



The Black Box ToolKit

Serious about science: Serious about timing

The Black Box ToolKit T-Pad v1

User Guide



Credits:

Author: Dr. Richard R. Plant, C.Psychol, CSci, AFBPsS

Covers the following hardware:

The Black Box ToolKit T-Pad - USB Response Pad with Integral Timing

For the following platforms:

Microsoft Windows XP SP3, Vista SP2 (32/64), Windows 7 SP1 (32/64)
Windows 8 (32/64), Windows 8.1 (32/64), Windows 10 (32/64), Windows 11 (64)
Mac OS 8/9, OS-X
Linux 2.4 and greater

Contact details:

Phone: +44 (0)114 303 00 56

Fax: +44 (0)114 303 46 56

Email: info@blackboxtoolkit.com
support@blackboxtoolkit.com
sales@blackboxtoolkit.com

Web address: www.blackboxtoolkit.com

The Black Box ToolKit Ltd (BBTK) makes no warranty or representation, either express or implied, with respect to this manual, its quality, accuracy, merchantability, or fitness for a particular purpose. In no event will The Black Box ToolKit Ltd be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect or inaccuracy in this manual, even if advised of the possibility of such damage.

Copyright 2023-2025 The Black Box ToolKit Ltd. All rights reserved.

Contents

1. Introduction	6
2. Hardware Overview	10
2.1. Rear Connectors	13
2.2. 2x RJ45 Event marking TTL output triggers	14
2.3. Operating Modes	14
3. Software App for Configuring & Testing your T-Pad	15
4. Automatically Setting Opto-detector Sensitivity Using the App	17
5. Setting Opto Sensitivity for a Remote Tablet or PC Using the App	20
6. Setting Microphone/Voice Key Sensitivity Using the App	25
7. Refresh Blocking, Microphone Smoothing & Button Debouncing	31
7.1. Leading Edges: Stimulus and Response Onsets	31
7.2. Trailing Edges: Stimulus and Response Offsets	32
7.3. Accounting for Refresh Blocking, Microphone Smoothing & Button Debouncing in Your Experiment	33
7.4. Refresh Blocking, Microphone Smoothing & Button Debouncing Raw Data Correction Factors	34
7.5. Practical Examples of Application of Correction Factors	35
7.5.1. Opto Correction	35
7.5.2. Microphone Smoothing Correction	35
7.5.3. Button Debouncing Correction	36
7.5.4. Automatic Correction	37
8. Using the Sensor Check Utility Built Into the App	38
8.1. Checking Optos with the SCU	39
8.2. Checking More than One Opto	41
8.3. Checking Voice Keys	42
8.4. Checking Cross Modal Sensors	43
8.5. Checking Keyboard and Mouse Responses	43
9. Capturing Activity Data Using the T-Pad Configuration and Test App	45
10. Analyzing Captured Activity Using the T-Pad Data Analyzer	47
10.1. An In-Depth Look at Using the T-Pad Data Analyzer	52
10.2. Checking a Simple Reaction Time Paradigm	53
10.3. Additional Features of the Logic Analyzer	57
10.3.1. Shortcut Keys	57
10.3.2. Using the Pan Jog Dial	58
10.3.3. Producing a Summary Report with the Current Logic Analyzer Plot	58
10.3.4. Copying a Logic Analyzer Plot to the Clipboard	59
10.3.5. Exporting the Logic Analyzer Plot to a PNG file	59
10.3.6. Line Change Graphs: Quickly Visualizing Line Change Data	60
10.4. Advanced use of the T-Pad Data Analyzer: Automatically Timelocking Data Captured by an Experiment Generator such as Gorilla	61
11. Testing T-Pad Functionality and an Explanation of Each Available Mode	69
11.1. Testing in Mode 3 using the App – Serial VCP & Keyboard HID: Orange LED	69
11.2. Testing in Mode 3 using a serial terminal and a text editor – Serial VCP & Keyboard HID: Orange LED	71
11.3. Testing in Mode 1 using the App – Keyboard HID: Red LED	73
11.4. Testing in Mode 1 using a text editor – Keyboard HID: Red LED	75
11.5. Testing in Mode 2 – Serial VCP: Green LED using the App	76
11.6. Testing in Mode 2 using a serial terminal and a text editor – Serial VCP: Green LED	78

11.7. Testing in Mode 4 using the App – Keyboard HID & Serial VCP: Flashing Red LED	79
11.8. Testing in Mode 4 using a serial terminal and a text editor – Keyboard HID & Serial VCP: Flashing Red LED	81
11.9. Testing in Mode 5 using the App – Serial VCP: Flashing Green LED	84
11.10. Testing in Mode 5 using a serial terminal – Serial VCP: Flashing Green LED	85
11.11. Serial Transmission Time From the T-Pad to Your PC	87
11.12. Setting TTL Output Lines Using the T-Pad Configuration & Test App	88
11.13. Setting TTL Output Lines Using a Serial Terminal	89
11.14. Mode 6 BBTk USB TTL Module Emulation Mode	91
11.14.1. Testing in Mode 6 using the App – Serial VCP: Flashing Orange LED	92
11.14.2. Testing in Mode 6 Using a Serial Terminal – Serial VCP: Flashing Orange LED	97
11.14.3. Decoding Pairs of Hex Bytes Sent to and From the T-Pad in USB TTL Module Emulation Mode	98
12. Using the T-Pad in Your own Software or Experiment Generator	101
12.1. Option 1: Mode 1 Keyboard HID	101
12.2. Option 2: Mode 5 XiD Protocol Over Serial	102
13. Pseudocode Example of how to use the T-Pad with XiD Mode 5	104
14. PsychoPy Example Using the XiD protocol	105
15. Checking Firmware Version	117
16. How to Update Firmware in the T-Pad	118
17. Serial Command API	122
18. Mode Summary	125
18.1. Mode 1 – Keyboard HID: Red LED	125
18.2. Mode 2 – Serial VCP: Green LED	125
18.3. Mode 3 – Serial VCP & Keyboard HID: Orange LED	125
18.4. Mode 4 – Keyboard HID & Serial VCP: Flashing Red LED	126
18.5. Mode 5 – Serial VCP: Flashing Green LED	128
18.6. Mode 6 – USB TTL Module Emulator Serial VCP: Flashing Orange LED	128
18.6.1. Serial Input From the T-Pad	128
18.6.2. Serial Output to the T-Pad	129
18.6.3. Binary to Hex Decoding Chart	130
19. Specifications	132
19.1. Key Features	132
19.2. Rear Connectors and Pinouts	133
19.3. Timing Specifications	134

Warnings and Cautions

Intended Usage

Unless otherwise labelled, all products offered for sale by The Black Box ToolKit Ltd are for academic study and/or research use only.

The Black Box ToolKit Ltd, instruments, components, and accessories are designed for educational and research oriented life science applications and investigations. The Black Box ToolKit Ltd, does not condone the use of its instruments for clinical medical applications. Instruments, components, and accessories provided by The Black Box ToolKit Ltd, are not intended for the diagnosis, cure, mitigation, treatment, or prevention of disease.

Electrostatic Discharge Instructions



T-Pad system components that are sensitive to electrostatic discharge are marked with the ESD symbol. For these T-Pad components, users should not directly touch any internal pins and should adhere to the following precautions when connecting electrodes or adapters to these components. Users should always ground themselves before handling these T-Pad components. Users should then ground the T-Pad component before connecting to other equipment by touching the screw terminals of the component cable or by touching the metal shell of the component.

Note that the information in this manual is subject to change, without notice. The manufacturer declines responsibility for the safety, reliability, and performance of T-Pad system components if not used in compliance with T-Pad documentation.

1. Introduction

Our advanced USB serial response pad, the T-Pad, is designed to quickly and easily help you achieve better timing in your Psychology experiments as timestamps recorded are no longer tied to the software or experiment generator you are using. It has a hardware based timer that is completely separate from your computer and gives independent millisecond accurate timings.

All activity the response pad detects is streamed to your experiment along with an independent timestamp. That might be the onset, duration and offset of a visual stimulus image, a button press or a vocal response.



By using photodiodes, or opto-detectors, that attach to your screen the T-Pad can detect visual stimulus onsets as they happen in the real world, independently timestamp them and send them to your experiment over USB using a Virtual COM Port/serial port. It also detects responses on up to 10 built-in buttons and provides millisecond accurate reaction times (RT's).

Up to 10 external hand-held response buttons and foot pedals can be used to capture responses along with TTL trigger input and a voice key microphone.

A second USB connection on the T-Pad allows you to plug a lead into a USB port on your PC/Mac/Linux system for it to simultaneously appear as a second keyboard. This also works on any platform including Apple iPads, Android tablets and phones with an appropriate USB adapter. Press a button, activate an opto-detector or trigger a voice key and a keystroke will be sent to your experiment as if you had pressed a key on a standard keyboard with no need to recode.

Key features of the T-Pad:

- USB Response Pad with Integral Timing
- Up to 10x Built-in Buttons, Hand-held buttons or Foot Pedals
- 2x Opto-detectors/photodiodes
- 1x Microphone/Voice Key
- 13x TTL Out
- 1x TTL In
- 1x USB Keyboard HID
- 1x USB Serial VCP

An example of the types of external response options that can be plugged in are shown below:



Hand-held response buttons/triggers

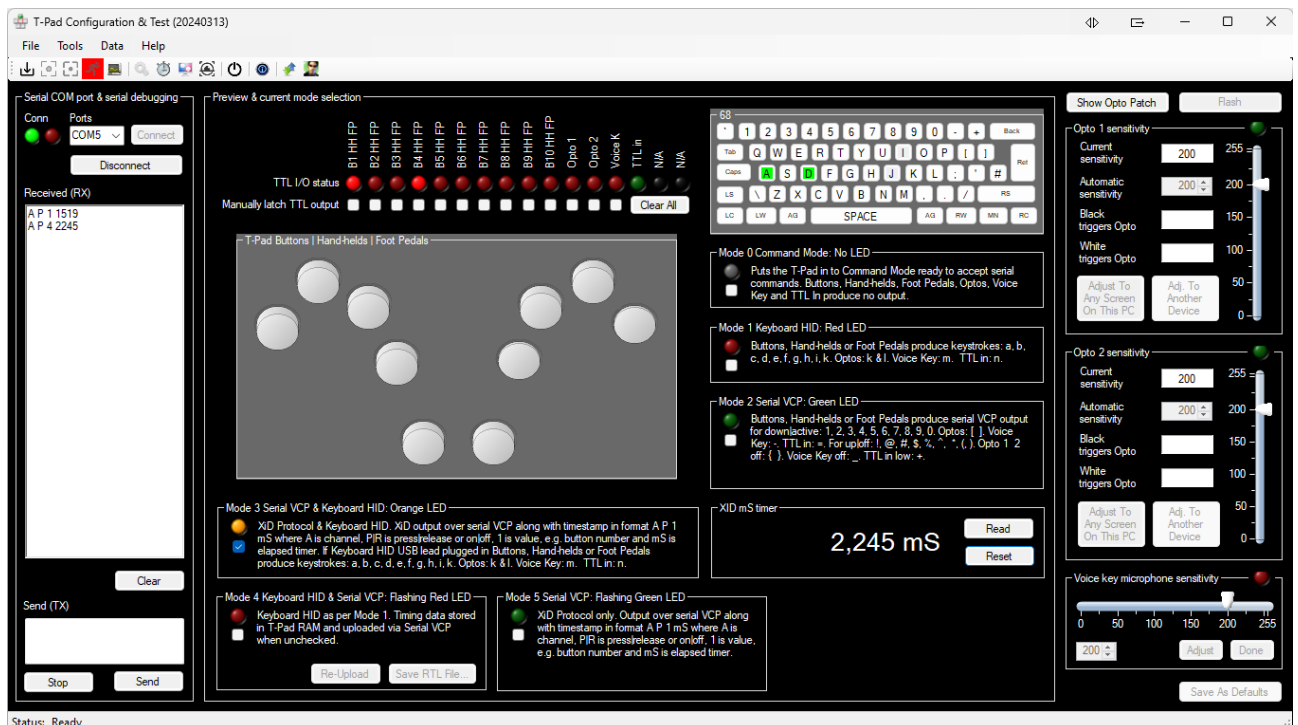
Thermoplastic rubber hand-held response button/trigger in black with single push button and 2 m cable. Terminated in mono 3.5 mm jack. Up to 10 hand-helds can be used with stereo splitters.



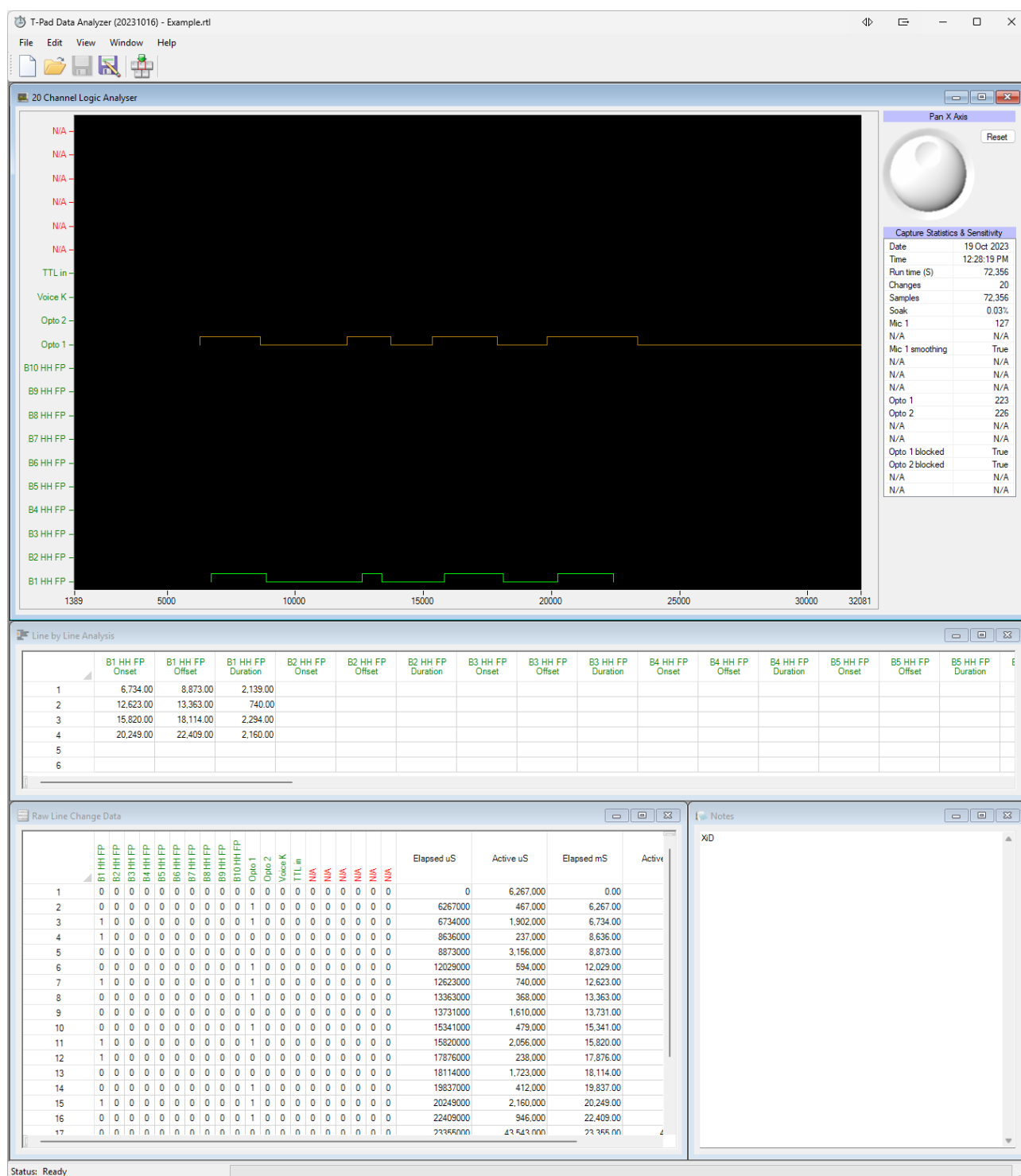
Foot switch pedal

Polarity switchable non-latching foot switch with a 1/4 inch jack connector which sends a make (down) or break (up) signal. A rubber grip ensures the pedal stays in place during use. 2 m cable terminated in mono 3.5 mm jack (via adapter). Up to 10 foot pedals can be used with stereo splitters.

The T-Pad comes with a partner Windows App that allows you to configure and test the T-Pad. What's more if you wish to use standard USB keyboard key presses our partner App enables you to quickly and easily integrate presentation and response measurements within your existing studies. The App will record all millisecond accurate timing data generated by your experiment as it runs. All you need to is concentrate on standard keyboard key presses and leave the heavy lifting to us!



Once your experiment has finished you can review the independent timing data collected by the T-Pad and use it alongside timing data collected by your own experiment as it ran. Our response pads aim is to let you add value to your experiments by collecting millisecond accurate timing data for presentations and responses without having to recode.



If you want to go deeper you can choose to read the streamed serial timing data directly into your own experiment using the simple T-Pad API. Examples are provided for popular experiment generators and programming languages.

What's more the T-Pad also features hardware based TTL trigger outputs which automatically event mark activity related to onsets and reaction times in the real world as they occur. The T-Pad is an easy to use fit and forget solution for TTL event marking/TTL triggering in psychology experiments that use EEG, fMRI, Eye Tracking or any study where you need to event mark with millisecond accuracy.

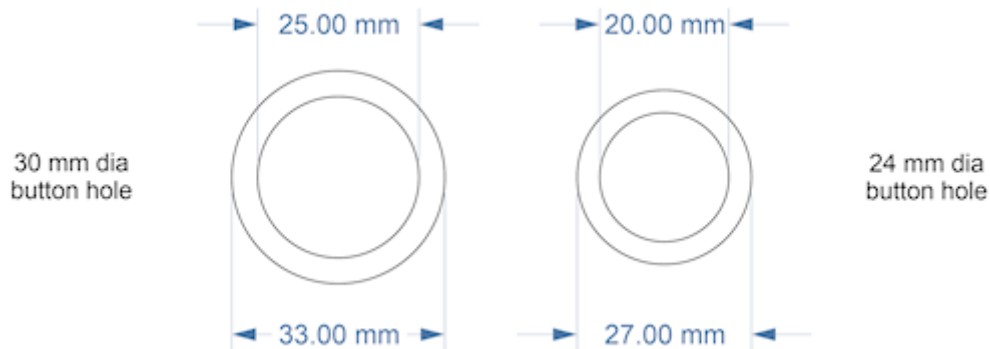
As an added bonus you can use the T-Pad to produce simple TTL event marks from your own studies. In short alongside automatically generated hardware TTL event marks you also have the option to generate them from your own experiments code at any point.

Finished in an attractive carbon fiber effect our response pad houses up to 10 buttons in a range of sizes and colors with the option to add tamper proof clear keycaps. If you opt for a custom laser cut solution you are free to design your own button layout to make your response pad truly bespoke to you.

The T-Pad's overriding aim is to reduce inconsistencies and improve your Response Time accuracy compared with using standard keyboards, mice or competitors devices.

2. Hardware Overview

The T-Pad is designed to be around the size of a paperback book and is manufactured using sleek carbon fiber effect plastic (250 mm x 160 mm x 35 mm LWH). It can house up to 10 inbuilt buttons in a range of colors. Buttons come in two sizes, 24 mm and 30 mm diameter and can be positioned where you choose with a custom laser cut layout.



Alternatively you can stick to one of our standard layouts. Tamper proof clear cap buttons are also available so that you can insert your own decals.

Alongside the inbuilt buttons up to 10 external momentary push-to-make buttons or foot pedals can be plugged in.



In addition to button inputs it has two opto-detectors/photodiodes which can be attached to the screen of your PC, tablet or phone to detect onsets, offsets and durations of visual stimuli with millisecond accuracy.

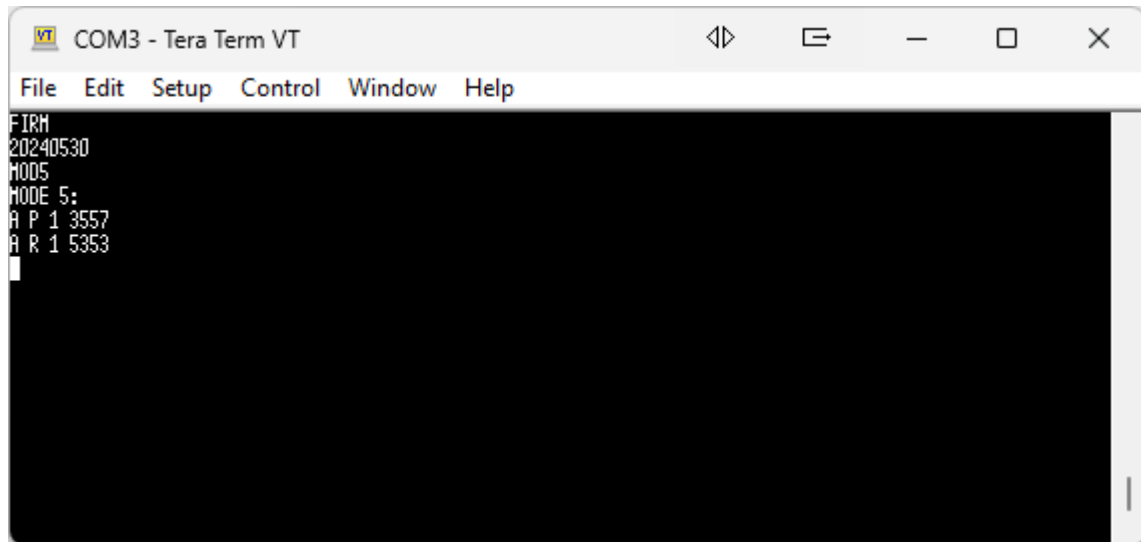
A microphone, or Voice Key, detects the onset, offset and duration of vocal responses once they pass a preset crossing-threshold for amplitude/volume. Finally a single TTL input can also be used to trigger the T-Pad.

Two USB connectors allow all activity to be sent to your PC, tablet or phone as either standard USB keyboard keystrokes or as serial over a serial/Virtual COM Port (VCP).

The first USB connection sends real-time event and timing data over serial in a range of formats to your experiment. The most commonly used format is the XiD Protocol. This industry standard protocol encodes the source of the activity, e.g. button, whether it was pressed on released and an independent millisecond accurate time stamp.

Running our own custom firmware the T-Pad can check for activity on any of its buttons or sensors between 3,205 to 5,263 times each second (3.24 to 5.26 kHz) whilst timestamping with millisecond accuracy. The T-Pad has an onboard RAM buffer and stores events with timestamps as they happen and streams them to your PC as quickly as possible in the order they occurred.

A XiD serial command sent to your experiment might look like the example shown below:



The first button press would decode as:

A P 1 3557

A	P	1	3557
Channel	Press Release	Sensor Number	Millisecond Timer
Buttons	Press	Button 1	3557 mS

A second USB connection on the T-Pad allows you to plug a lead into a USB port on your PC/Mac/Linux system and it will appear as a second keyboard. This also works on any platform including Apple iPads, Android tablets and phones with an appropriate USB adapter. Press a button, activate an opto-detector or trigger a voice key and a keystroke will be sent to your experiment as if you had pressed a key on a standard keyboard with no need to recode.

When running with the USB keyboard interface plugged in this operates at 1,000 Hz and can typically report a button presses etc. in 1 mS. Technically up to six events can be reported in a single USB packet.

The T-Pad has a total of six operating modes available to give you maximum flexibility with regards to how you use it in your own studies. These range from keyboard HID alone, XiD

serial as described above through to hybrid modes where both USB interfaces are active. Advanced modes even allow for all event and timing data to be stored on the T-Pad and uploaded when a study has finished.

A tricolor LED (red/green/orange) on the rear of the T-Pad indicates the mode it is currently running in. Modes can be cycled by pressing the Mode button on the rear of the unit or by sending MODx commands to the T-Pad over serial, i.e. MOD1, MOD2, MOD3, MOD4, MOD5 or MOD6. Mode 0 is Command Mode. To return to Mode 0 an X can be sent to the T-Pad at any time over serial. In this mode the LED will be unlit.

The T-Pad comes with a partner App which runs on Microsoft Windows and lets you test and configure it. Once configured all settings are stored in Non Volatile RAM (NVR) on the T-Pad itself and are recalled when either USB lead is plugged in. This means that once configured you shouldn't need to alter the settings which makes it easier to use with non PC platforms like iPads, Android tablets and phones. For example, once you have set the Opto-detector sensitivity using the App this is stored on the T-Pad so that when plugged into a non PC platform that sensitivity will be recalled so that a visual stimulus is detected correctly.

A full set of serial API commands are provided so that you can configure, check settings and use the T-Pad over serial from non PC platforms if you so wish. Our App uses this exact same serial API so nothing is hidden.

Shown below is a summary of the connectors/interface options available on the rear of the T-Pad together with pinouts for TTL event marking via two 8-bit RJ45 sockets. Finally the six modes of operation are described.

Full technical specifications and serial API documentation can be found at the end of this manual.

2.1. Rear Connectors



Connector	What does it do	How can I use it
Mic (3.5 mm stereo socket)	Allows you to plug-in the BBTK headset with condenser microphone or your own desktop or lapel mic.	Used to accept a vocal response rather than a button press within your experiment. Operates as a simple voice key with a crossing threshold for activation.
Opto (3.5 mm stereo socket)	Allows you to plug-in up to 2x BBTK opto-detectors, light sensors or photodiodes for detecting visual stimulus onsets or visual event markers.	Used for detecting and timestamping image onsets, durations and offsets. Can also be used for detecting visual event markers, e.g. black to white block transitions. Automatically produces sub-millisecond hardware TTL event markers or event triggers.
But 1/6 (3.5 mm stereo socket)	Up to 2x external momentary or push-to-make buttons.	Allows you to use up to 10x external BBTK handhelds, foot pedals or external buttons in total. These mirror the built-in buttons if present when pressed.
But 2/7 (3.5 mm stereo socket)	Up to 2x external momentary or push-to-make buttons.	
But 3/8 (3.5 mm stereo socket)	Up to 2x external momentary or push-to-make buttons.	
But 4/9 (3.5 mm stereo socket)	Up to 2x external momentary or push-to-make buttons.	
But 5/10 (3.5 mm stereo socket)	Up to 2x external momentary or push-to-make buttons.	
Serial (USB 'B' square socket to PC 'A' flat)	Connects to the PC you want to stream serial timing and event data to. Uses an industry standard FTDI Virtual COM Port (VCP) at 115,200.	Streams serial data in an easy to understand XiD compatible format which represents, light sensor events, button presses, voice key activation and TTL inputs together with an independent hardware based millisecond timer.
Keyboard (USB 'B' square socket to PC 'A' flat)	Connects to the PC you want to accept keyboard responses on.	Each button sends a standard key down response as might a normal keyboard if you pressed keys "a" through "j" for example. Opto or light sensor 1 would produce a "k" and Opto 2 a "l". Voice key activation would produce an "m" and TTL In produces an "n".
TTL I/O 1 (RJ45)	TTL 0/5V Digital I/O.	Independent hardware based TTL triggers for event marking via an 8-bit port.
TTL I/O 2 (RJ45)	TTL 0/5V Digital I/O.	Independent hardware based TTL triggers for event marking via an 8-bit port.
Mode LED	Indicates what mode the T-pad is operating in.	Shows which mode the T-Pad is in when set from PC over serial or when the mode button is pressed.
Mode Button	Allows you to select mode by pressing it.	Cycles through the available modes. Each time the button is pressed the LED will change color to indicate the current mode.

2.2. 2x RJ45 Event Marking TTL Output Triggers

RJ45 – TTL I/O 1			RJ45 – TTL I/O 2		
Pin	Button/Sensor	TTL Line	Pin	Button/Sensor	TTL Line
1	B1	TTL Out 1	1	B5	TTL Out 5
2	B2	TTL Out 2	2	B6	TTL Out 6
3	B3	TTL Out 3	3	B7	TTL Out 7
4	B4	TTL Out 4	4	B8	TTL Out 8
5	Opto 1	TTL Out 11	5	B9	TTL Out 9
6	Opto 2	TTL Out 12	6	B10	TTL Out 10
7	Voice Key	TTL Out 13	7	TTL In 1	TTL In 1
8	Ground	N/A	8	Ground	N/A

2.3. Operating Modes

Mode	Mode type	LED on rear	USB interface	What is output and over which interface
0	Command	No LED	Serial	Puts the T-Pad in to Command Mode ready to accept serial commands. Buttons, hand-helds, foot pedals, optos, voice key and TTL In deactivated.
1	Keyboard	Red	Keyboard	Buttons, hand-helds or foot pedals produce keystrokes: a, b, c, d, e, f, g, h, i, k. Optos: k & l. Voice key: m. TTL in: n.
2	Serial	Green	Serial	Buttons, hand-helds or foot pedals produce serial VCP output for down active: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. Optos: [&]. Voice key: -. TTL in: =. For up off: !, @, #, \$, %, ^, & *, (,). Opto 1 & 2 off: { & }. Voice key off: . TTL in low: +.
3	Serial & keyboard	Orange	Both	XiD protocol & keyboard HID. XiD output over serial VCP along with timestamp in format A P 1 mS where A is channel, P R is press release or on off, 1 is value, e.g. button number and mS is elapsed timer. If keyboard HID USB lead plugged in buttons, hand-helds or foot pedals produce keystrokes: a, b, c, d, e, f, g, h, i, k. Optos: k & l. Voice Key: m. TTL in: n.
4	Serial & keyboard	Flashing Red	Both	Keyboard HID as per Mode 1. Timing data stored in T-Pad RAM and uploaded via Serial VCP when your experiment has ended.
5	Serial	Flashing green	Serial	XiD Protocol only. Output over serial VCP along with timestamp in format A P 1 mS where A is channel, P R is press release or on off, 1 is value, e.g. button number and mS is elapsed timer.
6	Serial	Flashing orange	Serial	In this mode the T-Pad emulates a BBTK USB TTL Module and allows you to send TTL event marks using two serial bytes to create between 1 and 255 separate event marks (8 bits).

3. Software App for Configuring & Testing your T-Pad

Our partner PC App allows you to configure and test your T-Pad with ease. Simply connect both USB leads and you can preview timing data as well as the keyboard HID keystrokes that will be sent to your experiment.

Ensure both USB leads are connected directly to USB sockets on your motherboard. You are advised not to use USB Hubs or Switches as they can introduce latencies. Also you should use the high quality USB leads that shipped with the T-Pad. The USB specifications specify that USB cables should be no longer than a maximum of 5m.

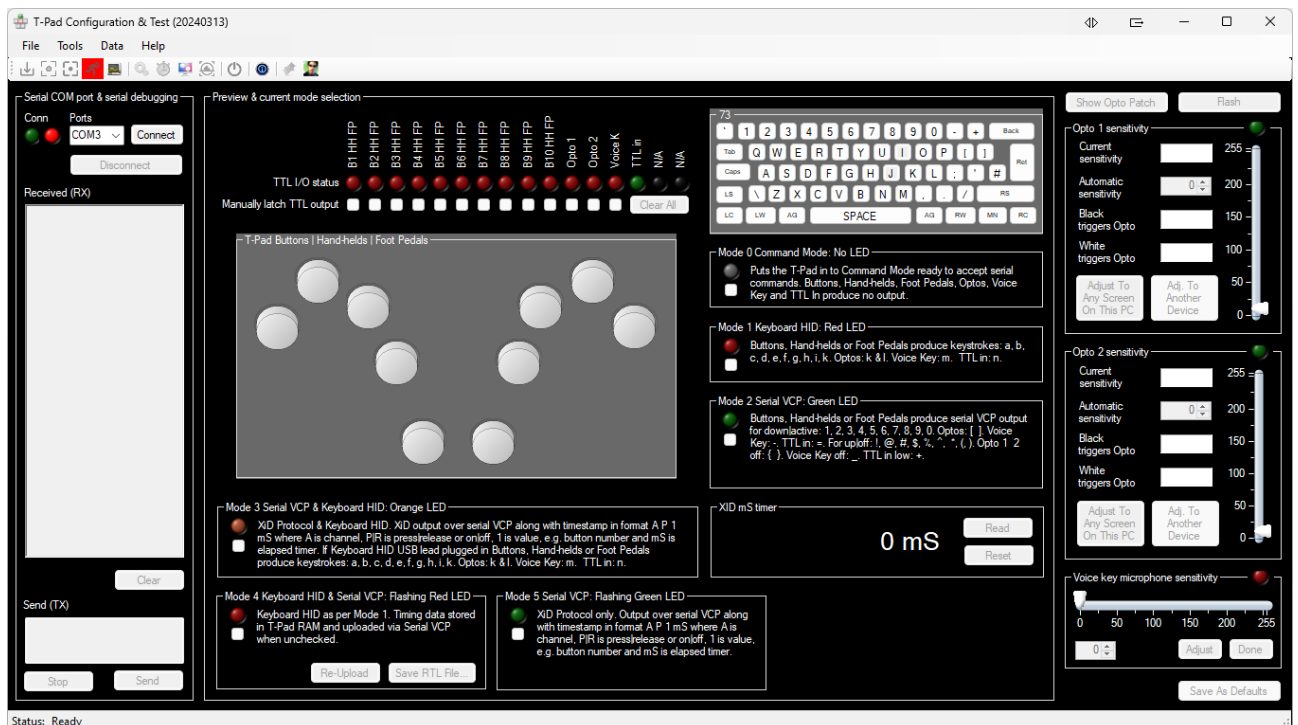
In addition to allowing you to test that everything is connected and working correctly it also helps you automatically set Opto-detector sensitivity for both optos.

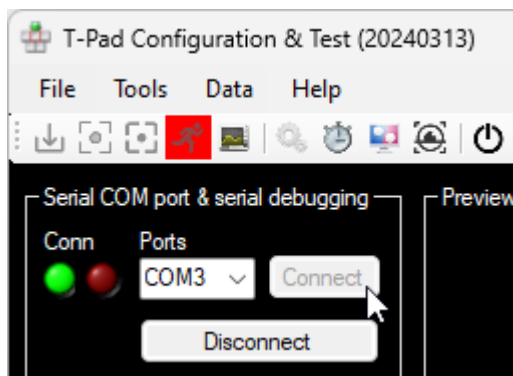
Using the same App you can manually set the Voice Key microphone sensitivity. This cannot be done automatically as volume is more subjective and you need to set the crossing threshold where the Voice Key triggers according to your own perception of the volume level.

Initially when you start to use your T-Pad your first task is to set the Opto-detector and Voice Key microphone sensitivity. Once set you can save the settings to the T-Pad as default values that are loaded each time it is powered up when either USB lead is connected.

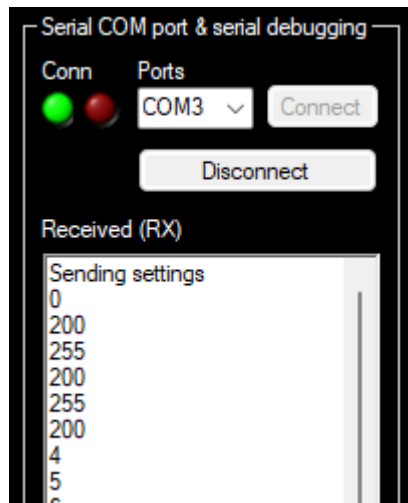
Once started the App will be shown disconnected from the T-Pad with a virtual red LED shown top left.

Note that your T-Pad may not have 10 buttons installed and your button sizes and layout may differ from the preview presented.





Begin by selecting the VCP Serial Port your T-Pad is connected to and then click on the Connect button.



If you have successfully connected to the T-Pad the LED will turn green and some settings data may flash past. If you can't connect first time try Disconnecting and reconnecting.

If you have successfully connected to the T-Pad you are now ready to set your Opto-detector sensitivity.

4. Automatically Setting Opto-detector Sensitivity Using the App

The goal of setting the Opto-detector sensitivity is to set the level at which a white image, i.e. a white Opto-detector marker, triggers the T-Pad to realize that a stimulus image has been displayed. The same sensitivity setting also needs to clearly distinguish when a black Opto-detector marker indicates a stimulus image has been removed from your display device.

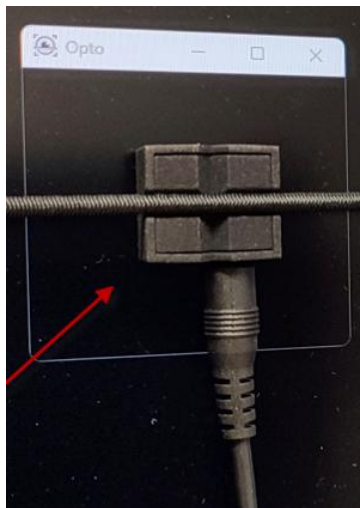
Setting the correct sensitivity level means that the T-Pad can detect a stimulus image being displayed as quickly as possible in the real world.

Before you can use the T-Pad to detect the onset, duration and offset of visual stimuli it is crucial that you set the Opto-detector(s) sensitivity correctly.

It is critical that you set the Opto-detector(s) sensitivity correctly so that stimulus timing data is accurate. You must set the Opto sensitivity for each display device you use or if you change the brightness or contrast on a device you have used previously.

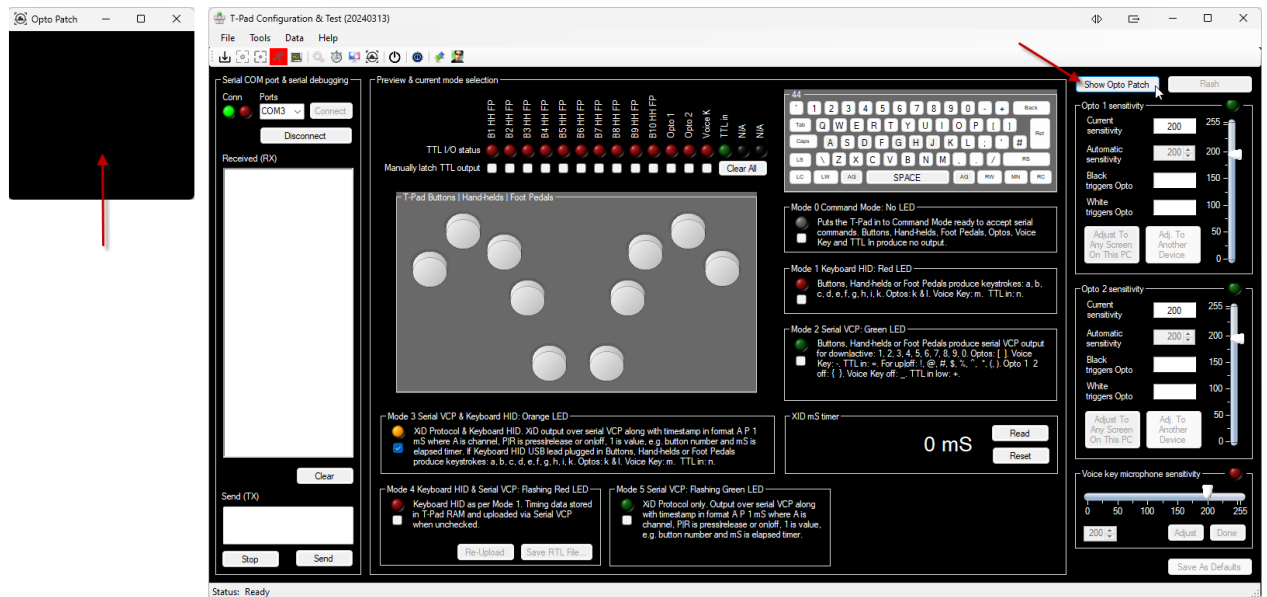
Once set you can save the sensitivity to the T-Pad so that it is used on start-up. Settings will not be lost when powered off, i.e. USB leads unplugged as they are stored in Non Volatile RAM (NVR).

To start position the Opto on-screen where you will flash up the visual stimulus marker, e.g. as shown below. Depending on your T-Pad the Opto-detector may look like the one shown below or may be a custom model.

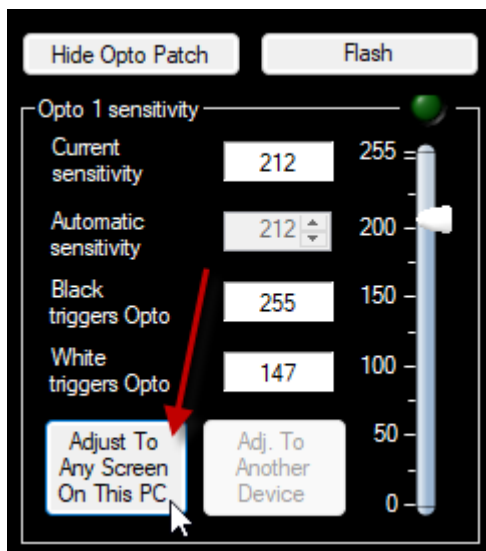


In this example the Opto is held in place with a bungee cord that straps around the monitor.

Next click on the Show Opto Patch button to display the Opto patch and move the patch window so that it is under your physical Opto-detector, or vice-versa.

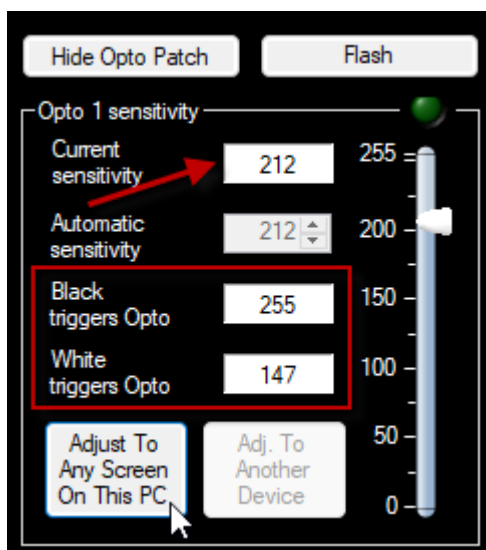


To start auto-calibrating the Opto sensitivity click on the Adjust To Any Screen On This PC button.

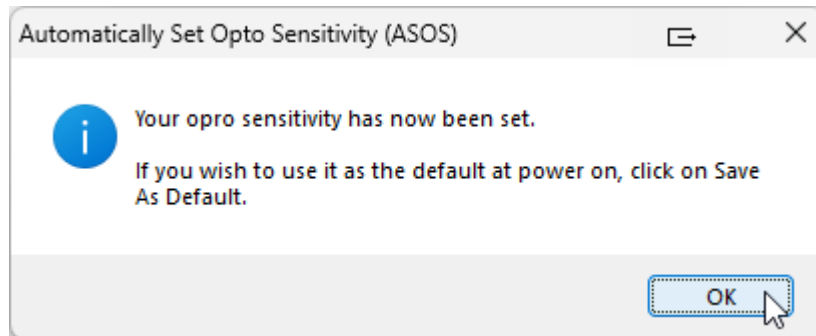


The Opto patch will turn black and the indicator will raise until the Opto-detector triggers on black.

Next a white patch will be shown and the white sensitivity will be lowered until the Opto-detector is not triggered.



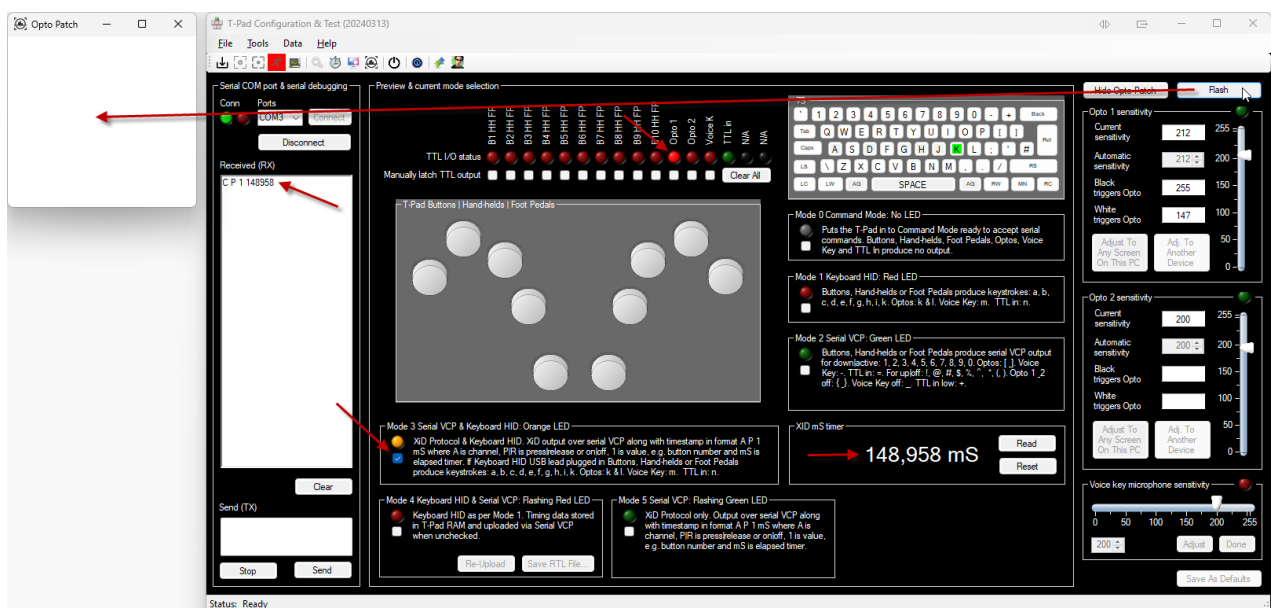
Once levels for black (off non-active) and white (the trigger) have been determined the best sensitivity will be automatically set. This ensures that timing accuracy for detecting visual stimulus onsets, durations and offsets will be optimal.



To save the sensitivity settings into the T-Pad click on the Save As Defaults button. These will then be used the next time the T-Pad is powered on.

To test the Opto sensitivity leave the Opto-detector in place and click on the Flash button and hold it down. As you do this the Opto patch will turn white and stay latched white until you release the mouse button.

To flash the Opto patch you can also click anywhere in the Opto patch window. Whilst you hold the mouse button down the patch will remain white and when you release it will turn black.



5. Setting Opto Sensitivity for a Remote Tablet or PC Using the App

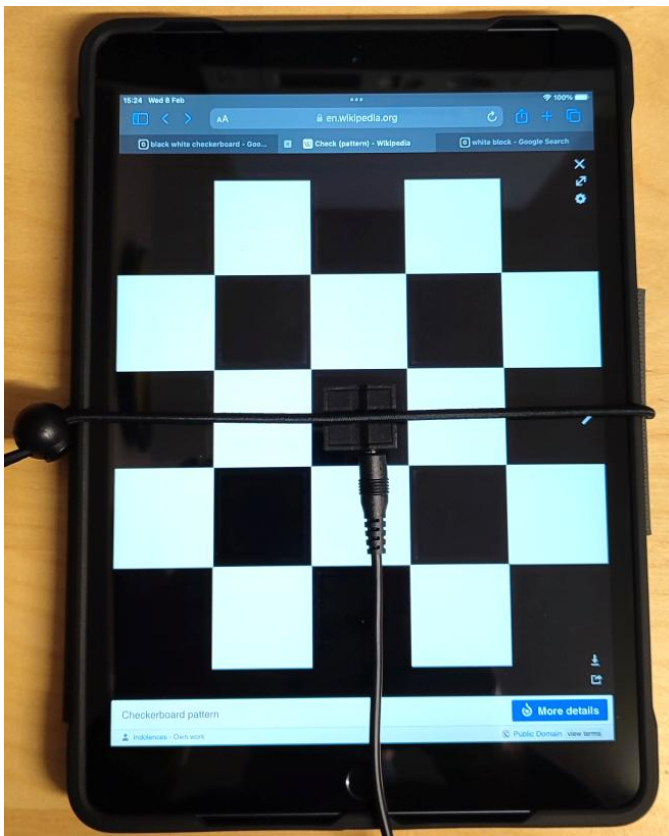
If you are using a second remote PC without the T-Pad Configuration & Test App installed or other display device/tablet/phone you will need to set the opto-detector sensitivity tailored to that screen.

In this example we will walkthrough setting the opto sensitivity using an iPad.

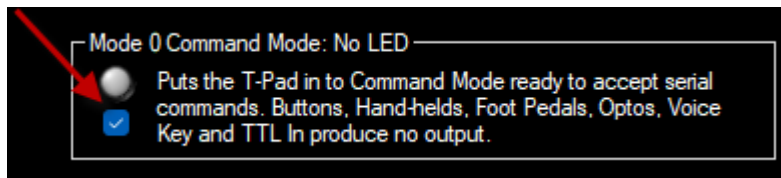
Before you begin you will need a pure black and white patch/square to display on your device to mimic an Opto marker patch in your experiment.

You can search for: “black white checkerboard” to find one or visit Wikipedia using a web browser:

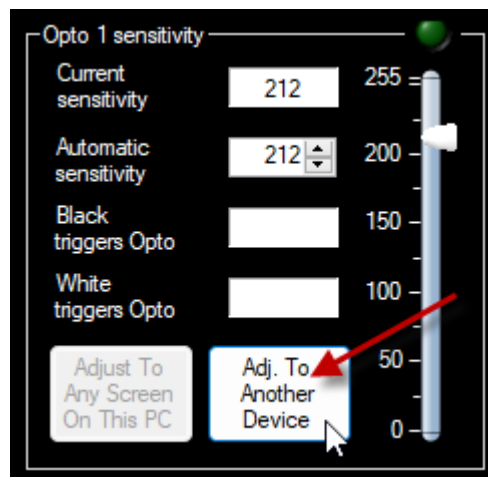
https://en.wikipedia.org/wiki/Check_%28pattern%29#/media/File:Checkerboard_pattern.svg



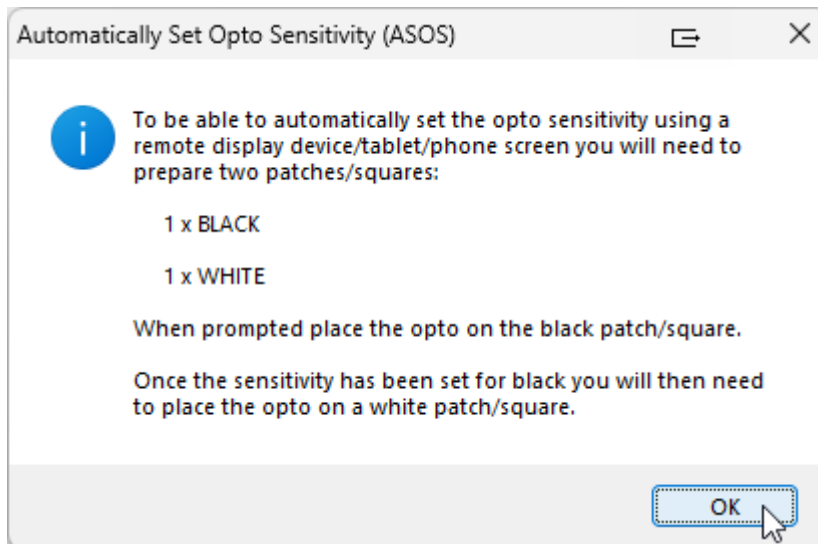
Before you begin place the Opto-detector over a black patch on the device you wish to calibrate.



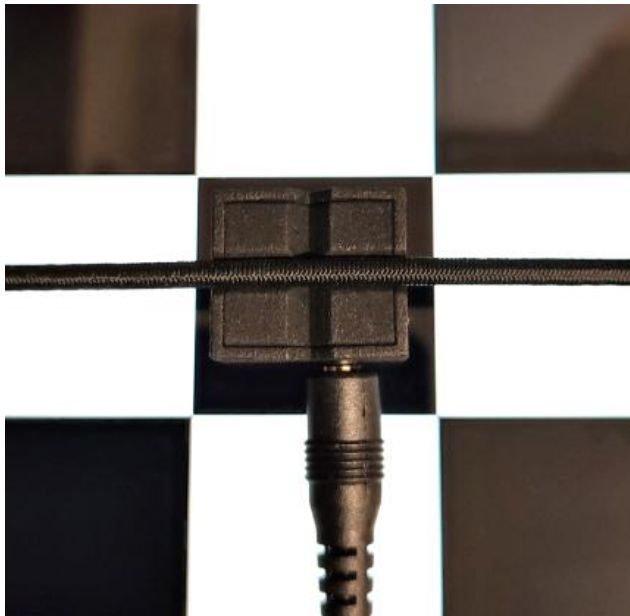
In the App ensure that you are connected to the T-Pad and that no virtual Mode LEDs are lit on screen. The physical LED on the rear of the T-Pad should also be unlit. If any of the Mode LEDs are lit or checkboxes ticked click on the Mode 0 tick box to return to command mode.



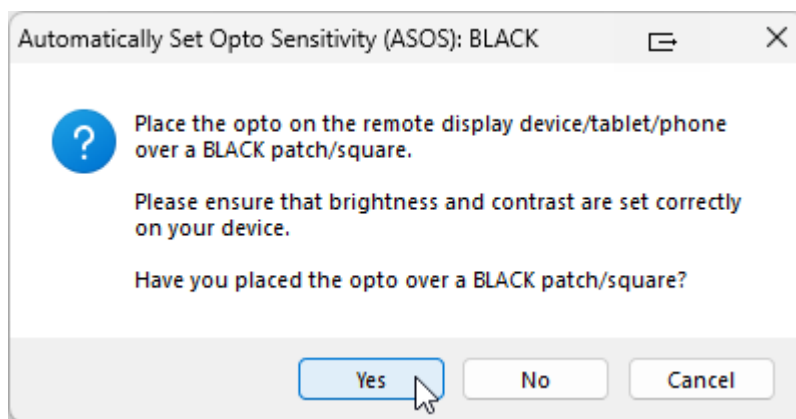
Once you are happy with your placement click on the Adj. To Another Device button.

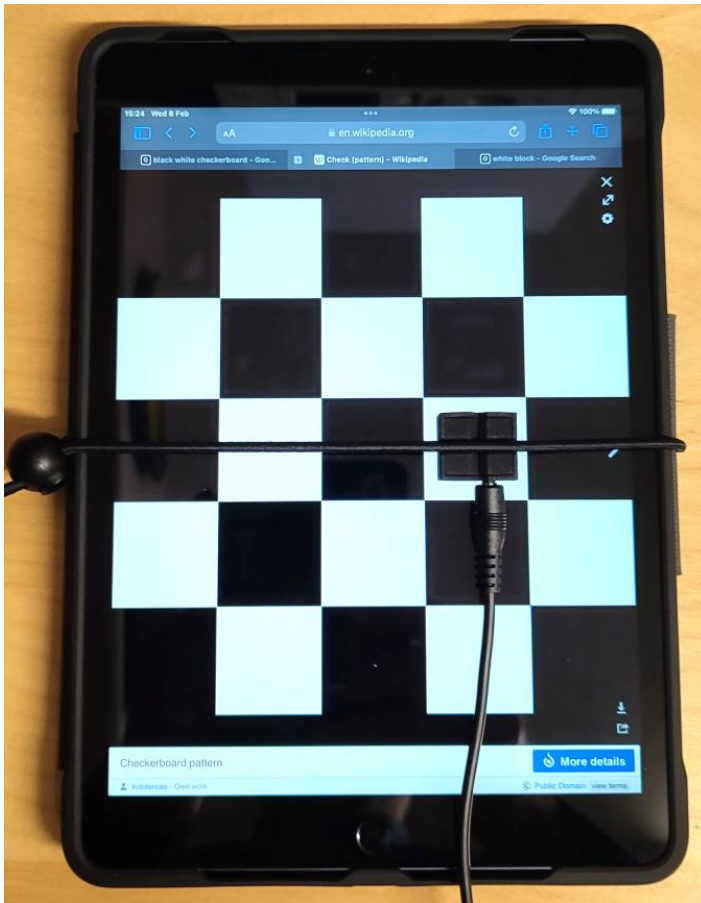


You will be reminded that you will need a black and a white patch on your devices screen in order to set each Optos sensitivity.

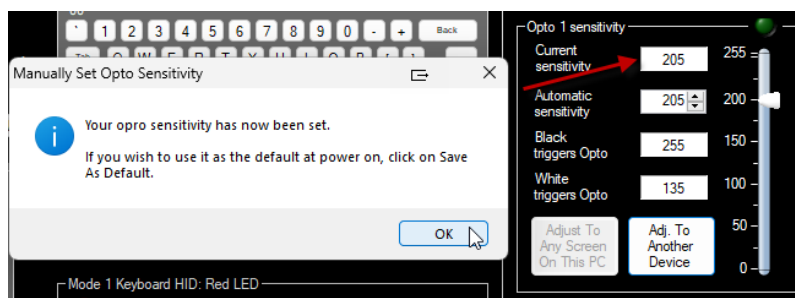
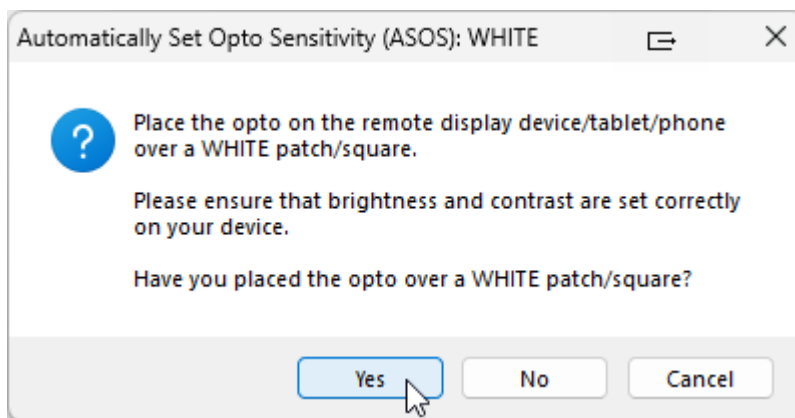


When prompted ensure that your Opto is placed over a black patch and then click on Yes.





Once the black levels are set you will be prompted to move the Opto on to a white patch/square.



In the “Current sensitivity” box the actual sensitivity will be displayed.

If you wish to save the Opto value click on the Save As Defaults button.

If you save the sensitivity this will be the default for that Opto each time the T-Pad is powered on.

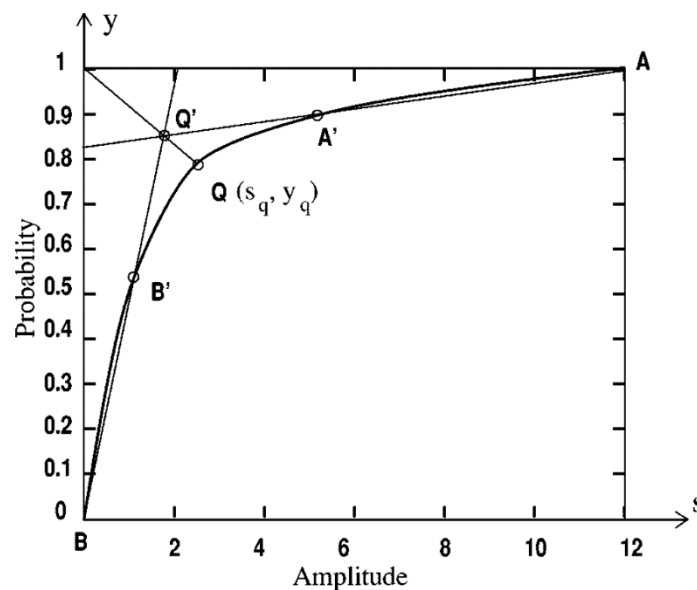
Remember that these Opto sensitivity values only apply to the device screen which you have calibrated them to and are not a global setting for all displays.

If you use a new display device, or change the brightness or contrast, you should consider rerunning the ASOS module again.

6. Setting Microphone/Voice Key Sensitivity Using the App

Before using the Voice Key component of your T-Pad you need to set the threshold at which it activates once the microphone being used reaches a given amplitude.

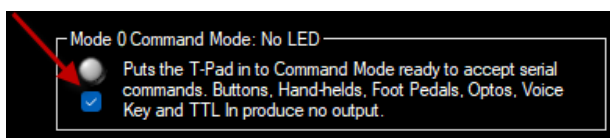
For example, you may need to adjust the activation point, or crossing threshold, for a condenser microphone so that it just triggers for the volume level produced by your own sound outputs. Put simply the T-Pad needs to know at which point on an analogue waveform it needs to trigger at. In the sample waveform shown below, trigger point Q' would need to be reached before the T-Pad registers an auditory stimulus. It is these points you are setting when you alter the microphone sensitivity on the T-Pad.



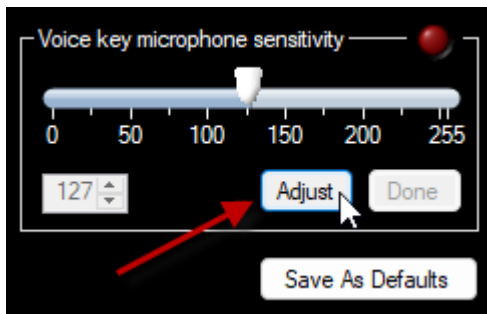
Note that the T-Pad doesn't record a sound file, it simply triggers once a set threshold has been reached and keeps triggering whilst that threshold is surpassed.

Unfortunately due to the complexity of sound waves and the fact that you could be using any one of many condenser microphones available, it would be impossible to construct an automatic method for setting the Voice Key sensitivity. This means that you will need to manually set and save the sensitivity.

In the App ensure that you are connected to the T-Pad and that no virtual Mode LEDs are lit on screen. The physical LED on the rear of the T-Pad should also be unlit.



If any of the Mode LEDs are lit or checkboxes ticked click on the Mode 0 tick box to return to command mode.

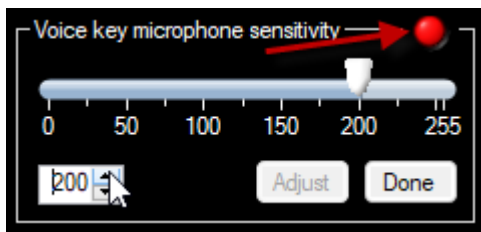


By default the Voice Key microphone threshold will be set at the midpoint of 127.

To adjust the sensitivity of the microphone click on the Adjust button.

Make a constant hum with your throat/mouth at about the amplitude you wish the Voice Key to trigger at. At the same time click on the numeric spinners and increase the value until the red virtual LED reliably illuminates.

Note that the sensitivity you have just dialled in is the level at which the mic picks up sound that is loud enough to pass the crossing-threshold to trigger the Voice Key.

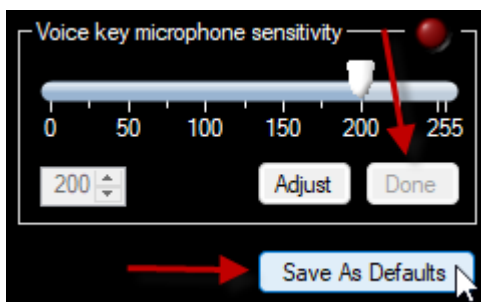


In the example opposite the LED has lit at a sensitivity of 200.

The LED basically shows you when the microphone starts to pick up a sound and can trigger.

Generally you are advised to set the actual threshold somewhere between the value where the numeric spinner first picks up a sound from the microphone and the maximum possible, i.e. 255.

So in this case we might set the sensitivity at $(255-200)/2 + 200 = 227.5$ or 228 when rounded.

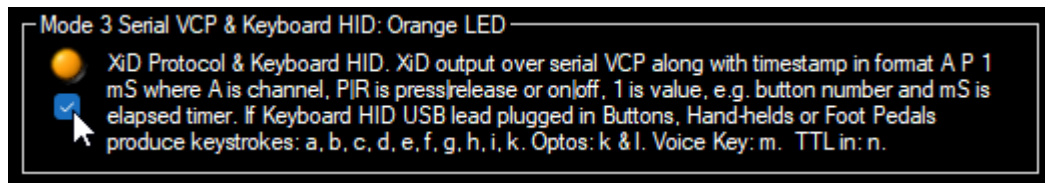


Once you are happy with your sensitivity click on Done so that the mic is no longer being adjusted and constantly checked and then on Save As Defaults.

Note this will also save the opto thresholds that were previously loaded.

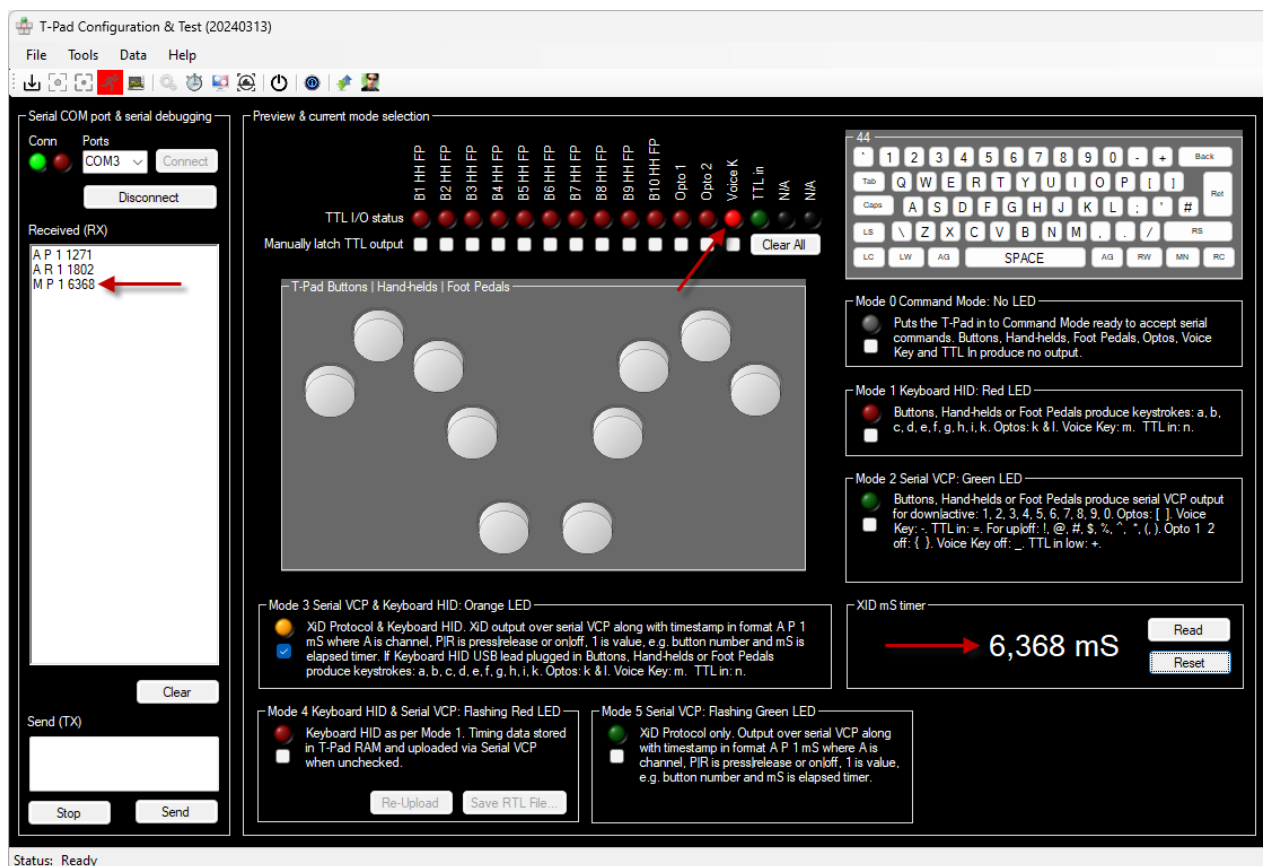
To test your Voice Key threshold is appropriately set you need to switch the T-Pad into one of the three available modes and test it. For this we recommend Mode 3: XiD protocol.

Do this by clicking on the Mode 3 checkbox.



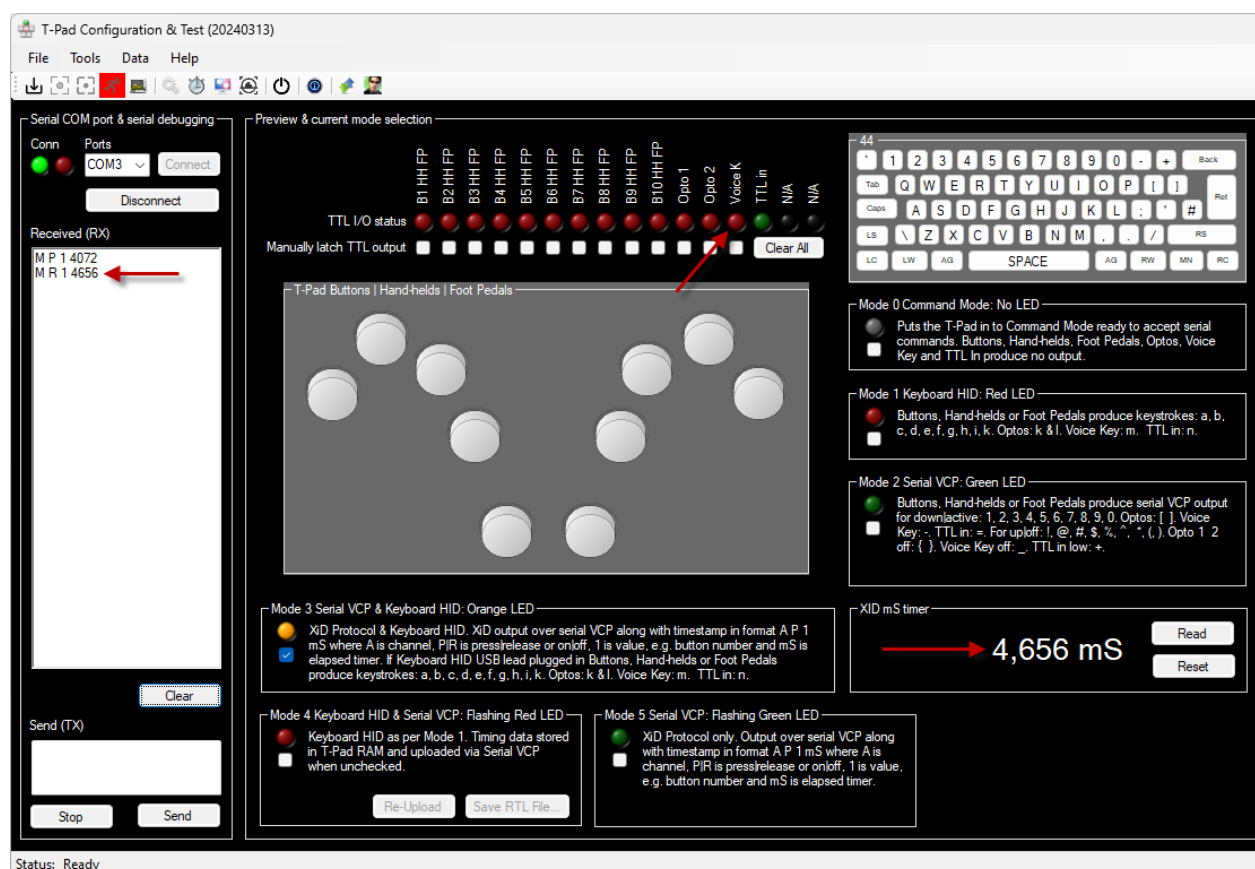
Check that you are in XiD protocol mode by pressing a response pad button. Note button 1 has been pressed and released.

Now make a sound at the sort of level you wish the Voice Key microphone to trigger at. You should now see the microphone produce an M P, Microphone Pressed XiD command followed by the mic number and an elapsed millisecond timestamp.



Keep making the sound to that the threshold you have set remained passed and the Voice Key Activated.

When the amplitude of the sounds falls below the threshold, i.e. the crossing-threshold you have set the Voice Key to, it will deactivate with a Release together with an elapsed millisecond timestamp.



As you can appreciate setting the crossing-threshold on Voice Keys can be tricky and will require some experimentation.

By way of some background in certain psychology experiments Voice Keys are used for recording responses and measuring reaction times. As outlined these operate on the basis of "crossing thresholds" or the number of times an auditory signal reaches or crosses a certain set threshold. The trigger threshold for a Voice Key is also susceptible to whether fricatives, plosives, voiced or unvoiced sounds are made in order to trigger it. That is, the type of sounds, or phonemes, a participant produces.

When using other third party Voice Keys often the researcher is unaware of what this threshold actually is and whether the setting is the same as one used previously. As with other response devices the electronics and the interface used can vary considerably from device to device which can add 10's if not 100's of milliseconds to actual response times.

Without independently checking trigger thresholds it is often hard to determine exactly what is happening. Poorly calibrated Voice Keys can account for a significant amount of variation within your experiment. Our highly sensitive Black Box ToolKit v3 and its digital microphone can help you calibrate you own Voice Keys for use in your own experiments.

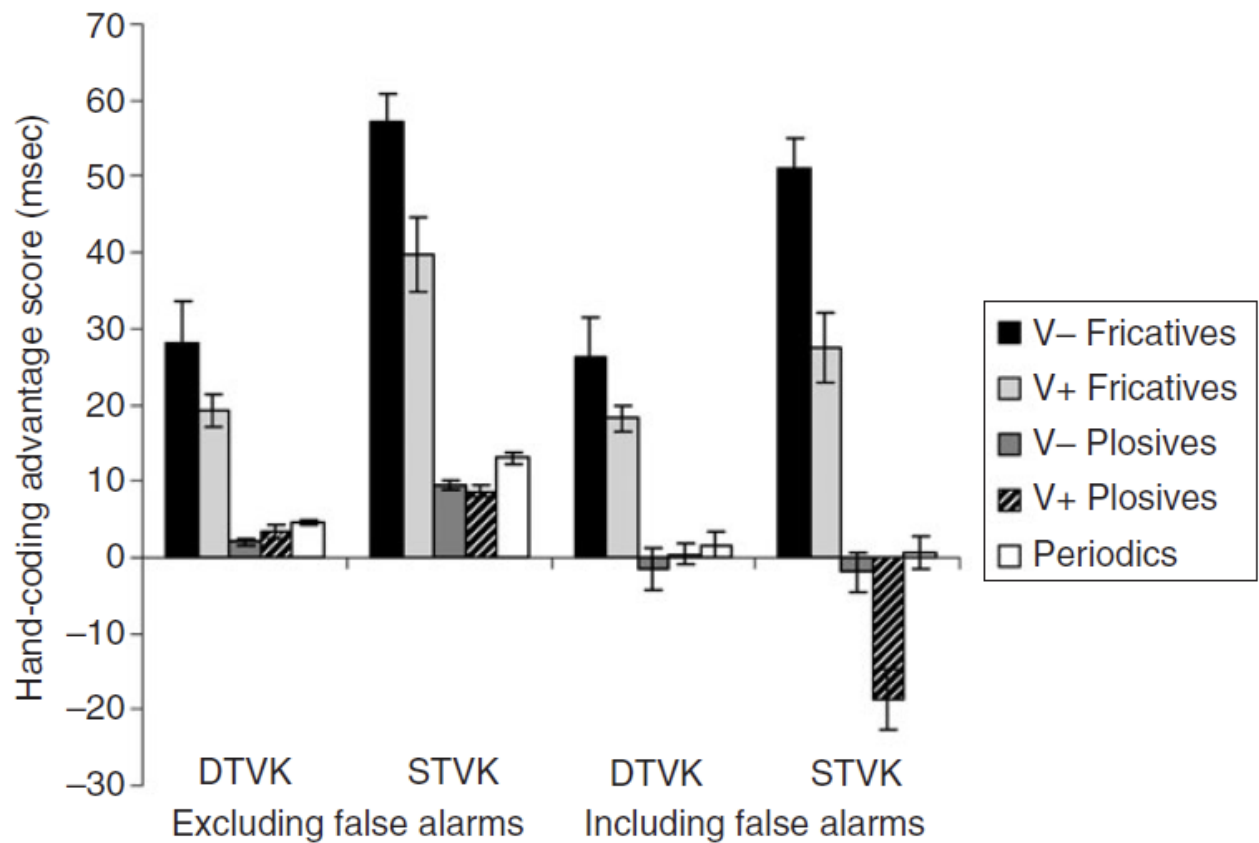
For a more in-depth discussion on Voice Keys and the magnitude of error they can introduce a good starting point are the two papers listed below:

Phonetic Biases in VoiceKey Response Time Measurements, Brett Kessler, Rebecca Treiman, John Mullennix, Journal of Memory and Language, Volume 47, Issue 1, July 2002, Pages 145-171,
<https://www.sciencedirect.com/science/article/abs/pii/S0749596X01928359>

Abstract: Voice response time (RT) measurements from 4 large-scale studies of oral reading of English monosyllables were analyzed for evidence that voice key measurements are biased by the leading phonemes of the response. Words with different initial phonemes did have significantly different RTs. This effect persisted after contributions of nine covariables, such as frequency, length, and spelling consistency, were factored out, as well as when variance associated with error rate was factored out. A breakdown by phoneme showed that voiceless, posterior, and obstruent consonants were detected later than others. The second phonemes of the words also had an effect on RT: Words with high or front vowels were detected later. Phoneme-based biases due to voice keys were large (range about 100 ms) and pervasive enough to cause concern in interpreting voice RT measurements. Techniques are discussed for minimizing the impact of these biases.

The delayed trigger voice key: An improved analogue voice key for psycholinguistic research, Michael D. Tyler, Leigh Tyler & Denis K. Burnham, Behavior Research Methods volume 37, pages139–147 (2005), <https://link.springer.com/article/10.3758/BF03206408>

Abstract: Many researchers rely on analogue voice keys for psycholinguistic research. However, the triggering of traditional simple threshold voice keys (STVKs) is delayed after response onset, and the delay duration may vary depending on initial phoneme type. The delayed trigger voice key (DTVK), a standalone electronic device that incorporates an additional minimum signal duration parameter, is described and validated in two experiments. In Experiment 1, recorded responses from a nonword naming task were presented to the DTVK and an STVK. As compared with hand-coded reaction times from visual inspection of the waveform, the DTVK was more accurate than the STVK, overall and across initial phoneme type. Rastle and Davis (2002) showed that an STVK more accurately detected an initial [s] when it was followed by a vowel than when followed by a consonant. Participants' responses from that study were presented to the DTVK in Experiment 2, and accuracy was equivalent for initial [s] in vowel and consonant contexts. Details for the construction of the DTVK are provided.



Typical timing error in standard voice keys (STVK) versus hand coding the recorded waveform (adapted from Tyler et al, 2005)

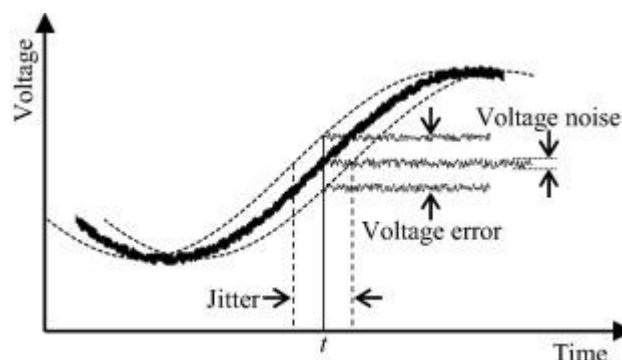
7. Refresh Blocking, Microphone Smoothing & Button Debouncing

7.1. Leading Edges: Stimulus and Response Onsets

In most psychology studies we are interested in the leading edge of an action relative to a stimulus, i.e. the Response Time or Reaction Time (RT). For example, the time between an image or sound being presented and a response being made by pressing a button down. In this example there are two leading edges, the start of the stimulus image and the start of the button down response. Generally the RT is the time between the image onset leading edge and the leading edge of the button down.

Detecting the onset or leading edges of both stimuli and responses is critical to accurate RT measurement. In the real world all signals have a degree of jitter in them when we measure them very accurately. This is why we set the sensitivity of both Opto's and Voice Key to be as high as possible so as to detect the onset or leading edge of a stimulus as quickly and as accurately as possible.

In the example below we can see that all signals have a degree of jitter in them. The amount of jitter in this case is contained between the two dotted lines. The solid line is where we might set the sensitivity or activation threshold. Ideally we would set the sensitivity to be as far left as possible relative to the onset of the jitter.



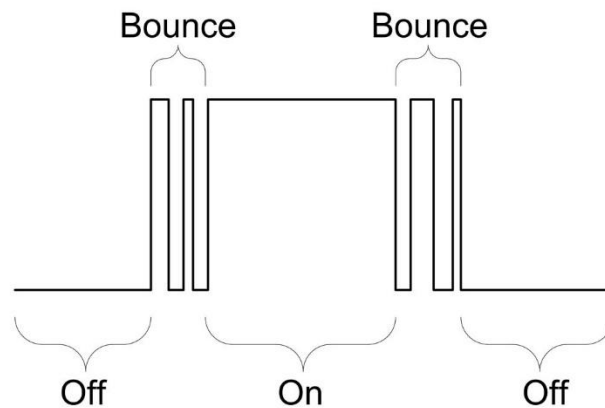
From: <https://www.sciencedirect.com/science/article/abs/pii/S026322411831145X>

At an electronics level detecting the onset or leading edge of a visual or auditory stimulus are similar.

These two examples provide an illustration of how at an electronics level we determine the onset or leading edges of a stimulus and responses. This is very similar to how a human would perceive the start of a visual or auditory stimulus for example.

Button presses are slightly different in the respect that when you press any button the signal you get from them is typically noisy due to the fact they bounce when pressed. Simply put when you press your finger on a button the electro-mechanical contacts typically bounce when you bottom out the switch. A good analogy is if you dropped a coin onto a desktop you would see and hear it bounce before it remained silent and static on the desktop.

Generally for button presses we take the first onset or leading edge of the signal as the response. But as the diagram below shows the button signal can bounce.



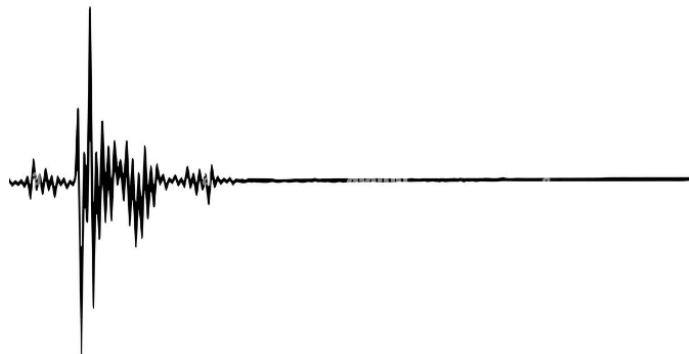
From: <https://stackoverflow.com/questions/20574040/pushbutton-debounce>

If we took the leftmost bounce as the on signal, i.e. onset or leading edge for the button down, we can see that there may be a few bounces just after, then a solid “on” period followed by more bouncing as the button is released.

7.2. Trailing Edges: Stimulus and Response Offsets

Trailing edges as you might expect are the opposite of leading edges and are the offset of a stimulus or response. In the case of visual stimuli that is when the image is no longer visible or when a sound can no longer be heard. For responses they might be when a button is flagged released or let up.

Typically trailing edges are longer and shallower than leading edges as the example sound wave below illustrates where the sound falls back to silence.



As with onsets or leading edges we need to decide where the offset or trailing edge is so that we can accurately calculate an RT. Again as with determining where the offset occurs is technically challenging as there will be a degree of jitter or bounce in the electrical signal.

For visual and auditory stimuli we use smoothing whereby we apply a fixed window which moves along the trailing curve of the stimulus in real time until the signal falls below a set sensitivity. In essence what we are doing is looking through a window of a fixed width to see when the signal falls below a set threshold. Once it has fallen below that threshold for the whole of the fixed window we determine that the offset has occurred and we can calculate the RT.

In the case of auditory stimuli being detected by the T-Pad we use a smoothing or silence window of +200 mS. This means that any raw data for Voice Key responses, e.g. those returned from XiD commands will be +200 mS in duration.

For visual responses we use a blocking period of +20 mS which means that visual stimuli or Opto markers must be black for +20 mS. The result is that raw data for visual stimuli responses will be +20 mS, e.g. those returned from XiD commands will be +20 mS in duration.

Visual and auditory onsets will be unaffected as they are measured on the leading edge of the signals which are typically much cleaner. This is very similar to how a human would perceive the offset of a visual or auditory stimulus for example.

Button releases are slightly different in the respect that when you release any button the signal you get from them is typically noisy due to the fact they bounce when released just as they did when pressed. Simply put when you release your finger from a button the electro-mechanical contacts typically bounce when you top out the switch.

To smooth out the bouncing to determine the actual on, or button down period, we use a technique called debouncing. Again we use a moving window approach but in this case we want the button to be signaled off or up for the whole of that period to signal an actual offset.

Button debouncing on the T-Pad is set at 20 mS. This means that a button must be cleanly up for 20 mS before we flag an offset. This means that any raw data for button presses, e.g. those returned from XiD commands will be +20 mS in duration.

If you make use of external buttons you should only use high quality devices that have good mechanical switches otherwise debouncing may mean that your offset is elongated due to a noisy signal that is much longer than the 20 mS debouncing window.

7.3. Accounting for Refresh Blocking, Microphone Smoothing & Button Debouncing in Your Experiment

In summary all raw timings returned from the T-Pad will be elongated by the refresh blocking, smoothing or debouncing windows described above. This applies to both XiD protocol timings returned over serial from the T-Pads inbuilt millisecond accurate hardware timer and to keystrokes when used as a USB HID keyboard. TTL event marking signals will also be elongated in terms of duration by the same amount. TTL onsets are unaffected.

If you are presenting visual stimuli faster than 20 mS you will need to turn refresh blocking off when detecting Opto markers. That is, you don't have a white to black off period of more than + 20mS. This may be the case is you are presenting different stimuli on each frame or refresh on a 240 Hz monitor for example. Each frame can be refreshed approximately every 4.17 mS on a 240 Hz monitor, i.e.

$$1000 \text{ mS} / 240 \text{ Hz} = 4.17 \text{ mS}$$

On a standard 60 Hz monitor that would be approximately every:

$$1000 \text{ mS} / 60 \text{ Hz} = 16.67 \text{ mS}$$

As a rule of thumb if you are presenting different visual stimuli and Opto markers on each frame or refresh then you should turn refresh blocking off in the T-Pads settings. In the majority of cases this will not apply unless you are running experiments with Rapid Serial Visual Presentation (RSVP) or similar.

If you are presenting sounds with less than 200 mS silence between them you are advised to switch off microphone smoothing in the T-Pads settings. With this switched off you may see more jitter in the data. Generally if using the Voice Key to accept human responses this should not be an issue.

If you are using the Voice Key microphone to measure computer generated audio stimulus presentations where there is less than a 200 mS silence between them you should switch off microphone smoothing in the T-Pads settings.

7.4. Refresh Blocking, Microphone Smoothing & Button Debouncing Raw Data Correction Factors

Regardless of whether you are using the T-Pads hardware timer via XiD commands or are measuring key down times when using the T-Pad in USB HID keyboard mode you should apply the duration corrections shown in the table below.

This also applies to the dictation of TTL event marks.

Sensor	Modality	Methodology	Value to subtract from raw data durations
Opto	Vision / Opto marker	Refresh blocking	- 20 mS
Voice Key Microphone	Sound / Voice Key	Microphone smoothing	- 200 mS
Button press	Biomechanical / Button	Button debouncing	- 20 mS

Remember you only need to apply the following corrections to duration timings as onsets are unaffected. If you have turned OFF refresh blocking, microphone smoothing or debouncing you do not need to apply these corrections to durations. By default these are set to ON and you will need to apply the duration corrections in the table above to your raw data.

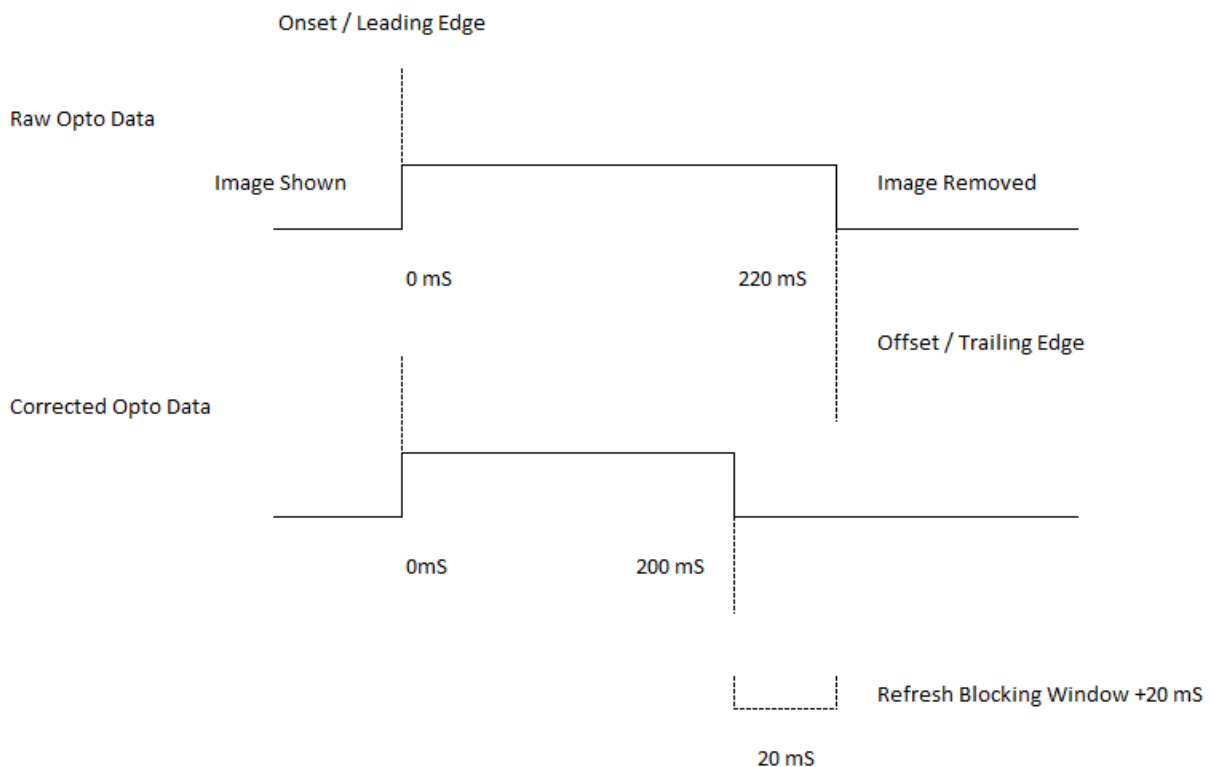
Note that our App applies these duration correction factors automatically when viewing data in the Line By Line spreadsheet view or on the Logic Analyzer plot.

7.5. Practical Examples of Application of Correction Factors

It is useful to illustrate how you should apply correction factors to actual data plots. Note that onsets or leading edges are unaffected. Conceptually the best way to think about refresh blocking, microphone smoothing and button debouncing windows is that you cannot know when something has ended until you wait so long after it has ended. A good analogy is an audience applauding at the theatre, you would only know when the applause had finally stopped if you waited so long afterwards. You could then go backwards in time and say exactly when the clapping stopped.

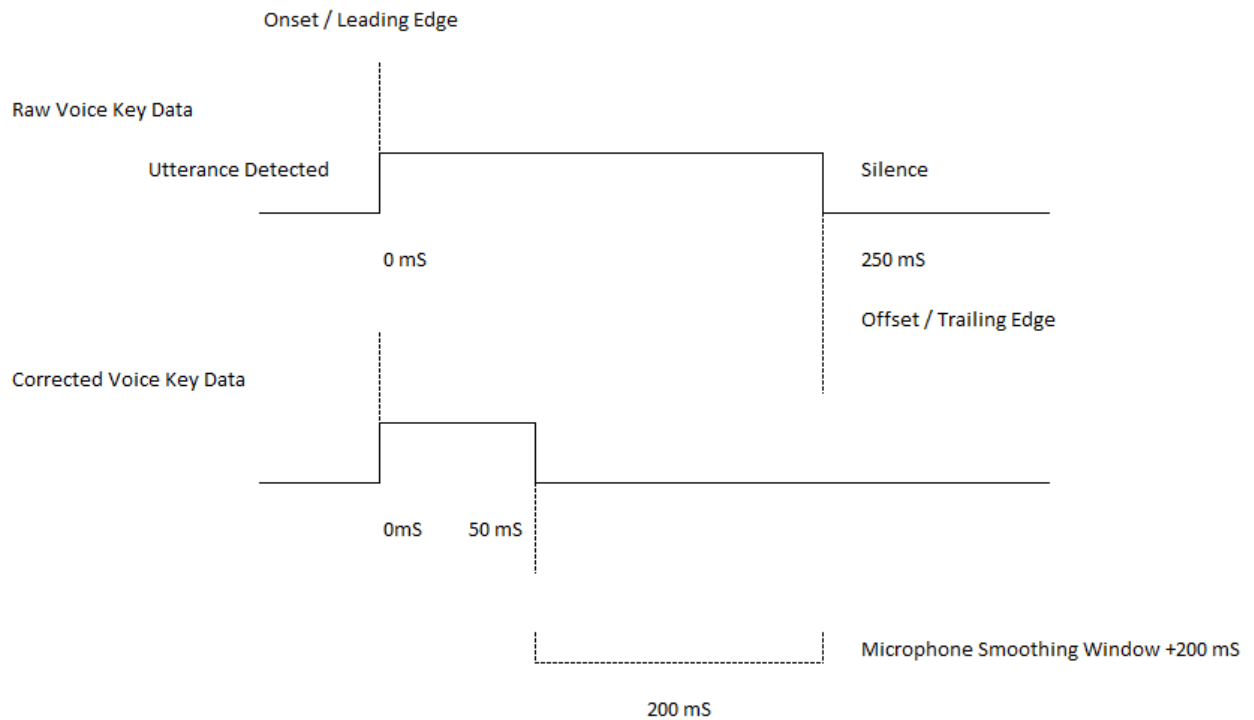
7.5.1. Opto Correction

In the example shown below a visual stimulus image has been detected and recorded with a duration of 220 mS (Raw Opto Data). If running in Mode 5, XiD data might show the offset/trailing edge timestamp as 220 mS, however the actual offset would be 200 mS once the refresh blocking window of +20 mS is subtracted. Note the image onset is unaffected.



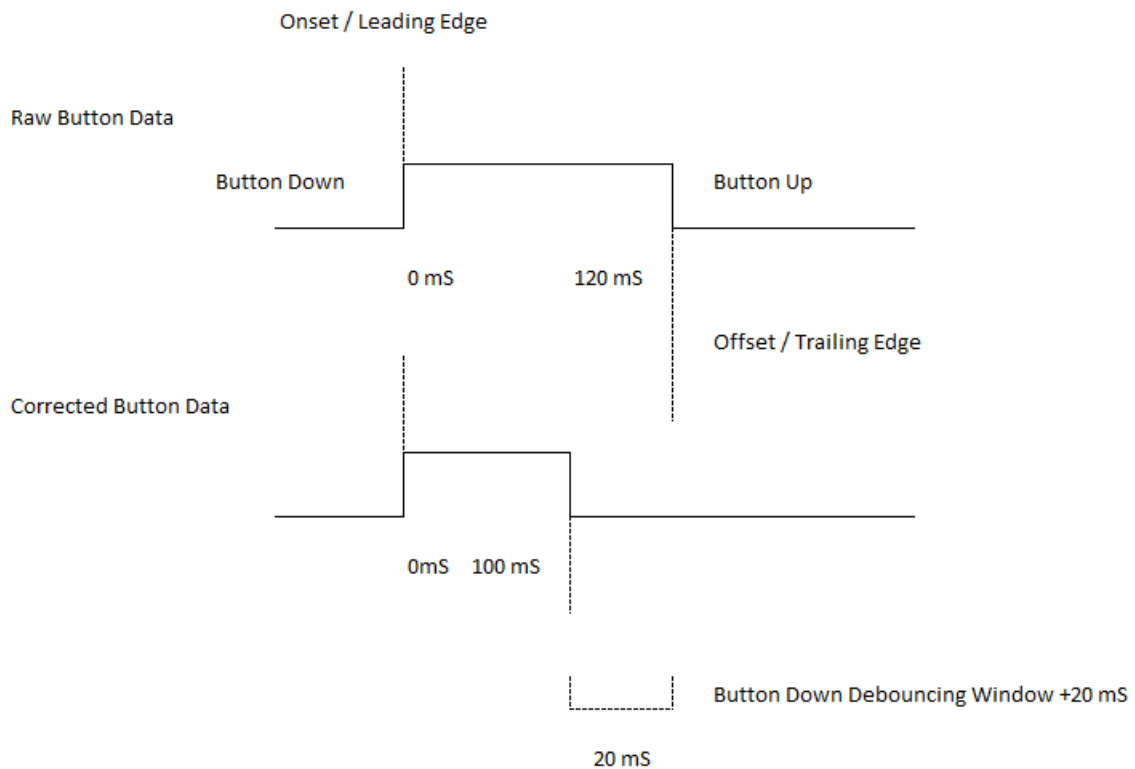
7.5.2. Microphone Smoothing Correction

In the example shown below an utterance has been detected and recorded with a duration of 250 mS (Raw Voice Key Data). If running in Mode 5, XiD data might show the offset/trailing edge or utterance duration timestamp as 250 mS, however the actual offset or silence would be 50 mS once the microphone smoothing window of +200 mS is subtracted. Note that the Voice Key onset is unaffected.



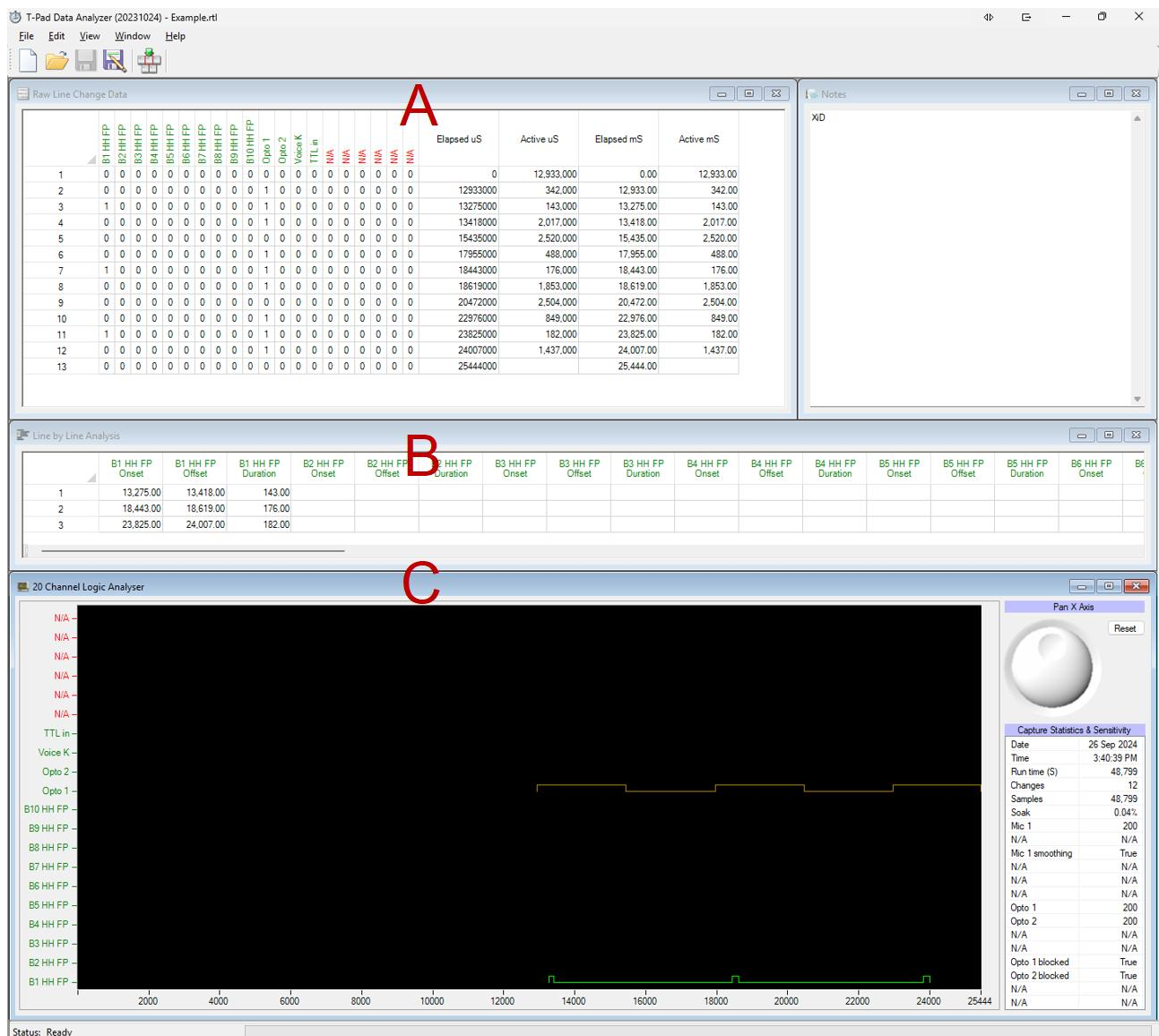
7.5.3. Button Debouncing Correction

In the example shown below a button down has been detected and recorded with a duration of 120 mS (Raw Button Data). If running in Mode 5, XiD data might show the offset/trailing edge or button up timestamp as 120 mS, however the actual offset or button up would be 100 mS once the button debouncing window of +20 mS is subtracted. Note that the button down onset is unaffected.



7.5.4. Automatic Correction

In you capture to a .RTL file using the T-Pad App the T-Pad's own Data Analyzer software automatically applies the corrections listed in the table above to the Line by Line Analysis spreadsheet (B) and to the 20 Channel Logic Analyzer plot (C) shown below.



Note it does not apply any corrections to the Raw Line Change Data spreadsheet (A) above. Also note that these may appear out of sequence as there is a row change each time there is a change on any sensor or button.

8. Using the Sensor Check Utility Built Into the App

The T-Pad App has an in-built Sensor Check Utility (SCU), or Experiment Simulator, which lets you check that your sensor sensitivity levels are set correctly and that data is being sent to the T-Pad when optos are activated, a mic threshold passed or button pressed.

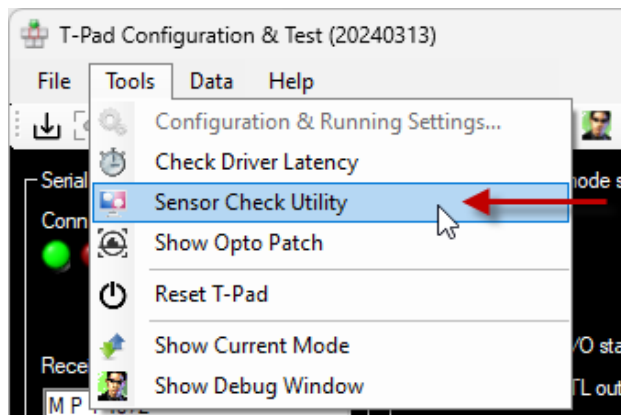
This utility can either be run on the PC the T-Pad is plugged in to or on a second PC display device you are using to check functionality. If you wish to use a second PC you will need to install the App on to that PC and run the SCU on that PC whilst the App runs on the PC with the T-Pad plugged into it.

If you are running on the same PC that the T-Pad is plugged into you will need to ensure that the SCU is displayed as the upper most window and has focus, i.e. it is the active window.

The SCU is designed to help you check whether the sensors are operating correctly, e.g. Optos, mic and buttons. It can also generate tones to trigger your Voice Key and simulate simple experimental paradigms, e.g. cross modal priming. It will also tell you which response pad button has been pressed or which mouse button has been clicked.

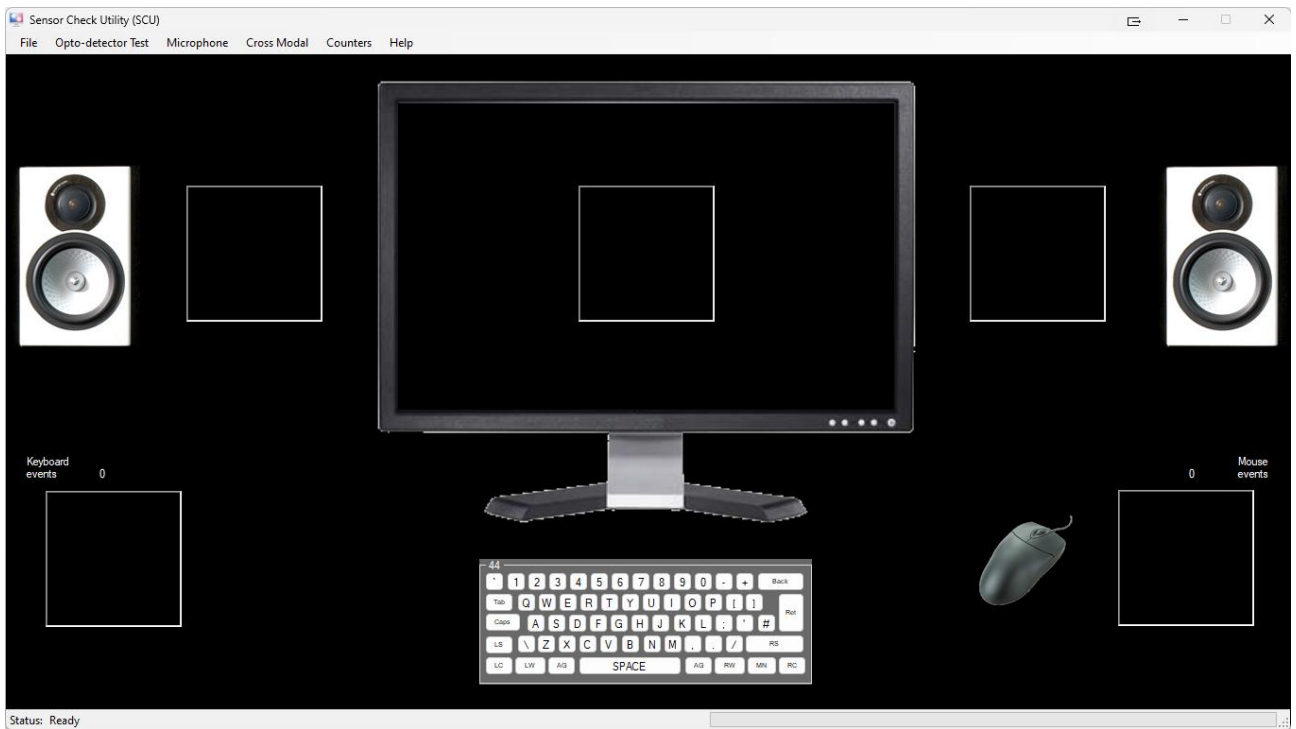
Note that the SCU will only tell you which response pad button has been pressed if the Keyboard USB lead has been connected and the keyboard HID is active. If you have chosen to operate using only the VCP serial connection you will need to use the main T-Pad Configuration & Test screen to observe responses.

If the Keyboard USB lead is connected when a button key is pressed or a mouse button is clicked it will helpfully show an Opto event marker on screen. This can then be used as an event marker by one of your Opto-detectors.



To start the SCU select Tools | Sensor Check Utility.

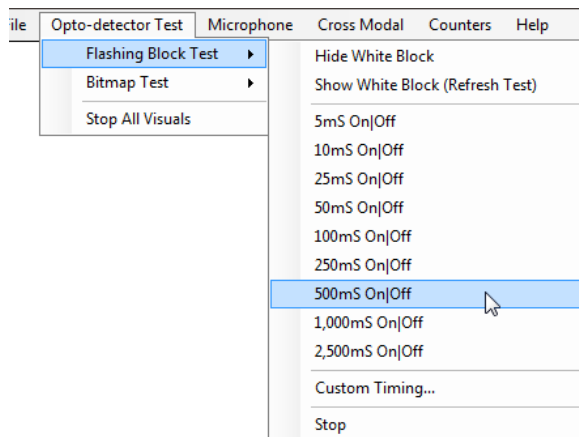
When the SCU starts you will be presented with a virtual PC and several Opto event marker patches over which you can place an Opto-detector.



8.1. Checking Optos with the SCU



To check your Optos are functioning correctly you could position an Opto mid screen and then select Opto-detector Test from the menu.



The Flashing Block Test allows you to select a number of timings. Initially you are advised to select a relatively slow duty cycle, e.g. 500 mS on|off.

You can then use this regularly flashing block to adjust the sensor thresholds of your Opto as described in the Opto and Sensor Threshold sections of this guide.

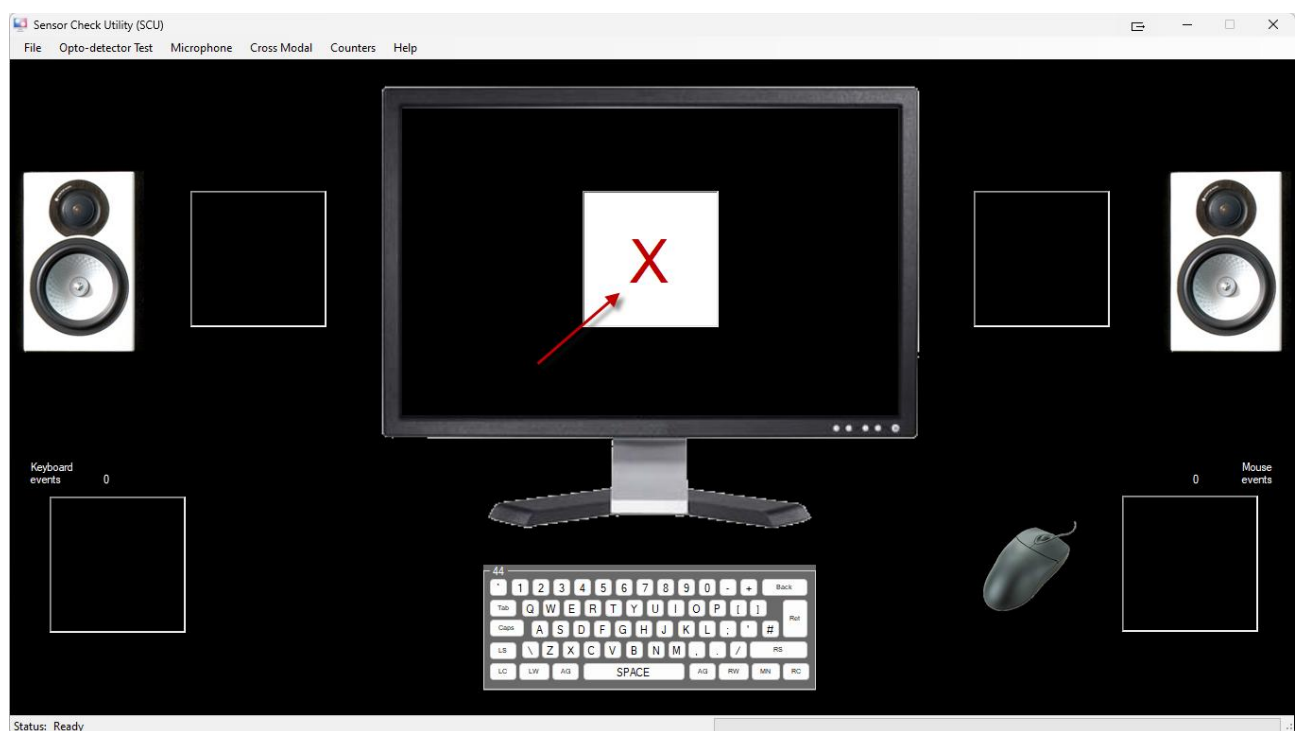
Below we can see that when an opto-detector is positioned at location X it triggers the Opto.

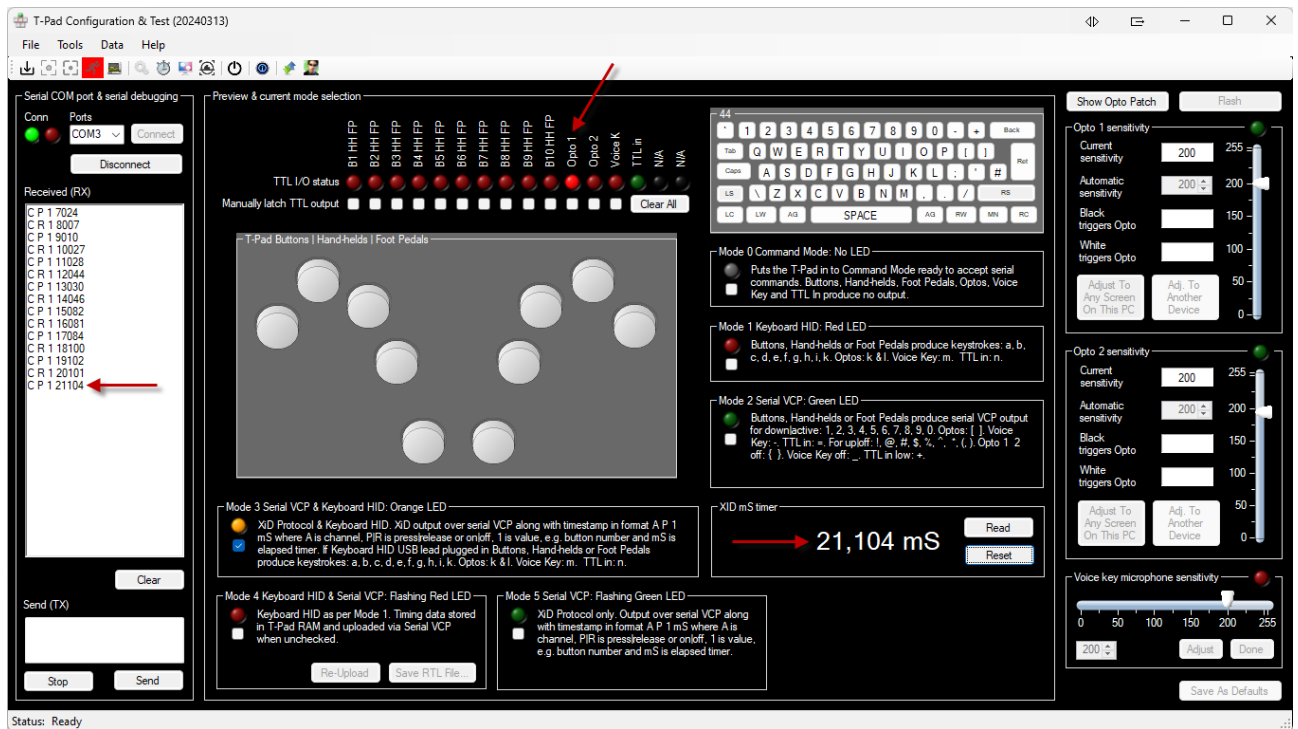


If the sensitivity of your Opto-detectors has been set correctly in Mode 3: XiD protocol this will produce a stream of XiD data.

Should you not see a regular stream of opto timing data such as that shown below you should follow the procedure for setting the sensitivity of your opto-detectors.

Even if your Keyboard USB lead is not connected and you are using only the serial interface, you can still check response button functionality as each press will be timestamped and appear as an XiD command in the Received (RX) window. The same also applies to Voice Key input and external TTL in triggers.

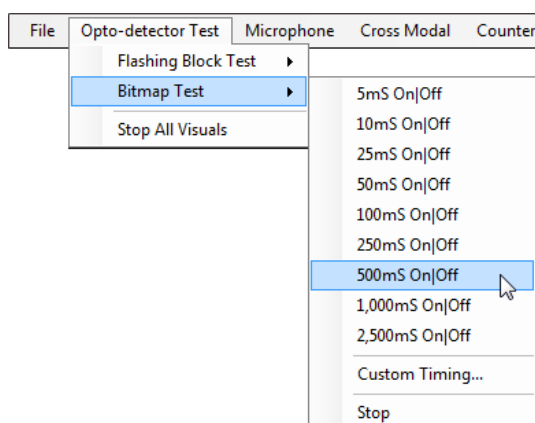




Note that the display and sound generation timings of the SCU are unlikely to be accurate. It is purely designed to help you ensure sensors are working and to set sensitivity and activation thresholds.

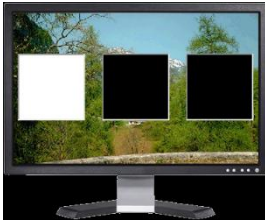
In all modules you are free to enter your own timings by selecting Custom Timing... from the relevant menu. You can also run more than one test simultaneously should you so wish as the SCU is fully multi-tasking. For example you could run the Flashing Block Test whilst also ping ponging tones between your left and right speakers.

8.2. Checking More than One Opto



Depending on your T-Pad model you may have more than one Opto-detector. To check more than one Opto, e.g. for priming work, you can select the Bitmap Test. This will show two Opto event markers on screen each time a bitmap is displayed.

Two bitmaps will then alternate at the chosen duty cycle.

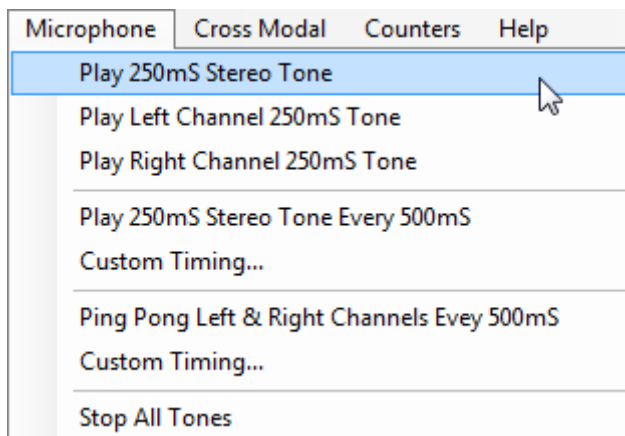


Two Optos can be positioned over each event marker (left and right white blocks).

You can also place an Opto over the middle Flashing Block Test event marker and run that test at the same time as the Bitmap Test at the same or different duty cycle. To stop activity click Stop or Stop All Visuals.

Note how Opto event markers are structured. Where you wish it to be triggered it should be pure white. When you wish an Opto not to be triggered you should ensure you have a black event marker in place.

8.3. Checking Voice Keys



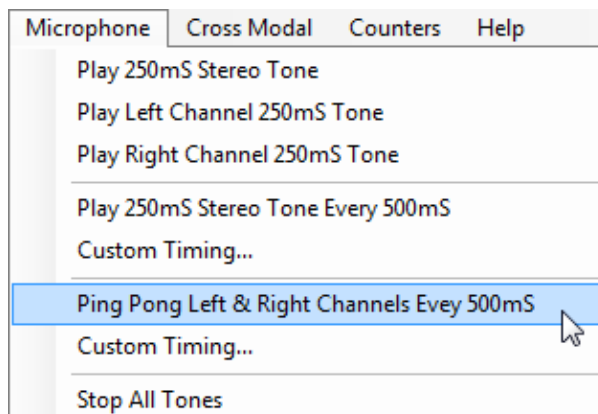
You can check Voice Keys by selecting Microphone from the menu.

You then have the option of playing either a single 250 mS stereo tone, a tone through the left speaker or the right speaker.



For more complex tests you can play a tone every 500 mS to help you set mic activation thresholds. If you select this option the two Opto event markers next to the virtual speakers will turn white when each tone is being played.

This helps you check timing between the Optos and mics. To stop tones click Stop All Tones.



Finally you can choose to ping pong a tone between the left and right speakers at the chosen duty cycle.

When each tone plays the relevant event Opto event marker will turn white for the duration of the tone.

8.4. Checking Cross Modal Sensors



Checking your sensors are set correctly for a Cross Modal paradigm can be achieved by selecting Cross Modal from the menu.

You can either choose a 500mS duty cycle or enter your own custom timing.



Bitmaps and tones will cycle at the chosen duty cycle and Opto event markers will be shown as each stimulus is displayed. To end the test click Stop.



8.5. Checking Keyboard and Mouse Responses

A virtual keyboard and mouse are shown along with two Opto event markers which show when a key or button is pressed.

Two sets of counters are also shown which keep a tally of key and button presses.



To check key presses and button down activity the SCU window must have focus. That is, you must have clicked on the title bar if the window was not in focus.

Now when you press a key on the keyboard the relevant key will turn green and the opto event marker will turn white. The key activity counter will increment based on the keyboard key repeat rate set on the system you are working on.



If you want to reset the counters, select Counters | Reset Counters from the menu.

You can combine any of the options in the SCU to check complex sensor and response setups if you wish. For example, you could use the Flashing Bitmap Test alongside playing a 250mS tone every 500mS whilst making keyboard responses. When each event occurred the relevant opto event marker would also be displayed.

To illustrate why getting accurate timing can be so difficult try running the Cross Modal test and pressing multiple keys on your physical keyboard as rapidly as possible. You should be able to notice that timing begins to drift noticeably! For example the tones being played may drift and become out of sync with the visuals being displayed.

9. Capturing Activity Data Using the T-Pad Configuration and Test App

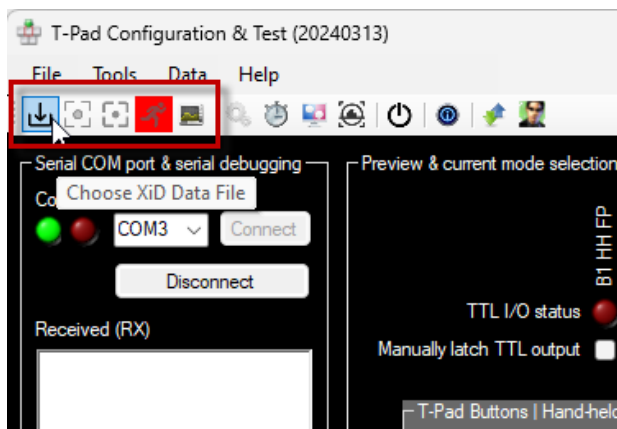
It is possible to use the T-pad App to capture and time all activity with millisecond accuracy. When operating in this mode all data will be stored to a BBTK .RTL format file which can be loaded into the T-Pad Data Analyzer once your experiment has finished.

A typical scenario might be where you wish to run your own experiment into which you have inserted Opto patch markers into your visual stimulus materials. When you wish for the onset of a visual stimulus to be recorded you should ensure that a white patch appears under an Opto-detector. For the duration of the stimulus image you should ensure that the pure white patch is shown. Prior to and after the stimulus is displayed you should ensure that a pure black Opto patch is shown at the same location. To make testing simpler you may choose to set your background to a constant pure black.

By using a black (no stimulus) and a white Opto patch (stimulus shown) this enables the Opto-detector to detect the onset of the visual stimulus when it first appears, its duration and its offset when it is removed.

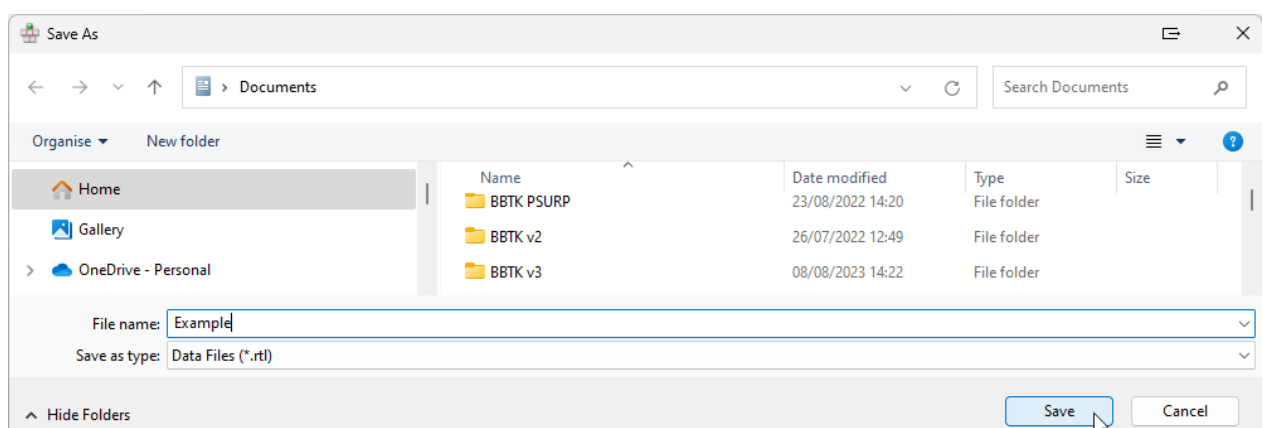
To capture events and timing data you will need to ensure that you are connected to the T-Pad and that your sensor sensitivity and thresholds are set correctly.

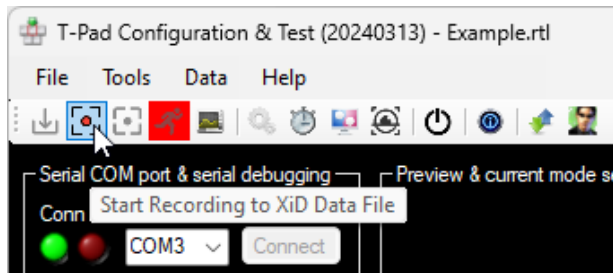
You also need to be running in Mode 3: XiD protocol.



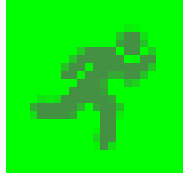
The first thing you need to do this click on the Choose XiD Data File button.

Next choose the folder and filename you wish to save data into. In effect this will save all streamed data from the T-Pad into this file.



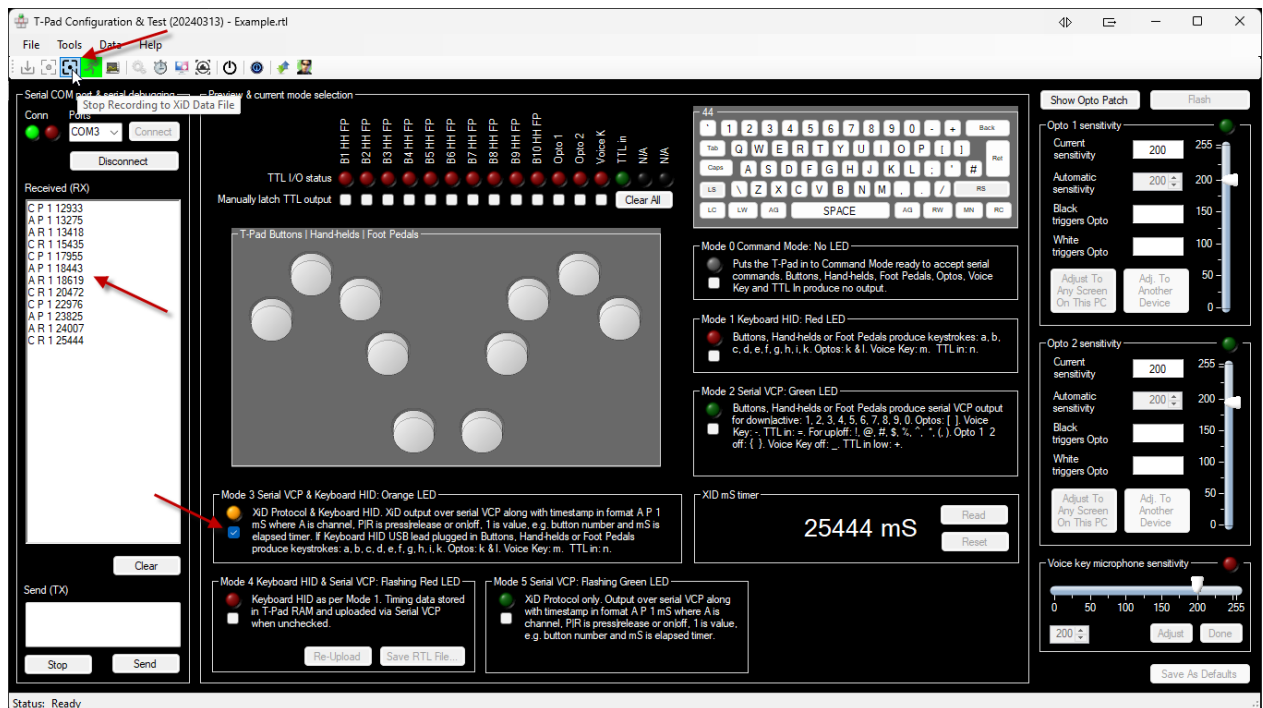


Now that you have chosen the actual filename click on the record button to start recording all activity.



Once you click on the record button the running man icon will turn green.

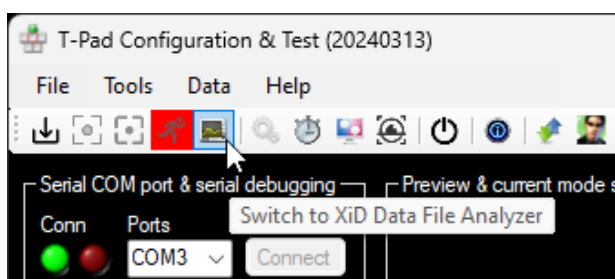
As data is recorded it will be shown on screen in XiD Mode. Once you have completed your run click on the stop button.



Data will now be written to the chosen .RTL file and can now be analyzed with the T-Pad Data Analyzer.

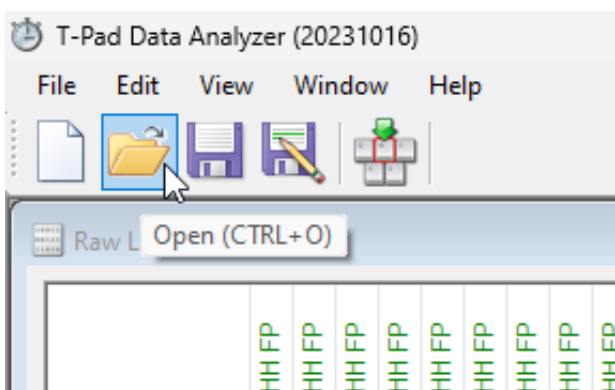
NOTE: DATA will ONLY BE SAVED to the .RTL file once the STOP BUTTON IS PRESSED.

10. Analyzing Captured Activity Using the T-Pad Data Analyzer

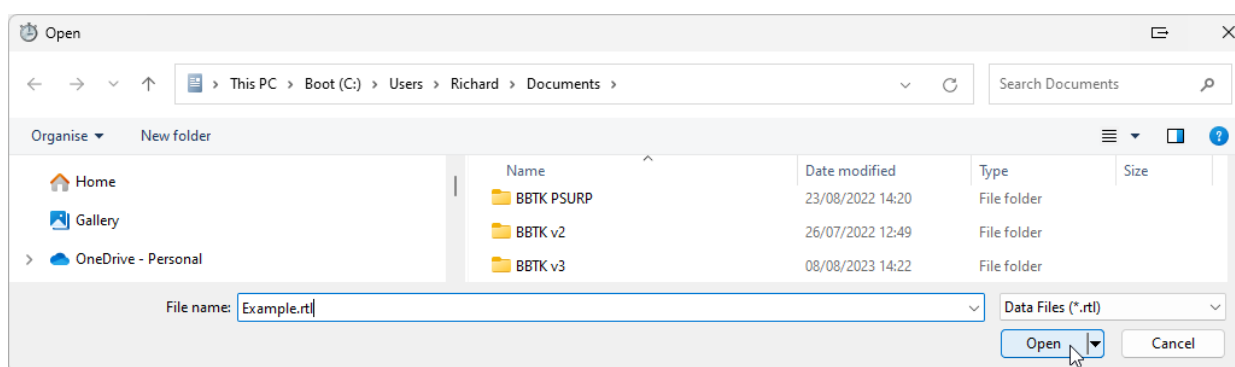


To analyze data you can either use the Switch To toolbar button or start the T-Pad Data Analyzer independently.

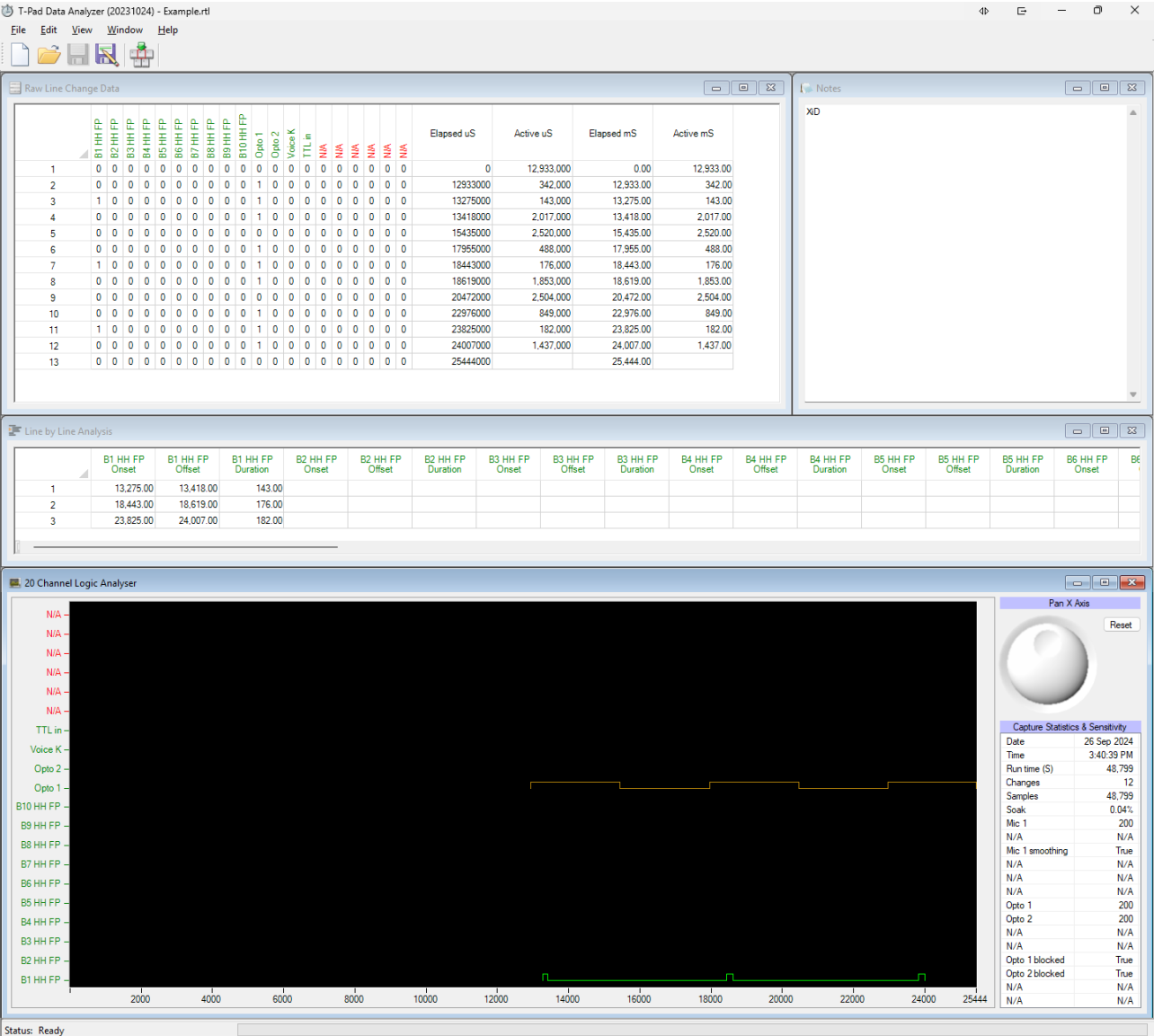
To analyze data you can either use the switch to toolbar button or start the T-Pad Data Analyzer independently.



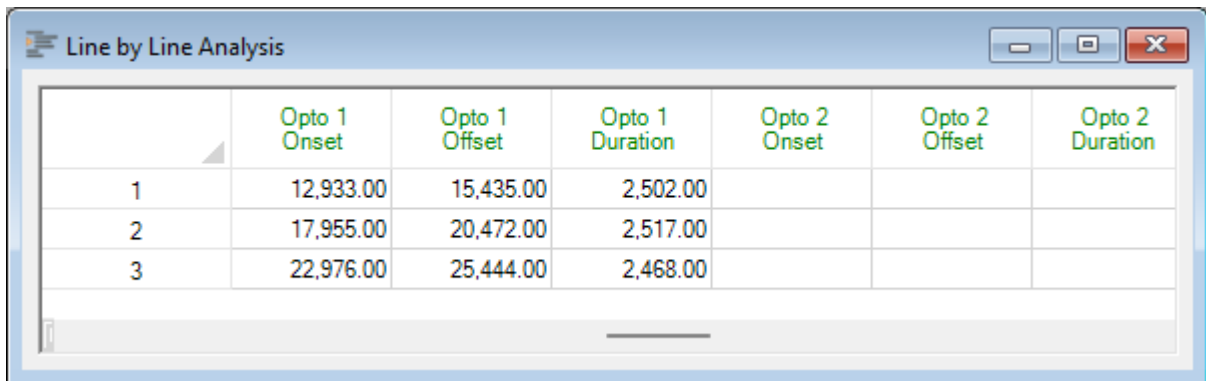
Once the T-Pad Data Analyzer has started load the chosen .RTL file.



The raw line change data will then be shown, parsed into an easy to use spreadsheet view and plotted on a graphical display against a millisecond timebase.



To look at individual line activity you can use the Line by Line Analysis spreadsheet. Here Opto 1 activity is shown:

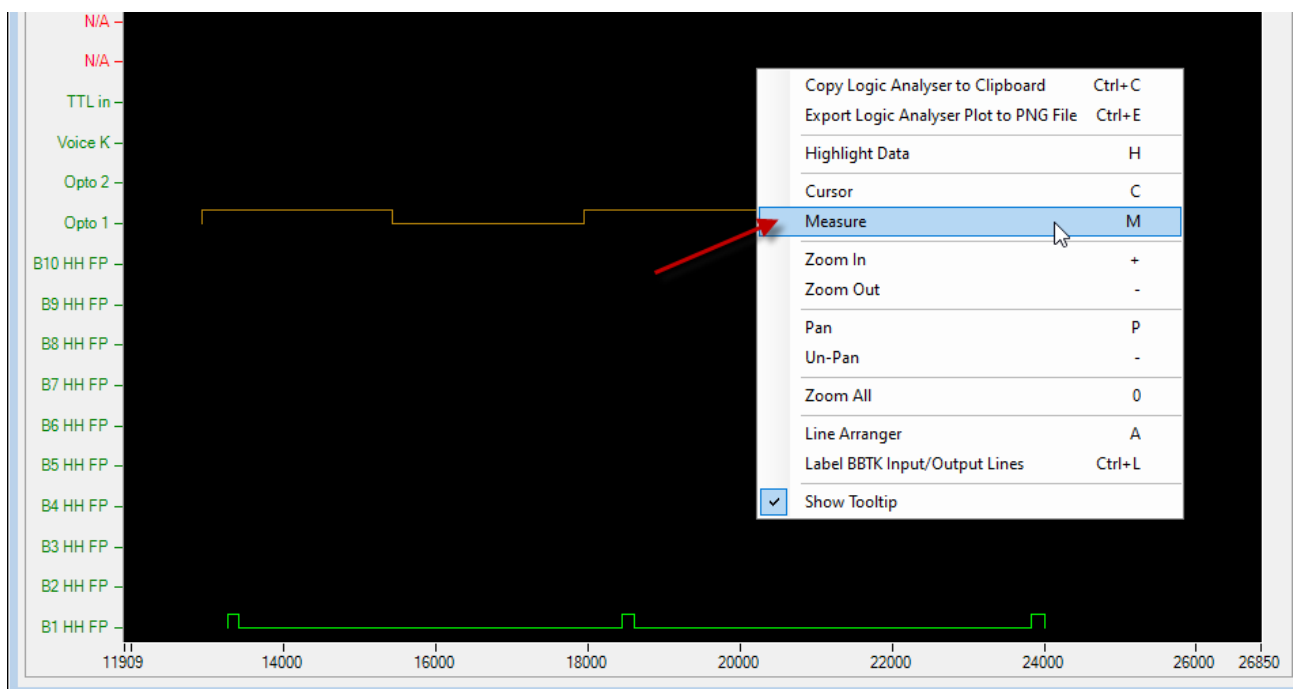
A screenshot of a software window titled "Line by Line Analysis". It contains a table with 7 columns: an empty column, "Opto 1 Onset", "Opto 1 Offset", "Opto 1 Duration", "Opto 2 Onset", "Opto 2 Offset", and "Opto 2 Duration". The first three columns have data for three rows, while the last four columns are empty.

	Opto 1 Onset	Opto 1 Offset	Opto 1 Duration	Opto 2 Onset	Opto 2 Offset	Opto 2 Duration
1	12,933.00	15,435.00	2,502.00			
2	17,955.00	20,472.00	2,517.00			
3	22,976.00	25,444.00	2,468.00			

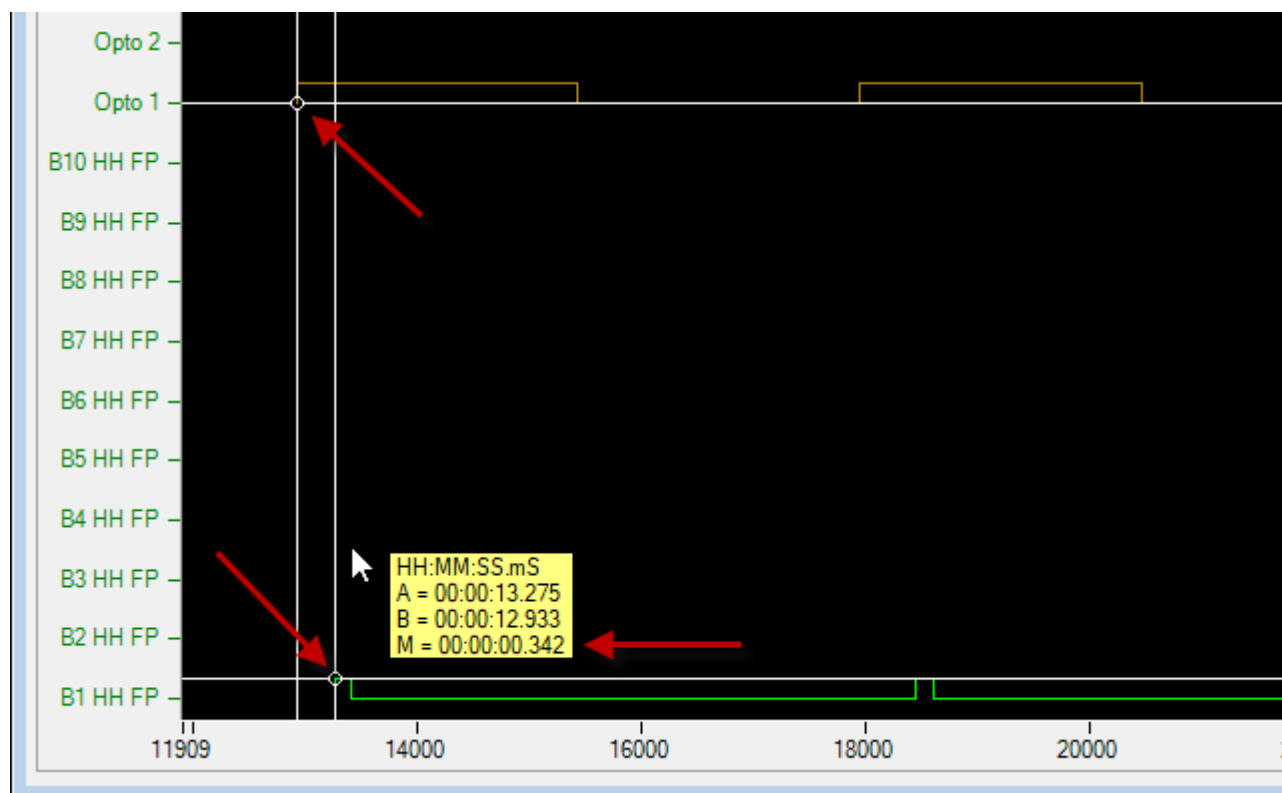
By using the 20 Channel Logic Analyzer you can examine things like Reaction Time (RT) relative to a visual image onset recorded by the Opto-detector.

Remember all these measures are when stimulus images actually appeared in the real world and when buttons etc. were pressed as they are timings from the internal clock of the T-Pad that is not tied to the accuracy of your PC, Windows or the experiment package you are using.

To check an RT right click on the 20 Channel Logic Analyzer and click on Measure from the context menu or press M on the keyboard.

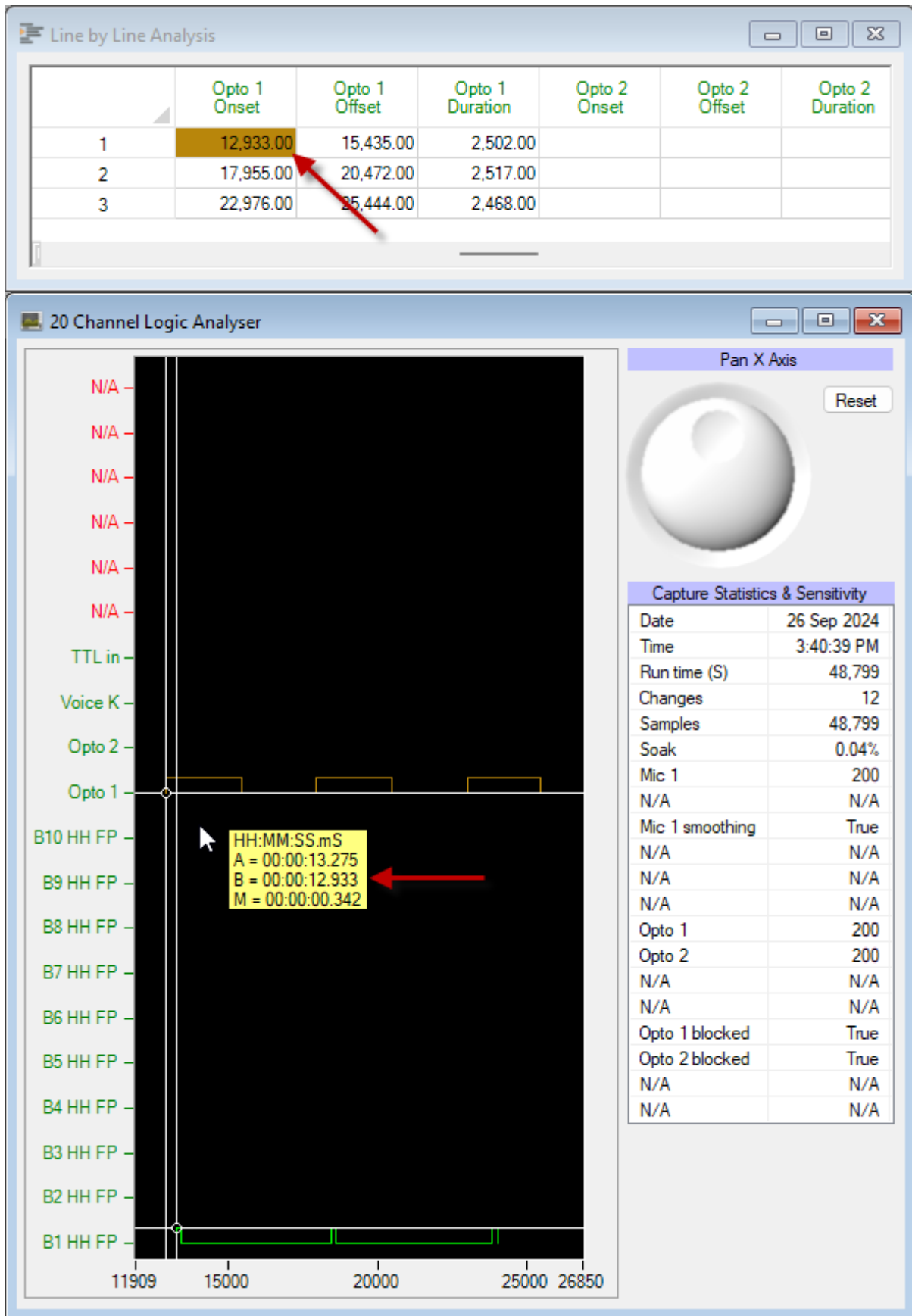


Two measurement cursors will now appear.



If you drag the first cursor to the onset of Opto 1 and the second to the onset of button 1 being pressed, i.e. going high, you can read off the M value of 342 mS. So we know that the real-world RT relative to that visual stimulus onset is 342 mS.

Whilst you have a cursor highlighted you can also press H on the keyboard to jump to that onset in the spreadsheet view:



Note the Logic Analyzer plot is only accurate to 1mS, whereas E is accurate to hundredths of a millisecond, i.e. two decimal places.

- F** Shows summary Capture Statistics & Thresholds. This provides details on capture date, time, runtime, number of line changes or events, samples taken, soak rate, sensor thresholds and whether the opto-detectors blocked together individual refreshes on a CRT.
- G** Shows the status bar and progress indicator. For example, when longer captures are plotted on the Logic Analyzer this will show which lines are being analyzed.

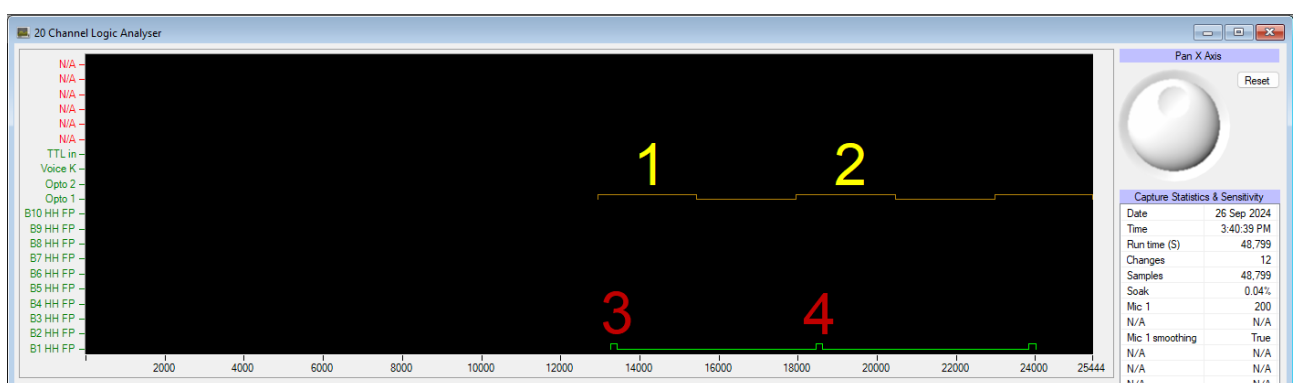
10.2. Checking a Simple Reaction Time Paradigm

Checking how accurate your own hardware and software are is made straightforward with use of the T-Pad. Usually when checking your timing the first place to start is the 20 Channel Logic Analyzer. This shows input lines and sensors in green and output or generation lines in red against a time base in mS (milliseconds). Each line is shown in a different color to help differentiate between them.

In the example shown lets imagine that the participant's task was to respond to visual images as rapidly as possible by pressing button 1 on the T-Pad. From the screen grab below we can see that only two lines are active, Opto 1 and Button 1 (B1 HH FP for Button 1, Hand-held or Foot Pedal).

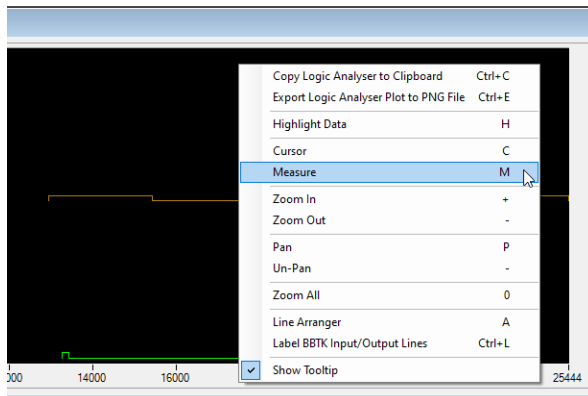
Hypothetically, a commercial Experiment Generator might also be recording the same actions and data via a scripted paradigm (this represents your experiment in this example). In effect the T-Pad is piggy backing on the Experiment Generator to check its timing.

In a perfect world both the T-Pad and the commercial Experiment Generator would record exactly the same time. However this won't be the case as the commercial Experiment Generator will be running on a non-realtime OS with non-realtime applications and will suffer from imperfect timing.

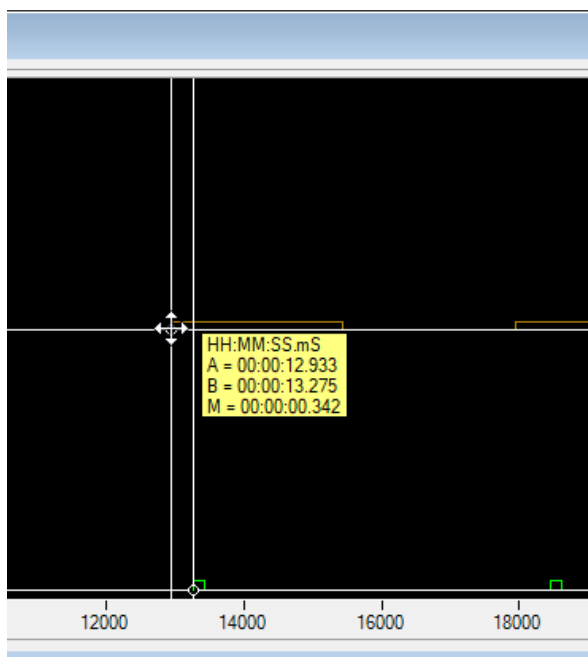


- 1 Shows the onset and duration of the first visual stimuli event.
- 2 Shows the onset and duration of the second visual stimuli event.
- 3 Shows the onset and duration of button 1 being pressed on the T-Pad in response to the first visual stimulus.
- 4 Shows the onset and duration of button 1 being pressed on the T-Pad pad in response to the second visual stimulus.

To work out the true Reaction Time, measured by the T-Pad with mS accuracy and uS precision, we would simply need to compare the onset of Opto 1 events with the onset of Button 1 events. With the T-Pad this is incredibly quick and easy to do.

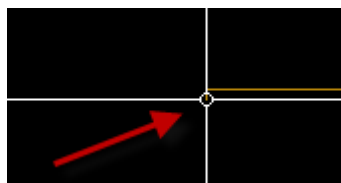


First we need to bring up two Measurement Cursors on the Logic Analyzer plot. You can do this by right clicking anywhere on the plot and clicking Measure from the Context Sensitive Menu (or you can press the shortcut key M on the keyboard).



Two Measurement Cursors will then appear.

These can be dragged by the reticle between any line and point in time on the plot.



Initially we would drag Cursor A (left) to approximately the onset of the visual event on Opto 1 and Cursor B to the onset of the response pad button 1 going down. Note how this is only done approximately in the first instance.

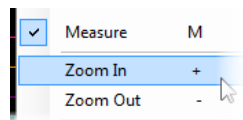
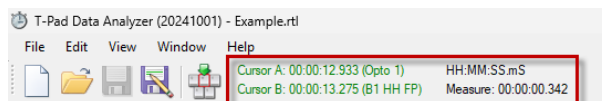
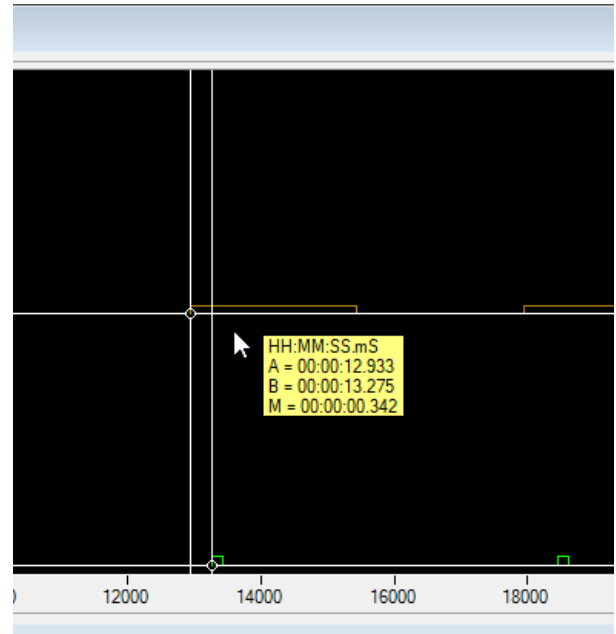
Time in mS is shown in the pop-up tooltip. A = Cursor A, B = Cursor B and M = the difference between the two, i.e. the Measure.

Timings are displayed in:

hours:mins:seconds.millesconds

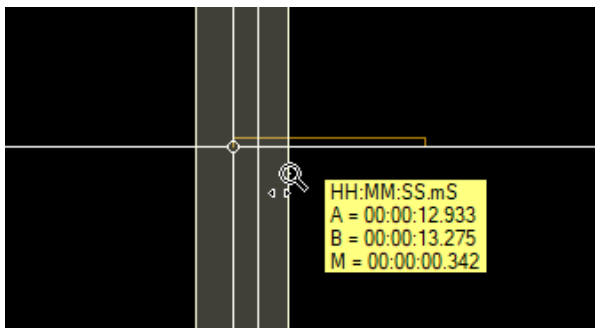
or

HH:MM:SS.mS



More detail is shown in the upper toolbar as you move the Measurement Cursors.

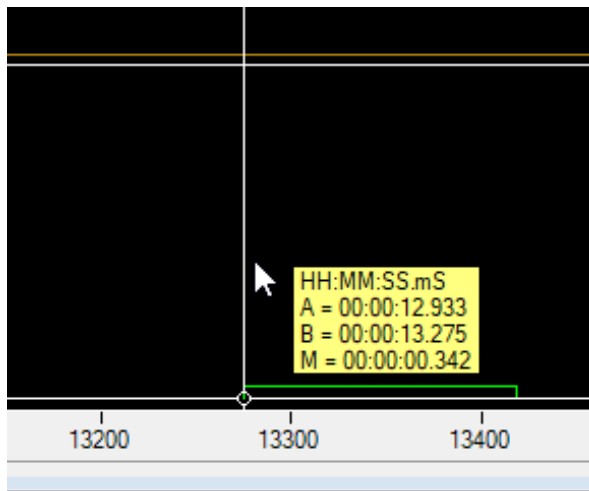
To obtain an accurate snap to each onset, right click and then click Zoom In. Or press + on the keyboard (next to backspace).



A magnifying glass cursor will then appear and you should click and drag around the vertical onset of the event you are interested in. This won't move the cursor line and you can Zoom in and out as many times as you like.

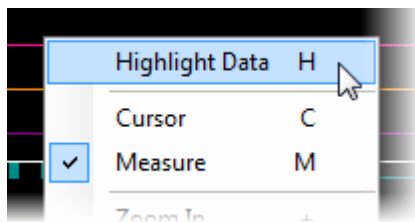


Here we can see that the onset of the actual event has been accurately snapped to after zooming in. You would repeat this process for Cursor B after Zooming out.



After Zooming in and snapping to the onset of both events we can read off that the actual **Reaction Time was 342 mS**.

This is the figure which we would compare with that recorded by our commercial Experiment Generator. Any difference would be error due to hardware and software issues, problems with the Experiment Generator or human error when producing its scripts.



To further aid in analysis it's often useful to jump to the exact onset timings for both events in the Line by Line Analysis spreadsheet. As previously mentioned this spreadsheet is accurate to 100ths of a mS whereas the plot is accurate to 1mS due to the volume of data being plotted.

To highlight the events click Highlight Data or press H on the keyboard.

Line by Line Analysis				
	B10 HH FP Duration	Opto 1 Onset	Opto 1 Offset	Opto 1 Duration
1		12.933.00	15.435.00	2.502.00
2		17.955.00	20.472.00	2.517.00
3		22.976.00	25.444.00	2.468.00

Line by Line Analysis				
	B1 HH FP Onset	B1 HH FP Offset	B1 HH FP Duration	
1	13.275.00	13.418.00	143.00	
2	18.443.00	18.619.00	176.00	
3	23.825.00	24.007.00	182.00	

The Line by Line Analysis spreadsheet will automatically highlight the two onsets. Remember the second highlighted onset may not be visible unless you scroll around so that it is in view.

Here we can see that the onset of the Opto 1 event at Cursor A was 12,933 mS and the button 1 down event at 13,275 mS. **So the Reaction Time was 342 mS.**

Best of all you don't have to be that accurate with your cursors when Highlighting as the PC Software will automatically find the nearest associated onset relative to each cursor!

10.3. Additional Features of the Logic Analyzer

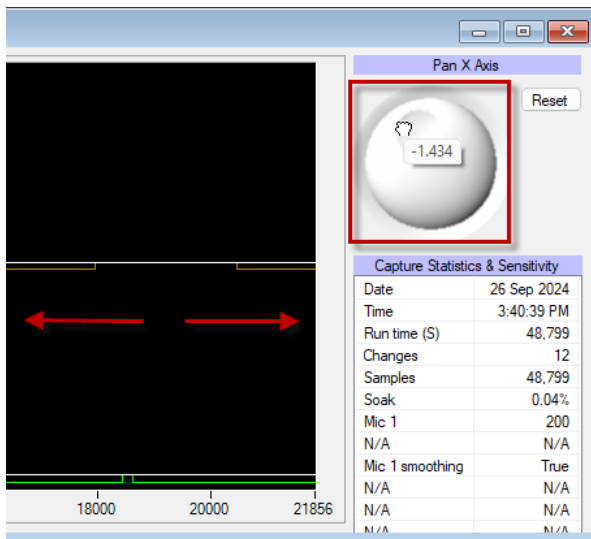
The 20 Logic Analyzer has a number of additional features which help you analyze and self-validate your timing accuracy.

10.3.1. Shortcut Keys

Shortcut keys help you quickly work with timing data.

Shortcut	Purpose	Description
Ctrl+C	Copy Plot	Copy Logic Analyzer plot to clipboard.
Ctrl+E	Export Plot	Export Logic Analyzer plot to PNG file.
H	Highlight Data	Highlights the event onset near either Cursor (A) or Measurement Cursors (A & B) in the Line by Line Analysis spreadsheet.
C	Cursor	Provides a single Cursor A which can be used to pinpoint an elapsed time in mS on any line. Pressing C again will hide the Cursor.
M	Measurement Cursors	Provides two Measurement Cursors (A & B) which can be use to pinpoint two elapsed times in mS on any two lines. A measure M in mS will show the difference between them. M to cancel.
+	Zoom In	Zoom In to any part of the plot to clearly identify event onsets and offsets. Click and drag around the points you want to Zoom In to after pressing + (next to backspace) and the magnifying glass cursor appears.
-	Zoom Out	Zoom out. Undo the last Zoom In command. Features multiple levels of undo.
P	Pan	Pan lets you move the Zoomed plot left or right around your cursor. To Pan press P and click and drag the plot left or right.
-	Un-Pan	Un-Pan will undo any panning commands you recently issued. Features multiple levels of undo.
0	Zoom All	Zoom All will show all the captured data for the full capture duration. Equivalent to Zoom Extents.
A	Line Arranger	Allows you to choose which lines are displayed on the logic analyzer plot.
Ctrl+L	Label I/O lines	Allows you to enter custom label lines.
Show Tooltip	Turn Off Measurement Tooltips	If you have a lot of data (or a slow computer) you might want to turn off the Tooltip to make manipulating the plot faster.

10.3.2. Using the Pan Jog Dial



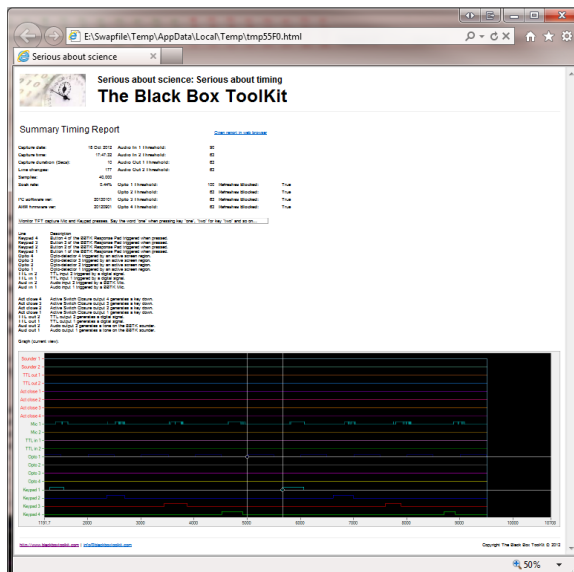
The Panning Jog Dial enables you to pan the whole of the Logic Analyzer plot left or right.

This is in contrast to if you Pan (P) from the context menu which enables you to pan left or right one screen at a time. To pan using this method click and drag with the hand cursor.



Panning works at whatever level of Zoom you are at.

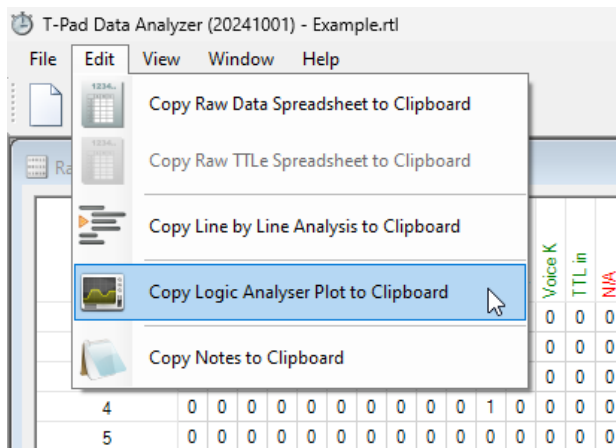
10.3.3. Producing a Summary Report with the Current Logic Analyzer Plot



When you produce a Summary Report the plot view that you can see will be the one that is used. So if you have Zoomed In it is that Zoomed view that you will see in the report.

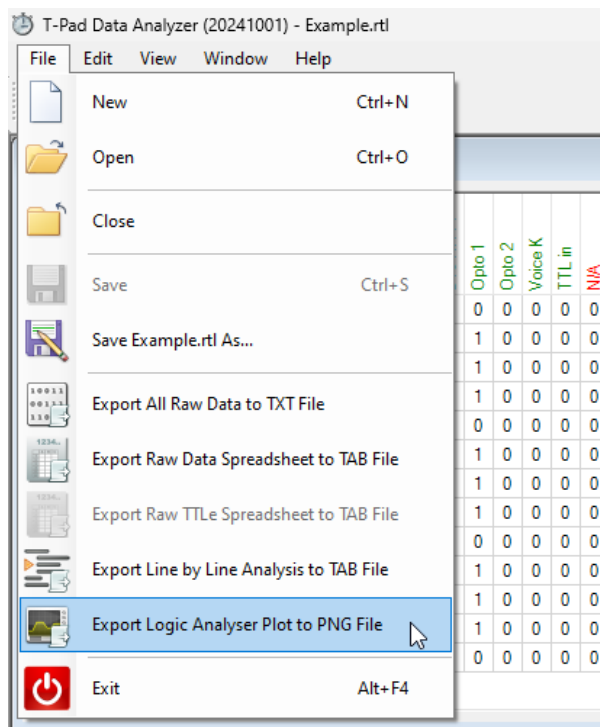
Press CTRL+R to bring up a Summary Report.

10.3.4. Copying a Logic Analyzer Plot to the Clipboard



You can copy and paste the current Logic Analyzer view into any other Windows application you like, e.g. Word. From the Edit menu select “Copy Logic Analyzer Plot to Clipboard” and paste into your chosen application.

10.3.5. Exporting the Logic Analyzer Plot to a PNG file



An alternative to Copy and Pasting the Logic Analyzer Plot is to save the current view as a standard PNG (Portable Network Graphics) file.

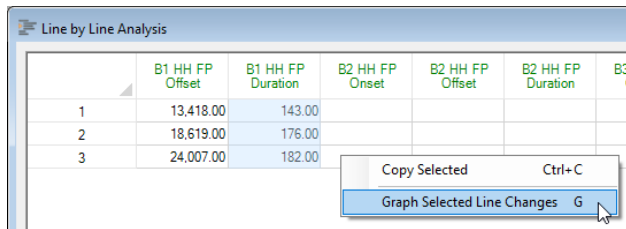
From the File menu select “Export Logic Analyzer Plot to Image”.

A standard File Save As dialog box will appear enabling you to choose where you save the image and what you call it.

You can save as many versions as you like, e.g. if you’ve zoomed and measured elapsed time you might want to save that view.

10.3.6. Line Change Graphs: Quickly Visualizing Line Change Data

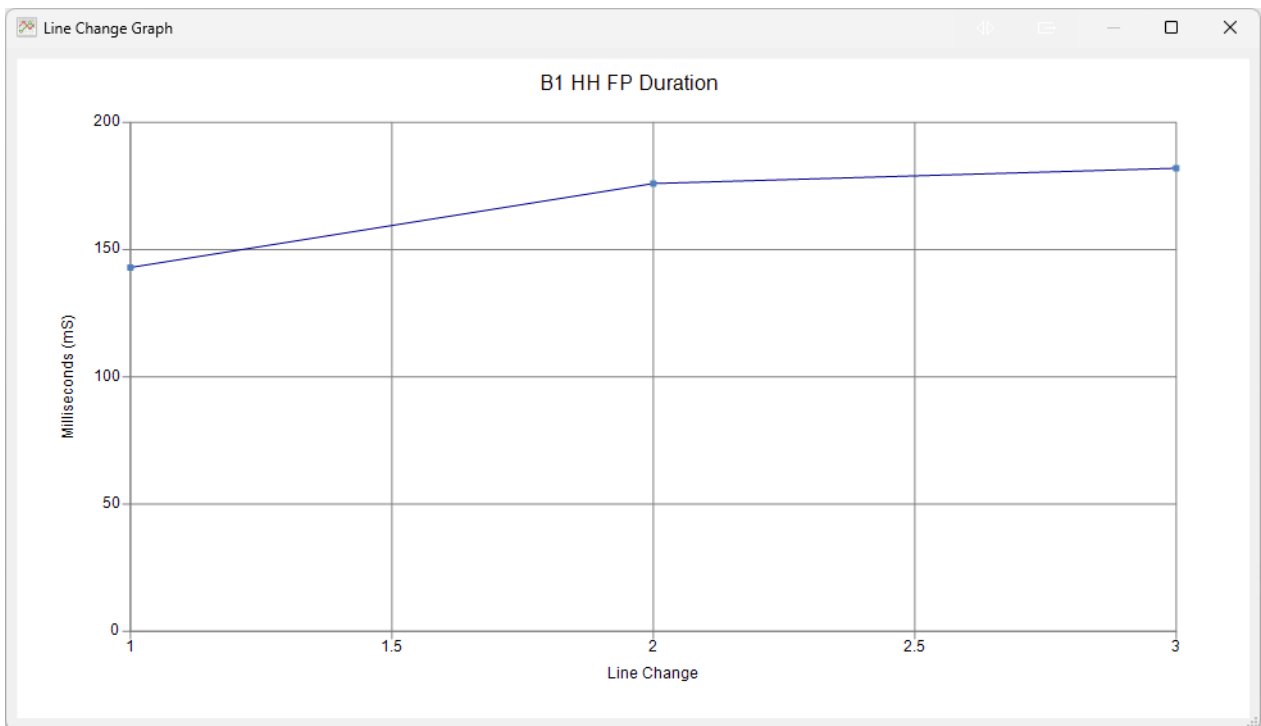
To help you quickly visualize line changes for a single set of data the Line by Line Analysis spreadsheet enables you to graph up to 10,000 data points, or changes, on a single graph.



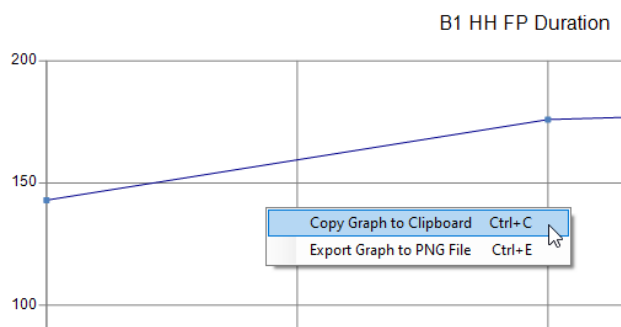
	B1 HH FP Offset	B1 HH FP Duration	B2 HH FP Onset	B2 HH FP Offset	B2 HH FP Duration	B3 C
1	13,418.00	143.00				
2	18,619.00	176.00				
3	24,007.00	182.00				

In the spreadsheet view highlight the line you are interested in by clicking and dragging over a range of data.

Then either right click and select Graph Selected Line Changes, or press G on the keyboard.



As can be seen in this example each selected data point has been plotted on a simple line graph. In this case the duration a response key was held down has been plotted.



If you wish to copy the line graph to the clipboard, or export it to a PNG file, right click and select Copy or Export.

Alternatively press CTRL+C for copy or CTRL+E for export.

10.4. Advanced use of the T-Pad Data Analyzer: Automatically Timelocking Data Captured by an Experiment Generator such as Gorilla

When you capture timing and event data using the T-Pad alongside your own Experiment Generator in Mode 3 as described earlier in section 9. a BBTK format .RTL file is stored on your PC. The .RTL file can be analyzed by hand as described above using the Data Analyzer.

As the .RTL (real-time log) file is created a second .TPE (T-Pad Event) file is also generated. The .TPE file stores real-time events timestamped from the T-Pads independent hardware timer alongside the timings reported by the Real-Time Clock (RTC) in the PC. The PC's RTC is the clock that is available to your Experiment Generator software and forms the basis of where its own timestamps originate.

It is important to remember that timings recorded by your own Experiment Generator are unlikely to be accurate as when operating in Mode 3 (Serial and Keyboard HID) it is simply using the T-Pad as a standard USB keyboard to record keystrokes. Its own timings are pulled from the PC's RTC via software when your Experiment Generator thinks they should be recorded and not when events happen in the real world.

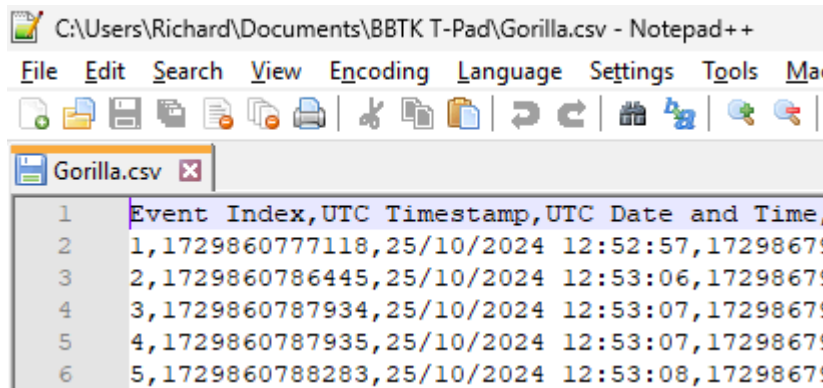
Although the PC's internal RTC is extremely precise and accurate your Experiment Generator may not pull an accurate timings from it, i.e. at the correct point in time. Think of an Atomic Clock which is extremely precise, but if you look at it at the wrong point in time your timing will be inaccurate despite the fact that it is precise. Precision is not accuracy. Most Experiment Generators claim millisecond precision and not millisecond accuracy. Precision in this context is simply the precision or units of time timings are reported in, i.e. milliseconds.

By storing exact hardware timings from the T-Pad alongside timings from the PC's own clock it is possible to match events recorded by your own Experiment Generator software to the exact timings recorded by the T-Pad. Simply put, it enables you to attempt to timelock events from your own software's data file to the stream of events recorded by the T-Pad. This then allows you to work out the exact RT's from the T-Pad's hardware based timings.

Note: Currently Timelocking using the T-Pad Data Analyzer only supports Gorilla format Comma Separated Value (CSV) data files. The online Experiment Generator Gorilla can be accessed here: <https://gorilla.sc/>.

THE TIMELOCKING FEATURE IS CURRENTLY EXPERIMENTAL AND THESE EXAMPLES ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

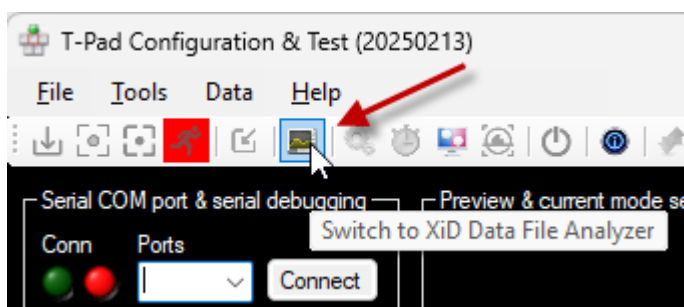
To be able to timelock events recorded by your own Experiment Generator to those recorded by the T-Pad it is assumed that your software has recorded a timestamp in a given format from the PC's RTC clock, i.e. Coordinated Universal Time (UTC) format (https://en.wikipedia.org/wiki/Coordinated_Universal_Time). Gorilla does this by default.



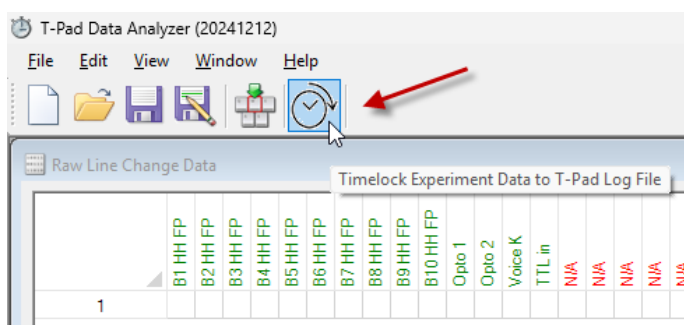
In the case of Gorilla CSV data files column 2, UTC Timestamp, is used as the software based timestamp

If you are using another Experiment Generator, e.g. PsychoPy, you can manually save the UTC into this column in your data file using script as your experiment runs and collects and stores its own timing data.

To timelock both sets of data files you need to load them into the Data Analyzer's timelock screen.



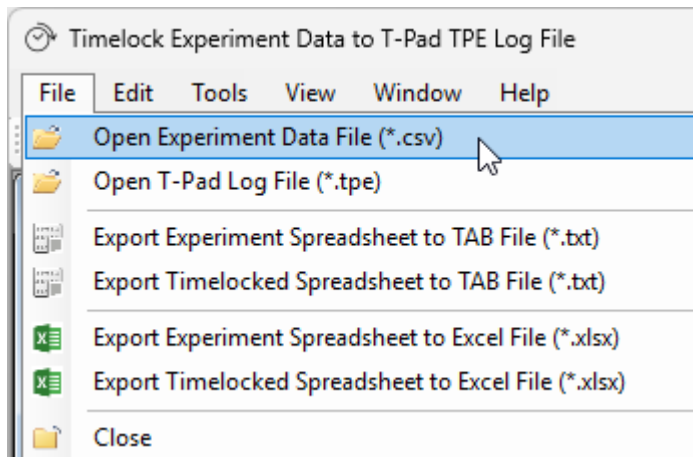
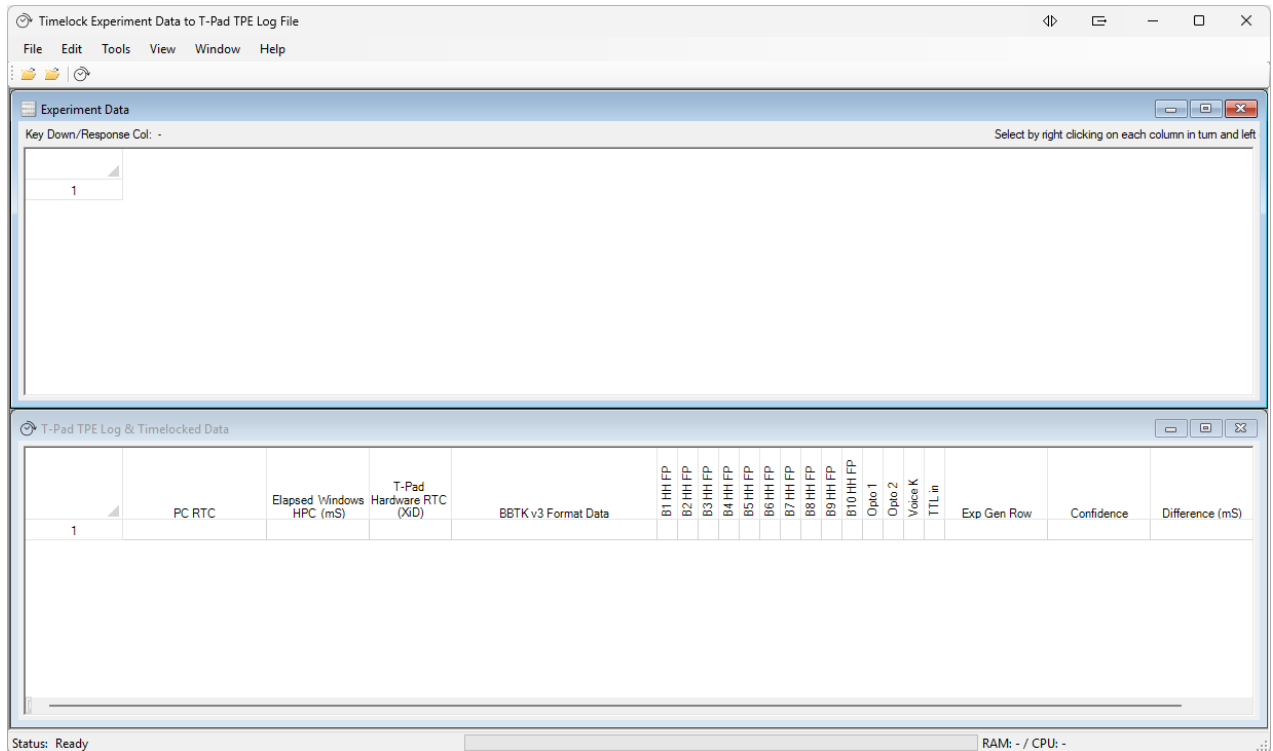
To do this start the Data Analyzer by clicking on it toolbar button from the T-Pad Configuration & Test App.



Once the Data Analyzer starts click on the timelock toolbar button.

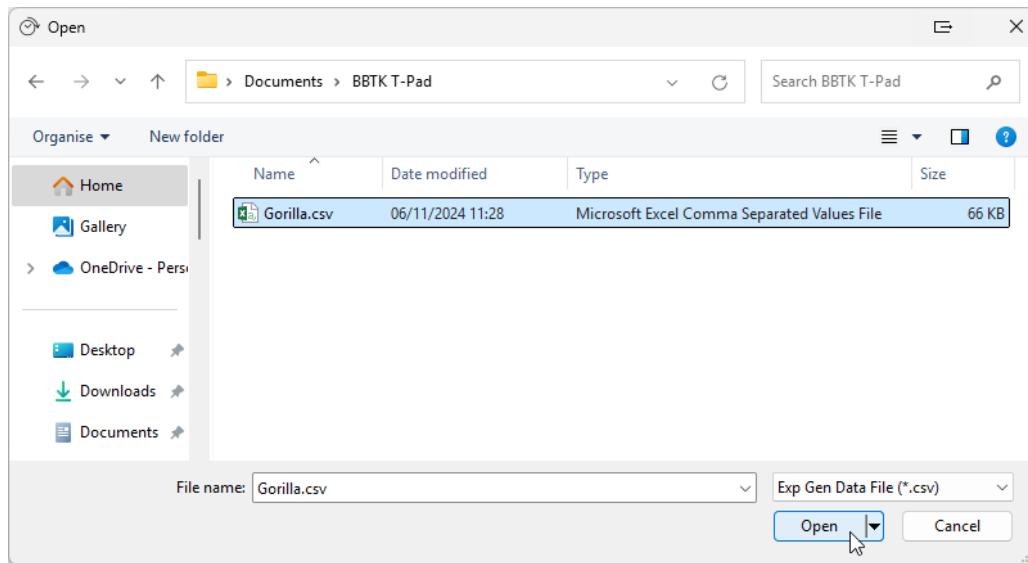
The top MDI spreadsheet is where the Gorilla CSV data file will be previewed when loaded and the lower the T-Pad TPE file.

The lower MDI spreadsheet is also where timelocked data will be displayed once both data files are loaded and timelocked.

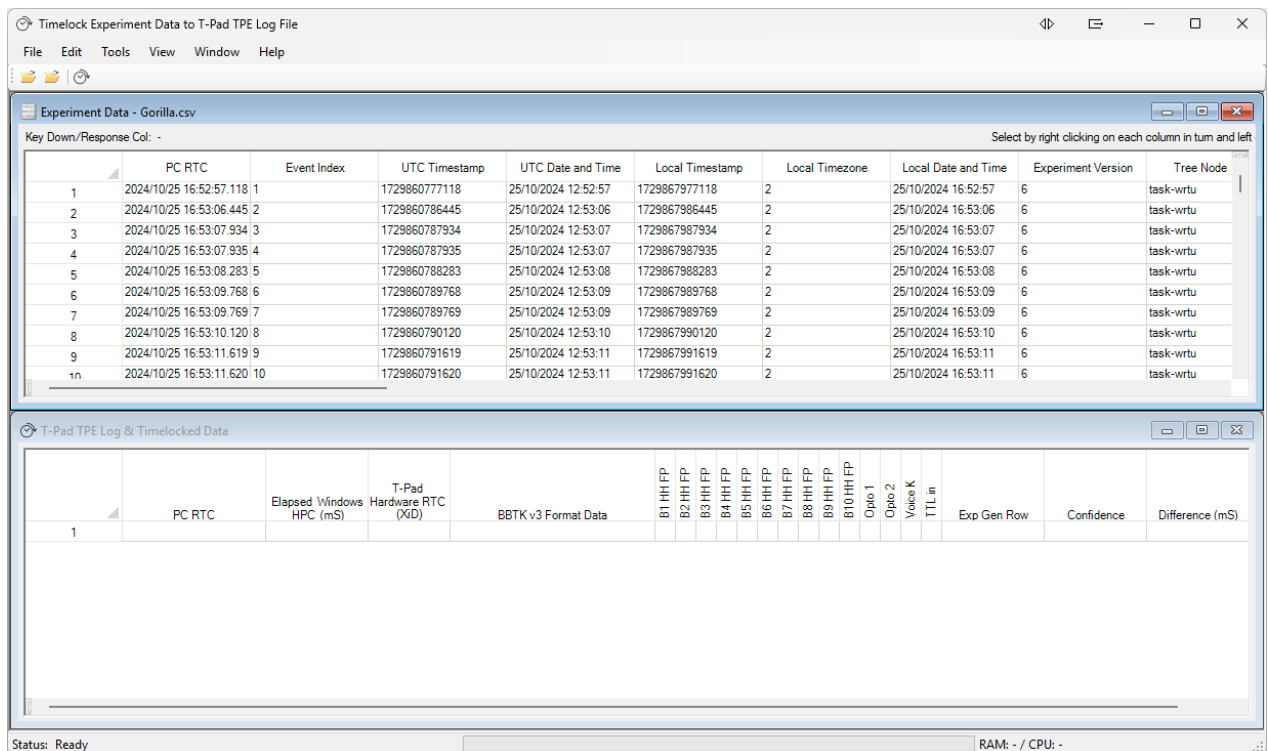


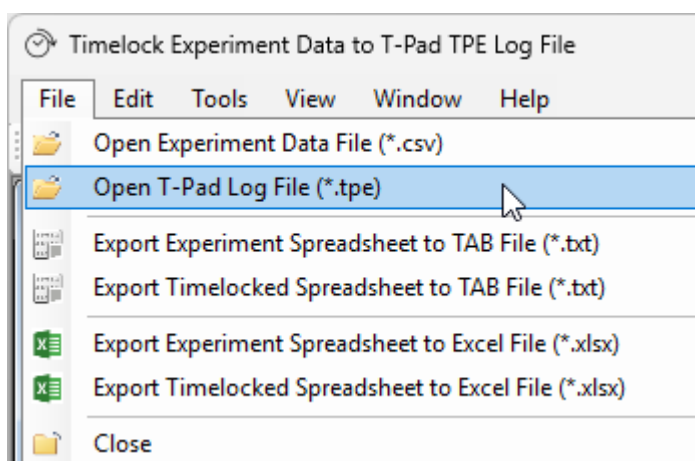
To load the Gorilla data file click on File|Open Experiment Data File (*.csv)

Select the file you wish to load.



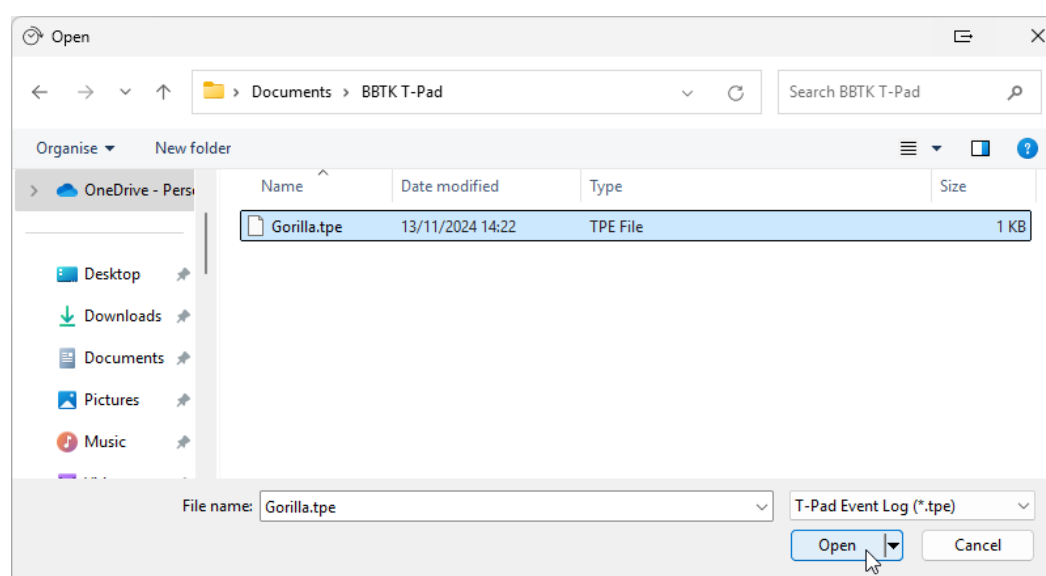
Once loaded the data file will be previewed. Note that the second column, UTC Timestamp, will be converted to a PC RTC column by the Data Analyzer.





To load the T-Pad data file click on File|Open T-Pad Log File (*.tpe)

Select the matching TPE file you wish to load.



Once loaded both data files will be previewed ready to attempt to timelock together. Note that there are blank columns in the lower TPE MDI window. This is where timelocked data will be displayed.

[illegible]

Experiment Data - Gorilla.csv

Key Down/Response Col: -

	Tree Node Key	Response			
1	task-wrtu	BEGIN			
2	task-wrtu				
3	task-wrtu				
4	task-wrtu		1501.203	1501.203	1501.203
5	task-wrtu	a	349.737	333.318	349.737
6	task-wrtu		1501.299	1501.299	1501.299
7	task-wrtu		1501.299	1501.299	1501.299
8	task-wrtu	a	352.096	350.036	352.096
9	task-wrtu		1501.269	1501.269	1501.269
10	task-wrtu		1501.269	1501.269	1501.269

Select Key Down/Response Col

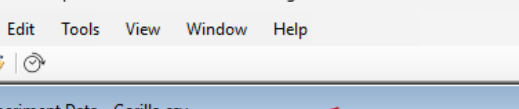
Copy Experiment Data Ctrl+E

Copy T-Pad Log & Timelocked Data Ctrl+F

Before you can timelock you need to select the column with the responses in you are interested in.

In this example the column headed Response which stores the keystrokes which came from the T-Pad's keyboard HID.

To do this left click on the column heading. Then right click and click on Select Key Down/Response Col from the flyout menu.



Timelock Experiment Data to T-Pad TPE Log File

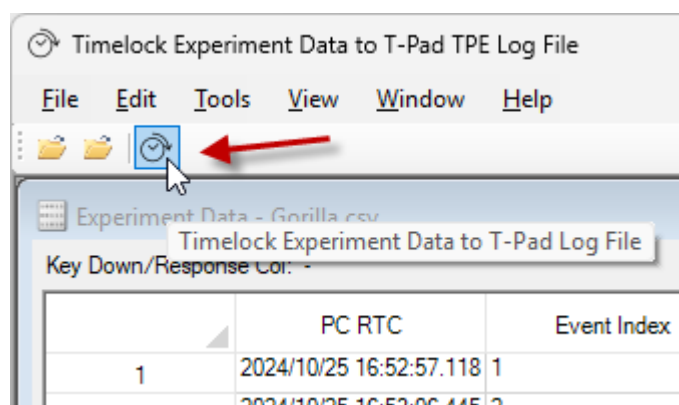
File Edit Tools View Window Help

Experiment Data - Gorilla.csv

Key Down/Response Col: Response (9)

	Tree Node Key	Response	Onset Time
1	task-wrtu	BEGIN	
2	task-wrtu		9337.183
3	task-wrtu		1501.203
4	task-wrtu		1501.203
5	task-wrtu	a	349.737

Once the response column has been selected it will be shown along with the column offset.



To attempt to timelock the two data files click on the timelock toolbar button.

Once timelocking has finished the three columns that were empty will now be filled in if timelocking was successful, i.e. Exp Gen Row, Confidence and Difference (mS).

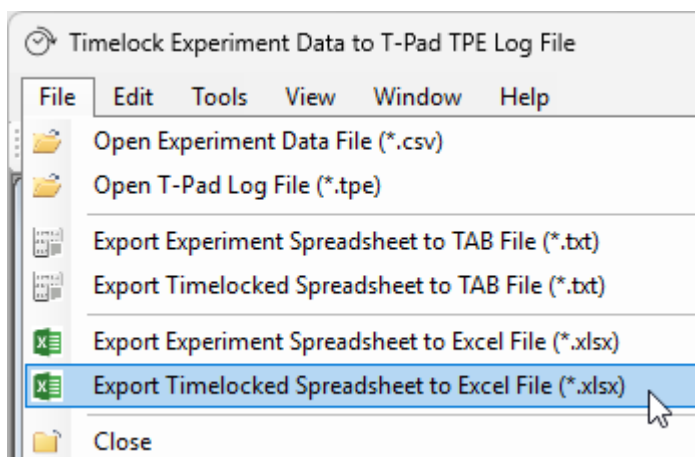
[illegible]

As can be seen from the sample above row 1 of the TPE T-Pad data file has been exactly matched with row 5 of the Gorilla data file as indicated in the Confidence column.

If an Opto had been attached while the experiment was in progress additional rows would be shown in the T-Pad TPE file spreadsheet for when the Opto was triggered.

By using the times from the T-Pad Hardware RTC (XiD) column this would enable you to calculate real-world RTs. To do this you would subtract the keydown XiD time in milliseconds from the Opto timestamp. This can be done in Microsoft Excel after exporting the TPE spreadsheet.

You can then compare the RT's recorded by Gorilla versus those recorded by the T-Pad.



To export the timelocked results select File|Export Timelocked Spreadsheet to Excel File.

This will export the lower spreadsheet which includes the additional timelocked information.

THE TIMELOCKING FEATURE IS CURRENTLY EXPERIMENTAL AND THESE EXAMPLES ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

11. Testing T-Pad Functionality and an Explanation of Each Available Mode

The T-Pad offers six operating modes to allow for maximum flexibility to help meet the requirements of researchers in most disciplines.

Modes range from simple where the T-Pad simply acts as a very fast second USB keyboard to complex where XiD serial protocol commands are streamed live from the T-Pad alongside independent hardware timestamps. Some modes combine simple keystroke operation with more complex XiD protocols.

The goal of the App is to let you test that the T-Pad is functioning correctly regardless of the mode you choose to use in your studies. In effect it is performing as would your own study, but displaying activity onscreen as it happens in near real time.

In the following worked examples each mode is tested using the App and where appropriate the same tests are repeated using a serial terminal and text editor. Each serial terminal and text editor example illustrates the exact same commands, or API, the App uses.

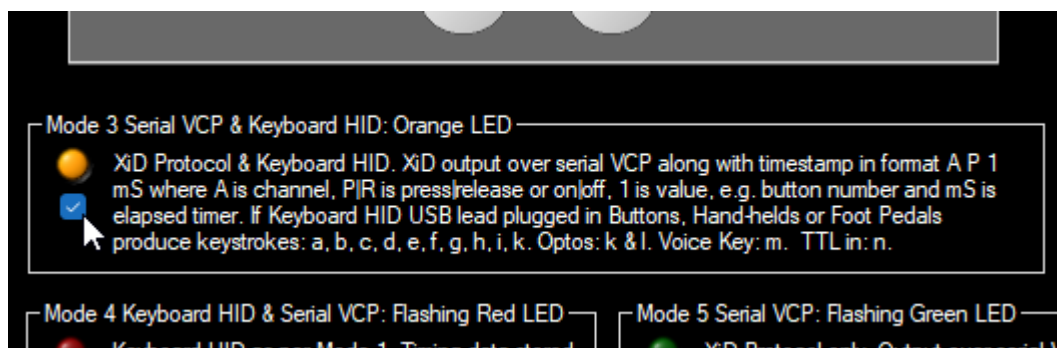
11.1. Testing in Mode 3 using the App – Serial VCP & Keyboard HID: Orange LED

Mode 3 is a hybrid operating mode where both the USB Keyboard HID lead and Serial lead are plugged in and both are live. Testing in Mode 3 enables you to quickly and easily check all aspects of T-Pad functionality.

For example in this mode when a button is pressed a keyboard keystroke is sent to the PC over the USB Keyboard HID as though a key on a keyboard had been pressed. Simultaneously an XiD protocol command and independent hardware timestamp is also sent over the Serial USB connection.

If your T-Pad operates correctly in Mode 3 over both Keyboard HID and Serial USB interfaces this means that it should function in all modes.

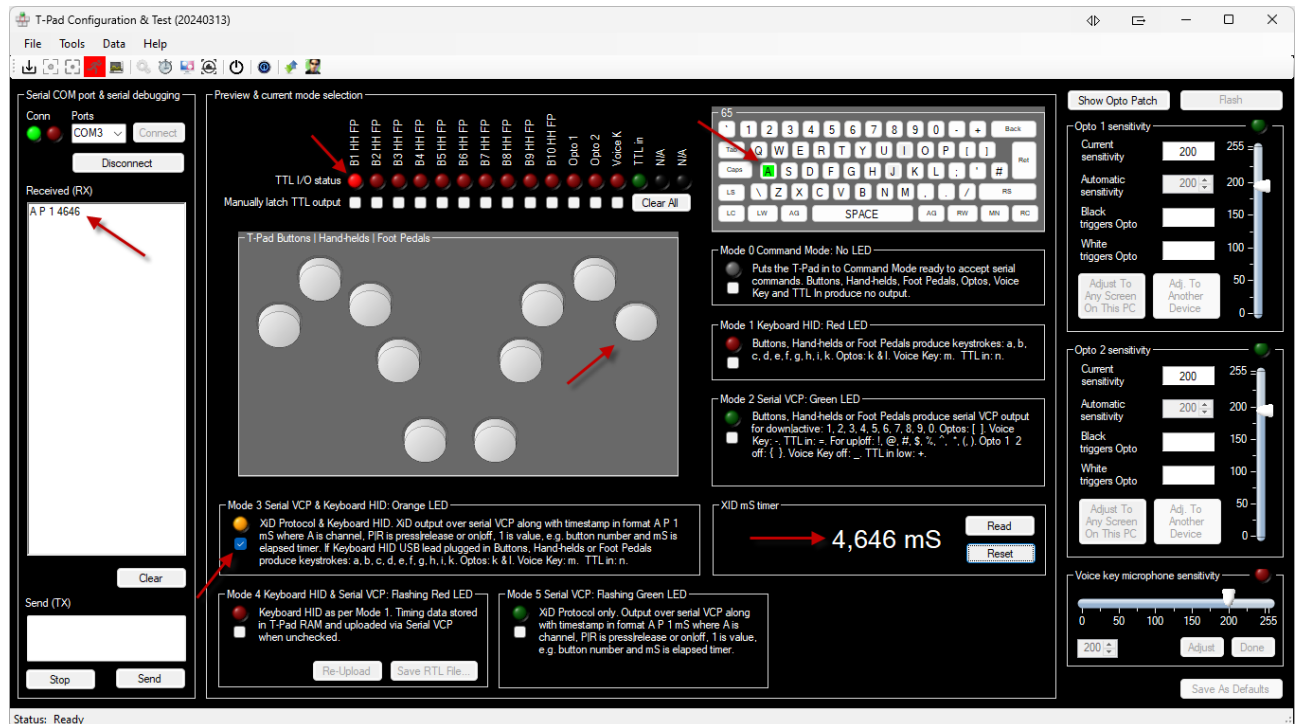
Once connected to check that everything is working correctly click on the Mode 3 checkbox:



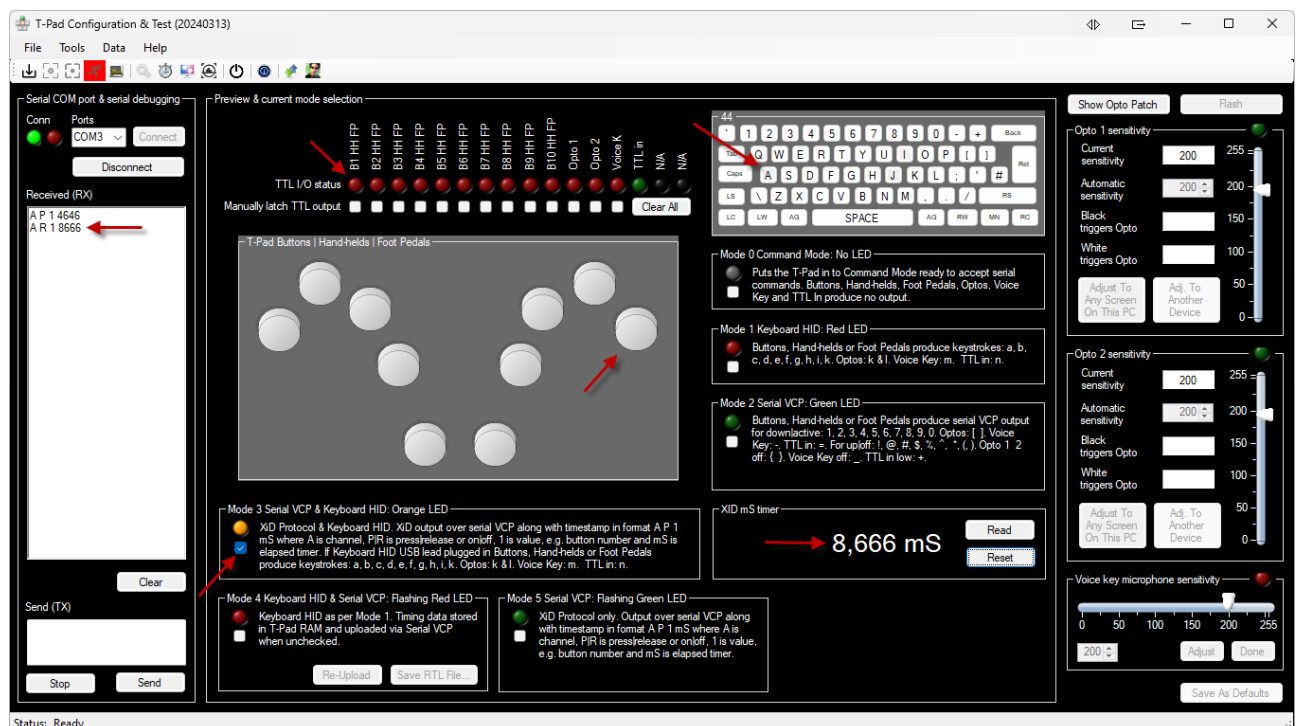
Then press button 1, Hand-held 1 or Foot Pedal 1.

As you press button 1 you will see the virtual button press down, a XiD command and timestamp sent to the Received (RX) window over serial showing a press, a virtual LED TTL

I/O light and an internal built-in millisecond elapsed time be displayed. The virtual on-screen keyboard will also show that the letter A has been pressed and held down as though pressed and held down on a standard USB keyboard.



As you release button 1 you will see the virtual button go up, an XiD command and timestamp sent to the Received (RX) window over serial showing a release, a virtual LED TTL I/O go out and an internal built-in millisecond elapsed time be displayed. The virtual on-screen keyboard will also show that the letter A has been let up as though let up on a standard USB keyboard.



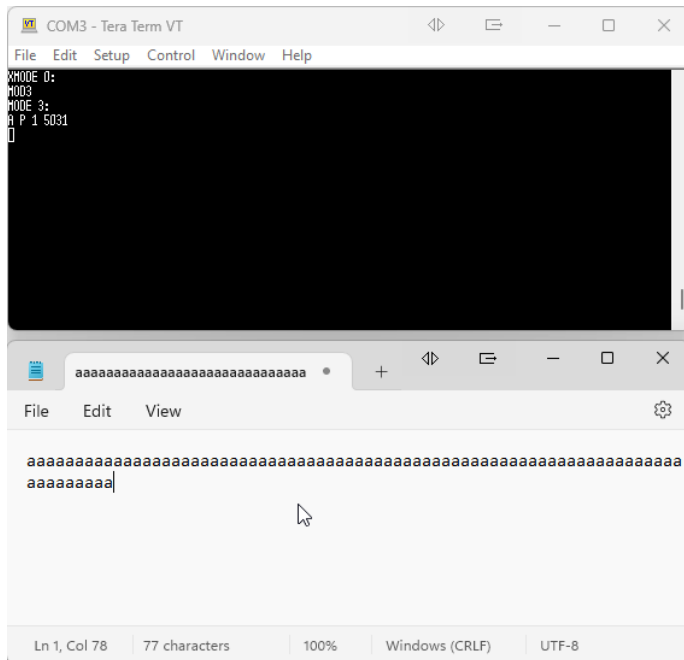
11.2. Testing in Mode 3 Using a Serial Terminal and a Text Editor – Serial VCP & Keyboard HID: Orange LED

If you wish to mirror testing in Mode 3 but this time in a serial terminal, e.g. Tera Term and a text editor, e.g. Notepad, this is easy to do as described below. In effect this simulates what you might do in any experiment generator or programmatically, but by hand so that you can see and understand the process more transparently.

For details of how to configure a typical serial terminal and for software please consult the Installation Guide, section 5. Checking Basic Functionality Using a Serial Terminal. All serial examples will be in Tera Term although any serial terminal could be used.

You should ensure that you close the BBTK T-Pad Configuration & Test App as you can ONLY have ONE Windows SERIAL APP RUNNING AT ANY ONE TIME under Windows or you will receive serial port locked errors.

Start Tera Term and connect to the VCP serial port the T-Pad is connected to. Next start Notepad and position them on the desktop as shown below so you can see both at the same time.



Click on Tera Term so that it has focus and then press capital X on your keyboard.

The T-Pad will respond with MODE 0: This is known as command mode.

Type MOD3 and press return.

The T-Pad will respond with MODE 3: This sets the T-Pad to Mode 3.

Click on Notepad so that it has focus. This is because you want the subsequent key presses and releases be sent to Notepad and not to Tera Term. Tera Term will receive serial output from the T-Pad even when it does not have focus.

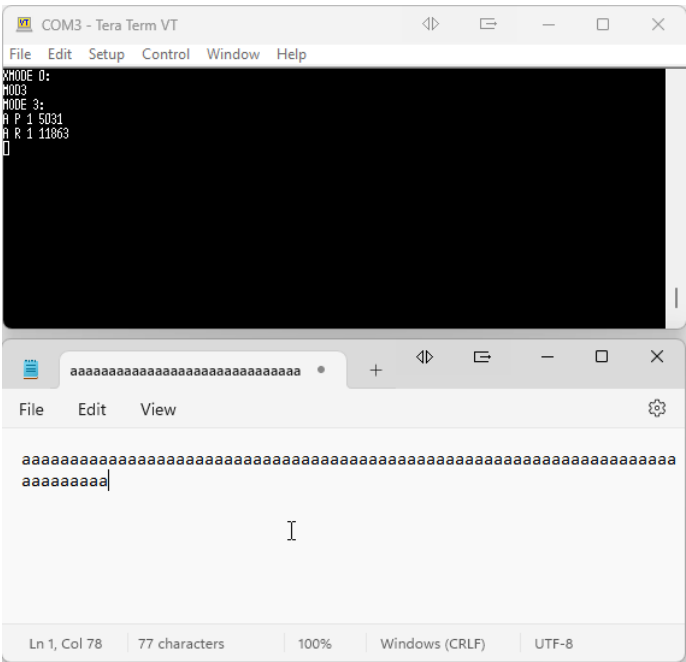
Press button 1 on the T-Pad and hold it down. As in the App the XiD command will be sent over serial showing the button down along with the elapsed timestamp, e.g. A P 1 5031.

A P 1 5031

translates to:

Channel [A] buttons | [P]ress | [1] button 1 | [5031] elapsed millisecond timer

As button 1 is held down you will see the letter a repeat in Notepad as though you had pressed the letter a and held it down on a standard USB keyboard. In most software including Experiment Generators this will be counted as a key down, i.e. a key press, but with no release.



Once you release button 1 a second XiD command will be sent over serial showing which button has been released along with the button up time, e.g. A R 1 11863 in this case.

A R 1 11863

translates to:

Channel [A] buttons
[R]elease
[1] button 1
[11863] elapsed millisecond timer

The key, i.e. a will also be released in Notepad as though you had released the letter a on a standard USB keyboard.

In most software including Experiment Generators this will be counted as a key being released, i.e. a key up.

As you can see the T-Pad App simply decodes the XiD serial and displays any HID key presses using a keyboard preview. In effect the App is doing exactly what your experiment would do when handling XiD commands and/or key presses.

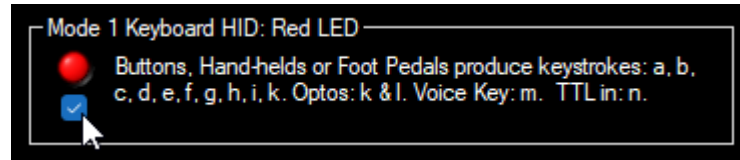
Each button, Opto, Voice Key or TTL In when pressed or active produces the following HID keyboard key presses.

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
HID Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n

11.3. Testing in Mode 1 using the App – Keyboard HID: Red LED

Using Mode 3 in the previous example you should have already confirmed that the Keyboard HID interface to the T-Pad is working correctly, but it is worth checking each mode to familiarize yourself with how the T-Pad operates.

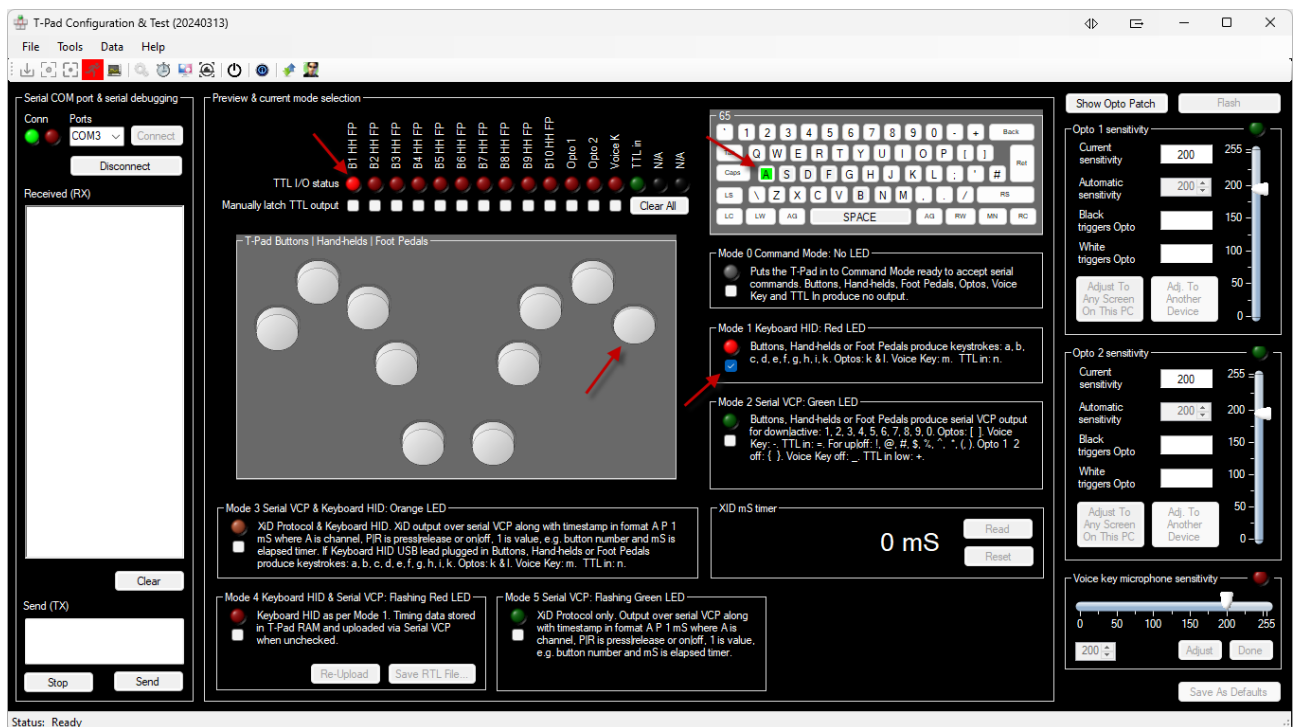
To check that Mode 1 works as expected click on the Mode 1 checkbox:



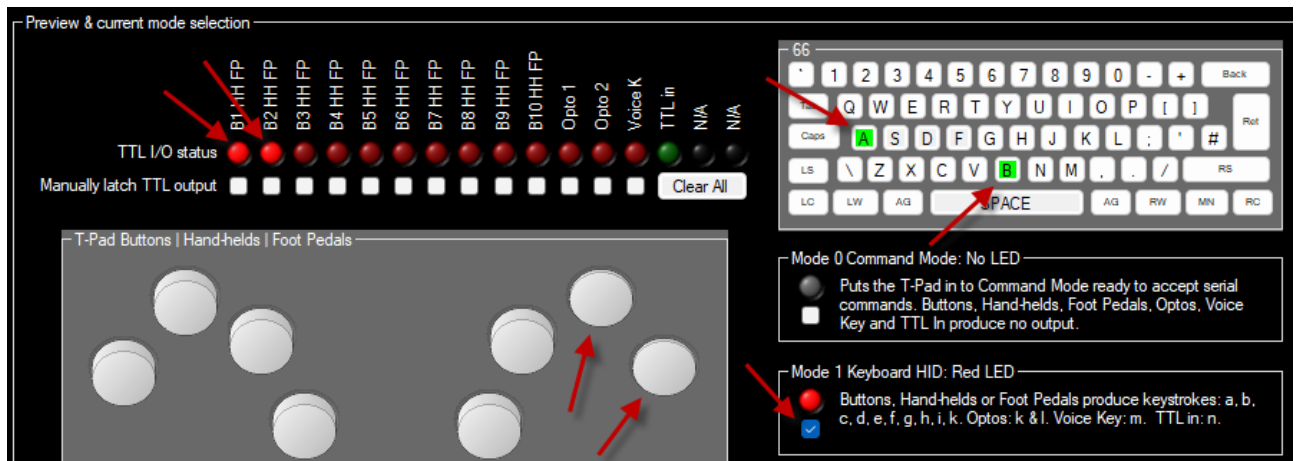
Then press button 1 & 2, Hand-held 1 & 2 or Foot Pedal 1 & 2 together. Press button 1 first and then button 2 slightly afterwards.

As you press button 1 you will a virtual button presses down followed by a second virtual press as you press button 2. Two virtual LED TTL I/O's will light. The virtual on-screen keyboard will also show that the letters a and b have been pressed as though pressed and held down on a standard USB keyboard.

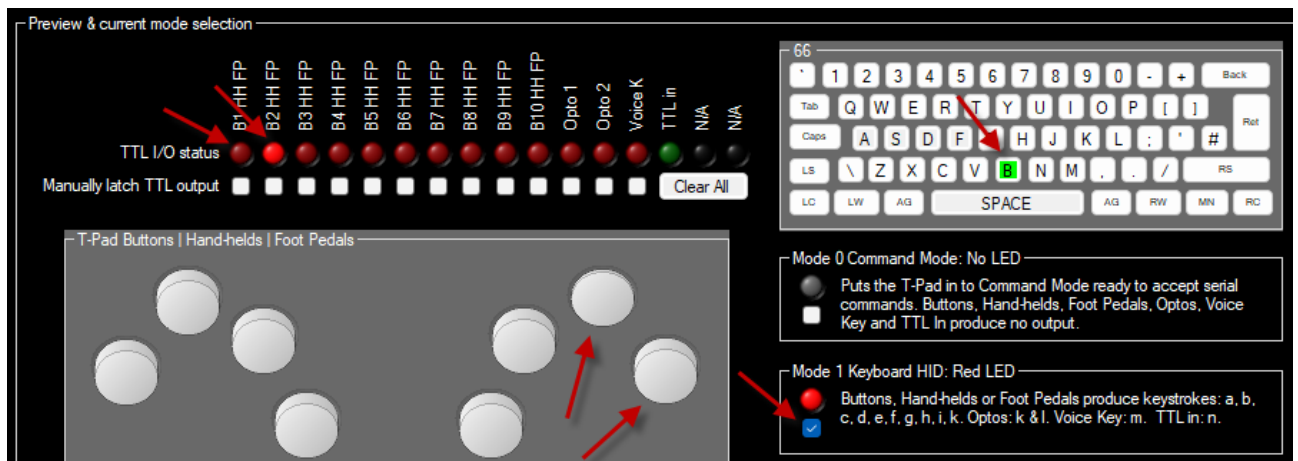
As you press button 1 you will see it previewed as below.



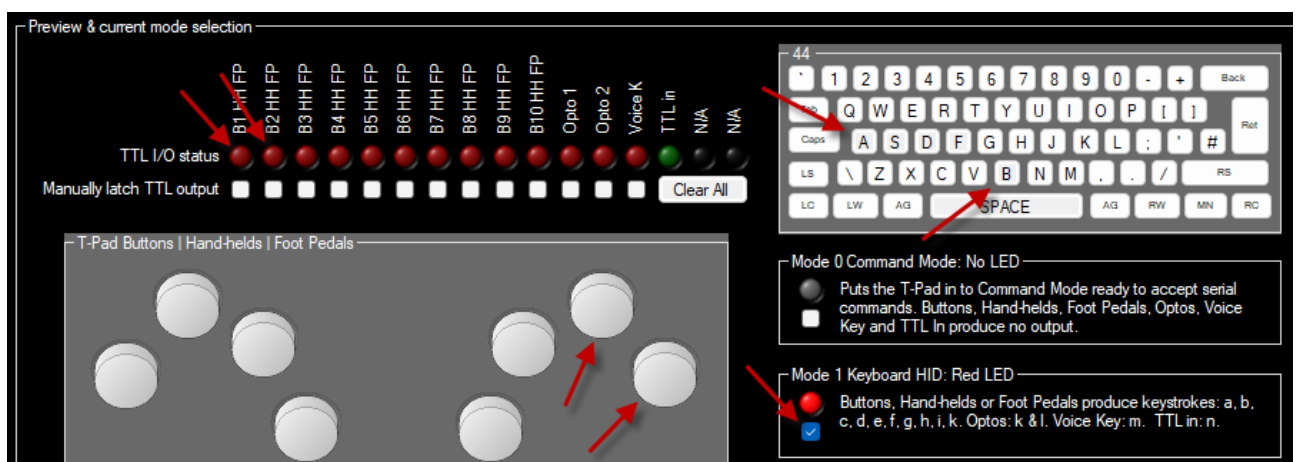
As you press button 2 at the same time this will be previewed along with button 1, i.e. key a for button 1 and key b for button 2.



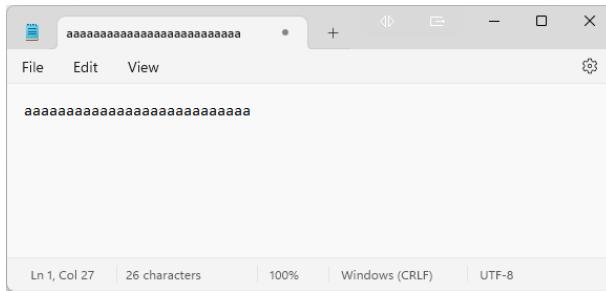
If you release button 1 you will see the letter a released as if the key was let up on a standard USB keyboard. The red TTL LED for button 1 will go out.



If you then release button 2 you will see letter b released as if the key was let up on a standard USB keyboard. The red TTL for button 2 will go out.

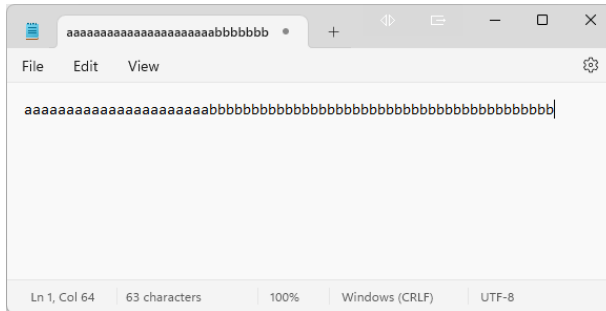


11.4. Testing in Mode 1 using a text editor – Keyboard HID: Red LED

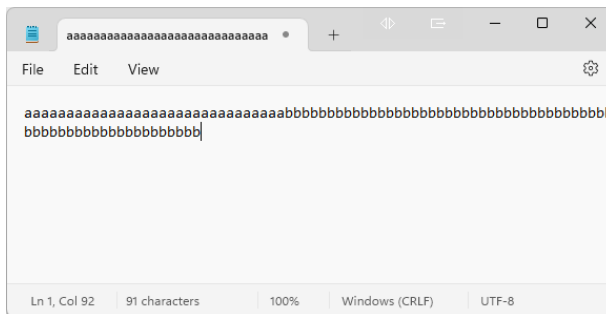


To replicate testing Mode 1 in the T-Pad App start Notepad, ensure it has the focus and then press and hold down button 1.

The letter a will then repeat as it is held down.



Then press button 2. The letter b will then repeat. Button 1 is still held down and the letter a is still flagged as being down to your computer but it is not displayed in Notepad.



When you release button 1, the letter b is still shown as being held down and repeated. The letter a is flagged to your computer as being released even though it is not displayed in Notepad.

Once button 2 is released letter b is no longer repeated as it is flagged as being released to your computer.

For checking and previewing keyboard HID activity in more depth you can use our online tool.

<https://www.blackboxtoolkit.com/tools/tltokeys/linechecker.html>



As you might expect this does the exact same thing as our T-Pad App, but live in a Web Browser.

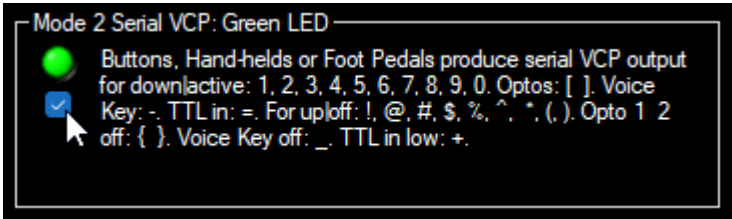
Each button, Opto, Voice Key or TTL In when pressed or active produces the following HID keyboard key presses.

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
HID Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n

11.5. Testing in Mode 2 – Serial VCP: Green LED using the App

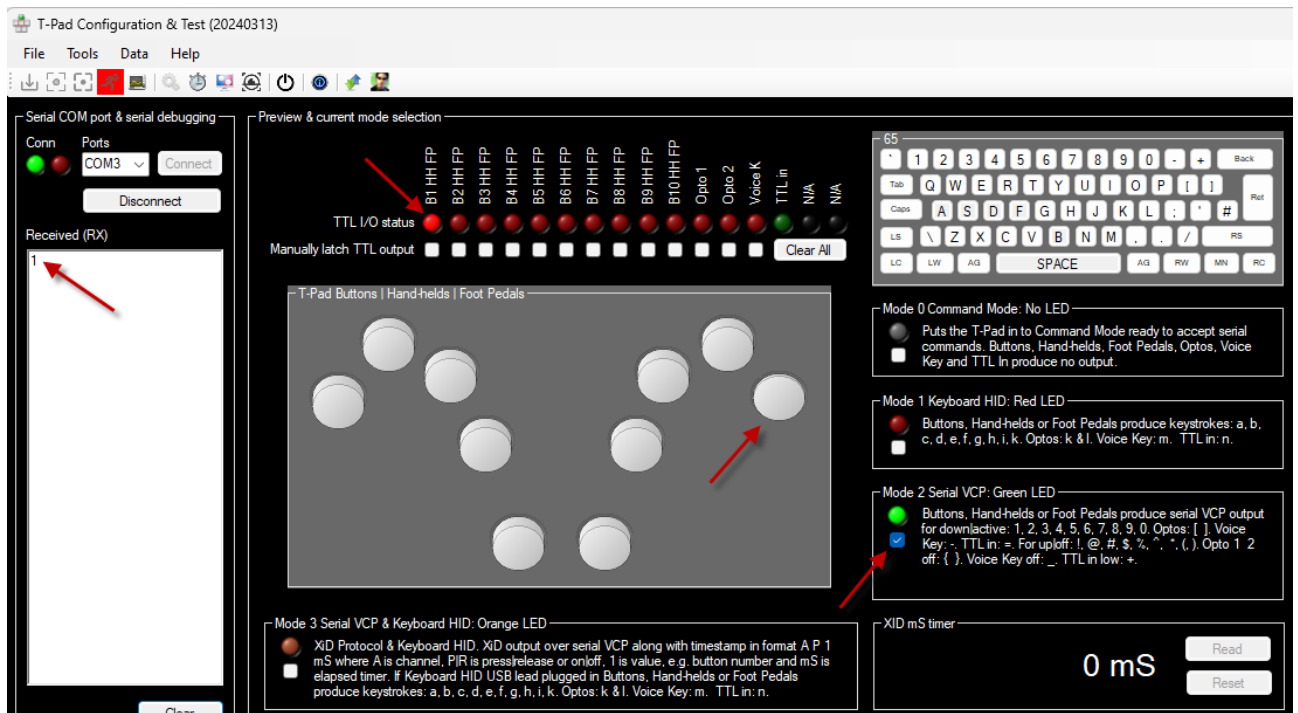
Mode 2 is a purely serial mode where each button press sends a unique character over serial to represent a given button down and another unique character to represent that button being released.

To check that Mode 2 works as expected click on the Mode 2 checkbox:

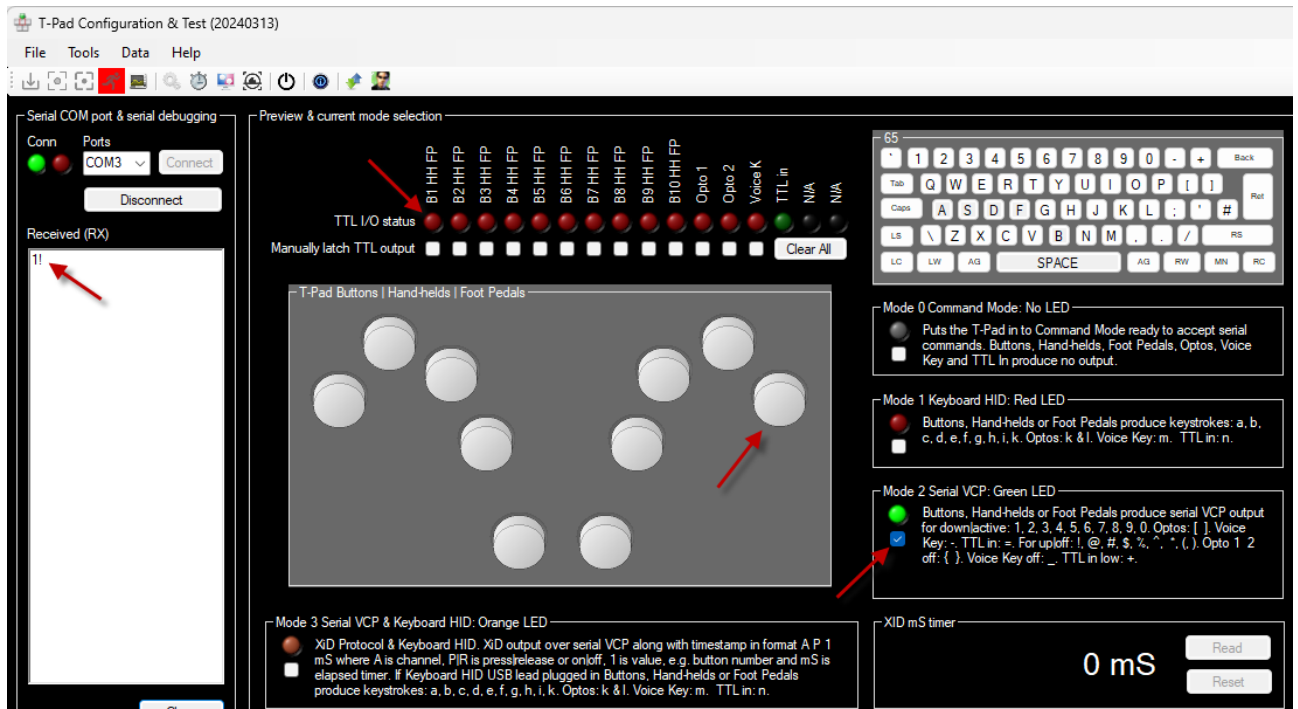


Then press button 1, Hand-held 1 or Foot Pedal 1.

As you press button 1 you will see the virtual button press down and the character '1' sent to the Received (RX) window over serial showing a press and a virtual LED TTL I/O light.



When you release button 1 you will see the virtual button go up and the character '!' sent to the Received (RX) window over serial showing a release and the virtual LED TTL I/O go out.

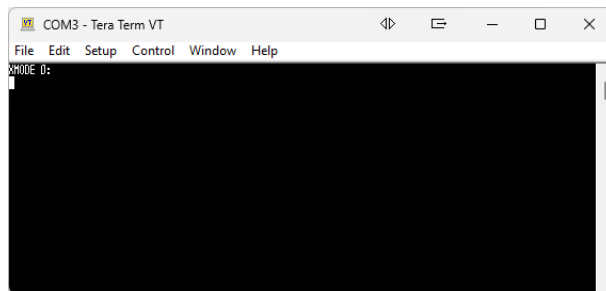


11.6. Testing in Mode 2 using a serial terminal and a text editor – Serial VCP: Green LED

To replicate testing Mode 2 but this time in a serial terminal, e.g. Tera Term, this is easy to do as shown below. In effect this simulates what you might do in any experiment generator or programmatically, but by hand so that you can see and understand the process more transparently.

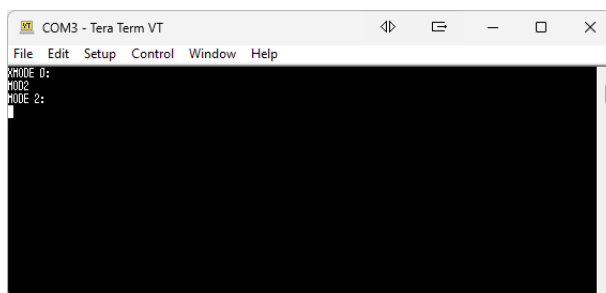
You should ensure that you close the BBTK T-Pad Configuration & Test App as you can ONLY have ONE Windows SERIAL APP RUNNING AT ANY ONE TIME under Windows or you will receive serial port locked errors.

Start Tera Term and connect to the VCP serial port the T-Pad is connected to.



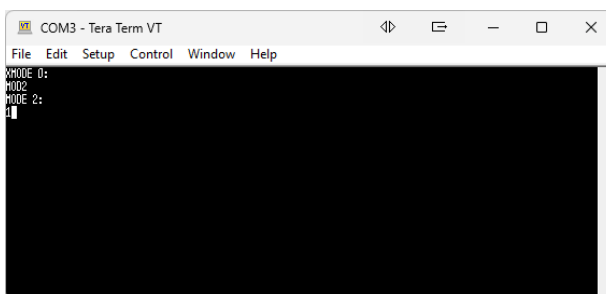
To replicate testing Mode 2 in the T-Pad App start Tera Term and then press capital X on your keyboard.

The T-Pad will respond with MODE 0: This is known as the command mode.



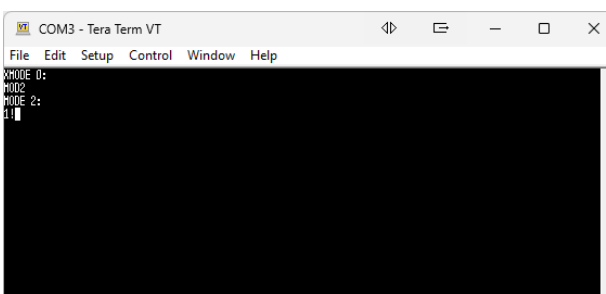
Type MOD2 and press return.

The T-Pad will respond with MODE 2: This sets the T-Pad to Mode 2.



Press button 1 and the character 1 will be sent over serial from the T-Pad.

Note that even if you hold button 1 down the character 1 will only be sent once.



When button 1 is released the character ! will be sent over serial from the T-Pad.

Again this will only be sent once.

Each button, Opto, Voice Key or TTL In when pressed or active produces the following characters sent over serial.

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
Down or Active	1	2	3	4	5	6	7	8	9	0	[	]	-	=
Up or Non-active	!	@	#	\$	%	^	&	*	(	)	{	}	_	+

11.7. Testing in Mode 4 using the App – Keyboard HID & Serial VCP: Flashing Red LED

Mode 4 is a hybrid Keyboard HID mode combined with Serial VCP and works in a similar way to Modes 1 and 3, but without the instant transmission of serial XiD commands to confirm which buttons have been pressed or sensors activated together with the elapsed timestamp.

In this mode the T-Pad functions as a standard USB keyboard but stores event marks and elapsed timestamps in its onboard internal memory and then uploads all event and timing data at the end of your experiment when you tell it to upload.

This means that your study only needs to concentrate on recording key presses and trial numbers rather than constantly reading from serial. Then at the end of all your trials you can instruct to the T-Pad to upload all data over serial and store all data in a CSV file to time-lock to the keystroke data you recorded.

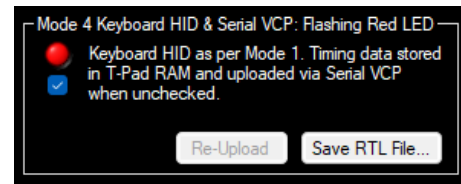
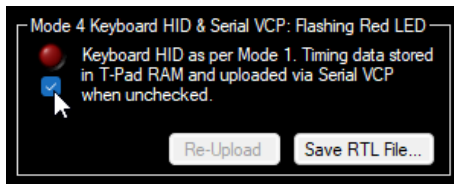
Each button, Opto, Voice Key or TTL In when pressed or active produces the following HID keyboard key presses.

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
HID Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n

For example, if you had a study which was made up of 10 trials, each of which presented a visual stimulus, you could use Opto 1 to event mark the onset of each trial/visual stimulus using a white block. This would mean you would know exactly when each trial was started and a visual stimuli presented. You would also know when a button was pressed as this would be recorded by the T-Pad. Recorded data for all 10 trials would then be uploaded at the end of your study. All you would need to do while the study was being run is record USB keyboard key presses.

It should be noted that TTL event marks for button presses or other activity are sent in real-time as normal alongside keyboard HID keystrokes.

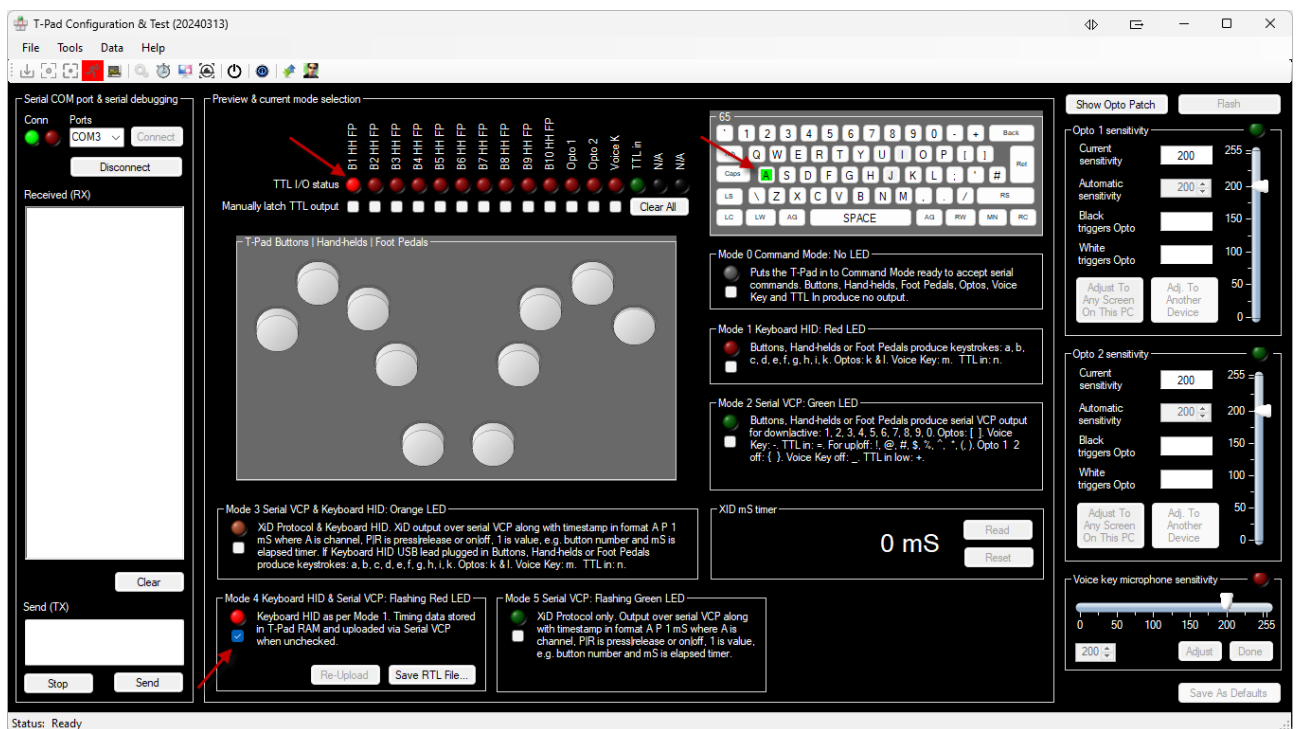
To check that Mode 4 works as expected click on the Mode 4 checkbox. As you switch to Mode 4 the hardware timer in the T-Pad will be reset and the virtual and physical LED on the rear of the T-Pad will flash red:



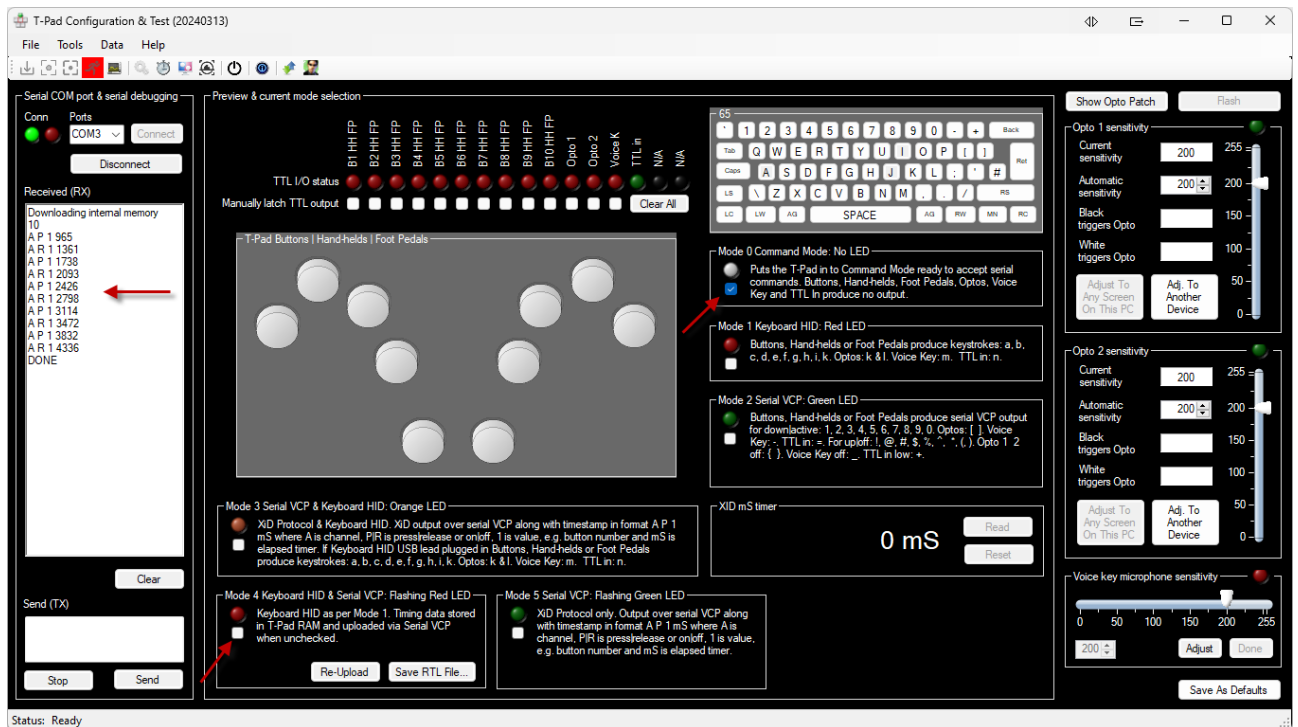
Then repeatedly press and release button 1, Hand-held 1 or Foot Pedal 1.

As you press button 1 you will see the virtual button press down and a virtual LED TTL I/O light. When you release the button will go up and the LED TTL I/O light will be unlit.

This process will continue as you press and release button 1.



Once you have pressed and released button 1 as many times as you wish click on the Mode 4 tick box again so that it is unticked. At this point all data will then be uploaded over serial as a stream of XiD timing data in event order and displayed in the serial Received (RX) window as shown below.



In this example we can see that once the data has been uploaded there has been 5x button 1 press and release events or 10 pieces of data in total.

Note that 'Downloading internal memory' is sent first followed by the number of data points to follow in XiD format. Once all data has been uploaded 'DONE' is sent to signal the end of the data stream.

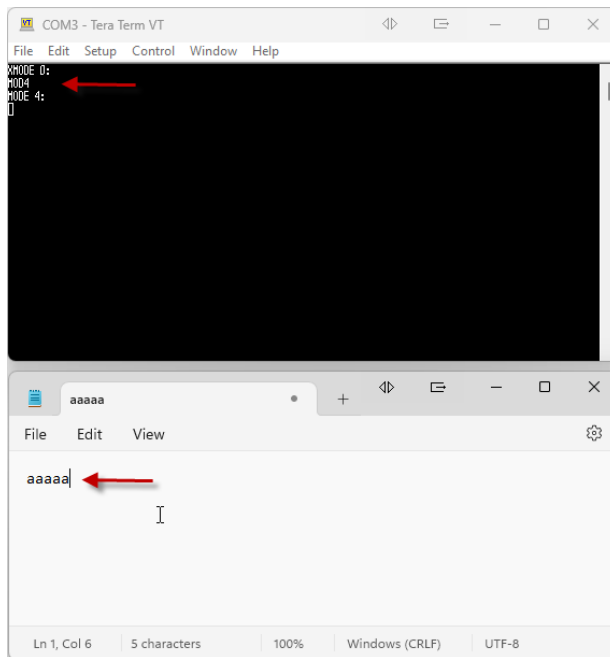
A maximum of 2,000 data points or events can be recorded by the T-Pad in each study before you need to upload your data and go back into Mode 4. Each data point could count as a button down, button up, Opto on, Opto off and so on...

11.8. Testing in Mode 4 using a serial terminal and a text editor – Keyboard HID & Serial VCP: Flashing Red LED

If you wish to replicate testing Mode 4 but this time in a serial terminal, e.g. Tera Term, this is easy to do as shown below. In effect this simulates what you might do in any experiment generator or programmatically, but by hand so that you can see and understand the process more transparently.

You should ensure that you close the BBTK T-Pad Configuration & Test App as you can ONLY have ONE Windows SERIAL APP RUNNING AT ANY ONE TIME under Windows or you will receive serial port locked errors.

Start Tera Term and connect to the VCP serial port the T-Pad is connected to. Next start Notepad and position them on the desktop as shown below so you can see both at the same time.



Click on Tera Term so that it has focus and then press capital X on your keyboard.

The T-Pad will respond with MODE 0: This is known as the command mode.

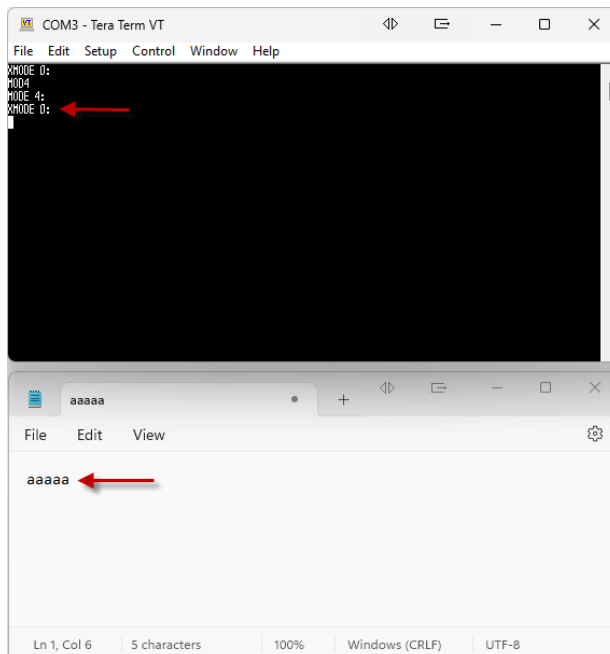
Type MOD4 and press return.

The T-Pad will respond with MODE 4: This sets the T-Pad to Mode 4.

Click on Notepad so that it has focus. This is because you want the subsequent key presses and releases be sent to Notepad and not to Tera Term. Tera Term will receive serial output from the T-Pad even when it does not have focus.

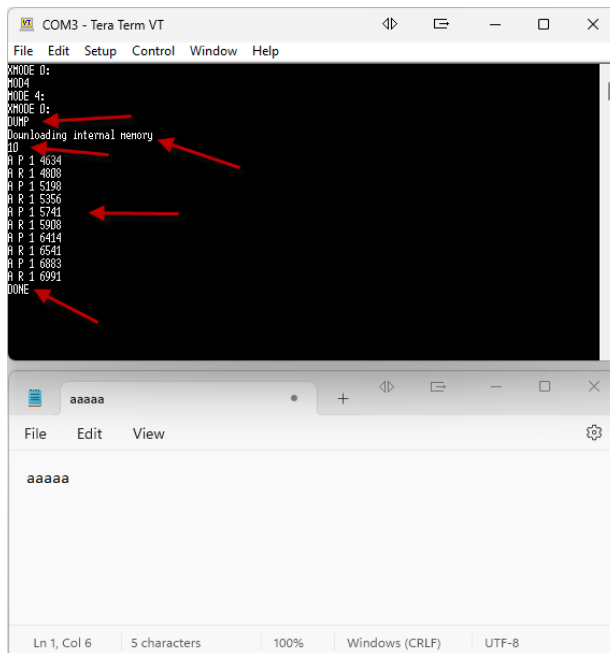
If you press and release button 1 five times five letter a key presses will be shown in Notepad.

Note that in Mode 4 nothing will be sent live over serial from the T-Pad.



To stop collecting button presses and other sensor activity click on Tera Term to set the focus to it and press capital X on your keyboard to return to Mode 0.

Note that no other activity will be shown in Notepad.



You can retrieve the data at any time once in Mode 0 by typing DUMP and pressing return.

The T-Pad will then upload event and timing data.

```

Downloading internal memory
10
A P 1 4634
A R 1 4808
A P 1 5198
A R 1 5356
A P 1 5741
A R 1 5908
A P 1 6414
A R 1 6541
A P 1 6883
A R 1 6991
DONE

```

The sequence of data will begin with 'Downloading internal memory' followed by the number of data points to follow, in this case 10.

In the example data sequence we can see five button presses and releases in XiD format. To signify the end of the data sequence DONE will be displayed.

If we look at the first press and release of button 1 we can decode the XiD as follows:

A P 1 4634

translates to:

Channel [A] buttons | [P]ress | [1] button 1 | [4634] elapsed millisecond timer

A R 1 4808

translates to:

Channel [A] buttons | [R]elease | [1] button 1 | [4808] elapsed millisecond timer

In this case we can see that button 1 has been held down for 174 mS, i.e. 4808 - 4634. Remember that due to button debouncing -20 mS will need to be subtracted, meaning that the button down duration is actually 174 - 20 = 154 mS

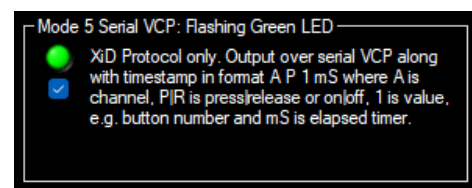
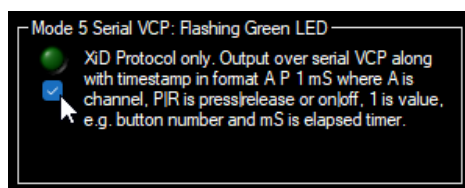
11.9. Testing in Mode 5 using the App – Serial VCP: Flashing Green LED

Mode 5 is a pure serial VCP mode and sends live XiD commands from the T-Pad as each button is pressed or other sensor is activated.

Note that even if the USB Keyboard HID lead is connected no keyboard keystrokes will be sent from the T-Pad.

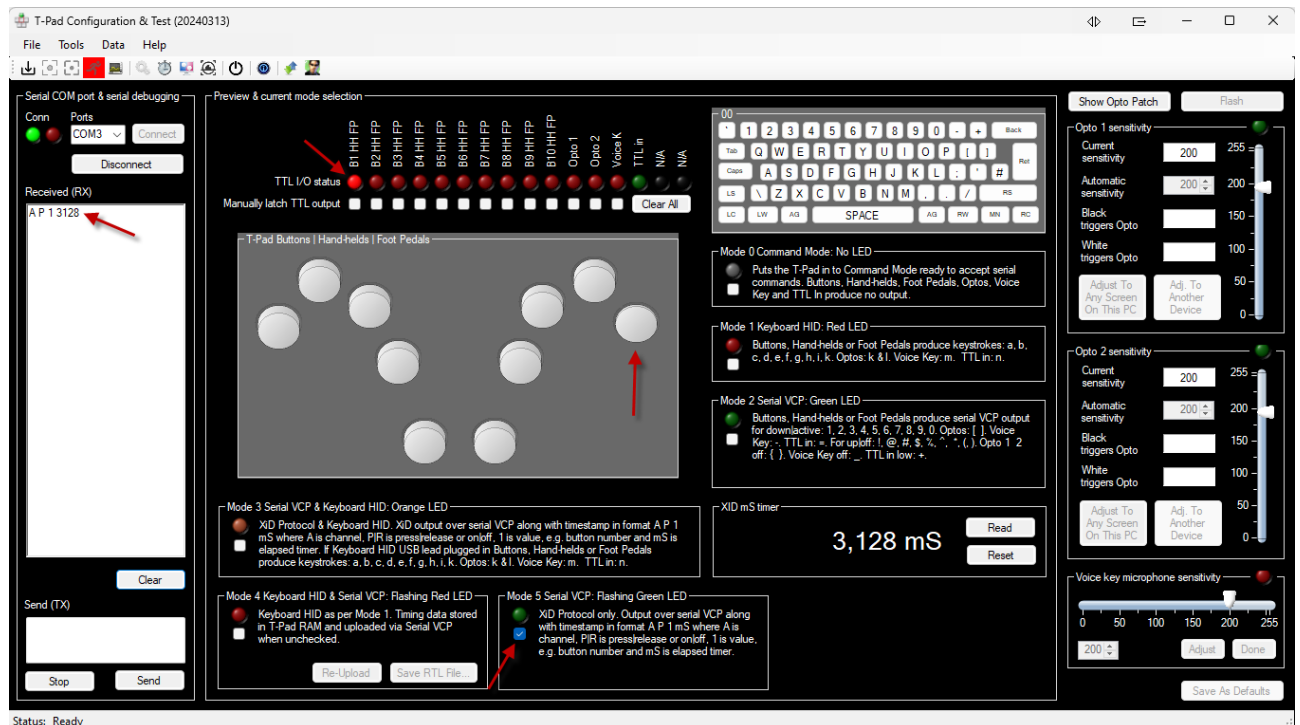
It should be noted that TTL event marks for button presses or other activity are sent in real-time as normal alongside keyboard HID keystrokes.

To check that Mode 5 works as expected click on the Mode 5 checkbox. As you switch to Mode 5 the hardware timer in the T-Pad will be reset and the virtual and physical LED on the rear of the T-Pad will flash green:

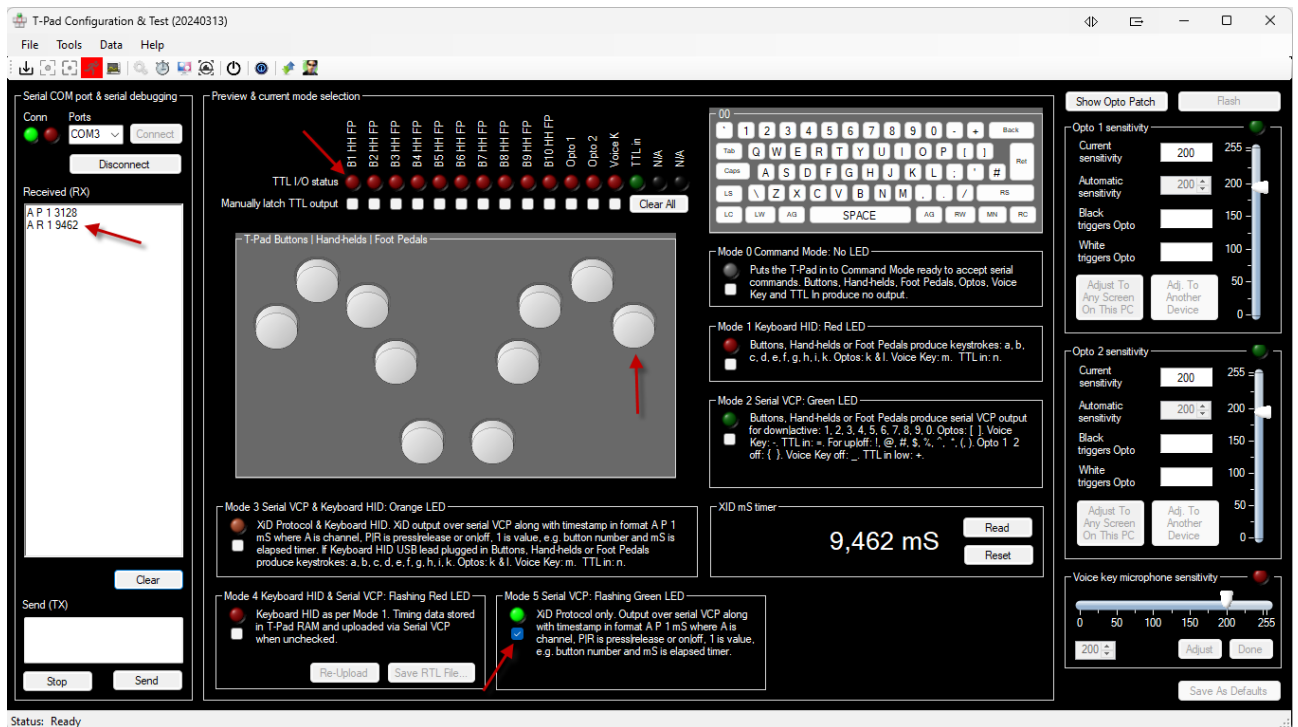


Then repeatedly press and release button 1, Hand-held 1 or Foot Pedal 1.

As you press button 1 you will see the virtual button press down and a virtual LED TTL I/O light. In the serial Received (RX) window you will see the XiD command sent from the T-Pad in near real-time.



When you release the button will go up and the LED TTL I/O light will be unlit as the button up XiD command is sent over serial.



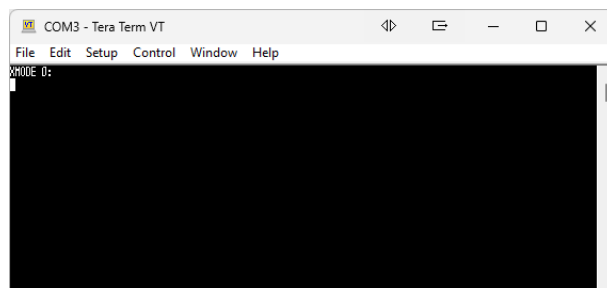
This process will continue as you press and release button 1...

11.10. Testing in Mode 5 using a serial terminal – Serial VCP: Flashing Green LED

If you wish to replicate testing Mode 5 but this time in a serial terminal, Tera Term, this is easy to do as shown below. In effect this simulates what you might do in any experiment generator or programmatically, but by hand so that you can see and understand the process more transparently.

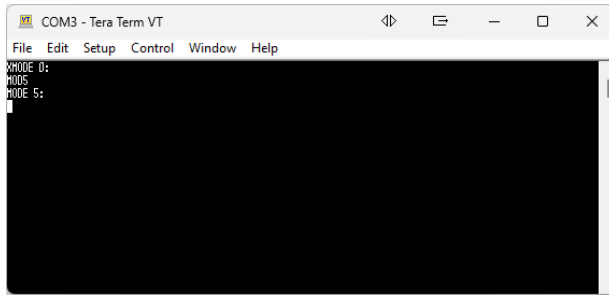
You should ensure that you close the BBTK T-Pad Configuration & Test App as you can ONLY have ONE Windows SERIAL APP RUNNING AT ANY ONE TIME under Windows or you will receive serial port locked errors.

Start Tera Term and connect to the VCP serial port the T-Pad is connected to.



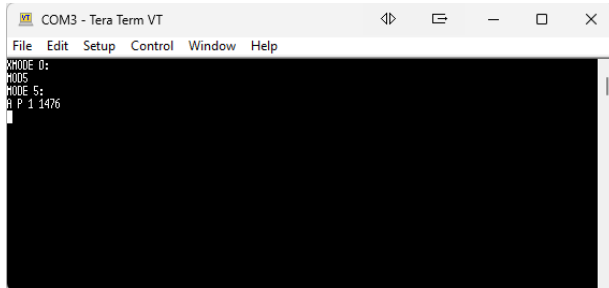
To replicate testing Mode 5 in the T-Pad App start Tera Term and then press capital X on your keyboard.

The T-Pad will respond with MODE 0: This is known as the command mode.



Type MOD5 and press return.

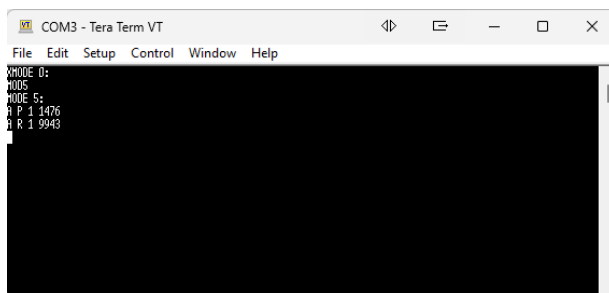
The T-Pad will respond with MODE 5: This sets the T-Pad to Mode 5.



Press button 1 and the XiD command will be sent over serial from the T-Pad.

In this example:

A P 1 1476



When button 1 is released the XiD command will be sent over serial from the T-Pad.

In this example:

A R 1 9943

If we look at the first press and release of button 1 we can decode the XiD as follows:

A P 1 1476

translates to:

Channel [A] buttons | [P]ress | [1] button 1 | [1476] elapsed millisecond timer

A R 1 9943

translates to:

Channel [A] buttons | [R]elease | [1] button 1 | [9943] elapsed millisecond timer

In this case we can see that button 1 has been held down for 8467 mS, i.e. 9943 - 1476. Remember that due to button debouncing -20 mS will need to be subtracted, meaning that the button down duration is actually 8467 - 20 = 8447 mS

Each button, Opto, Voice Key or TTL In when pressed or active produces an XiD command which is sent over serial.

Note that the millisecond timestamp will be millisecond accurate as that is the timestamp recorded by the T-Pads hardware timer, but as the T-Pad is sending a number of characters back to your computer this takes some time over serial.

11.11. Serial Transmission Time From the T-Pad to Your PC

Typically the T-Pad can send around 115,200 characters/bytes per second at 115,200 baud. That translates to one character or one byte around every 0.008 mS or 115 characters/bytes per millisecond.

So the first button press XiD command:

A P 1 3128

Is 12 characters or bytes if you include spaces and the invisible terminating characters, CRLF.

So that XiD command would take 12×0.009 mS or 0.108 mS to transmit to your PC.

However you should bear in mind that your PC and the software you use may not be able to receive and interpret or act on serial at this speed. Typically there will be some small overhead over and above this theoretical maximum transmission speed per XiD command received from the T-Pad.

It may also be that your software or Experiment Generator may block, or be blocking, whilst reading serial commands from the T-Pad. This is an advanced programming topic and can be researched by Googling, 'is serial blocking' for example.

In short this means that your own code, or Experiment Generator, may stop execution of subsequent code whilst it waits for the terminating CRLF on each XiD command sent from the T-Pad. This is so that it knows that an XiD command has been received.

If multiple XiD commands are sent one after another they may stack and depending on how your own code or Experiment Generator processes the serial stream that may take more time and prevent subsequent code executing until the stack is processed. This would be considered to be "blocking". "Non-blocking" serial code would not stop subsequent code executing as incoming serial is processed. Again this is an advanced programming topic that also encompasses multi-tasking and threading and can be researched by Googling, 'non-blocking serial'.

The T-Pad itself will keep collecting and timestamping event data in real-time ready to send over serial once the last XiD command has been sent and received by your computer. This is because the T-Pad is interrupt driven, has an independent millisecond accurate timer and makes use of advanced internal circular buffers and ultrafast RAM. In other words it is multi-threaded. It will not stop sending XiD commands to your computer, or throttle its output, as it has no flow control due to its intention to process events as quickly as possible.

11.12. Setting TTL Output Lines Using the T-Pad Configuration & Test App

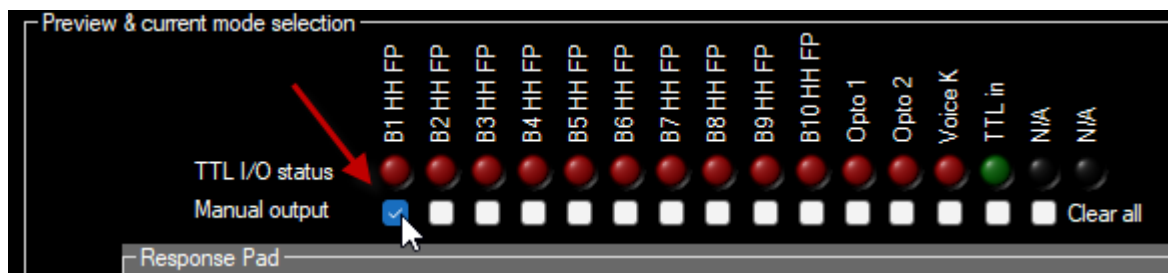
Two RJ45 ports on the rear of the T-Pad will mirror activity, i.e. press button 1 and TTL Out 1 will switch to high +5V and release button 1 and the line will fall to low 0V. The duration of the TTL signal will match the time the physical button is depressed, i.e. it will latch.



All other lines follow this pattern, i.e. Optos will remain high for the time that there is a white Opto patch displayed under the Opto-detector.

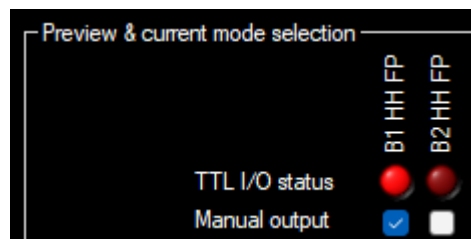
In addition to hardware mirroring of activity these lines can also be controlled programmatically using the T-Pad API, e.g. via PsychoPy.

To test they latch on/off in our App click on the relevant virtual check box:



Note that the LED will not be lit as no button has been pressed. All you are doing is manually turning on the TTL signal associated with button 1 being pressed.

If you physically press the button 1 the virtual LED will illuminate and the TTL signal will remain high, +5V.

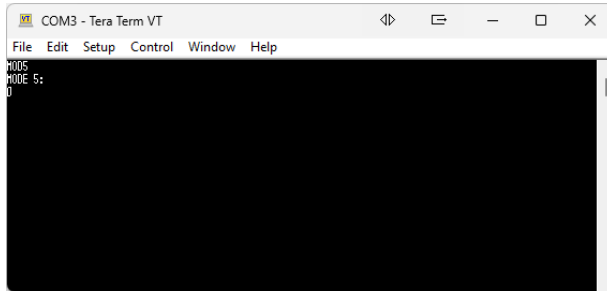


TTL Out line 1 will remain high even if you release button 1. To make it go low you will need click on the Clear All button or untick the check box.

11.13. Setting TTL Output Lines Using a Serial Terminal

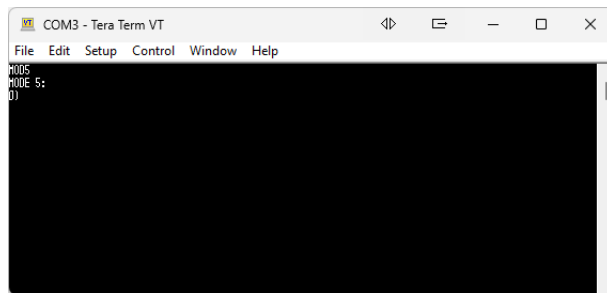
In all modes that use serial communication, i.e. Modes 2 through 5, you can turn any of the TTL Out lines on or off, high +5V or low 0V, by sending the appropriate serial character/byte to the T-Pad.

This is useful for event marking from your own Experiment Generator or custom code, e.g. on an EEG machines event marking channel.



In Mode 5 for example to turn the TTL line that mirrors button 10 being pressed you would send the character/byte 0.

Once you had sent 0, the line, i.e. pin 6 on RJ45 TTL I/O socket 2, would go active high or +5V and remain latched.



To turn the TTL line off for button 10 that mirrors the button being released you would send the character/byte).

Once you had sent), the line, i.e. pin 6 on RJ45 TTL I/O socket 2, would go low or 0V and remain at 0V.

You are free to use any TTL lines as you see fit, but remember that if you use a line that is already mirrored by a physical button installed in your T-Pad then if the button is pressed then the TTL line will go high +5V as it is connected at a hardware level. It will also go low 0V if released. In effect physical button presses or other activity override software set TTL line outputs.

For example, if your T-Pad only had four buttons installed, you would be free to use TTL lines for buttons 5-10 to signal other events, e.g. active high +5V on button 5 might signal the start or your experiment to an EEG machine and low 0V the end.

The individual character/bytes you need to send the T-Pad over serial to mirror a given button, Opto or Voice Key are shown below. Remember TTL lines latch once you have set them high.

	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key
High +5V	1	2	3	4	5	6	7	8	9	0	[	]	-
Low 0V	!	@	#	\$	%	^	&	*	(	)	{	}	_

The physical pinouts for the two RJ45 TTL sockets are shown below.

RJ45 – TTL I/O 1			RJ45 – TTL I/O 2		
Pin	Button/Sensor	TTL Line	Pin	Button/Sensor	TTL Line
1	B1	TTL Out 1	1	B5	TTL Out 5
2	B2	TTL Out 2	2	B6	TTL Out 6
3	B3	TTL Out 3	3	B7	TTL Out 7
4	B4	TTL Out 4	4	B8	TTL Out 8
5	Opto 1	TTL Out 11	5	B9	TTL Out 9
6	Opto 2	TTL Out 12	6	B10	TTL Out 10
7	Voice Key	TTL Out 13	7	TTL In 1	TTL In 1
8	Ground	N/A	8	Ground	N/A

11.14. BBTK USB TTL Module Emulation Mode

Mode 6 enables the T-Pad to mimic the basic event marking functionality of a physical BBTK USB TTL Module:



Designed as a parallel port replacement for simple event marking. Plug into a USB port on your PC and our USB TTL Module will appear as a Virtual Com Port (VCP). To event mark send two hex bytes from your experiment generator. Quickly and easily TTL event mark/TTL trigger with any experiment generator via standard one line serial commands, e.g. task events in E-Prime. Works out of the box with E-Prime, SuperLab, Presentation, Inquisit, PsychoPy or any other software that can read and write to a standard serial port.

When triggering from MRI, EEG, ERP, MEG systems wait for a sync pulse that has been converted to two incoming hex bytes. Think of the module as a USB to TTL adapter or TTL to USB interface.

<https://www.blackboxtoolkit.com/usbtll.html>

The real BBTK USB TTL Module acts as a USB parallel port replacement to allow basic TTL event marking/TTL triggers from any experiment generator, or other software, that supports a serial port. By mimicking this functionality this allows the T-Pad to event mark from your own code across 8x TTL output lines via the two RJ45 sockets. In addition it enables you to read the status of the first four buttons, two Optos, Voice Key and single TTL input line as though connected to a physical USB TTL Module.

Physical T-Pad TTL outputs for button 1 through 8 mirror USB TTL Module output lines 1 through 8. For example, if you send Hex bytes 01 to the T-Pad in emulation mode the physical TTL signal from the T-Pad is equivalent to pressing button 1 in T-Pad modes 1 through 5.

Note that pressing buttons does not automatically cause a physical TTL output signal to be generated and neither does an Opto activation, Voice Key or TTL In.

If you wish to output physical TTL signals for buttons 1-4 you would need to check for two serial bytes being sent FROM the T-Pad over serial as buttons were pressed and then send out the corresponding two bytes TO the T-Pad to turn them on, e.g. Hex bytes 01 through 0F. The same applies to Opto 1, Opto 2, Voice Key and TTL in.

That is, you need to check constantly for two Hex bytes coming FROM the T-Pad over serial and then act on them to manually trigger an event mark by sending the correct two byte code back TO the T-Pad. Remember in Mode 6, button presses are not automatically linked to a corresponding TTL output. This needs to be done manually in your own code should you wish to TTL event mark an event. You may also choose to use any of the 8x TTL output lines however you wish to event mark, i.e. you don't have to use them to TTL event mark events such as buttons. In short they can be used to event mark anything as you are controlling event marking from your own code and you can use any of the 8x TTL lines to represent anything you like.

Remember although the T-Pad hardware is millisecond accurate and helps you produce simple event marks in your experiments it cannot automatically correct for any mistimings that are inherent within the experiment generator or other hardware you use. In short it does not guarantee that your event marks or triggers are accurate as your experiment generator is responsible to all timings and all it provides is a simple USB-TTL interface between your computer and third party equipment.

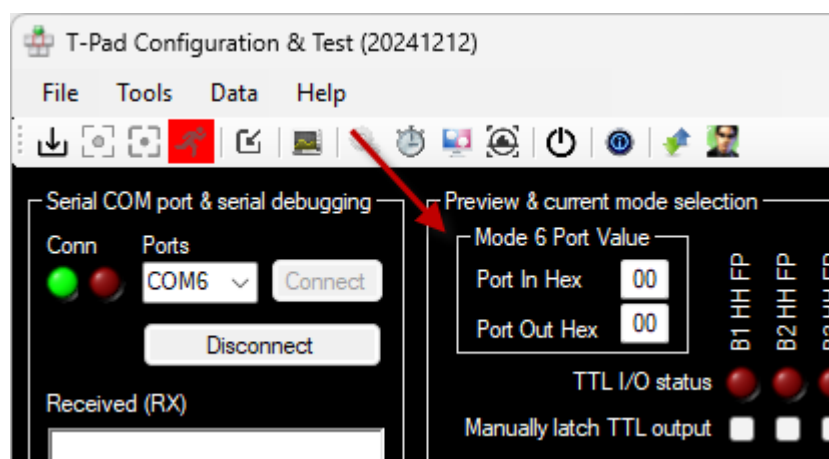
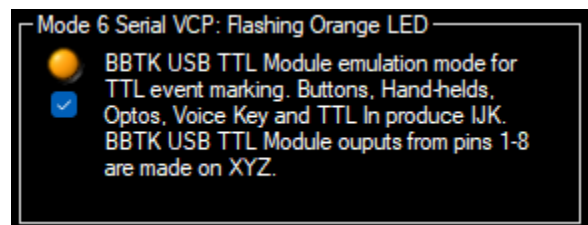
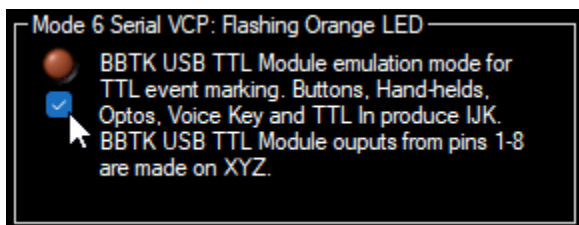
In USB TTL Module emulation mode the internal hardware clock of the T-Pad is not used and accurate timing cannot be guaranteed. All timings are the sole responsibility of your own software and code. For example, if your experiment generator presents an image and does not produce an event mark in a timely fashion, your OS intervenes, or an anti-virus check runs, your event marks will remain inaccurate regardless of the quality of the event marking device.

11.14.1. Testing in Mode 6 using the App – Serial VCP: Flashing Orange LED

Mode 6 enables the T-Pad to mimic the basic event marking functionality of a BBTK USB TTL Module. Mode 6 is a pure serial VCP mode and sends two byte port values in Hex FROM the T-Pad as each button is pressed or other sensor is activated.

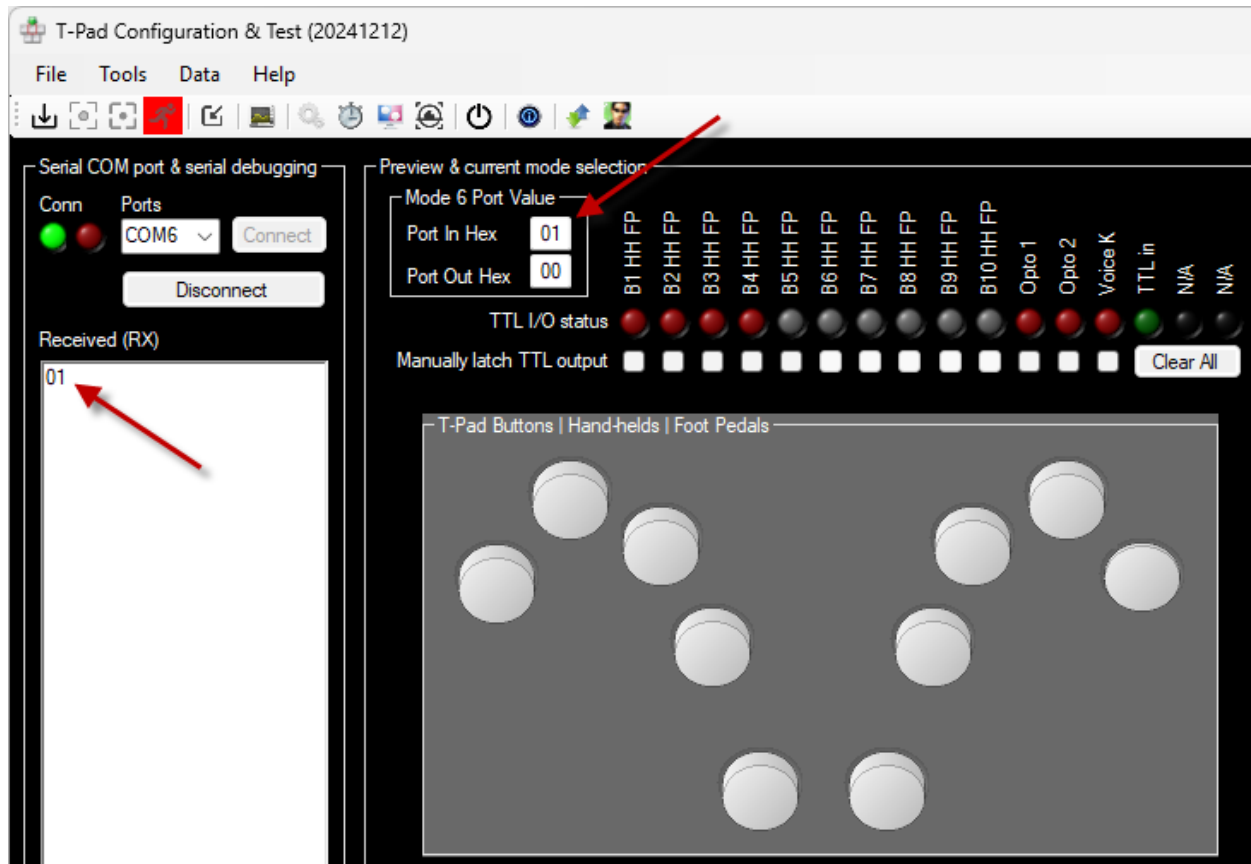
Note that even if the USB Keyboard HID lead is connected no keyboard keystrokes will be sent from the T-Pad.

To check that Mode 6 works as expected click on the Mode 6 checkbox. As you switch to Mode 6 a Mode 6 Port Value box will appear on screen and the virtual and physical LED on the rear of the T-Pad will flash orange.

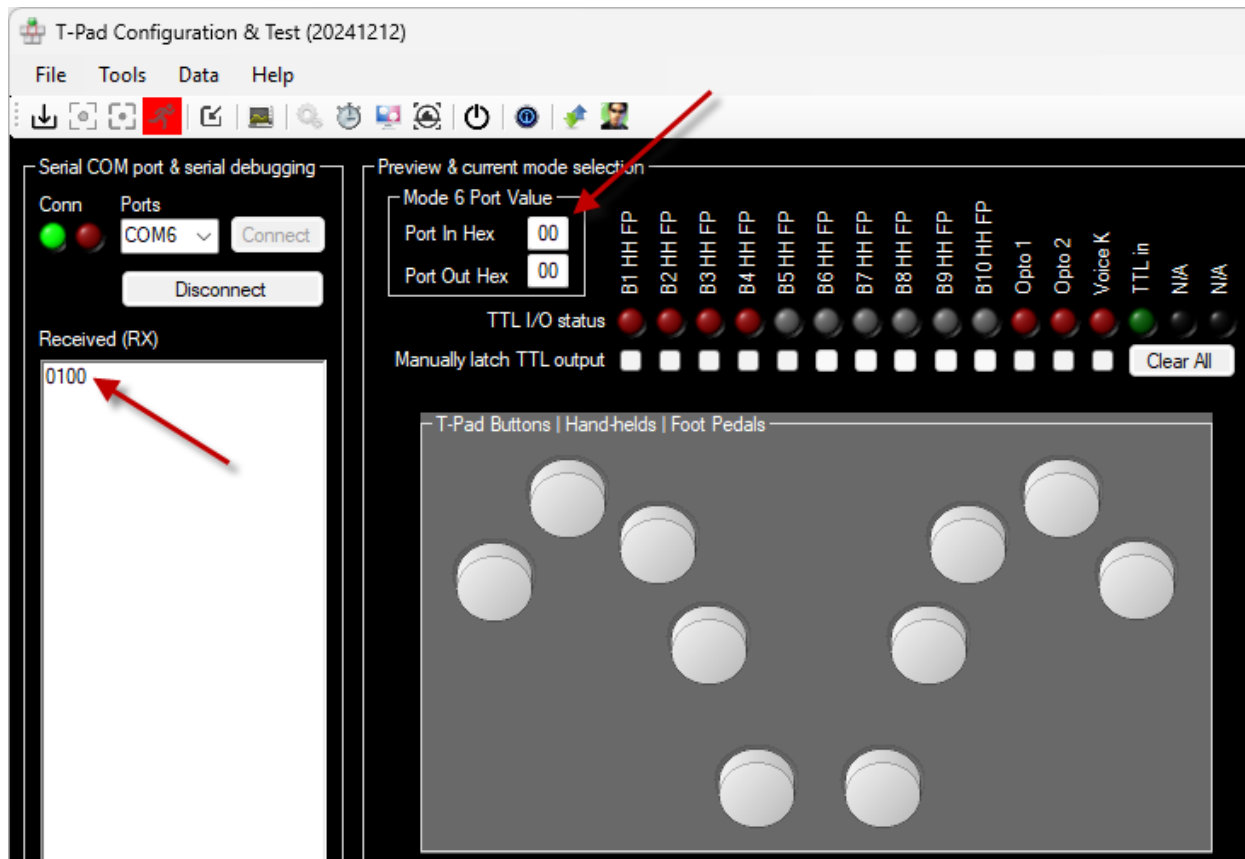


Then repeatedly press and release button 1, Hand-held 1 or Foot Pedal 1.

As you press button 1 you will see the virtual button press down and a 01 will be shown in the Port In Hex value box. In the serial Received (RX) window you will see the two byte port value sent FROM the T-Pad in near real-time.



When you release the button will go up, 00 will be shown in the Port In Hex value box and 00 in serial Received (RX) window.



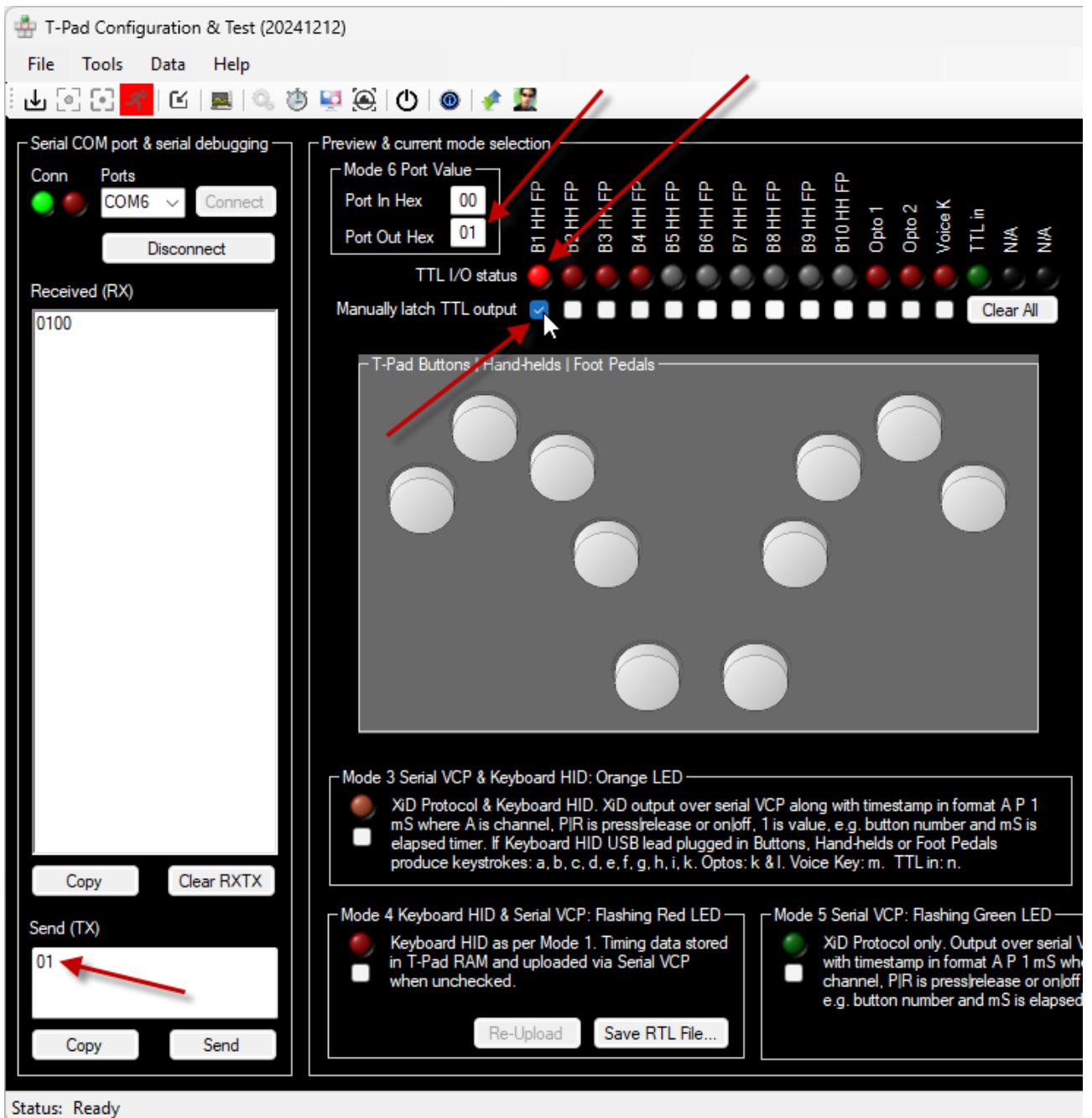
This process will continue as you press and release button 1...

Remember these are the Hex port values been sent FROM the T-Pad as though they had been sent from an actual USB TTL Module.

To test manual TTL output TO the T-Pad in USB TTL Module emulation mode click on the Manually latch TTL output tick box for B1 HH FP.

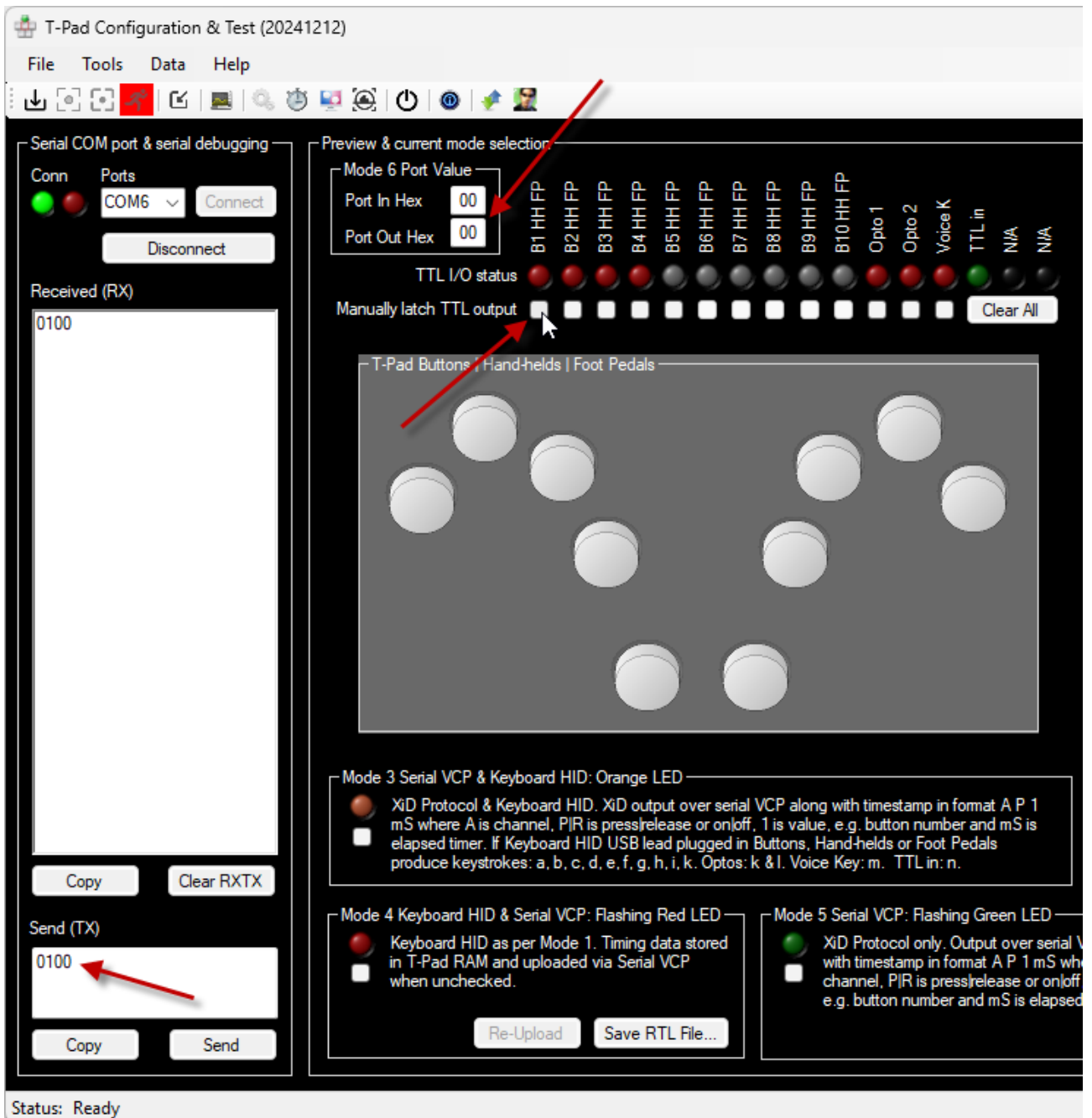
Once ticked 01 will be shown in the Port Out Hex value box, 01 in serial Send (TX) window and the virtual red LED will illuminate.

A physical TTL signal will remain on, high, +5V, for the button 1 TTL output line whilst the box is ticked.



Once unticked 00 will be shown in the Port Out Hex value box, 00 in serial Send (TX) window and the virtual red LED will be unlit.

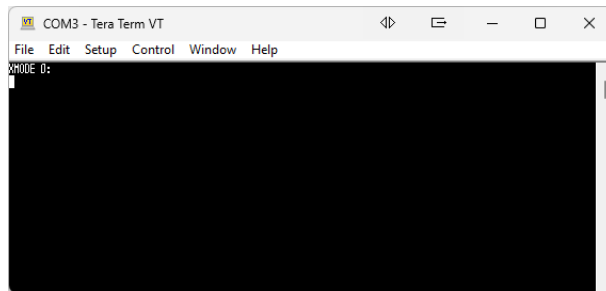
The physical TTL signal will go off, low, 0V, for the button 1 TTL output line once the box is unticked.



11.14.2. Testing in Mode 6 Using a Serial Terminal – Serial VCP: Flashing Orange LED

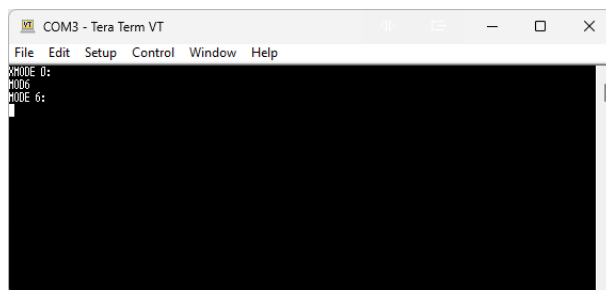
Start Tera Term and connect to the VCP serial port the T-Pad is connected to.

You should ensure that you close the BBTK T-Pad Configuration & Test App as you can ONLY have ONE SERIAL APP RUNNING AT ANY ONE TIME under Windows or you will receive serial port locked errors.



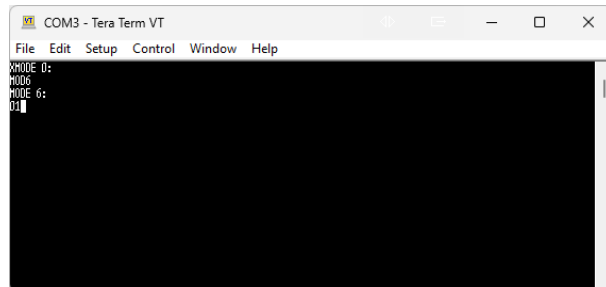
To replicate testing Mode 6 in the T-Pad App start Tera Term and then press capital X on your keyboard.

The T-Pad will respond with MODE 0: This is known as the command mode.



Type MOD6 and press return.

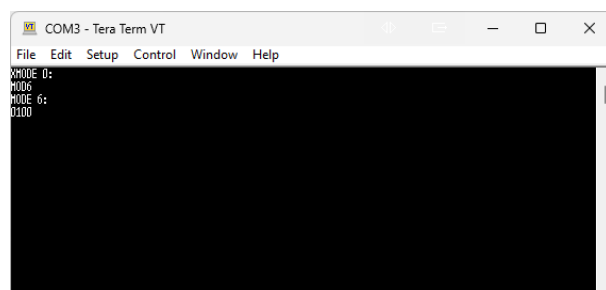
The T-Pad will respond with MODE 6: This sets the T-Pad to Mode 6.



To event mark on line 1, type 01. This will turn on and latch TTL output on button 1.

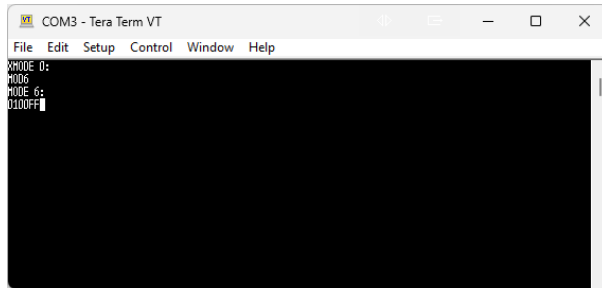
DO NOT press return as in USB TTL Module emulation mode you only need send two Hex bytes in capitals to turn between 1 and 8 lines on.

Hex values can be between 00 through to FF, i.e. all 8 lines off to all 8 lines on.

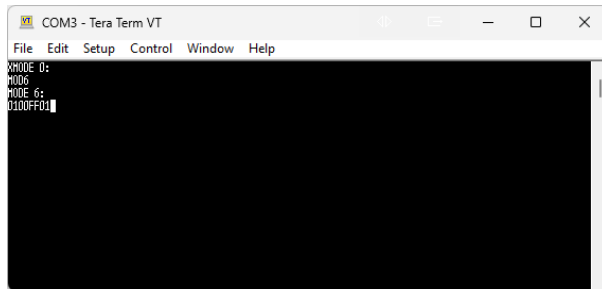


Once a TTL line is turned on it latches on and stays on, active high or +5V. To turn all lines off you need to send 00 as a pair of Hex bytes.

Remember you should not press return or send any CR or LF's when in USB TTL Module emulation mode.



In this example all lines have been turned on by sending FF to the T-Pad. This would turn on buttons 1 through 4 TTL outs along with TTL outs for Opto 1, Opto 2, Voice Key and TTL In.



If you wished to keep some lines on, in this example the TTL output line for button 1 you would need to use masking.

So 01 would turn all other TTL output lines off and leave line 1, i.e. button 1 still on, active high or +5V. Buttons 2-3 TTL outputs would turn off as would TTL outputs for Opto 1, Opto 2, Voice Key and TTL In.

11.14.3. Decoding Pairs of Hex Bytes Sent to and From the T-Pad in USB TTL Module emulation mode

Decoding the pairs of Hex bytes sent to and from the T-Pad in USB TTL Module emulation mode is straightforward. First you need to check which lines you are working with and check to see if they are on or off, button pressed or not pressed, active or not active, i.e. 1 for on in binary and 0 for off.

When individual buttons 1 to 4 are pressed in USB TTL Module emulation mode two Hex bytes are sent TO your PC from the T-Pad over serial, e.g. 01 for button 1, 02 for button 2, 04 for button 3 and 08 for button 4. In binary this would be 1, 2, 4, and 8 respectively.

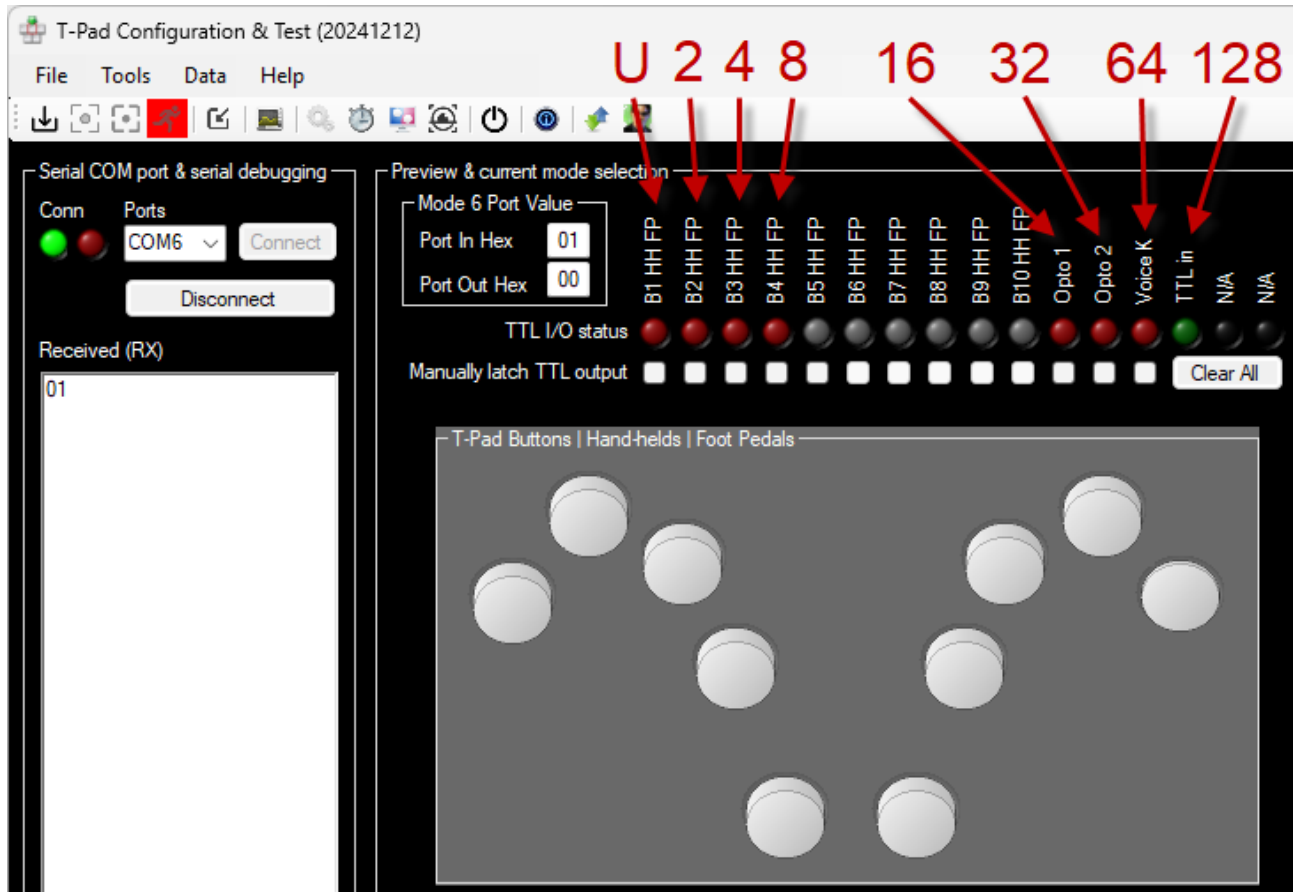
Opto 1 would send bytes 10, Opto 2 byte pair 20, Voice Key byte pair 40 and TTL in 80. In all these examples each two byte code assumes that no other buttons are pressed or that there is any other activity. In binary this would be 16, 32, 64 and 128 respectively.

A screen grab below illustrates the binary offsets for each button, Opto, Voice Key and TTL in.

Note the Least Significant Bit (LSB) is shown on the left in the screen grab, i.e. U for Units or 0|1.

Sending RR will reset it and clear the buffer. No CRLF should be sent.

So for example when button 1 is pressed, the Hex byte pair 01 is sent FROM the T-Pad. So if you were monitoring for button presses you would need to look for 01 to signify that button 1 had been pressed and held down. For button 1 to be flagged as up 00 would need to be received FROM the T-Pad. If other buttons or sensors were active you would need to mask off button 1 to detect it going down and up amongst other button and sensor activity.



Note there are no CR or LF pairs when two bytes are sent FROM the T-Pad in USB TTL Module emulation mode.

A full binary to hex decoding chart is shown below for reference.

Note the Least Significant Bit (LSB) is shown on the right in this table, i.e. U for Units or 0|1.

BBTK USB TTL Module Control Values

To reset USB TTL Module send RR - NEVER SEND CR or LF -- ONLY 2 byte upper case letters or numbers

Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)
0	00000000	00	29	00011101	1D	58	00111010	3A	87	01010111	57	116	01110100	74
1	00000001	01	30	00011110	1E	59	00111011	3B	88	01011000	58	117	01110101	75
2	00000010	02	31	00011111	1F	60	00111100	3C	89	01011001	59	118	01110110	76
3	00000011	03	32	00100000	20	61	00111101	3D	90	01011010	5A	119	01110111	77
4	00000100	04	33	00100001	21	62	00111110	3E	91	01011011	5B	120	01111000	78
5	00000101	05	34	00100010	22	63	00111111	3F	92	01011100	5C	121	01111001	79
6	00000110	06	35	00100011	23	64	01000000	40	93	01011101	5D	122	01111010	7A
7	00000111	07	36	00100100	24	65	01000001	41	94	01011110	5E	123	01111011	7B
8	00001000	08	37	00100101	25	66	01000010	42	95	01011111	5F	124	01111100	7C
9	00001001	09	38	00100110	26	67	01000011	43	96	01100000	60	125	01111101	7D
10	00001010	0A	39	00100111	27	68	01000100	44	97	01100001	61	126	01111110	7E
11	00001011	0B	40	00101000	28	69	01000101	45	98	01100010	62	127	01111111	7F
12	00001100	0C	41	00101001	29	70	01000110	46	99	01100011	63	128	10000000	80
13	00001101	0D	42	00101010	2A	71	01000111	47	100	01100100	64	129	10000001	81
14	00001110	0E	43	00101011	2B	72	01001000	48	101	01100101	65	130	10000010	82
15	00001111	0F	44	00101100	2C	73	01001001	49	102	01100110	66	131	10000011	83
16	00010000	10	45	00101101	2D	74	01001010	4A	103	01100111	67	132	10000100	84
17	00010001	11	46	00101110	2E	75	01001011	4B	104	01101000	68	133	10000101	85
18	00010010	12	47	00101111	2F	76	01001100	4C	105	01101001	69	134	10000110	86
19	00010011	13	48	00110000	30	77	01001101	4D	106	01101010	6A	135	10000111	87
20	00010100	14	49	00110001	31	78	01001110	4E	107	01101011	6B	136	10001000	88
21	00010101	15	50	00110010	32	79	01001111	4F	108	01101100	6C	137	10001001	89
22	00010110	16	51	00110011	33	80	01010000	50	109	01101101	6D	138	10001010	8A
23	00010111	17	52	00110100	34	81	01010001	51	110	01101110	6E	139	10001011	8B
24	00011000	18	53	00110101	35	82	01010010	52	111	01101111	6F	139	10001011	8B
25	00011001	19	54	00110110	36	83	01010011	53	112	01110000	70	140	10001100	8C
26	00011010	1A	55	00110111	37	84	01010100	54	113	01110001	71	141	10001101	8D
27	00011011	1B	56	00111000	38	85	01010101	55	114	01110010	72	142	10001110	8E
28	00011100	1C	57	00111001	39	86	01010110	56	115	01110011	73	143	10001111	8F

Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)
144	10010000	90	172	10101100	AC	201	11001001	C9	230	11100110	E6
145	10010001	91	173	10101101	AD	202	11001010	CA	231	11100111	E7
146	10010010	92	174	10101110	AE	203	11001011	CB	232	11101000	E8
147	10010011	93	175	10101111	AF	204	11001100	CC	233	11101001	E9
148	10010100	94	176	10110000	B0	205	11001101	CD	234	11101010	EA
149	10010101	95	177	10110001	B1	206	11001110	CE	235	11101011	EB
150	10010110	96	178	10110010	B2	207	11001111	CF	236	11101100	EC
151	10010111	97	179	10110011	B3	208	11010000	D0	237	11101101	ED
152	10011000	98	180	10110100	B4	209	11010001	D1	238	11101110	EE
153	10011001	99	181	10110101	B5	210	11010010	D2	239	11101111	EF
154	10011010	9A	182	10110110	B6	211	11010011	D3	240	11110000	F0
155	10011011	9B	183	10110111	B7	212	11010100	D4	241	11110001	F1
156	10011100	9C	184	10111000	B8	213	11010101	D5	242	11110010	F2
157	10011101	9D	185	10111001	B9	214	11010110	D6	243	11110011	F3
158	10011110	9E	186	10111010	BA	215	11010111	D7	244	11110100	F4
159	10011111	9F	187	10111011	BB	216	11011000	D8	245	11110101	F5
160	10100000	A0	188	10111100	BC	217	11011001	D9	246	11110110	F6
161	10100001	A1	189	10111101	BD	218	11011010	DA	247	11110111	F7
162	10100010	A2	190	10111110	BE	219	11011011	DB	248	11111000	F8
163	10100011	A3	191	10111111	BF	220	11011100	DC	249	11111001	F9
164	10100100	A4	192	11000000	C0	221	11011101	DD	250	11111010	FA
165	10100101	A5	193	11000001	C1	222	11011110	DE	251	11111011	FB
166	10100110	A6	194	11000010	C2	223	11011111	DF	252	11111100	FC
167	10100111	A7	195	11000011	C3	224	11100000	E0	253	11111101	FD
168	10101000	A8	196	11000100	C4	225	11100001	E1	254	11111110	FE
169	10101001	A9	197	11000101	C5	226	11100010	E2	255	11111111	FF
170	10101010	AA	198	11000110	C6	227	11100011	E3			
171	10101011	AB	199	11000111	C7	228	11100100	E4			
			200	11001000	C8	229	11100101	E5			

12. Using the T-Pad With Your Own Software or Experiment Generator

You have multiple options when using the T-Pad with software that you have written yourself or when using an off-the-shelf experiment generator. However the most common options are Keyboard HID or XiD Protocol over serial.

The simplest option is to use the T-Pad as a standard USB keyboard in Mode 1. In this mode the T-Pad acts as a dumb, but very fast and consistent keyboard funning at 1,000Hz, potentially reporting activity each millisecond. When you press a button it is equivalent to pressing a key on a standard keyboard. In this mode the status of each button, Opto, Voice Key or TTL In is sent as a keyboard keypress.

Note that in Keyboard HID mode your own software or Experiment Generator may not receive and interpret key presses as quickly as the T-Pad hardware can send reports over USB. This is because your own software is responsible for receiving and interpreting key presses and it may not do this in a timely fashion, i.e. timestamp accurately from the PC's internal clock. This can be due to your own code, how the software works, Operating System delays or other intervening processes.

The second more complex option is to use the T-Pad in XiD mode. In this mode the status of each button, Opto, Voice Key or TTL In is sent over serial along with a timestamp from the independent hardware clock built in to the T-Pad.

12.1. Option 1: Mode 1 Keyboard HID

The first option is to use the T-Pad in keyboard HID mode alone, i.e. mode 1 and only connect the keyboard HID USB lead. In this mode simply use the T-Pad as you would any standard USB keyboard.

The T-Pad keyboard interface runs in full-speed USB and sends key presses typically in under 1 millisecond as internally it runs at 1,000 Hz. Remember that although the T-Pad hardware can send key presses in under 1 millisecond there is no guarantee that your computer or the software you are using can processes the USB packets in a timely manner. In this mode the T-Pads inbuilt hardware timer is not used and all timings are taken from the PC clock using your own software. This means that timing may be inaccurate.

In Mode 1, each button, Opto, Voice Key or TTL In when pressed, or active, produces the following HID keyboard key presses.

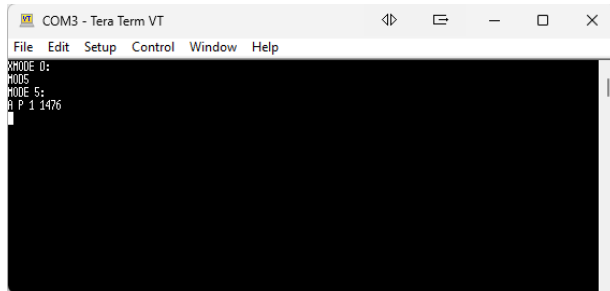
Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
HID Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n

Whilst pressed down, or active, the key press will be held down as though you had pressed and held down a key on a standard USB keyboard. When anything that was active becomes inactive a key up will be sent as though that key had been released on a keyboard.

12.2. Option 2: Mode 5 XiD Protocol Over Serial

The second option is to use the XiD compatible protocol over serial. This is a format that reports whether a button is pressed or released along with an independent timestamp from the T-Pads inbuilt millisecond accurate hardware timer.

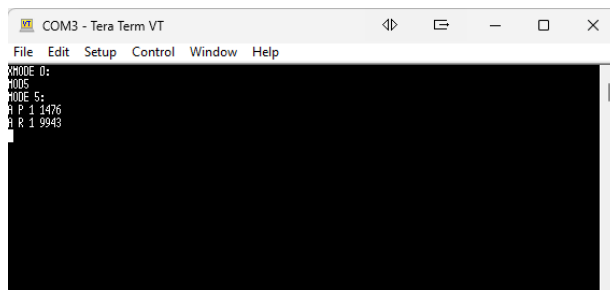
Each button, Opto, Voice Key or TTL In when pressed or active produces an XiD command sent over serial. For example when using a serial terminal like Tera Term.



Press button 1 and an XiD command will be sent over serial from the T-Pad.

In this example:

A P 1 1476



When button 1 is released an XiD command will be sent over serial from the T-Pad.

In this example:

A R 1 9943

If we look at the first press and release of button 1 we can decode the XiD as follows:

A P 1 1476

translates to:

Channel [A] buttons | [P]ress | [1] button 1 | [1476] elapsed millisecond timer

A R 1 9943

translates to:

Channel [A] buttons | [R]elease | [1] button 1 | [9943] elapsed millisecond timer

In this case we can see that button 1 has been held down for 8467 mS, i.e. 9943 - 1476.

Note that the timestamp will be millisecond accurate as that is the timestamp recorded by the T-Pads hardware timer, but as the T-Pad is sending a number of characters back to your computer this takes some time to send over serial.

Typically the T-Pad can send around 115,200 characters/bytes per second at 115,200 baud. Theoretically that translates to one character or one byte around every 0.008 mS or 115 characters/bytes per millisecond.

So the first button press XiD command:

A P 1 3128

Is 12 characters or bytes if you include spaces and the invisible terminating characters, CRLF.

So that XiD command would take 12×0.009 mS or 0.108 mS to transmit to your PC.

However you should bear in mind that your PC and the software you use may not be able to receive and interpret or act on serial at this speed. Typically there will be some small overhead over and above this theoretical maximum transmission speed per XiD command received from the T-Pad.

It may also be that your software or Experiment Generator may block, or be blocking, whilst reading serial commands from the T-Pad. This is an advanced programming topic and can be researched by Googling, “is serial blocking” for example.

In short this means that your own code, or Experiment Generator, may stop execution of subsequent code whilst it waits for the terminating CRLF on each XiD command sent from the T-Pad. This is so that it knows that an XiD command has been received.

If multiple XiD commands are sent one after another in quick succession they may stack and depending on how your own code or Experiment Generator processes the serial stream that may take more time and prevent subsequent code executing until the stack is processed. This would be considered to be “blocking”. “Non-blocking” serial code would not stop subsequent code executing as incoming serial is processed. Again this is an advanced programming topic that also encompasses multi-tasking and threading and can be researched by Googling, “non-blocking serial”.

The T-Pad itself will keep collecting and timestamping event data in real-time ready to send over serial once the last XiD command has been sent and received by your computer. This is because the T-Pad is interrupt driven, has an independent millisecond accurate timer and makes use of advanced internal circular buffers and ultrafast RAM. In other words it is multi-threaded. It will not stop sending XiD commands to your computer, or will throttle its output, as it has no flow control due to its intention to process events as quickly as possible.

13. Pseudocode Example of how to use the T-Pad with XiD in Mode 5

To illustrate the principles that should be followed when using the T-Pad with your own code or Experiment Generator it is useful to outline how it can be used independent of programming or scripting language.

Although the following serial example uses Pseudocode the same rationale and algorithms apply to most programming languages. A simple visual stimulus-response study might look something like:

- Start experiment
- Open T-Pad serial port
- Send X over serial to T-Pad to put into command mode
- Send MOD5 to T-Pad to put into Mode 5, i.e. XiD protocol only
- Show experiment instructions on screen
- Send R to the T-Pad to reset the T-Pad hardware timer to 0 mS
- Start continuously reading from serial port in a tight loop
 - Present a visual stimuli with a white Opto Marker on it, i.e. the first trial.
 - Read XiD command that shows the visual stimuli onset as the white Opto Marker is first detected, e.g. C R 1 2500.
 - This decodes as: C – Channel C Optos, P – Press or Opto activated, 1 – Opto 1, 2500 – Elapsed time from millisecond accurate hardware timer. This is the onset of the visual stimulus appearing on screen and being visible in the real world. The elapsed running timer of 2500 mS should be taken as 0 mS in effect, i.e. the start of the trial.
 - Keep waiting for a button down that will produce a new XiD command being sent from the T-Pad over serial, e.g. in a While loop.
 - When a button is pressed another XiD command will be sent from the T-pad, e.g. A P 1 4000.
 - This decodes as: A – Channel A buttons, P – Button press, 1 – Button 1, 4000 – Elapsed time from millisecond accurate hardware timer. From the elapsed we can work out what the RT or Response Time was, i.e. $4000 - 2500 = 1500$ mS.
 - We might then choose to remove the stimulus image and display a black screen or black Opto Marker. The T-Pad would then send an XiD command that signifies that change, e.g. C R 1 5000.
 - This decodes as: C – Channel C Optos, R – Release or Opto not activated, 1 – Opto 1, 5000 – Elapsed time from millisecond accurate hardware timer. From the elapsed we can work out what the stimulus image duration was, i.e. $5000 - 2500 = 2500$ mS.
 - It might be the case that button 1 on the T-Pad would then be released generating a button up XiD command, e.g. A P 1 5500.
 - This decodes as: A – Channel A buttons, R – Button release, 1 – Button 1, 5500 – Elapsed time from millisecond accurate hardware timer. From the elapsed time we can work out how long the button was held down for, i.e. $5500 - 4000 = 1500$ mS.
 - Store all data for trial one
 - Present subsequent trials in a tight loop that constantly monitors for XiD commands being sent from the T-Pad over serial

As this example shows we now have the real-world, hardware independent, times for stimulus onset, Response Time, stimulus duration and button down duration.

14. PsychoPy Example Using the XiD Protocol

The previous Pseudocode example of a simple visual-stimulus study when implemented in PsychoPy might look something similar to the example shown below. It is intended to give you a flavor of how the T-Pad could work for you and how easy it is to use.

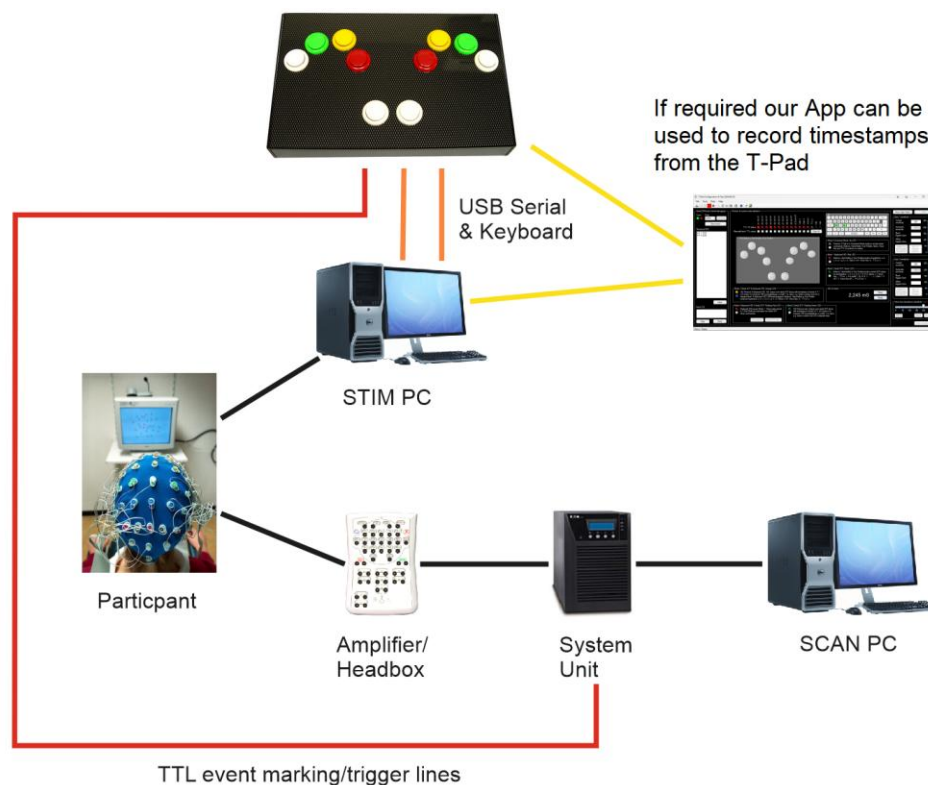
We provide example scripts and code snippets for many of the popular experiment generators, e.g. PsychoPy in this example. Even if you are not familiar with PsychoPy and the Python programming language the rationale is very similar for any Experiment Generator or software which you have written yourself.

These examples have been developed using: PsychoPy 2024.2.4. (<https://www.psychopy.org/>)

THESE EXAMPLES ARE PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

The example schematic below shows a simple EEG study and how the T-Pad might be connected to the STIM PC you present stimuli to your participant on. Two USB leads connect to the STIM PC, one for a keyboard HID (orange line) and another for streaming serial timestamped data (second orange line).

An Event Marking line, or cable, is connected to the EEG System Unit (red) so that a +5V TTL signal will appear on the STIM TTL Trigger Input port and ultimately on the EEG Event Marking channels when a visual stimulus is presented or a response made by pressing a button. The TTL line tied to a given button will be active high, +5V, whilst the button is being held down. It will go off, or 0V, when the button is released.



In this example the participant has to press button one when a green smiley face stimulus is shown on the STIM PC's screen and button two when a red sad face is shown. When a button is pressed the exact moment the button is pressed is timestamped and sent to the STIM PC over serial in XiD format.

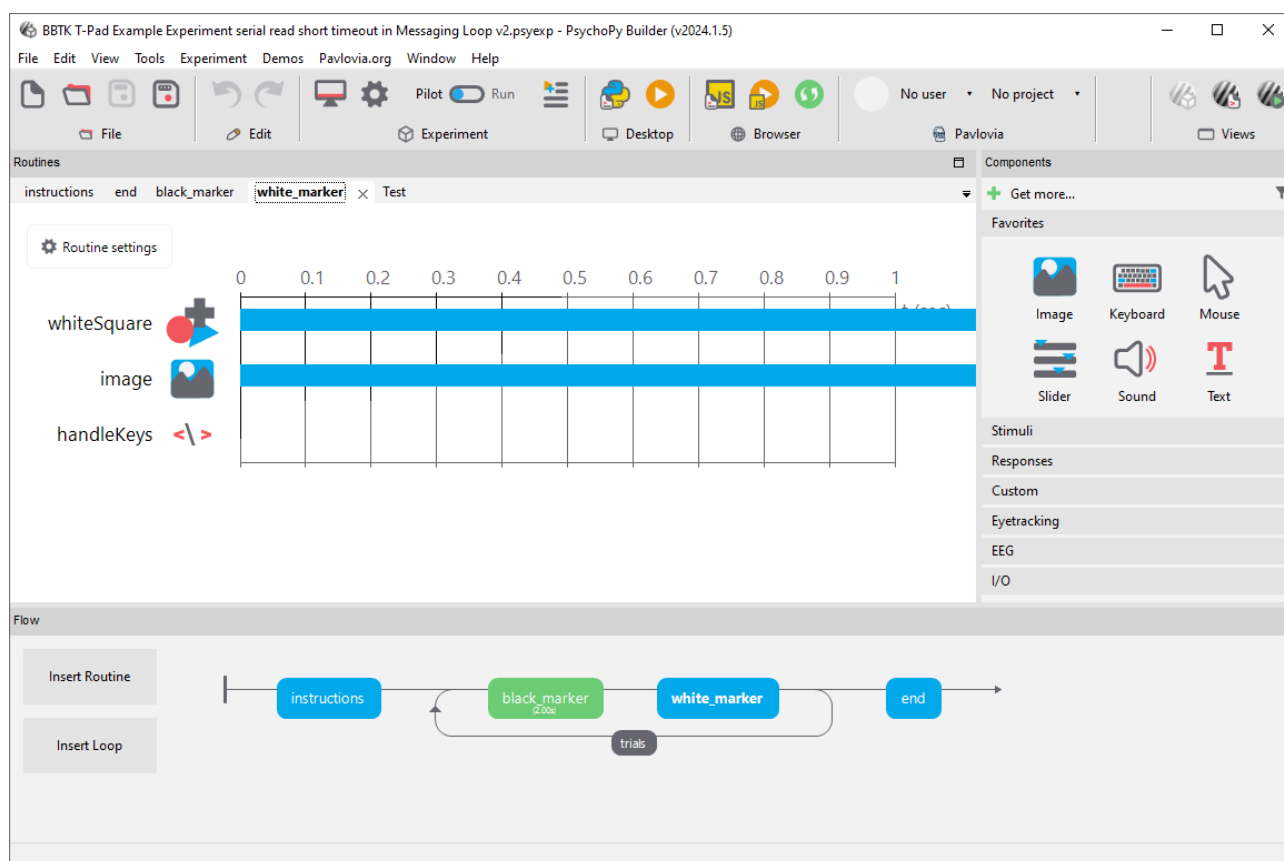


Note on the stimulus images shown above there is a white Opto marker to the right which can be detected by the T-Pad's light sensor, Opto-detector or photodiode. This means that the T-Pad can timestamp the exact onset of the stimulus image accurately using its internal hardware based millisecond accurate timer.

Once the exact onset of the stimulus image has been detected and timestamped in the real world you can be sure that your Response Times (RTs) are also millisecond accurate as the button down will also be independently timestamped by the T-Pad.

In addition to every stimulus and response event being timestamped and sent to the PC over serial, a TTL event mark, or trigger, is also sent to any other equipment that needs to receive accurate onsets, duration and offsets, e.g. in this example the SCAN PC.

In the example below you can see how each stimulus image is presented and that there is some script to handle the timestamped data being sent from the T-Pad as it detects visual stimuli being presented and response buttons pressed, i.e. handleKeys in this case.



Images are presented in sequential order from the conditions.xlsx file shown below, i.e. the happy and sad images of Emoji's.

conditions.xlsx - Excel

	A	B	C	D	E	F
1	imageFile	corrAns	stimType			
2	images/green.jpg	a	happy			
3	images/red.jpg	b	sad			
4	images/green.jpg	a	happy			
5	images/red.jpg	b	sad			
6	images/green.jpg	a	happy			
7	images/red.jpg	b	sad			
8	images/green.jpg	a	happy			

The actual visual onsets and button presses are handled in the `handleKeys` code component. In effect this is a generic piece of code that applies to any experiment which uses the T-Pad.

This code simply waits for serial data being sent from the T-Pad to the STIM PC in a loop whilst carrying out other tasks, e.g. displaying an image or playing a sound. When a "\n" byte has been detected in the serial stream being sent, i.e. a complete XiD command has been received, this is then parsed to determine the Channel, Press or Release, Sensor number and running Millisecond timer.

An overview of the Python code in `handleKeys` is shown below.

handleKeys Properties

Name: Code type: ☐ disabled

Before Experiment * Begin Experiment * Begin Routine * Each Frame * End Routine End Experiment *

```

60 if "\n" in strDecodedSerialData: #and booSettingWhiteOptoSensitivity == 1: #check if there has b
61     datRTC = datetime.now().strftime('%Y/%m/%d-%H:%M:%S.%f')[:-3] #get the PC date and time we h
62     booProcessSerialReceived = 1 #flag that serial has been received and we need to process XiD
63
64     strXiDChannel = "" + strDecodedSerialData[0] #channel
65     strXiDPRESSRelease = "" + strDecodedSerialData[2] #press/release
66     strXiDSensorNo = "" + strDecodedSerialData[4] #sensor
67     strXiDmSTimer = "" + strDecodedSerialData[6:] #running mS timer
68
69 if strXiDChannel == 'A': #channel A is for buttons 1 to 10
70     if strXiDPRESSRelease == 'P': #P is for button press/button down
71         if strXiDSensorNo == '1': #check the button number, i.e. button 1 in this case
72             booXiDButton1Down = 1 #flag button 1 as being pressed/down
73         if strXiDSensorNo == '2':
74             booXiDButton2Down = 1
75         if strXiDSensorNo == '3':
76             booXiDButton3Down = 1
77         if strXiDSensorNo == '4':
78             booXiDButton4Down = 1
79         if strXiDSensorNo == '5':
80             booXiDButton5Down = 1
81         if strXiDSensorNo == '6':
82             booXiDButton6Down = 1
83         if strXiDSensorNo == '7':
84             booXiDButton7Down = 1
85         if strXiDSensorNo == '8':
86             booXiDButton8Down = 1
87         if strXiDSensorNo == '9':
88             booXiDButton9Down = 1
89         if strXiDSensorNo == '0':
90             booXiDButton10Down = 1
91     if strXiDPRESSRelease == 'R': #R is for button release/button up
92         if strXiDSensorNo == '1': #check the button number, i.e. button 1 in this case
93             booXiDButton1Down = 0 #flag button 1 as being released/up
94         if strXiDSensorNo == '2':
95             booXiDButton2Down = 0
96         if strXiDSensorNo == '3':
97             booXiDButton3Down = 0
98         if strXiDSensorNo == '4':
99             booXiDButton4Down = 0
100         if strXiDSensorNo == '5':
101             booXiDButton5Down = 0
102         if strXiDSensorNo == '6':
103             booXiDButton6Down = 0
104         if strXiDSensorNo == '7':
105             booXiDButton7Down = 0
106         if strXiDSensorNo == '8':
107             booXiDButton8Down = 0
108         if strXiDSensorNo == '9':
109             booXiDButton9Down = 0
110         if strXiDSensorNo == '0':
111             booXiDButton10Down = 0
112
113 if strXiDChannel == 'C': #channel C is for optos 1 & 2
114     if strXiDPRESSRelease == 'P': #P is for opto active/on
115         if strXiDSensorNo == '1': #check the opto number, i.e. opto 1 in this case
116             booXiDOpto1Active = 1
117         if strXiDSensorNo == '2':
118             booXiDOpto2Active = 1
119     if strXiDPRESSRelease == 'R': #R is for opto non active/off
120         if strXiDSensorNo == '1':
121             booXiDOpto1Active = 0
122         if strXiDSensorNo == '2':
123             booXiDOpto2Active = 0
124
125 if strXiDChannel == 'M': #channel M is for voice key or mic
126     if strXiDPRESSRelease == 'P': #P is for microphone active/on
127         if strXiDSensorNo == '1': #microphone 1
128             booXiDVKActive = 1
129     if strXiDPRESSRelease == 'R': #R is for microphone non active/off
130         if strXiDSensorNo == '1':
131             booXiDVKActive = 0
132
133 if strXiDChannel == 'B': #channel B is for TTL in
134     if strXiDPRESSRelease == 'P': #P is for TTL in active 5V/on
135         if strXiDSensorNo == '1':
136             booXiDTTLActive = 1
137     if strXiDPRESSRelease == 'R': #R is for TTL in non active 0V/off
138         if strXiDSensorNo == '1':
139             booXiDTTLActive = 0

```

Help OK Cancel

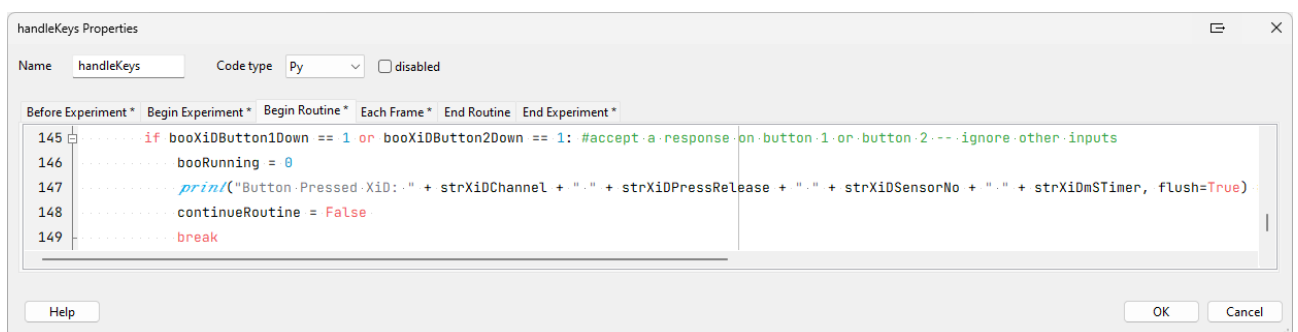
A complete XiD command sent from the T-Pad is shown below along with how it would be decoded by the code snippet above.

C P 1 6190			
C	P	1	6190
Channel	Press Release	Sensor Number	Millisecond Timer
Optos	On	Opto 1	Timestamp

Because each XiD command is sent over serial and is individually timestamped you don't have to worry about timing accuracy being an issue.

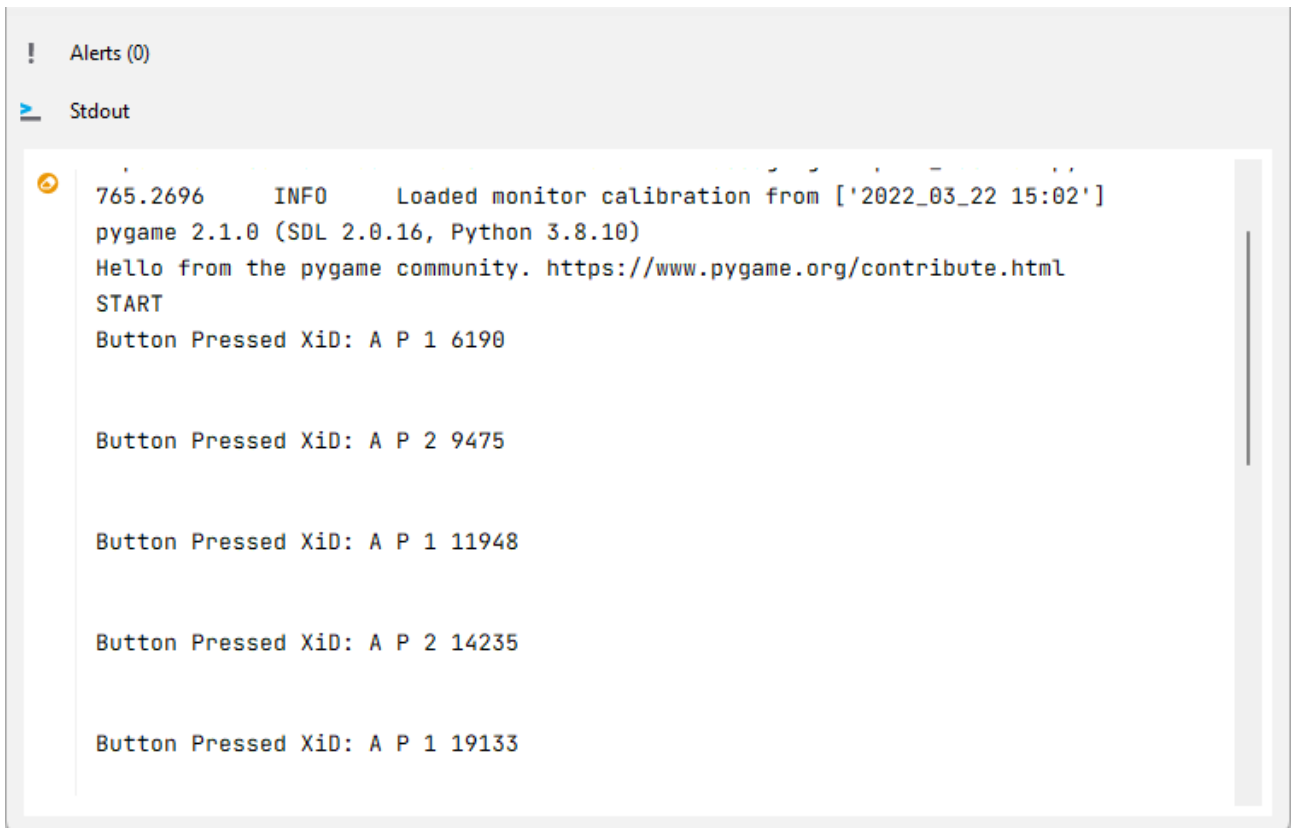
Once you have decoded XiD serial command sent from the T-pad in your experiment you are free to do whatever you want with that information. Usually you would log the activity and timestamps to your PsychoPy data file, but you can use responses for conditional branching, triggering new stimuli or TTL event marking and triggering.

For example it is easy to record the onset of either button 1 or button 2 using some simple code once the XiD command has been parsed out using the code example above.



In this case the timestamp of the onset of either button 1 or button 2 will be printed to the Stdout window for debugging purposes. All other button presses and button releases are ignored.

Here we can see the XiD information for the first button pressed in response to each trial.



```
! Alerts (0)
> Stdout

765.2696 INFO Loaded monitor calibration from ['2022_03_22 15:02']
pygame 2.1.0 (SDL 2.0.16, Python 3.8.10)
Hello from the pygame community. https://www.pygame.org/contribute.html
START
Button Pressed XiD: A P 1 6190

Button Pressed XiD: A P 2 9475

Button Pressed XiD: A P 1 11948

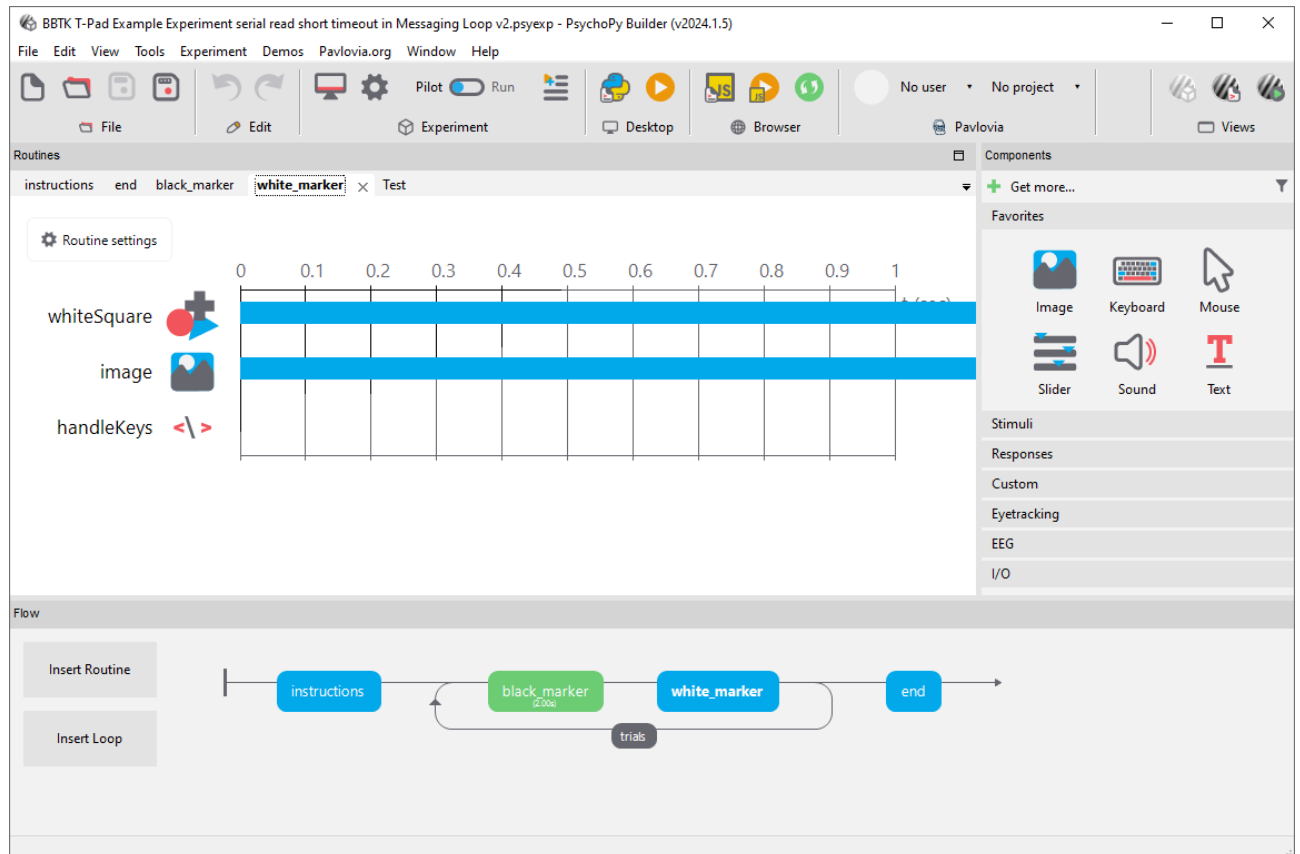
Button Pressed XiD: A P 2 14235

Button Pressed XiD: A P 1 19133
```

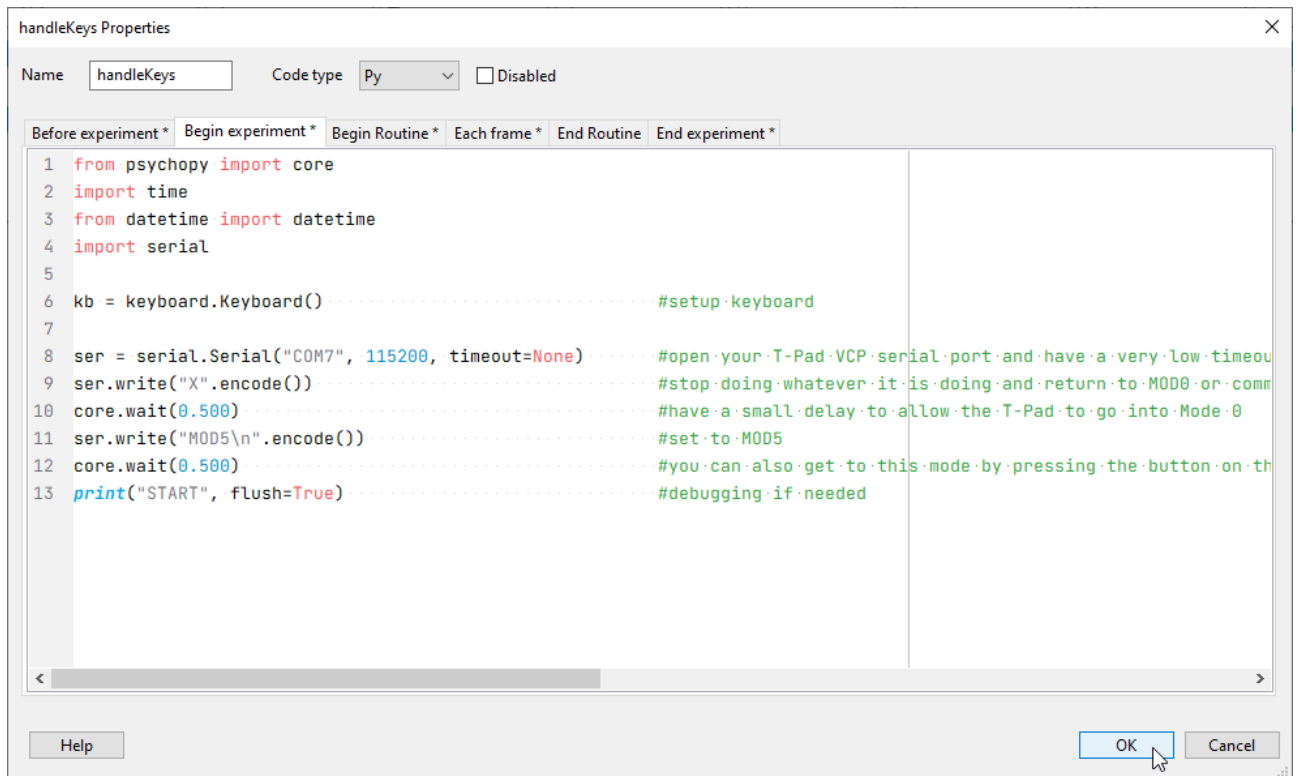
If you are running your experiment on a MS Windows platform however, remember that our T-Pad partner App has the ability to record all XiD timestamped data so that you only need to concentrate on USB keyboard key presses to make your experiment even simpler.

Although this example is specific to PsychoPy, the Python code is fairly generic and the pseudocode outlined in the previous section is applicable to most programming languages and can be easily adapted.

In PsychoPy the main PsychoPy code that interacts with the T-Pad is contained within the `white_marker` routine under the `handleKeys <\>` code component.



To view the Python code under the handleKeys <\> code component click on it so that the Properties window is shown.



The key tabs/Python code is contained under the Begin experiment, Begin routine and End experiment tabs.

The Begin experiment tab imports the required libraries, initializes the HID keyboard and opens the T-Pads virtual COM port. Next an X is sent to return the T-Pad to Mode 0 or command mode. Note that the HID keyboard is only used to detect the [q] key being pressed to quit the experiment early.

Once initialized the T-Pad is then switched to Mode 5, by sending MOD5 to the T-Pad.

BEGIN EXPERIMENT

```
-----
from psychopy import core
import time
from datetime import datetime
import serial

kb = keyboard.Keyboard()                                #setup keyboard

ser = serial.Serial("COM7", 115200, timeout=None)        #open your T-Pad VCP serial port and have a very
low timeout value to stop blocking, i.e. zero
ser.write("X".encode())                                  #stop doing whatever it is doing and return to
                                                         #MOD0 or command mode
core.wait(0.500)                                         #have a small delay to allow the T-Pad to go into
                                                         #Mode 0
ser.write("MOD5\n".encode())                             #set to MOD5, i.e. XiD only mode
core.wait(0.500)                                         #you can also get to this mode by pressing the
                                                         #button on the rear of the T-Pad 4 times until
                                                         #it flashes red
print("START", flush=True)                               #debugging if needed
-----
```

Once the serial port is open the Python code that keeps checking for T-Pad button presses, Optos, Voice Key and TTL in is under the Begin Routine tab. This is in effect a tight loop that keeps monitoring the T-Pad serial port and checks for XiD commands being sent from the T-Pad.

Once a complete XiD command is received from the T-Pad it is decoded so that the code determines if an Opto marker, a button press, a Voice Key response or TTL in has been detected. The elapsed time is also parsed. Remember that the elapsed millisecond time is the millisecond accurate hardware timer in the T-Pad and not the PC's clock available to PsychoPy through software.

Note that the code within the Begin Routine tab also puts a white Opto marker on to the stimulus image screen by using the `whiteSquare.draw()` so that the T-Pad can detect the image onset in the real world with millisecond accuracy.

If you wish to store any Response Times, event marks, onsets etc. you should do that within this routine using the standard method for doing so in PsychoPy.

As discussed in the previous section PsychoPy itself handles the actual flow of the experiment and determines what stimuli to display along with recording basic data.

```
BEGIN ROUTINE
-----
strSerialData = "" #used to store raw serial data received
strDecodedSerialData = "" #decoded serial data in utf-8
intCounter = 0 #counter that increments by +1 each time there is a change on anything on the T-pad,
that could be a button going down or up. multiple changes count as 1 sample or a report
KeysPressed = "" #array string to store which keys are pressed
booProcessSerialReceived = 0 #boolean flag as to whether there has been a change that needed to be
processed, i.e. displayed on screen
strSerialData = ""
strDecodedSerialData= ""
strDecodedSerialDataBuffer = ""
booXiDButton1Down= 0 #the boolean status of each input, i.e. 0 for button up or off and 1 for button
down or on
booXiDButton2Down= 0
booXiDButton3Down= 0
booXiDButton4Down= 0
booXiDButton5Down= 0
booXiDButton6Down= 0
booXiDButton7Down= 0
booXiDButton8Down= 0
booXiDButton9Down= 0
booXiDButton10Down= 0
booXiDOpto1Active = 0
booXiDOpto2Active = 0
booXiDVKActive = 0
booXiDTTLActive = 0
strXiDChannel = "" #variable to store channel from XiD command sent over serial
strXiDPressRelease = "" #variable to store Press or Release from XiD command sent over serial
strXiDSensorNo = "" #variable to store button or sensor number
strXiDmSTimer = "" #variable to store the running external hardware mS timer built into the T-Pad
strXiDStatusMsg = "" #variable to store the status of each button or sensor built up from receiving
XiD commands over serial
outputlength = 0 #variable to store the length of the last serial bytes sent from the T-Pad
booRunning = 1

#flush serial buffer and keyboard
ser.flush()
ser.reset_input_buffer()
kb.clearEvents()

whiteSquare.draw()
image.draw()
win.flip() #show the status stimulus image on screen

while booRunning == 1:
    KeysPressed = kb.getKeys(['q'], waitRelease=False, clear=True) #check which keys have been pressed
```

```

    if KeysPressed: #if there has been a key release then check which key it is?
        for k in KeysPressed:
            if k.name == 'q':
                ser.write("X".encode()) #stop doing whatever and return the T-pad to MOD0, i.e. mode
0 or command mode
                core.wait(0.500) #add a small delay for the T-Pad to come out of whatever mode it is
in
                #flush serial buffer and keyboard
                ser.flush()
                ser.reset_input_buffer()
                ser.close()
                kb.clearEvents()
                print("END", flush=True) #debugging if needed
                core.quit() #quit PsychoPy

    strSerialData = ser.read() #read a single byte from the serial stream, if there is no return from
the T-Pad carry on as using non blocking serial
    strDecodedSerialData = strSerialData.decode() #decode into string
    strDecodedSerialDataBuffer += strDecodedSerialData #build up a XiD return from individual bytes
until a full XiD command has been received terminated in a \n

    if "\n" in strDecodedSerialData: #and booSettingWhiteOptoSensitivity == 1: #check if there has
been a return message from the T-Pad by looking for a terminating \n in the serial stream
        datRTC = datetime.now().strftime('%Y/%m/%d %H:%M:%S.%f')[:-3] #get the PC date and time we
have detected there has been a change on the keyboard we need to report
        booProcessSerialReceived = 1 #flag that serial has been received and we need to process XiD
later on in the messaging loop, i.e. display status of each button, opto etc.

        strXiDChannel = "" + strDecodedSerialDataBuffer[0] #channel
        strXiDPressRelease = "" + strDecodedSerialDataBuffer[2] #press|release
        strXiDSensorNo = "" + strDecodedSerialDataBuffer[4] #sensor
        strXiDmSTimer = "" + strDecodedSerialDataBuffer[6:] #running mS timer

        strDecodedSerialDataBuffer = "" #wipe the strDecodedSerialDataBuffer ready for the next
complete XiD command

    if strXiDChannel == 'A': #channel A is for buttons 1 to 10
        if strXiDPressRelease == 'P': #P is for button press/button down
            if strXiDSensorNo == '1': #check the button number, i.e. button 1 in this case
                booXiDButton1Down = 1 #flag button 1 as being pressed/down
            if strXiDSensorNo == '2':
                booXiDButton2Down = 1
            if strXiDSensorNo == '3':
                booXiDButton3Down = 1
            if strXiDSensorNo == '4':
                booXiDButton4Down = 1
            if strXiDSensorNo == '5':
                booXiDButton5Down = 1
            if strXiDSensorNo == '6':
                booXiDButton6Down = 1
            if strXiDSensorNo == '7':
                booXiDButton7Down = 1
            if strXiDSensorNo == '8':
                booXiDButton8Down = 1
            if strXiDSensorNo == '9':
                booXiDButton9Down = 1
            if strXiDSensorNo == '0':
                booXiDButton10Down = 1
        if strXiDPressRelease == 'R': #R is for button release/button up
            if strXiDSensorNo == '1': #check the button number, i.e. button 1 in this case
                booXiDButton1Down = 0 #flag button 1 as being released/up
            if strXiDSensorNo == '2':
                booXiDButton2Down = 0
            if strXiDSensorNo == '3':
                booXiDButton3Down = 0
            if strXiDSensorNo == '4':
                booXiDButton4Down = 0
            if strXiDSensorNo == '5':
                booXiDButton5Down = 0
            if strXiDSensorNo == '6':
                booXiDButton6Down = 0
            if strXiDSensorNo == '7':
                booXiDButton7Down = 0
            if strXiDSensorNo == '8':
                booXiDButton8Down = 0
            if strXiDSensorNo == '9':
                booXiDButton9Down = 0
            if strXiDSensorNo == '0':

```

```

        booXiDButton10Down = 0

    if strXiDChannel == 'C': #channel C is for optos 1 & 2
        if strXiDPressRelease == 'P': #P is for opto active/on
            if strXiDSensorNo == '1': #check the opto number, i.e. opto 1 in this case
                booXiDOpto1Active = 1
            if strXiDSensorNo == '2':
                booXiDOpto2Active = 1
        if strXiDPressRelease == 'R': #R is for opto non active/off
            if strXiDSensorNo == '1':
                booXiDOpto1Active = 0
            if strXiDSensorNo == '2':
                booXiDOpto2Active = 0

    if strXiDChannel == 'M': #channel M is for voice key or mic
        if strXiDPressRelease == 'P': #P is for microphone active/on
            if strXiDSensorNo == '1': #micriohone 1
                booXiDVKActive = 1
        if strXiDPressRelease == 'R': #R is for microphone non active/off
            if strXiDSensorNo == '1':
                booXiDVKActive = 0

    if strXiDChannel == 'B': #channel B is for TTL in
        if strXiDPressRelease == 'P': #P is for TTL in active 5V/on
            if strXiDSensorNo == '1':
                booXiDTTLActive = 1
        if strXiDPressRelease == 'R': #R is for TTL in non active 0V/off
            if strXiDSensorNo == '1':
                booXiDTTLActive = 0

    if booProcessSerialReceived == 1: #check if there has been a change on any key and if we need to
update the status of each key on screen
        booProcessSerialReceived = 0 #flag that there is no more serial XiD commands to process
        if booXiDButton1Down == 1 or booXiDButton2Down == 1: #accept a response on button 1 or button
2 -- ignore other inputs
            booRunning = 0
            print("Button Pressed XiD: " + strXiDChannel + " " + strXiDPressRelease + " " +
strXiDSensorNo + " " + strXiDmSTimer, flush=True) #debugging if needed
            continueRoutine = False
            break
        else:
            print("Last Raw XiD: " + strXiDChannel + " " + strXiDPressRelease + " " + strXiDSensorNo
+ " " + strXiDmSTimer, flush=True) #debugging if needed
            -----

```

At the end of the experiment the T-Pad is set back to Mode 0 or command mode by sending an X. The serial port is flushed as is the input serial buffer and finally the T-Pad serial port is closed and any HID keyboard events cleared.

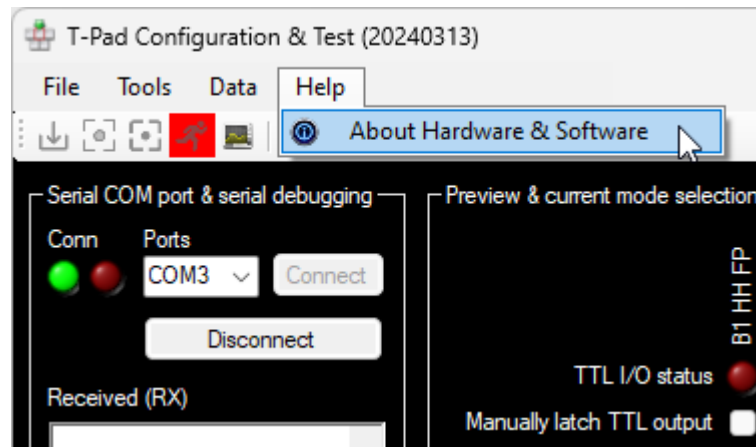
```

END EXPERIMENT
-----
ser.write("X".encode())      #stop doing whatever and return the T-pad to MOD0, i.e. mode 0 or command
                             #mode
core.wait(0.500)            #add a small delay for the T-Pad to come out of whatever mode it is in
ser.flush()                 #flush serial buffer and keyboard
ser.reset_input_buffer()
ser.close()
kb.clearEvents()
print("END", flush=True) #debugging if needed
-----

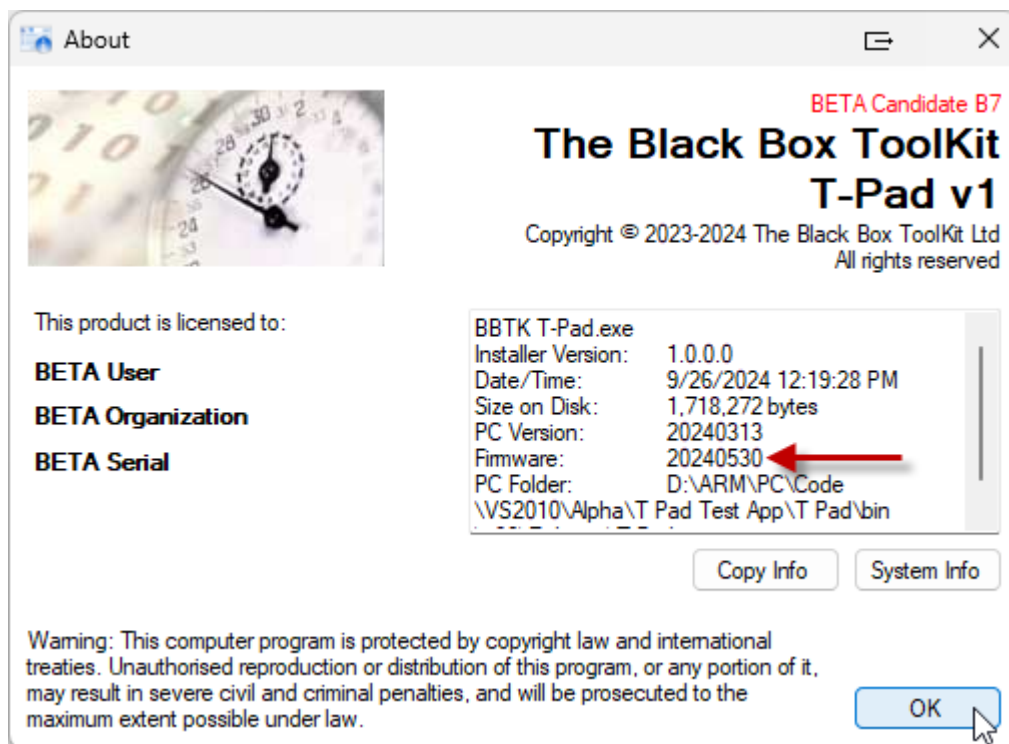
```

15. Checking Firmware Version

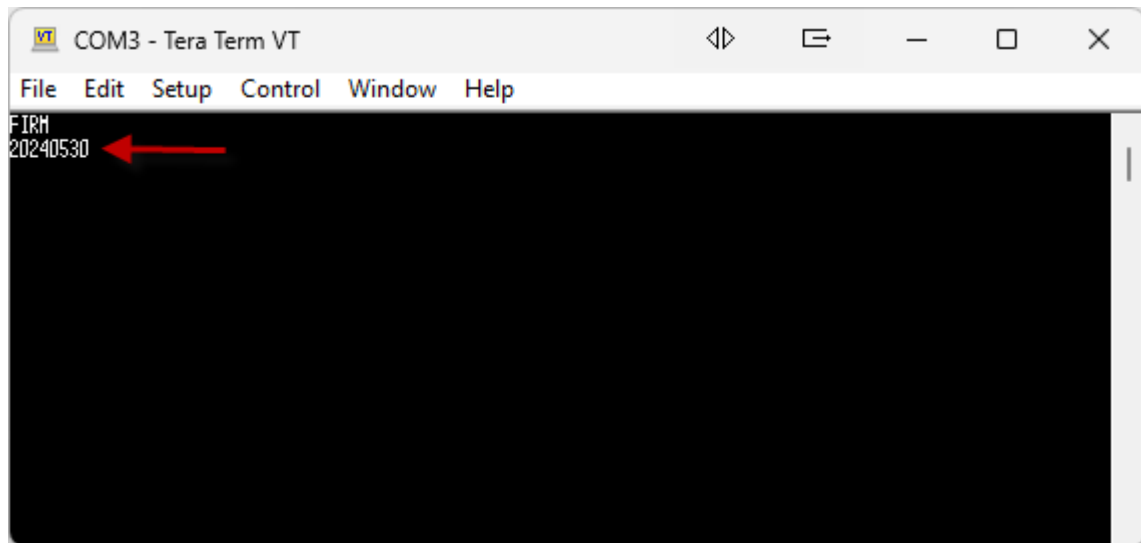
To check the firmware version the easiest method is to use the T-Pad Configuration and Test App and select Help | About Hardware & Software.



An About dialog will then appear showing the firmware version in the format YYYYMMDD along with other useful information.



If you wish to use a serial terminal such as Tera Term you can issue the FIRM API call to retrieve the firmware version.



16. How to Update Firmware in the T-Pad

To update firmware in the T-pad we use Tiny Multi Bootloader+. This is simple to use, but can appear daunting at first glance!

You SHOULD NOT alter any settings in Tiny Multi Bootloader+ and you should not modify either of the INI configuration files.

By default a folder called Firmware will be created under:

C:\BBTK T-Pad\

When you browse the Firmware folder using explorer you will see two configuration INI files, the TMB+ executable used to flash your T-Pad, and two firmware files, i.e. a 'Reset to default startup values.hex' and the version of the firmware your T-Pad shipped with, e.g. 'Firmware V7.08.X.hex'.

Name	Date modified	Type	Size
 config.ini	16/07/2024 15:08	INI File	1 KB
 Firmware V8.10.X.hex	30/05/2024 08:58	HEX File	82 KB
 piccodes.ini	20/10/2023 11:56	INI File	17 KB
 Reset to default startup values.hex	20/10/2023 12:31	HEX File	65 KB
 TinyMultiBootloader+.exe	27/03/2018 11:57	Application	137 KB

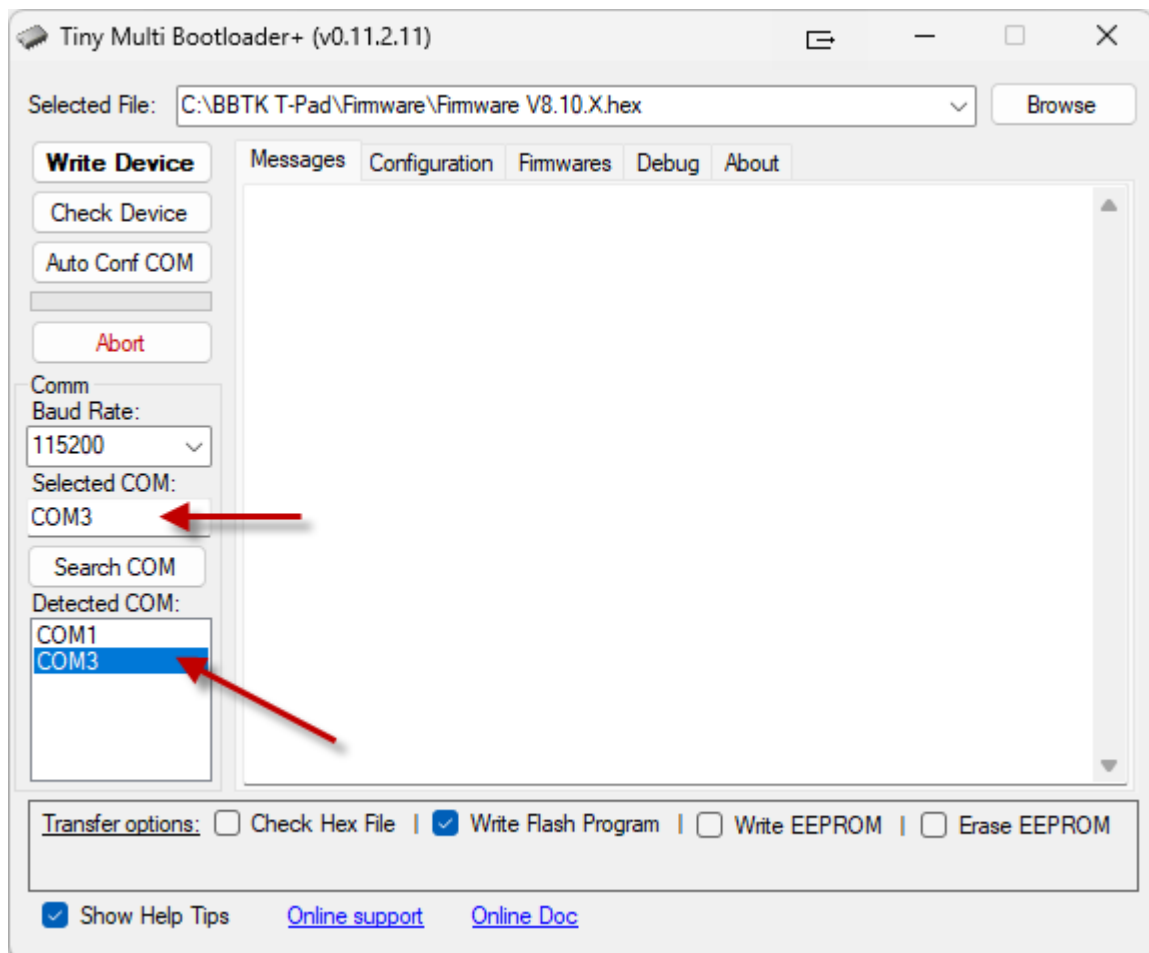
Using Explorer double click on TinyMultiBootloader+.exe to start TMB.

Next select the VCP COM Port your T-Pad is connected to in the Detected COM: box. The Selected COM: should now change to match.

Finally choose the .hex firmware file you wish to flash the T-Pad with. In this example we are going to use, e.g.:

Firmware V8.10.X.hex

DO NOT ALTER ANY OTHER SETTINGS.



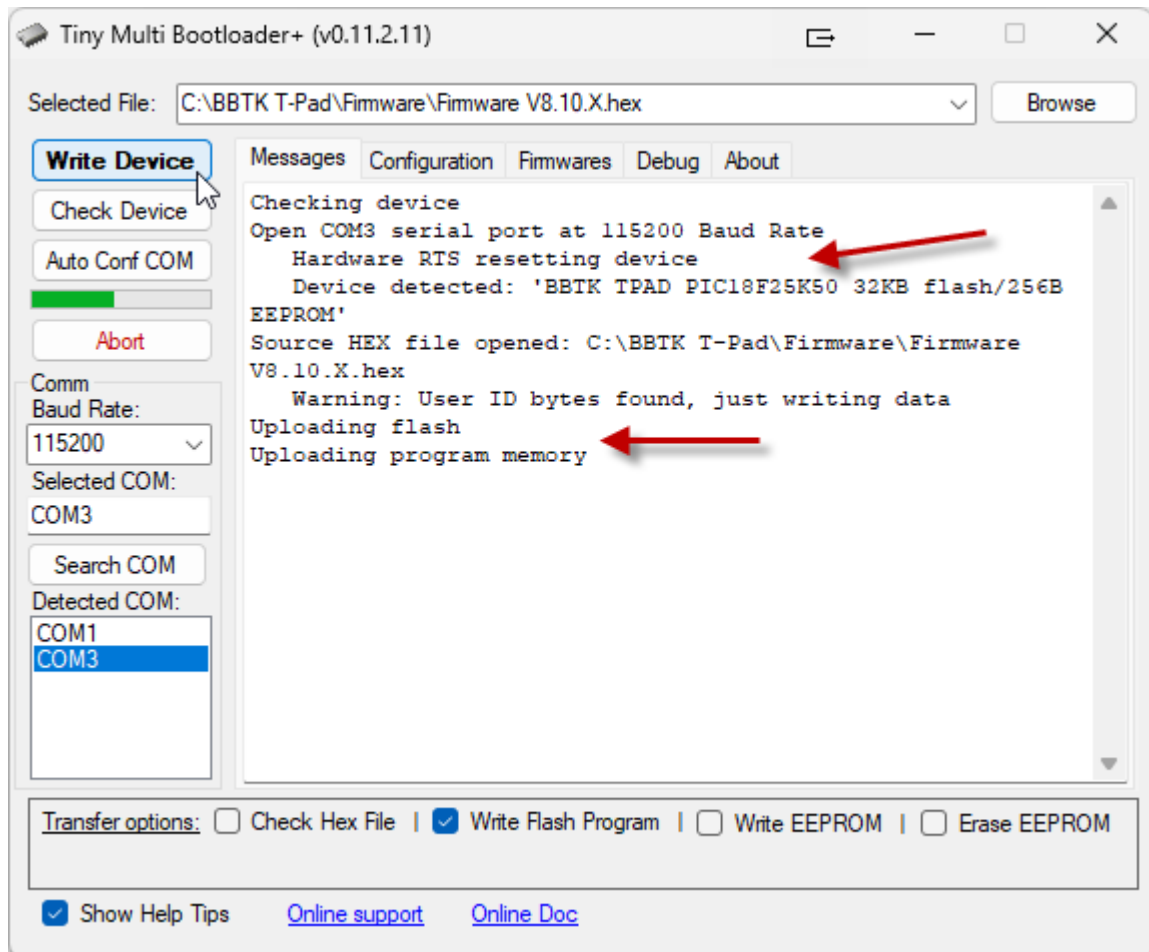
To actually update or flash the firmware into your T-Pad you need to put it in to bootloader mode. Do this by unplugging and re-plugging the serial USB lead.

DO NOT UNPLUG THE USB LEAD WHILE UPDATING THE FIRMWARE.

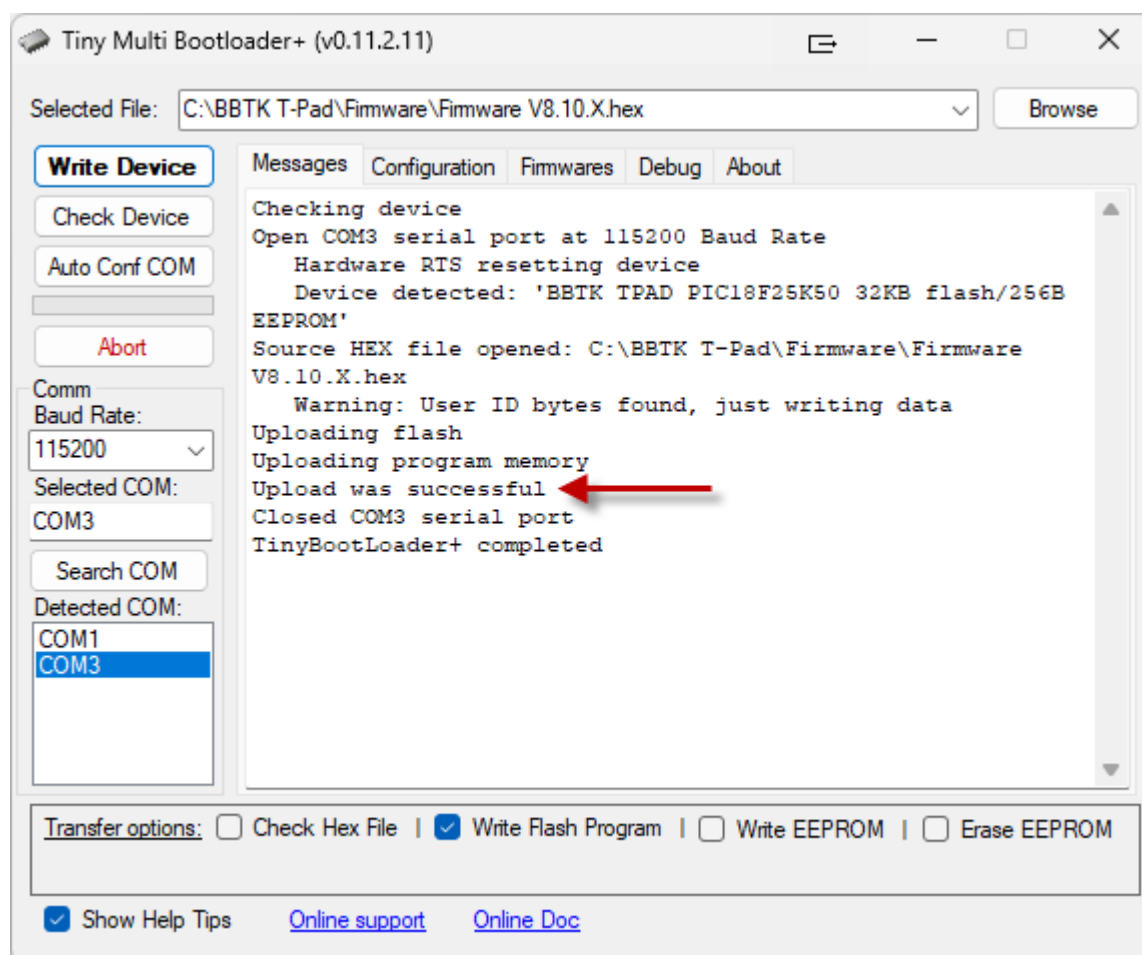
Once you have power-cycled your T-Pad you have 10 seconds in which to start the firmware update using TMB before the T-Pad goes into normal operating mode.

Whilst in this 10 second firmware update window, to begin updating click on the Write Device button.

A device check will then be carried out and if successful Uploading flash will be shown followed by Uploading program memory as shown below.



Once the firmware has been updated, Upload was successful will be displayed and the COM port will be closed.



17. Serial Command API

If using any of the modes that offer a serial connection you should ensure that the Serial USB lead is connected, that the drivers have been installed and latency set to 1 mS in the driver settings.

You should verify latency using the supplied Windows App before collecting timing critical data.

Generic serial connection settings are:

COMx, 115200, 8, N, 1

Sending Serial Commands to the T-Pad

When sending commands to the T-Pad you should wait 25 mS between commands to allow it to process them.

When Reading Serial From the T-Pad

Typically it is best to set a small timeout for when processing serial commands, e.g. 0.0001 which equates to 100 μ S (100 Microseconds).

The reason for setting a very small timeout when reading from the T-Pad is so that your code is more likely to be non-blocking. That is, it will not pause execution for long when waiting for a serial command, i.e. 100 μ S. The T-Pad will send data to your software over serial as fast as possible but so as not to flood, or block, your code, i.e. pause execution, this small delay should give your code chance to process XiD data/commands as they come in.

You should consult the PsychoPy Python code and other examples for details on how this is can be accomplished.

Two examples of how you would open a serial port for reading are shown below:

PsychoPy

```
ser = serial.Serial("COM3", 115200, timeout=0.0001)
```

MATLAB

```
s = serial('COM3');  
set(s, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, 'FlowControl',  
'none', 'Terminator', '\n', 'Timeout', 0.0001);
```

Command	Description	Example Returns
FIRM	Firmware version	YYYYMMDD
RESE	Reset the T-Pad	Resetting – while resetting BBTK – when reset
GETS	Get all settings from the T-Pad	Sending settings x y z... DONE
SAVS	Save all settings to internal NVR RAM so that they will be recalled on next power on	Saving settings – while saving DONE – when saved
TEST	Echo back T-Pads internal hardware mS timer	nnnnn – running elapsed mS hardware timer
REST	Reset T-Pads internal hardware mS timer when in Mode 0, i.e. command mode	0 – when reset to 0
R	Reset T-Pads internal hardware mS timer when in modes 1-5. No CR/LF pair needed and just single character/byte to be sent.	0 – when reset to 0
AAOx val	Where x is Opto 1 or 2 and val can be values 0 to 255. 0 is the least sensitive, i.e. off to 255 which is the most sensitive, e.g. always on.	AAO1 200 – set Opto 1 to sensitivity level 200 Returns 0 1: 0 if Opto not triggered, 1 if triggered
AAVK val	Where val can be 0 to 255. 0 is the least sensitive, i.e. off to 255 which is the most sensitive, e.g. always on.	AAVK 200 – set voice key 1 to sensitivity level 200 Returns 0 1: 0 if voice key not triggered, 1 if triggered
X	Go back to command mode 0 when in modes 1-5. No CR/LF pair needed and just single character/byte to be sent.	MODE 0:
MOD1	Mode 1 – Keyboard HID: Red LED	MODE 1:
MOD2	Mode 2 – Serial VCP: Green LED	MODE 2:
MOD3	Mode 3 – Serial VCP & Keyboard HID: Orange LED	MODE 3:
MOD4	Mode 4 – Keyboard HID & Serial VCP: Flashing Red LED	MODE 4:
MOD5	Mode 5 – Serial VCP: Flashing Green LED	MODE 5:
MOD6	Mode 6 – Serial VCP & BBTK USB TTL emulation mode: Flashing Orange LED	MODE 6:
Z	Returns current mode, i.e. 0 to 5 when in modes 0-5. No CR/LF pair needed and just single character/byte to be sent.	0 – mode 0 etc.

Command	Description	Example Returns
DUMP	In mode 4 returns collected timestamps in XiD format. Each time the T-Pad enters mode 4 the internal memory and timer will reset and start from 0.	Downloading internal memory – signifies start of data upload from the T-Pad 2 – number of samples A P 1 1272 – XiD format data A R 1 2003 – XiD format data DONE – end of uploaded data stream
>>	Check serial driver latency – Start round trip hardware timer in the T-Pad. Note: driver latency testing can only be done in Mode 0, Command Mode.	Nothing
<<	Stop round trip hardware timer in the T-Pad and return round trip time between receiving the two commands, i.e. >> and <<	

18. Mode Summary

18.1. Mode 1 – Keyboard HID: Red LED

Each button, Opto, Voice Key or TTL In when pressed or active produces the following HID keyboard key presses. Key down when button pressed or other sensor active. Key up when button released or other sensor becomes non-active.

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
HID Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n

18.2. Mode 2 – Serial VCP: Green LED

Mode 2 is a purely serial mode where each button press sends a unique character over serial to represent a given button down and another unique character to represent that button being released.

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
Down or Active	1	2	3	4	5	6	7	8	9	0	[	]	-	=
Up or Non-active	!	@	#	\$	%	^	&	*	(	)	{	}	_	+

18.3. Mode 3 – Serial VCP & Keyboard HID: Orange LED

Mode 3 is a hybrid operating mode where both the USB Keyboard HID lead and Serial lead are plugged in and both are live.

For example in this mode when a button is pressed a keyboard keystroke is sent to the PC over the USB Keyboard HID as though a key on a keyboard had been pressed. Simultaneously an XiD protocol command and independent hardware timestamp is also sent over the Serial USB connection.

If button 1 was pressed and held down an XiD command will be sent over serial showing the button down along with the elapsed timestamp, e.g. A P 1 5031.

A P 1 5031

translates to:

Channel [A] buttons | [P]ress | [1] button 1 | [5031] elapsed millisecond timer

[R]elease indicates a button release or sensor going inactive.

The channels that represent buttons and sensors are decoded as below:

Channel	Description			
	Buttons	Opto	Voice Key	TTL In
	A	C	M	T

Keyboard HID

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
HID Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n

18.4. Mode 4 – Keyboard HID & Serial VCP: Flashing Red LED

Mode 4 is a hybrid Keyboard HID mode combined with Serial VCP and works in a similar way to Modes 1 and 3, but without the instant transmission of serial XiD commands to signal which buttons have been pressed or sensors activated together with the elapsed timestamp.

In this mode the T-Pad functions as a standard USB keyboard alone and stores XiD events and hardware timestamps in its onboard internal memory and then uploads all event and timing data at the end of your experiment when you tell it to upload using the keyword DUMP.

Keyboard HID

Input	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Opto 1	Opto 2	Voice Key	TTL In
HID Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n

Note that in Mode 4 nothing will be sent live over serial from the T-Pad.

To stop collecting button presses and other sensor activity send a capital X to the T-Pad over serial to return to Mode 0. Once in Mode 0, send the DUMP command to return all the collected button presses and other sensor data along with timestamps in XiD format.

```

COM3 - Tera Term VT
File Edit Setup Control Window Help
XMODE 0:
MODE 4:
XMODE 0:
DUMP
Downloading internal memory
10
A P 1 4634
A R 1 4808
A P 1 5198
A R 1 5356
A P 1 5741
A R 1 5908
A P 1 6414
A R 1 6541
A P 1 6883
A R 1 6991
DONE
  
```

You can retrieve the data at any time once in Mode 0 by sending DUMP.

The T-Pad will then upload event and timing data. For example:

```

Downloading internal memory
10
A P 1 4634
A R 1 4808
A P 1 5198
A R 1 5356
A P 1 5741
A R 1 5908
A P 1 6414
A R 1 6541
A P 1 6883
A R 1 6991
DONE
  
```

For example from the 10 XiD events uploaded:

A P 1 4634

translates to:

Channel [A] buttons | [P]ress | [1] button 1 | [4634] elapsed millisecond timer

[R]elease indicates a button release or sensor going inactive.

The channels that represent buttons and sensors are decoded as below:

Channel	Description			
	Buttons	Opto	Voice Key	TTL In
	A	C	M	T

18.5. Mode 5 – Serial VCP: Flashing Green LED

Mode 5 is a pure serial VCP mode and sends live XiD commands from the T-Pad as each button is pressed or other sensor is activated.

For example if button 1 was pressed and held down an XiD command will be sent over serial showing the button down along with the elapsed timestamp, e.g. A P 1 5031.

A P 1 5031

translates to:

Channel [A] buttons | [P]ress | [1] button 1 | [5031] elapsed millisecond timer

[R]elease indicates a button release or sensor going inactive.

The channels that represent buttons and sensors are decoded as below:

Channel	Description			
	Buttons	Opto	Voice Key	TTL In
	A	C	M	T

[R] indicates a button release or sensor going inactive.

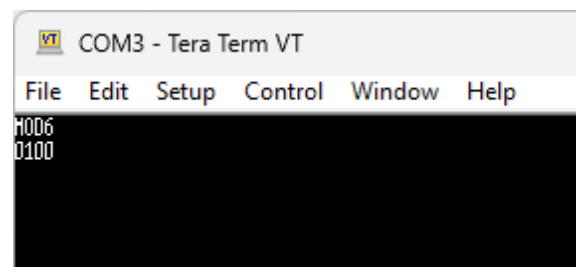
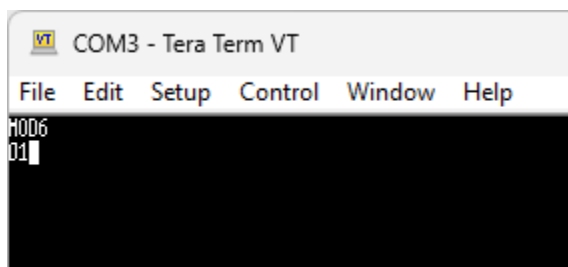
18.6. Mode 6 – USB TTL Module Emulator Serial VCP: Flashing Orange LED

Mode 6 is a pure serial VCP mode and allows the T-Pad to emulate the key features of a BBTK USB TTL Module.

18.6.1. Serial Input From the T-Pad

In this mode two uppercase serial Hex bytes are sent FROM the T-Pad when buttons 1 to 4 are pressed, Opto's 1 or 2 are active, the Voice Key is triggered or there is a TTL In. Note if buttons 5 to 10 are present and pressed and released no button activity will be registered.

For example, if button 1 was pressed, the Hex bytes 01 would be sent from the T-Pad to your PC over serial.



If button 1 was then released Hex bytes 00 would be sent.

Note only two Hex bytes will be sent per change in status. No CR or LF characters are sent FROM the T-Pad.

	Hex received for single button/sensor pressed or active	Input button or sensor mirror	TTL I/O RJ45 socket	Pin
1	01	Button 1	N/A	N/A
2	02	Button 2	N/A	N/A
3	04	Button 3	N/A	N/A
4	08	Button 4	N/A	N/A
5	10	Opto 1	N/A	N/A
6	20	Opto 2	N/A	N/A
7	40	Voice Key	N/A	N/A
8	80	TTL In	2	7

Note that if multiple buttons are pressed or sensors are active you should use the Hex lookup table to determine what is pressed or active. Note that two Hex bytes will only be sent when there is a change. Hex 00 translates to all buttons up and all sensors inactive.

18.6.2. Serial Output to the T-Pad

Individual TTL lines on the T-Pads RJ45 TTL sockets can be turned on and off for event marking by sending two uppercase Hex bytes. No CR or LF characters should be sent TO the T-Pad.

	Hex sent for single button/sensor pressed or active	Output button or sensor mirror	TTL I/O RJ45 socket	Pin
1	01	Button 1	1	1
2	02	Button 2	1	2
3	04	Button 3	1	3
4	08	Button 4	1	4
5	10	Opto 1	1	5
6	20	Opto 2	1	6
7	40	Voice Key	1	7
8	80	TTL In (Button 5)	2	1

Note that TTL outputs are decoupled unlike when running in other modes. That is, if a button is pressed there will be no automatic hardware TTL out to indicate a button had been pressed.

When the two Hex bytes 01 are received from the T-Pad to indicate button 1 had been pressed, you would need to send 01 back out to turn on the TTL corresponding to button 1 being pressed, i.e. TTL I/O RJ45 socket 1 pin 1.

Once a TTL output is active, +5V, it will remain active until that line is turned off when another Hex byte pair is sent. Alternatively all lines can be turned off by sending the Hex byte pair 00. Thus TTL output duration is not tied to continuing physical input and is solely under software control, i.e. you are responsible for the duration and timing of the TTL output once triggered.

Note that TTL In when TTL out event marked is shared with Button 5, i.e. sending Hex byte pair 80 will produce a signal on TTL I/O RJ45 socket 2 pin 1 which represents Button 5 in all other modes.

Commands:

R or RR resets TTL states & input byte buffer
 X quits mode 6 back to mode 0
 >> starts round trip timer
 << stops round trip timer and outputs time taken on the serial port
 ## return ping marker on the serial port
 VV return version number on the serial port

18.6.3. Binary to Hex Decoding Chart

A full binary to hex decoding chart is shown below for reference.

BBTK USB TTL Module Control Values

To reset USB TTL Module send RR - NEVER SEND CR or LF -- ONLY 2 byte upper case letters or numbers

Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)
0	00000000	00	29	00011101	1D	58	001101010	3A	87	01010111	57	116	01110100	74
1	00000001	01	30	00011110	1E	59	001101011	3B	88	01011000	58	117	01110101	75
2	00000010	02	31	00011111	1F	60	00111100	3C	89	01011001	59	118	01110110	76
3	00000011	03	32	00100000	20	61	00111101	3D	90	01011010	5A	119	01110111	77
4	00000100	04	33	00100001	21	62	00111110	3E	91	01011011	5B	120	01111000	78
5	00000101	05	34	00100010	22	63	00111111	3F	92	01011100	5C	121	01111001	79
6	00000110	06	35	00100011	23	64	01000000	40	93	01011101	5D	122	01111010	7A
7	00000111	07	36	00100100	24	65	01000001	41	94	01011110	5E	123	01111011	7B
8	00001000	08	37	00100101	25	66	01000010	42	95	01011111	5F	124	01111100	7C
9	00001001	09	38	00100110	26	67	01000011	43	96	01100000	60	125	01111101	7D
10	00001010	0A	39	00100111	27	68	01000100	44	97	01100001	61	126	01111110	7E
11	00001011	0B	40	00101000	28	69	01000101	45	98	01100010	62	127	01111111	7F
12	00001100	0C	41	00101001	29	70	01000110	46	99	01100011	63	128	10000000	80
13	00001101	0D	42	00101010	2A	71	01000111	47	100	01100100	64	129	10000001	81
14	00001110	0E	43	00101011	2B	72	01001000	48	101	01100101	65	130	10000010	82
15	00001111	0F	44	00101100	2C	73	01001001	49	102	01100110	66	131	10000011	83
16	00010000	10	45	00101101	2D	74	01001010	4A	103	01100111	67	132	10000100	84
17	00010001	11	46	00101110	2E	75	01001011	4B	104	01101000	68	133	10000101	85
18	00010010	12	47	00101111	2F	76	01001100	4C	105	01101001	69	134	10000110	86
19	00010011	13	48	00110000	30	77	01001101	4D	106	01101010	6A	135	10000111	87
20	00010100	14	49	00110001	31	78	01001110	4E	107	01101011	6B	136	10001000	88
21	00010101	15	50	00110010	32	79	01001111	4F	108	01101100	6C	137	10001001	89
22	00010110	16	51	00110011	33	80	01010000	50	109	01101101	6D	138	10001010	8A
23	00010111	17	52	00110100	34	81	01010001	51	110	01101110	6E	139	10001011	8B
24	00011000	18	53	00110101	35	82	01010010	52	111	01101111	6F	139	10001011	8B
25	00011001	19	54	00110110	36	83	01010011	53	112	01110000	70	140	10001100	8C
26	00011010	1A	55	00110111	37	84	01010100	54	113	01110001	71	141	10001101	8D
27	00011011	1B	56	00111000	38	85	01010101	55	114	01110010	72	142	10001110	8E
28	00011100	1C	57	00111001	39	86	01010110	56	115	01110011	73	143	10001111	8F

Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)	Dec	Binary Line Output	2 byte Hex to send (no CR or LF)
144	10010000	90	172	10101100	AC	201	11001001	C9	230	11100110	E6
145	10010001	91	173	10101101	AD	202	11001010	CA	231	11100111	E7
146	10010010	92	174	10101110	AE	203	11001011	CB	232	11101000	E8
147	10010011	93	175	10101111	AF	204	11001100	CC	233	11101001	E9
148	10010100	94	176	10110000	B0	205	11001101	CD	234	11101010	EA
149	10010101	95	177	10110001	B1	206	11001110	CE	235	11101011	EB
150	10010110	96	178	10110010	B2	207	11001111	CF	236	11101100	EC
151	10010111	97	179	10110011	B3	208	11010000	D0	237	11101101	ED
152	10011000	98	180	10110100	B4	209	11010001	D1	238	11101110	EE
153	10011001	99	181	10110101	B5	210	11010010	D2	239	11101111	EF
154	10011010	9A	182	10110110	B6	211	11010011	D3	240	11110000	F0
155	10011011	9B	183	10110111	B7	212	11010100	D4	241	11110001	F1
156	10011100	9C	184	10111000	B8	213	11010101	D5	242	11110010	F2
157	10011101	9D	185	10111001	B9	214	11010110	D6	243	11110011	F3
158	10011110	9E	186	10111010	BA	215	11010111	D7	244	11110100	F4
159	10011111	9F	187	10111011	BB	216	11011000	D8	245	11110101	F5
160	10100000	A0	188	10111100	BC	217	11011001	D9	246	11110110	F6
161	10100001	A1	189	10111101	BD	218	11011010	DA	247	11110111	F7
162	10100010	A2	190	10111110	BE	219	11011011	DB	248	11110000	F8
163	10100011	A3	191	10111111	BF	220	11011100	DC	249	11111001	F9
164	10100100	A4	192	11000000	C0	221	11011101	DD	250	11111010	FA
165	10100101	A5	193	11000001	C1	222	11011110	DE	251	11111011	FB
166	10100110	A6	194	11000010	C2	223	11011111	DF	252	11111100	FC
167	10100111	A7	195	11000011	C3	224	11100000	E0	253	11111101	FD
168	10101000	A8	196	11000100	C4	225	11100001	E1	254	11111110	FE
169	10101001	A9	197	11000101	C5	226	11100010	E2	255	11111111	FF
170	10101010	AA	198	11000110	C6	227	11100011	E3			
170	10101010	AA	199	11000111	C7	228	11100100	E4			
171	10101011	AB	200	11001000	C8	229	11100101	E5			

19. Specifications

19.1. Key Features

- Sleek carbon fiber enclosure (300 mm x 250 mm LW)
- 48 MHz CPU
- 128K byte SPI memory
- Hardware based timer (max time approx. 279 mins)
- Firmware updateable by provided app
- USB HID keyboard connection
- USB virtual serial com port connection @ 115,200 baud
- Up to 10 buttons (24 mm or 30 mm diameter arcade quality with choice of colors) which can be located anywhere within the lid
- 5 external 3.5mm double button connections allows for 10x hand-held buttons or 10x foot pedals (or combination of the two)
- 1 or 2 Opto's (auto sensitivity setting)
- 1 Voice Key or microphone
- Built-in TTL event marking/TTL triggers for EEG, biofeedback, fMRI and time audit on 2 x RJ45 sockets (13 x TTL Out corresponding 10 buttons, 2 Opto's & 1 Voice Key)
- 1 x TTL In
- Settings stored in non-volatile RAM once set using the App, e.g. activation thresholds, keystrokes to output, set TTL inputs and outputs etc.
- Platforms supported:
 - Microsoft Windows XP SP3, Vista SP2 (32/64), Windows 7 SP1 (32/64), Windows 8 (32/64), Windows 8.1 (32/64), Windows 10 (32/64), Windows 11 (64 bit)
 - macOS[#]
 - Linux[#]

[#]BBTK Microsoft Windows software Apps not supported directly. Full API provided for control over serial regardless of platform used. USB Keyboard HID natively supported via Plug and Play (PnP).

19.2. Rear Connectors and Pinouts

RJ45 – TTL I/O 1

Pin 1 - TTL Out 1 corresponding to button 1
Pin 2 - TTL Out 2 corresponding to button 2
Pin 3 - TTL Out 3 corresponding to button 3
Pin 4 - TTL Out 4 corresponding to button 4
Pin 5 - TTL Out 11 corresponding to opto 1
Pin 6 - TTL Out 12 corresponding to opto 2
Pin 7 - TTL Out 13 corresponding to the voice key
Pin 8 - Ground

RJ45 – TTL I/O 2

Pin 1 - TTL Out 5 corresponding to button 5
Pin 2 - TTL Out 6 corresponding to button 6
Pin 3 - TTL Out 7 corresponding to button 7
Pin 4 - TTL Out 8 corresponding to button 8
Pin 5 - TTL Out 9 corresponding to button 9
Pin 6 - TTL Out 10 corresponding to button 10
Pin 7 - TTL In 1
Pin 8 - Ground

USB – Keyboard

USB keyboard HID

USB – Serial

VCP serial port

3.5 mm stereo connectors – But 1/6, 2/7, 3/8, 4/9, 5/10

Each connector can accept 2 push to make contacts to enable external buttons
Button 1 for example would be on the tip of the 3.5mm connector and button 6 on the ring

3.5 mm stereo connector – Opto

The stereo 3.5mm connector can accept 2 BBTK optos

3.5mm stereo connector – Mic

Voice key using condenser microphone

19.3. Timing Specifications

Mode 1 – Keyboard HID: Red LED

- All inputs scanned every - Min 190 μ S, Max 300 μ S
- Button down to TTL out - Min 50 μ S, Max 350 μ S
- Opto/VK on to TTL out - Min 50 μ S, Max 350 μ S
- Button down to key onto HID - Min 135 μ S, Max 400 μ S
- Opto/VK on to key onto HID - Min 135 μ S, Max 400 μ S
- Maximum input change rate - 200 changes per sec

Mode 2 – Serial VCP: Green LED

- All inputs scanned every - Min 190 μ S, Max 300 μ S
- Button down to TTL out - Min 50 μ S, Max 350 μ S
- Opto/VK on to TTL out - Min 50 μ S, Max 350 μ S
- Button down to end of serial transmission - Min 140 μ S, Max 440 μ S
- Maximum input change rate - 200 changes per sec

Mode 3 – Serial VCP & Keyboard HID: Orange LED

- All inputs scanned every - Min 250 μ S, Max 1 mS
- Button down to TTL out - Min 165 μ S, Max 460 μ S
- Opto/VK on to TTL out - Min 165 μ S, Max 460 μ S
- Button down to key onto HID - Min 250 μ S, Max 400 μ S
- Opto/VK on to key onto HID - Min 250 μ S, Max 400 μ S
- Button down to end of serial transmission - ~2.12 mS
- Maximum input change rate - 200 changes per sec

Mode 4 – Keyboard HID & Serial VCP: Flashing Red LED

- All inputs scanned every - Min 190 μ S, Max 300 μ S
- Button down to TTL out - Min 50 μ S, Max 350 μ S
- Opto/VK on to TTL out - Min 50 μ S, Max 350 μ S
- Button down to key onto HID - Min 135 μ S, Max 400 μ S
- Opto/VK on to key onto HID - Min 135 μ S, Max 400 μ S
- Maximum input change rate - 200 changes per sec

Mode 5 – Serial VCP: Flashing Green LED

- All inputs scanned every - Min 250 μ S, Max 1 mS
- Button down to TTL out - Min 165 μ S, Max 460 μ S
- Opto/VK on to TTL out - Min 165 μ S, Max 460 μ S
- Button down to key onto HID - Min 250 μ S, Max 400 μ S
- Opto/VK on to key onto HID - Min 250 μ S, Max 400 μ S
- Button down to end of serial transmission - ~2.12 mS
- Maximum input change rate - 200 changes per sec

Mode 6 – Serial VCP: Flashing Orange LED

- Receiving of serial command to TTL out - Min 350 μ S, Max 520 μ S
- TTL in to serial data sent - Min 260 μ S, Max 360 μ S

Internal Clock

- +/- 0.00202% error