



PowerTools Pro Application Note
PTAN #3, rev. 2, 1/14/2010
Applicable Products: Epsilon EP-P, FM4E

**Modbus Master
&
Modbus Gateway**



SCIGATE AUTOMATION (S) PTE LTD
No.1 Bukit Batok Street 22 #01-01 Singapore 659592
Tel: (65) 6561 0488 **Fax:** (65) 6562 0588
Email: sales@scigate.com.sg **Web:** www.scigate.com.sg
Business Hours: Monday - Friday 8.30am - 6.15pm

Introduction

Modbus Master allows the EP-P drive to communicate, as a Modbus master, to any other Modbus device on an RTU (serial) and/or TCP (ethernet) network. Typical uses are to stop/start another drive, read I/O from another drive, read I/O from an external I/O block.

The step for using Modbus master

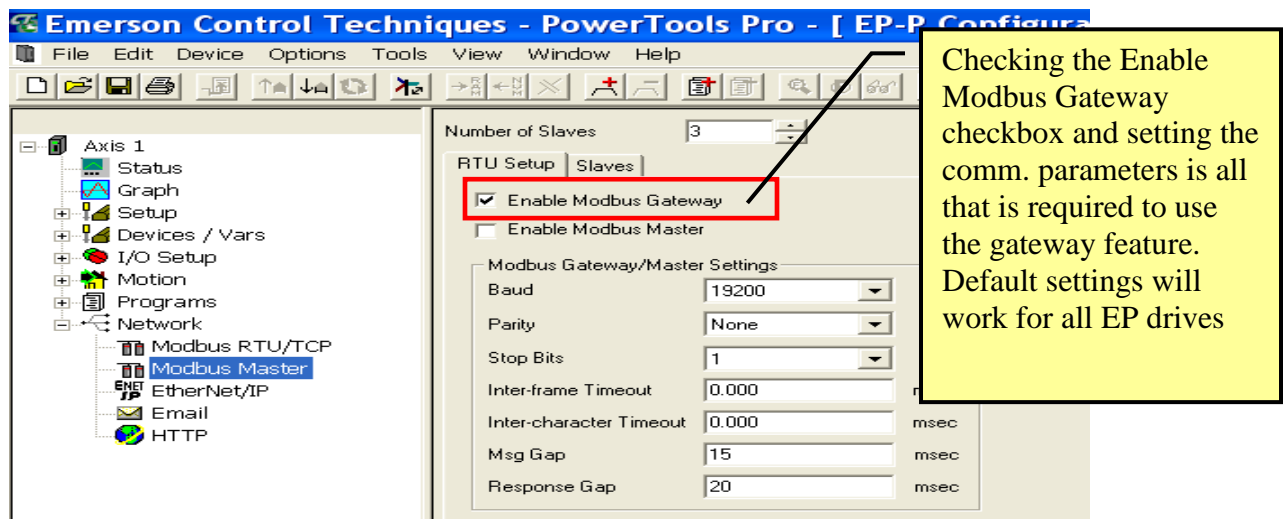
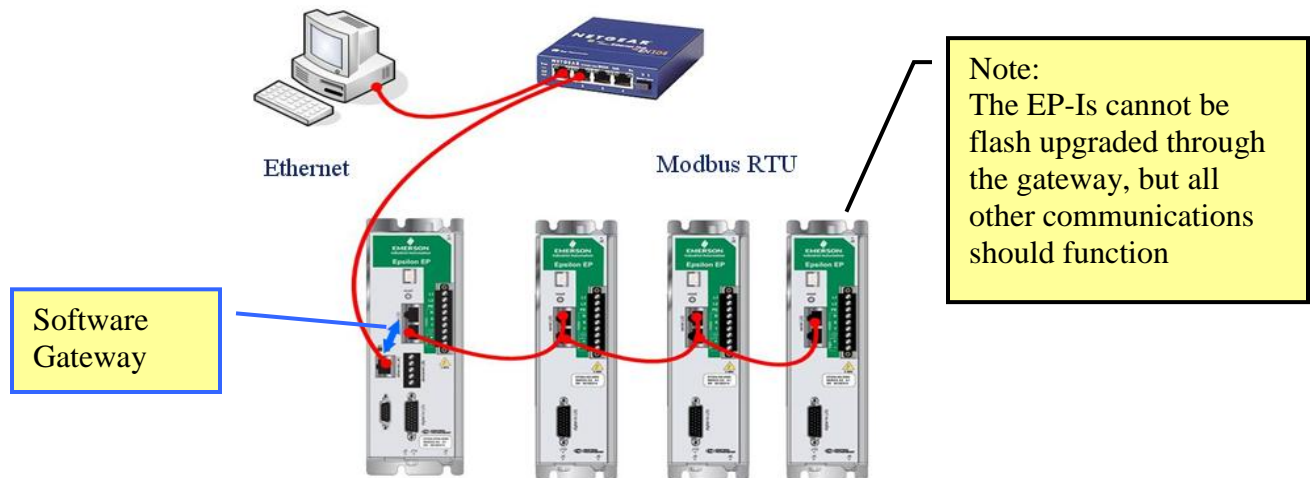
1. Configure/commission each slave device
2. Configure the slave network communication parameters in PowerTools in the Master view
3. Configure the slave map in PowerTools under Modbus Master, Slave tab
4. Program Read/Write commands in the PowerTools program

Modbus Gateway is a feature that allows the EP-P drive to pass through all communications from the ethernet port to the Modbus RTU serial port. The gateway is independent of the Modbus Master feature.

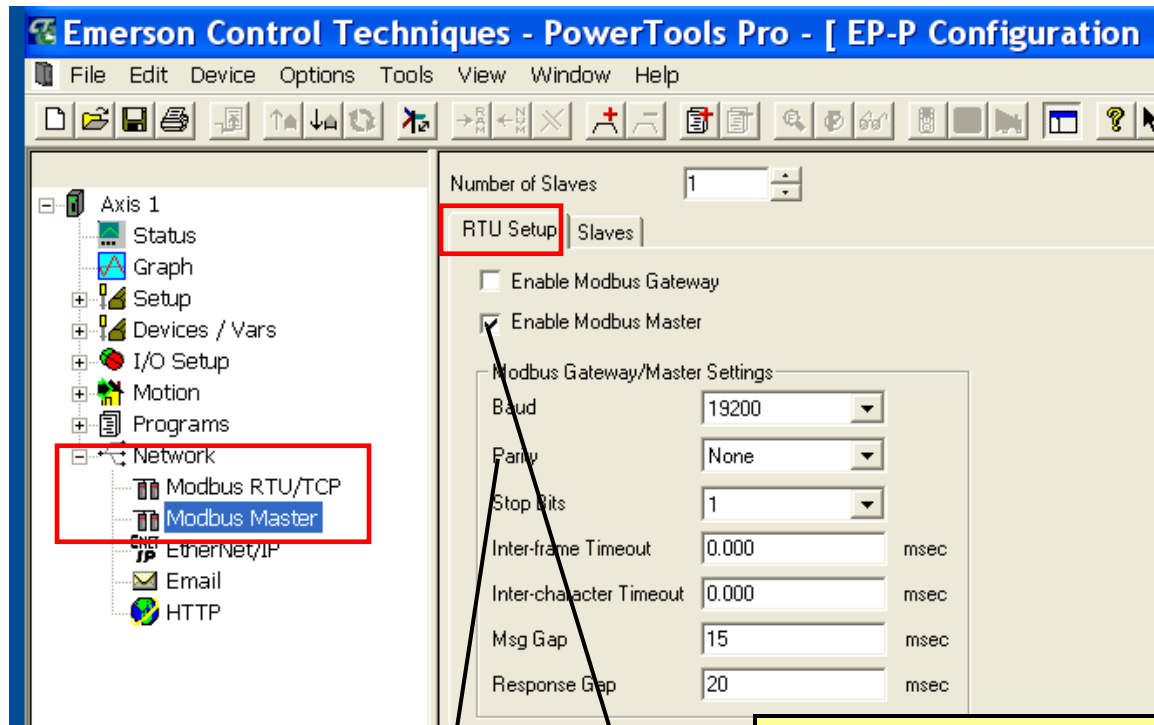
Modbus Gateway

Modbus Gateway is a feature that allows the EP-P drive to pass through all communications from the ethernet port to the Modbus RTU serial port. **The gateway is independent of the Modbus Master feature.**

In this example, with the Gateway enabled, PowerTools software has communications ability with the EP-I drive(s) over Modbus RTU through the EP-P gateway as shown in this graphic:



Modbus Master View



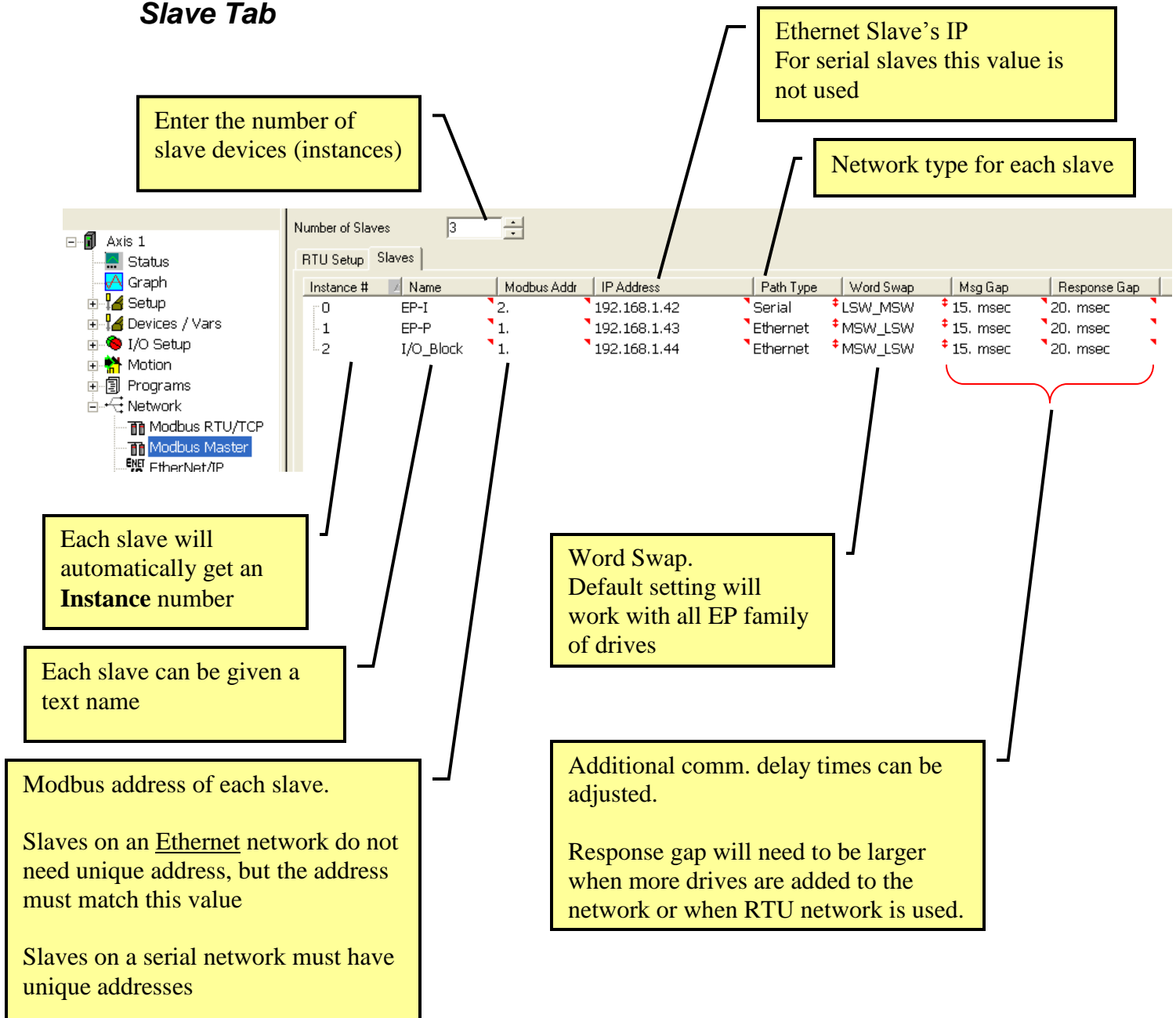
Enter the Modbus RTU slave network communication settings. Default settings will work with all EP family of drives.

Check the Enable Modbus Master checkbox - This allows the drive to become the Modbus master on the RTU port.

For RTU only 1 master can exist on the network, this implies that PowerTools and the EP-P cannot both be a master. Communications issues will occur if multiple master are on the same RTU network

For TCP, multiple masters can exist.

Slave Tab



Enter the number of slave devices (instances)

Ethernet Slave's IP
For serial slaves this value is not used

Network type for each slave

Instance #	Name	Modbus Addr	IP Address	Path Type	Word Swap	Msg Gap	Response Gap
0	EP-I	2.	192.168.1.42	Serial	LSW_MSW	15. msec	20. msec
1	EP-P	1.	192.168.1.43	Ethernet	MSW_LSW	15. msec	20. msec
2	I/O_Block	1.	192.168.1.44	Ethernet	MSW_LSW	15. msec	20. msec

Each slave will automatically get an **Instance number**

Each slave can be given a text name

Modbus address of each slave.
Slaves on an Ethernet network do not need unique address, but the address must match this value

Slaves on a serial network must have unique addresses

Word Swap.
Default setting will work with all EP family of drives

Additional comm. delay times can be adjusted.

Response gap will need to be larger when more drives are added to the network or when RTU network is used.

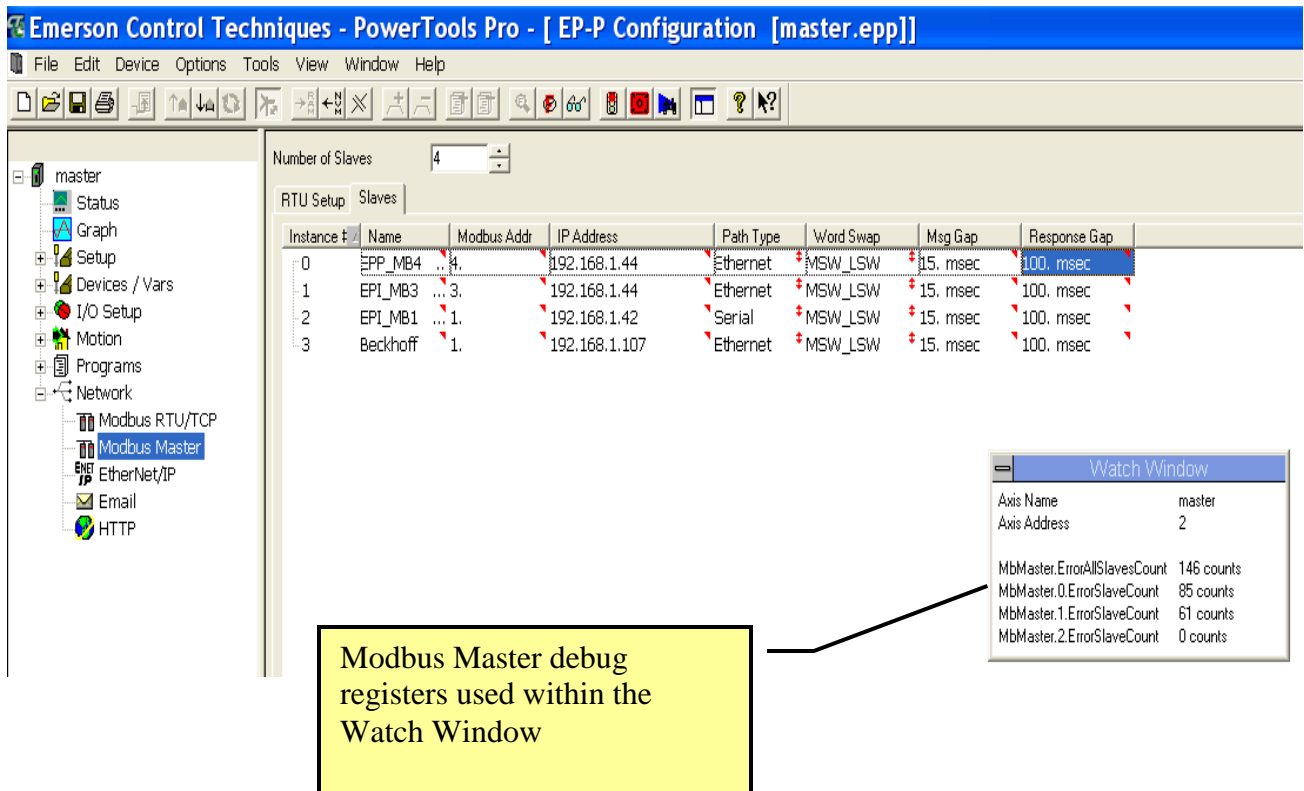
Setting the Response Gap

When the response gap is too small the drive will detect the network's slave slow response as an error and increment the parameters MbMaser.x.ErrorSlaveCount value. These errors are not a problem if they occur infrequently, since the program can reissue the request for the dropped message. However if the counts are incrementing with every message, then the response gap should be increased.

For RTU networks or for a drive using the TCP/RTU gateway a response gap of 50-100 ms is typical.

For TCP only networks a Response Gap of 20-40 ms is typical

More nodes on the network will also require an increase in the Response Gap value.



The screenshot shows the PowerTools Pro configuration window for an EP-P drive. The 'RTU Setup' tab is active, displaying a table of configured slaves. The 'Response Gap' column is highlighted in blue for the first slave (EPP_MB4). A 'Watch Window' is open, showing the 'MbMaster.ErrorSlaveCount' for the selected slave (master, address 2). The counts are: MbMaster.0.ErrorSlaveCount: 146 counts, MbMaster.1.ErrorSlaveCount: 85 counts, MbMaster.2.ErrorSlaveCount: 61 counts, and MbMaster.3.ErrorSlaveCount: 0 counts. A yellow box with an arrow points to the Watch Window, containing the text: 'Modbus Master debug registers used within the Watch Window'.

Instance	Name	Modbus Addr	IP Address	Path Type	Word Swap	Msg Gap	Response Gap
0	EPP_MB4	4	192.168.1.44	Ethernet	MSW_LSW	15. msec	100. msec
1	EPI_MB3	3	192.168.1.44	Ethernet	MSW_LSW	15. msec	100. msec
2	EPI_MB1	1	192.168.1.42	Serial	MSW_LSW	15. msec	100. msec
3	Beckhoff	1	192.168.1.107	Ethernet	MSW_LSW	15. msec	100. msec

Watch Window	
Axis Name	master
Axis Address	2
MbMaster.ErrorAllSlavesCount	146 counts
MbMaster.0.ErrorSlaveCount	85 counts
MbMaster.1.ErrorSlaveCount	61 counts
MbMaster.2.ErrorSlaveCount	0 counts

Modbus Master debug registers used within the Watch Window

Establishing Communications

On the very first time a drive attempts to communicate with a Modbus/TCP slave node, a socket needs to be established. The socket typically takes 5-10 seconds to establish.

If you plan on using Modbus master messages in the Cyclical Program, code a standard user program to establish the socket, and then enable the cyclical program for messaging. Otherwise the Cyclical Program will fault when it has to wait too long for the socket to establish

If you are using the message commands only in a standard program this approach is not required.

Understanding Modbus Protocol

Modbus protocol is very simple, writing (or reading) a value to a given register will allow the master to communicate to a slave device.

Each type of register has specific ranges:

Outputs (or coils) are mapped to address range of 0-9999.

Input Status are mapped to address range of 10000 - 29999.

Output Integers (or Holding Registers) are mapped to address range of 30000 - 39999.

Input Integers (or Input Registers) are mapped to address range of 40000 - 49999.

All devices that use Modbus will have a manual that states the Modbus address and its associated parameter meaning.

Examples:

Writing a 1 to address 99 would start a drive's index

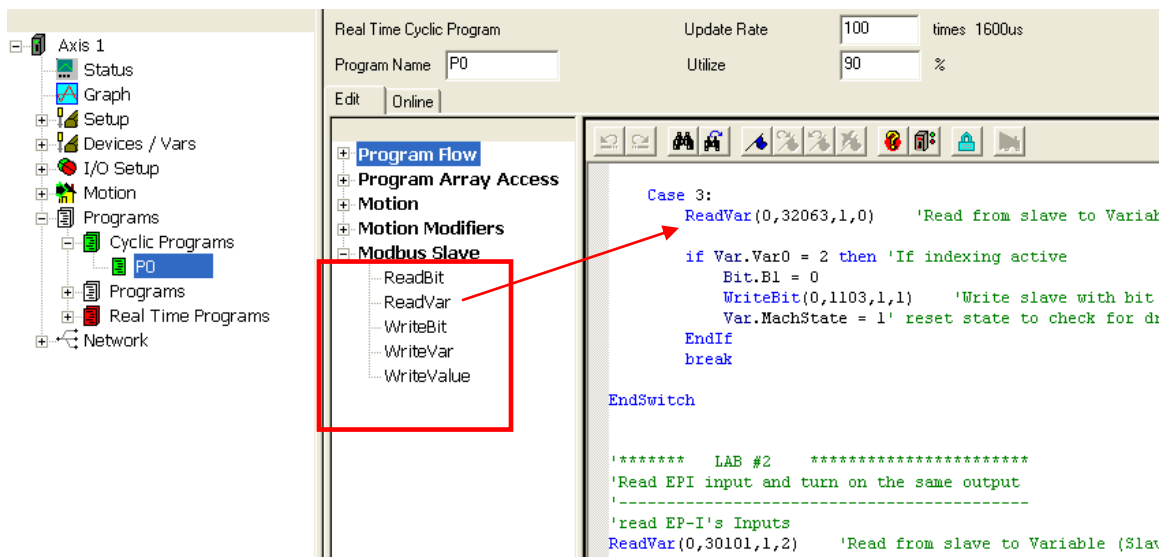
Reading from address 40001 would return the drive's velocity

For all Epsilon drive products refer to the 'Drive Parameter Manual'.

Program View

Within user programs or the cyclical program in PowerTools, the Modbus Slave tree can be used to access the correct syntax of the Modbus message formats. Simply Drag & Drop the Read/Write syntax to the programming area and fill in the data.

It is highly recommended to limit the amount of communications on the network and to limit the number of communication requests to devices. This can be easily accomplished by using cyclical programs with the Update Rate set to as high a number as practical and by not making repeated requests to a device, until it is ready and able to execute the request. For example, repeatedly sending a Index Initiate request to a drive is not recommended, instead test to see when the drive is idle, then send the request (see program example #1).

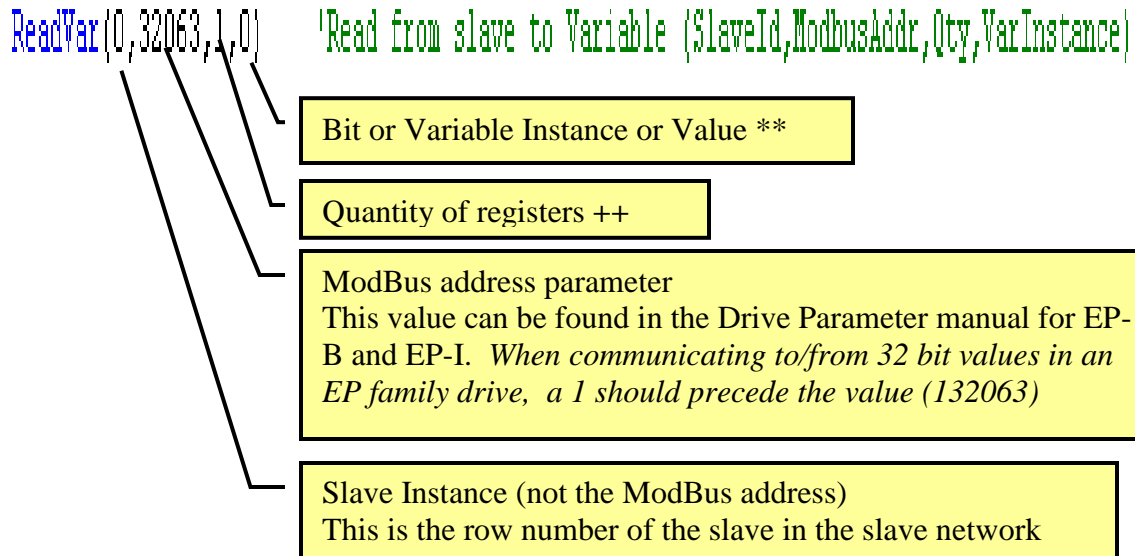


The screenshot displays the PowerTools Pro software interface. On the left, a tree view shows the project structure, including 'Axis 1', 'Status', 'Graph', 'Setup', 'Devices / Vars', 'I/O Setup', 'Motion', 'Programs', 'Cyclic Programs', 'P0', 'Programs', 'Real Time Programs', and 'Network'. The 'Modbus Slave' tree is highlighted with a red box, showing options: 'ReadBit', 'ReadVar', 'WriteBit', 'WriteVar', and 'WriteValue'. A red arrow points from 'ReadVar' in the tree to the 'ReadVar' function block in the code editor. The code editor shows a 'Real Time Cyclic Program' with an 'Update Rate' of 100 times 1600us and 'Utilize' of 90%. The code includes a 'Case 3:' block with 'ReadVar(0,32063,1,0)' and 'WriteBit(0,1103,1,1)', followed by an 'if Var.Var0 = 2 then' block with 'Bit.B1 = 0' and 'Var.MachState = 1'. The code also includes a 'LAB #2' section with 'Read EPI input and turn on the same output' and 'read EP-I's Inputs'.

Syntax

There are 5 ModBus master commands
ReadVar, WriteVar, WriteVal, ReadBit and WriteBit

The syntax for all the commands is similar, differences are explained below



For Reads: the value(s) read from the slave device will be placed in the bit or variable instance registers, with this instance being the starting register

For Writes: the values contained in the bit or variable instance register(s) will be written to the slave device, with this instance being the starting register

For **ReadVar** or **WriteVar** this value is the Variable instance (0 implies Var.Var0, a quantity of 3 with var instance 4, would use registers var.var4, var.var5 and var.var6)

For **ReadBit** or **WriteBit** this value is the Bit instance (0 implies Bit.B0, a quantity of 3 with bit instance 4, would use bit bit.b4, bit.b5 and bit.b6)

For **WriteValue:** the last parameter is a constant (not a variable instance) and this constant will be written to the slave device

++

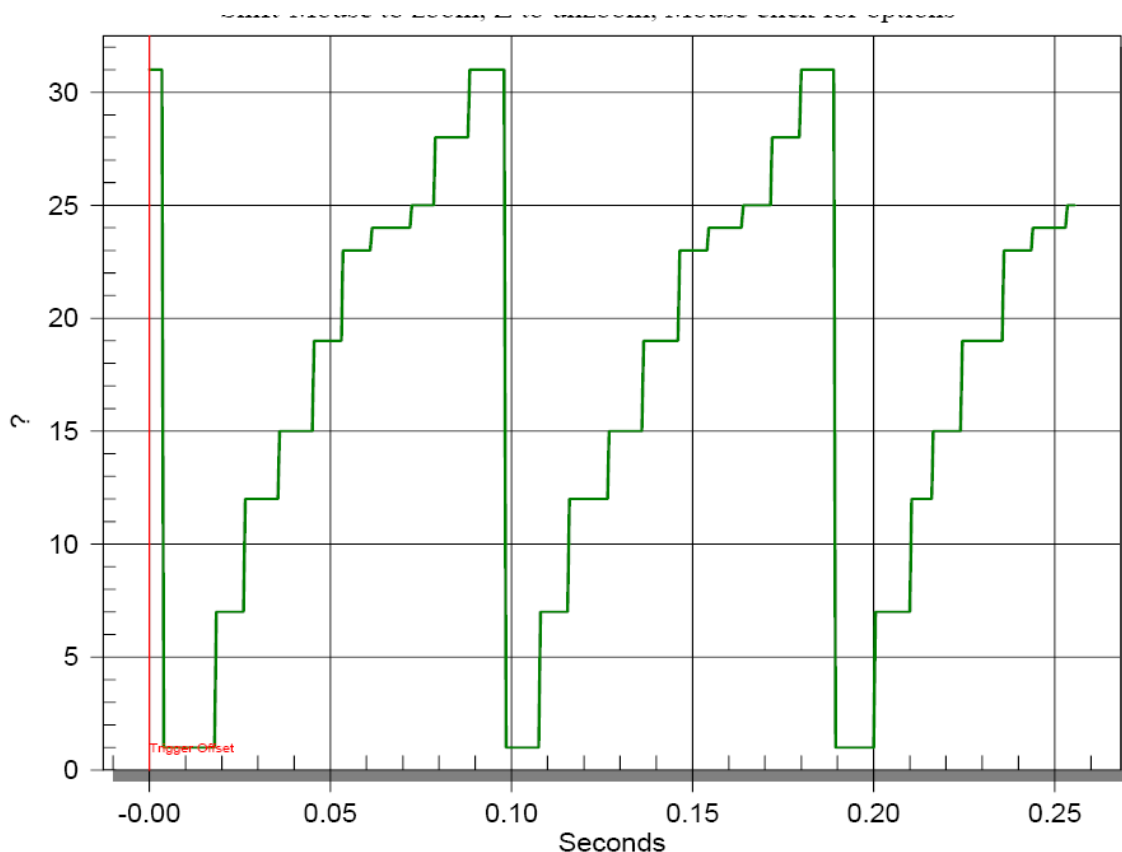
Reading or Writing multiple register is much more efficient (faster execution time) than performing multiple Reads or Write on a single register.

Message Performance

Message throughput can depend upon many factors including hardware (routers, switches, hubs), network size and configuration, slave response time, etc.

Typical message time is 3-10ms per message and its corresponding reply. A program line that issues the message will wait on that line until the message reply is received or until the message Response Gap is exceeded. If the message Response Gap is exceeded the Slave Error Count is incremented.

Below is a graph showing a single EP-P master drive communicating with 32 slaves on Modbus/TCP. All 32 slaves were individually sent a message from the master and returned a reply message. Total turn around time for 32 consecutive messages was approximately 80 milliseconds.



Program Example 1

This program uses a 'state machine' method of controlling the program.

- Case 1 checks for the EP-I drive to be idle, when idle it sets the state machine to Case2
- Case 2 issues the Index command to the EP-I drive, then sets the state machine to Case3
- Case 3 check for the index to be active, then resets the index command and sets the state machine to Case1

***** EXAMPLE 1 *****

Issue an index to the EP-I when it's idle

Do While True

Switch Var.MachState

Case 1:

'Check for Indexer drive's motion state

ReadVar(0,32063,1,0) 'Read from slave to Variable (SlaveId,ModbusAddr,Qty,VarInstance)

'after the Read is complete var.var0 will contain the value of the indexer's motion state

if Var.Var0 = 0 then ' if Indexer state = 0 (drive is idle)

Var.MachState = 2

EndIf

break

Case 2:

Bit.B1 = 1

'write to the Indexer's motion command register to initiate index 0

WriteBit(0,1103,1,1) 'Write slave with bit (SlaveId,ModbusAddr,Qty,BitInstance)

Var.MachState = 3

break

Case 3:

ReadVar(0,32063,1,0) 'Read from slave to Variable (SlaveId,ModbusAddr,Qty,VarInstance)

if Var.Var0 = 2 then 'If indexing active, then reset the Initiate command

Bit.B1 = 0

WriteBit(0,1103,1,1) 'Write slave with bit (SlaveId,ModbusAddr,Qty,BitInstance)

Var.MachState = 1 ' reset state to check for drive idle (index complete)

EndIf

break

EndSwitch

Loop



PowerTools Pro Application Note

PTAN #3, rev. 2, 1/14/2010
Applicable Products: Epsilon EP-P, FM4E

Program Example 2

This program reads the EP-I's input switches and sets the EP-I's outputs to match the input state.

```
***** EXAMPLE #2 *****
'Read EP-I input and turn on the outputs
*****

'read EP-I's Inputs
ReadVar(0,30101,1,2)      'Read from slave to Variable (SlaveId,ModbusAddr,Qty,VarInstance)

'set Output force
WriteValue(0,40104,1,63)  'Write slave with Value(SlaveId,ModbusAddr,Qty,Value)

'turn on output(s)
If var.var3 > 0 then      'don't turn on Outputs, if the var3 <0
    WriteVar(0,40103,1,3)  'Write slave with Variable(SlaveId,ModbusAddr,Qty,VarInstance)
Else
    WriteValue(0,40103,1,0) 'Write slave with Value(SlaveId,ModbusAddr,Qty,Value)
EndIf
```

Program Example 3

```
***** EXAMPLE #3 *****
Update EP-I's Index 0 velocity and distance parameters
*****

'Send Index0's Distance and Velocity values in variable 0 and 1
'*****
var.var0 = 10000
var.var1 = 5000

'send both variables using 2 as the Qty argument
WriteVar(0,143002,2,0) 'Write slave with Variable(SlaveInstance#,

'Initiate the Index using EP-I Motion Command Array:
,*****
' initialize the B1 Bit instance to 1
Bit.B1 = 1
WriteBit(0,1103,1,1)    'Write slave with bit (SlaveInstance#,ModbusAddr,Qty,BitInstance)

Wait For Time .01 'seconds

'reset the B1 Bit instance
Bit.B1 = 0
WriteBit(0,1103,1,1)    'Write slave with bit (SlaveInstance#,ModbusAddr,Qty,BitInstance)
```

Notice the '1' is needed to instruct the EP-I that 2 consecutive 16 bit registers should be treated as a single 32 bit word (e.g. 143002)

This single line sends four 16 bit registers, the EP-I reads them as 2 32 bit words