MITSUBISHI ELECTRIC

Mitsubishi Motion AOI Solutions

J4TM AOI User Manual

Version 1.000

R72-164-SLSASG-001-*

This document applies to the Mitsubishi Electric Corporation product components and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

MITSUBISHI ELECTRIC CORPORATION PROVIDES THIS DOCUMENT "AS IS," WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

This document could contain technical inaccuracies or typographical errors. Changes are made periodically to the information herein. Mitsubishi Electric Corporation may make improvements and changes at any time to the product(s) and/or program(s) described in this document.

Table of Content

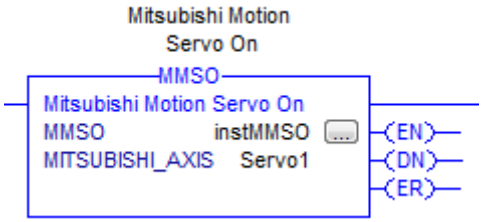
1. State AOIs

- **True/False** means the AOI can start with either a true rung or a one-shot. Once started, the logic will continue whether the rung is true or false.
- **Rising Edge** means it will execute on the rising edge and then lock out. There is no continuing logic when the rung is false
- **Latched True** means the AOI only executes when the rung is true.

MMSO - Mitsubishi Motion Axis Servo On

Use the Motion Servo On (MMSO) instruction to enable the servo amplifier

- MMSO is **Rising Edge**.
- It is enable the drive when the rung makes a false-to-true transition

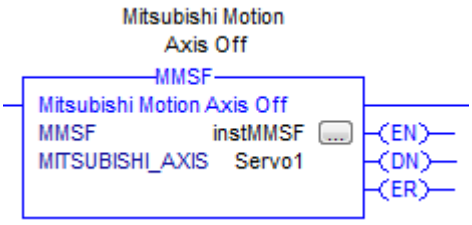


See Servo States

MMSF - Mitsubishi Motion Axis Servo Off

Use the Motion Servo Off (MMSF) instruction to deactivate the drive output for the specified axis .

- MMSO is **Rising Edge**.

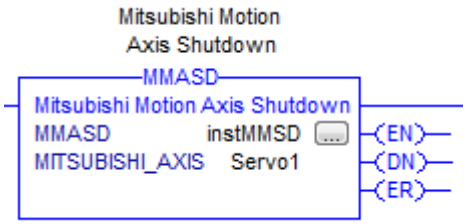


MMASD - Mitsubishi Motion Axis Shutdown

Use the Motion Axis Shutdown (MMASD) instruction to force a specified axis into the Shutdown state. The Shutdown state of an axis is the condition where the drive output is disabled. The axis remains in the Shutdown state until either an Axis or Group Shutdown Reset is executed.

- MMASD is **Rising Edge**

See Servo States

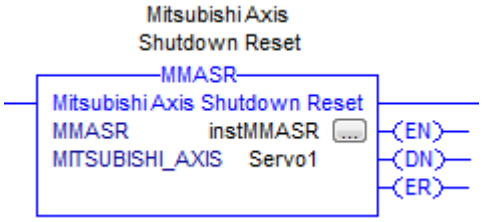


MMASR - Mitsubishi Motion Axis Shutdown Reset

Use the Motion Axis Shutdown (MMASR) instruction to transition an axis from an existing Shutdown state to an Axis Ready state. All faults associated with the specified axis are automatically cleared.

- MMASR is **Rising Edge**

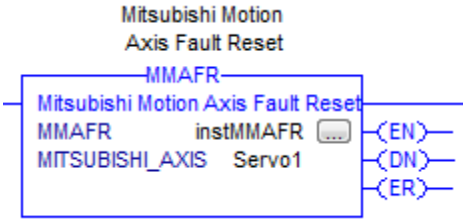
See Servo States



MMAFR - Mitsubishi Motion Axis Fault Reset

Use the Motion Axis Fault Reset (MMAFR) instruction to clear all motion Errors and waring for an axis. This is the only method for clearing axis motion errors.

- MMAFR is **Rising Edge**



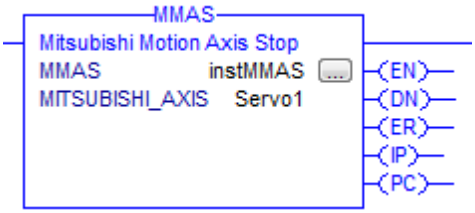
2. Motion AOIs

- **True/False** means the AOI can start with either a true rung or a one-shot. Once started, the logic will continue whether the rung is true or false.
- **Rising Edge** means it will execute on the rising edge and then lock out. There is no continuing logic when the rung is false
- **Latched True** means the AOI only executes when the rung is true.

MMAS - Mitsubishi Motion Axis Stop

Use the Motion Axis Stop (MAMS) instruction to stop a specific motion process on an axis or to stop the axis completely.

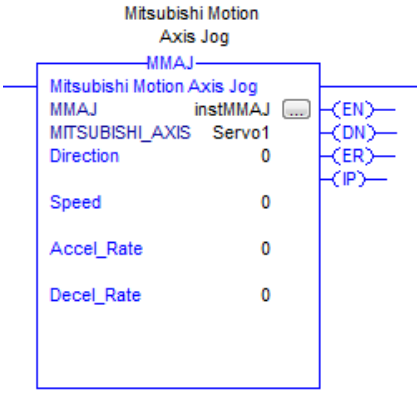
- MMAS (Motion Axis Stop) is **True/False**..
- If you use it with a one-shot, it will stop motion but any motion command can start the servo again.
- When latched and active, the servo does not move and ignores all motion commands,
- If you try to move with MMAS active - Error = 1017.
- MMAS stops motion and clears the buffer.
    - If you want to be able to resume after stopping an MMAM move, use absolute moves.
    - If you want to be able to resume after stopping an MMAPT move, use the Pause input to the MMAPT AOI.



MMAJ - Mitsubishi Motion Axis Jog

Use the Motion Axis Jog (MMAJ) instruction to move an axis at a constant speed until you tell it to stop.

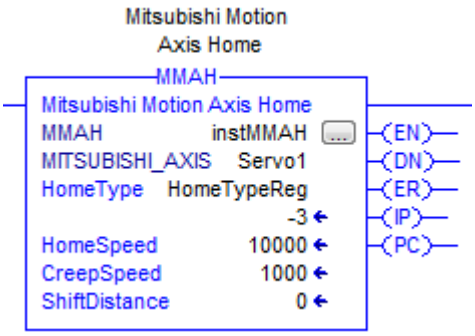
- MMAJ (Motion Axis Jog) is **True/False**.
- Direction is 0 or 1.
- Speed (unit rpm x0.01) can be negative (Direction=0 & speed=5000 is equivalent to Direction=1 & speed= -5000)
- Multiple Moves - During a jog, a second jog command will merge immediately.
- Accel_Rate is value in ms to reach the rated speed
- Decel_Rate is value in ms from rated speed to zero speed.
- To stop, use MMAS Mitsubishi Motion Axis Stop.



MMAH – Mitsubishi Motion Axis Home

Use the Motion Axis Home (MMAH) instruction to home an axis .

- MMAH - Homing is **True/False**
- Direction is based on Home type



It has the **MMWriteR** AOI embedded inside it for Parameter Mapping – HomeType, HomeSpeed and CreepSpeed.

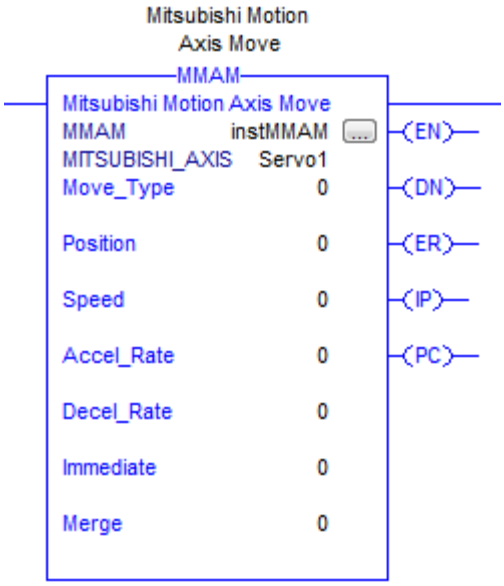
Parameter	Description	Comment
HomeType	Used in all homing routines	See details in user manual
HomeSpeed	Used in all homing routines	
CreepSpeed	Used in all homing routines	
ShiftDistance	After homing, the servo moves to this shift position and zeros the position register	Polarity has no effect. Absolute value is loaded into J4. (Not implermented yet)

For detailed description of all homing commands, **MR-J4-TM SERVO AMPLIFIER INSTRUCTION MANUAL (EtherNet/IP) SH030226**

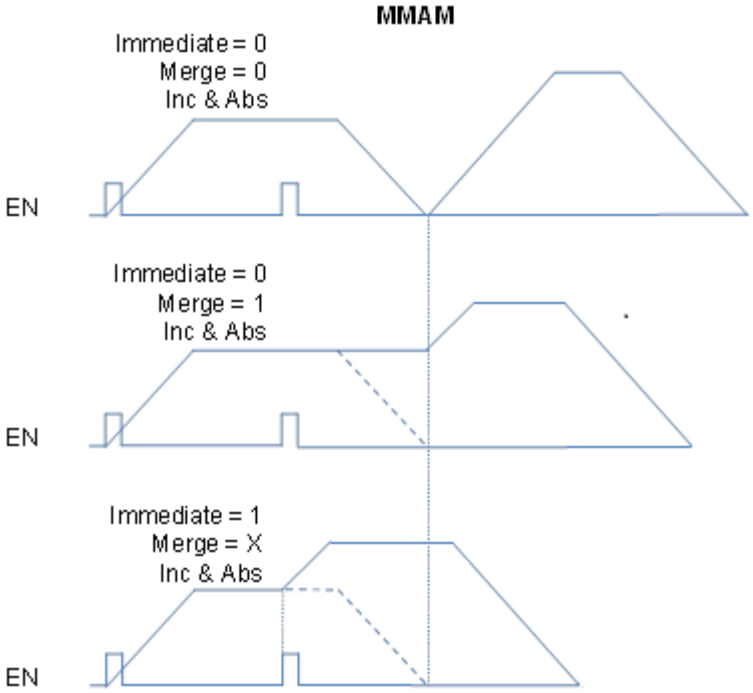
MMAM – Mitsubishi Motion Axis Move

Use the Motion Axis Move (MMAM) instruction to move an axis to a specified position.

- MMAM (Motion Axis Move) is **True/False**.
- Move Type: 0→ Absolute and 1→Incremental
- Position(pulse): Absolute system→Position to move and Incremental system → Distance to move (polarity determines direction)
- Position command range -999999 to 999999
- Speed (rpmx0.01) polarity has no effect.
- Multiple Moves using Merge and Immediate
- Accel_Rate is value in ms to reach the rated speed
- Decel_Rate is value in ms from rated speed to zero speed.



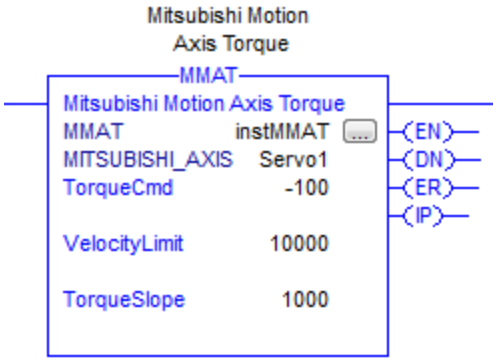
You can merge a second move command using Immediate and Merge inputs. However, note the distance of the second command is added to the distance of the first command.



MMAT - Mitsubishi Motion Axis Torque

Use the Motion Axis Torque (MMAT) instruction to move an axis at a constant torque until you tell it to stop.

- MMAT (Motion Axis Torque) is **True/False**.
- ***Velocitylimit*** *cannot be negative.* The servo uses the torque polarity for direction. (If you could change the speed polarity, that new direction will ***not*** have torque limited, it will be FULL torque and will take off.)
- **Direction is determined by TorqueCmd polarity.**
- **TorqueCmd** is Rated Torque/10
- **TorqueSlope** is Rated Torque/10/sec
- To stop, use MMAS Mitsubishi Motion Axis Stop.

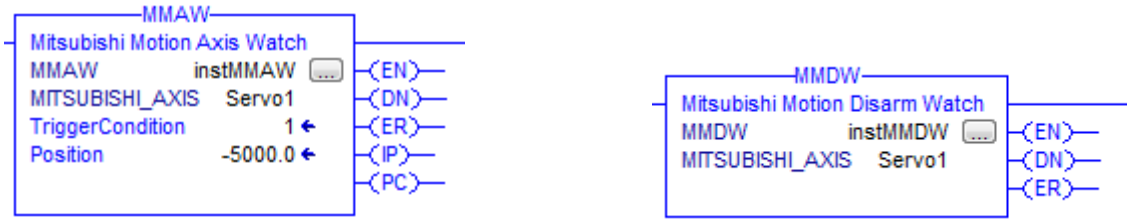


3. Event AOIs

- **True/False** means the AOI can start with either a true rung or a one-shot. Once started, the logic will continue whether the rung is true or false.
- **Rising Edge** means it will execute on the rising edge and then lock out. There is no continuing logic when the rung is false
- **Latched True** means the AOI only executes when the rung is true.

MMAW - Mitsubishi Motion Arm Watch & MMDW - Mitsubishi Motion Disarm Watch

- MMAW is **True/False**
- MMDW is **Rising Edge**



- **TriggerCondition** –
  - 0 = Forward - the servo module looks for the actual position to change from less than the watch position to greater than the watch position.
  - 1 = Reverse – the servo module looks for the actual position to change from greater than the watch position to less than the watch position.
- **Position** that triggers PC output
- **PC** - will go high if servo moves in correct direction from outside to inside the postion.

If **TriggerCondition** is 0 and actual position is greater than **Postion**, **PC** is set. It is reset by another rising edge trigger to MMAW or MMDW.
If **TriggerCondition** is 1 and actual position is less than **Position**, **PC** is set. It is reset by another rising edge trigger to MMAW or MMDW.

In the MITSUBISHI_SERVO UDT:

- **WatchEventArmedStatus** is a Boolean set if the Watch logic is active. Reset when event occurs or MMDW.
- **WatchEventStatus** is a Boolean flag set when event occurs. It is reset by another rising edge trigger to MMAW or MMDW.

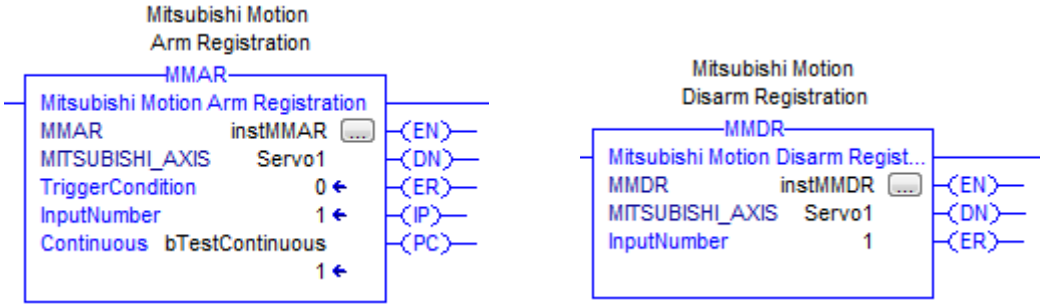
To reset Process Complete (PC) you have to re-trigger MMAW. (MMDW has no effect).

MMAR - Mitsubishi Motion Arm Registration & MMDR - Mitsubishi Motion Disarm Registration

- MMAR is **True/False**
- MMDR is **Rising Edge**

When the Touch Probe input is asserted, the actual position at that moment is captured and stored in Registration registers.

The Touch Probe inputs are in CN3 pins 1 and 10.
The captured positions are in the AOI instance – x.**A_TouchProbeValue** and x.**B_TouchProbeValue.**



Input	Range	Description
Trigger Condition	= 0 trigger on positive edge = 1 trigger on negative edge	
InputNumber	0, 1 or 2	Which probe to use. 0 = disable that channel
Continuous	= 0 Once captured, AOI must be toggled to re-arm = 1 AOI will automatically re-arm after every probe input assertion.	If AOI is not **Continuous**, PC (Process Complete) will come on with the first probe input assertion. PC will stay on and AOI will not re-arm until AOI enable is toggled. If AOI is set **Continuous**, PC is a one-shot for each probe input assertion.

RegEventArmedStatus in the MITSUBISHI_SERVO UDT is set when the AOI is enabled in Logic. It is reset:

- after a capture if not in Continuous mode.
- with MMDR
- with an error.

RegEventStatus in SDT goes low when the AOI is enabled. It goes high when a capture occurs. It is reset:

- RegEventArmedStatus is toggled

Output	Continuous	Description
A_PC B_PC (these are not visible – use dot operator on instance)	= 0 Latched	Set when position is captured. Stays set until the AOI is toggled or initialized
	= 1 Continuous update	Pulsed output for 1 scan only when position is captured.

There are two probe inputs 1 & 2. These are independent of the two Channels A & B. You assign the inputs to any channel.

Example 1

Channel A is Touch Probe input 2, falling edge and continuous.
Channel B is Touch Probe input 2, rising edge and continuous.

Example 2

Channel A is Touch Probe input 1, rising edge and continuous.
Channel B is Touch Probe input 2, rising edge and continuous.

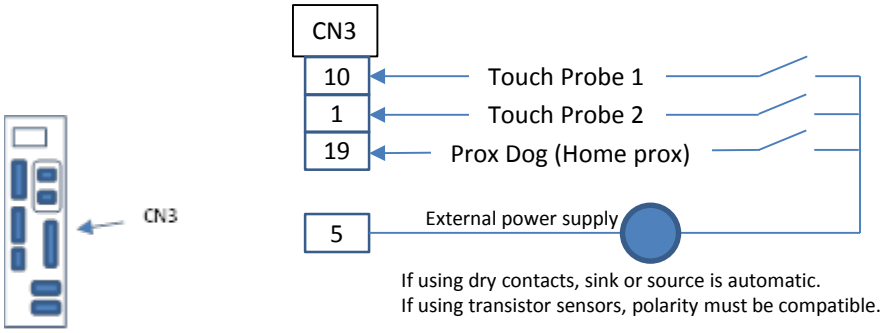
Example 3

Channel A is Touch Probe disabled (= 0).
Channel B is Touch Probe input 2, rising edge and latched.

Example 4

Channel A is Touch Probe input 1, rising edge and ***latched***.
Channel B is Touch Probe input 1, falling edge and ***continuous*** <<< NOT ALLOWED.
You cannot have both latched and continuous on the same input.

The **MMAR** AOI will not have window registration. The registration function in the servo works all the time with no limitations. Rather than embed windowing code inside the AOI, it is left to the user to write their own windowing logic if desired.



4. Configuration AOIs

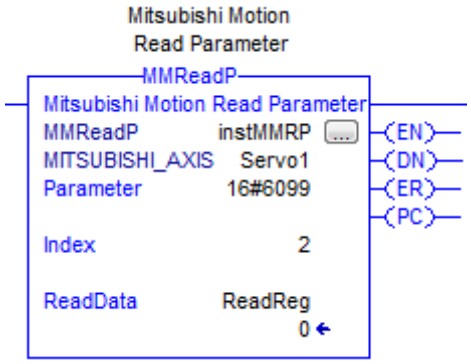
- **True/False** means the AOI can start with either a true rung or a one-shot. Once started, the logic will continue whether the rung is true or false.
- **Rising Edge** means it will execute on the rising edge and then lock out. There is no continuing logic when the rung is false
- **Latched True** means the AOI only executes when the rung is true.

MMReadP - Mitsubishi Motion Read Parameter

- MMReadP is **True/False**

This is a single read AOI that uses CANOpen objects.

This example reads CANOpen register 6099h sub index 2 which is homing creep speed.



MMWriteR - Mitsubishi Motion Write RAM Parameter

- MMWriteR is **True/False**

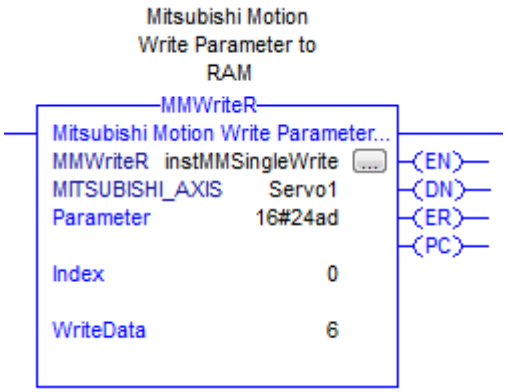
This is an auxiliary AOI for writing parameters with the J4 Servo. MMWriteR writes to the RAM memory. When you cycle power, these values are not retained.

To permanently load into memory EEPROM, see MMWriteE.

You can program a loop to do multiple writes. See Write Loop examples in Tables & Lists

This is a single write AOI that uses CANOpen addresses.

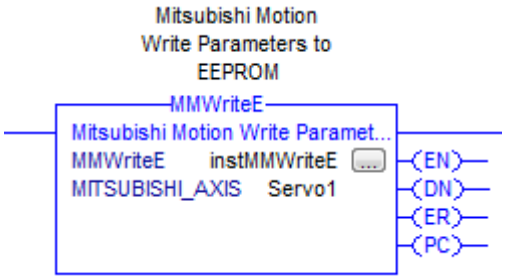
This example write parameter (PT45 ,object number16#24AD) Home type to 6. Subindex = 0.



MMWriteE - Mitsubishi Motion Write EEPROM Parameters

- MMWriteE is **True/False**
- You do not enter parameters in this AOI. It will save what is already in RAM at the time of execution.

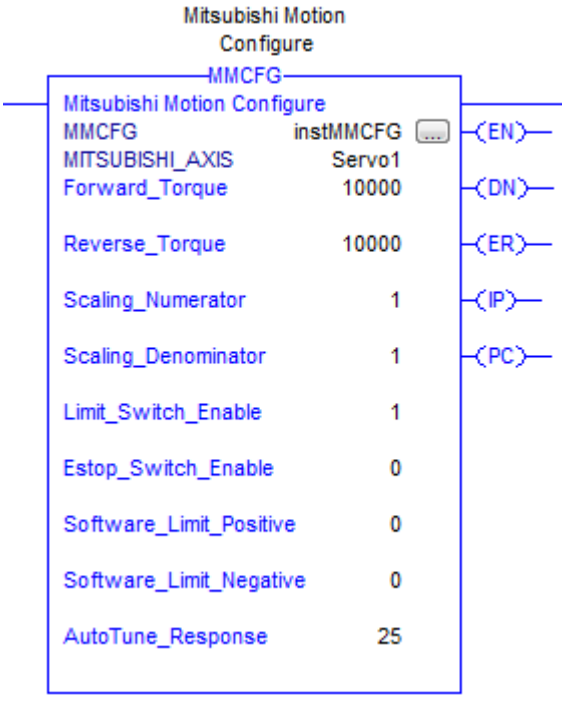
MMWriteE writes all the RAM settings to EEPROM.
MMWriteR only writes to RAM – when the servo reboots, the values in the EEPROM load to servo parameters



Note: This can take up to 10 seconds - it stores all the parameters from RAM to EEPROM. Do not turn off power to recycle power for at least 10 seconds

MMCFG – Mitsubishi Motion Axis Configuration

- **MMCFG** is **Rising Edge**
- You must cycle power to the servo after executing



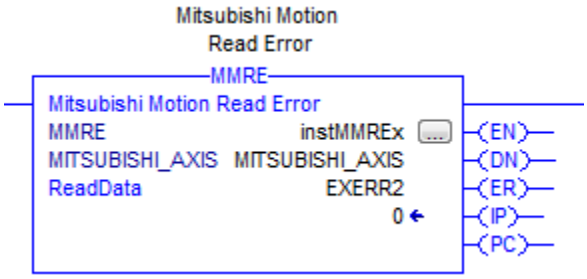
- **Torque** –
  - 0 – 10,000 (1000.0%)
- **Limit_Switch_Enable** –
  - 0 = disabled, no limit switches (0C00)
  - 1 = Positive limit switch on (0800)
  - 2 = Negative limit switch on (0400)
  - 3 = Both limit switches on (0000)
- **Estop_Switch_Enable** –
  - 0 = disabled, no Estop switch (2100)
  - 1 = Estop switch on, no deceleration (0000)
  - 2 = Estop switch on, deceleration (2000)
- **Software_Limit** –
  - Full range of DINT

MMRE - Mitsubishi Motion Read Error

- MMRE is **Rising Edge**

MMRE (Motion Read Error) - Is a stripped down AOI designed to be embedded in other AOIs (MMAH, MMAM, MMAJ, MMAT, etc). This AOI reads the servo error using Parameter Mapping. It can be used as a standalone but it does not have LO state logic, only Parameter Mapping state logic. It does not have any error checking on its own.

Most other AOIs that have MMRE embedded with a local instance tag of MMRE in their own parameter tags.
In the event of a Servo Error, MMRE will output the servo error into EXERR and the parent AOI will move Error Code 1020 into ERROR.

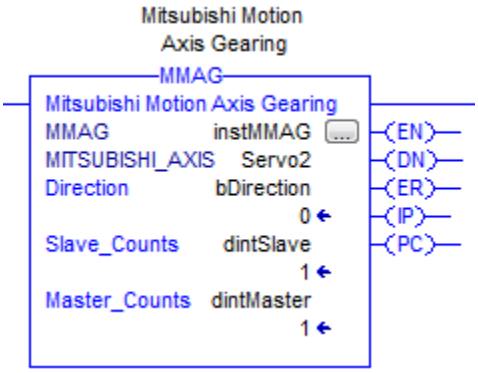


5. Gearing AOI

MMAG - Mitsubishi Motion Axis Gearing

The slave axis will follow the master axis. The master encoder counts are ported to the slave.

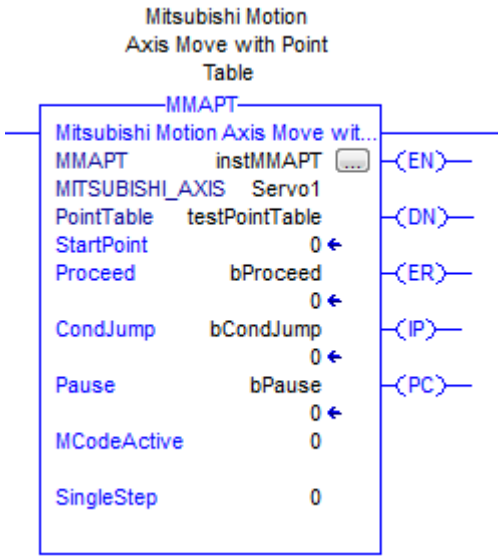
- **MMAG** is **Latched True**. When active, the slave will follow. When not active the slave drops out.
- **Direction** = 0 slave moves in same direction. **Direction** = 1 slave moves in opposite direction.
- **Slave Counts/Master Counts**



6. Point Table AOI

MMAPT - Mitsubishi Motion Point Table.

- **MMAPT** is **Latched True**



Point tables are an array of simple moves. They are not merged or blended. At this time, the point tables are set at 20. An index consists of an absolute or relative move, with a command at the end of the move (continue, timer, jump, wait, etc).

The MMAPT is not transitional, i.e. if not active, the point table stops at the end of the current move and command.

If you turn off the rung, the index will stop at the end of the index (after move and command). However if you re-activate it, the Point table resets.

- If you want to pause, use the Pause input
- If you want to single step, use the Single Step input.

Inputs

- **StartPoint** is which index to start with, typically 0.
- **Proceed** is an input used with 3 functions
    - To override **TimerOverride (4)** command. If Proceed goes high during the timer, it will stop the timer and move to the next index. If Proceed does not go high during the timer, the timer will time out normally and move to the next index.
    - To override **Wait (5)** command.
    - To override **Pause** Input. **Pause** will stop the table at any time. You resume by turning **Proceed** on (If **Pause** is not active). This needs a rising edge. If **Proceed** is already on before a **Pause**, it will need to be toggled. (An MMAS will clear the buffers, this does not)

  Proceed a rising edge trigger.

- **CondJump** is an input used for a Conditional Jump command. If active it will jump to the index loaded in Value. If not active, the next index will be executed. **CondJump** is active as long as it is true.

- If **Pause** turns on during a move, it will stop immediately. When the Pause input goes low, and **Proceed** is toggled, the command will finish and the table will continue.

  Remember that some commands are single scan operations.

- **MCodeActive** is an input to specify if the M code is loaded at the beginning (=0) or the end (=1) of the index, after both position and command. If loaded at the end, the initial index MCode value will be 0.
- **SingleStep** is an input that causes the AOI to stop at the end of each index (move and command). You press Proceed to execute the next index.
  - If SingleStep is set before executing MMAPT, you will have to toggle Proceed to execute the first index.
  - If SingleStep is set before the end of an index (move and command), the index will stop at the end.
  - If SingleStep is reset before the end of an index, the index will still stop at the end. Proceed will clear it.
  - If the command is a 5 = Wait until Proceed, a single Proceed will go to the next index.

If you turn off the rung, the index will stop at the end of the move and command. However if you re-activate it, the Point table resets.

Outputs

- **Mcode** is a user defined number. It is loaded into the Mcode register at the beginning or end of the move depending on the **MCodeActive** input. It is a DINT so you can put -2,147,483,648 to +2,147,483,647. You can have duplicates.
- **CurrentIndex** shows the index number that is active.
- Transitions can be monitored by with the MCodes.

Point Table UDT

The Point Table uses an array of UDTs.

TypeMove	DINT	Decimal	0=Absolute 1=Incremental 2=No Move	Read/Write
PositionData	DINT	Decimal		Read/Write
SpeedData	DINT	Decimal		Read/Write
AccelTime	DINT	Decimal		Read/Write
DecelTime	DINT	Decimal		Read/Write
M_Code	DINT	Decimal	Active at start or end of step (depends on MCodeActive input)	Read/Write
Command	DINT	Decimal	Commands are executed at the end of a move	Read/Write
Value	DINT	Decimal	Depends on the command	Read/Write

- If you want the line to be non-motion, set **TypeMove** = 2, the **Command** will still execute.
- If you want the line to be a complete NOP, set **TypeMove** = 2 and the **Command** = 1 (continue).
- **Command** and its **Value** are active at the end of a move.
- **Mcode** is simply a user selected number to identify each point. You can match two different point tables even if the actual indexes are not the same. You can have duplicates. This number is loaded into the output register at the start of the step. It is a DINT so you can put -2,147,483,648 to +2,147,483,647.
- **Wait** until ***Proceed*** Input is active at the end of a move, then execute next index.
- **Timer Override** is a Timer that can be overridden by the ***Proceed*** input. If the ***Proceed*** input is not toggled, the timer will time out normally and the next index will execute.
- **Jump** will jump to the index in ***Value***

- **ConditionalJump** will jump to the index in Value if the ***CondJump*** input is high. If not, it will go to the next index.

Commands

Command		Value	Description
0	End	<don't care>	Servo will stop – end of point table
1	Continue	<don't care>	Execute next step
2	Timer	Timer duration in milliseconds	Start timer at end of move
3	Jump	Destiination index	Jump to index at end of move
4	TimerOverride (Proceed)	Timer duration in milliseconds	Start timer at end of move until timer times out or Proceed is active
5	Wait until Proceed	<don't care>	Stop at end of move and wait for Proceed input
6	ConditionalJump	if CondJump = 1, Destination index if CondJump = 0, <don't care>	At the end of the move, if CondJump input = 1 jump to destination index if CondJump input = 0 move to next index

Examples

(MCodeActive = 0)

Index	M_code	TypeMove	Position	Speed	Command	Value
3	4	0 (abs)	1000	10000	1 (cont)	<don't care>

The servo will output an Mcode of 4, do an absolute move to position 1000 at a speed of 10000. Then it will continue to next index.

(MCodeActive = 0)

Index	M_code	TypeMove	Position	Speed	Command	Value
5	12	1 (rel)	1500	20000	2 (timer)	3000

The servo will output an Mcode of 12, do an incremental move of 1500 counts at a speed of 20000, then wait 3 seconds before executing the next step.

(MCodeActive = 1)

Index	TypeMove	Position	Speed	Command	Value	M_code
3	0 (abs)	21000	10000	6 (condjump)	10	4
4	0 (abs)	30000	20000	1 (cont)	<don't care>	5
10	1 (rel)	2000	20000	1 (cont)	<don't care>	17

The servo will do an absolute move to 21000 at a speed of 10000. When it arrives it will output an Mcode of 4, then perform a conditional jump. If the CondJump input is high, the next point will be index 10. If the CondJump input is low, index 4 will be executed.

(MCodeActive = 1)

Index	TypeMove	Position	Speed	Command	Value	M_code
15	0 (abs)	34000	20000	5 (wait)	<don't care>	11

The servo will do an absolute move to 34000 counts at a speed of 20000, then will stop and wait until the Proceed input goes high. When the Proceed input goes high, it will output an Mcode of 11 and continue to the next index.

(MCodeActive = 1)

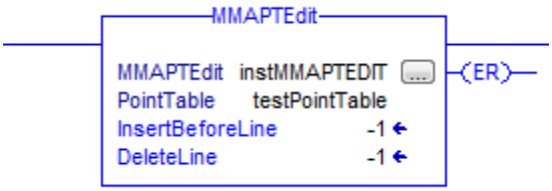
Index	TypeMove	Position	Speed	Command	Value	M_code
15	0 (abs)	34000	20000	1 (cont)	<don't care>	11
16	3 (no move)	<don't care>	<don't care>	2 (timer)	5000	12

The servo will do an absolute move to 34000 counts at a speed of 20000. When it arrives, it will output an MCode of 11. The next index is a non-motion index and will simply start a timer for 5 seconds. When it times out, it will output 12 and move to the next index.

MMAPTEdit

MMAPTEdit is an editing tool for inserting lines or deleting lines in a point table. Otherwise you will have to copy and paste.

- MMAPTEdit is Rising Edge only



Save after an edit.

- AOI is rising edge, to avoid multiple executions. You must toggle off to reset it.
- Delete or Insert function disabled by (-1)
- If both inputs are (-1) nothing is done
- For a 30 element array, ranges are 0 to 29.
- If ***both*** inputs have values of 0 or greater, you will get an error.
- If any input value is 30 or greater, you will get an error.

DeleteLine

- After a valid Delete, the Delete line number is changed to -1 for safety.
- If you delete a line, any jump label (Command 3 or 6) greater than DeleteLine will be decremented
    - If you delete a line, and there is a jump to the old line, the jump line number will not be modified. E.g. Index 8 has a jump to index 3. If you delete index 3, the jump from index 8 still goes to index 3

InsertBeforeLine

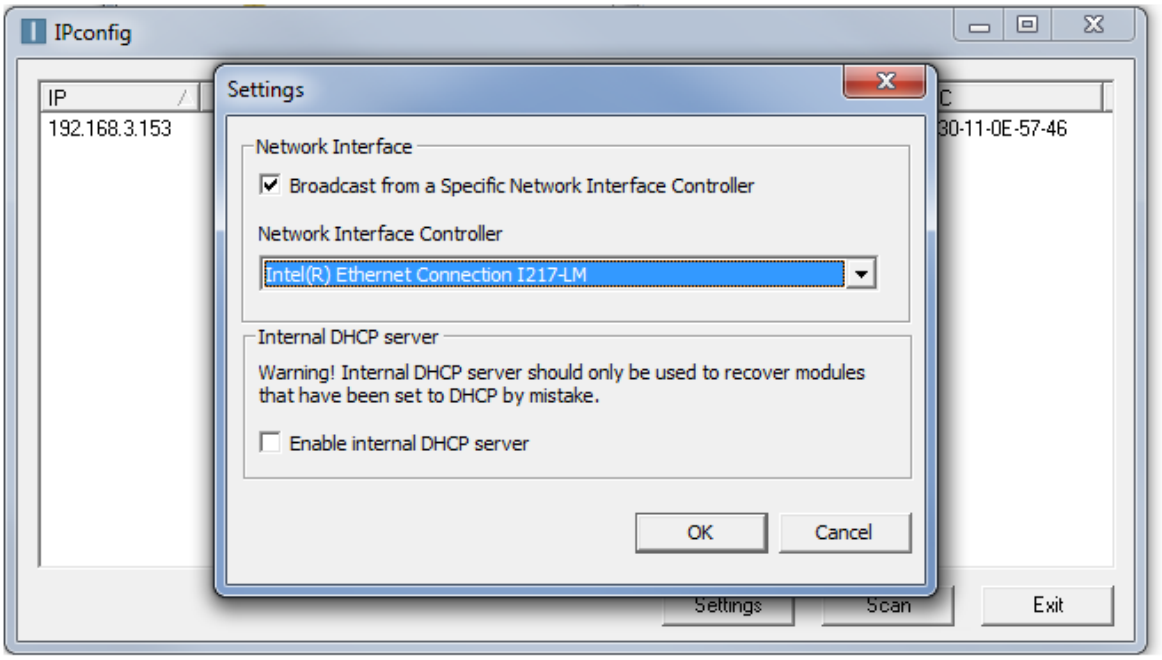
- After a valid Insert, the Insert line number is changed to -1 for safety.
- ***An INSERT will insert a "Do Nothing" index with a MoveType of 2 (no motion), a Command of 1 (continue), an MCode of -12345. All other elements will be 0.***
- E.g. If you insert before 5, the new "Do Nothing" index will be 5. The original index 5 will then be 6.

- After a valid Insert, the last index of the array is lost.
- If you insert a line, any jump label (Command 3 or 6) greater than or equal to InsertBeforeLine will be incremented
    - If you insert a line, and there is a jump to the first moved line, the jump line number will be modified.
      E.g. Index 8 has a jump to index 3. If you insert a line before index 3, the jump from index 8 will go to index 4.

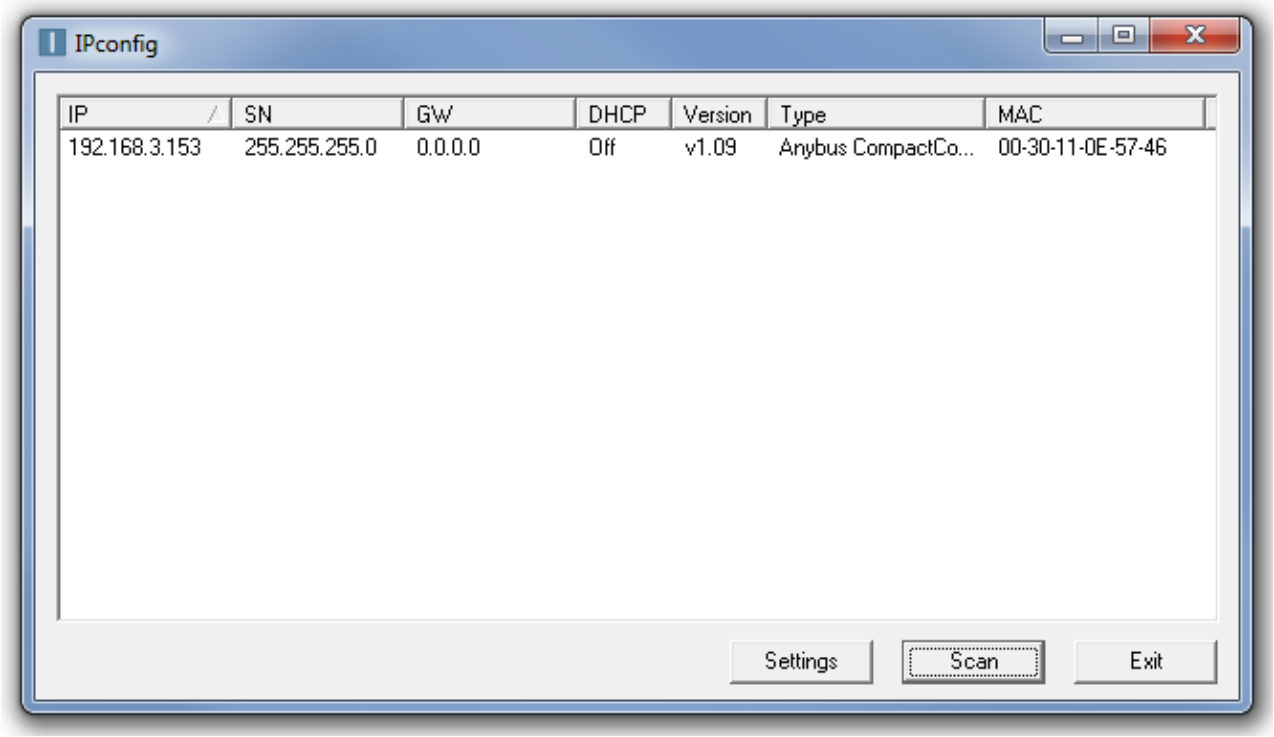
7. Setting IP Address

To change the IP address of the servo amplifier Anybus module, use IPConfig tools from HMS Anybus

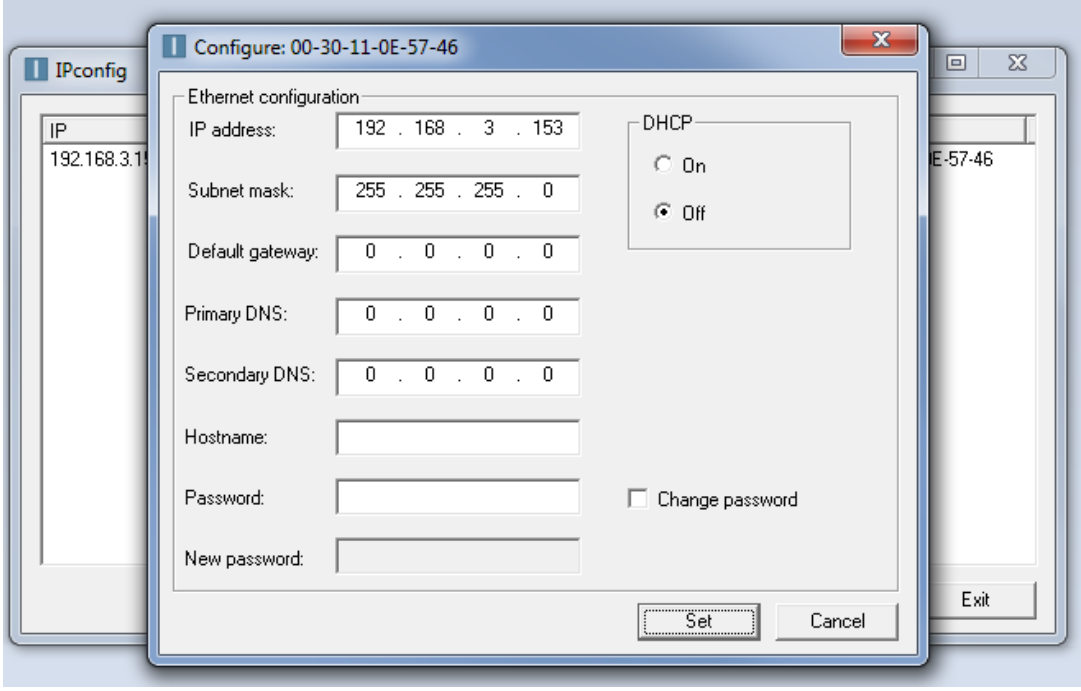
1) Launch the Ipconfi tools
2) Click on settings and make sure you are not wireless.



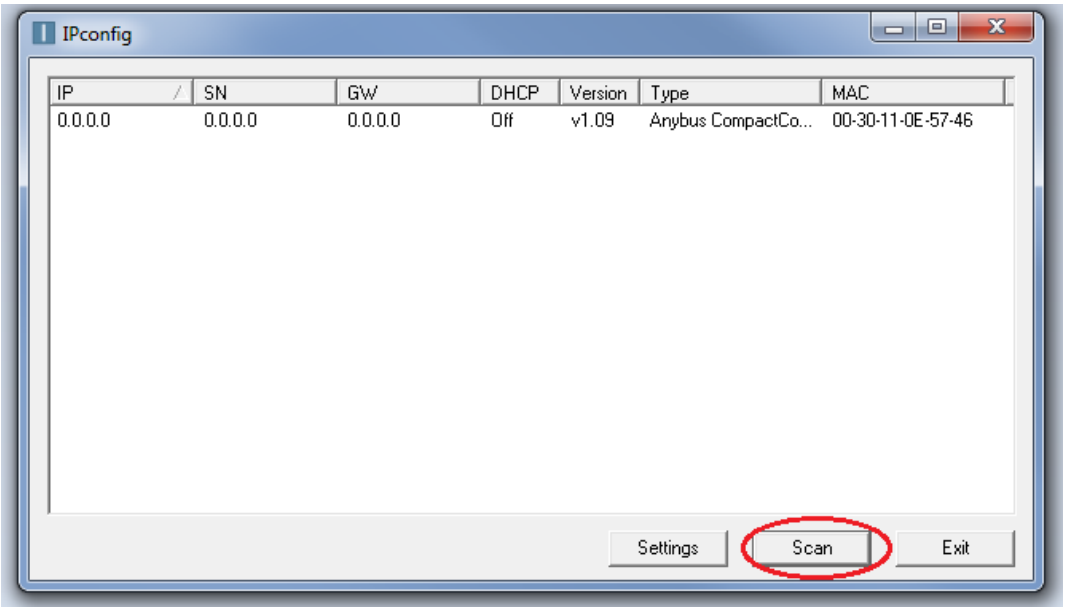
3) Click on **Scan** to find Anybus module



4) To change setting, first double click on the entry to get to settings and Change the IP address here.



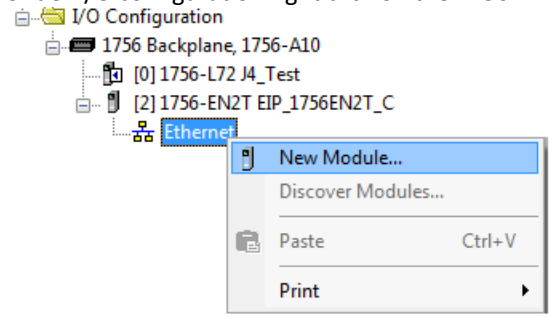
5) If you change anything, entry will go blank, click on **Scan** again to verify change

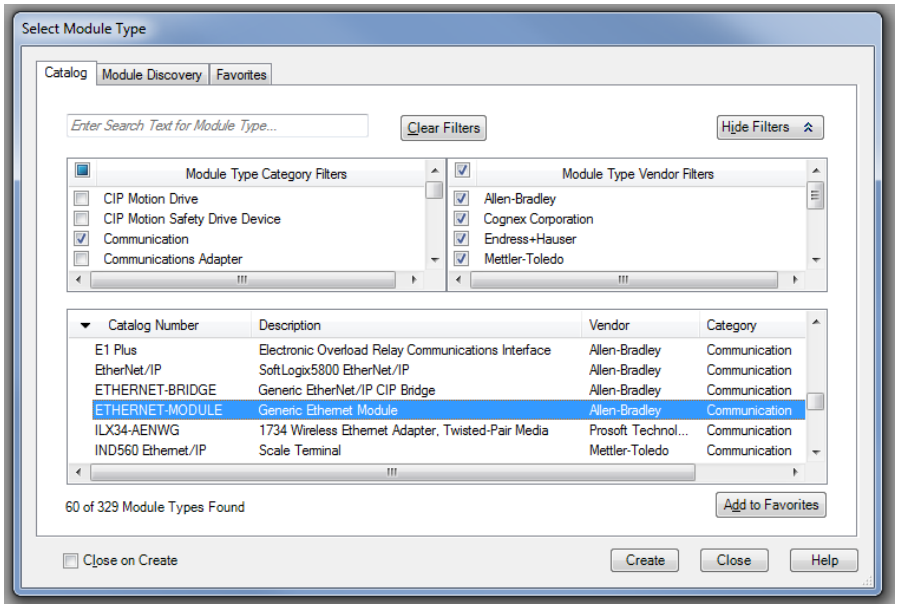


8. CompactLogix PLC configuration

The configuration steps of a CompactLogix project are described this section. These steps are used to communicate with a servo amplifier. It is assumed that the user has basic knowledge in using RSLogix5000 software to perform the basic configuration steps.

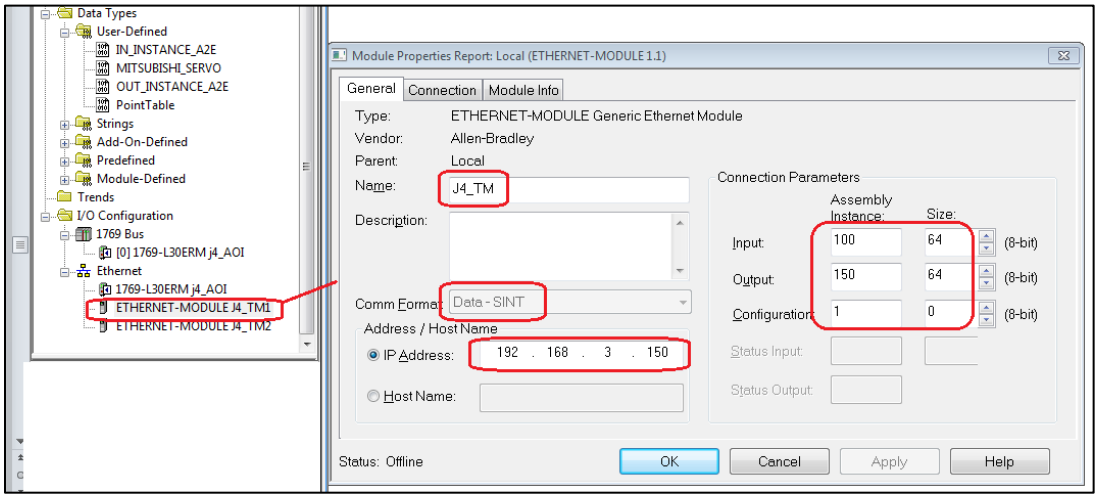
1) Create a new project in the RSLogix5000 using the proper revision level as the ControlLogix controller. In this example, the revision level is 20.
2) Under I/O configuration right click on the 1756 EN2T and Ethernet icon and and choose “New Module…”



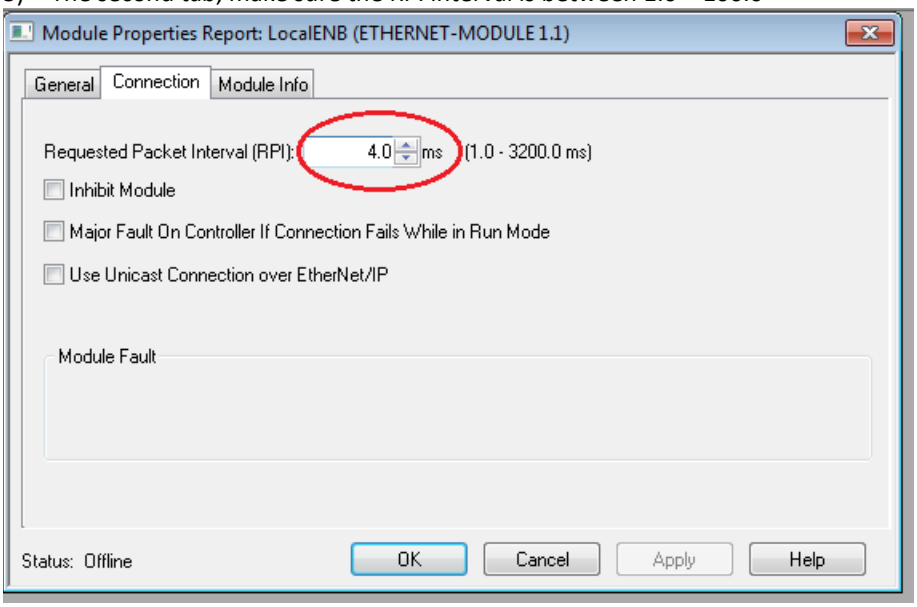


3) In the “Select Module” pop-up window, choose the “Communications” and then select "Generic Ethernet Module".

4) Enter the proper Name, Communication Format, IP Address and Connection Parameters and the Input and Output instances with their sizes. Be sure to specify the Comm Format is a SINT.

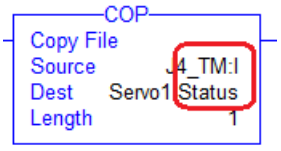


5) The second tab, make sure the RPI interval is between 1.0 – 100.0



At the top of the program file put the COP (read from <servo>:I) instruction. Use the Generic Ethernet module as the source and put the file into the MITSUBISHI_SERVO UDT in the field **Status**.

For this example:



the Generic Ethernet module is **J4_TM**

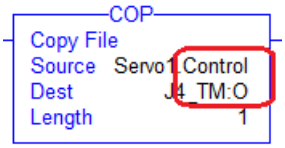
The Mitsubishi servo UDT is **Servo1** and you must have Servo1.***Status***

At the bottom of the program file, put the COP (write to <servo>:O) instruction. Use the MITSUBISHI_SERVO UDT field **Control** and write it to the Generic Ethernet Module.

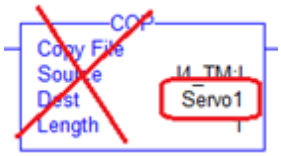
For this example:

the Generic Ethernet module is **J4_TM**

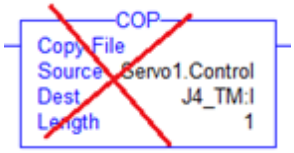
The Mitsubishi servo UDT is **Servo1** and you must have Servo1.***Control***



Note Servo1 does not have DOT.Status



Note here, the Control word (which is an ***output***) is incorrectly matched with the module ***Input***



9. Appendices

Error Codes

Error codes have Extended Errors EXERR to further define the fault.

Error Code	EXERR	Description
1013	0	Invalid Axis
	1	
	2	Invalid Move Type – Absolute/Incremental (MMAM, MMAPT)
	3	Invalid Position
	4	Invalid Speed out of range (MMAJ, MMAM, MMAH)
	5	Invalid Acceleration rate (MMAJ, MMAM)
	6	Invalid Deceleration rate (MMAJ, MMAM)
	7	Invalid Home Type (MMAH)
	8	Invalid Torque Slope
	9	
	10	Invalid Speed = 0 (MMAM, MMAJ)
	11	Invalid Home Speed (MMAH)
	12	Invalid Home Creep Speed (MMAH)
	13	Invalid Parameter Group Invalid (MMWP, MMRP)
	14	Invalid Speed cannot be negative (MMAT)
	15	Invalid Parameter Invalid (MMWP) E.g. < 0
	16	Invalid input number (MMAR)
	17	Invalid AutoTune Response (MMCFG)
	18	Invalid Scaling Numerator (MMCFG)
	19	Invalid Scaling Denominator (MMCFG)
	20	Invalid Limit Switch Enable (MMCFG)
	21	Invalid Estop Switch Enable (MMCFG)
	22	
	23	
	24	Invalid Forward Torque Limit (MMCFG)
	25	Invalid Reverse Torque Limit (MMCFG)
	26	A single probe cannot be both continuous and latched (MMAR)
	27	
	28	

Error Code	EXERR	Description
1014	AOI ID[1]	Not Homed (MMAM, MMAPT, MMAW)
1015	<error code from servo>	Servo in Warning condition SW.7
1016	AOI ID[1]	Cannot change modes unless servo is stopped/done e.g. attempt MMAM if in Jog (EXERR = 1) e.g. attempt MMAJ if Homing (EXERR = 4) e.g. attempt MMAH if in Gearing (EXERR = 6)
1017	AOI ID[1]	MMAS is on, cannot try to move anything.
1020	<error code from servo>	Servo in Error condition SW.3
1021	AOI ID[1]	Servo is disabled and a move instruction was attempted – servo may be shutdown, not enabled
1022	<Which parameter>	EXERR value 16#10 (16) = Illegal code number. • Check Parameter InstructionCode2 E.g. You wrote "6083" instead of "16#6083", Subindex out of range
1023	<Which parameter>	16#20 (32) = Unreachable parameter. Check WriteData –
1024	<Which parameter>	16#30 (48) = Out of range Address out of range or value out of range E.g. For writing data I out 16#6064. I should have put 16#6064_0000
1025	<index line>	An invalid entry in the Point Table UDT. EXERR gives the index number
1050	1	Gearing dropped out (2D16.14 went low)

[1] In the Mitsubishi_Servo UDT, this is placed in the extended register EXERR to indicate which AOI instruction set this error
(e.g. 1017, 1021) or which mode/AOI is active (e.g. 1016)
1 = MMAJ - Jog
2 = MMAM - Move/Position
3 = MMAT - Torque
4 = MMAH - Home
5 = MMAPT – Point Table
6 = MMAG – Gearing
7 = MMAW - Watch

Cross Reference of CANOpen Object vs J4 Parameters

J4	CANOpen
PA_ _	2001h to 2020h
PB_ _	2081h to 20C0h
PC_ _	2101h to 2150h
PD_ _	2181h to 21B0h
PE_ _	2201h to 2240h
PF_ _	2281h to 22C0h
PL_ _	2401h to 2430h
PT_ _	2481h to 24D0h
PN_ _	2581h to 25A0h

In order to convert Mitsubishi J4 parameters (e.g. PA03, PT45, etc) to the CANOpen addressing, there is a convenient conversion table (shown left).

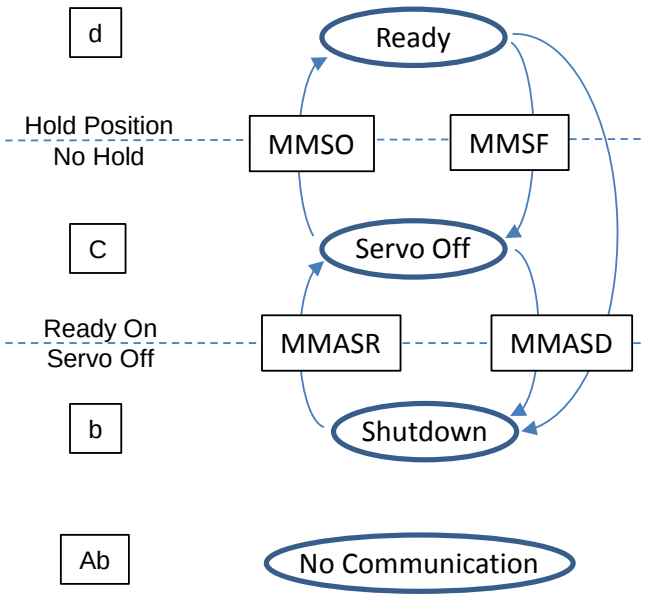
Therefore, there are 2 addresses for most parameters, the original CANOpen address, and the mapping table.

E.g. PT1 = 2481h = 9345_{10}
PT45 = 24ADh = 9389_{10}

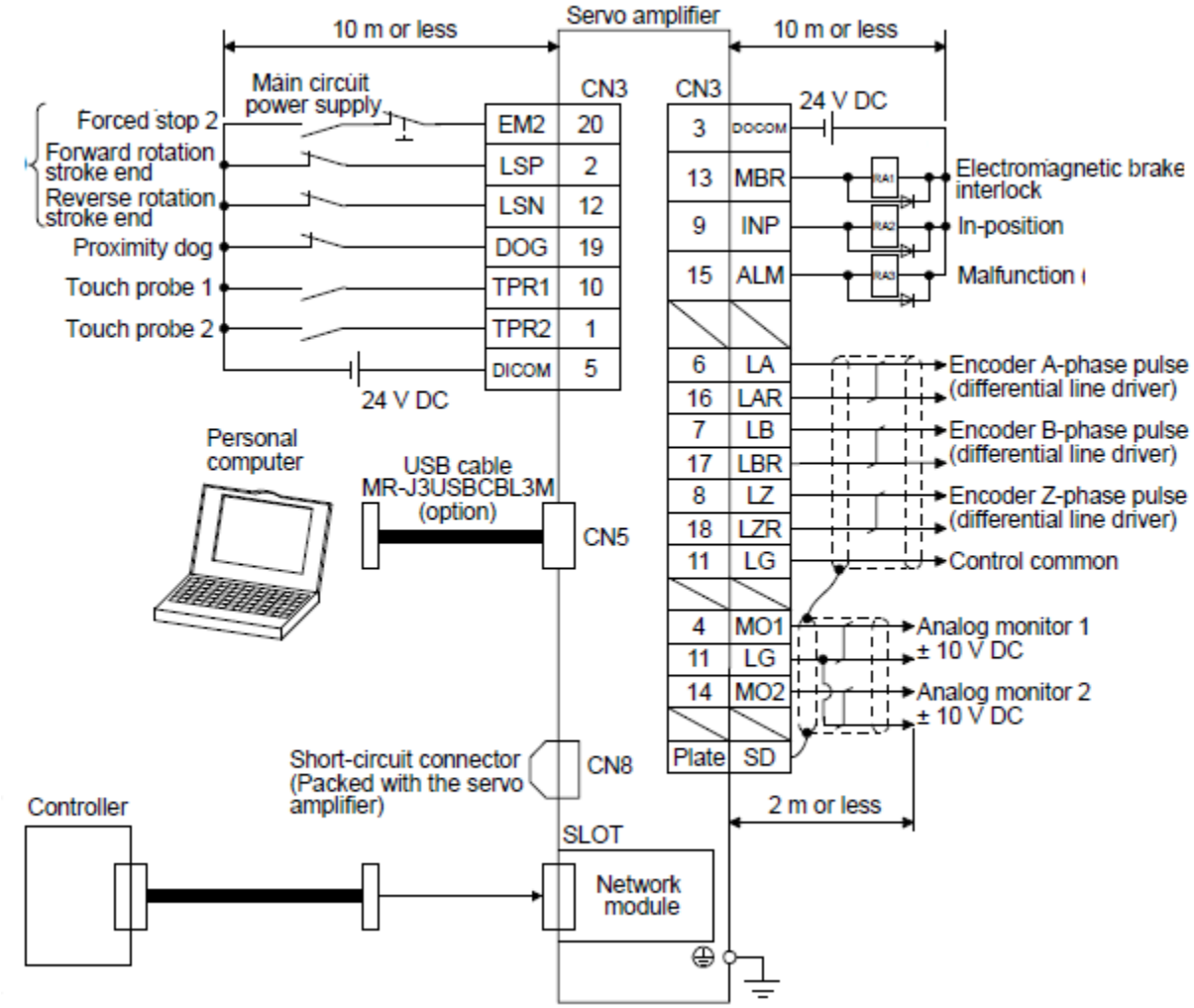
LED & Servo States

d	"d" : Indicates ready-on and servo-on status MMSO Servo On - enables the servo to "d"
C	"C": Indicates ready-on and servo-off status. MMSF Servo Off – • disables the servo to "C" from "d" • If at "C" or "b", no effect. MMASR Shutdown Reset – resets <servo>.ShutdownStatus flag • If at "b" enables the servo to "C" • If at "C" or "d", no effect.
b	"b" : Indicates ready-off and servo-off status. MMASD Shutdown - electronically disables the servo to "b". Must trigger MMASR to reset it internally. Uses <servo>.ShutdownStatus flag.
Ab	No communication between J4 and PLC. Ping both IP addresses for J4 Anybus and PLC Generic Ethernet module. If they ping, then the problem is in the PLC program.

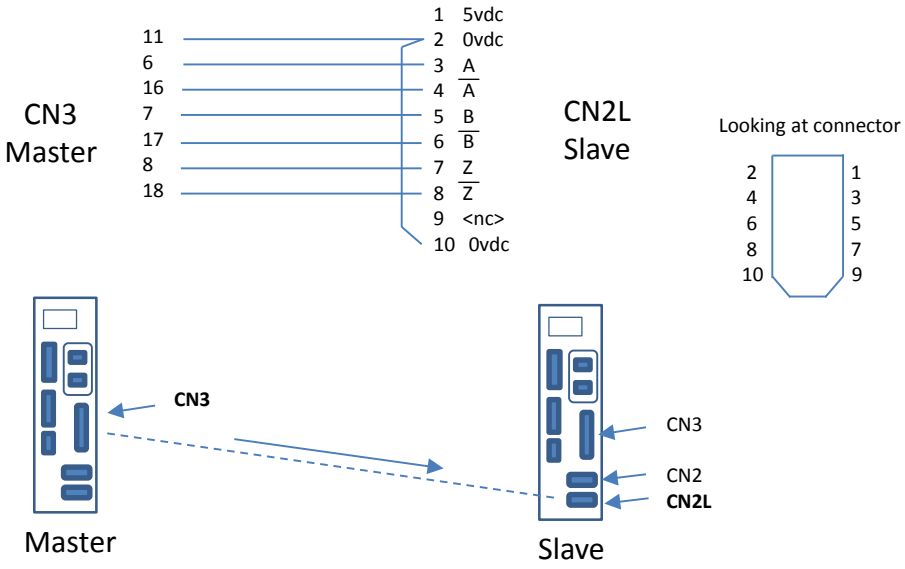
(Power On state based on state at Power Off)



Servo I/O interface Connections

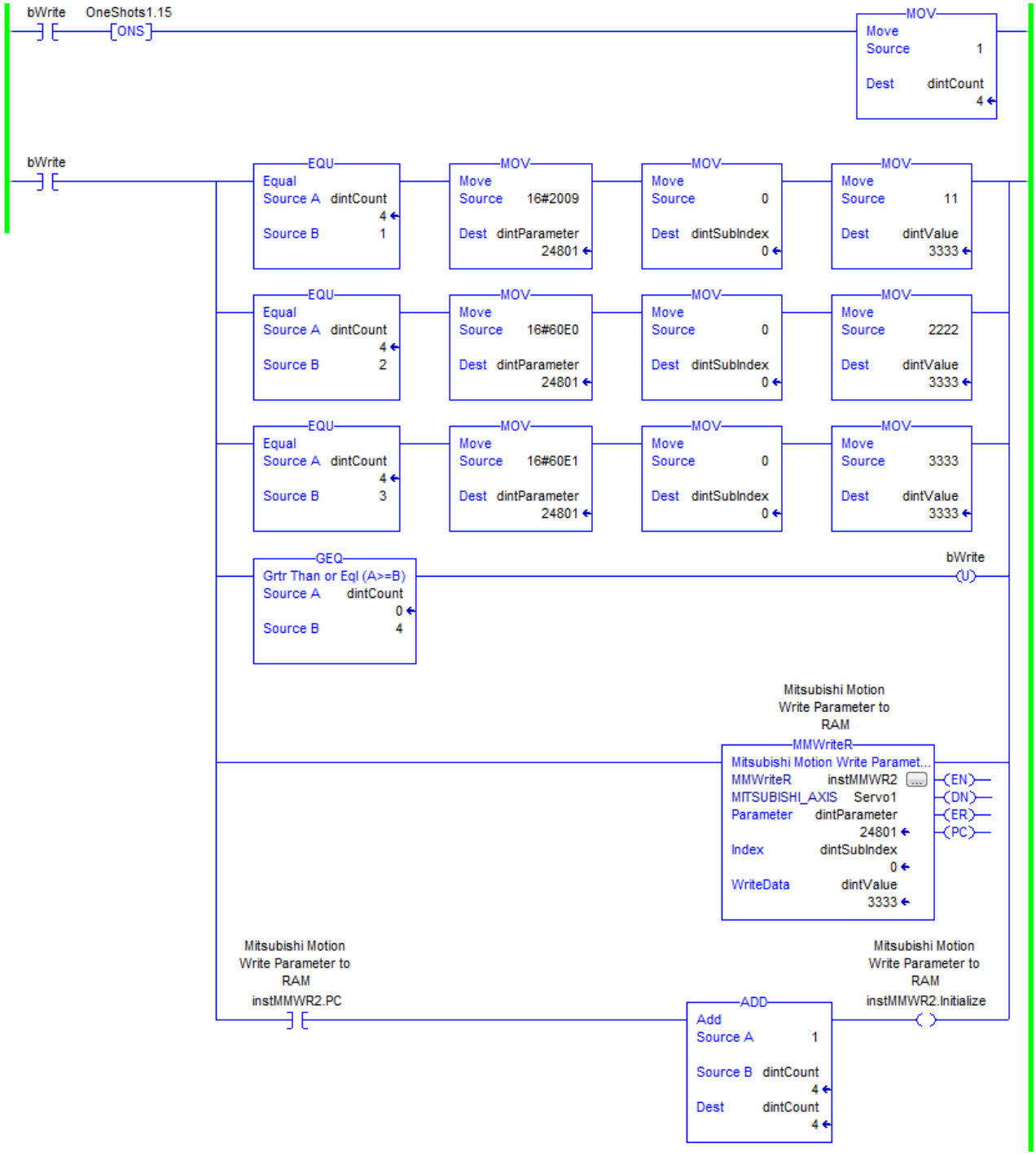


Master slave interface Connections for gearing



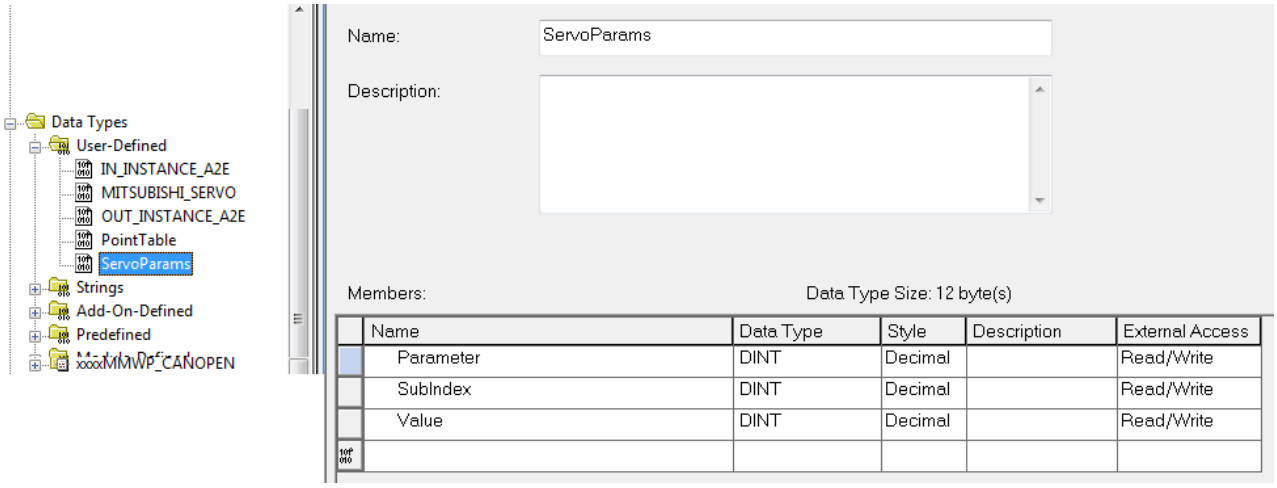
Write Loop 1

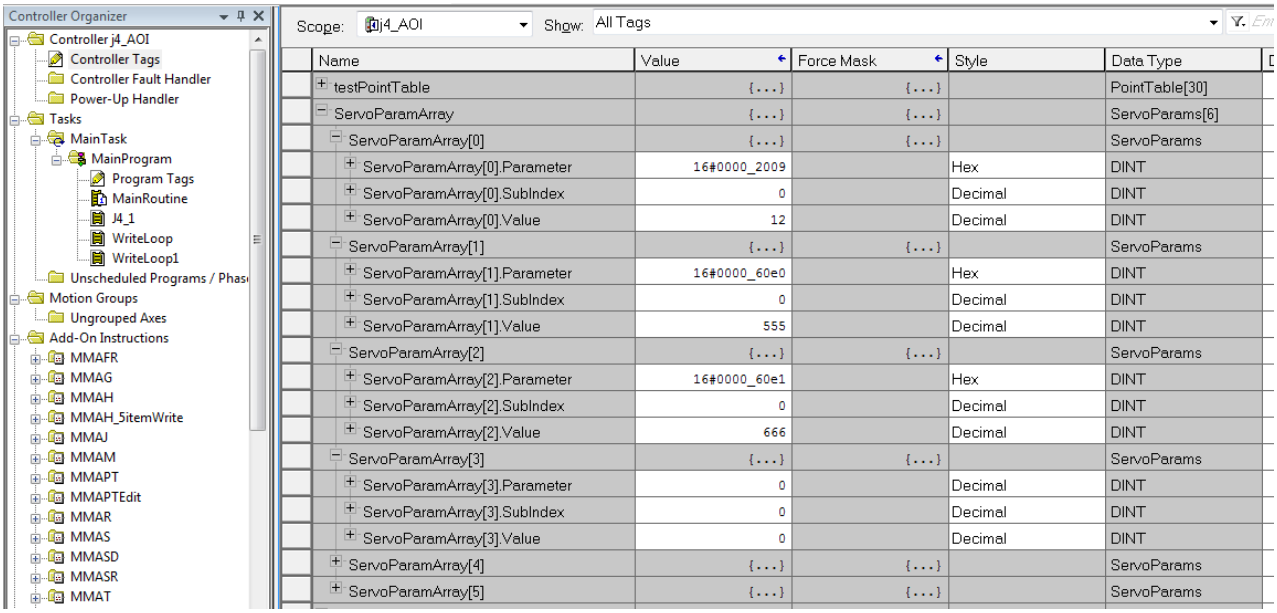
In the following example, 3 parameters are written to the servo. You set **bWrite** high and it will reset itself after the loop is done. You can increase the number of parameters, you simply update the **GEQ** to turn off the loop.

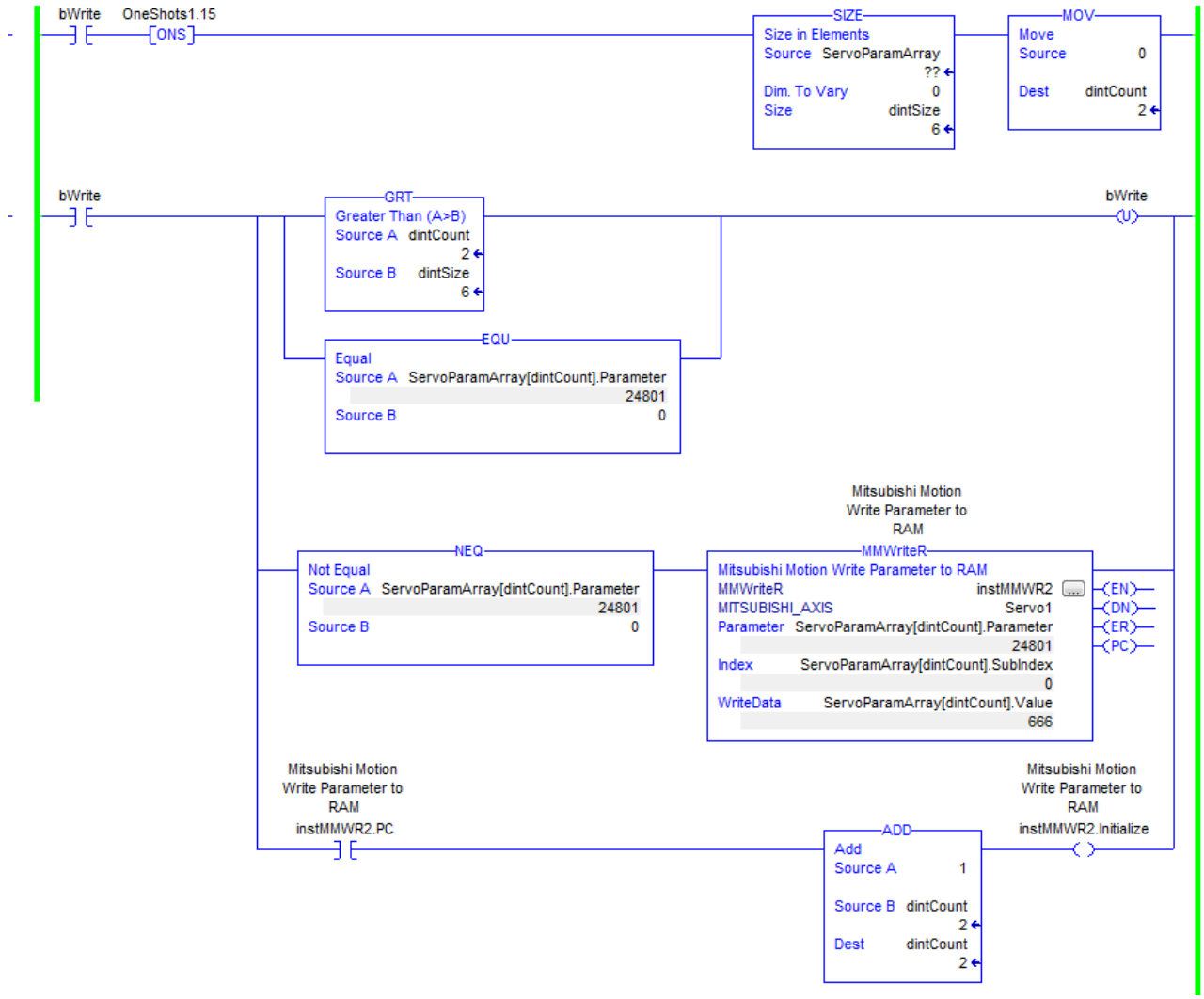


Write Loop 2

For this example, the parameters are stored as an array. The ladder logic sequences thru the array to the end or if it encounters a 0 parameter.






bWrite
OneShots1.15
ONS
SIZE
Size in Elements
Source ServoParamArray
??
Dim. To Vary 0
Size dintSize
6
MOV
Move
Source 0
Dest dintCount
2
bWrite
GRT
Greater Than (A>B)
Source A dintCount
2
Source B dintSize
6
bWrite
U
EQU
Equal
Source A ServoParamArray[dintCount].Parameter
24801
Source B 0
Mitsubishi Motion
Write Parameter to
RAM
NEQ
Not Equal
Source A ServoParamArray[dintCount].Parameter
24801
Source B 0
MMWriteR
Mitsubishi Motion Write Parameter to RAM
MMWriteR instMMWR2
MITSUBISHI_AXIS Servo1
Parameter ServoParamArray[dintCount].Parameter
24801
Index ServoParamArray[dintCount].SubIndex
0
WriteData ServoParamArray[dintCount].Value
666
EN
DN
ER
PC
Mitsubishi Motion
Write Parameter to
RAM
instMMWR2.PC
Mitsubishi Motion
Write Parameter to
RAM
instMMWR2.Initialize
ADD
Add
Source A 1
Source B dintCount
2
Dest dintCount
2

10. Revisions

April 15th 2016 – Document created and Released, Version 1.0