USBEE SUITE

PRO AND STANDARD

USERS MANUAL

CWAV

www.usbee.com

USBEE SUITE

Pro and Standard

USERS MANUAL

CWAV
www.usbee.com
(951) 694-6808
support@usbee.com

USBee Suite License Agreement

The following License Agreement is a legal agreement between you (either an individual or entity), the end user, and CWAV, Inc. makers of the USBee Test Pods and USBee Suite software. You have received or downloaded the USBee Suite Package ("Software"), which consists of the USBee Suite Software and Documentation. If you do not agree to the terms of the agreement, do not install and run this software.

By installing this USBee Suite software, 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 Test Pods and grants you license to use this Software under the following conditions: a) You may use the USBee Software only in conjunction with a USBee Test 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, Inc, 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 Software, 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, Inc, with the sole exception of making backup copies of the Software for restoration purposes on your own machine. You may not reverse engineer, decompile, disassemble, merge or alter the USBee Software or USBee Pod in any way.

Limited Warranty

The USBee Software 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.

Software Updates

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. The Software will change from time to time to add new functionality, fix bugs and disable the use of non-CWAV made products that attempt to use this Software.

Liability

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 Suite User's Manual, Version 2.5

CONTENTS

INTRODUCING THE USBEE SUITE 11

USBee Suite Standard Overview 11
See the information you want FAST! 13
Data Acquisition over USB 2.0 13
Superior Quality Design - Professional Results 13
Fast and Detailed Waveform Viewing 13
Measure It 14
Fully configurable Look and Feel 14
Serial Bus Analysis 15
Data Storage 17
Scalability 17
USBee Suite PRO Overview 18
PacketPresenter 19
Fast Pan Bus Viewing 20
Smart Search 21
Sample and Smart Markers 21
Annotations and Sticky Notes 22
Acquisition Control 22
Display Modes 23
Analog Channels Scaling 23
ULD Data File Importing 24
Browser-like Navigation 24
Analog Triggering 24
Relative Time Decode 25
USBee Protolyzer Control Panel 26
PC System Requirements 27
Software Installation 27
Enabling the USBee Suite Pro Features 32

QUICK START 33

USING THE USBEE SUITE STANDARD 34

Initialization 34
Analyzer Setup 35
Quick Setup Configuration 35
Signal Names 38
Buffer Sizes and Sample Rate settings 38
Setting Triggers 40
Capturing Waveform Data 41

VIEWING CAPTURED DATA 41
Scrolling, Zooming and Panning Waveforms *41*
Modifying Wave lines *42*
Decoding Bus Traffic Inline *44*
Decoded Data List *46*
Manual Measurements and Cursors *47*
Insta-Measurements *48*
BUS DECODING OPTIONS 49
CAN Bus Setup *49*
USB Bus Setup *51*
I2C Bus Setup *52*
Async Bus Setup *53*
Parallel Bus Setup *55*
1-Wire Bus Setup *56*
SPI Bus Setup *57*
SM Bus Bus Setup *58*
Serial Bus Setup *59*
I2S Bus Setup *60*
PS/2 Bus Setup *61*
SETTING VIEWING PREFERENCES 62
Cursor Colors *62*
Background Color *62*
Glass On Vista *62*
FILE OPERATIONS 63
Creating a New File *63*
Saving a Capture File *63*
Open and Existing Capture File *63*
Recently Used File List *63*
Exporting Captured Data to a File *63*
Export Signal Data to Binary File 64
Export Signal Data to Text/CSV File 65
Export Bus Data to Text/CSV File 65
PRINTING 66
CREATING SCREEN SHOTS 66
SOFTWARE UPDATES 66
DEVELOPING YOUR OWN CUSTOM DECODERS 67
Using the Custom Decoder *67*
Building the Custom Decoder *69*
Example Class Library Code *70*

Custom Decoder Parameters 72
Accessing Sample Data to Perform Decode 73
Outputting Entries that will get displayed on the Screen 73
Changing the Background Color of Outputted Entries 74

USING THE USBEE SUITE PRO 75

SMART SEARCH 76
Adding a Search Line 76
Viewing Search Matches 77
Entering a Smart Search 78
Digital Signal Edges 79
Analog Signal Edges 80
Bus Data 81
Digital Signal States and Ranges 82
Analog Signal States and Ranges 84
Time Window Qualifier 86
FAST PAN BUS VIEWING 89
SAMPLE AND SMART MARKERS 90
Sample Markers 90
SMart Markers 91
ANNOTATIONS AND STICKY NOTES 92
Annotations 93
Sticky Notes 94
ACQUISITION CONTROL 95
DISPLAY MODES 96
ANALOG CHANNELS SCALING 98
ULD DATA FILE IMPORTING 99
BROWSER-LIKE NAVIGATION 100
ANALOG TRIGGERING 100
RELATIVE TIME DECODE 102
PACKETPRESENTER™ 104
Overview 104
Setting Up the PacketPresenter 106
Viewing the PacketPresenter Output 106
Saving PacketPresenter Data to Text or RTF Files 107
Copying PacketPresenter Output to Other Programs 108
Changing the PacketPresenter Size 110
Searching For Packets 111
Filtering Packets 112
Multiple Decode Display 113

PacketPresenter to Waveform Association *114*
Cursors on the PacketPresenter Output *115*
PacketPresenter Definition File Format *116*
Comments in the PacketPresenter Definition File 116
Constants in the PacketPresenter Definition File 116
PacketPresenter Definition File Sections 117
Protocol Section 117
Byte-wise busses vs. Bit-wise busses 117
Bus Events 118
Data Channels and Multiple Data Signals 119
Packet Section 120
Start and End Sections 120
type = Next 120
type = Signal 121
type = Value 121
type = Length 121
type = Event 122
type = Timeout 122
CHANNELX, CHANNELY or CHANNELXorY 123
Decode Section 123
Substitutions 124
Fields Section 124
Field Lines Processing 124
Unconditional Field Lines 125
Conditional Field Lines 125
Field Line Format 125
Field Format 125
Bus Events in the middle of a packet 126
Lookup Tables 127
Examples of Field Lines and Fields 127
Just Plain Data 127
Conditional Packet Format 128
String Lookup 129
Conditional Route of data to another Protocol 129
PacketPresenter Add-In API *130*
Sample PacketPresenter Add-In Decoders 131
Loopback Decoder 131
Inverting Decoder 131
Expanding Decoder 131
Compressing Decoder 132
Multiple Decoders 133
PacketPresenter Definition File Debugging *133*

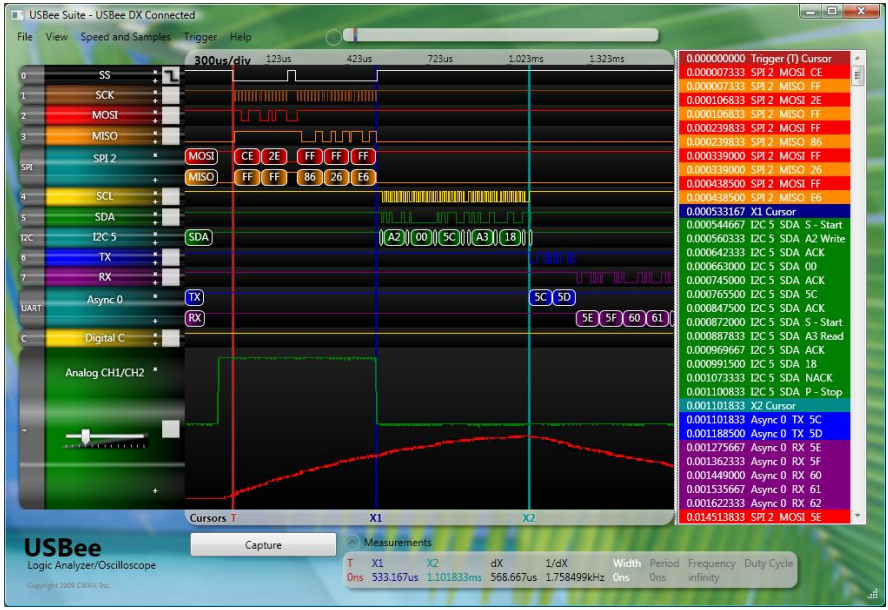
PacketPresenter Specifications *134*
Example Protocol Files and Output Examples *135*
Async Protocol Example 135
I2C Protocol Example 136
SPI Protocol Example 137
CAN Protocol Example 139
1-Wire Protocol Example 140
Parallel Protocol Example 141
Serial Protocol Example 142
USB Protocol Example 143
PS2 Protocol Example 145

USBEE PROTOLYZER CONTROL PANEL 146

INSTALLING THE USBEE SUITE WITH A USBEE PROTOLYZER 147
1) PC Software Setup *148*
2) Connecting Probes *148*
3) Connecting your Protolyzer to the PC *148*
RUNNING THE USBEE PROTOLYZER CONTROL PANEL 149

GETTING HELP 152

INTRODUCING THE USBEE SUITE



The USBee Suite is powerful electronic signal analysis software for your USBee Test Pod. It starts out as an easy to use Logic Analyzer and Oscilloscope and adds serial bus decoding and world class configurability that lets you solve your electronic problems quickly!

There are two versions of the USBee Suite, Standard and Pro. The free **USBee Suite Standard** version includes features you need to capture and analyze your digital, analog and embedded serial bus signals. The upgraded **USBee Suite Pro** (additional purchase) adds many additional features including our top-of-the-line PacketPresenter Protocol Decoder, enhanced importing and exporting of data, documentation tools and visual configuration capabilities.

This User's Manual details the operation of the USBee Suite Standard and Pro versions. The USBee Suite Standard features are described first followed by the USBee Suite Pro features. All USBee Suite Standard features also apply to the USBee Suite Pro version.

USBEE SUITE STANDARD OVERVIEW

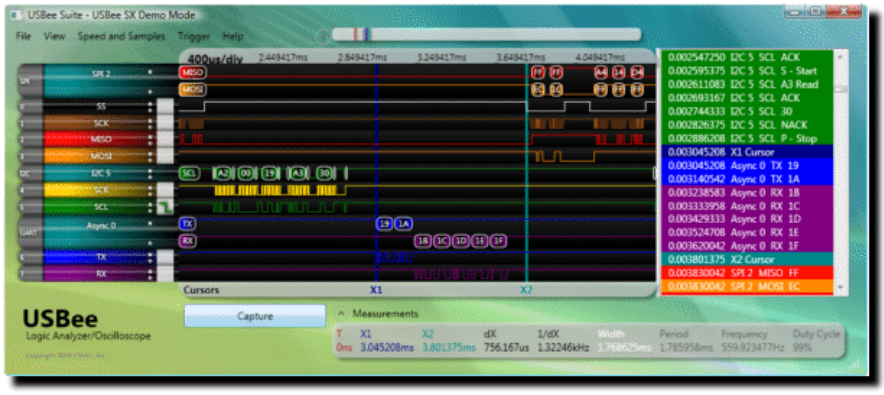
The USBee Suite Standard is a powerful mixed signal analyzer that runs with any of the current USBee Test Pods. It is available for free from USBee.com and will run without restriction. Simply download the USBee Suite from our web site, install it on a system with a USBee Pod installed and it will help you debug your systems faster.

Below is a quick list of the USBee Suite Standard features.

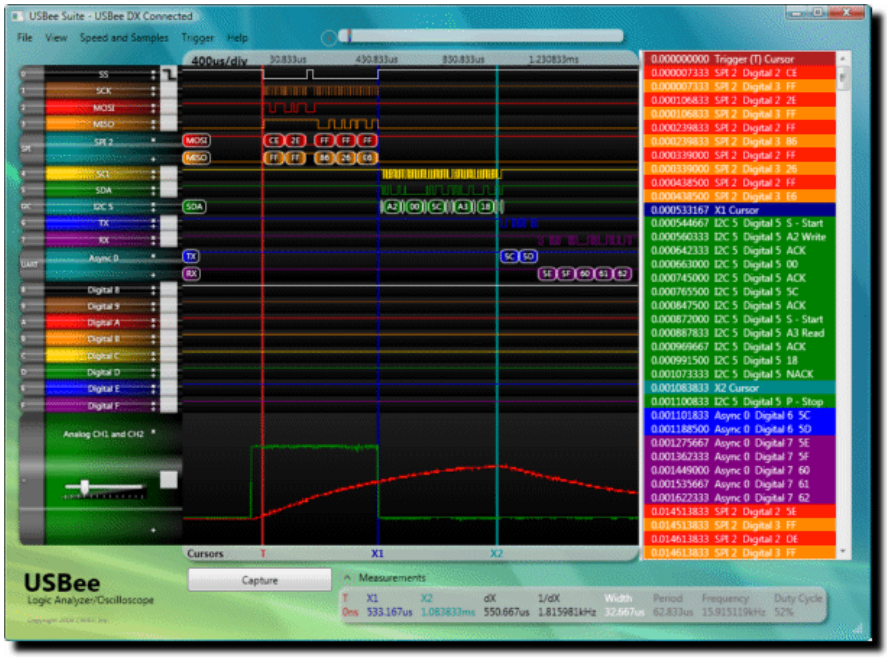
	USBee SX	USBee ZX	USBee AX	USBee DX
Oscilloscope channels			1	2
Logic Analyzer channels	8	8	8	16
In-line protocol decoding	✔	✔	✔	✔
USB Decoder	✔	✔	✔	✔
I2C Decoder	✔	✔	✔	✔
SPI Decoder	✔	✔	✔	✔
Async Decoder	✔	✔	✔	✔
1-Wire Decoder	✔	✔	✔	✔
CAN Decoder	✔	✔	✔	✔
I2S Decoder	✔	✔	✔	✔
SM Bus Decoder	✔	✔	✔	✔
PS/2 Decoder	✔	✔	✔	✔
Synchronous Serial Decoder	✔	✔	✔	✔
Parallel Decoder	✔	✔	✔	✔
Custom Decoder development kit for creating your own decoder	✔	✔	✔	✔

USBee Suite Standard Features

The USBee Suite Standard will run in Demo mode without a USBee, or on any USBee SX, USBee ZX, USBee AX-Standard, USBee AX-Plus, USBee AX-Pro, or USBee DX Test Pod. Because each USBee Test Pod has different combinations of digital and analog channels, the USBee Suite will only support the features your USBee is capable of. Below are example screenshots of what is available on the USBee SX versus the USBee DX test pods.



USBee SX features shown above



USBee DX Features shown above

SEE THE INFORMATION YOU WANT FAST!

Setup of the USBee Suite is fast! Capturing the data you need to solve your problems is just as fast. You can see your design in action with just one click thanks to the easy to use trigger settings, color coded signals and automatic buffer and sample rate settings.

DATA ACQUISITION OVER USB 2.0

Capture up to 24 million bytes per second directly into your PC's RAM, for sample buffer depths of hundreds of millions of samples.

SUPERIOR QUALITY DESIGN - PROFESSIONAL RESULTS

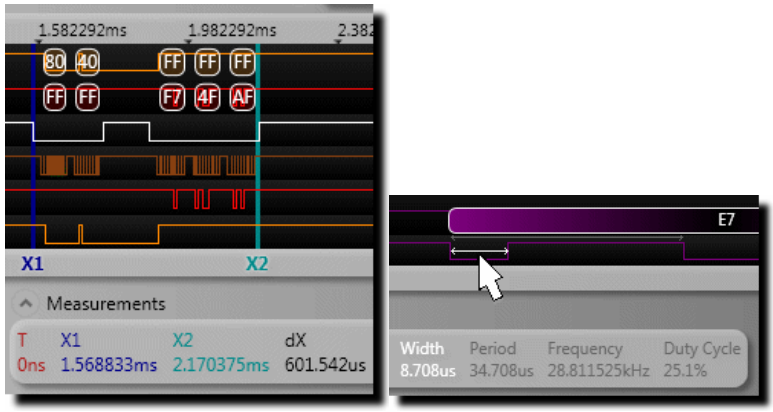
The USBee Suite takes full advantage of the power of each of the various USBee Test Pods. Each USBee comes with the best color coded highly flexible test leads, the best test clips and our signature small and sleek design that can fit right in your pocket. We are proud to say the entire USBee product line is designed and manufactured in the USA!

FAST AND DETAILED WAVEFORM VIEWING

The USBee Suite lets you capture a huge amount of data. Go exactly to the section of that data you want using the Quick Zoom with your mouse scroll wheel, or use the Overview bar to rip through your millions of samples or hone in on a specific section.

MEASURE IT

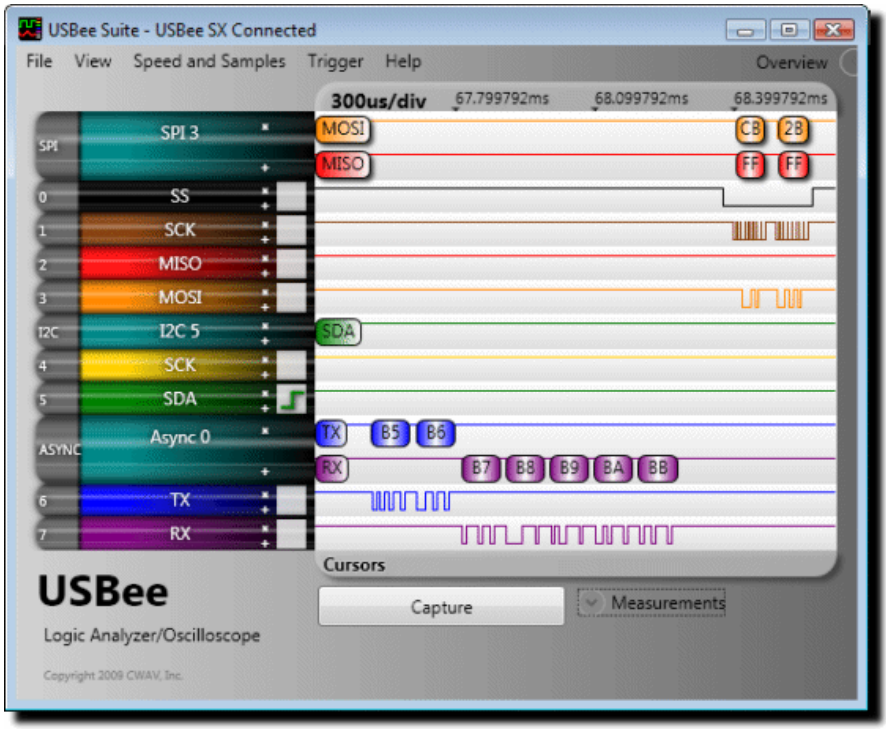
Should that pulse be 10ms? Measure it using our super easy edge snapping timing cursors. Better yet, you can use our Insta-Measure feature to instantly calculate the width, period, frequency and duty cycle of the waveform under the cursor.



FULLY CONFIGURABLE LOOK AND FEEL

View your signals like you like them. Want to add decoded bus traffic to the waveforms? Done! Want to delete waves from the screen? Done! Want to reorder waveforms for easier readability? Done! Want to resize the screen for easier reading or more data per screen? Done!

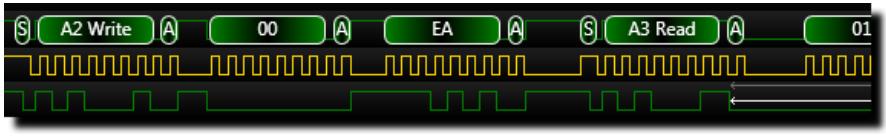
And you like Magenta? Well you can change cursor colors to suite your desires. Waveform backgrounds can also be customized, and you can even give the entire application that cool Glassy look that Vista has made so popular. Then again, if you like simple, white and black are also available. It's good that white ink cartridges are free!



USBee Suite showing SPI, I2C and Async decoding

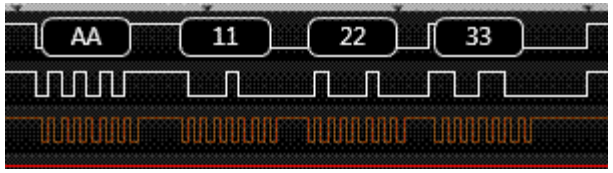
SERIAL BUS ANALYSIS

USBee Suite has decoding support for your favorite serial busses such as I2C, SPI and Async. Bus traffic is decoded in-line with the waveforms and can be displayed on top of, underneath, or instead of the voltage versus time waveform. Just place the cursor over the decoded traffic and get a see-through image that shows you the wiggles that made that byte!



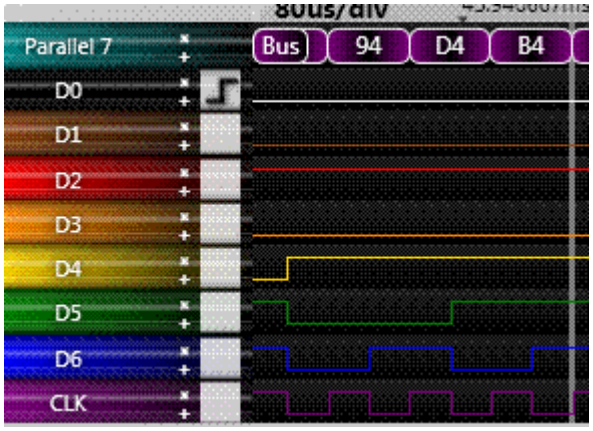
I2C Transaction

USB Transaction

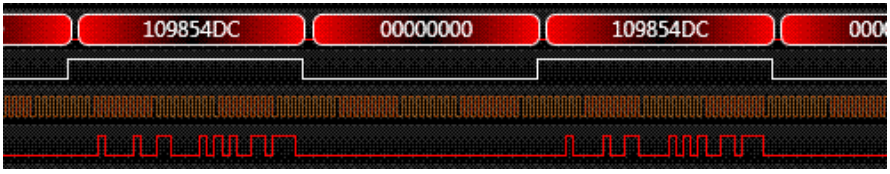


Synchronous Serial Transaction

PS2 Transaction



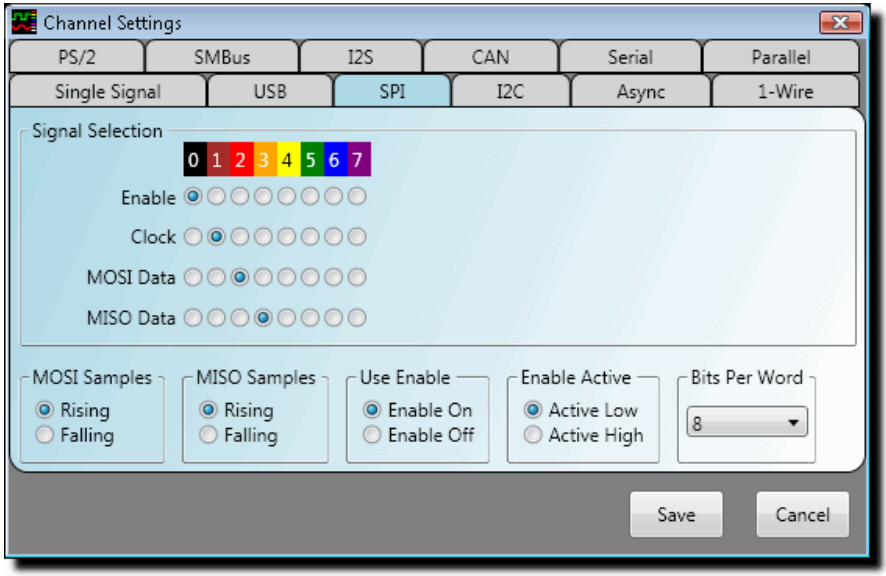
Parallel Bus Decode



I2S Decode

CAN Transaction

Serial Bus Setup is simple and straightforward - simply choose the signals for the bus and set how the bus is configured. Not sure how your design works? Not a problem. You can try different configuration settings and the busses will be decoded using those settings on the fly so you can get it right!



(USBee SX options shown above)

DATA STORAGE

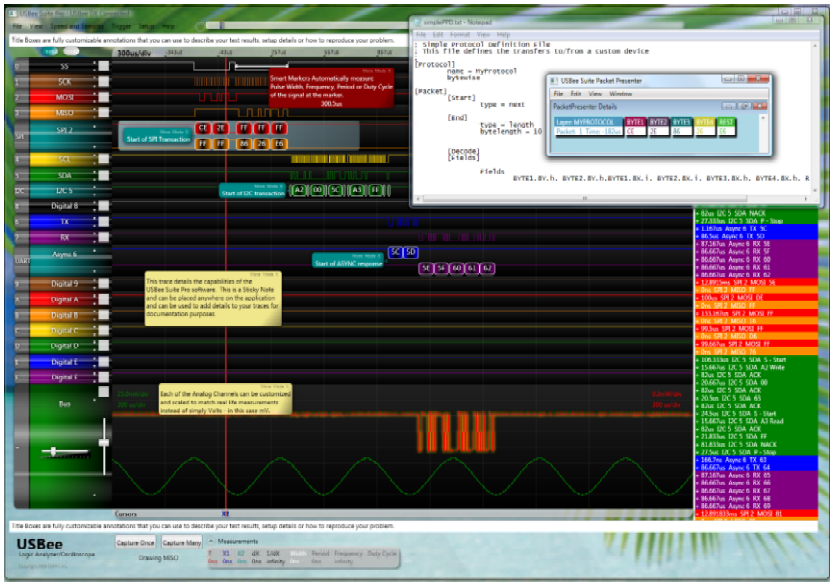
Save your entire data capture to file quickly using the USBee Suite data format to be read back in later for viewing. Or you can export your captured data to data files that you can work with. Want to import your waveform data into Excel? No problem! Just export it as a comma separated file and it imports directly without modification. Need the data in raw binary format? We've got that too!

SCALABILITY

The USBee Suite operates on our full line of USBee Test Pods, from the affordable USBee SX to the USBee DX powerhouse. So if you need more channels or analog capability, you can count on the ease of use and power of the USBee Suite no matter what tool you are using!

USBEE SUITE PRO OVERVIEW

The USBee Suite Pro contains all of the features of the USBee Suite Standard and adds many powerful features to assist your debugging to get you to the root of your problems quickly.

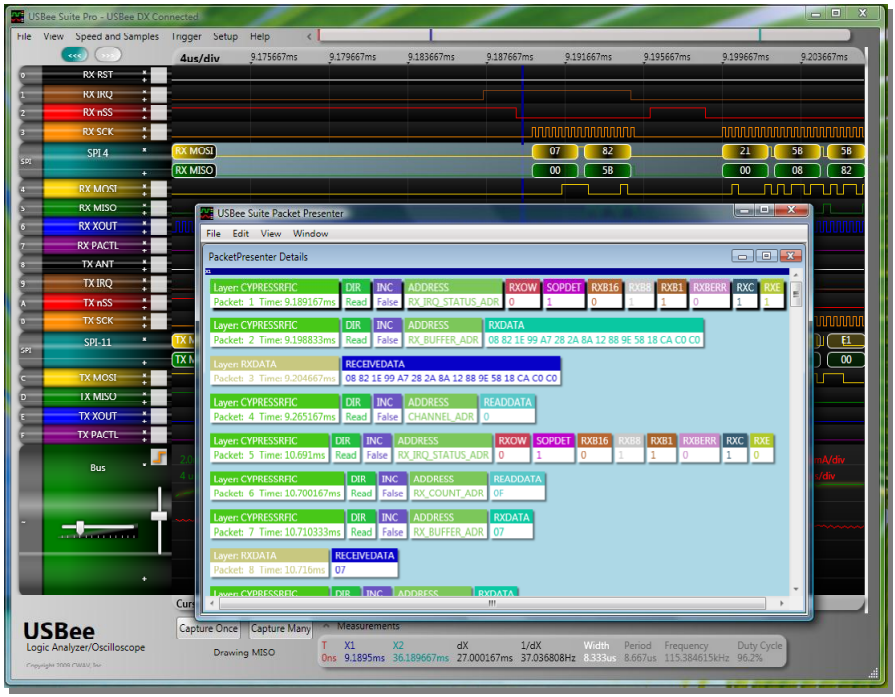


The USBee Suite Pro adds the following features:

- Smart Search – automatically finds the data you are looking for
- Fast Pan Bus Viewing – quickly pans through your decoded bus data
- Sample and Smart Markers – place markers to highlight important areas
- Annotations and Sticky Notes – add annotations to document your findings
- Advanced Acquisition Control – Auto or Normal Triggering, Capture Once or Capture Many
- Display Modes – configure the display as you like
- Analog Channel Scaling – scale the analog channels to your custom scale, offset and units
- Browser-like Navigation – jump forward and backward to previous locations in your trace
- Analog Triggering – trigger off of the analog channels
- ULD Data File Importing – import USBee DX native capture files
- Relative Time Decode – view timestamps for decoded data in relative format
- PacketPresenter – Display bus data as high level packets

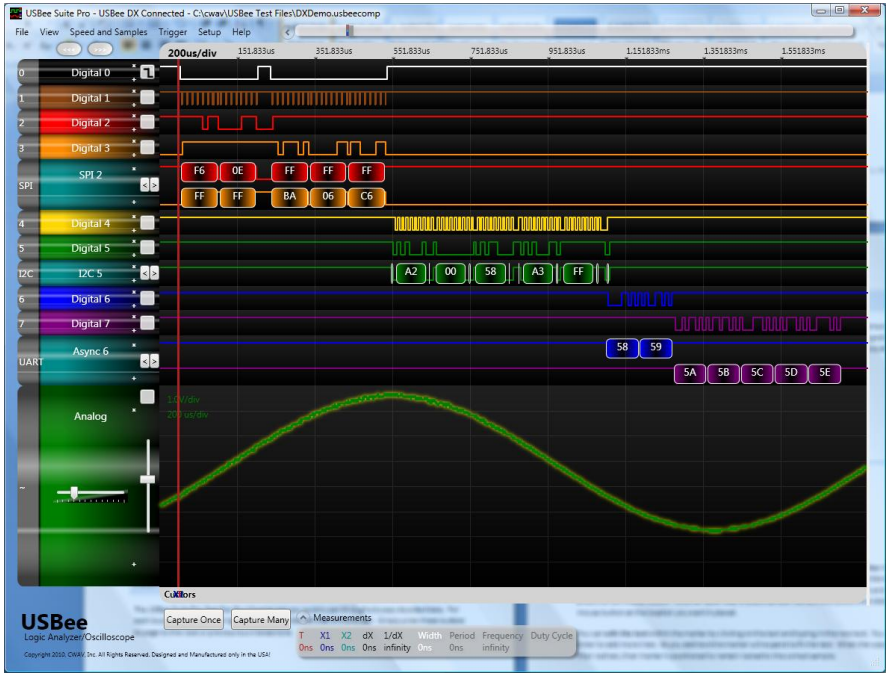
PACKETPRESENTER

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



FAST PAN BUS VIEWING

The USBee Suite Pro Fast Pan Bus Viewing lets you quickly pan through a busses decoded data. For each bus there is a left and right pan button on the left side of the screen. Simply press these buttons to page to the next or previous bus transactions.

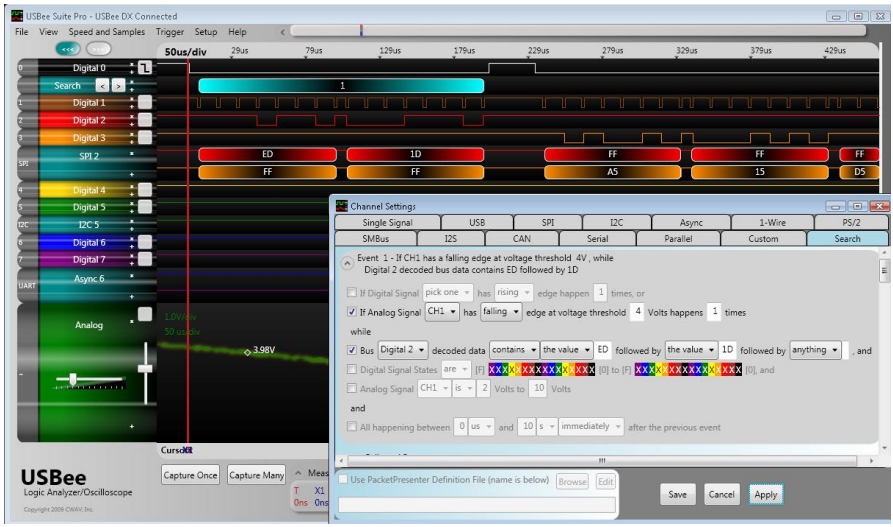


SMART SEARCH

USBee Suite Pro Smart Search highlights the sections of your trace matching your areas of interest so that you don't need to waste time hunting for the data you need.

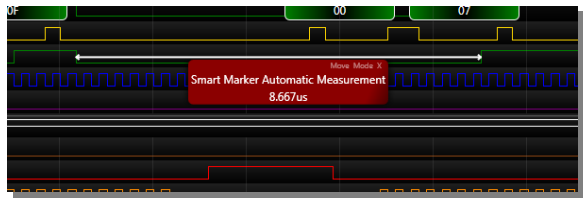
You can specify up to 32 levels of search events that are any combination of bus decoded traffic, states or edges of digital or analog signals, inside or outside of analog voltage ranges and/or digital ranges, and all validated by time specific windows.

Once specified you can pan through the occurrences of your searched items with the click of the mouse and see the total number of times the searched events occur.



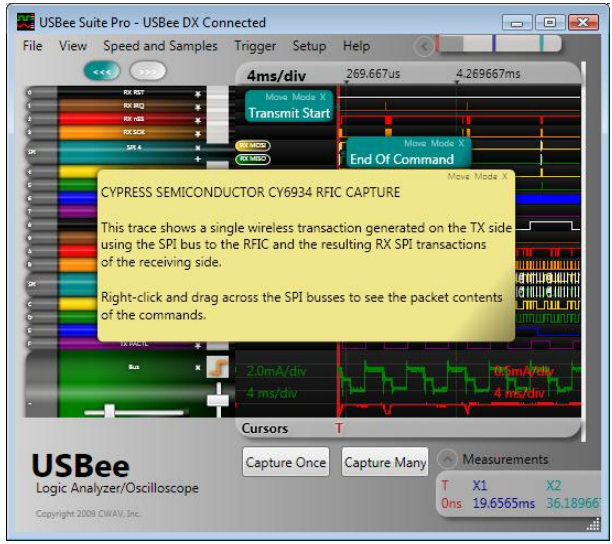
SAMPLE AND SMART MARKERS

Placing markers in your traces can help detail what is happening in your design. There are two types of markers that can be used. The first marker type locks itself to a sample on a waveform and lets you specify the text. The second is a Smart Marker that automatically measures the pulse width, frequency, period or duty cycle of the waveform at the marker location.



ANNOTATIONS AND STICKY NOTES

The USBee Suite Pro adds Sticky Notes which you can use to further detail your traces for documentation purposes. You can also add Title and Footer text to your display that is saved with the trace file.



ACQUISITION CONTROL

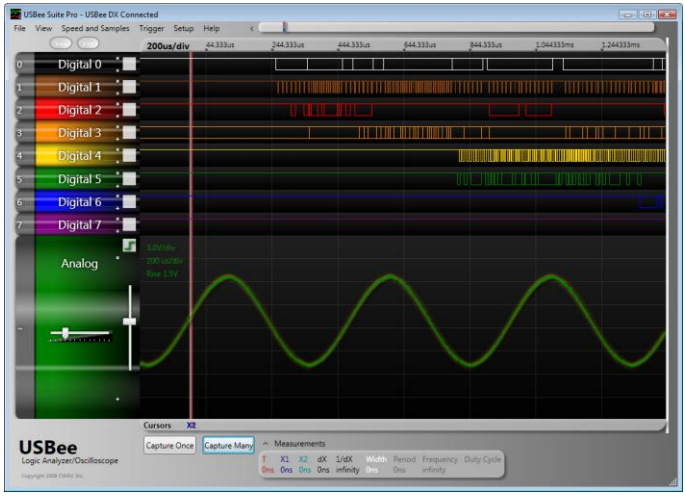
The USBee Suite Pro adds more trace acquisition and triggering controls such as Normal Mode, Automatic Mode, Single Capture and Multiple Capture.

Normal mode will wait for the trigger event to occur before capturing. Automatic Mode will wait a set time for the trigger and will automatically trigger if it is not found.

Single Capture mode performs a one-shot capture of the signals. Multiple Capture repeatedly captures and displays the signals.

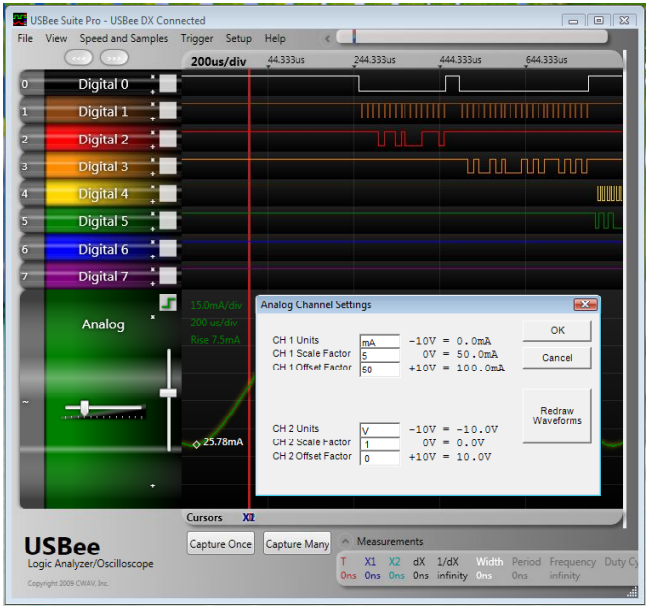
DISPLAY MODES

The USBee Suite Pro lets you widen the trace waveforms, display the analog waveforms as vectors or single sample points, and persist the display from one trace to the next.



ANALOG CHANNELS SCALING

The USBee Suite Pro provides a scaling ability to convert the analog voltages into other units of measurement.

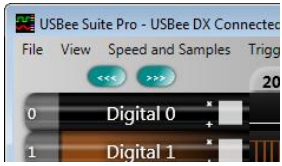


ULD DATA FILE IMPORTING

The USBee DX saves files in the ULD file format which can be imported into the USBee Suite Pro.

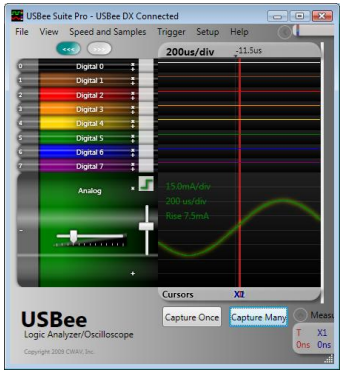
BROWSER-LIKE NAVIGATION

The USBee Suite Pro adds browser-like Forward and Back buttons that let you quickly navigate through your trace display.



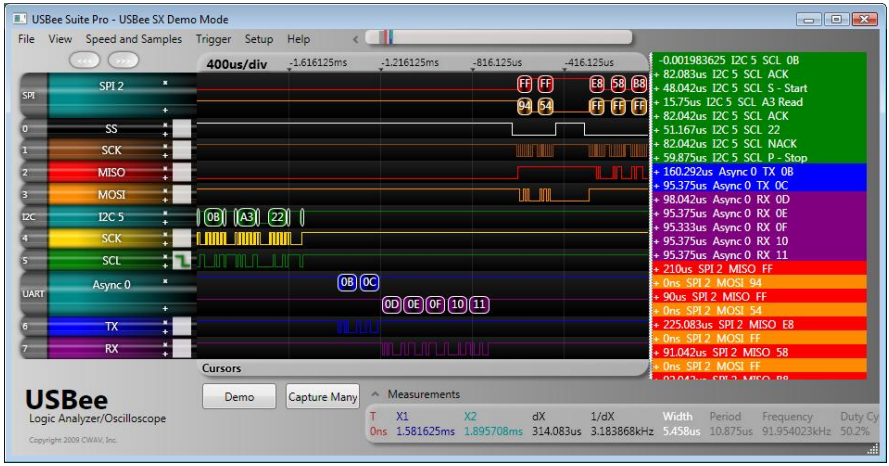
ANALOG TRIGGERING

The USBee Suite Pro adds the ability to trigger on a rising or falling edge of any analog channel.



RELATIVE TIME DECODE

The USBee Suite Pro also adds a Relative Time or Absolute Time setting for the decoded data lists.



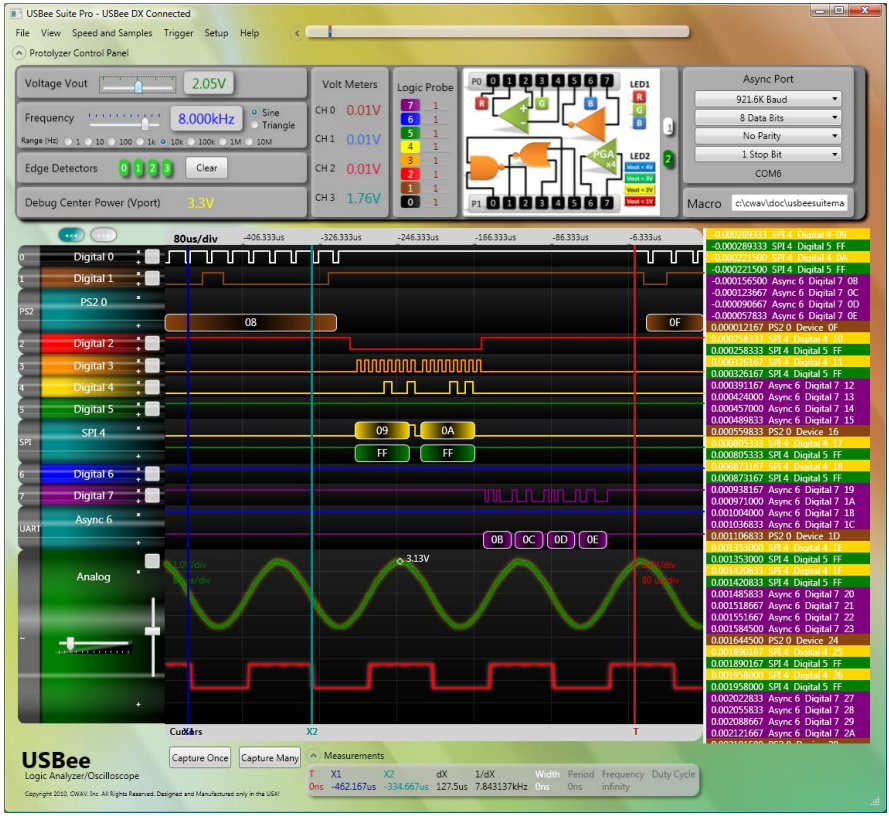
USBEE PROTOLYZER CONTROL PANEL

The USBee Suite contains a USBee Protolyzer Control Panel that appears when any USBee Protolyzer is connected to the PC when the USBee Suite is run.

The USBee Protolyzer is a unique system specifically built for Electronic Prototyping with Built-In Oscilloscopes, Logic Analyzers, Signal Generators, Protocol Analyzers and more! The USBee Protolyzer combines state of the art electronic prototyping development components with a complete range of digital and analog test equipment in a single PC-based USB connected device.

Focus on designing your next greatest invention. When it's time to turn on the power, strap it to the USBee Protolyzer to have all the tools necessary to bring up your board, validate your design and analyze your systems performance. View your embedded firmware in action and monitor your bus protocols as never before. Combined with the USBee Suite, the USBee Protolyzer gives you the power to create!

With a single USB connection to your laptop or PC, the USBee Protolyzer gives you the power to design, prototype, test, and validate your mixed signal electronic designs with seamless ease.



PC SYSTEM REQUIREMENTS

The USBee Suite requires the following minimum PC features:

- Windows® XP, Vista or Windows 7 32-bit or 64-bit operating system
- .NET Framework 3.5 SP1 or greater. This is installed automatically during installation if not already on your PC.
- 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)

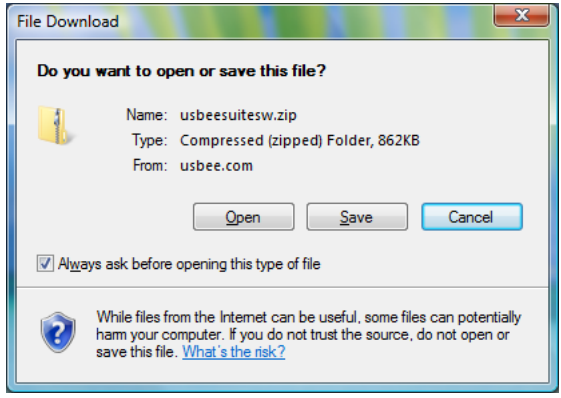
SOFTWARE INSTALLATION

The USBee Suite software is available for download from www.usbee.com/download.htm. It will run in a demonstration mode if you do not have a USBee Pod installed and attached.

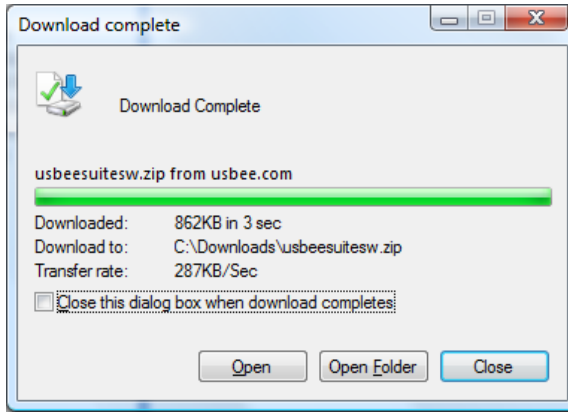
To install the software for demo purposes, just install the USBee Suite software. To install the USBee Suite to run on a USBee Pod, you must first install the USBee software package for your specific USBee device and Operating System variant (32 or 64-bit). These installations install the drivers and application specific to your device. Once these are installed and tested, you can then install and run the USBee Suite software.

To install the USBee Suite software:

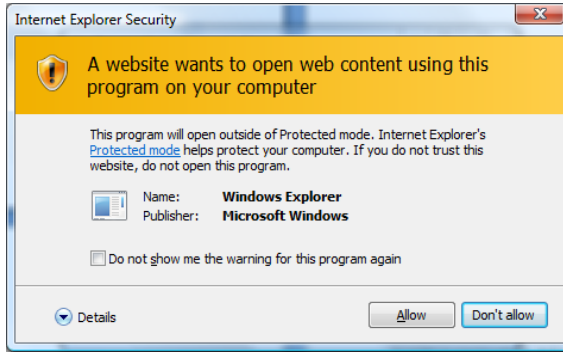
- Click the USBee Suite Software link at http://www.usbee.com/download.htm and click SAVE to save the software to a known directory.



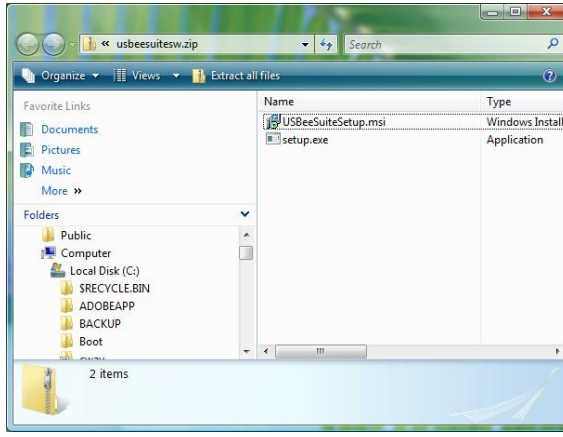
- Open the ZIP file you just downloaded by clicking OPEN.



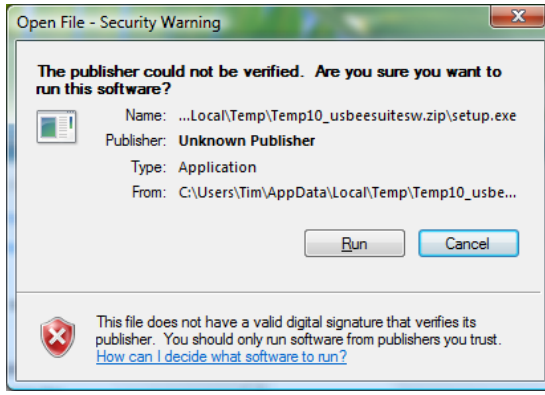
- If you receive messages such as below, press ALLOW or CONTINUE ANYWAY to continue with installation of the software.



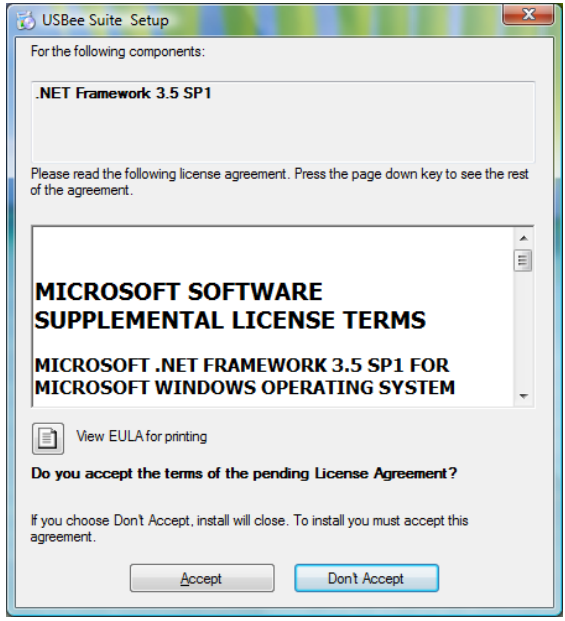
- Run the SETUP.EXE file that is included in the ZIP file that you downloaded to start the installation..



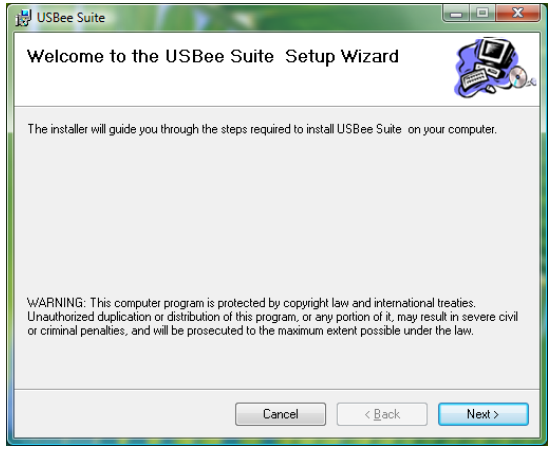
- If you get the following warning, click RUN to continue with the installation.



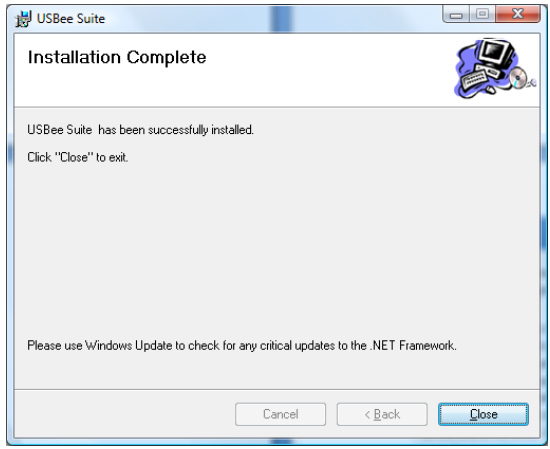
- The first part of the installation installs Microsoft requirements, including Microsoft .NET Frameworks Version 3.5 Service Pack 1. If you do not have this on your PC it will install it for you as shown below – click Accept to install the .NET Frameworks. This installation takes a LONG time, so please be patient since it is worth the wait! If you already have it installed, you will automatically see the "Welcome to the USBee Suite Setup Wizard" screen.



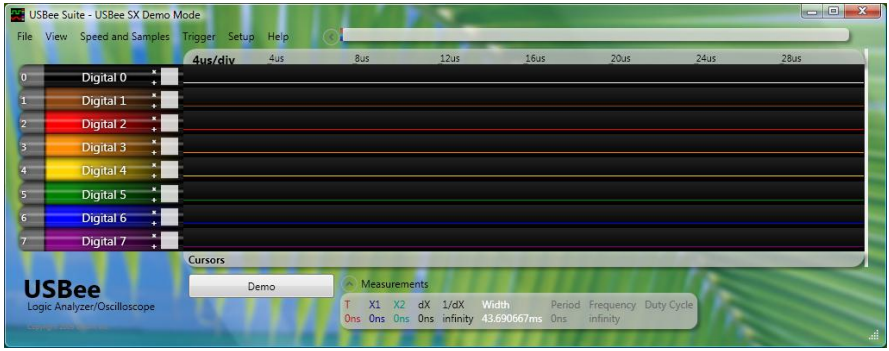
- You will see the Welcome to the USBee Suite Setup Wizard screen as shown below



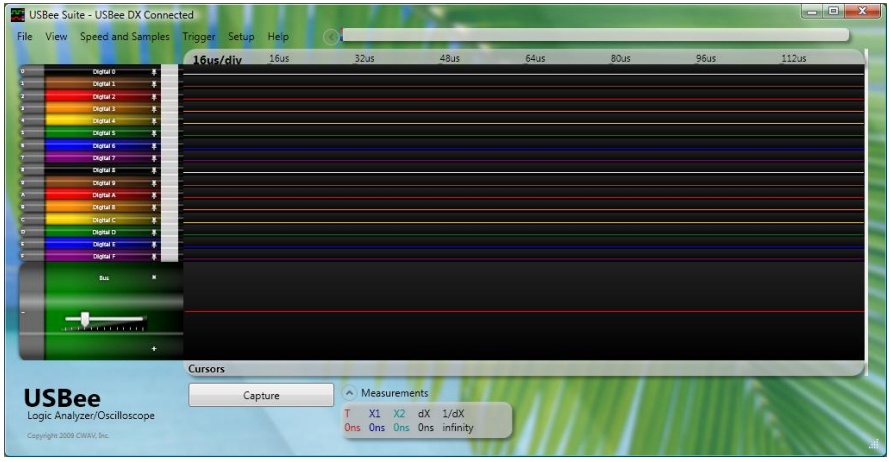
- Follow the instructions (clicking NEXT each time) on the screen to install the USBee Suite software on your hard drive. This may take several minutes. When completed you will see the following screen.



- Click CLOSE and the USBee Suite software is now installed.
- To run the USBee Suite software, choose the USBee Suite icon from the Windows Start Menu. If no USBee is plugged in or installed, you will see the following screen running in Demo Mode (see the top title bar).



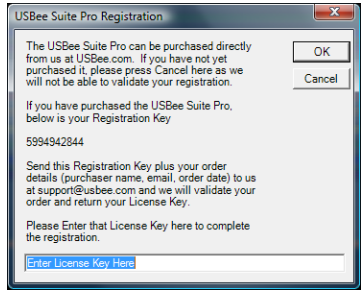
- If you have a USBee plugged in and installed correctly, you will see a screen with all available channels shown. Below is the USBee DX version showing 16 digital channels and 2 analog channels. You can also see that the device is connect (and not in demo mode) in the top title bar.



ENABLING THE USBEE SUITE PRO FEATURES

Users who have purchased the USBee Suite Pro can enable the Pro Features by entering their License Key into the software.

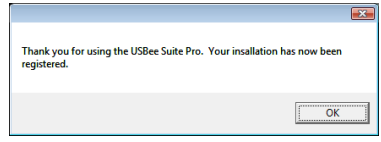
To get your license key, **FIRST** plug in your USBee Test Pod to your PC. **Second**, run the USBee Suite software and click the menu item Setup | Register USBee Suite Pro. You will see the following screen:



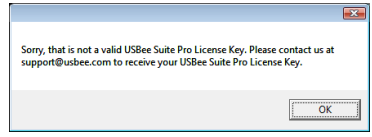
The dialog box contains you Registration Key. Copy this Registration Key into an email and send it to us at support@usbee.com requesting your License Key. You can press Alt-PrtScr to capture the dialog box into your clipboard and then simply paste the image into an email. We will then send you back your License Key.

Enter you License Key into the dialog box and press OK.

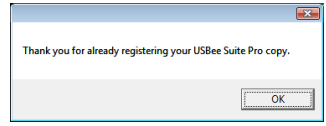
If your License Key is correct you will see this screen and the USBee Suite Pro features will be enabled.



If the License Key is incorrect you will see this screen.



If you have already registered your copy you will see this dialog box indicating that your software has the USBee Suite Pro features enabled.



QUICK START

In order to quickly get up and running using The USBee Suite application, here is a step by step list of the things you need to do to view a waveform trace, after you have installed the software and hardware.

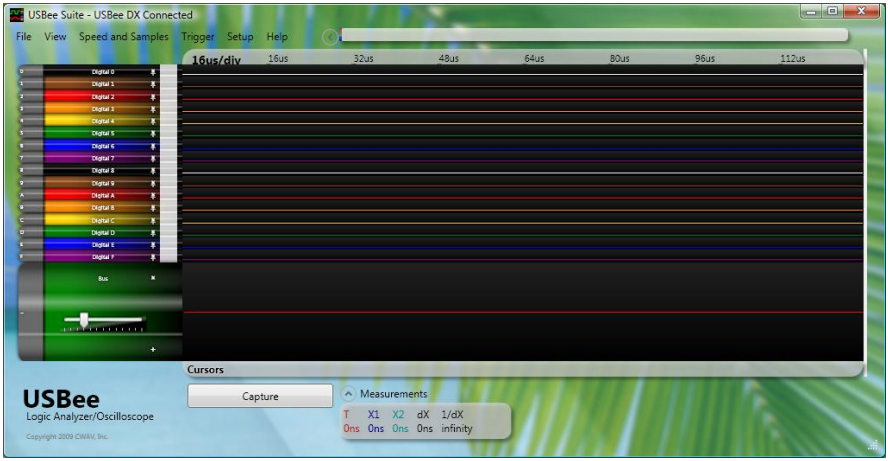
- **Plug in the USBee Pod** - Plug the USBee into your computer USB High Speed port
- **Connect Ground** - Connect the GND wire to the Ground of your circuit you would like to test. You can either use the socket to plug onto a header post, or connect it to one of the mini-grabber clips and then attach it to the Ground.
- **Connect Signals** - Connect any of the USBee inputs on the USBee pod 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 USBee Suite** - Run the USBee Suite Application from the Start Menu.
- **Press the Capture button** - This will capture and display the current activity on all of the signals.
- **View the Waveforms** - 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.
- **Make Measurements** - You can make simple measurements by using the Cursors area (gray bars under 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.

USING THE USBEE SUITE STANDARD

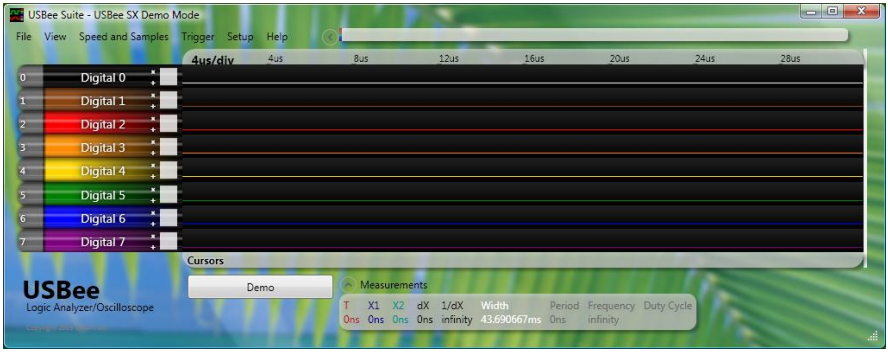
This section details the operation of the USBee Suite Standard application that runs on the USBee SX, AX, ZX or DX.

INITIALIZATION

When the USBee Suite is first run, you will see a screen containing all of the available signals for the USBee Pod plugged into the PC. For example, if you have a USBee DX, you will see the following screen with 16 digital lines and 2 analog signals.



The following screen will show if you have a USBee SX attached showing the available 8 digital channels only.



If you do not have a USBee plugged into or installed on your PC, the software will assume Demo mode and will operate as a USBee SX. In the Demo mode, you can see an example trace capture by clicking the Demo button. This loads a trace that includes a number of serial busses and lets you see how the USBee Suite can decode the bus traffic, manipulate the waveform data, and use the features of the USBee Suite.

The USBee Suite display shows the USBee connection status in the title bar of the application. When a USBee is connected to the computer when the application starts, the title bar indicates what type of USBee is connected.

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.

If you are running in Demo mode and you want to connect to your USBee pod, you must exit the USBee Suite, connect the USBee and then rerun the USBee software.

The USBee Suite maintains its last configuration and will reload that configuration when it is run again. This configuration is located in your \Users\NAME\AppData\Local\USBeeSuite directory where NAME is your username. To reset the software to the initial state you can delete the files in that directory.

ANALYZER SETUP

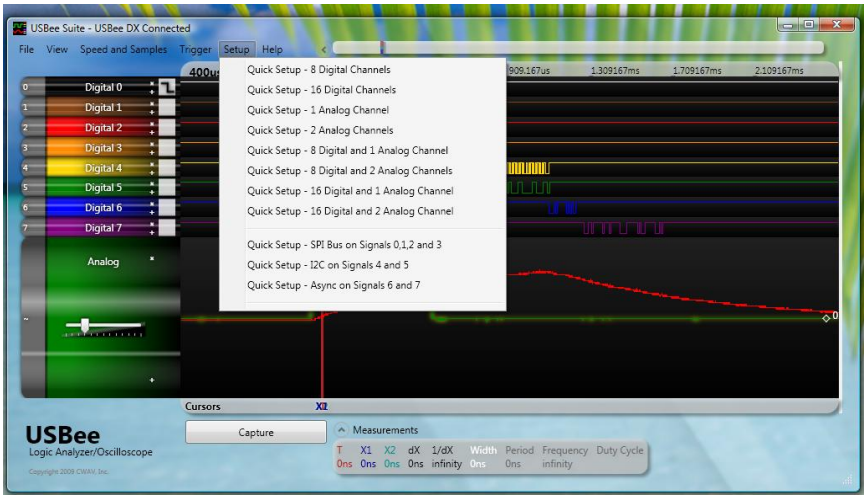
QUICK SETUP CONFIGURATION

The USBee Suite can capture various channels based on the type of USBee plugged in. With a DX plugged in, it can capture 16 channels of digital and 2 channels of analog at the same time. With a USBee ZX you have 8 digital channels. 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.

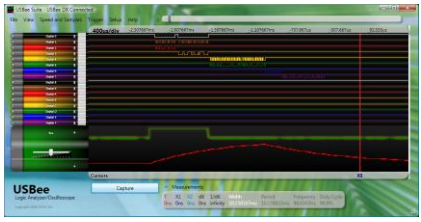
The configurations available are as follows:

Analog Channels	Digital Channels	Max Sample Rate	USBee
0	8	24 Msps	SX, ZX, AX or DX
0	16	12 Msps	DX
1	0	24 Msps	DX
1	8	12 Msps	AX or DX
1	16	8 Msps	DX
2	0	12 Msps	DX
2	8	8 Msps	DX
2	16	6 Msps	DX

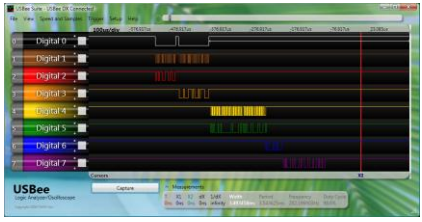
To select a configuration, click **Setup** on the menu and select the Quick Setup configuration of your choice. Below shows the available Quick Setup options for a USBee DX.



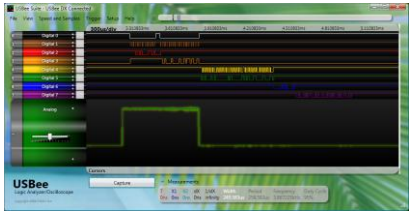
Below are examples of the application in various modes.



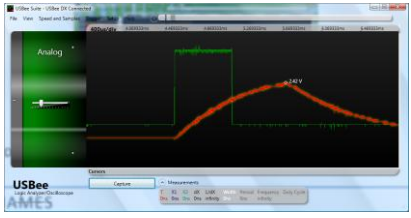
16 Digital–2 Analog Channels



8 Digital–0 Analog Channels



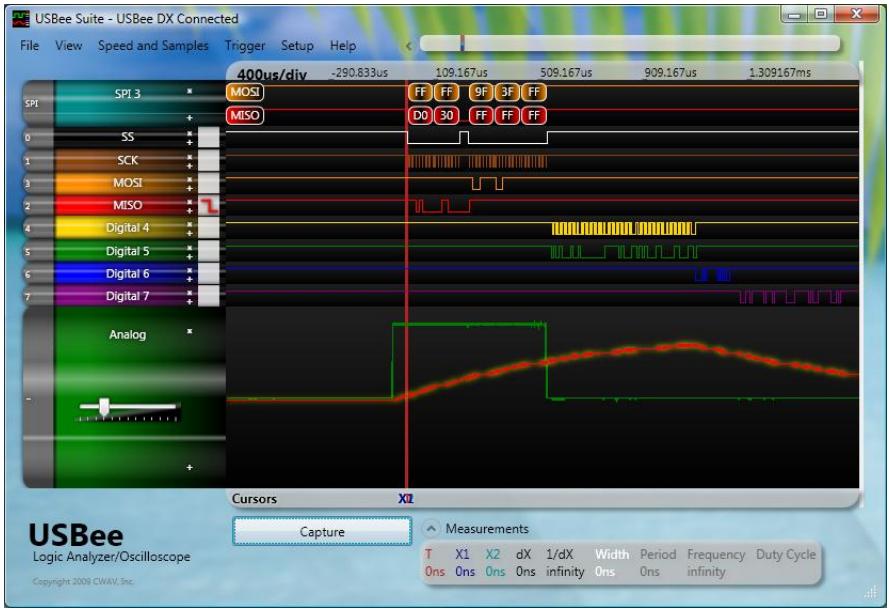
8 Digital–1 Analog Channels



0 Digital–2 Analog Channels

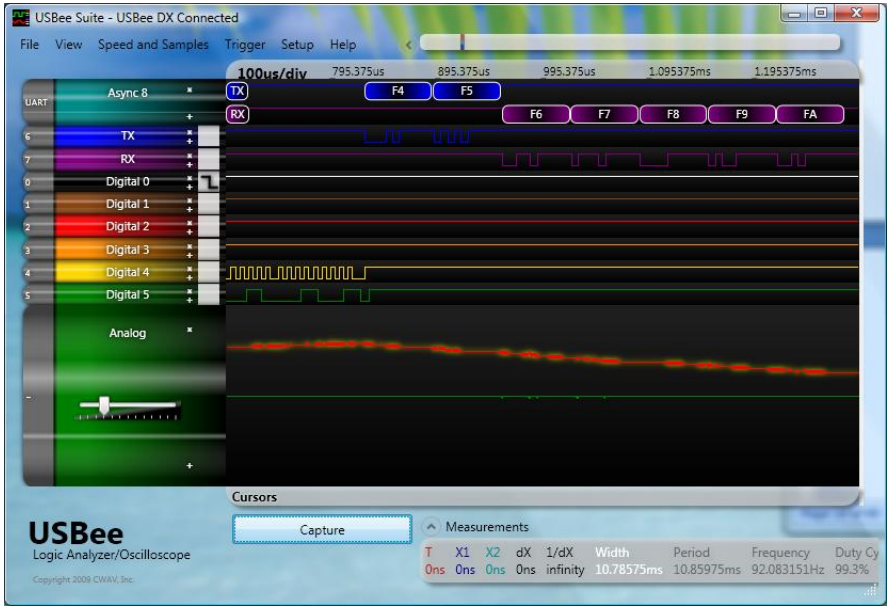
There are also three other Quick Setup features that let you instantly setup an I2C, SPI or ASYNC decoder line.

The **Quick Setup – SPI** configures the first 4 lines to be an SPI bus with the SS, SCK, MOSI and MISO lines. It also adds a decoder line to the screen with this data decoded as below.



The **Quick Setup – I2C** configures the signals 4 and 5 to be an I2C bus with the SDA and SCL lines. It also adds a decoder line to the screen with this data decoded as below.

The **Quick Setup – ASYNC** configures the signals 6 and 7 to be a full duplex ASYNC bus with the TX and RX lines. You will need to change the baud rate, data bits and parity to match your bus. It also adds a decoder line to the screen with this data decoded as below.

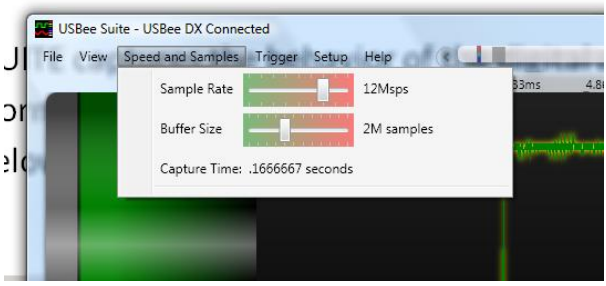


SIGNAL NAMES

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

BUFFER SIZES AND SAMPLE RATE SETTINGS

The USBEE SUITE captures the behavior of the digital and analog signals and displays them as “traces” in the waveform window. The Speed and Samples menu lets you choose how the traces are captured. Below shows the Speed and Samples menu.

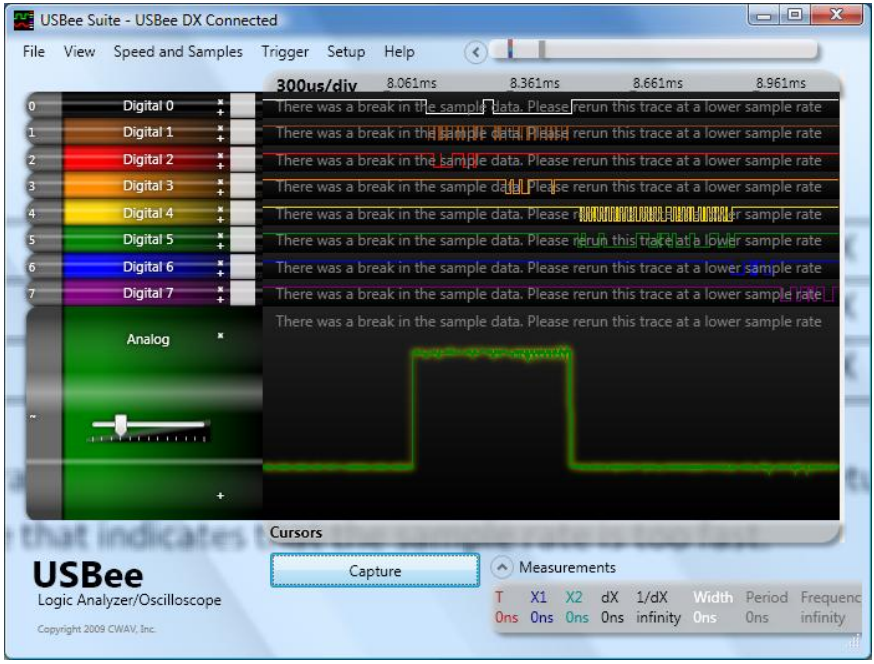


The **Buffer Size** lets you select the size of the Sample Buffer that is used. For each trace, the buffer is completely filled, and then the waveform is displayed. You can choose buffers that will capture the information that you want to see, but remember that the larger the buffer, the longer it will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

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	USBee
0	8	24 Msps	SX, ZX, AX or DX
0	16	12 Msps	DX
1	0	24 Msps	DX
1	8	12 Msps	AX or DX
1	16	8 Msps	DX
2	0	12 Msps	DX
2	8	8 Msps	DX
2	16	6 Msps	DX

If you choose a sample rate that is too fast for your PC configuration and setup, you will receive a warning after each trace that indicates that the sample rate is too fast.

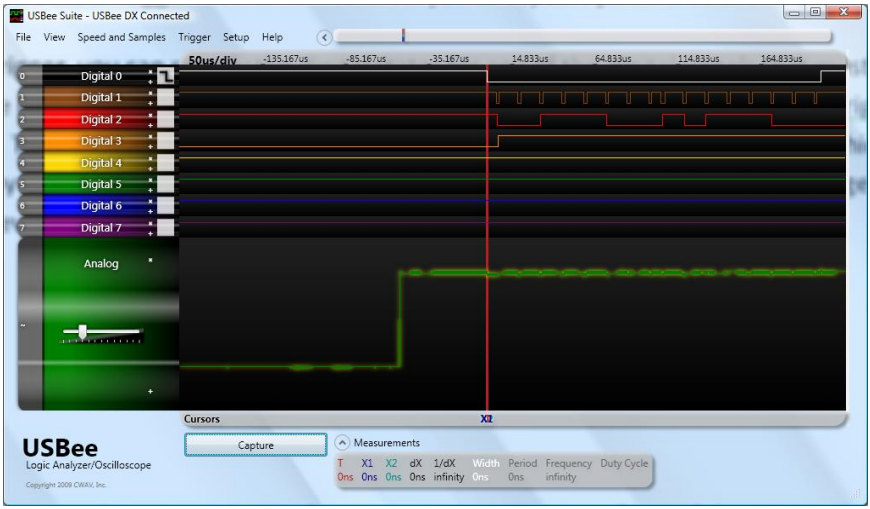


If you continue to press the Capture button without lowering the sample rate, the USBee Suite software will automatically lower the sample rate until it reaches a rate that is at the correct level for your configuration.

SETTING TRIGGERS

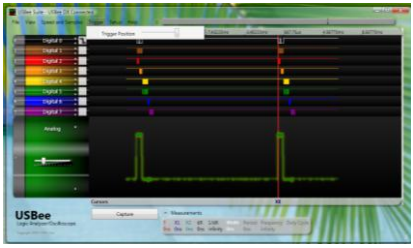
The USBee Suite uses a Trigger mechanism to allow you to capture just the data that you want to see.

For a **Digital trigger**, you can specify the digital states for any of the digital 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 0, 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 rising edge to falling edge.

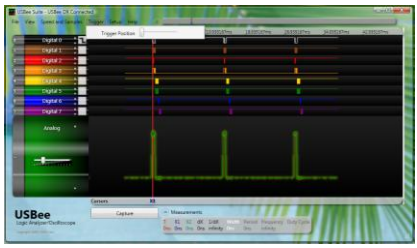


The waveforms are shown with a trigger position which represents where the trigger occurred. This sample point is marked on the waveform display with a Vertical red cursor line and a "T" in the horizontal cursors bar.

You can use the **Trigger Position** menu 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

CAPTURING WAVEFORM DATA

Press Capture to start capturing the waveform data from your hardware design. If you are running in Demo mode, the button reads Demo.

It will look for the trigger condition, fill the buffer with samples of the signals and stop.

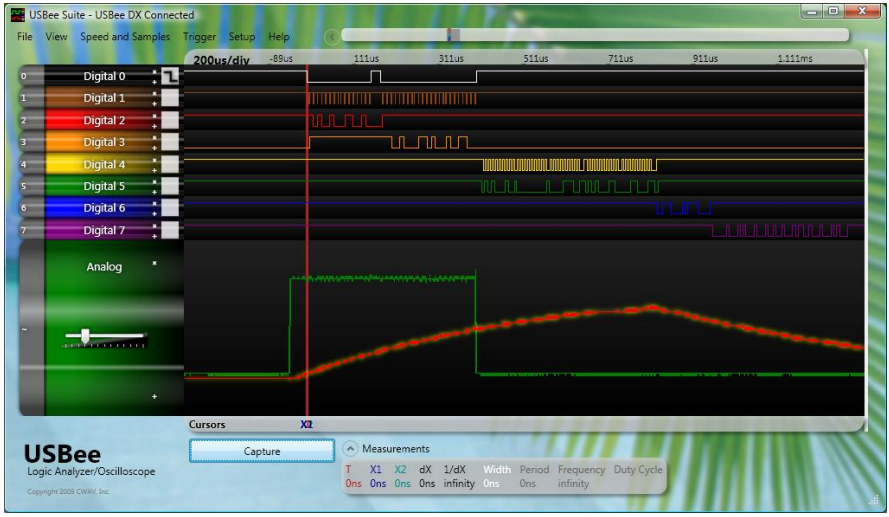
If you would like to stop the capture before it is completed just press the same button again (which reads STOP during a capture).

After a trace is captured, the waveform data is gathered, decoded (if needed) and displayed in the waveform window.

VIEWING CAPTURED DATA

SCROLLING, ZOOMING AND PANNING WAVEFORMS

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.



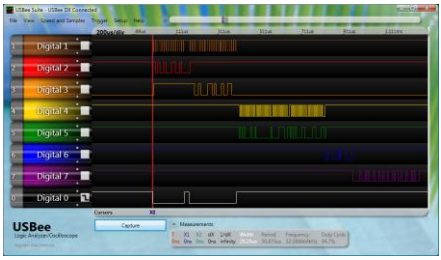
The position of the waveform defaults to show the actual trigger position in the center of the screen after a capture. However, you can move the display to see what happened before or after the trigger position.

To **Scroll the Waveforms in Time** left and right, you can use the overview bar at the bottom of the waveform display, 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 simply click and drag the waveform itself by selecting and dragging using the mouse.

To **Zoom In** or **Zoom Out**, or other words change the number of **Seconds per Division,** you can use the scroll wheel or single click on the waveform. To zoom in, scroll up or click the left mouse on the waveform window. To zoom out in time, scroll down or click the right mouse button on the waveform window. To change the number of **Volts per Division** for an analog channel, highlight the channel you want to change, hold the left mouse button down and use the scroll wheel. You can also highlight the signal and use the slider bar to the left of the waveform.

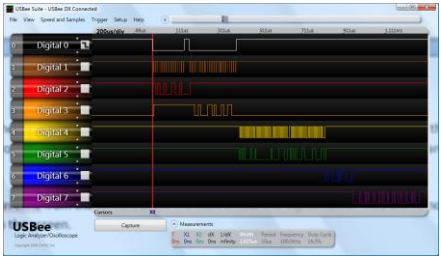
To **Stretch and Shrink** the display, you can click and drag the edges of the application to the size you want. All waves will scale to fit. Below you see two examples of different size displays.

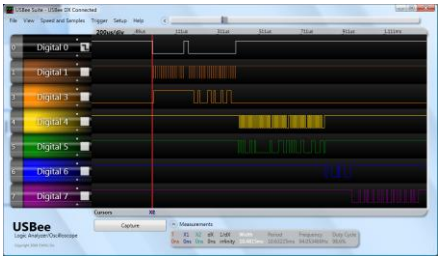


MODIFYING WAVE LINES

Each line on the display is called a **Waveline**. Wavelines can be modified to your liking so that you see the data you need to solve your problem. You can delete, add, move, or reconfigure any waveline.

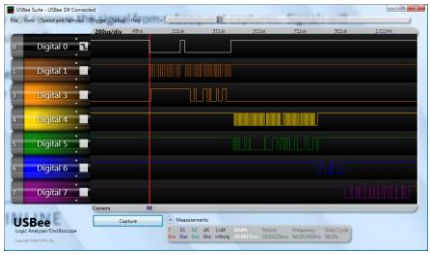
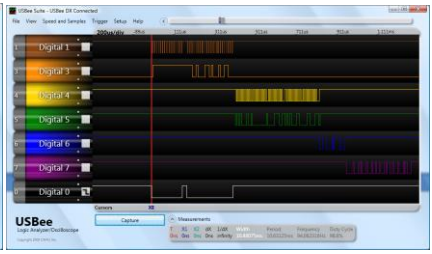
To **Delete a Waveline** from the screen, press the little X near the signal name. This will remove the waveline from the screen. Below shows the USBee Suite after deleting the Digital 2 waveline.



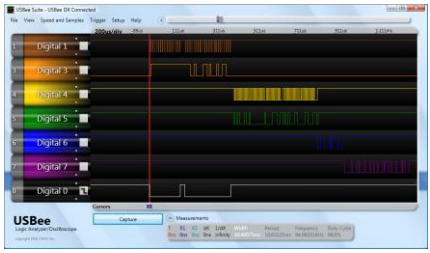
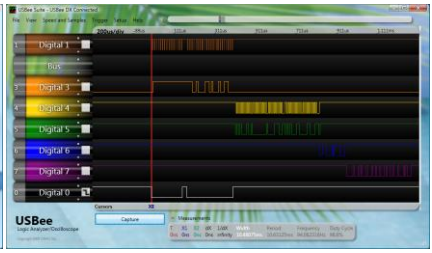


Removing a signal from the screen may not remove the signal from subsequent captures. Signals will only be eliminated from captures if all signals from a given byte lane are removed.

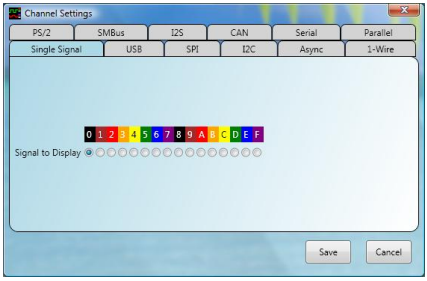
To **Move a Waveline**, simply click on the gray tab on the left and drag it to the new position. Below shows the Digital 0 signal moved to the bottom.

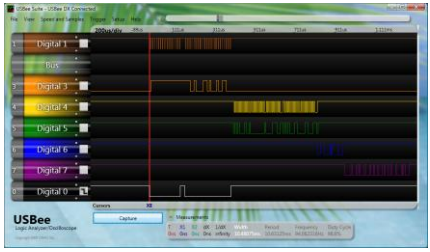
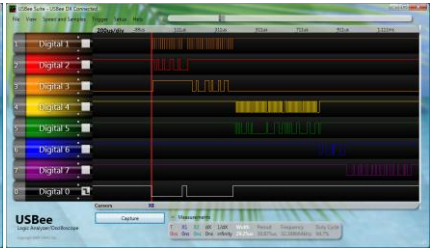
To Add a Waveline, click on the small + sign above where you want to insert the new waveline. Below you see a new waveline inserted after the first waveline.

When you insert a new waveline the Channel Selection dialog box appears for you to choose the settings for that waveline. Below shows the Channel Settings Dialog Box.



Once you select the properties of the new waveline, it will be displayed with the other signals. Below shows a new line added that shows the Digital 2 single signal.

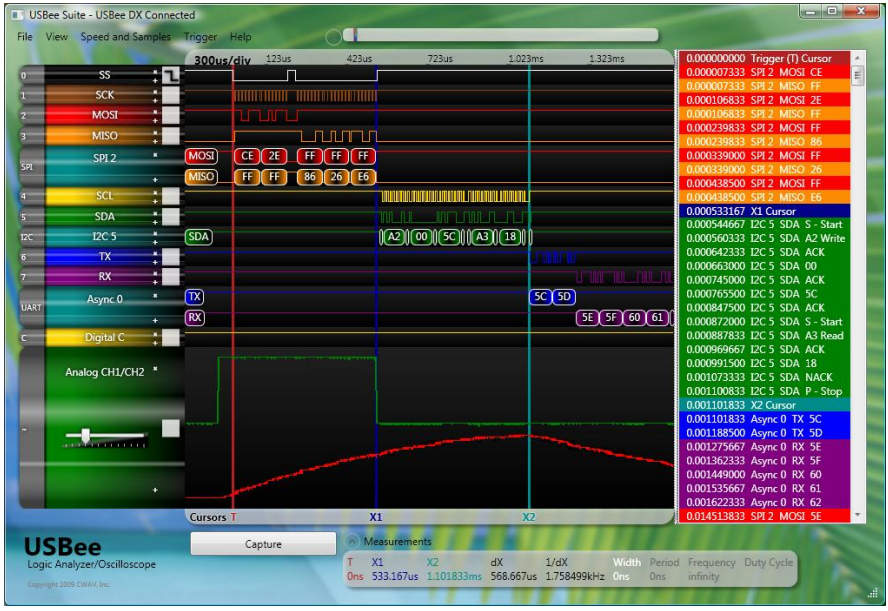



We will go through creating Bus wavelines that decode bus traffic in-line in the next section.

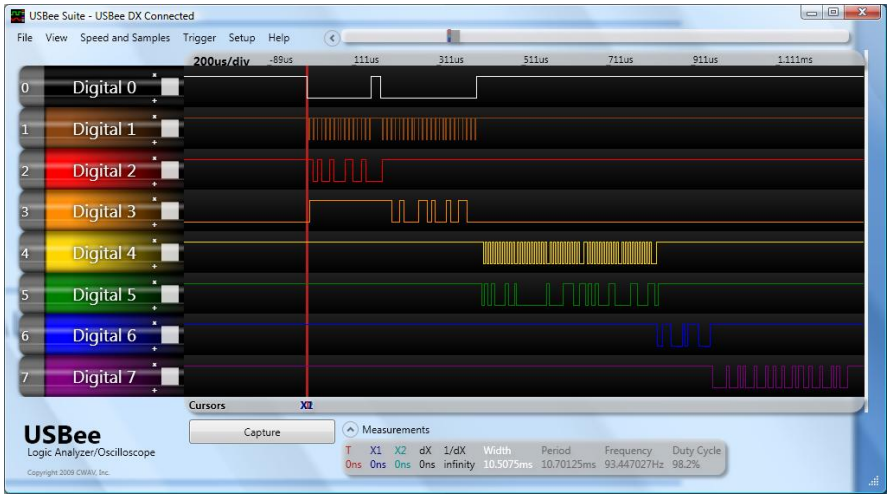
To **Modify an Existing Waveline** click on the grey tab on the left of the waveline. This will bring up the Channel Settings dialog box and allow you to change the settings for that line. Below we modified the last line to show Digital 2 signal instead of the Digital 0 signal.

DECODING BUS TRAFFIC INLINE

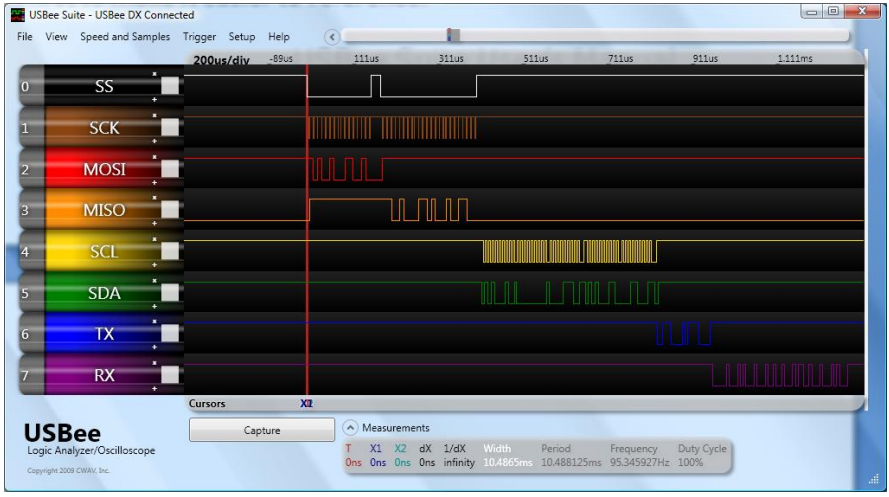
The USBee Suite software can decode certain types of serial busses automatically and display that information in-line with the waveforms. Below is an example of a screen that shows 3 different serial bus decoders at the same time, one SPI, one I2C, and one full duplex ASYNC channel.



We will go through an example that shows how to setup various busses. We start with a capture of 8 digital lines that have a number of busses included.



In this example, we will name the bus signals first to make it easier to reference.

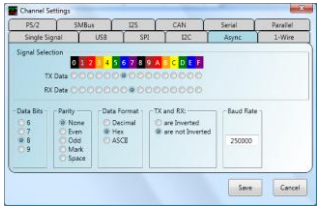


Now we will add an SPI bus which is made up of the first 4 signals. We press the small + sign near the MISO label to insert the waveline below that line. We then get the Channel Settings Dialog and choose the SPI tab. The following screen is then displayed.

From here we select the parameters for this bus (shown above) and press Save. Once we press Save, the line is added to the screen, the current trace is decoded, and the decoded information is shown on the waveline.

We then add the I2C bus using the Channel Settings dialog box as below with the resulting waveline.

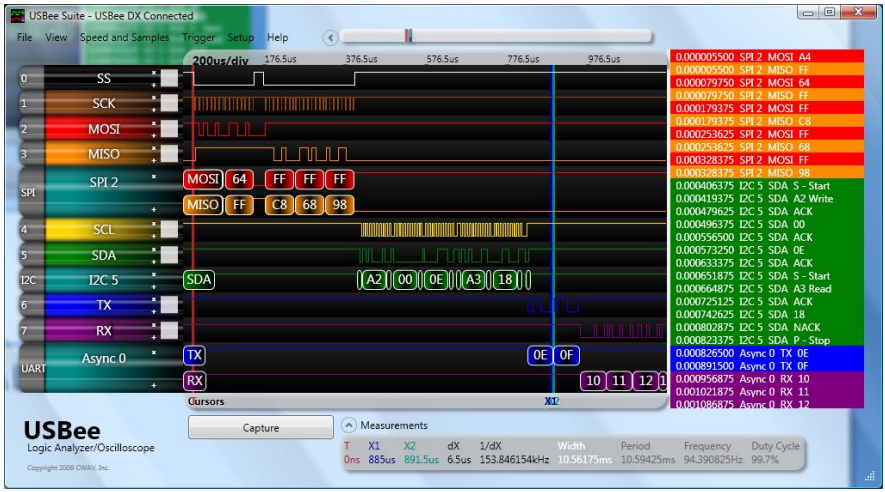
We then add the Async bus using the Channel Settings dialog box as below with the resulting waveline.



Each bus type has various parameters that can be tailored to get the data out of your bus the way you need it.

DECODED DATA LIST

You can see the decoded list data in vertical format using the View/Show Decode Bus Listing menu item. This opens a window on the right side of the screen that displays the decoded data in vertical format. The data shown is the data that is decoded from the left side of the waveform screen. This data is synchronized to the waveforms as you pan and zoom.



You can hide or change the width of the Decoded Bus Listing window by clicking and dragging the vertical line to the left of the window.

MANUAL MEASUREMENTS AND CURSORS

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

The **X1 and X2 Cursors** are placed on any horizontal sample time. This lets you measure the time at a specific location or the time between the two cursors. To place the X cursors, move the mouse to the gray Cursors 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.

These cursors will snap to the exact edge of a digital signal when the mouse moves close to the edge. This lets you easily get exact measurements between edges of signals.

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
- **dX** – time difference between X1 and X2 cursors
- **1/dX** – the frequency computed using the period between X1 and X2 cursors

INSTA-MEASUREMENTS

The Insta-Measure feature lets you quickly and accurately measure events and levels by simply hovering the mouse over a signal and without placing cursors.

Inta-Measurements available are as follows:

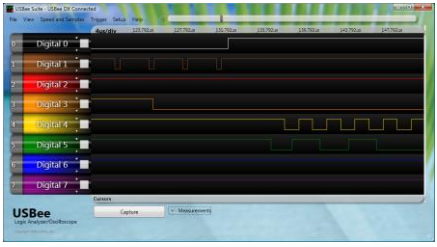
Analog Insta-Measurements

- Voltage At Cursor

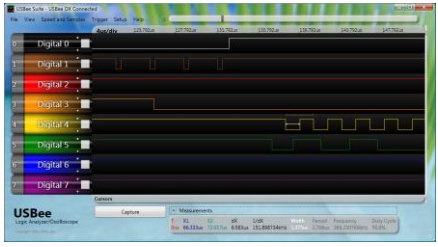
Digital Insta-Measurements

- Width
- Period
- Frequency
- Duty Cycle

To turn on Insta-Measurements, expand the Measurements window by clicking the expander arrow.



Insta-Measure Off



Insta-Measure On

BUS DECODING OPTIONS

The USBee Suite software has a powerful embedded bus decoder feature that allows you to quickly analyze the contents of embedded communications captured by the pod. This section details each of the available bus types and the parameters required for proper setup.

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

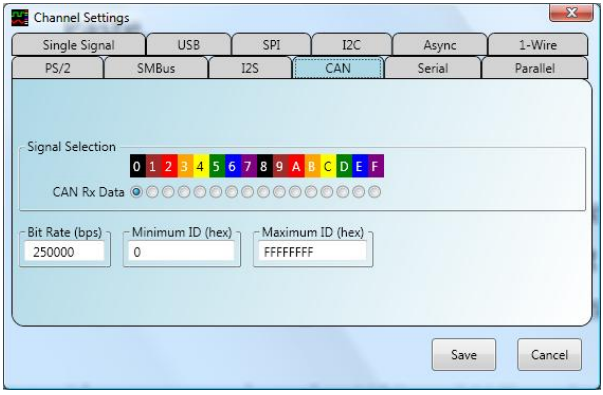
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 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 grey tab on the left of the signal names on the main application screen.



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

The bus traffic will be decoded as in the following screen.

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

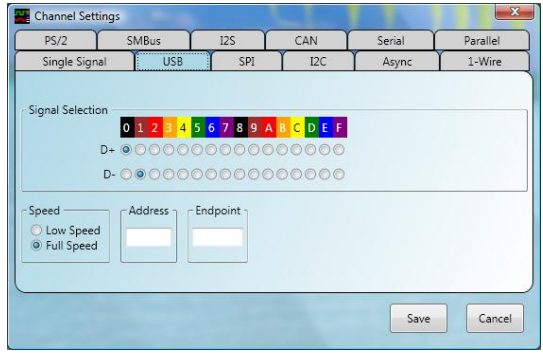
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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 Test Pod before connecting the signals to the unit.

Connect two of the 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 grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the D+ and D- signals and what speed the bus is operating at. 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.

The bus traffic will be decoded as in the following screen.

I2C BUS SETUP

The I2C Bus Decoder takes the captured data from a I2C bus.

Hardware Setup

To use the Decoder you need to connect the USBee 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 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 Test Pod before connecting the signals to the unit.

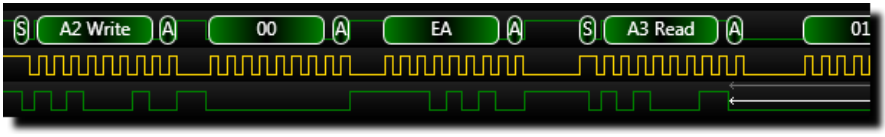
The I^2C Bus Decoder connects to the SDA and SCL lines of the I^2C bus. Use one signal as the SDA data line and one signal as the SCL clock line. Also connect the GND line to the digital ground of your system. Connect these signals to the I^2C bus using the test clips provided.

Software Setup

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

On the above dialog box, select the SDA and SCL signals.

The bus traffic will be decoded as in the following screen.



ASYNC BUS SETUP

The Async Bus Decoder takes the captured data from an asynchronous bus (UART).

Hardware Setup

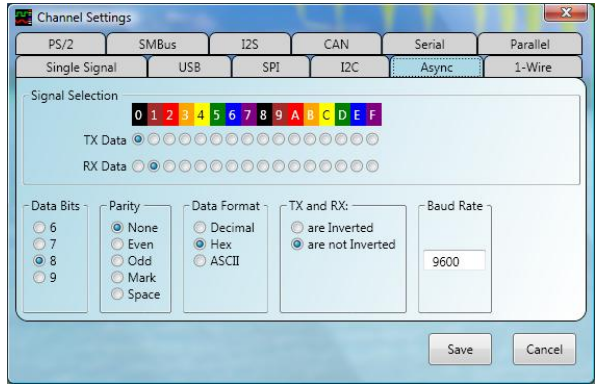
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 Test Pod before connecting the signals to the unit.

The Async Bus Data decoder 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.

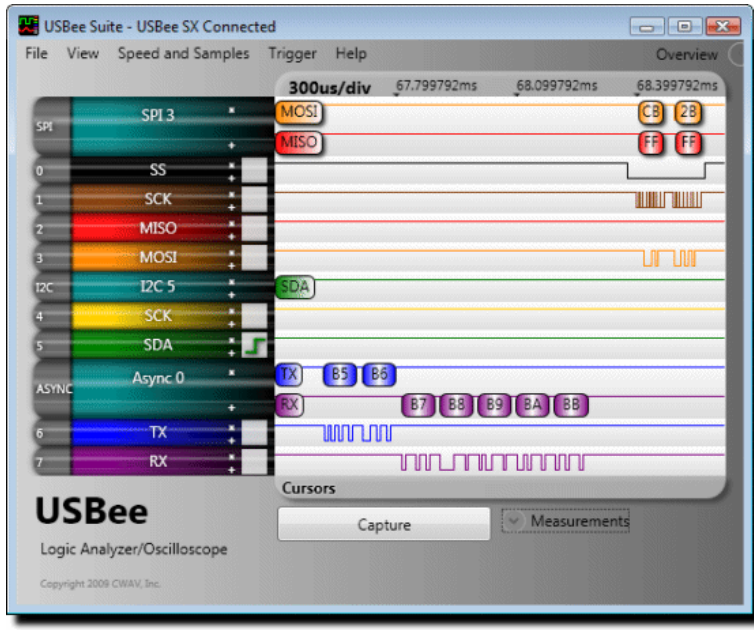
Software Setup

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



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

The bus traffic will be decoded as in the following screen.


USBee Suite - USBee SX Connected
File
View
Speed and Samples
Trigger
Help
Overview
300us/div
67.799792ms
68.099792ms
68.399792ms
SPI
SPI 3
MOSI
MISO
CB
2B
FF
FF
0
SS
1
SCK
2
MISO
3
MOSI
I2C
I2C 5
SDA
4
SCK
5
SDA
ASYNC
Async 0
TX
B5
B6
RX
B7
B8
B9
BA
BB
6
TX
7
RX
Cursors
USBee
Logic Analyzer/Oscilloscope
Capture
Measurements
Copyright 2009 CWAV, Inc.

PARALLEL BUS SETUP

The Parallel Bus Decoder takes the captured data from a parallel bus. The Parallel Bus decoder is also a way to capture the data using an external clock.

Hardware Setup

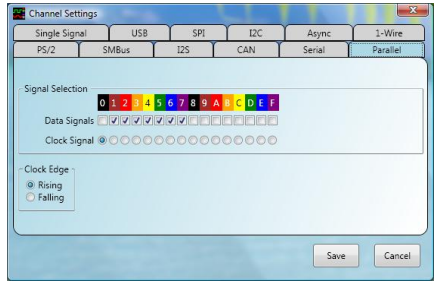
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 Test Pod before connecting the signals to the unit.

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

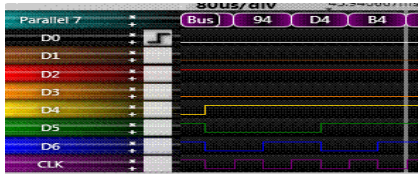
Software Setup

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



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

The bus traffic will be decoded as in the following screen.



1-WIRE BUS SETUP

The 1-Wire Bus Decoder takes the captured data from a 1-Wire bus.

Hardware Setup

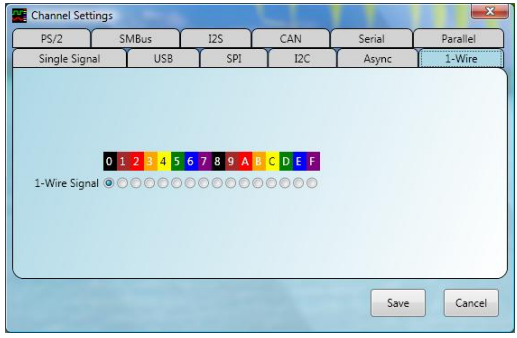
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 Test Pod before connecting the signals to the unit.

The 1-Wire Bus Data decoder 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 grey tab on the left of the signal names on the main application screen.



On the above dialog box, select the signal running your 1-Wire protocol.

SPI BUS SETUP

The SPI Bus Decoder takes the captured data from an SPI bus.

Hardware Setup

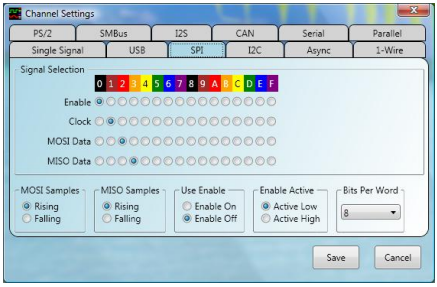
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 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.

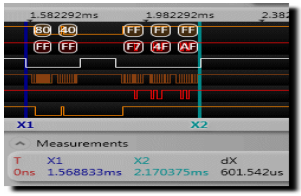
Software Setup

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



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

The bus traffic will be decoded as in the following screen.



SM BUS BUS SETUP

The SM Bus Decoder takes the captured data from an SM bus.

Hardware Setup

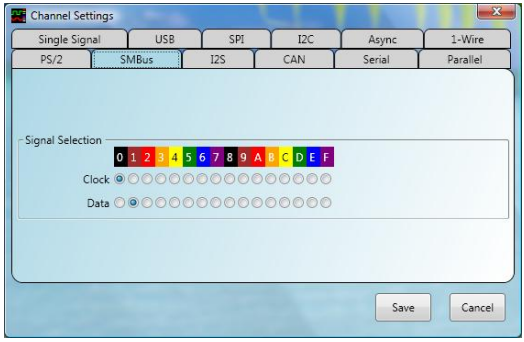
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 Test Pod before connecting the signals to the unit.

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

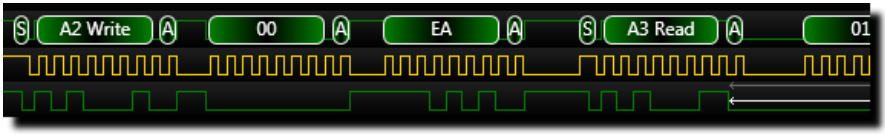
Software Setup

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



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

The bus traffic will be decoded as in the following screen.



SERIAL BUS SETUP

The Serial Bus Decoder takes the captured data from a Serial bus. 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

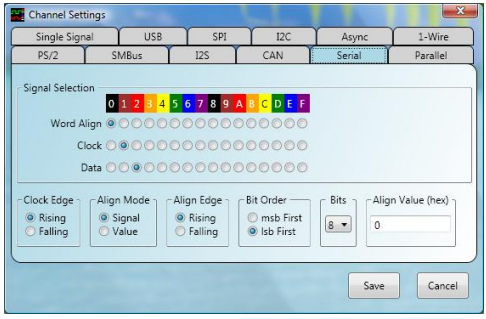
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 Test Pod before connecting the signals to the unit.

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

Software Setup

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



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

The bus traffic will be decoded as in the following screen.

I2S BUS SETUP

The I2S Bus Decoder takes the captured data from an I2S bus.

Hardware Setup

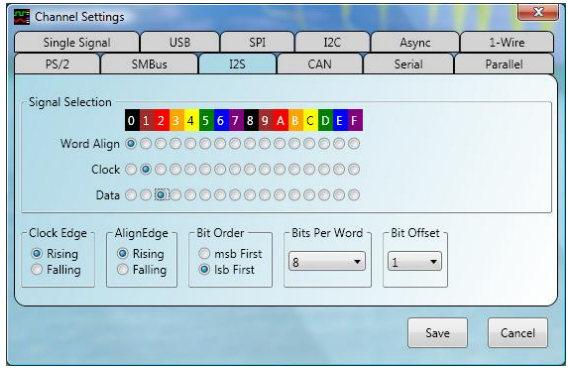
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 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.

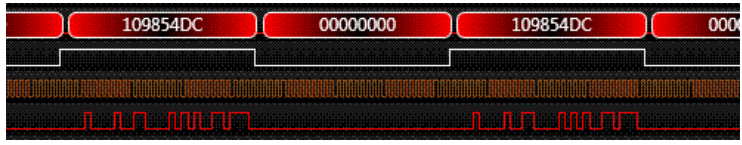
Software Setup

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



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

The bus traffic will be decoded as in the following screen.



PS/2 BUS SETUP

The PS/2 Bus Decoder takes the captured data from an PS/2 bus.

Hardware Setup

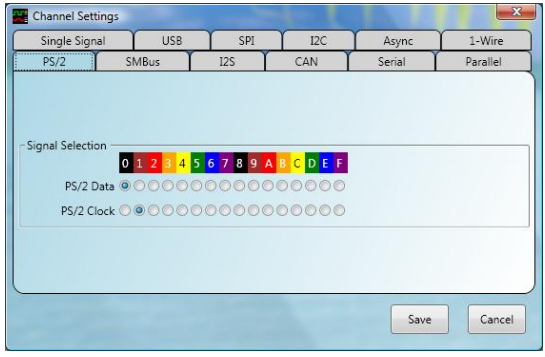
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee 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 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 grey tab on the left of the signal names on the main application screen.



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

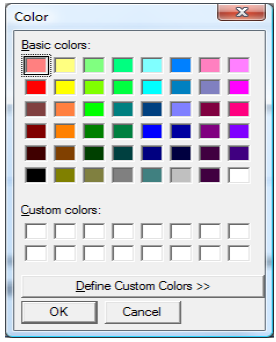
The bus traffic will be decoded as in the following screen.

SETTING VIEWING PREFERENCES

The USBee Suite has many ways that you can customize the display of your data, beyond the placement of the waveforms.

CURSOR COLORS

You can change the color of the Trigger, X1 and X2 cursors using the View Menu. When chosen you will see the Color Selection dialog box below. To change the colors back to their original state, use the View/Reset Colors To Default menu item.

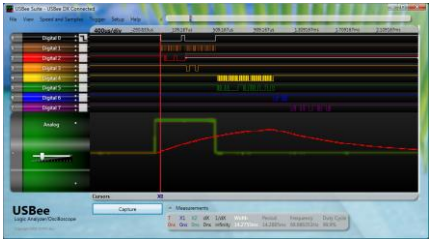


BACKGROUND COLOR

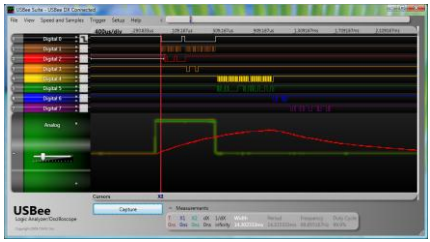
The background of the waveform screen can be set to white or black using the View | Background White or View | Background Black menu items.

GLASS ON VISTA

Windows Vista has added the ability to have a Glassy appearance on applications. If your system is capable of this glassy look you can use the View/Show Glass If Possible menu item to turn it on. If it is not possible, or you turn off Glass, the display shows a grey background.



With Glass On



With Glass Off

FILE OPERATIONS

Using the File menu you can start a New file, Save trace data, Open previously saved traces and Export trace data to other file formats.

CREATING A NEW FILE

To start a new file, choose File/New. This will configure the screen to the default state with all available channels enabled.

SAVING A CAPTURE FILE

After capturing a trace, you can save it to disk using the File/Save As menu item. This saves all trace data, cursor positions and screen format. The files can be saved in either Uncompressed format (.usbeesuite extension) or in Compressed format (.usbeecomp extension) to reduce drive space requirements. Saving and Opening large buffer sizes can take a while to perform the compression and saving but can greatly reduce disk space.

OPEN AND EXISTING CAPTURE FILE

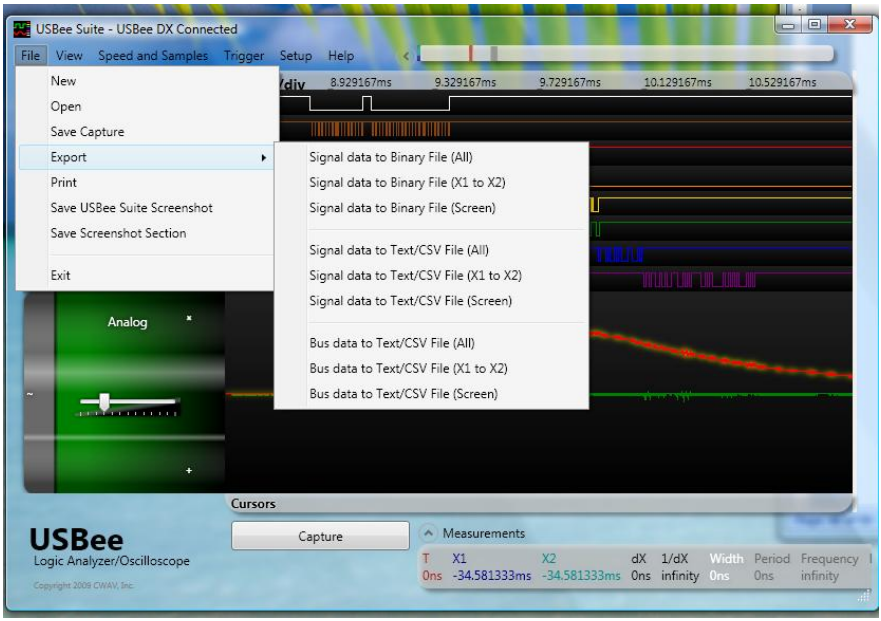
To view a previously saved capture file, us the File/Open menu item. This will load the trace data and screen format including decoder setup. Opening files with large buffer sizes can take a while to decompress and display.

RECENTLY USED FILE LIST

The USBee Suite maintains a recently used file list that allows you to quickly load any of the last 5 previously used files. Simply click on the file name in the File, Recently Used file list to open it.

EXPORTING CAPTURED DATA TO A FILE

Since the compressed trace files are not in easily useable format, you can use the File/Export menu items to save the trace and decoded data into formats that are easy to use.



The available options for exporting are:

- Save the Signal Data to a Binary File
- Save the Signal Data to a Text/CSV File
- Save the Bus Data to a Text/CSV file

You can specify the range of samples to export by using the All, X1 to X2, or Screen versions. Screen will output all samples viewed on the current screen, X1 to X2 will output all samples between the X1 and X2 cursor, and All will output all samples in the sample buffer. Choosing All will create VERY large files, so use with caution.

EXPORT SIGNAL DATA TO BINARY FILE

When exporting signal data to a binary file, each sample is made up of 4 bytes. Each sample was taken at the sample rate that was set at the time of capture. A single sample (4 bytes) is formatted as follows:

Byte	Description
1	Digital channels 0 to 7 (lsb= signal 0, msb = signal 7)
2	Digital channels 8 to F (lsb= signal 8, msb = signal F)
3	Channel 1 Analog voltage in 78.125mV steps (0=-10V, 128 = 0V, 255 = +9.92V)
4	Channel 2 Analog voltage in 78.125mV steps (0=-10V, 128 = 0V, 255 = +9.92V)

EXPORT SIGNAL DATA TO TEXT/CSV FILE

When exporting signal data to a text/csv file, each sample is output to a single line with each signal separated by a comma. Each sample was taken at the sample rate that was set at the time of capture. An example output file is formatted as follows showing a header that specifies the column labels and which signal is associated:

```
Time,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
0.008739333,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008739500,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008739667,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008739833,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740000,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740167,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740333,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740500,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740667,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
0.008740833,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1
```

EXPORT BUS DATA TO TEXT/CSV FILE

When exporting bus data to a text/csv file, each decoded element is output to a single line with each field separated by a comma. An example output file is formatted as follows showing a header that specifies the column labels and which signal is associated:

```
Time(seconds),Bus Name,Signal Name,Data
0.000007167,SPI 3,MOSI,FF
0.000007167,SPI 3,MISO,24
0.000106333,SPI 3,MOSI,FF
0.000106333,SPI 3,MISO,A4
0.000238667,SPI 3,MOSI,48
0.000238667,SPI 3,MISO,FF
0.000338000,SPI 3,MOSI,A8
0.000338000,SPI 3,MISO,FF
0.000437167,SPI 3,MOSI,18
0.000437167,SPI 3,MISO,FF
0.000542667,I2C 5,SDA,S - Start
0.000558500,I2C 5,SDA,A2 Write
0.000640333,I2C 5,SDA,ACK
0.000661167,I2C 5,SDA,00
0.000743000,I2C 5,SDA,ACK
0.000763667,I2C 5,SDA,0D
0.000845500,I2C 5,SDA,ACK
0.001098833,I2C 5,SDA,P - Stop
```

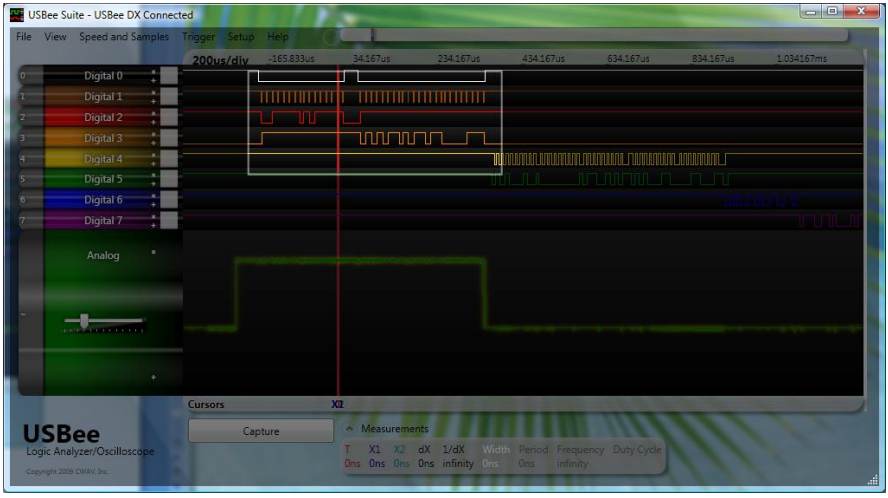
PRINTING

To print an image of the current screen, choose File/Print from the menu.

CREATING SCREEN SHOTS

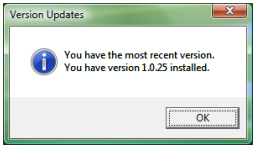
An easy way to create documentation is to take screen shots, or portions of the screen image, and save them to graphics files. You can save the entire USBee Suite application window to a file using the File/Save USBee Suite Screenshot menu item. This lets you save the image as a BMP, JPG, PNG, GIF, TIF, or WMF file to be used by your favorite graphics program.

You can use the File/Save Screenshot Section menu item to select just a portion of the screen to save. Use the left mouse button to start a rectangle that selects the region to save. When you let up on the button it will prompt you for the filename to save the image as.



SOFTWARE UPDATES

New versions of the USBee Suite software are posted on the USBee.com web site. To have the USBee Suite software check if a new version exists, use the Help/Check for Updates menu item. It will connect to the USBee.com server and determine if there is a newer version available for download. If you are up to date, the following screen will appear.



DEVELOPING YOUR OWN CUSTOM DECODERS

The USBee Suite allows you to create your own custom protocol decoders.

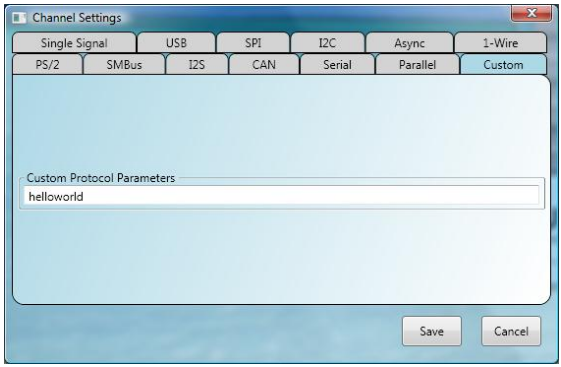
To implement a custom decoder you must create a Class Library (DLL) using the code below as an example. You can build this Class Library using the free Microsoft Visual Studio 2008 Express or newer. Our example is in Visual Basic, but can easily be ported to C or other language supported in Visual Studio.

We will first show how to use a Custom Decoder and then show how to design one.

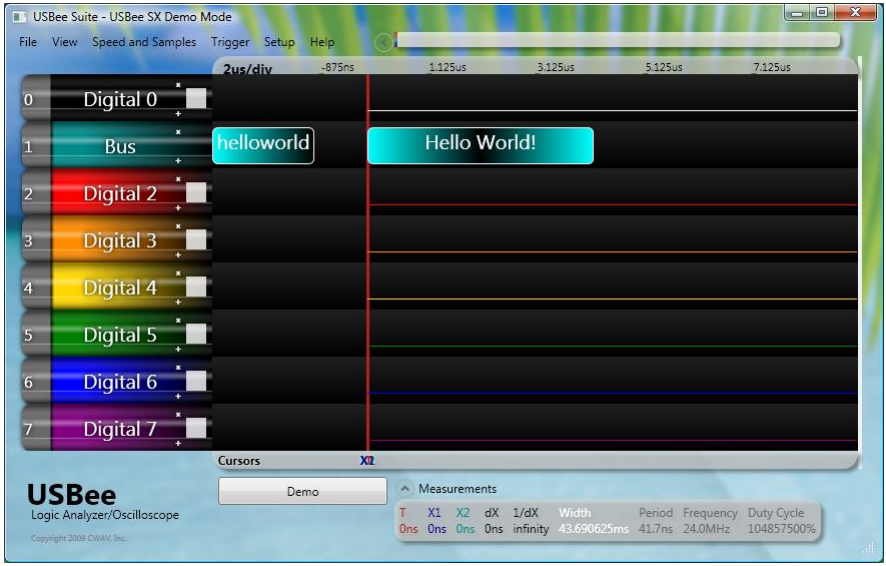
USING THE CUSTOM DECODER

Using the Custom Decoder that you build is simple. To select to use your Custom Decoder you select the Custom tab in the Channel Settings dialog box. You can then enter a set of parameters that are sent to your decoder. These parameters can specify anything you may need to determine how to decode your protocol, including which protocol, which signals to use, baud rates, inversions, etc. and is purely defined by you.

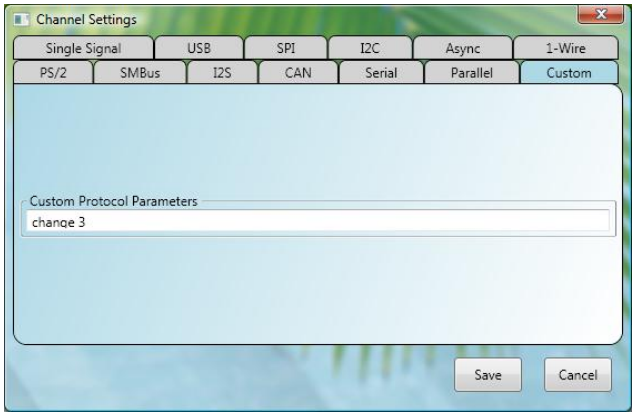
Below is the Channel Setting screen showing that we want to use our Custom Decoder on this waveline. We are also passing the text string “helloworld” to the decoder when it runs.



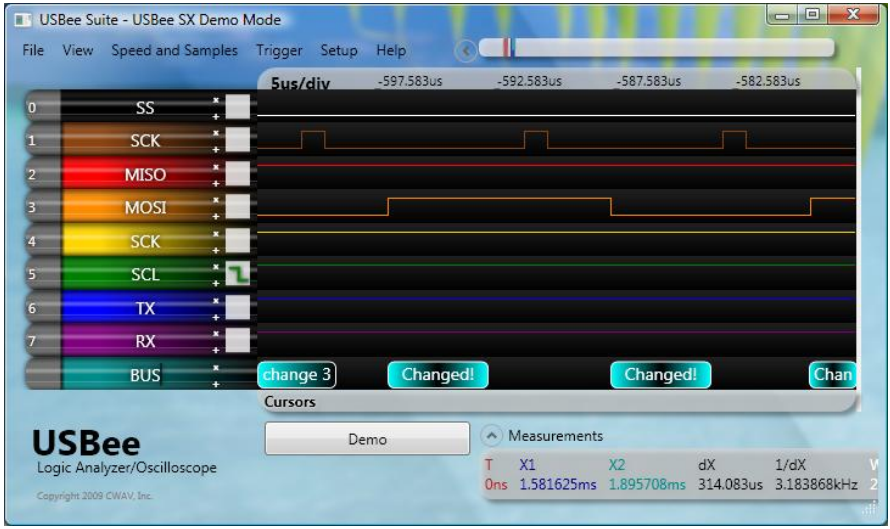
When we press Save our Decoder is run, passing the parameters to it and the resulting Entries are displayed. Below is the output from our VERY simple Hello World decoder which puts a “Hello World!” at the first sample.



On a different Waveline you can specify a different set of Parameters (in this case “change 3”) that indicate to the Custom Decoder to perform an entirely different decode.



As in our example decoder, this “Change” indicates to place an Entry at each change of state of the specified signal (in this case 3). The resulting display is as follows.



Obviously much more complicated protocols can be decoded using these simple methods of parameter passing and Entry displaying.

BUILDING THE CUSTOM DECODER

To implement a custom decoder you must create a Class Library (CustomUSBeeSuiteDecoder.DLL) using the code below as an example. This example code is also included when you install the USBee Suite software in the \Program Files\CWAV Inc\USBee Suite\CustomUSBeeSuiteDecoder\ directory. You can build this Class Library using the free Microsoft Visual Studio 2008 Express. Our example is in Visual Basic, but can easily be ported to C or other language supported in VS2008.

The main function of a Custom Decoder code is below.

1. Receive parameters for the protocol from the User Interface
2. Access the sample data and decode the protocol based on the parameters
3. Output “Entries” that consist of a Start Sample, End Sample and a Text String

Once you create your own CustomUSBeeSuiteDecoder.DLL file, you simply copy your new file over the one that was provided with the original install in the \Program Files\CWAV Inc\USBee Suite directory. You may need to locate this file on your system and have administrator rights in order to replace it.

EXAMPLE CLASS LIBRARY CODE

Below is our example Class Library source code that performs 3 different protocol decodes and displays the results on the waveline. A version that includes an actual NEC IR decoder is installed with the USBee Suite. Use this example to start your own.

```
Imports System.IO


Public Class CustomUSBeeSuiteDecoder

    Declare Function SampleData Lib "usbeeste.dll" Alias _
                "?LoggedData@@YGJK@Z"  (ByVal Index As Integer) As UInteger
    ' The SampleData routine returns a 4 byte value that contains a single
    ' sample of all the signals
    ' The format of the 32 bits is as follows:
    '
    ' MSB                           LSB
    ' XXXXXXXXYYYYYYYYFEDCBA9876543210
    '
    ' where XXXXXXXX is Channel 2 Analog value (0=-10V, 255 = +10V)
    '       YYYYYYYY is Channel 1 Analog value (0=-10V, 255 = +10V)
    '       F is logic level (0 or 1) for channel F
    '       E is logic level (0 or 1) for channel E
    '       D is logic level (0 or 1) for channel D
    '       ...
    '       0 is logic level (0 or 1) for channel 0

    Public Sub DecodeCustom(ByVal OutFile As String, _
                            ByVal NumberOfSamples  As Integer, _
                            ByVal RateIndex As Byte, _
                            ByVal Parameters As String)

        Dim OldSample As UInteger

        Try

            ' This is a custom bus decoder Processing Routine
            '
            ' The passed in variables are as follows:
            ' OutFile         - the file that all of the decoded Entries get
            '                   written to.  This is the file that the Suite
            '                   will read to display the data on the waveline.
            ' RateIndex       - Index of the sample rate samples were taken.
            '                   17=1Msps,27=2Msps,37=3Msps,47=4Msps
            '                   67=6Msps,87=8Msps,127=12Msps,167=16Msps
            '                   247=24Msps
            ' Parameters      - User defined string passed from the Suite user
            '                   interface Channel Setting for this waveline.
            '                   Use this string to pass in any parameters that
            '                   your decoder needs to know, such as what
            '                   channels to use in decoding, which protocol if
            '                   you have multiple protocols supported here,
            '                   and how you want the data formatted.

            ' Below is an example set of Custom Protocol decoders that show
            ' how to access the sample buffer and how to generate output that
            ' get sent to the screen.

            ' Setup the File Stream that stores the Output Entry Information
            Dim FS As New FileStream(OutFilename, FileMode.Append, _
                                     FileAccess.Write)
            Dim BW As New BinaryWriter(FS)
```

```
        ' Since this file supports 3 different custom decoders, we need to
        ' see which one to run for this pass based on the Parameters
        ' string

        If InStr(Parameters.ToUpper, "CHANGE") Then
            ' Sample Decoder that just detects when a signal changes state
            ' The signal to use for the detection is specified in the
            ' Parameters as the second parameter

            Dim Params() = Parameters.Split(" ,-")
            Dim SignalToUse = Val(Params(1))
            ' Make the mask to mask off the channel we want in the sample
            Dim SignalMask = 1 << SignalToUse

            ' Now go from the start to the end and process the samples
            For Sample = 0 To NumberOfSamples - 1
                Dim DigitalChannel = SampleData(Sample) And SignalMask
                If DigitalChannel <> OldSample Then
                    WriteEntry(BW, Sample, Sample + 100, "Changed!")
                End If
                OldSample = DigitalChannel
            Next

        ElseIf InStr(Parameters.ToUpper, "RISE") Then

            ' Sample Decoder that detects when a signal has a Rising Edge
            ' The signal to use for the detection is specified in the
            ' Parameters as the second parameter

            Dim Params() = Parameters.Split(" ,-")
            Dim SignalToUse = Val(Params(1))
            ' Make the mask to mask off the channel we want in the sample
            Dim SignalMask = 1 << SignalToUse

            ' Now go from the start to the end and process the samples
            For Sample = 0 To NumberOfSamples - 1
                Dim DigitalChannel = SampleData(Sample) And SignalMask
                If (DigitalChannel <> OldSample) And (OldSample = 0) Then
                    WriteEntry(BW, Sample, Sample + 100, _
                                                        "Rising Edge!")
                End If
                OldSample = DigitalChannel
            Next

        ElseIf InStr(Parameters.ToUpper, "HELLOWORLD") Then

            ' Simplest Decoder Possible
            ' Print Hello World at the start of the buffer

            WriteEntry(BW, 0, 100, "Hello World!")

        End If

    Catch ex As Exception

    End Try

End Sub
```

```
    Public Sub WriteEntry(ByRef BW As BinaryWriter, _
                          ByVal StartSample As UInt32, _
                          ByVal EndSample As UInt32, _
                          ByRef TextString As String)

        ' DO NOT CHANGE THIS ROUTINE!!!
        ' This routine writes the Entry in the file format that is used by the
        ' Custom Decoder
        ' This entry specifies the Start Sample, End Sample and the text
        ' string to display

        Try

            BW.Write(StartSample)
            BW.Write(EndSample)

            ' Write the length of the string in bytes (include the 0 at the
            ' end in the count)
            Dim tStrLen As UInt32
            tStrLen = TextString.Length + 1
            BW.Write(tStrLen)

            ' Now write out the characters one byte at a time and put a 0 at
            ' the end
            For x = 0 To tStrLen - 2
                BW.Write(CByte(Asc(TextString.Chars(x))))
            Next
            BW.Write(CByte(0))

        Catch ex As Exception

        End Try


    End Sub

End Class
```

CUSTOM DECODER PARAMETERS

As you can see in the above code, the Parameter string that is passed from the User Interface to your decoder can be used for a number of purposes. First, it can specify which decoder to run. If you have more than one protocol that you want your decoder to handle, you can select which decoder runs using this text string. For example, in our example Custom Decoder we have three decoders possible, “CHANGE”, RISE”, and “HELLOWORLD”. Each decoder processes the data differently and outputs different results based on the algorithms.

You can also supply additional parameters in the text string as well as we do in the CHANGE and RISE decoders. The additional parameter in these examples indicates which signal to use to decode.

Again, the definition and use of the Parameter string is entirely up to you but provides an easy to use and simple to implement way to control the behavior of your decoder.

ACCESSING SAMPLE DATA TO PERFORM DECODE

To access each individual sample stored in the sample buffer you use the SampleData call as shown above. This returns a 32-bit value that includes all of the channels levels at that sample time. The format of the 32 bits is as follows:

```
MSB                              LSB
XXXXXXXXYYYYYYYYFEDCBA9876543210

where XXXXXXXX is Channel 2 Analog value (0=-10V, 255 = +10V)
      YYYYYYYY is Channel 1 Analog value (0=-10V, 255 = +10V)
      F is logic level (0 or 1) for channel F
      E is logic level (0 or 1) for channel E
      ...
      0 is logic level (0 or 1) for channel 0
```

Decoding any given protocol then entails going through the samples from beginning to end and masking off the channels you need to decode, accumulating decoded bits/bytes along the way and determining what the result is that you want to display.

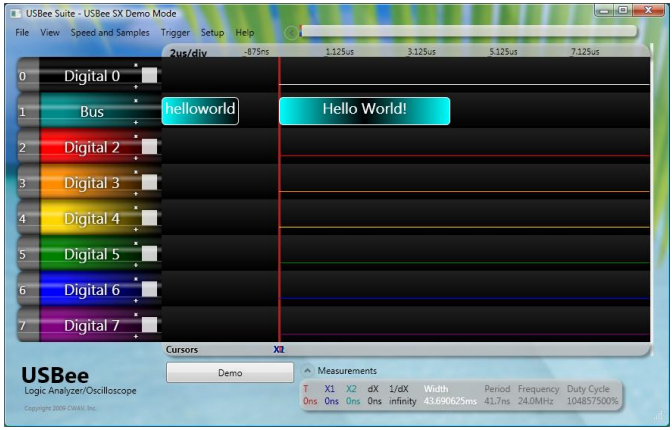
OUTPUTTING ENTRIES THAT WILL GET DISPLAYED ON THE SCREEN

Once your decoder has determined the result of the decode that you want displayed on the screen, you call the WriteEntry routine above. This gets passed the Start Sample, End Sample and Text String. Entries appear as rounded rectangles on the screen on the associated wave line and are locked to the samples that you specify.

For example, the following call places the “Hello World!” string on the screen stretching from the first sample to the 100th sample.

```
WriteEntry(OutFile, 0, 100, "Hello World!")
```

The output is as follows:



CHANGING THE BACKGROUND COLOR OF OUTPUTTED ENTRIES

You can control the background color of the entire WriteEntry item by embedding a color code anywhere into the text string. Color codes are in the following format:

[RGB]

Where:

R is the Red value and is a single digit of 0 thru 9.

G is the Green value and is a single digit of 0 thru 9.

B is the Blue value and is a single digit of 0 thru 9.

For example:

To output an Entry that has a bright green background, use the following:

```
WriteEntry(OutFile, 0, 100, "Hello World![090]")
```

To output an Entry that has a dark red background, use the following:

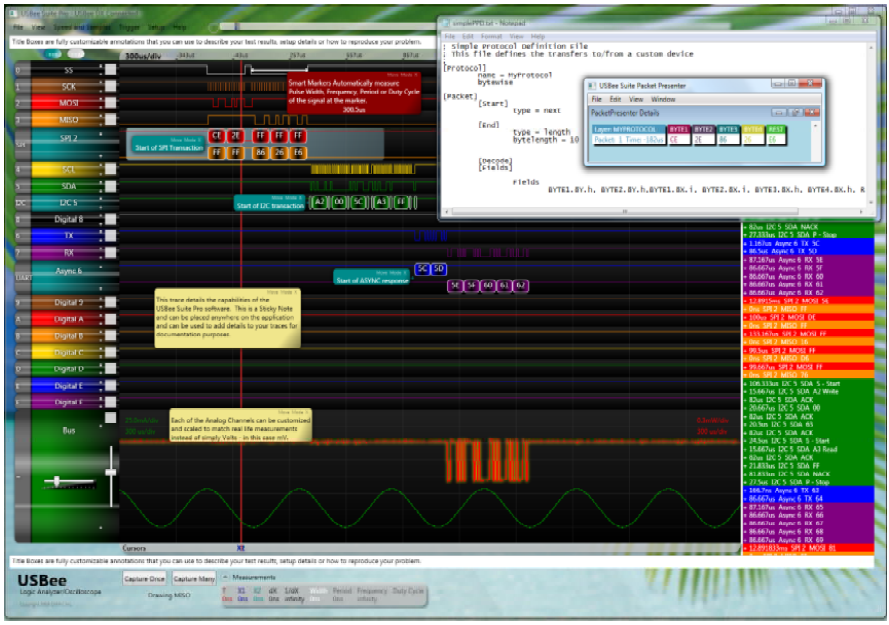
```
WriteEntry(OutFile, 0, 100, "Hello World![400]")
```

Some common color codes are as follows:

If you do not embed a color code, the background color will be a light cyan.

USING THE USBEE SUITE PRO

The USBee Suite Pro contains all of the features of the USBee Suite Standard and adds many powerful features to assist your debugging to get you to the root of your problems quickly. This section details the operation of these features.



To enable the USBee Suite Pro Features you first need to register your product. To register your purchase please follow the steps in the "Enabling the USBee Suite Pro Features" section at the beginning of this manual. The USBee Suite Pro software license is an additional purchase and can be done at any time.

The USBee Suite Pro adds the following features:

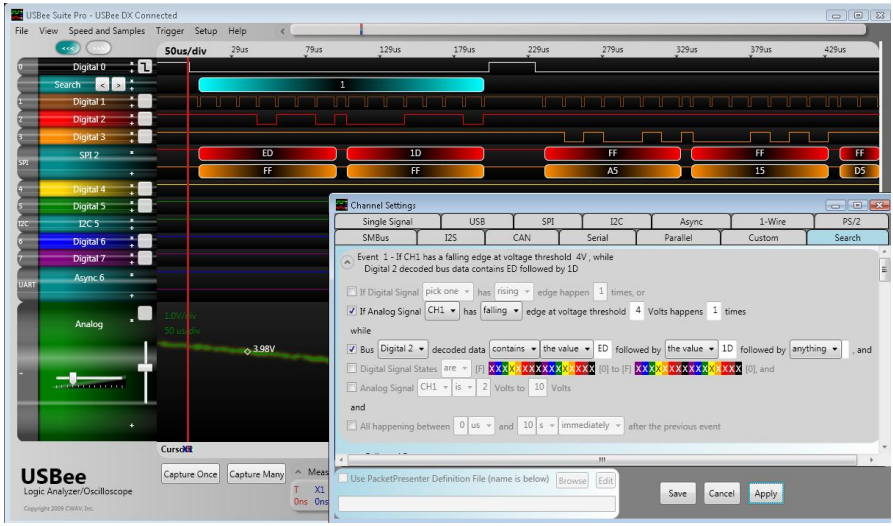
- Smart Search – automatically finds the data you are looking for
- Fast Pan Bus Viewing – quickly pans through your decoded bus data
- Sample and Smart Markers – place markers to highlight important areas
- Annotations and Sticky Notes – add annotations to document your findings
- Advanced Acquisition Control – Auto or Normal Triggering, Capture Once or Capture Many
- Display Modes – configure the display as you like
- Analog Channel Scaling – scale the analog channels to your custom scale, offset and units
- Browser-like Navigation – jump forward and backward to previous locations in your trace
- Analog Triggering – trigger off of the analog channels
- ULD Data File Importing – import USBee DX native capture files
- Relative Time Decode – view timestamps for decoded data in relative format
- PacketPresenter – Display bus data as high level packets

SMART SEARCH

The USBee Suite Pro Smart Search highlights the sections of your trace matching your areas of interest so that you don't need to waste time hunting for the data you need.

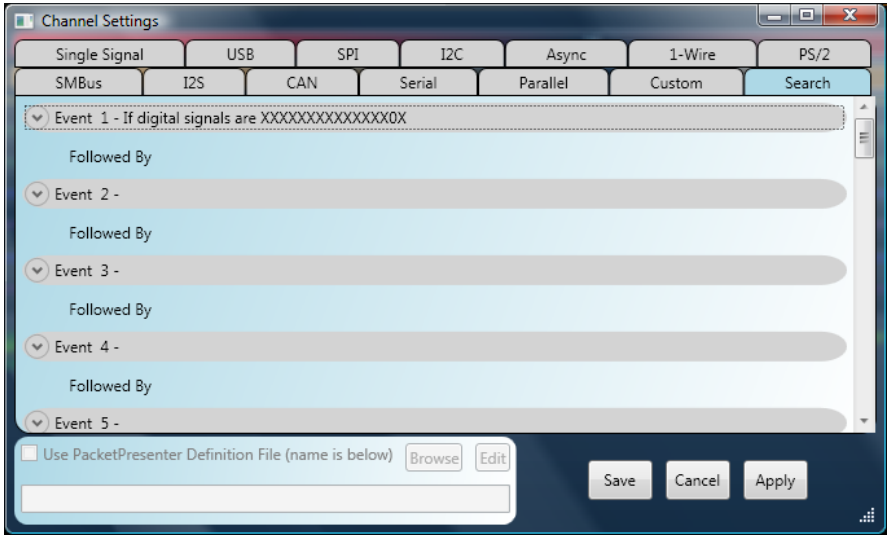
You can specify up to 32 levels of search events that are any combination of bus decoded traffic, states or edges of digital or analog signals, inside or outside of analog voltage ranges and/or digital ranges, and all validated by time specific windows.

Once specified you can pan through the occurrences of your searched items with the click of the mouse and see the total number of times the searched events occur.

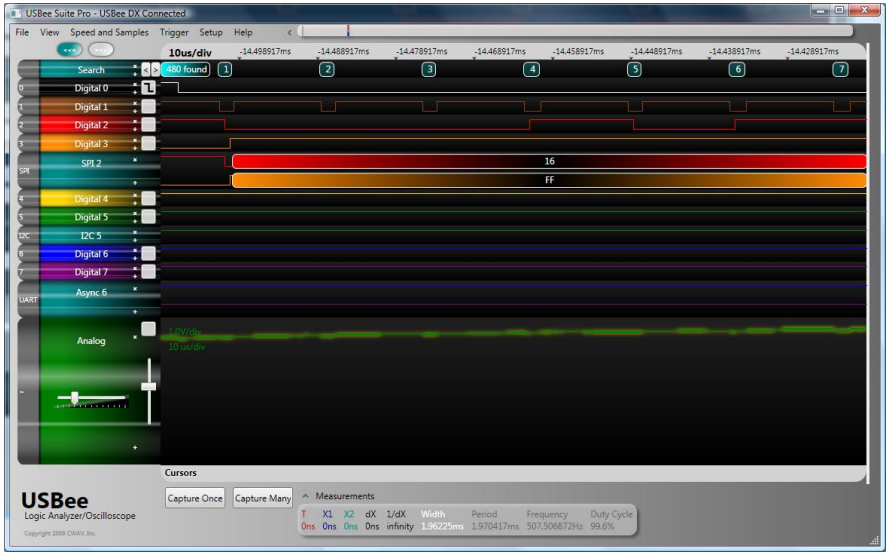


ADDING A SEARCH LINE

To add a Smart Search line on the display, click on the grey tab to the left of the line you want to change (or click the "+" sign to add a new wave line). Once you click on the **Search Tab** you will see the Channel Settings dialog box as below.



When you specify a Smart Search and click on **Save,** the new Search line will be added to the display. If your captured data matches these events anywhere in the trace it will display a bar at that location and mark it with a unique number. The trace below has found 480 matches and the display is showing match 1 through 7 on the first line.



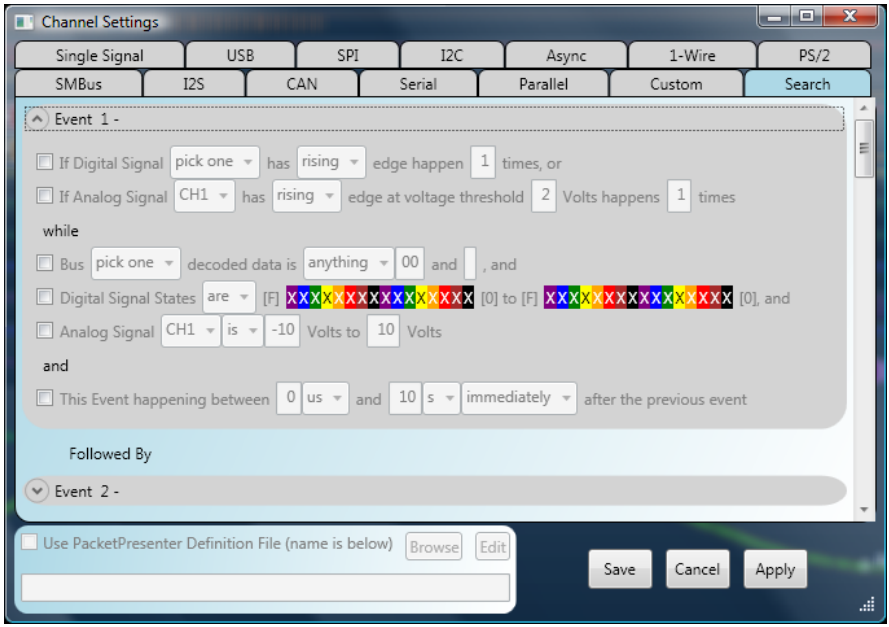
VIEWING SEARCH MATCHES

Once a search has been entered and processed, you can pan through the found matches by using the **left and right arrow buttons** on the search line. You can also see the total number of matches by hovering over the line.

ENTERING A SMART SEARCH

The Smart Search uses the specification that you choose in the channel settings Search window to determine what to highlight on the waveform display.

The search can be made up of up to 32 consecutive **Events**. Each Event is a selection of various bus, analog, and/or digital states. To edit an Event, click on the expander to show the Event details as below.



To enable a line in the Event search, click on the checkbox to the left of the items you want to include in your search.

You can specify any combination of the available lines in a search Event. All search criteria in a single Event must occur simultaneously for an event to be considered a match. For example, if you are looking for an edge on a digital signal, and looking for an analog voltage range, the edge must occur **while** the voltage is inside the range to be considered a match.

The following sections detail the available search items for each event and how they function.

DIGITAL SIGNAL EDGES

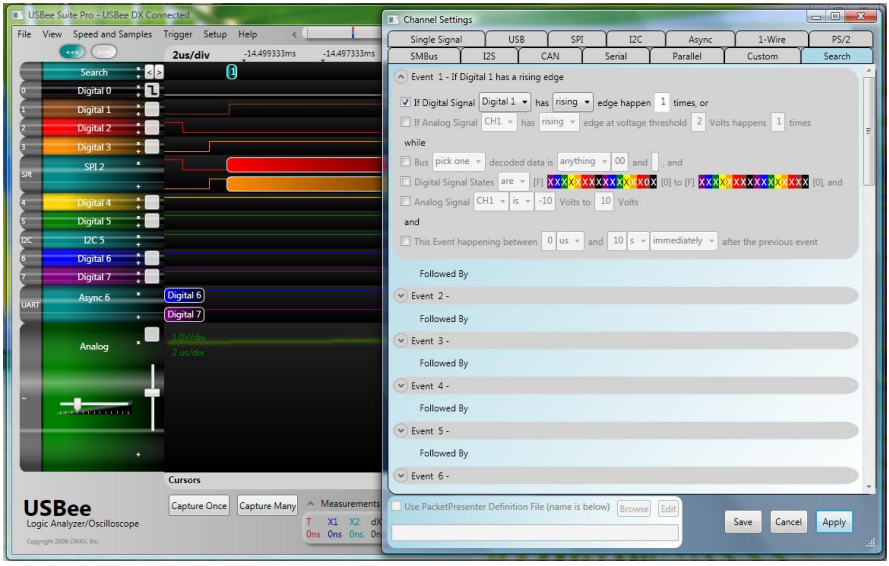
The first line lets you find edges on the digital signals.

First select the Digital line you want to search using the first dropdown box. This box is filled with only the single signals that are displayed on the screen.

Then choose the edge you want to search for: rising, falling or either rising or falling.

Finally, choose how many edges you need to find consecutively. If you want to find areas that have NO edges you can specify 0.

Below shows a search that finds all rising edges of the signal Digital 1.



ANALOG SIGNAL EDGES

The second line lets you find edges on the analog signals, if your USBee has them.

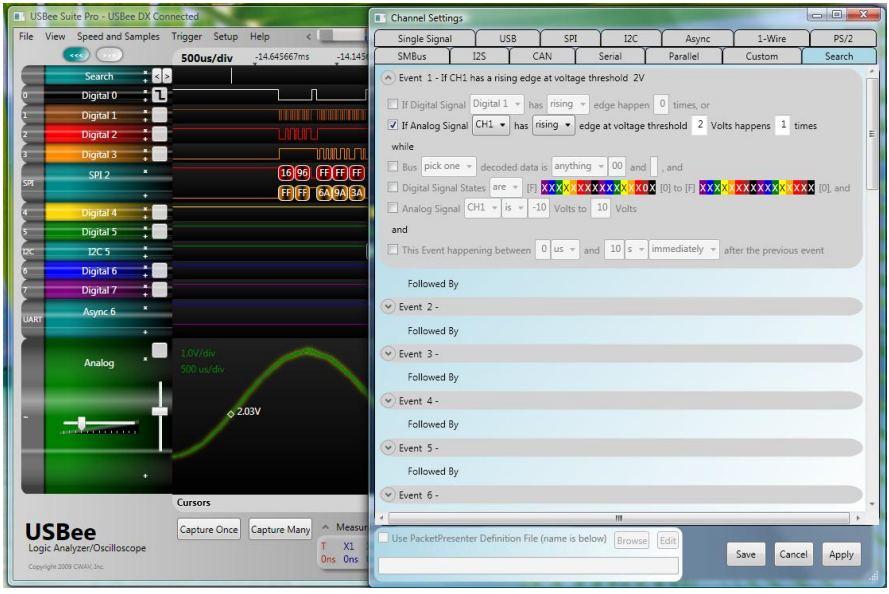
First select the Analog channel you want to search using the first dropdown box.

Then choose the edge you want to search for: rising, falling or either rising or falling.

Next enter the voltage threshold at which you consider the edge occurring. This value can be anywhere between -10 and 10 and can include decimal places.

Finally, choose how many edges you need to find consecutively. If you want to find areas that have NO edges you can specify 0.

Below shows a search that finds all rising edges of the analog signal CH1 with a threshold set at 2.0 Volts. Since the waveform display below is zoomed out the match is shown as simply a line. Zooming in on the search will show the details.



BUS DATA

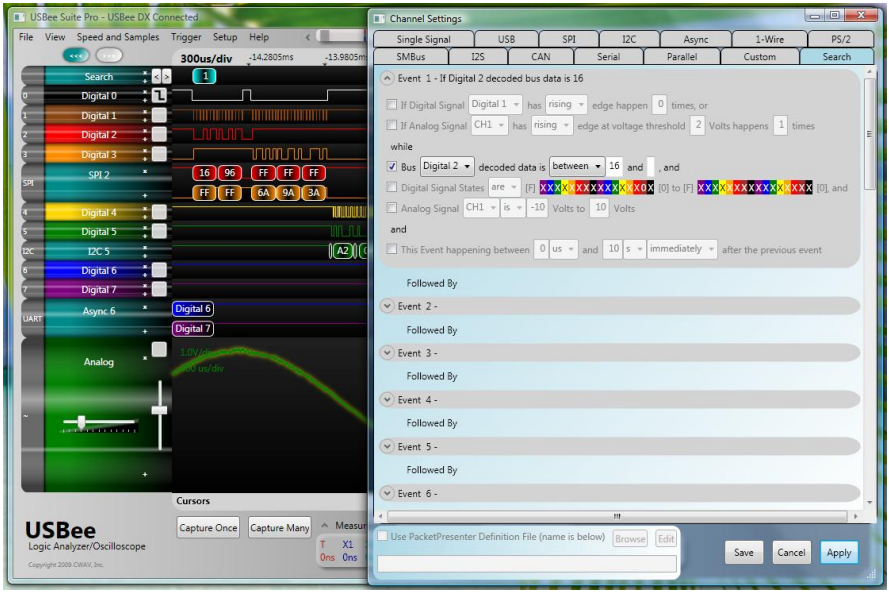
The next line lets you find values of decoded bus data if your waveform display includes them. You can specify specific values to find, values to exclude, or ranges of values.

First select the decoded bus channel you want to search using the first dropdown box.

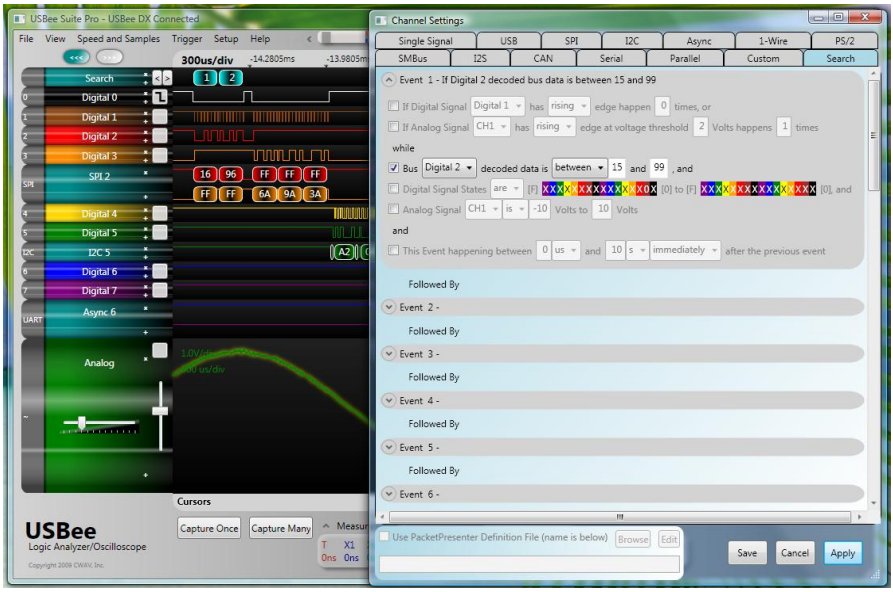
Then choose if you want to find values between (equals) or not between (not equals) the value(s) that follow.

Next enter the actual value of decoded data you are interested in. If it is a single value, leave the second field blank. If you are interested in a range of values, specify the end of the range in the second field.

Below shows a search that finds all decoded data that equals a fixed value of 16.



Below shows a search that finds all decoded data that is between 15 and 99.



DIGITAL SIGNAL STATES AND RANGES

The next line lets you find states, or ranges of states, on the digital signals.

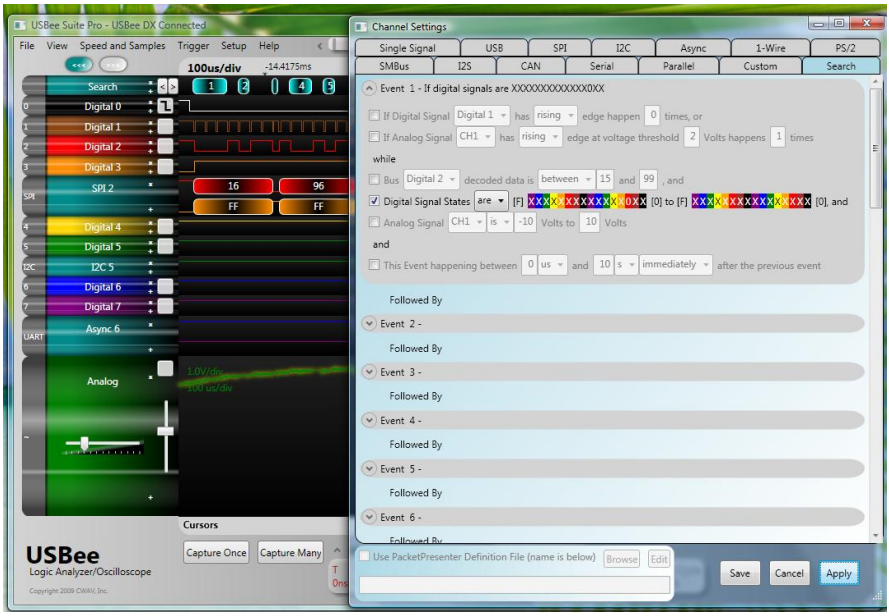
First select if you are looking for the digital states to include the values (**are**), or not include the values (**are not**), that follow.

Then specify the states of all of the digital lines by clicking on each individual signal to change from 0, 1, and X (don't care).

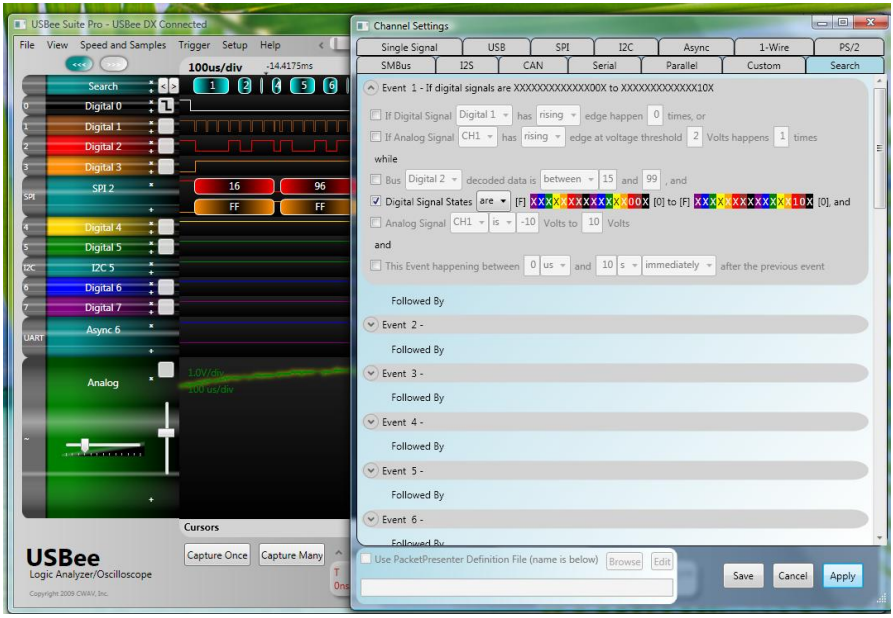
If you are looking for a single set of digital states, leave the second set of signals at all X.

If you want to find a range of states, enter the ending value as the second set of signals. The matched range will then include all values from the first set to the second set, inclusive.

Below shows a search that finds all occurrences of when the Digital 2 signal is logic 0.



Below shows a search that finds all occurrences of when the Digital 2 and Digital 3 signals are 0-0, 0-1, and 1-0. The range starts at the first setting and increments until the second setting.



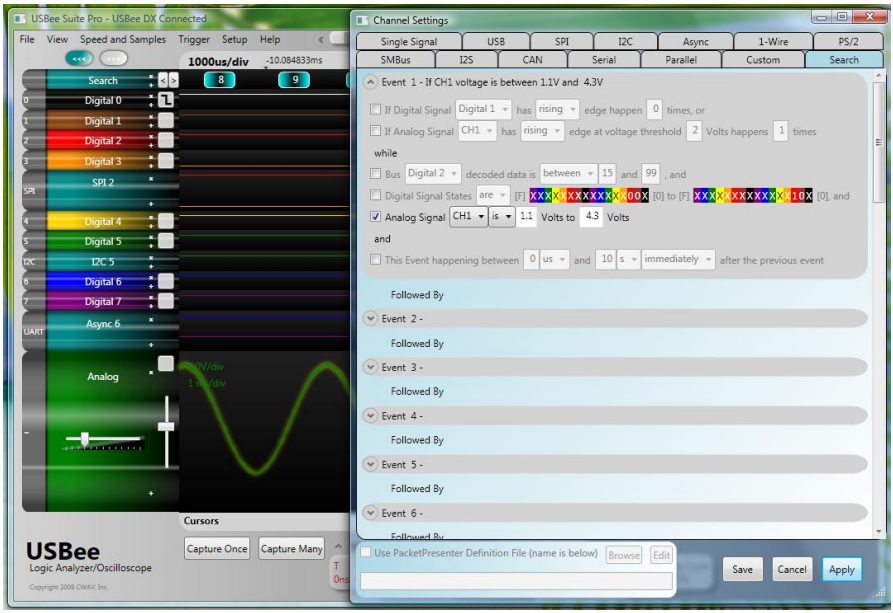
ANALOG SIGNAL STATES AND RANGES

The next line lets you find voltage ranges on the analog signals, if your USBee has them.

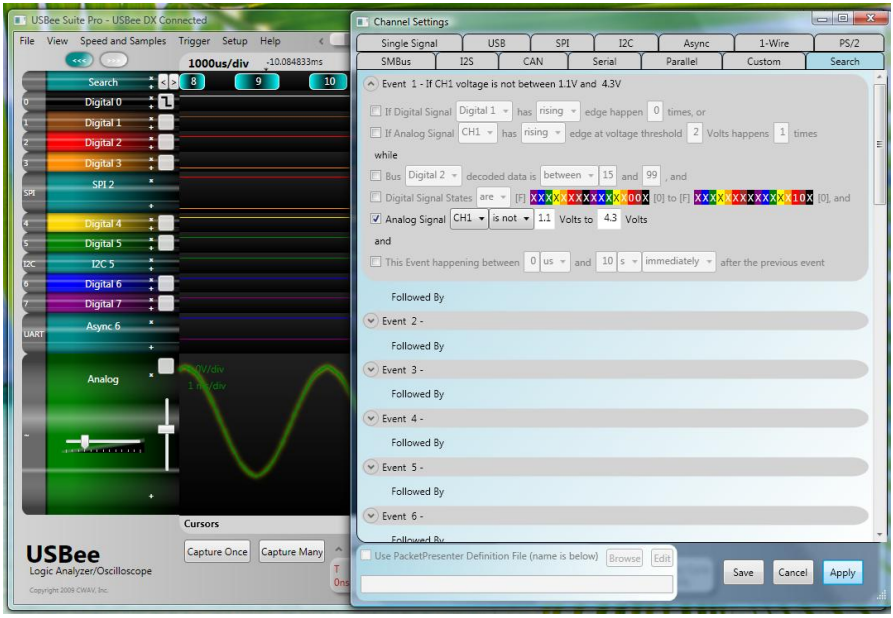
First select if you are looking for the analog voltages to include the values (**is**), or not include the values (**is not**), that follow.

Then specify the voltage range start and end in volts. The matched range will then include all values from the first set to the second set, inclusive.

Below shows a search that finds all times that the analog signal CH1 is between 1.1V and 4.3V.



Below shows a search that finds all times that the analog signal CH1 is NOT between 1.1V and 4.3V.



TIME WINDOW QUALIFIER

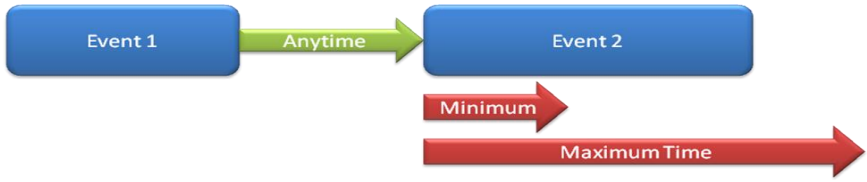
The final line of each Event specifies a time window for the event to occur.

There are two selections to specify a time window: **immediately** or **anytime**.

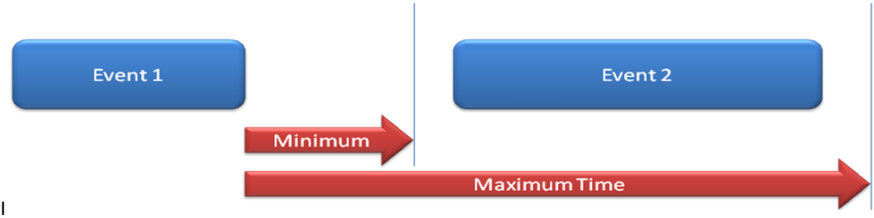
For the first Event, only **anytime** is available. This specifies a time period in which the entire event must occur for a match to be found. For example, if you want to find when two falling edges occur within 10usecs of each other, specify: *This Event happening between 0s and 10us anytime*.

For the Events 2-32, both **anytime** and **immediately** are available.

Anytime specifies a time period minimum and maximum in which the entire event must occur for a match to be found. The match can occur anytime after the previous event, but must occur within the time window. For example, if you want to find when two falling edges occur within 10usecs of each other, specify: *This Event happening between 0s and 10us anytime after the previous event*. Below shows a diagram for the **anytime** setting. As long as Event 2 is at least Minimum and at most Maximum time it is considered a match.

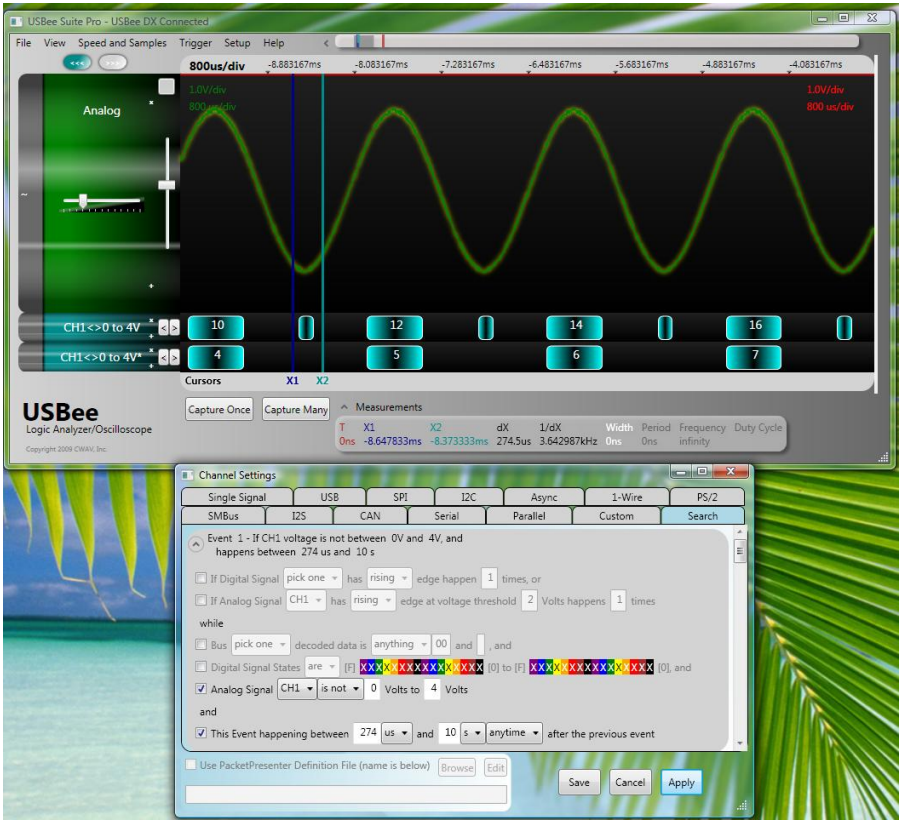


Immediately specifies a time period relative to the end of the previous event in which the entire event must occur for a match to be found. For example, if you want to find when two falling edges occur within 10usecs of each other immediately following the previous event, specify: *This Event happening between 0s and 10us immediately after the previous event*. Below shows a diagram for the **immediately** setting. As long as Event 2 occurs between the Minimum time and the Maximum time it is considered a match.



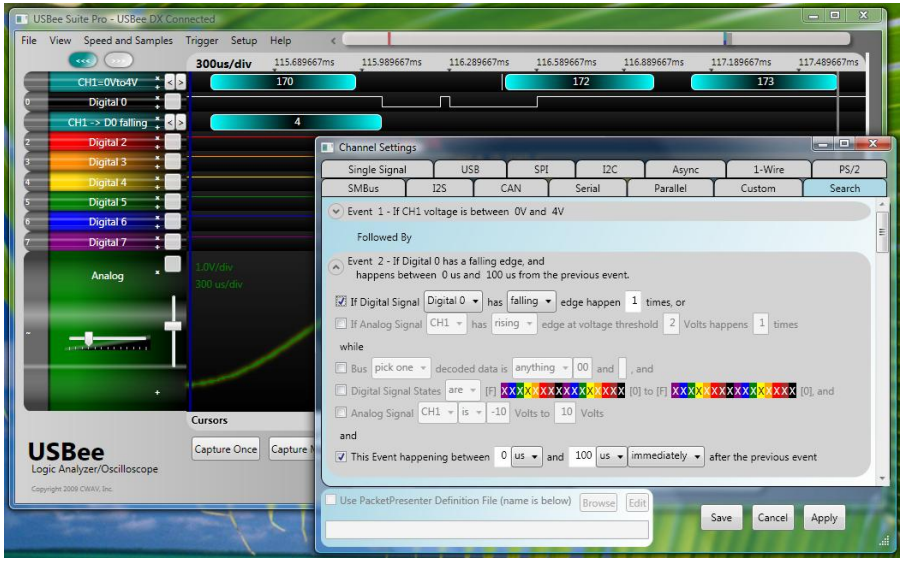
I

Below shows an example of applying a time window. The first search line shows all occurrences of when the analog CH1 is outside the range 0V to 4V. The second search line is the same criteria with the addition of a time window applied. Only matches that are between 274us and 10seconds are displayed. If you are looking for occurrences that are less than a specific time set the first entry to 0s and the second to the upper limit.



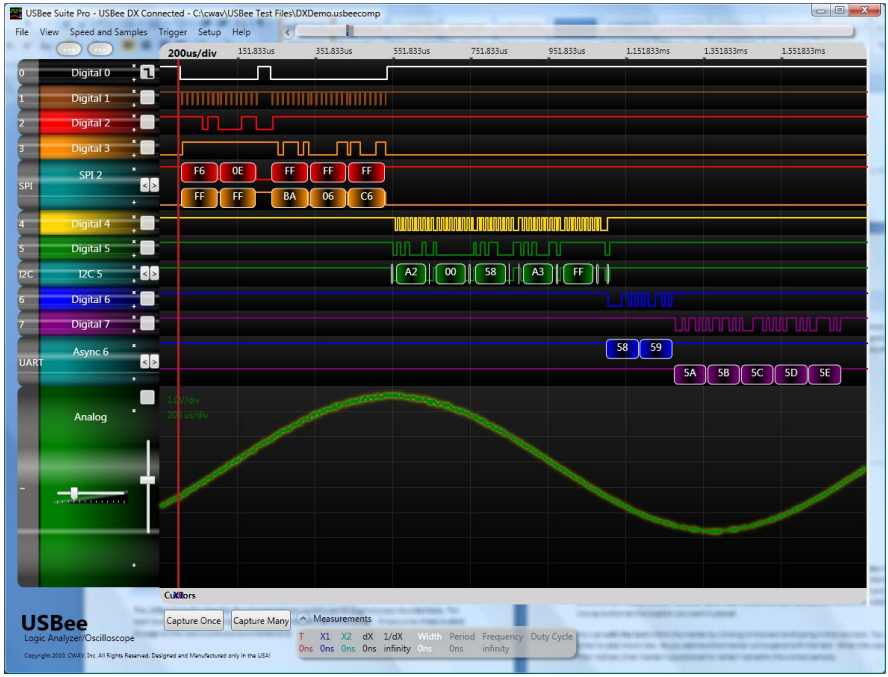
For Events other than the first Event (2-32) you can also specify immediately to indicate that the event must happen between the first time and second time starting at the end of the previous event.

The following example shows two search lines. The first search line shows all occurrences of the CH1 being between 0V and 4V. The second search line shows all occurrences of when the Digital 0 line has a falling edge within 100usecs of the end of the first search.



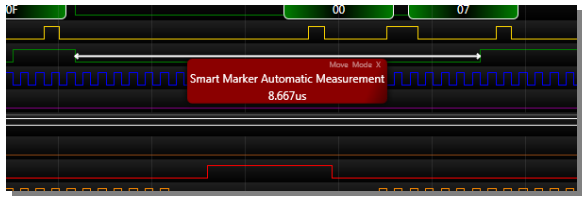
FAST PAN BUS VIEWING

The USBee Suite Pro Fast Pan Bus Viewing lets you quickly pan through a busses decoded data. For each bus there is a left and right pan button on the left side of the screen. Simply press these buttons to page to the next or previous bus transactions.

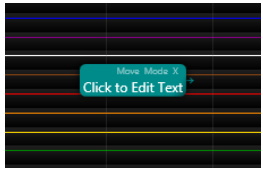


SAMPLE AND SMART MARKERS

Placing markers in your traces can help detail what is happening in your design. There are two types of markers that can be used. The first marker type locks itself to a sample on a waveform and lets you specify the text. The second is a Smart Marker that automatically measures the pulse width, frequency, period or duty cycle of the waveform at the marker location.



SAMPLE MARKERS



Markers that are locked to a specific sample are placed using the **View | Add Sample Marker** menu item. Once you select this menu item your cursor changes to 4-way arrows and a Sample Marker moves wherever you move the cursor. To **place the marker**, position it where you want it and then press the left mouse button. Another faster way to place Sample Markers is to press the middle mouse button at the location you want it placed.

You can **edit the text** within the marker by clicking on the text and typing in the new text. You can hit Enter to add more lines. As you add text the marker will expand to fit the text. When the waveforms then redraw, then marker is positioned to remain locked to the correct sample.

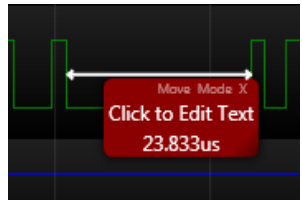
To **move the marker**, click on the Move at the top of the marker. Once you have it moved to the new location use the left mouse button to place it.

The Mode at the top of the marker **changes the direction** of the marker. Click on Mode to change from Left pointing marker to Right pointing marker and vice versa.

To hide all of the markers without deleting them, uncheck the menu item **View | Show Marker Labels**. To turn on the markers, make sure this menu item is checked.

To **delete the marker**, click on the X at the top of the marker. You can delete all markers using the menu item **View | Delete All Markers**. This will delete all Sample Markers and Smart Markers.

SMART MARKERS



Smart Markers are locked to a specific sample and measure the waveform underneath. They are placed using the **View | Add Smart Marker** menu item. Once you select this menu item your cursor changes to 4-way arrows and a Smart Marker moves wherever you move the cursor. To place the marker, position it where you want it and then press the left mouse button.

You can **edit the text** within the marker by clicking on the text and typing in the new text. You can hit Enter to add more lines. As you add text the marker will expand to fit the text. When the waveforms then redraw, then marker is positioned to remain locked to the correct sample.

To **move the marker**, click on the Move at the top of the marker. Once you have it moved to the new location use the left mouse button to place it.

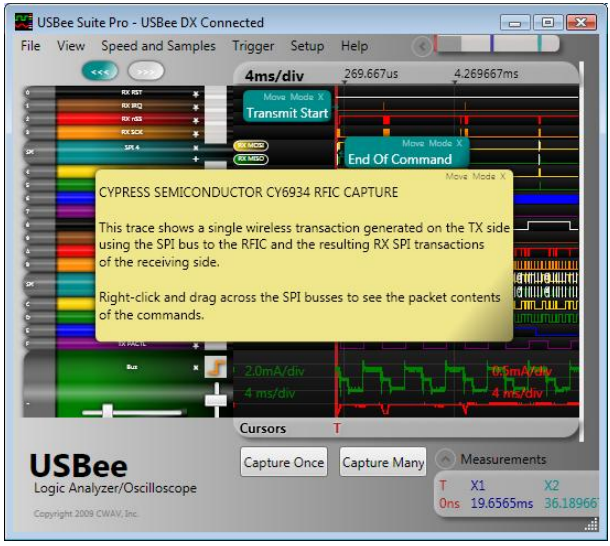
The Mode at the top of the marker **changes the measure mode** of the marker. Click on Mode to cycle through No Measurement, Width, Frequency, Period, and Duty Cycle. An arrow shows the measured area and the measurement shows up as the last line of the marker.

To hide all of the markers without deleting them, uncheck the menu item **View | Show Marker Labels**. To turn on the markers, make sure this menu item is checked.

To **delete the marker**, click on the X at the top of the marker. You can delete all markers using the menu item **View | Delete All Markers**. This will delete all Sample Markers and Smart Markers.

ANNOTATIONS AND STICKY NOTES

The USBee Suite Pro adds Sticky Notes which you can use to further detail your traces for documentation purposes. You can also add Title and Footer text to your display that is saved with the trace file.



ANNOTATIONS

Annotation Text Boxes are editable text blocks that are located at the Top and Bottom of the USBee Suite Pro window. Annotation Text Boxes are enabled and disabled using the **View | Annotation Text Boxes** menu item.

To edit the text, simply select the box and edit the text. This text is then saved with your capture files.

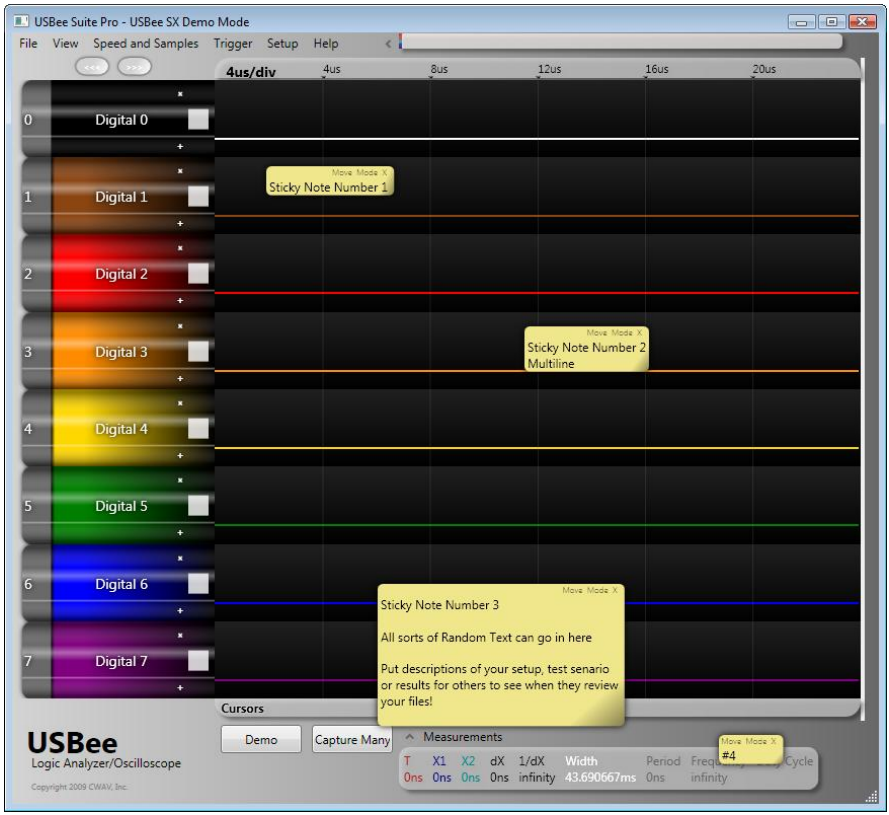
STICKY NOTES

Sticky Notes are editable text blocks that look like sticky notes. They can be positioned anywhere in the application window. Sticky Notes are placed using the **View | Add Sticky Note** menu item. Once you select this menu item your cursor changes to 4-way arrows and a Sticky Note moves wherever you move the cursor. To place the note, position it where you want it and then press the left mouse button.

You can **edit the text** within the Sticky Note by clicking on the text and typing in the new text. You can hit Enter to add more lines. As you add text the marker will expand to fit the text.

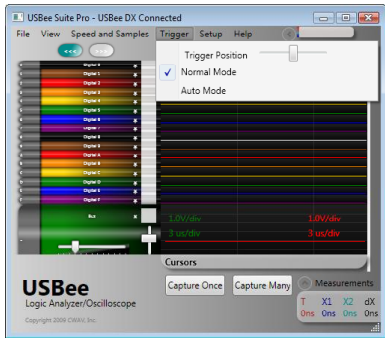
To **move the Sticky Note**, click on the Move at the top of the note. Once you have it moved to the new location use the left mouse button to place it.

To **delete the Sticky Note**, click on the X at the top of the note. You can delete all Sticky Notes using the menu item **View | Delete All Sticky Notes**.



ACQUISITION CONTROL

The USBee Suite Pro adds more trace acquisition and triggering controls such as Normal Mode, Automatic Mode, Single Capture and Multiple Capture.



When the USBee Suite Pro is first started, no acquisition is taking place. You need to press one of the acquisition buttons, **Capture Once** or **Capture Many**, at the bottom of the window to capture data.

The **Capture Many** button 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 same button again (now reading Stop) and the updating will stop. This mode is great for signals that repeat over time.

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

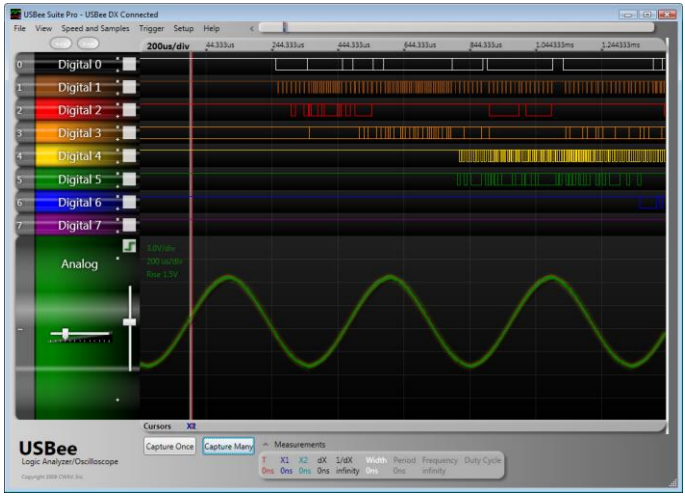
The USBee Suite Pro adds the ability to have either Normal Mode (default n the Standard version) or Automatic Mode triggering. This determines when the signals start getting sampled once you press the Capture buttons. You specify the trigger event by selecting one of the USBee signals to trigger on (rising or falling edge).

Normal mode will wait for the trigger event to occur before capturing. Select this option using the **Trigger | Normal Mode** menu item. If the trigger event does not occur you can press the Stop button to terminate the capture.

Automatic Mode will wait a set time for the trigger and will automatically trigger if it is not found. Select this option using the **Trigger | Automatic Mode** menu item. If the trigger event does not occur within a specified time, it will automatically start a capture of whatever is on the signals at the time. You can press the Stop button to terminate the capture.

DISPLAY MODES

The USBee Suite Pro lets you widen the trace waveforms, display the analog waveforms as vectors or single sample points, and persist the display from one trace to the next.

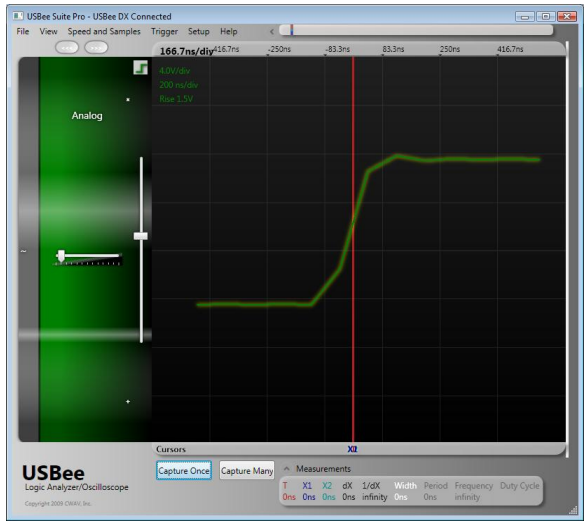


The **Wide** setting shows the waves using a wider pixel setting. This makes the waves easier to see. You can toggle this setting using the menu item **View | Wide Lines.**

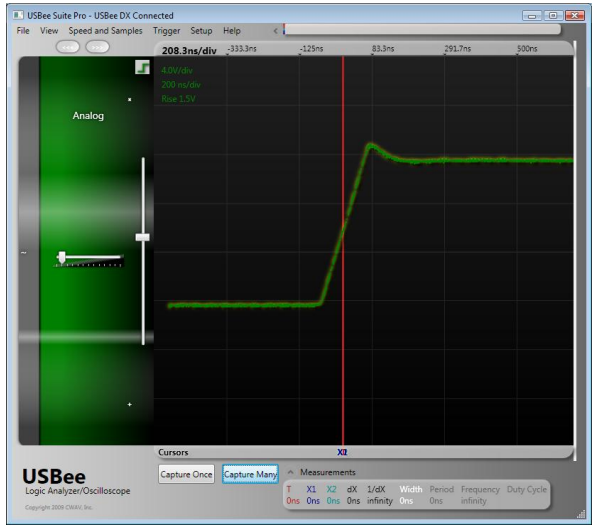
The **Vectors** setting draws the waveforms as a line between adjacent samples. With this mode turned off, the samples are shown simply as dots on the display at the sample position. You can toggle this setting using the menu item **View | Vector-based Waveforms.**

The **Persist** mode does not clear the display and writes one trace on top of the other trace. You can toggle this setting using the menu item **View | Waveform Persistence.**

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 second one shows much more wave detail over a short time of display updates.



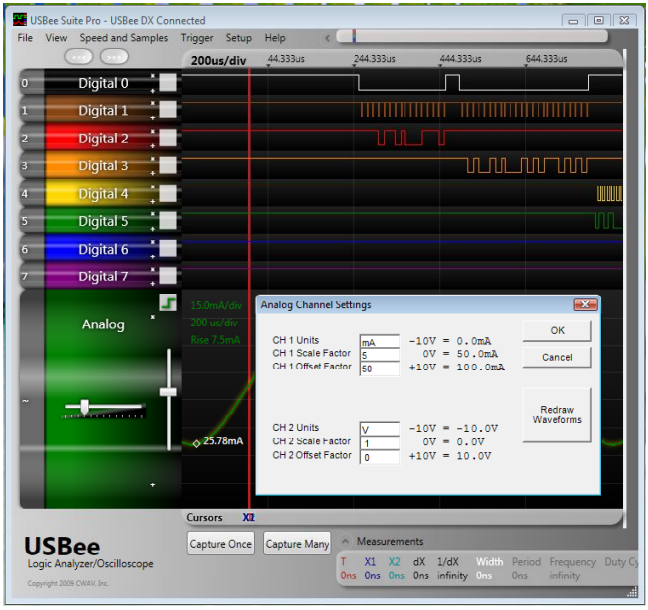
Persist = OFF, Vectors = ON, Wide = ON



Persist = ON, Vectors = OFF, Wide = ON

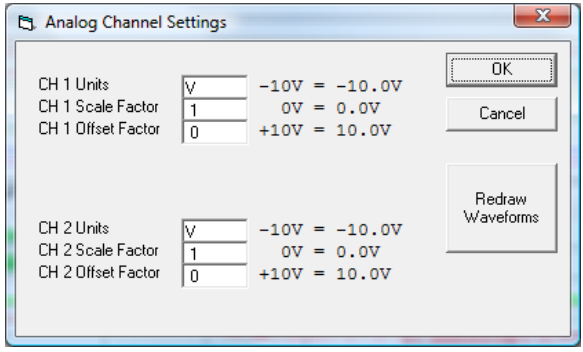
ANALOG CHANNELS SCALING

The USBee Suite Pro provides a scaling ability to convert the analog voltages into other units of measurement.

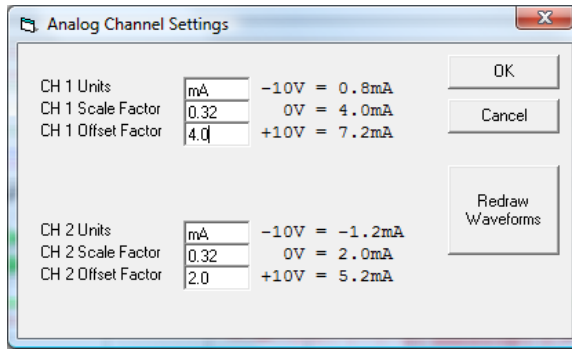


By default, each analog channel is set to display the measurements in Volts where 1V is shown as 1V on the display. Sometimes the measurement might actually mean a different thing than voltage. The menu item **Setup | Analog Channel Settings** lets you specify the units of measurement as well as a scale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

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

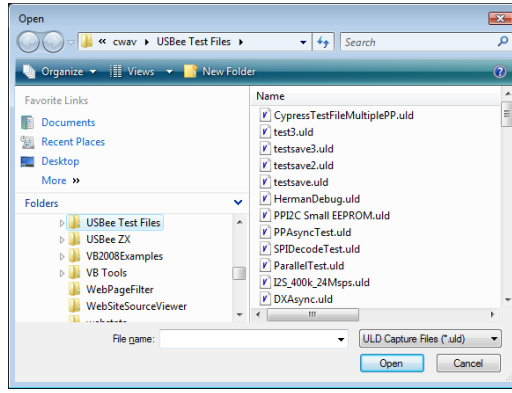


Below shows a setting of mA with various gains and offsets. Instead of displaying the actual value measured in volts, the display will show the scaled value in the new units.



ULD DATA FILE IMPORTING

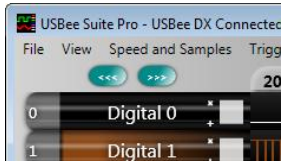
The USBee DX saves files in the ULD file format which can be imported into the USBee Suite Pro. To import a ULD file into the USBee Suite Pro, choose the menu item **File | Import File**. A file selection dialog box will appear as below.



Select the file you want to import and press Open. The ULD file will then be imported and displayed. You can then save the traces as the new USBee Suite file format if desired using the menu item File | Save As.

BROWSER-LIKE NAVIGATION

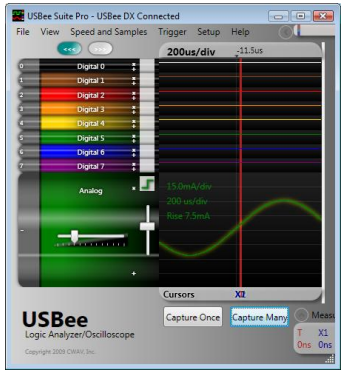
The USBee Suite Pro adds browser-like Forward and Back buttons that let you quickly navigate through your trace display.



Each time you stop at a certain point when viewing your waveforms, the location is saved to the history buffer. This allows you to quickly jump back to the previous locations within your trace without having to scroll, pan or zoom. Press the Back Button (Cyan oval with <<<) to go backwards in the history buffer. Press the Forward Button (Cyan oval with >>>) to go forward in the history buffer.

ANALOG TRIGGERING

The USBee Suite Pro adds the ability to trigger on a rising or falling edge of any analog channel.

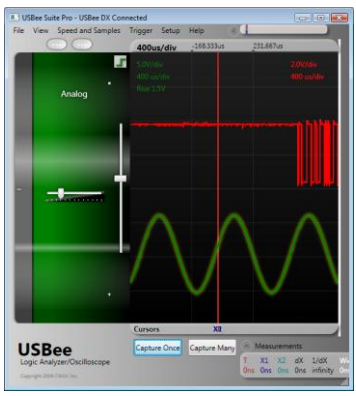


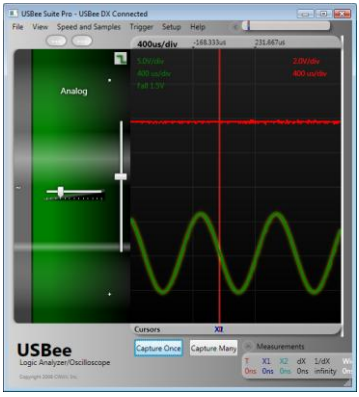
The USBee Suite Pro uses a trigger mechanism to allow you to capture just the data that you want to see. You can use either a digital channel trigger or an analog trigger. You cannot use a combination of analog and digital.

For an **Analog trigger** you must specify the Channel to use, Rising or Falling Edge, and the Trigger Level. Click on the Trigger Settings Box (to the right of the waveline delete "X") repeatedly to toggle through Channel 1 Rising, Channel 1 Falling, Channel 2 Rising, Channel 2 Falling and None. You then specify the trigger voltage level (-10V to +10V) by using the vertical slider on the left hand side of the analog waveform display. The trigger level edge and value will be shown as you scroll this level underneath the Volts/Div and Secs/Div labels within the waveform area.

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.

The following figures show a trace captured on each of the edges.



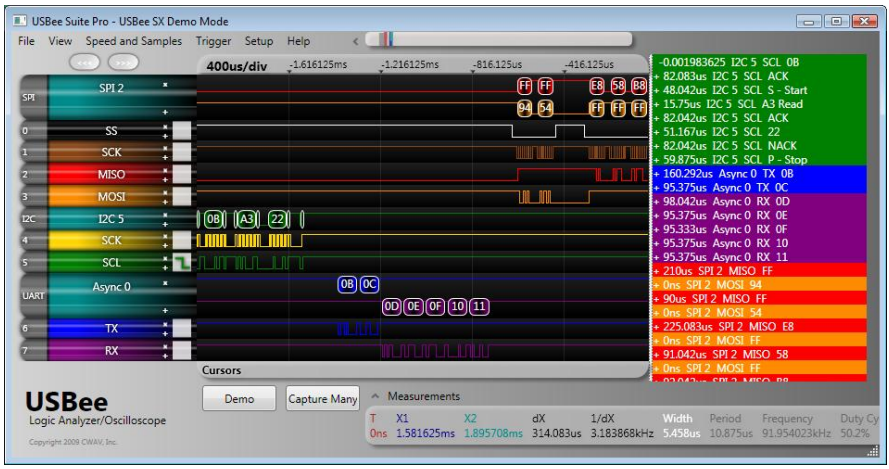


Analog Trigger Slope = Rising Edge Analog Trigger Slope = Falling Edge

The Trigger position is placed where the actual signal crosses the trigger voltage with the proper slope. The USBee pods allow 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.

RELATIVE TIME DECODE

The USBee Suite Pro also adds a Relative Time or Absolute Time setting for the decoded data lists.



Absolute Timestamps display the sample time for each decoded bus transaction relative to the trigger location. You can turn on Absolute Timestamps using the menu item **View | Decoder Timestamps Absolute**.

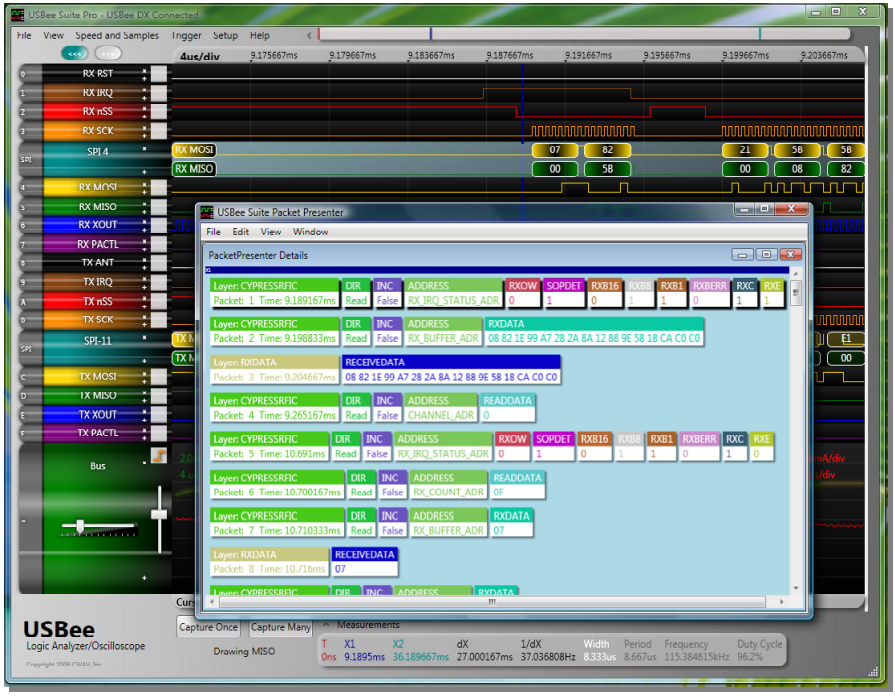
```
0.017596417,SPI 2,MISO,4A
0.017596417,SPI 2,MOSI,FF
0.017688458,SPI 2,MISO,AA
0.017688458,SPI 2,MOSI,FF
0.017802750,I2C 5,SCL,S - Start
0.017818458,I2C 5,SCL,A2 Write
0.017900542,I2C 5,SCL,ACK
0.017933167,I2C 5,SCL,00
0.018015250,I2C 5,SCL,ACK
0.018049208,I2C 5,SCL,4A
0.018131292,I2C 5,SCL,ACK
0.018179375,I2C 5,SCL,S - Start
0.018195083,I2C 5,SCL,A3 Read
0.018277167,I2C 5,SCL,ACK
0.018328333,I2C 5,SCL,61
0.018410417,I2C 5,SCL,NACK
0.018470208,I2C 5,SCL,P - Stop
0.018633208,Async 0,TX,4A
0.018728542,Async 0,TX,4B
0.018826583,Async 0,RX,4C
0.018921958,Async 0,RX,4D
0.019017333,Async 0,RX,4E
0.019112708,Async 0,RX,4F
0.019208042,Async 0,RX,50
0.019417042,SPI 2,MISO,FF
0.019417042,SPI 2,MOSI,16
0.019509083,SPI 2,MISO,FF
0.019509083,SPI 2,MOSI,96
0.019733167,SPI 2,MISO,6A
```

Relative Timestamps display the time difference since the last bus transaction and the current transaction. You can turn on Relative Timestamps using the menu item **View | Decoder Timestamps Relative**.

```
+82.042us,I2C 5,SCL,ACK
+51.167us,I2C 5,SCL,22
+82.042us,I2C 5,SCL,NACK
+59.875us,I2C 5,SCL,P - Stop
+160.292us,Async 0,TX,0B
+95.375us,Async 0,TX,0C
+98.042us,Async 0,RX,0D
+95.375us,Async 0,RX,0E
+95.333us,Async 0,RX,0F
+95.375us,Async 0,RX,10
+95.375us,Async 0,RX,11
+210us,SPI 2,MISO,FF
+0ns,SPI 2,MOSI,94
+90us,SPI 2,MISO,FF
+0ns,SPI 2,MOSI,54
+225.083us,SPI 2,MISO,E8
+0ns,SPI 2,MOSI,FF
+91.042us,SPI 2,MISO,58
+0ns,SPI 2,MOSI,FF
+92.042us,SPI 2,MISO,B8
+0ns,SPI 2,MOSI,FF
+114.292us,I2C 5,SCL,S - Start
+15.708us,I2C 5,SCL,A2 Write
+82.083us,I2C 5,SCL,ACK
+32.667us,I2C 5,SCL,00
+82.042us,I2C 5,SCL,ACK
+33.958us,I2C 5,SCL,12
+82.042us,I2C 5,SCL,ACK
+48.083us,I2C 5,SCL,S - Start
+15.708us,I2C 5,SCL,A3 Read
+82.083us,I2C 5,SCL,ACK
+51.167us,I2C 5,SCL,29
```

PACKETPRESENTER™

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



OVERVIEW

Using the USBee Suite Pro 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 in-line with the waveform.

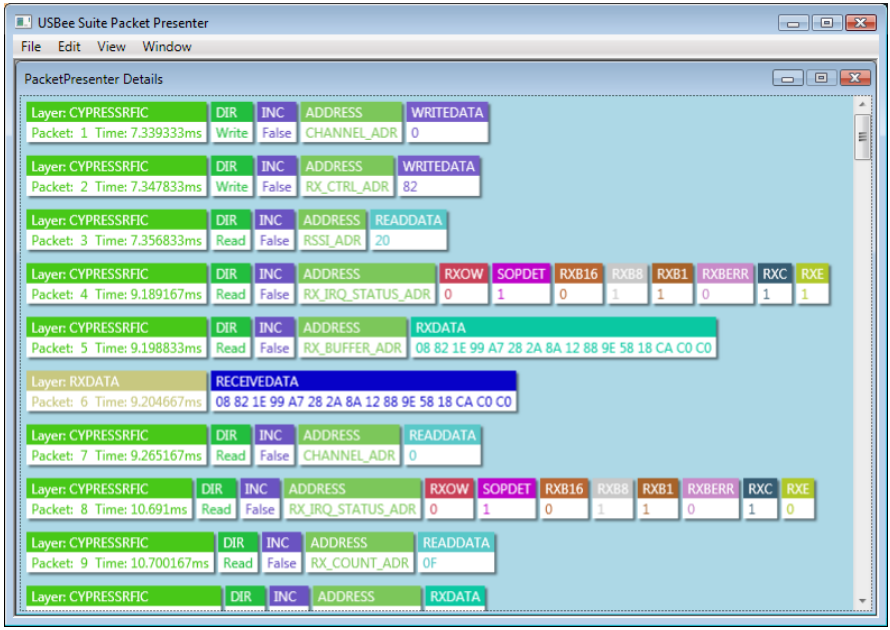
The PacketPresenter™ feature runs alongside of the existing bus decoders of the USBee Suite. The PacketPresenter™ takes the output of raw binary data from the bus decoder and parses the stream according to users PacketPresenter Definition File for the intent of displaying the communications in easily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

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).

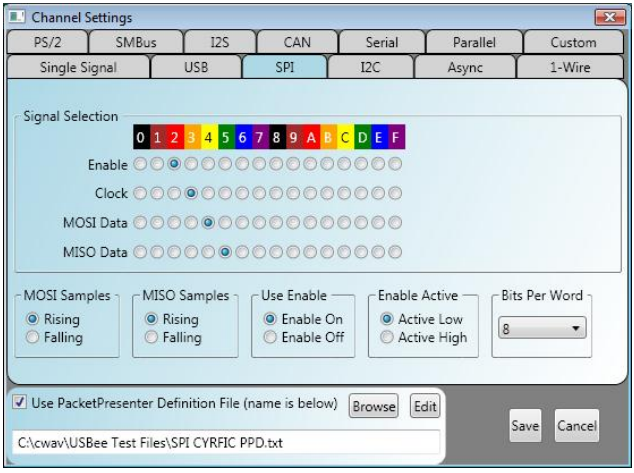
Each Protocol Processor defines how to break the stream into Packets, and how to break the Packets into Fields. These Fields can then be displayed or sent to another Data Stream for further processing.

Below shows a sample PacketPresenter output screen.



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.



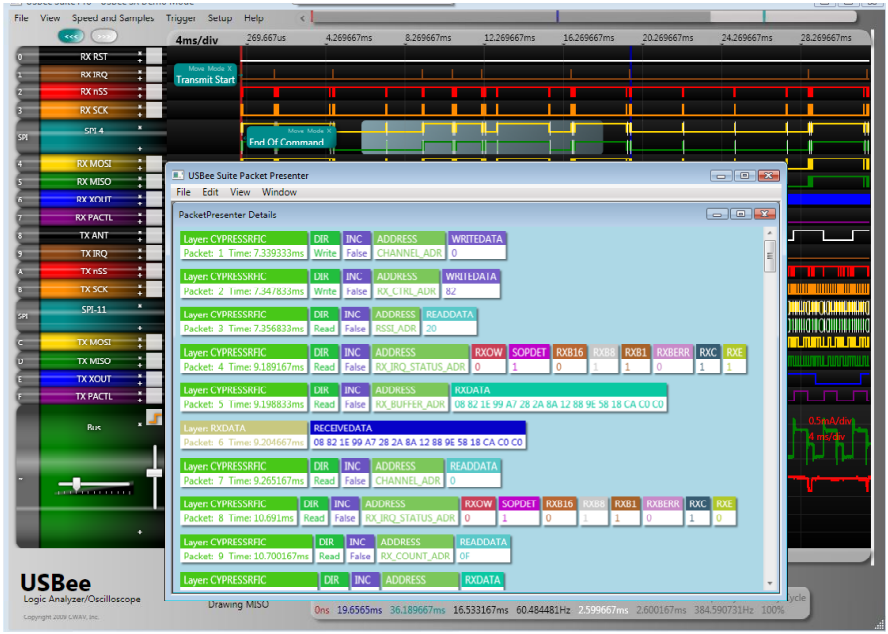
To enable the PacketPresenter for this channel, check the "**Use PacketPresenter definition file (name is below)**" checkbox. Then choose the PacketPresenter definition file by clicking the **Browse** button to the right. Once you choose the file, you can edit the contents by clicking the "Edit File" 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 parse the decoded bus data based on your PacketPresenter definition file and display the PacketPresenter results.

The area selected is highlighted with a colored bar background. When you release the right mouse button, the PacketPresenter results for that section are displayed. The background of the PacketPresenter window is colored the same as the highlighted section on the waveform.



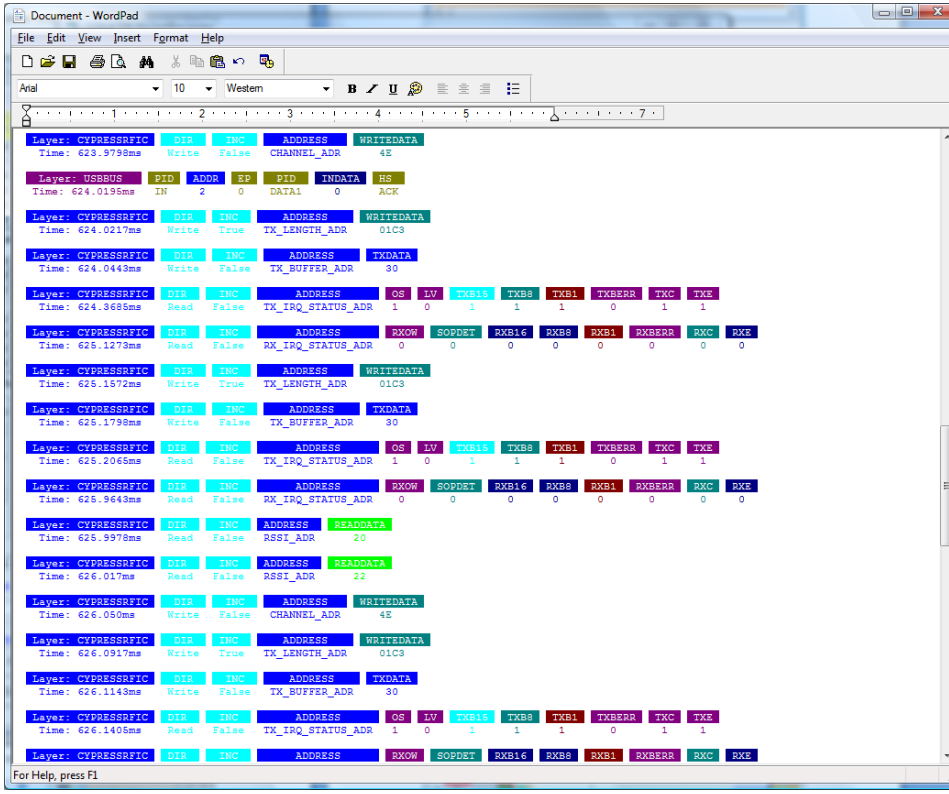
You can show the raw decoded data at the same time by restoring the minimized window as shown in the following screenshot.

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. Access these features through the **File |Save As Text** or **File | Save As RTF** menu items.

```
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.



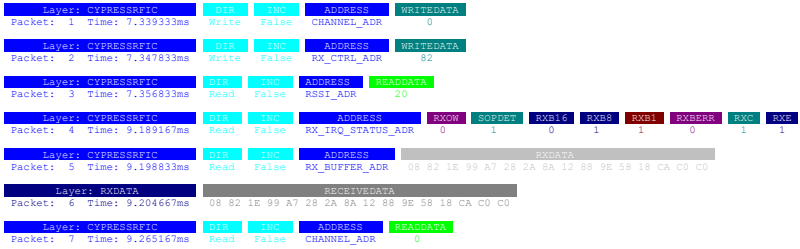
In order to maintain correct position of the graphical portions of the RTF file, all spaces are converted to the character "~" and set to the background color. Viewed or printed in the RTF format will look correct as above. If you copy only the text of this output, you will want to search and replace 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 as Text** or **Edit | Copy as RTF** menu items. All packets are copied to the clipboard in the format specified. Below is a sample of packets copied in RTF format and pasted into Word.

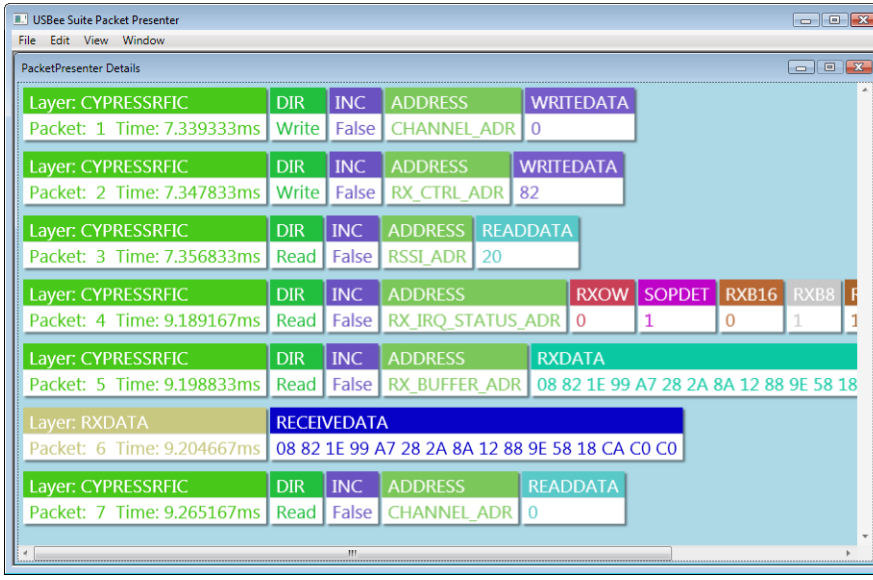


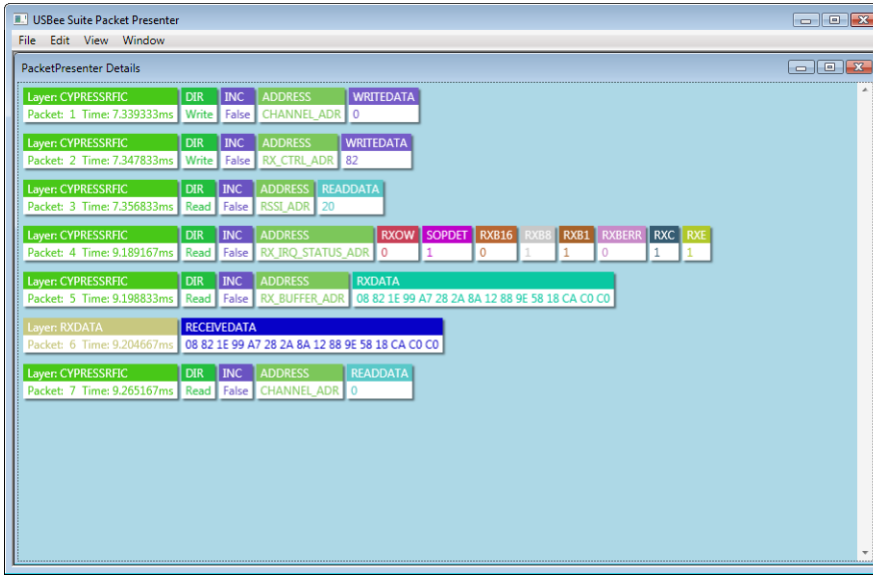
Below is a sample of packets copied in Text format and pasted into Notepad.

```
     Layer: CYPRESSRFIC         DIR     INC      ADDRESS     WRITEDATA
Packet:  1  Time: 7.339333ms   Write   False   CHANNEL_ADR       0

     Layer: CYPRESSRFIC         DIR     INC      ADDRESS     WRITEDATA
Packet:  2  Time: 7.347833ms   Write   False   RX_CTRL_ADR       82

     Layer: CYPRESSRFIC        DIR     INC     ADDRESS     READDATA
Packet:  3  Time: 7.356833ms   Read    False   RSSI_ADR       20

     Layer: CYPRESSRFIC        DIR     INC          ADDRESS         RXOW    SOPDET   RXB16   RXB8    RXB1    RXBERR    RXC    RXE
Packet:  4  Time: 9.189167ms   Read    False   RX_IRQ_STATUS_ADR    0       1        0       1       1       0       1      1

     Layer: CYPRESSRFIC        DIR     INC       ADDRESS                          RXDATA
Packet:  5  Time: 9.198833ms   Read    False   RX_BUFFER_ADR   08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0

       Layer: RXDATA                          RECEIVEDATA
Packet:  6  Time: 9.204667ms   08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0

     Layer: CYPRESSRFIC        DIR     INC      ADDRESS     READDATA
Packet:  7  Time: 9.265167ms   Read    False   CHANNEL_ADR       0
```

CHANGING THE PACKETPRESENTER SIZE

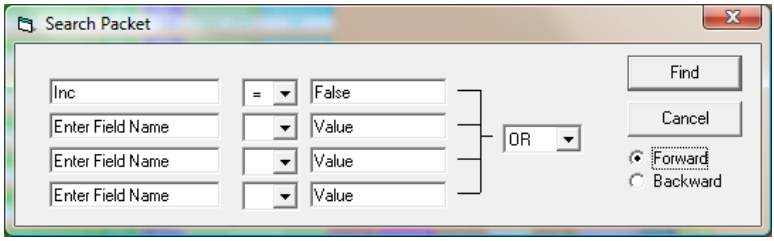
You can change the size of the fonts used by the PacketPresenter by selecting the View | Larger or View | Smaller menu items. Below are examples of different size fonts.





SEARCHING FOR PACKETS

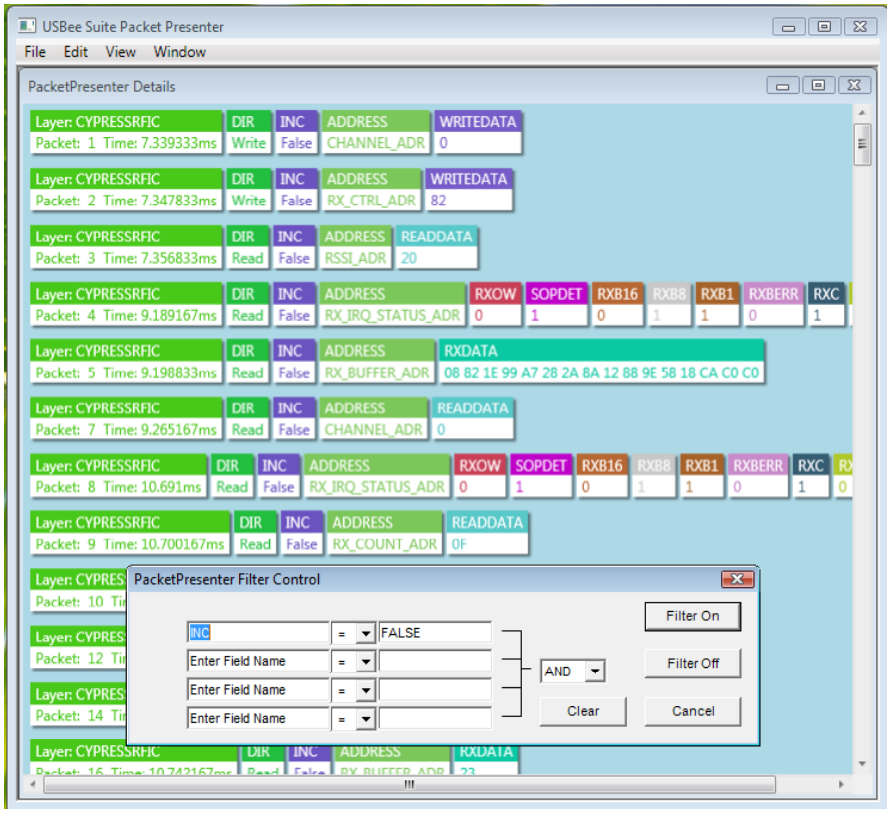
Once displayed, you can search for the next packet that contains certain fields that match your criteria. Below is the Search Packet dialog box that is shown by using the **View | Packet Search** menu item.



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

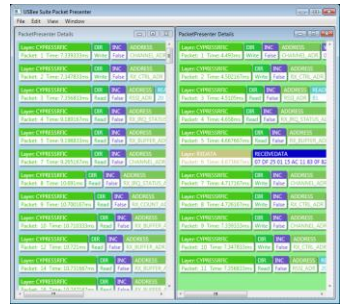
Once displayed, you can filter the output to only show packets that contains certain fields that match your criteria. Below is the Filter Packet dialog box that is shown by selecting the **View | Packet Filter** along with the resulting PacketPresenter output.

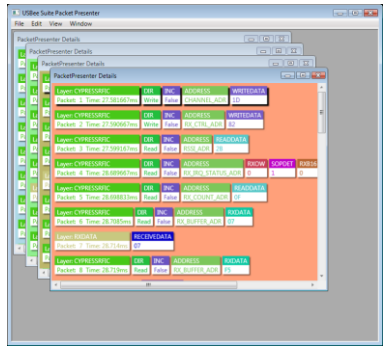


In the leftmost textboxes, type the Field Label. Then select the comparator operator (equals, not equals, less than, greater than…) and finally the value that the field is to be compared against. Finally, if there is more than one field in the search list, choose whether to AND or OR the search terms. 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

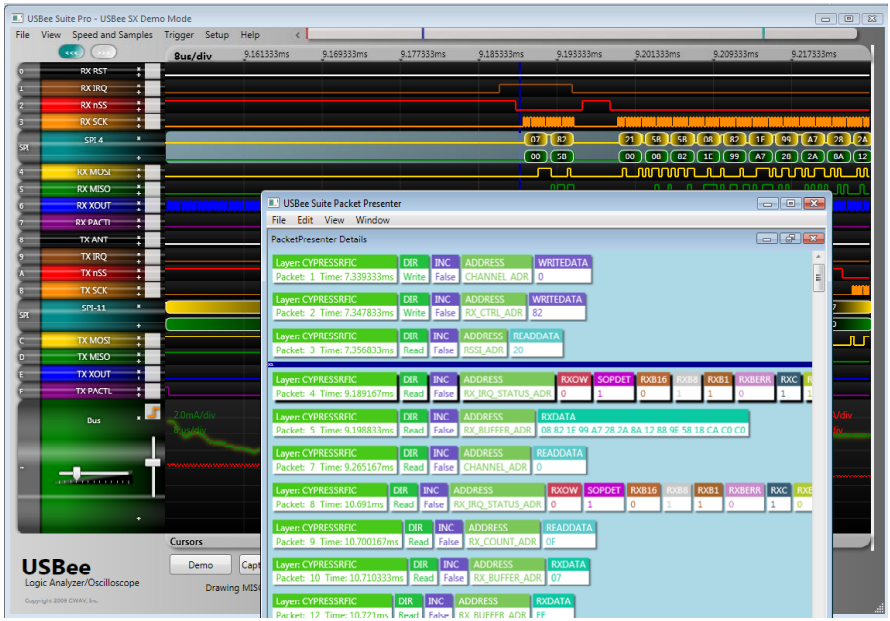
Using the **Window | Tile** menu you can choose to show the open windows Horizontally, Vertically or Cascaded as displayed below.





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.

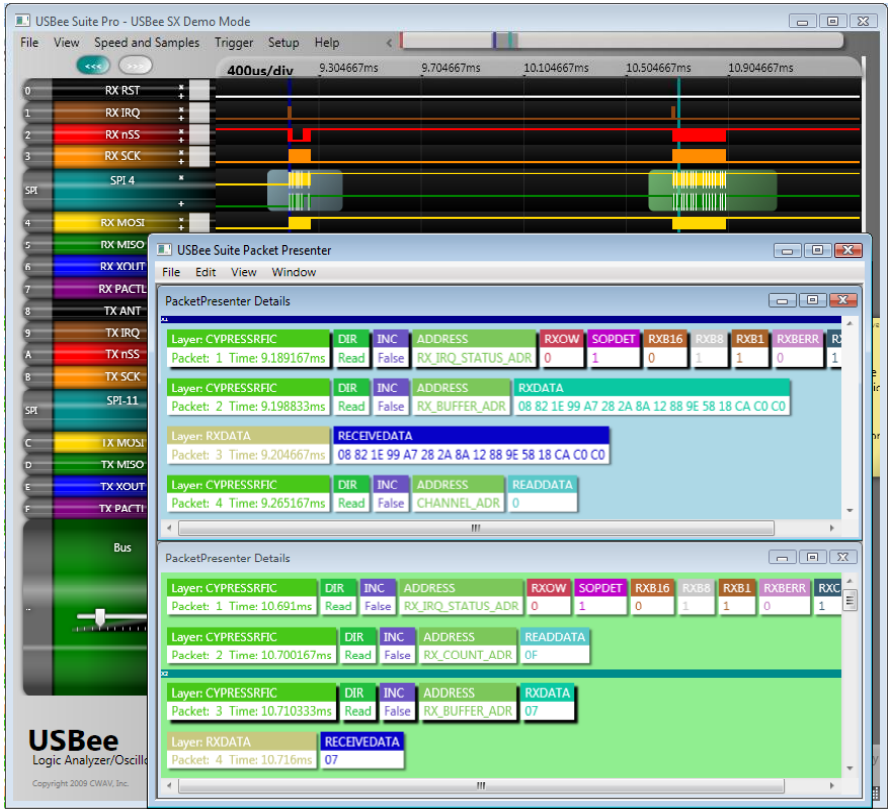


This feature allows you to correlate what is shown in the PacketPresenter window to the actual waveform on the logic analyzer that created that packet.

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 is shown in the Measurement Window.

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.



PACKETPRESENTER DEFINITION FILE FORMAT

Each PacketPresenter Definition file defines how the incoming data stream is represented in the PacketPresenter screen of the USBee Suite Pro 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 Suite Pro 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. All values are assumed MSB first.

```
[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
- B: invert the Byte order of this multibyte field
- 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 ***ALL*** [DEBUG] sections 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.

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

EXAMPLE PROTOCOL FILES AND OUTPUT EXAMPLES

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
```

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)
```

SPI PROTOCOL EXAMPLE

```
; 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
```

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
```

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
```

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
```

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
```

USB PROTOCOL EXAMPLE

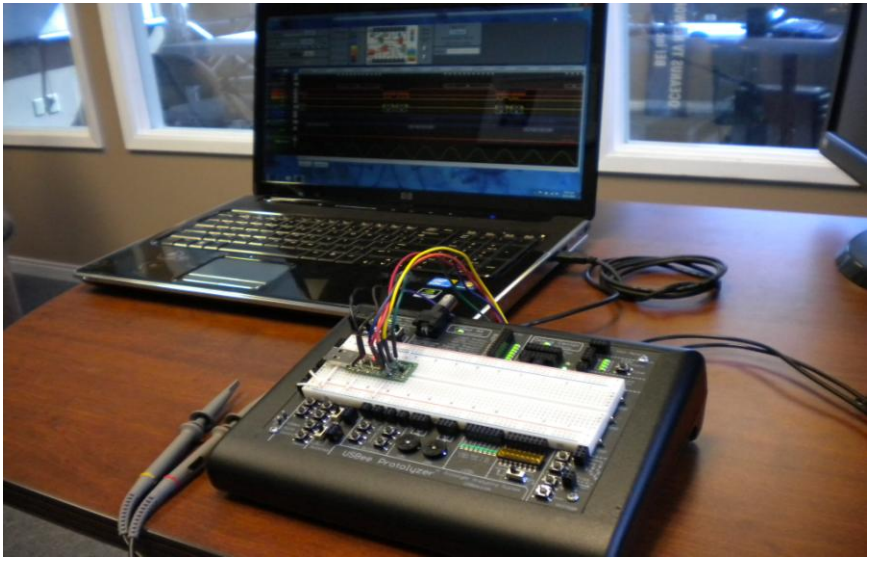
```
; USB Bus Protocol Definition File
; This file defines the transfers to/from a custom USB device
;
[Protocol]
          name = USBBus
          bitwise
[DEBUG]
          ;DebugOn            ; Uncomment this to turn on Debug Packets
[Packet]
          [Start]
                    type = event
                    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
```

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
```

USBEE PROTOLYZER CONTROL PANEL



The USBee Suite contains a USBee Protolyzer Control Panel that appears when any USBee Protolyzer is connected to the PC when the USBee Suite is run.

The USBee Protolyzer is a unique system specifically built for Electronic Prototyping with Built-In Oscilloscopes, Logic Analyzers, Signal Generators, Protocol Analyzers and more! The USBee Protolyzer combines state of the art electronic prototyping development components with a complete range of digital and analog test equipment in a single PC-based USB connected device.

Focus on designing your next greatest invention. When it's time to turn on the power, strap it to the USBee Protolyzer to have all the tools necessary to bring up your board, validate your design and analyze your systems performance. View your embedded firmware in action and monitor your bus protocols as never before. Combined with the USBee Suite, the USBee Protolyzer gives you the power to create!

With a single USB connection to your laptop or PC, the USBee Protolyzer gives you the power to design, prototype, test, and validate your mixed signal electronic designs with seamless ease.

INSTALLING THE USBEE SUITE WITH A USBEE PROTOLYZER

The USBee Protolyzer can be operated by itself, but to access all of its debugging features it must be attached to a PC running Windows XP, Vista or 7 or greater.

Installing and using the USBee Protolyzer involves the following steps.

1) PC Software Setup - The USBee Protolyzer contains up to 6 separate USB devices, all of which need to be installed on your PC. Follow the directions below to install all of the software necessary.

2) Connecting Probes

3) Connecting Your Protolyzer to the PC

3) Running the USBee Protolyzer Control Panel

4) Connecting Your Target Hardware

5) Refer to the USBee DX Users Manual, USBee AX Users Manual or the USBee Suite Users Manual for detailed operating instructions.

1) PC SOFTWARE SETUP

These steps will prepare your PC to operate the USBee Protolyzer. These steps only need to be completed once per PC.

- DO NOT plug in the USBee Protolyzer until the below software is installed on your PC.

- Download the USBee DX software from our USBee.com Downloads page. Make sure you download the version appropriate for your OS (32 or 64 bit).

- Run the USBee DX install program and follow the instructions until complete.

- Download the USBee AX software from our Downloads page.

- Run the USBee AX install program and follow the instructions until complete.

- Download and install the USBee Suite software from our Downloads Page. The USBee Suite software contains the Protolyzer Control Panel.

2) CONNECTING PROBES

With the USBee DX Protolyzer, you will need to connect the Oscilloscope Probes to the BNC connectors on the back panel, the 2x10 set of Digital Test Leads to the digital header, and the USB cable to the USB connector.

With the USBee AX Protolyzer, you will need to connect the Oscilloscope Probe to the BNC connector on the panel, the 1x11 set of Digital Test Leads to the digital header, and both USB cables between the USB connector and your PC.

For the Oscilloscope probes, make sure that the probe is turned to lock the connector in place. Each Oscilloscope probe has a x1 and x10 switch that lets you choose the input voltage level. Set it to x1 for - 10V to +10V. Make sure that the Digital Test leads are pushed fully onto the pins, and that the two ground leads are on the left side when looking at the back of the unit.

The USB connector supplies all of the power and communications for the USBee Protolyzer. If the USB connector is unplugged, you will need to restart the USBee Suite.

3) CONNECTING YOUR PROTOLYZER TO THE PC

Now plug in the USBee Protolyzer into your PC. For the USBee DX Protolyzer, there is a single USB cable that connects from the back of the unit to your PC. For the USBee AX Protolyzer there are two USB connectors on the bottom of the board that must be connected to two USB connectors on your PC. One USB cable provides the power and USBee AX connection. The other USB connector provides

the Debug Center and Protolyzer Control Panel functionality. In the USBee DX Protolyzer unit, these two connections are made internally using an internal USB hub.

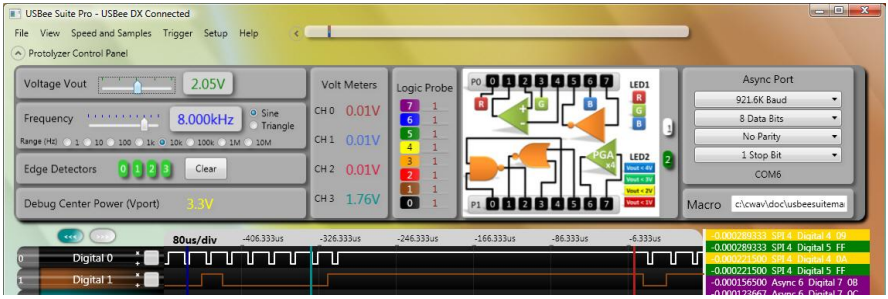
You will receive MANY New Hardware found indications. Accept or Continue to all prompts until they are complete. Remember, there are over 6 physical USB devices that need to be installed in the USBee Protolyzer.

Now you can run the USBee Suite software (with the Protolyzer Control Panel) or any of the USBee Test Pod applications for your version of Protolyzer.

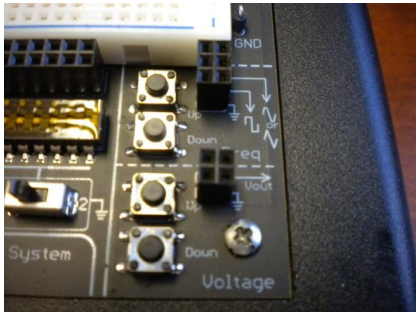
RUNNING THE USBEE PROTOLYZER CONTROL PANEL

The heart of the USBee Protolyzer system is the USBee Protolyzer Control Panel. To access the Protolyzer Control Panel, run the USBee Suite software with the Protolyzer plugged into the PC.

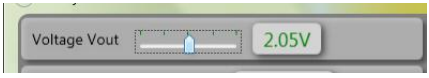
The software will auto-detect the presence of the Protolyzer and enable the Protolyzer Control Panel as shown below.



Each section of the USBee Suite Protolyzer Control Panel controls a different part of the Protolyzer. The sectons below detail the operation of the various Control Panel functions:



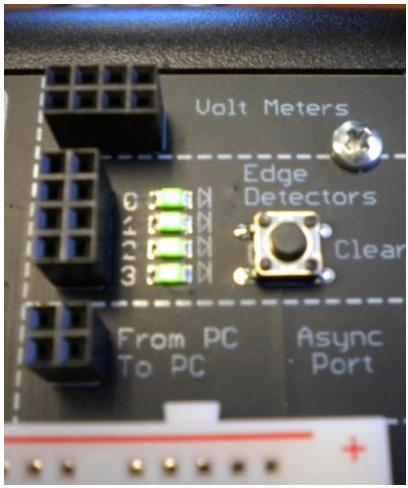
Voltage Out level - The header labeled "Voltage" on the Protolyzer provides a constant output voltage. You can change the voltage using either the buttons (Up and Down) on the board, or by using the following slider in the Protolyzer Control panel below.



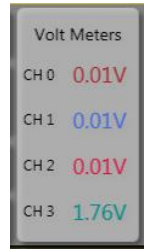
Frequency and Sine/Triangle setting

The header labeled "Freq" on the Protolyzer provides a set of waveforms, one which is either a triangle or sine wave, and another which is a square wave. You can change the setting and frequency using either the buttons (Up and Down) on the board, or by using the following slider in the Protolyzer Control panel below. You

can change the range as well with the radio buttons of the setting.



Volt Meters - measure up to 4 channels of DC voltage from GND to VPort. Great for battery monitoring, reference voltages and logic levels. Simply apply your desired voltage and view the reading in the Control Panel as shown.



Edge Detector states and clear - Detects if the signals ever cross the Vport/2 voltage threshold. Need to run a test over the long haul? Make it toggle a pin in the event of an error. These LEDS will then light if this event ever happens. The current state is also available on the Control Panel.

Async Port - This port shows up as a COM port on the PC. Simply choose your baud rate and your device can send data to Hyperterminal, putty, etc., as well as be controlled by the PC. The COM port number is displayed in the USBee Suite Protolyzer Control Panel. The settings are fixed at 8 data bits, no parity and one stop bit. The Baud rate can be changed by clicking on the baud rate.

USBee Suite Control Keys

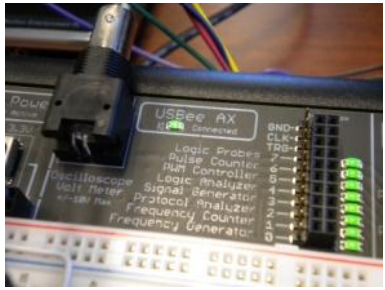
There are keys on the surface of the Protolyzer that allow you to start and stop an acquisition and navigate your captured data by panning and zooming. Just press the buttons to take a new trace and find your problems without having to leave your work surface.

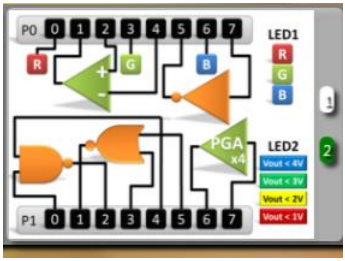
Macro Button

You can assign any command line to execute whenever you press the Macro button on the Protolyzer surface. You can type the name of a file to open, a URL to navigate to, or a program to run as shown here.

Logic Probe

Included with the complete USBee AX Test Pod is a Logic Probe that lets you see the state of the signals without taking traces. If the light is lit, the signal is logic high (1). If it is off it is logic low (0). Blinking means it is toggling.

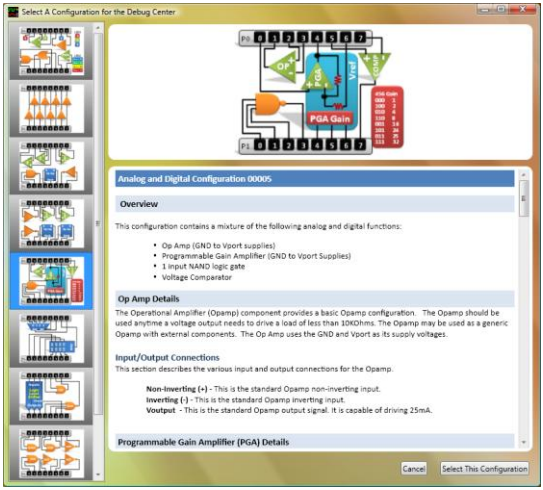




Debug Center

The Debug Center is like having an entire set of digital and analog electronic components available at your demand. Click on the Schematic in the USBee Suite to configure the Debug Center to the components you need in less than 10 seconds. Once you hear the whistle your new setting is active.

When you click on the schematic, the following screen will appear to let you select which configuration is used.



Some of the types of components that can be selected are:

- Op Amps and Voltage Followers
- Comparators
- Programmable Gain Amplifiers
- Digital Logic (OR, AND, NAND, NOR, XOR, NOT, Multiplexers / Demultiplexers)
- D Flip Flops
- Logic Level Shifters
- And more

For specifications of the USBee Protolyzer, please refer to the USBee Protolyzer Specification available at USBee.com.

GETTING HELP

We are always eager to help you to get the most out of all USBee products. If you have any questions, comments, bug reports or suggestions, please contact us. We actively improve our product line and your feedback is the key to making the USBee the best embedded development tool on the market.

Email us at support@usbee.com

or

Call us at (951) 694-6808

Printed in the USA

Version 2.5