United Electronic Industries

The High-Performance Alternative

DNx 429-512/566

—

User Manual

12-Channel ARINC 429 Transmitter/Receiver Interface Layer for the PowerDNA Cube and PowerDNR RACKtangle

Release 4.5

December 2013

PN Man-DNx-429-512/566-1213

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 see
http://www.ueidaq.com/partners/

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

Chapter 1 Introduction 1

1.1 Organization of Manual 1

1.2 The 429-512/566 Interface Board 3

1.3 Features 4

1.4 Indicators 4

1.5 Specification 5

1.6 Device Architecture 6
1.6.1 Word Format 8
1.6.2 Receiver Block 10
1.7.3 Label Acceptance Filter 12
1.8.4 Transmitter Block 13
1.8.5 Scheduler 14

1.9 Wiring & Connectors 17

Chapter 2 Programming with the High Level API 18

2.1 Creating a Session 18

2.2 Configuring the Resource String 18

2.3 Configuring the Timing 19

2.4 Read Data 20

2.5 Write Data 20

2.6 Programming the Output Scheduler 21

2.7 Program the Label Filter 22

2.8 Cleaning-up the Session 22

Chapter 3 Programming with the Low-level API 23

List of Figures

1-1 Typical Schematic Diagram for FET Digital Output (DOUT0) 3
1-2 The DNA-429-512/566 ARINC-429 Layer 4
1-3 DNA/DNR-429-512/566 Logic Block Diagram 6
1-4 ARINC 429 Waveform Characteristics 7
1-5 General ARINC Word Format 8
1-6 Receiver Diagram 11
1-7 Transmitter Block Diagram (Hardware Abstraction Layer) 13
1-8 DNx-429-512/566 Pinout Diagram 17

Chapter 1 Introduction

This document outlines the feature-set of the DNR- and DNA-429-512/566 layer and its use for synchronous serial-line communications applications.

1.1 Organization of Manual

This 429-512/566 User Manual is organized as follows:

- **Introduction**
  This section provides an overview of the 429-512/566 ARINC interface board 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**
  Describes low-level API commands for configuring and using the 429-512/566 series layer for serial operating modes.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for use with the DNx-429-512/566 serial-line communication interface 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**.”

Text formatted in `fixed` typeface generally represents source code or other text that should be entered verbadim 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 The 429-512/566 Interface Board

The DNA- and DNR-429-512/566 layer is a 12-channel ARINC transmit/receive interface for the Cube/RACKtangle I/O chassis respectively, designed for serial communication in avionics applications using the ARINC 429 protocol. It has 12 ARINC 429 channels that can be specified in either of two configurations: (1) as 6 TX and 6 RX channels (DNx-429-566), or (2) 12 RX channels (DNx-429-512).

The boards comply with the ARINC 429 specification and can run at either high speed (100 kHz) or low speed (12.5 kHz). The speed is software-selectable on a per port pair basis (see page 10). To ensure data integrity, 256-word FIFOs are provided on every TX and RX channel.

In addition, the DNx-429-566/512 ARINC layer has 3x current sinking (350 mA max), low-side FET, general purpose digital output with 500 mA resettable fuse.

Software supplied with the boards permits you to select options such as:

- Receive Filter size (1 to 255) or disable
- Source/Destination filter (SDI) enable/disable
- “Forward changed data only” filter enable/disable
- Parity check enable/disable
- Date/Time Stamping enable/disable by label or globally
- Transmit mode – Scheduled or Asynchronous
- Scheduling Table size – Up to 256 labels per channel
- Asynchronous TX Modes –
  High priority – transmit immediately regardless of schedule
  Standard priority – transmit when no scheduled data is present
  FIFO based – transmit when no other data is being sent

With this layer, you can realize the benefits of using a UEI Cube in avionics applications in any of its forms — PowerDNA, UEILogger, or UEIPAC — with communications handled by the ARINC 429 protocol.

1.3 Features

The common features of the DNx-429-512/566 are listed below:

- 12 ARINC 429 channels configured as 6TX/6RX channels (DNA-429-566) or 12 RX channels (DNA-429-512)
- High Current (350 mA max), Low-Side FET General Purpose Digital Output with 500 mA resettable fuse
- High Speed (100 kHz) or Low Speed (12.5 kHz), selectable per port pair
- Hardware Label filtering
- Hardware Transmit Scheduler (100 uS timing resolution)
- Automatic timestamping of data, software enabled/disabled
- Tested to withstand 5g Vibration, 50g Shock, -40 to +85°C Temperature, and Altitude up to 70,000 ft or 21'000 meters.
- Weight of 104 g or 3.7 oz for DNA-429-512/566; 4.1 oz for DNR-429.
- 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, DASYLab and any application supporting ActiveX or OPC.

1.4 Indicators

A photo of the 429-512/566 unit is illustrated below.

The front panel has two LED indicators:

- RDY: indicates that the layer is receiving power and operational.
- STS: can be set by the user using the low-level framework.

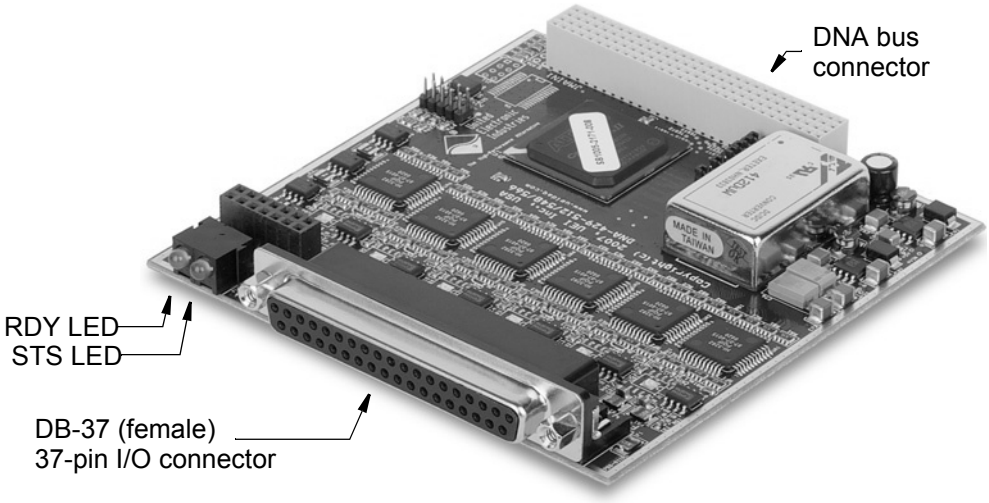


Figure 1-1. The DNA-429-512/566 ARINC-429 Layer

1.5 Specification

The technical specification for the 429-512/566 is provided in the table below:

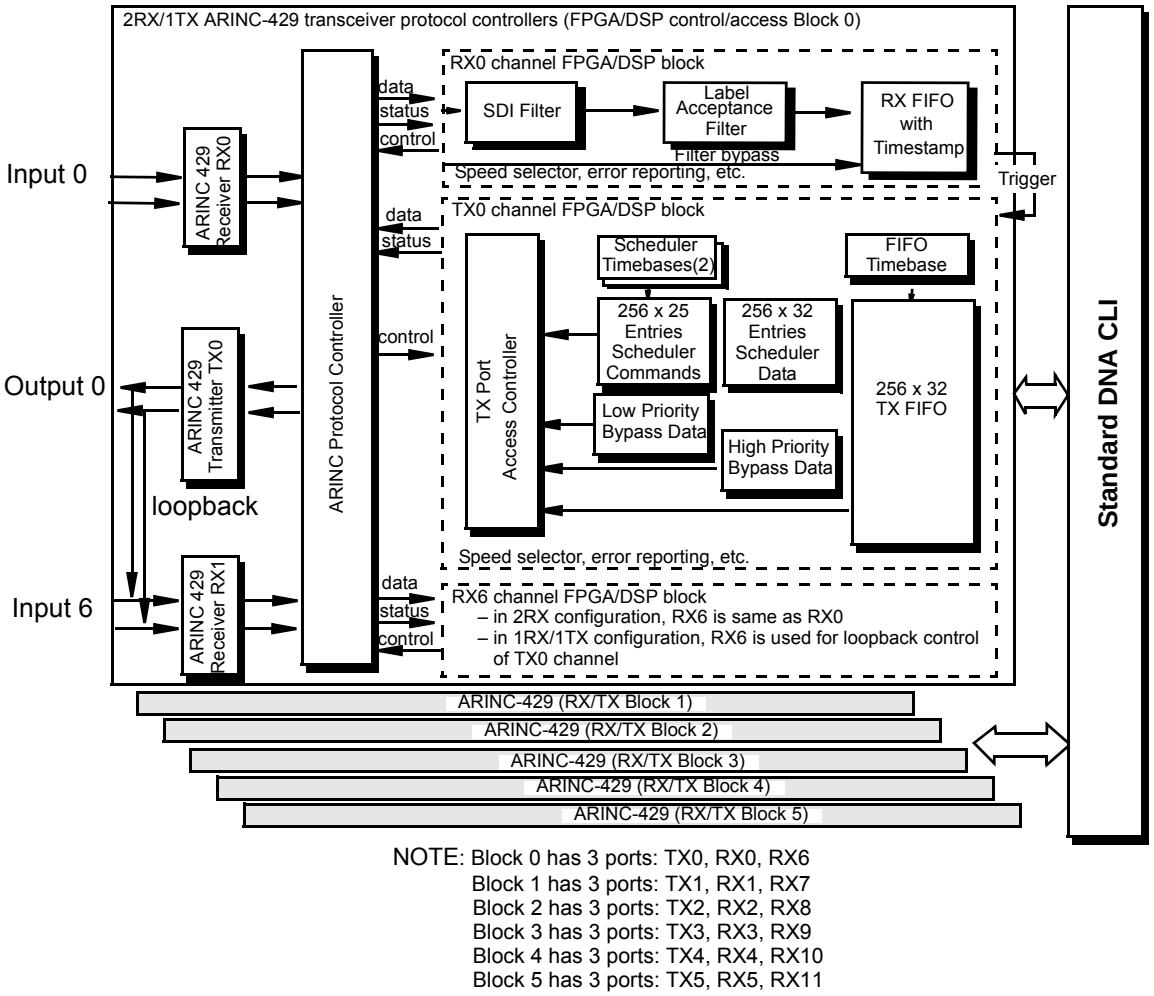
Table 1-1. DNx-429-512/566 Technical Specifications

Channel Configurations	
Number of channels	12
DNA-429-566	6 TX and 6 RX
DNA-429-512	0 TX and 12 RX
ARINC Compliance	Fully compliant with ARINC 429
Digital outputs	1 current sinking, FET based
Digital output drive	350 mA max, (500 mA resettable fuse)
Receive Specifications	
Data rate	100 kHz or 12.5 kHz*
FIFO size	up to 256 32-bit words, user selectable
Receive filter size	1 to 255 Labels or disabled
SDI filter	enabled or disabled
New data only filter	enabled or disabled by label or globally
Parity checking	enabled or disabled
Date/Time stamping	enabled or disabled by label or globally
Transmit Specifications	
Data rate	100 kHz or 12.5 kHz*
FIFO size	256 words
Transmit modes	Scheduled or asynchronous
Scheduler specifications	
timing resolution	100 microseconds
table size	Schedule up to 256 labels per channel
Asynchronous TX modes	
High priority	transmit immediately upon completion of current transmission regardless of schedule
Standard priority	transmit when no scheduled data
FIFO based	transmit when no scheduled, standard or high priority data is being sent
General Specifications	
Loop back testing	Internal loop back connections on the DNx-429-566 allow automatic self-test
Operating temperature	tested -40 °C to +85 °C
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
Humidity	0 to 95%, non-condensing
MTBF	470,000
Power consumption	3.5 Watt, maximum

* For 429-512 communication configuration (i.e. speed) is shared by every two two ports.
For 429-566 communication configuration is individually selectable per port/channel.

1.6 Device Architecture

The DNx-429-512/566 Layer has 6 parallel ARINC processors running at 66MHz. Sub-processor DSPs run inside each ARINC processor. A block diagram of the board is shown in **Figure 1-2**.



NOTE: Block 0 has 3 ports: TX0, RX0, RX6
Block 1 has 3 ports: TX1, RX1, RX7
Block 2 has 3 ports: TX2, RX2, RX8
Block 3 has 3 ports: TX3, RX3, RX9
Block 4 has 3 ports: TX4, RX4, RX10
Block 5 has 3 ports: TX5, RX5, RX11

Figure 1-2. DNA/DNR-429-512/566 Logic Block Diagram

As shown in the diagram above, each RX/TX block contains two ARINC Receivers and one ARINC Transmitter. The ARINC Receiver/Transmitter and communication protocol functions are handled by Holt **(www.holtic.com)** HI-3282 serial transmitter/dual receiver chips (for both the 429-512 and -566) and associated HI-8585 line drivers (for the 429-566 output only).

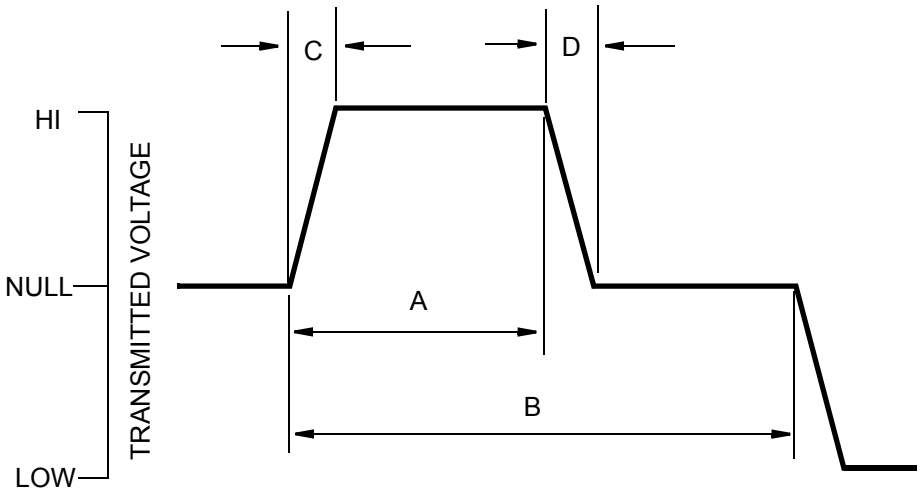
As stated in the ARINC specification, the following detection levels are required for the three signal states:

State	Differential Voltage
ONE	+6.5 V to +13 V
NULL	+2.5 V to -2.5 V
ZERO	-6.5 V to -13 V

If the received signal for any state is outside the specified range, the chip rejects the data.

The board can operate at either of two speeds, 100kHz or 12.5kHz, which is software selectable on a per channel or port pair basis (see page 10). The transmission medium for an ARINC 429 bus is 78-ohm, twisted, shielded-pair cable, grounded at both ends and at any break in the cable shield. Each bus has only one transmitter and up to 20 receivers. Since data transmission is uni-directional only, transmitters and receivers are on separate ports.

The waveform characteristics must conform to the specifications illustrated in **Figure 1-3**.



		Hi Speed	Low Speed
A	First half of pulse	5usec ± 5%	B/2 ± 5%
B	Full pulse cycle	10 usec ±2.5%	1/bit rate ± 5%
C	Pulse Rise Time	1.5 ±0.5 usec	10 ± 5 usec
D	Pulse Fall Time	1.5 ±0.5 usec	10 ± 5 usec

Received Voltage	
HI	6.5V to 13 V
NULL	-2.5V to +2.5V
LOW	-6.5V to -13V

Figure 1-3. ARINC 429 Waveform Characteristics

1.6.1 Word Format

Each ARINC-429 BNR word is a 32-bit value containing 5 fields, as shown in **Figure 1-4**.

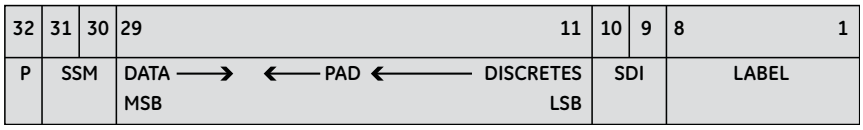


Figure 1-4. General ARINC Word Format

The bits have the following meanings:

- Bit 32 — parity bit, used to determine validity of word
- Bits 31, 30 — Sign/Status Matrix (SSM), indicates status of data

  OP [1 1] indicates data is normal
  TEST [1 0] indicates data comes from a test source
  FAIL [0 0] indicates a hardware failure causing data to be missing
  NCD [0 1] indicates data missing because of non-hardware reason
- Bits 11 to 29 — contain the data (bit-field, BCD, BNR, or mixed)
- Bits 9 and 10 — Source/Destination Identifiers (SDI) indicate the intended receiver or the transmitter sending the data
- Bits 1 to 8 — a label, in octal, identifying the data type

NOTE: Use of SDI and SSM is not mandatory.

Although the order in which bits are transmitted/received is somewhat complex, the HI-3282 chips handle the decoding automatically. The bits are transmitted on the ARINC bus in the following order.

8,7,6,5,4,3,2,1, 9,10,11,12, . . . 32

The Label (first 8 bits) is transmitted before the data (MSB first). After the label is sent, the LSB of each byte is transmitted first.

1.6.1.1 Scheduler ARINC Data Format

The Scheduler ARINC Data Format is as shown in the table below.

Bit	Name	Description	Reset State
31-0	**ADATA**	ARINC data	0

ARINC data is transferred via the ARINC-429 bus whenever a corresponding scheduler command entry is set for execution.

ARINC data should be preformatted in a form that is common for industry protocol controllers such as the Holt HI-3282.

Note that the ARINC serial communication standard numbers bits from 32 (MSB) to 1 (LSB). The UEI DNx-429-566/512 numbers them from 31 to 0, and the Holt controllers use two 16-bit words with bits numbered from 15 to 0. Also note that only LABEL, PARITY, SSM, and SDI bits are strictly defined by the ARINC-429 standard. The rest of the bits may be used as a payload by various sub-protocols.

The table below illustrates the differences in bit numbering for various protocols.

Table 1-2. Differences in Bit Numbering for Various Protocols

Holt Bit No.		DNA Bit No.	ARINC Standard Bit No.	Name	Description
15	Word 2	31	29	**SIGN**	Sign Bit
14-0		30-16	28-14	**DATA**	15 main data bits
15-13	Word 1	15-13	13-11	**DATA**	3 additional data bits
12-11		12-11	10-9	**SDI**	Source/Destination Identifier, 2-bit field used as an additional filter, programmed by a direct write to the HI-3282 configuration word and a write to the **AR566_ARDA0** register.
10-9		10-9	31-30	**SSM**	Sign/Status matrix or data bits. See ARINC-429 protocol documentation for details.
8		8	32	**PARITY**	Parity bit – auto inserted by the transmitter and calculated by the receiver. Normal parity for the ARINC-429 is “odd”. (Parity bit should be set if the number of “1”s in the rest of the bits is even, and should be cleared if not.) DNA-429 layers support even, odd, or no parity modes (the ARINC standard is “odd” parity). In “no parity” mode, the PARITY bit is transmitted to the bus, but is ignored.
7		7	1	**LABEL**	Label — This bit is used to identify data types and associated parameters. DNA-429 layers have a very flexible acceptance filter with an option to put only “new” data into the FIFO or to trigger the Scheduler when the selected label is received. When SDI bits are enabled, the Label Filter functions as a 10-bit identifier.
6		6	2		
5		5	3		
4		4	4		
3		3	5		
2		2	6		
1		1	7		
0		0	8		

For the parity bit above, transmission and reception have different behaviors:

- Transmit: parity can be selected to be Odd, Even, or None. Selecting Odd/Even enables parity bit insertion into transmitter data bit 32. The parity generator counts the ONES in the 31-bit word. If Odd is selected, the 32nd bit transmitted will make parity odd. If Even is selected, the parity is even. The parity bit is automatically set and you can't "force" it when sending a label. If None is selected, data is inserted on bit 32 (taking whatever bit 32 you supply).
- Receive: parity bit is is calculated by the chip, the value from the bus is not used. The receiver parity circuit counts 1's received, including the parity bit, ARINC bit 32. If the result is odd, then "0" will appear in bit 32. When receiving, the received parity bit is matched against the expected value. If the received data is correct and parity is even, then the parity bit should be 1 in the received word; if it is 0 it is a parity error. If the parity is odd it should be 0; and error is 1. If the parity is disabled it sets zero or one (depending on calculations) and you need to re-calculate parity for bits 1-31 to deduce the actual value of the transmitted parity bit.

1.6.2 Receiver Block

As indicated in **Figure 1-2** on page 6, the RX0 receiver in each Building Block has a software-selectable label acceptance filter that accepts or rejects incoming data words. As an option, you can choose to accept only "changed data". This option uses the "last received data" memory and places only "changed" data into the receive FIFO. Unchanged data is discarded. The RX FIFO can store up to 256 last received ARINC-429 32-bit frames, if timestamping is not selected. If timestamping is selected, the RX FIFO is limited to 128 frames and 128 timestamps. If the label filter is not selected, data is placed directly into the 256-word FIFO.

Each building block shares communication configuration settings. This means that the RX0, RX6, and TX0 of Building Block 1 share speed settings. Thus on the 429-512 the pair of receivers (RX0 and RX6) will use the same speed. The 429-566 RX0 and TX0 channel also uses the same speeds.

The receivers are electrically connected in different ways on the 429-512 and 566. On the 429-512 board each of the two receivers is individually addressable. If the board is a -566 version, however, (one receiver and one transmitter per channel), the second receiver is used as a loopback controller for the associated transmitter. To make the Loopback Connection between a TX1 and its associated loopback RX1 on the -566 version, you must set up a port on the channel.

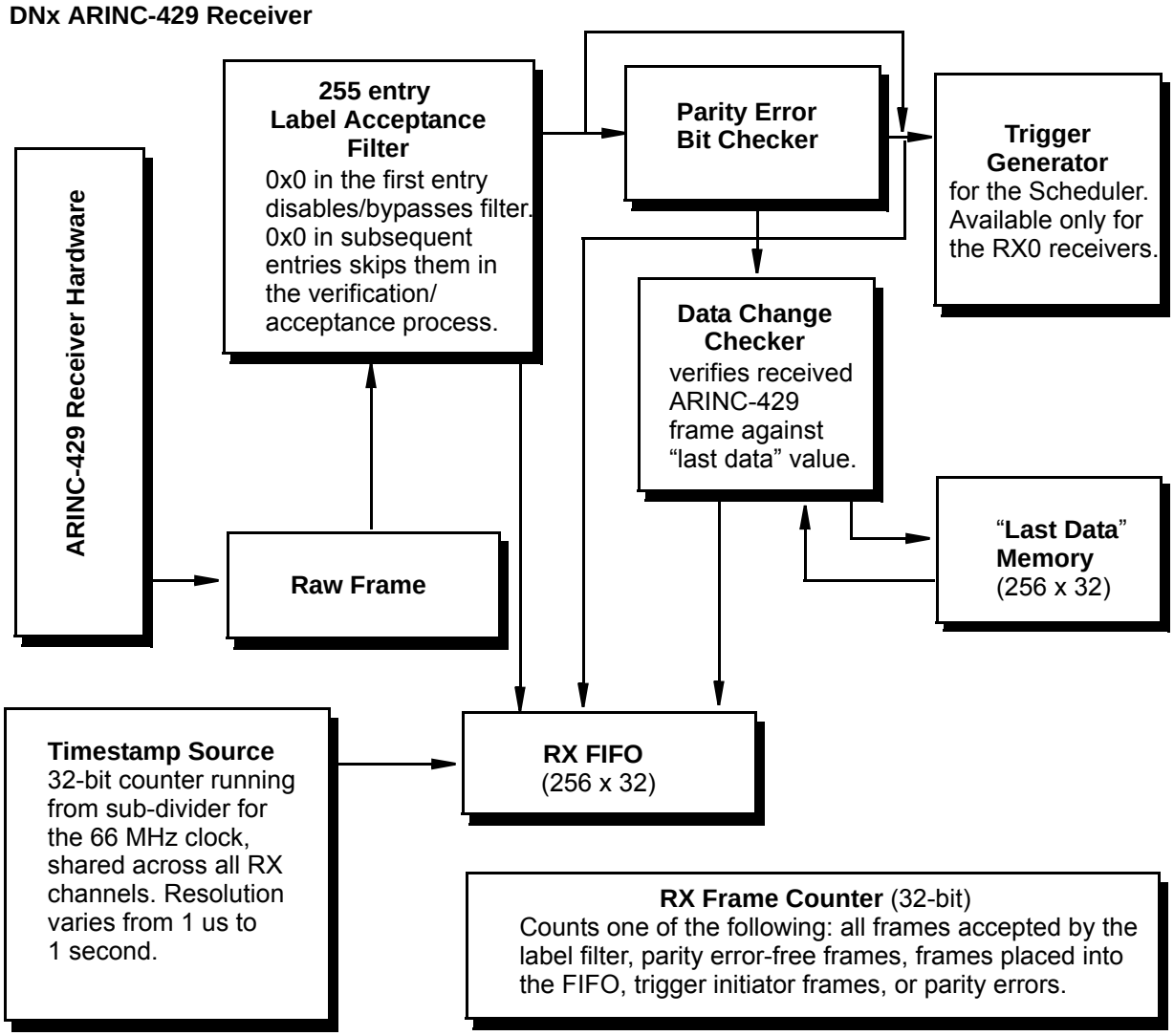


Figure 1-5. Receiver Diagram

Figure 1-5 is a block diagram of an ARINC receiver function. As illustrated, received data may be placed directly into the 256-word FIFO by writing 0x0 in the first label entry or may be passed through the label filter and the data change filter before being accepted. Similarly, the parity checker and last data checker can also be bypassed, if desired.

1.7.3 Label Acceptance Filter

The Label Acceptance Filter is a table with 255 10-bit entries. **Table 1-3** describes the format of each entry.

Table 1-3. Label Acceptance Filter

Bit	Name	Description	Reset State
31-9	**RSV**	Reserved/unused (should be written with 0 for compatibility with future enhancements)	0
9	**TE**	Trigger Enable. • For RX0 only. A “1” in this bit indicates that a Scheduler entry with the same index should be set for execution (scheduled for output on the ARINC-429 bus). Note that only the “master” entry in the Scheduler may be triggered this way. • For RX1, this bit is ignored.	
8	**ND**	“New Data Only” flag. In “new data only” mode (set via RX0 Control Register **AR566_Rx**, and if this bit is set, only data that has changed since last reception is placed into the FIFO and the “last data” memory. **NOTE:** For the 566 model, the “new data only” mode is not available on loopback receivers.	0
7	**LABEL**	Acceptance label. If the label matches bits 7-0 of the received frame, the frame is accepted by the filter. **NOTE:** 1. 0x0 in the label and flag fields of first entry (with index 0) in this table disables filtering, causing the board to accept all ARINC data. If the FIFO receiver is programmed to reject frames with parity errors, however, such frames are still rejected. 2. 0x0 in any label other than those with index 0 disables the entry in the label acceptance filter. When this occurs, the next entry is processed instead. 3. Labels may be programmed, enabled, or disabled at any time during operation. 4. For the 566 model, the Label Filter feature is not available when RX1 is used as a loopback receiver.	0

1.8.4 Transmitter Block

As shown in **Figure 1-2** on page 6, the TX0 transmitter in each Building Block contains a TX port access controller function, a Scheduler, a TX FIFO, and low- and high-priority bypass data memory registers. Data is passed from this block to the ARINC protocol controller and transmitter hardware for transmission to designated receivers. A block diagram of the Hardware Abstraction Layer (HAL) is shown in **Figure 1-6**.

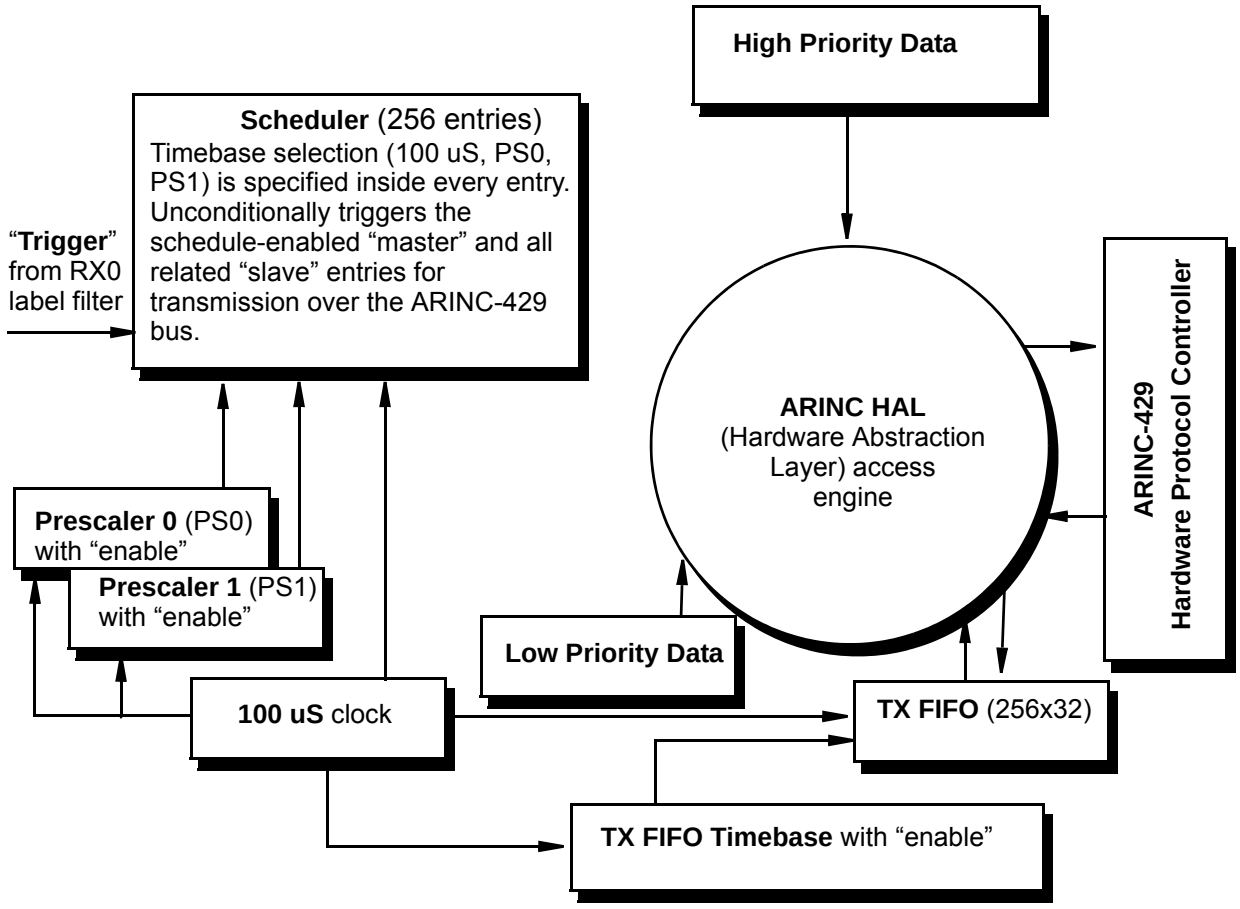


Figure 1-6. Transmitter Block Diagram (Hardware Abstraction Layer)

As illustrated in **Figure 1-6**, a transmitter block consists of a Scheduler, two independent 16-bit prescalers, a 256-word output FIFO, and an emergency transfer transmitter with a high- and a low-priority register.

The Scheduler transfers its data to the ARINC bus at a specified time interval either one time or continuously at user-specified intervals. Scheduled data may also be transferred in blocks based on a "master/slave" entry scheme. Also, the Scheduler may be configured to transfer data when a predefined label with index is received from the label filter.

The Prescalers are used to define the time delay for each master entry in the schedule and may be disabled, keeping entries in the pause state.

The 256-word output FIFO runs at a lower priority than the Scheduler and may output data in either of two modes: (1) whenever the interface IC can accept data and none is available from the Scheduler or (2) on a “paced” mode, based on user-defined clock intervals.

An Emergency Transfer Transmitter transfers data from the high-priority register immediately after the current TX operation is completed, and from the low-priority register only when the data from the Scheduler and high-priority register are not available.

Transmitter sources are assigned ARINC bus access according to the following priorities:

1. High-priority Data Register OR*1. FIFO
2. Scheduler Data2. Scheduler Data
3. Low-priority Data Register3. Low-priority Data Register
4. FIFO4. High-priority Data Register

The HAL accepts data based on the priority scheme above and sends confirmation after the data is accepted for transmission

NOTE: *The Scheduler and FIFO priorities may be interchanged by setting the FIFOHP bit in the **AR566_ACCR** register.

1.8.5 Scheduler

The Scheduler is programmed using two arrays in the memory address space dedicated to the Scheduler command and associated ARINC data.Because of space limitations, the arrays all share the same memory locations and may only be accessed one block at a time using the **A566_TXFSR9** register as a selector. The TX Scheduler ARINC data area occupies 256 32-bit locations and the Scheduler Command/Status area occupies another 256 locations in layer address space. Each command corresponds to one data location (the command with index 0 corresponds to the data location with the matching index). Both command and data areas allow read/write access and the command area, when read, also incorporates status bits. Some of the status bits are “sticky.” To clear them, the command entry must be re-written. As a general rule, the Scheduler should be pre-programmed before enabling the ARINC transmitter, but may also be changed at any time during operation.

The Scheduler Command/status Format is shown in **Figure 1-4** below.

Table 1-4. Scheduler Command/Status Format

Bit	Name	Description	Reset State
31-25	**RSV**	Reserved	0
24	**ECO**	Status only, **sticky**, =1 - Execution Completed Once	0
23	**ME**	Status-only, =1 - Marked for execution	0
22	**EO**	Status only, **sticky**, =1 - Execution Overrun	0
21	**PS1**	PS1,PS0 - Prescaler Source/	0
20	**PS0**	Entry Disable: 00-Disabled 01 - 100 uS 10 - Prescaler 0 11 - Prescaler 1	0
19	**RSV**	Reserved	0
18	**RSV**	Reserved	0
17	**MA**	=1 – Master Entry =0 – Slave entry	
16	**RC**	=1 –Recyclable Entry (valid if MA=1)	0
15-0	**TD**	Time Delay for the selected clock (valid if MA=1)	0

Note that bits 31:22, 19,18 should be reset to 0 for correct operation and for compatibility with future enhancements.

1.8.5.1 Bit Descriptions

Bit Descriptions for the Scheduler (see **Figure 1-4**) are as follows:

- **PS1, PS0** – Prescaler/clock selector and entry disable flag
  - 00 (0)Entry disabled. Disabled entries are ignored by the Scheduler
  - 01 (1)Entry enabled and using 100uS clock as source for time delay
  - 10 (2)Entry disabled and using Prescaler 0
  - 11 (3)Entry enabled and using Prescaler 1
  - If entry is “slave”, this field should match corresponding master entry
- **MA** — “Master” Bit. If set, indicates that entry is a “master” and when scheduled for output should also schedule all related slave entries for output at the same time.
- **RC** — Recyclable Bit. If set, indicates that entry is recyclable. When scheduled for output, the internal value for the time delay counter is cleared and the entry is output again when the time delay expires.

- **TD** — 16-bit time delay. Used to set time intervals for output of master entries (slave entries follow master entries in the schedule). This field is used as a maximum value for the counter, which increments its value each time the 100uS clock or corresponding prescaler produces output. The actual clock source is selected by the PS field.

NOTE: Another powerful feature of the DNx-429-566/512 layer is the ability to generate a “cross-trigger” of the scheduled entry when a selected label with the same index is accepted by Receiver 0 of the same ARINC block. This feature, which may be user-enabled, works only for enabled master entries and may be turned on/off by a dedicated bit in the label filter. When a cross-trigger is detected, the corresponding entry is used to set entries in the Scheduler that may be triggered by the receiver, as clocked by prescaler 0 or 1, but which keep that prescaler disabled. In such cases, only “cross-triggers” can set an entry for output.

Descriptions of the Status-only bits are as follows:

- **ECO** — “Execution Complete at Least Once”. If set, this bit indicates that the entry was output by the ARINC transmitter at least once. This is a sticky bit that can be cleared only by writing a command to the Scheduler. This command may change or may stay the same.

NOTE: Writing to the Scheduler command area clears some internal status bits. It is recommended that you fill all 256 entries with initial information and that you write 0x0 in all unused entries. The DNA-429-566/512 Scheduler command and data areas are accessible regardless of the status of the LIOE (1<<31) bit in LCR and may be accessed at any time.

- **ME** — “Marked for Execution”. This bit is set by the time scheduler for the entry marked for execution (pending transmission) and is cleared for executed (transmitted) entries.

- **EO** — “Execution Overrun”. This bit indicates that a recyclable entry was scheduled for execution while the ME bit was still set. If the EO bit is set, it is likely that data is scheduled in such a way that the ARINC bus does not have enough capacity to transfer it or that too much of the unscheduled data is pumped through the transmit FIFO. This bit is a sticky bit that may be cleared only by writing a command to the Scheduler. The command may change or be the same. The EO bit also triggers an “execution overrun” interrupt for the given transmitter.

1.9 Wiring & Connectors

The following signals are located at the DB-37 female connector on the 429-512/566 layer:

- Rx-A-*n*: Receive line on bus A of channel *n*
- Rx-B-*n*: Receive line on bus B of channel *n*
- Tx-A/B-*n*: Transmit line on bus A (or B) on the 429-566
- Trig-In: Trigger input
- Dout*n*: Digital output line

The pinout for the 429-512/566 is provided below:

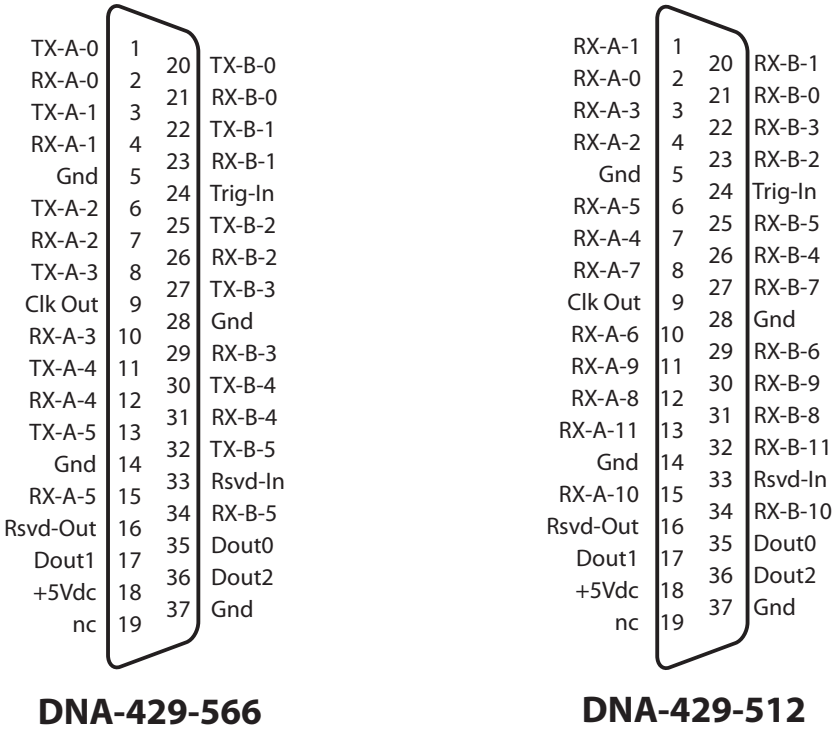


Figure 1-7. DNx-429-512/566 Pinout Diagram

Note that in the DNx-429-566, the six TX channels are numbered as TX-A-0 to TX-A-5 (and TX-B-0 to TX-B-5); and the six RX channels are numbered as RX-A-0 to RX-A-5 (and RX-B-0 to RX-B-5).

In the DNx-429-512, the 12 RX lines are numbered as RX-A-0 to RX-A-11 (and RX-B-0 to RX-B-11).

Each general purpose digital output line has a current sinking (350mA max) low-side FET with 100kΩ pull-up for +5V. Output pin may be left floating or grounded. Each line is protected by a 500 mA resettable fuse between FET and output pin.

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

Figure 1-8 shows an example of how to wire the ARINC-429 bus in a multi-drop network configuration (which is the most common) with the DNx-429-5xx. In **Figure 1-8**, the DNX-429-566's Channel0 is acting as the master transmitter and all other devices are slaves (receiving only). The physical medium between master and slave(s) is two twisted-pair wires carrying a differential signal:

- ***a*** is for the non-inverting signal (may be inaccurately labelled as +)
- ***b*** is for the inverting signal (may also be inaccurately labelled as -)

The signals from a/b lines are with reference to the ground. Even though the configuration in **Figure 1-8** is common, other configurations are also possible.

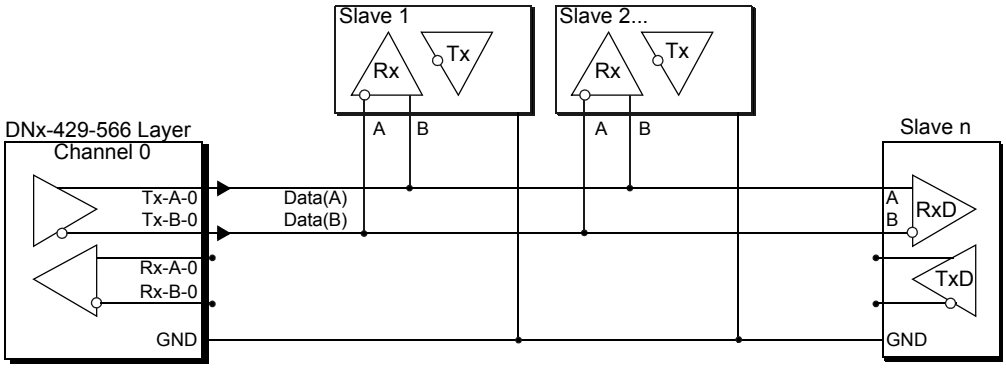


Figure 1-8. Wiring for a ARINC-429 network

Chapter 2 Programming with the High Level API

This section describes how to control the DNx-429-512/566 using the UeiDaq Framework High Level API.

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, LabVIEW, or DASYLAB.

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

Start » Programs » UEI » Framework » Examples

The following section focuses on the C++ API, but the concept is the same no matter what programming language you use.

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

2.1 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 session;
```

2.2 Configuring the Resource String

UeiDaq Framework uses resource strings to select which device, subsystem and channels to use 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**.

ARINC layers have dedicated input and output ports. Use the ATX token for the output subsystem and the ARX token for the input subsystem.
For example, the following resource string selects ARINC input ports 0,2,3 on device 1 at IP address 192.168.100.2: “pdna://192.168.100.2/Dev1/ARX0,2,3”.

In addition to the resource, you will also configure:

- **Port Speed** of 12.5K or 100K for a selected Building Block. Note that for the 429-512 it corresponds to configuration of port pairs.
- **Parity Mode** to use for controlling transmission integrity.

For input channels, you will configure these additional parameters:

- **SDI Filter**: used to filter incoming words based on their SDI bits.
- **SDI Mask**: if the result is other than zero, each incoming word is ANDed with this mask and the word is passed on to the software.

```
// Configure ARINC input ports 0,2,3 on device 0
session.CreateARINCInputPort( "pdna://192.168.100.2/Dev0/ARX0,2,3",
                              UeiARINCBitsPerSecond12500,
                              UeiARINCParityOdd,
                              false, 0);
```

For output channels on the 429-566, no additional parameters are necessary:

```
// Configure ARINC output ports 0, 2, 5 on device 0

session.CreateARINCOutputPort("pdna://192.168.100.2/Dev0/ATX0,2,5",
                              UeiARINCBitsPerSecond12500,
                              UeiARINCParityOdd);
```

2.3 Configuring the Timing

The application must configure the 429-566 to use the “messaging” timing mode. Messages are ARINC words, represented in C++ with the structure `tUeiARINCWord`.

```
typedef struct _tUeiARINCWord
{
   // The label of the word. It is used to determine the data type of the
   // Data field, and therefore, the method of data translation to use.
   uInt32   Label;

   // Sign/Status Matrix or SSM. This field contains hardware equipment
   // condition, operational mode, or validity of data content.
   uInt32   Ssm;

   // Source/Destination Identifier or SDI. This is used for multiple
   // receivers to identify the receiver for which the data is destined.
   uInt32   Sdi;

   // The parity bit.
   uInt32   Parity;

   // The payload of the word. Its format depends on the label.
   // Most common formats are BCD (binary-coded-decimal) encoding,
   // BNR (binary) encoding or discrete format where each bit represents
   // a Pass/Fail, True/False or Activated/Non-Activated condition.
   uInt32   Data;

} tUeiARINCWord;
```

The ARINC-566 can be programmed to wait for a certain number of messages 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 messages before notifying the session.

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

```
// configure timing

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

2.4 Read Data

Reading data from the 429-566 is done using a reader object. Since 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 input 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 words from the ARINC bus.

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

reader = new CUeiARINCReader(session.GetDataStream(), 1);

// read up to 10 words, numWordsRead contains the
// number of words actually read.

tUeiARINCWord words[10];
reader->Read(10, words, &numWordsRead);
```

2.5 Write Data

Writing data to the 429-566 is done using a writer object. Since 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 output 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 one word to the ARINC bus.

```
// Create a writer and link it to the session's stream, port 2

writer = new CUeiARINCWriter(session.GetDataStream(), 2);

// store the one word we want to write out

tUeiARINCWord word;
word.Label = 2;        // Set the label
word.Data = 0x123;     // Set the payload
word.Sdi = 0
word.Ssm = 0
word.Parity = 0;

// write 1 word, numWordsWritten contains number of words actually sent

writer->Write(1, &word, &numWordsWritten);
```

2.6 Programming the Output Scheduler

Each output channel is equipped with a hardware Scheduler that you can use to send sequences of ARINC words at a given rate without intervention of the host or and/or software.

You first need to add a pointer on the output channel that you wish to program:

```
// Configure ARINC output ports 0, 2, 5 on device 0

CUeiARINCOutputPort* pPort =
dynamic_cast<CUeiARINCOutputPort*>(session.GetChanel(index));
```

You can then add up to 256 Scheduler entries. Each entry is represented by the following structure:

```
typedef struct _tUeiARINCSchedulerEntry
{
   // Specifies whether this is a master or slave entry
   Int32  Master;

   // Specifies whether this entry should be scheduled periodically
   Int32   Periodic;

   // Scheduling delay count in us
   uInt32  Delay;

   // Word to be sent when this entry is processed.
   tUeiARINCWord  Word;

} tUeiARINCSchedulerEntry;
```

Add each entry one by one using the AddSchedulerEntry() method on the port object:

```
// Add one Scheduler entry

tUeiARINCSchedulerEntry entry = {1, 1,1000000, {0x12, 0, 0, 0, 0x300} };
pPort->AddSchedulerEntry(entry);
```

As soon as the session starts, the output port starts sending the scheduled words in sequence. You can update the scheduled words while the session is running with CUeiARINCWriter::WriteScheduler() as follows:

```
// Update the second word in scheduler table

tUeiARINCWord word;
word.Label = 23; // Set the label
word.Data = 0x102; // Set the payload
word.Sdi = 0
word.Ssm = 0
writer->WriteScheduler(2, 1, &word, &numWordsWritten);
```

To change the number of scheduled words and/or their frequencies, you must first stop the session, and then add additional words in the existing sequence or clear the Scheduler with the following call:

```
// Clear the scheduler

pPort->ClearSchedulerEntries();
```

2.7 Program the Label Filter

Each input channel is equipped with a label filter that lets you filter out words with a given label before they are received by the software.

You first need to get a pointer on the ARINC input channel you wish to program:

```
// Get pointer to the input channel

CUeiARINCInputPort* pPort =
dynamic_cast<CUeiARINCInputPort*>(session.GetChanel(index));
```

You can then add up to 255 filter entries. Each entry is represented by the following structure:

```
typedef struct _tUeiARINCFilterEntry
{

   // Label to accept
   uInt32                    Label;

   // Accept word only if it carries different data from a
   // previously received word with the same label.
   Int32                     NewData;

   // Trigger the scheduler entry with the same index as this
   // filter entry in the scheduler table of the output port
   // that has the same index as this input port.
   Int32                     TriggerSchedulerEntry;

} tUeiARINCFilterEntry;
```

Add each entry one by one using the `AddFilterEntry()` method on the port object:

```
// Add an entry to filter

tUeiARINCFilterEntry entry = {12, 1, 1};
pPort->AddFilterEntry(entry);
```

As soon as the session starts, the input port starts filtering words that match the specified label(s).

To reprogram the filter, you must stop the session first. You can then add additional filters or clear the existing filters with the following call:

```
// Clear the filter

pPort->ClearFilterEntries();
```

2.8 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

session.CleanUp();
```

Chapter 3 Programming with the Low-level API

This chapter illustrates how to program the PowerDNA cube using the low-level API. The low-level API offers direct access to PowerDNA DAQBios protocol and also allows you to access device registers directly.

However, we recommend that, when possible, you use the UeiDaq Framework High-Level API, because it is easier to use. You should need to use the low-level API only if you are using an operating system other than Windows.

For additional information about low-level programming of the 429-512/566, please refer to the PowerDNA API Reference Manual document under:

Start » Programs » UEI » PowerDNA » Documentation

Refer to the PowerDNA API Reference Manual on how to use the following low-level functions of 429-512/566, as well as others related to cube operation:

Function	Description
DqAdv566BuildPacket/ DqAdv566ParsePacket	Assemble an ARINC message out of individual fields for transmission and split a received packet into individual fields.
DqAdv566BuildFilterEntry	Assembles filter entries from the separate fields.
DqAdv566BuildSchedEntry	Assembles a scheduler entry from the separate fields.
DqAdv566SetConfig	Configures basic ARINC device parameters.
DqAdv566SetMode	Configures the mode of operation for each channel.
DqAdv566SetFilter	This function writes to or reads from the label filter.
DqAdv566SetScheduler	This function writes to or reads from scheduler table(s)
DqAdv566SetSchedTimebase	Sets one of two programmable scheduler timebases at a time
DqAdv566SetFifoRate	Sets up a timebase for writing output packets to the Tx FIFO
DqAdv566SetChannelCfg	Set up operating mode for each channel
DqAdv566Enable	Enables and disables all operations on the layer
DqAdv566SendPacket	Sends a packet to the specified channel
DqAdv566SendFifo	Put packets of data into the Tx FIFO
DqAdv566RecvPacket	Receives a packet of data from the Rx interface.
DqAdv566RecvFifo	This function retrieves messages from the Rx FIFO.
DqAdv566ReadWriteFifo	Writes and receives packets from a specified channel FIFO.
DqAdv566ReadWriteAll	Writes and receives packets from specified channel FIFOs.
DqAdv566GetStatus	Requests the error and status of the interface.
DqAdv566EnableByChip	Enables/disables operations on the layer on IC per IC basis.
DqAdv566SetChannelList	Sets up the channel list for ARINC 566 operations in messaging mode.

Appendix

A. Accessories

The following cables and STP boards are available for the 429-512/566 layer.

DNA-CBL-37

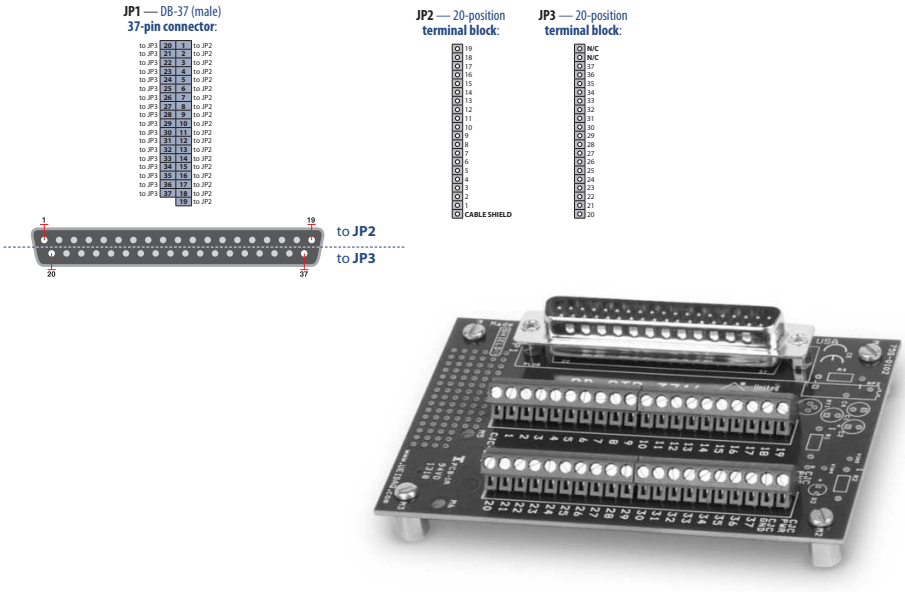
This is a 37-conductor flat ribbon cable with 37-pin male D-sub connectors on both ends. The length is 3ft and the weight is 3.4 ounces or 98 grams.

DNA-CBL-37S

This is a 37-conductor round shielded cable with 37-pin male D-sub connectors on both ends. It is made with round, heavy-shielded cable; 3 ft (90 cm) long, weight of 10 ounces or 282 grams; also available in 10ft and 20ft lengths.

DNA-STP-37

The DNA-STP-37 provides easy screw terminal connections for all DNA and DNR series I/O boards which utilize the 37-pin connector scheme. The DNA-STP-37 is connected to the I/O board via either DNA-CBL-37 or DNA-CBL-37S series cables. The dimensions of the STP-37 board are 4.2w x 2.8d x1.0h inch or 10.6 x 7.1 x 7.6 cm (with standoffs). The weight of the STP-37 board is 2.4 ounces or 69 grams.



DNA-STP-37D

The DNA-STP-37D is a direct-connect terminal panel.

Index

B
Block diagram 6

C
Cable(s) 24
Cleaning-up the Session 20
Configurations 3
Configuring the Resource String 18
Conventions 2
Creating a Session 18

D
DNx-429-512 3
DNx-429-566 3

E
Emergency Transfer Transmitter 14

F
FIFO 13

H
Hardware Abstraction Layer 13
High Level API 18

J
Jumper Settings 17

L
Label Acceptance Filter 12
Loopback Connection 10
Low-level API 23

O
Organization 1

P
Prescalers 13
Priority Registers 13

R
Receiver Block 10
Receiver Diagram 11

S
Scheduler 13, 14
Scheduler ARINC Data Format 8
Scheduler Command/Status Format 15
Screw Terminal Panels 24
Setting Operating Parameters 4
Software Options 3
Support ii

T
Transmitter Block 13
Transmitter Block Diagram 13

W
Waveform Characteristics 7
Wiring 17
Word Format 8