

# Converting from Allen-Bradley Analog Modules to RMC Motion Controllers



These instructions describe in detail how to convert your motion application from Allen-Bradley 1756-HYD02, 1756-M02AS, or 1756-M02AE motion modules to a Delta Motion RMC motion controller. These instructions also reference other sources, such as the RMC Startup Guides, videos, RMCTools help topics, etc.

## Contents

Overview .....	2
1. Choosing an RMC.....	3
2. Set up the Axes in the RMC.....	5
3. Communications Overview .....	7
4. Setting up the RMC for Communications.....	10
5. Setting up the PLC for Communications .....	11
6. Importing Tags from RMCTools for Cyclic I/O Data.....	15
7. Ladder Logic for Sending Commands.....	19
8. Using the MSG Instruction .....	20
9. PLC Motion Instructions Crossover.....	22
10. Homing.....	26
11. Registration.....	27
12. Support .....	27
A1. Addendum 1 – Example Code Functional Description .....	28

## Overview

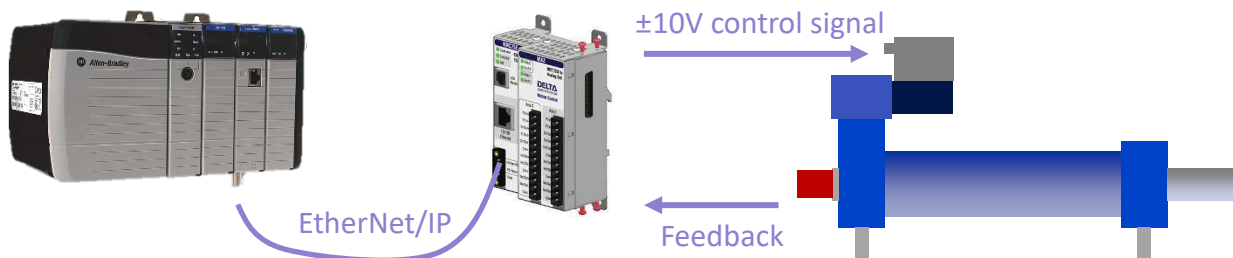
### Feature-Complete Alternative

The RMC motion controllers have all the features required to replace the Allen-Bradley motion modules, including:

- $\pm 10V$  signal to valve or motor drive
- Feedback: Start/Stop, PWM, SSI, or Quadrature Encoder (A+,A-,B+,B-,Z+,Z-)
- Homing
- Registration
- Motion instructions including gearing, splines (cams), and more
- Easy tuning and advanced control algorithms

### Connection Method

The RMC motion controllers are stand-alone controllers that communicate with the ControlLogix PLC via EtherNet/IP.



Converting requires setting up the EtherNet/IP communication and replacing the motion instructions in the ladder logic with code that sends motion commands to the RMC. The PLC instructions and RMC motion commands differ, so further code modification is typically required.

### Example Code

Delta Motion has provided example code for Rockwell Studio 5000 Logix Designer to assist customers with the transition to the RMC. The example includes AOI's that allow the new RMC code to follow a similar structure as existing applications for the 1756-HYD02, 1756-MO2AS or 1756-MO2AE module.

For details on the example code and AOI's, see section **A1. Addendum 1 – Example Code Functional Description**.

# 1. Choosing an RMC

For more details, see:

- Products webpage: <https://deltamotion.com/products/>

To configure an RMC and request a quote, see <https://deltamotion.com/rmcquote/rmcselect.php>.

## 1. Review Requirements

Review the requirements of your existing application. Note that the Allen-Bradley modules have the following features:

Module	Axes	Features (per axis)					
		Output	Feedback	Fault Input	Enable Output	Home Input	Registration Input
HYD02	2	±10V	Start/Stop or PWM	✓	✓	✓	✓
M02AS	2	±10V	SSI	✓	✓	✓	✓
M02AE	2	±10V	Quadrature encoder	✓	✓	✓	✓

Determine which of these features your application uses.

## 2. Choose RMC Controller

Choose an RMC motion controller based on the number of axes of control required:

If your application has:	then use:
1 or 2 axes	RMC75E
Up to 18 axes	RMC200L
Up to 50 axes	RMC200

Note: The RMC150 is an option, but Delta encourages use of the newer RMC75E and RMC200 controllers.

## 3. Choose RMC modules

Choose the required modules for your RMC, based on the required amount and type of I/O.

### RMC75E

Required Signals:	RMC75E Modules	Additional I/O on Module (per axis)
±10V and Start/Stop or PWM	MA1 (1 axis)	Fault Input, Enable Output
±10V and SSI	MA2 (2 axis)	
±10V and Quadrature Encoder	QA1 (1 axis) QA2 (2 axis)	Fault Input, Enable Output, Home Input, 2 Registration Inputs

The RMC75E also had additional expansion modules available, such as discrete I/O, analog inputs, and a quadrature input.

### RMC200

The RMC200 Standard requires:

- Base module: B5, B7, B11, or B15
- Power supply module: PS4D or PS6D (B15 only)
- CPU40 module

The RMC200 Lite requires:

- Base module: B5L or B7L
- CPU20L module (includes built-in power supply)

Choose I/O modules to meet your requirements:

Required Signal:	RMC200 or RMC200L Modules	Additional Discrete I/O on Module
±10V	CA4 (4)	4 Fault Inputs, 4 Enable Outputs
	CV8 (8)	4 Fault Inputs, 4 Enable Outputs
	U14 (2)	8 I/O, individually configurable as Fault Inputs or Enable Outputs
Start/Stop or PWM	S8 (8)	
SSI	S8 (8)	
Quadrature Encoder	QA4 (4)	8 Home Inputs, 8 Registration Inputs
	U14 (2)	4 High-Speed Inputs for registration or Z 4 configurable I/O for Fault Inputs or Enable Outputs
	S8 (1 RS-422 encoder input, no Z)	

#### 4. Verify RMC Specs

Make sure to review the datasheets of the RMC to verify the modules will meet your needs:

[RMC70 Datasheet](#)

[RMC200 Datasheet](#)

#### 5. Request Quote

To configure an RMC and request a quote, see <https://deltamotion.com/rmcquote/rmcselect.php>.

## 2. Set up the Axes in the RMC

For more details, refer to the Startup Guide for your controller, which guides you in detail through wiring, defining axes, and configuring axes:

- [RMC70 Startup Guide](#)
- [RMC200 Startup Guide](#)

The general steps for setting up, configuring, and tuning the axes is as follows:

### 1. Wire the RMC

The existing HYD02, M02AS, or M02AE wiring should readily connect to the RMC modules with only slight modifications, if any. Follow the wiring diagrams in the Startup Guide or in the RMCTools online help wiring topics:

[https://deltamotion.com/support/webhelp/rmctools/#t=Wiring\\_and\\_Installation%2FWiring\\_Guidelines.htm](https://deltamotion.com/support/webhelp/rmctools/#t=Wiring_and_Installation%2FWiring_Guidelines.htm)

### 2. Install RMCTools

The RMCTools software is available to download at no cost from Delta Motion's download page:

<https://deltamotion.com/dloads/downloads.php>.

### 3. Start a New Project in RMCTools

After connecting your PC to the RMC via USB or Ethernet, start RMCTools. In the **Startup** dialog, select **Create a New Project** and complete the Startup Wizard, which will lead you to go online with the RMC.

### 4. Define the Axes

The RMC will start with default axis definitions. You can assign the physical hardware to the internal software axes as your application requires.

To change these definitions, do the following in RMCTools:

- In the **Project** tree, expand the **Axes** folder and double-click **Axis Definitions**.
- Use the dialog to view the axis definitions and change them if you would like. For more details, click the **Help** button in the dialog.

### 5. Test the Actuator

For each axis, there may be axis parameters that need to be set for the RMC to correctly connect to the actuator. Once they are set, you will send the Direct Output command to send a voltage to the valve or motor and check that it moves. Refer to the Startup Guide for detailed instructions.

## 6. Configure and test the Feedback

For each axis, there may be axis parameters that need to be set for the RMC to correctly interpret the feedback signal. Once configured, you will again move the actuator to test the feedback. Refer to the Startup Guide for detailed instructions.

## 7. Tune the Axes

Use the **Tuning Tools** to tune the axes. The Tuning Tools pane offers several methods for tuning:

- **Auto-tuning**

Use the Tuning Wizard and choose the Auto-tuning wizard. The RMC will automatically make a move. RMCTools will determine the system model, and the Gain Calculator will open. Move the axis back and forth while moving the slider bar to select from a range of gains, from conservative to aggressive.

- **Tuning with Existing Plot**

You can use the Proportional gain to generate motion and then capture those plots of the motion. Then, in the Tuning Wizard, choose those plots to determine a system model, then the Gain Calculator will open. Move the axis back and forth while moving a slider bar to select from a range of gains.

- **Manually Tune**

You can manually tune axis using the procedure in the RMCTools help:

<https://deltamotion.com/support/webhelp/rmctools/#t=Starting Up the RMC%2FTuning%2FTuning a Position Axis.htm>

Now your axes should be set up and tuned so that motion can be performed on the system. The next step is to set up communications with the PLC so that the PLC can send motion commands to the RMC.

## 3. Communications Overview

For more details, see:

- ControlLogix to RMC Communication Video: [deltamotion.com/controllogixvideo](http://deltamotion.com/controllogixvideo)
- ControlLogix to RMC Communication Step-by-Step Instructions: [www.deltamotion.com/controllogixformc](http://www.deltamotion.com/controllogixformc)
- RMCTools Help Topic: [EtherNet/IP Overview](#)

The RMC communicates with the ControlLogix via EtherNet/IP, using cyclic I/O data and also the MSG instruction. Typically, most of the frequent communication is performed via cyclic I/O, and larger or infrequent communications are performed with the MSG instruction.

### Commands or User Programs

When communicating between a PLC and an RMC, a decision must be made as to where the motion sequence logic is performed. There are two main methods:

- **PLC Handles Motion Sequencing**  
The motion sequencing is in the PLC, and the PLC sends motion commands to the axes in the RMC. The advantages are that the motion logic can easily be viewed in the PLC, and when converting from Allen Bradely 1756 analog modules, this method likely requires less changes.
- **RMC User Programs Handle Motion Sequencing**  
The motion sequencing is in the RMC user programs and the user programs send the motion commands to the axes. In this situation, the PLC would start user programs in the RMC, either by sending a command to start a user program, or by writing to a variable that triggers a user program to start. The advantages are that the user programs respond at the loop time of the RMC (typically 1 msec), without a communication delay, and that the PLC can be offloaded, meaning less logic in the PLC.

These two methods can also be used together.

### Recommended Method for Allen Bradely Analog Module Conversion

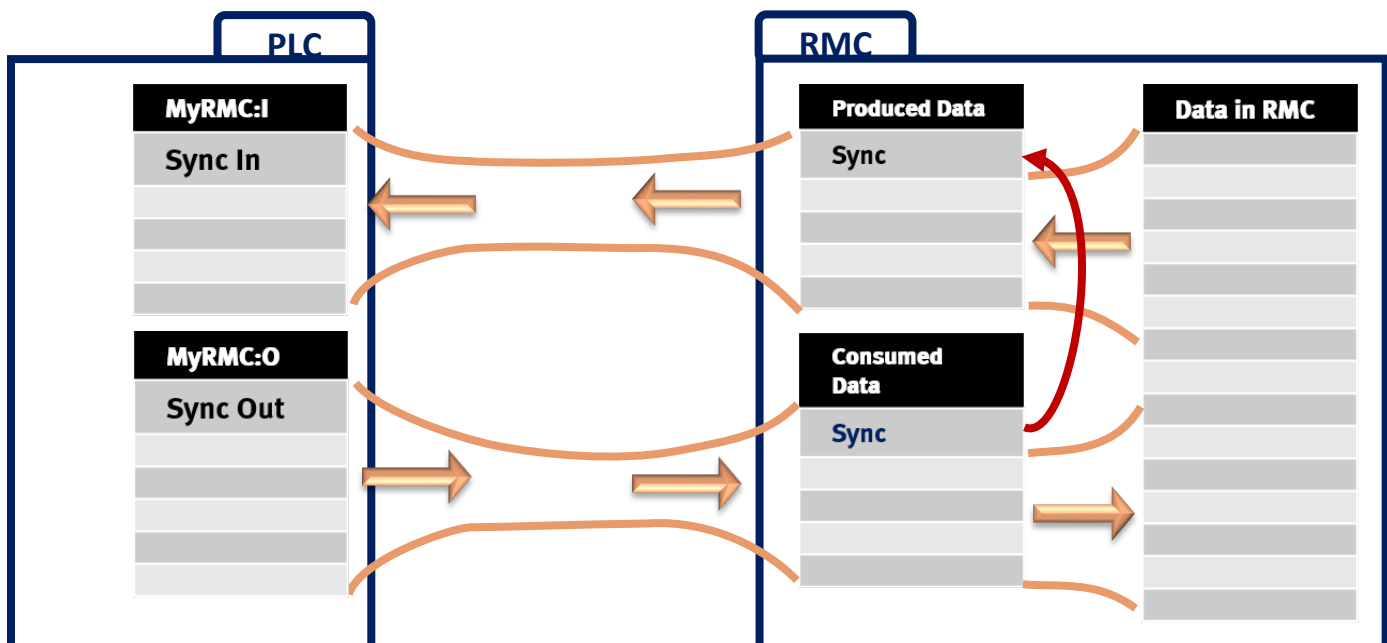
When converting from the Allen Bradely 1756 analog modules to an RMC, the conversion process will likely be simpler if the PLC sends motion commands to the RMC, since logic already does so for the 1756 analog modules. Any motion that needs to respond quickly to motion events may need to be placed in the RMC user programs.

## Cyclic I/O Basics

Cyclic I/O is useful for data that is communicated frequently. It is not intended for large data transfers or infrequently communicated data.

As shown in the image below, the cyclic I/O operates as follows:

- The PLC's incoming data, which appears in the **:I** array in the PLC, is continuously received from the RMC at the specified requested packet interval (RPI). Typical RPI's are 10 or 20 milliseconds.
- The PLC's outgoing data, which appears in the **:O** array in the PLC, is continuously sent to the RMC at the same specified RPI.
- The RMC's **Consumed Data** is not applied to the data in the RMC until the Sync register value changes. This makes it easy in the PLC logic to build up the data, and when it is ready to send, simply change the Sync Out value (the first array item). This ensures the data is applied simultaneously in the RMC. Notice that if multiple I/O connections are used, each connection will have its own Sync register, and the data between connections is not guaranteed to be applied simultaneously.
- When the PLC's Sync Out value is changed, the RMC not only applies the data, but also echoes the Sync value back to the PLC. This allows the PLC ladder logic to know when the data has been applied, so that it can proceed to the next logic operations.
- The user defines the **Data in RMC** that is communicated. The **Produced Data** is usually status information and comes from the Indirect Data Map, where the user can enter any data tags from the RMC as explained below. The **Consumed Data** is usually for commanding the RMC and typically goes to the Command Area or the Indirect Data Map. The options for **Data in RMC** are explained in the **Cyclic I/O Data in the RMC** section below.






### Cyclic I/O Data in the RMC

The user must define which registers will be communicated in the cyclic I/O.

- **Produced Data**

Produced data is sent from the RMC to the PLC. It is typically status information, such as Target Positions, Actual Positions, Status bits, and Error Bits.

To choose which registers to include in the Produced Data:

- In RMCTools, in **Project View**, double-click **Address Maps**, then choose the **Indirect Data Map**.
- In the first row, click in the **Map To** cell, then click the  button.
- Browse to the desired tag, click it then click **Add**. Or, press the spacebar to add the tag and automatically select the next tag.
- If your application has multiple axes, use the **Duplicate for Axes** button to quickly add the same tags for many axes.

- **Consumed Data**

Consumed data is sent from the PLC to the RMC. It is typically command registers or variables. If the data is commands, the **Consumed Data** can directly point to the command area. If the data is variables or other registers, then choose a section of the Indirect Data Map for the consumed data, and add the variables and possible other registers to the area of the Indirect Data Map, in the same way as described above for produced data.

## 4. Setting up the RMC for Communications

For Allen-Bradley analog module conversions, you will likely want to set the produced and consumed cyclic data as follows:

**Produced Data:** Indirect Data Map, starting with item 0

**Consumed Data:** Command Area, starting with Axis 0 command

This will allow the PLC to send commands to the RMC, similarly to how the PLC sends motion instructions to the HYD02, M02AS, and M02AE. The PLC will also receive status data from the RMC.

**Note:** For RMC200 applications with many axes, you may need to use multiple I/O connections to fit all the data. Take special care, as the data in each connection is not guaranteed to arrive at the same time as the other connections.

### Configuring the Ethernet IP Address

1. In the RMCTools project tree, expand **Modules** and double-click the **RMC (CPU)** module.
2. On the **Ethernet** page, set the IP address.

### Configuring the EtherNet/IP Properties

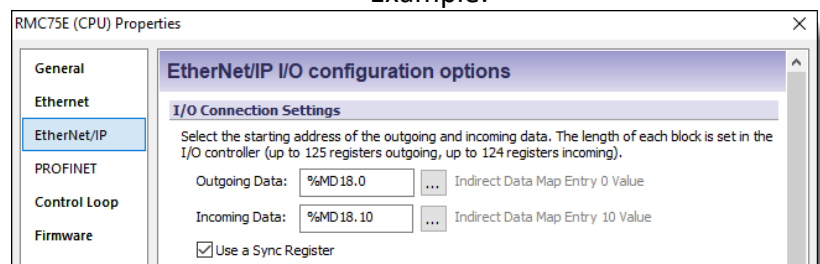
1. In the RMCTools project tree, expand **Modules** and double-click the **RMC (CPU)** module.
2. On the **EtherNet/IP** page, select the following options:
  - **Outgoing Data:**
    - **RMC75E:** %MD18.0 (Indirect Data Item 0)
    - **RMC200:** %MD8.0 (Indirect Data Item 0)

For large RMC200 applications, you may need to fill out the other connections also.
  - **Incoming Data:**
    - **RMC75E:** %MD25.0 (Axis 0 Command)
    - **RMC200:** %MD16.0 (Axis 0 Command)

For large RMC200 applications, you may need to fill out the other connections also.
  - Make sure **Use a Sync Register** is checked.

3. Click **OK**. The settings will be applied to the controller.

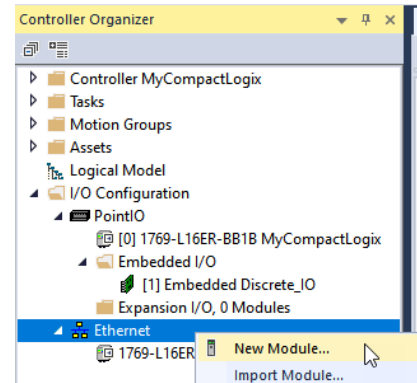
Example:



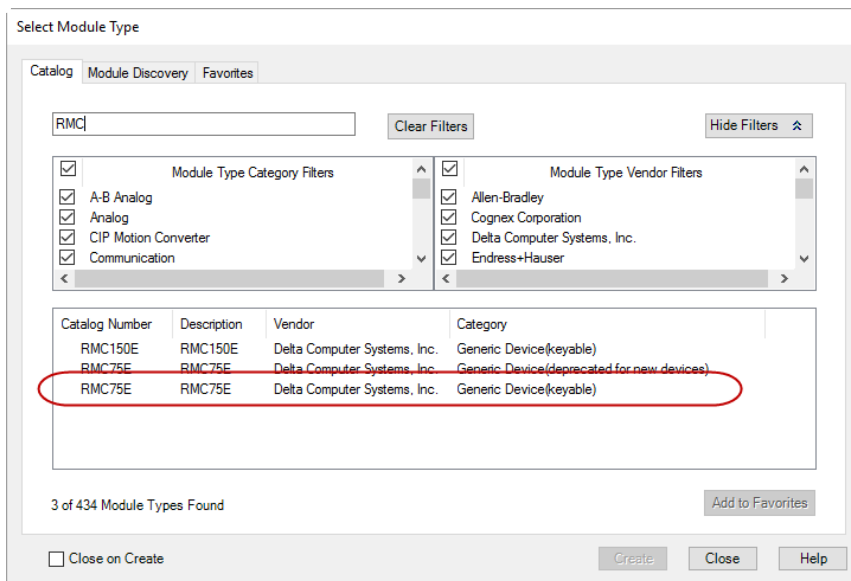
## 5. Setting up the PLC for Communications

You will add the RMC to the ControlLogix I/O Configuration and verify that the communications works. Later, you will export tags from the RMC and import them into the PLC to make the data more readable.

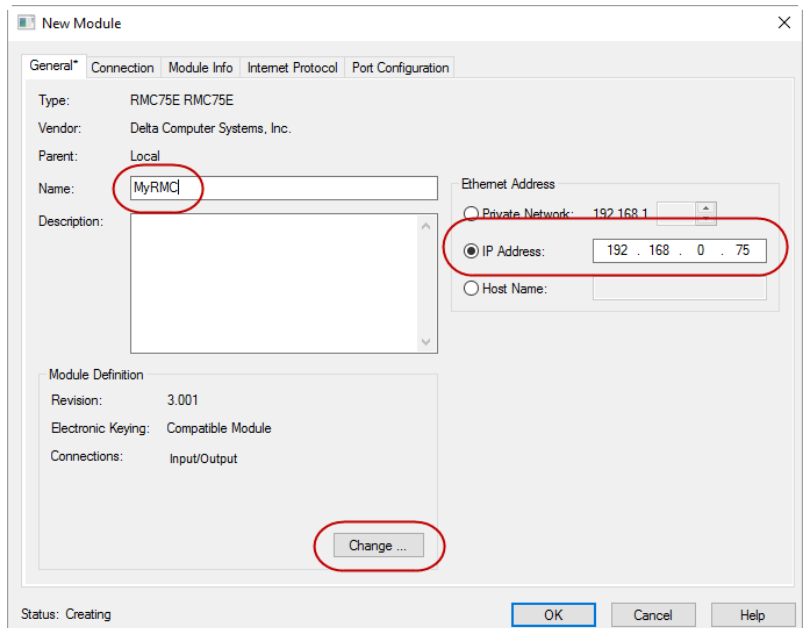
1. Start Studio 5000. Stay offline until you have completed the I/O Configuration described below.
2. In Studio 5000, on the **Tools** menu, click **EDS Hardware Installation Tool**.
3. Click **Next**, choose **Register an EDS file** and click **Next**.
4. Browse to **C:\Program Files\RMCTools\EDS** and choose the most recent EDS file for your RMC and click **Open**.
5. Complete the wizard.
6. In the **Controller Organizer** window, at the bottom of the list, right-click **Ethernet** and choose **New Module**.
7. In the list, search for **RMC**. Choose your RMC. Make sure to choose the one that is keyable. Click **Create**.



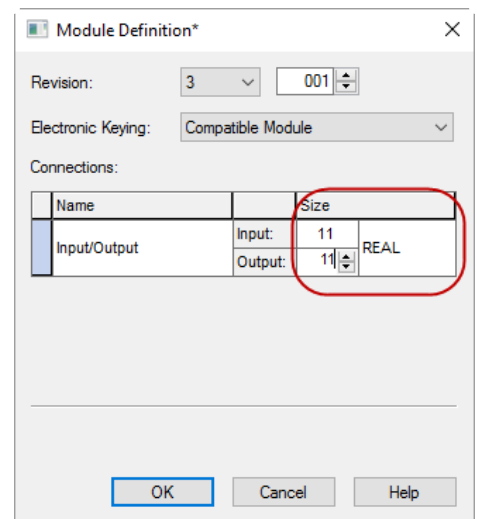
Example:



8. In the **New Module** dialog, enter a **Name** (hereafter referred to as [your RMC name]) and **IP Address** as shown below, then click **Change**.



9. In the **Module Definition** dialog, in the **Size** area, choose **REAL**.



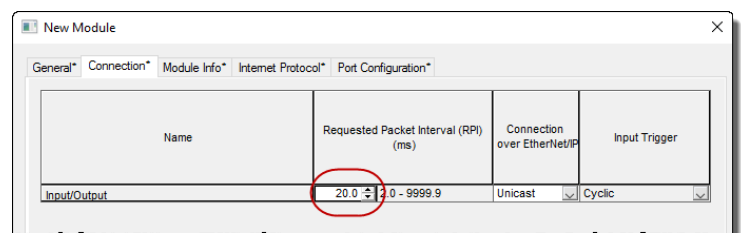
10. Set the **Input** to the number of items you entered in the Indirect Data Map, plus one for the Sync Register. For example, if you have 18 items, enter 19.

11. Set the **Output** to the number of axes you want to send command to, times 10, plus one for the Sync Register. For example, for 3 axes, enter 31.

**Note:** For RMC200 applications with many axes, you may need to use multiple I/O connections to fit all the data. Fill out the Input and Output boxes for each connection you will be using. Remember that the data in each connection is not guaranteed to arrive at the same time as the other connections.

12. Click **OK**.

13. On the **Connection** tab, set the **Requested Packet Interval** to the desired value, such as 20 milliseconds, then click **OK**.



14. Close the **New Module** dialog.

15. Download to the PLC and go online.
16. The I/O connection should now be established automatically between the RMC and PLC.
17. Verify that the I/O connection is established by checking the following items:
  - a. In Studio 5000, the your RMC node under **I/O Configuration** should not have a warning shield next to it.
  - b. On the RMC, the **Net** LED should be solid green.
  - c. In RMCTools, in the Event Log, you will see that the I/O connection was established.
  - d. In RMCTools, on the **Controller** menu, choose **View Communication Statistics**. On the **EtherNet/IP Diagnostics** page, you will see information on the CIP I/O connection.

**Verifying the Cyclic Input Data from the RMC**

To verify that the PLC is receiving cyclic data from the RMC:

1. In Studio 5000, open **Controller Tags**, and select the **Monitor Tags** tab.
2. Expand the **[your RMC name]:I** tag and the **[your RMC name]:I.Data** array under it. This array shows all the data coming from the RMC.

Name	Value	Force Mask	Style	Data Type	Description
MyRMC:I1	{...}	{...}		_024E:R200_C...	
MyRMC:I1.ConnectionFaulted	0		Decimal	BOOL	
MyRMC:I1.Data	{...}	{...}	Float	REAL[21]	
MyRMC:I1.Data[0]	0.0		Float	REAL	
MyRMC:I1.Data[1]	0.0		Float	REAL	
MyRMC:I1.Data[2]	0.0		Float	REAL	
MyRMC:I1.Data[3]	0.0		Float	REAL	
MyRMC:I1.Data[4]	0.0		Float	REAL	
MyRMC:I1.Data[5]	0.0		Float	REAL	
MyRMC:I1.Data[6]	0.0		Float	REAL	
MyRMC:I1.Data[7]	0.0		Float	REAL	

3. In RMCTools, open the Indirect Data Map. Compare the Indirect Data Map register values in the Current column with the **MyRMC:I.Data** array. You should see the same values, but notice that the **MyRMC:I.Data** registers are shifted down by one register because the first register, item .Data[0], is the Sync Register.

## Verifying the Cyclic Output Data to the RMC

The PLC's cyclic output data, as set up according to the instructions above, goes to the RMC's command area. To verify that this is working:

1. In Studio 5000, in **Controller Tags** on the **Monitor Tags** tab, expand the **[your RMC name]:O.Data** tag.

Name	Value	Force Mask	Style	Data Type	Description
MyRMC:O1	{...}	{...}		_024E:R200_C...	
MyRMC:O1.Data	{...}	{...}	Float	REAL[41]	
MyRMC:O1.Data[0]	0.0		Float	REAL	
MyRMC:O1.Data[1]	0.0		Float	REAL	
MyRMC:O1.Data[2]	0.0		Float	REAL	
MyRMC:O1.Data[3]	0.0		Float	REAL	
MyRMC:O1.Data[4]	0.0		Float	REAL	
MyRMC:O1.Data[5]	0.0		Float	REAL	
MyRMC:O1.Data[6]	0.0		Float	REAL	
MyRMC:O1.Data[7]	0.0		Float	REAL	

Array element .Data[0] is the Sync Register. The command area for each axis is 10 elements, so Axis 0 command starts at element 1, Axis 1 starts at element 11, and so on.

2. First, make sure that it is safe to send commands to the axis, even erroneous commands if there is some problem with the communications. Setting the RMC's axis in simulate mode may be useful.
3. Enter an RMC command number and its parameters, starting at array element 1 for Axis 0. For example, the following values would be a Move Absolute (20) command, where 20.0 is the number for the Move Absolute and the values that follow are the command parameters:

Name	Value	Force Mask	Style	Data Type	Description
MyRMC:O1	{...}	{...}		_024E:R200_C...	
MyRMC:O1.Data	{...}	{...}	Float	REAL[41]	
MyRMC:O1.Data[0]	0.0		Float	REAL	
MyRMC:O1.Data[1]	20.0		Float	REAL	
MyRMC:O1.Data[2]	15.0		Float	REAL	
MyRMC:O1.Data[3]	10.0		Float	REAL	
MyRMC:O1.Data[4]	100.0		Float	REAL	
MyRMC:O1.Data[5]	100.0		Float	REAL	
MyRMC:O1.Data[6]	0.0		Float	REAL	
MyRMC:O1.Data[7]	0.0		Float	REAL	

These values are now in the PLC and are being sent to the RMC each RPI. However, the RMC will not apply them until the Sync Value is changed.

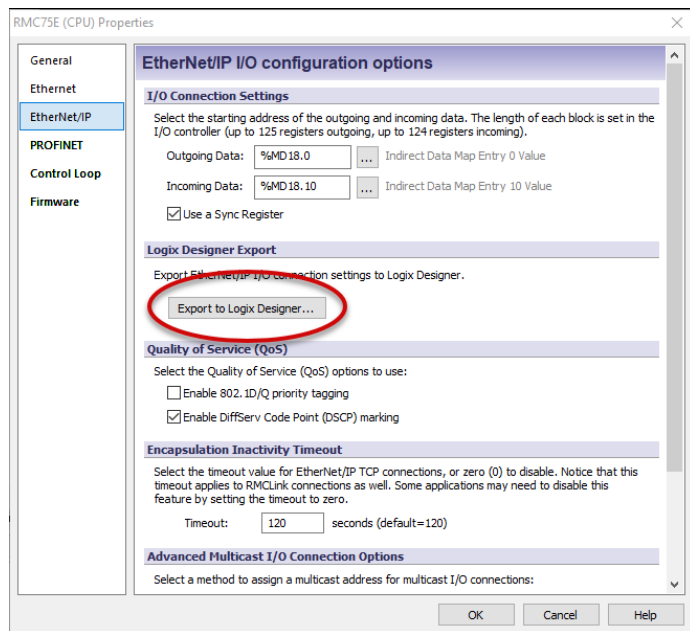
4. Change the value of the first item in the array, which is the Sync value. This will cause the RMC to apply the data, and the RMC will execute the command.
5. In RMCTools, open the Event Log and review the latest entries. You should see where the Sync value changed and all the values of the PLC's output data at that moment.

## 6. Importing Tags from RMCTools for Cyclic I/O Data

RMCTools is able to export tags that can be imported to the ControlLogix. This will provide readable tag names for the I/O data, including names for the individual bits of the Status and Error Bits. The export/import will also include three rungs of ladder logic to simplify the Sync register. This logic is not required to be used, but may be helpful. Make sure you have first set up the communications in the RMC and PLC, as described above.

### Exporting the Tags from RMCTools

1. Before starting, make sure to have the following information:
  - a. The name of the RMC as entered in your Logix Designer project.
  - b. If using the RMC200, the number of I/O connections.
  - c. The number of PLC Input Data registers (not including the Sync register).
  - d. The number of PLC Output Data registers (not including the Sync register).
2. In RMCTools, in the **Project** pane, expand the **Modules** folder and double click the CPU module.
3. On the EtherNet/IP page, click **Export to Logix Designer** then click **OK**.



4. In the **RMC I/O Module Name in Logix Designer** box, type the name of the RMC exactly as entered in your Logix Designer project (*[your RMC name]*), then click **Next**.

Logix Designer Export Wizard

**Welcome to the Logix Designer Export Wizard**

Use this wizard to export EtherNet/IP I/O communication data tags and example synchronization logic. The exported components can then be imported to a Rockwell Logix Designer project.

Steps to export EtherNet/IP I/O communication data to Logix Designer:

1. Configure the RMC EtherNet/IP I/O Connection Settings in RMCTools.
2. Set up the Indirect Data Map (or Variable Table) with the data to be sent and received.
3. Configure the RMC I/O module in Logix Designer.
4. Use this wizard to export the data tags to a Logix Designer .LSX file.
5. Import the .LSX file into Logix Designer as program rungs.

If the connection was configured in RMCTools to include the sync register:

- The exported data will include example ladder logic rungs for using the sync register.
- Make sure to define the data length correctly in the PLC. The length should be the number of data registers plus one for the sync register.

Enter the following information:

RMC I/O Module Name in Logix Designer:

< Back   Next >   Cancel   Help

5. Enter the number of incoming and outgoing data registers not including the sync register, then click **Next**.

Logix Designer Export Wizard

**Register Setup**

Set the number of RMC registers to be transferred as Input and Output data for each connection, not including the Sync Register, if used.

Enter the number of data registers to transfer in each direction:

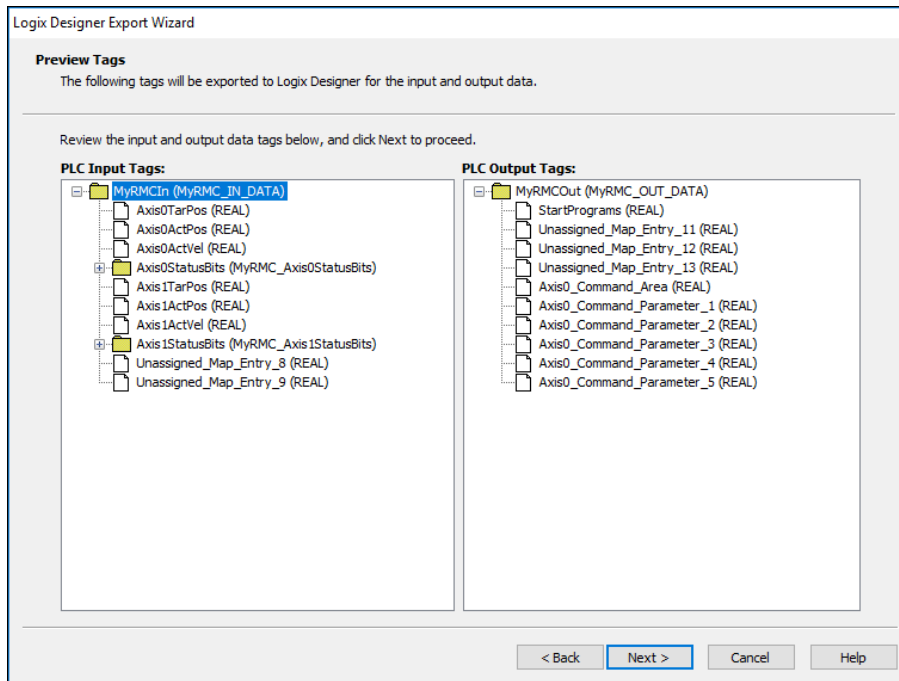
PLC Input Data (RMC Outgoing Data)	Start Address: <input type="text" value="%MD18.0"/>	Indirect Data Map Entry 0 Value	Number of Registers: <input type="text" value="10"/>
PLC Output Data (RMC Incoming Data)	<input type="text" value="%MD18.10"/>	Indirect Data Map Entry 10 Value	<input type="text" value="10"/>

**Note:** Do not include the Sync Register in the number of registers above.

< Back   Next >   Cancel   Help



6. Review the information and click **Next**.



7. Click **Browse** to browse to a location to save the exported file (such as the desktop) and click **Next**.

8. Review the location and name of the saved file, review the **Next Steps** section, then click **Finish**.

### Importing the RMCTools Tags to ControlLogix

1. In Studio 5000, go offline.
2. In the desired location in the ladder logic, do the following:
  - a. Right-click a rung and choose **Import Rungs**.
  - b. Browse to the file that you saved earlier and click **Import**.
  - c. In the **Import Configuration** window, click **OK** to accept defaults.
3. Review the **controller tags** that were added:
  - **[your RMC name]In**
  - **[your RMC name]Out**
  - **[your RMC name]SendReq**
  - **[your RMC name]Busy**

4. In the **Controller Organizer**, expand **Assets, Data Types**, then **User Defined**. Review the User-Defined Data Types that were added by the import:

- **RMC\_IN\_DATA**
- **RMC\_OUT\_DATA**
- **RMC\_STATUS\_BITS**

5. Review the ladder rungs that were added:

**Rung 1:** Continuously copies the data from **[your RMC name]:I** into **[your RMC name]In**

**Rungs 2 and 3:** Handles writing the data from **[your RMC name]Out** to **[your RMC name]:O** when requested and updating the Sync Registers accordingly.

6. Download the project to the PLC and make sure it is in Run mode.

7. Verify that the **[your RMC name]In** tag shows that same data as in the RMCTools Indirect Data Map, Current column.

8. To verify sending commands with the **[your RMC name]Out** tag:

- a. Enter a command number and parameters into the **[your RMC name]Out** tag.
- b. Set **RMCSendReq** to 1.

9. In the RMCTools Event Log, you should see that an EtherNet/IP request was made, and the command was issued.

## 7. Ladder Logic for Sending Commands

In order to send commands to the RMC via cyclic I/O, with the PLC's output data going to the RMC's command area, the ladder logic must populate the **[your RMC name]Out** tag with the commands and command parameters. Delta suggests using the logic described below. Parts of it are included when importing tags from RMCTools (described in section 6 above), such as the SendReq and SendBusy bits, and logic to handle the Sync register.

1. Throughout the ladder logic, when the code is to send a command to an axis, it populates the correct area in the **[your RMC name]Out** tag.
2. Toward the end of the ladder, check to see if any of the **[your RMC name]Out** tag is non-zero. If it is non-zero, set the **[your RMC name]SendReq** bit.
3. In some following rung, if the SendReq bit is on, apply the data to the RMC by doing the following:
  - a. Check that the **[your RMC name]Busy** bit is off and the SyncIn and SyncOut match. If either of these conditions are not true, it would mean a transaction is in process.
  - b. If no transaction is in process, set the **[your RMC name]Busy** bit to indicate that a transaction is in process.
  - c. Copy **[your RMC name]Out** to the **[your RMC name]:O** tag with the CPS instruction, which guarantees the data is copied all at once, and that subsequent logic operations related to that copy will occur after the data is all copied.
  - d. Clear the **[your RMC name]Out** tag.
  - e. Change the Sync out by incrementing it by one, and modulo it so that it doesn't overflow.
  - f. Wait for the SyncIn and SyncOut to match, which means the RMC has applied the data and echoed the changed Sync value.
  - g. Clear the Busy bit.

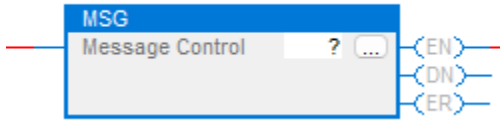
This method makes it fairly easy to send commands, as any part of the ladder that wishes to send a command simply populates the Out tag. The synchronizing logic toward the end of the ladder takes care of making sure the data is applied to the RMC and data that was sent is properly cleared in preparation for future send requests.

The cyclic I/O Requested Packet Interval (RPI) will define the communications delay. If the delay is too long for your application, use a shorter RPI, or consider writing some motion sequences in the RMC user programs.

If you decide to heavily use user programs in the RMC, you may wish to trigger them by writing to variables instead of sending motion commands from the PLC, so your logic may be different than described here.

## 8. Using the MSG Instruction

Use the [MSG instruction](#) for large or infrequent communications, such as writing curve data or occasionally writing to an individual axis parameter.



### Address Format

The MSG instruction requires using the DF1 address format for registers in the RMC. The RMC supports the F or L register types, for example L8:0 or F56:0. See the [DF1 Addressing](#) help topic for more details.

In the RMC200, not all registers have DF1 addresses. For those that do not, the user can assign DF1 addresses. All registers in the RMC75E have DF1 addresses. To find DF1 addresses of RMC registers, see the [DF1 Address Map](#) help topic.

You can access any DF1-addressed register in the RMC as an L or F type. If the PLC must convert the data type of any RMC data (such as when reading a block of mixed-type data), use the COP instruction, which preserves all the bits in the data and just interprets it as the desired data type.

### Configuring the MSG Instruction

To configure the ControlLogix MSG instruction for reading from or writing to the RMC:

1. Right-click the tag box "?" and choose **NewTag**.
2. Type a name in the **Name** field.
3. Make sure the **Data Type** is MESSAGE and the **Scope** is at the controller level.
4. Click **Create**.
5. In the MSG instruction, click the ... button. The Message Configuration dialog will open.
6. On the **Configuration** tab, enter the following:
  - a. **Message Type:** PLC-5 Typed Read or PLC-5 Typed Write.
  - b. **Reads:**  
**Source Element:** Enter the address in DF1 format of the first RMC register you want to read.

**Destination Element:** Enter the tag in the ControlLogix or CompactLogix into which you want to read the RMC data. The MSG instruction requires an array. UDT tags are not supported.

c. **Writes:**

**Source Element:** Enter the tag in the ControlLogix or CompactLogix that you want to send to the RMC. The MSG instruction requires an array. UDT tags are not supported.

**Destination Element:** Enter the address in DF1 format of the first RMC register you want to write. See the [Registers Maps](#) for help on addresses.

d. **Number of Elements:** Enter the number of RMC registers to read or write in this field.

7. On the **Communication** tab, enter the **Path** as the RMC name as it appears in the I/O Configuration in the Controller Organizer.

8. Click **OK**.

**Using the MSG Instruction**

The Allen-Bradley MSG instruction may take multiple ladder scans to initialize. Therefore, it is important to enable the MSG block for the correct amount of time. Specifically, the MSG block must be energized until the message control's enable (EN) bit turns on. Following the ladder samples to ensure proper functionality:

**Read or Write Once**

This sample takes care to keep the MSG block energized until the MSG block starts, as indicated by the enable (EN) bit turning on. Once this happens, the application-controlled MSG\_Trigger bit should be turned off to ensure only a single read or write occurs. The message control's Done (DN) or Error (ER) bits can be used to process the results of the transaction.



## 9. PLC Motion Instructions Crossover

This section explains which RMC commands to use in place of common ControlLogix motion instructions. For details on each RMC command, see the RMCTools help:

[https://deltamotion.com/support/webhelp/rmctools/#t=Command\\_Reference%2FCommand\\_List.htm](https://deltamotion.com/support/webhelp/rmctools/#t=Command_Reference%2FCommand_List.htm).

For information on motion other than the common instructions listed here, refer to the [RMCTools help](#), or contact Delta Motion.

### Motion States Overview

The axis motion states in the RMC differ from the Logix 5000. The RMC states are as follows:

#### Disabled Axis State

The Enabled axis status bit indicates the state of the axis.

**RMC75/RMC150:** Motion commands other than Direct Output (9) are not allowed.

**RMC200:** No motion commands are allowed.

The axis can become disabled in the following ways:

- When the axis starts up and the RMC is not configured to start in Run mode.
- Enable/Disable Axis (97) command.
- RMC200 Only:
  - A Disable Axis AutoStop.
  - Faulting the entire controller. Disable all axes and the controller state is disabled.

#### Enabled Axis State

In the Enabled state, when a position command is sent, the axis will automatically change to the state required, whether open-loop or closed-loop.

The axis can be enabled in the following ways:

- Enable/Disable Axis (97) command: enable the axis to which the command was sent.
- Enable Controller (7) command: enables all axes. The RMC200 has an option to not enable all axes when the controller is enabled.
- The controller enters Run mode. Notice that the controller can be configured to start up in Run mode, in which case all axes will be enabled as well.

### Axis Halts

After an axis halts, then normally, any motion command is allowed. If the axis halted due to an AutoStop that was triggered by an Error bit turning on, then as long as the underlying error condition has gone away, the command will be accepted. If the error condition persists, it must be resolved before motion commands will be allowed.

The RMC200 AutoStops have a Disable Axis option. If the RMC200 axis is disabled, it must first be enabled before any motion commands are allowed.

### Run/Program Mode

The Run/Program mode state does not affect axis states. All motion can be performed in either state. The Run/Program refers to user programs. User programs can only run when the controller is in Run mode.

### MASD – Motion Axis Shutdown

The MASD is similar to disabling an axis in the RMC. Notice in the RMC75 and RMC150, the Direct Output command is still allowed in the disabled state.

To disable an axis:

**RMC75/RMC150:** Use the Enable/Disable Axis (97) command

**RMC200:** Use the Enable/Disable Axis (97) command or the Fault Controller (8) command. All axes will be disabled.

### MASR – Motion Axis Shutdown Reset

The MASR is similar to enabling an axis in the RMC. To enable the axes, use the Controller (7) command or the Enable/Disable Axis (97) command, or enter Run mode.

### MAFR – Motion Axis Fault Reset

In the RMC, it is usually not necessary to specifically reset the axis after an error halts the axis, except in the case of a Disable Axis Autostop on the RMC200, in which case the axis must first be enabled. Normally, any motion command will clear all the axis Error bits whose underlying error condition no longer exists, and the motion command will be processed as long as no underlying error conditions exist.

Error bits are latched and may be on even though the underlying error condition no longer exists. To clear error bits without sending a motion command, use the Clear Faults (4) command In the RMC.

**MSO – Motion Servo On**

The MSO instruction enables the servo loop so that it is on in closed-loop control. In the RMC, a command like the MSO is not necessary. As long as the axis is enabled, and no errors conditions exist, any motion command can be issued to put the axis in closed-loop control. To put the axis into closed-loop control and hold the current position, use the Hold Current Position (5) command.

**MSF – Motion Servo Off**

To turn the servo loop off, send the Open Loop Rate (10) or Direct Output (9) command to put the axis in open loop. The Direct Output (9) command puts the axis in a state where it ignores any errors and is recommended to be used only during machine setup.

**MDO – Motion Direct Drive On**

The MDO instruction sets the servo output voltage in open loop. The Open Loop Rate (10) and Direct Output (9) commands are similar to the MDO instruction. Both commands send an open loop voltage or percent output. During normal operation, you would typically use the Open Loop Rate (10) command, because the axis halts will operate normally. The Direct Output (9) command ignores all errors on the axis and should be used during setup only.

**MDF – Motion Direct Drive Off**

The MDF instruction sets the servo output voltage to the output offset voltage. To achieve similar behavior, use the Open Loop Rate (10) or Direct Output (9) command with a zero voltage or percentage output.

**MAM – Motion Axis Move**

Use the following commands based on the type of motion:

Point-to-point absolute move: Move Absolute (20)

Point-to-point relative move: Move Relative (21)

Gearing offset: Phasing (34)

Camming (curve) master offset: you generally need to resend the curve command with the new master offset

**MAS – Motion Axis Stop**

Use the Stop (Closed Loop) (6) command. Notice that this stops the position motion in closed-loop control, regardless of what motion was running, whereas the MAS can specify which type of position motion to stop.



**MAH – Motion Axis Home**

See the **Homing** section below.

**MAJ – Motion Axis Jog**

To jog the axis, you must send the Move Absolute (20) command to start the move, then send the Stop (Closed Loop) (6) command to stop the axis.

**MAG – Motion Axis Gear**

There are many types of gearing commands in the RMC. See the [RMCTools Gearing help topic](#).

**MCCP – Motion Calculate Cam Profile**

Similar to the Curve Add (82) command. Curves in the RMC are different than the Logix 5000. See the [RMCTools Curves Overview help topic](#).

**MAPC and MATC – Motion Axis Position Cam and Motion Axis Time Cam**

Similar to the Curve Start Advanced (88) command. Curves in the RMC are different than the Logix 5000. See the [RMCTools Curves Overview help topic](#).

## 10. Homing

For more details, see:

- [RMCTools Homing help topic](#)

**Note:** The HYD02 and M02AS modules support absolute position feedback types, so homing is normally not necessary. Adjusting the Actual Position can be done with the Set Actual Position (48) or Offset Position (47) commands.

The M02AE encoder feedback is incremental and requires homing. The RMC supports homing for quadrature encoder feedback based on a Home input, the Z input, or a combination thereof.

To home with the RMC:

1. **Arm the Home**

Send the Arm Home (50) command to arm the home. In this command, you will specify the Home Position, Trigger Type, Repeat Mode, and which input to use.

2. **Move the Axis Until it Homes**

You will need to send a motion command to move the axis. For complex homing sequences, you can write a user program to perform the homing. The PLC would send the Start Task (90) command to the RMC to start the user program for the homing.

When the home has occurred, the Home Latched axis status bit will be set and the Home Armed axis status bit will be set. If the **Repeat Mode** parameter was set to **Repeat**, both the Home Latched and the Home Armed status bits will remain set. These status bits can be used in a user program to check that the home has occurred. The Home Input status bit also shows the current state of the home input.

## 11. Registration

For more details, see:

- [RMCTools Registration help topic](#)

The RMC supports high-speed registration for quadrature encoder inputs. For each input, two registration values can be stored.

To perform registration with the RMC:

1. **Arm the Registration**

Send the Arm Registration (52) command to arm the home. In this command, you will specify the Registration Number, Registration Input, and Registration Edge.

2. **Wait for Registration to Occur**

If the registration is armed and a registration trigger occurs, the following is done:

- The corresponding latched axis status bit will be set (**Registration 0 Latched** or **Registration 1 Latched**).
- The corresponding armed status bit will be cleared (**Registration 0 Armed** or **Registration 1 Armed**).
- The latched position is reported in the **Registration 0 Position** or **Registration 1 Position** status register.

## 12. Support

For assistance with converting from the Allen-Bradley analog modules to an RMC, contact Delta Motion at +1 360-254-8688 or [www.deltamotion.com](http://www.deltamotion.com)

## A1. Addendum 1 – Example Code Functional Description

To assist conversion to the RMC, example Studio 5000 Logix Designer code is available on Delta Motion's forum. This addendum explains from a functional view how the example code is intended to be used.

### Example overview

The example code is provided based on the understanding that there will be existing code, previously written to communicate with a 1756-HYD02, 1756-MO2AS or 1756-MO2AE module. The existing code will need to be modified in order to replace the 1756 modules with an RMC.

This fully functional code demonstrates both a typical HYD02 implementation and a fully functional RMC implementation. This makes it easy for the customer to see how a conversion can be done. The code also includes AOI's that allow the new RMC code to follow a similar structure as existing HYD02/MO2As/MO2AE code.

The example code consists of the following program files:

- HYD02\_to\_RMC\_Example.rmcproj RMC Project File
- HYD02 to RMC Example - AOI.ACD PLC Project File

For details on each RMC command used in the example, see the RMCTools help:

[https://deltamotion.com/support/webhelp/rmctools/#t=Command\\_Reference%2FCommand\\_List.htm](https://deltamotion.com/support/webhelp/rmctools/#t=Command_Reference%2FCommand_List.htm).

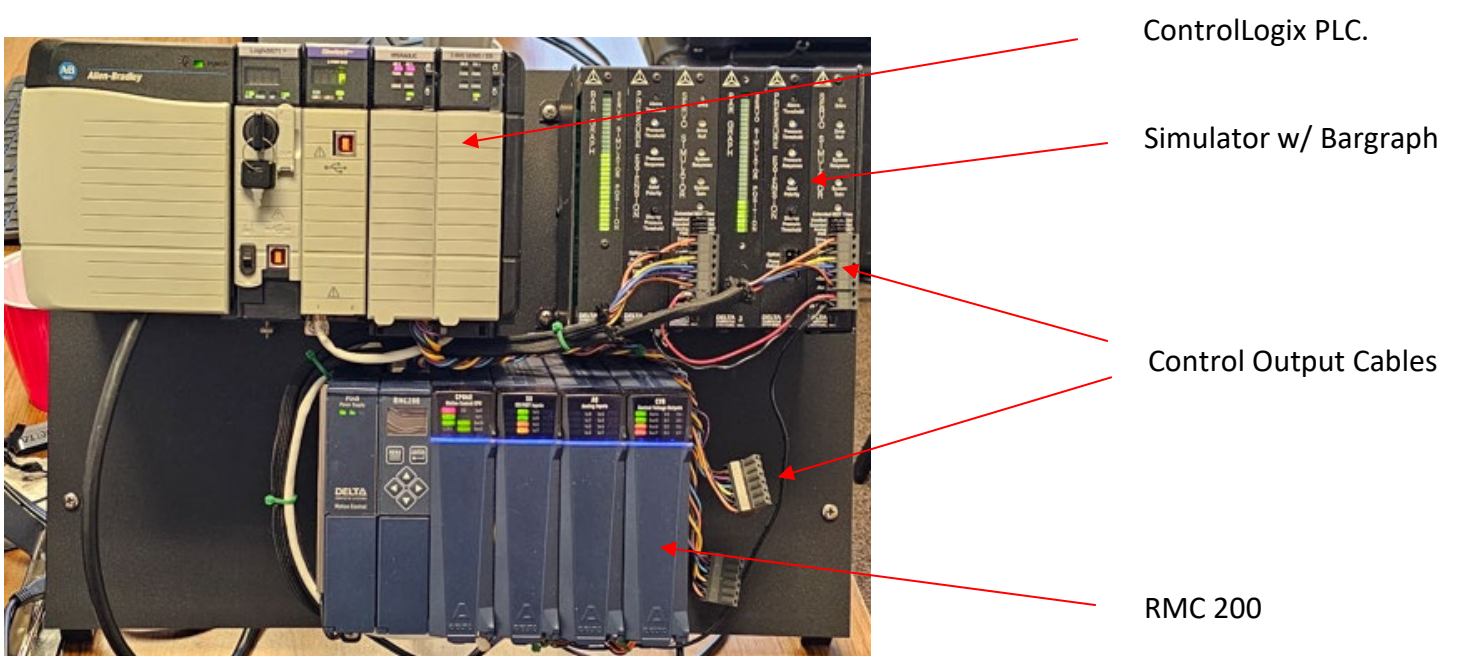
For information on motion other than the common instructions listed here, refer to the [RMCTools help](#), or contact Delta Motion.

## Hardware

The example was created using a functioning demo system. While the user will not have this hardware, it is useful to have an understanding of the system configuration in order to understand the code.

The hardware consisted of the following:

- ControlLogix PLC; 1756-L71 Ver 35.011
- RMC200; R200-B5 R200-PS4D R200-CPU40 R2-KL004 (1) R200-S8 (1) R200-A8 (1) R200-CV8
- Position Simulator with Bar Graph; RMC-SS2-PE2-BG2



This hardware demonstrated that the control output (+/- 10Vdc) from either the PLC HYD02 module or the RMC200 controlled the 2-axis position system simulator in the same manner. The cables from either the PLC or the RMC were connected, depending on which control was selected within the PLC code described below.

## Software

The example code was implemented on the operational demo described above to demonstrate how typical motion instructions for the HYD02 module can be implemented to command an RMC200. For this example, 2 axes of control are created. Additional axes can be applied depending on users' applications. Up to 50 are available on the RMC200.

The logic is written with 2 individual tasks, as shown below.

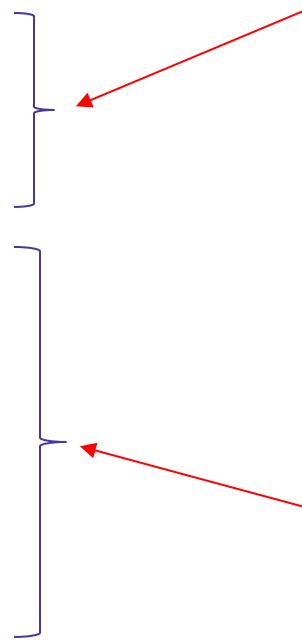
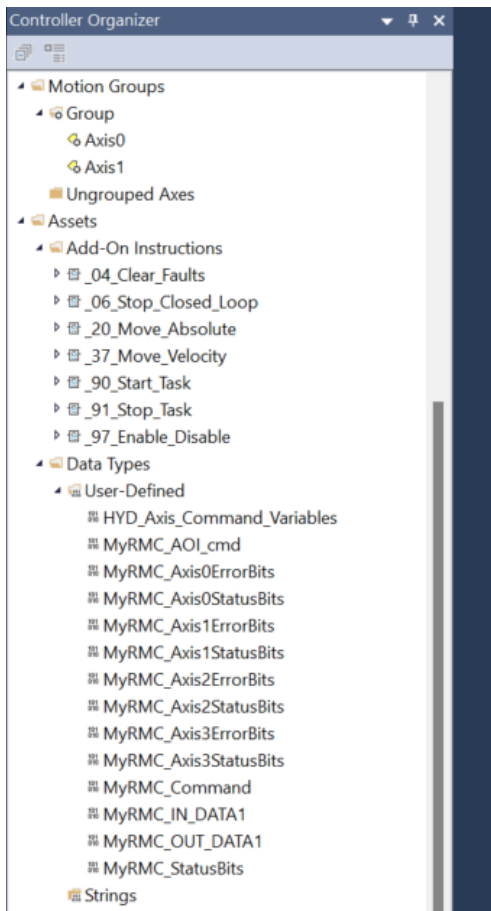


These Tasks represent the existing program that would typically be running in the PLC.

These Tasks are examples that detail the new connection to the RMC. This entire Task is new, and would not exist prior to conversion.

In order to use the RMC, and emulate the motion instructions that are used for the 1756 motion modules, we have created some basic AOI's (Add on Instructions). These are intended to be examples and can be modified and copied for individual applications. If additional AOI's are desired, copy one of them, give it a new name, then modify the logic to give the desired result. Typically, this would consist of changing the rung that defines the condition for the "done" bit (DN).

Along with the AOI's, there are new User Defined Data Types (UDT's). These provide an easy way to associate and manage the axes by their designated Axis names, or naming conventions. Again, like the AOI's these are examples and can be configured to meet the criteria of individual applications.



The AOI's are created here, and the naming convention ties the AOI to its corresponding motion instruction within the RMC.

These UDT's help to translate the data types between the PLC, and the RMC.

## 1756-HYD02 Example Logic - PLC

The example logic is written to simulate an operator controlling the Servo axes using either buttons or an HMI. For this demo, a user defined tag structure has been created, which can be manipulated rather than a real-world button or HMI. To use the tag, open up the Controller Tags and find the tag named “HYD02AxisCMDs”.

HYD02AxisCMDs	{...}	{...}	HYD_Axis_Corn
HYD02AxisCMDs.RMCSelectCMD	0	Decimal	BOOL
HYD02AxisCMDs.ServosOnCMD	0	Decimal	BOOL
HYD02AxisCMDs.AutoModeCMD	1	Decimal	BOOL
HYD02AxisCMDs.MotionStartCMD	0	Decimal	BOOL
HYD02AxisCMDs.MotionStopCMD	0	Decimal	BOOL
HYD02AxisCMDs.JogPlusCMD	0	Decimal	BOOL
HYD02AxisCMDs.JogMinusCMD	0	Decimal	BOOL
HYD02AxisCMDs.ShutdownCMD	0	Decimal	BOOL
HYD02AxisCMDs.Axis_FaultRstCMD	0	Decimal	BOOL
HYD02AxisCMDs.StartTaskCMD	0	Decimal	BOOL
HYD02AxisCMDs.StopTaskCMD	0	Decimal	BOOL
HYD02AxisCMDs.RMCProgramNo	0.0	Float	REAL
HYD02AxisCMDs.ServosON	0	Decimal	BOOL
HYD02AxisCMDs.AutoMode	1	Decimal	BOOL
HYD02AxisCMDs.ManualMode	0	Decimal	BOOL
HYD02AxisCMDs.Axis0POS1	14.0	Float	REAL
HYD02AxisCMDs.Axis0POS2	34.0	Float	REAL
HYD02AxisCMDs.Axis1POS1	34.0	Float	REAL
HYD02AxisCMDs.Axis1POS2	14.0	Float	REAL
HYD02AxisCMDs.Axis0JogPlusPOS	35.9	Float	REAL
HYD02AxisCMDs.Axis0JogMinusPOS	0.1	Float	REAL
HYD02AxisCMDs.Axis1JogPlusPOS	35.9	Float	REAL
HYD02AxisCMDs.Axis1JogMinusPOS	0.1	Float	REAL
HYD02AxisCMDs.AxesDelayTMR	{...}	{...}	TIMER[10]
HYD02AxisCMDs.State	0	Decimal	DINT
HYD02AxisCMDs.ONS	{...}	{...}	Decimal BOOL[32]

The tag is configured so that the most used commands are at the top, with the status and setup parameters located near the bottom. The commands can be identified by the suffix CMD at the end of the tag name, i.e., “HYD02AxisCMDs.ServosOn**CMD**”.

The code was first written with only the 1756-HYD02 module in use. It is operational code in a demonstration environment providing some basic motion commands for the HYD02. This would be the case, if an existing program has been in operation using the HYD02 module.

As an example, for how one might replace the HYD02 module with an RMC, code was later added to show how the same HYD02 functions can be redirected from the existing code, to the RMC. This is an example only,



it will be up to the user to modify the code for each specific application. A tag has been added to select between either the HYD02 or the RMC. Let's start with the PLC selected.

For this example, make sure the Boolean bit “.RMCSelectCMD” is set to a 0. This will allow the PLC to control the HYD02 module, not the RMC200.

HYD02AxisCMDs	{...}
HYD02AxisCMDs.RMCSelectCMD	0
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit LO

### Initialize

The code demonstrates a typical sequence for powering up a servo system:

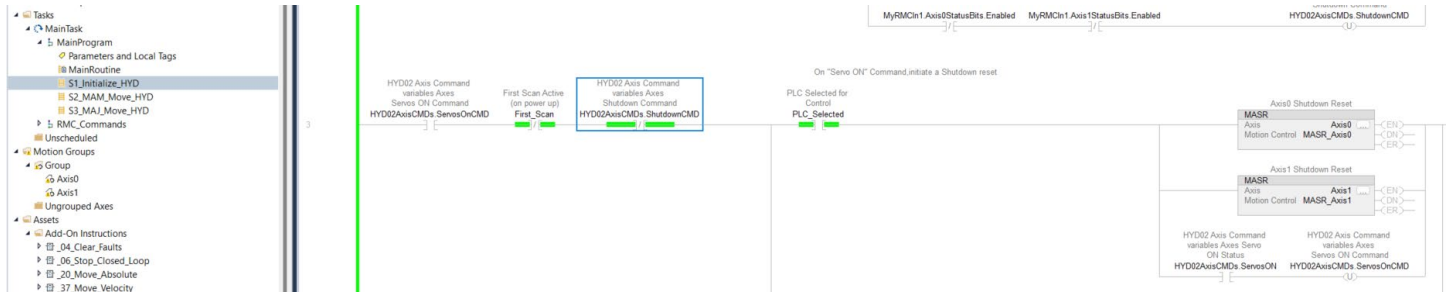
1. Turn power on for the system (MASR)
2. Attempt to clear alarms and faults (MAFR)
3. Enable the servo axes (MSO)

Using the control tag, set the “.ServosOnCMD “ bit to a 1, to turn the Servo's on.

HYD02AxisCMDs	{...}
HYD02AxisCMDs.RMCSelectCMD	0
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

This will execute a simple routine within the MainProgram (S1\_Initialize\_HYD, starting at rung 3) that clears faults and turns the Motion Servo On for each axis. This command is a one shot, so it will go low as soon as the command has been received.



Reviewing the code, you will see the commands MASR, then MAFR, and finally MSO.

You will see the corresponding status bit “.ServosON” set to 1 that indicates both axes are “ON”.

Now that the axes are initialized and enabled, you will be able to command servo Motion.

**Motion Axis Move (MAM)**

This demo has been programmed to control 2 axes, in a simple coordinated manner. The 2 axes will utilize the MAM instructions to move between the 2 programmed positions with in the command array. To achieve this, a routine within the MainProgram (S2\_MAM\_Move\_HYD) will command Motion for each axis.

Using the control tag, set the “.AutoModeCMD “ bit to a 1, to enable Auto Mode.

HYD02AxisCMDs	{...}
HYD02AxisCMDs.RMCSelectCMD	0
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

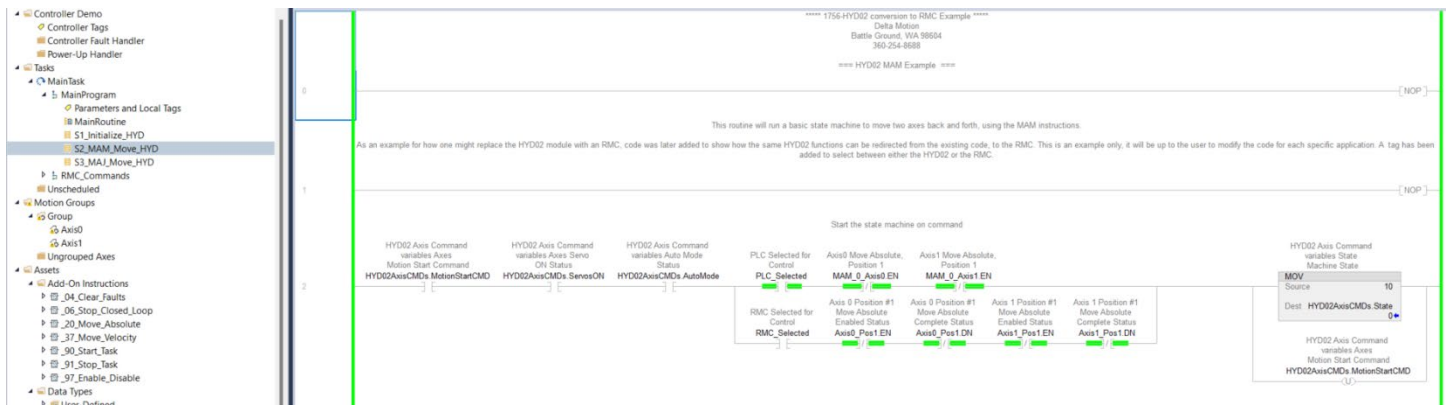
This will Enable the Auto mode within the PLC, allowing Auto functions to be performed, such as programmed motion profiles, etc.

Using the control tag, set the “.MotionStartCMD “ bit to a 1, to execute the motion profiles that are programmed.

HYD02AxisCMDs	{...}
HYD02AxisCMDs.RMCSelectCMD	0
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

This will command the Auto Mode motion within the MainProgram (S2\_MAM\_Move\_HYD), starting at rung 2). This will start a basic state machine that moves 2 axes back and forth. This command is a one shot, so it will go low as soon as the command has been received. The motion will continue until commanded to stop, using the “.MotionStopCMD” bit.

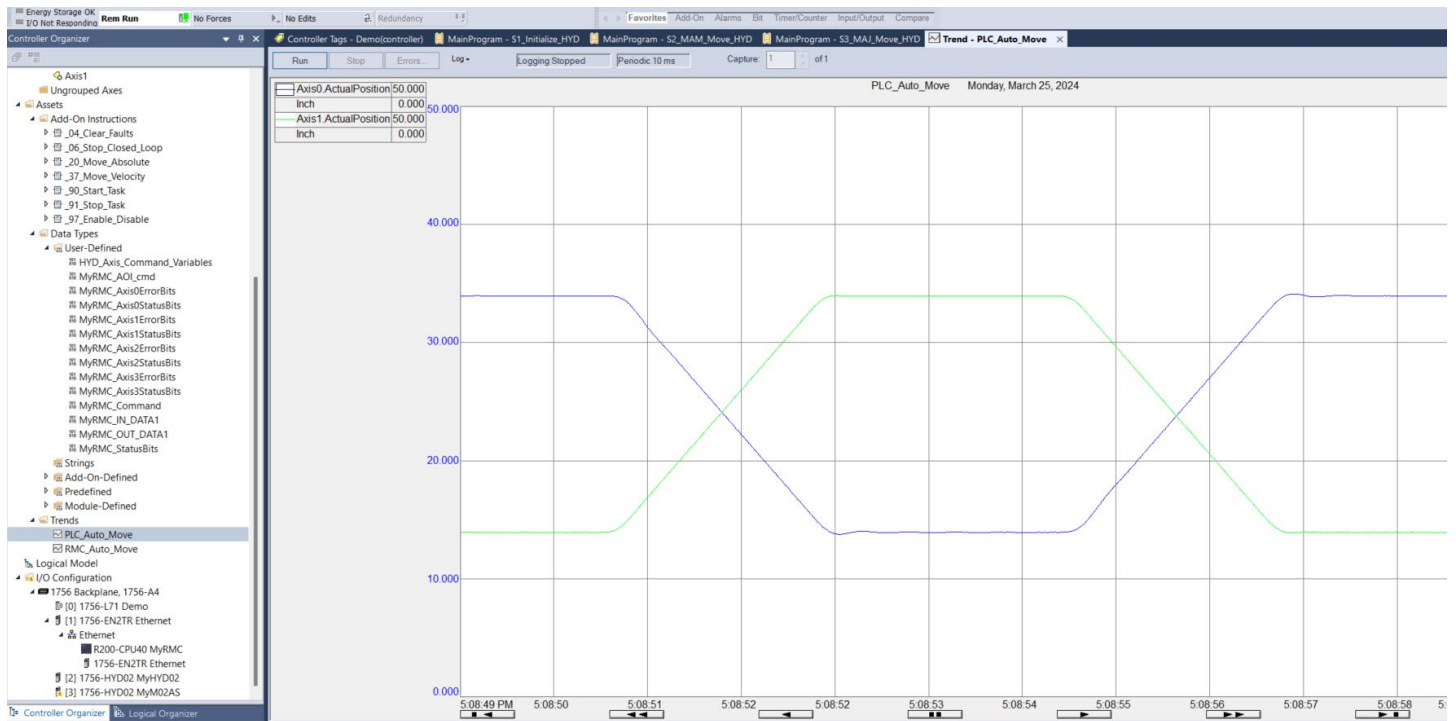


Reviewing the code, you will see the state machine uses MAM commands to move the axes between 2 positions, with 2 seconds delay between each move.

As you can see the 2 axes are moving opposite of each other between 14” – 34”, and 34” – 14” respectively.

HYD02AxisCMDs.Axis0POS1	14.0	Float	REAL
HYD02AxisCMDs.Axis0POS2	34.0	Float	REAL
HYD02AxisCMDs.Axis1POS1	34.0	Float	REAL
HYD02AxisCMDs.Axis1POS2	14.0	Float	REAL

You can monitor the positional moves using the Trend feature. Open up the Trend “PLC\_Auto\_Move” to see the axes movements.



### Motion Axis Jog (MAJ)

The demo has been programmed to allow jogging for each of the axes. The 2 axes will utilize the MAJ instructions to jog either plus or minus. A routine within the MainProgram (S2\_MAJ\_Move\_HYD) will command Motion for each axis.

Using the control tag, set the “.AutoModeCMD “ bit to a 0, to enable Manual Mode.

HYD02AxisCMDs		{...}
HYD02AxisCMDs.RMCSelectCMD		0
HYD02AxisCMDs.ServosOnCMD		0
HYD02AxisCMDs.AutoModeCMD		0
HYD02AxisCMDs.MotionStartCMD		0
HYD02AxisCMDs.MotionStopCMD		0
HYD02AxisCMDs.JogPlusCMD		0
HYD02AxisCMDs.JogMinusCMD		0
HYD02AxisCMDs.ShutdownCMD		0
HYD02AxisCMDs.Axis_FaultRstCMD		0
HYD02AxisCMDs.StartTaskCMD		0
HYD02AxisCMDs.StopTaskCMD		0

Set this bit LO

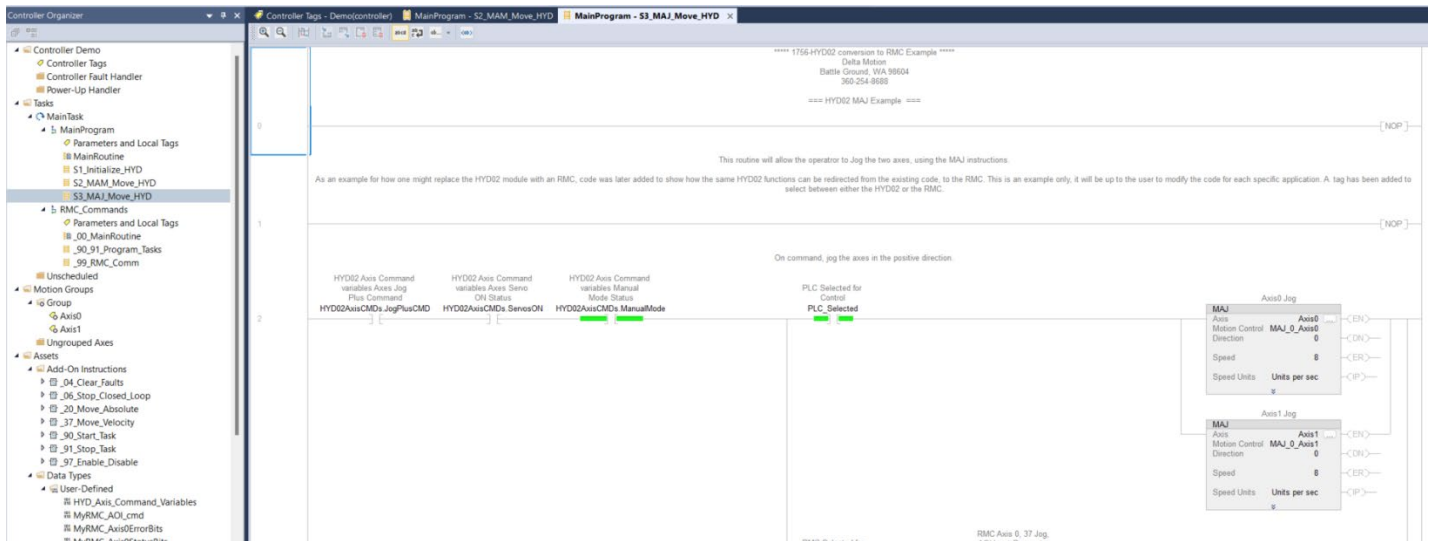
This will Enable the Manual mode within the PLC, allowing Manual functions to be performed, such as Homing, Jogging, Calibration, etc.

Using the control tag, set the “.JogPlusCMD “ bit to a 1, to move the axes in the positive direction. Setting it to a zero will stop motion. Setting the “.JogMinusCMD “ bit to a 1, to move the axes in the negative direction.

HYD02AxisCMDs	{...}
HYD02AxisCMDs.RMCSelectCMD	0
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

This command is maintained, simulating holding a jog button. You will have to set to 0 to stop the motion. When the PLC sees that there is no “button” pressed for a jog, the Motion Axis Stop (MAS) command will be sent stopping the motion.



## 1756-HYD02 Example Logic - RMC

For RMC portion of the example code demonstrates how the PLC sends commands to the RMC200 rather than using the PLC motion commands for the HYD02 module. Additional logic has been created to emulate the PLC commands, and AOI's have been created that will give a similar look and feel for the same functionality.

For this example, make sure the Boolean bit “.RMCSelectCMD” is set to a 1. This will allow the PLC to control the RMC200, not the HYD02 module.

HYD02AxisCMDs	[...]
HYD02AxisCMDs.RMCSelectCMD	1
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

### Initialize

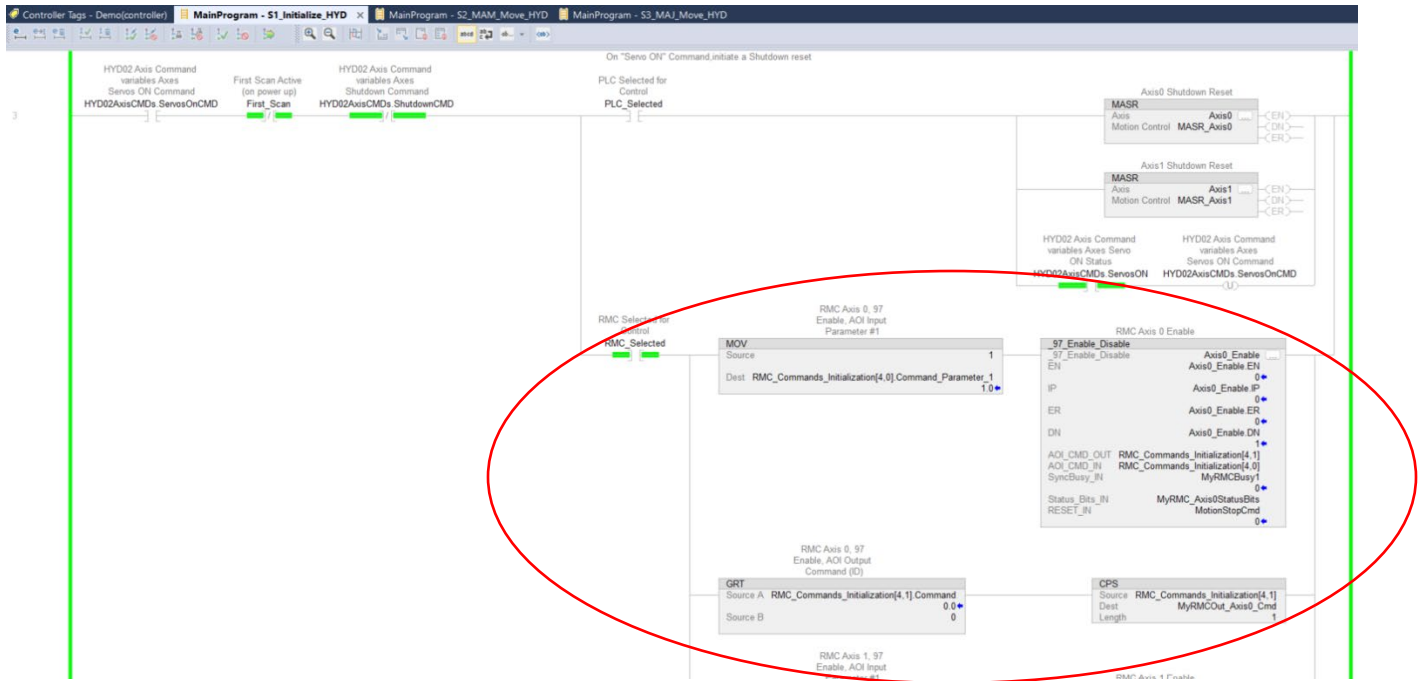
As in the example above for the HYD02, the first thing one would do is to turn power on for the system, either an electric servo drive, or an HPU for hydraulics. During a power up phase, the system would attempt to clear alarms and faults, then enable the servo axes.

Using the control tag, set the “.ServosOnCMD “ bit to a 1, to turn the Servo's on.

HYD02AxisCMDs	[...]
HYD02AxisCMDs.RMCSelectCMD	1
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

This will execute a simple routine within the MainProgram (S1\_Initialize\_HYD, starting at rung 3) that clears faults and turns the Motion Servo On for each axis. This command is a one shot, so it will go low as soon as the command has been received.



Reviewing the code, you will see the **NEW AOI** commands `_97_Enable_Disable`, then `_04_Clear_Faults`, and finally `_06_Stop_Closed_Loop`. Notice the additional branches of logic in each rung that use these. See the description for the AOI's below (Add on Instructions (AOI's)).

You will again see the corresponding status bit `".ServosON"` set to 1 that indicates both axes are "ON".

Now that the axes are initialized and enabled, you will be able to command servo Motion from the RMC.

**Motion Axis Move (MAM)**

As in the example above for the HYD02, the 2 axes will be controlled in the same simple coordinated manner. However; the 2 axes will now use the new AOI instructions for the RMC, to move between the 2 programmed positions within the command array.

Using the control tag, set the `".AutoModeCMD "` bit to a 1, to enable Auto Mode.

HYD02AxisCMDs	{...}
HYD02AxisCMDs.RMCSelectCMD	1
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	1
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

This will Enable the Auto mode within the PLC, allowing Auto functions to be performed, such as programmed motion profiles, etc.

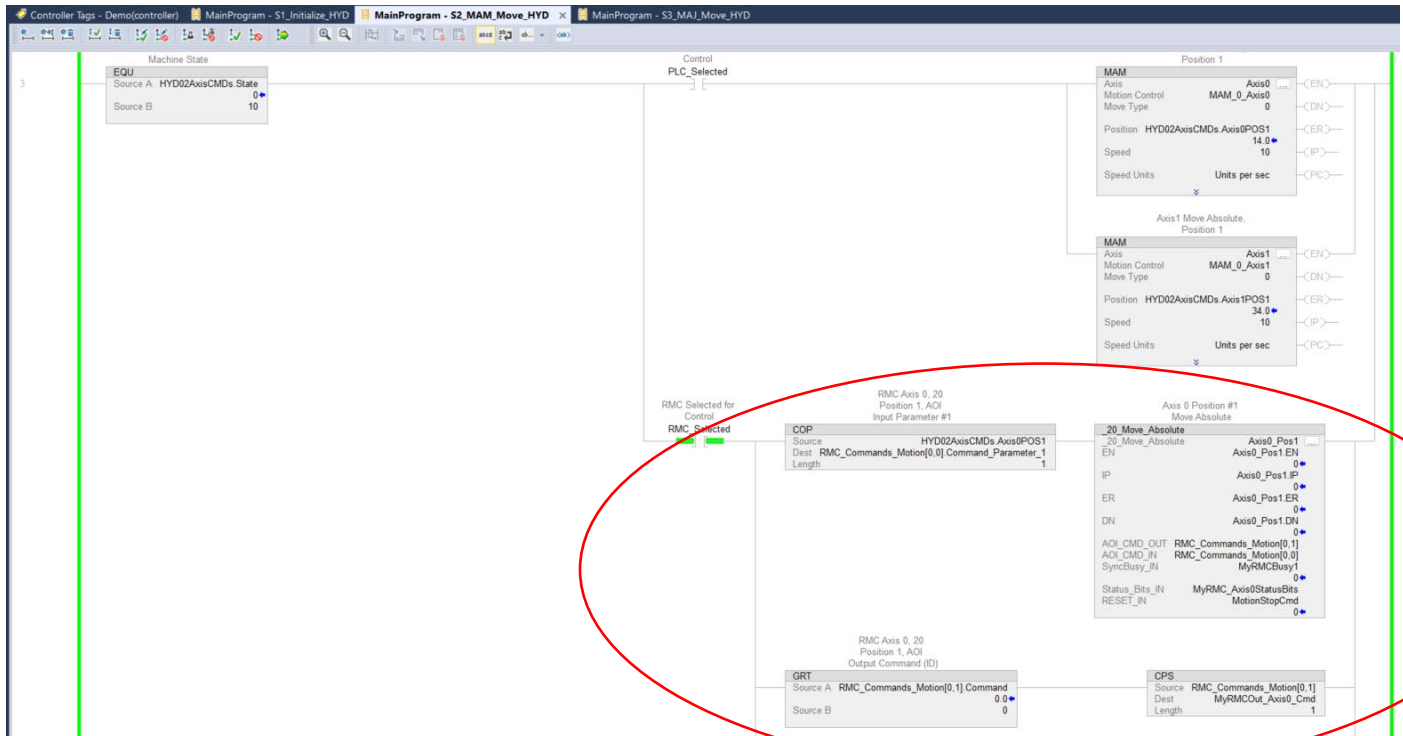
As before, use the control tag to set the “.MotionStartCMD “ bit to a 1, to execute the motion profiles that are programmed.

HYD02AxisCMDs	{...}
HYD02AxisCMDs.RMCSelectCMD	1
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	1
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

This will command the Auto Mode motion within the MainProgram (S2\_MAM\_Move\_HYD), starting at rung 2). This will start the same basic state machine as before, that moves 2 axes back and forth. This command is a one shot, so it will go low as soon as the command has been received. The motion will continue until commanded to stop, using the “.MotionStopCMD” bit.



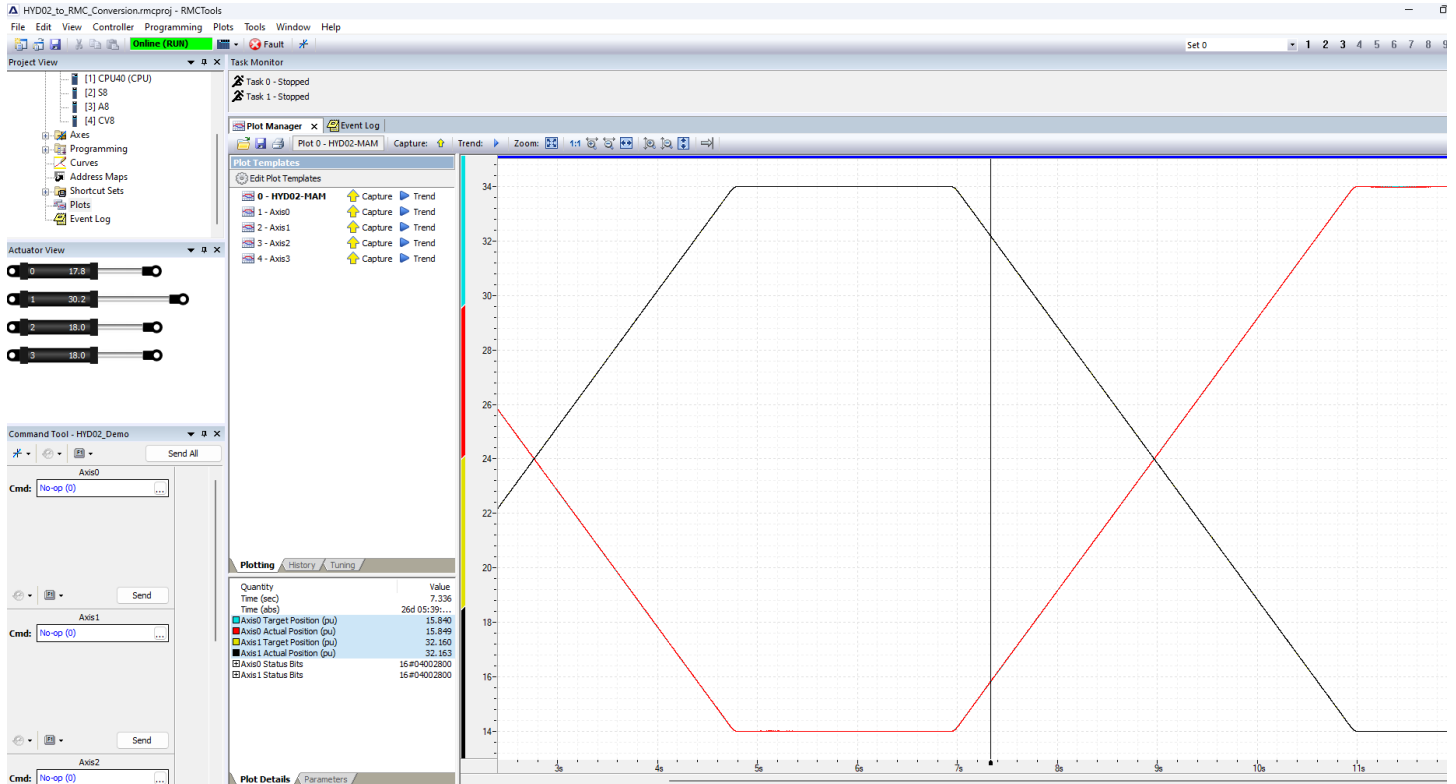


Reviewing the code, you will now see the state machine uses the new AOI instructions to move the axes between 2 positions, with 2 seconds delay between each move.

As you can see the 2 axes are still moving opposite of each other between 14" – 34", and 34" – 14" respectively.

HYD02AxisCMDs.Axis0POS1	14.0	Float	REAL
HYD02AxisCMDs.Axis0POS2	34.0	Float	REAL
HYD02AxisCMDs.Axis1POS1	34.0	Float	REAL
HYD02AxisCMDs.Axis1POS2	14.0	Float	REAL

You can monitor the positional moves using the Plot Manager within RMCTools. Open up the Plot "0-HYD02-MAM" to see the axes movements.



### Motion Axis Jog (MAJ)

New AOI instructions have been added for jogging the axes as well. The same routine within the MainProgram (S2\_MAJ\_Move\_HYD) will command the jogging motion for each axis as before; however the RMC will now control them.

Using the control tag, set the “.AutoModeCMD “ bit to a 0, to enable Manual Mode.

HYD02AxisCMDs	(...)
HYD02AxisCMDs.RMCSelectCMD	1
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit LO

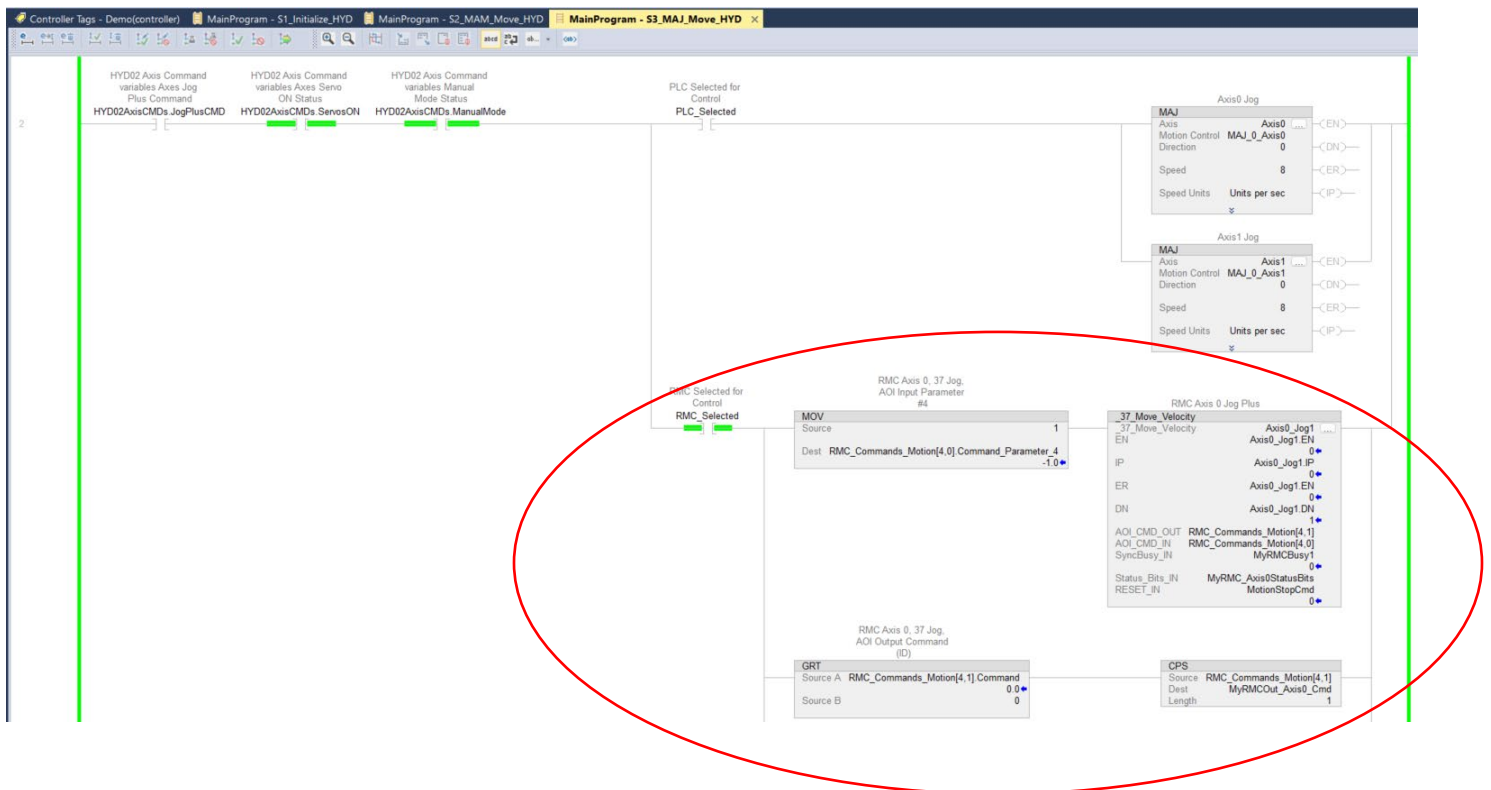
This will Enable the Manual mode within the PLC, allowing Manual functions to be performed, such as Homing, Jogging, Calibration, etc.

Using the control tag, set the “.JogPlusCMD “ bit to a 1, to move the axes in the positive direction. Setting it to a zero will stop motion. Setting the “.JogMinusCMD “ bit to a 1, to move the axes in the negative direction.

HYD02AxisCMDs	(...)
HYD02AxisCMDs.RMCSelectCMD	1
HYD02AxisCMDs.ServosOnCMD	0
HYD02AxisCMDs.AutoModeCMD	0
HYD02AxisCMDs.MotionStartCMD	0
HYD02AxisCMDs.MotionStopCMD	0
HYD02AxisCMDs.JogPlusCMD	0
HYD02AxisCMDs.JogMinusCMD	0
HYD02AxisCMDs.ShutdownCMD	0
HYD02AxisCMDs.Axis_FaultRstCMD	0
HYD02AxisCMDs.StartTaskCMD	0
HYD02AxisCMDs.StopTaskCMD	0

Set this bit HI

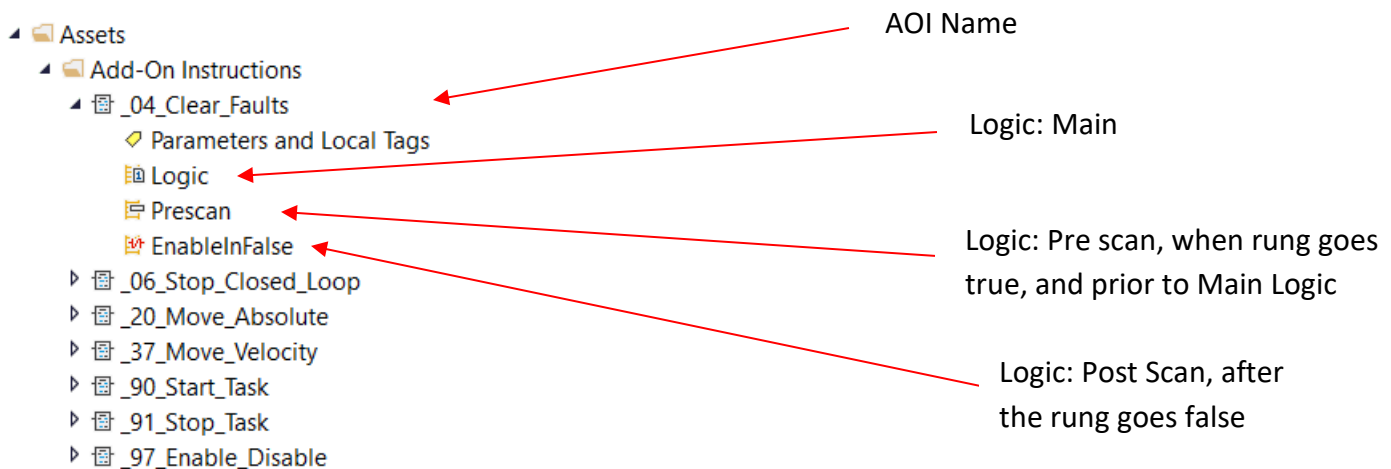
This command is maintained, simulating holding a jog button. You will have to set to 0 to stop the motion. When the PLC sees that there is no “button” pressed for a jog, the “Stop Closed Loop” (\_O6\_Stop\_Closed\_Loop) command will be sent to the RMC stopping the motion.



## Add on Instructions (AOI's)

As stated above, in order to use the RMC, and emulate the motion instructions that are used for the 1756 motion modules, some basic AOI's (Add on Instructions) have been created. These are intended to be examples and can be modified and copied for individual applications. If additional AOI's are desired, copy one of them, give it a new name, then modify the logic to give the desired result. Typically, this would consist of changing the rung that defines the condition for the "done" bit (DN).

The following AOI's have been created as examples:



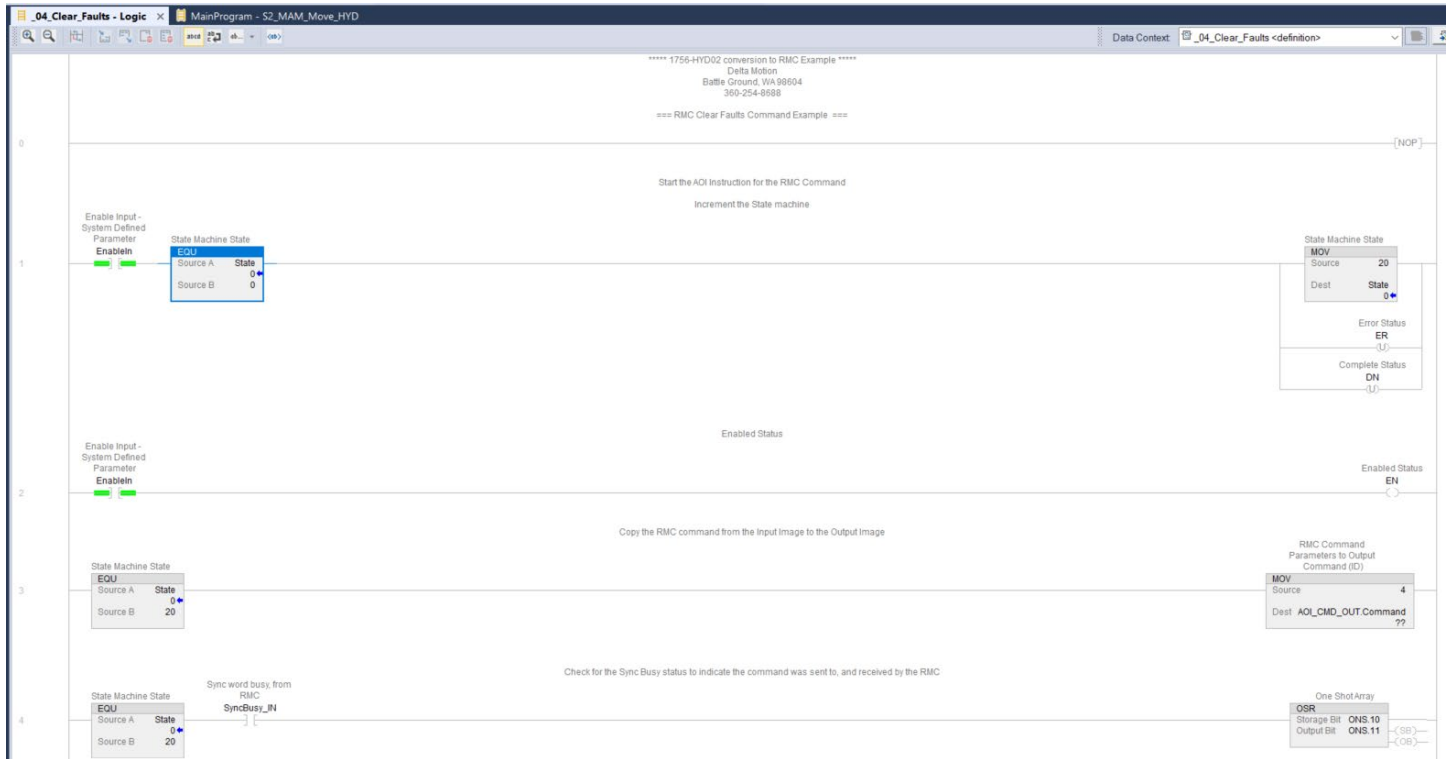
Each AOI executes as a state machine. The intent is to emulate the motion instructions the PLC used, so the following status bits have been created:

- EN: Enable status Instruction has been called
- IP: In Process status Instruction currently executing
- DN: Done Status Instruction completed successfully
- ER: Error status An error has occurred

The design is intended to be generic in nature, so the basic code can be reused between all instructions that the RMC is capable of receiving.

The AOI is written in ladder logic, with the bulk of the functionality residing in the "Logic" folder. This is the logic that is scanned while the rung has the instruction enabled (true). If needed, use the "Prescan" or the "EnableInFalse" routines to add the desired control response.

"Double click" the Logic folder under the AOI name to open up the ladder logic.



Once open you will be able to modify to meet the specific application requirements for your program.

For more detailed information regarding AOI's, refer to Rockwell's knowledgebase and support documentation.

### AOI's and PLC Comms

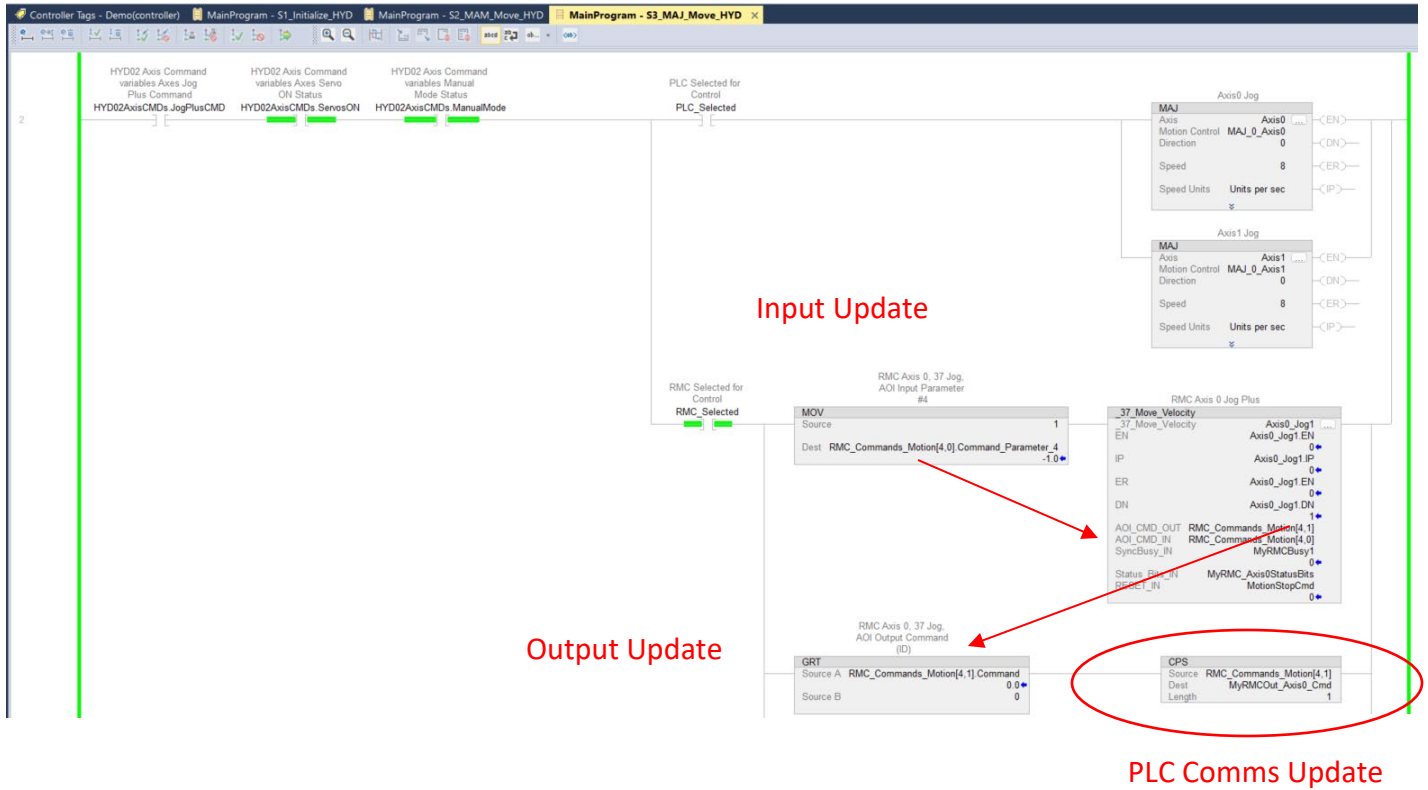
The AOI instructions rely on the same communications method to the PLC as described above in section 3. Communications Overview, employing the cyclic I/O with the Sync register. It is necessary to create additional UDT tags that allow the AOI's to interface with the existing comm tags.

Additionally, as seen in the individual rungs, the AOI's have both an Input and Output tag structure for the instruction. You can pass variables to the instruction through the Input, and then use the result from the instruction using the Output variables. The Input and Output variables use the same basic array structure that the RMC commands use. Here is the basic command array that is native to the RMC for commands:

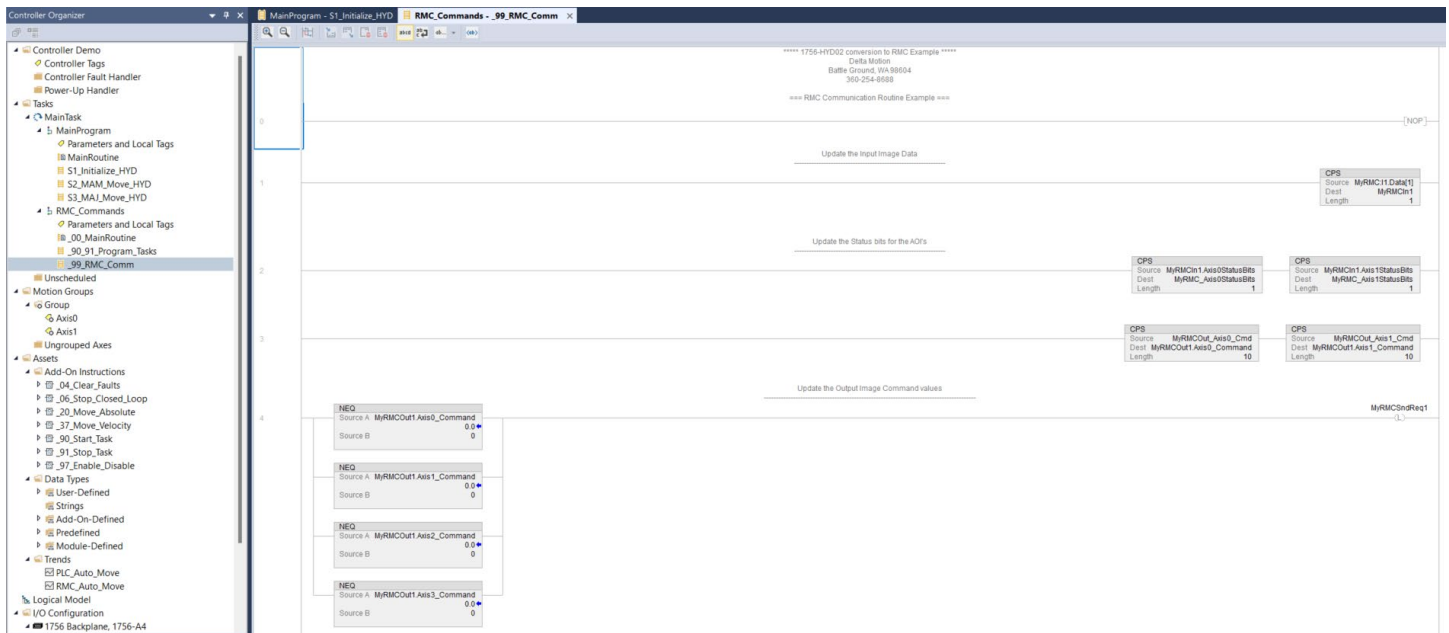
▾ RMC_Commands_Motion	{...}	{...}	MyRMC_AOI_cm
▾ RMC_Commands_Motion[0,0]	{...}	{...}	MyRMC_AOI_cm
RMC_Commands_Motion[0,0].Command	20.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_1	14.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_2	5.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_3	50.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_4	50.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_5	0.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_6	0.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_7	0.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_8	0.0	Float	REAL
RMC_Commands_Motion[0,0].Command_Parameter_9	0.0	Float	REAL
▾ RMC_Commands_Motion[0,1]	{...}	{...}	MyRMC_AOI_cm
RMC_Commands_Motion[0,1].Command	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_1	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_2	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_3	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_4	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_5	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_6	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_7	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_8	0.0	Float	REAL
RMC_Commands_Motion[0,1].Command_Parameter_9	0.0	Float	REAL

A UDT array was created for this example to move into the Input for the AOI's. For example, it uses (0,0) for the input to the AOI and (0,1) for the output of the AOI. Subsequent AOIs would use (1,0) and (1,1), (2,0) and (2,1), etc.

The input parameters can be either hardcoded, or modified prior to the AOI. This example uses each method, depending on the command.



When the AOI updates the Output command variables, they will then be moved into an array that will work with the RMC communication routine previously described.



These additional rungs of code were created to update the communication routine, with each AOI instruction triggering an RMC command request. This will update the Output image table for the Cyclic I/O as described above.