

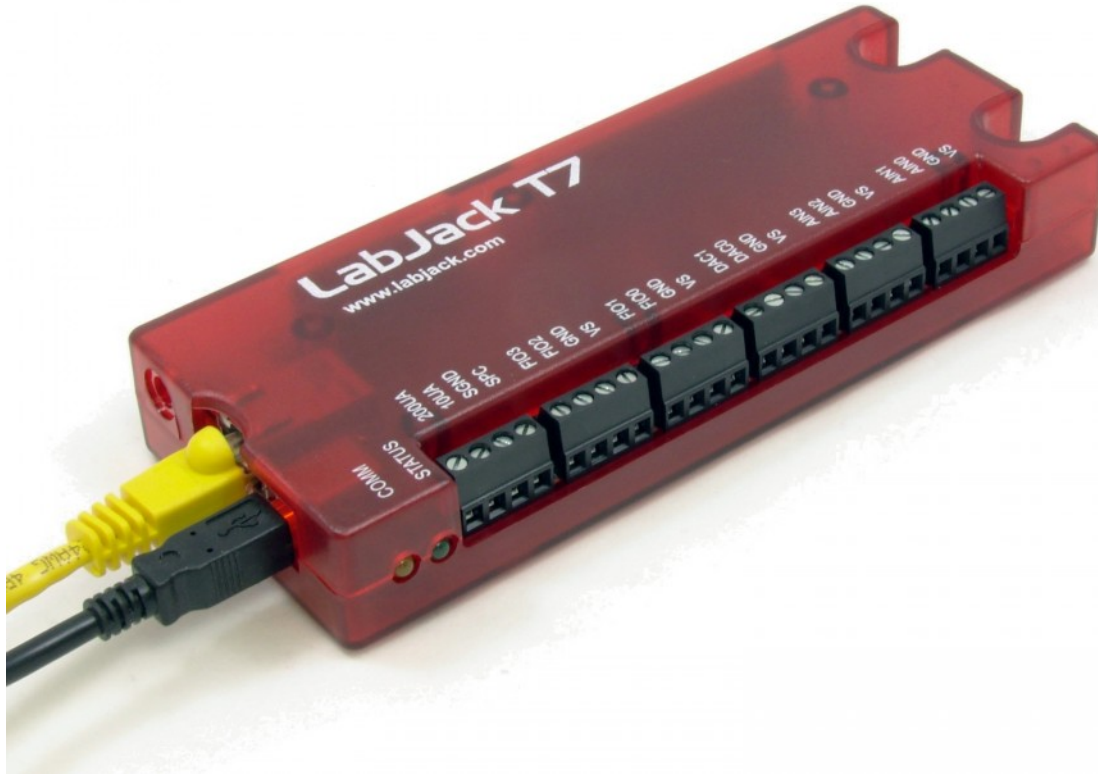
T-Series Datasheet

[Log in](#) or [register](#) to post comments

T7 Series

Stock: In Stock

Price: \$429.00

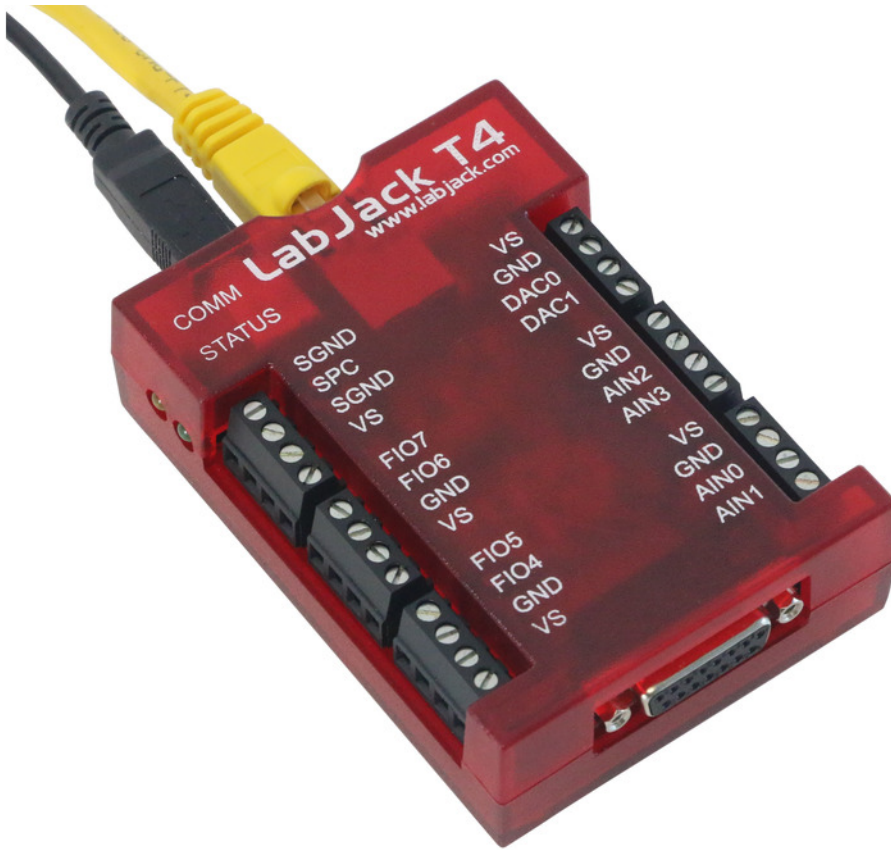


[Click here to order!](#)

T4 Series

Stock: Out of Stock

Price: \$199.00



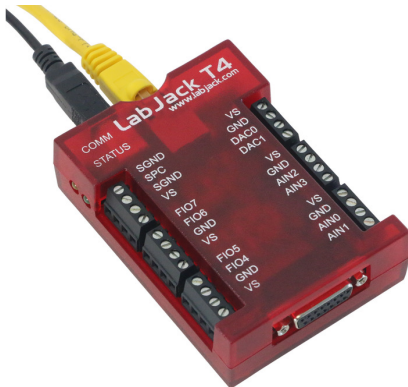
[Click here to order!](#)

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

This datasheet covers all T-series variants:

- T4
- T4-OEM
- T7
- T7-OEM
- T7-PRO
- T7-PRO-OEM

These HTML pages form the complete datasheet, manual, and user's guide for the [T4](#) and [T7](#). Most information in this datasheet applies to all T4 and T7 variants. Information about WiFi and the high-resolution ADC (ResolutionIndex = 9-12) only applies to the T7-Pro variants. There is an [OEM section](#) with information specific to the build of OEM versions.





Reading This Document

Searching The Datasheet

To search this datasheet you can use the search box (available on every page), and to further refine your results include "t-series" or "t-series datasheet" in your search term. For more information see the main [Search Page](#).

Table of Contents Note

Navigating using the Table of Contents

An efficient way to navigate this online document is to use the Table of Contents button to the left.

Offline PDF Note

Offline PDF

If you are looking at a PDF, hardcopy, or other downloaded offline version of this document, realize that it is possibly out-of-date since the original is an online document. Also, this document is designed as online documentation, so the formatting of an offline version might be less than perfect.

To make a PDF of this entire document including all child pages, click "Save as PDF" towards the bottom-right of this 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 "Save as PDF" do a right-click and select "Save link as" or similar. Then wait 20-30 seconds and a dialog box will pop up asking you where to save the PDF. Then you can open it in the real Acrobat Reader rather than embedded in a browser.

Rather than downloading, 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 this document, the 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 page and post a comment. When appropriate we can then immediately add/change content on that page to address the question.

Periodically we use the "Save as PDF" feature to export a PDF and attach it to this page (below).

Preface: Warranty, Liability, Compliance [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

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

Copyright 2017, LabJack Corporation

Warranty

Warranty:

All LabJack hardware is covered by a 5-year limited warranty, covering products and parts against defects in material or workmanship. We will troubleshoot, repair, or replace with like product to make sure you have a device that is operating to specifications.

LabJack products are very robust, but subject to the influence of user connections. The warranty does not apply if damaging mistakes were made in connecting our device, or if inspection reveals obvious signs of improper use, however we will still make a reasonable attempt to fix such devices for free if possible.

The warranty cannot be honored on discontinued products if replacements are unavailable and repair is not reasonably possible.

LabJack has taken great care of our customers since we sold our first device in 2001, and your satisfaction is our highest priority. If you suspect a problem with a device contact us at support@labjack.com.

LabJack is not liable for any losses, expenses or damages beyond the LabJack device itself. See our [Limitation of Liability](#) for more details.

Limitation of Liability

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.

T-Series Compliance

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 200, Lakewood, CO 80227, USA

Declares that these products

Product Name: LabJack T4
Model Number: LJ T4

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

conform to the following Product Specifications:

EMC Directive: 2004/104/EEC

EN 55011 Class A
EN 61326-1: General Requirements

and is marked with CE

RoHS2:

The T4 and T7 (-Pro) are RoHS compliant to Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

REACH:

The T4 and T7 (-Pro) are REACH compliant. REACH Product Compliance Program has been implemented in accordance with Regulation No. 1907/2006 of the European Parliament and the Council of 18 December 2006. LabJack Corporation does not currently have a direct REACH obligation to pre-register substances. LabJack's REACH Product Compliance is determined by a certification from our supply chain. LabJack products are deemed to be REACH compliant when they do not contain Substances of Very High Concern (SVHCs) beyond the specified concentration limits of less than 0.1% by weight as outlined in REACH 1907/2006/EU regulation.

CFM:

LabJack Corporation does not knowingly use these minerals or any by-products, as specified by the Conflict Minerals Trade Act.

Compliance Information for the WiFi module in the T7-Pro and T7-Pro-OEM

FCC:

Contains FCC ID: T9J-RN171

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy, and if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

To satisfy FCC RF Exposure requirements for mobile and base station transmission devices, a separation distance of 20 cm or more should be maintained between the antenna of this device and persons during operation. To ensure compliance, operation at closer than this distance is not recommended. The antenna(s) used for this transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

Canada:

Contains transmitter module IC: 6514A-RN171

This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes: (1) l'appareil ne doit pas produire de brouillage, et (2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

Under Industry Canada regulations, this radio transmitter may only operate using an antenna of a type and maximum (or lesser) gain approved for the transmitter by Industry Canada. To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (e.i.r.p.) is not more than that necessary for successful communication.

Conformément à la réglementation d'Industrie Canada, le présent émetteur radio peut fonctionner avec une antenne d'un type et d'un gain maximal (ou inférieur) approuvé pour l'émetteur par Industrie Canada. Dans le but de réduire les risques de brouillage radioélectrique à l'intention des autres utilisateurs, il faut choisir le type d'antenne et son gain de sorte que la puissance isotrope rayonnée équivalente (p.i.r.e.) ne dépasse pas l'intensité nécessaire à l'établissement d'une communication satisfaisante.

This radio transmitter (identify the device by certification number, or model number if Category II) has been approved by Industry Canada to operate with the antenna types listed below with the maximum permissible gain and required antenna impedance for each antenna type indicated. Antenna types not included in this list, having a gain greater than the maximum gain indicated for that type, are strictly prohibited for use with this device.

Conformément à la réglementation d'Industrie Canada, le présent émetteur radio peut fonctionner avec une antenne d'un type et d'un gain maximal (ou inférieur) approuvé pour l'émetteur par Industrie Canada. Dans le but de réduire les risques de brouillage radioélectrique à l'intention des autres utilisateurs, il faut choisir le type d'antenne et son gain de sorte que la puissance isotrope rayonnée équivalente (p.i.r.e.) ne dépasse pas l'intensité nécessaire à l'établissement d'une communication satisfaisante.

1.0 Device Overview [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

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

- T4
- T4-OEM
- T7
- T7-Pro
- T7-OEM
- T7-Pro-OEM

This family introduces a new line of high-quality analog and Ethernet data acquisition hardware combined 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 [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T4:

High Voltage Analog Inputs

- 4 dedicated high voltage analog inputs ($\pm 10V$, 12-bit resolution)
- Configurable resolution settings

Flexible I/O:

- 8 configurable low voltage analog inputs (0-2.5V, 12-bit resolution) that can function as digital I/O lines

Digital I/O:

- 8 dedicated digital I/O lines (EIO4-EIO7 and CIO0-CIO3)

Analog Outputs:

- 2 Analog Outputs (10-bit, 0-5 volts)
- Additional analog outputs are possible via the LJTick-DAC

T7:

Analog I/O

- 14 Analog Inputs (16-18+ Bits Depending on Speed), expand to 84 with Mux80
- 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, $\sim 0-5$ Volts)

Digital I/O

- 23 Digital I/O
- Supports SPI, I2C, 1-Wire 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
- Software counter
- Software counter w/ debounce
- Quadrature Input
- Easy Frequency Input

Analog Input Extended Features

- User Defined Slope & Offset
- Thermocouple type E, J, K, R, T, and C calculations
- RTDs
- Thermistors

Other highlights

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

Fixed Current Outputs

- 200 μA
- 10 μA

1.2 Family Variants Info [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T4 vs T7

The two devices are similar due to them being compatible with the [LJM](#) driver and their support/use of Modbus TCP.

T4 characteristics that differ from the T7:

- 12-bit effective resolution on HV and LV lines.
- AIN (0-3) are high voltage ($\pm 10V$).
- Flexible I/O lines: The T4 I/O lines FIO (4-7) and EIO (0-3) are software-configurable to be either low voltage analog inputs (0-2.5V) or digital I/O lines (3.3V logic level).

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
- Battery-Backed Real Time Clock to assist Scripts.

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

T7-OEM and T7-Pro-OEM

There are also OEM versions 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, allowing customization as needed. See [22.0 OEM Versions](#) for details.

2.0 Installation [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

1. Install the LabJack [software and driver bundle](#) based on your operating system.
2. Connect the T4 or T7 to the local computer via USB.
3. Proceed through any steps to add new hardware.
4. If using Windows, Mac or Linux, open Kipling (installed with the software and driver bundle).
5. Use the dashboard in Kipling to view analog inputs, digital I/O, DAC outputs, etc.
6. Go to the relevant quickstart tutorial:
 - [T4 quickstart tutorial](#)
 - [T7 quickstart tutorial](#)

3.0 Communication [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T-series devices communicate through a protocol named Modbus TCP, which is used via USB, Ethernet, and WiFi (T7-Pro only). "Modbus TCP" is commonly shortened to "Modbus" in this document.

Modbus TCP is a protocol where values are written to and/or read from Modbus registers. Any given Modbus register is readable, writable, or both.

All T-series device configurations and data is read from and/or written to Modbus register(s). Thus, the process for reading the serial number, an analog input, or a waveform is functionally the same, you simply provide a different address.

Communication Options

The software used to communicate with a T-series device depends on what tasks need to be performed.

Applications

Applications are appropriate for common and simple tasks.

LabJack [Kipling](#) provides a graphical interface for many types of device configuration. It also provides a [Register Matrix](#) which can read or write arbitrary register values.

LabJack [LJLogM](#) periodically samples and logs data.

LabJack [LJStreamM](#) streams data at much higher sampling rates.

High-level [LJM library](#)

Programming with the LJM library is appropriate for custom, complex, and automated tasks.

LJM is a cross-platform library which allows users to access registers by name, such as "AIN4" for analog input 4. With [example code](#) in over a dozen different programming languages, it is possible to integrate the T4 and T7 into a variety of existing software frameworks.

Example workflow:

1. [Open](#) a connection to the T4/T7.
2. [Read from and write to Modbus registers](#).
3. [Close](#) the connection.

Direct Modbus TCP, Clients

T-series devices are Modbus TCP servers. Any software capable of acting as a Modbus TCP client can read from and write to a Modbus TCP server. For example, all software we know of that describes itself as "SCADA" is capable of being a Modbus TCP client and can talk to T-series devices.

It is straightforward to integrate a T-series device, over Ethernet or WiFi (not USB), into standard commercial off-the-shelf (COTS) Modbus client platforms. People who already use Modbus software will find this option convenient. Some COTS Modbus clients are very powerful, and will save users the time and money required to develop their own software.

A Modbus TCP client can read/write any single-value numeric register without any driver software or libraries from us. More complex registers such as strings and arrays will be difficult if not impossible to use from a standard Modbus client, which notably prohibits stream mode and serial protocols. Custom Modbus clients, however, can realize all functionality.

Typical workflow:

1. Configure the power-up-default registers on the T4 or T7 using the [Kipling](#) software program. Change [Ethernet/WiFi](#) IP settings, any relevant analog input settings, etc. Note that '..._DEFAULT' registers indicate that they are power-up-defaults.
2. Open COTS Modbus client program.
3. Specify the Modbus registers by address, such as 6, for AIN3. Find applicable registers with the [register look-up tool](#) (Modbus map), or by referencing the datasheet etc.
4. See data directly from the T4 or T7 in COTS software.

For more details, see the [Software Options](#) page.

Communication Speed Considerations

There are two alternate methods for data transfer to occur.

- Command-response offers the lowest latency.
- Streaming offers the highest data throughput.

The LJM library simplifies both command-response and stream mode.

COTS Direct Modbus software uses command-response and is unlikely to be capable of stream mode.

Command-Response

This is the default mode for communication with a device. It is the simplest communication mode.

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

Command-response mode is generally best for minimum-latency applications such as feedback control. Latency, in this context, means 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 command-response data rates.

Stream Mode

Stream mode is generally best for maximum-throughput applications. However, streaming is usually 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.

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. Since 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. For the T7-Pro, 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 samples. 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 [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Modbus Map Tool

Device:

Tags:
All Tags
AIN
AIN_EF
ASYNCH
CONFIG
CORE
DAC
DIO
DIO_EF
ETHERNET

Expand addresses:

An error has occurred.

The filter and search tool above displays information about the Modbus registers of T-series devices.

- Name: The string name that can be used with the LJM library to access each register.
- Address: The starting address of each register, which can be used through LJM or with direct Modbus.
- Details: A quick description of the register.
- Type: Specifies the datatype, which specifies how many registers each value uses.
- Access: Each register is readable, writable, or both.
- Tags: Used to associate registers with particular functionality. Useful for filtering.

For the U3, U6 and UE9, see the deprecated Modbus system called [UD Modbus](#).

For a printer-friendly version, see the [printable Modbus map](#).

Also On This Page

- [0-Based Addressing](#)
- [Single Overlapping Map of Addresses from 0-65535](#)
- [Big-Endian](#)
- [Data Type Constants](#)
- [Sequential Addresses](#)
- [ljm_constants.json](#)

Protocol

Modbus protocol is described on the [Protocol Details](#) page.

0-Based Addressing

The addresses defined in the map above are the same addresses in the actual Modbus packet, and range from 0 to 65535.

Some clients subtract 1 from all addresses. You tell the client you want to read address 2000, but the client puts 1999 in the actual Modbus packet. That means if you want to read Modbus address 2000, you have to tell the client 2001. We use 0-65535 addressing everywhere, so if you want to read an address we document as 2000, then 2000 should be in the Modbus packet.

Single Overlapping Map of Addresses from 0-65535

We have a single map of addresses from 0 to 65535. Any type of register can be located anywhere in that range regardless of data type or whether it is read-only, write-only, or read-write.

Some client software uses addresses written as 4xxxx. In this case, the 4 is a special code that means to use the Modbus read function 0x03 and the xxxx is an

address from 0-9999 that might additionally have 1 subtracted before being put in the Modbus packet. The magic number of 40000 (or 40001) has no mention in the Modbus spec that we can find. What this means in terms of how to talk to the LabJack depends on what exactly the client is doing, but if you want to read from an address we have defined as x (0-65535), then x should be the address in the Modbus packet sent out over TCP.

Big-Endian

Modbus is specified as big-endian, which means the most significant value is at the lowest address. With a read of a 16-bit (single register) value, the 1st byte returned is the MSB (most significant byte) and the 2nd byte returned is the LSB (least significant byte). With a read of a 32-bit (2 register) value, the value is returned MSW then LSW, where each word is returned MSB then LSB, so the 4 bytes come in order from most significant to least significant.

We have seen some clients that expect Modbus to be implemented with big-endian bytes but with the least significant word before the most significant word. In other words, the client software flips the order of the words within a 32-bit value. For example, a read of TEST (address 55100) should return 0x00112233, but the client returns 0x22330011.

Data Type Constants

Type	Integer Value
LJM_UINT16	0
LJM_UINT32	1
LJM_INT32	2
LJM_FLOAT32	3
LJM_BYTE	99
LJM_STRING	98

Sequential Addresses

Many registers are sequentially addressed. The Modbus Map gives you the starting address for the first register, and then—depending on whether the data type is 16-bits or 32-bits—you increment the address by 1 or 2 to get the next value:

```
Address = StartingAddress + 1*Channel# (UINT16)
```

```
Address = StartingAddress + 2*Channel# (UINT32, INT32, FLOAT32)
```

Note that the term "register" is used 2 different ways throughout documentation:

- A "register" is a location that has a value you might want to read or write (e.g. AIN0 or DAC0).
- The term "Modbus register" generally refers to the Modbus use of the term, which is a 16-bit value pointed to by an address of 0-65535.

Therefore, most "registers" consist of 1 or 2 "Modbus registers".

For example, the first entry in the Modbus Map has the name AIN#(0:254), which is shorthand notation for 255 registers named AIN0, AIN1, AIN2, ..., AIN254. The AIN# data type is FLOAT32, so each value needs 2 Modbus registers, and thus the address for a given analog input is `channel*2`.

ljm_constants.json

LabJack distributes a [constants file](#) called `ljm_constants.json` that defines information about the Modbus register map. The filter and search tool above pulls data from that JSON file.

The [ljm_constants GitHub repository](#) contains up-to-date text versions of the Modbus register map:

- **JSON** - [ljm_constants.json](#)
- **C/C++ header** - [LabJackMModbusMap.h](#)

3.1.1 Buffer Registers [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

Most registers are written / read by address, but other registers are a special kind of register known as a Buffer Register. Buffer Registers are for cases when multiple values must be written / read, but the number of values are able to change. Buffer Registers produce multiple values when being read from and consume all values being written. Buffer registers allow users to write a sequence of values to a single Modbus address. Typically buffer registers have a companion `_SIZE` register that defines how many sequential values are about to be sent to or read from a buffer register. Some would call them array registers, because you basically define the array size, and then pass the array of data into a single Modbus address.

For example, consider the difference between AIN0 and FILE_IO_PATH_READ:

Normal register:

- AIN0 is at address 0 and is followed by AIN1 at address 2
- AIN0 is a normal register

- Reading an array of 4 registers starting at address 0 would read 2 registers from AIN0 and 2 registers AIN1 (AIN values are FLOAT32, which each consist of 2 registers)

Buffer register:

- FILE_IO_PATH_READ is at address 60652 and is followed by FILE_IO_WRITE at address 60654
- FILE_IO_PATH_READ is a Buffer Register
- Reading an array of 4 registers starting at address 60652 would read 4 registers from FILE_IO_PATH_READ. FILE_IO_WRITE would not be read.
- Note that users would first designate that 8 bytes are about to be read by writing a value of 8 to FILE_IO_PATH_READ_LEN_BYTES.

In practice, the important differences are:

1. You don't need to know what registers follow a Buffer Register, you can simply write / read without worrying about colliding with other registers
2. You can only write / read values sequentially. E.g. you cannot modify previously written values.
3. Define how much data to send/receive to/from the buffer register using the associated _NUM_BYTES, or _SIZE, or _LEN register.
4. Often it is necessary to complete the transaction with an action register, such as _GO, or _OPEN, or _ENABLE.

Buffer Registers, and their size definitions:

Serial Comm Systems

- ASYNCH_DATA_RX
- ASYNCH_DATA_TX
- ASYNCH_NUM_BYTES_RX
- ASYNCH_NUM_BYTES_TX
- I2C_DATA_RX
- I2C_DATA_TX
- I2C_NUM_BYTES_RX
- I2C_NUM_BYTES_TX
- ONEWIRE_DATA_RX
- ONEWIRE_DATA_TX
- ONEWIRE_NUM_BYTES_RX
- ONEWIRE_NUM_BYTES_TX
- SPI_DATA_RX
- SPI_DATA_TX
- SPI_NUM_BYTES

File IO System

- FILE_IO_PATH_READ
- FILE_IO_PATH_WRITE
- FILE_IO_PATH_READ_LEN_BYTES
- FILE_IO_PATH_WRITE_LEN_BYTES
- FILE_IO_READ
- FILE_IO_WRITE
- FILE_IO_SIZE_BYTES

Lua Scripts/Debug Info

- LUA_SOURCE_WRITE
- LUA_SOURCE_SIZE
- LUA_DEBUG_DATA
- LUA_DEBUG_NUM_BYTES

Stream Out System

- STREAM_OUT#(0:3)_BUFFER_F32
- STREAM_OUT#(0:3)_BUFFER_U16
- STREAM_OUT#(0:3)_BUFFER_U32
- STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES

User RAM FIFOs

- USER_RAM_FIFO#(0:3)_DATA_F32
- USER_RAM_FIFO#(0:3)_DATA_I32
- USER_RAM_FIFO#(0:3)_DATA_U16
- USER_RAM_FIFO#(0:3)_DATA_U32
- USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES

WIFI

- WIFI_SCAN_DATA
- WIFI_SCAN_NUM_BYTES

Internal Flash

- INTERNAL_FLASH_READ
- INTERNAL_FLASH_WRITE

The above buffer registers can be identified in the [LJM constants file](#) as buffer registers by the **"isBuffer":true** designation.

LJM

When using LJM, the Array functions are useful when reading from a buffer ([LJM_eReadAddressArray](#) or [LJM_eReadNameArray](#)) or writing to a buffer ([LJM_eWriteAddressArray](#) or [LJM_eWriteNameArray](#)).

3.2 Stream Mode [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Stream Mode Overview

Streaming is a fast data input mode. It is more complicated than command-response mode, so it requires more configuration. Using stream is simplified by the [LJM stream functions](#); to stream without them, see [3.2.2 Low-Level Streaming](#).

For a given stream session, a list of channels/addresses are sampled as input to the device. This list of channels (known as a scan list) is input, as quickly as possible, immediately after a clock pulse. Stream clock pulses are hardware-timed at a constant scan rate. By default, a stream session begins scanning immediately after being started and continuously scans until stopped.

Stream can also output data.

Stream sessions can be configured to collect a limited number of scans. (See [Burst Stream](#))

T7 only:

The T7 supports some advanced stream features:

- Stream sessions can be configured to delay scanning until after the T7 detects a trigger pulse.
- Stream clock pulses can also be read externally at either a constant or a variable rate.

On This Page

- [Maximum Stream Speed](#)
- [Stream-In and/or Stream-out](#)
- [Streamable Registers](#)
- [16-bit or 32-bit Data](#)
- [Configuring AIN for Stream](#)
- [Stream Timing](#)
- [Channel-to-Channel Timing](#)
- [Burst Stream](#)
- [Externally-Clocked Stream - T7 Only](#)
- [Triggered Stream - T7 Only](#)

Maximum Stream Speed

T4 Max Sample Rate: 40 ksamples/second

The T4 max sample rate is 40 ksamples/second. This is achievable for any single-address stream, but for a multi-address stream this is only true when resolution index = 0 or 1.

T7 Max Sample Rate: 100 ksamples/second

The T7 max sample rate is 100 ksamples/second. This is achievable for any single-address stream, but for a multi-address stream this is only true when resolution index = 0 or 1 and when range = +/-10V for all analog inputs.

The **max scan rate** depends on how many addresses you are sampling per scan:

- Address => The Modbus address of one channel. (See [Streamable Registers](#), below.)
- Sample => A reading from one address.
- Scan => One reading from all addresses in the scan list.
- $SampleRate = NumAddresses * ScanRate$

Examples:

- For a T4 streaming 4 channels at resolution index=0, the max scan rate is 10 kscans/second (calculated from 40 ksamples/second divided by 4).
- For a T7 streaming 5 channels at resolution index=0 and all at range=+-10V, the max scan rate is 20 kscans/second (calculated from 100 ksamples/second divided by 5).

Ethernet provides the best throughput: Ethernet is capable of the fastest stream rates. USB is typically a little slower than Ethernet, and WiFi is much slower. For more information on speeds, see the [Data Rates Appendix](#).

Stream-In and/or Stream-Out

There are three input/output combinations of stream mode:

Stream-in: The device collects data and streams it to the host.

Stream-out: The device does not collect data but streams it out. (See [3.2.1 Stream-Out](#))

Stream-in-out: The device collects data and streams it to the host. It also streams data out.

The stream channels determine which of these modes are used. Streamable channels may be either stream-in or stream-out.

Streamable Registers

The [Modbus map](#) shows which registers can be streamed (by expanding the "details" area). Input registers that can be streamed include:

	For more information
AIN#	See 14.0 Analog Inputs .
FIO_STATE	See 13.0 Digital I/O .
EIO_STATE	See 13.0 Digital I/O .
CIO_STATE	See 13.0 Digital I/O .
MIO_STATE	See 13.0 Digital I/O .
FIO_EIO_STATE	See 13.0 Digital I/O .
EIO_CIO_STATE	See 13.0 Digital I/O .
DIO#(0:22)_EF_READ_A	See 13.2 DIO Extended Features .
DIO#(0:22)_EF_READ_A_AND_RESET	See 13.2 DIO Extended Features .
DIO#(0:22)_EF_READ_B	See 13.2 DIO Extended Features .
CORE_TIMER	See 4.0 Hardware Overview .
SYSTEM_TIMER_20HZ	See 4.0 Hardware Overview .
STREAM_DATA_CAPTURE_16	See below .

For stream-out registers, see [3.2.1 Stream-Out](#).

16-bit or 32-bit Data

Stream data is transferred as 16-bit values, but 32-bit data can be captured by using STREAM_DATA_CAPTURE_16.

16-bit: In the normal case of an analog input such as AIN0, the 16-bit binary value is actually what is transferred and LJM converts it to a float on the host using the calibration constants that LJM reads before starting the stream.

32-bit: Some streamable registers (e.g. DIO4_EF_READ_A) have 32-bit data. When streaming a register that produces 32-bit data, the lower 16 bits (LSW) will be returned and the upper 16 bits (MSW) will be saved in STREAM_DATA_CAPTURE_16. To get the full 32-bit value, add STREAM_DATA_CAPTURE_16 to the stream scan list after any applicable 32-bit register, then combine the two values in software (LSW + 65536*MSW). Note that STREAM_DATA_CAPTURE_16 may be placed in multiple locations in the scan list.

Configuring AIN for Stream

STREAM_SETTLING_US and STREAM_RESOLUTION_INDEX override the normal [AIN configuration](#) settling and resolution registers.

Name	Start Address	Type	Access
STREAM_SETTLING_US	4008	FLOAT32	R/W
STREAM_RESOLUTION_INDEX	4010	UINT32	R/W

STREAM_SETTLING_US - Address: 4008

Time in microseconds to allow signals to settle after switching the mux. Does not apply to the 1st channel in the scan list, as that settling is controlled by scan rate (the time from the last channel until the start of the next scan). Default=0. When set to less than 1, automatic settling will be used. The automatic settling behavior varies by device.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - The T7 will select the settling time based on resolution and gain settings. When the sample rate is above 60 kHz, settling time will be set to ~5 us. Max is 4400.

- T4:
 - The T4 will default to 10 us. When the scan rate is above 20 kHz, settling time will be set to 5 us.

STREAM_RESOLUTION_INDEX - Address: 4010

The resolution index for stream readings. A larger resolution index generally results in lower noise and longer sample times.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Valid values: 0-8. Default value of 0 corresponds to an index of 1.
- T4:
 - Valid values: 0-5. Default value of 0 corresponds to an index of 1.

The normal AIN configuration registers for range and negative channel still apply to stream.

T7 only: Stream mode is not supported on the hi-res converter. (Resolution indices 9-12 are not supported in stream.)

Stream Timing

When using LJM, there are three ways that stream can be too slow:

1. Sample rate is too high
2. Device buffer overflow
3. LJM buffer overflow

Sample rate is too high: When the sample rate is too high, it causes a `STREAM_SCAN_OVERLAP` error and stream is terminated.

Scans are triggered by hardware interrupts. If a scan begins and the previous scan has not finished, the device stops streaming and returns a `STREAM_SCAN_OVERLAP` error (errorcode 2942), which LJM returns immediately upon the next call to `LJM_eStreamRead`.

Device buffer overflow: When the device buffer overflows, LJM inserts a dummy sample (with the value -9999.0) in place of each skipped sample, or it causes a `STREAM_AUTO_RECOVER_END_OVERFLOW` error and stream is terminated.

As samples are collected, they are placed in a FIFO buffer on the device until retrieved by the host. The size of the buffer is variable and can be set to a maximum of 32764 bytes. Write to `STREAM_BUFFER_SIZE_BYTES` to set the buffer size.

Name	Start Address	Type	Access
<code>STREAM_BUFFER_SIZE_BYTES</code>	4012	UINT32	R/W

STREAM_BUFFER_SIZE_BYTES - Address: 4012

Size of the stream data buffer in bytes. A value of 0 equates to the default value. Must be a power of 2. Size in samples is `STREAM_BUFFER_SIZE_BYTES/2`. Size in scans is `(STREAM_BUFFER_SIZE_BYTES/2)/STREAM_NUM_ADDRESSES`. Changes while stream is running do not affect the currently running stream.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T4:
 - Max size is 32768. Default size is 8192.
- T7:
 - Max size is 32768. Default size is 4096.

If the device buffer overflows, the device will continue streaming but will discard data until the buffer is emptied, after which data will be stored in the buffer again. The device keeps track of how many scans are discarded and reports that value. Based on the number of scans discarded, the LJM library adds the proper number of dummy samples (with the value -9999.0) such that the correct timing is maintained. This will only work if the first channel in the scan is an analog channel.

If the device buffer overflows for too much time, a `STREAM_AUTO_RECOVER_END_OVERFLOW` error occurs and stream is terminated.

If the device buffer is overflowing, see the [LJM stream help](#) page for some mitigation strategies.

LJM buffer overflow: When the LJM buffer overflows, it causes a `LJME_LJM_BUFFER_FULL` error and stream is terminated.

LJM reads samples from the device buffer and buffers them internally. LJM reads these samples in an internal thread, regardless of what your code does. LJM's buffer can run out of space if it is not read often enough using `LJM_eStreamRead`, so make sure the `LJMScanBacklog` parameter does not continually increase.

`LJM_eStreamRead` blocks until enough data is read from the device, so your code does not need to perform waits.

If the LJM buffer is overflowing, see the [LJM stream help](#) page for some mitigation strategies.

Channel-to-Channel Timing

Channels in a scan list are input or output as quickly as possible after the start of a scan, in the order of the scan list.

Timing pulses are generated on `SPC` so that the channel-to-channel timing can be measured. Pulses on `SPC` are as follows:

- Falling edge at the start of a scan.
- Rising edge at the start of a sample.
- Falling edge at the end of a sample.

- Rising edge at the end of a scan.

Burst Stream

Burst stream is when stream collects a pre-determined number of scans, then stops. To set the stream burst size, write to STREAM_NUM_SCANS:

Name	Start Address	Type	Access
STREAM_NUM_SCANS	4020	UINT32	R/W

STREAM_NUM_SCANS - Address: 4020

The number of scans to run before automatically stopping (stream-burst). 0 = run continuously. Limit for STREAM_NUM_SCANS is $2^{32}-1$, but if the host is not reading data as fast as it is acquired you also need to consider STREAM_BUFFER_SIZE_BYTES.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

The LJM library collects burst stream data with the [StreamBurst\(\)](#) function.

It may be beneficial to set STREAM_BUFFER_SIZE_BYTES to a large value for fast burst stream. See above for details about STREAM_BUFFER_SIZE_BYTES.

T7-Pro only: Burst stream is well-suited for WiFi connections, because WiFi has a lower throughput than other connection types.

Externally Clocked Stream - T7 Only

Externally-clocked stream allows T-series devices to stream from external pulses. It also allows for variable stream scan rates.

Clock Source: The scan rate is generated from the internal crystal oscillator. Alternatively, the scan rate can be a division of an external clock provided on CIO3.

Name	Start Address	Type	Access
STREAM_CLOCK_SOURCE	4014	UINT32	R/W

STREAM_CLOCK_SOURCE - Address: 4014

Controls which clock source will be used to run the main stream clock. 0 = Internal crystal, 2 = External clock source on CIO3. Rising edges will increment a counter and trigger a stream scan after the number of edges specified in STREAM_EXTERNAL_CLOCK_DIVISOR. T7 will expect external stream clock on CIO3. All other values reserved.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0110

To subdivide the external clock pulses for a slower scan rate, use STREAM_EXTERNAL_CLOCK_DIVISOR.

Name	Start Address	Type	Access
STREAM_EXTERNAL_CLOCK_DIVISOR	4022	UINT32	R/W

STREAM_EXTERNAL_CLOCK_DIVISOR - Address: 4022

The number of pulses per stream scan when using an external clock.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

To use externally clocked stream with LJM, see the [externally clocked stream](#) section of the LJM User's Guide.

Triggered Stream - T7 Only

T7 minimum firmware 1.0186

Stream can be configured to start scanning when a trigger is detected. Trigger sources are DIO_EF modes:

- [Frequency In](#)
- [Pulse Width In](#)
- [Conditional Reset](#)

Frequency In and Conditional Reset allow you to select rising or falling edges and Pulse Width In will trigger from either edge.

See [Appendix A](#) for hysteresis voltage information.

Configuring stream to use a trigger requires setting up a DIO_EF and adding the STREAM_TRIGGER_INDEX register to normal stream configuration.

Name	Start Address	Type	Access
STREAM_TRIGGER_INDEX	4024	UINT32	R/W

STREAM_TRIGGER_INDEX - Address: 4024

Controls when stream scanning will start. 0 = Start when stream is enabled, 2000 = Start when DIO_EF0 detects an edge, 2001 = Start when DIO_EF1 detects an edge. See the stream documentation for all supported values.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0186

STREAM_TRIGGER_INDEX (address 4024):

- 0 = No trigger. Stream will start when Enabled.
- 2000 = DIO_EF0 will start stream.
- 2001 = DIO_EF1 will start stream.
- 2002 = DIO_EF2 will start stream.
- 2003 = DIO_EF3 will start stream.
- 2006 = DIO_EF6 will start stream.
- 2007 = DIO_EF7 will start stream.

To use triggered stream with LJM, see the [triggered stream](#) section of the LJM User's Guide.

3.2.1 Stream-Out (Advanced) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Stream-Out (Advanced) Overview

Stream-out is a set of streamable registers that move data from a buffer to an output. The output can be digital I/O (DIO) or a DAC. The buffer can be read linearly to generate an irregular waveform or be read in a looping mode to generate a periodic waveform.

A T-series device can output up to 4 waveforms using stream out.

Alternate waveform generation techniques are described in the [Waveform Generation](#) App Note.

Performing Stream-Out

For each waveform being streamed out:

1. Choose which target channel will output the waveform
2. Configure stream-out
 1. STREAM_OUT#_TARGET
 2. STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES
 3. STREAM_OUT#(0:3)_ENABLE
3. Update the stream-out buffer
 1. STREAM_OUT#(0:3)_LOOP_NUM_VALUES
 2. STREAM_OUT#(0:3)_BUFFER_F32 or STREAM_OUT#(0:3)_BUFFER_U16
 3. STREAM_OUT#(0:3)_SET_LOOP
4. Start stream with STREAM_OUT#(0:3) in the scan list
5. Stream loop: read and update buffer as needed
6. Stop stream

Executing stream-out for multiple output waveforms is a matter of performing the above steps in the order above and using corresponding STREAM_OUT#(0:3) addresses in the scan list.

1. Target Selection

The following target list represents the I/O on the device that can be configured to output a waveform using stream out. The list includes the analog and digital output lines.

- DAC0
- DAC1
- FIO_STATE
- FIO_DIRECTION
- EIO_STATE
- EIO_DIRECTION
- CIO_STATE
- CIO_DIRECTION
- MIO_STATE
- MIO_DIRECTION

2. Configure Stream-Out

Configuration will set the buffer size and target. The target specifies which physical I/O to use. Data in the buffer will be output onto the target I/O as a generated waveform.

Stream-Out Configuration			
Name	Start Address	Type	Access
STREAM_OUT#(0:3)_TARGET	4040	UINT32	R/W
STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES	4050	UINT32	R/W
STREAM_OUT#(0:3)_ENABLE	4090	UINT32	R/W

STREAM_OUT#(0:3)_TARGET - Starting Address: 4040

Channel that data will be written to. Before writing data to `_BUFFER_###`, you must write to `_TARGET` so the device knows how to interpret and store values.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0012

Expanded Names	Addresses
STREAM_OUT0_TARGET, STREAM_OUT1_TARGET, STREAM_OUT2_TARGET, STREAM_OUT3_TARGET Show All	4040, 4042, 4044, 4046 Show All

STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES - Starting Address: 4050

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written. Before writing data to `_BUFFER_###`, you must write to `_BUFFER_ALLOCATE_NUM_BYTES` to allocate RAM for the data. Max is 16384.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 0.9300

Expanded Names	Addresses
STREAM_OUT0_BUFFER_ALLOCATE_NUM_BYTES, STREAM_OUT1_BUFFER_ALLOCATE_NUM_BYTES, STREAM_OUT2_BUFFER_ALLOCATE_NUM_BYTES, STREAM_OUT3_BUFFER_ALLOCATE_NUM_BYTES Show All	4050, 4052, 4054, 4056 Show All

STREAM_OUT#(0:3)_ENABLE - Starting Address: 4090

Write 1 to enable, 0 to disable. When enabled, you get 1 update per target per stream scan, so a stream must be active for updates to happen.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 0.9300

Expanded Names	Addresses
STREAM_OUT0_ENABLE, STREAM_OUT1_ENABLE, STREAM_OUT2_ENABLE, STREAM_OUT3_ENABLE Show All	4090, 4092, 4094, 4096 Show All

Configuration can be done before or after stream has started.

3. Update Buffer

Each stream-out has its own buffer. Data is loaded into the buffer by writing to the appropriate buffer register. Output waveform data points are stored in the buffer as 16-bit values, so values greater than 16-bits will be converted automatically before being stored in the buffer. Use only one buffer per `STREAM_OUT` channel.

For outputting an analog waveform (DAC output), write an array of floating-point numbers to the `STREAM_OUT#(0:3)_BUFFER_F32` register.

For outputting a digital waveform, pass an array of integer 0 or 1 values to the `STREAM_OUT#(0:3)_BUFFER_U16` register.

Stream-Out Buffers			
Name	Start Address	Type	Access
STREAM_OUT#(0:3)_BUFFER_U16	4420	UINT16	W
STREAM_OUT#(0:3)_BUFFER_F32	4400	FLOAT32	W

STREAM_OUT#(0:3)_BUFFER_U16 - Starting Address: 4420

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 0.9300

Expanded Names	Addresses
STREAM_OUT0_BUFFER_U16, STREAM_OUT1_BUFFER_U16, STREAM_OUT2_BUFFER_U16, STREAM_OUT3_BUFFER_U16 Show All	4420, 4421, 4422, 4423 Show All

STREAM_OUT#(0:3)_BUFFER_F32 - Starting Address: 4400

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

- Data type: FLOAT32 (type index = 3)
- Write-only
- Default value: 0
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 0.9300

Expanded Names	Addresses
STREAM_OUT0_BUFFER_F32, STREAM_OUT1_BUFFER_F32, STREAM_OUT2_BUFFER_F32, STREAM_OUT3_BUFFER_F32 Show All	4400, 4402, 4404, 4406 Show All

Once the waveform data points are stored, configure STREAM_OUT#(0:3)_LOOP_SIZE and STREAM_OUT#(0:3)_SET_LOOP.

Stream-Out Waveform Periodicity

Name	Start Address	Type	Access
STREAM_OUT#(0:3)_LOOP_SIZE	4060	UINT32	R/W
STREAM_OUT#(0:3)_SET_LOOP	4070	UINT32	W

STREAM_OUT#(0:3)_LOOP_SIZE - Starting Address: 4060

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 0.9300

Expanded Names	Addresses
STREAM_OUT0_LOOP_NUM_VALUES, STREAM_OUT1_LOOP_NUM_VALUES, STREAM_OUT2_LOOP_NUM_VALUES, STREAM_OUT3_LOOP_NUM_VALUES Show All	4060, 4062, 4064, 4066 Show All

STREAM_OUT#(0:3)_SET_LOOP - Starting Address: 4070

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

- Data type: UINT32 (type index = 1)
- Write-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 0.9300

Expanded Names	Addresses
STREAM_OUT0_SET_LOOP, STREAM_OUT1_SET_LOOP, STREAM_OUT2_SET_LOOP, STREAM_OUT3_SET_LOOP Show All	4070, 4072, 4074, 4076 Show All

4. Start stream

Next, start stream with STREAM_OUT#(0:3) in the scan list.

Name	Start Address	Type	Access
STREAM_OUT#(0:3)	4800	UINT16	R

STREAM_OUT#(0:3) - Starting Address: 4800

Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0:127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

- Data type: UINT16 (type index = 0)
- Read-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0012

Expanded Names	Addresses
STREAM_OUT0, STREAM_OUT1, STREAM_OUT2, STREAM_OUT3 Show All	4800, 4801, 4802, 4803 Show All

The order of STREAM_OUT#(0:3) in the scan list determines when the target updated. For example, if STREAM_OUT3 is before STREAM_OUT0 in the scan list, STREAM_OUT3_TARGET will be updated before STREAM_OUT0_TARGET.

5. Stream Loop

Read from stream, if there are stream-in channels.

Also, if the output waveform needs to be updated, read STREAM_OUT#(0:3)_BUFFER_STATUS to determine when to write new values to the buffer. When to write values depends on how large the buffer is and how many values need to be written.

Name	Start Address	Type	Access
STREAM_OUT#(0:3)_BUFFER_STATUS	4080	UIN32	R
<p>STREAM_OUT#(0:3)_BUFFER_STATUS - Starting Address: 4080</p> <p>The number of values in the buffer that are not currently being used.</p> <ul style="list-style-type: none"> Data type: UIN32 (type index = 1) Read-only Default value: 0 T7: <ul style="list-style-type: none"> Minimum <u>firmware</u> version: 0.9300 			
Expanded Names			Addresses
STREAM_OUT0_BUFFER_STATUS, STREAM_OUT1_BUFFER_STATUS, STREAM_OUT2_BUFFER_STATUS, STREAM_OUT3_BUFFER_STATUS Show All			4080, 4082, 4084, 4086 Show All

For a more thorough description of how a Stream-Out buffer works, see [3.2.1.1 Stream-Out Description](#).

6. Stop stream

Stopping a stream that streams out is no different from stopping stream-in.

Example

This example demonstrates how to configure DAC0 to output an analog waveform that resembles a triangle wave, and also quickly measure two analog inputs AIN0 and AIN2 in streaming context.

Configuration steps specific to stream-out

```
STREAM_OUT0_ENABLE = 0    -> Turn off just in case it was already on.
STREAM_OUT0_TARGET = 1000 -> Set the target to DAC0.
STREAM_OUT0_BUFFER_SIZE = 512 -> A buffer to hold up to 256 values.
STREAM_OUT0_ENABLE = 1    -> Turn on Stream-Out0.
```

With the LJM library, write these registers with a call to `eWriteNames` or multiple calls to `eWriteName`.

General stream configuration

```
STREAM_SCANLIST_ADDRESS0= AIN0    -> Add AIN0 to the list of things to stream in.
STREAM_SCANLIST_ADDRESS1= STREAM_OUT0 -> Add STREAM_OUT0 (DAC0 is target) to the list of things to stream out.
STREAM_SCANLIST_ADDRESS2= AIN2    -> Add AIN2 to the list of things to stream in.
STREAM_ENABLE = 1                 -> Start streaming. LJM\_eStreamStart does this.
```

With the LJM library, this is all done with the call to `eStreamStart`.

Other settings related to streaming analog inputs have been omitted here but are covered under the section for [stream mode](#).

Load the waveform data points

The following data points have been chosen to produce the triangle waveform: 0.5V, 1V, 1.5V, 1V, so the next step is to write these datum to the appropriate buffer. Because it is a DAC output (floating point number), use the `STREAM_OUT0_BUFFER_F32` register.

```
STREAM_OUT0_BUFFER_F32 = [0.5, 1, 1.5, 1] -> Write the four values one at a time or as an array.
STREAM_OUT0_LOOP_SIZE = 4                 -> Loop four values.
STREAM_OUT0_SET_LOOP = 1                  -> Begin using new data set immediately.
```

With the LJM library, write the array using `eWriteNameArray`, and write the other 2 values with a call to `eWriteNames` or multiple calls to `eWriteName`.

Observe result with stream mode

Every time the stream is run, AIN0 is read, then DAC0 is updated with a data point from Stream-Out0's buffer, then AIN2 is read. Thus, the streaming speed dictates the frequency of the output waveform.

Sequential Data

Once a sequence of values has been set via the `STREAM_OUT#_SET_LOOP` register, that sequence of values will loop and only be interrupted at the end of the sequence. Therefore, to have stream-out continuously output a sequence of values that is larger than the size of one stream out buffer, probably the easiest way to do so is to:

1. Start by dividing the stream out buffer into 2 halves,
2. Write one half of the buffer with your sequential data,
3. In a loop, every time the `STREAM_OUT#_BUFFER_STATUS` reads as being half full/empty, write another half buffer-worth of values.

Note that the buffer is a circular array, so you could end up overwriting values if you're not careful.

Here's an example:

Stream-out buffer is 512 bytes, divide that by 2 to get the number of samples the buffer can hold => 256 samples

256 samples divided by 2 to get the "loop" size, AKA the set-of-data-to-be-written-at-a-time size => 128 samples

Write 128 samples:

Write 128 to STREAM_OUT0_LOOP_SIZE

Write 128 samples to STREAM_OUT0_BUFFER_F32 (This should probably be done by array write, which is much faster than writing values individually.)

Write 1 to STREAM_OUT0_SET_LOOP

Loop while you have more sequential data to write:

Read STREAM_OUT0_BUFFER_STATUS

If STREAM_OUT0_BUFFER_STATUS is 128 or greater, write the next 128 samples, along with STREAM_OUT0_LOOP_SIZE = 128 and STREAM_OUT0_SET_LOOP = 1

Sleep for something like $1 / \text{scanRate}$ seconds to prevent unnecessary work for the hardware

3.2.1.1 Stream-Out Description [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

3.2.2 Low-Level Streaming [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

Stream mode is complicated but can easily be executed using the high-level LJM stream functions. LJM is recommend for all users, except users that need to integrate a T-series device into a system that cannot use LJM. The rest of this section is about manually executing stream protocol without LJM. For an introduction to stream and for additional configurations, see 3.2 Stream Mode.

Executing stream mode involves the following:

- Stream setup
- Stream start
- Stream-in data collection, if any stream includes stream-in channels
- Stream-out buffer updates, if stream includes stream-out channels (See 3.2.1 Stream-Out)
- Stream stop

Spontaneous Stream vs. Command-Response Stream:

Data can be sent to the host in one of two data collection modes:

- Spontaneous: In spontaneous mode, packets are automatically sent to the host as soon as there is enough data to fill a packet. The packet size is adjustable. See the register definitions below.
- Command-Response (CR): In CR mode, the stream data is stored in the device's buffer and must be read out using a command. CR mode is useful for when the connection is unreliable.

T-series devices connected via either USB and Ethernet are capable of both spontaneous stream and command-response stream.

T7-Pro only: T7-Pro devices connected via WiFi are capable of only command-response stream.

Setup

Manual stream setup requires configuration of the registers that `LJM_eStreamStart` automatically configures:

Manual Stream Setup

Name	Start Address	Type	Access
STREAM_SCANRATE_HZ	4002	FLOAT32	R/W
STREAM_NUM_ADDRESSES	4004	UINT32	R/W
STREAM_SAMPLES_PER_PACKET	4006	UINT32	R/W
STREAM_AUTO_TARGET	4016	UINT32	R/W
STREAM_SCANLIST_ADDRESS#(0:127)	4100	UINT32	R/W
STREAM_ENABLE	4990	UINT32	R/W

STREAM_SCANRATE_HZ - Address: 4002

Write a value to specify the number of times per second that all channels in the stream scanlist will be read. Max stream speeds are based on Sample Rate which is $\text{NumChannels} * \text{ScanRate}$. Has no effect when using and external clock. A read of this register returns the actual scan rate, which can be slightly different due to rounding. For scan rates > 152.588 , the actual scan interval is multiples of 100 ns. Assuming a core clock of 80 MHz the internal roll value is $(80M / (8 * \text{DesiredScanRate})) - 1$ and the actual scan rate is then $80M / (8 * (\text{RollValue} + 1))$. For slower scan rates the scan interval resolution is changed to 1 us, 10 us, 100 us, or 1 ms as needed to achieve the longer intervals.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0

STREAM_NUM_ADDRESSES - Address: 4004

The number of entries in the scanlist

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

STREAM_SAMPLES_PER_PACKET - Address: 4006

Specifies the number of data points to be sent in the data packet. Only applies to spontaneous mode.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

STREAM_AUTO_TARGET - Address: 4016

Controls where data will be sent. Value is a bitmask. bit 0: 1 = Send to Ethernet 702 sockets, bit 1: 1 = Send to USB, bit 4: 1 = Command-Response mode. All other bits are reserved.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 0.9108

STREAM_SCANLIST_ADDRESS#(0:127) - Starting Address: 4100

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

Expanded Names

Expanded Names	Addresses
STREAM_SCANLIST_ADDRESS0, STREAM_SCANLIST_ADDRESS1, STREAM_SCANLIST_ADDRESS2, STREAM_SCANLIST_ADDRESS3, STREAM_SCANLIST_ADDRESS4, STREAM_SCANLIST_ADDRESS5, STREAM_SCANLIST_ADDRESS6, STREAM_SCANLIST_ADDRESS7, STREAM_SCANLIST_ADDRESS8, STREAM_SCANLIST_ADDRESS9, STREAM_SCANLIST_ADDRESS10, STREAM_SCANLIST_ADDRESS11, STREAM_SCANLIST_ADDRESS12, STREAM_SCANLIST_ADDRESS13, STREAM_SCANLIST_ADDRESS14, STREAM_SCANLIST_ADDRESS15, STREAM_SCANLIST_ADDRESS16, STREAM_SCANLIST_ADDRESS17, STREAM_SCANLIST_ADDRESS18, STREAM_SCANLIST_ADDRESS19, STREAM_SCANLIST_ADDRESS20, STREAM_SCANLIST_ADDRESS21, STREAM_SCANLIST_ADDRESS22, STREAM_SCANLIST_ADDRESS23, STREAM_SCANLIST_ADDRESS24, STREAM_SCANLIST_ADDRESS25, STREAM_SCANLIST_ADDRESS26, STREAM_SCANLIST_ADDRESS27, STREAM_SCANLIST_ADDRESS28, STREAM_SCANLIST_ADDRESS29, STREAM_SCANLIST_ADDRESS30, STREAM_SCANLIST_ADDRESS31, STREAM_SCANLIST_ADDRESS32, STREAM_SCANLIST_ADDRESS33, STREAM_SCANLIST_ADDRESS34, STREAM_SCANLIST_ADDRESS35, STREAM_SCANLIST_ADDRESS36, STREAM_SCANLIST_ADDRESS37, STREAM_SCANLIST_ADDRESS38, STREAM_SCANLIST_ADDRESS39, STREAM_SCANLIST_ADDRESS40, STREAM_SCANLIST_ADDRESS41, STREAM_SCANLIST_ADDRESS42, STREAM_SCANLIST_ADDRESS43, STREAM_SCANLIST_ADDRESS44, STREAM_SCANLIST_ADDRESS45, STREAM_SCANLIST_ADDRESS46, STREAM_SCANLIST_ADDRESS47, STREAM_SCANLIST_ADDRESS48, STREAM_SCANLIST_ADDRESS49, STREAM_SCANLIST_ADDRESS50, STREAM_SCANLIST_ADDRESS51, STREAM_SCANLIST_ADDRESS52, STREAM_SCANLIST_ADDRESS53, STREAM_SCANLIST_ADDRESS54, STREAM_SCANLIST_ADDRESS55, STREAM_SCANLIST_ADDRESS56, STREAM_SCANLIST_ADDRESS57, STREAM_SCANLIST_ADDRESS58, STREAM_SCANLIST_ADDRESS59, STREAM_SCANLIST_ADDRESS60, STREAM_SCANLIST_ADDRESS61, STREAM_SCANLIST_ADDRESS62, STREAM_SCANLIST_ADDRESS63, STREAM_SCANLIST_ADDRESS64, STREAM_SCANLIST_ADDRESS65, STREAM_SCANLIST_ADDRESS66, STREAM_SCANLIST_ADDRESS67, STREAM_SCANLIST_ADDRESS68, STREAM_SCANLIST_ADDRESS69, STREAM_SCANLIST_ADDRESS70, STREAM_SCANLIST_ADDRESS71, STREAM_SCANLIST_ADDRESS72, STREAM_SCANLIST_ADDRESS73, STREAM_SCANLIST_ADDRESS74, STREAM_SCANLIST_ADDRESS75, STREAM_SCANLIST_ADDRESS76, STREAM_SCANLIST_ADDRESS77, STREAM_SCANLIST_ADDRESS78, STREAM_SCANLIST_ADDRESS79, STREAM_SCANLIST_ADDRESS80, STREAM_SCANLIST_ADDRESS81, STREAM_SCANLIST_ADDRESS82, STREAM_SCANLIST_ADDRESS83, STREAM_SCANLIST_ADDRESS84, STREAM_SCANLIST_ADDRESS85, STREAM_SCANLIST_ADDRESS86, STREAM_SCANLIST_ADDRESS87, STREAM_SCANLIST_ADDRESS88, STREAM_SCANLIST_ADDRESS89, STREAM_SCANLIST_ADDRESS90, STREAM_SCANLIST_ADDRESS91, STREAM_SCANLIST_ADDRESS92, STREAM_SCANLIST_ADDRESS93, STREAM_SCANLIST_ADDRESS94, STREAM_SCANLIST_ADDRESS95, STREAM_SCANLIST_ADDRESS96, STREAM_SCANLIST_ADDRESS97, STREAM_SCANLIST_ADDRESS98, STREAM_SCANLIST_ADDRESS99, STREAM_SCANLIST_ADDRESS100, STREAM_SCANLIST_ADDRESS101, STREAM_SCANLIST_ADDRESS102, STREAM_SCANLIST_ADDRESS103, STREAM_SCANLIST_ADDRESS104, STREAM_SCANLIST_ADDRESS105, STREAM_SCANLIST_ADDRESS106, STREAM_SCANLIST_ADDRESS107, STREAM_SCANLIST_ADDRESS108, STREAM_SCANLIST_ADDRESS109, STREAM_SCANLIST_ADDRESS110, STREAM_SCANLIST_ADDRESS111, STREAM_SCANLIST_ADDRESS112, STREAM_SCANLIST_ADDRESS113, STREAM_SCANLIST_ADDRESS114, STREAM_SCANLIST_ADDRESS115, STREAM_SCANLIST_ADDRESS116, STREAM_SCANLIST_ADDRESS117, STREAM_SCANLIST_ADDRESS118, STREAM_SCANLIST_ADDRESS119, STREAM_SCANLIST_ADDRESS120, STREAM_SCANLIST_ADDRESS121, STREAM_SCANLIST_ADDRESS122, STREAM_SCANLIST_ADDRESS123, STREAM_SCANLIST_ADDRESS124, STREAM_SCANLIST_ADDRESS125, STREAM_SCANLIST_ADDRESS126, STREAM_SCANLIST_ADDRESS127	4100, 4102, 4104, 4106, 4108, 4110, 4112, 4114, 4116, 4118, 4120, 4122, 4124, 4126, 4128, 4130, 4132, 4134, 4136, 4138, 4140, 4142, 4144, 4146, 4148, 4150, 4152, 4154, 4156, 4158, 4160, 4162, 4164, 4166, 4168, 4170, 4172, 4174, 4176, 4178, 4180, 4182, 4184, 4186, 4188, 4190, 4192, 4194, 4196, 4198, 4200, 4202, 4204, 4206, 4208, 4210, 4212, 4214, 4216, 4218, 4220, 4222, 4224, 4226, 4228, 4230, 4232, 4234, 4236, 4238, 4240, 4242, 4244, 4246, 4248, 4250,

STREAM_SCANLIST_ADDRESS76, STREAM_SCANLIST_ADDRESS77, STREAM_SCANLIST_ADDRESS78, STREAM_SCANLIST_ADDRESS79,	4252, 4254, 4256, 4258,
STREAM_SCANLIST_ADDRESS80, STREAM_SCANLIST_ADDRESS81, STREAM_SCANLIST_ADDRESS82, STREAM_SCANLIST_ADDRESS83,	4260, 4262, 4264, 4266,
STREAM_SCANLIST_ADDRESS84, STREAM_SCANLIST_ADDRESS85, STREAM_SCANLIST_ADDRESS86, STREAM_SCANLIST_ADDRESS87,	4268, 4270, 4272, 4274,
STREAM_SCANLIST_ADDRESS88, STREAM_SCANLIST_ADDRESS89, STREAM_SCANLIST_ADDRESS90, STREAM_SCANLIST_ADDRESS91,	4276, 4278, 4280, 4282,
STREAM_SCANLIST_ADDRESS92, STREAM_SCANLIST_ADDRESS93, STREAM_SCANLIST_ADDRESS94, STREAM_SCANLIST_ADDRESS95,	4284, 4286, 4288, 4290,
STREAM_SCANLIST_ADDRESS96, STREAM_SCANLIST_ADDRESS97, STREAM_SCANLIST_ADDRESS98, STREAM_SCANLIST_ADDRESS99,	4292, 4294, 4296, 4298,
STREAM_SCANLIST_ADDRESS100, STREAM_SCANLIST_ADDRESS101, STREAM_SCANLIST_ADDRESS102,	4300, 4302, 4304, 4306,
STREAM_SCANLIST_ADDRESS103, STREAM_SCANLIST_ADDRESS104, STREAM_SCANLIST_ADDRESS105,	4308, 4310, 4312, 4314,
STREAM_SCANLIST_ADDRESS106, STREAM_SCANLIST_ADDRESS107, STREAM_SCANLIST_ADDRESS108,	4316, 4318, 4320, 4322,
STREAM_SCANLIST_ADDRESS109, STREAM_SCANLIST_ADDRESS110, STREAM_SCANLIST_ADDRESS111,	4324, 4326, 4328, 4330,
STREAM_SCANLIST_ADDRESS112, STREAM_SCANLIST_ADDRESS113, STREAM_SCANLIST_ADDRESS114,	4332, 4334, 4336, 4338,
STREAM_SCANLIST_ADDRESS115, STREAM_SCANLIST_ADDRESS116, STREAM_SCANLIST_ADDRESS117,	4340, 4342, 4344, 4346,
STREAM_SCANLIST_ADDRESS118, STREAM_SCANLIST_ADDRESS119, STREAM_SCANLIST_ADDRESS120,	4348, 4350, 4352, 4354
STREAM_SCANLIST_ADDRESS121, STREAM_SCANLIST_ADDRESS122, STREAM_SCANLIST_ADDRESS123,	Show All
STREAM_SCANLIST_ADDRESS124, STREAM_SCANLIST_ADDRESS125, STREAM_SCANLIST_ADDRESS126, STREAM_SCANLIST_ADDRESS127	Show All

STREAM_ENABLE - Address: 4990

Write 1 to start stream. Write 0 to stop stream. Reading this register returns 1 is stream is enabled. When using a triggered stream the stream is considered enabled while waiting for the trigger.

- Data type: UINT32 (type index = 1)
- Readable and writable
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 0.9421

Additional Configuration Notes

Additionally, address 4018 (STREAM_DATATYPE) must be written with the value 0. Note that address 4018 (STREAM_DATATYPE) is not in [ljm_constants.json](#) and is not compatible with [LJM_NameToAddress](#).

STREAM_ENABLE must be written last.

For other stream configuration registers, which are not required for all streams, see [3.2 Stream Mode](#).

Data Collection

Spontaneous Stream: Once stream has been initiated with STREAM_ENABLE, the device sends data to the target indicated by STREAM_AUTO_TARGET until STREAM_ENABLE is written with the value 0. Stream-out streams that do not contain stream-in channels (see above) do not send data.

Modbus Feedback Spontaneous Packet Protocol:

Bytes 0-1: Transaction ID

Bytes 2-3: Protocol ID

Bytes 4-5: Length, MSB-LSB

Bytes 6: 1 (Unit ID)

Byte 7: 76 (Function #)

Byte 8: 16

Byte 9: Reserved

Bytes 10-11: Backlog Bytes

Bytes 12-13: Status Code

Byte 14-15: Additional status information

Byte 16+: Stream Data (raw sample = 2 bytes MSB-LSB)

Command-Response Stream: When collecting data using command-response stream mode, data must be read from STREAM_DATA_CR (address 4500). Data is automatically discarded as it is read.

Modbus Feedback Command-Response Packet Protocol:

Bytes 0-1: Transaction ID

Bytes 2-3: Protocol ID

Bytes 4-5: Length, MSB-LSB

Bytes 6: 1 (Unit ID)

Byte 7: 76 (Function #)

Bytes 8-9: Number of samples in this read

Bytes 10-11: Backlog Bytes

Bytes 12-13: Status Code

Byte 14-15: Additional status information

Byte 16+: Stream Data (raw sample = 2 bytes MSB-LSB)

Backlog Bytes:

Backlog Bytes is the number bytes contained in the device stream buffer after reading. To convert BacklogBytes to the number of scans still on the device:

$$\text{BacklogScans} = \text{BacklogBytes} / (\text{bytesPerSample} * \text{samplesPerScan})$$

Where bytesPerSample is 2 and samplesPerScan is the number of channels.

Status Codes:

- 2940: Auto-recovery Active.
- 2941: Auto-recovery End. Additional Status Information is the number of scans skipped. A scan consisting of all 0xFFFF values indicates the separation between old data and new data.
- 2942: Scan Overlap
- 2943: Auto-recovery End Overflow
- 2944: Stream Burst Complete

Stop

To stop stream, write 0 to STREAM_ENABLE. All stream modes expect to be stopped, except for burst stream (see STREAM_NUM_SCANS for more information on bust stream).

Code Example

A general low-level stream example written in C/C++ can be found [here](#).

4.0 Hardware Overview [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Above are links to specific hardware information for the different T-series devices.

Following is a listing of registers for general device information.

General Device Information Registers

Name	Start Address	Type	Access
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
ETHERNET_MAC	60020	UINT64	R
WIFI_MAC	60024	UINT64	R
SERIAL_NUMBER	60028	UINT32	R
DEVICE_NAME_DEFAULT	60500	STRING	R/W

PRODUCT_ID - Address: 60000

The numeric identifier of the device. Such as 7 for a T7 / T7-Pro.

- Data type: FLOAT32 (type index = 3)
- Read-only

HARDWARE_VERSION - Address: 60002

The hardware version of the device.

- Data type: FLOAT32 (type index = 3)
- Read-only

FIRMWARE_VERSION - Address: 60004

The current firmware version installed on the main processor.

- Data type: FLOAT32 (type index = 3)
- Read-only

BOOTLOADER_VERSION - Address: 60006

The bootloader version installed on the main processor.

- Data type: FLOAT32 (type index = 3)
- Read-only

WIFI_VERSION - Address: 60008

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

- Data type: FLOAT32 (type index = 3)
- Read-only

HARDWARE_INSTALLED - Address: 60010

Bitmask indicating installed hardware options. bit0: High Resolution ADC, bit1: WiFi, bit2: RTC, bit3: microSD.

- Data type: UINT32 (type index = 1)
- Read-only
- Default value: 0

ETHERNET_MAC - Address: 60020

The MAC address of the wired Ethernet module.

- Data type: UINT64 (type index = N/A)
- Read-only

WIFI_MAC - Address: 60024

The MAC address of the WiFi module.

- Data type: UINT64 (type index = N/A)
- Read-only

SERIAL_NUMBER - Address: 60028

The serial number of the device.

- Data type: UINT32 (type index = 1)
- Read-only

DEVICE_NAME_DEFAULT - Address: 60500

Reads return the current device name. Writes update the default and current device name. A reboot is necessary to update the name reported by NBNS. Up to 49 characters, cannot contain periods.

- Data type: STRING (type index = 98)
- Readable and writable

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

Miscellaneous Device Registers

Name	Start Address	Type	Access
CORE_TIMER	61520	UINT32	R
SYSTEM_TIMER_20HZ	61522	UINT32	R
SYSTEM_REBOOT	61998	UINT32	W
WAIT_US_BLOCKING	61590	UINT32	R/W

CORE_TIMER - Address: 61520

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

- Data type: UINT32 (type index = 1)
- Read-only
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0071

SYSTEM_TIMER_20HZ - Address: 61522

Internal 32-bit system timer running at 20Hz.

- Data type: UINT32 (type index = 1)
- Read-only
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0071

SYSTEM_REBOOT - Address: 61998

Issues a device reboot. Must write 0x4C4Axxxx, where xxxx is number of 50ms ticks to wait before reboot. To reboot immediately write 0x4C4A0000 (d1279918080).

- Data type: UINT32 (type index = 1)
- Write-only
- T7:
 - Minimum [firmware](#) version: 0.9419
- T4:
 - Minimum [firmware](#) version: 0.2020

WAIT_US_BLOCKING - Address: 61590

Delays for x microseconds. Range is 0-100000.

- Data type: UINT32 (type index = 1)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 1.0026

4.1 T4 Hardware [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T4 Hardware Overview

The T4 has 3 different I/O areas:

- **Communication Edge:** The communication edge (top edge in Figure 4.1-1) has a USB Type-B connector and an RJ45 Ethernet connector. Power is always provided through the USB connector, even if USB communication is not used.
- **Screw Terminal Edges:** The screw terminal edges have convenient connections for the 2 analog outputs, 4 high-voltage analog inputs, and 4 flexible I/O (digital I/O, low-voltage analog inputs, or DIO-EF. The screw terminals are arranged in blocks of 4, with each block consisting of VS, GND, and two I/O. Also on the left screw-terminal edge are two LEDs. The Comm LED generally blinks with communication traffic, while the Status LED is used for other indications.
- **DB Edge:** The DB Edge has a 15-pin female D-sub type connector (DB15) which has 12 digital I/O called EIO and CIO. The first 4 EIO lines can be configured as low-voltage analog inputs. The first 2 EIO lines and the 4 CIO lines support some DIO-EF features (timers, counters, etc.).

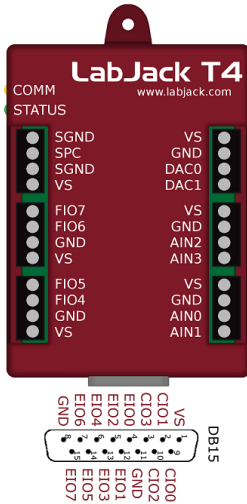


Figure 4.1-1. LabJack T4

USB: Can be used for host communication. Power is always provided through this connector.

Ethernet: 10/100Base-T Ethernet connection can be used for host communication.

LEDs: The Power and Status LEDs convey different information about the device.

VS: All VS terminals are the same. These are outputs that can be used to source about 5 volts.

GND/SGND: All GND terminals are the same. SGND has a self-resetting thermal fuse in series with GND.

FIO#/EIO#/CIO#: These are the 16 digital I/O, and are also referred to as DIO4-DIO19. Besides basic digital I/O operations, some of these terminals can be configured with [Extended Features](#) (frequency input, PWM output, etc.), some can be configured as low-voltage analog inputs, and all can be configured for various serial protocols: [I²C](#) serial, [SPI](#) serial, [S₁BUS](#) serial (EI-1050, SHT sensors), [1-Wire](#) serial, and [Asynchronous](#) serial.

AIN#: AIN0-AIN3 are the 4 high-voltage ($\pm 10V$) analog inputs.

DAC#: DAC0 & DAC1 are the 2 analog outputs. Each DAC can be set to a voltage between about 0.01 and 5 volts with 10-bits of resolution.

For information about reading inputs, start in [Section 3](#). For information about setting outputs, start with the [Waveform Generation Application Note](#).

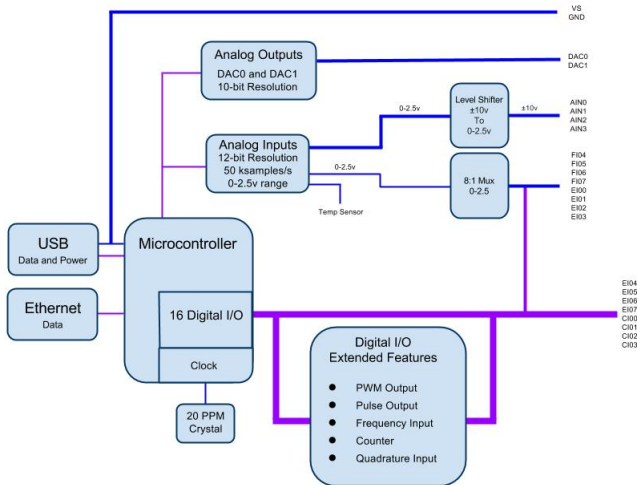


Figure 4.1-2. Block Diagram

4.2 T7 Hardware [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T7 Hardware Overview

The T7 has 3 different I/O areas:

- **Communication Edge:** The T7 has a USB Type-B connector and an RJ45 Ethernet connector. The T7-Pro has those and also has an SMA-RP female connector and a WiFi antenna. Power is always provided through the USB connector, even if USB communication is not used.
- **Screw Terminal Edge:** The screw terminal edge has convenient connections for 4 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.
- **DB Edge:** The DB Edge has 2 D-sub type connectors: a DB15 and DB37. The DB15 has 12 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.

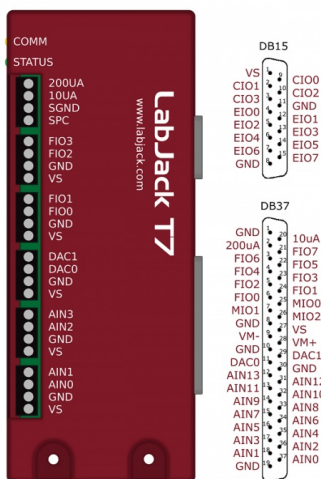


Figure 4.2-1. Enclosure & Connectors

USB: Can be used for host communication. Power is always provided through this connector.

Ethernet: 10/100Base-T Ethernet connection can be used for host communication.

WiFi (T7-Pro only): 2.4 GHz 802.11 b/g WiFi connection can be used for host communication.

LEDs: The Power and Status LEDs convey different information about the device.

VS: All VS terminals are the same. These are power outputs that can be used to source about 5 volts.

GND/SGND: All GND terminals are the same. SGND has a self-resetting thermal fuse in series with GND.

10uA/200uA: Fixed current sources providing 10uA/200uA at a max voltage of about 3 volts.

FIO#/EIO#/CIO#/MIO#: These are the 23 digital I/O, and are also referred to as DIO0-DIO22. Besides basic digital I/O operations, some of these terminals can also be configured with Extended Features (frequency input, PWM output, etc.), and all can be configured for various serial protocols: I²C serial, SPI serial, S_{BU}S serial (EI-1050, SHT sensors), 1-Wire serial,

and Asynchronous serial.

AIN#: AIN0-AIN13 are the 14 analog inputs.

DAC#: DAC0 & DAC1 are the 2 analog outputs. Each DAC can be set to a voltage between about 0.01 and 5 volts with 12-bits of resolution.

For information about reading inputs, start in Section 3. For information about setting outputs, start with the Waveform Generation Application Note.

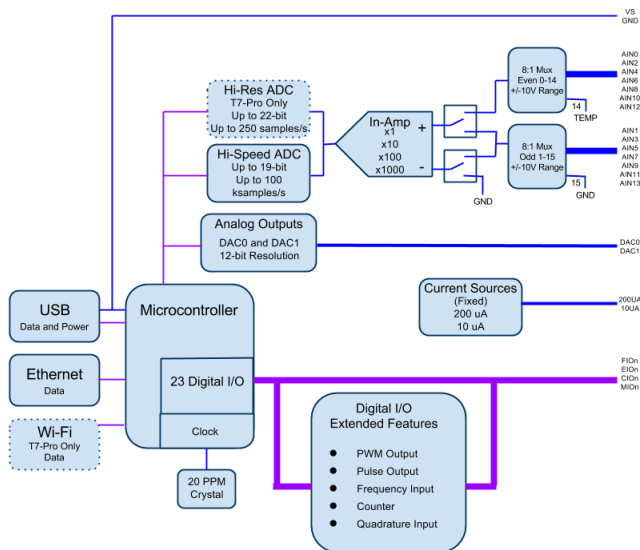


Figure 4.2-2. Block Diagram

4.3 Hardware Revisions [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T4 Hardware Revisions:

- 1.2: Initial Release of the T4

T7 Hardware Revisions:

- 1.31: Initial release of the T7
- 1.35: Several changes were made to improve testing and manufacturing reliability.
 - 1.35a: Switched to a new flash chip. Previous one was discontinued by the manufacturer. This version will not operate properly if a firmware version lower than 1.0218 is installed. Attempting to load firmware below 1.0218 will cause an error to be thrown.

T7-OEM Hardware Revisions:

Same revision history as T7 hardware. HW 1.35 also changed the style of LEDs that are installed. HW 1.31 came with through hole LEDs. HW 1.35 has SMD LEDs installed next to the through hole component locations making them easier to remove. They also draw less power and are lower profile.

4.4 RAM [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

Maximum system RAM: 64 KB

When enough RAM is allocated, the next attempt to allocate RAM will fail with the error `SYSTEM_MEMORY_BEREFT` (errorcode: 2400).

If `SYSTEM_MEMORY_BEREFT` is occurring, your application must either use smaller amounts of RAM or free previously allocated RAM. Some common things to try:

- To free allocated stream RAM, stop stream. To use less stream RAM, use a smaller STREAM_BUFFER_SIZE_BYTES and/or smaller STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES
- To free other allocated RAM:
 - Write 0 LUA_RUN
 - Write 0 USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES
 - Write 0 AIN#(0:149)_EF_INDEX

RAM-Allocating Registers

The below registers allocate system RAM (and may return SYSTEM_MEMORY_BEREF):

Stream			
Name	Start Address	Type	Access
STREAM_ENABLE	4990	UINT32	R/W
STREAM_OUT#(0:2)_ENABLE	4090	UINT32	R/W

STREAM_ENABLE - Address: 4990

Write 1 to start stream. Write 0 to stop stream. Reading this register returns 1 is stream is enabled. When using a triggered stream the stream is considered enabled while waiting for the trigger.

- Data type: UINT32 (type index = 1)
- Readable and writable
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 0.9421

STREAM_OUT#(0:2)_ENABLE - Starting Address: 4090

Write 1 to enable, 0 to disable. When enabled, you get 1 update per target per stream scan, so a stream must be active for updates to happen.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 0.9300

Expanded Names	Addresses
STREAM_OUT0_ENABLE, STREAM_OUT1_ENABLE, STREAM_OUT2_ENABLE	4090, 4092, 4094

STREAM_BUFFER_SIZE_BYTES does not allocate RAM, but it does set the amount of RAM allocated by STREAM_ENABLE. `LJM_eStreamStart` writes to STREAM_ENABLE.

Similarly, STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES does not allocate RAM, but sets the amount of RAM allocated by STREAM_OUT#(0:3)_ENABLE.

Lua			
Name	Start Address	Type	Access
LUA_RUN	6000	UINT32	R/W
LUA_SOURCE_SIZE	6012	UINT32	R/W
LUA_DEBUG_ENABLE	6020	UINT32	R/W

LUA_RUN - Address: 6000

Writing 1 compiles and runs the Lua script that is loaded in RAM. Writing zero stops the script and begins cleaning up memory. Users may poll the register after writing a value of 0 to verify that the VM is unloaded, and garbage collection is complete. 0 = VM fully unloaded. 1 = Running/in-progress

- Data type: UINT32 (type index = 1)
- Readable and writable
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0018

LUA_SOURCE_SIZE - Address: 6012

Allocates RAM for the source code.

- Data type: UINT32 (type index = 1)
- Readable and writable
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0018

LUA_DEBUG_ENABLE - Address: 6020

Write 1 to this register to enable debugging.

- Data type: UINT32 (type index = 1)
- Readable and writable
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0018

Lua scripts dynamically allocate RAM.

User			
Name	Start Address	Type	Access

USER_RAM_FIFO#(0:2)_ALLOCATE_NUM_BYTES - Starting Address: 47900

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0163

Expanded Names

Addresses

USER_RAM_FIFO0_ALLOCATE_NUM_BYTES, USER_RAM_FIFO1_ALLOCATE_NUM_BYTES, USER_RAM_FIFO2_ALLOCATE_NUM_BYTES 47900, 47902, 47904

AIN Extended Features

Name	Start Address	Type	Access
AIN#(0:148)_EF_INDEX	9000	UINT32	R/W
AIN#(0:148)_EF_READ_A	7000	FLOAT32	R

AIN#(0:148)_EF_INDEX - Starting Address: 9000

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0030
- T4:
 - The T4 does not support thermocouple modes.

Expanded Names

Addresses

AIN0_EF_INDEX, AIN1_EF_INDEX, AIN2_EF_INDEX, AIN3_EF_INDEX, AIN4_EF_INDEX, AIN5_EF_INDEX, AIN6_EF_INDEX, AIN7_EF_INDEX, AIN8_EF_INDEX, AIN9_EF_INDEX, AIN10_EF_INDEX, AIN11_EF_INDEX, AIN12_EF_INDEX, AIN13_EF_INDEX, AIN14_EF_INDEX, AIN15_EF_INDEX, AIN16_EF_INDEX, AIN17_EF_INDEX, AIN18_EF_INDEX, AIN19_EF_INDEX, AIN20_EF_INDEX, AIN21_EF_INDEX, AIN22_EF_INDEX, AIN23_EF_INDEX, AIN24_EF_INDEX, AIN25_EF_INDEX, AIN26_EF_INDEX, AIN27_EF_INDEX, AIN28_EF_INDEX, AIN29_EF_INDEX, AIN30_EF_INDEX, AIN31_EF_INDEX, AIN32_EF_INDEX, AIN33_EF_INDEX, AIN34_EF_INDEX, AIN35_EF_INDEX, AIN36_EF_INDEX, AIN37_EF_INDEX, AIN38_EF_INDEX, AIN39_EF_INDEX, AIN40_EF_INDEX, AIN41_EF_INDEX, AIN42_EF_INDEX, AIN43_EF_INDEX, AIN44_EF_INDEX, AIN45_EF_INDEX, AIN46_EF_INDEX, AIN47_EF_INDEX, AIN48_EF_INDEX, AIN49_EF_INDEX, AIN50_EF_INDEX, AIN51_EF_INDEX, AIN52_EF_INDEX, AIN53_EF_INDEX, AIN54_EF_INDEX, AIN55_EF_INDEX, AIN56_EF_INDEX, AIN57_EF_INDEX, AIN58_EF_INDEX, AIN59_EF_INDEX, AIN60_EF_INDEX, AIN61_EF_INDEX, AIN62_EF_INDEX, AIN63_EF_INDEX, AIN64_EF_INDEX, AIN65_EF_INDEX, AIN66_EF_INDEX, AIN67_EF_INDEX, AIN68_EF_INDEX, AIN69_EF_INDEX, AIN70_EF_INDEX, AIN71_EF_INDEX, AIN72_EF_INDEX, AIN73_EF_INDEX, AIN74_EF_INDEX, AIN75_EF_INDEX, AIN76_EF_INDEX, AIN77_EF_INDEX, AIN78_EF_INDEX, AIN79_EF_INDEX, AIN80_EF_INDEX, AIN81_EF_INDEX, AIN82_EF_INDEX, AIN83_EF_INDEX, AIN84_EF_INDEX, AIN85_EF_INDEX, AIN86_EF_INDEX, AIN87_EF_INDEX, AIN88_EF_INDEX, AIN89_EF_INDEX, AIN90_EF_INDEX, AIN91_EF_INDEX, AIN92_EF_INDEX, AIN93_EF_INDEX, AIN94_EF_INDEX, AIN95_EF_INDEX, AIN96_EF_INDEX, AIN97_EF_INDEX, AIN98_EF_INDEX, AIN99_EF_INDEX, AIN100_EF_INDEX, AIN101_EF_INDEX, AIN102_EF_INDEX, AIN103_EF_INDEX, AIN104_EF_INDEX, AIN105_EF_INDEX, AIN106_EF_INDEX, AIN107_EF_INDEX, AIN108_EF_INDEX, AIN109_EF_INDEX, AIN110_EF_INDEX, AIN111_EF_INDEX, AIN112_EF_INDEX, AIN113_EF_INDEX, AIN114_EF_INDEX, AIN115_EF_INDEX, AIN116_EF_INDEX, AIN117_EF_INDEX, AIN118_EF_INDEX, AIN119_EF_INDEX, AIN120_EF_INDEX, AIN121_EF_INDEX, AIN122_EF_INDEX, AIN123_EF_INDEX, AIN124_EF_INDEX, AIN125_EF_INDEX, AIN126_EF_INDEX, AIN127_EF_INDEX, AIN128_EF_INDEX, AIN129_EF_INDEX, AIN130_EF_INDEX, AIN131_EF_INDEX, AIN132_EF_INDEX, AIN133_EF_INDEX, AIN134_EF_INDEX, AIN135_EF_INDEX, AIN136_EF_INDEX, AIN137_EF_INDEX, AIN138_EF_INDEX, AIN139_EF_INDEX, AIN140_EF_INDEX, AIN141_EF_INDEX, AIN142_EF_INDEX, AIN143_EF_INDEX, AIN144_EF_INDEX, AIN145_EF_INDEX, AIN146_EF_INDEX, AIN147_EF_INDEX, AIN148_EF_INDEX [Show All](#)

AIN#(0:148)_EF_READ_A - Starting Address: 7000

Function dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Read-only
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names

Addresses

AIN0_EF_READ_A, AIN1_EF_READ_A, AIN2_EF_READ_A, AIN3_EF_READ_A, AIN4_EF_READ_A, AIN5_EF_READ_A, AIN6_EF_READ_A, AIN7_EF_READ_A, AIN8_EF_READ_A, AIN9_EF_READ_A, AIN10_EF_READ_A, AIN11_EF_READ_A, AIN12_EF_READ_A, AIN13_EF_READ_A, AIN14_EF_READ_A, AIN15_EF_READ_A, AIN16_EF_READ_A, AIN17_EF_READ_A, AIN18_EF_READ_A, AIN19_EF_READ_A, AIN20_EF_READ_A, AIN21_EF_READ_A, AIN22_EF_READ_A, AIN23_EF_READ_A, AIN24_EF_READ_A, AIN25_EF_READ_A, AIN26_EF_READ_A, AIN27_EF_READ_A, AIN28_EF_READ_A, AIN29_EF_READ_A, AIN30_EF_READ_A, AIN31_EF_READ_A, AIN32_EF_READ_A, AIN33_EF_READ_A, AIN34_EF_READ_A, AIN35_EF_READ_A, AIN36_EF_READ_A, AIN37_EF_READ_A, AIN38_EF_READ_A, AIN39_EF_READ_A, AIN40_EF_READ_A, AIN41_EF_READ_A, AIN42_EF_READ_A, AIN43_EF_READ_A, AIN44_EF_READ_A, AIN45_EF_READ_A, AIN46_EF_READ_A, AIN47_EF_READ_A, AIN48_EF_READ_A, AIN49_EF_READ_A, AIN50_EF_READ_A, AIN51_EF_READ_A, AIN52_EF_READ_A, AIN53_EF_READ_A, AIN54_EF_READ_A, AIN55_EF_READ_A, AIN56_EF_READ_A, AIN57_EF_READ_A, AIN58_EF_READ_A, AIN59_EF_READ_A, AIN60_EF_READ_A, AIN61_EF_READ_A, AIN62_EF_READ_A, AIN63_EF_READ_A, AIN64_EF_READ_A, AIN65_EF_READ_A, AIN66_EF_READ_A, AIN67_EF_READ_A, AIN68_EF_READ_A, AIN69_EF_READ_A, AIN70_EF_READ_A, AIN71_EF_READ_A, AIN72_EF_READ_A, AIN73_EF_READ_A, AIN74_EF_READ_A, AIN75_EF_READ_A, AIN76_EF_READ_A, AIN77_EF_READ_A, AIN78_EF_READ_A, AIN79_EF_READ_A, AIN80_EF_READ_A, AIN81_EF_READ_A, AIN82_EF_READ_A, AIN83_EF_READ_A, AIN84_EF_READ_A, AIN85_EF_READ_A, AIN86_EF_READ_A, AIN87_EF_READ_A, AIN88_EF_READ_A, AIN89_EF_READ_A, AIN90_EF_READ_A, AIN91_EF_READ_A, AIN92_EF_READ_A, AIN93_EF_READ_A, AIN94_EF_READ_A, AIN95_EF_READ_A, AIN96_EF_READ_A, AIN97_EF_READ_A, AIN98_EF_READ_A, AIN99_EF_READ_A, AIN100_EF_READ_A, AIN101_EF_READ_A, 7000, 7002, 7004, 7006, 7008, 7010, 7012, 7014, 7016, 7018, 7020, 7022, 7024, 7026, 7028, 7030, 7032, 7034, 7036, 7038, 7040, 7042, 7044, 7046, 7048, 7050, 7052, 7054, 7056, 7058, 7060, 7062, 7064, 7066, 7068, 7070, 7072, 7074, 7076, 7078, 7080, 7082, 7084, 7086, 7088, 7090, 7092, 7094, 7096, 7098, 7100, 7102, 7104, 7106, 7108, 7110, 7112, 7114, 7116, 7118, 7120, 7122, 7124, 7126, 7128, 7130, 7132, 7134, 7136, 7138, 7140, 7142, 7144, 7146, 7148, 7150, 7152, 7154, 7156, 7158, 7160, 7162, 7164, 7166, 7168, 7170, 7172, 7174, 7176, 7178, 7180, 7182, 7184, 7186, 7188, 7190, 7192, 7194, 7196, 7198, 7200, 7202,

Register Name	Start Address	Type	Access
AIN100_EF_READ_A, AIN103_EF_READ_A, AIN104_EF_READ_A, AIN105_EF_READ_A, AIN106_EF_READ_A, AIN107_EF_READ_A, AIN108_EF_READ_A, AIN109_EF_READ_A, AIN110_EF_READ_A, AIN111_EF_READ_A, AIN112_EF_READ_A, AIN113_EF_READ_A, AIN114_EF_READ_A, AIN115_EF_READ_A, AIN116_EF_READ_A, AIN117_EF_READ_A, AIN118_EF_READ_A, AIN119_EF_READ_A, AIN120_EF_READ_A, AIN121_EF_READ_A, AIN122_EF_READ_A, AIN123_EF_READ_A, AIN124_EF_READ_A, AIN125_EF_READ_A, AIN126_EF_READ_A, AIN127_EF_READ_A, AIN128_EF_READ_A, AIN129_EF_READ_A, AIN130_EF_READ_A, AIN131_EF_READ_A, AIN132_EF_READ_A, AIN133_EF_READ_A, AIN134_EF_READ_A, AIN135_EF_READ_A, AIN136_EF_READ_A, AIN137_EF_READ_A, AIN138_EF_READ_A, AIN139_EF_READ_A, AIN140_EF_READ_A, AIN141_EF_READ_A, AIN142_EF_READ_A, AIN143_EF_READ_A, AIN144_EF_READ_A, AIN145_EF_READ_A, AIN146_EF_READ_A, AIN147_EF_READ_A, AIN148_EF_READ_A	7206, 7208, 7210, 7212, 7214, 7216, 7218, 7220, 7222, 7224, 7226, 7228, 7230, 7232, 7234, 7236, 7238, 7240, 7242, 7244, 7246, 7248, 7250, 7252, 7254, 7256, 7258, 7260, 7262, 7264, 7266, 7268, 7270, 7272, 7274, 7276, 7278, 7280, 7282, 7284, 7286, 7288, 7290, 7292, 7294, 7296	UINT32	W

File I/O

Name	Start Address	Type	Access
FILE_IO_PATH_WRITE_LEN_BYTES	60640	UINT32	W

FILE_IO_PATH_WRITE_LEN_BYTES - Address: 60640

Write the length (in bytes) of the file path or directory to access.

- Data type: UINT32 (type index = 1)
- Write-only
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0134

SPI

Name	Start Address	Type	Access
SPI_GO	5007	UINT16	W

spi_go - Address: 5007

Write 1 to begin the configured SPI transaction.

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)

Serial

Name	Start Address	Type	Access
ASYNCH_NUM_BYTES_TX	5440	UINT16	R/W
ASYNCH_ENABLE	5400	UINT16	R/W

ASYNCH_NUM_BYTES_TX - Address: 5440

The number of bytes to be transmitted after writing to GO. Max is 256.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)

ASYNCH_ENABLE - Address: 5400

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)

5.0 USB [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

Communication Protocol: **Modbus TCP**

Connector Type: **USB-B Receptacle**

Compatible: **USB 1.1+**

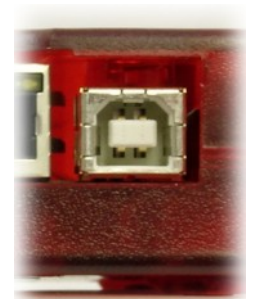
Max Cable Length: **5 meters**

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

Power is supplied to the T-series device through the 5V USB connection. If the Ethernet or WiFi 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 T-series Device

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



functions and programming flexibility.

Power Considerations

USB ground is connected to the T-series device's ground (GND), since standard USB is non-isolated. USB ground is generally the same as the ground of the PC chassis and AC mains.

If electrical isolation between the T-series device and host is desired, use Ethernet (or WiFi) instead, or add a [USB isolator](#).

Any host port or self-powered hub port should provide 4.75 - 5.25 volts at any current up to 500 mA, per USB specifications. We have found that some are not up to specification and at higher currents the supplied voltage drops below 4.75 volts. See [Appendix A-5](#) for details on power supply requirements.



If designing your own driver...

The LabJack vendor ID is 0x0CD5.

- T4 product ID is 0x0004.
- T7 product ID 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, 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 T-series device to the host.

6.0 Ethernet [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

Communication Protocol: **Modbus TCP**

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

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

PoE (Power over Ethernet): **Not built-in. See PoE App Note.**

Max Cable Length: **100 meters typical**

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

The T-series devices each have 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.



Ports

T-series devices will respond to connections or packets on following ports:

- 502 (TCP) - Responds to Modbus TCP packets received over this port.
- 702 (TCP) - Sends spontaneous data such as stream to devices connected to this port.
- 52362 (UDP) - Responds to Modbus UDP packets received over this port. Usually used to search for T-series devices on a network.

If you need a wireless connection instead of Ethernet, you can buy a wireless bridge. Connect the T-series device to the bridge and the bridge will connect to the wireless network. Find more information in the [Convert Ethernet to WiFi App Note](#).

Config (_DEFAULT) Registers

Use the following _DEFAULT registers to configure Ethernet:

Ethernet Config Registers			
Name	Start Address	Type	Access
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_APPLY_SETTINGS	49190	UINT32	W

ETHERNET_IP_DEFAULT - Address: 49150

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

- Data type: UINT32 (type index = 1)
- Readable and writable

ETHERNET_SUBNET_DEFAULT - Address: 49152

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

- Data type: UINT32 (type index = 1)
- Readable and writable

ETHERNET_GATEWAY_DEFAULT - Address: 49154

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

- Data type: UINT32 (type index = 1)
- Readable and writable

ETHERNET_DNS_DEFAULT - Address: 49156

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

- Data type: UINT32 (type index = 1)
- Readable and writable

ETHERNET_ALTDNS_DEFAULT - Address: 49158

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

- Data type: UINT32 (type index = 1)
- Readable and writable

ETHERNET_DHCP_ENABLE_DEFAULT - Address: 49160

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

- Data type: UINT16 (type index = 0)
- Readable and writable

ETHERNET_APPLY_SETTINGS - Address: 49190

Writing 1 to this register power-cycles Ethernet. It tells the device to wait 1s before turning off Ethernet and then 500ms before turning it back on.

- Data type: UINT32 (type index = 1)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0103
- T4:
 - Minimum [firmware](#) version: 0.2000

These registers can also be read. Configure the Ethernet [network configurations](#) in Kipling software.

These _DEFAULT registers are non-volatile. Whatever value you write to a _DEFAULT register will be retained through a reboot or power-cycle. New values written to these _DEFAULT registers are not applied until power-up or until a 1 is written to ETHERNET_APPLY_SETTINGS.

Status Registers

Use the following read-only registers to read the status of Ethernet:

Ethernet Status Registers

Name	Start Address	Type	Access
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 - Address: 49100

Read the current IP address of wired Ethernet.

- Data type: UINT32 (type index = 1)
- Read-only

ETHERNET_SUBNET - Address: 49102

Read the current subnet of wired Ethernet.

- Data type: UINT32 (type index = 1)
- Read-only

ETHERNET_GATEWAY - Address: 49104

Read the current gateway of wired Ethernet.

- Data type: UINT32 (type index = 1)
- Read-only

ETHERNET_DNS - Address: 49106

Read the current DNS of wired Ethernet.

- Data type: UINT32 (type index = 1)
- Read-only

ETHERNET_ALTDNS - Address: 49108

Read the current Alt DNS of wired Ethernet.

- Data type: UINT32 (type index = 1)

- Read-only

ETHERNET_DHCP_ENABLE - Address: 49110

Read the current Enabled/Disabled state of Ethernet DHCP.

- Data type: UINT16 (type index = 0)
- Read-only

Config (_DEFAULT) vs. Status Registers

Example: If you write ETHERNET_IP_DEFAULT="192.168.1.207" (you would 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.

Power

The following registers configure Ethernet power:

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

POWER_ETHERNET - Address: 48003

The current ON/OFF state of the Ethernet module.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8600
- T4:
 - Minimum [firmware](#) version: 0.0100

POWER_ETHERNET_DEFAULT - Address: 48053

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8600
- T4:
 - Minimum [firmware](#) version: 0.0100

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.

Isolation

The Ethernet connection on any T-series device has 1500 volts of galvanic isolation. All power supplies shipped by LabJack Corporation with T-series devices have at least 500 volts of isolation.

Note that if you power the T-series device from a USB host/hub, ground from the host/hub is typically connected to upstream USB ground, which often finds its way back to AC mains ground and thus there would be no isolation.

LEDs on the Ethernet jack

Both the green and orange LEDs on the Ethernet jack will illuminate on connection to an active Ethernet cable. The orange LED turns on when an active link is detected and blinks when packets are received/processed. The green LED illuminates when the connection is 100Mbps.



The orange LED is closest to the USB connector.

7.0 WiFi (T7 only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview - T7-Pro Only

Communication Protocol: **Modbus TCP**

Connector Type: **Female RP-SMA**

Transceiver: **2.4 GHz 802.11 b/g** (Compatible with ac, and n routers)

Security: **None or WPA2-PSK (WPA2-Personal)**

Range: **Similar to laptops and other WiFi devices with stock antenna**

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

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](#).

Data Speed: It's possible to get data faster on a T7-Pro using its Ethernet interface instead of its WiFi interface, both for command response and streaming modes. Wireless bridges and access points do not introduce a speed bottleneck, so a bridge is a good way to get fast wireless data from any T-series device. See the [Convert Ethernet to WiFi App Note](#) for setup information.

Port

The T7's WiFi module responds to connections or packets on port 502 for both UDP and TCP.

Config (_DEFAULT) Registers

Use the following _DEFAULT registers to configure WiFi:

WiFi Config Registers

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

WIFI_IP_DEFAULT - Address: 49250

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

- Data type: `UINT32` (type index = 1)
- Readable and writable

WIFI_SUBNET_DEFAULT - Address: 49252

The new subnet of WiFi. Use `WIFI_APPLY_SETTINGS`.

- Data type: `UINT32` (type index = 1)
- Readable and writable

WIFI_GATEWAY_DEFAULT - Address: 49254

The new gateway of WiFi. Use `WIFI_APPLY_SETTINGS`.

- Data type: `UINT32` (type index = 1)
- Readable and writable

WIFI_DHCP_ENABLE_DEFAULT - Address: 49260

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

- Data type: `UINT16` (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.9014

WIFI_SSID_DEFAULT - Address: 49325

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

- Data type: `STRING` (type index = 98)
- Readable and writable

WIFI_PASSWORD_DEFAULT - Address: 49350

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

- Data type: `STRING` (type index = 98)
- Write-only



WIFI_APPLY_SETTINGS - Address: 49400

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

- Data type: UINT32 (type index = 1)
- Write-only

These registers can also be read, except WIFI_PASSWORD_DEFAULT. Configure the WiFi [network configurations](#) in Kipling software.

These _DEFAULT registers are non-volatile. Whatever value you write to a _DEFAULT register will be retained through a reboot or power-cycle. New values written to these _DEFAULT registers are not applied until power-up or until a 1 is written to WIFI_APPLY_SETTINGS.

Status Registers

Use the following read-only registers to read the status of WiFi:

WiFi Status Registers

Name	Start Address	Type	Access
WIFI_IP	49200	UINT32	R
WIFI_SUBNET	49202	UINT32	R
WIFI_GATEWAY	49204	UINT32	R
WIFI_DHCP_ENABLE	49210	UINT16	R
WIFI_SSID	49300	STRING	R
WIFI_STATUS	49450	UINT32	R
WIFI_RSSI	49452	FLOAT32	R

WIFI_IP - Address: 49200

Read the current IP address of WiFi.

- Data type: UINT32 (type index = 1)
- Read-only

WIFI_SUBNET - Address: 49202

Read the current subnet of WiFi.

- Data type: UINT32 (type index = 1)
- Read-only

WIFI_GATEWAY - Address: 49204

Read the current gateway of WiFi.

- Data type: UINT32 (type index = 1)
- Read-only

WIFI_DHCP_ENABLE - Address: 49210

Read the current Enabled/Disabled state of WiFi DHCP.

- Data type: UINT16 (type index = 0)
- Read-only
- T7:
 - Minimum [firmware](#) version: 0.9014

WIFI_SSID - Address: 49300

Read the current SSID (network name) of WiFi

- Data type: STRING (type index = 98)
- Read-only

WIFI_STATUS - Address: 49450

Status Codes: ASSOCIATED = 2900, ASSOCIATING = 2901, ASSOCIATION_FAILED = 2902, UNPOWERED = 2903, BOOTING = 2904, START_FAILED = 2905, APPLYING_SETTINGS = 2906, DHCP_STARTED = 2907, OTHER = 2909

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum [firmware](#) version: 0.7500

Constant	Value
ASSOCIATED	2900
ASSOCIATING	2901
ASSOCIATION_FAILED	2902
UNPOWERED	2903
BOOTING	2904
START_FAILED	2905
APPLYING_SETTINGS	2906
DHCP_STARTED	2907
OTHER	2909

WIFI_RSSI - Address: 49452

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.

- Data type: FLOAT32 (type index = 3)
- Read-only
- T7:
 - Minimum [firmware](#) version: 0.8000

Config (_DEFAULT) vs. Status Registers

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.

Power

The following registers configure WiFi power:

WiFi Power Registers			
Name	Start Address	Type	Access
POWER_WIFI	48004	UINT16	R/W
POWER_WIFI_DEFAULT	48054	UINT16	R/W

POWER_WIFI - Address: 48004
 The current ON/OFF state of the WiFi module.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8600

POWER_WIFI_DEFAULT - Address: 48054
 The ON/OFF state of the WiFi module after a power-cycle to the device. Provided to optionally reduce power consumption.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8600

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.

Updating WiFi Firmware

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.

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:

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

WIFI_VERSION - Address: 60008
 The current firmware version of the WiFi module, if available.

- Data type: FLOAT32 (type index = 3)
- Read-only

WIFI_FIRMWARE_UPDATE_TO_VERSIONX - Address: 49402
 Start an update by using USB or Ethernet to write the desired version to update to.

- Data type: FLOAT32 (type index = 3)
- Write-only
- T7:
 - Minimum [firmware](#) version: 0.9206

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.

Read from `WIFI_FIRMWARE_UPDATE_STATUS` to monitor progress:

Name	Start Address	Type	Access
WIFI_FIRMWARE_UPDATE_STATUS	49454	UINT32	R

WIFI_FIRMWARE_UPDATE_STATUS - Address: 49454
 CONFIGURING = 2920, IN_PROGRESS = 2921, REBOOTING = 2923, UPDATE_SUCCESS = 2924, UPDATE_FAILED = 2925

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum [firmware](#) version: 0.9206

Constant	Value
CONFIGURING	2920
IN_PROGRESS	2921
REBOOTING	2923
UPDATE_SUCCESS	2924
UPDATE_FAILED	2925

CONFIGURING	Time
IN_PROGRESS	2921
REBOOTING	2923
UPDATE_SUCCESS	2924
UPDATE_FAILED	2925

Antenna Details

The T7-Pro ships with a standard RP-SMA 2.4 GHz antenna similar to the [W1030](#), or the [A24-HASM-450](#). To put an antenna further away from the T7-Pro you can use any standard male-female RP-SMA WiFi extension cable.

The connection to the WiFi module on the T7-Pro PCB is made via 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 [Taoglas Limited CAB.622](#), [Emerson 415-0100-150](#), [Laird 1300-00041](#), [Amphenol 336306-14-0150](#), or [Amphenol 336306-12-0150](#).

To search for U.FL to RP-SMA cable options at Digi-Key, 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.

T7-Pro-OEM ships with a simple 30mm U.FL whip antenna such as the [Anaren 66089-2406](#).

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 -75dBm, and the connection will cut off entirely around -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.

Note that the RSSI value you can read (WIFI_RSSI) is only updated when WiFi communication occurs. That is, if you talk to the T7 over USB to read the RSSI value, you will just get the value from the last WiFi communication that occurred.

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.



Network Schemes

Figure 1 demonstrates a basic network diagram where the T7-Pro is connected to the "Office WiFi" network. The T7-Pro can be controlled by either host option (1 or 2). For much more information on Ethernet and WiFi networking see the [Basic Networking & Troubleshooting App Note](#).

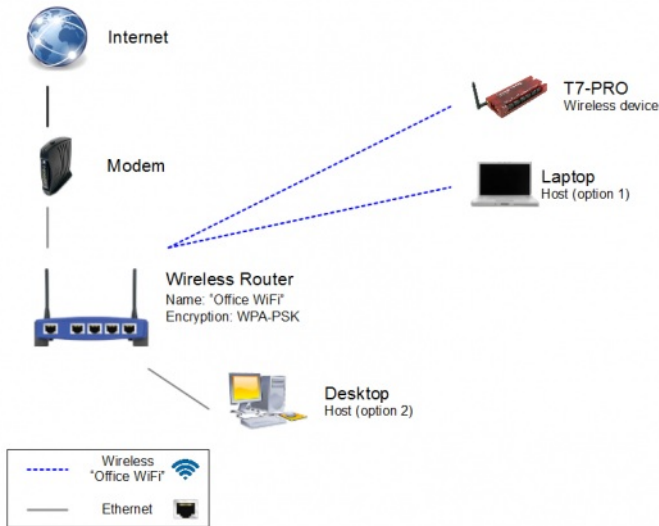


Figure 1. Most common configuration for a home, or small office network.

8.0 LEDs [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

STATUS - The Green LED

The status LED indicates when the T-series device is performing autonomous tasks, such as running an on-board [Lua script](#) or [streaming](#). The LED will blink when a Lua script accesses a Modbus register or when a stream packet is prepared.

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

COMM - The Yellow LED

The primary indicator for packet transfer. If the T-series device is communicating the COMM LED will be blinking. A few blinks after connecting to the PC indicates that the T-series device 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 T-series device 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 if more than 10 packets are being processed per second.



Normal Power-Up Behavior

1. Both LEDs blink rapidly for about 1 second.
2. COMM solid off and STATUS solid on.
3. If USB enumerates, COMM blinks a few times and then stays solid on.

Combined LED Activity

When the LEDs blink together, the T4/T7 is computing checksums.

When the LEDs are alternating, the T4/T7 is copying a firmware image.

LED Configuration

The LED mode can be set using the following registers:

Name	Start Address	Type	Access
POWER_LED	48006	UINT16	R/W
POWER_LED_DEFAULT	48056	UINT16	R/W

POWER_LED - Address: 48006

Sets the LED operation: 0 = Off. Useful for lower power applications. 1 = normal. 2 = Lower power, LEDs will still blink but will normally be off. 3 = Reserved. 4 = Manual, in this mode the LEDs can be user controlled.

- Data type: UINT16 (type index = 0)
- Readable and writable

- T7:
 - Minimum [firmware](#) version: 0.8600
- T4:
 - Minimum [firmware](#) version: 1.0008

POWER_LED_DEFAULT - Address: 48056

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8600
- T4:
 - Minimum [firmware](#) version: 1.1000

When POWER_LED is set to manual mode, the LEDs can be turned on and off using the following registers:

Name	Start Address	Type	Access
LED_COMM	2990	UINT16	W
LED_STATUS	2991	UINT16	W

LED_COMM - Address: 2990

Sets the state of the COMM LED when the LEDs are set to manual, see the POWER_LED register.

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0000
- T4:
 - Minimum [firmware](#) version: 1.0008

Constant	Value
Off	0
On	1

LED_STATUS - Address: 2991

Sets the state of the STATUS LED when the LEDs are set to manual, see the POWER_LED register.

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0000
- T4:
 - Minimum [firmware](#) version: 1.0008

Constant	Value
Off	0
On	1

9.0 VS, Power Supply [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

VS and Power Supply Specifications By Device



T4

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

Device Supply Current: **210 mA Max**

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

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

VS Voltage: **Equal to Supply Voltage**

VS Max Current: **290 mA (500 mA - Device Supply Current)**

T7

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

Device Supply Current: **300 mA Max**

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

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

VS Voltage: **Equal to Supply Voltage**

VS Max Current: **200 mA (500 mA - Device Supply Current)**

VS Terminals

The supply voltage (see below) goes through some protection circuitry and then is presented on the 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:

```
Max total current = 500mA - DeviceSupplyCurrent
```

...so if the T4/T7 needs 300mA to run, that leaves 200mA available from the VS terminals.

The voltage on VS can be noisy and can change unexpectedly. Circuits that are sensitive to changing or noisy supply voltage, such as bridge circuits, should not be supplied from VS. The voltage on VS will also drop as higher amounts of current are drawn by peripherals powered by the T4/T7 due to the inline 0.1 ohm resistors R15 and R21 which allow for the device's current draw to be measured.

Measuring Current Draw

One way to measure how much current the T4/T7 is drawing is by measuring the voltage across R15. R15 is a 0.1 ohm resistor, so if you measure 0.025 volts, that means the current through the resistor is 250 mA. R15 is a large resistor located on the top of the PCB just behind the green LED. To measure the voltage across R15, connect the positive lead of your meter to the test point "Vhost" and connect the negative lead of your meter to the test point "Vs".

R15 is in series with the 5 volt supply from the USB connector. If powering from J5 (see "Alternate Power Supply" in the [OEM section](#)) use R21 instead. The "Vs" test point is also the negative for R21, but there is no positive test point so you just have to touch the upstream side of R21.

Note that the "Vs" test point is actually the Vsupply bus described under "J5 - Alternate Power Supply" in the [OEM section](#), and technically not exactly the same as the "VS" bus documented in this section.

Power Supply

Power supply 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).
- Battery with solar panel (e.g. Anker 79AN15K-BA with 71ANSCP-B145A).
- Pigtail a cable with a USB-B connector to get at the red and black wires, and make some sort of custom cable for your 5V power supply.

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 information about Power over Ethernet (PoE) see the [PoE App Note](#).

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	VA-PSU-	JX-B0520C-1-

Kingdom	UK1	JX-B0520D-1-
Australia	-	B
China	-	JX-B0520H-1-
		B

Note that the JX-B0520 supply is rated for 0 to 40 deg C operation.

Some VS Terminals Not Working?

We sometimes get a report that some VS terminals on a device are not working, but this is almost never a real problem. If the device has LED activity or you can talk to the device, the VS terminals almost certainly do have a reasonable voltage on them. The problem is sometimes using a bad DMM (check if it has a low battery), but the most common problem is touching DMM leads to the screw heads of loose terminals. The screw head is often a valid contact point when the screw terminal is fully clamped down, but the only guaranteed valid connection is to clamp a conductor securely inside the screw terminals.

10.0 SGND and GND [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

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 T4 or T7.



See the AIN, DAC, and Digital I/O [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.

For more information about grounding, see the [14.0 AIN](#), [15.0 DAC](#), and [13.0 Digital I/O](#) sections.

The max total current that can be sunk into GND is:

```
Max total current = 500mA - DeviceSupplyCurrent
```

For example, 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. For information about device supply current, see [9.0 VS, Power Supply](#).

11.0 SPC [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

The SPC terminal has several uses:

- Outputs diagnostic timing signals while streaming (see [Stream Section](#) for details)
- Can be used to force special startup behavior.



Startup Behavior

To force special startup behavior, securely install a short jumper wire from SPC to one of the following digital I/O lines as described below. The jumper needs to be installed before reset, so make sure the jumper is securely clamped in SPC and the given FIO terminal, then power up the device.

T4

- **FIO4:** Force boot to main firmware (internal) image. Used to boot the internal firmware even if its checksum is bad. Lua scripts will not be loaded during boot up when FIO4 is connected to SPC.
- **FIO5:** Force copy of backup image to overwrite internal image. Used to load the external firmware even if its checksum is bad.
- **FIO6:** Factory reset. Sets the start up configuration to factory settings. Disables Lua script at startup.
- **FIO7:** Load emergency image. This option loads a firmware image with minimal functionality (similar to Windows safe-mode). Used to recover from firmware corruption or bugs. The update process is about all that can be done while in this mode.

- **FIO0:** Force boot to main firmware (internal) image. Used to boot the internal firmware even if its checksum is bad. Lua scripts will not be loaded during boot up when FIO0 is connected to SPC.
- **FIO1:** Force copy of backup image to overwrite internal image. Used to load the external firmware even if its checksum is bad.
- **FIO2:** Factory reset. Sets the start up configuration to factory settings. Disables Lua script at startup.
- **FIO3:** Load emergency image. This option loads a firmware image with minimal functionality (similar to Windows safe-mode). Used to recover from firmware corruption or bugs. The update process is about all that can be done while in this mode.

If the device has become unresponsive, the easiest order is:

1. Factory reset
2. Force copy of backup image to overwrite internal image.
3. Load emergency image.

The T-Series devices have two different firmware images. They have a "primary firmware image" which is synonymous to main which is the firmware image used when the device is working properly. The primary firmware image (when being used) exists in the microcontroller's internal flash (for execution) as well as on the external flash chip as a backup image. The second firmware image is an emergency image that implements a minimal amount of features. This image can be loaded using the SPC terminal and allows users to forcefully load a new version of firmware onto the device.

12.0 200uA and 10uA (T7 Only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview - T7 Only

The T7 has 2 fixed current source terminals useful for measuring resistance (thermistors, RTDs, resistors). The 10UA terminal provides approximately 10 μ A and the 200UA terminal provides approximately 200 μ A, but the actual values should be read from the calibration constants, or better yet measured in real-time using a fixed shunt resistor.



Using the equation $V=IR$, with a known current and voltage, it is possible to calculate the resistance of the item in question. Figure 12-1 shows a simple setup measuring 1 resistor.

The factory value of each current source is noted during calibration and stored with the calibration constants on the device. These can be viewed using the [Device Info tab](#) in Kipling, or read programmatically. Note that these are fixed constants stored during calibration, not some sort of real-time readings.

To read the constants, read from the following registers:

Constant Current Sources			
Name	Start Address	Type	Access
CURRENT_SOURCE_200UA_CAL_VALUE	1902	FLOAT32	R
CURRENT_SOURCE_10UA_CAL_VALUE	1900	FLOAT32	R

CURRENT_SOURCE_200UA_CAL_VALUE - Address: 1902
 Fixed current source value in Amps for the 200UA terminal. This value is stored during factory calibration, it is not a current reading. Using the equation $V=IR$, with a known current and voltage, it is possible to calculate resistance of RTDs.

- Data type: FLOAT32 (type index = 3)
- Read-only

CURRENT_SOURCE_10UA_CAL_VALUE - Address: 1900
 Fixed current source value in Amps for the 10UA terminal. This value is stored during factory calibration, it is not a current reading. Using the equation $V=IR$, with a known current and voltage, it is possible to calculate resistance of RTDs.

- Data type: FLOAT32 (type index = 3)
- Read-only

Example:

To read the factory 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.

Examples Of Measuring Resistance

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.

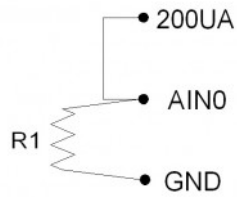


Figure 12-1

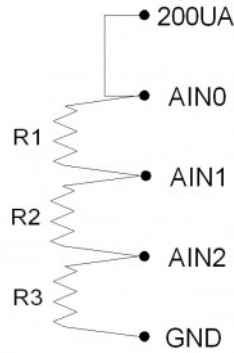


Figure 12-2

Figure 12-1 shows a simple setup measuring 1 resistor. If $R_1=3k$, the voltage at AIN0 will be 0.6 volts.

Figure 12-2 shows a setup to measure 3 resistors using single-ended analog inputs. If $R_1=R_2=R_3=3k$, 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.

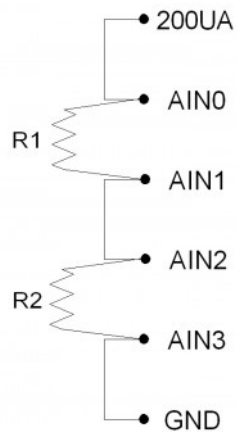


Figure 12-3

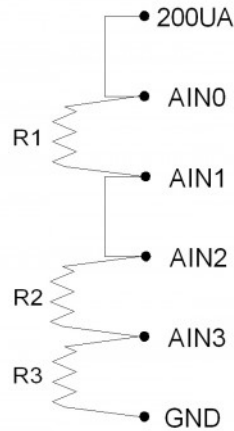


Figure 12-4

Figure 12-3 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 12-4, which shows R1 and R2 measured differentially and R3 measured single-ended.

Remarks

Maximum load resistance: The current sources can drive about 3 volts max, thus limiting the maximum load resistance to about $300\text{ k}\Omega$ (10uA) and $15\text{ k}\Omega$ (200uA). Keep in mind that high source resistance could cause settling issues for analog inputs.

Using a fixed resistor to calculate actual current: For some applications the accuracy and temperature coefficient of the current sources is sufficient, but for improvement a fixed resistor can be used as one of the resistors in the figures above. The Y1453-100 and Y1453-1.0K from Digi-Key have excellent accuracy and very low tempco. By measuring the voltage across one of these you can calculate the actual current at any time.

Handling load changes resulting in noise: The current sources are not particularly fast in reacting to load changes. This can show up as noise when rapidly sampling multiple channels using the same current source. Improve behavior by adding a $1\text{ }\mu\text{F}$ ceramic capacitor from the current source to GND and/or increasing settling time.

Temperature coefficients: Figures 12-5 and 12-6 show the typical current source output variation over temperature. Both sources typically have low temperature coefficients at or near 25C . Beyond 25C , the temperature coefficient variation may need to be accounted for, depending on application requirements.

Current Source Temperature Coefficient Vs Temperature

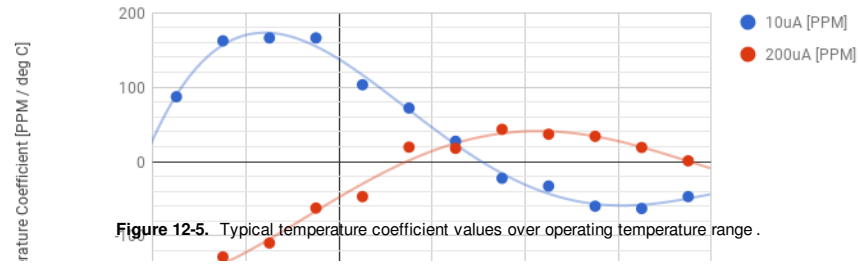


Figure 12-5. Typical temperature coefficient values over operating temperature range .

Current Source Deviation Vs Temperature

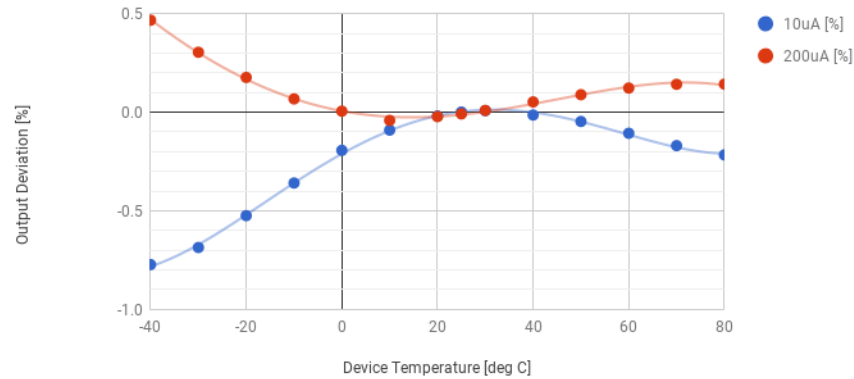


Figure 12-6. Typical current source deviation from 25C output over operating temperature range [1].

Example - PT100 or PT1000 RTD

Assume that R1 in Figure 12-1 is a PT100 RTD. A PT100 RTD is 100 ohms at 0 degC. The response of an RTD is nonlinear, but the linear slope 0.384 ohms/degC works well from about -40 to +150 degC. That leads to the following expression:

$$R = (0.384 * \text{DegC}) + 100$$

...which can be rearranged to:

$$\text{DegC} = (2.604 * R) - 260.4$$

We are determining R by measuring the voltage that results from a known current passed through R, that is $R = V/I$, so we can say:

$$\text{DegC} = (2.604 * V/I) - 260.4$$

This tells us that the slope is 2.604/I and the offset is -260.4. To determine I, you can just use 0.0002 amps, or use the factory calibration value read from CURRENT_SOURCE_200UA_CAL_VALUE, or use a precision fixed resistor as mentioned above to measure I in real time. Assume we read the factory calibration value as 0.000200 amps, and thus use a constant slope of $2.604/0.0002 = 13020$. We can now use the [AIN-EF Offset and Slope feature](#) to apply this slope and offset:

```
AIN0_EF_INDEX = 1           // feature index for Offset and Slope
AIN0_EF_CONFIG_D = 13020.0 // slope
AIN0_EF_CONFIG_E = -260.4  // offset
```

Now reads of AIN0_EF_READ_A will return $(13020.0 * \text{volts}) - 260.4$.

Note that you can come up with your own slope and offset for your temperature region of interest. For example, we made this [Google Spreadsheet](#) and decided that Slope=2.720 (degC/ohm) and Offset=-277.5 works best for the region of 100 to 300 degC.

Note that a PT1000 simply has 10x the response of a PT100 (~3.84 ohms/degC). The offset still works out to -260.4, but the slope is 0.260.

13.0 Digital I/O [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Digital I/O Overview

Basics: An digital I/O is an digital input or output. DIO is a generic name used for all digital I/O.

Common Uses: For wiring information on open-collector signals, driven signals, controlling relays, and mechanical switches, see the [Digital I/O \(App Note\)](#).

How to read and write DIO: See [3.0 Communication](#) for communication basics. Also, LabJack [Kipling's Dashboard](#) tab shows live DIO values.

On this page

- [DIO Summary By Device](#)
- [Usage](#)
- [Individual DIO Channels](#)
- [DIO Bitmask Registers](#)
- [DIO vs. FIO/EIO/CIO/MIO](#)
- [Other Considerations](#)

DIO Extended Features: T-series [DIO Extended Features](#) expose more complicated features such as:

- Timers, Counters, PWM, Quadrature Input, and more.

Digital Communication Protocols: T-series DIO lines can also be used to communicate with a large number of sensors that require the use of various digital communication protocols. The T-series devices implement the following protocols:

- [I2C](#) - Also reference the [I2C Lua Library](#) and Lua [I2C Sensor](#) examples.
- [SPI](#)
- [SBUS](#)
- [1-Wire](#)
- [Asynchronous Serial \(UART\)](#)

DIO Summary By Device

T4

Digital I/O: Up to 16 DIO lines (DIO4-DIO19)

- 8 [flexible I/O](#) (DIO4-DIO11)
- 8 dedicated I/O (DIO12-DIO19)

Logic Level: 3.3V (Adjustable using a [LJTick-LVDDigitalIO](#)).

T7

Digital I/O: Up to 23 DIO lines (DIO0-DIO22)
Logic Level: 3.3V (Adjustable using a [LJTick-LVDDigitalIO](#)).

Usage

There are two basic ways to use DIO:

1. Read or write [individual DIO channels](#) one-at-a-time. Individual DIO channels are automatically configured.
2. Read or write multiple DIO channels at once with the [DIO Bitmask Registers](#), which are manually configured.

In addition, digital I/O registers have "DIO" names and alternate "FIO/EIO/CIO/MIO" names. See [DIO vs. FIO/EIO/CIO/MIO](#) for more details.

Individual DIO Channels

Each T-Series device exposes:

- Some DIOs on the screw terminals.
- Additional DIOs on a connector (either a [DB15](#) or a [DB37](#)).

T4

The LabJack T4 has up to 16 built-in digital input/output lines. They can be written/read as registers named DIO4-DIO19.

DIO4-DIO11 are flexible I/O lines. (See 13.1 Flexible I/O.) These are lines that can be configured for analog input or digital input/output:

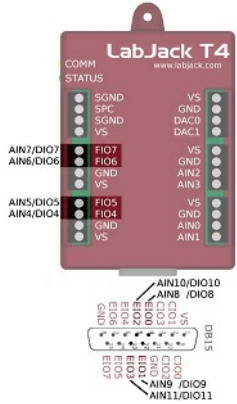


Figure 13.0-1 T4 Flexible I/O

DIO12-DIO19 are dedicated (digital-only) I/O lines:

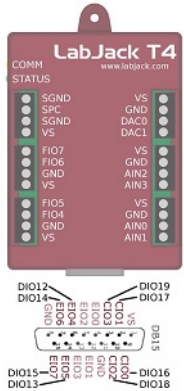


Figure 13.0-2 T4 Dedicated Digital I/O

The registers DIO4-DIO19 can also be accessed using their alternate register names, FIO4-7, EIO0-7, and CIO0-3:

Table 13.0-1. T4 DIO Mapping

	AIN (0-3)			FIO (4-7)				EIO (0-7)							CIO (0-3)					
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
DIO Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Available Digital I/O					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Available Analog I/O	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									

T4 DIO Channel Registers			
Name	Start Address	Type	Access
DIO#(4:19)	2004	UINT16	R/W
DIO#(4:19) - Starting Address: 2004 Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High. <ul style="list-style-type: none"> Data type: UINT16 (type index = 0) Readable and writable 			
Expanded Names		Addresses	
DIO4, DIO5, DIO6, DIO7, DIO8, DIO9, DIO10, DIO11, DIO12, DIO13, DIO14, DIO15, DIO16, DIO17, DIO18, DIO19 Show All		2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019 Show All	

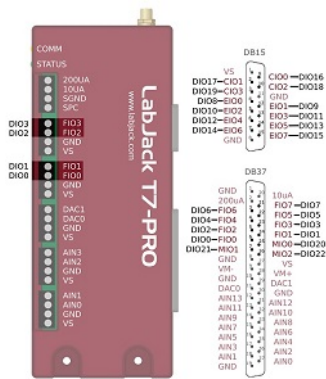


Figure 13.0-3 T7 Digital I/O

The LabJack T7 has 23 built-in digital input/output lines. They can be written/read as registers named DIO0-DIO22. They can also be accessed using their alternate register names: FIO0-7, EIO0-7, CIO0-3, and MIO0-2:

Table 13.0-2. T7 DIO Mapping

	FIO (0-7)							EIO (0-7)							CIO (0-3)			MIO (0-2)					
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	0	1	2
	DIO																						
DIO Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Available Digital I/O	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T7 DIO Channel Registers			
Name	Start Address	Type	Access
DIO#(0:22)	2000	UINT16	R/W
DIO#(0:22) - Starting Address: 2000			
Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.			
<ul style="list-style-type: none"> Data type: UINT16 (type index = 0) Readable and writable 			
Expanded Names		Addresses	
DIO0, DIO1, DIO2, DIO3, DIO4, DIO5, DIO6, DIO7, DIO8, DIO9, DIO10, DIO11, DIO12, DIO13, DIO14, DIO15, DIO16, DIO17, DIO18, DIO19, DIO20, DIO21, DIO22 Show All		2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022 Show All	

Example:

1. To set DIO4 to a digital input and determine if the channel is high or low, read register DIO4. The result will either 1 or 0.
2. To set DIO4 to a digital output and set it high or low, write either 1 or 0 to DIO4.

DIO Bitmask Registers

The digital I/O bitmask registers allow for the direction (input or output) and state (high or low) of multiple digital I/O lines to be set during a single communication packet. Each bit in the value written to these registers corresponds to an individual I/O line on the device. The number of valid bits in the bitmask depends on which device is being used. To see which bits are valid for each device, see the above reference tables [13.0-1 T4 DIO Mapping](#) and [13.0-2 T7 DIO Mapping](#).

DIO Bitmask Registers			
Name	Start Address	Type	Access
DIO_INHIBIT	2900	UINT32	R/W
DIO_DIRECTION	2850	UINT32	R/W
DIO_STATE	2800	UINT32	R/W
DIO_INHIBIT - Address: 2900			
A single binary-encoded value where each bit determines whether _STATE, _DIRECTION or _ANALOG_ENABLE writes affect that bit of digital I/O. 0=Default=Affected, 1=Ignored.			
<ul style="list-style-type: none"> Data type: UINT32 (type index = 1) Readable and writable Default value: 0 			
Constant Value			
Affected	0		
Ignored	1		
DIO_DIRECTION - Address: 2850			
Read or write the direction of all digital I/O in a single binary-encoded value. 0=Input and 1=Output. Writes are filtered by the value in DIO_INHIBIT.			

- Data type: UINT32 (type index = 1)
- Readable and writable

Constant	Value
Input	0
Output	1

DIO_STATE - Address: 2800

Read or write the state of all digital I/O in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. Writes are filtered by the value in DIO_INHIBIT.

- Data type: UINT32 (type index = 1)
- Readable and writable

Constant	Value
Low	0
High	1

T4 only:

The lower four bits of these DIO bitmask registers don't apply to the T4.

Writing DIO Bitmask Registers

DIO_INHIBIT, DIO_DIRECTION, and DIO_STATE should typically be written together. Each true bit in DIO_INHIBIT prevents a corresponding bit in DIO_DIRECTION and DIO_STATE from being modified. For more details about the DIO_INHIBIT register, see the examples section below, as well as the manual configuration section of the [flexible I/O](#) page.

The [LJM multiple value functions](#) provide an easy way to write DIO_INHIBIT, DIO_DIRECTION, and DIO_STATE in a single packet.

Write Example:

To configure DIO4 and DIO5 as a digital outputs set to high, do the following:

1. Build a bitmask based on the DIO channel numbers being controlled. For this example we are controlling DIO channels 4 and 5. All of the following values are equivalent:
 $(1 \ll 4) | (1 \ll 5)$, $(2^4 + 2^5)$, and $16 + 32$ are all equal to `0b00110000` , `0x30` , and `48` .
2. Subtract the bitmask value `0x30` from a value with 23 bits of "1"s (`0x7FFFFFF`) to calculate the value that needs to be written to the DIO_INHIBIT register:
 $0x7FFFFFF - (1 \ll 4) | (1 \ll 5)$ which equals `0b11111111111111111001111` , `0x7FFFCF` , or `8388559`
3. Write the bitmask value `0x7FFFCF` to the DIO_INHIBIT register to inhibit all DIO channels except 4 and 5.
4. Write the bitmask value `0x30` to the DIO_DIRECTION register to set the I/O lines as output.
5. Write the bitmask value `0x30` to the DIO_STATE register to have the two I/O lines output 3.3V.

Reading DIO Bitmask Registers

The typical workflow for reading the DIO Bitmask Registers is to only read DIO_STATE. This is because DIO_INHIBIT and DIO_DIRECTION are typically known.

DIO vs. FIO/EIO/CIO/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](#).

Alternate Digital Channel Names

The following shows the alternate DIO channel registers names:

DIO Name	Alternate Name	
DIO0	FIO0	T7 only
DIO1	FIO1	T7 only
DIO2	FIO2	T7 only
DIO3	FIO3	T7 only
DIO4	FIO4	
DIO5	FIO5	

	DIO6	FIO6	
	DIO7	FIO7	
	DIO8	EIO0	
	DIO9	EIO1	
	DIO10	EIO2	
EIO	DIO11	EIO3	
	DIO12	EIO4	
	DIO13	EIO5	
	DIO14	EIO6	
	DIO15	EIO7	
	DIO16	CIO0	
CIO	DIO17	CIO1	
	DIO18	CIO2	
	DIO19	CIO3	
	DIO20	MIO0	T7 only
MIO	DIO21	MIO1	T7 only
	DIO22	MIO2	T7 only

Digital I/O

Name	Start Address	Type	Access
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) - Starting Address: 2000

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

- Data type: UINT16 (type index = 0)
- Readable and writable

Expanded Names	Addresses
FIO0, FIO1, FIO2, FIO3, FIO4, FIO5, FIO6, FIO7 Show All	2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007 Show All

EIO#(0:7) - Starting Address: 2008

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

- Data type: UINT16 (type index = 0)
- Readable and writable

Expanded Names	Addresses
EIO0, EIO1, EIO2, EIO3, EIO4, EIO5, EIO6, EIO7 Show All	2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015 Show All

CIO#(0:3) - Starting Address: 2016

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

- Data type: UINT16 (type index = 0)
- Readable and writable

Expanded Names	Addresses
CIO0, CIO1, CIO2, CIO3 Show All	2016, 2017, 2018, 2019 Show All

MIO#(0:2) - Starting Address: 2020

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

- Data type: UINT16 (type index = 0)
- Readable and writable

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

Example:

Writing 0 to FIO4 (address 2004) is the same as writing 0 to DIO4 (address 2004).

FIO/EIO/CIO/MIO Bitmask Registers

The following FIO/EIO/CIO/MIO bitmask registers are similar to the above [DIO bitmask registers](#). However, instead of having a dedicated register designated for the inhibit bits, the inhibit bits are the upper 8-bits of each register.

T4 only:

- Lower order bits of the FIO_STATE and FIO_DIRECTION have no affect on the T4.
- The MIO_STATE and MIO_DIRECTION registers have no affect on the T4.

FIO/EIO/CIO/MIO State Bitmask Registers

Name	Start Address	Type	Access
FIO_STATE	2500	UINT16	R/W

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

FIO_STATE - Address: 2500

Read or write the state of the 8 bits of FIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

- Data type: UINT16 (type index = 0)
- Readable and writable
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0000
- T4:
 - Minimum [firmware](#) version: 0.2000

EIO_STATE - Address: 2501

Read or write the state of the 8 bits of EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

- Data type: UINT16 (type index = 0)
- Readable and writable
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0000
- T4:
 - Minimum [firmware](#) version: 0.2000

CIO_STATE - Address: 2502

Read or write the state of the 4 bits of CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

- Data type: UINT16 (type index = 0)
- Readable and writable
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0000
- T4:
 - Minimum [firmware](#) version: 0.2000

MIO_STATE - Address: 2503

Read or write the state of the 3 bits of MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

- Data type: UINT16 (type index = 0)
- Readable and writable
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0000
- T4:
 - Minimum [firmware](#) version: 0.2000

Example:

To read the digital state of all FIO lines in a bitmask, read FIO_STATE. If the result is 0b11111011, FIO2 is logic low and all other FIO lines are logic high.

FIO/EIO/CIO/MIO Direction Bitmask Registers

Name	Start Address	Type	Access
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 - Address: 2600

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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.9402
- T4:
 - Minimum [firmware](#) version: 0.2000

EIO_DIRECTION - Address: 2601

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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.9402
- T4:
 - Minimum [firmware](#) version: 0.2000

CIO_DIRECTION - Address: 2602

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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.9402

- T4:
 - Minimum [firmware](#) version: 0.2000

MIO_DIRECTION - Address: 2603

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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.9402
- T4:
 - Minimum [firmware](#) version: 0.2000

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.

Combination FIO/EIO/CIO/MIO State Registers

These registers are a combination of the FIO/EIO/CIO/MIO State registers.

Combination Direction Bitmask Registers

Name	Start Address	Type	Access
FIO_EIO_STATE	2580	UINT16	R/W
EIO_CIO_STATE	2581	UINT16	R/W
CIO_MIO_STATE	2582	UINT16	R/W

FIO_EIO_STATE - Address: 2580

Read or write the state of the 16 bits of FIO-EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

- Data type: UINT16 (type index = 0)
- Readable and writable
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0005
- T4:
 - Minimum [firmware](#) version: 0.2000

EIO_CIO_STATE - Address: 2581

Read or write the state of the 12 bits of EIO-CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. As of firmware 1.0172, MIO states are included in the upper nibble of the CIO byte.

- Data type: UINT16 (type index = 0)
- Readable and writable
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0005
- T4:
 - Minimum [firmware](#) version: 0.2000

CIO_MIO_STATE - Address: 2582

Read or write the state of the 12 bits of CIO-MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

- Data type: UINT16 (type index = 0)
- Readable and writable
- This register may be streamed
- T7:
 - Minimum [firmware](#) version: 1.0172
- T4:
 - Minimum [firmware](#) version: 0.2000

Other Considerations

Specifications

See [Appendix A-2](#) for specs including:

- Low Level Input Voltage
- High Level Input Voltage
- Hysteresis Voltage
- Maximum Input Voltage
- Output Low Voltage
- Output High Voltage
- Short Circuit Current
- Output Impedance

Streaming DIO

For details about which DIO registers can be streamed look at section [3.2 Stream Mode](#). In short, only the FIO/EIO/CIO/MIO [State registers](#) can be streamed because stream data is transferred as 16-bit values.

Electrical Overview

All digital I/O on T-series devices 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).

If a DIO terminal is at about 3.3 volts, and you are not sure if it is set to input or output-high, a couple ways to tell are:

1. Look for a slight change on a terminal with nothing connected except a DMM. For example, a DMM measurement of an input might show 3.30V whereas that same terminal as output-high reads 3.31V.
2. Add a load resistor. If you add a 100k from FIO7 to GND, the terminal should measure about 1.6V for input and 3.3V for output-high.

See [Appendix A-2](#) 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 [13.2 DIO Extended Feature](#).)

Power-up Defaults

The default condition of the digital I/O can be configured [using Kipling](#). 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. For more information [see this forum topic](#).

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 [Appendix A-2](#).

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

On-board DACs: The DAC0 and DAC1 channels can be set to 5 volts, providing 2 output lines with such capability.

LabJack LJTick-DigitalOut5V: We sell the [LJTick-DigitalOut5V](#) that converts our 3.3V outputs to 5V outputs.

Logic Buffer IC: 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).

Open-collector: 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. T-series devices have 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.

13.1 Flexible I/O (T4 Only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Flexible I/O Overview - T4 Only

Basics: Flexible I/O are ports, channels, or lines on a LabJack device that may be configured as analog inputs, as digital inputs, or as digital outputs.

Digital I/O: [13.0 Digital I/O](#)

Analog I/O: [14.0 Analog Inputs](#)

Available T4 Flexible I/O Channels

As Figure 13.1-1 shows below, the LabJack T4 has 8 flexible I/O lines:

- Four screw terminals labeled FIO4 through FIO7 (also named as DIO4-DIO7 and as AIN4-AIN7)
- Four [DB15](#) pins labeled EIO0 through EIO3 (also named as DIO8-DIO11 and as AIN8-AIN11)

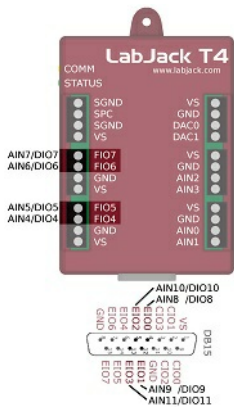


Figure 13.1-1 T4 Flexible I/O

The flexible I/O lines are readable/writable as digital I/O using the register names DIO4 through DIO11:

Name	Start Address	Type	Access
DIO#(4:11)	2004	UINT16	R/W
DIO#(4:11) - Starting Address: 2004 Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High. <ul style="list-style-type: none"> Data type: UINT16 (type index = 0) Readable and writable 			
Expanded Names		Addresses	
DIO4, DIO5, DIO6, DIO7, DIO8, DIO9, DIO10, DIO11 Show All		2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011 Show All	

The flexible I/O lines are readable as analog inputs using the register names AIN4 through AIN11:

Name	Start Address	Type	Access
AIN#(4:11)	8	FLOAT32	R
AIN#(4:11) - Starting Address: 8 Returns the voltage of the specified analog input. <ul style="list-style-type: none"> Data type: FLOAT32 (type index = 3) Read-only This register may be streamed 			
Expanded Names		Addresses	
AIN4, AIN5, AIN6, AIN7, AIN8, AIN9, AIN10, AIN11 Show All		8, 10, 12, 14, 16, 18, 20, 22 Show All	

For examples on how to use these registers, see [13.0 Digital I/O](#) and [14.0 Analog Inputs](#).

Flexible I/O Auto-Configuration

Flexible I/O lines will be auto-configured in some situations. Flexible I/O can also be manually configured, as described below.

Digital Inputs - Always Auto-Configured

Reading DIO4-DIO11 always auto-configures the given line to be a digital input (before returning the digital state a 1 or 0).

Digital Outputs - Not Always Auto-Configured

Writing DIO4-DIO11 only auto-configures the given line to be a digital output if the line is currently a digital input. If the channel is currently configured as an analog input, the channel will remain configured as an analog input and the write command will be ignored.

To force a flexible I/O line to be a digital output, you can read it as digital, then write to it as digital. Be aware that an analog sensor may be damaged by driving voltage into its output.

Analog Inputs — Always Auto-Configured

Reading from AIN4-AIN11 always auto-configures a channel to be an analog input.

Flexible I/O Manual and Bulk Configuration

The following registers can configure multiple flexible I/O lines at once:

Name	Start Address	Type	Access
DIO_INHIBIT	2900	UINT32	R/W

Name	Start Address	Type	Access
ANALOG_ENABLE			
DIO_DIRECTION	2850	UINT32	R/W
DIO_STATE	2800	UINT32	R/W

DIO_INHIBIT - Address: 2900

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

Constant	Value
Affected	0
Ignored	1

DIO_ANALOG_ENABLE - Address: 2880

Read or write the analog configuration of all digital I/O in a single binary-encoded value. 1=Analog mode and 0=Digital mode. When switching from analog to digital, the lines will be set to input. Writes are filtered by the value in `DIO_INHIBIT`.

- Data type: UINT32 (type index = 1)
- Readable and writable

Constant	Value
Digital mode	0
Analog mode	1

DIO_DIRECTION - Address: 2850

Read or write the direction of all digital I/O in a single binary-encoded value. 0=Input and 1=Output. Writes are filtered by the value in `DIO_INHIBIT`.

- Data type: UINT32 (type index = 1)
- Readable and writable

Constant	Value
Input	0
Output	1

DIO_STATE - Address: 2800

Read or write the state of all digital I/O in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. Writes are filtered by the value in `DIO_INHIBIT`.

- Data type: UINT32 (type index = 1)
- Readable and writable

Constant	Value
Low	0
High	1

To configure multiple flexible I/O lines, set the relevant bits of `DIO_INHIBIT`, `DIO_ANALOG_ENABLE`, `DIO_DIRECTION`, and `DIO_STATE`—where the relevant bits are the same as the DIO channel numbers. Examples:

- To configure DIO4 (screw terminal FIO4), set bit 4 of `DIO_INHIBIT`, `DIO_ANALOG_ENABLE`, etc.
- To configure DIO5 (screw terminal FIO5), set bit 5 of `DIO_INHIBIT`, `DIO_ANALOG_ENABLE`, etc.
- To configure DIO8 (DB15 pin EIO0), set bit 8 of `DIO_INHIBIT`, `DIO_ANALOG_ENABLE`, etc.

To configure digital input(s):

- Set the relevant bit(s) of `DIO_INHIBIT` to 0
- Set the relevant bit(s) of `DIO_ANALOG_ENABLE` to 0
- Set the relevant bit(s) of `DIO_DIRECTION` to 0
- Read the relevant DIO register(s) or read `DIO_STATE`

For example, to configure DIO4 (screw terminal FIO4) as a digital input:

- Set bit 4 of `DIO_INHIBIT` to 0
- Set bit 4 of `DIO_ANALOG_ENABLE` to 0
- Set bit 4 of `DIO_DIRECTION` to 0
- Read DIO4 or read bit 4 of `DIO_STATE`

To configure digital output(s):

- Set the relevant bit(s) of `DIO_INHIBIT` to 0
- Set the relevant bit(s) of `DIO_ANALOG_ENABLE` to 0
- Set the relevant bit(s) of `DIO_DIRECTION` to 1
- Write the relevant DIO register(s) or write to `DIO_STATE`

For example, to configure DIO4 (screw terminal FIO4) as a digital output:

- Set bit 4 of `DIO_INHIBIT` to 0
- Set bit 4 of `DIO_ANALOG_ENABLE` to 0
- Set bit 4 of `DIO_DIRECTION` to 0
- Write to DIO4 or write bit 4 of `DIO_STATE`

To configure analog input(s):

- Set the relevant bit(s) of `DIO_INHIBIT` to 0
- Set the relevant bit(s) of `DIO_ANALOG_ENABLE` to 1
- Read the relevant AIN register(s)

For example, to configure AIN4 (screw terminal FIO4) as a analog input:

- Set bit 4 of DIO_INHIBIT to 0
- Set bit 4 of DIO_ANALOG_ENABLE to 1
- Read AIN4

Tips for Constructing Bitmasks

The DIO_INHIBIT value for allowing a write command to only affect DIO4 and DIO5 is as follows:

`0x7FFFFF - (1<<4)|(1<<5)` which equals `0b1111111111111111001111` , `0x7FFFCF` , or `8388559` .

After writing `0x7FFFCF` to the DIO_INHIBIT register, the DIO_ANALOG_ENABLE value for configuring DIO4 and DIO5 as analog inputs is as follows:

`(1<<4)|(1<<5)` which equals `0b110000` , `0x30` , or `48` .

Flexible I/O Pull-Up

Most users will not need to use this register.

Name	Start Address	Type	Access
DIO_PULLUP_DISABLE	2890	UINT32	R/W

DIO_PULLUP_DISABLE - Address: 2890

This register will prevent pullups from being enabled on lines set to digital input. This is a binary coded value where bit 0 represent FIO0 and bit 11 represents EIO3. 1 = pullup disabled, 0 = pullup enabled. This register only affects flex-lines which can be configured as analog or digital. This register is not affected by the inhibit register.

- Data type: UINT32 (type index = 1)
- Readable and writable

13.2 DIO Extended Features [T-Series Datasheet]

[Log in or register](#) to post comments

DIO Extended Features Overview

Basics: DIO Extended Features, commonly referred to as "DIO-EF", allow T-Series devices to measure and generate digital waveforms that are more advanced than logic high or logic low. They expose features such as PWM output for servo motor control, Quadrature input for [incremental/quadrature encoders](#), and more.

Register Numbering: DIO-EFs are configured and used through the DIO#(0:22)_EF registers. The numbering of these registers corresponds with the DIO numbers documented in section [13.0 Digital I/O](#).

Configuration and how to use: The meanings of each of the DIO#_EF_CONFIG registers and DIO#_EF_READ registers changes depending on what DIO-EF index (DIO#_EF_INDEX) is configured, however the general configuration process is the same and is described below. It is helpful to think of DIO-EF features as "sub-systems" that need to be configured before they are started. Once they are started, they can be interacted with by reading the system state and updating system configurations.

DIO-EF System Configurations:

1. Select a feature and determine the number of required DIO lines using table 13.2-1 or 13.2-2.
2. Ensure that the DIO-EF is disabled by writing a 0 to the appropriate DIO#_EF_ENABLE register.
3. If required by the selected DIO-EF feature, configure the [DIO-EF clock source](#).
4. Write the selected feature's index value to the appropriate DIO#_EF_INDEX register.
5. If required by the selected DIO-EF feature, write to the DIO#_EF_OPTIONS register.
6. If required by the selected DIO-EF feature, write to the DIO#_EF_CONFIG registers.
7. Enable the selected DIO-EF feature by writing a 1 to the appropriate DIO#_EF_ENABLE register.

Once a DIO-EF has been started, it can be interacted with using the following registers

- If the selected DIO-EF produces data, read the results from the DIO#_EF_READ registers.
 - (E.g., if DIO6 is configured as an [Interrupt Counter](#), you can read the current count from DIO6_EF_READ_A.)
- If the selected DIO-EF can be updated on the fly, write to the DIO#_EF_CONFIG registers.
 - (E.g., if DIO0 is configured as a [PWM Out](#), you can update the duty cycle by writing to DIO0_EF_CONFIG_A.)

Available DIO-EF Features: A brief overview of each of the features is as follows:

- **0:** [PWM Out](#)
- **1:** [PWM Out with Phase](#)
- **2:** [Pulse Out](#)
- **3,4:** [Frequency In](#)
- **5:** [Pulse Width In](#)
- **6:** [Line-to-Line In](#)
- **7:** [High-Speed Counter](#)

- [8: Interrupt Counter](#)
- [9: Interrupt Counter with Debounce](#)
- [10: Quadrature In](#)
- [11: Interrupt Frequency In](#)
- [12: Conditional Reset](#)

Kipling Walkthroughs: Kipling's [Register Matrix](#) can be used to perform DIO-EF features. Some examples:

- [Configuring & Reading a Counter](#)
- [Configuring & Reading Frequency](#)
- [Configuring a PWM Output](#)

DIO-EF Enable/Disable

This register is used to disable a DIO-EF feature (in order to configure it) and also used to start or enable the DIO-EF subsystem.

A DIO-EF doesn't always need to be disabled for it to be configured, depending on the DIO-EF being enabled.

Name	Start Address	Type	Access				
DIO#(0:22)_EF_ENABLE	44000	UINT32	R/W				
<p>DIO#(0:22)_EF_ENABLE - Starting Address: 44000</p> <p>1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.</p> <ul style="list-style-type: none"> • Data type: UINT32 (type index = 1) • Readable and writable • T7: <ul style="list-style-type: none"> ◦ Minimum firmware version: 0.8800 <p>Expanded Names</p> <table border="1"> <thead> <tr> <th>Expanded Names</th> <th>Addresses</th> </tr> </thead> <tbody> <tr> <td>DIO0_EF_ENABLE, DIO1_EF_ENABLE, DIO2_EF_ENABLE, DIO3_EF_ENABLE, DIO4_EF_ENABLE, DIO5_EF_ENABLE, DIO6_EF_ENABLE, DIO7_EF_ENABLE, DIO8_EF_ENABLE, DIO9_EF_ENABLE, DIO10_EF_ENABLE, DIO11_EF_ENABLE, DIO12_EF_ENABLE, DIO13_EF_ENABLE, DIO14_EF_ENABLE, DIO15_EF_ENABLE, DIO16_EF_ENABLE, DIO17_EF_ENABLE, DIO18_EF_ENABLE, DIO19_EF_ENABLE, DIO20_EF_ENABLE, DIO21_EF_ENABLE, DIO22_EF_ENABLE</td> <td>44000, 44002, 44004, 44006, 44008, 44010, 44012, 44014, 44016, 44018, 44020, 44022, 44024, 44026, 44028, 44030, 44032, 44034, 44036, 44038, 44040, 44042, 44044</td> </tr> </tbody> </table> <p>Show All</p>				Expanded Names	Addresses	DIO0_EF_ENABLE, DIO1_EF_ENABLE, DIO2_EF_ENABLE, DIO3_EF_ENABLE, DIO4_EF_ENABLE, DIO5_EF_ENABLE, DIO6_EF_ENABLE, DIO7_EF_ENABLE, DIO8_EF_ENABLE, DIO9_EF_ENABLE, DIO10_EF_ENABLE, DIO11_EF_ENABLE, DIO12_EF_ENABLE, DIO13_EF_ENABLE, DIO14_EF_ENABLE, DIO15_EF_ENABLE, DIO16_EF_ENABLE, DIO17_EF_ENABLE, DIO18_EF_ENABLE, DIO19_EF_ENABLE, DIO20_EF_ENABLE, DIO21_EF_ENABLE, DIO22_EF_ENABLE	44000, 44002, 44004, 44006, 44008, 44010, 44012, 44014, 44016, 44018, 44020, 44022, 44024, 44026, 44028, 44030, 44032, 44034, 44036, 44038, 44040, 44042, 44044
Expanded Names	Addresses						
DIO0_EF_ENABLE, DIO1_EF_ENABLE, DIO2_EF_ENABLE, DIO3_EF_ENABLE, DIO4_EF_ENABLE, DIO5_EF_ENABLE, DIO6_EF_ENABLE, DIO7_EF_ENABLE, DIO8_EF_ENABLE, DIO9_EF_ENABLE, DIO10_EF_ENABLE, DIO11_EF_ENABLE, DIO12_EF_ENABLE, DIO13_EF_ENABLE, DIO14_EF_ENABLE, DIO15_EF_ENABLE, DIO16_EF_ENABLE, DIO17_EF_ENABLE, DIO18_EF_ENABLE, DIO19_EF_ENABLE, DIO20_EF_ENABLE, DIO21_EF_ENABLE, DIO22_EF_ENABLE	44000, 44002, 44004, 44006, 44008, 44010, 44012, 44014, 44016, 44018, 44020, 44022, 44024, 44026, 44028, 44030, 44032, 44034, 44036, 44038, 44040, 44042, 44044						

DIO-EF Index (Feature Selection)

This register is used to select the extended feature that will get enabled on a given DIO line. The valid DIO lines differ by device. For more specific details look at reference tables 13.2-1 and 13.2-2 as well as the appropriate DIO-EF feature subsection.

Name	Start Address	Type	Access				
DIO#(0:22)_EF_INDEX	44100	UINT32	R/W				
<p>DIO#(0:22)_EF_INDEX - Starting Address: 44100</p> <p>An index to specify the feature you want.</p> <ul style="list-style-type: none"> • Data type: UINT32 (type index = 1) • Readable and writable • T7: <ul style="list-style-type: none"> ◦ Minimum firmware version: 0.8800 <p>Expanded Names</p> <table border="1"> <thead> <tr> <th>Expanded Names</th> <th>Addresses</th> </tr> </thead> <tbody> <tr> <td>DIO0_EF_INDEX, DIO1_EF_INDEX, DIO2_EF_INDEX, DIO3_EF_INDEX, DIO4_EF_INDEX, DIO5_EF_INDEX, DIO6_EF_INDEX, DIO7_EF_INDEX, DIO8_EF_INDEX, DIO9_EF_INDEX, DIO10_EF_INDEX, DIO11_EF_INDEX, DIO12_EF_INDEX, DIO13_EF_INDEX, DIO14_EF_INDEX, DIO15_EF_INDEX, DIO16_EF_INDEX, DIO17_EF_INDEX, DIO18_EF_INDEX, DIO19_EF_INDEX, DIO20_EF_INDEX, DIO21_EF_INDEX, DIO22_EF_INDEX</td> <td>44100, 44102, 44104, 44106, 44108, 44110, 44112, 44114, 44116, 44118, 44120, 44122, 44124, 44126, 44128, 44130, 44132, 44134, 44136, 44138, 44140, 44142, 44144</td> </tr> </tbody> </table> <p>Show All</p>				Expanded Names	Addresses	DIO0_EF_INDEX, DIO1_EF_INDEX, DIO2_EF_INDEX, DIO3_EF_INDEX, DIO4_EF_INDEX, DIO5_EF_INDEX, DIO6_EF_INDEX, DIO7_EF_INDEX, DIO8_EF_INDEX, DIO9_EF_INDEX, DIO10_EF_INDEX, DIO11_EF_INDEX, DIO12_EF_INDEX, DIO13_EF_INDEX, DIO14_EF_INDEX, DIO15_EF_INDEX, DIO16_EF_INDEX, DIO17_EF_INDEX, DIO18_EF_INDEX, DIO19_EF_INDEX, DIO20_EF_INDEX, DIO21_EF_INDEX, DIO22_EF_INDEX	44100, 44102, 44104, 44106, 44108, 44110, 44112, 44114, 44116, 44118, 44120, 44122, 44124, 44126, 44128, 44130, 44132, 44134, 44136, 44138, 44140, 44142, 44144
Expanded Names	Addresses						
DIO0_EF_INDEX, DIO1_EF_INDEX, DIO2_EF_INDEX, DIO3_EF_INDEX, DIO4_EF_INDEX, DIO5_EF_INDEX, DIO6_EF_INDEX, DIO7_EF_INDEX, DIO8_EF_INDEX, DIO9_EF_INDEX, DIO10_EF_INDEX, DIO11_EF_INDEX, DIO12_EF_INDEX, DIO13_EF_INDEX, DIO14_EF_INDEX, DIO15_EF_INDEX, DIO16_EF_INDEX, DIO17_EF_INDEX, DIO18_EF_INDEX, DIO19_EF_INDEX, DIO20_EF_INDEX, DIO21_EF_INDEX, DIO22_EF_INDEX	44100, 44102, 44104, 44106, 44108, 44110, 44112, 44114, 44116, 44118, 44120, 44122, 44124, 44126, 44128, 44130, 44132, 44134, 44136, 44138, 44140, 44142, 44144						

DIO-EF Options and Clock Source Selection

This register isn't used by all DIO-EF features.

If a DIO-EF feature requires the configuration or selection of a clock source (such as PWM Out does), the configuration of this register is required, since it is required for selecting a clock source. See [13.2.1 EF Clock Source](#) for more details about clock source selection.

Name	Start Address	Type	Access				
DIO#(0:22)_EF_OPTIONS	44200	UINT32	R/W				
<p>DIO#(0:22)_EF_OPTIONS - Starting Address: 44200</p> <p>Function dependent on selected feature index.</p> <ul style="list-style-type: none"> • Data type: UINT32 (type index = 1) • Readable and writable • T7: <ul style="list-style-type: none"> ◦ Minimum firmware version: 0.8800 <p>Expanded Names</p> <table border="1"> <thead> <tr> <th>Expanded Names</th> <th>Addresses</th> </tr> </thead> <tbody> <tr> <td>DIO0_EF_OPTIONS, DIO1_EF_OPTIONS, DIO2_EF_OPTIONS, DIO3_EF_OPTIONS, DIO4_EF_OPTIONS, DIO5_EF_OPTIONS, DIO6_EF_OPTIONS, DIO7_EF_OPTIONS, DIO8_EF_OPTIONS, DIO9_EF_OPTIONS, DIO10_EF_OPTIONS, DIO11_EF_OPTIONS, DIO12_EF_OPTIONS, DIO13_EF_OPTIONS, DIO14_EF_OPTIONS, DIO15_EF_OPTIONS, DIO16_EF_OPTIONS, DIO17_EF_OPTIONS, DIO18_EF_OPTIONS, DIO19_EF_OPTIONS, DIO20_EF_OPTIONS, DIO21_EF_OPTIONS,</td> <td>44200, 44202, 44204, 44206, 44208, 44210, 44212, 44214, 44216, 44218, 44220, 44222, 44224, 44226, 44228, 44230, 44232, 44234, 44236, 44238, 44240, 44242, 44244</td> </tr> </tbody> </table> <p>Show All</p>				Expanded Names	Addresses	DIO0_EF_OPTIONS, DIO1_EF_OPTIONS, DIO2_EF_OPTIONS, DIO3_EF_OPTIONS, DIO4_EF_OPTIONS, DIO5_EF_OPTIONS, DIO6_EF_OPTIONS, DIO7_EF_OPTIONS, DIO8_EF_OPTIONS, DIO9_EF_OPTIONS, DIO10_EF_OPTIONS, DIO11_EF_OPTIONS, DIO12_EF_OPTIONS, DIO13_EF_OPTIONS, DIO14_EF_OPTIONS, DIO15_EF_OPTIONS, DIO16_EF_OPTIONS, DIO17_EF_OPTIONS, DIO18_EF_OPTIONS, DIO19_EF_OPTIONS, DIO20_EF_OPTIONS, DIO21_EF_OPTIONS,	44200, 44202, 44204, 44206, 44208, 44210, 44212, 44214, 44216, 44218, 44220, 44222, 44224, 44226, 44228, 44230, 44232, 44234, 44236, 44238, 44240, 44242, 44244
Expanded Names	Addresses						
DIO0_EF_OPTIONS, DIO1_EF_OPTIONS, DIO2_EF_OPTIONS, DIO3_EF_OPTIONS, DIO4_EF_OPTIONS, DIO5_EF_OPTIONS, DIO6_EF_OPTIONS, DIO7_EF_OPTIONS, DIO8_EF_OPTIONS, DIO9_EF_OPTIONS, DIO10_EF_OPTIONS, DIO11_EF_OPTIONS, DIO12_EF_OPTIONS, DIO13_EF_OPTIONS, DIO14_EF_OPTIONS, DIO15_EF_OPTIONS, DIO16_EF_OPTIONS, DIO17_EF_OPTIONS, DIO18_EF_OPTIONS, DIO19_EF_OPTIONS, DIO20_EF_OPTIONS, DIO21_EF_OPTIONS,	44200, 44202, 44204, 44206, 44208, 44210, 44212, 44214, 44216, 44218, 44220, 44222, 44224, 44226, 44228, 44230, 44232, 44234, 44236, 44238, 44240, 44242, 44244						

DIO-EF Configuration

Configuration registers serve two purposes. They provide a location for settings that need to be configured upon DIO-EF enable and they provide a location for settings that users may need to use to update a DIO-EF feature once it has been enabled.

Initial Configuration: Configuration is the initial setup of the Extended Feature. Configuration requires that any DIO-EF running at the pin in question first be disabled. Options can then be loaded. Then the DIO-EF can be enabled.

Update: Some DIO#_CONFIG registers can be updated while a DIO-EF is running. Updating allows the DIO-EF to change its operation parameters without restarting. Note that the clock source and feature index cannot 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 feature for more information.

Name	Start Address	Type	Access
DIO#(0:22)_EF_CONFIG_A	44300	UINT32	R/W
DIO#(0:22)_EF_CONFIG_B	44400	UINT32	R/W
DIO#(0:22)_EF_CONFIG_C	44500	UINT32	R/W
DIO#(0:22)_EF_CONFIG_D	44600	UINT32	R/W

DIO#(0:22)_EF_CONFIG_A - Starting Address: 44300	
Function dependent on selected feature index.	
<ul style="list-style-type: none"> Data type: UINT32 (type index = 1) Readable and writable T7: <ul style="list-style-type: none"> Minimum firmware version: 0.8800 	
Expanded Names	Addresses
DIO0_EF_CONFIG_A, DIO1_EF_CONFIG_A, DIO2_EF_CONFIG_A, DIO3_EF_CONFIG_A, DIO4_EF_CONFIG_A, DIO5_EF_CONFIG_A, DIO6_EF_CONFIG_A, DIO7_EF_CONFIG_A, DIO8_EF_CONFIG_A, DIO9_EF_CONFIG_A, DIO10_EF_CONFIG_A, DIO11_EF_CONFIG_A, DIO12_EF_CONFIG_A, DIO13_EF_CONFIG_A, DIO14_EF_CONFIG_A, DIO15_EF_CONFIG_A, DIO16_EF_CONFIG_A, DIO17_EF_CONFIG_A, DIO18_EF_CONFIG_A, DIO19_EF_CONFIG_A, DIO20_EF_CONFIG_A, DIO21_EF_CONFIG_A, DIO22_EF_CONFIG_A	44300, 44302, 44304, 44306, 44308, 44310, 44312, 44314, 44316, 44318, 44320, 44322, 44324, 44326, 44328, 44330, 44332, 44334, 44336, 44338, 44340, 44342, 44344
Show All	

DIO#(0:22)_EF_CONFIG_B - Starting Address: 44400	
Function dependent on selected feature index.	
<ul style="list-style-type: none"> Data type: UINT32 (type index = 1) Readable and writable T7: <ul style="list-style-type: none"> Minimum firmware version: 0.8800 	
Expanded Names	Addresses
DIO0_EF_CONFIG_B, DIO1_EF_CONFIG_B, DIO2_EF_CONFIG_B, DIO3_EF_CONFIG_B, DIO4_EF_CONFIG_B, DIO5_EF_CONFIG_B, DIO6_EF_CONFIG_B, DIO7_EF_CONFIG_B, DIO8_EF_CONFIG_B, DIO9_EF_CONFIG_B, DIO10_EF_CONFIG_B, DIO11_EF_CONFIG_B, DIO12_EF_CONFIG_B, DIO13_EF_CONFIG_B, DIO14_EF_CONFIG_B, DIO15_EF_CONFIG_B, DIO16_EF_CONFIG_B, DIO17_EF_CONFIG_B, DIO18_EF_CONFIG_B, DIO19_EF_CONFIG_B, DIO20_EF_CONFIG_B, DIO21_EF_CONFIG_B, DIO22_EF_CONFIG_B	44400, 44402, 44404, 44406, 44408, 44410, 44412, 44414, 44416, 44418, 44420, 44422, 44424, 44426, 44428, 44430, 44432, 44434, 44436, 44438, 44440, 44442, 44444
Show All	

DIO#(0:22)_EF_CONFIG_C - Starting Address: 44500	
Function dependent on selected feature index.	
<ul style="list-style-type: none"> Data type: UINT32 (type index = 1) Readable and writable T7: <ul style="list-style-type: none"> Minimum firmware version: 0.8800 	
Expanded Names	Addresses
DIO0_EF_CONFIG_C, DIO1_EF_CONFIG_C, DIO2_EF_CONFIG_C, DIO3_EF_CONFIG_C, DIO4_EF_CONFIG_C, DIO5_EF_CONFIG_C, DIO6_EF_CONFIG_C, DIO7_EF_CONFIG_C, DIO8_EF_CONFIG_C, DIO9_EF_CONFIG_C, DIO10_EF_CONFIG_C, DIO11_EF_CONFIG_C, DIO12_EF_CONFIG_C, DIO13_EF_CONFIG_C, DIO14_EF_CONFIG_C, DIO15_EF_CONFIG_C, DIO16_EF_CONFIG_C, DIO17_EF_CONFIG_C, DIO18_EF_CONFIG_C, DIO19_EF_CONFIG_C, DIO20_EF_CONFIG_C, DIO21_EF_CONFIG_C, DIO22_EF_CONFIG_C	44500, 44502, 44504, 44506, 44508, 44510, 44512, 44514, 44516, 44518, 44520, 44522, 44524, 44526, 44528, 44530, 44532, 44534, 44536, 44538, 44540, 44542, 44544
Show All	

DIO#(0:22)_EF_CONFIG_D - Starting Address: 44600	
Function dependent on selected feature index.	
<ul style="list-style-type: none"> Data type: UINT32 (type index = 1) Readable and writable T7: <ul style="list-style-type: none"> Minimum firmware version: 0.8800 	
Expanded Names	Addresses
DIO0_EF_CONFIG_D, DIO1_EF_CONFIG_D, DIO2_EF_CONFIG_D, DIO3_EF_CONFIG_D, DIO4_EF_CONFIG_D, DIO5_EF_CONFIG_D, DIO6_EF_CONFIG_D, DIO7_EF_CONFIG_D, DIO8_EF_CONFIG_D, DIO9_EF_CONFIG_D, DIO10_EF_CONFIG_D, DIO11_EF_CONFIG_D, DIO12_EF_CONFIG_D, DIO13_EF_CONFIG_D, DIO14_EF_CONFIG_D, DIO15_EF_CONFIG_D, DIO16_EF_CONFIG_D, DIO17_EF_CONFIG_D, DIO18_EF_CONFIG_D, DIO19_EF_CONFIG_D, DIO20_EF_CONFIG_D, DIO21_EF_CONFIG_D, DIO22_EF_CONFIG_D	44600, 44602, 44604, 44606, 44608, 44610, 44612, 44614, 44616, 44618, 44620, 44622, 44624, 44626, 44628, 44630, 44632, 44634, 44636, 44638, 44640, 44642, 44644
Show All	

DIO-EF Basic Read Registers

Some DIO-EF produce results or provide status information that can be read. This information is usually a binary integer. When possible, the T-series device 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”.

Name	Start Address	Type	Access
------	---------------	------	--------

Name	Start Address	Type	Access
DIO#(0:21)_EF_READ_A		UINT32	R
DIO#(0:21)_EF_READ_B	3200	UINT32	R
DIO#(0:21)_EF_READ_A_F	3500	FLOAT32	R
DIO#(0:21)_EF_READ_B_F	3700	FLOAT32	R

DIO#(0:21)_EF_READ_A - Starting Address: 3000

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

- Data type: UINT32 (type index = 1)
- Read-only
- This register may be streamed

Expanded Names	Addresses
DIO0_EF_READ_A, DIO1_EF_READ_A, DIO2_EF_READ_A, DIO3_EF_READ_A, DIO4_EF_READ_A, DIO5_EF_READ_A, DIO6_EF_READ_A, DIO7_EF_READ_A, DIO8_EF_READ_A, DIO9_EF_READ_A, DIO10_EF_READ_A, DIO11_EF_READ_A, DIO12_EF_READ_A, DIO13_EF_READ_A, DIO14_EF_READ_A, DIO15_EF_READ_A, DIO16_EF_READ_A, DIO17_EF_READ_A, DIO18_EF_READ_A, DIO19_EF_READ_A, DIO20_EF_READ_A, DIO21_EF_READ_A Show All	3000, 3002, 3004, 3006, 3008, 3010, 3012, 3014, 3016, 3018, 3020, 3022, 3024, 3026, 3028, 3030, 3032, 3034, 3036, 3038, 3040, 3042 Show All

DIO#(0:21)_EF_READ_B - Starting Address: 3200

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

- Data type: UINT32 (type index = 1)
- Read-only
- This register may be streamed

Expanded Names	Addresses
DIO0_EF_READ_B, DIO1_EF_READ_B, DIO2_EF_READ_B, DIO3_EF_READ_B, DIO4_EF_READ_B, DIO5_EF_READ_B, DIO6_EF_READ_B, DIO7_EF_READ_B, DIO8_EF_READ_B, DIO9_EF_READ_B, DIO10_EF_READ_B, DIO11_EF_READ_B, DIO12_EF_READ_B, DIO13_EF_READ_B, DIO14_EF_READ_B, DIO15_EF_READ_B, DIO16_EF_READ_B, DIO17_EF_READ_B, DIO18_EF_READ_B, DIO19_EF_READ_B, DIO20_EF_READ_B, DIO21_EF_READ_B Show All	3200, 3202, 3204, 3206, 3208, 3210, 3212, 3214, 3216, 3218, 3220, 3222, 3224, 3226, 3228, 3230, 3232, 3234, 3236, 3238, 3240, 3242 Show All

DIO#(0:21)_EF_READ_A_F - Starting Address: 3500

Reads a floating point value. The meaning of value is dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Read-only

Expanded Names	Addresses
DIO0_EF_READ_A_F, DIO1_EF_READ_A_F, DIO2_EF_READ_A_F, DIO3_EF_READ_A_F, DIO4_EF_READ_A_F, DIO5_EF_READ_A_F, DIO6_EF_READ_A_F, DIO7_EF_READ_A_F, DIO8_EF_READ_A_F, DIO9_EF_READ_A_F, DIO10_EF_READ_A_F, DIO11_EF_READ_A_F, DIO12_EF_READ_A_F, DIO13_EF_READ_A_F, DIO14_EF_READ_A_F, DIO15_EF_READ_A_F, DIO16_EF_READ_A_F, DIO17_EF_READ_A_F, DIO18_EF_READ_A_F, DIO19_EF_READ_A_F, DIO20_EF_READ_A_F, DIO21_EF_READ_A_F Show All	3500, 3502, 3504, 3506, 3508, 3510, 3512, 3514, 3516, 3518, 3520, 3522, 3524, 3526, 3528, 3530, 3532, 3534, 3536, 3538, 3540, 3542 Show All

DIO#(0:21)_EF_READ_B_F - Starting Address: 3700

Reads a floating point value. The meaning of value is dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Read-only

Expanded Names	Addresses
DIO0_EF_READ_B_F, DIO1_EF_READ_B_F, DIO2_EF_READ_B_F, DIO3_EF_READ_B_F, DIO4_EF_READ_B_F, DIO5_EF_READ_B_F, DIO6_EF_READ_B_F, DIO7_EF_READ_B_F, DIO8_EF_READ_B_F, DIO9_EF_READ_B_F, DIO10_EF_READ_B_F, DIO11_EF_READ_B_F, DIO12_EF_READ_B_F, DIO13_EF_READ_B_F, DIO14_EF_READ_B_F, DIO15_EF_READ_B_F, DIO16_EF_READ_B_F, DIO17_EF_READ_B_F, DIO18_EF_READ_B_F, DIO19_EF_READ_B_F, DIO20_EF_READ_B_F, DIO21_EF_READ_B_F Show All	3700, 3702, 3704, 3706, 3708, 3710, 3712, 3714, 3716, 3718, 3720, 3722, 3724, 3726, 3728, 3730, 3732, 3734, 3736, 3738, 3740, 3742 Show All

DIO-EF Read-and-Reset Registers

Some DIO-EF can be reset while they are running. Resetting can have different results depending on the feature. For instance, counters are reset to zero.

Name	Start Address	Type	Access
DIO#(0:21)_EF_READ_A_AND_RESET	3100	UINT32	R
DIO#(0:21)_EF_READ_A_F_AND_RESET	3600	FLOAT32	R

DIO#(0:21)_EF_READ_A_AND_RESET - Starting Address: 3100

Reads the same value as DIO#(0:22)_EF_READ_A and forces a reset.

- Data type: UINT32 (type index = 1)
- Read-only
- This register may be streamed

Expanded Names	Addresses
DIO0_EF_READ_A_AND_RESET, DIO1_EF_READ_A_AND_RESET, DIO2_EF_READ_A_AND_RESET, DIO3_EF_READ_A_AND_RESET, DIO4_EF_READ_A_AND_RESET, DIO5_EF_READ_A_AND_RESET, DIO6_EF_READ_A_AND_RESET, DIO7_EF_READ_A_AND_RESET, DIO8_EF_READ_A_AND_RESET, DIO9_EF_READ_A_AND_RESET, DIO10_EF_READ_A_AND_RESET, DIO11_EF_READ_A_AND_RESET, DIO12_EF_READ_A_AND_RESET, DIO13_EF_READ_A_AND_RESET, DIO14_EF_READ_A_AND_RESET, DIO15_EF_READ_A_AND_RESET, DIO16_EF_READ_A_AND_RESET, DIO17_EF_READ_A_AND_RESET, DIO18_EF_READ_A_AND_RESET, DIO19_EF_READ_A_AND_RESET, DIO20_EF_READ_A_AND_RESET, DIO21_EF_READ_A_AND_RESET Show All	3100, 3102, 3104, 3106, 3108, 3110, 3112, 3114, 3116, 3118, 3120, 3122, 3124, 3126, 3128, 3130, 3132, 3134, 3136, 3138, 3140, 3142 Show All

DIO#(0:21)_EF_READ_A_F_AND_RESET - Starting Address: 3600

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Read-only

Expanded Names	Addresses
DIO0_EF_READ_A_F_AND_RESET, DIO1_EF_READ_A_F_AND_RESET, DIO2_EF_READ_A_F_AND_RESET, DIO3_EF_READ_A_F_AND_RESET, DIO4_EF_READ_A_F_AND_RESET, DIO5_EF_READ_A_F_AND_RESET, DIO6_EF_READ_A_F_AND_RESET, DIO7_EF_READ_A_F_AND_RESET, DIO8_EF_READ_A_F_AND_RESET, DIO9_EF_READ_A_F_AND_RESET, DIO10_EF_READ_A_F_AND_RESET, DIO11_EF_READ_A_F_AND_RESET, DIO12_EF_READ_A_F_AND_RESET, DIO13_EF_READ_A_F_AND_RESET, DIO14_EF_READ_A_F_AND_RESET, DIO15_EF_READ_A_F_AND_RESET, DIO16_EF_READ_A_F_AND_RESET, DIO17_EF_READ_A_F_AND_RESET, DIO18_EF_READ_A_F_AND_RESET, DIO19_EF_READ_A_F_AND_RESET, DIO20_EF_READ_A_F_AND_RESET, DIO21_EF_READ_A_F_AND_RESET Show All	3600, 3602, 3604, 3606, 3608, 3610, 3612, 3614, 3616, 3618, 3620, 3622, 3624, 3626, 3628, 3630, 3632, 3634, 3636, 3638, 3640, 3642 Show All

Streaming DIO-EF Results

Though all operations discussed in this section are supported in [command-response](#) mode, some DIO-EF features can be read fast enough to be [streamed](#):

- Frequency In
- Pulse Width In
- High-Speed Counter
- Interrupt Counter
- Interrupt Counter with Debounce
- Quadrature In
- Interrupt Frequency In

In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in [3.2 Stream](#), those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Other Considerations

Specifications

See [Appendix A-2](#) for specs including:

- Frequency Output
- Counter Input Frequency
- Minimum High & Low Time
- "Interrupt" Total Edge Rate

System Timer

Complications can occur if streaming while enabling a DIO-EF that requires the use of a system timer. Please contact LabJack support if you need to do this.

Available DIO-EF By Device/Reference Tables

T4

Table 13.2-1.T4 Digital I/O Extended Features

T4 Digital I/O Extended Features		AIN (0-3)			FIO (4-7)				EIO (0-7)							CIO (0-3)				
Feature	Index#	DIO																		
		4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19			
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			✓	✓	✓	✓	✓	✓											
Conditional Reset	12			✓	✓	✓	✓	✓	✓											

T7

Table 13.2-2.T7 Digital I/O Extended Features

T7 Digital I/O Extended Features		FIO (0-7)				EIO (0-7)								CIO (0-3)				MIO (0-2)						
		DIO																						
Feature	Index#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
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	✓	✓	✓	✓			✓	✓															
Conditional Reset	12	✓	✓	✓	✓			✓	✓															

* Line-to-Line In and Quadrature In both require two DIO lines.

13.2.1 EF Clock Source [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

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

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

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1.

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 clock source 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 clock source 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:

Digital EF Clock Source			
Name	Start Address	Type	Access
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_ENABLE - Address: 44900

1 = enabled. 0 = disabled. Must be disabled during configuration.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8700

DIO_EF_CLOCK0_DIVISOR - Address: 44901

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.9320

DIO_EF_CLOCK0_OPTIONS - Address: 44902

Bitmask: bit0: 1 = use external clock. All other bits reserved.

- Data type: UINT32 (type index = 1)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8700

DIO_EF_CLOCK0_ROLL_VALUE - Address: 44904

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. 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.

- Data type: UINT32 (type index = 1)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 0.8700

A clock source can be enabled after DIO#_EF_INDEX has been configured. This allows several DIO-EFs to be started at the same time.

DIO_EF_CLOCK0_ROLL_VALUE 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.

Read

To read the clock, read DIO_EF_CLOCK0_COUNT:

Digital EF Clock Source

Name	Start Address	Type	Access
DIO_EF_CLOCK0_COUNT	44908	UINT32	R

DIO_EF_CLOCK0_COUNT - Address: 44908

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

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum [firmware](#) version: 0.9430

This can be useful for generating timestamps.

Update

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 and CLOCK2 are actually the LSW & MSW of CLOCK0. The frequency of CLOCK1 is the same as CLOCK0. If DIO_EF_CLOCK0_ROLL_VALUE is $\geq 2^{16}$, then the frequency of CLOCK2 is CLOCK0_freq divided by the modulus (remainder portion) of $\text{CLOCK0_freq} / 2^{16}$. If $(\text{CLOCK0_ROLL_VALUE} - 1)$ is $< 2^{16}$, then the frequency of CLOCK2 is 0. CLOCK1_ROLL_VALUE is the modulus of $(\text{CLOCK0_ROLL_VALUE} - 1) / 2^{16}$ and CLOCK2_ROLL_VALUE is the quotient (integer portion) of $(\text{CLOCK0_ROLL_VALUE} - 1) / 2^{16}$.

13.2.2 PWM Out [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO6, DIO7** (aka FIO6, FIO7)

T7 Capable DIO: **DIO0, DIO2, DIO3, DIO4, DIO5** (aka FIO0, FIO2, FIO3, FIO4, FIO5)

Requires Clock Source: **Yes**

Index: **0**

Streamable: **No**

This PWM Out Extended Feature generates 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 count is zero the DIO line will be set high. When the count matches Config A the line will be set low. Therefore Config 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
```

For the common case of CoreFrequency = 80 MHz we can rewrite output frequency as:

```
PWMFrequency = 80M / (DIO_EF_CLOCK#_DIVISOR * DIO_EF_CLOCK#_ROLL_VALUE)
```

... and for 50% duty-cycle (square wave) simply set:

```
DIO#_EF_CONFIG_A = DIO_EF_CLOCK#_ROLL_VALUE / 2
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies.

The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the [DIO-EF Clock Source](#) section.

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 ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits are reserved and should be set to 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 DIO#_EF_CONFIG_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.

Read

No information is returned by PWM Out.

Reset

Reset has no affect on this feature.

Example

To 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 MHz / (1 * 8000) = 10 kHz
```

Now set the clock 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 calculate CONFIG_A:

```
DC = 25% = 100 * 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
```

For a more detailed walkthrough, see [Configuring a PWM Output](#).

T7 PWM Output Config

Configure settings to generate LJM pseudocode

T4 DIO channels: DIO6 and DIO7

T7 DIO channels: DIO0 and DIO2-DIO5

Desired DIO Channel:

Desired Frequency (Hz):

Desired Duty Cycle (%):

Select Clock Source:

Select Clock Divisor:

Pseudocode:

```
// Configure PWM Functionality
handle = LJM_OpenS("ANY", "ANY", "ANY"); // Open the first found device

// Configure Clock Registers:
LJM_eWriteName(handle, "DIO_EF_CLOCK0_ENABLE", 0); // Disable clock source
// Set Clock0's divisor and roll value to configure frequency: 80MHz/1/80000 = 1kHz
LJM_eWriteName(handle, "DIO_EF_CLOCK0_DIVISOR", 1); // Configure Clock0's divisor
LJM_eWriteName(handle, "DIO_EF_CLOCK0_ROLL_VALUE", 80000); // Configure Clock0's roll value
LJM_eWriteName(handle, "DIO_EF_CLOCK0_ENABLE", 1); // Enable the clock source

// Configure EF Channel Registers:
LJM_eWriteName(handle, "DIO0_EF_ENABLE", 0); // Disable the EF system for initial configuration
LJM_eWriteName(handle, "DIO0_EF_INDEX", 0); // Configure EF system for PWM
LJM_eWriteName(handle, "DIO0_EF_OPTIONS", 0); // Configure what clock source to use: Clock0
LJM_eWriteName(handle, "DIO0_EF_CONFIG_A", 40000); // Configure duty cycle to be: 50%
LJM_eWriteName(handle, "DIO0_EF_ENABLE", 1); // Enable the EF system, PWM wave is now being outputted

// Process to reconfigure PWM frequency of active EF Channel:
// Set Clock0's divisor and roll value to configure frequency: 80MHz/1/80000 = 1kHz
LJM_eWriteName(handle, "DIO_EF_CLOCK0_DIVISOR", 1); // Re-configure Clock0's divisor
LJM_eWriteName(handle, "DIO_EF_CLOCK0_ROLL_VALUE", 80000); // Re-configure Clock0's roll value

// Process to reconfigure PWM Duty Cycle of active EF Channel:
LJM_eWriteName(handle, "DIO0_EF_CONFIG_A", 40000); // Re-configure CONFIG_A to set duty cycle to: 50%
```

Information regarding the functions used in this example are available on [LJM's function reference](#) page. Particularly, the generated pseudocode above ignores errors, but real applications should handle errors.

Support - General Pseudocode Compiler

Support - T7 PWM Output Config Script

13.2.3 PWM Out with Phase [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO6, DIO7** (aka FIO6, FIO7)

T7 Capable DIO: **DIO0, DIO2, DIO3, DIO4, DIO5** (aka FIO0, FIO2, FIO3, FIO4, FIO5)

Requires Clock Source: **Yes**

Index: **1**

Streamable: **No**

This PWM Out with Phase Extended Feature is similar to the [PWM Out](#) DIO-EF, but allows for phase control.

Operation

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
PhaseOffset = 360° * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE
```

When the count matches CONFIG_B, the DIO line will be set high. When the count matches CONFIG_A, the line will be set low. Therefore CONFIG_B minus CONFIG_A controls the duty cycle.

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the [DIO-EF Clock Source](#) section.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 1

DIO#_EF_OPTIONS: Bits 0-2 specify which [clock source](#) to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 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_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 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.

Example

See [13.2.2 PWM Out](#) for an example.

13.2.4 Pulse Out [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO6, DIO7** (aka FIO6, FIO7)

T7 Capable DIO: **DIO0, DIO2, DIO3, DIO4, DIO5** (aka FIO0, FIO2, FIO3, FIO4, FIO5)

Requires Clock Source: **Yes**

Index: **2**

Streamable: **No**

Operation

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](#).

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR // typically 80M/Divisor
PulseOutFrequency = Clock#Frequency / DIO_EF_CLOCK#_ROLL_VALUE
DutyCycle% = 100 * (DIO#_EF_CONFIG_A - DIO#_EF_CONFIG_B) / DIO_EF_CLOCK#_ROLL_VALUE // if A > B
```

For the common case of CoreFrequency = 80 MHz and CONFIG_B fixed at 0, we can rewrite these as:

```
PulseOutFrequency = 80M / (DIO_EF_CLOCK#_DIVISOR * DIO_EF_CLOCK#_ROLL_VALUE)
DutyCycle% = 100 * DIO#_EF_CONFIG_A / DIO_EF_CLOCK#_ROLL_VALUE
```

... and thus for 50% duty cycle simply set:

```
DIO#_EF_CONFIG_A = DIO_EF_CLOCK#_ROLL_VALUE / 2
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the [DIO-EF Clock Source](#) section.

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 ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits are reserved and should be set to 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

Results are read from the following registers.

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

DIO#_EF_READ_B: The target number of pulses.

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 the clock frequency is:

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

and PWM frequency is:

```
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, the duty cycle is:

```
duty cycle = 100 * (2000 - 0) / 10000 = 20%
```

The LabJack will now output 5000 pulses over 5 seconds at 20% duty cycle.

13.2.5 Frequency In [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5** (aka FIO4, FIO5)

T7 Capable DIO: **DIO0, DIO1** (aka FIO0, FIO1)

Requires Clock Source: **Yes**

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

Streamable: **Yes—integer READ registers only.**

Operation

Frequency In will measure the period/frequency of a digital input signal 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.

```
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
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the [DIO-EF Clock Source](#) section.

Roll value for this feature 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 forced to use a lower roll value due to another needed feature such as [PWM Out](#).

A couple of 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

By default, Frequency In operates in one-shot mode where it will measure the frequency once after being enabled and a new measurement only once after each read of a READ_A register. The other option is continuous mode, where the frequency is constantly measured (every edge is processed) and READ registers return the most recent result. Running in continuous mode puts a greater load on the processor.

If you do another read before a new edge has occurred, you will get the same value as before. Some applications will want to use the read-and-reset option so that a value is only returned once and extra reads will return 0. (See Reset below.)

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 3 or 4

DIO#_EF_OPTIONS: Default = 0. Bits 0-2 specify which clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.

DIO#_EF_CONFIG_A: Default = 0. Bit 1: 0 = one-shot, 1 = continuous. All other bits reserved.

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

Results are read from the following registers.

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

DIO#_EF_READ_B: Returns the same value as READ_A.

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

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

Note that all "READ_B" registers are capture registers. All "READ_B" registers are only updated when any "READ_A" register is read. Thus it would be unusual to read any B registers without first reading at least one A register.

Stream Read

All operations discussed in this section are supported in command-response mode. In stream mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the Stream Section those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

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 (2 new edges).

Note that even in continuous mode, with reads happening faster than the signal frequency using a _RESET read will result in measurements of every other cycle not every cycle.

Example

Most applications can use default clock settings, so to configure frequency input on DIO0 you can simply write to 3 registers:

```
DIO_EF_CLOCK0_ENABLE = 1
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 3
DIO0_EF_ENABLE = 1
```

Now you can read the period in seconds from a 4th register DIO0_EF_READ_A_F.

Roll value considerations:

Sometimes, other DIO-EF might interact with this feature. For example, roll value would usually be set to 0 to provide the maximum measurable period, but assume that we have to use 10000 because it is set to that for PWM output on another channel:

```
DIO_EF_CLOCK0_DIVISOR = 8 // Clock0Frequency = 80M/8 = 10 MHz.
DIO_EF_CLOCK0_ROLL_VALUE = 10000 // Roll value needed for PWM output on another channel.
DIO_EF_CLOCK0_ENABLE = 1
```

This clock configuration results in:

```
Resolution = 1 / 10M = 0.1 us
```

and

```
MaxPeriod = 10000 / 10M = 1 ms
```

For a more detailed walkthrough, see Configuring & Reading Frequency.

Rate Limits

T-Series IC Rate Limits

The maximum measurable frequency varies based on the one-shot setting, concurrent stream rate, and other DIO-EFs set to an input mode.

One-shot or Continuous: When one-shot is enabled, the T-series devices will take a single measurement, then wait for a READ_A register to be read before taking another measurement. This means that with one-shot, only a small fraction of the total periods of a signal are measured. One-shot allows for higher maximum measurable frequency than continuous does.

Stream: The stream process is the highest priority process on T-series devices. When stream needs the processor, all other operations are put on hold. That hold occurs more frequently at higher stream speeds. When a DIO-EF process has to wait, the max frequency that can be measured is reduced.

Multiple DIO-EFs: DIO-EFs on other different lines also require processor time. The amount of processor time required depends on the signal being processed and the DIO-EF's settings. The maximum frequencies in the below table give the total max frequency for all running DIO-EFs—divide the max frequencies below by the number of enabled DIO-EFs to get the max frequency for each individual DIO-EF.

Refer to the following table for maximum measurable frequencies in various combinations of stream and one-shot.

Index	One-shot or Continuous	Stream Rate	Max Frequency
3 or 4	Continuous	Stream not running	200 kHz
3 or 4	One-shot	Stream not running	750 kHz
3 or 4	Continuous	10 kHz	75 kHz
3 or 4	One-shot	10 kHz	750 kHz
3 or 4	Continuous	100 kHz	20 kHz
3 or 4	One-shot	100 kHz	250 kHz
5	Continuous	Stream not running	200 kHz
5	One-shot	Stream not running	750 kHz
5	Continuous	10 kHz	75 kHz
5	One-shot	10 kHz	750 kHz
5	Continuous	100 kHz	20 kHz
5	One-shot	100 kHz	250 kHz

13.2.6 Pulse Width In [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5** (aka FIO4, FIO5)

T7 Capable DIO: **DIO0, DIO1** (aka FIO0, FIO1)

Requires Clock Source: **Yes**

Index: **5**

Streamable: **Yes—integer READ registers only.**

Operation

Pulse Width In will measure the high time and low time of a digital input signal, by counting the number of clock source ticks while the signal is high and low. This could also be referred to as duty-cycle input or PWM input. The number of high ticks can be read from DIO#_EF_READ_A and the number of low ticks can be read from DIO#_EF_READ_B.

```

Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor
HighTime(s) = DIO#_EF_READ_A / Clock#Frequency
LowTime(s) = DIO#_EF_READ_B / Clock#Frequency
Resolution(s) = 1 / Clock#Frequency
Max High or Low Time(s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency

```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the [DIO-EF Clock Source](#) section.

Roll value for this feature 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 Out](#).

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

- Divisor = 1, Resolution = 12.5 nanoseconds, MaxTime = 53.7 seconds
- Divisor = 256, Resolution = 3.2 microseconds, MaxTime = 229 minutes

Once this feature is enabled, a new measurement happens on every applicable edge and both result registers are updated on every rising edge. If you do another read before a new rising edge has occurred, you will get the same values 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. (See Reset below.)

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 5

DIO#_EF_OPTIONS: Bits 0-2 specify which [clock source](#) to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits reserved and should be set to 0.

DIO#_EF_CONFIG_A: Bit 1: 1=continuous, 0=OneShot. All other bits reserved.

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

Results are read from the following registers.

DIO#_EF_READ_A: Returns the measured high time in clock source 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 clock source ticks. This is a capture register ... it is only updated when one of the READ_A registers is 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 a capture register ... it is only updated when one of the READ_A registers is read.

Only reading DIO#_EF_READ_A or DIO#_EF_READ_A_F triggers a new measurement.

Stream Read

All operations discussed in this section are supported in [command-response](#) mode. In [stream](#) mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the [Stream Section](#) those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Performs the same read as READ_A, 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 READ_A_F, but then also clears the register so that zero is returned 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 us
```

and

```
MaxPeriod = 10000 / 10M = 1 ms
```

Now configure the DIO_EF on DIO0 as pulse width input.

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

After a full period (rising edge, falling edge, rising edge) has occurred, the values are stored in the result registers. This repeats at each rising edge. 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 are from the same waveform cycle.

Rate Limits

T-Series IC Rate Limits

The maximum measurable frequency varies based on the one-shot setting, concurrent stream rate, and other DIO-EFs set to an input mode.

One-shot or Continuous: When one-shot is enabled, the T-series devices will take a single measurement, then wait for a READ_A register to be read before taking another measurement. This means that with one-shot, only a small fraction of the total periods of a signal are measured. One-shot allows for higher maximum measurable frequency than continuous does.

Stream: The stream process is the highest priority process on T-series devices. When stream needs the processor, all other operations are put on hold. That hold occurs more frequently at higher stream speeds. When a DIO-EF process has to wait, the max frequency that can be measured is reduced.

Multiple DIO-EFs: DIO-EFs on other different lines also require processor time. The amount of processor time required depends on the signal being processed and the DIO-EF's settings. The maximum frequencies in the below table give the total max frequency for all running DIO-EFs—divide the max frequencies below by the number of enabled DIO-EFs to get the max frequency for each individual DIO-EF.

Refer to the following table for maximum measurable frequencies in various combinations of stream and one-shot.

Index	One-shot or Continuous	Stream Rate	Max Frequency
3 or 4	Continuous	Stream not running	200 kHz
3 or 4	One-shot	Stream not running	750 kHz
3 or 4	Continuous	10 kHz	75 kHz
3 or 4	One-shot	10 kHz	750 kHz
3 or 4	Continuous	100 kHz	20 kHz
3 or 4	One-shot	100 kHz	250 kHz
5	Continuous	Stream not running	200 kHz
5	One-shot	Stream not running	750 kHz
5	Continuous	10 kHz	75 kHz
5	One-shot	10 kHz	750 kHz
5	Continuous	100 kHz	20 kHz
5	One-shot	100 kHz	250 kHz

13.2.7 Line-to-Line In [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5** (aka FIO4, FIO5)

T7 Capable DIO: **DIO0, DIO1** (aka FIO0, FIO1)

Requires Clock Source: **Yes**

Index: **6**

Streamable: **No**

Operation

Line-to-Line In measures the time between an edge on one DIO line and an edge on another DIO line by counting the number of clock source ticks between the two edges. The edges can be individually specified as rising or falling.

```
Clock#Frequency = CoreFrequency / DIO_EF_CLOCK#_DIVISOR //typically 80M/Divisor
Time(s) = DIO#_EF_READ_A / Clock#Frequency
Resolution(s) = 1 / Clock#Frequency
Max Time(s) = DIO_EF_CLOCK#_ROLL_VALUE / Clock#Frequency
```

CoreFrequency is always 80 MHz at this time, but in the future some low-power operational modes might result in different core frequencies. The valid values for DIO_EF_CLOCK#_DIVISOR are 1, 2, 4, 8, 16, 32, 64, or 256, and a value of 0 (default) equates to a divisor of 1. For more details about Clock#Frequency and DIO_EF_CLOCK#_DIVISOR, see the [DIO-EF Clock Source](#) section.

Roll value for this feature 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 Out](#).

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

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 (DIO0 and DIO1 only) as Line-to-Line In feature 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 clock source to use ... 000 for Clock0, 001 for Clock1, and 010 for Clock2. All other bits are reserved and should be set to 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

Results are read from the following registers.

DIO#_EF_READ_A: Returns the one-shot measured time in clock source ticks. If the specified combination of edges has not yet been observed this value will be zero. DIO0 & DIO1 both return the same value.

DIO#_EF_READ_A_F: Returns the time 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.

Example

First, configure the clock source:

```
DIO_EF_CLOCK0_DIVISOR = 1 // Clock0Frequency = 80M/1 = 80 MHz
DIO_EF_CLOCK0_ROLL_VALUE = 0
DIO_EF_CLOCK0_ENABLE = 1
```

This clock configuration results in:

```
Resolution = 1 / 80M = 12.5 ns
```

and

```
MaxPeriod = 2^32 / 80M = 53.7 seconds
```

Now configure the DIO-EF on DIO0 and DIO1 as line-to-line:

```
DIO0_EF_ENABLE = 0
DIO1_EF_ENABLE = 0
```

```
DIO0_EF_INDEX = 6 // Index for line-to-line feature.
DIO0_EF_OPTIONS = 0 // Select the clock source.
DIO0_EF_CONFIG_A = 0 // Detect falling edge.
DIO0_EF_ENABLE = 1 // Turn on the DIO-EF
```

```
DIO1_EF_INDEX = 6 // Index for line-to-line feature.
DIO1_EF_OPTIONS = 0 // Select the clock source.
DIO1_EF_CONFIG_A = 0 // Detect falling edge.
DIO1_EF_ENABLE = 1 // Turn on the DIO-EF
```

At this point the device is watching DIO0 for a falling edge. Once that happens it watches for a falling edge on DIO1. Once that happens it stores the time between those 2 edges, which you can read from the READ registers described above.

To do another measurement, repeat the DIO0-EF configuration above, or read from DIO0_EF_READ_A_AND_RESET.

Rate Limits

Line-to-line can achieve high resolution while requiring very little processor time. The time between the edges can be as little as 50 ns. Once a measurement has been completed the system will not measure again until reconfigured or reset. Unless you are reading at a high rate, line-to-line will have little impact on other systems.

13.2.8 High-Speed Counter [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

Capable DIO: **DIO16, DIO17, DIO18, DIO19** (aka CIO0, CIO1, CIO2, CIO3)

Requires Clock Source: **No**

Index: **7**

Streamable: **Yes—integer READ registers only.**

T-series devices support up to 4 high-speed rising-edge 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: 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 with High-Speed Counter.

Read

Results are read from the following register.

DIO#_EF_READ_A: Returns the current count which is incremented on each rising edge.

Stream Read

All operations discussed in this section are supported in [command-response](#) mode. In [stream](#) mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the [Stream Section](#) those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

DIO#_EF_READ_A_AND_RESET: Reads the current count then clears the counter. 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 are not acceptable, then do not use reset but rather just note the "virtual reset" counter value in software and subtract it from other values.

Frequency Measurement

Counters are often used to measure frequency by taking change in count over change in time:

$$\text{Frequency} = (\text{CurrentCount} - \text{PreviousCount}) / (\text{CurrentTimestamp} - \text{PreviousTimestamp})$$

Typically the timestamps are from the host clock (software), but for more accurate timestamps read the CORE_TIMER register (address=61520, UINT32) in the same Modbus packet as the READ registers. CORE_TIMER is a 32-bit system timer running at 1/2 the core speed, and thus is normally 80M/2 => 40 MHz.

Also note that other [digital extended features](#) are available to measure frequency by [timing individual pulses rather than counting over time](#).

Example

Enable CounterC on DIO18/CIO2:

```
DIO18_EF_ENABLE = 0
DIO18_EF_INDEX = 7
DIO18_EF_ENABLE = 1
```

Results can be read from the READ registers defined above.

Edge Rate Limits

See [Appendix A-2](#).

13.2.9 Interrupt Counter [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **8**

Streamable: **Yes—integer READ registers only.**

Interrupt Counter counts the rising edge of pulses on the associated IO line. This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

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.

There are 3 basic techniques for device configuration:

1. Power-up defaults. Most registers related to I/O configuration are part of the [IO Config system](#) where their startup values can be defined by the user. This is easily done with the [Power-up Defaults tab](#) in Kipling.
2. Real time software configuration. Software can write any needed configuration values at the beginning of execution, but you might have to consider how to handle if the device later reboots during software execution. An advantage to this method is that a factory device will work out-of-the-box without requiring the user to first configure the device.
3. Startup script. A [Lua script](#) can be set to run at startup and can write any values to any registers.

Update

No update operations can be performed on Interrupt Counter.

Read

Results are read from the following register.

DIO#_EF_READ_A: Returns the current Count

Stream Read

All operations discussed in this section are supported in [command-response](#) mode. In [stream](#) mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the [Stream Section](#) those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

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.

Frequency Measurement

Counters are often used to measure frequency by taking change in count over change in time:

$$\text{Frequency} = (\text{CurrentCount} - \text{PreviousCount}) / (\text{CurrentTimestamp} - \text{PreviousTimestamp})$$

Typically the timestamps are from the host clock (software), but for more accurate timestamps read the CORE_TIMER register (address=61520, UINT32) in the same Modbus packet as the READ registers. CORE_TIMER is a 32-bit system timer running at 1/2 the core speed, and thus is normally 80M/2 => 40 MHz.

Also note that other [digital extended features](#) are available to measure frequency by [timing individual pulses rather than counting over time](#).

Example

Enable a counter on DIO0:

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 8
DIO0_EF_ENABLE = 1
```

Use the [Register Matrix](#) in Kipling to write those 2 registers above, and also add DIO0_EF_READ_A so you see its value. Now take a wire connected to a GND terminal and tap that wire to the inside-back of the DIO0 screw-terminal (labeled "FIO0"). DIO0_EF_READ_A should increment by 1 or more counts each time you tap the ground wire in DIO0.

As [another test](#) you can set DAC1_FREQUENCY_OUT_ENABLE = 1 to enable the 10 Hz test signal (T7 requires firmware 1.0234+). Jumper DAC1 to DIO0 and you should see DIO0_EF_READ_A increment by about 10 counts per second.

For a more detailed walkthrough, see [Configuring & Reading a Counter](#).

Edge Rate Limits

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 μ s to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 μ s, that edge will be lost. If channel B has another edge during the first 14 μ s, the initial edge will be lost. If channel B has another edge during the second 14 μ s (during the ISR for channel B), the new edge will be lost.

13.2.10 Interrupt Counter with Debounce [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **9**

Streamable: **Yes—integer READ registers only.**

Interrupt Counter with Debounce will increment its count by 1 when it receives a rising edge, a falling edge, or any edge (2x counting). After seeing an applicable edge, any further edges will be ignored during the debounce time.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

Debounce Modes (DIO#_EF_CONFIG_B)

The exact behavior of the counting/debouncing is controlled by an index value written to DIO#_EF_CONFIG_B.

- 0: Count falling, debounce all, self-restarting timeout.
- 1: Count rising, debounce all, self-restarting timeout.
- 2: Count & debounce all, self-restarting timeout.
- 3: Count & debounce falling, fixed timeout.
- 4: Count & debounce rising, fixed timeout.
- 5: Timeout starts on falling edge. During timeout, a rising edge cancels and a falling edge restarts the timeout. After timeout any edge causes a count.
- 6: Timeout starts on rising edge. During timeout, a falling edge cancels and a rising edge restarts the timeout. After timeout any edge causes a count.

Self-restarting timeout means that during timeout any edge will restart the timeout with the value specified with DIO#_EF_CONFIG_A.

Mode 0 is commonly used with a normally-open push-button switch that is actuated by a person. We only want to count the push (falling edge), but expect bounce on the push & release (falling & rising edges) so need to debounce both.

Mode 4 might be used with some sort of device that outputs a fixed length positive pulse. For example, say a device provides a 1000 μ s pulse, and there is always at least 5000 μ s between pulses. Set the debounce timeout to 2000 μ s so that the timeout period safely covers the entire pulse, but the timeout will for sure be done before another pulse can occur.

Modes 5 & 6 implement a requirement that the state of the line must remain low or high for some amount of time. For example, if you use mode 5 with a push-button switch and set DIO#_EF_CONFIG_A = 50000, that means that someone must push the switch and hold it down solidly for at least 50ms, and then the count will occur when they release the switch. An advantage to these modes is that they will ignore brief transient signals.

Configure

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable
DIO#_EF_INDEX: 9
DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Debounce timeout in microseconds (μ s, 0-1000000).
DIO#_EF_CONFIG_B: Debounce mode index.
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

Results are read from the following register.

DIO#_EF_READ_A: Returns the current Count

Stream Read

All operations discussed in this section are supported in [command-response](#) mode. In [stream](#) mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the [Stream Section](#) those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

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.

Example

Enable a debounce counter on DIO0:

```
DIO0_EF_ENABLE = 0
DIO0_EF_INDEX = 9
DIO0_EF_CONFIG_A = 20000 // 20 ms debounce time
DIO0_EF_CONFIG_B = 0 // count falling, debounce all, self-restarting timeout
DIO0_EF_ENABLE = 1
```

Results can be read from the READ registers defined above.

Edge Rate Limits

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 μ s to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 μ s, that edge will be lost. If channel B has another edge during the first 14 μ s, the initial edge will be lost. If channel B has another edge during the second 14 μ s (during the ISR for channel B), the new edge will be lost.

13.2.11 Quadrature In [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **10**

Streamable: **Yes—integer READ registers only.**

Quadrature input uses two DIOs to track a quadrature signal. Quadrature is a directional count often used in rotary encoders when you need to keep track of absolute position with forward & reverse movement. If you have movement in only one direction, or another way to know direction, you can simply use a normal counter with one phase of the encoder output.

T-series devices uses 4x quadrature decoding, meaning that every edge observed (rising & falling on both phases) will increment or decrement the count.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

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, which 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.

Some quadrature encoders also include a third output channel, called a zero (Z-phase) or index or reference signal, which supplies a single pulse per revolution. This single pulse is used for absolute determination of position. T-series devices support resets according to this reference signal. Z-phase will reset the count when a high state is detected on the specified DIO line at the same time any edge occurs on A or B phases. If the reference pulse is wider than A/B pulses, consider using the [Conditional Reset](#) feature instead of this Z-phase support. If the reference pulse is only high in between A/B edges, consider some sort of RC circuit to elongate it or consider using the [Conditional Reset](#) feature. If set to one-shot mode, Z-phase will clear the count only once and must be "re-armed" by disabling/re-enabling the feature or a read from DIO#_EF_READ_A_AND_RESET.

Configure

Two DIO must be configured for Quadrature In. The two lines must be an adjacent even/odd pair:

T4: DIO4/5, DIO6/7, and DIO8/9 are valid pairs.

T7: DIO0/1, DIO2/3, and DIO6/7 are valid pairs.

The even IO line will be phase A and the odd will be phase B.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 10

DIO#_EF_OPTIONS: Not used.
DIO#_EF_CONFIG_A: Z-phase control: 0 = Z-phase off, 1 = Z-phase on, 3 = Z-phase on in one shot mode.
DIO#_EF_CONFIG_B: Z-phase DIO number.
DIO#_EF_CONFIG_C: Not used.
DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed with Quadrature In.

Read

The quadrature count and error count are available from the even channel. The odd channel will return zeros.

Results are read from the following registers.

DIO#_EF_READ_A - Returns the current count as a signed 2's complement value.
DIO#_EF_READ_B - Returns the number of detected errors.
DIO#_EF_READ_A_F - Returns the count in a single precision float (float32).

Only reading DIO#_EF_READ_A or DIO#_EF_READ_A_F triggers a new measurement.

Stream Read

All operations discussed in this section are supported in [command-response](#) mode. In [stream](#) mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the [Stream Section](#) those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

Reset

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

Example

Configure DIO6 and DIO7 as quadrature inputs:

```
DIO6_EF_ENABLE = 0 //Make sure DIO-EF is disabled on DIO6
DIO7_EF_ENABLE = 0 //Make sure DIO-EF is disabled on DIO7

DIO6_EF_INDEX = 10 //Set feature index for DIO6 to quadrature.
DIO7_EF_INDEX = 10 //Set feature index for DIO7 to quadrature.

DIO6_EF_ENABLE = 1 //Enable quadrature DIO-EF on DIO6, for A phase.
DIO7_EF_ENABLE = 1 //Enable quadrature DIO-EF on DIO7, for B phase.
```

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 DIO6_EF_READ_A or DIO6_EF_READ_A_AND_RESET.

To enable z-phase connected to DIO5 you would do:

```
DIO6_EF_CONFIG_A=1
DIO6_EF_CONFIG_B=5
DIO7_EF_CONFIG_A=1
DIO7_EF_CONFIG_B=5
```

Testing

On some LabJack devices you can physically tap a GND wire into the two DIO channels to cause some counts, but that does not work on T-series devices. Testing needs to be done with a proper quadrature signal.

If testing with an actual encoder, start with DIO-EF enabled and simply watch (e.g. in Kipling) the digital inputs as you slowly turn the encoder to see if the inputs change between high and low. That confirms a valid electrical connection.

You can test using 2 digital inputs to create a quadrature sequence. For example, configure quadrature on DIO6/7 as shown above, and connect DIO0 to DIO6 and DIO1 to DIO7.

```
DIO0 = 1 //Initialize DIO0 to output-high.
DIO1 = 1 //Initialize DIO1 to output-high.

DIO6_EF_ENABLE = 0 //Make sure DIO-EF is disabled on DIO6
DIO7_EF_ENABLE = 0 //Make sure DIO-EF is disabled on DIO7
DIO6_EF_INDEX = 10 //Set feature index for DIO6 to quadrature.
DIO7_EF_INDEX = 10 //Set feature index for DIO7 to quadrature.
DIO6_EF_ENABLE = 1 //Enable quadrature DIO-EF on DIO6, for A phase.
DIO7_EF_ENABLE = 1 //Enable quadrature DIO-EF on DIO7, for B phase.
```

Now we can simulate a quadrature signal by toggling DIO0 and DIO1—aka FIO0 and FIO1—in the proper sequence. We will use the FIO_STATE register (address=2500) which operates on all 8 FIO bits at once, but we will set the inhibit bits for FIO2-FIO7 (bits 10-15) so they are not affected. To set bits 10-15 we can simply add 64512 to the desired FIO0/1 state value:

```
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A = 0
Write FIO_STATE = 1 + 64512, then should read DIO6_EF_READ_A = 1
Write FIO_STATE = 0 + 64512, then should read DIO6_EF_READ_A = 0
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A = -1
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A = -2
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A = -1
Write FIO_STATE = 0 + 64512, then should read DIO6_EF_READ_A = 0
Write FIO_STATE = 1 + 64512, then should read DIO6_EF_READ_A = 1
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A = 2
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A = 3
Write FIO_STATE = 0 + 64512, then should read DIO6_EF_READ_A = 4
Write FIO_STATE = 1 + 64512, then should read DIO6_EF_READ_A = 5
Write FIO_STATE = 3 + 64512, then should read DIO6_EF_READ_A = 6
Write FIO_STATE = 2 + 64512, then should read DIO6_EF_READ_A = 7
```

Quadrature Decoding or Simple Counting?

Quadrature decoding is only needed when you need to keep track of absolute position with changes in direction included.

If you want to track absolute position but the direction does not change, or for whatever reason you always know the direction of movement, then you can just count the pulses from one phase (A or B) using a simple counter. The [interrupt counters](#) available on 6 of the FIO0 lines have the same 70k edge rate limit discussed below, but they only do 1x counting and only use 1 timer. The [high-speed counters](#) on the 4 CIO lines can handle up to 5 MHz per counter.

If you are just trying to measure frequency, you can use a counter as described above and note the change in count over some time interval, or you can use a [DIO-EF](#) that measures the time of individual pulses. Either way, just one phase (A or B) is needed.

Edge Rate Limits

Keep in mind that T-series devices 4x quadrature counting. For example, a 100 pulses/revolution encoder will generate 400 edges/revolution.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 μ s to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 μ s, that edge will be lost. If channel B has another edge during the first 14 μ s, the initial edge will be lost. If channel B has another edge during the second 14 μ s (during the ISR for channel B), the new edge will be lost.

For faster quadrature tracking, one option is to use a chip such as the [LFLS7366R-S](#) from US Digital and then use the SPI ability of the T-series device to talk to that chip. In fact, a Lua script can be used to handle the SPI communication with the chip and periodically put the current count in a user-ram register than can be easily read by any host application or Modbus client. If you don't find an example for the LFLS7366 ask us about it.

13.2.12 Interrupt Frequency In [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No. Uses core clock / 2.**

Index: **11**

Streamable: **Yes—integer READ registers only.**

Interrupt Frequency In will measure the frequency of a signal on the associated DIO line.

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. See the discussion of edge rate limits at the bottom of this page.

To measure the frequency, the T-series device 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 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 multiplied by the maximum expected period must be less than 107 s or the result will overflow:

```
107 > (NumToAverage * MaxPeriod)
```

By default, Interrupt Frequency In operates in one-shot mode where it will measure the frequency once after being enabled and a new measurement only once after each read. The other option is continuous mode, where the frequency is constantly measured (every edge is processed) and READ registers return the most recent result. Running in continuous mode 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: Default = 0. Bit 0: Edge select; 0 = falling, 1 = rising. Bit 1: 0 = one-shot, 1 = continuous.

DIO#_EF_CONFIG_B: Default = 0 which equates to 1. Number of periods to be measured and averaged.

DIO#_EF_CONFIG_C: Not used.

DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed with Interrupt Frequency In.

Read

Results are read from the following registers.

DIO#_EF_READ_A: Returns the average period per cycle in ticks (core clock ticks / 2).

DIO#_EF_READ_B: Returns the total core clock tick count.

DIO#_EF_READ_A_F: Returns the average period per cycle in seconds. Takes into account the number of periods to be averaged and the core clock speed.

DIO#_EF_READ_B_F: Returns the average frequency in Hz. Takes into account the number of periods to be averaged and the core clock speed.

Note that all "READ_B" registers are capture registers. All "READ_B" registers are only updated when any "READ_A" register is read. Thus it would be unusual to read any B registers without first reading at least one A register.

Stream Read

All operations discussed in this section are supported in [command-response](#) mode. In [stream](#) mode, you can read from the integer READ registers (A, B, A_AND_RESET), but as mentioned in the [Stream Section](#) those reads only return the lower 16 bits so you need to also use STREAM_DATA_CAPTURE_16 in the scan list to get the upper 16 bits.

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 (2 new edges).

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 (2 new edges).

Note that even in continuous mode, with reads happening faster than the signal frequency using a _RESET read will result in measurements of every other cycle not every cycle.

Example

To configure Interrupt Frequency In on DIO6 you can simply write to 2 registers:

```
DIO6_EF_ENABLE = 0
DIO6_EF_INDEX = 11
DIO6_EF_ENABLE = 1
```

Now you can read the period in seconds from a 4th register DIO6_EF_READ_A_F.

Edge Rate Limits

This interrupt-based digital I/O extended feature (DIO-EF) is not purely implemented in hardware, but rather firmware must service each edge. This makes it substantially slower than other DIO-EF that are purely hardware-based. To avoid missed edges, the aggregate limit for edges seen by all interrupt-based DIO-EF is 70k edges/second. If stream mode is active, the limit is reduced to 20k edges/second. Excessive processor loading (e.g. a busy Lua script) can also reduce these limits.

The more proper way to think of the edge limit, and understand error that could be introduced when using multiple interrupt-based DIO-EF, is to consider that the interrupt that processes an edge can take up to 14 μ s to complete. When a particular channel sees an applicable edge, an IF (interrupt flag) is set for that channel that tells the processor it needs to run an ISR (interrupt service routine) for that channel. Once an ISR is started, it runs to completion and no other ISR can run until it is done (except that stream interrupts are higher priority and will preempt other interrupts). When an ISR completes, it clears the IF for that channel. So it is okay to have edges on multiple channels at the same time, as long as there is not another edge on any of those channels before enough time to process all the initial edges.

Say that channel A & B have an edge occur at the same time and an ISR starts to process the edge on channel A. If channel A has another edge during the first 14 μ s, that edge will be lost. If channel B has another edge during the first 14 μ s, the initial edge will be lost. If channel B has another edge during the second 14 μ s (during the ISR for channel B), the new edge will be lost.

13.2.13 Conditional Reset [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T4 Capable DIO: **DIO4, DIO5, DIO6, DIO7, DIO8, DIO9** (aka FIO4, FIO5, FIO6, FIO7, EIO0, EIO1)

T7 Capable DIO: **DIO0, DIO1, DIO2, DIO3, DIO6, DIO7** (aka FIO0, FIO1, FIO2, FIO3, FIO6, FIO7)

Requires Clock Source: **No**

Index: **12**

Streamable: **No**

DIO-EF Conditional Reset will reset a specified DIO-EF after a specified number of edges have been detected.

Configure

To set up a DIO-EF Conditional Reset is simple. Just set the DIO number of the DIO-EF you would like to reset and then set the other options. More options are likely to be added, so be sure to leave unused bits cleared to zero.

DIO#_EF_ENABLE: 0 = Disable, 1 = Enable

DIO#_EF_INDEX: 12

DIO#_EF_OPTIONS: Not used.

DIO#_EF_CONFIG_A: Reset Options bitmask:

- bit 0: Edge select. 1 = rising, 0 = falling
- bit 1: reserved
- bit 2: OneShot. 1 = only reset once. 0 = reset every n edges.

DIO#_EF_CONFIG_B: Number of edges per reset.

DIO#_EF_CONFIG_C: IO number of DIO-EF to be reset.

DIO#_EF_CONFIG_D: Not used.

Update

No update operations can be performed on Conditional Reset.

Read

Results are read from the following registers.

DIO#_EF_READ_A – Returns the current count.

Example

This example assumes that DIO0 has a running extended feature such as quadrature or a counter. Now we will set up DIO2 as a falling edge trigger that will reset the count of DIO0_EF.

```
DIO2_EF_ENABLE = 0 // Ensure that the DIO-EF is not running so that it can be configured.
DIO2_EF_INDEX = 12 // Set to Conditional Reset
DIO2_EF_CONFIG_A = 0 // Falling edges
DIO2_EF_CONFIG_B = 1 // Reset every edges
DIO2_EF_CONFIG_C = 0 // Reset events clear the count of DIO0_EF
DIO2_EF_ENABLE = 1 // Turn on the DIO-EF
```

Now falling edges on DIO2 will set the count of DIO0_EF to zero.

13.3 I2C [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

I2C Overview

[I²C](#) or [I2C](#) is a two-wire synchronous serial protocol typically used to send data between chips. I2C is considered an advanced topic. A good knowledge of the protocol is recommended. Troubleshooting may require a logic analyzer or oscilloscope.

T-series I2C Support

T-series devices support master-mode Inter-Integrated Circuit ([I²C](#) or [I2C](#)) communication. Two digital IO are required to act as clock (SCL) and data (SDA). Any T-series digital I/O line can be SDA or SCL.

The I²C bus generally requires pull-up resistors of perhaps 4.7 kΩ from SDA to VS and SCL to VS.

EIO and CIO digital I/O lines should be used, instead of the FIO lines, due to the extra lower output impedances on those lines. Look at the [Digital I/O Specifications](#) page for more details.

[Lua scripting](#) is often convenient for serial applications. For example, a script can run the serial communication at a specified interval, and put the result in USER_RAM registers. The host software can read from the USER_RAM registers when convenient. This puts the complications of serial communication in a script running on the T-series device. The [I²C Sensor Lua examples](#) contain several serial communication examples.

This I²C bus is not an alternative to the USB connection. Rather, the host application will write/read data to/from the LabJack, and the LabJack will communicate with some other device using I²C.

Data Rates

I²C is done in [command-response](#) mode ... either from a host application or an onboard Lua script. Throughput can be estimated by looking at the bit rate (I2C_SPEED_THROTTLE) and number of bits to transfer, and for external host communication adding the packet overhead estimates from the beginning of [Appendix A-1](#). A further consideration is how much you can fit in 1 packet or whether multiple packets will be required.

How-To

Running an I2C operation on a T-series device requires:

1. Initial configuration.
2. Transmit load data.
3. Specify the number of bytes to read.
4. Execute the I²C operations.
5. Read the data that was read from the slave device, if any.
6. Debugging (optional): Read the acknowledgement array to determine which bytes were acknowledged by the slave device.

Steps 2-5 can be repeated as long as I²C has not been used to communicate with a different device.

1. Initial Configuration

Several registers need to be written to configure the T-series device.

Digital IO lines need to be selected to act as SDA and SCL:

Name	Start Address	Type	Access
I2C_SDA_DIONUM	5100	UINT16	R/W
I2C_SCL_DIONUM	5101	UINT16	R/W

I2C_SDA_DIONUM - Address: 5100

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

I2C_SCL_DIONUM - Address: 5101

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

- Data type: UUINT16 (type index = 0)
- Readable and writable
- Default value: 0

Set the clock speed:

Name	Start Address	Type	Access
I2C_SPEED_THROTTLE	5102	UUINT16	R/W

I2C_SPEED_THROTTLE - Address: 5102

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, 65516 is ~100 kHz.

- Data type: UUINT16 (type index = 0)
- Readable and writable
- Default value: 0

The options register controls several compatibility settings:

Name	Start Address	Type	Access
I2C_OPTIONS	5103	UUINT16	R/W

I2C_OPTIONS - Address: 5103

Advanced. Controls details of the I2C protocol to improve device compatibility. bit 0: 1 = Reset the I2C bus before attempting communication. bit 1: 0 = Restarts will use a stop and start, 1 = Restarts will not use a stop. bit 2: 1 = disable clock stretching.

- Data type: UUINT16 (type index = 0)
- Readable and writable
- Default value: 0

Load the address of the slave device:

Name	Start Address	Type	Access
I2C_SLAVE_ADDRESS	5104	UUINT16	R/W

I2C_SLAVE_ADDRESS - Address: 5104

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

- Data type: UUINT16 (type index = 0)
- Readable and writable
- Default value: 0

2. Transmit load data.

Load an array of bytes to send to the slave device. This array does not include the device address.

Name	Start Address	Type	Access
I2C_NUM_BYTES_TX	5108	UUINT16	R/W
I2C_DATA_TX	5120	BYTE	W

I2C_NUM_BYTES_TX - Address: 5108

The number of data bytes to transmit. Zero is valid and will result in a read-only I2C operation.

- Data type: UUINT16 (type index = 0)
- Readable and writable
- Default value: 0

I2C_DATA_TX - Address: 5120

Data that will be written to the I2C bus. This register is a buffer.

- Data type: BYTE (type index = 99)
- Write-only
- Default value: 0
- This register is a [Buffer Register](#)

3. Specify the number of bytes to read.

Write the number of bytes to be read from the slave device. Read operations will always be performed after any write operations.

Name	Start Address	Type	Access
I2C_NUM_BYTES_RX	5109	UUINT16	R/W

I2C_NUM_BYTES_RX - Address: 5109

The number of data bytes to read. Zero is valid and will result in a write-only I2C operation.

- Data type: UUINT16 (type index = 0)
- Readable and writable

- Default value: 0

4. Execute the I²C operations.

Instruct the T-series device to run the write and read operations specified in the last two steps.

Name	Start Address	Type	Access
I2C_GO	5110	UINT16	R/W

i2c_go - Address: 5110

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 0.9504

5. Read the data that was read from the slave device, if any.

Data received from the slave device will be saved in buffer on the T-series device. Use the I2C_DATA_RX to read the buffer:

Name	Start Address	Type	Access
I2C_DATA_RX	5160	BYTE	R

i2c_data_rx - Address: 5160

Data that has been read from the I2C bus. This register is a buffer. Underrun behavior - fill with zeros.

- Data type: BYTE (type index = 99)
- Read-only
- Default value: 0
- This register is a [Buffer Register](#)

6. Debugging: Read the acknowledgement array to determine which bytes were acknowledged by the slave device.

The below register can help troubleshoot I²C issues. The ACKs register will record all Acknowledgement signals transmitted from the slave device to the master (master is the T-series device). Data is always transmitted over I²C in the following sequence: Slave Address, Write data, Slave Address, Read data. If the number of bytes to write or read is zero then the address preceding the zero operation will be skipped. Only bytes transmitted to the slave can produce an ACK that will be recorded in the I2C_ACKS register. After each byte sent to the slave device an acknowledgement will be stored in bit 0. A 1 indicates that the bytes was acknowledged, a 0 indicates no acknowledgement. Before saving the an ACK or no-ACK the register is shifted left.

For example, if two bytes are transmitted and three are read there will be four ACKs. Bit 3 represents the acknowledgement to the first slave address which starts the write operation, bit 2 is the first data byte, Bit 1 is the seconds data byte, bit 0 is the second slave address which starts the read operation.

Name	Start Address	Type	Access
I2C_ACKS	5114	UINT32	R/W

i2c_acks - Address: 5114

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 0.9504

Example

This demonstrates I²C communications with an LJTick-DAC connected to FIO0/FIO1.

First, configure the I²C settings:

```
I2C_SDA_DIONUM = 1 // Set SDA pin number = 1(FIO1)
I2C_SCL_DIONUM = 0 // Set SCL pin number = 0 (FIO0)
I2C_SPEED_THROTTLE = 0 // Set speed throttle = 0 (max)
I2C_OPTIONS = 0 // Set options = 0
I2C_SLAVE_ADDRESS = 80 // Set 7-bit slave address of the I2C chip = 80 (0x50)
```

Read from EEPROM bytes 0-3 in the user memory area. We need a single I²C transmission that writes the chip's memory pointer and then reads the data.

```
I2C_NUM_BYTES_TX = 1 // Set the number of bytes to transmit to 1
I2C_NUM_BYTES_RX = 4 // Set the number of bytes to receive to 4
I2C_DATA_TX = {0} // Set the TX data. byte 0: Memory pointer = 0.
I2C_GO = 1 // Do the I2C communications.
I2C_DATA_RX = {?, ?, ?, ?} // Get the RX data (4 bytes).
```

Write EEPROM bytes 0-3 in the user memory area, using the page write technique. Note that page writes are limited to 16 bytes max, and must be aligned with the 16-byte page intervals. For instance, if you start writing at address 14, you can only write two bytes because byte 16 is the start of a new page.

```
I2C_NUM_BYTES_TX = 5 // Set the number of bytes to transmit to 5
I2C_NUM_BYTES_RX = 0 // Set the number of bytes to receive to 0 (not receiving data)
I2C_DATA_TX = {0, 156, 26, 2, 201} // Set the TX data. byte 0: Memory pointer = 0, bytes 1-4: EEPROM bytes 0-3.
I2C_GO = 1 // Do the I2C communications.
```

If using multiple I2C busses, just include writes to set the DIONUMs each time you want to communicate on a different bus. Other configuration is global also, so if different busses need different configuration include those writes also:

```
I2C_SDA_DIONUM = 2 // Set SDA pin number = 2 (FIO2)
I2C_SCL_DIONUM = 3 // Set SCL pin number = 3 (FIO3)
I2C_SPEED_THROTTLE = 65516 // Set speed throttle for ~100 kbps
I2C_NUM_BYTES_TX = 1 // Set the number of bytes to transmit to 1
I2C_NUM_BYTES_RX = 4 // Set the number of bytes to receive to 4
I2C_DATA_TX = {0} // Set the TX data. byte 0: Memory pointer = 0.
I2C_GO = 1 // Do the I2C communications.
I2C_DATA_RX = {?, ?, ?, ?} // Get the RX data (4 bytes).
```

Note: When writing the TX and RX data, LJM functions [eWriteNameArray](#) and [eReadNameArray](#) functions are recommended for ease of use.

13.3.1 I2C Simulation Tool [T-Series Datasheet]

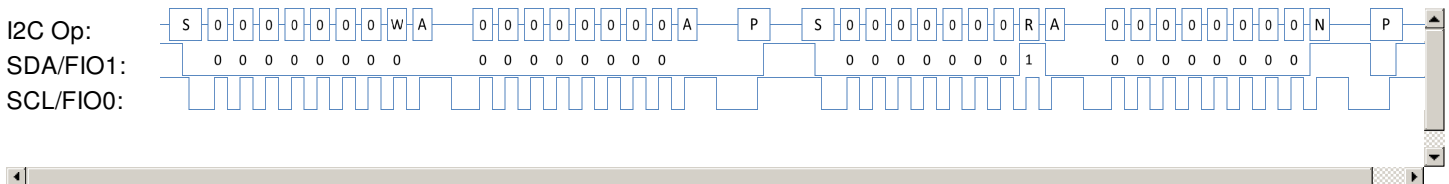
[Log in](#) or [register](#) to post comments

I2C Simulator

This JavaScript application is designed to help give an understanding of the [LabJack T4](#) and [LabJack T7](#)'s I2C functionality. For more information about the T4 and T7's I2C functionality and register descriptions, see [13.3 I2C](#). Each I2C register that affects the output is shown below. The two registers that are omitted are I2C_GO and I2C_ACKS:

- I2C_GO: Executes the configured I2C request.
- I2C_ACKS: Reads the received ACKS & NACKs packed into a binary array.

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



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:

--Input below in hexadecimal--

I2C_SLAVE_ADDRESS:

I2C_NUM_BYTES_TX:

I2C_NUM_BYTES_RX:

I2C_DATA_TX:

Key: I2C_DATA_RX:

- **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

Please feel free to leave comments below.

[Log in](#) or [register](#) to post comments

13.4 SPI [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

SPI Overview

SPI (serial peripheral interface) is a four-wire synchronous serial protocol. SPI is considered an advanced topic. A good knowledge of the protocol is recommended. Troubleshooting may require a logic analyzer or oscilloscope.

T-series SPI Support

T-Series devices support Serial Peripheral Interface (SPI) communication as the master only. Four digital IO lines are used: MISO (data line; Master In Slave Out), MOSI (data line; Master Out Slave In), CLK (clock), and CS (chip select).

[Lua scripting](#) is often convenient for serial applications. For example, a script can run the serial communication at a specified interval, and put the results in USER_RAM registers. The host software can read from the USER_RAM registers when convenient. This puts the complications of serial communication in a script running on the T-series device. The [Lua examples](#) contain several serial communication examples.

SPI is not an alternative to the USB connection. Rather, the host application will write/read data to/from the T-series device, and the T-series device communicates with some other device using the serial protocol.

How-To

Running a SPI operation on a T-series device requires:

1. Initial configuration.
2. Send load data to the slave device.
3. Execute the SPI operation.
4. Read the data that was read from the slave device.

1. Initial configuration.

Several registers need to be written to configure the T-series device.

DIO lines need to be selected to act as MISO, MOSI, CLK, and CS:

Name	Start Address	Type	Access
SPI_CS_DIONUM	5000	UINT16	R/W
SPI_CLK_DIONUM	5001	UINT16	R/W
SPI_MISO_DIONUM	5002	UINT16	R/W
SPI_MOSI_DIONUM	5003	UINT16	R/W

SPI_CS_DIONUM - Address: 5000

The DIO line for Chip-Select.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

SPI_CLK_DIONUM - Address: 5001

The DIO line for Clock.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

SPI_MISO_DIONUM - Address: 5002

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

SPI_MOSI_DIONUM - Address: 5003

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

The SPI mode controls which edge of clock signals value data, and whether the clock will idle high or low:

Name	Start Address	Type	Access
SPI_MODE	5004	UINT16	R/W

SPI_MODE - Address: 5004

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. For CPOL and CPHA explanations, see Wikipedia article: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

Set the clock speed with SPI_SPEED_THROTTLE:

Name	Start Address	Type	Access
SPI_SPEED_THROTTLE	5005	UINT16	R/W

SPI_SPEED_THROTTLE - Address: 5005

This value controls the SPI clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~800 kHz. 65500 = ~100 kHz, 65100 = ~10 kHz, 61100 = ~1 kHz, 21000 = ~100 Hz, and 1 = ~67 Hz. Avoid setting too low such that the entire transaction lasts longer than the 250 millisecond timeout of the internal watchdog timer.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

Starting with a low clock speed and increasing the speed after everything is working is usually a good idea.

The following table lists approximate clock rates measured for various values of SPI_SPEED_THROTTLE with T7 firmware 1.0150:

Throttle value	Clock Speed (kHz)
0	780
65530	380
65500	100
65100	10
61100	1
21000	0.1
1	0.067

Lastly, set SPI_OPTIONS:

Name	Start Address	Type	Access
SPI_OPTIONS	5006	UINT16	R/W

SPI_OPTIONS - Address: 5006

Bit 0 is Auto-CS-Disable. When bit 0 is 0, CS is enabled. When bit 0 is 1, CS is disabled. Bit 1: 0 = Set DIO directions before starting the SPI operations, 1 = Do not set DIO directions. Bit 2: 0 = Transmit data MSB first, 1 = LSB first. Bits 4-7: This value sets the number of bits that will be transmitted during the last byte of the SPI operation. Default is 8, valid options are 1-8.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

Setting SPI_OPTIONS is normally not needed, but can help with compatibility in the following situations:

- If the hardware setup does not require a CS line.
- If the hardware setup requires special digital IO configuration.
- If data needs to be transmitted LSB first.
- If the number of bits to be transferred is not an even multiple of eight.

2. Send load data to the slave device.

Name	Start Address	Type	Access
SPI_NUM_BYTES	5009	UINT16	R/W
SPI_DATA_TX	5010	BYTE	W

SPI_NUM_BYTES - Address: 5009

The number of bytes to transfer.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

SPI_DATA_TX - Address: 5010

Write data here. This register is a buffer.

- Data type: BYTE (type index = 99)
- Write-only
- Default value: 0
- This register is a [Buffer Register](#)

SPI is full duplex. That means that data is sent in both directions at the same time. The number of bytes read from the slave must always equal the number of bytes written to the slave. To read data from a slave device without sending data to it, load dummy data into SPI_DATA_TX.

3. Execute the SPI operation.

Name	Start Address	Type	Access
SPI_GO	5007	UINT16	W

SPI_GO - Address: 5007

Write 1 to begin the configured SPI transaction.

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)

4. Read the data that was read from the slave device.

Name	Start Address	Type	Access
SPI_DATA_RX	5050	BYTE	R

SPI_DATA_RX - Address: 5050

Read data here. This register is a buffer. Underrun behavior - fill with zeros.

- Data type: BYTE (type index = 99)
- Read-only
- Default value: 0
- This register is a [Buffer Register](#)

Common Issues

1. Timeouts

T-Series devices have an internal watchdog that will timeout, and cause the device to reboot, if a single SPI transaction lasts longer than 250 ms. Stay safely above the following measured minimum values where device did not reboot for different numbers of bytes:

#Bytes	ThrottleValue	ClockRateHz
1	1	67 Hz (minimum)
2	4900	73 Hz
3	23900	106 Hz
4	33900	140 Hz
10	52600	342 Hz
16	57400	544 Hz
32	61500	1095 Hz

2. Noise Immunity

The clock input on the slave device can be vulnerable to noise. The clock input will notice quick edges caused by noise. If you suspect this to be a problem, add a capacitor close to the clock input of the slave. Suggested value is equal to or less than:

$$1 / (2 * \pi * f * R)$$

...where f is the SPI frequency and R is the source impedance—which is dominated in this case by the DIO (use 550 ohms for FIO and 180 ohms for EIO).

Example

The following demonstrates a simple loop-back test. Connect a jumper between DIO2 and DIO3. Data will be sent out DIO3 and read from DIO2.

```
SPI_CS_DIONUM = 0 // Use DIO0 as chip select.
SPI_CLK_DIONUM = 1 // Use DIO1 as Clock
SPI_MISO_DIONUM = 2 // Use DIO2 as MISO
SPI_MOSI_DIONUM = 3 // Use DIO3 as MOSI
SPI_MODE = 0 // Select mode zero.
SPI_SPEED_THROTTLE = 65500 // Set clock speed to ~100 kHz.
SPI_OPTIONS = 0 // No special operation
SPI_NUM_BYTES = 1 // Transfer one byte.
SPI_DATA_TX = 0x55 // Load a test data byte. eWriteNameByteArray is the easiest way to write the outgoing SPI data.
SPI_GO = 1 // Initiate the transfer
```

At this point the T4/T7 will run the SPI transaction. If no errors are returned, the received data can be read.

SPI_DATA_RX = ? - Read a test data byte. [eReadNameByteArray](#) is the easiest way to read the received SPI data.

Because of the loop-back, the data read from SPI_DATA_RX will match the data written to SPI_DATA_TX.

13.5 SBUS [T-Series Datasheet]

[Log in or register](#) to post comments

Overview

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

T-Series SBUS Support

T-series devices support SBUS at a higher level when compared to other serial communication protocols. Protocols such as SPI and I2C provide direct access to the bus so that arrays of bytes can be sent and received, but T-series devices implement all the functionality necessary to run the SBUS and convert the binary data to relative humidity (RH) or temperature.

[Lua scripting](#) is often convenient for serial applications. For example, a script can run the serial communication at a specified interval, and put the result in USER_RAM registers. The host software can read from the USER_RAM registers when convenient. This puts the complications of serial communication in a script running on the T-series device. The [Lua examples](#) contain several serial communication examples.

Default DIO Assignments

The lines used for data, clock and power are set at startup according to the below list. The assignments can be changed as desired.

T4

- **FIO4**: Data Line
- **FIO5**: Clock Line
- **FIO6**: Power line. Will be set to output high. Can be disabled.

T7

- **FIO0**: Data Line
- **FIO1**: Clock Line
- **FIO2**: Power line. Will be set to output high. Can be disabled.

How-To

SBUS on T-series LabJack devices is set up to work with the [EI-1050](#) probe. SHT1x and 7x sensors can be directly used as well. The main difference is that the EI-1050 adds an enable line.

EI-1050

The [EI-1050 datasheet](#) should be referenced for hardware connections. The temperature and humidity can be read using the SBUS#_TEMP and SBUS#_RH registers. The register index number (#) informs the T-series device which DIO line enable line is connected to. When one of the above registers are read the following steps are performed by firmware:

1. The power line will be set to output high.
2. The enable line will be set to output.
3. The EI-1050 sensor will be instructed to measure temperature or humidity.
4. When the measurement is complete, the result and the checksum will be read from the sensor.
5. Firmware will then verify the checksum.
6. If the checksum failed, an error will be returned. If the checksum passed the binary value will be converted to a decimal number which is then returned.

SHT1x or 7x sensor

To use a SHT1x and 7x sensor, connect the data and clock lines to DIO of your choosing. The sensor should be powered from 3.3 V. A digital IO set to output high will work. The direction settings can be handled by firmware if desired. 5 V should not be used to power the SHT1x and 7x sensors. Doing so will create a logic level mismatch.

The Enable line control and power line control can be disabled.

1. Disable Enable line control. The enable line will be disabled if set to the same line as the clock or data line.
2. Set power control (optional). Set the SBUS_ALL_POWER_DIONUM register to match the hardware configuration. Power control can be disabled by setting the power line to an invalid number, like 100.
3. Set the data and clock lines to match the hardware configuration.
4. Read from the temperature or humidity register as desired.

Troubleshooting

Checksum or acknowledgement errors can indicate that the clock speed is too fast for the hardware configuration. Use the SBUS_ALL_CLOCK_SPEED register to reduce the clock speed.

Register Listing

SBUS Registers

Name	Start Address	Type	Access
SBUS#(0:22)_TEMP	30100	FLOAT32	R
SBUS#(0:22)_RH	30150	FLOAT32	R
SBUS#(0:22)_DATA_DIONUM	30200	UINT16	R/W
SBUS#(0:22)_CLOCK_DIONUM	30225	UINT16	R/W
SBUS_ALL_DATA_DIONUM	30275	UINT16	R/W
SBUS_ALL_CLOCK_DIONUM	30276	UINT16	R/W
SBUS_ALL_POWER_DIONUM	30277	UINT16	R/W
SBUS_ALL_CLOCK_SPEED	30278	UINT16	R/W

SBUS#(0:22)_TEMP - Starting Address: 30100

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

- Data type: FLOAT32 (type index = 3)
- Read-only
- T7:
 - Minimum [firmware](#) version: 1.0056

Expanded Names

Addresses

SBUS0_TEMP, SBUS1_TEMP, SBUS2_TEMP, SBUS3_TEMP, SBUS4_TEMP, SBUS5_TEMP, SBUS6_TEMP, SBUS7_TEMP, SBUS8_TEMP, SBUS9_TEMP, SBUS10_TEMP, SBUS11_TEMP, SBUS12_TEMP, SBUS13_TEMP, SBUS14_TEMP, SBUS15_TEMP, SBUS16_TEMP, SBUS17_TEMP, SBUS18_TEMP, SBUS19_TEMP, SBUS20_TEMP, SBUS21_TEMP, SBUS22_TEMP Show All	30100, 30102, 30104, 30106, 30108, 30110, 30112, 30114, 30116, 30118, 30120, 30122, 30124, 30126, 30128, 30130, 30132, 30134, 30136, 30138, 30140, 30142, 30144 Show All
--	--

SBUS#(0:22)_RH - Starting Address: 30150

Reads humidity in % from an external 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.

- Data type: FLOAT32 (type index = 3)
- Read-only
- T7:
 - Minimum [firmware](#) version: 1.0056

Expanded Names

Addresses

SBUS0_RH, SBUS1_RH, SBUS2_RH, SBUS3_RH, SBUS4_RH, SBUS5_RH, SBUS6_RH, SBUS7_RH, SBUS8_RH, SBUS9_RH, SBUS10_RH, SBUS11_RH, SBUS12_RH, SBUS13_RH, SBUS14_RH, SBUS15_RH, SBUS16_RH, SBUS17_RH, SBUS18_RH, SBUS19_RH, SBUS20_RH, SBUS21_RH, SBUS22_RH Show All	30150, 30152, 30154, 30156, 30158, 30160, 30162, 30164, 30166, 30168, 30170, 30172, 30174, 30176, 30178, 30180, 30182, 30184, 30186, 30188, 30190, 30192, 30194 Show All
--	--

SBUS#(0:22)_DATA_DIONUM - Starting Address: 30200

This is the DIO# that the external sensor's data line is connected to.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Default = FIO0
 - Minimum [firmware](#) version: 1.0056
- T4:
 - Default = FIO4

Expanded Names

Addresses

SBUS0_DATA_DIONUM, SBUS1_DATA_DIONUM, SBUS2_DATA_DIONUM, SBUS3_DATA_DIONUM, SBUS4_DATA_DIONUM, SBUS5_DATA_DIONUM, SBUS6_DATA_DIONUM, SBUS7_DATA_DIONUM, SBUS8_DATA_DIONUM, SBUS9_DATA_DIONUM, SBUS10_DATA_DIONUM, SBUS11_DATA_DIONUM, SBUS12_DATA_DIONUM, SBUS13_DATA_DIONUM, SBUS14_DATA_DIONUM, SBUS15_DATA_DIONUM, SBUS16_DATA_DIONUM, SBUS17_DATA_DIONUM, SBUS18_DATA_DIONUM, SBUS19_DATA_DIONUM, SBUS20_DATA_DIONUM, SBUS21_DATA_DIONUM, SBUS22_DATA_DIONUM Show All	30200, 30201, 30202, 30203, 30204, 30205, 30206, 30207, 30208, 30209, 30210, 30211, 30212, 30213, 30214, 30215, 30216, 30217, 30218, 30219, 30220, 30221, 30222 Show All
---	--

SBUS#(0:22)_CLOCK_DIONUM - Starting Address: 30225

This is the DIO# that the external sensor's clock line is connected to.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 1
- T7:
 - Default = FIO1
 - Minimum [firmware](#) version: 1.0056
- T4:
 - Default = FIO5

Expanded Names

Addresses

SBUS0_CLOCK_DIONUM, SBUS1_CLOCK_DIONUM, SBUS2_CLOCK_DIONUM, SBUS3_CLOCK_DIONUM, SBUS4_CLOCK_DIONUM, SBUS5_CLOCK_DIONUM, SBUS6_CLOCK_DIONUM, SBUS7_CLOCK_DIONUM, SBUS8_CLOCK_DIONUM, SBUS9_CLOCK_DIONUM, SBUS10_CLOCK_DIONUM, SBUS11_CLOCK_DIONUM, SBUS12_CLOCK_DIONUM, SBUS13_CLOCK_DIONUM, SBUS14_CLOCK_DIONUM, SBUS15_CLOCK_DIONUM, SBUS16_CLOCK_DIONUM, SBUS17_CLOCK_DIONUM, SBUS18_CLOCK_DIONUM, SBUS19_CLOCK_DIONUM, SBUS20_CLOCK_DIONUM, SBUS21_CLOCK_DIONUM, SBUS22_CLOCK_DIONUM Show All	30225, 30226, 30227, 30228, 30229, 30230, 30231, 30232, 30233, 30234, 30235, 30236, 30237, 30238, 30239, 30240, 30241, 30242, 30243, 30244, 30245, 30246, 30247 Show All
--	--

SBUS_ALL_DATA_DIONUM - Address: 30275

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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

SBUS_ALL_CLOCK_DIONUM - Address: 30276

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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 1
- T7:
 - Minimum [firmware](#) version: 1.0056

SBUS_ALL_POWER_DIONUM - Address: 30277

Sets the power line. This DIO is set to output-high upon any read of SBUS#_TEMP or SBUS#_RH. Default is FIO6 for the T4 and FIO2 for the T7. 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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:
 - Default value: 2
 - Minimum [firmware](#) version: 1.0056
- T4:
 - Default value: 6

SBUS_ALL_CLOCK_SPEED - Address: 30278

Sets the clock speed. The clock is software generated so the resulting frequency is not exact. Valid range is 0-65535. Larger values are faster. 0 is the fastest option and is equivalent to 65536. A value of 0 is ~200 kHz. A value of 65000 is ~9.1 kHz.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 65000
- T7:
 - Minimum [firmware](#) version: 1.0142

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 wires from each probe to the lines specified by the default config:

<u>T7</u>	<u>T4</u>	<u>EI-1050</u>
GND	GND	Ground (black)
FIO0 / DIO0	FIO4 / DIO4	Data (green)
FIO1 / DIO1	FIO5 / DIO5	Clock (white)
FIO2 / DIO2	FIO6 / DIO6	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. Let's use:

<u>T7</u>	<u>T4</u>	<u>EI-1050</u>
FIO3 / DIO3	FIO7 / DIO7	Enable ProbeA (brown)
EIO0 / DIO8	EIO0 / DIO8	Enable ProbeB (brown)
EIO1 / DIO9	EIO1 / DIO9	Enable ProbeC (brown)
EIO2 / DIO10	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.

SHT1x or SHT7x sensor using default configuration:

This example was made for the T7. To adapt it for the T4 simply change the line numbers 0-2 to 4-6.

In this example we connect the 4 connections from the raw Sensirion sensor to the lines specified by the default config:

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

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 (FIO0 to FIO2) and perhaps even a capacitor from Clock to GND. The filter cap should be near the sensor pins, not the LabJack terminals. Suggested value is equal to or less than $1 / (2 * \pi * f * R)$, where f is the clock frequency and R is the source impedance—which is dominated in this case by the DIO (use 550 ohms for FIO and 180 ohms for EIO). 1nF or 10nF should be good for any DIO at the default SBUS clock frequency of 9100 Hz.

You can now read from SBUS0_TEMP & SBUS0_RH without writing any config values. In LJLogM, for example, just put the desired register name in any row. The SHT71 does not have an enable, so we set "#" equal to the data line (0) or clock line (1) which is a signal to the T-series device to not control an enable line.

Since the raw SHT sensors do not have an enable, each sensor must have its own Data and Clock lines.

If an SHT sensor is not working at this point, an oscilloscope or logic analyzer will likely be required to troubleshoot.

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

This example was made for the T7. To adapt it for the T4 the line numbers need to be changed as FIO0-3 do not exist on the T4.

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 enable line 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
```

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.

13.6 1-Wire [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

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.

1-Wire is considered an advanced topic. A good knowledge of the protocol is recommended. Troubleshooting may require a logic analyzer or oscilloscope.

The [1-Wire app-note](#) is a good place to start.

1-Wire Support

T-series devices can send and receive data over the 1-Wire bus. The below sections cover the necessary hardware setup considerations and describe how to use the various operations that can be performed over the 1-Wire bus.

[Lua scripting](#) is often convenient for serial applications. For example, a script can run the serial communication at a specified interval, and put the results in USER_RAM registers. The host software can read from the USER_RAM registers when convenient. This puts the complications of serial communication in a script running on the T-series device. The Lua examples contain several serial communication examples.

Hardware

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

FIO lines cannot be used for 1-Wire. They have too much impedance, which prevents the signal from reaching the logic thresholds.

T-series devices support 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_TX: The number of bytes to transmit to the device. Has no effect when the ROM function is set to Search or Read.

ONEWIRE_NUM_BYTES_RX: The number of bytes to read from the device. Has no effect 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_BRANCHS_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_RX and write the data to ONEWIRE_DATA_RX.

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_TX and write the data to ONEWIRE_DATA_TX.

Example

Configure the T4/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 = 14
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 T4/T7 will read the ROM from the connected device and place it in ONEWIRE_SEARCH_RESULT. 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 0x44.

```
ONEWIRE_FUNCTION = 0x55
ONEWIRE_ROM = 0x1D000005908D4728
ONEWIRE_NUM_BYTES_TX = 1
ONEWIRE_DATA_TX = [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_TX = 1
ONEWIRE_NUM_BYTES_RX = 9
ONEWIRE_DATA_TX = [0xBE]
ONEWIRE_GO = 1
```

We can now read the 9 bytes from ONEWIRE_DATA_RX:

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 $\text{data}[0] + \text{data}[1]*256$. 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.

Register Listing

1-Wire Registers

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

ONEWIRE_DQ_DIONUM - Address: 5300

The data-line DIO number.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_DPU_DIONUM - Address: 5301

The dynamic pullup control DIO number.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_OPTIONS - Address: 5302

Controls advanced features. Value is a bitmask. bit 0: reserved, bit 1: reserved, bit 2: 1=DPU Enabled 0=DPU Disabled, bit 3: DPU Polarity 1=Active state is high, 0=Active state is low (Dynamic Pull-Up)

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_FUNCTION - Address: 5307

Set the ROM function to use. 0xF0=Search, 0xCC=Skip, 0x55=Match, 0x33=Read.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_NUM_BYTES_TX - Address: 5308

Number of data bytes to be sent.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_NUM_BYTES_RX - Address: 5309

Number of data bytes to be received.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_GO - Address: 5310

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

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_ROM_MATCH_H - Address: 5320

Upper 32-bits of the ROM to match.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_ROM_MATCH_L - Address: 5322

Lower 32-bits of the ROM to match.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_ROM_BRANCHS_FOUND_H - Address: 5332

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

- Data type: UINT32 (type index = 1)
- Read-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_ROM_BRANCHS_FOUND_L - Address: 5334

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

- Data type: UINT32 (type index = 1)
- Read-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_SEARCH_RESULT_H - Address: 5328

Upper 32-bits of the search result.

- Data type: UINT32 (type index = 1)
- Read-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_SEARCH_RESULT_L - Address: 5330

Lower 32-bits of the search result.

- Data type: UINT32 (type index = 1)
- Read-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_PATH_H - Address: 5324

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_PATH_L - Address: 5326

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_DATA_TX - Address: 5340

Data to be transmitted over the 1-wire bus. This register is a buffer.

- Data type: BYTE (type index = 99)
- Write-only
- Default value: 0
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 1.0056

ONEWIRE_DATA_RX - Address: 5370

Data received over the 1-wire bus. This register is a buffer. Underrun behavior - buffer is static, old data will fill the extra locations, firmware 1.0225 changes this to read zeros.

- Data type: BYTE (type index = 99)
- Read-only
- Default value: 0
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 1.0056

13.7 Asynchronous Serial [T-Series Datasheet]

[Log in or register](#) to post comments

Overview

The T-Series devices have universal asynchronous receiver-transmitter (UART) functionality available that supports 3.3V logic level (CMOS/TTL) asynchronous (asynch) serial communication. The TX (transmit) and RX (receive) lines can appear on any digital I/O. Baud rates up to 38400 are supported, but the device's processor is heavily loaded at that rate. The number of data bits, number of stop bits, and parity are all controllable.

Asynchronous (UART) vs. RS-232:

The T4/T7's asynchronous support and the RS-232 standard are the same in terms of timing and protocol, but different in terms of electrical specifications. Connection to an RS-232 device will require a converter chip such as the MAX233, which inverts the logic and shifts the voltage levels. On the T4/T7, a low is 0 volts (inputs recognize 0.0 to 0.5) and a high (1) is 3.3 volts (inputs recognize 2.64 to 5.8 volts). With RS-232, a low (0) is 3 to 25 volts and a high (1) is -3 to -25 volts; RS-232 has unique voltage levels and is inverted.

Lua Scripting:

[Lua scripting](#) is often convenient for serial applications. For example, you might write a script that does the serial communication to get a new reading from the serial device once per second, and puts that reading in a USER_RAM register. This puts the complications of serial communication in a script running on the T4/T7 itself, and then the host software can just do a simple read of the USER_RAM register when convenient. We have many serial [examples available for Lua scripting](#).

A direct connection to a serial device is preferable:

This serial link is not an alternative to the USB/Ethernet/WiFi connection. Rather, the host application will write/read data to/from the T4/T7 over USB/Ethernet/WiFi, and the T4/T7 communicates with some other device using the serial protocol. Using this serial protocol is considered an advanced topic. A good knowledge of the protocol is recommended, and a logic analyzer or oscilloscope might be needed for troubleshooting.

If it is practical to run a cable directly from the host computer to the serial device, that is usually a better than putting the T4/T7 in between. Use a standard USB<=>RS-232 adapter/converter/dongle (or RS-485 or RS-422).

Multiple asynchronous ports on a single LabJack:

The asynchronous feature can only be enabled on one pair of pins at a time, and to be more specific only one RX pin can read data at a time. When the asynchronous feature is enabled on a pair of pins, a buffer is set up and the RX pin reads any data that comes in and stores it in the buffer. This is useful for devices that spontaneously send out data where all that data is wanted all the time. Most serial devices, however, act in a command-response manner where the LabJack sends a command that requests a reading and the device responds with the reading. For these it is easy to do multiple ... just re-do the configuration writes whenever communication is desired on different pins.

How-To

1. Initial Configuration
2. Transmit Data
3. Receive Data
4. Debugging data parity errors (if enabled)

1. Initial Configuration

Several registers need to be written to in order to configure the T-Series device for Asynch communication.

- TX/RX data lines (ASYNCH_TX_DIONUM, ASYNCH_RX_DIONUM)
- Baud rate configuration (ASYNCH_BAUD)
- Configure RX buffer size (ASYNCH_RX_BUFFER_SIZE_BYTES)
- Configure number of bits, number of stop bits, and the parity. (ASYNCH_NUM_DATA_BITS, ASYNCH_NUM_STOP_BITS, ASYNCH_PARITY)

After configuring the various registers, the ASYNCH feature should be enabled by writing a 1 to (ASYNCH_ENABLE).

Asynchronous Serial Configuration Registers

Name	Start Address	Type	Access
ASYNCH_TX_DIONUM	5410	UINT16	R/W
ASYNCH_RX_DIONUM	5405	UINT16	R/W
ASYNCH_BAUD	5420	UINT32	R/W
ASYNCH_RX_BUFFER_SIZE_BYTES	5430	UINT16	R/W
ASYNCH_NUM_DATA_BITS	5415	UINT16	R/W
ASYNCH_NUM_STOP_BITS	5455	UINT16	R/W
ASYNCH_PARITY	5460	UINT16	R/W
ASYNCH_ENABLE	5400	UINT16	R/W

ASYNCH_TX_DIONUM - Address: 5410

The DIO line that will transmit data. (TX)

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

ASYNCH_RX_DIONUM - Address: 5405

The DIO line that will receive data. (RX)

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

ASYNCH_BAUD - Address: 5420

The symbol rate that will be used for communication. 9600 is typical. Up to 38400 works, but heavily loads the T7's processor.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

ASYNCH_RX_BUFFER_SIZE_BYTES - Address: 5430

Number of bytes to use for the receiving buffer. Max is 2048. 0 = 200

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

ASYNCH_NUM_DATA_BITS - Address: 5415

The number of data bits per frame. 0-8, 0=8.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

ASYNCH_NUM_STOP_BITS - Address: 5455

The number of stop bits. Values: 1 = one stop bit, 2 = two stop bits, 0 = one stop bit.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

ASYNCH_PARITY - Address: 5460

Parity setting: 0=none, 1=odd, 2=even.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

ASYNCH_ENABLE - Address: 5400

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

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)

2. Transmit Data

In order to transmit data a user must do the following:

1. Configure the number of bytes that needs to be sent (ASYNCH_NUM_BYTES_TX)
2. Send data to the T-Series device using the [LJM_eWriteNameByteArray](#) function (ASYNCH_DATA_TX)
3. Write a 1 to the "GO" register (ASYNCH_TX_GO)
note: The process of writing a 1 to the GO register instructs the T-Series device to transmit the buffered data via the TX line.

Asynchronous Serial Data Transmission Registers

Name	Start Address	Type	Access
ASYNCH_NUM_BYTES_TX	5440	UINT16	R/W
ASYNCH_DATA_TX	5490	UINT16	W
ASYNCH_TX_GO	5450	UINT16	W

ASYNCH_NUM_BYTES_TX - Address: 5440

The number of bytes to be transmitted after writing to GO. Max is 256.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)

ASYNCH_DATA_TX - Address: 5490

Write data to be transmitted here. This register is a buffer.

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0
- This register is a [Buffer Register](#)

ASYNCH_TX_GO - Address: 5450

Write a 1 to this register to initiate a transmission.

- Data type: UINT16 (type index = 0)
- Write-only
- Default value: 0

3. Receive Data

T-Series devices buffer received Asynch data up to the size defined in the configuration step when writing to the register "ASYNCH_RX_BUFFER_SIZE_BYTES". The usual method for reading data from the buffer is to do the following:

1. Read how many bytes of information have been received by the device (ASYNCH_NUM_BYTES_RX)
2. Read data from the T-Series device RX buffer using the `LJM_eReadNameByteArray` function (ASYNCH_DATA_RX)
note: When ever possible, it is recommended to read an even number of bytes from the DATA_RX buffer.

Asynchronous Serial Data Receiving Registers

Name	Start Address	Type	Access
ASYNCH_NUM_BYTES_RX	5435	UINT16	R
ASYNCH_DATA_RX	5495	UINT16	R

ASYNCH_NUM_BYTES_RX - Address: 5435

The number of data bytes that have been received.

- Data type: UINT16 (type index = 0)
- Read-only
- Default value: 0

ASYNCH_DATA_RX - Address: 5495

Read received data from here. This register is a buffer. Underrun behavior - fill with zeros.

- Data type: UINT16 (type index = 0)
- Read-only
- Default value: 0
- This register is a [Buffer Register](#)

4. Debugging Data Parity Errors

Asynchronous Serial Data Parity Register

Name	Start Address	Type	Access
ASYNCH_NUM_PARITY_ERRORS	5465	UINT16	R/W

ASYNCH_NUM_PARITY_ERRORS - Address: 5465

The number of parity errors that have been detected. Cleared when UART is enabled. Can also be cleared by writing 0.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0

Examples

For performing asynchronous communication from a computer, see the [LJM C examples](#) or the [LJM LabView examples](#).

For performing asynchronous communication on device, see the [Lua scripting examples](#).

14.0 Analog Inputs [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Analog Inputs Overview

Basics: An [analog input](#) (commonly referred as AIN or AI) uses an analog-to-digital converter (ADC) to convert a voltage level into a digital value. LabJack T-series devices have multiple analog inputs.

Common Uses: For information on measuring various analog signals such as typical analog sensors, measuring small voltages, measuring current (4-20mA), and measuring resistance see the [Analog Input \(App Note\)](#). There are also application specific app notes for [temperature sensors](#) and [thermocouples](#).

How to read AIN: See [3.0 Communication](#) for communication basics. Also, LabJack [Kipling's Dashboard](#) tab shows live AIN values.

Configuration: T-series AIN readings can be configured. See below for information on:

- Resolution Index
- Flexible I/O - T4 Only
- Range - T7 Only
- Single-ended or Differential - T7 Only
- Settling
- Other Considerations

AIN Extended Features: T-series [AIN Extended Features](#) simplify operations such as:

- Reading thermocouples and thermistors
- Calculating RMS, RTD, average/max/min, average/threshold, and circuit element resistance

Extended Channels - T7 Only: The T7's [Extended Channels](#) range provides extra AIN channels.

AIN Summary By Device

T4

Analog Inputs: 4 high-voltage (AIN0-AIN3)
8 low-voltage (AIN4-AIN11)

Voltage Ranges: $\pm 10V$ or $0-2.5V$ ([Appendix A-3](#) and [Appendix A-3-2](#))

Resolution: 12-bit

Max Data Rate: 40 ksamples/second in stream mode ([Appendix A-1](#))

Sampling Modes: Single-ended

T7

Analog Inputs: 14 (AIN0-AIN13)

Voltage Ranges: $\pm 10V$, $\pm 1V$, $\pm 0.1V$, and $\pm 0.01V$ ([Appendix A-3](#) and [Appendix A-3-2](#))

ADC

Resolution: T7 = 16-bit

T7- = 16-bit and 24-

Pro bit

Effective

Resolution: T7 = 16-bit to 19-bit

T7-Pro = 16-bit and 22-bit

At Gain 1x. For more details see [Appendix A-3-1](#).

Max Data Rate: 100 ksamples/second in stream mode ([Appendix A-1](#))

Sampling Modes: Configurable as single-ended or differential

Extended Channels: The number of analog inputs can be extended to 84 with a [Mux80](#) (AIN48-AIN127).

Available AIN Channels

Each T-series device exposes:

- Some AINs on the screw terminals.
- Additional AINs on a connector (either a [DB15](#) or a [DB37](#)).

T4

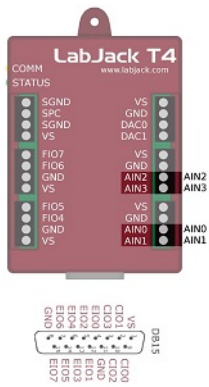


Figure 14.0-1 T4 Dedicated Analog Inputs

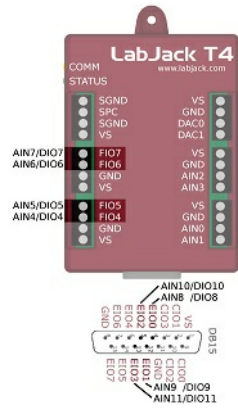


Figure 14.0-2 T4 Flexible I/O

The LabJack T4 has up to 12 built-in analog inputs, readable as AIN0-AIN11:

- AIN0-AIN3 are dedicated ± 10 volt inputs:
 - Available on screw terminals AIN0-AIN3.
 - Always ± 10 volt analog inputs.
- AIN4-AIN7 are flexible 0 to +2.5 volt inputs:
 - Available on screw terminals FIO4-FIO7.
 - Configurable to be digital inputs/outputs. See "Flexible I/O" below.
- AIN8-AIN11 are flexible 0 to +2.5 volt inputs:
 - Available on the DB15 connector.
 - Configurable to be digital inputs/outputs. See "Flexible I/O" below.

T4 AIN Channel Registers			
Name	Start Address	Type	Access
AIN#(0:11)	0	FLOAT32	R
AIN#(0:11) - Starting Address: 0 Returns the voltage of the specified analog input. <ul style="list-style-type: none"> • Data type: FLOAT32 (type index = 3) • Read-only • This register may be streamed 			
Expanded Names		Addresses	
AIN0, AIN1, AIN2, AIN3, AIN4, AIN5, AIN6, AIN7, AIN8, AIN9, AIN10, AIN11 Show All		0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22 Show All	

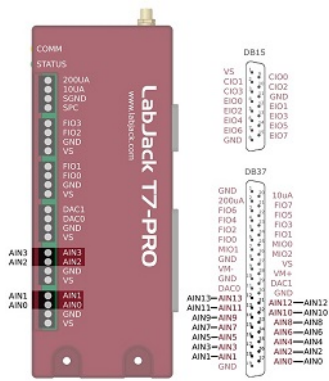


Figure 14.0-3 T7 Analog Inputs

The LabJack T7 has 14 built-in analog inputs, readable as AIN0-13:

- AIN0-AIN3 are available on the screw terminals and on the [DB37](#) connector. See "Duplicated Terminals" below.
- AIN4-AIN13 are available only on the [DB37](#) connector.

T7 AIN Channel Registers			
Name	Start Address	Type	Access
AIN#(0:13)	0	FLOAT32	R
AIN#(0:13) - Starting Address: 0			
Returns the voltage of the specified analog input.			
<ul style="list-style-type: none"> • Data type: FLOAT32 (type index = 3) • Read-only • This register may be streamed 			
Expanded Names		Addresses	
AIN0, AIN1, AIN2, AIN3, AIN4, AIN5, AIN6, AIN7, AIN8, AIN9, AIN10, AIN11, AIN12, AIN13		0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26	

In addition to the 14 built-in analog inputs, the T7 has special and extended channels.

- - AIN14 is internally connected to an internal [temperature sensor](#).
 - AIN15 is internally connected to GND. Useful for measuring noise or looking at offset error.
 - AIN16-AIN47 are optional extended channels that can be created with custom analog input muxing circuitry. See [Section 14.2 Extended Channels](#) for more information.
 - AIN48-AIN127 are extended channels that are available when using a [Mux80](#).

T7 Extended AIN Channel Registers			
Name	Start Address	Type	Access
AIN#(48:127)	96	FLOAT32	R
AIN#(48:127) - Starting Address: 96			
Returns the voltage of the specified analog input.			
<ul style="list-style-type: none"> • Data type: FLOAT32 (type index = 3) • Read-only • This register may be streamed 			
Expanded Names		Addresses	
AIN48, AIN49, AIN50, AIN51, AIN52, AIN53, AIN54, AIN55, AIN56, AIN57, AIN58, AIN59, AIN60, AIN61, AIN62, AIN63, AIN64, AIN65, AIN66, AIN67, AIN68, AIN69, AIN70, AIN71, AIN72, AIN73, AIN74, AIN75, AIN76, AIN77, AIN78, AIN79, AIN80, AIN81, AIN82, AIN83, AIN84, AIN85, AIN86, AIN87, AIN88, AIN89, AIN90, AIN91, AIN92, AIN93, AIN94, AIN95, AIN96, AIN97, AIN98, AIN99, AIN100, AIN101, AIN102, AIN103, AIN104, AIN105, AIN106, AIN107, AIN108, AIN109, AIN110, AIN111, AIN112, AIN113, AIN114, AIN115, AIN116, AIN117, AIN118, AIN119, AIN120, AIN121, AIN122, AIN123, AIN124, AIN125, AIN126, AIN127		96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254	

Example:

To read a voltage connected to AIN1, read AIN1 (address 2). The result will be in the form of a floating point number, like 8.82332V.

Resolution Index

A higher resolution index results in lower noise and higher effective resolution but increases sample times.

Resolution Index Registers			
Name	Start Address	Type	Access
AIN#(0:13)_RESOLUTION_INDEX	41500	UINT16	R/W
AIN_ALL_RESOLUTION_INDEX	43903	UINT16	R/W

AIN#(0:13)_RESOLUTION_INDEX - Starting Address: 41500

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Valid values: 0-8 for T7, 0-12 for T7-Pro. Default value of 0 corresponds to an index of 8 (T7) or 9 (T7-Pro).
- T4:
 - Valid values: 0-5. Default value of 0 corresponds to an index of 5.

Expanded Names

AIN0_RESOLUTION_INDEX, AIN1_RESOLUTION_INDEX, AIN2_RESOLUTION_INDEX, AIN3_RESOLUTION_INDEX,
AIN4_RESOLUTION_INDEX, AIN5_RESOLUTION_INDEX, AIN6_RESOLUTION_INDEX, AIN7_RESOLUTION_INDEX,
AIN8_RESOLUTION_INDEX, AIN9_RESOLUTION_INDEX, AIN10_RESOLUTION_INDEX, AIN11_RESOLUTION_INDEX,
AIN12_RESOLUTION_INDEX, AIN13_RESOLUTION_INDEX [Show All](#)

Addresses

41500, 41501, 41502, 41503, 41504,
41505, 41506, 41507, 41508, 41509,
41510, 41511, 41512, 41513 [Show All](#)

AIN_ALL_RESOLUTION_INDEX - Address: 43903

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times. 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.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- T7:
 - Valid values: 0-8 for T7, 0-12 for T7-Pro. Default value of 0 corresponds to an index of 8 (T7) or 9 (T7-Pro).
 - Minimum [firmware](#) version: 0.9328
- T4:
 - Valid values: 0-5. Default value of 0 corresponds to an index of 5.

Defaults

Setting resolution index to 0 sets the default resolution index to:

- 5 for command-response mode on a T4
- 8 for command-response mode on a T7
- 9 for command-response mode on a T7-Pro

Example:

To read AIN1 with resolution index 4:

1. Set the AIN1 resolution index to 4 by writing 5 to AIN1_RESOLUTION_INDEX (address 41501)
2. Read AIN1

Remarks

Resolution index ranges:

- T4 resolution index ranges from 0 to 5
- T7 resolution index ranges from 0 to 8
- T7-Pro resolution index ranges from 0 to 12
 - Settings 9-12 use the alternate high-resolution converter (24-bit sigma-delta)

AIN#(0:13)_RESOLUTION_INDEX and AIN_ALL_RESOLUTION_INDEX do not apply to [stream mode](#), for which there is a dedicated resolution index register.

For typical noise levels and sample times at different combinations of resolution index and gain, see [Appendix A-3-1 Noise and Resolution](#).

For general discussion on the meaning of resolution, see the [Noise and Resolution App Note](#).

Flexible I/O - T4 Only

Flexible I/O are I/O ports that may be configured as analog inputs, or as digital inputs or outputs.

Flexible I/O Configuration			
Name	Start Address	Type	Access
DIO_INHIBIT	2900	UINT32	R/W
DIO_ANALOG_ENABLE	2880	UINT32	R/W

DIO_INHIBIT - Address: 2900

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0

Constant	Value
Affected	0
Ignored	1

DIO_ANALOG_ENABLE - Address: 2880

Read or write the analog configuration of all digital I/O in a single binary-encoded value. 1=Analog mode and 0=Digital mode. When switching from analog to digital, the lines will be set to input. Writes are filtered by the value in `DIO_INHIBIT`.

- Data type: UINT32 (type index = 1)
- Readable and writable

Constant	Value
Digital mode	0
Analog mode	1

AIN4-AIN11 are flexible I/O. To configure these channels as analog inputs:

- Set the correct bit of `DIO_INHIBIT` to 0
- Set the correct bit of `DIO_ANALOG_ENABLE` to 1

The bit to set of `DIO_INHIBIT` and of `DIO_ANALOG_ENABLE` is the same as the channel number. For example, to configure AIN4 (screw terminal FIO4) as an analog input:

- Set bit 4 of `DIO_INHIBIT` to 0
- Set bit 4 of `DIO_ANALOG_ENABLE` to 1

AIN0-AIN3 are dedicated analog inputs and cannot be configured as digital I/O.

Note that simply doing an analog read on AIN4-AIN11 automatically configures that line (FIO4-EIO3) as analog, so few people use the 2 registers mentioned above. See "Flexible I/O Auto-Configuration" in [Section 13.1 Flexible I/O](#).

Single-ended or Differential - T7 Only

Single-ended AIN readings are read with ground (GND) as a reference point. Differential readings use a second AIN as a reference point. For more, see the [Differential Readings App Note](#).

Negative Channel Registers			
Name	Start Address	Type	Access
AIN#(0:13)_NEGATIVE_CH	41000	UINT16	R/W
AIN_ALL_NEGATIVE_CH	43902	UINT16	R/W

AIN#(0:13)_NEGATIVE_CH - Starting Address: 41000

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 199
- T7:
 - For base differential channels, positive must be even channel from 0-12 and negative must be positive+1. For extended channels 16-127, see Mux80 datasheet.

Expanded Names	Addresses
AIN0_NEGATIVE_CH, AIN1_NEGATIVE_CH, AIN2_NEGATIVE_CH, AIN3_NEGATIVE_CH, AIN4_NEGATIVE_CH, AIN5_NEGATIVE_CH, AIN6_NEGATIVE_CH, AIN7_NEGATIVE_CH, AIN8_NEGATIVE_CH, AIN9_NEGATIVE_CH, AIN10_NEGATIVE_CH, AIN11_NEGATIVE_CH, AIN12_NEGATIVE_CH, AIN13_NEGATIVE_CH Show All	41000, 41001, 41002, 41003, 41004, 41005, 41006, 41007, 41008, 41009, 41010, 41011, 41012, 41013 Show All

AIN_ALL_NEGATIVE_CH - Address: 43902

A write to this global parameter affects all AIN. Writing 1 will set all AINs to differential. Writing 199 will set all AINs to single-ended. A read will return 1 if all AINs are set to differential and 199 if all AINs are set to single-ended. If AIN configurations are not consistent 0xFFFF will be returned.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 199
- T7:
 - Minimum [firmware](#) version: 0.9328

The AIN#(0:13)_NEGATIVE_CH and AIN_ALL_NEGATIVE_CH parameters configure whether the AIN performs differential vs. single-ended readings (not to be confused with bipolar and unipolar—the T7 is always bipolar).

Example:

To take a differential reading on AIN2, set AIN3 as its negative channel (AIN2-AIN3):

- Write 3 to AIN2_NEGATIVE_CH (address 41002)
- Read AIN2

To read AIN2 single-ended again (AIN2-GND):

- Write 199 to AIN2_NEGATIVE_CH (address 41002)
- Read AIN2

Built-in AIN

For AIN0 through AIN13, differential channels are adjacent even/odd pairs such that the positive channel is even and the negative channel is greater than the positive channel by 1. Odd channels, such as AIN3_NEGATIVE_CH (address 41003), should not be written to because only an even channel can have an associated negative channel.

Temperature and Ground

AIN14 is the [internal temperature sensor](#) and AIN15 is GND. Neither can be read differentially.

Extended AIN

For AIN16 and greater, the rules for pairing differential channels is different. See [14.2 Extended Channels](#) or the [Mux80 datasheet](#) for more.

Range / Gain - T7 Only

T7 Internal Amplifier: The analog inputs are connected to a high-impedance instrumentation amplifier, as shown in [Figure 4.2-2](#). This in-amp does the following:

- Buffers the signal for the internal ADCs
- Allows for single-ended or differential conversions
- Provides gains of x1, x10, x100, and x1000 (corresponding to ranges of $\pm 10V$, $\pm 1V$, $\pm 0.1V$, and $\pm 0.01V$, respectively).

Range Registers			
Name	Start Address	Type	Access
AIN#(0:13)_RANGE	40000	FLOAT32	R/W
AIN_ALL_RANGE	43900	FLOAT32	R/W

AIN#(0:13)_RANGE - Starting Address: 40000

The range/span of each analog input. Write the highest expected input voltage.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - Valid values/ranges: 0.0=Default=> +/-10V, 10.0=> +/-10V, 1.0=> +/-1V, 0.1=> +/-0.1V, or 0.01=> +/-0.01V.
- T4:
 - Valid values/ranges: 0.0=Default=> 0-2.5 V on LV lines and ± 10 V on HV lines.

Expanded Names	Addresses
AIN0_RANGE, AIN1_RANGE, AIN2_RANGE, AIN3_RANGE, AIN4_RANGE, AIN5_RANGE, AIN6_RANGE, AIN7_RANGE, AIN8_RANGE, AIN9_RANGE, AIN10_RANGE, AIN11_RANGE, AIN12_RANGE, AIN13_RANGE	40000, 40002, 40004, 40006, 40008, 40010, 40012, 40014, 40016, 40018, 40020, 40022, 40024, 40026

[Show All](#)

AIN_ALL_RANGE - Address: 43900

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.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 0.9328

Example:

If the voltage source connected to AIN1 has a known range of 0 to 0.7V, the appropriate range for AIN1 is the $\pm 1V$ range. To read AIN1 with the $\pm 1V$ range:

1. Write 1.0 to AIN1_RANGE (address 40002)
2. Read AIN1

The range registers (AIN#(0:13)_RANGE and AIN_ALL_RANGE) control the gain of the T7's internal instrumentation amplifier. The in-amp supports gains of x1, x10, x100, and x1000.

- A range of 10 sets gain=x1 so that the analog input range is ± 10 volts. This is the default range.
- A range of 1 sets gain=x10 so that the analog input range is ± 1 volts.
- A range of 0.1 sets gain=x100 so that the analog input range is ± 0.1 volts.
- A range of 0.01 sets gain=x1000 so that the analog input range is ± 0.01 volts.

Values written to RANGE are rounded up (except for values greater than 10.0, which are rounded down). For example, writing 0.5 to AIN_ALL_RANGE will set the analog input range to ± 1 volts.

The T7 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.

Settling

The settling registers (AIN#(0:13)_SETTLING_US and AIN_ALL_SETTLING_US) set the time from a step change in the input signal to when the signal is sampled by the ADC, as 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 value 0 sets automatic settling, which is recommended for most applications. This "auto" settling ensures that the device meets specifications at any gain and resolution for source impedance up to at least 1000 ohms.

Settling Registers			
Name	Start Address	Type	Access
AIN#(0:13)_SETTLING_US	42000	FLOAT32	R/W
AIN_ALL_SETTLING_US	43904	FLOAT32	R/W

AIN#(0:13)_SETTLING_US - Starting Address: 42000

Settling time for command-response and AIN-EF readings.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - 0 = Auto. Max is 50000 (microseconds).
 - Minimum [firmware](#) version: 0.9328
- T4:
 - 0 = Auto. Max is 10000 (microseconds).

Expanded Names	Addresses
AIN0_SETTLING_US, AIN1_SETTLING_US, AIN2_SETTLING_US, AIN3_SETTLING_US, AIN4_SETTLING_US, AIN5_SETTLING_US, AIN6_SETTLING_US, AIN7_SETTLING_US, AIN8_SETTLING_US, AIN9_SETTLING_US, AIN10_SETTLING_US, AIN11_SETTLING_US, AIN12_SETTLING_US, AIN13_SETTLING_US Show All	42000, 42002, 42004, 42006, 42008, 42010, 42012, 42014, 42016, 42018, 42020, 42022, 42024, 42026 Show All

AIN_ALL_SETTLING_US - Address: 43904

Settling time for command-response and AIN-EF readings. 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. Max is 50,000 us.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - 0 = Auto. Max is 50000 (microseconds).
 - Minimum [firmware](#) version: 0.9328
- T4:
 - 0 = Auto. Max is 10000 (microseconds).

Example:

To read AIN3 with a manual settling time of 500 μ s:

1. Write 500 to AIN3_SETTLING_US (address 42006)
2. Read AIN3

Remarks

AIN#(0:13)_SETTLING_US and AIN_ALL_SETTLING_US do not apply to [stream mode](#), though they do apply to all [command-response](#) communications, including [AIN EF](#).

The timings in [Appendix A](#) are measured with "auto" settling.

See the [Analog Input Settling Time \(App Note\)](#) for more details.

Other Considerations

Streaming AIN: See [3.2 Stream Mode](#) for streaming analog inputs. Some stream configurations override the normal AIN configurations:

- Use STREAM_SETTLING_US instead of AIN_SETTLING_US.
- Use STREAM_RESOLUTION_INDEX instead of AIN_RESOLUTION_INDEX.

Floating Inputs: 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 the [floating input application note](#).

Connections: For information regarding typical analog input connections, please see the [Analog Input App Note](#).

Address Step Size:

Addresses of the FLOAT32 type increment in steps of 2 because FLOAT32 uses two sets of 16-bits. FLOAT32 registers include AIN#, AIN#_RANGE, and AIN#_SETTLING_US.

Addresses of the UINT16 type increment in steps of 1 because UNIT16 uses only one set of 16-bits. UINT16 registers include AIN#_NEGATIVE_CH and AIN#_RESOLUTION_INDEX.

T7 Only: Duplicated Terminals (AIN0-AIN3):

AIN0-AIN3 appear on the built-in screw-terminals and also on the DB37 connector. Users should only connect to one or the other, not both at the same time.

To prevent damage due to accidental short circuit, both connection paths have their own series resistor. All AIN lines have a 2.2k series resistor, and in the case of AIN0-AIN3 the duplicated connections each have their own series resistor, so if you measure the resistance between the duplicate terminals you will see about 4.4k.

Analog Input Channels Reference

Table 14.0-1 Analog Input Channel Overview

Built-in Analog Inputs	Extended Channels

AIN T4	AIN (0-3) High-Voltage $\pm 10V$	AIN (4-11) Low-Voltage 0-2.5V	N/A	AIN (48-127) N/A
T7	AIN (0-13) High-Voltage $\pm 10V$ Options: SE/Diff, Gains (1000x, 100x, 10x, 1)		Special Channels	AIN (48-127) Mux80

14.1 AIN Extended Features [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

AIN Extended Features Overview

Analog Extended Features (AIN-EF) simplify some common analog input applications. Each AIN-EF feature:

- collects one or more input samples
- performs math on the collected samples

AIN-EF is only supported in command-response mode and not in stream mode.

Kipling Walkthrough: Kipling's [Register Matrix](#) can be used to perform AIN-EF features. For example:

- [Configuring & Reading a Thermocouple](#)

Available AIN Extended Features

For any given AIN channel, one AIN-EF feature may be selected. AIN-EF indices:

Index	AIN-EF Name	Supported Devices
0:	None (disabled)	All T-series
1:	Offset and Slope	All T-series
3:	Max, Min, Avg	All T-series
4:	Resistance	All T-series
5:	Average and Threshold	All T-series
10:	RMS Flex	All T-series
11:	RMS Auto	All T-series
20:	Thermocouple type E	T7 only
21:	Thermocouple type J	T7 only
22:	Thermocouple type K	T7 only
23:	Thermocouple type R	T7 only
24:	Thermocouple type T	T7 only
25:	Thermocouple type S	T7 only
30:	Thermocouple type C	T7 only
40:	RTD PT100	All T-series
41:	RTD PT500	All T-series
42:	RTD PT1000	All T-series
50:	Thermistor using Steinhart-Hart equation	All T-series
51:	Thermistor using Beta equation	All T-series

AIN-EF Usage

To use any AIN-EF:

1. Set the EF_INDEX to select an extended feature
2. Configure the extended feature using the EF_CONFIG registers
3. Configure normal AIN configurations through the normal AIN registers
4. Read from READ_A to perform the extended feature operation
5. Read additional results from B, C, and D

For a quick example of setting up an AIN-EF in thermocouple mode, see [Configuring & Reading a Thermocouple](#).

1. Set the AIN#_EF_INDEX to select an extended feature

Name	Start Address	Type	Access
AIN#(0:13)_EF_INDEX	9000	UINT32	R/W
AIN_ALL_EF_INDEX	43906	UINT32	R/W

AIN#(0:13)_EF_INDEX - Starting Address: 9000

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple

type T; 25=Thermocouple type S; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0030
- T4:
 - The T4 does not support thermocouple modes.

Expanded Names	Addresses
AIN0_EF_INDEX, AIN1_EF_INDEX, AIN2_EF_INDEX, AIN3_EF_INDEX, AIN4_EF_INDEX, AIN5_EF_INDEX, AIN6_EF_INDEX, AIN7_EF_INDEX, AIN8_EF_INDEX, AIN9_EF_INDEX, AIN10_EF_INDEX, AIN11_EF_INDEX, AIN12_EF_INDEX, AIN13_EF_INDEX Show All	9000, 9002, 9004, 9006, 9008, 9010, 9012, 9014, 9016, 9018, 9020, 9022, 9024, 9026 Show All

AIN_ALL_EF_INDEX - Address: 43906

Write 0 to deactivate AIN_EF on all AINs. No other values may be written to this register. Reads will return the AIN_EF index if all 128 AINs are set to the same value. If values are not the same returns 0xFFFF (65535).

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0115
- T4:
 - The T4 does not support thermocouple modes.

Write to AIN#(0:14)_EF_INDEX or AIN_ALL_EF_INDEX to select the AIN extended feature.

2. Configure the extended feature using the AIN#_EF_CONFIG registers

Name	Start Address	Type	Access
AIN#(0:13)_EF_CONFIG_A	9300	UINT32	R/W
AIN#(0:13)_EF_CONFIG_B	9600	UINT32	R/W
AIN#(0:13)_EF_CONFIG_C	9900	UINT32	R/W
AIN#(0:13)_EF_CONFIG_D	10200	FLOAT32	R/W
AIN#(0:13)_EF_CONFIG_E	10500	FLOAT32	R/W
AIN#(0:13)_EF_CONFIG_F	10800	FLOAT32	R/W
AIN#(0:13)_EF_CONFIG_G	11100	FLOAT32	R/W

AIN#(0:13)_EF_CONFIG_A - Starting Address: 9300

Function dependent on selected feature index.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names	Addresses
AIN0_EF_CONFIG_A, AIN1_EF_CONFIG_A, AIN2_EF_CONFIG_A, AIN3_EF_CONFIG_A, AIN4_EF_CONFIG_A, AIN5_EF_CONFIG_A, AIN6_EF_CONFIG_A, AIN7_EF_CONFIG_A, AIN8_EF_CONFIG_A, AIN9_EF_CONFIG_A, AIN10_EF_CONFIG_A, AIN11_EF_CONFIG_A, AIN12_EF_CONFIG_A, AIN13_EF_CONFIG_A Show All	9300, 9302, 9304, 9306, 9308, 9310, 9312, 9314, 9316, 9318, 9320, 9322, 9324, 9326 Show All

AIN#(0:13)_EF_CONFIG_B - Starting Address: 9600

Function dependent on selected feature index.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names	Addresses
AIN0_EF_CONFIG_B, AIN1_EF_CONFIG_B, AIN2_EF_CONFIG_B, AIN3_EF_CONFIG_B, AIN4_EF_CONFIG_B, AIN5_EF_CONFIG_B, AIN6_EF_CONFIG_B, AIN7_EF_CONFIG_B, AIN8_EF_CONFIG_B, AIN9_EF_CONFIG_B, AIN10_EF_CONFIG_B, AIN11_EF_CONFIG_B, AIN12_EF_CONFIG_B, AIN13_EF_CONFIG_B Show All	9600, 9602, 9604, 9606, 9608, 9610, 9612, 9614, 9616, 9618, 9620, 9622, 9624, 9626 Show All

AIN#(0:13)_EF_CONFIG_C - Starting Address: 9900

Function dependent on selected feature index.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names	Addresses
AIN0_EF_CONFIG_C, AIN1_EF_CONFIG_C, AIN2_EF_CONFIG_C, AIN3_EF_CONFIG_C, AIN4_EF_CONFIG_C, AIN5_EF_CONFIG_C, AIN6_EF_CONFIG_C, AIN7_EF_CONFIG_C, AIN8_EF_CONFIG_C, AIN9_EF_CONFIG_C, AIN10_EF_CONFIG_C, AIN11_EF_CONFIG_C, AIN12_EF_CONFIG_C, AIN13_EF_CONFIG_C Show All	9900, 9902, 9904, 9906, 9908, 9910, 9912, 9914, 9916, 9918, 9920, 9922, 9924, 9926 Show All

AIN#(0:13)_EF_CONFIG_D - Starting Address: 10200

Function dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names	Addresses
AIN0_EF_CONFIG_D, AIN1_EF_CONFIG_D, AIN2_EF_CONFIG_D, AIN3_EF_CONFIG_D, AIN4_EF_CONFIG_D, AIN5_EF_CONFIG_D, AIN6_EF_CONFIG_D, AIN7_EF_CONFIG_D, AIN8_EF_CONFIG_D, AIN9_EF_CONFIG_D, AIN10_EF_CONFIG_D, AIN11_EF_CONFIG_D, AIN12_EF_CONFIG_D, AIN13_EF_CONFIG_D Show All	10200, 10202, 10204, 10206, 10208, 10210, 10212, 10214, 10216, 10218, 10220, 10222, 10224, 10226 Show All
AIN#(0:13)_EF_CONFIG_E - Starting Address: 10500 Function dependent on selected feature index. <ul style="list-style-type: none"> Data type: FLOAT32 (type index = 3) Readable and writable Default value: 0 T7: <ul style="list-style-type: none"> Minimum firmware version: 1.0030 	
Expanded Names	Addresses
AIN0_EF_CONFIG_E, AIN1_EF_CONFIG_E, AIN2_EF_CONFIG_E, AIN3_EF_CONFIG_E, AIN4_EF_CONFIG_E, AIN5_EF_CONFIG_E, AIN6_EF_CONFIG_E, AIN7_EF_CONFIG_E, AIN8_EF_CONFIG_E, AIN9_EF_CONFIG_E, AIN10_EF_CONFIG_E, AIN11_EF_CONFIG_E, AIN12_EF_CONFIG_E, AIN13_EF_CONFIG_E Show All	10500, 10502, 10504, 10506, 10508, 10510, 10512, 10514, 10516, 10518, 10520, 10522, 10524, 10526 Show All
AIN#(0:13)_EF_CONFIG_F - Starting Address: 10800 Function dependent on selected feature index. <ul style="list-style-type: none"> Data type: FLOAT32 (type index = 3) Readable and writable Default value: 0 T7: <ul style="list-style-type: none"> Minimum firmware version: 1.0030 	
Expanded Names	Addresses
AIN0_EF_CONFIG_F, AIN1_EF_CONFIG_F, AIN2_EF_CONFIG_F, AIN3_EF_CONFIG_F, AIN4_EF_CONFIG_F, AIN5_EF_CONFIG_F, AIN6_EF_CONFIG_F, AIN7_EF_CONFIG_F, AIN8_EF_CONFIG_F, AIN9_EF_CONFIG_F, AIN10_EF_CONFIG_F, AIN11_EF_CONFIG_F, AIN12_EF_CONFIG_F, AIN13_EF_CONFIG_F Show All	10800, 10802, 10804, 10806, 10808, 10810, 10812, 10814, 10816, 10818, 10820, 10822, 10824, 10826 Show All
AIN#(0:13)_EF_CONFIG_G - Starting Address: 11100 Function dependent on selected feature index. <ul style="list-style-type: none"> Data type: FLOAT32 (type index = 3) Readable and writable Default value: 0 T7: <ul style="list-style-type: none"> Minimum firmware version: 1.0030 	
Expanded Names	Addresses
AIN0_EF_CONFIG_G, AIN1_EF_CONFIG_G, AIN2_EF_CONFIG_G, AIN3_EF_CONFIG_G, AIN4_EF_CONFIG_G, AIN5_EF_CONFIG_G, AIN6_EF_CONFIG_G, AIN7_EF_CONFIG_G, AIN8_EF_CONFIG_G, AIN9_EF_CONFIG_G, AIN10_EF_CONFIG_G, AIN11_EF_CONFIG_G, AIN12_EF_CONFIG_G, AIN13_EF_CONFIG_G Show All	11100, 11102, 11104, 11106, 11108, 11110, 11112, 11114, 11116, 11118, 11120, 11122, 11124, 11126 Show All

Each AIN-EF index requires different configuration parameters, so the meaning of the AIN#_EF_CONFIG registers depend on which AIN#_EF_INDEX is set.

3. Configure normal AIN configurations through the normal AIN registers

Analog input range, resolution, settling, and negative channel settings are configured through the normal [AIN registers](#).

4. Read from AIN#_EF_READ_A to perform the extended feature operation

Name	Start Address	Type	Access
AIN#(0:13)_EF_READ_A	7000	FLOAT32	R
AIN#(0:13)_EF_READ_A - Starting Address: 7000 Function dependent on selected feature index. <ul style="list-style-type: none"> Data type: FLOAT32 (type index = 3) Read-only Default value: 0 This register uses system RAM. The maximum RAM is 64KB. For more information, see 4.4 RAM T7: <ul style="list-style-type: none"> Minimum firmware version: 1.0030 			
Expanded Names	Addresses		
AIN0_EF_READ_A, AIN1_EF_READ_A, AIN2_EF_READ_A, AIN3_EF_READ_A, AIN4_EF_READ_A, AIN5_EF_READ_A, AIN6_EF_READ_A, AIN7_EF_READ_A, AIN8_EF_READ_A, AIN9_EF_READ_A, AIN10_EF_READ_A, AIN11_EF_READ_A, AIN12_EF_READ_A, AIN13_EF_READ_A Show All	7000, 7002, 7004, 7006, 7008, 7010, 7012, 7014, 7016, 7018, 7020, 7022, 7024, 7026 Show All		

Only reading AIN#_EF_READ_A will trigger the selected AIN-EF operation. The AIN#_EF_READ_A result is returned. Additional results are saved for later retrieval.

If the AIN-EF index uses stream-burst, reading AIN#_EF_READ_A will block for the length of time it takes to collect the necessary samples.

5. Read additional results B, C, and D

Name	Start Address	Type	Access
AIN#(0:13)_EF_READ_B	7300	FLOAT32	R/W
AIN#(0:13)_EF_READ_C	7600	FLOAT32	R/W
AIN#(0:13)_EF_READ_D	7900	FLOAT32	R
AIN#(0:13)_EF_READ_B - Starting Address: 7300			

Function dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names

AIN0_EF_READ_B, AIN1_EF_READ_B, AIN2_EF_READ_B, AIN3_EF_READ_B, AIN4_EF_READ_B, AIN5_EF_READ_B, AIN6_EF_READ_B, AIN7_EF_READ_B, AIN8_EF_READ_B, AIN9_EF_READ_B, AIN10_EF_READ_B, AIN11_EF_READ_B, AIN12_EF_READ_B, AIN13_EF_READ_B [Show All](#)

Addresses

7300, 7302, 7304, 7306, 7308, 7310, 7312, 7314, 7316, 7318, 7320, 7322, 7324, 7326 [Show All](#)

AIN#(0:13)_EF_READ_C - Starting Address: 7600

Function dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names

AIN0_EF_READ_C, AIN1_EF_READ_C, AIN2_EF_READ_C, AIN3_EF_READ_C, AIN4_EF_READ_C, AIN5_EF_READ_C, AIN6_EF_READ_C, AIN7_EF_READ_C, AIN8_EF_READ_C, AIN9_EF_READ_C, AIN10_EF_READ_C, AIN11_EF_READ_C, AIN12_EF_READ_C, AIN13_EF_READ_C [Show All](#)

Addresses

7600, 7602, 7604, 7606, 7608, 7610, 7612, 7614, 7616, 7618, 7620, 7622, 7624, 7626 [Show All](#)

AIN#(0:13)_EF_READ_D - Starting Address: 7900

Function dependent on selected feature index.

- Data type: FLOAT32 (type index = 3)
- Read-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0030

Expanded Names

AIN0_EF_READ_D, AIN1_EF_READ_D, AIN2_EF_READ_D, AIN3_EF_READ_D, AIN4_EF_READ_D, AIN5_EF_READ_D, AIN6_EF_READ_D, AIN7_EF_READ_D, AIN8_EF_READ_D, AIN9_EF_READ_D, AIN10_EF_READ_D, AIN11_EF_READ_D, AIN12_EF_READ_D, AIN13_EF_READ_D [Show All](#)

Addresses

7900, 7902, 7904, 7906, 7908, 7910, 7912, 7914, 7916, 7918, 7920, 7922, 7924, 7926 [Show All](#)

Reading from result registers other than AIN#_EF_READ_A will read the saved values and will not initiate a new reading.

14.1.0.1 Excitation Circuits [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN-EF indices that measure sensors usually need an excitation circuit. The excitation circuit provides the conditions that the sensor needs to operate and controls how the output from the sensor is measured.

Individual AIN-EF indices may only support a subset of the circuits listed here.

Three types of excitation circuits are based on a current source and three are based on a voltage divider.

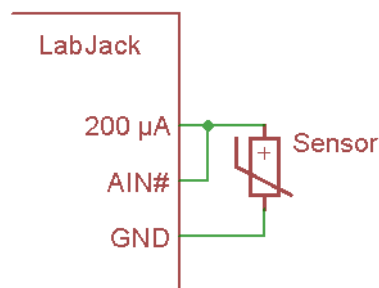
For AIN-EF indices that require excitation circuits, the circuit indices below are written to AIN#_EF_CONFIG_B.

Current Source Excitation Circuits

Current source circuits are the simplest of the supported excitation circuits.

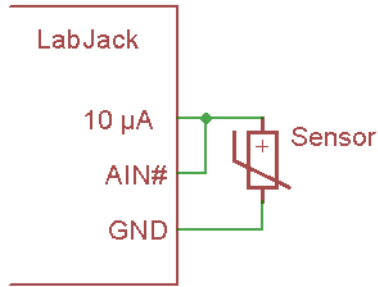
Circuit 0 – 200 μ A Current Source - T7 Only:

This excitation circuit uses the [200 \$\mu\$ A current source](#) available on the T7 to power a sensor. The current source is not exactly 200 μ A, so the value measured during calibration will be used for calculations. Current source excitation circuits are useful for resistance based sensors like RTDs or bridge circuits.



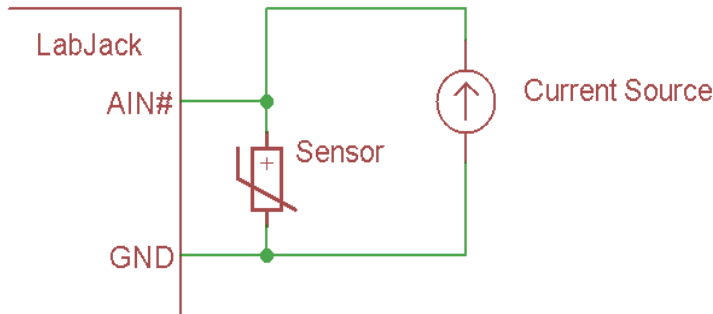
Circuit 1 – 10 μ A Current Source - T7 Only:

This excitation circuit uses the 10 μ A current source available on the T7 to power a sensor. The current source is not exactly 10 μ A, so the value measured during calibration will be used for calculations. Current source excitation circuits are useful for resistance based sensors like RTDs or bridge circuits.



Circuit 2 – Custom Current Source:

This excitation circuit uses a current source external to the LabJack. The current provided by the source is specified during configuration of the AIN#_EF.



Required configuration registers:

- AIN#_EF_CONFIG_D - Excitation Amps

Voltage Divider Excitation Circuits

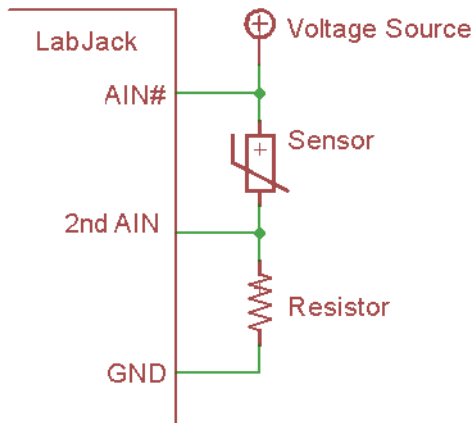
Voltage divider circuits rely on combinations of supplied voltage and a fixed resistor in series with the sensor.

Circuit 3 – Shunt Resistor:

This excitation circuit uses a resistor in series with a sensor. When using this circuit, the LabJack will first measure the voltage across the sensor, then the voltage across the shunt resistor. The second measurement uses a second analog input, which is specified in the AIN#_EF configuration. The shunt resistance is also specified during configuration. After measuring both voltages, the current through and the resistance of the sensor can be calculated.

T7 only: Differential readings can reduce noise. For more information, see the Differential Readings note below.

This circuit attempts to compensate for power supply noise.

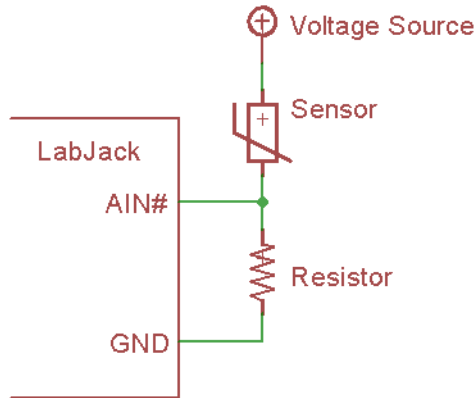


Required configuration registers:

- AIN#_EF_CONFIG_C - Excitation AIN Channel Number
- AIN#_EF_CONFIG_E - Excitation Ohms

Circuit 4 – Known Voltage Source:

This excitation circuit uses a voltage source and a shunt resistor. Values for the output of the voltage source and the resistor must be provided during AIN#_EF configuration. When using this circuit, the LabJack will measure the voltage between the sensor and the resistor, then calculate the resistance of the sensor.



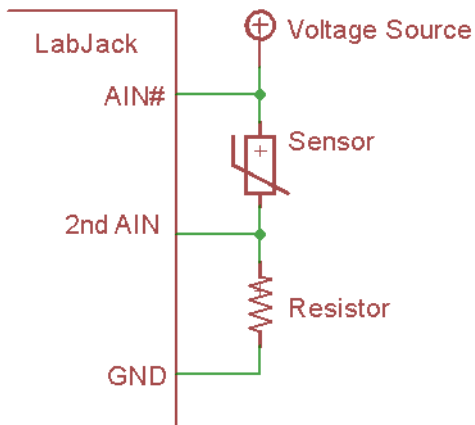
Required configuration registers:

- AIN#_EF_CONFIG_D - Excitation Volts
- AIN#_EF_CONFIG_E - Excitation Ohms

Circuit 5 – Voltage Source:

This excitation circuit is set up the same as circuit 4, but it is measured differently. Only the value of the shunt resistor needs to be specified. When using this circuit, the LabJack will measure the voltage across the shunt resistor, then measure the voltage across the sensor. Resistance of the sensor can then be computed. The voltage source should be low-noise.

T7 only: If the analog input is set to differential, then multiple sensors can be used in series—for more information, see the Differential Readings note below.



Required configuration registers:

- AIN#_EF_CONFIG_C - Excitation AIN Channel Number
- AIN#_EF_CONFIG_E - Excitation Ohms

Differential Readings - T7 Only

Multiple sensors can be connected in series. When using a series of sensors, both sides of the sensor need to be connected to AINs and those AINs need to be set to differential. Care must also be taken to ensure that the total voltage drop across the sensors can not exceed the current source's maximum output.

For circuit types 3 and 5 above, differential readings can reduce noise. To configure, the AIN-EF channel must be set as the positive AIN channel and the negative AIN channel must then be used as the second AIN (which is used to measure the voltage across the shunt resistor).

See [14.0 AIN](#) for more on differential readings.

14.1.1 Thermocouple (T7 Only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview - T7 Only

This feature is only supported on the T7.

AIN#_EF_INDEX values:

- 20:** Thermocouple type E
- 21:** Thermocouple type J
- 22:** Thermocouple type K
- 23:** Thermocouple type R
- 24:** Thermocouple type T
- 25:** Thermocouple type S
- 30:** Thermocouple type C

This Thermocouple Extended Feature automatically performs calculations for the thermocouple types listed above.

Thermocouple AIN-EF indices read two analog inputs—one AIN connected to a thermocouple and a second AIN connected to a Cold-Junction Compensation (CJC) sensor.

For more information, see the [Thermocouples Application Note](#) and the [Thermocouples with the T7 App Note](#).

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Options: Select temperature units for AIN#_EF_READ_A and AIN#_EF_READ_C:

- 0 = K
- 1 = °C
- 2 = °F

AIN#_EF_CONFIG_B - CJC Modbus address: The Modbus address of the second AIN channel that will be read to acquire the CJC reading

The default is the on-board temperature sensor (TEMPERATURE_DEVICE_K, at address 60052).

AIN#_EF_CONFIG_D - CJC Slope: A slope to be applied to the CJC reading

This value is always in units of K/volt, regardless of AIN#_EF_CONFIG_A, so for the internal sensor it will nominally be 1.00 and for an LM34 it will be 55.56.

Default is 1.0.

AIN#_EF_CONFIG_E - CJC Offset: An offset to be applied to the CJC reading

This value is always in units of K, regardless of AIN#_EF_CONFIG_A, so for the internal sensor it will nominally be 0.0 and for an LM34 it will be 255.37.

Default is 0.0.

Remarks

The normal [analog input settings](#) are used for negative channel, resolution index, settling, and range.

Differential thermocouple readings have the advantage of mitigating bad ground loops and ground offset problems. See the [Thermocouples App Note](#) for more information.

The ± 0.1 volt range is automatically used if the range of the applicable channel (AIN#_RANGE) is set to the default ± 10 volts. Otherwise, the specified range will be used.

CJC calculations are always done in kelvin, regardless of whether AIN#_EF_CONFIG_A is used to change the output units. AIN#_EF_CONFIG_D and AIN#_EF_CONFIG_E should be used as needed to convert the CJC sensor reading to kelvin.

Results

Results are read from the following registers.

AIN#_EF_READ_A: Final calculated temperature of the remote end of the thermocouple

AIN#_EF_READ_B: Measured thermocouple voltage.

AIN#_EF_READ_C: CJC temperature.

AIN#_EF_READ_D: Thermocouple voltage calculated for CJC temperature.

Only reading AIN#_EF_READ_A triggers a new measurement.

Example

Assume a type K thermocouple is connected to AIN3/GND. To configure:

```
AIN3_EF_INDEX = 22 // feature index for type K thermocouple
AIN3_EF_CONFIG_B = 60052 // Set the CJC source (the address for device
// temperature sensor). 60052 is TEMPERATURE_DEVICE_K
AIN3_EF_CONFIG_D = 1.0 // slope for CJC reading
AIN3_EF_CONFIG_E = 0.0 // offset for CJC reading
```

Read AIN3_EF_READ_A to get the calculated temperature. If the remote end is at room temperature, it will read as approximately 298 kelvin.

For a more detailed walkthrough, see [Configuring & Reading a Thermocouple](#).

14.1.2 Offset and Slope [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN#_EF_INDEX: 1

This Offset and Slope Extended Feature automatically adds a slope and an offset to analog readings.

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_D - Slope: Custom slope to be applied to the analog voltage reading. Default is 1.00.

AIN#_EF_CONFIG_E - Offset: Custom offset to be applied to the analog voltage reading. Default is 0.00.

Remarks

The normal [analog input settings](#) are used for negative channel, resolution index, settling, and range.

Results

For results, read AIN#_EF_READ_A.

AIN#_EF_READ_A - Returns the calculated voltage:

```
measured volts * slope + offset
```

Only reading AIN#_EF_READ_A triggers a new measurement.

Example

To configure Offset and Slope AIN-EF for AIN3:

```
AIN3_EF_INDEX = 1 // feature index for Offset and Slope
AIN3_EF_CONFIG_D = 2.0 // slope
AIN3_EF_CONFIG_E = -0.5 // offset
```

Now each read of AIN3_EF_READ_A will return `(AIN3 volts * 2.0) - 0.5`.

14.1.3 RTD [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN#_EF_INDEX values:

40: PT100

41: PT500

42: PT1000

This RTD Extended Feature automatically performs calculations for a Resistance Temperature Detector (RTD). RTD types are listed above.

When AIN#_EF_READ_A is read, the T-series device reads an analog input and calculates the resistance of the RTD. Temperature is then calculated using the rational polynomial technique.

An RTD (aka PT100, PT1000) is a type of temperature sensor. See the [Temperature Sensors App Note](#).

An RTD provides a varying resistance, but the LabJack measures voltage, so some sort of circuit must be used to convert the varying resistance to a varying voltage. This AIN-EF supports various excitation circuits, the best and easiest is usually the [LJTick-Resistance](#), which would be [excitation circuit #4](#).

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Options: Selects temperature units:

- 0 = K
- 1 = °C
- 2 = °F

AIN#_EF_CONFIG_B - Excitation Circuit Index: Specifies which excitation circuit is used.

See [14.1.0.1 Excitation Circuits](#) for circuit options.

AIN#_EF_CONFIG_C - Excitation AIN Channel Number: AIN channel number of the AIN used to measure the RTD's excitation.

AIN#_EF_CONFIG_D - Excitation Volts or Amps: The number of volts or amps provided as the excitation circuit voltage source or current source.

AIN#_EF_CONFIG_E - Excitation Ohms: The resistance in ohms of the shunt resistor.

Remarks

The normal [analog input settings](#) are used for negative channel, resolution index, settling, and range.

Result

For results, read the following registers.

AIN#_EF_READ_A: Final calculated temperature.

AIN#_EF_READ_B: Resistance of the RTD.

AIN#_EF_READ_C: Voltage across the RTD.

AIN#_EF_READ_D: Current through the RTD.

Only reading AIN#_EF_READ_A triggers a new measurement.

Example

Shunt resistance, circuit #3:

To configure AIN0 to use the shunt resistor circuit with shunt resistor value = 2700 Ω and RTD type 100:

```
AIN0_EF_INDEX = 40 // Set AIN_EF0 to RTD100.  
AIN0_EF_CONFIG_A = 0 // Set result units to kelvin.  
AIN0_EF_CONFIG_B = 3 // Set excitation circuit to 3.  
AIN0_EF_CONFIG_C = 1 // Set the second AIN to AIN1.  
AIN0_EF_CONFIG_E = 2700 // Set the shunt resistance.
```

Now each read of AIN0_EF_READ_A will measure the voltages on AIN0 and AIN1 and return the calculated temperature.

14.1.4 RMS [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN#_EF_INDEX values:

10: RMS Flex

11: RMS Auto

This RMS Extended Feature calculates the true root mean square voltage of a signal connected to an analog input. It also calculates peak-to-peak voltage, DC offset (average voltage), and period (1/frequency) of the analog waveform.

RMS Flex: RMS Flex uses registers to specify the number of samples to collect and the frequency at which to collect those samples. The LabJack will then use the entire data set to calculate RMS.

RMS Auto: RMS Auto will collect samples for more than one period and attempt to find a period within the data set. The LabJack will then compute RMS for only the detected period's period.

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Number of Samples: The number of samples to be acquired.

- Default: 200
- Max: 16384

AIN#_EF_CONFIG_B - Hysteresis (RMS Auto only): The smallest step (analog voltage in binary) that will trigger period detection when using RMS Auto. Larger values will better reject noise. Smaller values can increase the accuracy of the period calculation.

- Default: 100 (16-bit counts)
- This value is meaningless for RMS Flex

AIN#_EF_CONFIG_D - Scan Rate: The frequency at which samples will be collected.

- Default: 6000

Sample Time

The maximum possible sample time is 180 ms.

Sample time is the number of samples divided by the sample frequency. For example, the period is 16.7 ms when using a scan rate of 6000 Hz and 100 samples:

$$100 \text{ samples} / 6000 \text{ samples per second} = 16.7 \text{ ms}$$

RMS Flex Considerations

RMS Flex requires the sample time to be an even multiple of the period to be measured.

RMS Auto Considerations

RMS Auto requires the sample time to be set so that 1.5 to 4 periods will be observed.

The hysteresis value also needs to be set to control the sensitivity of period detection.

Accuracy

Stream-burst is used to acquire the specified number of samples at hardware timed intervals. The true RMS is then calculated using the following piece-wise math:

$$(\sum(\text{each sample}^2)) / \text{number of samples}) ^{0.5}$$

The accuracy of this calculation depends on the number of samples per cycle. Smooth waveforms will not require many samples per cycle, but waveforms with lots of harmonic content will require more samples per cycle.

Stream Configuration

This extended feature internally uses Stream-Burst to acquire the data set, so stream AIN configurations apply.

Results

For results, read the following registers.

AIN#_EF_READ_A: RMS Voltage

AIN#_EF_READ_B: Peak-to-Peak Voltage

AIN#_EF_READ_C: DC Offset Voltage (Average)

AIN#_EF_READ_D: Period (Seconds)

Only reading AIN#_EF_READ_A triggers a new measurement. Because multiple measurements are taken, a read of AIN#_EF_READ_A blocks for the length of the sample time.

14.1.5 Thermistor [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN#_EF_INDEX values:

50: calculate temperature using the Steinhart-Hart equation

51: calculate temperature using the beta equation

This Thermistor Extended Feature automatically performs the necessary calculations for thermistors using the Steinhart-Hart equation or the beta equation.

Steinhart-Hart vs. beta: The beta function works well over a limited range of about 50 °C. Typical error is $\sim\pm 0.5$ °C. The Steinhart-Hart is usually more accurate than ± 0.01 °C across a larger range.

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Thermistor Options: Selects temperature units:

- 0 = K
- 1 = °C
- 2 = °F

AIN#_EF_CONFIG_B - Excitation Circuit Index: The index of the voltage divider excitation circuit to be used

See [14.1.0.1 Excitation Circuits](#) for circuit indices.

AIN#_EF_CONFIG_C - Excitation AIN Channel Number: The AIN channel number of the AIN line used to measure the RTD's excitation.

Ignored if the Excitation Circuit Index does not use a second AIN.

AIN#_EF_CONFIG_D - Excitation Detail Volts: The number of volts provided as the excitation circuit voltage source.

AIN#_EF_CONFIG_E - Excitation Detail Ohms: The resistance in ohms of the shunt resistor.

AIN#_EF_CONFIG_F - R25 Ohms: The nominal resistance in ohms of the thermistor at 25 °C.

Steinhart-Hart Beta

AIN#_EF_CONFIG_G A constant	β
AIN#_EF_CONFIG_H B constant	°C at which β was calculated
AIN#_EF_CONFIG_I C constant	No meaning
AIN#_EF_CONFIG_J D constant	No meaning

The G, H, I, and J config registers have different meaning for Steinhart-Hart and beta. Steinhart-Hart constants are normally provided in the thermistor's datasheet or obtained from the manufacturer.

Remarks

The normal [analog input settings](#) are used for negative channel, resolution index, settling, and range.

T7 only: If [voltage will stay below 1.0V](#), use the 1.0V range for improved resolution and accuracy.

Results

For results, read the following registers.

AIN#_EF_READ_A: Thermistor Temperature
AIN#_EF_READ_B: Thermistor Resistance
AIN#_EF_READ_C: Thermistor Voltage

Only reading AIN#_EF_READ_A triggers a new measurement.

Example

This example configures a LabJack to read from a [Vishay NTCLE100E3103 10k Thermistor](#) using a LabJack [LJTick-Resistance](#) to complete the excitation circuit. The LJTick-Resistance is connected to the AIN0/1 terminal block. The thermistor is connected between the Vref and INA terminals on the LJTick-Resistance.

```
AIN0_EF_INDEX = 50      -- Steinhart-Hart
AIN0_EF_CONFIG_A = 1   -- Output degrees Celsius.
AIN0_EF_CONFIG_B = 4   -- Excitation circuit #4.
AIN0_EF_CONFIG_C = 0   -- Second AIN, not used for excitation circuit #4.
AIN0_EF_CONFIG_D = 2.5 -- 2.5 V provided by the LJTick-Resistance
AIN0_EF_CONFIG_E = 10000 -- 10 kΩ shunt resistor provided by the LJTick-Resistance-10k.
AIN0_EF_CONFIG_F = 10000 -- R25 The nominal resistance of the thermistor at 25 °C.
AIN0_EF_CONFIG_G = 0.003354016 -- Constants from the thermistor's datasheet.
AIN0_EF_CONFIG_H = 0.000256985
AIN0_EF_CONFIG_I = 0.000002620
AIN0_EF_CONFIG_J = 0.0000006383
```

Results:

```
AIN0_EF_READ_A = 23.19 -- Temperature of the thermistor. (°C)
AIN0_EF_READ_B = 10829.4 -- Calculated resistance. (Ω)
AIN0_EF_READ_C = 1.299774 -- Voltage across the thermistor. (V)
```

Overview

AIN#_EF_INDEX values:

- 50: calculate temperature using the Steinhart-Hart equation
- 51: calculate temperature using the beta equation

This Thermistor Extended Feature automatically performs the necessary calculations for thermistors using the Steinhart-Hart equation or the beta equation.

Steinhart-Hart vs. beta: The beta function works well over a limited range of about 50 °C. Typical error is $\sim\pm 0.5$ °C. The Steinhart-Hart is usually more accurate than ± 0.01 °C across a larger range.

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Thermistor Options: Selects temperature units:

- 0 = K
- 1 = °C
- 2 = °F

AIN#_EF_CONFIG_B - Excitation Circuit Index: The index of the voltage divider excitation circuit to be used

See [14.1.0.1 Excitation Circuits](#) for circuit indices.

AIN#_EF_CONFIG_C - Excitation AIN Channel Number: The AIN channel number of the AIN line used to measure the RTD's excitation.

Ignored if the Excitation Circuit Index does not use a second AIN.

AIN#_EF_CONFIG_D - Excitation Detail Volts: The number of volts provided as the excitation circuit voltage source.

AIN#_EF_CONFIG_E - Excitation Detail Ohms: The resistance in ohms of the shunt resistor.

AIN#_EF_CONFIG_F - R25 Ohms: The nominal resistance in ohms of the thermistor at 25 °C.

Steinhart-Hart Beta

AIN#_EF_CONFIG_G A constant	β
AIN#_EF_CONFIG_H B constant	°C at which β was calculated
AIN#_EF_CONFIG_I C constant	No meaning
AIN#_EF_CONFIG_J D constant	No meaning

The G, H, I, and J config registers have different meaning for Steinhart-Hart and beta. Steinhart-Hart constants are normally provided in the thermistor's datasheet or obtained from the manufacturer.

Remarks

The normal [analog input settings](#) are used for negative channel, resolution index, settling, and range.

T7 only: If voltage will stay below 1.0V, use the 1.0V range for improved resolution and accuracy.

Results

For results, read the following registers.

AIN#_EF_READ_A: Thermistor Temperature
 AIN#_EF_READ_B: Thermistor Resistance
 AIN#_EF_READ_C: Thermistor Voltage

Only reading AIN#_EF_READ_A triggers a new measurement.

Example

This example configures a LabJack to read from a [Vishay NTCLE100E3103 10k Thermistor](#) using a LabJack [LJTick-Resistance](#) to complete the excitation circuit. The LJTick-Resistance is connected to the AIN0/1 terminal block. The thermistor is connected between the Vref and INA terminals on the LJTick-Resistance.

```
AIN0_EF_INDEX = 50      -- Steinhart-Hart
AIN0_EF_CONFIG_A = 1   -- Output degrees Celsius.
AIN0_EF_CONFIG_B = 4   -- Excitation circuit #4.
AIN0_EF_CONFIG_C = 0   -- Second AIN, not used for excitation circuit #4.
AIN0_EF_CONFIG_D = 2.5 -- 2.5 V provided by the LJTick-Resistance
AIN0_EF_CONFIG_E = 10000 -- 10 kΩ shunt resistor provided by the LJTick-Resistance-10k.
AIN0_EF_CONFIG_F = 10000 -- R25 The nominal resistance of the thermistor at 25 °C.
AIN0_EF_CONFIG_G = 0.003354016 -- Constants from the thermistor's datasheet.
AIN0_EF_CONFIG_H = 0.000256985
AIN0_EF_CONFIG_I = 0.000002620
AIN0_EF_CONFIG_J = 0.0000006383
```

Results:

AIN0_EF_READ_A = 23.19 -- Temperature of the thermistor. (°C)
AIN0_EF_READ_B = 10829.4 -- Calculated resistance. (Ω)
AIN0_EF_READ_C = 1.299774 -- Voltage across the thermistor. (V)

14.1.6 Resistance [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN#_EF_INDEX: 4

This Resistance Extended Feature will calculate the resistance of a given circuit element.

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_B - Excitation Circuit Index: The index of the voltage divider excitation circuit to be used.

See [14.1.0.1 Excitation Circuits](#) for circuit indices.

AIN#_EF_CONFIG_C - Excitation AIN Channel Number: The AIN channel number of the AIN line used to measure the RTD's excitation.

Ignored if the Excitation Circuit Index does not use a second AIN.

AIN#_EF_CONFIG_D - Excitation Detail Volts: The number of volts provided as the excitation circuit voltage source.

AIN#_EF_CONFIG_E - Excitation Detail Ohms: The resistance in ohms of the shunt resistor.

Remarks

The normal [analog input settings](#) are used for negative channel, resolution index, settling, and range.

Results

For results, read the following registers.

AIN#_EF_READ_A: Resistance

AIN#_EF_READ_B: Voltage

AIN#_EF_READ_C: Current

Only reading AIN#_EF_READ_A triggers a new measurement.

Example

An unknown resistor to be measured (a 10 kΩ resistor is used in this test) is connected in series with a 200 μA current source.

AIN0_EF_INDEX = 4
AIN0_EF_CONFIG_B = 0 -- Excitation circuit 0 (200 μA current source)

Reading results:

- AIN0_EF_READ_A returns 10089.7 Ω
- AIN0_EF_READ_B returns 2.012 V
- AIN0_EF_READ_C returns 199 μA

14.1.7 Average Min Max [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN#_EF_INDEX: 3

This Average Mix Max Extended Feature will sample an analog input a specified number of times at a specified rate, then calculate the average, min, and max voltage.

If you are using this feature to reduce noise by oversampling-and-averaging, consider maximizing resolution first, since that is the best and fastest way to reduce noise.

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Number of Samples: The number of samples to be acquired.

- Default: 200
- Max: 16384

AIN#_EF_CONFIG_D - Scan Rate: The frequency at which samples will be collected.

- Default: 6000

Sample Time

The maximum possible sample time is 180 ms.

Sample time is the number of samples divided by the sample frequency. For example, the period is 16.7 ms when using a scan rate of 6000 Hz and 100 samples:

```
100 samples / 6000 samples per second = 16.7 ms
```

If the signal has a known periodic component, then setting the sample time to an even multiple of the period will generally improve results.

Stream Configuration

This extended feature internally uses Stream-Burst to acquire the data set, so stream AIN configurations apply.

Results

For results, read the following registers.

AIN#_EF_READ_A: Average volts

AIN#_EF_READ_B: Max volts

AIN#_EF_READ_C: Min volts

Only reading AIN#_EF_READ_A triggers a new measurement. Because multiple measurements are taken, a read of AIN#_EF_READ_A blocks for the length of the sample time.

Example

To measure a 10 Hz sine wave with amplitude 0.1 V and DC offset of 1.2 V, set the number of samples to 100 and the scan rate to 1000. The total acquisition time is 100 ms, which is an even multiple of the signal's period.

```
AIN0_EF_INDEX = 3
AIN0_EF_CONFIG_A = 100 -- Number of samples
AIN0_EF_CONFIG_D = 1000 -- Scan rate
```

Results, with noise levels less than 600 μ V:

```
AIN0_EF_READ_A = 1.201 V -- Average volts
AIN0_EF_READ_B = 1.301 V -- Max volts
AIN0_EF_READ_C = 1.101 V -- Min volts
```

We can change the sample time to emphasize the benefit of matching to the period. If we change AIN0_EF_CONFIG_A to 70 (an odd multiple of the signal's period), then the results are still centered on 1.2, 1.3, and 1.1 V—but noise levels are ± 5 mV.

14.1.8 Average and Threshold [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

AIN#_EF_INDEX: 5

This Average and Threshold Extended Feature will read an input a specified number of times at a specified rate, then average the readings and set a flag if the average is above a specified threshold. Can be used with analog inputs or digital inputs.

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Number of Samples: Number of samples to be acquired.

- Default: 200
- Max: 16384

AIN#_EF_CONFIG_B - Digital Override: Selects whether this AIN-EF takes samples from the default analog line or from a specified digital line.

- When set to zero, the normal AIN# will be used.
 - This is the default.
 - For example, if AIN2_EF_CONFIG_B is 0, samples are collected from AIN2 when AIN2_EF_READ_A is read.
- When set to non-zero, sets the Modbus address of a digital IO.
 - For example, if AIN2_EF_CONFIG_B is 2001 (the address of FIO1), samples are collected from FIO1 when AIN2_EF_READ_A is read.

AIN#_EF_CONFIG_D - Scan Rate: The frequency at which samples will be collected.

- Default: 6000

AIN#_EF_CONFIG_E - Threshold: If the computed average is above this value, then the threshold flag (AIN#_EF_READ_A) will be set.

Sample time

The maximum possible sample time is 180 ms.

Sample time is the number of samples divided by the sample frequency. For example, the period is 16.7 ms when using a scan rate of 6000 Hz and 100 samples:

```
100 samples / 6000 samples per second = 16.7 ms
```

If the signal has a known periodic component, then setting the sample time to an even multiple of the period will generally improve results.

Stream Configuration

This extended feature internally uses Stream-Burst to acquire the data set, so stream AIN configurations apply.

Results

For results, read the following registers.

AIN#_EF_READ_A - Threshold Flag: Returns 1.0 if the average is greater than the threshold, 0.0 if not.

AIN#_EF_READ_B - Average: The average of the collected samples.

Only reading AIN#_EF_READ_A triggers a new measurement. Because multiple measurements are taken, a read of AIN#_EF_READ_A blocks for the length of the sample time.

Example - Analog Input

A 10 Hz sine wave with magnitude 0.1 V and DC offset of 1.2 V is connected to AIN2. A threshold value less than the expected 1.2 V average will produce a positive result.

```
AIN2_EF_INDEX = 5
AIN2_EF_CONFIG_A = 100 -- 100 samples
AIN2_EF_CONFIG_B = 0 -- Use AIN2 to collect samples
AIN2_EF_CONFIG_D = 1000 -- Scan rate
AIN2_EF_CONFIG_E = 1.15 -- Threshold
```

Reading register AIN2_EF_READ_A returns 1.0, indicating that the average is greater than the threshold. If, instead, the threshold written to AIN2_EF_CONFIG_E was 1.25, then AIN2_EF_READ_A would then return 0.0 to indicate that the threshold was greater than the average.

Example - Digital Input

A 10 Hz sine wave with magnitude 1.2 V and DC offset of 1.6 V is connected to FIO0. The input signal should be interpreted as high two out of three times. In this case, the threshold of 0.65 is a ratio. If the average of the readings is 0.65 or greater, the AIN2_EF_READ_A register should be 1.0.

```
AIN2_EF_INDEX = 5
AIN2_EF_CONFIG_A = 100 -- 100 samples
AIN2_EF_CONFIG_B = 2000 -- Modbus address of FIO0
AIN2_EF_CONFIG_D = 1000 -- Scan rate
AIN2_EF_CONFIG_E = 0.65 -- Threshold
```

Reading register AIN2_EF_READ_A returns 1.0 and the calculated average is 66%.

14.2 Extended Channels (T7 Only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview - T7 Only

The T7 has 14 built-in analog inputs (AIN0 through AIN13). For applications that need more, up to 112 total external analog inputs can be utilized by multiplexing the built-in analog inputs. Each built-in analog channel may be multiplexed into 8 new channels, where the starting extended channel number is:

```
startingExtendedChannel = builtinChannel * 8 + 16
```

The extended channel range begins at AIN16.

When a built-in channel is multiplexed, the original channel number becomes unavailable.

For example, if AIN4 is multiplexed, AIN48 through AIN55 are exposed as usable AIN and AIN4 is no longer available.

Differential Channels Pairings

For differential channels in the extended range, a channel may be the positive channel if it is one of the following ranges:

- AIN16 through AIN23
- AIN32 through AIN39
- AIN48 through AIN55
- AIN64 through AIN71
- AIN80 through AIN87
- AIN96 through AIN103
- AIN112 through AIN119

The negative channel is 8 higher than the positive channel. For example, the positive channel AIN48 is paired with AIN56 as the negative channel.

The differential channel pairs for the built-in AIN0 through AIN13 are adjacent even/odd pairs such that the positive channel is even and the negative channel is greater than the positive channel by 1. Since the built-in AIN are multiplexed as described above, an extended negative channel is 8 higher than its corresponding extended positive channel. For example, a valid differential extended channel pair would be a positive channel of AIN70 and AIN78 as the negative channel, since:

- AIN70 maps to AIN6 and
- AIN78 maps to AIN7.

For more information on differential extended channels, see the [Mux80 Datasheet](#).

MIO

The [DB37 connector](#) has 3 MIO lines designed to address expansion multiplexer ICs (integrated circuits), allowing for up to 112 total external analog inputs. Whenever a read is done on analog input channel numbers 16 to 127, the T7 will automatically control the MIO lines to read the correct multiplexed AIN.

Mux80

The [Mux80](#) accessory is used to multiplex AIN4 through AIN13, expanding them from 10 inputs to 80 inputs. AIN0 through AIN3 are still available (on the screw terminals of the T7). The extended channels can be read using the following registers. For further details, see the [Mux80 Datasheet](#).

Mux80 Extended Channels			
Name	Start Address	Type	Access
AIN#(48:127)	96	FLOAT32	R
AIN#(48:127) - Starting Address: 96			
Returns the voltage of the specified analog input.			
<ul style="list-style-type: none">• Data type: FLOAT32 (type index = 3)• Read-only• This register may be streamed			
Expanded Names		Addresses	
AIN48, AIN49, AIN50, AIN51, AIN52, AIN53, AIN54, AIN55, AIN56, AIN57, AIN58, AIN59, AIN60, AIN61, AIN62, AIN63, AIN64, AIN65, AIN66, AIN67, AIN68, AIN69, AIN70, AIN71, AIN72, AIN73, AIN74, AIN75, AIN76, AIN77, AIN78, AIN79, AIN80, AIN81, AIN82, AIN83, AIN84, AIN85, AIN86, AIN87, AIN88, AIN89, AIN90, AIN91, AIN92, AIN93, AIN94, AIN95, AIN96, AIN97, AIN98, AIN99, AIN100, AIN101, AIN102, AIN103, AIN104, AIN105, AIN106, AIN107, AIN108, AIN109, AIN110, AIN111, AIN112, AIN113, AIN114, AIN115, AIN116, AIN117, AIN118, AIN119, AIN120, AIN121, AIN122, AIN123, AIN124, AIN125, AIN126, AIN127 Show All		96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254 Show All	

Note that when using the Mux80 board, the T7's MIO(0-2) lines are consumed for multiplexer signaling.

Other Multiplexers

If you are interested in adding multiplexers yourself, see further information in [Section 2.6.1 of the U6 Datasheet](#). The DG408 from Intersil is a recommended multiplexer, and a convenient ± 12 volt power supply is available on the DB37 so the multiplexers can pass bipolar signals (see Vm+/Vm- documentation in [Section 16.0](#)).

Differential Tables

The tables below show which channels are positive (Pos) or negative (Neg) when paired differentially.

Built-in AIN (not multiplexed):

Pos Neg

<u>Pos</u>	<u>Neg</u>
2	3
4	5
6	7
8	9
10	11
12	13

AIN0 and AIN1 (multiplexed with an external multiplexer):

<u>Pos</u>	<u>Neg</u>
16	24
17	25
18	26
19	27
20	28
21	29
22	30
23	31

AIN2 and AIN3 (multiplexed with an external multiplexer):

<u>Pos</u>	<u>Neg</u>
32	40
33	41
34	42
35	43
36	44
37	45
38	46
39	47

AIN4 and AIN5 multiplexed with the Mux80 (or an external multiplexer):

<u>Pos</u>	<u>Neg</u>
48	56
49	57
50	58
51	59
52	60
53	61
54	62
55	63

AIN6 and AIN7 multiplexed with the Mux80 (or an external multiplexer):

<u>Pos</u>	<u>Neg</u>
64	72
65	73
66	74
67	75
68	76
69	77
70	78
71	79

AIN8 and AIN9 multiplexed with the Mux80 (or an external multiplexer):

<u>Pos</u>	<u>Neg</u>
80	88
81	89
82	90
83	91
84	92
85	93
86	94
87	95

AIN10 and AIN11 multiplexed with the Mux80 (or an external multiplexer):

<u>Pos</u>	<u>Neg</u>
96	104
97	105
98	106
99	107

Pos Neg

101 109

102 110

103 111

AIN12 and AIN13 multiplexed with the Mux80 (or an external multiplexer):

Pos Neg

112 120

113 121

114 122

115 123

116 124

117 125

118 126

119 127

15.0 DAC [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Range: **~0V to ~5V**

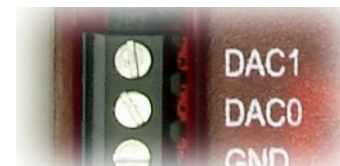
Resolution: **T4 = 10-bit**

T7 = 12-bit

Source Impedance: **50 ohms**

Max Output Current: **20mA***

*See [Appendix A-4](#) for details on voltage drop related to current draw.



Overview

There are two DACs (digital-to-analog converters, also known as analog outputs) on T-series devices. Each DAC can be set to a voltage between about 0 and 5 volts with 10 bits of resolution (T4) or 12 bits of resolution (T7).

For electrical specifications, See [Appendix A-4](#).

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 voltage is limited to slightly less than VS.

The T7 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.

Register Listing

To set DAC output voltage, write to the following registers:

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

DAC#(0:1) - Starting Address: 1000

Pass a voltage for the specified analog output.

- Data type: FLOAT32 (type index = 3)
- Readable and writable

Expanded Names	Addresses
DAC0, DAC1	1000, 1002

Output Range

The gain of the DAC output amplifiers is designed so the nominal max output is 5.0 volts. The actual max voltage can vary and will always be limited to the supply voltage. To determine the actual max of each DAC, write an impossibly high value such as 6.0 to the DAC, and then read back the value. This is the expected full-scale output with no load and with a supply voltage greater than the returned value.

The output range is also limited by the ability of the output amps to drive near the power rails (0 and VS). They can drive quite close, especially at light load, but will never be able to drive all the way to exactly 0.0 or VS.

The output range is further limited under load by the drive ability of the output amp and the 50 ohms of source resistance. The latter is dominant at lower currents, so for example if you set a DAC to 4.000 volts and draw 2 mA from it, the output will be closer to 3.900 volts.

See [Appendix A-4](#) for more information.

For voltages all the way to 0.0 and 5.0 volts (and beyond), the [LJTick-DAC](#) is a great solution.

Power-up Defaults

The power-up condition of the DACs can be configured by the user. From the factory, the DACs default to be 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.

10 Hz Square Wave Output

DAC1 can be configured to output a 10 Hz, 3.3 V square wave. Writing a 1 to DAC1_FREQUENCY_OUT_ENABLE will enable the 10 Hz output:

Name	Start Address	Type	Access
DAC1_FREQUENCY_OUT_ENABLE	61532	UINT32	W

DAC1_FREQUENCY_OUT_ENABLE - Address: 61532

0 = off, 1 = output 10 Hz signal on DAC1. Note that writing to DAC1 or enabling a stream out which is targeting DAC1 will disable this feature.

- Data type: UINT32 (type index = 1)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0234

The square wave output will be disabled:

- when 0 is written to DAC1_FREQUENCY_OUT_ENABLE,
- when any value is written DAC1,
- or when a stream-out channel targeting DAC1 is enabled.

T7 requires firmware version 1.0234 or later.

Increasing Output to ±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 many can be added to a T-series device.

Testing

A good way to test the DACs is to see if they output the expected voltage when loaded. The source impedance of the DACs is about 50 ohms, and that source impedance interacts with load impedance to form a voltage divider. Remove all user connections and connect a 470 ohm resistor from DAC# to GND. A voltage divider is formed such that:

$$V_{out} = V_{set} * 470 / (470 + 50) = V_{set} * 0.9$$

So whatever the DAC is set to, the output at the terminal should be about 10% less. You will see the 10% drop with test voltages up to about 4 volts. Above 4 volts, other effects come into play since this test is drawing substantial current from the DAC line.

16.0 DB37 (T7 Only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview - T7 Only

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.



The **CB37** is a connector board that provides convenient screw-terminals for the DB37 lines, but the CB37 is not required to access I/O on the DB37. Any method you see fit can be used to access the DB37 lines.

Pinout

Some signals appear on both the DB37 connector and screw terminals, so care must be taken to avoid contention. For such signals, only connect to one location, not both. Signals duplicated on the T7 screw terminals and the DB37 are denoted in **bold**:

Table 16-1. DB37 Connector Pinouts

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

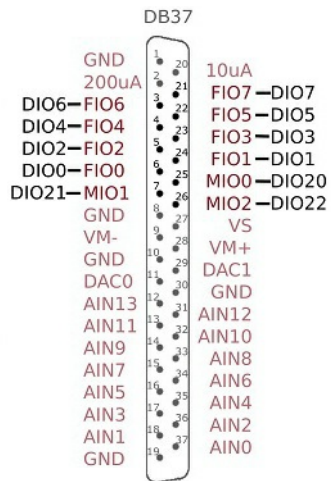


Figure 16-2. Standard DB37 pin numbers looking into the female connector on the T7

Remarks

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.

Duplicated Input Terminals (AIN0-AIN3 and FIO0-FIO3)

AIN0-AIN3 and FIO0-FIO3 appear on the built-in screw-terminals and also on the DB37 connector. You should only connect to one or the other, not both at the same time.

To prevent damage due to accidental short circuit, both connection paths have their own series resistor.

All FIO lines have a 470 ohm series resistor (that is included in the 550 ohm total impedance), and in the case of FIO0-FIO3 the duplicated connections each have their own series resistor, so if you measure the resistance between the duplicate terminals you will see about 940 ohms.

All AIN lines have a 2.2k series resistor, and in the case of AIN0-AIN3 the duplicated connections each have their own series resistor, so if you measure the resistance between the duplicate terminals you will see about 4.4k.

CB37 Terminal Board

The [CB37 terminal board](#) from LabJack connects to the DB37 connector and provides convenient screw terminal access to all lines. The CB37 is designed to connect directly to the DB37, but can also connect via a 37-line 1:1 male-female cable.

When using the analog connections on the CB37, the effect of ground currents should be considered, particularly when a cable is used and substantial current is sourced/sunk through the CB37 terminals. When any sizable cable lengths are involved, a good practice is to separate current carrying ground from ADC reference ground. An easy way to do this on the CB37 is to use GND as the current source/sink, and use AGND as the reference ground. This works well for passive sensors (no power supply), such as a thermocouple, where the only ground current is the return of the input bias current of the analog input.

EB37 Experiment Board

The [EB37 experiment board](#) connects to the DB37 connector and provides convenient screw terminal access. Also provided is a solderless breadboard and useful power supplies. The EB37 is designed to connect directly to the DB37, but can also connect via a 37-line 1:1 male-female cable.

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 [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

Number of Pins: **15**

Screw type: **#4-40**

Contacts: **Gold-coated**

Form factor: **D-Sub**

The DB15 connector has the potential to be used as an expansion bus, where the 8 EIO are data lines and the 4 CIO are control lines.

The [CB15](#) is a connector board that provides convenient screw-terminals for the DB15 lines, but the CB15 is not required to access I/O on the DB15. 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. For details, see the "Protection" section of [13.0 Digital I/O](#).

All digital I/O on T-series devices have 3 possible states: input, output-high, or output-low. For details, see the "Electrical Overview" section of [13.0 Digital I/O](#).

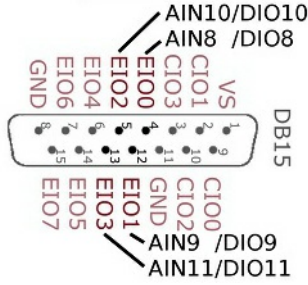
Pinout By Device

T4

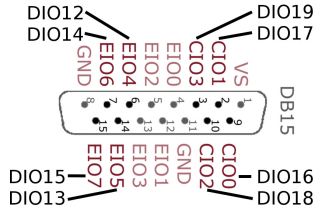


The DB15 connector brings out 4 flexible I/O as well as 8 dedicated digital I/O.

The flexible I/O ports EIO0-EIO3 can be configured to be the analog inputs AIN8-AIN11 or the digital I/O ports DIO8-DIO11:

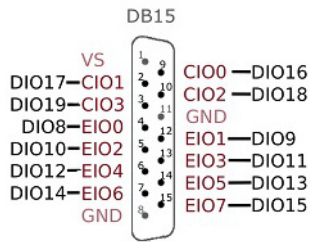


The dedicated digital I/O ports EIO4-CIO3 are DIO12-DIO19:



T7

The DB15 connector brings out 12 additional digital I/O, EIO0-CIO3, which can also be addressed as DIO8-DIO19:



Remarks

CB15

The CB15 terminal board connects to the DB15 connector. It provides convenient screw terminal access to the 12 I/O channels available on the DB15 connector. The CB15 is designed to connect directly to the DB15, or can connect via a standard 15-line 1:1 male-female DB15 cable.

RB12

The RB12 relay board provides a convenient interface for T-series devices to industry standard digital I/O modules, allowing electricians, engineers, and other qualified individuals to interface a LabJack with high voltages/currents. The RB12 relay board connects to the DB15 connector on the LabJack, using the 12 EIO/CIO lines to control up to 12 I/O modules. Output or input types of digital I/O modules can be used. The RB12 is designed to accept G4 series digital I/O modules from Opto22, and compatible modules from other manufacturers such as the G5 series from Grayhill. Output modules are available with voltage ratings up to 200 VDC or 280 VAC, and current ratings up to 3.5 amps.

OEM

OEM T-series devices have 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 [T-Series Datasheet]

[Log in or register](#) to post comments

Overview By Device

T4

Sensor Range: **-50° C to 150° C**

T4 Operating Range: **-40° C to 85° C**

Accuracy (20° C to 40° C): **±1.5° C***

Accuracy (-20° C to 50° C): **±2.0° C***

Accuracy (-45° C to 85° C): **±3.5° C***

The T4 has an LM94021 temperature sensor (with GS=10) connected to an internal analog input. The sensor is physically located on the top of the PCB behind the VS screw terminal of the FIO4 and FIO5 screw terminal block.



T7

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 (-20° C to 50° C): **±2.0° C***

Accuracy (-45° C to 85° C): **±3.5° C***

The T7 has an LM94021 temperature sensor (with GS=10) 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.



*Accuracy of measuring device temperature, which is typically warmer than ambient air temperature. Includes error from LM94021 specifications and error due to linear equation fit.

Device Temperature

Read `TEMPERATURE_DEVICE_K` to get the device temperature:

Name	Start Address	Type	Access
TEMPERATURE_DEVICE_K	60052	FLOAT32	R

TEMPERATURE_DEVICE_K - Address: 60052

Takes a reading from the internal temperature sensor using range= $\pm 10V$ and resolution=8, and applies the formula $\text{Volts} \cdot -92.6 + 467.6$ to return kelvins.

- Data type: FLOAT32 (type index = 3)
- Read-only
- T7:
 - The internal temperature sensor, 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.
 - Minimum [firmware](#) version: 1.0105
- T4:
 - The internal temperature sensor is internally connected to an LM94021 (U24) with GS=10 which is physically located on the top of the PCB behind the VS screw terminal of the FIO4 and FIO5 screw terminal block.
 - Minimum [firmware](#) version: 0.2020

T7 only:

AIN14 Temperature Sensor Voltage

Alternatively, reading from AIN14 returns the temperature sensor voltage, which can be converted to device temperature using the formula:

$$\text{Device temperature K} = \text{volts} \cdot -92.6 + 467.6$$

Stream Mode

You can read the raw temp sensor voltage from AIN14 in stream mode, but cannot read `TEMPERATURE_DEVICE_K` or `TEMPERATURE_AIR_K` since the internal math that is required for them is too slow. If streaming, use AIN14 to get volts and use the above formula to get device temperature in K. For an estimate of air temperature, see the following "Air Temperature" section.

Air Temperature

TEMPERATURE_AIR_K is an estimate of the ambient air temperature outside the device:

Name	Start Address	Type	Access
TEMPERATURE_AIR_K	60050	FLOAT32	R

TEMPERATURE_AIR_K - Address: 60050

Returns the estimated ambient air temperature just outside of the device 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.

- Data type: FLOAT32 (type index = 3)
- Read-only
- T7:
 - Minimum [firmware](#) version: 1.0105
- T4:
 - Minimum [firmware](#) version: 0.2020

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 - T7 Only:

The value from 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.

The internal sensor has a specified accuracy of ± 2.0 °C across the range of -20 to +50 °C. Allowing for a slight difference between the sensor temperature and the temperature of the screw-terminals, expect the returned value minus 3 °C to reflect the temperature of the built-in screw-terminals with an accuracy of ± 2.5 °C.

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 (T7 Only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview - T7-Pro Only

The T7-Pro has a battery-backed RTC (real-time clock) which is useful for assigning timestamps to data that is stored on the microSD 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. The system time is stored in seconds since 1970, also known as the Epoch and Unix timestamp.

Battery Life

Typically, the CR2032 battery can be expected to last 12 years. Several factors can influence the expected life including:

- Temperatures - Cold temperatures will decrease the battery's life.
- Battery age - Batteries naturally discharge as they age. This also has a temperature dependency, but 20% every 10 years is a good rule of thumb.
- External Power - When the T7 is powered up the battery is not used to maintain time. This will extend the battery life.

The RTC requires 925 nA to maintain its time. A typical CR2032 battery has 210 mAH capacity which works out to a life of 25.9 years. We derate our expectations by ~ 2 to get the above life expectancy.

Reading Time

Read the system time in seconds with address 61500:

Name	Start Address	Type	Access
RTC_TIME_S	61500	UINT32	R

RTC_TIME_S - Address: 61500

Read the current time in seconds since Jan, 1970, aka Epoch or Unix time. This value is calculated from the 80 MHz crystal, not the RTC 32 kHz crystal. Non-pro devices do not have a real time clock, so the reported time is either seconds since device startup or the time reported by the network time protocol over Ethernet. Pro devices have a real time clock that will be used to initialize this register at startup, the time can then be updated by NTP.

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum [firmware](#) version: 1.0128
- T4:
 - Minimum [firmware](#) version: 0.2020

Get a simple calendar time representation by reading six consecutive addresses, starting with address 61510:

Name	Start Address	Type	Access
RTC_TIME_CALENDAR	61510	UINT16	R

RTC_TIME_CALENDAR - Address: 61510

Read six consecutive addresses of type UINT16, starting with this address. The result will be in year, month, day, hour, minute, second calendar format. i.e. [2014, 10, 21, 18, 55, 35]

- Data type: UINT16 (type index = 0)
- Read-only
- T7:
 - Minimum [firmware](#) version: 1.0128
- T4:
 - Minimum [firmware](#) version: 0.2020

To time events faster than 1 second apart, it is possible to read the CORE_TIMER (address 61520) and see how it changes from second to second. To access the core timer value in Lua scripts, use the LJ.Tick() function.

Setting Time

Set the system time by writing a new timestamp (in seconds) to address 61504:

Name	Start Address	Type	Access
RTC_SET_TIME_S	61504	UINT32	W

RTC_SET_TIME_S - Address: 61504

Write a new timestamp to the RTC in seconds since Jan, 1970, aka Epoch or Unix timestamp.

- Data type: UINT32 (type index = 1)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0128
- T4:
 - Minimum [firmware](#) version: 0.2020

To set the T7-Pro's time from a SNTP server, use RTC_SET_TIME_SNTP and SNTP_UPDATE_INTERVAL:

Name	Start Address	Type	Access
RTC_SET_TIME_SNTP	61506	UINT32	W
SNTP_UPDATE_INTERVAL	49702	UINT32	R/W

RTC_SET_TIME_SNTP - Address: 61506

Write any value to instruct the T7 to update its clock from a SNTP server. Requires that SNTP_UPDATE_INTERVAL is non-zero.

- Data type: UINT32 (type index = 1)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0128
- T4:
 - Minimum [firmware](#) version: 0.2020

SNTP_UPDATE_INTERVAL - Address: 49702

Sets the SNTP retry time in seconds. A value of zero will disable SNTP(0=default). Values must be 10 or greater.

- Data type: UINT32 (type index = 1)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 1.0150
- T4:
 - Minimum [firmware](#) version: 0.2020

Examples

[Lua] Read the value of the RTC in a Lua script

```
table = {}  
table[1] = 0 --year  
table[2] = 0 --month
```

```

table[3] = 0 --day
table[4] = 0 --hour
table[5] = 0 --minute
table[6] = 0 --second

table, error = MB.RA(61510, 0, 6)
print(string.format("%04d/%02d/%02d %02d:%02d.%02d", table[1], table[2], table[3], table[4], table[5], table[6]))

>> 2014/10/15 18:55.22

```

[C/C++] Read the value of the RTC in C/C++

```

int LJMError;
double newValue;
LJMError = LJM_eReadAddress(handle, 61500, 1, &newValue);
printf(newValue);
//returned value of 1413398998 would correspond with Wed, 15 Oct 2014 18:49:58 GMT

```

20.0 Internal Flash [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

T-series devices have 4 MB of internal flash.

Internal flash memory is divided into two regions:

- The User Area is the first 2 MB and is available for storing user-data.
- The Reserved Area is the remaining 2 MB and is used to store important device information such as calibration constants.

Each region has a starting address, a length, and a key.

Flash Addresses

Internal flash is addressed by byte, but accessed by 32-bit values. That means that the first value is at address zero and the second is at address 4. (These are flash addresses—not to be confused with Modbus addresses.)

Following are some important flash addresses:

Address Name	Address	Length	
User Area	0x0	2 MB	Key: 0x6615E336 (1712710454 in decimal)
Reserved Area	0x200000	2 MB	
<u>Calibration Constants</u> (in Reserved Area)	0x3C7000	4 kB	Key: 0x43A24C42 (1134709826 in decimal)

Reading

To read from flash, write the desired address to INTERNAL_FLASH_READ_POINTER and then read an even number of registers from INTERNAL_FLASH_READ:

Name	Start Address	Type	Access
INTERNAL_FLASH_READ_POINTER	61810	UIN32	R/W
INTERNAL_FLASH_READ	61812	UIN32	R

INTERNAL_FLASH_READ_POINTER - Address: 61810
 The address in internal flash that reads will start from.

- Data type: UIN32 (type index = 1)
- Readable and writable

INTERNAL_FLASH_READ - Address: 61812
 Data read from internal flash.

- Data type: UIN32 (type index = 1)
- Read-only

Writing and Erasing

Key

For a region to be written to or to be erased, the key for that region must be written to the INTERNAL_FLASH_KEY register:

Name	Start Address	Type	Access
------	---------------	------	--------

Name	Start Address	Type	Access
INTERNAL_FLASH_KEY - Address: 61800			
Sets the region of internal flash to which access is allowed.			
<ul style="list-style-type: none"> Data type: UINT32 (type index = 1) Readable and writable 			

The key prevents accidental overwrites. The value in INTERNAL_FLASH_KEY will be cleared when the Modbus packet that wrote it has been fully processed, so INTERNAL_FLASH_KEY must be written in the same packet that does INTERNAL_FLASH_WRITE or INTERNAL_FLASH_ERASE. The [LJM Multiple Value functions](#) simplify this.

Writing

To write to flash, the area to be written to must first be erased. Once erased, write the key for the desired region to INTERNAL_FLASH_KEY, write the desired address to INTERNAL_FLASH_WRITE_POINTER, and then write an even number of registers to INTERNAL_FLASH_WRITE:

Name	Start Address	Type	Access
INTERNAL_FLASH_WRITE_POINTER	61830	UINT32	R/W
INTERNAL_FLASH_WRITE	61832	UINT32	W

INTERNAL_FLASH_WRITE_POINTER - Address: 61830
Address in internal flash where writes will begin.

- Data type: UINT32 (type index = 1)
- Readable and writable

INTERNAL_FLASH_WRITE - Address: 61832
Data written here will be written to internal flash. This register is a buffer.

- Data type: UINT32 (type index = 1)
- Write-only
- This register is a [Buffer Register](#)

Erasing

Flash is erased 4 kB at a time. Erasing sets all bits to 1. To erase a 4 kB region, write the key for the desired region to INTERNAL_FLASH_KEY and the desired address to INTERNAL_FLASH_ERASE:

Name	Start Address	Type	Access
INTERNAL_FLASH_ERASE	61820	UINT32	W

INTERNAL_FLASH_ERASE - Address: 61820
Erases a 4k section of internal flash starting at the specified address. This register is a buffer.

- Data type: UINT32 (type index = 1)
- Write-only
- This register is a [Buffer Register](#)

The address will be rounded down to the nearest 4 kB boundary. Boundaries are easy to identify when the address is displayed in hexadecimal because the lower three digits will be zero. 4 kB is hexadecimal is 0x1000.

Endurance

Flash memory can only be erased so many times before bit errors will start to occur; it is important not to erase or write flash needlessly. Typical life of flash memory is at least 10,000 cycles.

20.0.0 T4 Calibration Constants [T-Series Datasheet]

[Log in or register](#) to post comments

The T4 automatically returns calibrated readings, so most people need not concern themselves with this section.

The factory applied calibration constants are stored in Internal Flash and can be accessed at any time through the use of the Modbus registers discussed in the parent to this section ([Internal Flash section](#)).

The calibration constants begin at memory address 0x3C4000, or in decimal format d3948544. The structure (location) of each calibration value can be seen in the C code snippet below:

```
typedef struct{
    struct {
        float Slope;
        float Offset;
    } HV[4];
    struct {
        float Slope;
        float Offset;
```

```

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

float Temp_Slope;
float Temp_Offset;

float I_Bias;
} DeviceCalibrationT4;

```

Nominal Calibration Values

	Slope	Offset
HV[0] (AIN0)	3.235316E-04	-10.532965
HV[1] (AIN1)	3.236028E-04	-10.534480
HV[2] (AIN2)	3.235439E-04	-10.530597
HV[3] (AIN3)	3.236133E-04	-10.530210
LV	3.826692E-05	0.002484
SpecV	-3.839420E-05	2.507430
DAC0	1.310768E+04	54.091066
DAC1	1.310767E+04	54.044314
Temp	-	467.600000
	9.260000E+01	

AIN Bias Current: 0.000000015

20.0.1 T7 Calibration Constants [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

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

The factory applied calibration constants are stored in Internal Flash and can be accessed at any time through the use of the Modbus registers discussed in the parent to this section ([Internal Flash section](#)).

The calibration constants begin at memory address 0x3C4000, or in decimal format d3948544. The structure (location) of each calibration value can be seen in the C code snippet below.

```

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

Nominal Calibration Values

±10V Range:

- Positive Slope: 0.000315805780
- Negative Slope: -0.000315805800
- Binary Center: 33523
- Voltage Offset: -10.586956522

±1V Range:

- Positive Slope: 0.000031580578
- Negative Slope: -0.000031580600
- Binary Center: 33523
- Voltage Offset: -1.0586956522

±0.1V Range:

- Positive Slope: 0.000003158058
- Negative Slope: -0.000003158100
- Binary Center: 33523
- Voltage Offset: -0.1058695652

±0.01V Range:

- Positive Slope: 0.000000315806
- Negative Slope: -0.000000315800
- Binary Center: 33523
- Voltage Offset: -0.010586956

DACs:

- Slope: 13200
- Offset: 0

Temperature:

- Slope: -92.6
- Offset: 467.6

Current Sources:

- 10 μ A: 0.000010
- 200 μ A: 0.000200

AIN Bias Current: 0.000000015

21.0 SD Card (T7 Only) [T-Series Datasheet]

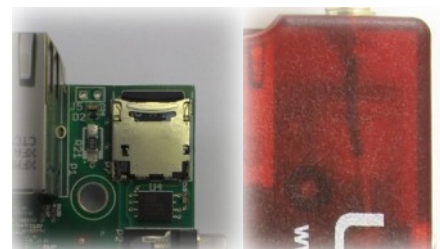
[Log in](#) or [register](#) to post comments

Overview - T7 Only

The T7-Pro ships with a 2GB (or larger) microSD card installed (SLC technology). It might also be referred to as uSD, μ SD, or just SD.

The T7 does not have the microSD card installed, but does have the card holder installed so a compatible microSD card can be installed in the field.

The retainer opens by sliding the metal piece forward, then lifting.



Currently microSDXC is not supported. Generally speaking, anything above 2GB is 'HC' meaning high capacity, and HC cards might need to be reformatted before they work.

The T7 supports FAT and FAT32 file systems, but some makes and sizes behave differently. We recommend the following SD card format:

File System: FAT
Allocation unit size: 64 kilobytes

FAT32 with an allocation unit size of 16 kB or 32 kB sometimes works, but smaller allocation sizes generally do not. On 2 GB cards it's possible to select FAT format with a 32 kB allocation size, and that sometimes works.

Care must be taken to ensure that power is not lost during file writing or disk corruption could occur. The rated operating temperature of the SD card is -25° C to 85° C. For extremely low temperatures, customers can buy industrial grade SD cards, such as the [AF1GUDI-OEM](#), from ATP Electronics, Inc.

File and directory names are limited to ASCII characters only.

The maximum number of file handles that can be open at once on a T7 is 4. One file handle is reserved for Modbus communication to allow programs to read files off of the SD card and the remaining 3 are available for Lua Scripts.

Standalone Data Logging

The microSD card is generally only useful for people doing standalone data logging through [Lua scripting](#), since normal T7 operation is with a host connected (so the host can store data). Standalone logging is an advanced topic. We provide Lua examples for logging data to the microSD, but options for retrieving the data are somewhat limited. Standalone logging is generally limited to [command-response data rates](#).

Retrieving Data from the microSD Card

It is pretty easy to write data to the microSD card in a Lua script. At this time there are a few options for retrieving the data from the microSD:

1. Remove (or swap) the microSD from the T7 and put it in a card reader on a computer. On non-OEM versions of the T7/T7-Pro, the enclosure must be opened to access the microSD holder.
2. Use the [beta SD utility](#) for downloading files from the SD card via USB/Ethernet/WiFi on Windows, only. Note that the microSD and WiFi share a serial bus inside the T7, so sometimes extra thought is required if using both at the same time.
3. Use the registers described below to read data off the microSD.

Testing the microSD Card

Not all SD cards are compatible. If you install a new or different SD card, we recommend the following tests to make sure it works. If you have access to a Windows computer, we recommend downloading and running the [T7uSD testing application](#) published on the [T-Series Additional Utility Applications](#) page.

- A. Check Kipling to make sure that the SD card is properly recognized and that the calibration status of the device is still good.
 1. Connect to the device with Kipling
 2. On the [Device Info tab](#) and make sure there is a green checkmark next to the SD Card Installed hardware option.
 3. Check to make sure the device's calibration status is still "Good".
- B. Check to make sure the SD card can be read and written.
 1. Connect to the device with Kipling.
 2. Download and run the [Lua example](#) script titled "Log voltage to file".
 3. Exit Kipling and open the [SD Utility](#) to verify that the file was written properly.
- C. Check to make sure the T7's calibration constants can be read from the internal flash chip.
 1. If you are on a Windows computer, use LJStreamM to try streaming an analog input register from a T7. If you get stream errors LJME_USING_DEFAULT_CALIBRATION (203) or LJME_INVALID_VALUE (1305), the microSD card being used is likely not compatible with the T7.

Errors Caused by Unsupported microSD Cards

1. A device may get stuck in its recovery firmware version and report upgrade issues because the flash chip is not responding properly.
2. A device may report that it is using default calibration values and it is unable to be used in stream mode. The device's calibration constants are read before streaming is started.
3. A device may have issues when running Lua scripts that try to save data to the microSD card or errors may be reported by the SD Card utility indicating that a file can't be opened.

Accessing the microSD Card While Using WiFi

WiFi shares an internal serial bus with the SD card and Internal Flash, and at the start of joining WiFi needs about 3 seconds of uninterrupted access on this serial bus. If a Lua script does file I/O operations during this time, then WiFi initialization will fail and the WiFi module will immediately try again. If Lua file I/O occurs every 3 seconds or less, it is likely that WiFi will never be able to join. A simple way to make sure WiFi can join, perhaps with a retry or two needed, is to only write every 5 seconds (or longer).

Use the following Lua pseudocode to write to a file once every 5 seconds, and read an analog input once every 500ms. See the "Log voltage to file" [Lua examples](#) in Kipling for an actual script example.

```
LJ.IntervalConfig(0, 500) --set the DAQ interval to 500ms, should divide evenly into file access interval
LJ.IntervalConfig(1, 5000) --set the file access interval to 5 seconds

TableSize = 5000/500
data = {}
DAQcount = 0

for i=1, TableSize do
    data[i] = 0
end

while true do
    if LJ.CheckInterval(0) then --if a data point needs to be collected
        data[DAQcount] = MB.R(0, 3)--collect a new reading from AIN0
        DAQcount = DAQcount + 1
    end
    if LJ.CheckInterval(1) then --file access interval complete
        appendToFile(data) --save the data to a file
        DAQcount = 0
    end
end
```

File I/O General Info

The T7 uses Unix-style file paths where the separator character is a slash: /

The root directory doesn't require a "/". For example, reading a file named "test.txt" in the "tmp" folder could be done with either this following path:

```
tmp/test.txt
```

or, alternately:

```
/tmp/test.txt
```

Common File I/O Operations

See the Register Listing section below for register information.

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

1. Write a value of 1 to FILE_IO_DIR_CURRENT. The error returned indicates whether there is a directory loaded as current. No error (0) indicates a valid directory.
2. Read FILE_IO_PATH_READ_LEN_BYTES.
3. Read an array of size FILE_IO_PATH_READ_LEN_BYTES from FILE_IO_PATH_READ.
4. Resultant string will be something like "/" for the root directory, or "/DIR1/DIR2" for a directory.

Get list of items in the CWD:

1. Write a value of 1 to FILE_IO_DIR_FIRST. The error returned indicates whether anything was found. No error (0) indicates that something was found. FILE_IO_NOT_FOUND (2960) indicates that nothing was found.
2. Read FILE_IO_PATH_READ_LEN_BYTES, FILE_IO_ATTRIBUTES, and FILE_IO_SIZE_BYTES. Store the attributes and size associated with each file.
3. Read an array of size FILE_IO_PATH_READ_LEN_BYTES from FILE_IO_PATH_READ. This is the name of the file/folder.
4. Write a value of 1 to FILE_IO_DIR_NEXT. The error returned indicates whether anything was found. No error (0) indicates that there are more items->go back to step 2. Each of the following errors indicate that there are no more items:
 - o FILE_IO_END_OF_CWD (2966)
 - o FILE_IO_INVALID_OBJECT (2809)
 - o FILE_IO_NOT_FOUND (2960)

Change the CWD:

1. Find from the list of items a directory to open, e.g. "/DIR1". Directories can be parsed out of the list of items by analyzing their FILE_IO_ATTRIBUTES bitmask. If bit 4 of the FILE_IO_ATTRIBUTES bitmask is set, then the item is a directory.
2. Write the directory name length in bytes to FILE_IO_PATH_WRITE_LEN_BYTES (ASCII, so each char is 1 byte, also don't forget to add 1 for the null terminator).
3. Write the directory string (converted to an array of bytes, with null terminator) to FILE_IO_PATH_WRITE. (array size = length from step 2)
4. Write a value of 1 to FILE_IO_DIR_CHANGE.
5. Done. Optionally get a list of items in the new CWD.

Get disk size and free space:

1. Read FILE_IO_DISK_SECTOR_SIZE_BYTES, FILE_IO_DISK_SECTORS_PER_CLUSTER, FILE_IO_DISK_TOTAL_CLUSTERS, FILE_IO_DISK_FREE_CLUSTERS. All disk parameters are captured when you read FILE_IO_DISK_SECTOR_SIZE_BYTES.
2. Total size = SECTOR_SIZE * SECTORS_PER_CLUSTER * TOTAL_CLUSTERS.
3. Free size = SECTOR_SIZE * SECTORS_PER_CLUSTER * FREE_CLUSTERS.

Get disk format:

1. Read FILE_IO_DISK_FORMAT_INDEX
2. 2=FAT, 3=FAT32

Read a file:

1. Write the length of the file name (including the null terminator) to FILE_IO_PATH_WRITE_LEN_BYTES
2. Write the name to FILE_IO_PATH_WRITE (with null terminator)
3. Write any value to FILE_IO_OPEN
4. Read file data from FILE_IO_READ using the size from FILE_IO_SIZE_BYTES (FILE_IO_SIZE_BYTES can be read while getting a list of items in the CWD)
5. Write a value of 1 to FILE_IO_CLOSE

Write a file:

1. Modbus interface unimplemented. Users are typically expected to create files from onboard Lua scripts. Contact support@labjack.com if you are interested in creating files via Modbus.

Create a directory:

Unimplemented. Since Lua scripts currently do not have the ability to write files anywhere except the root directory, this feature is not implemented.

Register Listing

File IO Navigation

Name	Start Address	Type	Access
FILE_IO_PATH_WRITE_LEN_BYTES	60640	UINT32	W
FILE_IO_PATH_READ_LEN_BYTES	60642	UINT32	R
FILE_IO_PATH_WRITE	60650	BYTE	W
FILE_IO_PATH_READ	60652	BYTE	R
FILE_IO_DIR_FIRST	60610	UINT16	W
FILE_IO_DIR_NEXT	60611	UINT16	W
FILE_IO_DIR_CHANGE	60600	UINT16	W
FILE_IO_DIR_CURRENT	60601	UINT16	W
FILE_IO_DIR_MAKE	60602	UINT16	W
FILE_IO_DIR_REMOVE	60603	UINT16	W

FILE_IO_PATH_WRITE_LEN_BYTES - Address: 60640

Write the length (in bytes) of the file path or directory to access.

- Data type: UINT32 (type index = 1)
- Write-only
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_PATH_READ_LEN_BYTES - Address: 60642

Read the length (in bytes) of the next file path or directory to access.

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_PATH_WRITE - Address: 60650

Write the desired file path. Must first write the length of the file path string (in bytes) to FILE_IO_PATH_WRITE_LEN_BYTES. File paths should be null terminated. This register is a buffer.

- Data type: BYTE (type index = 99)
- Write-only
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_PATH_READ - Address: 60652

Read the next file path in the CWD. Length of the string (in bytes) determined by FILE_IO_PATH_READ_LEN_BYTES. File paths will be null terminated. This register is a buffer. Underrun behavior - fill with zeros.

- Data type: BYTE (type index = 99)
- Read-only
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_DIR_FIRST - Address: 60610

Write any value to this register to initiate iteration through files and directories in the CWD. Typical sequence: FILE_IO_DIR_FIRST(W), then loop through: [FILE_IO_NAME_READ_LEN(R), FILE_IO_NAME_READ(R), FILE_IO_ATTRIBUTES(R), FILE_IO_SIZE_BYTES(R), FILE_IO_DIR_NEXT(W)] ..until any of the following error8FILE_IO_END_OF_CWD (2966), FILE_IO_INVALID_OBJECT (2809), or FILE_IO_NOT_FOUND (2960).

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_DIR_NEXT - Address: 60611

Write any value to this register to continue iteration through files and directories in the CWD.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_DIR_CHANGE - Address: 60600

Write any value to this register to change the current working directory (CWD). Must first designate which directory to open by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0156

FILE_IO_DIR_CURRENT - Address: 60601

Write any value to this register to load the current working directory into FILE_IO_PATH_READ, and its length into FILE_IO_PATH_READ_LEN_BYTES. Can be used to identify current position in a file tree.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_DIR_MAKE - Address: 60602

Unimplemented.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 99.0000

FILE_IO_DIR_REMOVE - Address: 60603

Unimplemented.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 99.0000

File IO File Operations

Name	Start Address	Type	Access
FILE_IO_OPEN	60620	UINT16	W
FILE_IO_CLOSE	60621	UINT16	W
FILE_IO_WRITE	60654	BYTE	W
FILE_IO_READ	60656	BYTE	R
FILE_IO_DELETE	60622	UINT16	W
FILE_IO_ATTRIBUTES	60623	UINT16	R
FILE_IO_SIZE_BYTES	60628	UINT32	R

FILE_IO_OPEN - Address: 60620

Write any value to this register to open a file. Must first designate which file to open by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_CLOSE - Address: 60621

Write any value to this register to close the open file.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_WRITE - Address: 60654

Unimplemented. This register is a buffer.

- Data type: BYTE (type index = 99)
- Write-only
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 99.0000

FILE_IO_READ - Address: 60656

Read the contents of a file. Must first write to FILE_IO_OPEN. Size of the file (in bytes) determined by FILE_IO_SIZE_BYTES. This register is a buffer. Underrun behavior - throws an error.

- Data type: BYTE (type index = 99)
- Read-only
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 1.0134

FILE_IO_DELETE - Address: 60622

Write any value to this register to delete the active file. Must first designate which file to delete by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

- Data type: UINT16 (type index = 0)
- Write-only
- T7:
 - Minimum firmware version: 1.0134

FILE_IO_ATTRIBUTES - Address: 60623

Bitmask: Bit0: Reserved, Bit1: Reserved, Bit2: Reserved, Bit3: Reserved, Bit4: 1=Directory, Bit5: 1=File. Used to differentiate files from directories/folders.

- Data type: UINT16 (type index = 0)
- Read-only
- T7:
 - Minimum firmware version: 1.0134

FILE_IO_SIZE_BYTES - Address: 60628

The size of the file in bytes. Directories have 0 size.

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum firmware version: 1.0134

Read from the disk information registers to get free space and other information.

File IO Disk Information

Name	Start Address	Type	Access
FILE_IO_DISK_SECTOR_SIZE_BYTES	60630	UINT32	R
FILE_IO_DISK_SECTORS_PER_CLUSTER	60632	UINT32	R
FILE_IO_DISK_TOTAL_CLUSTERS	60634	UINT32	R
FILE_IO_DISK_FREE_CLUSTERS	60636	UINT32	R
FILE_IO_DISK_FORMAT_INDEX	60638	UINT32	R

FILE_IO_DISK_SECTOR_SIZE_BYTES - Address: 60630

The size of each sector in the SD card in bytes. In Windows this is called the Allocation Size.

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum firmware version: 1.0134

FILE_IO_DISK_SECTORS_PER_CLUSTER - Address: 60632

The number of sectors in each cluster. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum firmware version: 1.0134

FILE_IO_DISK_TOTAL_CLUSTERS - Address: 60634

The total number of clusters in the SD card. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum firmware version: 1.0134

FILE_IO_DISK_FREE_CLUSTERS - Address: 60636

Free (available) clusters in the SD card. Used to determine free space. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum firmware version: 1.0134

FILE_IO_DISK_FORMAT_INDEX - Address: 60638

Used to determine the format of the SD card. 0=None or Unknown, 1=FAT12, 2=FAT16(Windows FAT), 3=FAT32

- Data type: UINT32 (type index = 1)
- Read-only
- T7:
 - Minimum firmware version: 1.0160

The registers that are type BYTE are AAI binary.

FILE IO and Lua

Name	Start Address	Type	Access
FILE_IO_LUA_SWITCH_FILE	60662	UINT32	R/W

FILE_IO_LUA_SWITCH_FILE - Address: 60662

Write any value to this register to instruct Lua scripts to switch to a new file. Lua script should periodically check LJ.CheckFileFlag() to receive instruction, then call LJ.ClearFileFlag() after file switch is complete. Useful for applications that require continuous logging in a Lua script, and on-demand file access from a host.

- Data type: UINT32 (type index = 1)
- Readable and writable
- T7:

22.0 OEM Versions [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

The T-Series device variants can be ordered in an OEM form factor. The OEM versions of the T-Series devices are designed for companies or individuals to be able to easily customize the connection ports with pin-headers to build them into larger systems. The devices are sold without cases, cables, and other accessories, in order to reduce cost. The devices are also manufactured without screw terminals and the DB15 and DB37 connectors which allow them to be sold at a lower cost from the primary devices.

Ordering

For pricing/ordering, go to the main product pages and select the appropriate OEM variant:

- [T4 Product Page](#)
- [T7 Product Page](#)

Customization: Custom OEM boards carry additional cost, but they are often necessary for specialized enclosures, and seamless integration with other products. LabJack offers a device customization service that allows for -OEM devices to be ordered with custom parts installed prior to shipping. Through this service, LabJack acquires the parts from Digikey or Mouser (with a price mark-up) and performs the required through-hole soldering that needs to be performed to install the parts onto the PCB. LabJack also re-calibrates and tests the device after performing the modifications to ensure the devices are still working properly before shipping. For customization, please send us an email or [contact us](#)—we will most likely need to generate a custom quote and order to fulfill and bill against. Please don't hesitate to contact us.

Lead Time: Custom OEM board lead times can vary from 1-3 weeks depending on complexity, part availability, etc. Assuming that we don't keep stock of the custom parts required for a build we will order and ship components via UPS ground from a part distributor when the order gets placed. After receiving the parts it will take us an additional 1-2 weeks to install the components and run various device tests to ensure the custom OEM device(s) are in proper working condition before shipping the devices to the customer.

Pinouts: The OEM versions of the T4, T7, and T7-Pro are shown below: with the pinouts of the (T4) H2, H3, J2, J5 and (T7) J2, J3, J5 connectors:

T4-OEM:

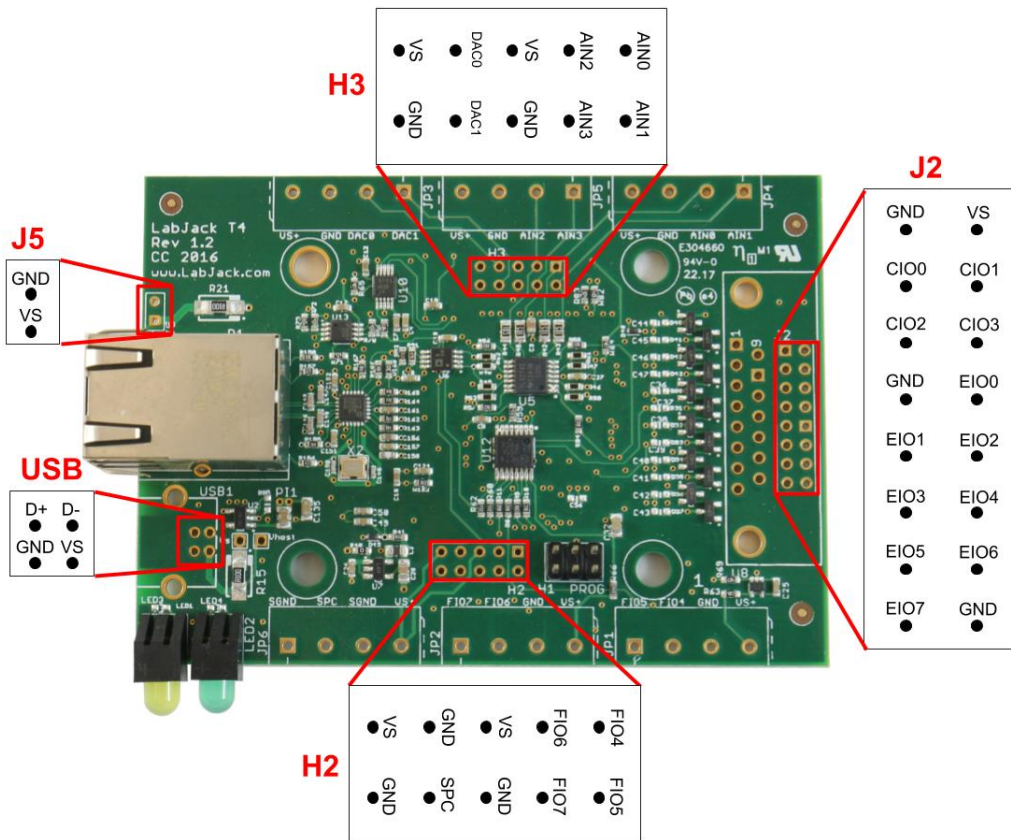


Image 22-1. T4-OEM Pinout

The T4-OEM exposes all of its I/O lines through the pin-headers H2, H3, and J2. The device can be externally powered with a regulated 5V supply using the J5 connector. If needed, a USB connector can also be installed, see the USB section below for more details. More details about the PCB dimensions for the T7 OEM versions can be found in Appendix B-2 T4 Enclosure and PCB Drawings.

T7-OEM and T7-Pro-OEM:

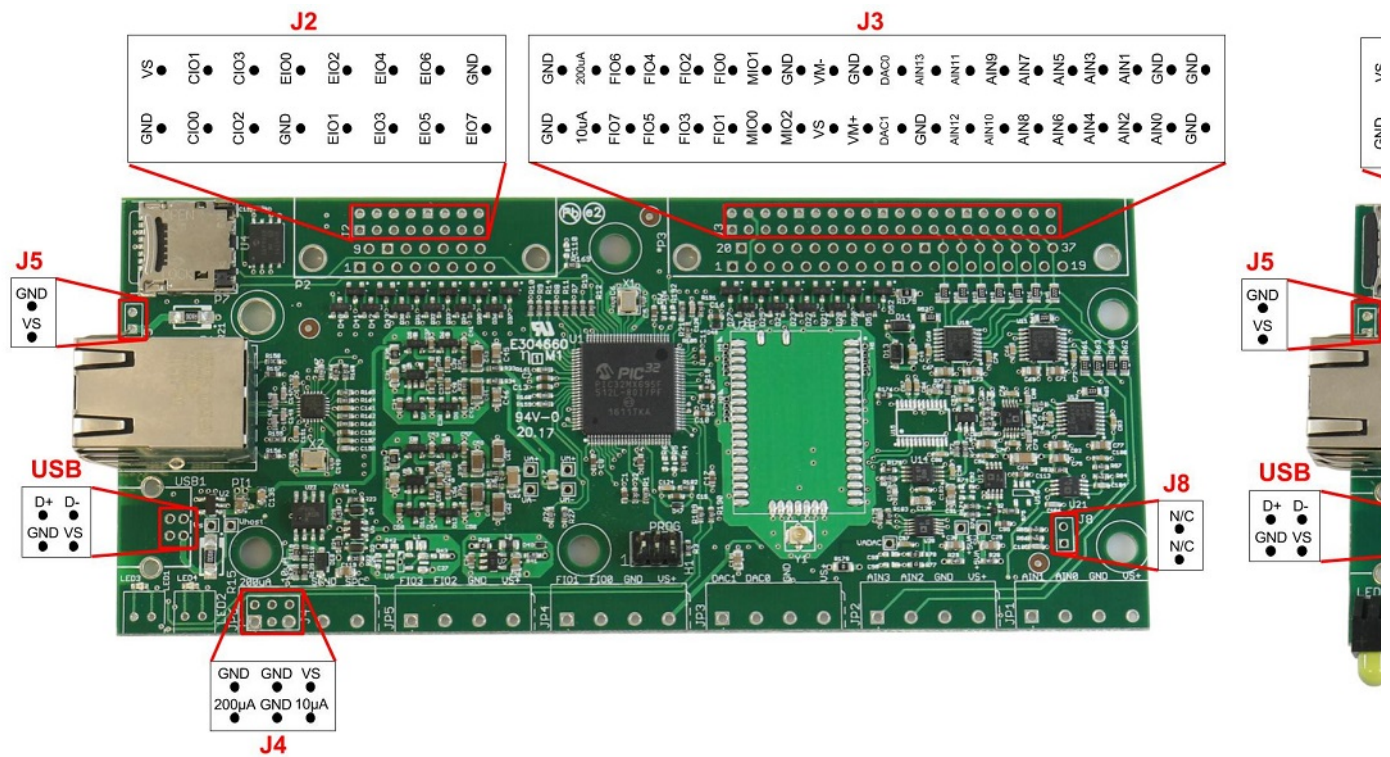


Image 22-2. T7-OEM Pinout

The T7-OEM and T7-Pro-OEM devices expose all of their I/O lines through the J2 and J3 pin-header locations. Both devices can be externally powered with below for more details. More details about the PCB dimensions for the T7 OEM versions can be found in [Appendix B-2 T7 Enclosure and PCB Drawings](#).

Suggested Part Orientations: We suggest that customers install pin headers on the top side of our devices to prevent there from being any issues with device re-calibration. During the calibration process, our OEM devices are mounted to the top of our test jigs (pogo pins make contact with the holes on the bottom side). Our device calibration test jigs use a combination of the screw terminal holes as well as the DB15 and DB37 holes to test each of the I/O lines.

Pin-Headers: The T4 and T7 OEM devices have pin-outs compatible with pin-headers that have a pin pitch of 0.100" (2.54mm). Below is a list of the standard pin-headers that we install into the J2, J3, J5, H2, and H3 locations. Other parts can be installed upon request including shrouded or directional pin-headers.

Name PinParts

J5	1x2 Jameco 1x2 2.54mm pin-header.
H2 and H3	2x5 Jameco 2x5 2.54mm pin-header.
J2	2x8 Jameco 2x8 2.54mm pin-header.
J3	2x20 Jameco 2x20 2.54mm pin-header.

Table 22-1. Pin-Headers

PCB Dimensions: The PCB dimensions and a variety of mechanical drawings can be found in the [Enclosure and PCB Drawings](#) section of the T-Series datasheet.

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

Part Categories: Optional parts that can be installed can be broken down into the following categories:

- [USB](#)
- [Alternate Power Supply \(J5\)](#)
- [DB15 and DB37 equivalent Pin-Header Locations \(J2, J3\)](#)
- [DB15/DB37 \(D-Sub\) Locations \(P2 and P3\)](#)
- [Screw Terminals \(JP1-JP6\)](#)
- [Ethernet Connector](#)
- [WiFi Antenna \(T7-Pro Only, Y1\)](#)
- [Current Sources \(T7 Only, J4\)](#)
- [Mechanical Header \(T7 Only, J8\)](#)

USB

The USB connector is not installed on any of the T-Series OEM devices. The T-Series devices use through-hole Type-B connectors and there are a large number of compatible connectors.

Mounting Location: The USB connector must be installed on the component side of the PCB.

Suggested Parts: We have several suggested USB connectors however most USB Type-B through hole connectors are compatible with the T-Series OEM devices. Standard retention USB connectors are what gets installed on all LabJack DAQ devices and are enough for most OEM device applications. High retention USB connectors are higher in cost however they more securely connect a USB cable to a device which is important in many industrial applications.

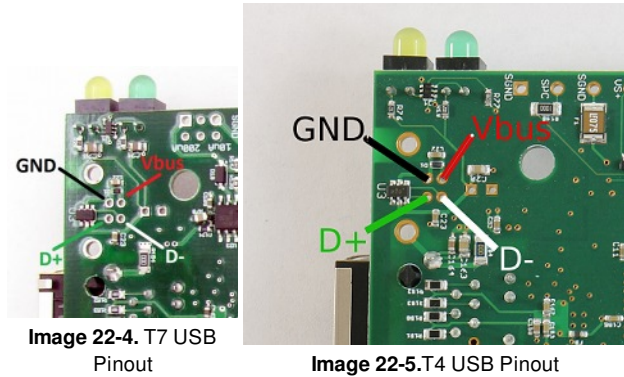
Standard Retention:

- [On Shore Technology Inc USB-B1HSW6](#)
- [FCI 61729-0010BLF](#)
- [TE Connectivity 292304-2](#)

High Retention:

- [Samtec USBR-B-S-S-O-TH](#)

USB Pin-Out: Below are two pictures of the bottom side of the T4 and T7 devices indicating the purpose of the four USB pins. Pin-outs relative to the top side are documented in the more general T-Series OEM device pinout pictures above.



USB Cables: A normal USB cable has a shield, and the normal Type-B connector connects the cable shield to the mounting tabs on the connector which are then soldered into the large USB mounting holes on the PCB. If you are not using a normal USB connector, and have a shield in your USB cable, we recommend that the shield be connected to either of the large USB mounting holes on the PCB. Usually the USB shield wires are aluminum, which doesn't take solder very well, so use a crimp connector like the [Molex 02-06-2103](#), [TE 61388-1](#), [TE 350015-2](#), or the [TE 60017-3](#). Secure the crimp connector to USB shield wires, then squish down the tip of the connector to fit into the large USB mounting holes on the PCB.

Alternate Power Supply (J5)

The T-Series OEM devices can be powered through either their USB connectors or through the J5 pin-header holes. The J5 pin-header is useful for individuals who only need Ethernet or WiFi device connections. The square shaped pad of the J5 pin-header is V+, and the circular pad is GND. 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](#).

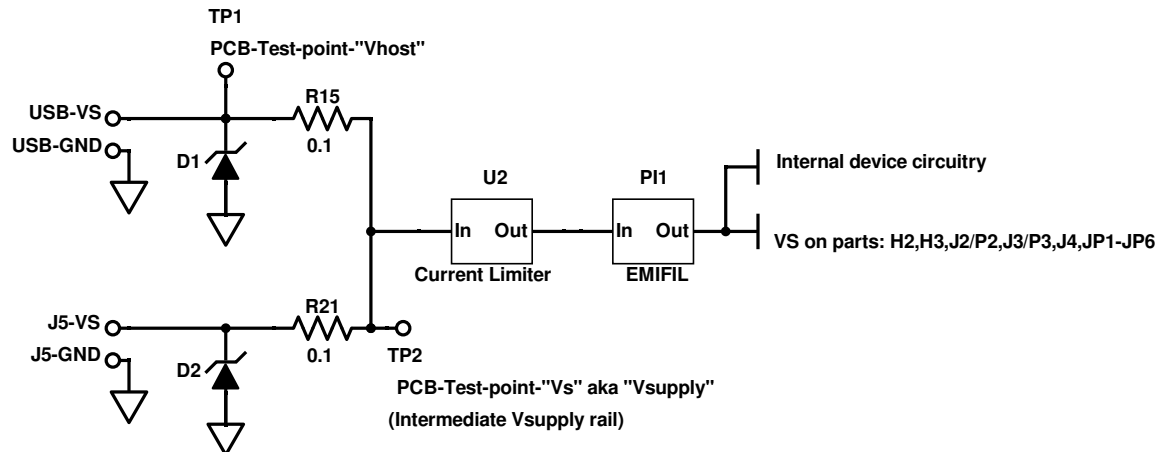


Image 22-6. T-Series Power Supply

Both the J5 pin-header and the USB voltage supply rails require a 5V input voltage (see the power supply input section of [Appendix-A-5](#) for detailed specs). The 5V supply from J5 (J5-VS) is protected by a TVS (transient voltage suppressor diode D2), then goes through R21 (0.1 ohms), and then connects to the internal Vsupply rail. The 5V supply from the USB (USB-VS) is exposed through a test point on the PCB labeled "Vhost" and is protected by the TVS diode D1 and R15 (0.1 ohms) before connecting to the internal Vsupply rail. The internal Vsupply rail can be measured using the test point on the PCB labeled "Vs". After joining, power flows through a current limiting chip (currently the Diodes AP2141WG) and an EMIFIL (EMI suppression filter) before connecting to the device-wide VS bus and out to each of the screw terminals labeled as VS.

Dual Power Supply: On all T-Series devices, R15 and R21 are installed by default and thus the connections for both sources are essentially shorted to each other. Both power supply options should not be connected to a voltage source at the same time. If this happens, one power supply could back-feed the other

which may cause damage. If a device is going to be powered using the J5 connector and there is a possibility of power at the USB connection (and USB power is not required) then it is recommended to remove R15 so that only the J5 pin-header is used to power the device. If a dual-power supply method is required it is recommended to replace the R15 and R21 resistors with Schottky diodes (SMA package). There will still be a small voltage drop that needs to be considered but the device should operate correctly as long as the voltage present on the VS screw terminals is above 4.75V (reference).

DB15 and DB37 equivalent Pin-Header Locations (J2, J3)

Connectors J2 & J3 provide pin-header alternatives to the [DB15](#) and [DB37](#) connectors. The J2 & J3 holes are always present, but are obstructed when the DB15 and DB37 are installed. The T4-OEM only has the J2 header while both the T7-OEM and T7-Pro-OEM have J2 and J3.

J2 & J3 can be seen in the pictures at the top of this page which show the component side of each of the T-Series device PCBs. For both the DBs and pin headers, holes with a square solder pad indicate pin number 1, 10, 20, or 30. For both the DBs and pin headers, pin 1 is at the lower-left. For the DB connectors the pin numbers increment from 1 left to right across the bottom row, and then continue left to right across the top row. For the pin headers, the odd pins increment left to right across the bottom row (1, 3, 5, ...) and the even pins increment left to right across the top row (2, 4, 6, ...).

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.

Table 22-2. J2 Pinouts

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

Table 22-3. J3 Pinouts

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	14	MIO1	15	MIO2
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

DB15/DB37 (D-Sub) Locations (P2 and P3)

The [DB15](#) and [DB37](#) connectors are not installed on OEM T-Series devices. 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 variant and specify custom parts that need to be installed. 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](#)
- [FCI D15S33E4GV00LF](#)
- [Sullins Connector Solutions SDS101-PRW2-F15-SN13-1](#)

- [FCI 10090099-S374VLF](#)
- [FCI D37S33E4GV00LF](#)
- [Sullins Connector Solutions SDS101-PRW2-F37-SN83-6](#)

Screw Terminals (JP1-JP6)

The screw terminals are not installed on the T-Series OEM device variants. 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 variant and specify custom parts that need to be installed. 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.

Ethernet Connector

The same Ethernet connector is installed on all T-Series device variations 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 Antenna (T7-Pro Only, Y1)

The T7-Pro-OEM ships with a simple 30mm U.FL whip antenna such as the [Anaren 66089-2406](#). See "Antenna Details" in the [WiFi section](#) of this datasheet for additional information.

Current Sources (T7 Only, J4)

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.

1	200 μ A
2	GND
3	10 μ A
4	GND
5	GND
6	VS

Table 22-
4.J4
Pinout

Mechanical Header (T7 Only, J8)

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

23.0 Watchdog [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Overview

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

A typical usage is to first use the [IO Config](#) system to set the power-up defaults for everything as desired, and then configure the watchdog to reset the device on timeout. For example, consider a software program that enables the watchdog to reset the T-Series device with a 60 second timeout, and then the software 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 T-Series device will reset every 60 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—see [11.0 SPC](#) for details.

The timeout period is reset when a response to a command-response packet is sent to the host. Alternatively, "strict" mode can be enabled. When strict mode is

enabled the timeout period is reset by writing the key value to WATCHDOG_STRICT_CLEAR. The key can be set by writing WATCHDOG_STRICT_KEY_DEFAULT. While strict is disabled writing any value to WATCHDOG_STRICT_CLEAR will clear the watchdog. Note that writing to WATCHDOG_STRICT_CLEAR will clear the watchdog when the write is processed, not when a response packet is sent.

Normal spontaneous stream data does not reset the watchdog timeout. The timeout period is reset when a response to a command-response packet is sent to the host, so if streaming you must do some periodic command-response communication to reset the watchdog timeout.

Register Listing

Use the following registers to configure the watchdog system:

Watchdog Registers

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

WATCHDOG_ENABLE_DEFAULT - Address: 61600

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).

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_ADVANCED_DEFAULT - Address: 61602

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.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_TIMEOUT_S_DEFAULT - Address: 61604

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.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_STARTUP_DELAY_S_DEFAULT - Address: 61606

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.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_STRICT_ENABLE_DEFAULT - Address: 61610

Set to 1 to enable strict mode.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_STRICT_KEY_DEFAULT - Address: 61612

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_STRICT_CLEAR - Address: 61614

When running in strict mode, writing the key to this register is the only way to clear the watchdog. Writing to this register while not using strict mode will clear the watchdog.

- Data type: UINT32 (type index = 1)
- Write-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_RESET_ENABLE_DEFAULT - Address: 61620

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DIO_ENABLE_DEFAULT - Address: 61630

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DIO_STATE_DEFAULT - Address: 61632

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DIO_DIRECTION_DEFAULT - Address: 61634

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DIO_INHIBIT_DEFAULT - Address: 61636

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DAC0_ENABLE_DEFAULT - Address: 61640

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DAC0_DEFAULT - Address: 61642

The voltage of DAC0 after a Watchdog timeout.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DAC1_ENABLE_DEFAULT - Address: 61650

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

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

WATCHDOG_DAC1_DEFAULT - Address: 61652

The voltage of DAC1 after a Watchdog timeout.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0016

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. In conjunction, you would often use the [IO Config](#) system to configure the power-up defaults as desired.

24.0 IO Config, _DEFAULT [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

IO_Config Overview

IO_Config controls the default configuration that will be used when the device boots up, and can set an already running device to a known state. The default configuration sets the values that will be used during boot-up for digital directions, digital states, DACs, AIN_EF, and DIO_EF. IO_Config can also apply either the default or factory configuration to a device that is already running.

Configuration

_DEFAULT

Registers ending with "_DEFAULT" will store non-volatile settings. These non-volatile settings will be used to configure the T-Series device during boot-up.

Terms

- **DEFAULT** – A saved configuration set. These are also the settings that will be used when the device boots up.
- **FACTORY** – The default settings that are loaded when the device is tested at LabJack.
- **CURRENT** – The device's current settings.

Register Listing

Use the following registers to configure IO_Config:

Name	Start Address	Type	Access
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 - Address: 49002

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. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

- Data type: UINT32 (type index = 1)
- Write-only

IO_CONFIG_SET_DEFAULT_TO_FACTORY - Address: 49004

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. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

- Data type: UINT32 (type index = 1)
- Write-only

IO_CONFIG_SET_CURRENT_TO_FACTORY - Address: 61990

Write a 1 to set current values to factory configuration. The factory values are retrieved from flash and written to the current configuration registers. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

- Data type: UINT16 (type index = 0)
- Write-only

IO_CONFIG_SET_CURRENT_TO_DEFAULT - Address: 61991

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. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

- Data type: UINT16 (type index = 0)
- Write-only

Checksum

The IO_CONFIG_CURRENT_CRC32 register returns a CRC32 of the configuration data:

Name	Start Address	Type	Access
IO_CONFIG_CURRENT_CRC32	49020	UINT32	R

IO_CONFIG_CURRENT_CRC32 - Address: 49020

Collects the current IO configuration and calculates of a CRC32.

- Data type: UINT32 (type index = 1)

- Read-only
- T7:
 - Minimum [firmware](#) version: 1.0220

The CRC can be used to detect configuration changes. Calculating the checksum takes between 5 and 15 ms. Some configurations are excluded from the CRC because they are expected to change during normal operations. The excluded configurations are:

- Digital States
- Digital Directions
- DAC Voltages

Factory Reset

IO_Config settings can be cleared by a factory reset. See the [11.0 SPC](#) section for more information.

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.
```

24.1 Cleanse (T7 Only) [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T7 Series Only

Overview

The Cleanse function will reset non-volatile user data and settings to the factory defaults.

To prevent errors, the WiFi module will be disabled.

Requires firmware 1.0225.

Triggering a Cleanse

To trigger a cleanse, write 0x5317052E to the CLEANSE register:

Name	Start Address	Type	Access
CLEANSE	49090	UINT32	W

CLEANSE - Address: 49090

Writing 0x5317052E to this register will trigger a system cleanse.

- Data type: UINT32 (type index = 1)
- Write-only

The LEDs will blink in an alternating pattern to show that the cleanse is in progress. If the LEDs blink in unison, then an error has occurred.

The CLEANSE function will take a few seconds to complete. This will cause LJM to throw a LJME_RECONNECT_FAILED error (#1239). To prevent this error, [LJM's timeout](#) needs to be increased. Alternatively, LJME_RECONNECT_FAILED errors can be ignored.

Once successfully cleared, the items listed below will be cleared.

Items Cleared:

- Device Name
- AIN_EF Settings
- DIO_EF Settings
- DIO States and Directions
- AIN Settings
- Watchdog Timer
- Lua Startup Settings
- Lua Startup Script
- Flash User Space
- Battery Backed RAM (Pro only)
- Ethernet Settings
- Some WiFi Settings

Not Cleared:

- Settings stored in the WiFi module are not erased. Settings in the WiFi module are SSID and Password. To clear these settings, use Kipling to write dummy values.

25.0 Lua Scripting [T-Series Datasheet]

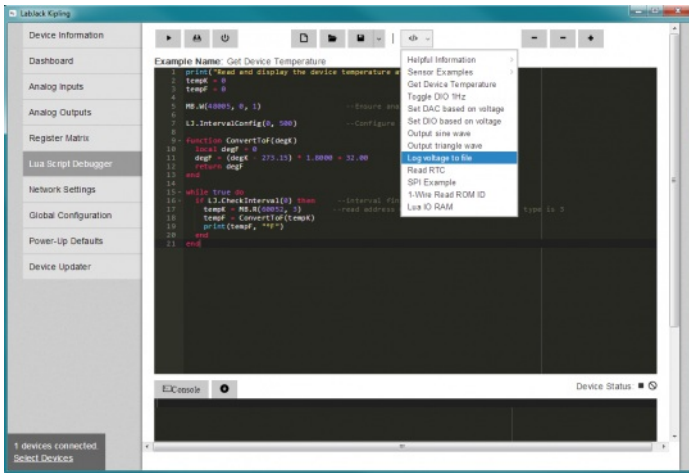
[Log in](#) or [register](#) to post comments

Lua Scripting Overview

T-Series devices can execute Lua code to allow independent operation. A Lua script can be used to collect data without a host computer or to perform complex tasks producing simple results that a host can read. For a good overview on the capabilities of scripting, see this [LabJack Lua blog post](#).

Getting Started

1. Connect your device to your computer, launch [Kipling v3](#), and navigate to the Lua Script Debugger tab.



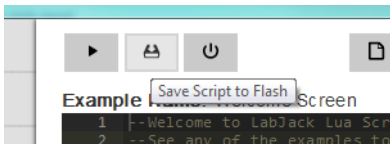
2. Open the "Get Device Temperature" example and click the Run button. Now click the Stop button. Clicking Stop clears the Lua VM, so even if a program has concluded, it is still necessary to press the Stop button.

3. Try out some other examples.

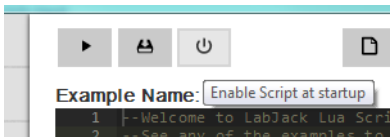
Running a script when the device powers up

A T-Series device can be configured to run a script when it powers on or resets. Typically you should test script for a while with the Run/Stop buttons while viewing the debug output in the console. Then once everything is working correctly, enable the script at startup and close Kipling.

To enable the script at startup:



1. Click Save Script to Flash.



2. Click Enable Script at Startup. Now when the device is powered on or reset, it will run your script.

3. After power cycling the device, if it becomes un-usable and the COMM/Status lights are constantly blinking the Lua Script is likely causing the device to enter an invalid state. The device can be fixed by connecting a jumper wire between the SPC terminal and either FIO0 or FIO4, see the [SPC](#) section of the datasheet for more details.

Learning more about Lua

Learning Lua is very easy. There are good tutorials on [Lua.org](#) as well as on 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.

Lua for T-Series Devices

Try to keep names short. String length directly affects execution speed and code size.

Use local variables instead of global variables (it's faster and less error-prone). For example, use:

```
local a = 10
```

instead of:

```
a = 10
```

You can also assign a function to a local for further optimization:

```
local locW = MB.W  
locW(2003, 0, 1) --Write to address 2003
```

Lua supports multi-return:

```
table, error = MB.RA(Address, dataType, nValues)
```

Both table and error are returned values.

On T-Series devices, Lua's only numeric data type is IEEE 754 single precision (float). This is more important than it sounds. Here is a good article on floating point numbers and their pitfalls: [Floating Point Numbers](#). There are some workarounds for reading values greater than 24-bits in the "Lua Limitations" section.

Examples contain comments, and currently comments consume a lot of code space. You may want to limit comments if you are making a long script (>200 lines).

LabJack's Lua library

Based on eLua 0.8 which is based on Lua 5.1.4.

Data Types - Same types as the [LJM Library](#)

- **0** - unsigned 16-bit integer
- **1** - unsigned 32-bit integer
- **2** - signed 32-bit integer
- **3** - single precision floating point (float)
- **98** - string
- **99** - byte - The "byte" dataType is used to pass arrays of bytes in what Lua calls tables

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

MB.R

```
value, error = MB.R(Address, dataType)
```

Modbus read. Reads a single value from a Modbus register. The type can be a u16, u32, a float, or a string. Any errors encountered will be returned in error.

MB.W

```
error = MB.W(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. Any errors encountered will be returned in error.

MB.WA

```
error = MB.WA(Address, dataType, nValues, table)
```

Modbus write array. Reads `nValues` from the supplied table, interprets them according to the `dataType` and writes them as an array to the register specified by `Address`. The table must be indexed with numbers from 1 to `nValues`.

MB.RA


```
table, error = MB.RA(Address, dataType, nValues)
```

Modbus read array. Reads `nValues` of type `dataType` from `Address` and returns the results in a Lua table. The table is indexed from 1 to `nValues`.

LJ.ledtog

```
LJ.ledtog() --Toggles status LED. Note that reading AINs also toggles the status LED.
```

LJ.Tick

```
Ticks = LJ.Tick() --Reads the core timer (1/2 core frequency).
```

LJ.DIO_D_W

```
LJ.DIO_D_W(3, 1) --Quickly change FIO3 direction _D_ to output.
```

LJ.DIO_S_W

```
LJ.DIO_S_W(3, 0) --Quickly change the state _S_ of FIO3 to 0 (output low)
```

LJ.DIO_S_R

```
state = LJ.DIO_S_R(3) -- Quickly read the state _S_ of FIO3
```

LJ.CheckFileFlag

```
flag = LJ.CheckFileFlag() and LJ.ClearFileFlag
```

`LJ.CheckFileFlag` and `LJ.ClearFileFlag` work together to provide an easy way to tell a Lua script to switch files. This is useful for applications that require continuous logging in a Lua script and on-demand file access from a host. Since files cannot be opened simultaneously by a Lua script and a host, the Lua script must first close the active file if the host wants to read file contents. The host writes a value of 1 to `FILE_IO_LUA_SWITCH_FILE`, and the Lua script is setup to poll this parameter using `LJ.CheckFileFlag()`. If the file flag is set, Lua code should switch files:

Example:

```
fg = LJ.CheckFileFlag() --poll the flag every few seconds
if fg == 1 then
  NumFn = NumFn + 1          --increment filename
  Filename = Filepre..string.format("%02d", NumFn)..Filesuf
  f:close()
  LJ.ClearFileFlag()        --inform host that previous file is available.
  f = io.open(Filename, "w") --create or replace a new file
  print ("Command issued by host to create new file")
end
```

LJ.IntervalConfig & LJ.CheckInterval

`LJ.IntervalConfig` and `LJ.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)
```

Sets an interval timer, starting from the current time.

`handle` : 0-7. Identifies an interval.

`time_ms` : Number of milliseconds per interval.

```
timed_out = LJ.CheckInterval(handle)
```

handle : 0-7. Identifies an interval.

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.setLuaThrottle

```
LJ.setLuaThrottle(newThrottle)
```

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.

LJ.getLuaThrottle

```
ThrottleSetting = LJ.getLuaThrottle()
```

Reads the current throttle setting

Passing data into/out of Lua

User RAM consists of a list of Modbus addresses where data can be sent to and read from a Lua script. Lua writes to the Modbus registers, and then a host device can read that information.

There are a total of 200 registers of pre-allocated RAM, which is split into several groups so that users may access it conveniently with different data types.

Use the following USER_RAM registers to store information:

User RAM Registers			
Name	Start Address	Type	Access
USER_RAM#(0:39)_F32	46000	FLOAT32	R/W
USER_RAM#(0:9)_I32	46080	INT32	R/W
USER_RAM#(0:39)_U32	46100	UINT32	R/W
USER_RAM#(0:19)_U16	46180	UINT16	R/W

USER_RAM#(0:39)_F32 - Starting Address: 46000

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 1.0023

Expanded Names	Addresses
USER_RAM0_F32, USER_RAM1_F32, USER_RAM2_F32, USER_RAM3_F32, USER_RAM4_F32, USER_RAM5_F32, USER_RAM6_F32, USER_RAM7_F32, USER_RAM8_F32, USER_RAM9_F32, USER_RAM10_F32, USER_RAM11_F32, USER_RAM12_F32, USER_RAM13_F32, USER_RAM14_F32, USER_RAM15_F32, USER_RAM16_F32, USER_RAM17_F32, USER_RAM18_F32, USER_RAM19_F32, USER_RAM20_F32, USER_RAM21_F32, USER_RAM22_F32, USER_RAM23_F32, USER_RAM24_F32, USER_RAM25_F32, USER_RAM26_F32, USER_RAM27_F32, USER_RAM28_F32, USER_RAM29_F32, USER_RAM30_F32, USER_RAM31_F32, USER_RAM32_F32, USER_RAM33_F32, USER_RAM34_F32, USER_RAM35_F32, USER_RAM36_F32, USER_RAM37_F32, USER_RAM38_F32, USER_RAM39_F32 Show All	46000, 46002, 46004, 46006, 46008, 46010, 46012, 46014, 46016, 46018, 46020, 46022, 46024, 46026, 46028, 46030, 46032, 46034, 46036, 46038, 46040, 46042, 46044, 46046, 46048, 46050, 46052, 46054, 46056, 46058, 46060, 46062, 46064, 46066, 46068, 46070, 46072, 46074, 46076, 46078 Show All

USER_RAM#(0:9)_I32 - Starting Address: 46080

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

- Data type: INT32 (type index = 2)
- Readable and writable
- T7:
 - Minimum [firmware](#) version: 1.0162

Expanded Names	Addresses
USER_RAM0_I32, USER_RAM1_I32, USER_RAM2_I32, USER_RAM3_I32, USER_RAM4_I32, USER_RAM5_I32, USER_RAM6_I32, USER_RAM7_I32, USER_RAM8_I32, USER_RAM9_I32 Show All	46080, 46082, 46084, 46086, 46088, 46090, 46092, 46094, 46096, 46098 Show All

USER_RAM#(0:39)_U32 - Starting Address: 46100

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

- Data type: UINT32 (type index = 1)
- Readable and writable
- T7:

- Minimum [firmware](#) version: 1.0162

Expanded Names	Addresses
USER_RAM0_U32, USER_RAM1_U32, USER_RAM2_U32, USER_RAM3_U32, USER_RAM4_U32, USER_RAM5_U32, USER_RAM6_U32, USER_RAM7_U32, USER_RAM8_U32, USER_RAM9_U32, USER_RAM10_U32, USER_RAM11_U32, USER_RAM12_U32, USER_RAM13_U32, USER_RAM14_U32, USER_RAM15_U32, USER_RAM16_U32, USER_RAM17_U32, USER_RAM18_U32, USER_RAM19_U32, USER_RAM20_U32, USER_RAM21_U32, USER_RAM22_U32, USER_RAM23_U32, USER_RAM24_U32, USER_RAM25_U32, USER_RAM26_U32, USER_RAM27_U32, USER_RAM28_U32, USER_RAM29_U32, USER_RAM30_U32, USER_RAM31_U32, USER_RAM32_U32, USER_RAM33_U32, USER_RAM34_U32, USER_RAM35_U32, USER_RAM36_U32, USER_RAM37_U32, USER_RAM38_U32, USER_RAM39_U32 Show All	46100, 46102, 46104, 46106, 46108, 46110, 46112, 46114, 46116, 46118, 46120, 46122, 46124, 46126, 46128, 46130, 46132, 46134, 46136, 46138, 46140, 46142, 46144, 46146, 46148, 46150, 46152, 46154, 46156, 46158, 46160, 46162, 46164, 46166, 46168, 46170, 46172, 46174, 46176, 46178 Show All

USER_RAM#(0:19)_U16 - Starting Address: 46180

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

- Data type: UINT16 (type index = 0)
- Readable and writable
- T7:

- Minimum [firmware](#) version: 1.0162

Expanded Names	Addresses
USER_RAM0_U16, USER_RAM1_U16, USER_RAM2_U16, USER_RAM3_U16, USER_RAM4_U16, USER_RAM5_U16, USER_RAM6_U16, USER_RAM7_U16, USER_RAM8_U16, USER_RAM9_U16, USER_RAM10_U16, USER_RAM11_U16, USER_RAM12_U16, USER_RAM13_U16, USER_RAM14_U16, USER_RAM15_U16, USER_RAM16_U16, USER_RAM17_U16, USER_RAM18_U16, USER_RAM19_U16 Show All	46180, 46181, 46182, 46183, 46184, 46185, 46186, 46187, 46188, 46189, 46190, 46191, 46192, 46193, 46194, 46195, 46196, 46197, 46198, 46199 Show All

USER_RAM Example script:

```
while true do
  if LJ.CheckInterval(0) then
    Enable = MB.R(46000, 3) --host may disable portion of the script
    if Enable >= 1 then
      val = val + 1
      print("New value:", val)
      MB.W(46002, 3, val) --provide a new value to host
    end
  end
end
```

There is also a more advanced system for passing data to/from a Lua script referred to as FIFO buffers. These buffers are useful if you want to send an array of information in sequence to/from a Lua script. Usually 2 buffers are used for each endpoint, one buffer dedicated for each communication direction (read and write). A host may write new data for the Lua script into FIFO0, then once the script reads the data out of that buffer, it responds by writing data into FIFO1, and then the host may read the data out of FIFO1.

User RAM FIFO Registers - Advanced

Name	Start Address	Type	Access
USER_RAM_FIFO#(0:3)_DATA_U16	47000	UINT16	R/W
USER_RAM_FIFO#(0:3)_DATA_U32	47010	UINT32	R/W
USER_RAM_FIFO#(0:3)_DATA_I32	47020	INT32	R/W
USER_RAM_FIFO#(0:3)_DATA_F32	47030	FLOAT32	R/W
USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES	47900	UINT32	R/W
USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO	47910	UINT32	R
USER_RAM_FIFO#(0:3)_EMPTY	47930	UINT32	W

USER_RAM_FIFO#(0:3)_DATA_U16 - Starting Address: 47000

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it. This register is a buffer. Underrun behavior - throws an error.

- Data type: UINT16 (type index = 0)
- Readable and writable
- Default value: 0
- This register is a [Buffer Register](#)
- T7:

- Minimum [firmware](#) version: 1.0163

Expanded Names	Addresses
USER_RAM_FIFO0_DATA_U16, USER_RAM_FIFO1_DATA_U16, USER_RAM_FIFO2_DATA_U16, USER_RAM_FIFO3_DATA_U16 Show All	47000, 47001, 47002, 47003 Show All

USER_RAM_FIFO#(0:3)_DATA_U32 - Starting Address: 47010

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it. This register is a buffer. Underrun behavior - throws an error.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- This register is a [Buffer Register](#)
- T7:

- Minimum [firmware](#) version: 1.0163

Expanded Names	Addresses
USER_RAM_FIFO0_DATA_U32, USER_RAM_FIFO1_DATA_U32, USER_RAM_FIFO2_DATA_U32, USER_RAM_FIFO3_DATA_U32 Show All	47010, 47012, 47014, 47016 Show All

USER_RAM_FIFO#(0:3)_DATA_I32 - Starting Address: 47020

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it. This register is a buffer. Underrun behavior - throws an error.

- Data type: INT32 (type index = 2)
- Readable and writable
- Default value: 0
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 1.0163

Expanded Names	Addresses
USER_RAM_FIFO0_DATA_I32, USER_RAM_FIFO1_DATA_I32, USER_RAM_FIFO2_DATA_I32, USER_RAM_FIFO3_DATA_I32 Show All	47020, 47022, 47024, 47026 Show All

USER_RAM_FIFO#(0:3)_DATA_F32 - Starting Address: 47030

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it. This register is a buffer. Underrun behavior - throws an error.

- Data type: FLOAT32 (type index = 3)
- Readable and writable
- Default value: 0
- This register is a [Buffer Register](#)
- T7:
 - Minimum [firmware](#) version: 1.0163

Expanded Names	Addresses
USER_RAM_FIFO0_DATA_F32, USER_RAM_FIFO1_DATA_F32, USER_RAM_FIFO2_DATA_F32, USER_RAM_FIFO3_DATA_F32 Show All	47030, 47032, 47034, 47036 Show All

USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES - Starting Address: 47900

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

- Data type: UINT32 (type index = 1)
- Readable and writable
- Default value: 0
- This register uses system RAM. The maximum RAM is 64KB. For more information, see [4.4 RAM](#)
- T7:
 - Minimum [firmware](#) version: 1.0163

Expanded Names	Addresses
USER_RAM_FIFO0_ALLOCATE_NUM_BYTES, USER_RAM_FIFO1_ALLOCATE_NUM_BYTES, USER_RAM_FIFO2_ALLOCATE_NUM_BYTES, USER_RAM_FIFO3_ALLOCATE_NUM_BYTES Show All	47900, 47902, 47904, 47906 Show All

USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO - Starting Address: 47910

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value. At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

- Data type: UINT32 (type index = 1)
- Read-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0163

Expanded Names	Addresses
USER_RAM_FIFO0_NUM_BYTES_IN_FIFO, USER_RAM_FIFO1_NUM_BYTES_IN_FIFO, USER_RAM_FIFO2_NUM_BYTES_IN_FIFO, USER_RAM_FIFO3_NUM_BYTES_IN_FIFO Show All	47910, 47912, 47914, 47916 Show All

USER_RAM_FIFO#(0:3)_EMPTY - Starting Address: 47930

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

- Data type: UINT32 (type index = 1)
- Write-only
- Default value: 0
- T7:
 - Minimum [firmware](#) version: 1.0163

Expanded Names	Addresses
USER_RAM_FIFO0_EMPTY, USER_RAM_FIFO1_EMPTY, USER_RAM_FIFO2_EMPTY, USER_RAM_FIFO3_EMPTY Show All	47930, 47932, 47934, 47936 Show All

USER_RAM_FIFO Example script:

```
aF32_Out= {} --array of 5 values(floats)
aF32_Out[1] = 10.0
aF32_Out[2] = 20.1
aF32_Out[3] = 30.2
aF32_Out[4] = 40.3
aF32_Out[5] = 50.4

aF32_In = {}
numValuesFIFO0 = 5
ValueSizeInBytes = 4
numBytesAllocFIFO0 = numValuesFIFO0*ValueSizeInBytes
MB.W(47900, 1, numBytesAllocFIFO0) --allocate USER_RAM_FIFO0_NUM_BYTES_IN_FIFO to 20 bytes
```

```
LJ.IntervalConfig(0, 2000)
while true do
  if LJ.CheckInterval(0) then
    --write out to the host with FIFO0
    for i=1, numValuesFIFO0 do
      ValOutOfLua = aF32_Out[i]
```

```

numBytesFIFO0 = MB.R(47910, 1)
if (numBytesFIFO0 < numBytesAllocFIFO0) then
  MB.W(47030, 3, ValOutOfLua) --provide a new array to host
  print ("Next Value FIFO: ", ValOutOfLua)
else
  print ("FIFO0 buffer is full.")
end
end
--read in new data from the host with FIFO1
--Note that an external computer must have previously written to FIFO1
numBytesFIFO1 = MB.R(47912, 1) --USER_RAM_FIFO1_NUM_BYTES_IN_FIFO
if (numBytesFIFO1 == 0) then
  print ("FIFO1 buffer is empty.")
end
for i=1, ((numBytesFIFO1+1)/ValueSizeInBytes) do
  ValIntoLua = MB.R(47032, 3)
  aF32_In[i] = ValIntoLua
  print ("Next Value FIFO1: ", ValIntoLua)
end
end
end
end

```

Lua Limitations

Lua is using a single precision float for its data-type. This means that working with 32-bit integer registers is difficult (see examples below). If any integer exceeds 24-bits, the lower bits will be lost. The workaround is to access the Modbus register using two numbers, each 16-bits. Lua can specify the data type for the register being written, so if users are expecting a large number that will not fit in a float(>24bits), such as a MAC, then read or write the value as a series of 16-bit integers. If you expect the value to be counting up or down, use `MB.RA` or `MB.RW` to access the U32 as a contiguous set of 4 bytes. If the value isn't going to increment (e.g. the MAC address) it is permissible to read it in two separate packets using `MB.R`.

Read a 32-bit register

```

--Value is expected to be changing, and >24 bits (use MB.RA)
aU32[1] = 0x00
aU32[2] = 0x00
aU32, error = MB.RA(3000, 0, 2) --DIO0_EF_READ_A. Type is 0 instead of 1
DIO0_EF_READ_A_MSW = aU32[1]
DIO0_EF_READ_A_LSW = aU32[2]
--Value constant, and >16,777,216 (24 bits)
--Read ETHERNET_MAC (address 60020)
MAC_MSW = MB.R(60020, 0) --Read upper 16 bits. Type is 0 instead of 1
MAC_LSW = MB.R(60021, 0) --Read lower 16 bits.
--Value <16,777,216 (24 bits)
--Read AIN0_EF_INDEX (address 9000)
AIN0_index = MB.R(9000, 1) --Type can be 1, since the value will be smaller than 24 bits.

```

Write a 32-bit register

```

--Value might be changed or incremented by the T7, and >24 bits (use MB.WA)
aU32[1] = 0xFF2A
aU32[2] = 0xFB5F
error = MB.WA(44300, 0, 2, aU32) --Write DIO0_EF_VALUE_A. Type is 0 instead of 1
--Value constant, and >24 bits
MB.W(44300, 0, 0xFF2A) --Write upper 16 bits. Type is 0 instead of 1
MB.W(44301, 0, 0xFB5F) --Write lower 16 bits.
--Value <16,777,216 (24 bits)
--Write DIO0_EF_INDEX (address 44100)
MB.W(44100, 1, 7) --Type can be 1, since the value(7) is smaller than 24 bits.

```

Loading a Lua Script to a T7 Manually

Load Lua Script Manually To Device

While LabJack's [Kipling](#) program [handles Lua scripting](#) details easily and automatically, the example below shows how to load a Lua script to a T7 manually. The general process as well as some example psuedo code can be found below:

1. Define or load a Lua Script and make sure a device has been opened.
2. Make sure there is a null-character at the end of the string.
3. Make sure a Lua Script is not currently running. If one is, stop it and wait for it to be stopped.
4. Write to the "LUA_SOURCE_SIZE" and "LUA_SOURCE_WRITE" registers to instruct the T7 to allocate space for a script and to transfer it to the device.
5. (Optional) Enable debugging.
6. Instruct the T-Series device to run the loaded Lua Script.

The C example below opens a T7, shuts down any Lua script that may be running, loads the script, and runs the script.

```
const char * luaScript =
```

```
"LJ.IntervalConfig(0, 500)\n"while true do\n" if LJ.CheckInterval(0) then\n" print(LJ.Tick())\n" end\n"end\n"\0";
```

```
const unsigned scriptLength = strlen(luaScript) + 1;\n// strlen does not include the null-terminating character, so we add 1\n// byte to include it.
```

```
int handle = OpenOrDie(LJM_dtT7, LJM_ctANY, "LJM_idANY");
```

```
// Disable a running script by writing 0 to LUA_RUN twice\nWriteNameOrDie(handle, "LUA_RUN", 0);\n// Wait for the Lua VM to shut down (and some T7 firmware versions need\n// a longer time to shut down than others):\nMillisecondSleep(600);\nWriteNameOrDie(handle, "LUA_RUN", 0);
```

```
// Write the size and the Lua Script to the device\nWriteNameOrDie(handle, "LUA_SOURCE_SIZE", scriptLength);\nWriteNameByteArrayOrDie(handle, "LUA_SOURCE_WRITE", scriptLength, luaScript);
```

```
// Start the script with debug output enabled\nWriteNameOrDie(handle, "LUA_DEBUG_ENABLE", 1);\nWriteNameOrDie(handle, "LUA_DEBUG_ENABLE_DEFAULT", 1);\nWriteNameOrDie(handle, "LUA_RUN", 1);
```

The above example is valid C code, where the following functions are error-handling functions that cause the program to exit if an error occurs:

- `OpenOrDie` wraps `LJM_Open`
- `WriteNameOrDie` wraps `LJM_eWriteName`
- `WriteNameByteArrayOrDie` wraps `LJM_eWriteNameByteArray`

The Lua script in the example above is in the form of a C-string, e.g. a string with a null-terminator as the last byte.

To download a version of the above example, see [utilities/lua_script_basic.c](#), which includes a function that reads debug data from the T7.

Reading Debug Data/Print Statements

After a script has started (with debugging enabled) any information printed by a lua script can be read by a user application using the "LUA_DEBUG_NUM_BYTES" and "LUA_DEBUG_DATA" registers.

25.1 I2C Library [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

I2C Library Overview

The I2C library abstracts most of the Modbus calls needed to run I2C. The abstraction allows users to focus on I2C rather than Modbus, and reduces the memory requirements of scripts. Several I2C examples can be found on the [I2C Sensor Examples](#) page. Before using the I2C library on a T7 make sure the device is running at least FW v1.0225 or greater. See the [Kipling Device Updater](#) tab for more details on how to update a T-Series device.

I2C.config

```
Error = I2C.config(SDA, SCL, Speed, Options, Address)
```

Sets parameters that are not normally changed. Values set by this function will remain unchanged until this function is called again or the equivalent Modbus registers are written to.

Parameters:

- `SDA` - DIO pin # that will be used as the I2C data line
- `SCL` - DIO pin # that will be used as the I2C clock line
- `Speed` - See I2C documentation
- `Options` - See I2C documentation
- `Address` - Left Justified

Returns:

- `Error` - standard LabJack T-Series error codes.

I2C.writeRead

```
RxData, Error = I2C.writeRead(TxData, NumToRead)
```

This function will first write the data in `TxDATA` to the preset address, then will read `NumToRead` bytes from that same address.

Parameters:

- `TxDATA` - This is a Lua table containing the values to be transmitted. The size of the table determines the number of bytes that will be transmitted.
- `NumToRead` - The number of data bytes to be read.

Returns:

- `RxDATA` - A Lua table of the values read
- `Error` - standard LabJack T-Series error codes.

I2C.read

```
RxDATA, Error = I2C.read(NumToRead)
```

This function will read `NumToRead` bytes from the preset address.

Parameters:

- `NumToRead` - The number of data bytes to be read.

Returns:

- `RxDATA` - A Lua table of the values read
- `Error` - standard LabJack T-Series error codes.

I2C.write

```
Error = I2C.write(TxDATA)
```

This function will write the data in `TxDATA` to the preset address.

Parameters:

- `TxDATA` - This is a Lua table containing the values to be transmitted. The size of the table determines the number of bytes that will be transmitted.

Returns:

- `Error` - standard LabJack T-Series error codes.

I2C.search

```
AddressList, Error = I2C.search(FirstAddress, LastAddress)
```

This function will scan the I2C bus addresses are acknowledged. An acknowledged address means that at least one device is set to that address. Addresses are tested sequentially between the first and last address parameters.

Parameters:

- `FirstAddress` - The first address to be tested.
- `LastAddress` - The last address to be tested.

Returns:

- `AddressList` - A Lua table containing all the addresses that responded.
- `Error` - standard LabJack T-Series error codes.

Appendix A - Specifications [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Appendix A - Specifications Overview

Specifications for describing the T-Series devices can be broken down into several primary sections with a few sub-sections. Navigate the following sections to see specifications.

A-1 Data Rates [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Communication Modes

Communication between the host computer and a T-series device occurs using one of two modes:

1. Command-response

Command-response mode is appropriate for most applications. In command-response mode, the host sends a command data packet, to which the T-series device sends a response data packet.

2. Stream

Stream mode is when the device collects periodic sampling events automatically. Collected data is stored in the device's memory until it retrieved by the host application. The [LJM library stream functions](#) simplify data collection from T-series devices. Not all functionality is supported in stream mode. Please refer to the [Stream Mode](#) section of the user's manual for more details.

For more information about command-response and stream, see [3.0 Communication](#).

Figure A1.1.1 depicts the two operating modes.

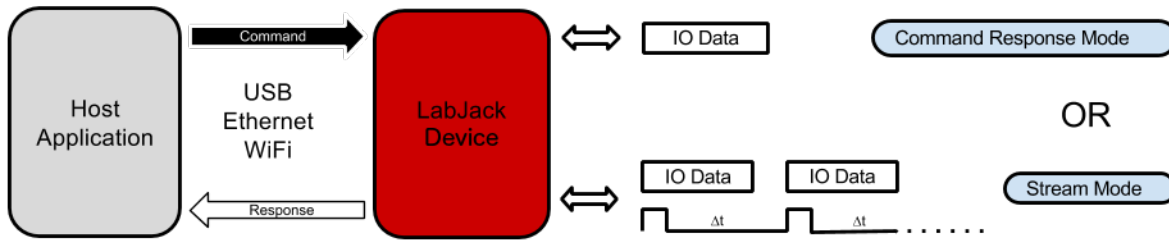


Figure A1.1. Communication modes

The use of a particular mode will depend on functionality and the hardware response time required by the end application.

Command-Response Data Rates

All communication performed with T-series devices is accomplished using the [Modbus TCP](#) protocol, thus allowing direct communication with the device via low-level TCP commands. As an alternative, the [LJM library](#) may be used as a higher level communications layer for added convenience and minimal additional overhead. Tables A.1.1 and A.1.2 list expected communication overhead times associated with Modbus TCP and LJM Library communication options. These times are similar for all T-series devices, but the following were measured on a T7 (LJM: 1.0706, Firmware: 1.046).

Table A.1.1. Typical communication overhead using direct Modbus TCP.

	USB High-High	USB Other	Ethernet	WiFi
	[ms]	[ms]	[ms]	[ms]
No I/O - Overhead	0.6	2.1	1.0	6.5
Read All DI	0.7	2.2	1.1	6.6
Write All DO	0.7	2.2	1.1	6.6
Write Both DACs	0.7	2.2	1.1	6.6

Table A.1.2. Typical communication overhead using LJM library.

	USB High-High	USB Other	Ethernet	WiFi
	[ms]	[ms]	[ms]	[ms]
No I/O - Overhead	0.6	2.2	1.1	6.7
Read All DI	0.7	2.3	1.2	6.8
Write All DO	0.7	2.3	1.2	6.8
Write Both DACs	0.7	2.3	1.2	6.8

The times shown in table A.1.2 were measured using a LabVIEW program running on Windows where all read and write operations are conducted with a single LJM_eNames() call. The LJM_eNames() functions is used to minimize the number of Modbus packets sent from the host (one packet per command/response set). The test program executes one of the listed tasks within a loop for a specified number of iterations, over a 1-10 second period. The overall execution time is divided by the total number of iterations, providing the average time per iteration for each task. The execution time includes LabVIEW overhead, LJM library overhead, Windows overhead, communication time (USB/Ethernet/WiFi), and device 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 often 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 OS X, and Windows are generally "best-effort" operating systems and not "real-time", meaning that the listed CR speeds 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 [1].

USB and Ethernet:

These times are quite predictable. Software issues mentioned above are important—but, in terms of hardware, the times will be consistent. The device communication does not consume a major portion of total USB or Ethernet bandwidth. Therefore, the overhead times listed are typically maintained even with substantial activity on the bus.

WiFi - T7-Pro Only:

WiFi latency tends to vary more than USB or Ethernet latency. With a solid connection, most WiFi packets have an overhead of 3 to 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. The results were:

- 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, causing a greater number of packets retries. An RSSI greater than -35 reflects a very strong connection, typically within a few feet of the access point. This also results in a greater numbers of retries due to saturation of the RF signal.

ADC Conversions:

Analog to digital conversions (ADC) will increase the command-response time depending on the number of channels, the input gain (T7), and the resolution index being used. The following tables list conversion times for various different settings when reading a single analog input channel. The total command-response time (CRT) when reading analog inputs is equal to the overhead time from tables A.1.1 and A.1.2 added to the conversion times for the requested channels:

$$\text{CRT (milliseconds)} = \text{overhead} + (\#\text{AINs} * \text{AIN Sample Time})$$

Table A.1.3. Typical C-R milliseconds per sample, T4.

Resolution Index	Effective Resolution [bits]	Effective Resolution [μV]	AIN Sample Time [ms/sample]
High-Voltage Channels (AIN0-AIN3)			
1			
2			
3			
4			
5			
Low-Voltage Channels (Applicable FIO & EIO)			
1			
2			
3			
4			
5			

Table A.1.4. Typical C-R milliseconds per sample, T7.

Resolution Index	Effective Resolution [bits]	Effective Resolution [μV]	AIN Sample Time [ms/sample]
Gain/Range: 1\pm10V			
1	16.0	316	0.04
2	16.5	223	0.04
3	17.0	158	0.1
4	17.5	112	0.1
5	17.9	84.6	0.2
6	18.3	64.1	0.3

7	18.8	45.3	0.6
8	19.1	36.8	1.1
9	19.6	26.0	3.5
10	20.5	14.0	13.4
11	21.3	8.02	66.2
12	21.4	7.48	159
Gain/Range:10/±1V			
1	15.4	47.9	0.2
2	16.0	31.6	0.2
3	16.5	22.3	0.6
4	16.9	16.9	0.6
5	17.4	12.0	1.2
6	17.9	8.46	2.3
7	18.3	6.41	2.6
8	18.7	4.86	3.1
9	19.5	2.79	3.5
10	20.5	1.40	13.4
11	21.4	0.748	66.2
12	21.5	0.698	159
Gain/Range: 100/±0.1V			
1	13.3	20.5	1.0
2	14.2	11.0	2.0
3	14.7	7.78	5.1
4	15.2	5.50	5.1
5	15.7	3.89	5.2
6	16.3	2.57	10.3
7	16.7	1.94	10.6
8	17.2	1.37	11.1
9	18.3	0.641	3.5
10	19.1	0.368	13.4
11	19.6	0.260	66.2
12	19.7	0.243	159
Gain/Range: 1000/±0.01V			
1	10.9	10.8	5.0
2	12.3	4.10	10.0
3	12.7	3.11	10.1
4	13.3	2.05	10.1
5	13.8	1.45	10.2
6	14.4	0.96	10.3
7	14.7	0.778	10.6
8	15.0	0.632	11.1
9	15.4	0.479	3.5
10	16.1	0.295	13.4
11	16.4	0.239	66.2
12	16.4	0.239	159

Streaming Data Rates

The fastest data rates on T-series devices occur when operating in stream mode. Much of the command-response overhead is eliminated in stream mode because the device is responsible for initiating IO operations. The device collects scans in its stream buffer, then the host application then retrieves multiple scans at once. The end result is a continuous data stream, sampled at regular intervals, collected with a minimum number of communication packets [2].

There is an important distinction between *scans* and *samples*. Definitions are as follows:

- Address: Also called a *channel*. An address usually returns the value of 1 input connection.
- Sample: A reading from one address.
- Scan: One reading from every address in the scan list.
- Scan list: The list of one or more addresses in a scan.

The scan rate is the rate at which scans are collected. It is a fraction of the sample rate, where the fraction is the inverse of the number of channels being read in a single scan. The scan rate is defined as:

$$\text{ScanRate} = \text{SampleRate} / \text{NumAddresses}$$

The sample rate and scan rate are equal when the NumAddresses is 1.

T4 Stream Rates

- **Ethernet** generally maxes out right around 50 ksamples/second.
- **USB** generally maxes out right around 50 ksamples/second.

The T4 has a **typical maximum sample rate** of 50 ksamples/second. This maximum sample rate is achievable when a stream is configured with the default RESOLUTION_INDEX = 0 or 5 [3.]. This maximum is reflected in the first row of data in the following table (highlighted). The scan rates reported are the maximum sample rates divided by the number of channels in the scan list (within ~10%).

The scan rates in the following tables are continuous over USB or Ethernet.

Table A.1.5. T4 Stream: Scan rates for different values of resolution index. Applies to USB and Ethernet. Applies to all streamable addresses including low-voltage and high-voltage analog inputs.

	Maximum Scan Rate				Maximum Sample Rate
	1 Channel	2 Channels	4 Channels	8 Channels	>1 Channel
	[Hz]	[Hz]	[Hz]	[Hz]	[Hz]
Resolution Index = 1	50k	25k	12.5k	6.25k	50k
Resolution Index = 2					
Resolution Index = 3					
Resolution Index = 4					
Resolution Index = 5					

Table A.1.6. T4 Stream: Typical noise and interchannel delay values depending on resolution index.

Resolution Index	Peak-to-Peak Noise	Interchannel Delay
	[16-bit counts]	[μ s]
1	\pm TBD	<20
2		
3		
4		
5		

T7 Stream Rates

- **Ethernet** can usually maintain just under 120 ksamples/second.
- **USB** generally maxes out right around 100 ksamples/second.
- When using **WiFi**, the device can acquire data at the fastest rates, but transfer of data to the host is limited to about 1 ksamples/second, so the fastest stream rates cannot be maintained continuously. In this case, stream-burst can be used rather than continuous stream, where each stream is limited to a specified number of scans that fits in the device's stream buffer. For high-speed wireless streaming, use the Ethernet connection with an external Ethernet-WiFi bridge.

The T7 has a **typical maximum sample rate** of 100 ksamples/second. This maximum sample rate is achievable when a stream is configured with RANGE = \pm 10V and RESOLUTION_INDEX = 0 or 1 [3.]. This maximum is reflected in the first row of data in table A.1.4 (highlighted). The scan rates reported in table A.1.4 are the maximum sample rates divided by the number of channels in the scan list (within ~10%).

The scan rates in the following tables are continuous over USB or Ethernet.

Table A.1.7. T7 Stream: Scan rates over various gain, resolution index, channel count combinations. Applies to USB and Ethernet.

Gain : Range	Maximum Scan Rate				Maximum Sample Rate
	1	2	4	8	

		Channel [Hz]	Channels [Hz]	Channels [Hz]	Channels [Hz]	>1 Channel [Hz]
Resolution Index = 1	1 : ±10V	100k	50k	25k	12.5k	100k
	10 : ±1V	100k	4.1k	1.4k	585	8.2k
	100 : ±0.1V	100k	850	315	120	1.7k
	1000 : ±0.01V	100k	N.S.	N.S.	N.S.	N.S.
Resolution Index = 2	1 : ±10V	48k	19.8k	9.0k	4.0k	39.6k
	10 : ±1V	48k	3.6k	1.3k	550	7.2k
	100 : ±0.1V	48k	400	N.S.	N.S.	800
	1000 : ±0.01V	48k	N.S.	N.S.	N.S.	N.S.
Resolution Index = 3	1 : ±10V	22k	9.9k	4.5k	2.4k	19.8k
	10 : ±1V	22k	1.4k	500	225	2.8k
	100 : ±0.1V	22k	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	22k	N.S.	N.S.	N.S.	N.S.
Resolution Index = 4	1 : ±10V	11k	4.9k	2.2k	1.3k	9.8k
	10 : ±1V	11k	1.3k	45	N.S.	2.6k
	100 : ±0.1V	11k	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	11k	N.S.	N.S.	N.S.	N.S.
Resolution Index = 5	1 : ±10V	5500	2.2k	990	630	4.4k
	10 : ±1V	5500	630	23	N.S.	1.3k
	100 : ±0.1V	5500	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	5500	N.S.	N.S.	N.S.	N.S.
Resolution Index = 6	1 : ±10V	2500	1.3k	630	315	2.6k
	10 : ±1V	2500	320	N.S.	N.S.	640
	100 : ±0.1V	2500	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	2500	N.S.	N.S.	N.S.	N.S.
Resolution Index = 7	1 : ±10V	1200	650	315	N.S.	1.3k
	10 : ±1V	1200	220	N.S.	N.S.	440
	100 : ±0.1V	1200	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	1200	N.S.	N.S.	N.S.	N.S.
Resolution Index = 8	1 : ±10V	600	315	N.S.	N.S.	630
	10 : ±1V	600	200	N.S.	N.S.	400
	100 : ±0.1V	600	N.S.	N.S.	N.S.	N.S.
	1000 : ±0.01V	600	N.S.	N.S.	N.S.	N.S.

*N.S. indicates settings not supported in stream mode.

The maximum scan rate will decrease at higher resolution index and range settings simply because analog conversions take longer to complete. Table A.1.5 illustrates how analog conversion times increase at different resolution index and range settings.

Table A.1.8. T7 Stream: Typical noise and interchannel delay values depending on range and resolution index.

Resolution Index	Peak-to-Peak Noise [16-bit counts]	Interchannel Delay [µs]
Gain/Range: 1/±10V		
1	±3.0	15/8*
2	±2.0	25
3	±1.5	45
4	±1.0	90
5	±1.0	170
6	±0.5	335
7	±0.5	670
8	±0.5	1,335
Gain/Range: 10/±1V		
1	±4.5	210
2	±3.0	220
3	±2.0	545
4	±1.5	585
5	±1.0	1,200
6	±0.5	2,415
7	±0.5	2,750
8	±0.5	3,415
Gain/Range: 100/±0.1V		

2	±13.0	±14.0
3	N.S.	N.S.
4	N.S.	N.S.
5	N.S.	N.S.
6	N.S.	N.S.
7	N.S.	N.S.
8	N.S.	N.S.
Gain/Range: 1000/±0.01V		
1	N.S.	N.S.
2	N.S.	N.S.
3	N.S.	N.S.
4	N.S.	N.S.
5	N.S.	N.S.
6	N.S.	N.S.
7	N.S.	N.S.
8	N.S.	N.S.

N.S. indicates settings not supported in stream mode.

*8 μ s is used when streaming faster than 60 kSPS.

Interchannel Delay:

Interchannel delay is the time between each sample within a scan. For example, say 3 channels are streamed at 1000 scans/second with ResolutionIndex=1 and Range=10. That is a sample rate of 3000 samples/second, so from the table above the interchannel delay is 15 μ s. The stream interrupt will fire every 1000 μ s, at which time it takes about 5 μ s until the 1st channel is sampled, then 15 μ s later the 2nd channel is sampled, then 15 μ s later the 3rd channel is sampled, and then about 965 μ s later the next scan starts.

What if in the above example we wanted the 8 μ s delay rather than 15 μ s? The sample rate must be greater than 60 ksamples/second for that, so the solution is increase sample rate by scanning more channels (channels can be repeated in the scan list) or scanning faster and discarding the extra data.

The interchannel delay is a fixed time with little jitter, so the known time can be accounted for in user software to adjust phase if those microseconds are important. As an alternative to using the table above, the user can measure interchannel delay on their device by using a scope to look at the SPC timing output described in the [Stream Mode](#) section.

Notes:

1. Various software issues need consideration when implementing a feedback loop that executes at the desired time interval. Some considerations are: thread priority, logging to file, updating the screen, and other programs running on the machine.
2. The number of packets used to retrieve stream data depends on the number of data points allowed to accumulate in the stream buffer.
3. Setting the resolution index to 0 (default) in stream mode is equivalent to a resolution equal to 1. The default resolution index in stream mode behaves different than command-response mode.

A-2 Digital I/O [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

General Info

T-series Digital Input/Output lines information:

Table A2-1. IO Information

Parameter	Conditions	Min	Typical	Max	Units
Low Level Input Voltage		-0.3		0.5	Volts
High Level Input Voltage		2.64		5.8	Volts
Hysteresis Voltage [1]					
---Low to High Transition				1.15	Volts
---High to Low Transition		0.90			Volts
Maximum Input Voltage [2]	FIO	-10		10	Volts
	EIO/CIO/MIO	-6		6	Volts

Output Low Voltage [3][4]	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 [3][4]	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 [3][4]	FIO	6.3	mA
	EIO/CIO/MIO	22.9	mA
Output Impedance [3][4]	FIO	550	Ω
	EIO/CIO/MIO	180	Ω

[1] The "Low Level" and "High Level" input voltage specify input voltage ranges guaranteed to produce the correct logic state at any digital input. The "Hysteresis Voltage" represents the voltage where a logic transition will actually occur. Input hysteresis will result in different transition voltages, depending on transition from HIGH to LOW or LOW to HIGH logic states. The overall hysteresis band is ~0.25 volts.

[2] 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 device is unpowered, as the voltage will attempt to supply operating power to the device possibly causing poor start-up behavior.

[3] 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.

[4] It is recommended to use the EIO/CIO digital I/O lines for UART, SPI, I²C, 1-Wire, and other digital communication protocols.

Extended Features

T-series DIO-EF information:

Table A2-2. DIO extended features information

Extended Features	Conditions	Min	Typical	Max	Units
Frequency Output [1]		0.02		5 M	Hz
Counter Input Frequency [2]				5	MHz
Minimum High & Low Time [2]				50	ns
"Interrupt" Total Edge Rate [3][4]	No Stream			70k	edges/s
	T7 Streaming @ 50 kHz			20k	edges/s
	T4 Streaming @ 20 kHz			20k	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] This is for the "Interrupt" modes. To avoid missing edges, keep the total number of applicable edges on all applicable timers below this limit.
[4] Excessive processor loading could reduce these limits further.

Serial Communication

T-series serial communication abilities information is below. T-series 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.

Table A2-3. Serial communication information

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 [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

General Info

Table A.3-1. T7 Analog Input Information

	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
Max Source Impedance [4]			1		kΩ
Integral Linearity Error	Range=10, 1, 0.1			±0.01	%FS
	Range=0.01			±0.1	%FS
Absolute Accuracy	Range=10, 1, 0.1			±0.01	%FS
	Range=10			±2000	μV
	Range=1			±200	μV
	Range=0.1			±20	μV
	Range=0.01			±0.1	%FS
	Range=0.01			±20	μV
Temperature Drift			15		ppm/°C
Channel Crosstalk [5]	< 1kHz		-100		dB
	1kHz - 50kHz		20		dB/dec

-3dB Frequency [6]	Gain=1-10 Gain=100	445 337		kHz
	Gain=1000	63		kHz
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 Appendix A-3-2 Signal Range . 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 one front-end mux and all odd channels on a second 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 your 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.				
[5] Typical crosstalk on a grounded AIN pin, with 20Vpp sine wave on adjacent AIN pin. An adjacent AIN pin refers to multiplexer channel location not channel number, e.g. AIN0-AIN2 or AIN1-AIN3 pairs.				
[6] This is the bandwidth of the analog hardware. Any frequencies less than this will go through the analog system to the ADC and be part of the digitized waveform. For DC measurements this is of little concern as ResolutionIndex and averaging can be used to get rid of extra noise. For AC measurements, frequency components below the nyquist point can be removed after digitizing, but frequency components above the nyquist point must be removed before digitizing as they will alias. If unwanted signals with frequencies between the nyquist point and analog cutoff frequency are expected, and they are expected to have sufficient magnitude to be above the acceptable noise level, then an external hardware filter must be used.				

See also: Appendix A-3-1 [Noise and Resolution](#)

A-3-1 Noise and Resolution [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

ADC Noise and Resolution

T-series devices use an internal analog-to-digital converter (ADC) to convert analog voltage into digital representation. The ADC reports an analog voltage in terms of ADC counts, where a single ADC count is the smallest change in voltage that will affect the reported ADC value. A single ADC count is also known as the converter's least significant bit (LSB) voltage. The ADC's resolution defines the number of discrete voltages represented over a given input range. For example, a 16-bit ADC with a ± 10 input range can report 65536 discrete voltages (2^{16}) and has an LSB voltage of 0.305 mV ($20 \text{ V} \div 2^{16}$).

The stated resolution for an ADC is a theoretical, best-case value assuming no channel noise. In reality, every ADC works in conjunction with external circuitry (amplifiers, filters, etc.) which all possess some level of inherent noise. The noise of supporting hardware, in addition to noise of the ADC itself, all contribute to the channel resolution. In general, the resolution for an ADC and supporting hardware will be less than what is stated for the ADC. The combined resolution for an in-system ADC is termed effective resolution. Simply put, the effective resolution is the equivalent resolution where analog voltages less than the LSB voltage are no longer differentiable from the inherent hardware noise. In addition to defining the smallest measurable analog voltage, the effective resolution also defines the RMS peak-to-peak noise on a given analog channel.

The effective resolution is closely related to the error free code resolution (EFCR) or *flicker-free* code resolution. The EFCR represents the resolution on a channel immune to "bounce" or "flicker" from the inherent system noise. The EFCR is not reported in this appendix. However, it may be closely approximated by the following equation:

$$\text{EFCR} = \text{effective resolution} - 2.7 \text{ bits} \quad [1]$$

The T4 and the T7 offer user-selectable resolution through the resolution index parameter on any one AIN channel. Internally, the ADC hardware uses modified sampling methods to increase measurement resolution beyond the ADC's base resolution. Valid resolution index values are:

- 0-5 for the T4
- 0-8 for the T7
- 0-12 for the T7-Pro [2][3]

Increasing the resolution index value will improve the channel resolution, but doing so will usually extend channel sampling times. See section 14.0 AIN for more information on the resolution index parameter and its use.

Noise and Resolution Data

The data shown below summarizes typical effective resolutions and expected channel sampling times over all resolution index values. Data for the T7 and T7-Pro data are combined and presented together for convenience, where resolution index values 9-12 only apply to the T7-Pro.

The AIN sampling time is the typical amount of time required for the ADC hardware to make a single analog to digital conversion on any channel and is reported in milliseconds per sample. The AIN sampling time does not include command/response and overhead time associated with the host computer/application.

Noise and Resolution Test procedure

Noise and resolution data was generated by collecting 512 successive voltage readings, using a short jumper between the test channel and ground. The resulting data set represents typical noise measured on any one analog input channel in ADC counts. The effective resolution is calculated by subtracting the RMS channel noise (represented in bits) from 16-bits.

$$\text{Effective Resolution} = 16 \text{ bits} - \log_2(\text{RMS Noise [in ADC counts]})$$

Table A.3.1.1. T7 resolution data. Effective resolution and sampling times for various gains and resolution index settings. Resolution index settings 9-12 apply to the T7-Pro only.

Resolution Index	Effective Resolution [bits]	Effective Resolution [μV]	AIN Sample Time [ms/sample]
Gain/Range: 1/±10V			
1	16.0	316	0.04
2	16.5	223	0.04
3	17.0	158	0.06
4	17.5	112	0.09
5	17.9	85	0.16
6	18.3	64	0.29
7	18.8	45	0.56
8	19.1	37	1.09
9	19.6	26	3.50
10	20.5	14	13.4
11	21.4	7.5	66.2
12	21.8	5.7	159
Gain/Range: 10/±1V			
1	15.4	48	0.23
2	16.0	32	0.23
3	16.5	22	0.55
4	16.9	17	0.58
5	17.4	12	1.15
6	17.9	8.5	2.28
7	18.3	6.4	2.55
8	18.7	4.9	3.08
9	19.5	2.8	3.50
10	20.5	1.4	13.4
11	21.4	0.7	66.2
12	21.7	0.6	159
Gain/Range: 100/±0.1V			
1	13.3	21	1.03
2	14.2	11	2.03
3	14.7	7.8	5.05
4	15.2	5.5	5.08
5	15.7	3.9	5.15
6	16.3	2.6	10.28
7	16.7	1.9	10.55
8	17.2	1.4	11.08
9	18.3	0.6	3.50

Effective Resolution Vs Resolution Index

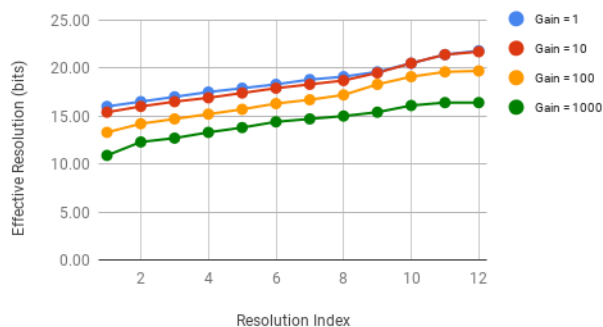


Figure A.3.1.2. T7 analog input effective resolution over various gains and resolution index settings.

LSB Voltage Vs Resolution Index

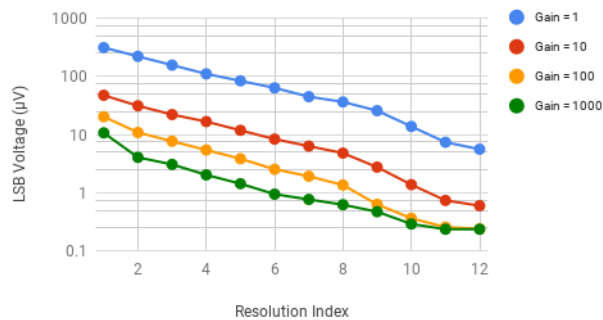


Figure A.3.1.3. T7 analog input LSB voltage over various gains and resolution index settings.

10	19.1	0.4	13.4
11	19.6	0.3	66.2
12	19.7	0.2	159
Gain/Range: 1000/±0.01V			
1	10.9	11	5.03
2	12.3	4.1	10.0
3	12.7	3.1	10.1
4	13.3	2.1	10.1
5	13.8	1.5	10.2
6	14.4	1.0	10.3
7	14.7	0.8	10.6
8	15.0	0.6	11.1
9	15.4	0.5	3.50
10	16.1	0.3	13.4
11	16.4	0.2	66.2
12	16.4	0.2	159

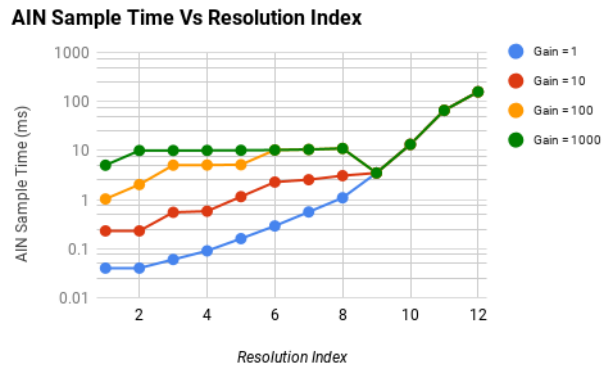


Figure A.3.1.4. T7 AIN sample times for analog inputs over various gains resolution index settings.

Notes:

- [1] The equation used to approximate the EFCR is determined using +/-3.3 standard deviations from the RMS noise measured on an AIN channel.
- [2] The default value for RESOLUTION_INDEX is 0, which equates to 8 for T7 command-response reads, 9 for T7-Pro command-response reads, and 1 for T7 & T7-Pro stream reads.
- [3] The T7-Pro is equipped with a 24-bit delta-sigma ADC, in addition to the standard 16-bit ADC. Analog conversions occur on the 16-bit ADC when resolution index values 0-8 are used. Analog conversions occur on the 24-bit ADC when resolution index values 9-12 are used (command response mode only).
- [4] The hi-resolution 24-bit ADC is not supported in stream mode.

Support - Table Styling Fix

A-3-2 Signal Range [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T4 AIN Signal Range

Analog inputs on the T4 are single-ended only. That means the voltage of a given input terminal is acquired versus GND, and thus the signal range is simply the same as the analog input ranges of ±10V or 0-2.5V discussed in various places. See [Appendix A-3](#) for further analog input specs.

T7 AIN Signal Range

The [instrumentation amplifier](#) in the T7 (see [Figure 4.2-2](#)) provides 4 different gains:

- x1 (RANGE is ±10 volts)
- x10 (RANGE is ±1 volts)
- x100 (RANGE is ±0.1 volts)
- x1000 (RANGE is ±0.01 volts)

The input ranges are straightforward for single-ended measurements, but can be a little tricky for [differential measurements](#) if neither channel (positive or negative) is at 0 volts.

The figures below show the approximate signal range of the T7 analog inputs at gains of x1 and x1000.

Input Common-Mode Voltage, known as V_{cm} , is:

$$V_{cm} = (V_{pos} + V_{neg})/2$$

The voltage of any input compared to GND should be within the VM+ and VM- rails by at least 1.5 volts, so if VM+ and VM- is the typical ±13 volts, the signals

should be within ± 11.5 volts compared to GND. See [Table A5-8](#) for more information on VM+ and VM-.

Example #1 - invalid because $V_{cm}=10.0$ with $V_{out}=10.0$ is invalid:

Suppose a differential signal is measured, where:

- V_{pos} is 10.05 volts compared to GND
- V_{neg} is 9.95 volts compared to GND
- $G=100$ (RANGE= ± 0.1)

That means:

- $V_{cm}=10.0$ volts,
- $V_{diff}=0.1$ volts,
- and the expected $V_{out}=10.0$ volts.

Figures for $G=10$ and $G=100$ are not shown, but $V_{cm}=10.0$ volts and $V_{out}=10.0$ volts is not valid at $G=1$ or $G=1000$, so it is not valid for gains in between.

Example #2 - invalid because V_{pos} compared to GND is too high:

Suppose a differential signal is measured, where:

- V_{pos} is 12.0 volts compared to GND
- V_{neg} is 8.0 volts compared to GND
- $G=1$ (RANGE= ± 10)

That means:

- $V_{cm}=10.0$ volts,
- $V_{diff}=4.0$ volts,
- and the expected $V_{out}=4.0$ volts.

This looks almost okay in the $G=1$ figure below, but the voltage of V_{pos} compared to GND is too high so this is not valid.

Example #3 - valid:

Suppose a single-ended signal is measured, where:

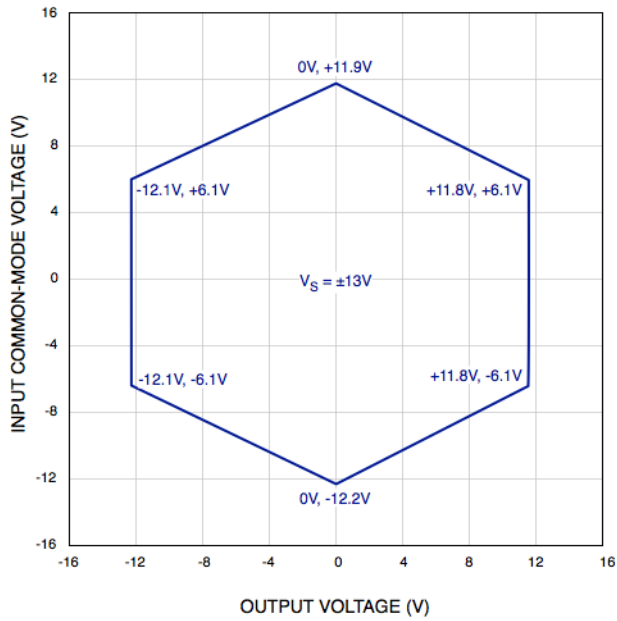
- V_{pos} is 10.0 volts compared to GND
- $G=1$ (RANGE= ± 10)

That means:

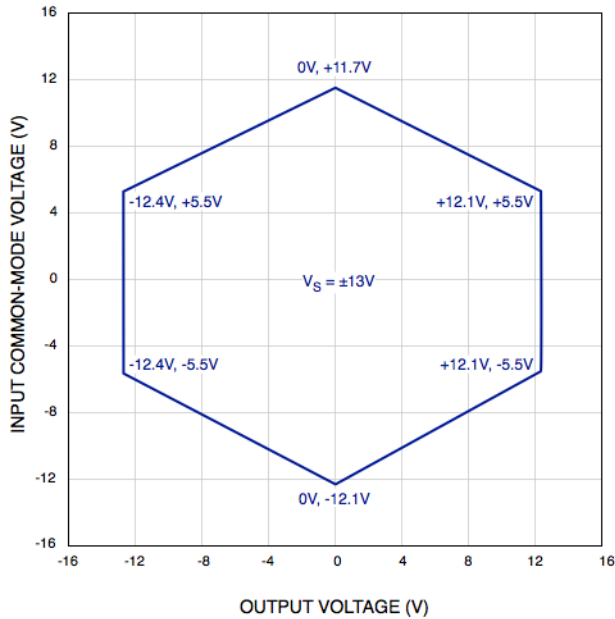
- $V_{cm}=5.0$ volts,
- $V_{diff}=10.0$ volts,
- and the expected $V_{out}=10.0$ volts.

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 [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Specifications for [analog output](#) channels (DAC0 and DAC1) are shown below.

Table A4-1. T4 DAC Information

	Conditions	Min	Typical	Max	Units
Nominal Output Range [1]	No Load	0.01		4.98	Volts
	@ ± 2.5 mA	0.30		4.69	Volts
Resolution			10		Bits
Absolute Accuracy	5% to 95% FS		± 0.15		% FS
Integral Linearity				± 1	counts

Differential Linearity Error		±0.1	±0.5	counts
Noise [2]		±1		counts
Source Impedance [3]		50		Ω
Current Limit [4]	Max to GND	30		mA
Time Constant		1		μs
<p>[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.</p>				
<p>[2] TBD</p>				
<p>[3] For currents up to about TBDmA, this source impedance dominates the error due to loading. For example, if you load DAC0 with a 1000 ohm resistor from DAC0 to GND, and set DAC0 to 3.0V, the actual voltage at the DAC0 terminal will be about $3.0 \cdot 1000 / (50 + 1000) = 2.86V$. For currents > TBDmA, you increasingly get added droop due to the ability of the output buffer to drive substantial current close to the power rails.</p>				
<p>[4] The output buffer will limit current to about TBDmA and can maintain this value continuously without damage. Take, for example, a 100 ohm resistor from DAC0 to GND, with the internal source impedance of 50 ohms, and DAC0 set to 4.5V. A simple calculation would predict a current of $4.5 / (50 + 100) = 30mA$, but the output buffer will limit the current to TBDmA. A simple calculation taking into account only the voltage droop due to the internal 50 ohm resistance would predict a voltage at the DAC0 terminal of $4.5 \cdot 100 / (50 + 100) = 3.0V$, but since the current is limited to TBDmA the actual voltage at DAC0 would be more like $100 \cdot 0.02 = 2.0V$.</p>				

Table A4-2. T7 DAC Information

	Conditions	Min	Typical	Max	Units
Nominal Output Range [5]	No Load	0.01		4.99	Volts
	@ ±2.5 mA	0.25		4.75	Volts
Resolution			12		Bits
Absolute Accuracy	5% to 95% FS		±0.06		% FS
Integral Linearity Error			±1.5	±2	counts
Differential Linearity Error			±0.25	±0.5	counts
Noise [6]			±100		μV
Source Impedance [7]			50		Ω
Current Limit [8]	Max to GND		20		mA
Time Constant			4		μs
<p>[5] 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.</p>					
<p>[6] With load, the noise increases if operating too close to VS. With a 1000 ohm load, noise increases noticeably at 4.4V and higher. With a 330 ohm load, noise increases noticeably at 3.7V and higher. With a 100 ohm load, noise increases noticeably at 2.7V and higher.</p>					
<p>[7] For currents up to about 8mA, this source impedance dominates the error due to loading. For example, if you load DAC0 with a 1000 ohm resistor from DAC0 to GND, and set DAC0 to 3.0V, the actual voltage at the DAC0 terminal will be about $3.0 \cdot 1000 / (50 + 1000) = 2.86V$. For currents > 8mA, you increasingly get added droop due to the ability of the output</p>					

buffer to drive substantial current close to the power rails.

[8] The output buffer will limit current to about 20mA and can maintain this value continuously without damage. Take, for example, a 100 ohm resistor from DAC0 to GND, with the internal source impedance of 50 ohms, and DAC0 set to 4.5V. A simple calculation would predict a current of $4.5/(50+100) = 30\text{mA}$, but the output buffer will limit the current to 20mA. A simple calculation taking into account only the voltage droop due to the internal 50 ohm resistance would predict a voltage at the DAC0 terminal of $4.5 \cdot 100/(50+100) = 3.0\text{V}$, but since the current is limited to 20mA the actual voltage at DAC0 would be more like $100 \cdot 0.02 = 2.0\text{V}$.

A-5 General Specs [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Power Supply Input

The following table shows the supply voltage that is required. The USB hub or 5V USB adapter in use should fall within the acceptable range.

Table A5-1.

Parameter	Condition	Min	Typical	Max	Units
Supply Voltage		4.75		5.25	Volts
Supply Current	T7, No connected loads [1]	4	250	290	mA
	T4, No connected loads [1]		210		mA

[1] Supply current will vary, depending on communication and enabled functionality. See Power Consumption section below for more details. Total continuous supply current should be kept to 500 mA or less.

vs Outputs

The following table provides specifications for the VS outputs.

Table A5-2.

Parameter	Condition	Min	Typical	Max	Units
Voltage [1]		4.75		5.25	Volts
Max Current	500 mA - Supply Current		200		mA

[1] VS voltage is the same as the power supply voltage.

System Clock

Table A5-3.

Parameter	Condition	Min	Typical	Max	Units
Clock Error	~ 25 °C			±20	ppm
	-10 to 60 °C			±50	ppm
	-40 to 85 °C			±100	ppm

Physical

Table A5-4.

Parameter	Condition	Min	Typical	Max	Units
-----------	-----------	-----	---------	-----	-------

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	
Enclosure Screws (x6)	PH1 pan head		#4-20 x 5/8"		Phil #1

Power Consumption

At this time USB and Core speed are not intended for user level control, but have been included in the following table to show the capabilities of the device. The values shown are typical.

Table A5-5. T4 Power Consumption

Core Speed	Eth [1]	Eth Linked	LEDs	USB [1]	USB Linked	Draw (mA)	Typical Deviation (%)
80M	ON	Yes	ON	ON	Yes	210	±4
80M	ON	Yes	ON	ON	No	195	±4
80M	ON	No	ON	ON	Yes	170	±4
80M	ON	No	OFF	ON	Yes	TBD	±10
80M	ON	No	OFF	OFF	No	TBD	±10
20M	ON	No	OFF	OFF	No	TBD	±20
2M	OFF	No	OFF	OFF	No	TBD	±20
250k	OFF	No	OFF	OFF	No	TBD	±33

[1] Ethernet and USB require that the core be running at least 20MHz.

Table A5-6. T7 Power Consumption

Core Speed	Eth [1]	Eth Linked	AINs	WiFi	WiFi Linked	LEDs	USB [1]	Draw (mA)	Typical Deviation (%)
80M	ON	Yes	ON	ON	Yes	ON	ON	280	+/- 4
80M	ON	Yes	ON	ON	No	ON	ON	280	+/- 3
80M	ON	Yes	ON	OFF	No	ON	ON	250	+/- 4
80M	ON	No	ON	OFF	No	ON	ON	200	+/- 5
80M	OFF	No	ON	OFF	No	ON	ON	170	+/- 4
80M	OFF	No	OFF	OFF	No	ON	ON	140	+/- 4
80M	OFF	No	OFF	OFF	No	OFF	ON	94	+/- 11
80M	OFF	No	OFF	OFF	No	OFF	OFF	86	+/- 9
20M	OFF	No	OFF	OFF	No	OFF	OFF	28	+/- 18
2M	OFF	No	OFF	OFF	No	OFF	OFF	10	+/- 20
250k	OFF	No	OFF	OFF	No	OFF	OFF	6	+/- 33

[1] Ethernet and USB require that the core be running at least 20MHz.

200µA and 100µA Current Sources - T7 Only

Table A5-7.

Parameter	Condition	Min	Typical	Max	Units
Accuracy vs. Cal Value [1]	~ 25 °C		±0.1	±0.2	%
Accuracy vs. Nominal [1]	~ 25 °C			±5	%
TempCo 200UA [2]	~ 25 °C				ppm/°C
TempCo 10UA [2]	~ 25 °C				ppm/°C
Maximum Voltage			2.0V less than vs		volts
[1] First spec is the accuracy compared to the value stored during calibration. The second spec is the accuracy compared to the nominal value (e.g. 200.0 µA for the 200UA source).					
[2] Tempco varies strongly with temperature. See the charts in 12.0 200uA and 10uA .					

VM+ and VM- (T7 Only)**Table A5-8.**

Parameter	Condition	Min	Typical	Max	Units
Typical Voltage	No-load		±13		volts
	@ 2.5 mA		±12		volts
Maximum Current			2.5		mA

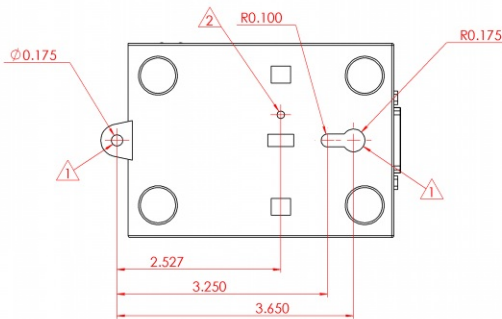
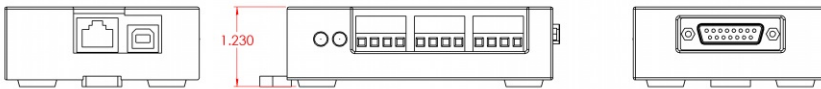
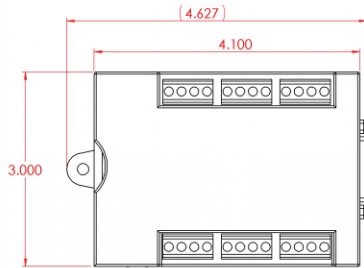
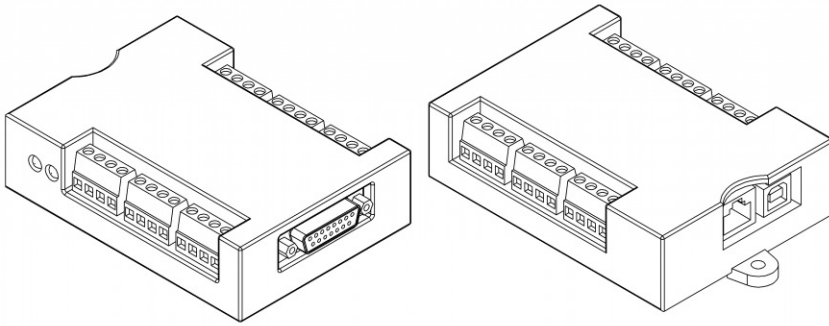
Appendix B - Drawings and CAD Models [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T-Series enclosure and OEM drawings and models are provided in several neutral file formats. Please see device-specific subsections below.

B-1 T4 Drawings and CAD Models [T-Series Datasheet]

[Log in](#) or [register](#) to post comments



Notes:

1. Mounting holes are sized for #8 panhead screws.
2. Receptacle holes for plastic DIN rail clip ([TKAD](#)).

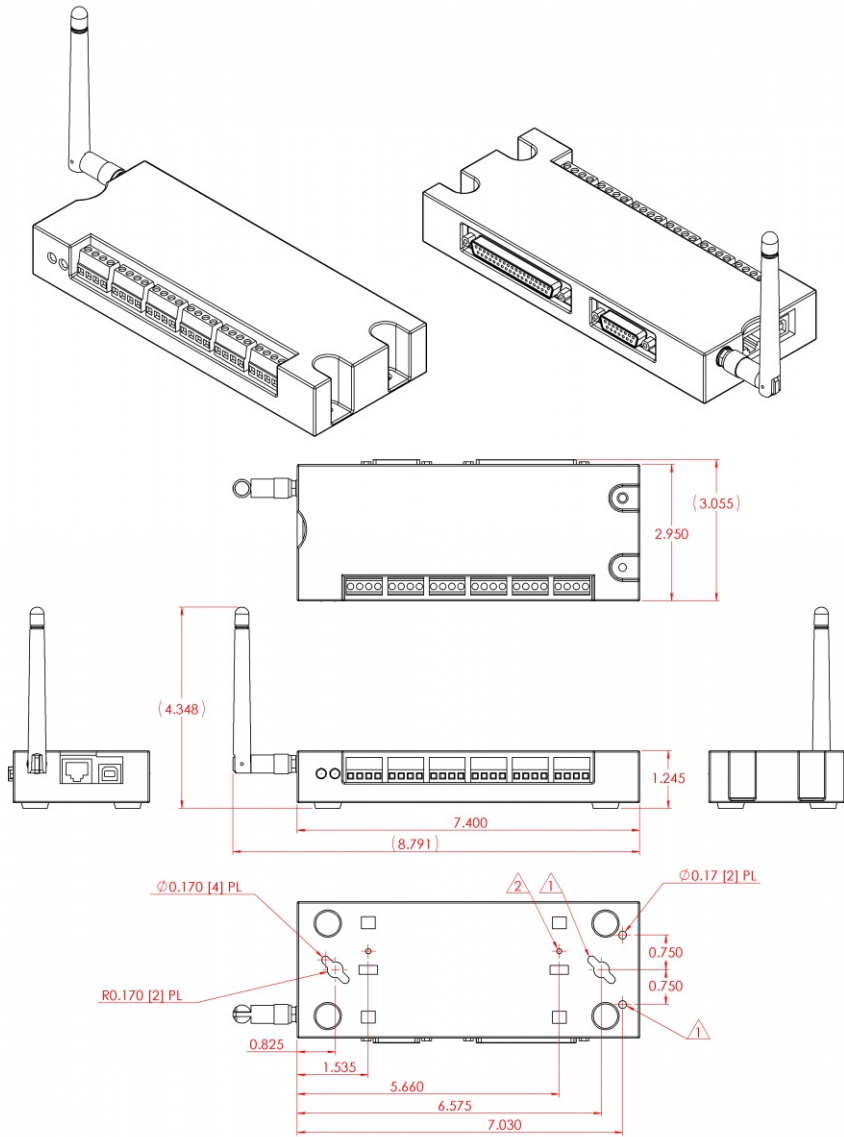
Common neutral format CAD models are provided below. Right-click and select the "Save link as..." option to download STEP files.

File Attachment:

- [T4.STEP](#)
- [T4_OEM.STEP](#)
- [T4_PCB.DXF](#)

B-2 T7 Drawings and CAD Models [T-Series Datasheet]

[Log in](#) or [register](#) to post comments



Notes:

1. Mounting holes are sized for #8 panhead screws.
2. Receptacle holes for plastic DIN rail clip (TKAD).
3. The T7-Pro is depicted above. The standard T7 does not have a wireless antenna.

Common neutral format CAD models are provided below. Right-click and select the "Save link as..." option to download STEP files.

File Attachment:

- [T7.IGS](#)
- [T7.STEP](#)
- [T7_Pro.IGS](#)
- [T7_Pro.STEP](#)
- [T7_OEM.STEP](#)
- [T7_Pro_OEM.STEP](#)
- [T7_PCB.DXF](#)

Appendix C - Firmware Revision History [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

C-1 T4 Firmware Revision History [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T4 Firmware Overview

Use the [Kipling](#) software program to load the firmware files onto a T4. Also use Kipling to identify the current firmware version on your T4.

Change Log

1.0002: Initial release firmware for the T4.

C-2 T7 Firmware Revision History [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

T7 Firmware Overview

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

1.0225:

Additions:

- *I2C lua library.
- *Cleanse Function.
- *Unit options to AIN_EF Thermistor.
- *IO_Config checksum.
- *Stream start time stamp.
- *Firmware will now prevent incompatible firmware versions from being loaded.
- *SNTP last time register.
- *SPC debugging waveform output.
- *TDAC_SPEED_THROTTLE register.
- *Timeout feature for DIO_EF Frequency In (indices 3, 4, and 11). *UART buffer protection error checks.
- *New error code to indicated that a directory search has completed. This change breaks versions 1.23 and below of the microSD Card Utility, please download version 1.24 or higher.

Changes:

- *TDAC functions will now default to a lower speed: ~100 kHz instead of 400.
- *Stream will now set the settling time for single channel stream to 10 us regardless of range and resolution settings.
- *The FIO0 jumper will now prevent startup scripts from running.
- *Minimum value of I2C throttle is now 46000. Smaller values were causing errors.
- *Changed SNTP server to pool.ntp.org
- *The USB interface will now reset itself if a packet going to the host takes more than 100 ms.
- *SNTP minimum update interval set to 10 s.
- *Updated AIN_ALL_NEGATIVE_CH to allow reads and writes of 1 to represent "differential" without requiring that the proper channel be specified.
- *The externally clocked stream divisor will now default to 1, was 0.

Fixes:

- *Bug that was disabling the watchdog time if the LEDs were set to low-power mode.
- *Bug that was causing AIN_EF Average and Threshold to erroneously throw stream errors.
- *Bug in the Lua interval function.
- *Bug that was preventing Stream_Out3 from being written to.
- *Bug that was causing DIO_EF to return old values after being reset.
- *Bug that was causing stream special channels to corrupt AIN channels.
- *Bug that was causing stream to return erroneous data for hardware counters.
- *Bug that was causing the Thermocouple Type-c calculation errors above 1600 degrees C.

Warnings:

- *Updating to this version from 1.0206 or lower will reset the startup configuration.

[Click To Expand Change Log](#)

1.0188:

Changes / Additions:

- *Added AIN_EF support for RTDs, thermistors , Type-C thermocouples, and Type-S thermocouples.
- *Updated DIO_EF modes 3, 4 and 5. In continuous mode they will measure every edge. In one-shot mode reading the value will no longer clear the result (that same value will be returned until a new measurement has been completed).
- *Added digital channel support to stream.
- *IO memory system revised. Old IOMEM functions removed from lua.
- *Added LFN support to the modbus file IO interface.
- *SD card format can now be read.
- *Added FIFO user mem.

- *device_name will now immediately update when the default value is written.
- *RTC functions will allow volatile writes when RTC is not present
- *Ethernet will now allow two connections to port 502.
- *Changed I2C buffer sizes to 128, was 40.
- *SHT enable-line control will now default to ON.
- *Added stream trigger support.

Bug fixes:

- *DACs can now handle being interrupted by stream.
- *Ethernet will now reconnect properly after link is lost.
- *Fixed a bug that was causing the RTC to set to invalid values.
- *Fixed a bug causing DIO_EF to use the wrong value for calculations when the clock source period was set to zero.
- *Fixed a bug that was causing C-R streams to get stuck in auto-recover.

1.0146: Fixed a bug that would prevent UART from being disabled once enabled. Also fixed a bug involving AIN_EF power-up default settings. On previous versions of firmware, configuring some AIN_EF channels as thermocouples, and then saving those settings as power-up defaults would cause the device to enter a continuously-resetting state on the next power reset wherein the LEDs would blink rapidly. It is possible to recover from the continuously-resetting state by jumpering a wire between SPC and FIO3, or between SPC and FIO2.

1.0144: Fixed FIO2 jumper detection. Fixed a bug causing DIO_EF options to always report zero.

1.0142: DIO_EF indices 3, 4, 5, and 11 will not reset automatically if in one-shot mode and not currently working on a measurement. Improved stream performance under certain conditions. Opening a file that does not exist using mode 'r' will no longer produce an invalid handle. Some UART registers will no longer allow writes while UART is enabled. Added SNTP update interval control register, default is 0 which is OFF.

1.0137: HTTP templating will now only be used on files with htm or html extensions. DIO_EF L2L will now be rearmed when reset. Fixed a bug causing 32-bit clock sources to be truncated to 16-bits.

1.0135: Added checksum verification to SHT communication, which is the type of communication used by the EI-1050 temperature/humidity probe. This solves a potential problem that arises with EI-1050 probes using long (>6ft) of wire.

1.0134: Added DIO_EF debounce modes 5 and 6. Fixed calendar time reads. Changed file system to FatFS, this solved the file creation issues. Added AIN_EF RTD modes, excitation modes 3,4,5 are new. Fixed a crash that would occur during lua compilation if a syntax error is encountered. Added DIO_EF index 12 (Conditional reset). Changed File IO modbus interface, this interface is not backwards compatible with code that was written to access the SD card on previous firmware versions.

1.0126: Fixed a bug causing STREAM_SETTLING to always return zero. Fixed a bug causing stream to go into digital only mode erroneously.

1.0124: Fixed a bug that was causing PWM run improperly when initialized to 0% duty cycle. Fixed error codes produced by Test_U32. Fixed bugs causing AIN_SETTLING and UART_BAUD_RATE registers to report incorrect values when read. Added AIN_EF_AutoRMS index 11.

1.0119: Fixed a stream bug that could cause overlaps to go unreported. Changed firmCounter with debounce modes. Added AIN_EF index 10, RMS_Flex.

1.0117: Bug fixes. DIO_EF set to quadrature will now return the count in READ_A_F.

1.0114: StreamNumScans will now be set to zero when stream is disabled. Increased I2C buffers to 40 bytes. Changed read_B for DIO_EF indices 3, 4, and 11. Changed analog multiplexer update sequence. Fixed a bug could cause stream to incorrectly calculate USB packet size.

1.0109: Fixed a bug causing the MSb in UART data to be cleared. The data type of the UART data registers was changed from BYTE to UINT16.

1.0108: Fixed bugs causing DIO_EF index 3, 4, and 11 to return bad readings when reading multiple signals. Added ASYNCH parity and number of stop bit options.

1.0105: Slight improvement to stream performance. Modified temperature sensor calibration constants for better accuracy between 0 and 40 °C. Added eFrequencyIn.

1.0103: Stream data analog results capped to 0xFFFFE to ensure proper auto-recover scan placement. Changed Ethernet_Apply_Settings to uint32. AIN#(0:254)_NEGATIVE_CH will now allow extended channels to specify Channel + 8 to signify differential.

1.0102: Bug fixes related to externally clocked stream. Using a packet sent over Ethernet to adjust Ethernet power will now cause an error. Added Ethernet_Apply_Settings register. Changed DIO_EF 3, 4, 5, and 11 to default to one-shot mode. WiFi IP will now be set to 0 when powered down. Increased max stream-out buffer to 16384.

1.01: New features added: DIO_EF, AIN_EF, Stream-Out, Software Watchdog, 1-Wire, UART, Scripting (Alpha), HTTP Server (Alpha). Bug Fixes.

1.0091: Hardware watchdog enabled. Unused DIO_EF binary results will now return zero instead of internal numbers.

1.0090: Reversed logic on the one-shot option of DIO_EF index 11. DIO_EF index 3,4,11 will not return period for READ_A, READ_A_F and frequency for READ_B_F. The binary version of READ_B returns 0.

1.0088: Bug fixes. Added LSb first option to SPI. DIO_EF index 11 result registers changed to match index 3 and 4.

1.0087: Added array functions for Lua. Added support for reading DIO_EF indexes 7, 8, 9, and 10 from stream.

1.0086: Bug fixes. Added DIO_EF reads to stream, only quadrature is currently supported others are on the way.

1.0083: Added an error code for invalid channels added to the stream scan list. Quadrature will now determine the initial direction during initialization. Fixed a problem causing DIO#(0:22)_EF_READ_B to throw inappropriate errors.

1.0081: Added lua "bit" library subset: band, bnot, bor, bxor, lshift, rshift. Added SPI_GO register

1.0080: Ethernet will now be powered up more slowly. Lua intervals now work with fractional values.

1.0079: Bug fixes.

1.0078: Updated stream error codes. Added Lua DEFAULT registers.

1.0074: Added asynch.

1.0071: Added streamable registers for system and core timers and a capture register for the upper 16-bits of data from 32-bit channels. Bug fixes.

1.0065: New feature to read files from the SD card through modbus. Fixed a bug causing improper number conversions during Lua compilation.

1.0059: Added save and run at startup features to scripting.

1.0058: Changed ReST to use read and write instead of R and W. HTTP bug fixes. 1-wire support added.

1.0051: Added ReST interface. Fixed a bug preventing Ethernet from clearing the watchdog. HTTP updated to chunked transfers. Default lines for SBUS will now be 0 for data, 1 for clock, 2 for power. SBUS power delay set to 20 ms.

1.0044: Fixed a bug causing bad steps from the HS converter with internal FW 1.0026. AIN_EF big changes, most modes disabled. Thermocouple mode added. Added Thermocouple options for °C and °F. K is default. Added option for AutoZero. Fixed bugs in TC calculations. Changed ambient temp adjustment to -4.3 °C. Added new Lua interval feature. Added clock_source smooth update feature for prescale and reload value. Fixed a bug preventing divisors from working on 16-bit clock sources. Other clock-source bug fixes. Added new AIN_EF system to startup config, no read function exists at this time. Clock-source smooth update will not take effect until a new roll value has been written. AIN_EF CJC default address has been set to device temperature. Bug fixes related to startup configuration. Uncalibrated devices will now return binary results. Register 43990 has been added to switch to nominal calibration. Fixed a bug causing multiple power register writes to get the wrong value. Added slope offset AIN_EF type. Changed analog startup delay to 10 ms. WiFi status will no longer return WIFI_UNPOWERED while the module is being prepared for sleep. When a script is stopped any opened files will now be closed and any lua IO mem writes will be deleted. Fixed lua interval bugs. Several changes to allow power levels to switch more gracefully.

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.

0.9421: Added stream support. EF clock source roll value is now specified as the number of counts instead of the roll value. SHT can now be read. Both H and T are locked in high-resolution mode. T is being returned in Kelvin. Changed AIN settling to a float and number of microseconds. Added stream overlap detection. Analog EF will now run normal settling time before starting the acquisition stream. DIO port registers changed back to contain inhibit in the upper byte. Changed Stream channel numbers from analog channel numbers to modbus addresses. Added custom AF mode. Channels limited to 0-15. Added WiFi constant ping and reset upon timeout. Added block to prevent reading the flash chip while WiFi initializes. Shortened WiFi rejoin times. Changed stream enable/disable to one modbus address. Added core timer reg. Fixed a C-R settling time bug. I2C Ack array moved to its own register.

0.9308: Improved response time to WiFi joining failures. AIN negative channels will now default to 199. Fixed a problem that could cause AIN settings to be applied to the wrong channel.

0.9305: Added O-Stream synchronization. Fixed several O-Stream bugs. WiFi Data-Rate parameter moved. All comm settings will be reset to factory default. Fixed problems with the PORT IO inhibit parameter.

0.9301: Due to several issues this firmware is no longer available-Added some External Flash error codes. Added support for four O-Stream channels. Fixed an error in DIO port write functions that was caused by the switch from mask to inhibit. Fixed a bug in stream that was causing scans of 3 channels to not terminate until after a forth channel. Fixed several O-Stream bugs. No more known O-Stream bugs. Fixed a bug in stream buffer status reporting.

0.9206: Added support for WiFi 3.10. Added WiFi data rate, WiFi Upgrade, and WiFi Upgrade Status registers. Reading device name now returns the device's current name rather than what is saved in flash.

0.9203: Changed WiFi options data type from u32 to u16. USB will now resume properly. Changed I2C address to lower 7 bits. Added support for WiFi 3.04, support for previous versions removed due to protocol incompatibility. Added current source value reads. Major changes to WiFi modbus interface. Moved WiFi SSID and passphrase registers. Added WiFi factory reset command. Added AF (analog features). Added new stream target (no target). To signal C-R mode. If target is set to zero firmware will send data to the source that calls stream enable. Some changes to A-R stream. Stream registers will now tolerate individual access. Changed stream to use normal AIN settings. Fixed a bug causing SHT reads to return I2C info. Amended the startup configuration functions to work with the new clock source write functions. Added stream sample signaling. Cleaned up an endianness confusion in the Ethernet stack.

0.9013: The WiFi options register can now be used to control DHCP. USB will now force a system reset if it gets zero length packets. It is now possible to turn DSFs off. DSF mode 2 will now throw an error if the output line is set high. Improved P cap and N cap and PWM readings.

0.9010: All non user-configurable WiFi settings will now be set every time the joining process is started.

0.9009: Fixed random error codes from WiFi scan start. WiFi will now respond or UDP or TCP packets. There is still only one socket.

0.9007: Changed WiFi to default to DHCP ON.

0.9006: A zero length modbus packet received over USB will cause a device reset. Changed FIO port writes over to the new inhibit system. The clear register will now reset the WDT. Added fix to change MACs with 0x89 to 0x87. Extended the MAC fix checks.

Appendix D - Packaging Information [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

D-1 T4 Packaging Information [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

Package Contents:

The normal retail packaged T4 consists of:

- T4 unit itself in red enclosure
- USB cable (6ft / 1.8m)
- Ethernet Cable (6ft / 1.8m)
- USB 5V power supply
- Screwdriver



Other package details:

There is no software CD included, so an internet connection is required to download software. Go to the T4 Support Homepage (labjack.com/support/t4) to get started.

Contact support@labjack.com for additional information on shipping.

Package size: 8.5" x 4" x 3"

Package wt: 1.0lb

D-2 T7 Packaging Information [T-Series Datasheet]

[Log in](#) or [register](#) to post comments

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
