United Electronic Industries

The High-Performance Alternative

DNx-SL-514 — User Manual

Synchronous Serial Interface Board
with Differential Inputs/Outputs for the
PowerDNA Cube and RACK Series Chassis

May 2018

PN Man-DNx-SL-514

See the UEI website for complete terms and conditions of sale:

http://www.ueidaq.com/cms/terms-and-conditions/

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please contact a member of our support team:

Support:

Telephone:	(508) 921-4600
Fax:	(508) 668-2350

Also see the FAQs and online “Live Help” feature on our web site.

Internet Support:

Support:	support@ueidaq.com
Website:	www.ueidaq.com
FTP Site:	ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document are subject to change without notice. Check with UEI for current status.

Table of Contents

List of Figures

Chapter 1 Introduction

This document outlines the feature set and use of the DNx-SL-514 interface board for synchronous serial interface (SSI) applications.

The following sections are provided in this chapter:

- Organization of this Manual (Section 1.1)
- SL-514 Board Overview (Section 1.2)
- Features (Section 1.3)
- Specification (Section 1.4)
- Indicators (Section 1.5)
- Device Architecture (Section 1.6)
- Device Description (Section 1.7)
- Wiring & Connectors (pinout) (Section 1.8)

1.1 Organization of this Manual

This SL-514 User Manual is organized as follows:

- **Introduction**
  Chapter 1 provides an overview of DNx-SL-514 features, device architecture, connectivity, and logic.
- **Programming with the High-Level API**
  Chapter 2 provides an overview of the how to create a session, configure the session, and interpret results with the high-level framework API.
- **Programming with the Low-Level API**
  Chapter 3 is an overview of low-level API commands for configuring and using the SL-514 series board.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for use with the DNx-SL-514 board.
- **Index**
  The index provides an alphabetical listing of the topics covered in this manual.

NOTE: A glossary of terms used with the PowerDNA Cube/RACK and I/O boards can be viewed or downloaded from www.ueidaq.com.

Manual Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

Tips are designed to highlight quick ways to get the job done or to reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.

CAUTION! *Caution advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”

Bold typeface will also represent field or button names, as in “Click **Scan Network**.”

Text formatted in `fixed` typeface generally represents source code or other text that should be entered verbatim into the source code, initialization, or other file.

Examples of Manual Conventions

Before plugging any I/O connector into the Cube or RACKtangle, be sure to remove power from all field wiring. Failure to do so may cause severe damage to the equipment.

Usage of Terms

Throughout this manual, the term “Cube” refers to either a PowerDNA Cube product or to a PowerDNR RACKtangle^{TM} rack mounted system, whichever is applicable. The term DNR is a specific reference to the RACKtangle, DNA to the PowerDNA I/O Cube, and DNx to refer to both.

1.2 SL-514 Board Overview

The DNx-SL-514 boards are high performance synchronous serial interface boards.

DNA-SL-514, DNR-SL-514, and DNF-SL-514 boards are compatible with the Cube, RACKtangle, and FLATRACK chassis respectively. These board versions are electronically identical and differ only in mounting hardware. The DNA version is designed to stack in a Cube chassis. The DNR/F versions are designed to plug into the backplane of a RACK chassis.

SL-514 boards provide four independent channels, each having overvoltage protection and opto-isolation.

1.2.1 Synchronous Serial Interface

The SL-514 offers four synchronous serial interface (SSI) channels. Each SSI channel can be programmed independently, and each SSI channel offers a master and slave port. Each master or slave port can be enabled independently.

1.2.2 Data Rates

Supported data rates are between 300 bps to 1.3 Mbps, programmed independently per channel.

The master output clock is generated with 0.1% or better accuracy.

1.2.3 Data Word Length

Supported data word lengths are programmable from 3 to 32 bits, programmed independently per channel.

1.2.4 FIFO Storage

Master (receive) data storage is provided via a 2048 word x 32 bit FIFO. Slave (transmit) data storage is provided via a 1024 word x 32 bit FIFO.

1.2.5 Software Support

Software included with the DNx-SL-514 provides a comprehensive yet easy to use API that supports all popular operating systems including Windows, Linux, real-time operating systems such as QNX, RTX, VxWorks and more.

1.3 Features

A summary of features of the SL-514 synchronous serial interface is provided below:

- Up to 4 independent synchronous serial channels. Each channel offers a master and a slave serial port.
- Electrical specifications: RS-485/422 complaint. Fully differential I/O using at RS-422 / RS-485 logic voltage levels
- Data rates: programmable up to 1.3 Mbps
- Data word length: programmable from 3 to 32 bits
- Master clock source is generated with 0.1% or better accuracy
- FIFO storage: 2048x32 FIFO for master (allowing 1024 data words along with 1024 timestamps) and 1024x32 FIFO for slave, with watermark. Asynchronous event/interrupt generated upon FIFO full.
- Start/stop all channels simultaneously
- Debouncing/glitch removal on clock and data (when debouncing is enabled, the maximum baud rate is restricted to 1.65 Mbaud)
- Protection 7 kV ESD, 350V isolation
- Power consumption 2W

1.4 Specification

Technical specifications for the DNx-SL-514 board are listed in **Table 1-1**.

Table 1-1 DNx-SL-514 Technical Specifications

SSI Ports	
Baud Rate	up to 1.3 MHz
Baud Rates available	User selectable 0.1% accuracy or better
Data Word Length	3 - 32 bits
FIFO (on each channel)	Input: 2048 word, Output: 1024 word
GENERAL SPECIFICATIONS	
Protection	7 kV ESD, 350V isolation
Input High / Low voltage	RS-422/485 compatible
Output High / Low voltage	RS-422/485 compatible
RS-422/485 termination resistors	Software selectable, 100 Ω
Electrical Isolation	350 Vrms, chan-chan and chan-chassis
Input/output buffer chip	LTC1687 or equivalent
Power consumption	< 3W
Operating range	Tested -40 to +85 °C
Humidity range	0 - 95%, noncondensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500 Hz, broad-band random
Shock *IEC 60068-2-27*	50 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
MTBF	350,000 hours

1.5 Indicators

The DNx-SL-514 indicators are described in **Table 1-2** and illustrated in **Figure 1-1**.

Table 1-2 SL-514 Indicators

LED Name	Description
RDY	Indicates board is powered up and operational
STS	Indicates which mode the board is running in: • **OFF**: Configuration mode, (e.g., configuring channels, running in point-by-point mode) • **ON**: Operation mode

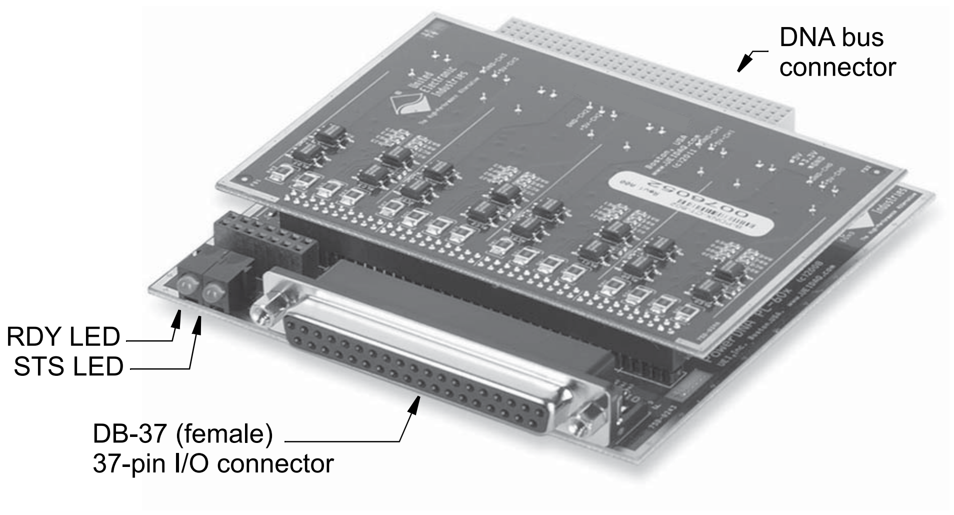


Figure 1-1 Photo of DNA-SL-514 Synchronous Serial Board

1.6 Device Architecture

The SL-514 board offers 4 independent synchronous serial interface (SSI) channels, and each channel provides a master and slave SSI port. All inputs and outputs are optically isolated and overvoltage protected.

Figure 1-2 shows a block diagram of the SL-514. Refer to Section 1.8 for pinout.

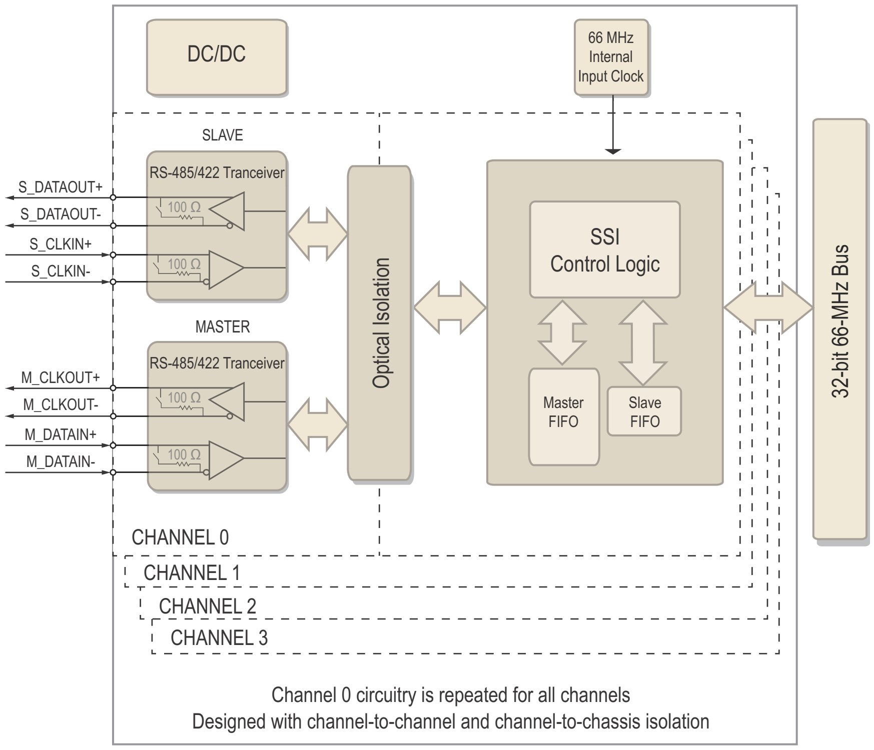


Figure 1-2 Block Diagram of SL-514

1.7 Device Description

The SL-514 is a synchronous serial interface (SSI) that supports point-to-point serial transmissions between a master controller and a slave device.

The SL-514 includes 4 SSI channels, and each SSI channel provides a master port consisting of differential CLKOUT and DATAIN pins and a slave port consisting of differential CLOCKIN and DATAOUT pins.

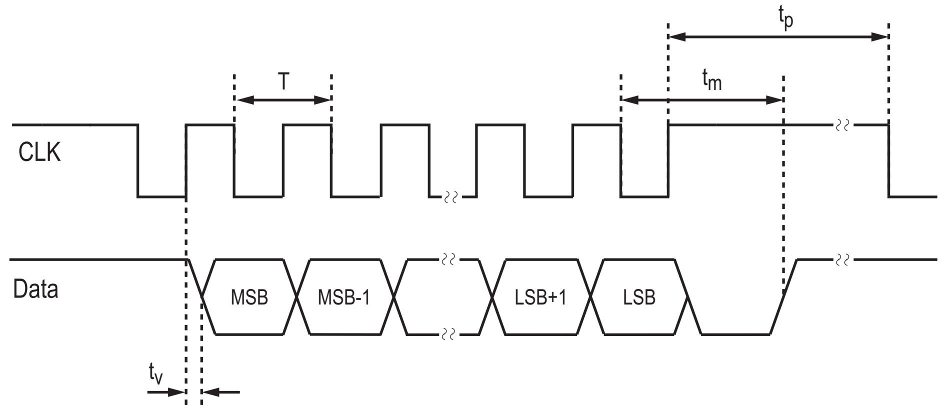
1.7.1 Master/Slave Device Description

The SL-514 master device generates an output clock (CLKOUT) at the user-specified baud rate. Data is received by the master device on DATAIN as most significant bit (MSB) first, with no particular start or stop sequence required (no start bit or stop bit).

The slave device receives an input clock on CLOCKIN.The clock transition causes the slave device to start shifting the serial data word out its slave transmitter (DATAOUT pin), MSB first.

The data word consists of a user-programmed number of bits, from 3 to 32 bits.

Refer to **Figure 1-3**.



t_v = data delay time
t_p = pause time
t_m = transfer timeout (monoflop time)
T = 1/baud rate

Figure 1-3 SSI Transmission Waveform

SSI data bit transfers occur using the following transmission sequence (refer to **Figure 1-3**):

- Clock and data are held high when devices are in an idle state.
- When the master controller needs data, it starts outputting its clock pulse train.
- When the slave device is in idle and detects a low on the clock pin, it shifts the MSB of the data word queued for transmission out its DataOut pin on the rising edge of the clock, and subsequently each next data bit transmits on the next rising edge of the master clock until the LSB shifts out.
- The master controller latches each incoming bit on the falling edge of its clock.
- When the full word is received by the master controller, the master holds its clock high for the user-programmed pause time (t_p).
- When the full word is transmitted out of the slave device, the slave holds its data low for a user-programmable transfer timeout period (t_m), which starts on the falling edge of the clock synchronized to the LSB of data.
- After the t_m period, the slave drives its DataOut pin high.

The master controller and slave device use the pause time (t_p) duration to reset their state machines to idle, and setup for the next word for transmission/ reception.

NOTE: A master defect protocol error will occur if the slave device does not drive its data output high at the end of the t_p period or low between the last rising edge of the clock (plus t_v delay * 2) and end of t_m period.
A slave defect protocol error will occur if the master clock is not high for the full t_p period (clock high is too short vs the programmed value).

1.7.2 Serial Port Configuration Options

Baud rate. Baud rates are customizable per channel. The master and slave port on each channel runs the same baud rate (refer to **T** in **Figure 1-3**).

Data word length. The number of bits in the data word can be customized by the user. Supported word lengths are 3 bits (MSB, MSB-1, LSB) to 32 bits.

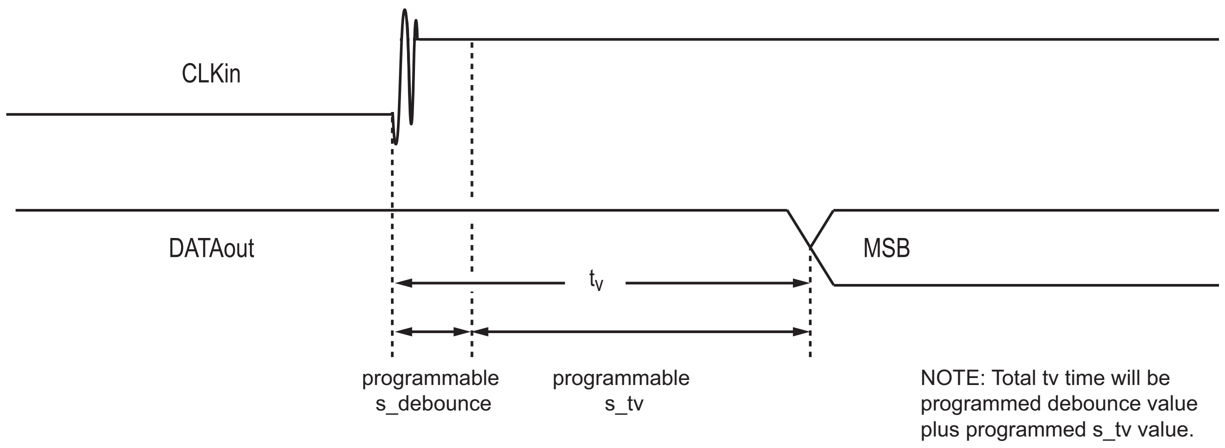

CLKin
DATAout
MSB
t_v
programmable s_debounce
programmable s_tv
NOTE: Total tv time will be programmed debounce value plus programmed s_tv value.

Figure 1-4 Example of Debouncing and Tv time Delays

Debouncing. Debouncing circuits can be enabled on each master and slave port on each channel. Debouncing is available on both clock and data signals.

T_v time delay. Tv represents the time between the rising edge of the clock and the transmission of a data bit (on the slave side) or the latching of data bits (on the receive/master side). The **tv** delay can be programmed independently per channel.

The `m_tv` or `s_tv` parameter can be programmed as 0 to 255 number of 66 MHz clocks.

T_p time delay. Tp represents the time between the rising edge of the clock after the LSB transmission and the falling edge of the first clock of the next data word (refer to **tp** in **Figure 1-3**).

The `m_tp` or `s_tp` parameter can be programmed a 32-bit number of 66 MHz clocks.

T_m time delay. The transfer timeout delay, also called the monoflop time, represents the time delay between the DataOut pin driving low after the LSB is transmitted and the time the DataOut drives high, signifying the port has entered an IDLE state (refer to **tm** in **Figure 1-3**).

The `m_tm` or `s_tm` parameter can be programmed as up to 65535 number of 66 MHz clocks. (The minimum tm is based on your system and cannot be less than half the baud rate, T/2).

Termination resistors. Termination can be enabled or disabled per channel (refer to Section 1.7.5 for more information).

Trigger. Master and slave ports can set a data transfer trigger condition per channel: start immediately as soon as data is in the FIFO buffer, or wait for a global trigger.

Timestamps. Master ports can latch a timestamp to be stored with each received serial data word (up to 1024 serial data words and 1024 accompanying timestamps).

Master clock source. Master ports can assign which clock source to use to derive their baud rate clock:

- derived from 66 MHz system clock (default)*
- derived from onboard PLL*

NOTE: *In most cases the baud rate should be derived from the 66 MHz system clock, which is the default configuration. However, in some instances when programming baud rates greater then 1 MHz, users may choose to use the onboard PLL to derive the clock rate, which can yield a finer frequency granularity.

For either master clock source, the baud rate is generated by dividing down the clock source. When using the 66 MHz system clock, this will result in dividing down from 33 MHz. If your application requires a baud rate that is not evenly divisible by 33 MHz, the actual baud you will get is one that is as close as possible to your requested rate *and* evenly divisible by 33 MHz. If your application requires finer granularity, the PLL can be used.

1.7.3 FIFO Operation & Timestamping

Data storage for each master controller is provided by a 2048 x 32-bit FIFO. This supports storage for 1024 received words and allows each word to be tagged with a timestamp, if the timestamping option is enabled. If timestamping is not enabled, all 2048 locations can be used for data. Received words are stored in the FIFO until read by CPU; users can program master data transfers based on a programmable FIFO watermark, FIFO timeout, or FIFO overrun.

Data storage for each slave controller is provided by a 1024 x 32-bit FIFO. Users can store up to 1024 words in the slave output FIFO at a time.

1.7.4 Error Checking & Status Reporting

The SL-514 will report errors and status based on the following conditions:

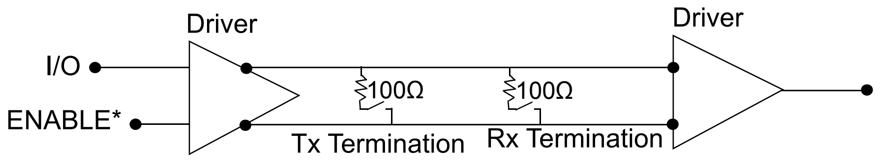
- Slave detected master timing error (SL514_CSTS_TXERR):
  - if master clock pulses low during pause time (t_p) delay (transmission starts too early)
- Master detected slave timing error (SL514_CSTS_RXERR):
  - if slave data pulses high during transfer timeout/monoflop time (t_m) delay
  - if slave drives data low before rising edge of first master clock after idle state
- Slave TX FIFO empty (SL514_CSTS_TXFE)
- Master RX FIFO empty (SL514_CSTS_RXFF)
- Slave channel busy, sending data (SL514_CSTS_TXBSY)
- Master channel busy, receiving data (SL514_CSTS_RXBSY)
- Slave TX FIFO is below watermark, currently set to half the FIFO size (SL514_CSTS_TXFHF)
- Slave RX FIFO is above watermark, currently set to half the FIFO size (SL514_CSTS_RXFHF)

The SL-514 master controllers only store SSI words that are received without timing errors in the FIFO.

1.7.5 Termination

The SL-514 features termination resistors on both the receiver and transmitter lines to provide a driver load impedance of 100 Ω.

Refer to **Figure 1-5**.



* Note: Only used channels are enabled. Unused channels are automatically disabled by the firmware.

Figure 1-5 Settable Termination Circuit Diagram

Each of these lines can be set through software in the low-level API. Refer to **Chapter 3** for more information.

1.7.6 Electrical Specification for Serial Port Lines

The SL-514 is compliant with RS-422 and RS-485 standards for electrical characteristics of drivers and receivers used in serial communication.

Refer to TIA/EIA-422 and TIA/EIA-485 Standards documentation for more information.

1.8 Wiring & Connectors (pinout)

Figure 1-6 below illustrates the pinout of the SL-514.

Each of the four channels on the SL-514 can be configured as a master SSI port and/or as a slave SSI port.

The SL-514 board uses a 37-pin D-sub connector.

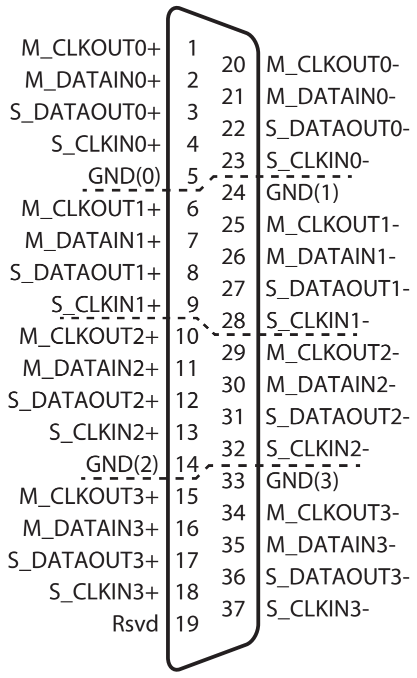


Figure 1-6 Pinout Diagram of the SL-514 Board

All signals are referenced relative to isolated ground (iGND).

NOTE: If you are using a accessory panel with the SL-514, please refer to the Appendix for a description of the panel.

Chapter 2 Programming with the High-Level API

This chapter provides the following information about using the UeiDaq Framework High-level API to control the DNx-SL-514:

- About the High-level Framework (Section 2.1)
- Creating a Session (Section 2.2)
- Configuring the Resource String (Section 2.3)
- Configuring an SSI Master Port (Section 2.4)
- Configuring an SSI Slave Port (Section 2.5)
- Configuring Minimum Pulse Widths (Section 2.6)
- Configuring the Timing (Section 2.7)
- Configuring Timestamps (Section 2.8)
- Reading Data (Section 2.9)
- Writing Data (Section 2.10)
- Cleaning-up the Session (Section 2.11)

2.1 About the High-level Framework

UeiDaq Framework is object oriented and its objects can be manipulated in the same manner from different development environments, such as Visual C++, Visual Basic, or LabVIEW.

UeiDaq Framework is bundled with examples for supported programming languages. Examples are located under the UEI programs group in:

- *Start » Programs » UEI » Framework » Examples*

The following sections focus on C++ API examples, but the concept is the same regardless of which programming language you use.

Please refer to the “UeiDaq Framework User Manual” for more information on use of other programming languages.

2.2 Creating a Session

The Session object controls all operations on your PowerDNx device. Therefore, the first task is to create a session object:

```
// create a session object for input, and a session object for output
CUeiSession ssiSession;
```

2.3 Configuring the Resource String

UeiDaq Framework uses resource strings to select which device, subsystem and channels are used within a session. The resource string syntax is similar to a web URL:

```
<device class>://<IP address>/<Device Id>/<Subsystem><Channel list>
```

For PowerDNA and RACKtangle, the device class is **pdna**.

For example, the following resource string selects SSI ports 0,1,3 on device 1 at IP address 192.168.100.2: “pdna://192.168.100.2/Dev1/ssi0,1,3”

The SL-514 is programmed using the subsystem **ssi** to configure channels as SSI channels.

2.4 Configuring an SSI Master Port

Use the method `CreateSSIMasterPort()` to configure one or more channel(s) as a master SSI port.

The following call configures SSI master ports 0 and 1 of a SL-514 set as device 1:

```
// Configure session’s ports

ssiSession.CreateSSIMasterPort(“pdna://192.168.100.2/Dev1/ssi0,1”,
                               125000,
                               8,
                               TRUE,
                               FALSE,
                               10000.0,
                               16.03,
                               0.45);
```

It configures the following parameters:

- **Bits per second**: the number of bits per second transferred over the synchronous port (unsigned integer)
- **Word size**: The number of bits per word (3 to 32) (unsigned integer)
- **Clock Enable**: Enable or disable the clock output (boolean)
- **Termination Enable**: Enable or disable termination resistor (boolean)
- **Pause Time**: Specifies the time delay (t_p) in microseconds between two consecutive clock sequences from the master (double)
- **Transfer Timeout**: Specifies the minimum time (t_m) in microseconds required by the slave to realize that the data transmission is complete (double)
- **Bit Update Time**: Specifies the time (t_v) in microseconds from the rising clock edge to the data in transitioning high to low or low to high (double)

2.5 Configuring an SSI Slave Port

Use the method `CreateSSISlavePort()` to configure one or more channel(s) as a slave SSI port.

The following call configures SSI slave ports 0 and 1 of a SL-514 set as device 1:

```
// Configure session’s ports

ssiSession.CreateSSISlavePort(“pdna://192.168.100.2/Dev1/ssi0,1”,
                              125000,
                              8,
                              TRUE,
                              FALSE,
                              10000.0,
                              16.09,
                              0.0);
```

It configures the following parameters:

- **Bits per second**: the number of bits per second transferred over the synchronous port (unsigned integer)
- **Word size**: The number of bits per word (3 to 32) (unsigned integer)
- **Transmit Enable**: Enable or disable the data output (boolean)
- **Termination Enable**: Enable or disable termination resistor (boolean)
- **Pause Time**: Specifies the time delay (t_p) in microseconds between two consecutive clock sequences from the master (double)
- **Transfer Timeout**: Specifies the minimum time (t_m) in microseconds required by the slave to realize that the data transmission is complete (double)
- **Bit Update Time**: Specifies the time (t_v) in microseconds from the rising clock edge to the data out transitioning high to low or low to high (double)

2.6 Configuring Minimum Pulse Widths

Users can optionally configure the SL-514 to only recognize signal transitions that last longer than a user-programmed minimum pulse width setting. The default setting for the minimum pulse width is 0, which results in no restriction.

To set the minimum pulse width parameter:

- On master ports, use the `SetMinimumDataPulseWidth()` method.
- On slave ports, use the `SetMinimumClockPulseWidth()` method.

The minimum pulse width is programmed in µs.

To read the current setting of the minimum pulse width parameter:

- On master ports, use the `GetMinimumDataPulseWidth()` method.
- On slave ports, use the `GetMinimumClockPulseWidth()` method.

NOTE: This value can be incremented in ~15 ns steps and should be set to the smallest possible setting that solves an issue of signal ringing (debounces the signal).

The following code reads the minimum pulse width parameters for the master port on channel 1 and the slave port on channel 2:

```
// Read minimum pulse width parameters

CUeiSSIMasterPort* masterMPW;
CUeiSSISlavePort* slaveMPW;
double slaveValue, masterValue;

masterMPW = dynamic_cast<CUeiSSIMasterPort*>(ssiSession.GetChannel(1));
masterValue = masterMPW->GetMinimumDataPulseWidth();

slaveMPW = dynamic_cast<CUeiSSISlavePort*>(ssiSession.GetChannel(2));
slaveValue = slaveMPW->GetMinimumClockPulseWidth();
```

The following code sets the minimum pulse width parameters for the master port on channel 1 to 50 ns and the slave port on channel 2 to 100 ns:

```
// Set minimum pulse width parameters

masterMPW = dynamic_cast<CUeiSSIMasterPort*>(ssiSession.GetChannel(1));
masterMPW->SetMinimumDataPulseWidth(0.050);

slaveMPW = dynamic_cast<CUeiSSISlavePort*>(ssiSession.GetChannel(2));
slaveMPW->SetMinimumClockPulseWidth(0.100);
```

2.7 Configuring the Timing

The application must configure the SL-514 to use the “messaging” timing mode.

A message is represented by an array of bytes.

The SL-514 can be programmed to wait for a certain number of bytes to be received before notifying the session.

It is also possible to program the maximum amount of time to wait for the specified number of bytes before notifying the session.

The following sample shows how to configure the messaging I/O mode to be notified when 10 bytes have been received or every second, whichever is less. (Note that if the serial port receives less than 10 bytes per second, it will return whatever number of bytes are available every second).

```
// configure timing of serial port

ssiSession.ConfigureTimingForMessagingIO(10, 1.0);
```

2.8 Configuring Timestamps

Users can check whether timestamping is enabled and optionally enable it on each received frame. Timestamping can only be enabled on master ports.

- Use the `IsTimestampingEnabled()` method to check whether timestamping is enabled. The method returns a boolean true or false.
- Set the `EnableTimestamping` property to true to enable timestamping for input data on one or more master port(s). Set the property to false to disable timestamping.

The following code enables timestamping on port 0:

```
// Enable timestamping on port 0 (channel0) if not already enabled

CUeiSSIMasterPort* master0;
bool TSenabled;

master0 = dynamic_cast<CUeiSSIMasterPort*>(ssiSession.GetChannel(0));
TSenabled = master0->IsTimestampingEnabled();

if(!TSenabled) master0->EnableTimestamping(true);
```

NOTE: When reading data, the timestamp is read directly following the data word.

2.9 Reading Data

Reading data from the SL-514 is done using a *reader* object. As there is no multiplexing of data (contrary to what's being done with AI, DI, or CI sessions), you need to create one reader object per master port to be able to read from each port in the port list.

NOTE: The number of bits read in the input word is programmed as `Word size` (3-32 bits) in the `CreateSSIMasterPort()` API.

The following sample code shows how to create a reader object tied to port 1 (`ssi1`) and read up to 10 data words from that `ssi` port.

```
// Create a reader and link it to the session’s stream, port 1

reader = new CUeiSSIReader(ssiSession.GetDataStream(), 1);

// we’ll want to store for 10 data words (uInt32-sized)

uInt32 dwords[10];

// enable grayEncoding (true), read up to 10 data words

reader->Read(true, 10, dwords, &numRead);
```

2.10 Writing Data

Writing data to the SL-514 is done using a *writer* object. As there is no multiplexing of data (contrary to what's being done with AO, DO, or CO sessions), you need to create one writer object per slave port to be able to write to each port in the port list.

NOTE: The number of bits written in the output word is programmed as `Word size` (3-32 bits) in the `CreateSSISlavePort()` API.

The following sample code shows how to create a writer object tied to port 0 (`ssi0`) and send a frame of 128 data words to the SSI port.

```
// Create a writer and link it to the session’s stream, port 0

writer = new CUeiSSIWriter(ssiSession.GetDataStream(), 0);

// initialize 128 uInt32 data words to 0x34 that we will write out

uInt32 dwords[128];
for(int i=0; i< 128; i++) dwords[i]=0x34;

// enable grayEncoding (true), write 128 data words,
//    numWritten contains number of data words actually sent

writer->Write(true, 128, dwords, &numWritten);
```

2.11 Cleaning-up the Session

The session object will clean itself up when it goes out of scope or when it is destroyed. To reuse the object with a different set of channels or parameters, you can manually clean up the session as follows:

```
// clean up the sessions

ssiSession.CleanUp();
```

Chapter 3 Programming with the Low-Level API

This chapter provides the following information about programming the SL-514 using the low-level API:

- About the Low-level API (Section 3.1)
- Low-level Functions (Section 3.2)
- Low-level Programming Techniques (Section 3.3)

3.1 About the Low-level API

The low-level API provides direct access to the DAQBIOS protocol structure and registers in C. The low-level API is intended for speed-optimization, when programming unconventional functionality, or when programming under Linux or real-time operating systems.

When programming in Windows OS, however, we recommend that you use the UeiDaq high-level Framework API (see **Chapter 2**). The Framework extends the low-level API with additional functionality that makes programming easier and faster.

For additional information regarding low-level programming, refer to the "PowerDNA API Reference Manual" located in the following directories:

- On Linux systems:
  <PowerDNA-x.y.z>/docs
- On Windows systems:
  Start » All Programs » UEI » PowerDNA » Documentation

3.2 Low-level Functions

Table 3-1 provides a summary of SL-514-specific functions. All low-level functions are described in detail in the PowerDNA API Reference Manual.

Table 3-1 Summary of Low-level API Functions for DNx-SL-514

Function	Description
`DqAdv514Config`	Configures synchronous serial interface channels: • enables master/slave ports • sets clock source • sets baud rate • sets number of bits in transmission/reception word • sets data delay time (tv) • sets transfer timeout time (tm) • sets time delay between transmissions (tp, pause time) • sets trigger condition to read/fill FIFO • enables debouncing • enables termination resistors
`DqAdv514SetPLL`	Programs the PLL clock on the SL-514 board (for baud rate programming), if the PLL source is selected. *The default clock source for baud rate programming is the 66 MHz clock, not the PLL.*
`DqAdv514Status`	Reads error states/status states

Table 3-1 Summary of Low-level API Functions for DNx-SL-514 (Cont.)

Function	Description
`DqAdv514Enable`	Enables or disables serial channels on the board
`DqAdv514WriteFIFO`	Writes data out slave SSI port
`DqAdv514ReadFIFO`	Reads data into master SSI port; returns status information

3.3 Low-level Programming Techniques

Application developers are encouraged to explore the existing source code examples when first programming the SL-514. Sample code provided with the installation is self-documented and serves as a good starting point.

Code examples are located in the following directories:

- On Linux systems: <PowerDNA-x.y.z>/src/DAQLib_Samples
- On Windows: *Start » All Programs » UEI » PowerDNA » Examples*

Code examples specifically for the SL-514 have `514` specified in the name, (i.e., `Sample514.c`).

SL-514 can be operated using the immediate (point-to-point) data acquisition protocol. `Sample514.c` provides an example of acquiring data using this mode.

3.3.1 Configuring Serial Interface

The SL-514 master and slave ports are configured using the `DqAdv514Config()`API:

```
DqAdv514Config(
     int hd,            // Handle to IOM received from DqOpenIOM()
     int devn,          // Board device # inside the IOM chassis
     int chan,          // Channel to be configured
     pL514_CONFIG config);// Structure to hold config settings
```

The `pL514_CFG` structure consists of the following elements:

```
// SSI channel configuration
typedef struct {
     uint32 ch_cfg; // channel configuration
     uint32 flags;  // <Reserved>

     uint32 clk_source;   // clock source for master
     uint32 baud_rate;   // baud rate for master (100Hz to 1.3 MHz)

     // Master SSI setting (sends clocks, receives data)
     uint32 m_word_sz;  // master word size (3 to 32 bits)
     uint32 m_debounce; // debouncing settings (0=bypass)
     uint32 m_trigger;  // Tx trigger source
     uint32 m_Tv;       // Tv, master
     uint32 m_Tp;       // Tp, master
     uint32 m_Tm;       // Tm, master

     // Slave SSI setting (sends data upon receiving clocks)
     uint32 s_word_sz;  // master clock size
     uint32 s_debounce; // debouncing settings
     uint32 s_trigger;  // Tx trigger source
     uint32 s_Tv;       // Tv, master
     uint32 s_Tp;       // Tp, master
     uint32 s_Tm;       // Tm, master
     } L514_CONFIG, *pL514_CONFIG;
```

Table 3-2 lists configuration options for each element. All parameters listed in **Table 3-2** are uint32.

Table 3-2 SL-514 Configuration Options

Configuration Parameter	Options
`ch_cfg`	The following #define settings can be logically ORed together for the channel configuration (`ch_cfg)` parameter: • `L514CFG_SLAVE_EN`: enable slave port for the channel • `L514CFG_MASTER_EN`: enable master port for the channel • `L514CFG_MASTER_TS`: add timestamp to the master data • `L514CFG_OUTPUTS_TERMN`: enable termination on channel outputs • `L514CFG_INPUTS_TERMN`: enable termination on channel inputs • `L514CFG_SLAVE_DATA_EN`: enable line drive Tx • `L514CFG_MASTER_CLK_EN`: enable line drive Rx • `L514CFG_SLAVE_Tprm`: enable slave T parameters (0 = use defaults) • `L514CFG_MASTER_Tprm`: program master T parameters (0 = use defaults)
`flags`	<Reserved>
`clock_source`	Sets clock source to use to derive master clock (for baud): • `L514CFG_CLK_BASE`: use system 66MHz clock and divide it • `L514CFG_CLK_PLL`: use on-board PLL as a clock source
`baud_rate`	uint32 300 baud to 1.3 Megabaud (uint32)
`m_word_sz`	master 3 to 32 bits in the word (uint32)
`m_debounce`	master debouncing settings: • 0=bypass • 1 thru 15 = 4 thru 18, 15 ns clocks, (i.e., programming a “1” results in 4*15ns, or ~60 ns debouncing delay)
`m_trigger`	master trigger source that allows data to start transmission: • `L514CFG_TRIG_IMM`: start immediately after enable, transmit when the data is in the buffer • `L514CFG_TRIG_GLOBAL`: wait for the global trigger to start
`m_Tv`	master tv time delay: see Figure 1-3 on page 8 and refer to Section 1.7.2 on page 9 for more information.
`m_Tp`	master pause time delay setting: see Figure 1-3 on page 8 and refer to Section 1.7.2 on page 9 for more information.
`m_Tm`	master transfer timeout/monoflop time setting: see Figure 1-3 on page 8 and refer to Section 1.7.2 on page 9 for more information.
`s_word_sz`	slave 3 to 32 bits in the word (uint32)
`s_debounce`	slave debouncing settings: • 0=bypass • 1 thru 15 = 4 thru 18, 15 ns clocks, (i.e., programming a “1” results in 4*15ns, or ~60 ns debouncing delay)

Table 3-2 SL-514 Configuration Options

Configuration Parameter	Options
`s_trigger`	slave trigger source that allows data to start transmission: • `L514CFG_TRIG_IMM`: start immediately after enable, transmit when the data is in the buffer • `L514CFG_TRIG_GLOBAL`: wait for the global trigger to start
`s_Tv`	slave tv time delay: see Figure 1-3 on page 8 and refer to Section 1.7.2 on page 9 for more information.
`s_Tp`	slave pause time delay setting: see Figure 1-3 on page 8 and refer to Section 1.7.2 on page 9 for more information.
`s_Tm`	slave transfer timeout/monoflop time setting: see Figure 1-3 on page 8 and refer to Section 1.7.2 on page 9 for more information.

3.3.2 Setting the Baud Rate Using the PLL

The SL-514 can use on-board phase locked loop (PLL) circuitry to generate the master output clock baud rate, or alternatively, divide a 66 MHz onboard system clock to generate the master clock.

This section provides an overview of how to setup the PLL. To use this function, you must use `L514CFG_CLK_PLL` as the `clock_source` configuration parameter (`see` Section 3.3.1 on page 21 for configuration options.)

```
DqAdv514SetPll(
     int hd,           // Handle to IOM received from DqOpenIOM()
     int devn,         // Board device # inside the IOM chassis
     int chan,         // Channel to be configured
     float baudrate,   // Desired baud rate
     float* &actual_baud);// Actual baud rate
```

This function returns an `actual_baud` parameter.

To set the baud rate, the user-defined `baudrate` value is used to program the PLL. The PLL dividers may not be able to produce the exact programmed rate; in which case, the value closest to the user-programmed rate is used. Users can check the `actual_baud` parameter to know what baud rate is used.

3.3.3 Enabling SSI Channels

The SL-514 offers four independent SSI channels, and each channel has a master port and a slave port. Each of the four channels can be enabled with the `DqAdv514Enable()` function described below.

NOTE: Individual master and slave ports are enabled or disabled through the `DqAdv514Config()` API (Section 3.3.1).

```
DqAdv514Enable(
      int hd,            // Handle to IOM received from DqOpenIOM()
      int devn,          // Board device # inside the IOM chassis
      int channel_mask);// Bitmask of channels:
                         //  “1” to enable (or keep running),
                         //  and “0” to disable.
```

As an example, setting `channel_mask = 0x3` will enable channel 0 and channel 1.

3.3.4 Writing Slave Data for Transmit

Each slave port on the SL-514 has a 1024 FIFO for holding data to be output.

The FIFO can be filled using the `DqAdv514WriteFIFO()` API.

```
DqAdv514WriteFIFO(
      int hd,            // Handle to IOM received from DqOpenIOM()
      int devn,          // Board device # inside the IOM chassis
      int chan,          // Channel to be configured
      int flags,         // <Reserved>
      int size,          // size of transmit buffer
      uint32* buffer,    // Pointer to the array of transmit data
      int* written,      // Number of data words stored in FIFO
      int* available);   // Amount of free space in FIFO
```

3.3.5 Reading Received Master Data

Each master port on the SL-514 has a 2048 FIFO for holding data that was received and for optionally holding timestamps associated with each piece of data.

The FIFO can be read using the `DqAdv514ReadFIFO()` API.

```
DqAdv514ReadFIFO(
      int hd,            // Handle to IOM received from DqOpenIOM()
      int devn,          // Board device # inside the IOM chassis
      int chan,          // Channel to be configured
      int flags,         // <Reserved>
      int size,          // maximum receive buffer size
      uint32* buffer,    // Pointer to the array for receive data
      int* retrieved,    // Number of data words retrieved from the FIFO
      int* available,    // Amount of free space in FIFO
      uint32* status);   // Status of board
```

The `size` parameter limits the number of samples that will be copied into the receive `buffer.` The maximum allowed value is `DQ_PL_601_LISTSZ` (350), which is the maximum number allowed in a packet.

The status parameter that is returned represents the status of the one channel programmed with this function (`chan`).

Alternatively, the `DqAdv514Status()` API reads status conditions from all channels listed in a bitmask of channels. Refer to Section 3.3.6 for more information about reading status and for a list of status conditions that can be read from channels.

3.3.6 Reading Status

The `DqAdv514Status()` API is used to read channel status. This API allows you to read status information from one or more channels at a time.

```
DqAdv514Status(
      int hd,            // Handle to IOM received from DqOpenIOM()
      int devn,          // Board device # inside the IOM chassis
      int channel_mask, // Bitmask of channels:
                         //  “1” to capture status information,
                         //  and “0” to ignore.
      pL514_STATUS status);// Pointer to store array of L514_STATUS
                            //  1 for each channel enabled in channel_mask
```

The `channel_mask` parameter identifies which channels you will read. For example, `channel_mask=0x9;` will read status on channel 0 and channel 3.

The returned `status` parameter is an array. For example, if your `channel_mask` indicates 2 channels, you will receive an array of 2.

Refer to **Table 3-3** for a list of status values returned, and refer to Section 1.7.4 on page 11 for additional descriptions of error conditions and status bits.

Status bits are returned as type `pL514_STATUS.` The `pL514_STATUS` structure consists of two elements:

```
 // SSI channel status
```

```
typedef struct {
     uint32 ch_status; // channel status
     uint32 flags;     // <Reserved>
     } L514_STATUS, *pL514_STATUS;
```

The `ch_status` element is encoded with status flags described in **Table 3-3**.

NOTE: Note that some bits are sticky and are cleared after the status is read, and some are static and return the current status value. Sticky bits are indicated in the bit descriptions.

#defined Value	Bit Representation	Description
SL514_CSTS_TXERR	(1L<<19)	Slave TX Timing Error: • 1= error detected, master clock arrived too early (sticky bit, cleared after read)
SL514_CSTS_RXERR	(1L<<18)	Master RX Timing Error: • 1= error detected, slave drove RX data high during mono-flop time (sticky bit, cleared after read)
SL514_CSTS_TXFE	(1L<<17)	Slave TX FIFO is empty: • 1= FIFO empty (sticky bit, cleared after read)
SL514_CSTS_RXFF	(1L<<16)	Master RX FIFO is full: • 1= FIFO full (sticky bit, cleared after read)
SL514_CSTS_TXBSY	(1L<<3)	Slave TX is transmitting data: • 1= sending data
SL514_CSTS_RXBSY	(1L<<2)	Master RX is receiving data: • 1= receiving data
SL514_CSTS_TXFHF	(1L<<1)	Slave TX is below watermark: • 1= below watermark
SL514_CSTS_RXFHF	(1L<<0)	Master RX is above watermark: • 1= above watermark

Table 3-3 SL-514 Status Bit Meanings

Appendix

A.1 Accessories

The following cables and STP boards are available for the SL-514 board.

DNA-CBL-37

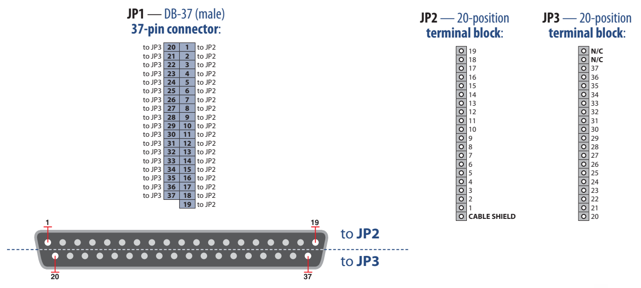
3ft, 37-way flat ribbon cable; connects SL-514 to panels to DNA-STP-37.

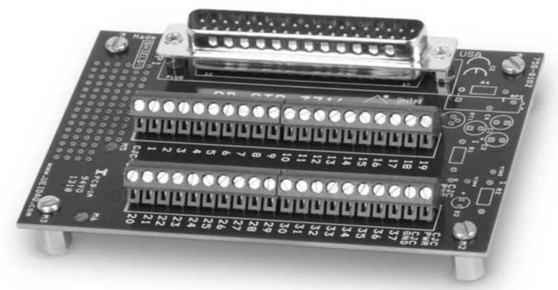
DNA-CBL-37S

3ft, 37-way shielded cable; connects SL-514 to panels to DNA-STP-37.

DNA-STP-37

37-way screw terminal panel.





DNA-STP-37D

37-way direct-connect screw terminal panel.

Index