DNx-VR-608
User Manual

8-Channel Variable Reluctance Sensor Interface
for the PowerDNA Cube and RACK Series Chassis

February 2024

PN Man-DNx-VR-608

CE

Contacting United Electronic Industries

Mailing Address:
249 Vanderbilt Avenue
Norwood, MA 02062
U.S.A.

Shipping Address:
24 Morgan Drive
Norwood, MA 02062
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: uei.support@ametek.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

List of Tables

Chapter 1 Introduction

This document outlines the feature-set of the DNx-VR-608 Variable Reluctance Interface boards and their use in a wide variety of motion and rotation monitoring applications.

The following sections are provided in this chapter:

- Organization of this Manual (Section 1.1)
- Manual Conventions (Section 1.2)
- Naming Conventions (Section 1.3)
- Related Resources (Section 1.4)
- Before You Begin (Section 1.5)
- DNx-VR-608 Features (Section 1.6)
- Technical Specification (Section 1.7)

1.1 Organization of this Manual

This DNx-VR-608 User Manual is organized as follows:

- **Introduction**
  Chapter 1 summarizes the features and specifications of the DNx-VR-608.
- **Functional Description**
  Chapter 2 describes the device architecture, logic, and connectivity of the DNx-VR-608.
- **PowerDNA Explorer**
  Chapter 3 shows how to explore DNx-VR-608 features through a GUI-based application.
- **Programming with the High-level API**
  Chapter 4 describes how to create a session, configure the session, and interpret results with the UeiDaq Framework API.
- **Programming with the Low-level API**
  Chapter 5 provides an overview of programming the DNx-VR-608 using the low-level C API.
- **Accessories**
  Appendix A provides a list of accessories available for use with the DNx-VR-608.

1.2 Manual Conventions

The following conventions are used throughout this manual:

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

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

NOTE: Notes alert you to important information.

Typeface	Description	Example
bold	field or button names	Click **Scan Network**
»	hierarchy to get to a specific menu item	File » New
`fixed`	source code to be entered verbatim	`session.CleanUp()`
<brackets>	placeholder for user-defined text	`pdna://<IP address>`
italics	path to a file or directory	*C:\Program Files*

1.3 Naming Conventions

The DNA-VR-608, DNR-VR-608, and DNF-VR-608 board versions are compatible with the UEI Cube, RACKtangle, and FLATRACK chassis respectively. These boards are electronically identical and differ only in mounting hardware. The DNA version stacks in a Cube chassis, while the DNR and DNF versions plug into the backplane of a Rack chassis. Throughout this manual, the term DNx-VR-608 refers to both Cube and Rack products.

1.4 Related Resources

This manual only covers functionality specific to the DNx-VR-608. To get started with the UEI IOM, please see the documentation included with the software installation. On Windows, these resources can be found from the desktop by clicking **Start » All Programs » UEI**

UEI's website includes other user resources such as application notes, FAQs, tutorials, and videos. In particular, the glossary of terms may be helpful when reading through this manual: https://www.ueidaq.com/glossary

Additional questions? Please email UEI Support at uei.support@ametek.com or call 508-921-4600.

1.5 Before You Begin

No Hot Swapping!

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

Check Your Firmware

Ensure that the firmware installed on the Cube or Rack CPU matches the UEI software version installed on your PC. The IOM is shipped with pre-installed firmware and a matching software installation. If you upgrade your software installation, you must also update the firmware on your Cube or RACK CPU. See “FirmwareUpdatingProcedures” for instructions on checking and updating the firmware. These instructions are located in the following directories:

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

1.6 DNx-VR-608 Features

DNx-VR-608 boards are 8-channel, variable reluctance (VR) sensor interface boards. The DNA/DNR/DNF-VR-608 boards are electronically identical and differ only in the hardware needed to mount the board in the selected chassis. The board is designed for use in a wide variety of motion and rotation monitoring applications. It can also be used as a general-purpose counter to measure AC signals and TTL-level digital signals, including the output from a Hall Effect sensor. This manual includes information for both Revision 1 and Revision 2 VR-608 boards.

Features of the DNx-VR-608 (Rev. 1 and Rev. 2) include:

- 8 fully differential analog input channels
- Input range from 50 mV to 250 V peak-to-peak
- Bipolar (VR) inputs rates up to 300 kHz
- Unipolar (Hall/TTL) input rates up to 18 kHz
- Conditioned inputs can be routed to TTL digital outputs
- Adaptive Threshold input mode
- True zero-cross detection
- Inter-tooth timing allows acceleration tracking
- Open circuit detection

Additional features available on Rev. 2 boards only:

- 4 isolated TTL digital inputs can be routed to internal counters
- Alarms for open circuit, stalled rotation, and overspeed conditions. Alarms can be routed to digital outputs.
- Torque measurement for dual VR sensor systems
- Digital Hall Effect sensor simulator
- Timed Pulse Period Measurement (TPPM) Mode that features a dynamic sampling window based on input frequency.

1.6.1 Measurement Capabilities

The DNx-VR-608 contains eight differential analog input channels designed to support an extremely wide variety of variable reluctance sensors. Sensor compatibility is made possible by the board's wide input range (50 mVpp to 250 Vpp) and high input impedance (40 kΩ). The board supports a maximum pulse rate of 300 kHz for inputs ≥ 3.2 Vpp; measurable pulse rates for lower signal levels are shown in **Table 1-1**.

Signal peaks and zero crossings may be configured for auto-detection. For example, an "Adaptive Peak Threshold" (APT) mode sets the threshold to 30% of a time-averaged input. A watchdog circuit resets the input to the minimum threshold level if the input "drops out" for 85 milliseconds. Another APT mode auto-computes the thresholds from the minimum and maximum voltages measured by the board.

Users can also set zero crossing thresholds and peak thresholds to fixed voltage levels for special use cases such as Hall Effect measurements and custom signal conditioning applications.

Revision 2 boards provide four digital input channels for reading sensors which already output a conditioned TTL-level signal. Digital inputs bypass the front-end VR circuitry for a faster read rate.

1.6.2 Operating Modes

The DNx-VR-608 supports the following per-channel modes:

- **Timed Count/Frequency:** counts the number of teeth detected during a specified time interval and returns velocity in teeth/sec or RPM.
- **N-Pulse:** measures the time taken to detect N teeth and returns velocity in RPM.
- **Z-Pulse:** measures the number of teeth and the time elapsed between two Z/Index pulses (The Z/Index tooth is usually a gap or a double tooth on the encoder wheel).
- **Timed Pulse Period Measurement (TPPM):** counts the number of edges detected within a default measurement window to calculate RPM. The measurement window dynamically increases when needed to accommodate slower input frequencies. TPPM is for continuous signals only, i.e., wheels with no Z-tooth (Revision 2 only).

Channel pairs may be configured for the following modes:

- **Quadrature Encoder:** measures relative position and rotational direction from a quadrature encoder sensor
- **Torque:** measures torque in dual-sensor systems by comparing the timing of the pulses from each sensor (Revision 2 only). Torque Mode is for analog signals only.

1.6.3 Digital Outputs

The DNx-VR-608 provides four isolated TTL-level digital outputs that change state on the threshold crossing of the input waveform. This allows the board to be used as a signal conditioning front end for existing TTL-based counter test systems. The board supports a matrix configuration where any input can be directed to any or all of the digital outputs. See Section 2.2.5 for more information on supported digital output features for the DNx-VR-608.

On Revision 2 boards, digital outputs may also be used for Z-tooth detection, for open circuit, stalled rotation, or overspeed alarms, or for generating simulated VR pulse trains. Also, signals on the internal sync bus can be routed to the digital outputs.

For applications where the VR-608 is primarily used as a signal conditioner, the input to DOut mapping can be set to start running at power up, even if the host Cube or RACK is not running any application software.

Dividers of 2, 4, 8, or 16 can be applied to signals that are routed to the digital outputs. The signals can be inverted, i.e., phase shifted. The output pulse will always have a 50% duty cycle.

1.6.4 Isolation

The DNx-VR-608 offers 350 Vrms of isolation between itself and other I/O boards as well as between the I/O connections and the chassis. In addition, the inputs are divided into four sets of channel pairs, and each two-channel pair is isolated from the others. Each isolated channel pair is also supported with a logic level digital output.

1.6.5 Environmental Conditions

Like all UEI I/O boards, the DNx-VR-608 offers operation in extreme environments and has been tested to 5 g vibration, 100 g shock, temperatures from -40 to +85 °C, and will function at altitudes up to 70,000 feet.

1.6.6 Accessories

All field-wiring connections to the DNx-VR-608 are made through a standard 37-pin D connector, allowing OEM users to build custom cabling systems through off-the-shelf components. Users may also connect the DNx-VR-608 board to UEI's DNA-STP-37 screw terminal panel via the DNA-CBL-37 cables.

1.6.7 Software Support

The DNx-VR-608 includes a software suite supporting Windows, Linux, QNX, VxWorks, RTX, and most other popular real-time operating systems. Windows users may use the UEIDAQ Framework, which provides a simple and complete software interface to Windows programming languages and DAQ applications (e.g., LabVIEW, MATLAB). All software support includes extensive example programs that make it easy to cut-and-paste the I/O software into your applications.

1.7 Technical Specification

Table 1-1 summarizes the technical specifications for the DNx-VR-608 board. All specifications are at 23 °C ±5 °C and apply to both Rev. 1 and Rev. 2 boards unless otherwise noted.

Table 1-1 Specifications

Number of VR channels	8
Channel configuration	Differential
Channel isolation	4 isolated banks of two channels
Input impedance	> 40 kΩ and < 250 pF
Input voltage range	Up to 250 Vpp
Maximum pulse rate (bipolar, VR input mode)	300 kHz at ≥ 3.2 Vpp 200 kHz at ≥ 2.1 Vpp 100 kHz at ≥ 1.1 Vpp 50 kHz at ≥ 0.5 Vpp 25 kHz at ≥ 0.25 Vpp 10 kHz at ≥ 0.125 Vpp ≤ 5 kHz at ≥ 0.08 Vpp
Minimum detectable input (VR input mode)	50 mVpp, fixed input mode
Max pulse rate (unipolar/TTL mode)	500 kHz (V_{low} < 1.2 V, V_{high} > 2.2 V)
Measurement Accuracy	see Section 2.3
Inter-tooth timing	
Time-base resolution	15.15 ns
Counter depth	32-bits
Timing measurement range	20 µs to 65 seconds
Overvoltage protection	300 Vpp
Input FIFO size	**Rev. 1:** 512 x 32 **Rev. 2:** 2k x 32
DOut simulation accuracy (continuous mode)	0.5% of expected frequency
General Specifications	
Power dissipation	< 3 W
Isolation	350 Vrms
Operating temperature	Tested -40 °C to +85 °C
Operating humidity	0% to 95%, non-condensing
Vibration IEC 60068-2-6 IEC 60068-2-64	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500 Hz, broadband random
Shock IEC 60068-2-27	100 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
Altitude	120,000 ft
MTBF	180,000 hours

Chapter 2 Functional Description

This chapter describes the device architecture, hardware, and functionality of the DNx-VR-608 Variable Reluctance Interface. The following sections are provided in this chapter:

- VR Sensor Overview (Section 2.1)
- DNx-VR-608 Device Architecture (Section 2.2)
- Measurement Resolution and Accuracy (2.3)
- Indicators and Connectors (Section 2.4)
- DNA Cube Jumpers (Section 2.5)
- Pinout (Section 2.6)

2.1 VR Sensor Overview

Variable reluctance (VR) sensors are electromechanical transducers used to measure the rotational speed and position of a crankshaft. The VR sensor consists of a permanent magnet surrounded by a coil of wire. A toothed wheel made of a ferrous material is attached to the crankshaft, and the VR sensor is positioned close to the teeth (**Figure 2-1**).

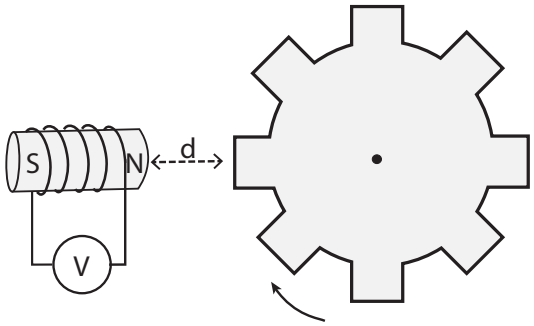

Figure 2-1 Variable Reluctance Sensor and Toothed Wheel

When a tooth rotates past the sensor, the distance between the wheel and sensor changes, changing the magnetic flux through the pickup coil. An AC voltage is induced across the coil as shown in **Figure 2-2**. In the default polarity, voltage is positive when the tooth approaches the sensor and negative when the tooth moves away. Notably, the positive-to-negative zero crossing coincides with the center of the tooth.

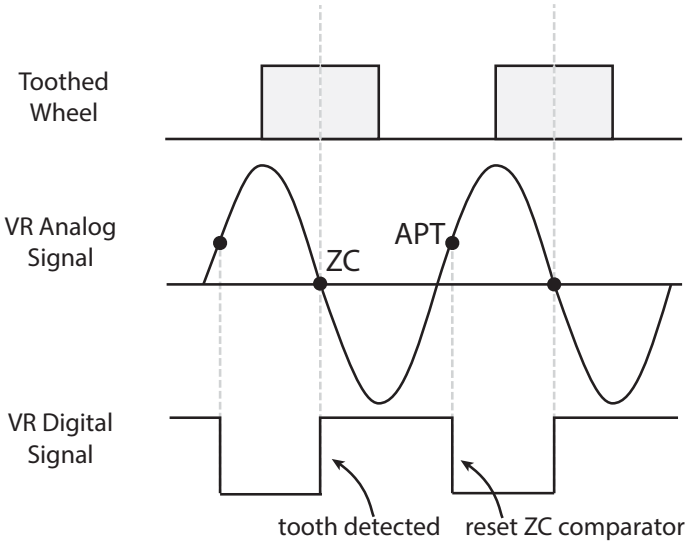

Figure 2-2 Example VR Sensor Signals

A VR interface board reads in the analog waveform and conditions it into a digital signal (**Figure 2-2**). The digital signal goes high when a positive to negative zero crossing (ZC) is detected and resets to low when the signal level rises above a certain threshold. Therefore, rising edges of the pulse train correspond to tooth detections. The pulse train feeds into a digital counter/timer which measures the number of teeth and the time between teeth. Raw data can then be used to calculate shaft properties such as rotational speed, acceleration, position, and torque.

Note that VR sensors are sensitive to noise. Use best practices when wiring the device and shielding signals.

2.1.1 Index Tooth

On some toothed wheels, one tooth is a different height or width than the others. This tooth is called the “index tooth” or “Z-tooth.” The Z-tooth changes the frequency and amplitude of the AC waveform as shown in **Figure 2-3**. For each Z-tooth type, the top row shows the physical teeth, the middle row shows the simplified AC waveform, and the bottom row shows the digital output signal. The index tooth determines the position of the shaft (i.e., the VR interface resets the count upon detecting the index tooth).

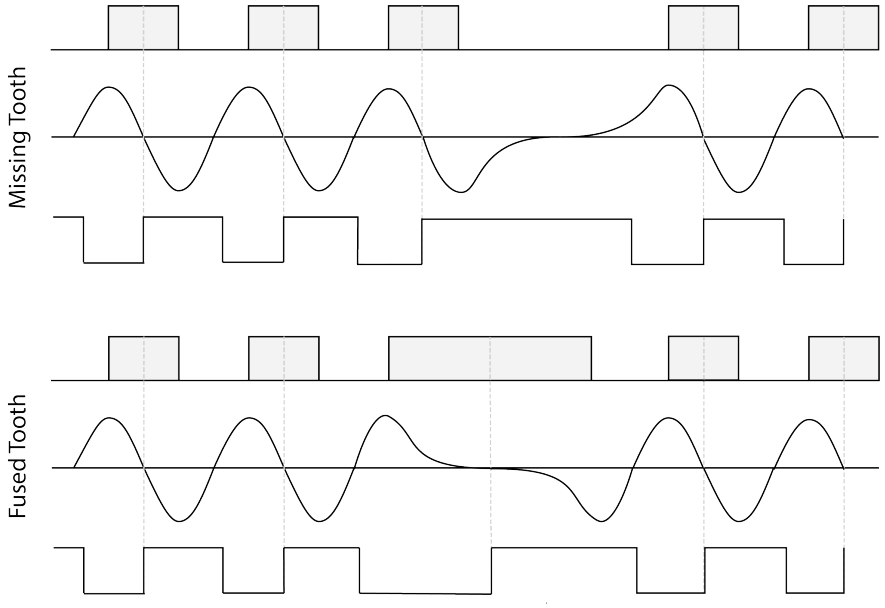

Figure 2-3 Z-Tooth Types

2.1.2 Torque Measurement

Torque measurements require two toothed wheels on the same shaft: one toothed wheel is under torque and the other wheel is under zero torque. As the shaft twists under torque, the two wheels become slightly misaligned. The VR interface indirectly measures the degree offset by counting the time between when a tooth on the leading wheel passes the sensor and when a tooth on the lagging wheel passes the sensor. The torque pulse signal is shown in **Figure 2-4**. This offset can be converted to torque from the shaft specification (see Section 2.2.4.2).

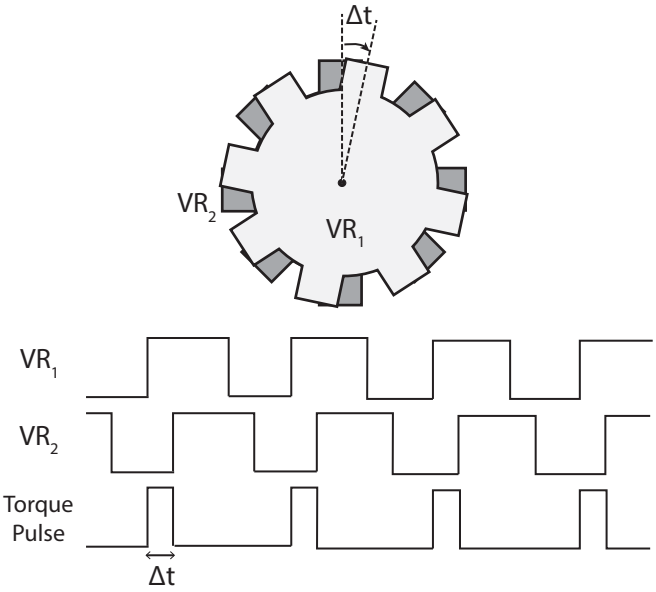

Figure 2-4 Dual VR Sensor Torque Measurement

2.2 DNx-VR-608 Device Architecture

A simplified block diagram of the DNx-VR-608 Variable Reluctance Interface board is illustrated in **Figure 2-5**.

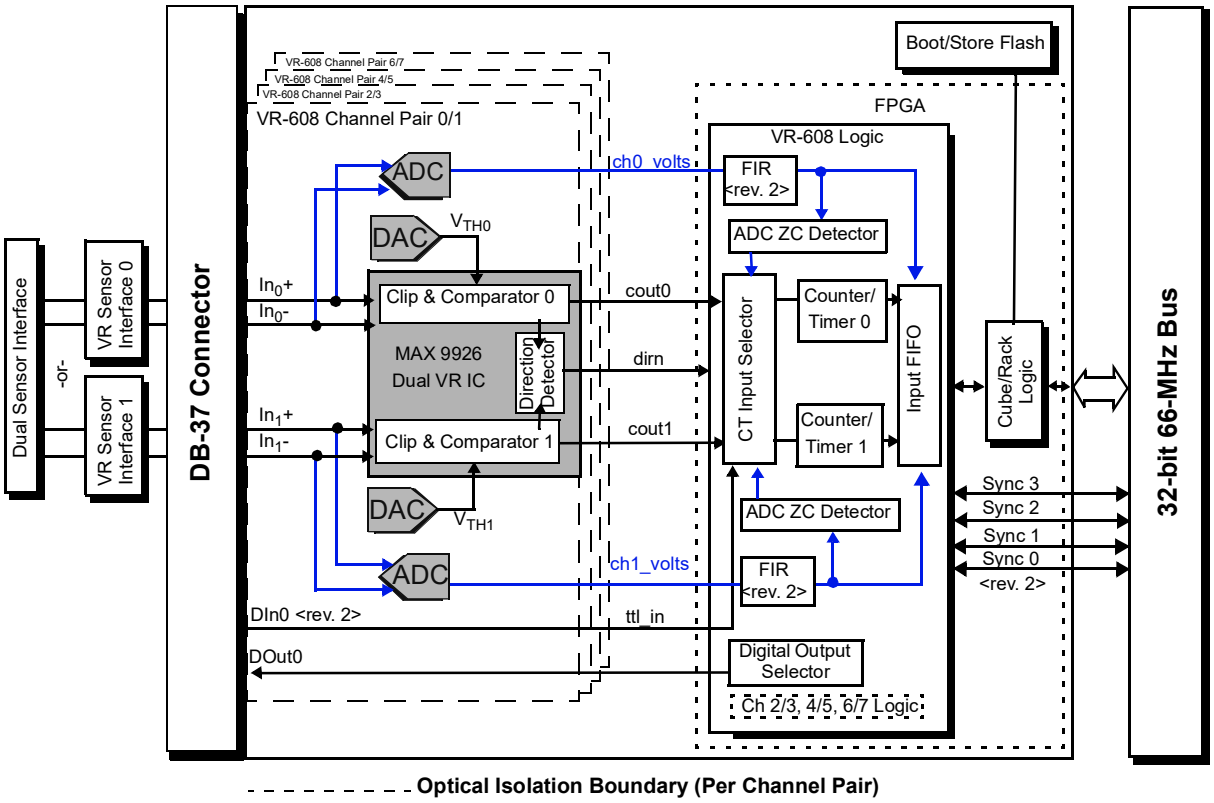

Figure 2-5 Simplified Block Diagram of the DNx-VR-608

2.2.1 Inputs

The front-end of the DNx-VR-608 exposes eight differential analog inputs through the DB-37 connector. The eight channels are grouped and isolated in pairs (0&1, 2&3, 4&5, 6&7), where each pair can accept either two individual VR sensors or one out-of-phase dual VR sensor. Revision 2 boards also provide four TTL-level digital inputs which route directly to the counter timer modules.

The DNx-VR-608 is designed to accept the following input signal sources:

- **AC waveform from a VR sensor** - The VR sensor's signal and return wires connect to a single channel (e.g., In_0+/- of Channel 0). The board supports systems with or without a zero/index/z-tooth.
- **AC waveforms from a dual VR system** - The two VR sensors are connected to a channel pair (e.g., one sensor on In_0+/- of Channel 0 and one on In_1+/- of Channel 1). The board detects rotational direction ("dirn") by checking for the leading channel. Revision 2 boards can also measure the phase shift between the two channels; this feature is used to compute torque in setups where one toothed wheel is under torque and the other is not.

- **TTL-level digital pulse train** - Some sensors, such as Hall Effect sensors, already output a conditioned 0-5 V square wave.
  - **Revision 1:** The digital sensor's signal and return wires connect to a single VR-608 channel (e.g., In_0+/- of Channel 0); the sensor's power wire does not connect to the VR-608.
  - **Revision 2:** The digital sensor can either connect to a VR-608 channel or connect directly to a counter/timer module through DIn/DIGnd. The direct DIn connection is the recommended configuration as it bypasses the zero crossing and peak threshold stages for a faster read rate.

2.2.1.1 Input Conditioning

All VR analog inputs pass through a limiter circuit before entering the Dual VR IC and ADCs.

Revision 2 boards provide a digital FIR filter after each ADC to smooth out noise on the VR inputs. It is useful when the ADC tooth rate is greater than 10 kHz. By default, the FIR filter is disabled. It may be enabled in board configuration.

2.2.2 Zero Crossing Detection

Zero crossing detection is performed by either the Dual VR IC (default) or the channel's ADC. The ADC input path is highlighted in blue in **Figure 2-5**.

The DNx-VR-608 supports the following zero crossing modes:

- `VR608_ZC_ONCHIP`: automatically determines the mid-range of the analog sine wave signal using the built-in zero-cross detector of the Dual VR IC.
- `VR608_ZC_FIXED`: fixed to a particular voltage level by the user and detected by the ADC module. If the desired zero crossing level is 0 V, UEI recommends setting the desired zero crossing level to a value slightly above 0 to avoid false positives, e.g., 0.1 V.
- `VR608_ZC_LOGIC`: calculated by the ADC module as (min+max)/2, subject to a configurable moving average.

Because the Dual VR IC expects a bipolar input, Hall Effect sensors and other unipolar digital inputs should use an ADC-based mode (`ZC_FIXED` OR `ZC_LOGIC`). ADC-based modes cap the input pulse rate at 18 kHz.

DIn connections will use `VR608_ZC_ONCHIP` by default. This is the recommended configuration. Using `VR608_ZC_ONCHIP` mode is also recommended with analog inputs.

2.2.2.1 Dual VR IC

The Dual VR IC's purpose is to convert the VR sensor's analog signal into a digital output. The Dual VR IC electronic chip is the Maxim MAX9926 Variable Reluctance Sensor Interface.

Figure 2-6 takes a closer look at the logic for one channel in the Dual VR IC. After the input signal has passed the limiter circuit, it enters the Dual VR IC and is clipped from ±125 V down to ±5 V using Zener diodes. The ±5 V input is amplified before entering the zero crossing detection circuit.

The zero crossing circuit centers around an inverting comparator. The reference voltage at the comparator's non-inverting (+) terminal switches between the zero crossing level and a configurable peak threshold level depending on the current state of the comparator output (cout). When cout is low, the input is compared to the zero crossing level to determine when cout should be raised. When cout is high, the input is compared to the peak

threshold to determine when cout should be dropped. The peak threshold allows the comparator to ignore zero crossings due to noise. The resulting digital output signal is illustrated in **Figure 2-2**.

2RX/1TX ARINC-429 transceiver protocol controllers (FPGA/DSP control/access Block 1)

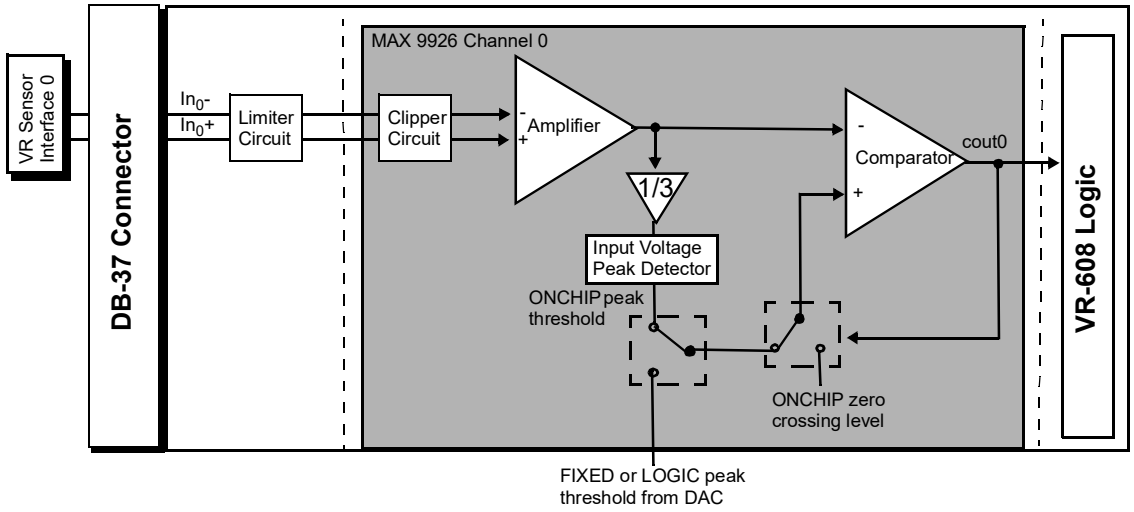

Figure 2-6 Logic Block Diagram for a Dual VR IC Channel

2.2.2.2 Peak Threshold Levels

Setting a peak threshold level introduces hysteresis to the zero crossing detector. The DNx-VR-608 provides the following peak threshold configurations:

- `VR608_APT_ONCHIP`: automatically adjusts the threshold to 1/3 of the peak of the previous analog input cycle, using the built-in APT detector of the Dual VR IC (see **Figure 2-7**). Note that if the input voltage remains lower than the adaptive threshold for 85 ms, the threshold will be dropped to the chip's minimum value, driving the comparator output low; this ensures pulse recognition in the presence of an intermittent sensor connection.
- `VR608_APT_LOGIC`: calculated as (max)/2^n, where n=1, 2, 3, or 4 and "max" is the moving average of previous analog input peaks (Rev. 2 only).

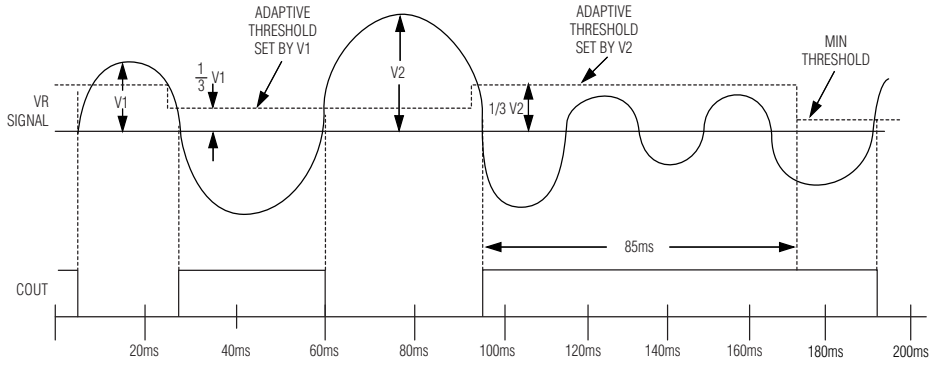

Figure 2-7 Adaptive Peak Threshold (APT) from the MAX9926 Specification

Since the amplitude of the VR sensor signal varies with RPM, an adaptive peak threshold (APT_ONCHIP or APT_LOGIC) ensures the best performance over a wide range of speeds.

The Dual VR IC can either use an internal adaptive algorithm (APT_ONCHIP) or accept an external peak threshold from the DAC (APT_LOGIC). APT_ONCHIP is not compatible with the ADC module's zero crossing detector. APT_LOGIC is not compatible with ZC_ONCHIP. Supported peak threshold and zero crossing mode combinations are summarized in **Table 2-1**.

Table 2-1 Peak Threshold and Zero Crossing Modes

Zero Crossing Mode	Adaptive Peak Threshold Mode	
	APT_ONCHIP	APT_LOGIC
ZC_ONCHIP	●	
ZC_FIXED		●
ZC_LOGIC		●

2.2.3 ADC Module

As shown in **Figure 2-5**, each VR input channel is monitored by its own 14-bit ADC. The ADC captures the raw voltage level, which is used by the VR-608 logic for a number of different features including:

- Zero crossing detection in ZC_FIXED and ZC_LOGIC modes
- Minimum & maximum voltage peak detection in APT_LOGIC mode
- Open circuit detection
- Stall detection
- Debug/diagnostics

2.2.4 Counter/Timer Module

The VR-608 Logic chip is responsible for gathering data from each channel's Dual VR IC and ADCs, accumulating the digital data readings inside internal FIFOs, calculating results (velocity, position, etc.) within the Counter/Timer modules, and buffering the data for the firmware to return to the application software. As shown in **Figure 2-5**, all inputs and outputs are optically isolated and over-voltage protected.

The VR-608 Logic chip contains eight Counter/Timer modules, one per VR input channel. Depending on the user configuration, a multiplexer routes digital waveforms into the Counter/Timer from either the Dual VR IC, the ADC module zero crossing detector, a Digital Input line (Rev. 2 only), or the Torque module (Rev. 2 only).

The internal structure of a Counter/Timer module is shown in **Figure 2-8**.

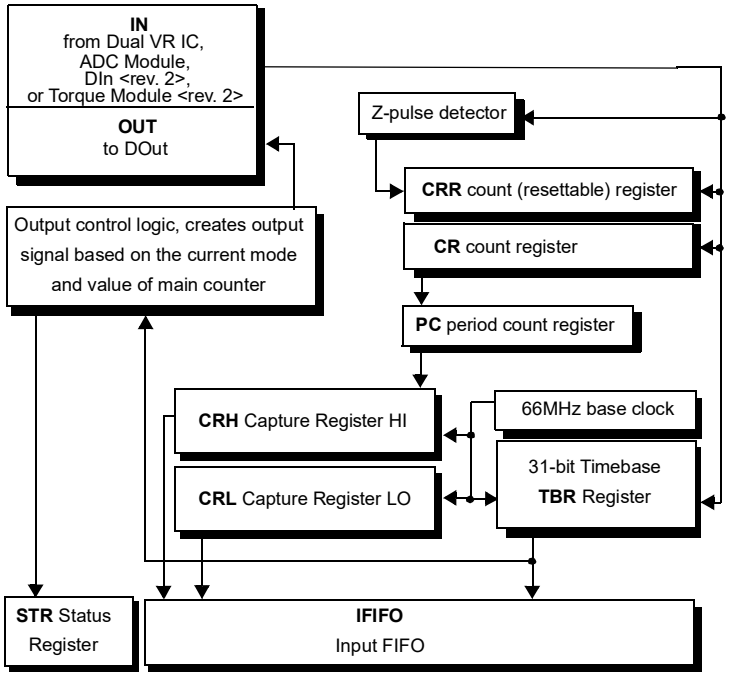

Figure 2-8 DNx-VR-608 Counter/Timer Module

Results from the counting process are stored in the CRR, CR, CRH, and CRL registers. The data stored in these registers depends on the counting mode as described in Section 2.2.4.1.

Count register data can optionally be time stamped and copied over into the input FIFO; this feature is useful for tracking how the data changes over time. Either the FIFO or immediate register values may be read upon request from the firmware. Users can configure the FIFO watermark level to control data flow from the Cube/Rack.

2.2.4.1 Counter/Timer Modes

The following Counter/Timer modes are per-channel configurable:

- **Timed:** counts the number of pulses in a given time interval.
- **Timed Pulse Period Measurement (TPPM):** counts the number of edges detected within a default measurement window in order to calculate RPM. Two cycles of the input waveform are needed for the velocity calculation. The measurement window may be dynamically increased by a configurable timeout period to accommodate slower input frequencies so that the measurement can be completed. TPPM is for continuous signals only, i.e., wheels with no Z-tooth (Revision 2 only)
- **Z-Pulse:** counts the number of pulses and time interval between missing or fused Z-tooth detections.
- **N-Pulse:** measures the time to detect a given number of pulses.

The following Counter/Timer modes are designed for dual VR sensors and must therefore be configured as a pair:

- **Torque:** measures the time delay between pulses on the even and odd channels (Revision 2 only).
- **Quadrature Decoder:** counts pulses on the even channel; the counter increments or decrements depending on whether the odd channel leads or lags.

Table 2-2 summarizes the contents of the Counter/Timer registers in each mode. Some registers remain unused in certain modes.

Table 2-2 Counter/Timer Register Data

Mode	CRR	CR	CRH	CRL
Timed	Number of rising edges (reset by Z-tooth detection)	Cumulative number of rising edges	Time in 66 MHz clocks from the first to last rising edge in the configured interval	Number of rising edges during CRH
TPPM	n/a	n/a	Counts (clock ticks)	Number of pulses observed in the measurement window
Z-Pulse	Number of rising edges (reset by Z-tooth detection)	Cumulative number of rising edges	Time in 66 MHz clocks from the first to last rising edge in the configured interval	Number of rising edges during CRH
N-Pulse	n/a	n/a	Total HI time in 66 MHz clocks	Total LO time in 66 MHz clocks
Torque	n/a	n/a	Total HI time in 66 MHz clocks (time between rising edges on the even and odd channels)	Total LO time in 66 MHz clocks
Quadrature Decoder	Pulse count (reset by Z-tooth detection)	Pulse count (adds or subtracts from 0x80000000 depending on direction)	n/a	n/a

NOTE: On Revision 2 boards, a stalled input clears the CRR, CRH, and CRL counters. The measurement interval restarts when the input is reconnected.

2.2.4.2 Interpreting Count Register Data

This section describes how to compute the shaft's rotational properties from the raw counter data. The software automatically converts raw counter data when reading immediate values in Timed, Z-Pulse, TPPM, or N-Pulse mode. Torque mode data and any data read from the FIFO need to be converted manually.

Timed and Z-Pulse Modes:

$$\text{position} = \text{CRR}$$

$$\text{tooth count} = \text{CR}$$

$$\text{velocity} = \frac{\text{\# of teeth}}{\text{second}} = \frac{\text{CRL}(n/(n-z))\text{-}1}{\text{CRH}/66{,}000{,}000}$$

where “n” is the total number of teeth on the wheel (including the Z-tooth) and “z” is the width of the Z-tooth in units of teeth. For example, a gap that takes the place of two normal teeth would be z=2.

CRL is the number of detected rising edges over the time given by CRH. If the toothed wheel has a missing or fused Z-tooth, CRL is less than the actual number of encountered teeth and is therefore scaled up by a factor n/(n-z). We subtract 1 because there is one more rising edge than the number of periods.

NOTE: If “n” and “z” are configured in the application, the software returns velocity in units of RPM instead of teeth/second.

TPPM Mode:

$$\text{velocity} = \frac{1.0}{\left(\text{CRH}\frac{(1.0/(66,000,000))}{(\text{CRL}-1)}\right)}$$

CRL is the number of pulses observed in the measurement window over the time given by CRH. Note that CRL must be greater than one.

N-Pulse Mode:

$$\text{velocity} = \frac{\text{\# of teeth}}{\text{second}} = \frac{n+z}{(\text{CRH}+\text{CRL})/66{,}000{,}000}$$

CRH+CRL represents the total time to detect “n” teeth, where “n” is the number of teeth on the wheel. If the toothed wheel has a missing or fused Z-tooth of width “z”, the actual number of encountered teeth is n+z.

It is possible to recreate acceleration profiles by recording the speed at regular time intervals. To obtain intertooth timing data, use N-Pulse Mode configured for n=1, store data into the FIFO, and enable time stamping. Acceleration would be the rate of change of velocity, i.e., $(v_2\text{-}v_1)/(t_2\text{-}t_1)$.

Torque Mode:

The Torque Mode channel measures the angular offset between the two wheels as a fraction of the inter-tooth spacing on one wheel. If the torsional properties of the shaft are known, this offset ratio may be converted to torque.

$$\text{torque} = k \times \left(\frac{CRH}{CRH + CRL}\right)\left(\frac{360°}{n}\right)$$

where “k” is a shaft constant with units of Newton-meters/degree (or pound-feet/degree) and “n” is the total number of teeth on the wheel (including the Z-tooth).

2.2.5 Digital Outputs

The VR-608 Logic chip controls four digital output lines which can be routed in hardware to different internal signals. Revision 1 and 2 boards both support:

- **Zero Crossings from Dual VR IC:** goes high when a tooth is detected in `ZC_ONCHIP` mode; this is the “cout” output from the Dual VR IC comparator.
- **Zero Crossings from ADC Module:** goes high when a tooth is detected in `ZC_FIXED` or `ZC_LOGIC` modes.
- **Z-Tooth Detections:** pulses high when an index tooth is detected.
- **Quadrature Decoder Direction:** indicates direction of rotation for a quadrature-connected dual VR sensor; this is the “dirn” output from the Dual VR IC.
- **Fixed Output:** set to logical 0 or logical 1.

Revision 2 boards also support:

- **Torque Module Output:** this is the digital signal measured by the Counter/Timer in Torque Mode. The output goes high when a tooth is detected on the leading channel and goes low when a tooth is detected on the lagging channel. When a a torque pulse is routed to a digital output, the maximum output frequency is 75 kHz.
- **Alarms:** alarms can be configured to cause DOut signals to go high for any of the following conditions:
  - Open circuit (disconnected input); for the physical wiring of analog inputs only; does not apply to TTL inputs. If the input signal remains within the range that indicates an open circuit (0.5 V to 1.5 V), for a period of 0.12 ms, a status bit will be set to indicate the open circuit alarm condition.
  - Stalled rotation. If the input signal remains within a lower and upper threshold without any zero crossings being detected for a period of 0.5 seconds, a status bit will be set to indicate the stalled alarm condition.
  - overspeed condition.
- **VR Sensor Simulator:** simulates a conditioned VR sensor output, including an optional missing or fused Z-tooth. It can also generate a simulated dual-VR sensor signal on a pair of digital outputs.
- **Sync:** monitors the Sync bus (Sync [3:0]). Sync signals can be routed to the digital outputs See Section 2.2.6.
- **End of Measurement:** outputs a 1 microsecond pulse when a measurement is complete. The measurement interval depends on the count mode configuration.

- **Dividers for Digital Outputs:** Dividers can be applied to signals that are routed to the digital outputs. Each digital output can separately be configured with its own divider value. Available divisors are 2, 4, 8, and 16. The output signals can also be inverted. The output pulse will always have a 50% duty cycle.

2.2.6 Sync Out

The VR-608 has an internal Sync bus (Sync[3:0]) that allows signals to be sent to the sync lines on the backplane. With the exception of the alarm signals, all of the signals that can be sent to the digital outputs can also be sent to the sync out lines. This means that sync lines can be routed to other sync lines. The sync bus signals can also be routed to the digital outputs.

A multiplexer is used to route all sync out signals. The multiplexer is also used when signals need to be routed from a channel to a digital output outside of the channel's block, e.g., when a Z-pulse signal from channel 1 is routed to DOut 2.

Note that the sync signals have priority over the multiplexer lines. This means that in a configuration where both a sync out signal and a digital output signal need to be routed through the multiplexer, only the sync out pulse will occur.

Signals from the same channel can be routed to a sync out line and a digital output as long as the digital output is in the channel's block.

2.3 Measurement Resolution and Accuracy

The DNx-VR-608 accuracy is largely based on the chassis master clock. The 66 MHz master clock offers the following accuracy:

- 10 ppm initial accuracy
- 15 ppm temperature dependent drift over the -40 to +85 °C operating range
- 5 ppm drift over the first year, less per year thereafter

For the calculations here, we will assume a flat 30 ppm accuracy of the 66 MHz clock, or a period of 15.15 nanoseconds. Unlike most analog measurements, the count mode of the DNx-VR-608, along with the input frequency, have a great impact on overall measurement accuracy and resolution.

2.3.1 Timed Mode

Timed Mode counts the number of teeth detected during a specified time interval and returns the velocity in teeth per second or RPM. In this mode, the accuracy is determined by the input frequency and the duration of the sampling. The rough equation to determine resolution is simply:

$$\text{Resolution} = \frac{100}{(\text{input frequency}) \times (\text{sample duration})}$$

For example, if the input frequency is 10 kHz, sampled for 1 second, the resolution is one part in 10,000 or 0.01%. If the input frequency is 1 kHz and the sample duration is 0.1 seconds, the resolution is one part in 100, or 1%. To compensate for crystal error you need only add a flat 0.003% (for the 30 ppm master clock error).

$$\text{Accuracy} = \frac{100}{(\text{input frequency}) \times (\text{sample duration})} + 0.003\%$$

Note that if you experiment with these numbers, you will see that this measurement mode is well suited for applications with high input frequencies, that do not have to be updated quickly. For lower frequency inputs and/or for systems requiring quicker data updates, consider using N-pulse Mode.

2.3.2 N-Pulse Mode

N-Pulse (aka N-teeth) Mode measures the time taken to detect N teeth and returns velocity in RPM. Unlike Timed Mode, N-Pulse mode determines RPM by measuring the time delay between teeth. This mode simply counts the number of 66 MHz master clock pulses that occur between teeth. Note that to return RPM, you need to enter the number of teeth on the toothed wheel into your software configuration. The resolution of this mode is:

$$\text{Resolution} = \frac{100 \times (\text{input frequency})}{\text{N} \times 66{,}000{,}000}$$

where N is the number of teeth you wish to include after the first tooth. For example if you want to measure the time between adjacent teeth, N=1. If you wish to measure the time between the first tooth and the fourth, N=3.

This mode is subject to the same 0.003% accuracy of the 66 MHz master clock, so:

$$\text{Accuracy} = \frac{100 \times (\text{input frequency})}{\text{N} \times 66{,}000{,}000} + 0.003\%$$

For example, if your input frequency is 20 kHz, and you select N=1, your system accuracy would be 0.0333%. If the input frequency remains 20 kHz, but you sample the time between the first tooth and the third (N=2), the accuracy becomes 0.0182%.

Selecting a larger “N” increases the resolution of the measurement, but also decreases the frequency at which you can make measurements. This is a trade-off that must be made based on your applications’ accuracy requirement, the input frequency and how quickly you need to sample the data. If your frequency is high and/or your need to acquire the data is not fast, you may wish to consider using Timed Mode.

2.3.3 Timed Pulse Period Measurement Mode

Timed Pulse Period Measurement (TPPM) Mode counts the number of edges detected within a default measurement window to calculate RPM. Resolution and accuracy are calculated using the same equations that are used for Timed Mode as described in Section 2.3.1. However, the sampling duration (measurement window) may be dynamically increased to accommodate slower input frequencies.

2.3.4 Torque Mode

Torque Mode measures the time delay between pulses on the even and odd channels. The measured value is accurate to within 2% of full scale.

2.4 Indicators and Connectors

Figure 2-9 shows the locations of the LEDs and connectors on the DNx-VR-608. The LED indicators are described in **Table 2-3**.

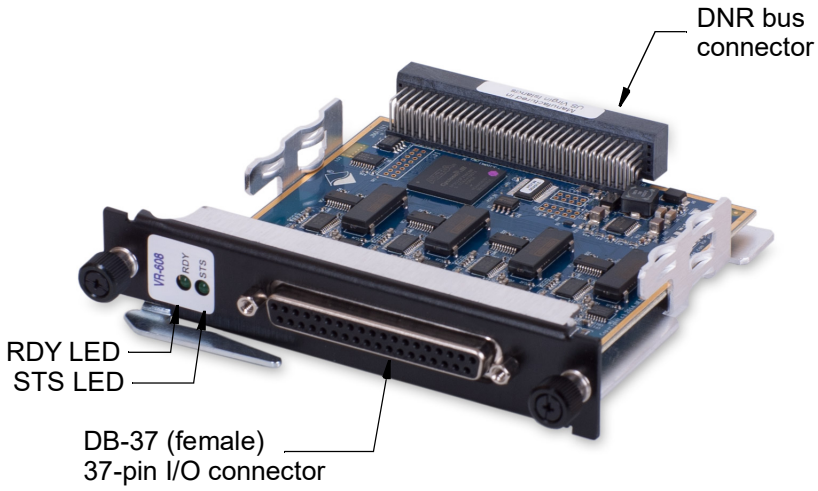

Figure 2-9 Photo of DNR-VR-608 Board

Table 2-3 DNx-VR-608 LED Indicators

LED Name	Description
RDY	Board is powered up and operational
STS	STATUS: **OFF**: Configuration mode (e.g., configuring channels, running in Point-by-Point mode) **ON**: Operation mode (e.g., running in DMap)

2.5 DNA Cube Jumpers

The DNA-VR-608 jumper location to be used with the “PowerDNA Field Installation Guide” is shown in **Figure 2-10**:

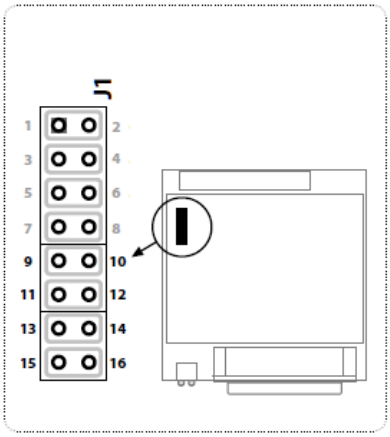

Figure 2-10 DNA-VR-608 Jumper Address Location

2.6 Pinout

Figure 2-11 illustrates the pinout of the DNx-VR-608 for Rev. 1 and Rev. 2 boards. Connections are made through a B-size 37-pin D-sub connector. The only functional difference between Rev. 1 and Rev. 2 pinouts is the new D/TTL In lines on Rev. 2 boards.

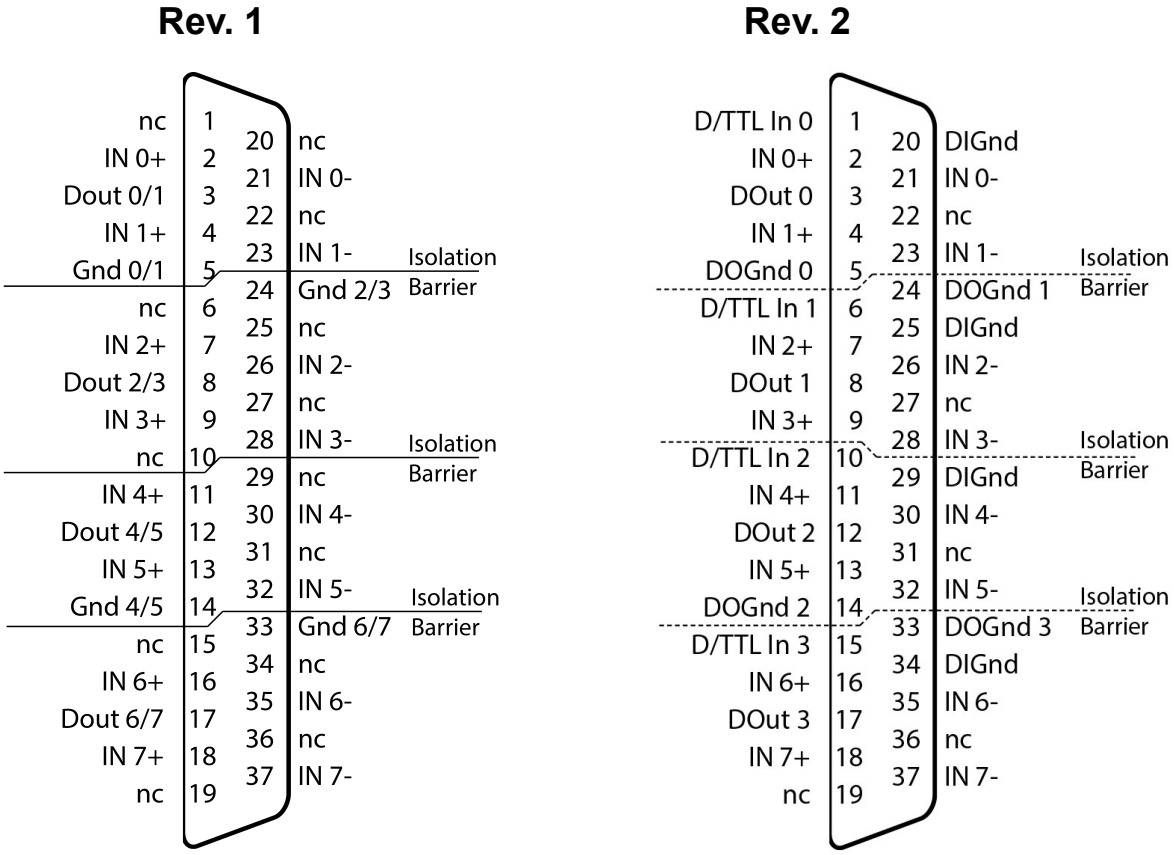

Figure 2-11 Pinout Diagram of the DNx-VR-608

Table 2-4 DNx-VR-608 Pinout Descriptions

Rev. 1	Rev. 2	Description
IN0+...IN7+ IN0-...IN7-	IN0+...IN7+ IN0-...IN7-	Differential analog input channel 0...7; for Variable Reluctance sensors
DOut 0/1...6/7 Gnd 0/1...6/7	DOut 0...3 DOGnd 0...3	TTL digital outputs
	DIn 0...3 DIGnd	TTL digital inputs; for logic-level signals including Hall Effect sensors

Unused pins may be left open/disconnected. Do not connect anything to the pins marked as “nc”.

VR Sensor Polarity

On the DNx-VR-608, tooth detections correspond to negative slope zero crossings. If your VR sensor is wired for positive slope zero crossings (or if the wheel has holes instead of teeth), simply switch the In+ and In- wires. You can monitor both types of zero crossings by reading the ADC status.

2.6.1 Loopback Test Wiring (Rev. 2)

Revision 2 boards can generate simulated digital VR signals. By wiring the digital output to an input channel, basic software functionality may be tested without needing to connect a physical VR sensor.

- When simulating a digital sensor, connect DOut -> DIn and DOGnd -> DIGnd as you would for a real sensor.
- When simulating an analog sensor, an external capacitor between DOut and In+ converts the 0-5 V simulator output to the bipolar input expected by the Dual VR IC. This allows you to transition from simulator to sensor without software changes. Coupling capacitors are necessary for Torque Mode loopback tests (see **Figure 2-12**).The capacitor is optional for Timed, TPPM, N-Pulse, and Z-Pulse mode simulations as long as the front end is configured for a digital input i.e., `ZC_LOGIC` or `ZC_FIXED`. Real VR sensors connect directly to the DNx-VR-608 without a capacitor.

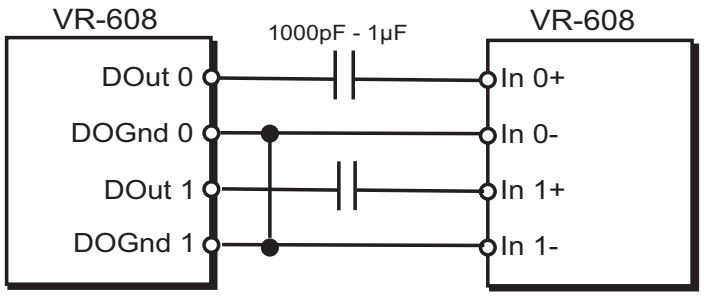

Figure 2-12 Example Wiring for Torque Simulator Loopback

Chapter 3 PowerDNA Explorer

This chapter provides the following information about exploring the DNx-VR-608 with the PowerDNA Explorer application.

- Introduction (Section 3.1)
- VR-608 Configuration (Section 3.2)

3.1 Introduction

PowerDNA Explorer is a GUI-based application for communicating with your RACK or Cube system. You can use it to start exploring a system and individual boards in the system. PowerDNA Explorer can be launched from the Windows startup menu:

Start » All Programs » UEI » PowerDNA » PowerDNA Explorer

The DNx-VR-608 is supported in PowerDNA version 5.2+.

When using PowerDNA Explorer to configure DNx-VR-608 boards, resetting the IOM or changing the DNx-VR-608 configuration outside of PowerDNA Explorer (e.g., via C code or LabVIEW) is not recommended. PowerDNA Explorer will not display changed parameters until Scan Network or Reload Configuration is clicked again.

When configuring the DNx-VR-608 with PowerDNA Explorer, the right-hand panel contains the following tabs:

- **Configuration**: Configures the DNx-VR-608 channels.
- **Initialization**: Configures the initial state of the board at power-up.

Pressing **Reload Configuration** will read the current board state, which is useful if you just restarted PowerDNA Explorer.

3.2 VR-608 Configuration

As shown in Figure 3-1, the Configuration tab in PowerDNA Explorer allows entry of configuration parameters for all DNx-VR-608 channels. The Initialization tab is used for configuration of the state of the board at power-up.

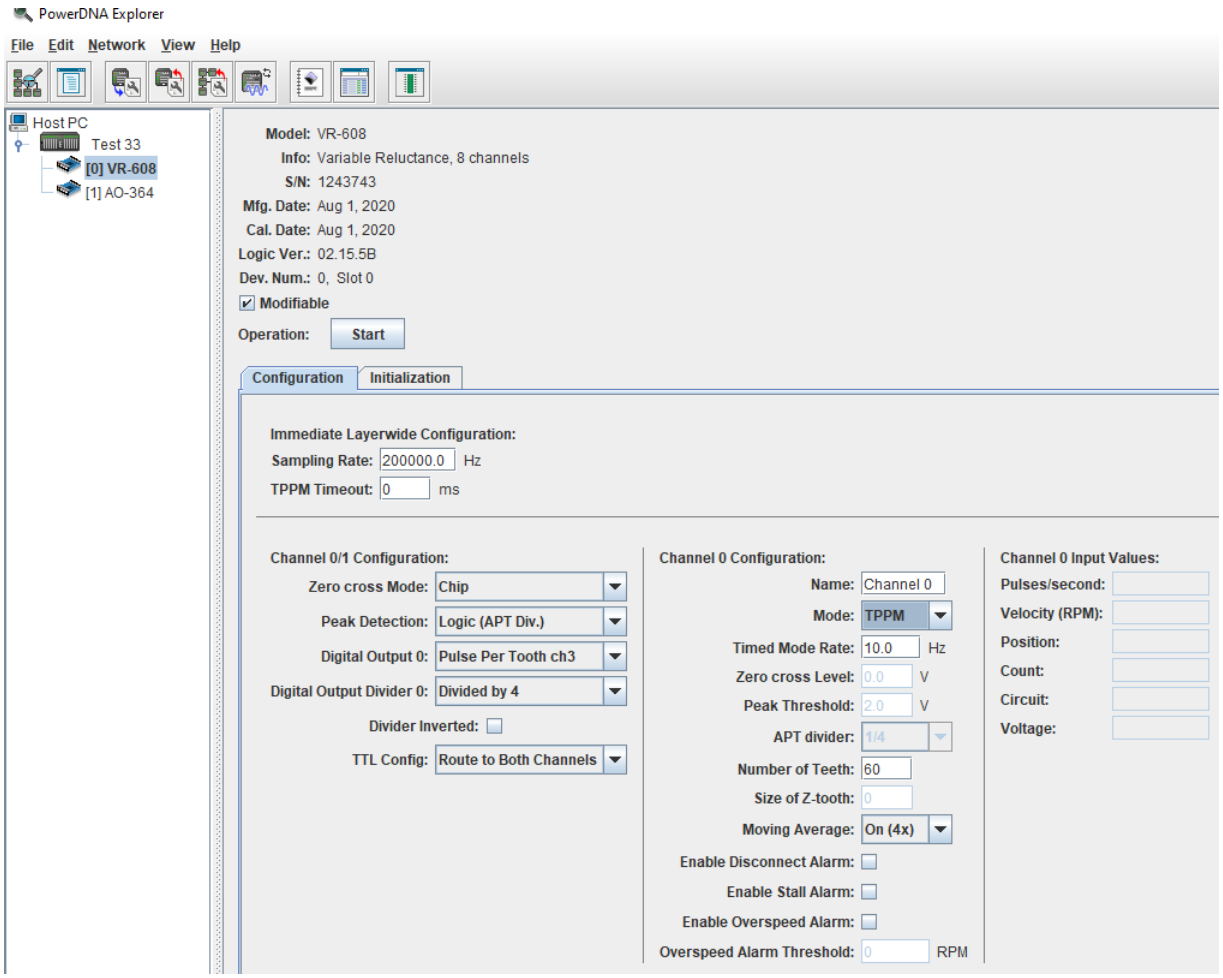

Figure 3-1 PowerDNA Explorer for DNx-VR-608

Chapter 4 Programming with the High-level API

This chapter provides the following information about programming the DNx-VR-608 using the UeiDaq Framework API:

- About the High-level API (Section 4.1)
- Example Code (Section 4.2)
- Create a Session (Section 4.3)
- Resource Strings (Section 4.4)
- Configure VR Channels (Section 4.5)
- Configure Digital Output Channels (Section 4.6)
- Configure Sync Out (Rev. 2) (Section 4.7)
- Configure the Timing (Section 4.8)
- Start the Session (Section 4.9)
- Read Data (Section 4.10)
- Stop the Session (Section 4.11)

4.1 About the High-level API

The UeiDaq Framework is object oriented and its objects can be manipulated in the same manner from different development environments, such as Visual C++, MATLAB, LabVIEW, and more. The Framework is supported in Windows 7 and up. It is generally simpler to use compared to the low-level API, and it includes a generic simulation device to assist in software development. Therefore, we recommend that Windows users use the high-level API unless unconventional functionality is required. Users programming in Linux or a real-time operating system should instead use the Low-level API (Chapter 5).

For more detail regarding the Framework's architecture, please see the *"UeiDaq Framework User Manual"* located under:

Start » All Programs » UEI

For information on the Framework's classes, structures, and constants, please see the *"UeiDaq Framework Reference Manual"* located under:

Start » All Programs » UEI

4.2 Example Code

The UeiDaq Framework is bundled with examples for supported programming languages. The example code is located in language-specific folders under:

C:\Program FIles (x86)\UEI\Framework

For example:

Start » All Programs » UEI » Visual C++ Examples

Examples for the DNx-VR-608 will have "Variable Reluctance" in its title.

Each high-level example follows the same basic structure:

1. Create a session.
2. Configure the session for a particular device and subsystem.
3. Configure the timing.
4. Start the session.
5. Read or write data.
6. Stop the session.

The remainder of this chapter is intended as a supplement to the example code. Examples are presented using C++ API, but the concepts are the same no matter which programming language you use. The "*UeiDaq Framework User Manual*" provides additional information about programming in other languages.

4.3 Create a Session

The session object manages all communications with the DNx-VR-608. Therefore, the first step is always to create a new session.

```
//create a session object
CUeiSession vrSession;
```

4.4 Resource Strings

The Framework uses a resource string to link the session to the hardware. The resource string syntax is similar to a web URL; it should not have any spaces and is case insensitive. The resource string is formatted as follows:

"<device class>://<IP address>/<device number>/<subsystem><channel list>"

- *<device class>* – Set the device class to `pdna`.
- *<IP address>* – IP address of the IOM.
- *<device number>* – position of the DNx-VR-608 within the chassis, relative to the other I/O boards.
- <subsystem> – the DNx-VR-608 uses the `VR` subsystem.
- *<channel list>* – desired channels, 0...7

For example, the following resource strings select variable reluctance channels 0, 1, 2, and 3 on device 2 at IP address 192.168.100.2:

```
"pdna://192.168.100.2/Dev2/Vr0,1,2,3"
"pdna://192.168.100.2/Dev2/Vr0:3"
```

4.5 Configure VR Channels

The `CreateVRChannel()` method adds a VR channel to the session, sets the channel mode, and returns a pointer to the channel.

```
//Configure session to access VR channel 0.
//Configure channel 0 for a Timed measurement.

CUeiVRChannel* vr0 = vrSession.CreateVRChannel(
                                    "pdna://192.168.100.2/Dev1/vr0",
                                    UeiVRModeCounterTimed);
```

The input parameters are:

- `resource` – specifies the device and subsystem (see Section 4.4).
- `mode` – Counter/timer mode (see descriptions in Section 2.2.4.1). One of the following `tUeiVRMode` values.
  - `UeiVRModeDecoder`
  - `UeiVRModeCounterTimed`
  - `UeiVRModeCounterNPulses`
  - `UeiVRModeCounterZPulse`
  - `UeiVRModeCounterTorque`
  - `UeiVRModeCounterTimedPulsePeriodMeasurement`

You can configure additional channel settings by calling the `CUeiVRChannel` methods summarized in **Table 4-1**. Settings apply to either an individual channel (**C**), a pair of channels (**P**), or the entire board/layer (**L**). For example, setting the ZC mode on Channel 0 automatically sets the same mode for Channel 1. Configuration status can be obtained by calling the corresponding "get" or "is" method.

Table 4-1 High-level API for VR Channel Configuration

Method	C/P/L	Rev.1	Rev.2	Description
`SetMode`	C/P	●	●	Select counter/timer mode.
`SetZCMode`	P	●	●	Select zero crossing detection mode.
`SetZCLevel`	C	●	●	Set zero crossing level for `UeiZCModeFixed` (-5 V to +5 V). If the desired zero crossing level is 0 V, UEI recommends setting the desired zero crossing level to a value slightly above 0 to avoid false positives, e.g., 0.1 V.
`SetAPTMode`	P	●	●	Select peak threshold detection mode.
`SetAPTThresholdDivider`	C		●	Set peak threshold divider for `UeiAPTModeLogic` (1...4). Threshold will be a fraction of moving average peak value, where 1 = ½, 2 = ¼, 3 = 1/8th, and 4 = 1/16th.
`SetADCMovingAverage`	P		●	Moving average for `UeiZCModeLogic` and `UeiAPTModeLogic` (0...3, where 0 = off, 1 = 3 samples, 2 = 4 samples, 3 = 8 samples).

Table 4-1 High-level API for VR Channel Configuration (Cont.)

Method	C/P/L	Rev.1	Rev.2	Description
SetNumberOfTeeth	C	●	●	Number of teeth on the wheel for RPM calculation (1...3599). Also used as N in N-pulse mode.
SetZToothSize	C	●	●	Number of missing or fused teeth (0...3).
SetTimedModeRate	C	●	●	Set rate at which data is stored in Timed Mode. (Max 1000 Hz)
SetTorqueLead	P		●	Switch the leading channel in Torque Mode.
InvertTorqueSignal	P		●	Invert the Torque Mode input signal.
SetFIFOMode	C	●	●	Enable FIFO writes with or without time stamps.
SetADCRate	L	●	●	Set analog input rate for the entire board. (Max 320 kHz).
SetSource	C		●	Connect TTL digital input to counter/timer.
EnableStallAlarm	C		●	Enable alarm when no zero crossings are detected at the input (must also set digital output to one of the UeiVRDigitalSourceAlarm enum values).
EnableOpenAlarm	C		●	Enable alarm when sensor is disconnected from input (must also set digital output to one of the UeiVRDigitalSourceAlarm enum values). Does not apply to TTL inputs.
EnableOverspeedAlarm	C		●	Enable alarm when overspeed condition is detected. Uses number of teeth and Z-tooth size to determine if configured RPM is exceeded. (must also set digital output to one of the UeiVRDigitalSourceAlarm enum values).
SetOverspeedLimit	C		●	Set the RPM overspeed limit on the channel.
SetAlarmNoiseLevel	C		●	Set the noise floor level to prevent stall and open circuit alarms from triggering off of a noisy signal. This is scaled to the input signal.
EnableFIRFilter	L		●	Enable high frequency FIR filter (26 taps) with cutoff of 10-15 kHz
SetTimedPulsePerPeri-odModeTimeout	L		●	Set the timeout value (ms) when using TPPM Mode. Extends the default time base.

Refer to the CUeiVRChannel class definition and/or the *"UeiDaq Framework Reference Manual"* for more information about these methods and their accepted input parameters. Some usage examples of these methods are provided in the following subsections of Section 4.5.

4.5.1 Configure Analog VR Sensor

The following example shows a typical configuration for a VR sensor connected to Channel 0.

```
//Configure channel 0 to measure velocity every N teeth.

CUeiVRChannel* vr0 = vrSession.CreateVRChannel(
                                    "pdna://192.168.100.2/Dev2/vr0",
                                    UeiVRModeCounterNPulses);

//Use Dual VR IC for zero crossing detection and adaptive peak threshold.

vr0->SetZCMode(UeiZCModeChip);
vr0->SetAPTMode(UeiAPTModeChip);

//Toothed wheel has N = 36 teeth (including the Z-tooth).
//Z-tooth is one missing tooth.

vr0->SetNumberOfTeeth(36);
vr0->SetZToothSize(1);
```

You can optionally stream channel data into the FIFO with the following call:

```
//Store raw counter data and time stamps into the FIFO.

vr0->SetFIFOMode(UeiFIFOModePosAndTS);
```

4.5.2 Configure Torque Sensor

The following example configures a dual VR torque sensor wired to Channels 2 and 3. For torque sensor configuration, set the odd-numbered channel in a channel pair to Torque mode and the even-numbered channel to N-Pulse Mode.

```
//Channel 2 reports rpm and channel 3 reports torque.

CUeiVRChannel* npulse_ch = vrSession.CreateVRChannel(
                                    "pdna://192.168.100.2/Dev2/vr2",
                                    UeiVRModeCounterNPulses);
CUeiVRChannel* torque_ch = vrSession.CreateVRChannel(
                                    "pdna://192.168.100.2/Dev2/vr3",
                                    UeiVRModeCounterTorque);

//Use Dual VR IC zero crossing detection and adaptive peak threshold.
//ZC and APT are automatically set for channel 2 (npulse_ch).

torque_ch->SetZCMode(UeiZCModeChip);
torque_ch->SetAPTMode(UeiAPTModeChip);

//Odd channel leads even channel. Do not invert inputs.

torque_ch->SetTorqueLead(true);
torque_ch->InvertTorqueSignal(false);

//Store data after 36 teeth have been counted.

npulse_ch->SetNumberOfTeeth(36);
```

4.5.3 Configure Digital Input (Rev. 1)

You can read TTL-level digital pulses through the VR front-end by configuring the board for ADC-based zero crossing detection (i.e., either `UeiZCModeFixed` or `UeiZCModeLogic` mode). The following example uses the Fixed zero crossing mode to read a digital sensor wired to In4+/-.

```
//Configure Channel 4 to count teeth over a specific time interval.

CUeiVRChannel* vr4 = vrSession.CreateVRChannel(
                                    "pdna://192.168.100.2/Dev2/vr4",
                                    UeiVRModeCounterTimed);

//Set the zero crossing level to a positive value, e.g. 0.8 V.

vr4->SetZCMode(UeiZCModeFixed);
vr4->SetZCLevel(0.8);

//Set the APT mode to Logic

vr4->SetAPTMode(UeiAPTModeLogic);

//Store data at a rate of 10 Hz. Always set to less than input_freq/2.

vr4->SetTimedModeRate(10);
```

This functionality is supported on both Rev. 1 and Rev. 2 boards, though we recommend Rev. 2 users route digital signals directly to the counter-timer modules as described in Section 4.5.4

4.5.4 Configure Digital Input (Rev. 2)

On Revision 2 boards, TTL-level digital pulses on DIn can connect directly to the counter-timer modules. When a counter-timer is used for TTL inputs, it is disconnected from the VR input. Peak threshold settings are ignored and while zero crossing detection is bypassed, the zero crossing mode must be set to `ZC_ONCHIP`.

```
//Configure counter-timer 6 to count teeth over a specific time interval.

CUeiVRChannel* vr6 = vrSession.CreateVRChannel(
                                    "pdna://192.168.100.2/Dev2/vr6",
                                    UeiVRModeCounterTimed);

//Set zero crossing mode to ZC_ONCHIP

vr6->SetZCMode(UeiZCModeChip);

//Route DIn0 to counter-timer module on channel 6.

vr6->SetSource(UeiVRSourceTTL0);

//Store data at a rate of 10 Hz. Always set to less than input_freq/2.

vr6->SetTimedModeRate(10);
```

A TTL input can connect to multiple counter-timers if desired.

4.6 Configure Digital Output Channels

The `CreateSimulatedVRChannel()` method adds a digital output channel to the session. The vr0:3 resources correspond to DOut[0:3] respectively. Supported digital output signals are summarized in Section 2.2.5.

4.6.1 Generate Simulated VR Signal

You can configure a digital output channel for VR simulation on Revision 2 boards. The simulated signal is a square wave with a 50% duty cycle and frequency equal to `rpm*ppr/60`. The minimum allowed frequency is 0.12336 Hz and the maximum is 500 kHz. For example, if `ppr`=36, `rpm` can range from 0.2056 to 833,333.

```
//Output a continuous pulse train on DOut0.
//Simulate 500 rpm and 36 teeth on the wheel.

CUeiSimulatedVRChannel* dout0 = vrSession.CreateSimulatedVRChannel(
                                    "pdna://192.168.100.2/Dev2/vr0",
                                    UeiVRSimulatedModeContinuous,
                                    500,
                                    36,
                                    0);
```

The input parameters are:

- `resource` – specifies the device and subsystem (see Section 4.4)
- `mode` – one of the following digital output modes defined in `tUeiVRSimulationMode`:
  - `UeiVRSimulationModeContinuous` – Simulate continuous rotation with no Z-tooth.
  - `UeiVRSimulationModeMissingPulse` – Simulate one missing tooth per revolution.
  - `UeiVRSimulationModeFusedPulses` – Simulate one fused tooth per revolution.
  - `UeiVRSimulationModePhasedPair` – Simulate a torque measurement using a pair of simulator channels. The `vr0 resource` selects the DOut0/1 pair and `vr2` selects the DOut2/3 pair. Note that `vr1` and `vr2` should also be configured as `UeiVRSimulationModePhasedPair` even though the other parameters will be ignored.
  - `UeiVRSimulationModeStatic` – Turn off simulator and use digital output source (Section 4.6.2).
- `rpm` – rotations per minute to simulate
- `ppr` – pulses per revolution to simulate (0...4095, which includes missing/fused pulses in the pulse count)
- `percentDelay` – Only used in `UeiVRSimulationModePhased-Pair`, this is the percent of 1 period that the odd channel is delayed from the even channel.

4.6.2 Route Source to Digital Output

You can connect internal digital signal sources to digital outputs by setting the channel mode to Static. Supported sources are listed in **Table 4-2**.

```
//Output alarm for CUeiVRChannel 4 and 5 on DOut2.

CUeiSimulatedVRChannel* dout2 = vrSession.CreateSimulatedVRChannel(
                                    "pdna://192.168.100.2/Dev2/vr2",
                                    UeiVRSimulatedModeStatic,
                                    0,
                                    0,
                                    0);
dout2->SetDigitalOutputSource(UeiVRDigitalSourceAlarm4_5);
```

Table 4-2 Digital Output Sources for UeiVRSimulatedModeStatic

Digital Output Source	Rev.1	Rev.2	Description
`UeiVRDigitalSourceDisable`	●	●	Disable output.
`UeiVRDigitalSourceForceHigh`	●	●	Drive output high.
`UeiVRDigitalSourceForceLow`	●	●	Drive output low.
`UeiVRDigitalSourceTooth0` ... `UeiVRDigitalSourceTooth7`	●	●	Zero crossings on Channel 0...7.
`UeiVRDigitalSourceZTooth0` ... `UeiVRDigitalSourceZTooth7`	●	●	`ZCModeChip`: Pulses high when a missing or fused tooth is detected. `ZCModeFixed` or `ZCModeLogic`: Pulses high when a 30% taller tooth is detected (Rev. 2 only)
`UeiVRDigitalSourceDirection0_1` ... `UeiVRDigitalSourceDirection6_7`	●	●	Direction of rotation for a quadrature encoder pair.
`UeiVRDigitalSourceTorque0_1` ... `UeiVRDigitalSourceTorque6_7`		●	Torque module output, where the pulse width = time between zero crossings on a channel pair..
`UeiVRDigitalSourceAlarm0_1` ... `UeiVRDigitalSourceAlarm6_7`		●	Open circuit, stalled rotation, and overspeed alarms for a channel pair. (see note after table).
`UeiVRDigitalSourceEndofMea-surement0` ... `UeiVRDigitalSourceEndofMea-surement7`		●	Output pulse when end of measurement period is reached
`UeiVRDigitalSourceSyncLine0` ... `UeiVRDigitalSourceSyncLine3`		●	Output signal from sync line

NOTE: An alarm can only be assigned to DOut on the same channel block, e.g., Alarm0_1 only works on DOut0. The other sources (TorqueX, DirectionX, ZToothX, ToothX) can connect to your choice of one or more DOuts.

4.6.3 Digital Output Dividers

Dividers of 2, 4, 8, or 16 can be applied to signals routed to the digital outputs. The output pulse will always have a 50% duty cycle.
The `CUeiSimulatedVRChannel` class provides methods for setting and returning the divisors as well as setting and returning signal polarity.

Table 4-3 CUeiSimulatedVRChannel Methods for Digital Output Dividers

Method Name	Description
`SetDigitalOutputDivider (int divider)`	Sets the digital output divider value. Specify values as follows: 0 - no division 1 - divide by 2 2 - divide by 4 3 - divide by 8 4 - divide by 16
`GetDigitalOutputDivider()`	Returns the divider value (0..4)
`InvertDigitalOutputPolarity (bool invert)`	Set `invert` to true to invert polarity on digital outputs
`IsDigitalOutputPolarityInverted()`	Returns true if polarity has been inverted, false if default polarity

4.7 Configure Sync Out (Rev. 2)

For DNx-VR-608 Rev. 2 boards, the `CreateVRSyncOutLine()` method adds one or more sync out lines to the VR sync out line list associated with the session. See Section 2.2.6 for more information on sync out signals,

```
//Configuration to enable a sync out line pulse when end of measurement
//period is reached on channel 5.

CUeiVRSyncOutLine* sync0 = vrSession.CreateVRSyncOutLine(
                                    "pdna://192.168.100.2/Dev2/vr0",
                                    UeiVRSyncOutLineEndofMeasurement5);
```

The input parameters are:

- `resource` – specifies the device and subsystem (see Section 4.4)
- `source` – one of the sync out sources defined in `tUeiVRSyncLineOutSource` and described in **Table 4-4**.

Table 4-4 Sync Out Sources Supported by the CUeiVRSyncOutLine Class

Sync Out Source	Description
`UeiVRSyncOutLineDisable`	Disable output
`UeiVRSyncOutLineForceHigh`	Set sync out line to high
`UeiVRSyncOutLineForceLow`	Set sync out line to low

Table 4-4 Sync Out Sources Supported by the CUeiVRSyncOutLine Class (Cont.)

Sync Out Source	Description
UeiVRSyncOutLineTooth0 ... UeiVRSyncOutLineTooth7	Set sync out line to high when any tooth is detected on channel n
UeiVRSyncOutLineZTooth0 ... UeiVRSyncOutLineZTooth7	Set sync out line to high when Z tooth is detected on channel n
UeiVRSyncOutLineDirection0_1 ... UeiVRSyncOutLineDirection6_7	Set sync out line to high when detected quadrature direction is clockwise on the channel pair
UeiVRSyncOutLineTorque0_1 ... UeiVRSyncOutLineTorque6_7	Sync out line pulse whose width is equal to time between zero crossings on the channel pair
UeiVRSyncOutLineEndofMeasurement0 ... UeiVRSyncOutLineEndofMeasurement7	Sync out line pulse when end of measurement period is reached on channel n
UeiVRSyncOutLineSyncLine0 ... UeiVRSyncOutLineSyncLine3	Sync out line signal from sync line n

4.8 Configure the Timing

Only Point-by-Point data acquisition mode can be used to transfer data between a Framework application and the DNx-VR-608.

Point-by-Point mode transfers one sample at a time to/from each configured channel of the I/O board. The delay between samples is controlled by the host application (e.g., by using a Sleep function), thus limiting the data transfer rate to a maximum of 100 Hz. This mode is also known as immediate mode or simple mode.

```
//configure session to use Point-by-Point DAQ mode
vrSession.ConfigureTimingForSimpleIO();
```

4.9 Start the Session

After the session is configured, you can start the session manually:

```
//Start the session.
vrSession.Start();
```

If you don't explicitly start the session, it will automatically start the first time you try to transfer data.

4.10 Read Data

Channels on the VR-608 are read independently. You need to create a reader object for each configured channel:

```
//Create a reader for channel 0 and link it to the session's data stream.
CUeiVRReader reader(vrSession.GetDataStream(), 0);
```

The `CUeiVRReader` object can read the following types of data:

- `Read()` – converted VR sensor data, raw data, FIFO data, or ADC voltage data
- `ReadTorque()` – torque sensor data
- `ReadADCStatus()` – diagnostic data from the ADC
- `ReadAlarmStatus()` – check for triggered alarms on VR input lines
- `ReadDigitalInputStatus()` – read status of digital inputs
- `ReadRevision()` – read DNx-VR-608 revision

4.10.1 Read Velocity, Position, Tooth Count

If you pass a `tUeiVRData` structure into the overloaded `Read()` method, the reader returns a structure (a cluster in LabVIEW) containing the velocity, position, total tooth count since the session started, time stamp, and open/closed circuit flags.

```
//Read one value.

tUeiVRData vrData;
reader.Read(1, &vrData, NULL);
```

You can print the data as follows:

```
//Print VR data and flags.

std::cout << vrData.velocity << std::endl;
std::cout << vrData.position << std::endl;
std::cout << vrData.teethCount << std::endl;
std::cout << vrData.timestamp << std::endl;

if (vrData.flags==3)
   std::cout << "Open Circuit" << std::endl;
else if (vrData.flags==2)
   std::cout << "Closed Circuit" << std::endl;
else
   std::cout << "no data available" << std::endl;
```

Some of the structure members will not have data in every mode. When there is no data, they will be set to 0. This is summarized in **Table 4-5**.

Table 4-5 VR Data returned by Read()

Mode	Velocity	Position	Teeth Count
Timed	●	●	●
Z-Pulse	●	●	●
N-Pulse	●		
Quadrature Decoder		●	●
Torque			
TPPM	●		

4.10.2 Read Torque (Rev. 2)

The `ReadTorque()` method returns the torque ratio from a channel configured for Torque Mode. See Section 4.5.2 for information on torque sensor configuration. The other channel in the pair, which is configured for N-Pulse Mode, is read using the `Read()` method described in Section 4.10.1. The N-Pulse channel returns RPM.

```
//Read one value.

double torqueData
reader.ReadTorque(1, &torqueData, NULL);

//Print torque data.

std::cout << "% torque:" << torqueData << std::endl;
```

4.10.3 Read FIFO Data

If FIFO mode is enabled, you can read count register data from the FIFO with the following call:

```
//Request 10 uint32's from the FIFO.

uInt32 fifoData[10];
reader.Read(10, fifoData, &numVals);
```

Table 4-6 summarizes what events will cause data to be stored in the FIFO for each mode. The data will consist of various counter/timer registers and optionally, the time stamp. Each returned parameter takes up one uInt32, and all are returned in each call.

Table 4-6 Data read from FIFO

Mode	Event	Data[0]	Data[1]	Data[2]	Data[3]
Timed	At TimedModeRate	CRH	CRR in [31:16] CRL in [15:0]	CR	[TS]
Z-Pulse	Z-tooth found	CRH	CRL	CR	[TS]
N-Pulse	N-teeth encountered	CRH	CRL	[TS]	
Quadrature Decoder	At TimedModeRate	CR	CRR	[TS]	
Torque	N-teeth encountered	CRH	CRL	[TS]	

4.10.4 Read ADC Diagnostics

You can read ADC voltage data for a channel pair. Data is returned for the even and odd channels simultaneously. For example, a channel 0 reader will return interleaved channel 0 and channel 1 data.

```
//Read raw ADC data from channel pair.

double adcData[fifosize*2];
reader.Read(fifosize, &adcData, &numVals);

//Print ADC FIFO data.

for (i=0; i < numVals; i++){
   std::cout << "Even: " << adcData[k*2];
   std::cout << ", Odd: " << adcData[k*2+1] << std::endl;
}
```

NOTE: The data returned is not calibrated (<1% error in hardware) and there is no guarantee that it is gapless.

You can also read the ADC module's zero crossing detection status:

```
//Read ADC status, used as a diagnostic to troubleshoot VR inputs

uInt32 sts;
reader.ReadADCStatus(&sts);
```

The status data is sticky; after you read the status, the status bits are cleared for the pair. Therefore, you should only call `ReadADCStatus()` for either the even or odd channel to avoid losing your data. Status bits are mapped as shown in **Table 4-7**.

Table 4-7 ADC Status Register

Bit	Description
20	Rising edge zero crossing was detected on the odd channel.
19	Falling edge zero crossing was detected on the odd channel.
18	Rising edge zero crossing was detected on the even channel.
17	Falling edge zero crossing was detected on the even channel.
16	Data is ready for both channels in the pair.
15...0	reserved

4.10.5 Read Status Information

Additional status information can be read by using the `CUeiVRReader` methods listed in **Table 4-8**.

Table 4-8 CUeiVRReader Status Methods

Method	Description
`ReadAlarmStatus()`	Returns 32-bit status word showing alarm status on VR input lines for open circuit, stalled rotation, and overspeed alarms. See **Table 4-9**. Note that status will be returned for all alarms regardless if the alarm has been configured.
`ReadDigitalInputStatus()`	Shows routing of TTL input bits to timer/counters for the VR input channel.
`ReadRevision()`	Returns the revision of the VR-608 board. 0 for Rev. 1 2 for Rev. 2

The bit assignments for the 32-bit status word returned by `ReadAlarmStatus()` are listed in **Table 4-9**.

Table 4-9 Bit Assignments for ReadAlarmStatus()

Bit	Description
31-30	RSV - Reserved
29	OSPD7 - overspeed detected on sensor 7
28	OSPD6 - overspeed detected on sensor 6
27	OPEN7 - Sensor 7 detected as open
26	OPEN6 - Sensor 6 detected as open
25	STAL7 - Stall detected on sensor 7
24	STAL6 - Stall detected on sensor 6
23-22	RSV - Reserved
21	OSPD5 - overspeed detected on sensor 5
20	OSPD4 - overspeed detected on sensor 4
19	OPEN5 - Sensor 5 detected as open
18	OPEN4 - Sensor 4 detected as open
17	STAL5 - Stall detected on sensor 5
16	STAL4 - Stall detected on sensor 4
15-14	RSV - Reserved
13	OSPD3 - overspeed detected on sensor 3
12	OSPD2 - overspeed detected on sensor 2
11	OPEN3 - Sensor 3 detected as open
10	OPEN2 - Sensor 2 detected as open
9	STAL3 - Stall detected on sensor 3
8	STAL2 - Stall detected on sensor 2
7-6	RSV - Reserved
5	OSPD1 - overspeed detected on sensor 1
4	OSPD0 - overspeed detected on sensor 0
3	OPEN1 - Sensor 1 detected as open
2	OPEN0 - Sensor 0 detected as open
1	STAL1 - Stall detected on sensor 1
0	STAL0 - Stall detected on sensor 0

4.11 Stop the Session

The session will automatically stop and clean itself up when the session object goes out of scope or when it is destroyed. To manually stop the session:

```
//Stop the session.
vrSession.Stop();
```

To reuse the object with a different set of channels or parameters, you can manually clean up the session as follows:

```
//clean up session and free resources
vrSession.CleanUp();
```

Chapter 5 Programming with the Low-level API

This chapter provides the following information about programming the DNx-VR-608 using low-level API:

- About the Low-level API (Section 5.1)
- Example Code (Section 5.2)
- Data Acquisition Modes (Section 5.3)
- Point-by-Point API (Section 5.4)
- RtDMap API (Section 5.5)

5.1 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, we recommend that you use the UeiDaq high-level Framework API (see Chapter 4). The Framework simplifies the Low-level API, making programming easier and faster while still providing access to the majority of Low-level API features. Additionally, the Framework supports a variety of programming languages and the use of scientific software packages such as LabVIEW and MATLAB.

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

- On Linux: *<PowerDNA-x.y.z>/docs*
- On Windows: *C:\Program Files (x86)\UEI\PowerDNA\Documentation*

NOTE: The DNx-VR-608 is supported in PowerDNA version 5.2+. If you're unsure if your version supports the board, please contact Technical Support at support@ueidaq.com

5.2 Example Code

Application developers are encouraged to explore the self-documented source code examples to get started programming UEI products. The example code is located in the following directories:

- On Linux: *<PowerDNA-x.y.z>/src/DAQLib_Samples*
- On Windows: *C:\Program Files (x86)\UEI\PowerDNA\SDK\Examples*

The I/O board number is embedded in the name of the example code. For example, the Sample608 folder contains example code specific to the DNx-VR-608. The example code should run out of the box after inputting the IOM's IP address and the board's Device Number (DEVN).

5.3 Data Acquisition Modes

The following data acquisition (DAQ) mode is available for transferring data between the DNx-VR-608 and the low-level user application:

- **Point-by-Point:** Transfers one data point at a time to/from each configured channel of a single I/O board. Timing is controlled by the user application, which limits the transfer rate to 100Hz. This mode is also known as immediate mode or simple mode.
- **Real-Time Data Map (RtDMap):** Transfers a packet containing one data point for each channel in the user-defined map. The newest data is transferred and old data is discarded. RtDMap is designed for closed-loop (control) applications and may include channels across multiple I/O boards.

Please refer to *"FAQ - Data Acquisition Modes"* for an overview and comparison of all the different acquisition modes offered by UEI. The *"PowerDNx Protocol Manual"* includes more detailed information about the protocols. Both of these documents are located in the directories listed in Section 5.1.

5.4 Point-by-Point API

Table 5-1 summarizes the low-level API used to configure, read from, and write to the DNx-VR-608 in Point-by-Point DAQ mode.

Table 5-1 Low-level DNx-VR-608 API Functions

Function	Description
`DqAdv608SetCfg`	Configure a VR-608 Rev.1 board.
`DqAdv608GetCfg`	Get the presently configured parameters for a Rev.1 board.
`DqAdv608SetCfgExt`	Configure a VR-608 Rev. 2 board.
`DqAdv608GetCfgExt`	Get the presently configured parameters for a Rev.2 board.
`DqAdv608SetADCClock`	Set the ADC rate for the VR-608 device.
`DqAdv608ConvertRawData`	Converts raw counter data to velocity, position, and tooth count.
`DqAdv608Read`	Reads position, velocity, tooth count, time stamp inputs.
`DqAdv608ReadADCStatus`	Reads ADC status, most recent calibrated voltages, and min/max voltages.
`DqAdv608ReadADCFifo`	Read raw ADC data stored in the FIFO and enable ADC writes to the FIFO.
`DqAdv608ReadFifo`	Read VR channel data from the FIFO.
`DqAdv608ReadStatus`	Reads VR alarm, TTL input routing, and revision information.
`DqAdv608SetSensorSim`	Configure VR simulator outputs.
`DqAdv608SetWatermark`	Set the buffer watermark level to control dataflow from the IOM.
`DqAdv608Enable`	Start or stop a VR-608 device.

The functions and parameters are described in detail in the *"PowerDNA API Reference Manual"*. See *Sample608.c* for a comprehensive example which includes typical initialization, error handling, and usage of these functions.

The tutorial in this section is intended as a supplement to the example code and the API reference manual.

5.4.1 Configuration

To configure a Revision 1 board, populate `DQ_VR608_CFG` structure using the `DqAdv608SetCfg()` function. To configure a Revision 2 board, populate a `DQ_VR608_CFG_R2` structure via `DqAdv608SetCfgExt()`.

DNx-VR-608 configuration parameters apply to either an individual channel (**C**), a pair of channels (**P**), or the entire board/layer (**L**). The parameters, their application scope (C/P/L), and supported revision(s) are listed in **Table 5-2**. Some parameters take in a single value and some accept a logically grouped combination of constants. Please refer to the *"PowerDNA API Reference Manual"* for a complete description of each parameter.

Table 5-2 Low-level API Configuration Parameters

Parameter	C/P/L	Rev.1	Rev.2	Description
`cfg_flags`	L	●	●	OR in flags to change associated parameters
`adc_rate`	L	●	●	Analog input rate for the board
`tmode_rate`	C	●	●	Rate at which data is stored in Timed Mode
`zc_level`	C	●	●	Zero crossing level for ZC_FIXED. If the desired zero crossing level is 0 V, UEI recommends setting `zc_level` to a value slightly above 0 to avoid false positives, e.g., 0.1 V.
`apt_div`	C		●	Peak threshold divider for APT_LOGIC
`adc_mv_avg`	P	●	●	Moving average for ZC_LOGIC and APT_LOGIC
`sync_out`	L		●	Output signals on Sync [3:0]
`d_out`	L	●	●	Output signals on DOut [3:0]
`num_teeth`	C	●	●	Number of teeth on the wheel for RPM calculation
`z_tooth_sz`	C	●	●	Number of missing or fused teeth
`alarm_src`	L		●	Configure open circuit, stalled rotation, and overspeed alarms. Open circuit alarms do not apply to TTL inputs.
`torque_cfg`	P		●	Torque sensor polarity and rotation direction
`ttl_cfg`	L		●	Route DIn[3:0] to internal counters
`stall_lvl`	C		●	Set noise floor level for the stall alarm. The value is a point on the input range. Input values below this point are ignored
`dout_div`	L		●	Configure dividers on digital outputs
`tppm_timeout`	L		●	Set TPPM timeout in 10 ms increments

Table 5-2 Low-level API Configuration Parameters (Cont.)

Parameter	C/P/L	Rev.1	Rev.2	Description
ov_speed	C		●	Set limit for overspeed alarm in RPM

Note that cfg_flags defines what other parts of the configuration structure are valid. For example, the DQ_VR608_CFGVLD_ZC_LEVEL flag is required to set zc_level. If DQ_VR608_CFGVLD_ZC_LEVEL is not included in cfg_flags, then the zero crossing value is ignored and remains unchanged. Parameters that have not been configured are configured with default values.

By using this strategy, configuration calls can be additive, so each following call adds or changes a parameter to the card's configuration (or uses the default value if not configured). To reset the entire configuration back to the default state, call DqAdv608SetCfg() or DqAdv608SetCfgExt() with only the DQ_VR608_CFGFLG_CLEAR bit set in cfg_flags.

The examples in the following sections use the |= operator in order to update the existing configuration with the new values shown in the examples.

5.4.1.1 VR Sensor Configuration

The following example shows a typical configuration for an analog VR sensor wired to Channel 0.

```
//Use Dual VR IC for zero crossing detection and adaptive peak threshold.

front_cfg = DQ_VR608_ZC_ONCHIP|DQ_VR608_APT_ONCHIP;

//Measure velocity every N teeth.

mode = DQ_VR608_MODE_COUNTER|DQ_VR608_MODE_NPULSE;

//Toothed wheel has N = 36 teeth (including the Z-tooth).
//Z-tooth is one missing tooth.

extcfg.cfg_flags |=
   DQ_VR608_CFGVLD_NUM_TEETH |
   DQ_VR608_CFGVLD_Z_TOOTH_SZ;
extcfg.num_teeth = 36;
extcfg.z_tooth_sz = 1;

//Program configuration for Channel 0.

DqAdv608SetCfg(hd, DEVN, 0, front_cfg, mode, &extcfg);
```

Remember that front_cfg will be set for the channel pair, i.e., both Channels 0 and 1. Timed Mode is only set for Channel 0.

5.4.1.2 Torque Sensor Configuration (Rev. 2)

The following example configures a dual VR torque sensor wired to Channels 2 and 3. For torque sensor configuration, always set the odd channel in a channel pair to Torque Mode and the even channel to N-Pulse Mode.

```
//Use Dual VR IC zero crossing detection and adaptive peak threshold.

front_cfg = DQ_VR608_ZC_ONCHIP|DQ_VR608_APT_ONCHIP;

//Channel 3 reports torque and Channel 2 reports rpm.

mode[3] = DQ_VR608_MODE_COUNTER|DQ_VR608_MODE_TORQUE;
mode[2] = DQ_VR608_MODE_COUNTER|DQ_VR608_MODE_NPULSE;

//Even channel leads odd channel
//Store data after 36 teeth have been counted.

extcfg.cfg_flags |=
   DQ_VR608_CFGVLD_TORQUE_CFG |
   DQ_VR608_CFGVLD_NUM_TEETH;
extcfg.torque_cfg = 0;
extcfg.num_teeth = 36;

//Program VR-608 Rev.2 Channels 2 and 3.

DqAdv608SetCfgExt(hd, DEVN, 2, front_cfg, mode[2], &extcfg);
DqAdv608SetCfgExt(hd, DEVN, 3, front_cfg, mode[3], &extcfg);
```

5.4.1.3 Digital Input Configuration (Rev. 1)

You can read TTL-level digital pulses through the VR front-end by configuring the board for ADC-based zero crossing detection (i.e., either `ZC_FIXED` or `ZC_LOGIC` mode). The following example uses `ZC_FIXED` mode to read a digital sensor wired to In4+/-:

```
//Use a fixed zero crossing level and peak threshold.

front_cfg = DQ_VR608_ZC_FIXED | DQ_VR608_APT_LOGIC;

//Count teeth over a specific time interval.

mode = DQ_VR608_MODE_COUNTER|DQ_VR608_MODE_TIMED;

//Store data at a rate of 10 Hz. Always set to less than input_freq/2.
//Set the zero crossing level to a positive value, e.g. 0.8 V.
//Set the apt divider (exponent) to 1 - divides by 2^1

extcfg.cfg_flags |=
   DQ_VR608_CFGVLD_TMODE_RATE |
   DQ_VR608_CFGVLD_ZC_LEVEL |
   DQ_VR608_CFGVLD_APT_DIV;
extcfg.tmode_rate = 10;
extcfg.zc_level = 0.8;
extcfg.apt_div = 1;

//Program configuration for Channel 4.

DqAdv608SetCfg(hd, DEVN, 4, front_cfg, mode, &extcfg);
```

This functionality is supported on both Rev. 1 and Rev. 2 boards, though we recommend Rev. 2 users route digital signals directly to the counter-timer modules as described in Section 5.4.1.4.

5.4.1.4 Digital Input Configuration (Rev. 2)

On Revision 2 boards, TTL-level digital pulses on DIn can connect directly to the counter-timer modules. When a counter-timer is used for TTL inputs, it is disconnected from the VR input and ignores `front_cfg` settings. However, while zero crossing detection is bypassed, zero crossing mode must be set to `ZC_ONCHIP`.

```
//Count teeth over a specific time interval. For higher rates (over
//10 kHz) an FIR filter can be enabled by ORing in
//DQ_VR608_FLAG_CH_FIR_ENABLE,

mode = DQ_VR608_MODE_COUNTER|DQ_VR608_MODE_TIMED;

//Store data at a rate of 10 Hz. Always set to less than input_freq/2.
//Route DIn0 to counter-timer module on channel 6.

extcfg.cfg_flags |=
   DQ_VR608_CFGVLD_TMODE_RATE |
   DQ_VR608_CFGVLD_TTL_CFG;
extcfg.tmode_rate = 10;
extcfg.ttl_cfg = DQ_VR608_TTL_SRC(1, 6, 0);

//Set zero crossing mode to ZC_ONCHIP

front_cfg = DQ_VR608_ZC_ONCHIP

//Program mode and tmode_rate on channel 6.
//ttl_cfg is programmed for the entire layer.

DqAdv608SetCfgExt(hd, DEVN, 6, front_cfg, mode, &extcfg);
```

5.4.1.5 Alarm Configuration (Rev. 2)

On Revision 2 boards, you can configure digital outputs as any combination of open circuit, stalled rotation, or overspeed alarms. An alarm can only be assigned to DOut on the same channel block, e.g., Alarm0 to DOut0.

Use the `stall_lvl` member of the `DQ_VR608_CFG_R2` structure to set a noise floor for open/stalled alarms. Use `ov_speed` to specify RPM for overspeed alarms.

```
//Set DOut0 to pulse high when input is open on Ch0 and/or Ch1.

extcfg.cfg_flags |=
   DQ_VR608_CFGVLD_ALARM_SRC |
   DQ_VR608_CFGVLD_D_OUT;
extcfg.alarm_src =
   DQ_VR608_CFG_ALARM0_OPEN0 |
   DQ_VR608_CFG_ALARM0_OPEN1;
extcfg.d_out = DQ_VR608_DBG_N(
   DQ_VR608_CFG_DO_DBG_OFF,
   DQ_VR608_CFG_DO_DBG_OFF,
   DQ_VR608_CFG_DO_DBG_OFF,
   DQ_VR608_CFG_DO_ALARM0);
```

5.4.1.6 Digital Output Dividers (Rev. 2)

On Revision 2 boards, dividers can be independently applied to the digital outputs. Dividers of 2, 4, 8, or 16 are supported. The output pulse will always have a 50% duty cycle. To set divider values, use the `DQ_VR608_DOUT_DIV` macro to populate the `dout_div` member of the `DQ_VR608_CFG_R2` structure. The macro takes arguments for each of the digital output lines (D3...D0) in the form of one of the `DQ_VR608_DIVSEL` values. The digital output signals can also be inverted. Please refer to the *"PowerDNA API Reference Manual"* for a complete list of supported divider values.

```
//Divide by 4 on DOut3, Divide by 2 on DOut2,
//Divide by 2 and invert output on DOut1, no division on DOut0

extcfg.cfg_flags |=
   DQ_VR608_CFGVLD_DO_DIV |
   DQ_VR608_CFGVLD_D_OUT;
extcfg.dout_div = DQ_VR608_DOUT_DIV(
   DQ_VR608_DIVSEL_4,
   DQ_VR608_DIVSEL_2,
   DQ_VR608_DIVSEL_2_I,
   DQ_VR608_DIVSEL_1);
```

5.4.1.7 Sync Out (Rev. 2)

On Revision 2 boards, use the `sync_out` member of the `DQ_VR608_CFG_R2` structure to configure the output signals on the SYNC bus. The `DQ_VR608_SYNC` macro takes arguments for each of the sync out lines (D3...D0). Signal selection works the same as for digital outputs.

```
//Set Sync 2 to pulse on channel 5 end of measurement

extcfg.sync_out = DQ_VR608_SYNC(
   DQ_VR608_CFG_SYNC_OFF,
   DQ_VR608_CFG_DO_EM5,
   DQ_VR608_CFG_SYNC_OFF,
   DQ_VR608_CFG_SYNC_OFF);
```

5.4.1.8 VR Simulator Configuration (Rev. 2)

Digital outputs can be used to generate simulated VR signals on Rev.2 boards. A simulator can only be assigned to its corresponding DOut, e.g., Sim0 to DOut0.

```
//Connect DOut0 to virtual simulator channel 0.

extcfg.cfg_flags |=
   DQ_VR608_CFGVLD_D_OUT;
extcfg.d_out = DQ_VR608_DBG_N(
   DQ_VR608_CFG_DO_DBG_OFF,
   DQ_VR608_CFG_DO_DBG_OFF,
   DQ_VR608_CFG_DO_DBG_OFF,
   DQ_VR608_CFG_DO_SIG_SIM0);

//Program VR-608 rev.2 and enable operations.

DqAdv608SetCfgExt(hd, DEVN, 0, front_cfg, mode, &extcfg);
DqAdv608Enable(hd, DEVN, 0xFF);

//Configure the simulator after layer is enabled.
//Simulate 2000 rpm, 36 ppr, and one missing tooth on
//simulator channel 1.

DqAdv608SetSensorSim(hd, DEVN, 1, 2000, 36, DQ_VR608_SIM_Z_PULSE, 0);
```

5.4.2 Enable Channels

To verify that parameters were set as desired, read back the configuration as:

```
// Get the configuration for each channel on a VR-608 rev.1

DqAdv608GetCfg(hd, DEVN, chnl, &front_cfg, &mode, &extcfg);
```

Once all desired channels have been configured, enable operation with a mask:

```
// Enable all eight channels.

DqAdv608Enable(hd, DEVN, 0xFF);
```

The VR-608 is now in operating mode.

There is no benefit to enabling or disabling specific channels, so we recommend enabling all eight channels with 0xFF.

5.4.3 Read Inputs

To read the latest input data, call:

```
// Read input data from the channel list.

DqAdv608Read(hd, DEVN, sizeof(ch_list)/sizeof(uint32), ch_list, NULL,
   NULL, data);
```

to obtain a `VR608_READ_DATA` structure for all channels in the `ch_list` array. The `VR608_READ_DATA` structure will contain converted VR data (Section 2.2.4.2) and optional time stamp and status data. Refer to the *"PowerDNA API Reference Manual"* for more information about the returned data.

Note that unconnected channels will still produce data if queried.

Raw data is returned in `bin_data`. For example:

```
// Read raw data.

DqAdv608Read(hd, DEVN, sizeof(ch_list)/sizeof(uint32), ch_list,
   bin_data, bin_data_ptr, data);
```

Information about returned raw data can be found in the *"PowerDNA API Reference Manual"* and **Table 2-2** of this document.

5.4.3.1 Read FIFO Data

You can enable FIFO writes for an individual channel when setting its mode:

```
//Measure velocity every N teeth.
//Store raw counter data and time stamp into FIFO.

mode = DQ_VR608_MODE_COUNTER|DQ_VR608_MODE_NPULSE|
   DQ_VR608_FIFO_POS | DQ_VR608_FIFO_TS;
```

After configuring and enabling the channel, use `DqAdv608ReadFifo()` to read its FIFO data. The FIFO contains the raw counter register data (Section 2.2.4.1). Actual data size returned is limited to network packet size, so multiple calls must be made from the host application to retrieve the entire FIFO buffer.

```
//Request the maximum number of data points from the FIFO for
//VR channel 0 on a Rev. 2 board.

DqAdv608ReadFifo(hd, DEVN, 0, DQ_VR608_FIFOSZ_R2, &size, fifo_data);
```

5.4.3.2 Read Diagnostic Data

For debugging purposes, it is possible to directly read ADC voltage and status information. ADC data is always returned for both channels in a pair. Actual data size returned is limited to network packet size, so multiple calls must be made from the host application to retrieve the entire FIFO buffer.

```
//Enable ADC writes to FIFO and read the maximum number of data points
//for a Rev. 2 board.
//data_a and data_b contain channel 0 and 1 voltage respectively
//First, clear the FIFO data by passing in 0 for sizemax

DqAdv608ReadADCFifo(hd, DEVN, 0, 0, &size, &avail, data_a, data_b, NULL);
DqAdv608ReadADCFifo(hd, DEVN, 0, DQ_VR608_FIFOSZ_R2, &size, &avail,
   data_a, data_b, NULL);

//Return status data structure for channels 0 and 1.

DQ_VR608_ADC_STS ain_sts;
DqAdv608ReadADCStatus(hd, DEVN, 0, &ain_sts);
```

5.4.4 Read Status

Information for alarm status, status for the digital inputs, and revision information for the VR-608 board can be obtained by calling `DqAdv608ReadStatus()`.

```
// Read Status.

DqAdv608ReadStatus(hd, devn, pRdSts);
```

`pRdSts` is a pointer to a `DQ_VR608_RD_STS` struct that includes the following members:

- `alarm` - provides alarm status See **Table 4-9** for bit assignments.
- `ttl` - shows routing of TTL inputs.
- `rev2` - returns 2 for Rev. 2 DNx-VR-608 boards or 0 for Rev. 1 boards.

5.4.5 Stop Cleanly

To stop operation, call `DqAdv608Enable()` with a FALSE parameter:

```
// Disable operation.

DqAdv608Enable(hd, DEVN, 0x00);
```

This stops operation of the DNx-VR-608 board without changing the configuration.

5.5 RtDMap API

Real Time Data Map (RtDMap) mode uses the same API as Point-by-Point mode for channel configuration (Section 5.4); however, generic DMap functions are used for reading data. The DMap API is documented in the *"PowerDNA API Reference Manual"*.

Refer to *SampleRTDMap608* for an example of how to set up a Data Map and read data from DNx-VR-608 input channels. The available DNx-VR-608 data channels are shown in **Table 5-3**.

Table 5-3 DMap Channels

Channel	Description
DQ_VR608_CHNLTYPE_CRR	Raw counter/timer data for a VR input channel. See DqAdv608Read() in the "*PowerDNA API Reference Manual*" for a description of the raw data returned in each input mode. **Table 2-2** in this document also describes Count Register Data for each mode.
DQ_VR608_CHNLTYPE_CR	
DQ_VR608_CHNLTYPE_CRH	
DQ_VR608_CHNLTYPE_CRL	
DQ_VR608_CHNLTYPE_TSTAMP	Time stamp in 10 µs increments
DQ_VR608_CHNLTYPE_STATUS	reserved
DQ_VR608_CHNLTYPE_AINSTS	Edge detection status for a channel pair. See DqAdv608ReadADCStatus() for a description of the status register bits (DQ_VR608_ADC_STS.status).
DQ_VR608_CHNLTYPE_ALARMSTS	Status of all open circuit, stalled rotation, and overspeed alarms. Bits are defined as: 31-30 RSV - Reserved 29 OSPD7 - overspeed detected on sensor 7 28 OSPD6 - overspeed detected on sensor 6 27 OPEN7 - Sensor 7 detected as open 26 OPEN6 - Sensor 6 detected as open 25 STAL7 - Stall detected on sensor 7 24 STAL6 - Stall detected on sensor 6 23-22 RSV - Reserved 21 OSPD5 - overspeed detected on sensor 5 20 OSPD4 - overspeed detected on sensor 4 19 OPEN5 - Sensor 5 detected as open 18 OPEN4 - Sensor 4 detected as open 17 STAL5 - Stall detected on sensor 5 16 STAL4 - Stall detected on sensor 4 15-14 RSV - Reserved 13 OSPD3 - overspeed detected on sensor 3 12 OSPD2 - overspeed detected on sensor 2 11 OPEN3 - Sensor 3 detected as open 10 OPEN2 - Sensor 2 detected as open 9 STAL3 - Stall detected on sensor 3 8 STAL2 - Stall detected on sensor 2 7-6 RSV - Reserved 5 OSPD1 - overspeed detected on sensor 1 4 OSPD0 - overspeed detected on sensor 0 3 OPEN1 - Sensor 1 detected as open 2 OPEN0 - Sensor 0 detected as open 1 STAL1 - Stall detected on sensor 1 0 STAL0 - Stall detected on sensor 0

A DMap holds one data point per configured channel. One copy of the map is stored on the IOM and another is stored on the host. The IOM updates its version of the map at the `DMAP_RATE` specified during initialization, and the ADCs are clocked at this rate. Calling `DqRtDmapRefresh()` transfers the latest input data from the IOM to the host; the data can then be read from the host map using `DqRtDmapReadRawData32()`. For more information on RtDMap, see the "*PowerDNx Protocol Manual*."

Appendix A

Accessories

A.1 Cables

DNA-CBL-37

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

DNA-CBL-37S

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

A.2 Screw Terminal Panels

DNA-STP-37

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

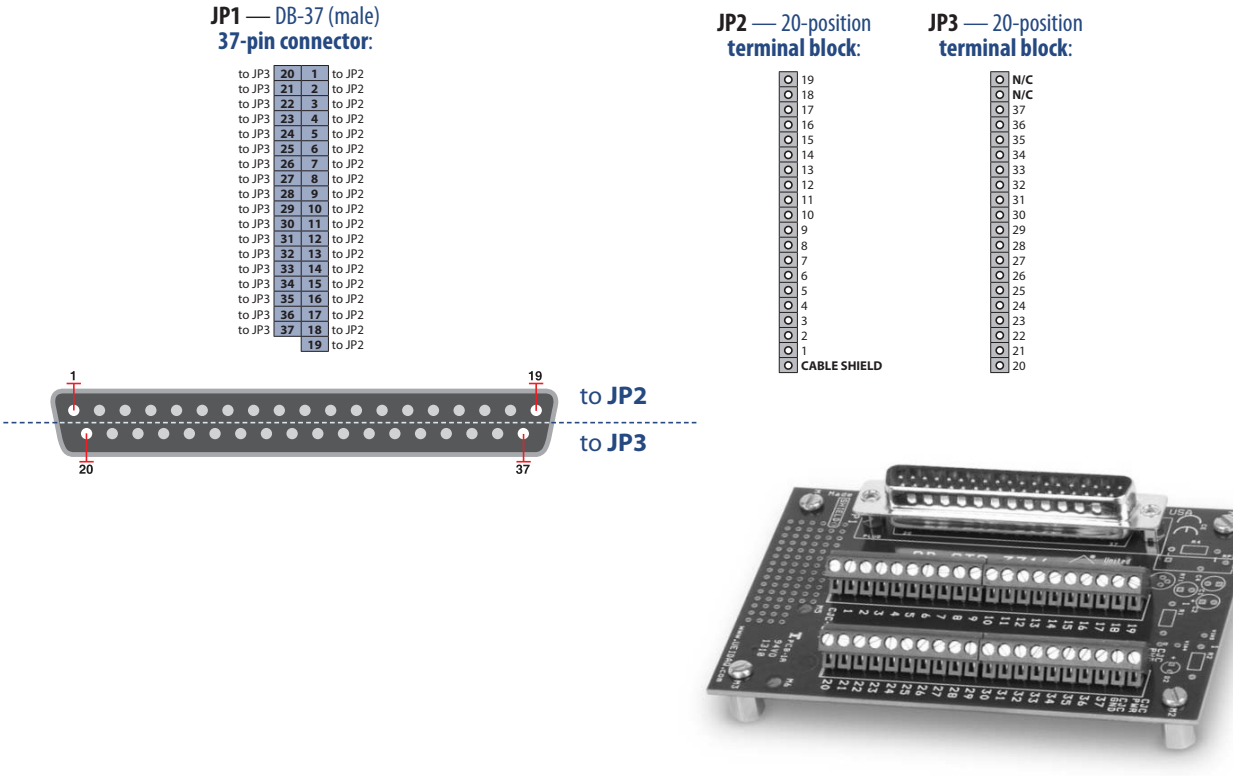

Figure A-1 Pinout and Photo of DNA-STP-37 Screw Terminal Panel