United Electronic Industries

The High-Performance Alternative

DNx-429-516

—

User Manual

**16 ARINC-429 channels, configured as Receivers or Transmitters,
plus 8 dedicated ARINC-429 Receiver-only channels
-
Interface board with Guardian functionality
for the PowerDNA Cube and RACK series chassis**

February 2019

PN Man-DNx-429-516

Information furnished in this manual is believed to be accurate and reliable. However, no responsibility is assumed for its use or for any infringement of patents or other rights of third parties that may result from its use.

All product names listed are trademarks or trade names of their respective companies.

See the UEI website for complete terms and conditions of sale:
http://www.ueidaq.com/cms/terms-and-conditions/

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 a member of our support team:

Support:

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

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

Internet Support:

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

Product Disclaimer:

WARNING!

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

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

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

Table of Contents

List of Figures

Chapter 1 Introduction

This document outlines the feature set and use of the DNx-429-516 ARINC-429 boards.

The DNx-429-516 is an ARINC-429 interface that provides 24 ARINC channels: 16 channels that can be individually configured as ARINC-429 transmitters or receivers plus an additional 8 dedicated ARINC-429 receiver channels.

NOTE: Note that the information in this manual is also applicable for the preliminary revision (p/n DNx-429-516-024).

This chapter contains the following sections:

- Organization of This Manual (Section 1.1)
- DNx-429-516 Board Overview (Section 1.2)
- Features (Section 1.3)
- Indicators (Section 1.4)
- Specification (Section 1.5)
- Device Architecture (Section 1.6)
- ARINC-429 Word Format (Section 1.7)
- Receiver Block (Section 1.8)
- Transmitter Block (Section 1.9)
- Wiring & Connections (pinout) (Section 1.10)

1.1 Organization of This Manual

This *DNx-429-516 User Manual* is organized as follows:

- **Introduction**
  Chapter 1 provides an overview of the DNx-429-516 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**
  Chapter 3 describes low-level API commands for configuring and using the DNx-429-516 series board for serial operating modes.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for use with the DNx-429-516 serial-line communication interface board.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

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

Manual Conventions

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

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

NOTE: Notes alert you to important information.

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

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

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

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

Before you begin:

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

No HOT SWAP

Always turn POWER OFF before performing maintenance on a UEI system. Failure to observe this warning may result in damage to the equipment and possible injury to personnel.

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 DNx-429-516 Board Overview

The DNx-429-516 board is an ARINC 429 communications interface for UEI's Cube and RACK chassis. The DNx-429-516 offers 16 channels that are individually configurable as transmitters or receivers and additionally 8 dedicated receiver channels.

The TX drivers on the TX/RX channels can be disabled on a channel-by-channel basis. To use a TX/RX channel as RX you do not enable the TX driver on that channel; enabling TX drivers is controlled via UEI API.

1.2.1 Data Rate

The boards comply with the ARINC-429 specification and run at either a high speed (100 kHz) or low speed (12.5 kHz) baud rate. Additionally, the channel speed can be set to frequencies other than 100 kHz and 12.5 kHz to support legacy devices. The speed is software-selectable on a per channel basis. To ensure data integrity, 256-word FIFOs are provided on every TX and RX channel.

1.2.2 Guardian Diagnostic Support

DNx-429-516 boards are part of UEI's Guardian series, which equips the boards with additional test and diagnostic capabilities. Each transmit channel connects to an on-board ARINC-429 receiver, allowing data being transmitted on the 429 bus to be monitored using the on-board receivers.

1.2.3 ARINC Transmitters

Channels may be set to transmit asynchronously or based on a hardware controlled scheduler. Each channel supports a transmission table that allows up to 256 unique schedules. Transmission schedule resolution is 100 µs.

Asynchronous (non-scheduled) data may be sent with three priorities. High priority data is sent immediately upon completion of the current transmission, regardless of scheduled messages. Data sent with standard priority is transmitted during times when no scheduled data is being sent. Finally, the lowest priority is data streamed from a 256 word FIFO which is sent when no scheduled, high or standard priority data is being transmitted.

1.2.4 ARINC Receivers

The DNx-429-516 series provides a variety of filtering capabilities. The board can be configured to only return data from specific labels. Data from up to 255 labels may be selectively read or the board can be set to read data from all labels.

A "new data only" filter can be enabled, which compares the received label data to the most recent previous reading and only returns data if something has changed.

Data may also be filtered based on the SDI bits and parity errors.

1.2.5 Software Support

Software support for the DNx-429-516 is included with the board. Factory written and supported drivers are included for Linux and are available for other popular real-time operating systems including QNX and VxWorks.

Users can develop applications using "low-level" C-based libraries or using the UEIDAQ Framework, which provides a comprehensive, easy to use API supporting all popular Windows programming languages. The UEIDAQ Framework supplies complete support for those creating applications in all popular data acquisition and control packages, including LabVIEW, MATLAB, as well as any application which supports ActiveX or OPC servers.

1.3 Features

DNx-429-516 features are listed below:

- 16 ARINC-429 individually configurable as TX or RX plus 8 dedicated RX channels (allowing for up to 24 RX channels)
- High Speed (100 kHz) or Low Speed (12.5 kHz) standard baud rate, selectable per channel; programmable custom rates from 10 kbaud to 200 kbaud
- Hardware Label filtering
- Hardware Transmit Scheduler (100 µS timing resolution)
- Automatic timestamping of data, software enabled/disabled
- Guardian Series Diagnostics, on-board 429 RXs allow read-back on each TX channel
- 350 V_{rms} isolation between blocks of three channels and between chassis and channels
- 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-429-516

1.4 Indicators

The DNx-429-516 indicators are described in **Table 1-1** and illustrated in **Figure 1-1**.

Table 1-1 429-516 Indicators

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

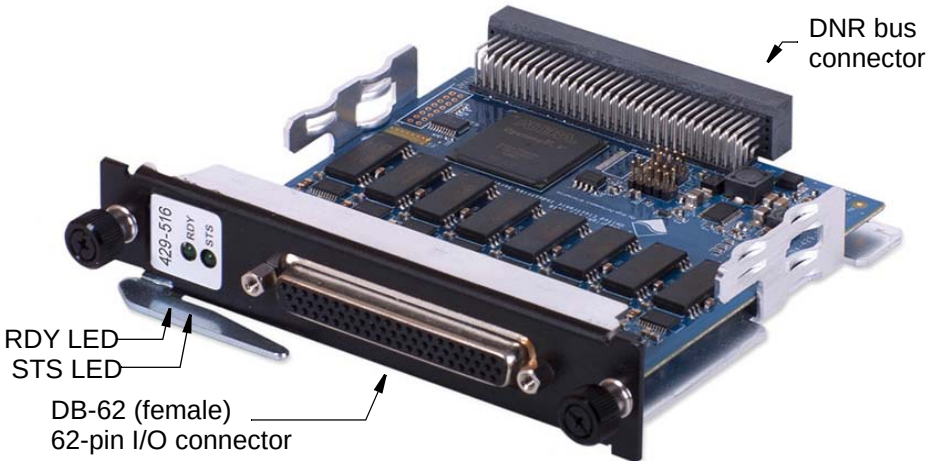


Figure 1-1 The DNR-429-516 ARINC-429 Board

1.5 Specification

The technical specification for the DNx-429-516 is provided in the table below:

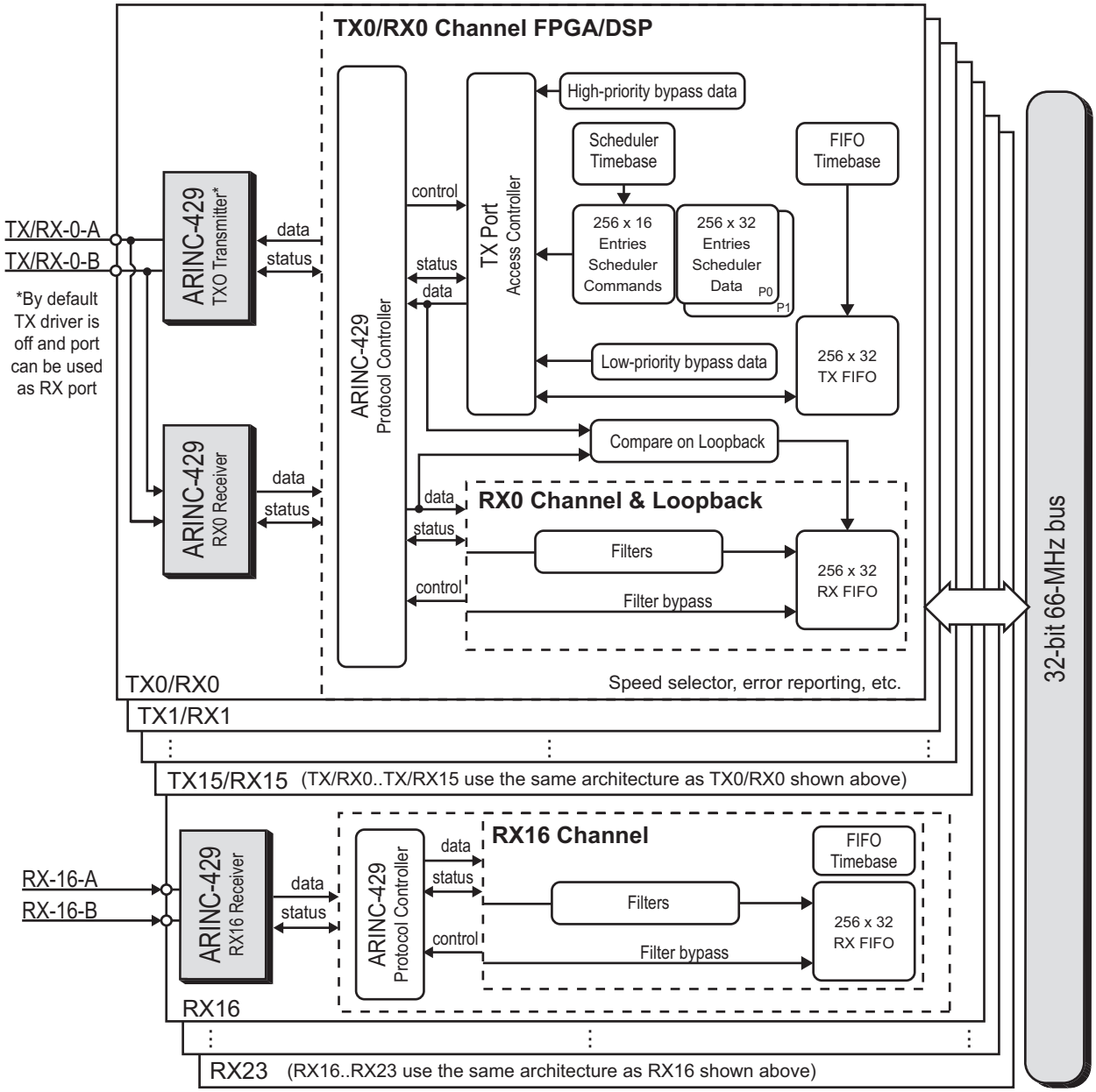
Table 1-2 . DNx-429-516 Technical Specifications

Channel Configurations	
Number of channels	16 TX or RX and 8 dedicated RX channels
ARINC Compliance	Fully compliant with ARINC 429
Total RX loads	20 per channel, 128 maximum per board
Transmit Specifications	
Standard Data rate	100 kHz or 12.5 kHz selectable per channel
Custom data rates	10 kHz to 200 kHz for special applications
FIFO size (TX or RX)	256 words
Transmit modes	Scheduled or asynchronous. TX outputs may be disabled allowing a channel to be used as a Input.
TX Scheduler specifications	
timing resolution	100 microseconds
table size	Schedule up to 256 labels per channel
Minor/Major Frames	16 Minor frames with double buffering of data array
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
Receive Specifications	
Standard Data rate	100 kHz or 12.5 kHz selectable per port
Custom data rates	10 kHz to 200 kHz for special applications
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
General Specifications	
Isolation	350 Vrms. Isolation provided in channel pairs. Channels 0-1, 2-3,...14-15 share a common ground
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	7 Watt, maximum

NOTE: A DNx-429-516 transmitter can connect to 1 to 20 receiver(s) on one twisted pair, per ARINC-429 hardware specifications. However, note that due to power restrictions, the DNx-429-516 board can only drive a maximum of 128 receivers per board.

1.6 Device Architecture

The DNx-429-516 board incorporates 24 parallel ARINC channels. Refer to **Figure 1-2** for block diagram.



Channels are grouped into 8 isolated blocks of 3 channels each: (TX0/RX0, TX1/RX1, RX16), (TX2/RX2, TX3/RX3, RX17), etc.

Figure 1-2 DNA/DNR-429-516 Logic Block Diagram

Each of the 16 TX/RX channels contains one ARINC transmitter and one receiver (channel 0..15). On power up, TX drivers are off, and the channel can be used as an ARINC receiver after enabled. If TX is enabled, the channel is set up to transmit messages and the receiver side of the channel can be used to loopback transmitted data for diagnostic purposes.

Additionally, the 429-516 provides 8 dedicated RX-only channels (RX16..RX23) that contain an ARINC receiver each.

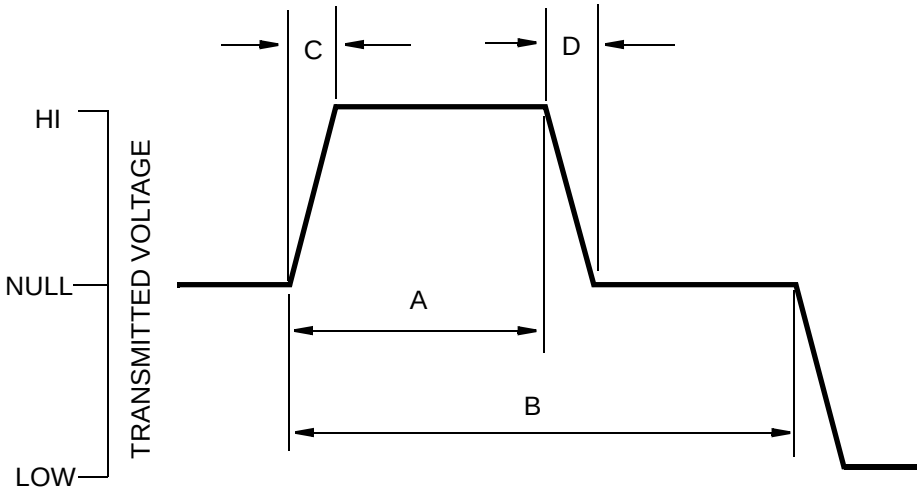
The DNx-429-516 ARINC communication protocol functions are embedded into the FPGA. TX channels are driven by associated Holt HI-8596 line drivers. Dedicated RX channels use Holt HI-8455 receivers.

Channels are grouped into eight isolated blocks of 3 channels each:

- Block 0: TX0/RX0, TX1/RX1, RX16
- Block 1: TX2/RX2, TX3/RX3, RX17
- Block 2: TX4/RX4, TX5/RX5, RX18
- Block 3: TX6/RX6, TX7/RX7, RX19
- Block 4: TX8/RX8, TX9/RX9, RX20
- Block 5: TX10/RX10, TX11/RX11, RX21
- Block 6: TX12/RX12, TX13/RX13, RX22
- Block 7: TX14/RX14, TX15/RX15, RX23

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, which can drive up to 20 external receivers.

Waveform characteristics must conform to the specifications described in **Figure 1-3**.



		Hi Speed	Low Speed
A	First half of pulse	5 usec ± 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

1.7 ARINC-429 Word Format

Each ARINC-429 word is a 32-bit value containing 5 fields, as shown in **Figure 1-4** and described in **Table 1-3**.

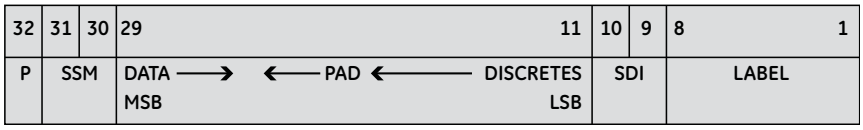


Figure 1-4 General ARINC Word Format

Table 1-3 Field Descriptions for ARINC-429 Words

Bit Values	Name	Description
32	PARITY	PARITY is used to check the validity of the ARINC word. Supported values are Odd, Even, and No Parity. Refer to Section 1.7.1 for more information.
31:30	SSM	Sign/Status Matrix (SSM) indicates the status of data. Values for BCD data formats: [11] Minus, South, West, Left, From, Below [10] TEST indicates data comes from a test source [01] NCD indicates data is missing because of non-hardware reason [00] Plus, North, East, Right, To, Above Values for BNR data formats: [11] OP indicates data is normal [10] TEST indicates data comes from a test source [01] NCD indicates data is missing because of non-hardware reason [00] FAIL indicates a hardware failure causing data to be missing See ARINC-429 protocol documentation for additional information regarding format details.
29:11	DATA	DATA contains 19 bits of data to transmit. Supported formats include bit-field, BCD, BNR, or mixed.
10:9	SDI	Source/Destination Identifiers (SDI) indicate the intended transmitter sending the data.
8:1	LABEL	LABEL identifies the data type. The Label is encoded in octal format and transmitted MSB first.

NOTE: Use of SDI and SSM is not mandatory.

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 first, MSB first. After the label is sent, all other data fields are transmitted LSB first.

1.7.1 Parity

Parity can be configured as Odd, Even, or No Parity.

For transmitters:

Selecting Odd or Even enables the DNx-429-516 transmitters to calculate parity and set the PARITY bit value in the outgoing word. When Odd or Even parity is selected, parity cannot be “forced” by the user.

- For TX Odd parity, the DNx-429-516 parity generator counts the number of 1s in bits 31..1 of the outgoing word. It auto-inserts bit 32 as a 1 or 0 to make the total number of ONES in the 32-bit ARINC word odd.
- For TX Even parity, the DNx-429-516 auto-inserts a 1 or 0 in bit 32 to make the total number of ONES even.
- When TX No Parity is selected, the DNx-429-516 outputs whatever bit 32 value is supplied by the user in the ARINC data word. This can be used for injecting parity errors by the user.

For receivers:

The DNx-429-516 receivers count the ONES in the received frame and calculate the value of the expected parity bit, according whether RX parity is programmed as Odd or Even.

The PARITY bit of the received message will be set as follows:

- For RX Odd parity, the 429-516 reports bit 32 (the parity bit) as
  - 1 if there is a parity error (Even number of ONES in incoming word)
  - 0 if the incoming word is correct (no parity error)
- For RX Even parity, the 429-516 reports bit 32 (the parity bit) as
  - 1 if there is a parity error (Odd number of ONES in incoming word)
  - 0 if the incoming word is correct (no parity error)
- When RX No Parity is selected, the 429-516 writes bit 32 as the actual PARITY bit that was received, regardless of parity errors.

The actual received parity bit (bit 32) is matched against the expected value; if the actual parity bit doesn’t match the expected, a parity error is generated. Users have the option of programming the DNx-429-516 to ignore data from received frames with parity errors.

1.7.2 DNx-429-516 Word Format

The DNx-429-516 provides industry standard ARINC-429 protocol controllers to process data.

ARINC input data will need to be preformatted in a form that is common to protocol controllers, such as the Holt HI-3282 format (www.holtic.com).

When programming using the DNx-429-516 Framework, users provide Label, Data, SDI, SSM, and Parity fields and the Framework handles formatting (refer to **Chapter 2**). When programming using low-level API, functions are provided to handle formatting (refer to **Chapter 3**).

Table 1-4 below provides a cross reference of Holt HI-3282 format, DNx-429-516 format, and ARINC-429 standard industry format.

Table 1-4 . Cross Reference of Bit Numbering for Various Protocols

Holt Bit No. (compatible with DNx-429-566/512)		DNx-429-516 Bit No.	ARINC Standard Bit No.	Name	Description
15	Word 2	31	29	**SIGN**	Sign bit for data
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 (SDI). See ARINC-429 protocol documentation for details
10-9		10-9	31-30	**SSM**	Sign/Status Matrix (SSM) or data bits. See ARINC-429 protocol documentation for details
8		8	32	**PARITY**	Parity Bit. See Section 1.7.1 above for more information.
7		7	1	**LABEL**	Label. Used to identify data types and associated parameters. For RX ports, DNx-429 boards have a flexible acceptance filter with an option to put only new data into the RX 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		

NOTE: The ARINC serial communication standard numbers bits from 32 (MSB) to 1 (LSB). The UEI DNx-429-516 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.

NOTE: Use of SDI and SSM is not mandatory.

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 first, MSB first. After the label is sent, all other data fields are transmitted LSB first.

1.8 Receiver Block

The DNx-429-516 provides the following ARINC receivers:

- 16 ARINC receivers (RX0..RX15) that share I/O pins with the 16 ARINC transmitters (TX0..TX15)
- 8 dedicated ARINC receivers (RX16..RX23)

RX0..RX15 ports can receive ARINC messages from an external source or receive looped back messages from corresponding DNx-429-516 transmit channels (TX0..TX15). When transmitters are enabled, transmit frames are looped back into the corresponding ARINC receiver for storage in the RX FIFO and/or comparison with the last transmitted frame. These receivers include additional comparators and diagnostic circuitry to provide loopback status and data read from transmitters.

RX16..RX23 function as 8 ARINC receive-only ports.

1.8.1 Receiver Block Components

Each receiver block consists of the following components:

- 256-word RX FIFO
- RX filters: Label acceptance filter, store new data only circuitry, parity error data rejection filter, SDI filtering enable/disable
- “Last received” data memory

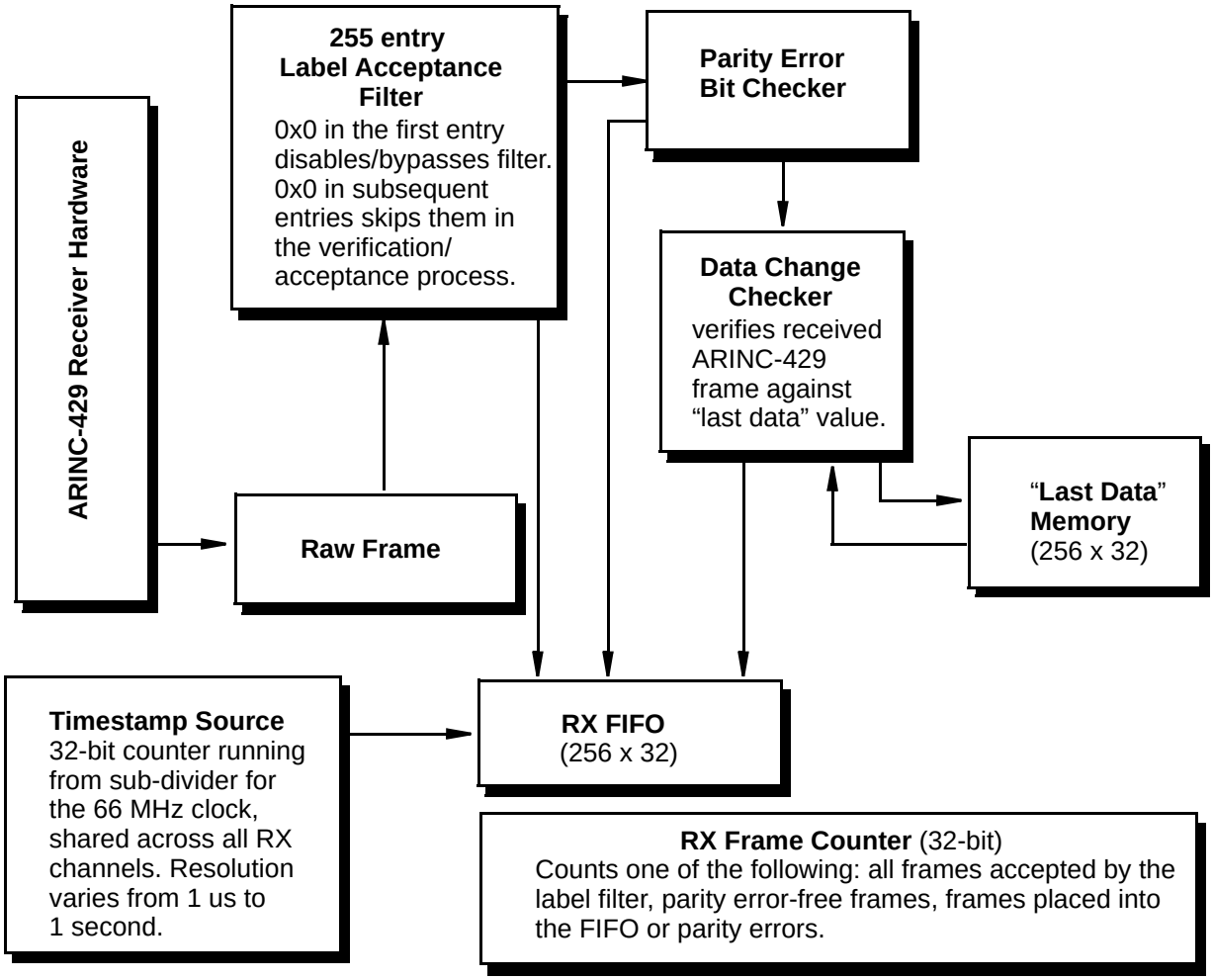


Figure 1-5 Receiver Diagram

Figure 1-5 shows a block diagram of the ARINC receiver functionality.

As illustrated, received data is stored directly in the 256-word FIFO when filters are not enabled.

Filtering and timestamping can be enabled by choosing options when setting up the receiver mode.

Refer to **Chapter 2** for more information about programming using the high-level Framework API. Refer to **Chapter 3** for more information about using the low-level C-based API.

1.8.2 RX FIFO

The RX FIFO can store up to 256 of the 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 filters are not selected, data is placed directly into the 256-word FIFO.

The timestamp source is a 32-bit counter running from a subdivider of the 66 MHz clock. Timestamps are shared across all channels.

1.8.3 RX Filters

The DNx-429-516 has the capability of accepting or rejecting incoming ARINC data frames on an individual port using any of the following features:

- Filtering Frames Based on Predefined Labels
- Filtering Frames with Parity Errors
- Filtering Frames Based on a Predefined SDI

1.8.3.1 Filtering Frames Based on Predefined Labels

Users can filter received frames based on a set of up to 255 labels that are user-programmed into the Label Acceptance Filter table.

As an option, the receiver can additionally be programmed to accept only changed data. This option compares incoming ARINC frames with the last received frame and stores only changed frames into the RX FIFO. Unchanged data is discarded.

1.8.3.2 Filtering Frames with Parity Errors

429-516 hardware calculates the parity of each incoming ARINC word based on whether the channel is configured as even or odd parity. The actual received parity bit (bit 32) is matched against the expected value; if the actual parity bit doesn't match the expected, a parity error is generated.

Users have the option of rejecting incoming frames with parity errors and only storing frames with correct parity to the RX FIFO.

1.8.3.3 Filtering Frames Based on a Predefined SDI

Users additionally have the option of filtering incoming frames based on the SDI field. When SDI filtering is enabled, users can specify a mask (SDIMASK) of the SDI value they wish to filter for and only messages with that SDI value will be stored in the RX FIFO.

1.8.4 TX to RX Loopback Features

Hardware loopback through the DNx-429-516 RX0..RX15 receivers supports the same filtering options as receivers in non-loopback mode, as well as additional diagnostic features and status information.

For the RX0..RX15 receivers, hardware that compares the last TX frame with the looped-back RX frame provides an option to store the following data if the comparison fails:

- Expected TX frame
- Received RX frame
- Timestamp
- Error counters
- TX frame source, (e.g., Scheduler with entry number, TX FIFO, etc.) and error flags

The comparator may include or exclude the parity bit from the compare algorithm. This option ignores data with a bad parity bit setting.

1.9 Transmitter Block

Each transmitter port (TX0..TX15) is composed of the following hardware:

- sources for the data being transmitted
- scheduler and timebase hardware
- port access control state machine
- an ARINC-429 protocol controller

Users can write to four separate prioritized hardware sources to store data for transmission. Hardware sources for queuing transmit data include a high-priority data register, 256-word scheduler arrays, a low-priority data register, and a 256-word TX FIFO.

Data is passed to the ARINC protocol controller and transmitter hardware for transmission to externally designated receivers. Data can also be looped back to internal DNx-429-516 receivers as a diagnostic capability.

NOTE: Each transmit channel can be disabled for connection to redundant systems. Disabled transmit channels will be tristated when the board is powered up. When the board is powered down, transmit channels are grounded.

A block diagram of hardware sources for transmit data and timebase hardware is shown in **Figure 1-6**.

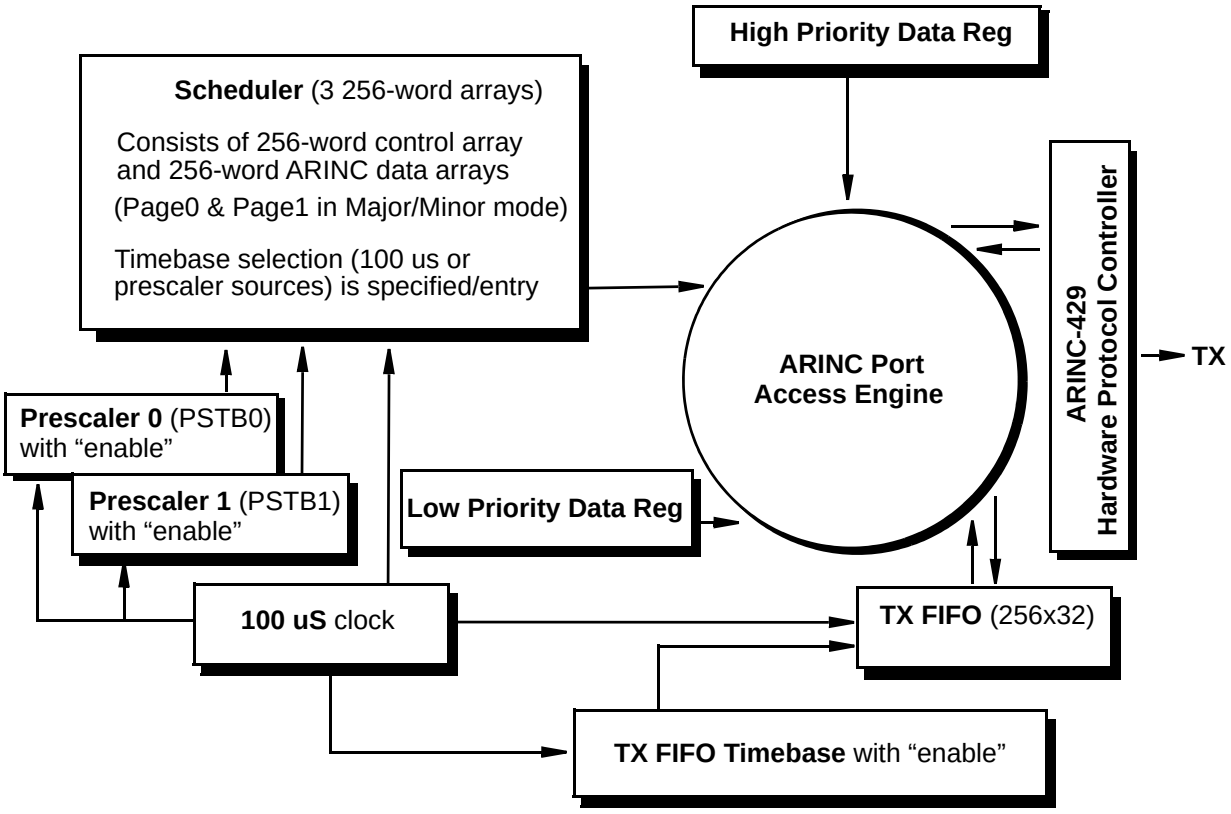


Figure 1-6 Transmitter Block Diagram

1.9.1 Hardware Sources for Transmit Data

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

Priority	Transmitter Source	Description
Highest	High-priority Data Register	The high-priority register transfers data immediately after the current TX operation is completed.
Second	Scheduler Data	The Scheduler transfers data from a data array to the ARINC bus based on the Scheduler mode configuration and also based on the timing specified in the corresponding Scheduler control word. The 429-516 Scheduler supports the following operational modes: • Scheduler in Default operation • Scheduler in Frame Clock mode • Scheduler in Major/Minor Frame mode Refer to Section 1.9.2 for more information.
Third	Low-priority Data Register	The low-priority register only transfers data when the data from the Scheduler and high-priority register are not available.
Lowest	TX FIFO	The 256-word output TX FIFO runs at the lowest priority and may output data in either of two modes: • whenever the interface IC can accept data and none is available from the Scheduler • on a “paced” mode, based on user-defined clock intervals The TX FIFO is used by the Scheduler and priority data registers as temporary storage. The TX FIFO should be enabled for all transmissions. Additionally, the TX FIFO supports a delayed transmission mode. Refer to Section 1.9.3 for more information.

Table 1-5 Transmitter Source Priority

NOTE: The Scheduler and FIFO priorities may be swapped by setting a command bit using the `DqAdv566SetMode()` low-level API function.

Refer to Section 2.7 on page 32 for more information regarding programming using the high-level Framework API, and refer to **Chapter 3** for more information about low-level programming.

1.9.2 Scheduler

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

As a general rule, the Scheduler should be pre-programmed before enabling the ARINC transmitter; however, programming can be changed at any time during operation.

The 429-516 Scheduler supports the following operational modes:

Scheduler Mode	Description	Section
Default Operation	Scheduling is based on master/slave word groupings. Master control words are configured with a user-programmable delay from the enabling of the TX channel, which is also the delay between periodic master transmissions. Slave words always directly follow master words.	Section 1.9.2.2
Frame Clock	Scheduling is based on master/slave word groupings. Master control words are configured with a user-programmable delay and frame repeat rates. Slave words can be scheduled on multiples of master word frames.	Section 1.9.2.3
Major/Minor Frame	Scheduling is based on major/minor frames. A major frame clock triggers a sequence of minor frames, which are configured with user-programmable frame repeat rates and user-programmable time offsets relative to the start of a major frame. • ARINC data is scheduled for transmission based on which minor frame it is mapped to • the same data can be mapped to multiple minor frames	Section 1.9.2.4

Table 1-6 Summary of Scheduler Operational Modes

1.9.2.1 Scheduler Hardware

The Scheduler table consists of a control word array, which contains scheduling information and control flags, and a corresponding 256-word data array that contains the associated ARINC data to be transmitted.

When the Scheduler is in Major/Minor Frame mode, two data arrays are available, which provide two 256-word pages, Page0 and Page1, and allow data to be transmitted from one page while new data for transmit is written to the other page.

Each control word and corresponding data location are indexed as one entity (the control word with index 0 corresponds to the data location with the matching index).

Each scheduler entry can be enabled and disabled separately, and control and data arrays are read and write accessible. The control word, when read, also incorporates status bits, some of which are “sticky.” To clear sticky status bits, the control entry must be re-written. Refer to **Table 1-7** for status bit descriptions.

Table 1-7 Scheduler Status Bit Descriptions

Status Bit	Bit Name	Description
ECO	Execution Complete Status Flag	Indicates the scheduled entry has executed and 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.
ME	Marked for Execution Status Flag	Indicates the following conditions: • Set by the time scheduler when an entry is marked for execution (pending transmission) • Cleared by the time scheduler for executed (transmitted) entries
EO	Execution Overrun Status Flag	Indicates that a periodic entry was scheduled for execution while the ME bit was still set. NOTE: If the EO bit is set, the data may be scheduled such 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 EO bit also triggers an “execution overrun” interrupt for the given transmitter.

1.9.2.2 Scheduler in Default Operation

In Default operation, the scheduler array is written with a series of user-defined master/slave control words and corresponding ARINC transmit messages (data). The control word specifies the timing and master/slave groupings of the word(s) to be shifted out of the transmitter.

In Default operation, the slave words directly follow the master word. Scheduling is set by the master control word entry.

Scheduler Table

Array Index	Scheduler Control Word	Scheduler Data Word (for TX)
0	Master=TRUE; Periodic=TRUE; Prescaler=100us; DelayCounter=dly_ctr	Master Data Word: MData 1
1	Master=FALSE; Periodic=TRUE; Prescaler=100us; DelayCounter=dly_cntr	Slave 1 Data Word: SData 1
2	NULL	NULL
⋮	⋮	⋮
255	NULL	NULL

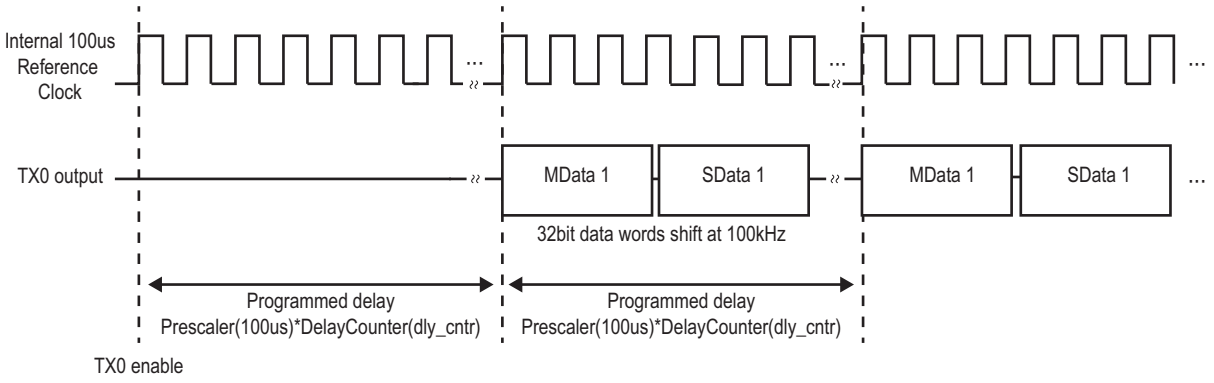


Figure 1-7 Example of Programming the Scheduler in Default Operation

The following settings describe control parameters in Default mode:

Master/Slave Control

Each of the 256 Scheduler control words is tagged as a NULL, Master, or Slave:

- **NULL** indicates no operation.
- **Master** indicates a master entry which determines scheduling, identifying the entry as periodic or one time and setting the transmittal time delay.
- **Slave** indicates entry is a slave to the previously defined master. In Default operation, slave entries use the same scheduling information of the master.

Once the transmitter is enabled, the Scheduler executes the first master entry and all following valid slave entries. Execution stops upon a new master entry or zero entry.

Periodic Control

Enabling Periodic Control causes the entry to execute on a periodic basis and disabling it causes the entry to execute once.

Delay Control

Delay control defines the time delay relative to enabling the transmission and between transmission of periodic master entries.

The time delay is based on a prescaler **timebase** multiplied by a 16-bit **delay counter**.

Timebase: Available DNx-429-516 prescalers include the following:

- DQ_AR_SCHED_PS100us – main 100us prescalers
- DQ_AR_SCHED_PSTB0 – first user-programmable timebase
- DQ_AR_SCHED_PSTB1 – second user-programmable timebase
- Set to zero to disable entry

Delay counter: The delay counter specifies the number of prescaler cycles to wait before executing the master entry. Slave entries directly follow master entries in the schedule and cannot be programmed with separate scheduling in Default operation.

Refer to “Programming the Output Scheduler” on page 32 for an example of programming the scheduler using the Framework.

Refer to “Writing Scheduler in Default Operation” on page 53 for a low-level programming example for **Figure 1-7** above.

1.9.2.3 Scheduler in Frame Clock Mode

In Frame Clock mode, the scheduler array is written with a series of user-defined master/slave control words and corresponding transmit messages (data). The master control word specifies the frame rate for groupings of the master/slave ARINC data word(s) to be shifted out of the transmitter.

The slave words in Frame Clock mode can be programmed to transmit on multiples of the frame clock.

Scheduler Table

Array Index	Scheduler Control Word	Scheduler Data Word (for TX)
0	NULL	NULL
1	Master=TRUE; Periodic=TRUE; Prescaler=PSTB1; DelayCounter=dly_ctr	Master Data Word: MData 1 (Label, SDI, Data, SSM, PARITY)
2	Master=FALSE; Periodic=TRUE; Prescaler=PSTB1; DelayCounter=2	Slave 1 Data Word: SData 1 (Label, SDI, Data, SSM, PARITY)
3	NULL	NULL
⋮	⋮	⋮
255	NULL	NULL

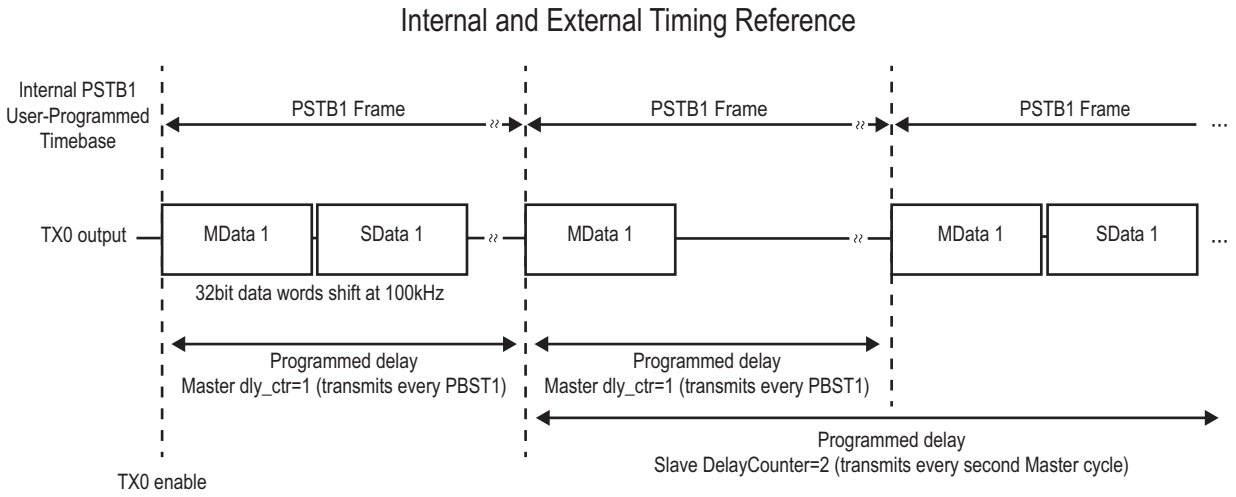


Figure 1-8 Example of Programming the Scheduler in Frame Clock Mode

The following settings describe control parameters for Frame Clock mode:

Master/Slave Control

Similar to the Default operation, Scheduler control words are tagged as follows:

- **NULL** indicates no operation.
- **Master** indicates a master entry which defines the frame rate timebase.
- **Slave** indicates entry is a slave to the previously defined master. Slave entries can be transmitted at user-programmable multiples of the frame rate.

NOTE: In Frame Clock mode, the first entry programmed in the scheduler array must be disabled (NULL). Every PSTB1-based master entry is followed by subsequent slave entries, and the last slave entry should be separated from the following master by a disabled entry. All unused scheduler entries should also be disabled.

Periodic Control

Enabling Periodic Control causes the entry to execute on a periodic basis and disabling it causes the entry to execute once.

Delay Control

The delay control settings define the timebase/frame rate for the master entries and sets the delay control for the slave entries.

Timebase: In Frame Clock mode, master entries must be programmed using the PSTB1 user-programmable timebase in order to operate as frames. Although the scheduler array can have master entries programmed with the 100 us or PSTB0 prescalers, those entries will be operating in Default operation and not frame mode, and they may interfere with frame timing.

When programming with the low-level API, the frame rate is set based on the PSTB1 timebase value multiplied by the 16-bit delay counter of the master.

Delay counter:

- For master entries, the delay counter is used to set the frame rate (PSTB1 prescaler value * delay_counter).
- For slave entries, the delay counter is used to schedule which frames the slave entry will transmit in.
  For example, for TD=0 or 1, the slave entry will transmit with every master frame; for TD=2, the slave entry will transmit every second master frame; for TD=n, the slave entry will transmit every nth master frame, etc.

Refer to “Writing Scheduler in Frame Clock Mode” on page 55 for a low-level programming example for **Figure 1-8** above.

1.9.2.4 Scheduler in Major/Minor Frame Mode

In Major/Minor Frame mode, a major frame clock triggers a sequence of minor frames, which are configured with user-programmable frame repeat rates and user-programmable time offsets relative to the start of a major frame. ARINC data is scheduled for transmission based on which minor frame the ARINC data is mapped to.

The scheduler arrays are written with up to 256 control words and 256 corresponding ARINC transmit data words.

Scheduler Control Array

The 256-word control array provides control information (enable/disable, periodic, and minor frame mapping) for corresponding ARINC data words. Each control word configures the corresponding data word to be transmitted based on the minor frames identified in the control word. There are 16 user-programmable minor frames that data can be mapped to. Minor frame execution reinitializes on every major frame clock boundary.

The major frame timebase is programmed via the user-programmable timebase, prescaler PSTB1. Minor frame output rates are programmed as multiples of the 100 us prescaler.

Scheduler ARINC Data Array Page 0 and Page 1

Data array paging is only available in Major/Minor Frame mode. The data arrays consist of a 256x32 Page0 array and a 256x32 Page1 array.

Paging allows one page to be filled with scheduled ARINC data, which will be transmitted based on minor frame scheduling, and allows new data to be written to the second page while the transmitter is shifting out the scheduled data from the first.

This buffering allows the user to write data words as needed and not overwrite data in the process of transmitting. Pages can be programmed to swap on a new major frame boundary or immediately after the scheduler has completed writing pending data.

Refer to **Figure 1-9** on page 23 for an example of major and minor frame scheduling. ARINC data shifts out of the transmitter based on minor frame timing relative to major frame boundaries.

Example Scheduler Table for Major/Minor Frame Mode

Array Index	Scheduler Control Word	Scheduler Data Word (Page 0)	Data Word (Page 1)
0	Periodic=TRUE; Minor Frame mapping=0x0001*	Pg0 Data Word 0	Pg1 Data Word 0
1	Periodic=TRUE; Minor Frame mapping=0x0001	Pg0 Data Word 1	Pg1 Data Word 1
2	Periodic=TRUE; Minor Frame mapping=0x0002	Pg0 Data Word 2	Pg1 Data Word 2
3	Periodic=TRUE; Minor Frame mapping=0x0003	Pg0 Data Word 3	Pg1 Data Word 3
4	NULL/Disable	NULL	NULL
⋮	⋮	⋮	⋮
255	NULL/Disable	NULL	NULL

*Minor Frame mapping for Scheduler control word is set as a bitmask:
0x0001 uses Minor Frame 0; 0x0002 uses Minor Frame 1; 0x0003 uses Minor Frame 1 and 0

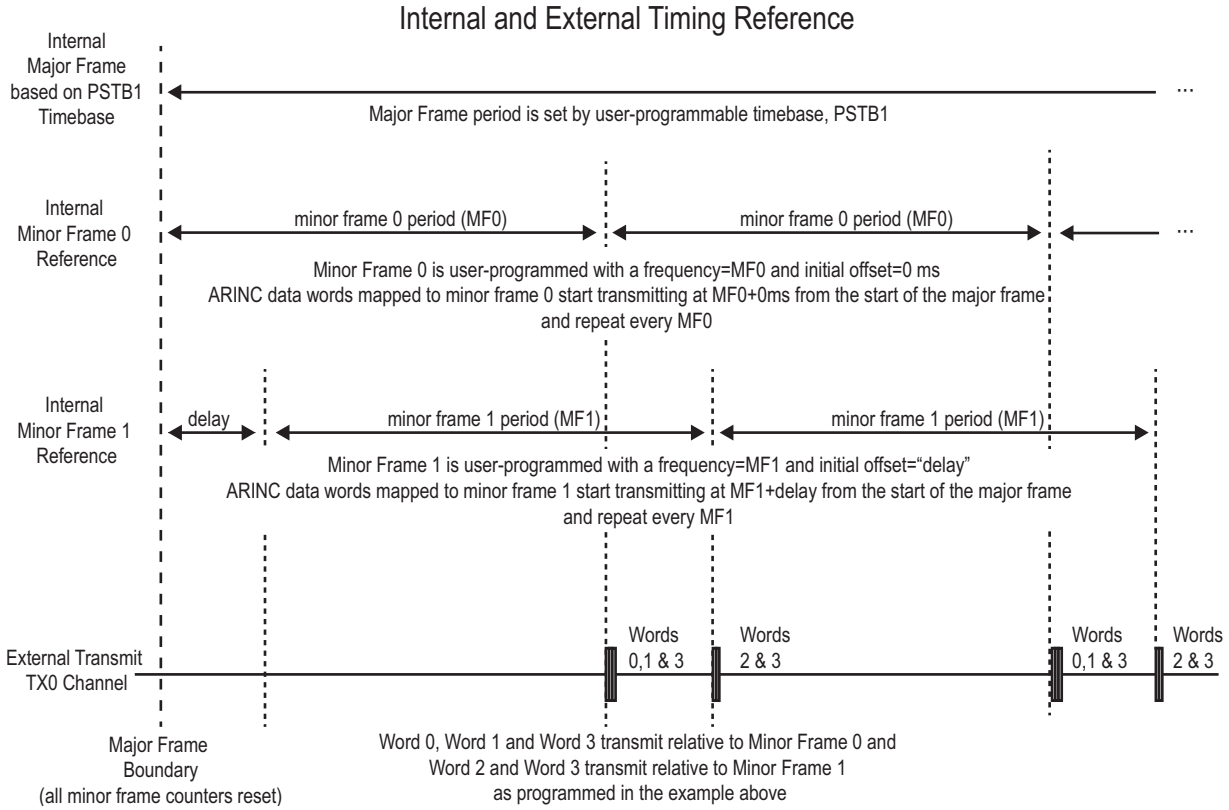


Figure 1-9 Example Scheduler Table in Major/Minor Frame Mode

The following settings describe control parameters for Major/Minor Frame mode:

Periodic Control

Enabling Periodic Control causes the entry to execute on a periodic basis and disabling it causes the entry to execute once.

Delay Control

Timebase: In Major/Minor Frame mode, minor frame entries are always programmed using the 100 us timebase (prescaler).

Delay counter:

- Minor frames can be programmed to start executing at a user-programmable time offset relative to the start of a major frame clock.
- Minor frames can be programmed to repeat at a user-programmable frequency.

Refer to “Scheduling Outputs Using Major/Minor Frames” on page 33 for an example of programming the major/minor frames using the Framework.

Refer to “Writing Scheduler in Major/Minor Frame Mode” on page 58 for a low-level programming example.

1.9.3 TX FIFO

Each DNx-429-516 transmit channel is built with a 256-word output FIFO.

The TX FIFO is used by the Scheduler and priority data registers as temporary storage. The TX FIFO should be enabled for all transmissions.

1.9.3.1 TX FIFO in Delayed Transmission Mode

The TX FIFO supports a delayed transmission mode where users can enable TX FIFO delay circuitry when setting up the transmit channel modes and configuration.

In this mode, the FIFO stores both `<delay control words>` and `<ARINC message output words>`.

The **delay control word** is 32-bits:

- bit[31:24] is the number of following ARINC messages that will have the delay applied to them
- bit[23:0] is the delay in microseconds (up to 254 ms)

Delay control is written to the FIFO first, followed by the ARINC messages. The following is an example sequence:

FIFO Index	FIFO Control/Data Word
0	**<0x2 Messages><delay1>** delay1 for 2 Messages in Block1
1	ARINC message1 in Block1
2	ARINC message2 in Block1
3	**<0x3 Messages><delay2>** delay2 for 3 Messages in Block2
4	ARINC message1 in Block2
5	ARINC message2 in Block2
6	ARINC message3 in Block2
7	0 to indicate an end to the sequence

Table 1-8 Writing TX FIFO in Delayed Transmission Mode

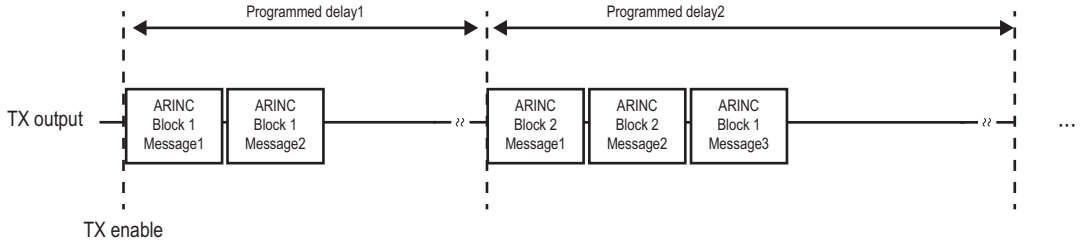


Figure 1-10 Example of Transmit Sequence for TX FIFO in Delayed Mode

1.10 Wiring & Connections (pinout)

The following signals are located at the DB-62 female connector on the DNx-429-516 board:

- TX/RX-*n*-A: Transmit/Receive line on bus A on the DNx-429-516
- TX/RX-*n*-B: Transmit/Receive line on bus B on the DNx-429-516
- RX-*n*-A: Receive-only line on bus A on the DNx-429-516
- RX-*n*-B: Receive-only line on bus B on the DNx-429-516
- GND CH $n_0/n_1/n_2$: Isolated ground for Channel n_0, Channel n_1, and Channel n_2.
  Channels are grouped into 8 isolated blocks consisting of three channels each, (i.e., Channels 0, 1, and 16 share an isolated block, Channels 2, 3, and 17 share an isolated block, etc.)
- rsvd: Reserved pins, do not connect

The pinout for the DNx-429-516 is provided below:

Pin	*Signal*	*Pin*	*Signal*	*Pin*	*Signal*
1	TX/RX-0-B	22	TX/RX-0-A	43	TX/RX-1-A
2	RX-16-B	23	RX-16-A	44	TX/RX-1-B
3	GND CH2/3/17	24	GND CH0/1/16	45	TX/RX-3-A
4	TX/RX-2-B	25	TX/RX-2-A	46	TX/RX-3-B
5	RX-17-B	26	RX-17-A	47	TX/RX-4-A
6	TX/RX-5-B	27	TX/RX-5-A	48	TX/RX-4-B
7	RX-18-B	28	RX-18-A	49	GND CH 4/5/18
8	TX/RX-6-B	29	TX/RX-6-A	50	GND CH 6/7/19
9	TX/RX-7-B	30	TX/RX-7-A	51	RX-19-A
10	TX/RX-8-B	31	TX/RX-8-A	52	RX-19-B
11	TX/RX-9-B	32	TX/RX-9-A	53	GND CH 8/9/20
12	RX-20-B	33	RX-20-A	54	GND CH 10/11/21
13	TX/RX-10-B	34	TX/RX-10-A	55	TX/RX-11-A
14	RX-21-B	35	RX-21-A	56	TX/RX-11-B
15	TX/RX-12-B	36	TX/RX-12-A	57	TX/RX-13-A
16	RX-22-B	37	RX-22-A	58	TX/RX-13-B
17	GND CH 14/15/23	38	GND CH 12/13/22	59	TX/RX-14-A
18	TX/RX-15-B	39	TX/RX-15-A	60	TX/RX-14-B
19	RX-23-B	40	RX-23-A	61	rsvd
20	rsvd	41	rsvd	62	rsvd
21	rsvd	42	rsvd		

Isolation boundaries .....

Figure 1-11 DNx-429-516 Pinout Diagram

NOTE: *Because the 429-516 board provides isolation between three-channel groupings, you must connect ground to each of the GND CH $n_0/n_1/n_2$ pins.
For example, GND CH0/1/16 is the only source of grounding for TX/RX channels 0, 1 and 16 (output pins: TX/RX-0-A, TX/RX-0-B, TX/RX-1-A, TX/RX-1-B, RX-16-A, RX-16-B).
Failure to do so will result in ungrounded ARINC channels.

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

1.10.1 ARINC-429 Bus in Multi-drop Network

Figure 1-12 illustrates an example of wiring the ARINC-429 bus in a multi-drop network configuration (which is the most common) with the DNx-429-5xx.

In the example configuration, the DNX-429-516 Channel 0 acts 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 the non-inverting signal (may be labeled as +)
- ***B*** is the inverting signal (may be labeled as -)

The signals from A/B lines are with reference to the ground. Even though the configuration in **Figure 1-12** is common, other configurations are also possible.

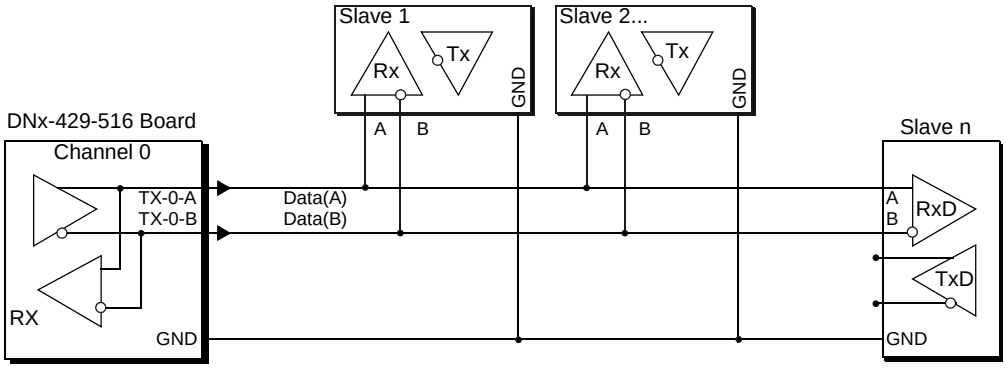


Figure 1-12 Wiring for an ARINC-429 Network

NOTE: A DNx-429-516 transmitter can connect to 20 receiver(s) on one twisted pair, which meets the ARINC-429 hardware specification. However, note that due to power restrictions, the DNx-429-516 board can only drive a maximum of 128 receivers per board.

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-429-516:

- 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)
- Programming the Output Scheduler (Section 2.7)
- Scheduling Outputs Using Major/Minor Frames (Section 2.8)
- Programming the Label Filter (Section 2.9)
- Cleaning-up the Session (Section 2.10)

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 no matter what programming language you use.

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

2.2 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 output

CUeiSession session;
```

2.3 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 Cube and RACKtangle, the device class is **pdna**.

ARINC boards 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 0 at IP address 192.168.100.2:

```
// 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);
```

`CreateARINCInputPort` parameters include the following:

- `bitsPerSecond`: port speed of 12.5K or 100K for a selected port.
- `parity`: boolean parity mode to use for controlling transmission integrity.
- `enableSDIFilter`: used to filter incoming words based on their SDI bits.
- `SDIMask`: if the result is other than zero, each incoming word is ANDed with this mask and the word is passed on to the software.

To program output channels on the DNx-429-516, you configure the resource string, baud rate, and parity:

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

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

2.4 Configuring the Timing

The application must configure the DNx-429-516 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 DNx-429-516 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);
```

2.5 Reading Data

Reading data from the DNx-429-516 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);
```

2.6 Writing Data

Writing data to the DNx-429-516 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.

Users can write ARINC messages for output to the TX FIFO or schedule output messages using the TX scheduler.

The following sample code shows how to create a writer object tied to port 2 and write one word to the TX FIFO for transmittal 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);
```

NOTE: The following section provides a description of how to use the scheduler for transmitting data.
For more information about the FIFO and scheduler hardware options, refer to Section 1.9.1 on page 15.

2.7 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 and/or software.

To program the output scheduler, you first add a pointer on the output channel that you wish to program:

```
// Configure ARINC output port 0

CUeiARINCOutputPort* pPort =
dynamic_cast<CUeiARINCOutputPort*>(session.GetChannel(0));
```

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.
   // Refer to the previous Section 2.6 for a description of Word structure
   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();
```

2.8 Scheduling Outputs Using Major/ Minor Frames

The hardware Scheduler on each output channel can be configured to transmit a table of messages where the periodic start of the transmission of messages is controlled by a major frame clock and the transmission of each individual TX message within that major clock period is controlled using minor frame clocks.

Users can configure to up to 16 minor clocks, which are programmed with user-programmable time offsets and user-programmable frame repeat rates relative to the start of a major frame. The major frame clock resets the minor clocks.

The scheduler table for each TX output can contain up to 256 TX messages. ARINC data is scheduled for transmission based on which minor frame(s) it is mapped to; the same data can be mapped to multiple minor frames.

2.8.1 Set MJ/MN Mode

To program the output scheduler in Major/Minor (MJ/MN) mode, you first add a pointer on the output channel that you wish to program:

```
// Configure ARINC output port 0

CUeiARINCOutputPort* pOutPort =
dynamic_cast<CUeiARINCOutputPort*>(session.GetChannel(0));
```

You next enable the scheduler and program the scheduler mode to Major/Minor mode:

```
// Enable and set MJ/MN mode

pOutPort->EnableScheduler(true);
pOutPort->SetSchedulerType(UeiARINCSchedulerTypeMajorMinorFrame);
```

2.8.2 Configure MJ/MN Frame Clock Rates

Configure the major frame clock rate, which is programmed in Hz.

This triggers the sequence of data transmissions (controlled by minor frames) and controls the rate at which they're periodically repeated.

```
// Configure major frame rate (i.e. 100.0 Hz)

double majorFrameRate = 100.0;
pOutPort->SetSchedulerRate(majorFrameRate);
```

Next configure minor frames, which will determine your schedule for transmitting individual ARINC words relative to the major frame.

The following example configures 2 minor frames as multiples of the major frame rate.

NOTE: The minor frame delay and period are programmed in 1 us increments.

```
// Configure minor frames

tUeiARINCMinorFrameEntry minorFrame;


//Program Frame 1: Period is programmed in divisions of major frame (10ms)
//   (minor frame 1 period is 1/2 the major frame = 5000us)

minorFrame.Delay = 0; // no delay
minorFrame.Period = uInt32((1000000.0 / majorFrameRate) / 2);

pOutPort->AddMinorFrameEntry(minorFrame);


//Program Frame 2:
//   (minor frame 2 period is 1/3 the major frame = 3333us)

minorFrame.Delay = 0; // no delay
minorFrame.Period = uInt32((1000000.0 / majorFrameRate) / 3);

pOutPort->AddMinorFrameEntry(minorFrame);
```

2.8.3 Add Scheduler Entries

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

```
typedef struct _tUeiARINCSchedulerEntry
{
   // this Master parameter is not used in MJ/MN mode
   Int32 Master;

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

   // this Delay parameter is not used in MJ/MN mode
   uInt32 Delay;

   // Word to be sent when this entry is processed.
   // Refer to Section 2.6 for a description of the tUeiARINCWord structure
   tUeiARINCWord Word;

   // Minor frame mapping (one bit per minor frame)
   uInt32 MinorFrameMask;

} tUeiARINCSchedulerEntry;
```

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

```
// Example of adding one Scheduler entry

tUeiARINCSchedulerEntry schedEntry;

// entry is periodic
schedEntry.Periodic = 1;

// Program word to be emitted by this entry
schedEntry.Word.Label = 0x05;
schedEntry.Word.Sdi = 1;
schedEntry.Word.Ssm = 1;
schedEntry.Word.Data = 0x1234;

// Add word to one of the minor frames --
// 0x1 (bit 1) maps to 1st minor frame; 0x2 (bit 2) maps to 2nd minor frame,
// 0x3 (bits 1 and 2) map to 1st and 2nd minor frames, 0x4 maps to 3rd, etc
schedEntry.MinorFrameMask = (1); // will transmit at 5 ms, etc.

pOutPort->AddSchedulerEntry(schedEntry);
```

When you start the session, the scheduled entries will start transmitting.

2.8.4 Update Scheduler Table

As soon as the session starts, the output port starts sending the scheduled words in sequence. You can update the scheduled words 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);
```

2.8.5 Update Scheduler Using TX Pages

MJ/MN mode provides an alternative to updating scheduled words as they're transmitting. The scheduler in MJ/MN mode can have two copies of scheduler entries, which are accessed via two different page areas (page 0 and page 1). This allows users to update a scheduler table on one page while transmitting from the other without affecting the data being transmitted.

The `CUeiARINCWriter::SetTransmitPage()` method is used to program which page will be transmitted from and which page will be written to for all TX channels. Channels are accessed as a bitmask (bit 0 maps to TX0, bit1 to TX1, etc.), and pages are specified as 0 (for page 0) and 1 for page 1.

The following programs future writes to write to page 1, and TX ports will transmit data stored in page 0 to the ARINC bus:

```
// Set write page to 1 and TX page to 0 (default)

writer->>SetTransmitPage(true,    // true switches pages immediately
                         0xffff,  // writes to TX ch15..0 will write page 1
                         0x0000); // transmitted data for ch15..0 will come
                                  //   from page 0
```

Now you can write the another set of scheduled words with `CUeiARINCWriter::WriteScheduler()` to the page 1 set above as follows:

```
// Write location 0 in scheduler table of page 1 while TX is still
//    transmitting from page 0

tUeiARINCWord word;
word.Label = 5;     // Set the label
word.Data = 0xABAB; // Set the payload
word.Sdi = 2;
word.Ssm = 2;
writer->WriteScheduler(0, 1, &word, &numWordsWritten);
```

Once you are finished updating a new scheduler table on page 1, you can swap pages to transmit the data from the new scheduler table out TX ports to the ARINC bus.

At this point your new data will transmit, and you have the ability to write the original page with updated data:

```
// Set write page to 0 and TX page to 1

writer->>SetTransmitPage(false,   // swap at start of next major frame
                         0x0000,  // configures ch15..0 writes to page 0
                         0xffff); // configures ch15..0 transmit from page 1
```

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

pOutPort->ClearSchedulerEntries();
```

2.9 Programming the Label Filter

Each input channel is equipped with a label filter that lets you filter received input words and only keep words that are tagged with a specified label.

To use this feature, you first need to get a pointer on the ARINC input channel you wish to program:

```
// Get pointer to the input channel 1

CUeiARINCInputPort* pPort =
dynamic_cast<CUeiARINCInputPort*>(session.GetChannel(1));
```

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();
```

2.10 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 provides the following information about programming the DNx-429-516 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)
- Configuring Board & Channels (Section 3.4)
- Setting Modes of Operation (Section 3.5)
- Setting the Baud Rate (Section 3.6)
- Reading RX Data (Section 3.7)
- Writing TX Data Using the FIFO (Section 3.8)
  - Configuring for FIFO Transmission (Section 3.8.1)
  - Configuring for FIFO Transmission with Delays (Section 3.8.2)
- Writing TX Data Using the Scheduler (Section 3.9)
  - Initializing the Scheduler Table (Section 3.9.1)
  - Programming Scheduler Timebase Values (Section 3.9.2)
  - Writing Scheduler Transmit Data (Section 3.9.3)
- Programming RX Filters (Section 3.10)
  - Filtering Messages Based on Labels (Section 3.10.1)
  - Filtering Messages Based on Parity (Section 3.10.2)
  - Filtering Messages Based on SDI (Section 3.10.3)

3.1 About the Low-level API

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

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

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

- 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 429-516-specific functions.

NOTE: The DNx-429-516 uses many of the same API functions as UEI's other I/O boards that support the ARINC protocol, (i.e., DNx-429-512/566). Functions unique to the DNx-429-516 series begin with `DqAdv516`.

Table 3-1 Summary of Low-level API Functions for DNx-429-516

Function	Description
`DqAdv566BuildPacket`	Assembles an ARINC message out of individual fields for transmission
`DqAdv566ParsePacket`	Splits a received packet into individual fields (Rx)
`DqAdv566BuildFilterEntry`	Assembles filter entries from the separate fields (Rx)
`DqAdv566BuildSchedEntry`	Assembles the control message for a scheduler entry from the separate fields (Tx)
`DqAdv516BuildFrameEntry`	Assembles the control message for a frame entry from the separate fields. Used when the Scheduler is in Major/Minor Frame mode (Tx)
`DqAdv566SetConfig`	Configures basic board-level ARINC device parameters
`DqAdv566SetMode`	Configures the mode of operation for each channel
`DqAdv566SetFilter`	Writes to or reads from the label filter (Rx)
`DqAdv566SetScheduler`	Writes to or reads from scheduler table (control message array and data message array) (Tx)
`DqAdv566SetSchedTimebase`	Sets one of two programmable scheduler timebases (Tx)
`DqAdv516MajorFrameDelay`	Sets the delay and clock divider for minor frames. Used when the Scheduler is in Major/Minor Frame mode (Tx)
`DqAdv566SetFifoRate`	Sets up a timebase for writing output packets to the Tx FIFO
`DqAdv566SetChannelCfg`	Sets operating configurations for each channel, (e.g., enables Tx channel or Rx loopback, enables Scheduler modes, enables FIFOs, etc.)
`DqAdv516SetTxPage`	Sets which page the host will write new data messages to and which page is actively used for transmitting data. Used when the Scheduler is in Major/Minor Frame mode (Tx)
`DqAdv516EnableTransmitters`	Enables all transmit operations on the board (Tx)
`DqAdv516ChangeBaudRate`	Allows programmable baud rate per channel (10 kbaud..200 kbaud)
`DqAdv566Enable`	Enables and disables all operations on the board

Table 3-1 Summary of Low-level API Functions for DNx-429-516 (Cont.)

Function	Description
DqAdv566SendPacket	Sends a packet of data to a priority register of the specified channel (Tx)
DqAdv566SendFifo	Puts packets of data into the Tx FIFO
DqAdv566RecvPacket	Receives a packet of data from the Rx interface
DqAdv566RecvFifo	Retrieves messages (packets of data) 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

3.3 Low-level Programming Techniques

Application developers are encouraged to explore the existing source code examples when first programming the 429-516. 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*

Sample code for data acquisition modes have the name of the mode and the name of the I/O board(s) being programmed embedded in the sample name. For example, SampleVMap516 contains sample code for running a 429-516 using VMAP data acquisition mode. Note that immediate mode samples are named Sample<I/O board name>, (i.e., Sample516).

3.3.1 Data Transfer Modes

The 429-516 supports the following acquisition modes:

- Immediate (point-to-point): Designed to provide easy access to a single I/O board at a non-deterministic pace. Reads or writes one ARINC message per channel. Runs at a maximum of 100 Hz.
- RTVMAP: Designed for messaging applications or closed-loop (control) applications. Users set up a “map” of I/O boards and channels from which to read or write incoming or outgoing messages. A single API call (refresh) paced by the user application causes messages to be transferred directly between I/O board FIFOs and the user application.

API that describe how to implement data acquisition modes and additional mode descriptions are provided in the *PowerDNA API Reference Manual*.

3.4 Configuring Board & Channels

The 429-516 uses two low-level APIs for board and channel configuration:

- **`DqAdv566SetConfig()`** configures the 429-516 at the board level
- **`DqAdv566SetChannelCfg()`** configures at the channel level

3.4.1 Board Configuration

Use **`DqAdv566SetConfig()`** to configure the 429-516 at the board level:

```
DqAdv566SetConfig(
         int hd,         // Handle to IOM received from DqOpenIOM()
         int devn,       // Board device # inside the IOM chassis
         int ss,         // Subsystem (see NOTE below)
         int timeout_us, // Time to wait in case TX is not ready or
                         //   RX loopback message is not available
         uint32 config   // Program as NULL (DIO are not supported)
);
```

NOTE: For transmit ports (`ss=DQ_SS0OUT`), the `timeout_us` parameter applies to all 16 ARINC transmit channels, and for receive ports (`ss=DQ_SS0IN`), `timeout_us` applies to all 24 ARINC receivers.

3.4.2 Channel Configuration

Channels 0 through 15 can be used as ARINC transmit or receive ports. On power-up, transmitters are disabled, defaulting to channels configured as receivers. Channels 16 through 24 are ARINC receiver ports only.

Use **`DqAdv566SetChannelCfg()`** to configure the 429-516 individual channels for transmitter and/or receiver functionality as summarized in **Table 3-2**. Each 429-516 channel can be configured independently.

```
DqAdv566SetChannelCfg(
         int hd,           // Handle to IOM received from DqOpenIOM()
         int devn,         // Board device # inside the IOM chassis
         int ss,           // Subsystem (TX=DQ_SS0In, RX=DQ_SS0OUT)
         int chan,         // Channel number (ARINC port number)
         uint32 actions    // See actions listed in Table 3-2
);                         //   actions are ORed together to configure
```

The `DqAdv566SetChannelCfg()` function includes actions for enabling the transmitter and receiver. As a general rule, if using the Scheduler, timebases and the scheduler table should be pre-programmed before enabling the ARINC transmitter. Programming can be changed at any time during operation.

Table 3-2 Channel Configuration Actions

Configuration Action	Description
`DQ_AR_ENABLE_Tx`	Enable transmit operations (Tx)
`DQ_AR_ENABLE_Rx`	Enable receive operations (Rx)

Table 3-2 Channel Configuration Actions (Cont.)

Configuration Action	Description
DQ_AR_ENABLE_TxFIFO	Enable transmit FIFO (Tx) **NOTE:** The TX FIFO is used for temporary storage during Scheduler operations and should be enabled when using the Scheduler.
DQ_AR_ENABLE_RxFIFO	Enable receive FIFO (Rx)
DQ_AR_ENABLE_SCHEDULER	Enable scheduler (Tx) The Scheduler can operate in one of three modes: • Default Operation: Refer to "Scheduler in Default Operation" on page 18 • Frame Clock mode (DQ_AR_SCHED_FRAMECLK): Refer to "Scheduler in Frame Clock Mode" on page 20 • Major/Minor Frame mode (DQ_AR_SCHED_MJMN): Refer to "Scheduler in Major/Minor Frame Mode" on page 22
DQ_AR_SCHED_FRAMECLK	Set scheduler to Frame Clock mode (Tx) **NOTE:** DQ_AR_SCHED_FRAMECLK and DQ_AR_SCHED_MJMN are mutually exclusive parameters.
DQ_AR_SCHED_SLAVETD	Allow scheduling slave entries using the delay_counter field when in Frame Clock mode (Tx)
DQ_AR_SCHED_MJMN	Set scheduler to Major/Minor Frame mode (Tx) **NOTE:** DQ_AR_SCHED_FRAMECLK and DQ_AR_SCHED_MJMN are mutually exclusive parameters.
DQ_AR_ENABLE_LOOPBACK	Enable loopback validation of transmitted packets (Tx/Rx) When this bit is set, the TX word is compared with RX data received on the corresponding receiver, which results in 5 words in the RX FIFO: 1. Expected TX frame 2. Received RX frame (0xFFFFFFFF if the frame is missing) 3. 32-bit timestamp 4. 32-bit flags [31-16] missing frame counter; [15:0] mismatched frame counter 5. TX source (Scheduler, FIFO, Low Priority Register, or High Priority Register) [10] = 1 if missing frame/mismatched error was detected [9..8] = source of TX transmission (0=Scheduler; 1=TX FIFO; 2=High priority register; 3=Low priority register) [7..0] = Scheduler entry number if [9..8]=0
DQ_AR_ENABLE_FILTER	Enable receive filter operations (Rx) Configures Label Acceptance Filter, including ignore bad data feature. This bit does not need to be set for SDI or parity filtering.
DQ_AR_LOGIC_LOOPBACK	<Reserved> Enable internal logic tests (internal to HI-3282); not used on DNx-429-516

3.5 Setting Modes of Operation

The 429-516 supported modes of operation are summarized in **Table 3-3**.

To set the mode of a channel, use the low-level function, `DqAdv566SetMode():`

```
DqAdv566SetMode(
         int hd,     // Handle to IOM received from DqOpenIOM()
         int devn,   // Board device # inside the IOM chassis
         int ss,     // Subsystem (TX=DQ_SS0In, RX=DQ_SS0OUT)
         int chnl,   // Channel number (0-23)
         uint32 cmd);// See NOTE below
```

NOTE: The `cmd` parameter is created by logically combining the #define parameters wanted for each mode of operation (see **Table 3-3** for parameters).
For example, to set the transmit baud rate as 100 kHz and set odd parity, `cmd` is set as follows:
`cmd = DQ_AR_RATEHIGH | DQ_AR_PARITYODD;`

Table 3-3 DNx-429-516 Modes of Operation

Mode of Operation	Description and #define Parameters
Rate control* (Tx)	Sets baud rate of transmitter. • `DQ_AR_RATEHIGH` – set I/O rate to 100 kbaud • `DQ_AR_RATELOW` – set I/O rate to 12.5 kbaud Note that you can alternatively program baud rates between 10 kbaud and 200 kbaud by using the `DqAdv516ChangeBaudRate` API
Parity control (Rx/Tx)	Sets parity control. • `DQ_AR_PARITYODD` – odd parity • `DQ_AR_PARITYEVEN` – even parity • `DQ_AR_PARITYOFF` – no parity check/generation in hardware/parity error injection
Tx Slow slew rate enable* (Tx only)	Enables slow slew rate option • `DQ_AR_SLOWSLEW_ENABLED` – enable slow slew rate • `DQ_AR_SLOWSLEW_DISABLED` – disable slow slew rate
Control zero label and FIFO priority behavior (Tx only)	Enables special case options. By default, zero labels are disabled and the Scheduler priority is higher than the FIFO (refer to **Table 1-5** in **Chapter 1**) • `DQ_AR_ALLOW_ZERO_LBL` – allow scheduler to output zero labels • `DQ_AR_ALLOW_FIFO_HIGH` – set FIFO priority higher than scheduler

*Note: 100 kbaud transmission rates require that slow slew rate is disabled, `DQ_AR_SLOWSLEW_DISABLED.`

Table 3-3 DNx-429-516 Modes of Operation

Mode of Operation	Description and #define Parameters
Rx Timestamp enabled (Rx only)	Enables timestamp. • DQ_AR_TIMESTAMP_ENABLED – write data packet into the receive FIFO along with the timestamp • DQ_AR_TIMESTAMP_DISABLED – no timestamp When timestamp is enabled, each received message generates two entries in the FIFO – one is the actual message, and the second is a timestamp count. The timestamp count is cleared when the board is enabled for operation. The timestamp count increments every 10us by default; however DqCmdResetTimestamp() provides an option to set a different timestamp increment period.
SDI filtering (Rx only)	Enables filtering on the SDI field of the received loopback message. • DQ_AR_SDI_ENABLED – SDI filtering is enabled • DQ_AR_SDI_DISABLED – SDI filtering is disabled Note when SDI filtering is enabled, DQ_AR_SDIMASK0 and DQ_AR_SDIMASK1 bits below determine the SDI value you are filtering for. For example, if you want to accept all ARINC messages with an SDI value of 1, you would OR the following into the cmd: cmd |= DQ_AR_SDI_ENABLED | DQ_AR_SDIMASK0;
Enable SDI mask (Rx only)	Sets masking of the SDI field of the received message. • DQ_AR_SDIMASK0 – bit 0 • DQ_AR_SDIMASK1 – bit 1 Note only used when mode is DQ_AR_SDI_ENABLED.
Rx FIFO control (Rx only)	Enables RX FIFO control options of the received message. • DQ_AR_IGNORE_BAD_DATA - if received ARINC word has a parity error, rejects it and does not write it into the RX FIFO • DQ_AR_LB_CHECK_PARITY - include parity check into loopback comparison • DQ_AR_ADD_TIMESTAMP - add timestamp into FIFO data frame counter mode (Rx only) • DQ_FRCNT_COUNT_ALL - count all frames • DQ_FRCNT_COUNT_GOOD - only correctly received frames • DQ_FRCNT_COUNT_FIFO - only placed into the FIFO • DQ_FRCNT_COUNT_TRIGGER - count frames that triggered scheduler • DQ_FRCNT_COUNT_PAR_ERR - count frames with parity error
Tx FIFO delay (Tx only)	Enables TX FIFO delay control option. • DQ_AR_TXFIFO_DELAY Allows a control word containing a delay setting to be programmed as the leading word in the FIFO and subsequent ARINC TX messages will be delayed by the programmed amount. See example in Section 3.8.2.

3.6 Setting the Baud Rate

Users can set the baud rate on a per-channel basis.

The 429-516 supports the high speed (100 kHz) and low speed (12.5 kHz) specification of the ARINC 429 protocol.

These baud rates can be programmed by ORing the DQ_AR_RATEHIGH or DQ_AR_RATELOW flags using the DqAdv566SetMode() API when configuring the mode for each channel. Refer to Section 3.5 for more information about programming channel modes.

The 429-516 can alternatively be programmed to baud rates other than the 12.5 kHz and 100 kHz ARINC standard using the DqAdv516ChangeBaudRate() API.

The following example updates the baud rate on TX port 0 and RX port 0 to NEW_BAUDRATE:

```
uint32 chmask =0;
chmask = 0x1 | DQ_AR_CHANGE_TX | DQ_AR_CHANGE_RX;

DqAdv516ChangeBaudRate(hd, DEVN, chmask, NEW_BAUDRATE);
```

where

- chmask is a bitwise mask indicating which RX and/or TX channels will use the new baudrate.
  OR DQ_AR_CHANGE_TX into chmask to change rate on transmitter
  OR DQ_AR_CHANGE_RX into chmask to change rate on receiver
- NEW_BAUDRATE is the baud rate from 10 kHz to 200 kHz

3.7 Reading RX Data

To program the 429-516 to receive ARINC words via an RX port and store them in a RX input FIFO, you will need to do the following:

- Configure the board (see Section 3.4.1)
- Configure the mode (per ARINC RX port)
- Configure the port (per ARINC RX port)
- Enable the board
- Read received message

NOTE: If transmitters (TX ports) are enabled and transmitting, you can also use the following steps to read looped back TX messages on the corresponding RX ports. This diagnostic feature can be helpful when debugging your application.

Configure the mode:

The following API configures RX port 0 to run at a 100 kbaud rate, disables SDI filtering, configures RX logic to expect odd parity, and configures receivers to not expect a slow slew rate. Note that `mode_rx` is defined as uint32.

```
mode_rx = DQ_AR_RATEHIGH       |
          DQ_AR_SDI_DISABLED   |
          DQ_AR_PARITYODD      |
          DQ_AR_SLOWSLEW_DISABLED;

DqAdv566SetMode(hd, DEVN, DQ_SS0IN, 0, mode_rx);
```

Refer to Section 3.5 for additional mode options.

Configure the port:

The following configures RX port 0 to be enabled. `chcfg_rx` is uint32.

```
chcfg_rx = DQ_AR_ENABLE_Rx | DQ_AR_ENABLE_RxFIFO;

DqAdv566SetChannelCfg(hd, DEVN, DQ_SS0OUT, 0, chcfg_tx);
```

Refer to Section 3.4 for additional configuration options.

Enable the configured channels:.

```
DqAdv566Enable(hd, DEVN, TRUE);
```

NOTE: On power-up, channel 0..15 transmit drivers are off; RX configured channels can be used as receivers and receive RX messages from external drivers. To use channel 0..15 as transmitters, you must enable the transmitters (`DQ_AR_ENABLE_Tx`).
Refer to Section 3.8 or Section 3.9 for more information about configuring a port as an ARINC transmitter.

Read received message:

The `DqAdv566RecvFifo()` reads the ARINC RX messages that have been stored in the RX FIFO.

```
// The following API requests 100 words from the RX port 0 FIFO
// the API reports the number of retrieved (copied) words
//     and how many received words remain in the FIFO

DqAdv566RecvFifo(hd, DEVN, 0, 100, datarx, &copied, &remains);

// The following API parses the received ARINC word (datarx[0]) and
// returns payload data, label, sdi, ssm, and parity values for datarx[0]

DqAdv566ParsePacket(datarx[0],&data, &label, &sdi, &ssm, &parity);
```

3.8 Writing TX Data Using the FIFO

Users can store outgoing ARINC messages for transmittal by writing them to the port's TX FIFO or by using the output Scheduler.

Section 3.8.1 describes configuring writes using the TX FIFO, and Section 3.8.2 describes configuring writes with programmable delays using the TX FIFO.

See Section 3.9 for information about programming the Scheduler.

3.8.1 Configuring for FIFO Transmission

To program the 429-516 to store ARINC words in an output FIFO and transmit them from the TX ports, you will need to do the following:

- Configure the board (see Section 3.4.1)
- Configure the mode (per ARINC TX port)
- Set the TX FIFO timebase (per ARINC TX port)
- Configure the port (per ARINC TX port)
- Enable the configured channels
- Write TX message to the FIFO (for transmittal)

Configure the mode:

The following configures TX port 0 to run at a 100 kbaud rate, disable SDI filtering, use odd parity, and not use a slow slew rate. `mode_tx` is uint32.

```
mode_tx = DQ_AR_RATEHIGH|
          DQ_AR_SDI_DISABLED|
          DQ_AR_PARITYODD|
          DQ_AR_SLOWSLEW_DISABLED;

DqAdv566SetMode(hd, DEVN, DQ_SS0OUT, 0, mode_tx);
```

Refer to Section 3.5 for additional mode options.

Set the TX FIFO timebase:

The following configures FIFO transmissions from channel 0 to be paced by the TX ready signal (when TX is done outputting one message, the next is pulled from the FIFO for transmittal):

```
DqAdv566SetFifoRate(hd,DEVN, DQ_SS0OUT, 0, DQ_AR_FIFO_PSTB0,1000);
```

Note that the `rate_us` (set to 1000) is not used when configured as `DQ_AR_FIFO_PSTB0` (paced by TX ready).

Configure the port:

The following enables TX port 0 and enables its TX FIFO. `chcfg_tx` is uint32.

```
chcfg_tx = DQ_AR_ENABLE_Tx | DQ_AR_ENABLE_TxFIFO;

DqAdv566SetChannelCfg(hd, DEVN, DQ_SS0OUT, 0, chcfg_tx);
```

Refer to Section 3.4 for additional configuration options.

Enable the configured channels:.

```
DqAdv566Enable(hd, DEVN, TRUE);
```

Write TX message to the FIFO:

The `DqAdv566BuildPacket()` builds an ARINC word (where d is a uint32 data value, 0x4 is the label, 0 is the SDI, and 0 is the SSM.) When a TX port is configured as Even or Odd, calculated parity is auto-inserted by the channel logic.

The `DqAdv566SendFifo()` writes to the port 0 FIFO: 1 ARINC data word built with `DqAdv566BuildPacket()`. The API returns the number of accepted words and how much space is still available in the FIFO.

```
datatx[0] = DqAdv566BuildPacket(d, 0x4, 0, 0, 0);

DqAdv566SendFifo(hd, DEVN, 0, 1, datatx, &accepted, &available);
```

NOTE: `datatx` is a uint32 array; `accepted` and `available` are also uint32s.

3.8.2 Configuring for FIFO Transmission with Delays

To program the 429-516 to transmit ARINC words with delays from the output FIFO, you will need to configure the transmit channel in the TX-FIFO-delay mode and additionally write delay information to the FIFO (along with the ARINC TX messages).

- Configure the board (see Section 3.4.1)
- Configure the mode (per ARINC TX port)
- Set the TX FIFO timebase (per ARINC TX port) (see Section 3.8.1)
- Configure the port (per ARINC TX port) (see Section 3.8.1)
- Enable the configured channels (see Section 3.8.1)
- Write TX message to the FIFO (for transmittal)

Configure the mode:

FIFO delay mode is configured by ORing in `DQ_AR_TXFIFO_DELAY`. The rest of the mode configuration is the same as Section 3.8.1.

```
mode_tx = DQ_AR_RATEHIGH  | DQ_AR_SDI_DISABLED |
          DQ_AR_PARITYODD |
          DQ_AR_SLOWSLEW_DISABLED | DQ_AR_TXFIFO_DELAY;

DqAdv566SetMode(hd, DEVN, DQ_SS0OUT, 0, mode_tx);
```

Refer to Section 3.5 for additional mode options.

Write TX messages with programmed delay to the FIFO:

Writing TX messages with a programmed delay consists of first writing a control word that provides the delay and number of ARINC output words that will use that delay and then writing the ARINC output messages.

The delay control word is in the following format:

- bit[31:24] is number of ARINC words that will use the delay
- bit[23:0] is the delay (in 1 µs counts up to 254 ms)

The following example writes to the FIFO on TX port 0 and configures port 0 to output 2 ARINC TX messages using a delay of 5 ms:

```
// configure to send 2 ARINC words with a 5 ms delay until the next set
datatx[0] = (2<<24)|(1000*5);
// ARINC message #1: data is 0xD1, label is 4, SDI is 0, SSM is 0
datatx[1] = DqAdv566BuildPacket(0xD1, 0x4, 0, 0, 0);
// ARINC message #2: data is 0xD2, label is 4, SDI is 0, SSM is 0
datatx[2] = DqAdv566BuildPacket(0xD2, 0x4, 0, 0, 0);

DqAdv566SendFifo(hd, DEVN, 0, 3, datatx, &accepted, &available);
```

Then write then next set of messages in the FIFO on TX port 0 and configure port 0 to output 3 ARINC TX using a delay of 8 ms:

```
// configure to send 3 more ARINC words with an 8 ms delay until the next
datatx[3] = (3<<24)|(1000*8);
// ARINC message #3: data is 0xD3, label is 4, SDI is 0, SSM is 0
datatx[4] = DqAdv566BuildPacket(0xD3, 0x4, 0, 0, 0);
// ARINC message #4 & #5
datatx[5] = DqAdv566BuildPacket(0xD4, 0x4, 0, 0, 0);
datatx[6] = DqAdv566BuildPacket(0xD5, 0x4, 0, 0, 0);

DqAdv566SendFifo(hd, DEVN, 0, 4, datatx, &accepted, &available);
```

The `DqAdv566SendFifo()` writes the 7 `datatx` words to the port 0 FIFO. The API returns the number of accepted words and how much space is still available in the FIFO.

The following is example of the transmission on port 0:

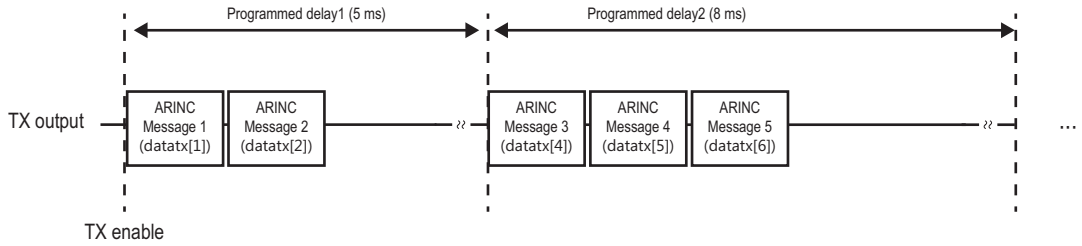


Figure 3-1 Example of Transmit from TX FIFO in Delayed Mode

NOTE: The next delay entry in the FIFO is processed when a previous delay expires. If the next delay is less than the time it takes to output the current data word(s), the next sequence of transmission word(s) will start immediately. An ARINC word transmitted at 100 kbaud takes 360 µs and at 12.5 kbaud takes 2.88 ms takes to output.

3.9 Writing TX Data Using the Scheduler

When using the Scheduler, Scheduler timing and control should be configured, and the Scheduler data array should be initialized and written with transmit messages before enabling the channel.

3.9.1 Initializing the Scheduler Table

After setting up channel configurations, the scheduler table (the 256-word control array and 256-word ARINC data array) should be initialized with NULL data.

`DqAdv566SetScheduler()` controls writing to and reading from the Scheduler arrays. The following example initializes both arrays on channel 0 to NULL:

```
// create an array of 1 location, 0 value
uint32 null_array[1] = {0};
// initialize scheduler table from 0 to 255 with 0s
DqAdv566SetScheduler(hd0, DEVN, 0, DQ_AR_SETSCHED_FILL_TABLE, 0, 255,
null_array, null_array);
```

3.9.2 Programming Scheduler Timebase Values

The Scheduler provides two user-programmable timebases (prescalers) and a 100 µs fixed timebase, which allow for the creation of schedules driven from independent clocks:

#define Constant	Description
DQ_AR_SCHED_PSTB0	Uses timebase 0 (programmed in us)
DQ_AR_SCHED_PSTB1	Uses timebase 1 (programmed in us)
DQ_AR_SCHED_PS100us	Uses a 100 us timebase

`DqAdv566SetSchedTimebase()` controls the programming of PSTB0 or PSTB1 timebases. The following example sets timebase 0 (PSTB0) to 1 ms for channel 0:

```
// set channel 0: prescaler-timebase PSTB0 and set to 1000 us
DqAdv566SetSchedTimebase(hd0, DEVN, 0, DQ_AR_SCHED_PSTB0, 1000);
```

3.9.3 Writing Scheduler Transmit Data

The Scheduler supports three modes of operation:

- Default Operation
- Frame Clock
- Major/Minor Frame

Each mode provides a different method of transmit word scheduling and requires mode-specific programming.

3.9.3.1 Writing Scheduler in Default Operation

The scheduler table is composed of two 256-word arrays: one array contains timing and control information and the other array contains the corresponding transmit messages to be output.

Refer to “Scheduler in Default Operation” on page 18 for more information.

The following example writes index 0 and 1 of the scheduler table. Refer to **Figure 3-2** for reference timing.

Build Scheduler table:

```
// software array to hold control words

uint32 sched_cmd[256] = {0};

// software array to hold control words

uint32 sched_data[256] = {0};
```

Build scheduler entries:

```
// DelayCounter=50; Delay will be 5 ms (50ticks * 100us)

int j=0;
int dly_cntr = 50;

// Build a Master Entry (control word and data)
//   control word: master=TRUE, periodic=TRUE, timebase is fixed 100us
//                 prescaler, programmable delay is dly_cntr (50ms)

sched_cmd[j] =
   DqAdv566BuildSchedEntry(TRUE, TRUE, DQ_AR_SCHED_PS100us, dly_cntr);

//   message: ARINC data=0x1234, Label=5, SSM=1, SDI=1

sched_data[j++] =
   DqAdv566BuildPacket(0x1234, 5, 1, 1, 0);

// Build a Slave Entry to previous master (control word and data)
//   control word: master=FALSE, periodic=TRUE (must be same as master),
//                 timebase is fixed 100us (must be same as master),
//                 programmable delay (must be same as master)

sched_cmd[j] =
   DqAdv566BuildSchedEntry(FALSE, TRUE, DQ_AR_SCHED_PS100us, dly_cntr);

//   message: ARINC data=0xFFFF, Label=5, SSM=1, SDI=1

sched_data[j++] =
   DqAdv566BuildPacket(0xFFFF, 5, 1, 1, 0);
```

NOTE: In Default Operation, slaves always directly follow masters.

Write scheduler table to hardware:

```
// Write the Master and Slave entry to the hardware Scheduler Table
//   writes (puts) index 0 through j into scheduler table in hardware
//   with sched_cmd and sched_data arrays built above

DqAdv566SetScheduler(hd0, DEVN, 0, DQ_AR_SETSCHED_PUT,
                     0, j, sched_cmd, sched_data);
```

Configure the port for Scheduler default operation:

The following configures TX port 0 with an output baud rate of 100 kHz, even parity, and slow slew rate and enables the default scheduler.

```
chcfg_tx =  DQ_AR_RATEHIGH          | DQ_AR_PARITYEVEN |
            DQ_AR_SLOWSLEW_ENABLED | DQ_AR_ENABLE_SCHEDULER ;

DqAdv566SetChannelCfg(hd, DEVN, DQ_SS0OUT, 0, chcfg_tx);
```

Scheduler Table

Array Index	Scheduler Control Word	Scheduler Data Word (for TX)
0	Master=TRUE; Periodic=TRUE; Prescaler=100us; DelayCounter=dly_ctr	Master Data Word: MData 1
1	Master=FALSE; Periodic=TRUE; Prescaler=100us; DelayCounter=dly_cntr	Slave 1 Data Word: SData 1
2	NULL	NULL
⋮	⋮	⋮
255	NULL	NULL

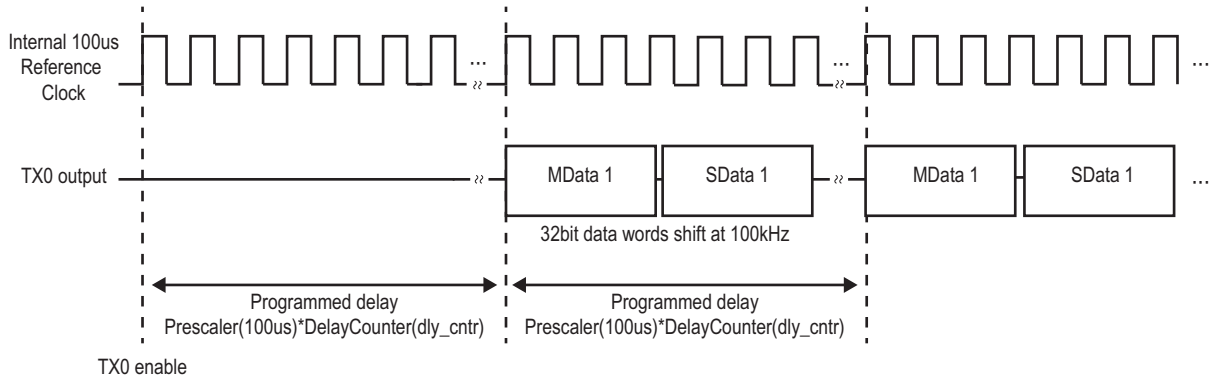


Figure 3-2 Example of Programming the Scheduler in Default Operation

3.9.3.2 Writing Scheduler in Frame Clock Mode

The following example sets up the scheduler in Frame Clock mode.

It configures user-programmable timebase 1 (PSTB1) and then configures and writes the Scheduler Table. Refer to **Figure 3-3** for timing references.

Note that scheduler timing in Frame Clock mode requires using user-programmable prescaler 1 (PSTB1) as the timebase.

Refer to “Scheduler in Frame Clock Mode” on page 20 for more information.

Initialize arrays for writing Scheduler table:

```
// software array to hold control words

uint32 sched_cmd[256] = {0};

// software array to hold control words

uint32 sched_data[256] = {0};
```

Set timebase for Frame mode:

```
// DelayCounter=1; Delay will be 10 ms (1 frame * 10ms) for PSTB1

int j=0;
int dly_cntr = 1;

// For channel 0: Program Prescaler 1 timebase
//                  to repeat every 10 ms (10000 us)

DqAdv566SetSchedTimebase(hd0, DEVN, 0, DQ_AR_SCHED_PSTB1, 10000);
```

Build required null entry in first index:

```
// Build the first entry as NULL - required in Frame Clock mode
// Build a NULL Control Word (DQ_AR_SCHED_PSDISABLED)

sched_cmd[j] =
   DqAdv566BuildSchedEntry(FALSE, TRUE, DQ_AR_SCHED_PSDISABLED, 0);

// Build a NULL Data Word

sched_data[j++] =
   DqAdv566BuildPacket(0x0, 0, 0, 0, 0);
```

Build master and slave entries:

```
// Build a Master Entry (control word and data)
//   control word: master=TRUE (Master word defines timing),
//                 periodic=TRUE,
//                 Frame clock timebase is PSTB1 prescaler,
//                 delay is 1 PSTB1 frame (transmits every 10 ms)

sched_cmd[j] =
   DqAdv566BuildSchedEntry(TRUE, TRUE, DQ_AR_SCHED_PSTB1, dly_cntr);

//   message #1 (master): ARINC data=0x1234, Label=5, SSM=1, SDI=1

sched_data[j++] =
   DqAdv566BuildPacket(0x1234, 5, 1, 1, 0);

// Build a Slave Entry to previous master (control word and data)
//   control word: master=FALSE, periodic=TRUE (must be same as master),
//                 timebase is PSTB1 (must be same as master),
//                 dly_cntr field in Frame clock mode defines
//                   number of master frames in which slave will transmit
//                   a 2 means every 2nd frame

sched_cmd[j] =
   DqAdv566BuildSchedEntry(FALSE, TRUE, DQ_AR_SCHED_PSTB1, 2);

//   message #2 (slave): ARINC data=0xABCD, Label=5, SSM=1, SDI=1

sched_data[j++] =
   DqAdv566BuildPacket(0xABCD, 5, 1, 1, 0);
```

Build required null entry in before next master entry:

```
// Build NULL entry before masters - required in Frame Clock mode
// Build a NULL Control Word (DQ_AR_SCHED_PSDISABLED)

sched_cmd[j] =
   DqAdv566BuildSchedEntry(FALSE, TRUE, DQ_AR_SCHED_PSDISABLED, 0);

// Build a NULL Data Word

sched_data[j++] =
   DqAdv566BuildPacket(0x0, 0, 0, 0, 0);
```

Write scheduler table to hardware:

```
// Write the Master and Slave entry to the hardware Scheduler Table
//   writes (puts) index 0 through j into scheduler table in hardware
//   with sched_cmd and sched_data arrays built above

DqAdv566SetScheduler(hd0, DEVN, 0, DQ_AR_SETSCHED_PUT,
                     0, j, sched_cmd, sched_data);
```

Configure the port for Scheduler in Frame Clock mode:

The following API configures TX port 0 with an output baud rate of 100 kHz, even parity, slow slew rate, and enabled scheduler in Frame Clock mode.

```
chcfg_tx =  DQ_AR_RATEHIGH          | DQ_AR_PARITYEVEN |
            DQ_AR_SLOWSLEW_ENABLED |
            DQ_AR_ENABLE_SCHEDULER |
            DQ_AR_SCHED_FRAMECLK    | DQ_AR_SCHED_SLAVETD;

DqAdv566SetChannelCfg(hd, DEVN, DQ_SS0OUT, 0, chcfg_tx);
```

Scheduler Table

Array Index	Scheduler Control Word	Scheduler Data Word (for TX)
0	NULL	NULL
1	Master=TRUE; Periodic=TRUE; Prescaler=PSTB1; DelayCounter=dly_ctr	Master Data Word: MData 1 (Label, SDI, Data, SSM, PARITY)
2	Master=FALSE; Periodic=TRUE; Prescaler=PSTB1; DelayCounter=2	Slave 1 Data Word: SData 1 (Label, SDI, Data, SSM, PARITY)
3	NULL	NULL
⋮	⋮	⋮
255	NULL	NULL

Internal and External Timing Reference

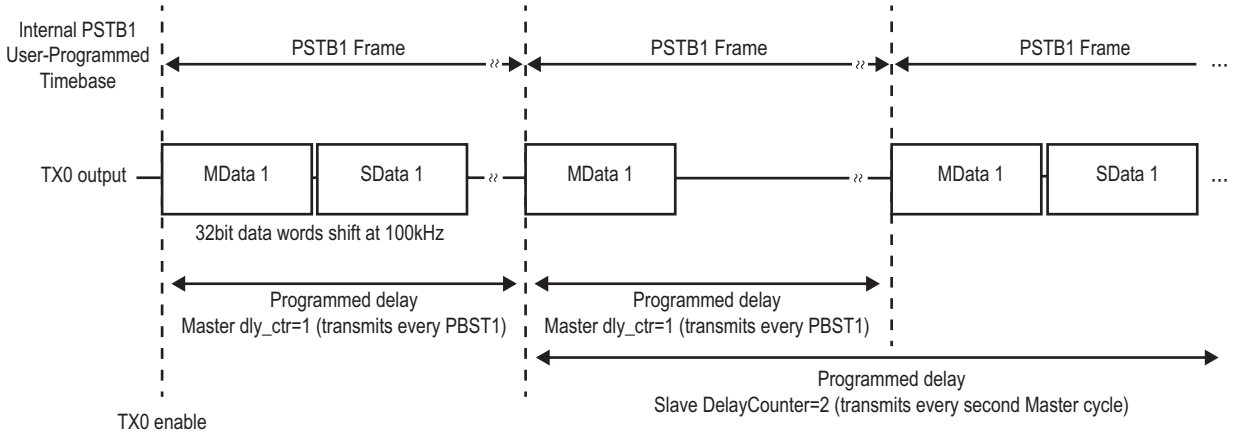


Figure 3-3 Example of Programming the Scheduler in Frame Clock Mode

3.9.3.3 Writing Scheduler in Major/Minor Frame Mode

The following example sets up the Scheduler to run in Major/Minor Frame mode. Initializing data array pages and initializing major and minor frames are programmed first.Then the Scheduler Table is written with control words and transmit messages.

Note that scheduler timing in Major/Minor Frame mode requires using user-programmable prescaler 1 (PSTB1) as the timebase for the major frame and the fixed 100 µs prescaler as the timebase for the minor frames.

Refer to “Scheduler in Major/Minor Frame Mode” on page 22 for a description of Major/Minor Frame Mode.

Initialize default pages for all frames:

Pages are only available in MJ/MN mode.

```
// program writes to occur immediately;
//   first 0 sets all channels to write to Scheduler page 0;
//   second 0 sets all channels to transmit data from Scheduler page 0

DqAdv516SetTxPage(hd0, DEVN, DQ_AR_SETTXPAGE_IMMEDIATE, 0, 0, NULL);
```

Set timebase for Major Frame clock in MJ/MN mode:

```
// For channel 0: Program Prescaler 1 timebase
//                 to run the major frame clock at 1 s (1000000 us)

DqAdv566SetSchedTimebase(hd0, DEVN, 0, DQ_AR_SCHED_PSTB1, 1000000);
```

Program two minor frames:

Set up two minor frames: both run periodically at 2 Hz, but the second is offset from the first by 100 ms.

```
// first minor frame runs at 2 Hz (500 ms between frames) with 0 offset

frmdiv[0] = (uint32)(10000*0.500); // program in 100 us increments
frmdelay[0] = (uint32)(10000*0.000);

// second minor frame also runs at 2 Hz (500 ms between frames)
//  with an offset (delay) of 100 ms to the frame boundary)

frmdiv[1] = (uint32)(10000*0.500); // program in 100 us increments
frmdelay[1] = (uint32)(10000*0.100);
```

```
// write first minor frame to hardware: set channel 0, Frame 0

DqAdv516SetMajorFrameDelay(hd0, DEVN, 0, 0, frmdelay[0], frmdiv[0]);

// write second minor frame to hardware: set channel 0, Frame 1

DqAdv516SetMajorFrameDelay(hd0, DEVN, 0, 1, frmdelay[1], frmdiv[1]);
```

Build first entry based on minor frame 0 timing:

```
// Build first entry (control word and data)
//   control word: periodic=TRUE,
//                 bitmask (0x0001) to assign minor frame 0 timing
// alternatively use macro DQ_AR516_FRM_MASK(0) to point to mn frame 0

sched_cmd[j] =
   DqAdv566BuildFrameEntry(TRUE, 0x0001);

//   message: ARINC data=0x1234, Label=5, SSM=1, SDI=1

sched_data[j++] =
   DqAdv566BuildPacket(0x1234, 5, 1, 1, 0);
```

Build second entry based on minor frame 0 timing:

```
// Build second entry (control word and data)
//   control word: periodic=TRUE,
//                 bitmask (0x0001) to assign minor frame 0 timing

sched_cmd[j] =
   DqAdv566BuildFrameEntry(TRUE, 0x0001);

//   message: ARINC data=0xABCD, Label=5, SSM=1, SDI=1

sched_data[j++] =
   DqAdv566BuildPacket(0xABCD, 5, 1, 1, 0);
```

Build third entry based on minor frame 1 timing:

```
// Build third entry (control word and data)
//   control word: periodic=TRUE,
//                 bitmask (0x0002) to assign minor frame 1 timing

sched_cmd[j] =
   DqAdv566BuildFrameEntry(TRUE, 0x0002);

//   message: ARINC data=0x6789, Label=5, SSM=1, SDI=1

sched_data[j++] =
   DqAdv566BuildPacket(0x6789, 5, 1, 1, 0);
```

Write scheduler table to hardware:

```
// Write the entries to the hardware Scheduler Table
//   writes (puts) index 0 through j into scheduler table in hardware
//   with sched_cmd and sched_data arrays built above

DqAdv566SetScheduler(hd0, DEVN, 0, DQ_AR_SETSCHED_PUT,
                     0, j, sched_cmd, sched_data);
```

Configure paging:

Subsequent writes to the scheduler table will be written to page 1 in hardware (pages are only available in MJ/MN mode)..

```
// program page swap to occur on the next major frame boundary;
//   1 sets channel 0 to write to page 1 (ch 1..15 still write to pg 0);
//   0 sets all channels to transmit data from page 0

DqAdv516SetTxPage(hd0, DEVN, DQ_AR_SETTXPAGE_ONMF, 1, 0, NULL);
```

Configure the port to run in MJ/MN mode:

The following API configures TX port 0 with an output baud rate of 100 kHz, even parity, slow slew rate, and enabled scheduler in Major/Minor Frame mode.

```
chcfg_tx =  DQ_AR_RATEHIGH          | DQ_AR_PARITYEVEN |
            DQ_AR_SLOWSLEW_ENABLED |
            DQ_AR_ENABLE_SCHEDULER |
            DQ_AR_SCHED_MJMN;

DqAdv566SetChannelCfg(hd, DEVN, DQ_SS0OUT, 0, chcfg_tx);
```

Refer to “Scheduler in Major/Minor Frame Mode” on page 22 for an example timing diagram.

3.10 Programming RX Filters

The DNx-429-516 has the capability of accepting or rejecting incoming ARINC data frames on an individual port using any of the following features:

- accept ARINC RX messages based on user-defined labels and optionally accept only new data on those labels
- accept or reject ARINC RX messages with parity errors
- accept or reject ARINC RX messages based on SDI field

3.10.1 Filtering Messages Based on Labels

To use the label acceptance filter, you'll configure the RX port as described in "Reading RX Data" on page 47 and additionally do the following:

Initialize the filter table hardware with zeros:

```
// Initialize channel 0 filter table
//   writes 255 labels starting at index 0

uint32 zero_label = 0;

DqAdv566SetFilter(hd, DEVN, 0, DQ_AR_SETFILTER_FILL_TABLE,
                  0, 255, &zero_label)
```

Build an array of labels you wish to accept:

Use the `DqAdv566BuildFilterEntry()` API to build an array of labels you wish to accept.

You can set the `new_data_only` flag to `TRUE` to store only new incoming messages in the RX FIFO:

```
// accept label 4 and 5; for label 5, only store new messages in the FIFO

lbl_filter[0] = DqAdv566BuildFilterEntry(4, FALSE, FALSE);
lbl_filter[1] = DqAdv566BuildFilterEntry(5, TRUE, FALSE);
```

Write the array of labels to hardware:

```
// Update channel 0 filter table with lbl_filter array built above
//   writes (puts) 2 labels starting at index 0

DqAdv566SetFilter(hd, DEVN, 0, DQ_AR_SETFILTER_PUT,
                  0, 2, lbl_filter);
```

Configure the mode to enable the label filter:

OR in `DQ_AR_ENABLE_FILTER` on channel 0 to enable.

```
mode_rx = DQ_AR_RATEHIGH          | DQ_AR_SDI_DISABLED |
          DQ_AR_PARITYODD         |
          DQ_AR_SLOWSLEW_DISABLED |
          DQ_AR_ENABLE_FILTER;

DqAdv566SetMode(hd, DEVN, DQ_SS0IN, 0, mode_rx);
```

3.10.2 Filtering Messages Based on Parity

To filter based on parity errors, you'll configure the RX port as described in "Reading RX Data" on page 47 and additionally configure the mode with the DQ_AR_IGNORE_BAD_DATA flag.

With this flag set, if you configure DQ_AR_PARITYODD and the received frame has an even number of ONES, the frame will be rejected. Likewise, If you configure DQ_AR_PARITYEVEN and the frame has an odd number of ONES, the frame will be rejected.

Configure the mode to reject frames with parity errors:

OR in DQ_AR_IGNORE_BAD_DATA on channel 0 to configure:

```
mode_rx = DQ_AR_RATEHIGH          | DQ_AR_SDI_DISABLED |
          DQ_AR_PARITYODD         |
          DQ_AR_SLOWSLEW_DISABLED |
          DQ_AR_IGNORE_BAD_DATA;

DqAdv566SetMode(hd, DEVN, DQ_SS0IN, 0, mode_rx);
```

If you configure DQ_AR_PARITYOFF, no frames will be rejected due to parity errors. The parity bit in the received word that gets stored in the FIFO will be the actual parity bit (bit 32) that was received from the ARINC bus, allowing users to evaluate parity errors in their application.

3.10.3 Filtering Messages Based on SDI

To filter based on the SDI field, you'll configure the RX port as described in "Reading RX Data" on page 47 and additionally configure the mode with the DQ_AR_SDI_ENABLED flag.

When DQ_AR_SDI_ENABLED flag is set, you can specify which SDI to accept by ORing in the DQ_AR_SDIMASK0 and/or DQ_AR_SDIMASK1 flags:

- neither: accept frames with SDI=0
- DQ_AR_SDIMASK0: accept frames with SDI=1
- DQ_AR_SDIMASK1: accept frames with SDI=2
- DQ_AR_SDIMASK1|DQ_AR_SDIMASK0: accept frames with SDI=3

Configure the mode to accept frames based on SDI:

The following example enables SDI filtering on channel 0 to only accept messages with SDI=2:

```
mode_rx = DQ_AR_RATEHIGH          |
          DQ_AR_SDI_ENABLED      | DQ_AR_SDIMASK1 |
          DQ_AR_PARITYODD         |
          DQ_AR_SLOWSLEW_DISABLED ;

DqAdv566SetMode(hd, DEVN, DQ_SS0IN, 0, mode_rx);
```

Appendix

A.1 Accessories

The following cables and STP boards are available for the DNx-429-516 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.

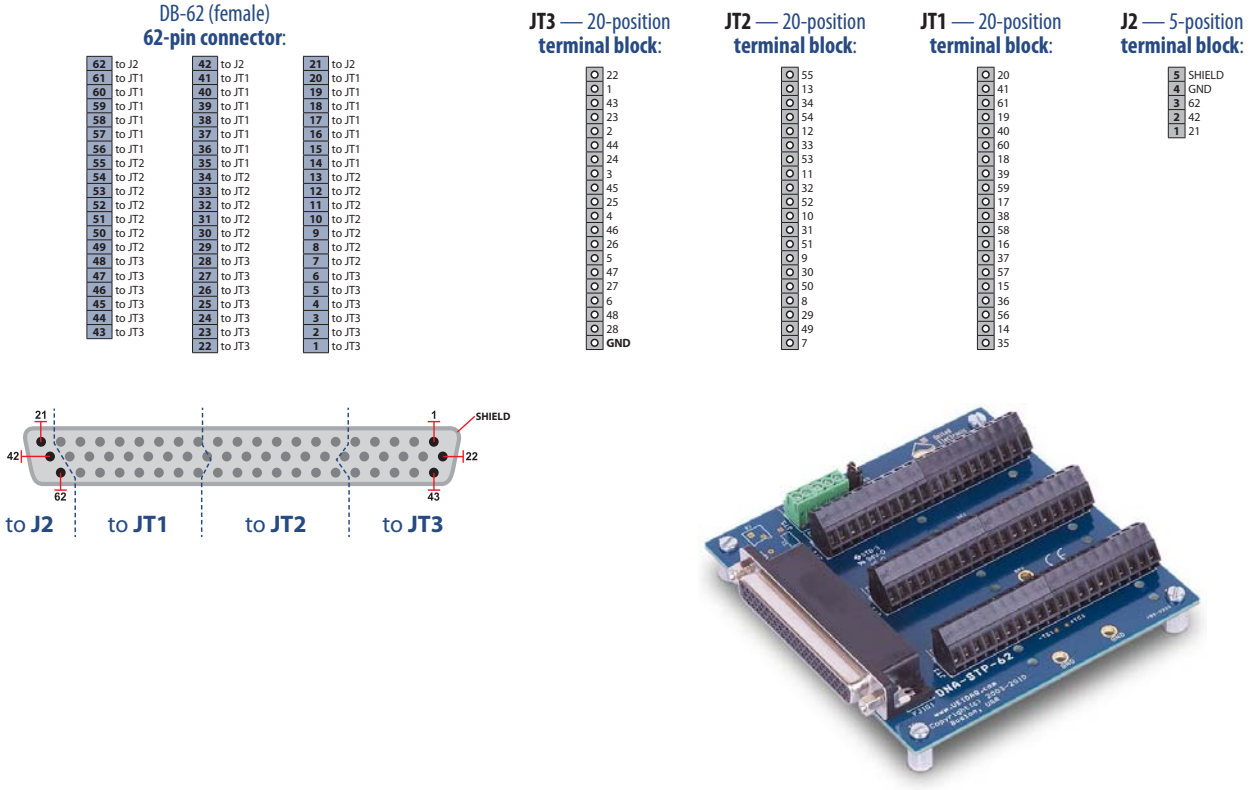


Figure A-1 Pinout and Photo of DNA-STP-62 Screw Terminal Panel

Index