Published on *LabJack* (https://labjack.com)

3.0 Communication [T-Series Datasheet]

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]

Modbus Map Tool

Device: All Devices

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

Usage

T-series devices are controlled by reading or writing Modbus registers as described on the Communication page.

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

Type	Integer Value
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, an 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]

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]

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 _

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. (See3.2.1 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 is 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 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.	4008	FLOAT32	R/W

Name		Start Address	Type	Access
STREAM_RESOLUTION_INDEX	The resolution index for stream readings. A larger resolution index generally results in lower noise and longer sample times.	4010	UINT32	R/W

&print=true

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 32768 bytes. Write to STREAM_BUFFER_SIZE_BYTES to set the buffer size.

Name		Start Address	Type	Access
STREAM_BUFFER_SIZE_BYTES	Size of the stream data buffer in bytes. A value of 0 equates to the default value. Must be a power of 2. Si… 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.	4012	UINT32	R/W

&print=true

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 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.	4020	UINT32	R/W

&print=true

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 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.	4014	UINT32	R/W

&print=true

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 The number of pulses per stream scan when using an external clock.	4022	UINT32	R/W

&print=true

To use externally clocked stream with LJM, see theexternally 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 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.	4024	UINT32	R/W

&print=true

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.

A more complicated stream trigger can be implemented with a Lua script. For example, a Lua script could check for an arbitrary stream trigger condition in conjunction with triggered stream being started as normal. Once the Lua script detects that the stream condition is fulfilled, it writes a pulse to a digital out (such as DIO3) which is then detected by the normal trigger (as specified by STREAM_TRIGGER_INDEX).

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

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 a 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.

In terms of timing and data rates, stream-out channels count the same as input channels so see the normal documentation of Streaming Data Rates.

Alternate waveform generation techniques are described in theWaveform 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 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.	4040	UINT32	R/W
STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.	4050	UINT32	R/W
STREAM_OUT#(0:3)_ENABLE 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.	4090	UINT32	R/W

&print=true

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 Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.	4420	UINT16	W
STREAM_OUT#(0:3)_BUFFER_F32 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.	4400	FLOAT32	W

&print=true

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 The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.	4060	UINT32	R/W
STREAM_OUT#(0:3)_SET_LOOP 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.	4070	UINT32	W

&print=true

4. Start stream

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

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

&print=true

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 The number of values in the buffer that are not currently being used.	4080	UINT32	R

&print=true

For a more thorough description of how a Stream-Out buffer works, see3.2.1.1 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 byarray 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 / scanRate` seconds to prevent unnecessary work for the hardware

3.2.1.1 Stream-Out Description [T-Series Datasheet]

T-Series Stream-Out Animation/Presentation

The T-Series Stream-Out presentation is only viewable online. Please go to our website https://labjack.com/support/datasheets/t-series/communication/stream-mode/stream-out/stream-out-description.

3.2.2 Low-Level Streaming [T-Series Datasheet]

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 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 NumChannels*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*DesiredScanRate))-1 and the actual scan rate is then 80M/(8*(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.	4002	FLOAT32	R/W

Name		Start Address	Type	Access
STREAM_NUM_ADDRESSES	The number of entries in the scanlist	4004	UINT32	R/W
STREAM_SAMPLES_PER_PACKET	Specifies the number of data points to be sent in the data packet. Only applies to spontaneous mode.	4006	UINT32	R/W
STREAM_AUTO_TARGET	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.	4016	UINT32	R/W
STREAM_SCANLIST_ADDRESS#(0:127)	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.	4100	UINT32	R/W
STREAM_ENABLE	Write 1 to start stream. Write 0 to stop stream. Reading this register returns 1 when stream is enabled. When using a triggered stream the stream is considered enabled while waiting for the trigger.	4990	UINT32	R/W

&print=true

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, see3.2 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:

```
BacklogScans = BacklogBytes / (bytesPerSample * 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 foundhere.
