

Produkt-Datenblatt

Technische Daten, Spezifikationen

Kontakt

**Technischer und kaufmännischer Vertrieb, Preis-
auskünfte, Angebote, Test-Geräte, Beratung vor Ort:**

Tel: (0 81 41) 52 71-0

FAX: (0 81 41) 52 71-129

Aus dem Ausland:

Tel: ++49 - 81 41 - 52 71-0

FAX: ++49 - 81 41 - 52 71-129

E-Mail: sales@meilhaus.com

Internet:

www.meilhaus.com

Web-Shop:

www.MEsstechnik24.de | www.MEasurement24.com

Web Kontakt-Formular:

www.meilhaus.de/infos/Kontakt.htm

Per Post:

Meilhaus Electronic GmbH

Am Sonnenlicht 2

D-82239 Alling bei München

MEsstechnik fängt mit ME an.

www.meilhaus.com

Erwähnte Firmen- und Produktnamen sind zum Teil
eingetragene Warenzeichen der jeweiligen Hersteller.
Preise in Euro zzgl. gesetzl. MwSt. Irrtum und Änderung
vorbehalten.

© Meilhaus Electronic bzw. Hersteller:
www.meilhaus.de/infos/impressum.htm


MEILHAUS
ELECTRONIC

LabJack

Published on LabJack (<http://labjack.com>)

[Home](#) > T7 Datasheet

T7 Datasheet

[details](#)

High performance multifunction DAQ with USB, Ethernet, and WiFi.

This datasheet covers all T7 variants: T7, T7-OEM, T7-PRO, and T7-PRO-OEM.

Most information in this datasheet applies to all T7 variants. Information about WiFi and the high-resolution ADC (ResolutionIndex = 9-12) only applies to the -Pro variants. There is an [OEM section](#) with information specific to the build of OEM versions.

PDF Datasheet

If you are looking at a PDF or hardcopy, realize that it is likely out-of-date as the original is an online document. Also, this datasheet is designed as online documentation, so the formatting of a PDF is often less than ideal.

To make a PDF of the whole manual, click "Export all" towards the upper-right of this page. To make a PDF of a particular section go to that page and click "Export all" towards the upper-right of that page. Doing so converts these pages to a PDF on-the-fly, using the latest content, and can take 20-30 seconds. Make sure you have a current browser (we mostly test in Firefox and Chrome) and the current version of Acrobat Reader. If it is not working for you, rather than a normal click of "Export all" do a right-click and select "Save link as" or similar. If exporting the entire datasheet, it can take 20-30 seconds and then a dialog box will pop up asking you where to save the PDF. Then you can open it in the real Acrobat Reader, not embedded in a browser. If you still have problems, try the "Print all" option instead.

Rather than using a PDF, though, we encourage you to use this web-based documentation. Some advantages:

- We can quickly improve and update content.
- Click-able links to further or related details throughout the online document.
- The site search includes the datasheet, forum, and all other resources at labjack.com. When you are looking for something try using the site search.
- For support, try going to the applicable datasheet page and post a comment. When appropriate we can then immediately add/change content on that page to address the question.

Occasionally we export a PDF and attach it to this page (below).

Navigating the Datasheet using the Table of Contents

An efficient way to navigate this online datasheet is to browse the table of contents to the left. Rather than clicking on all the links to browse, you can click on the small black triangles to expand without reloading the whole page.



[Datasheet](#)

Preface: Warranty, Liability, Compliance

For the latest version of this and other documents, go to www.labjack.com.

Copyright 2013, LabJack Corporation

Warranty:

The LabJack T7 is covered by a 1 year limited warranty from LabJack Corporation, covering this product and parts against defects in material or workmanship. The LabJack can be damaged by misconnection (such as connecting 120 VAC to any of the screw terminals), and this warranty does not cover damage obviously caused by the customer. If you have a problem, contact support@labjack.com for return authorization. In the case of warranty repairs, the customer is responsible for shipping to LabJack Corporation, and LabJack Corporation will pay for the return shipping.

Limitation of Liability:

LabJack designs and manufactures measurement and automation peripherals that enable the connection of a PC to the real-world. Although LabJacks have various redundant protection mechanisms, it is possible, in the case of improper and/or unreasonable use, to damage the LabJack and even the PC to which it is connected. LabJack Corporation will not be liable for any such damage.

Except as specified herein, LabJack Corporation makes no warranties, express or implied, including but not limited to any implied warranty or merchantability or fitness for a particular purpose. LabJack Corporation shall not be liable for any special, indirect, incidental or consequential damages or losses, including loss of data, arising from any cause or theory.

LabJacks and associated products are not designed to be a critical component in life support or systems where malfunction can reasonably be expected to result in personal injury. Customers using these products in such applications do so at their own risk and agree to fully indemnify LabJack Corporation for any damages resulting from such applications.

LabJack assumes no liability for applications assistance or customer product design. Customers are responsible for their applications using LabJack products. To minimize the risks associated with customer applications, customers should provide adequate design and operating safeguards.

Reproduction of products or written or electronic information from LabJack Corporation is prohibited without permission. Reproduction of any of these with alteration is an unfair and deceptive business practice.

Conformity Information (FCC, CE, RoHS):

See the [Conformity Page](#) and the text below:

FCC PART 15 STATEMENTS:

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense. The end user of this product should be aware that any changes or modifications made to this equipment without the approval of the manufacturer could result in the product not meeting the Class A limits, in which case the FCC could void the user's authority to operate the equipment.

Declaration of Conformity:

Manufacturers Name: LabJack Corporation
Manufacturers Address: 3232 S Vance St STE 100, Lakewood, CO 80227, USA

Declares that the product

Product Name: LabJack T7 (-Pro)
Model Number: LJT7 (-Pro)

conforms to the following Product Specifications:

EMC Directive: 2004/104/EEC

EN 55011 Class A
EN 61326-1: General Requirements

and is marked with CE

RoHS:

The T7 (-Pro) is RoHS compliant per the requirements of Directive 2002/95/EC.

Preface

1.0 Device Overview

This document contains device-specific information for the following devices:

- T7
- T7-Pro
- T7-OEM
- T7-Pro-OEM

This family introduces a new line of high-quality analog and Ethernet data acquisition hardware, with the main traditional advantage of all LabJack data acquisition hardware, namely, high performance and rich feature set at a competitive price point. These features make the T series a logical choice for many high-performance applications, where Ethernet, WiFi, and cost are primary considerations.

1.1 Core Features

Analog I/O

- 14 Analog Inputs (16-18+ Bits Depending on Speed)
- Single-Ended Inputs (14) or Differential Inputs (7)
- Instrumentation Amplifier Inputs
- Software Programmable Gains of x1, x10, x100, and x1000
- Analog Input Ranges of ± 10 , ± 1 , ± 0.1 , and ± 0.01 Volts
- 2 Analog Outputs (12-Bit, ~ 0 -5 Volts)

Digital I/O

- 23 Digital I/O
- Supports SPI, I2C, and Asynchronous Serial Protocols (Master Only)
- Supports Software or Hardware Timed Acquisition
- Maximum Input Stream Rate of 100 kHz (Depending on Resolution)
- Capable of Command/Response Times Less Than 1 Millisecond

Digital I/O Extended Features

- Simple PWM Output (1-32 bit)
- PWM Output w/ phase control
- Pulse Output w/ phase control
- Positive edge capture
- Negative edge capture
- PWM measure
- Edge capture & compare
- High speed counter (TBD ~ 40 MHz)
- Software counter (TBD ~ 200 kHz)
- Software counter w/ debounce
- Quadrature Input

Other highlights

- Built-In CJC Temperature Sensor
- Watchdog system
- Field Upgradable Firmware
- Programmable Startup Defaults
- LJTick Compatible

Fixed Current Outputs

- 200 μ A
- 10 μ A

1.2 Family Variants Info

T7 vs T7-Pro

The T7-Pro has all features of the normal T7, with the following added:

- Wireless Ethernet 802.11b/g
- 24-bit Low-Speed ADC for 22-Bit Effective Resolution

Also see the block diagram in the [hardware overview section](#).

T7-OEM and T7-Pro-OEM





There is also an OEM version of the T7 and T7-Pro. The OEM versions are the same in terms of features, but the enclosure, and most connectors are not installed on the OEM versions, which allows customers to easily configure as needed. See [Appendix A - OEM Versions](#) for details.

2.0 Installation

1. First install LabJack software and driver bundle based on your operating system.

T7/Digit Devices



	Windows Installer	49.87 MB	2014-02-27 15:24
	Mac OS X Package	2.73 MB	2014-01-22 19:29
	Linux 32-bit Package	1.2 MB	2014-01-22 17:17
	Linux 64-bit Package	1.19 MB	2014-01-22 19:32

2. Connect the T7 to the local computer via USB.
3. Proceed through any steps to add new hardware.
4. If using Windows, open Kipling (installed with package above). Utility apps for other operating systems are still under development.
5. Use the dashboard in Kipling to view analog inputs, digital I/O, DAC outputs, etc.
6. Go to [quickstart page](#) to see more about Kipling and its use with the T7.

3.0 Communication

Modbus TCP is the protocol used by all connections on the T7(USB, Ethernet, WiFi). All important values and data from the device can be read and/or written by using the associated Modbus register(s). Thus, the process for reading the serial number, an analog input, or a waveform is all functionally the same, you simply provide a different address. There are two main ways to communicate with a T7 using Modbus TCP.

Communication Options

High-level [LJM library](#)

Among other useful features, this cross-platform library allows users to access registers by name, such as "AIN4" for analog input 4. Most people will use the LJM library since they're familiar with writing code, and want to integrate a T7 into an existing software framework.

Conceptual workflow:

1. Find [example code/wrappers](#) for your desired programming language.
2. Use the [LJM_Open\(\)](#) function to open a connection to the T7.
3. Perform reads and writes to [Modbus registers](#) using [LJM_eReadName\(\)](#) or [LJM_eWriteName\(\)](#).
4. Use the [Close\(\)](#) function to close the connection.

Direct [Modbus TCP, Clients](#)

It is easy to integrate a T7 over Ethernet or Wi-Fi into standard COTS Modbus software platforms, since the T7 is directly compatible. People who already use Modbus software will find this option convenient. Some COTS Modbus software is very powerful, and will save users the time and money required to develop their own software.

Conceptual workflow:

1. Configure the power-up-default registers on the T7 using the [Kipling](#) software program. Change [Ethernet/WiFi](#) IP settings, any relevant analog input settings, etc. '..._DEFAULT' registers indicate that they are power-up-defaults.
2. Open COTS Modbus program.
3. Specify the Modbus registers by address, such as 8, for AIN4. Find applicable registers with the [register look-up tool](#), or by referencing the datasheet etc.
4. See data directly from the T7 in COTS software.

Communication Speed Considerations

There are two alternate methods for data transfer to occur, command response is the lowest latency, and streaming offers the highest data throughput, the following sections provide more detail.

Command-Response

This is the default behavior for communication with a device, and most people find the data throughput satisfactory. Direct Modbus interactions will always use command-response. The high-level LJM library also uses command-response.

Communication is initiated by a command from the host which is followed by a response from the device. In other words, data transfer is software-paced. Command-response is generally used at 1000 scans/second or slower and is generally simpler than stream mode.

Command-response mode is generally best for minimum-latency applications such as feedback control. By latency here we mean the time from when a reading is acquired to when it is available in the host software. A reading or group of readings can be acquired in times on the order of a millisecond. See [Appendix A-1](#) for details on c-r data rates.

Stream Mode

Stream mode is generally best for maximum-throughput applications. However, streaming is not recommended for feedback control operations, due to the latency in data recovery. Data is acquired very fast, but to sustain the fast rates it must be buffered and moved from the device to the host in large chunks. Streaming is only available through the high-level LJM library, and is not possible using direct Modbus communication.

Stream mode is a continuous hardware-paced input mode where a list of addresses is scanned at a specified scan rate. The scan rate specifies the interval between the beginning of each scan. The samples within each scan are acquired as fast as possible. As samples are collected automatically by the device, they are placed in a buffer on the device, until retrieved by the host. Stream mode is generally used when command-response is not fast enough. Stream mode is not supported on the hi-res converter (resolutions 9-12 not supported in stream).

For example, a typical stream application might set up the device to acquire a single analog input at 100,000 samples/second. The device moves this data to the host in chunks of 25 samples each. The LJM library moves data from the USB host memory to the software memory in chunks of 2000 samples. The user application might read data from memory once a second in a chunk of 100,000 samples. The computer has no problem retrieving, processing, and storing, 100k samples once per second, but it could not do that with a single sample 100k times per second. See [Appendix-A-1](#) for details on stream mode data rates.

Command-response can be done while streaming, but streaming needs exclusive control of the analog input system so analog inputs (including the internal temperature sensor) cannot be read via command-response while a stream is running.

3.1 Modbus Map

This utility is used to find Modbus registers that pertain to the T7, simply select the T7 from the device filter, and then narrow down results using either the search tool, or the tag filters. All of these registers use command-response, unless specified as STREAM.

<http://labjack.com/support/modbus/map>

We distribute a constants file called "ljm_constants.json" that defines all information about the Modbus register map. The dynamic filter & search tool below pulls it's data from that JSON file.

- Name: A string name that can be used with the LJM library to access each register.
- Address: The starting address of each register.
- Details: Click to get text pulled from the description field in the JSON.
- Type: Specifies the datatype, which also tells you how many registers each value uses.
- Access: Read-only, write-only, or read & write.
- Tags: Used to associate registers with particular functionality. Useful for filtering.

For a U3/U6 with firmware less than 2.0, or for the UE9, see the deprecated Modbus system called [UD Modbus](#).

Device: All Devices

All Tags
 AIN
 AIN_EF
 ASYNCH
 CONFIG
 CORE

Tags:

Expand addresses:

Show 10 entries		Search:				
name	address	type	access	tags	details	
AIN#(0:254)	0	FLOAT32	R	AIN, CORE		
DAC#(0:1)	1000	FLOAT32	R / W	DAC, CORE		
CURRENT_SOURCE_10UA_CAL_VALUE	1900	FLOAT32	R	CONFIG		
CURRENT_SOURCE_200UA_CAL_VALUE	1902	FLOAT32	R	CONFIG		
FIO#(0:7)	2000	UINT16	R / W	DIO, CORE		
DIO#(0:7)	2000	UINT16	R / W	DIO, CORE		
EIO#(0:7)	2008	UINT16	R / W	DIO, CORE		
DIO#(8:15)	2008	UINT16	R / W	DIO, CORE		
CIO#(0:3)	2016	UINT16	R / W	DIO, CORE		
DIO#(16:19)	2016	UINT16	R / W	DIO, CORE		

Showing 1 to 10 of 342 entries

2 3 4 5 Next Last

3.2 Stream Mode

Streaming is for applications where data throughput above 1000 scans/seconds is required. Command-response is generally used for 1000Hz and slower because it is simpler than stream mode. Functions use command-response unless the function name directly mentions stream, or streaming. The following list of registers is used to perform streaming with the T7.

Stream Configuration

Name	Start Address	Type	Access	Default
STREAM_SCANRATE_HZ	4002	FLOAT32	R/W	0
STREAM_NUM_ADDRESSES	4004	UINT32	R/W	0
STREAM_SAMPLES_PER_PACKET	4006	UINT32	R/W	0
STREAM_SETTLING_US	4008	FLOAT32	R/W	0
STREAM_RESOLUTION_INDEX	4010	UINT32	R/W	0
STREAM_BUFFER_SIZE_BYTES	4012	UINT32	R/W	0
STREAM_AUTO_TARGET	4016	UINT32	R/W	0
STREAM_NUM_SCANS	4020	UINT32	R/W	0
STREAM_ENABLE	4990	UINT32	W	
STREAM_SCANLIST_ADDRESS#(0:127)	4100	UINT32	R/W	0

STREAM_SCANRATE_HZ

The number of times per second that all channels in the scanlist will be read.

STREAM_NUM_ADDRESSES

The number of entries in the scanlist

STREAM_SAMPLES_PER_PACKET

Specifies the number of data points to be sent in the data packet.

STREAM_SETTLING_US

Time in microseconds to allow signals to settle after switching the mux. Default 10 us.

STREAM_RESOLUTION_INDEX

Index specifying the resolution of the data. High settings will have lower max speeds.

STREAM_BUFFER_SIZE_BYTES

Size of the stream data buffer in bytes. Must be a power of 2. 0 is updated to the default size upon stream starting. Changes while stream is running do not affect the currently running stream.

STREAM_AUTO_TARGET

Controls where data will be sent.

STREAM_NUM_SCANS

The number of scans to run before automatically stopping.

STREAM_ENABLE

Write 1 to start stream. Write 0 to stop stream.

STREAM_SCANLIST_ADDRESS#(0:127)

A list of addresses to read each scan.

Names	Addresses
STREAM_SCANLIST_ADDRESS0,	4100, 4102, 4104, Show All
STREAM_SCANLIST_ADDRESS1,	
STREAM_SCANLIST_ADDRESS2, Show All	

```
$( document ).ready(function() { $('<div data-bbox="47 287 371 312" data-label="Section-Header">
<h2>Stream-Out (Advanced)</h2>
</div>
<div data-bbox="42 326 99 343" data-label="Section-Header">
<h3>Usage</h3>
</div>
<div data-bbox="42 353 959 408" data-label="Text">
<p>Along with the standard stream mode (stream-in), the T7 also has the ability to output a data stream. Users create an array of values to output to an output capable I/O, such as DAC or Digital I/O, and the T7 will output the values at up to 100kHz. Digital waveform generation is typically handled by the extended features of the Digital I/O system, but it's possible to create a very unique waveform using stream-out. Stream-out is best suited to create fast (>1000Hz) analog output signals on the DAC output lines.</p>
</div>
<div data-bbox="42 420 125 436" data-label="Section-Header">
<h3>Summary</h3>
</div>
<div data-bbox="42 447 935 475" data-label="Text">
<p>Stream-out is a ring buffer from which data is read and sent to a target. Upon reaching the end of the supplied data, stream-out has three loop triggers.</p>
</div>
<div data-bbox="42 484 487 500" data-label="Text">
<p>End of data loop triggers (set by STREAM_OUT#(0:3)_SET_LOOP):</p>
</div>
<div data-bbox="65 508 301 549" data-label="List-Group">
<ul>
<li>• 1: Use new data set and loop size</li>
<li>• 2: Wait for synch</li>
<li>• 3: Synch</li>
</ul>
</div>
<div data-bbox="42 559 510 574" data-label="Text">
<p>(A more detailed descriptions of these data loop triggers is yet to come.)</p>
</div>
<div data-bbox="42 582 949 636" data-label="Text">
<p>The T7 firmware handles up to four stream-out data streams, each of which has its own buffer and target. Target refers to the physical I/O line that will be updated with the values in the data stream. A stream-out channel updates its target when the channel is called in the scan list, so the order of outputs and inputs during a scan is completely controllable. In other words, it's possible to mix stream-in and stream-out channels.</p>
</div>
<div data-bbox="42 645 943 673" data-label="Text">
<p>Note that the register list below only shows those registers specifically related to the stream-out subsystem. For a complete list of registers required for operation, combine this list with those in the Stream Mode section.</p>
</div>
<div data-bbox="42 683 200 698" data-label="Section-Header">
<h3>Stream-Out Registers</h3>
</div>
<div data-bbox="42 696 956 843" data-label="Table">
<table border="1">
<thead>
<tr>
<th>Name</th>
<th>Start Address</th>
<th>Type</th>
<th>Access</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM_OUT#(0:3)_TARGET</td>
<td>4040</td>
<td>UINT32</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_BUFFER_SIZE</td>
<td>4050</td>
<td>UINT32</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_LOOP_SIZE</td>
<td>4060</td>
<td>UINT32</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_SET_LOOP</td>
<td>4070</td>
<td>UINT32</td>
<td>W</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_BUFFER_STATUS</td>
<td>4080</td>
<td>UINT32</td>
<td>R</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_ENABLE</td>
<td>4090</td>
<td>UINT32</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_BUFFER_F32</td>
<td>4400</td>
<td>FLOAT32</td>
<td>W</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_BUFFER_U32</td>
<td>4410</td>
<td>UINT32</td>
<td>W</td>
<td>0</td>
</tr>
<tr>
<td>STREAM_OUT#(0:3)_BUFFER_U16</td>
<td>4420</td>
<td>UINT16</td>
<td>W</td>
<td>0</td>
</tr>
</tbody>
</table>
</div>
<div data-bbox="47 852 231 866" data-label="Section-Header">
<h3>STREAM_OUT#(0:3)_TARGET</h3>
</div>
<div data-bbox="47 865 249 879" data-label="Text">
<p>Channel that data will be written to.</p>
</div>
<div data-bbox="47 881 534 920" data-label="Table">
<table border="1">
<thead>
<tr>
<th>Names</th>
<th>Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM_OUT0_TARGET, STREAM_OUT1_TARGET,</td>
<td>4040, 4042, 4044, <a href="#">Show All</a></td>
</tr>
<tr>
<td>STREAM_OUT2_TARGET, <a href="#">Show All</a></td>
<td></td>
</tr>
</tbody>
</table>
</div>
<div data-bbox="47 930 267 945" data-label="Section-Header">
<h3>STREAM_OUT#(0:3)_BUFFER_SIZE</h3>
</div>
</pre>
</div>
<div data-bbox="0 981 34 1000" data-label="Page-Footer">of 81</div>
<div data-bbox="888 981 1000 1000" data-label="Page-Footer">4/8/2014 1:27 P</div>
```

Stream-Out (Advanced)

Usage

Along with the standard stream mode (stream-in), the T7 also has the ability to output a data stream. Users create an array of values to output to an output capable I/O, such as DAC or Digital I/O, and the T7 will output the values at up to 100kHz. Digital waveform generation is typically handled by the extended features of the Digital I/O system, but it's possible to create a very unique waveform using stream-out. Stream-out is best suited to create fast (>1000Hz) analog output signals on the DAC output lines.

Summary

Stream-out is a ring buffer from which data is read and sent to a target. Upon reaching the end of the supplied data, stream-out has three loop triggers.

End of data loop triggers (set by STREAM_OUT#(0:3)_SET_LOOP):

- 1: Use new data set and loop size
- 2: Wait for synch
- 3: Synch

(A more detailed descriptions of these data loop triggers is yet to come.)

The T7 firmware handles up to four stream-out data streams, each of which has its own buffer and target. Target refers to the physical I/O line that will be updated with the values in the data stream. A stream-out channel updates its target when the channel is called in the scan list, so the order of outputs and inputs during a scan is completely controllable. In other words, it's possible to mix stream-in and stream-out channels.

Note that the register list below only shows those registers specifically related to the stream-out subsystem. For a complete list of registers required for operation, combine this list with those in the Stream Mode section.

Stream-Out Registers

Name	Start Address	Type	Access	Default
STREAM_OUT#(0:3)_TARGET	4040	UINT32	R/W	0
STREAM_OUT#(0:3)_BUFFER_SIZE	4050	UINT32	R/W	0
STREAM_OUT#(0:3)_LOOP_SIZE	4060	UINT32	R/W	0
STREAM_OUT#(0:3)_SET_LOOP	4070	UINT32	W	0
STREAM_OUT#(0:3)_BUFFER_STATUS	4080	UINT32	R	0
STREAM_OUT#(0:3)_ENABLE	4090	UINT32	R/W	0
STREAM_OUT#(0:3)_BUFFER_F32	4400	FLOAT32	W	0
STREAM_OUT#(0:3)_BUFFER_U32	4410	UINT32	W	0
STREAM_OUT#(0:3)_BUFFER_U16	4420	UINT16	W	0

STREAM_OUT#(0:3)_TARGET

Channel that data will be written to.

Names	Addresses
STREAM_OUT0_TARGET, STREAM_OUT1_TARGET, STREAM_OUT2_TARGET, Show All	4040, 4042, 4044, Show All

STREAM_OUT#(0:3)_BUFFER_SIZE

Size of the buffer in bytes. Should be at least twice the size of updates that will be written.

Names	Addresses
STREAM_OUT0_BUFFER_SIZE,	4050, 4052, 4054, Show All
STREAM_OUT1_BUFFER_SIZE,	
STREAM_OUT2_BUFFER_SIZE, Show All	

STREAM_OUT#(0:3)_LOOP_SIZE

The number of value that will be repeated after reaching the end of supplied data.

Names	Addresses
STREAM_OUT0_LOOP_SIZE,	4060, 4062, 4064, Show All
STREAM_OUT1_LOOP_SIZE,	
STREAM_OUT2_LOOP_SIZE, Show All	

STREAM_OUT#(0:3)_SET_LOOP

Controls when new data and loop size are used.

Names	Addresses
STREAM_OUT0_SET_LOOP,	4070, 4072, 4074, Show All
STREAM_OUT1_SET_LOOP,	
STREAM_OUT2_SET_LOOP, Show All	

STREAM_OUT#(0:3)_BUFFER_STATUS

The number of entries in the buffer that are not currently being used.

Names	Addresses
STREAM_OUT0_BUFFER_STATUS,	4080, 4082, 4084, Show All
STREAM_OUT1_BUFFER_STATUS,	
STREAM_OUT2_BUFFER_STATUS, Show All	

STREAM_OUT#(0:3)_ENABLE

Write 1 to enable, 0 to disable.

Names	Addresses
STREAM_OUT0_ENABLE, STREAM_OUT1_ENABLE,	4090, 4092, 4094, Show All
STREAM_OUT2_ENABLE, Show All	

STREAM_OUT#(0:3)_BUFFER_F32

Data destination when sending floating point data.

Names	Addresses
STREAM_OUT0_BUFFER_F32,	4400, 4402, 4404, Show All
STREAM_OUT1_BUFFER_F32,	
STREAM_OUT2_BUFFER_F32, Show All	

STREAM_OUT#(0:3)_BUFFER_U32

Data destination when sending 32-bit integer data.

Names	Addresses
STREAM_OUT0_BUFFER_U32,	4410, 4412, 4414, Show All
STREAM_OUT1_BUFFER_U32,	
STREAM_OUT2_BUFFER_U32, Show All	

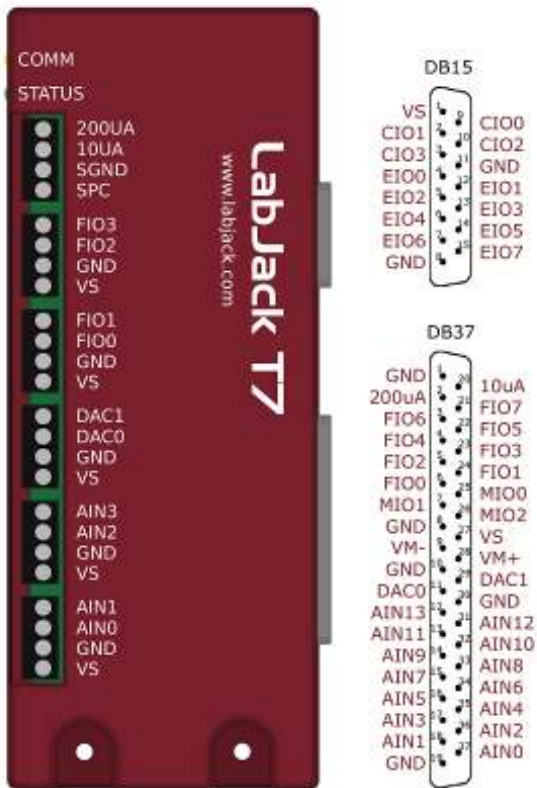
STREAM_OUT#(0:3)_BUFFER_U16

Data destination when sending 16-bit integer data.

Names	Addresses
STREAM_OUT0_BUFFER_U16,	4420, 4421, 4422, Show All
STREAM_OUT1_BUFFER_U16,	
STREAM_OUT2_BUFFER_U16, Show All	

```
$( document ).ready(function() { $(''.collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $(''.collapsed-content-expander').click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });
```

4.0 Hardware Overview



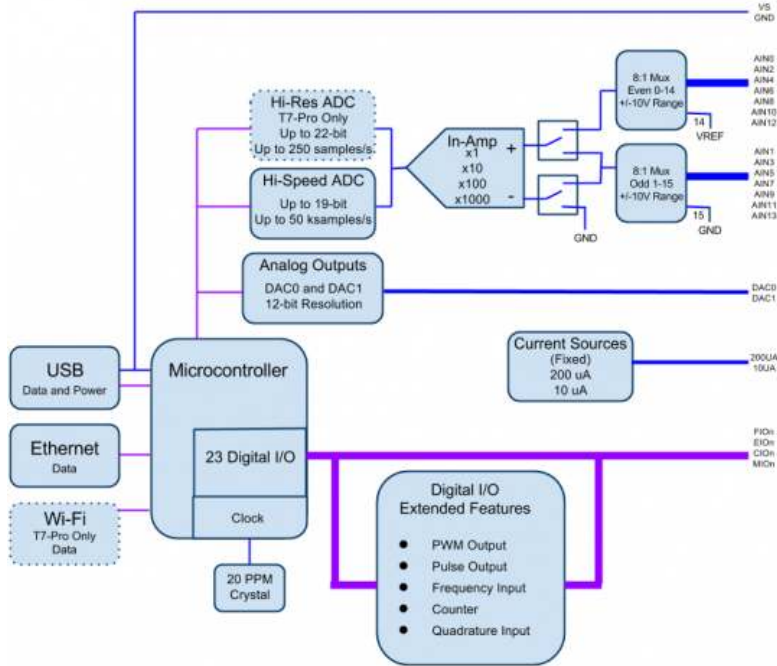
The T7 has 3 different I/O areas:

- Communication Edge
- Screw Terminal Edge
- DB Edge

The communication edge has a USB type B connector, an RJ45 Ethernet connector, and in the case of the T7-Pro also has a SMA-RP Female Connector and a WiFi antenna. Power is always provided through the USB connector, even if USB communication is not used.

The screw terminal edge has convenient connections for 4 analog inputs, both analog outputs, 4 digital I/O, and both current sources. The screw terminals are arranged in blocks of 4, with each block consisting of VS, GND, and two I/O. Also on this edge are two LEDs. The Comm LED generally blinks with communication traffic, while the Status LED is used for other indications.

The DB Edge has 2 D-sub type connectors: a DB15 and DB37. The DB15 has 12 additional digital I/O. The DB37 has the same I/O as the screw-terminals, plus additional analog inputs and digital I/O, for a total of 14 analog inputs, 2 analog outputs, 2 fixed current sources, and 11 digital I/O.



Digital waveforms can be output/input on various digital I/O lines, using extended features.

General Device Information

Name	Start Address	Type	Access	Default
PRODUCT_ID	60000	FLOAT32	R	
HARDWARE_VERSION	60002	FLOAT32	R	
FIRMWARE_VERSION	60004	FLOAT32	R	
BOOTLOADER_VERSION	60006	FLOAT32	R	
WIFI_VERSION	60008	FLOAT32	R	
HARDWARE_INSTALLED	60010	UINT32	R	0
ETHERNET_MAC	60020	UINT64	R	
WIFI_MAC	60024	UINT64	R	
SERIAL_NUMBER	60028	UINT32	R	
DEVICE_NAME_DEFAULT	60500	STRING	R/W	

PRODUCT_ID

The numeric identifier of the device. Such as 3 for a U3-HV.

HARDWARE_VERSION

The hardware version of the device.

FIRMWARE_VERSION

The current firmware version installed on the main processor.

BOOTLOADER_VERSION

The bootloader version installed on the main processor.

WIFI_VERSION

The current firmware version of the WiFi module, if available.

HARDWARE_INSTALLED

Bitmask indicating installed hardware options. 0 = High-Res ADC, 1 = WiFi.

ETHERNET_MAC

The MAC address of the wired Ethernet module.

WIFI_MAC

The MAC address of the WiFi module.

SERIAL_NUMBER

The serial number of the device.

DEVICE_NAME_DEFAULT

The current device name. Up to 49 characters, cannot contain periods.

There are several other useful device control registers listed below. These registers have device-wide impact.

Miscellaneous Device Registers

Name	Start Address	Type	Access	Default
CORE_TIMER	61520	UINT32	R	
SYSTEM_REBOOT	61998	UINT32	W	
WAIT_US_BLOCKING	61590	UINT32	R/W	

CORE_TIMER

Internal 32-bit system timer running at 1/2 core speed, thus normally 80M/2 => 40 MHz.

SYSTEM_REBOOT

Issues a device reboot. Must write 0x4C4Axxxx, where xxxx is number of 50ms ticks to wait before reboot.

WAIT_US_BLOCKING

Delays for x microseconds. Range is 0-100000.

5.0 USB

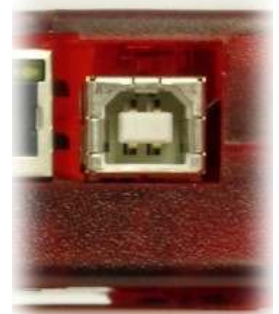
Communication Protocol: **Modbus TCP**

Connector Type: **USB-B Receptacle**

Compatible: **USB 1.1+**

Max Cable Length: **5 meters**

Max Packet Size: **64 bytes/packet**



Power is supplied to the T7 through the 5V USB connection. If the Ethernet or Wi-Fi connection is preferred for communication, use the provided AC USB 5V adapter for power. When used for communication, it is a full-speed USB connection compatible with USB version 1.1 or higher.

Interface - Talk to the T7

Modbus TCP is the protocol used by all connections on the T7 (USB, Ethernet, WiFi). If you want to handle USB communication yourself (find/open/write/read/close), you can use the Modbus protocol. Most customers, however, will use our [LJM library](#) which provides convenient device discovery, high-level functions, and programming flexibility.

Power Considerations

USB ground is connected to the T7 ground (GND), and USB ground is generally the same as the ground of the PC chassis and AC mains, since standard USB is non-isolated.

It is possible to isolate USB, and thereby protect the T7 from a power surge coming through the computer, if you use a USB isolator. USB isolators typically go for \$40 to \$100 USD, depending on the capabilities.

The T7-Pro will generally require a powered USB hub when in operating at full-power, since some USB ports/hubs will not supply the current necessary (500mA). Our experience with cheap USB supplies has shown them to be unreliable above 200mA. We recommend a powered USB hub rated for battery charging applications, since these are typically rated for 1-2A. See electrical specifications for details on USB current requirements.



If designing your own driver...

The LabJack vendor ID is 0x0CD5. The product ID for the T7 is 0x0007.

The USB interface consists of the normal bidirectional control endpoint (0 OUT & IN), 3 used bulk endpoints (1 OUT, 2 IN, 3 IN), and 1 dummy endpoint (3 OUT). Endpoint 1 consists of a 64 byte OUT endpoint (address = 0x01). Endpoint 2 consists of a 64 byte IN endpoint (address = 0x82). Endpoint 3 consists of a dummy OUT endpoint (address = 0x03) and a 64 byte IN endpoint (address = 0x83). Endpoint 3 OUT is not supported by the firmware, and should never be used.

All commands should always be sent on Endpoint 1, and the responses to commands will always be on Endpoint 2. Endpoint 3 is only used to send stream data from the T7 to the host.

6.0 Ethernet

Communication Protocol: **Modbus TCP**



Connector Type: **RJ-45 Socket, Cat 5**

Compatible: **10/100Base-T**

POE Compatible: **No^[1]**

Max Cable Length: **100 meters typical**

Max Packet Size: **1040 bytes/packet (TCP), 64 bytes/packet (UDP)**



Overview

The T7 has a 10/100Base-T Ethernet connection. This connection only provides communication, so power must be provided through the USB connector. Refer to this [WiFi and Ethernet tutorial](#) to get started.

Config (_DEFAULT) Registers versus Status Registers

The first list below is the config registers. These are generally written to configure default Ethernet settings, but can also be read. Configure the Ethernet parameters in [Kipling software](#).

Any register with _DEFAULT at the end is non-volatile. Whatever value you write to a _DEFAULT register will be retained through a reboot or power-cycle.

The second list below consists of read-only registers to read the status of Ethernet.

For example, if you write `ETHERNET_IP_DEFAULT="192.168.1.207"` (you actually write/read the 32-bit numeric equivalent not an IP string), then that value will be retained through reboots and is the default IP address. If DHCP is disabled, this will be the static IP of the device and what you get if you read `ETHERNET_IP`. If DHCP is enabled, then a read of `ETHERNET_IP` will return the IP set by the DHCP server.

Ethernet Config Registers

Name	Start Address	Type	Access	Default
ETHERNET_IP_DEFAULT	49150	UINT32	R/W	
ETHERNET_SUBNET_DEFAULT	49152	UINT32	R/W	
ETHERNET_GATEWAY_DEFAULT	49154	UINT32	R/W	
ETHERNET_DNS_DEFAULT	49156	UINT32	R/W	
ETHERNET_ALTDNS_DEFAULT	49158	UINT32	R/W	
ETHERNET_DHCP_ENABLE_DEFAULT	49160	UINT16	R/W	

ETHERNET_IP_DEFAULT

The IP address of wired Ethernet after a power-cycle to the device.

ETHERNET_SUBNET_DEFAULT

The subnet of wired Ethernet after a power-cycle to the device.

ETHERNET_GATEWAY_DEFAULT

The gateway of wired Ethernet after a power-cycle to the device.

ETHERNET_DNS_DEFAULT

The DNS of wired Ethernet after a power-cycle to the device.

ETHERNET_ALTDNS_DEFAULT

The Alt DNS of wired Ethernet after a power-cycle to the device.

ETHERNET_DHCP_ENABLE_DEFAULT

The Enabled/Disabled state of Ethernet DHCP after a power-cycle to the device.

Ethernet Status Registers

Name	Start Address	Type	Access	Default
ETHERNET_IP	49100	UINT32	R	
ETHERNET_SUBNET	49102	UINT32	R	
ETHERNET_GATEWAY	49104	UINT32	R	
ETHERNET_DNS	49106	UINT32	R	
ETHERNET_ALTDNS	49108	UINT32	R	
ETHERNET_DHCP_ENABLE	49110	UINT16	R	

ETHERNET_IP

Read the current IP address of wired Ethernet.

ETHERNET_SUBNET

Read the current subnet of wired Ethernet.

ETHERNET_GATEWAY

Read the current gateway of wired Ethernet.

ETHERNET_DNS

Read the current DNS of wired Ethernet.

ETHERNET_ALTDNS

Read the current Alt DNS of wired Ethernet.

ETHERNET_DHCP_ENABLE

Read the current Enabled/Disabled state of Ethernet DHCP.

Some Examples

Read IP Example: To read the wired IP Address of a device, perform a modbus read of address 49100. The value will be returned as an unsigned 32-bit number, such as 3232235691. Change this number to an IP address by converting each binary group to an octet, and adding decimal points as necessary. The result in this case would be "192.168.0.171".

Change IP Example: To change the Ethernet IP Address of a device, perform a modbus write to address 49150. The value must be passed as an unsigned 32-bit number, such as 3232235691. Change this IP address "192.168.0.171" by converting each octet to a binary group, and sticking them together.

More Details

Once default Ethernet configuration register(s) are changed, the current settings will be updated on the next power cycle. Alternatively, toggle power to the Ethernet module by writing a 0, then a 1 to the **POWER_ETHERNET** address.

Ethernet Power Settings

Name	Start Address	Type	Access	Default
POWER_ETHERNET	48003	UINT16	R/W	
POWER_ETHERNET_DEFAULT	48053	UINT16	R/W	

POWER_ETHERNET

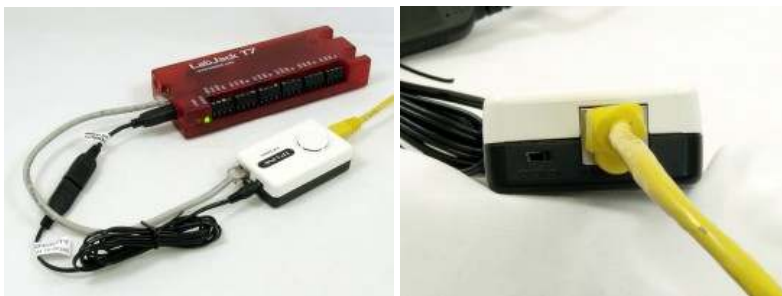
The current ON/OFF state of the Ethernet module. Provided to optionally reduce power consumption.

POWER_ETHERNET_DEFAULT

The ON/OFF state of the Ethernet module after a power-cycle to the device.

[1] The T7 cannot be directly powered via POE cable. However, it is relatively easy to find a POE splitter that converts 48V on POE to the 5V necessary for the T7. Such adapters run from ~\$30 to ~\$50 USD. Used in combination, the following parts work to split POE:

- [TP-LINK TL-POE10R](#) - To split 48V from the Ethernet cable into a 5.5mm OD, 2.1mm ID center positive barrel receptacle.
- [Tensility International Corp 10-00240](#) - To convert 5.5mm OD, 2.1mm ID center positive barrel connector to USB-A male plug.
- [Tensility International Corp 10-00648](#) - A Female to B Male USB cable. This will fit on the USB-A male plug (Tensility 10-00240), and insert into the T7.



7.0 WiFi

Communication Protocol: **Modbus TCP**

Connector Type: **Female RP-SMA**

Compatible: **802.11 b/g**



Range: **With stock antenna ... similar to laptops and other WiFi devices**

Max Packet Size: **500 bytes/packet**



Overview

The T7-Pro has a wireless module. Refer to this [WiFi and Ethernet tutorial](#) to get started.

DHCP is enabled from the factory, so to get WiFi going from the factory write the desired SSID string (case sensitive) to `WIFI_SSID_DEFAULT` and the proper password string (case sensitive) to `WIFI_PASSWORD_DEFAULT`. Then write a 1 to `WIFI_APPLY_SETTINGS` and watch the status codes. If you get back code 2900 the WiFi chip is associated to your network, and you can then read the assigned IP from `WIFI_IP`. Find more details and troubleshooting tips in the [WiFi and Ethernet tutorial](#).

Use the T7-Pro in the same way you would use a standard T7 over Ethernet, but with the WiFi IP address.

Config (_DEFAULT) Registers versus Status Registers

The first list below is the config registers. These are generally written to configure default WiFi settings, but can also be read (except password). Configure the WiFi parameters in [Kipling software](#).

Any register with `_DEFAULT` at the end is non-volatile. Whatever value you write to a `_DEFAULT` register will be retained through a reboot or power-cycle. WiFi is unique compared to other systems on the T7, in that new values written to `_DEFAULT` registers are not actually saved until a 1 is written to `WIFI_APPLY_SETTINGS`.

The second list below consists of read-only registers to read the status of WiFi.

For example, if you write `WIFI_IP_DEFAULT="192.168.1.208"` (you actually write/read the 32-bit numeric equivalent not an IP string), then that value will be retained through reboots and is the default IP address. If DHCP is disabled, this will be the static IP of the device and what you get if you read `WIFI_IP`. If DHCP is enabled, then a read of `WIFI_IP` will return the IP set by the DHCP server.

WiFi Config Registers

Name	Start Address	Type	Access	Default
<code>WIFI_IP_DEFAULT</code>	49250	UINT32	R/W	
<code>WIFI_SUBNET_DEFAULT</code>	49252	UINT32	R/W	
<code>WIFI_GATEWAY_DEFAULT</code>	49254	UINT32	R/W	
<code>WIFI_DHCP_ENABLE_DEFAULT</code>	49260	UINT16	R/W	
<code>WIFI_SSID_DEFAULT</code>	49325	STRING	R/W	
<code>WIFI_PASSWORD_DEFAULT</code>	49350	STRING	W	
<code>WIFI_APPLY_SETTINGS</code>	49400	UINT32	W	

WIFI_IP_DEFAULT

The new IP address of WiFi. Use `WIFI_APPLY_SETTINGS`.

WIFI_SUBNET_DEFAULT

The new subnet of WiFi. Use `WIFI_APPLY_SETTINGS`.

WIFI_GATEWAY_DEFAULT

The new gateway of WiFi. Use `WIFI_APPLY_SETTINGS`.

WIFI_DHCP_ENABLE_DEFAULT

The new Enabled/Disabled state of WiFi DHCP. Use `WIFI_APPLY_SETTINGS`.

WIFI_SSID_DEFAULT

The new SSID (network name) of WiFi. Use `WIFI_APPLY_SETTINGS`.

WIFI_PASSWORD_DEFAULT

Write the password for the WiFi network, then use `WIFI_APPLY_SETTINGS`.

WIFI_APPLY_SETTINGS

Apply all new WiFi settings: IP, Subnet, Gateway, DHCP, SSID, Password. 1=Apply

WiFi Status Registers

Name	Start Address	Type	Access	Default
<code>WIFI_IP</code>	49200	UINT32	R	
<code>WIFI_SUBNET</code>	49202	UINT32	R	
<code>WIFI_GATEWAY</code>	49204	UINT32	R	

Name	Start Address	Type	Access	Default
WIFI_DHCP_ENABLE	49210	UINT16	R	
WIFI_SSID	49300	STRING	R	
WIFI_STATUS	49450	UINT32	R	
WIFI_RSSI	49452	FLOAT32	R	

WIFI_IP

Read the current IP address of WiFi.

WIFI_SUBNET

Read the current subnet of WiFi.

WIFI_GATEWAY

Read the current gateway of WiFi.

WIFI_DHCP_ENABLE

Read the current Enabled/Disabled state of WiFi DHCP.

WIFI_SSID

Read the current SSID (network name) of WiFi

WIFI_STATUS

Status Codes.

WIFI_RSSI

WiFi RSSI (signal strength). Typical values are -40 for very good, and -75 for very weak. The T7 microcontroller only gets a new RSSI value from the WiFi module when WiFi communication occurs.

WiFi Power Registers

Name	Start Address	Type	Access	Default
POWER_WIFI	48004	UINT16	R/W	
POWER_WIFI_DEFAULT	48054	UINT16	R/W	

POWER_WIFI

The current ON/OFF state of the WiFi module. Provided to optionally reduce power consumption.

POWER_WIFI_DEFAULT

The ON/OFF state of the WiFi module after a power-cycle to the device.

Some Examples

Read IP Example: To read the wireless IP address of a device, perform a modbus read of address 49200. The value will be returned as an unsigned 32-bit number, such as 3232235691. Change this number to an IP address by converting each binary group to an octet, and adding decimal points as necessary. The result in this case would be "192.168.0.171"

Write IP Example: To change the Wireless IP Address of a device, perform a modbus write to address 49250. The IP address must be passed as an unsigned 32-bit number, such as 3232235691. Change this IP address "192.168.0.171" by converting each octet to a binary group, and sticking them together.

More Details

Once default wireless configuration register(s) are changed, it is necessary to also write 1 to the **WIFI_APPLY_SETTINGS** register. Alternatively, the default settings will be updated on the next power cycle.

Update WiFi Firmware

The WiFi chip on the T7 is a separate chip from the main processor, and it can be updated using the **WIFI_FIRMWARE_UPDATE_TO_VERSIONX** register. If connected to the internet, the WiFi chip can download new firmware files from an ftp server. To initiate a download and update, write a new firmware version to the **WIFI_FIRMWARE_UPDATE_TO_VERSIONX** register. *At the time of this writing we recommend using [Kipling to update WiFi firmware](#), since Kipling connects to the FTP server to identify what firmware is available, and monitors the **WIFI_FIRMWARE_UPDATE_STATUS** register automatically.*

WiFi Firmware Registers

Name	Start Address	Type	Access	Default
WIFI_VERSION	60008	FLOAT32	R	
WIFI_FIRMWARE_UPDATE_TO_VERSIONX	49402	FLOAT32	W	

Name	Start Address	Type	Access	Default
WIFI_FIRMWARE_UPDATE_STATUS	49454	UINT32	R	

WIFI_VERSION

The current firmware version of the WiFi module, if available.

WIFI_FIRMWARE_UPDATE_TO_VERSIONX

Start an update by using USB or Ethernet to write the desired version to update to.

WIFI_FIRMWARE_UPDATE_STATUS

See status codes in the constants array.

```
#define WIFI_ASSOCIATED           2900
#define WIFI_ASSOCIATING         2901
#define WIFI_ASSOCIATION_FAILED  2902
#define WIFI_UNPOWERED           2903
#define WIFI_BOOTING_UP          2904
#define WIFI_COULD_NOT_START     2905
#define WIFI_APPLYING_SETTINGS   2906
#define WIFI_DHCP_STARTED        2907
#define WIFI_OTHER                2909
```

WiFi Range

The WiFi range on the T7 is typical for a modern WiFi device. In direct line-of-sight with the router, it's possible to get a decent connection at 100m. The table below shows signal strength at varying distances with a stock T7 antenna, and a simple WiFi router. Both the T7 and the router were positioned 3ft off of the ground, with direct line-of-sight.

Distance	RSSI
10m	-44dBm
25m	-45dBm
50m	-55dBm
100m	-59dBm



During testing, it was noted that the T7 had slightly better WiFi range than an HTC One V cell phone. The WiFi signal is spotty at RSSI lower than -70dBm, and the connection will cut off entirely at -80dBm. Note that 90° antenna orientation was used in testing above. That is to say, keep the antenna in the fully bent upright position, don't try to point it at the router, or accidentally leave it at 45° bent. At 45° bent, or directly pointed towards the router, the signal strength is reduced by about 5dBm.

OEM Whip Antenna

The OEM whip antenna is a short segment of wire, only 30mm in length. This whip antenna provides an inexpensive solution for adding WiFi to an OEM board, without the need to figure out mounting of a bigger antenna. The signal strength of a 30mm whip antenna is on average 11dB less than that of the stock antenna. The table below demonstrates the 30mm antenna signal strength at various distances.

Distance	RSSI
10m	-49dBm
25m	-58dBm
50m	-70dBm
100m	-73dBm

**Improve signal strength**

The first step to improve the signal strength at large distances is to insure direct line-of-sight. Beyond that, the next best thing is to elevate the transmitter and receiver antennas. Even an elevation of 1m off the ground helps quite a bit. Be sure to consider the probability of lightning strikes if the antenna is high relative to the surroundings.

The next step to improve signal strength is to use a directional WiFi antenna. Directional antennas improve range substantially, such that even a homemade solution can increase range to fifteen times that of a non-directional antenna. If you need something to work at 500m, it's possible to buy a simple yagi antenna for \$60 USD approx.

**8.0 LEDs****STATUS - green LED**

The status LED is mainly reserved to indicate when Lua scripts are running. The LED will blink when the script does something.



The status LED also activates during firmware updates to indicate various stages of the process, refer to the Combined LED Activity section.

COMM - yellow LED

The primary indicator for packet transfer. If the T7 is communicating the COMM LED will be blinking. A few blinks after connecting to the PC indicates that the T7 is enumerating. Enumeration is when the standard USB initialization takes place, and the host is gathering device information.

The COMM LED will blink when the T7 receives Modbus commands, or when streaming data. Each packet will produce a single blink. If commands are issued rapidly, the LED will blink rapidly. At high packet transfer rates the LED will blink at 10Hz, even though more than 10 packets are being processed per second.

Combined LED Activity

When the LEDs blink together, the T7 is computing checksums.

When the LEDs are alternating, the T7 is copying a firmware image.



9.0 VS, Power Supply

Supply Voltage: **4.75 - 5.25 volts (5V ±5% Regulated)**

Typical Active Current: **250 mA**

Typical Sleep Current: **8 mA**

Normal Power Connector: **USB-B Receptacle**

Typical Power Supply: **Any USB-Style Supply**



VS Terminals

The VS terminals are designed as outputs for the supply voltage. The supply voltage is nominally 5 volts and typically provided through the USB connector.

All VS terminals are the same.

The VS connections are outputs, not inputs. Do not connect a power source to VS in normal situations.

The max total current that can be drawn from VS is 500mA - DeviceSupplyCurrent, so if the T7 needs 250mA to run, that leaves 250mA available from the VS terminals.

Power Supply

Power supply for the T7 is typically provided through the USB connector. For a different board-level connection option see "Alternate Power Supply" in the [OEM section](#). Typical power supply sources include:

- USB host or hub.
- Wall-wart power supply with USB connection (included with normal retail units ... not OEM).
- Power-over-Ethernet splitter (e.g. TP-Link TL-POE10R with Tensility 10-00240 with Tensility 10-00648).
- Car charger with USB ports (e.g. Anker 71AN2452C-WA).
- Rechargeable battery with USB ports (e.g. Anker Astro E5 79AN15K-BA perhaps with Belkin F3U133-06INCH).
- Battery with car charger (e.g. Anker 79AN15K-BA with 71AN2452C-02WA).

The supply range for specified operation is 4.75 to 5.25 volts, which is the same as the USB specification for voltage provided to a device. Nonetheless, we have seen some USB host ports providing a lower voltage. If your USB host port has this problem, add a USB hub with a strong power supply.

See related data in the General section of [Specifications](#).

Normal retail units (not OEM) include a 5V, 2A wall-wart style power supply:

Compatibility	Make	Mfr. Model No.
---------------	------	----------------

North America	VA-PSU-US1	JX-B0520B-1-B
Europe	VA-PSU-EU1	JX-B0520A-1-B
United Kingdom	VA-PSU-UK1	JX-B0520C-1-B
Australia	-	JX-B0520D-1-B

10.0 SGND and GND

SGND

SGND is located near the upper-left of the device. This terminal has a self-resetting thermal fuse in series with GND. This is often a good terminal to use when connecting the ground from another separately powered system that could unknowingly already share a common ground with the T7.

See the AIN, DAC, and Digital I/O (FIO, EIO, CIO, MIO) application notes for more information about grounding.



GND

The GND connections available at the screw-terminals and DB connectors provide a common ground for all LabJack functions. All GND terminals are the same and connect to the same ground plane.

GND is also connected to the ground pin on the USB connector, so if there is a connection to a USB port on a hub/host (as opposed to just a power supply connection), then GND is the same as the ground line on the USB connection, which is often the same as ground on the PC chassis, which is often the same as AC mains ground.

See the AIN, DAC, and Digital I/O (FIO, EIO, CIO, MIO) Sections for more information about grounding.

The max total current that can be sunk into GND is 500mA - DeviceSupplyCurrent, so if the T7 needs 250mA to run, the current sunk into GND terminals should be limited to 250mA. Note that sinking substantial current into GND can cause slight voltage differences between different ground terminals, which can cause noticeable errors with single-ended analog input readings.



11.0 SPC

Unless the T7 has problems SPC is not typically needed.

During startup the T7 will look for connections between digital I/O and the SPC terminal. The following list describes what will happen when a jumper wire is placed between SPC and a listed IO.

SPC wired to:

- **FIO0**: Force boot to main firmware (internal) image.
- **FIO1**: Force copy of backup image to overwrite internal image.
- **FIO2**: Factory reset.
- **FIO3**: Load emergency image. This option loads a firmware image with minimal functionality (kinda like Windows safe-mode). The update process is about all that can be done while in this mode.



12.0 200uA and 10uA

Overview

The T7 has 2 fixed current source terminals useful for measuring resistance (thermistors, RTDs, resistors). The 10UA terminal provides about 10 μ A and the 200UA terminal provides about 200 μ A.

The actual value of each current source is noted during factory calibration and stored with the calibration constants on the device. These can be viewed using the Kipling software, or read programmatically. Note that these are fixed constants stored during calibration, not some sort of current readings.



Using the equation $V=IR$, with a known current and voltage, it is possible to calculate the resistance of the item in question. Figure 2.5-1 shows a simple setup measuring 1 resistor.

Constant Current Sources

Name	Start Address	Type	Access	Default
CURRENT_SOURCE_200UA_CAL_VALUE	1902	FLOAT32	R	
CURRENT_SOURCE_10UA_CAL_VALUE	1900	FLOAT32	R	

CURRENT_SOURCE_200UA_CAL_VALUE

CURRENT_SOURCE_10UA_CAL_VALUE

For example: To read the actual value of the 200uA current source, perform a read of Modbus address 1902, and the result would be in the form of a floating point number, e.g. 0.000197456 amps.

Some Examples

Multiple resistances can be measured by putting them in series and measuring the voltage across each. Some applications might need to use differential inputs to measure the voltage across each resistor, but for many applications it works just as well to measure the single-ended voltage at the top of each resistor and subtract in software.

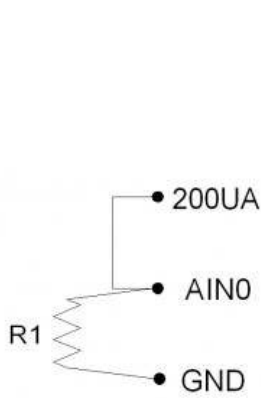


Figure 2.5-1

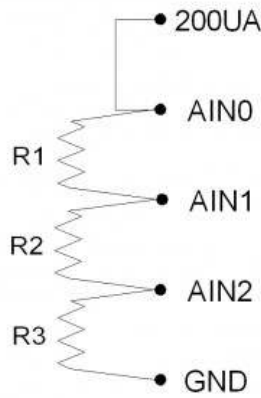


Figure 2.5-2

Figure 2.5-1 shows a simple setup measuring 1 resistor. If $R1=3k$, the voltage at AIN0 will be 0.6 volts.

Figure 2.5-2 shows a setup to measure 3 resistors using single-ended analog inputs. If $R1=R2=R3=3k$, the voltages at AIN0/AIN1/AIN2 will be 1.8/1.2/0.6 volts. That means AIN0 and AIN1 would be measured with the +/-10 volt range, while AIN2 could be measured with the +/-1 volt range. This points out a potential advantage to differential measurements, as the differential voltage across R1 and R2 could be measured with the +/-1 volt range, providing better resolution.

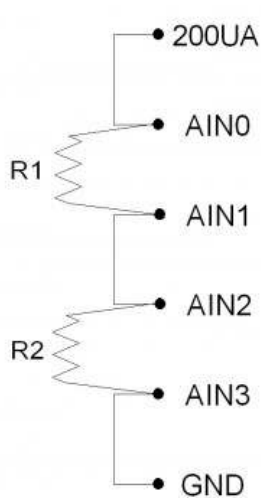


Figure 2.5-3

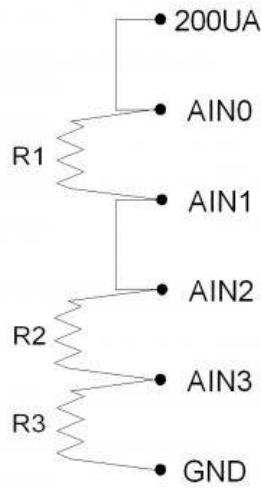


Figure 2.5-4

Figure 2.5-3 shows a setup to measure 2 resistors using differential analog inputs. AIN3 is wasted in this case, as it is connected to ground, so a differential measurement of AIN2-AIN3 is the same as a single-ended measurement of AIN2. That leads to Figure 2.5-4, which shows

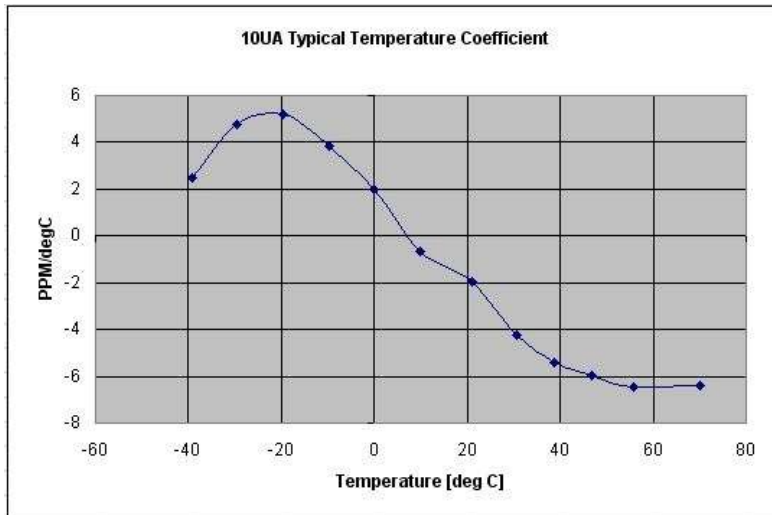
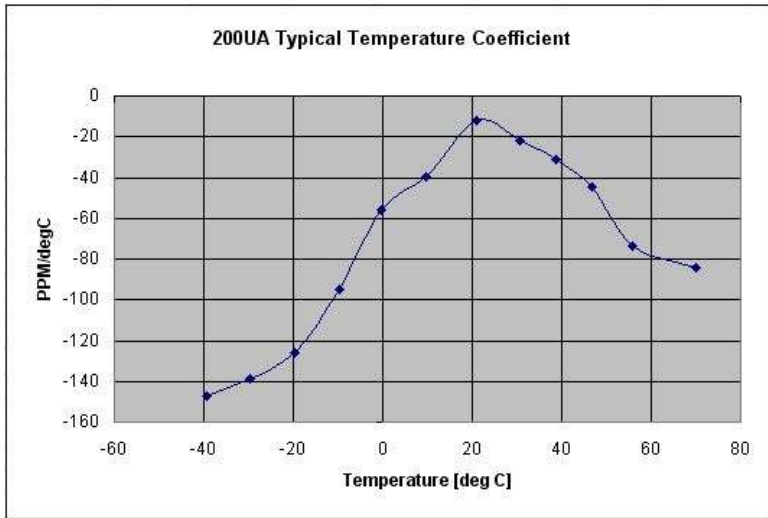
R1 and R2 measured differentially and R3 measured single-ended.

Specifications

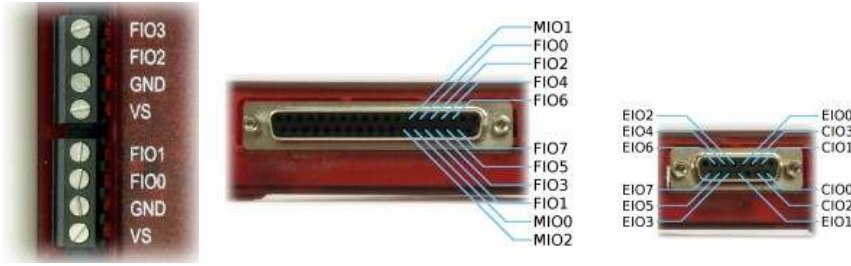
The current sources can drive about 3 volts max, thus limiting the maximum load resistance to about 300 k Ω (10 μ A) and 15 k Ω (200 μ A). Keep in mind that high source resistance could cause settling issues for analog inputs.

The current sources have good accuracy and tempco, but for improvement a fixed resistor can be used as one of the resistors in the figures below. The Y1453-100 and Y1453-1.0K from Digikey have excellent accuracy and very low tempco. By measuring the voltage across one of these you can calculate the actual current at any time.

The following charts show the typical tempco of the current sources over temperature. The 10 μ A current source has a very low tempco across temperature. The 200 μ A current source has a good tempco from about 0-50 degrees C, and outside of that range the effect of tempco will be more noticeable.



13.0 Digital I/O



Digital I/O: **23**

Logic Level: **3.3V**

DIO is a generic name used for all digital I/O. The DIO are subdivided into different ports called FIO, EIO, CIO, and MIO.

	FIO (0-7)							EIO (0-7)							CIO (0-3)			MIO (0-2)					
DIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
DIO Mapping																							

There are 4 types of registers used for digital I/O interaction: Simple DIO#, DIO State, DIO Direction, and DIO Inhibit

Simple Digital I/O#

Read or set the state of 1 digital I/O. Automatically configures the direction to input (when reading) or to output (when writing).

FIO0-FIO7 = DIO0-DIO7
 EIO0-EIO7 = DIO8-DIO15
 CIO0-CIO3 = DIO16-DIO19
 MIO0-MIO2 = DIO20-DIO22

Digital I/O

Name	Start Address	Type	Access	Default
FIO#(0:7)	2000	UINT16	R/W	
EIO#(0:7)	2008	UINT16	R/W	
CIO#(0:3)	2016	UINT16	R/W	
MIO#(0:2)	2020	UINT16	R/W	

FIO#(0:7)

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
FIO0, FIO1, FIO2, Show All	2000, 2001, 2002, Show All

EIO#(0:7)

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
EIO0, EIO1, EIO2, Show All	2008, 2009, 2010, Show All

CIO#(0:3)

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
CIO0, CIO1, CIO2, Show All	2016, 2017, 2018, Show All

MIO#(0:2)

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names	Addresses
MIO0, MIO1, MIO2	2020, 2021, 2022

```
$( document ).ready(function() {
    $('<div>
    </div>
```

Digital I/O State Bit Masks

Each of these is a single binary-encoded value representing the state of 8 bits of I/O. Each bit represents an I/O line. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of these values are inhibits. The inhibit bits prevent the corresponding state bit from being modified.

State Bit Masks

Name	Start Address	Type	Access	Default
FIO_STATE	2500	UINT16	R/W	
EIO_STATE	2501	UINT16	R/W	
CIO_STATE	2502	UINT16	R/W	
MIO_STATE	2503	UINT16	R/W	

FIO_STATE

Read or write the state of the 8 bits of FIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

EIO_STATE

Read or write the state of the 8 bits of EIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

CIO_STATE

Read or write the state of the 4 bits of CIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

MIO_STATE

Read or write the state of the 3 bits of MIO in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. The upper 8-bits of this value are inhibits.

For example: To read the digital state of all FIO lines in a bit mask, read FIO_STATE. The value will be something like 0b11111011 representing 1 for logic high, and 0 for logic low. FIO2 is currently logic low.

Digital I/O Direction Bit Masks

Each of these is a single binary-encoded value representing the direction of 8 bits of I/O. Each bit designates an I/O line. 0=Input and 1=Output. The upper 8-bits of this value are inhibits. The inhibit bits prevent the corresponding direction bit from being modified.

Direction Bit Masks

Name	Start Address	Type	Access	Default
FIO_DIRECTION	2600	UINT16	R/W	
EIO_DIRECTION	2601	UINT16	R/W	
CIO_DIRECTION	2602	UINT16	R/W	
MIO_DIRECTION	2603	UINT16	R/W	

FIO_DIRECTION

Read or write the direction of the 8 bits of FIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

EIO_DIRECTION

Read or write the direction of the 8 bits of EIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

CIO_DIRECTION

Read or write the direction of the 4 bits of CIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

MIO_DIRECTION

Read or write the direction of the 3 bits of MIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

For example: To set FIO1-7 to output, write a value of 0x01FF to FIO_DIRECTION. FIO0 is the least significant bit, so to prevent modification the corresponding inhibit bit is set with 0x01 in the most significant byte. The least significant byte is 0xFF, which is all 8 bits of FIO set to output.

Alternative DIO Registers

The DIO registers are the same thing as EIO, FIO, CIO, and MIO but with a more intuitive naming scheme, and a more compact register allotment.

DIO Registers

Name	Start Address	Type	Access	Default
DIO#(0:22)	2000	UINT16	R/W	
DIO_STATE	2800	UINT32	R/W	
DIO_DIRECTION	2850	UINT32	R/W	
DIO_INHIBIT	2900	UINT32	R/W	

DIO#(0:22)

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output.

Names

DIO0, DIO1, DIO2, [Show All](#)

Addresses

2000, 2001, 2002, [Show All](#)

DIO_STATE

Read or write the state of all digital I/O in a single binary-encoded value. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. Writes only apply to bits with mask set.

DIO_DIRECTION

Read or write the direction of all digital I/O in a single binary-encoded value. 0=Input and 1=Output. Writes only apply to bits with mask set.

DIO_INHIBIT

A single binary-encoded value where each bit determines whether `_STATE` or `_DIRECTION` writes affect that bit of digital I/O. 0=Default=Affected, 1=Ignored.

```
$( document ).ready(function() { $('<code>'.collapsed-content-expander</code>').closest('<code>'.content</code>').find('<code>'.sometimes-shown</code>').hide(); $('<code>'.collapsed-content-expander</code>').click(function(e) { $(e.target).closest('<code>'.content</code>').find('<code>'.collapsed-content-expander</code>').fadeOut(function () { $(e.target).closest('<code>'.content</code>').find('<code>'.sometimes-shown</code>').fadeIn(); }); return false; }); });
```

Electrical Overview

All digital I/O on the T7 have 3 possible states: input, output-high, or output-low. Each bit of I/O can be configured individually. When configured as an input, a bit has a ~100 kΩ pull-up resistor to 3.3 volts (all digital I/O are at least 5 volt tolerant). When configured as output-high, a bit is connected to the internal 3.3 volt supply (through a series resistor). When configured as output-low, a bit is connected to GND (through a series resistor).

See electrical specifications for more details.

By default, the DIO lines are digital I/O, but they can also be configured as PWM Output, Quadrature Input, Counters, etc (see Extended Feature section of this Datasheet).

FIO vs. EIO vs. CIO vs. MIO

DIO is a generic name used for all digital I/O. The DIO are subdivided into different groups called FIO, EIO, CIO, and MIO.

Sometimes these are referred to as different "ports". For example, FIO is an 8-bit port of digital I/O and EIO is a different 8-bit port of digital I/O. The different names (FIO vs. EIO vs. CIO vs. MIO) have little meaning, and generally you can call these all DIO0-DIO22 and consider them all the same. There are a couple details unique to different ports:

- The source impedance of an FIO line is about 550 ohms, whereas the source impedance of EIO/CIO/MIO lines is about 180 ohms. Source impedance might be important when sourcing or sinking substantial currents, such as when controlling relays.
- The MIO lines are automatically controlled when using analog input channel numbers from 16 to 127. This is for controlling external multiplexers or the [Mux80 expansion board](#).

Power-up Defaults

The default condition of the digital I/O can be configured by the user. From the factory, all digital I/O are configured as inputs by default. Note that even if the default for a line is changed to output-high or output-low, there could be a small time (milliseconds) during boot-up where all digital I/O are in the factory default condition.

Protection

All the digital I/O include an internal series resistor that provides overvoltage/short-circuit protection. These series resistors also limit the ability of these lines to sink or source current. Refer to the [Digital I/O Specifications](#).

The fact that the digital I/O are specified as 5-volt tolerant means that 5 volts can be connected to a digital input without problems (see the actual limits in the specifications in Appendix A).

Increase logic level to 5V

In some cases, an open-collector style output can be used to get a 5V signal. To get a low set the line to output-low, and to get a high set the line to input. When the line is set to input, the voltage on the line is determined by a pull-up resistor. The T7 has an internal ~100k resistor to 3.3V, but an external resistor can be added to a different voltage. Whether this will work depends on how much current the load is going to draw and what the required logic thresholds are. Say for example a 10k resistor is added from EIO0 to VS. EIO0 has an internal 100k pull-up to 3.3 volts and a series output resistance of about 180 ohms. Assume the load draws just a few microamps or less and thus is negligible. When EIO0 is set to input, there will be 100k to 3.3 volts in parallel with 10k to 5 volts, and thus the line will sit at about 4.85 volts. When the line is set to output-low, there will be 180 ohms in series with the 10k, so the line will be pulled down to about 0.1 volts.

The surefire way to get 5 volts from a digital output is to add a simple logic buffer IC that is powered by 5 volts and recognizes 3.3 volts as a high input. Consider the CD74ACT541E from TI (or the inverting CD74ACT540E). All that is needed is a few wires to bring VS, GND, and the signal from the LabJack to the chip. This chip can level shift up to eight 0/3.3 volt signals to 0/5 volt signals and provides high output drive current (+/-24 mA).

Note: DAC0, DAC1 channels on the T7 can be set to 5 volts, providing 2 output lines with such capability.

13.1 DIO Extended Features

Digital extended features measure and generate digital waveforms. Almost every digital I/O line can be assigned a feature and many can be active simultaneously. Features include things like PWM, Quadrature, and pulse generation. Features are assigned to DIOs using their type index, and configured using the options, and value registers.

The table below lists the features available on each DIO. The Digital I/O of the LabJack are on the top of the table, with the features to the left.

Feature	Index#	FIO (0-7)							EIO (0-7)							CIO (0-3)				
		DIO																		
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
PWM Out	0	✓		✓	✓	✓	✓													
PWM Out with Phase	1	✓		✓	✓	✓	✓													
Pulse Out	2	✓		✓	✓	✓	✓													
Frequency In	3,4	✓	✓																	
Pulse Width In	5	✓	✓																	
Line-to-Line In	6	✓	✓																	
High-Speed Counter	7																✓	✓	✓	✓
Interrupt Counter	8	✓	✓	✓	✓			✓	✓											
Interrupt Counter with Debounce	9	✓	✓	✓	✓			✓	✓											
Quadrature In	10	✓	✓	✓	✓			✓	✓											
Interrupt Frequency In	11	✓	✓	✓	✓			✓	✓											
Digital I/O Extended Features																				

- **PWM Out** - Produces a rectangular output with variable frequency and variable duty cycle.
- **PWM Out with Phase** - Allows a phase difference between multiple PWM outputs.
- **Pulse Out** - You can specify the number of pulses, frequency of pulses, and pulse-width.
- **Frequency In** - Measures the period/frequency.
- **Pulse Width In** - Measures the high and low time, and thus also measures duty cycle.
- **Line-to-Line In** - Measures the time between edges on 2 different DIO lines.
- **High-Speed Counter** - Hardware-based edge counter.
- **Interrupt Counter** - A hardware edge counter that must service an interrupt for each edge.
- **Interrupt Counter with Debounce** - Use to avoid counting bounces from mechanical switches.
- **Quadrature In** - Tracks the forward/reverse count provided by a quadrature signal.
- **Interrupt Frequency In** - Frequency measurement that must service an interrupt for each edge.

Each digital IO has a set of registers dedicated to the configuration of and results produced by the Extended Features. These registers are used to perform four operations on the Extended Feature: Configure, Read, Update, and Reset. Below you will find general descriptions of the four operations. Details about each feature can be found in their corresponding sections.

Configure:

Configuration is the initial setup of the Extended Feature. Configuration requires that any EF running at the pin in question first be disabled. Options can then be loaded. Then the EF can be enabled. The following seven registers are used for configuration:

DIO#_EF_ENABLE – 0 = Disable, 1 = Enable
 DIO#_EF_INDEX – Index number specifying the Extended Feature
 DIO#_EF_OPTIONS – Bits 2-0: Specifies the clock source to use
 DIO#_EF_VALUE_A – Extended Feature specific value
 DIO#_EF_VALUE_B – Extended Feature specific value
 DIO#_EF_VALUE_C – Extended Feature specific value
 DIO#_EF_VALUE_D – Extended Feature specific value

Read:

Some Extended Features produce results or provide status information that can be read. This information is usually a binary integer. When possible, the LabJack will convert the binary integer into a real-world unit such as seconds. When available, converted values can be read from the registers designated with “_F.” The following registers are used to read results from a DIO Extended Feature:

DIO#_EF_READ_A – Extended feature specific value. Reading this value takes a snapshot of READ_B, READ_B_F.
 DIO#_EF_READ_B – Extended feature specific value. Reading this returns the snapshot acquired by READ_A.
 DIO#_EF_READ_A_F – Returns READ_A converted to a real-world value and takes a snapshot of READ_B, READ_B_F.
 DIO#_EF_READ_B_F – Returns the READ_B snapshot converted to a real-world value.

Update:

Some Extended Features can be updated while running. Updating allows the Extended Feature to change its operation parameters without restarting. Note that the ClockSource and Feature Index can not be changed in an update. Depending on the feature, reads and writes to the update registers have small differences. See the Update portion of each features dedicated section for more information. The following four registers can be used to update an active Extended Feature:

DIO#_EF_VALUE_A – Extended feature specific value
 DIO#_EF_VALUE_B – Extended feature specific value
 DIO#_EF_VALUE_C – Extended feature specific value
 DIO#_EF_VALUE_D – Extended feature specific value

Reset:

Some Extended Features can be reset while they are running. Resetting can have different results depending on the feature. For instance counters are reset to zero. There is only one register associated with resetting:

DIO#_EF_READ_A_AND_RESET – Extended feature specific value. Reading this resets the Extended Feature and takes a snapshot of READ_B so that it can be read as in the Read section. Values are read before the reset.

DIO#_EF_READ_A_F_AND_RESET – Returns the same information as DIO#_EF_READ_A_F. Reading this resets the Extended Feature and takes a snapshot of READ_B_F. Values are read before the reset.

Digital Extended Features

Name	Start Address	Type	Access	Default
DIO#(0:22)_EF_ENABLE	44000	UINT32	R/W	
DIO#(0:22)_EF_INDEX	44100	UINT32	R/W	
DIO#(0:22)_EF_OPTIONS	44200	UINT32	R/W	
DIO#(0:22)_EF_VALUE_A	44300	UINT32	R/W	
DIO#(0:22)_EF_VALUE_B	44400	UINT32	R/W	
DIO#(0:22)_EF_VALUE_C	44500	UINT32	R/W	
DIO#(0:22)_EF_VALUE_D	44600	UINT32	R/W	
DIO#(0:22)_EF_READ_A	3000	UINT32	R	
DIO#(0:22)_EF_READ_A_AND_RESET	3100	UINT32	R	
DIO#(0:22)_EF_READ_B	3200	UINT32	R	

DIO#(0:22)_EF_ENABLE**Names**

DIO0_EF_ENABLE, DIO1_EF_ENABLE,
 DIO2_EF_ENABLE, [Show All](#)

Addresses

44000, 44002, 44004, [Show All](#)

DIO#(0:22)_EF_INDEX

An index to specify the feature you want.

Names

DIO0_EF_INDEX, DIO1_EF_INDEX,
 DIO2_EF_INDEX, [Show All](#)

Addresses

44100, 44102, 44104, [Show All](#)

DIO#(0:22)_EF_OPTIONS**Names**

DIO0_EF_OPTIONS, DIO1_EF_OPTIONS,
 DIO2_EF_OPTIONS, [Show All](#)

Addresses

44200, 44202, 44204, [Show All](#)

DIO#(0:22)_EF_VALUE_A**Names**

DIO0_EF_VALUE_A, DIO1_EF_VALUE_A,
 DIO2_EF_VALUE_A, [Show All](#)

Addresses

44300, 44302, 44304, [Show All](#)

DIO#(0:22)_EF_VALUE_B**Names**

DIO0_EF_VALUE_B, DIO1_EF_VALUE_B,
 DIO2_EF_VALUE_B, [Show All](#)

Addresses

44400, 44402, 44404, [Show All](#)

DIO#(0:22)_EF_VALUE_C**Names**

DIO0_EF_VALUE_C, DIO1_EF_VALUE_C,
 DIO2_EF_VALUE_C, [Show All](#)

Addresses

44500, 44502, 44504, [Show All](#)

DIO#(0:22)_EF_VALUE_D**Names**

DIO0_EF_VALUE_D, DIO1_EF_VALUE_D,
DIO2_EF_VALUE_D, [Show All](#)

Addresses

44600, 44602, 44604, [Show All](#)

DIO#(0:22)_EF_READ_A**Names**

DIO0_EF_READ_A, DIO1_EF_READ_A,
DIO2_EF_READ_A, [Show All](#)

Addresses

3000, 3002, 3004, [Show All](#)

DIO#(0:22)_EF_READ_A_AND_RESET**Names**

DIO0_EF_READ_A_AND_RESET,
DIO1_EF_READ_A_AND_RESET,
DIO2_EF_READ_A_AND_RESET, [Show All](#)

Addresses

3100, 3102, 3104, [Show All](#)

DIO#(0:22)_EF_READ_B**Names**

DIO0_EF_READ_B, DIO1_EF_READ_B,
DIO2_EF_READ_B, [Show All](#)

Addresses

3200, 3202, 3204, [Show All](#)

```
$( document ).ready(function() { $('<div>.collapsed-content-expander').closest('<div>.content').find('<div>.sometimes-shown').hide(); $('<div>.collapsed-content-expander').click(function(e) { $(e.target).closest('<div>.content').find('<div>.collapsed-content-expander').fadeOut(function () { $(e.target).closest('<div>.content').find('<div>.sometimes-shown').fadeIn(); }); return false; }); });
```

How-To: Use a digital extended feature

1. Disable features on the DIO using ..._EF_ENABLE
2. Select a feature, and assign the corresponding type index to ..._EF_INDEX
3. Write to ..._EF_OPTIONS (if necessary)
4. Write to ..._EF_VALUE_A, ..._EF_VALUE_B, ..._EF_VALUE_C, ..._EF_VALUE_D (if necessary)
5. Enable feature on the DIO using ..._EF_ENABLE
6. Read results using ..._EF_READ_A, ..._EF_READ_B, or ..._EF_READ_A_AND_RESET

13.1.1 EF Clock Source

The ClockSources produce the reference frequencies used to generate output waveforms and measure input waveforms. ClockSource settings control output frequency, PWM resolution, maximum measurable period, and measurement resolution.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor

There are 3 DIO EF clock sources available. Each clock source has an associated bit size and several mutual exclusions. Mutual exclusions exist because the clock sources share hardware with other features. A ClockSource is created from a hardware counter. CLOCK1 uses COUNTER_A (CIO0) and CLOCK2 uses COUNTER_B (CIO1). The 32-bit clock source (CLOCK0) is created by combining the 2 16-bit clock sources (CLOCK1 CLOCK2). The following list provides ClockSource bit sizes and mutual exclusions.

CLOCK0: 32-bit. Mutual Exclusions: CLOCK1, CLOCK2, COUNTER_A (CIO0), COUNTER_B(CIO1)

CLOCK1: 16-bit. Mutual Exclusions: CLOCK0, COUNTER_A (CIO0)

CLOCK2: 16-bit. Mutual Exclusions: CLOCK0, COUNTER_B (CIO1)

The clock source is not a DIO EF feature, but the four basic operations of Configure, Read, Update, and Reset still apply:

Configure:

There are four registers associated with the configuration of clock sources:

DIO_EF_CLOCK#_ENABLE: 1 = Enable, 0 = Disable. Must be disabled to change the configuration.

DIO_EF_CLOCK#_DIVISOR: 1, 2, 4, 8, 16, 32, 64, or 256 (if this value is zero the divisor will be set to 1).

DIO_EF_CLOCK#_OPTIONS: Reserved for future use. Write 0.

DIO_EF_CLOCK#_ROLL_VALUE: The ClockSource will count to VALUE-1 then roll to zero and repeat. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock. 0 results in the max roll value possible.

A ClockSource can be enabled after DIO EF_INDEX has been configured. This allows several DIO EFs to be started at the same time.

Read:

DIO_EF_CLOCK#_COUNT: Returns the current value of a clock source's counter. This can be useful for generating timestamps.

Update:

At this time there are no update operations available for the DIO EF clock sources. A clock source must be disabled to change any settings.

A smooth update feature has been added in firmware 1.0035. Both the roll_value and the divisor can be written while a clock source is running. As long as the clock source's period is greater than 50 μ s the clock will seamlessly switch to the new settings.

Reset:

At this time there are no reset operations available for the DIO EF clock sources.

Example:

Configure CLOCK0 as a 10 MHz clock source with a roll-value of 1000000.

```
DIO_EF_CLOCK0_ENABLE = 0
DIO_EF_CLOCK0_DIVISOR = 8
DIO_EF_CLOCK0_ROLL_VALUE = 1000000
DIO_EF_CLOCK0_ENABLE = 1
```

With this clock configuration, PWM output (index=0) will have a frequency of 10 Hz. A frequency input measurement (index=3/4) will be able to count from 0-999999 with each count equal to 0.1 microseconds, and thus a max period of about 0.1 seconds.

Advanced:

If CLOCK0 is enabled and CLOCK1 and CLOCK2 are disabled, you can still select CLOCK1 or CLOCK2 as the source for a DIO EF channel. CLOCK1 CLOCK2 are actually the LSW & MSW of CLOCK0. The frequency of CLOCK1 is the same as CLOCK0. If $DIO_EF_CLOCK0_ROLL_VALUE \geq 2^{16}$, then the frequency of CLOCK2 is $CLOCK0_freq / 2^{16}$ divided by the modulus (remainder portion) of $CLOCK0_freq / 2^{16}$. If $(CLOCK0_ROLL_VALUE - 1) < 2^{16}$, then the frequency of CLOCK2 is 0. $CLOCK1_ROLL_VALUE$ is the modulus of $(CLOCK0_ROLL_VALUE - 1) / 2^{16}$ and $CLOCK2_ROLL_VALUE$ is the quotient (integer portion) of $(CLOCK0_ROLL_VALUE - 1) / 2^{16}$.

Digital EF Clock Source

Name	Start Address	Type	Access	Default
DIO_EF_CLOCK0_ENABLE	44900	UINT16	R/W	
DIO_EF_CLOCK0_DIVISOR	44901	UINT16	R/W	
DIO_EF_CLOCK0_OPTIONS	44902	UINT32	R/W	
DIO_EF_CLOCK0_ROLL_VALUE	44904	UINT32	R/W	
DIO_EF_CLOCK0_COUNT	44908	UINT32	R	

DIO_EF_CLOCK0_ENABLE**DIO_EF_CLOCK0_DIVISOR****DIO_EF_CLOCK0_OPTIONS****DIO_EF_CLOCK0_ROLL_VALUE****DIO_EF_CLOCK0_COUNT**

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

13.1.2 PWM Out

Capable DIO: **FIO0, FIO2, FIO3, FIO4, and FIO5**

Requires Clock Source: **Yes**

Index: **0**

PWM Out will generate a pulse width modulated wave form.

Operation:

PWM output will set the DIO high and low relative to the clock source's count. When the the count is zero the DIO line will be set high. When the count matches Value A the line will be set low. Therefore Value A is used to control the duty cycle and the resolution is equal to the roll value.

$Clock\#Frequency = CoreFrequency / DIO_EF_CLOCK\#_DIVISOR$ // typically 80M/Divisor

$PWMFrequency = Clock\#Frequency / DIO_EF_CLOCK\#_ROLL_VALUE$

$DutyCycle\% = 100 * DIO\#_EF_CONFIG_A / DIO_EF_CLOCK\#_ROLL_VALUE$

PWM Out is capable of glitch-free updates in most situations. A glitch-free update means that the PWM will finish the current period

consisting of the high time then the low time before loading the new value. The next period will then have the new duty cycle. This is true for all cases except zero. When setting the duty cycle to zero the line will be set low regardless of the current position. This means that a single high pulse with duration between zero and the previous high time can be output before the line goes low.

Configure:

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 0

DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use. All other bits reserved (write 0).

DIO#_EF_CONFIG_A: When the specified clocks source's count matches this value the line will transition from high to low.

DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_C: Not used.

DIO#_EF_CONFIG_D: Not used.

Update:

The duty cycle can be updated at any time. To update, write the new value to Value_A. The new value will not be used until the clock source rolls to zero. This means that at the end of the current period the new value will be loaded resulting in a glitch-free transition.

DIO#_EF_CONFIG_A: Values written here will set the new duty cycle. The new value will not take effect until the selected clock source rolls to zero.

Read:

No information is returned by PWM Out

Reset:

Reset has no affect on this feature.

Example:

Generate a 10 kHz PWM starting at 25% DC.

First configure the clock source. The higher the roll value the greater the duty cycle resolution will be. For the highest resolution, we want to maximize the roll value, so use the smallest clock divisor that will not result in a roll value greater than the clock source's maximum (32-bits or 16-bits). With a divisor of 1 the roll value will be 8000: $80 \text{ MHz} / (1 * 8000) = 10 \text{ kHz}$. Now set the registers accordingly:

```
DIO_EF_CLOCK0_ENABLE = 0
DIO_EF_CLOCK0_DIVISOR = 1
DIO_EF_CLOCK0_ROLL_VALUE = 8000
DIO_EF_CLOCK0_ENABLE = 1
```

Once the clock source is configured we can use the roll value to calculated Config_A: $DC = 25\% = 100 * \text{Config_A} / 8000$. So Config_A = 2000. Now the PWM can be turned on by writing the proper registers:

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 0
DIO0_EF_CONFIG_A = 2000
DIO0_EF_ENABLE = 1
```

13.1.3 PWM Out with Phase

Capable DIO: **FIO0, FIO2, FIO3, FIO4, FIO5**

Requires Clock Source: **Yes**

Index: **1**

PWM Output with phase control generates PWM waveforms with the pulse positioned at different points in the period. This is achieved by setting the DIO line high and low relative to the clock source's count.

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR // typically 80M/Divisor

PWMFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE

*DutyCycle% = 100 * (DIO#_EF_CONFIG_A - DIO#_EF_CONFIG_B) / DIO_EF_CLOCK#_ROLL_VALUE*

$$\text{PhaseOffset} = 360^\circ * \text{DIO\#_EF_CONFIG_A} / \text{DIO_EF_CLOCK\#_ROLL_VALUE}$$

When the the count matches Value_B the DIO line will be set high. When the count matches Value A the line will be set low. Therefore Value_A minus Value_B controls the duty cycle.

Configure:

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 1

DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use. All other bits reserved (write 0).

DIO#_EF_CONFIG_A: When the clock source's count matches this value the line will transition from high to low.

DIO#_EF_CONFIG_B: When the clock source's count matches this value the line will transition from low to high.

DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_B: Not used.

Update:

The duty cycle can be updated at any time. To update, write the new value to Config_A then Config_B. The value written to Config_A is stored until Config_B is written. After writing Config_B the new value will be loaded at the start of the next period. Updates are glitch-less unless switching from a very high to very low duty cycle or a very low to very high duty cycle.

DIO#_EF_CONFIG_A: Values written here will set the new falling position. The new value will not take effect until Config_B is written.

DIO#_EF_CONFIG_B: Values written here will set the new rising position. When Config_B is written the new Config_A is also loaded.

Read:

No information is returned by PWM Out with Phase.

Reset:

Reset has no affect on this feature.

13.1.4 Pulse Out

Capable DIO: **FIO0, FIO2, FIO3, FIO4, FIO5**

Requires Clock Source: **Yes**

Index: **2**

Pulse output will generate a specified number of pulses. The high time and the low time are specified relative to the clock source the same way as PWM with Phase Control.

$\text{Clock\#Frequency} = \text{CoreFrequency} / \text{DIO_EF_CLOCK\#_DIVISOR}$ // typically 80M/Divisor

$\text{PWMFrequency} = \text{Clock\#Frequency} / \text{DIO_EF_CLOCK\#_ROLL_VALUE}$

$\text{DutyCycle\%} = 100 * (\text{DIO\#_EF_CONFIG_A} - \text{DIO\#_EF_CONFIG_B}) / \text{DIO_EF_CLOCK\#_ROLL_VALUE}$ // if A > B

Configure:

DIO#: First set the DIO line low (DIO#=0). The line must start low for proper pulse generation.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 2

DIO#_EF_OPTIONS: Bits 0-2 specify which clock source to use. All other bits reserved (write 0).

DIO#_EF_CONFIG_A: When the specified clock source's count matches this value the line will transition from high to low.

DIO#_EF_CONFIG_B: When the specified clock source's count matches this value the line will transition from low to high.

DIO#_EF_CONFIG_C: The number of pulses to generate.

DIO#_EF_CONFIG_D: Not used.

Update:

DIO#_EF_CONFIG_A: Sets a new high to low transition point. Will take effect when writing Config_C.

DIO#_EF_CONFIG_B: Sets a new low to high transition point. Will take effect when writing Config_C.

DIO#_EF_CONFIG_C: Writing to this value will start a new pulse sequence. If a sequence is already in progress it will be aborted. Numbers previously written to Config_A or Config_B will take effect when Config_C is written.

Read:

DIO#_EF_READ_A: The number of pulses that have been completed.

DIO#_EF_READ_B: The target number of pulses.

DIO#_EF_READ_C: Not used.

DIO#_EF_READ_D: Not used.

Reset:

DIO#_EF_READ_A_AND_RESET: Reads number of pulses that have been completed. Then restarts the pulse sequence.

Example:

First configure a clock source to drive the pulse generator. Assuming the core frequency is 80 MHz writing the following registers will produce a 1 kHz pulse frequency.

```
DIO_EF_CLOCK0_DIVISOR = 8
DIO_EF_CLOCK0_ROLL_VALUE = 10000
DIO_EF_CLOCK0_ENABLE = 1
```

Thus Clock0Frequency = 80 MHz / 8 = 10 MHz, and PWMFrequency = 10 MHz / 10000 = 1 kHz.

Now that we have a clock to work with we can configure our pulse.

```
DIO0_EF_ENABLE = 0
DIO0 = 0 // set DIO0 to output-low
DIO0_EF_INDEX = 2 // pulse out type index
DIO0_EF_CONFIG_A = 2000 // high to low count
DIO0_EF_CONFIG_B = 0 // low to high count
DIO0_EF_CONFIG_C = 5000 // number of pulses
DIO0_EF_ENABLE = 1
```

Thus duty cycle = $100 * (2000 - 0) / 10000 = 20\%$. The LabJack will now output 5000 pulses over 5 seconds at 20% duty cycle.

13.1.5 Frequency In

Capable DIO: **FIO0, FIO1**

Requires Clock Source: **Yes**

Index: **3 (positive edges) or 4 (negative edges)**

Frequency In will measure a period by counting the number of clock source ticks between two edges ... rising-to-rising (index=3) or falling-to-falling (index=4). The number of ticks can be read from DIO#_EF_READ_A. The following formula will produce period in seconds.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor
Period (s) = DIO#_EF_READ_A / Clock#Frequency
Frequency (Hz) = Clock#Frequency / DIO#_EF_READ_A
```

```
Resolution (s) = 1 / Clock#Frequency
Max Period (s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency
```

Roll value for this purpose would typically be left at the default of 0, which is the max value (2^{32} for the 32-bit Clock0), but you might be using a lower roll value for another feature such as PWM output.

A couple typical scenarios with roll value = 0 and using the 32-bit clock (Clock0):

```
Divisor = 1, Resolution = 12.5 nanoseconds, MaxPeriod = 53.7 seconds
Divisor = 256, Resolution = 3.2 microseconds, MaxPeriod = 229 minutes
```

Once this feature is enabled, a new measurement happens on every applicable edge and the result registers are updated. If you do another read before a new edge has occurred, you will get the same value as before. Many applications will want to use the read-and-reset option so that a value is only read once and extra reads will return 0.

Configure:

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 3 or 4

DIO#_EF_OPTIONS: Bits 0-2 specify which ClockSource to use. All other bits reserved (write 0).

DIO#_EF_CONFIG_A: Not used.

DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_C: Not used.

DIO#_EF_CONFIG_D: Not used.

Update:

No update operations can be performed on Frequency In.

Read:

DIO#_EF_READ_A: Returns the measured time in ClockSource ticks from one edge to another. If a full period has not yet been observed this value will be zero.

DIO#_EF_READ_B: Not used by this feature.

DIO#_EF_READ_A_F: Returns the period in seconds. If a full period has not yet been observed this value will be zero.

Reset:

DIO#_EF_READ_A_AND_RESET: Returns the same data as DIO#_EF_READ_A and then clears the result so that zero is returned by subsequent reads until another full period is measured.

Example:

First, configure the clock source. Roll value would usually be set to 0 to provide the maximum measurable period, but assume for this example that we have to use 10000 because of PWM output on another channel:

```
DIO_EF_CLOCK0_DIVISOR = 8 // Clock0Frequency = 80M/8 = 10 MHz
```

```
DIO_EF_CLOCK0_ROLL_VALUE = 10000
```

```
DIO_EF_CLOCK0_ENABLE = 1
```

This clock configuration results in Resolution = $1 / 10M = 0.1 \text{ us}$ and MaxPeriod = $10000 / 10M = 1 \text{ ms}$.

Now configure the DIO_EF on FIO0 as frequency input.

```
DIO0_EF_INDEX = 3 or 4 // Select rising or falling edges.
```

```
DIO#_EF_OPTIONS = 0 // Select the clock source.
```

```
DIO0_EF_ENABLE = 1 // Turn on the DIO_EF
```

At this point the LabJack is watching the IO lines for the specified edge. After the first two edges have been observed the time between them is stored, and this repeats for each subsequent edge. Results can be read from the READ registers defined above.

13.1.6 Pulse Width In

Capable DIO: **FIO0, FIO1**

Requires Clock Source: **Yes**

Index: **5**

Pulse width in will measure the high time and low time of a periodic signal. The maximum measurable period is controlled by the SourceClock's PulseFrequency. The measurement resolution is controlled by the ClockSourceFrequency and RollValue.

Configure:

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 5

DIO#_EF_OPTIONS: Bits 0-2 specify which ClockSource to use. All other bits reserved (write 0).

DIO#_EF_CONFIG_A: Not used.

DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_C: Not used.

DIO#_EF_CONFIG_D: Not used.

Update:

No update operations can be performed on Pulse Width In.

Read:

DIO#_EF_READ_A: Returns the measured high time in ClockSource ticks and saves the low time so that it can be read later. If a full period has not yet been observed this value will be zero.

DIO#_EF_READ_B: Returns the measured low time in ClockSource ticks. This is the value saved when READ_A or READ_A_F was read.

DIO#_EF_READ_A_F: Returns the measured high time in seconds and saves the low time so that it can be read later. If a full period has not yet been observed this value will be zero.

DIO#_EF_READ_B_F: Returns the measured low time in seconds. This is the value saved when READ_A or READ_A_F was read.

Reset:

DIO#_EF_READ_A_AND_RESET: Performs the same read as described above, but then also clears the register so that zero is returned until another full period is measured.

DIO#_EF_READ_A_F_AND_RESET: Performs the same read as described above, but then also clears the register so that zero is returned until another full period is measured.

Example:

First, configure a clock source to compare the signal against. ($\text{RollValue} / \text{Clock\#Frequency}$) is the maximum measurable period. Be sure that this is greater the maximum expected high or low time for you signal. Higher values for Clock#Frequency result in greater measurement resolution.

```
DIO_EF_CLOCK0_DIVISOR = 8
DIO_EF_CLOCK0_ROLL_VALUE = 10000
DIO_EF_CLOCK0_ENABLE = 1
```

With those settings our maximum measurable period is 1 ms and resolution is 100 ns.

Now configure the DIO_EF on FIO0 as pwm measurement.

```
DIO0_EF_INDEX = 5 // Pulse width type index.
DIO0_EF_OPTIONS = 0 // Set to use clock source zero.
DIO0_EF_ENABLE = 1 // Enable the DIO_EF
```

After a full period beginning with a rising edge has transpired the below result registers can be read:

```
DIO#_EF_READ_A - The high time in clock source counts.
DIO#_EF_READ_B - The low time in clock source counts.
DIO#_EF_READ_A_F - This will return the high time in seconds.
DIO#_EF_READ_B_F - This will return the low time in seconds.
```

READ_A and READ_A_F both return the high time and save the low time that can be read from READ_B and READ_B_F. This ensures that both readings occur at the same time.

13.1.7 Line-to-Line In

Capable DIO: **FIO0, FIO1**

Requires Clock Source: **Yes**

Index: **6**

Line-to-Line In measures the time between an edge on one DIO line to an edge on another DIO line. The edges can be individually specified as rising or falling. The maximum measurable period is controlled by the selected ClockSource's PulseFrequency. The resolution is controlled by $\text{Clock\#Frequency} / \text{DIO_EF_CLOCK\#_ROLL_VALUE}$.

```
MaximumMeasurablePeriod = 1 / PulseFrequency
Resolution = 1 / (PulseFrequency * DIO_EF_CLOCK#\_ROLL_VALUE)
```

Line-to-Line In operates in a one-shot mode. Once the specified combination of edges is observed the data is saved and measuring stops. Another measurement can be started by resetting or performing the configuration procedure again.

Configure:

Configuring Line-to-Line In requires configuring two digital I/O lines as Line-to-Line type index 6. The first DIO configured should be the one expecting the first edge. Any extended features on either DIO should be disabled before beginning configuration.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 6

DIO#_EF_OPTIONS: Bits 0-2 specify which ClockSource to use. All other bits reserved (write 0).

DIO#_EF_CONFIG_A: 0 = falling edge. 1 = rising edge.

DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_C: Not used.

DIO#_EF_CONFIG_D: Not used.

Update:

No update operations can be performed on Line-to-Line In.

Read:

DIO#_EF_READ_A: Returns the measured time in ClockSource ticks. If the specified combination of edges has not yet been observed this value will be zero.

DIO#_EF_READ_B: Not used by this feature.

DIO#_EF_READ_A_F: Returns the time between edges in seconds.

Reset:

DIO#_EF_READ_A_AND_RESET: Performs the same operation as DIO#_EF_READ_A, then clears the result and starts another measurement.

13.1.8 High-Speed Counter

Capable DIO: **CIO0, CIO1, CIO2, CIO3**

Requires Clock Source: **No**

Index: **7**

The T7 supports up to 4 high-speed counters that use hardware to achieve high count rates. These counters are shared with other resources as follows:

CounterA (DIO16/CIO0): Used by EF Clock0 & Clock1.

CounterB (DIO17/CIO1): Used by EF Clock0 & Clock2.

CounterC (DIO18/CIO2): Always available.

CounterD (DIO19/CIO3): Used by stream mode.

Configure:

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 7

DIO#_EF_OPTIONS: Bits 0-2 specify which ClockSource to use. All other bits reserved (write 0).

DIO#_EF_CONFIG_A: When the ClockSource's count matches this value the line will transition from high to low.

DIO#_EF_CONFIG_B: When the ClockSource's count matches this value the line will transition from low to high.

DIO#_EF_CONFIG_C: Not used.

DIO#_EF_CONFIG_D: Not used.

Update:

No update operations can be performed with High-Speed Counter.

Read:

DIO#_EF_READ_A: Returns the current Count

Reset:

DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30us. If missed edges at this point can not be tolerated then reset should not be used.

13.1.9 Interrupt Counter

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, and FIO7**

Requires Clock Source: **No**

Index: **8**

Interrupt Counter counts pulses on the associated IO line. This type of counter is not purely implemented in hardware. The firmware must service each edge. This makes it quite a bit slower than the pure hardware high-speed counter (Index 7). Expect it to top out around TBD ~100kHz.

Configure:

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 8

DIO#_EF_OPTIONS: Not used.

DIO#_EF_CONFIG_A: Not used.

DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_B: Not used.

DIO#_EF_CONFIG_B: Not used.

Update:

No update operations can be performed on Interrupt Counter.

Read:

DIO#_EF_READ_A: Returns the current Count

Reset:

DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30us. If missed edges at this point can not be tolerated then reset should not be used.

13.1.10 Interrupt Counter with Debounce

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, and FIO7**

Requires Clock Source: **No**

Index: **9**

Interrupt Counter with Debounce will count when it receives the specified edge. After counting a timer is started. No received edges will be counted until the timer expires. This type of counter is not purely implemented in hardware. The firmware must service each edge. This makes it quite a bit slower than the pure hardware high-speed counter (Mode 7). Expect it to top out around TBD ~100kHz.

Configure:

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 9
 DIO#_EF_OPTIONS: Not used.
 DIO#_EF_CONFIG_A: Debounce time in microseconds (μ s).
 DIO#_EF_CONFIG_B: bit 0: 1 = Count on Rising edges, 0 = falling edges, 2 = both edges.
 DIO#_EF_CONFIG_B: Not used.
 DIO#_EF_CONFIG_B: Not used.

Update:

No update operations can be performed on Interrupt Counter with Debounce.

Read:

DIO#_EF_READ_A: Returns the current Count

Reset:

DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. Note that there is a brief period of time between reading and clearing during which edges can be missed. During normal operation this time period is 10-30 μ s. If missed edges at this point can not be tolerated then reset should not be used.

13.1.11 Quadrature In

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, and FIO7**

Requires Clock Source: **No**

Index: **10**

Quadrature input uses two DIOs to measure a quadrature signal. Quadrature is a directional count often used in rotary encoders. The T7 uses 4x quadrature decoding, meaning that every edge observed (rising & falling on both phases) will increment or decrement the count. This feature can be used if the expected frequency does not exceed the device-wide edge rate limitation.

Quadrature is prone to error if the edge rate is exceeded. This is particularly likely during direction change where the time between edges can be very small. Errors can occur when two edges come in too quickly for the device to process can result in missed counts or missed change in direction. These errors will be recorded and the quantity encountered can be read. If three edges come in too quickly an undetectable error can occur.

Configure:

Configuring the T7 for quadrature requires configuring two DIO. The first configured will be considered the A line.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
 DIO#_EF_INDEX: 10
 DIO#_EF_OPTIONS: Not used.
 DIO#_EF_CONFIG_A: Not used.
 DIO#_EF_CONFIG_B: Not used.
 DIO#_EF_CONFIG_C: Not used.
 DIO#_EF_CONFIG_D: Not used.

Update:

No update operations can be performed with Quadrature In.

Read:

DIO#_EF_READ_A - Returns the current count.
 DIO#_EF_READ_B - Returns the number of detected errors.

Reset:

DIO#_EF_READ_A_AND_RESET - Performs the same operation as DIO#_EF_READ_A, then sets the count to zero.

Example:

Configure FIO2 & FIO3 as quadrature inputs:

```
DIO2_EF_INDEX = 10
DIO2_EF_ENABLE = 1 //A phase on FIO2
DIO3_EF_INDEX = 10
DIO3_EF_ENABLE = 1 //B phase on FIO3
```

Edges on the two lines will now be decoded and the count will be incremented or decremented according to the edge sequence.

The current count can be read from:

```
DIO2_EF_READ_A or DIO2_EF_READ_A_AND_RESET
```

13.1.12 Interrupt Frequency In

Capable DIO: **FIO0, FIO1, FIO2, FIO3, FIO6, FIO7**

Requires Clock Source: **Yes**

Index: **3 (positive edges) or 4 (negative edges)**

Interrupt Frequency In will measure the frequency of a signal on the associated DIO line. To measure the frequency the LabJack will measure the duration of one or more periods. There are several options available to control the way the LabJack does this. The number of periods to be averaged, the edge direction to trigger on and whether to measure continuously or in a one-shot mode can all be specified.

The clock source for this feature is simply half the core frequency:

```
ClockFrequency = CoreFrequency / 2 //Typically 80M/2 = 40 MHz
```

```
Period (s) = DIO#_EF_READ_A / ClockFrequency
Frequency (Hz) = ClockFrequency / DIO#_EF_READ_A
```

The maximum measurable time is 107 s. The number of periods to be averaged times the maximum expected period must be less than 107 s or the result will overflow: $107 < (\text{NumToAverage} * \text{MaxPeriod})$

By default Interrupt Frequency In will measure the frequency once and return that same result until it is reconfigured or reset. At which point a second measurement will be made. The other option is continuous mode. In continuous mode the frequency is constantly being measured and read returns the most recent result. Running continuous puts a greater load on the processor.

Configure:

```
DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 11
DIO#_EF_OPTIONS: Not Used.
DIO#_EF_CONFIG_A: bit 1: Edge select; 1 = rising, 0 = falling. Bit 2: 1=continuous, 0=OneShot.
DIO#_EF_CONFIG_B: Number of periods to be measured.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.
```

Update:

No update operations can be performed with Interrupt Frequency In.

Read:

```
DIO#_EF_READ_A: Returns the measured time in ticks. This represents the total time elapsed during Value_A averaged periods. Until the specified number of periods has been observed this value will be zero.
DIO#_EF_READ_B: Not used by this feature.
DIO#_EF_READ_A_F: Returns calculated frequency. Takes into account the number of periods to be averaged and the core clock speed.
```

Reset:

```
DIO#_EF_READ_A_AND_RESET: Returns the same data as DIO#_EF_READ_A and then clears the result so that zero is returned by subsequent reads until another full period is measured.
DIO#_EF_READ_A_AND_RESET_F: Returns the same data as DIO#_EF_READ_A_F and then clears the result so that zero is returned by subsequent reads until another full period is measured.
```

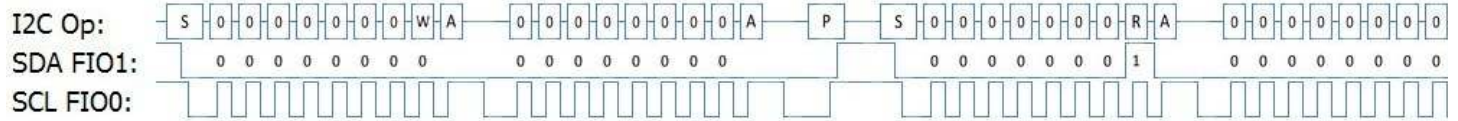
Example:

13.2 I2C

I2C Configuration Utility

A helpful utility to help visualize the T7's I2C functionality and experiment with various configurations can be found [here](#).

Interactive I2C Config Tool



Enter I2C Configuration Settings Below

I2C_SDA_DIONUM:

I2C_SCL_DIONUM:

I2C_SPEED_THROTTLE:

I2C_OPTIONS: Enable Reset-On-Start:

Enable No-Stop on Restart:

Enable Clock Stretching:

Result:

I2C_SLAVE_ADDRESS:

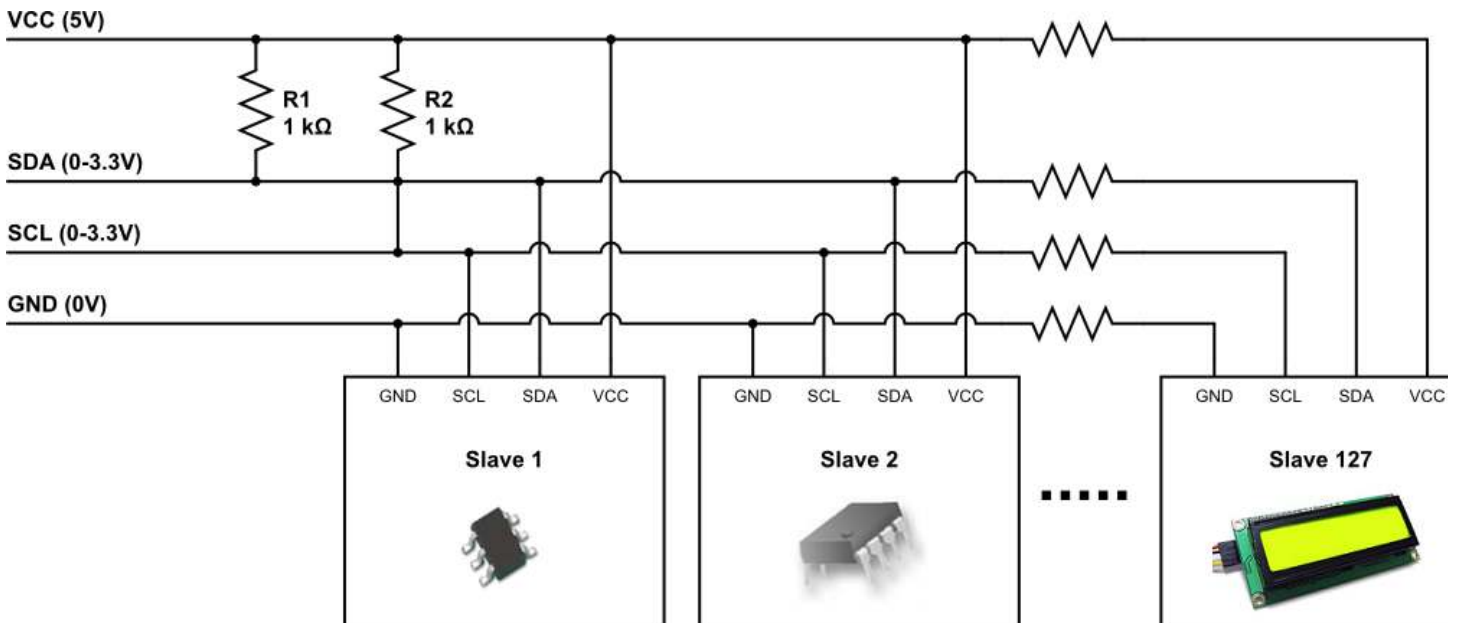
I2C_NUM_BYTES_TX:

I2C_NUM_BYTES_RX:

I2C_WRITE_DATA:

I2C_READ_DATA:

Example I2C Circuit



13.2.1 Configuration Registers

I2C Registers

Name	Start Address	Type	Access	Default
I2C_SDA_DIONUM	5100	UINT16	R/W	0
I2C_SCL_DIONUM	5101	UINT16	R/W	0
I2C_SPEED_THROTTLE	5102	UINT16	R/W	0
I2C_OPTIONS	5103	UINT16	R/W	0
I2C_SLAVE_ADDRESS	5104	UINT16	R/W	0
I2C_NUM_BYTES_TX	5108	UINT16	R/W	0
I2C_NUM_BYTES_RX	5109	UINT16	R/W	0
I2C_WRITE_DATA	5120	BYTE	R/W	0
I2C_READ_DATA	5160	BYTE	R/W	0
I2C_GO	5110	UINT16	R/W	0
I2C_ACKS	5114	UINT32	R/W	0

I2C_SDA_DIONUM

The number of the DIO line that is to be used as the I2C data line. Ex: Writing 0 will force FIO0 to become the I2C-SDA line.

I2C_SCL_DIONUM

The number of the DIO line that is to be used as the I2C clock line. Ex: Writing 1 will force FIO1 to become the I2C-SCL line.

I2C_SPEED_THROTTLE

This value controls the I2C clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~450 kHz. 1 results in ~40 Hz.

I2C_OPTIONS

Advanced. Controls details of the I2C protocol to improve device compatibility.

I2C_SLAVE_ADDRESS

The 7-bit address of the slave device. Value is shifted over automatically to allow room for the I2C R/W bit.

I2C_NUM_BYTES_TX

The number of data bytes to transmit.

I2C_NUM_BYTES_RX

The number of data bytes to read.

I2C_WRITE_DATA

Data that will be written to the I2C bus.

I2C_READ_DATA

Data that has been read from the I2C bus.

I2C_GO

Writing to this register will instruct the LabJack to perform an I2C transaction.

I2C_ACKS

An array of bits used to observe ACKs from the slave device.

13.2.2 I2C Simulation Tool

Click [HERE](#) for a larger version of this tool

This javascript application is designed to help give an understanding of the LabJack's I2C functionality. Each I2C register that effects the output is shown below. The two registers that are omitted are "I2C_GO" and "I2C_ACKS".

"I2C_GO" Executes the configured I2C request, and "I2C_ACKS" reads the received ack's & nack's packed into a binary array.

Test out various I2C configuration settings and view the expected result.



Enter I2C Configuration Settings Below

I2C_SDA_DIONUM:	<input type="text" value="1"/>
I2C_SCL_DIONUM:	<input type="text" value="0"/>
I2C_SPEED_THROTTLE:	<input type="text" value="0"/>
<hr/>	
I2C_OPTIONS:	Enable Reset-On-Start: <input type="checkbox"/> Enable No-Stop on Restart: <input type="checkbox"/> Enable Clock Stretching: <input type="checkbox"/> Result: <input type="text" value="0"/>
I2C_SLAVE_ADDRESS:	<input type="text" value="0"/>
I2C_NUM_BYTES_TX:	<input type="text" value="1"/>
I2C_NUM_BYTES_RX:	<input type="text" value="1"/>
I2C_WRITE_DATA:	<input type="text" value="0"/>
I2C_READ_DATA:	<input type="text" value="xx"/>

Table Of Contents:

- **A:** Indicates an I2C-Ack bit writing a byte of data
- **C:** Indicates an I2C-Clock stretch occurring to slow down transmission
- **N:** Indicates an I2C-Nack bit writing a byte of data
- **P:** Indicates an I2C-Stop Condition
- **R:** Indicates an I2C-Read bit appearing after the slave address is sent
- **RESET:** Indicates an I2C-Reset Condition
- **S:** Indicates an I2C-Start Condition
- **W:** Indicates an I2C-Write bit appearing after the slave address is sent

13.3 SPI

SPI ...

SPI Registers

Name	Start Address	Type	Access	Default
SPI_CS_DIONUM	5000	UINT16	R/W	0
SPI_CLK_DIONUM	5001	UINT16	R/W	0
SPI_MISO_DIONUM	5002	UINT16	R/W	0
SPI_MOSI_DIONUM	5003	UINT16	R/W	0
SPI_MODE	5004	UINT16	R/W	0
SPI_SPEED_THROTTLE	5005	UINT16	R/W	0
SPI_OPTIONS	5006	UINT16	R/W	0
SPI_NUM_BYTES	5009	UINT16	R/W	0
SPI_DATA_WRITE	5010	BYTE	W	0
SPI_DATA_READ	5050	BYTE	R	0

SPI_CS_DIONUM

The DIO line for Chip-Select.

SPI_CLK_DIONUM

The DIO line for Clock.

SPI_MISO_DIONUM

The DIO line for Master-In-Slave-Out.

SPI_MOSI_DIONUM

The DIO line for Master-Out-Slave-In.

SPI_MODE

The SPI mode controls the clock idle state and which edge clocks the data. Bit 1 is CPOL and Bit 0 is CPHA, so CPOL/CPHA for different decimal values: 0 = 0/0 = b00, 1 = 0/1 = b01, 2 = 1/0 = b10, 3 = 1/1 = b11.

SPI_SPEED_THROTTLE

This value controls the SPI clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~1 MHz. 1 results in ~100 Hz.

SPI_OPTIONS

Bit 0 is Auto-CS-Disable.

SPI_NUM_BYTES

The number of bytes to transfer.

SPI_DATA_WRITE

Write data here. Doing so causes the communication to happen.

SPI_DATA_READ

Read data here.

13.4 SBUS

SBUS is a serial protocol used with SHT1X and SHT7x sensors from [Sensirion](#). It is similar to I2C, but not the same. The [EI-1050](#) uses the SHT11 sensor. Other available sensors are the SHT10, SHT15, SHT71, and SHT75.

SBUS Registers

Name	Start Address	Type	Access	Default
SBUS#(0:22)_TEMP	30100	FLOAT32	R	
SBUS#(0:22)_RH	30150	FLOAT32	R	
SBUS#(0:22)_DATA_DIONUM	30200	UINT16	R/W	0
SBUS#(0:22)_CLOCK_DIONUM	30225	UINT16	R/W	1
SBUS_ALL_DATA_DIONUM	30275	UINT16	R/W	0
SBUS_ALL_CLOCK_DIONUM	30276	UINT16	R/W	1
SBUS_ALL_POWER_DIONUM	30277	UINT16	R/W	2

SBUS#(0:22)_TEMP

Reads temperature in degrees Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Names	Addresses
SBUS0_TEMP, SBUS1_TEMP, SBUS2_TEMP, Show All	30100, 30102, 30104, Show All

SBUS#(0:22)_RH

Reads humidity in % from an SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Names	Addresses
SBUS0_RH, SBUS1_RH, SBUS2_RH, Show All	30150, 30152, 30154, Show All

SBUS#(0:22)_DATA_DIONUM

This is the DIO# that the sensor's data line is connected to. Default = FIO0

Names	Addresses
SBUS0_DATA_DIONUM, SBUS1_DATA_DIONUM, SBUS2_DATA_DIONUM, Show All	30200, 30201, 30202, Show All

SBUS#(0:22)_CLOCK_DIONUM

This is the DIO# that the sensor's clock line is connected to. Default = FIO1

Names	Addresses
SBUS0_CLOCK_DIONUM, SBUS1_CLOCK_DIONUM, SBUS2_CLOCK_DIONUM, Show All	30225, 30226, 30227, Show All

SBUS_ALL_DATA_DIONUM

A write to this global parameter sets all SBUS data line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

SBUS_ALL_CLOCK_DIONUM

A write to this global parameter sets all SBUS clock line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

SBUS_ALL_POWER_DIONUM

Sets the power line. This DIO is set to output-high upon any read of SBUS#_TEMP or SBUS#_RH. Default = FIO2. An FIO line can power up to 4 sensors while an EIO/CIO/MIO line or DAC line can power up to 20 sensors. Set to 9999 to disable. To use multiple power lines, use a DAC line for power, or otherwise control power yourself, set this to 9999 and then control power using writes to normal registers such as FIO5, EIO0, or DAC0.

```
$( document ).ready(function() { $(' .collapsed-content-expander').closest(' .content').find(' .sometimes-shown').hide(); $(' .collapsed-content-expander').click(function(e) { $(e.target).closest(' .content').find(' .collapsed-content-expander').fadeOut(function () { $(e.target).closest(' .content').find(' .sometimes-shown').fadeIn(); }); return false; }); });
```

Examples:**EI-1050 probes using default configuration:**

The EI-1050 has an enable line that allows multiple probes to use the same pair of data/clock lines. In this example we connect the 4 basic wires from each probe to the lines specified by the default config:

```
GND    Ground (black)
FIO0   Data (green)
FIO1   Clock (white)
FIO2   Power (red)
```

Since power is provided by FIO2, and FIO lines can only power 4 EI-1050 probes, that is the limitation on number of probes using the default config. We can now connect the enable line from each probe to any DIO we want. Lets use:

```
FIO3      Enable ProbeA (brown)
EIO0/DIO8 Enable ProbeB (brown)
EIO1/DIO9 Enable ProbeC (brown)
EIO2/DIO10 Enable ProbeD (brown)
```

You can now read from SBUS#_TEMP and SBUS#_RH for each probe without writing any config values. In LJLogM, for example, just put the desired register name in any row. A read from SBUS3_TEMP will return the temperature from ProbeA. A read from SBUS9_RH will return the humidity from ProbeC.

Note that when using multiple probes this way, you might need to read one value from each probe before they will work. By default digital I/O are set to input, which has a 100k pull-up, so all 4 probes in this example will be enabled at the same time, which will likely result in a read error. At the end of a read the enable line is set to output-low, so once you do an initial read from each, they will all be disabled and on further reads only one will be enabled at a time.

EI-1050 probes using enable with custom configuration:

Say you connect 2 probes as follows:

```
GND          Ground (black)
EIO0/DIO8    Data (green)
EIO1/DIO9    Clock (white)
EIO2/DIO10   Power (red)

EIO3/DIO11   Enable ProbeA (brown)
EIO4/DIO12   Enable ProbeB (brown)
```

Write the following registers to configure and disable the probes:

```
SBUS_ALL_DATA_DIONUM = 8
SBUS_ALL_CLOCK_DIONUM = 9
SBUS_ALL_POWER_DIONUM = 10
EIO3 = 0
EIO4 = 0
```

You can now read from SBUS11_TEMP/SBUS11_RH for ProbeA values or SBUS12_TEMP/SBUS12_RH for ProbeB values.

EI-1050, SHT1x, or SHT7x, using individual data/clock lines and DAC0 for power:

Say you connect an EI-1050 and SHT71 as follows:

GND Ground (black)
 DAC0 Power (red)

FIO0 Data for EI-1050 (green)
 FIO1 Clock for EI-1050 (white)
 DAC0 Enable EI-1050 (brown)
 FIO2 Data for SHT71
 FIO3 Clock for SHT71

Since the EI-1050 is tied to power, it will always be enabled. We can do that because we have assigned it dedicated DIO for data and clock. The SHT71 does not have an enable. Note that the [SHT7x datasheet](#) shows an added 10k pull-up resistor from Data to Power. The LabJack has an internal 100k pull-up that usually works, but some applications might need the stronger 10k pull-up and perhaps even a capacitor from Clock to GND near the sensor pins.

Write the following registers to configure and power the probes:

```
SBUS0_DATA_DIONUM = 0
SBUS0_CLOCK_DIONUM = 1
SBUS3_DATA_DIONUM = 2
SBUS3_CLOCK_DIONUM = 3
SBUS_ALL_POWER_DIONUM = 9999
DAC0 = 3.3
```

You can now read from SBUS0_TEMP/SBUS0_RH for the EI-1050 values or SBUS3_TEMP/SBUS3_RH for the SHT71 values.

Note that the "#" in the register names above can be about anything you want. Say for the SHT71 you instead did:

```
SBUS7_DATA_DIONUM = 2
SBUS7_CLOCK_DIONUM = 3
```

Now if you read SBUS7_TEMP/SBUS7_RH, the LabJack will use FIO2/3 to talk to the sensor. A possible problem, though, is that the LabJack will also control FIO7 as an enable. It will set FIO7 to output-high, talk to the sensor, and then set FIO7 to output-low. The way to prevent control of an enable line is to use a "#" that is the same as the data or clock line.

13.5 1-Wire

This document assumes that the reader has a basic understanding of the 1-wire protocol.

1-Wire is a serial protocol that uses only one data line. Multiple devices can be connected to a single 1-Wire bus and are differentiated using a unique 64-bit number referred to as ROM.

Hardware:

Devices on the 1-wire bus need to be connected to GND, Vs and the data line DQ. DQ also needs a pullup resistor of 2.2-4.7 kΩ to Vs.

FIO lines can not be used for 1-Wire. They have too much impedance which prevent the signal from reaching logic thresholds.

The T7 supports a DPU (dynamic pull up). A dynamic pull up uses an external circuit such as a transistor to provide extra power to the DQ line at proper times. This can be helpful if the line is large or you are using parasitic power.

Configuration:

ONEWIRE_DQ_DIONUM: This is the DIO line to use for the data line, DQ.

ONEWIRE_DPU_DIONUM: This is the DIO line to use for the dynamic pullup control.

ONEWIRE_OPTIONS: A bit-mask for controlling operation details.

bit 0: Reserved, write 0.

bit 1: Reserved, write 0.

bit 2: DPU Enable. Write 1 to enable the dynamic pullup.

bit 3: DPU Polarity. Write 1 to set the active state as high, 0 to set the active state as low.

ONEWIRE_FUNCTION: This controls how the ROM address of 1-wire devices will be used.

ONEWIRE_NUM_BYTES_WRITE: The number of bytes to transmit to the device. Has no affect when the ROM function is set to Search or Read.

ONEWIRE_NUM_BYTES_READ: The number of bytes to read from the device. Has no affect when the ROM function is set to Search or Read.

ONEWIRE_ROM_MATCH_H: The upper 32-bits of the ROM of the device to attempt to connect to when using the Match ROM function.

ONEWIRE_ROM_MATCH_L: The lower 32-bits of the ROM of the device to attempt to connect to when using the Match ROM function.

ONEWIRE_PATH_H: Upper 32-bits of the search path.

ONEWIRE_PATH_L: Lower 32-bits of the search path.

ROM Functions:

0xF0: Search – This function will read the ROM of one device on the bus. The ROM found is placed in ONEWIRE_SEARCH_RESULT and if other devices were detected the branch bits will be set in ONEWIRE_ROM_BRANCHES_FOUND.

0xCC: Skip – This function will skip the ROM addressing step. For this to work properly only one device may be connected to the bus.

0x55: Match – When using this function data will be sent to and read from a device whose ROM matches the ROM loaded into the ONEWIRE_ROM_MATCH registers.

0x33: Read – Reads the ROM of the connected device. For this to work properly only one device may be connected to the bus.

Sending data:

When using the Match or Skip Rom functions data can be sent to the device. To do so, set the number of bytes to send by writing to ONEWIRE_NUM_BYTES_READ and write the data to ONEWIRE_DATA_READ.

Reading data:

When using the Match or Skip Rom functions data can be read from the device. To do so, set the number of bytes to send by writing to ONEWIRE_NUM_BYTES_WRITE and write the data to ONEWIRE_DATA_WRITE.

Example:

Configure the T7's 1-Wire interface, and obtain a temperature reading from a DS18B22.

Configuration:

Write the common configuration that will not change; the DQ line, DPU, and options. For this example we will use EIO6 (14) as DQ, and the DPU will be left disabled.

```
ONEWIRE_DQ_DIONUM = 8
ONEWIRE_DPU_DIONUM = 0
ONEWIRE_OPTIONS = 0
```

Read ROM:

The 64-bit ROM can be read from the device using the Read ROM function if it is the only device on the bus.

```
ONEWIRE_FUNCTION = 0x33
ONEWIRE_GO = 1
```

The T7 will read the ROM from the connected device and place it in. This test resulted in ROM code 0x1D000005908D4728

Search for ROM:

If there is more than one device on the bus the search function can be used to find the ROM of one of the devices. Note that this method does not provide any information about which device has the ROM discovered.

```
ONEWIRE_PATH = 0
ONEWIRE_FUNCTION = 0xF0
ONEWIRE_GO = 1
```

The LabJack will perform the 1-Wire search function. If a ROM is found it will be placed in ONEWIRE_ROM_SEARCH_RESULT and any branches detected will be indicated in ONEWIRE_BRANCHES. The ONEWIRE_PATH field can be used to direct the LabJack to take a different path in subsequent searches.

Results:

```
ROM - 0x1D000005908D4728
Branches - 0x000000000000002
Now repeat the search with path set to 2.
```

Results:

```
ROM - 0xFF00000024AD2C22
Branches - 0x000000000000002
```

The search can be repeated to find the ROM codes of all devices on the bus.

Write start conversion command to the device:

Do instruct the sensor to start a reading we need to match the device's ROM and send one data byte. The data byte contains the instruction 0xBE.

```

ONEWIRE_FUNCTION = 0x55
ONEWIRE_ROM = 0x1D000005908D4728
ONEWIRE_NUM_BYTES_WRITE = 1
ONEWIRE_DATA_WRITE = [0x44]
ONEWIRE_GO = 1

```

The sensor will now start a conversion. Depending on the sensor and it's settings up to 500 ms may be needed to complete the conversion.

Read conversion result from the device:

After a conversion has been complete we can begin the reading process. This time we need to write the read instruction which is 0x44 and then read 9 bytes of data.

```

ONEWIRE_FUNCTION = 0x55
ONEWIRE_ROM = 0x1D000005908D4728
ONEWIRE_NUM_BYTES_WRITE = 1
ONEWIRE_NUM_BYTES_READ = 9
ONEWIRE_DATA_WRITE = [0xBE]
ONEWIRE_GO = 1

```

We can now read the 9 bytes from ONEWIRE_DATA_READ:

```
0x6A, 0x0A, 0x00, 0x00, 0x24, 0xAD, 0x2C, 0x22, 0x00
```

The 9 bytes contain the binary reading, a checksum, and some other information about the device. The devices used was set to 12-bit resolution so the conversion is 0.0625°C/bit. The binary result is data[0] + data[1]*256. The binary temperature reading is 1*256 + 0x6A = 256 + 106 = 362. To convert that to °C multiply by 0.0625. So the final temperature is 22.6 °C.

1-Wire Registers

Name	Start Address	Type	Access	Default
ONEWIRE_DQ_DIONUM	5300	UINT16	R/W	0
ONEWIRE_DPU_DIONUM	5301	UINT16	R/W	0
ONEWIRE_OPTIONS	5302	UINT16	R/W	0
ONEWIRE_FUNCTION	5307	UINT16	R/W	0
ONEWIRE_NUM_BYTES_TX	5308	UINT16	R/W	0
ONEWIRE_NUM_BYTES_RX	5309	UINT16	R/W	0
ONEWIRE_GO	5310	UINT16	W	0
ONEWIRE_ROM_MATCH_H	5320	UINT32	R/W	0
ONEWIRE_ROM_MATCH_L	5322	UINT32	R/W	0
ONEWIRE_ROM_BRANCHS_FOUND_H	5332	UINT32	R	0
ONEWIRE_ROM_BRANCHS_FOUND_L	5334	UINT32	R	0
ONEWIRE_SEARCH_RESULT_H	5328	UINT32	R	0
ONEWIRE_SEARCH_RESULT_L	5330	UINT32	R	0
ONEWIRE_PATH_H	5324	UINT32	R/W	0
ONEWIRE_PATH_L	5326	UINT32	R/W	0
ONEWIRE_DATA_TX	5340	BYTE	R/W	0
ONEWIRE_DATA_RX	5370	BYTE	R/W	0

ONEWIRE_DQ_DIONUM

The data-line DIO number.

ONEWIRE_DPU_DIONUM

The dynamic pullup control DIO number.

ONEWIRE_OPTIONS

Controls advanced features.

ONEWIRE_FUNCTION

Set the ROM function to use, such as match, skip, search.

ONEWIRE_NUM_BYTES_TX

Number of data bytes to be send.

ONEWIRE_NUM_BYTES_RX

Number of data bytes to be received.

ONEWIRE_GO

Instructs the T7 to perform the configured 1-wire transaction.

ONEWIRE_ROM_MATCH_H

Upper 32-bits of the ROM to match.

ONEWIRE_ROM_MATCH_L

Lower 32-bits of the ROM to match.

ONEWIRE_ROM_BRANCHS_FOUND_H

Upper 32-bits of the branches detected during a search.

ONEWIRE_ROM_BRANCHS_FOUND_L

Lower 32-bits of the branches detected during a search.

ONEWIRE_SEARCH_RESULT_H

Upper 32-bits of the search result.

ONEWIRE_SEARCH_RESULT_L

Lower 32-bites of the search result.

ONEWIRE_PATH_H

Upper 32-bits of the path to take during a search.

ONEWIRE_PATH_L

Lower 32-bits of the path to take during a search.

ONEWIRE_DATA_TX

Data to be transmitted over the 1-wire bus.

ONEWIRE_DATA_RX

Data received over the 1-wire bus.

13.6 Asynchronous Serial

As of firmware 1.0075 the T7 has support for Asynchronous Serial.

Asynchronous Serial

Name	Start Address	Type	Access	Default
ASYNCH_ENABLE	5400	UINT16	R/W	0
ASYNCH_BAUD	5420	UINT32	R/W	0
ASYNCH_RX_DIONUM	5405	UINT16	R/W	0
ASYNCH_TX_DIONUM	5410	UINT16	R/W	0
ASYNCH_NUM_BITS	5415	UINT16	R/W	0
ASYNCH_RX_BUFFER_SIZE_BYTES	5430	UINT16	R/W	0
ASYNCH_NUM_BYTES_RX	5435	UINT16	R	0
ASYNCH_NUM_BYTES_TX	5440	UINT16	R/W	0
ASYNCH_TX_GO	5450	UINT16	W	0
ASYNCH_DATA_TX	5490	BYTE	R/W	0
ASYNCH_DATA_RX	5495	BYTE	R/W	0

ASYNCH_ENABLE

1 = Turn on Asynch. Configures timing hardware, DIO lines and allocates the receiving buffer.

ASYNCH_BAUD

The symbol rate that will be used for communcation.

ASYNCH_RX_DIONUM

The DIO line that will receive data. (RX)

ASYNCH_TX_DIONUM

The DIO line that will transmit data. (TX)

ASYNCH_NUM_BITS

The number of data bits per frame.

ASYNCH_RX_BUFFER_SIZE_BYTES

Number of bytes to use for the receiving buffer. Max is 2048.

ASYNCH_NUM_BYTES_RX

The number of data bytes that have been received.

ASYNCH_NUM_BYTES_TX

The number of bytes to be transmitted after writing to GO.

ASYNCH_TX_GO

Write a 1 to this register to initiate a transmission.

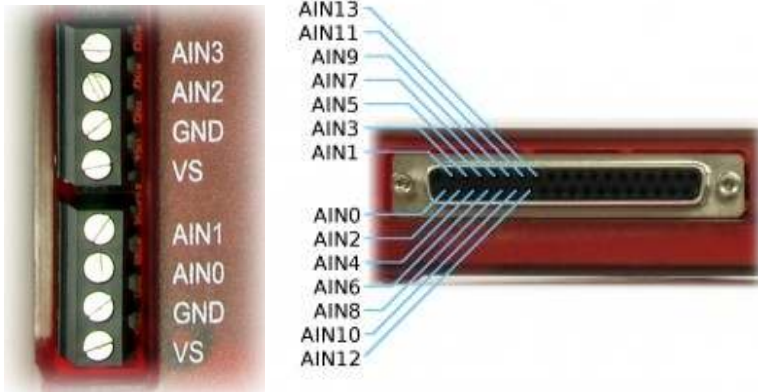
ASYNCH_DATA_TX

Write data to be transmitted here.

ASYNCH_DATA_RX

Read received data from here.

14.0 AIN



Analog Inputs: **14**

Voltage Ranges: **±10V, ±1V, ±0.1V, and ±0.01V**

Analog Input Registers

Name	Start Address	Type	Access	Default
AIN#(0:13)	0	FLOAT32	R	
AIN#(0:13)_RANGE	40000	FLOAT32	R/W	0
AIN#(0:13)_NEGATIVE_CH	41000	UINT16	R/W	199
AIN#(0:13)_RESOLUTION_INDEX	41500	UINT16	R/W	0
AIN#(0:13)_SETTLING_US	42000	FLOAT32	R/W	0
AIN_ALL_RANGE	43900	FLOAT32	R/W	0
AIN_ALL_NEGATIVE_CH	43902	UINT16	R/W	199
AIN_ALL_RESOLUTION_INDEX	43903	UINT16	R/W	0
AIN_ALL_SETTLING_US	43904	FLOAT32	R/W	0

AIN#(0:13)

Returns the voltage of the specified analog input.

Names

AIN0, AIN1, AIN2, [Show All](#)

Addresses

0, 2, 4, [Show All](#)

AIN#(0:13)_RANGE

The range/span of each analog input. If all channels are the same, this reads the correct value, but otherwise this reads -9999.0.

Names

AIN0_RANGE, AIN1_RANGE, AIN2_RANGE, [Show All](#)

Addresses

40000, 40002, 40004, [Show All](#)

AIN#(0:13)_NEGATIVE_CH

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended. If all channels are the same, this reads the correct value, but otherwise this reads 0xFFFF.

Names

AIN0_NEGATIVE_CH, AIN1_NEGATIVE_CH, AIN2_NEGATIVE_CH, [Show All](#)

Addresses

41000, 41001, 41002, [Show All](#)

AIN#(0:13)_RESOLUTION_INDEX

The resolution index for each analog input. A larger resolution index generally results in lower noise and longer sample times. If all channels are the same, this reads the correct value, but otherwise this reads 0xFFFF.

Names	Addresses
AIN0_RESOLUTION_INDEX, AIN1_RESOLUTION_INDEX, AIN2_RESOLUTION_INDEX, Show All	41500, 41501, 41502, Show All

AIN#(0:13)_SETTLING_US

Settling time for command-response readings. If all channels are the same, this reads the correct value, but otherwise this reads -9999.0.

Names	Addresses
AIN0_SETTLING_US, AIN1_SETTLING_US, AIN2_SETTLING_US, Show All	42000, 42002, 42004, Show All

AIN_ALL_RANGE

A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999.

AIN_ALL_NEGATIVE_CH

A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFFFF.

AIN_ALL_RESOLUTION_INDEX

A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFFFF.

AIN_ALL_SETTLING_US

A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999.

```
$( document ).ready(function() { $('<div data-bbox="47 419 177 435" data-label="Section-Header">

## Some Examples


```

Analog Input Example: To read a voltage connected to AIN2, perform a read of AIN2 (or 4), and the result would be in the form of a floating point number, like 8.82332V.

Range Example: It is known that the voltage source connected to AIN1 will be 0 to 0.7V, so write 1.0 (or anything >0.1 and <=1.0) to AIN1_RANGE (40002), and the device will use the ±1V range.

Differential Analog Input Example: To do a differential reading on AIN2, you need to set AIN3 as its negative channel so that the measurement is AIN2-AIN3, rather than the default AIN2-GND (single-ended). Write a value of 3 to AIN2_NEGATIVE_CH (41002). To set it back to single-ended write a value of 199.

Resolution Index Example: Change the AIN1 resolution index to 5 by writing a value of 5 to AIN1_RESOLUTION_INDEX (41501).

Settling Example: Change the settling time of AIN3 to 500uS by writing a value of 500 to AIN3_SETTLING_US (42006), although we recommend a value of 0 which corresponds to automatic settling.

Extra Details

The analog inputs are not artificially pulled to 0.0 volts, as that would reduce the input impedance, so readings obtained from floating channels will generally not be 0.0 volts. The readings from floating channels depend on adjacent channels and sample rate and have little meaning. See related [floating input application note](#).

Notice that the addresses for AIN#, AIN#_RANGE, and AIN#_SETTLING_US increment in steps of 2. That is because they are FLOAT32 and thus each needs 2 16-bit registers. AIN#_NEGATIVE_CH and AIN#_RESOLUTION_INDEX are UNIT16 and thus just use 1 16-bit register each, so their addresses step in increments of 1.

The **AIN#(0..13)_RANGE** parameter is actually controlling the gain of the internal instrumentation amplifier. The in-amp supports gains of x1, x10, x100, and x1000. If you set range=10, you get gain=x1, and the analog input range is ±10 volts. If you set range=1, you get gain=x10, and the analog input range is ±1 volts. Note that the device knows what the internal gain is set to and adjusts the return values to give the voltage at the input terminals, so if you connect a 0.8 volt signal to the input terminals, it will be amplified to 8.0 volts before being digitized, but the reading you get back will be 0.8 volts. Write range=10 to get a range of ±10V (default), range=1 to get a range of ±1V, range=0.1 to get a range of ±0.1V, or range=0.01 to get a range of ±0.01V. If you write a value in between the valid ranges, the larger range will be used.

The **AIN#(0:254)_NEGATIVE_CH** parameter pertains to [differential](#) readings. On the T7, differential channels are adjacent even/odd pairs only, such as AIN2-AIN3. Thus the positive channel must be even and the negative channel must be +1. Only an even channel can have an associated negative channel, so you will never write to odd channels of this register (e.g. never write to AIN3_NEGATIVE_CH/41003). Channel numbers in the extended range (above AIN15), are connected to AIN0-AIN13 and those dictate the even/odd rule, not the extended channel numbers (see the [Mux80 datasheet](#)).

The **AIN#(0:254)_RESOLUTION_INDEX** parameter affects the ADC. A higher Resolution_Index results in lower noise and thus higher effective & noise-free resolution, with the tradeoff of longer sample times. The value passed for Resolution_Index is from 0-8, where 0

corresponds to default, 1 is roughly 16-bit resolution (RMS or effective), and 8 is roughly 19-bit resolution. The T7-Pro has additional Resolution_Index settings 9-12 that use the alternate high-resolution converter (24-bit sigma-delta) and correspond to roughly 19-bit to 22-bit resolution. For command-response readings, the default value of 0 corresponds to Resolution_Index=8 on a T7 and Resolution_Index=9 on a T7-Pro. For stream readings the default of 0 corresponds to Resolution_Index=1.

The **AIN#(0:254)_SETTLING_US** parameter is the time from a step change in the input signal to when the signal is sampled by the ADC, measured in microseconds. A step change in this case is caused when the internal multiplexers change from one channel to another. In general, more settling time is required as gain and resolution are increased. The default “auto” settling time ensures that the device meets specifications at any gain and resolution for source impedance up to at least 1000 ohms. This parameter applies to command/response mode and AIN EF. Stream mode has its own settling parameter. The timings in [Electrical Characteristics](#) are measured with “auto” settling.

14.1 AIN Extended Features

The AIN-EF system is considered in beta status.

Analog extended features (AIN-EF) are used for advanced operations involving analog inputs. Possible features:

- Linear Scaling: Apply a simple slope and offset.
- Thermocouples: Perform the math to handle cold junction compensation (CJC) and voltage to temperature conversion.
- RMS: Acquire a burst of samples and calculate RMS.
- 60Hz Power: Acquire a burst of samples from 2 channels with voltage and current sensors, scale as necessary, and then multiply and integrate to calculate power & energy. This is a future possibility, and not implemented at this time.

Configuration:

Before an AIN_EF can be used a few settings need to be written. The desired operation is selected by writing the associated index number to AIN#(0:13)_EF_INDEX. Depending on the selected index several configuration registers can be written. The details of the configuration registers can be found in the associated index section.

Analog input settings such as range, resolution, and settling, and the negative channel are configured through the normal AIN registers.

Reading:

When READ_A is read from the LabJack will read from analog inputs, run calculations, then return the result. If the selected index produces more than one result they will be saved so that they can be read later. Reading from result registers other than A read from the saved values and do not initiate a new reading.

List of index values:

0: None (disabled)
 1: Slope Offset
 20: Thermocouple type E
 21: Thermocouple type J
 22: Thermocouple type K
 23: Thermocouple type R
 24: Thermocouple type T

Index Values:

1: Slope offset:

Slope-offset allow simple calibration calculations to be performed in the LabJack.

Configuration Registers:

CONFIG_D: CJC_Slope – Custom slope to be applied to the CJC reading. Default is 1.00.

CONFIG_E: CJC_Offset – Custom Offset to be applied to the CJC reading. Default it 0.00.

Result Registers:

READ_A: returns measured volts * slope + offset.

20-24: Thermocouples:

Thermocouple modes read an Analog input connected to a thermocouple and a second specified AIN connected to a CJC sensor. The CJC slope and offset are used to compute the CJC temperature then the thermocouple temperature is calculated. The on-board temperature sensor is the default CJC channel.

Thermocouple readings will use $\pm 0.1V$ for the AIN range if it is set to $\pm 10V$. This is to minimize the amount of configuration needed.

Configuration Registers:

CONFIG_A: TC_Options – Bitmask containing additional options. Bits 0 and 1 control temperature units. The default is kelvin. Bit 0: 1 = Report in °C, Bit 1: 1 = Report in °F.

CONFIG_B: CJC_ModbusAddress – This is the modbus address that will be read to acquire the CJC reading. The default is 60052;

TEMPERATURE_DEVICE_K.

CONFIG_D: CJC_Slope – Custom slope to be applied to the CJC reading. Default is 1.00.

CONFIG_E: CJC_Offset – Custom Offset to be applied to the CJC reading. Default it 0.00.

Result Registers:

READ_A: Final calculated temperature.

READ_B: Final voltage used to calculate temperature.

READ_C: CJC Temperature

READ_D: CJC Volts

Analog Extended Features

Name	Start Address	Type	Access	Default
AIN#(0:13)_EF_READ_A	7000	FLOAT32	R	0
AIN#(0:13)_EF_READ_B	7300	FLOAT32	R/W	0
AIN#(0:13)_EF_READ_C	7600	FLOAT32	R/W	0
AIN#(0:13)_EF_READ_D	7900	FLOAT32	R	0
AIN#(0:13)_EF_INDEX	9000	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_A	9300	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_B	9600	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_C	9900	UINT32	R/W	0
AIN#(0:13)_EF_CONFIG_D	10200	FLOAT32	R/W	0
AIN#(0:13)_EF_CONFIG_E	10500	FLOAT32	R/W	0
AIN#(0:13)_EF_CONFIG_F	10800	FLOAT32	R/W	0
AIN#(0:13)_EF_CONFIG_G	11100	FLOAT32	R/W	0

AIN#(0:13)_EF_READ_A

Names

AIN0_EF_READ_A, AIN1_EF_READ_A,
AIN2_EF_READ_A, [Show All](#)

Addresses

7000, 7002, 7004, [Show All](#)

AIN#(0:13)_EF_READ_B

Names

AIN0_EF_READ_B, AIN1_EF_READ_B,
AIN2_EF_READ_B, [Show All](#)

Addresses

7300, 7302, 7304, [Show All](#)

AIN#(0:13)_EF_READ_C

Names

AIN0_EF_READ_C, AIN1_EF_READ_C,
AIN2_EF_READ_C, [Show All](#)

Addresses

7600, 7602, 7604, [Show All](#)

AIN#(0:13)_EF_READ_D

Names

AIN0_EF_READ_D, AIN1_EF_READ_D,
AIN2_EF_READ_D, [Show All](#)

Addresses

7900, 7902, 7904, [Show All](#)

AIN#(0:13)_EF_INDEX

An index to specify the feature you want.

Names

AIN0_EF_INDEX, AIN1_EF_INDEX,
AIN2_EF_INDEX, [Show All](#)

Addresses

9000, 9002, 9004, [Show All](#)

AIN#(0:13)_EF_CONFIG_A

Names

AIN0_EF_CONFIG_A, AIN1_EF_CONFIG_A,
AIN2_EF_CONFIG_A, [Show All](#)

Addresses

9300, 9302, 9304, [Show All](#)

AIN#(0:13)_EF_CONFIG_B

Names

AIN0_EF_CONFIG_B, AIN1_EF_CONFIG_B,
AIN2_EF_CONFIG_B, [Show All](#)

Addresses

9600, 9602, 9604, [Show All](#)

AIN#(0:13)_EF_CONFIG_C**Names**AIN0_EF_CONFIG_C, AIN1_EF_CONFIG_C,
AIN2_EF_CONFIG_C, [Show All](#)**Addresses**9900, 9902, 9904, [Show All](#)**AIN#(0:13)_EF_CONFIG_D****Names**AIN0_EF_CONFIG_D, AIN1_EF_CONFIG_D,
AIN2_EF_CONFIG_D, [Show All](#)**Addresses**10200, 10202, 10204, [Show All](#)**AIN#(0:13)_EF_CONFIG_E****Names**AIN0_EF_CONFIG_E, AIN1_EF_CONFIG_E,
AIN2_EF_CONFIG_E, [Show All](#)**Addresses**10500, 10502, 10504, [Show All](#)**AIN#(0:13)_EF_CONFIG_F****Names**AIN0_EF_CONFIG_F, AIN1_EF_CONFIG_F,
AIN2_EF_CONFIG_F, [Show All](#)**Addresses**10800, 10802, 10804, [Show All](#)**AIN#(0:13)_EF_CONFIG_G****Names**AIN0_EF_CONFIG_G, AIN1_EF_CONFIG_G,
AIN2_EF_CONFIG_G, [Show All](#)**Addresses**11100, 11102, 11104, [Show All](#)

```
$( document ).ready(function() { $('<div>
.collapsed-content-expander').closest('content').find('.sometimes-shown').hide(); $('<div>
.collapsed-content-expander').click(function(e) { $(e.target).closest('content').find('<div>
.collapsed-content-expander').fadeOut(function () {
$(e.target).closest('content').find('.sometimes-shown').fadeIn(); }); return false; }); });
```

14.1.1 Thermocouples

The AIN extended features system can automatically perform the necessary calculations for Type J, K, thermocouples and a few others.

14.1.2 Offset and Slope

The T7 AIN extended feature system can automatically add a slope or offset to analog readings, the results of which can be read through the AIN EF registers.

14.2 Special Channels

The T7 has special channels to allow for various analog input features. All the analog inputs (0-13) can be used for single-ended readings. For differential readings, channels are grouped into pairs, the positive channel is an even number, and the negative channel is odd. Only adjacent channel numbers can be used as differential, e.g. 0 and 1 form a pair, but 0 and 3 cannot form a pair.

Differential channel pairs

Differential Pair	Positive AIN	Negative AIN
0	0	1
1	2	3
2	4	5
3	6	7
4	8	9
5	10	11
6	12	13

Other unique analog channels

AIN	Function
14	Internal Temperature Sensor (volts)
15,199	Ground (GND)

Special AINs

Name	Start Address	Type	Access	Default
AIN#(14:15)	28	FLOAT32	R	

AIN#(14:15)

Returns the voltage of the specified analog input.

Names	Addresses
AIN14, AIN15	28, 30

It is possible to read GND directly via AIN15, or AIN199. Read more on the temperature sensor in [temperature sensor](#) section.

14.3 Extended Channels

The Mux80 is a ready-made analog input expansion board which adds 80 analog inputs when used in conjunction with a T7. The extended channels can be read using the following registers. For details about differential readings, and physical mapping of pins, see the Mux80 datasheet. Note that when using the Mux80 board, the T7s MIO(0-3) lines are consumed for multiplexer signaling.

Mux80 Extended Channels

Name	Start Address	Type	Access	Default
AIN#(48:127)	96	FLOAT32	R	

AIN#(48:127)

Returns the voltage of the specified analog input.

Names	Addresses
AIN48, AIN49, AIN50, Show All	96, 98, 100, Show All

```
$( document ).ready(function() { $('.'+collapsed-content-expander').closest('.content').find('.sometimes-shown').hide(); $('.'+collapsed-content-expander).click(function(e) { $(e.target).closest('.content').find('.collapsed-content-expander').fadeOut(function () { $(e.target).closest('.content').find('.sometimes-shown').fadeIn(); }); return false; }); });
```

15.0 DAC

Output: **0V to 5V**

Resolution: **12-bit**

Source Impedance: **50 ohms**



DAC Registers

Name	Start Address	Type	Access	Default
DAC#(0:1)	1000	FLOAT32	R/W	

DAC#(0:1)

Pass a voltage for the specified analog output.

Names	Addresses
DAC0, DAC1	1000, 1002

Overview

There are two DACs (digital-to-analog converters or analog outputs) on the T7. Each DAC can be set to a voltage between about 0.02 and 5 volts with 12-bits of resolution.

For electrical specifications, See Appendix TBD.

Although the DAC values are based on an absolute reference voltage, and not the supply voltage, the DAC output buffers are powered internally by Vs and thus the maximum output is limited to slightly less than Vs.

The DACs appear both on the screw terminals and on the DB37 connector. These connections are electrically the same, and the user must exercise caution only to use one connection or the other, and not create a short circuit.

Power-up Defaults

The power-up condition of the DACs can be configured by the user. From the factory, the DACS default to enabled at minimum voltage (~0 volts). Note that even if the power-up default for a line is changed to a different voltage or disabled, there is a delay of about 100 ms at power-up where the DACs are in the factory default condition.

Protection

The analog outputs can withstand a continuous short-circuit to ground, even when set at maximum output.

Voltage should never be applied to the analog outputs, as they are voltage sources themselves. In the event that a voltage is accidentally applied to either analog output, they do have protection against transient events such as ESD (electrostatic discharge) and continuous overvoltage (or undervoltage) of a few volts.

Increase Output to $\pm 10V$

There is an accessory available from LabJack called the [LJTick-DAC](#) that provides a pair of 14-bit analog outputs with a range of ± 10 volts. The LJTick-DAC plugs into any digital I/O block, and thus up to 10 of these can be used per T7 to add 20 analog outputs.

16.0 DB37

Number of Pins: **37**

Screw type: **#4-40**

Contacts: **Gold-coated**

Form factor: **D-Sub**

This high-density connector provides access to the T7 features that are not available on the screw terminal edge of the unit. It brings out analog inputs (AIN), analog outputs (DAC), digital I/O (FIO, MIO), and other signals. Some signals appear on both the DB37 connector and screw terminals, so care must be taken to avoid a short circuit.

Signals shared between T7 screw terminals and the DB37 are denoted in **bold**.



Pinout

DB37 Pinouts						
1	GND	14	AIN9	27	Vs	
2	200uA	15	AIN7	28	Vm+	
3	FIO6	16	AIN5	29	DAC1	
4	FIO4	17	AIN3	30	GND	
5	FIO2	18	AIN1	31	AIN12	
6	FIO0	19	GND	32	AIN10	
7	MIO1	20	10uA	33	AIN8	
8	GND	21	FIO7	34	AIN6	
9	Vm-	22	FIO5	35	AIN4	
10	GND	23	FIO3	36	AIN2	
11	DAC0	24	FIO1	37	AIN0	
12	AIN13	25	MIO0			
13	AIN11	26	MIO2			
DB37 Connector Pinouts						

VS, GND, FIO/MIO, AIN, DAC, 200UA/10UA

Descriptions of these can be found in their related sections of this datasheet.

VM+/VM-

Vm+/Vm- are bipolar power supplies intended to power external multiplexer ICs such as the DG408 from Intersil. The multiplexers can only pass signals within their power supply range, so Vm+/Vm- can be used to pass bipolar signals. Nominal voltage is ± 13 volts at no load and ± 12 volts at 2.5 mA. Both lines have a 100 ohm source impedance, and are designed to provide 2.5 mA or less. This is the same voltage supply used internally by the T7 to bias the analog input amplifier and multiplexers. If this supply is loaded more than 2.5 mA, the voltage can droop to the point that the maximum analog input range is reduced. If this supply is severely overloaded (e.g. short circuited), then damage could eventually occur. If Vm+/Vm- are used to power multiplexers, series diodes are recommended as shown in Figure 9 of the Intersil DG408 datasheet. Not so much to protect the mux chips, but to prevent current from going back into Vm+/Vm-. Use Schottky diodes to minimize voltage drop.

OEM

The OEM T7 has a separate header location to bring out the same connections as the DB37 connector. This OEM header location is labeled J3. The J3 holes are always present, but are obstructed when the DB37 connector is installed. Find the pinout, and other OEM information for J3 in [OEM Versions](#).

17.0 DB15

Number of Pins: **15**

Screw type: **#4-40**

Contacts: **Gold-coated**

Form factor: **D-Sub**

The DB15 connector brings out 12 additional digital I/O. It has the potential to be used as an expansion bus, where the 8 EIO are data lines and the 4 CIO are control lines. EIO0-CIO3 can also be addressed as DIO8-DIO19.



EIO0-EIO7 aka DIO8-DIO15
 CIO0-CIO3 aka DIO16-DIO19

The **CB15** is connector board that provides convenient screw-terminals for the DB15 lines, but the CB15 is not required. Any method you see fit can be used to access the DB15 lines.

These 12 channels include an internal series resistor that provides overvoltage/short-circuit protection. These series resistors also limit the ability of these lines to sink or source current. Refer to the specifications in "Appendix A":/support/u6/users-guide/appendix-a. All digital I/O on the U6 have 3 possible states: input, output-high, or output-low. Each bit of I/O can be configured individually. When configured as an input, a bit has a ~100 kΩ pull-up resistor to 3.3 volts. When configured as output-high, a bit is connected to the internal 3.3 volt supply (through a series resistor). When configured as output-low, a bit is connected to GND (through a series resistor).

DB15 Pinouts				
1	Vs	9	CIO0	
2	CIO1	10	CIO2	
3	CIO3	11	GND	
4	EIO0	12	EIO1	
5	EIO2	13	EIO3	
6	EIO4	14	EIO5	
7	EIO6	15	EIO7	
8	GND			
DB15 Connector Pinouts				

OEM

The OEM T7 has a separate header location to bring out the same connections as the DB15 connector. This OEM header location is labeled J2. The J2 holes are always present, but are obstructed when the DB15 connector is installed. Find the pinout, and other OEM information for J2 in [OEM Versions](#).

18.0 Internal Temp Sensor

Sensor Range: **-50°C to 150°C**

T7 Operating Range: **-40°C to 85°C**

Accuracy (20°C to 40°C): **±1.5°C***

Accuracy (-50°C to 90°C): **±2.1°C***

*Accuracy of measuring device temperature, which is typically warmer than ambient air temperature.



The T7 has an LM94021 temperature sensor connected to internal analog input channel 14 (AIN14). The sensor is physically located on the bottom of the PCB between the AIN0/1 and AIN2/3 screw-terminals. A reading from AIN14 returns volts, which can be converted to device temperature using the formula volts*92.6 + 467.6. Alternatively, read the temperature in degrees Kelvin using the registers TEMPERATURE_AIR_K and TEMPERATURE_DEVICE_K.

Internal Temp Sensor

Name	Start Address	Type	Access	Default
TEMPERATURE_AIR_K	60050	FLOAT32	R	
TEMPERATURE_DEVICE_K	60052	FLOAT32	R	

TEMPERATURE_AIR_K

Returns the estimated ambient air temperature just outside a T7 in its red plastic enclosure. This register is equal to TEMPERATURE_DEVICE_K - 4.3. If Ethernet and/or WiFi is enabled, subtract an extra 0.6 for each.

TEMPERATURE_DEVICE_K

Takes a reading from AIN14 using range= $\pm 10V$ and resolution=8, and applies the formula $\text{Volts} \times 92.6 + 467.6$ to return degrees K. AIN14 is internally connected to an LM94021 (U24) with GS=10 which is physically located on the bottom of the PCB between the AIN0/1 and AIN2/3 screw-terminals.

Offset considerations

The unadjusted sensor reading best reflects the temperature of the device inside the enclosure and the temperature of the AIN0-3 screw-terminals. This is what you get from TEMPERATURE_DEVICE_K in degrees Kelvin.

TEMPERATURE_AIR_K is an estimate of the ambient air temperature outside the device. It is calculated depending on whether Ethernet and/or WiFi is enabled as follows:

USB	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 4.3
USB & Ethernet	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 4.9
USB & WiFi	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 4.9
USB & Ethernet & WiFi	TEMPERATURE_AIR_K = TEMPERATURE_DEVICE_K - 5.5

These offsets were determined from measurements with the enclosure on and in still air. We noted that the time constant was about 12 minutes, meaning that 12 minutes after a step change you are 63% of the way to the new value.

Note on thermocouples

The value from register TEMPERATURE_DEVICE_K best reflects the temperature of the built-in screw-terminals AIN0-AIN3, so use that for cold junction compensation (CJC) if [thermocouples](#) are connected there.

If thermocouples are connected to the CB37, you want to know the temperature of the screw-terminals on the CB37. The CB37 is typically at the same temperature as ambient air, so use the value from register TEMPERATURE_AIR_K for CJC. Better yet, add a [sensor such as the LM34CAZ](#) to an unused analog input on the CB37 to measure the actual temperature of the CB37.

19.0 RTC

The SD card features are in beta. The T7-Pro has a battery-backed RTC (real-time clock) which is useful for assigning timestamps to data that is stored on the SD card during scripting operations. Particularly in situations where the device could experience power failure or other reboots, and does not have a communication connection that can be used to determine real-time after reboot.

As of this writing, the T7-Pro ships with a 2GB SD card, RTC, and battery/battery-holder installed. The T7 has none of these, but does have the socket installed to hold an SD card.

20.0 Internal Flash

The T7 has a flash built-in flash memory chip, and a micro SD card. The built-in flash is used to store calibration constants, among other things. The following registers are used to access built-in flash memory. For information about the SD card, see the related section.

61812: Read 1-512 registers starting from this address to get the data. Flash is read in 32-bit chunks, so you must read an even number of registers. You can only read multiple registers starting from this Modbus address ... you can't read 61812, then read 61814, and so on. The number of registers you can read at once might be further limited by the maximum packet size of the particular interface ... if you don't want to worry about that just stick to 13 values (26 registers) or less per read.

For example: To read 8 floats out of memory, starting at external flash address 3948544, initialize the read pointer (Modbus address 61810) to a value of 3948544 using `eWriteAddress()`, then read Modbus addresses starting at address 61812 using `eReadAddresses()`. The read pointer (address 61810) does not automatically increment.

Calibration Constants

The T7 automatically returns calibrated readings, so most people should not concern themselves with this section.

If the factory applied calibration constants are of interest, they are stored on internal memory and can be accessed at any time through the use of the Modbus registers listed in the table above.

The cal constants begin at memory address 0x3C4000, or in decimal format 3948544. The structure(location) of each calibration value can be seen in the code snippet below.

Follow the above example to read out the first 8 values: PSlope, NSlope, Center, Offset (HS Gain= x1), and PSlope, NSlope, Center, Offset (HS Gain= x10).

```
typedef struct{
    float PSlope;
    float NSlope;
    float Center;
    float Offset;
}Cal_Set;

typedef struct{
    Cal_Set HS[4];
    Cal_Set HR[4];

    struct{
        float Slope;
        float Offset;
    }DAC[2];

    float Temp_Slope;
    float Temp_Offset;

    float ISource_10u;
    float ISource_200u;

    float I_Bias;
}Device_Calibration;
```

The full size of the calibration section is 164 bytes, or 41 floats.

The reason that there are 'Cal_Set's for each High Speed 'HS' and High Resolution 'HR', is that there are 2 analog converters on a T7-Pro. A standard T7 uses only the High Speed analog converter, so only the HS[4] calibration values will be populated with valid information. A T7-Pro will have calibration information for both high speed, and high resolution converters.

Additionally, there are distinct sets of positive slope(Pslope), negative slope(Nslope), Center, and Offset values for each of the 4 gain settings on the device.

High speed AIN calibration values **HS[4]**:

HS[0] = calibration for gain x1
HS[1] = calibration for gain x10
HS[2] = calibration for gain x100
HS[3] = calibration for gain x1000

High resolution (-Pro only) AIN calibration values **HR[4]**:

HR[0] = calibration for gain x1
HR[1] = calibration for gain x10
HR[2] = calibration for gain x100
HR[3] = calibration for gain x1000

21.0 SD Card

The SD card features are in beta. Note that the SD card is really only useful for people who are using scripting, since all other interactions with the T7 can be saved to the host PC harddrive.

As of this writing, the T7-Pro ships with a 2GB SD card, RTC, and battery/battery-holder installed. The T7 has none of these, but does have the socket installed to hold an SD card.

Names are limited to ASCII characters only. Directories and file writing are currently disabled.

Get the name of the current working directory (CWD):

1. Read from FILE_IO_DIR_CURRENT and FILE_IO_NAME_READ_LEN.
2. Read an array of size FILE_IO_NAME_READ_LEN from FILE_IO_NAME_READ.

Get list of items in the CWD:

1. Read from MA_FILE_IO_DIR_FIRST. The value returned indicates whether anything was found. 0 = something was found. FILE_IO_NOT_FOUND (2960) indicates that nothing was found.
2. Read FILE_IO_NAME_READ_LEN, FILE_IO_ATTRIBUTES, and FILE_IO_SIZE.

3. Read an array from FILE_IO_NAME_READ of size FILE_IO_NAME_READ_LEN. This is the name of the file/folder.
4. Read FILE_IO_DIR_NEXT. If the value returned is 2960 we're done, otherwise return to step 2.

Get disk size and free space:

1. Read FILE_IO_DISK_SECTOR_SIZE, FILE_IO_DISK_SECTORS_PER_CLUSTER, FILE_IO_DISK_TOTAL_CLUSTERS, FILE_IO_DISK_FREE_CLUSTERS. Disk operations are performed when you read sector size, all others are snapshots.
2. Total size = sector_size * sectorsPerCluster * Total_Clusters. Free size = sector_size * sectorsPerCluster * free_Clusters

Read a file:

1. Write the length of the file name to FILE_IO_NAME_WRITE_LEN
2. Write the name to FILE_IO_NAME_WRITE
3. Read from FILE_IO_OPEN
4. Read file data from FILE_IO_READ
5. read from FILE_IO_CLOSE

Registers:

Registers related to writing files and directory creation/destruction are place holders.

```
// Directory operations
FILE_IO_CHDIR 60600 // u16
FILE_IO_GETDIR 60601 // u16
FILE_IO_MKDIR 60602 // u16
FILE_IO_RMDIR 60603 // u16
// Directory search
FILE_IO_DIR_FIRST 60610 // u16
FILE_IO_DIR_NEXT 60611 // u16
// File operations
FILE_IO_OPEN 60620 // u16
FILE_IO_CLOSE 60621 // u16
FILE_IO_DEL 60622 // u16
FILE_IO_FILE_ATTRIBS 60623 // u16
FILE-IO_FILE_SIZE 60628 // u32
// Disk information
FILE_IO_DISK_SCTRSIZE 60630 // u32
FILE_IO_DISK_SCTSCLST 60632 // u32
FILE_IO_DISK_CLSTRS 60634 // u32
FILE_IO_DISK_FREECLST 60636 // u32
FILE_IO_DISK_FORMAT 60638 // u32
// Names. This is where strings/arrays are hanled.
FILE_IO_NAME_WRITE_LEN 60640 // u32
FILE_IO_NAME_READ_LEN 60642 // u32
FILE_IO_NAME_WRITE 60650 // AAI binary writing
FILE_IO_NAME_READ 60652 // AAI binary reading
FILE_IO_DATA_WRITE 60654 // AAI binary writing
FILE_IO_DATA_READ 60656 // AAI binary reading
```

22.0 OEM Versions

For pricing/ordering, go to the main [T7 Product Page](#).

The OEM version of the T7 and T7-Pro are shown below. The enclosure, and most connectors are not installed on the OEM version, which allows customers to choose custom connectors.



The following list describes parts that we know to be compatible with the T7 OEM hole patterns. Simply select a connector from each category, and we can order the parts and construct a custom OEM. Custom OEM boards carry additional cost, but they are often necessary for specialized enclosures, and seamless integration with other products.

Of course there are many other connector options available; we can just as easily order/install something not mentioned below. Please don't hesitate to contact us.

The PCB Dimensions can be found in the [Enclosure and PCB Drawings](#) section.

Note: Proper ESD precautions should be taken when handling the PCB directly. Many of the parts are ESD resistant, but depending on the size of the shock, or location, the board might be damaged.

USB

The USB connector is not installed on the T7 OEM. Reference the T7 PCB dimensions for mechanical mating details. Many through-hole Type-B USB connectors are compatible. [On Shore Technology Inc USB-B1HSW6](#), [FCI 61729-0010BLF](#), and [TE Connectivity 292304-2](#) are all good options. The USB connector must be installed on the component side of the PCB.

A special high-retention connector such as the [Samtec USBR-B-S-S-O-TH](#) can also be used, but it does take a good deal of force to unplug a cable from these so they are only recommended when you don't want to unplug very often.

It also possible to simply solder the wires directly, using the image below as a reference.



If you have a shield wire, it can be connected to either of the large mounting holes.

J5 - Alternate Power Supply

Through the use of J5, users can supply 5V to the T7 if a USB connection is not required. The square shaped pad is V+, and the circular pad is GND. It is useful for individuals who only need Ethernet or WiFi. The J5 connector is a 2 pin 0.1" pitch rectangular header. To prevent accidentally switching V+ and GND, use a keyed connector such as [TE Connectivity 3-641215-2](#).

J5				
1	V+	2	GND	
J5 OEM Pin-Header				

The 5V supply from J5 goes through R21 (0.1 ohms) and then connects to the device-wide VS bus. The 5V supply from USB goes through R15 (0.1 ohms) and then connects to VS. On the T7-T7Pro, **R15 & R21 are both installed by default, and thus the connections for both sources are essentially shorted to each other**, and both should not be connected at the same time as one could back-feed the other. If you are going to connect to J5, and there is a possibility of power at the USB connection also, remove R15. You can also replace R15 and R21 with diodes (SMA package) to prevent back-feeding, but even Schottky diodes will have voltage drop that needs to be considered.

Ethernet

The same Ethernet connector is installed on all versions of the T7 due to the inherent magnetic complexities. However, it is possible to 'bring

out' a duplicate Ethernet jack to any custom enclosure with one of the following:

- A short Ethernet cable segment and an RJ45 coupler(Plug to Plug). These couplers come in a few varieties: Free hanging (in-line), Chassis Mount, Panel Mount, Bulkhead, Wall Plate, etc. [Conec 33TS3101S-88N](#) and [Emerson 30-1008KUL](#) are both good options.
- A RJ45 Jack to Plug cable, which is just a standard Ethernet plug on one end, and a Jack (female) on the other end. Again, these come in a wide variety of mounting styles, the simplest of which is the panel mount. [TE Connectivity 1546414-4](#) and [Amphenol RJFEZ2203100BTX](#) are both good options.

If selecting your own Ethernet interconnect, insure that it is RJ45, straight-through, and without magnetics.

WiFi

The WiFi antenna jack is a snap-on/snap-off ultra miniature coaxial connector called male [U.FL](#) (aka AMC, IPEX, IPAX, IPX, MHF, UMC or UMCC). The normal T7-Pro uses a U.FL to bulkhead RP-SMA cable with a length of 140mm, similar to the [Emerson 415-0100-150](#), [Laird 1300-00041](#), [Amphenol 336306-14-0150](#), or [Amphenol 336306-12-0150](#). It also includes a standard RP-SMA 2.4 GHz antenna similar to the [Pulse W1030](#).

To search for U.FL to RP-SMA cable options at Digikey, go to the "[Cable Assemblies => Coaxial Cables \(RF\)](#)" section, and filter by Style = "RP-SMA to IPX" or "RP-SMA to MHF1" or "RP-SMA to UMC" or "RP-SMA to UMCC". Then look at the picture and make sure it looks correct, as the application of the terms male and female are not totally standardized.

As of this writing the T7-Pro-OEM includes the same cable & antenna as the normal T7-Pro, described above.

There are many options for cables and antennas, but one simple option is a wire U.FL whip antenna such as the [Anaren 66089-2406](#) or [Anaren 66089-2430](#).

JP1-JP6 - Screw terminal Locations

The screw terminals are not installed on the OEM T7. Customers will typically use the rectangular header locations (J2, J3) instead of the screw terminals. However, if a different screw terminal style is required, it is possible to buy an OEM T7 and order a custom variety. The screw terminal holes are compatible with almost all 4 position, 0.198" (5.00mm) pitch terminal blocks. A [Weidmuller 9993300000](#) works quite well, and accepts 14-24 AWG wire.

P2, P3 - DB(D-Sub) Locations

The DB15 and DB37 connectors are not installed on an OEM T7. Customers will typically use the rectangular header locations (J2, J3) instead of the DB connectors. However, if a different DB mating style is required, it is possible to buy an OEM T7 and order a custom variety. The DB connectors are standard D-Sub two row receptacles(female sockets), through hole, 15 pin, and 37 pin. The following represent a few valid options.

- [FCI 10090099-S154VLF](#)
- [Sullins Connector Solutions SDS101-PRW2-F15-SN13-1](#)
- [FCI 10090099-S374VLF](#)
- [Sullins Connector Solutions SDS101-PRW2-F37-SN83-6](#)

J2, J3 - Header Locations

Connectors J2 and J3 provide pin-header alternatives to the DB15 and DB37 connectors. The J2 and J3 holes are always present, but are obstructed when the DB15 and DB37 are installed.

J2 - 16 position, 2 row, 0.1" pitch, male pin rectangular header

- Unshrouded - [Harwin Inc M20-9980846](#)
- Unshrouded 3x Taller - [Samtec Inc TSW-108-17-T-D](#)
- Shrouded, Gold Finish - [On Shore Technology Inc 302-S161](#)
- Shrouded, Right Angle - [TE Connectivity 1-1634689-6](#)

J3 - 40 position, 2 row, 0.1" pitch, male pin rectangular header

- Unshrouded - [Harwin Inc M20-9762046](#)
- Unshrouded 3x Taller - [Samtec Inc TSW-120-17-T-D](#)
- Shrouded, Gold Finish - [On Shore Technology Inc 302-S401](#)
- Shrouded, Right Angle - [TE Connectivity 5103310-8](#)
- Shrouded, Gold-Palladium Finish - [TE Connectivity 5104338-8](#)

Sometimes customers order tall pin headers that mate directly to a separate custom PCB. Refer to the pinout details below for electrical connections.

J2					
1	GND	2	VS		
3	CIO0	4	CIO1		
5	CIO2	6	CIO3		
7	GND	8	EIO0		
9	EIO1	10	EIO2		
11	EIO3	12	EIO4		
13	EIO5	14	EIO6		
15	EIO7	16	GND		
J2 OEM Pin-Header					
J3					
1	GND	2	GND	3	PIN20 (10uA)
4	PIN2 (200uA)	5	FIO7	6	FIO6
7	FIO5	8	FIO4	9	FIO3
10	FIO2	11	FIO1	12	FIO0
13	MIO0/CIO0	14	MIO1/CIO1	15	MIO2/CIO2
16	GND	17	Vs	18	Vm-
19	Vm+	20	GND	21	DAC1
22	DAC0	23	GND	24	AIN13
25	AIN12	26	AIN11	27	AIN10
28	AIN9	29	AIN8	30	AIN7
31	AIN6	32	AIN5	33	AIN4
34	AIN3	35	AIN2	36	AIN1
37	AIN0	38	GND	39	GND
40	GND				
J3 OEM Pin-Header					

J4 - Constant Current Sources

Since the screw terminals are not installed on an OEM T7, the J4 header location can be used to gain access to the constant current sources. Any 6 position 0.1" pitch rectangular header will work.

J4					
1	200uA	2	GND		
3	GND	4	GND		
5	10uA	6	VS		
J4 OEM Pin-Header					

J8 - Mechanical

The J8 pin header location is purely for mechanical support for that region of the board. There are no electrical connections to this area. It is a 2 position 0.1" pitch rectangular header.

Pricing/Ordering

For pricing & ordering, go to the main [T7 Product Page](#).

23.0 Watchdog

The Watchdog system can perform various actions if the T7 does not receive any communication within a specified timeout period.

A typical usage example is a program that enables the watchdog to reset the T7 with a 60 second timeout, and then has a loop that talks to the device once per second. If something goes wrong with the software, or some other problem that causes communication to stop, the T7 will reset every 10 seconds until communication resumes.

The watchdog timeout can be set as low as 1 second, but such a low value is usually not a good idea. For example, when a USB device

resets it takes a little time for USB to re-enumerate and software to be able to talk to the device again, so you could get in a situation where the device keeps resetting so often that you can't start talking to it again. This might require using the reset-to-factory jumper (FIO2 <=> SPC).

Watchdog Registers

Name	Start Address	Type	Access	Default
WATCHDOG_ENABLE_DEFAULT	61600	UINT32	R/W	0
WATCHDOG_ADVANCED_DEFAULT	61602	UINT32	R/W	0
WATCHDOG_TIMEOUT_S_DEFAULT	61604	UINT32	R/W	0
WATCHDOG_STARTUP_DELAY_S_DEFAULT	61606	UINT32	R/W	0
WATCHDOG_STRICT_ENABLE_DEFAULT	61610	UINT32	R/W	0
WATCHDOG_STRICT_KEY_DEFAULT	61612	UINT32	R/W	0
WATCHDOG_STRICT_CLEAR	61614	UINT32	W	0
WATCHDOG_RESET_ENABLE_DEFAULT	61620	UINT32	R/W	0
WATCHDOG_DIO_ENABLE_DEFAULT	61630	UINT32	R/W	0
WATCHDOG_DIO_STATE_DEFAULT	61632	UINT32	R/W	0
WATCHDOG_DIO_DIRECTION_DEFAULT	61634	UINT32	R/W	0
WATCHDOG_DIO_INHIBIT_DEFAULT	61636	UINT32	R/W	0
WATCHDOG_DAC0_ENABLE_DEFAULT	61640	UINT32	R/W	0
WATCHDOG_DAC0_DEFAULT	61642	FLOAT32	R/W	0
WATCHDOG_DAC1_ENABLE_DEFAULT	61650	UINT32	R/W	0
WATCHDOG_DAC1_DEFAULT	61652	FLOAT32	R/W	0

WATCHDOG_ENABLE_DEFAULT

Write a 1 to enable the watchdog or a 0 to disable. The watchdog must be disabled before writing any of the other watchdog registers (except for WATCHDOG_STRICT_CLEAR).

WATCHDOG_ADVANCED_DEFAULT

A single binary-encoded value where each bit is an advanced option. If bit 0 is set, IO_CONFIG_SET_CURRENT_TO_FACTORY will be done on timeout. If bit 1 is set, IO_CONFIG_SET_CURRENT_TO_DEFAULT will be done on timeout.

WATCHDOG_TIMEOUT_S_DEFAULT

When the device receives any communication over USB/Ethernet/WiFi, the watchdog timer is cleared. If the watchdog timer is not cleared within the timeout period, the enabled actions will be done.

WATCHDOG_STARTUP_DELAY_S_DEFAULT

This specifies the initial timeout period at device bootup. This is used until the first time the watchdog is cleared or timeout ... after that the normal timeout is used.

WATCHDOG_STRICT_ENABLE_DEFAULT

Set to 1 to enable strict mode.

WATCHDOG_STRICT_KEY_DEFAULT

When set to strict mode, this is the value that must be written to the clear register.

WATCHDOG_STRICT_CLEAR

When running in strict mode, writing the key to this register is the only way to clear the watchdog.

WATCHDOG_RESET_ENABLE_DEFAULT

Timeout action: Set to 1 to enable device-reset on watchdog timeout.

WATCHDOG_DIO_ENABLE_DEFAULT

Timeout action: Set to 1 to enable DIO update on watchdog timeout.

WATCHDOG_DIO_STATE_DEFAULT

The state high/low of the digital I/O after a Watchdog timeout. See DIO_STATE

WATCHDOG_DIO_DIRECTION_DEFAULT

The direction input/output of the digital I/O after a Watchdog timeout. See DIO_DIRECTION

WATCHDOG_DIO_INHIBIT_DEFAULT

The inhibit mask of the digital I/O after a Watchdog timeout. See DIO_INHIBIT

WATCHDOG_DAC0_ENABLE_DEFAULT

Timeout action: Set to 1 to enable DAC0 update on watchdog timeout.

WATCHDOG_DAC0_DEFAULT

The voltage of DAC0 after a Watchdog timeout.

WATCHDOG_DAC1_ENABLE_DEFAULT

Timeout action: Set to 1 to enable DAC1 update on watchdog timeout.

WATCHDOG_DAC1_DEFAULT

The voltage of DAC1 after a Watchdog timeout.

Example

The most common way to use Watchdog is to write:

```
WATCHDOG_ENABLE_DEFAULT=0
WATCHDOG_TIMEOUT_S_DEFAULT=60
WATCHDOG_RESET_ENABLE_DEFAULT=1
WATCHDOG_ENABLE_DEFAULT=1
```

If the device does not receive any communication for 60 seconds, the watchdog will cause the device to reset. So if nothing is talking to the device, it will reset every 60 seconds.

24.0 IO Config, _DEFAULT

_DEFAULT: Any register with _DEFAULT at the end is non-volatile. Whatever value you write to a _DEFAULT register will be retained through a reboot or power-cycle.

IO CONFIG: IO Config is a system that concerns the configuration of many registers, mostly related to I/O on the device. This system includes all writable registers for AIN, DAC, and DIO, among others. IO Config does not include registers that have a _DEFAULT version, which is ETHERNET, WIFI, and WATCHDOG, among others.

- **Default:** Values at reboot/power-up.
- **Current:** Current values.
- **Factory:** Factory values.

IO Config Registers

Name	Start Address	Type	Access	Default
IO_CONFIG_SET_DEFAULT_TO_CURRENT	49002	UINT32	W	
IO_CONFIG_SET_DEFAULT_TO_FACTORY	49004	UINT32	W	
IO_CONFIG_SET_CURRENT_TO_FACTORY	61990	UINT16	W	
IO_CONFIG_SET_CURRENT_TO_DEFAULT	61991	UINT16	W	

IO_CONFIG_SET_DEFAULT_TO_CURRENT

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Current values are retrieved and saved as the new defaults.

IO_CONFIG_SET_DEFAULT_TO_FACTORY

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Factory values are retrieved and saved as the new defaults.

IO_CONFIG_SET_CURRENT_TO_FACTORY

Write a 1 to set current values to factory configuration. The factory values are retrieved from flash and written to the current configuration registers.

IO_CONFIG_SET_CURRENT_TO_DEFAULT

Write a 1 to set current values to default configuration. The default values are retrieved from flash and written to the current configuration registers, thus this behaves similar to reboot/power-up.

Example

Use normal current configuration registers to write some values, and then save those as defaults so they are in effect at power-up:

```
AIN_ALL_RANGE = 0.1 //Set current range of all AIN to +/-0.1V
AIN_ALL_RESOLUTION_INDEX = 12 //Set current resolution index of all AIN to 12.
IO_CONFIG_SET_DEFAULT_TO_CURRENT = 1 //Set power-up defaults to current values.
```

25.0 Scripting (alpha)

Scripting is actively being developed, be sure to update the JSON constants file and firmware.

For simplicity the last known working set of files is attached below. The T7's firmware, ljm_constants.json, and the program all need to work together. ljm_constants.json is under c:\programdata\labjack\ljm

Recent Additions/Changes:

- New application adds the ability to run a script when the T7 starts up.
- New firmware solves a bug with converting constant during compilation.

The T7 can execute Lua code to allow independent operation. This can be used to collect data without a host computer or perform complex tasks producing simple results that a host can read. And probably other things we haven't thought of.

Getting Started:

1. Download the two attachments at the bottom of this page. One is example scripts, the other is an application which will load your code onto the T7 and display any print messages.
2. Make sure your T7 has 1.0039 or later.
3. Launch Lua_Simple_PoC.exe - an example is already loaded. Connect your T7 to your computer and press run. The temperature will be displayed 10 times. Now press stop. Note that pressing stop clears the Lua VM, so even if a program has concluded pressing the stop button is still necessary.
4. Try out other examples attached to the bottom of this page.

Running a script when the T7 powers up.

The T7 can be configured to run a script when it powers on or resets. To use start-up scripts be sure you have the latest Lua_Simple_PoC.exe and follow the following steps:

1. Write and test your script.
2. Click "Save to T7. The script will be saved to the T7's flash memory.
3. Click Enable startup script.

That's it, now when the T7 completes a start-up or reset it will run your script.

Learning more about Lua:

Learning Lua is very easy. There are good tutorials on Lua.org as well as several other independent sites. If you are familiar with the basics of programming such as loops and functions then you should be able to get going just by looking at the examples. If you have suggestions or comments, please email support@labjack.com.

Not sure how to accomplish a goal:

Shoot us an email. We will make a new example and add functions as necessary.

Some things to keep in mind while writing Lua for the T7:

- Names are short and cryptic. String length directly affects execution speed and code size.
- Based on eLua 0.8 which is based on Lua 5.1.4
- Lua supports multi-return: D, T = LJ.TickDelta(LT). Both D and T are returned values.
- Library functions are going to be added and changed as I get feedback. Names are likely to change too.
- On the T7 Lua's one and only numeric data type is IEEE 754 single precision, aka float. This is more important than it sounds. Here a good article on floating point numbers and their pitfalls: [Floating Point Numbers](#)
- Currently examples do not contain any comments. This is because they would consume a lot of code space. Eventually they will be removed during the loading process.

LabJack's Lua library:

The basic Lua libraries are extended by a LabJack specific library. Below are the available functions.

MB.R:

Value = MB.R(Address, dataType)

Modbus read. Will read a single value from a modbus register. That item can be a u16, u32, a float or a string.

MB.W:

MBW(Address, dataType, value)

Modbus write. Writes a single value to a modbus register. The type can be a u16, u32, a float, or a string.

LJ.ledtog:

Toggles status LED. This is just for testing and will be removed.

LJ.Tick:

Ticks = LJ.Tick()

// Reads the core timer. (1/2 core freq).

LJ.IntervalConfig & LJ.CheckInterval:

IntervalConfig and CheckInterval work together to make an easy to use timing function. Set the desired interval time with IntervalConfig, then use CheckInterval to watch for timeouts. The interval period will have some jitter but no overall error. Jitter is typically $\pm 30 \mu\text{s}$ but can be greater depending on processor loading. A small amount of error is induced when the processor's core speed is changed.

Up to 8 different intervals can be active at a time.

LJ.IntervalConfig(handle, time_ms)

handle: 0-7

time_ms: Number of milliseconds per interval.

timeout LJ.CheckInterval(handle)

handle: 0-7

Returns: 1 if the interval has expired. 0 if not.

Example:

```
LJ.IntervalConfig(0, 1000)
```

```
while true do
```

```
  if LJ.CheckInterval(0) then
```

```
    --Code to run once per second here.
```

```
  end
```

```
end
```


LJ.TickDelta: (Deprecated)

TranspiredTicks, CurrnetTicks = LJ.TickDelta(LastTickCount)

This is the main timing function. When called it will determine how many ticks have transpired since LastTickCount and return current ticks. The main way to use this is:

```
local D, T, LT
while true do
  D, T = LJ.TickDelta(LT)
  if D > 40000000 then
    LT = LT + 40000000
  end
end
end
```

That will execute the if condition once per second. There will be some jitter depending on what is going on in the processor. But the average should be spot on.

Lua_SetThrottle:

Set the throttle setting. This controls Lua's processor priority. Value is number of Lua instruction to execute before releasing control to the normal polling loop. After the loop completes Lua will be given processor time again.

Lua_GetThrottle:

Reads the current throttle setting

```
dataTypes:      // These will need to be made consistent with LJM.
#define DP_U16   0
#define DP_U32   1
#define DP_F     2
#define DP_STRING 3
```

Modbus IO: (Help me with the name please)

Ram IO consists of a list of modbus addresses where data can be sent to and read from a Lua script.

Data transferring from the Lua script to modbus is handled with simple RAM. Lua writes to the locations and another modbus master can read that information.

Data transferring to the Lua script, from modbus, is handled differently. When a Lua_IO_Write address is written to the address and data are stored in a linked list. The linked list is read as a FIFO by the Lua script. This prevents issues arising from write order and multiple writes to

the same address. The dataType currently associated with these operations is Float, F32.

The number of floats dedicated to data transferring to Lua is specified. Note that RAM usage is four times this number.

Associated registers:

```
#define MA_LUA_NUM_FLOATS 6002
#define MA_LUA_IO_R 46000
#define MA_LUA_IO_W 47000
```

Lua functions:

IOMem.R

Address, Value = IOMem.R()

Reads from the FIFO a value written to the 47000 range. Address is zero if FIFO is empty.

IOMem.W

void IOMem.W(Address, Value)

Writes to the IO RAM. Values here are simply RAM you can overwrite then and they can be read through modbus at any time.

Example script:

```
MB.W(6006,1,10)
while true do
  add, val = IOMem.R()
  if add > 0 then
    print(string.format("New MB Write: %0.0f %f", add, val))
    IOMem.W(add - 1000, val + 100)
  end
end
```

Writing to 6006 sets the number of floats that you would like to allocate to transferring information from Lua. You can now use a host program to write to the 47000 range, the script will display the address and value received then save that value to a read location. The location is the write address - 1000 and 100 is added to the value. Read from the 46000 range to see the data.

Examples in the works:

- New interval timing function
- Low power logging.

Future Features:

Firmware Related:

- Save a script to flash
- Read a script from flash
- Run saved script at startup
- New easier to use timing interval function.
- Write data to web services such as DAQConnect.





Application (IDE) Related:

- Scan for constants that can be converted using Names To Addresses function in LJM.
- Scripting tool built into Kipling.
- File save and load

Known Issues:

- Lua is using a single precision float for its data-type. This means that working with 32-bit integer registers is difficult.

File attachment:

-  [Examples.zip](#)
-  [T7firmware_010078_2014-03-31.bin](#)
-  [ljm_constants.json](#)
-  [Lua_Simple_PoC.zip](#)

Appendix A - Specifications

Specifications for describing the T7 can be broken down into several primary sections with a few sub-sections. Navigate the following sections to see specifications.

A-1 Data Rates

Command Response Data Rates

Everything besides streaming is done in command/response mode, meaning that all communication is initiated by a command from the host which is followed by a response from the T7.

All communication performed with the T7 is done using the [Modbus TCP](#) protocol. The tests below are done through the [LJM library](#). All writes & reads are done with a single eNames() call, so LJM will use the minimum number of Modbus packets possible, which usually means 1 packet.

Testing Procedure:

The times shown in these graphs were measured using a LabVIEW program on Windows that executes for 1-10 seconds, and then divides the total execution time by numIterations to get an average time per iteration. Thus the execution time includes LabVIEW overhead, LJM library overhead, Windows overhead, communication time (USB/Ethernet/WiFi), and T7 processing time.

A "USB high-high" configuration means the T7 is connected to a high-speed USB2 hub which is then connected to a high-speed USB2 host. Even though the T7 is not a high-speed USB device, such a configuration does provide improved performance. Typical examples of "USB other" would be a T7 connected to an old full-speed hub (hard to find) or more likely the T7 is connected directly to the USB host (your PC) even if the host supports high-speed.

Preemptive Operating Systems and Thread Priority:

It is important to understand that Linux, Mac, and Windows are generally "best-effort" operating systems and not "real-time", meaning that the speeds below can vary based on each individual computer, the hardware inside of it, its currently enabled peripherals, current network traffic, strength of signal, design of the application software, other running software, and many more variables.

If you are trying to make a feedback loop that executes reliably at the desired iteration interval, there are various software issues that should be considered, including thread priority, logging to file, updating the screen, and other programs running on the machine.

Ethernet & USB:

These times are quite predictable. Software issues mentioned above are important, but in terms of hardware the times below will be pretty consistent. The T7 is not maxing out the bandwidth of the USB or Ethernet interface, and these times can usually be maintained even with other substantial activity on the bus.

WiFi:

The WiFi times tend to vary much more than USB or Ethernet. With a solid connection most WiFi packets have an overhead of 3-8 ms, but many will take longer. For example, a test was done in a typical office environment of 1000 iterations that produced an average time of 7.0 ms. 92% of the packets took 3-8 ms, 99% took < 30 ms, and 3 packets took 300 ms.

All WiFi tests were done with an RSSI between -40 (very strong) and -70 (good). An RSSI less than -75 generally reflects a weak connection and the number of packets that experiences retries goes up quickly. An RSSI greater than -35 reflects a very strong connection, typically within a few feet of the access point, and also results in increasing numbers of retries due to saturation of the RF signal.

Speed Results:

Below are time results for typical read and write commands to a T7 with no analog inputs:

	USB High-High (ms)	USB Other (ms)	Ethernet (ms)	Wifi (ms)
No I/O	0.7	2.1	1.1	7
Read All DI	0.7	2.1	1.1	7
Write All DO	0.7	2.1	1.1	7
Write Both DACs	0.7	2.1	1.1	7

Table 21.2.3.1

Below are results for reading 1 or 8 analog inputs at various gain and resolution indices:

Res Index	RMS Res [bits]	1 AIN, USB [ms]	1 AIN, Eth [ms]	1 AIN, WiFi [ms]	8 AIN, USB [ms]	8 AIN, Eth [ms]	8 AIN, WiFi [ms]
Gain = x1 or Range ±10V							
1	16.1	0.6	1.1	7	1.0	1.6	7
2	16.4	0.6	1.1	7	1.1	1.7	5
3	16.9	0.6	1.6	7	1.5	1.7	4
4	17.5	0.6	1.6	7	1.5	1.8	4
5	17.9	0.8	1.6	7	2.5	2.6	24
6	18.4	1.0	1.6	7	3.3	3.7	6
7	18.8	1.3	1.7	7	5.8	5.9	8
8	19.0	1.9	2.4	8	10.2	10.3	13
9	19.7	4.8	4.7	10	29.5	29.9	32
10	20.6	14.9	14.8	23	109.3	110.3	113
11	21.3	68.0	68.4	74	533.9	536.5	539
12	22.0	161.0	162.3	164	1,276.9	1,278.4	1,300
Gain = x10 or Range ±1V							
1	15.5	0.8	1.7	7	3.0	3.1	6
2	15.9	0.8	1.7	7	3.1	3.2	6
3	16.5	1.3	1.7	7	5.8	5.8	9
4	17.1	1.2	1.6	8	6.0	6.1	8
5	17.5	2.4	2.0	8	10.6	10.7	13
6	18.1	3.6	3.6	8	19.6	19.6	22
7	18.3	3.7	3.9	9	21.7	21.8	24
8	18.7	4.4	4.4	9	26.1	26.2	29
9	19.6	4.8	4.9	10	29.5	29.6	32
10	20.3	14.8	14.8	23	109.3	110.9	133
11	21.3	68.1	69.3	74	533.9	537.4	538
12	21.8	161.0	161.7	164	1,277.0	1,279.5	1,306
Gain = x100 or Range ±0.1V							
1	13.9	1.8	2.1	7	9.5	9.6	12
2	14.3	2.9	3.3	8	17.6	17.6	20
3	14.8	6.3	6.4	8	42.0	42.3	44
4	15.3	6.4	6.5	10	42.2	42.2	44
5	15.8	6.4	6.5	11	42.7	43.0	45
6	16.4	11.7	11.6	16	83.8	84.2	86
7	16.8	12.0	12.0	17	86.0	86.6	88
8	17.2	12.5	12.4	18	90.2	91.5	93
9	18.6	4.8	4.8	10	29.6	29.5	32
10	19.3	14.9	14.8	23	109.3	110.0	113
11	19.7	68.1	68.4	74	533.9	538.0	536
12	19.7	161.0	162.6	164	1,277.0	1,278.3	1,300
Gain = x1000 or Range ±0.01V							
1	12.1	6.4	6.4	12	41.7	42.1	64
2	12.6	11.5	11.4	16	81.8	82.2	84
3	13.0	11.5	11.4	16	81.9	82.6	85
4	13.5	11.5	11.5	16	82.3	83.5	85
5	14.0	11.6	11.5	17	82.7	82.8	86
6	14.5	11.7	11.6	17	83.8	84.5	90
7	14.9	12.0	11.9	19	85.9	86.3	92
8	15.2	12.5	12.5	20	90.2	91.2	95
9	15.6	4.8	4.8	10	29.6	30.4	37
10	16.2	14.9	14.8	23	109.3	110.0	114
11	16.4	68.1	68.4	74	533.9	536.8	541
12	16.6	160.9	161.9	164	1,277.0	1,278.3	1,283

Table 21.2.3.2

Note 1: Wi-Fi latency varies depending on network traffic and signal strength. 300ms to 900ms is not uncommon.

Streaming Data Rates

The tables related to this section provide typical stream-related performance results. These results are useful for determining what types of signals can be analyzed using a T7. A T7 is capable of streaming analog data at a steady rate so that various discrete time signal analysis tools can be utilized to interpret data. Depending on your network speed, congestion, computer performance and other factors, you may be able to get results faster than displayed below however the typical user should not rely on this extra performance before individual

environment-based testing has been performed. Maximum Speeds will be different based on what interface is being used to stream data, ethernet or USB. Please note that WiFi streaming is not currently supported.

The data below shows test results from various stream performance parameters. Quite often it may be possible to obtain results indicating faster performance than what is listed. To obtain performance results matching or exceeding the results below it may be necessary control various attributes regarding the use of your device. Stream rates can be limited by a number of different factors, USB connection speed, network traffic, program efficiency, and the running programs priority. Quite often the maximum stream rate is capped by the computer's processing capabilities as calibration of the data coming from the device is done in LJM instead of on the device to increase performance.

Raw Data

Res. Index	Max Stream (Samples/s)	ENOB (RMS)	ENOB (Noise-Free)	Noise (16-bit Counts)	Interchannel Delay (µs)
Gain = x1 or Range ±10V (PRELIMINARY)					
1	TBD	16.31	14.28	3.33	15
2	TBD	16.92	14.57	2.72	26
3	TBD	17.30	14.93	2.11	47
4	TBD	17.97	15.59	1.33	94
5	TBD	18.44	16.04	0.98	180
6	TBD	18.91	16.48	0.72	360
7	TBD	19.35	16.96	0.52	720
8	TBD	19.74	17.37	0.39	1,440
Gain = x10 or Range ±1V (PRELIMINARY)					
1	TBD	16.03	13.69	5.00	210
2	TBD	16.46	14.10	3.76	220
3	TBD	16.83	14.58	2.69	560
4	TBD	17.53	15.12	1.85	590
5	TBD	17.98	15.62	1.32	1,220
6	TBD	18.50	16.07	0.95	2,450
7	TBD	19.00	16.58	0.67	2,800
8	TBD	19.38	16.98	0.50	3,550
Gain = x100 or Range ±0.1V (PRELIMINARY)					
1	TBD	13.83	11.40	24.35	1,040
2	TBD	14.34	11.95	16.62	2,100
3	TBD	14.76	12.33	12.79	4,200
4	TBD	15.28	12.87	8.80	4,250
5	TBD	15.80	13.40	6.08	4,400
6	TBD	16.30	13.86	4.44	4,600
7	TBD	16.76	14.38	3.09	4,900
8	TBD	17.20	14.84	2.26	5,600
Gain = x1000 or Range ±0.01V (PRELIMINARY)					
1					
2					
3					
4					
5					
6					
7					
8					

Table 21.2.4.1

A-2 Digital I/O

General Info

Below you can find information regarding the T7's Digital Input/Output lines.

Parameter	Conditions	Min	Typical	Max	Units
Low Level Input Voltage		-0.3		0.5	Volts
High Level Input Voltage		2.64		5.8	Volts
Maximum Input Voltage (1)	FIO	-10		10	Volts
	EIO/CIO/MIO	-6		6	Volts
Output Low Voltage (2)	No Load		0.01		Volts
---FIO	Sinking 1 mA		0.55		Volts
---EIO/CIO	Sinking 1 mA		0.15		Volts
---EIO/CIO	Sinking 5 mA		0.75		Volts
Output High Voltage (2)	No Load		3.3		Volts
---FIO	Sourcing 1 mA		2.75		Volts
---EIO/CIO	Sourcing 1 mA		3.15		Volts
---EIO/CIO	Sourcing 5 mA		2.6		Volts
Short Circuit Current (2)	FIO		6.3		mA
	EIO/CIO/MIO		22.9		mA
Output Impedance (2)	FIO		550		Ω
	EIO/CIO/MIO		180		Ω

(1) Maximum voltage to avoid damage to the device. Protection works whether the device is powered or not, but continuous voltages over 5.8 volts or less than -0.3 volts are not recommended when the T7 is unpowered, as the voltage will attempt to supply operating power to the T7 possibly causing poor start-up behavior.

(2) These specifications provide the answer to the question. "How much current can the digital I/O sink or source?". For instance, if EIO0 is configured as output-high and shorted to ground, the current sourced by EIO0 is configured as output-high and shorted to ground, the current sourced by EIO0 into ground will be about 16 mA (3.3/180). If connected to a load that draws 5 mA, EIO0 can provide that current but the voltage will droop to about 2.4 volts instead of the nominal 3.3 volts. If connected to a 180 ohm load to ground, the resulting voltage and current will be about 1.65 volts @ 9 mA.

Extended Features

Below you can find information regarding the T7's DIO EF.

Extended Features	Conditions	Min	Typical	Max	Units
Frequency Output (1)		0.02		5 M	Hz
Counter Input Frequency (2)				5	MHz
Input Timer Total Edge Rate (3)(4)	No Stream		100 k	140 k	edges/s
	While Streaming @ 50 kHz			30 k	edges/s

(1) Frequencies up to 40MHz are possible, but they are heavily filtered.

(2) Hardware counters. 0 to 3.3 volt square wave.

(3) To Avoid missing edges, keep the total number of applicable edges on all applicable timers below this limit.

(4) Highly dependant on processor loading. Operation such as stream will greatly reduce the maximum edge rate.

Serial Communication

Below you can find information regarding the T7's Serial Communication abilities. Please keep in mind our devices use 3.3V logic levels and provide 5V output along the VS screw terminal. Some ICs require the same logic level as provided to the chip's VCC line so extra steps may be required to integrate specific sensors.

Serial Communication	Conditions	Min	Max	Units
SPI Characteristics				
Clock Frequencies		0.08718	870	kHz
I2C Characteristics				
Clock Frequencies		9.3	472	kHz

A-3 Analog Input

General Info

Below you can find general information regarding the T7's Analog input lines. For further details consult the Noise and Resolution subsection.

Parameter	Conditions	Min	Typical	Max	Units
Typical Input Range (1)	Gain=1	-10.5		10.1	Volts
Max AIN Voltage to GND (2)	Valid Readings	-11.5		11.5	Volts
Max AIN Voltage to GND (3)	No Damage	-20		20	Volts
Input Bias Current (4)			20		nA
Input Impedance (4)			1		GΩ
Source Impedance (4)			1		kΩ
Integral Linearity Error	Gain=1, 10, 100			±0.01	%FS
	Gain=1000			±0.1	%FS
Absolute Accuracy	Gain=1, 10, 100			±0.01	%FS
	Gain=1000			±0.1	%FS
Temperature Drift			15		ppm/°C
Noise (Peak-To-Peak)	See A-3-1			<1	μV
Effective Resolution (RMS)	See A-3-1			22	bits
Noise-Free Resolution	See A-3-1			20	bits

(1) Differential or single-ended

(2) This is the maximum voltage on any AIN pin compared to ground for valid measurements on that channel. For single-ended readings on the channel itself, inputs are limited by the "Typical Input Range" above, and for differential readings consult the signal range tables in Appendix-3-2. Further, if a channel has over 13.0 volts compared to ground, readings on other channels could be affected. Because all even channels are on 1 front-end mux, and all odd channels on a 2nd front-end mux, an overvoltage (>13V) on a single channel will generally affect only even or only odd channels.

(3) Maximum voltage, compared to ground, to avoid damage to the device. Protection level is the same whether the device is powered or not.

(4) The key specification here is the maximum source impedance. As long as the source impedance is not over this value, there will be no substantial errors due to impedance problems. For source impedance greater than this value, more settling time might be needed.

A-3-1 Noise And Resolution

Overview & Testing procedure

The graphs and raw data table under this section provides typical noise levels of the T7 under ideal conditions. The resulting voltage resolution is then calculated based on the noise levels.

Measurements were taken with AIN0 connected to GND with a short jumper wire, or from internal ground channel #15.

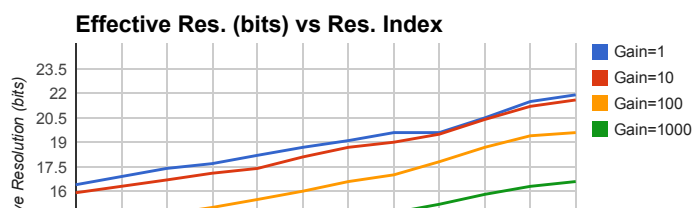
All "counts" data are aligned as 24-bit values. To equate to counts at a particular resolution (Res) use the formula $\text{counts}/(2^{(24-\text{Res})})$. For instance, with the T7 set to resolution=1 and the ±10 volt range, there are 1024 counts of noise when looking at 24-bit values. To equate this to 16-bit data, we take $1024/(2^8)$ which equals 4 counts of noise when looking at 16-bit values.

Noise-free data is determined by taking 2000 readings and subtracting the minimum value from the maximum value.

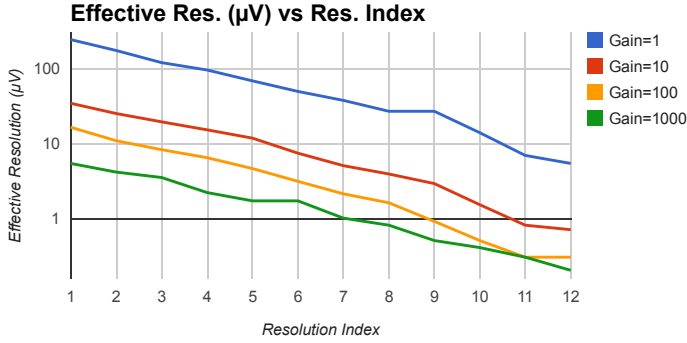
RMS and Effective data are determined from the standard deviation of 2000 readings. In other words, the RMS data represents most readings, whereas noise-free data represents all readings.

Graphical Results

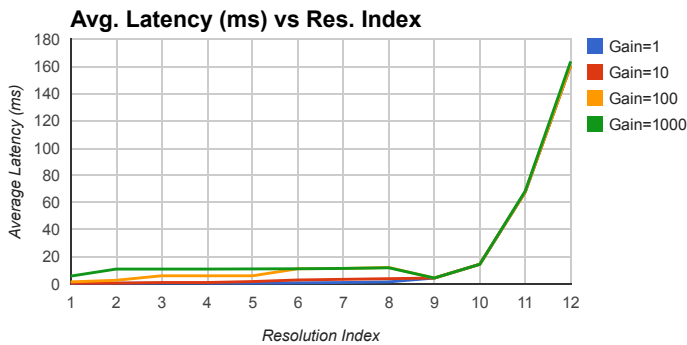
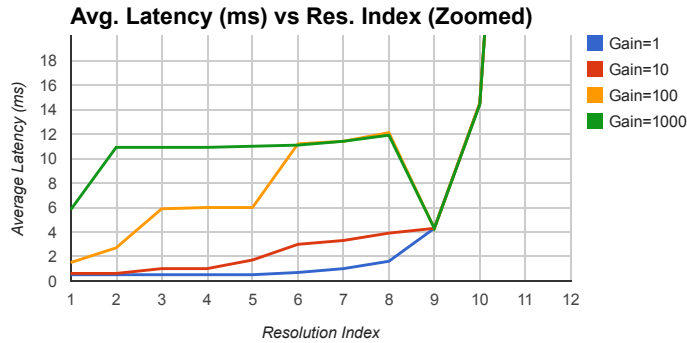
The graph below shows the Effective Resolution in bits that the LabJack is able to produce that correlate to a given input voltage at different gain and resolution-index configurations. It is clear to see that a higher resolution-index produces a more precise result.



The graph below shows the Effective Resolution in μV that the LabJack is able to produce that correlate to a given input voltage at different gain and resolution-index configurations. It is clear to see that a higher resolution-index produces a more precise result. It also becomes clear in this graph that choosing a proper gain level that corresponds to the expected voltage is important.



The graph below shows the average time it takes for LabVIEW to capture a single reading for various resolution-index values. The first chart shows a zoomed in view of the data, the second shows the full range of latencies. When using a T7-Pro the included high resolution converter starts being used at resolution-index 9. Relating this to the graphs, there is a noticeable drop in command response latency when jumping from resolution-index 8 to resolution-index 9 at all gain levels. Shown too by these graphs, the high resolution converter has a higher input impedance than the high speed converter used for resolution-index values 1-8, therefore it requires less time at all gain levels to acquire data.



Raw Data

The data table below shows all of the information collected to produce the above graphs along with some more useful data pertaining to the T7.

Resolution Index	Average Latency	Rounded P2P Noise	Peak-To-Peak Resolution	Noise-Free Resolution	Rounded RMS Noise	Effective Resolution	Effective Resolution
	ms	24-bit counts	bits	μV	24-bit counts	bits	μV
Gain/Range: 1/±10							
1	0.5	1280	13.7	1579.4	197	16.4	243.5
2	0.5	768	14.4	947.7	141	16.9	174.2
3	0.5	640	14.7	789.7	97	17.4	119.8
4	0.5	512	15	631.8	78	17.7	95.8
5	0.5	384	15.4	473.8	55	18.2	68.3
6	0.7	256	16	315.9	40	18.7	49.4
7	1	192	16.4	236.9	30	19.1	37.4
8	1.6	128	17	157.9	22	19.6	26.8
9	4.3	157	16.7	193.8	22	19.6	26.9
10	14.5	79	17.7	97.5	11	20.5	13.9
11	67.6	39	18.7	48.1	6	21.5	6.9
12	160.5	33	19	40.7	4	21.9	5.4
Gain/Range: 10/±1							
1	0.6	2048	13	252.7	280	15.9	34.5
2	0.6	1408	13.5	173.7	202	16.3	25
3	1	1280	13.7	157.9	156	16.7	19.3
4	1	832	14.3	102.6	122	17.1	15.1
5	1.7	512	15	63.2	95	17.4	11.7
6	3	448	15.2	55.3	60	18.1	7.4
7	3.3	256	16	31.6	40	18.7	5
8	3.9	256	16	31.6	32	19	3.9
9	4.3	173	16.6	21.4	23	19.5	2.9
10	14.5	91	17.5	11.2	12	20.4	1.5
11	67.6	61	18.1	7.5	7	21.2	0.8
12	160.5	43	18.6	5.3	5	21.6	0.7
Gain/Range: 100/±0.1							
1	1.5	12545	10.4	154.8	1328	13.6	16.4
2	2.7	9601	10.8	118.4	874	14.2	10.8
3	5.9	7232	11.2	89.2	665	14.6	8.2
4	6	6400	11.4	79	516	15	6.4
5	6	5312	11.6	65.5	373	15.5	4.6
6	11.2	2944	12.5	36.3	252	16	3.1
7	11.4	1856	13.1	22.9	168	16.6	2.1
8	12.1	1472	13.5	18.2	127	17	1.6
9	4.3	1604	13.4	19.8	76	17.8	0.9
10	14.5	503	15	6.2	39	18.7	0.5
11	67.5	335	15.6	4.1	25	19.4	0.3
12	160.5	285	15.8	3.5	22	19.6	0.3
Gain/Range: 1000/±0.01							
1	5.8	68818	7.9	84.9	4354	11.9	5.4
2	10.9	80075	7.7	98.8	3300	12.3	4.1
3	10.9	89828	7.5	110.8	2877	12.5	3.5
4	10.9	49375	8.4	60.9	1778	13.2	2.2
5	11	35113	8.9	43.3	1362	13.6	1.7
6	11.1	38950	8.8	48	1413	13.5	1.7
7	11.4	22129	9.6	27.3	822	14.3	1
8	11.9	10425	10.7	12.9	643	14.7	0.8
9	4.3	7622	11.1	9.4	437	15.2	0.5
10	14.4	2837	12.5	3.5	285	15.8	0.4
11	68.2	1725	13.2	2.1	210	16.3	0.3
12	163.5	2102	13	2.6	171	16.6	0.2

Table 21.2.2.1

A-3-2 Signal Range

The following figures show the approximate signal range of the T7 analog inputs. "Input Common-Mode Voltage" or V_{cm} is $(V_{pos} + V_{neg})/2$.

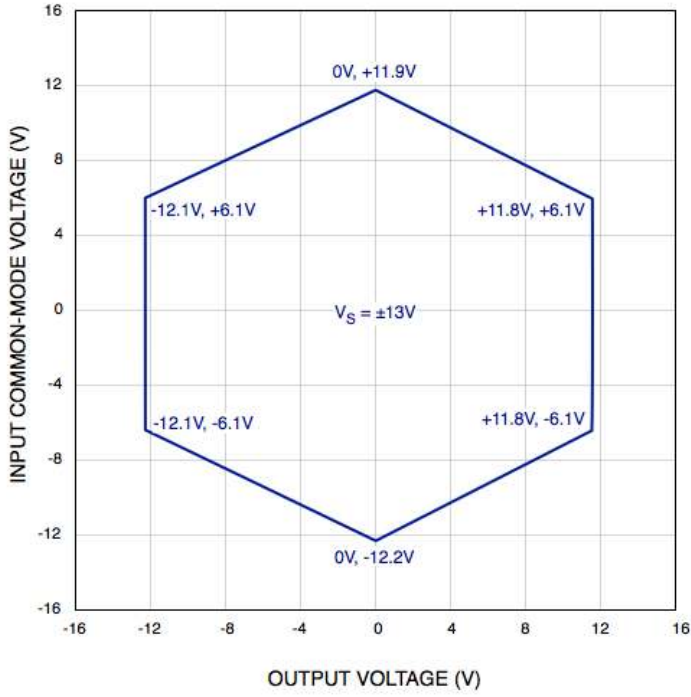
Keep in mind that the voltage of any input compared to GND should be within the V_{m+} and V_{m-} rails by at least 1.5 volts, so if V_m is the typical ± 13 volts, the signals should be within ± 11.5 volts compared to GND.

Example #1: Say a differential signal is measured where V_{pos} is 10.05 volts compared to GND and V_{neg} is 9.95 volts compared to ground, and $G=100$. That means $V_{cm}=10.0$ volts, $V_{diff}=0.1$ volts, and the expected $V_{out}=10.0$ volts. There is not figure for $G=100$ below, but $V_{cm}=10.0$ volts and $V_{out}=10.0$ volts is not valid at $G=1$ or $G=1000$, so is certainly not valid in between.

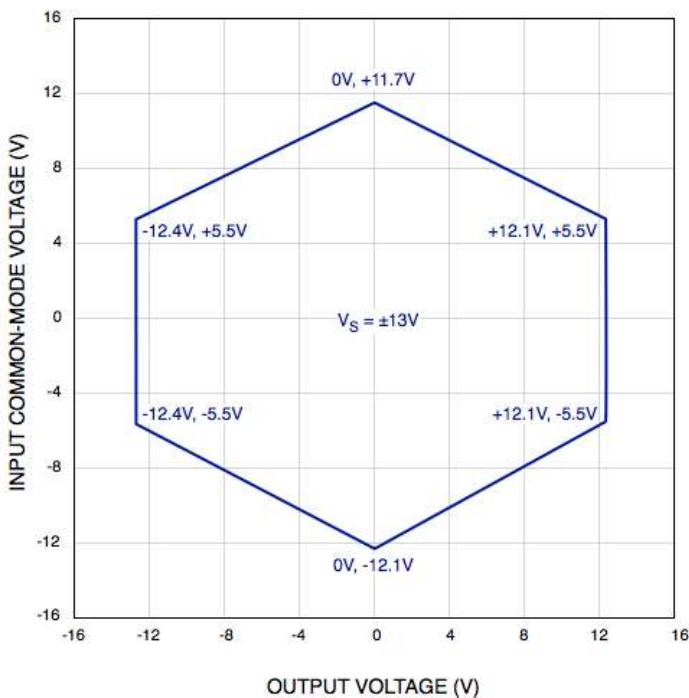
Example #2: Say a differential signal is measured where V_{pos} is 15.0 volts compared to GND and V_{neg} is 14.0 volts compared to ground, and $G=1$. That means $V_{cm}=14.5$ volts, $V_{diff}=1.0$ volts, and the expected $V_{out}=1.0$ volts. The voltage of each input compared to GND is too high, so this would not work at all.

Example #3: Say a single-ended signal is measured where V_{pos} is 10.0 volts compared to GND and $G=1$.; That means $V_{cm}=5.0$ volts, $V_{diff}=10.0$ volts, and the expected $V_{out}=10.0$ volts. This is fine according to the figure below.

Input Common-Mode Voltage Range vs. Output Voltage, $G = 1$



Input Common-Mode Voltage Range vs. Output Voltage, $G = 1000$



A-4 Analog Output

21.3.0 General Information

The T7 supports two analog output channels labeled "DAC0" and "DAC1". General characteristics of the two channels are available below.

Parameter	Conditions	Min	Typical	Max	Units
Nominal Output Range (1)	No Load	0.01		4.99	Volts
	@ ±2.5 mA	0.25		.025	Volts
Resolution			12		Bits
Absolute Accuracy	5% to 95% FS		TBD		% FS
Integral Linearity Error			±1.5	±2	counts
Differential Linearity Error			±0.25	±0.5	counts
Error Due To Loading	@ 100 µA		0.15		%
	@ 1mA		2.3		%
Source Impedance			TBD		Ω
Short Circuit Current (2)	Max to GND		20.5		mA
Time Constant			4		µs

(1) Maximum and minimum analog output voltage is limited by the supply voltages (Vs and GND). The specifications assume Vs is 5.0 volts. Also, the ability of the DAC output buffer to driver voltages close to the power rails, decreases with increasing output current, but in most applications the output is not sinking/source much current as the output voltage approaches GND.

(2) Continuous short circuit will not cause damage.

21.3.1 Speed and Settling

Below you can find some characteristics involving the speed & settling times of the DAC channels.

TBD.

A-5 General Specs

Supply

The following table shows the supply voltage that is required by the T7. The USB hub, or 5V USB adapter should fall within the acceptable range.

Parameter	Condition	Min	Typical	Max	Units
Supply Voltage		4.75		5.25	Volts
Supply Current	No connected loads	8.1	250		mA

Power Consumption

The T7 has several power domains. USB and Core speed are not yet ready user level control, but have been included in the following table to show the capabilities of the device. The values shown are typical.

Core Speed	Eth (1)	Eth Linked	AINs	WiFi	WiFi Linked	LEDs	USB (1)	Draw (mA)
80M	ON	Yes	ON	ON	Yes	ON	ON	290
80M	ON	Yes	ON	ON	No	ON	ON	285
80M	ON	Yes	ON	OFF	No	ON	ON	253
80M	ON	No	ON	OFF	No	ON	ON	210
80M	OFF	No	ON	OFF	No	ON	ON	174
80M	OFF	No	OFF	OFF	No	ON	ON	142
80M	OFF	No	OFF	OFF	No	OFF	ON	105
80M	OFF	No	OFF	OFF	No	OFF	OFF	79
20M	OFF	No	OFF	OFF	No	OFF	OFF	23
2M	OFF	No	OFF	OFF	No	OFF	OFF	8.8
250k	OFF	No	OFF	OFF	No	OFF	OFF	8.1

1) Ethernet and USB require that the core be running at least 20MHz.

200µA and 100µA Current Sources

<i>Parameter</i>	<i>Condition</i>	<i>Min</i>	<i>Typical</i>	<i>Max</i>	<i>Units</i>
Absolute Accuracy	~ 25 °C		±0.1	±0.2	%
Temperature Coefficient			TBD		ppm/°C
Maximum Voltage			VS - 2.0		volts

VM+/VM-

<i>Parameter</i>	<i>Condition</i>	<i>Min</i>	<i>Typical</i>	<i>Max</i>	<i>Units</i>
Typical Voltage	No-load		±13		volts
	@ 2.5 mA		±12		volts
Maximum Current			2.5		mA

System Clock

<i>Parameter</i>	<i>Condition</i>	<i>Min</i>	<i>Typical</i>	<i>Max</i>	<i>Units</i>
Clock Error	~ 25 °C			±20	ppm
	-10 to 60 °C			±50	ppm
	-40 to 85 °C			±100	ppm

Mechanical

<i>Parameter</i>	<i>Condition</i>	<i>Min</i>	<i>Typical</i>	<i>Max</i>	<i>Units</i>
USB Cable Length			2	5	meters
Operating Temperature		-40		85	°C
Screw Terminal Wire Gauge			26	14	AWG
Mounting Screws	wood screw sizes	#4	#6	#8	

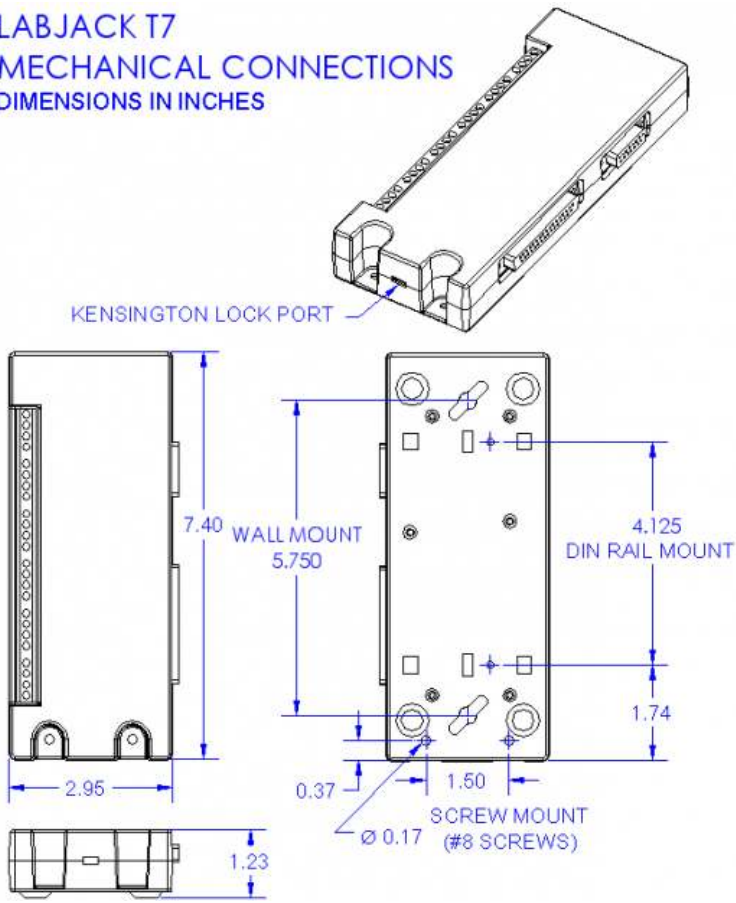
Appendix B - Enclosure and PCB Drawings

See below drawings of the T7.

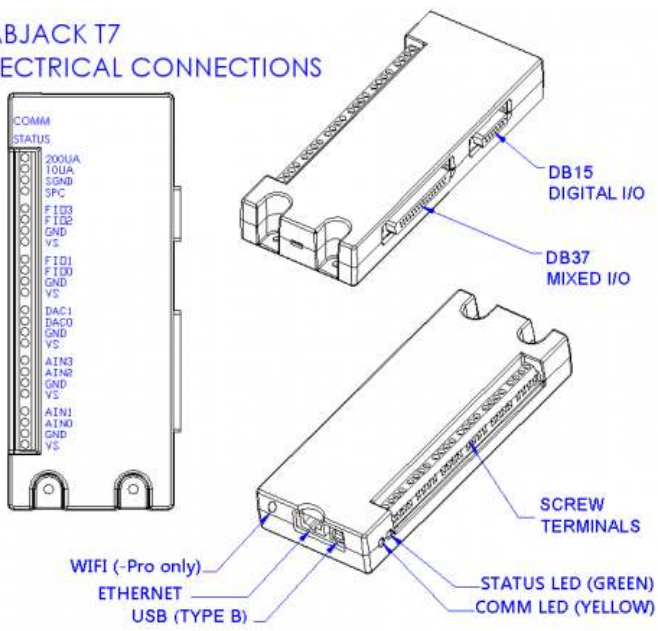
The square holes on the back of the enclosure are for DIN rail mounting adapters (TE Connectivity(formerly Tyco) part #TKAD), or [Newark PN 50F2979](#).

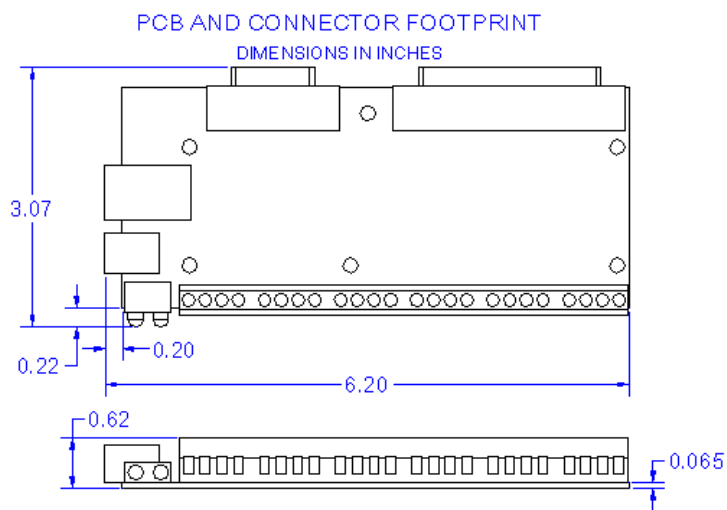
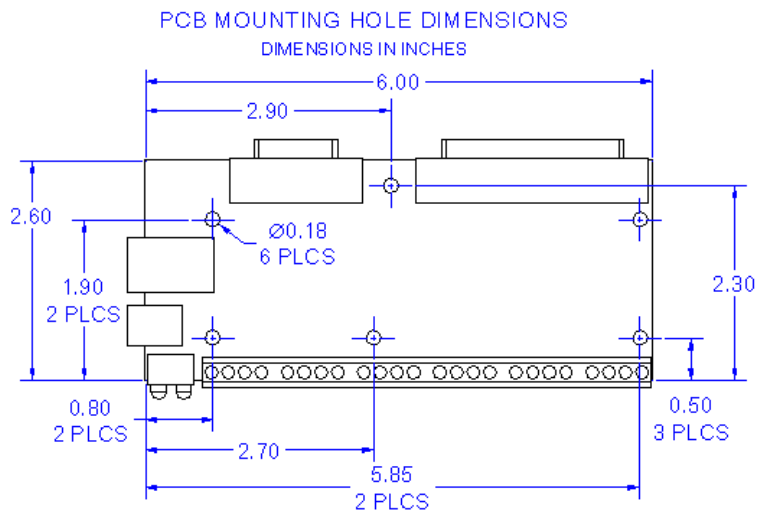
CAD drawings of the T7 enclosure are attached to the bottom of this page. (DWG, DXF, IGES, STEP)

LABJACK T7 MECHANICAL CONNECTIONS DIMENSIONS IN INCHES



LABJACK T7 ELECTRICAL CONNECTIONS



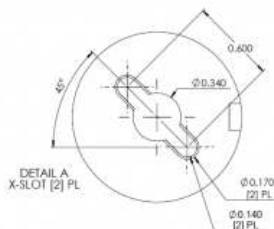
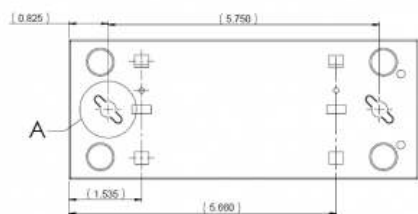


LabJack T7 Enclosure

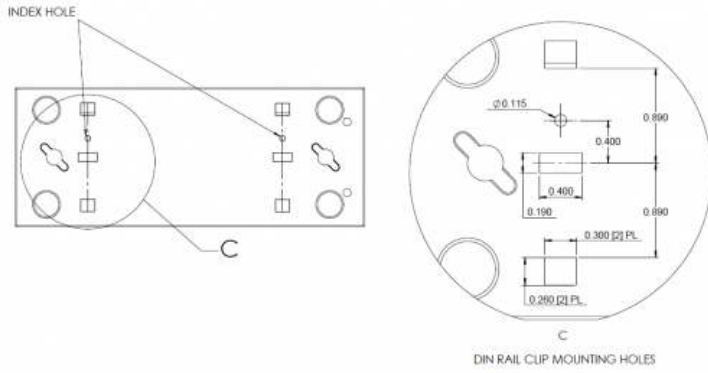
6-2-2012

Note: Dimensions in parentheses are reference only.

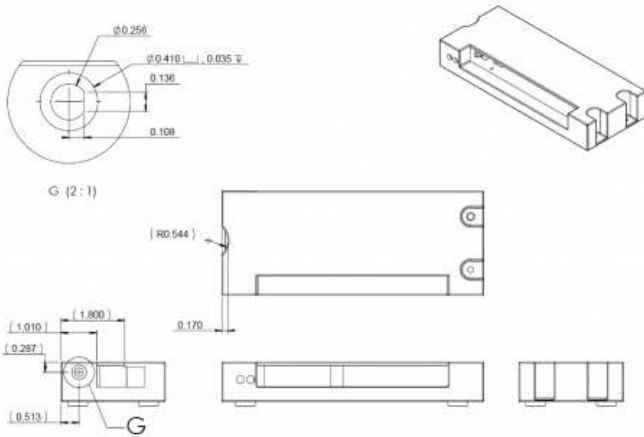
T7 ENCLOSURE BASE



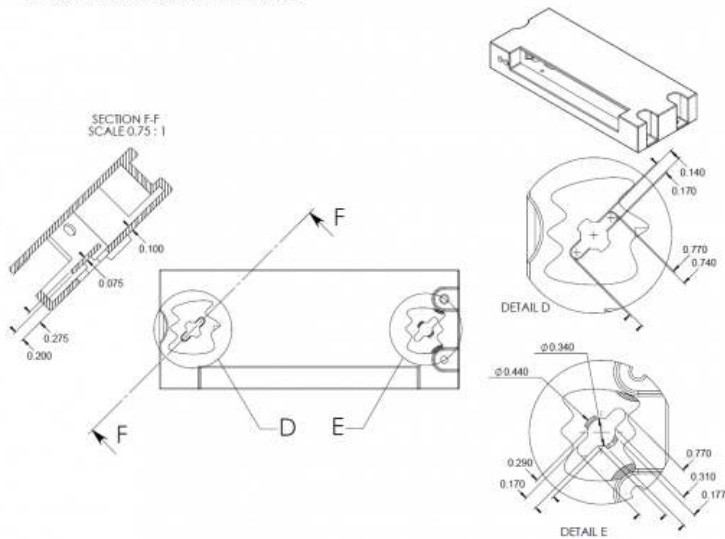
T7 ENCLOSURE BASE - DIN CLIP DETAILS



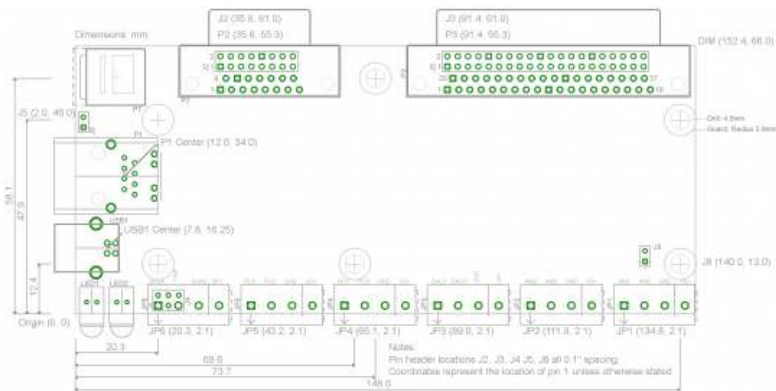
T7 ENCLOSURE TOP CLEARANCE FOR SHROUDED ETHERNET CABLE ANTENNA NUT BOSS DETAILS



T7 ENCLOSURE BASE - SCREW MOUNT DETAILS



T7 OEM PCB Dimensions

**File attachment:**

- [T7 Enclosure DWG](#)
- [T7 Enclosure DXF](#)
- [T7 Enclosure IGS](#)
- [T7 PCB Dimensions](#)
- [T7 Enclosure STEP](#)
- [T7 Pro Enclosure DWG](#)
- [T7 Pro Enclosure DXF](#)
- [T7 Pro Enclosure IGS](#)
- [T7 Pro Enclosure STEP](#)

Appendix C - Firmware Revision History

The latest T7 firmware is listed on the [T7 firmware page](#). You will need the [Kipling](#) software program to load the firmware files onto a T7. Also use Kipling to identify the current WiFi and Firmware versions on your T7.

Change Log

1.0017: Updated I2C so higher numbers correspond to higher speeds. Fixed a bug causing bad data to be read from power settings. Attempted to alleviate an erroneous stream overlap error when starting stream. Updated stream to support O-Stream using modbus addresses. Changed wifi to set the ping and reset timers to 150% normal times after receiving a modbus packet. Updated to USB 2.9. Factory jumper will now reset Ethernet, Power, WiFi, IO, and Watchdog settings. Fixed a bug that prevented stream out from accepting data as an array. Fixed a bug that could cause random error codes when enabling the swdt. AIN14 will now return volts instead of kelvin. Device and ambient temperature registers added.

1.0000: Stream features added and timing calibrated. Stream bug fixes. I2C modbus interface updated. Added z-phase support to quadrature. Digital EF bug fixes and feature additions.

[Click To Expand Full Change Log](#)

Appendix D - Packaging Information

Package Contents:

The normal retail packaged T7 or T7-Pro consists of:

- T7 (-Pro) unit itself in red enclosure
- USB cable (6ft / 1.8m)
- Ethernet Cable (6ft / 1.8m)
- USB 5V power supply
- Screwdriver
- Antenna (T7-Pro only)



Other package details:

There is no software CD included, so an internet connection is required to download software. Go to the T7 Support Homepage (labjack.com/support/t7) to get started.

Contact support@labjack.com for additional information on shipping.

Package size: 10" x 7" x 3"

Package wt: 1.2lb

Appendix E - Datasheet Revision History

Revision E (Feb 2014) Added sections for Flash Memory, SD Card, Stream-Out, and SPI. The internal temperature sensor section now has complete information. Also updated a few descriptions to registers. Moved SPI, I2C, and Digital I/O Extended Features into Digital I/O. Moved many sections into appendices - most notably the specifications page is now Appendix A.

Revision D (Jan 2014) Added Scripting, I2C sections, updated many sections to include register information directly from the constants file. Updated DIO_EF information.

Revision C (Oct 2013) Added calibration constants information. Modified URLs. Updated many links to related support material. Updated DIO information.

Revision B (April 2013) Added many descriptions of Digital I/O extended features. Modified a bunch of URLs.

Revision A (February 2013) Original data sheet for the T7 family of devices.