Model 5100 Communications Guide

The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions. See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

Model Number	Hardware Revision	Firmware Revision
5101/5102/5103/5104	B, C, and E	>= 4.05.48

WARNING: Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.

TABLE OF CONTENTS

1.0 Communications Summary

With the release of the 5100 firmware revision 4.05 and above, a number of new features are available or have been enhanced. Many of these features are in the area of communications, while a number of significant ones allow for greater programming flexibility. This manual's focus is on those features relevant to the area of communications, some of which are listed below

- ***Communications***
  - (2) Serial ports capable of the CTNet Binary protocol, CTC ASCII Protocol, raw mode, COM1 also supporting the Modbus RTU/ASCII Master and Slave protocols.
  - COM1 and COM2 configurable, stop bits, data bits, and parity (COM2 only on manufacturing builds after April 15, 2003).
  - Telnet Server for remote administration interface
  - FTP Server
  - Modbus/TCP RTU Master and Slave
  - UDP Peer to Peer
  - TCP client/server raw socket interface, bidirectional
  - CTNet Binary protocol
  - Up to 7 serial ports, 2 local and 5 virtual TCP to terminal servers or host applications
  - Configurable connection throttling to enhance overall system performance
  - String formatted output messages with embedded register values from within Quickstep (printf format).
  - SNTP Time Server synchronization for real time clock.
  - Virtual IO between controllers over Ethernet
  - DHCP support
  - DNS name registration via DHCP
  - 'C' Programming for custom protocols

1.1 Serial Communications Overview

The controller contains two RS-232 serial ports. These ports support numerous communications protocols, many of which are detailed elsewhere within this document. This section is meant as a general overview.

1.1.1 Port Settings

The default communication settings for the two serial ports are:

Baud Rate: 19200
Data Bits: 8

Parity: None
Stop Bits: 1

All parameters may be changed using available registers. Use register 12000 to select either port by storing a 1 or 2. Set the following registers based on the configuration desired:
Register 12301 to select the baud rate as follows:

2 – 1,200
3 – 2,400
4 – 4,800
5 – 9,600
6 – 19,200 (default)
7 – 38,400

Register 12308 to select the parity as follows:

0 – None (default)
1 – Odd
2 – Even

Register 12309 to select the stop bits as follows:

1 – Stop bit on transmit (default)
2 – Stop bits on transmit

Register 12309 to select the data bits as follows:

7 – Data bits (default, not including parity)
8 – Data bits (not including parity)

For example the following Quickstep instructions will change the baud rate on port 1 to 9600 Baud:

```
store 1 to Reg_12000
store 5 to Reg 12301
```

Note: Only Manufacturing builds after April 15, 2003 are capable of supporting parity, stop bit and data bit configuration on COM2. To verify the capability reference U6 on the main board. The component AT90S8515 (older designs) can not support port configuration, new designs, using the ATMega8115 do support configuration.

1.1.2 ASCII Protocol

Each serial communications port supports a number of protocols. The default is the ASCII and CTC Binary Protocol, controlled by register 12320. By setting register 12000 to the active port number, register 12320 will display the active protocol on that port, with 0 being the default.

The ASCII Protocol allows access to the various data points within the controller; digital inputs and outputs, analog input and outputs, registers, counters data table and flags. In addition commands may be issued to start/stop/reset the controller. There are two modes to the ASCII Protocol, computer and terminal (default). The computer mode simply terminates strings with a <CR> (carriage return, 0D Hex), while the terminal mode uses a <CR> <LF> (carriage return, linefeed 0A Hex) combination and sends a leading <LF>, prior to transmission.

In the following command descriptions <CR> equals 0D Hex and <LF> equals 0A Hex. <??? number/value> includes the greater than and less than signs for clarity, they are not in the data stream. Digital inputs and output values use the number 0 for off and 1 for on, flags are 0 for clear and 1 for set.

Initiate computer mode

Send - PC<CR>
Response – PC0<CR>

Initiate terminal mode

Send – PT<CR>
Response - <LF>PT<CR><LF>

Read a digital output

Send – O<output number><CR>
Response:
Computer mode - <output number><CR>
Terminal mode - <LF><output value><CR><LF>

Write a digital output

Send – O<output number>=<new value> <CR>
Response:
Computer mode - <CR>
Terminal mode - <LF>

Read a digital input

Send - I<input number><CR>
Response:
Computer mode – <input value><CR>
Terminal mode - <LF><input value><CR><LF>

Read an analog output

Send - AO<output number><CR>
Response:
Computer mode – <output value><CR>
Terminal mode - <LF><output value><CR><LF>

Write an analog output

Send – AO<output number>=<new value> <CR>
Response:
Computer mode - <CR>
Terminal mode - <LF>

Read an analog input

Send - AI<input number><CR>
Response:
Computer mode – <input value><CR>
Terminal mode - <LF><input value><CR><LF>

Read a counter/register

Send - R<counter/register number><CR>
Response:
Computer mode – < counter/register number ><CR>
Terminal mode - <LF>< counter/register number ><CR><LF>
Note: Register read/write commands can be chained together using a ';' as a separator, each command will be responded to uniquely.
Example: R1000=5;R1005;R1006<CR>

Write a counter/register

Send - R<counter/register number>=<new value><CR>
Response:
Computer mode – <CR>
Terminal mode - <LF>
Note: Register read/write commands can be chained together using a ';' as a separator, each command will be responded to uniquely.
Example: R1000=5;R1005;R1006<CR>

Read a flag

Send - F<flag number><CR>
Response:
Computer mode – <flag value><CR>
Terminal mode - <LF><flag value><CR><LF>

Write a flag

Send - F<flag number>=<new value><CR>
Response:
Computer mode – <CR>
Terminal mode - <LF>

Read a data table location

Send - D<row number>,<column number><CR>

Response:
Computer mode – <table value><CR>
Terminal mode - <LF><table value><CR><LF>

Write a data table location

Send - D<row number>,<column number>=<new value><CR>
Response:
Computer mode – <CR>
Terminal mode - <LF>

Issue Start Command to Controller

Send - +<CR>
Response:
Computer mode – <CR>
Terminal mode - <LF>

Issue Stop Command to Controller

Send: -<CR>
Response:
Computer mode – <CR>
Terminal mode - <LF>

Issue Reset Command to Controller

Send - *<CR>
Response:
Computer mode – <CR>
Terminal mode - <LF>

Returned Error Messages

Number too small – If an input, output, register, or flag number is specified as zero then the controller sends the following error message:
Computer mode - <less than sign,< > <bell, 07H><CR>
Terminal mode - <LF><less than sign,< ><bell, 07H><CR><LF>

Number too large – If an input, output, register, or flag number is specified that is greater than the number supported, then the controller sends the following error message:
Computer mode - <greater than sign,> > <bell, 07H><CR>
Terminal mode - <LF><greater than sign,> ><bell, 07H><CR><LF>

Protocol error – If a "P" command (protocol) is not in the correct format then the controller will send the following error message:
Computer mode – P<bell, 07H><CR>
Terminal mode - <LF>P<bell, 07H><CR><LF>

Syntax error – If the controller can not make any sense of the command, then it sends the following message:
Computer mode - ?<bell, 07H><CR>
Terminal mode - <LF>?<bell, 07H><CR>

1.2 Ethernet Communications

The 5100 series controllers can be configured to communicate over Ethernet using one of several transport protocols: CTNet, UDP, and TCP.

CTNet

CTNet is a proprietary non-routable protocol typically used for legacy communications to the 2700 products. It tends to be faster than UDP or TCP/IP due to the lack of processing overhead but like UDP, lacks acknowledgement of each packet.

UDP

User Datagram Protocol is used to send packets across an IP Network in an unreliable manner, no packet acknowledgement. The protocol is fully routable across the internet, unlike CTNet. It is the preferred interface for many products when performance is required and the application can perform error recovery. The 5100 supports UDP packet transport for peer to peer communications, virtual IO, CTCMon, and CTServer products.

TCP

Transmission Control Protocol is used to establish connection-oriented, sequenced, and error free sessions over an IP Network. The protocol is fully routable across the internet, unlike CTNet, and each data packet is acknowledged when received correctly by the receiver. Retransmission of lost packets are built into the protocol. Typical retry timers of 250 milliseconds limit the uses of TCP in a real-time controller. The 5100 supports TCP packet transport for FTP, Telnet, Modbus TCP Master/Slave, RAW client/server connections, CTCMon, and CTServer products.

When using any of these protocols it is important to note that whenever the 5100 is placed on a network, it should be connected to a switch, not a hub. Refer to Section 7.5 for further details. The following section details the initialization required prior to network operation.

1.2.1 Configuring CTNet Node

To use CTNet, a valid CTNet node number between 1 and 32767 must be set. To use UDP protocol, the controller must be set up with a TCP/IP address, subnet mask, and optional gateway.

The CTNet node number, of the controller, is stored in register 20000. Simply write the Node number to register 20000, and then write a 1 to register 20096, and cycle power on the controller for the change to be accepted.

```
Store 21 to Reg_20000
Store 1 to Reg_20096
```

1.2.2 Configuring IP Address Manually

If you are not using DHCP to automatically obtain your IP Address then the TCP/IP address is configured statically, as follows:

Example IP Address 168.254.132.34 (example)
Example Subnet Mask: 255.255.255.0 (typical)
Example Gateway 168.254.132.88 (example)

The actual values to use will depend on the network that the controller is connected to. Contact your IT department to determine acceptable addresses for your network.

Registers 20048 to 20051 are the 4 parts of the IP Address:

```
store 168 to Reg_20048
store 254 to Reg_20049
store 132 to Reg_20050
store 34 to Reg_20051
```

Registers 20064 to 20067 are the 4 parts of the Subnet Mask:

```
store 255 to Reg_20064
store 255 to Reg_20065
store 255 to Reg_20066
store 0 to Reg_20067
```

Registers 20080 to 20083 are the 4 parts of the Gateway Address (optional). A gateway is only required if the controller needs to communicate over a Wide-Area Network (WAN). If not using a gateway then set these registers to 0 (default). The controller can talk to devices on a Local Area Network without using a gateway, but not over the Internet or outside its subnet.

```
store 168 to Reg_20080
store 254 to Reg_20081
store 132 to Reg_20082
store 88 to Reg_20083
```

To save the IP address and all other modified parameters to non-volatile memory:

```
store 1 to Reg_20096
```

Finally, cycling power to the controller to activate the new IP information active.

The IP address can be set up through a Quickstep program or with CTC Monitor. Note that if you set the IP Address registers to 0, then write 1 to Reg_20096, cycle power, the controller will use DHCP to obtain its network information, automatically. You will be aware that the controller is attempting to connect to a DHCP server when the FAULT LED is flashing repeatedly, at a high rate (100ms/second). The FAULT LED will stop flashing once the 5100 has obtained an IP address from a DHCP server. While searching for a valid DHCP address, serial port CTC Monitor access will be available but Quickstep and Ethernet communications will be disabled. Once an IP address is available the 5100 will continue to boot, initializing the network and starting Quickstep application software.

1.2.3 Configuring the IP Address automatically with DHCP

The controller is capable of retrieving its IP information automatically, from a DHCP server, RFC 2131. The Dynamic Host Configuration Protocol (DHCP) is a communication protocol that lets network administrators automate assigning of IP addresses within a network.

All devices (computers, controllers, etc.) which reside on a TCP/IP network must have an IP address assigned. Without DHCP, the IP address must be entered manually at each device, such as detailed in the previous section. If devices move to another location in another part of the network, a new IP address must be entered. DHCP allows a network administrator to supervise and distribute IP addresses from a central point and automatically assigns a new IP address when a computer is plugged into a different location on the network. DHCP also provides other services beyond that of just an IP address. It provides Domain Name Service (DNS) server addresses, gateway information, Simple Network Time Protocol (SNTP, section 6.0) servers, etc., thus allowing for fully automatic configuration of the controller IP parameters.

DHCP uses the concept of a "lease" or amount of time that a given IP address will be valid for a computer. The lease time can vary depending upon how long a user is likely to require the network connection at a particular location. DHCP also supports static addresses for devices that need a permanent IP address.

DHCP is enabled by default in the controller. At power up, the controller will request to use whatever IP address is set in the 20048 block (except 0.0.0.0 which enables DHCP), the DHCP server will either allow it or supply a new IP address. This final address will temporarily be written to the 20048 block, but not permanently. Although not stored permanently it is still the active IP address for the system. Only the user or Quickstep can make this IP address permanent, by storing a 1 to register 20096. If you do not care to use DHCP, it can only be disabled by setting an actual IP address and subnet mask (Section 1.2.2).

1.2.4 Setting the Controller's DNS Name

When the controller communicates with a DHCP server it also requires a unique system name that is typically used for DNS resolution (assuming the server is using dynamic DNS). Presently this name is derived from the controller's serial number, placing "CTC_BF_", before the number. For example if the serial number was 100-52801 then the DNS name entry for the controller would become CTC_BF_10052801. User settable names are also possible and may be set using the 'set systemname <name>' command within the telnet administration screen, followed by writing a 1 to register 20096 (to save change), and rebooting the controller.

Note that many software packages and other devices with CTC communications drivers do not have the capability to identify controllers by name, only by IP Address. Depending on how your network is configured, DHCP may change the IP address of the controller without warning, causing devices and software to lose connection or connect to the wrong controller. In this case, it is better to manually assign a static IP address to the controller. The network administrator should be contacted prior to assigning any IP address, to avoid conflicts.

1.2.5 Communicating to the Controller Using CTNet

CTNet is a lightweight non-routable Ethernet protocol used by legacy CTC controllers. It is recommended that UDP be used, instead, whenever possible, since it is routable.

In order to communicate with the Controller from a PC using CTNet protocol, the WinPCap driver must be installed on the PC and an updated ctccom32v2.dll file must be installed in the Windows system32 directory.

The latest version of the WinPCap driver may be downloaded from the customer care section of CTC's website www.ctc-control.com. Compatibility information will be included with the download. Currently Windows 95, 98, ME, NT4, 2000, and XP are supported.

To install the driver:

1. First, uninstall any previously installed CTNet drivers including CTC Transport and CTC Packet Driver. If you have not previously installed these drivers, this step can be skipped. DO NOT INSTALL WinPCap OVER AN EXISTING CTNet DRIVER.
2. Double click the WinPCap.exe file and run through the installation program.
3. In your Windows system32 directory (typically Windows\system for Windows 95, 98, and ME and WINNT\system32 for Windows NT/2000/XP) replace the existing ctccom32v2.dll file with the file included with the WinPCap download.
4. Restart the PC.

Once the driver is installed, CTC Monitor 2.8 or later can be used to communicate to the controller. Every controller on the network must have a unique node number, and each PC based connection must use a unique Host node number.

2.0 TCP/IP Raw Sockets

Up to 5 TCP Client/Server RAW Socket sessions are supported by the controller. These socket sessions provide a virtual pipe, with no formatting of data. To the controller they merely appear as another serial port, even though the connected device can reside virtually anywhere on a network connection. This interface is extremely useful for connection to external programs, such as Visual Basic or Ethernet based terminal servers such as the Lantronix or Newport devices described below.

2.1 TCP Client

A TCP Client RAW Socket session is when the host computer runs a TCP Server and the controller connects to it. Typically a well-known IP address and public TCP port number is available for this connection. Once the connection is made any data sent to the actively selected serial port (12000 register) is sent to the host and anything sent by the host to the controller is placed in its receive buffer, exactly like an actual serial port. In order to initiate a connection a number of registers must be configured.

The RAW Socket session register blocks are based 22000 and extend to 22049, one block for each serial port supported. The actual block used has nothing to do with the serial port itself when referenced from Quickstep, it is configurable as a parameter within the block. Blue Fusion Controllers, Models 5101, 5102, 5103, and 5104 have 2 physical serial ports (COM1=1, COM2=2, 0 not used) within the controller. They can also access virtual serial ports 3 to 7 are virtual ports that may be assigned as desired. Remember that Server connections will use the next available port when allowing connections from a host client therefore it is important to reserve your port first prior to enabling a Server register block.

Registers are defined based on their offset from their base, repeating after each 10. Therefore beginning at register 22000:

REGISTER	*DESCRIPTION*	*USE*
22XX0	Controller Serial port ID register	Enter 3 to 7 for virtual port identifier.
22XX1	Client/Server register	To initiate a connection set this register to 0, if the controller is a server, set to a 1. Therefore, in the case of this section, we are a client, this would be a 0, we initiate the connection.
22XX2	IPA register	Most significant octet of IP

		Address to connect to.
22XX3	IPB register	
22XX4	IPC register	
22XX5	IPD register	Least significant octet of IP Address to connect to.
22XX6	TCP Port Connection register	TCP Port to connect to (client) or listen on (server).
22XX7	Connection Status register	On read, -1 = not initialized, 0 = offline, 1 = online, write a 1 to initiate connection or start server thread.
22XX8	Index register	Provides access to special purpose registers and general data access. Recommend using serial port buffer for data, not this interface but available to mimic the peer to peer interface.
22XX9	Data Array register	R/W access to register selected by the Index register.

An example for a script program to initialize a connection to a host at IP address (Lantronix Cobox example is detailed in section 2.3) 12.40.53.185 and TCP port 3001 is shown below, note the controller Serial Port ID Register, 22000 must be setup first:

```
22000 = 3       # setup this client connection as controller port 3
22001 = 0       # set that we are the client, initiating connection
22002 = 12      # most significant octet of ip address 12.40.53.185
22003 = 40
22004 = 53
22005 = 185     # least significant octet of ip address 12.40.53.185
22006 =  3001   # TCP port to attempt connection to
22007 = 1       # To initiate a connection write a 1 to the status register then read
                # it until it is a 1
                # which means connected, 0 is offline, -1 is not initialized.
```

Once register 22007 is read as a 1 then port 3 will appear as a standard serial port to a Quickstep application. As with any serial port the port must be selected first, by writing the port number to register 12000, prior to transferring data or initiating commands. The port is available for reading and writing upon connection to the host., register 22007 = 1. Should a connection ever be lost, 22007 will contain a 0 and a read of 12000 (Message status register) will return a 1, indicating transmitter busy, or in this case, offline. With TCP the transmitter will never be busy unless offline. The controller will periodically retry the client connection.

2.2 TCP Server

A TCP Server RAW Socket session is when the host computer is the client, connecting to the controller on a public TCP port number. Once the connection is made any data sent to the actively selected serial port is sent to the host and anything sent by the host is placed in the receive buffer, exactly like a controller serial port. In order to allow a server to be active the same registers as detailed in Client, must be configured except a 1 is placed in register 22XX1 and our port number to listen on is stored in 22XX6:

Registers are defined based on their offset from their base, repeating after each 10. Therefore beginning at register 22000:

REGISTER	DESCRIPTION	USE
22XX0	Controller Serial port ID register	Enter 3 to 7 for virtual port identifier.
22XX1	Client/Server register	To initiate a connection set this register to 0, if the controller is a server, set to a 1. Therefore, in the case of this section, we are a server, this would be a 1, we listen for the connection.
22XX2	IPA register	Most significant octet of IP Address to connect to.
22XX3	IPB register	
22XX4	IPC register	
22XX5	IPD register	Least significant octet of IP Address to connect to.
22XX6	TCP Port Connection register	TCP Port to connect to (client) or listen on (server).
22XX7	Connection Status register	On read, -1 = not initialized, 0 = offline, 1 = online, write a 1 to initiate connection or start server thread.
22XX8	Index register	Provides access to special purpose registers and general data access. Recommend using serial port buffer for data, not this interface but available to mimic the peer to peer interface.
22XX9	Data Array register	R/W access to register selected by the Index register.

A server thread will be launched as soon as a 1 is written to the status register. Note that only one connection is allowed at a time since all information is directed to and from a controller virtual serial port. If more than one connection attempt is made to

the same port number defined in the configuration block, it will be initially accepted and then rejected.

2.3 Lantronix CoBox/Xpress interface Example

The Lantronix CoBox-DR1-IAP or Xpress-DR-IAP (Device Server (www.lantronix.com) is one of several serial to Ethernet converter devices which will work with the controller using the TCP RAW Client socket protocol. To the controller this device is communicated to over TCP port 3001 and becomes a simple virtual serial port to Quickstep. It operates exactly as a resident local port, supporting the same communication protocols. Communications is tunneled over the network to the device. Even a serial port version of CTMon or a CTC 4010 User Interface can be connected and run over this interface, allowing for easy port expansion. Modbus is also supported.

By encapsulating serial data and transporting it over Ethernet, devices such these, allow virtual serial links to be established over Ethernet and distributed virtually anywhere within a plant or global enterprise.

Figure 2.1: Lantronix CoBox Serial to Ethernet Converters

3.0 Modbus

The Modbus Protocol is a messaging structure developed by Modicon in 1979. It is used for master-slave/client-server communications between intelligent devices and has become an industry standard. Details of the protocol may be found at the web site www.modbus.org. This protocol allows a master to periodically poll the controller to collect the desired information. Modbus supports two major flavors of

data representation, RTU and ASCII. RTU is a more compact protocol, consisting of binary characters while ASCII represents each binary nibble as a separate character, hence doubling the length of transmissions. RTU is also more secure in that it includes a CRC-16 at the end of the message while ASCII only has a single LRC. **The CTC 5100 controllers support Modbus Master/Slave TCP RTU, Modbus Master Serial RTU/ASCII, and Modbus Slave Serial RTU/ASCII.**

Tools used to test the protocol are available from a number of sources. The 5100 controller was tested using those available from www.win-tech.com, namely their ModScan32 for RTU/ASCII Slave testing and ModSim32 for Master.

3.1 Modbus Slave RTU TCP & RTU/ASCII Serial

A polling master can drive a slave controller using the Modbus protocol. The 5100 controller supports slave mode both over an Ethernet TCP connection and/or a serial connection. Modbus allows for interfaces to such things as coils, analog, register, etc. Since the 5100 controller is able to access all of its resources via its register interface, typically only the Holding Register commands are used: Write Single Register (function code 0x06), Write Multiple Registers (function code 0x10), and Read Holding Registers (0x03). Alternatively, the "Read Input Discrete" (maps to digital in modules), "Read coils", and "Write Single Coil" (maps to digital output modules) are supported.

	Function codes		
	Code	Sub code	(hex)
Read input discrete	02		02
Read coils	01		01
Write single coil	05		05
Write multiple coils	15		0F
Read input register	04		04
Read multiple registers	03		03
Write Single register	06		06
Write multiple registers	16		10
Read/write multiple registers	23		17
Mask write register	22		16
Read file record	20	6	14
Write file record	21	6	15
Read device identification	43	14	2B

Figure 3.1: Modbus Function codes from Modbus.org (highlighted yellow are supported by 5100)

You should also note that Modbus registers are 16 bits in width and that of the 5100 controller are 32 bits, since Modbus is Big Endean, this means when reading register 1 in the 5100 controller, the high 16 bits equates to Modbus register 1 and the low 16 bits to Modbus register 2. Modbus register 3 would be the high 16 bits of register #2, and so on. There are limitations to the number of registers that can be read by a polling master at one time:

Modbus RTU TCP – 120 Modbus 16 bit registers (60 5100 registers).
Modbus RTU Serial - 120 Modbus 16 bit registers (60 5100 registers).
Modbus ASCII Serial - 56 Modbus 16 bit registers (28 5100 registers).

This maximum is a limitation imposed by the Modbus TCP specification, not the controller since it limits receive buffers to 255 bytes.

Modbus TCP Slave is always enabled and available for requests on TCP port 502 (standard). Either a Quickstep program or other means **must manually enable Modbus RTU/ASCII Serial**. This is done simply by writing a 3 to the "Serial Active Protocol Selection" Register, **12320**, for RTU, or a 4 for ASCII. Prior to enabling it is recommended that the 5100 controller Modbus Unit/Device address also be set, using register 12321. Should a non-volatile controller wide default Modbus address be desired, set register 12322 with the address followed by a write to register 20096. Differences between TCP and serial implementations are detailed in section 3.1.1.

As a demonstration of the functionality of the Modbus RTU TCP/Slave interface this section details the interface of Win-Tech's ModScan32 software and how it applies with regards to the 5100 controller. As mentioned before, CTC only supporta the Holding Register interface. Upon installation of ModScan32 a screen such as Figure 3.3 will appear. Note that the 'Address' field is set to 1, but the display screen starts at 40001. This is Modbus nomenclature. 'Address' of 1 is the same as the upper 16 bits of the controller register 1. Note 'Length' is set to 50 (120 max), Device ID is ignored since TCP is point to point (Device ID is not ignored when the controller operates as a Master or serial slave, only when in TCP slave mode).

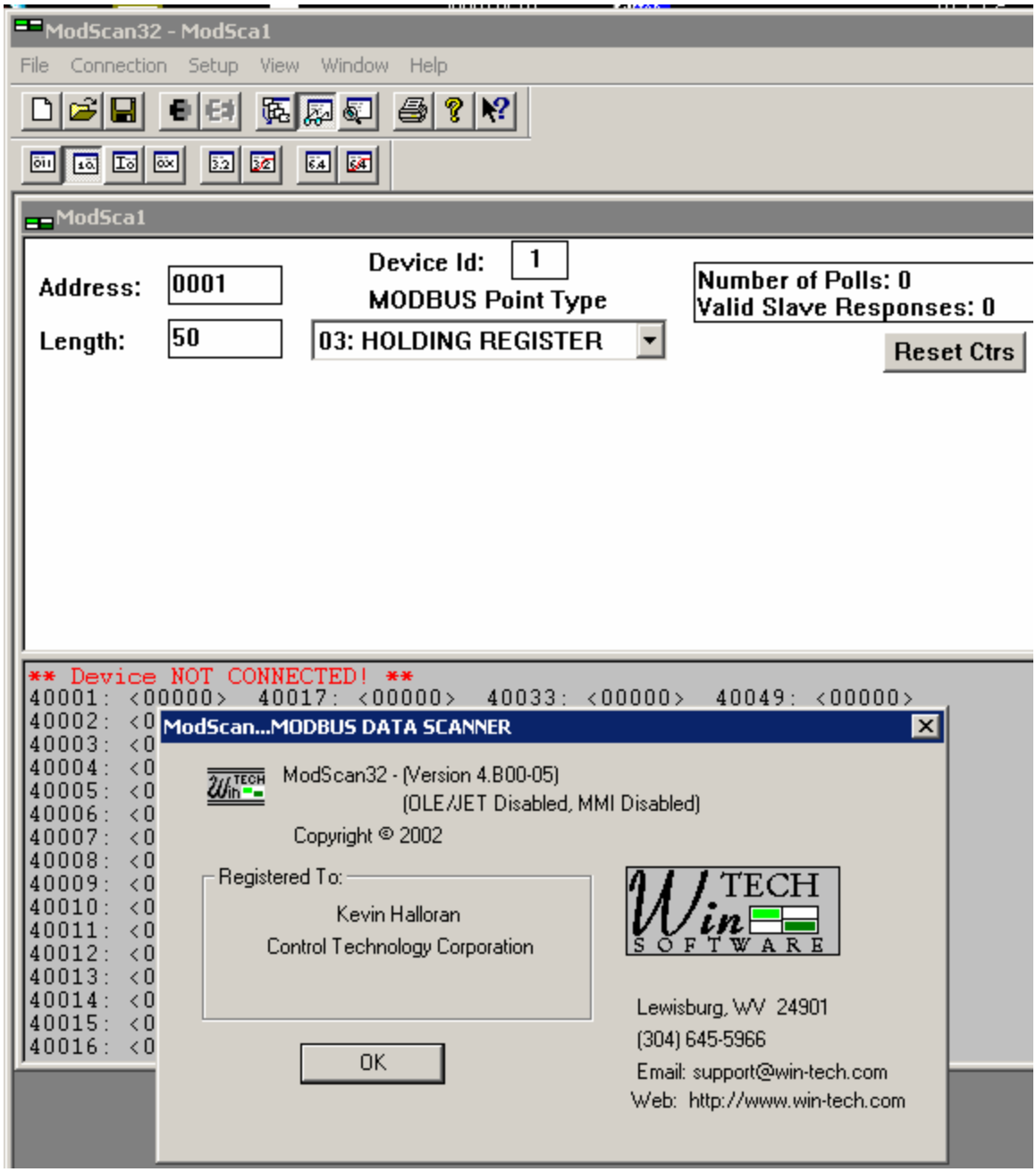


Figure 3.3: ModScan32 Master Scanning Program (only Holding Register supported)

Figure 3.4 shows the setup for an interface to a controller with a TCP address of 12.40.53.199 and the Modbus Slave running a server on the standard port of 502:

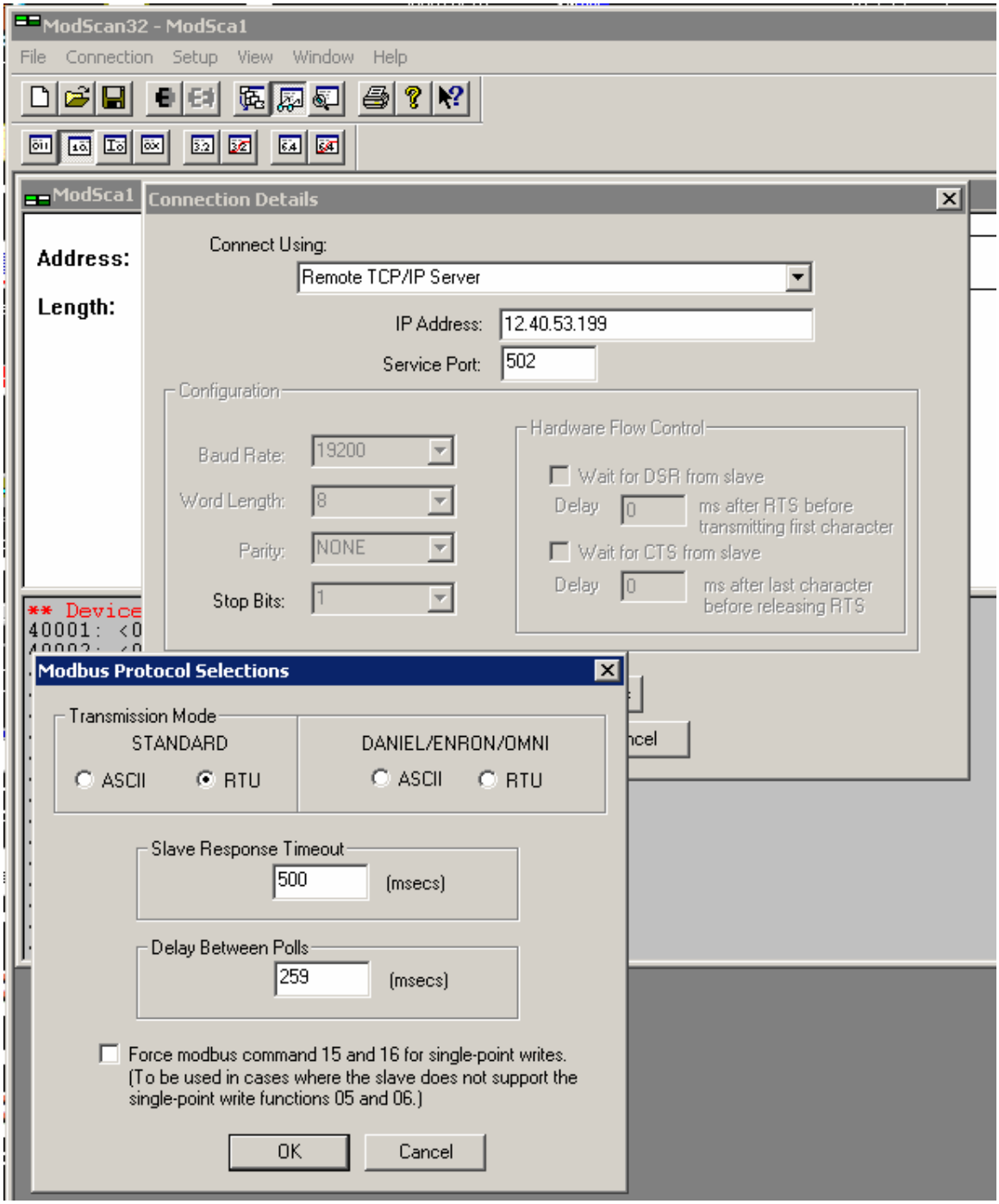


Figure 3.4: ModScan32 Master Scanning Program TCP Connection Setup

In order to do a single register write to a Modbus 16 bit register double click that register. Below shows changing Modbus register 40002 (Address 2) to a value of 5, this would translate to the lower 16 bits of Quickstep register 1. Remember Modbus Address 1 is the upper 16 bits.

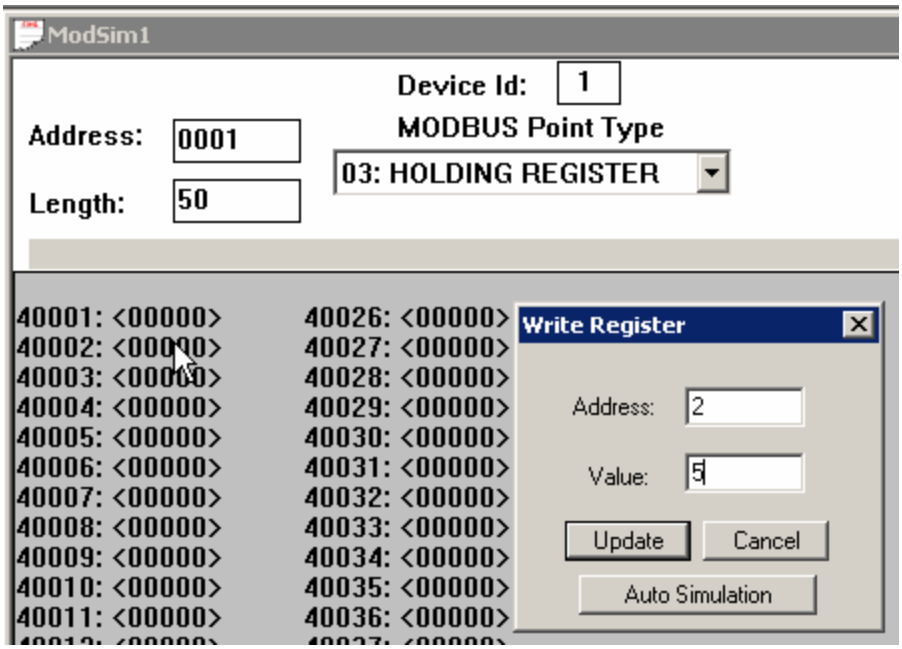


Figure 3.5: Single register write, value 5 to 40002

Changing a number of register all at once is known as a Write Multiple Register access. This can be done using the Extended Access option:

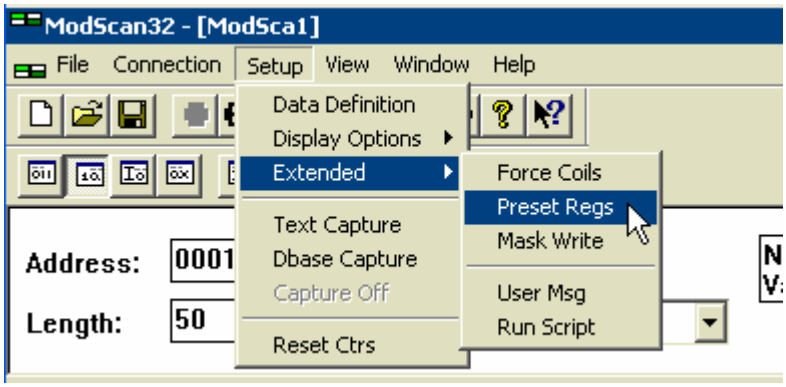


Figure 3.6: Write Multiple register (Preset Regs) selection

The Preset Multiple Registers will appear. Note that in TCP the 5100 controller ignores any slave or node identifiers since it is a single device and not acting as a gateway. Set the Modbus register you wish to start changes with and the number of registers to change, up to a maximum of that you are viewing:

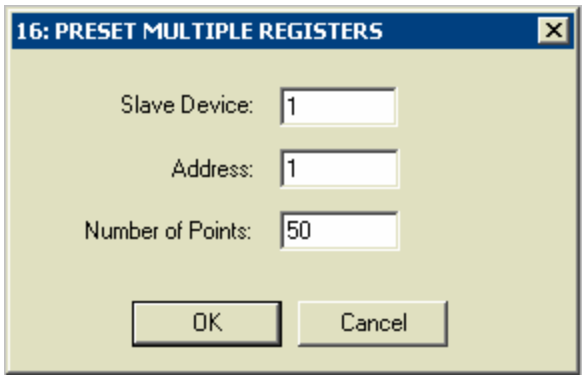


Figure 3.7: Preset Multiple register dialog

In this case we will change Addresses 1 to 10 to sequential numbers 1 to 10:

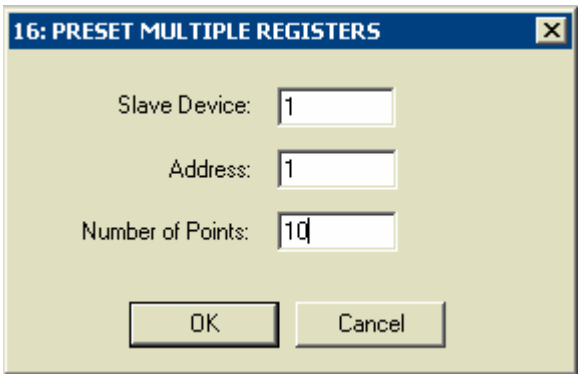


Figure 3.8: Select number of multiple writes to do

As shown below the current register values are displayed in the dialog box.

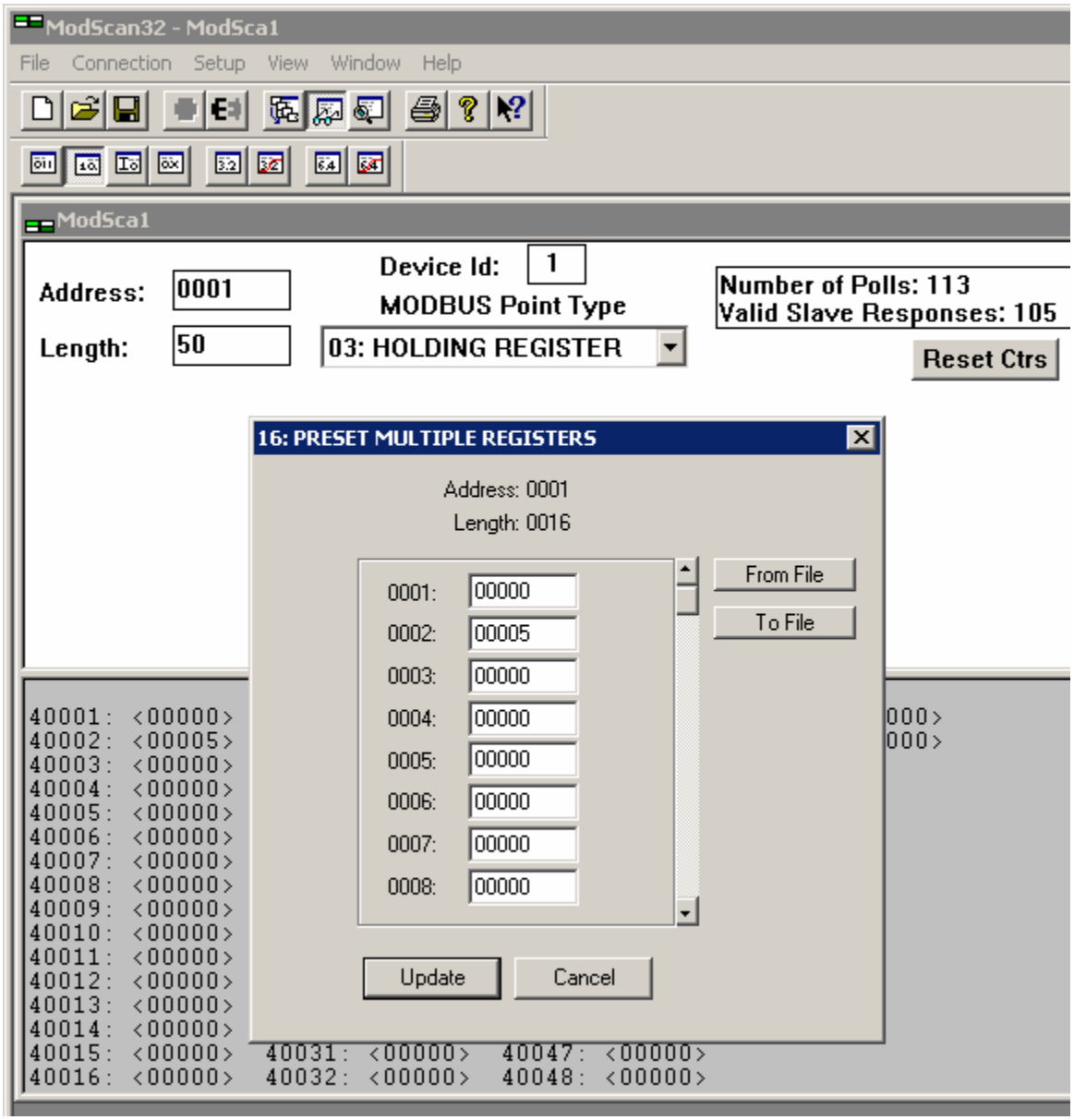


Figure 3.9: Preset Multiple register dialog viewing existing values

Note below that each register value has been changed, also we scrolled down so we could get to register 10. Click Update and note the changed register values from the previous display, 40002 is no longer 5 but now 2.

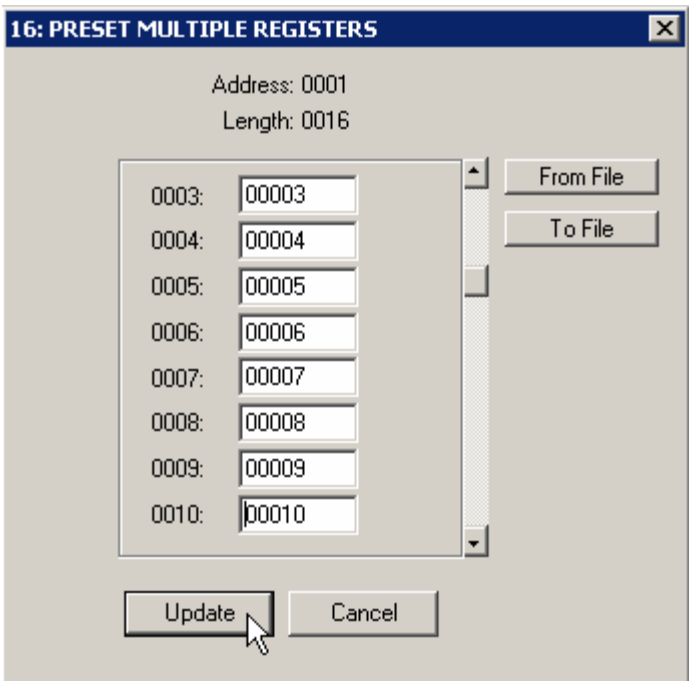


Figure 3.10: Preset Multiple new values entered

Upon clicking the Update key, the new values are written to the controller registers and new values read back using the Read Multiple Register command.

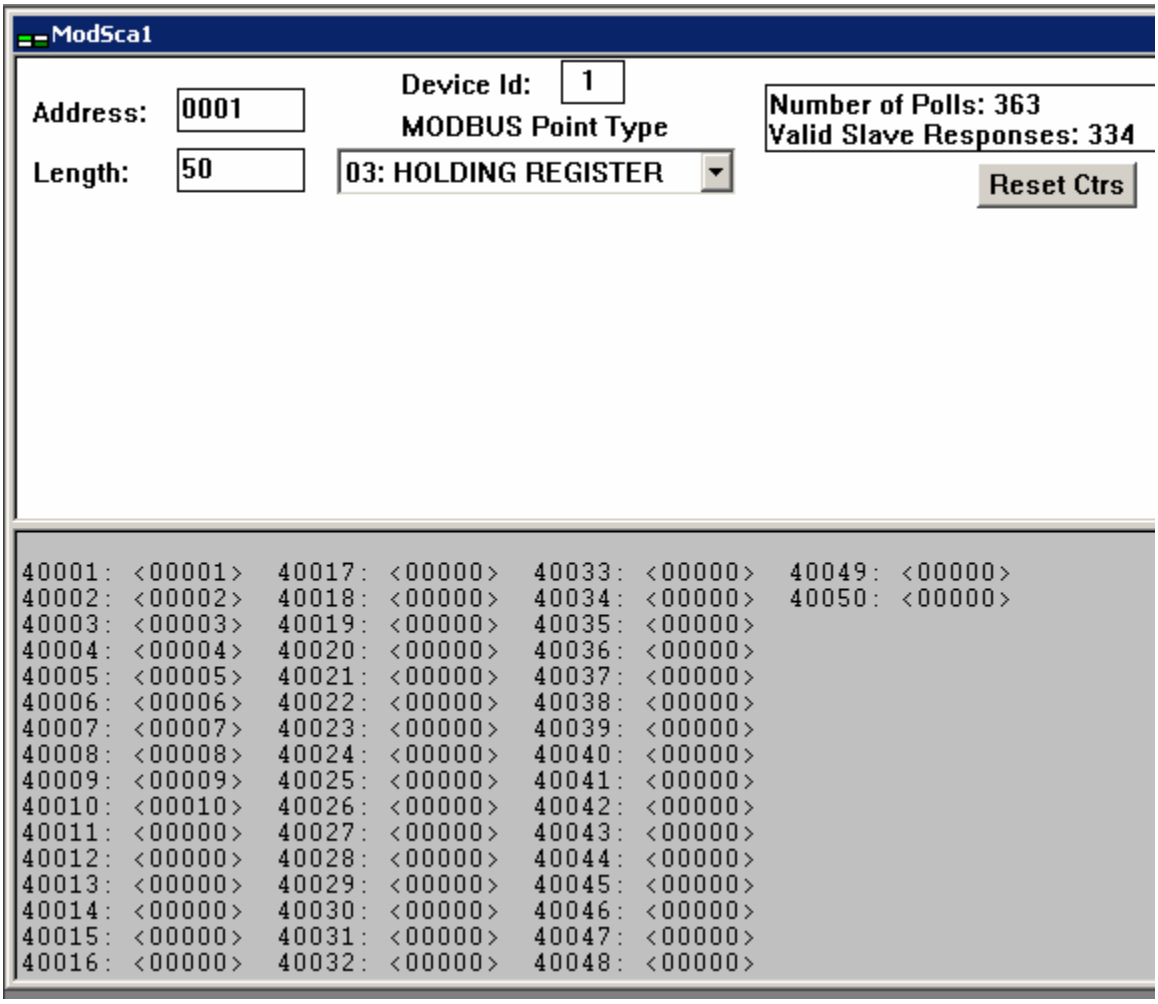


Figure 3.11: New values written and read back, Quickstep registers 1 to 5, Modbus 1 to 10

Should any errors occur a Modbus exception will occur. One such common error is attempting to read too many registers or illegal registers. Below is what is returned if > 120 Modbus registers are attempted:

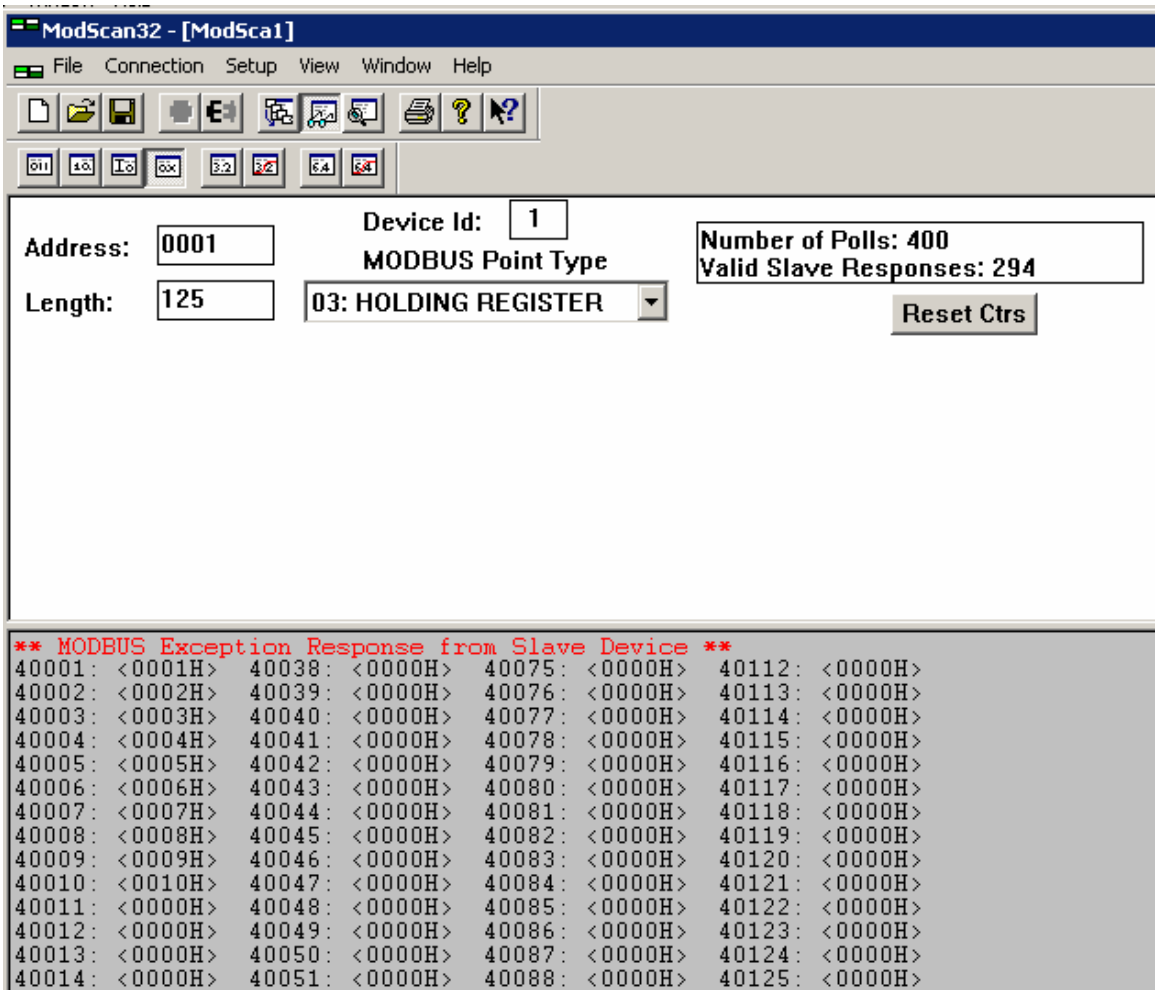


Figure 3.12: Modbus Exception Example > 120 registers

Editing the 125 appropriately will update the error. Below is an example of displaying registers in the 13002 block of the 5100 controller. 13002 is the system millisecond tic counter, real time clock/date values can also be seen incrementing in other register dynamically. Note that 26003 is the high 16 bits of 13002 and 26004 (13002 X 2) is the base lower 16 bits.

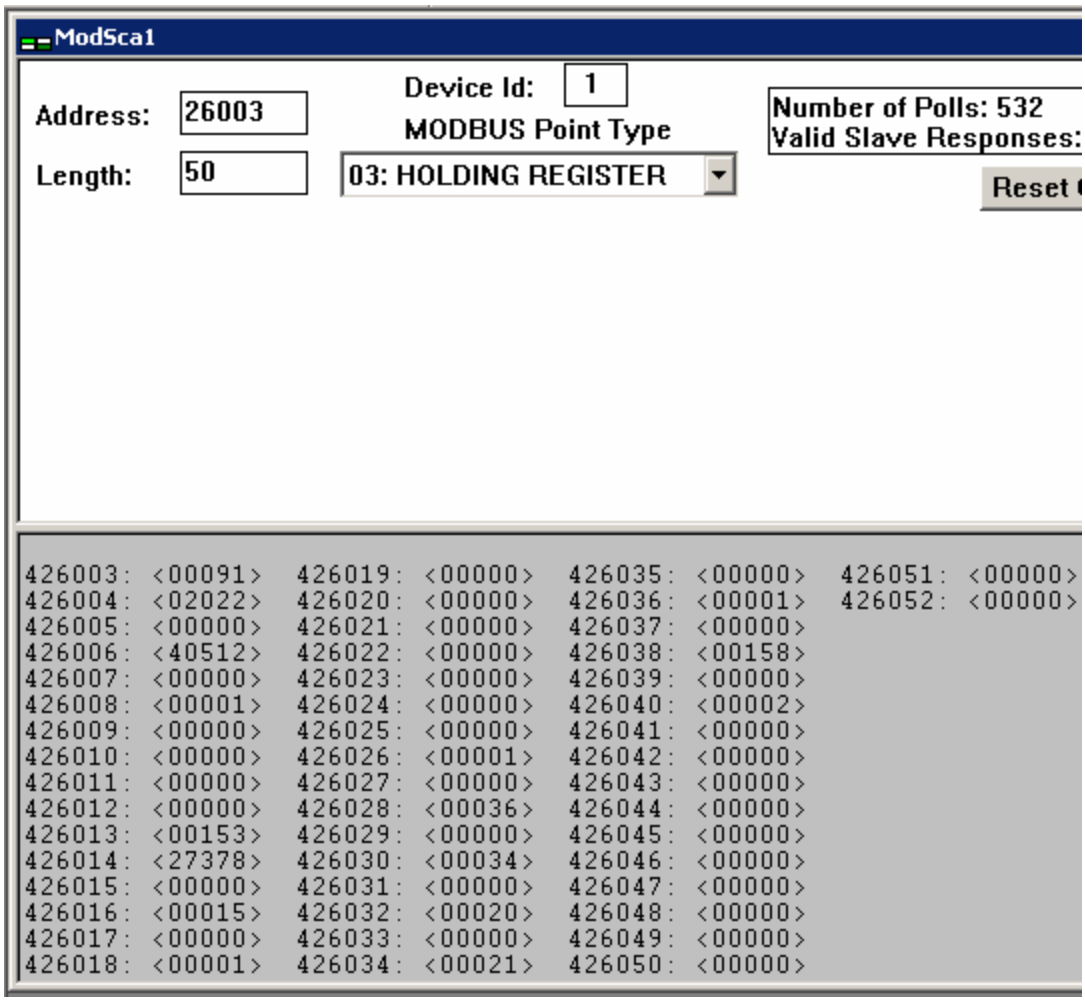


Figure 3.13: Display of controller system tic, dynamically updating, 426003/4

3.1.1 Modbus Slave Serial RTU/ASCII

The Modbus Slave Serial RTU and ASCII protocol functions exactly like that of Modbus TCP Slave with regards to how to access information and ModScan32 operation (see figure 3.14 for serial port setup versus TCP). There are some key differences since an RS232 connection is used versus a network connection.

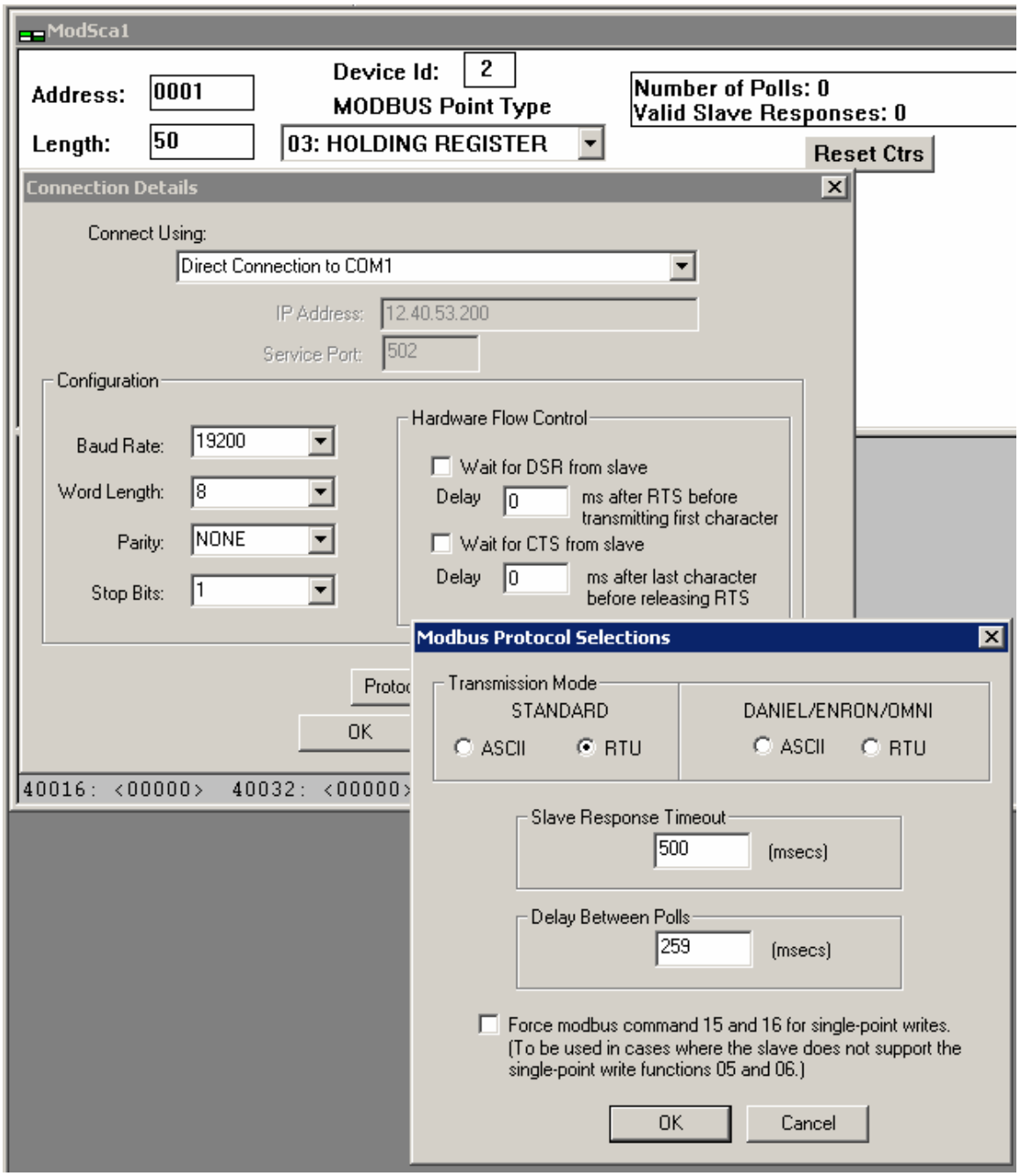


Figure 3.14: ModScan32 Master Scanning Program Serial Connection Setup, select RTU or ASCII Transmission Mode.

They are as follows:

- Only COM1 can be used for Modbus Serial RTU/ASCII protocol. COM2 uses an intelligent controller chip which does not currently support the protocol.
- The virtual TCP communication ports may also be used except for point to point operations with a single address present. In other words the communications

traffic of other Modbus nodes should not be present (can be on COM1). This is necessary because Modbus specifies a 3.5 character quiet time between packets and a maximum of 1.5 intercharacter delay during the continuous transmission of a packet data stream in RTU mode (1 second for ASCII mode). The virtual ports can not guarantee these timing constraints, although from a high level protocol viewpoint, the ports do comply.

- By default the Modbus protocol is disabled on the serial and virtual ports. To enable the port it must be the active port in the 12000 register and the proper Modbus protocol written to register 12320. Note that by default the slave port address is 2 and that any value written as the Modbus slave address will be that used on all serial ports, system wide. Note that writing a value of 0 to register 12320 will disable Modbus and return the port to normal CTC protocol operation.

Note: When Modbus is enabled on a serial port using CTCMON no further communications will be available on that port except with Modbus. In other words you will loose your CTCMON link if talking on the same port that is selected as active in register 12000.

3.2 Modbus Master TCP RTU & Serial RTU/ASCII

The Modbus Master protocol allows the controller to poll a Modbus TCP or Serial slave device, periodically requesting the registers for a particular device ID. As described in the Modus TCP Slave section, the protocol allows for interfaces to such things as coils, analog, registers, etc. The 5100 controller is capable of only polling and writing to the Holding Registers of a remote device. Write Single Register (function code 0x06), Write Multiple Registers (function code 0x10), and Read Holding Registers (0x03) commands are supported. Be advised, Modbus Master, as implemented on the 5100 controller, only polls a single device ID. The active device ID register must be changed in order to begin polling a different device. Those who require slow scanning of multiple devices may change the device ID within the Modbus Master Register Control Block (21000-21299, shared with the UDP Peer to Peer register block) by the use of a Quickstep program or it may be remotely modified. This will cause all subsequent polls to use that device ID and hence allow the reading/writing of multiple devices.

A maximum of 256 sequential Modbus registers (16 bit) can be polled, each optionally mapped to a corresponding controller register (32 bit, 21XX8, index 1007). You may also adjust the active start register by changing register 21XX4, described in 3.2.1, dynamically. The controller will read a maximum of 120 RTU (56 ASCII) registers per packet request. This means if the number of registers desired is 50 then 50 will be read with each poll. If the number of registers is greater than 120 then multiple requests are made. If 256 in RTU mode, for example, the first 120 are read, then the next 120, then the remaining 16, all transparent to the user/programmer. When using the remapping register option, all registers will appear sequential within the 23000-24999 register blocks. Simply read and write as desired.

3.2.1 Registers 21000-21299

The 5100 controller can run numerous Modbus TCP Master connections and a single RTU/ASCII Serial connection at the same time, to differing devices, limited only by the performance desired. Each is configured using the Modbus Master Register Control Block (MMRCB), this same block serves multiple purposes and is shared with the UDP Peer to Peer Protocol register block detailed in section 5.1.1.

21XX0 - First Octet IP Address Register (Most Significant) - R/W

This is the first octet of the IP address (XXX.000.000.000) of the Modbus Slave to connect to. If a serial port is used, set to anything other than 0.

21XX1 - Second Octet IP Address Register - R/W

This is the second octet of the IP address (000.XXX.000.000) of the Modbus Slave to connect to. If a serial port is used, set to anything other than 0.

21XX2 - Third Octet IP Address Register - R/W

This is the third octet of the IP address (000.000.XXX.000) of the Modbus Slave to connect to. If a serial port is used, set to anything other than 0.

21XX3 - Fourth Octet IP Address Register (Least Significant) - R/W

This is the fourth octet of the IP address (000.000.000.XXX) of the Modbus Slave to connect to.

Once a connection is attempted, you cannot change the IP octet register settings.

21XX4 - Start Register - R/W

This register stores the starting register address which is to be read from the remote Modbus Slave device. It may be modified at any time to select a different register block. Typically address 1 will represent Holding Register 40001 on the device.

21XX5 - Sequential Number Register - R/W

This register stores the number of sequential registers (starting with Register 21XX4) you want to read during a polling session. The value 1 represents a single register and the maximum number of registers allowed is 256. Configure this register before setting up any other registers. Do not change this value during a transaction or all data will be lost and new values will have to be entered. If you modify this register, it lets you reset the connection. All register reads from remote devices will be the same block size.

21XX6 - Poll Timer Register - R/W

Set this register to 0 for a single read request. Specify a value (in units of ms/count) if this register is going to receive periodic updates from the server controller (the controller sending information to the register). The minimum value allowed is 10 ms.

For example; the value 500 would refresh the data registers with new remote data every ½ second. You can access this register at any time once you have initialized the Sequential Number Register (Register 21XX5).

Data registers are mentioned in numerous places throughout the listings below. These registers are represented by Register 21XX9, which is a phantom register. For more information, refer to the 21XX9 listing in this section.

21XX7 - Status Flag Register - Read-Only

This register reflects the current status of the data registers. Its value is based on any requested operations. Typically, you initiate an operation and then wait for a status of '1'. Possible values are:

STATUS	*DESCRIPTION*
0	Offline; no connection is present.
1	Last request is successful and completed. Data is available in the data registers if requested.
-1	Requested operation has failed, typically a Modbus Exception error.
-2	Busy; connecting to the desired host.
-3	Busy; reading data.
-4	Busy; writing data.
-5	Timed out, retrying.
-10	Aborted operation; out of local memory or resources.

21XX8 - Index Offset Register - R/W

This register lets you access each of the requested sequential data registers. It works in conjunction with Register 21XX9 and acts as its pointer. You can store the number of a general or special purpose register in 21XX8 and 21XX9 can then access the resource contained in the pointer. By default 0, points to the very first data element read from the remote device. This would be equivalent to what you set the Start Register to begin with (21XX4). Incrementing this register allows you to access other data elements, like an array. Register 21XX9 can then be read or written accordingly. If access using an index register for accessing general data is not desired, the data may also be mapped to sequential registers of your choosing, refer to index register 1007 discussed below. This is the preferred method. Although do not modify the index register while a write is occurring otherwise strange results may occur.

The index register also has a few special features when you set it to 1000 or above. Modifications are made by writing to the data register and setting the index register appropriately as described below (only registers used by Modbus appear):

1003 - Protocol Index Register - This register tells the data register what protocol to use for setting the peer block registers. You must set this register before setting the Start Register (21XX4). Default mode is 0 for UDP Peer-to-Peer protocol. 2 is used for Modbus TCP Master mode, 3 is used for Modbus Master RTU Serial, and 4 for Modbus Master ASCII Serial.

1004 – TCP/Serial Client Port Index Register - This register points the data register to the destination TCP Port address for your connection, or serial port. You must set this register after setting the Protocol Index Register, otherwise default values will overwrite any new values. When Protocol Index Register is set to '2' (TCP) the default client port is 502, when '3' (Serial), then the client port is set to 1, referencing COM1. For Modbus TCP Master mode 502 is the industry standard port to connect to. Any client port less than 10 is assumed to be a serial port.

1005 - Modbus Master Unit ID Index Register - This register points the data register to the Unit/Device ID field value used in the Modbus Master request packet. The default ID is '1' but you can set it to any desired value. This ID affects all subsequent transmissions and allows multiplexed devices to be addressed in a Modbus environment.

1006 - Modbus Master Exception Index Register - This register allows you to interrogate the last Modbus Exception error code received from the data register (21XX9). Referencing this register helps to interpret failure types. Typically you would reference this register if a '-1' appears as the current status in register 21XX7.

1007 – Register Remapping Start Index Register – This option allows remote registers to be mapped into the 23000 to 24999 consecutive memory space. Previously an index register at 21XX8 needed to be set then data read from 21XX9. This can result in slow operation if a lot of data needs to be transferred. Setting 21XX8 to 1007 and then writing the register value from 23000 to 24999 will allow all data to be remapped to that register block area, consecutively, based upon the block size (21XX5). A write to the remapped area will result in a remote write. By default re-mapping is not active.

1008 - Modbus Master MAX Retries Register – (R/W) This register allows you to change the maximum number of retry attempts on a Unit ID before giving up. Default is 2.

1009 - Modbus Master Retry Counter Register – (R/W) This register allows you to observe and change the current number of message retries to the current Unit ID.

1010 - Modbus Master Timeout Register – (R/W) This register allows you to change the default Unit ID timeout from 250 milliseconds to that desired, in milliseconds. Note that TCP needs a value > 200 milliseconds when talking to many applications, especially if PC based. If this is not used there will be many timeouts and retries. For example response times of up to 200 milliseconds have been observed the ModSim32 PC programs. The controller can handle smaller values without a problem, it is the PC side that is slow to respond. For Modbus Master RTU Serial the value in 21XX5 is added to the base timeout of 250 milliseconds. 1000 milliseconds is the base timeout for Modbus Master ASCII Serial.

1011 - Modbus Master Block Size Register – (R/W) This register sets the number of Holding Registers to be accessed. Must be the same or smaller than the Sequential Number Register, defaults to the same. Used to access Unit ID's with varying block sizes when manually changing the Unit ID under program control.

21XX9 - Data Registers - R/W

This phantom register contains peer data that is read or written in a peer transaction. It is a "window" into a register array in the controller. The array size is set by Register 21XX5 and the offset is specified by Register 21XX8. Data integrity is indicated in Register 21XX7. If remapping of registers is not used then set 21XX8 to the array element desired, with 0 being the first.

3.2.2 Example Modbus TCP & RTU Serial Master Initialization

An example of Quickstep initialization code is shown below to setup a connection to the following remote device:

Modbus TCP Master Sample Program:

IP Address – 12.40.53.168
Device ID – 1
Number of sequential registers to read – 160
Scan time – 100 ms. (set last to initiate)
Starting Register - 1
Re-map registers to consecutive block beginning at registers 23000.
This is the first setup so use 21000, next would be 21010… 21020, etc…

```
[1] Initialize_ModbusMaster
   ;;; This program is used to initialize the TCP port
   ;;; for Modbus TCP Master operation.  A single
```

```
   ;;; device is polled using device ID 1 and 160 registers
   ;;; are read and mapped into the 23000 block.  Therefore
   ;;; registers 23000 - 23159 are used, with 23000 referencing
   ;;; Modbus Register #1.  Make sure your Modbus device has
   ;;; at least 160 consecutive registers starting at '1'
   ;;; otherwise Modbus Exceptions will occur.
   ;;; Begin by doing the following:
   ;;; 21005 = Maximum number of registers to read (160)
   ;;; 21000 - 21003 = Set this to be the IP address to
   ;;;                 connect to.  In this example we
   ;;;                 will use 12.40.53.168
   ;;; 21004 = Modbus start register (1)
   ;;; 21008 = 1003  = Set index to point to protocol register
   ;;; 21009 = 2     = Set protocol to Modbus TCP Master
   ;;; 21008 = 1004  = Set TCP port to connect to, default is 502
   ;;; 21009 = 502   = For demo set port to 502 even though default
   ;;; 21008 = 1007  = Set index to point to where to view data
   ;;; 21009 = 23000 = Start remapped area at 23000 for 160 regs.
   ;;; 21008 = 0     = Always set the index back to 0 before begin
   ;;; 21006 = 100   = Set scan poll time to 100 ms./block read,
   ;;;                 min is 50ms.  This also initiates polling.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 160 to reg_21005
   store 12 to reg_21000
   store 40 to reg_21001
   store 53 to reg_21002
   store 168 to reg_21003
   store 1 to reg_21004
   store 1003 to reg_21008
   store 2 to reg_21009
   store 1004 to reg_21008
   store 502 to reg_21009
   store 1007 to reg_21008
   store 23000 to reg_21009
   store 0 to reg_21008
   store 100 to reg_21006
   goto Next

[2] Wait_For_Online
   ;;; Once Modbus Master starts to poll we must wait until
   ;;; it is online before proceeding.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   if reg_21007=1 goto Modbus_Online
   delay 500 ms goto Wait_For_Online

[3] Modbus_Online
   ;;; It is OK to read and process data now since Modbus
   ;;; is online to the device.  If you wish to monitor another
   ;;; device other than Unit ID 1, then change the index register
   ;;; 21008 to 1005 and write the desired Unit ID to register
```

```
;;; 21009, then set 21008 back to 0 and monitor 21007 for
;;; a 1 for online state, once again.  Results will appear
;;; in the 23000 block.
--------------------------------------------------------------------------
<NO CHANGE IN DIGITAL OUTPUTS>
--------------------------------------------------------------------------
delay 1000 ms goto Modbus_Online
```

When Reg_21007 is equal to a '1' then the connection is active and you may interact with the remote device. If a 3 had been written to 1003 then Modbus Master RTU Serial on COM1 would be used.

Modbus RTU Serial Master Sample Program:

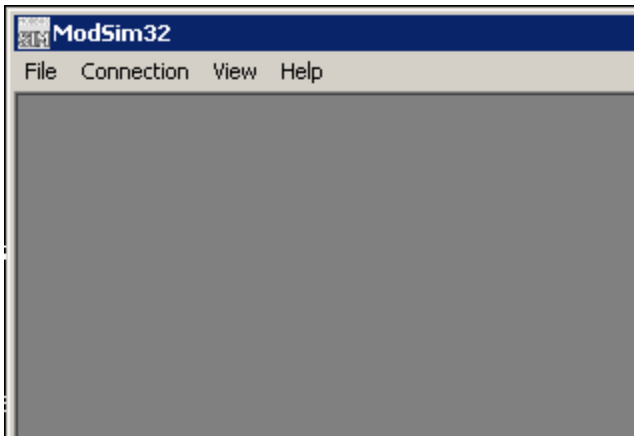
IP Address – 12.40.53.168 (can be set to any value other than –1)
Device ID – 1
Number of sequential registers to read – 160
Scan time – 100 ms. (set last to initiate)
Starting Register – 1
Serial Port – COM1
Remap registers to consecutive block beginning at registers 23000.
This is the first setup so use 21000, next would be 21010… 21020, etc…

```
[1] Initialize_ModbusMaster
   ;;; This program is used to initialize the COM1 port
   ;;; for Modbus RTU Serial Master operation.  A single
   ;;; device is polled using device ID 1 and 160 registers
   ;;; are read and mapped into the 23000 block.  Therefore
   ;;; registers 23000 - 23159 are used, with 23000 referencing
   ;;; Modbus Register #1.  Make sure your Modbus device has
   ;;; at least 160 consecutive registers starting at '1'
   ;;; otherwise Modbus Exceptions will occur.
   ;;; Begin by doing the following:
   ;;; 21005 = Maximum number of registers to read (160)
   ;;; 21000 - 21003 = Any value, required to unlock register
   ;;;                 group, on Modbus TCP this is the IP
   ;;;                 address for a connection.
   ;;; 21004 = Modbus start register (1)
   ;;; 21008 = 1003  = Set index to point to protocol register
   ;;; 21009 = 3  = Set protocol to Modbus RTU Serial (4 for ASCII Serial)
   ;;; 21008 = 1004  = Set serial port to use, default is 1
   ;;; 21009 = 1     = For demo set port to 1 even though default
   ;;; 21008 = 1007  = Set index to point to where to view data
   ;;; 21009 = 23000 = Start remapped area at 23000 for 160 regs.
   ;;; 21008 = 0     = Always set the index back to 0 before begin
   ;;; 21006 = 100   = Set scan poll time to 100 ms./block read,
   ;;;                 min is 10ms.  This also initiates polling.
   --------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   --------------------------------------------------------------------------
   store 160 to reg_21005
   store 10 to reg_21000
```

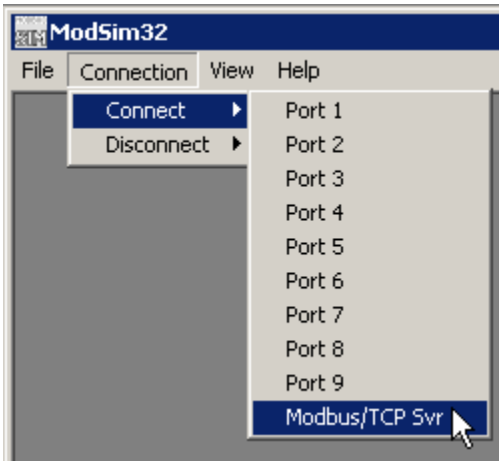
```
   store 10 to reg_21001
   store 10 to reg_21002
   store 10 to reg_21003
   store 1 to reg_21004
   store 1003 to reg_21008
   store 3 to reg_21009
   store 1004 to reg_21008
   store 1 to reg_21009
   store 1007 to reg_21008
   store 23000 to reg_21009
   store 0 to reg_21008
   store 100 to reg_21006
   goto Next

[2] Wait_For_Online
   ;;; Once Modbus Master starts to poll we must wait until
   ;;; it is online before proceeding.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   if reg_21007=1 goto Modbus_Online
   delay 500 ms goto Wait_For_Online

[3] Modbus_Online
   ;;; It is OK to read and process data now since Modbus
   ;;; is online to the device.  If you wish to monitor another
   ;;; device other than Unit ID 1, then change the index register
   ;;; 21008 to 1005 and write the desired Unit ID to register
   ;;; 21009, then set 21008 back to 0 and monitor 21007 for
   ;;; a 1 for online state, once again.  Results will appear
   ;;; in the 23000 block.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   delay 1000 ms goto Modbus_Online
```

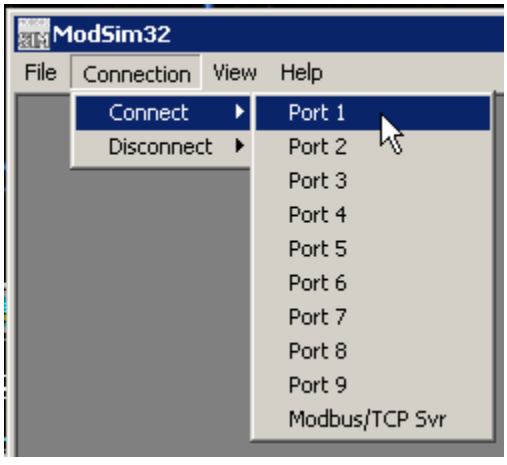
3.2.3 Testing With Win-Tech's ModSim32

As a demonstration of the functionality of the controller Modbus Master interface this section details the interface of Win-Tech's ModSim32 software and how it applies with regard to our product. It is assumed that the controller Modbus TCP Master or Serial is setup to point to the PC and attempting a connection. As mentioned before, we only support the Holding Register interface. Upon invoking of ModSim32 the below screen will appear.

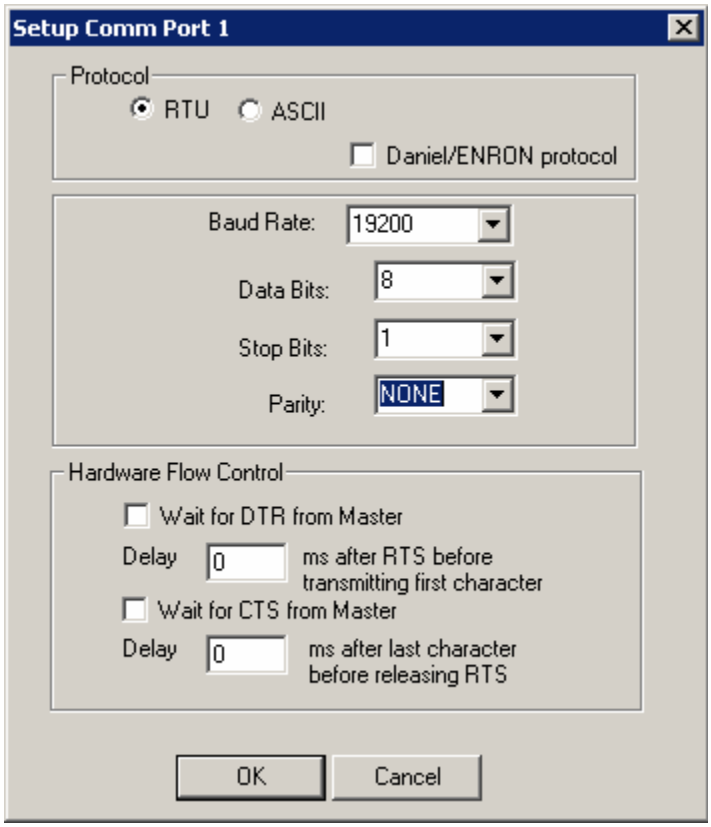


In order to activate the Modbus slave, you must select the Connection menu item and the method of the connection, Modbus/TCP Svr for network or the appropriate 'Port #' for a serial port.

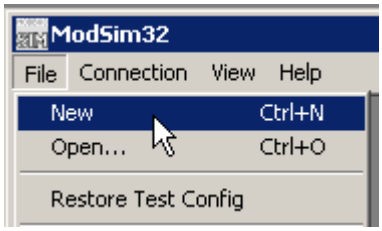


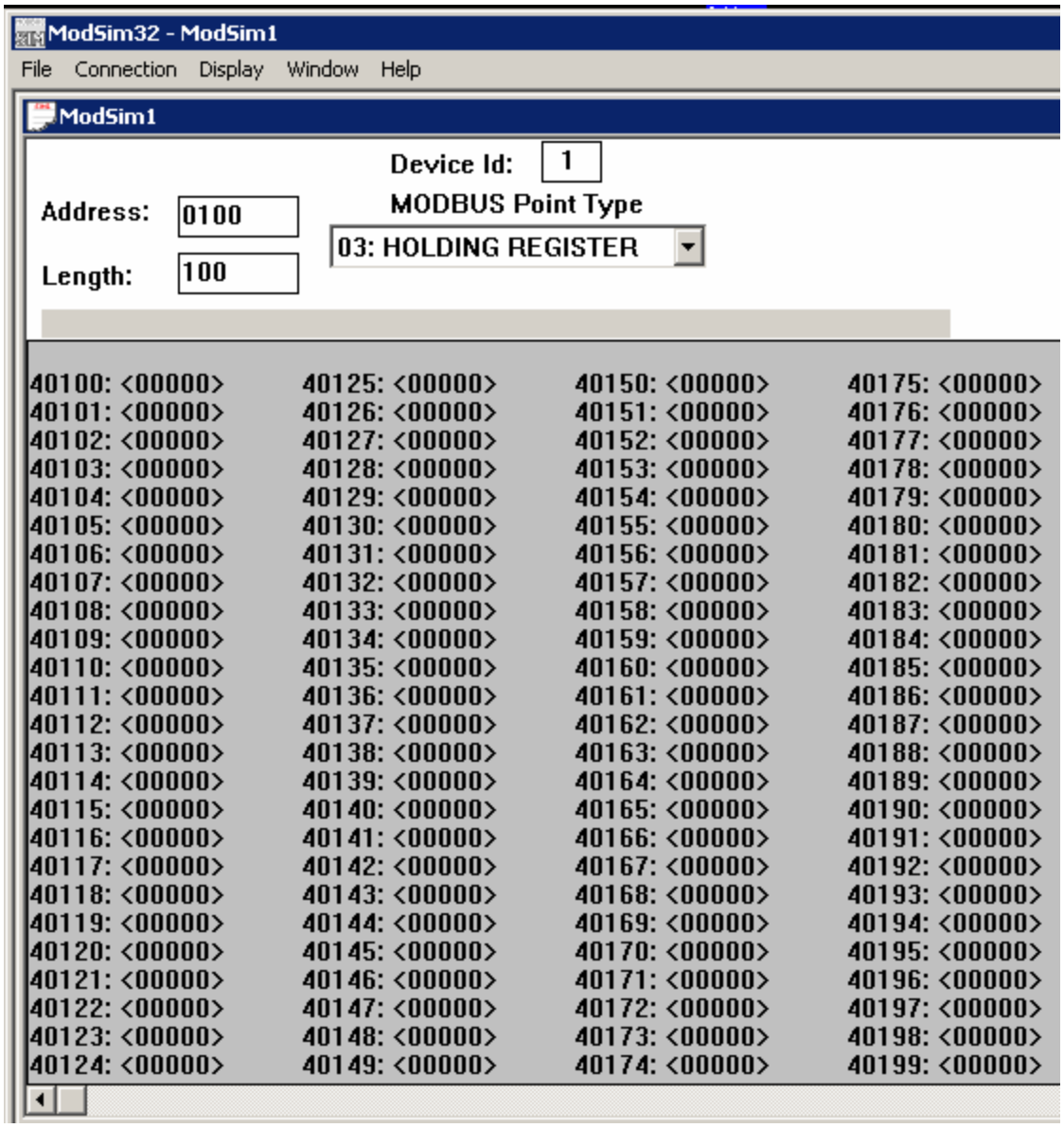


If Serial select RTU or ASCII and set the baud rate, stop bits, and parity appropriately. Default for the 5100 is 19.2K baud, 1 stop bit, 8 data bits, no parity, this is not the default for ModSim and must be changed as shown below:

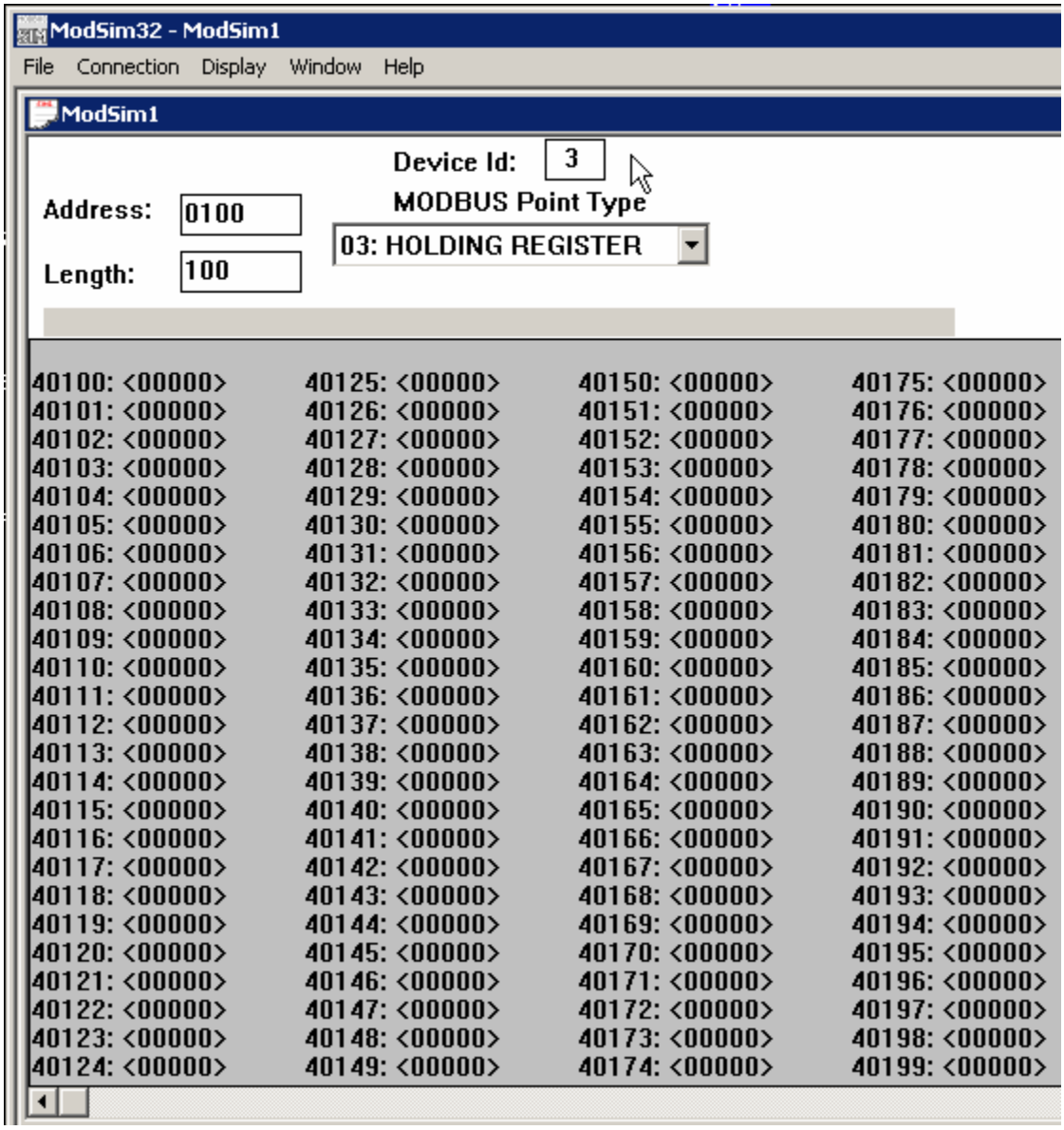


Next devices must be created to listen to the requests. This is done using menu selection: File-> New

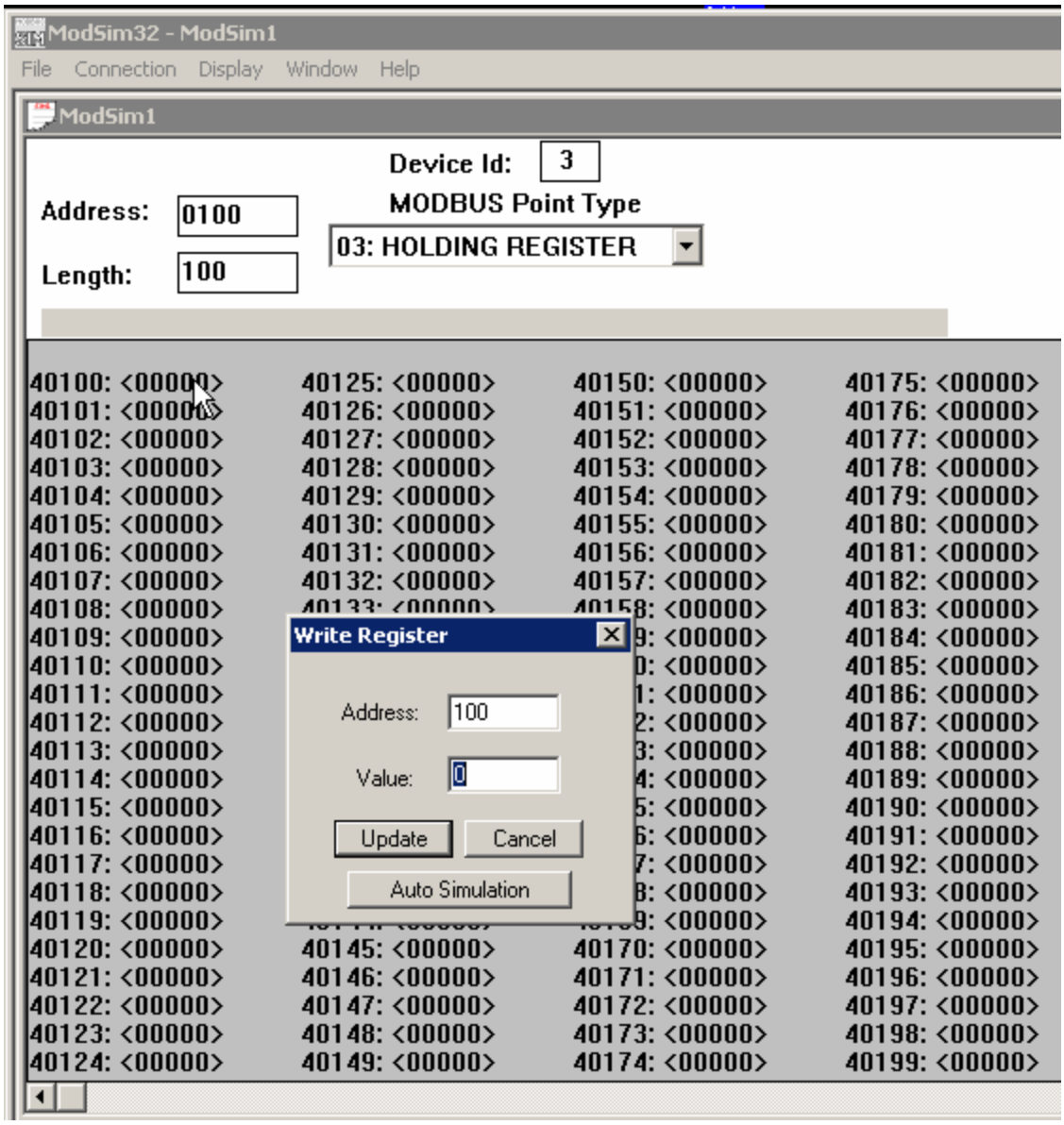




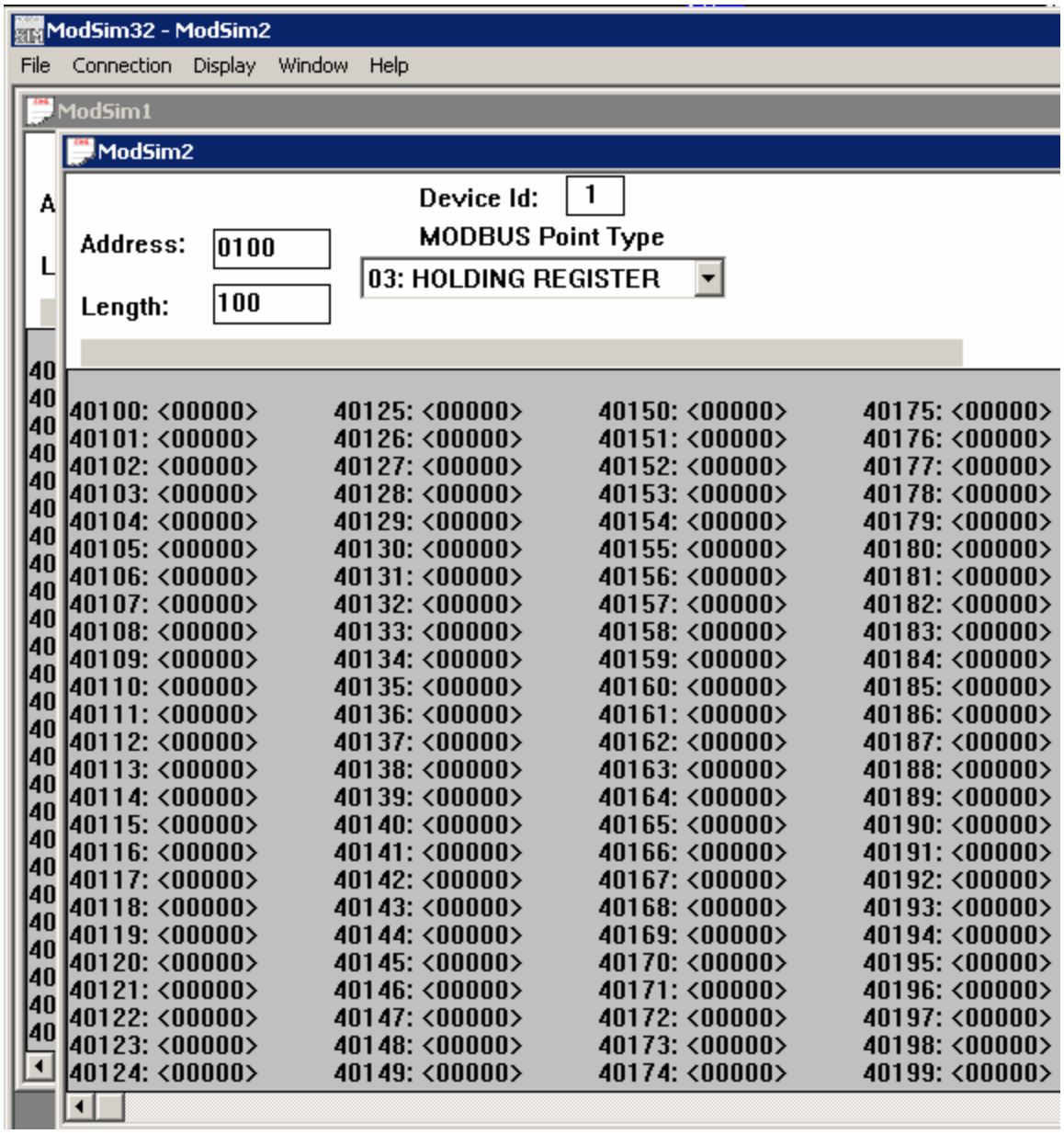
In order to access this device the controller must have its Device ID set to 1 (the default) and the Starting Address set to 100. If not set correctly, an exception status will be returned upon connection and 21XX7 register will contain a -1. If we wanted to set the Device ID to a 3, as in our example, modify as below:



Note that the 'Address' field is set to 100, but the display screen starts at 40100. This is Modbus nomenclature. To modify a device Holding Register contents simply double click on the address and enter the new value in the dialog that appears:



Above shows the modification of address 100. Additional devices can also be created by once again selecting File->New. This allows for the testing of multiple Modbus Slave devices at the same time:



Above shows multiple devices enabled. If there are further questions about the use of ModSim32 simply select the 'Help' menu item and a manual will appear.

4.0 CTNet Binary Protocol (Server)

The CTNet binary protocol is a high-speed, non-routable protocol that has checksum and error reporting capabilities. It is used in cases where data integrity, response time, and processing time are the major criteria. Data transmission is fast for the following reasons:

- Both the commands and data are represented in binary form instead of ASCII.
- The information density is higher and fewer characters are transmitted during large data transfers.
- The controller can use the data "as is" and does not have to perform binary to ASCII conversion.

This results in shorter execution times. Note that the binary protocol is non-routable. Non-routable protocols do not contain a networking layer (IP stack), so they cannot cross a router and are limited to local subnets or intranets. However, lack of an IP stack reduces overhead by at least 20 bytes/packet. A smaller packet size increases the transmission rate, which is ideal for industrial controllers. Routable protocols such as TCP/IP result in a larger packet and more processor overhead to process.

CTNet uses a node number is used in place of an IP address. This node number is defined by writing to Register 20000. You can also determine the node number by reading the value in Register 20000. Set this value within the 5100.ini file by defining the CTNET_DEVICENODE parameter.

For more information on the CTNet binary protocol, refer to the *CTC Serial Data Communications Guide*.

5.0 UDP Peer to Peer Protocol Overview

5.1 Peer-to-Peer Protocol Registers

The controller can only perform peer-to-peer operations with other 5100 modules. It is not compatible with the 2217 communications module but is with the 2717. The 5100's peer-to-peer registers let it communicate directly with other 5100 modules without requiring a dedicated server. It can also gather register information locally for different network protocols.

Registers 21000-21299 are read/write registers that are reserved for peer-to-peer networks. Each block of 10 sequential registers is assigned to a designated peer node and defines the peer environment for that connection. You can retrieve data from and automatically update up to 100 sequential registers with a single request. This register

block is used for many functions by different network protocols (Peer-to-Peer, Modbus TCP Master, etc.) that all interface with the registers in the same manner.

5.1.1 Registers 21000-21299

21XX0 - First Octet IP Address Register (Most Significant) - R/W

This is the first octet of the IP address (XXX.000.000.000) that is used to make peer requests.

21XX1 - Second Octet IP Address Register - R/W

This is the second octet of the IP address (000.XXX.000.000) that is used to make peer requests.

21XX2 - Third Octet IP Address Register - R/W

This is the third octet of the IP address (000.000.XXX.000) that is used to make peer requests.

21XX3 - Fourth Octet IP Address Register (Least Significant) - R/W

This is the fourth octet of the IP address (000.000.000.XXX) that is used to make peer requests. Once a peer connection is attempted, you cannot change the IP octet register settings.

21XX4 - Start Register - R/W

This register stores the starting register address in the controller for peer-to-peer communications. You can change this register number after a peer connection is attempted, but the number of sequential registers must stay the same (see *Register 21XX5* for more information).

21XX5 - Sequential Number Register - R/W

This register stores the number of sequential registers (starting with Register 21XX4) you want to read during a peer-to-peer session. The value 1 represents a single register and the maximum number of registers allowed is 100. Configure this register before setting up any other registers. Do not change this value during a peer-to-peer transaction or all data will be lost and new values will have to be entered. If you modify this register, it lets you reset the peer connection.

21XX6 - Poll Timer Register - R/W

Set this register to 0 for a single read request. Specify a value (in units of ms/count) if this register is going to receive periodic updates from the server controller (the controller sending information to the register). The minimum value allowed is 50 ms. For example, the value 500 would refresh the data registers with new peer data every ½ second.

You can write to this register at any time. Writing a 0 to this register while actively conducting a peer-to-peer session cancels the periodic update and causes a new single read transaction to occur. A time-out (Status Flag Register 21XX7 = 0) occurs if the server has not refreshed peer data in a time equal to 2-½ multiplied by the poll timer value. You can access this register at any time once you have initialized the Sequential Number Register (Register 21XX5).

Data registers are mentioned in numerous places throughout the listings below. These registers are represented by Register 21XX9, which is a phantom register. For more information, refer to the 21XX9 listing in this section.

21XX7 - Status Flag Register - Read-Only

This register reflects the current status of the data registers. Its value is based on any requested operations. Typically, you initiate an operation and then wait for a status of 1. Possible values are:

STATUS	*DESCRIPTION*
0	Offline; no connection is present.
1	Last request is successful and completed. Data is available in the data registers if requested.
-1	Requested operation has failed.
-2	Busy; connecting to the desired host.
-3	Busy; reading data.
-4	Busy; writing data.
-10	Aborted operation; out of local memory or resources.

21XX8 - Index Offset Register - R/W

This register lets you access each of the requested sequential data registers. It works in conjunction with Register 21XX9 and acts as its pointer. You can store the number of a general or special purpose register in 21XX8 and 21XX9 can then access the resource contained in the pointer. The first register (with an index of 0) is the Start Register (Register 21XX4). 1 is the next register, and so forth. Once Register 21XX5 (the Sequential Number Register) is initialized, you can change this register's value at any time. For more information on how pointer registers function, refer to the *Register Reference Guide*.

The index register also has a few special features when you set it to 1000 or above. Modifications are made by writing to the data register and setting the index register appropriately as described below:

1000 - Peer Request Time-Out Register - The timer starts when a peer node request is initiated and stops (times out) if no response is received within the time specified by this register. Retries only occur if automatic updates are active (Register 21XX6 is set to a value other than 0). Defaults are 500 ms for single register reads and time-out value*2.5 for automatically updated register read transactions.

1001 - Peer Request Failed Index Register - This register indicates when a peer transaction fails and an error occurs. The Status Flag Register (21XX7) is set to a value other than 1. Any data that was read or written when the error occurred has an offset value that is stored in 1001. If you read the data register, it returns the offset failure value. Data written before this offset value is valid. For example, if your process continuously updates 50 registers and the register returns a value of 25, it means the process failed while trying to write the 25th element of data. All data written before this element was written correctly.

1002 - Peer Request Retry Counter Index Register - This debugging register points the data register to the retry counter. Quickstep can set this register to any value. The register is incremented by 1 when a time-out occurs because of waiting for data from a peer node.

1003 - Protocol Index Register - This register tells the data register what protocol to use for setting the peer block registers. You must set this register before setting the Start Register (21XX4). Default mode is 0 for UDP Peer-to-Peer protocol. 2 is used for Modbus TCP Master mode, 3 for Modbus Serial Master.

1004 - Peer Request TCP Client Port Index Register - This register points the data register to the destination TCP Port address for your connection. You must set this register before setting the Start Register (21XX4). 1004 is currently used for Modbus TCP Master mode with a default port number of 502 (the industry standard).

1005 - Modbus Master Unit ID Index Register - This register points the data register to the Unit ID field value used in the Modbus Master request packet. The default ID is 00 but you can set it to any desired value. This ID affects all subsequent transmissions and allows multiplexed nodes to be addressed in a Modbus environment.

1006 - Modbus Master Exception Index Register - This register tells the data register where the last Modbus Exception error code is stored from a previously received message. Referencing this register helps to interpret failure types.

1007 – Register Remapping Start Index Register – This option allows remote registers to be mapped into the 23000 to 24999 consecutive memory space. Previously an index register at 21XX8 needed to be set then data read from 21XX9. This can result in slow operation if a lot of data needs to be transferred. Setting 21XX8 to 1007 and then writing the register value from 23000 to 24999 will allow all data to be remapped to that register block area, consecutively, based upon the block size (21XX5). A write to the remapped area will result in a remote write. By default re-mapping is not active.

1998 - Write Enable Index Register - This register is used to control the updating of writes to the peer. When enabled (default, 1), any write to a data register (21XX9 or remapped area) will cause a single write to the remote host. Setting this register to a 0 inhibits write operations. This allows the programmer to update the register block, as required, then set the Write Enable Register back to a '1' to update the entire block, on the remote host, by sending a single packet. The Write Enable Register will not return a '1' on a read operation of '1998' until an acknowledge from the remote host has been received, verifying the write took occurred. The transition of the Write Enable Register from 0 to 1 causes the block write, writing a 1 when a 1 already exists has no effect.

21XX9 - Data Registers/Peer Request Time-Out Register - R/W

This phantom register contains peer data that is read or written in a peer transaction. It is a "window" into a register array in the controller. The array size is set by Register 21XX5 and the offset is specified by Register 21XX8. Data integrity is indicated in Register 21XX7. For more information on how phantom registers function, refer to the *5100 Register Reference Guide*.

Initiating a Peer to Peer Session

In general, the initializing of the peer-to-peer mechanism works as follows:

1. Write the desired number of registers to 21XX5 register.
2. Write the slaves IP address to 21XX0 – 21XX3 register
3. Write the register to begin reading from of the slave device to 21XX4
4. Write a 1007 to register 21XX8 to select the re-mapping area.
5. Write where in the 23000-24999 register area you want it to appear to register 21XX9.
6. Write a 0 to the index register 21XX8 to default it back to viewing the first data item.
7. Write the scan time, typically 100ms to register 21XX6 to initiate the connection and begin peer to peer.
8. Monitor status register 21XX7 for a '1' prior to reading/writing to either 21XX9 data area or the re-mapped area in the 23000-24999 block.

6.0 SNTP Simple Network Time Protocol (RFC-2030)

The 5100 controller supports the Simple Network Time Protocol (SNTP) as a client connecting to a server. This protocol provides a means to synchronize a computer system clock to that of the world clock, via the internet. Government agencies provide this service for computers to query the current atomic clock time and adjust their clocks appropriately. For more detailed information reference www.time.gov and www.boulder.nist.gov/timefreq/service/its.htm.

The time returned is based on Coordinated Universal Time (UTC), which is Greenwich Mean Time (GMT). As such, there is no adjustment for daylight savings time or time zones, that must be done locally. To avoid daylight savings time problems it is recommended that you base the controller time on GMT but provisions have been provided to automatically set the clock based on the time zone you are in, using an offset from GMT.

Use of SNTP is not a requirement but typically real time clocks can be expected to drift up to 30 seconds per week. The 5100 controller will drift up to 12 seconds per week, depending on the tolerance of crystals, components, etc. Synchronization allows the real time clock to be automatically set with regards to date, year, day of week, and time.

6.1 SNTP Implementation

By default the controller will use the IP address of 192.43.244.18, port 123. Updates will be performed once/day and the clock is set to GMT. This may be changed by modifying the following registers:

20025 - First Octet IP Address Register (Most Significant) for SNTP Server - R/W

This is the first octet of the IP address (XXX.000.000.000) that is used to connect to the SNTP server. Default is 192.

20026 - Second Octet IP Address Register) for SNTP Server - R/W

This is the second octet of the IP address (000.XXX.000.000) that is used to connect to the SNTP server. Default is 43.

20027 - Third Octet IP Address Register) for SNTP Server - R/W

This is the third octet of the IP address (000.000.XXX.000) that is used to connect to the SNTP server. Default is 244.

20028 - Fourth Octet IP Address Register for SNTP Server - R/W

This is the fourth octet of the IP address (000.000.000.XXX) that is used to connect to the SNTP server. Default is 18.

The unit must be reset for a new IP address to take effect.

20041 – SNTP Server Port to connect to - R/W

This register contains the TCP port that should be used for SNTP connections. The Default is 123. Default is 123.

20042 – SNTP Update Time - R/W

This register contains the number of seconds before the next synchronization request with the SNTP server. For example 3600 would be an hour, 86400 would be 24 hours. Default is 86400. When a change in time is made to this value it typically takes about 1 minute before the new value will take effect. Power cycling of the controller is not required.

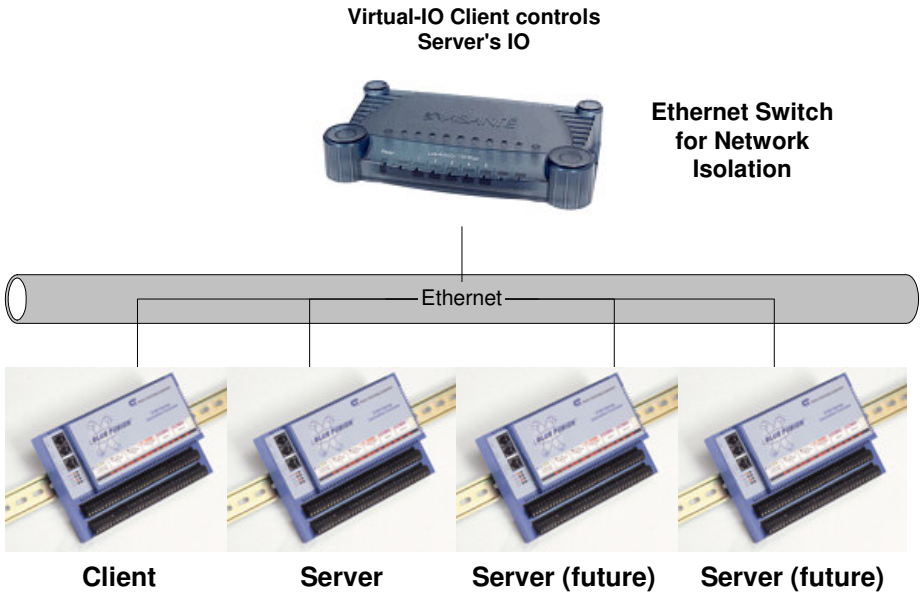
20043 – SNTP Offset from GMT - R/W

This register contains the number of seconds to add or subtract from GMT. The default is 0, which means to set the clock to GMT. –14400 would be the value used for Eastern Standard Time during daylight savings time. Note that the value is both positive and negative.

Note that a '1' must be written to register 20096 whenever the above changes are made in order to store those changes to non-volatile storage. Also to disable SNTP simply set the IP Address of the SNTP Host to 0.0.0.0.

7.0 Virtual I/O Mapping

Virtual IO allows a 5100 controller (client) to extend its module bus to that of another unit, via Ethernet. In essence the 2nd 5100 controller (server) allows the client to use its I/O as though it was local to the client, transparent to the application programmer. Each remote module is added to the list of available modules. If there are 8 local digital inputs and a client attaches to a virtual IO server that has 16 digital inputs, then the client will report a total of 24 digital inputs available to Quickstep. The remote IO is added to the end of the local IO. In the previous example inputs 1-8 are local and 9-24, remote.



Note: Motion Modules are not supported on the second controller (server).

Some applications of Virtual I/O Mapping Include:

- Adding additional I/O resources beyond the capacity of a single controller unit.
- Coordinating the actions of different controllers controlling interacting machines or different operations on a single machine.
- Collecting Data from another 5100 controller.

The attachment to a remote controller is transparent to the local one, except that it will cause some performance degradation. A server can allow up to (1) client controller to use its local bus. In addition a client can connect up to (1) server. A critical input or output should never be remote. For example, damage to equipment could occur if you were to enable an output to heat an oven, or move an actuator, and someone disconnected the Ethernet cable. With the cable disconnected the output could not be turned off. A better way to perform critical functions is to use peer to peer to request a remote Quickstep program to control the function with its local I/O. Both peer-to-peer and Virtual IO can be used at the same time, between the same units.

In addition to sharing its module bus, a Virtual IO server makes its first 96 volatile and 96 non-volatile registers public. These registers can be accessed anywhere in the 23000 register block, as detailed later. Although this same function can be done using peer to peer, Virtual IO, has an added feature, which can be extremely important. In addition to being able to adjust the scan rate of IO and public registers,

independently, a high priority data mask exists on all virtual objects. By setting the appropriate mask bit, any change of state with regards to that object will immediately be reported back to the client. This is extremely useful so you can set a slow scan rate (100 ms.) but still want to be notified immediately (a few milliseconds) if the contents of a specific public register changes, or a digital input/output line changes state.

7.1 Quickstep Configuration

Eight register blocks of 100 registers are reserved for configuration (only one supported currently), starting at register 21600. Note the peer to peer register blocks are 21000 to 21199. Although there are a large number of registers, most are used for advanced high priority data masking and are not necessarily accessed during normal operation, if at all. By default priority data masking is disabled, set to 0.

The following steps are needed to configure a Virtual I/O connection:

1. Make sure all units have properly configured IP addresses.
2. Set the Virtual I/O unit's IP Address (Reg 21600 – 21603)
3. Set the Start Register for the Mapped registers (Reg 21604)
4. Write a 1 to the Command Register (Reg_21607) to Initialize the Setup
5. Check to see that the Status Register (Reg_21605) is -2.
6. Write a 2 to the Command Register to Initiate the Connection
7. The connection is made when the command register returns a 1.

The following Quickstep code shows how this is done:

```
[2] Setup_Virtual_IO_Blocks
    ;;; Set up and Initialize Peer Block 1
    ------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------
    store 12 to Reg_21600
    store 40 to Reg_21601
    store 53 to Reg_21602
    store 246 to Reg_21603
    store 23000 to Reg_21604
    store 1 to Reg_21607
    if Reg_21605=1 goto Next   ;;; If already Connected!
    if Reg_21605=-2 goto Next

[3] Initiate_Virtual_IO_Connection
    ;;; Create connection for Peer Block 1
    ------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------
    store 2 to Reg_21607
    if Reg_21605=1 goto Next
```

7.2 Registers

The following are the Registers used for Virtual I/O Connections

21600 – 21699 – Virtual I/O Block 1

(Note that the following blocks are available, but not supported at this time)
21700 – 21799 – Virtual I/O Block 2
21800 – 21899 – Virtual I/O Block 3
21900 – 21999 – Virtual I/O Block 4

21X00 VIRTUALIO_IPA
First Octet of the IP dot address (XXX.000.000.000) to connect to.

21X01 VIRTUALIO_IPB
Second Octet of the IP dot address (000.XXX.000.000) to connect to.

21X02 VIRTUALIO_IPC
Third Octet of the IP dot address (000.000. XXX.000) to connect to.

21X03 VIRTUALIO_IPD
Fourth Octet of the IP dot address (000.000.000. XXX) to connect to.

21X04 VIRTUALIO_REMAP_STARTREG
This register defines where in the 23000 - 24999 block all the Virtual IO registers are to be viewed. There are 192 registers, the first 96 volatile, and the second block, 96 nonvolatile. This same block is also used for peer-to-peer register mapping also so be careful not to overlap areas. This register must be set. Typically the first connection would be set to 23000, the second to 23200, third to 23400, etc… Setting this value to 0, at initialization, will disable Virtual IO register updates, thereby decreasing network traffic.

21X05 VIRTUALIO_STATUS
Read only register:

STATUS	*DESCRIPTION*
-1	Not initialized or never connected to a server
-2	Initialized only.
0	Offline; no connection is present.
1	Connecting
2	Retrieving module bus information.
3	Retrieving first complete set

	of system data.
10	Aborted operation; out of local memory or resources.

21X06 VIRTUALIO_WINDOWSIZE

This register may be used to reduce the number of registers that will appear in the 23000 block based upon the VIRTUALIO_REMAP_STARTREG setting. By default this value is 192, but can be made smaller. Note that registers 1 to 96 are the volatile registers, registers 97 to 192 are the non-volatile registers starting at 501 in the remote controller. Setting this value to 0 at anytime will disable all register scanning updates.To notify the server of the change a VIRTUALIO_REQUEST_SCANUPDATE command must be issued.

21X07 VIRTUALIO_COMMAND

1 - *VIRTUALIO_COMMAND_CREATE_SERVER*
First command initiated to create a server block prior to connection, once IP address and register viewing blocks are defined.

2 - *VIRTUALIO_COMMAND_INITIATE_CONNECTION*
This is the second command, used to actually connect to a Virtual IO server and add its IO to your controller. This command only needs to be initiated once, retries are automatic.

The following commands are used to send the current mask configuration to the server, changing it's high priority mask accordingly or setting new scan rates.

9 - *VIRTUALIO_REQUEST_SCANUPDATE*
Sets the scan rates of the server to that defined in the VIRTUALIO_SCANIO_RATE and VIRTUALIO_SCANREGS_RATE registers. No change is made to the server until this command is initiated.

10 - *VIRTUALIO_REQUEST_MASK_DIGITALINPUT_UPDATE*
This command sets the server to the high priority mask defined in the VIRTUALIO_DI_MASK block, there are 32 registers, representing 1024 bits or input points.

11 - *VIRTUALIO_REQUEST_MASK_DIGITALOUTPUT_UPDATE*
This command sets the server to the high priority mask defined in the VIRTUALIO_DO_MASK block, there are 32 registers, representing 1024 bits or input points.

12 - *VIRTUALIO_REQUEST_MASK_REG_UPDATE*
This command sets the server to the high priority mask defined in the VIRTUALIO_VOLATILE_REG_MASK block, there are 3 registers, representing 96 bits or input points.

13 - *VIRTUALIO_REQUEST_MASK_NVREG_UPDATE*
This command sets the server to the high priority mask defined in the VIRTUALIO_NONVOLATILE_REG_MASK block, there are 3 registers, representing 96 bits or input points.

14 - *VIRTUALIO_REQUEST_MASK_FULL_UPDATE*
This command sets the server to the high priority mask and scan rates defined in all the masks, all at once. Note that this is automatically done upon initial connection but is a fast way to make massive changes, all at once, during operation.

21X08 VIRTUALIO_SCANIO_RATE

This register displays/sets the current rate at which all IO data points available in the server are transferred to the client, by default 100 milliseconds. Note that all Analog Inputs are transmitted at this rate and there is no high priority mask available for them. The VIRTUALIO_REQUEST_SCANUPDATE command must be initiated to activate any changes.

21X09 VIRTUALIO_SCANREGS_RATE

This register displays/sets the current rate at which all public data registers available in the server are transferred to the client, by default 100 milliseconds. The VIRTUALIO_REQUEST_SCANUPDATE command must be initiated to activate any changes.

21X10 VIRTUALIO_WRITEREG_INDEX

This register can be set to point to any valid register in the server. The data written to the VIRTUALIO_WRITEREG_VALUE register will be written to that register. This register is not limited to the public registers. Note that you can only write the register, not read it back. There are no limitations as to what this register is.

21X11 VIRTUALIO_WRITEREG_VALUE

This value written to this register will be written to the server register whose index is contained in the VIRTUALIO_WRITEREG_INDEX.

21X12 to 21X15 Reserved for future use

Note: The appropriate MASK command must be initiated for any of the following MASK register changes to take effect. Changes may be made in multiple registers and then a single command used to activate them all at once.

21X16 – 21X47 VIRTUALIO_DI_MASK

These 32 registers consist of the high priority digital input mask registers. To activate set one of 1024 bits, representing an IO point in the remote controller. Bit 0 in the first register represents IO point 1. When a bit is set, any change of state will be sent immediately to the client. Note the more bits you set the higher the load on the controller given the additional network and monitoring traffic. It is advised that this be used for change of states that are important to know about quickly but that do not change that often.

21X48 – 21X79 VIRTUALIO_DO_MASK

These 32 registers consist of the high priority digital output mask registers. To activate set one of 1024 bits, representing an IO point in the remote controller. Bit 0 in the first register represents IO point 1. When a bit is set, any change of state will be sent immediately to the client. Note the more bits you set the higher the load on the controller given the additional network and monitoring traffic. It is advised that this be used for change of states that are important to know about quickly but that do not change that often.

21X80 – 21X82 VIRTUALIO_VOLATILE_REG_MASK

These 3 registers consist of the high priority volatile register mask registers. To activate set one of 96 bits, representing a volatile register point in the remote controller (register 1 to 96). Bit 0 in the first register represents remote register 1. When a bit is set, any change of state will be sent immediately to the client. Note the more bits you set the higher the load on the controller given the additional network and monitoring traffic. It is advised that this be used for change of states that are important to know about quickly but that do not change that often.

21X83 – 21X85 VIRTUALIO_NONVOLATILE_REG_MASK

These 3 registers consist of the high priority non-volatile register mask registers. To activate set one of 96 bits, representing a non-volatile register point in the remote controller (register 501 to 596). Bit 0 in the first register represents remote register 501. When a bit is set, any change of state will be sent immediately to the client. Note the more bits you set the higher the load on the controller given the additional network and monitoring traffic. It is advised that this be used for change of states that are important to know about quickly but that do not change that often.

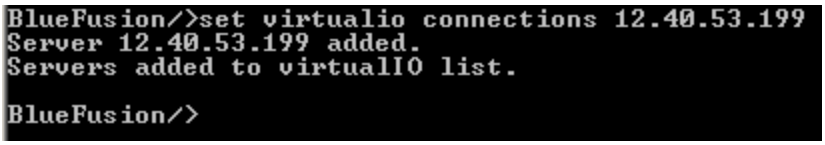
7.3 Script Configuration

Virtual IO may be configured either from a script text file (.ini) and/or from within Quickstep via registers. Either way, Quickstep can be used to redefine any initialized values during operation. A number of script commands exist to initialize Virtual IO for operation. The first, “set virtualio connections”, lists the IP addresses of other controller’s to connect to, and is required, prior to any other commands.

set virtualio connections [IP address list]

where [IP address list] is up to 4 IP addresses separated by a comma
(this command can only be executed once per power-up, executing it again will have no effect).

This initializes a list of connections that are to be made, once Virtual IO is enabled. The modules present on these remote systems will appear as though they are local. Once this list is initialized it cannot be changed without cycling power or executing a ‘telnet’ reset command.



set virtualio state change [IP address] [resource type] [range list]

Where:

[IP address] = one of the IP addresses set in the connection list.

[resource type] = DI for digital input, DO for digital output, REG for volatile register, and NVREG for non-volatile register.

[range list] = resource number as referenced in local system. For example register 1 would be 1, digital outputs 10-20 would be 10-20. This sets the high priority monitoring mask for immediate change of state notification.

This sets the high priority monitor bit mask. There are a possible 1024 digital inputs, 1024 digital outputs, 96 volatile registers and 96 non-volatile registers. The non-volatile registers start with register 1, when setting the mask (references 501 in the controller). Note that after setting the state change mask for a particular resource type, the current settings are output the telnet screen, along with volatile and nonvolatile register contents (assuming manual telnet mode versus a script file).

```
BlueFusion/>set virtualio state change 12.40.53.199 DI 1-8
State change modified:
VirtualIO Server Connection state:
IP = 12.40.53.199:
  State = INITIALIZED ONLY.
  Scan rate = 100 MS. IO, 100 MS. Regs.
  Digital Input Mask = 000000FF 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000
  Digital Output Mask= 00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000
  Volatile Reg. Mask = 00000000 00000000 00000000
  NVolatile Reg. Mask= 00000000 00000000 00000000

  Volatile Regs      = 00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000

  NONVolatile Regs   = 00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
                       00000000 00000000 00000000 00000000 00000000 00000000
```

Note above that the Digital Input Mask is set to 0x000000FF, this equates to DI 1-8 shown in the 'state change' command line.

set virtualio scan IO rate [value in milliseconds]

Set the update scan rate to be used for the IO, default = 100 milliseconds.

set virtualio scan Reg rate [value in milliseconds]

Set the update scan rate to be used for the public registers, default = 100 milliseconds.

get virtualio server connections
List all present server connections defined, state, and initialization information.

get virtualio client connections
List all present client connections using this server's IO

enable virtualio
This command will cause a connection to be made to all defined servers and their IO attached to this client. Control is not returned until all servers are ready for operation.

7.4 Sample VirtualIO Program: VIOCount.dsp

Below is a sample program allowing a client to share the I/O of a server at IP address 12.40.53.197. ***Note, that whenever Virtual IO is used it should be setup and online prior to any other controller operations being done. Should a connection to the server be lost the Quickstep program will fault with a code of '49', Unknown Step and the server will maintain outputs in the last known state. It is suggested that a Fault Handler be used (Section 7.6), to trap the error and recover if needed. The below code includes an example of such a handler.***

Symbols:

Registers	*Symbol Name*
1	lastTime
2	avgTime
3	maxTime
4	sum
5	cyclecount
6	outputVal
10	FaultFlag
10101	Reg_10101
11101	reg11101
13002	time
13038	FaultStepRegister
13039	FaultTaskRegister
13040	FaultMaskRegister
13041	FaultClearRegister
21600	v_IPA
21601	v_IPB
21602	v_IPC
21603	v_IPD
21604	v_remapstart
21605	v_status
21607	v_command

```
        21608       v_IOscanrate
        21616       v_DI_MASK

[1] init
   ;;; This program is setup to interface with a single remote 5100
   ;;; via VirtualIO.  Two Digital Input (5110) and two Digital Output
   ;;; (5120) modules exists in the remote controller.  The Outputs of
   ;;; the 5120 module are looped back to the inputs of the 5110.
   ;;; .
   ;;; Upon power up we must see what state we were in.  This
   ;;; may simply be a program restart and the connection could
   ;;; already be active.
   ;;; .
   ;;; Check the Virtual IO status
   ;;; 1  = ready
   ;;; -1 = requires full initialization
   ;;; -2 = already initialized, just need to initiate connection
   ;;; .
   ;;; Also note that a Fault Handler is installed in the first
   ;;; step to monitor for communications failure.  The FaultMaskRegister
   ;;; must be set after the FaultStepRegister, otherwise the handler
   ;;; will be invoke for all faults (default).
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 0 to sum
   store 0 to cyclecount
   store 8 to FaultStepRegister
   store 32 to FaultMaskRegister
   if v_status=1 goto execute
   if v_status=-1 goto v_setup
   if v_status=-2 goto go_online

[2] v_setup
   ;;; Initialize Virtual IO for operation, remote IP address
   ;;; is 12.40.53.197.  Also set high priority mask for
   ;;; Digital Inputs 1-8.  Init the register remap capability
   ;;; to the 23000 block.  192 registers will be present.
   ;;; To disable this feature simply set the v_remapstart register
   ;;; to a 0.  Finally create the server register block by writing
   ;;; a 1 to v_command.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 12 to v_IPA
   store 40 to v_IPB
   store 53 to v_IPC
   store 197 to v_IPD
   store 23000 to v_remapstart
   store 1 to v_command
   store 50 to v_IOscanrate
   goto Next
```

```
[3] go_online
   ;;; Now initiate the actual connection to other 5100 now and add its
   ;;; sticks to our buss.  Also note the high priority mask
   ;;; was set to the Digital Inputs.  Do not use the high
   ;;; priority mask on fast acting, repetitive signals.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store 255 to v_DI_MASK
   store 2 to v_command
   goto Next

[4] wait_for_virtualIO
   ;;; Loop here until we finish connecting to other 5100
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   if v_status=1 goto execute
   delay 100 ms goto wait_for_virtualIO

[5] execute
   ;;; Now begin main program execution.
   ;;; .
   ;;; This program sets a digital output, then looks for that
   ;;; output on its input, timing how many milliseconds it
   ;;; took (time).  Begin by reading the current output value
   ;;; and incrementing it by one.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store Reg_10101 + 1 to outputVal
   if outputVal <=65535 goto monitor_change
   store 0 to outputVal
   goto Next

[6] monitor_change
   ;;; Now wait for the set digital output to be read as an
   ;;; active input.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store outputVal to Reg_10101
   store 0 to time
   if outputVal=reg11101 goto IO_changed

[7] IO_changed
   ;;; Save the number of milliseconds took for Quickstep to
   ;;; set and detect the input change to "lastTime".  Also
   ;;; set the largest time it took so far in "maxTime" and
   ;;; maintain a running average in "avgTime"
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store time to lastTime
```

```
    store cyclecount + 1 to cyclecount
    store sum + time to sum
    store sum / cyclecount to avgTime
    if time <=maxTime goto execute
    store time to maxTime
    goto execute


[8] FaultHandler
    ;;; This step is invoked should a fault occur, such as a
    ;;; network disconnect.  The FaultMaskRegister controls
    ;;; under what circumstances the handler is invoked.  This
    ;;; example is very simple.  It basically shuts all the
    ;;; local outputs off and sets a flag in FaultFlag that
    ;;; has no purpose.  Note that no other tasks will be running
    ;;; in the system nor can this task fault when the handler
    ;;; is invoked.
    ------------------------------------------------------------------------
    <TURN OFF ALL DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    store 1 to FaultFlag
    delay 3 sec goto ClearFault

[9] ClearFault
    ;;; Now attempt to recover from the fault by issuing a RESTART
    ;;; command
    ------------------------------------------------------------------------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
    store 2 to FaultFlag
    store 5 to FaultClearRegister
    goto FaultHandler
```

7.5 Important Considerations

7.5.1 Network Isolation

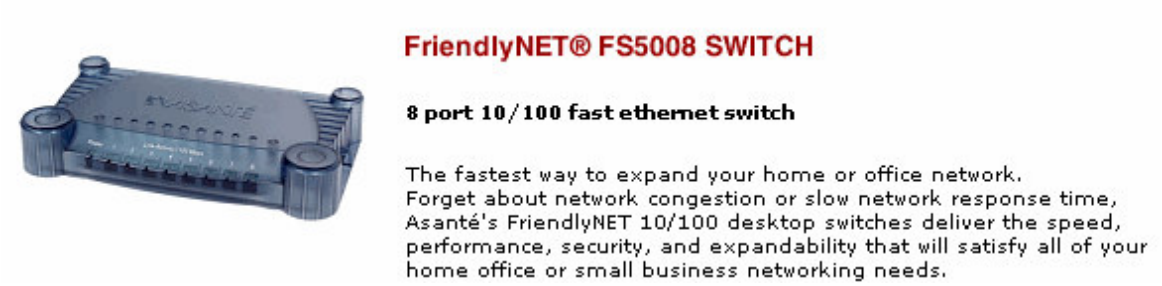
Always isolate the controller network from the main network. A switch, or a switch integrated firewall, should be used in order to help restrict the amount of data packets flowing on the controller network. Several inexpensive devices are available by Asante, with a list price under $100. The FS5005 and FS5008 switches are recommended, with the FS5008 costing around $65.

FriendlyNET FR3004FLC Internet Router with Firewall

Safely share your high-speed Internet connection with all your computers!

Graphic files, video streams, MP3 music files. And even everyday email with large attachments. Sometimes you wonder how you've been getting along without a broadband Internet connection. Now you want to share that power with others in your office or home.

http://www.asante.com/products/routers/FR3004FLC/index.html



http://www.asante.com/products/switches/5000/index.html

The ideal situation is a crossover cable between the client and a controller server, thereby total isolation, but this is only useful when connectivity to the outside world can be over a serial port. Ethernet is like a garden hose, the more you have flowing through it the harder it is to be deterministic with when your data will arrive at the desired points. Typically it is fast, less than 5 milliseconds from a Quickstep output command to it changing on the remote server, but if packets collide or get lost this could be delayed up to 250 milliseconds. The majority of the time high priority updates will arrive within 10 milliseconds of their detection but you need to design your system to handle the one time that a packet could be delayed, a connection lost, or someone disconnects an Ethernet cable. A good rule of thumb is if something will break if the Ethernet cable was disconnected and you could not access your IO, then don't make it remote.

7.5.2 Remote Write Operations

All network write operations, register, analog, and digital, are write verified. This means that the local copy of a value is not updated until the controller will receive verification that the write completed successfully. This increases network traffic since for every write to a remote server, that server will immediately turn around and send back the exact same packet, thereby updating the local copy. The local copy will also be updated upon every scan, default of 100ms. If an output is critical for a particular application and you need to verify that it reached it proper state 100% of the time and not just 99.9%, then it is suggested that after you do a write, read back from the same location to confirm the value is what you wrote. This should be done waiting with a timer since the value change will take several milliseconds. In most cases 2 ms to send it and 2-3ms to get it back. Other network traffic can also add several milliseconds should collisions occur.

7.5.3 Performance and Determinism Guidelines

Performance and determinism is not only controlled by traffic on the Ethernet but also system load. High system load will affect the capabilities of the controller to respond to incoming Ethernet traffic in a timely manner. Operations such as servo profiling, thermocouple calculations, user loadable "C" program, ftp, telnet, displays, peer-to-peer, CTCMON, etc, can all affect performance. Basically this just means that you must perform system tests under real-world load conditions. There are too

many variables to define absolute performance rules for determinism. However testing at CTC labs has shown it to offer on average a very high speed results.

CTC Test Timings

Our testing was done on a system set-up with two virtual servers as described below. The tests showed that it typically 2 – 9 ms to write to a remote IO module and receive verification that the write has occurred. This is extremely fast compared to typical PLC scan rates, but again, keep in mind your actual results will depend on what applications are running on the client and server as well as your overall network configuration and traffic.

CTC Test Set-up

For the test executed the test code enclosed within this document was used. No applications were running on the servers and no collisions occurred. A sampling over a period of 12 million write operations was collected. The configurations of the controller's were as follows:

Client: 5102-XXXJXX
Server: 5102-AABBJX
(A and B modules on the server were looped back to each other)
Asante FS5008 switch (all devices, including PC plugged into switch)
CTCMON connected to Client from PC running UDP monitoring registers 1-16
Ethernet @ 10Mb
Scan Rate – 50ms for modules, 100ms for 192 byte virtual registers

Results were as follows:

- Minimum Single Write Acknowledge time – 1.4 ms.
- Maximum Single Write Acknowledge time – 6 ms (state change packet on output written was sent first and ACK was queued, resulting in delay. In actuality this packet would have resulted in the output state containing the update in 3.5ms.).
- Average Program dual output write with Acknowledgement and detecting change of state – 9 ms
- Maximum Program dual output write with Acknowledgement and detecting change of state – 25ms

Note: The dual output write is due to the fact that the test program logic does a 16 bit write to the outputs. Each module update information is sent as a separate data packet on the network, hence two writes for high/low bytes and two acknowledge packets. Change of state is sent to the client 5100 as soon as the looped back input is detected (sampled every millisecond).

Good uses for Virtual IO are:

1. Expansion of a client controller module bus with Analog inputs, thermocouples, non-critical digital inputs/outputs, etc…

2. To share registers and utilize the high priority mask capability not available in peer-to-peer.
3. A way to synchronize and trigger Quickstep operations based on events happening in another controller.

Notes:

1. ***The COM1 Serial port should be run at only 9600 baud when using Virtual IO with change of state enabled. Higher baud rates can be used but heavy Ethernet traffic could result in overruns.***
2. ***Issuing the Software Reset command will cause the controller to do a hard reset and reboot when Virtual IO is active. When Virtual IO sessions are not active a normal Software Reset will be done. This should have no effect on operation except that if diagnostics detects a problem with a module the unit will enter a fault state. Typically this fault would be caused by a module being detected by failing to respond. This is not normal operation and means the unit requires service.***

7.6 Fault Task Handler

When Quickstep programs encounter problems they fault, removing control from the programmer. A new feature is called the "Fault Task Handler". The "Fault Task Handler" is a regular Quickstep task that can be branched to and executed when a soft fault occurs. The Handler is simply a standard Quickstep program. Either a separate task that is looping on a 'delay' instruction awaiting the fault, or a main program which sets the "Fault Task Handler" step and continues executing. Later, branching to the step designated to handle the fault.

There can only be one "Fault Task Handler" active at a time. Any task can be activated as a handler by writing a step number to branch to in register 13038, the TASK_FAULT_STEP_REGISTER. A branch will occur to the designated step when a Fault occurs. You can change which task is the handler or where to branch to at any time, by setting 13038 to a different step, or to 0 to disable the handler. Register 13040, TASK_FAULT_MASK_REGISTER can be set to enable which faults will cause the branch to occur. Each bit is OR'd as required to enable each type of Fault:

FAULT MASK	*FAULT TYPE*
0x0001 (1)	Fatal Errors
0x0002 (2)	Program Errors
0x0004 (4)	Motion Errors
0x0008 (8)	Analog Errors
0x0010 (16)	Digital Errors
0x0020 (32)	Communications Errors

When a Handler is executing it will ignore further soft faults and continue executing. The fault state must be cleared for normal operation to continue. This is controlled by register 13041, the TASK_FAULT_CLEAR_REGISTER (Write Only). This register controls the state of program execution:

PROGRAM STATE	DESCRIPTION
1	RESET – Reset the controller only and then stop..
5	RESTART – Reset the controller and begin running again at step 1.
6	STOPPED – Stop the controller but do not reset.
8	RUNNING – Ignore the fault and continue running.
9	FAULT – Continue to fault as usual.
10	SHUTDOWN – Reset the controller and shutdown, requires a power cycle to exit.

Important registers are as follows:

REGISTER	DESCRIPTION
13032	Fault Code – (R) Contains the fault code for what caused the fault.
13033	Fault Step – (R) Step in which fault occurred.
13034	Fault Task – (R) Task number, starting at 1, which caused the fault..
13035	Fault Data – (R) Any relevant error data.
13038	Fault Step Register – (R/W) Step to branch to when fault occurs. Write a 0 to disable.
13039	Fault Task Register – (R) Task number that is the active Fault Handler, 0 means none.
13040	Fault Mask Register – (R/W) Bit OR of types of fault that will invoke the handler, by default all enabled (-1) when the Fault Step Register is written
13041	Fault Clear Register – (W) Used to write the recovery state when done processing the Fault.

Refer to Section 7.5 for an example of a fault handler.