

MODBUS TCP

Feature Overview and Configuration Guide

Introduction

This guide describes the MODBUS protocol and how to configure it on AlliedWare Plus™ devices. The MODBUS protocol is a request-response protocol used in Supervisory Control And Data Acquisition (SCADA) systems for industrial automation devices. It provides a simple open and standardized method for devices to communicate using a client /server model.

Caution: While initially deployed over RS232 interfaces, versions of the MODBUS protocol exist for serial lines (MODBUS RTU and MODBUS ASCII) and for Ethernet (MODBUS TCP). This guide focuses on MODBUS TCP, the variant intended for use over Ethernet and IP networks.

The most common use of the protocol is for Ethernet attachment of Programmable Logic Controllers (PLCs), I/O modules, and gateways to other simple field buses or I/O networks. It is used widely by many manufacturers throughout many industries. MODBUS is typically used to transmit signals from instrumentation and control devices back to a main controller or data gathering system.

Products and software version that apply to this guide

This guide applies to AlliedWare Plus™ products that support MODBUS, running version **5.4.8.2** or later. To see whether your product supports MODBUS see the following documents:

- The product's [Datasheet](#)
- The product's [Command Reference](#)

These documents are available from the above links on our website at alliedtelesis.com.

New features and licensing

- From software version 5.4.9 onwards, the VCStacking feature is available on MODBUS.
- The MODBUS feature requires a license.



Contents

Introduction	1
Products and software version that apply to this guide	1
MODBUS overview	3
How does the MODBUS protocol work?	3
MODBUS TCP packets.....	4
MODBUS TCP messages	5
Security	6
Configuring MODBUS TCP	6
Basic example	6
Changing server parameters	6
Write access	7
Managing client connections.....	7
MODBUS TCP heartbeats	8
Monitoring.....	11
Logging	13
Clear commands.....	13
How do MODBUS clients and servers communicate?	14
VCStack support for MODBUS.....	15
Limitations.....	15
Register mapping	16
Register value definition.....	19

MODBUS overview

MODBUS is commonly used in industrial environments to monitor, gather, process, and transfer real-time data between devices. Many devices such as PLCs, intelligent devices, sensors, and instruments use MODBUS for the purposes of SCADA. SCADA is a system of software and hardware elements that allows industrial organizations to control industrial processes locally or at remote locations.

The MODBUS feature allows AlliedWare Plus™ devices to be used for some SCADA processes, such as gathering sensor and alarm information and to control and monitor the state of ports and their PoE state.

How does the MODBUS protocol work?

The MODBUS protocol is a request-response protocol. In simple terms, this means a client can request the MODBUS server to perform an action and the server will respond with that action. The simplified process is as follows.

The client (PLC or some other device):

1. Connects to the MODBUS server IP address on port 502 (default).
2. Sends MODBUS TCP request packets to the server.

The MODBUS server (AlliedWare Plus device):

1. Receives the client MODBUS TCP request packet.
2. Actions the request.
3. Responds to the client with an acknowledgment of success or failure.

The client can stay connected for as long as it needs to, making MODBUS requests.

MODBUS TCP packets

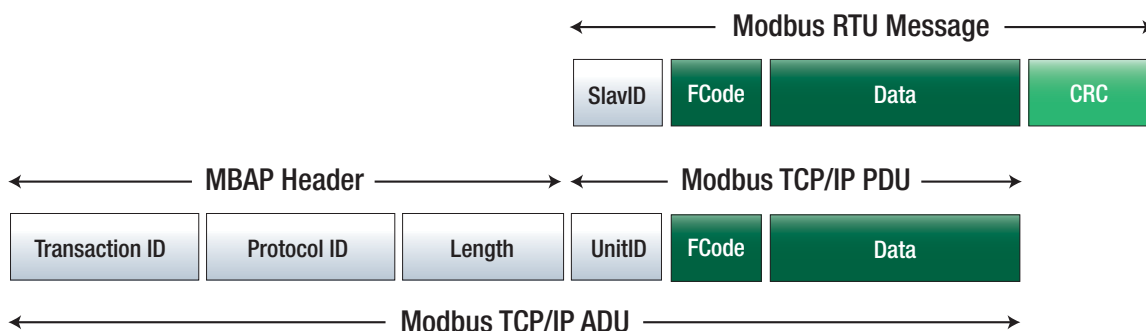
The MODBUS TCP protocol is basically a MODBUS RTU message transmitted with a TCP/IP wrapper and sent over a network instead of serial lines.

The wrapper, added at the start of the message, contains a 7-byte header called the MBAP header (MODBUS Application Header). The MBAP is a general-purpose header that depends on a reliable networking layer.

The MBAP header contains the following data:

- **Transaction Identifier:** 2 bytes set by the client to uniquely identify each request. These bytes are echoed by the server since its responses may not be received in the same order as the requests.
- **Protocol Identifier:** 2 bytes set by the client, which is always = 00 00
- **Length:** 2 bytes identifying the number of bytes in the message to follow.
- **Unit Identifier:** 1 byte set on the client, which uniquely identifies it on the network.

Figure 1: MODBUS packet structure



TCP must establish a connection before transferring data, since it is a connection-based protocol.

- The client in MODBUS TCP establishes a connection with the server.
- The server waits for an incoming connection from the client.
- Once a connection is established, the server then responds to the queries from the client until the client closes the connection.

MODBUS TCP messages

The MODBUS TCP protocol defines the content of a packet so a receiver can open it up, look at it and figure out what the message is.

The message could be an action like:

- Read and send me some data from a given memory address
- Write this data to a given memory address
- Start a process
- Terminate a process, and so on.

But how is the message conveyed at a packet level? Looking at [Figure 1 on page 4](#), we can see that the MODBUS TCP message contains a function code, an address, plus some extra instruction data.

Function codes Function codes are numbered 1, 2, 3, 4, etc. Each function does one thing: read or write one or more coils or registers. The Modbus protocol defines several function codes for accessing Modbus registers.

Supported function codes

Allied Ware Plus supports the following MODBUS function codes:

- Read Holding Registers (03)
- Read Input Registers (04)
- Write Holding Register (06)

Address The address, known as a **Coil** or **Register** which are just variable names for memory addresses.

The address represents the address of the register mapping to start reading from or writing to. So, if you wanted to turn on a valve, you would call the appropriate software function to write to the coil.

- A coil is a boolean (single bit) variable
- A register is an integer (16 bit, word) variable

Extra data The extra instruction data might be a quantity value, such as how many words from the start address to read/write or the value to write.

In summary, the software in the client uses the MODBUS protocol to issue a request (function code) for the values in the specified addresses. The server replies with a response which contains the data. If the client doesn't issue a request, the server remains silent.

Security

There is no security in MODBUS and it requires no authentication. This means that an unauthorized user could make a TCP connection to the MODBUS TCP server and access data, shut down ports, or disable PoE. This is a known issue of SCADA and MODBUS and is a consequence of the protocol.

Be aware that:

- MODBUS is disabled by default and must be globally enabled.
- Write access requires a separate command to be enabled.

See the section below: "[Configuring MODBUS TCP](#)", for examples of these commands.

Note: We recommend that you configure hardware ACLs to only permit connections to the TCP port from known authorized IP addresses and to block access from all other IP addresses. For more detail, see "[Limiting client access](#)" on page 7.

Configuring MODBUS TCP

To get the MODBUS TCP server working, first configure an IP address on the device so clients can connect to it, and then enable the MODBUS TCP server. Once this is done, clients can connect to the server.

Basic example

Step 1: Configure an IP address on the device

For example, to give VLAN1 an IP address of 198.51.100.48/24, use the following commands:

```
awplus#configure terminal
awplus(config)#int vlan1
awplus(config-if)#ip address 198.51.100.48/24
awplus(config)#exit
```

Step 2: Enable the MODBUS TCP server

```
awplus(config)#scada modbus tcp server
```

Changing server parameters

TCP port To change the port of the MODBUS TCP server, use the command:

```
awplus(config)#scada modbus tcp server port 31234
```

This changes the port that clients must use to connect to the MODBUS TCP server.

Connection limit

To change the maximum limit of concurrent connected clients, use the command:

```
awplus(config)#scada modbus tcp server connection
```

- If the value entered is less than the currently configured limit, then all the currently connected clients are disconnected.
- The MODBUS server supports a maximum of 5 concurrent connections and a default of 1.

Write access

To enable clients to have writable access to the MODBUS TCP server and use write function codes, use the following commands:

```
awplus#configure terminal
```

```
awplus(config)#scada modbus tcp server access read-write
```

Managing client connections

Existing MODBUS client connections can be checked using the command:

```
awplus#show scada modbus tcp server connections
```

An existing client can be forcefully disconnected by using the command:

```
awplus#clear scada modbus tcp server connection (A.B.C.D | X.X::X.X) <0-65535>
```

This will disconnect the client specified by the IP address and TCP port number. Also, see ["Clear commands" on page 13](#).

Limiting client access

MODBUS has no built-in authentication. To remedy this, access to MODBUS on the server can be restricted to the IP address of clients that are authorized, which can be accomplished via MODBUS configuration and/or by applying hardware ACLs. It is possible to use a combination of both features if required.

MODBUS clients can be restricted by IP address.

- Only permitted IP addresses are able to connect to the MODBUS TCP server once one is configured.
- When a permitted IP is configured, if currently connected clients are not permitted then they will be disconnected.
- When a permitted IP is de-configured, currently connected clients with this IP will be disconnected.

To remove 192.168.2.200 from the permitted MODBUS TCP server client IP addresses, use the following commands:

```
awplus#configure terminal
```

```
awplus(config)#no scada modbus tcp server access permit 192.168.2.200
```

To allow only the client with source IP address 198.51.100.200 to connect to MODBUS, use the command:

```
awplus#scada modbus tcp server access permit 198.51.100.200
```

To configure ACLs to restrict access, use the following process:

Step 1: Create an ACL that permits access from some source IP address(es) to the destination port.

```
awplus#configure terminal
awplus(config)#access-list 3000 permit tcp 198.51.100.200/32 any eq 502
```

Step 2: Create another ACL that denies access to all other IP addresses.

```
awplus(config)#access-list 3001 deny tcp any any eq 502
```

Step 3: Then apply the ACLs onto the ports that clients are connected to.

```
awplus(config)#int port1.0.5
awplus(config-if)#access-group 3000
awplus(config-if)#access-group 3001
```

MODBUS TCP heartbeats

From version 5.5.0-0.1, MODBUS provides three registers that can be used as heartbeats to detect liveliness of the MODBUS server running on the device. These registers are new in version 4 of the mapping register.

TCP connection uptime register 0x004A

The purpose of this register is to allow a MODBUS client to easily determine its own uptime for a particular MODBUS server using the same interface that it uses to generally query that server. This may be easier to work with than trying to access this data locally through other means.

Name	Length	Access	Description
0x004A	2 words (32 bits)	Read only	Returns the uptime in seconds of the TCP connection on which the MODBUS read request was received. MODBUS clients can only access their own TCP connection uptime.

Master heartbeat time register 0x004C

The purpose of this register is to allow a client to start a global timer on the server, indicating the frequency at which MODBUS requests should be received by the MODBUS server. Each time a MODBUS request is received, from any MODBUS client, the timer will be restarted. If the timer expires, it indicates that no MODBUS requests have been received from any MODBUS clients during the timeout interval.

A warning log message is generated when the timer expires. This allows you to optionally configure a log trigger to run configuration scripts on the device if this timeout occurs (such as disabling or enabling interfaces, or other potential diagnostic operations).

Name	Length	Access	Description
0x004C	1 word (16 bits)	Read /Write	<p>Stores heartbeat timer values between 0 and 255. Writing a value of 256 or greater will return a MODBUS error. These values are reserved for future use.</p> <ul style="list-style-type: none"> ■ Writing a non-0 value to this register will start an internal time. ■ Writing a 0 value to this register will halt the existing timer.

- While the timer is actively running, reads of this register return the value that was previously written.
- When the Master Heartbeat timer is running, any MODBUS requests received by the device from any MODBUS client restarts the currently running timer.
- If no requests have been received in a period equal to the configured timeout value, the timer expires and generates a warning-level log message.
- When the timer expires, reads of this register return 0.
- When the timer expires, reads of the 16th bit of register 0x004D (see below) return 1.
- Whenever any value is written to this register, the 16th bit of register 0x004D (slave heartbeat) is cleared and subsequent reads of that register will return 0 for that bit.
- The current state of the timer (Not Set, Active, Expired) as well as the configured timeout value is visible in the output of the command: **show scada modbus tcp server**

Log messages

- When the timer starts with a new non-0 value, a notice-level log with the following message is generated: **MODBUS: Master heartbeat timer started for X seconds**
- When the timer is halted, a notice-level log with the following message is generated: **MODBUS: Master heartbeat timer halted**
- When the timer expires after X seconds of not receiving any MODBUS requests the following message is generated: **MODBUS: Master heartbeat timer expired: X seconds with no requests**

Slave heartbeat register 0x004D

The purpose of this register is to act as a continually incrementing counter with a resolution of 100ms. This is stored in the lower 8 bits of the register. The upper 8 bits store a series of 'toggle bits', that alternate between values of 0 and 1.

Name	Length	Access	Description
0x004D	1 word (16 bits)	Read Only	Stores an incrementing counter so MODBUS clients can continually poll this register to confirm that the MODBUS server is alive, responsive and has not hung, and also tie specific scripted actions to be carried out when the toggle bits read a particular value.

- This register is broken into two sections, the lower 8 bits record a continually incrementing counter with a resolution of 100ms, so a value of 57 would represent 5.7 seconds have passed. Because this is only 8 bits of data, the value will wrap to 0 after every 25.5 seconds.
- The upper 8 bits are referred to as 'toggle bits' and their values automatically change between 0 and 1 at fixed intervals or in response to other events. These bits are numbered from least significant (rightmost) bit inclusive of the lower 8 bits. Refer to ["Register mapping" on page 16](#) for definition of bits 9 to 15.
- Bit 16 (0x8000) is tied to the Master Heartbeat Time register (0x004C) and has a value of 1 when the timer has expired. Writing any value to the Master Heartbeat Time register will clear the value of this bit. A MODBUS client can continually poll this bit to detect if the heartbeat timer has expired and thus detect whether the AlliedWare Plus device has actioned any scripts that are triggered from the heartbeat timer expiring.

Monitoring

Server information

To show the MODBUS TCP server information and statistics, use the command:

```
awplus(config)#show scada modbus tcp server
```

```
IE300#show scada modbus tcp server
SCADA MODBUS Summary Information

SCADA MODBUS is enabled for this device

SCADA MODBUS write access is disabled.

-----
TCP Server Characteristics:
  Port           :          502
  Max Connections :          5
  Permitted IP addresses:
    198.51.100.200
    2001:db8::1
-----
Statistics:
  Read Requests:
    coils           :          0
    discrete inputs :          0
    holding registers :          3
    input registers :          0

  Write Requests:
    single coil      :          0
    single register  :          1
    multiple coils   :          0
    multiple registers :          0

  Other Requests    :          0

  Exceptions:
    illegal function :          1
    illegal data address :          0
    illegal data value :          0
    slave device failure :          0
```

Table 1: Output descriptions for **show scada modbus tcp server**

OUTPUT	DESCRIPTION
SCADA MODBUS is enabled/disabled for this device	Indicates if the MODBUS TCP server is running.
SCADA MODBUS write access is enabled/disabled.	If disabled, only read MODBUS function codes can be used. If enabled, both read and write MODBUS function codes can be used
Port	The TCP port of the server. The port that clients must use to connect to the MODBUS TCP server.
Max Connections	The maximum limit of concurrent MODBUS TCP clients.
Permitted IP addresses	IP addresses of clients that are permitted to access MODBUS. If no IP addresses have been specified, there is no restriction on access to MODBUS and this section will not be displayed.
Read requests	Counts of total MODBUS requests for read function codes.
Write requests	Counts of total MODBUS requests for write function codes.

Table 1: Output descriptions for **show scada modbus tcp server** (continued)

OUTPUT	DESCRIPTION
Other requests	Counts of total MODBUS requests for all other function codes.
Exceptions	Counts of total MODBUS requests that have resulted in an exception.

Client information

To show the information and statistics of currently connected clients to the MODBUS TCP server, use the command:

```
awplus(config)#show scada modbus tcp server connections
```

```
IE300#show scada modbus tcp server connections
SCADA MODBUS Client Information
Connected client 1
  IP address      : 198.51.100.200
  Port           : 41617
  Uptime         : 739
Statistics:
  Read Requests:
    coils          :          0
    discrete inputs :          0
    holding registers :          2
    input registers :          0
  Write Requests:
    single coil     :          0
    single register :          0
    multiple coils  :          0
    multiple registers :          0
  Other Requests   :          0
  Exceptions:
    illegal function :          0
    illegal data address :          0
    illegal data value :          0
    slave device failure :          0

Connected client 2
  IP address      : 198.51.100.200
  Port           : 33229
  Uptime         : 16
Statistics:
  Read Requests:
    coils          :          0
    discrete inputs :          0
    holding registers :          1
    input registers :          0
  Write Requests:
    single coil     :          0
    single register :          1
    multiple coils  :          0
    multiple registers :          0
  Other Requests   :          0
  Exceptions:
    illegal function :          1
    illegal data address :          0
    illegal data value :          0
    slave device failure :          0
```

Table 2: Output descriptions for **show scada modbus tcp server connections**

OUTPUT	DESCRIPTION
IP address	Address of the client.
Port	The TCP port of the client.
Uptime	How long the client has been connected to the MODBUS TCP server in seconds.
Read requests	Counts of this client's MODBUS requests for read function codes.
Write requests	Counts of this client's MODBUS requests for write function codes.
Other requests	Counts of this client's MODBUS requests for all other function codes.
Exceptions	Counts of this client's MODBUS requests that have resulted in an exception.

Logging

By default, logging is enabled for MODBUS and each read or write request received by the MODBUS server will produce a human-readable translation of the request that is logged to syslog. While these log messages are convenient, they add additional overhead for each MODBUS request that is processed. Under high loads of MODBUS requests producing these log messages can impact overall system performance. You can disable MODBUS logging using the following command: **no scada modbus tcp server logging**.

Clear commands

Clear server statistics

To clear the MODBUS TCP server statistics, as shown in **show scada modbus tcp server**, use the command **clear scada modbus tcp server statistics**.

Force disconnect

To forcefully disconnect a connected MODBUS client from the MODBUS TCP server, use the command: **clear scada modbus tcp server connection**

The client is identified by their IPv4 or IPv6 address and TCP port number which can be seen in **show scada modbus tcp server connections**.

For example:

```
awplus#clear scada modbus tcp server connection 198.51.100.200 33229
```

How do MODBUS clients and servers communicate?

MODBUS clients and servers use register mapping to communicate with each other. MODBUS register mapping allows two devices to share the same protocol. MODBUS devices usually include a register map.

As described earlier in "[MODBUS TCP messages](#)" on page 5, clients must include a register address when they make a request for information. MODBUS devices usually include a Register Map which links to function operations. Modbus functions operate on register map registers to monitor, configure, and control module I/O.

The size of a MODBUS register address is measured in **words**. Each word is **16** bits (2 bytes).

For example, using an open source python modbus library, here is a request to read the configured port state for port1.0.1:

```
>>> result = client.read_holding_registers(0x5001, 1)
>>> print hex(result.registers[0])
0xf000
```

- The request is `read_holding_registers(0x5001, 1)`
- `0x5001` is the address to start reading from, and the `1` means read one word starting from the specified register address, i.e. write what's at that address.
- The result (in hex) is `0xf000`. This means that the admin state is up (f), the duplex is auto (the first 0), the polarity is auto (the second 0), and the speed is auto (the third 0).

Alternatively, to write the same value but with the admin state down, you would change the f to 0.

```
>>> result = client.write_register(0x5001, 0x0000)
>>> print result
WriteMultipleRegisterResponse (90,1)
```

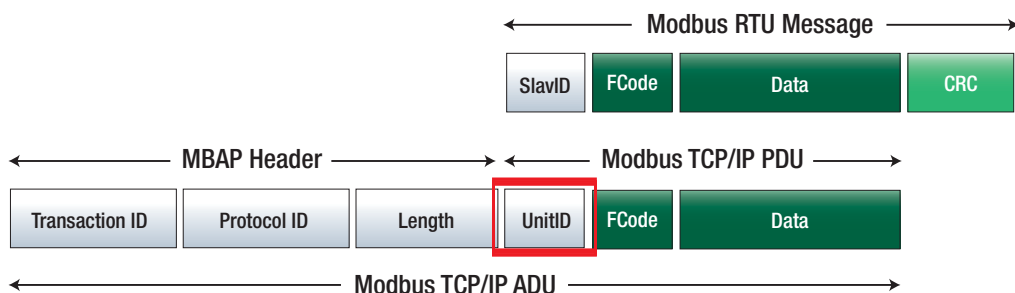
You can see the resulting status for port1.0.1 below:

```
23:22:06 x310 NSM[822]: Port down notification received for port1.0.1
23:22:07 x310 NSM[822]: Port down notification received for vlan1
IE300#show run int port1.0.1
!
interface port1.0.1
shutdown
switchport
switchport mode access
!
```

VCStack support for MODBUS

From software version 5.4.9-x onward, VCStack is supported for MODBUS.

AlliedWare Plus uses the Unit Identifier in the MODBUS packet to identify a stack member ID. The Unit identifier requests information for a specific stack member by setting the Unit Identifier as the stack member ID. If the Unit Identifier is zero then information for the stack Master is retrieved. If the device is not stacked, specify a value of 0 for this field.



Limitations

AlliedWare Plus uses the Unit Identifier to differentiate between stack members. Typically the Unit Identifier is not used in MODBUS over TCP but may be used when using serial RTU. If the Unit Identifier is being inspected by the network such as used by a MODBUS bridge or converted for RTU, then VCS support may not work correctly as the request may be rejected.

Register mapping

Table 3 lists the register mapping MODBUS uses for VCStack. From software version 5.5.0-1.1 onwards, version 5 register mapping is used. The changes from version 4 to 5 are as follows:

- Register 0x0048 previously displayed the total number of alarms in a stack, now it displays the total number of global alarms in stack.
- Register 0x0049 previously displayed if **vcs-member-output-control** was enabled/disabled, now it displays the total number of member alarms in a stack.
- Register 0x004A previously displayed the TCP connection uptime, now it displays if **vcs-member-output-control** is enabled/disabled.
- Register 0x004C previously displayed the master heartbeat time, now it is unused.
- Register 0x004D previously displayed the slave heartbeat time, now it is unused.
- Register 0x007A was previously unused, now it displays the TCP connection uptime.
- Register 0x007C was previously unused, now it displays the master heartbeat time.
- Register 0x007D was previously unused, now it displays the slave heartbeat time.

Table 3: Register mapping

ADDRESS (HEX)	SIZE	TYPE	ACCESS	DESCRIPTION
Stack Global System Information				
0x0000	1 word	UINT	R	Mapping Version
0x0001	32 words	ASCII	R	System Name
0x0021	32 words	ASCII	R	Software Version
0x0041	3 word	HEX	R	Stack System HW MAC Address
0x0044	1 word	HEX	R	Stack Members bitmap
0x0045	1 word	UINT	R	Total Number of Ports in the Stack
0x0046	1 word	UINT	R	Total Number of Faults in the Stack
0x0047	1 word	UINT	R	Total Number of Sensors in the Stack
0x0048	1 word	UINT	R	Total Number of Global Alarms in the stack
0x0049	1 word	UINT	R	Total Number of Member Alarms in the stack
0x004A	1 word	UINT	R/W	Alarm monitoring VCS member output control enabled
Heartbeat Information				
0x007A	2 words	UINT	R	TCP connection uptime (in seconds)
0x007C	1 word	HEX	R/W	Master heartbeat time (in seconds, up to 255s)
0x007D	1 word	HEX	R	Slave heartbeat
Global Alarms Information				
0x0080	1 word	ENUM	R	Alarm #1 Type
0x0081	2 words	HEX	R/W	Alarm #1 Config (output members 1-4)

Table 3: Register mapping (continued)

ADDRESS (HEX)	SIZE	TYPE	ACCESS	DESCRIPTION
0x0083	2 words	HEX	R/W	Alarm #1 Config (output members 5-8)
0x0085	1 word	BOOL	R	Alarm #1 Status
0x0086	1 word	ENUM	R	Alarm #2 Type
0x0087	2 words	HEX	R/W	Alarm #2 Config (output members 1-4)
0x0089	2 words	HEX	R/W	Alarm #2 Config (output members 5-8)
0x008A	1 word	BOOL	R	Alarm #2 Status
...additional alarms follow the same format as above up to 16 alarms				
0x00DA	1 word	ENUM	R	Alarm #16 Type
0x00DB	2 words	HEX	R/W	Alarm #16 Config (output members 1-4)
0x00DD	2 words	HEX	R/W	Alarm #16 Config (output members 5-8)
0x00DF	1 word	BOOL	R	Alarm #16 Status
Stack Member X System Information				
0x0120	1 word	UINT	R	Member X Total Board Count
0x0121	1 word	UINT	R	Member X Number of Ports
0x0122	1 word	UINT	R	Member X Number of Faults
0x0123	1 word	UINT	R	Member X Number of Sensors
0x0124	1 word	UINT	R	Member X Number of Alarms
Board Information for Member X				
0x0200	1 word	ENUM	R	Board #1 board class
0x0201	1 word	UINT	R	Board #1 slot number
0x0202	32 words	ASCII	R	Board #1 board name
0x0222	32 words	ASCII	R	Board #1 serial
0x0242	3 word	HEX	R	Board #1 HW MAC
0x0245	1 word	ENUM	R	Board #2 board class
...additional boards of member X follow the same format as above up to 32 boards				
0x0A9D	3 word	HEX	R	Board #32 HW MAC
Sensor Information for Member X				
0x1000	1 word	ENUM	R	Sensor #1 type
0x1001	2 word	FLOAT	R	Sensor # 1 reading
0x1003	1 word	ENUM	R	Sensor # 1 units
0x1004	1 word	BOOL	R	Sensor # 1 fault
0x1005	1 word	ENUM	R	Sensor # 2 type
0x1006	2 word	FLOAT	R	Sensor # 2 reading
0x1008	1 word	ENUM	R	Sensor # 2 units
0x1009	1 word	BOOL	R	Sensor # 2 fault
...additional sensors follow the same format as above up to 128 sensors				
0x127B	1 word	ENUM	R	Sensor #128 type

Table 3: Register mapping (continued)

ADDRESS (HEX)	SIZE	TYPE	ACCESS	DESCRIPTION
0x127C	2 word	FLOAT	R	Sensor #128 reading
0x127E	1 word	ENUM	R	Sensor #128 units
0x127F	1 word	BOOL	R	Sensor #128 fault
Per-stack-member Alarms Information for Member X				
0x3000	1 word	ENUM	R	Alarm #1 type
0x3001	2 words	HEX	R/W	Alarm #1 config (output members 1-4)
0x3003	2 words	HEX	R/W	Alarm #1 config (output members 5-8)
0x3005	1 word	BOOL	R	Alarm #1 Status
0x3006	1 word	ENUM	R	Alarm #2 Type
0x3007	2 words	HEX	R/W	Alarm #2 config (output members 1-4)
0x3009	2 words	HEX	R/W	Alarm #2 config (output members 5-8)
0x300A	1 word	BOOL	R	Alarm #2 Status
...additional alarms follow the same format as above up to 128 alarms				
0x32FA	1 word	ENUM	R	Alarm #128 type
0x32FB	2 words	HEX	R/W	Alarm #128 config (output members 1-4)
0x32FD	2 words	HEX	R/W	Alarm #128 config (output members 5-8)
0x32FF	1 word	BOOL	R	Alarm #128 status
Port Information for Member X				
0x5000	1 word	HEX	R	Port #1 Current State
0x5001	1 word	HEX	R/W	Port #1 Configured State
0x5002	1 word	HEX	R/W	Port #1 PoE Configuration
0x5003	1 word	UINT	R	Port #1 PoE Data Pairs Current Power Usage
0x5004	1 word	UINT	R	Port #1 PoE Spare Pairs Current Usage
0x5005	4 word	UINT	R	Port #1 Input Bytes
0x5009	4 word	UINT	R	Port #1 Output Bytes
0x500D	1 word	HEX	R	Port #2 Current State
0x500E	1 word	HEX	R/W	Port #2 Configured State
0x500F	1 word	HEX	R/W	Port #2 PoE Configuration
0x5010	1 word	UINT	R	Port #2 PoE Data Pairs Current Power Usage
0x5011	1 word	UINT	R	Port #2 PoE Spare Pairs Current Usage
0x5012	4 word	UINT	R	Port #2 Input Bytes
0x5016	4 word	UINT	R	Port #2 Output Bytes
...additional ports follow the same format as above up to 400 ports				
0x6443	1 word	HEX	R	Port #400 Current State
0x6444	1 word	HEX	R/W	Port #400 Configured State
0x6445	1 word	HEX	R/W	Port #400 POE Configuration
0x6446	1 word	UINT	R	Port #400 POE Data Pairs Current Power Usage

Table 3: Register mapping (continued)

ADDRESS (HEX)	SIZE	TYPE	ACCESS	DESCRIPTION
0x6447	1 word	UINT	R	Port #400 POE Spare Pairs Current Usage
0x6448	4 word	UINT	R	Port #400 Input Bytes
0x644C	4 word	UINT	R	Port #400 Output Bytes
0x6450	-	-	-	Start of unused address space

Register value definition

The register representing the **Stack Members Bitmap** (0x0044) has the following definition:

- Bit numbering starts from the lower bits (right most)
- The first bit represents that stack member 1 is present in the Virtual Chassis Stack.
- The second bit represents that stack member 2 is present in the Virtual Chassis Stack.

And so on.

The register representing the **board class** (0x0200, x0245...) has the following register addresses definition:

1. System base board. The main board of the system.
2. Expansion module.
3. Power supply.
4. Fan module.
5. Passive chassis backplane.
6. Sub-board of an expansion module.

The register representing the **sensor type** (0x1000, 0x1005...) has the following definition:

1. Voltage measurement
2. Temperature measurement
3. Fan speed measurement
4. Current measurement
5. Power measurement
6. Presence indication
7. Power supplied indication
8. Input state indication
9. External alarm input contact state indication
10. External alarm relay output state indication
11. Fault LED flashing indication

The register representing the **sensor units** (0x1003, 0x1008...) has the following definition:

1. Volts
2. Degrees Celsius
3. RPM
4. Amps
5. Watts

The register representing the **alarm type** (0x0081, 0x0087... and 0x3000, 0x3006...) has the following definition:

1. External PSU
2. EPSR
3. External contact input
4. Port link down
5. Loop detect
6. Main PSE failure
7. Port PoE Failure
8. Temperature
9. G8032
10. UFO

The register representing **alarm configuration for stack members 1-4** (0x0081, 0x0087... and 0x3001, 0x3007...) has the following definition:

- Bit numbering starts from the upper bits (left most)
- The first bit represents the LED config.
- The second bit represents the first relay config on the first stack member.
- The third bit represents the second relay config on the first stack member.
- The fourth bit represents the third relay config on the first stack member.
- The fifth bit represents the LED config on the second stack member.
- The seventh bit represents the second relay config on the second stack member.
- The eighth bit represents the third relay on the second stack member.
- And so on.
- 0 for not configured and 1 for configured.

Note:

- Writing bits beyond the first stack member when **vcs-member-output-control** is disabled will be ignored.
- Writing bits to a member that is not part of the stack will be ignored.

The register representing **alarm configuration for stack members 5-8** (0x0082, 0x0088... and 0x3002, 0x3008...):

- The bit definition of for this register follows the same pattern as the alarm configuration for stack members 1-4, except that the nibbles represent stack members 5, 6, 7 and 8 respectively rather than stack members 1, 2, 3 and 4.

The register representing the **port current state** (0x5000, 0x500D...) has the following definition:

- The bits in the first byte (0xF000) are the link state. 0 for link down and F for link up.
- The bits in the second byte (0x0F00) are the duplex. 0 for auto, 1 for full, 2 for half.
- The bits in the third byte (0x00F0) are the polarity. 0 for auto, 1 for MDI, 2 for MDI-X.
- The bits in the fourth byte (0x000F) are the speed. 0 for auto, 1 for 10Mb/s, 2 for 100Mb/s, 3 for 1Gb/s, 4 for 10Gb/s.

The register representing the **port configured state** (0x5001, 0x500E...) has the following definition:

- The bits in the first byte (0xF000) are the admin state. 0 for admin down and F for admin up.
- The bits in the second byte (0x0F00) are the duplex. 0 for auto, 1 for full, 2 for half.
- The bits in the third byte (0x00F0) are the polarity. 0 for auto, 1 for MDI, 2 for MDI-X.
- The bits in the fourth byte (0x000F) are the speed. 0 for auto, 1 for 10Mb/s, 2 for 100Mb/s, 3 for 1Gb/s, 4 for 10Gb/s

The register representing **port PoE configuration** (0x5002, 0x500F...) has the following definition:

- The bits in the first and second bytes (0xFF00) are the data pair config. FF for enabled or 00 for disabled.
- The bits in the third and fourth bytes (0x00FF) are the spare pair config. FF for enabled or 00 for disabled.

If the device does not use 4 pair PoE or it is not configured on the port then the spare pair bytes are ignored. The registers representing port PoE data and spare power usage are measure in milliwatts.