USBEE DX

TEST POD

USERS MANUAL

CWAV - Distributed by:

www.interworldna.com
Tel: 1-877-902-2979

USBEE DX TEST POD

USERS MANUAL

CWAV
www.interworldna.com
(877) 902-2979
support@interworldna.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.

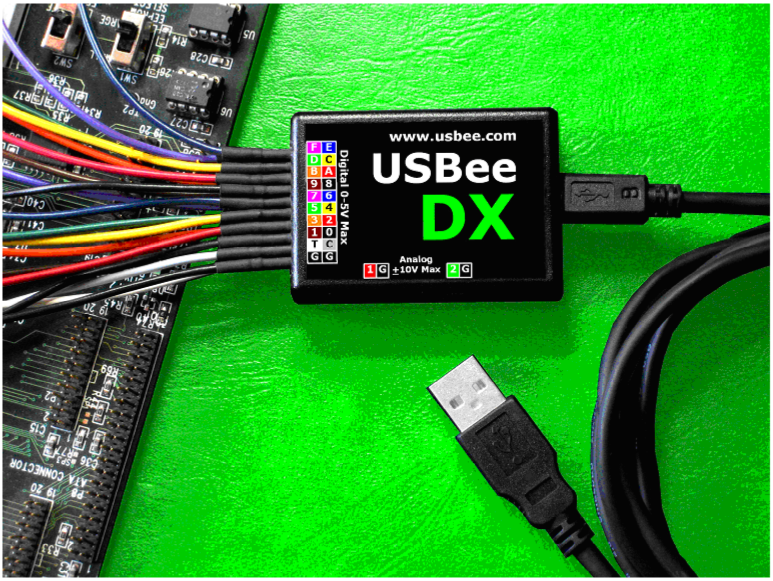
Term

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.

USBee DX Test Pod User's Manual, Version 3.1

TABLE OF CONTENTS

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

IMPORTANT! - The USBee Test Pod can only be connected to a target circuit which has the same ground reference level as your PC.

The USBee is NOT galvanically isolated. This mainly concerns systems where the target circuit AND the PC are plugged into AC power outlets. If your target system OR the PC (Laptop) are battery powered, there is no issue. If your PC and target circuit have different ground reference levels, connecting them together using the USBee GND signal can damage the devices.

To ensure both your PC and target system share the same ground reference, do the following:

1. Use polarized power cords for both the PC and target and plug them into the same AC circuit.

   If you use non-polarized power cords or use separate power circuits, the PC and target system may have different ground references which can damage the USBee, target and/or PC.

2. Ensure that a GND signal on the USBee is connected to the target ground (and not another voltage level).

Also,

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.

PC SYSTEM REQUIREMENTS

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

- Windows® 2000, XP or Vista 32-bit 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)

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

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.

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.

CALIBRATION

Your USBee DX has been calibrated at the factory and will not need calibration to start using it. This section is provided just as a reference in case you want to reproduce the calibration yourself.

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

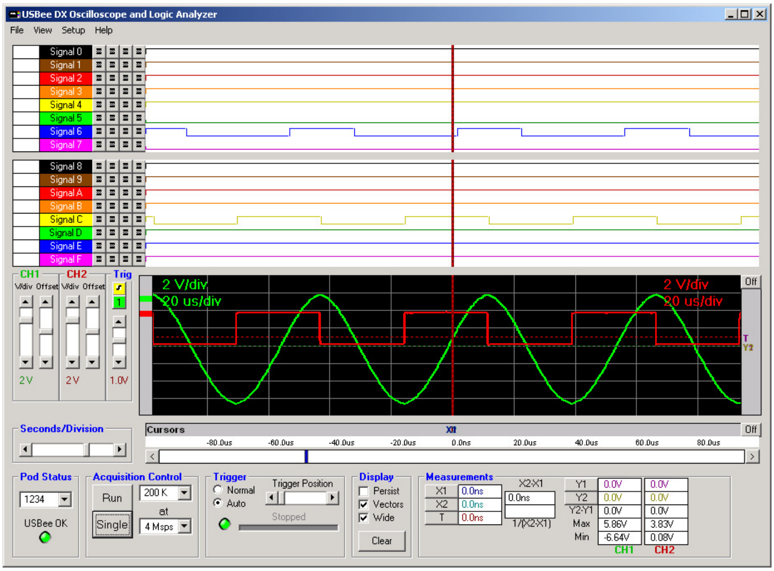
When you are ready to calibrate the USBee DX Pod, plug in the pod and run the Oscilloscope and Logic Analyzer application. Then go to the menu item Setup | Calibrate. You will be asked to confirm that you really want to do the calibration. If so, press Yes, otherwise press No. Then follow these steps:

- 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.

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.

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.

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: 1) Disconnect all other USB devices not needed from the PC, 2) 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)

FEATURES

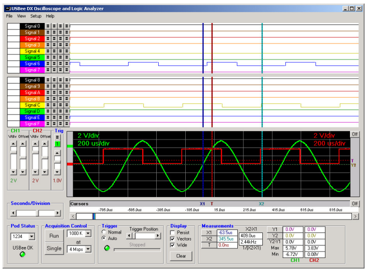
SETUP CONFIGURATION

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.

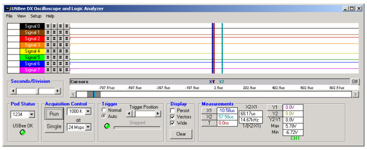
The configurations available are as follows:

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	8 Msps
2	0	12 Msps
2	8	8 Msps
2	16	6 Msps

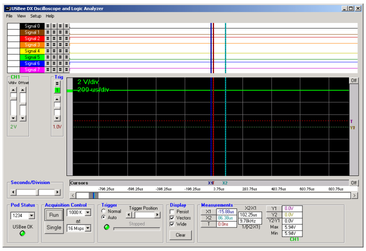
To select a configuration, click **Setup** on the menu and select the configuration of your choice. Below are examples of the application in various modes.



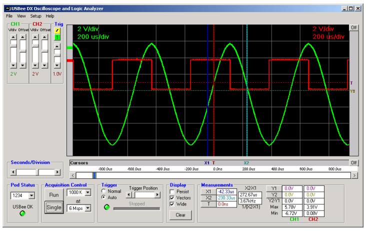
16 Digital–2 Analog Channels



8 Digital–0 Analog Channels



8 Digital–1 Analog Channels



0 Digital–2 Analog Channels

SIGNAL NAMES

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

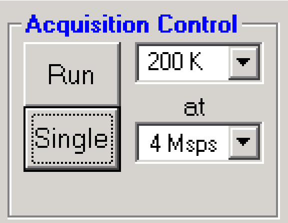
POD STATUS

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.

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.

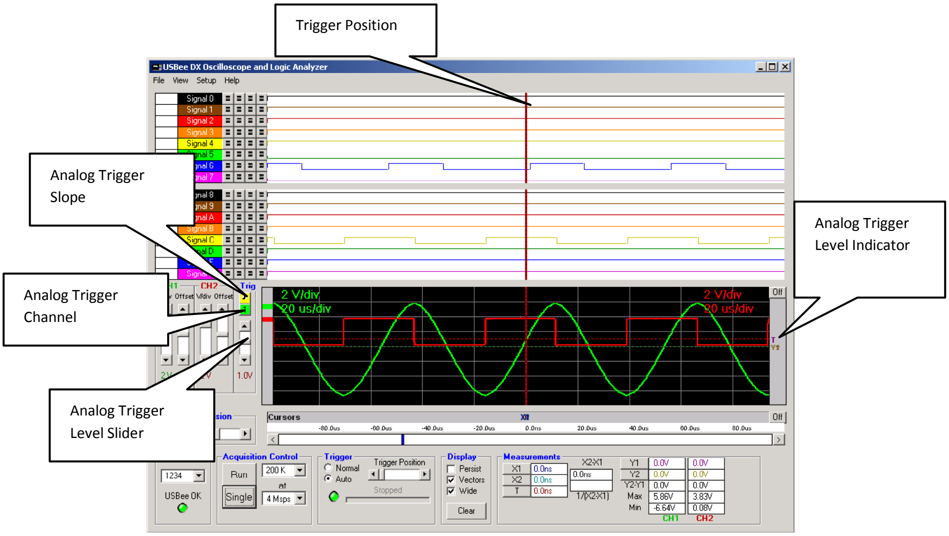
The **Buffer Size** lets you select the size of the Sample Buffer that is used. For each trace, the buffer is completely filled, and then the waveform is displayed. You can choose buffers that will capture the information that you want to see, but remember that the larger the buffer, the longer it will 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	8 Msps
2	0	12 Msps
2	8	8 Msps
2	16	6 Msps

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 digital.



Trigger Position

Analog Trigger Slope

Analog Trigger Channel

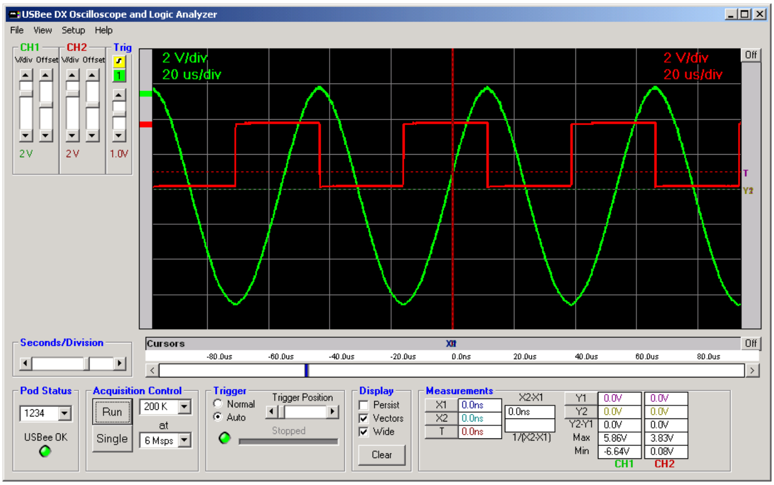
Analog Trigger Level Indicator

Analog Trigger Level Slider

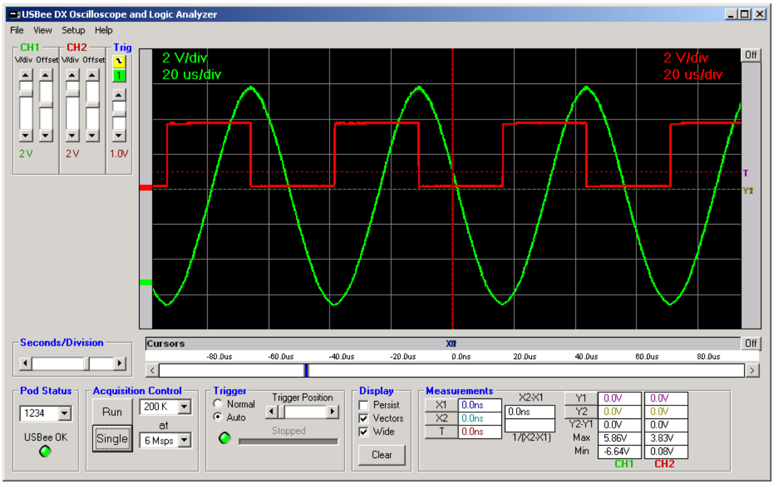
For an **Analog trigger**, you can specify the trigger voltage level (-10V to +10V) by using the slider on the left hand side of the analog waveform display. A red line that indicates the trigger level will momentarily be shown as you scroll this level. A small T will also be shown on the right hand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



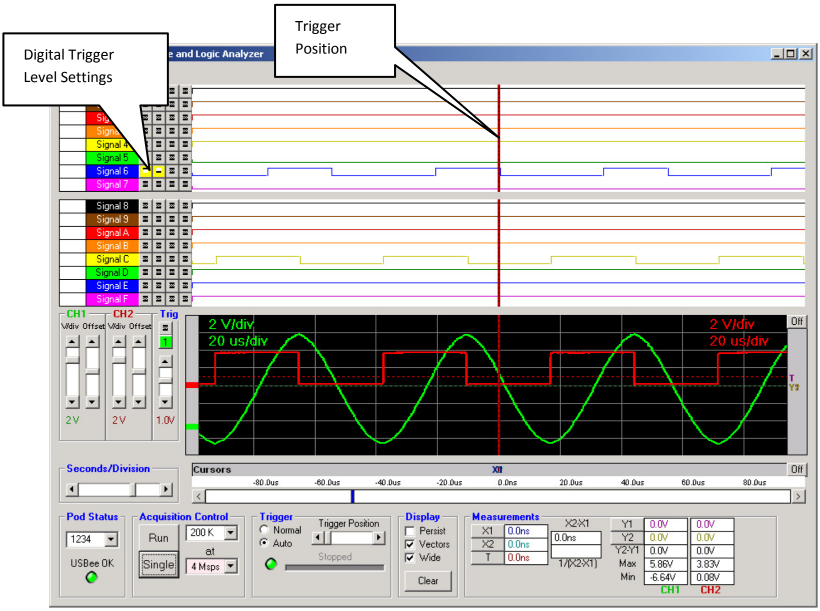
Analog Trigger Slope = Rising Edge



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.

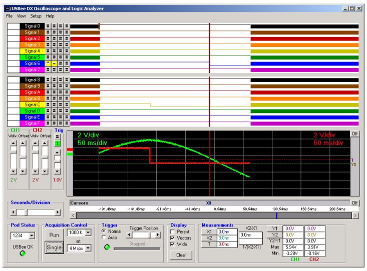

Digital Trigger Level Settings

Trigger Position

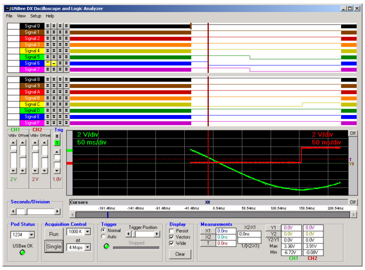
The digital trigger condition is made up of up to 4 sequential states of any of the 16 signals. Each state for a single signal can be high, low or don't care. This allows you to trigger on rising edges, falling edges, edges during another signals constant level, or one edge followed by another edge.

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.



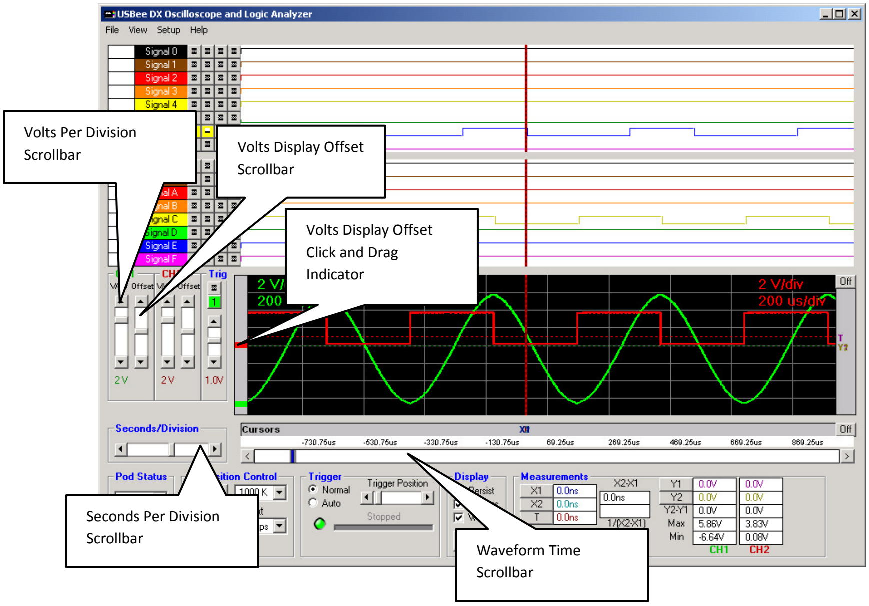
Trigger Position to the Right



Trigger Position to the Left

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.

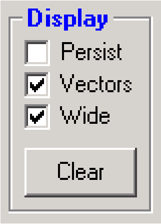

USBee DX Oscilloscope and Logic Analyzer
File View Setup Help
Volts Per Division Scrollbar
Volts Display Offset Scrollbar
Volts Display Offset Click and Drag Indicator
Seconds Per Division Scrollbar
Waveform Time Scrollbar
2 V/div
200 us/div
Seconds/Division
Cursors
Pod Status
Trigger
Normal
Auto
Trigger Position
Stopped
Display
Measurements
CH1
CH2

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.



☐ Persist
☑ Vectors
☑ Wide

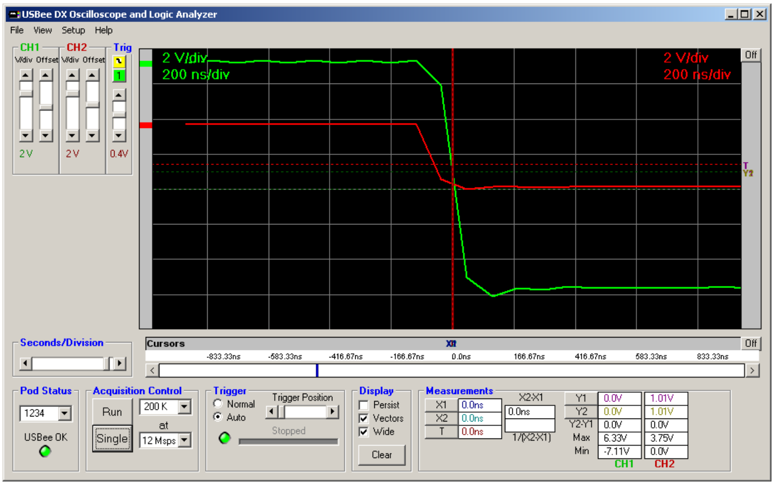
The Display section of the screen shows three selections that affect the way the waveform is displayed.

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.



Pod Status: 1234

Acquisition Control: 200 K at 12 Msps

Trigger: ○ Normal ● Auto

Display: ☐ Persist ☑ Vectors ☑ Wide

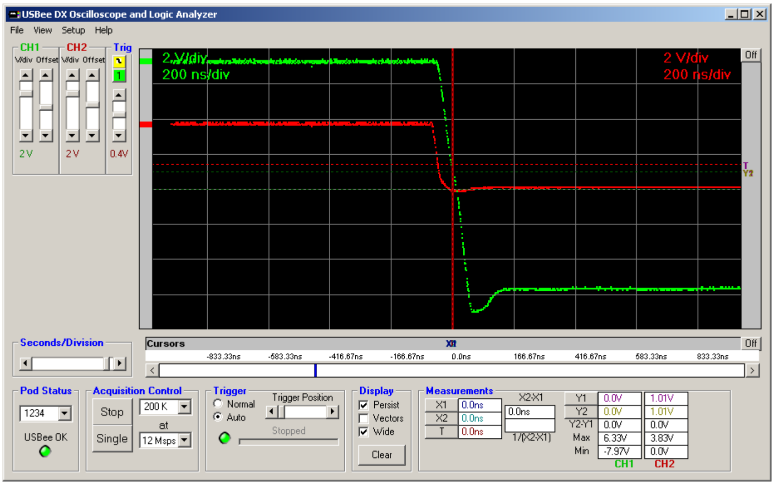
X1	0.0ns	
X2	0.0ns	
T	0.0ns	

X2-X1: 0.0ns

1/(X2-X1):

	CH1	CH2
Y1	0.0V	1.01V
Y2	0.0V	1.01V
Y2-Y1	0.0V	0.0V
Max	6.33V	3.75V
Min	-7.11V	0.0V

Persist = OFF, Vectors = ON, Wide = ON



Pod Status: 1234

Acquisition Control: 200 K at 12 Msps

Trigger: ○ Normal ● Auto

Display: ☑ Persist ☐ Vectors ☑ Wide

X1	0.0ns	
X2	0.0ns	
T	0.0ns	

X2-X1: 0.0ns

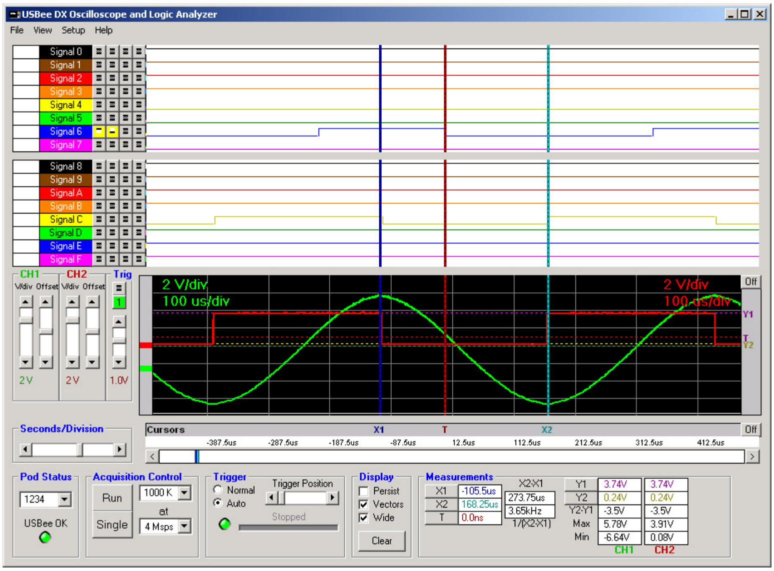
1/(X2-X1):

	CH1	CH2
Y1	0.0V	1.01V
Y2	0.0V	1.01V
Y2-Y1	0.0V	0.0V
Max	6.33V	3.83V
Min	-7.97V	0.0V

Persist = ON, Vectors = OFF, Wide = ON

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.



USBee DX Oscilloscope and Logic Analyzer

X1	-105.5us	X2-X1	273.75us
X2	168.25us	1/(X2-X1)	3.65kHz
T	0.0ns		

	CH1	CH2
Y1	3.74V	3.74V
Y2	0.24V	0.24V
Y2-Y1	-3.5V	-3.5V
Max	5.78V	3.91V
Min	-6.64V	0.08V

Pod Status: 1234, USBee OK
Acquisition Control: 1000 K at 4 Msps
Trigger: Auto; Stopped
Display: ☐ Persist ☑ Vectors ☑ Wide

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

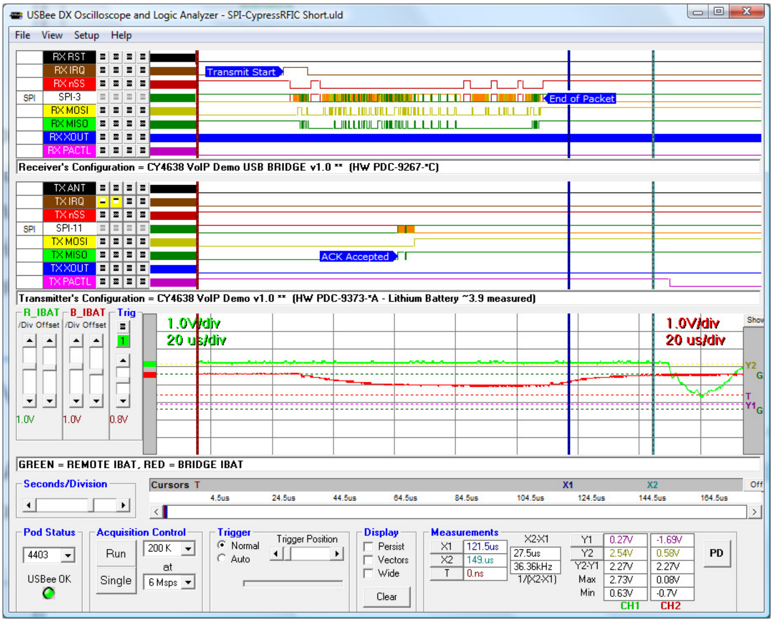
MARKERS

Markers can be placed on the waveform display to indicate to the viewer the occurrence of a certain event. A marker is small flag in blue that contains text that you define.

To place a marker on a waveform, position the mouse pointer at the location you want the marker placed and press the middle mouse button.

Left click on a marker to change the marker text. Right click on a marker to delete it. To delete all of the markers select the menu item View | Delete All Markers. Middle click on a marker to change its direction (left pointing or right pointing).

Below is a screenshot that includes three blue markers.



Receiver's Configuration = CY4638 VoIP Demo USB BRIDGE v1.0 ** (HW PDC-9267-*C)

Transmitter's Configuration = CY4638 VoIP Demo v1.0 ** (HW PDC-9373-*A - Lithium Battery ~3.9 measured)

GREEN = REMOTE IBAT, RED = BRIDGE IBAT

Pod Status: 4403, USBee OK

Acquisition Control: 200 K at 6 Msps

Trigger: ● Normal ○ Auto; Trigger Position

Display: □ Persist □ Vectors □ Wide

Measurements: X1 121.5us; X2 149.us; T 0.ns; X2-X1 27.5us; 36.36kHz 1/(X2-X1)

	CH1	CH2
Y1	0.27V	-1.69V
Y2	2.54V	0.58V
Y2-Y1	2.27V	2.27V
Max	2.73V	0.08V
Min	0.63V	-0.7V

Use the menu item View | Show Marker Labels to turn on or off the display of the text part of each marker. If the labels are off, only a small blue arrow is displayed at the marker position. The labels must be shown to change the text, change direction, or delete that marker.

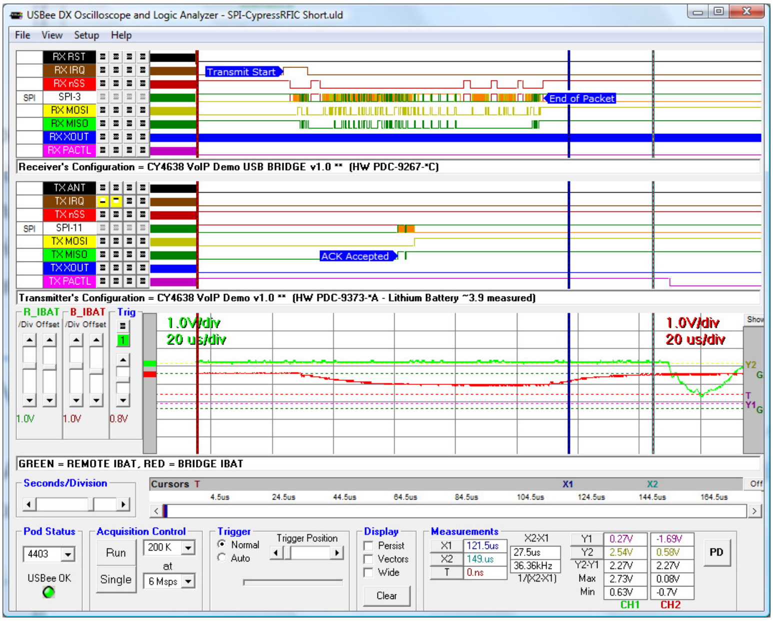
ANNOTATIONS

Text based annotations can be added to the display that can help document a particular capture. There are three annotation lines where text can be added. These lines are just below the digital waveforms and the analog waveforms.

To change the annotation text, select the text box and type the text you want to appear.

You can turn on or off the annotation text lines by using the menu item View | Show Annotation Text Boxes.

Below is a screenshot that shows the three annotation text lines below the waveforms.



Receiver's Configuration = CY4638 VoIP Demo USB BRIDGE v1.0 ** (HW PDC-9267-*C)

Transmitter's Configuration = CY4638 VoIP Demo v1.0 ** (HW PDC-9373-*A - Lithium Battery ~3.9 measured)

GREEN = REMOTE IBAT, RED = BRIDGE IBAT

Pod Status: 4403

Acquisition Control: 200 K at 6 Msps

Trigger: ● Normal ○ Auto

X1	121.5us	X2-X1
X2	149.us	27.5us
T	0.ns	36.36kHz
		1/(X2-X1)

Y1	0.27V	-1.69V
Y2	2.54V	0.58V
Y2-Y1	2.27V	2.27V
Max	2.73V	0.08V
Min	0.63V	-0.7V
	CH1	CH2

ANALOG CHANNEL BACKGROUND COLOR

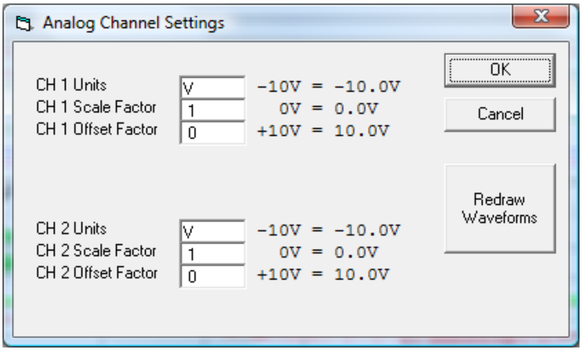
The background of the analog channel screen can be set to white or black using the View | Analog Background White or View | Analog Background Black menu items.

ANALOG CHANNEL SETTINGS

The analog channels can be assigned a text label to differentiate them on the display. To change the channel label, click on the label and type in the new name.

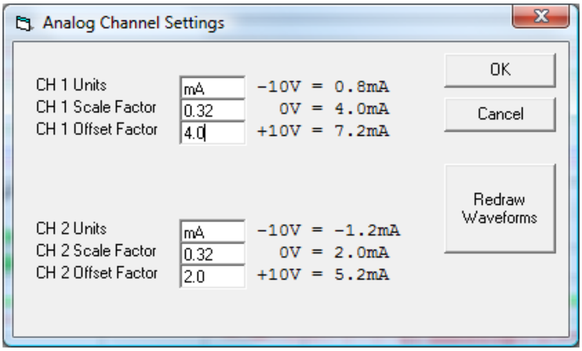
By default, each analog channel is set to display the measurements in Volts where 1V is shown as 1V on the display. Sometimes the measurement might actually mean a different thing than voltage. The menu item Setup | Analog Channel Settings lets you specify the units of measurement as well as a scale factor.

Below shows the default setting for the analog channels showing a gain value of 1, offset of 0 and units of Volts.


Analog Channel Settings

```
CH 1 Units          V     -10V = -10.0V
CH 1 Scale Factor   1       0V = 0.0V
CH 1 Offset Factor  0     +10V = 10.0V

CH 2 Units          V     -10V = -10.0V
CH 2 Scale Factor   1       0V = 0.0V
CH 2 Offset Factor  0     +10V = 10.0V
```

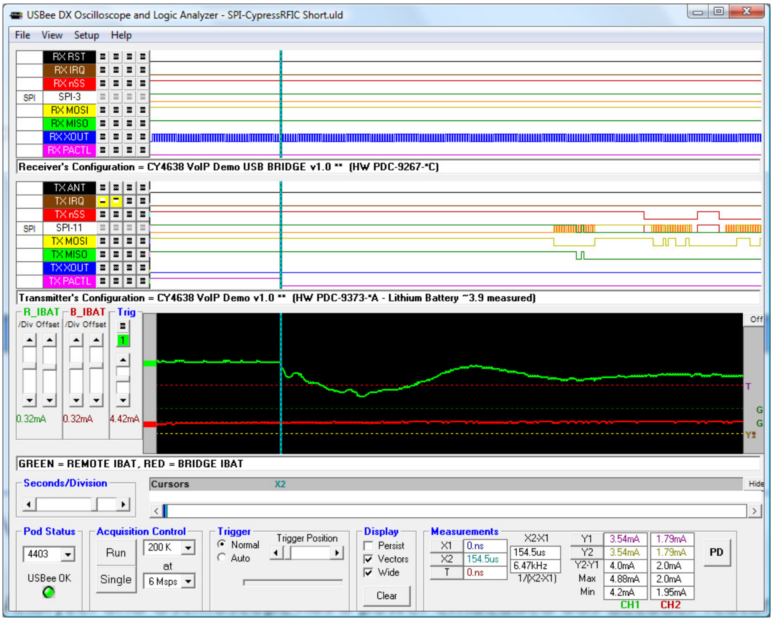
Below shows a setting of mA with various gains and offsets. Instead of displaying the actual value measured in volts, the display will show the scaled value in the new units.


Analog Channel Settings

```
CH 1 Units          mA     -10V = 0.8mA
CH 1 Scale Factor   0.32     0V = 4.0mA
CH 1 Offset Factor  4.0    +10V = 7.2mA

CH 2 Units          mA     -10V = -1.2mA
CH 2 Scale Factor   0.32     0V = 2.0mA
CH 2 Offset Factor  2.0    +10V = 5.2mA
```

ANALOG GRID LINES

To turn on or off the grid lines in the Analog display window, use the menu item View | Analog Grid Lines. Below shows the grid lines on and off.



Receiver's Configuration = CY4638 VoIP Demo USB BRIDGE v1.0 ** (HW PDC-9267-*C)

Transmitter's Configuration = CY4638 VoIP Demo v1.0 ** (HW PDC-9373-*A - Lithium Battery ~3.9 measured)

GREEN = REMOTE IBAT, RED = BRIDGE IBAT

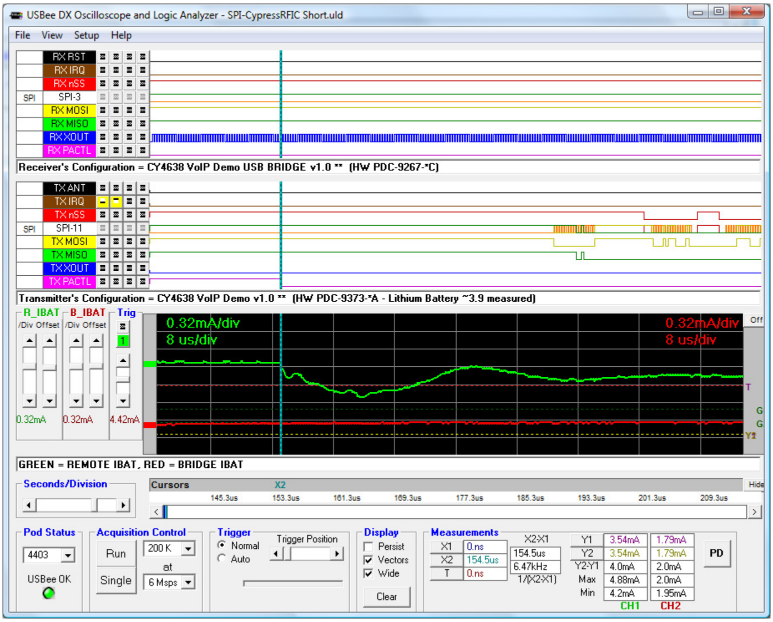
Pod Status: 4403

Acquisition Control: 200 K at 6 Msps

Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

X1	0.ns	X2-X1	Y1	3.54mA	1.79mA
X2	154.5us	154.5us	Y2	3.54mA	1.79mA
T	0.ns	6.47kHz	Y2-Y1	4.0mA	2.0mA
		1/(X2-X1)	Max	4.88mA	2.0mA
			Min	4.2mA	1.95mA
				CH1	CH2



Receiver's Configuration = CY4638 VoIP Demo USB BRIDGE v1.0 ** (HW PDC-9267-*C)

Transmitter's Configuration = CY4638 VoIP Demo v1.0 ** (HW PDC-9373-*A - Lithium Battery ~3.9 measured)

GREEN = REMOTE IBAT, RED = BRIDGE IBAT

Pod Status: 4403

Acquisition Control: 200 K at 6 Msps

Trigger: ● Normal ○ Auto

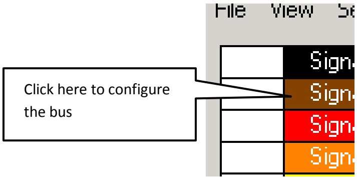
Display: □ Persist ☑ Vectors ☑ Wide

X1	0.ns	X2-X1	Y1	3.54mA	1.79mA
X2	154.5us	154.5us	Y2	3.54mA	1.79mA
T	0.ns	6.47kHz	Y2-Y1	4.0mA	2.0mA
		1/(X2-X1)	Max	4.88mA	2.0mA
			Min	4.2mA	1.95mA
				CH1	CH2

BUS DECODING

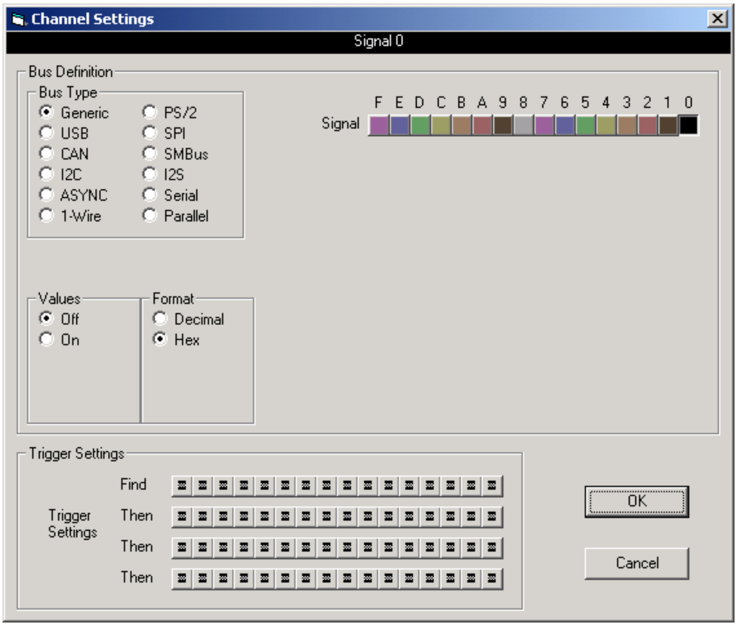
The USBee DX Logic Analyzer and Oscilloscope has a power embedded bus decoder feature that allows you to quickly analyze the contents of embedded communications captured by the pod.

BUS SETUP



Click here to configure the bus

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.



Channel Settings

Signal 0

Bus Definition

Bus Type

- ● Generic
- ○ USB
- ○ CAN
- ○ I2C
- ○ ASYNC
- ○ 1-Wire
- ○ PS/2
- ○ SPI
- ○ SMBus
- ○ I2S
- ○ Serial
- ○ Parallel

Signal: F E D C B A 9 8 7 6 5 4 3 2 1 0

Values

- ● Off
- ○ On

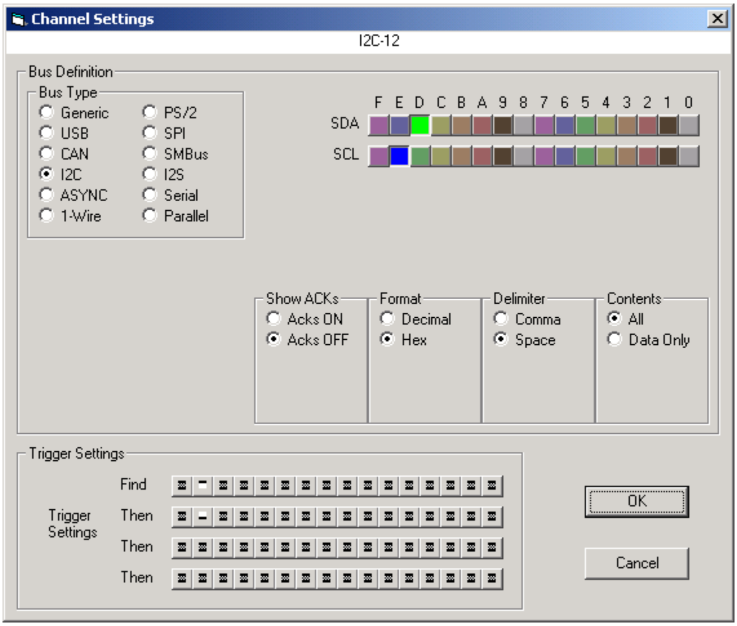
Format

- ○ Decimal
- ● Hex

Trigger Settings

Trigger Settings: Find / Then / Then / Then

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.



Channel Settings

I2C-12

Bus Definition

Bus Type

○ Generic ○ PS/2
○ USB ○ SPI
○ CAN ○ SMBus
● I2C ○ I2S
○ ASYNC ○ Serial
○ 1-Wire ○ Parallel

Show ACKs: ○ Acks ON ● Acks OFF

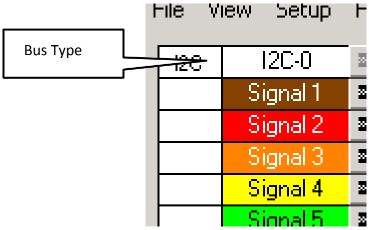
Format: ○ Decimal ● Hex

Delimiter: ○ Comma ● Space

Contents: ● All ○ Data Only

Trigger Settings

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

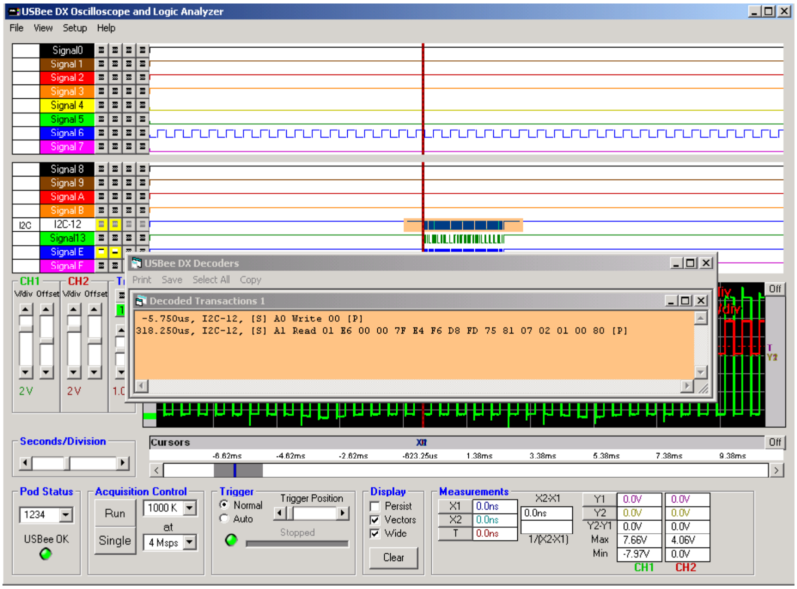


Each bus is renamed with the bus type followed by a number. This allows you to have many of the same types of busses, yet uniquely identify them in decoder listings.

DECODING BUS TRAFFIC – CLICK AND DRAG

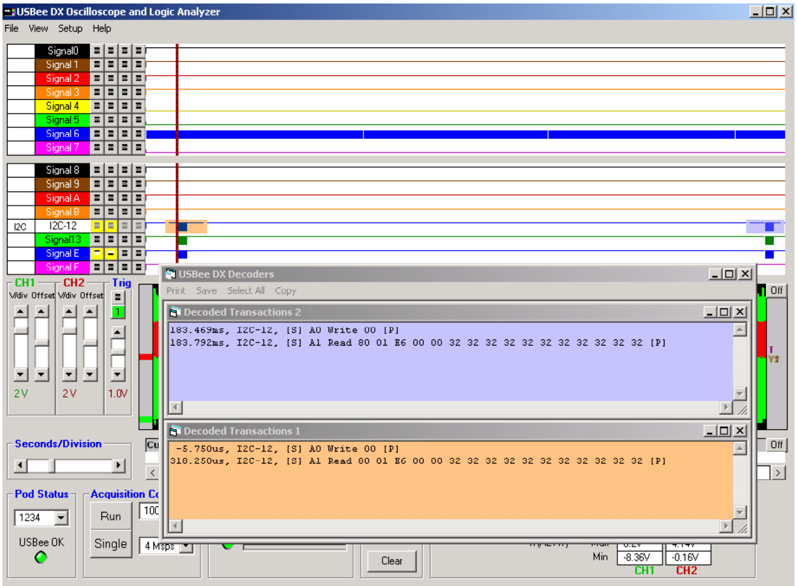
Once a bus is defined you can capture data as usual. You can then scroll and zoom to find the area of interest on that bus.

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.



```
 -5.750us, I2C-12, [S] A0 Write 00 [P]
318.250us, I2C-12, [S] A1 Read 01 E6 00 00 7F E4 F6 D8 FD 75 81 07 02 01 00 80 [P]
```

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.



Decoded Transactions 2:

```
183.469ms, I2C-12, [S] A0 Write 00 [P]
183.792ms, I2C-12, [S] A1 Read 80 01 E6 00 00 32 32 32 32 32 32 32 32 32 32 32 [P]
```

Decoded Transactions 1:

```
 -5.750us, I2C-12, [S] A0 Write 00 [P]
318.250us, I2C-12, [S] A1 Read 80 01 E6 00 00 32 32 32 32 32 32 32 32 32 32 32 [P]
```

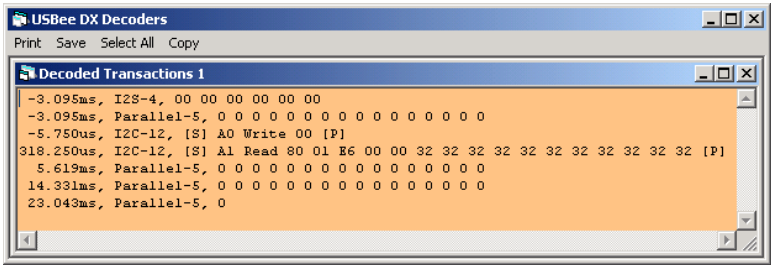
When you click on the text portion of the decode window, the main waveform screen will move to make sure that the decoded section for that window is displayed.

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.

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.

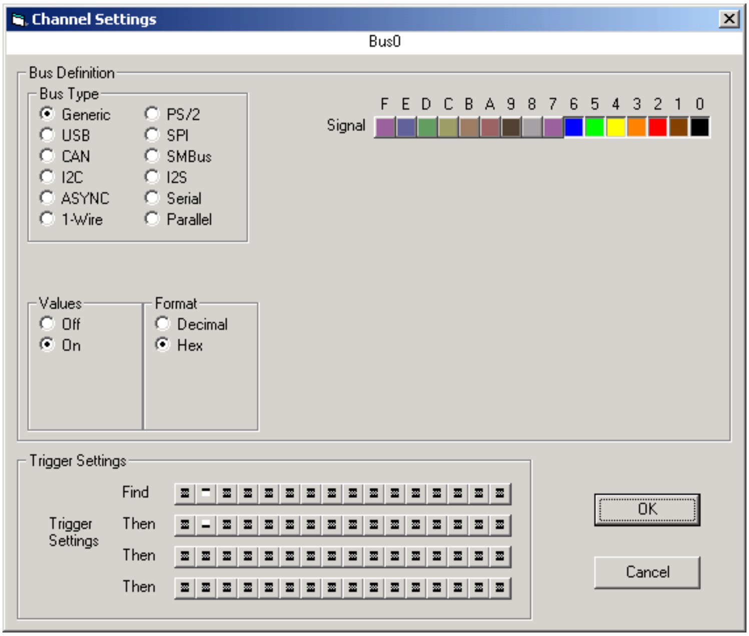


```
 -3.095ms, I2S-4, 00 00 00 00 00 00
 -3.095ms, Parallel-5, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 -5.750us, I2C-12, [S] A0 Write 00 [P]
318.250us, I2C-12, [S] A1 Read 80 01 E6 00 00 32 32 32 32 32 32 32 32 32 32 32 [P]
  5.619ms, Parallel-5, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 14.331ms, Parallel-5, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 23.043ms, Parallel-5, 0
```

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.



Bus0

Bus Definition

Bus Type: ● Generic ○ USB ○ CAN ○ I2C ○ ASYNC ○ 1-Wire ○ PS/2 ○ SPI ○ SMBus ○ I2S ○ Serial ○ Parallel

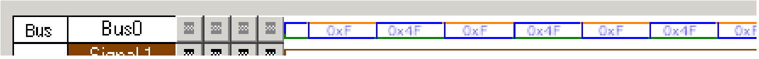
Signal F E D C B A 9 8 7 6 5 4 3 2 1 0

Values: ○ Off ● On

Format: ○ Decimal ● Hex

Trigger Settings: Find, Then, Then, Then

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.



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

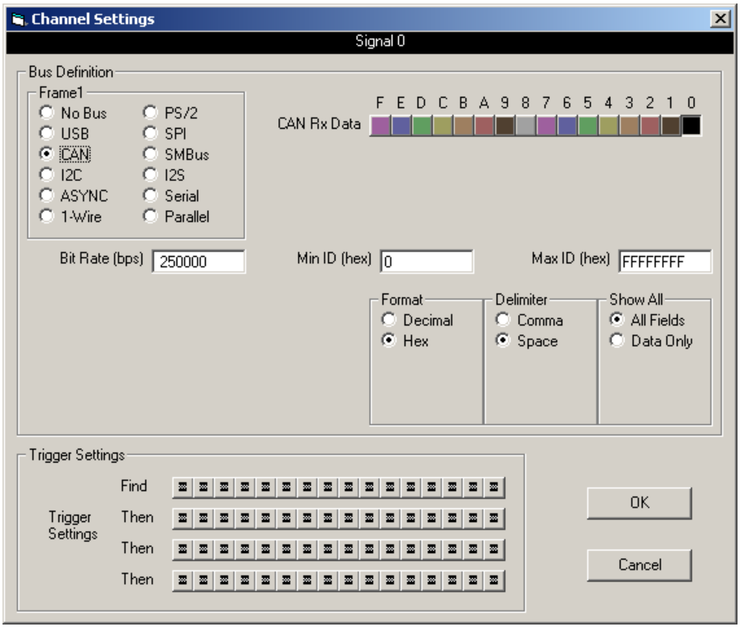
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

Signal 0

Bus Definition

Frame1

○ No Bus ○ PS/2
○ USB ○ SPI
● CAN ○ SMBus
○ I2C ○ I2S
○ ASYNC ○ Serial
○ 1-Wire ○ Parallel

Bit Rate (bps) 250000

Min ID (hex) 0

Max ID (hex) FFFFFFFF

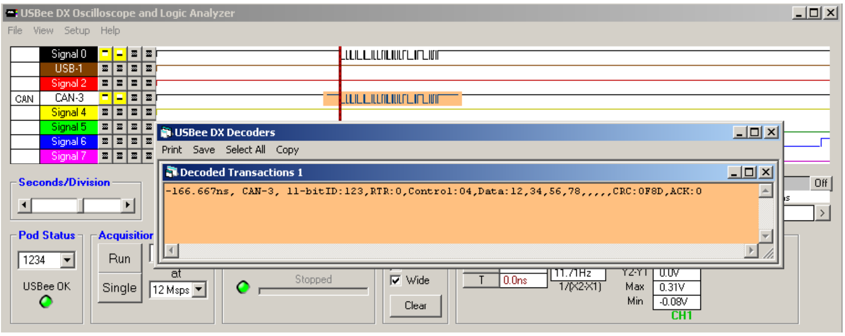
Format: ○ Decimal ● Hex

Delimiter: ○ Comma ● Space

Show All: ● All Fields ○ Data Only

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.



```
-166.667ns, CAN-3, 11-bitID:123,RTR:0,Control:04,Data:12,34,56,78,,,,,CRC:0F8D,ACK:0
```

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

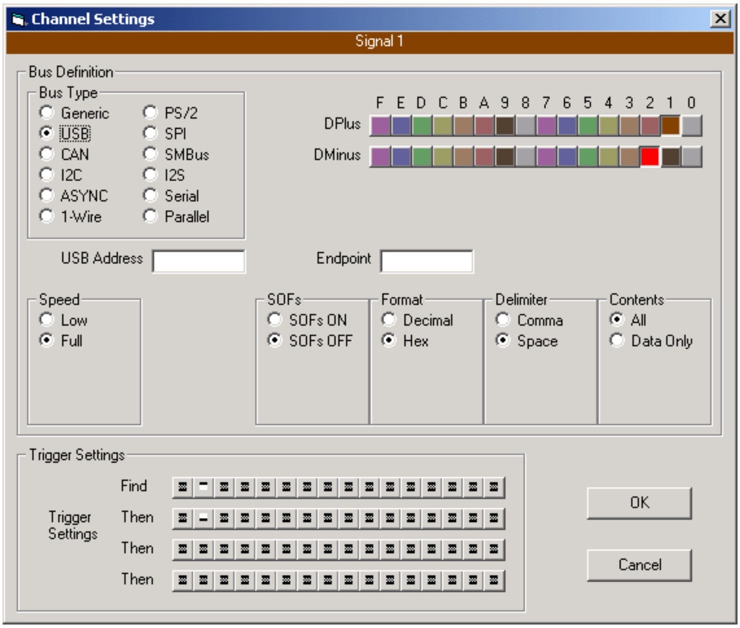
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

Signal 1

Bus Definition

Bus Type: ○ Generic ● USB ○ CAN ○ I2C ○ ASYNC ○ 1-Wire ○ PS/2 ○ SPI ○ SMBus ○ I2S ○ Serial ○ Parallel

DPlus, DMinus: F E D C B A 9 8 7 6 5 4 3 2 1 0

USB Address

Endpoint

Speed: ○ Low ● Full

SOFs: ○ SOFs ON ● SOFs OFF

Format: ○ Decimal ● Hex

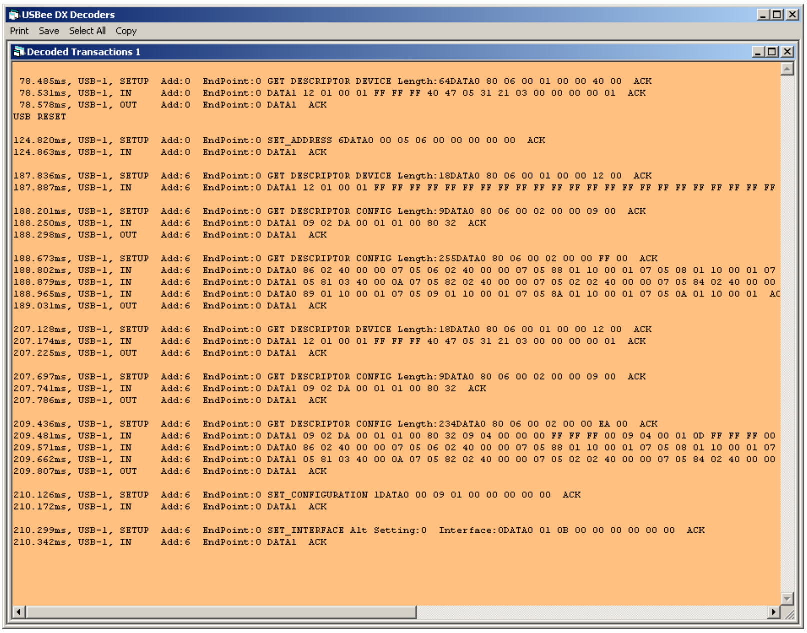
Delimiter: ○ Comma ● Space

Contents: ● All ○ Data Only

Trigger Settings

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.



```
 78.485ms, USB-1, SETUP  Add:0  EndPoint:0 GET DESCRIPTOR DEVICE Length:64DATA0 80 06 00 01 00 00 40 00  ACK
 78.531ms, USB-1, IN     Add:0  EndPoint:0 DATA1 12 01 00 01 FF FF FF 40 47 05 31 21 03 00 00 00 00 01  ACK
 78.578ms, USB-1, OUT    Add:0  EndPoint:0 DATA1  ACK
USB RESET

124.820ms, USB-1, SETUP  Add:0  EndPoint:0 SET_ADDRESS 6DATA0 00 05 06 00 00 00 00 00  ACK
124.863ms, USB-1, IN     Add:0  EndPoint:0 DATA1  ACK

187.836ms, USB-1, SETUP  Add:6  EndPoint:0 GET DESCRIPTOR DEVICE Length:18DATA0 80 06 00 01 00 00 12 00  ACK
187.887ms, USB-1, IN     Add:6  EndPoint:0 DATA1 12 01 00 01 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

188.201ms, USB-1, SETUP  Add:6  EndPoint:0 GET DESCRIPTOR CONFIG Length:9DATA0 80 06 00 02 00 00 09 00  ACK
188.250ms, USB-1, IN     Add:6  EndPoint:0 DATA1 09 02 DA 00 01 01 00 80 32  ACK
188.298ms, USB-1, OUT    Add:6  EndPoint:0 DATA1  ACK

188.673ms, USB-1, SETUP  Add:6  EndPoint:0 GET DESCRIPTOR CONFIG Length:255DATA0 80 06 00 02 00 00 FF 00  ACK
188.802ms, USB-1, IN     Add:6  EndPoint:0 DATA0 86 02 40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00 01 07 05 08 01 10 00 01 07
188.879ms, USB-1, IN     Add:6  EndPoint:0 DATA1 05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07 05 84 02 40 00 00
188.965ms, USB-1, IN     Add:6  EndPoint:0 DATA0 89 01 10 00 01 07 05 09 01 10 00 01 07 05 8A 01 10 00 01 07 05 0A 01 10 00 01  AC
189.031ms, USB-1, OUT    Add:6  EndPoint:0 DATA1  ACK

207.128ms, USB-1, SETUP  Add:6  EndPoint:0 GET DESCRIPTOR DEVICE Length:18DATA0 80 06 00 01 00 00 12 00  ACK
207.174ms, USB-1, IN     Add:6  EndPoint:0 DATA1 12 01 00 01 FF FF FF 40 47 05 31 21 03 00 00 00 00 01  ACK
207.225ms, USB-1, OUT    Add:6  EndPoint:0 DATA1  ACK

207.697ms, USB-1, SETUP  Add:6  EndPoint:0 GET DESCRIPTOR CONFIG Length:9DATA0 80 06 00 02 00 00 09 00  ACK
207.741ms, USB-1, IN     Add:6  EndPoint:0 DATA1 09 02 DA 00 01 01 00 80 32  ACK
207.786ms, USB-1, OUT    Add:6  EndPoint:0 DATA1  ACK

209.436ms, USB-1, SETUP  Add:6  EndPoint:0 GET DESCRIPTOR CONFIG Length:234DATA0 80 06 00 02 00 00 EA 00  ACK
209.481ms, USB-1, IN     Add:6  EndPoint:0 DATA1 09 02 DA 00 01 01 00 80 32 09 04 00 00 00 FF FF FF 00 09 04 00 01 0D FF FF FF 00
209.571ms, USB-1, IN     Add:6  EndPoint:0 DATA0 86 02 40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00 01 07 05 08 01 10 00 01 07
209.662ms, USB-1, IN     Add:6  EndPoint:0 DATA1 05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07 05 84 02 40 00 00
209.807ms, USB-1, OUT    Add:6  EndPoint:0 DATA1  ACK

210.126ms, USB-1, SETUP  Add:6  EndPoint:0 SET_CONFIGURATION 1DATA0 00 09 01 00 00 00 00 00  ACK
210.172ms, USB-1, IN     Add:6  EndPoint:0 DATA1  ACK

210.299ms, USB-1, SETUP  Add:6  EndPoint:0 SET_INTERFACE Alt Setting:0  Interface:0DATA0 01 0B 00 00 00 00 00 00  ACK
210.342ms, USB-1, IN     Add:6  EndPoint:0 DATA1  ACK
```

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

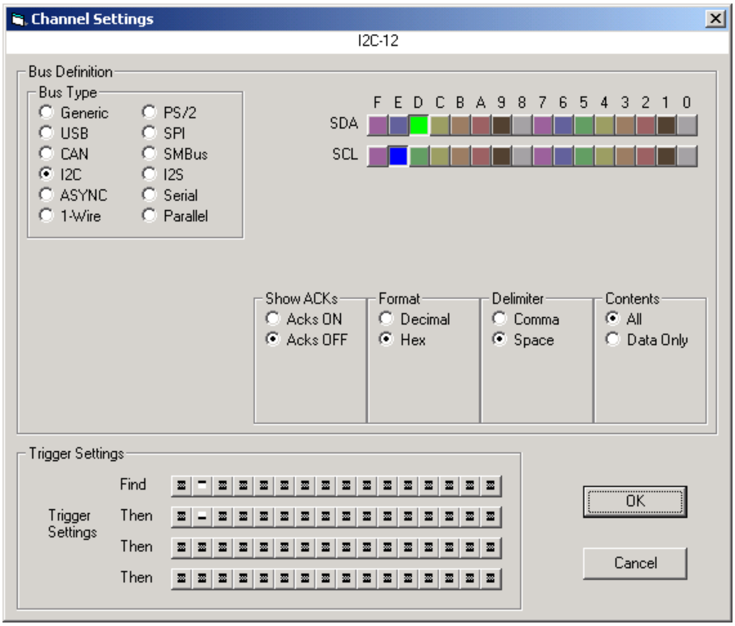
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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^{2}C Bus Decoder connects to the SDA and SCL lines of the I^{2}C 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^{2}C 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.



Channel Settings

I2C-12

Bus Definition

Bus Type

○ Generic ○ PS/2
○ USB ○ SPI
○ CAN ○ SMBus
● I2C ○ I2S
○ ASYNC ○ Serial
○ 1-Wire ○ Parallel

Show ACKs: ○ Acks ON ● Acks OFF

Format: ○ Decimal ● Hex

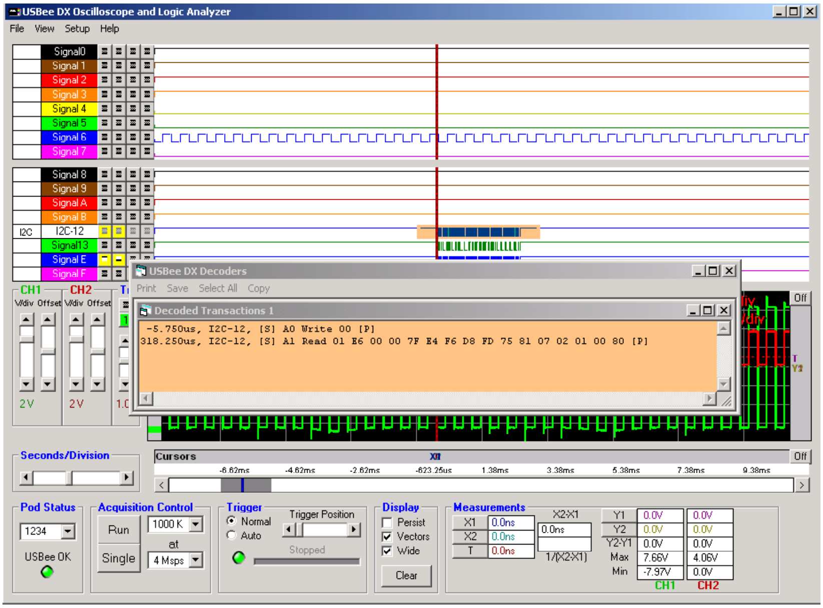
Delimiter: ○ Comma ● Space

Contents: ● All ○ Data Only

Trigger Settings

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.



```
 -5.750us, I2C-12, [S] A0 Write 00 [P]
318.250us, I2C-12, [S] A1 Read 01 E6 00 00 7F E4 F6 D8 FD 75 81 07 02 01 00 80 [P]
```

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

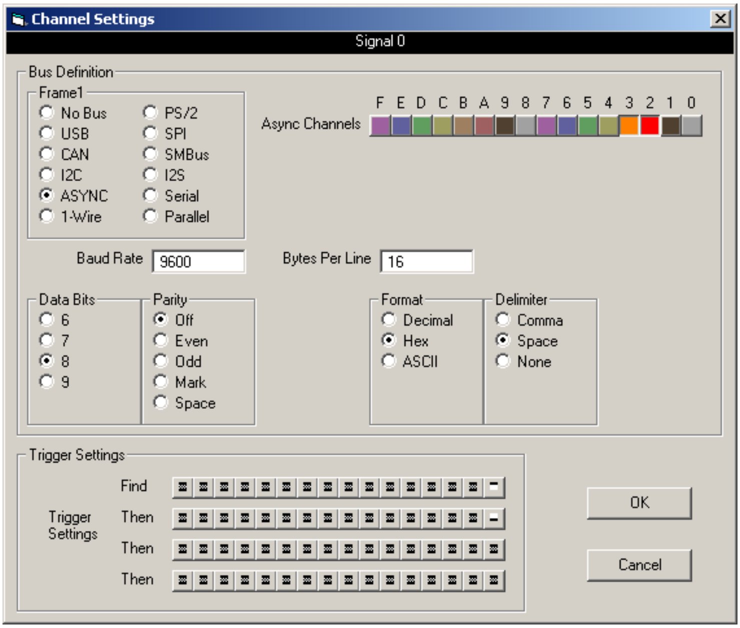
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

Signal 0

Bus Definition

Frame1

○ No Bus ○ PS/2
○ USB ○ SPI
○ CAN ○ SMBus
○ I2C ○ I2S
● ASYNC ○ Serial
○ 1-Wire ○ Parallel

Async Channels: F E D C B A 9 8 7 6 5 4 3 2 1 0

Baud Rate: 9600

Bytes Per Line: 16

Data Bits: ○ 6 ○ 7 ● 8 ○ 9

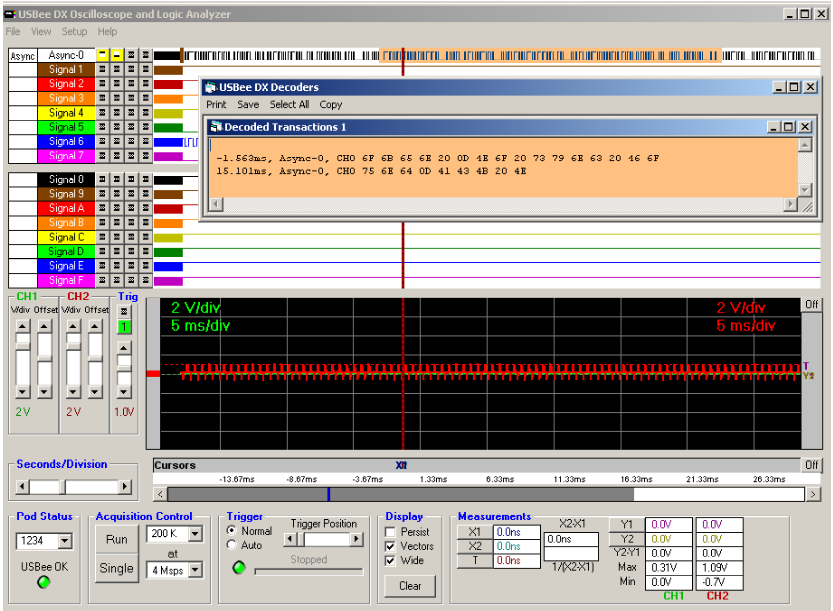
Parity: ● Off ○ Even ○ Odd ○ Mark ○ Space

Format: ○ Decimal ● Hex ○ ASCII

Delimiter: ○ Comma ● Space ○ None

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.



```
-1.563ms, Async-0, CH0 6F 6B 65 6E 20 0D 4E 6F 20 73 79 6E 63 20 46 6F
15.101ms, Async-0, CH0 75 6E 64 0D 41 43 4B 20 4E
```

Pod Status: 1234

Acquisition Control: 200 K at 4 Msps

Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

X1	0.0ns	
X2	0.0ns	
T	0.0ns	

X2-X1: 0.0ns

1/(X2-X1)

	CH1	CH2
Y1	0.0V	0.0V
Y2	0.0V	0.0V
Y2-Y1	0.0V	0.0V
Max	0.31V	1.09V
Min	0.0V	-0.7V

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

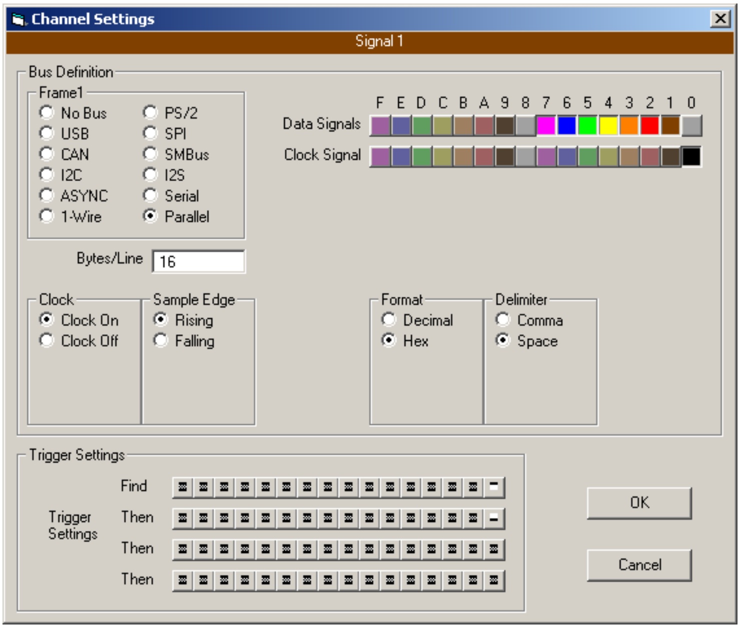
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

Signal 1

Bus Definition

Frame1

○ No Bus ○ PS/2
○ USB ○ SPI
○ CAN ○ SMBus
○ I2C ○ I2S
○ ASYNC ○ Serial
○ 1-Wire ● Parallel

Bytes/Line 16

Clock: ● Clock On ○ Clock Off

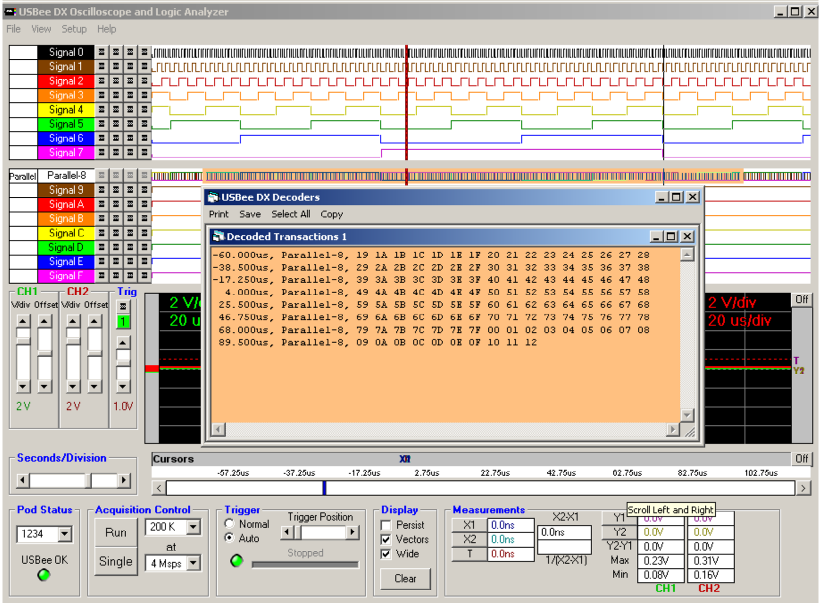
Sample Edge: ● Rising ○ Falling

Format: ○ Decimal ● Hex

Delimiter: ○ Comma ● Space

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.



```
-60.000us, Parallel-8, 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28
-38.500us, Parallel-8, 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38
-17.250us, Parallel-8, 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48
  4.000us, Parallel-8, 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58
 25.500us, Parallel-8, 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68
 46.750us, Parallel-8, 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78
 68.000us, Parallel-8, 79 7A 7B 7C 7D 7E 7F 00 01 02 03 04 05 06 07 08
 89.500us, Parallel-8, 09 0A 0B 0C 0D 0E 0F 10 11 12
```

Pod Status: 1234
Acquisition Control: 200 K at 4 Msps
Trigger: ○ Normal ● Auto
Display: □ Persist ☑ Vectors ☑ Wide

Measurements		X2-X1
X1	0.0ns	0.0ns
X2	0.0ns	1/(X2-X1)
T	0.0ns	

	CH1	CH2
Y1	0.0V	0.0V
Y2	0.0V	0.0V
Y2-Y1	0.0V	0.0V
Max	0.23V	0.31V
Min	0.08V	0.16V

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

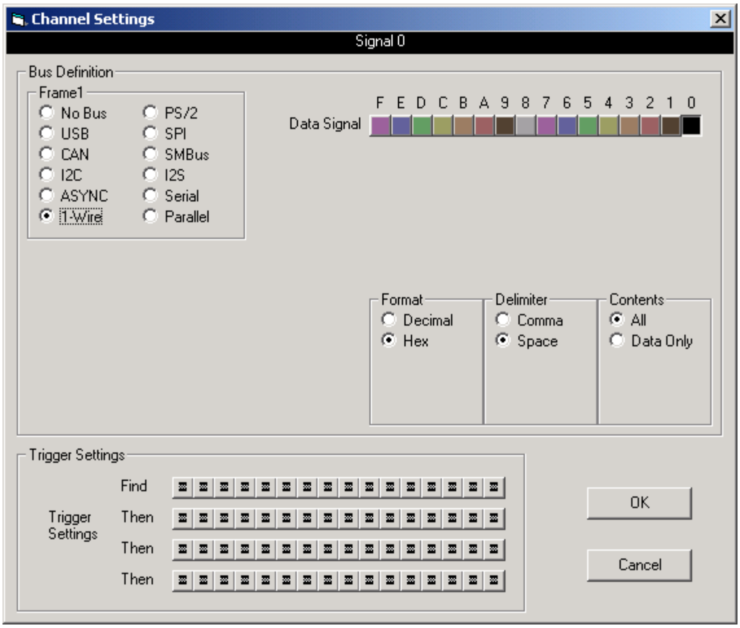
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

Signal 0

Bus Definition

Frame1

○ No Bus ○ PS/2
○ USB ○ SPI
○ CAN ○ SMBus
○ I2C ○ I2S
○ ASYNC ○ Serial
● 1-Wire ○ Parallel

Data Signal: F E D C B A 9 8 7 6 5 4 3 2 1 0 (0 selected)

Format: ○ Decimal ● Hex

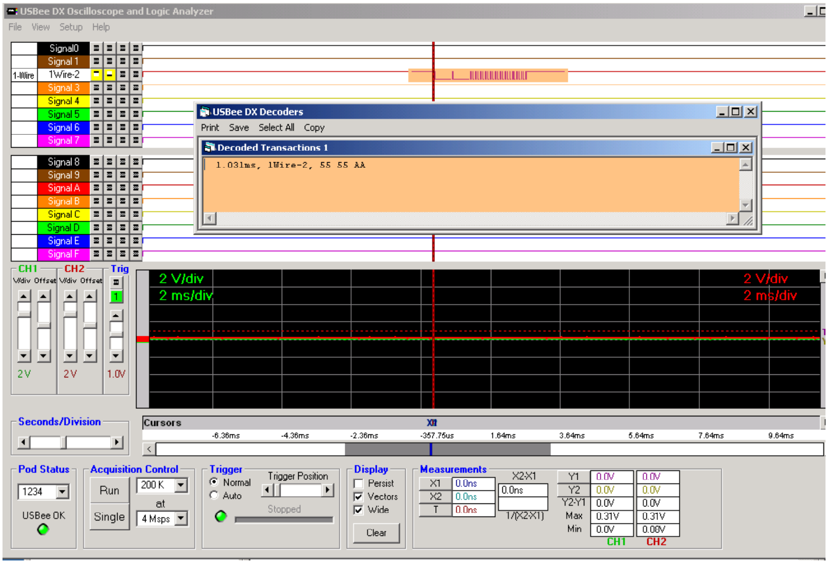
Delimiter: ○ Comma ● Space

Contents: ● All ○ Data Only

Trigger Settings

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.



```
1.031ms, 1Wire-2, 55 55 AA
```

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

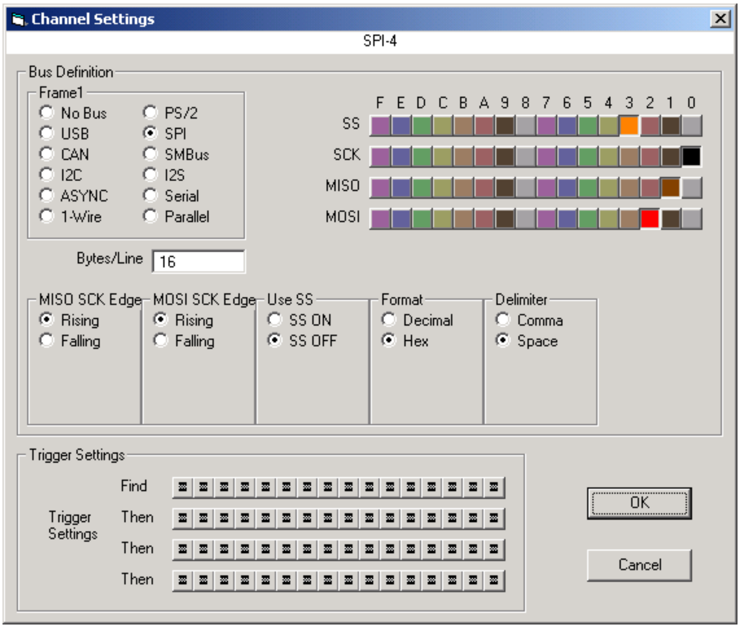
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

SPI-4

Bus Definition

Frame1

○ No Bus ○ PS/2
○ USB ● SPI
○ CAN ○ SMBus
○ I2C ○ I2S
○ ASYNC ○ Serial
○ 1-Wire ○ Parallel

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SS	□	□	□	□	□	□	□	□	□	□	□	□	■	□	□	□
SCK	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	■
MISO	□	□	□	□	□	□	□	□	□	□	□	□	□	□	■	□
MOSI	□	□	□	□	□	□	□	□	□	□	□	□	□	■	□	□

Bytes/Line 16

MISO SCK Edge: ● Rising ○ Falling

MOSI SCK Edge: ● Rising ○ Falling

Use SS: ○ SS ON ● SS OFF

Format: ○ Decimal ● Hex

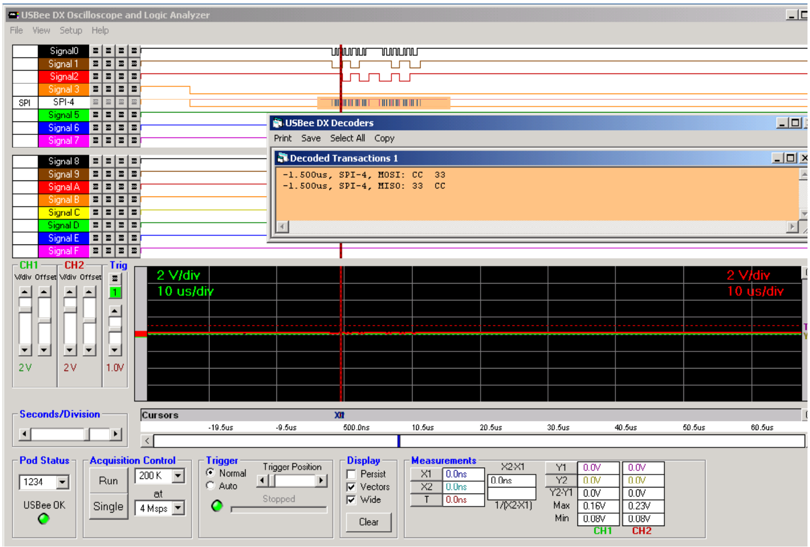
Delimiter: ○ Comma ● Space

Trigger Settings

Trigger Settings
Find ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒
Then ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒
Then ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒
Then ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒

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.



```
-1.500us, SPI-4, MOSI: CC  33
-1.500us, SPI-4, MISO: 33  CC
```

Pod Status: 1234, USBee OK

Acquisition Control: 200 K at 4 Msps

Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

		X2-X1		CH1	CH2
X1	0.0ns	0.0ns	Y1	0.0V	0.0V
X2	0.0ns		Y2	0.0V	0.0V
T	0.0ns	1/(X2-X1)	Y2-Y1	0.0V	0.0V
			Max	0.16V	0.23V
			Min	0.08V	0.08V

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

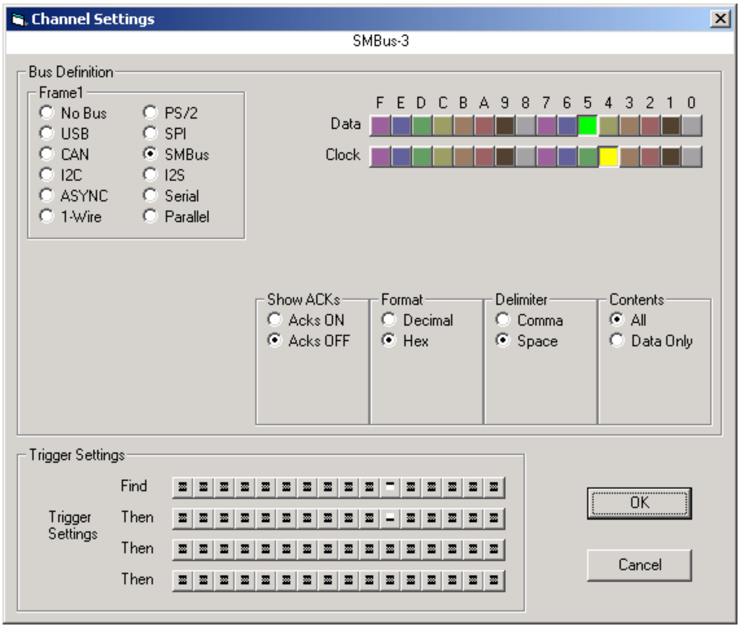
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

SMBus-3

Bus Definition

Frame1

○ No Bus ○ PS/2
○ USB ○ SPI
○ CAN ● SMBus
○ I2C ○ I2S
○ ASYNC ○ Serial
○ 1-Wire ○ Parallel

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Data											🟩					
Clock												🟨				

Show ACKs: ○ Acks ON ● Acks OFF

Format: ○ Decimal ● Hex

Delimiter: ○ Comma ● Space

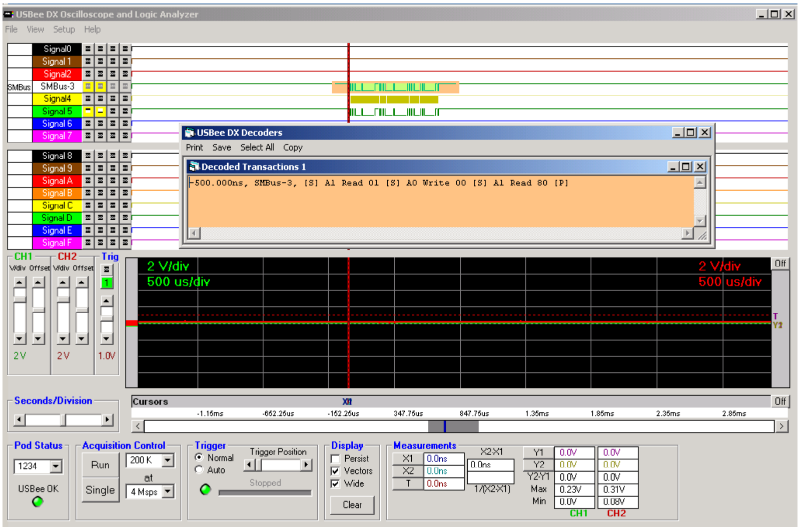
Contents: ● All ○ Data Only

Trigger Settings

Trigger Settings																	
Find	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	‾	☒	☒	☒	☒	☒
Then	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	_	☒	☒	☒	☒	☒
Then	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
Then	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒

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.



```
500.000ns, SMBus-3, [S] A1 Read 01 [S] A0 Write 00 [S] A1 Read 80 [P]
```

Pod Status: 1234 — USBee OK

Acquisition Control: 200 K at 4 Msps

Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

		X2-X1		CH1	CH2
X1	0.0ns	0.0ns	Y1	0.0V	0.0V
X2	0.0ns		Y2	0.0V	0.0V
T	0.0ns	1/(X2-X1)	Y2-Y1	0.0V	0.0V
			Max	0.23V	0.31V
			Min	0.0V	0.08V

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

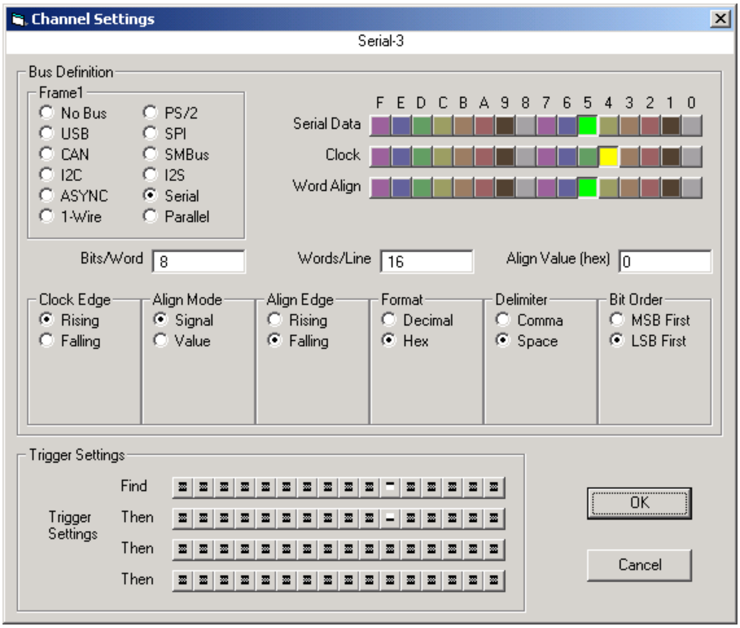
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



○ No Bus ○ USB ○ CAN ○ I2C ○ ASYNC ○ 1-Wire ○ PS/2 ○ SPI ○ SMBus ○ I2S ● Serial ○ Parallel

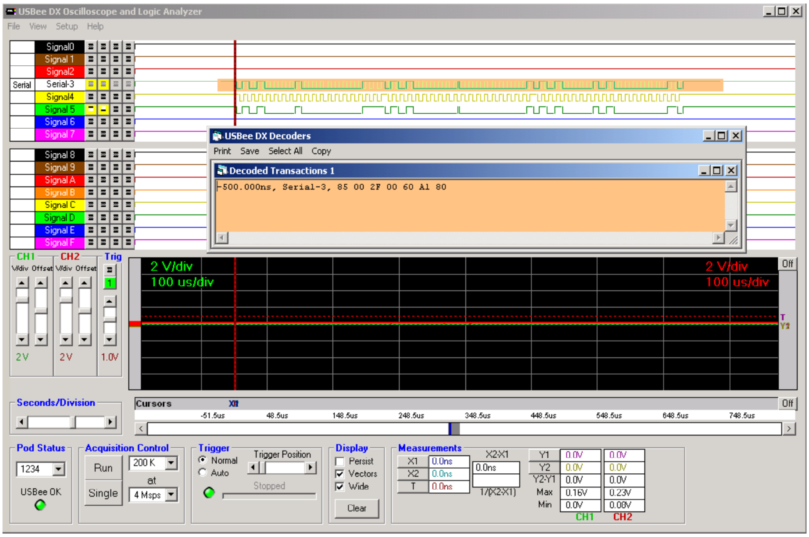
Serial Data: 5
Clock: 4
Word Align: 5

Bits/Word 8
Words/Line 16
Align Value (hex) 0

Clock Edge: ● Rising ○ Falling
Align Mode: ● Signal ○ Value
Align Edge: ○ Rising ● Falling
Format: ○ Decimal ● Hex
Delimiter: ○ Comma ● Space
Bit Order: ○ MSB First ● LSB First

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.



```
├500.000ns, Serial-3, 85 00 2F 00 60 A1 80
```

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

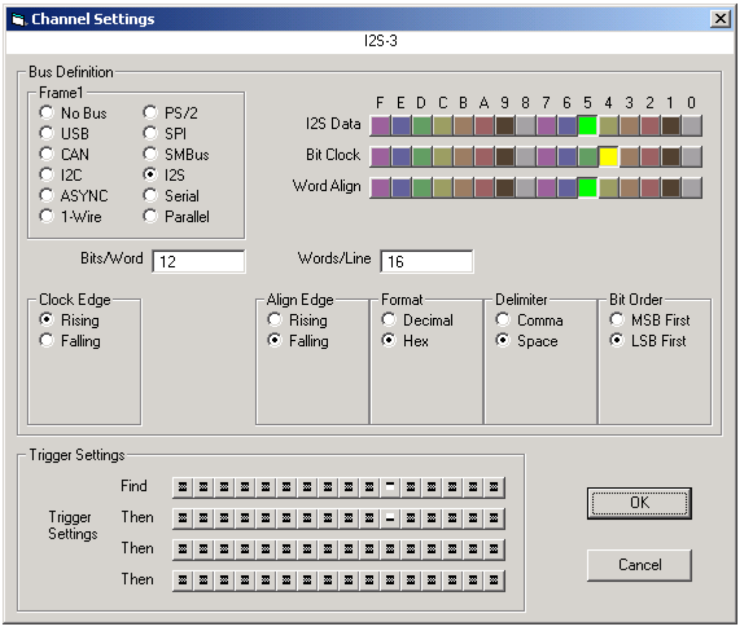
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

I2S-3

Bus Definition

Frame1

○ No Bus	○ PS/2
○ USB	○ SPI
○ CAN	○ SMBus
○ I2C	● I2S
○ ASYNC	○ Serial
○ 1-Wire	○ Parallel

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
I2S Data											🟩					
Bit Clock												🟨				
Word Align											🟩					

Bits/Word 12

Words/Line 16

Clock Edge: ● Rising ○ Falling

Align Edge: ○ Rising ● Falling

Format: ○ Decimal ● Hex

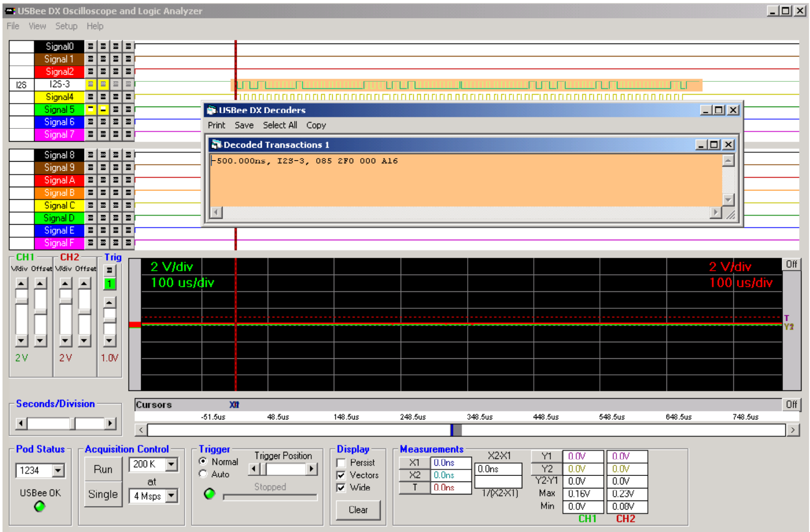
Delimiter: ○ Comma ● Space

Bit Order: ○ MSB First ● LSB First

Trigger Settings

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.



```
-500.000ns, I2S-3, 085 2F0 000 A16
```

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

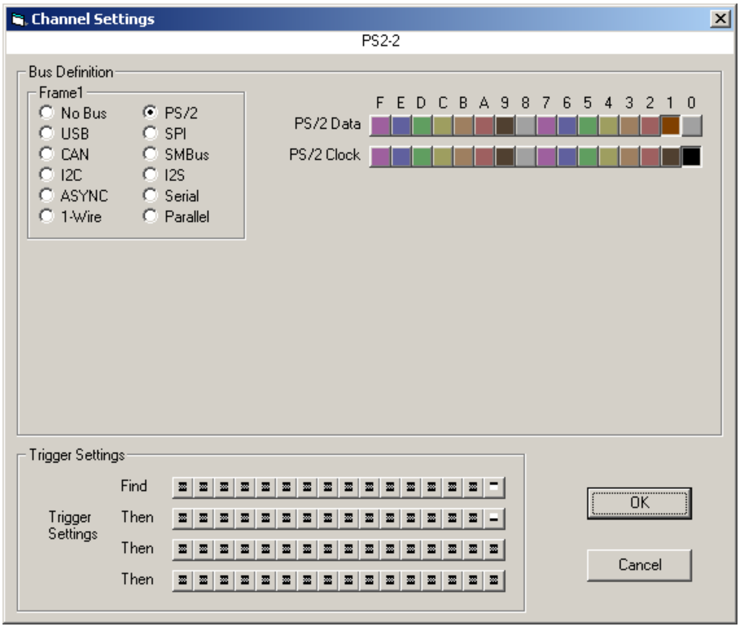
To use the Decoder you need to connect the USBee DX Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Channel Settings

PS2-2

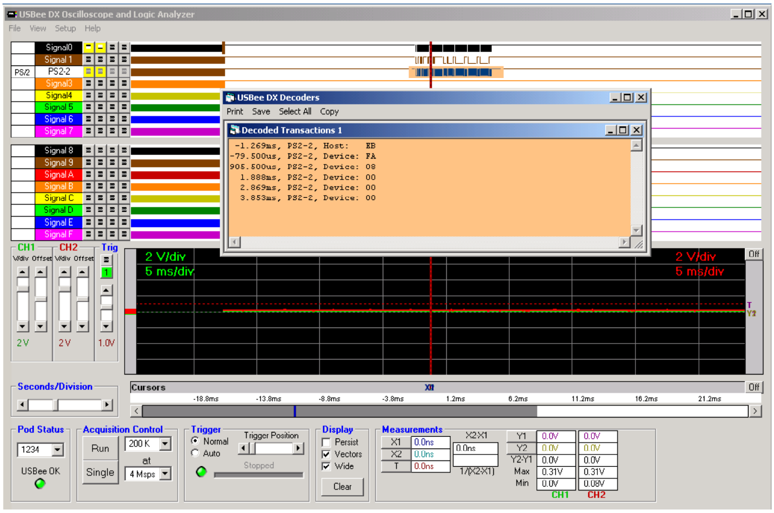
Bus Definition

Frame1

○ No Bus	● PS/2
○ USB	○ SPI
○ CAN	○ SMBus
○ I2C	○ I2S
○ ASYNC	○ Serial
○ 1-Wire	○ Parallel

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.



```
 -1.269ms, PS2-2, Host:   EB
-79.500us, PS2-2, Device: FA
905.500us, PS2-2, Device: 08
  1.888ms, PS2-2, Device: 00
  2.869ms, PS2-2, Device: 00
  3.853ms, PS2-2, Device: 00
```

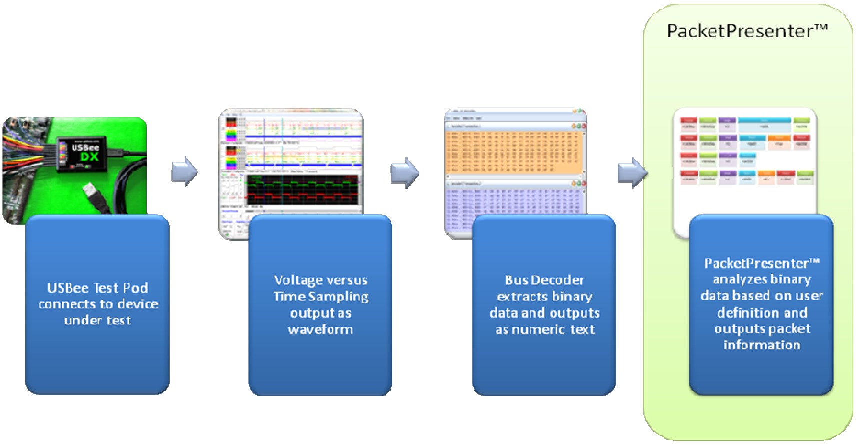
PACKETPRESENTER™

OVERVIEW

The USBee Test Pod functions as a number of standard electrical test equipment, such as Logic Analyzer, Oscilloscope and Signal Generator.

Using the Logic Analyzer/Oscilloscope application, it is normal for users to debug communication that is being transmitted between ICs or system components. This debugging can be performed by viewing the waveforms on the screen, or by viewing decoded bus traffic for the various types of busses. For example users can see the voltage versus time waveforms of an ASYNC bus Tx and Rx lines, or decode the waveform into a byte stream using the standard bus definition (ASYNC for example) that is then displayed in text form.

The PacketPresenter™ feature runs alongside of the existing bus decoders. The PacketPresenter™ takes the output of raw binary data from the bus decoder and parses the stream according to users PacketPresenter Definition File for the intent of displaying the communications in easily understood graphical displays.

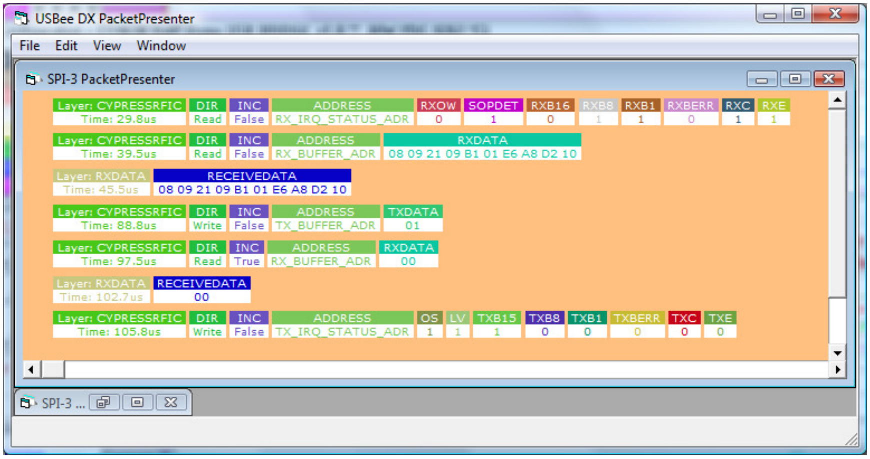

PacketPresenter™
USBee Test Pod connects to device under test
Voltage versus Time Sampling output as waveform
Bus Decoder extracts binary data and outputs as numeric text
PacketPresenter™ analyzes binary data based on user definition and outputs packet information

Protocols are defined using a text file, called a **PacketPresenter Definition File**, which specifies the fields within the protocol and how to display that information on the screen. It is intended to be generic enough that customers can create their own protocol decoders for their own custom bus types.

It is assumed that each **PacketPresenter Definition File** will correspond to one single bus type, and that the incoming bytes from that bus will be inputs for the decoding process. This steam of data is called an incoming **Data Stream** and it is handled by a **Protocol Processor**. Each Protocol Processor takes a single incoming Data Stream that is broken into **Packets**, parsed into **Fields** and either displayed as a field on the screen, ignored, and/or sent to a new Protocol for further processing (as in an N layer protocol).

Each Protocol Processor defines how to break the stream into Packets, and how to break the Packets into Fields. These Fields can then be displayed or sent to another Data Stream for further processing.

Below shows a sample PacketPresenter output screen.



Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXOW	SOPDET	RXB16	RXB8	RXB1	RXBERR	RXC	RXE
Time: 29.8us	Read	False	RX_IRQ_STATUS_ADR	0	1	0	1	1	0	1	1

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXDATA
Time: 39.5us	Read	False	RX_BUFFER_ADR	08 09 21 09 B1 01 E6 A8 D2 10

Layer: RXDATA	RECEIVEDATA
Time: 45.5us	08 09 21 09 B1 01 E6 A8 D2 10

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	TXDATA
Time: 88.8us	Write	False	TX_BUFFER_ADR	01

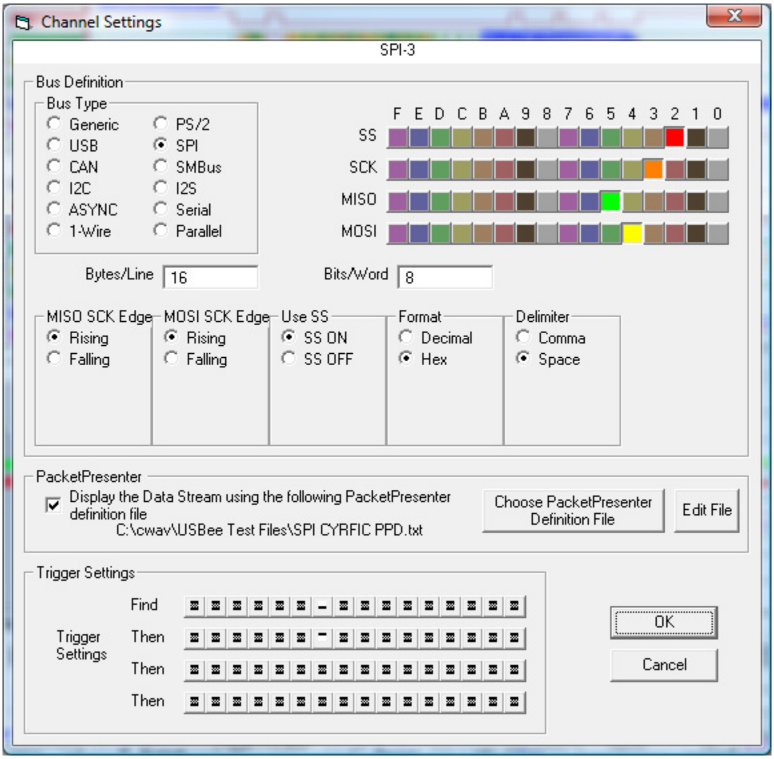
Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXDATA
Time: 97.5us	Read	True	RX_BUFFER_ADR	00

Layer: RXDATA	RECEIVEDATA
Time: 102.7us	00

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	OS	LV	TXB15	TXB8	TXB1	TXBERR	TXC	TXE
Time: 105.8us	Write	False	TX_IRQ_STATUS_ADR	1	1	1	0	0	0	0	0

SETTING UP THE PACKETPRESENTER

Each digital waveform on the screen can be defined as a different bus (I2C, SPI, etc.) in the Channel settings dialog box by clicking on the white box to the left of the signal name. Below shows the Channel Settings dialog box.



Channel Settings

SPI-3

Bus Definition

Bus Type

○ Generic ○ PS/2
○ USB ● SPI
○ CAN ○ SMBus
○ I2C ○ I2S
○ ASYNC ○ Serial
○ 1-Wire ○ Parallel

Bytes/Line 16

Bits/Word 8

MISO SCK Edge: ● Rising ○ Falling

MOSI SCK Edge: ● Rising ○ Falling

Use SS: ● SS ON ○ SS OFF

Format: ○ Decimal ● Hex

Delimiter: ○ Comma ● Space

PacketPresenter

☑ Display the Data Stream using the following PacketPresenter definition file
C:\cwav\USBee Test Files\SPI CYRFIC PPD.txt

Trigger Settings

F E D C B A 9 8 7 6 5 4 3 2 1 0
SS
SCK
MISO
MOSI
Choose PacketPresenter Definition File
Edit File
Trigger Settings
Find
Then
Then
Then
OK
Cancel

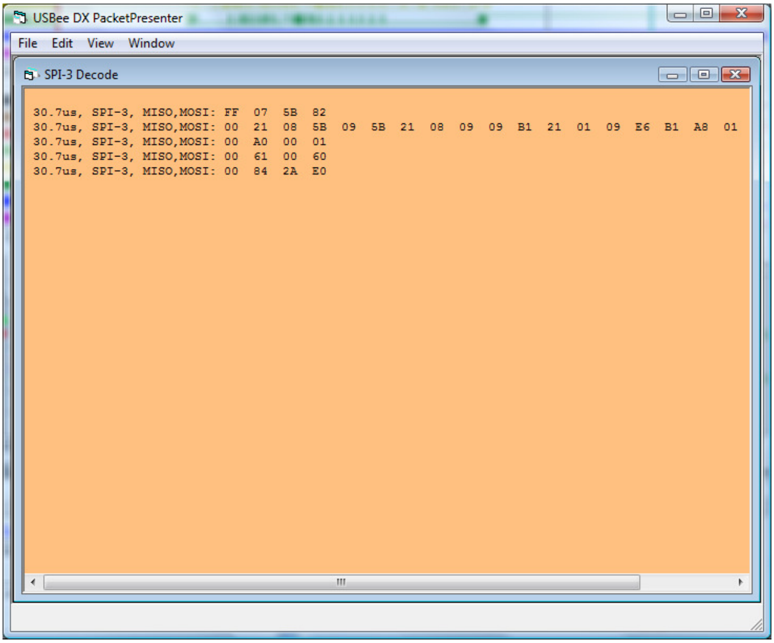
To enable the PacketPresenter for this channel, check the “Display the Data Stream using the following PacketPresenter definition file” checkbox. Then choose the PacketPresenter definition file by clicking the button to the right. Once you choose the file, you can edit the contents by clicking the “Edit File” button.

Once the PacketPresenter is enabled all bus decodes will be processed through the PacketPresenter as well as the original bus decoder.

VIEWING THE PACKETPRESENTER OUTPUT

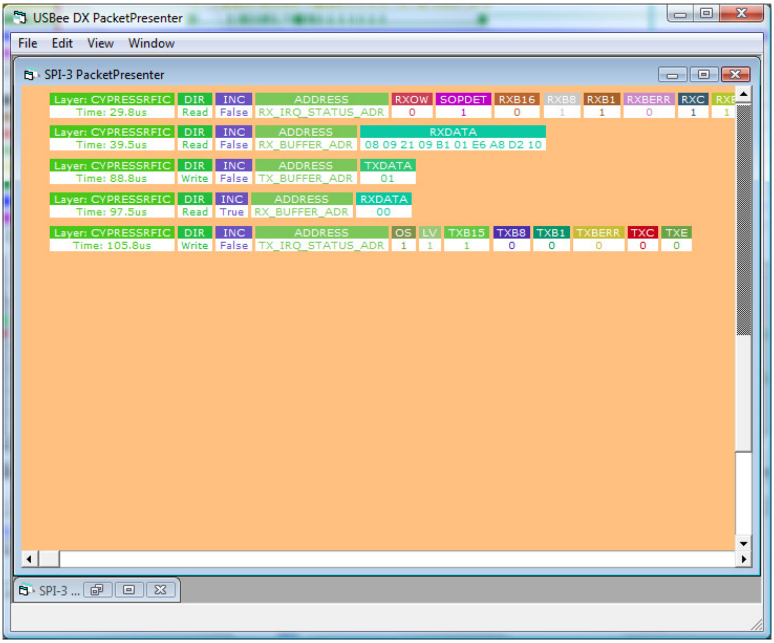
Once the bus is defined and the PacketPresenter is setup with a PacketPresenter definition file, right clicking and dragging on the waveform will not only decode the raw data from the bus (as specified in the Channel Settings), but will also parse the data based on your PacketPresenter definition file.

If the PacketPresenter is not enabled, only the decoded data is shown as seen below.



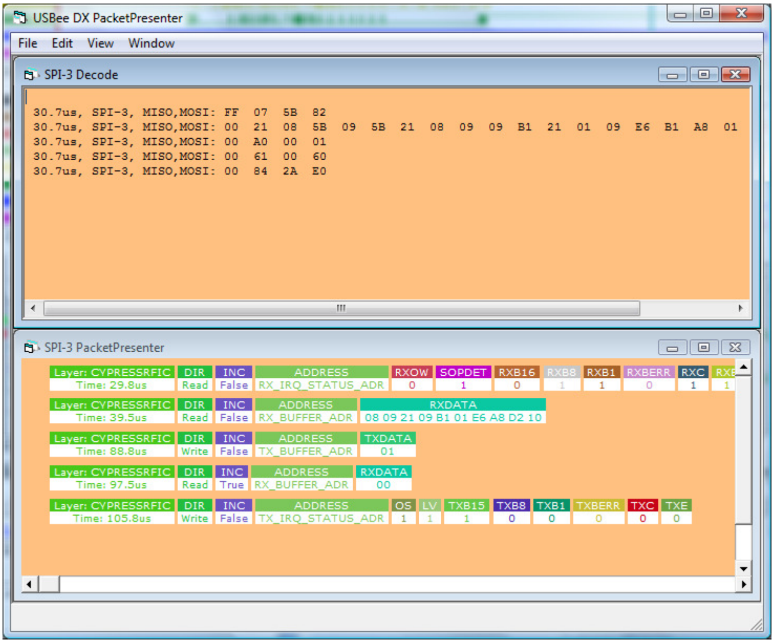
```
30.7us, SPI-3, MISO,MOSI: FF  07  5B  82
30.7us, SPI-3, MISO,MOSI: 00  21  08  5B  09  5B  21  08  09  09  B1  21  01  09  E6  B1  A8  01
30.7us, SPI-3, MISO,MOSI: 00  A0  00  01
30.7us, SPI-3, MISO,MOSI: 00  61  00  60
30.7us, SPI-3, MISO,MOSI: 00  84  2A  E0
```

Enablng the PacketPresenter shows the PacketPresenter output, with the original decoded data in a minimized window as in the following screenshot.



```
Layer: CYPRESSRFIC  DIR   INC    ADDRESS             RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RXE
  Time: 29.8us      Read  False  RX_IRQ_STATUS_ADR   0     1       0      1     1     0       1    1

Layer: CYPRESSRFIC  DIR   INC    ADDRESS          RXDATA
  Time: 39.5us      Read  False  RX_BUFFER_ADR    08 09 21 09 B1 01 E6 A8 D2 10

Layer: CYPRESSRFIC  DIR   INC    ADDRESS          TXDATA
  Time: 88.8us      Write False  TX_BUFFER_ADR    01

Layer: CYPRESSRFIC  DIR   INC    ADDRESS          RXDATA
  Time: 97.5us      Read  True   RX_BUFFER_ADR    00

Layer: CYPRESSRFIC  DIR   INC    ADDRESS             OS  LV  TXB15  TXB8  TXB1  TXBERR  TXC  TXE
  Time: 105.8us     Write False  TX_IRQ_STATUS_ADR   1   1   1      0     0     0       0    0
```

You can show the raw decoded data at the same time by restoring the minimized window as shown in the following screenshot.



```
30.7us, SPI-3, MISO,MOSI: FF  07  5B  82
30.7us, SPI-3, MISO,MOSI: 00  21  08  5B  09  5B  21  08  09  09  B1  21  01  09  E6  B1  A8  01
30.7us, SPI-3, MISO,MOSI: 00  A0  00  01
30.7us, SPI-3, MISO,MOSI: 00  61  00  60
30.7us, SPI-3, MISO,MOSI: 00  84  2A  E0
```

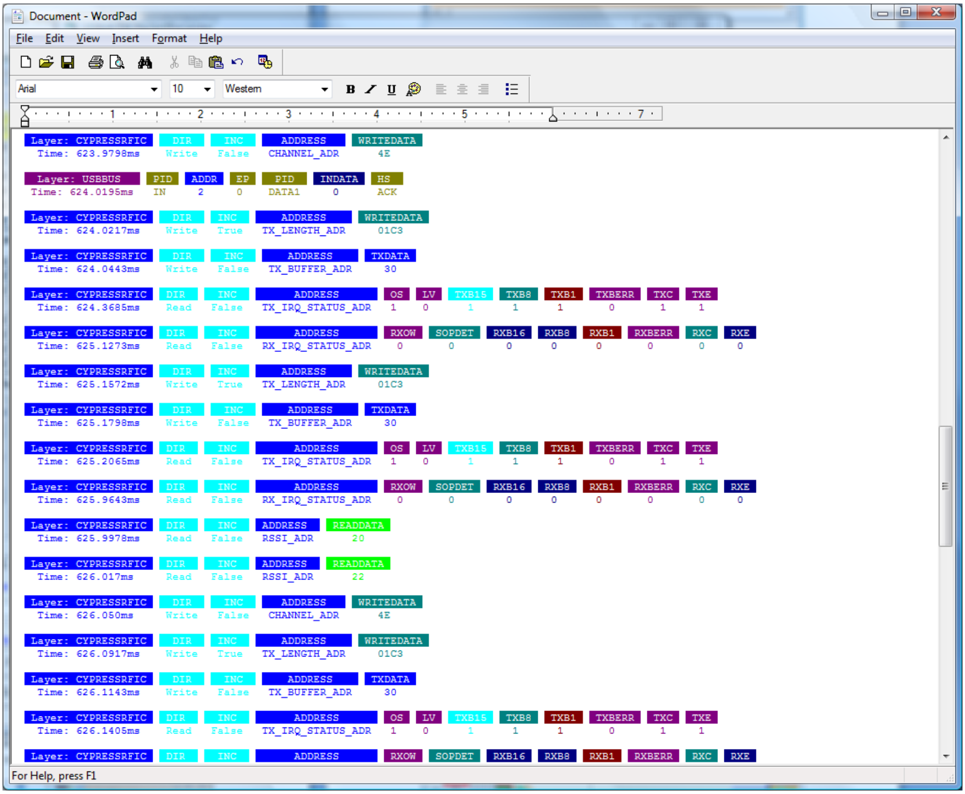
```
Layer: CYPRESSRFIC  DIR   INC    ADDRESS             RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RXE
 Time: 29.8us       Read  False  RX_IRQ_STATUS_ADR   0     1       0      1     1     0       1    1

Layer: CYPRESSRFIC  DIR   INC    ADDRESS          RXDATA
 Time: 39.5us       Read  False  RX_BUFFER_ADR    08 09 21 09 B1 01 E6 A8 D2 10

Layer: CYPRESSRFIC  DIR   INC    ADDRESS          TXDATA
 Time: 88.8us       Write False  TX_BUFFER_ADR    01

Layer: CYPRESSRFIC  DIR   INC    ADDRESS          RXDATA
 Time: 97.5us       Read  True   RX_BUFFER_ADR    00

Layer: CYPRESSRFIC  DIR   INC    ADDRESS             OS  LV  TXB15  TXB8  TXB1  TXBERR  TXC  TXE
 Time: 105.8us      Write False  TX_IRQ_STATUS_ADR   1   1   1      0     0     0       0    0
```

SAVING PACKETPRESENTER DATA TO TEXT OR RTF FILES

The PacketPresenter output can be saved to either a Text file or an RTF file (Rich Text Format). The text file output is a textual representation of the packets as seen below.

```
Layer: CYPRESSRFIC   DIR     INC      ADDRESS      READDATA
 Time: 615.2797ms    Read   False   CHANNEL_ADR       0

 Layer: USBBUS      PID    ADDR    EP     PID             INDATA             HS
Time: 616.0198ms    IN       2     0     DATA0    22 2A 00 07 05 81 03 08    ACK

 Layer: USBBUS      PID    ADDR    EP     PID             INDATA             HS
Time: 617.0197ms    IN       2     0     DATA1    00 0A 09 04 01 00 01 03    ACK

 Layer: USBBUS      PID    ADDR    EP     PID             INDATA             HS
Time: 618.0197ms    IN       2     0     DATA0    01 02 00 09 21 11 01 00    ACK

 Layer: USBBUS      PID    ADDR    EP     PID             INDATA             HS
Time: 619.0197ms    IN       2     0     DATA1    01 22 D1 00 07 05 82 03    ACK

 Layer: USBBUS      PID    ADDR    EP     PID     INDATA    HS
Time: 620.0197ms    IN       2     0     DATA0    0A0008    ACK
```

Saving data to an RTF file format saves the graphical nature of the packets and can be read by many word processing programs, such as Microsoft Word and WordPad. Below is a screenshot of data saved to an RFT file and viewed using WordPad.



```
 Layer: CYPRESSRFIC   DIR    INC      ADDRESS        WRITEDATA
  Time: 623.9798ms   Write  False   CHANNEL_ADR        4E

 Layer: USBBUS      PID   ADDR   EP    PID    INDATA   HS
 Time: 624.0195ms   IN     2     0    DATA1     0     ACK

 Layer: CYPRESSRFIC   DIR    INC      ADDRESS        WRITEDATA
  Time: 624.0217ms   Write  True    TX_LENGTH_ADR      01C3

 Layer: CYPRESSRFIC   DIR    INC      ADDRESS        TXDATA
  Time: 624.0443ms   Write  False   TX_BUFFER_ADR      30

 Layer: CYPRESSRFIC   DIR    INC        ADDRESS          OS  LV  TXB15  TXB8  TXB1  TXBERR  TXC  TXE
  Time: 624.3685ms   Read   False   TX_IRQ_STATUS_ADR    1   0     1     1     1      0     1    1

 Layer: CYPRESSRFIC   DIR    INC        ADDRESS          RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RXE
  Time: 625.1273ms   Read   False   RX_IRQ_STATUS_ADR     0      0       0      0     0      0     0    0

 Layer: CYPRESSRFIC   DIR    INC      ADDRESS        WRITEDATA
  Time: 625.1572ms   Write  True    TX_LENGTH_ADR      01C3

 Layer: CYPRESSRFIC   DIR    INC      ADDRESS        TXDATA
  Time: 625.1798ms   Write  False   TX_BUFFER_ADR      30

 Layer: CYPRESSRFIC   DIR    INC        ADDRESS          OS  LV  TXB15  TXB8  TXB1  TXBERR  TXC  TXE
  Time: 625.2065ms   Read   False   TX_IRQ_STATUS_ADR    1   0     1     1     1      0     1    1

 Layer: CYPRESSRFIC   DIR    INC        ADDRESS          RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RXE
  Time: 625.9643ms   Read   False   RX_IRQ_STATUS_ADR     0      0       0      0     0      0     0    0

 Layer: CYPRESSRFIC   DIR    INC    ADDRESS   READDATA
  Time: 625.9978ms   Read   False   RSSI_ADR     20

 Layer: CYPRESSRFIC   DIR    INC    ADDRESS   READDATA
  Time: 626.017ms    Read   False   RSSI_ADR     22

 Layer: CYPRESSRFIC   DIR    INC      ADDRESS      WRITEDATA
  Time: 626.050ms    Write  False   CHANNEL_ADR      4E

 Layer: CYPRESSRFIC   DIR    INC      ADDRESS        WRITEDATA
  Time: 626.0917ms   Write  True    TX_LENGTH_ADR      01C3

 Layer: CYPRESSRFIC   DIR    INC      ADDRESS        TXDATA
  Time: 626.1143ms   Write  False   TX_BUFFER_ADR      30

 Layer: CYPRESSRFIC   DIR    INC        ADDRESS          OS  LV  TXB15  TXB8  TXB1  TXBERR  TXC  TXE
  Time: 626.1405ms   Read   False   TX_IRQ_STATUS_ADR    1   0     1     1     1      0     1    1

 Layer: CYPRESSRFIC   DIR    INC        ADDRESS          RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RXE
```

In order to maintain correct position of the graphical portions of the RTF file, all spaces are converted to the character "~" and set to the background color. Viewed or printed in the RTF format will look correct as above. If you copy only the text of this output, you will want to search and replace every "~" with a space.

COPYING PACKETPRESENTER OUTPUT TO OTHER PROGRAMS

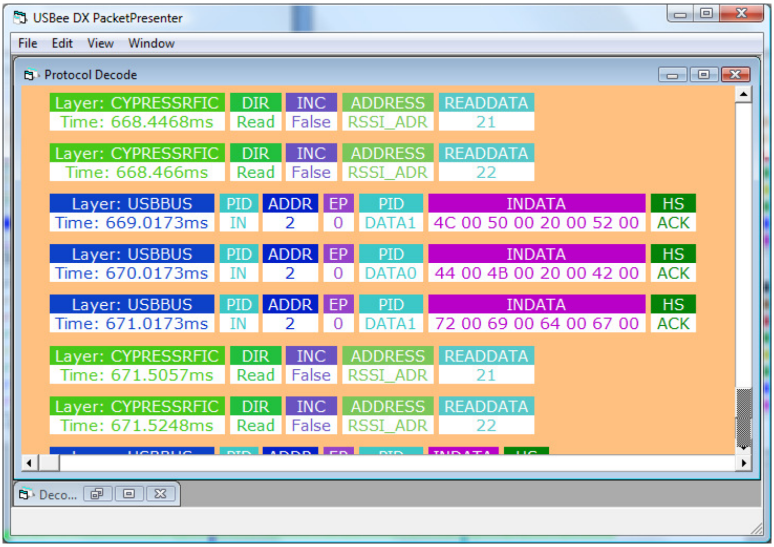
You can copy the contents of the PacketPresenter output window to other programs in a number of ways.

First, you can copy the screenshot of the window by selecting the window and pressing Alt-PrtScr on your keyboard. This copies the image of the window to the Windows clipboard and you can paste that image into any program that accepts images.

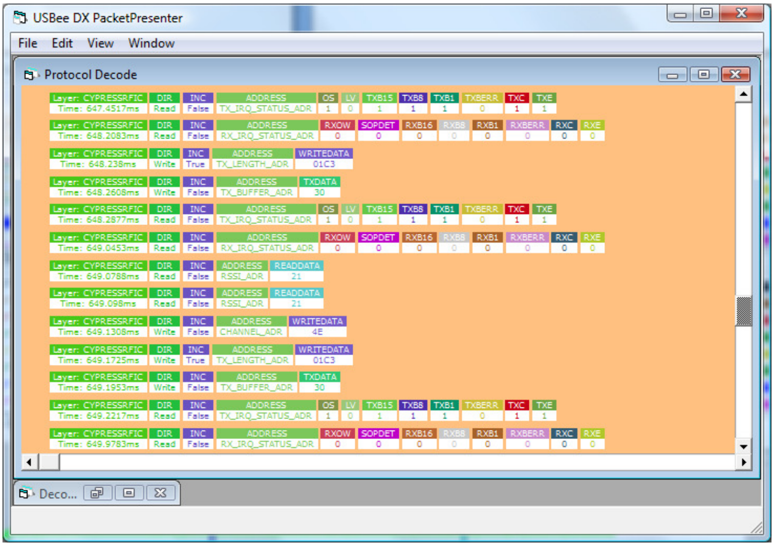
You can also use the Edit | Copy menu item. If the textual decode data window is active, the selected text is copied to the clipboard. To select text, just click and drag across the text you would like to highlight. If the PacketPresenter output window is highlighted, all packets starting with the packet at the top of the window are copied to the clipboard. When pasting the data to other programs, it will paste the data as an RTF file if possible and text otherwise.

CHANGING THE PACKETPRESENTER SIZE

You can change the size of the fonts used by the PacketPresenter by selecting the View | Larger or View | Smaller menu items. Below are examples of different size fonts.



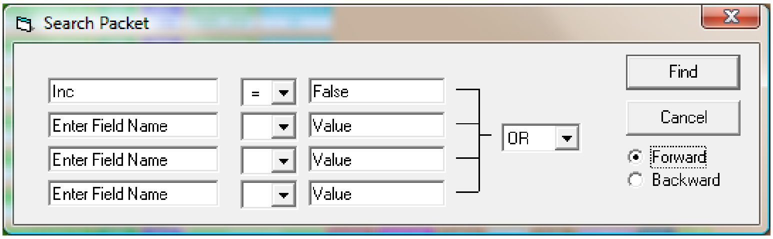
```
Layer: CYPRESSRFIC   | DIR  | INC   | ADDRESS  | READDATA
Time: 668.4468ms     | Read | False | RSSI_ADR | 21

Layer: CYPRESSRFIC   | DIR  | INC   | ADDRESS  | READDATA
Time: 668.466ms      | Read | False | RSSI_ADR | 22

Layer: USBBUS        | PID | ADDR | EP | PID   | INDATA                  | HS
Time: 669.0173ms     | IN  | 2    | 0  | DATA1 | 4C 00 50 00 20 00 52 00 | ACK

Layer: USBBUS        | PID | ADDR | EP | PID   | INDATA                  | HS
Time: 670.0173ms     | IN  | 2    | 0  | DATA0 | 44 00 4B 00 20 00 42 00 | ACK

Layer: USBBUS        | PID | ADDR | EP | PID   | INDATA                  | HS
Time: 671.0173ms     | IN  | 2    | 0  | DATA1 | 72 00 69 00 64 00 67 00 | ACK

Layer: CYPRESSRFIC   | DIR  | INC   | ADDRESS  | READDATA
Time: 671.5057ms     | Read | False | RSSI_ADR | 21

Layer: CYPRESSRFIC   | DIR  | INC   | ADDRESS  | READDATA
Time: 671.5248ms     | Read | False | RSSI_ADR | 22
```



```
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | OS | LV | TXB15 | TXB8 | TXB1 | TXBERR | TXC | TXE
Time: 647.4517ms   | Read  | False | TX_IRQ_STATUS_ADR  | 1  | 0  | 1     | 1    | 1    | 0      | 1   | 1

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | RXOW | SOPDET | RXB16 | RXB8 | RXB1 | RXBERR | RXC | RXE
Time: 648.2083ms   | Read  | False | RX_IRQ_STATUS_ADR  | 0    | 0      | 0     | 0    | 0    | 0      | 0   | 0

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | WRITEDATA
Time: 648.238ms    | Write | True  | TX_LENGTH_ADR      | 01C3

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | TXDATA
Time: 648.2608ms   | Write | False | TX_BUFFER_ADR      | 30

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | OS | LV | TXB15 | TXB8 | TXB1 | TXBERR | TXC | TXE
Time: 648.2877ms   | Read  | False | TX_IRQ_STATUS_ADR  | 1  | 0  | 1     | 1    | 1    | 0      | 1   | 1

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | RXOW | SOPDET | RXB16 | RXB8 | RXB1 | RXBERR | RXC | RXE
Time: 649.0453ms   | Read  | False | RX_IRQ_STATUS_ADR  | 0    | 0      | 0     | 0    | 0    | 0      | 0   | 0

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | READDATA
Time: 649.0788ms   | Read  | False | RSSI_ADR           | 21

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | READDATA
Time: 649.098ms    | Read  | False | RSSI_ADR           | 21

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | WRITEDATA
Time: 649.1308ms   | Write | False | CHANNEL_ADR        | 4E

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | WRITEDATA
Time: 649.1725ms   | Write | True  | TX_LENGTH_ADR      | 01C3

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | TXDATA
Time: 649.1953ms   | Write | False | TX_BUFFER_ADR      | 30

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | OS | LV | TXB15 | TXB8 | TXB1 | TXBERR | TXC | TXE
Time: 649.2217ms   | Read  | False | TX_IRQ_STATUS_ADR  | 1  | 0  | 1     | 1    | 1    | 0      | 1   | 1

Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | RXOW | SOPDET | RXB16 | RXB8 | RXB1 | RXBERR | RXC | RXE
Time: 649.9783ms   | Read  | False | RX_IRQ_STATUS_ADR  | 0    | 0      | 0     | 0    | 0    | 0      | 0   | 0
```

SEARCHING FOR PACKETS

Once displayed, you can search for the next packet that contains certain fields that match your criteria. Below is the Search Packet dialog box that is shown by using the View | Packet Search menu item.



Search Packet

Inc = False

Enter Field Name Value

Enter Field Name Value

Enter Field Name Value

OR

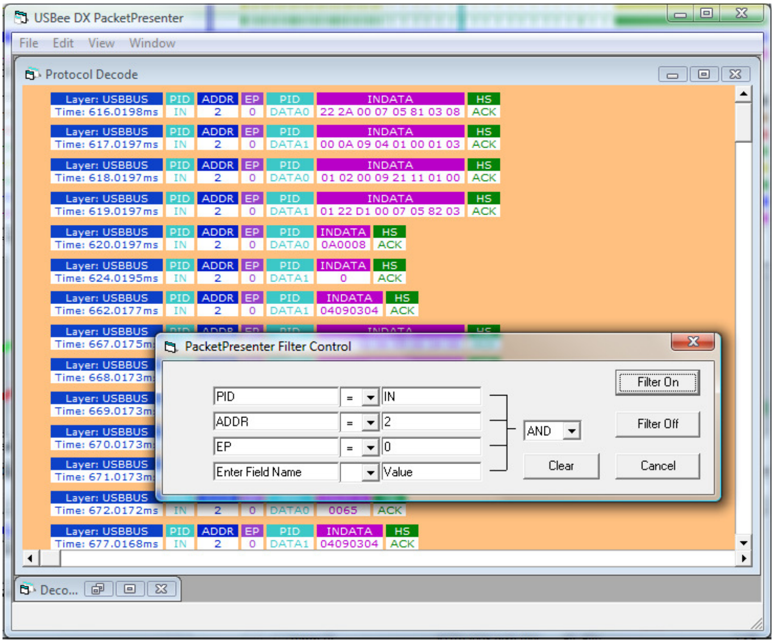
● Forward

○ Backward

In the leftmost textboxes, type the Field Label. Then select the comparator operator (equals, not equals, less than, greater than…) and finally the value that the field is to be compared against. Finally, if there is more than one field in the search list, choose whether to AND or OR the search terms. When you click Find, the next packet in the list (starting from the top of the window) will be placed at the top of the window. You can search forward or backward by selecting the appropriate radio button on the right.

FILTERING PACKETS

Once displayed, you can filter the output to only show packets that contains certain fields that match your criteria. Below is the Filter Packet dialog box that is shown by selecting the View | Packet Filter along with the resulting PacketPresenter output.



Layer: USBBUS	PID	ADDR	EP	PID	INDATA	HS
Time: 616.0198ms	IN	2	0	DATA0	22 2A 00 07 05 81 03 08	ACK
Time: 617.0197ms	IN	2	0	DATA1	00 0A 09 04 01 00 01 03	ACK
Time: 618.0197ms	IN	2	0	DATA0	01 02 00 09 21 11 01 00	ACK
Time: 619.0197ms	IN	2	0	DATA1	01 22 D1 00 07 05 82 03	ACK
Time: 620.0197ms	IN	2	0	DATA0	0A0008	ACK
Time: 624.0195ms	IN	2	0	DATA1	0	ACK
Time: 662.0177ms	IN	2	0	DATA1	04090304	ACK
Time: 667.0175m						
Time: 668.0173m						
Time: 669.0173m						
Time: 670.0173m						
Time: 671.0173m						
Time: 672.0172ms	IN	2	0	DATA0	0065	ACK
Time: 677.0168ms	IN	2	0	DATA1	04090304	ACK

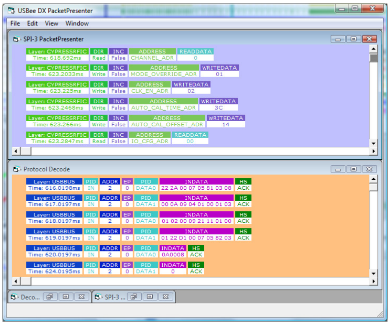
PID	=	IN
ADDR	=	2
EP	=	0
Enter Field Name		Value

AND

In the leftmost textboxes, type the Field Label. Then select the comparator operator (equals, not equals, less than, greater than…) and finally the value that the field is to be compared against. Finally, if there is more than one field in the search list, choose whether to AND or OR the search terms. When you click **Filter On**, only the packets matching the criteria are displayed. To turn off the filtering, click on the **Filter Off** button.

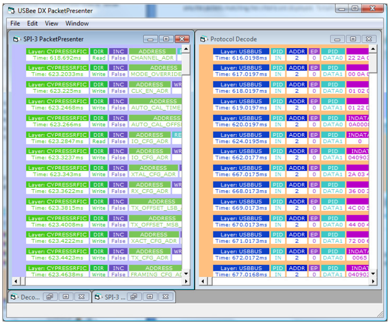
MULTIPLE DECODE DISPLAY

Using the Window | Tile menu you can choose to show the open windows Horizontally, Vertically or Cascaded as displayed below.



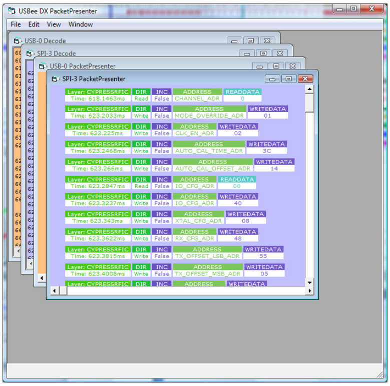
```
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | READDATA
Time: 618.692ms    | Read  | False | CHANNEL_ADR         | 0
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.2033ms   | Write | False | MODE_OVERRIDE_ADR   | 01
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.225ms    | Write | False | CLK_EN_ADR          | 02
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.2468ms   | Write | False | AUTO_CAL_TIME_ADR   | 3C
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.266ms    | Write | False | AUTO_CAL_OFFSET_ADR | 14
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | READDATA
Time: 623.2847ms   | Read  | False | IO_CFG_ADR          | 00
```

```
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA                  | HS
Time: 616.0198ms | IN  | 2    | 0  | DATA0 | 22 2A 00 07 05 81 03 08 | ACK
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA                  | HS
Time: 617.0197ms | IN  | 2    | 0  | DATA1 | 00 0A 09 04 01 00 01 03 | ACK
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA                  | HS
Time: 618.0197ms | IN  | 2    | 0  | DATA0 | 01 02 00 09 21 11 01 00 | ACK
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA                  | HS
Time: 619.0197ms | IN  | 2    | 0  | DATA1 | 01 22 D1 00 07 05 82 03 | ACK
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA | HS
Time: 620.0197ms | IN  | 2    | 0  | DATA0 | 0A0008 | ACK
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA | HS
Time: 624.0195ms | IN  | 2    | 0  | DATA1 | 0      | ACK
```



```
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 618.692ms    | Read  | False | CHANNEL_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.2033ms   | Write | False | MODE_OVERRIDE
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS     | WR
Time: 623.225ms    | Write | False | CLK_EN_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.2468ms   | Write | False | AUTO_CAL_TIME
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.266ms    | Write | False | AUTO_CAL_OFFSI
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS     | RE
Time: 623.2847ms   | Read  | False | IO_CFG_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS     | WR
Time: 623.3237ms   | Write | False | IO_CFG_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.343ms    | Write | False | XTAL_CFG_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS     | WR
Time: 623.3622ms   | Write | False | RX_CFG_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.3815ms   | Write | False | TX_OFFSET_LSB_
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.4008ms   | Write | False | TX_OFFSET_MSB_
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.4222ms   | Write | False | XACT_CFG_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS     | WR
Time: 623.4423ms   | Write | False | TX_CFG_ADR
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS
Time: 623.4638ms   | Write | False | FRAMING_CFG_AD
```

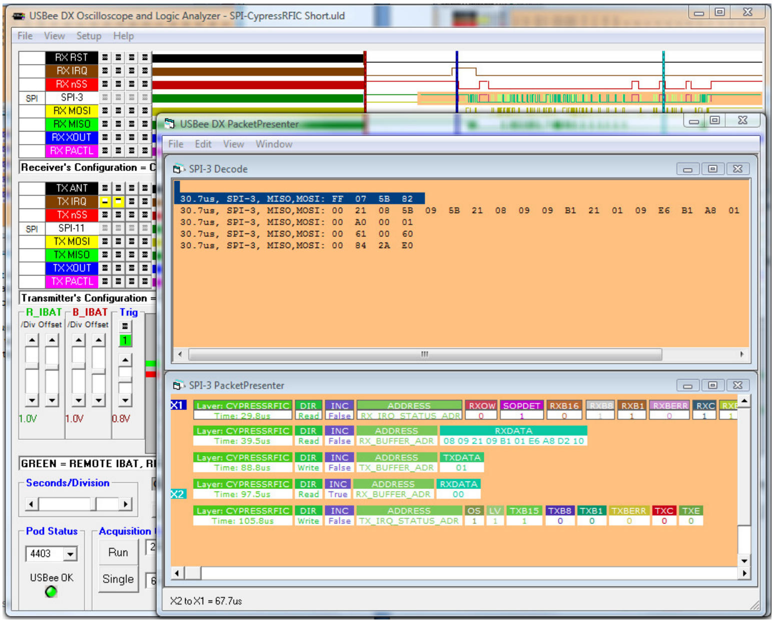
```
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 616.0198ms | IN  | 2    | 0  | DATA0 | 22 2A 0
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 617.0197ms | IN  | 2    | 0  | DATA1 | 00 0A 0
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 618.0197ms | IN  | 2    | 0  | DATA0 | 01 02 0
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 619.0197ms | IN  | 2    | 0  | DATA1 | 01 22 D
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA
Time: 620.0197ms | IN  | 2    | 0  | DATA0 | 0A000
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA
Time: 624.0195ms | IN  | 2    | 0  | DATA1 | 0
Layer: USBBUS    | PID | ADDR | EP | PID   | INDAT
Time: 662.0177ms | IN  | 2    | 0  | DATA1 | 040903
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 667.0175ms | IN  | 2    | 0  | DATA1 | 2A 03 4
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 668.0173ms | IN  | 2    | 0  | DATA0 | 36 00 3
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 669.0173ms | IN  | 2    | 0  | DATA1 | 4C 00 5
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 670.0173ms | IN  | 2    | 0  | DATA0 | 44 00 4
Layer: USBBUS    | PID | ADDR | EP | PID   |
Time: 671.0173ms | IN  | 2    | 0  | DATA1 | 72 00 6
Layer: USBBUS    | PID | ADDR | EP | PID   | INDATA
Time: 672.0172ms | IN  | 2    | 0  | DATA0 | 0065
Layer: USBBUS    | PID | ADDR | EP | PID   | INDAT
Time: 677.0168ms | IN  | 2    | 0  | DATA1 | 040903
```



```
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | READDATA
Time: 618.1463ms   | Read  | False | CHANNEL_ADR         | 0
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.2033ms   | Write | False | MODE_OVERRIDE_ADR   | 01
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.225ms    | Write | False | CLK_EN_ADR          | 02
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.2468ms   | Write | False | AUTO_CAL_TIME_ADR   | 3C
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.266ms    | Write | False | AUTO_CAL_OFFSET_ADR | 14
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | READDATA
Time: 623.2847ms   | Read  | False | IO_CFG_ADR          | 00
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.3237ms   | Write | False | IO_CFG_ADR          | 40
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.343ms    | Write | False | XTAL_CFG_ADR        | 08
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.3622ms   | Write | False | RX_CFG_ADR          | 48
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.3815ms   | Write | False | TX_OFFSET_LSB_ADR   | 55
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
Time: 623.4008ms   | Write | False | TX_OFFSET_MSB_ADR   | 05
Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | WRITEDATA
```

PACKETPRESENTER TO WAVEFORM ASSOCIATION

When you click on a packet in the PacketPresenter output window, the entire packet is highlighted and the associated raw decoded data is highlighted in the decode window. The original waveform screen is also shifted to center the start of the packet in the logic analyzer window.



```
30.7us, SPI-3, MISO,MOSI: FF  07  5B  82
30.7us, SPI-3, MISO,MOSI: 00  21  08  5B  09  5B  21  08  09  09  B1  21  01  09  E6  B1  A8  01
30.7us, SPI-3, MISO,MOSI: 00  A0  00  01
30.7us, SPI-3, MISO,MOSI: 00  61  00  60
30.7us, SPI-3, MISO,MOSI: 00  84  2A  E0
```

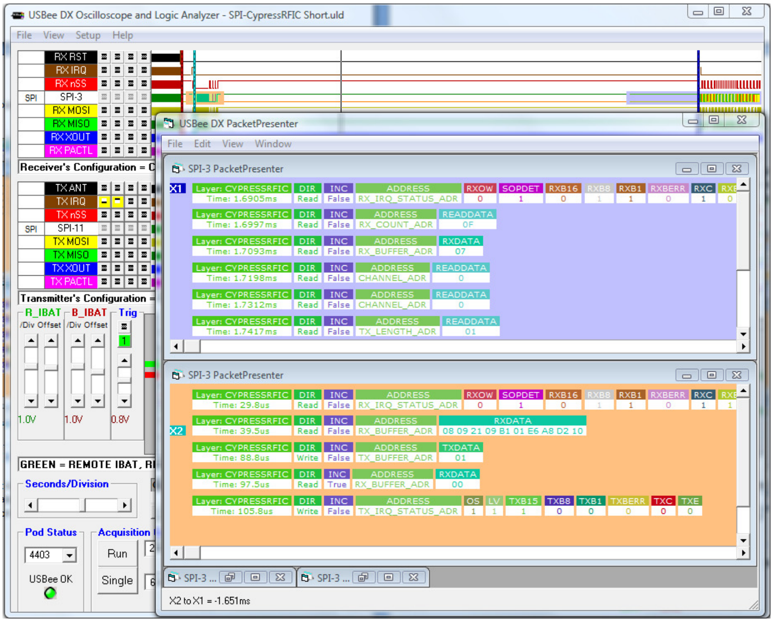
```
X1  Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS              | RXOW | SOPDET | RXB16 | RXB8 | RXB1 | RXBERR | RXC | RXE
    Time: 29.8us       | Read | False | RX_IRQ_STATUS_ADR    | 0    | 1      | 0     | 1    | 1    | 0      | 1   | 1

    Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS              | RXDATA
    Time: 39.5us       | Read | False | RX_BUFFER_ADR        | 08 09 21 09 B1 01 E6 A8 D2 10

    Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | TXDATA
    Time: 88.8us       | Write | False | TX_BUFFER_ADR       | 01

    Layer: CYPRESSRFIC | DIR  | INC  | ADDRESS               | RXDATA
X2  Time: 97.5us       | Read | True | RX_BUFFER_ADR         | 00

    Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS             | OS | LV | TXB15 | TXB8 | TXB1 | TXBERR | TXC | TXE
    Time: 105.8us      | Write | False | TX_IRQ_STATUS_ADR   | 1  | 1  | 1     | 0    | 0    | 0      | 0   | 0

X2 to X1 = 67.7us
```

This feature allows you to correlate what is shown in the PacketPresenter window to the actual waveform on the logic analyzer that created that packet.

CURSORS ON THE PACKETPRESENTER OUTPUT

You can place the cursors using the PacketPresenter window by using the left and right mouse buttons. Place the mouse over the packet you want to place the cursor on and click the left or right button. The cursors are placed at the beginning of the packets. The resulting difference between cursors s shown at the bottom of the screen.

If more than one bus is being shown, you can measure the time between packets on different busses using the cursors as shown in the following screen. Set the first cursor by left clicking in the first window and place the second by right clicking in the second window.



```
X1  Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS            | RXOW | SOPDET | RXB16 | RXB8 | RXB1 | RXBERR | RXC | RXE
      Time: 1.6905ms   | Read | False | RX_IRQ_STATUS_ADR  | 0    | 1      | 0     | 1    | 1    | 0      | 1   | 0
    Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS            | READDATA
      Time: 1.6997ms   | Read | False | RX_COUNT_ADR       | 0F
    Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS            | RXDATA
      Time: 1.7093ms   | Read | False | RX_BUFFER_ADR      | 07
    Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS            | READDATA
      Time: 1.7198ms   | Read | False | CHANNEL_ADR        | 0
    Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS            | READDATA
      Time: 1.7312ms   | Read | False | CHANNEL_ADR        | 0
    Layer: CYPRESSRFIC | DIR  | INC   | ADDRESS            | READDATA
      Time: 1.7417ms   | Read | False | TX_LENGTH_ADR      | 01
```

```
    Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | RXOW | SOPDET | RXB16 | RXB8 | RXB1 | RXBERR | RXC | RXE
      Time: 29.8us     | Read  | False | RX_IRQ_STATUS_ADR  | 0    | 1      | 0     | 1    | 1    | 0      | 1   | 1
    Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | RXDATA
X2    Time: 39.5us     | Read  | False | RX_BUFFER_ADR      | 08 09 21 09 B1 01 E6 A8 D2 10
    Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | TXDATA
      Time: 88.8us     | Write | False | TX_BUFFER_ADR      | 01
    Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | RXDATA
      Time: 97.5us     | Read  | True  | RX_BUFFER_ADR      | 00
    Layer: CYPRESSRFIC | DIR   | INC   | ADDRESS            | OS | LV | TXB15 | TXB8 | TXB1 | TXBERR | TXC | TXE
      Time: 105.8us    | Write | False | TX_IRQ_STATUS_ADR  | 1  | 1  | 1     | 0    | 0    | 0      | 0   | 0
```

X2 to X1 = -1.651ms

PACKETPRESENTER DEFINITION FILE FORMAT

Each PacketPresenter Definition file defines how the incoming data stream is represented in the PacketPresenter screen of the USBee DX MSO application. These PacketPresenter Definition files are in text format and are easily created using either a simple text editor.

Each bus defined in the USBee DX MSO application can have a different PacketPresenter Definition File.

The intent of the PacketPresenter is to produce a series of 2 dimensional arrays of labels and values to be displayed as below by the user interface.

Command	Length	Address	Data
45	2	84DF	34

Command	Value
Read RSSI	14.34

Command	Setting
23	Power Amp On

It is the PacketPresenter Definition File that defines how the data is to be parsed and displayed.

COMMENTS IN THE PACKETPRESENTER DEFINITION FILE

Comments are started with a semicolon (;) and go until the end of the line.

CONSTANTS IN THE PACKETPRESENTER DEFINITION FILE

Constants are fixed numbers anywhere in the file. These constants can be expressed as decimal, hex, or binary using suffixes after the value. Decimal has no suffix. Hex uses the suffix “h”. Binary uses the suffix “b”.

So,

```
16 = 10h = 10000b
244 = F4h = 11110100b
```

Gain and offset values used in the Fields section are always in decimal and can contain decimal places.

PACKETPRESENTER DEFINITION FILE SECTIONS

Each PacketPresenter Definition File has the following syntax that separates the file into sections that correspond to the Channel definition and each of the Protocol Processors.

```
[Protocol]
. . .
[Protocol]
. . .
[Protocol]
. . .
```

PROTOCOL SECTION

Each Protocol Section defines what the incoming data stream looks like, how to break the data stream into packets, and how to parse out the fields in each of the packets. Multiple Protocol Sections can be defined for passing data from one Protocol Section to another.

Each Protocol Section has the following syntax that specifies the packetizing and parsing into fields.

```
[Protocol]
name = ProtocolName
[Packet]
   packet processing settings
[Fields]
   packet field processing settings
   packet field processing settings
   packet field processing settings
   . . .
```

The *ProtocolName* is a label that uniquely identifies this protocol processor. This name is used in the Field definitions to define which Protocol to route a field of data (for use by multilayer protocols).

The highest level Protocol is the first protocol in the file. This is the Protocol Processor that is sent the incoming data stream from the bus as defined in the Channel Settings Dialog Box for that waveform.

BYTE-WISE BUSSES VS. BIT-WISE BUSSES

Some busses are by nature byte oriented, while others are bit oriented. The following table shows the type of bus.

Bytewise Busses

- Async
- I2C
- Parallel
- SPI
- PS2

Bitwise Busses

- Serial
- I2S
- OneWire
- CAN
- USB

BUS EVENTS

Each bus type also can have certain bus events that may be significant in the decoding of a protocol. One such event is an I2C Start Bit. While the Start bit is not an actual bit in the data stream, it does signify to the I2C slave that a certain transaction is taking place. These bus events are inserted into the data stream and can be used (or ignored) by the protocol processors. The list of Bus Events supported is in the following table.

Bus Type	Event
Async	1 – Parity Error
I2C	1 - Start Bit 2 - Stop Bit 4 - ACK 8 – NACK
SPI	1 - SS Active 2 - SS Inactive Note: You MUST have SS On in the channels settings for these events to occur
USB	1 – SETUP/IN/OUT Received 2 –ACK/NACK/Stall Received 4 – No Handshake received
CAN	1 – Start of CAN packet 2 – End Of CAN packet
1-Wire	1 - Reset Found 2 - Presence Found
Parallel	
Serial	
PS/2	1 – Device to Host byte follows 2 – Host to device byte follows
I2S	1 - WordSelect Active 2 - WordSelect InActive
SMBus	1 - Start Bit 2 - Stop Bit

Table 1. Bus Event Types

A Bus Event of 127 (7Fh) is a special event that occurs at the end of a packet of data that is sent from one protocol to another. This can be used to end the packet sent to the new layer using the [END] section and the type = event in the new protocol level.

DATA CHANNELS AND MULTIPLE DATA SIGNALS

Some buses can also have more than one data signal used in the protocol. One example of this is the SPI bus, where for each byte sent on the MOSI line there is one byte received on the MISO line. In the protocol definition you can specify which of the signals to expect the next field of data to be sent on. In the SPI example, you may get a Command and Length field on one signal, followed by the read data back on the other signal. The decoder would take that into account and show the command, Length and Data as a single transaction.

Multiple signals are differentiated in the PacketPresenter using the X and Y channel specifiers. These channels are specified by selecting the signals to use for that bus in the Channel Settings dialog box. The following table shows which signals are the X and Y signals.

Bus Type	**Channel Setting Dialog Box setup for Channel X**	**Channel Setting Dialog Box setup for Channel Y**	**Notes**
ASYNC	Least Significant Async Channel selected	Next Least Significant Async Channel selected	If more than 2 Async channels are selected to be decoded, the additional channels are not used by the PacketPresenter.
SPI	Signal chosen for MISO	Signal chosen for MOSI	Data Bytes alternate channels since there is one byte X for every one byte Y
1 Wire	Data Signal	Not used	
I2C	Data on SDA/SCL bus	Not Used	
Parallel	All Data Signals sampled together	Not Used	Each sample of all channels is the data word sent to channel X
Serial	Serial Data	Not Used	
CAN	Rx Data	Not Used	
PS/2	Data from Device to Host	Data from Host To Device	
USB	Data on D+/D- bus	Not Used	The data stream contains the Sync, PIDs, data fields and CRCs. The EOP is not included. See the USB Example file for example Field Lines.

Table 2. Channel X and Channel Y Definitions Per Bus Type

PACKET SECTION

The Packet section defines how a packet is bounded and what, if any, preprocessing needs to be done on the packet before the fields can be processed.

```
[Packet]
[Start]
                    . . .   ; How does a packet start?
[End]
                    . . .   ; How does a packet end?
[Decode]
                    . . .   ; What decoding needs to be
                            ; done to get real data?
```

START AND END SECTIONS

The Start and End sections define how a packet is bounded. The available packet bounding Types are defined below:

For [START]

- Next: The next byte or bit is assumed the start of a packet
- Signal: An external signal indicates the start of a packet
- Value: A specific value in the data indicates the start of a packet
- Event: A bus specific bus Event or Events indicates the start of a packet

For [END]

- Next: The next byte or bit is assumed the end of a packet
- Signal: An external signal indicates the end of a packet
- Value: A specific value in the data indicates the end of a packet
- Length: A specific or calculated length determines the end of a packet
- Event: A bus specific bus Event or Events indicates the end of a packet
- Timeout: A packet ends after a set timeout without data or events

TYPE = NEXT

The start or end of a packet is the next byte or bit to arrive.

```
[Packet]
[Start] or [End]
type = Next        ; Start/End of a packet is the
                    ; next byte/bit to arrive
```

TYPE = SIGNAL

The start or end of a packet can be indicated by a separate signal (such as a chip select or a frame signal) using the signal setting.

```
[Packet]
[Start] or [End]
type = signal                   ; Start/End of a packet is based
                                ; on a signal
signal = signalvalue            ; Signal number 0 - 15
level = 1                       ; level the signal needs to be
```

TYPE = VALUE

The start or end of a packet can be indicated by a certain data value contained in the data using the value setting. Multiple values can be used, where any one match starts or ends a packet. All bits in the Value are included in the resulting packet at the start of the packet. You must also specify the number of bits that the value covers (defaults to 8 bits if not specified) using the bits keyword. You can specify a mask value to apply to the start data and values. When the mask value has a bit that is a 1, that bit in the value and data are compared.

```
[Packet]
[Start] or [End]
type = value      ; Start/End of a packet is based on a data value
mask = bitmask    ; Bitmask to apply to the data stream
value = value1    ; value that the data needs to be to start/End
value = value2    ; value that the data needs to be to start/End
value = value3    ; value that the data needs to be to start/End
bits = 8          ; how many bits in the start/End word
```

You can use the EXCLUDE keyword in the [END} section to leave the end data on the data stream for the next packet. This is useful for when there is no indication of the end of a packet except for the arrival of the next packet.

TYPE = LENGTH

Only valid in the [END] section, the end of a packet can be indicated by a certain length of data. You use the BitLength or the ByteLength keywords to specify how long the packet is. The length can either be a fixed length expressed as a constant, or variable length based on the contents of a packet in the data stream.

```
type = length          ; End of a packet is based
                       ;    on a length
Bytelength = length    ; How many bytes per
                       ;    packet
or
Bitlength = length     ; How many bits per packet
```

To use the contents of one of the fields as the packet length, you use the name of the field defined in the Fields section. You can also do simple arithmetic on the field value to compute the final packet size.

```
type = length      ; End of a packet is based
                   ; on a length
Bytelength = fieldname * 2 + 2
             ; field holding packet size
             ; * (or /) a constant (optional)
             ; + (or -) a constant (optional)
```

If present, the * or / must come before the + or – offset and is executed first.

For example, if `fieldname` Field has the contents of 16, then the following is true:

fieldname * 2 + 2 = (16*2)+2 = 34

fieldname + 2 = 16+2 = 18

fieldname / 2 - 2 = (16/2)-2 = 6

fieldname / 2 = 16/2= 8

fieldname + 2 * 2 = invalid (* must come before offset)

fieldname - 2 / 2 = invalid (/ must come before offset)

The length of the packet includes ALL of the data from each of the data channels for that bus. If the bus contains only one data channel (such as I2C), the length counts all data on that one bus. If the bus has two data channels, the length refers to all data on both channels combined.

TYPE = EVENT

The start or end of a packet can be indicated by the reception of any of the bus specific Events. For example in I2C you get a Bus Event for each Start Bit and a Bus Event for each Stop Bit. In USB you get a Bus Event for each Sync word and a Bus Event for each EOP. Available bus types are defined in Table 1. Bus Event Types.

The event value is a bitmask that includes all events that you want to use. If any of the events occur, a packet will be started or ended.

```
type = Event     ; Start/End of a packet is
                 ;    signaled by event
event = 1        ; Use Event 1. Available events
                 ;    depend on bus type
or
event = 3        ; Use either Event 1 or Event 2
```

TYPE = TIMEOUT

The end of a packet is determined by a timeout since the last valid data or event on the bus. The timeout is defined in units of microseconds.

```
[Packet]
[Start]
type = timeout   ; End is after timeout
timeout = 45     ; microseconds since last data/event received
```

CHANNELX, CHANNELY OR CHANNELXORY

CHANNELX, CHANNELY or CHANNELXorY specifies what channel is used when an event or data is defined for starting or ending a packet. Channel X and Channel Y are different based on what the physical bus is and can be found in Table 2. Channel X and Channel Y Definitions Per Bus Type. If it does not matter which channel the data or event occurs on (it could be either), use the CHANNELXorY keyword.

```
[Packet]
[Start]
type = value     ; Start of a packet is based on
                 ;     a data value
value = 41h      ; value of data that starts the
                 ;     packet
bits = 8
channelX         ; data/event must be received
                 ;     on channel X
  or
channelY         ; data/event must be received
                 ;     on channel Y
  or
channelXorY      ; data/event must be received
                 ;     on either channel X or Y
```

DECODE SECTION

Each packet can have encoding on the data that needs to be removed in order to see the real data. This section defines what decoding should be done to the packet. The entire packet from start to end is sent through the decoders. If only select parts of the packet needs to be decoded, you must create your own Add-In decoder using the ADDIN keyword.

Available decoding types are:

Keyword	Definition
NRZI	A bit change on the input means a 1 bit on the output, no change a 0
MANCHESTER	Remove Manchester encoding from data
INVERT	Invert all bits
ZBI5	Zero-Bit Insertion removal (removes the 0 added after 5 1s)
ZBI6	Zero-Bit Insertion removal (removes the 0 added after 6 1s)
ADDIN	Call your own packet decoder using the PacketPresenter API routine APIDecode()
substring	Substitute bytes in the stream (no spaces allowed)

Multiple decoders can be used and are processed in the order listed.

SUBSTITUTIONS

Substitutions allow a sequence of bytes (up to 3) to be replaced with a different set (same size or less) of bytes. They can only be used on bytestreams, not bitstreams. Substrings define the bytes input and the bytes output. The Substrings must not contain any spaces. Examples of this are below:

```
[1]=[2]             ; Replaces all 1s with 2s
[1][2]=[3]          ; Replaces all 1 immediately
                    ;     followed by 2 with 3
[1][2]=[3][4]       ; Replaces all 1 immediately
                    ;    followed by 2 with 3
                    ;    immediately followed by 4
[1][2][3]=[4]       ; Replaces all 1, 2, 3 with 4
[1]=[2][3][4]       ;    INVALID, the number of
                    ;    output bytes must be less
                    ;    than or equal to the input
```

As an example, the HDLC protocol uses the byte value 7Eh as the start and end flag of the packets and replaces all 7Eh in the data with the bytes 7Dh followed by 5Eh. It also replaces all 7Dh in the data with the bytes 7Dh followed by 5Dh. To remove this coding you would use the lines:

```
[7Dh][5Eh]=[7Eh]
[7Dh][5Dh]=[7Dh]
```

FIELDS SECTION

Once the packet is delineated and decoded by the previous sections, it is ready to be displayed by the PacketPresenter. Since each packet is made up of fields, the Fields section defines how the packet is broken up into its fields and what to do with the field data.

FIELD LINES PROCESSING

During processing, the **Fields Section** is processed one **Field Line** at a time in the order that they are listed in the FIELDS section. Each Field Line is parsed against the incoming data packets.

Once a single Field Line is successfully processed and output, the PacketPresenter starts over at the top of the Filed Lines list for the next packet. This ensures that there is only one output packet for each input packet for a given protocol.

There are 2 types of Field Lines. A Field Line can be conditional or unconditional. Unconditional Field Lines are processed for any packet. Conditional Field Lines are only processed if certain fields match a specific value.

Any Unconditional Field Line (no conditionals) generates an output line on the PacketPresenter screen. Any Conditional Field Line that evaluates to True generates an output line on the PacketPresenter screen. Any Conditional Field Line that evaluates to False is skipped and produces no output line on the PacketPresenter screen.

The Field Lines should be listed with the conditional field lines first followed by an unconditional field line to catch all packets that are not explicitly defined in the conditional field lines.

UNCONDITIONAL FIELD LINES

Unconditional Field lines are parsed and decoded data is output for every packet that is input. The Fields specify how to interpret the data and how to output the data.

CONDITIONAL FIELD LINES

Conditional Field Lines provide a means for defining packets whose contents vary based upon the presence of a value in another field. An example of this is a packet that contains a Command Byte that determines the format of the rest of the packet. A Conditional Field Line contains at least one field in the packet that includes the =*Value* token in the input modifiers section.

If the data contained in the conditional fields of a packet matches the =*Value* specified for the field, the packet is parsed and the data is output. If the condition field =*Value* does not match the incoming data, then the processor moves on to the next Field Line until it reaches the end of the Fields section.

FIELD LINE FORMAT

Each Field Line in the Fields Section has the keyword FIELDS followed by a series of individual Fields. Individual fields in a packet are separated by commas. A Field line in the Fields Section defines an entire packet from start to end and has the form:

```
Fields  Field1,Field2,. . . ,FieldN
```

You can also insert a string to be printed out at that location in the packet by using the string ($) operator before the string to be printed. Below is an example of a field line with one string added between the fields.

```
Fields Field1,$String,. . . ,FieldN
```

Each field will be output with a Label and a Value. For String fields, the Label is blank and the Value is the String.

FIELD FORMAT

Each field in the Field Line is defined using the following syntax and contains no spaces:

```
FieldName.InputModifiers (= value).OutputModifiers
```

FieldName is the name of the field. No spaces, commas, semicolons, brackets, dollar signs, periods, or quotes are allowed in the fieldname.

Input and output modifiers change the way incoming data and output data are formatted.

InputModifiers are a string of characters that represent how many bits are in the field and how the input data is to be handled. First is the number of bits in the field, or N if the field is a variable length. Next is any of the following:

- M: native bit order from that which came off of the bus (default)

- L: inverted bit order from that which came off of the bus
- X or Y: which channel the data is on (for multiline busses)
- =*Value*: Indicates that this field MUST be this value for the entire line to be processed (**Conditional**)

Each modifier is a single character and multiple format modifiers can be combined.

OutputModifiers are a string of characters that represent how to output the contents of this data.

Output Modifiers are as follows:

- I Ignore - no output (entire field is ignored for output)
- D Decimal output
- H Hexadecimal output
- B Binary output
- A Ascii output
- TF True (nonzero) or False (zero)
- L Look up the text string to print out in a matching Lookup line
- **Value* or /*Value*: a value to multiply/Divide the output value by
- +*Value* or -*Value*: a value to offset the output value by
- $string: string to print after the data (or in place of the data if the i flag is used). String must be the last item in a field. No commas, quotes, semicolons or parenthesis allowed in the string.

BUS EVENTS IN THE MIDDLE OF A PACKET

Sometimes a specific bus event plays a role in the packet format. To specify that a specific bus event needs to occur at a specific time in the field sequence, place the single Bus Event value inside brackets in the Field Line. Multiple events in a single value are not allowed, however consecutive events are allowed. To indicate the absence of a specific bus event in the protocol, use the ! (Not) operator.

For example, if the bus is I2C, use the following to require that a Start Bit is present between field1 and field2:

```
Fields  Field1,[1],Field2
```

If there is a start bit between the 2 fields, then that Field Line will be processed.

And use the following to require that a Start Bit is NOT present between field1 and field2:

```
Fields  Field1,[!1],Field2
```

If there is a start bit between the 2 fields, then that Field Line will not be processed.

The Bus Events are defined in Table 1. Bus Event Types.

LOOKUP TABLES

Often fields contain values that mean something unrelated to the actual number of the data. Lookup Tables provide a way to output a string of text instead of a data value for a field. For each field wanting to use a lookup table, use the “L” output modifier in the field format and then define the table in the FIELDS section using the LOOKUP keyword.

The format of the Lookup table is as follows:

```
LOOKUP Fieldname
[value1]=$string1
[value2]=$string2
. . .
```

Fieldname is the name of the field associated with this lookup table. *valuen* refers to the actual data value of the field. *stringn* is the text string that is output instead of the *valuen*.

If a lookup entry is not present for the data value (not found in the Lookup Table or the Lookup Table does not exist), then the data value is output.

For example, the following table will assign the text strings for various values of the data for the CommandByte field. When the field CommandByte,8,L is processed, the strings are output instead of the value

```
Lookup CommandByte
                [0]=$Read
                [1]=$Write
                [2]=$Seek
                [3]=$Loc
                [4]=$Size
```

The Lookup Tables are only associated to the specific Protocol they are contained in. Therefore you can have a CommandByte lookup table in ProtocolA that is different from a CommandByte lookup table in ProtocolB. Within a single Protocol, you need to make sure that the Fieldnames are unique for all Lookup Tables so that the PacketPresenter can determine which table to use.

EXAMPLES OF FIELD LINES AND FIELDS

JUST PLAIN DATA

Fields contain data that may or may not be of interest to the user. Many times the data is information that just needs to be output to the viewer. Being binary data, each field may need to be translated numerically to mean something. To output a field of data, you can specify the radix (if it should be shown in Hex, Decimal, binary) as well as a gain and offset to scale the data. Finally you can add a string to the field to complete the information. All scaling is performed first using floating point and then the output formatting is applied.

Below is an example of a field to just output the data.

```
Fields Volts.16m.d*1.5-37.256$mV
```

This Field Line contains one field named “Volts”, which is 16 bits long in msbit first order. The output is to be displayed in decimal format, multiplied by 1.5, offset by - 37.256 and finally appended with “mV” before output to the PacketPresenter screen.

For an input packet as follows:

```
0000001100001100. . .
```

The output would be:

Volts
1132.744mV

which is the input 16 bits in msbfirst order (0x30C) times the gain of 1.5 plus the offset of -37.256 output in decimal format plus the “mV” string.

CONDITIONAL PACKET FORMAT

Using the Conditional input modifier, many different field arrangements can be defined for the same packet. Common uses are for parameter fields that exist for different types of commands. If packets contain commands that determine what the remaining fields are, this syntax defines what those remaining fields are.

Below is an example of various packet formats based on a single command field.

```
Fields Command.4m=0.h,Address.8m.h
Fields Command.4m=2.h,Address.8m.h,Data.8m.h
Fields Command.4m=4.h,Param1.8m.h,Param2.8m.h,Param3.8m.h
```

For an input packet as follows:

```
0010 00011101 00001000. . .
```

Followed by a packet:

```
0100 00011101 00001000 11111110. . .
```

The output would be:

Command	Address	Data
2	1D	08

Command	Param1	Param2	Param3
4	1D	08	FE

which are the fields associated with the Command=2 and Command=4 Field Lines.

STRING LOOKUP

Fields that can be better expressed as text strings can be outputted as such using a Lookup table.

Below is an example of a field that uses a lookup table.

```
[Fields]
Fields StartByte.8.H, CommandByte.8.L, EndByte.8.H
Lookup CommandByte
                                 [0]=$Read
                                 [1]=$Write
                                 [2]=$Seek
                                 [3]=$Loc
                                 [4]=$Size
```

For an input packet as follows:

```
00100001 00000001 00001000. . .
```

The output would be:

StartByte	Command	EndByte
21	Write	08

which is the text associated with the Command Field 4 bits in msbfirst order (0010b = 2).

CONDITIONAL ROUTE OF DATA TO ANOTHER PROTOCOL

Many embedded protocols support multiple layers of protocol, where each protocol layer handles a different set of services or functions. In these multilayer protocols, a field of data from one protocol layer may be the input data to another layer of protocol. Routing this field of data to a new Protocol is as easy as naming the Field the same name as the Protocol. If the Field name matches any protocol, the entire data for that field is passed to that Protocol for processing.

Below is an example that shows a field being sent to a new layer (Layer2) of protocol when the command field is a 1.

```
[Protocol]
name = Layer1
[Packet]
[Decode]
[Fields]
Fields Command.4=0.h,Address.8.h
Fields Command.4=1.h,Layer2.48.h

[Protocol]
name = Layer2
[Packet]
[Decode]
[Fields]
Fields L2Command.4=0.h,RSSI.8.d
Fields L2Command.4=1.h,QoS.16.d
Fields L2Command.4=2.h,Layer3.44.h
```

PACKETPRESENTER ADD-IN API

The USBee DX PacketPresenter automatically processes many types of data streams. However, it cannot decode custom coded data streams. Using the PacketPresenter Add-In API, the data stream can be decoded to the basic data values for any custom coding.

The USBee DX software package includes a sample DLL project in Microsoft VC6 format (in the installation directory of the USBee DX software) called AddIn that allows you to customize a decoder for your data streams.

The DLL library called usbeeai.dll (USBee Add-In) has the following interface routine that is called by the PacketPresenter if the ADDIN keyword is used in the DECODE section of the PacketPresenter Definition File.

```
CWAV_EXPORT unsigned int CWAV_API  APIDecode(
                                 char *Protocol,
                                 char bitIn,
                                 char &bitOut,
                                 char reset );
```

This routine is called for each bit of data in the data stream. Protocol is the string name of the Protocol being processed and allows you to create an add-in that handles many different kinds of decoding. The parameter “reset” is set to a 1 for the first bit of a packet and 0 for all bits following. The next bit from the stream is passed in using the parameter “bitIn” (1 or 0).

After your code decodes the stream, you can either send back no data (return value of 0), or send a new bits back using the “bitOut” pointer (one bit per char) and a return value of the number of bits returned.

The default Add-In routine simply is a pass through so that the output data stream equals the input data stream. Start with this library source code to add your custom decoding.

SAMPLE PACKETPRESENTER ADD-IN DECODERS

Custom decoders can perform complicated decryption and byte or bit manipulation. Ignoring the actual algorithm that is executed, these decoders may reduce, enlarge or keep constant the number of bits in the data stream. The following examples are intended to show how these streams can be shortened, lengthened or modified. Useful decoders will need to have the appropriate algorithms to compute the true values of the output bits.

LOOPBACK DECODER

This Add-In simply loops back the data (out = in).

```
CWAV_EXPORT unsigned int CWAV_API APIDecode(char *Protocol, char bitIn, char *bitsOut, char
reset )
{

          // This will be the Add-In routine that is called by the PacketPresenter
          // when the ADDIN keyword is used in the DECODE section of the
          // PacketPresenter Definition File.

          // This routine is called for each bit of data in a data packet.
          // The parameter "reset" is set to a 1 for the first bit of a packet and
          // 0 for all bits following.  The next bit from the stream is passed in
          // using the parameter "bitIn" (1 or 0). After your code decodes the stream,
          // you can either send back no data (return value of 0), or send new bits back
          // using the "bitOut" pointer (one bit per char) and a return value of the number
          // of bits returned. The default Add-In routine is simply is a pass through so
          // that the output data stream equals the input data stream.
          // Start with this library source code to add your custom decoding.

          *bitsOut = bitIn;

          return( 1 );                  // Indicates that there is 1 return data bit
}
```

INVERTING DECODER

This Add-In inverts the packet data (out = Not(in)).

```
CWAV_EXPORT unsigned int CWAV_API APIDecode(char *Protocol, char bitIn, char *bitsOut, char
reset )
{
          if (bitIn)
                    *bitsOut = 0;
          else
                    *bitsOut = 1;

          return( 1 );                  // Indicates that there is 1 return data bit
}
```

EXPANDING DECODER

This Add-In shows how to convert a stream to a larger stream (expanding the bits). In this case each bit becomes two output bits.

```
CWAV_EXPORT unsigned int CWAV_API  APIDecode(char *Protocol, char bitIn, char *bitsOut,
char reset )
{
          *bitsOut++ = bitIn;
          *bitsOut++ = bitIn;

          return( 2 );                  // Indicates that there is 2 return data bits
}
```

COMPRESSING DECODER

This Add-In shows how to remove bits from a stream (compressing the bits). In this case each bit pair becomes a single bit, basically throwing away the first bit.

```
CWAV_EXPORT unsigned int CWAV_API  APIDecode(char *Protocol, char bitIn, char *bitsOut,
char reset )
{
          static everyother = 0;

          if (reset)                              // Reset the state of the decoder if
reset=TRUE
                    everyother = 0;

          if (everyother)
          {
                    *bitsOut = bitIn;
                    return( 1 );                  // Indicates that there is 1 return data
bit
                    everyother = 0;
          }
          else
                    everyother = 1;

          return( 0 );                  // Indicates that there are no return data bits
}
```

MULTIPLE DECODERS

This Add-In shows how to use the Protocol string to selectively decode different types of packets.

```
CWAV_EXPORT unsigned int CWAV_API  APIDecode(char *Protocol, char bitIn, char *bitsOut,
char reset )
{
     static everyother = 0;

     if (!strcmp( Protocol, "COMPRESS")
     {
          if (reset)                // Reset the state of the decoder if reset=TRUE
               everyother = 0;
          if (everyother)
          {
               *bitsOut = bitIn;
               return( 1 );         // Indicates that there is 1 return data bit
               everyother = 0;
          }
          else
               everyother = 1;
          return( 0 );         // Indicates that there are no return data bits
     }
     else if (!strcmp( Protocol, "EXPAND")
     {
          *bitsOut++ = bitIn;
          *bitsOut++ = bitIn;
          return( 2 );         // Indicates that there is 2 return data bits
     }

     // No matching decoder label found so just loopback the data
     *bitsOut = bitIn;

     return(1);
}
```

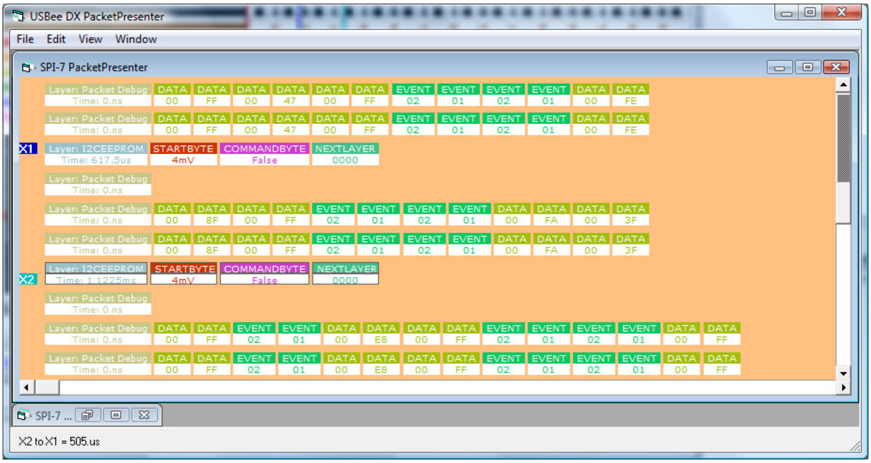
PACKETPRESENTER DEFINITION FILE DEBUGGING

Creating your PacketPresenter Definition File can be made simpler using the Debug mode. To turn on Debug mode, use the DebugOn keyword in a [DEBUG] section of the Definition File.

```
[Protocol]
                    name = I2CEEPROM
[DEBUG]
                    DebugOn          ; Turns On Debug Mode.
                                     ; Comment it out to turn it off.
[Packet]
```

When debug mode is on, each packet is output twice in its raw form, showing the data values as well as the events from the bus. The first debug line is the initial bus data. The second line is the bus data after any decoding is completed. Following the debug lines are the PacketPresenter output packets from this same data.

Below is a screen shot that shows the PacketPresenter that has Debug turned on.



```
   Layer: Packet Debug  DATA DATA DATA DATA DATA DATA EVENT EVENT EVENT EVENT DATA DATA
      Time: 0.ns         00   FF   00   47   00   FF   02    01    02    01    00   FE
   Layer: Packet Debug  DATA DATA DATA DATA DATA DATA EVENT EVENT EVENT EVENT DATA DATA
      Time: 0.ns         00   FF   00   47   00   FF   02    01    02    01    00   FE
X1 Layer: I2CEEPROM  STARTBYTE COMMANDBYTE NEXTLAYER
    Time: 617.5us       4mV       False       0000
   Layer: Packet Debug
      Time: 0.ns
   Layer: Packet Debug  DATA DATA DATA DATA EVENT EVENT EVENT EVENT DATA DATA DATA DATA
      Time: 0.ns         00   8F   00   FF   02    01    02    01    00   FA   00   3F
   Layer: Packet Debug  DATA DATA DATA DATA EVENT EVENT EVENT EVENT DATA DATA DATA DATA
      Time: 0.ns         00   8F   00   FF   02    01    02    01    00   FA   00   3F
X2 Layer: I2CEEPROM  STARTBYTE COMMANDBYTE NEXTLAYER
    Time: 1.1225ms      4mV       False       0000
   Layer: Packet Debug
      Time: 0.ns
   Layer: Packet Debug  DATA DATA EVENT EVENT DATA DATA DATA DATA EVENT EVENT EVENT EVENT DATA DATA
      Time: 0.ns         00   FF   02    01    00   E8   00   FF   02    01    02    01    00   FF
   Layer: Packet Debug  DATA DATA EVENT EVENT DATA DATA DATA DATA EVENT EVENT EVENT EVENT DATA DATA
      Time: 0.ns         00   FF   02    01    00   E8   00   FF   02    01    02    01    00   FF
```

X2 to X1 = 505.us

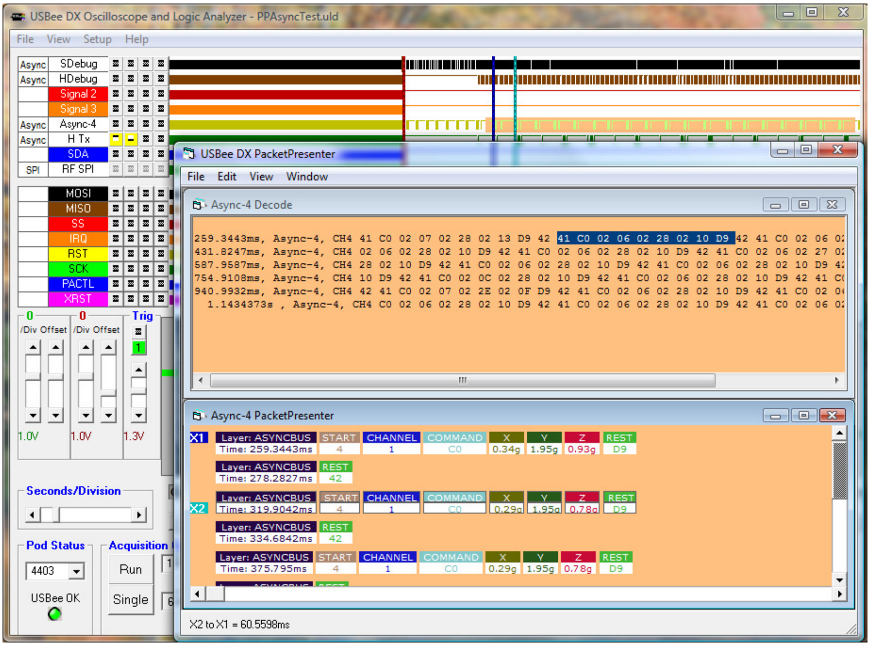
PACKETPRESENTER SPECIFICATIONS

The PacketPresenter system has the following limits regarding file size, packets, fields, lookup tables etc.

- 100K bytes per PacketPresenter Definition File
- 64K Data Records per Packet (min 64K bits, max 64K bytes)
- 7 Protocols
- 1024 Field Lines per Protocol
- 128 Fields per Field Line
- 64 Lookup Tables per Protocol
- 256 Lookup entries per Lookup Table
- 256 Decoder Substitutions per Protocol
- 3 Bytes per Substitution input or output
- 4 PacketPresenter Windows
- 2.1B bytes per PacketPresenter Output File

ASYNC PROTOCOL EXAMPLE

```
; Async Protocol Definition File
; This file defines the transfers to/from a custom device
; over an ASYNC bus
;
[Protocol]
    name = ASYNCBus
    bytewise
[DEBUG]
    ;DebugOn       ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = value
        value = 40h; Start command
        mask = F0h ; Mask out the channel number

    [End]
        type = timeout
        timeout = 3000 ; 3ms timeout ends the packet

    [Decode]
    [Fields]


        Fields
            Start.4.h,
            Channel.4=1.h,
            Command.8.h,
            X.16.d/20.48-25$g,
            Y.16.d/20.48-25$g,
            Z.16.d/20.48-25$g,
            Rest.N.h    ; Rest of the packet

        Fields
            Rest.N.h    ; Rest of the packet
```



```
259.3443ms, Async-4, CH4 41 C0 02 07 02 28 02 13 D9 42 41 C0 02 06 02 28 02 10 D9 42 41 C0 02 06 02
431.8247ms, Async-4, CH4 02 06 02 28 02 10 D9 42 41 C0 02 06 02 28 02 10 D9 42 41 C0 02 06 02 27 02
587.9587ms, Async-4, CH4 28 02 10 D9 42 41 C0 02 06 02 28 02 10 D9 42 41 C0 02 06 02 28 02 10 D9 42
754.9108ms, Async-4, CH4 10 D9 42 41 C0 02 0C 02 28 02 10 D9 42 41 C0 02 06 02 28 02 10 D9 42 41 C0
940.9932ms, Async-4, CH4 42 41 C0 02 07 02 2E 02 0F D9 42 41 C0 02 06 02 28 02 10 D9 42 41 C0 02 06
  1.1434373s , Async-4, CH4 C0 02 06 02 28 02 10 D9 42 41 C0 02 06 02 28 02 10 D9 42 41 C0 02 06 02
```

X1

Layer: ASYNCBUS	START	CHANNEL	COMMAND	X	Y	Z	REST
Time: 259.3443ms	4	1	C0	0.34g	1.95g	0.93g	D9

Layer: ASYNCBUS	REST
Time: 278.2827ms	42

X2

Layer: ASYNCBUS	START	CHANNEL	COMMAND	X	Y	Z	REST
Time: 319.9042ms	4	1	C0	0.29g	1.95g	0.78g	D9

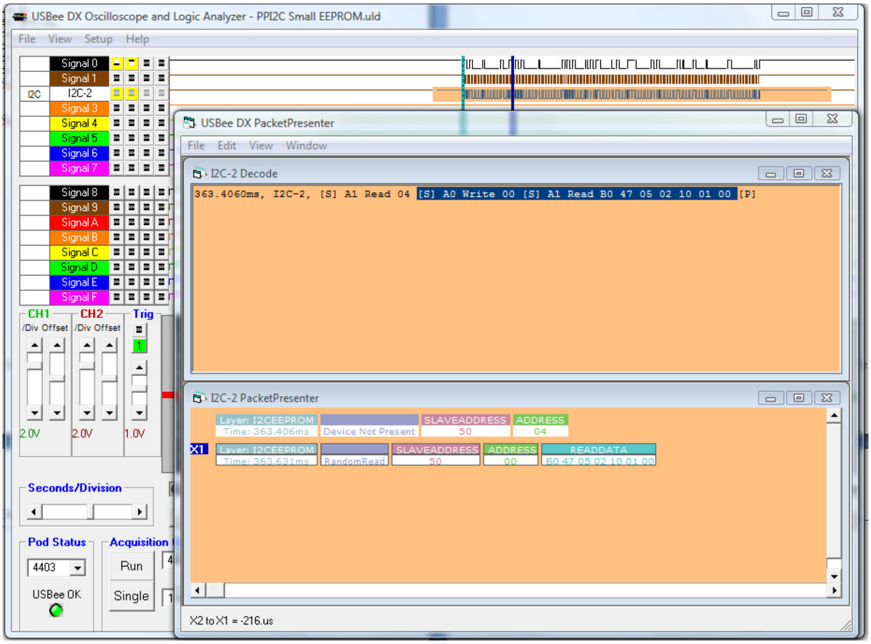
Layer: ASYNCBUS	REST
Time: 334.6842ms	42

Layer: ASYNCBUS	START	CHANNEL	COMMAND	X	Y	Z	REST
Time: 375.795ms	4	1	C0	0.29g	1.95g	0.78g	D9

X2 to X1 = 60.5598ms

I2C PROTOCOL EXAMPLE

```
; I2C EEPROM Protocol Definition File
; This file defines the transfers to/from an I2C EEPROM
; with 8 bit address
;
[Protocol]
     name = I2CEEPROM
     bytewise
[DEBUG]
     ;DebugOn       ; Uncomment this to turn on Debug Packets
[Packet]
     [Start]
          type = event
          event = 1 ; Start Bit

     [End]
          type = event
          event = 0Ah    ; Stop Bit Or NACK

     [Decode]
     [Fields]

          ; Device Not Present
          Fields
               $Device Not Present,               ; Printout this label if match
               SlaveAddress.7m.h,RW.1.i,          ; Control Byte
               Address.8m.h,                      ; 1 byte address
               [8]                                ; followed by a NACK condition

          ; Set Address
          Fields
               $SetAddressCmd,                    ; Printout this label if match
               SlaveAddress.7m.h,RW.1=0.i,        ; Control Byte
               Address.8m.h,                      ; 1 byte address
               [2]                                ; followed by a STOP condition

          ; Write Command
          Fields
               $WriteCommand,                     ; Printout this label if match
               SlaveAddress.7m.h,RW.1=0.i,        ; Control Byte
               Address.8m.h,                      ; 1 byte address
               [!1],                              ; NO START condition
               WriteData.Nm.h                     ; Written Data (Variable N)

          ; Current Address Read
          Fields
               $CurrentRead,                      ; Printout this label if match
               SlaveAddress.7m.h,RW.1=1.i,        ; Control Byte
               ReadData.Nm.h                      ; Read Data (Variable number N)

          ; Random Read
          Fields
               $RandomRead,                       ; Printout this label if match
               SlaveAddress.7m.h,RW.1=0.i,        ; Control Byte
               Address.8m.h,                      ; 1 byte address
               [1],                               ; START Condition
               SlaveAddress.7m.i,RW.1=1.i,        ; Control Byte
               ReadData.Nm.h,                     ; Read Data (Variable number N)
```

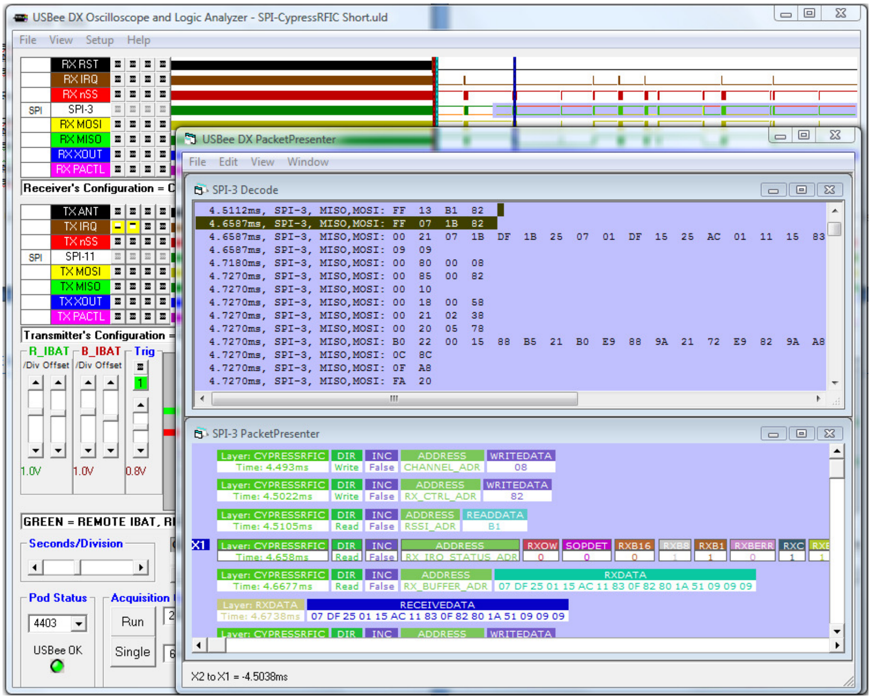


```
363.4060ms, I2C-2, [S] A1 Read 04 [S] A0 Write 00 [S] A1 Read B0 47 05 02 10 01 00 [P]
```

	Layer: I2CEEPROM		SLAVEADDRESS	ADDRESS	
	Time: 363.406ms	Device Not Present	50	04	
X1	Layer: I2CEEPROM		SLAVEADDRESS	ADDRESS	READDATA
	Time: 363.631ms	RandomRead	50	00	B0 47 05 02 10 01 00

X2 to X1 = -216.us

```
; Cypress RF IC Protocol Definition File
; This file defines the transfers to/from a CY6936 RF IC
; using the SPI bus
[Protocol]
     name = CypressRFIC
     bytewise
[DEBUG]
     ;DebugOn
[Packet]
     [Start]
          type = event
          event = 1 ; SS goes active

     [End]
          type = event
          event = 2 ; SS goes inactive
     [Decode]
     [Fields]
          ; RX_IRQ_STATUS_ADR Read and Write Command
          Fields    Dir.1y=0.L, Inc.1y.tf, Address.6y=07h.L, Dummy.8x.i, RXOW.1x.h,
                    SOPDET.1x.h, RXB16.1x.h, RXB8.1x.h, RXB1.1x.h, RXBERR.1x.h, RXC.1x.h,
                    RXE.1x.h
          Fields    Dir.1y=1.L, Inc.1y.tf, Address.6y=07h.L, RXOW.1y.h, SOPDET.1y.h,
                    RXB16.1y.h, RXB8.1y.h, RXB1.1y.h, RXBERR.1y.h, RXC.1y.h, RXE.1y.h

          ; TX_IRQ_STATUS_ADR Read and Write Command
          Fields    Dir.1y=0.L, Inc.1y.tf, Address.6y=04h.L, Dummy.8x.i, OS.1x.h, LV.1x.h,
                    TXB15.1x.h, TXB8.1x.h, TXB1.1x.h, TXBERR.1x.h, TXC.1x.h, TXE.1x.h
          Fields    Dir.1y=1.L, Inc.1y.tf, Address.6y=04h.L, OS.1y.h, LV.1y.h, TXB15.1y.h,
                    TXB8.1y.h, TXB1.1y.h, TXBERR.1y.h, TXC.1y.h, TXE.1y.h

          ; RX_BUFFER_ADR Read and Write Command
          Fields    Dir.1y=0.L,    Inc.1y.tf,     Address.6y=21h.L,  Dummy.8x.i,
                    RxData.Nx.h
          Fields    Dir.1y=1.L,    Inc.1y.tf,     Address.6y=21h.L,  RxData.Ny.h


          ; TX_BUFFER_ADR Read and Write Command
          Fields    Dir.1y=0.L,    Inc.1y.tf,     Address.6y=20h.L,  Dummy.8x.i,
                    TxData.Nx.h
          Fields    Dir.1y=1.L,    Inc.1y.tf,     Address.6y=20h.L,  TxData.Ny.h

          Fields    Dir.1y=0.L,    Inc.1y.tf,     Address.6y.L,      Dummy.8x.i,
                    ReadData.Nx.h
          Fields    Dir.1y=1.L,    Inc.1y.tf,     Address.6y.L,      WriteData.Nmy.h

          Lookup Dir
              [0]=$Read
              [1]=$Write

          Lookup Address
              [00h]=$CHANNEL_ADR
              [01h]=$TX_LENGTH_ADR
              [02h]=$TX_CTRL_ADR
              [03h]=$TX_CFG_ADR
              [04h]=$TX_IRQ_STATUS_ADR
              [05h]=$RX_CTRL_ADR
              [06h]=$RX_CFG_ADR
              [07h]=$RX_IRQ_STATUS_ADR
              [08h]=$RX_STATUS_ADR
              [09h]=$RX_COUNT_ADR
              [0ah]=$RX_LENGTH_ADR
              [0bh]=$PWR_CTRL_ADR
              [0ch]=$XTAL_CTRL_ADR
              [0dh]=$IO_CFG_ADR
              [0eh]=$GPIO_CTRL_ADR
              [0fh]=$XACT_CFG_ADR
              [10h]=$FRAMING_CFG_ADR
              [11h]=$DATA32_THOLD_ADR
              [12h]=$DATA64_THOLD_ADR
              [13h]=$RSSI_ADR
              [14h]=$EOP_CTRL_ADR
              [15h]=$CRC_SEED_LSB_ADR
              [16h]=$CRC_SEED_MSB_ADR
              [17h]=$TX_CRC_LSB_ADR
              [18h]=$TX_CRC_MSB_ADR
              [19h]=$RX_CRC_LSB_ADR
              [1ah]=$RX_CRC_MSB_ADR
              [1bh]=$TX_OFFSET_LSB_ADR
              [1ch]=$TX_OFFSET_MSB_ADR
              [1dh]=$MODE_OVERRIDE_ADR
              [1eh]=$RX_OVERRIDE_ADR
              [1fh]=$TX_OVERRIDE_ADR
              [26h]=$XTAL_CFG_ADR
              [27h]=$CLK_OVERRIDE_ADR
              [28h]=$CLK_EN_ADR
              [29h]=$RX_ABORT_ADR
              [32h]=$AUTO_CAL_TIME_ADR
              [35h]=$AUTO_CAL_OFFSET_ADR
              [39h]=$ANALOG_CTRL_ADR
              [20h]=$TX_BUFFER_ADR
              [21h]=$RX_BUFFER_ADR
              [22h]=$SOP_CODE_ADR
              [23h]=$DATA_CODE_ADR
              [24h]=$PREAMBLE_ADR
              [25h]=$MFG_ID_ADR

[Protocol]
     name = RxData
     bytewise
[DEBUG]
     ;DebugOn
[Packet]
     [Start]
          type = next
     [End]
          type = event
          event = 127     ; All Data passed in

     [Decode]
     [Fields]
          ; RX_IRQ_STATUS_ADR Read and Write Command
          Fields     ReceiveData.N.h
```



```
4.5112ms, SPI-3, MISO,MOSI: FF  13  B1  82
4.6587ms, SPI-3, MISO,MOSI: FF  07  1B  82
4.6587ms, SPI-3, MISO,MOSI: 00  21  07  1B  DF  1B  25  07  01  DF  15  25  AC  01  11  15  83
4.6587ms, SPI-3, MISO,MOSI: 09  09
4.7180ms, SPI-3, MISO,MOSI: 00  80  00  08
4.7270ms, SPI-3, MISO,MOSI: 00  85  00  82
4.7270ms, SPI-3, MISO,MOSI: 00  10
4.7270ms, SPI-3, MISO,MOSI: 00  18  00  58
4.7270ms, SPI-3, MISO,MOSI: 00  21  02  38
4.7270ms, SPI-3, MISO,MOSI: 00  20  05  78
4.7270ms, SPI-3, MISO,MOSI: B0  22  00  15  88  B5  21  B0  E9  88  9A  21  72  E9  82  9A  A8
4.7270ms, SPI-3, MISO,MOSI: 0C  8C
4.7270ms, SPI-3, MISO,MOSI: 0F  A8
4.7270ms, SPI-3, MISO,MOSI: FA  20
```

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	WRITEDATA
Time: 4.493ms	Write	False	CHANNEL_ADR	08

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	WRITEDATA
Time: 4.5022ms	Write	False	RX_CTRL_ADR	82

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	READDATA
Time: 4.5105ms	Read	False	RSSI_ADR	B1

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXOW	SOPDET	RXB16	RXB8	RXB1	RXBERR	RXC	RXE
Time: 4.658ms	Read	False	RX_IRQ_STATUS_ADR	0	0	0	1	1	0	1	1

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXDATA
Time: 4.6677ms	Read	False	RX_BUFFER_ADR	07 DF 25 01 15 AC 11 83 0F 82 80 1A 51 09 09 09

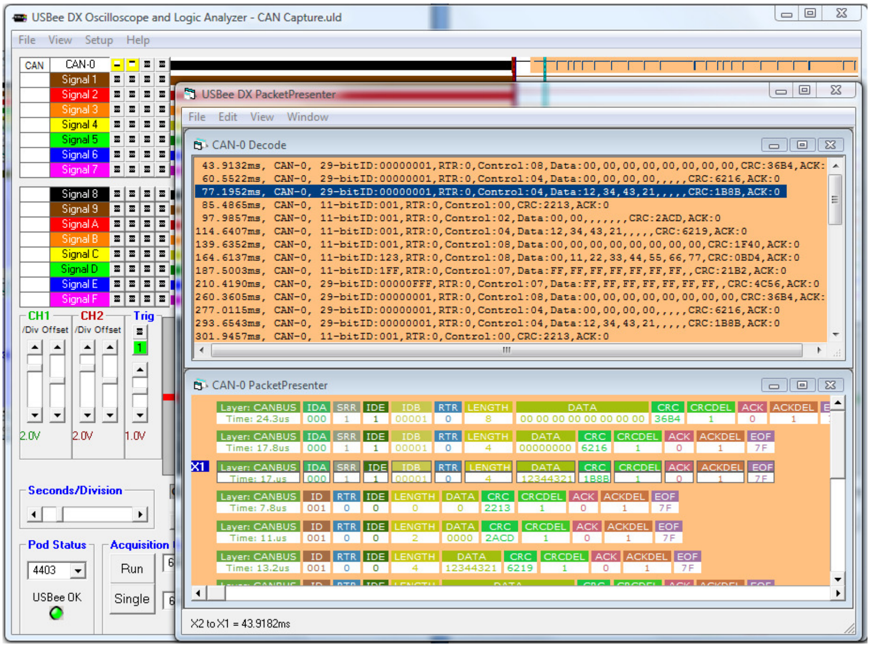
Layer: RXDATA	RECEIVEDATA
Time: 4.6738ms	07 DF 25 01 15 AC 11 83 0F 82 80 1A 51 09 09 09

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	WRITEDATA

X2 to X1 = -4.5038ms

CAN PROTOCOL EXAMPLE

```
; CAN Protocol Definition File
; This file defines the transfers to/from a custom CAN device
; over a the CAN bus
;
[Protocol]
          name = CANBus
          bitwise
[DEBUG]
          ;DebugOn            ; Uncomment this to turn on Debug Packets
[Packet]
          [Start]
                    type = event
                    event = 1 ; Start of CAN packet
          [End]
                    type = event
                    event = 2 ; End of CAN packet
          [Decode]
          [Fields]

                    ; Extended Frame Format
                    Fields    SOF.1.i, IDA.11.h, SRR.1.h, IDE.1=1.h, IDB.18.h, RTR.1.h,
                              Rsrv.2.i, Length.4.h, Data.N.h, CRC.15.h, CRCDel.1.h,
                              ACK.1.h, ACKDel.1.h, EOF.7.h

                    ; Base frame format
                    Fields    SOF.1.i, ID.11.h, RTR.1.h, IDE.1=0.h, Rsrv.1.i, Length.4.h,
                              Data.N.h, CRC.15.h, CRCDel.1.h, ACK.1.h, ACKDel.1.h,
                              EOF.7.h
```



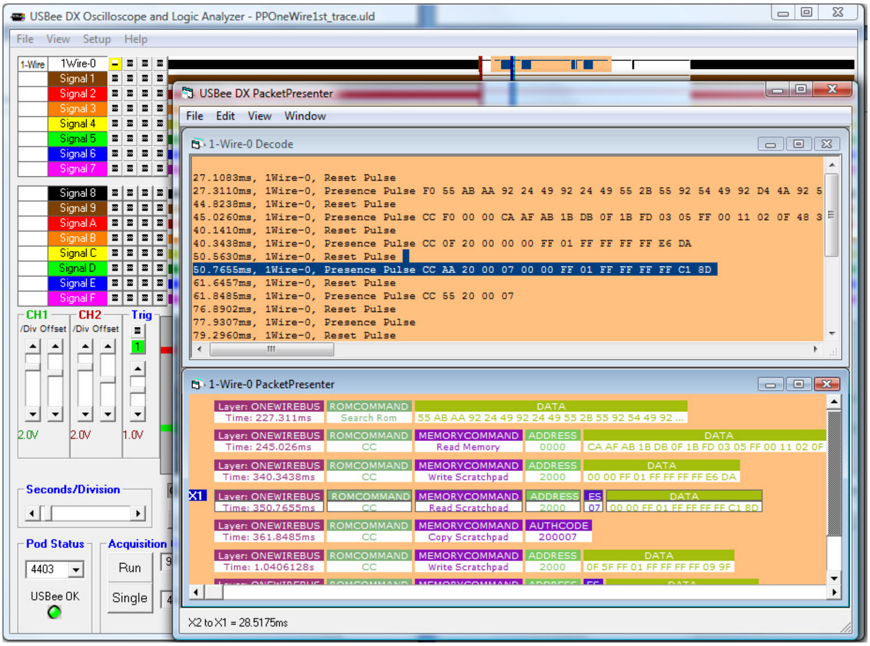
```
 43.9132ms, CAN-0, 29-bitID:00000001,RTR:0,Control:08,Data:00,00,00,00,00,00,00,00,CRC:36B4,ACK:
 60.5522ms, CAN-0, 29-bitID:00000001,RTR:0,Control:04,Data:00,00,00,00,,,,,CRC:6216,ACK:0
 77.1952ms, CAN-0, 29-bitID:00000001,RTR:0,Control:04,Data:12,34,43,21,,,,,CRC:1B8B,ACK:0
 85.4865ms, CAN-0, 11-bitID:001,RTR:0,Control:00,CRC:2213,ACK:0
 97.9857ms, CAN-0, 11-bitID:001,RTR:0,Control:02,Data:00,00,,,,,,,CRC:2ACD,ACK:0
114.6407ms, CAN-0, 11-bitID:001,RTR:0,Control:04,Data:12,34,43,21,,,,,CRC:6219,ACK:0
139.6352ms, CAN-0, 11-bitID:001,RTR:0,Control:08,Data:00,00,00,00,00,00,00,00,CRC:1F40,ACK:0
164.6137ms, CAN-0, 11-bitID:123,RTR:0,Control:08,Data:00,11,22,33,44,55,66,77,CRC:0BD4,ACK:0
187.5003ms, CAN-0, 11-bitID:1FF,RTR:0,Control:07,Data:FF,FF,FF,FF,FF,FF,FF,,CRC:21B2,ACK:0
210.4190ms, CAN-0, 29-bitID:00000FFF,RTR:0,Control:07,Data:FF,FF,FF,FF,FF,FF,FF,,CRC:4C56,ACK:0
260.3605ms, CAN-0, 29-bitID:00000001,RTR:0,Control:08,Data:00,00,00,00,00,00,00,00,CRC:36B4,ACK:
277.0115ms, CAN-0, 29-bitID:00000001,RTR:0,Control:04,Data:00,00,00,00,,,,,CRC:6216,ACK:0
293.6543ms, CAN-0, 29-bitID:00000001,RTR:0,Control:04,Data:12,34,43,21,,,,,CRC:1B8B,ACK:0
301.9457ms, CAN-0, 11-bitID:001,RTR:0,Control:00,CRC:2213,ACK:0
```

```
   Layer: CANBUS  IDA  SRR  IDE  IDB    RTR  LENGTH  DATA                     CRC   CRCDEL  ACK  ACKDEL  E
   Time: 24.3us   000  1    1    00001  0    8       00 00 00 00 00 00 00 00  36B4  1       0    1

   Layer: CANBUS  IDA  SRR  IDE  IDB    RTR  LENGTH  DATA      CRC   CRCDEL  ACK  ACKDEL  EOF
   Time: 17.8us   000  1    1    00001  0    4       00000000  6216  1       0    1       7F

X1 Layer: CANBUS  IDA  SRR  IDE  IDB    RTR  LENGTH  DATA      CRC   CRCDEL  ACK  ACKDEL  EOF
   Time: 17.us    000  1    1    00001  0    4       12344321  1B8B  1       0    1       7F

   Layer: CANBUS  ID   RTR  IDE  LENGTH  DATA  CRC   CRCDEL  ACK  ACKDEL  EOF
   Time: 7.8us    001  0    0    0       0     2213  1       0    1       7F

   Layer: CANBUS  ID   RTR  IDE  LENGTH  DATA  CRC   CRCDEL  ACK  ACKDEL  EOF
   Time: 11.us    001  0    0    2       0000  2ACD  1       0    1       7F

   Layer: CANBUS  ID   RTR  IDE  LENGTH  DATA      CRC   CRCDEL  ACK  ACKDEL  EOF
   Time: 13.2us   001  0    0    4       12344321  6219  1       0    1       7F
```

X2 to X1 = 43.9182ms

1-WIRE PROTOCOL EXAMPLE

```
; One Wire Protocol Definition File
; This file defines the transfers to/from some 1-Wire devices
; using the 1-Wire bus
;
[Protocol]
          name = OneWireBus
          bytewise
[DEBUG]
          ;DebugOn            ; Uncomment this to turn on Debug Packets
[Packet]
          [Start]
                    type = event
                    event = 2 ; Presence Pulse

          [End]
                    type = event
                    event = 1 ; Reset Pulse

          [Decode]
          [Fields]

                    ; These fields are used by Maxim/Dallas Digital Thermometers
                    Fields    ROMCommand.8=F0h.$Search Rom, Data.N.h
                    Fields    ROMCommand.8=33h.$Read Rom, Family.8.h, SerialNumber.48.h,
                              CRC.8.h
                    Fields    ROMCommand.8=55h.$Match Rom, Family.8.h, SerialNumber.48.h,
CRC.8.h
                    Fields    ROMCommand.8=CCh.$Skip ROM, Function.8=44h.$ConvertTemp
                    Fields    ROMCommand.8=CCh.$Skip ROM, Function.8=BEh.$Read Scratchpad,
Temp.16.d, TH.8.h, TL.8l.h, Rsvd.16.i, Remain.8.h, CpC.8.h, CRC.8.h

                    ; These fields are used by Dallas Serial Number iButtons
                    Fields    ROMCommand.8=33h.$Read Rom, Family.8.h, SerialNumber.48.h,
CRC.8.h
                    Fields    ROMCommand.8=0Fh.$Read Rom, Family.8.h, SerialNumber.48.h,
CRC.8.h


                    ; These packets are used by 1-Wire EEPROMS
                    Fields    ROMCommand.8=33h.$Read Rom, Family.8.h, SerialNumber.48.h,
CRC.8.h
                    Fields    ROMCommand.8.h, MemoryCommand.8=0Fh.$Write Scratchpad,
                                        Address.16.h, Data.N.h
                    Fields    ROMCommand.8.h, MemoryCommand.8=AAh.$Read Scratchpad,
                                        Address.16.h, ES.8.h, Data.N.h
                    Fields    ROMCommand.8.h, MemoryCommand.8=55h.$Copy Scratchpad,
                                        AuthCode.24.h
                    Fields    ROMCommand.8.h, MemoryCommand.8=F0h.$Read Memory,
                                            Address.16.h, Data.N.h
```



```
27.1083ms, 1Wire-0, Reset Pulse
27.3110ms, 1Wire-0, Presence Pulse F0 55 AB AA 92 24 49 92 24 49 55 2B 55 92 54 49 92 D4 4A 92 5
44.8238ms, 1Wire-0, Reset Pulse
45.0260ms, 1Wire-0, Presence Pulse CC F0 00 00 CA AF AB 1B DB 0F 1B FD 03 05 FF 00 11 02 0F 48 3
40.1410ms, 1Wire-0, Reset Pulse
40.3438ms, 1Wire-0, Presence Pulse CC 0F 20 00 00 00 FF 01 FF FF FF FF E6 DA
50.5630ms, 1Wire-0, Reset Pulse
50.7655ms, 1Wire-0, Presence Pulse CC AA 20 00 07 00 00 FF 01 FF FF FF FF C1 8D
61.6457ms, 1Wire-0, Reset Pulse
61.8485ms, 1Wire-0, Presence Pulse CC 55 20 00 07
76.8902ms, 1Wire-0, Reset Pulse
77.9307ms, 1Wire-0, Presence Pulse
79.2960ms, 1Wire-0, Reset Pulse
```

Layer: ONEWIREBUS	ROMCOMMAND	DATA			
Time: 227.311ms	Search Rom	55 AB AA 92 24 49 92 24 49 55 2B 55 92 54 49 92 ...			

Layer: ONEWIREBUS	ROMCOMMAND	MEMORYCOMMAND	ADDRESS	DATA
Time: 245.026ms	CC	Read Memory	0000	CA AF AB 1B DB 0F 1B FD 03 05 FF 00 11 02 0F

Layer: ONEWIREBUS	ROMCOMMAND	MEMORYCOMMAND	ADDRESS	DATA
Time: 340.3438ms	CC	Write Scratchpad	2000	00 00 FF 01 FF FF FF FF E6 DA

X1

Layer: ONEWIREBUS	ROMCOMMAND	MEMORYCOMMAND	ADDRESS	ES	DATA
Time: 350.7655ms	CC	Read Scratchpad	2000	07	00 00 FF 01 FF FF FF FF C1 8D

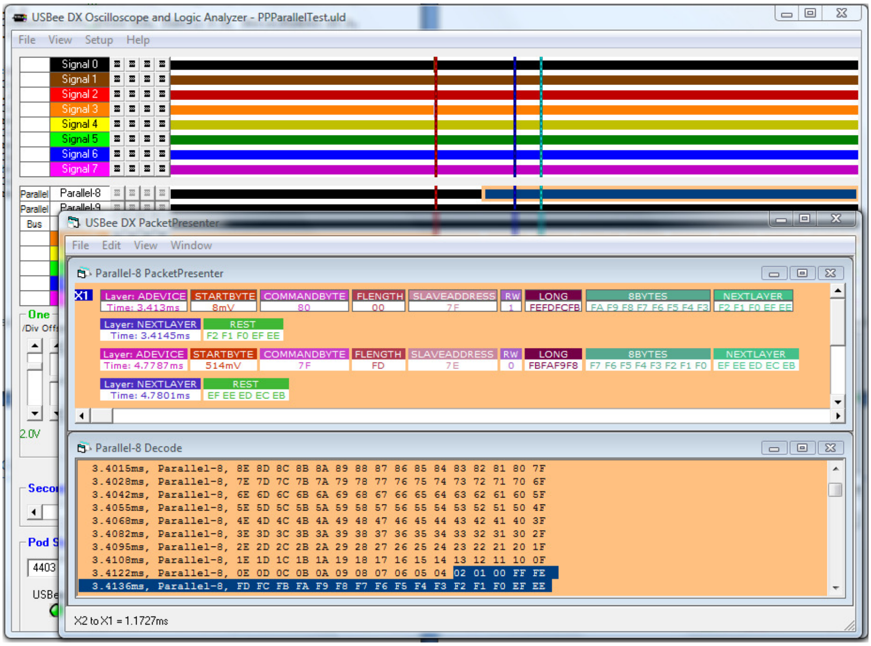
Layer: ONEWIREBUS	ROMCOMMAND	MEMORYCOMMAND	AUTHCODE
Time: 361.8485ms	CC	Copy Scratchpad	200007

Layer: ONEWIREBUS	ROMCOMMAND	MEMORYCOMMAND	ADDRESS	DATA
Time: 1.0406128s	CC	Write Scratchpad	2000	0F 5F FF 01 FF FF FF FF 09 9F

X2 to X1 = 28.5175ms

PARALLEL PROTOCOL EXAMPLE

```
; Sample Parallel Protocol Definition File
; This file defines the transfers to/from an unique device
;
[Protocol]
    name = ADevice
    bytewise
[DEBUG]
    ;DebugOn
[Packet]
    [Start]
        type = signal
        signal = 14
        level = 0

    [End]
        type = length
        Bytelength = 21

    [Decode]
    [Fields]
        Fields
            StartByte.8m.d*2+4$mV,
            CommandByte.8l.L,
            FLength.8m.h,
            SlaveAddress.7m.h,RW.1.L,
            Long.32m.h,
            8Bytes.64m.h,
            NextLayer.Nm.h


[Protocol]
    name = NextLayer
    bytewise
[Packet]
    [Start]
        type = next
    [End]
        type = Event   ;End of a packet is signaled by a event
        event = 127; Means the end of the data (only for higher
layers)

    [Decode]
    [Fields]
        Fields
            Rest.N.h       ; Just print out all the bytes
```



Layer: ADEVICE	STARTBYTE	COMMANDBYTE	FLENGTH	SLAVEADDRESS	RW	LONG	8BYTES	NEXTLAYER
Time: 3.413ms	8mV	80	00	7F	1	FEFDFCFB	FA F9 F8 F7 F6 F5 F4 F3	F2 F1 F0 EF EE

Layer: NEXTLAYER	REST
Time: 3.4145ms	F2 F1 F0 EF EE

Layer: ADEVICE	STARTBYTE	COMMANDBYTE	FLENGTH	SLAVEADDRESS	RW	LONG	8BYTES	NEXTLAYER
Time: 4.7787ms	514mV	7F	FD	7E	0	FBFAF9F8	F7 F6 F5 F4 F3 F2 F1 F0	EF EE ED EC EB

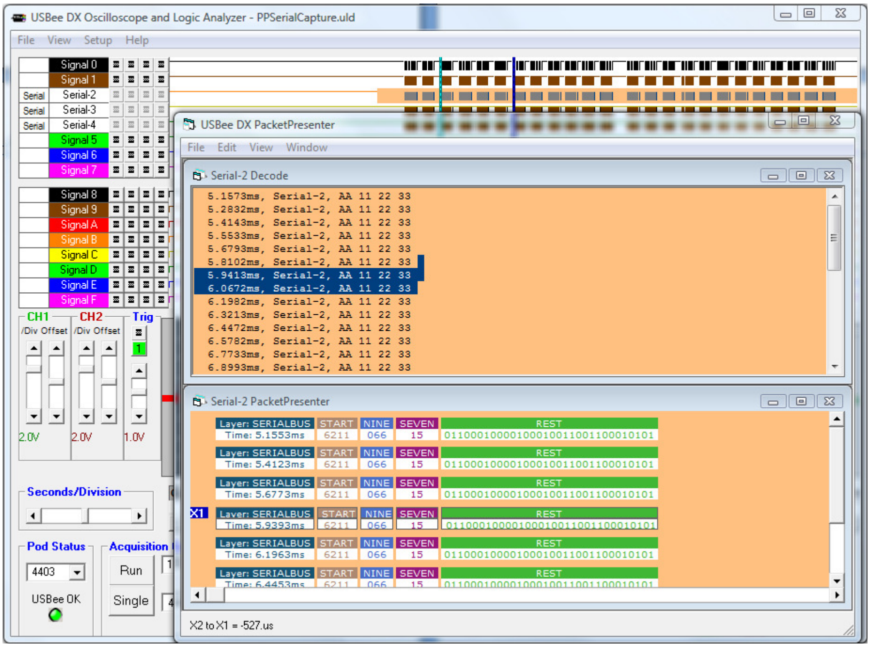
Layer: NEXTLAYER	REST
Time: 4.7801ms	EF EE ED EC EB

```
3.4015ms, Parallel-8, 8E 8D 8C 8B 8A 89 88 87 86 85 84 83 82 81 80 7F
3.4028ms, Parallel-8, 7E 7D 7C 7B 7A 79 78 77 76 75 74 73 72 71 70 6F
3.4042ms, Parallel-8, 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60 5F
3.4055ms, Parallel-8, 5E 5D 5C 5B 5A 59 58 57 56 55 54 53 52 51 50 4F
3.4068ms, Parallel-8, 4E 4D 4C 4B 4A 49 48 47 46 45 44 43 42 41 40 3F
3.4082ms, Parallel-8, 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32 31 30 2F
3.4095ms, Parallel-8, 2E 2D 2C 2B 2A 29 28 27 26 25 24 23 22 21 20 1F
3.4108ms, Parallel-8, 1E 1D 1C 1B 1A 19 18 17 16 15 14 13 12 11 10 0F
3.4122ms, Parallel-8, 0E 0D 0C 0B 0A 09 08 07 06 05 04 02 01 00 FF FE
3.4136ms, Parallel-8, FD FC FB FA F9 F8 F7 F6 F5 F4 F3 F2 F1 F0 EF EE
```

X2 to X1 = 1.1727ms

SERIAL PROTOCOL EXAMPLE

```
; Serial Protocol Definition File
; This file defines the transfers from a serial device
;
[Protocol]
    name = SerialBus
    bitwise
[DEBUG]
    ;DebugOn        ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = value    ; Look for a value in the data to start the
packet
        value = 6211h   ; NOTE: This value is assumed MSbit first in
                        ; the data stream!
        bits = 16
        mask = FFFFh
    [End]
        type = length
        bitlength = 64 ; End of command after 64 bits
    [Decode]
    [Fields]

        ; Send out the bits of the packet
        Fields Start.16.h, Nine.9.h, Seven.7.h, Rest.N.b
```



```
5.1573ms, Serial-2, AA 11 22 33
5.2832ms, Serial-2, AA 11 22 33
5.4143ms, Serial-2, AA 11 22 33
5.5533ms, Serial-2, AA 11 22 33
5.6793ms, Serial-2, AA 11 22 33
5.8102ms, Serial-2, AA 11 22 33
5.9413ms, Serial-2, AA 11 22 33
6.0672ms, Serial-2, AA 11 22 33
6.1982ms, Serial-2, AA 11 22 33
6.3213ms, Serial-2, AA 11 22 33
6.4472ms, Serial-2, AA 11 22 33
6.5782ms, Serial-2, AA 11 22 33
6.7733ms, Serial-2, AA 11 22 33
6.8993ms, Serial-2, AA 11 22 33
```

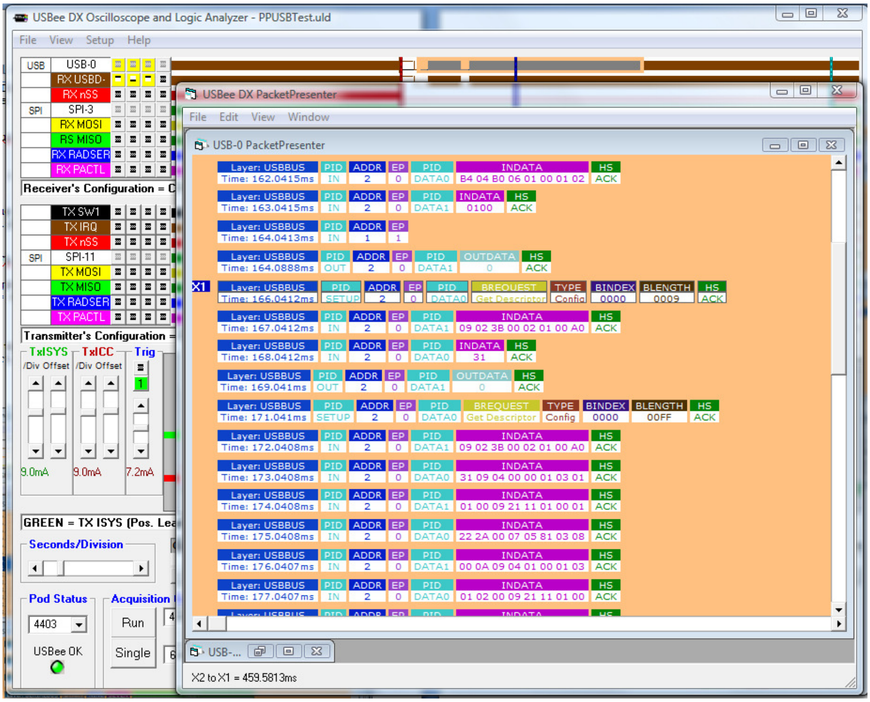
```
Layer: SERIALBUS  START  NINE  SEVEN  REST
Time: 5.1553ms    6211   066   15     01100010000100010011001100010101
Layer: SERIALBUS  START  NINE  SEVEN  REST
Time: 5.4123ms    6211   066   15     01100010000100010011001100010101
Layer: SERIALBUS  START  NINE  SEVEN  REST
Time: 5.6773ms    6211   066   15     01100010000100010011001100010101
Layer: SERIALBUS  START  NINE  SEVEN  REST
Time: 5.9393ms    6211   066   15     01100010000100010011001100010101
Layer: SERIALBUS  START  NINE  SEVEN  REST
Time: 6.1963ms    6211   066   15     01100010000100010011001100010101
Layer: SERIALBUS  START  NINE  SEVEN  REST
Time: 6.4453ms    6211   066   15     01100010000100010011001100010101
```

X2 to X1 = -527.us

USB PROTOCOL EXAMPLE

```
; USB Bus Protocol Definition File
; This file defines the transfers to/from a custom USB device
;
[Protocol]
          name = USBBus
          bitwise
[DEBUG]
          ;DebugOn            ; Uncomment this to turn on Debug Packets
[Packet]
          [Start]
                    type = event
                    event = 1 ; Setup/In or Out found
          [End]
                    type = event
                    event = 6 ; ACK, NAK or Stall found or no handshake found
          [Decode]
          [Fields]

                    ; Any Packet - No Response
                    Fields    Sync.8.i, PID.8.L, Addr.7l.d, EP.4l.d, CRC5.5.i, ; Token
                              [4]                                 ; No Handshake

                    ; Setup – Nakd                                ; Token
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, HS.8=01011010b.L          ; Handshake

                    ; IN - Nakd
                    Fields    Sync.8.i, PID.8=10010110b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
                              Sync.8.i, HS.8=01011010b.L         ; Handshake


                    ; OUT - Nakd
                    Fields    Sync.8.i, PID.8=10000111b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
                              Sync.8.i, HS.8=01011010b.L          ; Handshake

                    ; Setup
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=1.$Clear Feature, bValue.16L.h, bIndex.16L.H,
                              bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L

                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=0.$Get Status, bValue.16L.h, bIndex.16L.H,
                              bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=8.$Get Configuration, bValue.16L.h, bIndex.16L.H,
                              bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=6.$Get Descriptor, bValueL.8L.I, Type.8L.L,
                              bIndex.16L.H, bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=16.$Get Interface, bValue.16L.h, bIndex.16L.H,
                              bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=5.$Set Address, Address.16L.h, bLength.16L.i,
                              bLength.16L.i, CRC16.16.i, Sync.8.i, HS.8.L
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=9.$Set Configuration, Config.16L.h,
                              bLength.16L.i, bLength.16L.i, CRC16.16.i, Sync.8.i, HS.8.L
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=7.$Set Descriptor, bValue.16L.h, bIndex.16L.H,
                              bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
                    Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
                              Sync.8.i, PID.8.L, Rtype.8.i,
                              bRequest.8L=3.$Set Feature, bValue.16L.h, bIndex.16L.H,
                              bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
```

```
Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=10.$Get Interface, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H,  CRC16.16.i, Sync.8.i, HS.8.L
Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=11.$Set Interface, AltSetting.16L.h,
          Interface.16L.H, bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
Fields    Sync.8.i, PID.8=10110100b.L, Addr.7l.d, EP.4l.d, CRC5.5.i,
          Sync.8.i, PID.8.L, Rtype.8.i,
          bRequest.8L=12.$Sync Frame, bValue.16L.h, bIndex.16L.H,
          bLength.16L.H, CRC16.16.i, Sync.8.i, HS.8.L
; IN
Fields    Sync.8.i, PID.8=10010110b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, InData.NL.h, CRC16.16.i,   ; Data
          Sync.8.i, HS.8.L                    ; Handshake
; OUT
Fields    Sync.8.i, PID.8=10000111b.L, Addr.7L.d, EP.4L.d, CRC5.5.i,
          Sync.8.i, PID.8.L, OutData.NL.h, CRC16.16.i,   ; Data
          Sync.8.i, HS.8.L                    ; Handshake

; Catch all
Fields Data.NL.h

Lookup    Type
          [1]=$Device
          [2]=$Config
          [3]=$String

Lookup    PID
          [11000011b]=$DATA0
          [11010010b]=$DATA1
          [01001011b]=$ACK
          [01011010b]=$NAK
          [01111000b]=$STALL
          [10110100b]=$SETUP
          [10000111b]=$OUT
          [10010110b]=$IN
          [10100101b]=$SOF

Lookup    HS
          [01001011b]=$ACK
          [01011010b]=$NAK
          [01111000b]=$STALL
```

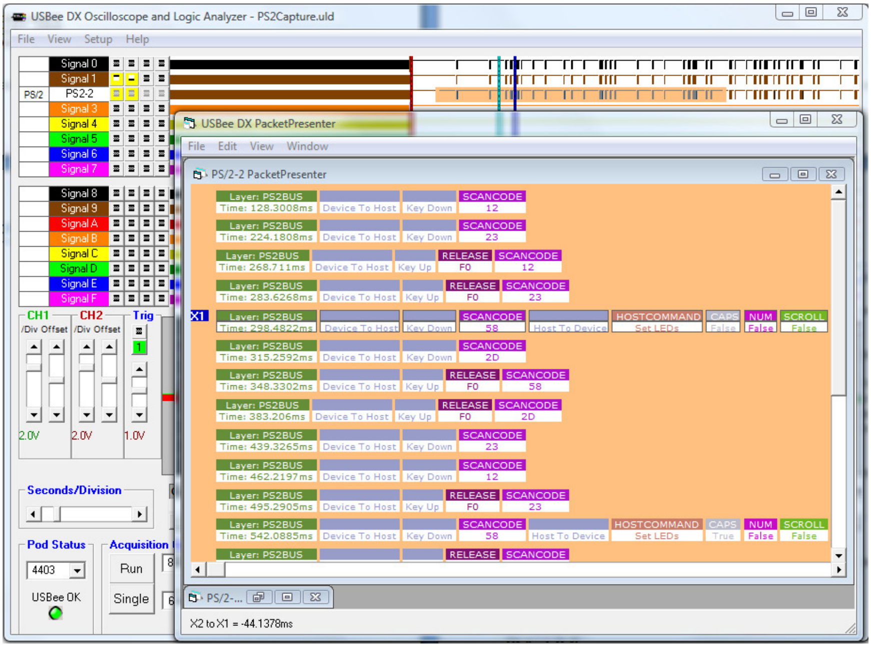


```
   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 162.0415ms IN     2     0   DATA0   B4 04 B0 06 01 00 01 02   ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA  HS
   Time: 163.0415ms IN     2     0   DATA1   0100    ACK

   Layer: USBBUS    PID    ADDR  EP
   Time: 164.0413ms IN     1     1

   Layer: USBBUS    PID    ADDR  EP  PID     OUTDATA  HS
   Time: 164.0888ms OUT    2     0   DATA1   0        ACK

X1 Layer: USBBUS    PID    ADDR  EP  PID     BREQUEST        TYPE    BINDEX  BLENGTH  HS
   Time: 166.0412ms SETUP  2     0   DATA0   Get Descriptor  Config  0000    0009     ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 167.0412ms IN     2     0   DATA1   09 02 3B 00 02 01 00 A0   ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA  HS
   Time: 168.0412ms IN     2     0   DATA0   31      ACK

   Layer: USBBUS    PID    ADDR  EP  PID     OUTDATA  HS
   Time: 169.041ms  OUT    2     0   DATA1   0        ACK

   Layer: USBBUS    PID    ADDR  EP  PID     BREQUEST        TYPE    BINDEX  BLENGTH  HS
   Time: 171.041ms  SETUP  2     0   DATA0   Get Descriptor  Config  0000    00FF     ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 172.0408ms IN     2     0   DATA1   09 02 3B 00 02 01 00 A0   ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 173.0408ms IN     2     0   DATA0   31 09 04 00 00 01 03 01   ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 174.0408ms IN     2     0   DATA1   01 00 09 21 11 01 00 01   ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 175.0408ms IN     2     0   DATA0   22 2A 00 07 05 81 03 08   ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 176.0407ms IN     2     0   DATA1   00 0A 09 04 01 00 01 03   ACK

   Layer: USBBUS    PID    ADDR  EP  PID     INDATA                    HS
   Time: 177.0407ms IN     2     0   DATA0   01 02 00 09 21 11 01 00   ACK
```

X2 to X1 = 459.5813ms

PS2 PROTOCOL EXAMPLE

```
; PS2 Protocol Definition File
; This file defines the transfers from a PS2 device
;
[Protocol]
    name = PS2Bus
    bytewise
[DEBUG]
    ;DebugOn       ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = next    ; Every byte is the start of the next packet
        CHANNELXORY    ; Either Device to Host or Host To Device
    [End]
        type = TIMEOUT
        TIMEOUT = 5000 ; End of command after 5msec
    [Decode]
    [Fields]

        ; Setting LEDs after command
        Fields [1], $Device To Host, $Key Down, Scancode.8x.h, [2],
                $Host To Device, HostCommand.8y=EDh.$Set LEDs,
                Ack.8x.i, Parameter.5y.i, Caps.1y.tf, Num.1y.tf,
                Scroll.1y.tf, Ack.8x.i
        Fields [1], $Device To Host, $Key Down, Scancode.8x.h, [2],
                $Host To Device, HostCommand.8y.h, Ack.8x.i,
                Parameter.8y.h, Ack.8x.i

        ; Device to Host
        Fields [1], $Device To Host, $Key Up, Release.8x=F0h.h,
                Scancode.Nx.h

        ; All other scancodes
        Fields [1], $Device To Host, $Key Down, Scancode.Nx.H

        ; Host to Device
        Fields [2], $Host To Device, Command.Ny.h
```



```
Layer: PS2BUS        |                | Key Down |          | SCANCODE
Time: 128.3008ms     | Device To Host | Key Down |          | 12

Layer: PS2BUS        |                |          |          | SCANCODE
Time: 224.1808ms     | Device To Host | Key Down |          | 23

Layer: PS2BUS        |                |          | RELEASE  | SCANCODE
Time: 268.711ms      | Device To Host | Key Up   | F0       | 12

Layer: PS2BUS        |                |          | RELEASE  | SCANCODE
Time: 283.6268ms     | Device To Host | Key Up   | F0       | 23

X1 Layer: PS2BUS     |                |          | SCANCODE |                | HOSTCOMMAND | CAPS  | NUM   | SCROLL
   Time: 298.4822ms  | Device To Host | Key Down | 58       | Host To Device | Set LEDs    | False | False | False

Layer: PS2BUS        |                |          | SCANCODE
Time: 315.2592ms     | Device To Host | Key Down | 2D

Layer: PS2BUS        |                |          | RELEASE  | SCANCODE
Time: 348.3302ms     | Device To Host | Key Up   | F0       | 58

Layer: PS2BUS        |                |          | RELEASE  | SCANCODE
Time: 383.206ms      | Device To Host | Key Up   | F0       | 2D

Layer: PS2BUS        |                |          | SCANCODE
Time: 439.3265ms     | Device To Host | Key Down | 23

Layer: PS2BUS        |                |          | SCANCODE
Time: 462.2197ms     | Device To Host | Key Down | 12

Layer: PS2BUS        |                |          | RELEASE  | SCANCODE
Time: 495.2905ms     | Device To Host | Key Up   | F0       | 23

Layer: PS2BUS        |                |          | SCANCODE |                | HOSTCOMMAND | CAPS  | NUM   | SCROLL
Time: 542.0885ms     | Device To Host | Key Down | 58       | Host To Device | Set LEDs    | True  | False | False

Layer: PS2BUS        |                |          | RELEASE  | SCANCODE
```

X2 to X1 = -44.1378ms

FILE SAVE, SAVE BETWEEN CURSORS, 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.

Choose the menu item File | Save Between Cursors to save the current configuration and sample data that is contained between the X1 and X2 cursors to a binary ULD file. Use this menu item to make smaller trace files that contain only the information that you are interested in. The minimum sample size for the Save is 200K samples. If the X1 and X2 contain less samples than 200K, the save will start at the first cursor and go for 200K samples.

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.

OUTPUT FILE FORMAT

The following is the Visual Basic source code that saves the ULD file format used by the Logic Analyzer/ Oscilloscope and Signal Generator application.

```
Write #1, "USBee DX Data File  " + Format(Date, "LONG DATE")

Write #1, "WaveHighlighted", WaveHighlighted

For x = 0 To 15
    Write #1, "BusType" & str(x), BusType(x)
    Write #1, "Bus" & str(x), Bus(x)
    Write #1, "ShowVal" & str(x), ShowVal(x)
    Write #1, "HexVal" & str(x), HexVal(x)
    Write #1, "Delimiter" & str(x), Delimiter(x)
    Write #1, "ShowAll" & str(x), ShowAll(x)
    Write #1, "BytesPerLine" & str(x), BytesPerLine(x)
    Write #1, "Channels" & str(x), Channels(x)
    Write #1, "ClockSignal" & str(x), ClockSignal(x)
    Write #1, "UseClock" & str(x), UseClock(x)
    Write #1, "ClockEdge" & str(x), ClockEdge(x)
    Write #1, "SerialChannel" & str(x), SerialChannel(x)
    Write #1, "AlignValue" & str(x), AlignValue(x)
    Write #1, "AlignEdge" & str(x), AlignEdge(x)
    Write #1, "AlignChannel" & str(x), AlignChannel(x)
    Write #1, "UseAlignChannel" & str(x), UseAlignChannel(x)
    Write #1, "ClockChannel" & str(x), ClockChannel(x)
    Write #1, "BitsPerValue" & str(x), BitsPerValue(x)
    Write #1, "msbfirst" & str(x), msbfirst(x)
    Write #1, "DPlusSignal" & str(x), DPlusSignal(x)
    Write #1, "DMinusSignal" & str(x), DMinusSignal(x)
    Write #1, "USBSpeed" & str(x), USBSpeed(x)
    Write #1, "USBAddr" & str(x), USBAddr(x)
    Write #1, "USBEndpoint" & str(x), USBEndpoint(x)
    Write #1, "SOF" & str(x), SOF(x)
    Write #1, "SDASignal" & str(x), SDASignal(x)
    Write #1, "SCLSignal" & str(x), SCLSignal(x)
    Write #1, "ShowAck" & str(x), ShowAck(x)
    Write #1, "SSsignal" & str(x), SSsignal(x)
    Write #1, "SCKsignal" & str(x), SCKsignal(x)
    Write #1, "MOSISignal" & str(x), MOSISignal(x)
    Write #1, "MISOSignal" & str(x), MISOSignal(x)
```

```
        Write #1, "MISOEdge" & str(x), MISOEdge(x)
        Write #1, "MOSIEdge" & str(x), MOSIEdge(x)
        Write #1, "SSOn" & str(x), SSOn(x)
        Write #1, "CanSignal" & str(x), CanSignal(x)
        Write #1, "BitRate" & str(x), BitRate(x)
        Write #1, "MinID" & str(x), MinID(x)
        Write #1, "MaxID" & str(x), MaxID(x)
        Write #1, "OneWireSignal" & str(x), OneWireSignal(x)
        Write #1, "I2SWordSelectSignal" & str(x), I2SWordSelectSignal(x)
        Write #1, "I2SClkSignal" & str(x), I2SClkSignal(x)
        Write #1, "I2SDataSignal" & str(x), I2SDataSignal(x)
        Write #1, "ClkSignal" & str(x), ClkSignal(x)
        Write #1, "DataSignal" & str(x), DataSignal(x)
        Write #1, "AsyncSignal" & str(x), AsyncSignal(x)
        Write #1, "BaudRate" & str(x), BaudRate(x)
        Write #1, "DataBits" & str(x), DataBits(x)
        Write #1, "Parity" & str(x), Parity(x)
        Write #1, "ASCII" & str(x), ASCII(x)
        Write #1, "PS2DataSignal" & str(x), PS2DataSignal(x)
        Write #1, "PS2ClockSignal" & str(x), PS2ClockSignal(x)
    Next x

    Write #1, "TCenterSample", TCenterSample
    Write #1, "Infinite", Infinite
    Write #1, "TimelineMode", TimelineMode
    Write #1, "OffsetValue", OffsetValue
    Write #1, "OffsetValue", OffsetValue
    Write #1, "TimePerDiv", TimePerDiv
    Write #1, "MaxNumberOfSamples", MaxNumberOfSamples
    Write #1, "ActualNumberOfSamples", ActualNumberOfSamples
    Write #1, "TimeFlag", TimeFlag
    Write #1, "Rate", Rate
    Write #1, "MaxRate", MaxRate
    Write #1, "Captured", Captured
    Write #1, "TRIGValidSetting", TRIGValidSetting
    Write #1, "CLKEdgeSetting", CLKEdgeSetting
    Write #1, "TriggerOffset", TriggerOffset
    Write #1, "KnobValue2", KnobValue2
    Write #1, "NumberOfSections", NumberOfSections
    Write #1, "ScopeVoltsPerDiv", ScopeVoltsPerDiv
    Write #1, "TCenterSample", TCenterSample
    Write #1, "ScreenMax", ScreenMax
    Write #1, "ScreenMin", ScreenMin
    Write #1, "Initialized", Initialized
    Write #1, "NumberOfSamples", NumberOfSamples

    For x = 0 To 255
        Write #1, "TBuffer" & str(x), TBuffer(x)
    Next x
    For x = 0 To 15
        For y = 0 To 3
            Write #1, "TriggerSetting" & str(x) & "-" & str(y), TriggerSetting(y, x)
            Write #1, "Trigg" & str(x) & "-" & str(y), Trigg(y, x)
        Next y
    Next x

    Write #1, "TriggerStates", TriggerStates
    Write #1, "ScaleP", ScaleP
    Write #1, "TOCursor", TOCursor
    Write #1, "TCurrentCursor", TCurrentCursor
    Write #1, "TXCursor", TXCursor
    Write #1, "TY1Cursor", TY1Cursor
    Write #1, "TY2Cursor", TY2Cursor
    Write #1, "TScale", TScale
    Write #1, "TSubScale", TSubScale
    Write #1, "TStartingSample", TStartingSample
    Write #1, "TCenterSample", TCenterSample
    Write #1, "CalibrationSlope", CalibrationSlope
    Write #1, "Scope1GroundCalibrationLevel", Scope1GroundCalibrationLevel
    Write #1, "Scope1DisplayCenterVolts", Scope1DisplayCenterVolts
    Write #1, "Scope1TriggerLevel", Scope1TriggerLevel
    Write #1, "Scope1TriggerSlope", Scope1TriggerSlope
    Write #1, "VoltsPerPixel", VoltsPerPixel
    Write #1, "NumberOfDiv", NumberOfDiv
    Write #1, "AnalogWaveIndex", AnalogWaveIndex
    Write #1, "DigitalHighOn", DigitalHighOn
    Write #1, "DigitalLowOn", DigitalLowOn
    Write #1, "AnalogHighOn", AnalogHighOn
    Write #1, "AnalogLowOn", AnalogLowOn
    For x = 0 To 16
        Write #1, "SigColor" & str(x), SigColor(x)
        Write #1, "SigBackColor" & str(x), SigBackColor(x)
        Write #1, "SigForeColor" & str(x), SigForeColor(x)
    Next x

    Write #1, "AnalogColor", AnalogColor
    Write #1, "XCursorsOn", XCursorsOn
    Write #1, "YCursorsOn", YCursorsOn

    For x = 0 To 15
        For y = 0 To 15
            Write #1, "SignalsInWave" & str(x) & "-" & str(y), SignalsInWave(y, x)
        Next y
    Next x

    Write #1, "GlobalCalValue", GlobalCalValue

    For x = 0 To 15
        Write #1, "SignalLabel" & str(x), Form1.SignalLabel(x).Caption
    Next x

    Write #1, "A0D8", Form1.A0D8.Checked
    Write #1, "A0D16", Form1.A0D16.Checked
    Write #1, "A1D0", Form1.A1D0.Checked
    Write #1, "A1D8", Form1.A1D8.Checked
    Write #1, "A1D16", Form1.A1D16.Checked
    Write #1, "A2D0", Form1.A2D0.Checked
    Write #1, "A2D8", Form1.A2D8.Checked
    Write #1, "A2D16", Form1.A2D16.Checked
    Write #1, "CH1V", Form1.CH1V.Value
    Write #1, "CH2V", Form1.CH2V.Value
    Write #1, "Ch1Offset", Form1.Ch1Offset.Value
    Write #1, "Ch2Offset", Form1.Ch2Offset.Value
    Write #1, "VScroll1", Form1.VScroll1.Value
    Write #1, "HScroll1", Form1.HScroll1.Value
    Write #1, "SizeList", Form1.SizeList.ListIndex
    Write #1, "RateList", Form1.RateList.ListIndex
    Write #1, "NormalMode", Form1.NormalMode.Value
    Write #1, "AutoMode", Form1.AutoMode.Value
    Write #1, "TriggerPositionScroll", Form1.TriggerPositionScroll.Value
    Write #1, "Persist", Form1.Persist.Value
    Write #1, "Vectors", Form1.Vectors.Value
    Write #1, "Wide", Form1.Wide.Value
    Write #1, "ScaleP", Form1.ScaleP.Text
    Write #1, "SubScale", Form1.SubScale.Text
    Write #1, "AnnotationAnalog", Form1.AnnotationAnalog.Text
    Write #1, "AnnotationDHigh", Form1.AnnotationDHigh.Text
    Write #1, "AnnotationDLow", Form1.AnnotationDLow.Text
    Write #1, "ShowAnn", Form1.ShowAnn.Checked
    Write #1, "AnWhite", Form1.AnWhite.Checked
    Write #1, "AnBlack", Form1.AnBlack.Checked
    Write #1, "CH1Units", CH1Units
    Write #1, "CH2Units", CH2Units
    Write #1, "CH1Frame", Form1.CH1Frame.Caption
    Write #1, "Frame3", Form1.Frame3.Caption
    Write #1, "CH1ScaleSlope", CH1ScaleSlope
    Write #1, "CH1ScaleOffset", CH1ScaleOffset
    Write #1, "CH2ScaleSlope", CH2ScaleSlope
    Write #1, "CH2ScaleOffset", CH2ScaleOffset

    For x = 0 To 100
        Write #1, "MarkerWave" & str(x), MarkerWave(x)
        Write #1, "MarkerPosition" & str(x), MarkerPosition(x)
        Write #1, "MarkerText" & str(x), MarkerText(x)
        Write #1, "MarkerDirection" & str(x), MarkerDirection(x)
        Write #1, "MarkerOn" & str(x), MarkerOn(x)
    Next x

    Write #1, "CH1Name", Form1.CH1Frame.Caption
    Write #1, "Frame3", Form1.Frame3.Caption
    Write #1, "ShowGrid", Form1.Grid.Checked

    For x = 0 To 15
        Write #1, "ProtocolOn" & str(x), ProtocolOn(x)
        Write #1, "ProtocolFile" & str(x), ProtocolFile(x)
    Next x

    ' The binary sample data follows this last record
    Write #1, "[Samples]"
```

After the “[Samples]” tag is the raw sample data. There are NumberOfSamples times 4 bytes in the data. Each sample is 4 bytes taken at the sample rate. The low 16 bits are the logic levels of each of the 16 digital channels. The high 2 bytes are the 8-bit ADC values for each of the two analog channels.

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
…
```

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

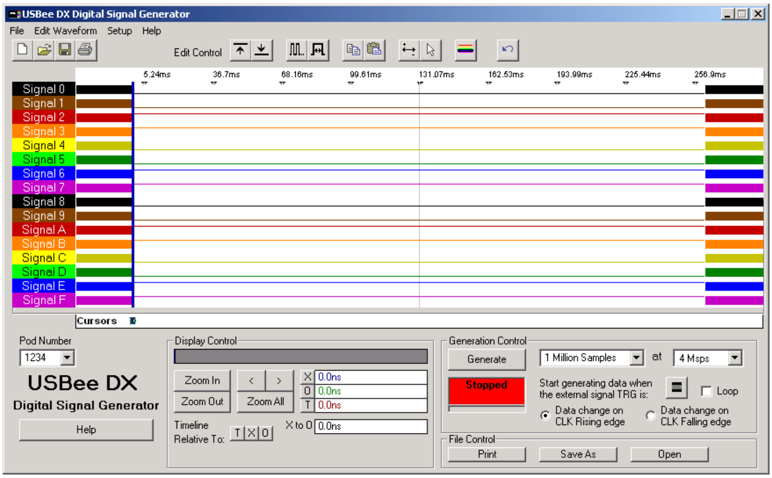
When you are ready to calibrate the USBee DX Pod, go to the menu item Setup | Calibrate. You will be asked to confirm that you really want to do the calibration. If so, press Yes, otherwise press No. Then follow these steps:

- 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.

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.



Pod Number: 1234

X 0.0ns

0 0.0ns

T 0.0ns

X to 0 0.0ns

1 Million Samples at 4 Msps

□ Loop

● Data change on CLK Rising edge ○ Data change on CLK Falling edge

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.

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

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.

FEATURES

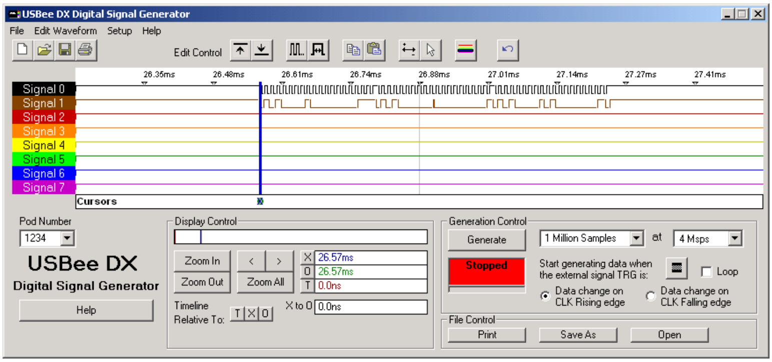
POD STATUS

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.

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.



Pod Number: 1234

X: 26.57ms
O: 26.57ms
T: 0.0ns
X to O: 0.0ns

1 Million Samples at 4 Msps

Start generating data when the external signal TRG is:

□ Loop

● Data change on CLK Rising edge
○ Data change on CLK Falling edge

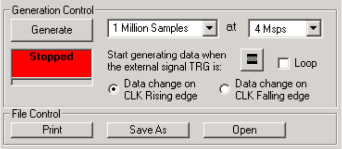
The maximum sample rate that your system can achieve varies depending on the number of channels you select.

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.

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.



1 Million Samples at 4 Msps

Start generating data when the external signal TRG is:

□ Loop

● Data change on CLK Rising edge

○ Data change on CLK Falling edge

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.

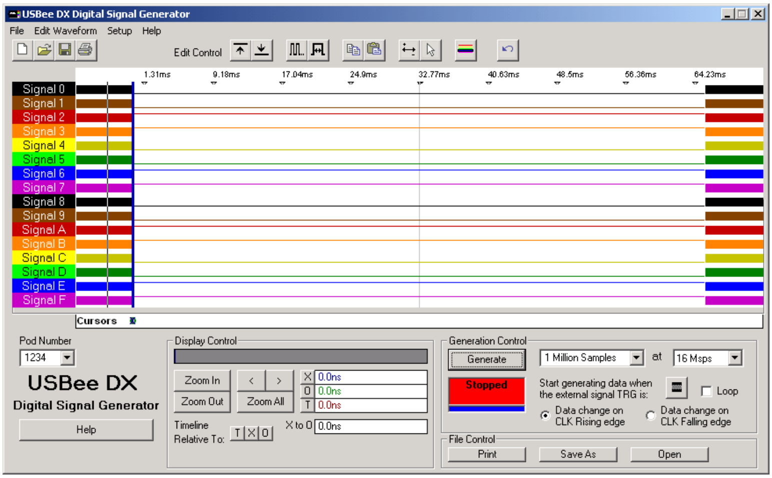
While the pod is generating the waveform on the pod signals, the CLK line is an output and toggles once for each of the samples provided. You can specify the **CLK Edge** that the output data changes on using the two radio buttons above.

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.

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.



Pod Number: 1234

X: 0.0ns
O: 0.0ns
T: 0.0ns
X to O: 0.0ns

1 Million Samples at 16 Msps

☐ Loop

● Data change on CLK Rising edge ○ Data change on CLK Falling edge

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.

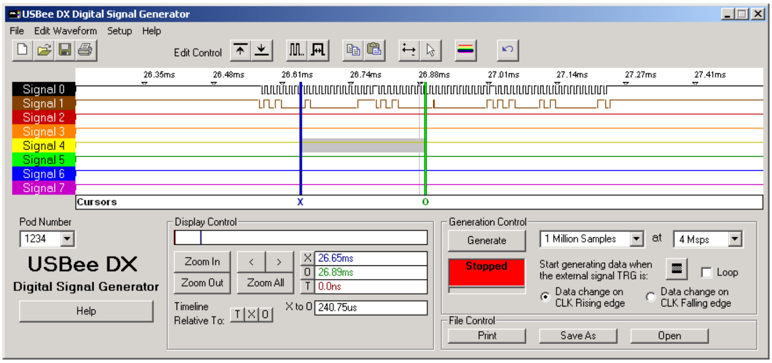
To change the zoom ratio for the time, click the **Zoom In** or **Zoom Out** buttons. You can also zoom in and out in time by clicking on the waveform. To zoom in, click the left mouse on the waveform window. To zoom out in time, click the right mouse button on the waveform window.

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.

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.



Pod Number: 1234

X 26.65ms
O 26.89ms
T 0.0ns
X to O 240.75us

1 Million Samples at 4 Msps

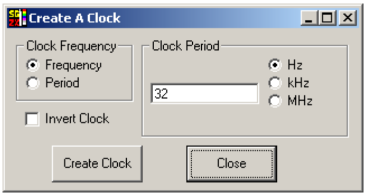
Start generating data when the external signal TRG is: ☐ Loop

● Data change on CLK Rising edge ○ Data change on CLK Falling edge

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.

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.



Clock Frequency
● Frequency
○ Period

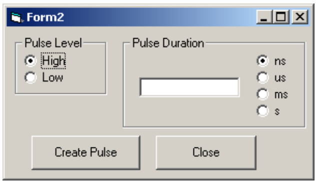
☐ Invert Clock

Clock Period: 32
● Hz
○ kHz
○ MHz

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.

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.



Form2

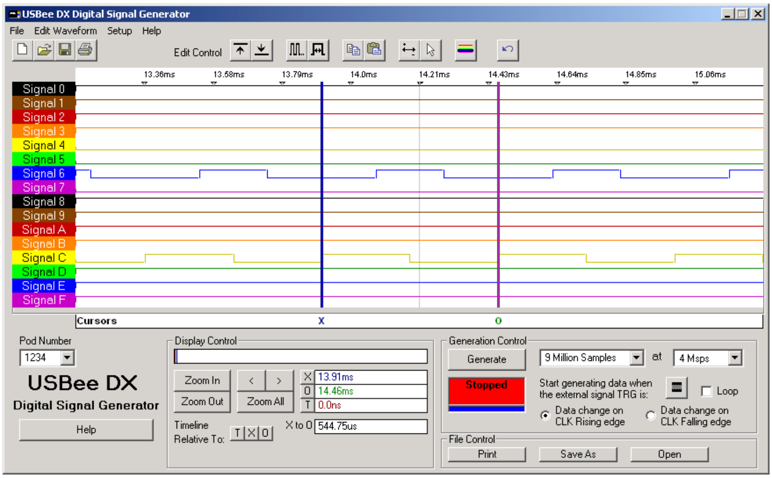
Pulse Level: ● High ○ Low

Pulse Duration: ● ns ○ us ○ ms ○ s

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.

MEASUREMENTS AND CURSORS

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



USBee DX Digital Signal Generator

Pod Number: 1234

Display Control: X 13.91ms, O 14.46ms, T 0.0ns, X to O 544.75us

Generation Control: 9 Million Samples at 4 Msps; Stopped

Start generating data when the external signal TRG is: ● Data change on CLK Rising edge ○ Data change on CLK Falling edge; □ Loop

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

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.

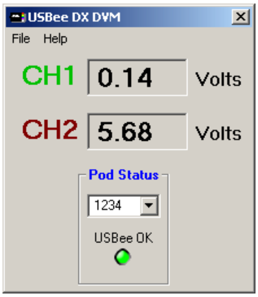
To load a previously saved waveform and display it, choose File | Open and specify the filename to load. This waveform will then be displayed as it was saved. If the loaded file is smaller than the current buffer size, the file will be loaded at the beginning of the current buffer. The ending samples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

PRINTING

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

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.



CH1 0.14 Volts

CH2 5.68 Volts

1234

USBee OK

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

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.

FEATURES

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.

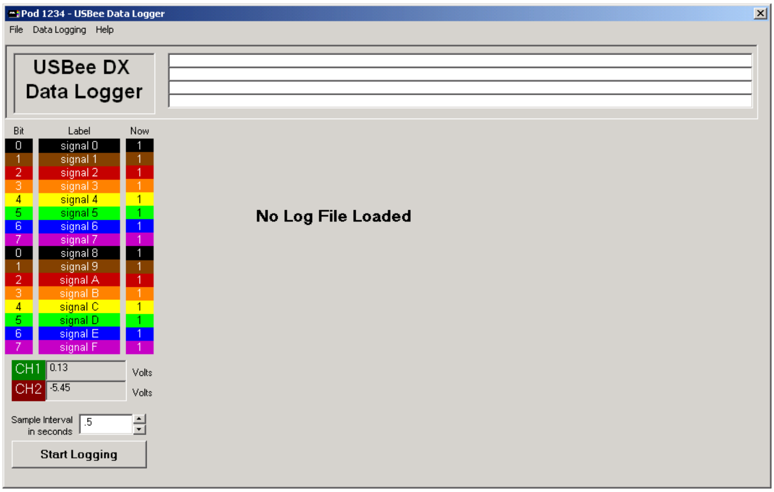
If you run the software with no pod attached, it will run in demonstration mode and simulate data so that you can still see how the software functions.

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.

DATA LOGGER

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



Bit	Label	Now
0	signal 0	1
1	signal 1	1
2	signal 2	1
3	signal 3	1
4	signal 4	1
5	signal 5	1
6	signal 6	1
7	signal 7	1
0	signal 8	1
1	signal 9	1
2	signal A	1
3	signal B	1
4	signal C	1
5	signal D	1
6	signal E	1
7	signal F	1

CH1 0.13 Volts

CH2 -5.45 Volts

Sample Interval in seconds .5

No Log File Loaded

DATA LOGGER SPECIFICATIONS

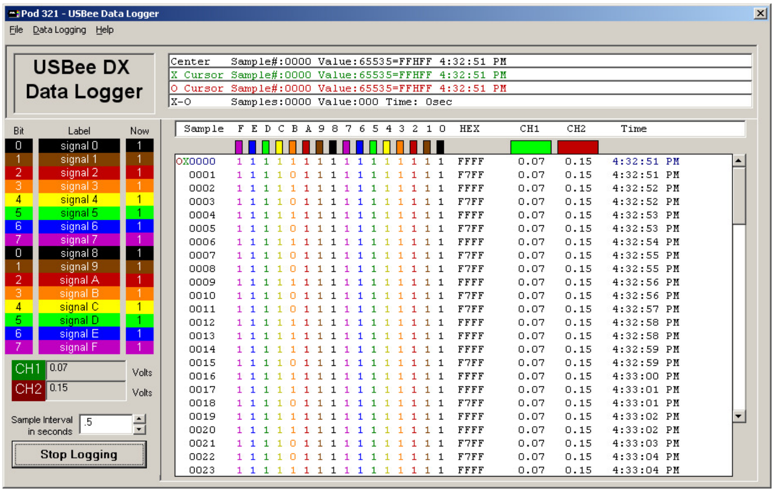
Digital Channels Logged	16
Analog Channels Logged	2
Sample Rates	500ms to 300sec

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.



```
Center     Sample#:0000 Value:65535=FFHFF 4:32:51 PM
X Cursor Sample#:0000 Value:65535=FFHFF 4:32:51 PM
O Cursor Sample#:0000 Value:65535=FFHFF 4:32:51 PM
X-O        Samples:0000 Value:000 Time: 0sec
```

Bit	Label	Now
0	signal 0	1
1	signal 1	1
2	signal 2	1
3	signal 3	1
4	signal 4	1
5	signal 5	1
6	signal 6	1
7	signal 7	1
0	signal 8	1
1	signal 9	1
2	signal A	1
3	signal B	1
4	signal C	1
5	signal D	1
6	signal E	1
7	signal F	1

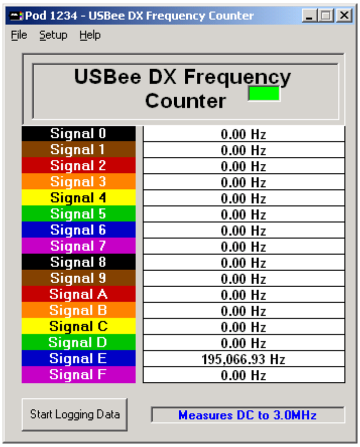
CH1 0.07 Volts
CH2 0.15 Volts

Sample Interval in seconds .5

```
 Sample  F E D C B A 9 8 7 6 5 4 3 2 1 0  HEX      CH1    CH2      Time
OX0000   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:51 PM
  0001   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:51 PM
  0002   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:52 PM
  0003   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:52 PM
  0004   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:53 PM
  0005   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:53 PM
  0006   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:54 PM
  0007   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:55 PM
  0008   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:55 PM
  0009   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:56 PM
  0010   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:56 PM
  0011   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:57 PM
  0012   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:58 PM
  0013   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:58 PM
  0014   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:32:59 PM
  0015   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:32:59 PM
  0016   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:33:00 PM
  0017   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:33:01 PM
  0018   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:33:01 PM
  0019   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:33:02 PM
  0020   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:33:02 PM
  0021   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:33:03 PM
  0022   1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1  F7FF     0.07   0.15   4:33:04 PM
  0023   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  FFFF     0.07   0.15   4:33:04 PM
```

FREQUENCY COUNTER

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


Pod 1234 - USBee DX Frequency Counter
File
Setup
Help
USBee DX Frequency Counter
Start Logging Data
Measures DC to 3.0MHz

Signal 0	0.00 Hz
Signal 1	0.00 Hz
Signal 2	0.00 Hz
Signal 3	0.00 Hz
Signal 4	0.00 Hz
Signal 5	0.00 Hz
Signal 6	0.00 Hz
Signal 7	0.00 Hz
Signal 8	0.00 Hz
Signal 9	0.00 Hz
Signal A	0.00 Hz
Signal B	0.00 Hz
Signal C	0.00 Hz
Signal D	0.00 Hz
Signal E	195,066.93 Hz
Signal F	0.00 Hz

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

QUICK START

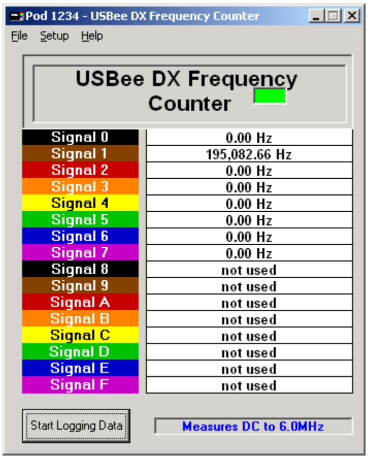
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to measure the frequency of a digital signal.

- 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.

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.

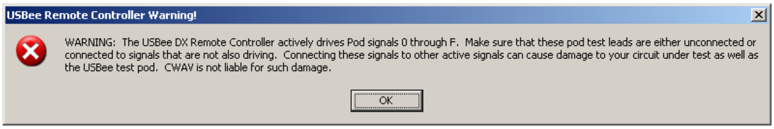
Change setup modes by clicking the menu item Setup and selecting the desired number of channels. Below shows the 8 channel setup mode.


Pod 1234 - USBee DX Frequency Counter
File
Setup
Help
USBee DX Frequency Counter
Start Logging Data
Measures DC to 6.0MHz

Signal 0	0.00 Hz
Signal 1	195,082.66 Hz
Signal 2	0.00 Hz
Signal 3	0.00 Hz
Signal 4	0.00 Hz
Signal 5	0.00 Hz
Signal 6	0.00 Hz
Signal 7	0.00 Hz
Signal 8	not used
Signal 9	not used
Signal A	not used
Signal B	not used
Signal C	not used
Signal D	not used
Signal E	not used
Signal F	not used

REMOTE CONTROLLER

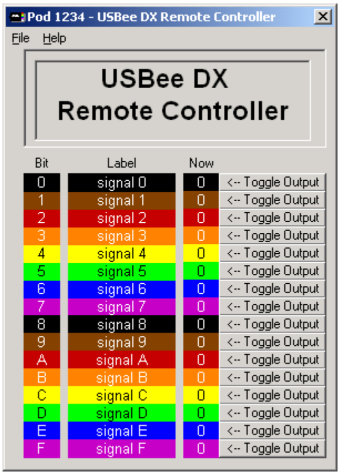
This section details the operation of the Remote Controller application that comes with the USBee DX. The Remote Controller application is a simple way to control the output settings for all of the 16 digital lines on the USBee DX. Since this application drives the digital signals, you will see a warning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



USBee Remote Controller Warning!

WARNING: The USBee DX Remote Controller actively drives Pod signals 0 through F. 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 your circuit under test as well as the USBee test pod. CWAV is not liable for such damage.

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



Pod 1234 - USBee DX Remote Controller

USBee DX
Remote Controller

Bit	Label	Now	
0	signal 0	0	<-- Toggle Output
1	signal 1	0	<-- Toggle Output
2	signal 2	0	<-- Toggle Output
3	signal 3	0	<-- Toggle Output
4	signal 4	0	<-- Toggle Output
5	signal 5	0	<-- Toggle Output
6	signal 6	0	<-- Toggle Output
7	signal 7	0	<-- Toggle Output
8	signal 8	0	<-- Toggle Output
9	signal 9	0	<-- Toggle Output
A	signal A	0	<-- Toggle Output
B	signal B	0	<-- Toggle Output
C	signal C	0	<-- Toggle Output
D	signal D	0	<-- Toggle Output
E	signal E	0	<-- Toggle Output
F	signal F	0	<-- Toggle Output

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

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

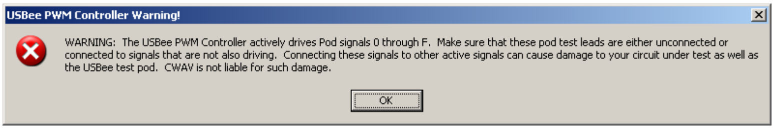
QUICK START

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)..

PWM CONTROLLER

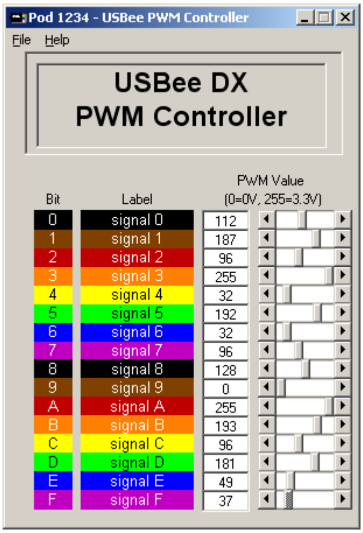
This section details the operation of the Pulse Width Modulator application that comes with the USBee DX. The Pulse Width Modulator application creates a Pulse Width Modulated output for all of the 16 digital lines on the USBee DX. Since this application drives the digital signals, you will see a warning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



USBee PWM Controller Warning!

WARNING: The USBee PWM Controller actively drives Pod signals 0 through F. 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 your circuit under test as well as the USBee test pod. CWAV is not liable for such damage.

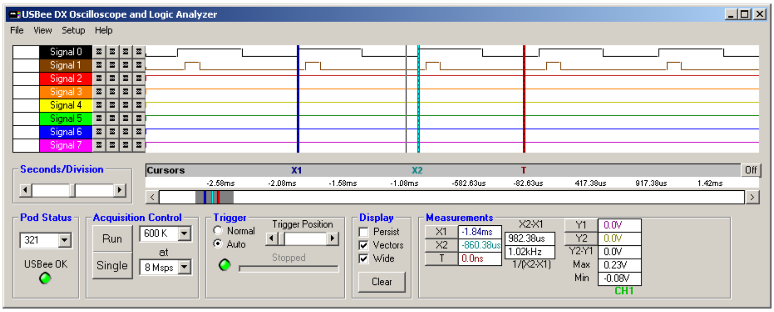
Click OK to enter the application. Below you see the application screen.



Bit	Label	PWM Value (0=0V, 255=3.3V)
0	signal 0	112
1	signal 1	187
2	signal 2	96
3	signal 3	255
4	signal 4	32
5	signal 5	192
6	signal 6	32
7	signal 7	96
8	signal 8	128
9	signal 9	0
A	signal A	255
B	signal B	193
C	signal C	96
D	signal D	181
E	signal E	49
F	signal F	37

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).

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

QUICK START

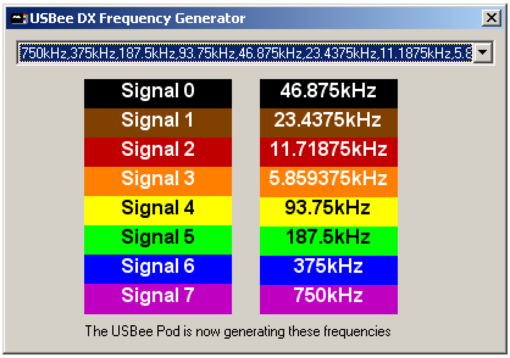
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to create 16 PWM signals.

- 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.

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.



750kHz,375kHz,187.5kHz,93.75kHz,46.875kHz,23.4375kHz,11.1875kHz,5.8

Signal 0	46.875kHz
Signal 1	23.4375kHz
Signal 2	11.71875kHz
Signal 3	5.859375kHz
Signal 4	93.75kHz
Signal 5	187.5kHz
Signal 6	375kHz
Signal 7	750kHz

The USBee Pod is now generating these frequencies

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.

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

QUICK START

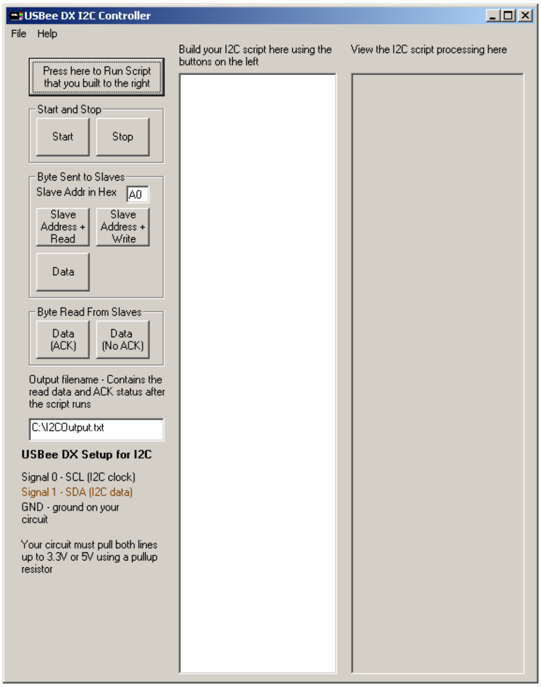
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to generate one of the fixed sets of frequencies on the digital lines.

- 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.

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.



Output filename - Contains the read data and ACK status after the script runs

C:\I2COutput.txt

USBee DX Setup for I2C

Signal 0 - SCL (I2C clock)
Signal 1 - SDA (I2C data)
GND - ground on your circuit

Your circuit must pull both lines up to 3.3V or 5V using a pullup resistor

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
```

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)

QUICK START

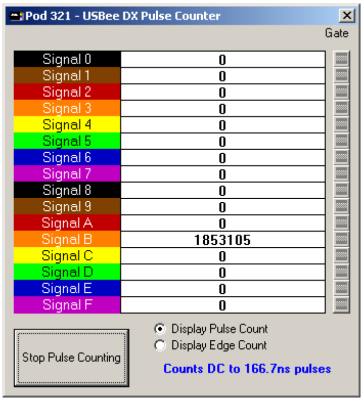
In order to quickly get up and running using this application, here is a step by step list of the things you need to do to generate I2C transactions.

- 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..

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.



Pod 321 - USBee DX Pulse Counter

		Gate
Signal 0	0	
Signal 1	0	
Signal 2	0	
Signal 3	0	
Signal 4	0	
Signal 5	0	
Signal 6	0	
Signal 7	0	
Signal 8	0	
Signal 9	0	
Signal A	0	
Signal B	1853105	
Signal C	0	
Signal D	0	
Signal E	0	
Signal F	0	

● Display Pulse Count
○ Display Edge Count

Counts DC to 166.7ns pulses

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

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

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.

USBEE TOOLBUILDER

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

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.

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.

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.

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**	**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

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.

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 (R/W#/TRG)	Read/Write# Output (bit-bang mode),TRG (Signal Generator Mode)
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.

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.

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.

USBEE DX TOOLBUILDER FUNCTIONS

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

INITIALIZING THE USBEE DX POD

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

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

BIT BANG-MODES

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

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

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)

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.

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

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

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

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

STOPCAPTURE

This routine terminates a pending capture.

Calling Convention

```
int StopCapture(void)
```

Return Value:

- 1 = Capture Stopped
- 0 = Stop Failed

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

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, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString)
```

- 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
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString)
```

- 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.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString)
```

- 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 delemeter, 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
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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,
char *ProtocolDefinitionFilename, char *ProtocolOutputFilename,
char *ErrorString)
```

- 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
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.

- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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

```
CWAV_EXPORT 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,
char *ProtocolDefinitionFilename, char *ProtocolOutputFilename,
char *ErrorString)
```

- 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
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.

- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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, char *ProtocolDefinitionFilename, char
*ProtocolOutputFilename, char *ErrorString)
```

- 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.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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,
char *ProtocolDefinitionFilename, char *ProtocolOutputFilename,
char *ErrorString)
```

- 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.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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,
char *ProtocolDefinitionFilename, char *ProtocolOutputFilename,
char *ErrorString)
```

- 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.
- ProtocolDefinitionFilename – filename for the Protocol Definition File to use to create a PacketPresenter file. If this value is 0 then the PacketPresenter feature is turned off.
- ProtocolOutputFilename – filename that is created for the output of the PacketPresenter.
- ErrorString – string that holds an error description of the routine returns an error.

Return Value:

- TRUE – No Error during processing
- FALSE – Error while processing. The ErrorString contains a description of the error to present to the user.

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);
```

DIGITAL SIGNAL GENERATOR FUNCTION

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

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

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

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

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

DIGITAL VOLTMETER (DVM) FUNCTION

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

GETANALOGAVERAGECOUNT

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

Calling Convention

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

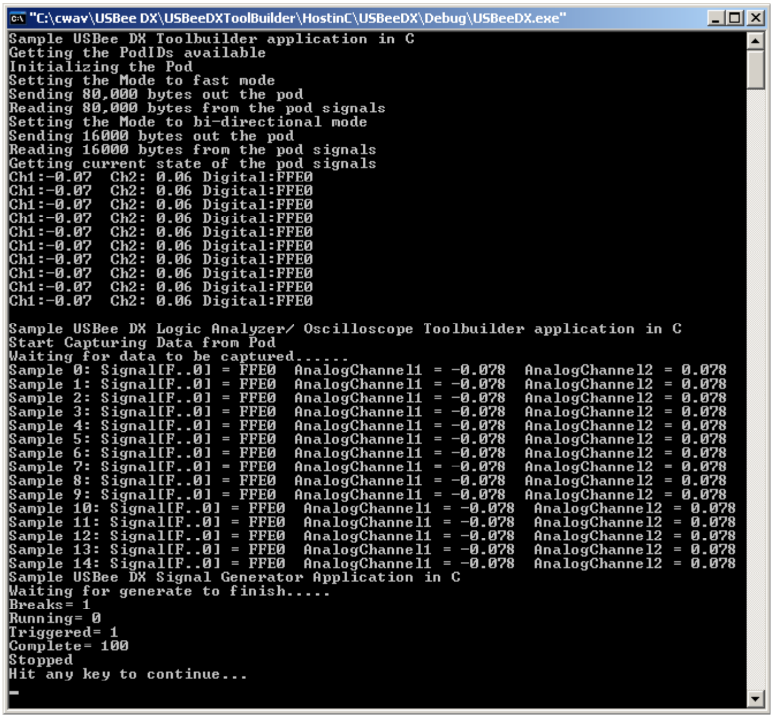
`*ch1` and `*ch2` will be filled with the analog average voltage for that channel. The value returned is 100 times the actual value so you need to divide this by 100 to get the measured value in volts.

`*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

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.


"C:\cwav\USBee DX\USBeeDXToolBuilder\HostinC\USBeeDX\Debug\USBeeDX.exe"

```
Sample USBee DX Toolbuilder application in C
Getting the PodIDs available
Initializing the Pod
Setting the Mode to fast mode
Sending 80,000 bytes out the pod
Reading 80,000 bytes from the pod signals
Setting the Mode to bi-directional mode
Sending 16000 bytes out the pod
Reading 16000 bytes from the pod signals
Getting current state of the pod signals
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0
Ch1:-0.07  Ch2: 0.06 Digital:FFE0

Sample USBee DX Logic Analyzer/ Oscilloscope Toolbuilder application in C
Start Capturing Data from Pod
Waiting for data to be captured......
Sample 0: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 1: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 2: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 3: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 4: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 5: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 6: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 7: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 8: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 9: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 10: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 11: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 12: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 13: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample 14: Signal[F..0] = FFE0  AnalogChannel1 = -0.078  AnalogChannel2 = 0.078
Sample USBee DX Signal Generator Application in C
Waiting for generate to finish.....
Breaks= 1
Running= 0
Triggered= 1
Complete= 100
Stopped
Hit any key to continue...
_
```

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 2008, 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);
```

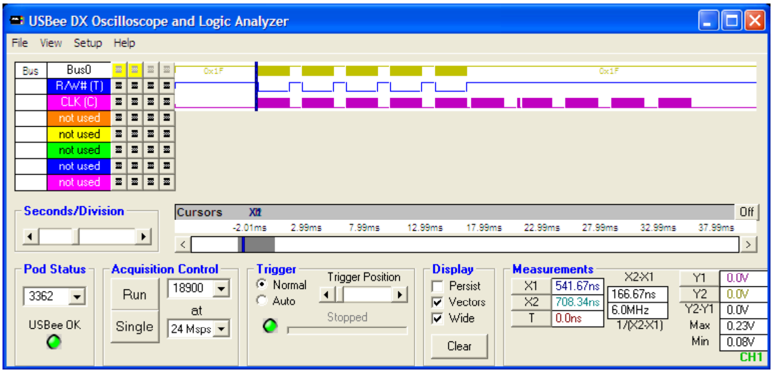
```
CWAV_IMPORT int CWAV_API StopCapture(void);       // End a Logic Analyzer trace
CWAV_IMPORT int CWAV_API CaptureStatus( char *breaks, char *running, char *triggered,
                            long *start, long *end, long *trigger, char *full );

// 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 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 to address your pod.

     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;
}
```

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.



Pod Status: 3362

USBee OK

Acquisition Control: 18900 at 24 Msps

Trigger: ● Normal ○ Auto

Stopped

Display: □ Persist ☑ Vectors ☑ Wide

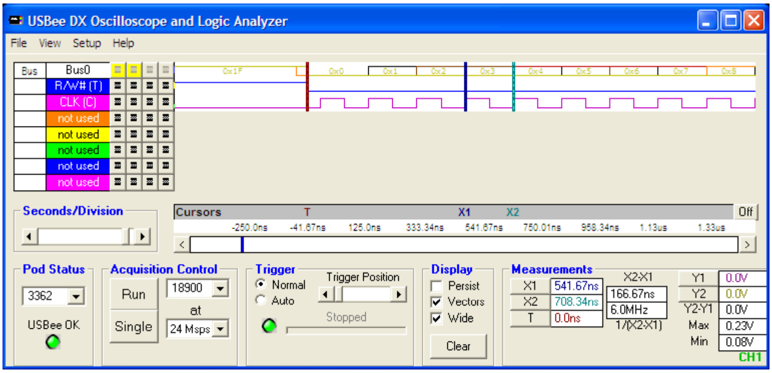
X1	541.67ns	X2-X1		Y1	0.0V
X2	708.34ns	166.67ns		Y2	0.0V
T	0.0ns	6.0MHz		Y2-Y1	0.0V
		1/(X2-X1)		Max	0.23V
				Min	0.08V

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.



Pod Status: 3362, USBee OK

Acquisition Control: 18900 at 24 Msps

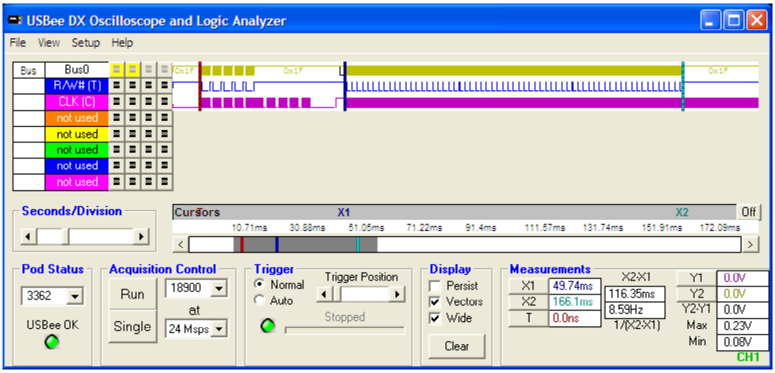
Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

X1 541.67ns, X2 708.34ns, T 0.0ns; X2-X1 166.67ns, 6.0MHz 1/(X2-X1)

Y1 0.0V, Y2 0.0V, Y2-Y1 0.0V, Max 0.23V, Min 0.08V

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.



Pod Status: 3362, USBee OK

Acquisition Control: 18900 at 24 Msps

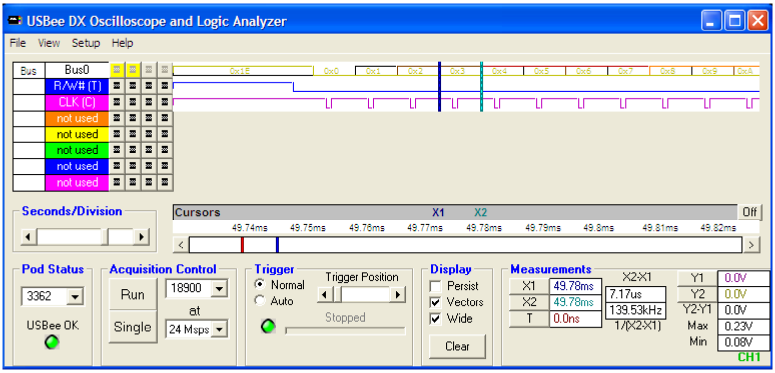
Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

X1 49.74ms, X2 166.1ms, T 0.0ns; X2-X1 116.35ms, 8.59Hz 1/(X2-X1)

Y1 0.0V, Y2 0.0V, Y2-Y1 0.0V, Max 0.23V, Min 0.08V

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



Pod Status: 3362, USBee OK

Acquisition Control: 18900 at 24 Msps

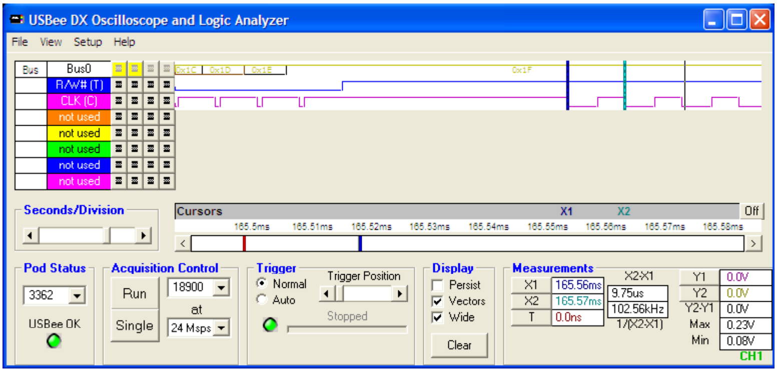
Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

X1 49.78ms, X2 49.78ms, T 0.0ns; X2-X1 7.17us, 139.53kHz 1/(X2-X1)

Y1 0.0V, Y2 0.0V, Y2-Y1 0.0V, Max 0.23V, Min 0.08V

Bi-Directional mode SetSignal byte timing



Pod Status: 3362, USBee OK

Acquisition Control: 18900 at 24 Msps

Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

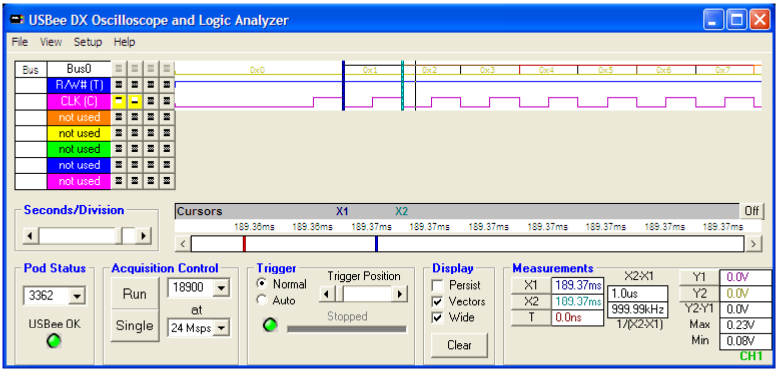
		X2-X1		
X1	165.56ms	9.75us	Y1	0.0V
X2	165.57ms	102.56kHz	Y2	0.0V
T	0.0ns	1/(X2-X1)	Y2-Y1	0.0V
			Max	0.23V
			Min	0.08V

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.



Pod Status: 3362, USBee OK

Acquisition Control: 18900 at 24 Msps

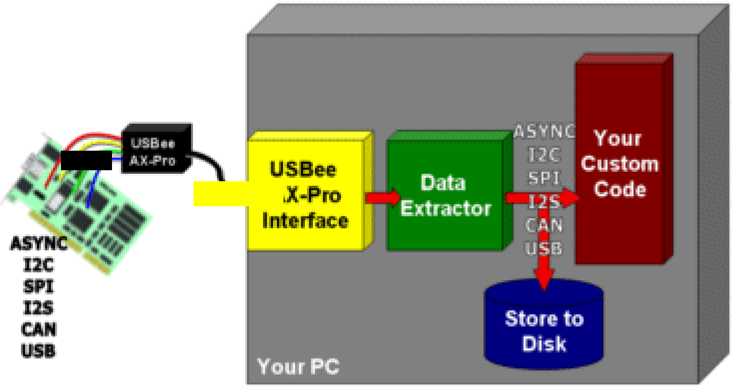
Trigger: ● Normal ○ Auto

Display: □ Persist ☑ Vectors ☑ Wide

		X2-X1		
X1	189.37ms	1.0us	Y1	0.0V
X2	189.37ms	999.99kHz	Y2	0.0V
T	0.0ns	1/(X2-X1)	Y2-Y1	0.0V
			Max	0.23V
			Min	0.08V

USBEE DX DATA EXTRACTOR OVERVIEW

The Data Extractors are an option software product for use with the USBee DX Test Pod that allows engineers to extract the raw data from various embedded busses to store off to disk or stream to another application. The Data Extractors will collect the raw data from Parallel, Serial, I2C, I2S, Async, USB Full and Low Speed, SMBus, 1-Wire or CAN busses and store the data to disk or pass it to your own processing application in real-time.



DATA EXTRACTOR FEATURES

- Uses the USBee DX pod to stream data from your embedded design into your PC
- Captures continuous real-time bus data
- Extracts the transaction data on the fly
- Stores data to disk or process it in real-time
- Runs indefinitely
- Captures entire test sequences
- Monitors embedded system data flows during normal operation
- Processes or stores Megabytes, Gigabytes or Terabytes of data
- Runs as a Windows Command Line executable from the Command Prompt and can be executed from Batch files containing the desired parameters
- Special Viewer to view and search through the extracted data files quickly
- Lets you write your own software to further process the extracted data using the Extractor API libraries.

BUS TYPES DECODED

- **Parallel** (internal or external clocking up to 12MHz)
- **Serial** (internal or external clocking up to 12MHz)
- **Async** (up to 12Mbaud)
- **I2C** (SCL up to 4MHz)
- **SPI** (SPI Clock up to 12MHz)
- **1-Wire** (Standard 1-Wire bit rates)
- **I2S** (bit clock up to 12MHz)
- **USB** (Low 1.5Mbps and Full Speed 12Mbps USB)
- **CAN** (up to 12Mbps)
- **SM Bus** (SM Clock up to 12MHz)

YOUR TESTING SYSTEM

The typical challenge in embedded streaming bus systems is to get the data out of your embedded system quickly and easily so that you can process it, either to capture a bug in progress or to evaluate performance. In any case, this can be done with the USBee DX Data Extractor System.

The USBee DX pod is used to stream raw sample data from its 8 digital input lines directly into the PC. The Data Extractor software modules then take that streaming data and extract your desired data out of the raw stream using the extractor processing threads. Our sample command line application, as well as any custom application you write, interfaces to the extractor through a simple Windows DLL consisting of five function calls. These calls are used to start and extraction, stop an extraction, gather the data (and how much data) and check for error status.

SYSTEM REQUIREMENTS

- The USBee DX Data Extractors require the following PC configuration:
- Windows® Vista, XP or Windows® 2000 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

It is HIGHLY recommended that the USBee DX and Data Extractors be run together on a separate PC than the PC controlling the system under test. If your PC is also controlling the system under test you may not be able to get the maximum sample rates needed for some of the extractors.

After installing the software as below, you can determine the maximum sample rate your system can achieve by plugging in the USBee DX, run the Logic Analyzer Application and choosing the Setup, Sample Rate Test menu option. The sample rate test may take up to 20 seconds. Once the sample rate test is complete, the Sample Rate drop down box will be filled with the available sample rates for you machine. The highest sample rate is what your PC can achieve.

To get the highest sample rates, you will want to use a Desktop PC with native USB 2.0 ports on the motherboard. Some modern Laptops can achieve the maximum of 24Msps, but you will want to disable all power saving features and run your laptop from the power supply, not the batteries.

SYSTEM SETUP

To configure a system to run these extractors you need the following:

- USBee DX Software Installed (follow instructions on the CD)
- USBee Data Extractors Software Installed (follow instructions on the CD)
- V File Viewer
- USBee DX Pod plugged into a USB 2.0 port on your PC.

INSTALLING THE USBEE DX CD

Do not plug in the USBee DX until after you install the USBee DX CD. Place the USBee DX CD in the drive and run the setup.exe. This will install all of the drivers and application programs in the proper directories. Choose the default settings for all installation screens.

INSTALLING THE USBEE DX DATA EXTRACTOR CD

Place the USBee DX Data Extractor CD in the drive and run the setup.exe. This will install all of the drivers and application programs in the proper directories. Choose the default settings for all installation screens.

INSTALLING THE V FILE VIEWER

The files that are created by the Data Extractor can be very large and require a special file viewer that can handle enormous files quickly and easily, both in ASCII text and binary Hexadecimal formats. With the Data Extractor comes an installation for the V File Viewer which efficiently views huge data files and allows for quick searching through your data to find the events you are looking for.

To install the V File Viewer, you can either run the v72.exe file from the Data Extractor CD or you can download it. To download the V File Viewer, go to http://www.fileviewer.com/Download.html and download the v72.exe file. This is a self-installing program that installs the V File Viewer.

For help on using the V File Viewer, please refer to the Help included with the viewer.

RUNNING THE COMMAND LINE EXTRACTORS

Once these components are installed correctly you can run the Extractor command prompt application .exe files. Each of the executables requires a series of command line parameters that tell the extractor how to process the bus data.

You will need to have full security access for the folders that you are running the applications from since they write to these directories for output data. If you do not have access, you will need to either move them or grant yourself access to those directories using the Window Security Settings.

To run the programs, you can do one of two options:

Open a Windows Command Prompt Window, change directory (cd) to your \ProgramFiles\USBeeDXDataExtractors, and enter the command line including all desired parameters.

or

Edit the batch files (goUSB.bat, goI2C.bat. etc.) to include the parameters you desire. You can then simply click on the Start Menu items ("Run I2C Batch File etc.) or double click on the batch files themselves in the Windows Explorer.

For all of the extractors you will need to use the USBee Pod ID on your Pod (on the back of the unit) as a command line parameter.

BUILDING YOUR OWN PROGRAMS USING THE API

You can also start to build your own processing programs using the source code for the command prompt applications as a reference point. Each Extractor has a sample project (Visual Studio C++ 6.0) in the \Program Files\USBee DX Data Extractors directory for you to start with.

In order for your programs to run, you must have installed both the USBee DX CD and the Data Extractors CD on that same machine.

ASYNC DATA EXTRACTOR

The Async Bus Data Extractor takes the real-time streaming data from up to 8 embedded asynchronous buses (UART), formats it and allows you to save the data to disk or process it as it arrives.

The DX Streaming Data Extractors are optional software modules for use with the USBee DX Test Pod (required) which must be purchased separately.

ASYNC BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- 8 digital channels
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Baud Rates from 1200 baud to 12 Mbaud *
- Data Bit Settings (5, 6, 7 or 8)
- Parity Bit Settings (Mark, Space, Odd, Even, Ignore, None)
- Time Stamps of start of bytes or packets
- Output to Text File (Hex, Decimal, Binary or ASCII)*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signal lines to an Async data bus. Connect the GND line to the digital ground of your system.

EXTRACTOR COMMAND LINE PROGRAM

The Async Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

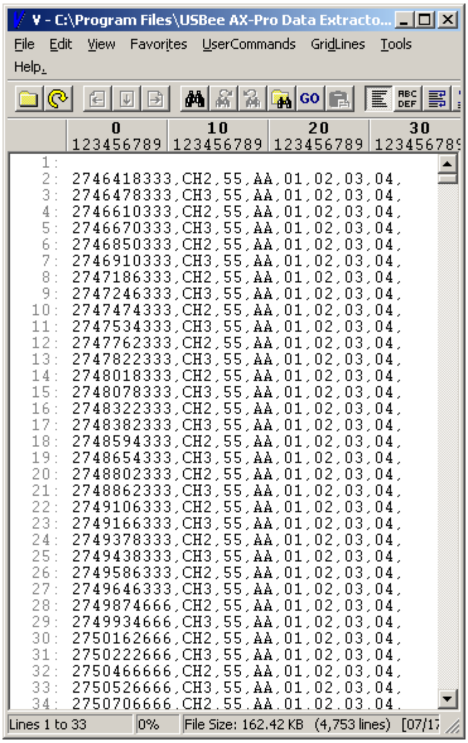
To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- ("cd \program files\USBee Data Extractor\Async")
- Run the executable using the following command line arguments:

```
AsyncExtractor [-?SADHBICGNX] [-R BaudRate] [-E DataBits] [-L
Parity] [-M SignalMask] [-Q NumberOfBytes] [-T BytesPesLine] [-V
Timestamp] [-O filename] -P PodID

    ?  - Display this help screen
    P  - Pod ID (required)
    O  - Output to filename (default off)
    S  - Output to the screen (default off)
    Q  - Number of output values (default = until keypress)
    R  - Baud Rate (9600 baud default)
    E  - Number of Data Bits (5,6,7,8-default)
    L  - Parity Type (0=none(default), 1=mark, 2=space, 3=even,
4=odd)
    M  - Which Signals to capture (1=signal0, 128=signal7, 255=all,
0=none (default))
    A  - ASCII Text Values ("1")
    D  - Decimal Text Values ("49")
    H  - Hex Text Values ("31") default
    B  - Binary Text Values ("00110001")
    I  - Binary Values (49)
    C  - Comma Delimited
    G  - Space Delimited (default)
    N  - Newline Delimited
    X  - No Delimeter
    T  - Force Bytes Per Line (no force default)
    V  - Timestamps (0=off, 1=each byte, 2=each channel start)
```

EXAMPLE OUTPUT FILES

```
AsyncExtractor -O output.dex -P 3209 -C -Q 100000 -R 1000000 -E 8 -L
0 -M 255 -H -V 2
```

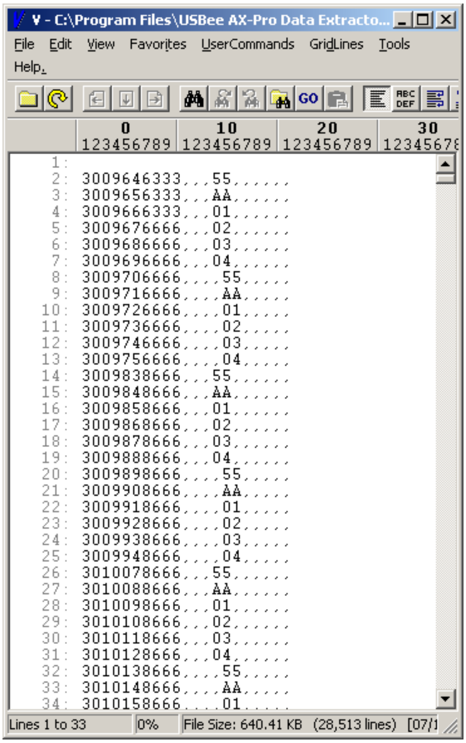


V - C:\Program Files\USBee AX-Pro Data Extracto...
File Edit View Favorites UserCommands GridLines Tools
Help.
GO
RBC DEF
0 10 20 30
123456789 123456789 123456789 12345678

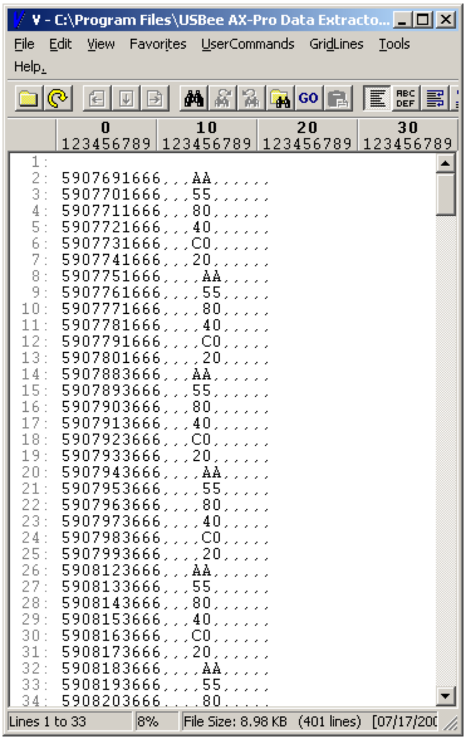
```

2746418333,CH2,55,AA,01,02,03,04,
2746478333,CH3,55,AA,01,02,03,04,
2746610333,CH2,55,AA,01,02,03,04,
2746670333,CH3,55,AA,01,02,03,04,
2746850333,CH2,55,AA,01,02,03,04,
2746910333,CH3,55,AA,01,02,03,04,
2747186333,CH2,55,AA,01,02,03,04,
2747246333,CH3,55,AA,01,02,03,04,
2747474333,CH2,55,AA,01,02,03,04,
2747534333,CH3,55,AA,01,02,03,04,
2747762333,CH2,55,AA,01,02,03,04,
2747822333,CH3,55,AA,01,02,03,04,
2748018333,CH2,55,AA,01,02,03,04,
2748078333,CH3,55,AA,01,02,03,04,
2748322333,CH2,55,AA,01,02,03,04,
2748382333,CH3,55,AA,01,02,03,04,
2748594333,CH2,55,AA,01,02,03,04,
2748654333,CH3,55,AA,01,02,03,04,
2748802333,CH2,55,AA,01,02,03,04,
2748862333,CH3,55,AA,01,02,03,04,
2749106333,CH2,55,AA,01,02,03,04,
2749166333,CH3,55,AA,01,02,03,04,
2749378333,CH2,55,AA,01,02,03,04,
2749438333,CH3,55,AA,01,02,03,04,
2749586333,CH2,55,AA,01,02,03,04,
2749646333,CH3,55,AA,01,02,03,04,
2749874666,CH2,55,AA,01,02,03,04,
2749934666,CH3,55,AA,01,02,03,04,
2750162666,CH2,55,AA,01,02,03,04,
2750222666,CH3,55,AA,01,02,03,04,
2750466666,CH2,55,AA,01,02,03,04,
2750526666,CH3,55,AA,01,02,03,04,
2750706666,CH2,55,AA,01,02,03,04,
```

Lines 1 to 33 | 0% | File Size: 162.42 KB (4,753 lines) [07/17

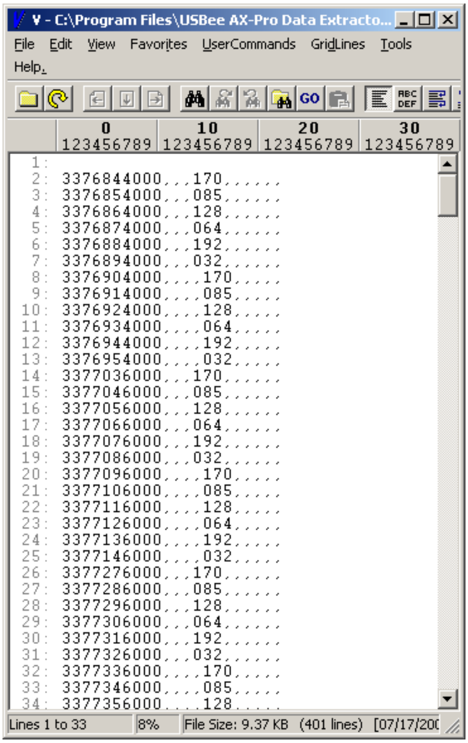
```
AsyncExtractor -O output.dex -P 3209 -C -Q 100000 -R 1000000 -E 8 -L
0 -M 255 -H -V 1
```



```

3009646333,,,55,,,,,,
3009656333,,,AA,,,,,,
3009666333,,,01,,,,,,
3009676666,,,02,,,,,,
3009686666,,,03,,,,,,
3009696666,,,04,,,,,,
3009706666,,,,55,,,,,
3009716666,,,,AA,,,,,
3009726666,,,,01,,,,,
3009736666,,,,02,,,,,
3009746666,,,,03,,,,,
3009756666,,,,04,,,,,
3009838666,,,55,,,,,,
3009848666,,,AA,,,,,,
3009858666,,,01,,,,,,
3009868666,,,02,,,,,,
3009878666,,,03,,,,,,
3009888666,,,04,,,,,,
3009898666,,,,55,,,,,
3009908666,,,,AA,,,,,
3009918666,,,,01,,,,,
3009928666,,,,02,,,,,
3009938666,,,,03,,,,,
3009948666,,,,04,,,,,
3010078666,,,55,,,,,,
3010088666,,,AA,,,,,,
3010098666,,,01,,,,,,
3010108666,,,02,,,,,,
3010118666,,,03,,,,,,
3010128666,,,04,,,,,,
3010138666,,,,55,,,,,
3010148666,,,,AA,,,,,
3010158666,,,,01,,,,,
AsyncExtractor -S -O output.dex -P 3209 -C -Q 400 -R 1000000 -E 8 -L
0 -M 255 -Z -H -V 1
```

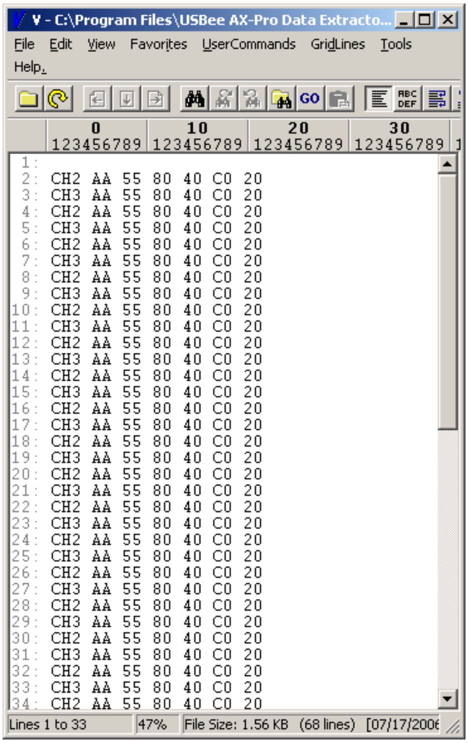


```

5907691666,,,AA,,,,,,
5907701666,,,55,,,,,,
5907711666,,,80,,,,,,
5907721666,,,40,,,,,,
5907731666,,,C0,,,,,,
5907741666,,,20,,,,,,
5907751666,,,,AA,,,,,
5907761666,,,,55,,,,,
5907771666,,,,80,,,,,
5907781666,,,,40,,,,,
5907791666,,,,C0,,,,,
5907801666,,,,20,,,,,
5907883666,,,AA,,,,,,
5907893666,,,55,,,,,,
5907903666,,,80,,,,,,
5907913666,,,40,,,,,,
5907923666,,,C0,,,,,,
5907933666,,,20,,,,,,
5907943666,,,,AA,,,,,
5907953666,,,,55,,,,,
5907963666,,,,80,,,,,
5907973666,,,,40,,,,,
5907983666,,,,C0,,,,,
5907993666,,,,20,,,,,
5908123666,,,AA,,,,,,
5908133666,,,55,,,,,,
5908143666,,,80,,,,,,
5908153666,,,40,,,,,,
5908163666,,,C0,,,,,,
5908173666,,,20,,,,,,
5908183666,,,,AA,,,,,
5908193666,,,,55,,,,,
5908203666,,,,80,,,,,
AsyncExtractor -S -O output.dex -P 3209 -C -Q 400 -R 1000000 -E 8 -L
0 -M 255 -Z -D -V 1
```



V - C:\Program Files\USBee AX-Pro Data Extracto...
File Edit View Favorites UserCommands GridLines Tools Help
GO
ABC DEF
0 123456789 10 123456789 20 123456789 30 123456789
Lines 1 to 33
8%
File Size: 9.37 KB (401 lines) [07/17/200

```

3376844000,,,170,,,,,
3376854000,,,085,,,,,
3376864000,,,128,,,,,
3376874000,,,064,,,,,
3376884000,,,192,,,,,
3376894000,,,032,,,,,
3376904000,,,,170,,,,
3376914000,,,,085,,,,
3376924000,,,,128,,,,
3376934000,,,,064,,,,
3376944000,,,,192,,,,
3376954000,,,,032,,,,
3377036000,,,170,,,,,
3377046000,,,085,,,,,
3377056000,,,128,,,,,
3377066000,,,064,,,,,
3377076000,,,192,,,,,
3377086000,,,032,,,,,
3377096000,,,,170,,,,
3377106000,,,,085,,,,
3377116000,,,,128,,,,
3377126000,,,,064,,,,
3377136000,,,,192,,,,
3377146000,,,,032,,,,
3377276000,,,170,,,,,
3377286000,,,085,,,,,
3377296000,,,128,,,,,
3377306000,,,064,,,,,
3377316000,,,192,,,,,
3377326000,,,032,,,,,
3377336000,,,,170,,,,
3377346000,,,,085,,,,
3377356000,,,,128,,,,
AsyncExtractor -S -O output.dex -P 3209 -Q 400 -R 1000000 -E 8 -L 0
-M 255 -Z -H -G -V 3
```


V - C:\Program Files\USBee AX-Pro Data Extracto...
File Edit View Favorites UserCommands GridLines Tools
Help
GO
0 10 20 30
123456789 123456789 123456789 123456789 1
Lines 1 to 33
47%
File Size: 1.56 KB (68 lines) [07/17/2006

```

CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
CH3 AA 55 80 40 C0 20
CH2 AA 55 80 40 C0 20
```

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedAsync.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned long BaudRate, unsigned int DataBits, unsigned int Parity,
unsigned char Channels, unsigned char MSFirst, unsigned char
StopBits)
```

PodNumber: Pod ID on the back of the USBee DX Test Pod

BaudRate: Baud rate of the async channels. All channels are decoded at the same rate.

Data Bits: Number of Data bits (5, 6, 7 or 8)

Parity:

- 0 = No parity bit
- 1 = Mark Parity
- 2 = Space Parity
- 3 = Even Parity
- 4 = Odd Parity

MSFirst:

- 0 = Least Significant Bit first
- 1 = Most Significant Bit first

Channels: Bit mask for which channels to decode (1 = signal 0, 128 = signal 7)

StopBits:

- 2 = 1 Stop Bit time
- 3 = 1.5 Stop Bit times
- 4 = 2 Stop Bit times

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

CWAV_EXPORT int CWAV_API StopExtraction(void);

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The Async Bus Extractor uses the following format for the data in this buffer:

```
Byte 0:  Timestamp LSByte  (in nanoseconds since start)
Byte 1:  Timestamp
Byte 2:  Timestamp
Byte 3:  Timestamp
Byte 4:  Timestamp
Byte 5:  Timestamp
Byte 6:  Timestamp
Byte 7: Timestamp MSByte
Byte 8: Record Type (bit 1 = 1 means character data is valid)
Byte 9:  Channel number (0 thru 7)
Byte 10: Character
Byte 11: Errors (Bit 0 = Parity Error, Bit 1 = Framing (Stop) error)
Byte 12: Control Signal States (all 8 signal bits except async
channels)
Byte 13:  reserved
Byte 14:  reserved
Byte 15:  reserved
(repeat) …
```

EXAMPLE SOURCE CODE

```
//*****************************************************
// USBee DX Data Extractor
// Async Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//*****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//*****************************************************
// Declare the Extractor DLL API routines
//*****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned long BaudRate,
unsigned int DataBits, unsigned int Parity, unsigned char Channels, unsigned char MSFirst,
unsigned char StopBits);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//*****************************************************
```

```
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char Y_LeastSignificantBitFirst = TRUE;
unsigned char Z_MostSignificantBitFirst = FALSE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned long M_SignalMask = 0xFFFFFFFF;
unsigned long Q_NumberOfBytes = 0;
unsigned long R_BaudRate = 9600;
unsigned long E_DataBits = 8;
unsigned long L_Parity = 0;
unsigned long V_Timestamps = 0;
unsigned long F_StopBits = 2;

typedef struct {

     __int64 TimeStamp;                 // 64-bit time stamp at the start of this character
or control signal change
     unsigned char RecordType;          // If the Character value is valid (1=Character is
good, 0=Character is don't care)
     unsigned char Signal;              // What channel this was sent on (0-7)
     unsigned char Character;      // Actual character data
     unsigned char Errors;              // Decodng error values (framing error, parity
error)
     unsigned char Control;             // Control signal states starting here

} AsyncEvent;

AsyncEvent *AEvent;

void DisplayHelp(void)
{
     fprintf(stdout,"\nAsyncExtractor [-?SADHBICGNXYZ] [-R BaudRate] [-E DataBits] [-L
Parity] [-M SignalMask] [-Q NumberOfBytes] [-V Timestamp] [-O filename] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Format Flags\n");

     fprintf(stdout,"    R  - Baud Rate (9600 baud default)\n");
     fprintf(stdout,"    E  - Number of Data Bits (5,6,7,8-default)\n");
     fprintf(stdout,"    L  - Parity Type (0=none(default), 1=mark, 2=space, 3=even,
4=odd)\n");
     fprintf(stdout,"    M  - Which Signals to capture (1=signal0, 128=signal7, 255=all,
0=none (default))\n");
     fprintf(stdout,"    Y  - LSBit first (default)\n");
     fprintf(stdout,"    Z  - MSBit first\n");
     fprintf(stdout,"    F  - Number of Stop Bits (2=1 (default), 3=1.5, 4=2)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");
     fprintf(stdout,"    A  - ASCII Text Values (\"1\")\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    B  - Binary Text Values (\"00110001\")\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    N  - Newline Delimited\n");
     fprintf(stdout,"    X  - No Delimeter\n");

     fprintf(stdout,"\n  Timestamp and Channel Labels\n");

     fprintf(stdout,"    V  - Timestamps and Labels (0=Both off(default),1=Time each
byte,2=Time and Labels,3=Labels Only)\n");




}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);

                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'Y':
                              Y_LeastSignificantBitFirst = TRUE;
                              Z_MostSignificantBitFirst = FALSE;
                              break;
                         case 'Z':
                              Z_MostSignificantBitFirst = TRUE;
                              Y_LeastSignificantBitFirst = FALSE;
                              break;
                         case 'A':
                              A_ASCIITextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'B':
                              B_BinaryTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'N':
                              N_NewlineDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'X':
                              X_NoDelimeter = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'E':
                              E_DataBits = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SignalMask = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'F':
                              F_StopBits = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_BaudRate = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'L':
                              L_Parity = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not
recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}
```

```
//****************************************************
// Main Entry Point.  The program starts here.
//****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("DX Data Extractor\n");
     printf("Async Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //****************************************************
     // Open up a file to store extracted data into
     //****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //****************************************************
     // Start the DX Pod extracting the data we want
     //****************************************************

     printf("BaudRate=%d DataBits=%d Parity=%d StopBits=%g\n", R_BaudRate, E_DataBits,
L_Parity, F_StopBits/2.0);

     RetValue = StartExtraction(P_PodID, R_BaudRate, E_DataBits, L_Parity, M_SignalMask,
Z_MostSignificantBitFirst, F_StopBits);

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************

               GetNextData( tempbuffer, length );

               if (I_BinaryValues)      // Just write out the binary data to a file
               {
                    totalbytes += length;

                    if (O_OutputFilename[0])
                         fwrite(tempbuffer, length, 1,  fout);    // Write it to a file

                    if (Q_NumberOfBytes)
                    {
                         if (Q_NumberOfBytes <= length)
                         {
                              goto Done;          // Done with that many bytes
                         }
                         Q_NumberOfBytes -= length;
                    }

               }
               else      // It's a text output so format it
               {

                    // Now figure out what to send to the output
                    for (unsigned long x = 0; x < length; x += sizeof(AsyncEvent))
                    {
                         AEvent = (AsyncEvent *)&tempbuffer[x];

                         if (AEvent->RecordType != 1)  // This type of record records the
edge changes of the other signals
                         {
                              continue;      // Since we only print out the characters
                         }
                         int Channel = AEvent->Signal;

                         //********************************************************
                         // Print the Timestamps and Channel Labels (if requested)
                         //********************************************************

                         if ((V_Timestamps == 1) || ((V_Timestamps >= 2) && (OldSignal !=
AEvent->Signal)))
                         {
                              if (V_Timestamps == 1)        // Print just the timestamp
                              {
                                   if (C_CommaDelimited)
                                        sprintf(outputstr,"\n%I64d,",AEvent->TimeStamp);

                                   if (G_SpaceDelimited)
                                        sprintf(outputstr,"\n%I64d ",AEvent->TimeStamp);

                                   // Now send it out to the screen or file
                                   if (S_Screen)
                                        fputs(outputstr, stdout);

                                   if (O_OutputFilename[0])
                                        fputs(outputstr, fout);

                                   outputstr[0] = 0;
                              }
                              else if (V_Timestamps == 2)         // Print timestamp and
channel number
                              {
                                   if (C_CommaDelimited)
                                        sprintf(outputstr,"\n%I64d,CH%d,",AEvent-
>TimeStamp,AEvent->Signal);

                                   if (G_SpaceDelimited)
                                        sprintf(outputstr,"\n%I64d CH%d ",AEvent-
>TimeStamp,AEvent->Signal);

                                   // Now send it out to the screen or file
                                   if (S_Screen)
                                        fputs(outputstr, stdout);

                                   if (O_OutputFilename[0])
                                        fputs(outputstr, fout);

                                   outputstr[0] = 0;
                              }
                              else if (V_Timestamps == 3)          // Print just the channel
number
                              {
                                   if (C_CommaDelimited)
                                        sprintf(outputstr,"\nCH%d,",AEvent->Signal);

                                   if (G_SpaceDelimited)
                                        sprintf(outputstr,"\nCH%d ",AEvent->Signal);

                                   if (S_Screen)
                                        fputs(outputstr, stdout);

                                   if (O_OutputFilename[0])
                                        fputs(outputstr, fout);

                                   outputstr[0] = 0;
                              }

                              OldSignal = AEvent->Signal;
                         }

                         //**********************************************************
                         // Print out the actual Async Channel Data
                         //**********************************************************

                         if (V_Timestamps == 1)   // Print the "Timestamp every byte"
format
                         {
                              for (int y = 0; y < 8;y++)
                              {
                                   if (Channel == y)        // Print a value here
                                   {
                                        OutputValue = AEvent->Character;

                                        // Now convert the value into the output text
                                        if (A_ASCIITextValues)
                                        {
                                             outputstr[0] = (unsigned char)OutputValue;
                                             outputstr[1] = 0;
                                        }
                                        if (D_DecimalTextValues)
                                        {
                                             sprintf(outputstr,"%03d",OutputValue);
                                        }
                                        if (B_BinaryTextValues)
                                        {
                                             int count;

                                             count = 8;

                                             unsigned int mask = 1 << (count - 1);
                                             for (int z = 0; z < count; z++)
                                             {
                                                  if (OutputValue & mask)
                                                       outputstr[z] = '1';
                                                  else
                                                       outputstr[z] = '0';
                                                  mask /= 2;
                                             }

                                             outputstr[z] = 0;

                                        }
                                        if (H_HexTextValues)
                                        {
                                             sprintf(outputstr,"%02X", OutputValue);
                                        }

                                        totalbytes++;

                                        if (Q_NumberOfBytes)
                                             if (--Q_NumberOfBytes == 0)
                                             {
                                                  goto Done;          // Done with that
many bytes
                                             }

                                   }
                                   // Now add  delimeters
                                   if (C_CommaDelimited)
```

```
               strcat(outputstr, ",");

          if (G_SpaceDelimited)
               strcat(outputstr, " ");

          if (N_NewlineDelimited)
               strcat(outputstr, "\n");

          // Now send it out to the screen or file
          if (S_Screen)
               fputs(outputstr, stdout);

          if (O_OutputFilename[0])
               fputs(outputstr, fout);

          outputstr[0] = 0;
     }

}
else // Print the "each line is a single channel" format
{

     OutputValue = AEvent->Character;

     // Now convert the value into the output text
     if (A_ASCIITextValues)
     {
          outputstr[0] = (unsigned char)OutputValue;
          outputstr[1] = 0;
     }
     if (D_DecimalTextValues)
     {
          sprintf(outputstr,"%03d",OutputValue);
     }
     if (B_BinaryTextValues)
     {
          int count;

          count = 8;

          unsigned int mask = 1 << (count - 1);
          for (int z = 0; z < count; z++)
          {
               if (OutputValue & mask)
                    outputstr[z] = '1';
               else
                    outputstr[z] = '0';
               mask /= 2;
          }

          outputstr[z] = 0;

     }
     if (H_HexTextValues)
     {
          sprintf(outputstr,"%02X", OutputValue);
     }

     totalbytes++;

     if (Q_NumberOfBytes)
          if (--Q_NumberOfBytes == 0)
          {
               goto Done;          // Done with that many bytes
          }

     // Now add  delimeters
     if (C_CommaDelimited)
          strcat(outputstr, ",");

     if (G_SpaceDelimited)
          strcat(outputstr, " ");

     if (N_NewlineDelimited)
          strcat(outputstr, "\n");

     // Now send it out to the screen or file
     if (S_Screen)
          fputs(outputstr, stdout);
```

```
                              if (O_OutputFilename[0])
                                   fputs(outputstr, fout);

                              outputstr[0] = 0;

                         }
                    }

               }

               if (timeout++ > 10  ) break;  // Let up once in a while to let the OS
process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //**************************************************
          // Check to see if we have fallen behind too far
          //**************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //**************************************************
          // Give the OS a little time to do something else
          //**************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //**************************************************
     // Close the file
     //**************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //**************************************************
     // Stop the extraction process
     //**************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

PARALLEL BUS DATA EXTRACTOR

The Parallel Bus Data Extractor takes the real-time streaming data from an embedded 8-bit parallel bus, formats it and allows you to save the data to disk or process it as it arrives.

PARALLEL BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- 8 digital channels
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Synchronous or Asynchronous Clocking
- Synchronous (external) clock 0 to 16MB/s*
- Asynchronous (internal) clock 1MB/s to 24MB/s*
- Input in 1, 2 or 4 byte serial words
- Little or Big Endian
- Output to Binary File*
- Output to Text File (Hex, Decimal, Binary or ASCII)*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Output Value Filtering
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 8 signal lines (0 thru 7), the GND (ground) line and optionally the CLK and TRG lines (for external timing). The signal 0 is represented in the bit 0 of each sampled byte. Connect the GND line to the digital ground of your system.

EXTRACTOR COMMAND LINE PROGRAM

The Parallel Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- (“cd \program files\USBee Data Extractor\Parallel”)
- Run the executable using the following command line arguments:

```
BasicExtractor [-?SADHBICGNX124YZ] [-E clock mode] [-Q
NumberOfBytes] [-T BytesPesLine] [-R SampleRate] [-M SignalMask] [-L
FilterValue] [-V FilterMask] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

1 - One Byte per value (default)

2 - Two Bytes per value

4 - Four Bytes per value

Y - Least significant byte first

A - ASCII Text Values ("1")

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

B - Binary Text Values ("00110001")

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

N - Newline Delimited

X - No Delimeter

T - Force Bytes Per Line (no force default)

M - Which Signals to capture (1=signal0,255=all(default))

L - Filter Mask (0=no filter,255=filter on all signals)

V - Filter Value (0=store when 0's,255=store when 1's)

E - Clocking mode (

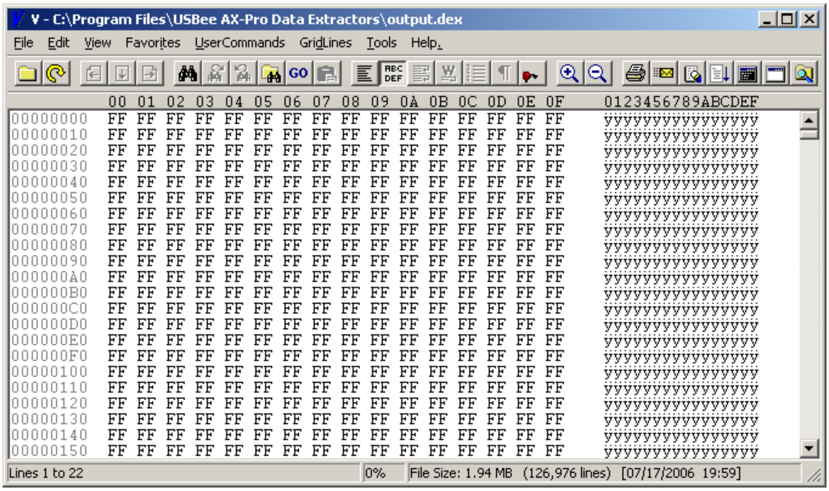
- 2=internal (default),
- 4=CLK rising, 5-CLK falling,
- 6-CLK rising AND TRG high, 7-CLK falling AND TRG high
- 8-CLK rising AND TRG low, 9-CLK falling AND TRG low

R - Internal CLK Sample Rate (1Msps default)

- 247 = 24MHz
- 167 = 16MHz
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz (default)

EXAMPLE OUTPUT

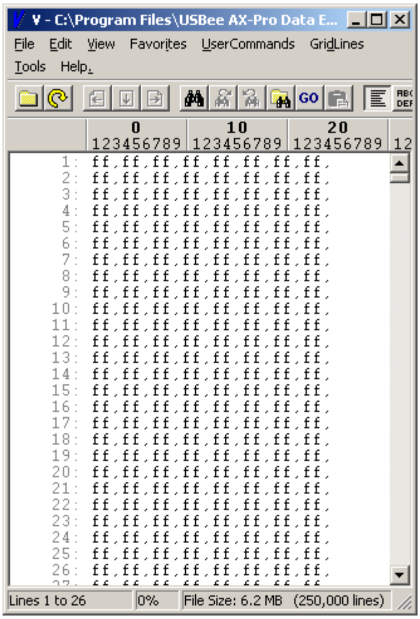
```
BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -I
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File
Edit
View
Favorites
UserCommands
GridLines
Tools
Help
GO

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F    0123456789ABCDEF
00000000  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000100  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000110  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000120  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000130  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000140  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000150  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

```
Lines 1 to 22 | 0% | File Size: 1.94 MB  (126,976 lines)  [07/17/2006  19:59]
```

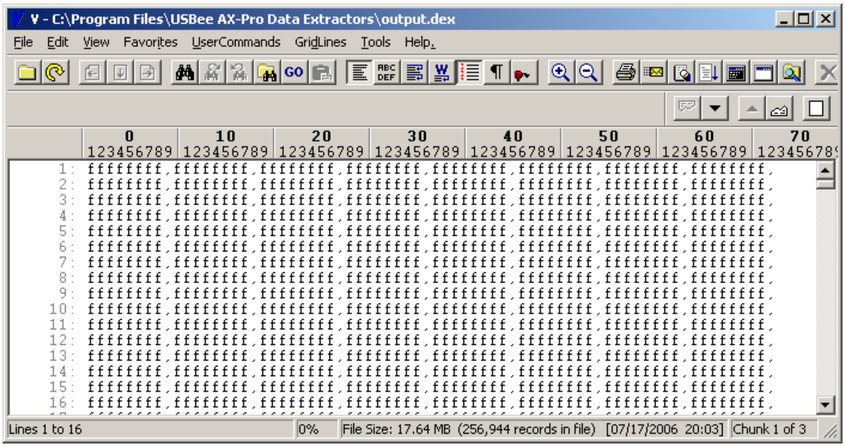
```
BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -C
```



```
          0         10         20
      123456789 123456789 123456789 12
   1: ff,ff,ff,ff,ff,ff,ff,ff,
   2: ff,ff,ff,ff,ff,ff,ff,ff,
   3: ff,ff,ff,ff,ff,ff,ff,ff,
   4: ff,ff,ff,ff,ff,ff,ff,ff,
   5: ff,ff,ff,ff,ff,ff,ff,ff,
   6: ff,ff,ff,ff,ff,ff,ff,ff,
   7: ff,ff,ff,ff,ff,ff,ff,ff,
   8: ff,ff,ff,ff,ff,ff,ff,ff,
   9: ff,ff,ff,ff,ff,ff,ff,ff,
  10: ff,ff,ff,ff,ff,ff,ff,ff,
  11: ff,ff,ff,ff,ff,ff,ff,ff,
  12: ff,ff,ff,ff,ff,ff,ff,ff,
  13: ff,ff,ff,ff,ff,ff,ff,ff,
  14: ff,ff,ff,ff,ff,ff,ff,ff,
  15: ff,ff,ff,ff,ff,ff,ff,ff,
  16: ff,ff,ff,ff,ff,ff,ff,ff,
  17: ff,ff,ff,ff,ff,ff,ff,ff,
  18: ff,ff,ff,ff,ff,ff,ff,ff,
  19: ff,ff,ff,ff,ff,ff,ff,ff,
  20: ff,ff,ff,ff,ff,ff,ff,ff,
  21: ff,ff,ff,ff,ff,ff,ff,ff,
  22: ff,ff,ff,ff,ff,ff,ff,ff,
  23: ff,ff,ff,ff,ff,ff,ff,ff,
  24: ff,ff,ff,ff,ff,ff,ff,ff,
  25: ff,ff,ff,ff,ff,ff,ff,ff,
  26: ff,ff,ff,ff,ff,ff,ff,ff,
```

```
Lines 1 to 26 | 0% | File Size: 6.2 MB  (250,000 lines)
```

```
BasicExtractor -O output.dex -P 3209 -1 -R 27 -T 8 -Q 2000000 -C -4
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File
Edit
View
Favorites
UserCommands
GridLines
Tools
Help
ABC DEF
GO

```
          0         10        20        30        40        50        60        70
  123456789 123456789 123456789 123456789 123456789 123456789 123456789 12345678
 1: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 2: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 3: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 4: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 5: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 6: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 7: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 8: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
 9: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
10: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
11: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
12: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
13: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
14: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
15: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
16: ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,
```

Lines 1 to 16 | 0% | File Size: 17.64 MB (256,944 records in file) [07/17/2006 20:03] | Chunk 1 of 3

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedBasic.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate,
unsigned long PodNumber, unsigned int ClockMode);
```

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee DX Test Pod

ClockMode:

- 2 = Internal Timing as in SampleRate parameter
- 4 – External Timing – sample on rising edge of CLK
- 5 – External Timing – sample on falling edge of CLK
- 6 – External Timing – sample on rising edge of CLK and TRG high
- 7 – External Timing – sample on falling edge of CLK and TRG high
- 8 – External Timing – sample on rising edge of CLK and TRG low
- 9 – External Timing – sample on falling edge of CLK and TRG low

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The Parallel Bus Extractor uses the following format for the data in this buffer:

Byte 0:	Byte 0 of the sampled data
Byte 1:	Byte 1 of the sampled data
Byte 2:	Byte 2 of the sampled data
Byte 3:	Byte 3 of the sampled data
…	
Byte N:	Byte N of the sampled data

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX Data Extractor
// Parallel Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber,
unsigned int ClockMode );
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//****************************************************
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char _1_BytePerValue = TRUE;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned long M_SignalMask = 0xFFFFFFFF;
unsigned long L_FilterMask = 0;
unsigned long V_FilterValue = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 17;
unsigned long Q_NumberOfBytes = 0;
// Not used yet J,K,Q,U,W

void DisplayHelp(void)
{
     fprintf(stdout,"\nBasicExtractor [-?SADHBICGNX124YZ] [-Q NumberOfBytes] [-T 
BytesPesLine] [-R SampleRate] [-M SignalMask] [-L FilterValue] [-V FilterMask] [-O 
filename] -P PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Number Format Flags\n");
     fprintf(stdout,"    1  - One   Byte  per value (default)\n");
     fprintf(stdout,"    2  - Two   Bytes per value\n");
     fprintf(stdout,"    4  - Four  Bytes per value\n");
     fprintf(stdout,"    Y  - Least significant byte first\n");
     fprintf(stdout,"    Z  - Most significant byte first\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");
     fprintf(stdout,"    A  - ASCII Text Values (\"1\")\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    B  - Binary Text Values (\"00110001\")\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    N  - Newline Delimited\n");
     fprintf(stdout,"    X  - No Delimeter\n");
     fprintf(stdout,"    T  - Force Bytes Per Line (no force default)\n");

     fprintf(stdout,"\n  Filter Values\n");
     fprintf(stdout,"    M  - Which Signals to capture (1=signal0,255=all(default))\n");
     fprintf(stdout,"    L  - Filter Mask (0=no filter,255=filter on all signals)\n");
     fprintf(stdout,"    V  - Filter Value (0=store when 0's,255=store when 1's)\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    E  - Clocking mode (2=internal (default),\n");
     fprintf(stdout,"                        4=CLK rising,5-CLK falling,\n");
     fprintf(stdout,"                        6-CLK rising AND TRG high,7-CLK falling AND 
TRG high\n");
     fprintf(stdout,"                        8-CLK rising AND TRG low,9-CLK falling AND TRG 
low\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (1Msps default)\n");


     exit(0);
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
```

```
}


//***************************************************
// Parse all of the command line options
//***************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case '1':
                              _1_BytePerValue = TRUE;
                              break;
                         case '2':
                              _2_BytePerValue = TRUE;
                              _1_BytePerValue = FALSE;
                              break;
                         case '4':
                              _4_BytePerValue = TRUE;
                              _1_BytePerValue = FALSE;
                              break;
                         case 'Y':
                              Y_LeastSignificantByteFirst = TRUE;
                              Z_MostSignificantByteFirst = FALSE;
                              break;
                         case 'Z':
                              Z_MostSignificantByteFirst = TRUE;
                              Y_LeastSignificantByteFirst = FALSE;
                              break;
                         case 'A':
                              A_ASCIITextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'B':
                              B_BinaryTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'N':
                              N_NewlineDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'X':
                              X_NoDelimeter = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'T':
                              T_ForceBytesPerLine = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SignalMask = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'L':
                              L_FilterMask = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_FilterValue = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'E':
                              E_ExternalClockMode = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              Error("Invalid Command Line Switch");
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}

unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{

     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);

}

//****************************************************
// Main Entry Point.  The program starts here.
//****************************************************
```

```
int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("DX Data Extractor\n");
     printf("Parallel Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //****************************************************
     // Open up a file to store extracted data into
     //****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //****************************************************
     // Start the DX Pod extracting the data we want
     //****************************************************


     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }

     printf("Processing and Saving Data to Disk.\n");

     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************
               StartTimer();

               GetNextData( tempbuffer, length );

               if (I_BinaryValues)      // Just write out the binary data to a file
               {
                    totalbytes += length;
                    if (O_OutputFilename[0])
                         fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

                    if (Q_NumberOfBytes)
                    {
                         if (Q_NumberOfBytes <= length)
                         {
                              goto Done;          // Done with that many bytes
                         }
                         Q_NumberOfBytes -= length;
                    }

               }
               else      // It's a text output so format it all pretty-like
               {

                    // Now figure out what to send to the output
                    for (unsigned long x = 0; x < length;)
                    {
                         // First get the value to print out
                         if (_1_BytePerValue)
                         {
                              OutputValue = tempbuffer[x];
                              x++;
                         }
                         if (_2_BytePerValue)
                         {
                              if (Y_LeastSignificantByteFirst)
                                   OutputValue = (tempbuffer[x+1] << 8) + tempbuffer[x+0];
                              else
                                   OutputValue = (tempbuffer[x+0] << 8) + tempbuffer[x+1];
                              x += 2;
                         }
                         if (_4_BytePerValue)
                         {
                              if (Y_LeastSignificantByteFirst)
                                   OutputValue =  (tempbuffer[x+3] << 24) +
                                                       (tempbuffer[x+2] << 16) +
                                                       (tempbuffer[x+1] << 8) +
                                                       tempbuffer[x+0];
                              else
                                   OutputValue =  (tempbuffer[x+0] << 24) +
                                                       (tempbuffer[x+1] << 16) +
                                                       (tempbuffer[x+2] << 8) +
                                                       tempbuffer[x+3];

                              x += 4;
                         }

                         // Perform the Masking
                         OutputValue &= M_SignalMask;

                         // Perform the filtering
                         if ((OutputValue & L_FilterMask) != V_FilterValue)
                              continue;      // Not for use to save so move on.

                         // Now convert the value into the output text
                         if (A_ASCIITextValues)
                         {
                              outputstr[0] = (unsigned char)OutputValue;
                              outputstr[1] = 0;
                         }
                         if (D_DecimalTextValues)
                         {
                              ultoa(OutputValue,outputstr,10);
                              // sprintf(outputstr,"%d",OutputValue);
                         }
                         if (B_BinaryTextValues)
                         {
                              int count;

                              if (_1_BytePerValue)
                                   count = 8;
                              if (_2_BytePerValue)
                                   count = 16;
                              if (_4_BytePerValue)
                                   count = 32;

                              unsigned int mask = 1 << (count - 1);
                              for (int z = 0; z < count; z++)
```

```
                              {
                                   if (OutputValue & mask)
                                        outputstr[z] = '1';
                                   else
                                        outputstr[z] = '0';
                                   mask /= 2;
                              }
                         }
                         if (H_HexTextValues)
                         {
                              if (_1_BytePerValue)
                                   ultoa(OutputValue, outputstr, 16);
                                   //sprintf(outputstr,"%02X", OutputValue);
                              if (_2_BytePerValue)
                                   ultoa(OutputValue, outputstr, 16);
                                   //sprintf(outputstr,"%04X", OutputValue);
                              if (_4_BytePerValue)
                                   ultoa(OutputValue, outputstr, 16);
                                   //sprintf(outputstr,"%08X", OutputValue);
                         }

                         // Now add any delimeters to the end of the value
                         if (C_CommaDelimited)
                              strcat(outputstr, ",");

                         if (G_SpaceDelimited)
                              strcat(outputstr, " ");

                         if (N_NewlineDelimited)
                              strcat(outputstr, "\n");

                         if (T_ForceBytesPerLine)
                         {
                              if (++ByteCounter >= T_ForceBytesPerLine)
                              {
                                   ByteCounter = 0;
                                   strcat(outputstr, "\n");
                              }
                         }

                         if (S_Screen)
                              fputs(outputstr, stdout);

                         if (O_OutputFilename[0])
                              fputs(outputstr, fout);

                         totalbytes++;

                         if (Q_NumberOfBytes)
                              if (--Q_NumberOfBytes == 0)
                              {
                                   goto Done;          // Done with that many bytes
                              }

                    }

               }

               // StopTimer();

               if (timeout++ > 10  ) break;  // Let up once in a while to let the OS
process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //**************************************************
          // Give the OS a little time to do something else
          //**************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //**************************************************
     // Close the file
     //**************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //**************************************************
     // Stop the extraction process
     //**************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

SERIAL BUS DATA EXTRACTOR

The Serial Bus Data Extractor takes the real-time streaming data from up to 8 serial data lines, formats it and allows you to save the data to disk or process it as it arrives.

SERIAL BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- 8 digital channels
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Synchronous or Asynchronous Clocking
- Synchronous (external) clock 0 to 16MB/s*
- Asynchronous (internal) clock 1MB/s to 24MB/s*
- Input in 1, 2 or 4 byte serial words
- Little or Big Endian
- Output to Binary File*
- Output to Text File (Hex, Decimal, Binary or ASCII)*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Output Value Filtering
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 Data Extractor uses any of the 8 signal lines (0 thru 7), the GND (ground) line and optionally the CLK and TRG lines (for external timing). Connect the GND line to the digital ground of your system.

EXTRACTOR COMMAND LINE PROGRAM

The Serial Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- ("cd \program files\USBee Data Extractor\Serial")
- Run the executable using the following command line arguments:

```
SerialExtractor [-?SADHBICGNX124YZ] [-Q NumberOfBytes] [-T
BytesPesLine] [-R SampleRate] [-E ClockingMode] [-M SignalMask] [-J
ChannelAlign] [-L SignalLevel] [-V AlignmentValue] [-O filename] -P
PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

1 - One Byte per value (default)

2 - Two Bytes per value

4 - Four Bytes per value

Y - Least significant bit first

Z - Most significant bit first

A - ASCII Text Values ("1")

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

B - Binary Text Values ("00110001")

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

N - Newline Delimited

X - No Delimeter

T - Force Bytes Per Line (no force default)

M - Which Signals to capture (1=signal0,255=all(default))

V - Align on Value

L - Align on Signal Level (0=low,1=high)

J - Which signal to use for alignment (1=signal0,128=signal7)

E - Clocking mode

- 2=internal (default),
- 4=CLK rising, 5-CLK falling,
- 6-CLK rising AND TRG high, 7-CLK falling AND TRG high
- 8-CLK rising AND TRG low, 9-CLK falling AND TRG low

R - Internal CLK Sample Rate (1Msps default)

- 247 = 24MHz
- 167 = 16MHz
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz (default)

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedSerial.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate,
unsigned long PodNumber, unsigned int ClockMode, unsigned long
AlignValue, unsigned char SignalLevel, unsigned char AlignChannel,
unsigned char BytePerValue);
```

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps

- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee DX Test Pod

ClockMode:

- 2 = Internal Timing as in SampleRate parameter
- 4 – External Timing – sample on rising edge of CLK
- 5 – External Timing – sample on falling edge of CLK
- 6 – External Timing – sample on rising edge of CLK and TRG high
- 7 – External Timing – sample on falling edge of CLK and TRG high
- 8 – External Timing – sample on rising edge of CLK and TRG low
- 9 – External Timing – sample on falling edge of CLK and TRG low

AlignValue: Value which the extractor syncs with to define bit 0 alignment.

SignalLevel: Level, 0 or 1, which the extractor syncs with to define bit 0 aligment

AlignChannel: Which signal the extractor uses for alignment, either via value or signal

BytesPerValue: 1, 2, or 4. Used for Value alignment size.

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The Serial Bus Extractor uses the following format for the data in this buffer:

```
Byte 0:         Channel 0, first byte extracted
Byte 1:         Channel 1, first byte extracted
Byte 2:         Channel 2, first byte extracted
Byte 3:         Channel 3, first byte extracted
Byte 4:         Channel 4, first byte extracted
Byte 5:         Channel 5, first byte extracted
Byte 6:         Channel 6, first byte extracted
Byte 7:         Channel 7, first byte extracted
Byte 8:         Channel 0, second byte extracted
Byte 9:         Channel 1, second byte extracted
…
Byte N:         Channel (N mod 8), byte (N/8)+1 extracted
```

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX Data Extractor
// Serial Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber,
unsigned int ClockMode, unsigned long AlignValue,
                                                   unsigned char SignalLevel, unsigned char
AlignChannel, unsigned char BytePerValue);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//****************************************************
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char BytePerValue = 1;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned long M_SignalMask = 0xFFFFFFFF;
unsigned char L_SignalLevel = 0;
unsigned long V_AlignValue = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char J_ChannelAlign = 0;
unsigned char R_SampleRate = 17;
unsigned long Q_NumberOfBytes = 0;

void DisplayHelp(void)
{
     fprintf(stdout,"\nSerialExtractor [-?SADHBICGNX124YZ] [-Q NumberOfBytes] [-T
BytesPesLine] [-R SampleRate] [-E ClockingMode] [-M SignalMask] [-J ChannelAlign] [-L
SignalLevel] [-V AlignmentValue] [-O filename] -P PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Number Format Flags\n");
     fprintf(stdout,"    1  - One   Byte  per value (default)\n");
     fprintf(stdout,"    2  - Two   Bytes per value\n");
     fprintf(stdout,"    4  - Four  Bytes per value\n");
     fprintf(stdout,"    Y  - Least significant byte first\n");
     fprintf(stdout,"    Z  - Most significant byte first\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");
     fprintf(stdout,"    A  - ASCII Text Values (\"1\")\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    B  - Binary Text Values (\"00110001\")\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    N  - Newline Delimited\n");
     fprintf(stdout,"    X  - No Delimeter\n");
     fprintf(stdout,"    T  - Force Bytes Per Line (no force default)\n");

     fprintf(stdout,"\n  Filter Values\n");
     fprintf(stdout,"    M  - Which Signals to capture (1=signal0,255=all(default))\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    E  - Clocking mode (2=internal (default),\n");
     fprintf(stdout,"                        4=CLK rising,5-CLK falling,\n");
     fprintf(stdout,"                        6-CLK rising AND TRG high,7-CLK falling AND 
TRG high\n");
     fprintf(stdout,"                        8-CLK rising AND TRG low,9-CLK falling AND TRG 
low\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (1Msps default)\n");

     fprintf(stdout,"\n  Bit Zero Alignment Setting\n");
     fprintf(stdout,"    V  - Align on Value\n");
     fprintf(stdout,"    L  - Align on Signal Level (0=Low, 1=High)\n");
     fprintf(stdout,"    J  - Align on Which Channel (1=Ch 0, 128=Ch 7)\n");

     exit(0);
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands 
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
```

```
case '?':
     DisplayHelp();
     break;
case 'S':
     S_Screen = TRUE;
     break;
case '1':
     BytePerValue = 1;
     break;
case '2':
     BytePerValue = 2;
     break;
case '4':
     BytePerValue = 4;
     break;
case 'Y':
     Y_LeastSignificantByteFirst = TRUE;
     Z_MostSignificantByteFirst = FALSE;
     break;
case 'Z':
     Z_MostSignificantByteFirst = TRUE;
     Y_LeastSignificantByteFirst = FALSE;
     break;
case 'A':
     A_ASCIITextValues = TRUE;
     H_HexTextValues = FALSE;
     break;
case 'D':
     D_DecimalTextValues = TRUE;
     H_HexTextValues = FALSE;
     break;
case 'H':
     H_HexTextValues = TRUE;
     break;
case 'B':
     B_BinaryTextValues = TRUE;
     H_HexTextValues = FALSE;
     break;
case 'I':
     I_BinaryValues = TRUE;
     H_HexTextValues = FALSE;
     break;
case 'C':
     C_CommaDelimited = TRUE;
     G_SpaceDelimited = FALSE;
     break;
case 'G':
     G_SpaceDelimited = TRUE;
     break;
case 'N':
     N_NewlineDelimited = TRUE;
     G_SpaceDelimited = FALSE;
     break;
case 'X':
     X_NoDelimeter = TRUE;
     G_SpaceDelimited = FALSE;
     break;
case 'T':
     T_ForceBytesPerLine = (DWORD)strtol(argv[++i],NULL,0);
     cont = FALSE;
     break;
case 'Q':
     Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
     cont = FALSE;
     break;
case 'M':
     M_SignalMask = (DWORD)strtol(argv[++i],NULL,0);
     cont = FALSE;
     break;
case 'L':
     L_SignalLevel = (BYTE)strtol(argv[++i],NULL,0);
     cont = FALSE;
     break;
case 'V':
     V_AlignValue = (DWORD)strtol(argv[++i],NULL,0);
     cont = FALSE;
     break;
case 'E':
     E_ExternalClockMode = (DWORD)strtol(argv[++i],NULL,0);
     cont = FALSE;
```

```
                        break;
                    case 'J':
                        J_ChannelAlign = (BYTE)strtol(argv[++i],NULL,0);
                        cont = FALSE;
                        break;
                    case 'R':
                        R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                        cont = FALSE;
                        break;
                    case 'w':
                        WordExample = (DWORD)strtol(argv[++i],NULL,0);
                        cont = FALSE;
                        break;
                    case 'b':
                        ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                        cont = FALSE;
                        break;
                    default:
                        DisplayHelp();
                        Error("Invalid Command Line Switch");
                }
        }
    }


    // Now check to see if they make sense
    if (P_PodID == 0)
    {
        DisplayHelp();
        Error("No Pod Number Specified");
    }

}

unsigned long StartTime;

void StartTimer()
{

    StartTime = GetTickCount();
}

void StopTimer()
{

    printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);

}

//****************************************************
// Main Entry Point.  The program starts here.
//****************************************************

int main(int argc, char* argv[])
{
    int RetValue;
    unsigned long totalbytes = 0;
    char *outputstr = new char [256];
    unsigned long ByteCounter = 0;
    unsigned long OutputValue;

    printf("DX Data Extractor\n");
    printf("Serial Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

    // Parse out the command line options
    ParseCommandLine( argc, argv );

    //****************************************************
    // Open up a file to store extracted data into
    //****************************************************

    FILE *fout;
    if (O_OutputFilename[0])
    {
        if (I_BinaryValues)
            fout = fopen((char*)O_OutputFilename, "wb");
        else
            fout = fopen((char*)O_OutputFilename, "w");
    }
```

```
     //****************************************************
     // Start the DX Pod extracting the data we want
     //****************************************************


     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode, V_AlignValue,
L_SignalLevel, J_ChannelAlign, BytePerValue);

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }

     printf("Processing and Saving Data to Disk.\n");

     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     int KeepLooping = TRUE;
     printf("BytePerValue = %d, M_SignalMask = %d\n",BytePerValue, M_SignalMask);
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************
               StartTimer();

               GetNextData( tempbuffer, length );

               if (I_BinaryValues)      // Just write out the binary data to a file
               {
                    totalbytes += length;

                    if (O_OutputFilename[0])
                         fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

                    if (Q_NumberOfBytes)
                    {
                         if (Q_NumberOfBytes <= length)
                         {
                              goto Done;          // Done with that many bytes
                         }
                         Q_NumberOfBytes -= length;
                    }

               }
               else      // It's a text output so format it all pretty-like
               {

                    // Now figure out what to send to the output
                    for (unsigned long x = 0; x < length; x+=(8 * BytePerValue))
     //Do multiple of 8 values at a time becuase each one is a data line
                    {
                         sprintf(outputstr, "\n%08X: ",x);
                         fputs(outputstr, fout);
                         //First, check which lines we want
                         for (unsigned char y = 0; y < 8; y++)
                         {
                              sprintf(outputstr, "%02X ",tempbuffer[x+y]);
                              fputs(outputstr, fout);

                              if (M_SignalMask & (2^y))     //Check mask value
                              {
                                   // First get the value to print out
                                   if (BytePerValue == 1)
                                   {
                                        OutputValue = tempbuffer[x + y];
                                   }
                                   if (BytePerValue == 2)
                                   {
                                        if (Y_LeastSignificantByteFirst)
                                             OutputValue = (tempbuffer[x+8+y] << 8) +
tempbuffer[x+0+y];
                                        else
                                             OutputValue = (tempbuffer[x+0+y] << 8) +
tempbuffer[x+8+y];
                                   }
                                   if (BytePerValue == 4)
                                   {
                                        if (Y_LeastSignificantByteFirst)
                                             OutputValue =  (tempbuffer[x+32+y] << 24) +
                                                                  (tempbuffer[x+16+y] << 16)
+
                                                                  (tempbuffer[x+8+y] << 8) +
                                                                  tempbuffer[x+0+y];
                                        else
                                             OutputValue =  (tempbuffer[x+0+y] << 24) +
                                                                  (tempbuffer[x+8+y] << 16)
+
                                                                  (tempbuffer[x+16+y] << 8)
+
                                                                  tempbuffer[x+32+y];

                                   }

                                   // Now convert the value into the output text
                                   if (A_ASCIITextValues)
                                   {
                                        outputstr[0] = (unsigned char)OutputValue;
                                        outputstr[1] = 0;
                                   }
                                   if (D_DecimalTextValues)
                                   {
                                        ultoa(OutputValue,outputstr,10);
                                        // sprintf(outputstr,"%d",OutputValue);
                                   }
                                   if (B_BinaryTextValues)
                                   {
                                        int count;

                                        if (BytePerValue == 1)
                                             count = 8;
                                        if (BytePerValue == 2)
                                             count = 16;
                                        if (BytePerValue == 4)
                                             count = 32;

                                        unsigned int mask = 1 << (count - 1);
                                        for (int z = 0; z < count; z++)
                                        {
                                             if (OutputValue & mask)
                                                  outputstr[z] = '1';
                                             else
                                                  outputstr[z] = '0';
                                             mask /= 2;
                                        }
                                   }
                                   if (H_HexTextValues)
                                   {
                                        if (BytePerValue == 1)
                                             ultoa(OutputValue, outputstr, 16);
                                             //sprintf(outputstr,"%02X", OutputValue);
                                        if (BytePerValue == 2)
                                             ultoa(OutputValue, outputstr, 16);
                                             //sprintf(outputstr,"%04X", OutputValue);
                                        if (BytePerValue == 4)
                                             ultoa(OutputValue, outputstr, 16);
                                             //sprintf(outputstr,"%08X", OutputValue);
                                   }
                                   // Now add any delimeters to the end of the value
                                   if (C_CommaDelimited)
                                        strcat(outputstr, ",");

                                   if (G_SpaceDelimited)
                                        strcat(outputstr, " ");

                                   if (N_NewlineDelimited)
                                        strcat(outputstr, "\n");

                                   if (T_ForceBytesPerLine)
                                   {
                                        if (++ByteCounter >= T_ForceBytesPerLine)
                                        {
                                             ByteCounter = 0;
                                             strcat(outputstr, "\n");
                                        }
                                   }

                                   if (S_Screen)
                                        fputs(outputstr, stdout);

                                   if (O_OutputFilename[0])
                                        fputs(outputstr, fout);

                                   totalbytes++;

                                   if (Q_NumberOfBytes)
                                        if (--Q_NumberOfBytes == 0)
                                        {
                                             goto Done;          // Done with that many
bytes
                                        }
                              }
                         }
                    }

               }

               // StopTimer();

               if (timeout++ > 10  ) break;  // Let up once in a while to let the OS
process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //**************************************************
          // Check to see if we have fallen behind too far
          //**************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //**************************************************
          // Give the OS a little time to do something else
          //**************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);
     //**************************************************
     // Close the file
     //**************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //**************************************************
     // Stop the extraction process
     //**************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

I2C DATA EXTRACTOR

The I^{2}C Bus Data Extractor takes the real-time streaming data from the I2C bus, formats it and allows you to save the data to disk or process it as it arrives.

I2C DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one I^{2}C Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma or Space Delimited files
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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^{2}C Bus Data Extractor connects to the SDA and SCL lines of the I^{2}C 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^{2}C bus using the test clips provided.

EXTRACTOR COMMAND LINE PROGRAM

The I^{2}C Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- (“cd \program files\USBee Data Extractor\I2C”)
- Run the executable using the following command line arguments:

```
I2CExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O
filename] [-M SDA] [-N SCL] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - SDA signal Mask (1-Ch0, 128=Ch7, Ch0 default)

N - SCL signal Mask (1-Ch0, 128=Ch7, Ch1 default)

A - All Packet Fields are output (default)

B – Only Data Bytes are output

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

V - Timestamps (0=off, 1=each packet start)

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedI2C.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned char All, unsigned char Decimal, unsigned char Hex,
unsigned char Binary, unsigned char Comma, unsigned char Space,
unsigned char Timestamps, unsigned long SDAMask, unsigned long
SCLMask)
```

PodNumber: Pod ID on the back of the USBee DX Test Pod

All:

- 0 – Only the data payload bytes are returned
- 1 – All I2C packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

SDAMask:

- The mask for the channel to use for SDA
- (1 = Ch0, 128 = Ch7)

SCLMask:

- The mask for the channel to use for SCL
- (1 = Ch0, 128 = Ch7)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

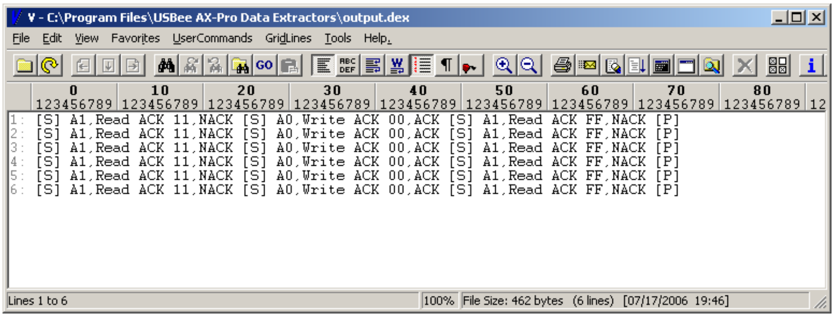
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

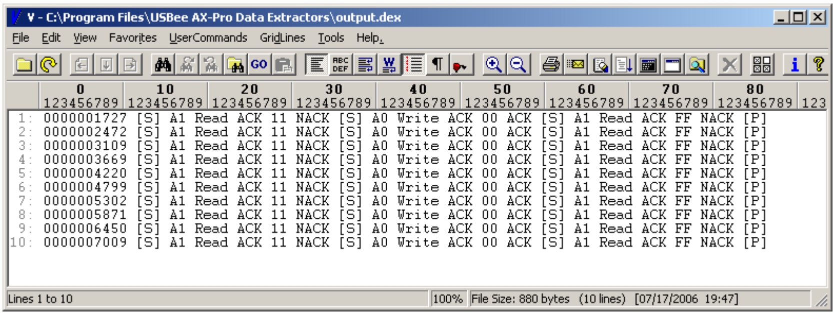
The I^{2}C Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the I2C traffic in Hex text form, separated by any specified delimiters.

```
I2CExtractor -O output.dex -P 3209 -Q 5000 -H -C -M 2 -N 1 -V 0
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help
Lines 1 to 6
100%
File Size: 462 bytes (6 lines) [07/17/2006 19:46]

```
[S] A1,Read ACK 11,NACK [S] A0,Write ACK 00,ACK [S] A1,Read ACK FF,NACK [P]
[S] A1,Read ACK 11,NACK [S] A0,Write ACK 00,ACK [S] A1,Read ACK FF,NACK [P]
[S] A1,Read ACK 11,NACK [S] A0,Write ACK 00,ACK [S] A1,Read ACK FF,NACK [P]
[S] A1,Read ACK 11,NACK [S] A0,Write ACK 00,ACK [S] A1,Read ACK FF,NACK [P]
[S] A1,Read ACK 11,NACK [S] A0,Write ACK 00,ACK [S] A1,Read ACK FF,NACK [P]
[S] A1,Read ACK 11,NACK [S] A0,Write ACK 00,ACK [S] A1,Read ACK FF,NACK [P]
```

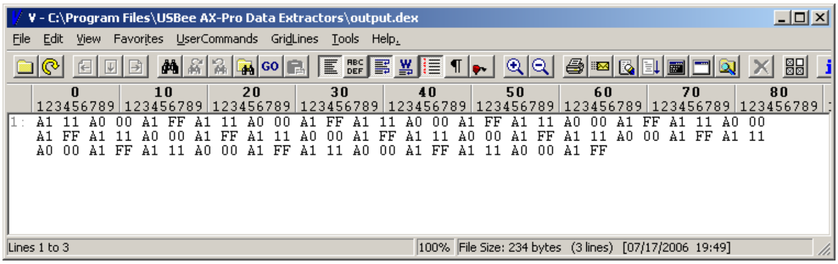
```
I2CExtractor -O output.dex -P 3209 -Q 5000 -H -G -M 2 -N 1 -V 1
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help

```
0000001727 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000002472 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000003109 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000003669 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000004220 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000004799 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000005302 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000005871 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000006450 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
0000007009 [S] A1 Read ACK 11 NACK [S] A0 Write ACK 00 ACK [S] A1 Read ACK FF NACK [P]
```

Lines 1 to 10 | 100% | File Size: 880 bytes (10 lines) [07/17/2006 19:47]

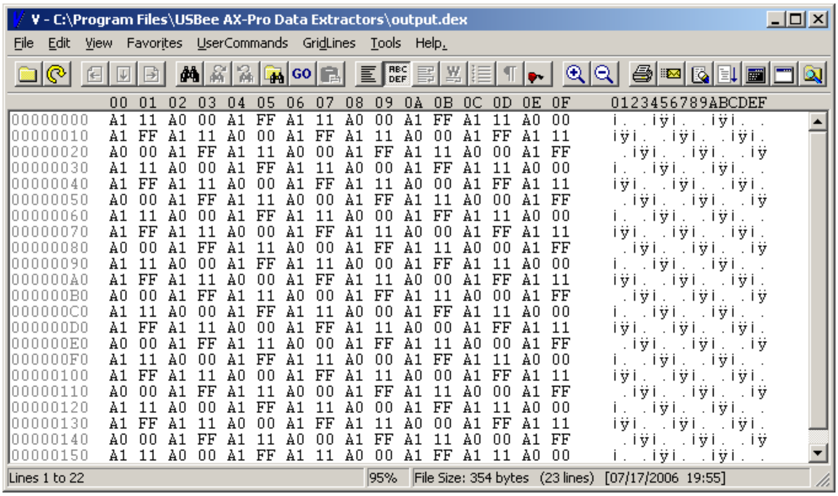
```
I2CExtractor -O output.dex -P 3209 -Q 5000 -B -M 2 -N 1
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help

```
A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00
A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11
A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF
```

Lines 1 to 3 | 100% | File Size: 234 bytes (3 lines) [07/17/2006 19:49]

```
I2CExtractor -O output.dex -P 3209 -Q 5000 -I -M 2 -N 1
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help

```
         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
00000000 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
00000010 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11   iÿi. .iÿi. .iÿi.
00000020 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF    .iÿi. .iÿi. .iÿ
00000030 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
00000040 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11   iÿi. .iÿi. .iÿi.
00000050 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF    .iÿi. .iÿi. .iÿ
00000060 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
00000070 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11   iÿi. .iÿi. .iÿi.
00000080 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF    .iÿi. .iÿi. .iÿ
00000090 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
000000A0 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11   iÿi. .iÿi. .iÿi.
000000B0 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF    .iÿi. .iÿi. .iÿ
000000C0 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
000000D0 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11   iÿi. .iÿi. .iÿi.
000000E0 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF    .iÿi. .iÿi. .iÿ
000000F0 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
00000100 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11   iÿi. .iÿi. .iÿi.
00000110 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF    .iÿi. .iÿi. .iÿ
00000120 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
00000130 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11   iÿi. .iÿi. .iÿi.
00000140 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF    .iÿi. .iÿi. .iÿ
00000150 A1 11 A0 00 A1 FF A1 11 A0 00 A1 FF A1 11 A0 00   i. .iÿi. .iÿi. .
```

Lines 1 to 22 | 95% | File Size: 354 bytes (23 lines) [07/17/2006 19:55]

EXAMPLE SOURCE CODE

```
//***************************************************
// USBee DX Data Extractor
// I2C Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//***************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//***************************************************
// Declare the Extractor DLL API routines
//***************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned char All,
unsigned char Decimal, unsigned char Hex, unsigned char Binary, unsigned char Comma,
unsigned char Space, unsigned char Timestamps, unsigned long SDA,unsigned long SCL);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//***************************************************
// Define the working buffer
//***************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = FALSE;
unsigned long Q_NumberOfBytes = 0;
unsigned long V_Timestamps = TRUE;
unsigned long M_SDA = 1;
unsigned long N_SCL = 2;


void DisplayHelp(void)
{
     fprintf(stdout,"\nI2CExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O
filename] [-M SDAMask] [-N SCLMask] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");
     fprintf(stdout,"\n  Input Format Flags\n");

     fprintf(stdout,"    R  - Bus Speed in bits/second (default = 250000)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");
     fprintf(stdout,"    M  - SDA signal (1=ch0, 128=ch7, ch0 default)\n");
     fprintf(stdout,"    N  - SCL signal (1=ch0, 128=ch7, ch1 default)\n");



}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);

                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SDA = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'N':
                              N_SCL = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not
recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}


//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("DX Data Extractor\n");
     printf("I2C Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //*****************************************************
     // Open up a file to store extracted data into
     //*****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //**************************************************
     // Start the DX Pod extracting the data we want
     //**************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, A_All, D_DecimalTextValues, H_HexTextValues,
I_BinaryValues, C_CommaDelimited, G_SpaceDelimited, V_Timestamps, M_SDA, N_SCL) ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //**************************************************
     // Loop and do something with the collected data
     //**************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //**************************************************
          // If there is data that has come in
          //**************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //**************************************************
               // Get the data into our local working buffer
               //**************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }

               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);
          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //****************************************************
          // Give the OS a little time to do something else
          //****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //****************************************************
     // Close the file
     //****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //****************************************************
     // Stop the extraction process
     //****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

SM BUS DATA EXTRACTOR

The SM Bus Data Extractor takes the real-time streaming data from the SM bus, formats it and allows you to save the data to disk or process it as it arrives.

SM BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one SM Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma or Space Delimited files
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 Data Extractor connects to the SMBClk and SMBData lines of the SM Bus. Use one signal as the SMBData line and one signal as the SMBClk line. Also connect the GND line to the digital ground of your system. Connect these signals to the SM Bus using the test clips provided.

EXTRACTOR COMMAND LINE PROGRAM

The SM Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- (“cd \program files\USBee Data Extractor\SMBus”)
- Run the executable using the following command line arguments:

```
SMBusExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O
filename] [-M SMBDatMask] [-N SMBClkMask] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - SMBData signal Mask (1-Ch0, 128=Ch7, Ch0 default)

N - SMBClk signal Mask (1-Ch0, 128=Ch7, Ch1 default)

A - All Packet Fields are output (default)

B – Only Data Bytes are output

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

V - Timestamps (0=off, 1=each packet start)

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedSMBus.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned char All, unsigned char Decimal, unsigned char Hex,
unsigned char Binary, unsigned char Comma, unsigned char Space,
unsigned char Timestamps, unsigned long SMBData, unsigned long
SMBClk);
```

PodNumber: Pod ID on the back of the USBee DX Test Pod

All:

- 0 – Only the data payload bytes are returned
- 1 – All SMBus packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

SMBData:

- The mask for the channel to use for Data
- (1 = Ch0, 128 = Ch7)

SMDClk:

- The mask for the channel to use for Clk
- (1 = Ch0, 128 = Ch7)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

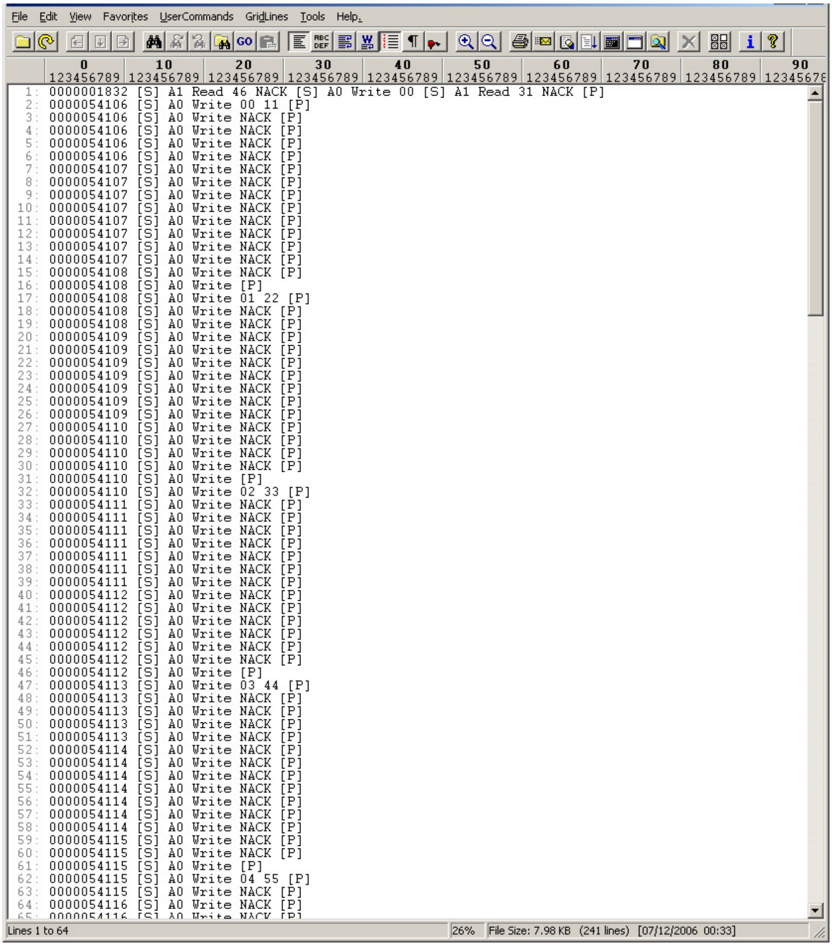
Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The SM Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the SMBus traffic in Hex text form, separated by any specified delimiters.



```
0000001832 [S] A1 Read 46 NACK [S] A0 Write 00 [S] A1 Read 31 NACK [P]
0000054106 [S] A0 Write 00 11 [P]
0000054106 [S] A0 Write NACK [P]
0000054106 [S] A0 Write NACK [P]
0000054106 [S] A0 Write NACK [P]
0000054106 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054107 [S] A0 Write NACK [P]
0000054108 [S] A0 Write NACK [P]
0000054108 [S] A0 Write [P]
0000054108 [S] A0 Write 01 22 [P]
0000054108 [S] A0 Write NACK [P]
0000054108 [S] A0 Write NACK [P]
0000054109 [S] A0 Write NACK [P]
0000054109 [S] A0 Write NACK [P]
0000054109 [S] A0 Write NACK [P]
0000054109 [S] A0 Write NACK [P]
0000054109 [S] A0 Write NACK [P]
0000054109 [S] A0 Write NACK [P]
0000054109 [S] A0 Write NACK [P]
0000054110 [S] A0 Write NACK [P]
0000054110 [S] A0 Write NACK [P]
0000054110 [S] A0 Write NACK [P]
0000054110 [S] A0 Write NACK [P]
0000054110 [S] A0 Write [P]
0000054110 [S] A0 Write 02 33 [P]
0000054111 [S] A0 Write NACK [P]
0000054111 [S] A0 Write NACK [P]
0000054111 [S] A0 Write NACK [P]
0000054111 [S] A0 Write NACK [P]
0000054111 [S] A0 Write NACK [P]
0000054111 [S] A0 Write NACK [P]
0000054111 [S] A0 Write NACK [P]
0000054112 [S] A0 Write NACK [P]
0000054112 [S] A0 Write NACK [P]
0000054112 [S] A0 Write NACK [P]
0000054112 [S] A0 Write NACK [P]
0000054112 [S] A0 Write NACK [P]
0000054112 [S] A0 Write NACK [P]
0000054112 [S] A0 Write [P]
0000054113 [S] A0 Write 03 44 [P]
0000054113 [S] A0 Write NACK [P]
0000054113 [S] A0 Write NACK [P]
0000054113 [S] A0 Write NACK [P]
0000054113 [S] A0 Write NACK [P]
0000054114 [S] A0 Write NACK [P]
0000054114 [S] A0 Write NACK [P]
0000054114 [S] A0 Write NACK [P]
0000054114 [S] A0 Write NACK [P]
0000054114 [S] A0 Write NACK [P]
0000054114 [S] A0 Write NACK [P]
0000054114 [S] A0 Write NACK [P]
0000054115 [S] A0 Write NACK [P]
0000054115 [S] A0 Write NACK [P]
0000054115 [S] A0 Write [P]
0000054115 [S] A0 Write 04 55 [P]
0000054115 [S] A0 Write NACK [P]
0000054116 [S] A0 Write NACK [P]
0000054116 [S] A0 Write NACK [P]
```

Lines 1 to 64 | 26% | File Size: 7.98 KB (241 lines) [07/12/2006 00:33]

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX Data Extractor
// SMBus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

#define CWAV_API __stdcall
```

```
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned char All,
unsigned char Decimal, unsigned char Hex, unsigned char Binary, unsigned char Comma,
unsigned char Space, unsigned char Timestamps, unsigned long SMBData,unsigned long SMBClk);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//**************************************************
// Define the working buffer
//**************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = FALSE;
unsigned long Q_NumberOfBytes = 0;
unsigned long V_Timestamps = TRUE;
unsigned long M_SDA = 1;
unsigned long N_SCL = 2;


void DisplayHelp(void)
{
     fprintf(stdout,"\nSMBusExtractor [-?SDHICGAB] [-Q NumberOfBytes] [-V Timestamp] [-O
filename] [-M SMBDatMask] [-N SMBClkMask] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");
     fprintf(stdout,"    M  - SMBData signal (1=ch0, 128=ch7, ch0 default)\n");
     fprintf(stdout,"    N  - SMBClk signal (1=ch0, 128=ch7, ch1 default)\n");



}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//**************************************************
```

```
// Parse all of the command line options
//****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SDA = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'N':
                              N_SCL = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not
recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}


//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("DX Data Extractor\n");
     printf("SMBus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //*****************************************************
     // Open up a file to store extracted data into
     //*****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //*****************************************************
     // Start the DX Pod extracting the data we want
     //*****************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, A_All, D_DecimalTextValues, H_HexTextValues,
I_BinaryValues, C_CommaDelimited, G_SpaceDelimited, V_Timestamps, M_SDA, N_SCL) ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //*****************************************************
     // Loop and do something with the collected data
     //*****************************************************

     char OldSignal = 99;
     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }

               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //****************************************************
          // Give the OS a little time to do something else
          //****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);
     //****************************************************
     // Close the file
     //****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //****************************************************
     // Stop the extraction process
     //****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

SPI DATA EXTRACTOR

The SPI Bus Data Extractor takes the real-time streaming data from an SPI bus, formats it and allows you to save the data to disk or process it as it arrives.

SERIAL BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one SPI Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- SPI Clock speeds up to 12MHz
- Asynchronous (internal) sampling of 1MB/s to 24MB/s*
- Output to Binary File*
- Output to Text File*
- Output to Screen*
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to Slave Select, MOSI, and MISO. Connect the GND line to the digital ground of your system.

EXTRACTOR COMMAND LINE PROGRAM

The SPI Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- (“cd \program files\USBee Data Extractor\SPI”)
- Run the executable using the following command line arguments:

```
SPIExtractor [-?SWT] [-Q NumberOfBytes] [-R SampleRate] [-M
SlaveSelect] [-L CLK] [-V MOSI] [-J MISO] [-K MOSISample] [-U
MOSISample] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - Slave Select Signal (1=signal0,128=signal7)

L - Clk Signal (1=signal0,128=signal7)

V - MOSI Signal (1=signal0,128=signal7)

J - MISO Signal (1=signal0,128=signal7)

K - MOSI Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)

U - MISO Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)

W - Insert Slave Select Boundaries

T - Insert Time Stamps

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz (default)
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz

- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedSPI.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate,
unsigned long PodNumber, unsigned int ClockMode, unsigned char
SlaveSelect, unsigned char CLK, unsigned char MOSI, unsigned char
MISO, unsigned char MOSIEdge, unsigned char MISOEdge, unsigned char
SSInsert, unsigned char Timestamp );
```

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee DX Test Pod

ClockMode: 2 = Internal Timing as in SampleRate parameter

SlaveSelect: Which signal the extractor uses for Slave Select (1=channel0,128=channel7)

CLK: Which signal the extractor uses for CLK (1=channel0,128=channel7)

MOSI: Which signal the extractor uses for MOSI (1=channel0,128=channel7)

MISO: Which signal the extractor uses for MISO (1=channel0,128=channel7)

MOSIEdge: When the MOSI signal is sampled, 0=Falling CLK Edge, 1=Rising CLK Edge

MISOEdge: When the MISO signal is sampled, 0=Falling CLK Edge, 1=Rising CLK Edge

SSInsert: Set to 1 to insert Slave Select boundaries into the extracted data stream

Timestamp: Set to 1 to insert Time Stamps into the extracted data stream

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

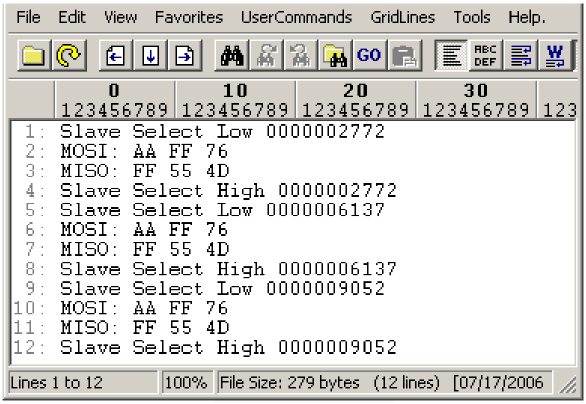
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The SPI Bus Extractor outputs MOSI and MISO values separated by newline characters with optional Slave Select and Timestamps inserted.

```
SPIExtractor -O output.dex -P 143 -Q 500000 -M 8 -L 1 -V 2 -J 4 -K 1
-U 0 -W -T
```


File Edit View Favorites UserCommands GridLines Tools Help.
GO
ABC DEF
0 10 20 30
123456789 123456789 123456789 123456789 123
Lines 1 to 12
100%
File Size: 279 bytes (12 lines) [07/17/2006

```
Slave Select Low 0000002772
MOSI: AA FF 76
MISO: FF 55 4D
Slave Select High 0000002772
Slave Select Low 0000006137
MOSI: AA FF 76
MISO: FF 55 4D
Slave Select High 0000006137
Slave Select Low 0000009052
MOSI: AA FF 76
MISO: FF 55 4D
Slave Select High 0000009052
```

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX Data Extractor
// SPI Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber,
unsigned int ClockMode,
                                                   unsigned char SlaveSelect, unsigned char
CLK, unsigned char MOSI,
                                                   unsigned char MISO, unsigned char
MOSIEdge, unsigned char MISOEdge,
                                                   unsigned char SSInsert, unsigned char
Timestamp );
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//****************************************************
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char _1_BytePerValue = TRUE;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned char M_SlaveSelect = 0;
unsigned char L_CLK = 0;
unsigned char V_MOSI = 0;
unsigned char J_MISO = 0;
unsigned char K_MOSIEdge = 0;
unsigned char U_MISOEdge = 0;
unsigned char W_SSInsert = 0;
unsigned char T_Timestamp = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 167;
unsigned long Q_NumberOfBytes = 0;

void DisplayHelp(void)
```

```
{
     fprintf(stdout,"\nSPIExtractor [-?SWT] [-Q NumberOfBytes] [-R SampleRate] [-M
SlaveSelect] [-L CLK] [-V MOSI] [-J MISO] [-K MOSISample] [-U MOSISample] [-O filename] -P
PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Signal Selection\n");
     fprintf(stdout,"    M  - Slave Select Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    L  - Clk Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    V  - MOSI Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    J  - MISO Signal (1=signal0,128=signal7)\n");
     fprintf(stdout,"    K  - MOSI Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)\n");
     fprintf(stdout,"    U  - MISO Sample Time (1=Rising CLK Edge,0=Falling CLK Edge)\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (16Msps default)\n");

     fprintf(stdout,"\n  Display Option\n");
     fprintf(stdout,"    W  - Insert Slave Select Boundaries\n");
     fprintf(stdout,"    T  - Insert Time Stamps\n");

     exit(0);
}
void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}

//**************************************************
// Parse all of the command line options
//**************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)      // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_SlaveSelect = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'L':
                              L_CLK = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_MOSI = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'J':
                              J_MISO = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'K':
                              K_MOSIEdge = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'U':
                              U_MISOEdge = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'W':
                              W_SSInsert = 1;
                              cont = FALSE;
                              break;
                         case 'T':
                              T_Timestamp = 1;
                              cont = FALSE;
                              break;
                         case 'E':
                              E_ExternalClockMode = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              Error("Invalid Command Line Switch");
                    }
          }
     }

     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}
unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{
     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);
}
//****************************************************
// Main Entry Point.  The program starts here.
//****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;

     printf("DX Data Extractor\n");
     printf("SPI Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //****************************************************
     // Open up a file to store extracted data into
     //****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //****************************************************
     // Start the DX Pod extracting the data we want
     //****************************************************

     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode, M_SlaveSelect,
L_CLK, V_MOSI, J_MISO, K_MOSIEdge, U_MISOEdge, W_SSInsert, T_Timestamp );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }
     printf("Processing and Saving Data to Disk.\n");

     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************
               StartTimer();

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }


               // StopTimer();

               if (timeout++ > 10  ) break;  // Let up once in a while to let the OS
process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //****************************************************
          // Give the OS a little time to do something else
          //****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //****************************************************
     // Close the file
     //****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //****************************************************
     // Stop the extraction process
     //****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;}
```

1-WIRE DATA EXTRACTOR

The 1-Wire Bus Data Extractor takes the real-time streaming data from an 1-Wire bus, formats it and allows you to save the data to disk or process it as it arrives.

1-WIRE BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one 1-Wire Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Asynchronous (internal) sampling from 1MB/s to 24MB/s*
- Output to Binary File*
- Output to Text File*
- Output to Screen*
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to the 1-Wire Signal. Connect the GND line to the digital ground of your system.

EXTRACTOR COMMAND LINE PROGRAM

The 1-Wire Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- (“cd \program files\USBee Data Extractor\1Wire”)
- Run the executable using the following command line arguments:

```
1WireExtractor [-?STW] [-Q NumberOfBytes] [-R SampleRate] [-M
Signal] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - 1 Wire Signal Mask (1=channel0,128=channel7)

W - Insert Reset/Presence Pulse

T - Insert Time Stamps

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz
- 27 = 2MHz
- 17 = 1MHz (default)

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
Usbed1Wire.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

CWAV_EXPORT int CWAV_API StartExtraction(unsigned int SampleRate, unsigned long PodNumber, unsigned int ClockMode, unsigned char Signal, unsigned char SSInsert, unsigned char Timestamp);

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee DX Test Pod

ClockMode: 2 = Internal Timing as in SampleRate parameter

Signal: Which signal the extractor uses for the 1-Wire Signal (1=channel0,128=channel7)

SSInsert: Set to 1 to insert Reset/Presence boundaries into the extracted data stream

Timestamp: Set to 1 to insert Time Stamps into the extracted data stream

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

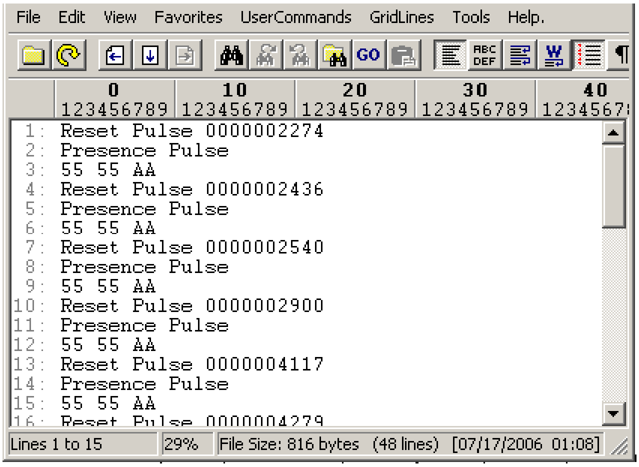
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The 1-Wire Bus Extractor outputs data values separated by newline characters with option Reset/Presence and Timestamps inserted.

```
1WireExtractor -O output.dex -P 143 -Q 500000 -M 1 -W -T -R 127
```



```
Reset Pulse 0000002274
Presence Pulse
55 55 AA
Reset Pulse 0000002436
Presence Pulse
55 55 AA
Reset Pulse 0000002540
Presence Pulse
55 55 AA
Reset Pulse 0000002900
Presence Pulse
55 55 AA
Reset Pulse 0000004117
Presence Pulse
55 55 AA
Reset Pulse 0000004279
```

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX Data Extractor
// 1 Wire Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber,
unsigned int ClockMode,
                                                    unsigned char Signal, unsigned char
SSInsert, unsigned char Timestamp );
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//****************************************************
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char _1_BytePerValue = TRUE;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned char M_Signal = 0;
unsigned char W_SSInsert = 0;
unsigned char T_Timestamp = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 167;
unsigned long Q_NumberOfBytes = 0;
// Not used yet W

void DisplayHelp(void)
{
     fprintf(stdout,"\n1WireExtractor [-?STW] [-Q NumberOfBytes] [-R SampleRate] [-M
Signal] [-O filename] -P PodID\n\n");
     fprintf(stdout,"    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");
     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");
     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Signal Selection\n");
     fprintf(stdout,"    M  - Signal (1=signal0,128=signal7)\n");

     fprintf(stdout,"\n  Clocking Modes\n");
     fprintf(stdout,"    R  - Internal CLK Sample Rate (16Msps default)\n");

     fprintf(stdout,"\n  Display Option\n");
     fprintf(stdout,"    W  - Insert Reset/Presence\n");
     fprintf(stdout,"    T  - Insert Time Stamps\n");

     exit(0);
}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)      // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_Signal = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'W':
                              W_SSInsert = 1;
                              cont = FALSE;
                              break;
                         case 'T':
                              T_Timestamp = 1;
                              cont = FALSE;
                              break;
                         case 'R':
                              R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              Error("Invalid Command Line Switch");
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}

unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{

     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);

}

//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;

     printf("DX Data Extractor\n");
     printf("1Wire Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //****************************************************
     // Open up a file to store extracted data into
     //****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //****************************************************
     // Start the DX Pod extracting the data we want
     //****************************************************

     RetValue = StartExtraction( R_SampleRate, P_PodID, E_ExternalClockMode, M_Signal,
W_SSInsert, T_Timestamp );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }

     printf("Processing and Saving Data to Disk.\n");

     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************
               StartTimer();

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }


               // StopTimer();

               if (timeout++ > 10  ) break;  // Let up once in a while to let the OS 
process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for 
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for 
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //****************************************************
          // Give the OS a little time to do something else
          //****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //****************************************************
     // Close the file
     //****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //****************************************************
     // Stop the extraction process
     //****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

I2S DATA EXTRACTOR

The I2S Bus Data Extractor takes the real-time streaming data from an I2S bus, formats it and allows you to save the data to disk or process it as it arrives.

I2S BUS DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one I2S Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- I2S Bit Clock up to 12MHz
- Supports I2S or Left Justified sample formats
- Supports MSBit first and non-standard LSBit first formats
- Asynchronous (internal) sampling from 1MB/s to 24MB/s*
- Output to Binary File*
- Output to Text File*
- Output to Screen*
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 Data Extractor uses any of the 8 signal lines (0 thru 7) and the GND (ground) line. Connect any of the 8 signals lines to Word Select, CLK, and Data. Connect the GND line to the digital ground of your system.

EXTRACTOR COMMAND LINE PROGRAM

The I2S Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- ("cd \program files\USBee Data Extractor\I2S")
- Run the executable using the following command line arguments:

```
I2SExtractor [-?ST1234JIYZ] [-Q NumberOfBytes] [-R SampleRate] [-M
WordSelect] [-L CLK] [-V Data] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

M - Word Select Signal (1=signal0,128=signal7)

L - Clk Signal (1=signal0,128=signal7)

V - Data Signal (1=signal0,128=signal7)

Y - Least significant bit first

Z - Most significant bit first

J - Left Justified (first rising edge after Word Select change is first bit)

I - I2S format (second rising edge after Word Select change is first bit)

T - Insert Word Select Boundaries

R - Internal CLK Sample Rate (16Msps default)

- 247 = 24MHz
- 167 = 16MHz (default)
- 127 = 12MHz
- 87 = 8MHz
- 67 = 6MHz
- 47 = 4MHz
- 37 = 3MHz

- 27 = 2MHz
- 17 = 1MHz

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedI2S.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction( unsigned int SampleRate,
unsigned long PodNumber, unsigned int ClockMode, unsigned char
WordSelect, unsigned char CLK, unsigned char Data, unsigned char
SSInsert, unsigned char BytesPerValue, unsigned char I2SMode,
unsigned char MSBFirstMode );
```

SampleRate:

- 17 = 1Msps
- 27 = 2Msps
- 37 = 3Msps
- 47 = 4Msps
- 67 = 6Msps
- 87 = 8Msps
- 127 = 12Msps
- 167 = 16Msps
- 247 = 24Msps

PodNumber: Pod ID on the back of the USBee DX Test Pod

ClockMode: 2 = Internal Timing as in SampleRate parameter

WordSelect: Which signal the extractor uses for Word Select (1=channel0,128=channel7)

CLK: Which signal the extractor uses for CLK (1=channel0,128=channel7)

Data: Which signal the extractor uses for Data (1=channel0,128=channel7)

SSInsert: Set to 1 to insert Word Select boundaries into the extracted data stream

BytesPerValue: 1, 2, 3, or 4 bytes per value. Allows capture of 8, 16, 24, or 32 bits of audio data

I2SMode: Set to 1 for I2S data format. Set to 0 for Left Justified data format.

MSBFirstMode: Bit order (1 = MSBit first on the wire, 0 = LSBit first on the wire)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

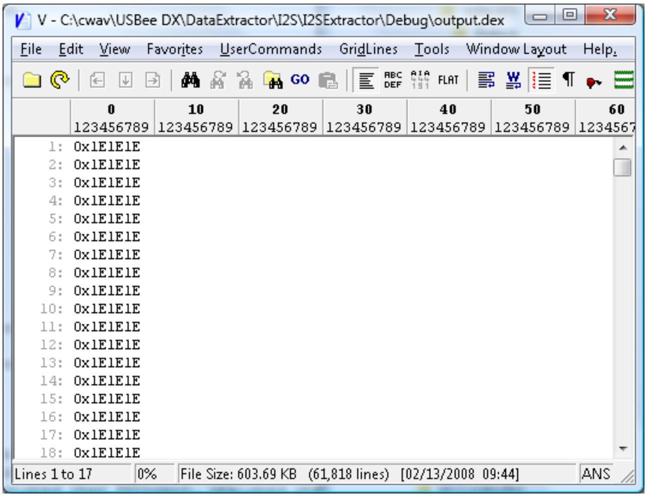
Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

```
I2SExtractor -O output.dex -P 123 -M 1 -L 2 -V 4 -3 -I
```



```
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
0x1E1E1E
```

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX-Pro Data Extractor
// I2S Bus Extractor Example Program
// Copyright 2008, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

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

```
CWAV_IMPORT int CWAV_API StartExtraction( unsigned int SampleRate, unsigned long PodNumber, 
unsigned int ClockMode,
                      unsigned char WordSelect, unsigned char CLK, unsigned char Data,
                      unsigned char SSInsert, unsigned char BytesPerValue,
                      unsigned char I2SMode, unsigned char MSBFirstMode );
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//****************************************************
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char BytePerValue = 1;
unsigned char _2_BytePerValue = FALSE;
unsigned char _4_BytePerValue = FALSE;
unsigned char Y_LeastSignificantByteFirst = FALSE;
unsigned char Z_MostSignificantByteFirst = TRUE;
unsigned char A_ASCIITextValues = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char B_BinaryTextValues = FALSE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned char N_NewlineDelimited = FALSE;
unsigned char X_NoDelimeter = FALSE;
unsigned long T_ForceBytesPerLine = 0;
unsigned char M_WordSelect = 0;
unsigned char L_CLK = 0;
unsigned char V_Data = 0;
unsigned char J_LeftJustifiedMode = FALSE;
unsigned char I_I2SMode = TRUE;
unsigned char K_DataEdge = 0;
unsigned char U_MISOEdge = 0;
unsigned char T_SSInsert = 0;
unsigned char E_ExternalClockMode = 2;
unsigned char R_SampleRate = 167;
unsigned long Q_NumberOfBytes = 0;

void DisplayHelp(void)
{
fprintf(stdout,"\nI2SExtractor [-?ST1234JIZY] [-Q NumberOfBytes] [-R SampleRate] [-M 
WordSelect] [-L CLK] [-V Data] [-O filename] -P PodID\n\n");
fprintf(stdout,"    ?  - Display this help screen\n");

fprintf(stdout,"\n  USBee DX-Pro Pod to Use\n");
fprintf(stdout,"    P  - Pod ID (required)\n");

fprintf(stdout,"\n  Output Location Flags\n");
fprintf(stdout,"    O  - Output to filename (default off)\n");
fprintf(stdout,"    S  - Output to the screen (default off)\n");

fprintf(stdout,"\n  When to Quit Flags\n");
fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

fprintf(stdout,"\n  Signal Selection\n");
fprintf(stdout,"    M  - Word Select Signal (1=signal0,128=signal7)\n");
fprintf(stdout,"    L  - Clk Signal (1=signal0,128=signal7)\n");
fprintf(stdout,"    V  - Data Signal (1=signal0,128=signal7)\n");

fprintf(stdout,"\n  Number of Bytes to Capture per channel\n");
fprintf(stdout,"    1  - One   Byte  per value (default)\n");
fprintf(stdout,"    2  - Two   Bytes per value\n");
fprintf(stdout,"    3  - Three Bytes per value\n");
fprintf(stdout,"    4  - Four  Bytes per value\n");

fprintf(stdout,"\n  Data Mode\n");
fprintf(stdout,"    I  - I2S Mode (data starts on second clock) (default)\n");
fprintf(stdout,"    J  - Left Justified (data starts on first clock)\n");

fprintf(stdout,"\n  Input Bit Order\n");
fprintf(stdout,"    Y  - Least Significant Bit First\n");
fprintf(stdout,"    Z  - Most Significant Bit First (default)\n");

fprintf(stdout,"\n  Clocking Modes\n");
fprintf(stdout,"    R  - Internal CLK Sample Rate (16Msps default)\n");

fprintf(stdout,"\n  Display Option\n");
fprintf(stdout,"    T  - Insert Word Select Boundaries\n");

exit(0);
}

void Error(char *err)
{
fprintf(stderr,"Error: ");
fprintf(stderr,"%s\n",err);
exit(2);
}


//*****************************************************
// Parse all of the command line options
//*****************************************************
void ParseCommandLine(int argc, char *argv[])
{
BOOL cont;
int       i,j;
DWORD WordExample;
BYTE ByteExample;

for(i=1; i < argc; ++i)
{
     if((argv[i][0] == '-') || (argv[i][0] == '/'))
     {
          cont = TRUE;
          for(j=1;argv[i][j] && cont;++j) // Cont flag permits multiple in a single argv
(like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case '1':
                              BytePerValue = 1;
                              break;
                         case '2':
                              BytePerValue = 2;
                              break;
                         case '3':
                              BytePerValue = 3;
                              break;
                         case '4':
                              BytePerValue = 4;
                              break;
                         case 'M':
                              M_WordSelect = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'Y':
                              Y_LeastSignificantByteFirst = TRUE;
                              Z_MostSignificantByteFirst = FALSE;
                              break;
                         case 'Z':
                              Z_MostSignificantByteFirst = TRUE;
                              Y_LeastSignificantByteFirst = FALSE;
                              break;
                         case 'L':
                              L_CLK = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Data = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'J':
                              J_LeftJustifiedMode = TRUE;
                              I_I2SMode = FALSE;
                              break;
                         case 'I':
                              J_LeftJustifiedMode = FALSE;
                              I_I2SMode = TRUE;
                              break;
                         case 'K':
                              K_DataEdge = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'U':
                              U_MISOEdge = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'T':
                              T_SSInsert = 1;
                              cont = FALSE;
                              break;
                         case 'E':
                              E_ExternalClockMode = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_SampleRate = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              Error("Invalid Command Line Switch");
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}

unsigned long StartTime;

void StartTimer()
{

     StartTime = GetTickCount();
}

void StopTimer()
{

     printf(" \nTime Delta = %d\n",GetTickCount() - StartTime);

}

//*****************************************************
// Main Entry Point.  The program starts here.
//*****************************************************
```

```
int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;

     printf("USBee DX Data Extractor\n");
     printf("I2S Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //****************************************************
     // Open up a file to store extracted data into
     //****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //****************************************************
     // Start the USBee DX Pod extracting the data we want
     //****************************************************

     RetValue = StartExtraction(   R_SampleRate, P_PodID, E_ExternalClockMode,
M_WordSelect, L_CLK, V_Data,
                    T_SSInsert, BytePerValue, I_I2SMode, Z_MostSignificantByteFirst );

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }

     printf("Processing and Saving Data to Disk.\n");

     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************
               StartTimer();

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file
               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }


               // StopTimer();

               if (timeout++ > 10  ) break;  // Let up once in a while to let the OS 
process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for 
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for 
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //****************************************************
          // Give the OS a little time to do something else
          //****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //****************************************************
     // Close the file
     //****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //****************************************************
     // Stop the extraction process
     //****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

LOW AND FULL SPEED USB DATA EXTRACTOR

The USB Data Extractor takes the real-time streaming data from the Full or Low Speed bus, formats it and allows you to save the data to disk or process it as it arrives.

USB DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- One USB Bus running at Low (1.5Mbps) or Full Speed (12Mbps) USB (not High Speed)
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V)
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma, Space, or Newline Delimited files
- Packet filter on Device Address, and/or Endpoint
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 USB Bus Data Extractor uses signal 0 and signal 1 as the DPlus and DMinus lines of the USB bus. Connect these signals to the USB bus using the test clips provided. Connect the GND line to the digital ground of your system.

EXTRACTOR COMMAND LINE PROGRAM

The USB Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- (“cd \program files\USBee Data Extractor\USB”)
- Run the executable using the following command line arguments:

```
Usbedtractor [-?SDHICGAB] [-R USBSpeed] [-Q NumberOfBytes] [-V
Timestamp] [-O filename] -P PodID
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

R - Bus Speed (0=Low Speed USB, 1=Full Speed USB)

A - All Packet Fields are output (default)

B – Only Data Bytes are output

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

V - Timestamps (0=off, 1=each packet start)

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedUSB.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned char Speed, unsigned char All, unsigned char Decimal,
unsigned char Hex, unsigned char Binary, unsigned char Comma,
unsigned char Space, unsigned char Timestamps)
```

PodNumber: Pod ID on the back of the USBee DX Test Pod

Speed:

- 0 = Low Speed
- 1 = Full Speed

All:

- 0 – Only the data payload bytes are returned
- 1 – All USB packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

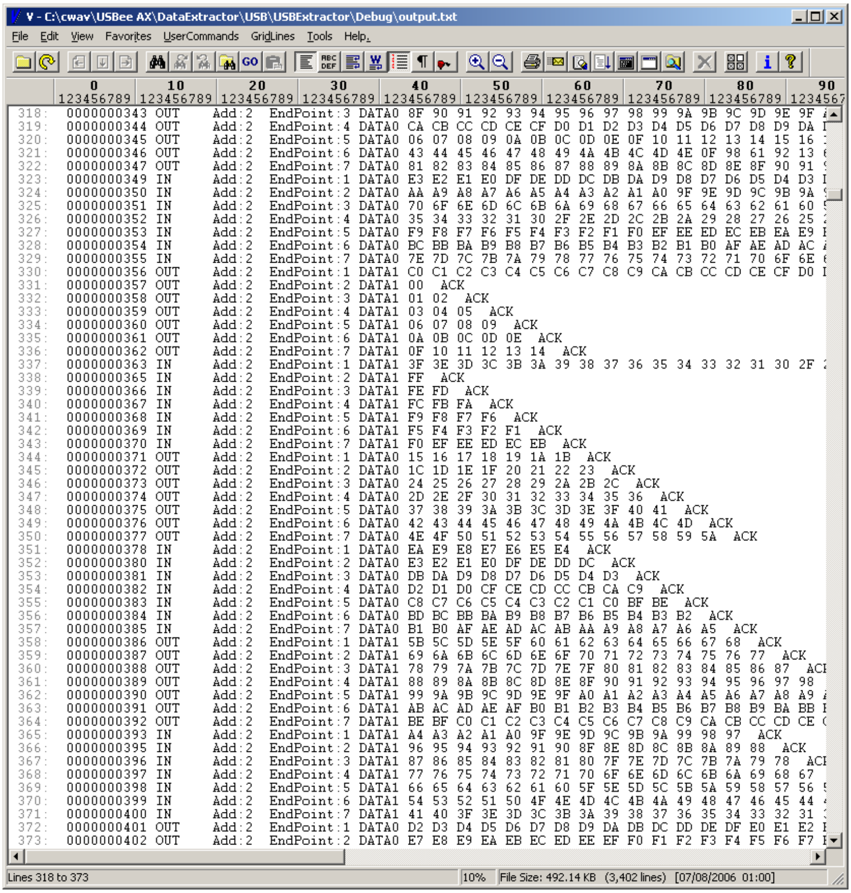
Return:

- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The USB Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the USB traffic in Hex text form, separated by any specified delimiters.

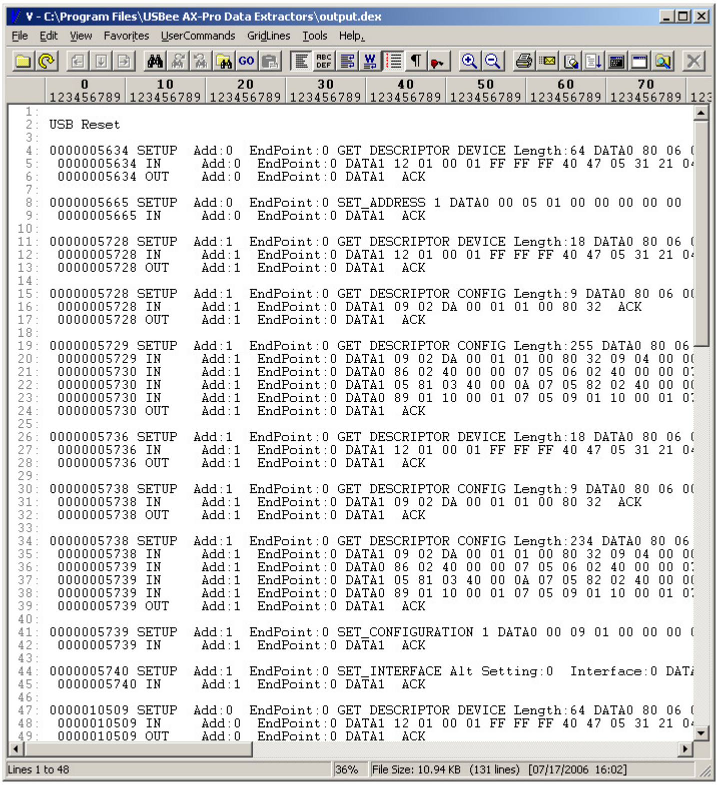


V - C:\cwav\USBee AX\DataExtractor\USB\USBExtractor\Debug\output.txt

```
0000000343 OUT    Add:2  EndPoint:3 DATA0 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
0000000344 OUT    Add:2  EndPoint:4 DATA0 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA
0000000345 OUT    Add:2  EndPoint:5 DATA0 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16
0000000346 OUT    Add:2  EndPoint:6 DATA0 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 0F 98 61 92 13
0000000347 OUT    Add:2  EndPoint:7 DATA0 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91
0000000349 IN     Add:2  EndPoint:1 DATA0 E3 E2 E1 E0 DF DE DD DC DB DA D9 D8 D7 D6 D5 D4 D3
0000000350 IN     Add:2  EndPoint:2 DATA0 AA A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 9F 9E 9D 9C 9B 9A
0000000351 IN     Add:2  EndPoint:3 DATA0 70 6F 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60
0000000352 IN     Add:2  EndPoint:4 DATA0 35 34 33 32 31 30 2F 2E 2D 2C 2B 2A 29 28 27 26 25
0000000353 IN     Add:2  EndPoint:5 DATA0 F9 F8 F7 F6 F5 F4 F3 F2 F1 F0 EF EE ED EC EB EA E9
0000000354 IN     Add:2  EndPoint:6 DATA0 BC BB BA B9 B8 B7 B6 B5 B4 B3 B2 B1 B0 AF AE AD AC
0000000355 IN     Add:2  EndPoint:7 DATA0 7E 7D 7C 7B 7A 79 78 77 76 75 74 73 72 71 70 6F 6E
0000000356 OUT    Add:2  EndPoint:1 DATA1 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0
0000000357 OUT    Add:2  EndPoint:2 DATA1 00  ACK
0000000358 OUT    Add:2  EndPoint:3 DATA1 01 02  ACK
0000000359 OUT    Add:2  EndPoint:4 DATA1 03 04 05  ACK
0000000360 OUT    Add:2  EndPoint:5 DATA1 06 07 08 09  ACK
0000000361 OUT    Add:2  EndPoint:6 DATA1 0A 0B 0C 0D 0E  ACK
0000000362 OUT    Add:2  EndPoint:7 DATA1 0F 10 11 12 13 14  ACK
0000000363 IN     Add:2  EndPoint:1 DATA1 3F 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32 31 30 2F
0000000365 IN     Add:2  EndPoint:2 DATA1 FF  ACK
0000000366 IN     Add:2  EndPoint:3 DATA1 FE FD  ACK
0000000367 IN     Add:2  EndPoint:4 DATA1 FC FB FA  ACK
0000000368 IN     Add:2  EndPoint:5 DATA1 F9 F8 F7 F6  ACK
0000000369 IN     Add:2  EndPoint:6 DATA1 F5 F4 F3 F2 F1  ACK
0000000370 IN     Add:2  EndPoint:7 DATA1 F0 EF EE ED EC EB  ACK
0000000371 OUT    Add:2  EndPoint:1 DATA0 15 16 17 18 19 1A 1B  ACK
0000000372 OUT    Add:2  EndPoint:2 DATA0 1C 1D 1E 1F 20 21 22 23  ACK
0000000373 OUT    Add:2  EndPoint:3 DATA0 24 25 26 27 28 29 2A 2B 2C  ACK
0000000374 OUT    Add:2  EndPoint:4 DATA0 2D 2E 2F 30 31 32 33 34 35 36  ACK
0000000375 OUT    Add:2  EndPoint:5 DATA0 37 38 39 3A 3B 3C 3D 3E 3F 40 41  ACK
0000000376 OUT    Add:2  EndPoint:6 DATA0 42 43 44 45 46 47 48 49 4A 4B 4C 4D  ACK
0000000377 OUT    Add:2  EndPoint:7 DATA0 4E 4F 50 51 52 53 54 55 56 57 58 59 5A  ACK
0000000378 IN     Add:2  EndPoint:1 DATA0 EA E9 E8 E7 E6 E5 E4  ACK
0000000380 IN     Add:2  EndPoint:2 DATA0 E3 E2 E1 E0 DF DE DD DC  ACK
0000000381 IN     Add:2  EndPoint:3 DATA0 DB DA D9 D8 D7 D6 D5 D4 D3  ACK
0000000382 IN     Add:2  EndPoint:4 DATA0 D2 D1 D0 CF CE CD CC CB CA C9  ACK
0000000383 IN     Add:2  EndPoint:5 DATA0 C8 C7 C6 C5 C4 C3 C2 C1 C0 BF BE  ACK
0000000384 IN     Add:2  EndPoint:6 DATA0 BD BC BB BA B9 B8 B7 B6 B5 B4 B3 B2  ACK
0000000385 IN     Add:2  EndPoint:7 DATA0 B1 B0 AF AE AD AC AB AA A9 A8 A7 A6 A5  ACK
0000000386 OUT    Add:2  EndPoint:1 DATA1 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68  ACK
0000000387 OUT    Add:2  EndPoint:2 DATA1 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77  ACK
0000000388 OUT    Add:2  EndPoint:3 DATA1 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87  AC
0000000389 OUT    Add:2  EndPoint:4 DATA1 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98
0000000390 OUT    Add:2  EndPoint:5 DATA1 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9
0000000391 OUT    Add:2  EndPoint:6 DATA1 AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB
0000000392 OUT    Add:2  EndPoint:7 DATA1 BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE
0000000393 IN     Add:2  EndPoint:1 DATA1 A4 A3 A2 A1 A0 9F 9E 9D 9C 9B 9A 99 98 97  ACK
0000000395 IN     Add:2  EndPoint:2 DATA1 96 95 94 93 92 91 90 8F 8E 8D 8C 8B 8A 89 88  ACK
0000000396 IN     Add:2  EndPoint:3 DATA1 87 86 85 84 83 82 81 80 7F 7E 7D 7C 7B 7A 79 78  AC
0000000397 IN     Add:2  EndPoint:4 DATA1 77 76 75 74 73 72 71 70 6F 6E 6D 6C 6B 6A 69 68 67
0000000398 IN     Add:2  EndPoint:5 DATA1 66 65 64 63 62 61 60 5F 5E 5D 5C 5B 5A 59 58 57 56
0000000399 IN     Add:2  EndPoint:6 DATA1 54 53 52 51 50 4F 4E 4D 4C 4B 4A 49 48 47 46 45 44
0000000400 IN     Add:2  EndPoint:7 DATA1 41 40 3F 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32 31
0000000401 OUT    Add:2  EndPoint:1 DATA0 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2
0000000402 OUT    Add:2  EndPoint:2 DATA0 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7
```

Lines 318 to 373 | 10% | File Size: 492.14 KB (3,402 lines) [07/08/2006 01:00]

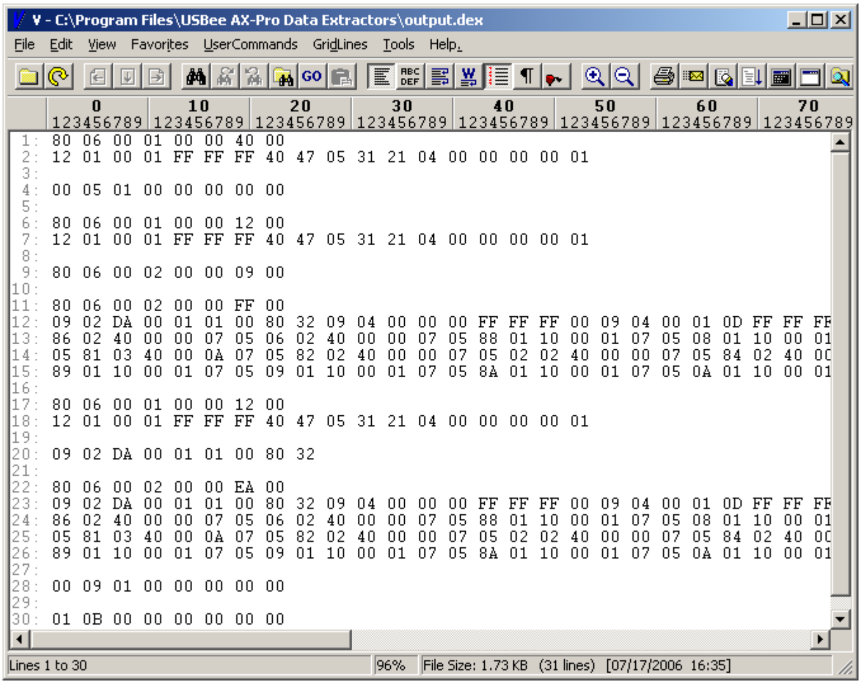
```
Usbedtractor  -O output.dex -P 3209 -G -Q 10000 -R 1 -A -H -V 1
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help
GO
0 10 20 30 40 50 60 70
123456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789 123

```

USB Reset

0000005634 SETUP  Add:0  EndPoint:0 GET DESCRIPTOR DEVICE Length:64 DATA0 80 06
 0000005634 IN     Add:0  EndPoint:0 DATA1 12 01 00 01 FF FF FF 40 47 05 31 21
 0000005634 OUT    Add:0  EndPoint:0 DATA1  ACK

0000005665 SETUP  Add:0  EndPoint:0 SET_ADDRESS 1 DATA0 00 05 01 00 00 00 00 00
 0000005665 IN     Add:0  EndPoint:0 DATA1  ACK

0000005728 SETUP  Add:1  EndPoint:0 GET DESCRIPTOR DEVICE Length:18 DATA0 80 06
 0000005728 IN     Add:1  EndPoint:0 DATA1 12 01 00 01 FF FF FF 40 47 05 31 21
 0000005728 OUT    Add:1  EndPoint:0 DATA1  ACK

0000005728 SETUP  Add:1  EndPoint:0 GET DESCRIPTOR CONFIG Length:9 DATA0 80 06
 0000005728 IN     Add:1  EndPoint:0 DATA1 09 02 DA 00 01 01 00 80 32  ACK
 0000005728 OUT    Add:1  EndPoint:0 DATA1  ACK

0000005729 SETUP  Add:1  EndPoint:0 GET DESCRIPTOR CONFIG Length:255 DATA0 80 06
 0000005729 IN     Add:1  EndPoint:0 DATA1 09 02 DA 00 01 01 00 80 32 09 04 00
 0000005730 IN     Add:1  EndPoint:0 DATA0 86 02 40 00 00 07 05 06 02 40 00 00
 0000005730 IN     Add:1  EndPoint:0 DATA1 05 81 03 40 00 0A 07 05 82 02 40 00
 0000005730 IN     Add:1  EndPoint:0 DATA0 89 01 10 00 01 07 05 09 01 10 00 01
 0000005730 OUT    Add:1  EndPoint:0 DATA1  ACK

0000005736 SETUP  Add:1  EndPoint:0 GET DESCRIPTOR DEVICE Length:18 DATA0 80 06
 0000005736 IN     Add:1  EndPoint:0 DATA1 12 01 00 01 FF FF FF 40 47 05 31 21
 0000005736 OUT    Add:1  EndPoint:0 DATA1  ACK

0000005738 SETUP  Add:1  EndPoint:0 GET DESCRIPTOR CONFIG Length:9 DATA0 80 06
 0000005738 IN     Add:1  EndPoint:0 DATA1 09 02 DA 00 01 01 00 80 32  ACK
 0000005738 OUT    Add:1  EndPoint:0 DATA1  ACK

0000005738 SETUP  Add:1  EndPoint:0 GET DESCRIPTOR CONFIG Length:234 DATA0 80 06
 0000005738 IN     Add:1  EndPoint:0 DATA1 09 02 DA 00 01 01 00 80 32 09 04 00
 0000005739 IN     Add:1  EndPoint:0 DATA0 86 02 40 00 00 07 05 06 02 40 00 00
 0000005739 IN     Add:1  EndPoint:0 DATA1 05 81 03 40 00 0A 07 05 82 02 40 00
 0000005739 IN     Add:1  EndPoint:0 DATA0 89 01 10 00 01 07 05 09 01 10 00 01
 0000005739 OUT    Add:1  EndPoint:0 DATA1  ACK

0000005739 SETUP  Add:1  EndPoint:0 SET_CONFIGURATION 1 DATA0 00 09 01 00 00 00
 0000005739 IN     Add:1  EndPoint:0 DATA1  ACK

0000005740 SETUP  Add:1  EndPoint:0 SET_INTERFACE Alt Setting:0  Interface:0 DAT
 0000005740 IN     Add:1  EndPoint:0 DATA1  ACK

0000010509 SETUP  Add:0  EndPoint:0 GET DESCRIPTOR DEVICE Length:64 DATA0 80 06
 0000010509 IN     Add:0  EndPoint:0 DATA1 12 01 00 01 FF FF FF 40 47 05 31 21
 0000010509 OUT    Add:0  EndPoint:0 DATA1  ACK
```

Lines 1 to 48 | 36% | File Size: 10.94 KB (131 lines) [07/17/2006 16:02]

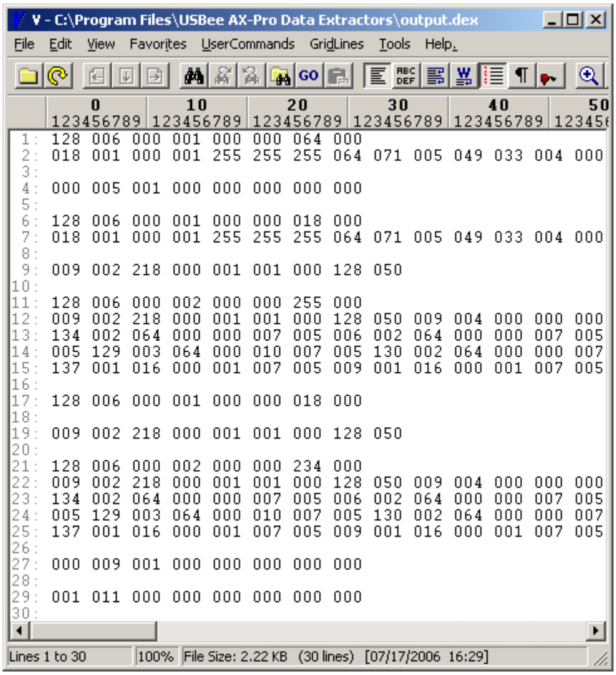
```
Usbedtractor  -O output.dex -P 3209 -G -Q 10000 -R 1 -B
```



```
80 06 00 01 00 00 40 00
12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01

00 05 01 00 00 00 00 00

80 06 00 01 00 00 12 00
12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01

80 06 00 02 00 00 09 00

80 06 00 02 00 00 FF 00
09 02 DA 00 01 01 00 80 32 09 04 00 00 00 FF FF FF 00 09 04 00 01 0D FF FF FF
86 02 40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00 01 07 05 08 01 10 00 01
05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07 05 84 02 40 00
89 01 10 00 01 07 05 09 01 10 00 01 07 05 8A 01 10 00 01 07 05 0A 01 10 00 01

80 06 00 01 00 00 12 00
12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01

09 02 DA 00 01 01 00 80 32

80 06 00 02 00 00 EA 00
09 02 DA 00 01 01 00 80 32 09 04 00 00 00 FF FF FF 00 09 04 00 01 0D FF FF FF
86 02 40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00 01 07 05 08 01 10 00 01
05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07 05 84 02 40 00
89 01 10 00 01 07 05 09 01 10 00 01 07 05 8A 01 10 00 01 07 05 0A 01 10 00 01

00 09 01 00 00 00 00 00

01 0B 00 00 00 00 00 00
```

Lines 1 to 30 | 96% | File Size: 1.73 KB (31 lines) [07/17/2006 16:35]

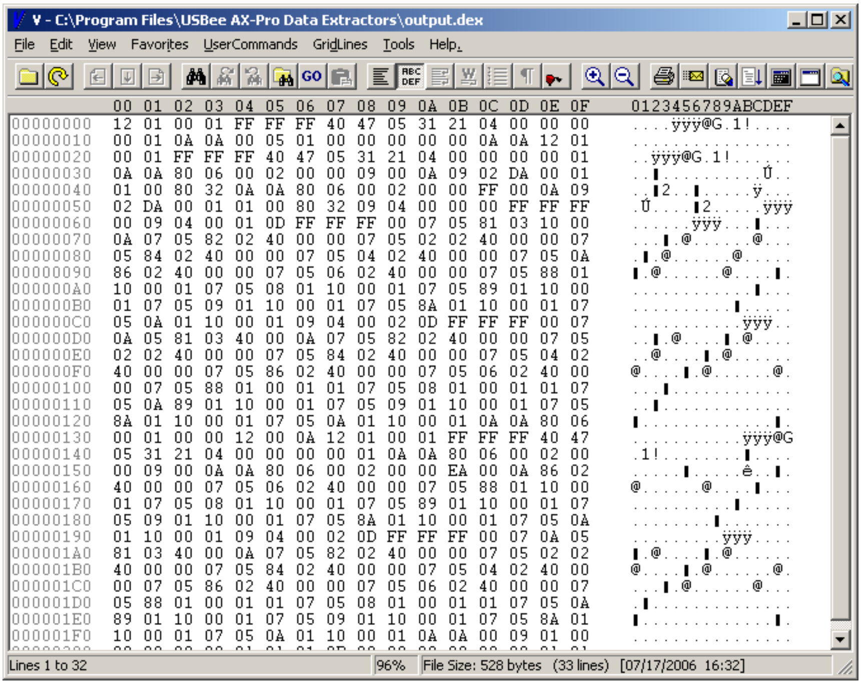
```
Usbedtractor  -O output.dex -P 3209 -G -Q 10000 -R 1 -A -H -V 1 -B -
D
```


V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help
GO
0 10 20 30 40 50
123456789 123456789 123456789 123456789 123456789 12345

```
128 006 000 001 000 000 064 000
018 001 000 001 255 255 255 064 071 005 049 033 004 000

000 005 001 000 000 000 000 000

128 006 000 001 000 000 018 000
018 001 000 001 255 255 255 064 071 005 049 033 004 000

009 002 218 000 001 001 000 128 050

128 006 000 002 000 000 255 000
009 002 218 000 001 001 000 128 050 009 004 000 000 000
134 002 064 000 000 007 005 006 002 064 000 000 007 005
005 129 003 064 000 010 007 005 130 002 064 000 000 007
137 001 016 000 001 007 005 009 001 016 000 001 007 005

128 006 000 001 000 000 018 000

009 002 218 000 001 001 000 128 050

128 006 000 002 000 000 234 000
009 002 218 000 001 001 000 128 050 009 004 000 000 000
134 002 064 000 000 007 005 006 002 064 000 000 007 005
005 129 003 064 000 010 007 005 130 002 064 000 000 007
137 001 016 000 001 007 005 009 001 016 000 001 007 005

000 009 001 000 000 000 000 000

001 011 000 000 000 000 000 000

```

Lines 1 to 30 | 100% | File Size: 2.22 KB (30 lines) [07/17/2006 16:29]

```
Usbedtractor  -O output.dex -P 3209 -G -Q 10000 -R 1 -B -I
```



V - C:\Program Files\USBee AX-Pro Data Extractors\output.dex
File Edit View Favorites UserCommands GridLines Tools Help
GO

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F    0123456789ABCDEF
00000000  12 01 00 01 FF FF FF 40 47 05 31 21 04 00 00 00    ....ÿÿÿ@G.1!....
00000010  00 01 0A 0A 00 05 01 00 00 00 00 00 0A 0A 12 01    ................
00000020  00 01 FF FF FF 40 47 05 31 21 04 00 00 00 00 01    ..ÿÿÿ@G.1!......
00000030  0A 0A 80 06 00 02 00 00 09 00 0A 09 02 DA 00 01    ..▌..........Û..
00000040  01 00 80 32 0A 0A 80 06 00 02 00 00 FF 00 0A 09    ..▌2..▌.....ÿ...
00000050  02 DA 00 01 01 00 80 32 09 04 00 00 00 FF FF FF    .Û....▌2.....ÿÿÿ
00000060  00 09 04 00 01 0D FF FF FF 00 07 05 81 03 10 00    ......ÿÿÿ...▌...
00000070  0A 07 05 82 02 40 00 00 07 05 02 02 40 00 00 07    ...▌.@......@...
00000080  05 84 02 40 00 00 07 05 04 02 40 00 00 07 05 0A    .▌.@......@.....
00000090  86 02 40 00 00 07 05 06 02 40 00 00 07 05 88 01    ▌.@......@....▌.
000000A0  10 00 01 07 05 08 01 10 00 01 07 05 89 01 10 00    ............▌...
000000B0  01 07 05 09 01 10 00 01 07 05 8A 01 10 00 01 07    ..........▌.....
000000C0  05 0A 01 10 00 01 09 04 00 02 0D FF FF FF 00 07    ...........ÿÿÿ..
000000D0  0A 05 81 03 40 00 0A 07 05 82 02 40 00 00 07 05    ..▌.@....▌.@....
000000E0  02 02 40 00 00 07 05 84 02 40 00 00 07 05 04 02    ..@....▌.@......
000000F0  40 00 00 07 05 86 02 40 00 00 07 05 06 02 40 00    @....▌.@......@.
00000100  00 07 05 88 01 00 01 01 07 05 08 01 00 01 01 07    ...▌............
00000110  05 0A 89 01 10 00 01 07 05 09 01 10 00 01 07 05    ..▌.............
00000120  8A 01 10 00 01 07 05 0A 01 10 00 01 0A 0A 80 06    ▌.............▌.
00000130  00 01 00 00 12 00 0A 12 01 00 01 FF FF FF 40 47    ...........ÿÿÿ@G
00000140  05 31 21 04 00 00 00 00 01 0A 0A 80 06 00 02 00    .1!.........▌...
00000150  00 09 00 0A 0A 80 06 00 02 00 00 EA 00 0A 86 02    .....▌.....ê..▌.
00000160  40 00 00 07 05 06 02 40 00 00 07 05 88 01 10 00    @......@....▌...
00000170  01 07 05 08 01 10 00 01 07 05 89 01 10 00 01 07    ..........▌.....
00000180  05 09 01 10 00 01 07 05 8A 01 10 00 01 07 05 0A    ........▌.......
00000190  01 10 00 01 09 04 00 02 0D FF FF FF 00 07 0A 05    .........ÿÿÿ....
000001A0  81 03 40 00 0A 07 05 82 02 40 00 00 07 05 02 02    ▌.@....▌.@......
000001B0  40 00 00 07 05 84 02 40 00 00 07 05 04 02 40 00    @....▌.@......@.
000001C0  00 07 05 86 02 40 00 00 07 05 06 02 40 00 00 07    ...▌.@......@...
000001D0  05 88 01 00 01 01 07 05 08 01 00 01 01 07 05 0A    .▌..............
000001E0  89 01 10 00 01 07 05 09 01 10 00 01 07 05 8A 01    ▌.............▌.
000001F0  10 00 01 07 05 0A 01 10 00 01 0A 0A 00 09 01 00    ................
```

Lines 1 to 32 | 96% | File Size: 528 bytes (33 lines) [07/17/2006 16:32]

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX Data Extractor
// USB Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned char Speed,
unsigned char All, unsigned char Decimal, unsigned char Hex, unsigned char Binary, unsigned
char Comma, unsigned char Space, unsigned char Timestamps, unsigned int Endpoint, unsigned
int Device)    ;
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//****************************************************
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = TRUE;
unsigned long Q_NumberOfBytes = 0;
unsigned long R_Speed = 1;                   // Full Speed
unsigned long V_Timestamps = TRUE;


void DisplayHelp(void)
{
     fprintf(stdout,"\nUsbedtractor [-?SDHICGAB] [-R USBSpeed] [-Q NumberOfBytes] [-V
Timestamp] [-O filename] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");
     fprintf(stdout,"\n   When to Quit Flags\n");

     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n   Input Format Flags\n");

     fprintf(stdout,"    R  - Bus Speed (0=Low Speed USB, 1=Full Speed USB)\n");

     fprintf(stdout,"\n   Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");



}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//****************************************************
// Parse all of the command line options
//****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);

                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_Speed = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not 
recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}


//****************************************************
// Main Entry Point.  The program starts here.
//****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("DX Data Extractor\n");
     printf("USB Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //****************************************************
     // Open up a file to store extracted data into
     //****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //****************************************************
     // Start the DX Pod extracting the data we want
     //****************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, R_Speed, A_All, D_DecimalTextValues,
H_HexTextValues, I_BinaryValues, C_CommaDelimited, G_SpaceDelimited, V_Timestamps,
Endpoint, Device)   ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }

               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //****************************************************
          // Give the OS a little time to do something else
          //****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //****************************************************
     // Close the file
     //****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //****************************************************
     // Stop the extraction process
     //****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

CAN DATA EXTRACTOR

The CAN Bus Data Extractor takes the real-time streaming data from the CAN bus, formats it and allows you to save the data to disk or process it as it arrives.

CAN DATA EXTRACTOR SPECIFICATIONS

- Continuous Real-Time Data Streaming
- Monitors one CAN Bus
- TTL Level inputs (**0-5V max**, Vih = 2.0V, Vil = 0.8V) – intended to be used on the digital side of a CAN bus transceiver (such as the Microchip MCP2551)
- 11 or 29-bit identifier supported
- Time Stamp for each packet
- Output to Text File*
- Output to Screen*
- Comma or Space Delimited files
- Packet filter on Identifier
- Output File Viewer (including binary, text, search and export functions)
- Extractor API libraries interface directly to your own software to further process the extracted data. Any language that supports calls to DLLs is supported.

* - output bandwidths are dependent on PC USB hardware, hard disk and/or screen throughput.

HARDWARE SETUP

To use the Data Extractor 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 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 Data Extractor 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.

EXTRACTOR COMMAND LINE PROGRAM

The CAN Bus Data Extractor includes a Windows Command Prompt executable that lets you operate the Data Extractor without writing any software. The program is executed in a Command Prompt window and is configured using command line arguments. The extracted data is then stored to disk or outputted to the screen depending on these parameters.

To run the Data Extractor:

- Install the USBee DX software on your PC
- Install the Data Extractor software on your PC
- Plug in your USBee DX Test Pod into your PC using a USB 2.0 High Speed Port
- Open a Windows Command Prompt window by clicking Start, All Programs, Accessories, Command Prompt.
- Change the working directory to the Data Extractor directory
- (“cd \program files\USBee Data Extractor\CAN”)
- Run the executable using the following command line arguments:

```
CANExtractor [-?SDHICGAB] [-R CANSpeed] [-Q NumberOfBytes] [-V
Timestamp] [-O filename] [-M MaxID] [-N MinID] -P
```

? - Display this help screen

P - Pod ID (required)

O - Output to filename (default off)

S - Output to the screen (default off)

Q - Number of output values (default = until keypress)

R - Bus Speed in bits/second (default = 250000)

A - All Packet Fields are output (default)

B – Only Data Bytes are output

D - Decimal Text Values ("49")

H - Hex Text Values ("31") default

I - Binary Values (49)

C - Comma Delimited

G - Space Delimited (default)

M - Maximum Identifier Filter

N - Minimum Identifier Filter

V - Timestamps (0=off, 1=each packet start)

EXTRACTOR API

The Data Extractor is implemented using a Windows DLL that interfaces to the existing USBee DX DLL and drivers. This DLL can be called using any software language that supports calls to DLLs. Below are the details of this DLL interface and the routines that are available for your use.

DLL FILENAME:

```
usbedCAN.dll in \Windows\System32
```

DLL EXPORTED FUNCTIONS AND PARAMETERS

ExtractionBufferCount – Returns the number of bytes that have been extracted from the data stream so far and are available to read using GetNextData.

```
CWAV_EXPORT unsigned long CWAV_API ExtractionBufferCount(void)
```

Returns:

- 0 – No data to read yet
- other – number of bytes available to read

GetNextData – Copies the extracted data from the extractor into your working buffer

```
CWAV_EXPORT char CWAV_API GetNextData(unsigned char *buffer,
unsigned long length);
```

buffer: pointer to where you want the extracted data to be placed

length: number of bytes you want to read from the extraction DLL

Returns:

- 0 – No data to read yet
- 1 – Data was copied into the buffer

StartExtraction – Starts the Data Extraction with the given parameters.

```
CWAV_EXPORT int CWAV_API StartExtraction(unsigned long PodNumber,
unsigned long Speed, unsigned char All, unsigned char Decimal,
unsigned char Hex, unsigned char Binary, unsigned char Comma,
unsigned char Space, unsigned char Timestamps, unsigned long
MaxIDFilter, unsigned long MinIDFilter)
```

PodNumber: Pod ID on the back of the USBee DX Test Pod

Speed: Bit rate of the CAN bus in bits per second

All:

- 0 – Only the data payload bytes are returned
- 1 – All CAN packet fields are returned

Decimal:

- 1 – Decimal Values (text) are output for the data bytes

Hex:

- 1 – Hex Values (text) are output for the data bytes

Binary:

- 1 – All data is in binary form, not text

Comma:

- 1 – Commas are placed between each field/data byte

Space:

- 1 – Spaces are placed between each field/data byte

Timestamp:

- 1 – Print Timestamps at the start of each packet

MaxIDFilter:

- The Maximum Identifier to log (0xFFFFFFFF default)

MinIDFilter:

- The Minimum Identifier to log (0 default)

Returns:

- 1 – if Start was successful
- 0 – if Pod failed initialization

StopExtraction – Stops the extraction in progress

```
CWAV_EXPORT int CWAV_API StopExtraction( void );
```

Returns:

- 1 – always

ExtractBufferOverflow – Returns the state of the overflow conditions

```
CWAV_EXPORT char CWAV_API ExtractBufferOverflow(void);
```

Return:

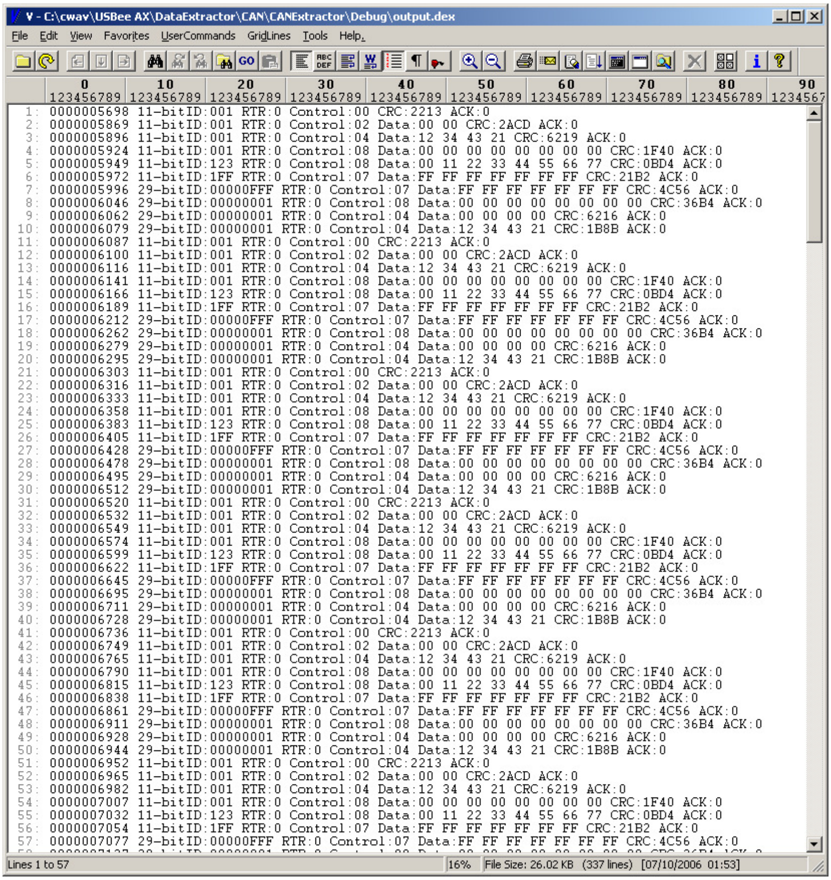
- 0 – No overflow
- 1 – Overflow Occurred. ExtractorBuffer Overflow condition cleared.
- 2 – Overflow Occurred. Raw Stream Buffer Overflow

EXTRACTION DATA FORMAT

The GetNextData routine gets a series of bytes that represent the extracted data stream and places these bytes into the buffer pointed to by the *buffer parameter.

The CAN Bus Extractor DLL sends the extracted data through the *buffer in the requested form based on the parameters in the StartExtraction call. For example, if Binary is set to a 0, then the *buffer will receive the binary bytes that make up the data stream. If Hex is set to a 1, the *buffer will contain a text string which is the data of the CAN traffic in Hex text form, separated by any specified delimiters.

```
CANExtractor -O output.dex -S -P 3209 -Q 500000 -R 250000 -A -H -V 1
```


V - C:\cwav\USBee AX\DataExtractor\CAN\CANExtractor\Debug\output.dex
File
Edit
View
Favorites
UserCommands
GridLines
Tools
Help.
Lines 1 to 57
16%
File Size: 26.02 KB (337 lines) [07/10/2006 01:53]

```
0000005698 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
0000005869 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
0000005896 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
0000005924 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:1F40 ACK:0
0000005949 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
0000005972 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
0000005996 29-bitID:00000FFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
0000006046 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:36B4 ACK:0
0000006062 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 CRC:6216 ACK:0
0000006079 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
0000006087 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
0000006100 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
0000006116 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
0000006141 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:1F40 ACK:0
0000006166 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
0000006189 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
0000006212 29-bitID:00000FFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
0000006262 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:36B4 ACK:0
0000006279 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 CRC:6216 ACK:0
0000006295 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
0000006303 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
0000006316 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
0000006333 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
0000006358 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:1F40 ACK:0
0000006383 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
0000006405 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
0000006428 29-bitID:00000FFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
0000006478 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:36B4 ACK:0
0000006495 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 CRC:6216 ACK:0
0000006512 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
0000006520 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
0000006532 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
0000006549 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
0000006574 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:1F40 ACK:0
0000006599 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
0000006622 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
0000006645 29-bitID:00000FFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
0000006695 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:36B4 ACK:0
0000006711 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 CRC:6216 ACK:0
0000006728 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
0000006736 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
0000006749 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
0000006765 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
0000006790 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:1F40 ACK:0
0000006815 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
0000006838 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
0000006861 29-bitID:00000FFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
0000006911 29-bitID:00000001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:36B4 ACK:0
0000006928 29-bitID:00000001 RTR:0 Control:04 Data:00 00 00 00 CRC:6216 ACK:0
0000006944 29-bitID:00000001 RTR:0 Control:04 Data:12 34 43 21 CRC:1B8B ACK:0
0000006952 11-bitID:001 RTR:0 Control:00 CRC:2213 ACK:0
0000006965 11-bitID:001 RTR:0 Control:02 Data:00 00 CRC:2ACD ACK:0
0000006982 11-bitID:001 RTR:0 Control:04 Data:12 34 43 21 CRC:6219 ACK:0
0000007007 11-bitID:001 RTR:0 Control:08 Data:00 00 00 00 00 00 00 00 CRC:1F40 ACK:0
0000007032 11-bitID:123 RTR:0 Control:08 Data:00 11 22 33 44 55 66 77 CRC:0BD4 ACK:0
0000007054 11-bitID:1FF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:21B2 ACK:0
0000007077 29-bitID:00000FFF RTR:0 Control:07 Data:FF FF FF FF FF FF FF CRC:4C56 ACK:0
```

EXAMPLE SOURCE CODE

```
//****************************************************
// USBee DX Data Extractor
// CAN Bus Extractor Example Program
// Copyright 2006, CWAV All Rights Reserved.
//****************************************************

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "windows.h"
#include <fcntl.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>

#define MAJOR_REV 1
#define MINOR_REV 0

//****************************************************
// Declare the Extractor DLL API routines
//****************************************************

#define CWAV_API __stdcall
#define CWAV_IMPORT __declspec(dllimport)

CWAV_IMPORT int CWAV_API StartExtraction(unsigned long PodNumber, unsigned long Speed,
unsigned char All, unsigned char Decimal, unsigned char Hex, unsigned char Binary, unsigned
char Comma, unsigned char Space, unsigned char Timestamps, unsigned long MaxID,unsigned
long MinID);
CWAV_IMPORT char CWAV_API GetNextData(unsigned char *buffer, unsigned long length);
CWAV_IMPORT int CWAV_API StopExtraction( void );
CWAV_IMPORT char CWAV_API ExtractBufferOverflow(void);
CWAV_IMPORT unsigned long CWAV_API ExtractionBufferCount(void);

//****************************************************
// Define the working buffer
//****************************************************

#define WORKING_BUFFER_SIZE   (65536*8)
unsigned char tempbuffer[WORKING_BUFFER_SIZE];

// Command Line Parameter Settings
unsigned long P_PodID = 0;
unsigned char O_OutputFilename[256] = {0};
unsigned char S_Screen = FALSE;
unsigned char A_All = TRUE;
unsigned char B_DataOnly = FALSE;
unsigned char D_DecimalTextValues = FALSE;
unsigned char H_HexTextValues = TRUE;
unsigned char I_BinaryValues = FALSE;
unsigned char C_CommaDelimited = FALSE;
unsigned char G_SpaceDelimited = FALSE;
unsigned long Q_NumberOfBytes = 0;
unsigned long R_Speed = 250000;
unsigned long V_Timestamps = TRUE;
unsigned long M_ID = 0xFFFFFFFF;
unsigned long N_ID = 0;


void DisplayHelp(void)
{
     fprintf(stdout,"\nCANExtractor [-?SDHICGAB] [-R CANSpeed] [-Q NumberOfBytes] [-V
Timestamp] [-O filename] [-M MaxID] [-N MinID] -P PodID\n");

     fprintf(stdout,"\n    ?  - Display this help screen\n");

     fprintf(stdout,"\n  USBee DX Pod to Use\n");

     fprintf(stdout,"    P  - Pod ID (required)\n");

     fprintf(stdout,"\n  Output Location Flags\n");

     fprintf(stdout,"    O  - Output to filename (default off)\n");
     fprintf(stdout,"    S  - Output to the screen (default off)\n");

     fprintf(stdout,"\n  When to Quit Flags\n");
     fprintf(stdout,"    Q  - Number of output values (default = until keypress)\n");

     fprintf(stdout,"\n  Input Format Flags\n");

     fprintf(stdout,"    R  - Bus Speed in bits/second (default = 250000)\n");

     fprintf(stdout,"\n  Output Number Format Flags\n");

     fprintf(stdout,"    A  - All Packet Fields are output (default)\n");
     fprintf(stdout,"    B  - Only data bytes are output\n");
     fprintf(stdout,"    D  - Decimal Text Values (\"49\")\n");
     fprintf(stdout,"    H  - Hex Text Values (\"31\") default\n");
     fprintf(stdout,"    I  - Binary Values (49)\n");
     fprintf(stdout,"    C  - Comma Delimited\n");
     fprintf(stdout,"    G  - Space Delimited (default)\n");
     fprintf(stdout,"    V  - Timestamps (0=off(default),1=Timestamp on\n");
     fprintf(stdout,"    M  - Maximum Identifier Filter\n");
     fprintf(stdout,"    N  - Minimum Identifier Filter\n");



}

void Error(char *err)
{
     fprintf(stderr,"Error: ");
     fprintf(stderr,"%s\n",err);
     exit(2);
}


//****************************************************
// Parse all of the command line options
//****************************************************
void ParseCommandLine(int argc, char *argv[])
{
     BOOL cont;
     int       i,j;
     DWORD WordExample;
     BYTE ByteExample;

     for(i=1; i < argc; ++i)
     {
          if((argv[i][0] == '-') || (argv[i][0] == '/'))
          {
               cont = TRUE;
               for(j=1;argv[i][j] && cont;++j)     // Cont flag permits multiple commands
in a single argv (like -AR)
                    switch(toupper(argv[i][j]))
                    {
                         case 'P':
                              P_PodID = (WORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'O':
                              strcpy((char*)O_OutputFilename, argv[++i]);
                              cont = FALSE;
                              break;
                         case '?':
                              DisplayHelp();
                              exit(0);

                              break;
                         case 'S':
                              S_Screen = TRUE;
                              break;
                         case 'A':
                              A_All = TRUE;
                              B_DataOnly = FALSE;
                              break;
                         case 'B':
                              A_All = FALSE;
                              B_DataOnly = TRUE;
                              break;
                         case 'D':
                              D_DecimalTextValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'H':
                              H_HexTextValues = TRUE;
                              break;
                         case 'I':
                              I_BinaryValues = TRUE;
                              H_HexTextValues = FALSE;
                              break;
                         case 'C':
                              C_CommaDelimited = TRUE;
                              G_SpaceDelimited = FALSE;
                              break;
                         case 'G':
                              G_SpaceDelimited = TRUE;
                              break;
                         case 'Q':
                              Q_NumberOfBytes = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'V':
                              V_Timestamps = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'R':
                              R_Speed = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'M':
                              M_ID = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'N':
                              N_ID = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'w':
                              WordExample = (DWORD)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         case 'b':
                              ByteExample = (BYTE)strtol(argv[++i],NULL,0);
                              cont = FALSE;
                              break;
                         default:
                              DisplayHelp();
                              fprintf(stdout,"\nCommand line switch %c not
recognized\n",toupper(argv[i][j]));
                              Error("Invalid Command Line Switch");
                              exit(0);
                    }
          }
     }


     // Now check to see if they make sense
     if (P_PodID == 0)
     {
          DisplayHelp();
          Error("No Pod Number Specified");
     }

}


//****************************************************
// Main Entry Point.  The program starts here.
//****************************************************

int main(int argc, char* argv[])
{
     int RetValue;
     unsigned long totalbytes = 0;
     char *outputstr = new char [256];
     unsigned long ByteCounter = 0;
     unsigned long OutputValue;

     printf("DX Data Extractor\n");
     printf("CAN Bus Extractor Version %d.%d\n", MAJOR_REV, MINOR_REV);

     // Parse out the command line options
     ParseCommandLine( argc, argv );

     //****************************************************
     // Open up a file to store extracted data into
     //****************************************************

     FILE *fout;
     if (O_OutputFilename[0])
     {
          if (I_BinaryValues)
               fout = fopen((char*)O_OutputFilename, "wb");
          else
               fout = fopen((char*)O_OutputFilename, "w");
     }

     //****************************************************
     // Start the DX Pod extracting the data we want
     //****************************************************

     int Endpoint = 999;
     int Device = 999;

     RetValue = StartExtraction(P_PodID, R_Speed, A_All, D_DecimalTextValues,
H_HexTextValues, I_BinaryValues, C_CommaDelimited, G_SpaceDelimited, V_Timestamps, M_ID,
N_ID)     ;

     if (RetValue == 0)
     {
          printf("Startup failed.  Is the USBee DX connected and is the PodNumber
correct?\n");
          printf("Press any key to continue...");
          getch();
          return(0);
     }


     //****************************************************
     // Loop and do something with the collected data
     //****************************************************

     char OldSignal = 99;

     int KeepLooping = TRUE;
     while(KeepLooping)       // Do this forever until we tell it to stop by pressing a key
     {

          if (kbhit())
          {
               KeepLooping = FALSE;          // Stop the processing loop
               StopExtraction();             // Stop the streaming of data from the USBee
          }

          //****************************************************
          // If there is data that has come in
          //****************************************************
          int timeout = 0;
          while (unsigned long length = ExtractionBufferCount())
          {
               if (length > WORKING_BUFFER_SIZE)
                    length = WORKING_BUFFER_SIZE;

               //****************************************************
               // Get the data into our local working buffer
               //****************************************************

               GetNextData( tempbuffer, length );

               totalbytes += length;

               if (O_OutputFilename[0])
                    fwrite(tempbuffer, length, 1,  fout);   // Write it to a file

               if (S_Screen)
                    fwrite(tempbuffer, length, 1,  stdout); // Write it to the screen

               if (Q_NumberOfBytes)
               {
                    if (Q_NumberOfBytes <= length)
                    {
                         goto Done;          // Done with that many bytes
                    }
                    Q_NumberOfBytes -= length;
               }
               if (timeout++ > 3  ) break;  // Let up once in a while to let the OS process
          }

          if (!S_Screen)
               printf("\rProcessed %d output values.", totalbytes);

          //****************************************************
          // Check to see if we have fallen behind too far
          //****************************************************

          int y = ExtractBufferOverflow();

          if (y == 1)
          {
               printf("\nExtractor Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }
          else if (y == 2)
          {
               printf("\nRaw Sample Buffer Overflow.\nYour data is streaming too fast for
your output settings.\nLower your data rate or change to output binary files.\n");
               goto Done;
          }

          //****************************************************
          // Give the OS a little time to do something else
          //****************************************************

          Sleep(15);

     }

Done:
     if (!S_Screen)
          printf("\rProcessed %d output values.", totalbytes);

     //****************************************************
     // Close the file
     //****************************************************

     if (O_OutputFilename[0])
          fclose(fout);

     //****************************************************
     // Stop the extraction process
     //****************************************************

     StopExtraction();

     if (kbhit()) getch();
     printf("\nPress any key to continue...");
     getch();

     return 0;
}
```

Printed in the USA

Version 3.1