checked
String, "A1D0", Form1.A1D0.Checked
String, "A1D8", Form1.A1D8.Checked
```

```
    String, "A1D16", Form1.A1D16.Checked
    String, "A2D0", Form1.A2D0.Checked
    String, "A2D8", Form1.A2D8.Checked
    String, "A2D16", Form1.A2D16.Checked

    String, "Ch1V", Form1.Ch1V.Value
    String, "CH2V", Form1.CH2V.Value
    String, "Ch1Offset", Form1.Ch1Offset.Value
    String, "Ch2Offset", Form1.Ch2Offset.Value
    String, "VScroll1", Form1.VScroll1.Value
    String, "HScroll1", Form1.HScroll1.Value
    String, "SizeList", Form1.SizeList.ListIndex
    String, "RateList", Form1.RateList.ListIndex

    String, "NormalMode", Form1.NormalMode.Value
    String, "AutoMode", Form1.AutoMode.Value

    String, "TriggerPositionScroll", Form1.TriggerPositionScroll.Value

    String, "Persist", Form1.Persist.Value
    String, "Vectors", Form1.Vectors.Value
    String, "Wide", Form1.Wide.Value

    String, "ScaleP", Form1.ScaleP.Text
    String, "SubScale", Form1.SubScale.Text

    ' The sample data follows this last record
    String "[Samples]"

    Sample Buffer: NumberOfSamples times 4 byte samples.  The low 16 bits are the digital
channels.  The high 2 bytes are the 8-bit ADC values for each of the two analog channels.
```

2.3.9.2 Export to Text Format

You can also export a specific portion of the sample data by placing the X1 and X2 cursors. When you choose File | Export to Text the samples between the X1 and X2 cursors will be written to a file in comma delimited text format as below.

The format of the text output file is a header that specifies Digital0-F, CH1, and CH2 titles. The following lines are the actual values of the 16 digital lines in hex format, and the CH1 and CH2 voltage level in volts.

```
Digital0-F, CH1, CH2
0xFF0F, -0.16,  3.67
0xFF0F, -0.08,  3.75
0xFF0F, -0.16,  3.67
0xFF0F, -0.08,  3.75
0xFF0F, -0.08,  3.67
0xFF0F, -0.08,  3.75
0xFF0F, -0.08,  3.67
0xFF0F,  0.00,  3.75
0xFF0F,  0.00,  3.75
…
```

2.3.10 Calibration

Since electronic components vary values slightly over time and temperature, the USBee DX Pod requires calibration periodically to maintain accuracy. The USBee DX has been calibrated during manufacturing and should maintain accuracy for a long time, but in case you want to recalibrate the device, follow these steps. The calibration values are stored inside the USBee DX pod. Without calibration the measurements of the oscilloscope may not be accurate as the pod ages.

To calibrate your USBee DX Pod you will need the following equipment:

- External Voltage Source (between 5V and 9V)
- High Precision Multimeter

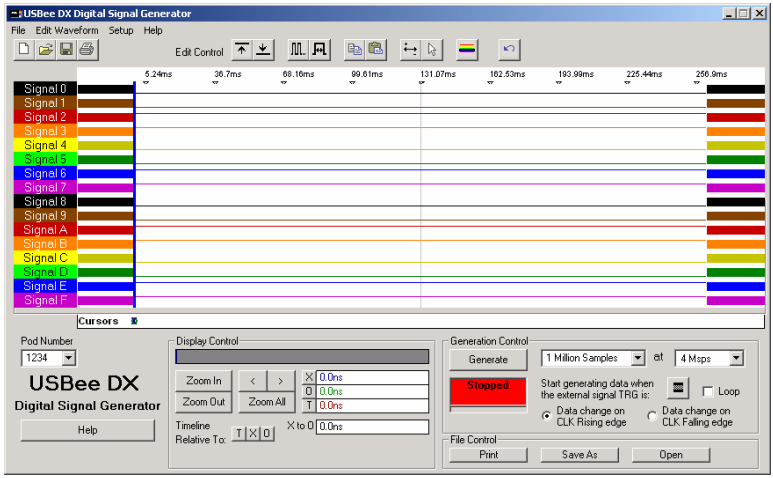
When you are ready to calibrate the USBee DX Pod, go to the menu item Setup | Calibrate. You will be asked to confirm that you really want to do the calibration. If so, press Yes, otherwise press No. Then follow these steps:

- Connect the CH1 and CH2 signals to the GND signal using the test leads and press OK. A measurement will be taken.
- Connect the GND signal to the ground and the CH1 and CH2 signals to the positive connection of the External Voltage Source (9V) using the test leads.
- With the Multimeter, measure the actual voltage between the GND signal and the CH1 signal and enter this value in the dialog box and press OK. A measurement will be taken.
- The calibration is now complete. The calibration values have been saved inside the pod.

The analog measurements of your USBee DX pod are only as accurate as the voltages supplied and measured during calibration.

3 Digital Signal Generator

This section details the operation of the Digital Signal Generator application that comes with the USBee DX. Below you see the application screen.



The Digital Signal Generator is used to actively drive the 16 digital signals with a voltage pattern that you define.

When using this application, the USBee DX signals 0 through F are actively driven. Do not connect these signals to your circuit if your circuit is also driving the signals or you will damage the USBee or your circuit or both.

To define the pattern that you want to generate, you will use the waveform screen and draw the timing of pulses that you require.

3.1 Digital Signal Generator Specifications

Digital Output Channels	16 or 8
Maximum Digital Sample Rate [1]	24 Msps for 8 channels, 12Msps for 16 channels
Internal Clocking	Yes
External Clocking	No

Number of Samples [2]	1 million samples up to PC RAM
Sample Rates [1]	1Msps to 24 Msps
Sample Clock Output	Yes
Channel Output Drive Current	4mA
Output Low Level	< 0.8V
Output High Level	> 2.4V
Looping	Yes
External Trigger Signal	Yes

3.2 Quick Start

In order to quickly get up and running using this application, here is a step by step list of the things you need to do to generate a set of digital waveforms.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect any of the Signal 0 thru F pins on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to your circuit you would like to actively drive.
- Run the Signal Generator Application.
- Draw a waveform you want to generate using the waveform edit controls at the top of the waveform window.
- Press the Generate button. This will generate the waveform you have drawn on the pod signals.

3.3 Features

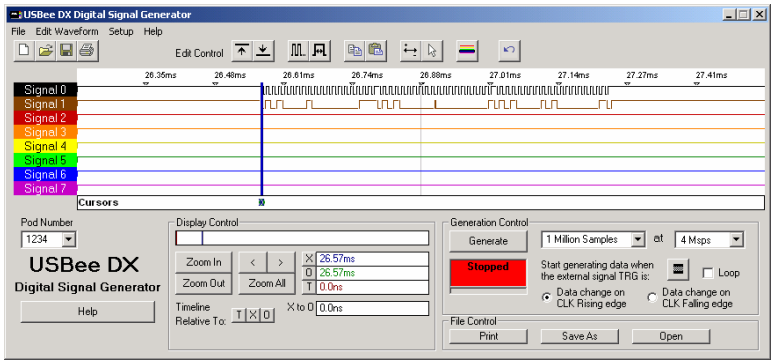
3.3.1 Pod Status

The Signal Generator display shows a list with the available **Pod ID List** for all of the USBee DX's that are connected to your PC. You can choose which one you want to use. The others will be unaffected. If a USBee DX is not connected, the list box will read Demo to indicate that there is no pod attached.

If you run the software with no pod attached, it will run in demonstration mode so that you can still see how the software functions.

3.3.2 Channel Setup

The Signal Generator operates in either an 8-channel or 16-channel mode. Select which mode you want to use by clicking the menu item Setup, 8 (or 16) Channels. Below you see the 8 Channel mode.



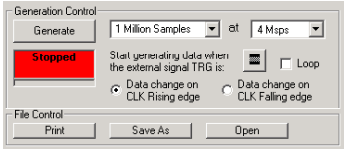
The maximum sample rate that your system can achieve varies depending on the number of channels you select.

For 8 Channel mode, the maximum sample rate is 24M samples per second.

For 16 Channel mode, the maximum sample rate is 12M samples per second.

3.3.3 Generation Control

The Signal Generator lets you draw the behavior of digital signals and then generates them as a “trace” on the pod signals. The Generation Control section of the display lets you choose how the traces are generated. Below is the Generation Control section of the display.



The **Generate** button starts and stops a data output. When the signal generator is first started, the Generate button is not pressed and is waiting for you to draw a waveform. The Generate button outputs a

single trace and stops, unless you check the **Loop** box. If the Loop checkbox is checked, the wave is played until the end and then restarted at the beginning sample without breaks in between the first and second trace.

The **Buffer Size** lets you select the size of the Sample Buffer that is used. For each trace, the buffer is completely played back. No partial buffers can be generated. You can choose buffers that will hold the information that you want to output, but remember that the larger the buffer, the longer it will take to generate.

You can also choose the **Sample Rate** that you want samples to be aligned to. This uses an internal clock at that sample rate you choose. You can choose from 1 Msps (samples per second) to up to 24 Msps. The actual maximum sample rate depends on your PC configuration. If the sample rate is too high for your system, you will see a dialog box appear when you generate the waveform that informs you that the rate is too high. You must lower the sample rate and try again.

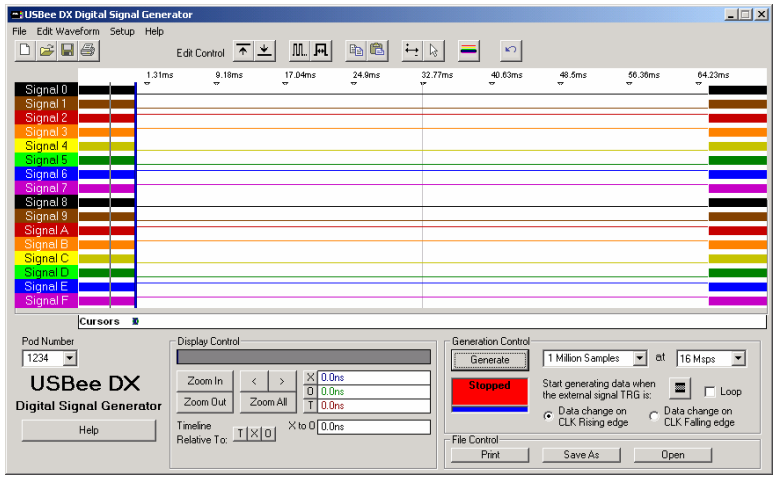
While the pod is generating the waveform on the pod signals, the CLK line is an output and toggles once for each of the samples provided. You can specify the **CLK Edge** that the output data changes on using the two radio buttons above.

The TRG signal can be used as an **External Trigger** for the pattern generation. Select the state of the TRG signal you want to start the output on by pressing the toggle pushbutton above.

The **Status Box** on the display will show red when the unit is not outputting samples, flash blue when it is waiting for a trigger, and glow green when the trigger condition has been met. It will glow red again when the generation is completed.

3.3.4 Waveform Edit, Display and Zoom Settings

The Waveform display area is where the signal information is shown. It is displayed with time increasing from left to right and voltage increasing from bottom to top. The screen is divided into **Divisions** to help in measuring the waveforms.



To **Scroll the Waveforms in Time** left and right, you can use the left and right arrows highlighted above, click and drag the Overview Bar (right under the Display Control title), or you can simply click and drag the waveform itself.

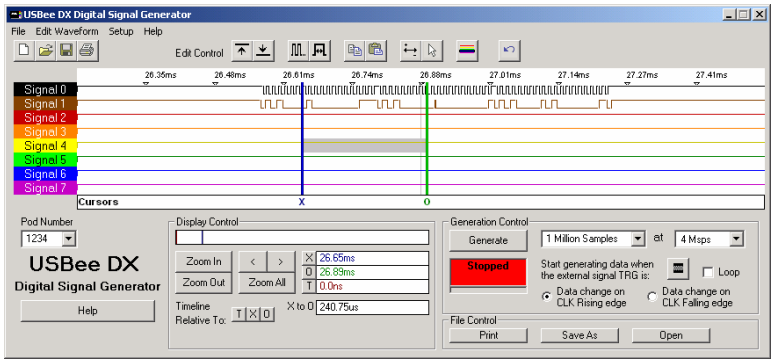
To change the zoom ratio for the time, click the **Zoom In** or **Zoom Out** buttons. You can also zoom in and out in time by clicking on the waveform. To zoom in, click the left mouse on the waveform window. To zoom out in time, click the right mouse button on the waveform window.

The cursor in the waveform window can be in one of two modes: **Pan and Zoom**, or **Select**. In pan and zoom, you can click and drag the waveform around on the screen. In Select, you click and drag to select a portion of the waveform to edit. Change modes by clicking the left-right arrow (pan and zoom), or the standard arrow (select).

Editing the Waveform is done by selecting the portion of the waveform by clicking and dragging to highlight a section, and then pressing one of the Edit Control buttons at the top. You can set the specified samples to a high level, low level, create a clock on that signal, create a single pulse, or copy and paste. You can also **Undo** the last change if needed.

3.3.4.1 Setting Waveform Sections

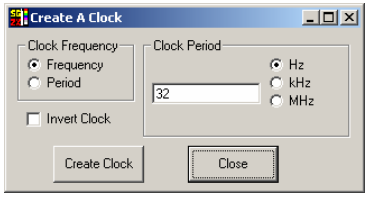
To create a waveform you need to scroll or zoom to the section of wave you want to change. Then change the cursor to an arrow by pressing the arrow button at the top.



Then select a section of a wave by using the left mouse button with a click and drag. Once the selection is highlighted you can press the High or Low button to set that section to the desired level.

3.3.4.2 Creating Clocks

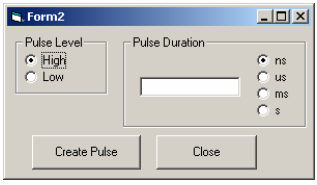
To create a clock on a given signal you first select the wave you want to set. Then click the Clock button at the top of the waveforms to get the following dialog box.



Select the period or the frequency that you would like and press Create Clock. Your selected channel will then be replaced by a clock with that frequency.

3.3.4.3 Creating Pulses

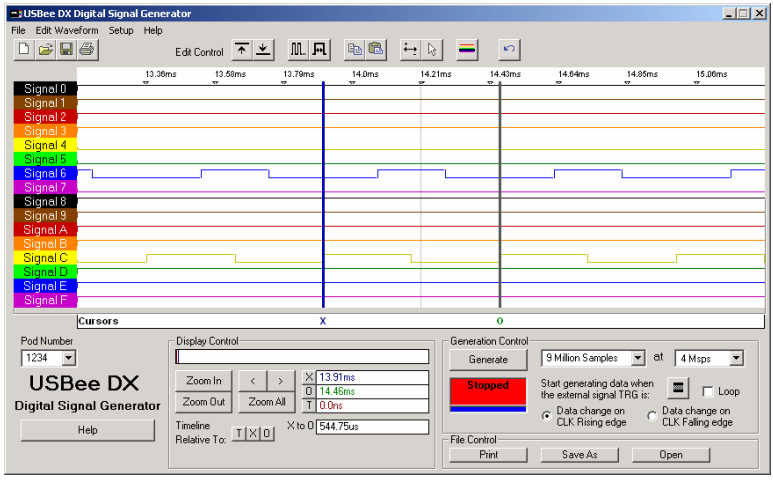
To create a series of pulses with known duration on a given signal you first select the wave you want to set. Then click the Pulses button at the top of the waveforms to get the following dialog box.



Set the duration time and voltage level and press Create Pulse. You can then create consecutive pulses just by entering the new duration and pressing the button again.

3.3.5 Measurements and Cursors

To help you create time accurate waveforms, the cursors can be used to get exact timing.



The **X and O Cursors** are placed on any horizontal sample time. This lets you measure the time at a specific location or the time between the two cursors. To place the X and O cursors, move the mouse to the white box just below the waveform. When you move the mouse in this window, you will see a temporary line that indicates where the cursors will be placed. Place the X cursor by left clicking the mouse at the current location. Place the O cursor by right clicking the mouse at the current location.

In the Measurement window, you will see the various measurements made off of these cursors. To change the selected relative cursor, click the T,X or O buttons next to the "Timeline Relative To" text.

- **X Position** – time at the X1 cursor relative to the selected cursor
- **O Position** – time at the X2 cursor relative to the selected cursor
- **X to O -** difference between X and O cursors

3.3.6 File Save and Open

Using the File menu functions, you can save and open a current set of configuration and trace sample data.

Choose the menu item File | Save As to save the current configuration and sample data to a binary ULC file.

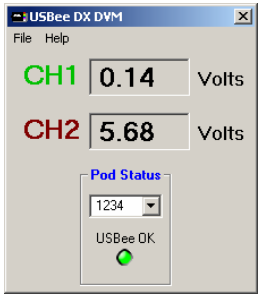
To load a previously saved waveform and display it, choose File | Open and specify the filename to load. This waveform will then be displayed as it was saved. If the loaded file is smaller than the current buffer size, the file will be loaded at the beginning of the current buffer. The ending samples in the buffer remain unchanged. If you load a file with more samples than the current buffer, the loaded samples will be truncated.

3.3.7 Printing

You can print the current screen to any printer by choosing the File | Print menu item.

4 Digital Voltmeter (DVM)

This section details the operation of the Digital Voltmeter (DVM) application that comes with the USBee DX. Below you see the application screen.



4.1 Digital Voltmeter Specifications

Analog Channels Displayed	2
Analog Input Voltage Range	-10V to +10V
Minimum Measurable Resolution	78mV
Analog Resolution	256 steps
Update Rate	3 samples per second

4.2 Quick Start

In order to quickly get up and running using this application, here is a step by step list of the things you need to do to measure two analog voltages.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the CH1 pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire. Connect the other end of the wire to your circuit you would like to test.

- Connect the CH2 pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire. Connect the other end of the wire to your circuit you would like to test.
- Run the DVM Application.
- The voltages of the CH1 and CH2 signal will be displayed and updated about three times per second.

4.3 Features

4.3.1 Pod Status

The DVM display shows a current USBee DX **Pod Status** by a red or green LED. When a USBee DX is connected to the computer, the Green LED shows and the list box shows the available **Pod ID List** for all of the USBee DX's that are connected. You can choose which one you want to use. The others will be unaffected. If a USBee DX is not connected, the LED will glow red and indicate that there is no pod attached.

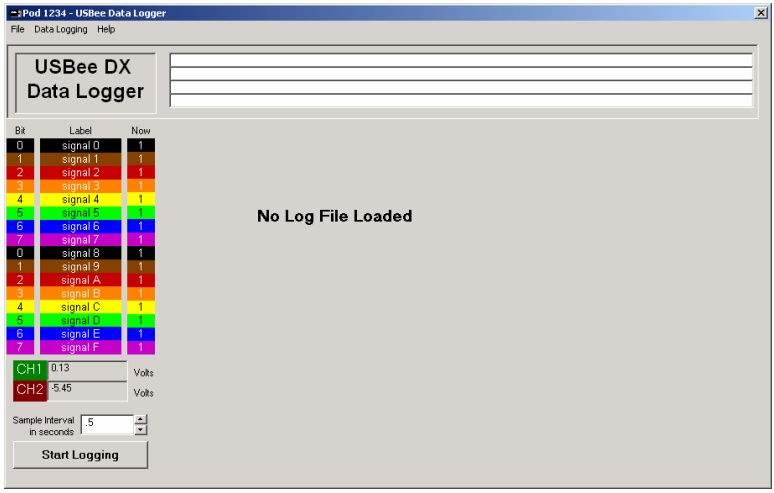
If you run the software with no pod attached, it will run in demonstration mode and simulate data so that you can still see how the software functions.

4.3.2 Voltage Measurement

The DVM takes a 250 msec measurement of each of the channels and displays the average voltage over that time period. Although the resolution of each individual sample is 78.125mV, the averaged values are far more accurate.

5 Data Logger

This section details the operation of the Data Logger application that comes with the USBee DX. Below you see the application screen.



5.1 Data Logger Specifications

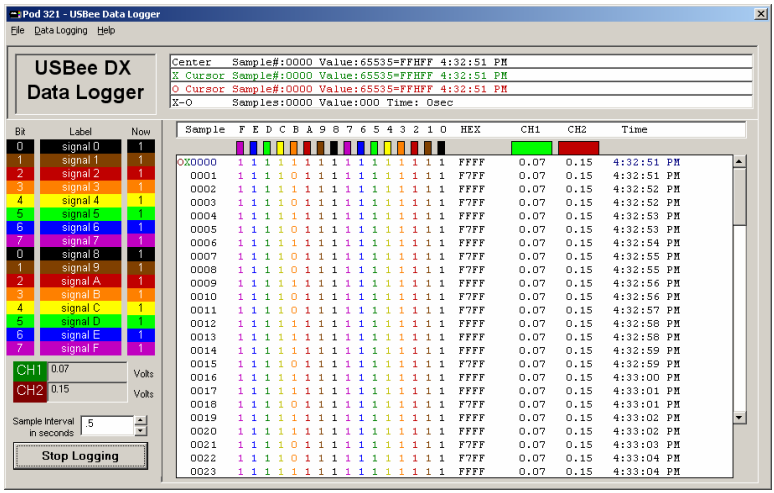
Digital Channels Logged	16
Analog Channels Logged	2
Sample Rates	500ms to 300sec

5.2 Quick Start

In order to quickly get up and running using this application, here is a step by step list of the things you need to do to log analog and digital data.

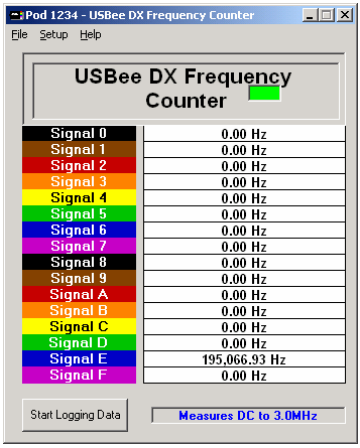
- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the CH1 and/or CH2 pins on the USBee DX pod to one of the signal wires you would like to test.

- Connect the digital Signal 0 thru F pins on the USBee DX pod to one of the signal wires you would like to test.
- Run the Data Logger Application.
- Select the sample time and press the Start Logging button. Select the filename for the logged data to be exported to and press OK.
- This will start the logging process. Data will be displayed as it is logged. When you are finished, press the Stop Logging button.
- The data is then displayed in the list format for review. You can also post process the text based log file using other programs.



6 Frequency Counter

This section details the operation of the Frequency Counter application that comes with the USBee DX. Below you see the application screen.



6.1 Frequency Counter Specifications

Digital Channels Measured	8 or 16
Analog Channels Measured	0
Maximum Measured Frequency [1]	12MHz (8-channel) or 6MHz (16-channel)
Maximum Digital Input Voltage	+5.5V
Resolution	1Hz
Gate Time	1 sec

6.2 Quick Start

In order to quickly get up and running using this application, here is a step by step list of the things you need to do to measure the frequency of a digital signal.

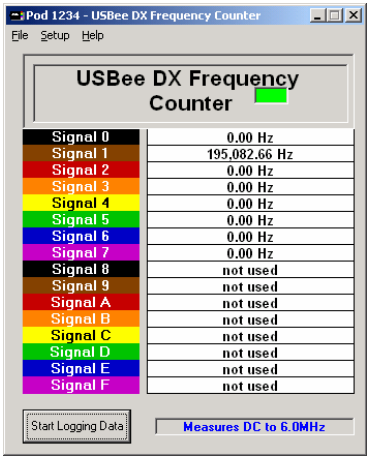
- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.

- Connect the Signal 0 thru F signals on the USBee DX pod to your circuit you would like to test.
- Run the Frequency Counter Application.
- The frequency of each of the 16 signal lines will then be displayed.
- You can log the frequency data to a file by pressing the “Start Logging Data” button.

6.3 Channel Setup

The Frequency Counter can operate on either 8 channels or 16 channels at a time. For 8 channels, the maximum frequency measured is 12MHz. For 16 channels, the maximum frequency measured is 6MHz.

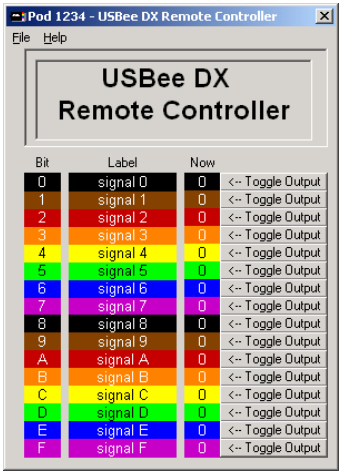
Change setup modes by clicking the menu item Setup and selecting the desired number of channels. Below shows the 8 channel setup mode.



7 Remote Controller

This section details the operation of the Remote Controller application that comes with the USBee DX. The Remote Controller application is a simple way to control the output settings for all of the 16 digital lines on the USBee DX. Since this application drives the digital signals, you will see a warning message alerting you to this fact before the lines are driven.

Click OK to enter the application. Below you see the application screen.



To change the digital output, simply press the Toggle Output button to change the output from a 1 to 0 or visa versa.

7.1 Remote Controller Specifications

Digital Channels Controlled	16
Analog Channels Controlled	0
Control Mechanism	Toggle Button per channel
Channel Output Drive Current	4mA

Output Low Level	< 0.8V
Output High Level	> 2.4V

7.2 Quick Start

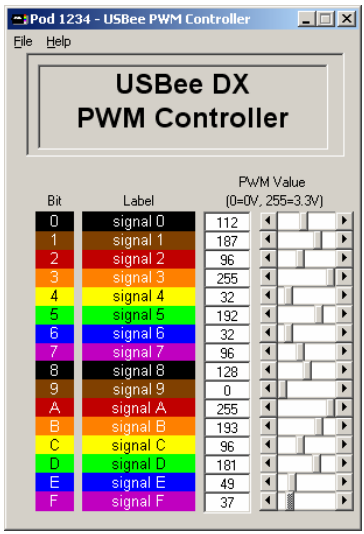
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to control the output of each of the digital signal lines.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the Signal 0 thru F lines on the USBee DX pod to your circuit you would like to actively drive.
- Run the Remote Controller Application.
- Press any of the Toggle buttons and the level of the output will toggle (Low to High, High to Low)..

8 PWM Controller

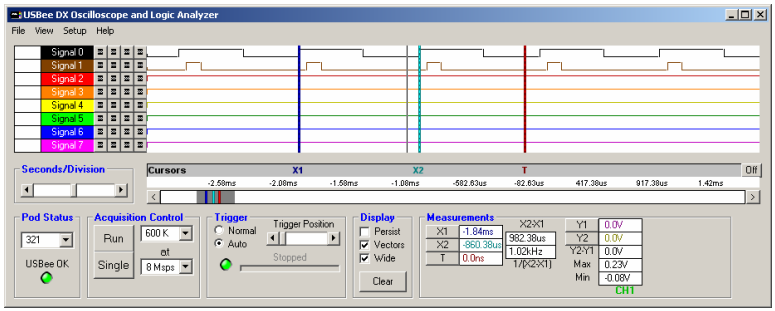
This section details the operation of the Pulse Width Modulator application that comes with the USBee DX. The Pulse Width Modulator application creates a Pulse Width Modulated output for all of the 16 digital lines on the USBee DX. Since this application drives the digital signals, you will see a warning message alerting you to this fact before the lines are driven.

Click OK to enter the application. Below you see the application screen.



Each channel outputs a repeating waveform with a 1kHz frequency. The period of the repeating waveform is made up of a high duration followed by a low duration and has 256 steps. The length of the High duration is the PWM value that is shown. The length of the Low duration is 256 – the High duration.

You can create a simple analog output voltage by using a series resistor and a capacitor to ground on each channel.



The above shows 2 outputs of the PWM Controller. Signal 1 shows the PWM value set to 31 (out of 255) and Signal 0 shows the PWM value of 137. A value of 0 is all low, and a value of 255 is mostly high (one out of 256 is low).

8.1 PWM Controller Specifications

Digital Channels Controlled	16
Analog Channels Controlled	0
Resolution	256 steps
PWM Frequency	1.02kHz
Control Mechanism	Slider Switch
Channel Output Drive Current	4mA
Output Low Level	< 0.8V
Output High Level	> 2.4V

8.2 Quick Start

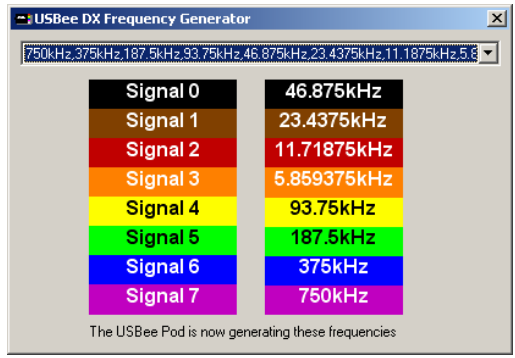
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to create 16 PWM signals.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the Signal 0 thru F lines on the USBee DX pod to your circuit you would like to actively drive with a PWM signal.
- Run the PWM Controller Application.
- Use the scroll bars to set the desired PWM level, with 0 being all low and 255 being all high outputs.

9 Frequency Generator

This section details the operation of the Frequency Generator application that comes with the USBee DX. The Frequency Generator is used to generate a set of commonly used digital frequencies on the low 8 digital channels.

Below you see the application screen.



To set the frequencies generated, use the drop down list box to choose which subset you would like to generate. Then refer to the screen for which signal is generating which frequency.

9.1 Frequency Generator Specifications

Digital Channels Controlled	8
Analog Channels Controlled	0
Sets of Frequencies	6
Set 1	1MHz, 500kHz, 250kHz, 62.5kHz,31.25kHz, 15.625kHz, 7.8125kHz
Set 2	32kHz, 16kHz, 8kHz, 4kHz, 2kHz, 1kHz, 500Hz, 250Hz
Set 3	750kHz, 375kHz, 187.5kHz, 93.75kHz, 46.875kHz, 23.4375kHz, 11.1875kHz, 5.5893kHz
Set 4	19.2kHz, 9600Hz, 4800Hz,

	2400Hz, 1200Hz, 600Hz, 300Hz, 150Hz
Set 5	64Hz, 32Hz, 16Hz, 8Hz, 4Hz, 2Hz, 1Hz, 0.5Hz
Set 6	1920Hz, 960Hz, 480Hz, 240Hz, 120Hz, 60Hz, 30Hz, 15Hz
Channel Output Drive Current	4mA
Output Low Level	< 0.8V
Output High Level	> 2.4V

9.2 Quick Start

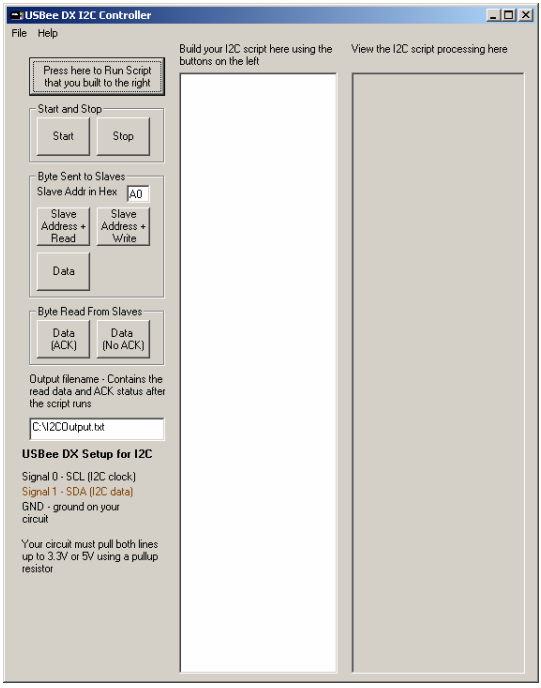
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to generate one of the fixed sets of frequencies on the digital lines.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the Signal 0 thru 7 lines on the USBee DX pod to your circuit you would like to actively drive.
- Run the Frequency Generator Application.
- From the dropdown list, select the set of frequencies that you want to generate out the pod.
- These frequencies are now being generated on the pod digital signals.

10 I2C Controller

This section details the operation of the I2C Controller application that comes with the USBee DX. The I2C Controller lets you control (be the I2C Master) an I2C device using the SDA and SCL lines of the device.

The Below you see the application screen.



The To control a device you must first create an I2C text script in the script window. You can either type in the window as you would a text editor or you can use the buttons on the left to quickly insert the correct tokens for the various parts of an I2C transaction.

The valid tokens are as follows:

<START>	To generate a Start condition
<STOP>	To generate a Stop conditon
<Slave Address Read: A0> <ACK=?>	To generate a Read Command
<Slave Address Write: A0> <ACK=?>	To generate a Write Command

<Data to Slave: 00> <ACK=?>	To send a byte to the slave
<Data from Slave: ??> <ACK>	To read a byte from the slave
<Data from Slave: ??> <No ACK>	To read a byte from the slave with no ACK following the byte

10.1 I2C Controller Specifications

I2C Clock Speed	2.2 KHz average
I2C Control Method	Text Script
I2C Script Tokens	Start, Stop, Ack, Nak, Read, Write, Data
Script Edit Functions	Cut, Copy, Paste, Save, Open, New
I2C Output Format	Text File (includes read data and Ack state)
Channel Output Drive Current	4mA
Output Low Level	< 0.8V
Output High Level	Open Collector (requires external pull-up resistor)

10.2 Quick Start

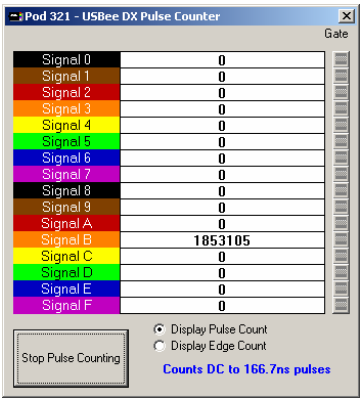
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to generate I2C transactions.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the Signal 0 pin on the USBee DX pod to your circuit SDA line.
- Connect the Signal 1 pin on the USBee DX pod to your circuit SCL line.
- Run the I2C Controller Application.
- Press the buttons to create a script of the I2C transaction you want to run.
- Press the Run Script button to generate the I2C transaction.
- The transaction result is written to the output window (and text file) including and read data and ACK states..

11 Pulse Counter

This section details the operation of the Pulse Counter application that comes with the USBee DX. The Pulse Counter is used to count the number of cycles or edges that are detected on up to 16 of the digital lines.

Below you see the application screen.



To start counting the pulses or edges on the signals press the Start Puls Counting button. The pulses are counted and the current range of pulses is displayed. In this case the system is counting all pulses down to 166.7nsec wide.

You can use any of the 15 lines as a gate to enable the counting during specified times. For example, you can count pulses only when Signal 0 is high by setting the Signal 0 Gate to High. Pulses that occur when Signal 0 is low are not counted

11.1 Pulse Counter Specifications

Digital Channels Measured	16
Analog Channels Measured	0
Minimum Pulse Width [1]	83.3nS
Pulse Count Control	Clear, Start and Stop
Display Mode	Pulse or Edge Count
External Gate Signals	up to 15
Gate Conditions	High or Low

11.2 Quick Start

In order to quickly get up and running using this application, here is a step by step list of the things you need to do to count the number of edges or pulses of a digital signal.

- Connect the GND pin on the USBee DX pod to one of the signal wires using the small socket on the end of the wire.
- Connect the other end of the wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- Connect the Signal 0 thru F signals on the USBee DX pod to your circuit you would like to test.
- Run the Pulse Counter Application.
- Press the Start Counting button.
- The number of pulses one each of the 8 digital signals is displayed.
- You can use any of the 15 lines as a gate to enable the counting during specified times. For example, you can count pulses only when Signal 0 is high by setting the Signal 0 Gate to High. Pulses that occur when Signal 0 is low are not counted.

12 USBee Toolbuilder

12.1 Overview

The USBee DX Test Pod System consists of the USBee DX Test Pod connected to a Windows® 2000, XP or Vista PC High Speed USB 2.0 port through the USB cable, and to your circuit using the multicolored test leads and clips. Once connected and installed, the USBee can then be controlled using either the USBee DX Windows Software or your own USBee DX Toolbuilder software.

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

The USBee DX Test Pod is ideal for students or designers 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 PC application to control and monitor the signal pins on the pod.

The USBee DX has headers that are the interface to your circuits. The signals on these headers represent a 16 bit data bus, a Read/Write#/TRG signal (T) and a clock line (C). Using the libraries and source code provided you can do reads and writes to these signals. The USBee DX acts as the master, driving the T and C signals to your circuit.

There are six modes of data transfers that you can use depending on your system needs.

- Voltmeter Mode
- Signal Capture
- Digital Signal Generator
- Bi-Directional “bit-bang” mode
- Uni-Directional High Speed mode

12.1.1 Voltmeter Mode

The simplest of the analog functions is the DVM (Digital Voltmeter) routine called GetAllSignals. It simply samples all of the signals on the USBee DX pod and measures the voltage on both analog channels. This measurement is taken over a second an the average is returned.

The routine GetAllSignals () samples the specified channel and returns the measurement.

12.1.2 Signal Capture

The USBee DX has the ability to capture samples from the 16 digital signals and two analog channels at the same time. Each analog sample is time synchronized with the corresponding digital samples.

In signal capture modes, there is a single capture buffer where each sample is a long value made up of 4 bytes. The low order 2 bytes represent the 16 digital channels. Digital Signal 0 is bit 0 of each long value. The Analog samples are the high two bytes where each byte is an 8-bit ADC value taken during that sample period for that channel. The samples range from 0 (at -10.0V) to 255 (at +10.0V). Each count of the ADC equates to 78.125mV, which is the lowest resolution possible on the USBee DX without averaging.

The maximum sample rate that is possible in Signal Capture mode is 24Msps. This value can depend on your PC system and available processing speed and how many byte lanes are sampling data. The basic rule of thumb is that the maximum bandwidth through USB 2.0 is near 24Mbytes/second. Therefore to capture 2 bytelanes (16 digital channels for example) would equate to a maximum sample rate of 12Msps.

The method for performing a single data capture, or sampling, using the Signal Capture routines is as follows:

- Allocate the sample buffers (MakeBuffer())
- Start the capture running (StartCapture(…))
- Monitor the capture in progress to determine if it is triggered, filling, or completed. (CaptureStatus()).
- End the capture when it is finished. (StopCapture())
- Process the sample data that is now contained in the sample buffers.

Once the data is captured into a buffer, you can call the Bus Decoder routines to extract the data from these busses.

12.1.3 Digital Signal Generator

The USBee DX has the ability to generate (output) samples from 8 or 16 digital signals at up to 24Msps or 12Msps in Signal Generator mode.

In this mode, there is a single buffer that stores the samples to generate. Each sample is a long value made up of 4 bytes. The low order 2 bytes represent the 16 digital channels. Digital Signal 0 is bit 0 of each long value. The high two bytes are not used. These samples can then be generated on command.

The maximum sample rate that is possible Signal Generator mode is 24Msps. This value can depend on your PC system and available processing speed and how many byte lanes are generating data. The basic rule of thumb is that the maximum bandwidth through USB 2.0 is near 24Mbytes/second. Therefore to generate 2 bytelanes (16

digital channels for example) would equate to a maximum sample rate of 12Msps.

The method for generating a single output pattern using the Signal Generator routines is as follows:

- Allocate the sample buffer (MakeBuffer())
- Fill the sample buffer with the pattern data you want to generate.
- Start the generation running (StartGenerate (…))
- Monitor the generation in progress to determine if it is triggered, filling, or completed. (GenerateStatus()).
- Terminate the generation. (StopGenerate())

The USBee DX can not generate analog output voltages using this mode. Variable analog outputs are possible using the PWM Controller and an external RC circuit.

12.1.4 Bi-Directional and Uni-Directional Modes

These two modes allow bit-level data transfers to and from the USBee DX pod. The first offers complete flexibility of the 8 digital signal lines, while the other gives you very high transfer rates.

In the Bi-Directional Mode, each of the 16 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 16 signals lines will return the actual value on the signal (whether it is an input or an output)

In the High-Speed Mode, all of the 16 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	Write-SetSignals	300k Bytes/sec	~300k Bytes/sec
Bi-Directional	Read-GetSignals	175k Bytes/sec	~175k Bytes/sec
High-Speed	Write-SetSignals	24M Bytes/sec	~20M Bytes/sec
High-Speed	Read-GetSignals	16M Bytes/sec	~13M Bytes/sec

12.2 System Software Architecture

The USBee DX Pod is controlled through a set of Windows DLL function calls. These function calls are defined in following sections and provide initialization and data transfer routines. This DLL can be called using a variety of languages, including C. We have included a sample application in C that show how you can use the calls to setup and control the pod. You can port this example to any language that can call DLL functions (Delphi, Visual Basic, …)

After installing the software on your computer, you can then plug in the USBee DX pod. Immediately after plugging in the pod, the operating system finds the USBEEDX.INF file in the \Windows\INF directory. This file specifies which driver to load for that device, which is the USBEEDX.SYS file in the \Windows\System32\Driver directory. This driver then remains resident in memory until you unplug the device.

Once you run your USBee Toolbuilder application, it will call the functions in the USBEEDX.DLL file in the \Windows\System32 directory. This DLL will then make the correct calls to the USBEEDX.SYS driver to perform the USB transfers that are required by the pod.

12.3 The USBee DX Pod Hardware

The USBee DX has two sets of header pins that can be connected to a standard 0.025” square socketed wire. One section of pins is for the digital interface and the other is for the analog channels. Below is the pinout for these two interfaces.

Digital 20 pin Header Pinout: (0-5V Max input levels)

Pin 0	Data In/Out Bit 0
Pin 1	Data In/Out Bit 1
Pin 2	Data In/Out Bit 2
Pin 3	Data In/Out Bit 3
Pin 4	Data In/Out Bit 4
Pin 5	Data In/Out Bit 5
Pin 6	Data In/Out Bit 6
Pin 7	Data In/Out Bit 7
Pin 8	Data In/Out Bit 8
Pin 9	Data In/Out Bit 9
Pin A	Data In/Out Bit 10
Pin B	Data In/Out Bit 11
Pin C	Data In/Out Bit 12
Pin D	Data In/Out Bit 13
Pin E	Data In/Out Bit 14
Pin F	Data In/Out Bit 15
Pin T	Read/Write# Output (bit-bang mode),TRG (Signal Generator Mode) (R/W#/TRG)
Pin C	Clock Output (CLK)
Pin G (x2)	Ground

Analog 4 pin Header Pinout: (-10V to +10V Max input levels)

Pin 1	Analog Channel 1 Input
Pin 2	Analog Channel 2 Input
Pin G (x2)	Ground

Each of the calls to the USBee DX interface libraries operate on a sample buffer. For each sample that is sent out the signal pins or read into the signal pins, the R/W#/TRG (T) line is set and the CLK line (C) toggles to indicate the occurrence of a new sample. Each of the bits in the sample transferred maps to the corresponding signal on the DX pod. For example, if you send out a byte 0x80 to the pod, first the Read/Write# line (T) will be driven low, then the signal on Pin 7 will go high and the others (pin 0-6 and pin 8 - F) will go low. Once the data

is on the pins, the Clock line (C) is toggled to indicate that the new data is present.

12.4 Installing the USBee DX Toolbuilder

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

The USBee DX Toolbuilder software is included as part of the installation with the USBee DX Installation CD and can be downloaded from www.usbee.com. Run the setup.exe install program in the downloaded file to install from the web. The install program will install the following USBee Toolbuilder files and drivers into their correct location on your system. Other files will also be installed, but are not necessary for Toolbuilder operation.

12.4.1 USBee DX Toolbuilder Project Contents

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

USBeeDX.dsp	Visual C Project File
USBeeDX.dsw	Visual C Workspace File
USBeeDX.cpp	Visual C program
UsbDXla.lib	USBee DX Interface library file

The USBee DX 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 DX pod to USB.

- USBDXLA.DLL in the Windows/System32 directory
- USBEEDX.INF in the Windows/INF directory
- USBEEDX.SYS in the Windows/System32/Drivers directory

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

12.5 USBee DX Toolbuilder Functions

This section details the functions that are available in the usbdxla.dll and defines the parameters to each call.

12.5.1 Initializing the USBee DX Pod

12.5.1.1 EnumerateDXPods

This routine finds all of the USBee DX pods that are attached to your computer and returns an array of the Pod IDs.

Calling Convention

```
int EnumerateDxPods(unsigned int *PodID);
```

where `PodID` is a pointer to the list of Pod IDs found.

Return Value:

Number of USBee DX Pods found

12.5.1.2 InitializeDXPod

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

Calling Convention

```
int InitializeDXPod(unsigned int PodNumber);
```

where `PodNumber` is the Pod ID of the pod used found on the back of the unit.

Return Value:

0 = Pod Not Found

1 = Pod Initialized

12.5.2 Bit Bang-Modes

12.5.2.1 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 16 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 16 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 16 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

12.5.2.2 SetSignals - Setting the USBee DX Output Signals

Calling Convention

```
int SetSignals (  unsigned long State,
                  unsigned int length,
                  unsigned long *Samples)
```

- State is not used for High-Speed Mode. In Bi-Directional mode, State is the Input/Output state of each of the 16 USBee signals (0 through F). 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 Samples() that will be shifted out the USBee pod. The maximum length is 16383.
- Samples() is the array that holds the series of samples 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 (T) line is set low prior to data available, and the CLK line (C) toggles for each output sample (Length times).

Return Value:

1 = Successful

0 = Failure

12.5.2.3 GetSignals - Reading the USBee DX Input Signals

Calling Convention

```
int GetSignals   (   unsigned long State,
                     unsigned int length,
                     unsigned long *Samples)
```

- State is not used for High-Speed Mode. In Bi-Directional mode, State is the Input/Output state of each of the 16 USBee digital signals (0 through F). 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 Samples() that will be read from the USBee pod. The maximum length is 16383.

- Samples() is the array that will hold the series of samples that represent the levels read on the input signals. The Read/Write# (T) line is set high prior to data available, and the CLK line (C) toggles for each input byte (Length times).
- Return Value is the digital level of all 16 USBee pod Signals (bit 0 is signal 0, bit 15 is signal F)

12.5.3 Logic Analyzer and Oscilloscope Functions

The following API describes the routines that control the Logic Analyzer and Oscilloscope functionality of the USBee DX Test Pod.

12.5.3.1 MakeBuffer

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

Calling Convention

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

where `Size` is the number of samples to allocate. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels.

Return Value:

0 = Failed to allocate the buffer

other = pointer to allocated buffer

12.5.3.2 DeleteBuffer

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

Calling Convention

```
unsigned int *DeleteBuffer(   unsigned long
                              *buffer)
```

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

Return Value:

0 = Failed to deallocate the buffer

other = Success

12.5.3.3 StartCapture

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

Calling Convention

```
int StartCapture( unsigned int Channels,
                  unsigned int Slope,
                  unsigned int AnalogChannel,
                  unsigned int Level,
                  unsigned int SampleRate,
                  unsigned int ClockMode,
                  unsigned long *Triggers,
                  signed int TriggerNumber,
                  unsigned long *buffer,
                  unsigned long length,
                  unsigned long poststore);
```

- Channels represent which samples to take:
    - Bit 0: 1 = Sample Digital 0-7 signals
    - Bit 1: 1 = Sample Digital 8-F signals
    - Bit 2: 1 = Sample Analog Channel 1
    - Bit 3: 1 = Sample Analog Channel 2
- Slope is as follows:
    - 0 = Analog Slope for Trigger is Don't Care. Uses Digital Triggers instead.
    - 1 = Analog Slope for Trigger is Rising Edge. Ignores digital triggers.
    - 2 = Analog Slope for Trigger is Falling Edge. Ignores digital triggers.
- AnalogChannel specifies which analog channel to use for triggering
    - 1 = Channel 1
    - 2 = Channel 2
- Level: if Slope is not 0, this value specifies the analog trigger level. This value is in ADC counts, which go from 0 at -10V to 255 at +10V (78.125mV per count).
- SampleRate is as follows:
    - 247 = 24Msps
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- ClockMode: Always 0 - reserved

- Triggers: array of Mask/Value sample pairs used for triggering on the digital samples. 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 corresponding bit is a 0 in Value. These triggers are only in effect if the Slope is 0.
- 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. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels.
- 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

12.5.3.4 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

12.5.3.5 StopCapture

This routine terminates a pending capture.

Calling Convention

```
int StopCapture(void)
```

Return Value:

1 = Capture Stopped

0 = Stop Failed

12.5.3.6 LoggedData

This routine returns the 4 byte value of a particular sample. The low 2 bytes contain the 16 digital channels. The high two bytes contain two 8-bit ADC values for the two analog channels.

Calling Convention

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

Index: sample number to return

Return Value:

Value of the given sample

12.5.3.7 DecodeUSB

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int DecodeUSB (unsigned long *SampleBuffer, unsigned char
*OutFilename, long StartSample, long EndSample, long
NumberOfSamples, long ShowEndpoint, long ShowAddress, long
```

```
DPlus, long DMinus, long Speed, long Rate, long SOF, long
delimiter, long showall, long hex);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels which are not used.
- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- ShowEndpoint: 999 = show all traffic, otherwise show only this USB endpoint number traffic
- ShowAddress: 999 = show all USB devices, otherwise only show the USB device with this USB address
- DPlus: Which signal (0 – 15) to use for the D Plus signal
- DMinus: Which signal (0 – 15) to use for the D Minus signal
- Speed: 0 = Low Speed USB, 1 = Full Speed USB
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- SOF: 0 = do not show the SOF (Start of Frames), 1 = show SOFs
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex

Return Value is always 0

12.5.3.8 DecodeSPI

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int DecodeSPI (unsigned long *SampleBuffer, unsigned char
*OutFilename, long StartSample, long EndSample, long Rate,
unsigned long SS, unsigned long SCK, unsigned long MOSI,
unsigned long MISO, unsigned long MISOEdge, unsigned long
```

```
MOSIEdge, unsigned long delimiter, unsigned long hex,
unsigned long UseSS, long BytesPerLine);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels which are not used.
- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- SS: Which signal (0 – 15) to use for the Slave Select signal
- SCK: Which signal (0 – 15) to use for the clock signal
- MISO: Which signal (0 – 15) to use for the MISO signal
- MOSI: Which signal (0 – 15) to use for the MOSI signal
- MOSIEdge: 0 = use falling edge of SCK to sample data on MOSI, 1 = use rising edge
- MISOEdge: 0 = use falling edge of SCK to sample data on MISO, 1 = use rising edge
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- UseSS: 0 = don't use an SS signal, 1 = use the SS signal
- BytesPerLine: How many output words are on each output line.

Return Value is always 0

12.5.3.9 DecodeI2C

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int DecodeI2C (unsigned long *SampleBuffer, unsigned char
*OutFilename, long StartSample, long EndSample, long Rate,
unsigned long SDA, unsigned long SCL, long showack, long
delimiter, long showall, long hex);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels which are not used.
- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- SDA: Which signal (0 – 15) to use for the SDA signal
- SCL: Which signal (0 – 15) to use for the SCL signal
- ShowAck: 0 = Do not show each byte ACK values, 1 = show the ACK value after each byte
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex

Return Value is always 0

12.5.3.10 DecodeCAN

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int DecodeCAN (unsigned long * SampleBuffer, unsigned char
*OutFilename, long StartSample, long EndSample, unsigned
long Rate, unsigned long Channel, unsigned long BitRate,
```

```
unsigned long maxID, unsigned long minID, long delimiter,
long showall, long hex);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels which are not used.
- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- Channel: Which signal (0 – 15) to use for the CAN signal
- BitRate: The value of the bit rate in bits per second (for 250kbps use 250000)
- MaxID: 0 = show all packets, otherwise this is the maximum ID to display
- MinID: 0 = show all packets, otherwise this is the minimum ID to display
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex

Return Value is always 0

12.5.3.11 Decode1Wire

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int Decode1Wire (unsigned long *SampleBuffer, unsigned char
*OutFilename, long StartSample, long EndSample, long Rate,
unsigned long Signal, long delimiter, long showall, long
hex);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels which are not used.
- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- Signal: Which signal (0 – 15) to use for the 1-Wire signal
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex

Return Value is always 0

12.5.3.12 DecodeParallel

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int DecodeParallel (unsigned long *SampleBuffer, unsigned
char *OutFilename, long StartSample, long EndSample, long
Rate, unsigned long Channels, unsigned long Clock, unsigned
long UseCLK, long CLKEdge, unsigned long delimiter,
unsigned long hex, long BytesPerLine);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels which are not used.
- OutFilename: pointer to the filename string to write the decoded data to.

- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- Channels: Bit mask which represents which signals are part of the parallel data bus. Bit 0 is Pod signal 0. Bit 15 is pod signal F.
- Clock: Which signal (0 – 15) to use for the clock signal
- UseCLK: 0 – don't use the Clock signal above, 1 – use the Clock signal above to qualify the samples
- CLKEdge: 0 = use falling edge of the Clock to sample data, 1 = use rising edge
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- BytesPerLine: How many output words are on each output line.

Return Value is always 0

12.5.3.13 DecodeSerial

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int DecodeSerial (unsigned long *SampleBuffer, unsigned
char *OutFilename, long StartSample, long EndSample,
unsigned long Rate, unsigned long Channel, unsigned long
AlignValue, unsigned long AlignEdge, unsigned long
AlignChannel, unsigned long UseAlignChannel, unsigned long
ClockChannel, unsigned long ClockEdge, unsigned long
BitsPerValue, unsigned long MSBFirst, unsigned long
delimiter, unsigned long hex, long BytesPerLine);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high

two bytes being two 8-bit ADC values for each of the two analog channels which are not used.

- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- Channel: Which signal (0 – 15) to use for the serial signal
- AlignValue: When using word aligning, bus value which is used for aligning the serial stream to byte boundaries.
- AlignEdge: When using an external signal for aligning, 0 = falling edge, 1 = rising edge.
- AlignChannel: When using an external signal for aligning, which signal (0 – 15) to use for the align signal
- UseAlignChannel: 0 = use word aligning, 1 = use external align signal
- ClockChannel: Which signal (0 – 15) to use for the clock signal
- CLKEdge: 0 = use falling edge of the Clock to sample data, 1 = use rising edge
- BitsPerValue: how many bits are in each word of the serial stream
- MSBFirst: 0 = LSBit is sent first, 1 = MSBit is sent first
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- BytesPerLine: How many output words are on each output line.

Return Value is always 0

12.5.3.14 DecodeASYNC

This routine decodes bus traffic and outputs the data to an output file. This routine works on a sample buffer captured using the StartCapture routine.

Calling Convention

```
int DecodeASYNC (unsigned long *SampleBuffer, unsigned char
*OutFilename, long StartSample, long EndSample, long Rate,
unsigned long Channels, unsigned long BaudRate, unsigned
long Parity, unsigned long DataBits, unsigned long
delimiter, unsigned long hex, unsigned long ascii, long
BytesPerLine);
```

- SampleBuffer: pointer to the sample buffer that contains the acquired sample data. Each sample is contained in a long (4 byte) value with the low two bytes being the 16 digital lines and the high two bytes being two 8-bit ADC values for each of the two analog channels which are not used.
- OutFilename: pointer to the filename string to write the decoded data to.
- StartSample: the index of the first sample to start decoding
- EndSample: the index of the last sample to decode
- NumberOfSamples: The total Sample Buffer Size
- Rate is the rate at which samples were taken during StartCapture:
    - 247 = 24Msps (must use this for Full Speed USB)
    - 167 = 16 Msps
    - 127 = 12 Msps
    - 87 = 8 Msps
    - 67 = 6 Msps
    - 47 = 4 Msps
    - 37 = 3 Msps
    - 27 = 2 Msps
    - 17 = 1 Msps
- Channels: Bit mask which represents which signals to decode. Bit 0 is Pod signal 0. Bit 15 is pod signal F.
- BaudRate: Baud Rate in bits per second (19.2K = 19200)
- Parity: 0 = No parity, 1 = Mark, 2 = Space, 3 = Even, 4 = Odd, 5 = Ignore
- DataBits: Number of data bits (4 to 24)
- Delimeter: 0 = no delimiter, 1 = Comma delimeter, 2 = Space delimeter
- Showall: 0 = Only show the data payload, 1 = show all packet details
- Hex: 0 = display data in decimal, 1 = display data in hex
- ASCII: 0 = show byte values, 1 = show ASCII equivalent
- BytesPerLine: How many output words are on each output line.

Return Value is always 0

12.5.3.15 DecodeSetName

This routine sets the string that is output during any of the above decoders and can represent a unique identifier for that bus.

Calling Convention

```
int DecodeSetName (char *name);
```

12.5.4 Digital Signal Generator Function

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

12.5.4.1 SetData

This routine sets the value of a given sample to the value specified. You can also write directly to the allocated buffer after calling MakeBuffer(). The low 2 bytes contain the 16 digital channels. The high two bytes contain two 8-bit ADC values for the two analog channels.

Calling Convention

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

- Index: sample number to change
- Value: 4-byte value to store in that sample

Return Value:

0 = Set failed

1 = Set successful

12.5.4.2 StartGenerate

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

Calling Convention

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

- Bits is the number of bits to generate
    - 8 = the low 8 digital signals (0 thru 7)
    - 16 = all digital signals (0 thru F)
- 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 (T) 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

12.5.4.3 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 Successful

12.5.4.4 StopGenerate

This routine stops a signal generation in progress and terminates a generation cycle.

Calling Convention

```
int StopGenerate(void );
```

Return Value:

0 = Stop Failed

1 = Stop Successful

12.5.5 Digital Voltmeter (DVM) Function

The following API describes the routine that samples both the digital and analog voltages.

12.5.5.1 GetAnalogAverageCount

This routine reads the average analog voltage at the specified channel.

Calling Convention

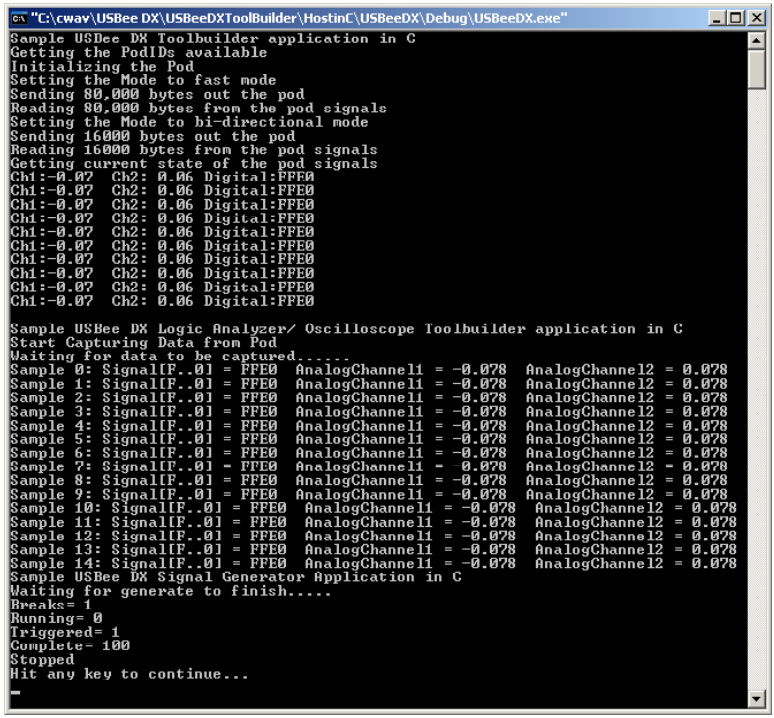
```
unsigned long GetAllSignals(
                            long *ch1,
                            long *ch2,
                            unsigned long *digital );
```

- `*ch1` and `*ch2` will be filled with the analog average voltage for that channel. The value returned is 100 times the actual value so you need to divide this by 100 to get the measured value in volts.
- `*digital` will be filled with the digital samples where each bit represents one digital channel. Bit 0 is digital signal 0. Bit 15 is digital signal F.

Return Value: Always 1

12.6 Example C Code

The following code listing is an example in very simple C that calls the DLL functions. It is a Command Prompt program that generates the following output when run.



File USBeeDX.cpp

```
//****************************************************************************
// USBee DX Toolbuilder Sample Application
//
// This file contains sample C code that accesses the USBee DX Toolbuilder functions
// that are contained in the USBDXLA.DLL file.  These routines are detailed in the
// USBee DX Toolbuilder document which includes the available routines and
// associated parameters.
//
// Copyright 2006, CWAV - All rights reserved.
// www.usbee.com
//****************************************************************************

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

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

// DX DLL Routine Declarations

// Basic Bit-Bang I/O Routines
CWAV_IMPORT int CWAV_API SetSignals (unsigned long State, unsigned int length, unsigned long *Bytes);
      // Sets the Digital signals
CWAV_IMPORT int CWAV_API GetSignals (unsigned long State, unsigned int length, unsigned long *Bytes);
      // Reads the Digital I/O signals
CWAV_IMPORT int CWAV_API SetMode (int Mode); // Sets the I/O Mode
CWAV_IMPORT unsigned long CWAV_API GetAllSignals( long *ch1, long *ch2, unsigned long *digital );

// 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 long * CWAV_API MakeBuffer( unsigned long Size );
      // Makes a Logic Analyzer/ OScope or Signal Generator buffer
CWAV_IMPORT int CWAV_API DeleteBuffer( unsigned long *buffer );   // Deletes the associated buffer
CWAV_IMPORT long CWAV_API SetData( unsigned long index, unsigned long value); // Sets the data in the logic buffer

CWAV_IMPORT int CWAV_API EnumerateDXPods( unsigned int *Pods );   // Find all USBee DX pods attached to this computer
CWAV_IMPORT int CWAV_API InitializeDXPod(unsigned int PodNumber);
       // Inits the specified Pod.  This must be done before operation.

// Logic Analyzer/ Oscilloscope Declarations
#define DIGITAL_HIGH          0x1
#define DIGITAL_LOW           0x2
#define ANALOG_LOW            0x4
#define ANALOG_HIGH           0x8


CWAV_IMPORT int CWAV_API StartCapture(       unsigned int Channels,
                                             unsigned int Slope,
                                             unsigned int AnalogChannel,
                                             unsigned int Level,
                                             unsigned int SampleRate,
                                             unsigned int ClockMode,
                                             unsigned long *Triggers,
                                             signed int TriggerNumber,
                                             unsigned long *buffer,
                                             unsigned long length,
                                             unsigned long poststore);

CWAV_IMPORT int CWAV_API StopCapture(void);          // End a Logic Analyzer trace
CWAV_IMPORT int CWAV_API CaptureStatus( char *breaks, char *running, char *triggered,// Monitor the capture in progress
                                        long *start, long *end, long *trigger, char *full );

// Signal Generator Declarations
CWAV_IMPORT int CWAV_API StartGenerate(unsigned long Bits, unsigned int SampleRate, unsigned char triggermode, unsigned
                                       long *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API GenerateStatus( char *breaks, char *running, char *triggered, char *complete );
      // Generation Status
CWAV_IMPORT int CWAV_API StopGenerate( void );  // Stops the Generation in progress

// 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

#define DONT_CARE_SLOPE       0
#define RISING_EDGE_SLOPE     1
#define FALLING_EDGE_SLOPE    2

// Protocol Decoders
CWAV_IMPORT int CWAV_API DecodeUSB (unsigned long *LoggedData, unsigned char *OutFilename,
                                    long StartSample, long EndSample, long NumberOfSamples,
                                    long ShowEndpoint, long ShowAddress, long DPlus, long DMinus,
                                    long Speed, long Rate, long SOF, long delimiter, long showall,
                                    long hex);

CWAV_IMPORT int CWAV_API DecodeSPI (unsigned long *SampleBuffer, unsigned char *OutFilename,
                                    long StartSample, long EndSample, long Rate,
```

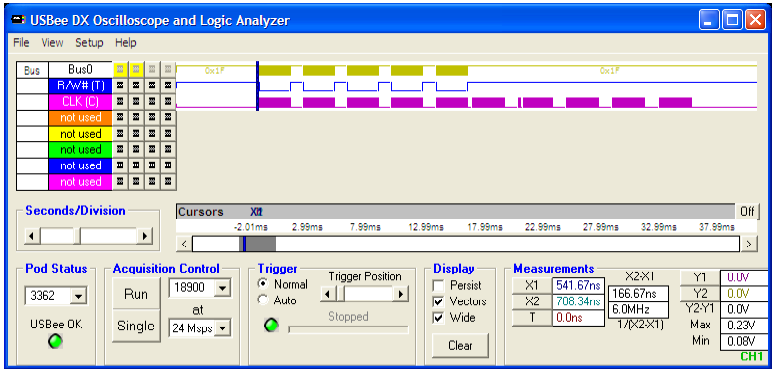
```
                                unsigned long SS,unsigned long SCK,unsigned long tMOSI,unsigned long tMISO,
                                unsigned long MISOEdge,unsigned long MOSIEdge,
                                unsigned long delimiter,unsigned long hex,unsigned long UseSS, long BytesPerLine);

CWAV_IMPORT int CWAV_API DecodeI2C (unsigned long *SampleBuffer, unsigned char *OutFilename,
                                long StartSample, long EndSample, long Rate, unsigned long SDA,
                                unsigned long SCL,
                                long showack,
                                long delimiter, long showall,
                                long hex);

CWAV_IMPORT int CWAV_API DecodeCAN (unsigned long *InputDecodeBuffer, unsigned char *OutFilename,
                                long StartSample, long EndSample, unsigned long Rate,
                                unsigned long Channel, unsigned long BitRate,
                                unsigned long maxID, unsigned long minID,
                                long delimiter, long showall,
                                long hex);

CWAV_IMPORT int CWAV_API Decode1Wire (unsigned long *SampleBuffer, unsigned char *OutFilename,
                                   long StartSample, long EndSample, long Rate, unsigned long Signal,
                                   long delimiter, long showall,
                                   long hex);

CWAV_IMPORT int CWAV_API DecodeParallel (unsigned long *SampleBuffer, unsigned char *OutFilename,
                                     long StartSample, long EndSample,
                                     long Rate, unsigned long Channels,unsigned long Clock,
                                     unsigned long UseCLK, long CLKEdge,
                                     unsigned long delimiter,unsigned long hex, long BytesPerLine);

CWAV_IMPORT int CWAV_API DecodeSerial (unsigned long *SampleBuffer, unsigned char *OutFilename,
                                   long StartSample, long EndSample, unsigned long Rate,
                                   unsigned long Channel,unsigned long AlignValue, unsigned long AlignEdge,
                                   unsigned long AlignChannel,unsigned long UseAlignChannel,
                                   unsigned long ClockChannel,unsigned long ClockEdge,
                                   unsigned long BitsPerValue, unsigned long MSBFirst,
                                   unsigned long delimiter,unsigned long hex, long BytesPerLine);

CWAV_IMPORT int CWAV_API DecodeASYNC (unsigned long *SampleBuffer, unsigned char *OutFilename,
                                   long StartSample, long EndSample, long Rate, unsigned long Channels,
                                   unsigned long BaudRate, unsigned long Parity, unsigned long DataBits,
                                  unsigned long delimiter,unsigned long hex,unsigned long ascii, long BytesPerLine);

CWAV_IMPORT int CWAV_API DecodeSetName (char *name);

unsigned char VoltsToCounts( float Volts )          // Converts Volts into ADC counts
{
        unsigned char counts;

        counts = (char) ((Volts + 10.0) / 0.078125);

        return(counts);
}
float CountsToVolts( unsigned long Counts )                     // Converts ADC counts into Volts
{
        double Volts;

        Volts = (float)((double)Counts * 0.078125) - 10.0;

        return((float)Volts);
}

int main(int argc, char* argv[])
{
        unsigned long DataInBuffer[65536], DataOutBuffer[65536];
        unsigned int PodNumber, PodID[10], NumberOfPods;
        int ReturnVal;
        unsigned long x;


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

        //***********************************
        // Pod Initializations Functions - must call InitializeDXPod before using any functions
        //***********************************
        printf("Getting the PodIDs available\n");
        NumberOfPods = EnumerateDXPods(PodID);
        if (NumberOfPods == 0) {
                printf("No USBee DX Pods found\n");
                getch();
                return 0;
        }

        PodNumber = PodID[0];         // Use the first one we find.  Change this to address your pod of choice.

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


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


        printf("Setting the Mode to fast mode\n");
        ReturnVal = SetMode(FAST_ONEWAY_DATA | DATA_CHANGES_ON_RISING_EDGE | _6MHz );

        if (ReturnVal != 1) {
                printf("Failure setting the mode\n");
                getch();
                return 0;
        }

        printf("Sending 80,000 bytes out the pod\n");
        for (x = 0; x < 5; x++)
```

```
{
        SetSignals (0xFFFF /* Don't Care */, 16000, DataOutBuffer);
}

printf("Reading 80,000 bytes from the pod signals\n");
for (x = 0; x < 5; x++)
{
        GetSignals (0x0000 /* Don't Care */, 16000, 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 16000 bytes out the pod\n");

SetSignals (0xFFFF, 16000, DataOutBuffer);

printf("Reading 16000 bytes from the pod signals\n");

GetSignals (0x0000, 16000, DataInBuffer);

long ch1;
long ch2;
unsigned long digital;

printf("Getting current state of the pod signals\n");

for (int y = 0; y < 10; y++)
{
        GetAllSignals ( &ch1, &ch2, &digital );

        float ch1f = (float)ch1 / (float)100;
        float ch2f = (float)ch2 / (float)100;

        printf("Ch1:%5.2f  Ch2:%5.2f Digital:%04X\n", ch1f, ch2f, digital);
}


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

printf("\nSample USBee DX Logic Analyzer/ Oscilloscope Toolbuilder application in C\n");

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

unsigned char Rate = 17;
// Sample Rate = 1Msps
unsigned char ClockMode = 2;                                                    // Internal Timing
unsigned long 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 long *SampleBuffer = MakeBuffer(SampleBufferLength);
long PostStore = SampleBufferLength;
unsigned char Slope = DONT_CARE_SLOPE;
unsigned char Level = VoltsToCounts(0.5);                   // Analog Trigger Level in ADC Counts
unsigned char AnalogTriggerChannel = 1;
                // Ch1 = 1, Ch2 = 2
PostStore = SampleBufferLength;
long Channels = ANALOG_HIGH + ANALOG_LOW + DIGITAL_HIGH + DIGITAL_LOW;
char Breaks;
char Running;
char Triggered;
long Start;
long End;
long Trigger;
char Full;


ReturnVal = StartCapture(Channels, Slope, AnalogTriggerChannel, Level, 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...");


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();

// The data is now available to read
for( x = 0; x < 15; x++)
```

```
        {
                printf("Sample %d: Signal[F..0] = %04X  AnalogChannel1 = %5.2g  AnalogChannel2 = %5.2g\n", x,
                                        (SampleBuffer[x] & 0xFFFF),
                                        CountsToVolts((SampleBuffer[x] >> 16) & 0xFF),
                                        CountsToVolts((SampleBuffer[x] >> 24) & 0xFF));
        }


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

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

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

        ReturnVal = StartGenerate (16, 17, DONT_CARE_TRIGGER, SampleBuffer, SampleBufferLength);

        printf("Waiting for generate to finish.");

        Running = 1;

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

                printf(".");

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

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

        DeleteBuffer(SampleBuffer);


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

        getch();

        return 0;
}
```

12.6.1 Performance Analysis of the “Bit-Bang” Routines

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) in FAST ONE-WAY mode. The Clock line (C) is the strobe for each of the samples transferred and the Data line (DATA) represents the data on each of the pod digital signal lines. The R/W# (T) indicates if it is a read or a write.

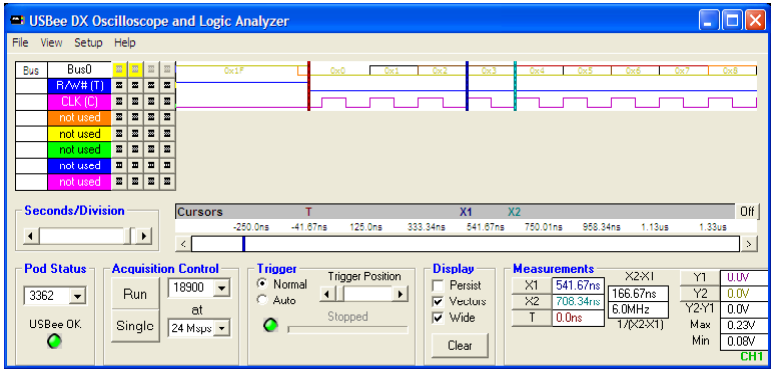


As you can see, this section takes about 38msec to execute. In this time we perform:

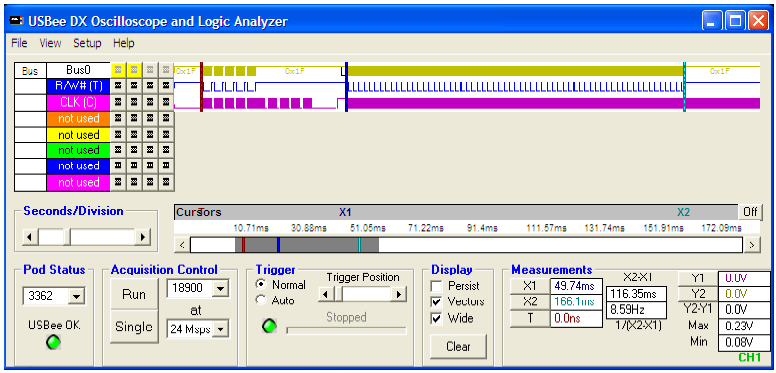
- Initializing the Pod
- Setting the Mode to High Speed mode
- Sending 80,000 samples out the pod using High Speed mode
- Reading 80,000 samples from the pod signals using High-Speed mode

The following trace shows the High-Speed Writes (80,000 samples) followed by Reads (80,000 samples). We first send out 5 blocks of 16,000 samples which take about 19msec. Then we follow with reads of 5 blocks of 16,000 samples which take about 19msec.

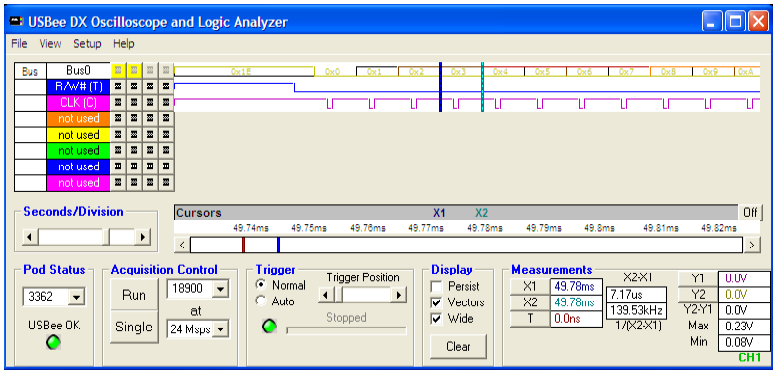
Below is a zoomed in trace showing the timing of each sample during the SetSignal call in Fast Mode. As you can see the clock is running at 6Msps and the data is changing on the rising edge of the clock. For Fast Mode writes and reads, each of the blocks of 16,000 bytes is bursted at 6Mbytes/sec (set using the SetMode parameters). 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.



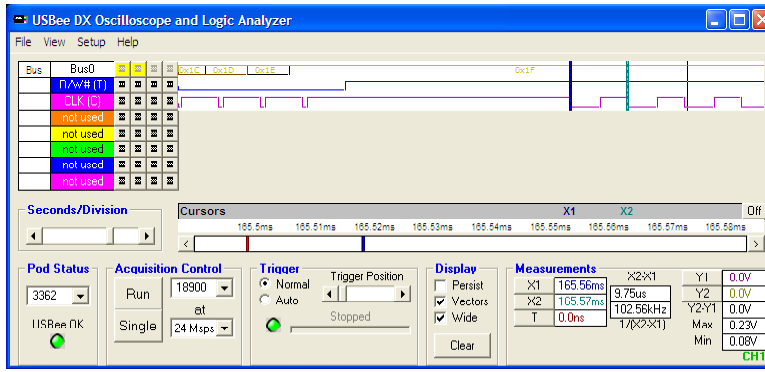
As a comparison between the modes, all transfers in high speed mode (all 160,000 samples) occur before the first dark blue cursor on the logic analyzer trace below. The Bi-Directional writes from the SetSignals (16000 samples) occur between the cursors, and the bi-direction reads occur after the second cursor.



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



Bi-Directional mode SetSignal byte timing

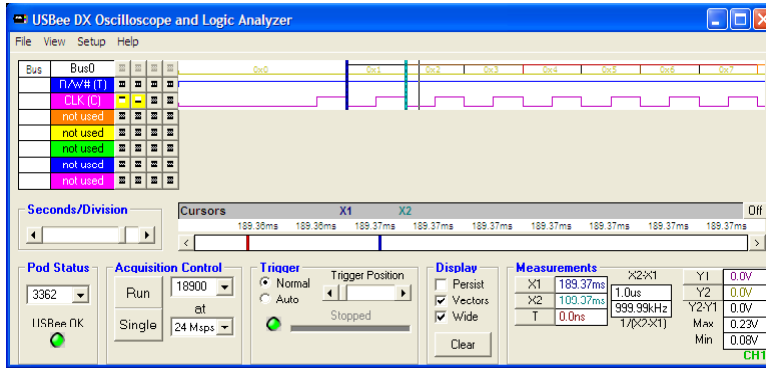


Bi-Directional mode GetSignal byte timing

The above trace shows the end of the SetSignals cycles and the following GetSignals 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.

In Signal Generator mode, the samples come out at a constant rate defined in the call the StartGenerate. Below you see a series of samples that are output using the StartGenerate routine and the resulting sample times.



Printed in the USA

Version 2.0