United Electronic Industries

The High-Performance Alternative

DNx-SL-504/ DNx-SL-504-801

—

User Manual

4-Channel RS-232 or RS-422/423/485 (serial port) boards for the PowerDNA Cube and RACK series chassis

Support for Synchronous Serial Data Communication protocols (SDLC & HDLC protocols)

May 2020

PN Man-DNx-SL-504

CE

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 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
Web-Site: 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-504 boards for synchronous serial-line communication applications.

The DNx-SL-504-1 board version supports RS-232, RS-422, and RS-485 recommended standards for full-duplex serial communication.

The DNx-SL-504-801 board version supports RS-232, RS-422, and RS-485 recommended standards for both full-duplex serial communication and RS-485/422 two-wire, half-duplex operation.

The following sections are provided in this chapter:

- Organization of Manual (Section 1.1)
- SL-504 Interface Board Overview (Section 1.2)
- Features (Section 1.3)
- Indicators (Section 1.4)
- Specification (Section 1.5)
- Serial Communication (Section 1.6)
- Architecture (Section 1.7)
- Wiring & Connectors (pinout) (Section 1.8)

1.1 Organization of Manual

This SL-504 User Manual is organized as follows:

- **Introduction**
  This section provides an overview of the SL-504 synchronous serial line communication interface features, device architecture, and connectivity.
- **Programming with the High-Level API**
  This chapter provides an overview of the how to create a session, configure the session, and format relevant data with the Framework API.
- **Programming with the Low-Level API**
  This chapter describes low-level API commands for configuring and using the SL-504 series board for serial operating modes.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for use with the SL-504 board.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

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™ 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-504 Interface Board Overview

The DNx-SL-504 is a 4-port interface board for synchronous serial communications for the Cube and RACK series chassis.

DNA-SL-504, DNR-SL-504, and DNF-SL-504 boards are compatible with the UEI Cube, RACKtangle, and FLATRACK chassis respectively. These board versions are electronically identical except for the 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.

1.2.1 Standards and Protocols

Each port on the SL-504 is independently configurable as RS-232, RS-485, RS-422/423 set for synchronous communications. Ports are fully isolated from each other as well as from the Cube or RACK chassis.

The DNx-SL-504 is based on the Zilog Z16C32 serial controller chip and supports synchronous serial protocols including high-level data link control (HDLC) and synchronous data link control (SDLC). The HDLC/SDLC interface provides full access to serial frames, which allows software to determine how to handle retries. The RS-485/422 implementation provides transmit and receive data, sync and clock interfaces. Implementations can also support CTS and DCD signals.

1.2.2 Baud Rates & Bit Configuration

The maximum transfer rate in RS-485/422 and RS-232 modes are 4 Mbaud and 230 kbaud respectively.

The on-board UART supports 5, 6, 7, or 8 data bits, plus optional even or odd parity. The transmitters will also supply 1, 2, or fractional stop bits per character and can provide a break output at any time.

1.2.3 Network Topologies

The DNx-SL-504 boards are compatible with RS-232 point-to-point or RS-485 network applications. The ports are driven by the Exar SP506CM-L series drivers and provide a wide variety of I/O configurations.

1.2.4 Electrical & Environmental Specifications

As with all UEI PowerDNA boards, the DNx-SL-504 can be operated in harsh environments and has been tested at 5*g* vibration, 50*g* shock, -40 to +85°C temperature, and altitudes up to 70,000 feet. Each board provides 350 V_{rms} Isolation between channels and also between the board and its enclosure or any other installed boards as well as electro-shock-discharge (ESD) isolation.

1.2.5 Software Support

The DNx-SL-504 is supported by the UEIDAQ Framework providing a simple and complete software interface to all popular Windows programming languages and data acquisition/control application packages, such as LabVIEW, MATLAB/Simulink, or any application that supports ActiveX or OPC servers. Support is also provided for all popular non-Windows operating systems including Linux, VXworks, QNX, RTX, INtime and more.

1.3 Features

The features of the DNx-SL-504 are listed below:

- Four (4) independent serial communication ports
- Each port software-configurable as RS-232 or RS-422/423/485
- Completely independent bit rate settings for every port
- Compatible with RS-422 networks when used in RS-485 mode
- Full-duplex support for RS-422/485 with both SL-504-1 and SL-504-801 board versions
- Two-wire, half-duplex support for RS-422/485 with the SL-504-801 board version
- Supports HDLC/SDLC synchronous communication protocols
- 350V isolation between ports, ports and circuitry; 15kV ESD
- Tested to withstand 5g vibration, 50g shock, -40 to +85°C temperatures, and altitudes up to 70,000 ft or 21,000 meters.
- Weight of 136 g or 4.79 oz for DNA-SL-504; 817 g or 28.8 oz with PPC5.
- UEI Framework Software API may be used with all popular Windows programming languages and most real time operating systems such as RT Linux, RTX, or QNX and graphical applications such as LabVIEW, MATLAB, and any application supporting ActiveX or OPC.

1.4 Indicators

The DNx-SL-504 indicators are described in **Table 1-1** and illustrated in **Figure 1-1**.

Table 1-1 SL-504 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) • **ON**: Operation mode

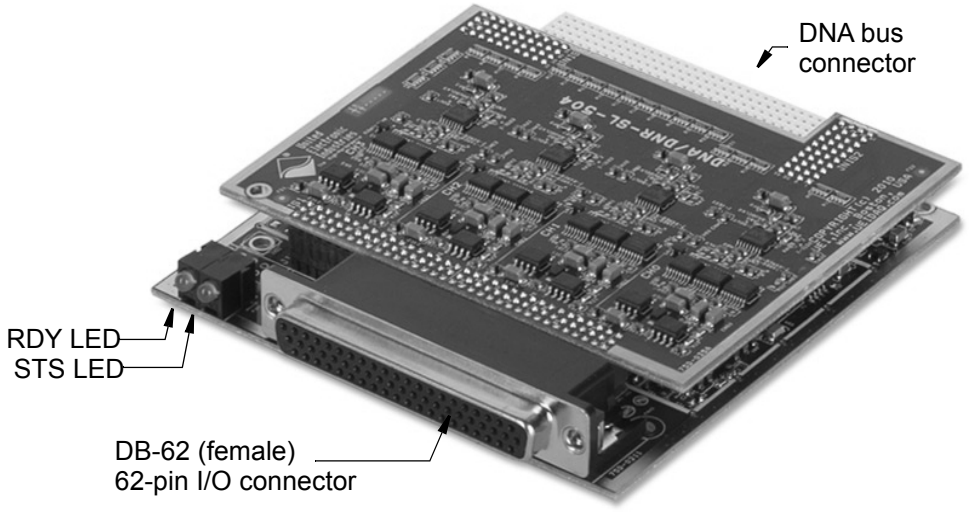


Figure 1-1 DNA-SL-504 Serial Board

1.5 Specification

The technical specification for the SL-504 is provided in the table below:

Table 1-2 . DNx-SL-504 Technical Specifications

Port Specifications	
Number ports	4, independently configurable
UART type	Zilog Z16C32
Interface types	RS-232, RS-422/423, RS-485
Protocols	HDLC, SDLC
FIFOs	32byte, input and output (per port)
Baud rate generator	Programmable, 1.2 kbaud to 4 Mbaud
RS-232 specifications	
RS-232 Synchronous	230 kbaud
RS-232 Signals	Tx, TxCLK Out, Rx, RxCLK In, CTS, Sync, DCD
RS-485/422 specifications	
RS-485/422 Synchronous	4 Mbaud
RS-485/422 Signals	Tx+, Tx-, TxCLK+, TxCLK-, RX+, RX-, RxCLK+, RxCLK- CTS+. CTS-, DCD+, DCD-
General Specifications	
Isolation	350 V port to port;
ESD protection	15 kV
Power Consumption	2-5W (RS-485 mode with max current drive)
Operating Temperature	Tested -40 to +65 °C
Operating Humidity	0 - 95%, non-condensing
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	290,000 hours

1.6 Serial Communication

This serial communication section provides an overview for the SL-504 supported recommended standards and protocols and is intended as a summary for the software programming chapters.

Refer to **Chapter 2** for programming specifics using the high-level framework API, and **Chapter 3** for programming specifics using the low-level API.

1.6.1 Standards and Protocols

The Open Systems Interconnection (OSI) model is a 7-layer conceptual model for standardizing communication, among computing systems and/or over a network.

The following is a list of standards and protocols the SL-504 can implement (see **Figure 1-2** for an overview of the OSI model):

- **Layer 1** is the Physical layer defining the hardware connection between sender and receiver. This layer defines the standard for transporting *raw bits,* rather than *logical data*.
  The SL-504 can implement RS-232, RS-422, or RS-485 standards.
  See Section 1.6.2 for a description of this hardware implementation.
- **Layer 2** is the Data-Link Layer. The data-link layer is structured into frames of *logical data* that transfer between sender and receivers. Layer 2 may detect errors experienced on Layer 1 and correct them.
  The SL-504 implements bit-oriented synchronous protocols, HDLC or SDLC.
  See Section 1.6.3 for a description of the Data-Link layer implementation.

NOTE: To allow flexibility when implementing any HDLC or SDLC dialect, the SL-504 only handles time-critical frame reception and transmission implementation of the data-link protocol. Users can choose how to handle the control implementation of the data-link protocol in their user application.

- **Layers 3 to 7**, when necessary, are implemented by the user application.

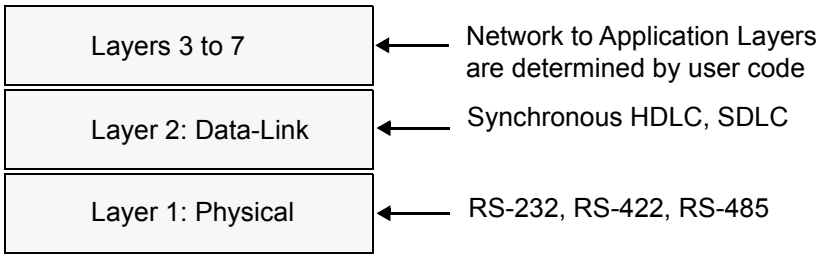


Figure 1-2 The OSI Model

1.6.2 The Physical Interface

Each of the four serial ports on the SL-504 boards can be configured independently to communicate using the RS-232, RS-422, and RS-485 standard. Port configuration is software programmable.

The physical layer offers the following services, provided by SL-504 hardware:

- Electrical interface
- Receiver/transmitter state in 4-wire (SL-504-1 or SL-504-801) bus topology or 2-wire (SL-504-801) topology.
- Line coding
- Bit-by-bit synchronized delivery

1.6.2.1 RS-232 Standard

RS-232 is a physical layer electrical specification that requires three wires (RxD, TxD, and common ground wires) for single-ended signaling. RS-232 uses bipolar voltages, (e.g, ±5V, ±10V), to provide a bidirectional, full-duplex, serial connection from 1 transmitter to 1 receiver (point-to-point, most commonly). The EIA/TIA RS-232-C (1969) standard recommends distances of less than 50 feet at signaling rates below 19200 baud.

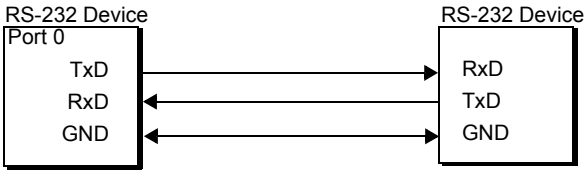


Figure 1-3 RS-232 Topology

1.6.2.2 RS-422 Standard

RS-422 is a physical layer electrical specification designating four wires (one twisted wire pair for Rx+/Rx- and one for Tx+/Tx-) for balanced differential signaling. RS-422 uses voltages not exceeding ±7V, to provide a unidirectional, full-duplex, serial connection from 1 transmitter to up to 10 receivers (multi-drop topology). The voltage difference between the two +/- wires represents the signal value, rather than the voltage level of just one wire. This eliminates a significant amount of noise in electrically noisy environments and permits higher data rates and cable lengths for RS-422 than RS-232.

1.6.2.3 RS-485 Standard

RS-485 is a physical layer electrical specification designating two wires (or 3, 4) for balanced differential signaling. RS-485 uses voltage differences of ±2.2V spanning over a common mode range of -7V to +21V, to provide a bidirectional serial connection between 32 transmitters and 32 receivers (multi-point).

See **Figure 1-4** for an example 4-wire RS-485 system using two twisted-pair connections (transmit/receive pairs) together on either the SL-504-1 or SL-504-801 board versions.

See **Figure 1-5** for an example 2-wire, half-duplex RS-485 system using one twisted wire pair. The SL-504 supports two-wire RS-485 systems with the SL-504-801 board version.

1.6.2.4 Applications and Network Topologies

RS-232 was defined as an interface between computers, devices, and terminals with modems, all of which are short links. Noise is a problem as baud rate and line length increase; longer distances are possible using low capacitance cable to reduce crosstalk, but the DTE and DCE share a common ground which can degrade between different power supplies. The signal can also become skewed or absorb noise from the external environment, becoming unreadable.

RS-422 master control units were designed to send commands in parallel to as many as ten slave receivers (yet the slave devices do not transmit by themselves in a multi-drop topology). In practice the RS-422 electrical interface is used in point-to-point topology with only two terminals and can be substituted for, or carry signals from, RS-232 links over long distances by using a converter.

RS-485 was designed to allow multi-point communication where 32 devices can both send and receive (not just receive, as with multi-drop). The user designs the access protocol, which usually involves one “master” device that coordinates one slave device (of 31) to transmit at a time.

Figure 1-4 shows the 4-wire topology for RS-422/485 operation. **Figure 1-5** shows the 2-wire topology for RS-422/485 operation for the SL-504-801.

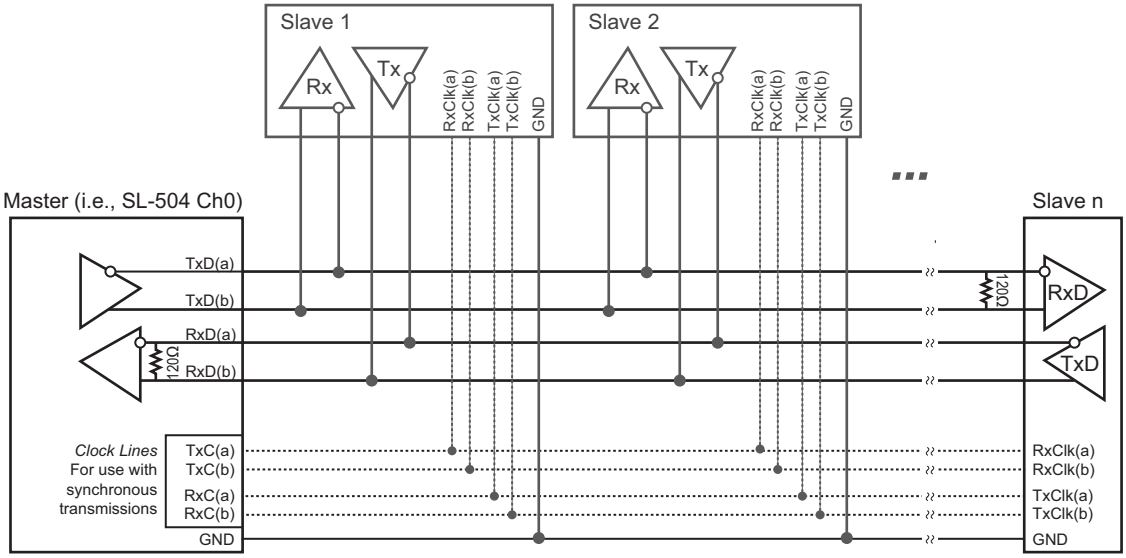


Figure 1-4 Four-wire Twisted-pair Full-duplex Network

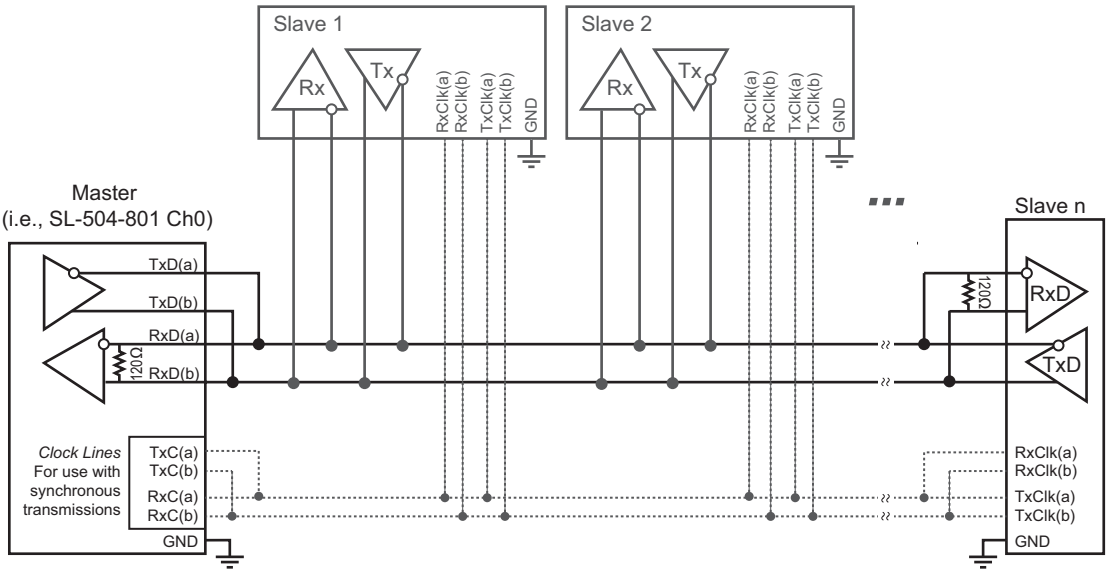


Figure 1-5 Two-wire Twisted-pair Half-duplex Network (SL-504-801 Board Version Only)

NOTE: 2-wire configuration is supported with the SL-504-801 board version only. SL-504-801 requires a configuration setting to use half-duplex mode, which tristates the TX driver when the port is receiving. Refer to Section 3.7 for 2-wire configuration programming details using the low-level API.

1.6.2.5 Data Frame Signaling

Figure 1-6 below shows UART data frame (top) representations for both single-ended ±5V signaling in RS-232 and twisted-pair RS-422/485.

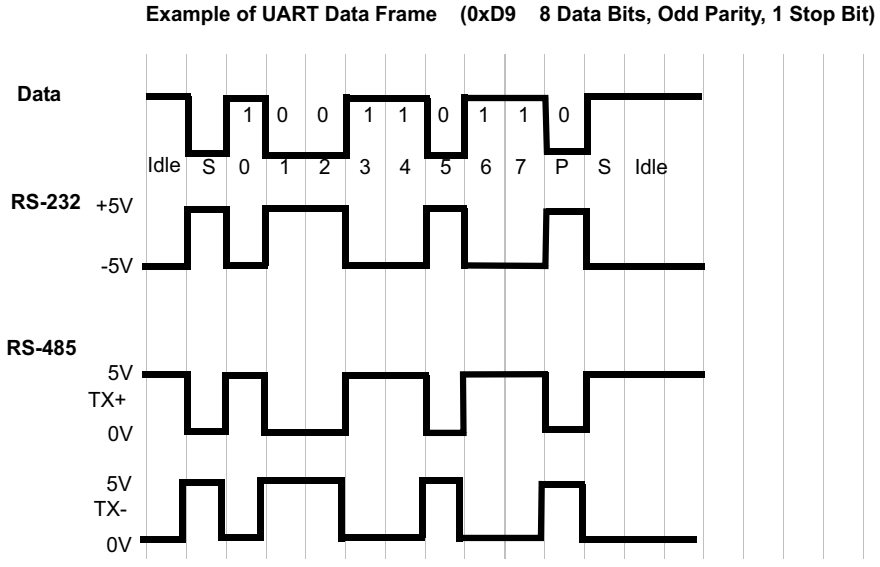


Figure 1-6 Diagram of UART Data Frames for RS-232 and RS-485

1.6.3 HDLC/SDLC Data-link Protocols

The SL-504 boards provide synchronous serial communication and support HDLC and SDLC bit-oriented data-link protocols.

Synchronous serial communication uses a separate clock signal to indicate that a new bit is ready on the data wire. Data characters of any number of bits can be grouped into frames (blocks of data) to be sent onto the serial hardware.

1.6.3.1 SDLC/HDLC Frame Overview

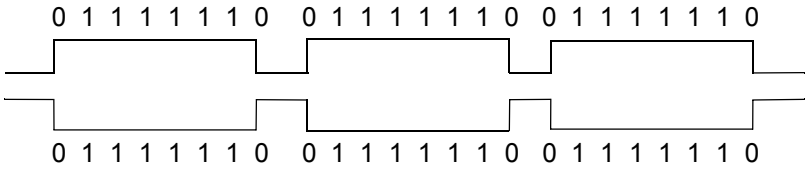
SDLC, and the extended HDLC, are bit-oriented data-link protocols. Various implementations of SDLC/HDLC conform to different dialects, but the information below is generally supported by most.

Both HDLC/SDLC organize bits into blocks of data (frames) that align to the following format:

Flag	Address	Control	Data	FCS	Flag
8 bits	8 or more bits	8 or 16 bits	Variable length 0 or more bits	16 or 32 bits	8 bits

- **Flag**: the start and end of the frame is marked with a flag, or bit sequence ‘01111110’ (0x7E hex), which acts as a frame delimiter. The flag is a unique sequence of bits guaranteed (using bit stuffing) not to be seen inside a frame.
- **Address**: the SDLC identifier of the secondary station(s) (individual, group, or broadcast). A primary is either a communication source or a destination, eliminating the need to include the primary address.
- **Control**: dependent on the type of frame:
  - Information Frame (user data)
    I-frames transport **user data** from the network layer, combined with flow and error control information. I-frames include a poll/final (P/F) bit, which is for performing flow and error control.
  - Supervisory Frame (control)
    An S-frame is used for flow and error **control** when no data is sent. They can request, re-request or suspend transmission, and acknowledge receipt of I-frames.
  - Unnumbered Frame (miscellaneous)
    U-frames are used for link management (e.g. initializing secondaries), status, test, and for transferring user data (in an information field).
- **Data**: logical data of variable-length (usually sent in multiples of 8 bits).
- **Frame Check Sequence (FCS)**: a 32- or 16-bit CCITT-CRC computed over the Address, Control, and Information fields intended to detect errors in transmission. The probability of an undetected error occurring increases with data length, so the FCS implicitly limits the practical size of the frame.

When no frames are being transmitted over a simplex or full-duplex sync link, a frame delimiter is continuously transmitted on the link. Using the standard NRZI encoding from bits to line levels, this generates one of two continuous waveforms, depending on the initial state, as shown below:



Flags are used by a receiver to synchronize its clock using a phase-locked loop. For half-duplex or multi-drop communication, a receiver will see continuous idling 1-bits in the inter-frame period when no transmitter is active.

Within any frame, a sequence of 7 or more 1-bits identifies an ‘Abort’ code.

1.7 Architecture

The architecture of the DNx-SL-504 is illustrated in the block diagram shown in **Figure 1-7**.

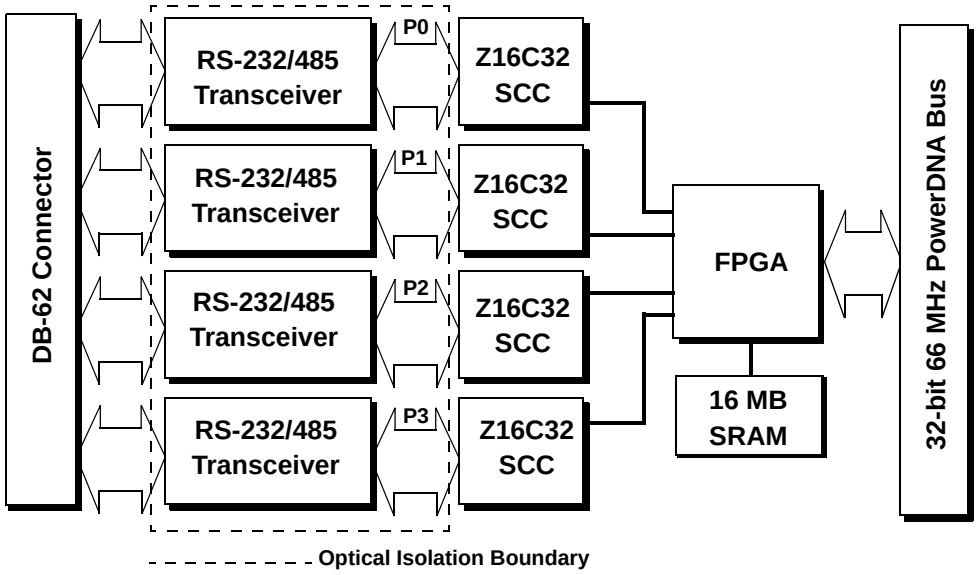


Figure 1-7 Logic Block Diagram: DNA/DNR-SL-504 Overview

The DNx-SL-504 comprises four serial ports designed for synchronized serial communication. Each port has a programmable physical signaling interface, baud rate, signal encoding, clocking scheme, and HDLC-specific format parameters including idle character, filtering, preamble size, preamble type, and user-configurable CRC.

1.7.1 RS-232/485 Transceiver

The DB-62 input/output traces are directly connected to four software-controlled EXAR SP506 single chip multi-mode serial line transceivers, one per channel, that select the type of DTE to DCE connectivity used in the actual physical signaling.

Each SP506 transceiver provides the following software enabled features, programmable for each SL-504 port:

- Internal loop-back, which eliminates the need for external loopback hardware. In loopback mode, driver outputs are internally connected to receiver inputs, creating an internal path for diagnostic testing.
- Physical interfaces (RS-232, RS-422, and RS-485 support).
- On-chip termination resistance, which can be enabled in RS-422 and RS-485 modes.

Signals pass between the SP506 transceiver and the Zilog Z16C32 serial communication control chip through optical isolation circuitry.

1.7.2 Universal Serial Controller

The Z16C32 Integrated Universal Serial Controller is a software-configurable multi-protocol data communications controller with on-chip dual-channel DMA. The serial controller offers many functions in an integrated design, rather than separate components, such as two baud rate generators (BSG0 and BSG1), a digital phase-locked loop, character counters, and 32-byte FIFOs for both receiver and transmitter.

The serial controller handles synchronous bit-oriented formats, such as HDLC, and supports virtually any serial data transfer application.

The serial controller can generate and check CRC in any synchronous mode. Direct access to the CRC value allows user software to resend or manipulate the CRC as needed. The Zilog document *UM014001-1002* provides additional information on the Z16C32.

The SP506 and Z16C32 are managed by a SL-504-specific logic module that works alongside the Core module logic found in all DNx products.

1.8 Wiring & Connectors (pinout)

Figure 1-8 shows the pinout of the 62-pin female D-Sub connector for the SL-504. The connector is divided into four 9-pin serial ports, as shown in the pinout.

Users can connect to the DB-62 connector either through a custom made cable or by connecting to a DNx-STP-62 accessory panel (see Appendix, page 31).

The following nomenclature is used for pin signals:

Name	RS-232	-/+ RS-422/485	Description
TxD	single-ended (a)	differential (a,b)	Transmit Data (output)
RxD	single-ended (a)	differential (a,b)	Receive Data (input)
CTS	single-ended (a)	differential (a,b)	Clear-to-send (input)
DCD	single-ended (a)	differential (a,b)	Data Carrier Detect (input)
TxC	single-ended (a)	differential (a,b)	Transmit Clock (output)
RxC	single-ended (a)	differential (a,b)	Receive Clock (input)
GND	single-ended	single-ended	Common Ground Reference/Port

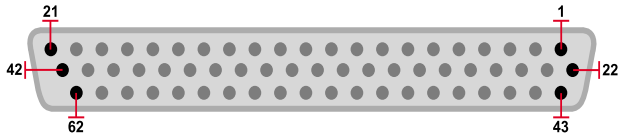
Table 1-3 . Abbreviations for pinout: (a) inverted, (b) non-inverted

For RS-232 electrical interfaces, each single-ended line is compared to the ground within the same cable that is shared between the two terminals.

For RS-422/485 two-wire twisted pairs form the a/b differential pairs:

- a is the inverting signal, sometimes labeled as -, for example TxD-
- b is the non-inverting signal, sometimes labeled as +, for example TxD+

The following pinout for the SL-504 is provided below:



Pin	signal	Pin	signal	Pin	signal
1	RESERVED01	22	GND-CH0	43	RxC(a)-0
2	TxC(b)-0	23	TxC(a)-0	44	RxC(b)-0
3	DCD(a)-0	24	DCD(b)-0	45	GND-CH0
4	RxD(b)-0	25	RxD(a)-0	46	RESERVED11
5	TxD(b)-0	26	TxD(a)-0	47	CTS(b)-0
6	GND-CH1	27	CTS(a)-0	48	RESERVED17
7	RxC(b)-1	28	RxC(a)-1	49	DCD(b)-1
8	TxC(b)-1	29	TxC(a)-1	50	DCD(a)-1
9	RxD(b)-1	30	RxD(a)-1	51	GND-CH1
10	TxD(b)-1	31	TxD(a)-1	52	CTS(a)-1
11	RESERVED33	32	CTS(b)-1	53	RxC(a)-2
12	TxC(b)-2	33	TxC(a)-2	54	RxC(b)-2
13	DCD(b)-2	34	GND-CH2	55	GND-CH2
14	RxD(b)-2	35	RxD(a)-2	56	DCD(a)-2
15	TxD(b)-2	36	TxD(a)-2	57	CTS(b)-2
16	GND-CH3	37	CTS(a)-2	58	RESERVED49
17	RxC(b)-3	38	RxC(a)-3	59	DCD(b)-3
18	TxC(b)-3	39	TxC(a)-3	60	RESERVED55
19	DCD(a)-3	40	GND-CH3	61	RxD(a)-3
20	TxD(b)-3	41	TxD(a)-3	62	RxD(b)-3
21	CTS(a)-3	42	CTS(b)-3		

Figure 1-8 DNx-SL-504 Connection Diagram

1.8.1 2-wire Wiring, Synchronous

Figure 1-9 provides a wiring example for 2-wire SL-504-801 synchronous communication.

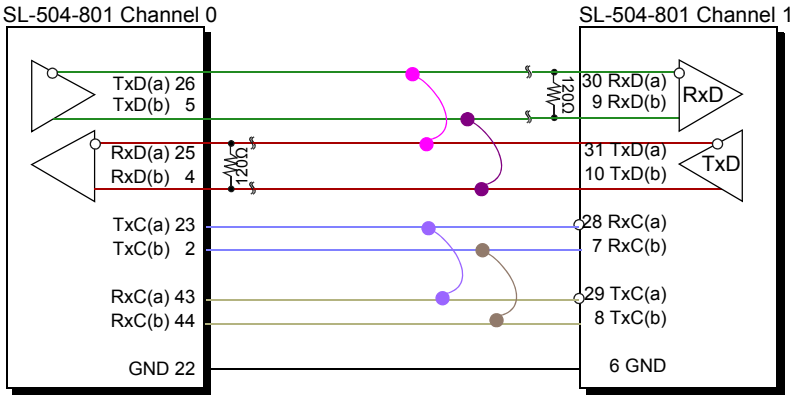


Figure 1-9 Two-wire Twisted-pair Half-duplex Sync Network (SL-504-801)

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-504:

- About the High-level Framework (Section 2.1)
- Creating a Session (Section 2.2)
- Configuring the Resource String (Section 2.3)
- Configuring the Timing (Section 2.4)
- Reading Data (Section 2.5)
- Writing Data (Section 2.6)
- Cleaning-up the Session (Section 2.7)

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 slSession;
```

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 HDLC ports 0,2,3 on device 1 at IP address 192.168.100.2: “pdna://192.168.100.2/Dev1/hdlc0,2,3”

The SL-504 is programmed using the subsystem **hdlc** to configure channels in HDLC mode.

2.3.1 Configuring the HDLC Port

Use the method **CreateHDLCPort()** to configure one or more channel(s) in synchronous mode.

The following call configures ports 2 and 3 of a SL-504 set as device 1:

```
// Configure session’s ports

hdlcSession.CreateHDLCPort(“pdna://192.168.100.2/Dev1/hdlc2,3”,
                           UeiHDLCPortRS232,
                           100000,
                           UeiHDLCPortEncodingNRZ,
                           UeiHDLCPortCRCNone,
                           UeiHDLCPortClockBRG,
                           UeiHDLCPortClockExternal);
```

It configures the following parameters:

- **Physical interface**: the physical interface used to transmit serial bytes.
  UeiHDLCPortRS232: RS-232
  UeiHDLCPortRS422: RS-422
  UeiHDLCPortRS485: RS-485
  UeiHDLCPortV35: V35
- **Bits per second**: the number of bits per second transmitted of the synchronous port
- **Encoding**: the method used to encode bits over synchronous serial line:
  UeiHDLCPortEncodingNRZ: NRZ encoding
  UeiHDLCPortEncodingNRZB: inverted NRZ encoding
  UeiHDLCPortEncodingNRZI: NRZI encoding
  UeiHDLCPortEncodingNRZIMark: NRZI encoding, invert state for 1
  UeiHDLCPortEncodingNRZISpace: NRZI encoding, invert state for 0
  UeiHDLCPortEncodingBiphaseMark: biphase encoding, with DPLL
  UeiHDLCPortEncodingBiphaseSpace: biphase encoding, with DPLL
  UeiHDLCPortEncodingBiphaseLevel: biphase encoding, with DPLL
  UeiHDLCPortEncodingBiphaseDiff: biphase encoding, used with DPLL
- **CRC**: the method used to calculate the cyclic redundancy code:
  UeiHDLCPortCRCNone: CRC is not checked neither for transmit nor receive
  UeiHDLCPortCRCUser: User responsible for inserting & checking CRC
  UeiHDLCPortCRC16CCITT: use 16-bit CCITT CRC (x^15+x^12+x^5+1)
  UeiHDLCPortCRC16: 16-bit polynomial
  UeiHDLCPortCRC32:-32-bit polynomial

- **TX clock source**: clock source used to synchronize transmitter:
  UeiHDLCPortClockExternalPin: Take clock from external RxC pin
  UeiHDLCPortClockBRG: Take clock from baud rate generator
  UeiHDLCPortClockDPLL: Take clock from DPLL divided by 32
  UeiHDLCPortClockDPLLDiv8: Take clock from DPLL divided by 8
  UeiHDLCPortClockDPLLDiv16: Take clock from DPLL divided by 16
- **RX clock source**: clock source used to synchronize receiver:
  UeiHDLCPortClockExternalPin: Take clock from RxC pin
  UeiHDLCPortClockBRG: Take clock from baud rate generator
  UeiHDLCPortClockDPLL: Take clock from DPLL divided by 32
  UeiHDLCPortClockDPLLDiv8: Take clock from DPLL divided by 8
  UeiHDLCPortClockDPLLDiv16: Take clock from DPLL divided by 16

In addition you can set the following parameter using the channel object methods (under LabVIEW use property node):

- **Termination**: Select whether to enable or disable termination resistors.
```
// enable termination resistor
pPort->EnableTerminationResistor(true);
```
- **Echo suppression**: Select whether to suppress echo in half duplex mode (RS-422)
```
// Disable echo
pPort->EnableHDEchoSuppression(true);
```
- **Loopback**: Select whether transmitter and receiver of the same port are tied.
```
// Disable loopback
pPort->EnableLoopback(false);
```
- **Abort Symbol**: The symbol used to abort
  UeiHDLCPortAbort7: Send 0x7F to abort
  UeiHDLCPortAbort15: Send 0x7FFF to abort
```
// Set abort symbol to 0x7F
pPort->SetAbortSymbol(UeiHDLCPortAbort7);
```
- **Underrun Action**: The action taken when underrun condition is detected:
  UeiHDLCPortUnderrunFinish: Close the frame by adding CRC to it
  UeiHDLCPortUnderrunFlags: Send flags
```
// Set underrun action
pPort->SetUnderrunAction(UeiHDLCPortUnderrunFinish);
```
- **Filter Mode**: The filter setting:
  UeiHDLCPortFilterNone: No filtering
  UeiHDLCPortFilterA16: +16 bits into RxFIFO if Addr matches or B/C as 2 bytes
  UeiHDLCPortFilterA24: +24 bits into RxFIFO if Addr matches or B/C as 3 bytes
  UeiHDLCPortFilterA32: +32 bits into RxFIFO if Addr matches or B/C as 4 bytes
  UeiHDLCPortFilterEALS: Places bytes while LS==0, then byte with LS==1 then 16 bits as 2 bytes into RxFIFO if EA matches or B/C
  UeiHDLCPortFilterEA24: Places 24 bits as 3 bytes into RxFIFO if EA matches or B/C
  UeiHDLCPortFilterEAMS: Places bytes while MS==0, then byte with MS==1 then 8 bits as 1 byte into RxFIFO if Ext Addr matches or B/C

UeiHDLCPortFilterEAMS16: Places bytes while MS==0, then byte with MS==1 then 16 bits as 2 bytes into RxFIFO if Ext Addr matches or B/C

```
// Disable filter
pPort->SetFilterMode(UeiHDLCPortFilterNone);
```

- **Filter address**: The address to filter

```
// Set address to filter
pPort->SetFilterAddress(0x52);
```

- **Idle flags**: The pattern to transmit when the link is idle.
  UeiHDLCPortIdleFlag: continuous flags
  UeiHDLCPortIdleZero: continuous zeroes
  UeiHDLCPortIdleOne: continuous ones
  UeiHDLCPortIdleMark: idle chars are marks
  UeiHDLCPortIdleSpace: idle chars are spaces
  UeiHDLCPortIdleMS: alternating Mark and Space
  UeiHDLCPortIdle01: .alternating 0 and 1

```
// Set idle pattern
pPort->SetidleCharacter(UeiHDLCPortIdleOne);
```

2.4 Configuring the Timing

The application must configure the SL-504 to use the “messaging” timing mode.

A message is represented by an array of bytes.

The SL-504 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
session.ConfigureTimingForMessagingIO(10, 1.0);
```

2.5 Reading Data

Reading data from the SL-504 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 serial port to be able to read from each port in the port list.

The following sample code shows how to create a reader object tied to port 1 and read up to 10 bytes from that HDLC port.

```
// Create a reader and link it to the session’s stream, port 1
reader = new CUeiHDLCReader(hdlcSession.GetDataStream(), 1);
// we’ll want to store for 10 bytes (char-sized)
uInt8 bytes[10];
// read up to 10 bytes
reader->Read(10, bytes, &numBytesRead);
```

2.6 Writing Data

Writing data to the SL-504 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 serial port to be able to write to each port in the port list.

The following sample code shows how to create a writer object tied to port 2 and send a frame of 128 bytes to the HDLC port.

```
// Create a writer and link it to the session's stream, port 2
writer = new CUeiSerialWriter(session.GetDataStream(), 2);
// store 128 bytes that we want to write out
unsigned char bytes[128];
memset(bytes,0x34,128);
// write 128 byte, numBytesWritten contains number of bytes actually sent
writer->Write(128, bytes, &numBytesWritten);
```

2.7 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
slSession.CleanUp();
```

Chapter 3 Programming with the Low-Level API

This chapter provides the following information about programming the SL-504 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)
- HDLC/SDLC Protocol Overview (Section 3.4)
- Enabling Ports (Section 3.5)
- Setting the Configuration (Section 3.6)
- Configuring 2-wire, Half-duplex Mode (SL-504-801) (Section 3.7)
- Configuring Tx and Rx Clocking (Section 3.8)
- Sending and Receiving HDLC Frames (Section 3.9)
- Reading the Link Status (Section 3.10)
- Aborting Transmission (Section 3.11)

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 Framework high-level API (see **Chapter 2**). The Framework extends the low-level API with additional functionality that makes programming easier, faster, and less error-prone.

For additional information regarding low-level programming, refer to the “PowerDNA API Reference Manual” located in:

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

3.2 Low-level Functions

Low-level functions are described in detail in the PowerDNA API Reference Manual. Table 3-1 provides a summary of SL-504-specific functions.

Table 3-1 Summary of Low-level API Functions for DNx-SL-504

Function	Description
`DqAdv504Enable`	Enables and disables requested channels on the SL-504
`DqAdv504GetStatus`	Requests status and accumulated statistics from the SL-504

Table 3-1 Summary of Low-level API Functions for DNx-SL-504 (Cont.)

Function	Description
DqAdv504SetConfig	Sets channel configuration parameters, such as protocol and physical interface use, baud rate, CRC mode, encoding type, preamble use, and more
DqAdv504SendFrame	Writes a frame of HDLC data of a specified channel to the on-board RAM (Tx). By default, sixteen 4096-byte frames are allocated for each channel.
DqAdv504SendMultFrames	Writes 1 to 16 frame(s) of HDLC data of a specified channel to the on-board RAM (Tx). By default, sixteen 4096-byte frames are allocated for each channel.
DqAdv504RecvFrame	Reads a frame of HDLC data of a specified channel from the on-board RAM (Rx). Along with frame data, this function also provides a frame-specific status block of data. By default, sixteen 4096-byte frames are allocated for each channel.
DqAdv504RecvMultFrames	Reads 1 to 16 frame(s) of HDLC data of a specified channel from the on-board RAM (Rx). Along with frame data, this function also provides a frame-specific status block of data. By default, sixteen 4096-byte frames are allocated for each channel.
DqAdv504AbortTx	Aborts transmissions and returns TX status.

3.3 Low-level Programming Techniques

Application developers are encouraged to explore the existing source code examples when first programming the SL-504. Sample code provided with the installation is self-documented and serves as a good starting point.

Code examples are located in the following directories:

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

3.4 HDLC/SDLC Protocol Overview

The HDLC protocol was developed in 1970s to facilitate synchronous transmission of data packets (or frames) over the RS-485 or RS-232 physical interface. Since this is a synchronous protocol, either clock & data lines or an encoding with an embedded clock can be used (effectively cutting bandwidth by half for the same baud rate).

When an HDLC transmitter is enabled, it constantly sends Idle characters on the bus. When a transmitter needs to start transmission, it sends a Flag sequence (0x7E). Six continuous “ones” on the bus signals to the receivers that this is a start of the frame, where a frame is a group of sequential characters ending with CRC for error-checking. While sending a frame, an HDLC transmitter continually checks whether any sequence of data bits could look like a Flag to the receiver. It does this without regard for character boundaries. Whenever the data presented to a transmitter includes a “zero” followed by five “ones”, the transmitter adds an extra “zero” bit after a fifth “one” bit. The receiver monitors the serial data stream as well and removes a trailing zero for any sequence that looks like 0111110, regardless of character boundaries.

Since the flag-matching hardware operates without regard for character boundaries, bit oriented synchronous protocols can handle any number of bits in length. The current implementation of the SL-504 card allows the transmission of multiple of 8-bit character and reception of any number of bits. It limits the maximum frame size of both receiver and transmitter to 4096 8-bit characters, not including CRC.

3.5 Enabling Ports

The `DqAdv504Enable()` function is used to enable and disable operations on a port (channel).

The function syntax is as follows:

```
DqAdv504Enable(int handle, int device, int chan_mask)
```

where `DqAdv504Enable()` parameters are defined as follows:

- `handle:` handle the IOM received when communications are opened with the function `DqOpenIOM()`
- `device`: board location within the chassis
- `chan_mask:` If the bit in the channel (port) mask (1<<port_number) is “one”, the port is enabled. If it is zero, the port is disabled. For example, 0xF enables all ports.

3.6 Setting the Configuration

The `DqAdv504SetConfig()`function is used to configure a channel:

```
DqAdv504SetConfig(int handle, int devn, int channel, SL504_SETCFG* config)
```

where the `SL504_SETCFG` structure elements are listed and described in **Table 3-2**:

Table 3-2 SL504_SETCFG Structure Elements for SL_504 Board Configuration

SL504_SETCFG Structure Element	Description	Supported #define Settings
protocol	Protocol setting	SL504_PROT_HDLC: HDLC and SDLC
modeflags	Additional flags for mode selections	SL504_HDLC_LOOP: <reserved> SL504_USE_CTS: CTS pin controls transmission SL504_USE_DCD: DCD pin enables receiver SL504_INHIBIT_TX: Do not enable transmitter SL504_INHIBIT_RX: Do not enable receiver SL504_MODE_NOTXONIDLE: (504-801 only) TX drivers disabled in idle, for 2-wire communication
physical	Physical interface setting	SL504_PHY_RS232: RS-232 up to 230k baud SL504_PHY_RS485: RS-485 up to 4Mbit SL504_PHY_RS422: RS-422 multidrop SL504_PHY_V35: balanced current data and clock and unbalanced voltage DCD and CTS SL504_PHY_TERM: enable termination in RS-485/ RS-422 modes SL504_PHY_LOOP : enable internal loopback in the selected mode SL504_PHY_NOECHO :suppress echo in RS-422/423 modes (disable RX while TX is transmitting)
hdlc_flags	Additional synchronous mode flags (HLDC)	SL504_HDLC_ABORT_7: send 0x7f as an abort symbol (default) SL504_HDLC_ABORT_15: send 0x7fff as an abort symbol SL504_HDLC_FINISH_UNDER: in underrun condition close the frame by adding CRC to it SL504_HDLC_FLAGS_UNDER: in underrun condition start sending flags SL504_HDLC_SHARED_ZEROES: send idle flags with shared zeroes

Table 3-2 SL504_SETCFG Structure Elements for SL_504 Board Configuration (Cont.)

SL504_SETCFG Structure Element	Description	Supported #define Settings
hdlc_encod	HDLC encoding type	SL504_HDLC_NRZ: NRZ encoding SL504_HDLC_NRZB: inverted NRZ encoding SL504_HDLC_NRZI: NRZI encoding SL504_HDLC_NRZI_MARK: NRZI encoding, invert state for 1 SL504_HDLC_NRZI_SPACE: NRZI encoding, invert state for 0 SL504_HDLC_BIPHASE_MARK: biphase encoding, with DPLL SL504_HDLC_BIPHASE_SPACE: biphase encoding, with DPLL SL504_HDLC_BIPHASE_LEVEL: biphase encoding, with DPLL SL504_HDLC_BIPHASE_DIFF: biphase encoding, used with DPLL
hdlc_baud	RxC or TxC clock rate in baud	N/A
hdlc_clk_src	Clock source for Tx/Rx synchronization	SL504_HDLC_FLAG_RXC_RXCPIN: RxClk from RxC pin (default) SL504_HDLC_FLAG_RXC_DPLL: RxClk from DPLL SL504_HDLC_FLAG_RXC_BRG: RxClk from BRG0 SL504_HDLC_FLAG_TXC_BRG: TxClk from BRG0 (default) SL504_HDLC_FLAG_TXC_DPLL: TxClk from DPLL SL504_HDLC_FLAG_TXC_RXCPIN: TxClk from RxC pin SL504_HDLC_FLAG_DPLL_DIV8: DPLL divider 8 SL504_HDLC_FLAG_DPLL_DIV16: DPLL divider 16
hdlc_crc_mode	CRC error-checking None, CRC-16/32/CCITT, user-supplied or automatic	SL504_HDLC_CRC_NONE: CRC is not checked, neither for TX or RX SL504_HDLC_CRC_USER: User is responsible for inserting and checking CRC SL504_HDLC_CRC_16_CCITT: 16-bit CCITT CRC SL504_HDLC_CRC_16: 16-bit polynomial CRC used SL504_HDLC_CRC_32: 32-bit Eth CRC used

Table 3-2 SL504_SETCFG Structure Elements for SL_504 Board Configuration (Cont.)

SL504_SETCFG Structure Element	Description	Supported #define Settings
hdlc_flt_mode	HDLC filter type, if filtering on RX packets is selected	SL504_HDLC_FLT_NONE: No filtering // see PowerDNA Reference Manual for descriptions of the following filtering options: SL504_HDLC_FLT_A_16 SL504_HDLC_FLT_A_24 SL504_HDLC_FLT_A_32 SL504_HDLC_FLT_EA_LS SL504_HDLC_FLT_EA_24 SL504_HDLC_FLT_EA_MS SL504_HDLC_FLT_EA_MS16
hdlc_filter	HDLC address filter value (8 bits), if filtering is enabled in hdlc_flt_mode	N/A
hdlc_preamble	HDLC preamble pattern	SL504_HDLC_PRMB_NONE: No preamble SL504_HDLC_PRMB_ZERO: All zeros SL504_HDLC_PRMB_ONE: All ones SL504_HDLC_PRMB_FLAG: All flags SL504_HDLC_PRMB_10: Alternating 1 and 0 SL504_HDLC_PRMB_01: Alternating 0 and 1
hdlc_prmbl_sz	Size of preamble, if preamble is selected	SL504_HDLC_PRMBSZ_16: 16-bit preamble used SL504_HDLC_PRMBSZ_32: 32-bit preamble used SL504_HDLC_PRMBSZ_64: 64-bit preamble used
hdlc_idle_ch	HDLC idle character representation (default 0x7E)	SL504_HDLC_IDLE_FLAG: Continuous flags (0x7E) SL504_HDLC_IDLE_ZERO: Continuous zeros SL504_HDLC_IDLE_ONE: Continuous ones SL504_HDLC_IDLE_MAR: Idle chars are marks SL504_HDLC_IDLE_SPACE: Idle chars are spaces SL504_HDLC_IDLE_MS: Alternating mark and space SL504_HDLC_IDLE_01: Alternating 0 and 1
async_baud, async_char_sz, async_start, async_stop, async_parity, async_msglen, async_tout are also elements found in the SL504_SETCFG structure. These elements are for Asynchronous communication, which is for UEI test/debug purposes and not currently supported in released versions of the software.		

Refer to the PowerDNA API Reference Manual for additional configuration descriptions for each parameter.

3.7 Configuring 2-wire, Half-duplex Mode (SL-504-801)

The SL-504-801 board version supports 2-wire, half-duplex communication. See Figure 1-5 on page 9 for topology.

To use 2-wire, half-duplex mode, the SL-504-801 board version must be used in conjunction with setting the `SL504_MODE_NOTXONIDLE` mode flag in software. The mode flag turns off TX drivers when the transmitter is in idle and the channel is in RS-422/485 configuration.

To enable 2-wire, half-duplex mode, the `SL504_MODE_NOTXONIDLE` #define flag is OR'ed with the `modeflags` parameter in the `pSL504_SETCFG` structure.

For example, if `config` is a structure defined as `pSL504_SETCFG config`, then flags would be or'ed as follows:

```
config.modeflags |= SL504_MODE_NOTXONIDLE;
```

The port configuration would then be updated with the `DqAdv504SetConfig()` function call:

```
DqAdv504SetConfig(hd, DEVN, ch_tx, &config);
```

Where `ch_tx` is the port (0,1, 2 or 3) on the SL-504-801 that is configured as 2-wire, half-duplex.

Refer to Section 3.6 for more information regarding `DqAdv504SetConfig()`.

For detailed information about all low-level API functions, refer to the PowerDNA API Reference Manual.

3.8 Configuring Tx and Rx Clocking

The `<hdlc_baud>` and `<hdlc_clk_src>` parameters set in the `DqAdv504SetConfig()` function are used to set up channel clocking. An description of the `DqAdv504SetConfig()`function is provided in Section 3.6.

The `<hdlc_baud>` parameter is used to select baud rate.

The `<hdlc_clk_src>` parameter defines the clock source for the SL-504 receiver and transmitter:

RX/TX	#define Constant	Clock Source
RX	`SL504_HDLC_FLAG_RXC_RXCPIN`	RxClk from external RxC pin (default)
RX	`SL504_HDLC_FLAG_RXC_DPLL`	RxClk from DPLL
RX	`SL504_HDLC_FLAG_RXC_BRG`	RxClk from bit rate generator on serial communication controller
TX	`SL504_HDLC_FLAG_TXC_BRG`	TxClk from bit rate generator on serial communication controller (default)
TX	`SL504_HDLC_FLAG_TXC_DPLL`	TxClk from DPLL
TX	`SL504_HDLC_FLAG_TXC_RXCPIN`	TxClk from external RxC pin

3.8.1 Clock Source Descriptions

The SL-504 provides three main sources for transmitter and receiver clocking.

- The **baud rate generator** (BRG) is a part of the Zilog Z16C32 chip. It can be programmed to generate a clock of a specific frequency derived from the clock synthesizer chip. There are three main base clock rates pre-selected for operations; firmware minimizes the error between the requested and actual clock by selecting the best combination of base clock rate and divider.
- The **RxC pin** is a clock input to the SL-504. The RxC pin is a standard clock source when setting a receiver encoding mode that doesn't use an embedded clock, (e.g., NRZ encoding). The RxC pin can be also a clock source for the transmit side.
- The **DPLL** clock is extracted from the data line transmission when encoding with embedded clock is used (Biphase encoding).

NOTE: When using DPLL as the clock source,#define constants, `SL504_HDLC_FLAG_DPLL_DIV8` and `SL504_HDLC_FLAG_DPLL_DIV16`, can be used to select a DPLL divider lower than the standard one of 32. DPLL is used in the clocking scheme when RxD provides data without RxC. In this case, BRG1 is programmed as a clock source and RxC is recovered from RxD and used as RxC and (optionally) TxC clock. DPLL mode in HCR is dependent on the encoding used.

3.9 Sending and Receiving HDLC Frames

The `DqAdv504SendFrame()` function is used to send a single frame and the `DqAdv504RecvFrame()` function is used to receive a single frame.

Both functions have similar parameters:

```
int DqAdv504SendFrame(int hd, int devn, int chnl, int flags, uint8 *data, int
rq_size, int *written, int *available)
int DqAdv504RecvFrame(int hd, int devn, int chnl, int flags, uint8 *data, int
rq_size, int *received, int *available, int *rsb)
```

- `hd`: handle to the IOM
- `devn`: position of board in the Cube or RACK
- `chnl`: channel to send or receive frame to/from
- `flags`: <reserved>
- `data`: data to send for TX or pointer to data to store received data for RX
- `rq_size`: number of bytes to write or size of the receive buffer
- `written` (or `received`): number of bytes written or received or error code
- `available`: number of frame entries left available
- `rsb`: (RX-only) frame-specfic block of RX status information

Each function writes or reads one frame at a time.

Internally, the Z16C32 is programmed to take advantage of DMA operations between its bus and the PSRAM chip on the board. By default, sixteen 4096 byte frames are allocated for each channel, for transmit and receive separately. The maximum number of frames is 256.

On transmission, a new frame is added to the list of filled frames and the number of empty frames is returned. The firmware stops accepting new frames when ¾ of the frames are used. At this point `DqAdv504SendFrame()` returns zero in the `<written>` field.

Note that when the `SL504_PHY_NOECHO` configuration flag is used, the number of Tx frames is limited to one. With this flag, firmware disables receiver from the moment the frame is sent to the moment DMA informs firmware via interrupt that the transmission is completed.

On reception, the receiver stops when all frames are filled.

3.10 Reading the Link Status

The `DqAdv504GetStatus()` function is used to retrieve accumulated link statistics:

int `DqAdv504GetStatus(handle, device, int chan_mask, pSL504_INT_STAT status)`

- `handle:` handle the IOM received when communications are opened with the function `DqOpenIOM()`
- `device`: board location within the chassis
- `chan_mask:` If the bit in the channel (port) mask (1<<port_number) is “one”, the port is enabled. If it is zero, the port is disabled. For example, 0xF enables all ports.
- `pSL504_INT_STAT stat:` pointer to the structure storing status/statistical information.

The `pSL504_INT_STAT` structure is defined as follows:

```
// Interface status
typedef struct {
    int cts;        // number of CTS transitions
    int dcd;        // number of DCD transitions
    int tx;         // bytes transmitted
    int ftx;        // frames transmitted
    int rx;         // bytes received
    int frx;        // frames received
    int frm_err;    // frame errors
    int ovr_err;    // overrun errors (Rx)
    int und_err;    // underrun error (Tx)
    int prt_err;    // parity errors
    int tx_abort;   // Tx frame aborts
    int rx_abort;   // Rx frame aborts
    int short_err;  // Rx frames too short to be valid
    int long_err;   // Rx frames too long (>4096) to be valid
    int lines;      // current line state
    int err_stat;   // most recent error status
    int brk;        // number of breaks
    int exithunt;   // ditto exited hunt mode
    int rxidle;     // ditto idle
```

```
    // there are a few important registers to return for debug purposes
    uint16 tdmr;    // Transmit DMA status register (see Table 3-3)
    uint16 rdmr;    // Receive DMA status register (see Table 3-4)
    uint16 ccsr;    // Channel command/status register (see Table 3-5)
    uint16 tcsr;    // Transmit command/status register (see Table 3-6)
    uint16 rcsr;    // Receive command/status register

} SL504_INT_STAT, *pSL504_INT_STAT;
```

Most fields are either collected when Z16C32 receives or sends frames or read directly from the chip registers.

The following status bits are useful in the *Transmit DMA Register*:

Bit	Name	Description of TDMR bit
7	Cont	Firmware has issued a Start/Continue command after loading next buffer address
6	GLink	The channel DMA is reading next address in the linked list
5	BUSY	The channel is operating, DMA waits to send/ continues to send data to the transmitter
4	INITG	The channel DMA is fetching information from the linked list or stopped while doing so
3	EOL	The channel DMA has reached the end of the linked list, there is no more data to transfer
2	EOB	The channel DMA has finished sending current frame data
1	HAbort	The channel stopped due to the firmware issued Abort
0	SAbort	The channel stopped due to the Abort command

Table 3-3 Transmit DMA Register

The following status bits are useful in *Receive DMA Register*:

Bit	Name	Description of RDMR bit
7	Cont	Firmware has issued a Start/Continue command after loading next buffer address
6	GLink	The channel DMA is reading next address in the linked list
5	BUSY	The channel is operating, DMA waits to receive/ continues to receive data from the receiver

Table 3-4 Receive DMA Register

Bit	Name	Description of RDMR bit
4	INITG	The channel DMA is fetching information from the linked list or stopped while doing so
3	EOL	The channel DMA has reached the end of the linked list, there are no more buffers to store data
2	EOB	The channel DMA has finished receiving current frame data or frame is too long and the end of buffer is reached
1	HAbort	The channel stopped due to the firmware issued Abort
0	SAbort	The channel stopped due to the Abort command

Table 3-4 Receive DMA Register (Cont.)

The following status bits are useful in *Channel Command/Status Register*:

Bit	Name	Description of CCSR bit
15	RCCF Overflow	RCC FIFO Overflow (should not occur if DMA is working properly and frames are read on time)
14	RCCF Avail	RCC FIFO has data (DMA takes care of emptying the FIFO)
12	DPLL Sync	DPLL is in sync with the input clock embedded in RxD
11	DPLL 2Miss	DPLL has seen two consecutive missing clocks
10	DPLL 1Miss	DPLL has seen a missing clock

Table 3-5 Channel Command/Status Register

The following status bits are useful in the *Transmit Command/Status Register*:

Bit	Name	Description of TCSR bit
7	PreSent	Transmitter has finished sending preamble
6	IdleSent	Transmitter has sent idle condition
5	AbortSent	Transmitter has sent Abort
4	EOF	Transmitter has sent end-of-frame
3	CRCSent	Transmitter has sent CRC code
2	AllSent	Last frame bit has gone out of transmitter
1	TxUnder	Transmitter has encountered underrun condition (starts sending idle character by default)
0	TxEmpty	TxFIFO is empty

Table 3-6 Transmit Command/Status Register

3.11 Aborting Transmission

The `DqAdv504AbortTx()` function is used to abort an HDLC frame transmission:

```
int DqAdv504AbortTx(int hd, int devn, int channel, uint32* status)
```

- `hd:` handle the IOM received when communications are opened with the function `DqOpenIOM()`
- `devn`: board location within the chassis
- `channel: Tx` channel number aborting
- `status:` transmit status/current value of TCSR register (`see` **`Table 3-6`**)

This function issues abort commands to the transmitter and clears all frames of the existing data. The status returned is the content of Z16C32 TCSR register.

Appendix A

A.1 Accessories

The following cables and STP boards are available for the SL-504 board.

DNA-CBL-62

This is a 62-conductor round shielded cable with 62-pin male D-sub connectors on both ends. It is made with round, heavy-shielded cable; 2.5 ft (75 cm) long, weight of 9.49 ounces or 269 grams; up to 10ft (305cm) and 20ft (610cm).

DNA-STP-62

The STP-62 is a Screw Terminal Panel with three 20-position terminal blocks (JT1, JT2, and JT3) plus one 3-position terminal block (J2). The dimensions of the STP-62 board are 4w x 3.8d x1.2h inch or 10.2 x 9.7 x 3 cm (with standoffs). The weight of the STP-62 board is 3.89 ounces or 110 grams.

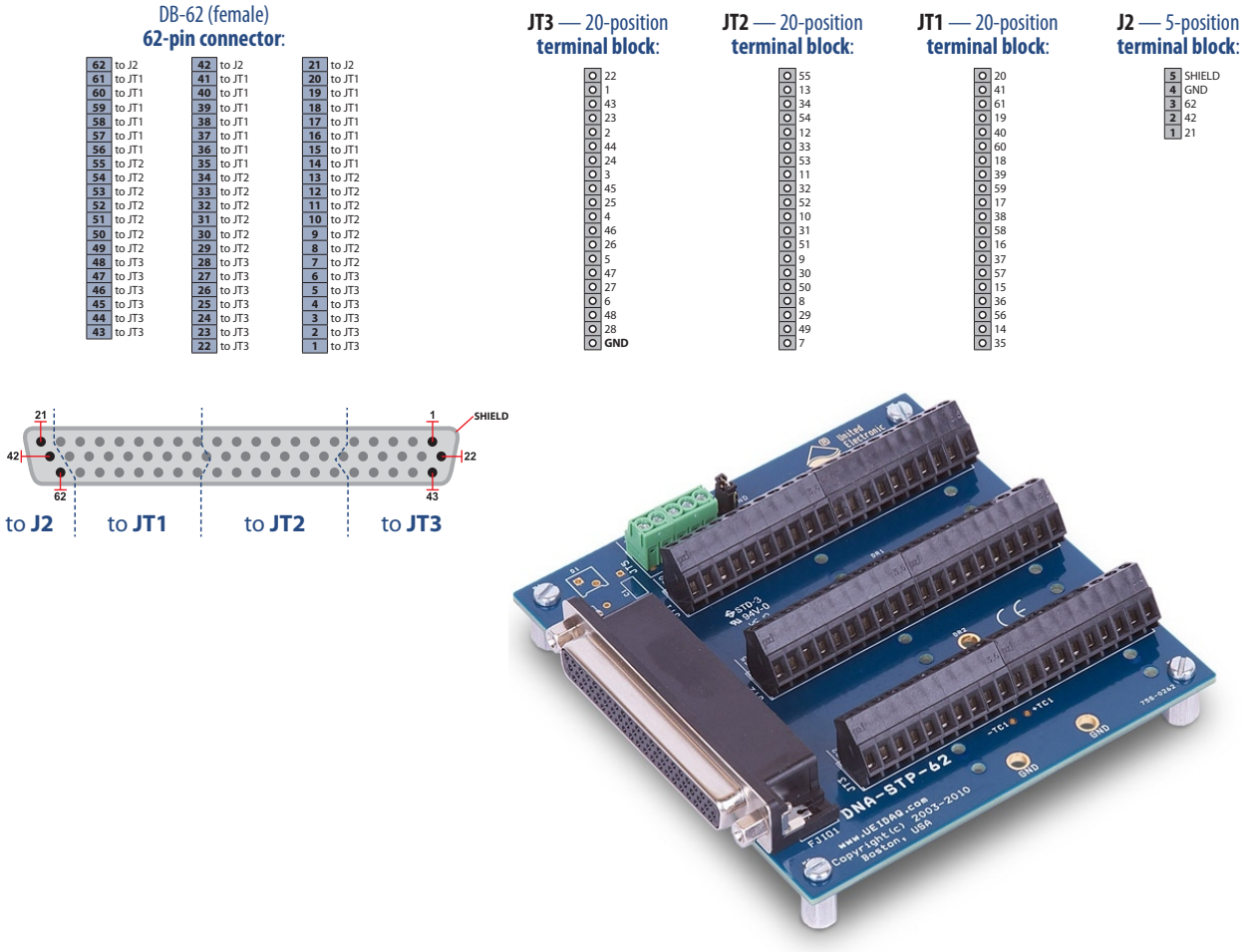


Figure A-1 Pinout and Photo of DNA-STP-62 Screw Terminal Panel

Index