United Electronic Industries

The High-Performance Alternative

UEIModbus User Manual 3.6.0

September 2019 Edition

United Electronic Industries
The High-Performance Alternative

Table of contents

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

1. Introduction

MODBUS is a messaging protocol developed by Modicon in 1979 and used to establish master-slave/client-server communication between intelligent devices. It is a de facto standard, truly open, and the most widely used network protocol in the industrial manufacturing environment. (Specifications available at www.modbus.org/specs.php).

MODBUS devices communicate using a master-slave technique in which only one device (the master) can initiate transactions (called queries). The other devices (slaves) respond by supplying the requested data to the master or by taking the action requested in the query.

A slave is any peripheral device (I/O transducer, valve, network drive, or other measuring device), which processes information and sends its output to the master using MODBUS.

Masters can address individual slaves or can initiate a broadcast message to all slaves. Slaves return a response to all queries addressed to them individually but do not respond to broadcast queries.

UEIModbus extends the capability of the PowerDNA distributed data acquisition system by turning it into a MODBUS/TCP slave that can be accessed by any software client that can act as a MODBUS/TCP master. Most popular HMI software packages support the MODBUS/TCP protocol.

1.1. Register tables

The MODBUS specification defines four register tables (or register maps) in which the input or output data for I/O devices can be read or written.

Register Table	Classic Name	Description	Data Representation
0xxxx	Coils (Read/Write)	A 0x reference address is used for single bit output data to be driven out by a digital output channel.	1 bit
1xxxx	Discrete Inputs (Read-only)	A 1x reference address holds the ON/OFF status of the corresponding digital input channel.	1 bit
3xxxx	Input Registers (Read-only)	A 3x reference register stores a 16-bit value received from an external source, such as an analog input signal.	16 bits
4xxxx	Holding Registers (Read/Write)	A 4x reference register is used to store 16-bits of numerical data (binary or decimal) that is sent to an output channel.	16 bits

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

For each of the four register tables, the MODBUS protocol allows a maximum of 65536 data items to be accessed. It is slave dependent, in which data items are accessible by a master.

Typically, a slave implements only a small memory area.

UEIModbus only implements memory areas big enough to hold the number of channels used on each board layer.

1.2. Mapping PowerDNA I/O boards to MODBUS register tables

By default, PowerDNA I/O boards are mapped to MODBUS register tables by associating the MODBUS register addresses for a board to the board position in the chassis. The first/top PowerDNA I/O board in a chassis is referenced as device number 0, the next is device number 1, and so on. This device number is used to calculate the address of the MODBUS registers associated with each layer by multiplying it by 1000, (i.e., the third board in a chassis is identified as device number 2; therefore, 2000 will be the starting address in MODBUS register space for this device).

Diagnostic data can also be accessed via MODBUS registers. PowerDNA I/O boards that are designated as Guardian series boards provide access to internal diagnostic data and map additional MODBUS registers for the diagnostic feedback, such as for internal voltages, temperatures, open/short circuit status and more. Diagnostic registers are mapped at an offset of 500 relative to the base address calculated from the device position.

PowerDNA Power and NIC boards also provide access to internal diagnostic data. Power and NIC boards are mapped to 12000 and 13000 respectively.

Example of addressing using register mapping relative to the board position in chassis:

Device 0 MODBUS registers will start at address 0
Device 0 diagnostic registers will start at address 500
Device 1 MODBUS registers will start at address 1000
Device 1 diagnostic registers will start at address 1500
…
Device x MODBUS registers will start at address 1000*x
Device x diagnostic registers will start at address 1000*x+500
…
Power board diagnostic registers will start at address 12000
NIC board diagnostic registers will start at address 13000

Alternatively, users can customize how PowerDNA I/O boards are mapped to Input Registers or Holding Registers by entering an address offset (**StartOffset**) in a control field when configuring the board.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

NOTE: Programming the StartOffset is described in section 2.6, “Configuring I/O channels” on page 18.

Example of customized addressing using register mapping programmed with a user-defined StartOffset:
(In this example device 0 is programmed with an offset of 0, and device 1 is programmed with an offset of 124, and device x is programmed with an offset of 300):

Device 0 MODBUS registers will start at address 0
Device 0 diagnostic registers will start at address 500

Device 1 MODBUS Input/Holding registers will start at address 124
Device 1 MODBUS Coil/Discrete Input registers will start at address 1000
Device 1 diagnostic registers will start at address 1500
…
Device x MODBUS Input/Holding registers will start at address 300
Device x MODBUS Coil/Discrete Input registers will start at address 1000*x
Device x diagnostic registers will start at address 1000*x+500
…
Power board diagnostic registers will start at address 12000
NIC board diagnostic registers will start at address 13000

NOTE: The MODBUS protocol uses zero-based addresses. In actual MODBUS register space, Device 0 will map to MODBUS address 0, as listed above. However, the convention for referencing registers in MODBUS address space adds +1 to the actual address, (i.e., the Device 1 MODBUS register may be addressed as 1001 in client/server register maps).

1.2.1. Data type representations in MODBUS registers

Coils and Discrete Input registers only represent one bit. They are used to hold the digital output and input line state of single bit digital devices.

Input and Holding registers are 16 bits. Each register can represent the raw data value for a digital port or an analog channel for devices with a resolution less than or equal to 16 bits.

To represent values greater than 16 bits, additional Input or Holding registers are needed:

- Two MODBUS registers are required to represent a scaled value of analog channels using 32-bit IEEE floating point data representations.
- Four MODBUS registers are required to represent a scaled value of analog channels using 64-bit IEEE double precision floating point data representations.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

1.2.2. Digital I/O device representation

Digital I/O devices are always mapped twice:

- Each bit line is mapped in the Coil (for output) or Discrete Input (for input) tables.
- Each port is mapped in the Holding Register (for output) or Input Register (for input) tables.

The following is an example of a 12-bit digital input board at device 2. Note that in this example the Holding Register mapping is based on board position (using default register mapping/not set with a customized start offset):

Discrete Input Table	Physical Channel
2001	Input port 0 line 0
2002	Input port 0 line 1
2003	Input port 0 line 2
…	…
2011	Input port 0 line 10
2012	Input port 0 line 11

Input Register Table	Physical Port
2001	Input port 0

The following is the same as above but with device 2 set with a customized start address offset of 64:

Discrete Input Table	Physical Channel
2001	Input port 0 line 0
2002	Input port 0 line 1
2003	Input port 0 line 2
…	…
2011	Input port 0 line 10
2012	Input port 0 line 11

Input Register Table	Physical Port
65	Input port 0

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

1.2.3. Digital I/O device with diagnostics representation

Digital I/O devices are mapped as bit lines in the Coil (for output) or Discrete Input (as input) tables and as a port in the Holding Register (for output) or Input Register (for input) tables, as described above in section 1.2.2 above.

Digital I/O boards that support Guardian diagnostics additionally provide access to diagnostic data, which maps in the Input Register table.

The following is an example of a 32-bit Guardian digital output board at device 2 that provides digitized diagnostic voltage and current values for each digital output channel (diagnostic values are simple precision floating point values: 32-bit values require 2 16-bit registers per channel). Note that in this example the Holding Register mapping is based on board position (not set with a customized register addressing):

Coil Table	Physical Channel
2001	Output line 0
2002	Output line 1
2003	Output line 2
…	…
2031	Output line 30
2032	Output line 31

Input Register Table	DIO Diagnostics
2501	Output line 0 diagnostic current
2503	Output line 0 diagnostic voltage
2505	Output line 1 diagnostic current
2507	Output line 1 diagnostic voltage
…	…
2625	Output line 31 diagnostic current
2627	Output line 31 diagnostic voltage

Holding Register Table	Physical Port
2001	Output port 0

NOTE: The diagnostic readback parameters that are available for each DIO board are dependent on the functionality of that board. The example above shows readback values for the current and voltage per channel for a digital output board; however, digital input boards may only provide the digitized input voltage, while other digital I/O boards may also provide temperature readings and other diagnostic values.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

1.2.4. Analog I/O device representation

Analog input devices are mapped in the Input Register table and the Holding Register table.

Analog output devices are mapped in the Holding Register table.

The following is an example for a 16-channel analog input board at device 3 and configured to acquire data as simple precision floating point values (32-bit values require 2 16-bit registers per channel). Note that in this example and the next example the Input Register and Holding Register mapping is based on board position (not set with a customized register addressing):

Input Register Table	Holding Register Table	Physical Channel
3001	3001	Channel 0
3003	3003	Channel 1
…	…	…
3031	3031	Channel 15

The following example is for an 8-channel analog output board at device 5 configured to use double precision floating point values (64-bit values require 4 16-bit registers per channel):

Holding Register Table	Physical Channel
5001	Channel 0
5005	Channel 1
…	…
5025	Channel 6
5029	Channel 7

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

1.2.5. Analog output devices with diagnostics representation

Analog output devices are mapped in the Holding Register table, as described above in section 1.2.4. Analog output boards that support Guardian diagnostic features additionally provide access to diagnostic data, which maps in the Input Register table.

The following example is for a Guardian 32-channel analog output board at device 5 configured to use double precision floating point values (64-bit values require 4 16-bit registers per channel). Note that in this example and the next example the Input Register and Holding Register mapping is based on board position (not set with a customized register addressing).
Guardian readback values include a digitized diagnostic voltage reading for each output channel (diagnostic readings are configured as simple precision floating point values: 32-bit values require 2 16-bit registers per channel):

Input Register Table	Physical Channel / Diagnostics
5501	Channel 0 (diagnostic voltage)
5503	Channel 1 (diagnostic voltage)
5505	Channel 2 (diagnostic voltage)
…	…
5561	Channel 30 (diagnostic voltage)
5563	Channel 31 (diagnostic voltage)

Holding Register Table	Physical Channel
5001	Channel 0
5005	Channel 1
…	…
5121	Channel 30
5125	Channel 31

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

1.2.6. Power/NIC boards diagnostic representation

Power and NIC boards installed in UEIModbus models based on the 8347 CPU (UEIMODBUS 300/600-1G, UEIMODBUS 600/1200R, or -MIL models) provide access to diagnostic data. Diagnostic data from the power and NIC boards map to the Input Register table and Holding Register table.

UEIModbus models based on the 5200 CPU (UEIMODBUS 300/600) do not support reading diagnostic data from power and NIC boards.

Available diagnostic data includes voltage readings (in Volts), current readings (in Amps), and temperature readings (in Kelvins). Values are configured as floating point numbers (32-bit values require 2 16-bit registers per channel):

Input Register Table	Holding Register Table	Power Diagnostics
12001	12001	2.5 V supply for slots 1-3
12003	12003	2.5 V supply for slots 4-6
12005	12005	3.3 V supply for slots 1-3
12007	12007	3.3 V supply for slots 4-6
12009	12009	24 V supply for slots 1-3
12011	12011	24 V supply for slots 4-6
12013	12013	Input Voltage
12015	12015	1.5 V supply for slots 1-6
12017	12017	1.2 V supply for slots 1-6
12019	12019	Fan Voltage
12021	12021	Input Current
12023	12023	Temperature 1
12025	12025	Temperature 2

Input Register Table	Holding Register Table	NIC Diagnostics
13001	13001	2.5 V supply for slots 4-6
13003	13003	Ground reference
13005	13005	3.3 V supply for slots 4-6
13007	13007	24 V supply for slots 4-6
13009	13009	1.5 V supply for slots 1-6
13011	13011	1.2 V supply for slots 1-6
13013	13013	3.3 V current for slots 4-6
13015	13015	Temperature 1
13017	13017	Temperature 2

United Electronic Industries
The High-Performance Alternative

1.2.7. View register tables

A list of Modbus registers used by each I/O channel can be viewed in UEI's web user interface (web UI). The UEI-supplied web interface for configuring UEIModbus systems is described in detail in section 2 starting on page 11.

Once the web interface is configured and open, to view register tables, you select the **Register Map** tab, and then click the **Coils**, **Discrete Inputs**, **Input Registers**, or **Holding Registers** tab.

The following shows the mapping when register addressing defaults to the board position in chassis (device 0 starts at 0, device 1 starts at 1000, device 2 starts at 2000…):

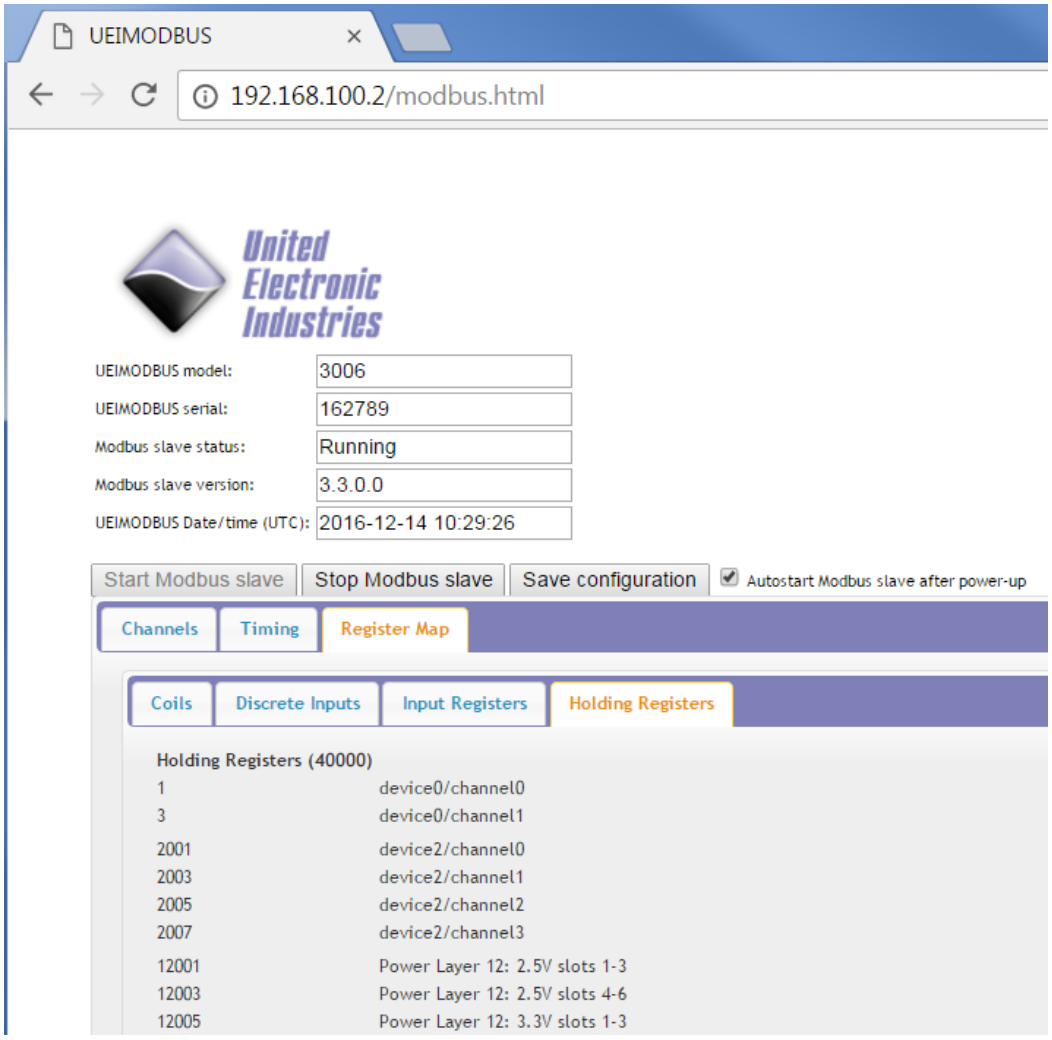

Figure 1 Register Map Display for UEIModbus Web UI

www.ueidaq.com
508.921.4600

The following shows customized mapping. Register addressing for device 8 is user-programmed with a start offset of “172”, and register addressing for device 9 is based on board position in chassis (device 9 starts at 9000):

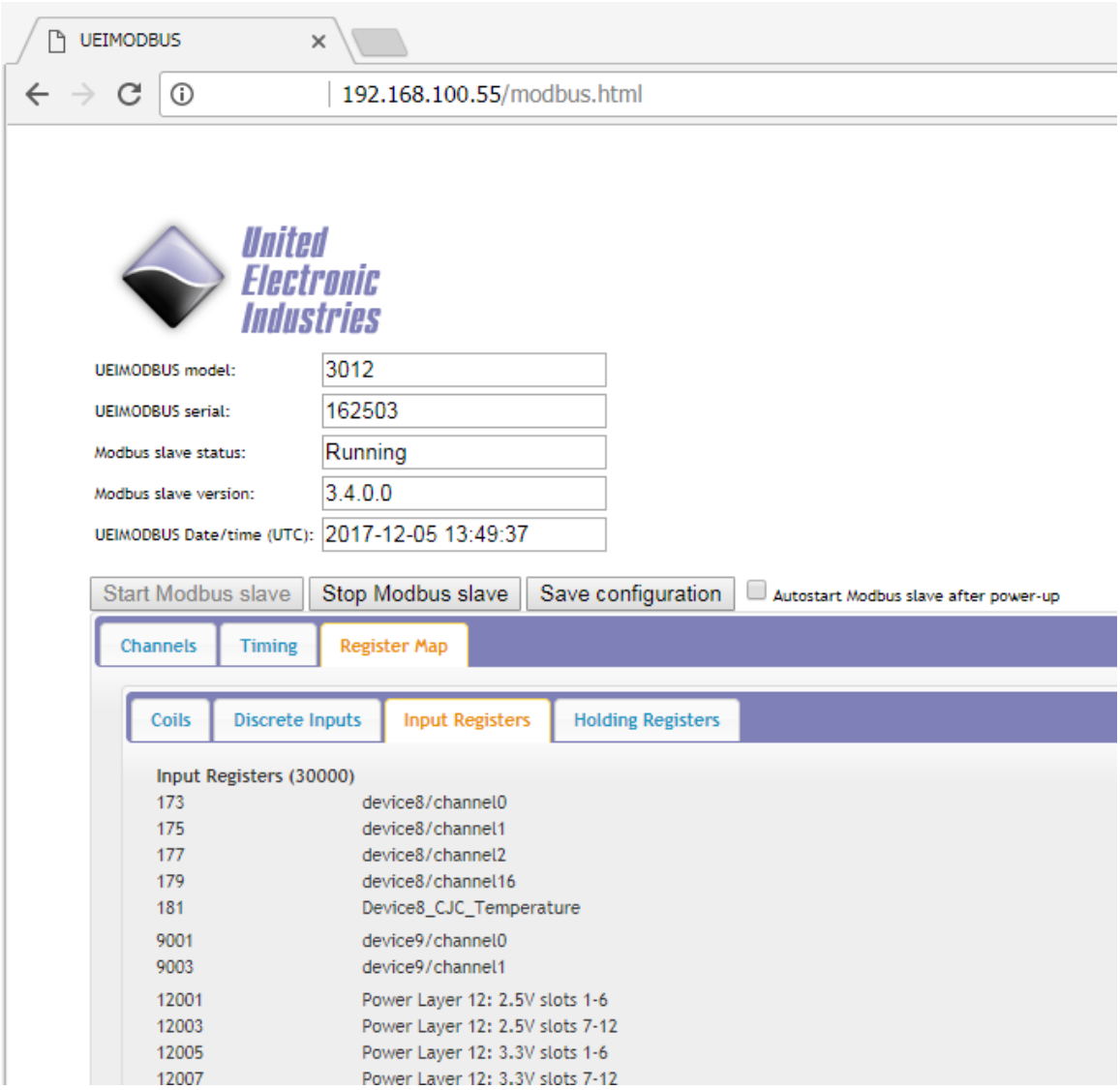

Figure 2 Register Map Display for UEIModbus Addressing with Custom Start Offsets

NOTE: UEIModbus updates the register map each time the Modbus slave server is started. Click **Stop Modbus slave** and then **Start Modbus slave** to implement and view changes that affect the register map, (e.g., after enabling or disabling channels)**.**

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2. Configuring UEIModbus

The UEIModbus configuration consists of configuring the IP address and the I/O channels you wish to make available to MODBUS/TCP masters.

The IP address must be configured using the serial port (sections 2.1 and 2.2).

The I/O channels are configured in the “/etc/modbusslave.conf” file, located on the root file system (SD card) (section 2.6).

Channels can be configured using the UEIModbus web UI or by editing the “/etc/modbusslave.conf” file manually at a command prompt (accessed via the serial port or by remote login to the UEIModbus using ftp or ssh).

2.1. Connecting through the serial port

The following section describes the procedure to connect the UEIModbus chassis (Cube or RACK) to your host PC.

1. Connect the serial cable between the serial port on the UEIModbus chassis and the serial port on your PC.

   You will need a serial communication program:
   - Windows: ucon, MTTTY or HyperTerminal.
   - Linux: minicom or cu (part of the uucp package).

2. Run your serial terminal program and configure the serial communication settings as follows: 57600 bits/s, 8 data bits, 1 stop bit, and no parity.

3. Connect the DC output of the power supply (24 VDC) to the “Power In” connector on the UEIModbus, and connect the AC input on the power supply to an AC power source.

Once power is connected, you should see boot up messages similar to the following in your serial terminal session window:

```
U-Boot 1.1.4 (Jan 10 2016 - 19:20:03)

CPU:   MPC5200 v1.2 at 396 MHz
       Bus 132 MHz, IPB 66 MHz, PCI 33 MHz

Board: UEI PowerDNA MPC5200 Layer
I2C:   85 kHz, ready
DRAM:  128 MB
Reserving 349k for U-Boot at: 07fa8000
FLASH:  4 MB
In:    serial
Out:   serial
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

```
Err:   serial
Net:   FEC ETHERNET

Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  5

## Booting image at ffc10000 ...
 Image Name:   Linux-2.6.16.1
 Created:      2006-11-10  16:07:06 UTC
 Image Type:   PowerPC Linux Kernel Image(gzip compressed)
   Data Size:    917636 Bytes = 896.1 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
id mach(): done
...
< lots of kernel messages >
...
BusyBox v1.2.2 (2006.11.03-19:16+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ #
```

After booting completes, the Linux prompt (#) will be available in your serial terminal window, allowing you to navigate the file system and enter standard Linux commands, such as `ls`, `ps`, and `cd`.

2.2. Configuring the IP address

Your UEIModbus Cube is configured at the factory with the IP address 192.168.100.2 to be part of a private network.

You can change the IP address for the current session using the command:

```
setip <new IP address>
```

The `setip` command changes the current IP address and stores it in a file so that the new IP address will be used next time the system is powered-up.

To test that the IP address is correctly set, type the `ping` command from the command prompt of your host PC.

2.3. Opening the web UI

Once the IP address is configured and confirmed, you can disconnect the serial port and configure the UEIModbus via the web interface by opening a web browser with the following URL:

`http://<UEIModbus IP address>/modbus.html`

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

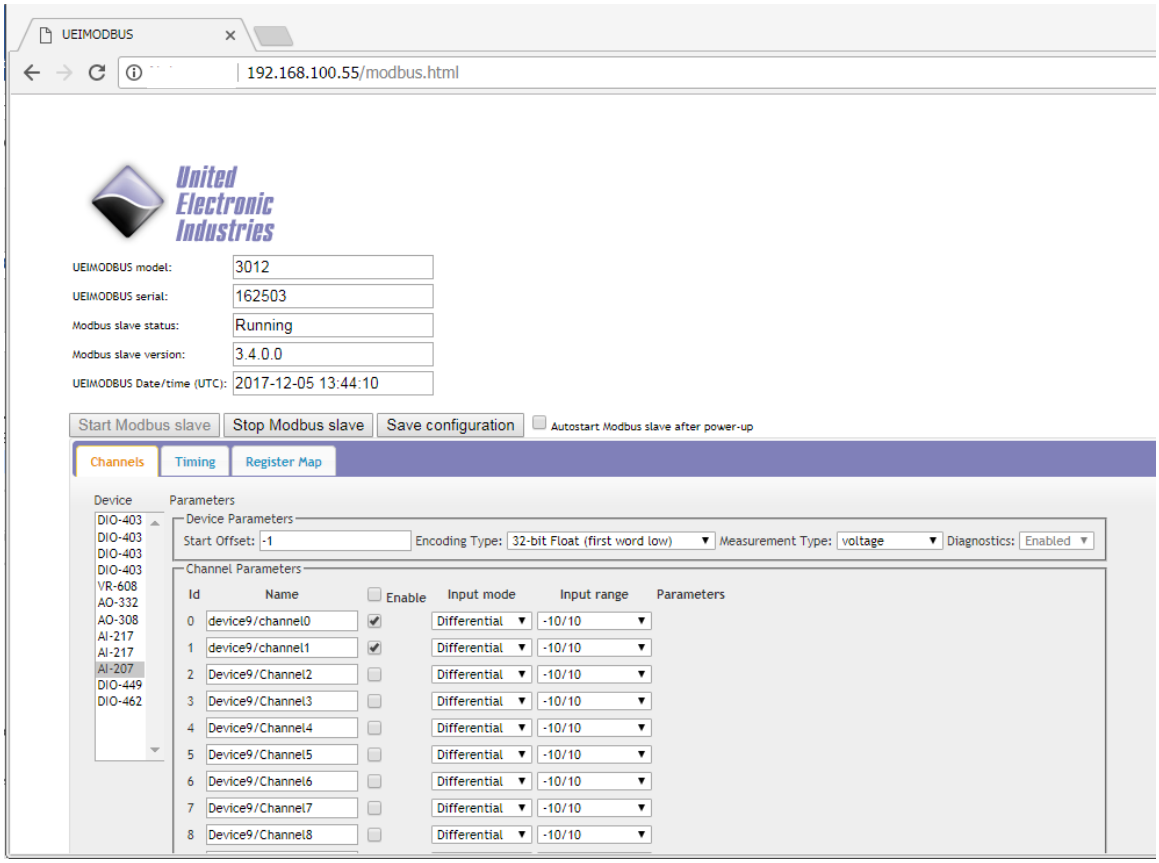

Figure 3 UEIModbus Web UI

We recommend using a Firefox, Google Chrome, or Safari browser to access the UEIModbus web UI. Microsoft Internet Explorer is not supported.

2.4. Starting/Stopping MODBUS slave service

MODBUS/TCP requests from MODBUS masters are handled by the MODBUS slave service.

The UEIModbus is pre-configured to automatically start the MODBUS slave service at boot time.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.4.1. Using the web UI to start/stop the slave service

To start the MODBUS slave server, click the **Start Modbus slave** button. The status indicator should show that the server is in the “Running” state.

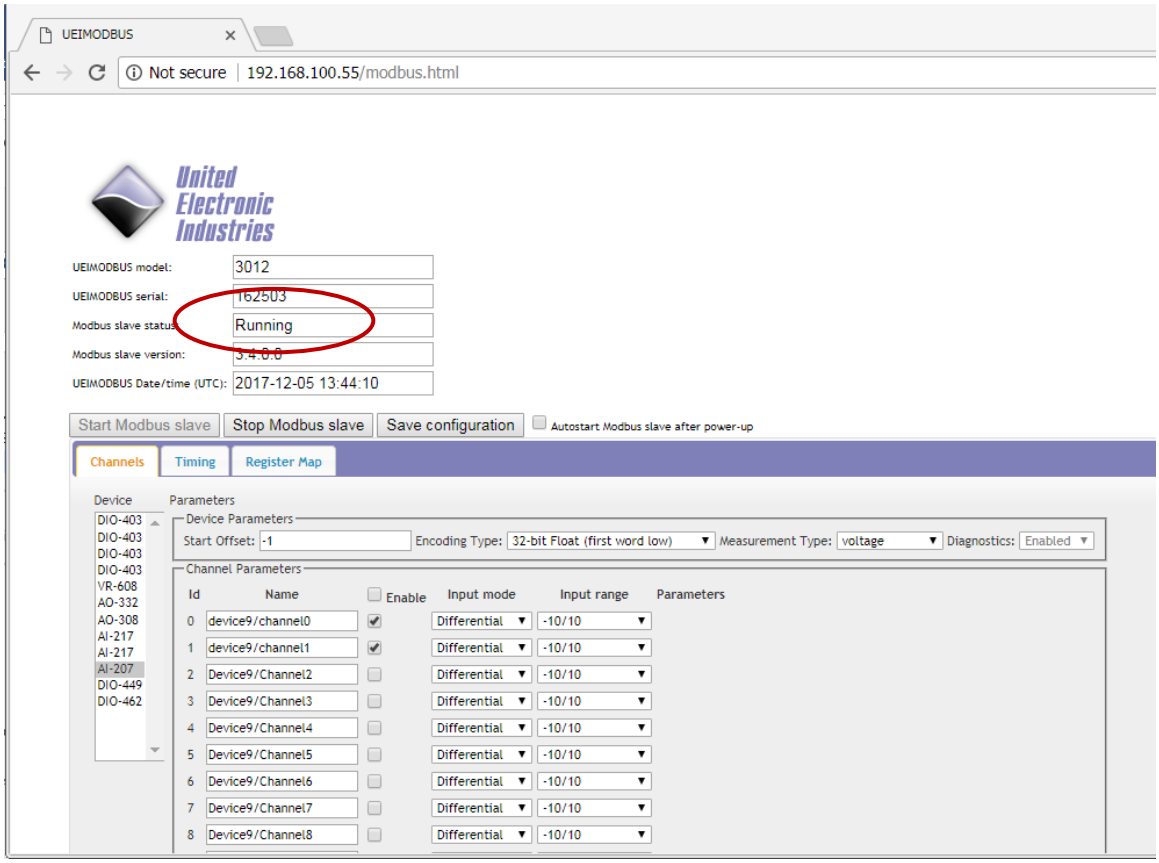

Figure 4 UEIModbus Slave Server Running

Click the **Stop Modbus slave** button to stop the slave server. The Modbus slave status will read “Stopped” when the slave server is not running.

Check **Autostart Modbus slave after power-up,** and click **Save configuration** to automatically start the UEIModbus slave service at boot time.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.4.2. Using the command line to start/stop the slave service

Use the following command to stop the MODBUS slave service:

```
/etc/init.d/modbusslave stop
```

Use the following command to start the MODBUS slave service:

```
/etc/init.d/modbusslave start
```

Use the following command to restart the MODBUS slave service:

```
/etc/init.d/modbusslave restart
```

Use the following command to disable automatic start of MODBUS slave service:

```
mv /etc/rc.d/S40modbusslave /etc/rc.d/K40modbusslave
```

Use the following command to enable automatic start of MODBUS slave service:

```
mv /etc/rc.d/K40modbusslave /etc/rc.d/S40modbusslave
```

2.5. Configuring the date and time

The UEIModbus on-board real-time clock (RTC) is used to produce the timestamp for each data point. The RTC is backed by a system battery.

You can either set the date and time manually, or you can configure the UEIModbus to automatically synchronize its clock with an NTP server over the network.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.5.1. Synchronizing time and date with the NTP server

2.5.1.1. Using the web UI to synchronize with NTP server

This feature is not implemented on the web UI.

2.5.1.2. Using the command line to synchronize with NTP server

To synchronize the UEIModbus with the NTP server, first verify that you can `ping` your NTP server.

Edit the file "/etc/init.d/ntp" and set the variable **NTP_SERVER** to the IP address or host name of your NTP server.

Use the following command to stop the NTP service:

```
/etc/init.d/ntp stop
```

Use the following command to start the NTP service:

```
/etc/init.d/ntp start
```

Use the following command to restart the NTP service:

```
/etc/init.d/ntp restart
```

Use the following commands to disable automatic start of NTP service:

```
mv /etc/rc.d/S50ntp /etc/rc.d/K50ntp
```

Use the following commands to enable automatic start of NTP service:

```
mv /etc/rc.d/K50ntp /etc/rc.d/S50ntp
```

2.5.2. Setting local date and time

The UEIModbus is equipped with a real-time clock (RTC) chip that preserves the date and time settings when the UEIModbus is not powered.

By default, the date is set to the current date and time in the UTC (GMT) time zone.

You can use the web UI or the command line to set the date and time.

2.5.2.1. Using the web UI to set local date and time

The **UEIMODBUS Date/time (UTC)** field displays the current UTC date and time programmed on the UEIModbus on-board RTC. If the display does not refresh automatically, refresh your web browser to update.

Under the **Timing** tab, the **New Date (UTC)** displays the current UTC date and **New Time (UTC)** displays the current UTC time (Figure 5 below).

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

To manually set a new time or date, update the **New Date (UTC)** and/or **New Time (UTC)** field/s, and click **Set UTC date/time and save** to save the new date and time to the UEIModbus RTC.

To set the time and date to the same values as your host system, click **Set UTC date/time from host and save** to align the UEIModbus RTC date and time with your host PC.

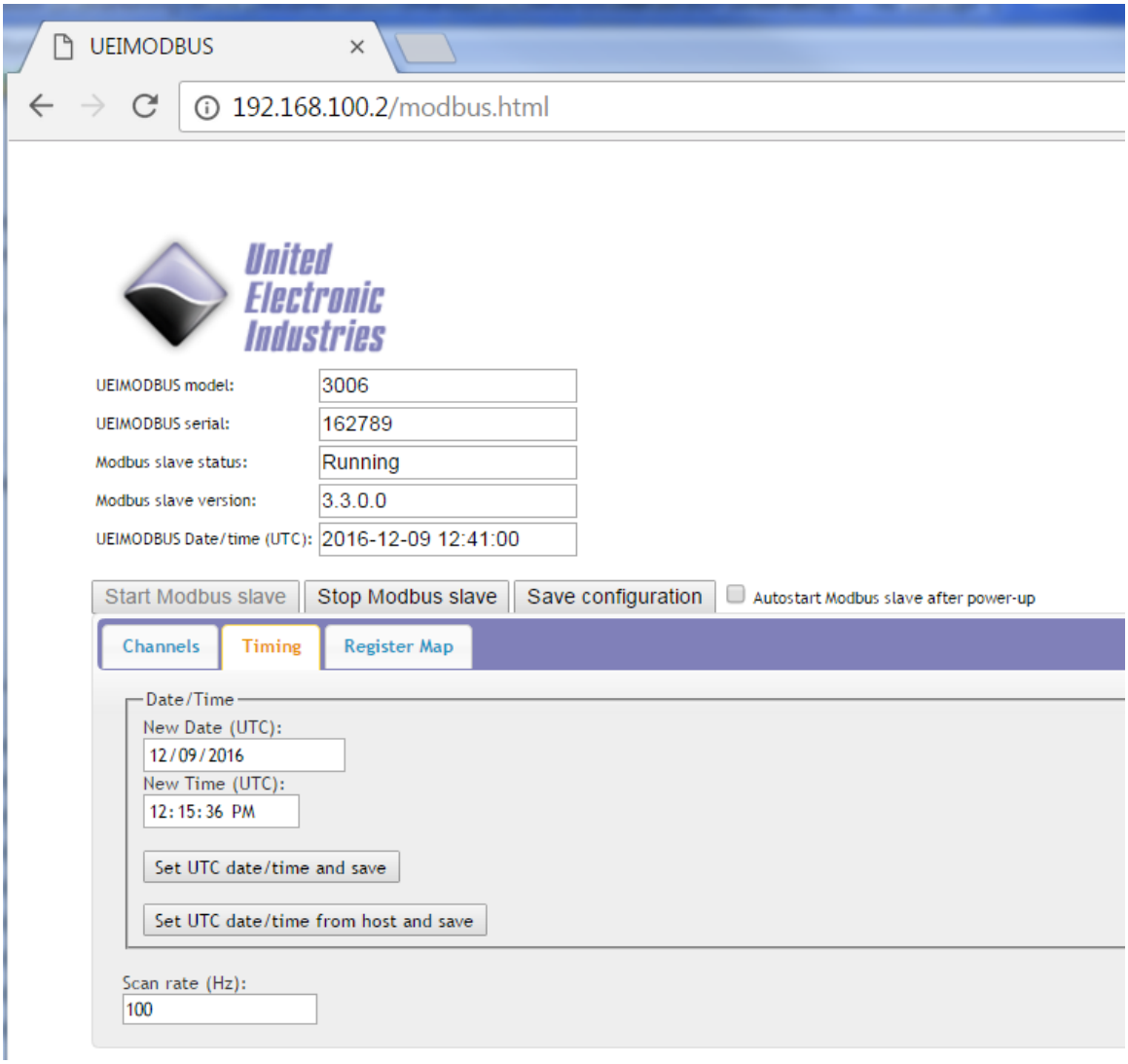

Figure 5 Setting Time and Date via the UEIModbus Web UI

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.5.2.2. Using the command line to set local date and time

To print the current date and time, use the following command:

```
date
```

To change the current date and time, use either of the following commands:

```
date -s MMDDhhmm
date -s YYYYMMDDhhmm.ss
```

For example, "`date -s 06021405`" will set the new date to June second, 2:05 PM.

To make this change permanent upon reboot, save the date to the RTC chip with the following command:

```
hwclock -w -u
```

2.5.3. Changing the time zone

To set the time zone, you need to set the environment variable **TZ**.

For example, the command below sets the time zone to eastern time with daylight saving time starting on the Sunday(0) of the second week(2) of March(3) and ending on Sunday(0) of the first week(1) of November(11):

```
export TZ=EST5EDT,M3.2.0,M11.1.0
```

To make this change permanent upon reboot, add the command to the "/etc/profile" file.

You can find a detailed explanation of the syntax for **TZ** at:
http://www.gnu.org/software/libtool/manual/libc/TZ-Variable.html

2.6. Configuring I/O channels

Configuration parameters for UEIModbus I/O channels are stored in the "/etc/modbusslave.conf" file.

Configuring channels using the UEIModbus web UI (section 2.6.1) updates the "/etc/modbusslave.conf" automatically, or you can edit and update the configuration file manually (section 2.6.2).

The "/etc/modbusslave.conf" file describes the devices and channels that will be available through the MODBUS protocol. The UEIModbus slave server opens and parses the configuration file at boot time to configure and start the I/O boards.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.1. Using the web UI to configure I/O channels

The UEIModbus I/O devices are listed under the **Channels** tab in the web UI display (see Figure 6 below).

The **Save configuration** button saves the current configuration to the UEIModbus. After updating channel parameters, you need to click **Save configuration,** and then stop and start the slave service to activate configuration changes.

To configure a channel, click the device where the channel you wish to configure is located. All I/O devices are listed in the **Device** pane under the **Channels** tab.

To enable a channel, click the corresponding **Enable** checkbox.

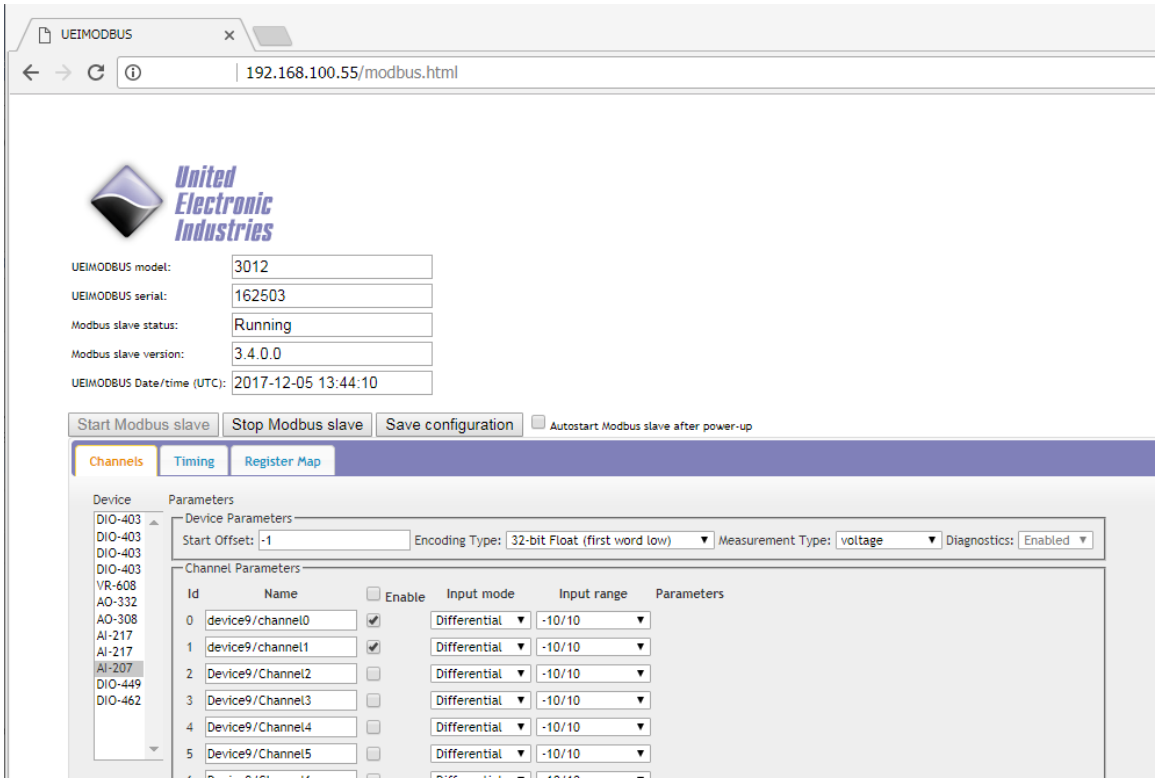

Figure 6 UEIModbus Channels Display

Channel Parameter settings (e.g., input mode, initialization values, gains, and scaling) for each enabled channel can be set independently. These parameters are described in section 2.6.2, “Using the command line to configure I/O channels”.

Configuration items under **Device Parameters** are set for all channels on the I/O board and are described below:

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

Start Offset: The start offset provides users a way of customizing the device address range for MODBUS register mapping of Input Registers and Holding Registers (refer to section 1.2 “Mapping PowerDNA I/O boards to MODBUS register tables” on page 2 for more information):

- **Start Offset = -1**: All applicable MODBUS registers (Coil, Discrete Input, Input and/or Holding Registers) for this I/O device will have the register start address mapped to the I/O board position in the chassis (board in slot 0 will have the start address mapped to 0, board in slot 1 will have the start address mapped to 1000, board in slot 2 will have the start addresses mapped to 2000, etc.) See Figure 6.

- **Start Offset = `<zero-based offset value>`**: Input and/or Holding Registers for the specified I/O device will have the register address mapped to the `<offset value>` (subsequent enabled channels on the I/O board will increment the address by the number of registers the data type requires).
  Coil and Discrete Input Registers for this I/O board will still have the register start address mapped to the I/O board position in the chassis (board in slot 0 will have the start address mapped to 0, board in slot 1 will have the start address mapped to 1000, board in slot 2 will have the start addresses mapped to 2000, etc.)
  See Figure 7.

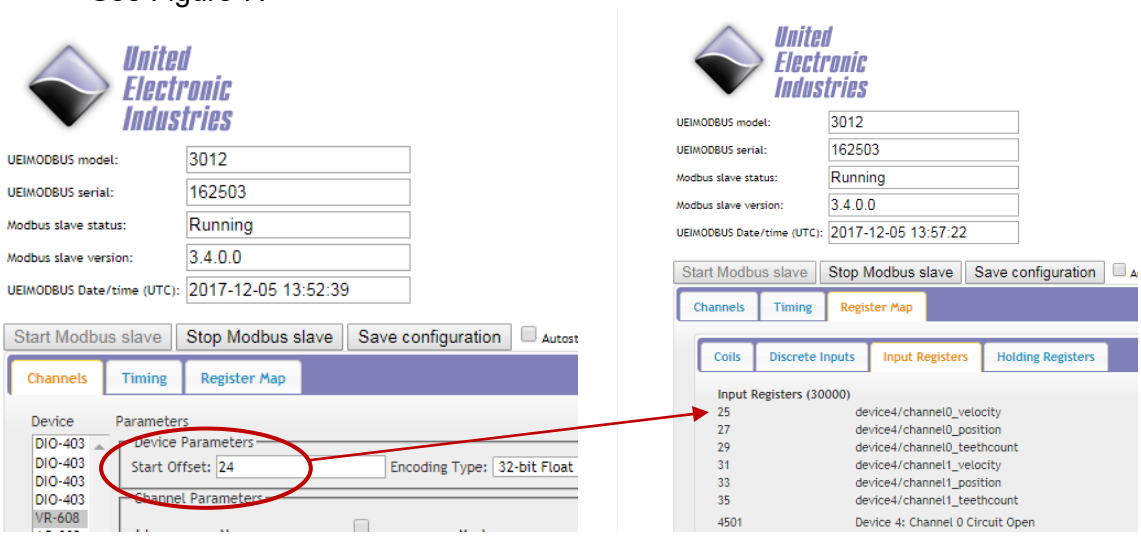

Figure 7 UEIModbus Start Offset Setting Mapped to Register Address

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

Encoding Type: The **Encoding Type** pull-down menu in the web UI corresponds to **DataType** parameters:

Encoding Type (GUI)	**DataType** (Parameter)
32-bit Integer (first word low)	i32
32-bit Integer (first word high)	swi32
32-bit Float (first word low)	f32
32-bit Float (first word high)	swf32
64-bit Double (first dword low)	f64
64-bit Double (first dword high)	swf64

Only supported data types for a device should be selected as the **Encoding Type**. The supported data types for each I/O device are listed in the **DataType** descriptions in sections 2.6.2.1 through 2.6.2.8 below.

Measurement Type: The measurement type indicates the board-specific type of measurements to be made. For example, the AI-207 can be configured to acquire voltage measurements or thermocouple measurements. Channel Parameters will have different options based on the Measurement Type.

Diagnostics: On boards that support internal diagnostic measurements (Guardian boards), the Diagnostics pulldown enables or disables acquiring diagnostic data.

2.6.1.1. Using the web UI to configure the Scan Rate

The Scan Rate is the rate at which samples are acquired and stored in the internal buffer. When an alarm is configured, the scan period is the maximum delay between the physical event and the alarm notification.

The scan rate (Hz) is set under the **Timing** tab (Figure 8).

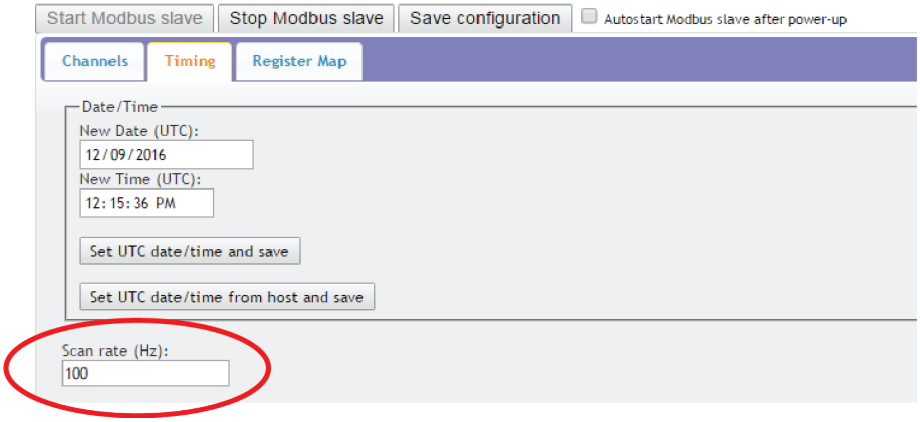

Figure 8 UEIModbus Scan Rate

United Electronic Industries
The High-Performance Alternative

2.6.2. Using the command line to configure I/O channels

Channels can be configured manually by editing the “/etc/modbusslave.conf” configuration file, located on the root file system (SD card).

The following syntax is used in the modbusslave.conf file to identify line entries as a device type or a comment:

- Each line starting with ‘#’ is a comment.
- Each line starting with “AI” configures an analog input device to measure voltage.
- Each line starting with “TC” configures an analog input device to measure temperatures from thermocouples.
- Each line starting with “SG” configures an analog input device to measure strain gauges.
- Each line starting with “RT” configures an analog input device to measure temperatures from RTDs.
- Each line starting with “IE” configures an analog input device to measure IEPE and accelerometer signals.
- Each line starting with “LV” configures an analog input device to measure LVDT or RVDT position.
- Each line starting with “SY” configures an analog input device to measure Synchro or Resolver position.
- Each line starting with “AO” configures an analog output device.
- Each line starting with “DI” configures a digital input device.
- Each line starting with “DO” configures a digital output device.
- Each line starting with “CI” configures a counter input device.
- Each line starting with “QU” configures a quadrature encoder input device.
- Each line starting with “VR” configures a variable reluctance input device.
- Each line starting with “CO” configures a frequency/PWM output device.

After editing the configuration file, you must restart the MODBUS slave service to activate changes:

```
/etc/init.d/modbusslave restart
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.1. Configuring analog input devices

The configuration line for a generic analog input device begins with “AI” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Device**={X}: ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **ScanRate**={X}: scan rate programmed on the AI device.
- **InputMode**={DIFF|RSE}: input mode.
- **Gain**={[X,Y,Z]}: comma separated list of gains for each channel.
  This is a 0 based index in list of gains supported by the device.
  For example AI-207 comes with the following gains:
  1,2,4,8,10,20,40,80,100,200,400,800
  Set **Gain**=6 to configure gain of 40, **Gain**=7 to configure gain of 80 and so on.
- **AutoZero**={0|1}: auto-zero enable. Some analog input devices come with a special channel to measure ground offset. Auto-zero subtracts the ground voltage measurement from the input voltage measurement.
- **Avg**: size of the moving average window applied to the measurements.
- **AvgThld**: percent change to reset moving average. If the value of an input sample changes by greater than the **AvgThld** percent, the array of moving average samples resets to the new value, allowing the moving average to settle at the new input level more quickly.
- **DataType**={i16|i32|swi32|f32|swf32|f64|swf64}: The data type used to transmit values between the slave and the master.
  i16: raw ADC value (only useful for 16-bit A/D devices).
  i32: raw ADC value.
  swi32: swapped raw ADC value.
  f32: single-precision floating point.
  swf32: single-precision floating point. Upper 16-bits and lower 16-bits are swapped.
  f64: double precision floating point.
  swf64: double precision floating point. Upper 32-bits and lower 32-bits are swapped.
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to map addresses. See section 2.6.1 on page 19 for more information).

The following example configures device 0 to acquire voltages on channels 0 to 3:

```
AI device=0 channellist=0,1,2,3 inputMode=DIFF gain=0,0,0,0
dataType=f32 startOffset=-1
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.1.1. Thermocouple (TC)

The thermocouple device inherits the parameters of the generic AI device (section 2.6.2.1). In addition the parameters below configure features specific to a thermocouple.

- **TCType**={E|J|K|S|R|T|B|N|C|0}: comma separated list of thermocouple types, or use “0” to acquire a voltage measurement.
- **TempScale**={C|F|K}:comma separated list of temperature scales, Celsius, Fahrenheit or Kelvin.
- **CJCType**={BUILTIN|CONSTANT}: the cold-junction compensation type. “BUILTIN” uses the sensor on the terminal panel. “CONSTANT” uses a constant value.
- **CJCConst**={X}: the cold-junction compensation temperature to use when CJCType is set to “CONSTANT”.

The following example configures device 2 to acquire temperatures on the first 4 channels (don’t wrap the line when typing it):

```
TC device=2 channellist=0,1,2,3 inputMode=DIFF gain=8,8,8,8
TCType=K,K,E,E tempScale=C,C,C,C CJCType=BUILTIN dataType=f32
```

NOTE: Thermocouple devices map an Input Register for the `CJC_Temperature` measurement that will be addressed directly after the Input Registers for the enabled channels.

2.6.2.1.2. Strain gage (SG)

The strain gage device inherits the parameters of the generic AI device (section 2.6.2.1). In addition the parameters below configure features specific to a strain gage.

- **ExcVoltage**: comma separated list of excitation voltages used to power load cells or strain gauges.
- **ScaleWithExcitation**: comma separated list containing 0 to read S-/S+ voltage or 1 to read ratiometric measurements in mV/V.
- **GAF**: comma separated list of gain adjustment factors obtained during shunt calibration procedure.

This is only available on devices designed to work with load cells or strain gauges (AI-208, AI-224, etc.)

The example below configures device 1 to acquire strains on channels 0 and 1:

```
SG device=1 channellist=0,1 inputMode=DIFF gain=8,8 dataType=f32
ExcVoltage=7.0,7.0 gaf=1.0,1.0
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.1.3. IEPE/Accelerometer (IE)

The IEPE device inherits the parameters of the generic AI device (section 2.6.2.1). In addition the parameters below configure features specific to an IEPE/accelerometer.

- **Coupling**: comma separated list containing DC, AC, AC_1 (1Hz HPF) or AC_0.1 (0.1Hz HPF)
- **Lowpass**: comma separated list containing 0 to disable low pass filter or 1 to enable low pass filter
- **Sensitivity**: the voltage read will be converted to mV and divided by the sensitivity specified in mVolts/[EU]
- **ExcCurrent**: Excitation current used to power the sensor (in mA)

This is only available on devices designed to work with IEPE sensors (AI-211).

The example below configures device 1 to acquire acceleration on the first 4 channels:

```
IE device=1 channellist=0,1,2,3 inputMode=DIFF gain=8,8,8,8
dataType=f32 ExcCurrent=1.0,2.0,3.0,4.0 LowPass=1,1,1,1
Sensitivity=1000.0,100.0,1000.0,1000.0 Coupling=AC,DC,AC,DC
```

2.6.2.1.4. RTD (RT)

The RTD device inherits the parameters of the generic AI device (section 2.6.2.1). In addition the parameters below configure features specific to an RTD.

- **Tempscale**: comma separated list of temperature scales: C for Celsius, F for Fahrenheit or K for Kelvin.
- **Wiring**: comma separated list containing 2 for two wires RTD, 3 for three wires RTD and 4 for four wires RTD.
- **RtdType**: comma separated list of temperature coefficients of resistance of the RTD. 3750, 3850, 3902, 3911, 3916, 3920, 3926, 3928.
- **RtdRes**: comma separated list of nominal resistances of the RTDs at 0 deg C.
- **LeadsResistance**: Resistance of the RTD leads (used in 2 wires mode only).

This is only available on devices designed to work with RTDs (AI-222).

The following example configures device 7 to acquire RTD temperatures on channels 0 and 1:

```
RT device=7 channellist=0,1 gain=0,0 Wiring=3,4 RtdType=3750,3750
RtdRes=100.0,100,0 tempScale=C,C LeadsResistance=0,0 dataType=f32
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.1.5. LVDT/RVDT (LV)

The LVDT device inherits the parameters of the generic AI device (section 2.6.2.1). In addition the parameters below configure features specific to the LVDT.

- **Wiring**: 4 for four wires connection, 5 for five wires connection.
- **ExcVoltage**: Excitation voltage used to power the LVDT.
- **ExcFreq**: Excitation frequency used to power the LVDT.
- **Sensitivity**: the voltage read will be converted to mV and divided by the excitation voltage and the sensitivity specified in mVolts/V/[EU].
- **ExtExc**: 0 to use internal excitation, 1 to use external excitation.

This is only available on devices designed to work with LVDT sensors (AI-254).

The following example configures device 1 to acquire positions on channels 0 and 1:

```
LV device=1 channellist=0,1 gain=0 dataType=f32 ExcVoltage=7.0
ExcFreq=1400.0 Sensitivity=1000.0 ExtExc=0
```

2.6.2.1.6. Synchro/Resolver (SY)

The Synchro/Resolver device inherits the parameters of the generic AI device (section 2.6.2.1). In addition the parameters below configure features specific to a Synchro or Resolver.

- **Mode**: "synchro", "resolver" or "synchrozground".
- **ExcVoltage**: Excitation voltage used to power the LVDT.
- **ExcFreq**: Excitation frequency used to power the LVDT.
- **ExtExc**: 0 to use internal excitation, 1 to use external excitation.

This is only available on devices designed to work with Synchro or Resolver sensors (AI-255 or AI-256).

The following example configures device 1 to acquire position angles on channels 0 and 1:

```
SY device=1 channellist=0,1 gain=0 dataType=f32 ExcVoltage=7.0
ExcFreq=1400.0 ExtExc=0
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.2. Configuring analog output devices

The configuration line for an analog output device begins with “AO” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Device**={X}: the ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **DataType**={f32|swf32|f64|swf64}: the data type used to transmit values between the master and the slave.
  f32: single-precision floating point
  swf32: single-precision floating point. Upper 16-bits and lower 16-bits are swapped
  f64: double precision floating point
  swf64: double precision floating point. Upper 32-bits and lower 32-bits are swapped
- **InitValue**={[X,Y,Z]}: comma separated list containing initial values to write to output channel when the Modbus slave is started.
- **EnableDiag**={1|0}: Enables (1) or disables (0) diagnostic functionality and diagnostic register mapping. Only applicable for Guardian devices, (i.e., Guardian analog output devices include AO-318 and AO-333).
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to map addresses. See section 2.6.1 on page 19 for more information).

The following example configures device 0 to generate on the first 4 channels:

```
AO device=0 channellist=0,1,2,3 dataType=f32
initValue=0.0,1.0,2.0,0.0
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.3. Configuring digital input devices

The configuration line for a digital input device begins with “DI” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Device**={X}: the ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **DataType**={i16|i32|swi32}: The data type used to transmit values between the slave and the master.
  i16: 16-bits integer
  i32: 32-bits integer
  swi32: swapped 32-bits integer. Upper 16-bits and lower 16-bits are swapped
- **LowHyst**={X}: Program the low hysteresis level.
- **HighHyst**={X}: Program the high hysteresis level.
- **EnableDiag**={1|0}: Enables (1) or disables (0) diagnostic functionality and diagnostic register mapping. Only applicable for Guardian devices, (i.e., Guardian digital input devices include DIO-448 and DIO-449).
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to map addresses. See section 2.6.1 on page 19 for more information).

The following example configures device 5 to acquire digital signals on its first port:

```
DI device=5 channellist=0 dataType=i32
```

DIO layer bits are mapped contiguously in all register tables, the data type setting doesn’t have any effect.

For example, on DIO-403, the six 8-bit ports are mapped in all four register tables, input registers, holding registers, discrete inputs and coils.

So if a DIO-403 is mapped at address 0:

Port 0 (lines 0 to 7) is mapped as bits 0-7 of register 0
Port 1 (lines 8 to 15) is mapped as bits 8-15 of register 0
Port 2 (lines 16 to 23) is mapped as bits 0-7 of register 1
Port 3 (lines 24 to 31) is mapped as bits 8-15 of register 1
Port 4 (lines 32 to 39) is mapped as bits 0-7 of register 2
Port 5 (lines 40 to 47) is mapped as bits 8-15 of register 2

The direction of a port doesn’t change its mapping but of course changes its behavior.
Reading from an output port will return its state (the last value written to it).
Writing to an input port will do nothing.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.4. Configuring digital output devices

The configuration line for a digital output device begins with “DO” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Device**={X}: the ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **OutputMask**={X}: selects the ports that are configured for output (for bi-directional devices such as the DIO-403).
- **DataType**={i16|i32|swi32}: The data type used to transmit values between the master and the slave.
  i16: 16-bits integer
  i32: 32-bits integer
  swi32: swapped 32-bits integer. Upper 16-bits and lower 16-bits are swapped
- **InitValue**={[X,Y,Z]}: comma separated list containing initial values to write to output port when the Modbus slave is started
- **EnableDiag**={1|0}: Enables (1) or disables (0) diagnostic functionality and diagnostic register mapping. Only applicable for Guardian devices, (i.e., Guardian digital output devices include DIO-432, DIO-433, DIO-462, DIO-463).
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to map addresses. See section 2.6.1 on page 19 for more information).

The following example configures device 1 to generate digital patterns on its first port:

```
DO device=1 channellist=0 dataType=i32 initValue=0x555
```

Special notes for DIO-403: The DIO-403 is the only device that can set the direction of its six 8-bit digital ports. Each bit contained in the **outputMask** parameter sets the direction of its respective port. For example “outputMask=0x11” will set ports 0 and 4 as outputs and ports 1, 2, 3 and 5 as inputs.

The following example configures DIO-403 at device 0 to generate digital patterns on ports 1, 3 and 5

```
DO device=0 numChannels=6 outputMask=0x2A dataType=i32
```

This configuration makes 48 coil registers accessible in the coil table:

- Port0 maps to coils 0 to 7 and writing to those coils are ignored (port is input)
- Port1 maps to coils 8 to 15
- Port2 maps to coils 16 to 23, writing to those coils are ignored
- Port3 maps to coils 24 to 31
- Port4 maps to coils 32 to 39, writing to those coils are ignored
- Port5 maps to coils 40 to 47

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.5. Configuring counter input devices

The configuration line for a counter input device begins with “CI” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Device**={X}: the ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **DataType**={i16|i32|swi32|f32|swf32|f64|swf64}: The data type used to transmit values between the slave and the master.
  i16: 16-bits integer
  i32: 32-bits integer
  swi32: swapped 32-bits integer. Upper 16-bits and lower 16-bits are swapped
  f32: single-precision floating point
  swf32: single-precision floating point. Upper 16-bits and lower 16-bits are swapped
  f64: double precision floating point
  swf64: double precision floating point. Upper 32-bits and lower 32-bits are swapped
- **Mode**={count|quad|period|pulsewidth}: The mode used to configure the counter to measure events, quadrature encoder position, period or pulse width.
- **Source**={internal|external}: The source signal to count or measure, internal uses the on-board 66MHz clock, external uses the signal connected to the counter’s input pin.
- **Gate**={internal|external}: The gate signal to enable/disable the counter, internal sets the gate automatically when counter starts, external uses a signal connected to the counter’s gate pin.
- **InputInverted**={0|1}: Set to 1 to invert the input signal.
- **SourceDebouncer**={X}: The minimum pulse width in micro-secs detected on the counter input.
- **GateDebouncer**={X}: The minimum pulse width in micro-secs detected on the counter gate.
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to start addresses. See section 2.6.1 on page 19 for more information).

The following example configures device 2 to measure quadrature encoders position connected to ports 0, 1, 2, and 3:

```
CI device=2 channellist=0,1,2,3 dataType=i32 mode=quad
source=external gate=internal inputinverted=0
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.6. Configuring quadrature encoder input devices (QU)

The configuration line for a quadrature input device begins with “QU” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Device**={X}: the ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **DataType**={f32|swf32|f64|swf64}: The data type used to transmit values between the slave and the master.
  f32: single-precision floating point
  swf32: single-precision floating point. Upper 16-bits and lower 16-bits are swapped
  f64: double precision floating point
  swf64: double precision floating point. Upper 32-bits and lower 32-bits are swapped
- **Initialpos**: Initial count loaded in counter (default is 0).
- **Decodingtype**: 1x, 2x or 4x. 2x and 4x decoding are more sensitive to smaller changes in position.
- **Zeroindex**: 1 to enable zero indexing. resets the measurement to the initial value when the Z, A and B inputs are in a given state.
- **Zeroindexphase**:
  - zhigh: resets measurement when Z goes high
  - alowblow: resets measurement when Z goes high, A is low and B is low
  - alowbhigh: resets measurement when Z goes high, A is low and B is high
  - ahighblow: resets measurement when Z goes high, A is high and B is low
  - ahighbhigh: resets measurement when Z goes high, A is high and B is high
- **ADebouncer**: the minimum pulse width in micro-secs on A input. Any pulse whose width is smaller will be ignored.
- **BDebouncer**: the minimum pulse width in micro-secs on B input. Any pulse whose width is smaller will be ignored.
- **ZDebouncer**: the minimum pulse width in micro-secs on Z input. Any pulse whose width is smaller will be ignored.
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to map addresses. See section 2.6.1 on page 19 for more information).

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.7. Configuring frequency/PWM output devices (CO)

The configuration line for a frequency output device begins with “CO” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Device**={X}: the ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **DataType**={i16|i32|swi32}: The data type used to transmit values between the master and the slave.
  i16: 16-bits integer
  i32: 32-bits integer
  swi32: swapped 32-bits integer. Upper 16-bits and lower 16-bits are swapped
- **Mode**={pulse|train}: The mode used to configure the counter to output pulse(s), “pulse” will output a single pulse each time a new value is written to the Modbus register. “train” will continuously output pulses.
- **lowticks**={X}: The initial number of clock ticks used to specify the low state duration.
- **highticks**={X}: The initial number of clock ticks used to specify the high state duration.
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to map addresses. See section 2.6.1 on page 19 for more information).

The following example configures device 2 to output pulses out of ports 0,1,2,3:

```
CO device=2 channellist=0,1,2,3 dataType=i32 mode=train lowticks=1000
highticks=1000
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.8. Configuring variable reluctance input devices (VR)

The configuration line for a variable reluctance input device begins with “VR” and can contain any of the following “name=value” pairs. If a parameter is omitted, a default value will be used instead.

Variable reluctance channels produce three values each in the input register table:

- Velocity in RPM
- Position in number of teeth
- Total teeth count since Modbus slave was started.

Parameters are not case sensitive.

- **Device**={X}: the ID of the device to configure.
- **ChannelList**={[X,Y,Z]}: comma separated list of channels.
- **DataType**={f32|swf32|f64|swf64}: The data type used to transmit values between the slave and the master.
  f32: single-precision floating point
  swf32: single-precision floating point. Upper 16-bits and lower 16-bits are swapped
  f64: double precision floating point
  swf64: double precision floating point. Upper 32-bits and lower 32-bits are swapped
- **EnableDiag**={1|0}: Enables (1) or disables (0) diagnostic functionality and diagnostic register mapping.
  For VR-608 device, diagnostic registers provide Open/Short status for each channel.
- **StartOffset**: start address for Input/Holding registers in Modbus register map for device (-1 uses default addressing; `<offset value>` uses user-programmed offset to map addresses. See section 2.6.1 on page 19 for more information).
- **VrMode**= {Decoder|Timed|Npulses|Zpulse}: The mode used to measure velocity, position or direction. The mode can be set to:
  - Decoder: Even and Odd channels are used in pair to determine direction and position
  - Timed: Count number of teeth detected during a timed interval
  - Npulses: Measure the time taken to detect N teeth (Number of teeth needs to be set)
  - Zpulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

- **ZcMode**={Chip|Logic|fixed}: Zero crossing finds the point in time where the VR sensor output voltage transitions from positive to negative voltage. This point is when the center of the tooth is lining up with the center of the VR sensor. The zero crossing mode can be set to:
  - Chip: The front-end IC will automatically calculate the ZC level
  - Logic: The device's FPGA measures the VR sensor signal and calculate the ZC level as (min+max)/2
  - Fixed: Use hard-coded ZC level (specified below)
- **APTMode**={Chip|Logic|fixed}: APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth. The APT mode can be set to:
  - Chip: The front-end IC will automatically set the AP threshold to 1/3 of the peak input voltage
  - Logic: The device's FPGA measures the VR sensor signal and sets the AP threshold to a programmable fraction of the peak input voltage
  - Fixed: Use hard-coded AP threshold (specified below)
- **ADCRate**: The rate in Hz at which the VR sensor signal is measured.
- **MovingAverage**: The size of the moving average window applied to the VR sensor signal while it is measured.
- **APTThresholdDivider**: The APT threshold divider is used when APT mode is set to "Logic". It specifies that the AP threshold will be set at a fraction of the peak input voltage. This is a value between 1 and 15: 1=1/2, 2=1/4, 3=1/8 etc...
- **APTThreshold**: The APT threshold is used when APT mode is set to "Fixed".
- **ZCThreshold**: The ZC threshold is used when ZC mode is set to "Fixed".
- **NumberOfTeeth**: The number of teeth on the encoder wheel.
- **ZToothSize**: A Z Tooth is usually materialized by one or more missing teeth or one or more fused teeth. This parameter specified the number of fused or missing teeth.
- **TimedRate**: The rate at which teeth are counted when VrMode is set to “timed”.

The following example configures device 2 to measure VR sensor signals connected to ports 0, 1, 2, and 3:

```
VR device=2 channellist=0,1,2,3 vrmode=timed timedrate=2.0
zcmode=chip aptmode=chip numberofteeth=72 ztoothsize=0 dataType=f32
```

NOTE: Each variable reluctance channel enabled yields 3 Input Registers in the MODBUS register table: a `_velocity`, `_position`, and `_teethcount` register.

Also, enabling Diagnostics yields a Diagnostic Open/Short Register for each channel, where reading a 1 means an open circuit is detected between the IN± channels and reading a 0 means a short circuit is detected.

Refer to Figure 9 and Figure 10 for web UI displays of VR-608 channel configuration and register mapping.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

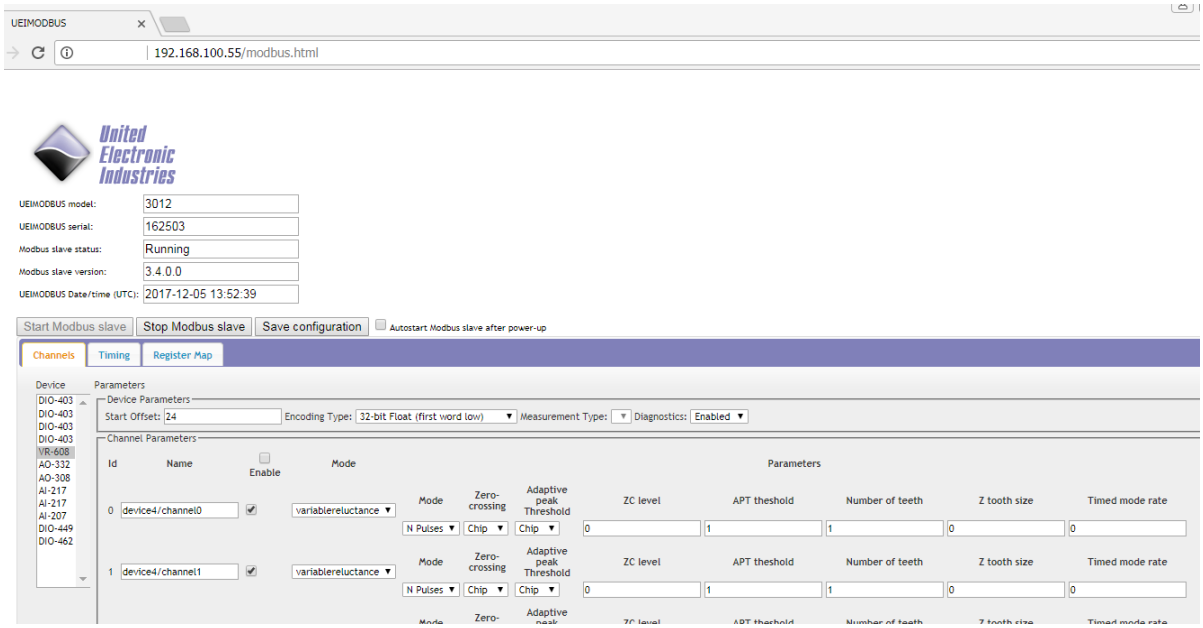

Figure 9 VR-608 Channels Configuration in Web UI

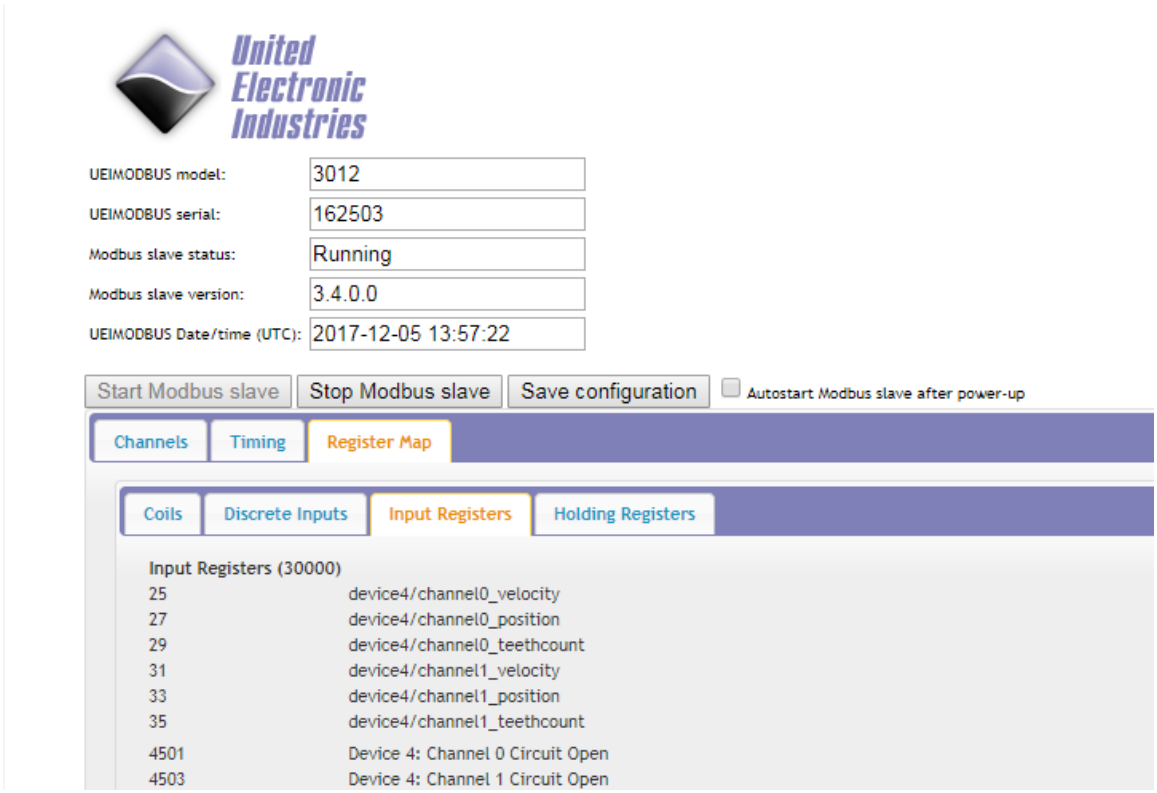

Figure 10 VR-608 Register Mapping when Diagnostics Are Enabled

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

2.6.2.9. Configuring an ARINC-429 device

Compatible devices are 429-566, 429-516-024, 429-516, 429-512.

Receive channels are mapped as input registers at offset "devn*1000"
Transmit channels are mapped as holding registers at offset "devn*1000"
The offset can be overridden with the "startOffset" attribute

Each receive/transmit channels must be configured to receive or transmit a specific list of labels.
Each Input label occupies six 16-bit registers: 2 registers for data, one register for SSM, one register for SDI and two registers for timestamp

16-bit register Offset in input register table	Value
0	Label0 Value Low
1	Label0 Value High
2	SSM0
3	SDI0
4	Timestamp0 Low
5	Timestamp0 High
6	Label1 Value Low
7	Label1 Value High
8	SSM1
9	SDI1
10	Timestamp1 Low
11	Timestamp1 High

Each Output label occupies four 16-bit registers: 2 registers for data, one register for SSM and one register for SDI

16-bit register Offset in holding register table	Value
0	Label0 Value Low
1	Label0 Value High
2	SSM0
3	SDI0
4	Label1 Value Low
5	Label1 Value High
6	SSM1
7	SDI1

ARINC-429 devices are configured exclusively using the XML configuration file format.
XML Example:

```
<device devn="0" type="arinc429" startOffset="-1">
    <channel id='0' name='RX0' direction='input'>
        <bitrate>100000</bitrate>
```

United Electronic Industries
The High-Performance Alternative

```
            <!-- Parity is used to detect errors on incoming labels
                 A parity error will set the data registers to 0xFFFFFFFF
                 -->
            <parity>none</parity>
            <timestamp>off</timestamp>
            <!-- This label is mapped at devn*1000
                 Data is mapped at devn*1000
                 SDI is mapped at devn*1000 + 2
                 SSM is mapped at devn*1000 + 3
                 Timestamp mapped at devn*1000 + 4
                 -->
            <label id='102' format='raw'/>
            <!-- This label is mapped at devn*1000+6
                 The data field uses the "Binary Coded Decimal" representation -->
            <label id='56' format='bcd'>
                <!-- resolution is a multiplier/divider to represent
                  real numbers (default is 1) -->
                <resolution>0.01</resolution>
            </label>
            <!-- This label is mapped at devn*1000+12
                 The data field uses the "Binary Number Representation"
                 This representation uses two's complement arithmetic -->
            <label id='55' format='bnr'>
                <!-- lsb of the data field (default is 11)
                     the example uses 16 bits values so lsb is 29-16 = 13-->
                <lsb>13</lsb>
                <length>14</length>
                <!-- range of the encoded value. The example of 16384 allows
                     the encoding of values between -16384 and +16384 -->
                <bnrRange>16384</bnrRange>
            </label>
        </channel>
        <!--Transmitted labels are mapped sequentially at offset
            "devn*1000 + label_index*4" -->
        <channel id='0' name='TX0' direction='output'>
            <bitrate>12500</bitrate>
            <!-- Parity is used to calculate the parity bit on outgoing labels -->
            <parity>odd</parity>
            <!-- Labels can be transmitted as soon as a new value is written to
                 the holding registers in 'oneshot' mode -->
            <label id='12' format='raw' mode='oneshot'/>
            <!-- Labels can also be transmitted periodically -->
            <label id='98' format='bcd' mode='periodic' periodus='100000'>
                <lsb>11</lsb>
                <length>18</length>
                <defaultData>100</defaultData>
                <defaultSSM>2</defaultSSM>
                <defaultSDI>1</defaultSDI>
            </label>
        </channel>
    </device>
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

The **channel** element for an ARINC-429 device represents an input (RX) or output (TX) channel.

The **direction** attribute determines whether a channel is input (RX) or output (TX).

The element can contain any of the following sub-elements. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **BitRate**: Configure the channel bit rate. Only two values are valid: 12500 or 100000 bits/sec.
- **Parity**: Configure the parity bit behavior (none, odd or even).
  On output channel the parity bit is automatically calculated to match this parameter.
  On input channels the parity bit is actually a parity error bit. It is set to 1 when the received bit doesn't match the calculated parity.

Each **channel** element must contain one or more **label** elements.
Input channels will filter out any incoming label that doesn't match one of the label elements.
Output channels will only send an output label from the list depending on its scheduling policy.

Label elements contain the following attributes:

- **Id**: The label itself in decimal, octal (prefixed with 0) or hexadecimal (prefixed with 0x)
- **Name**: The name of the label.
- **Format**: The format used to encode or decode the data value. Possible formats are: *BNR, BCD* or *Raw.*
- **Mode**: This parameter is only valid for output labels. It configures the scheduling policy to *Periodic* or *OneShot*.
  Periodic: The label is automatically transmitted periodically. The label data can be updated via Modbus/TCP
  OneShot: The label is only transmitted when a new value is written via Modbus/TCP.
- **PeriodUs**: The period in microseconds for re-transmitting a periodically scheduled label.

Label elements can contain any of the following sub-elements:

- **Lsb**: The least significant bit of the data field in the ARINC word. LSB must be set between 11 and 28. Default value is 11.
- **Length**: The number of bits used by the data field. Default value is 19
- **Resolution**: A scaling factor applied to the value. Default value is 1.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

- **BnrRange**: The maximum value encoded in BNR format
- **DefaultData**: The default data field transmitted by a periodic label (until it gets updated by a Modbus/TCP write)
- **DefaultSSM**: The default SSM field transmitted by a periodic label (until it gets updated by a Modbus/TCP write)
- **DefaultSDI**: The default SDI field transmitted by a periodic label (until it gets updated by a Modbus/TCP write)

BCD format:
Each digit of the encoded value is represented by four bits used to store a value between 0 and 9. The sign is set by the SSM field (0 for positive and 0x3 for negative)
Example: A data field of 0x12345 represents the decimal value 12345

BNR format:
The value is encoded in two's complement arithmetic.
The BnrRange parameter specifies the minimum and maximum value.
Example: with bnrRange set to 16384, a value between -16383 and 16384 can be encoded over 14 bits. The sign is stored in bit 29 (1 for negative, 0 for positive).

2.6.2.10. Configuring an I2C device

Compatible device is I2C-534.

Master and slave channels are mapped as holding registers at offset "devn*1000"
The offset can be overridden with the "startOffset" attribute.
I2C devices do not use the input register table.

Master and slave channels are controlled via a command register.
For master channels, data is transmitted and/or received on the bus each time the command register is written to.
For slave channels, data is read or written to the FIFO each time the command register is written to.

Master channels also come with two slave address registers to specify the address of the slave to write to or read from.

16-bit register Offset in holding register table	Value
0	Command register
1	Address1 register
2	Address2 register
3	Byte 0
4	Byte 1
5	Byte 2
6	Byte 3

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

…	…
100	Byte 97
101	Byte 98
102	Byte 99
103	<Next master or slave channel>

Slave channels come with one command register that specified the number of bytes to transfer and the direction (write or read)

16-bit register Offset in holding register table	Value
0	Command register
1	Byte 0
2	Byte 1
…	…
99	Byte 98
100	Byte 99
101	<Next master or slave channel>

I2C devices are configured exclusively using the XML configuration file format.

For example:

```
<device type="i2c" devn="4" dataType="i16">
    <channel id="2" name="master0">
        <enabled>on</enabled>
        <mode>master</mode>
        <bitRate>400000</bitRate>
        <ttlLevel>5</ttlLevel>
        <secureShell>on</secureShell>
        <multiMaster>off</multimaster>
        <termination>on</termination>
        <loopback>on</loopback>
        <xdcp>off</xdcp>
    </channel>
    <channel id="2" name="slave2">
        <enabled>on</enabled>
        <mode>slave</mode>
        <bitRate>400000</bitRate>
        <ttlLevel>3.3</ttlLevel>
        <slaveAddress>5</slaveAddress>
        <slaveAddressWidth>7</slaveAddressWidth>
    </channel>
</device>
```

The **channel** element for an I2C device represents a master or slave channel.

The **mode** element determines whether a channel is master or slave.

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

The channel element can contain any of the following sub-elements. If a parameter is omitted, a default value will be used instead.

Parameters are not case sensitive.

- **Mode**: Configure the channel as a master or as a slave.
- **BitRate**: Configure the channel bit rate. Only three values are valid: 100000, 400000 or 1000000 bits/sec.
- **TtlLevel**: Configure the TTL level for data and clock signals.
- **SecureShell**: Enable or disable secure shell which verifies that commands are transmitted without errors using CRC.
- **MultiMaster**: Enable or disable multi-master capability
- **Termination**: Enable or disable termination
- **Loopback**: Enable or disable loopback between master and slave channels sharing the same ID
- **Xdcp**: Enable or disable XDCP mode for devices that require it.
- **SlaveAddressWidth**: Configure the address width of a slave channel (7 or 10)
- **SlaveAddress**: Configure the address of a slave channel

www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

3. Booting strategies

The UEIModbus file system contains the libraries, executables, and configuration files needed to make the system functional.

By default, the file system is stored on the SD card inserted on the front panel of the UEIModbus.

The file system can alternatively be located in a RAM drive loaded from the FLASH memory or loaded from a remote server using the NFS protocol.

The standard UEIModbus file system is read/write to ease the configuration and allow uploading of files during the development phase.

Once an application/configuration is stable, it is recommended to convert the file system to read-only mode to render the UEIModbus file system resilient against unscheduled shutdowns.

3.1. Booting an SD card with system partition read-only

The procedure below converts the standard UEIModbus file system to a read-only one.

1. Edit /etc/fstab as below to mount a RAM disk at /var (ram disk maximum size is set to 2MBytes):

```
/dev/sdcard1    /           ext3    defaults,noatime    1   1
none            /proc       proc    defaults            0   0
none            /sys        sysfs   defaults            0   0
none            /dev/pts    devpts  defaults            0   0
tmpfs           /var        tmpfs   defaults,size=2M    0   0
```

2. Create a new script /etc/varsetup.sh with the content below. It sets up the folders needed in /var and maps a few writable folders at /tmp, /mnt and /home.

```
mkdir /var/tmp
mkdir /var/log
mkdir /var/lib
mkdir /var/lib/misc
mkdir /var/spool
mkdir /var/spool/cron
mkdir /var/spool/cron/crontabs
mkdir /var/run
mkdir /var/lock
mkdir /var/mnt
mkdir /var/home
```


www.ueidaq.com
508.921.4600

United Electronic Industries
The High-Performance Alternative

```
mount --bind /var/tmp /tmp
mount --bind /var/mnt /mnt
mount --bind /var/home /home
```

3. Edit /etc/inittab as shown below to execute varsetup.sh.

```
# Mount all filesystem listed in /etc/fstab
::sysinit:/bin/mount -a

# Create and mount non-persistent folders
::sysinit:/etc/varsetup.sh

# Configure local network interface
::sysinit:/sbin/ifconfig lo 127.0.0.1 up
::sysinit:/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo

# run rc scripts
::sysinit:/etc/rcS

# Start a shell on the console
ttyS0::respawn:-/bin/sh

# unmount root file system when shutting-down
::shutdown:/bin/umount -a -r
```

4. Create symbolic links to files stored in /etc that need to be kept writeable.

```
ln -s /var/resolv.conf /etc/resolv.conf
ln -s /var/layers.xml /etc/layers.xml
```

5. Connect the console serial port, power-up the UEIModbus, and press any key to enter U-Boot. Type the following commands to load the root file system read-only:

```
setenv bootargs console=ttyS0,57600 root=62:1 ro
saveenv
reset
```

3.2. Restoring or creating a new SD card

Restoring or initializing a new SD card can only be done on a Linux PC (real or virtual).

1. Locate the SD card image file rfs-x.y.z.tgz on your UEIModbus CDROM as well as the script containing the sequence of commands to partition, format, and initialize a new SD card.
2. Connect the SD card via a USB adapter (or directly if your computer has a built-in reader).

United Electronic Industries
The High-Performance Alternative

3. Type the command ***dmesg*** to find out what device node is associated with the SD card. (Linux kernel outputs messages when it detects a new removable drive).
4. Assuming that /dev/sdb is the SD card device node, type ***./createsdcard.sh /dev/sdb rfs-x.y.z.tgz*** to partition, format and copy files to the card.

www.ueidaq.com
508.921.4600