Understanding Digital I/O

TDT RZ Systems

Updated 2025-02-27

Tucker-Davis Technologies
11930 Research Circle
Alachua, FL 32615 USA
Phone: +1.386.462.9622
Fax: +1.386.462.5365

Notices

The information contained in this document is provided "as is," and is subject to being changed, without notice. TDT shall not be liable for errors or damages in connection with the furnishing, use, or performance of this document or of any information contained herein.

The latest versions of TDT documents are always online at https://www.tdt.com/docs/

Table of Contents

Understanding Digital I/O with TDT

Equipment setups for neuroscience experiments are complex. Even routine behavioral or optogenetic experiments normally involve multiple pieces of hardware and software that need to communicate with one another. Common examples of experiments where devices need to send signals back and forth include: Trigger pulses for external lasers, capture of behavioral events like lever presses, frame synchronization of video cameras, and timing synchronization between two recording platforms. This communication is done using digital input and output (digital I/O) signals on each device.

TDT hardware and software have the ability to communicate with external devices. The most important knowledge for understanding how to setup digital I/O in your experiment is to make sure the hardware connections are correct and that your software is receiving or sending digital I/O on the channel you expect.

TDT Hardware Page

Synapse Digital IO Page

What to Expect

This technical guide will offer readers a full understanding of how digital I/O communication works in TDT equipment and software. This will be done by going over the questions below.

Bit-related:

- What is a TTL pulse? What is bit communication and why do we use it?
- Where and how does TDT receive a TTL pulse?
- How can TDT make and send out a TTL pulse?
- What does this saved data look like?

Byte-related:

- What is a Byte? What is word-communication and why do we use it?
- What is Binary? How do binary numbers related to digital words?
- Where and how does TDT receive bytes?

- How can TDT make and send out a byte?
- What does this saved data look like?

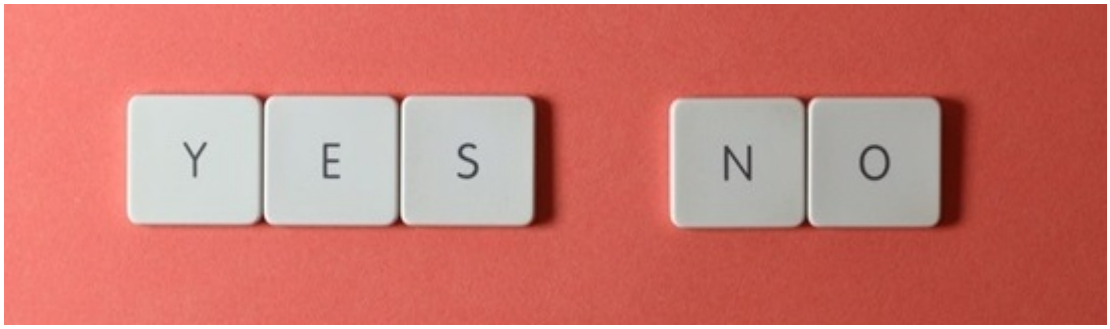
By the end of this guide, you will be able to easily add one or more peripheral devices that sends or receives triggers to or from TDT equipment. If the resources you are needing are not covered in this guide, or if you have any questions that you need help with, please do not hesitate to ask support@tdt.com for assistance.

Common Definitions

Here are some common definitions that you will be exposed to throughout the reading. These also link out to relevant sections:

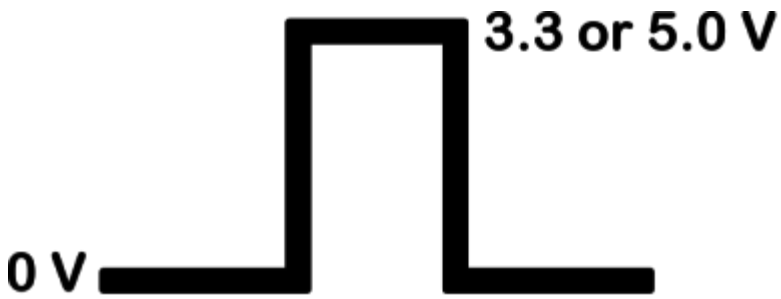
I/O	Input/ Output
TTL	Transistor-Transistor Logic. This is a fixed high voltage or low voltage signal. Often used in the context of 'TTL pulse'
Bit	The state of a signal that is either voltage high or voltage low
Binary	Something that only has two possible states. In the case of a digital bit, it's 'true' or 'not true'
BNC	A type of coaxial electrical cable commonly used in TTL communication
Epoc	A saved timestamp in your TDT data that marks a digital input or output event
Byte	A set of eight bits that are monitored simultaneously
Digital Word	The numeric decimal value (0 - 255) of a byte
Most-significant digit	The left-most digit in a number representing the largest base group value. The most-significant digit in the base 10 number '137' is the '1' which represents a value of '100 (10 groups of 10)'
Least-significant digit	The right-most digit in a number representing the largest base group value. The most-significant digit in the base 10 number '137' is the '7' which represents how many ones remain in the number when all larger groups have been accounted for
Ground	An important signal line needed for connecting TDT and third-party equipment

Bit Communication (TTL Pulses)



All TDT digital I/O communications use 'TTL signals'. TTL stands for 'transistor-transistor logic,' but you don't need to understand transistors or have an electrical engineering degree to use these signals. A TTL signal is very simple: it's a voltage signal that is set at either a fixed high value or zero to indicate whether something is on or off, kind of like a light switch. We call this high or low state a 'bit'. Unlike analog signals which can be any value within a specified range, a digital bit is binary - the signal is either true or it is not true.

TTL signals have two voltage states - high or low. The high voltage is either 3.3 V or 5.0 V and the low voltage is always 0 V.



Important

TDT digital I/O can tolerate up to 5.0 V, but bit (Port C) should only go up to 3.3 V. Our official recommendation is that if you have a consistent 5.0 V signals, then you need to use Ports A or B, which use bytes for communication. Your third-party device will indicate the TTL voltage.

We often refer to TTL signals as 'TTL pulses' because they are square. TTL pulses can become high or low as long as needed to indicate a state behavior, or they can be programmed for a set period of time to indicate an event. The latter is commonly used for synchronizing or triggering third-party equipment like external cameras, lasers, Arduinos, other recording systems, FED3 and peripheral behavioral devices. Some of these send out TTL pulses to mark short events

into TDT that we capture and record. Others require TDT to send TTL pulses to them to trigger events or synchronize recording clocks.

> **Note**
>
> The 'on' and 'off' state for a device does not need to be a high voltage for 'on' and zero for 'off.' These can actually be flipped. The external device's state behavior can be easily deduced in synapse by recording a few test TTL pulse inputs. This information should also be in the device manuals.

BNC Connections

The easiest and most common way to connect devices for individual TTL communication (0 - 3.3 V signals) is with a BNC cable.

You've likely seen these around any neuroscience lab. They are single-channel coaxial cables that you twist onto to secure into place.

> **Note**
>
> The term BNC is short for "Bayonet Neill–Concelman" named for being a bayonet connector and the two co-inventors Wiki Reference

Because of their commonality, TDT is well-suited for making connections to BNC cables. The RZ2, RZ5, RZ10x, and some iCon modules all have built-in BNC connections for digital I/O that map to bit-addressable memory. The RZ6 does not, but you can use our PP24 breakout connection board to easily provide BNC access to the RZ6 or any RZ device. This is convenient if you have more TTL signals than available BNC ports and want to use BNC cables for your digital connections.

> **Note**
>
> The bit I/O channels on the PP24 are labeled under 'Port C,' which happen to be connectors A1 - A8 on the PP24

Another option is to use a BNC to flying leads cable if you have a third-party device with screw terminals and want to connect to a TDT BNC, or you are using a device with a BNC and want to connect directly to the TDT DB25 connector.

Setup Bit Inputs in Synapse

Setting up a TTL inputs for bit communication (a single channel/ signal) in Synapse is very easy. If you have the physical connections to your other equipment ready and that equipment can send out test pulses, then all you have to do is activate the correct bit channel (normally one of the C0 - C3 bits) in Synapse.

> **Digital I/O - Bit Input**
>
> ### Digital I/O - Bit Input
>
> Read a TTL bit input from the RZ into Synapse.

From a TDT-perspective, receiving TTL pulses is passive. If you are not seeing signals appear in Synapse and you think you should, please do the following troubleshooting steps:

1. Verify that your BNC connection is correct and that you are connected to the correct port on your TDT device.

2. Verify that you have the digital input setup correctly in Synapse.
3. Autoscale your TTL trace in the Synapse RunTime plot if you see a flat green line.
4. If you don't see anything, verify that your third-party device is actually sending out a TTL pulse. This is usually done with an oscilloscope. Most of the time, this is the problem.
5. Sometimes BNC cables break. Try replacing it.
6. If you see TTL pulses in an oscilloscope but still do not see it in TDT, contact support@tdt.com for next steps. Please make notes or take pictures of steps up to this point for proof of troubleshooting.

Setup Bit Outputs in Synapse

TDT TTL ports can be setup for outputs as well. This will be accompanied by a signal in Synapse that you want to use to send to another device. TDT has a number of gizmos that can be used to send logic signals to a specified digital IO port to generate a 0 - 3.3 V TTL pulse.

The simplest example is using the Pulse Generator. In the below example, the gizmo generates a logic pulse that we then select in Port C0 in the RZ digital IO tab as an output signal. The bit-addressable ports in Synapse only accept logic signals (these are marked green on gizmo connection diagrams) for specified output signal IDs.

> **Pulse Gen with TTL Output**
>
> ### Pulse Gen with TTL Output
>
> Use the Pulse Generator Gizmo to make dynamically-controlled pulse trains that can be output at TTL signals to control external devices like lasers.

Similarly, you can generate user-defined single pulses with the User Input gizmo.

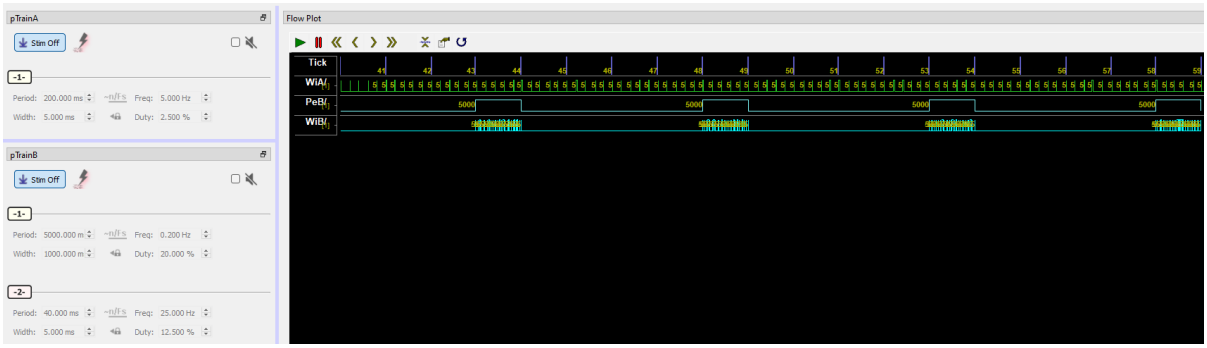
User Input

User Input

Use the User Input Gizmo to make user-defined triggers via software buttons, or to receive TTL bit inputs from the Digital I/O.

If you want to make more complicated pulse trains, such as bursting patterns with higher frequency pulse trains that occur at a specified interval, then check out the Pulse Train Generator. This gizmo uses a Parent-Child setup where a child train can only be true if the parent train is also true. You can change the train stack in the General > Configuration tab in the gizmo. The pTrain can also send out simple regular pulses like the Pulse Generator.

Below is an example output of two pTrain configurations. pTrainA (WiA/, colored green) is a simple 5 Hz pulsing pattern (pulse width = 5 ms). The other pTrainB (PeB/ and WiB/, colored cyan) shows a stacked logic pattern for bursts of pulses. The parent train has a Period of 5000 ms and a Width of 1000 ms - this means that every 5 seconds it will be ON for 1 second. The child train has a Period = 40 ms and a Width = 5 ms. The child train can only be active when the parent train is active, so the result is that every 5 seconds there is a burst of 25 Hz, 5 ms pulses that last for 1 second in total duration.



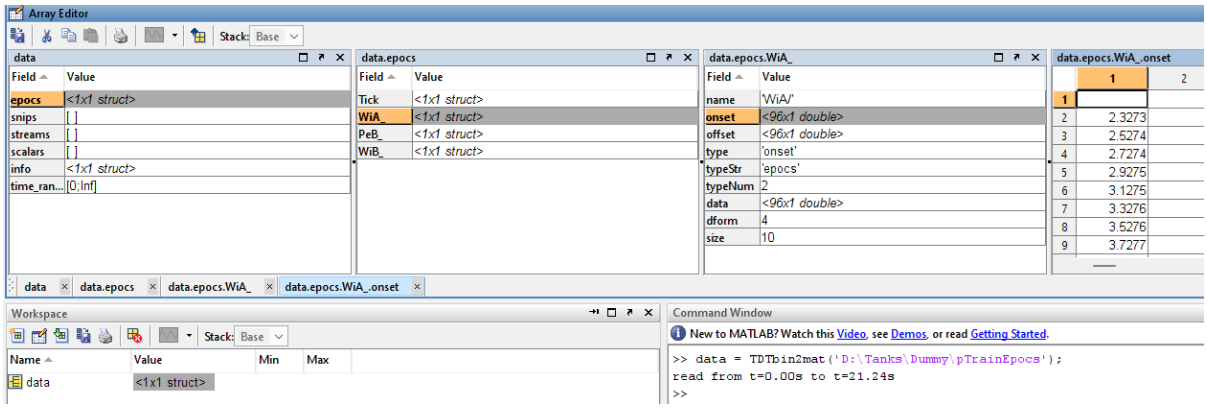
You can download the example experiment here: :material-download: Download Experiment File

There are other gizmos in synapse that can generate logic pulses to be used for outputs. The stimulation gizmos can output a 'StimSync' pulse when they are active (example using the Auditory Stimulation gizmo). This pulse can be routed to a digital output.

Bit Epocs in the Data

TTL data gets saved as an 'epoc' in Synapse. An Epoc is a timestamp event with distinct name, an onset time, offset time, and a data value. Epoc events are important for marking events in your data for time reference comparisons. Here is everything you need to know about importing data into Matlab or Python.

The below image shows the data structure that was saved from the example experiment TwoPTrains in the above Setup Bit Outputs in Synapse. In the image, we see one of the epocs, WiA from pTrainA, and look into the onset timestamps for every triggered event.



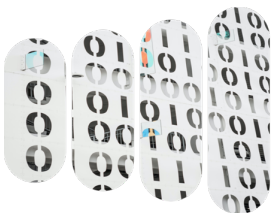
Epocs are commonly used for peri-event filtering to make raster plots, histograms, peri-event response plots, and much more. Below are three posted examples that use epocs for data-related analysis:

Averaging around an epoc event

PSTH using epocs and spike snippet data

Peri-event plot using detected behavioral input events and fiber photometry data

Byte Communication (Digital Words)

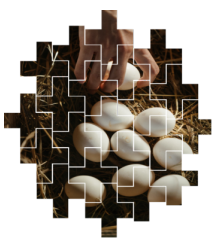


Individual TTL signals use bit-addressable memory in the TDT digital IO for communication. But what if we have many signals that we want to record from another device or send out to control several things? This is where byte-addressable memory comes in. Unlike bits, which are saved individually as 0 or 1, a 'byte' is a group of eight bits that is read as an entire set at once (whenever there is a state change). The value of these eight bits makes up the byte value, or 'digital word.'

This type of communication is useful for saving a large variety of information that can come from another device. For example, if you have an operant chamber with levers, lights, infrared sensors, lick spouts, shock grids, etc, then you will want to keep track of which equipment is active and not active. This information lets you know the overall behavioral state of the animal. These combinations of many states are difficult to track with individual TTL pulses. Instead, you likely want to read the combination of these signals as a single state to be able to easily identify an event code with a certain behavioral state.

Because reading byte values requires understanding binary numbers, we are going to make a quick detour into learning how to count in groups of 10 (which you already know how to do) and then in groups of 2 (just like computers!). After you know how binary numbers works, we will look how this applies to TDT digital IO by looking at the physical connections and software setup in Synapse.

How Do We Make Natural Numbers?



Positive whole numbers like 1, 2, 3, etc are called 'Natural Numbers.' If you see a basket full of eggs and want to know how many there are, you count them one at a time. One egg is 1, two eggs is 1 egg then another 1 egg, and so on. You can represent any number of eggs just by counting one egg at a time.

But what if you have more than one? Do you have say "I have 1 egg, 1 egg total?" No, you would say "I have two eggs." What if you had a dozen eggs, or one hundred eggs? Using one to represent those larger numbers doesn't make much sense And where do these whole numbers come from anyways? How do we know what value those numbers actually have?

Humans solved this problem a long time ago by creating 'number bases' that can be used for counting and representing any number larger than one.

To make a number base to count the natural numbers, all you have to do is pick a natural number bigger than 1 (2, 10, 21, N) and use that as a marker to track how many groups of that base number you have. If you have N groups of N things, then you keep track of that to make. Since you are splitting the larger set up into groups and only counting the groups, it makes the total number of things to keep track of much smaller. The way this works on paper is that base numbers are actually raised to an exponent that represents the 'zero group,' 'ones group,' 'twos group,' etc. But let's look at an example and see what this all really means.

Understanding Numbers in Base 10

Humans count in base 10. This makes a lot of sense because we have 10 fingers. So we naturally form our basis of representing numbers around 10 because that's the highest number we can keep track of with our digits before needing a place holder. But we don't just stop at 'how many groups of ten' we have. We keep track of how many ones, tens, hundreds, thousands, and beyond are in a total number we count. Or, you can think of it as how many ones, tens, tens of tens, hundreds of tens, and so forth.

> **Note**
>
> The prefix 'dec' comes from the latin root 'decas' which means 'set of ten,' and that's where decimal, decade, decimate, and other words come from

For a practical example, we are going to make the numbers 1, 2, 10, 11, 100, and 137 using the base 10 system. Each number is read from left to right, where the largest set in that number is on the left and the smallest group is on the right. We call that the 'Most significant digit' and the 'least significant digit.' Every next number to the left of the least significant digit represents, as its base, 10 times more than the previous digit's group. In a base 10 numbering system, the least significant digit is the ones group, which represents the remainder of items left over when all the larger groups have been counted.

To make the numbers 1 - 9, you just count the total individual ones you have. A 1 is represented in base 10 as:

$1 = 10^0$

It is raised to the power of 0 because you have 0 groups of ten so far.

To make the other 2 - 9, you just count the total individual ones you have. A 2 is represented in base 10 as:

$2 = 10^0 + 10^0$

To make the number ten you raise 10 to the power of 1 because you have one group of ten now:

$10 = 10^1$

But notice something **very** important. This is written as 10, not just 1. You have to keep track of all the groups you've included so far. A ten is one group of ten and zero groups of ones.

$10 = 1 \times 10^1 + 0 \times 10^0$

The number 11 is one group of tens and a one left over:

$11 = 10 + 1 = 10^1 + 10^0$

If you have ten groups of ten, you have 100:

$100 = 10^2$

or

$100 = 1 \times 10^2 + 0 \times 10^1 + 0 \times 10^0$

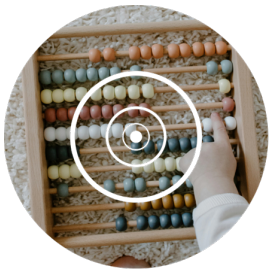
Finally, the number 137 is a combination of all three groups - one group of ten tens, three groups of ten, and seven ones left over:

$137 = 100 + 10 + 10 + 10 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 10^2 + 10^1 + 10^1 + 10^1 + 10^0 + 10^0 + 10^0 + 10^0 + 10^0 + 10^0 + 10^0$

or

$137 = 1 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$

Numbers in Base 2 and Reading Binary



Now that we understand how to make numbers using a base 10 system, we can apply that same knowledge to a base 2 system.

Binary numbers are represented using a base 2 system. Computers can only count up to 1 because they have 'bits' instead of fingers and these bits are either on or off, 1 or 0. That is a total of two states that can be represented in a bit.

However, representing numbers larger than 1 in computers is not a problem because of the base 2 number system. We can just keep track of the groups of ones, twos, two twos, and so forth using the same type of counting method as base 10. The only difference is that instead of using values up to 9 as a group multiplier, we only go up to 1. Our bits, which can be 1 or 0, will represent the group order we are keeping track of.

To make the number 1, you say you have zero groups of two as before. We set the zeroth bit as true (multiply it by 1):

$1 = 1 \times 2^0$

The number 2 is one group of 2 and no groups of ones. So we set the 1^{st} bit as true and the 0^{th} bit as false and write it like this:

2 = 10

Wait! What? Two does not equal ten. Don't be alarmed! Remember, we can only go up to a value of 1 to represent each group value. Break it down into its subgroups:

$2 = 10 = 1 \times 2^1 + 0 \times 2^0$

Keep in mind that the 2^1 is not incorrect or cheating because we are keeping track of how many groups of twos we have. For this reason, we can drop the N multiplier in front of each group since each group is either one or off, 1 or 0. If a group is not used, we still keep track of it as a 0 in the binary number.

Three is one group of two and one group of one:

$3 = 11 = 2^1 + 2^0$

How about 4, which is two groups of two and no ones:

$4 = 100 = 2^2$

Again, we can only go up to 1, which is why we don't write 4 as 20. We must increment our most significant digit to represent every next base power of 2.

$7 = 111 = 2^2 + 2^1 + 2^0$

$11 = 1011 = 2^3 + 2^1 + 2^0$

and so forth, up to 8 bits total (since we started at 0 for the base power, we can go up to 7 for the exponent):

$137 = 10001001 = 2^7 + 2^3 + 2^0$

If you get stuck, Windows has a really neat 'Programmer' feature on their calculator tool that you can use to put in a decimal number and see its binary representation. It breaks the bits up into groups of 4 for ease of reading.

Bits Make Bytes

Now that we know how to represent numbers in binary, we can move on to how this applies to digital I/O in TDT.

As we already covered, an individual TTL pulse is a single voltage trace that signals when something is on or off, 1 or 0. That sounds a lot like how a bit works. Indeed, TDT equipment is not just limited to reading individual TTL signals. You can combine these signals or 'bits' together as groups of 8 in order to read or write larger binary numbers. This group of 8 bits is called a byte, as mentioned previously.

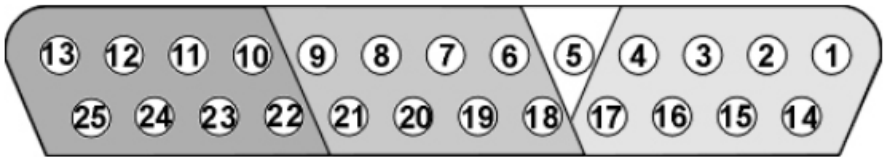
If you do the math, or play around with the calculator tool, you will find that the binary number `1111 1111` is the decimal number 255. This value represents every bit in a byte being on. A byte on the TDT digital I/O can either read values from 0 - 255 or it can write values from 0 -

255. Every bit is written or read at the same exact time and a single value is reported. We call this 'word-addressable' I/O, because you read all the individual bits together like a word. Calling it byte-addressable memory works, too.

TDT Digital IO DB25 Connector

The TDT RZ processors have a DB25 connector for the digital I/O. This connector is divided into groups A, B, and C that represent each 8-bit byte grouping. Don't get intimidated by all the numbers and letters. Remember that all this is for is to make numbers from 0 - 255 using signals that are either on or off.

DB25 Digital I/O Pinout

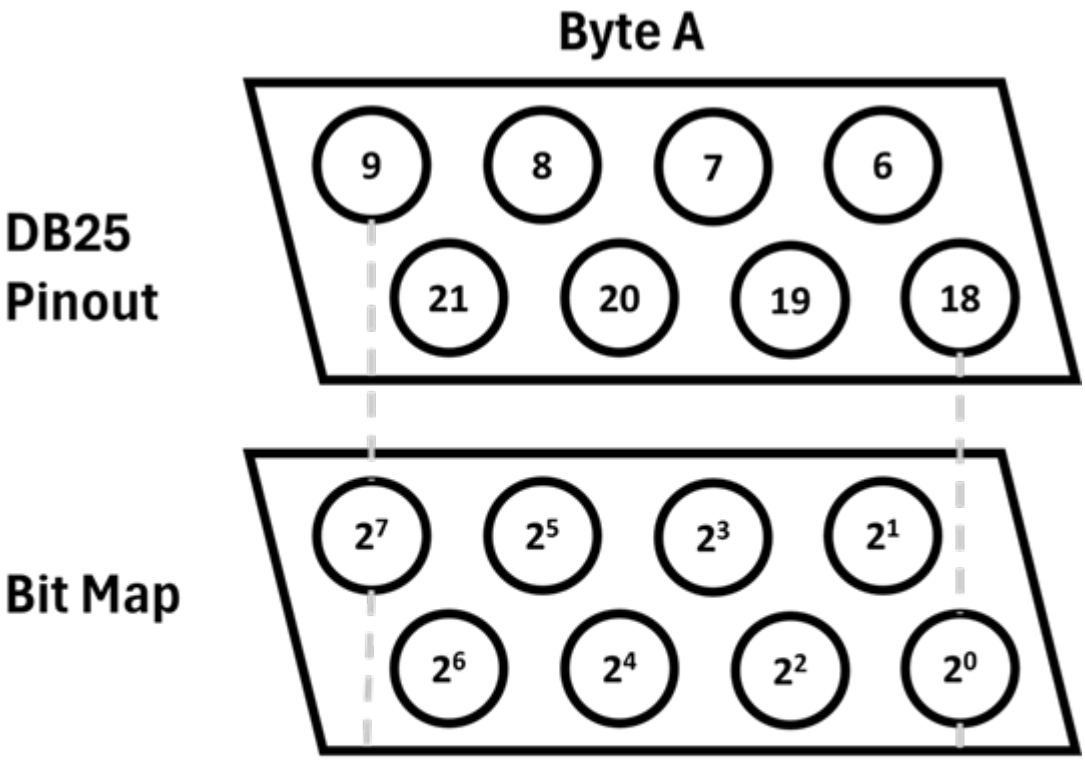


Pin	Name	Description	Pin	Name	Description
1	C0	Port C Bit Addressable	14	C1	Port C Bit Addressable
2	C2		15	C3	
3	C4		16	C5	
4	C6		17	C7	
5	GND	Digital I/O Ground	18	A0	Port A Word Addressable
6	A1	Port A Word Addressable	19	A2	
7	A3		20	A4	
8	A5		21	A6	
9	A7		22	B0	Port B Word Addressable
10	B1	Port B Word Addressable	23	B2	
11	B3		24	B4	
12	B5		25	B6	
13	B7				

The numbers 1 - 25 in the pinout are just the physical pin positions for each signal line. Signal number 5 is the ground, which leaves 24 pins left for digital I/O.

Byte C is normally used for the normal single TTL bit-addressable communication because each bit in the Byte can be individually read directly in Synapse. For the Word-addressable I/O, focus on Byte A and B.

Byte A is made up of pins 6 - 9 and 18 - 21 on the DB25 connector. Pin 18 is the least significant bit (or digit) in the byte and pin 9 is the most significant bit. Pin 18 is the 2^0 group. Pin 9 is the 2^7 group. That's what we mean by bit 0 and bit 7, respectively, in the table. Again, bits are counted from 0 to 7 in a byte because they represent the base power order of a particular twos group, just like how we showed in the earlier section on binary numbers Binary Numbers.



Bit #	7	6	5	4	3	2	1	0
Value	X	X	X	X	X	X	X	X

Byte A Bit Table

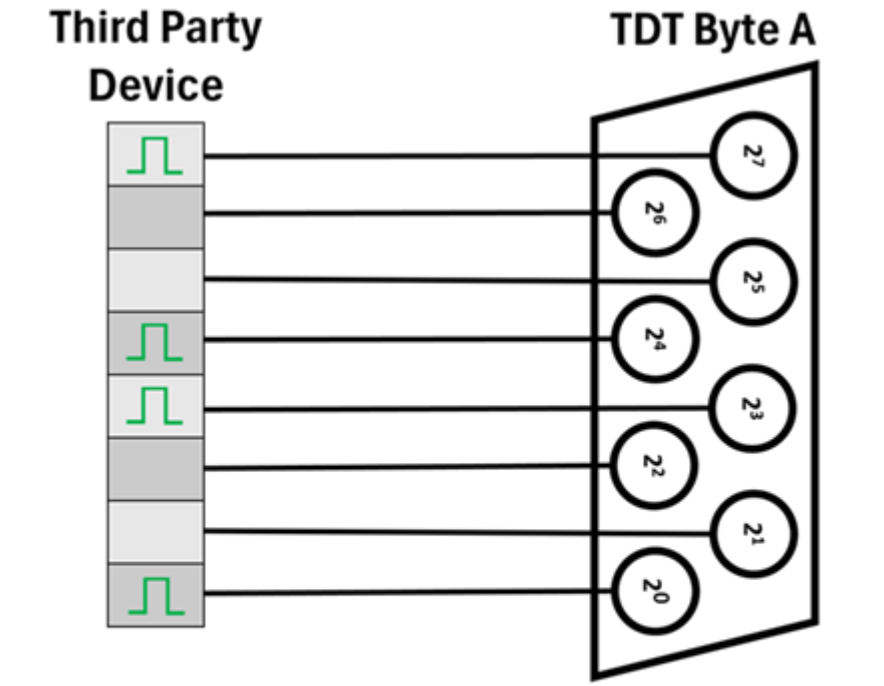
Connecting to Third-Party Devices

Unless you have an RZ2, which has eight BNC connectors that access each bit of Port A, then you will need to use the DB25 connector on the front bottom of your RZ device. You would also need to do this for Port B of the RZ2.

The DB25 can be accessed in a number of ways. TDT makes a DB25-DB25 cable to connect to MedAssociates' SuperPort. You can also use a PP24 to breakout each channel in the 24 bit digital IO into a BNC connector. Other methods that involve accessing pins directly or user-made DB25 cable connections are explained in A Note About Ground because extra considerations are needed for the ground signal.

Word Inputs

We now have a clear picture of what bit each pin corresponds to. Let's see an example of a third-party device communicating with TDT. The third party device is sending TTLs from its output ports. These outputs will physically connect to the pins of Byte A on the TDT side. When the values on the TDT side change all eight bits in Byte A are read at once and the word value is captured.



Bit #	7	6	5	4	3	2	1	0
Value	1	0	0	1	1	0	0	1

Word Value to or from TDT = 153

TDT captures word values upon changes in the byte, so if the state of any of the eight bits were to change, then a new word value would be recorded.

From now on, we will use a bit table to show any communication on TDT bytes.

Here is a video showing how to set up a word input for Byte A in Synapse.

> **Digital I/O - Word Input**
>
> ## Digital I/O - Word Input
>
> Read an 8-bit digital word input from the RZ into Synapse.

You can also save individual bits from each word input by using the LowerBits or UpperBits gizmos that TDT provides in Custom > TDT > Logic.

The Lower Bits gizmo attaches to your Port A (or Port B) and will parse out the first four bits 0 - 3 individually and make them individual outputs to which you can then attached an Epoc

Store. The same concept applies for the upper bits 4 – 7 (using the Upper Bits gizmo). This should let you save each individual bit from the port to which you are attached.

You can download the example experiment here (note that all the bits are not saved in this example and you can add more epoc stores as needed): :material-download: Download Experiment File

Word Outputs

The same principles for word (byte) inputs apply to word outputs from a TDT perspective. Eight bits of the target byte (usually byte A) are set and those bits simultaneously send out TTL pulses (or not) according to their output value.

In order to send words out of TDT you must provide the target byte with an integer. If you are targeting just byte A or byte B individually, that integer value should be between 0 - 255. Some example integer to bit assignment in the TDT byte are below:

Integer Value vs Bit Table Assignment

Integer = 1

Bit #	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	1

Integer = 28

Bit #	7	6	5	4	3	2	1	0
Value	0	0	0	1	1	1	0	0

Integer = 137

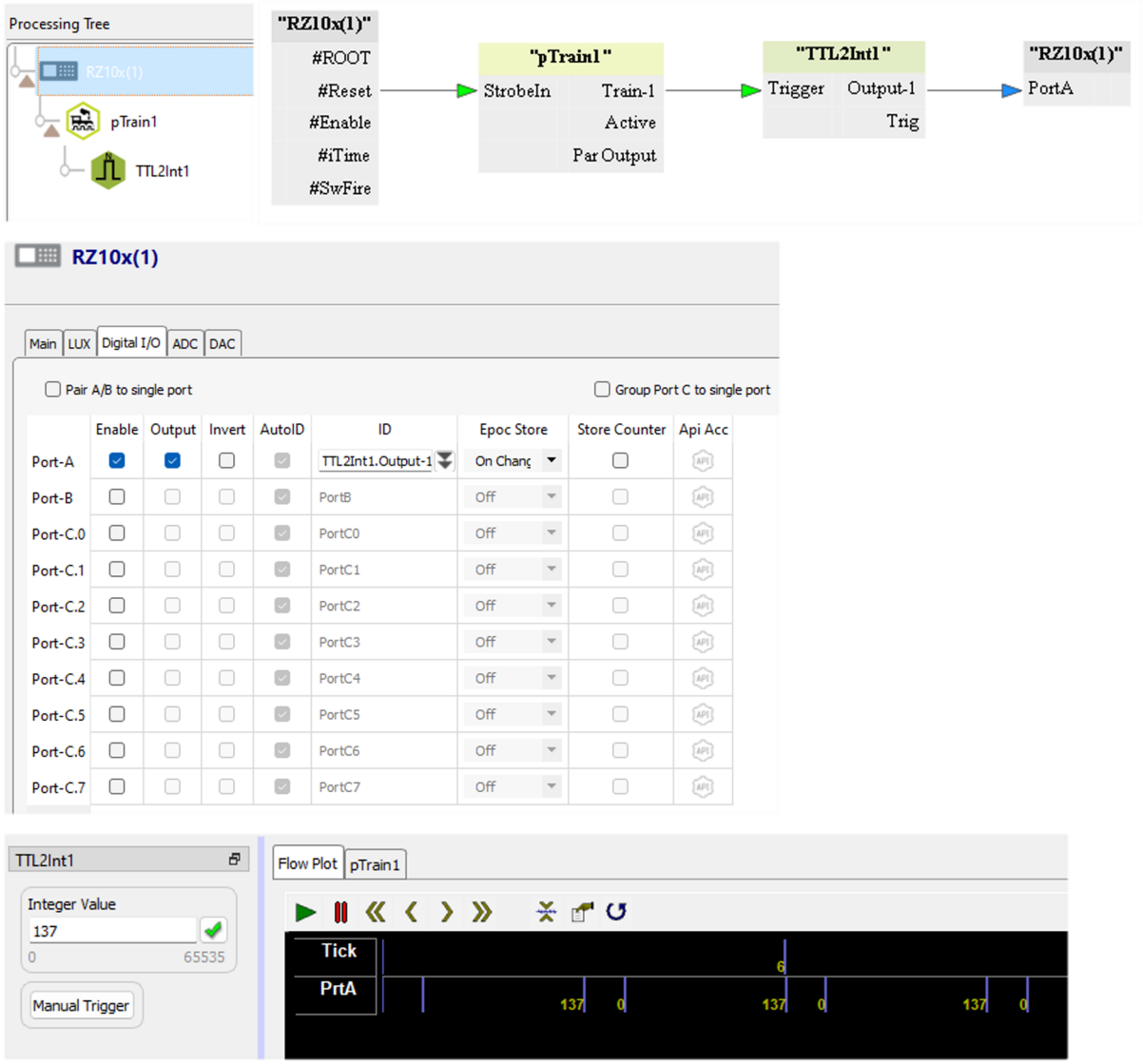
Bit #	7	6	5	4	3	2	1	0
Value	1	0	0	0	1	0	0	1

Note

Bytes A and B can be combined into a 16-bit word to send or receive much larger values up to 65535

TDT offers a TTL2Int gizmo in Synapse that is frequently used to send integers to bytes A or B for output. This gizmo can be controlled by an upstream logic strobe (from a pulse train, user input, external trigger, or any other logic signal) to output a user-set integer value upon being triggered. In the example experiment below, we use a logic pulse from a pTrain gizmo to trigger

the TTL2Int gizmo to output a value of 137 for the duration of the pTrain pulse and send it to PortA.



You can download a copy of this experiment here: :material-download: Download Experiment File

> **Note**
>
> The TT2Int gizmo (located in Custom > TDT > Logic) in v98 and earlier has an edge detect on the strobe input. This means that the integer output will last for a single sample. The provided experiment above has a modified version where the integer output is true for the duration of the logic strobe signal.

Byte Epocs in the Data

Byte data also gets saved as an epoc in Synapse. Unlike a bit epoc, which saves a distinct onset/ offset for each individual IO channel, bytes (words) from PortA or PortB save integer values 'On Change.' This means that a new integer value will get saved when there is a change of state in any of the 8 bits in the byte. Not every bit might have changed at that instant, but the total byte gets saved anyways.

Below is an example of how On Change tracks byte changes:

Integer = 1

Bit #	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	1

Integer = 3

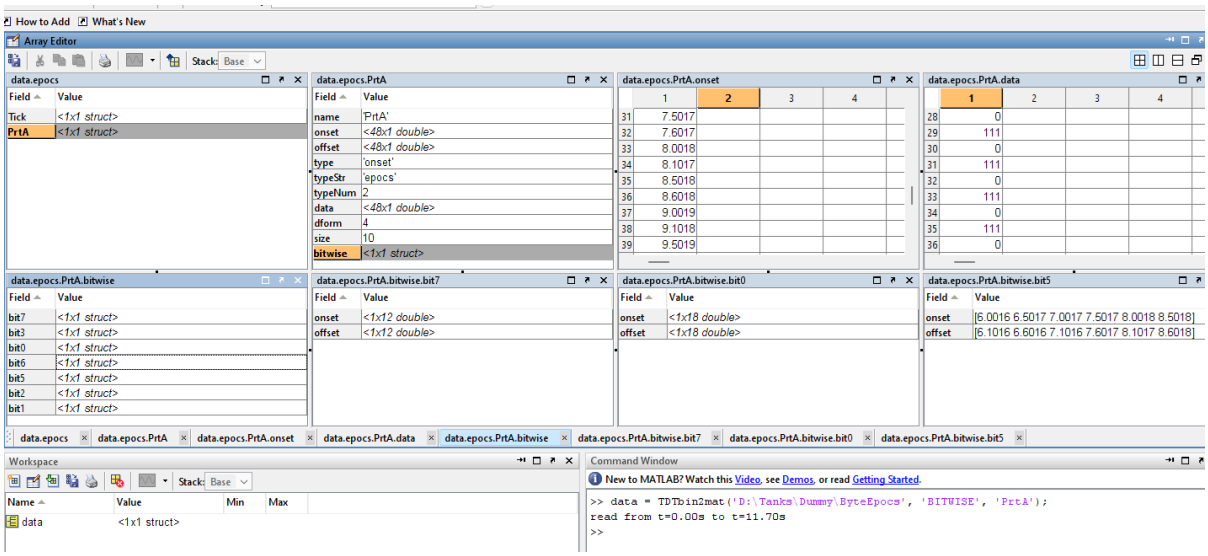
Bit #	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	1	1

Integer = 2

Bit #	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	1	0

Each integer is a word value from the state of all 8 bits in the Port A byte. When '1' is saved, bit 0 is true. If bit 1 also turns on, then a value of '3' is saved, but bit 0 remained on and did not change. Finally, if bit 0 turns off, then a value of '2' is saved. This might seem apparent with a small example of only two bits, but when many bits of different order are changing, it can be difficult to know which bits are on or off (except if the number is odd, then you always know bit 0 is true).

The byte data gets saved as an epoc with onset times, offset times, and data that represent the word value saved on any given state change. If you have a lot of bits in your byte changing and want to know when each was on or off individually, then you can use the BitBreakout experiment referenced in Word Inputs or you can use a function call in TDTbin2mat called 'Bitwise' that parses each of the bit states individually.



As you can see in the example data, bitwise parsed any bit of the 8 bit byte that changed when going through values of 137, 111, and 10. Note that bit 4 is not present in data.epocs.PortA.bitwise. That is because bit 4 never changed states in any of those word values.

You can filter around individual bits from a byte save, or you can filter around data values from a byte if you know which value corresponds to your event of interest. We have an example that does this here.

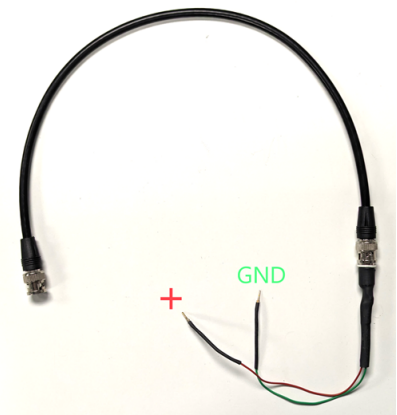
A Note About Ground

TDT digital signals use an active voltage and ground. Any third-party device will also have a ground line that needs to connect to the TDT ground, otherwise the two systems will be 'floating' with respect to one another and there could be a large enough voltage difference between each system to create false TTL signals.

Think about this like a trampoline - if you and your friends are jumping on the same surface, then you know how high you are with respect to one another. But if one of you was on the earth and the other was on an elevated trampoline, then there is already a height difference between you two. When the person on the earth jumps, they need to go much higher in order to be 'above' the trampoline. When the person on the trampoline jumps, they don't need to go as far before they reach a large absolute difference in height between themselves and their friend.

If you are using a BNC cable on both ends, then the ground is on the BNC shell and you don't need to worry about manually making the ground connection.

If you have a BNC on one end (usually the TDT side) and flying leads on the other (usually the third-party equipment side), then be sure to connect the negative side of the flying leads (usually green or black colored wire) to the third-party ground port. The red lead is your active signal.



If you are connecting a DB25 cable or individual wires to the TDT digital IO port DB25 connector, be sure to use pin 5 on the TDT side as a common ground for all your devices. If you have one third-party device, its ground must connect to pin 5 on the TDT DB25 connector. If you have multiple third party devices, they all must connect to pin 5.

You can find DB25 to terminal screw port adapters that are very helpful if you are connecting multiple bare wires to the DB25. Google "DB25 D-sub Male Adapter Plate RS232 to Terminal Signal Module Breakout Board"