3 Configuration as a Modbus Slave

3.1 Overview

When configuring the module as a Slave, you will be providing whoever is programming the Master side of the communications with a Modbus Memory Map.

Note: If you are using the Sample Ladder Logic, the transfer of data is already done.

Information that is to be read by the Modbus Master device will be placed in the **MCM.DATA.WRITEDATA** array as this will be pushed out to the module so that values from the ControlLogix processor can be read by the Modbus Master. Information that must be written to the ControlLogix processor from the Modbus Master device will be placed into the **MCM.DATA.READDATA** array.

To configure module as a Modbus Slave you must determine how much data you must transfer to and from the module, to the Modbus Master.

The sample ladder file is configured to transfer 600 16-bit registers in each direction. If more than that is required, please see Adjust the Input and Output Array Sizes (Optional) (page 25).

Find out if the Master can read from one Modbus address and write to another Modbus address, or, if the Master must use the same address to read and write data points.

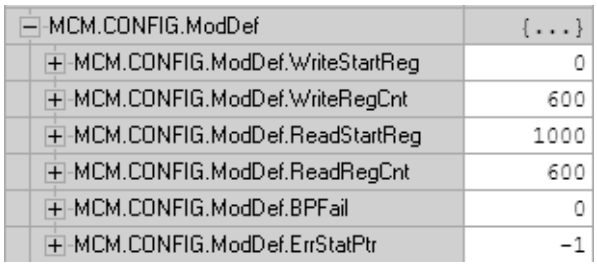
If a Modbus command must bypass the read and write areas of the slave's memory area and send Modbus commands directly to another device on the Modbus network (for example, to a PLC), you must use Pass-Through mode. This allows the **MCM.DATA.WRITEDATA** array to be used for all data transfer to the Master. Because the data transfer of the MVI56E-MCM module cannot be bidirectional, when the Master issues a Modbus Write command in Pass-Through mode, the MVI56E-MCM module builds a special block of information. This block is then parsed by the ladder logic, and the value written from the Modbus Master is then updated in the **MCM.DATA.WRITEDATA** array.

Note: You should only use Pass-Through mode when there is no other option, as there is a drawback to this mode that is not present in the standard mode.
Because the module must wait for the ladder logic to confirm receiving the new data from the Master, if the Master issues consecutive write commands, the module cannot process the second write command until it has finished with the first command. This will cause the module to respond with an error code of 6 (module busy) on the Modbus network.

3.2 ModDef Settings

To configure Modbus Slave mode, use the **MCM.CONFIG.MODDEF** settings.

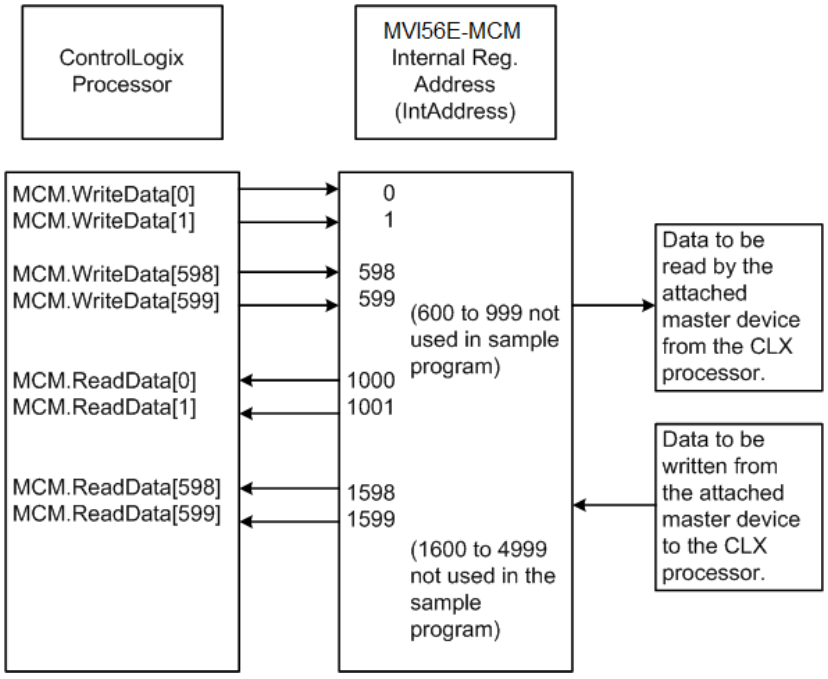
This section specifies which of the MVI56E-MCM module's 10,000 registers of memory to send from the ControlLogix processor to the MVI56E-MCM module (WriteData) and which registers to send from the MVI56E-MCM module to the ControlLogix processor (ReadData).



The **WRITESTARTREG** determines the starting register location for **WRITEDATA [0 TO 599]** and the **WRITEREGCNT** determines how many of the 10,000 registers to use for information to be written out to the module. The sample ladder file will configure 600 registers for Write Data, labeled **MCM.WRITEDATA[0 TO 599].**

Value	Description
WriteStartReg	Determines where in the 10,000 register module memory to place the data obtained from the ControlLogix processor from the WriteData tags.
WriteRegCnt	Sets how many registers of data the MVI56E-MCM module will request from the ControlLogix processor.
ReadStartReg	Determines where in the 10,000 register module memory to begin obtaining data to present to the ControlLogix processor in the ReadData tags.
ReadRegCnt	Sets how many registers of data the MVI56E-MCM module will send to the ControlLogix processor.
BPFail	Sets the consecutive number of backplane failures that will cause the module to stop communications on the Modbus network.
ErrStatPtr	This parameter places the STATUS data into the database of the module. This information can be read be the Modbus Master to know the status of the module.

With the sample configuration, the following is the layout of the tags and addressing.



The sample configuration values configure the module database for **WRITEDATA[0 TO 599]** to be stored in the module memory at register 0 to 599, and **READDATA[0 TO 599]** to be stored in the module memory at registers 1000 to 1599 as shown above.

3.2.1 Modbus Memory Map

Based on the configuration described above, below is the default Modbus address for the module. Each register within the module can be accessed as a 0xxx bit address, 1xxxx bit address, 3xxxx register address, or 4xxxx register address.

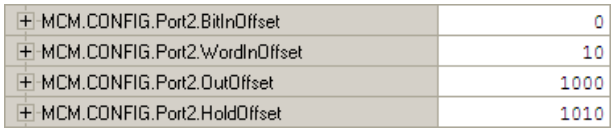
MVI Address	0xxx	1xxxx	3xxxx	4xxxx	Tag Address
0	0001 to 0016	10001 to 10016	30001	40001	WriteData[0]
1	0017 to 0032	10017 to 10032	30002	40002	WriteData[1]
2	0033 to 0048	10033 to 10048	30003	40003	WriteData[2]
3	0049 to 0064	10049 to 10064	30004	40004	WriteData[3]
4	0065 to 0080	10065 to 10080	30005	40005	WriteData[4]
5	0081 to 0096	10081 to 10096	30006	40006	WriteData[5]
6	0097 to 0112	10097 to 10112	30007	40007	WriteData[6]
7	0113 to 0128	10113 to 10128	30008	40008	WriteData[7]
8	0129 to 0144	10129 to 10144	30009	40009	WriteData[8]
9	0145 to 0160	10145 to 10160	30010	40010	WriteData[9]
10	0161 to 0176	10161 to 10176	30011	40011	WriteData[10]
50	0801 to 0816	10801 to 10816	30051	40051	WriteData[50]
100	1601 to 1616	11601 to 11616	30101	40101	WriteData[100]
200	3201 to 3216	13201 to 13216	30201	40201	WriteData[200]
500	8001 to 8016	18001 to 18016	30501	40501	WriteData[500]
598	9569 to 9584	19569 to 19584	30599	40599	WriteData[598]
599	9585 to 9600	19585 to 19600	30600	40600	WriteData[599]
600 to 999	N/A	N/A	N/A	N/A	Reserved
1000			31001*	41001	ReadData[0]
1001			31002*	41002	ReadData[1]
1002			31003*	41003	ReadData[2]
1003			31004*	41004	ReadData[3]
1004			31005*	41005	ReadData[4]
1005			31006*	41006	ReadData[5]
1006			31007*	41007	ReadData[6]
1007			31008*	41008	ReadData[7]
1008			31009*	41009	ReadData[8]
1009			31010*	41010	ReadData[9]
1010			31011*	41011	ReadData[10]
1050			31051*	41051	ReadData[50]
1100			31101*	41101	ReadData[100]
1200			31201*	41201	ReadData[200]
1500			31501*	41501	ReadData[500]
1598			31599*	41599	ReadData[598]
1599			31600*	41600	ReadData[599]

(*) Values listed in the **READDATA** array for 31001 to 31600.

Although these are valid addresses, they will not work in the application. The Master must issue a Write command to the addresses that correspond to the **READDATA** array. For Modbus addresses 3xxxx these are considered Input registers, and a Modbus Master does not have a function code for this type of data.

3.2.2 Customizing the Memory Map

In some cases, the above memory map will not work for the application. Sometimes a Master must read bits starting at address 0001, and also read a register starting at 40001. With the memory map in this Modbus Memory Map (page 63), this is not possible, as **WRITEDATA[0]** is seen as both 0001 to 0016, and 40001. To accommodate this, you can customize the starting location within the module for each device using the parameters shown below.



Parameter	Value	Description
BitInOffset	0	Defines the starting address within the module for 1xxxx Modbus addressing. A value of 0 sets 10001 to 10016 as address 0 in the MVI56E-MCM module.
WordInOffset	10	Defines the starting address within the module memory for 3xxxx registers.
OutOffset	1000	Defines the starting address within the module for 0xxx coils.
HoldOffset	1010	Defines the starting address within the module for 4xxxx addressing.

Based on the configuration described above for the ModDef section of the module and the values specified for the offset parameters, below is the Modbus addressing map for the module.

MVI Address	0xxx	1xxxx	3xxxx	4xxxx	Tag Address
0		10001 to 10016			WriteData[0]
1		10017 to 10032			WriteData[1]
9		10145 to 10160			WriteData[9]
10		10161 to 10176	30001		WriteData[10]
11		10177 to 10192	30002		WriteData[11]
100		11601 to 11616	30091		WriteData[100]
200		13201 to 13216	30191		WriteData[200]
500		18001 to 18016	30491		WriteData[500]
598		19569 to 19584	30489		WriteData[598]
599		19585 to 19600	30490		WriteData[599]
600 to 999	N/A	N/A	N/A	N/A	Reserved
1000	0001 to 0016				ReadData[0]
1001	0017 to 0032				ReadData[1]
1009	0145 to 0160				ReadData[9]
1010	0161 to 0176			40001	ReadData[10]
1011	0177 to 0192			40002	ReadData[11]
1050	0801 to 0816			40041	ReadData[50]
1100	1601 to 1616			40091	ReadData[100]

MVI Address	0xxx	1xxxx	3xxxx	4xxxx	Tag Address
1200	3201 to 3216			40191	ReadData[200]
1500	8001 to 8016			40491	ReadData[500]
1598	9569 to 9584			40589	ReadData[598]
1599	9585 to 9600			40590	ReadData[599]

With the offset parameters listed above, the Modbus Master could read from coils 10001 to 10176 using the tags **MCM.DATA.WRITEDATA[0] TO [9].** The Master could also read from address 30001 to 30490, and the data contained in those Modbus addresses would come from the tags **MCM.DATA.WRITEDATA[10] TO [499]** within the ControlLogix program.

The Master could then write to coils addressing 0001 to 0160 and this data would reside within the ControlLogix program in tags **MCM.DATA.READDATA[0] TO [9].** The Master could then write to registers using Modbus addresses 40001 to 40590, and this information would reside in addresses **MCM.DATA.READDATA[10] TO [599].**

Note: The offset parameter only sets the starting location for the data. As shown above, if the Master issues a Write command to address 40001, the data will go into the ControlLogix processor at address **MCM.DATA.READDATA[10].**

Likewise, a Write To bit address 0161 will also change to address **MCM.DATA.READDATA[10].0** within the program. Be careful not to overlap your data. You may want leave additional registers/bits unused to allow for future expansion in the program.

3.3 Slave Configuration

Any parameters not mentioned in this section are not used when the module is configured as a Modbus Master.

Value	Description
Enabled	1= enable port, 0 = disable port
Type	**1= Modbus Slave Port** The module also supports a variety of Pass-Through modes. See Pass-Through Blocks (page 132) for more information.
FloatFlag	As a Slave, emulates Enron/Daniel style floats. See Floating-Point Data Handling (Modbus Slave) (page 66) for more information.
FloatStart	Register offset in message for floating data point. See Floating-Point Data Handling (Modbus Slave) (page 66) for more information.
Protocol	0 = Modbus RTU mode, 1 = Modbus ASCII mode
Baudrate	Sets the baud rate for the port. Valid values for this field are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 384 (for 38,400 baud), 576 (for 57,600 baud) and 115 or 1152 (for 115,200 baud)
Parity	0 = None, 1 = Odd, 2 = Even
DataBits	8 = Modbus RTU mode, 8 or 7 = Modbus ASCII mode
StopBits	Valid values are 1 or 2
SlaveID	Valid values are 1 to 247

3.4 Floating-Point Data Handling (Modbus Slave)

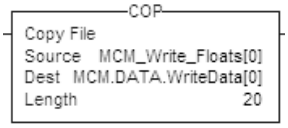
In most applications, the use of floating-point data requires no special handling.

1 Copy the data to and from the MVI56E-MCM module with a tag configured as a data type REAL in the ControlLogix processor.

Each floating-point value will occupy 2 registers on the Modbus network.
Some Master devices use Enron or Daniel Float data. These types of floats require one Modbus register for each float in the module memory. If your Master requires this addressing, refer to the following section.
For standard floating-point data handling, the following is an example of copying 10 floats to the module.

2 First, configure a tag within the ControlLogix processor.

3 Then configure a COP statement within the main routine to copy this tag to the module's **MCM.DATA.WRITEDATA** array.

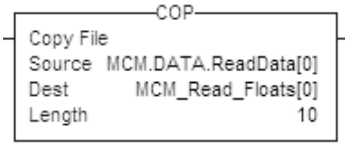


The length of the copy statement is determined by the Dest file size. To copy 10 floats from the MCM_Write_Floats array to the **MCM.DATA.WRITEDATA** array, the length of the COP statement must be set to a value of 20.

To copy data from the MVI56E-MCM module to a floating-point tag within the ControlLogix processor

1 Configure a tag within the ControlLogix processor as shown.

2 Then configure the COP statement to move data from the **MCM.DATA.READDATA** array, and over to the new tag **MCM_READ_FLOATS** tag as shown here.



Once again, the COP statement will take as many of the Source elements required to fill the Dest tag for the length specified. Therefore, the COP statement will take **MCM.DATA.READDATA[0] TO [19]** to fill the **MCM_READ_FLOATS[0] TO [9].**

3.4.1 Enron/Daniel Float Configuration

Sometimes it is necessary for the module to emulate Enron or Daniel floating-point addressing.

Copying the data to the **MCM.DATA.WRITEDATA** array and from the **MCM.DATA.READDATA** array is the same as described in the section above. The main difference is the addressing of the module.

For example, an Enron Float device is required to access address 47001 for floating-point data, and each Modbus register would emulate a single float value (does not require 2 Modbus addresses for 1 float value).

A Master device requiring this type of addressing, would require that for every count of 1, the MVI56E-MCM module responds to the request message with 4 bytes (one 32-bit REAL) value.

To emulate this addressing, the module has the parameters **MCM.CONFIG.PORTX.FLOATFLAG, FLOATSTART,** and **FLOATOFFSET.**

Value	Description
FloatFlag	Tells the module to use the FloatStart and FloatOffset parameters listed below
FloatStart	Determines what starting address on the Modbus network to treat as floating-point data. A value of 7000 will signal the module that address 47001 on the Modbus network is the starting location for Modbus floating-point data. Every address will occupy 2 registers within the modules database
FloatOffset	Determines the address within the module to which to associate the data from the FloatStart section.

Example configuration:

MCM.CONFIG.Port2.FloatFlag	1
MCM.CONFIG.Port2.FloatStart	7000
MCM.CONFIG.Port2.FloatOffset	100

With the above configuration, this would be the addressing for the module.

Module Address	Modbus Address	Tag Address
100	47001	MCM.DATA.WriteData[100]
102	47002	MCM.DATA.WriteData[102]
104	47003	MCM.DATA.WriteData[104]
110	47006	MCM.DATA.WriteData[110]
120	47011	MCM.DATA.WriteData[120]
200	47051	MCM.DATA.WriteData[200]
300	47101	MCM.DATA.WriteData[300]
500	47201	MCM.DATA.WriteData[500]