Envy (en-vee) n. What your coworkers will feel when you solve your bugs and go home.
CH1
CH2
USBee RX
www.usbee.com
Fast Sampling | Huge Buffer | Unmatched Features | Pocket-sized
100 Msps, 16 million transitions, +/-60V, 18 channel Logic Analyzer, 2 channel Oscilloscope, 9 channel Signal Generator and Protocol Analyzer

USBEE RX

USERS MANUAL

Interworld Elecronics, Inc.

www.interworldna.com - Distributor

USBEE RX

USERS MANUAL

Interworld Inc.
www.interworlna.com
(877) 902-2979
support@interworldna.com

USBee RX 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 RX Suite software. You have received or downloaded the USBee RX Suite Package ("Software"), which consists of the USBee RX 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 RX 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 RX User's Manual, Version 1.0

CONTENTS

INTRODUCING THE USBEE RX POD

The Tools You Need - All In One Place

Introducing the Ultimate PC-Based Programmable Multifunction Mixed Signal Oscilloscope - Logic Analyzer with I2C, SPI, Async, SDIO, 1-Wire, CAN protocol decoders, Digital Signal Generators, Frequency Counter and integrated Protocol Analyzers in a compact and easy to use USBee Test Pod.

Designed to Solve Your Toughest Problems

Using patented technology, the USBee RX has 2 analog and 18 digital channels, 100MHz sampling, 512 million bit internal buffers, Dual 10-bit ADC's, independent 100Msps 8 channel Digital Signal Generator, 7Msps Analog Signal Generator, +/- 60V protection on all inputs, Variable Logic Thresholds and USB overcurrent protection. And that's just the hardware.

Mixed-Signal Multi-level Protocol Decoding

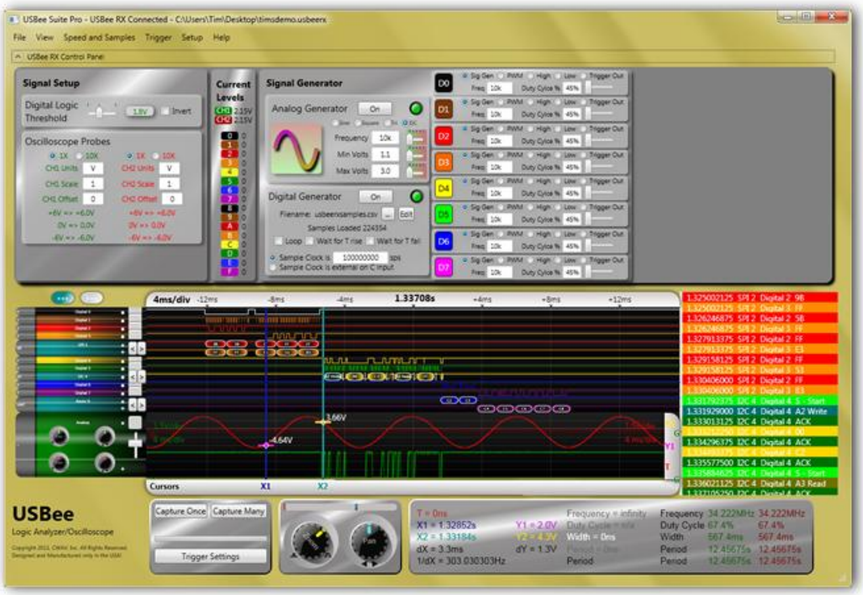
Combined with the USBee RX Suite software, the USBee RX becomes a multi-layer protocol decoder for many of today's serial and parallel busses, letting you debug at the waveform, bus data, or packet level.

In-Field Upgradable System Supports Addition of New Features

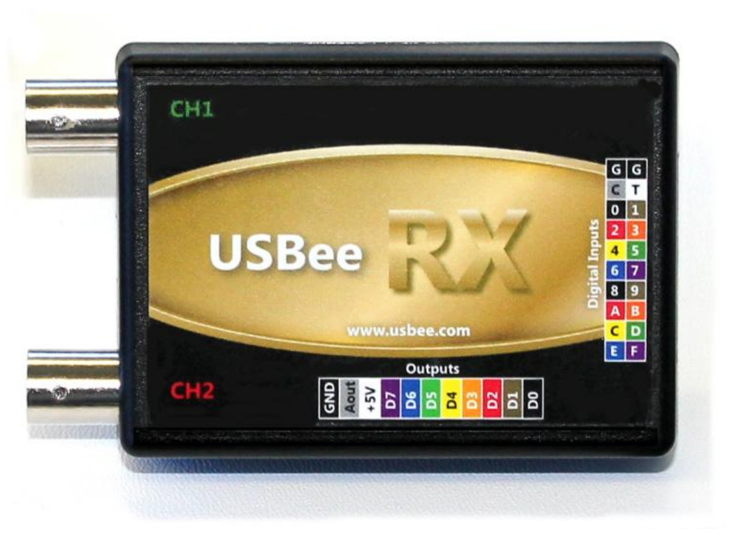
Based on a proprietary design that allows the software, firmware and hardware to be dynamically reconfigured using web-based software downloads, the USBee RX and its ever growing list of Applications ensures that you will always have the most current tools available for years to come.

With a single USB connection to your laptop or PC, the USBee RX gives you the power to design, prototype, test, and validate your mixed signal electronic designs with seamless ease.   Built upon the popular, award-winning and patented USBee mixed signal Oscilloscope, Logic Analyzer, Signal Generator, and Protocol Analyzers, the USBee RX takes full advantage of the powerful debug capabilities of the USBee RX Suite software to solve your problems fast!

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



THE USBEE RX TEST POD



The USBee RX Test Pod has four sets of connectors:

- USB
  - Connects to the PC via 6 foot USB cable
- CH1 and CH2
  - Analog Input Channels
  - BNC connectors for standard oscilloscope probes
  - +/- 60V tolerant
- Outputs connector
  - 11 pins – 1 Ground (GND), 1 Analog Aout, 1 +5V Supply, and 8 Digital Outputs (D0-D7)
  - 0.1” spaced 0.025” square header pins
  - 8 Digital output pins (0-3.3V logic) – D0 – D7
  - +5V output – actually VBus connected directly to the USB VBus signal from the PC through a resettable fuse
  - 1 Aout analog output pin used by the Analog Signal Generator function (0-3V levels)
  - Connects to 11x1 pin test lead set
- Digital Inputs Connector
  - 20 pins (10x2) – 2 Ground (G), Clock Input (C), Trigger Input (T), and 16 Digital inputs (0-F)
  - 0.1” spaced 0.025” square header pins
  - +/-60V tolerant inputs
  - Connects to 10x2 pin test lead set

WARNING

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

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

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

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

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

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

Also,

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

Specifically:

- ALWAYS connect at least one GND line to your circuits ground
- NEVER connect the digital signal lines (0 thru 7, TRG and CLK) to any voltage other than between -60 to +60 Volts
- NEVER connect the analog signal lines (CH1 and CH2) to any voltage other than between -60 and +60 Volts
- The USBee RX actively drives Pod signals D0 through D7 and AOUT. Make sure that these pod test leads are either unconnected or connected to signals that are not also driving. Connecting these signals to other active signals can cause damage to you, your circuit under test or the USBee RX test pod, for which CWAV is not responsible.
- Plug in the USBee RX Pod into a powered PC BEFORE connecting the leads to your design.

PC SYSTEM REQUIREMENTS

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

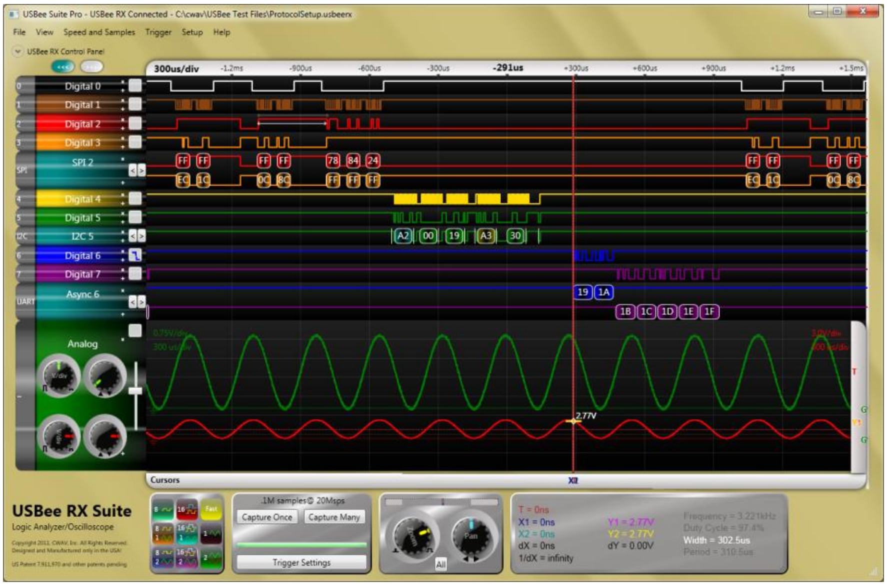
- Windows® XP, Vista or 7 32-bit or 64-bit operating system
- Pentium or higher processor
- One USB2.0 High Speed enabled port. It will not run on USB 1.1 Full Speed ports.
- 32MBytes of RAM
- 125MBytes of Hard disk space
- Internet Access (for software updates and technical support)

EACH PACKAGE INCLUDES

The USBee RX contains the following in each package:

- USBee RX Test Pod
- 2 100MHz oscilloscope probes with x1 and x10 selection
- 11 lead 9-inch signal generator cable set
- 20 lead 9-inch logic analyzer cable set
- 20 high performance micro grabber test clips
- 6 foot USB cable
- USBee RX Suite Software (downloaded)
- USBee RX and USBee RX Suite Manuals (downloaded)

INTRODUCING THE USBEE RX SUITE

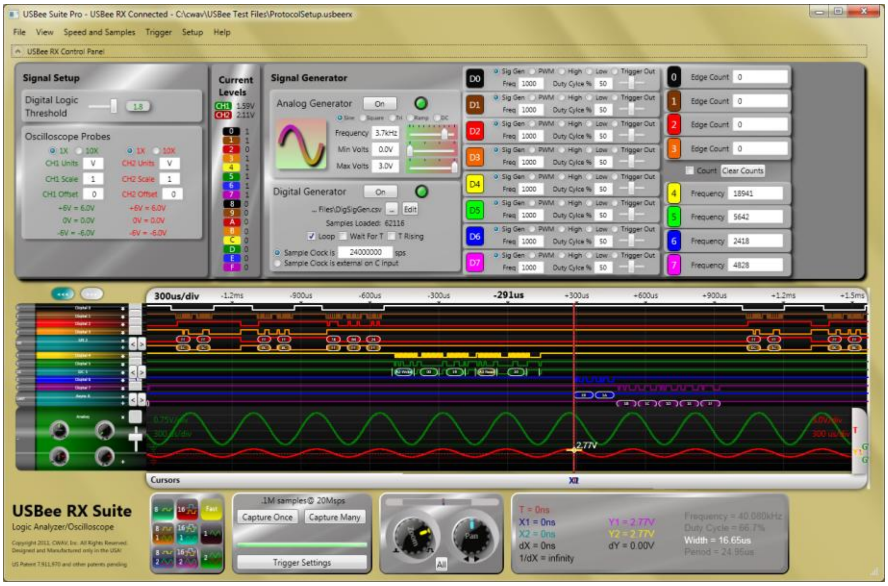


The USBee RX Suite is powerful electronic signal analysis software for your USBee RX Test Pod. It starts out as an easy to use Logic Analyzer, Oscilloscope, Signal Generator, Frequency Counter and PWM Controller and adds serial bus decoding and world class configurability that lets you solve your electronic problems quickly! This chapter details the operation of the USBee RX Test Pod running the USBee RX Suite software.

USBEE RX SUITE OVERVIEW

The USBee RX Suite is a powerful mixed signal analyzer that on the USBee RX Test Pod. It is available for free from USBee.com and will run without restriction.

The USBee RX Suite Standard will run in Demo mode without a USBee, or on any USBee RX Test Pod. It will not work on any other USBee test Pod model.



USBee RX Control Panel Visible

SEE THE INFORMATION YOU WANT FAST!

Setup of the USBee RX 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 100 million bytes per second for sample buffer depths of up to 16 million samples or 8 million transitions.

SUPERIOR QUALITY DESIGN - PROFESSIONAL RESULTS

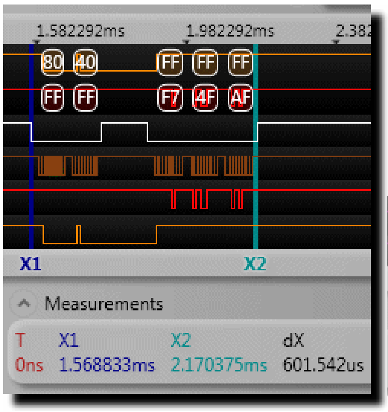
The USBee RX Suite takes full advantage of the power the USBee RX Test Pod. Each USBee RX comes with the best color coded highly flexible test leads, the best test clips and our signature small and sleek design that can fit right in your pocket. We are proud to say the entire USBee product line is designed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

FAST AND DETAILED WAVEFORM VIEWING

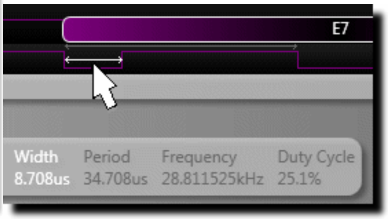
The USBee RX Suite lets you capture a huge amount of data. Go exactly to the section of that data you want using the Quick Zoom with your mouse scroll wheel, or use the Overview bar to rip through your millions of samples or hone in on a specific section.

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.



T	X1	X2	dX
0ns	1.568833ms	2.170375ms	601.542us

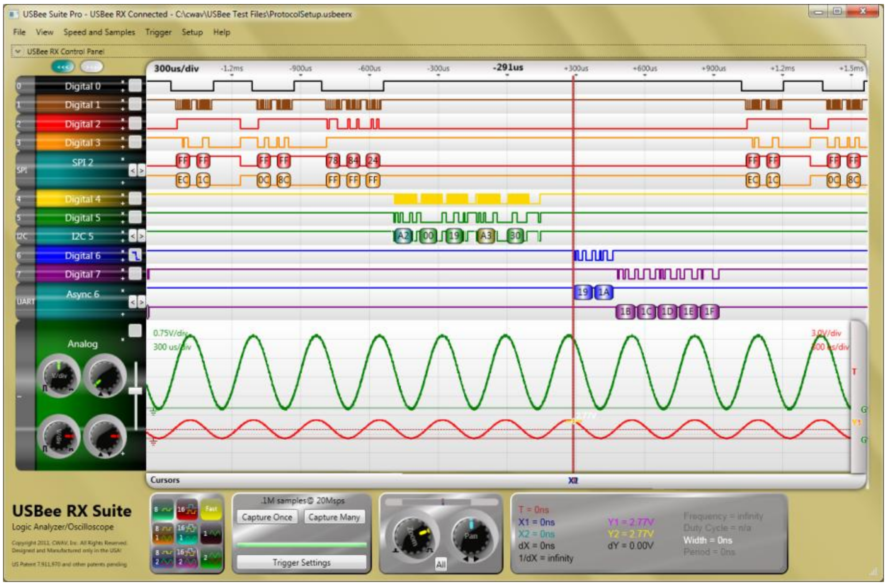


Width	Period	Frequency	Duty Cycle
8.708us	34.708us	28.811525kHz	25.1%

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!

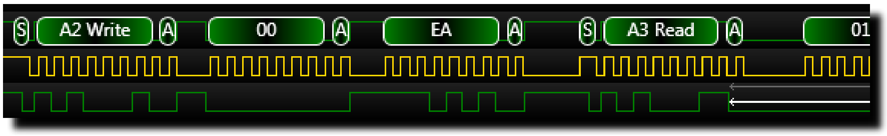
And you like Magenta? Well you can change cursor colors to suite your desires. Waveform backgrounds can also be customized, and you can even give the entire application that cool Glassy look that Vista has made so popular. Then again, if you like simple, white and black are also available. It's good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



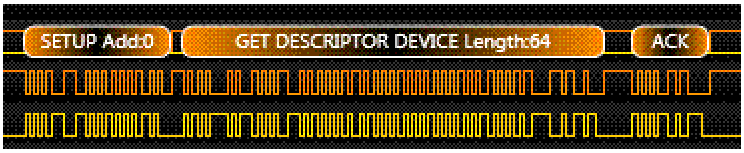
USBee RX Suite showing SPI, I2C and Async decoding

SERIAL BUS ANALYSIS

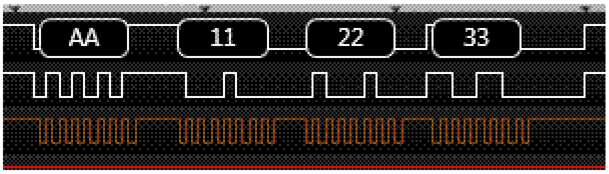
USBee RX Suite has decoding support for your favorite serial busses such as I2C, SPI and Async. Bus traffic is decoded in-line with the waveforms and can be displayed on top of, underneath, or instead of the voltage versus time waveform. Just place the cursor over the decoded traffic and get a see-through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



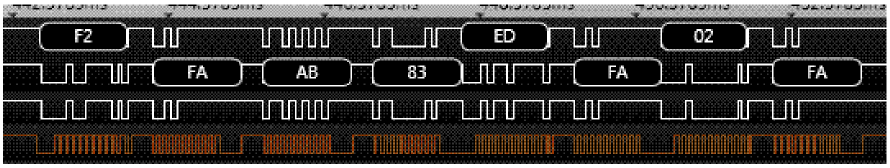
I2C Transaction



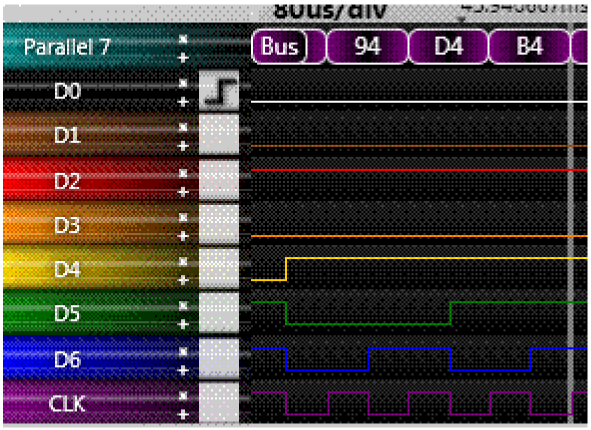
USB Transaction



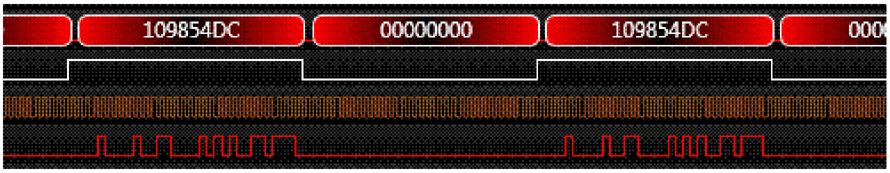
Synchronous Serial Transaction



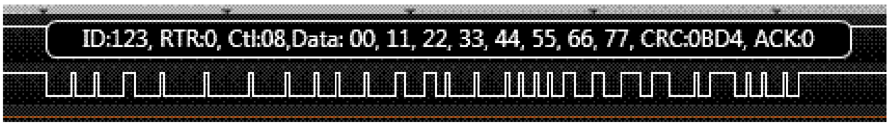
PS2 Transaction



Parallel Bus Decode

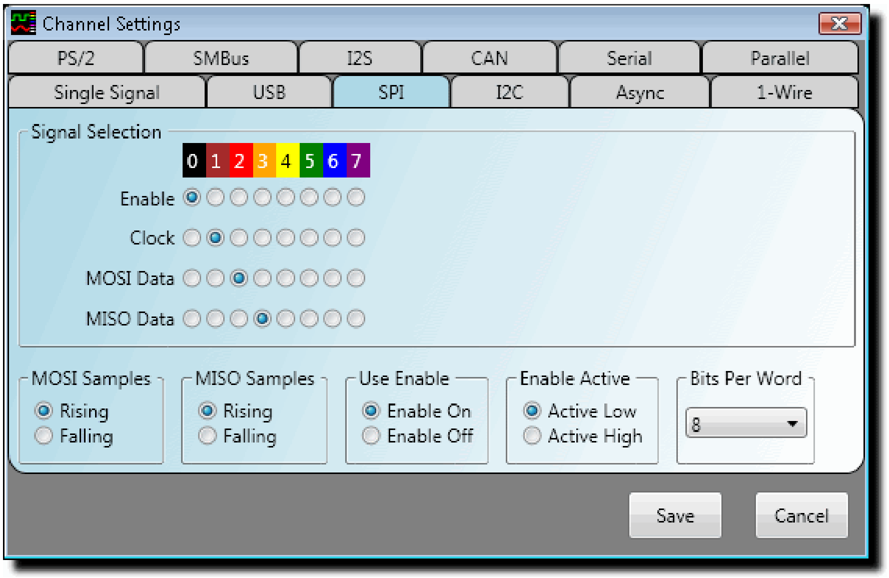


I2S Decode


ID:123, RTR:0, Ctl:08,Data: 00, 11, 22, 33, 44, 55, 66, 77, CRC:0BD4, ACK:0

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!



Signal Selection

	0	1	2	3	4	5	6	7
Enable	●	○	○	○	○	○	○	○
Clock	○	●	○	○	○	○	○	○
MOSI Data	○	○	●	○	○	○	○	○
MISO Data	○	○	○	●	○	○	○	○

MOSI Samples: [✓ Rising] Falling

MISO Samples: [✓ Rising] Falling

Use Enable: [✓ Enable On] Enable Off

Enable Active: [✓ Active Low] Active High

Bits Per Word: 8

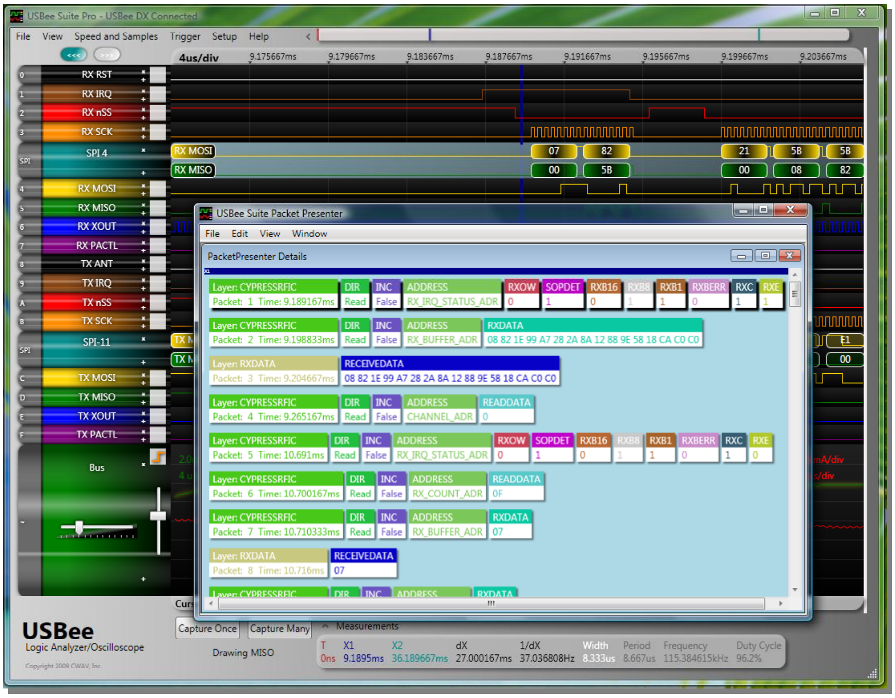
(USBee SX options shown above)

DATA STORAGE

Save your entire data capture to file quickly using the USBee RX 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!

PACKETPRESENTER

The USBee RX Suite 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.



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

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

Layer: RXDATA Packet: 3 Time: 9.204667ms | RECEIVEDATA 08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0

Layer: CYPRESSRFIC Packet: 4 Time: 9.265167ms | DIR Read | INC False | ADDRESS CHANNEL_ADR | READDATA 0

Layer: CYPRESSRFIC Packet: 5 Time: 10.691ms | DIR Read | INC False | ADDRESS RX_IRQ_STATUS_ADR | RXOW 0 | SOPDET 1 | RXB16 0 | RXB8 1 | RXB1 1 | RXBERR 0 | RXC 1 | RXE 0

Layer: CYPRESSRFIC Packet: 6 Time: 10.700167ms | DIR Read | INC False | ADDRESS RX_COUNT_ADR | READDATA 0F

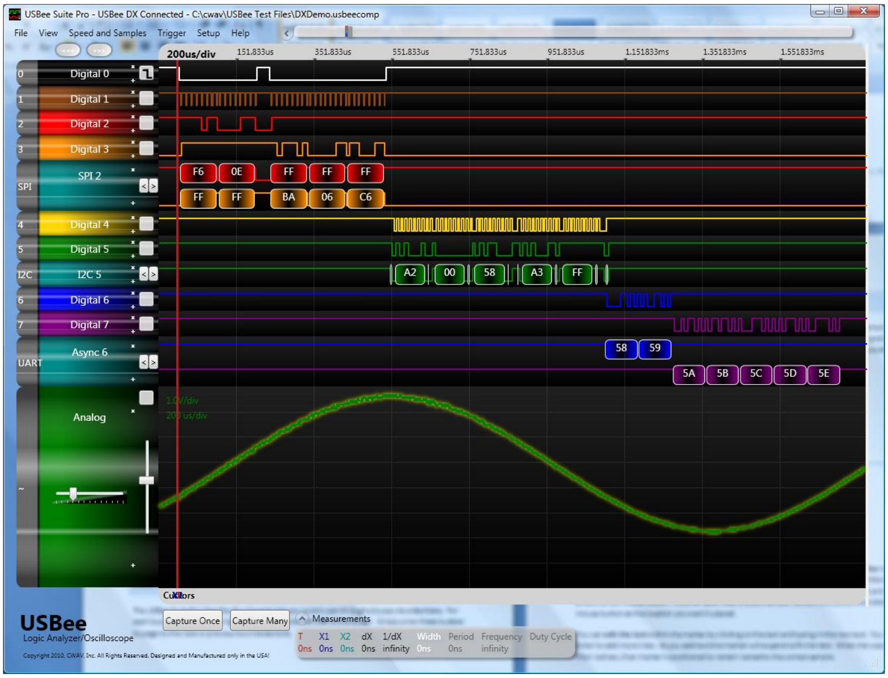
Layer: CYPRESSRFIC Packet: 7 Time: 10.710333ms | DIR Read | INC False | ADDRESS RX_BUFFER_ADR | RXDATA 07

Layer: RXDATA Packet: 8 Time: 10.716ms | RECEIVEDATA 07

T	X1	X2	dX	1/dX	Width	Period	Frequency	Duty Cycle
0ns	9.1895ms	36.189667ms	27.000167ms	37.036808Hz	8.333us	8.667us	115.384615kHz	96.2%

FAST PAN BUS VIEWING

The USBee RX Suite 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.

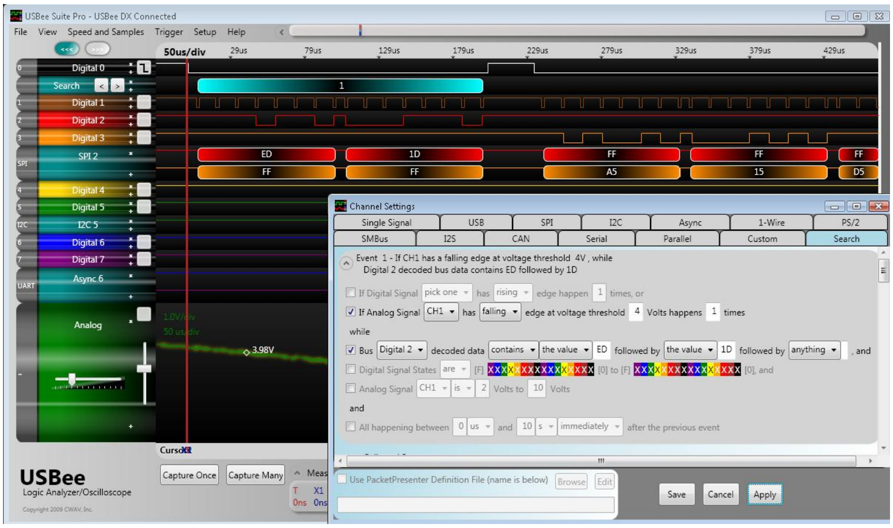


SMART SEARCH

USBee RX Suite 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.



USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
50us/div
29us 79us 129us 179us 229us 279us 329us 379us 429us
Digital 0
Search
Digital 1
Digital 2
Digital 3
SPI 2
Digital 4
Digital 5
I2C 5
Digital 6
Digital 7
Async 6
Analog
ED 1D FF FF FF
FF FF A5 15 D5
3.98V
Cursors
Capture Once
Capture Many
Meas
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search

Event 1 - If CH1 has a falling edge at voltage threshold 4V , while
Digital 2 decoded bus data contains ED followed by 1D

- [ ] If Digital Signal pick one has rising edge happen 1 times, or
- [✓] If Analog Signal CH1 has falling edge at voltage threshold 4 Volts happens 1 times

while

- [✓] Bus Digital 2 decoded data contains the value ED followed by the value 1D followed by anything , and
- [ ] Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and
- [ ] Analog Signal CH1 is 2 Volts to 10 Volts

and

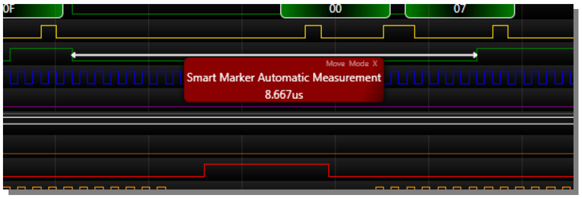
- [ ] All happening between 0 us and 10 s immediately after the previous event

- [ ] Use PacketPresenter Definition File (name is below) Browse Edit

Save Cancel Apply

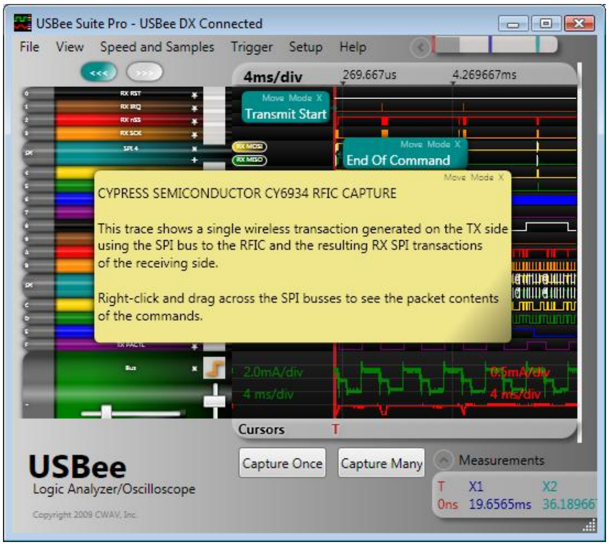
SAMPLE AND SMART MARKERS

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 RX Suite 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.



CYPRESS SEMICONDUCTOR CY6934 RFIC CAPTURE

This trace shows a single wireless transaction generated on the TX side using the SPI bus to the RFIC and the resulting RX SPI transactions of the receiving side.

Right-click and drag across the SPI busses to see the packet contents of the commands.

T	X1	X2
0ns	19.6565ms	36.18966

ACQUISITION CONTROL

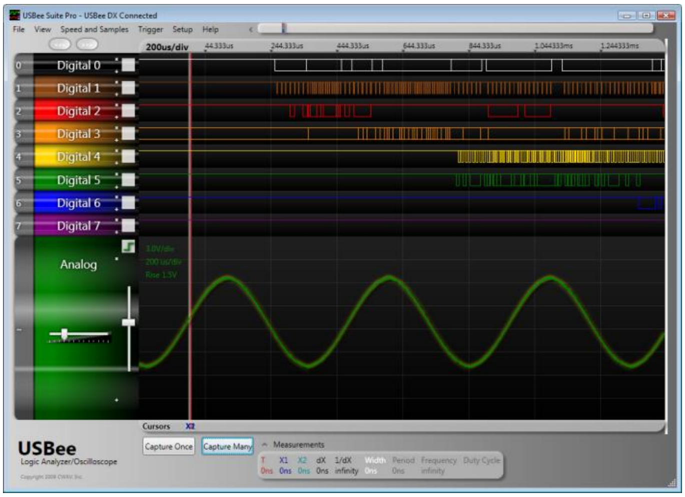
The USBee RX Suite adds more trace acquisition and triggering controls such as Normal Mode, Automatic Mode, Single Capture and Multiple Capture.

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.

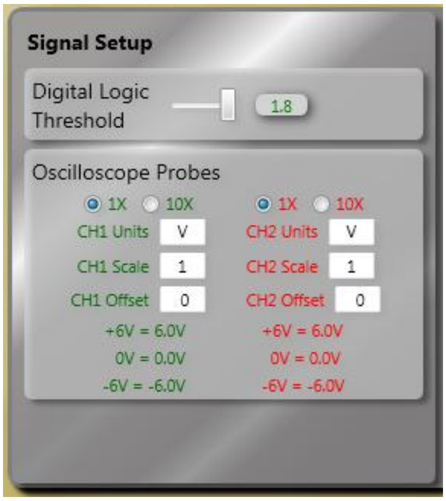
DISPLAY MODES

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



Signal Setup

Digital Logic Threshold 1.8

Oscilloscope Probes

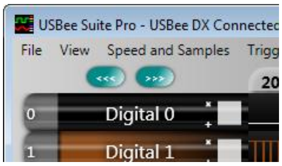
[✓ 1X] 10X		[✓ 1X] 10X	
CH1 Units	V	CH2 Units	V
CH1 Scale	1	CH2 Scale	1
CH1 Offset	0	CH2 Offset	0
+6V = 6.0V		+6V = 6.0V	
0V = 0.0V		0V = 0.0V	
-6V = -6.0V		-6V = -6.0V	

USBEE SUITE DATA FILE IMPORTING

The USBee DX, AX or ZX running the USBee Suite software saves files in their own file format. These older files can be imported into the USBee RX Suite.

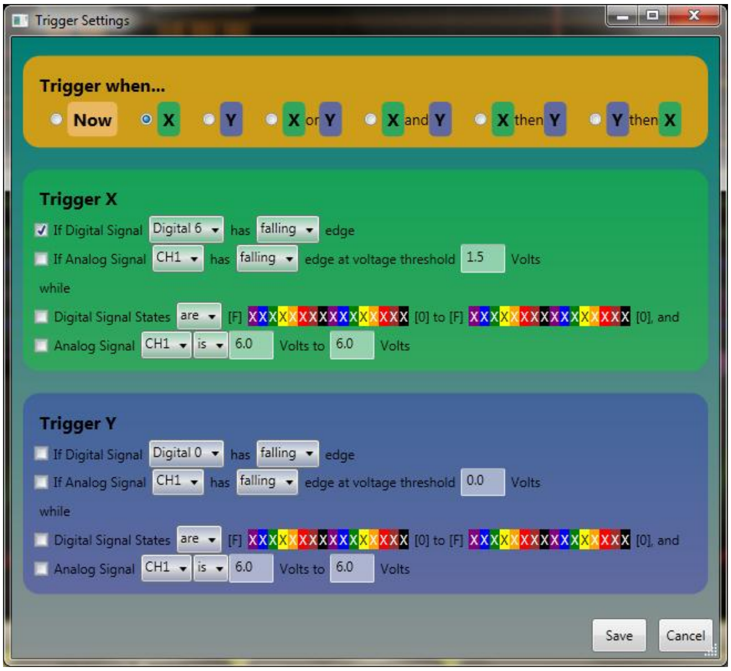
BROWSER-LIKE NAVIGATION

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



COMPLEX TRIGGERING

The USBee RX Suite adds a multi-level hardware trigger to capture the events you need to see to solve your problems.



Trigger Settings

Trigger when...

○ Now [✓ X] ○ Y ○ X or Y ○ X and Y ○ X then Y ○ Y then X

Trigger X

☑ If Digital Signal Digital 6 has falling edge

☐ If Analog Signal CH1 has falling edge at voltage threshold 1.5 Volts

while

☐ Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and

☐ Analog Signal CH1 is 6.0 Volts to 6.0 Volts

Trigger Y

☐ If Digital Signal Digital 0 has falling edge

☐ If Analog Signal CH1 has falling edge at voltage threshold 0.0 Volts

while

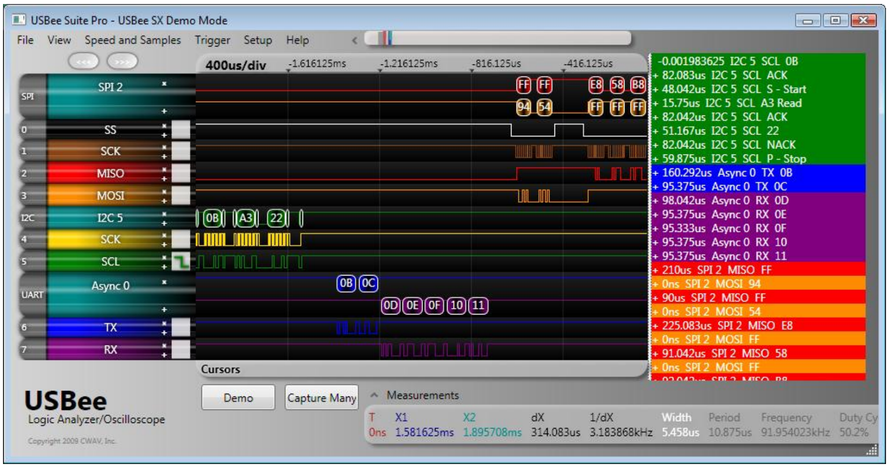
☐ Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and

☐ Analog Signal CH1 is 6.0 Volts to 6.0 Volts

Save Cancel

RELATIVE TIME DECODE

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

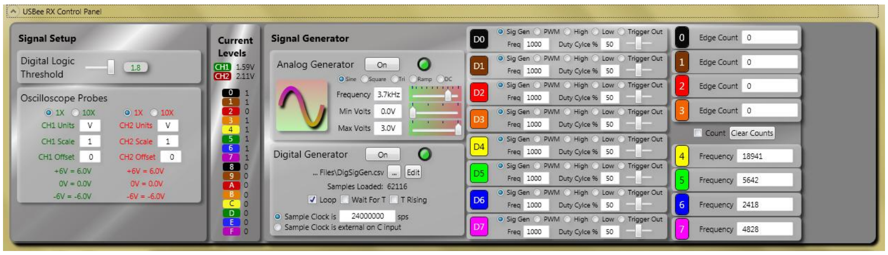


```
-0.001983625  I2C 5  SCL  0B
+ 82.083us  I2C 5  SCL  ACK
+ 48.042us  I2C 5  SCL  S - Start
+ 15.75us  I2C 5  SCL  A3 Read
+ 82.042us  I2C 5  SCL  ACK
+ 51.167us  I2C 5  SCL  22
+ 82.042us  I2C 5  SCL  NACK
+ 59.875us  I2C 5  SCL  P - Stop
+ 160.292us  Async 0  TX  0B
+ 95.375us  Async 0  TX  0C
+ 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
```

T	X1	X2	dX	1/dX	Width	Period	Frequency	Duty Cy
0ns	1.581625ms	1.895708ms	314.083us	3.183868kHz	5.458us	10.875us	91.954023kHz	50.2%

USBEE RX CONTROL PANEL

The USBee RX Suite contains a USBee RX Control Panel that controls many of the features of the USBee RX system, including the Variable Logic Threshold, Analog channel scaling, Current Logic and Analog levels, Analog Signal Generator, Digital Signal Generator, PWM controllers, Pulse Counters and Frequency Counters.



Digital Logic Threshold: 1.8

Oscilloscope Probes

[✓ 1X] 10X	[✓ 1X] 10X
CH1 Units: V	CH2 Units: V
CH1 Scale: 1	CH2 Scale: 1
CH1 Offset: 0	CH2 Offset: 0
+6V = 6.0V	+6V = 6.0V
0V = 0.0V	0V = 0.0V
-6V = -6.0V	-6V = -6.0V

CH1 1.59V
CH2 2.11V

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

Analog Generator: On
[✓ Sine] Square Tri Ramp DC
Frequency: 3.7kHz
Min Volts: 0.0V
Max Volts: 3.0V

Digital Generator: On
... Files\DigSigGen.csv ... Edit
Samples Loaded: 62116
[✓ Loop] Wait For T T Rising
[✓ Sample Clock is 24000000 sps]
Sample Clock is external on C input

D0	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50
D1	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50
D2	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50
D3	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50
D4	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50
D5	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50
D6	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50
D7	[✓ Sig Gen] PWM High Low Trigger Out	Freq 1000	Duty Cylce % 50

0	Edge Count	0
1	Edge Count	0
2	Edge Count	0
3	Edge Count	0

Count Clear Counts

4	Frequency	18941
5	Frequency	5642
6	Frequency	2418
7	Frequency	4828

QUICK START

This section goes through installing the software for your USBee RX and getting you using the USBee RX quickly.

PC SYSTEM REQUIREMENTS

The USBee RX Suite requires the following minimum PC features:

- Windows® XP SP3, Vista or Windows 7 32-bit or 64-bit operating system
- .NET Framework 4.0 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

To ensure that you are using the latest version of USBee RX software, you can download the software from our web site at www.usbee.com. You must install the software on the PC before you plug in the USBee RX device.

A quick guide to install the software follows:

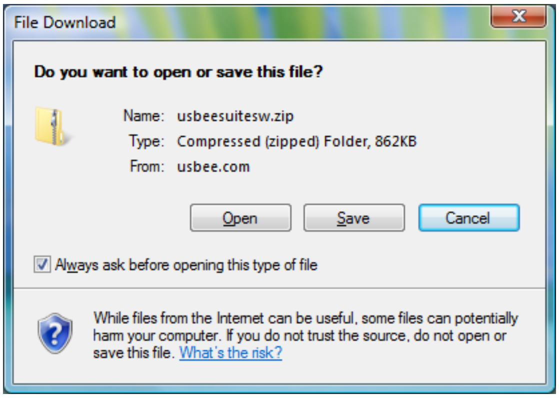
- Download the USBee RX Software from http://www.usbee.com/download.htm and open the usbeerxsuitesw.zip file.
- Run the SETUP.EXE.
- Follow the instructions on the screen to install the USBee RX software on your hard drive. This may take several minutes.
- Now, plug a USB A to USB Mini-B cable into the USBee RX and the other end into a free USB 2.0 High Speed port on your computer.
- You will see a dialog box indicating that it found new hardware and is installing the software for it. Follow the on screen directions to finish the driver install.
- The USBee RX Software is now installed.
- Run the USBee RX Suite software by going to the Start | Program Files |USBee RX Suite.
- If your PC does not already have .NET Frameworks version 4.0 or greater, you will be notified to download and install this from the Microsoft web site.

DETAILED SOFTWARE INSTALLATION

The USBee RX 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 RX Pod installed and attached. To install the software for demo purposes, just install the USBee RX Suite software.

To install the USBee RX Suite software:

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


File Download

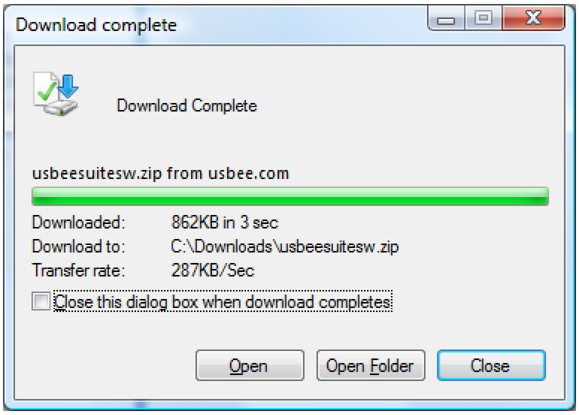
Do you want to open or save this file?

Name: usbeesuitesw.zip
Type: Compressed (zipped) Folder, 862KB
From: usbee.com

☑ Always ask before opening this type of file

While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?

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


Download complete

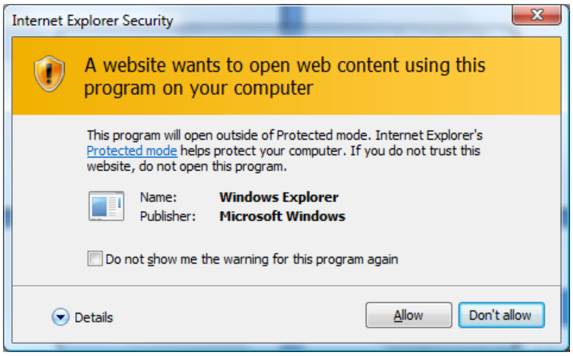
Download Complete

usbeesuitesw.zip from usbee.com

Downloaded: 862KB in 3 sec
Download to: C:\Downloads\usbeesuitesw.zip
Transfer rate: 287KB/Sec

☐ Close this dialog box when download completes

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


Internet Explorer Security

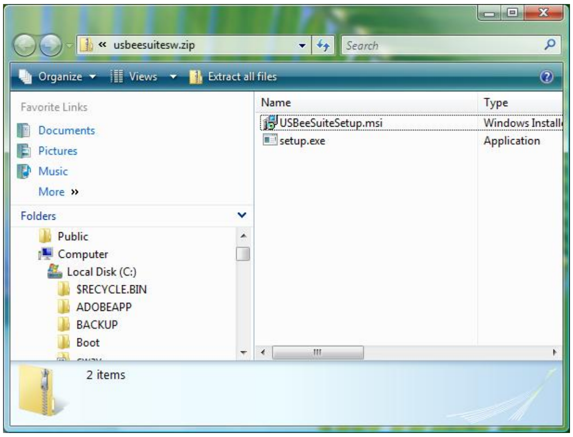
A website wants to open web content using this program on your computer

This program will open outside of Protected mode. Internet Explorer's Protected mode helps protect your computer. If you do not trust this website, do not open this program.

Name: **Windows Explorer**
Publisher: **Microsoft Windows**

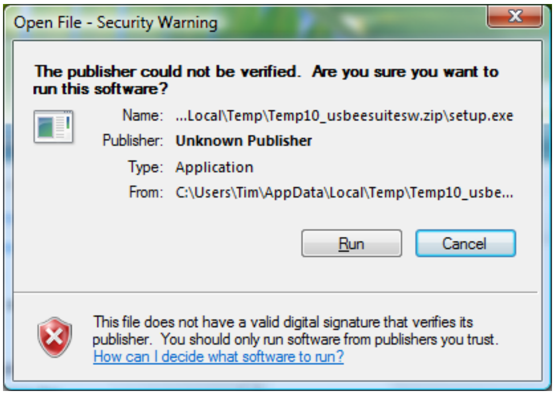
Do not show me the warning for this program again

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



Name	Type
USBeeSuiteSetup.msi	Windows Install
setup.exe	Application

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


Open File - Security Warning

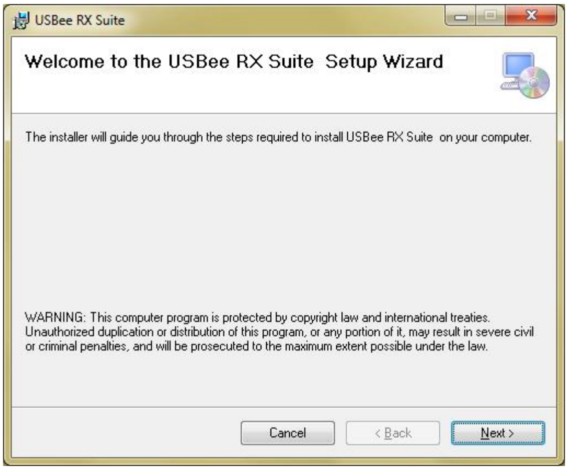
The publisher could not be verified. Are you sure you want to run this software?

Name: ...Local\Temp\Temp10_usbeesuitesw.zip\setup.exe
Publisher: **Unknown Publisher**
Type: Application
From: C:\Users\Tim\AppData\Local\Temp\Temp10_usbe...

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust. How can I decide what software to run?

- The first part of the installation installs Microsoft requirements, including Microsoft .NET Frameworks Version 4.0. 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 RX Suite Setup Wizard" screen.

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



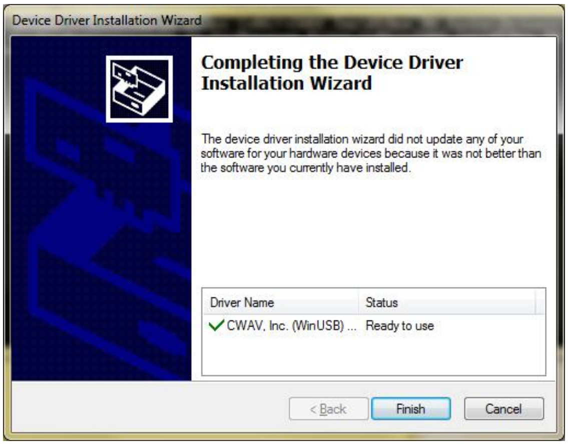
USBee RX Suite

Welcome to the USBee RX Suite Setup Wizard

The installer will guide you through the steps required to install USBee RX Suite on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

- During the installation you will see a driver installation dialog box that will install the USBee RX drivers. Click Finish to complete the installation after the drivers have been successfully installed as below.



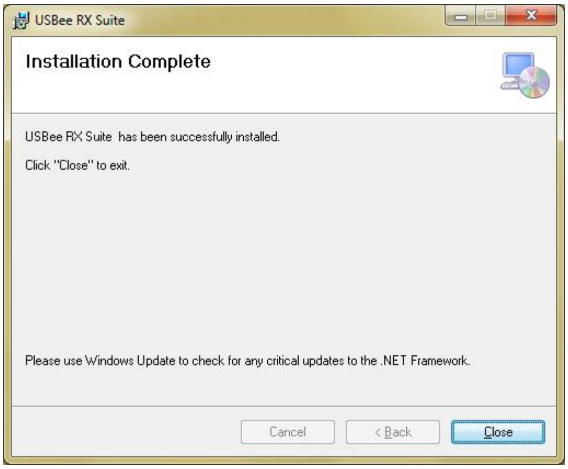
Device Driver Installation Wizard

Completing the Device Driver Installation Wizard

The device driver installation wizard did not update any of your software for your hardware devices because it was not better than the software you currently have installed.

Driver Name	Status
✓ CWAV, Inc. (WinUSB) ...	Ready to use

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



USBee RX Suite

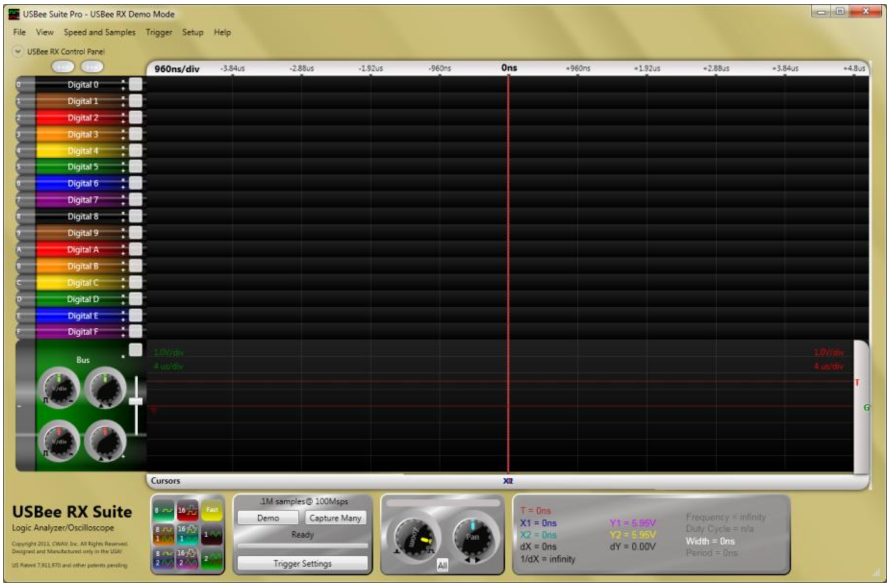
Installation Complete

USBee RX Suite has been successfully installed.

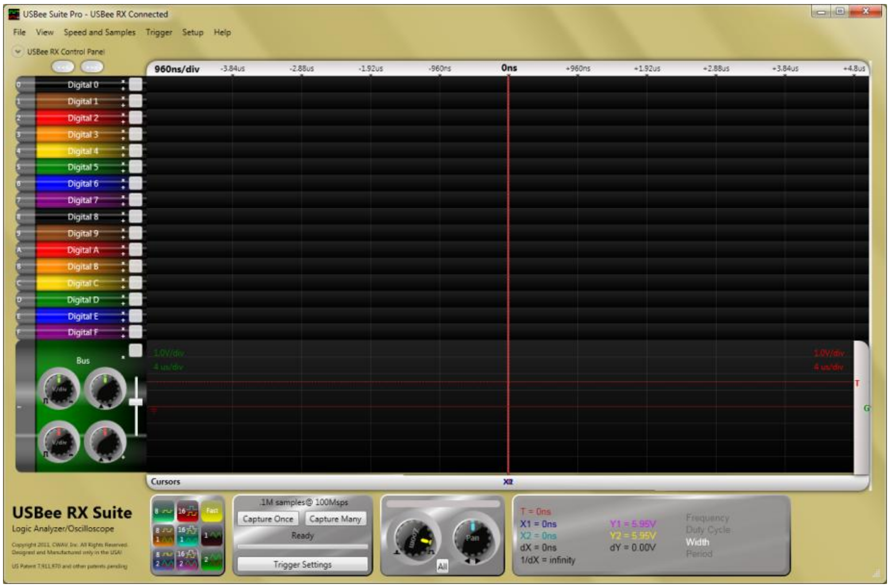
Click "Close" to exit.

Please use Windows Update to check for any critical updates to the .NET Framework.

- Click CLOSE and the USBee RX Suite software is now installed.
- To run the USBee RX Suite software, choose the USBee RX Suite icon from the Windows Start Menu or click on the USBee RX icon on the desktop. If no USBee RX 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 RX plugged in and installed correctly, you will see a screen with all available channels shown. Below is the USBee RX 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.



DEMO MODE

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 RX Suite can decode the bus traffic, manipulate the waveform data, and use the features of the USBee RX Suite.

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

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 RX pod, you must exit the USBee RX Suite, connect the USBee RX and then rerun the USBee RX Suite software.

TESTING YOUR CIRCUIT USING THE USBEE RX

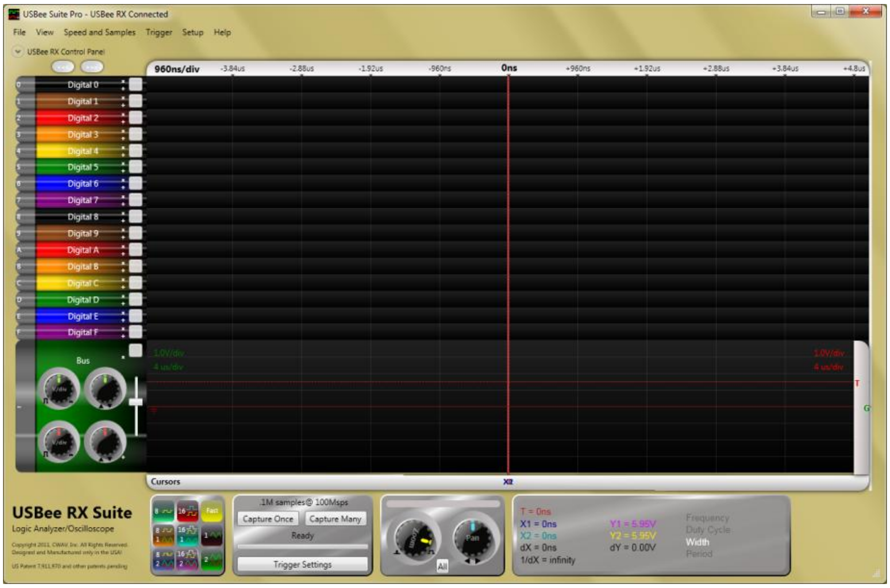
In order to quickly get up and running using The USBee RX Suite application, here is a step by step list of the things you need to do to view a waveform trace, after you have installed the software and hardware.

- **Plug in the USBee RX 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 RX Suite** - Run the USBee RX Suite Application from the Start Menu.
- **Press the Capture Once 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 RX SUITE MIXED SIGNAL OSCILLOSCOPE

This section details the operation of the Mixed Signal Oscilloscope, Logic Analyzer and Protocol Analyzer features of the USBee RX Suite application that runs on the USBee RX Test Pod.

When the USBee RX Suite is first run, you will see a screen containing all of the available input signals for the USBee RX Pod plugged into the PC.



The USBee RX 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.

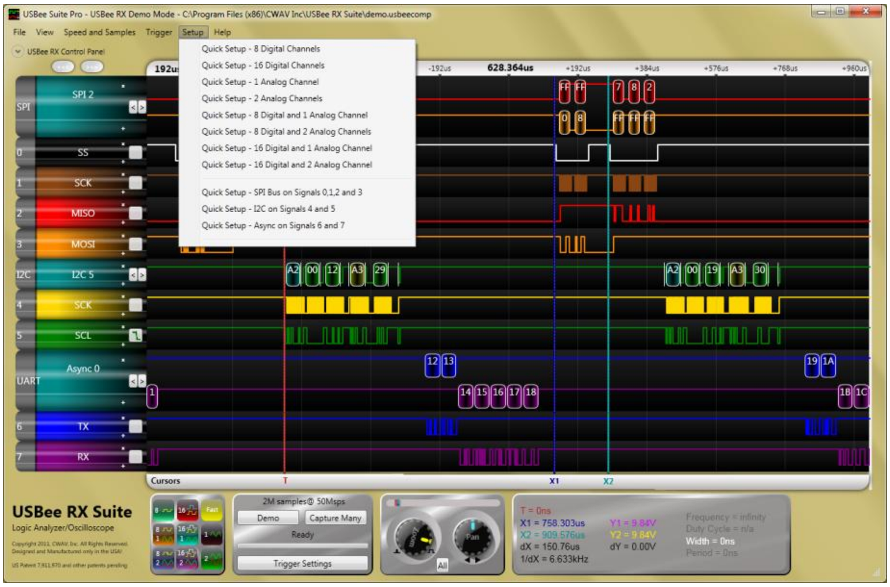
ANALYZER SETUP

QUICK SETUP CONFIGURATION

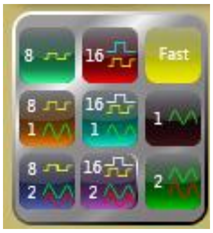
The USBee RX Suite can capture all of the input channels on the USBee RX. With a RX plugged in, it can capture 18 channels of digital (16 data lines, Clock and Trigger signals) and 2 channels of analog at the same time.

Although you can individually show each signal, there are a number of Quick Configurations that let you instantly select just the channels you need.

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 the USBee RX.

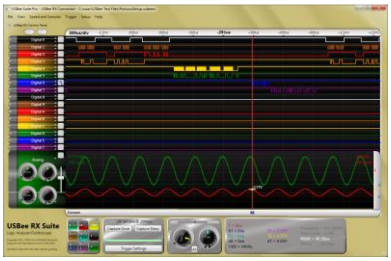

USBee Suite Pro - USBee RX Demo Mode - C:\Program Files (x86)\CWAV Inc\USBee RX Suite\demo.usbeecomp
File
View
Speed and Samples
Trigger
Setup
Help
Quick Setup - 8 Digital Channels
Quick Setup - 16 Digital Channels
Quick Setup - 1 Analog Channel
Quick Setup - 2 Analog Channels
Quick Setup - 8 Digital and 1 Analog Channel
Quick Setup - 8 Digital and 2 Analog Channels
Quick Setup - 16 Digital and 1 Analog Channel
Quick Setup - 16 Digital and 2 Analog Channel
Quick Setup - SPI Bus on Signals 0,1,2 and 3
Quick Setup - I2C on Signals 4 and 5
Quick Setup - Async on Signals 6 and 7
SPI 2
SS
SCK
MISO
MOSI
I2C 5
SCK
SCL
Async 0
UART
TX
RX
Cursors
USBee RX Suite
Logic Analyzer/Oscilloscope
2M samples@ 50Msps
Demo
Capture Many
Ready
Trigger Settings
X1 = 758.303us
X2 = 909.576us
dX = 150.76us
1/dX = 6.633kHz
Y1 = 9.84V
Y2 = 9.84V
dY = 0.00V
Width = 0ns

You can also select these modes using the buttons at the bottom of the screen:

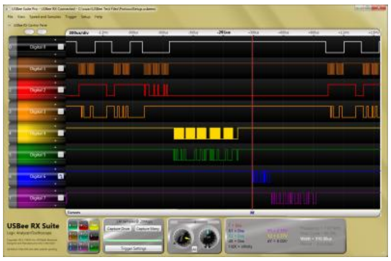


The **FAST** button sets the Sample Rate and buffer size to give the fastest screen update rates.

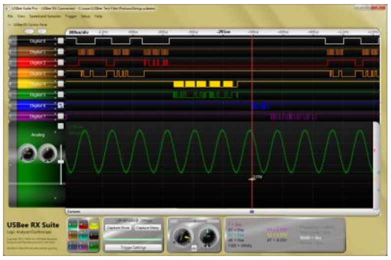
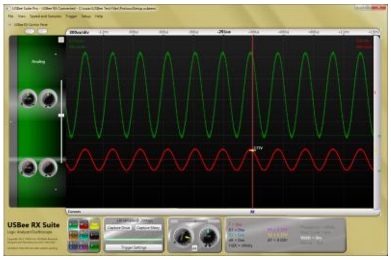
Below are examples of the application in various modes.



16 Digital–2 Analog Channels



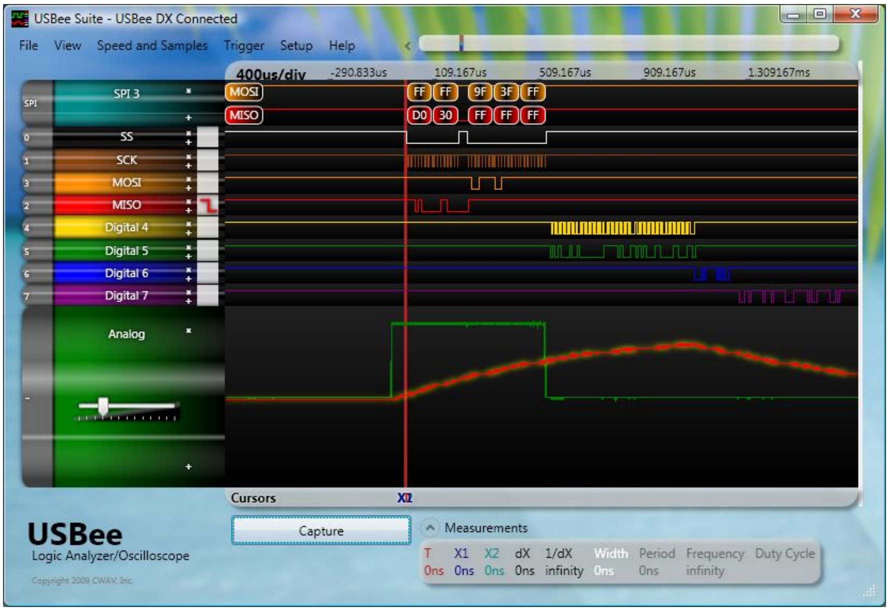
8 Digital–0 Analog Channels

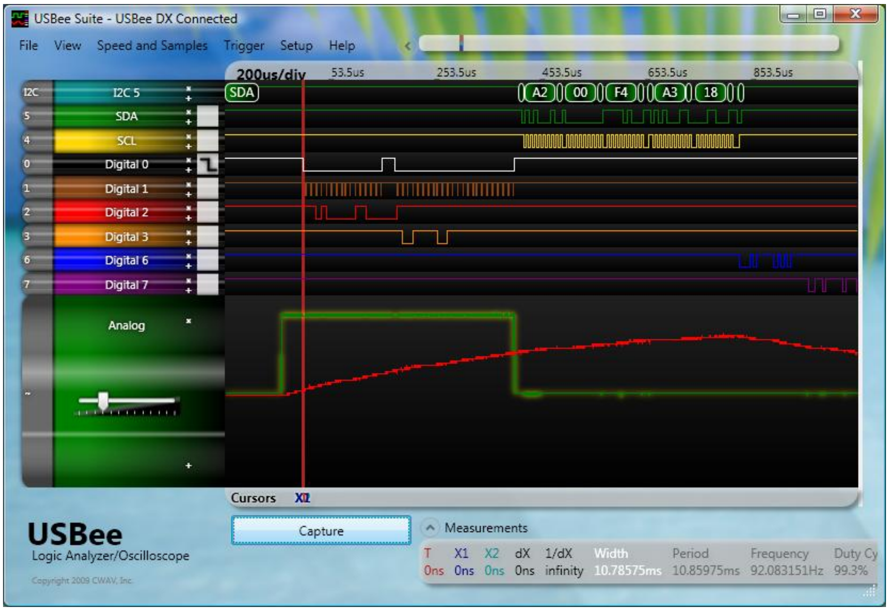
8 Digital–1 Analog Channels 0 Digital–2 Analog Channels

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

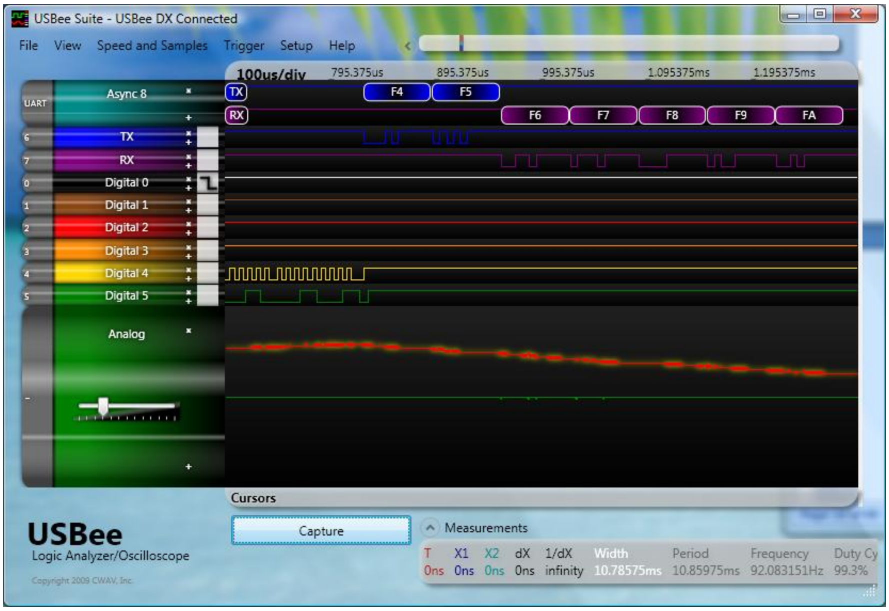
The **Quick Setup – SPI** configures the first 4 lines to be an SPI bus with the SS, SCK, MOSI and MISO lines. It also adds a decoder line to the screen with this data decoded as below.


USBee Suite - USBee DX Connected
File View Speed and Samples Trigger Setup Help
400us/div -290.833us 109.167us 509.167us 909.167us 1.309167ms
SPI 3
MOSI FF FF 9F 3F FF
MISO D0 30 FF FF FF
SS
SCK
MOSI
MISO
Digital 4
Digital 5
Digital 6
Digital 7
Analog
Cursors
Capture
Measurements
T X1 X2 dX 1/dX Width Period Frequency Duty Cycle
0ns 0ns 0ns 0ns infinity 0ns 0ns infinity
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

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.


USBee Suite - USBee DX Connected
File View Speed and Samples Trigger Setup Help
200us/div 53.5us 253.5us 453.5us 653.5us 853.5us
I2C 5
SDA
SCL
Digital 0
Digital 1
Digital 2
Digital 3
Digital 6
Digital 7
Analog
A2 00 F4 A3 18
Cursors X1
Capture
Measurements
T X1 X2 dX 1/dX Width Period Frequency Duty Cy
0ns 0ns 0ns 0ns infinity 10.78575ms 10.85975ms 92.083151Hz 99.3%
USBee Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

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.

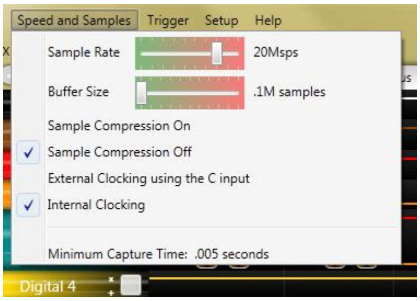

USBee Suite - USBee DX Connected
File View Speed and Samples Trigger Setup Help
100us/div 795.375us 895.375us 995.375us 1.095375ms 1.195375ms
UART Async 8
TX F4 F5
RX F6 F7 F8 F9 FA
TX
RX
Digital 0
Digital 1
Digital 2
Digital 3
Digital 4
Digital 5
Analog
Cursors
Capture
Measurements
T X1 X2 dX 1/dX Width Period Frequency Duty Cy
0ns 0ns 0ns 0ns infinity 10.78575ms 10.85975ms 92.083151Hz 99.3%
USBee Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

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 RX 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 100ksps (samples per second) to up to 100 Msps.

The USBee RX can use sample compression to lengthen the capture time. In **Sample Compression** mode, only the transitions are stored, effectively removing the samples between inactivity. To turn on Sample Compression, select Sample Compression On in the Speed and Samples menu.

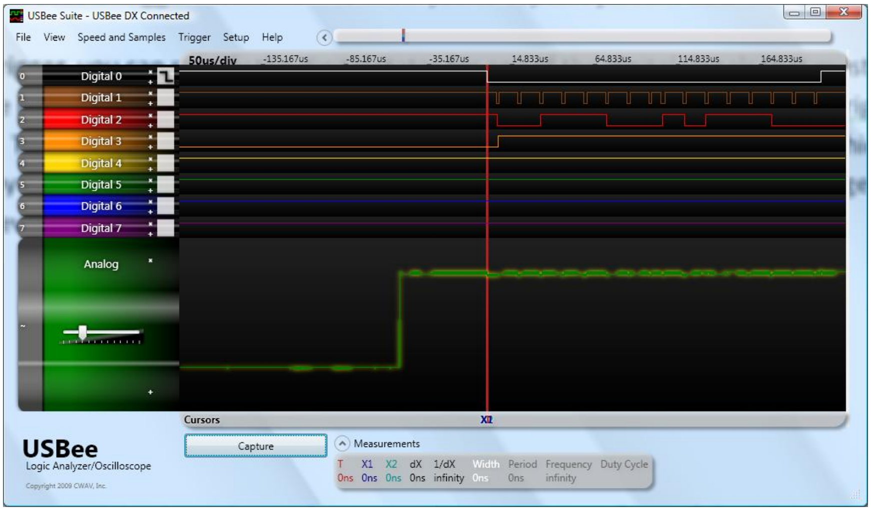
The Sample Compression works on just the channels that are shown on the screen, so for the longest trace buffers, only display the signals that you are interested in. All other channels will be masked out and not captured.

You can also use an external clock (using the C input) to gather each sample instead of the internal sample clock. To turn on **External Clocking**, select External Clocking using the C Input on the Speed and Samples menu. The external clock can run up to 50MHz. When external Clocking is enabled, Sample Compression is not available.

SETTING SIMPLE TRIGGERS

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

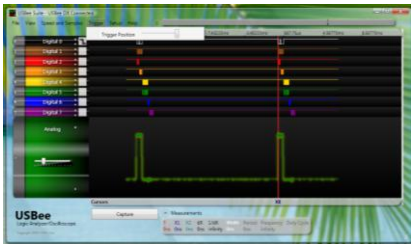
For a **Digital trigger**, you can specify the digital states for any of the digital signals that must be present on the digital lines before it will trigger. Below shows the trigger settings (to the right of the Signal labels). This example shows that we want to trigger on a falling edge of Signal 0, 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.

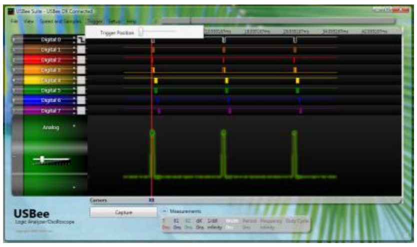


T	X1	X2	dX	1/dX	Width	Period	Frequency	Duty Cycle
0ns	0ns	0ns	0ns	infinity	0ns	0ns	infinity	

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.

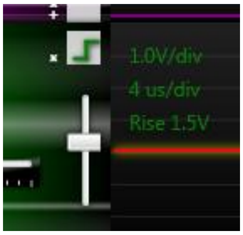
You can use the **Trigger Position** menu setting to specify how much of the data that is in the sample buffer comes before the actual trigger position. If you place the Trigger Position all the way to the left, most of the samples taken will be after the trigger sample. If you place Trigger Position 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.





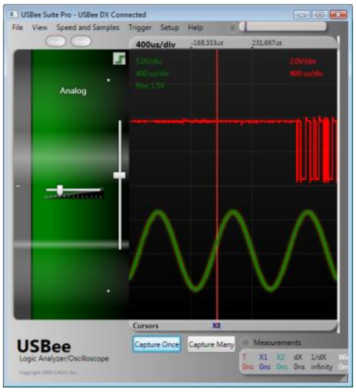
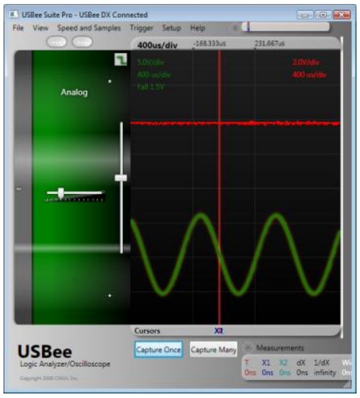
Trigger Position to the Right Trigger Position to the Left

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 (-6V to +6V) 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.

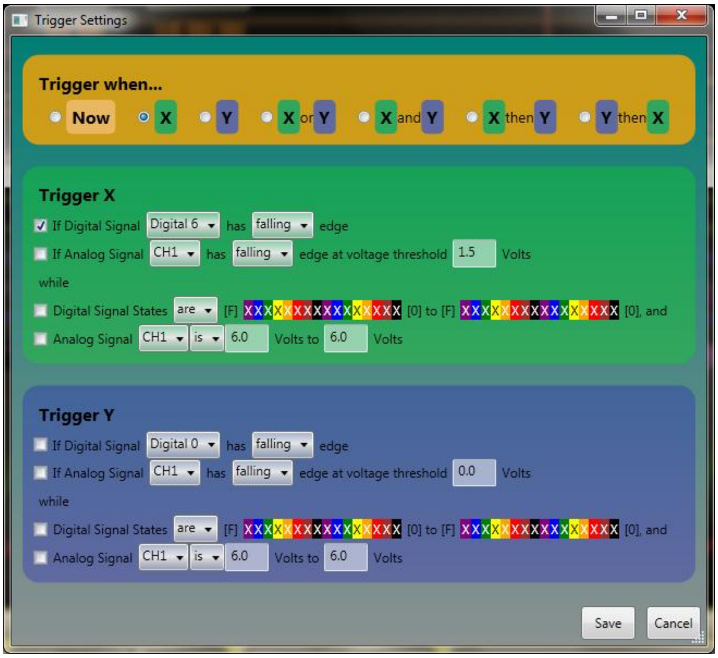

USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
400us/div
Analog
5.0V/div
400 us/div
Rise 1.5V
2.0V/div
400 us/div
Cursors
USBee
Logic Analyzer/Oscilloscope
Capture Once
Capture Many
Measurements

USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
400us/div
Analog
5.0V/div
400 us/div
Fall 1.5V
2.0V/div
400 us/div
Cursors
USBee
Logic Analyzer/Oscilloscope
Capture Once
Capture Many
Measurements

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.

SETTING COMPLEX TRIGGERS

The USBee RX Suite also allows you to set more complex triggers using the Triggers Dialog box as shown below.



Trigger Settings

Trigger when...

Now [✓ X] Y X or Y X and Y X then Y Y then X

Trigger X

[✓] If Digital Signal Digital 6 has falling edge

If Analog Signal CH1 has falling edge at voltage threshold 1.5 Volts

while

Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and

Analog Signal CH1 is 6.0 Volts to 6.0 Volts

Trigger Y

If Digital Signal Digital 0 has falling edge

If Analog Signal CH1 has falling edge at voltage threshold 0.0 Volts

while

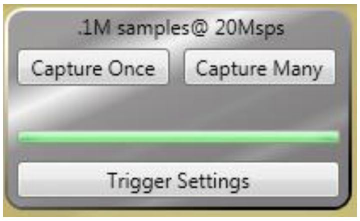
Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and

Analog Signal CH1 is 6.0 Volts to 6.0 Volts

There are two levels of triggers available, X and Y. Each trigger event (X or Y) can have any combination of analog or digital edges which can be qualified using digital patterns or analog voltage levels.

CAPTURING WAVEFORM DATA

Press the Capture buttons to start capturing the waveform data from your hardware design. If you are running in Demo mode, the button reads Demo. Capture Once performs a single capture, while Capture Many repeatedly captures and displays consecutive traces.

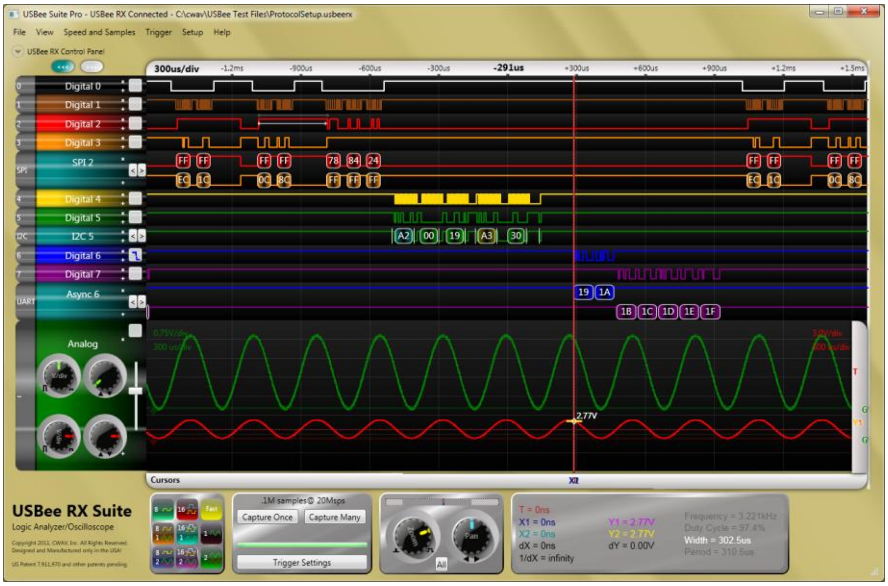


Each capture will look for the trigger condition, fill the buffer with samples of the signals and stop. If you would like to stop the capture before it is completed just press the same button again (which reads STOP during a capture). After a trace is captured, the waveform data is gathered, decoded (if needed) and displayed in the waveform window.

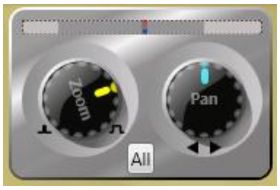
VIEWING CAPTURED DATA

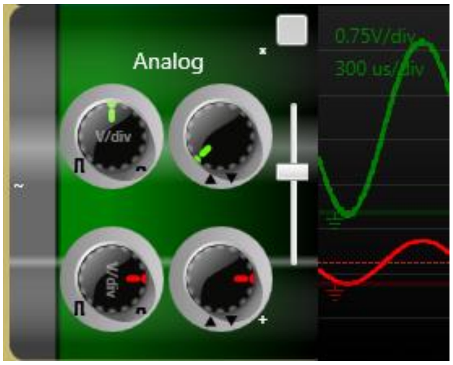
SCROLLING, ZOOMING AND PANNING WAVEFORMS

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.





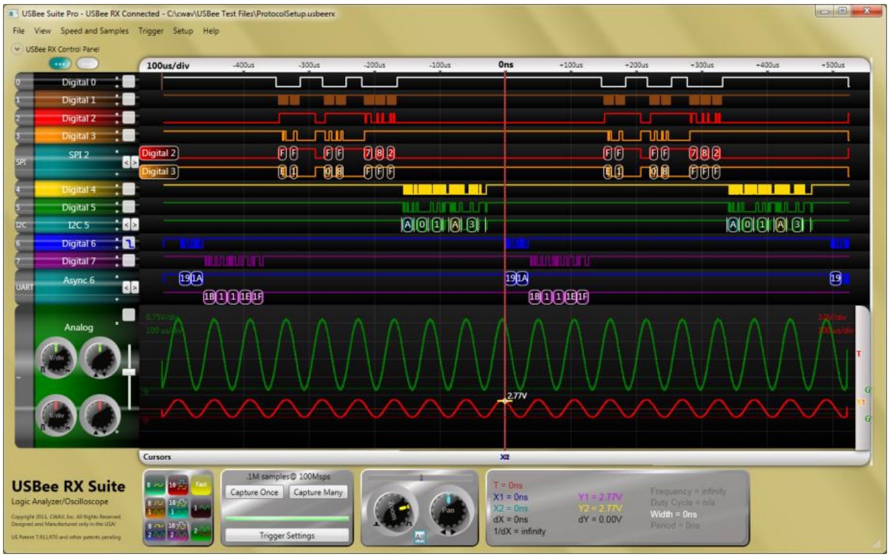
To **Scroll the Waveforms in Time** left and right, you can click on the overview bar above the knobs, scroll the **Pan knob** (click and drag or mouse wheel on the knob) or you can simply click and drag the waveform itself with the left mouse button.

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 or use the **analog offset knob** to the left of the analog waveforms.

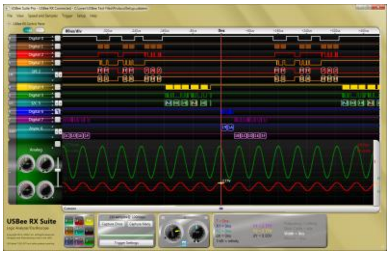
To **Zoom In** or **Zoom Out**, or other words change the number of **Seconds per Division,** you can use the scroll wheel or single click on the waveform. You can also use the **Zoom knob** (click and drag or mouse wheel on the knob). To zoom in, scroll up or click the left mouse on the waveform window. To zoom out in time, scroll down or click the right mouse button on the waveform window.

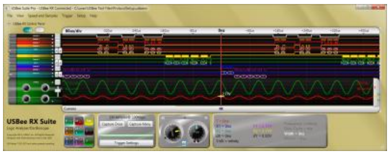
To change the number of **Volts per Division** for an analog channel, use the **V/div knob** (click and drag or mouse wheel on the knob)highlight the channel you want to change, or you can hold the left mouse button down and use the scroll wheel. You can also highlight the signal and use the slider bar to the left of the waveform.

To view the entire capture buffer on the display, press the **All** button. The screen below shows all of the collected samples on a single screen for that trace.



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.

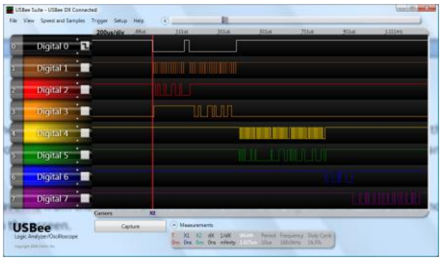
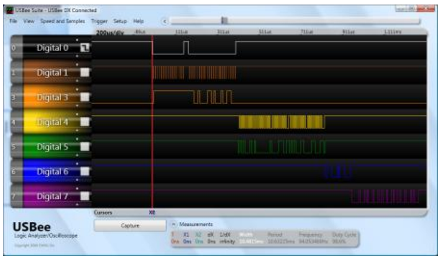




MODIFYING WAVE LINES

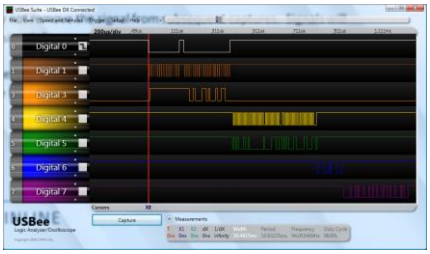
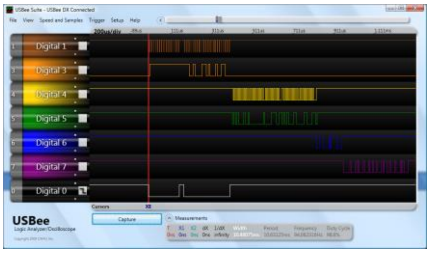
Each line on the display is called a **Waveline**. Wavelines can be modified to your liking so that you see the data you need to solve your problem. You can delete, add, move, or reconfigure any waveline.

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 RX Suite after deleting the Digital 2 waveline.

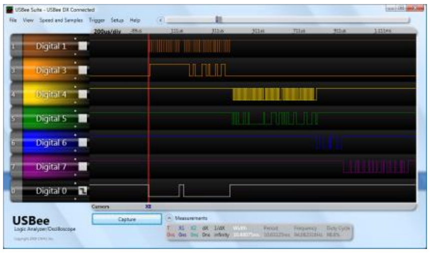
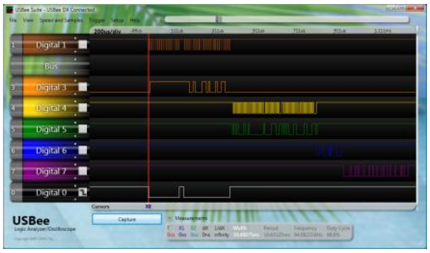



Removing a signal from the screen may not remove the signal from subsequent captures. Signals will only be eliminated from captures if all signals from a given byte lane are removed.

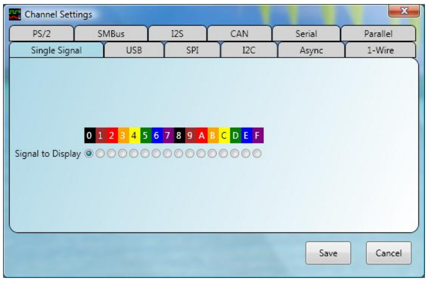
To **Move a Waveline**, simply click on the gray tab on the left and drag it to the new position. Below shows the Digital 0 signal moved to the bottom.

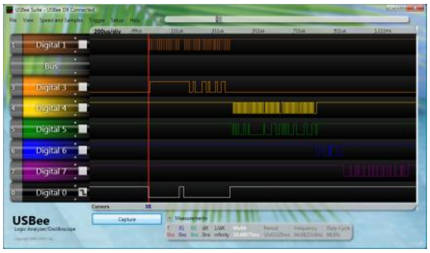
To Add a Waveline, click on the small + sign above where you want to insert the new waveline. Below you see a new waveline inserted after the first waveline.

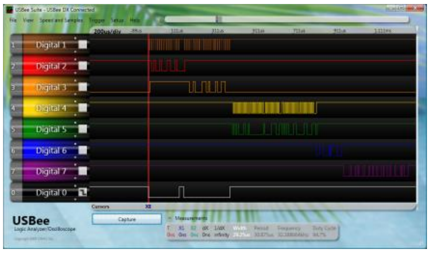



When you insert a new waveline the Channel Selection dialog box appears for you to choose the settings for that waveline. Below shows the Channel Settings Dialog Box.



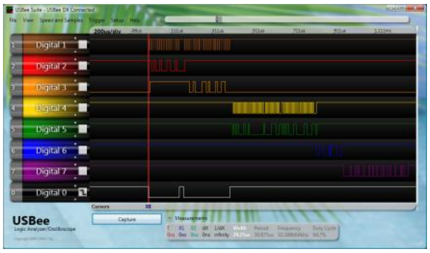
Once you select the properties of the new waveline, it will be displayed with the other signals. Below shows a new line added that shows the Digital 2 single signal.

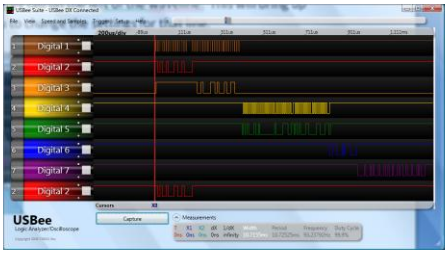




We will go through creating Bus wavelines that decode bus traffic in-line in the next section.

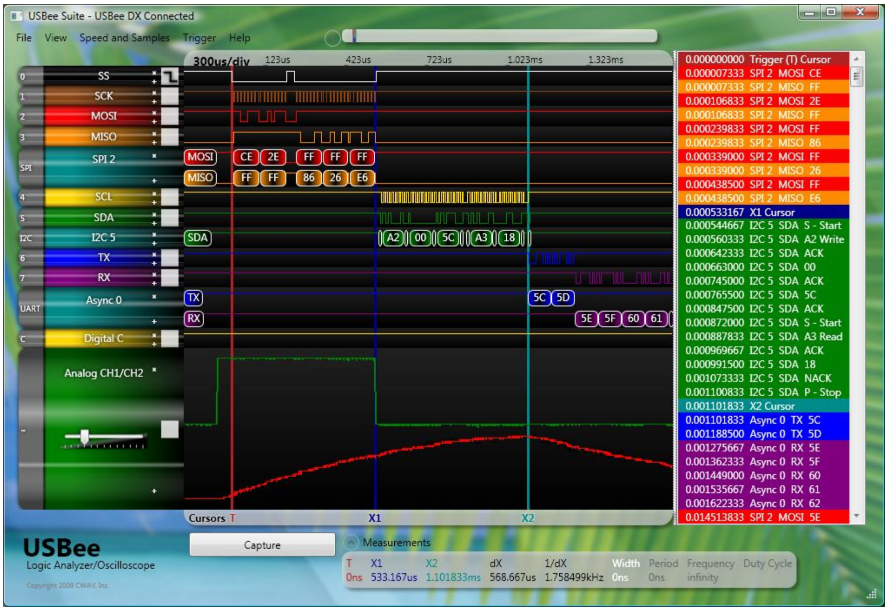
To **Modify an Existing Waveline** click on the grey tab on the left of the waveline. This will bring up the Channel Settings dialog box and allow you to change the settings for that line. Below we modified the last line to show Digital 2 signal instead of the Digital 0 signal.





DECODING BUS TRAFFIC INLINE

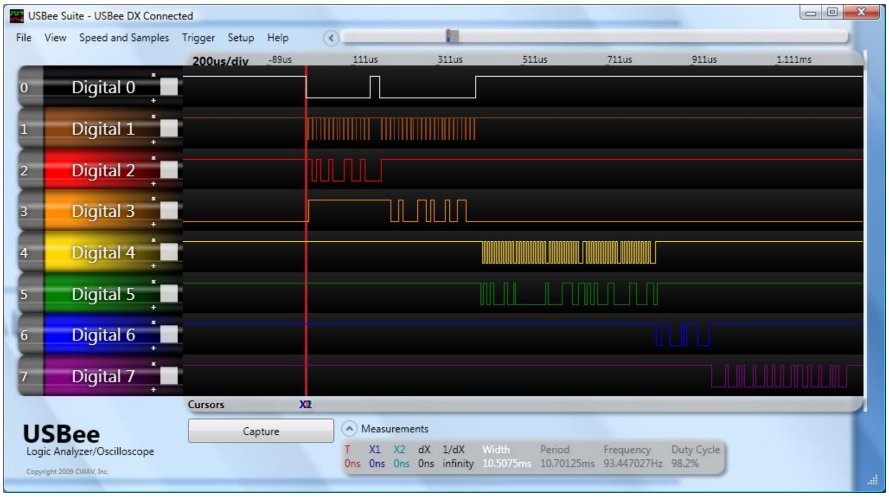
The USBee RX Suite software can decode certain types of serial busses automatically and display that information in-line with the waveforms. Below is an example of a screen that shows 3 different serial bus decoders at the same time, one SPI, one I2C, and one full duplex ASYNC channel.


USBee Suite - USBee DX Connected
File View Speed and Samples Trigger Help
300us/div 123us 423us 723us 1.023ms 1.323ms
SS SCK MOSI MISO SPI 2 SCL SDA I2C 5 TX RX Async 0 Digital C Analog CH1/CH2
MOSI CE 2E FF FF FF
MISO FF FF 86 26 E6
SDA A2 00 5C A3 18
TX 5C 5D
RX 5E 5F 60 61
Cursors T X1 X2
Capture
Measurements
USBee Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

```
0.000000000  Trigger (T) Cursor
0.000007333  SPI 2  MOSI  CE
0.000007333  SPI 2  MISO  FF
0.000106833  SPI 2  MOSI  2E
0.000106833  SPI 2  MISO  FF
0.000239833  SPI 2  MOSI  FF
0.000239833  SPI 2  MISO  86
0.000339000  SPI 2  MOSI  FF
0.000339000  SPI 2  MISO  26
0.000438500  SPI 2  MOSI  FF
0.000438500  SPI 2  MISO  E6
0.000533167  X1 Cursor
0.000544667  I2C 5  SDA  S - Start
0.000560333  I2C 5  SDA  A2 Write
0.000642333  I2C 5  SDA  ACK
0.000663000  I2C 5  SDA  00
0.000745000  I2C 5  SDA  ACK
0.000765500  I2C 5  SDA  5C
0.000847500  I2C 5  SDA  ACK
0.000872000  I2C 5  SDA  S - Start
0.000887833  I2C 5  SDA  A3 Read
0.000969667  I2C 5  SDA  ACK
0.000991500  I2C 5  SDA  18
0.001073333  I2C 5  SDA  NACK
0.001100833  I2C 5  SDA  P - Stop
0.001101833  X2 Cursor
0.001101833  Async 0  TX  5C
0.001188500  Async 0  TX  5D
0.001275667  Async 0  RX  5E
0.001362333  Async 0  RX  5F
0.001449000  Async 0  RX  60
0.001535667  Async 0  RX  61
0.001622333  Async 0  RX  62
0.014513833  SPI 2  MOSI  5E
```

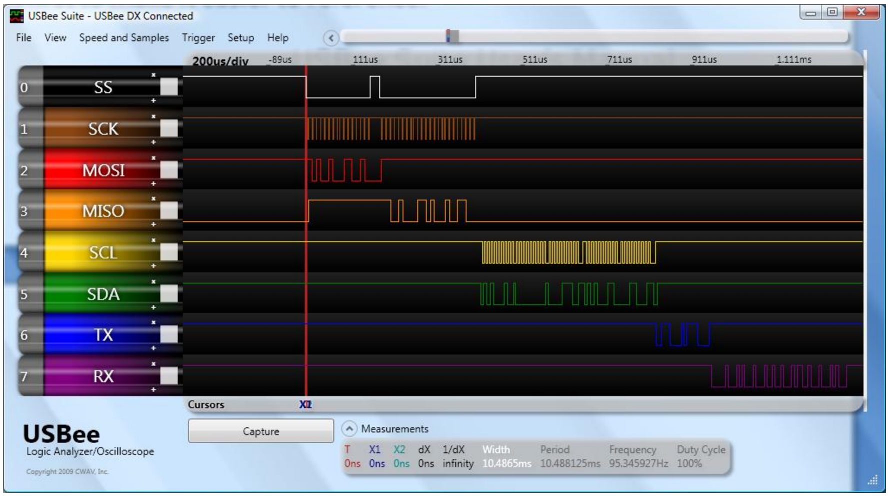
```
T    X1         X2          dX         1/dX         Width  Period  Frequency  Duty Cycle
0ns  533.167us  1.101833ms  568.667us  1.758499kHz  0ns    0ns     infinity
```

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.


USBee Suite - USBee DX Connected
File View Speed and Samples Trigger Setup Help
200us/div -89us 111us 311us 511us 711us 911us 1.111ms
Digital 0 Digital 1 Digital 2 Digital 3 Digital 4 Digital 5 Digital 6 Digital 7
Cursors X2
Capture
Measurements
USBee Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

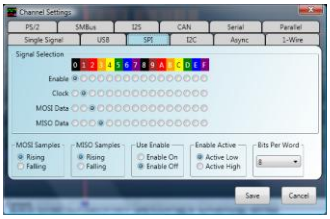
```
T    X1   X2   dX   1/dX      Width      Period      Frequency     Duty Cycle
0ns  0ns  0ns  0ns  infinity  10.5075ms  10.70125ms  93.447027Hz   98.2%
```

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

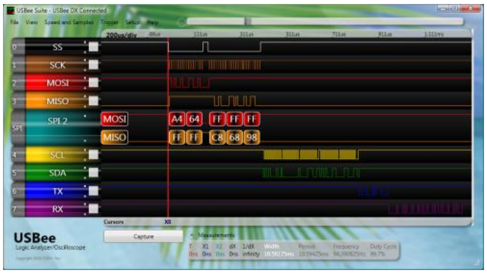

USBee Suite - USBee DX Connected
File View Speed and Samples Trigger Setup Help
200us/div -89us 111us 311us 511us 711us 911us 1.111ms
0 SS
1 SCK
2 MOSI
3 MISO
4 SCL
5 SDA
6 TX
7 RX
Cursors X12
Capture
Measurements
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

T X1 X2 dX 1/dX Width Period Frequency Duty Cycle
0ns 0ns 0ns 0ns infinity 10.4865ms 10.488125ms 95.345927Hz 100%

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.

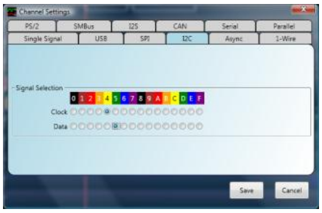

Channel Settings
PS/2 SMBus I2S CAN Serial Parallel
Single Signal USB SPI I2C Async 1-Wire
Signal Selection
0 1 2 3 4 5 6 7 8 9 A B C D E F
Save
Cancel

Enable: 0
Clock: 1
MOSI Data: 2
MISO Data: 3
MOSI Samples: [✓ Rising] Falling
MISO Samples: [✓ Rising] Falling
Use Enable: Enable On [✓ Enable Off]
Enable Active: [✓ Active Low] Active High
Bits Per Word: 8

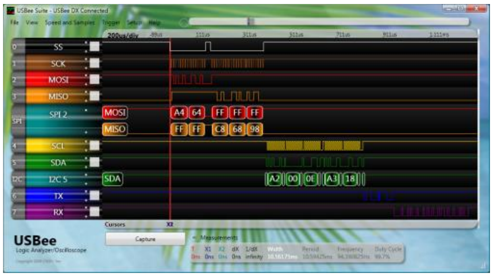


From here we select the parameters for this bus (shown above) and press Save. Once we press Save, the line is added to the screen, the current trace is decoded, and the decoded information is shown on the waveline.

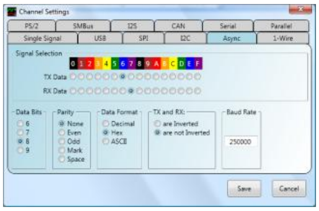
We then add the I2C bus using the Channel Settings dialog box as below with the resulting waveline.


Channel Settings
PS/2 SMBus I2S CAN Serial Parallel
Single Signal USB SPI I2C Async 1-Wire
Signal Selection
0 1 2 3 4 5 6 7 8 9 A B C D E F
Save
Cancel

Clock: 4
Data: 5



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



Channel Settings

Tabs: PS/2, SMBus, I2S, CAN, Serial, Parallel, Single Signal, USB, SPI, I2C, [✓ Async], 1-Wire

Signal Selection: TX Data [✓ 6]; RX Data [✓ 7]

Data Bits: 6, 7, [✓ 8], 9

Parity: [✓ None], Even, Odd, Mark, Space

Data Format: Decimal, [✓ Hex], ASCII

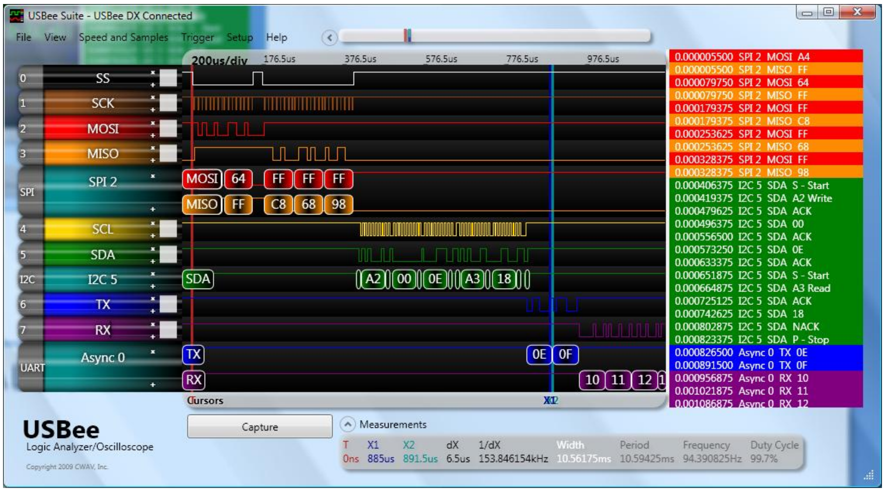
TX and RX: are Inverted, [✓ are not Inverted]

Baud Rate: 250000

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.



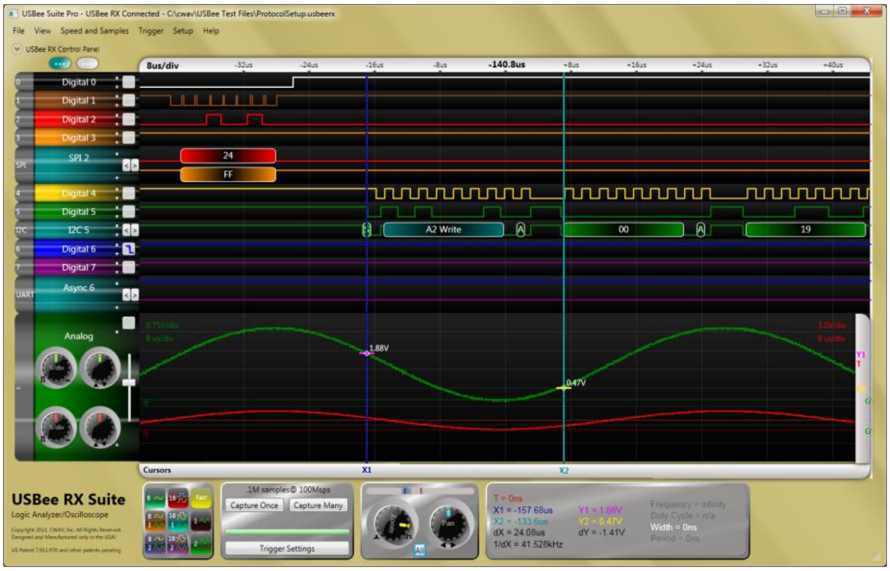
```
0.000005500  SPI 2  MOSI  A4
0.000005500  SPI 2  MISO  FF
0.000079750  SPI 2  MOSI  64
0.000079750  SPI 2  MISO  FF
0.000179375  SPI 2  MOSI  FF
0.000179375  SPI 2  MISO  C8
0.000253625  SPI 2  MOSI  FF
0.000253625  SPI 2  MISO  68
0.000328375  SPI 2  MOSI  FF
0.000328375  SPI 2  MISO  98
0.000406375  I2C 5  SDA  S - Start
0.000419375  I2C 5  SDA  A2 Write
0.000479625  I2C 5  SDA  ACK
0.000496375  I2C 5  SDA  00
0.000556500  I2C 5  SDA  ACK
0.000573250  I2C 5  SDA  0E
0.000633375  I2C 5  SDA  ACK
0.000651875  I2C 5  SDA  S - Start
0.000664875  I2C 5  SDA  A3 Read
0.000725125  I2C 5  SDA  ACK
0.000742625  I2C 5  SDA  18
0.000802875  I2C 5  SDA  NACK
0.000823375  I2C 5  SDA  P - Stop
0.000826500  Async 0  TX  0E
0.000891500  Async 0  TX  0F
0.000956875  Async 0  RX  10
0.001021875  Async 0  RX  11
0.001086875  Async 0  RX  12
```

T	X1	X2	dX	1/dX	Width	Period	Frequency	Duty Cycle
0ns	885us	891.5us	6.5us	153.846154kHz	10.56175ms	10.59425ms	94.390825Hz	99.7%

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 RX Suite uses cursors to help in these measurements.



T = 0ns
X1 = -157.68us
X2 = -133.6us
dX = 24.08us
1/dX = 41.528kHz
Y1 = 1.88V
Y2 = 0.47V
dY = -1.41V
Frequency = infinity
Duty Cycle = n/a
Width = 0ns
Period = 0ns

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 and Duty Cycle

BUS DECODING OPTIONS

The USBee RX 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

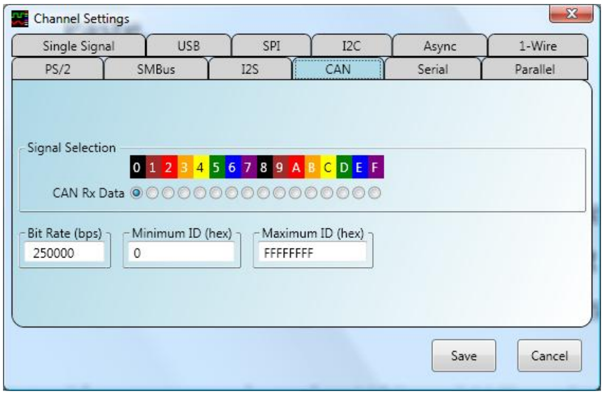
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



Signal Selection: CAN Rx Data = 0

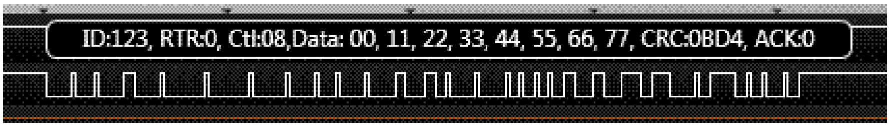
Bit Rate (bps): 250000

Minimum ID (hex): 0

Maximum ID (hex): FFFFFFFF

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.



ID:123, RTR:0, Ctl:08,Data: 00, 11, 22, 33, 44, 55, 66, 77, CRC:0BD4, ACK:0

USB BUS SETUP

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

Hardware Setup

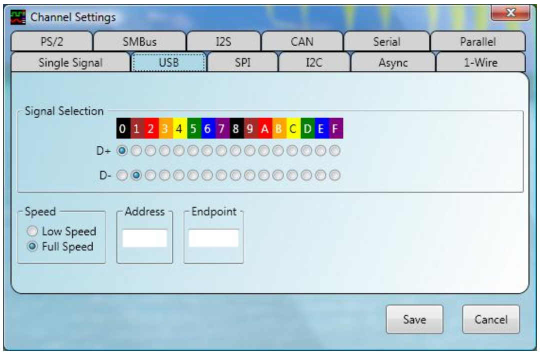
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

Please note that the USBee RX 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.



Signal Selection: D+ = 0, D- = 1

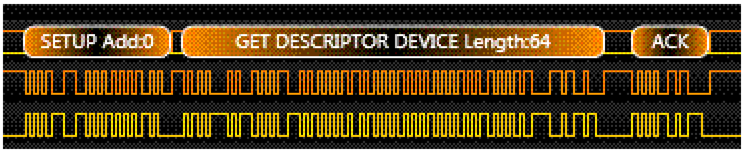
Speed: Low Speed [✓ Full Speed]

Address:

Endpoint:

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.



SETUP Add:0 | GET DESCRIPTOR DEVICE Length:64 | ACK

I2C BUS SETUP

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

Hardware Setup

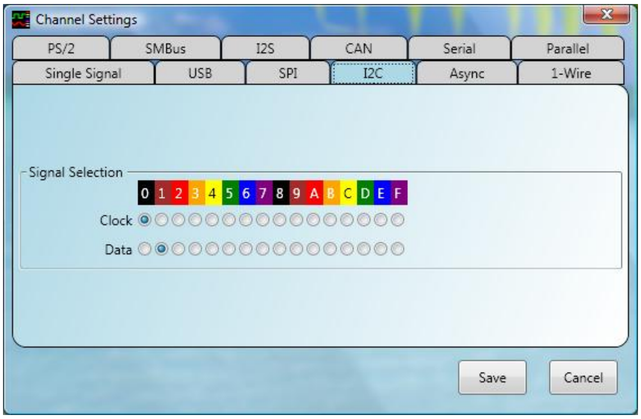
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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

Software Setup

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

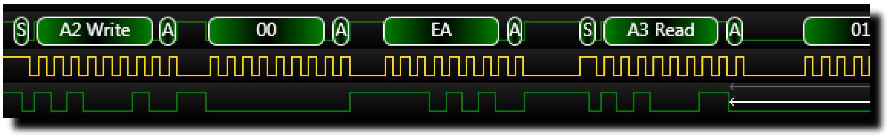


Signal Selection

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

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

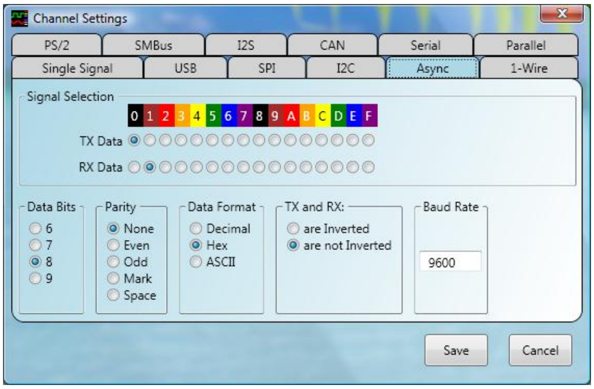
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



TX Data: [✓ 0]
RX Data: [✓ 1]
Data Bits: 6 7 [✓ 8] 9
Parity: [✓ None] Even Odd Mark Space
Data Format: Decimal [✓ Hex] ASCII
TX and RX: are Inverted [✓ are not Inverted]
Baud Rate: 9600

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.

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

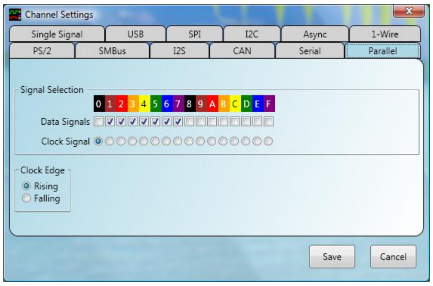
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.


Channel Settings

Single Signal | USB | SPI | I2C | Async | 1-Wire

PS/2 | SMBus | I2S | CAN | Serial | Parallel

Signal Selection

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Data Signals	☐	✓	✓	✓	✓	✓	✓	✓	☐	☐	☐	☐	☐	☐	☐	☐
Clock Signal	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

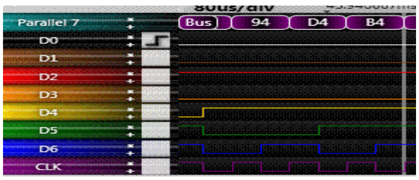
Clock Edge

[✓ Rising] Falling

Save Cancel

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

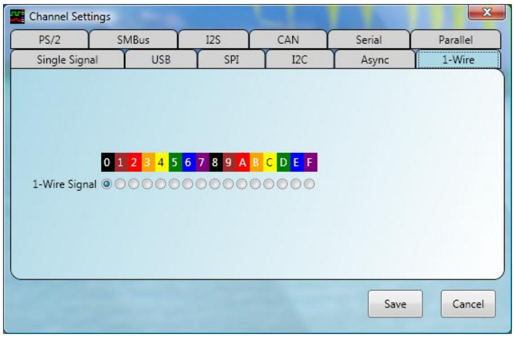
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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

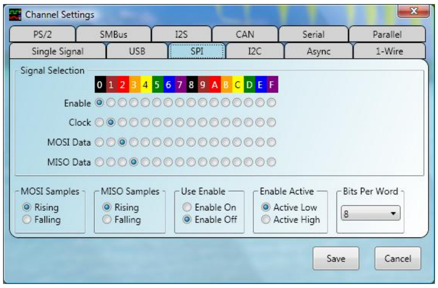
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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

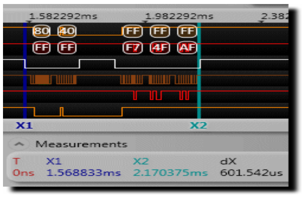
Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



Enable: [✓ 0]
Clock: [✓ 1]
MOSI Data: [✓ 2]
MISO Data: [✓ 3]
MOSI Samples: [✓ Rising] Falling
MISO Samples: [✓ Rising] Falling
Use Enable: Enable On [✓ Enable Off]
Enable Active: [✓ Active Low] Active High
Bits Per Word: 8

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.



```
T     X1          X2          dX
0ns   1.568833ms  2.170375ms  601.542us
```

SM BUS BUS SETUP

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

Hardware Setup

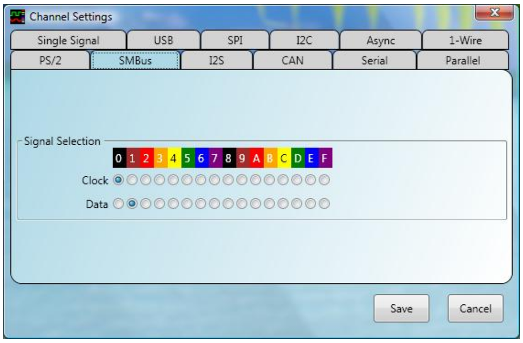
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.

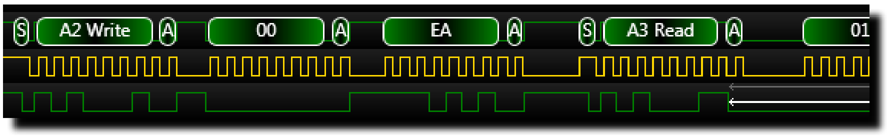
Software Setup

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

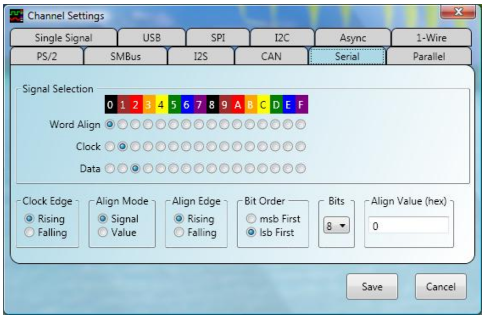
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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.



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

Word Align [✓ 0]

Clock [✓ 1]

Data [✓ 2]

Clock Edge: [✓ Rising] Falling

Align Mode: [✓ Signal] Value

Align Edge: [✓ Rising] Falling

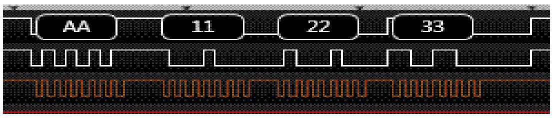
Bit Order: msb First [✓ lsb First]

Bits: 8

Align Value (hex): 0

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

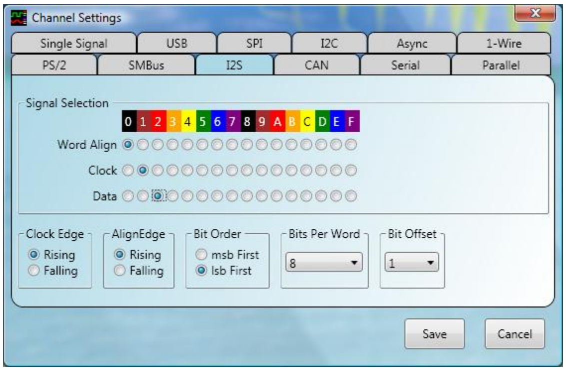
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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

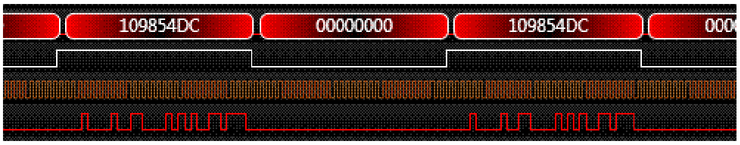
Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



Word Align: 0
Clock: 1
Data: 2
Clock Edge: Rising
AlignEdge: Rising
Bit Order: lsb First
Bits Per Word: 8
Bit Offset: 1

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

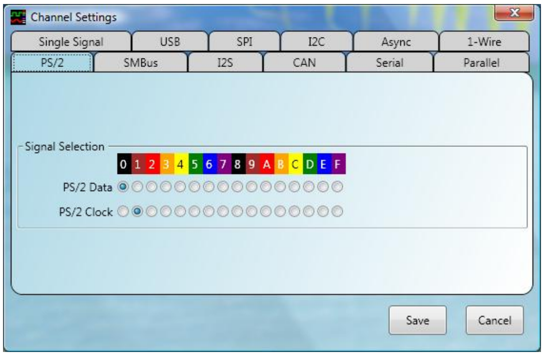
To use the Decoder you need to connect the USBee Test Pod to your hardware using the test leads. You can either connect the test leads directly to pin headers on your board, or use the test clips for attaching to your components.

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

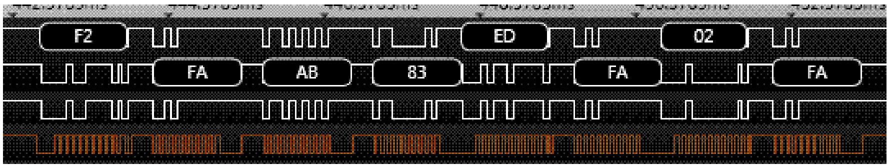
Activate the below Channel Settings Dialog by clicking the grey tab on the left of the signal names on the main application screen.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
PS/2 Data	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
PS/2 Clock	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○

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.

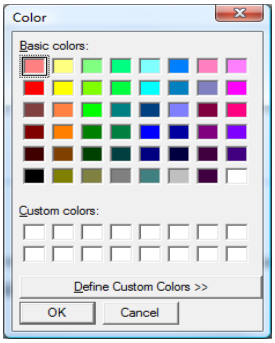


SETTING VIEWING PREFERENCES

The USBee RX 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.

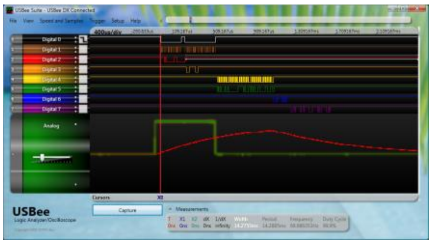


BACKGROUND COLOR

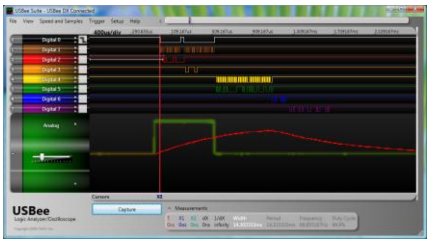
The background of the waveform screen can be set to white or black using the View | Background White or View | Background Black menu items.

GLASS APPEARANCE

Windows 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

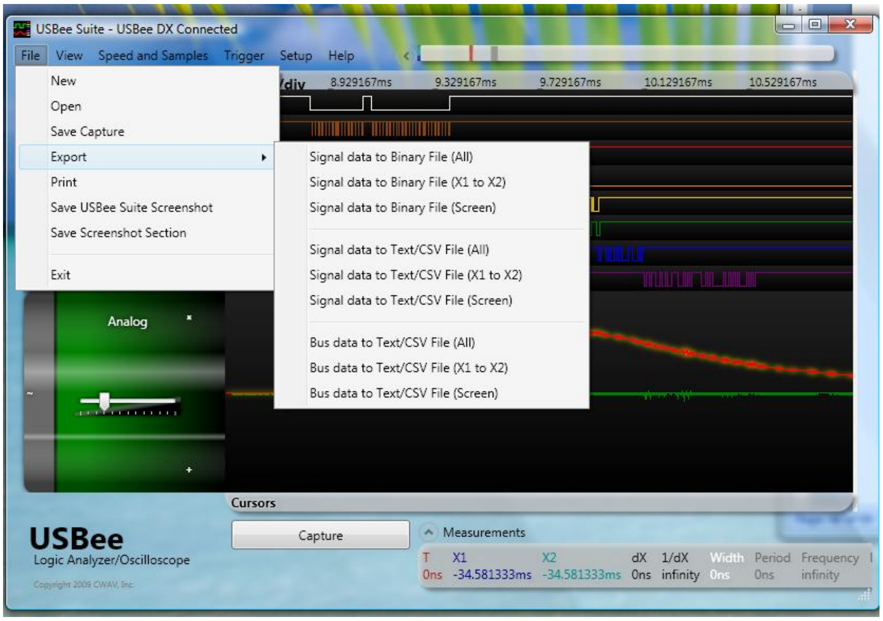
To view a previously saved capture file, us the File/Open menu item. This will load the trace data and screen format including decoder setup. Opening files with large buffer sizes can take a while to decompress and display.

RECENTLY USED FILE LIST

The USBee RX 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.


USBee Suite - USBee DX Connected
File
View
Speed and Samples
Trigger
Setup
Help
New
Open
Save Capture
Export
Print
Save USBee Suite Screenshot
Save Screenshot Section
Exit
Signal data to Binary File (All)
Signal data to Binary File (X1 to X2)
Signal data to Binary File (Screen)
Signal data to Text/CSV File (All)
Signal data to Text/CSV File (X1 to X2)
Signal data to Text/CSV File (Screen)
Bus data to Text/CSV File (All)
Bus data to Text/CSV File (X1 to X2)
Bus data to Text/CSV File (Screen)
8.929167ms
9.329167ms
9.729167ms
10.129167ms
10.529167ms
Analog
Cursors
Capture
Measurements
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.
T 0ns
X1 -34.581333ms
X2 -34.581333ms
dX 0ns
1/dX infinity
Width 0ns
Period 0ns
Frequency infinity

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 (0=-6V, 128 = 0V, 255 = +6V)
4	Channel 2 Analog voltage (0=-6V, 128 = 0V, 255 = +6V)

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
```

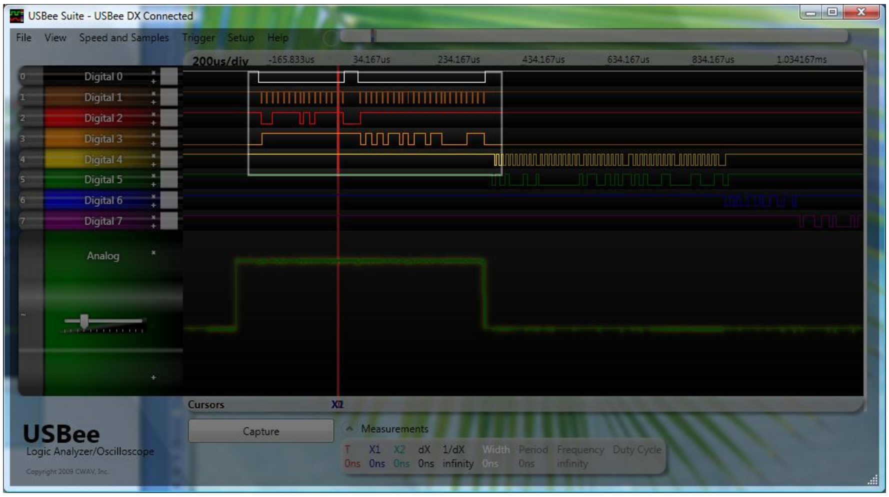
PRINTING

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

CREATING SCREEN SHOTS

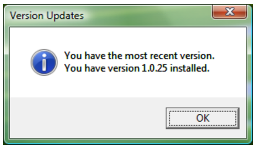
An easy way to create documentation is to take screen shots, or portions of the screen image, and save them to graphics files. You can save the entire USBee RX Suite application window to a file using the File/Save USBee RX Suite Screenshot menu item. This lets you save the image as a BMP, JPG, PNG, 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 RX Suite software are posted on the USBee.com web site. To have the USBee RX 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.



You have the most recent version.
You have version 1.0.25 installed.

DEVELOPING YOUR OWN CUSTOM DECODERS

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

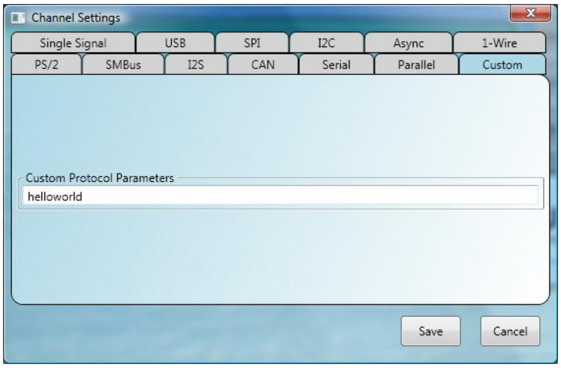
To implement a custom decoder you must create a Class Library (DLL) using the code below as an example. You can build this Class Library using the free Microsoft Visual Studio 2008 Express or newer. Our example is in Visual Basic, but can easily be ported to C or other language supported in Visual Studio.

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

USING THE CUSTOM DECODER

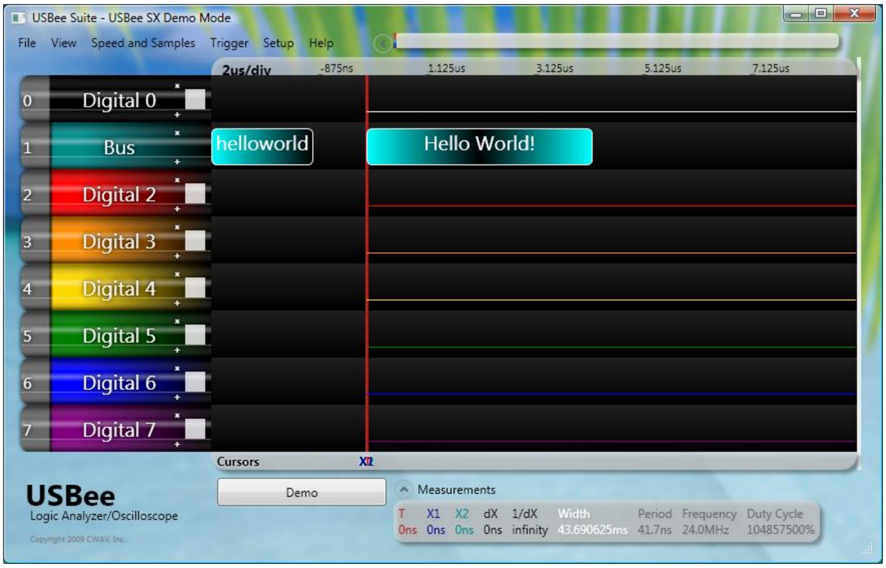
Using the Custom Decoder that you build is simple. To select to use your Custom Decoder you select the Custom tab in the Channel Settings dialog box. You can then enter a set of parameters that are sent to your decoder. These parameters can specify anything you may need to determine how to decode your 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.



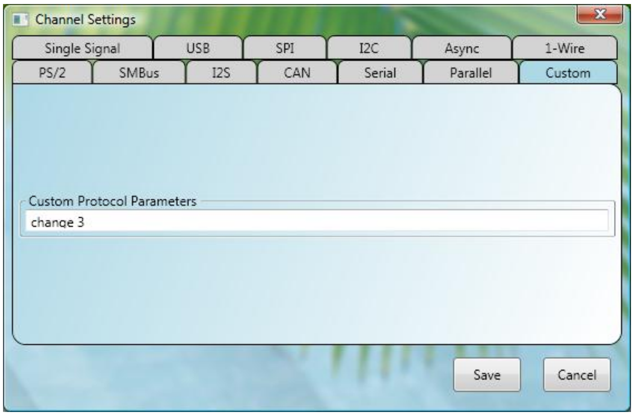
Custom Protocol Parameters: helloworld

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.


USBee Suite - USBee SX Demo Mode
File View Speed and Samples Trigger Setup Help
2us/div -875ns 1.125us 3.125us 5.125us 7.125us
0 Digital 0
1 Bus
helloworld
Hello World!
2 Digital 2
3 Digital 3
4 Digital 4
5 Digital 5
6 Digital 6
7 Digital 7
Cursors
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.
Demo
Measurements

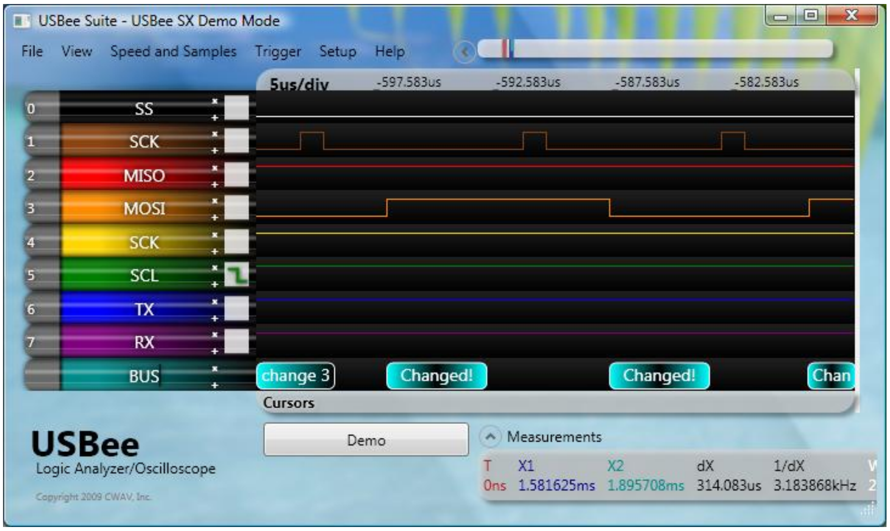
T	X1	X2	dX	1/dX	Width	Period	Frequency	Duty Cycle
0ns	0ns	0ns	0ns	infinity	43.690625ms	41.7ns	24.0MHz	104857500%

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.



Custom Protocol Parameters: change 3

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.



```
T     X1          X2          dX          1/dX
0ns   1.581625ms  1.895708ms  314.083us   3.183868kHz
```

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 RX Suite software in the \Program Files\CWAV Inc\USBee RX Suite\CustomUSBeeSuiteDecoderRX\ directory. You can build this Class Library using the free Microsoft Visual Studio 2010 Express. Our example is in Visual Basic, but can easily be ported to C or other language supported in VS2010.

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 CustomUSBeeSuiteDecoderRX.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 RX 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 a few different protocol decodes and displays the results on the waveline.   A version that includes an actual NEC IR decoder is installed with the USBee RX Suite. Use this example to start your own.

```
Option Explicit On
Option Strict On
Imports System.IO


Public Class CustomUSBeeSuiteDecoderRX

    Declare Function SampleData Lib "usbeerxste.dll" Alias "?LoggedData@@YGJ_J@Z" (ByVal Index As Int64) As Integer
    Declare Function FindNextEdge Lib "usbeerxste.dll" Alias "?FindNextEdge@@YG_J_JKK@Z" (ByVal start As Int64, ByVal Mask As Integer,
ByVal direction As Integer) As Int64

    ' The SampleData routine returns a 4 byte value that contains a single sample of all the signals
    ' The format of the 32 bits returned is as follows:
    '
    ' MSB                          LSB
    ' XXXXXXXXYYYYYYYYFEDCBA9876543210
    '
    ' where XXXXXXXX is Channel 2 Analog value (0=-6V, 255 = +6V)
    '       YYYYYYYY is Channel 1 Analog value (0=-6V, 255 = +6V)
    '       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
    '
    ' FindNextEdge routine scans the sample buffer starting at the start sample number to find the next sample
    ' with an edge on any masked channels.  Mask bits aligned as above with 1 considered, 0 being ignored.
    ' direction 1 = forward, 0 = backward.
    ' FindNextEdge should be used instead of SampleData whenever possible since it is much faster.
    ' Return value is the sample number at the next edge or -1 if there is no more edges in that direction.

    Dim GTriggerSample As Int64
    Dim GX1Sample As Int64
    Dim GX2Sample As Int64

    Public Sub SetCaptureParameters(ByVal TriggerSample As Int64, ByVal X1Sample As Int64, ByVal X2Sample As _ Int64)
        ' This routine is called at the end of a capture to pass the Trigger and X1 and X2 Cursors to the
        ' custom decoding process.
        GTriggerSample = TriggerSample
        GX1Sample = X1Sample
        GX2Sample = X2Sample

    End Sub


  Public Sub DecodeCustom(ByVal OutFilename As String, ByVal NumberOfSamples As Int64, ByVal SamplingRate As _ Integer, ByVal Parameters
As String)
        Dim OldSample As UInteger

        Try

            ' This is a custom bus decoder Processing Routine
            '
            ' The passed in variables are as follows:
            ' OutFilename     - the file that all of the decoded Entries get written to.  This is the file
            ' that the USBee Suite
            '                   will read to display the data on the waveline.
            ' ActualNumberOfSamples - How many samples are in the sample buffer
            ' SamplingRate    - The rate that the samples were taken in sps.  24000000 = 24Msps...
            ' Parameters      - User defined string passed from the USBee Suite user interface Channel Setting
            ' for the custom decoder.
            '                   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)
            Dim Sample As Int64

            SampleData(0)   ' Initializes the sample retrieval


            ' Since this file supports different custom decoders, we need to see which one to run based on the Parameters string
            If CBool(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(CChar(" ,-"))
                Dim SignalToUse As Double = Val(Params(1))
                Dim SignalMask As Integer = 1 << CInt(SignalToUse)   ' Make the mask that will mask off the channel we want in the sample

                ' Now go from the start of the samples to the end and process the decoder
                For Sample = 0 To NumberOfSamples - 1
                    ' This example decoder places a label at every transition of a digital signal
                    Dim NextEdge = FindNextEdge(Sample, SignalMask, 1)
                    If NextEdge >= 0 Then       ' Edge was found
                        WriteEntry(BW, NextEdge, NextEdge + 100, "Changed! S:" & Sample & "N:" & NextEdge)
                    Else
                        ' There are no more edges so end
                        Exit For
                    End If
                    Sample = NextEdge - 1

                Next

            ElseIf CBool(InStr(Parameters.ToUpper, "RISE")) 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(CChar(" ,-"))
                Dim SignalToUse As Double = Val(Params(1))
                Dim SignalMask As Integer = 1 << CInt(SignalToUse)  ' Make the mask that will mask off the channel we want in the sample

                ' Now go from the start of the samples to the end and process the decoder
                For Sample = 0 To NumberOfSamples - 1
                    ' This example decoder places a label at every transition of digital signal 0
                    Dim DigitalChannel As UInteger = CUInt(SampleData(Sample) And SignalMask)
                    If (DigitalChannel <> OldSample) And (OldSample = 0) Then
                        WriteEntry(BW, Sample, Sample + 100, "Rise!")
                    End If
                    OldSample = DigitalChannel
                Next

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

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

                WriteEntry(BW, 0, 100, "Hello World!")
                WriteEntry(BW, GTriggerSample, GTriggerSample + 100, "Trigger Is Here!")
                WriteEntry(BW, GX1Sample, GX1Sample + 100, "X1 Is Here!")
                WriteEntry(BW, GX2Sample, GX2Sample + 100, "X2 Is Here!")

            ElseIf CBool(InStr(Parameters.ToUpper, "NECIR")) 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
                WriteEntry(BW, 0, 100, "NEC IR Decoder 3.0")


                Dim Params() = Parameters.Split(CChar(" ,-"))
                Dim SignalToUse As Double = Val(Params(1))
                Dim SignalMask As Integer = 1 << CInt(SignalToUse)   'Make the mask that will mask off the channel we want in the sample

                Const LOOKING_FOR_HEADER As Integer = 1
                Const LOOKING_FOR_BITS As Integer = 2
                Dim DecodeState As Integer = LOOKING_FOR_HEADER     ' Holds what state of the decoder we are in

                Dim ByteAccumlator As Integer = 0                   ' Holds the accumulated bits for each byte
                Dim BitCounter As Integer = 0                       ' Holds how many bits we have accumulated in this byte so far
                Dim ByteStartSample As Int64                      ' Holds the sample at the start of the byte

                Dim Data As UInteger                                ' Holds the state of the signal at the current sample
                Dim tEdge1 As Int64                               ' Where the first edge is
                Dim tEdge2 As Int64                               ' Where the second edge is
                Dim tPulseWidth As Double                           ' The pulsewidth in seconds

                ' Now go from the start of the samples to the end and process the signal
                For Sample = 0 To NumberOfSamples - 1

                    Data = CUInt(SampleData(Sample) And SignalMask)

                    If DecodeState = LOOKING_FOR_HEADER Then

                        If Data <> 0 Then
                            ' We found a High which starts the Header
                            ' Now look for the next edge

                            tEdge1 = FindNextEdge(Sample, SignalMask, 1)

                            If tEdge1 >= 0 Then

                                ' Check to see if this falling edge is in the right timeframe
                                tPulseWidth = (tEdge1 - Sample) / SamplingRate

                                If (tPulseWidth >= 0.008) And (tPulseWidth <= 0.01) Then

                                    ' Now look for the rising edge

                                    tEdge2 = FindNextEdge(tEdge1, SignalMask, 1)

                                    If tEdge2 >= 0 Then
                                        ' Check to see if this rising edge is in the right timeframe
                                        tPulseWidth = (tEdge2 - tEdge1) / SamplingRate
                                        If (tPulseWidth >= 0.004) And (tPulseWidth <= 0.005) Then
                                            ' Great!  Valid Header Format!  Look for bits from this point on

                                            ' Write out a Header Marker (remove this if you don't need the header)
                                            WriteEntry(BW, Sample, tEdge2, "Lead Code")

                                            DecodeState = LOOKING_FOR_BITS
                                            Sample = tEdge2

                                            ' Initialize the Byte Accumulation variables
                                            ByteStartSample = Sample
                                            ByteAccumlator = 0
                                            BitCounter = 0

                                            Continue For
                                        ElseIf (tPulseWidth >= 0.002) And (tPulseWidth <= 0.003) Then
                                            ' Great!  Valid Repeat Format!

                                            ' Find the next falling edge to complete the Repeat
                                            tEdge2 = FindNextEdge(tEdge2, SignalMask, 1)

                                            ' Write out a Repeat Marker (remove this if you don't need the Repeat)
                                            WriteEntry(BW, Sample, tEdge2, "Repeat")

                                            Sample = tEdge2
                                            Continue For

                                        Else
                                            ' Pulse is not the right size so bail and keep looking
                                            Sample = tEdge2
                                            Continue For
                                        End If
                                    Else
                                        ' No edges at all!  So we are done
                                        Exit For
                                    End If

                                Else
                                    ' Pulse is not the right size so bail and keep looking
                                    Sample = tEdge1
                                    Continue For
                                End If
                            Else
                                ' No edges at all!  So we are done
                                Exit For
                            End If
                        End If

                    ElseIf DecodeState = LOOKING_FOR_BITS Then

                        If BitCounter = 8 Then
                            ' We have an entire byte worth of data so output the information
                            WriteEntry(BW, ByteStartSample, tEdge2, Hex(ByteAccumlator))
                            BitCounter = 0
                            ByteAccumlator = 0
                        End If

                        If Data <> 0 Then
                            ' We found a High which starts the bit
                            ' Now look for the next edge

                            tEdge1 = FindNextEdge(Sample, SignalMask, 1)

                            If tEdge1 >= 0 Then

                                ' Check to see if this falling edge is in the right timeframe
                                tPulseWidth = (tEdge1 - Sample) / SamplingRate

                                If (tPulseWidth >= 0.0005) And (tPulseWidth <= 0.0007) Then
                                    ' Good start of a bit
                                    ' Now look for the rising edge

                                    tEdge2 = FindNextEdge(tEdge1, SignalMask, 1)

                                    If tEdge2 >= 0 Then
                                        ' Check to see if this rising edge is in the right timeframe for a logic "0"
                                        tPulseWidth = (tEdge2 - tEdge1) / SamplingRate
                                        If (tPulseWidth >= 0.0004) And (tPulseWidth <= 0.0006) Then
                                            ' Great!  Valid 0 Bit Format!

                                            ' Write out a Bit Marker (remove this if you don't need the bit)
                                            'WriteEntry(BW, CUInt(Sample), CUInt(tEdge2), "0")

                                            ' Add this bit to the accumulators (LSB first)
                                            ByteAccumlator = ByteAccumlator >> 1    ' Shift the Accumulator
                                            ByteAccumlator = ByteAccumlator And &H7F ' Clear out the MSBit

                                            ' Mark the start of the byte if so
                                            If BitCounter = 0 Then ByteStartSample = Sample

                                            ' Next Bit next time
                                            BitCounter = BitCounter + 1

                                            Sample = tEdge2
                                            Continue For
                                        ElseIf (tPulseWidth >= 0.001) And (tPulseWidth <= 0.002) Then
                                            ' Great!  Valid 1 Bit Format!

                                            ' Write out a Bit Marker (remove this if you don't need the bit)
                                            'WriteEntry(BW, CUInt(Sample), CUInt(tEdge2), "1")

                                            ' Add this bit to the accumulators (LSB first)
                                            ByteAccumlator = ByteAccumlator >> 1    ' Shift the Accumulator
                                            ByteAccumlator = ByteAccumlator Or &H80 ' Set the MSBit

                                            ' Mark the start of the byte if so
                                            If BitCounter = 0 Then ByteStartSample = Sample

                                            ' Next Bit next time
                                            BitCounter = BitCounter + 1

                                            Sample = tEdge2
                                            Continue For

                                        Else
                                            ' Pulse is not the right size so bail and keep looking
                                            DecodeState = LOOKING_FOR_HEADER
                                            Sample = tEdge2
                                            Continue For
                                        End If
                                    Else
                                        ' No edges at all!  So we are done
                                        Exit For
                                    End If

                                Else
                                    ' Pulse is not the right size so bail and keep looking
                                    Sample = tEdge1
                                    DecodeState = LOOKING_FOR_HEADER
                                    Continue For
                                End If
                            Else
                                ' No edges at all!  So we are done
                                Exit For
                            End If

                        End If

                    End If
                Next

            End If

            ' Close the Output File
            FS.Close()

        Catch ex As Exception

        End Try

    End Sub


    Public Sub WriteEntry(ByRef BW As BinaryWriter, ByVal StartSample As Int64, ByVal EndSample As Int64, 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 = CUInt(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 As Integer = 0 To CInt(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 a few decoders possible, “CHANGE”, RISE”, “HELLOWORLD” and “NECIR”. 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 or FindNextEdge calls 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.

Since the USBee RX uses sample compression, there may be trillions of actual samples. Therefore, it is best, and will result in faster decoder times, to use the FindNextEdge routine instead of individual SampleData calls when processing a protocol. This routine understands the compression algorithm and does not waste time processing compressed samples.

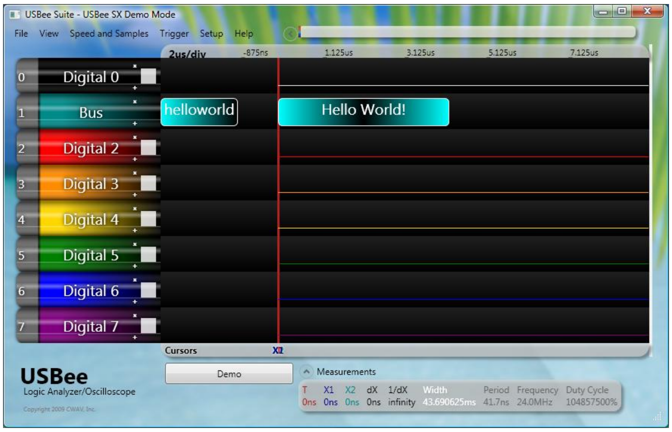
OUTPUTTING ENTRIES THAT WILL GET DISPLAYED ON THE SCREEN

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:

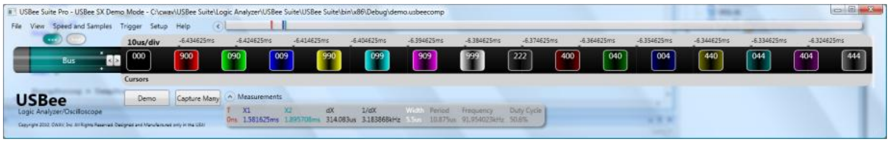
To output an Entry that has a bright green background, use the following:

```
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:



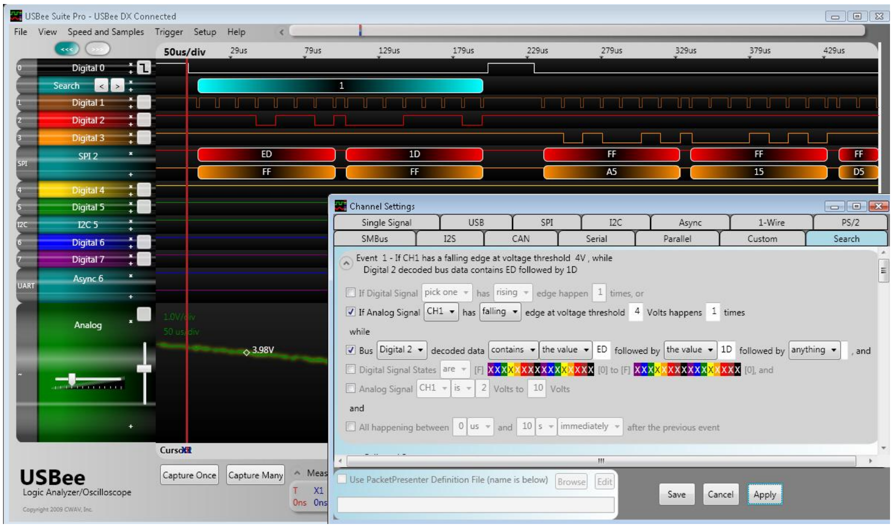
If you do not embed a color code, the background color will be a light cyan.

SMART SEARCH

The USBee RX Suite 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.



Event 1 - If CH1 has a falling edge at voltage threshold 4V , while
Digital 2 decoded bus data contains ED followed by 1D

☐ If Digital Signal pick one has rising edge happen 1 times, or

☑ If Analog Signal CH1 has falling edge at voltage threshold 4 Volts happens 1 times

while

☑ Bus Digital 2 decoded data contains the value ED followed by the value 1D followed by anything , and

☐ Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and

☐ Analog Signal CH1 is 2 Volts to 10 Volts

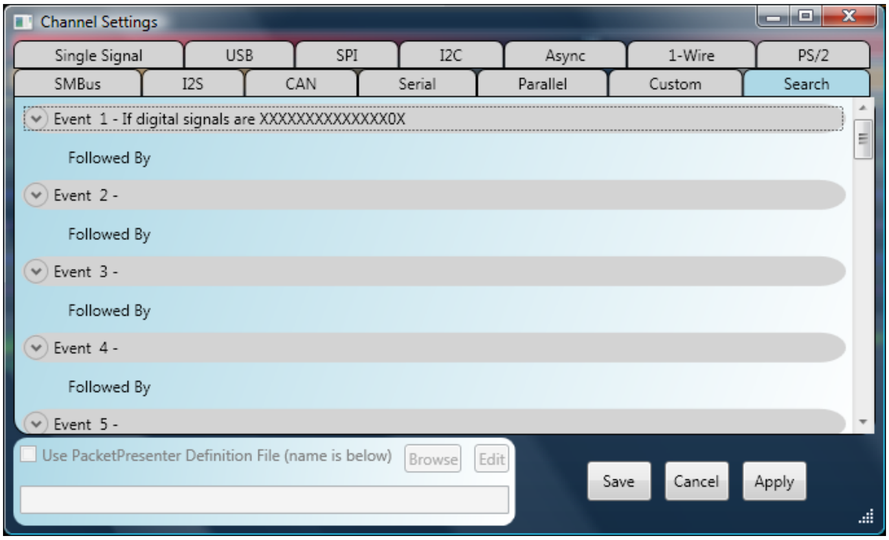
and

☐ All happening between 0 us and 10 s immediately after the previous event

☐ Use PacketPresenter Definition File (name is below) Browse Edit

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.



Event 1 - If digital signals are XXXXXXXXXXXXXX0X

Followed By

Event 2 -

Followed By

Event 3 -

Followed By

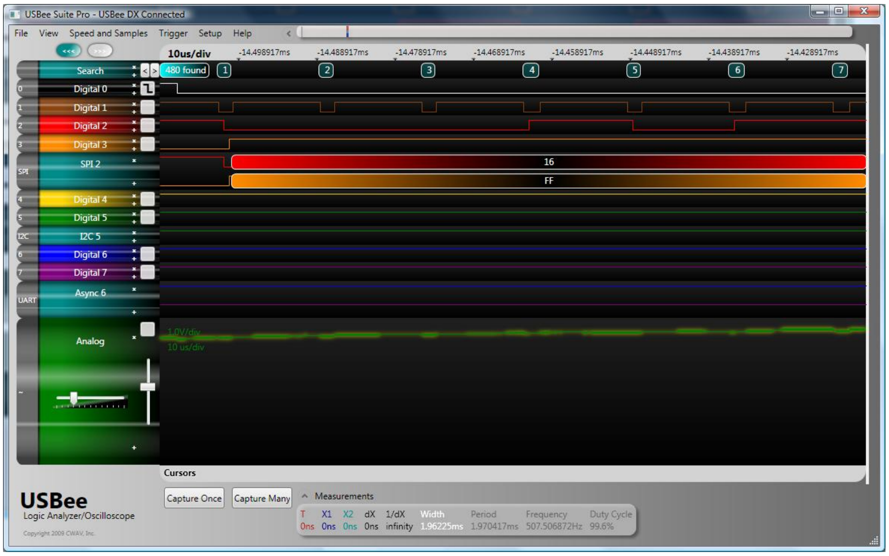
Event 4 -

Followed By

Event 5 -

Use PacketPresenter Definition File (name is 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.



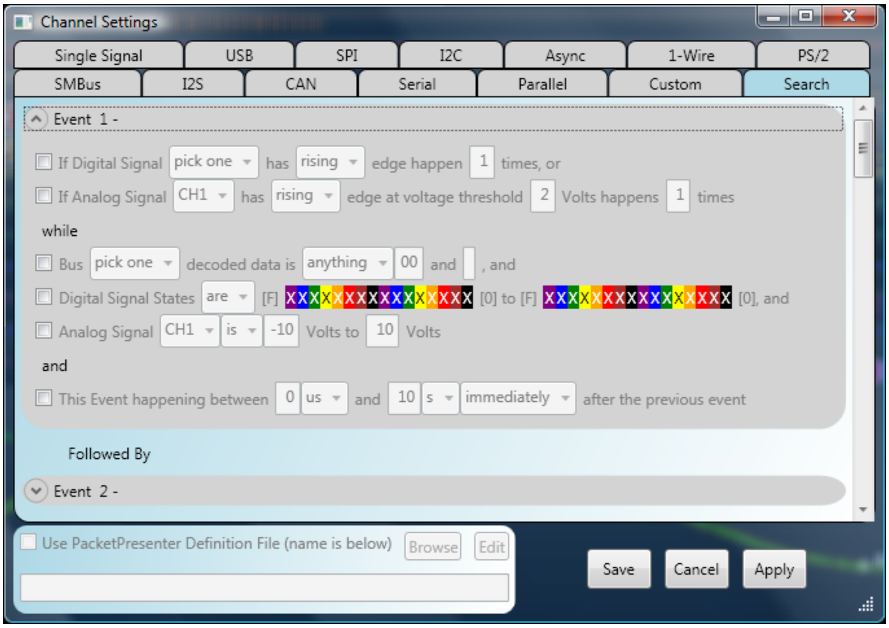
VIEWING SEARCH MATCHES

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.



Event 1 -

If Digital Signal pick one has rising edge happen 1 times, or

If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

Bus pick one decoded data is anything 00 and , and

Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and

Analog Signal CH1 is -10 Volts to 10 Volts

and

This Event happening between 0 us and 10 s immediately after the previous event

Followed By

Event 2 -

Use PacketPresenter Definition File (name is 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

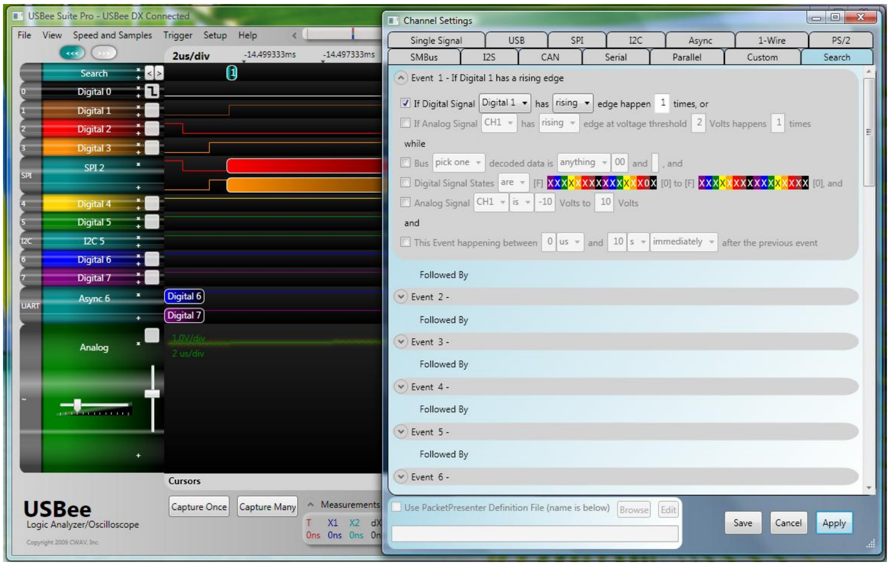
The first line lets you find edges on the digital signals.

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.



Event 1 - If Digital 1 has a rising edge

☑ If Digital Signal [Digital 1] has [rising] edge happen [1] times, or

☐ If Analog Signal [CH1] has [rising] edge at voltage threshold [2] Volts happens [1] times

while

☐ Bus [pick one] decoded data is [anything] [00] and [ ] , and

☐ Digital Signal States [are] [F] XXXXXXXXXXXXXX0X [0] to [F] XXXXXXXXXXXXXXXX [0], and

☐ Analog Signal [CH1] [is] [-10] Volts to [10] Volts

and

☐ This Event happening between [0] [us] and [10] [s] [immediately] after the previous event

Followed By

Event 2 -

Followed By

Event 3 -

Followed By

Event 4 -

Followed By

Event 5 -

Followed By

Event 6 -

☐ Use PacketPresenter Definition File (name is below) [Browse] [Edit]

ANALOG SIGNAL EDGES

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

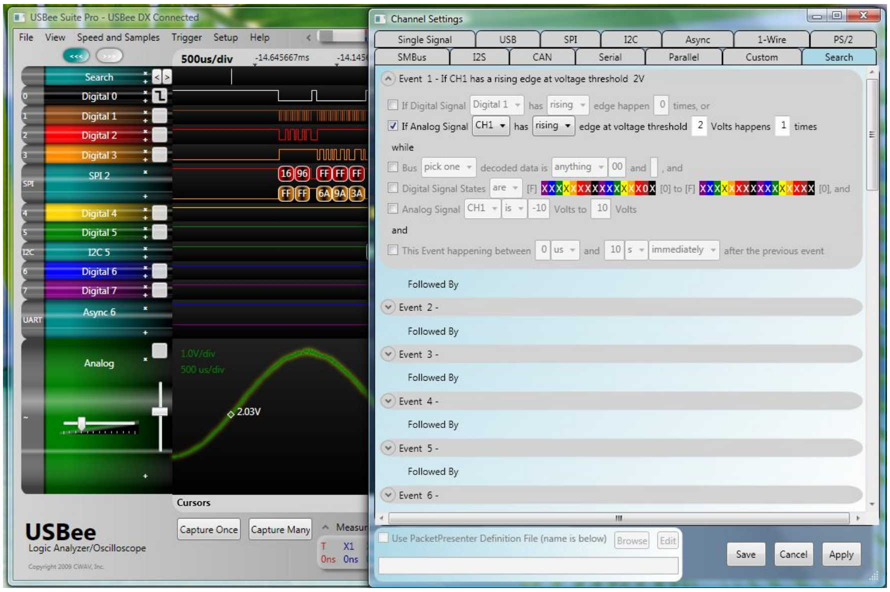
First select the Analog channel you want to search using the first dropdown box.

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.



Event 1 - If CH1 has a rising edge at voltage threshold 2V

☐ If Digital Signal Digital 1 has rising edge happen 0 times, or

☑ If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

☐ Bus pick one decoded data is anything 00 and , and

☐ Digital Signal States are [F] XXXXXXXXXXXXXX0X [0] to [F] XXXXXXXXXXXXXXXX [0], and

☐ Analog Signal CH1 is -10 Volts to 10 Volts

and

☐ This Event happening between 0 us and 10 s immediately after the previous event

Followed By

Event 2 -

Followed By

Event 3 -

Followed By

Event 4 -

Followed By

Event 5 -

Followed By

Event 6 -

☐ Use PacketPresenter Definition File (name is below) Browse Edit

BUS DATA

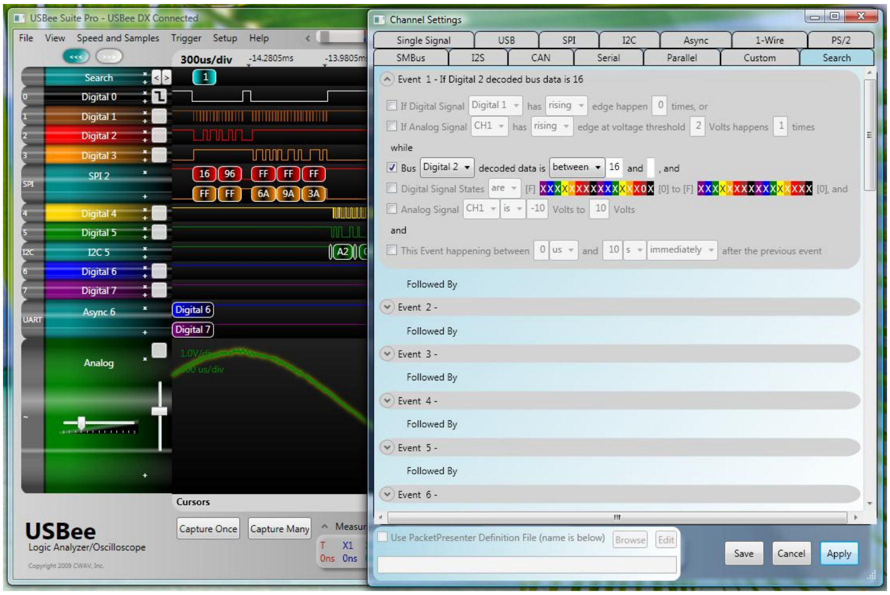
The next line lets you find values of decoded bus data if your waveform display includes them. You can specify specific values to find, values to exclude, or ranges of values.

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.


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
300us/div
Search
Digital 0
Digital 1
Digital 2
Digital 3
SPI 2
Digital 4
Digital 5
I2C 5
Digital 6
Digital 7
Async 6
Analog
Cursors
Capture Once Capture Many
USBee
Logic Analyzer/Oscilloscope
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search
Save Cancel Apply

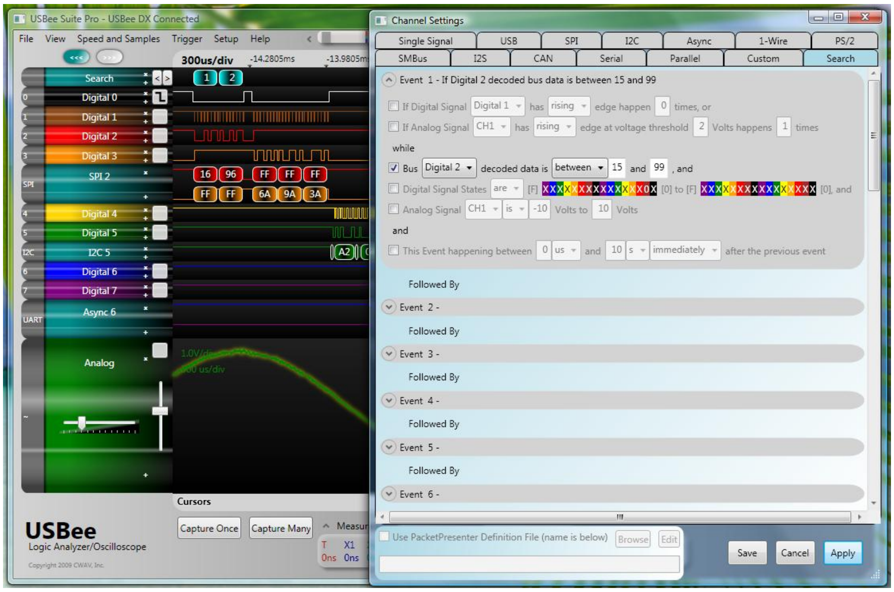
Event 1 - If Digital 2 decoded bus data is 16

[ ] If Digital Signal Digital 1 has rising edge happen 0 times, or
[ ] If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times
while
[✓] Bus Digital 2 decoded data is between 16 and , and
[ ] Digital Signal States are [F] XXXXXXXXXXXXXX0X [0] to [F] XXXXXXXXXXXXXXXX [0], and
[ ] Analog Signal CH1 is -10 Volts to 10 Volts
and
[ ] This Event happening between 0 us and 10 s immediately after the previous event

Followed By
Event 2 -
Followed By
Event 3 -
Followed By
Event 4 -
Followed By
Event 5 -
Followed By
Event 6 -

[ ] Use PacketPresenter Definition File (name is below) Browse Edit

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


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
300us/div
Search
Digital 0
Digital 1
Digital 2
Digital 3
SPI 2
Digital 4
Digital 5
I2C 5
Digital 6
Digital 7
Async 6
Analog
Cursors
Capture Once
Capture Many
USBee
Logic Analyzer/Oscilloscope
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search
Save Cancel Apply

Event 1 - If Digital 2 decoded bus data is between 15 and 99

If Digital Signal Digital 1 has rising edge happen 0 times, or

If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

Bus Digital 2 decoded data is between 15 and 99 , and

Digital Signal States are [F] XXXXXXXXXXXXXX0X [0] to [F] XXXXXXXXXXXXXXXX [0], and

Analog Signal CH1 is -10 Volts to 10 Volts

and

This Event happening between 0 us and 10 s immediately after the previous event

Followed By

Event 2 -

Followed By

Event 3 -

Followed By

Event 4 -

Followed By

Event 5 -

Followed By

Event 6 -

Use PacketPresenter Definition File (name is below) Browse Edit

DIGITAL SIGNAL STATES AND RANGES

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

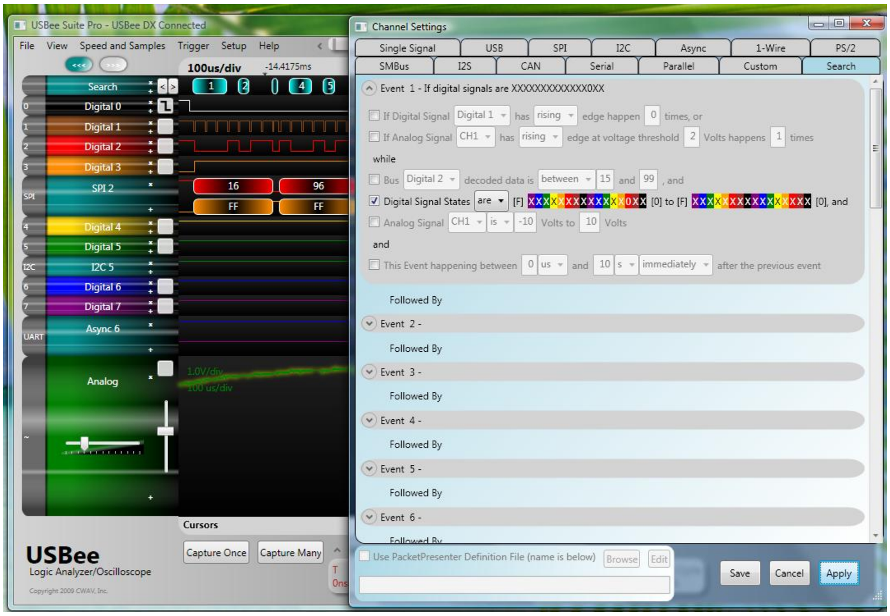
First select if you are looking for the digital states to include the values (**are**), or not include the values (**are not**), that follow.

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.


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
Search
Digital 0
Digital 1
Digital 2
Digital 3
SPI 2
Digital 4
Digital 5
I2C 5
Digital 6
Digital 7
Async 6
Analog
100us/div
-14.4175ms
1.0V/div
100 us/div
Cursors
Capture Once
Capture Many
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search
Followed By
Save Cancel Apply

Event 1 - If digital signals are XXXXXXXXXXXXX0XX

- [ ] If Digital Signal Digital 1 has rising edge happen 0 times, or
- [ ] If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

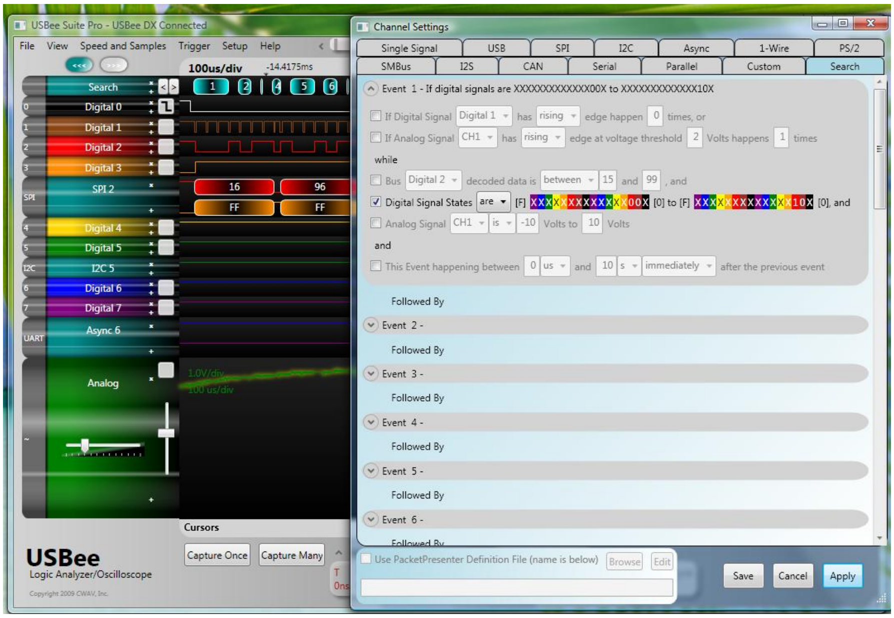
- [ ] Bus Digital 2 decoded data is between 15 and 99, and
- [✓] Digital Signal States are [F] XXXXXXXXXXXXX0XX [0] to [F] XXXXXXXXXXXXXXXX [0], and
- [ ] Analog Signal CH1 is -10 Volts to 10 Volts

and

- [ ] This Event happening between 0 us and 10 s immediately after the previous event

- [ ] Use PacketPresenter Definition File (name is below) Browse Edit

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.


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
Search
Digital 0
Digital 1
Digital 2
Digital 3
SPI 2
Digital 4
Digital 5
I2C 5
Digital 6
Digital 7
Async 6
Analog
100us/div
-14.4175ms
1.0V/div
100 us/div
Cursors
Capture Once
Capture Many
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search
Followed By
Save Cancel Apply

Event 1 - If digital signals are XXXXXXXXXXXX00X to XXXXXXXXXXXX10X

- [ ] If Digital Signal Digital 1 has rising edge happen 0 times, or
- [ ] If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

- [ ] Bus Digital 2 decoded data is between 15 and 99, and
- [✓] Digital Signal States are [F] XXXXXXXXXXXX00X [0] to [F] XXXXXXXXXXXX10X [0], and
- [ ] Analog Signal CH1 is -10 Volts to 10 Volts

and

- [ ] This Event happening between 0 us and 10 s immediately after the previous event

- [ ] Use PacketPresenter Definition File (name is below) Browse Edit

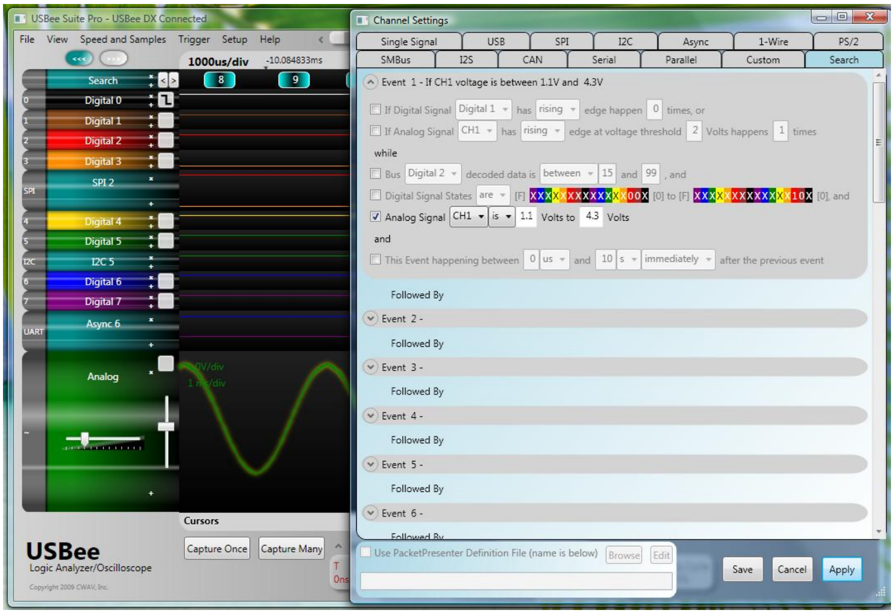
ANALOG SIGNAL STATES AND RANGES

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.


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
1000us/div
Search
Digital 0
Digital 1
Digital 2
Digital 3
SPI 2
Digital 4
Digital 5
I2C 5
Digital 6
Digital 7
Async 6
Analog
Cursors
Capture Once
Capture Many
USBee
Logic Analyzer/Oscilloscope
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search
Save Cancel Apply

Event 1 - If CH1 voltage is between 1.1V and 4.3V

☐ If Digital Signal Digital 1 has rising edge happen 0 times, or

☐ If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

☐ Bus Digital 2 decoded data is between 15 and 99 , and

☐ Digital Signal States are [F] XXXXXXXXXXXXX00X [0] to [F] XXXXXXXXXXXXX10X [0], and

☑ Analog Signal CH1 is 1.1 Volts to 4.3 Volts

and

☐ This Event happening between 0 us and 10 s immediately after the previous event

Followed By

Event 2 -

Followed By

Event 3 -

Followed By

Event 4 -

Followed By

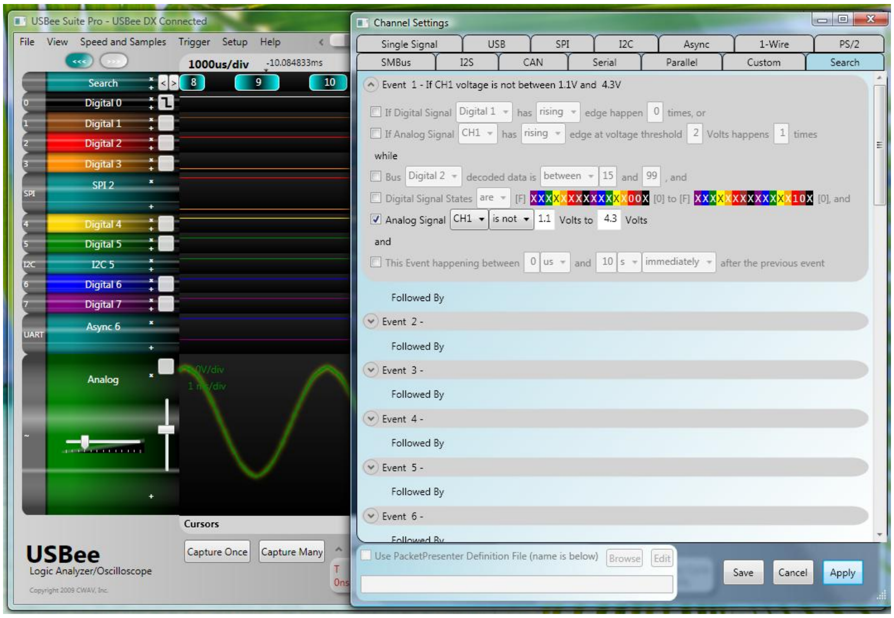
Event 5 -

Followed By

Event 6 -

☐ Use PacketPresenter Definition File (name is below) Browse Edit

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


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
1000us/div
Search
Digital 0
Digital 1
Digital 2
Digital 3
SPI 2
Digital 4
Digital 5
I2C 5
Digital 6
Digital 7
Async 6
Analog
Cursors
Capture Once
Capture Many
USBee
Logic Analyzer/Oscilloscope
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search
Save Cancel Apply

Event 1 - If CH1 voltage is not between 1.1V and 4.3V

☐ If Digital Signal Digital 1 has rising edge happen 0 times, or

☐ If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

☐ Bus Digital 2 decoded data is between 15 and 99 , and

☐ Digital Signal States are [F] XXXXXXXXXXXXX00X [0] to [F] XXXXXXXXXXXXX10X [0], and

☑ Analog Signal CH1 is not 1.1 Volts to 4.3 Volts

and

☐ This Event happening between 0 us and 10 s immediately after the previous event

Followed By

Event 2 -

Followed By

Event 3 -

Followed By

Event 4 -

Followed By

Event 5 -

Followed By

Event 6 -

☐ Use PacketPresenter Definition File (name is below) Browse Edit

TIME WINDOW QUALIFIER

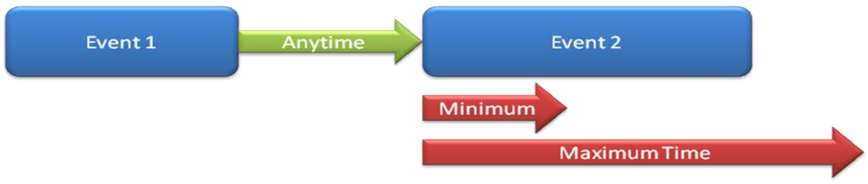
The final line of each Event specifies a time window for the event to occur.

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

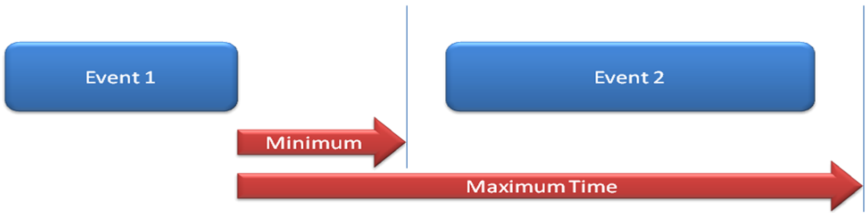
For the first Event, only **anytime** is available. This specifies a time period in which the entire event must occur for a match to be found. For example, if you want to find when two falling edges occur within 10usecs of each other, specify: *This Event happening between 0s and 10us anytime*.

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.

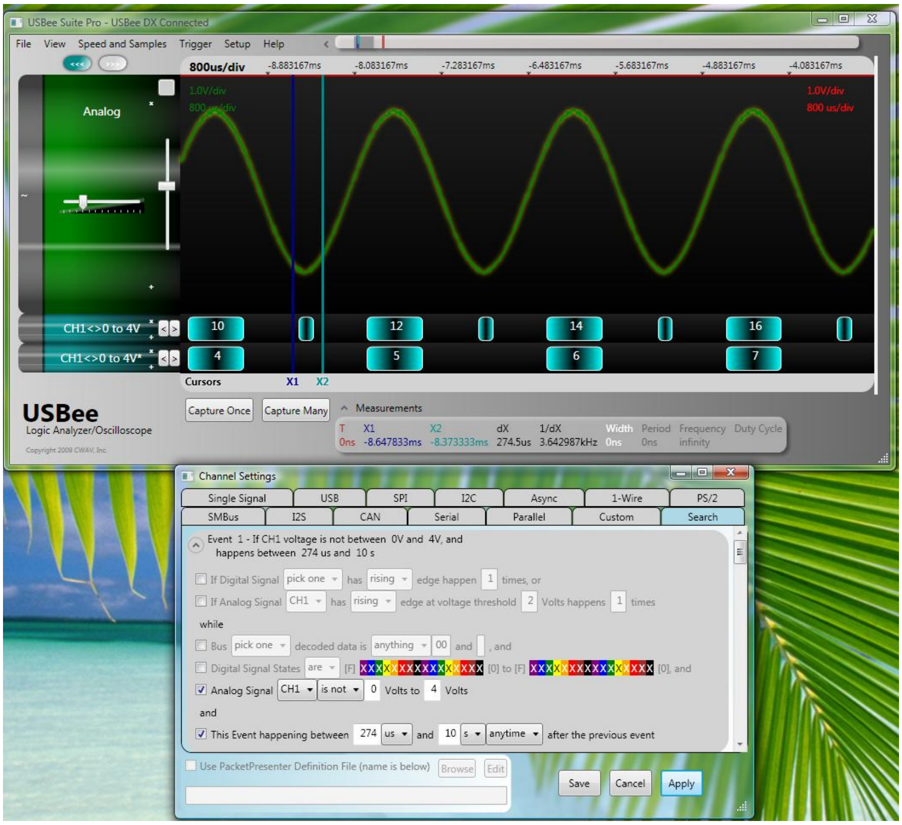


Immediately specifies a time period relative to the end of the previous event in which the entire event must occur for a match to be found. For example, if you want to find when two falling edges occur within 10usecs of each other immediately following the previous event, specify: *This Event happening 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.



l

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.



USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
800us/div
Analog
CH1<>0 to 4V
CH1<>0 to 4V*
Cursors X1 X2
Capture Once
Capture Many
USBee Logic Analyzer/Oscilloscope
Channel Settings
Single Signal USB SPI I2C Async 1-Wire PS/2
SMBus I2S CAN Serial Parallel Custom Search
Save Cancel Apply

Measurements: T 0ns, X1 -8.647833ms, X2 -8.373333ms, dX 274.5us, 1/dX 3.642987kHz, Width 0ns, Period 0ns, Frequency infinity, Duty Cycle

Event 1 - If CH1 voltage is not between 0V and 4V, and happens between 274 us and 10 s

- ☐ If Digital Signal pick one has rising edge happen 1 times, or
- ☐ If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

- ☐ Bus pick one decoded data is anything 00 and , and
- ☐ Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and
- ☑ Analog Signal CH1 is not 0 Volts to 4 Volts

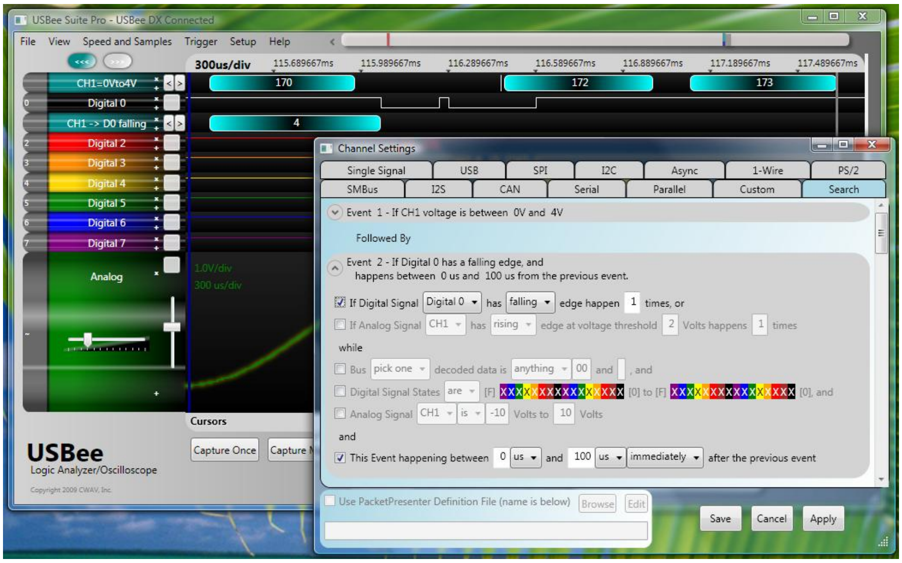
and

- ☑ This Event happening between 274 us and 10 s anytime after the previous event

☐ Use PacketPresenter Definition File (name is below) Browse Edit

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.



Event 1 - If CH1 voltage is between 0V and 4V

Followed By

Event 2 - If Digital 0 has a falling edge, and happens between 0 us and 100 us from the previous event.

[✓] If Digital Signal Digital 0 has falling edge happen 1 times, or

If Analog Signal CH1 has rising edge at voltage threshold 2 Volts happens 1 times

while

Bus pick one decoded data is anything 00 and , and

Digital Signal States are [F] XXXXXXXXXXXXXXXX [0] to [F] XXXXXXXXXXXXXXXX [0], and

Analog Signal CH1 is -10 Volts to 10 Volts

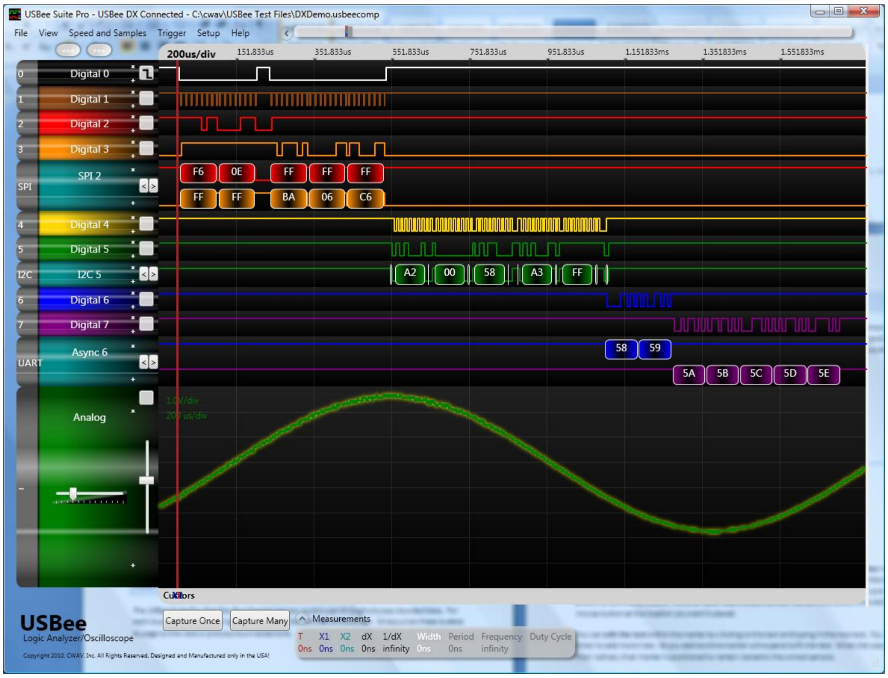
and

[✓] This Event happening between 0 us and 100 us immediately after the previous event

Use PacketPresenter Definition File (name is below) Browse Edit

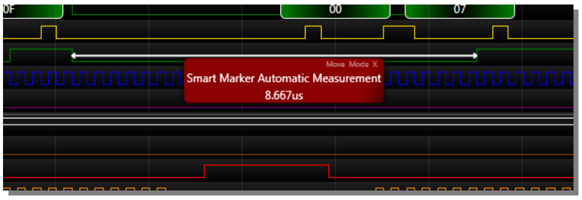
FAST PAN BUS VIEWING

The USBee RX Suite 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.

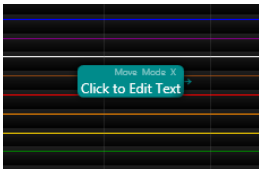


SAMPLE AND SMART MARKERS

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.

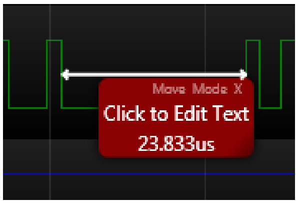
To **move the marker**, click on the Move at the top of the marker. Once you have it moved to the new location use the left mouse button to place it.

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.

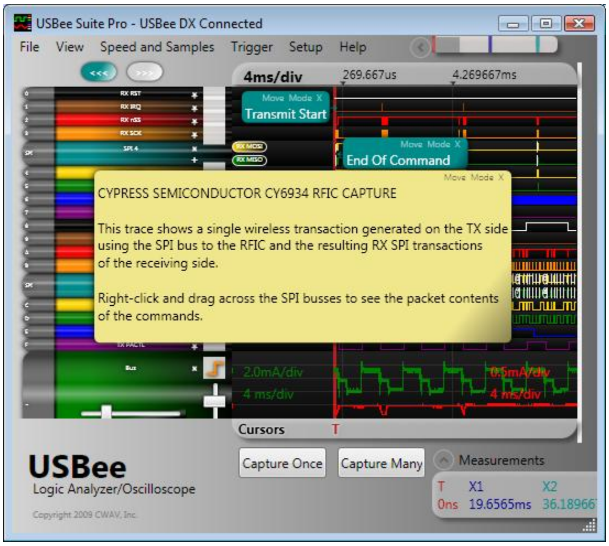
The Mode at the top of the marker **changes the measure mode** of the marker. Click on Mode to cycle through No Measurement, Width, Frequency, Period, and Duty Cycle. An arrow shows the measured area and the measurement shows up as the last line of the marker.

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 RX Suite 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.



CYPRESS SEMICONDUCTOR CY6934 RFIC CAPTURE

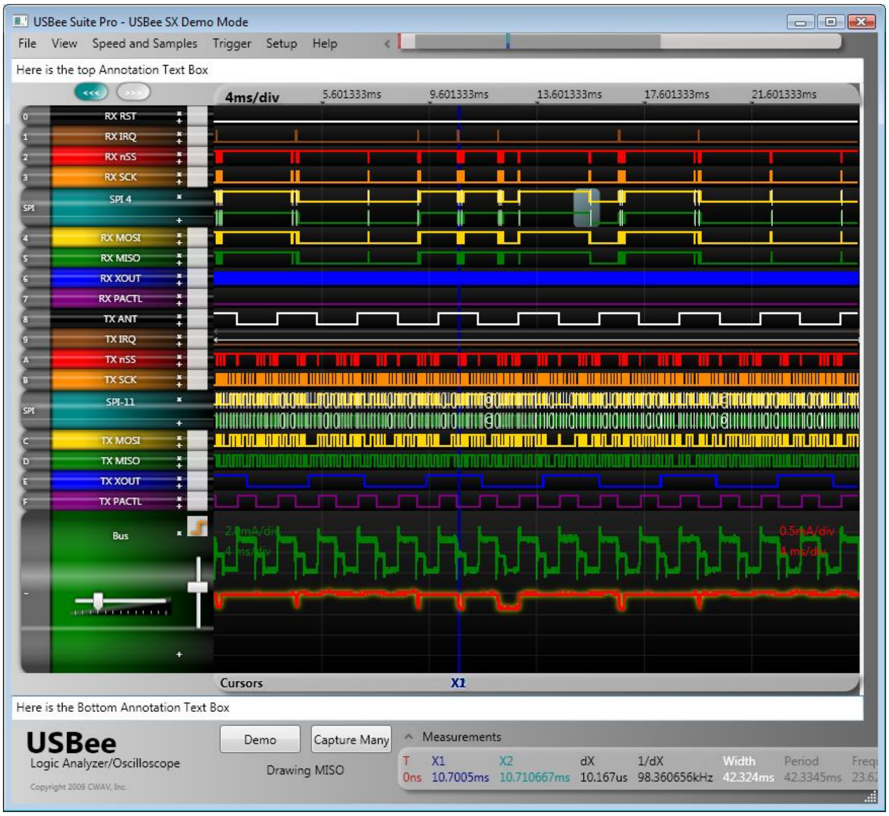
This trace shows a single wireless transaction generated on the TX side using the SPI bus to the RFIC and the resulting RX SPI transactions of the receiving side.

Right-click and drag across the SPI busses to see the packet contents of the commands.

ANNOTATIONS

Annotation Text Boxes are editable text blocks that are located at the Top and Bottom of the USBee RX Suite 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.



Here is the top Annotation Text Box

Here is the Bottom Annotation Text Box

T	X1	X2	dX	1/dX	Width	Period	Freq
0ns	10.7005ms	10.710667ms	10.167us	98.360656kHz	42.324ms	42.3345ms	23.62

Copyright 2009 CWAV, Inc.

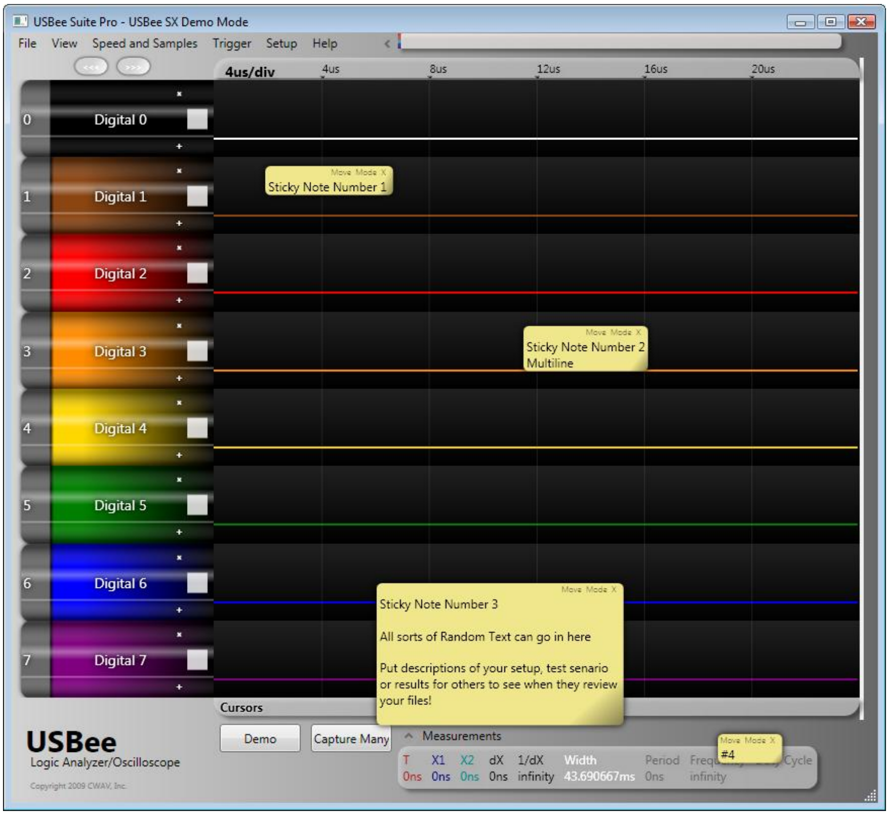
STICKY NOTES

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**.


USBee Suite Pro - USBee SX Demo Mode
File View Speed and Samples Trigger Setup Help
4us/div 4us 8us 12us 16us 20us
Digital 0
Digital 1
Digital 2
Digital 3
Digital 4
Digital 5
Digital 6
Digital 7
Cursors
Demo
Capture Many
Measurements
USBee
Logic Analyzer/Oscilloscope

Move Mode X
Sticky Note Number 1

Move Mode X
Sticky Note Number 2
Multiline

Move Mode X
Sticky Note Number 3

All sorts of Random Text can go in here

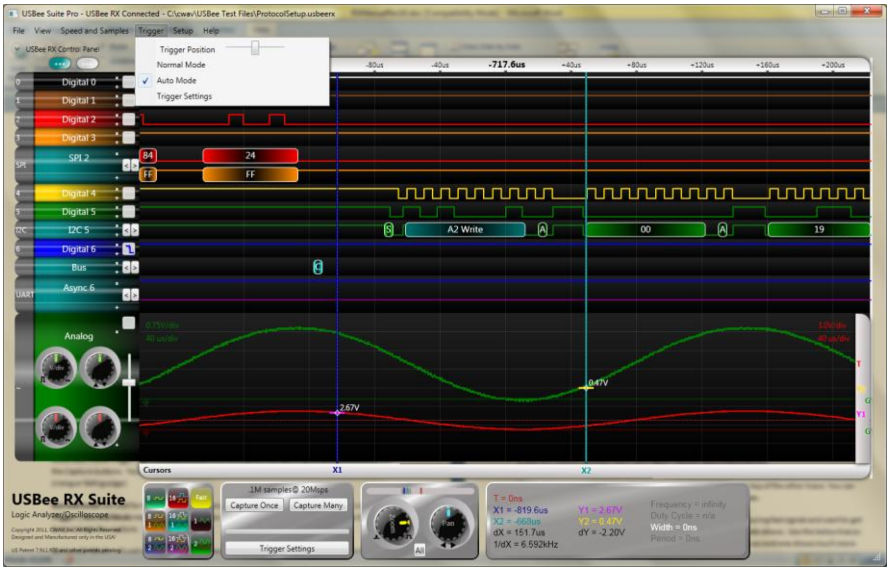
Put descriptions of your setup, test senario or results for others to see when they review your files!

Move Mode X
#4

T	X1	X2	dX	1/dX	Width	Period	Freq	Cycle
0ns	0ns	0ns	0ns	infinity	43.690667ms	0ns	infinity	

ACQUISITION CONTROL

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



When the USBee RX Suite 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.

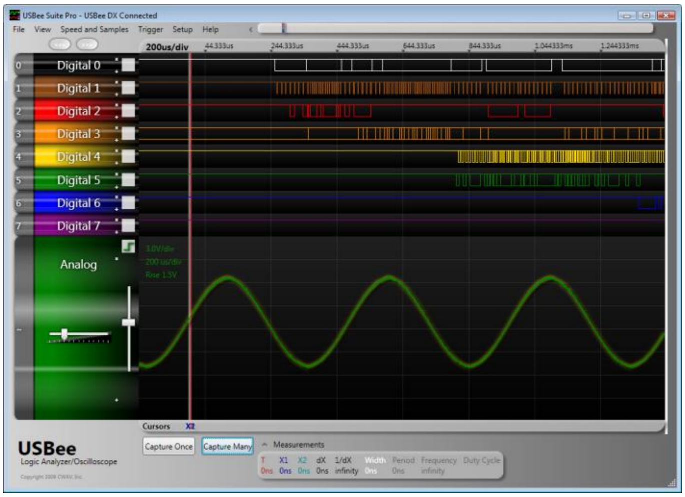
The USBee RX Suite adds the ability to have either Normal Mode (default n the Standard version) or Automatic Mode triggering. This determines when the signals start getting sampled once you press the Capture buttons. You specify the trigger event by selecting one of the USBee signals to trigger on (rising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

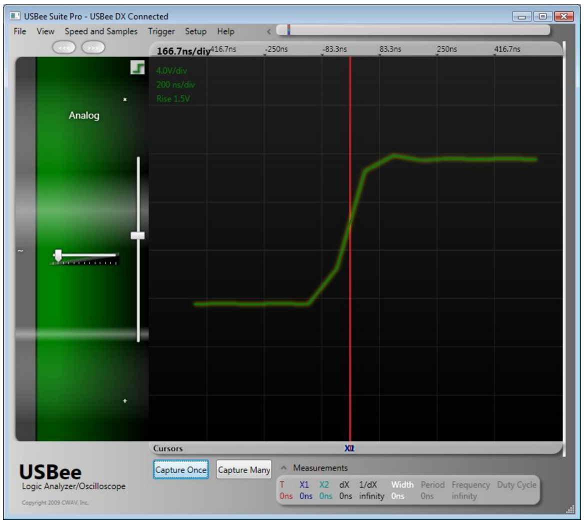


The **Wide** setting shows the waves using a wider pixel setting. This makes the waves easier to see. You can toggle this setting using the menu item **View | Wide Lines.**

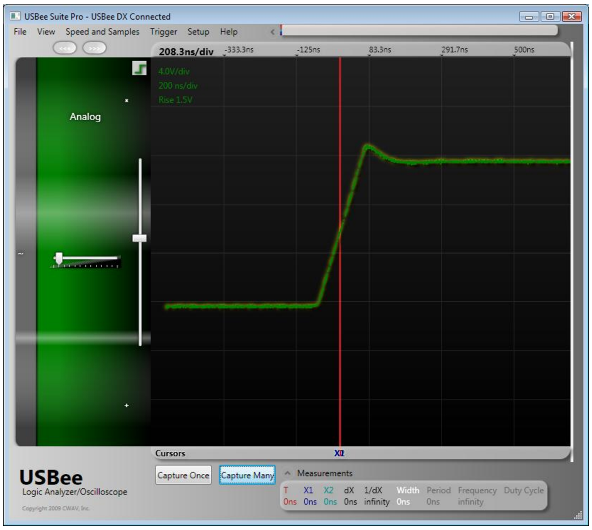
The **Vectors** setting draws the waveforms as a line between adjacent samples. With this mode turned off, the samples are shown simply as dots on the display at the sample position. You can toggle this setting using the menu item **View | Vector-based Waveforms.**

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.


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
166.7ns/div
-416.7ns -250ns -83.3ns 83.3ns 250ns 416.7ns
4.0V/div
200 ns/div
Rise 1.5V
Analog
Cursors
X1
Capture Once
Capture Many
Measurements
T X1 X2 dX 1/dX Width Period Frequency Duty Cycle
0ns 0ns 0ns 0ns infinity 0ns 0ns infinity
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

Persist = OFF, Vectors = ON, Wide = ON


USBee Suite Pro - USBee DX Connected
File View Speed and Samples Trigger Setup Help
208.3ns/div
-333.3ns -125ns 83.3ns 291.7ns 500ns
4.0V/div
200 ns/div
Rise 1.5V
Analog
Cursors
X1
Capture Once
Capture Many
Measurements
T X1 X2 dX 1/dX Width Period Frequency Duty Cycle
0ns 0ns 0ns 0ns infinity 0ns 0ns infinity
USBee
Logic Analyzer/Oscilloscope
Copyright 2009 CWAV, Inc.

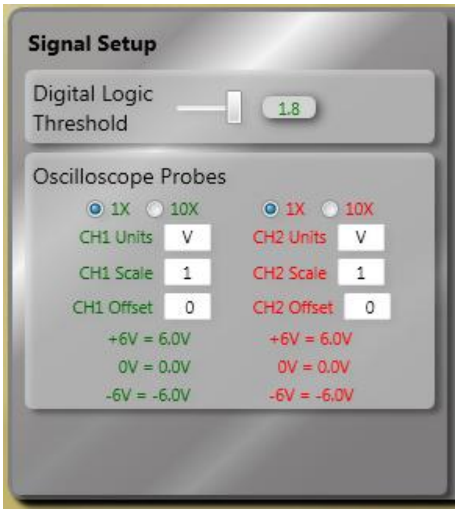
Persist = ON, Vectors = OFF, Wide = ON

ANALOG CHANNELS SCALING

The USBee RX Suite 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.


Signal Setup

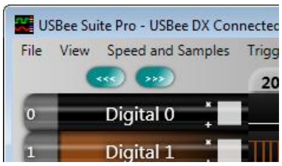
Digital Logic Threshold: 1.8

Oscilloscope Probes

[✓ 1X] 10X	[✓ 1X] 10X
CH1 Units: V	CH2 Units: V
CH1 Scale: 1	CH2 Scale: 1
CH1 Offset: 0	CH2 Offset: 0
+6V = 6.0V	+6V = 6.0V
0V = 0.0V	0V = 0.0V
-6V = -6.0V	-6V = -6.0V

BROWSER-LIKE NAVIGATION

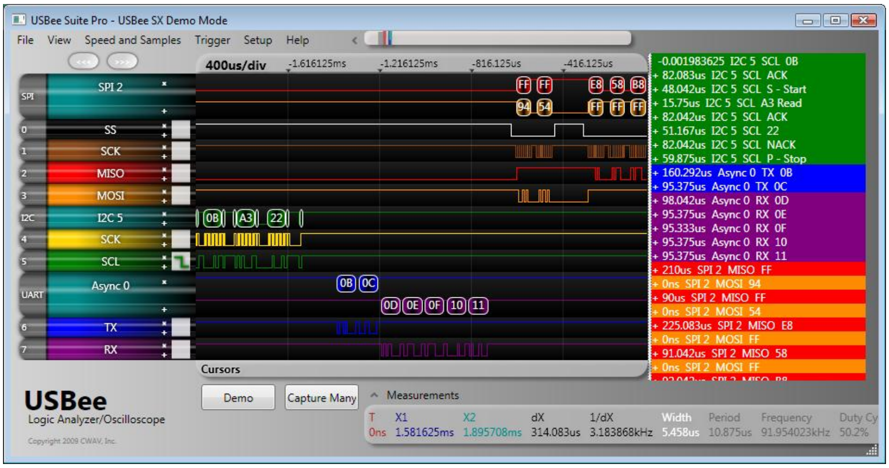
The USBee RX Suite adds browser-like Forward and Back buttons that let you quickly navigate through your trace display.



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.

RELATIVE TIME DECODE

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



```
-0.001983625 I2C 5 SCL 0B
+ 82.083us I2C 5 SCL ACK
+ 48.042us I2C 5 SCL S - Start
+ 15.75us I2C 5 SCL A3 Read
+ 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
```

T	X1	X2	dX	1/dX	Width	Period	Frequency	Duty Cy
0ns	1.581625ms	1.895708ms	314.083us	3.183868kHz	5.458us	10.875us	91.954023kHz	50.2%

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**.

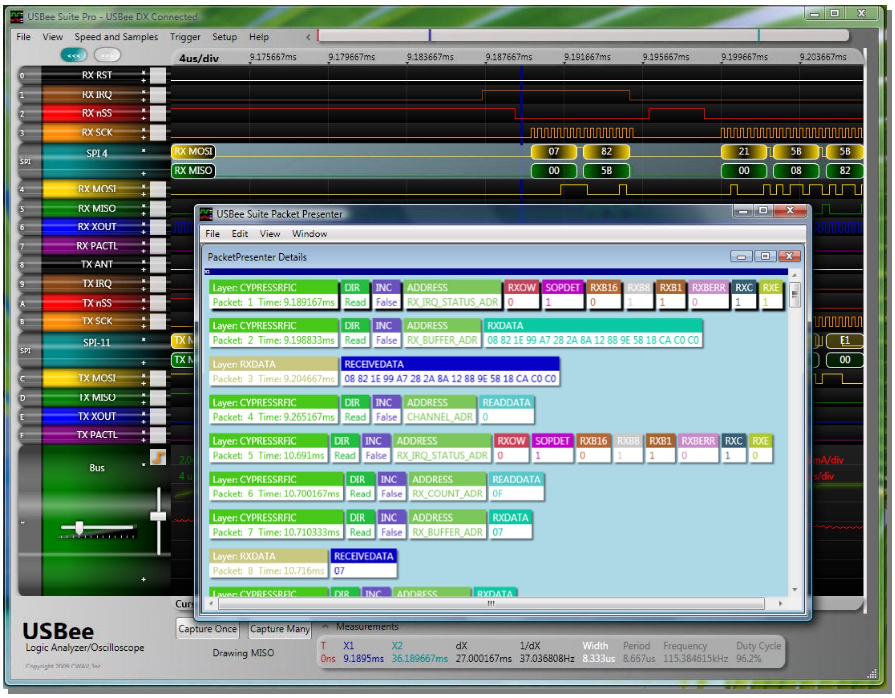
```
0.017596417,SPI 2,MISO,4A
0.017596417,SPI 2,MOSI,FF
0.017688458,SPI 2,MISO,AA
0.017688458,SPI 2,MOSI,FF
0.017802750,I2C 5,SCL,S - Start
0.017818458,I2C 5,SCL,A2 Write
0.017900542,I2C 5,SCL,ACK
0.017933167,I2C 5,SCL,00
0.018015250,I2C 5,SCL,ACK
0.018049208,I2C 5,SCL,4A
0.018131292,I2C 5,SCL,ACK
0.018179375,I2C 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 RX Suite 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.



Layer	DIR	INC	ADDRESS								
Layer: CYPRESSRFIC Packet: 1 Time: 9.189167ms	Read	False	RX_IRQ_STATUS_ADR	RXOW 0	SOPDET 1	RXB16 0	RXB8 1	RXB1 1	RXBERR 0	RXC 1	RXE 1
Layer: CYPRESSRFIC Packet: 2 Time: 9.198833ms	Read	False	RX_BUFFER_ADR	RXDATA 08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0							
Layer: RXDATA Packet: 3 Time: 9.204667ms	RECEIVEDATA 08 82 1E 99 A7 28 2A 8A 12 88 9E 58 18 CA C0 C0										
Layer: CYPRESSRFIC Packet: 4 Time: 9.265167ms	Read	False	CHANNEL_ADR	READDATA 0							
Layer: CYPRESSRFIC Packet: 5 Time: 10.691ms	Read	False	RX_IRQ_STATUS_ADR	RXOW 0	SOPDET 1	RXB16 0	RXB8 1	RXB1 1	RXBERR 0	RXC 1	RXE 0
Layer: CYPRESSRFIC Packet: 6 Time: 10.700167ms	Read	False	RX_COUNT_ADR	READDATA 0F							
Layer: CYPRESSRFIC Packet: 7 Time: 10.710333ms	Read	False	RX_BUFFER_ADR	RXDATA 07							
Layer: RXDATA Packet: 8 Time: 10.716ms	RECEIVEDATA 07										

T	X1	X2	dX	1/dX	Width	Period	Frequency	Duty Cycle
0ns	9.1895ms	36.189667ms	27.000167ms	37.036808Hz	8.333us	8.667us	115.384615kHz	96.2%

OVERVIEW

Using the USBee RX Suite 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.

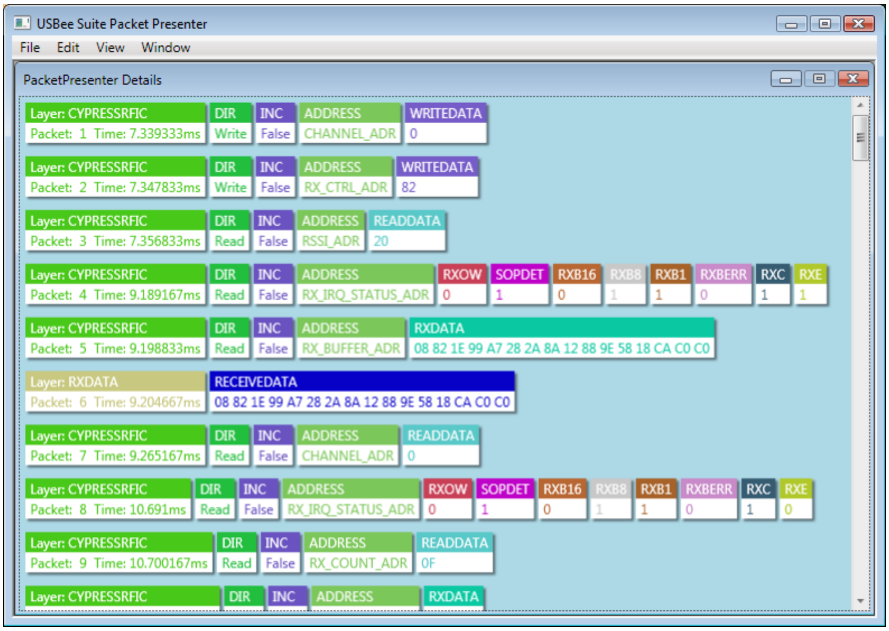
The PacketPresenter™ feature runs alongside of the existing bus decoders of the USBee RX Suite. The PacketPresenter™ takes the output of raw binary data from the bus decoder and parses the stream according to users PacketPresenter Definition File for the intent of displaying the communications in easily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

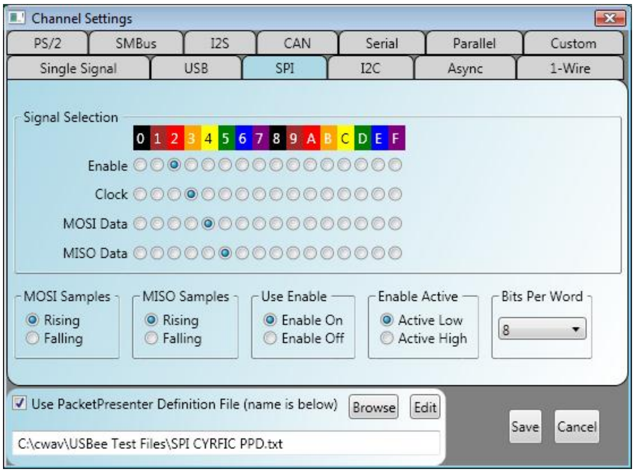
Below shows a sample PacketPresenter output screen.



```
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

Layer: CYPRESSRFIC             DIR   INC   ADDRESS            RXOW SOPDET RXB16 RXB8 RXB1 RXBERR RXC RXE
Packet: 8 Time: 10.691ms       Read  False RX_IRQ_STATUS_ADR  0    1      0     1    1    0      1   0

Layer: CYPRESSRFIC             DIR   INC   ADDRESS            READDATA
Packet: 9 Time: 10.700167ms    Read  False RX_COUNT_ADR       0F

Layer: CYPRESSRFIC             DIR   INC   ADDRESS            RXDATA
```

SETTING UP THE PACKETPRESENTER

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



Channel Settings

PS/2 | SMBus | I2S | CAN | Serial | Parallel | Custom

Single Signal | USB | SPI | I2C | Async | 1-Wire

Signal Selection

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Enable	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○
Clock	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○
MOSI Data	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○
MISO Data	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○

MOSI Samples: [✓ Rising] Falling

MISO Samples: [✓ Rising] Falling

Use Enable: [✓ Enable On] Enable Off

Enable Active: [✓ Active Low] Active High

Bits Per Word: 8

[✓] Use PacketPresenter Definition File (name is below) Browse Edit

C:\cwav\USBee Test Files\SPI CYRFIC PPD.txt

Save Cancel

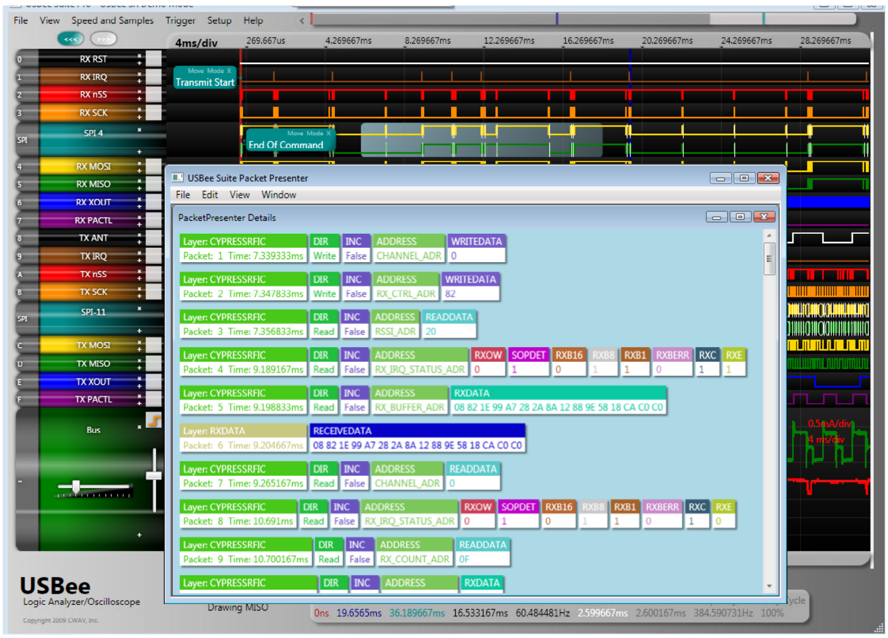
To enable the PacketPresenter for this channel, check the "**Use PacketPresenter definition file (name is below)**" checkbox. Then choose the PacketPresenter definition file by clicking the **Browse** button to the right. Once you choose the file, you can edit the contents by clicking the "Edit File" 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.



```
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

Layer: CYPRESSRFIC              DIR    INC    ADDRESS              RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RXE
Packet: 8  Time: 10.691ms       Read   False  RX_IRQ_STATUS_ADR    0     1       0      1     1     0       1    0

Layer: CYPRESSRFIC              DIR    INC    ADDRESS              READDATA
Packet: 9  Time: 10.700167ms    Read   False  RX_COUNT_ADR         0F

Layer: CYPRESSRFIC              DIR    INC    ADDRESS              RXDATA
```

```
0ns  19.6565ms  36.189667ms  16.533167ms  60.484481Hz  2.599667ms  2.600167ms  384.590731Hz  100%
```

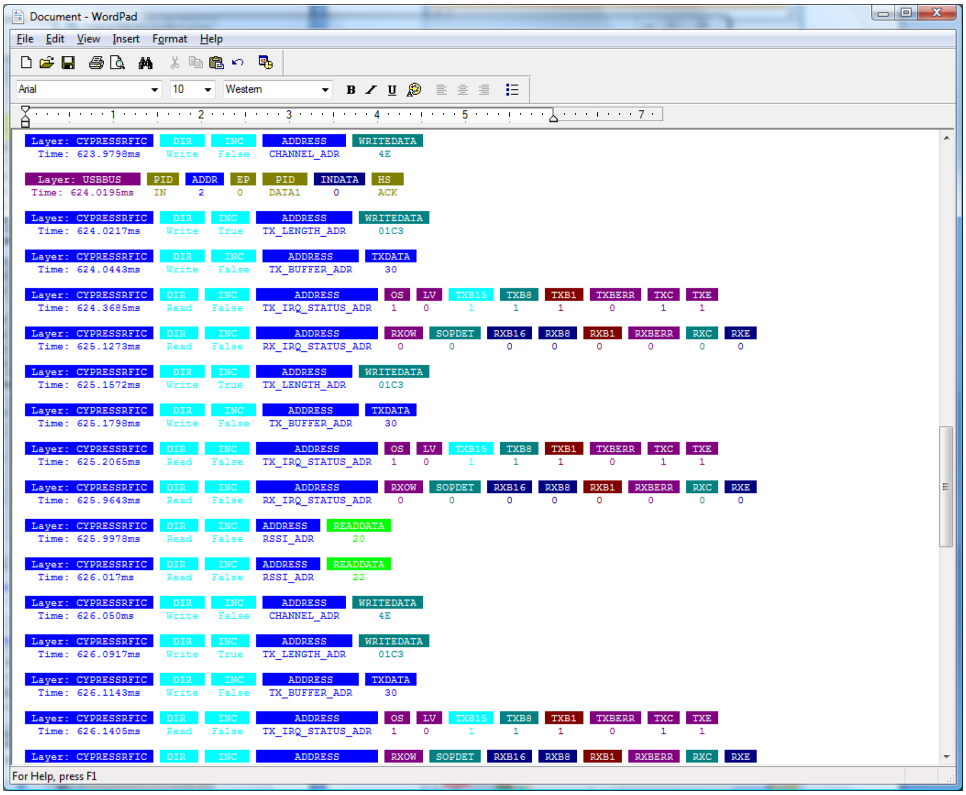
You can show the raw decoded data at the same time by restoring the minimized window as shown in the following screenshot.

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.



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COPYING PACKETPRESENTER OUTPUT TO OTHER PROGRAMS

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

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

You can also use the **Edit | Copy 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.

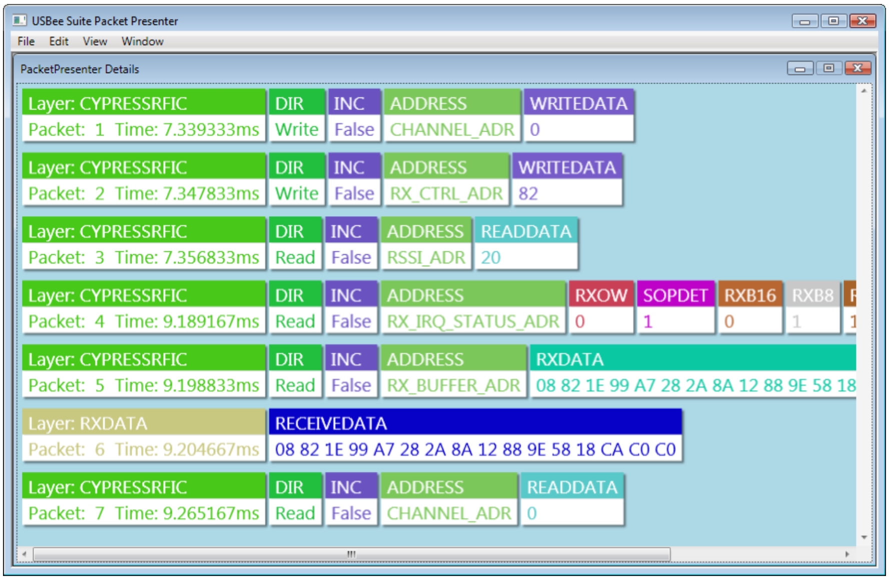
```
     Layer: CYPRESSRFIC       DIR     INC      ADDRESS     WRITEDATA
Packet:  1  Time: 7.339333ms  Write   False   CHANNEL_ADR      0

     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
```

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

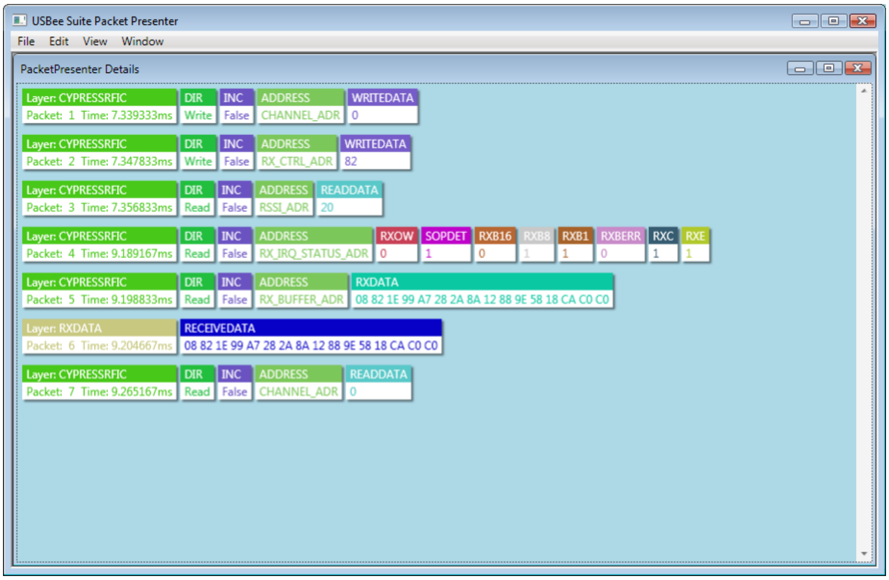
```
     Layer: CYPRESSRFIC       DIR     INC      ADDRESS     WRITEDATA
Packet:  1  Time: 7.339333ms  Write   False   CHANNEL_ADR      0

     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.



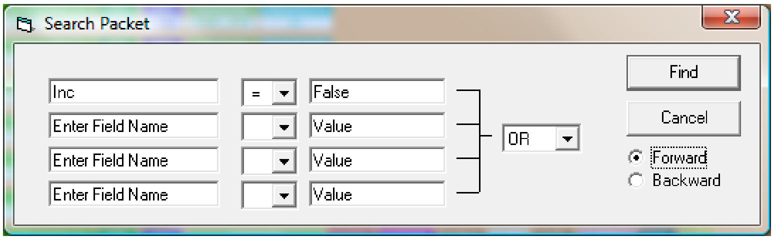
```
Layer: CYPRESSRFIC            DIR    INC    ADDRESS             WRITEDATA
Packet:  1  Time: 7.339333ms  Write  False  CHANNEL_ADR         0

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  R
Packet:  4  Time: 9.189167ms  Read   False  RX_IRQ_STATUS_ADR   0     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

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
```



```
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
```

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.

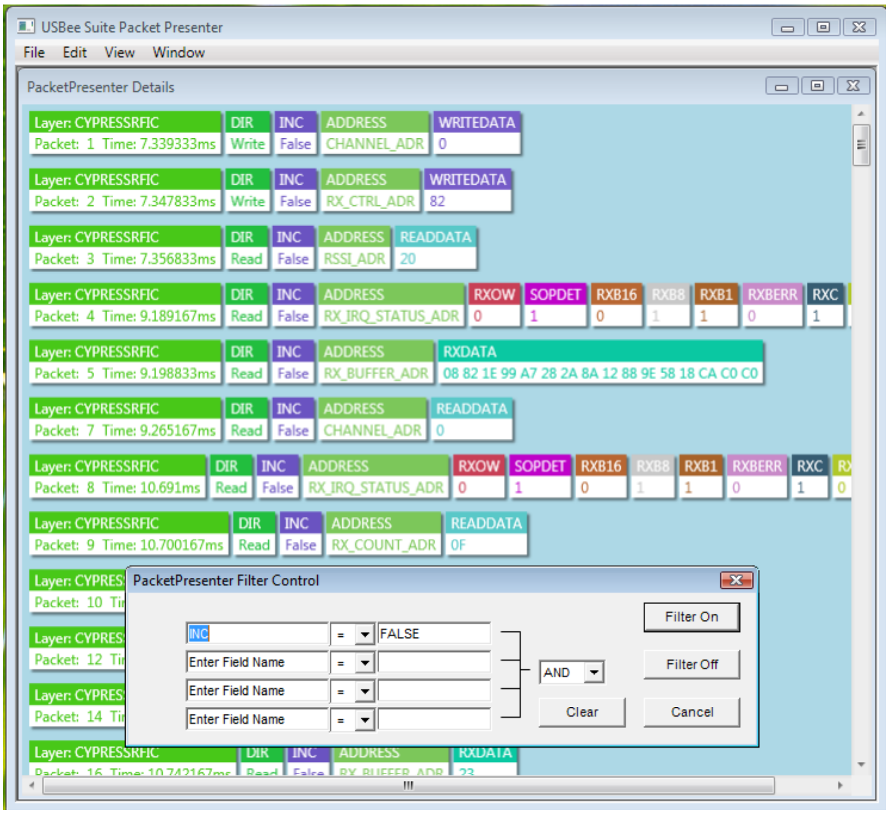


Inc = False
Enter Field Name Value
Enter Field Name Value
Enter Field Name Value
OR
[✓ Forward] Backward

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

FILTERING PACKETS

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



```
Layer: 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
Packet:  4  Time: 9.189167ms  Read   False  RX_IRQ_STATUS_ADR   0     1       0      1     1     0       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: CYPRESSRFIC            DIR    INC    ADDRESS             READDATA
Packet:  7  Time: 9.265167ms  Read   False  CHANNEL_ADR         0

Layer: CYPRESSRFIC            DIR    INC    ADDRESS             RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RX
Packet:  8  Time: 10.691ms    Read   False  RX_IRQ_STATUS_ADR   0     1       0      1     1     0       1    0

Layer: CYPRESSRFIC            DIR    INC    ADDRESS             READDATA
Packet:  9  Time: 10.700167ms Read   False  RX_COUNT_ADR        0F

Layer: CYPRES
Packet:  10  Ti

Layer: CYPRES
Packet:  12  Ti

Layer: CYPRES
Packet:  14  Ti

Layer: CYPRESSRFIC            DIR    INC    ADDRESS             RXDATA
Packet:  16  Time: 10.742167ms Read  False  RX_BUFFER_ADR       23
```

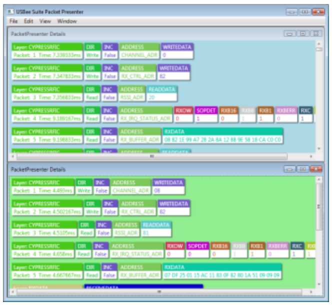
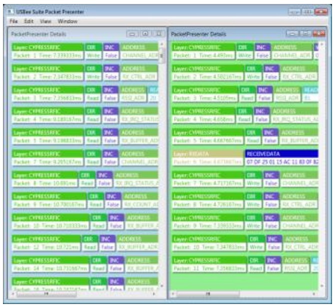
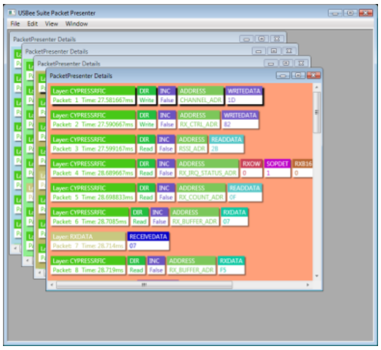
PacketPresenter Filter Control

```
INC               =  FALSE
Enter Field Name  =                 AND
Enter Field Name  =
Enter Field Name  =
```

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.

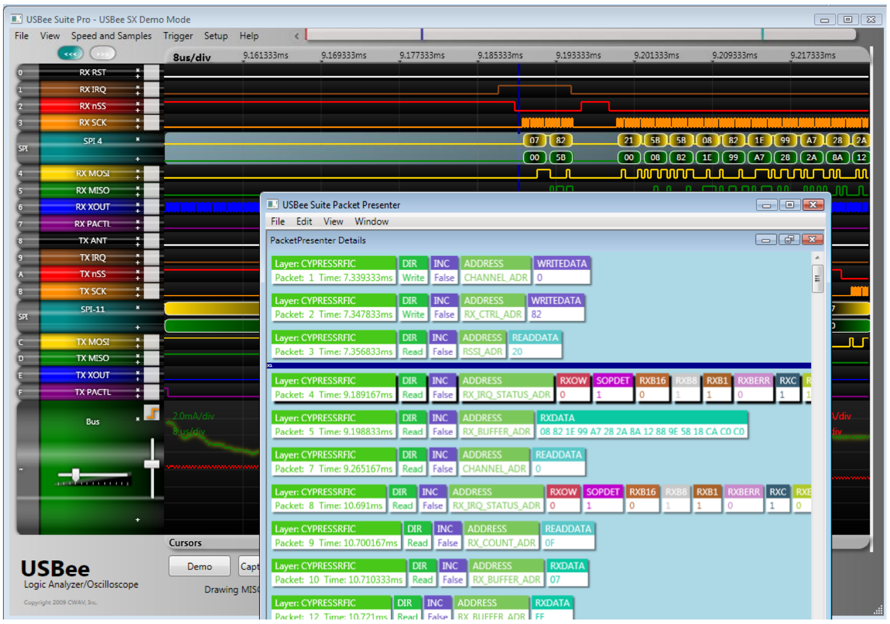
MULTIPLE DECODE DISPLAY

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.



USBee Suite Pro - USBee SX Demo Mode
File View Speed and Samples Trigger Setup Help
8us/div
9.161333ms 9.169333ms 9.177333ms 9.185333ms 9.193333ms 9.201333ms 9.209333ms 9.217333ms
RX RST
RX IRQ
RX nSS
RX SCK
SPI 4
RX MOSI
RX MISO
RX XOUT
RX PACTL
TX ANT
TX IRQ
TX nSS
TX SCK
SPI-11
TX MOSI
TX MISO
TX XOUT
TX PACTL
Bus
2.0mA/div
Cursors
Demo
USBee
Logic Analyzer/Oscilloscope
USBee Suite Packet Presenter
File Edit View Window
PacketPresenter Details

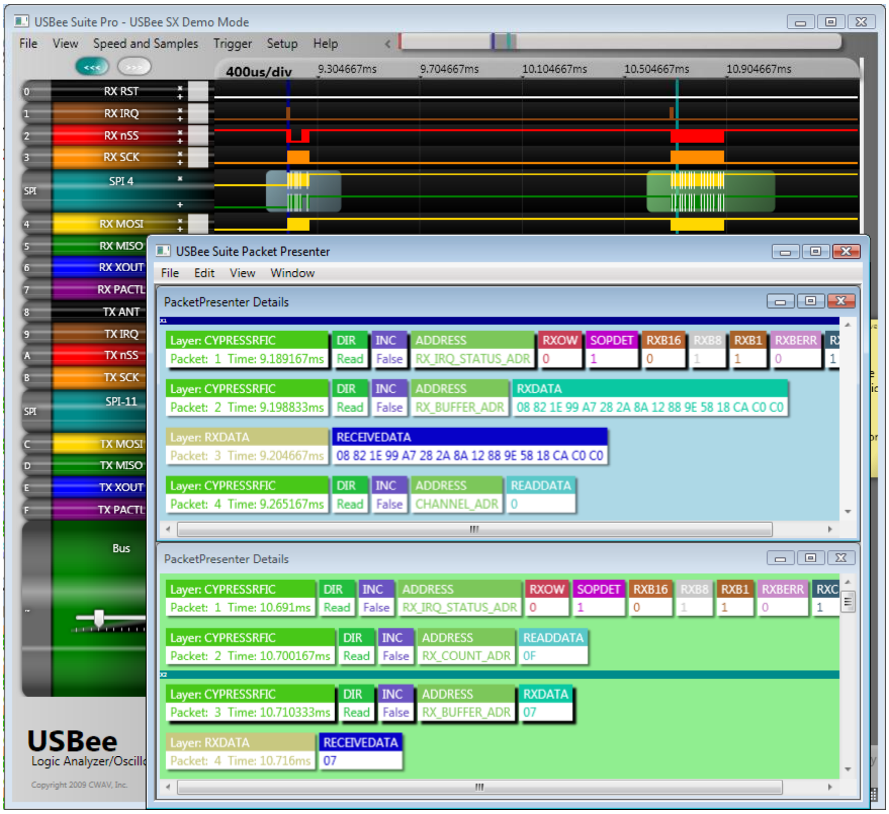
```
Layer: CYPRESSRFIC               DIR    INC    ADDRESS             WRITEDATA
Packet: 1  Time: 7.339333ms      Write  False  CHANNEL_ADR         0

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

Layer: CYPRESSRFIC               DIR    INC    ADDRESS             RXOW  SOPDET  RXB16  RXB8  RXB1  RXBERR  RXC  RXE
Packet: 8  Time: 10.691ms        Read   False  RX_IRQ_STATUS_ADR   0     1       0      1     1     0       1    0

Layer: CYPRESSRFIC               DIR    INC    ADDRESS             READDATA
Packet: 9  Time: 10.700167ms     Read   False  RX_COUNT_ADR        0F

Layer: CYPRESSRFIC               DIR    INC    ADDRESS             RXDATA
Packet: 10  Time: 10.710333ms    Read   False  RX_BUFFER_ADR       07

Layer: CYPRESSRFIC               DIR    INC    ADDRESS             RXDATA
Packet: 12  Time: 10.721ms       Read   False  RX_BUFFER_ADR       FF
```

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 Details (first window)

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

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXDATA
Packet: 2 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: 3 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: 4 Time: 9.265167ms	Read	False	CHANNEL_ADR	0

PacketPresenter Details (second window)

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXOW	SOPDET	RXB16	RXB8	RXB1	RXBERR	RXC
Packet: 1 Time: 10.691ms	Read	False	RX_IRQ_STATUS_ADR	0	1	0	1	1	0	1

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	READDATA
Packet: 2 Time: 10.700167ms	Read	False	RX_COUNT_ADR	0F

Layer: CYPRESSRFIC	DIR	INC	ADDRESS	RXDATA
Packet: 3 Time: 10.710333ms	Read	False	RX_BUFFER_ADR	07

Layer: RXDATA	RECEIVEDATA
Packet: 4 Time: 10.716ms	07

PACKETPRESENTER DEFINITION FILE FORMAT

Each PacketPresenter Definition file defines how the incoming data stream is represented in the PacketPresenter screen of the USBee RX Suite 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 RX Suite 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	Bitwise Busses
• Async	• Serial
• I2C	• I2S
• Parallel	• OneWire
• SPI	• CAN
• PS2	• 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 RX 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 RX software package includes a sample DLL project in Microsoft VC6 format (in the installation directory of the USBee RX 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);
}
```

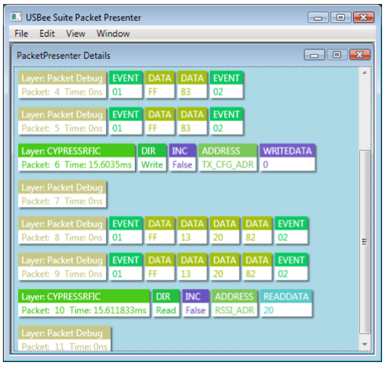
PACKETPRESENTER DEFINITION FILE DEBUGGING

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.



```
Layer: Packet Debug      EVENT  DATA  DATA  EVENT
Packet:  4  Time: 0ns    01     FF    83    02

Layer: Packet Debug      EVENT  DATA  DATA  EVENT
Packet:  5  Time: 0ns    01     FF    83    02

Layer: CYPRESSRFIC              DIR    INC    ADDRESS     WRITEDATA
Packet:  6  Time: 15.6035ms     Write  False  TX_CFG_ADR  0

Layer: Packet Debug
Packet:  7  Time: 0ns

Layer: Packet Debug      EVENT  DATA  DATA  DATA  DATA  EVENT
Packet:  8  Time: 0ns    01     FF    13    20    82    02

Layer: Packet Debug      EVENT  DATA  DATA  DATA  DATA  EVENT
Packet:  9  Time: 0ns    01     FF    13    20    82    02

Layer: CYPRESSRFIC                 DIR   INC    ADDRESS   READDATA
Packet:  10  Time: 15.611833ms     Read  False  RSSI_ADR  20

Layer: Packet Debug
Packet:  11  Time: 0ns
```

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

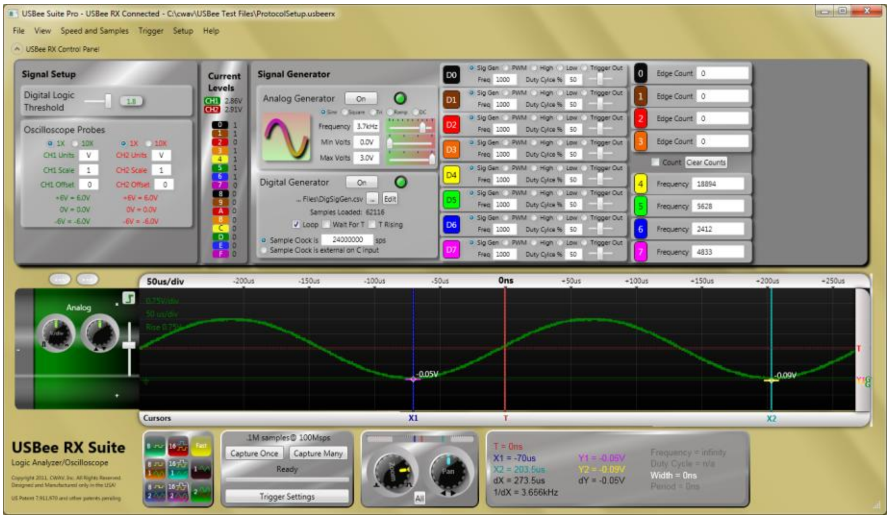
```
; USB Bus Protocol Definition File
; This file defines the transfers to/from a custom USB device
;
[Protocol]
          name = USBBus
          bitwise
[DEBUG]
          ;DebugOn            ; Uncomment this to turn on Debug Packets
[Packet]
          [Start]
                    type = event
    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

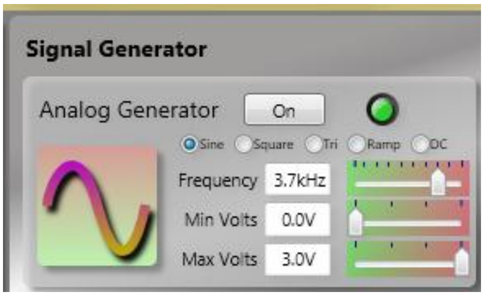
```
; PS2 Protocol Definition File
; This file defines the transfers from a PS2 device
;
[Protocol]
    name = PS2Bus
    bytewise
[DEBUG]
    ;DebugOn       ; Uncomment this to turn on Debug Packets
[Packet]
    [Start]
        type = next    ; Every byte is the start of the next packet
        CHANNELXORY    ; 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 RX ANALOG SIGNAL GENERATOR



The USBee RX has an Analog Signal Generator built in. It outputs a voltage pattern on the **Aout** signal out the side of the Pod.

To specify what waveform pattern is generated, you use the Analog Generator section of the USBee RX Control Panel as seen below.


Signal Generator

Analog Generator [On]

[✓ Sine] Square Tri Ramp DC

Frequency 3.7kHz

Min Volts 0.0V

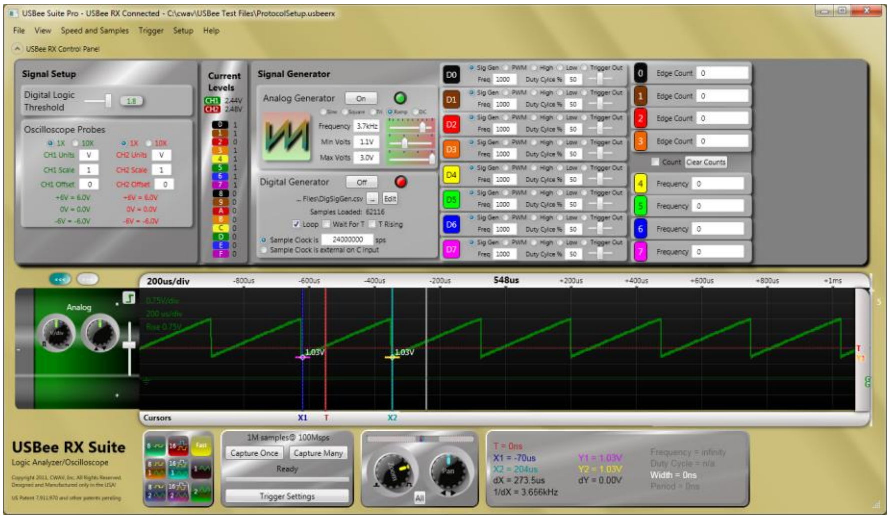
Max Volts 3.0V

There are 5 types of analog signals that can be generated by selecting the associated radio button:

- Sine
- Triangle
- Ramp
- Square and
- DC

You can also select the frequency generated by using the slider to the right of the Frequency line. The minimum and maximum voltages are also specified using the sliders from 0V to 3.0V.

Below are a few of the waveforms available.

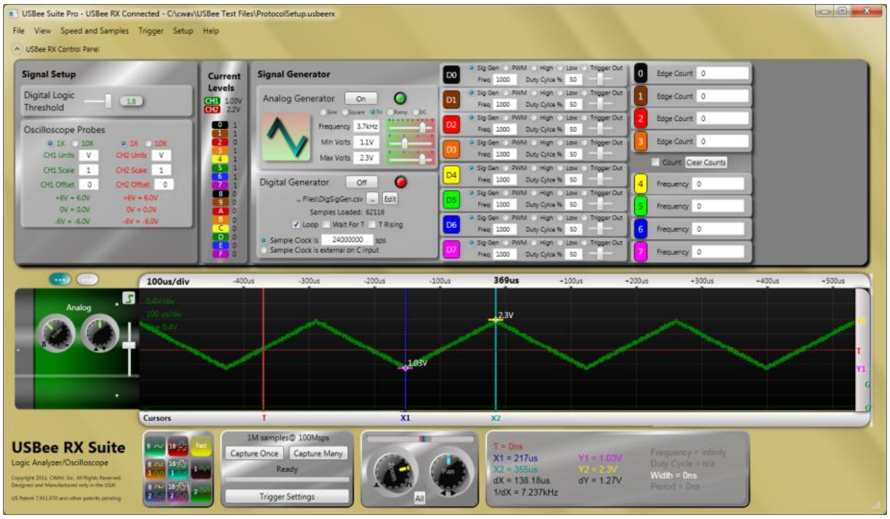

USBee Suite Pro - USBee RX Connected - C:\cwav\USBee Test Files\ProtocolSetup.usbeerx

Analog Generator: On
Frequency 3.7kHz
Min Volts 1.1V
Max Volts 3.0V

Current Levels: CH1 2.44V, CH2 2.48V

200us/div, 548us

T = 0ns
X1 = -70us
X2 = 204us
dX = 273.5us
1/dX = 3.656kHz
Y1 = 1.03V
Y2 = 1.03V
dY = 0.00V
Frequency = infinity
Duty Cycle = n/a
Width = 0ns
Period = 0ns


USBee Suite Pro - USBee RX Connected - C:\cwav\USBee Test Files\ProtocolSetup.usbeerx

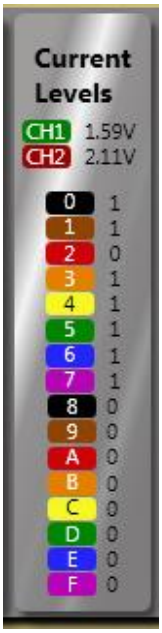
Analog Generator: On
Frequency 3.7kHz
Min Volts 1.1V
Max Volts 2.3V

Current Levels: CH1 1.03V, CH2 2.2V

100us/div, 369us

T = 0ns
X1 = 217us
X2 = 355us
dX = 138.18us
1/dX = 7.237kHz
Y1 = 1.03V
Y2 = 2.3V
dY = 1.27V
Frequency = infinity
Duty Cycle = n/a
Width = 0ns
Period = 0ns

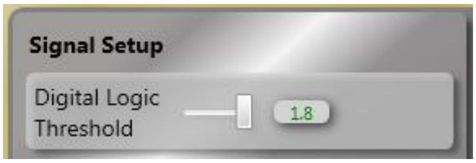
USBEE RX LOGIC PROBE AND DIGITAL VOLTAGE METER



The USBee RX constantly monitors the state of all of the digital inputs and the voltage levels of the CH1 and CH2 inputs. This way you can see the state of the channels even at times you are not capturing waveforms using the Mixed Signal Oscilloscope.

These levels are updated every 500msecs.

USBEE DIGITAL LOGIC THRESHOLD


Signal Setup

Digital Logic Threshold 1.8

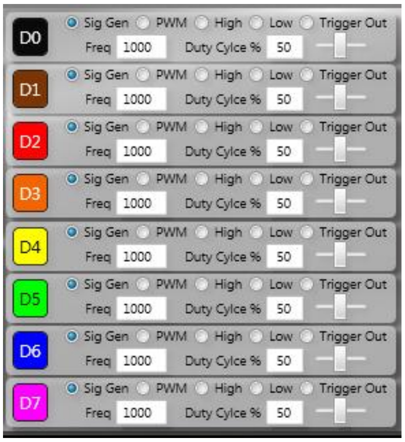
The USBee RX features variable Digital Logic Thresholds that can range from -1V to +2V. Set this logic threshold to define the voltage level that indicates the change from a logic “0” to a logic “1”.

If this level is set incorrectly, or too close to either the top or bottom of your logic range, you will see inconsistent waveforms and this level will need to be adjusted.

USBEE RX DIGITAL SIGNAL GENERATOR

The USBee RX has an Digital Signal Generator built in. It outputs a digital voltage pattern on the **D0 through D7** signals out the side of the Pod.

To specify what waveform pattern is generated on each of the output signals,, you use the section of the USBee RX Control Panel as seen below:



D0 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

D1 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

D2 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

D3 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

D4 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

D5 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

D6 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

D7 [✓ Sig Gen] PWM High Low Trigger Out — Freq 1000 — Duty Cycle % 50

Each of the 8 digital lines can have one of 5 different modes by selecting the associated radio button above:

- The output of the Arbitrary Digital Signal Generator
- Pulse Width Modulator
- Logic High Level
- Logic Low Level
- Trigger Out signal

The Logic **Low** level outputs 0V on the signal.

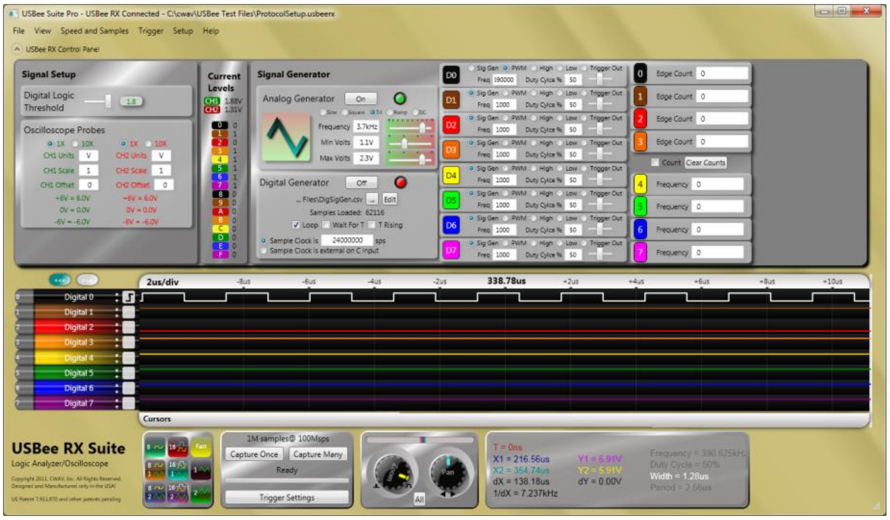
The Logic **High** level outputs 3.3V on the signal.

The **Trigger Out** level outputs 0V before the trigger event and 3.3V after the trigger event. This can be used to trigger other equipment to synchronize the captures of data.

PULSE WIDTH MODULATORS

Each digital output channel can be configured as a Pulse Width Modulator. You specify the frequency in Hz (0 to 390000) desired as well as the duty cycle in percent (0 to 100) of the waveform.

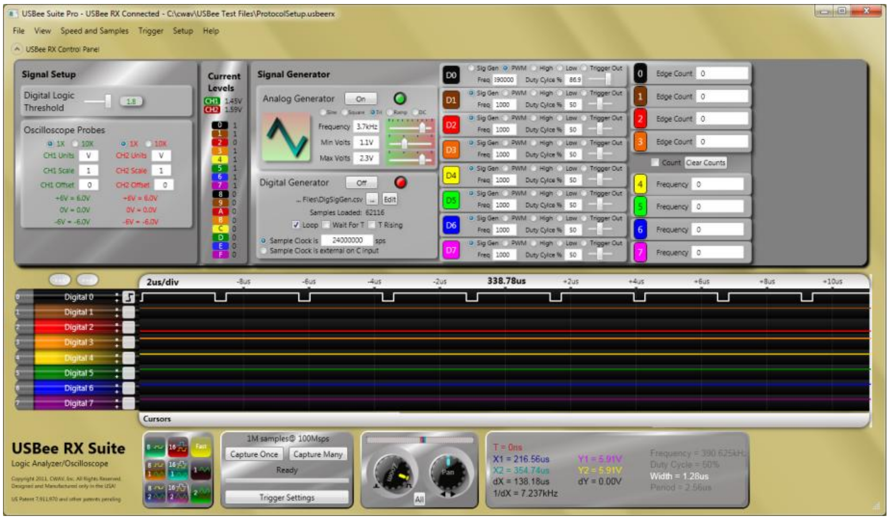
The following shows two examples of a PWM signal with different duty cycles.


USBee Suite Pro - USBee RX Connected - C:\cwav\USBee Test Files\ProtocolSetup.usbeerx

D0: PWM, Freq 390000, Duty Cycle % 50

Current Levels: CH1 1.88V, CH2 1.31V

T = 0ns
X1 = 216.56us
X2 = 354.74us
dX = 138.18us
1/dX = 7.237kHz
Y1 = 5.91V
Y2 = 5.91V
dY = 0.00V
Frequency = 390.625kHz
Duty Cycle = 50%
Width = 1.28us
Period = 2.56us


USBee Suite Pro - USBee RX Connected - C:\cwav\USBee Test Files\ProtocolSetup.usbeerx

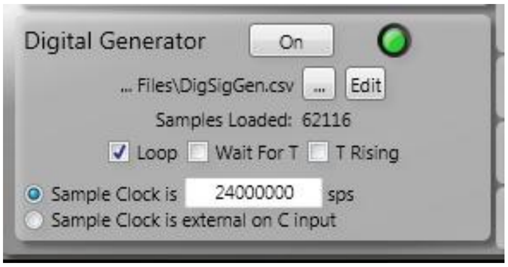
D0: PWM, Freq 390000, Duty Cycle % 86.9

Current Levels: CH1 1.45V, CH2 1.59V

T = 0ns
X1 = 216.56us
X2 = 354.74us
dX = 138.18us
1/dX = 7.237kHz
Y1 = 5.91V
Y2 = 5.91V
dY = 0.00V
Frequency = 390.625kHz
Duty Cycle = 50%
Width = 1.28us
Period = 2.56us

ARBITRARY DIGITAL PATTERN GENERATOR

Each digital output channel can be configured as an arbitrary digital pattern. You specify the pattern by using e Comma Delimited File (CSV) that specifies the Sample Rate (in Hz – 0 to 100000000) and each of the samples. Each pattern can have from 1 to 65535 samples. A CSV file can be created using a simple text editor, or as an output of a program such as Excel.



SPECIFYING THE PATTERN WITH A CSV FILE

To specify the file to use, select the browse button "…" and choose the file. The file will be parsed and the number of samples loaded and sample rate will be sent to the Digital Signal Generator.

The file format of the CSV files is as follows

```
500000, Sample Rate
23, First sample
51, Second sample
…
84, Last Sample
```

Only the first field of each line is considered as a sample. This allows you to use a spreadsheet that computes the samples in the first column. A sample Excel spreadsheet and the resulting .CSV file is included in the Program Files/CWAV Inc/USBee RX Suite directory for you to use.

For 65535 samples it takes about 30 seconds to download.

SAMPLE CLOCKING

The CSV file specifies the desired sample rate of the clocked out digital samples. You can then modify the sample rate by changing the frequency in the sps text box (from 0 to 100000000).

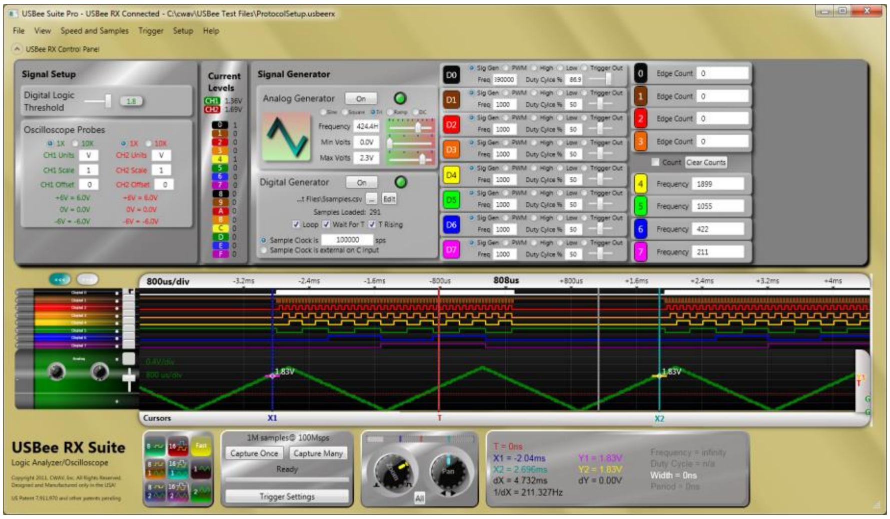
You can also specify to use the external C input as the sample clock by selecting the associated radio button. Every cycle of the C input will clock out the next sample in the sequence.

WAVEFORM GENERATION CONTROL

The **Wait for T** checkbox (and the associated **T Rising** checkbox to specify the polarity) makes the USBee RX wait for the T input to have an edge before starting the pattern. This allows for synchronizing the [pattern generation with an external event.

The **Loop** checkbox allows you to loop the generated pattern. If the Wait for T checkbox is selected, each cycle waits for the T to be asserted before starting the generation.

The following screenshot shows the Digital Signal Generator configured to generate 291 samples each time the T signal has a rising edge. In this case the T signal is tied to the CH1 analog signal. The Loop selection makes it repeat, but only after the T signal sees another rising edge. Since the Digital Logic Threshold is set to 1.8V, the T transitions to logic 1 at 1.8V.



USBee Suite Pro - USBee RX Connected - C:\cwav\USBee Test Files\ProtocolSetup.usbeerx
File View Speed and Samples Trigger Setup Help
USBee RX Control Panel
Signal Setup
Current Levels
Signal Generator
Capture Once
Capture Many
Trigger Settings
Zoom
Pan
All
Cursors
X1
T
X2
USBee RX Suite
Logic Analyzer/Oscilloscope

Digital Logic Threshold 1.8

Oscilloscope Probes: CH1 [✓ 1X] 10X, CH1 Units V, CH1 Scale 1, CH1 Offset 0, +6V = 6.0V, 0V = 0.0V, -6V = -6.0V; CH2 [✓ 1X] 10X, CH2 Units V, CH2 Scale 1, CH2 Offset 0, +6V = 6.0V, 0V = 0.0V, -6V = -6.0V

Current Levels: CH1 1.36V, CH2 1.69V

Analog Generator On — Sine, Square, [✓ Tri], Ramp, DC — Frequency 424.4H, Min Volts 0.0V, Max Volts 2.3V

Digital Generator On — ...t Files\Ssamples.csv — Samples Loaded: 291 — ✓ Loop ✓ Wait For T ✓ T Rising — [✓ Sample Clock is 100000 sps] Sample Clock is external on C input

D0: Freq 390000, Duty Cycle % 86.9; D1–D7: Freq 1000, Duty Cycle % 50

Edge Count 0 (0, 1, 2, 3); Frequency 1899 (4), 1055 (5), 422 (6), 211 (7)

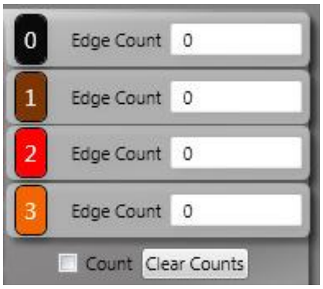
800us/div

1M samples@ 100Msps — Ready

T = 0ns
X1 = -2.04ms
X2 = 2.696ms
dX = 4.732ms
1/dX = 211.327Hz
Y1 = 1.83V
Y2 = 1.83V
dY = 0.00V
Frequency = infinity
Duty Cycle = n/a
Width = 0ns
Period = 0ns

USBEE RX EDGE/PULSE COUNTERS

The USBee RX can independently measure the number of edges that occur on four of the digital input signals (0 through 3). The number of edges counted is shown in the USBee RX Suite control panel as below:

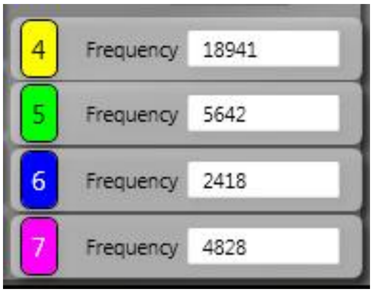


Counting occurs when the **Count** checkbox is selected. To clear the counts, press the **Clear Counts** button. Remember that the counts are affected by the **Digital Logic Threshold** setting, so you must set this correctly for the signals you are measuring for accurate counts.

Since each Pulse is made up of a rising edge and falling edge pair, you can compute the number of pulses by dividing the Number Of Edges by 2.

USBEE RX FREQUENCY COUNTERS

The USBee RX can independently measure the frequency four of the digital input signals (3 through 7). The frequency measured is shown in the USBee RX Suite control panel as below:



4 Frequency 18941
5 Frequency 5642
6 Frequency 2418
7 Frequency 4828

Remember that the measurements are affected by the **Digital Logic Threshold** setting, so you must set this correctly for the signals you are measuring for accurate counts.

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@interworldna.com

or

Call us at (877) 902-2979

Version 1.0