# Advanced Servo Motion Mastery

plcsupport@onlineplcsupport.com

Section: 1 — 1 / 1

**Introduction**

- 1. Introduction — 07:42

Section: 2 — 0 / 1

**Advanced Servo Motion Mastery Downloads Conversion Process**

- 2. File Conversion — 02:22

Section: 3 — 0 / 5

**Emulation Module**

- 3. Understanding Emulator — 06:12
- 4. Emulation From Scratch — 03:40
  - Servo_Motion_Mastery_101_V20Emulated.ACD.txt
- 5. Studio 5000 Emulation — 05:48
  - Servo_Motion_Mastery_101_V28Emulated.ACD.txt
- 6. Emulating Gear Change Logic — 05:25
- Quiz 1: Emulation Quiz

Section: 4 — 0 / 4

**Understanding Servo Rings**

- 7. Troubleshooting a Servo Ring — 16:11
- 8. Servo Motion Mastery Inhibiting An Axis — 13:49
- 9. Servo Motion Mastery Inhibiting A Kinetix 6000 Drive — 08:01
- 10. Servo Motion Mastery Motor Removal TroubleShooting — 15:16

Section: 5 — 0 / 2

**Understanding System Timing and Why**

- 11. Course Rate Update Timing or Better Known as (CUP) — 14:34
- 12. Periodic Task vs Continuous Task — 19:48

Section: 6 — 0 / 4

**Servo Testing and Tuning**

- 13. Servo Commissioning Test — 07:09
- 14. Servo Motion Mastery Trending A Servo HookUp Test — 11:22
- 15. Quick Start to Servo Tuning — 34:53
- 16. Tuning an S-Curve Movement from an Instruction — 21:35

Section: 7 — 0 / 2

**Servo Accel and Decel Jerks**

- 17. Servo Accel Explained — 14:02
- 18. Servo S-Curve Jerks Explained — 09:45

Section: 8 — 0 / 2

**Intro to Servo Conversion Constants**

- 19. Using the Conversion Constants Calculate Tool — 21:21
- 20. Understand a Simple Mechanical Drive System and Convert — 10:46

Section: 9 — 0 / 3

**Understanding Servo Cams**

- 21. The Difference Between Linear Cams and Cubic Cams — 17:57
- 22. Using a Servo Cam Editor — 25:09
- 23. Final Summary of Servo Cams — 06:34
  - ServoMastery_SimpleMATC_Emulated.ACD.txt

Section: 10 — 0 / 5

**Servo Motion Mastery MATC Module**

- 24. Motion Axis Time Cam Instruction Intro — 27:21
  - MATC_Intro.ACD.txt
- 25. Breakdown the Difference Between a Immediate and Pending MATC Use — 25:13
  - TimingCAMs.ACD.txt
- 26. MATC Logic Used With a Virtual Axis With TroubleShooting — 16:02
  - TimingCAMsEmulation.ACD.txt
- 27. MATC Scaling Explained To Limit Confusion — 15:13
- 28. MATC Instruction With A Natural Move Command — 12:40
  - SimpleMATC_WithMove.ACD.txt

Section: 11 0 / 9

## Servo Motion Mastery PCAM Module

- 29. Servo Motion Mastery PCAM Intro 10:11
- 30. Servo Mastery PCAM Explained In Detail 15:59
  - PCAM_Intro_Emulated.ACD.txt
- 31. Building a PCAM From Scratch 01:15:05
  - ServoAxis_PositionCam.ACD.txt
- 32. MAPC Common Issue With Master Reference 11:30
  - PCAM_Intro.ACD.txt
- 33. MAPC PC Bit explained 08:09
- 34. MAPC Adding Common Health Checks 22:11
  - PCAM_Intro_WithHealthChecks.ACD.txt
- 35. Use scaling in an MAPC instruction 16:13
- 36. MAPC Cam Lock Position Function Talk - Not program and Show 05:25
- 37. MAPC Master Lock Position Function - Not program and Show 17:41

Section: 12 0 / 2

## Difference in a Position System And A Time System

- 38. Difference In A MAPC and a MATC 10:09
- Quiz 2: MATC or MAPC

Section: 13 0 / 6

## MAPC Cam Blending - Hardest Task for Most people

- 39. Starting from sctratch 13:02
- 40. Setting up the Physical Axis 18:17
- 41. Adding elements for the state controls 09:19
- 42. Adding our first MAPC 19:22
- 43. Adding the second MAPC for Cam blending 15:23
- 44. Showing a more in-depth view of the Cams blended with added features 23:30

Section: 14 0 / 3

## Motion Cam Calculate Profile Instruction or MCCP Instruction Module

- 45. Using an MCCP Instruction 12:40
- 46. MCCP Instruction - Using Cam Elements 15:02
- 47. Using an MCCP then loading it into an MAPC instruction for use 15:29
  - Servo_Motion_Mastery_101_IOVersion.ACD.txt

Section: 15 0 / 8

## Servo Motion Mastery MAOC Module

- 48. MAOC Introduction 13:39
- 49. MAOC Cam Explained 04:55
- 50. MAOC Cam Output Control 09:32
- 51. MAOC Compensation Usage Explained 14:21
- 52. MAOC Input Enable Feature to Disarm or Arm Ouputs 18:04
- 53. MAOC Axis Setup - Avoid an ERR 35 code 14:57
  - MAOC_ServoMastery_Emulation.ACD.txt
  - MAOC_ServoMastery_EmulationV30.ACD.txt
  - MAOC_ServoMastery_EmulationV28.ACD.txt
  - MAOC_ServoMastery_EmulationLinearV28.ACD.txt
  - MAOC_ServoMastery_EmulationRotaryV28.ACD.txt
- 54. MAOC Cam Used In a Linear Simple Application 08:08
  - MAOC_ServoMastery_EmulationLinearV28.ACD.txt
- 55. MAOC Code Explained 09:10
  - MAOC_ServoMastery.ACD.txt

Section: 16 0 / 1

## Understanding Velocity StandStill

- 56. Using a Axis StandStill Bit 12:26

Section: 17 0 / 4

## Coordinated Motion Section Cartesian Module

- 57. Intro to a Coordinated Motion System 09:52
- 58. Coordinated Motion Section MCCM Instruction Example 16:57
- 59. Coordinated Motion Section MCLM Instruction Example 16:28
- 60. Coordinated Motion Section Cartesian System Logic Explained 46:53
  - Coord_Motion_Blend_Circle_Diamond_Square.ACD.txt

Section: 18 0 / 1

## Closing Comments....Intro to Advanced Servo Mastery 2

- 61. Closing Comments 07:38

# Contents

## Lecture 1: Introduction

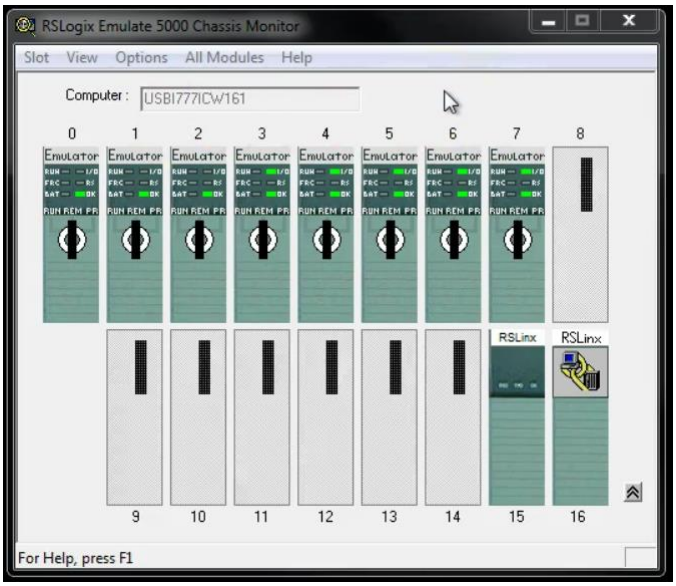
## Lecture 2: File Conversion

Simply rename the .txt file .ACD.  Do not open or edit at all while it is a .txt.  If you even open it in notepad it will be ruined, redownload.

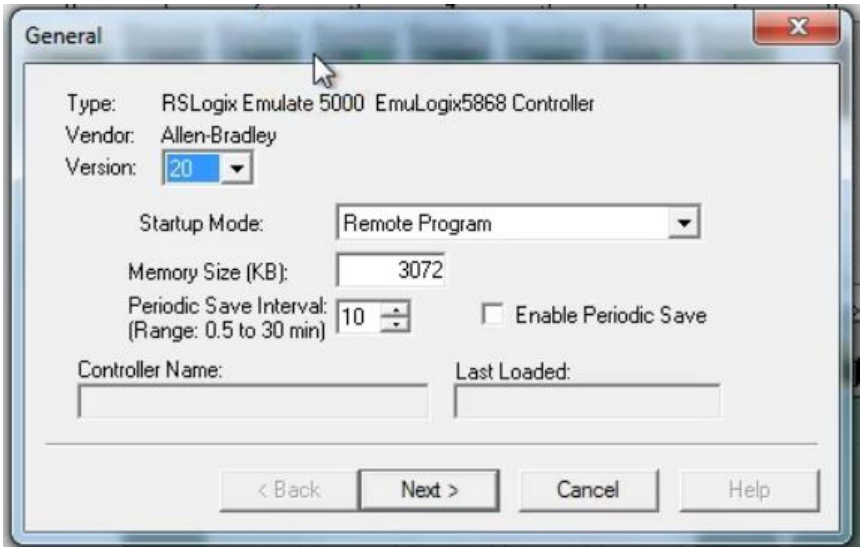
# Lecture 3: Understanding Emulator

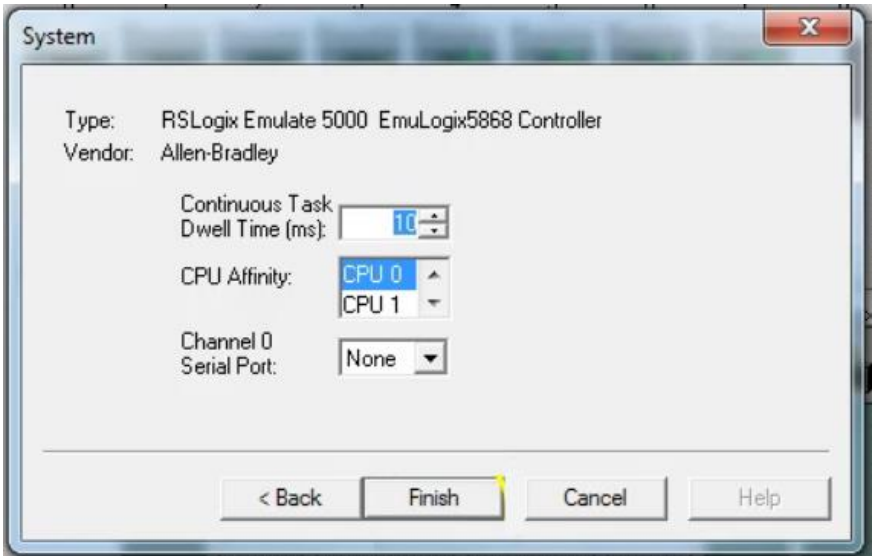
The version of the emulator must match the firmware version of your Logix Designer project.

To add a 'board' right click an open rack position and select create. Then select the board from the list of options. In this case we select 'Controller'.



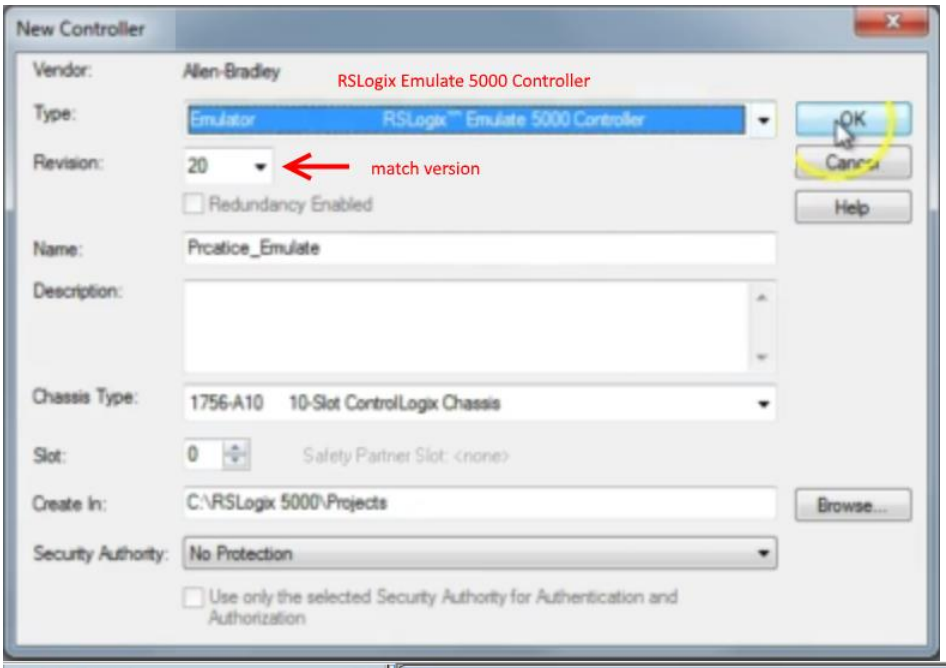




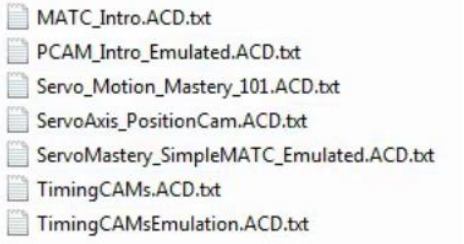



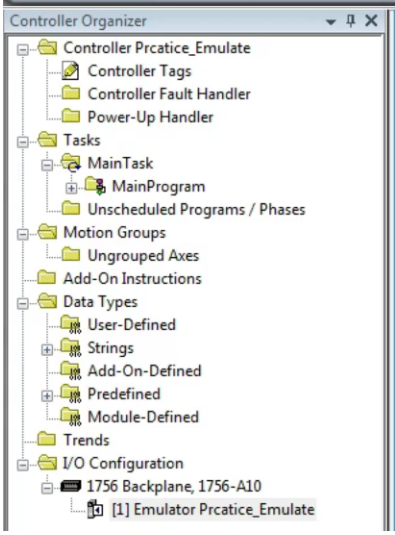
# Lecture 4: Emulation from Scratch

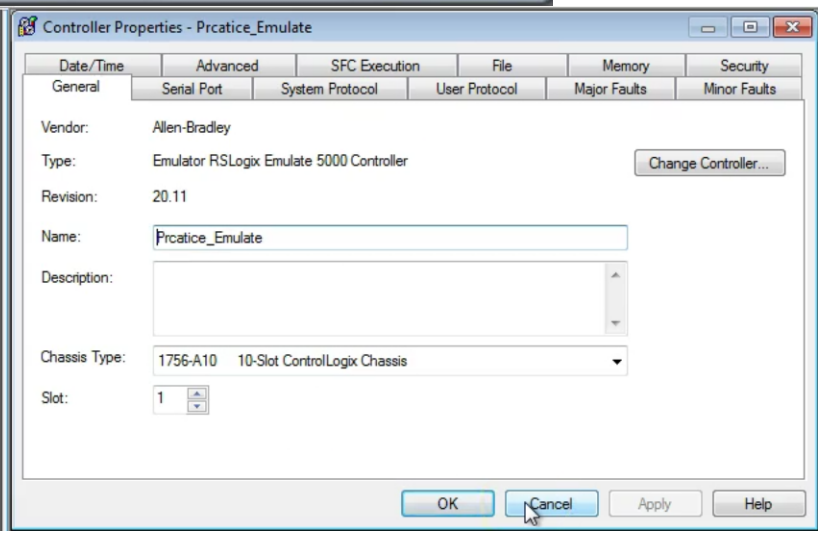
From the RSLogix compiler create a new project and select processor type ‘Emulate’.

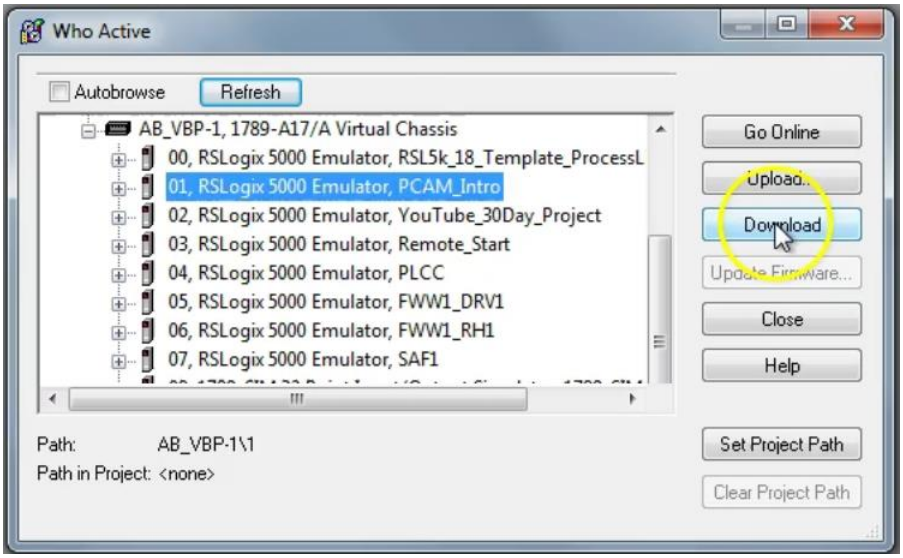



The below listed files are available in the course folder.






Use Who Active to download to the running emulator.

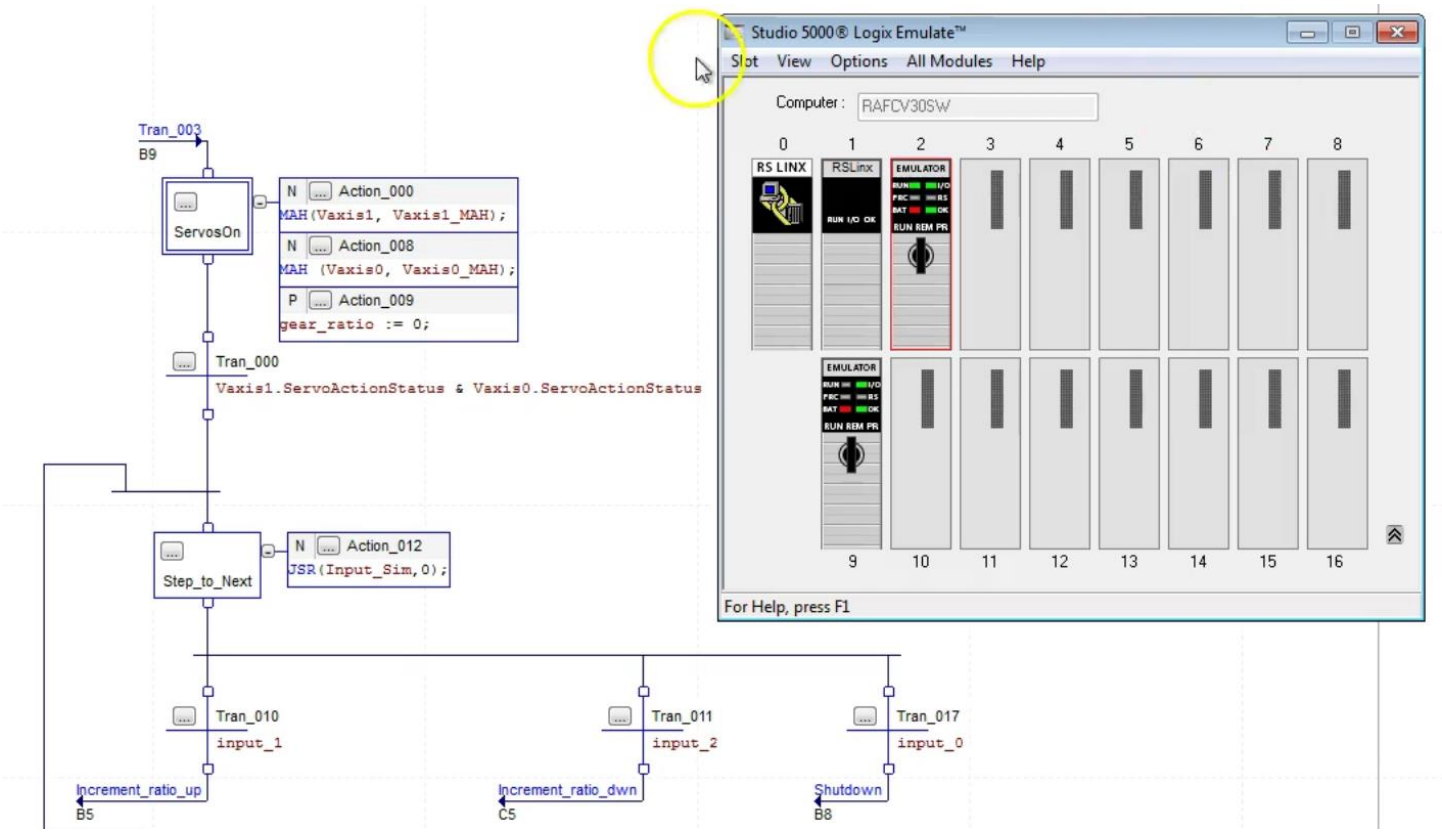
At this point you can create a project as you normally would and run it.

# Lecture 5: Studio 5000 Emulation

Pretty much the same as the RSLogix version.

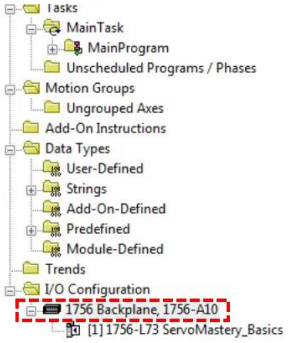
## Lecture 6: Emulating Gear Change Logic

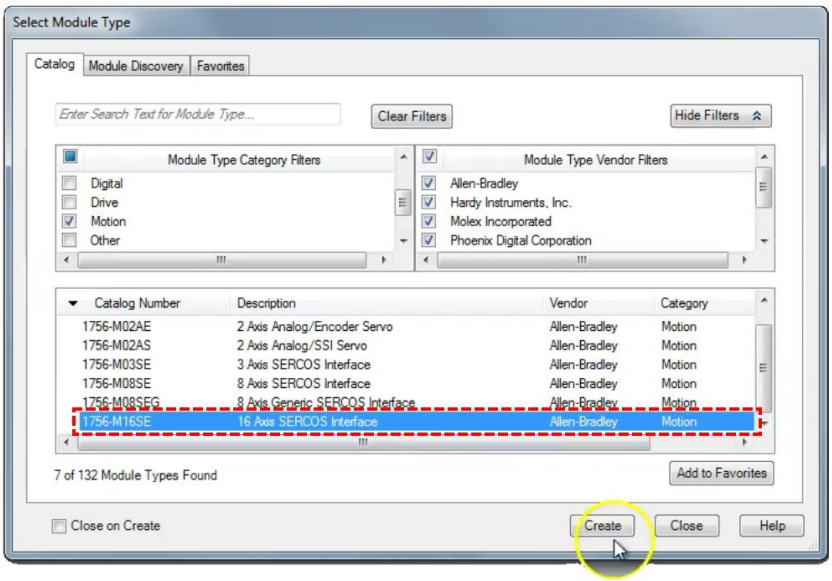
This section is more about downloading a project to the emulator, nothing about gear changing logic.

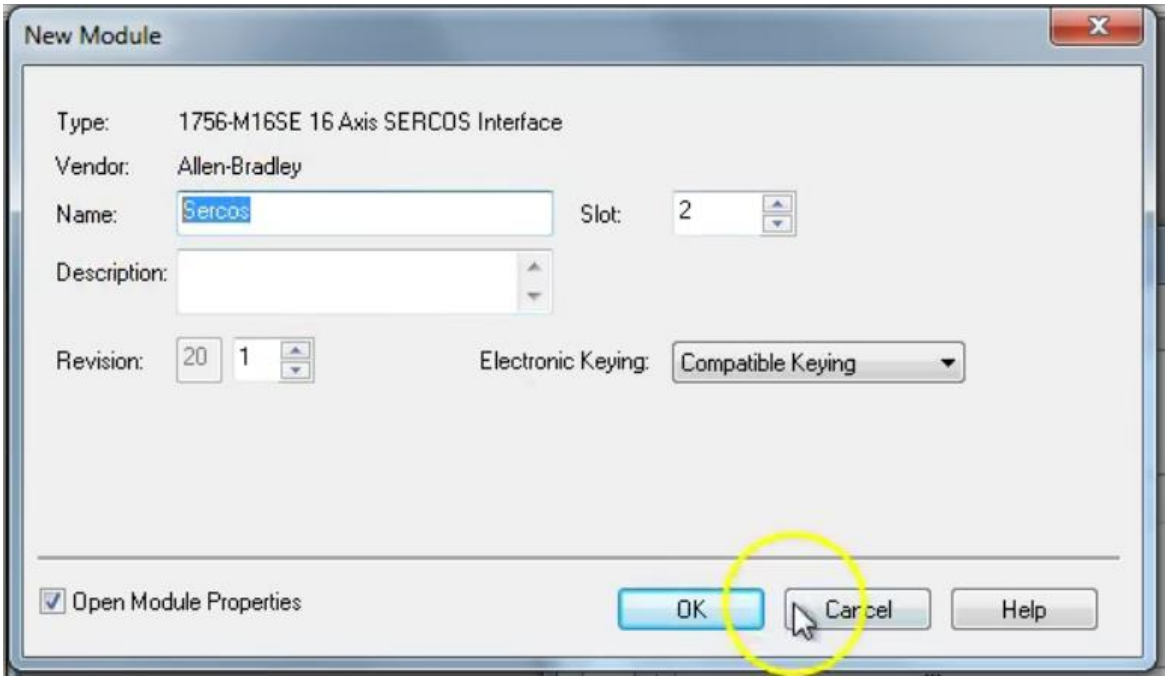



## Lecture 7: Troubleshooting a Servo Ring

Here we are talking about the proper setup of a servo ring what happens with a servo ring or CIP servo ring when the processes are powering up or when they are downloaded or whatever else the case may be. Begin by adding a motion card.

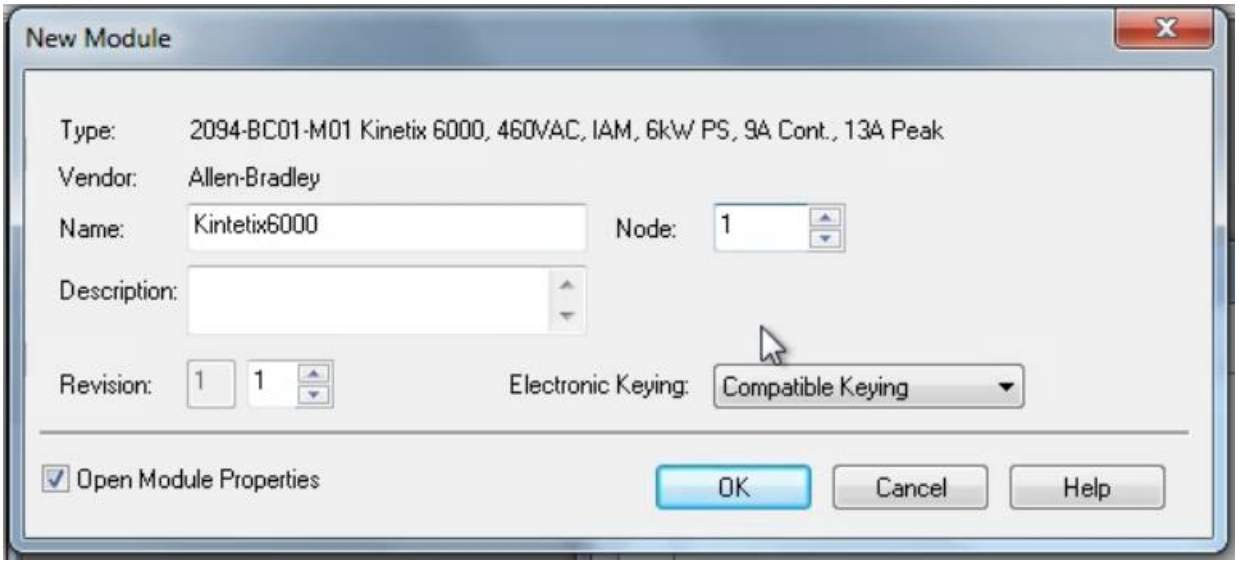


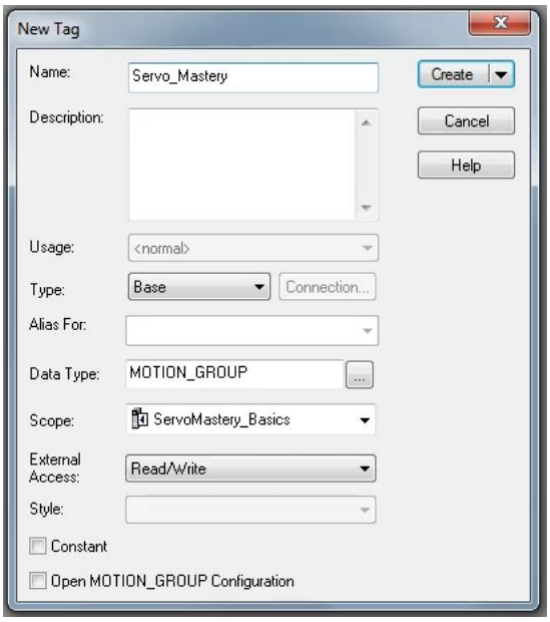


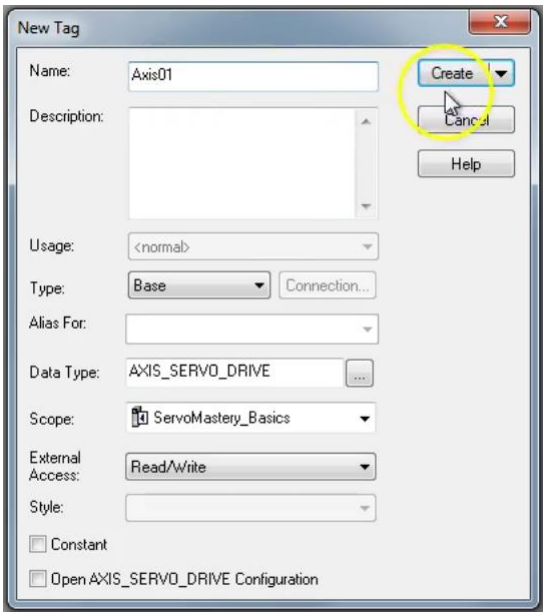

Next we want to add our motion controller (amp) under the SERCOS comm card.


Remember that the node number you configure the driver for is what ever number you have dialed in on the front of the amp.

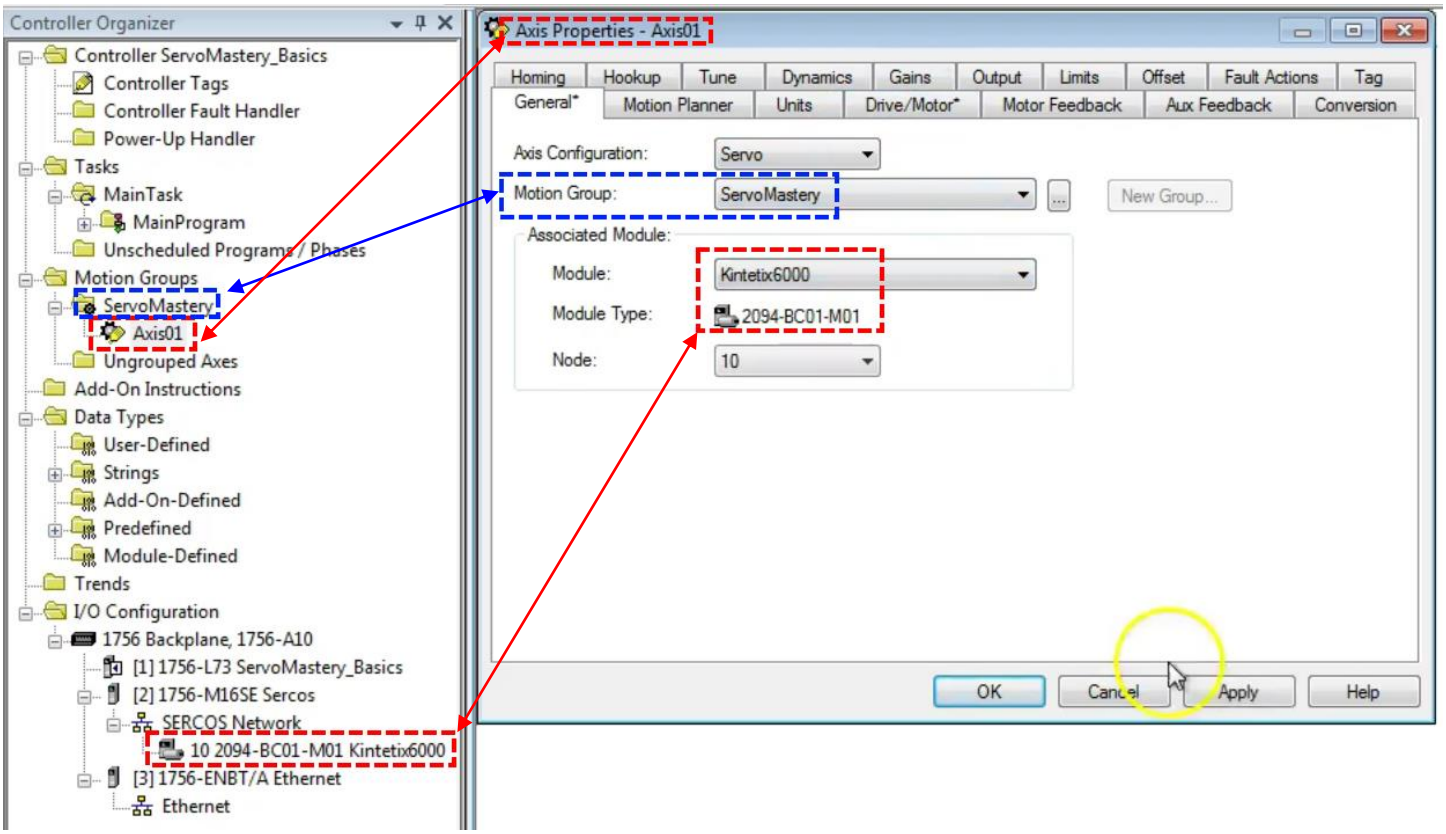


Now we want to create a motion group and add an axis of type AXIS_SERVO_DRIVE.

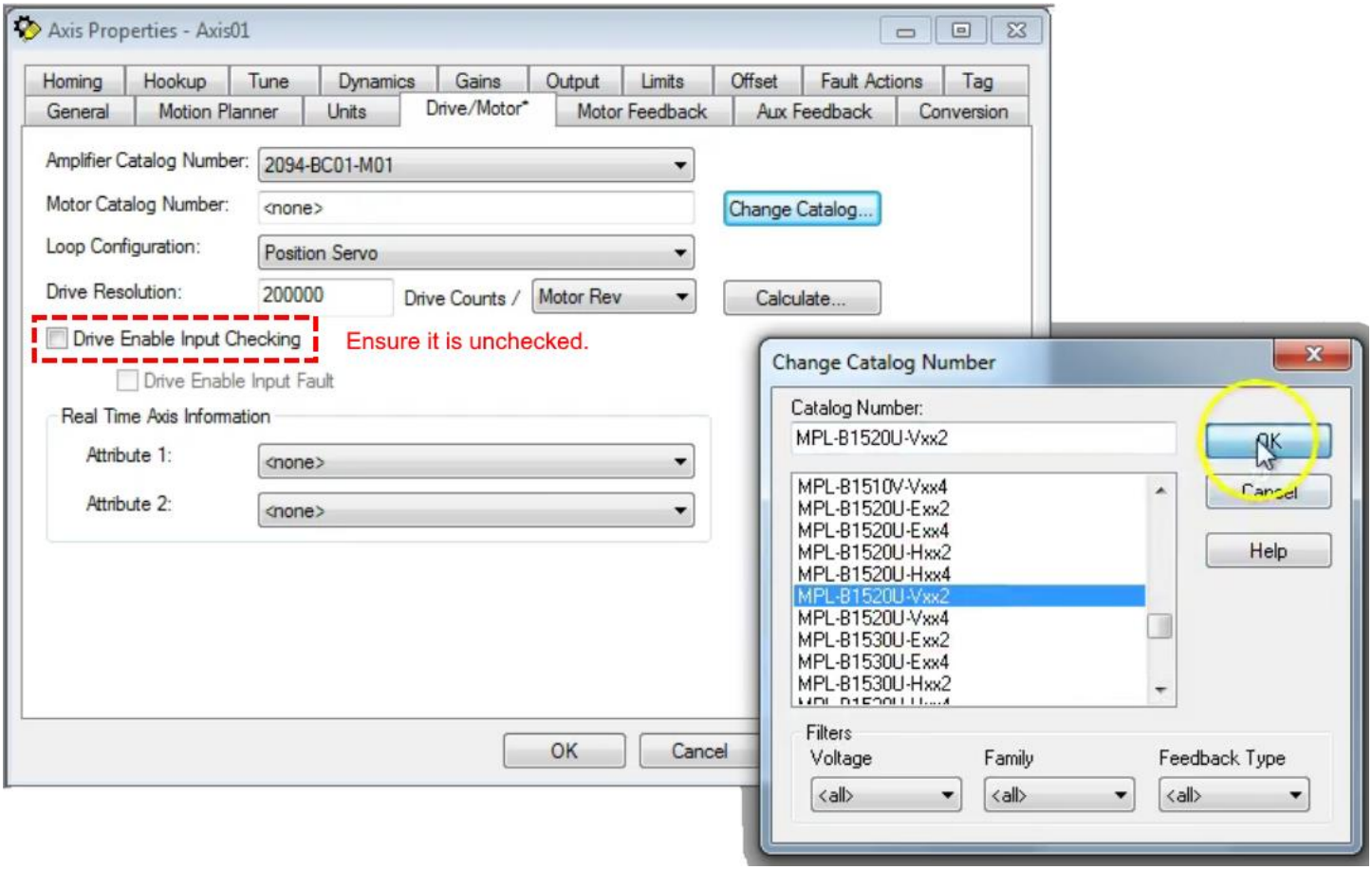


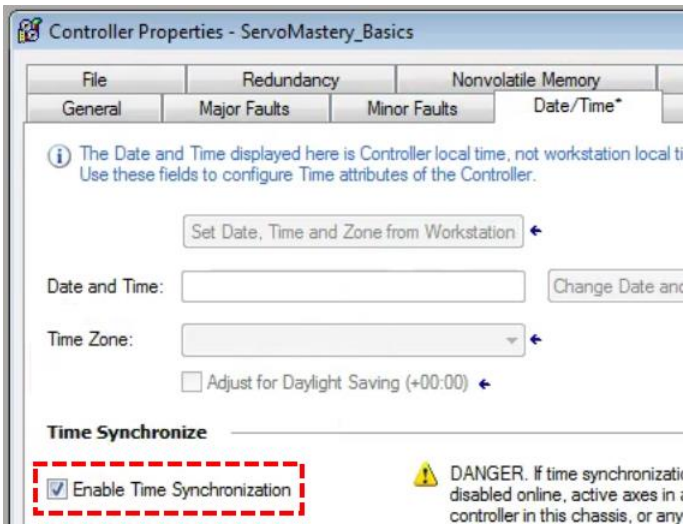

And then link the servo axis and the controller together.



Now add the motor that you are using.




Under controller properties Date/Time check Enable Time Synchronization.



The below table represents the natural scan order of the SERCOS. Active nodes refer to other motion controllers you may have on the network. Node = controller. Step 3 checks the nodes found on the network against those you have configured in the software. Step 4 is making sure you have your axis setup properly.

**The effect of an error is that the SERCOS system does not transition states. Watch the Status box in the lower left area to monitor the state transitions. You can also select the axis name under Motion Group to check the fault status of that equipment.**

Sercos System

0 = Looking for ring

1 = Looking for Active Nodes

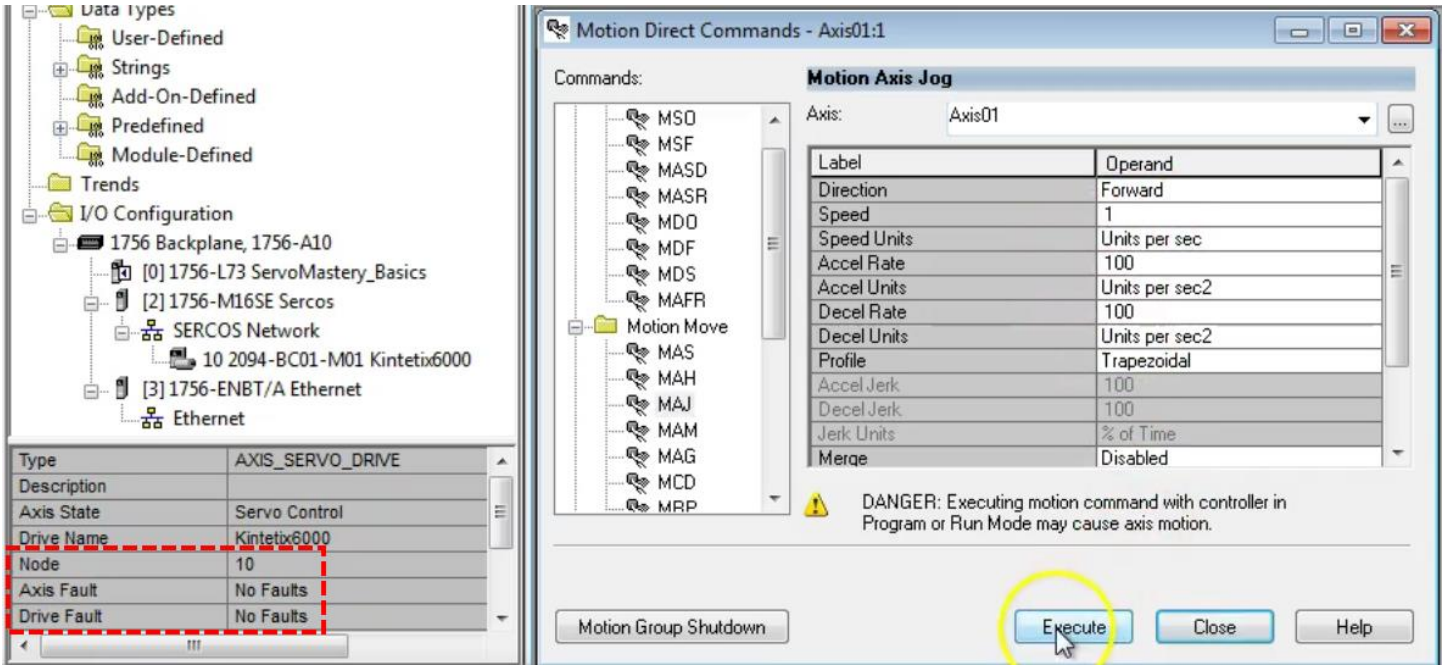
2 = Configuration Communication

3 = Configuration Nodes

4 = Configured Axis

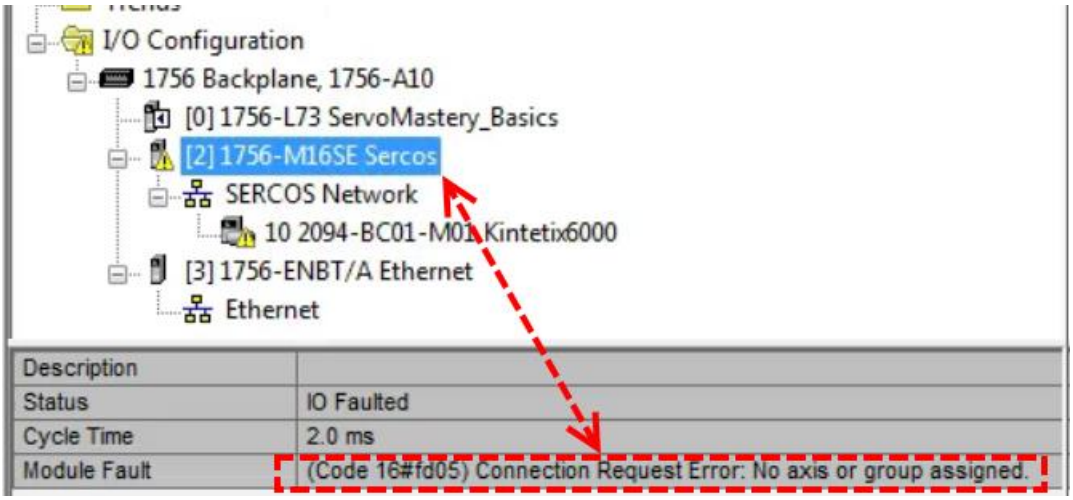
Note: if stuck in a state then open the quick view panel to see the axis state

If everything has gone well you are ready to start and operate the axis.

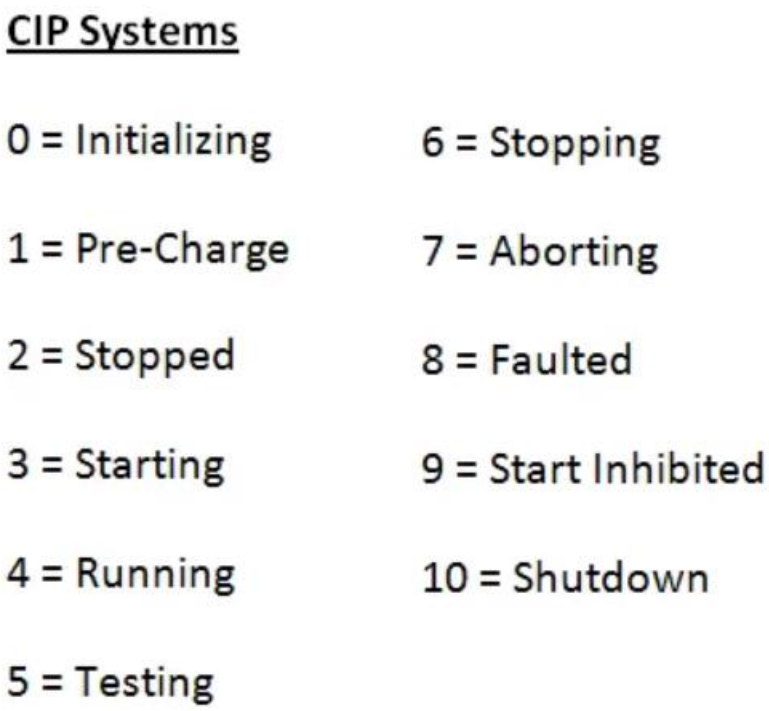



You can also view the numerical sequence on the front of the drive.

Fault message example.




Here is the ring sequence for CIP systems.



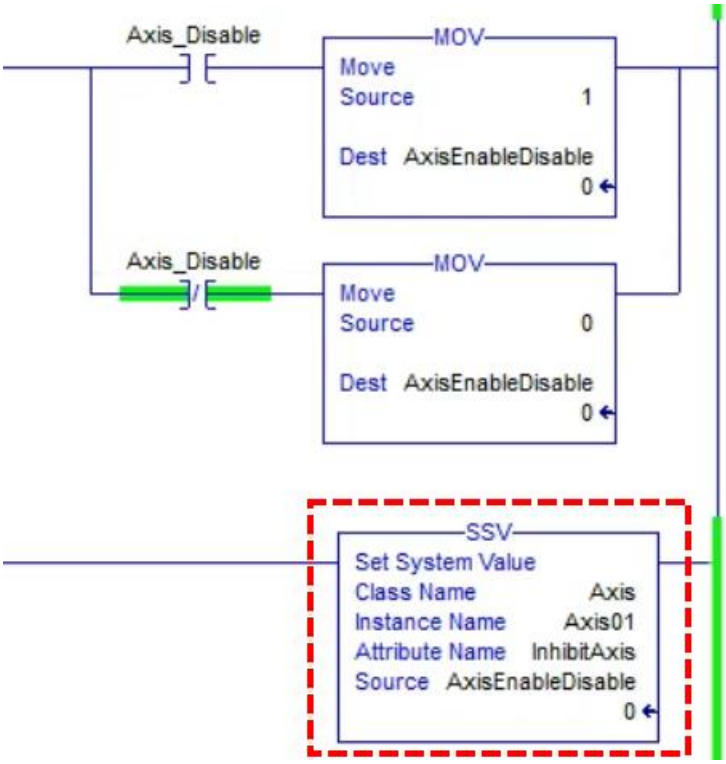
# Lecture 8: Inhibiting an Axis

We are not talking about the servo controller hardware, rather we are talking about the servo axis in the motion group. Although in some cases you can inhibit the hardware by going to the properties dialog and selecting that check box. This topic is covered in another lesson.

So now we are going to look at a method by which you can **inhibit a servo axis by means of a function in your ladder program**. We do this using the SSV, setting system value.

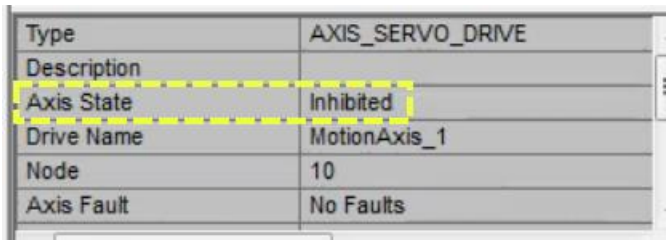
SSV to Inhibit Axis

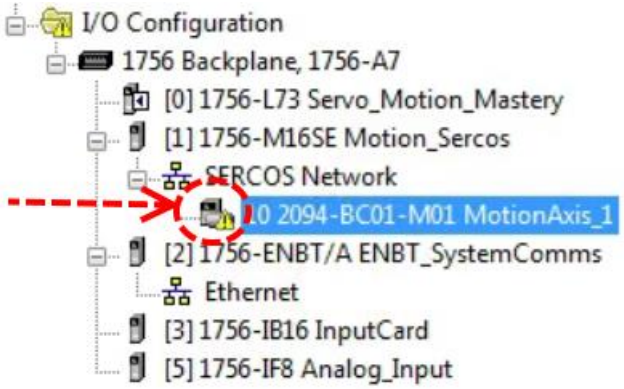
Class: Axis
Instance: Axis01 (the name of the axis in our motion group we wish to inhibit)
Attribute: Inhibit Axis
Source: (you must put in a tag, cannot use a digit. This is likely a tag you create for this purpose)




So we have put in a couple of move statements to load the AxisEnableDisable tag with 1 or 0 which triggers the function.

When we run the function and load the tag with a 1 we see that the axis is inhibited.



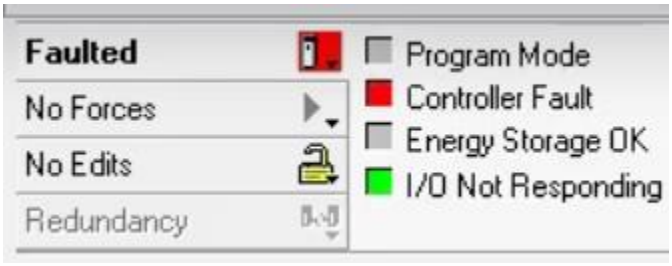




Now we will look at disabling the hardware component.

SSV to Inhibit Axis

| | |
|---|---|
| Class: | Module |
| Instance: | ModuleAxis_1 (the name of the axis in our motion group we wish to inhibit) |
| Attribute: | Mode |
| Source: | (you must put in a tag, cannot use a digit. This is likely a tag you create for this purpose) |

Result is a processor fault.



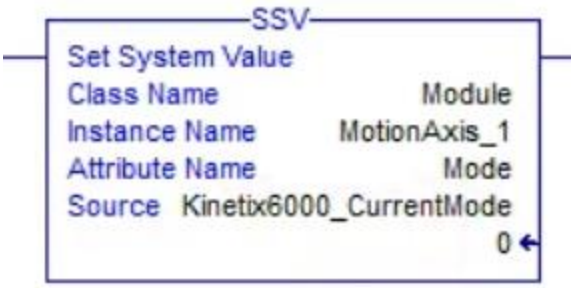

## Lecture 9: Inhibiting a kinetic 6000 Drive

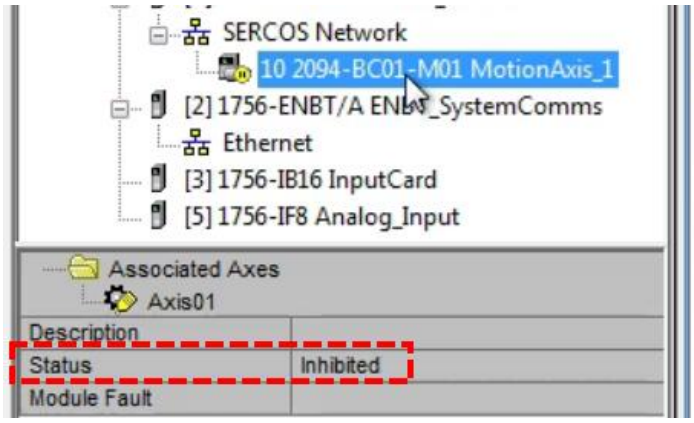
In this lecture we will concentrate on how to inhibit the module (hardware) itself. We are not going to do this by means of the property dialog (this method is not always available). We will use a code implementation. [the "CurrentMode" tag is not available in all drives, check, your looking for inhibit/shutdown]

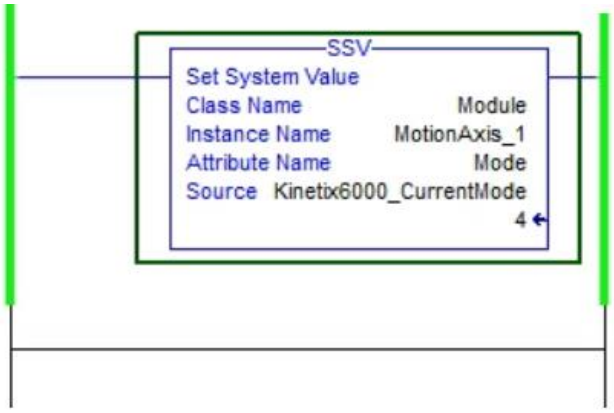
SSV to Inhibit Axis

| | |
|---|---|
| Class: | Module |
| Instance: | ModuleAxis_1 (the name of the axis in our motion group we wish to inhibit) |
| Attribute: | Mode |
| Source: | Kinteix6000_CurrentMode |

In this case changing the tag to the value 4 inhibits the drive.



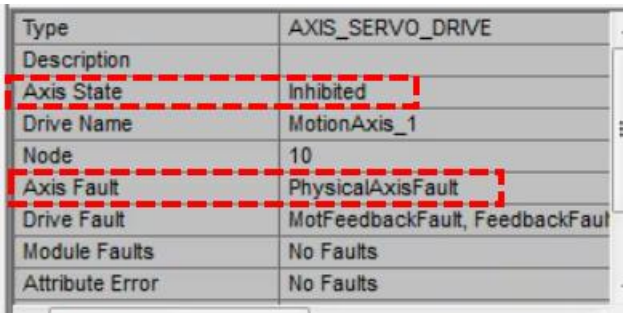




## Lecture 10: Motor Removal Troubleshooting

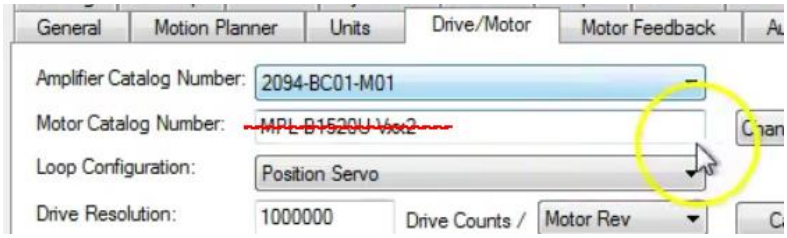
If an axis is inhibited and then goes into fault the system continues on, it is not concerned and is still fully functional. The IO Not Responding is good, not flashing.

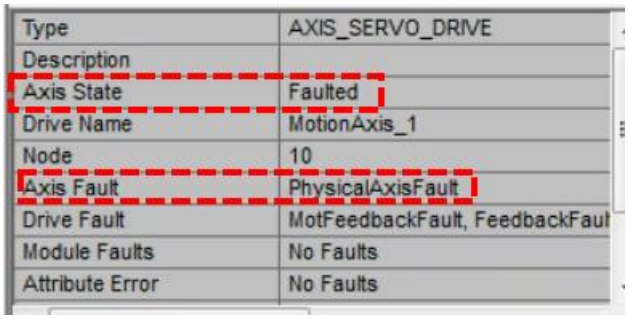
But once we enable the axis if the fault still exist the system will indicate so and the IO Not Responding LED will blink.

As an example, we will remove the motor from our hardware configuration.



| Type | AXIS_SERVO_DRIVE |
|---|---|
| Description | |
| Axis State | Inhibited |
| Drive Name | MotionAxis_1 |
| Node | 10 |
| Axis Fault | PhysicalAxisFault |
| Drive Fault | MotFeedbackFault, FeedbackFaul |
| Module Faults | No Faults |
| Attribute Error | No Faults |



| Type | AXIS_SERVO_DRIVE |
|---|---|
| Description | |
| Axis State | Faulted |
| Drive Name | MotionAxis_1 |
| Node | 10 |
| Axis Fault | PhysicalAxisFault |
| Drive Fault | MotFeedbackFault, FeedbackFaul |
| Module Faults | No Faults |
| Attribute Error | No Faults |

At this point the axis is no longer faulted, the application understands there is no motor for the faulted drive to affect so it continues on. This can be handy in troubleshooting. When trying to track down an axis fault you can try removing the motor. If the axis comes back up as okay you may have a motor problem that was causing the axis fault.

**Now we know how to disable an axis, disable the drive, and remove the motor from the possibilities.**

## Lecture 11: Course Rate Update Timing (CUP)

Whenever an application is using motion the first routine(s) processed is the motion group and the associated axis. The frequency at which the axis routines are solved is based on the CUP. The **Course Rate Update (CUP)** is defined as the milliseconds it takes to scan all the servos in the motion group, solve the logic, and update the drive positions, rates, torque, etc. This is how long that process takes. It is updated in .5 ms increments.

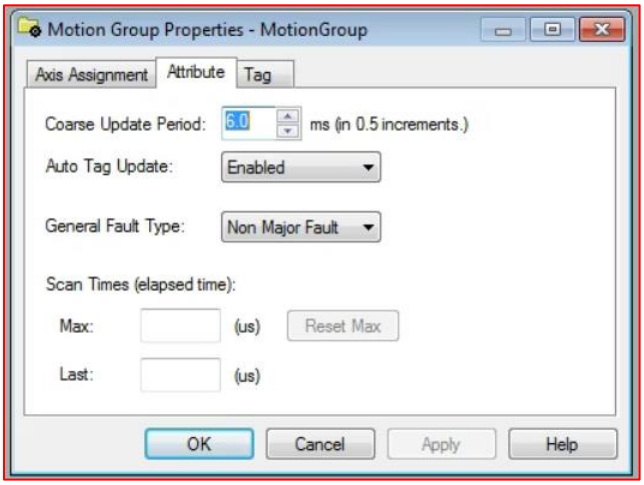
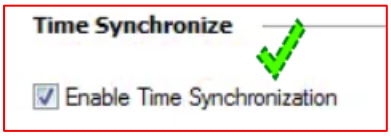



*Figure 1 Attribute tab of the Motion Group Properties.*

To determine the value you should use, a good rule of thumb is 1 ms per virtual axis, 0.5 ms per axis servo drive, and 2.0 ms per SERCOS physical axis. You may want to round up a ms.

**Servo Drives (physical drives)**

If running an **L7 processor** you can drop it down to 0.25 ms per axis.
If running an **L6 processor** you can drop it down to 0.5 ms per axis.
If running an **L55 processor** you can drop it down to 1.5 ms per axis.
If running an **L5 processor** you can drop it down to 2.0 ms per axis.

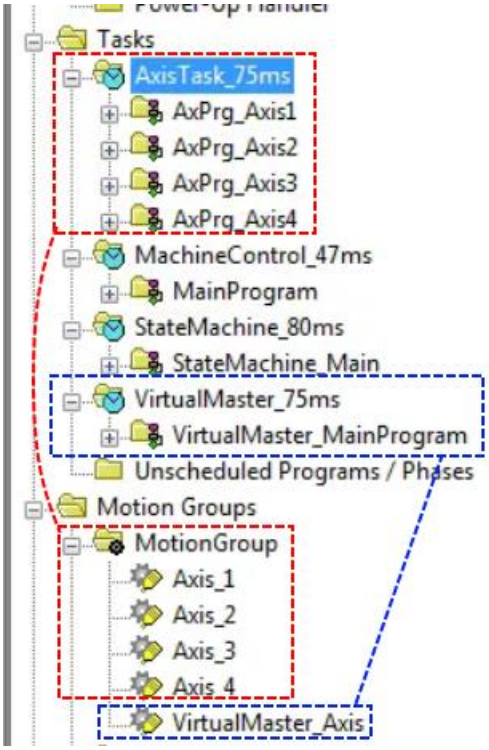



The above table assumes periodic task scanning. This is an interrupt with resume process.
L8 is a newer generation of drive and is likely faster than the rates above.
Needless to say improper timing entries can be a source of faults.

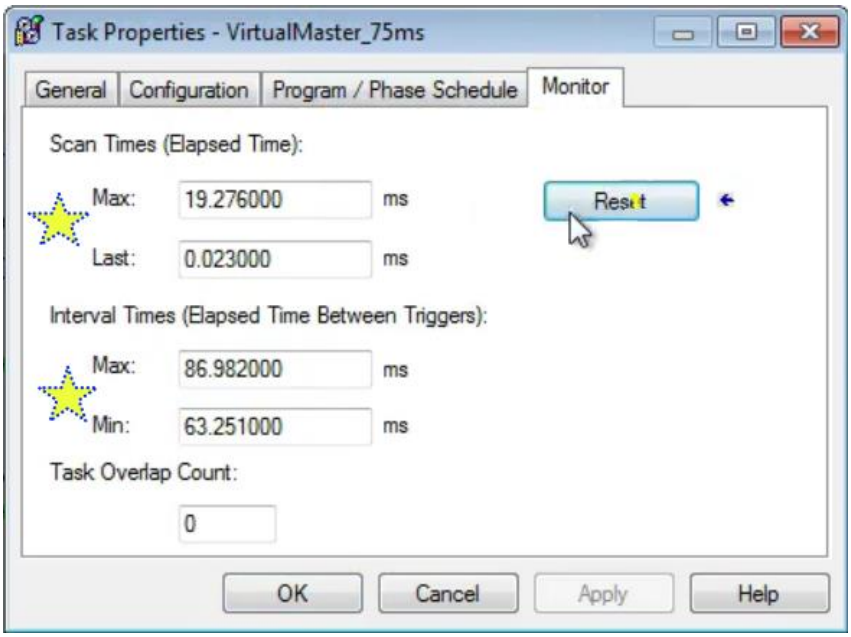
## Lecture 12: Tasks, Periodic and Continuous

Keep in mind it is a two step process to execute a servo command; the first cycle will write the command it wants the drive to execute and the second cycle will execute the command.

A good idea is to have your axis processed at least 4 times faster (as often) than the other tasks not associated with servos.



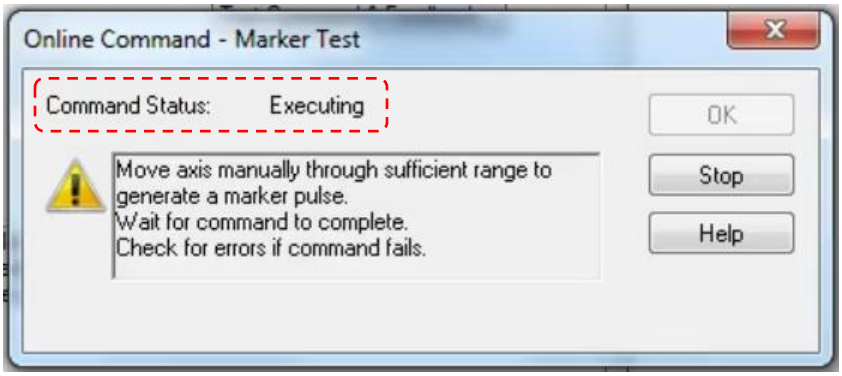
On the **Monitor** tab of any tasks properties dialog you can check the actual scan time for that task.




On a SERCOS card you can check and adjust the **Requested Packet Interval** under the Connection tab.

**Lecture 13: Servo Commissioning Testing**

Axis properties **Hookup** tab.



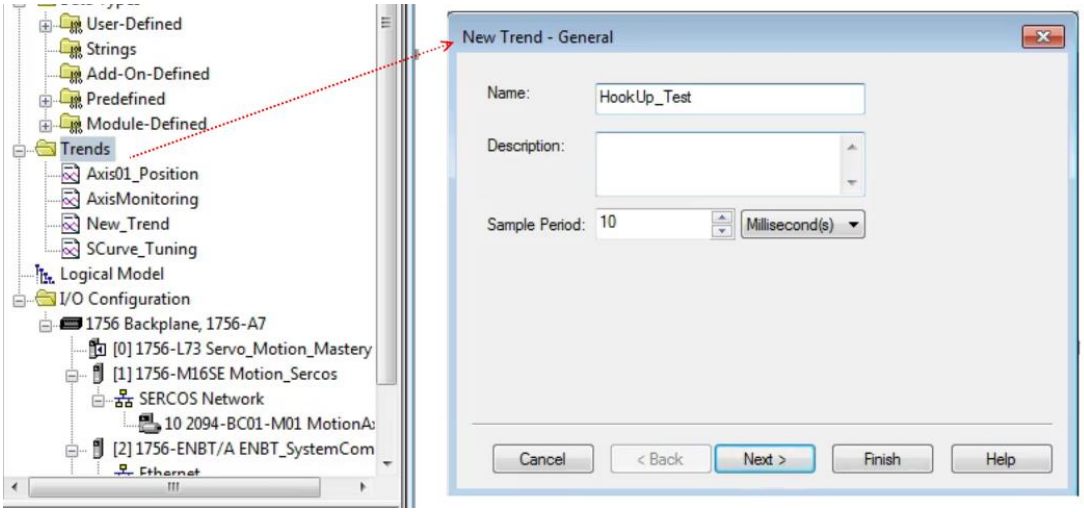

**Marker Test**: turn the motor shaft, the servo will determine the direction and make note. This is testing the marker of the decoder. When you press the button you will see the below dialog. When you turn the shaft the text "Executing" will change to "Command Complete".


**Test Feedback**: enter a value for Test Increment in Position Units then press **APPLY** and then click the Test Feedback button. You are going to have to rotate the shaft that distance so do not enter too large a number. Once the drive has made the reading the "Executing" text will change to "Command Complete". The result is that the feedback polarity is determined by the drive.

**Test Command and Feedback**: what this is going to do is turn the servo on and slowly increase the speed through a full rotation so that it can record the commutation. During this test watch the motor and confirm it is moving in the correct direction. The software will ask if the motor turned in the correct direction and it will adjust itself whether you say yes or no.

# Lecture 14: Trending a Servo Hookup Test

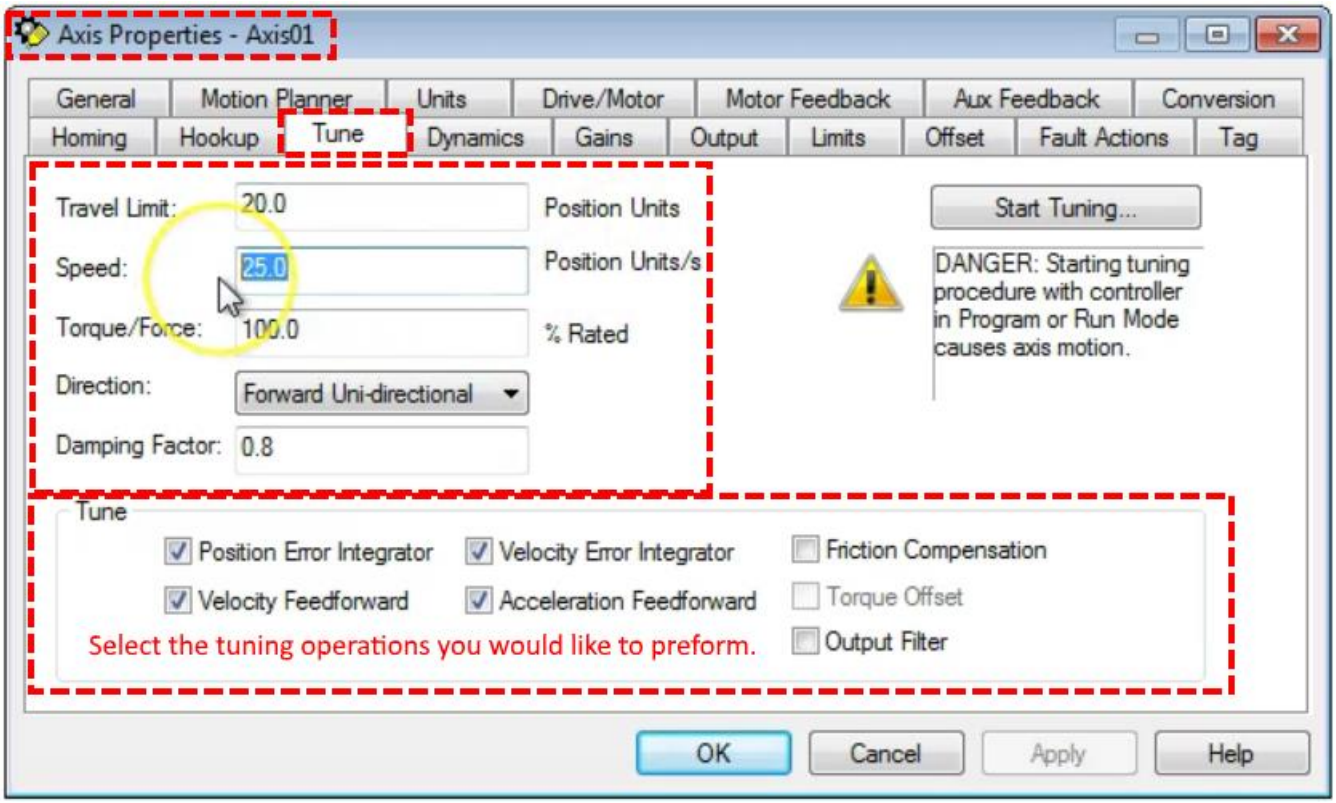
Create a new trend, call it Hookup_Test. From chart properties Pens tab you can add the pen Axis01.ActualPosition. Set the x-axis to 15 seconds.



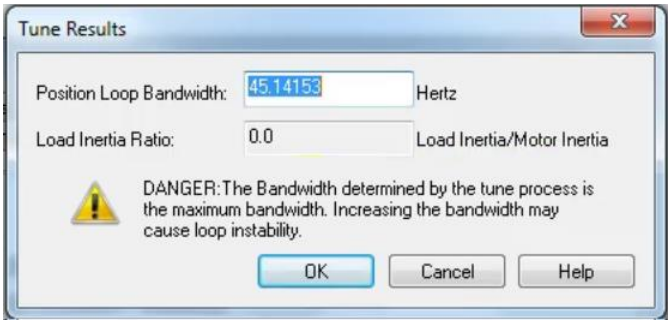

Before you begin running the axis use the motion direct commands to home the axis (MAH). At this point you are ready to run the hookup test. Run all three tests from the previous lecture.

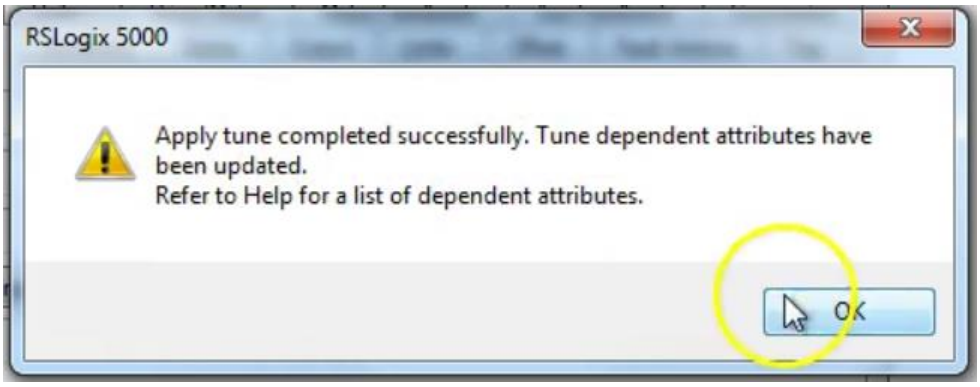
# Lecture 15: Quick Start to Servo Tuning

Here we will auto tune the motor. Often times it is best to have an uncoupled motor when doing the auto tune.



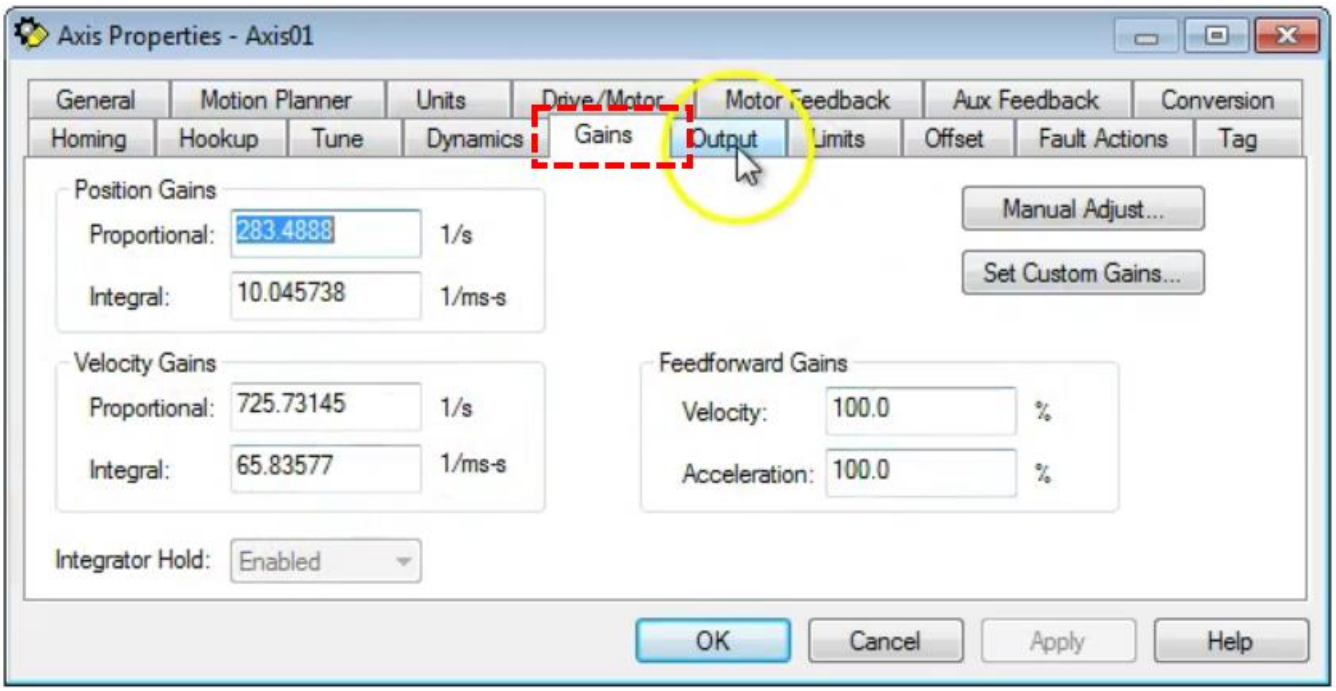

Click Start Tune. You may get some feedback from the algorithm asking that you make adjustments to things like speed. When the command completes you will get some feedback such as that shown below.



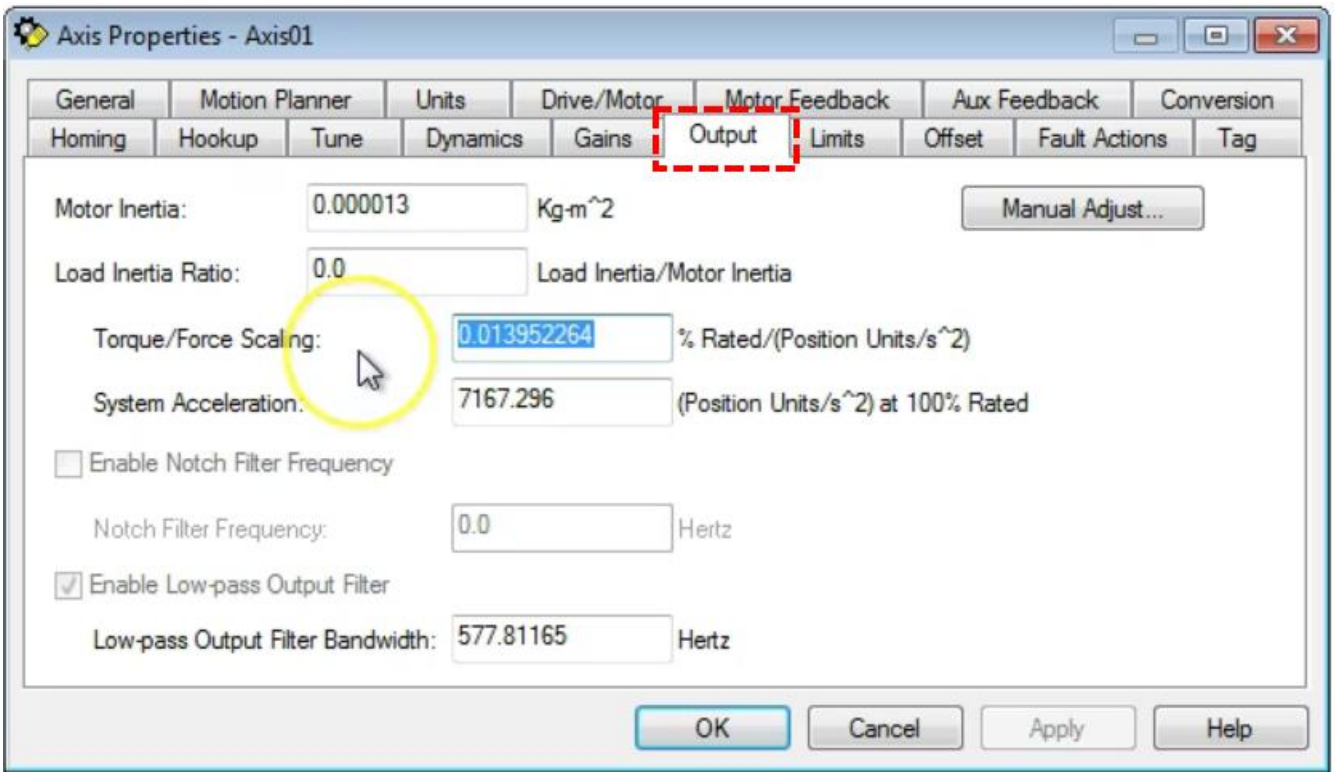




After an initial success you may wish to run it for a longer Travel Limit. Then use Motion Direct Commands to jog and make sure the motor seems to function properly (no high pitch or any such thing).

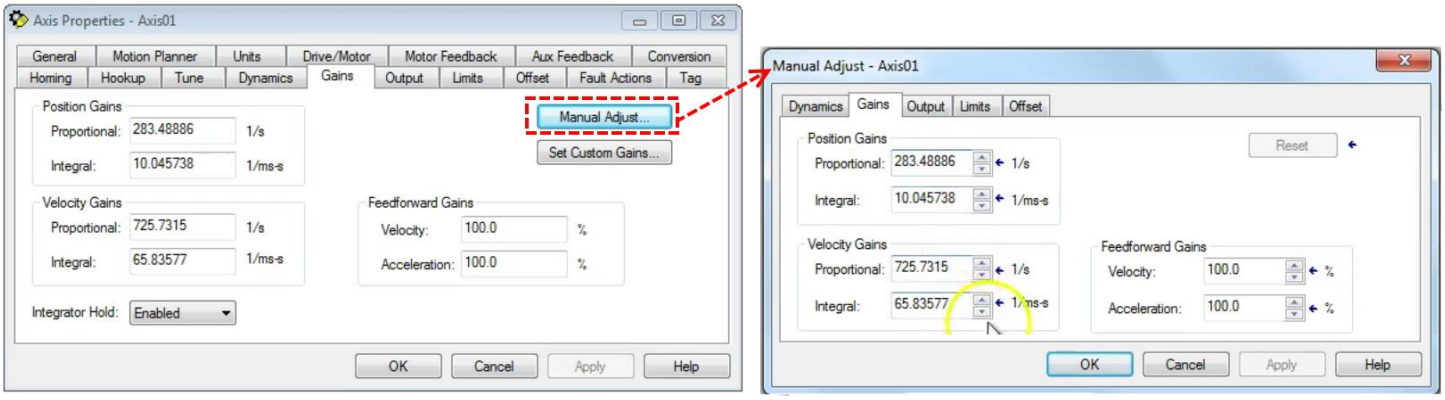
Axis properties has a Gains tab for adjustments.




The **Output tab** can also be helpful. Sometimes, depending on the application, adjustments to torque force scaling can be helpful in obtaining a certain response.

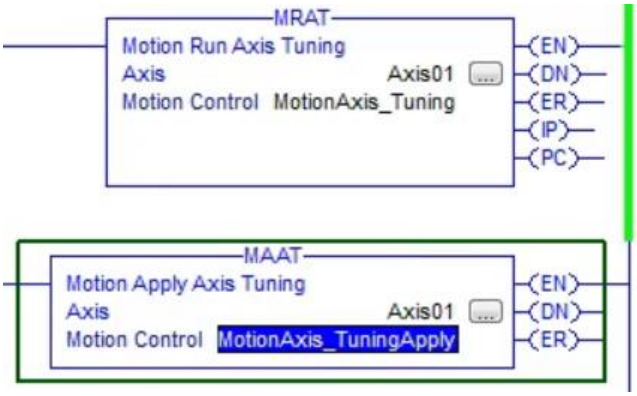


On the Gains tab click Manual Adjust to access a dialog which allows you to change gains while the motor is running. Same for Torque Force Scaling on the outputs tab. You can also add filtering.

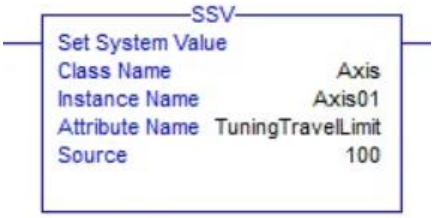



**Systems that use CIP-Motion protocol often are easier to tune when you have Notch Filter Frequency enabled on the outputs tab.**

The commands below allow you to run an auto tune and then apply the results from the ladder logic. These functions use the parameters you have in the system already (such as those discussed above).



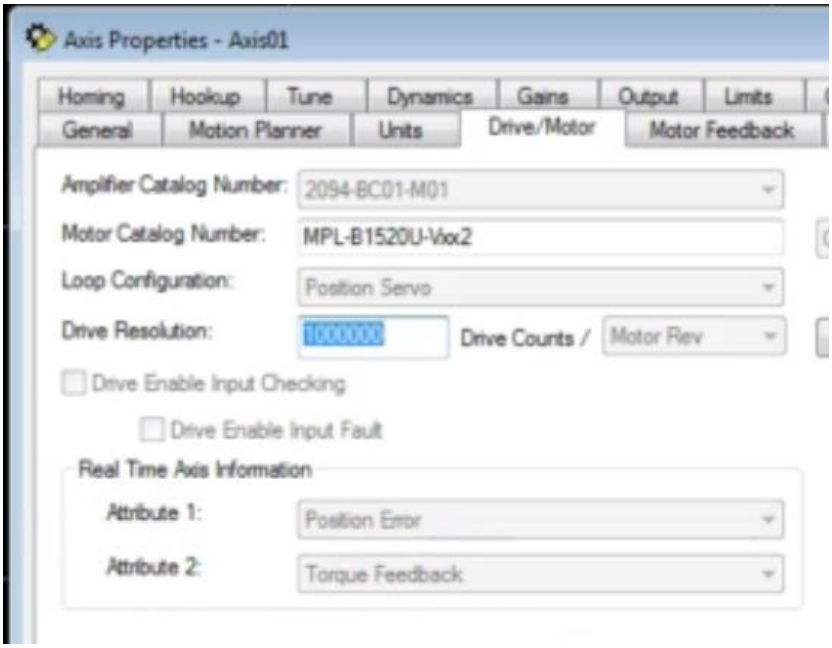

You can use a SSV to acquire or adjust specific values associated with tuning such as the example below.



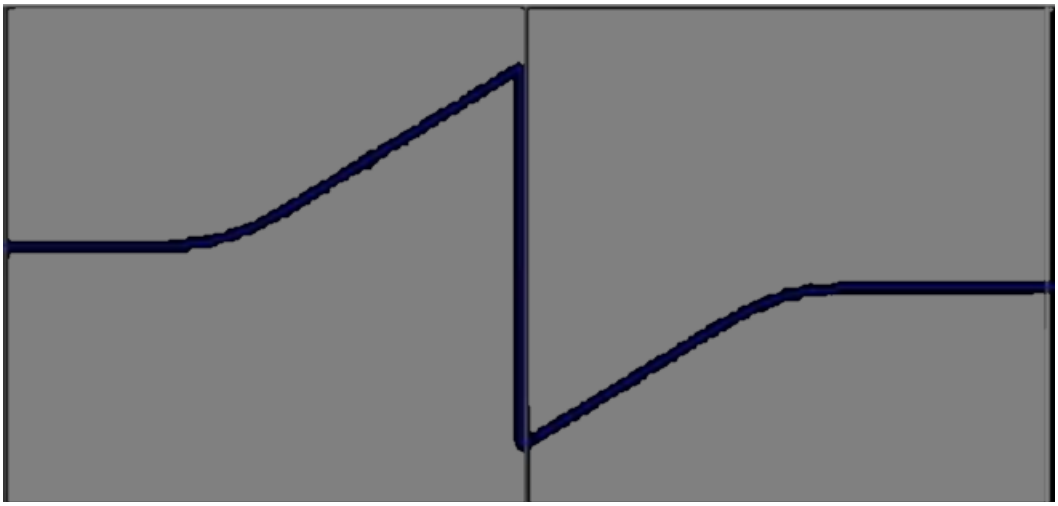
# Lecture 16: Tuning an S-Curve Movement from an Instruction

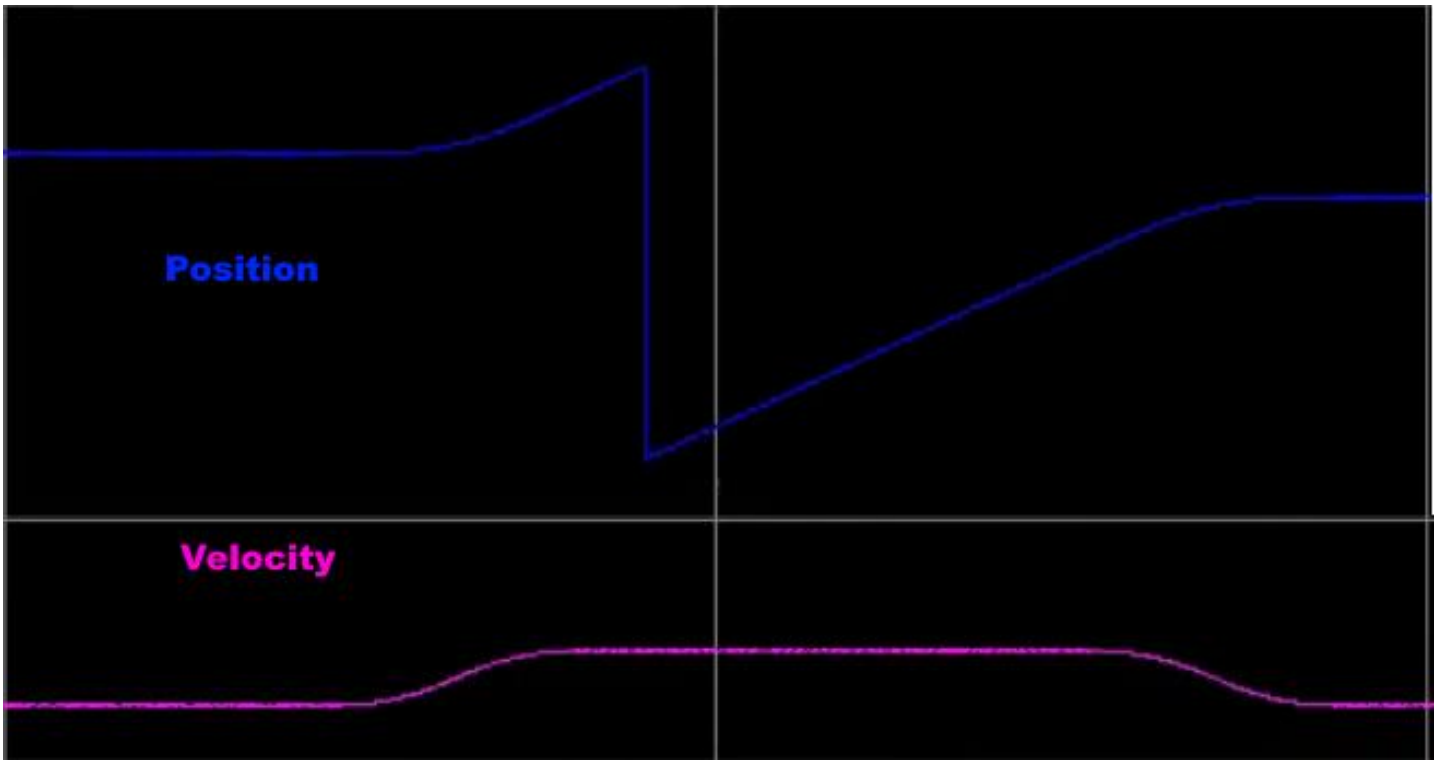
How to set up a trend for tuning an S-curve. Systems with a large inertia may do better with an S-curve type acceleration. So the question is, how do we determine when the system is responding smoothly.

Start with a trend with the following parameters: .ActualPosition, .AverageVelocity, .TorqueFeedback. Note that TorqueFeedback is one of the parameters that will not appear in the list of parameters that can be plotted unless you have entered (requested) it in the Properties dialog under the Drive/Motor tab in the Real Time Axis Information area.

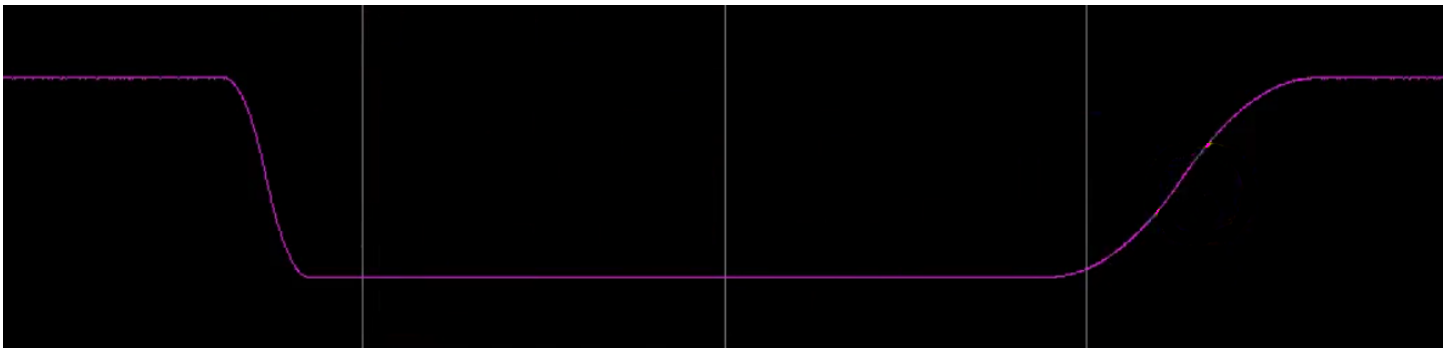



Below plot shows the S characteristic.






The “S” shows up in position but it shows up much more strongly in velocity. The activity of changing position with an S-curve is strongly indicated in the velocity curve. The Jerk value in the MAM command as well as the accel decel rates modifies the curve. In the below case we went from 50 to 100 Jerk and then down to 10.



The tradeoff is how long it takes a maneuver to complete as opposed to how smooth the transition is.

**NOTE THAT ACCEL AND DECEL JERK VALUES DO NOT APPLY WHEN USING TRAPAZIOD.**

However it ***is*** affected by accel/decel rates.

## Lecture 17: Servo Acceleration Explained

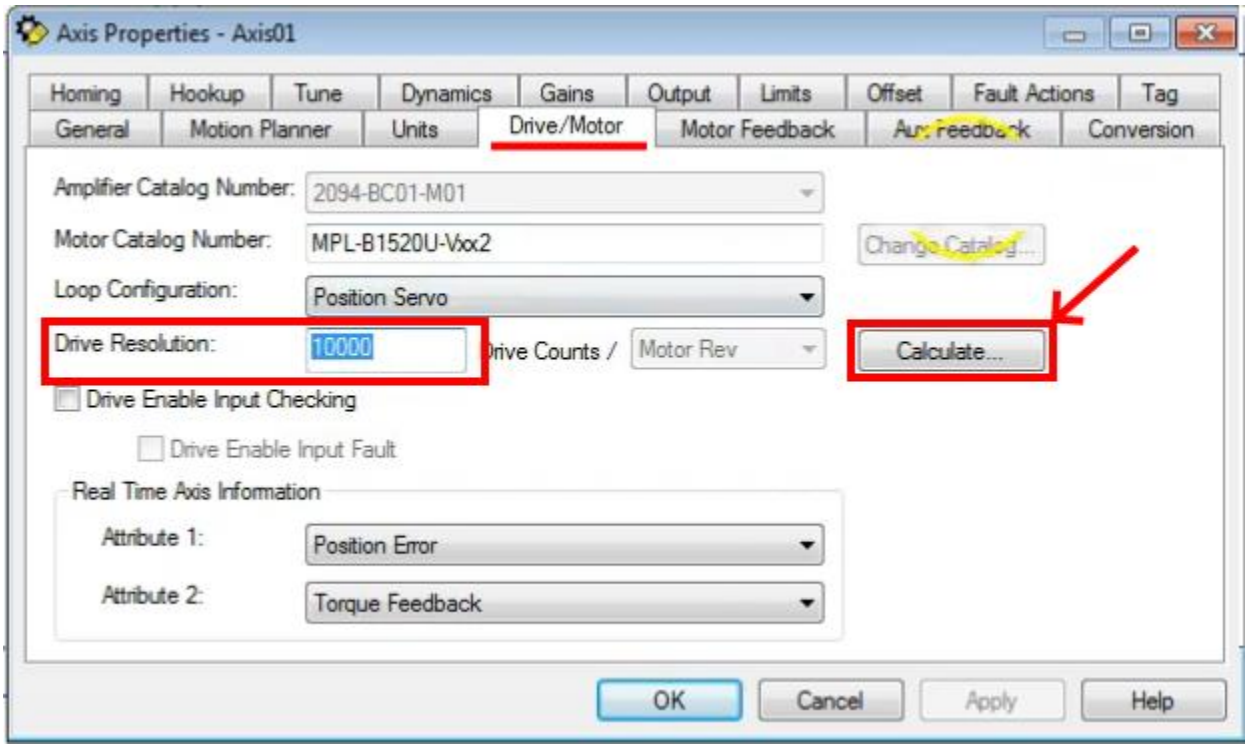
This chapter clears up a few items from previous section.

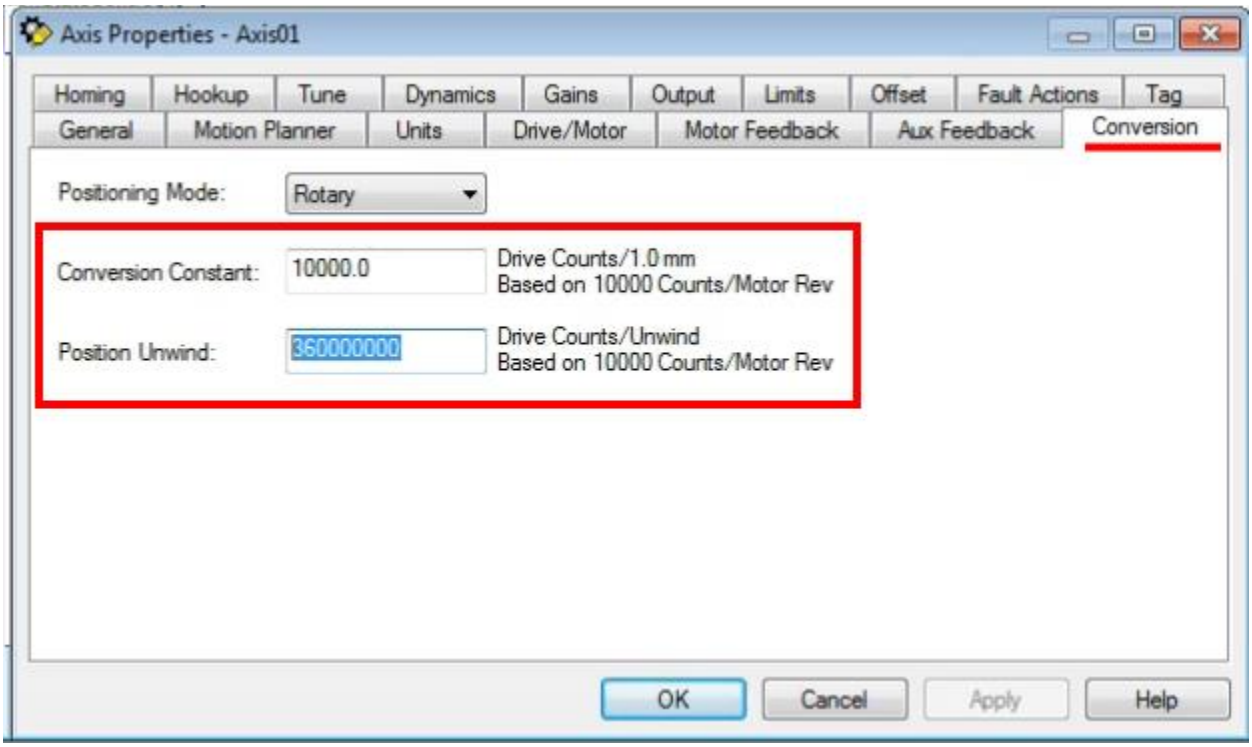
## Lecture 18: Servo S-Curve Jerks Explained

This lecture points out the difference between a S-Curve and a Trapizoid.

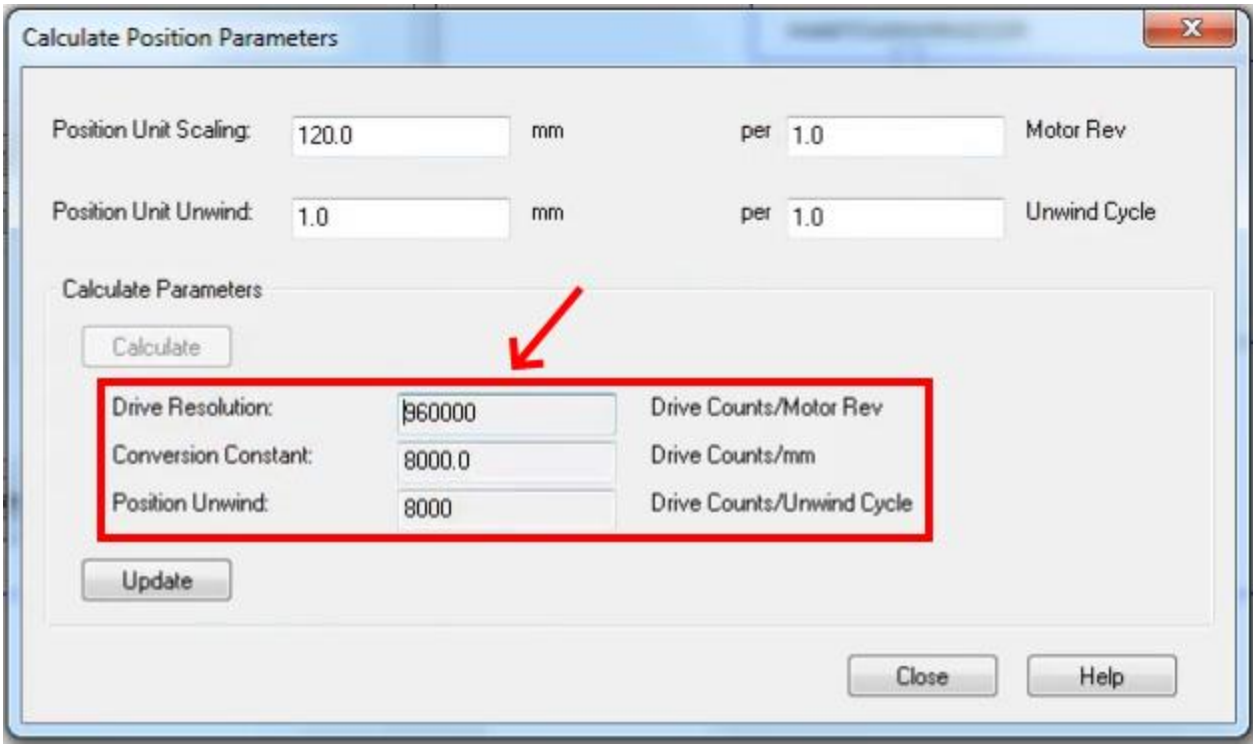
# Lecture 19: Using the Conversion Constants Calculate Tool

What are we referring to when we speak of conversion constants? **Drive Resolution** under the Drive/Motor tab, **Conversion Constant** and **Position Unwind** under the Conversion tab. The values will always be dependent on the application. The tool we are going to examine can be helpful for determining these values.

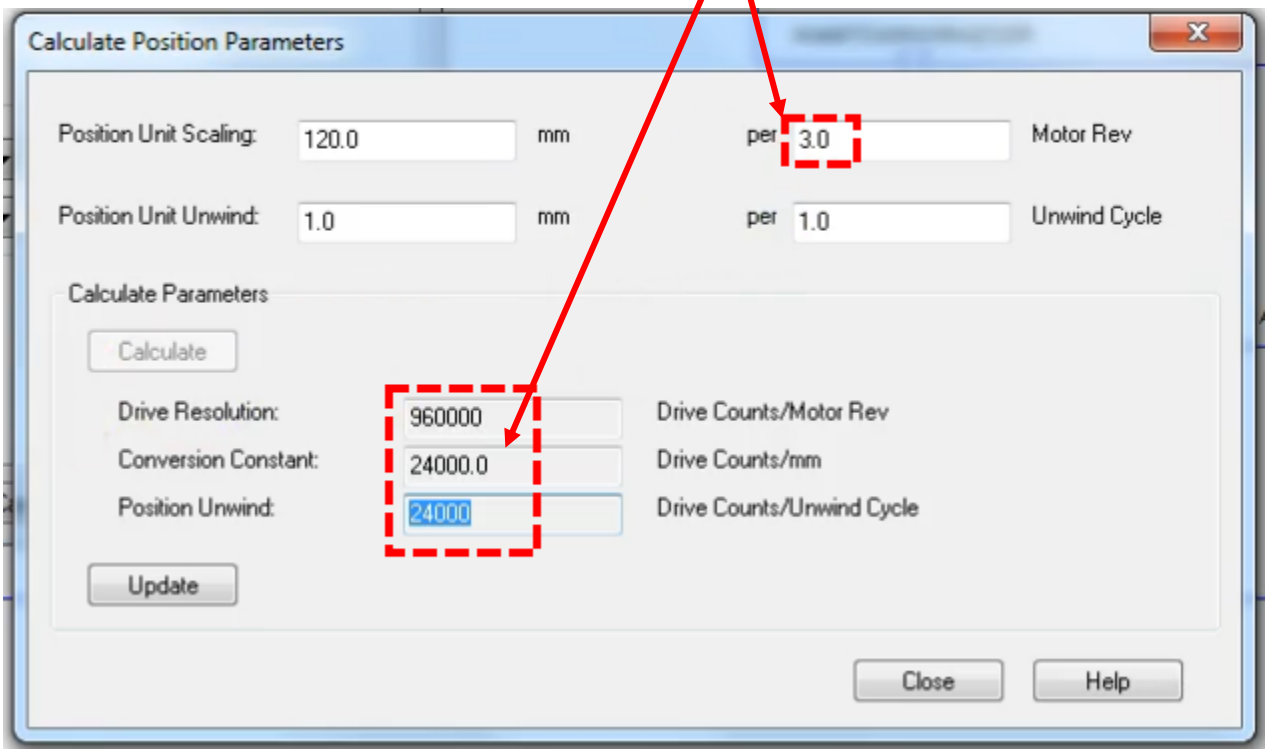


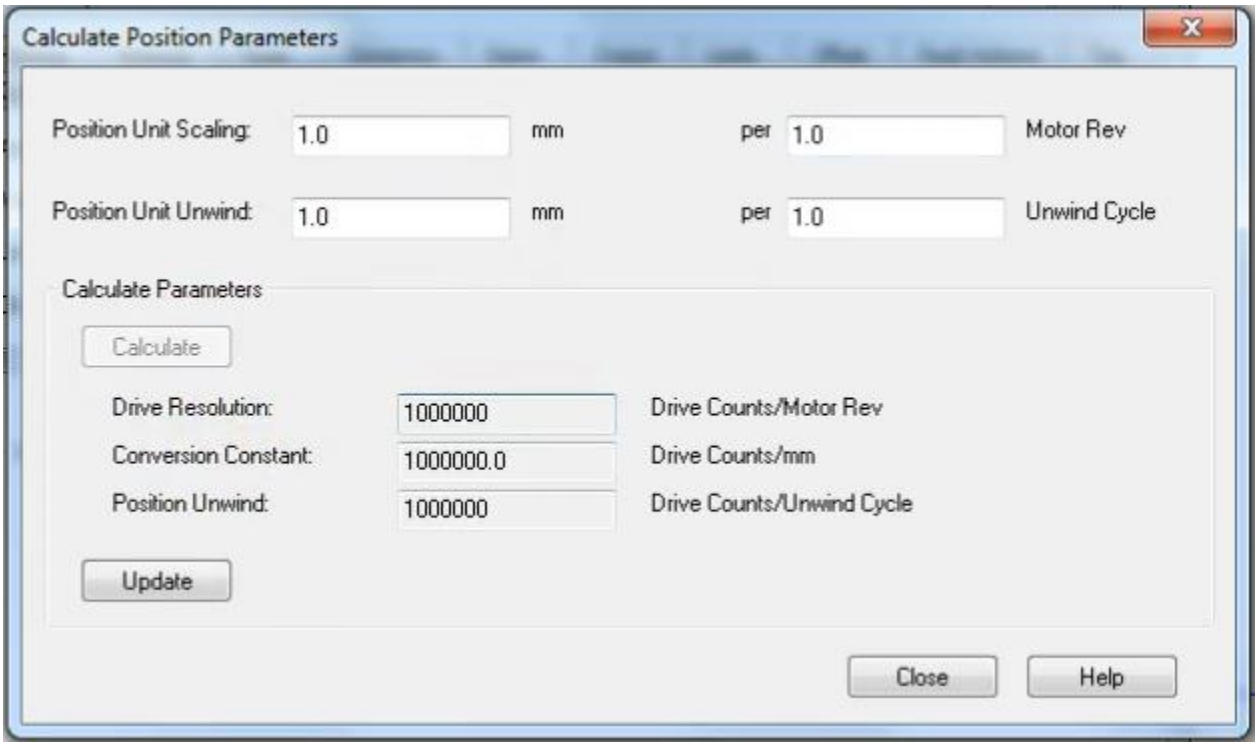

You must be **OFF LINE** to use this tool. On the Drive/Motor tab click on the **Calculate** button. This will bring up a dialog called **Calculate Position Parameters** (next page).



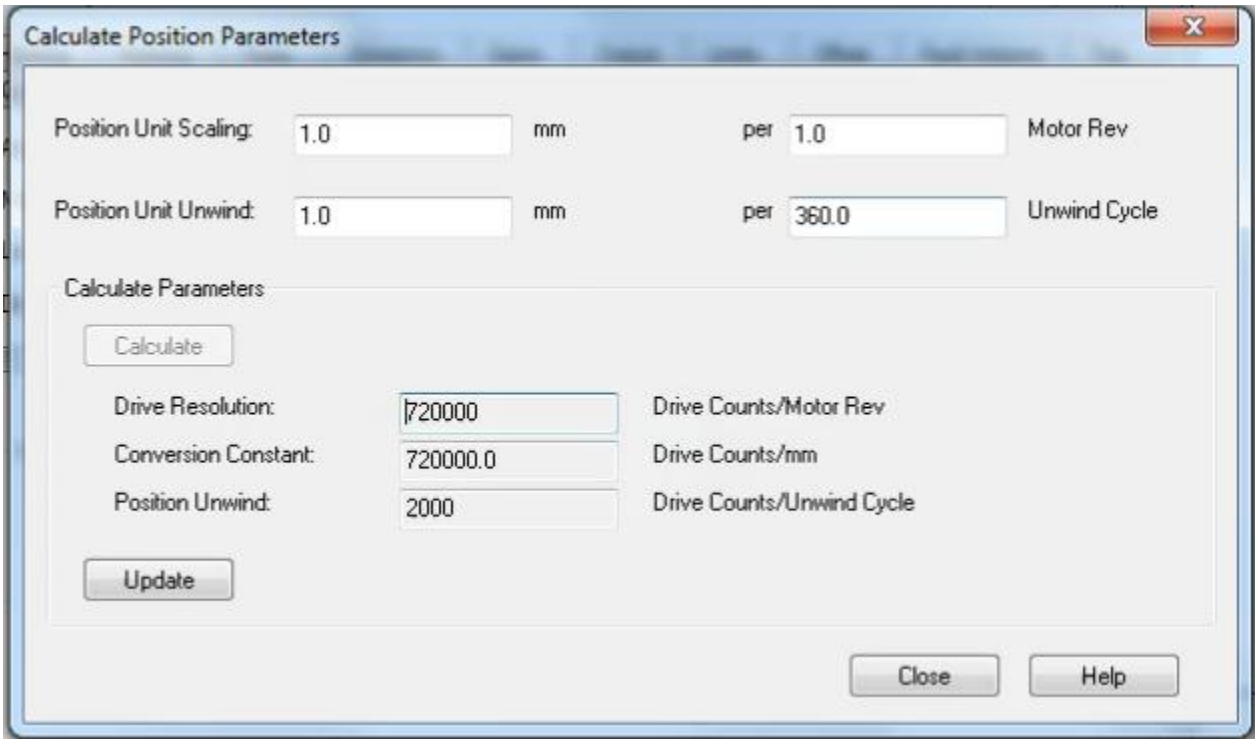

The units will match what you have configured for your application. Fill in the text boxes according to your applications needs and click on Calculate [? Update ?]. In the above example we want to move 120 mm per motor revolution. The tool calculates the three values we will have to enter into the setting on the two tabs shown on the previous page. This also applies to **gear boxes**, if we had a 3:1 gear box on our shaft in the above example we would enter the number 3 in the "per Motor Rev" text field (see below).



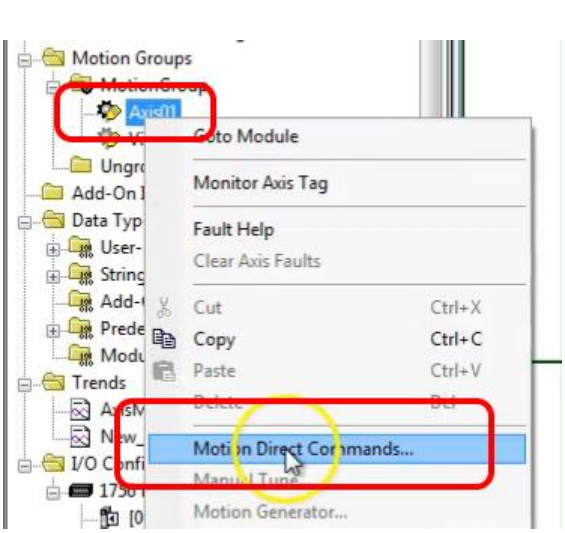
Another use is for calculating the Position Unwind value. Let's start with a 1:1 ratio…

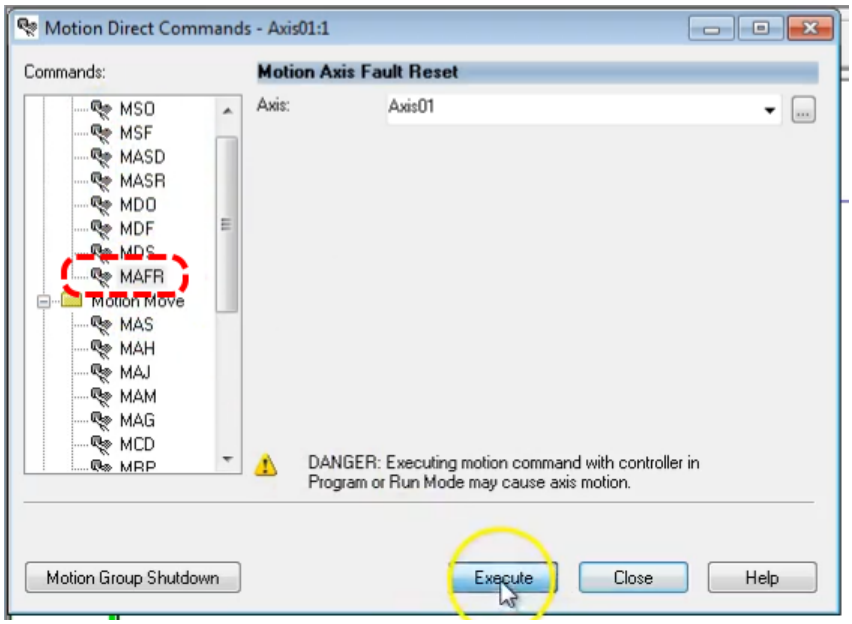



Now we want our shaft to rotate 360 degrees (rollover). Now click calculate.

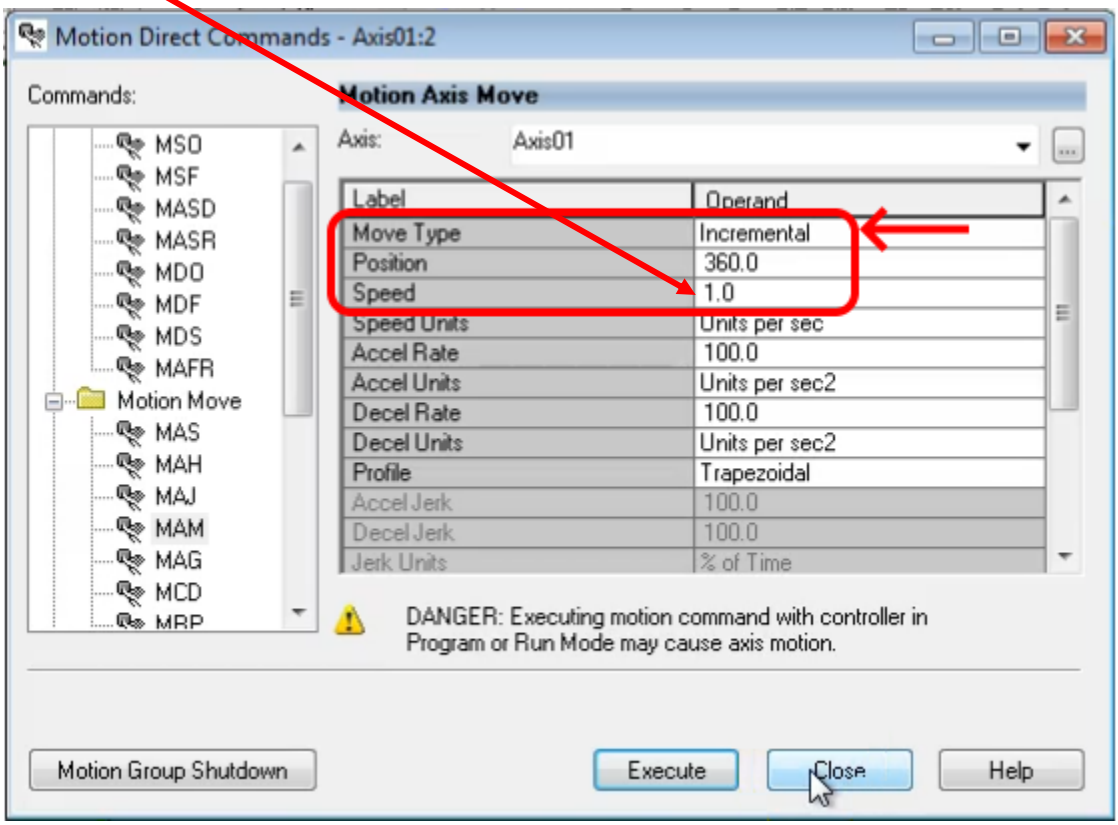



Now download this modified application and run… The shaft is going to turn 360 degrees. To do this access the Motion Direct Commands feature from the motion group Axis01. First issue the Reset Faults command (MAFR).

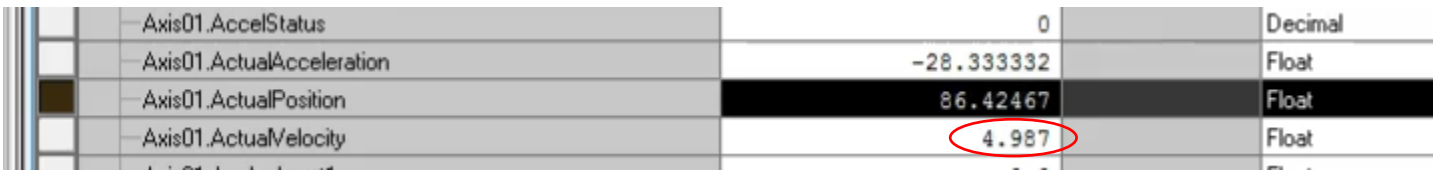



Now issue MSO for custom motor on (the motor will turn on). Next issue MAH for motion axis home. Then issue MSF for Motion Axis Off. Fill out the information as shown below. We want position at 360 and speed at 5. (**the order of commands that worked was Motion Axis Home, Motion Servo On, enter parameters, Execute**).




You can monitor the move in the tag display.



Another example, **Belt Drive**. You want the belt to run 500 mm and you have a 10:1 gear box. So **one revolution of the drive will produce 10 revolutions of the GB. Putting the 10 in the "per Motor Rev." field allows the application to take the GB into account when you click the calculate button**. In the end, **10 turns of the GB output will produce 500 mm of movement**. Servo Motion One (first course) may have some equations that will help here.

Note: if you are using a linear drive you will not get the lower text boxes in the calculate dialog (the unwind terms).

**Unwind** means that after moving for x units, the position is back to the starting point. For example, a servo is turning a wheel on a car. After traveling 360 degrees, it is back to where it started.

## 20. Understand a Simple Mechanical Drive System and Convert

Here we will look at conversion constants a little more. The below table is from course one. This is basicly doing what the calculator does.

Here is an Rotary Axis example:

How to calculate conversion constants

A gearbox of 3:1 ratio used

Pulley attached to the gearbox output is 48:40

The drive constants are set to 1,000,000 Cts/Motor rev

$$\text{Conv Contast} = \frac{(1{,}000{,}000 \text{ Counts/Motor Rev}) (3) (48)}{(40)} = 3{,}600{,}000 \text{ Counts per Unit}$$

Whatever the units are set to....this example does not note the units....but if you need to reference something then says the units are a small box so:

3,600,000 counts per box

So in the servo's attributes tab under Conversions:

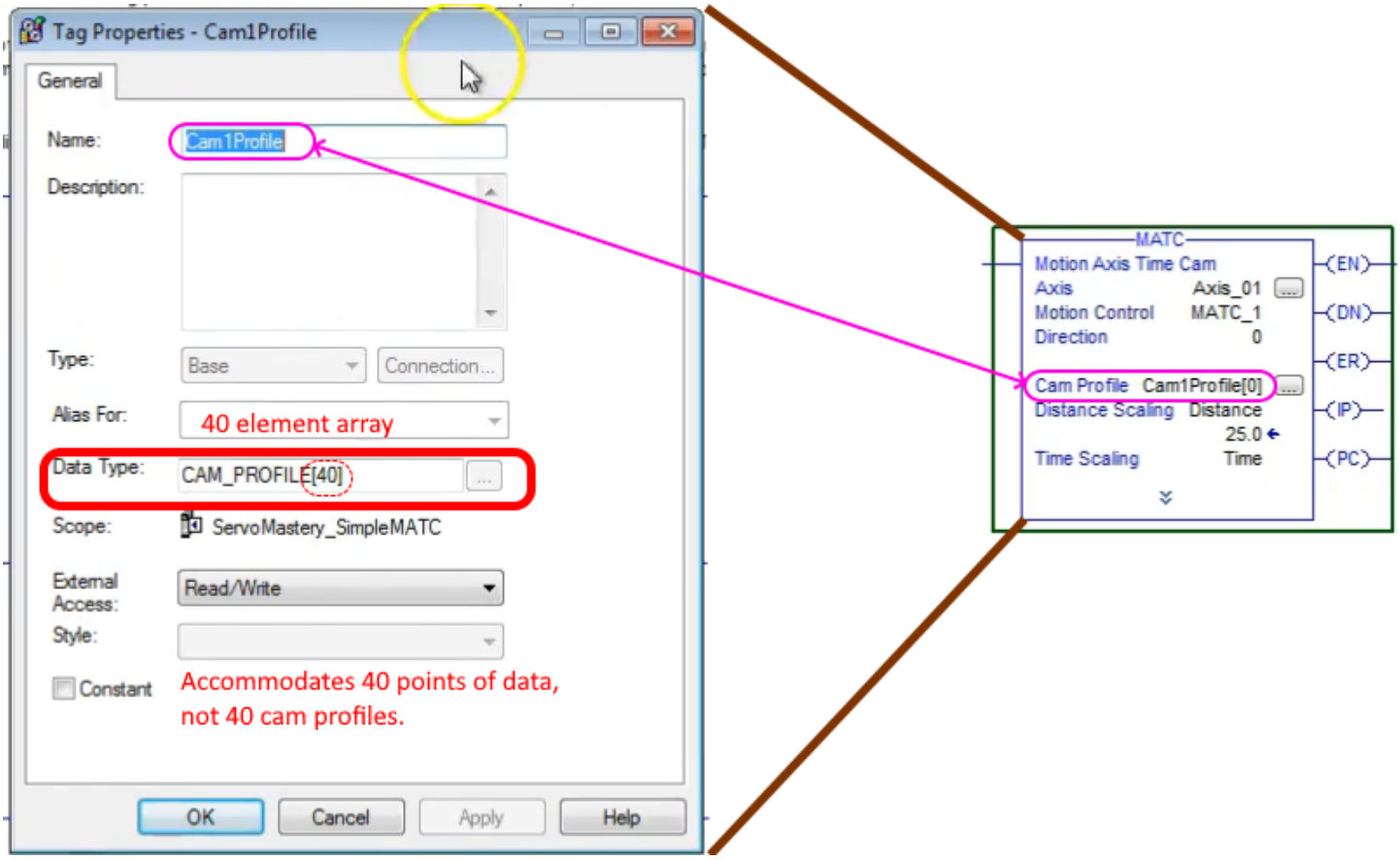
Conversion Constant: 3600000 Counts per box based on 1000000 counts per motor rev

Postion Unwind: 3600000 Drive counts per Unwind based on 1000000 counts per motor rev

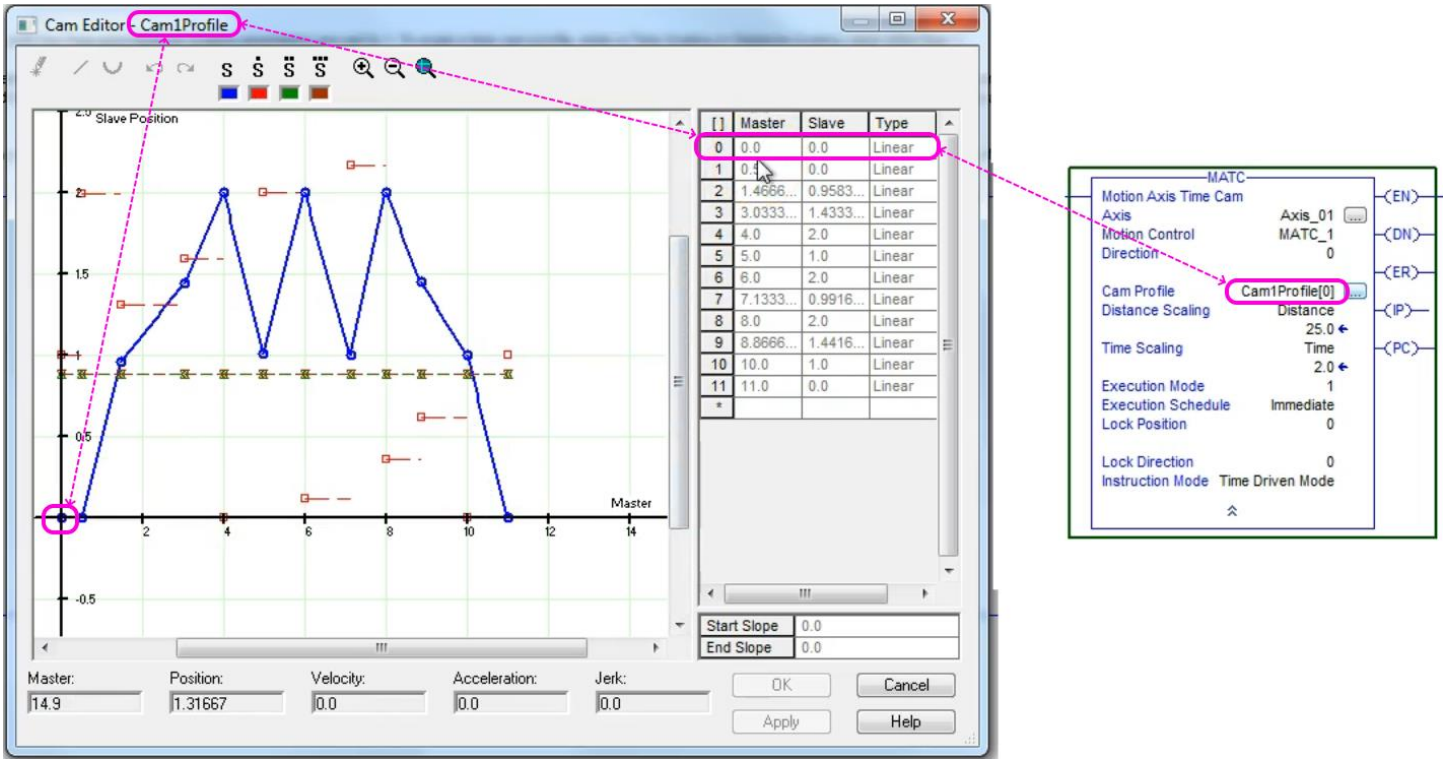
So the units being measured are a box, in this case the system is calculated to one box equals 3,600,000 counts

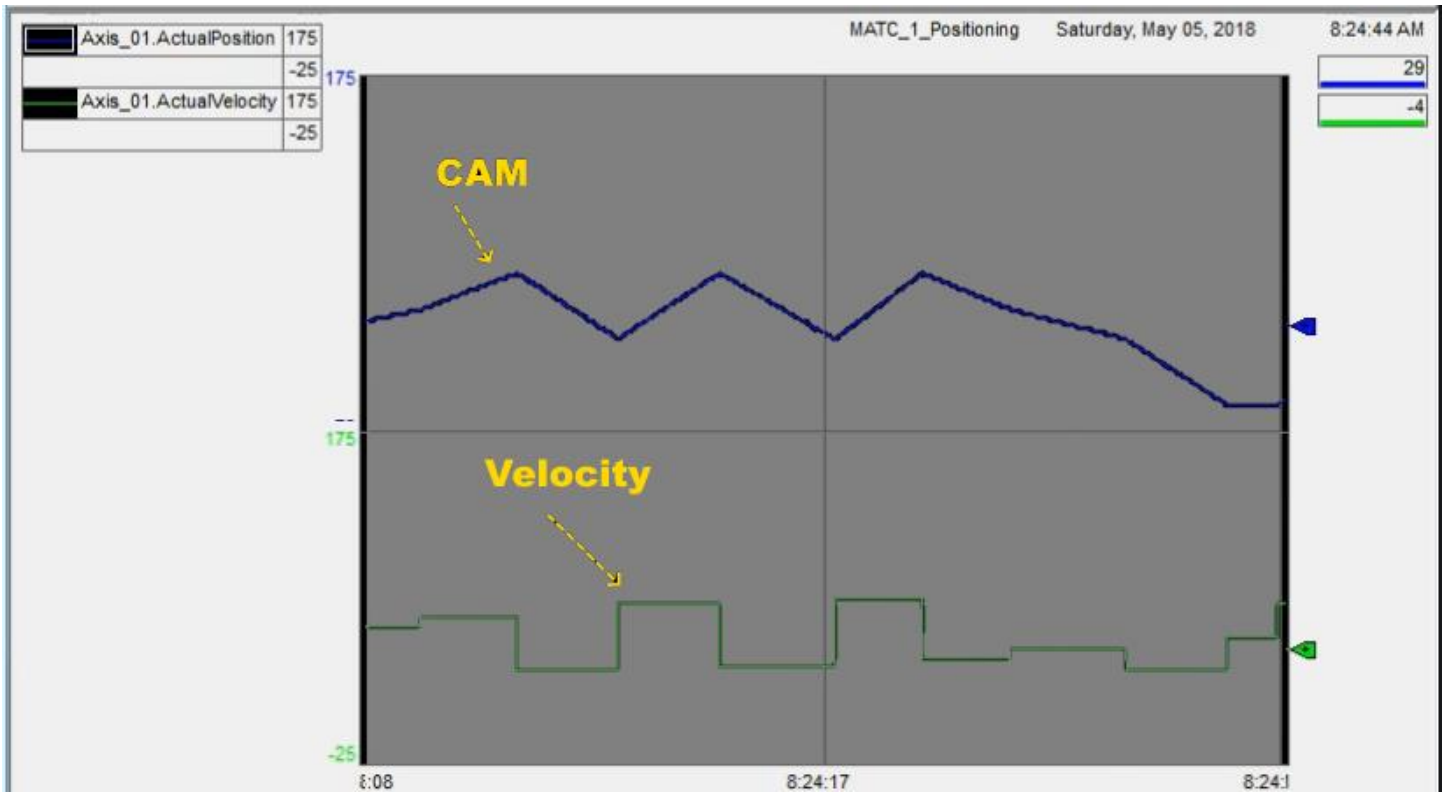
Note: just as the numerator is (3 GB)(48 Pully) the denominator is likely (1 GB)(40 Pully).
The equation is:
(Counts/Motor Rev)*(left side of GB * left side of Pully) / (right side of GB * right side of Pully)

# 21. The Difference Between Linear Cams and Cubic Cams

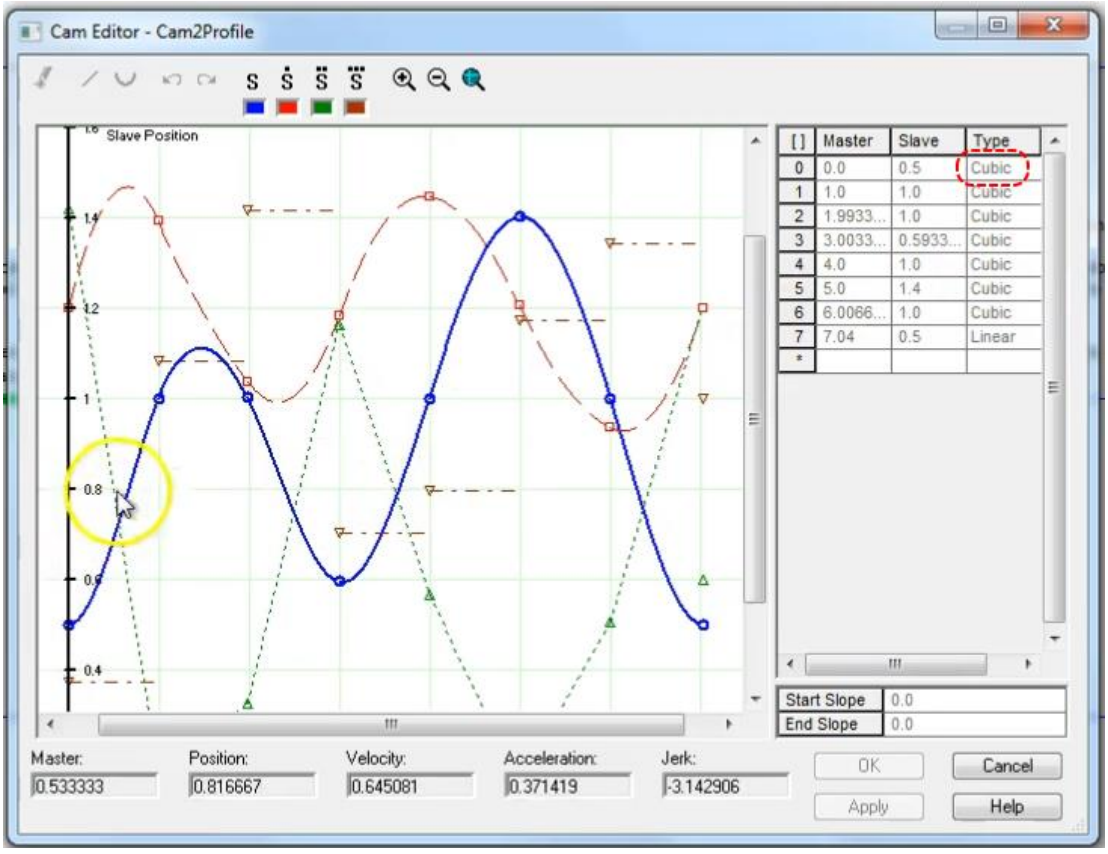



The Cam1Profile parameter reserved space for 40 points but the profile below only uses 12.



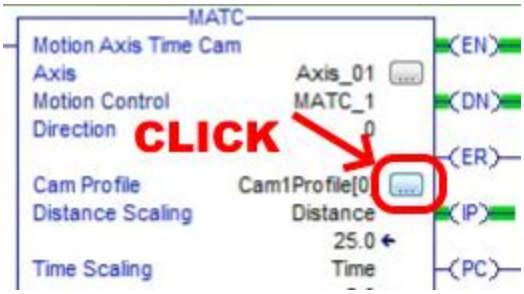


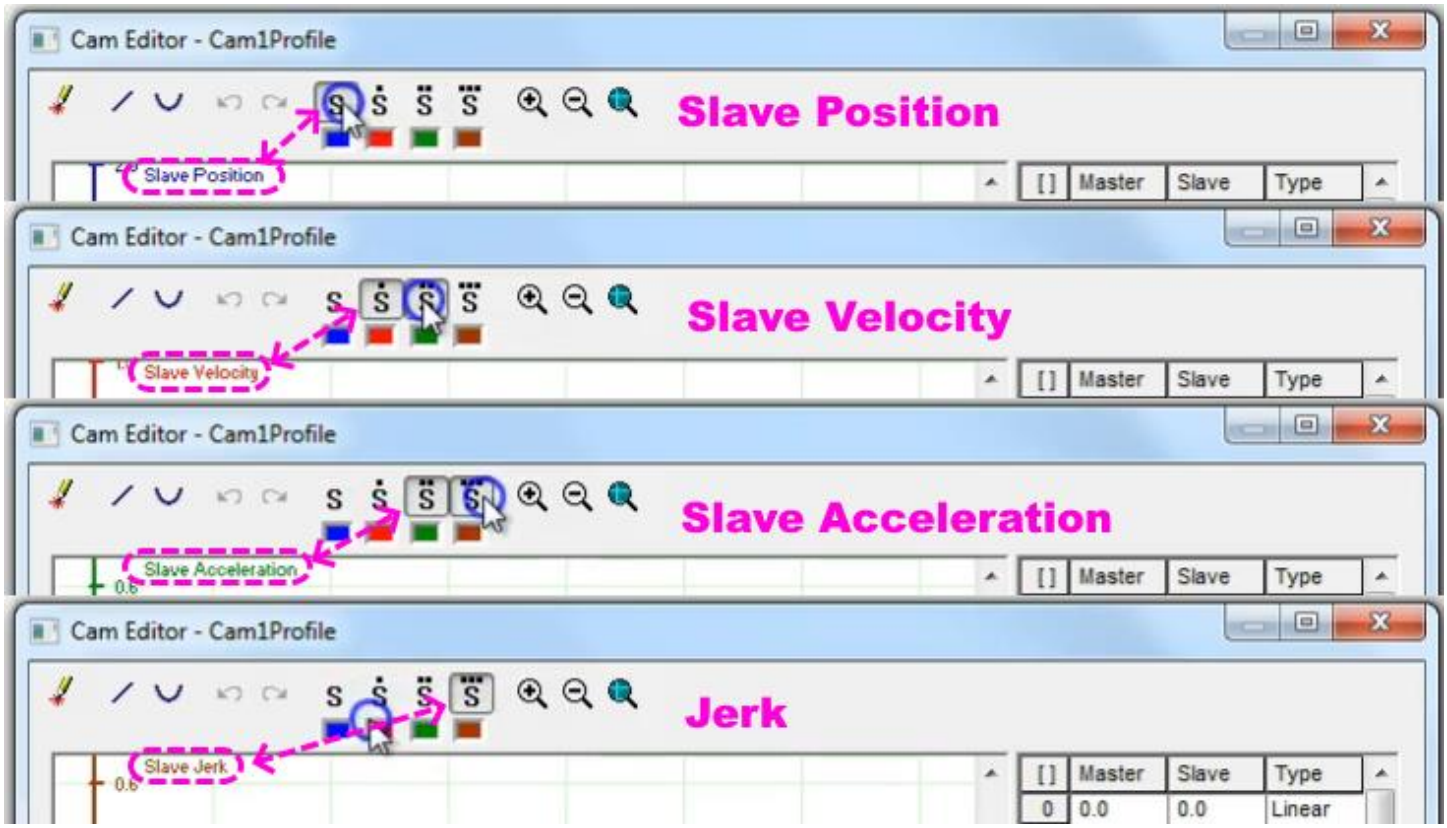

Below we have changed the TYPE of cam from Linear to Cubic. You can see it smooths it out for us. Much more practical.



## 22. Using a Servo Cam Editor

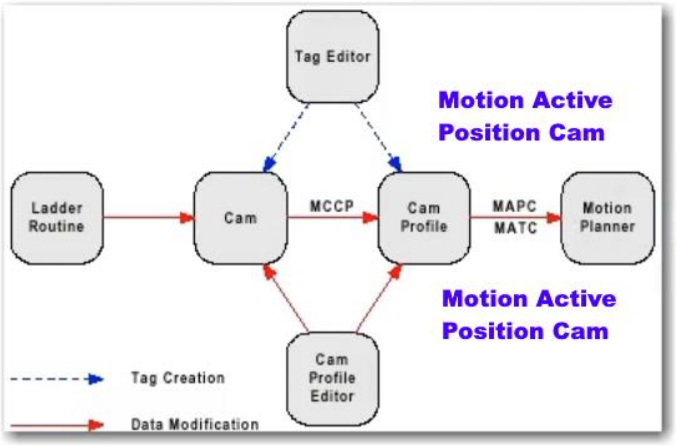
Make sure you are offline or you could invoke unexpected movement.



**Specifying the Cam Profile**

To execute a MATC instruction, a calculated Cam Profile data array tag must be specified. Cam Profile array tags may be created by the RSLogix 5000 tag editor or the MATC instruction using the built-in Cam Profile Editor, or by executing an Motion Calculate Cam Profile (MCCP) instruction on an existing Cam array.

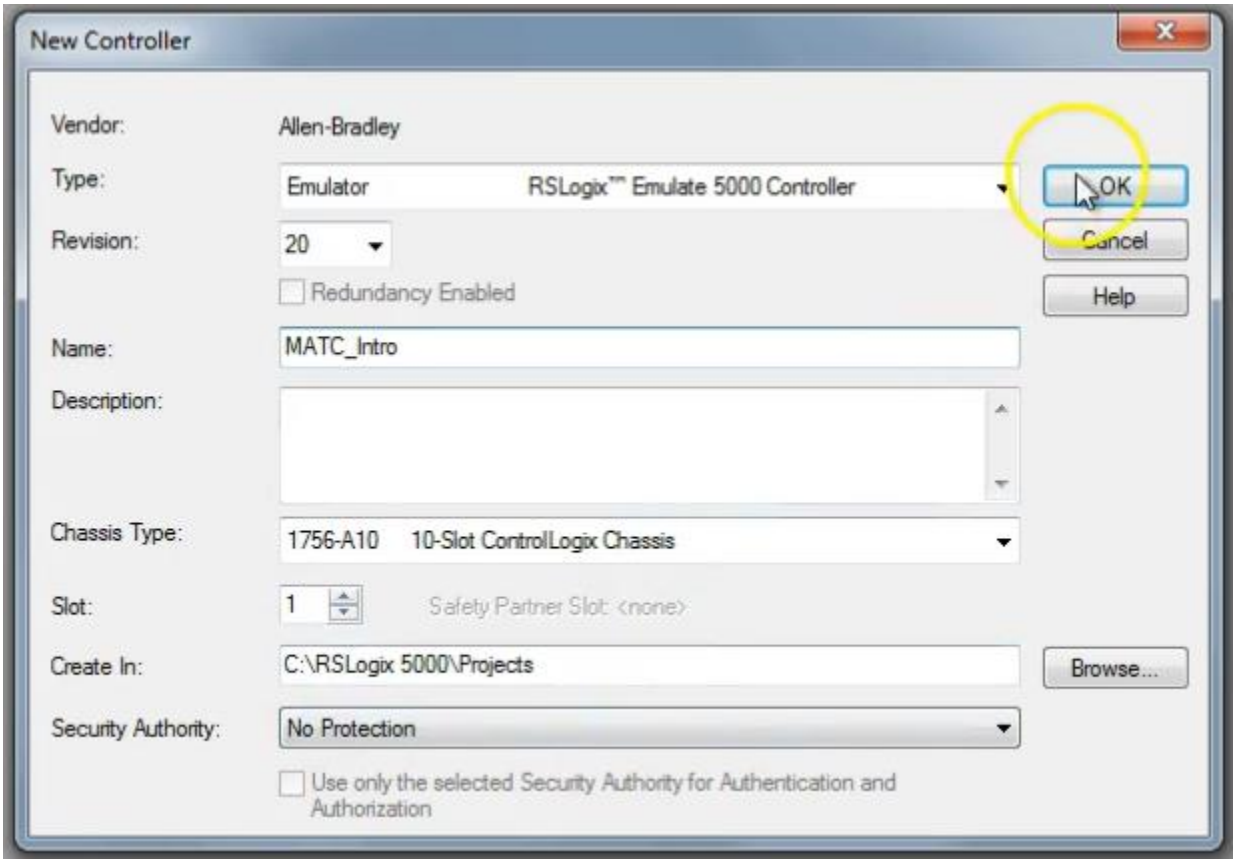



The data within the Cam Profile array can be modified at compile time using the Cam Profile Editor, or at run-time with the Motion Calculate Cam Profile (MCCP) instruction. In the case of run-time changes, a Cam array must be created in order to use the MCCP instruction.

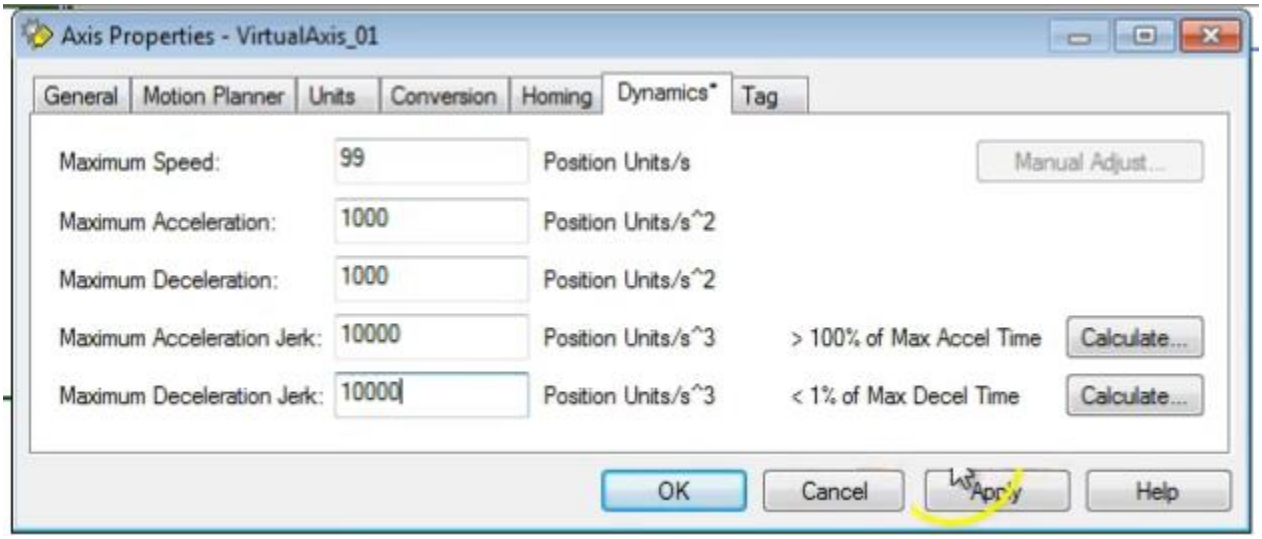
All but the status and type elements of the Cam Profile array element structure are "hidden" from the RSLogix 5000 tag editor. These hidden elements are of no value. The status parameter is used to indicate that the Cam Profile array element has been calculated. If execution of a camming instruction is attempted with any uncalculated elements in a cam profile, the instruction errors. The type parameter determines the type of interpolation applied between this cam array element and the next cam element.

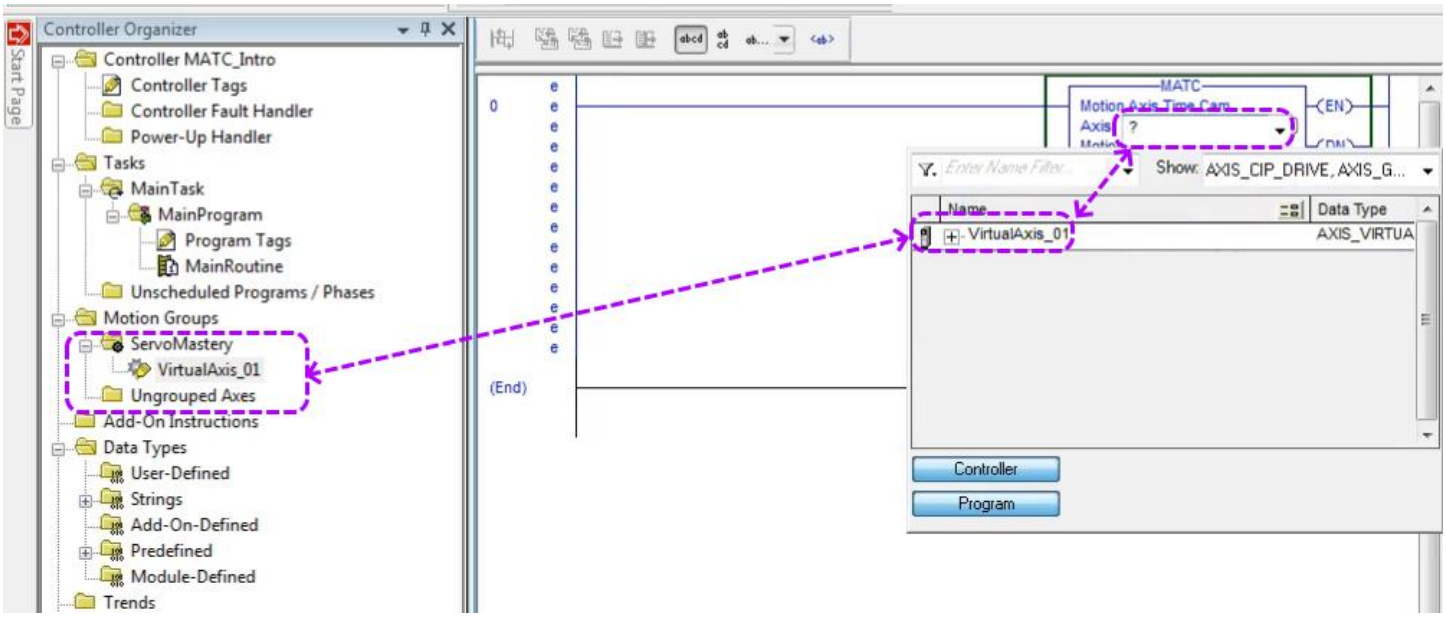
## 24. Motion Axis Time Cam Instruction Intro

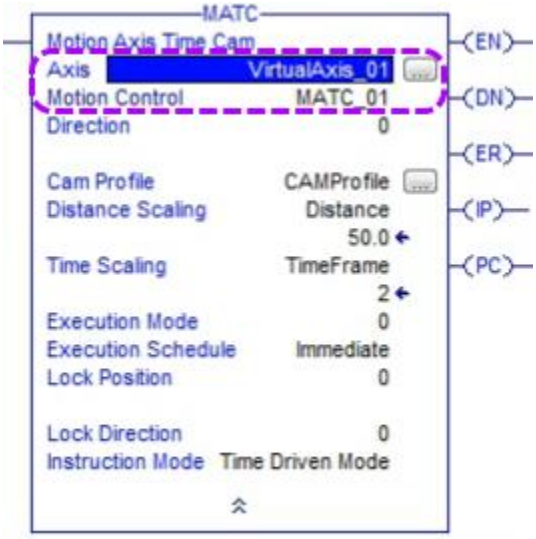
Begin by making a new controller application.

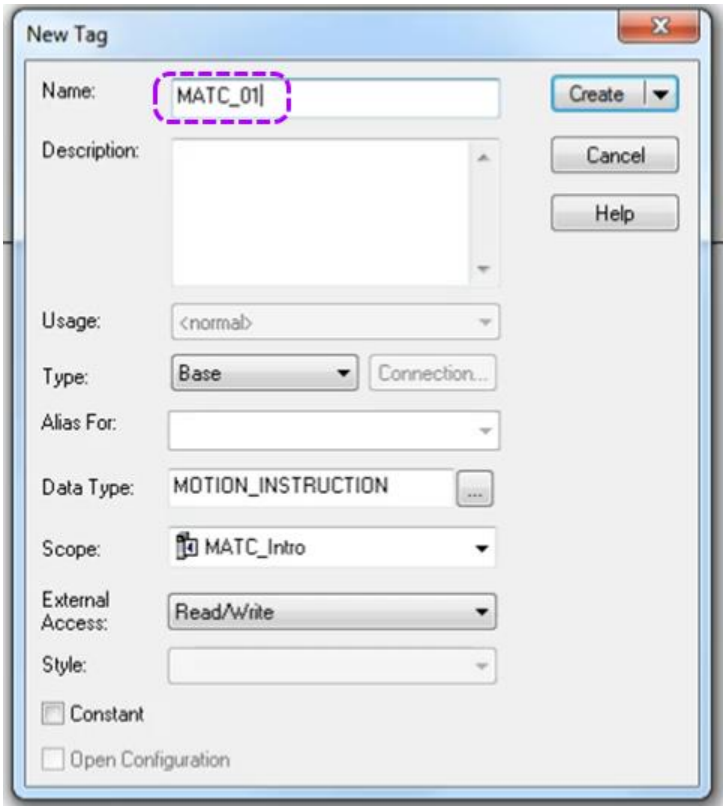


From the Motion Move tab select and add to program the Motion Axis Time Cam function block. Now add a motion group, we will call it ServoMastering. This is a virtual axis, name the tag VirtualAxis_01. In the controller properties Date/Time tab select Enable Time Synchronization, this allows the motion group to run properly. We will run with the below values.



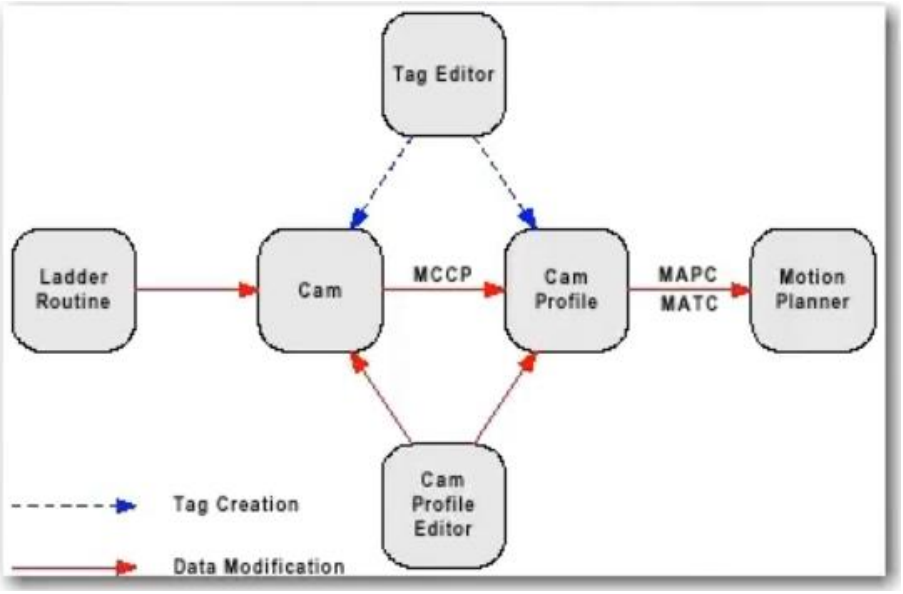


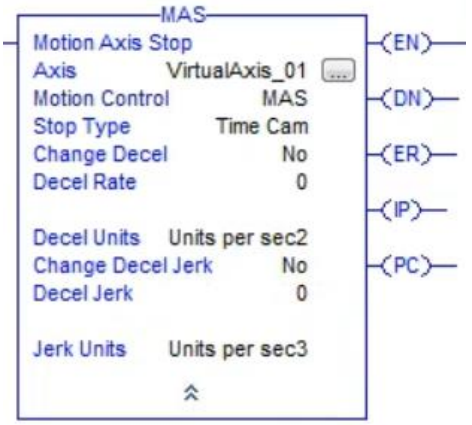

[Note: CAM Profile tag must be an array.]
At this point we have made an axis, now we want to make the motion control. The "**control**" is just the tag name we give it. This value goes in the Motion Control field of the function block. Keep in mind the HELP system is an excellent reference for filling out the function block.

.EN – set when rung goes true, stays set until rung goes false.
.DN – set when the axis time cam instruction is successfully **initiated**, ***not*** when the movement is complete.
.ER – indicates when the instruction detects an error, such as if the axis is not configured. (examine error code).
.IP – ***in process*** bit set on positive rung transition and cleared when terminated.
.PC – ***process complete*** is cleared on positive rung transition and set in Once Execution Mode when the time leaves the time range defined by the current active cam profile.

Refer to troubleshooting techniques discussed in course one. Error code explanations can be found in help file under topic Motion Error Codes (.ERR). Refer to HELP file for a discussion on cam topics such as **Camming Direction**, **Camming in the Same/Opposite Direction**, **Changing the Cam Profile**, **Changing the Camming Direction**, and **Specifying the Cam Profile**.



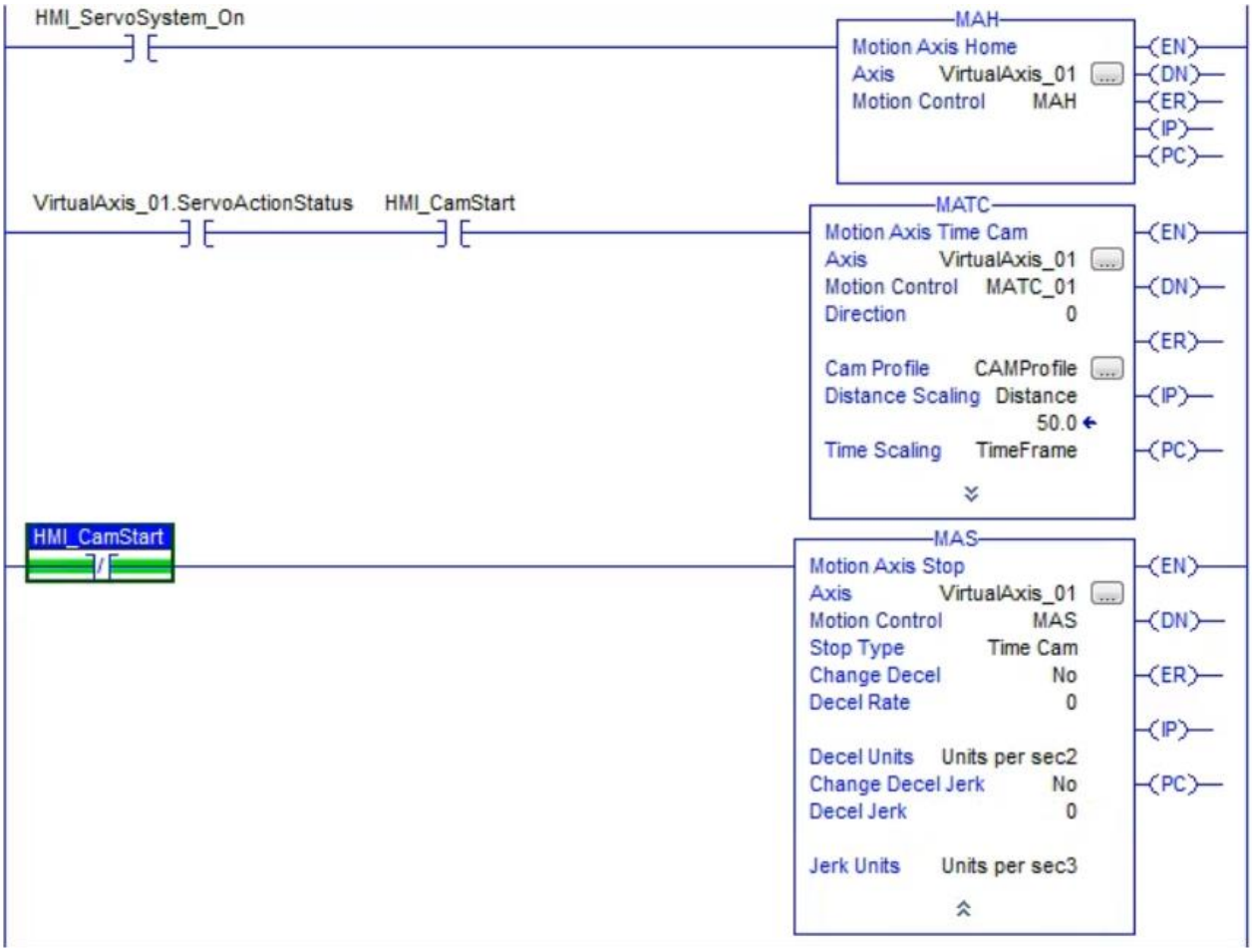




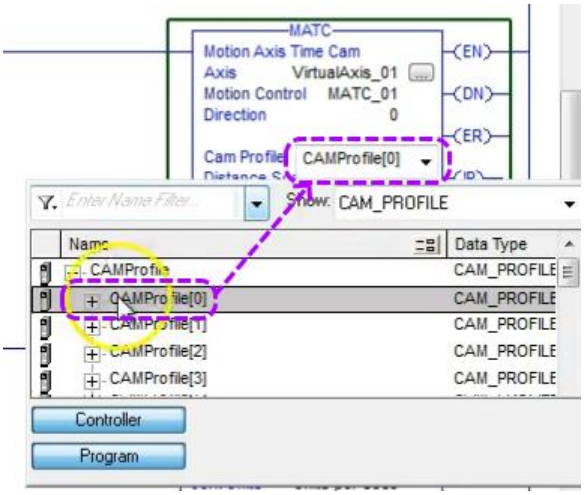

The above figure illustrates the order of operations. We will discuss MCCP coming up.

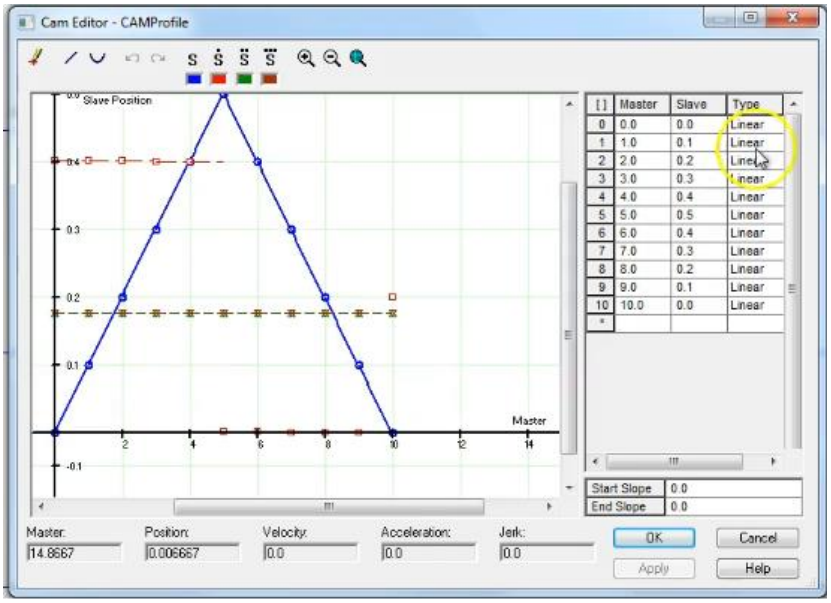
**Now we will look at how to turn the axis on. The only way to do this with a virtual axis is to use the homing command, MAH**. ServoActionStatus means the servo is on (first course). Now we want to start the cams.

You also want to stop the servo (MAS), do not leave the bit on. There are different types of stop. Here we will use Time Cam. Note that there could be other motion going on in the system so we have to select carefully.

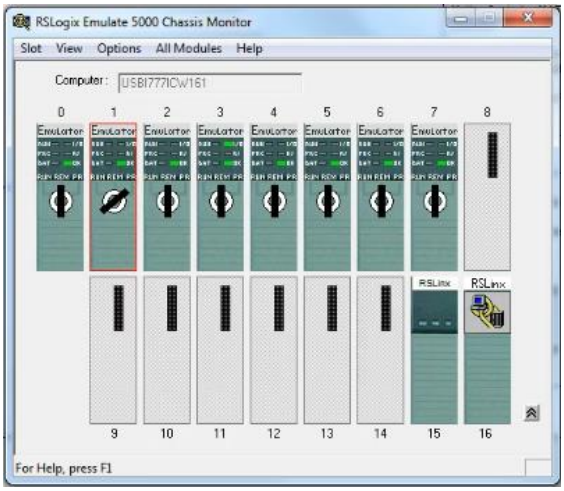


Now we want to create a cam profile and associate it with our MATC block. We are replacing the function MATC Cam Profile entry (started out as CAMProfile) with the first instance of that profile. Remember, CAMProfile is a control array. (I think he means replace CAMProfile with first element of CAMProfile control array CAMProfile[0].)
Now we open the cam editor and enter our cam profile (click Apply).

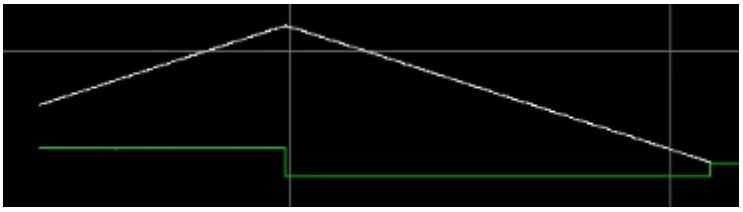


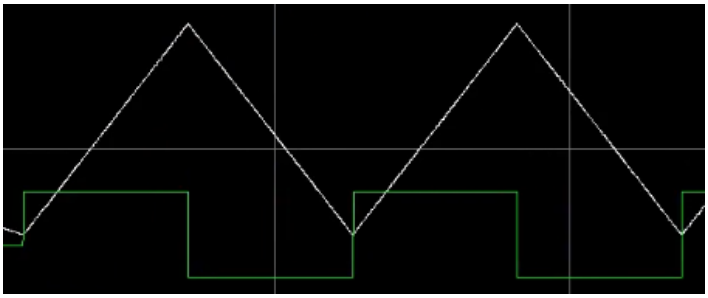

Now create a trend on the position and velocity parameters. Then download to the emulator. Make sure it is running.

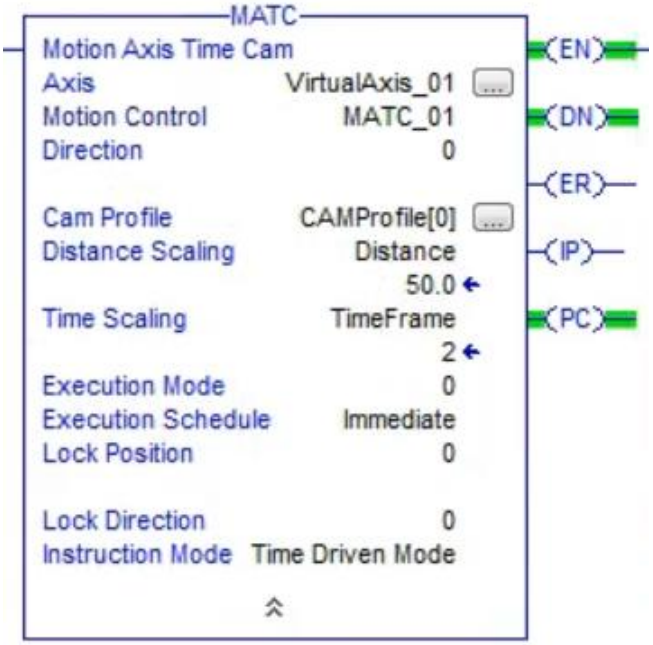



Put the emulator in RUN mode. Toggle HMI_ServoSystem_On tag and then set HMI_CamStart to on. Next run the trend. We had the cam set to run once as opposed to continuous.



Now we are going to add a contact closure with the cam tag MATC_01.IP (in process). Toggle HMI_CamStart on/off and then set MATC_01.IP on (?). Now open the trend. We see that the cam cycles, runs continuously. We can adjust the distance scaling, try 100 and notice that the velocity has to increase to keep up with the cam timing.

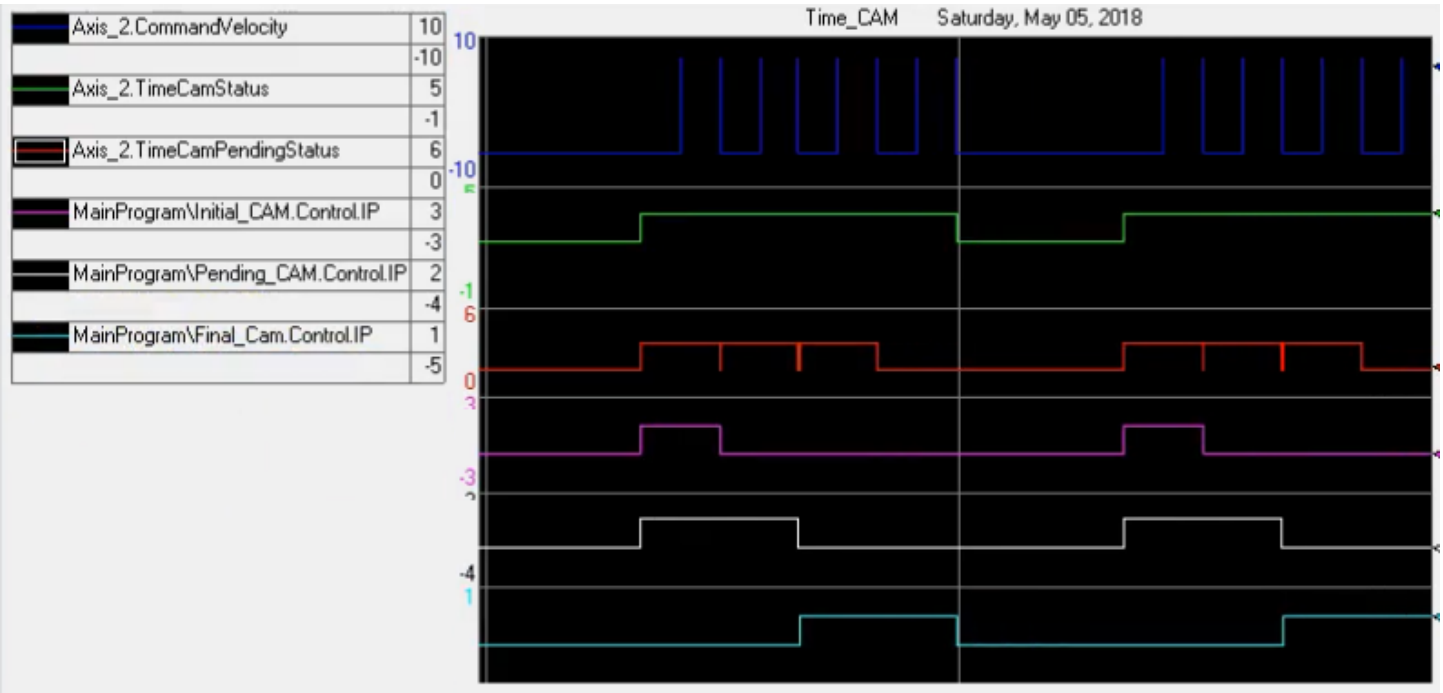




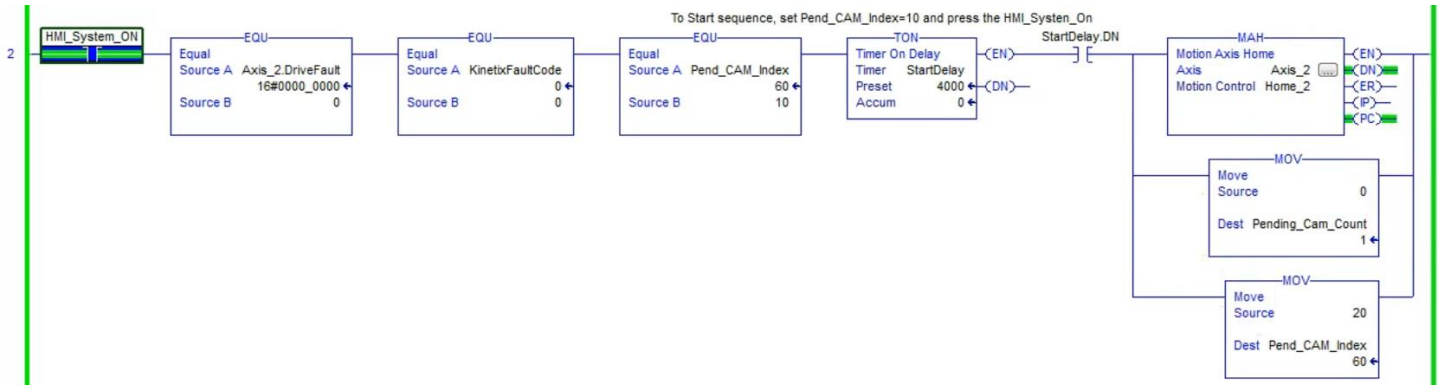

**The time cam describes the distance you want to travel in the amount of time you want the move to take.**

## 25. Breakdown the Difference Between an Immediate and Pending MATC Use

This topic falls under time cams. It is a different scenario. We have a small state machine with a time cam which itself runs another time cam twice. On our MATC function block (one of them) we see the parameter Execution Schedule set to pending (this is the second, the one that runs twice). This allows us to use the .TimeCamPendingStatus tag as a counter. At the last rung "it" is reset to another sequence. The program being used here is Timing_CAMs in TimingCAMs.ACD (exported as a graphic in same directory as SMM2).

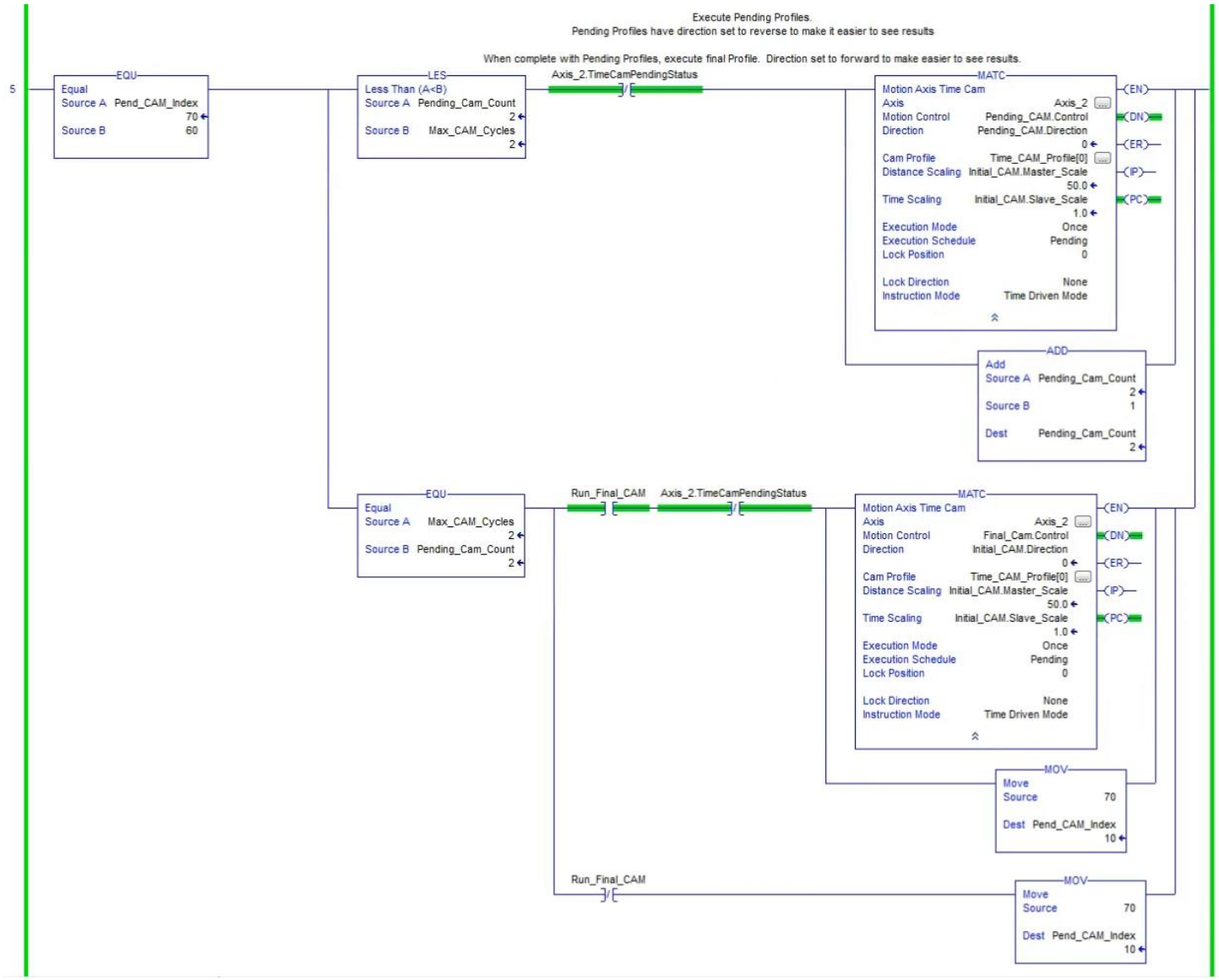



Sequence (blue) runs four times, waits 4 seconds, re-homes,

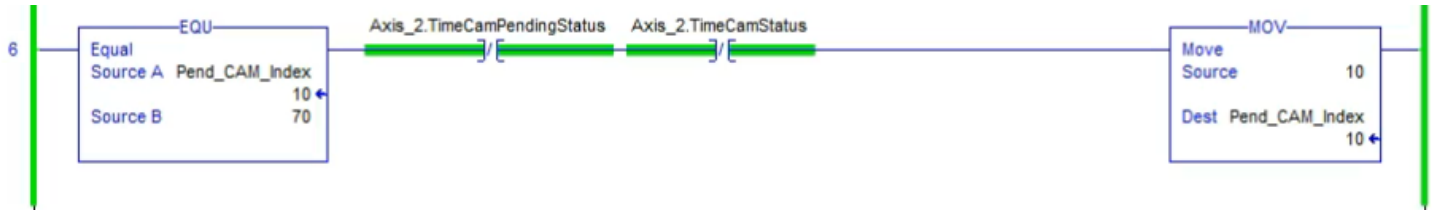



Then in a lower rung it runs the first cam …


Next it runs the pending cam twice …

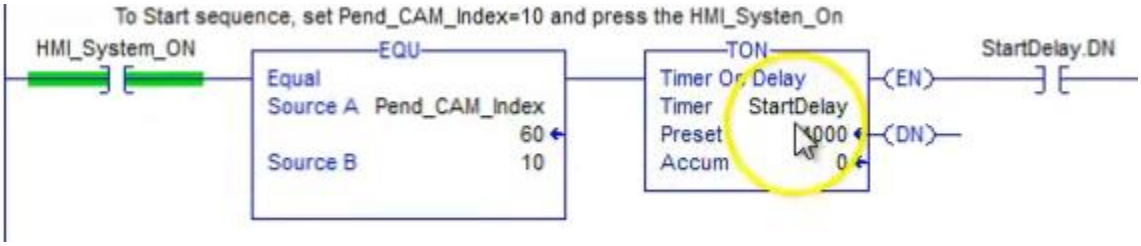



And concludes with …

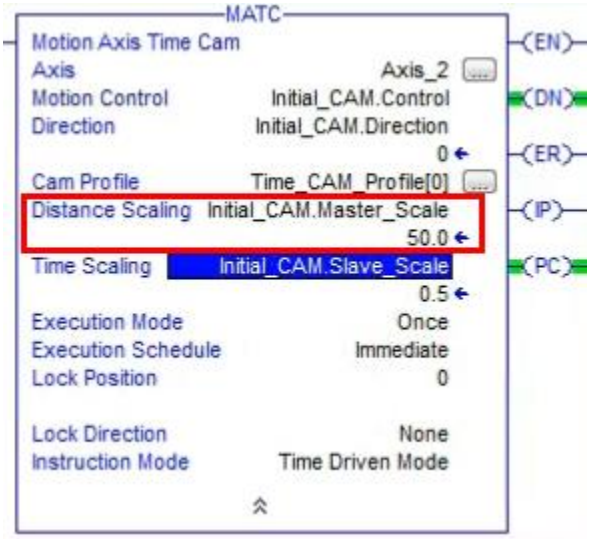


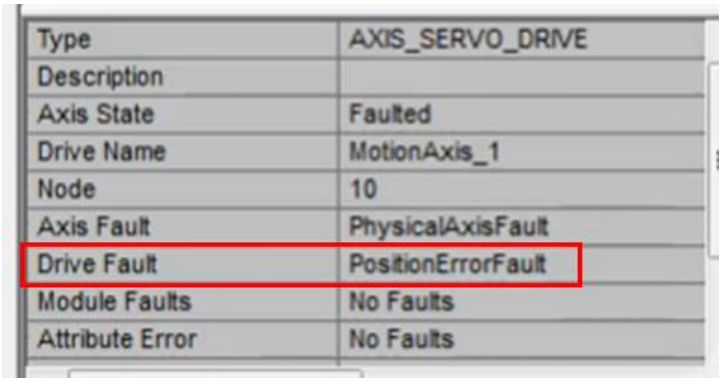

The idea here is to notice the difference between pending and immediate and how they affect the outputs.

## 26. MATC Logic Used with a Virtual Axis With Trouble Shooting



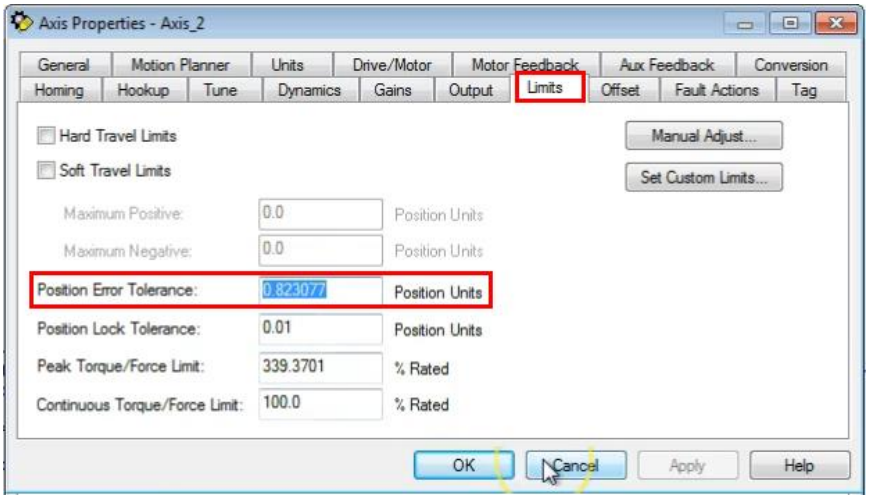

In this lecture we will be discussing this key area of code.



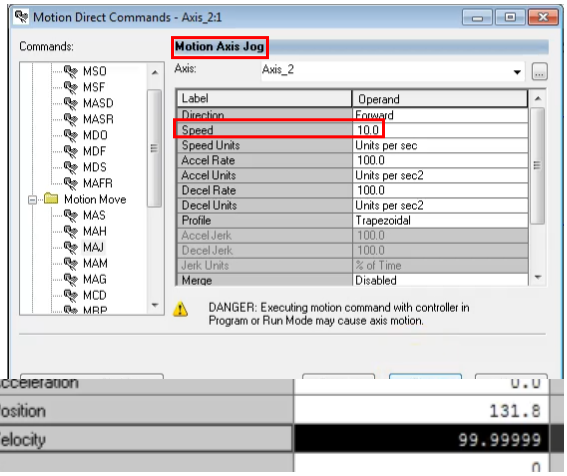
| | |
|---|---|
| Type | AXIS_SERVO_DRIVE |
| Description | |
| Axis State | Faulted |
| Drive Name | MotionAxis_1 |
| Node | 10 |
| Axis Fault | PhysicalAxisFault |
| Drive Fault | PositionErrorFault |
| Module Faults | No Faults |
| Attribute Error | No Faults |

To begin we start the application using the HMI_System_On tag. We are using the hardware application (not emulator) at the moment so we can create a fault by changing the MATC Distance Scaling parameter to 50 causing the motor to have to move too fast to keep up with the cam. This is a position error fault. **A position error means you have tried to do something the servo is not capable of**.




In this case we have set really tight limits (figure to left).

If you are wondering if the error is with something like the limits or with the motor, amp, or cable you can reset and try jogging the motor. If it works you know those things are okay.







Next we will try increasing the position error tolerance to 10.xxx, reset the fault, and run. It does run and does not generate a fault.

Make sure you try this type of troubleshooting before changing out the motor.

The errors we had on the last page while running the hardware will not appear when we do the same thing on the virtual axis. The virtual axis does not have those position error tolerances associated with it because there is no hardware.

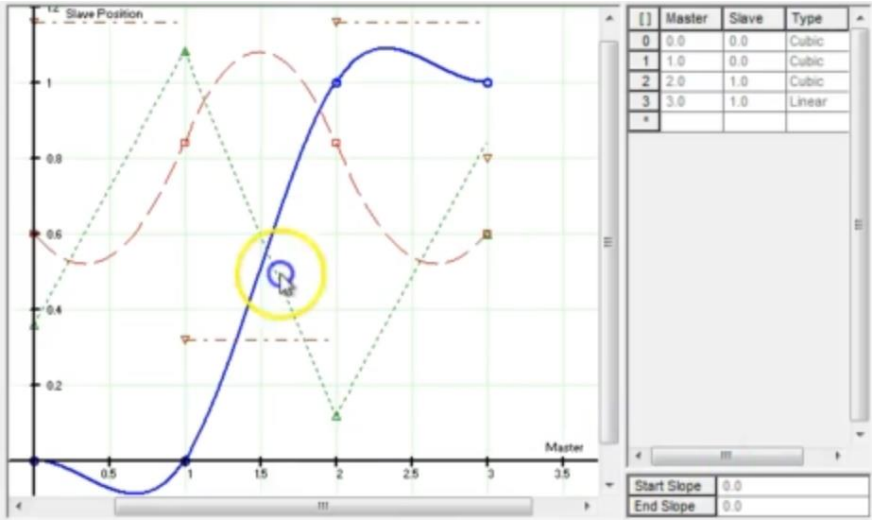
**When using a MATC cam you are basically specifying a length of time and a distance to cover in that timeframe**. The software will drive the motor to the required speed. Be aware of the physical aspects of the settings you enter. And keep in mind the virtual axis will not generate many of the faults that the hardware will.

## 27. MATC Scaling Explained To Limit Confusion

Do not issue a command unless .ServoActionStatus is ON, otherwise your MATC would generate an error.


In above graphic, by default, both the time and distance scaling parameters are set to 1. To scale a time cam profile enter a Time Scaling or Distance Scaling value other than 1. Increasing the time scaling value of a cam profile decreases the velocities and accelerations of the profile. To maintain the velocities and accelerations of the scaled profile approximately equal to those of the unscaled profile, the Time Scaling and Distance Scaling values should be equal. For example, if the Distance Scaling value of a profile is 2 then the Time Scaling value should also be 2 to maintain approximately equal velocities and accelerations during execution of the scaled time cam. Important: decreasing the Time Scaling value or increasing the Distance Scaling of a time cam increases the required velocities and accelerations of the profile. This can cause a motion fault if the capabilities of the drive are exceeded.

Before we run lets review what transpires in this cam profile. Keep in mind this is a time cam. When a cam profile is specified in a MATC instruction the Master Coordinate Value defined by the profile array takes on time units in seconds and the slave units are the slave values which take on the units of the slave servo. Contrast this to the fact that the time and distance scaling parameters are unitless, the values are basically used as multipliers to the cam profile. So the cam profile is relevant but at the same time the distance and time are more or less multipliers. This is what controls the velocity of the servo. The same is true for the speed and acceleration. So say we

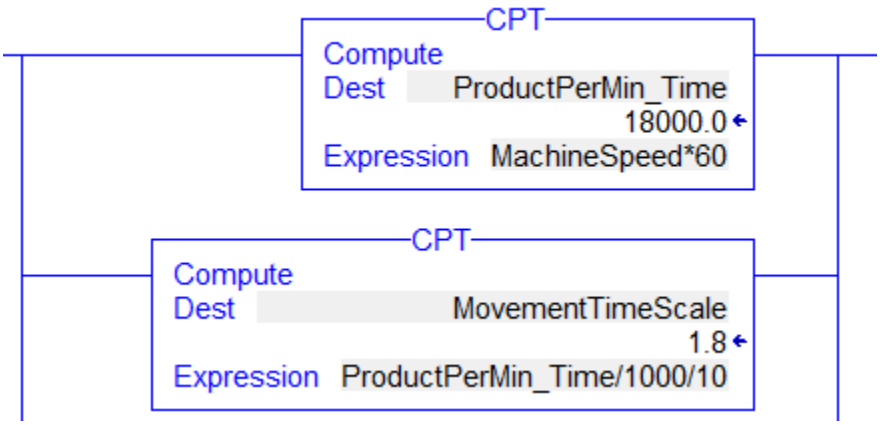


have a distance of 10 and a time of 1, what will happen? Toggle HMI_StartCam. We have the execution mode set to zero so it will run one time. Likewise, having execution mode set to continuous (1) will run the cam multiple times.


*Figure 2 one possible cam start logic.*

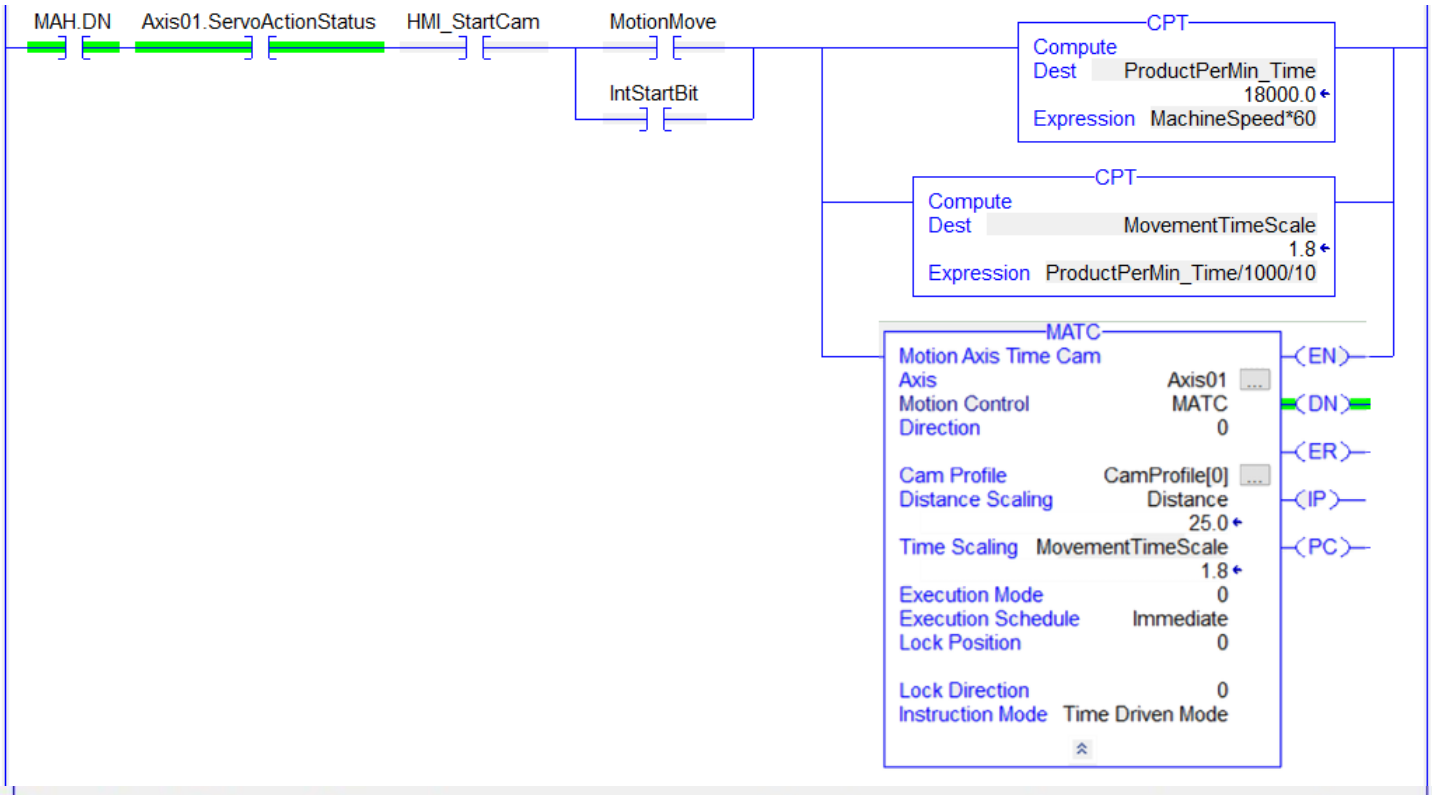
| MATC bits |
|---|
| .EN – set when rung goes true, stays set until rung goes false. |
| .DN – set when the axis time cam successfully **initiated**. |
| .ER – indicates when the instruction detects an error. |
| .IP – ***in process*** bit set on positive rung transition. |
| .PC – ***process complete*** is cleared on positive rung transition. |

## 28. MATC Instruction With A Natural Move Command




In this lecture we will add a 'rolling time change' to our time cam. We will add a couple of compute statements, one will be **ProductPerMinTime** (REAL) and the other **MachineSpeed** (DINT). We multiply MachineSpeed by 60 since we are working in seconds. Parameter **MovementTimeScale** will be REAL.

So now MotionMovementTime is used in the MATC block in the TimeScaling field. We call this a rolling system. And we initialize MachineSpeed at 250 (items per minute). You can try changing the distance, this will change the velocity of the servo motor.




By default, both the Time and Distance Scaling parameters are set to 1. To scale a time cam profile, enter a Time Scaling or Distance Scaling value other than 1.

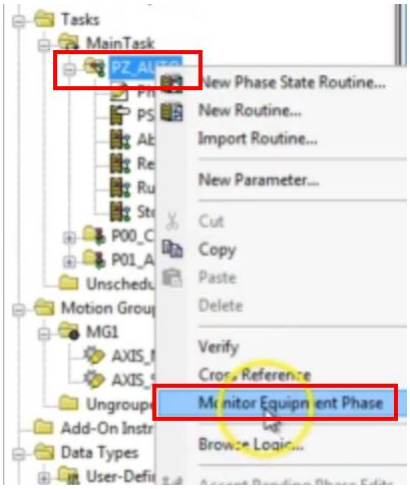
Increasing the Time Scaling value of a cam profile decreases the velocities and accelerations of the profile, while increasing the Distance Scaling value increases the velocities and accelerations of the profile. To maintain the velocities and accelerations of the scaled profile approximately equal to those of the unscaled profile, the Time Scaling and Distance Scaling values should be equal. For example, if the Distance Scaling value of a profile is 2, the Time Scaling value should also be 2 to maintain approximately equal velocities and accelerations during execution of the scaled time cam.

Important:
Decreasing the Time Scaling value or increasing the Distance Scaling of a time cam increases the required velocities and accelerations of the profile. This can cause a motion fault if the capabilities of the drive system are exceeded

SimpleMATC_WithMove.ACD

# 29. Servo Motion Mastery PCAM Intro

The topics of this lecture are the basics of a **position cam** and how to calculate a position cam. Here we are using the phase manager.



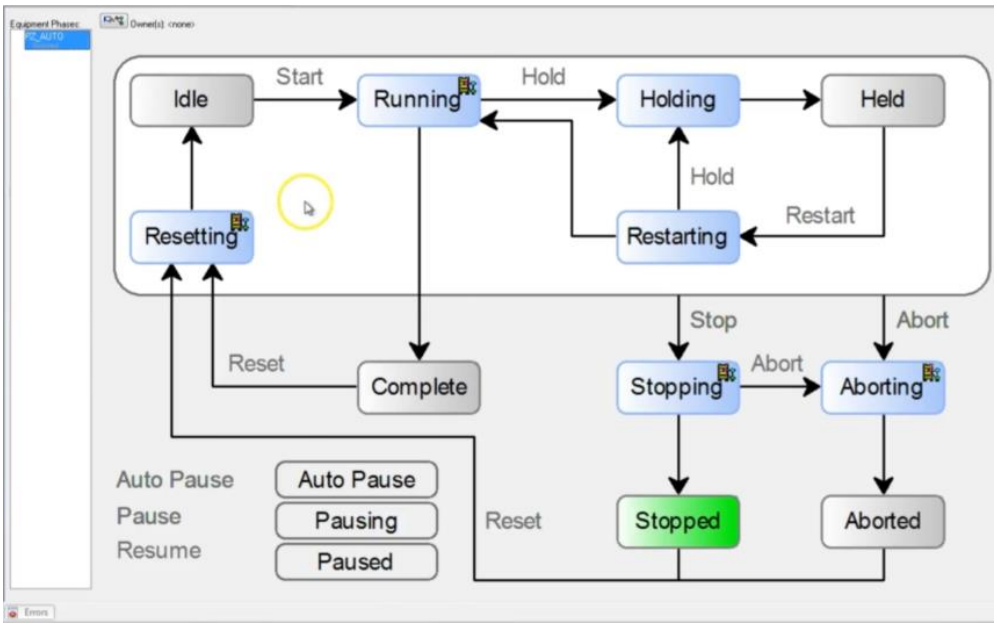




Figure 3 elements of a phased system.

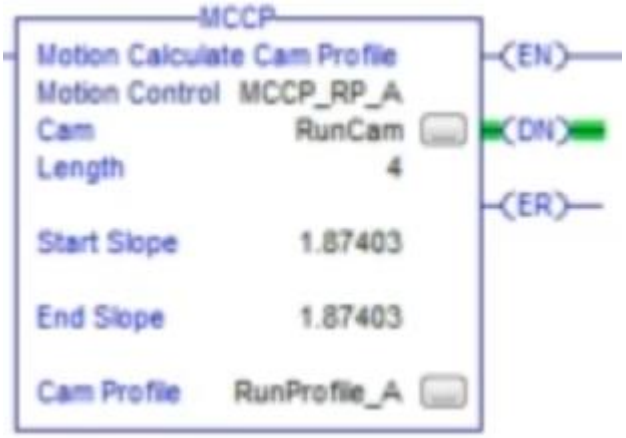
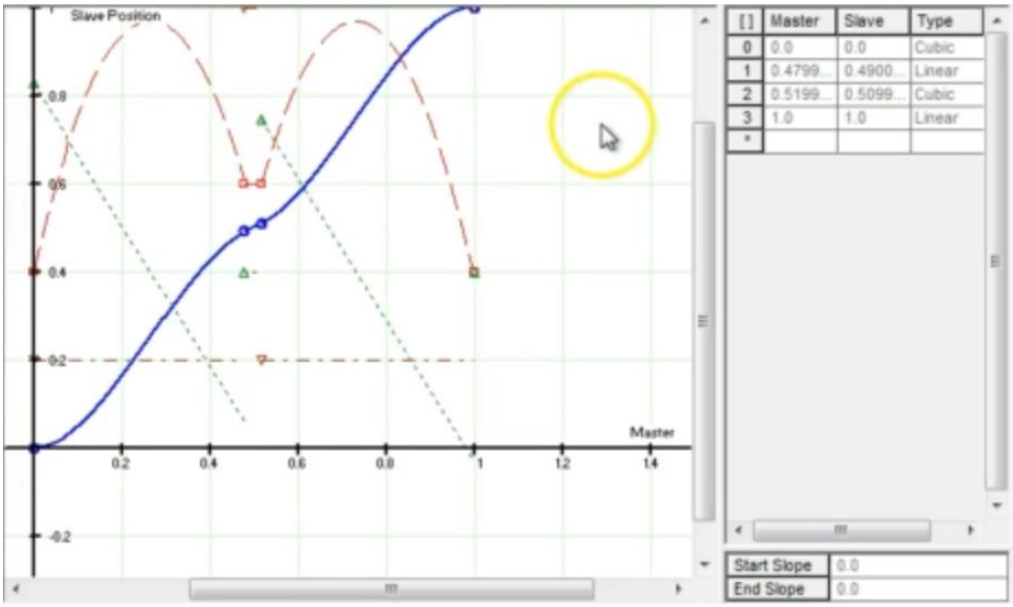
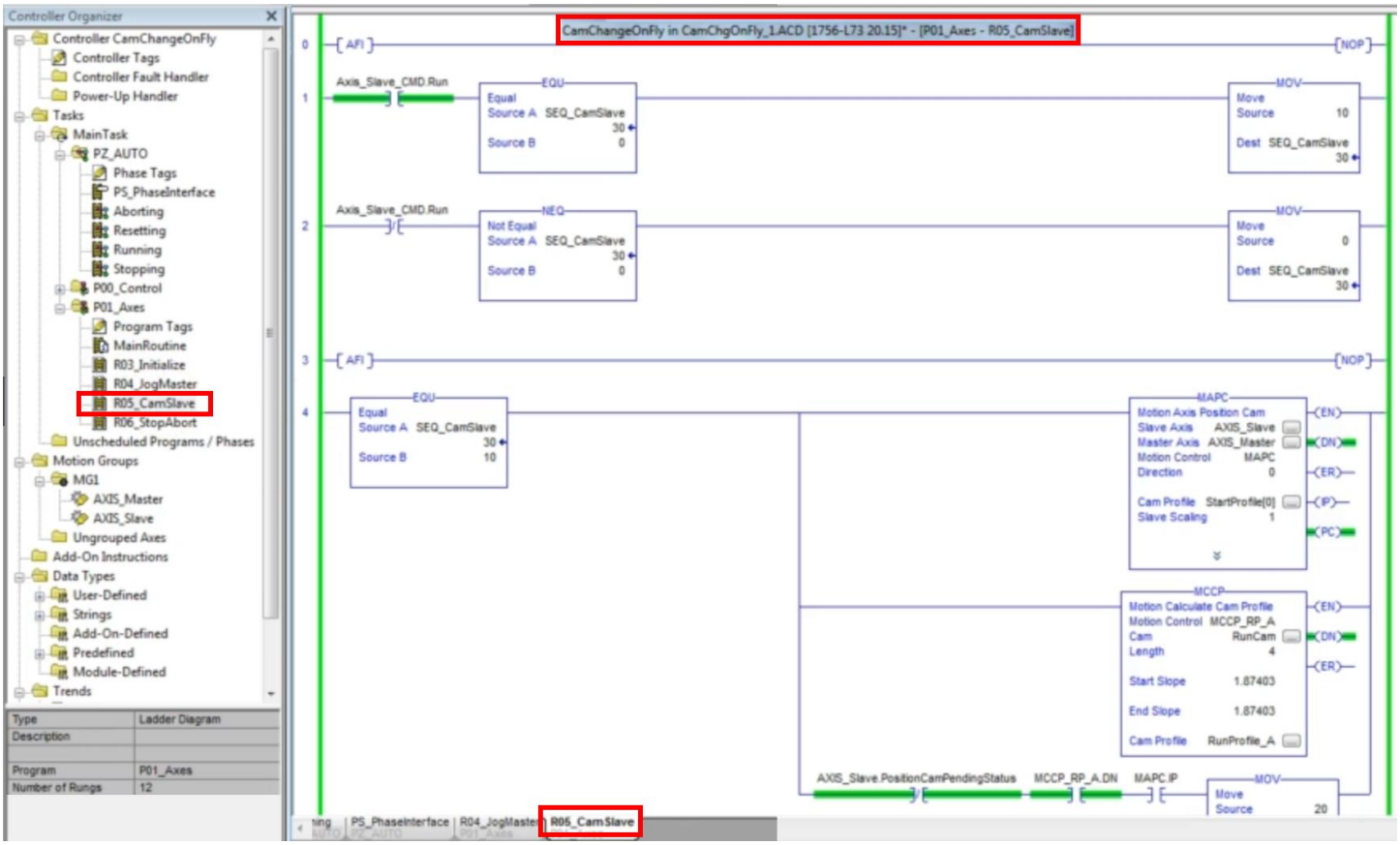




Figure 4 position cam Motion Calculate Cam Profile function block.

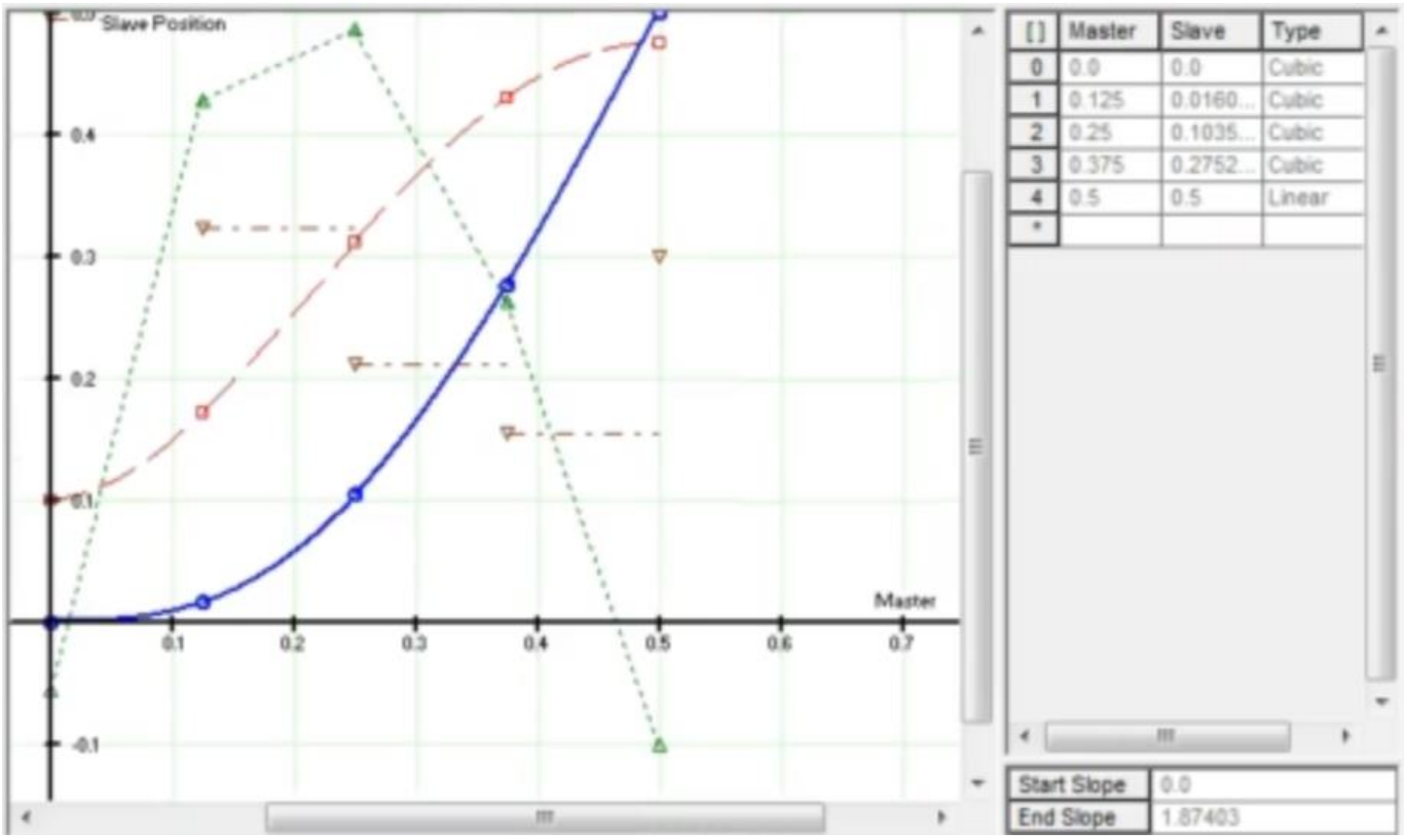

Slave axis, in blue, is running relative to the master in red. Note we are plotting velocity.



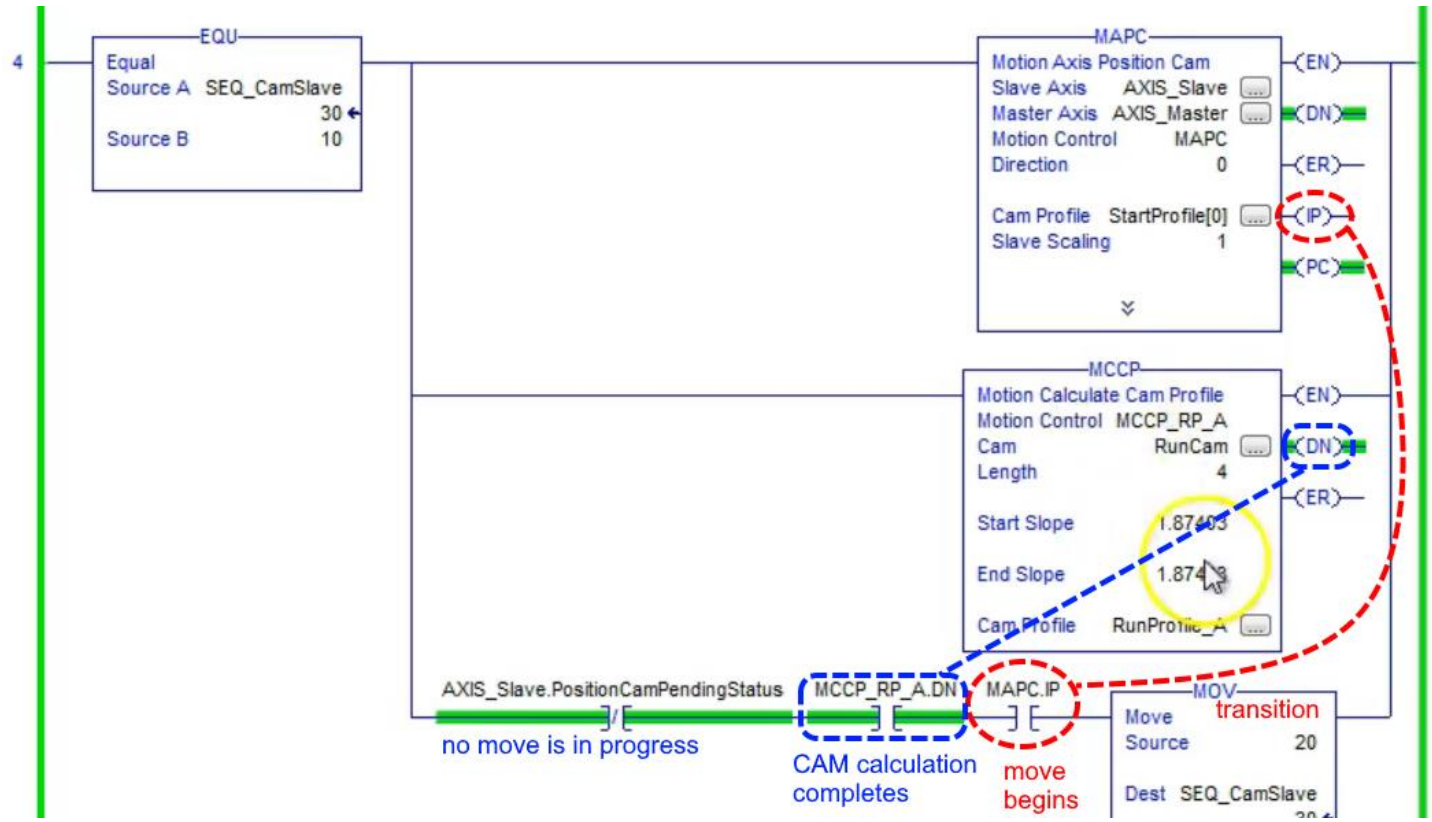

The MCCP cam being implemented, start and end slopes are both 1.87403 (on Motion Calculate Cam Profile [MCCP] function block, not cam table). This cam profile will be used by the Motion Axis Position Cam (MAPC) function block (not clear why they are not on separate states with MCCP coming first).

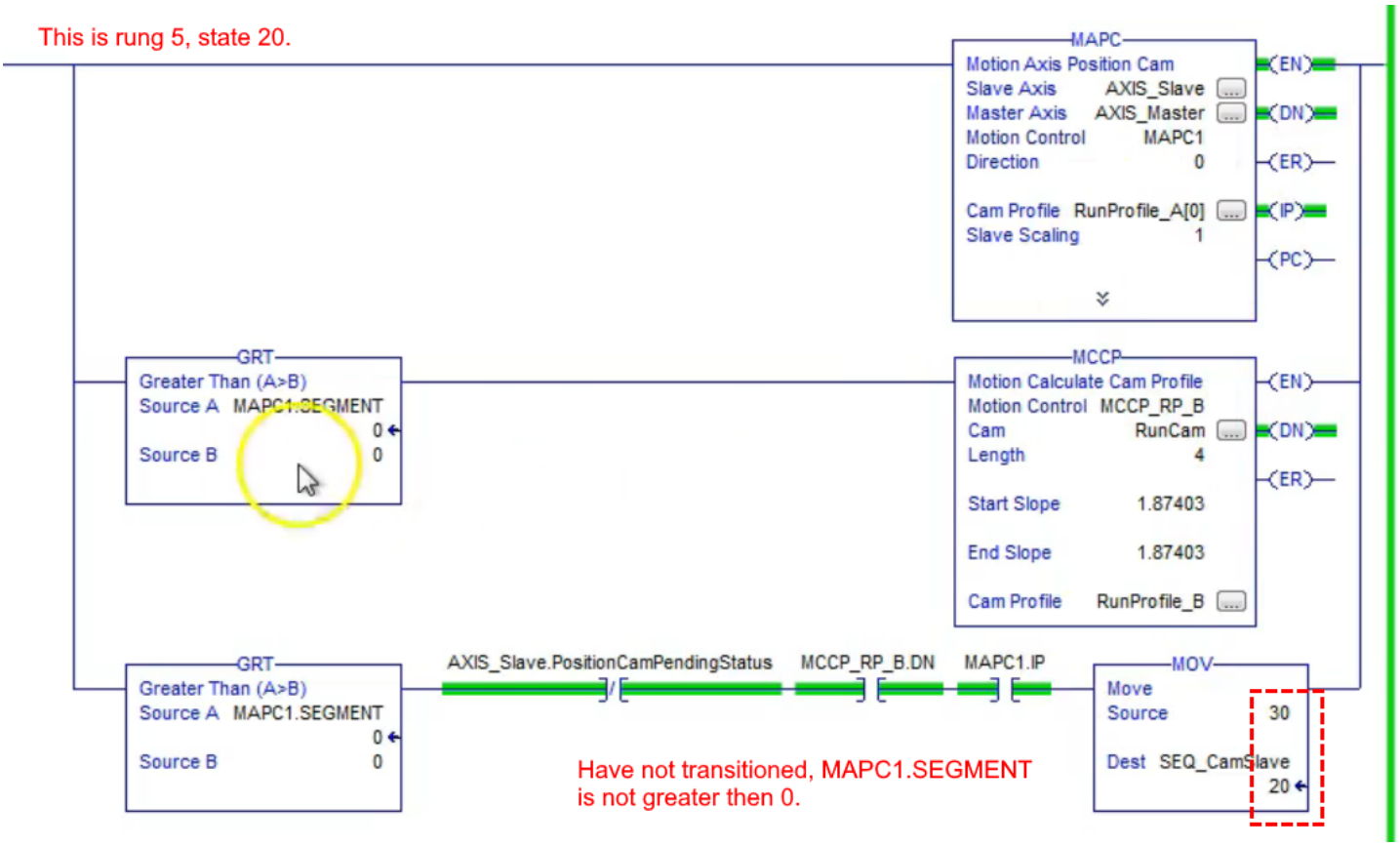



First and foremost, the program calculates the cam using the MCCP block and that information will be used by the MAPC block. Click the ellipse in the MCCP block beside CAM / RunCam and the Cam Editor appears (shown above). Note the Start Slope and End Slope entries at MCCP cam profile above. Now in the MACP block click the ellipse beside the CamProfile / StartProfile[0] entry to see the cam profile below.

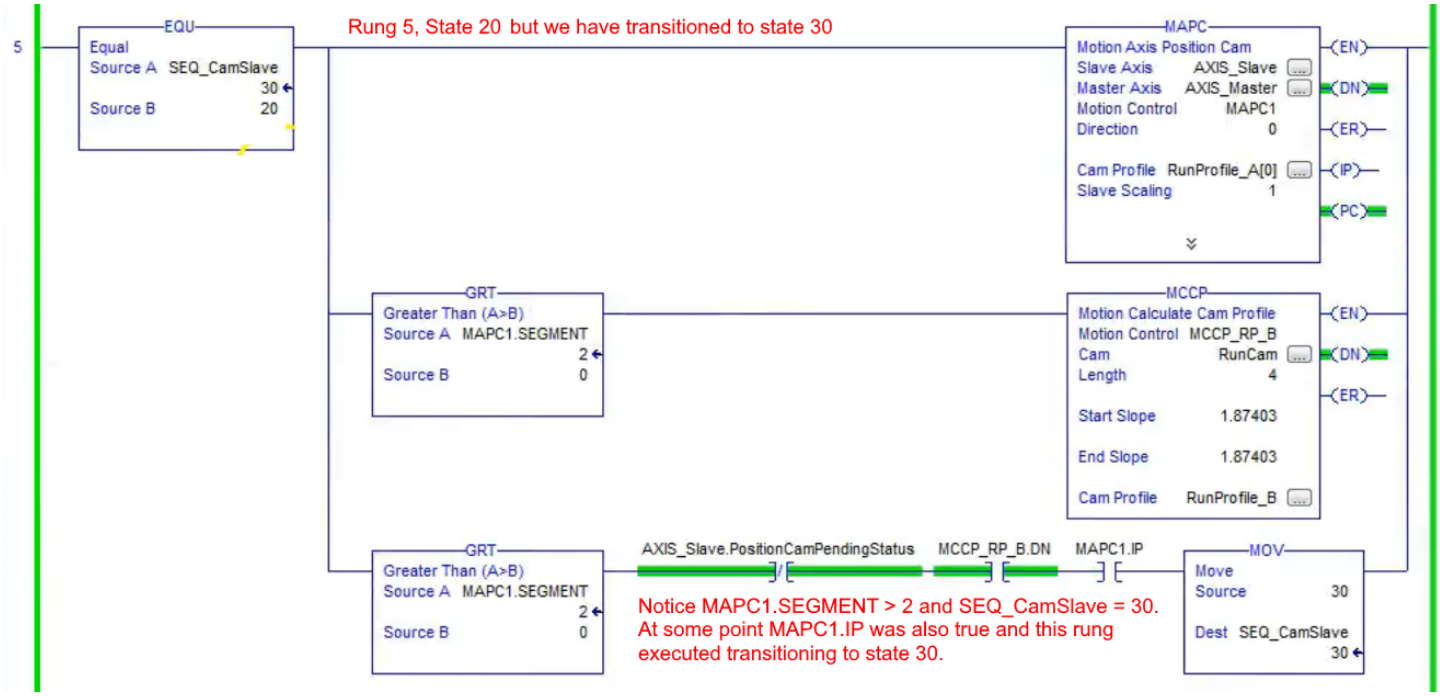


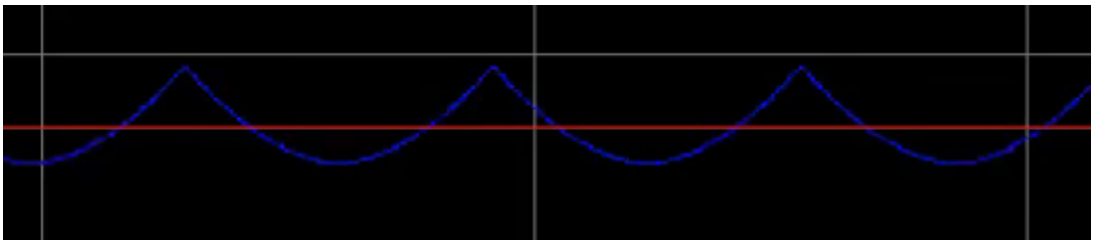

As soon as the position cam indexes and the IP bit goes true it will transition to the next state. The code is set up so no move may be in progress, CAM calc must be complete, and the move begins. Implies that MCCP fires off MAPC. Need to know the link between MCCP and MAPC, they do not seem to access the same "profile" arrays. Consider the code at states 20 and beyond below. The trend shows the result.



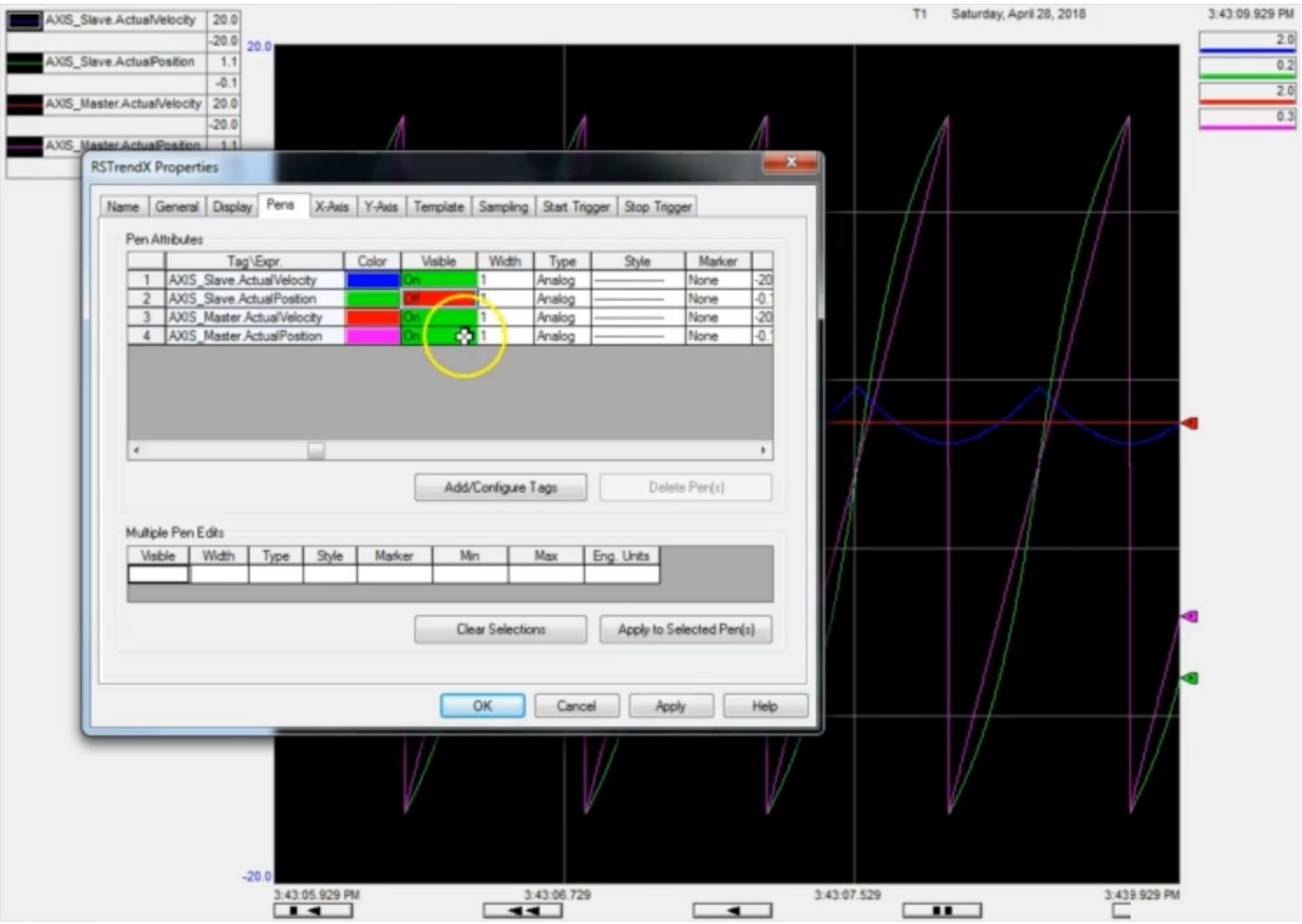



So the IP and SEGMENT terms are what is used to transition. This code oscillates between states 20 and 30. MPAC1.SEGMENT toggles as well. The trend plot on next page shows velocities and positions.

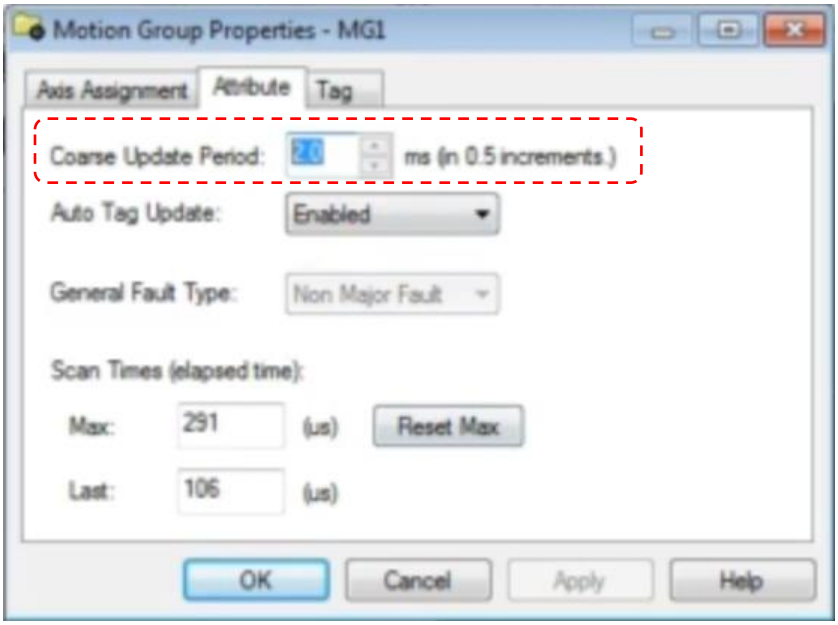


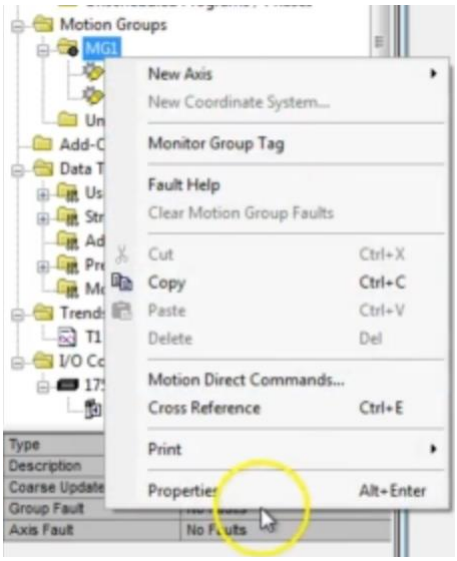



Here we are looking at velocity only, this can help us identify errors or areas in need of improvement.




The timing of the servos is really the key to the system. Also **consider the course rate update of the master (axis) group as it plays a role in the calculations**. Find this value in the properties of motion group MT1. In this case we are using 2 ms. (We are running virtual axis).



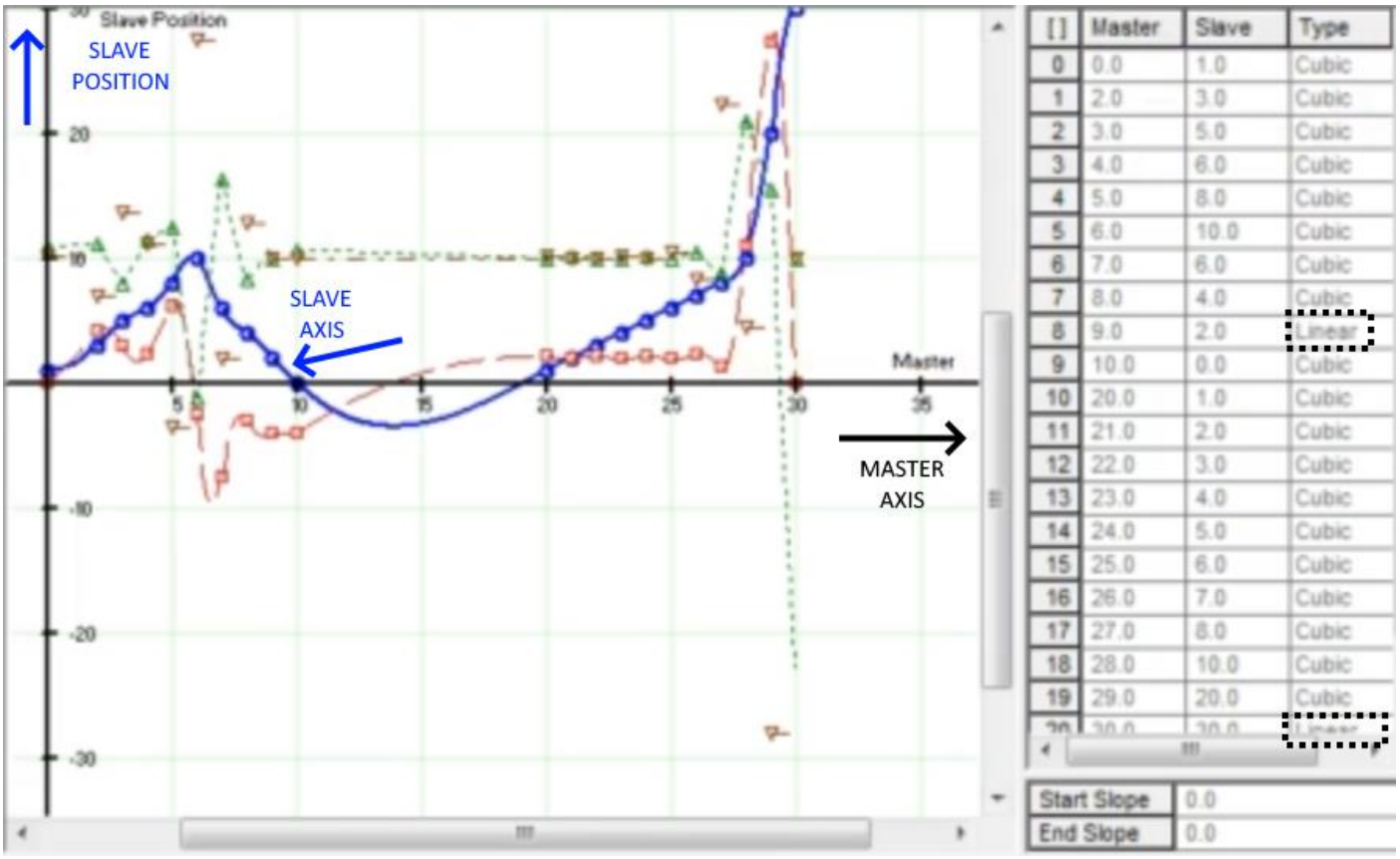




**Timing and Course Update Rate are the keys to servo motion systems, examine them carefully.**

Note that it is not necessary to use the MCCP block, you can run MAPC with a hard coded cam (see below). This method is used when you have to adjust CAMs on the fly. See lessons below for more on this topic.

## 30. Servo Mastery PCAM Explained In Detail

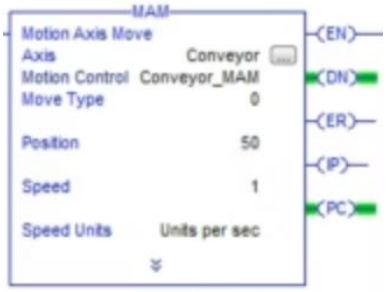
At this point we have moved the code above (ch 29) into a program without the phase structure.
Here we are still using a virtual axis running a motion axis position cam (MAPC). In this module we will examine what have done above at a slower pace. In lecture 31 below we will build a program from scratch, make a position cam, use the position cam (using real hardware, not emulator). This is a simplified system.
Now let's look at the cam used in MACP, ellipse button associated with cam profile.

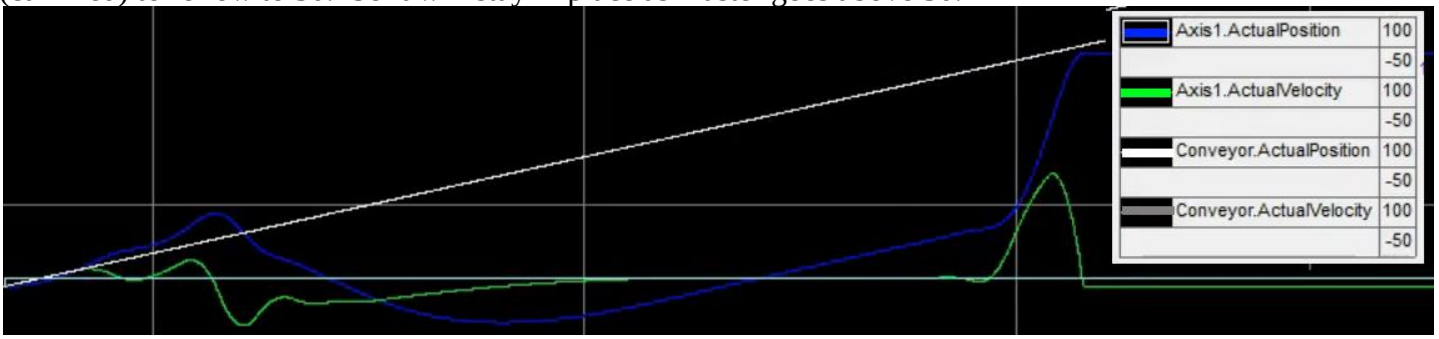


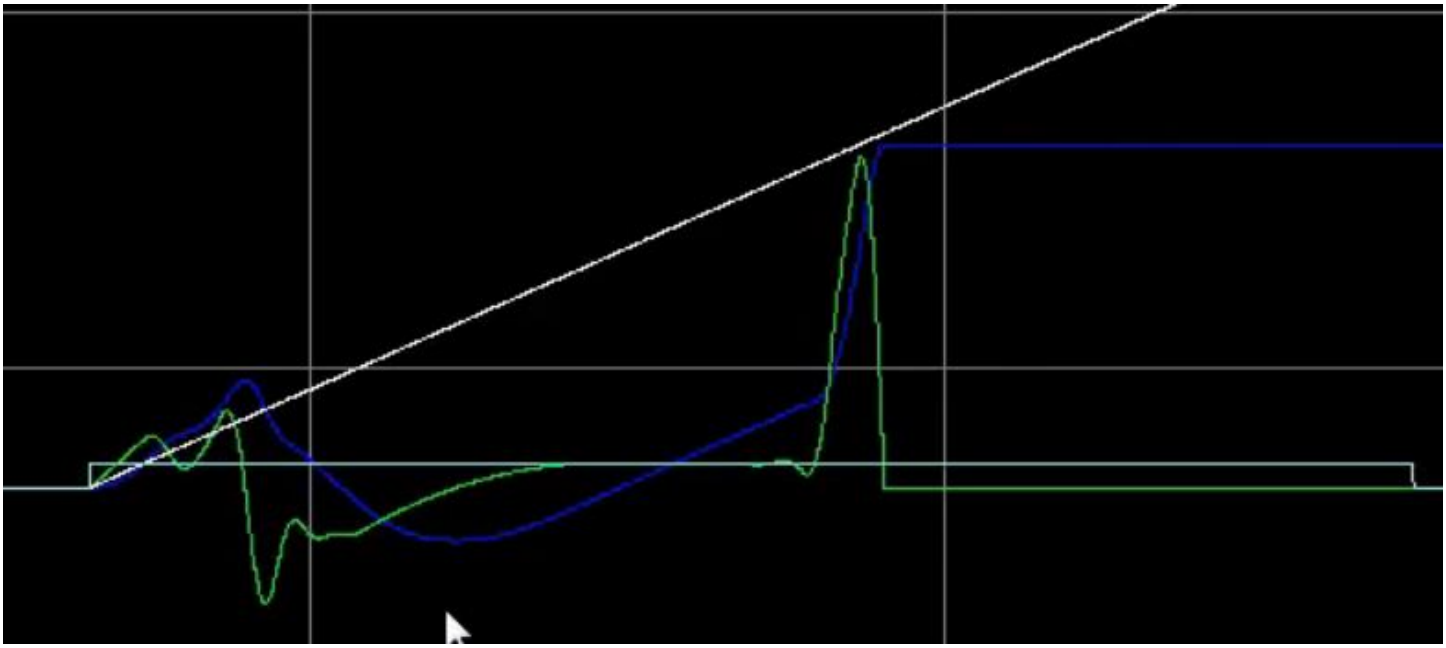

| [ ] | Master | Slave | Type |
|---|---|---|---|
| 0 | 0.0 | 1.0 | Cubic |
| 1 | 2.0 | 3.0 | Cubic |
| 2 | 3.0 | 5.0 | Cubic |
| 3 | 4.0 | 6.0 | Cubic |
| 4 | 5.0 | 8.0 | Cubic |
| 5 | 6.0 | 10.0 | Cubic |
| 6 | 7.0 | 6.0 | Cubic |
| 7 | 8.0 | 4.0 | Cubic |
| 8 | 9.0 | 2.0 | Linear |
| 9 | 10.0 | 0.0 | Cubic |
| 10 | 20.0 | 1.0 | Cubic |
| 11 | 21.0 | 2.0 | Cubic |
| 12 | 22.0 | 3.0 | Cubic |
| 13 | 23.0 | 4.0 | Cubic |
| 14 | 24.0 | 5.0 | Cubic |
| 15 | 25.0 | 6.0 | Cubic |
| 16 | 26.0 | 7.0 | Cubic |
| 17 | 27.0 | 8.0 | Cubic |
| 18 | 28.0 | 10.0 | Cubic |
| 19 | 29.0 | 20.0 | Cubic |
| 20 | 30.0 | 30.0 | Linear |

| | |
|---|---|
| Start Slope | 0.0 |
| End Slope | 0.0 |

In this case Axis1(the slave) is linear and the Conveyor is a rotary (cubic?). Master axis is moving from zero to thirty and the cam is controlling the slave axis to the blue line (slave is following the master). So we can see that the position of the master axis determines the position of the slave axis by means of the table. Now we are going to use the Motion Axis Move command to move to position 50 at a speed of 1 (slowly). We should see the shape of the cam represented on the trend. Light color is master (running), blue is the slave (following). Green is the velocity. So we see that the blue line in the trend resembles the cam profile. As the master goes from 0 to 30 (in this case) it will the slave axis (axis1) will follow according to the blue line in the cam profile. The position trend should look like the blue cam line. In this first example we tell the master (conveyor) to run to position 50, but the slave (axis1) is only programmed (cammed) to follow to 30. So it will stay in place as master goes above 30.



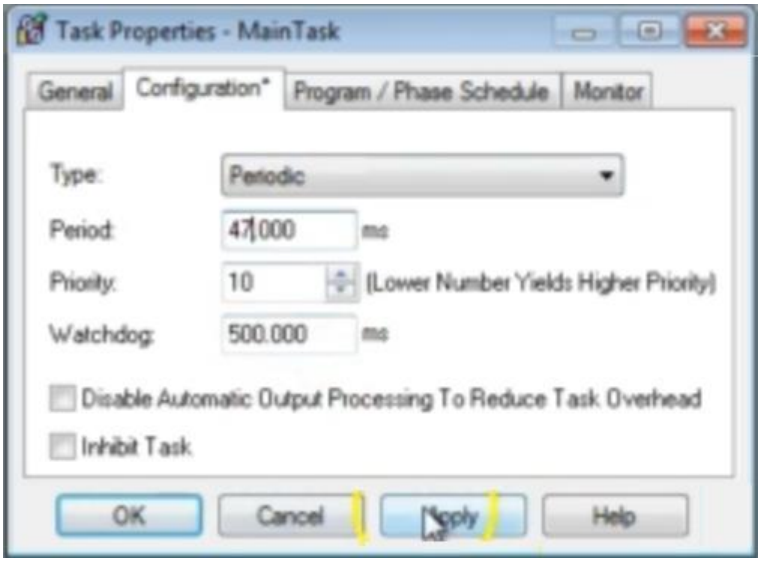





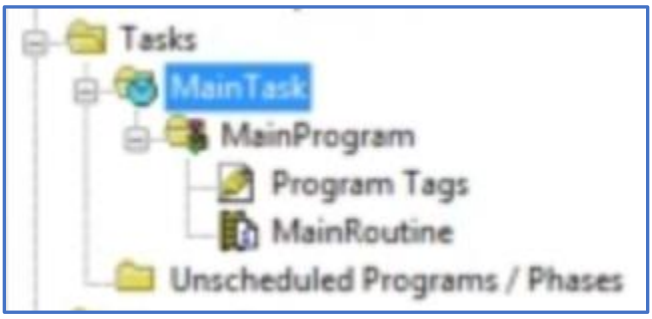
This is a second run where we increased the speed of the conveyor.  We can see that the cam caused the speed of the slave (axis1) to greatly increase so it can keep up.  The velocity lines are sharper.

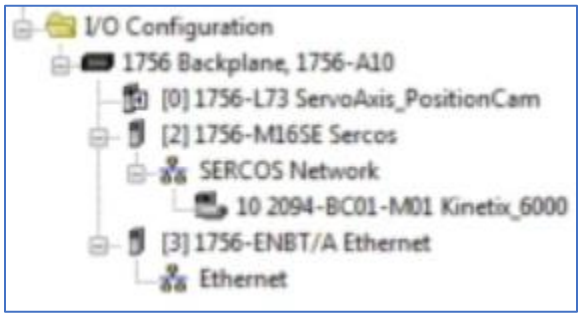
# 31. Building a PCAM From Scratch

In this lecture we will build an application with a cam from scratch. The name is **ServoAxis_PositionCam**. We will create a periodic task so that we can better time the system.



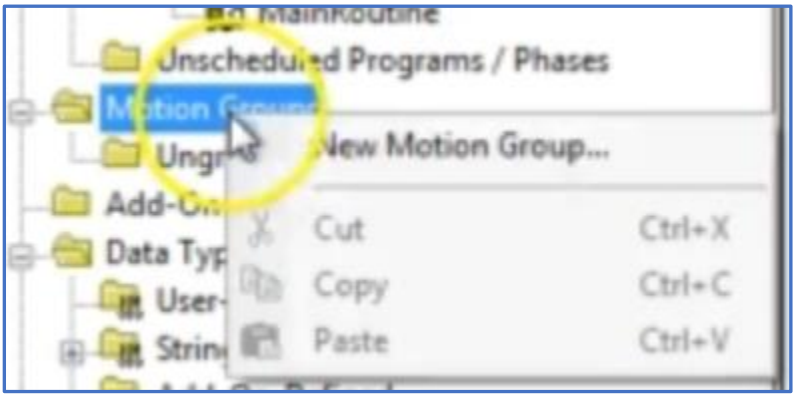

In this case we will create a periodic task, this allows us to better time the system, particularly in this example.



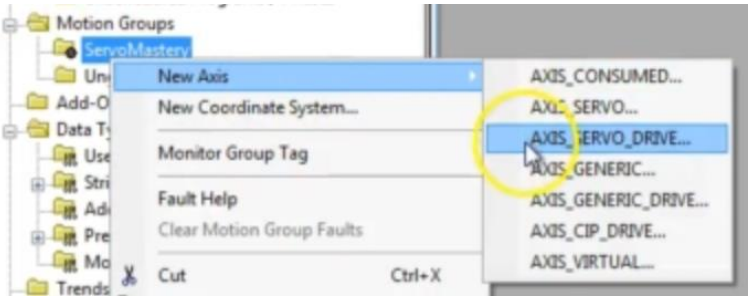

Next we add some IO as you see in this graphic.

Next create a motion group.

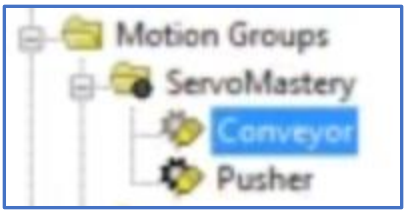




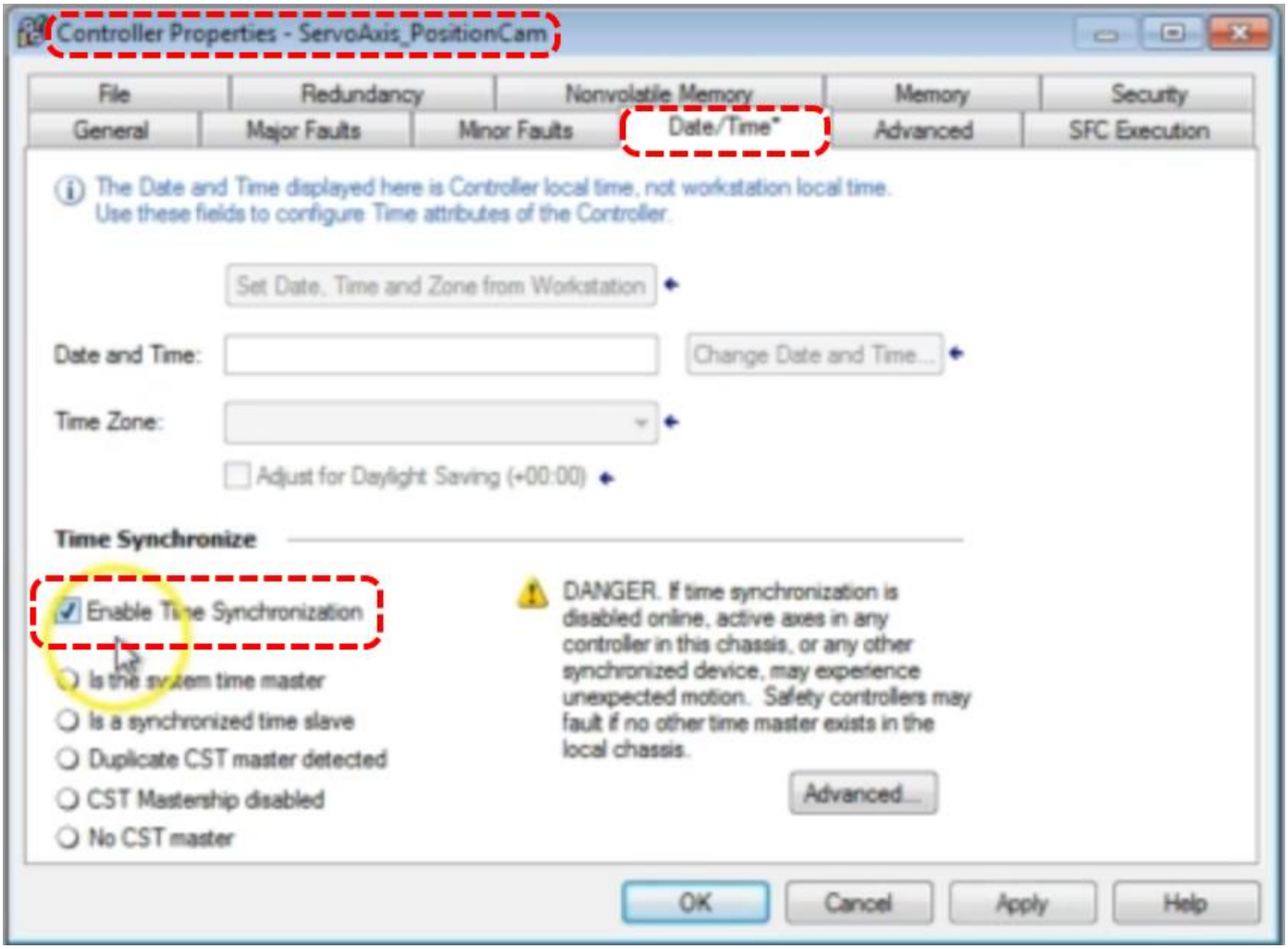
Now create the servo drive, call it Pusher. This is a physical drive [AXIS_SERVO_DRIVE].



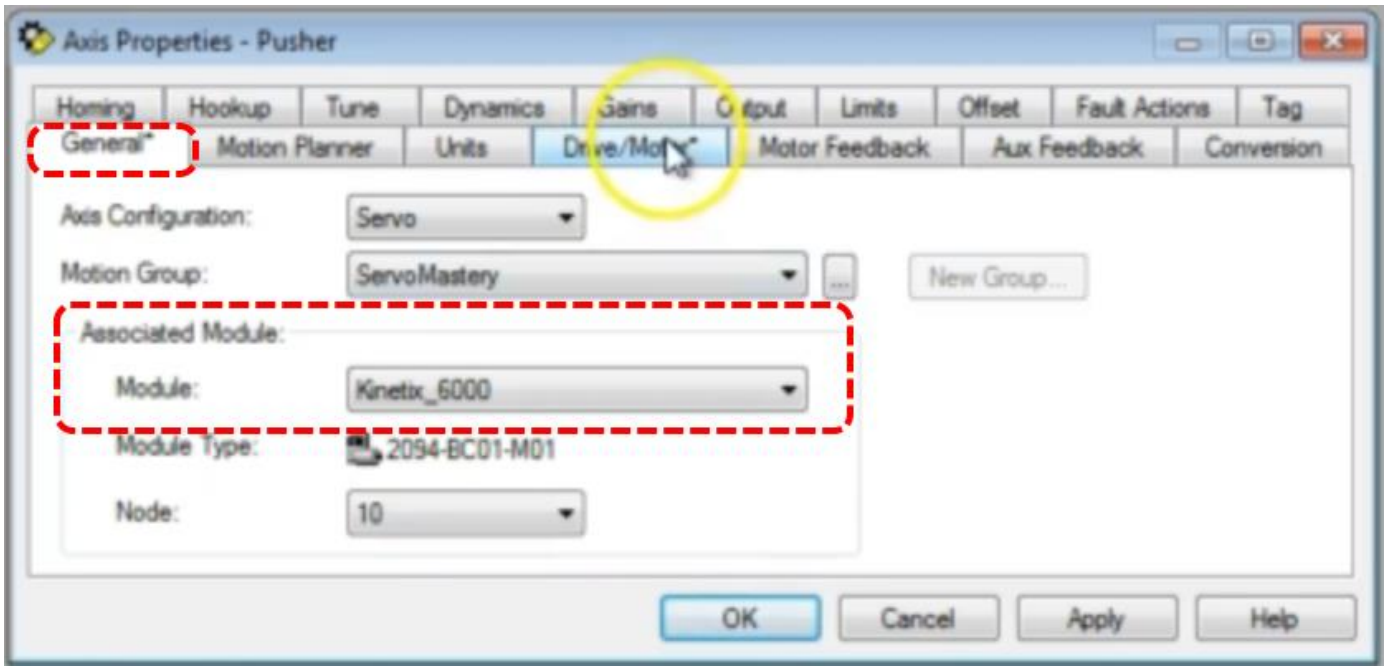

Now create a virtual drive called Conveyor.



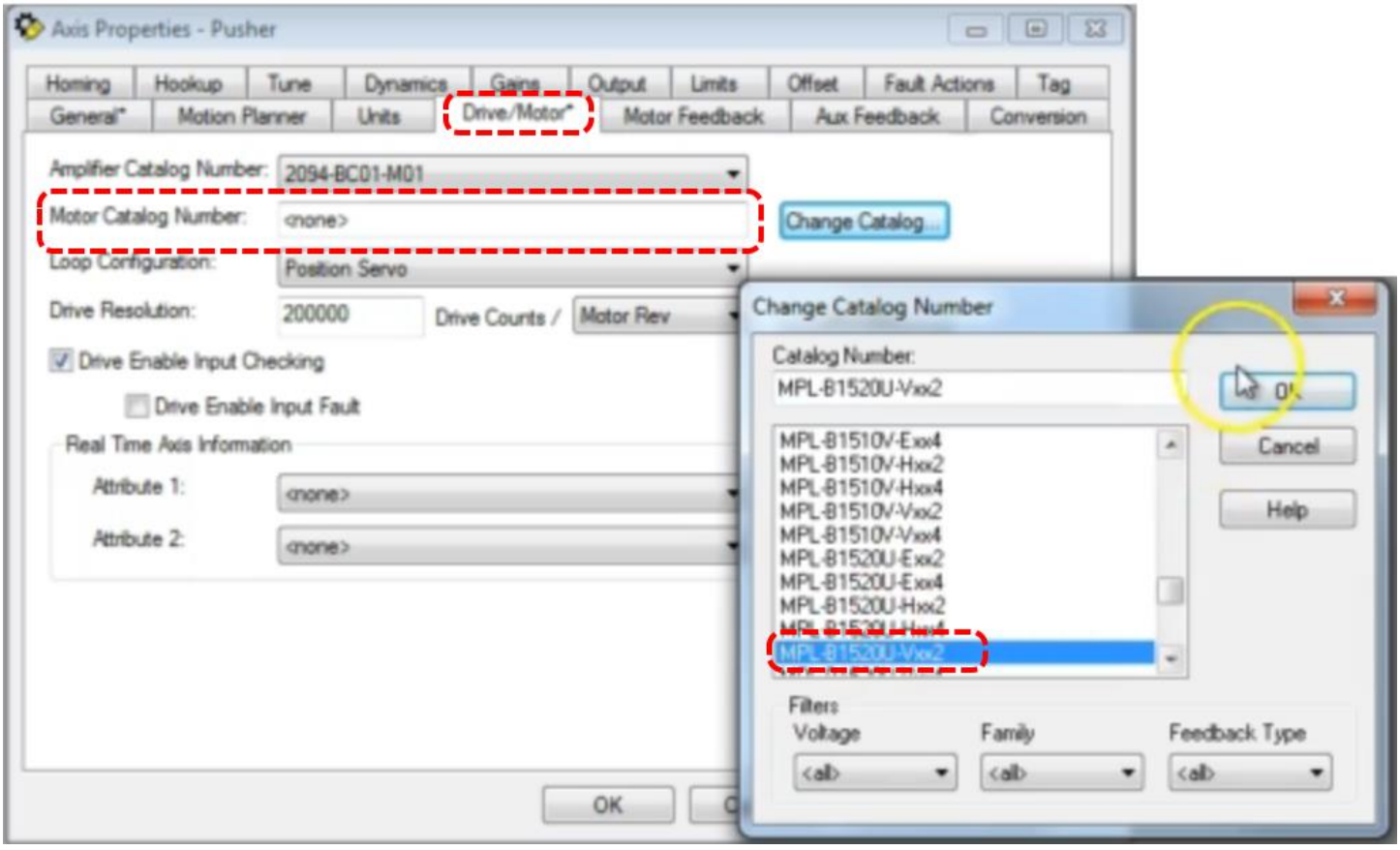

An important step, go to controller properties Date/Time tab and enable time synchronization as shown below.



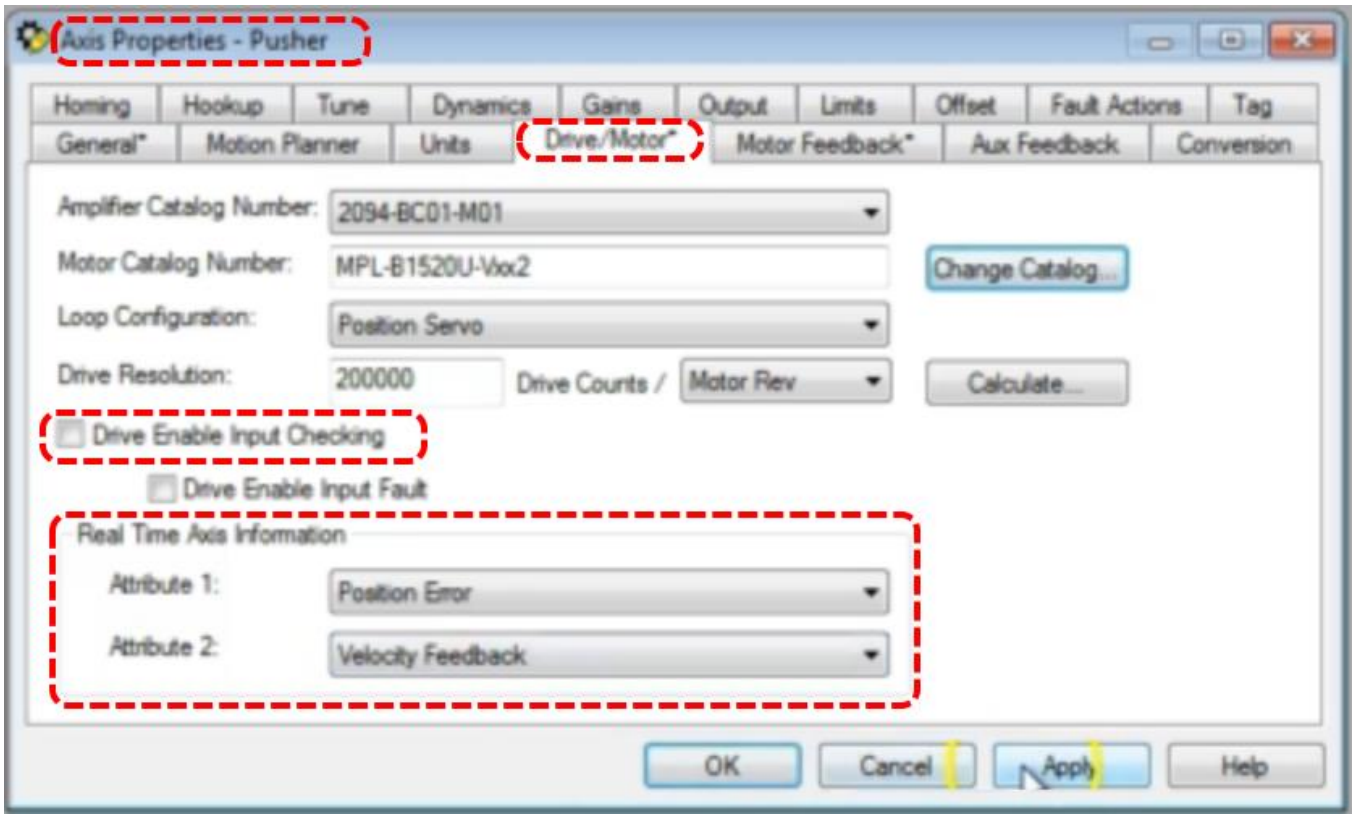

Now go to the Pusher axis properties and give it an associated module, the one down in the hardware IO list (Kinetic_6000).



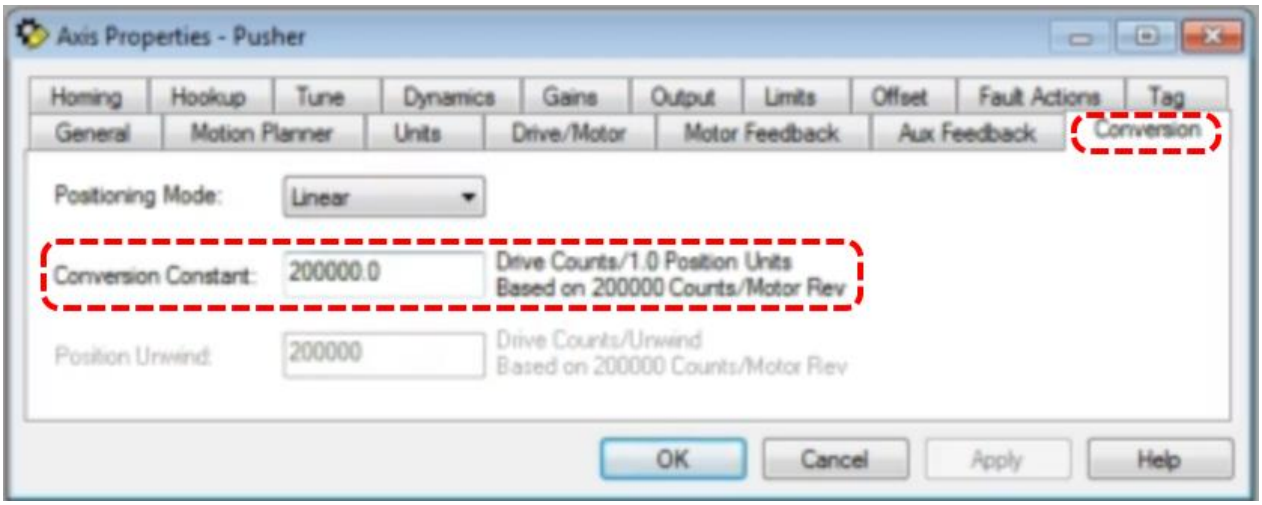

Now go to the Drive/Motor tab and select a motor.



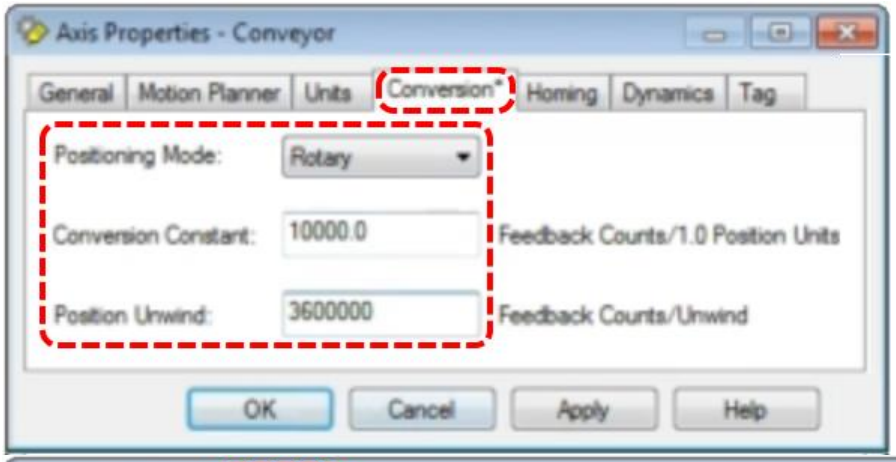

Also select the Real Time Axis Information attributes, in this case we will use Position Error and Velocity Feedback. In this case we do not want to use Drive Enable Input Checking so **uncheck** it.




Below is the value of the conversion constant we will use in the Pusher case. Recall that the value of the conversion factor is determined, to some extent, by the application.

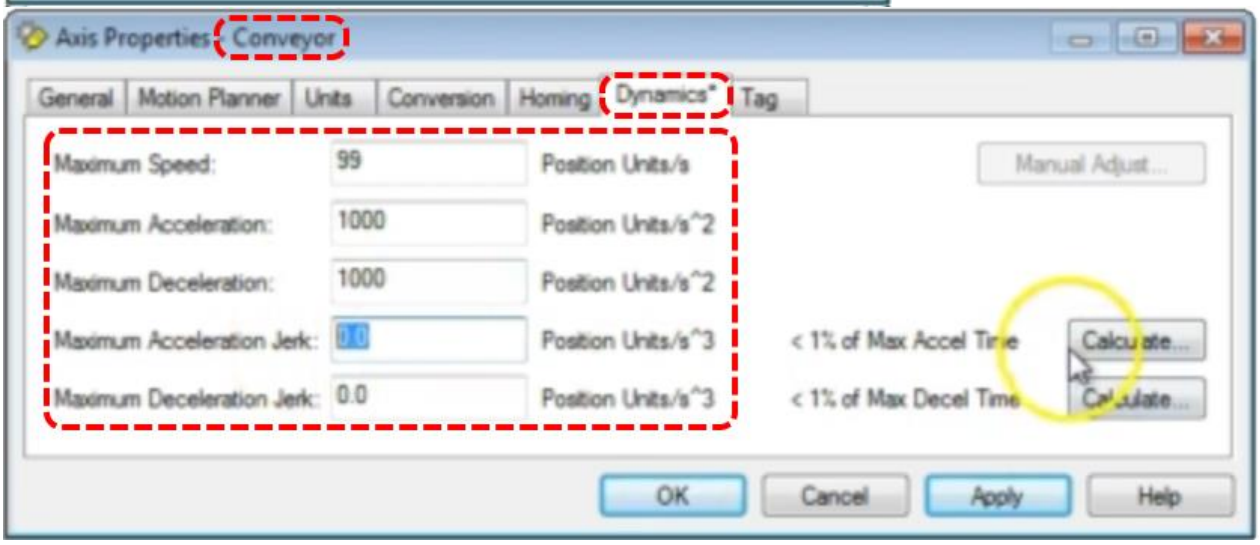



Now we do the same for the **Conveyor**. Select positioning mode Rotary and enter 36000 for Position Unwind. This means the axis values will reset after each 360 degree rotation. Use 10000 for conversion constant.

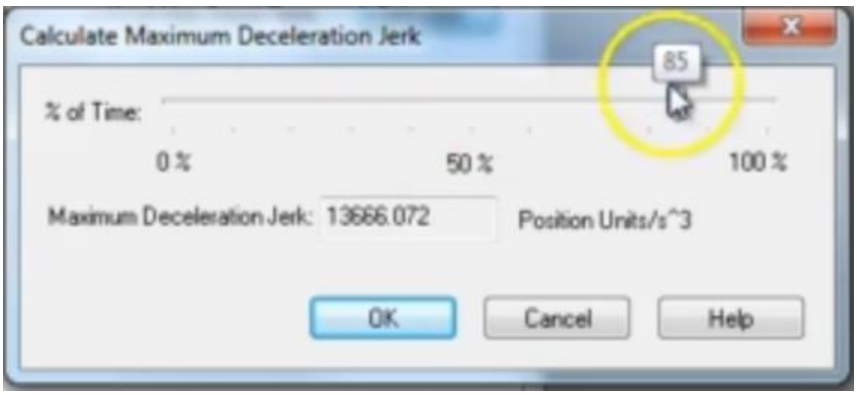



Position unwind is 10,000 feedback counts/1 position units times 360 degrees for the axis to return to the "home" position. This equals 3.6M.

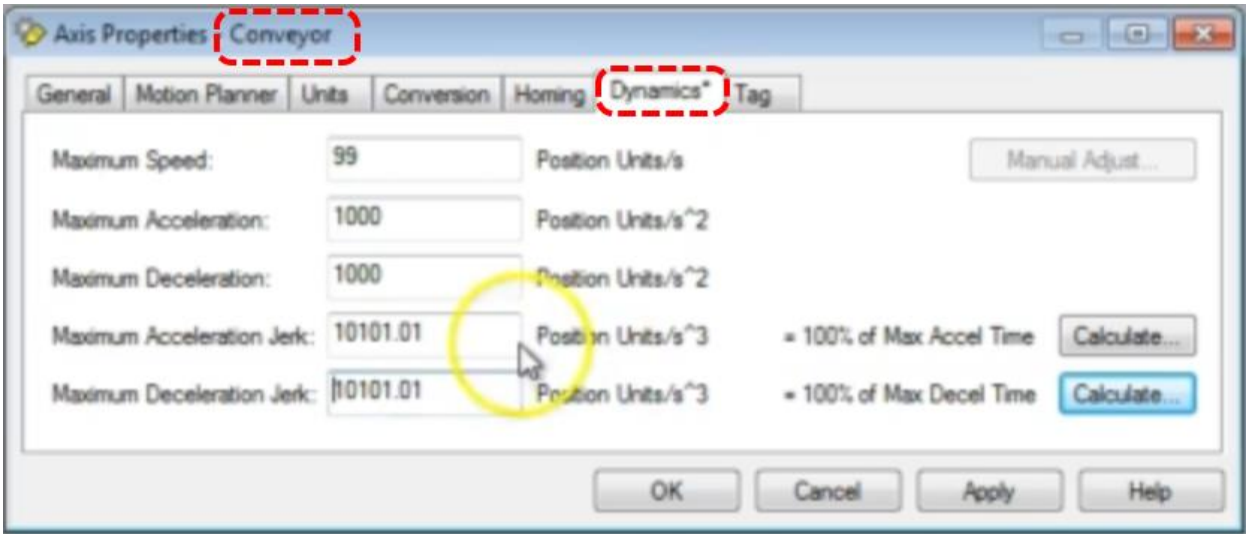
Now go to the dynamics tab of the Conveyor properties dialog. We have to enter values for the various max and min setting. Since this is a virtual drive the values entered do not have to be realistic (like those shown below).



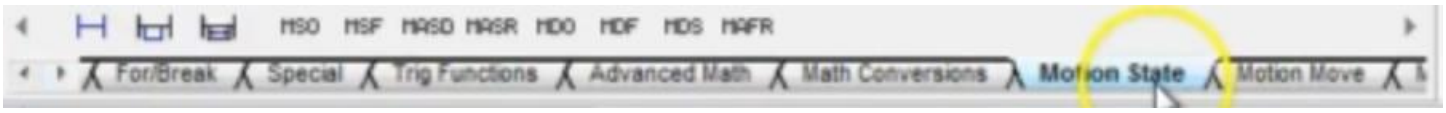

Now click both Calculate buttons and make an entry on the slider and then click OK.

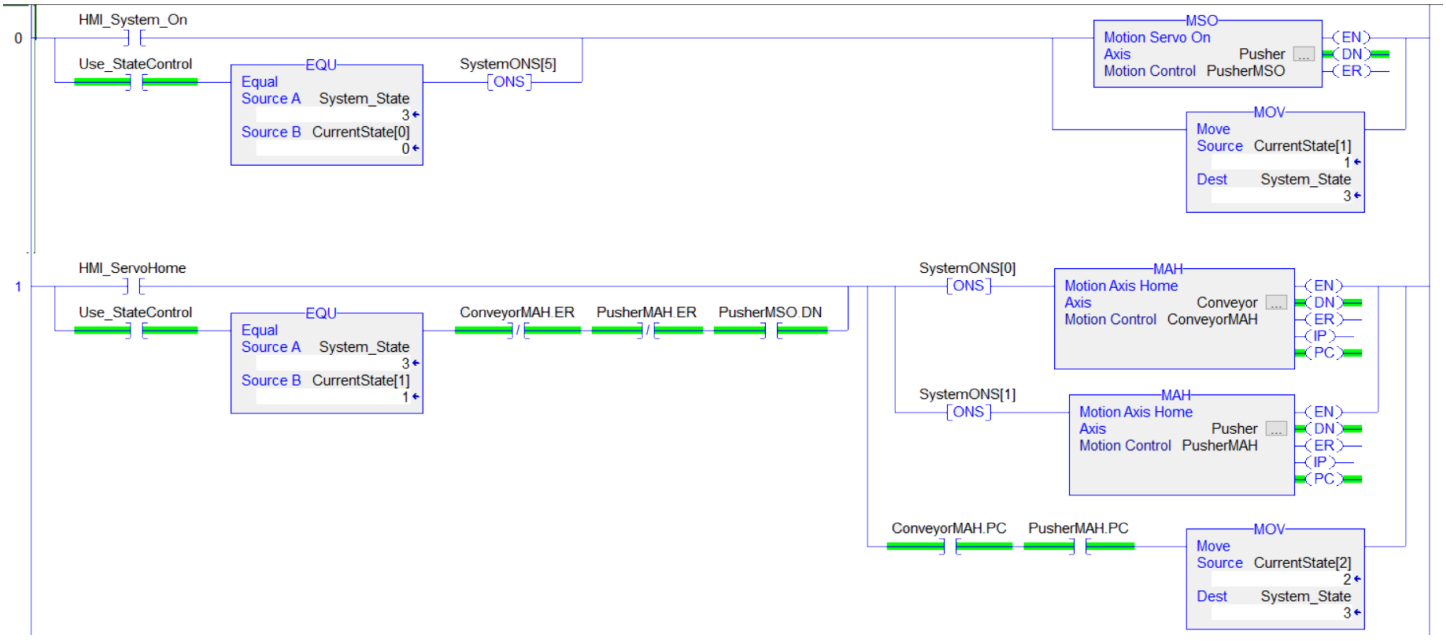
These calculations are based on the max and min values we entered.  This will modify the values we had entered.



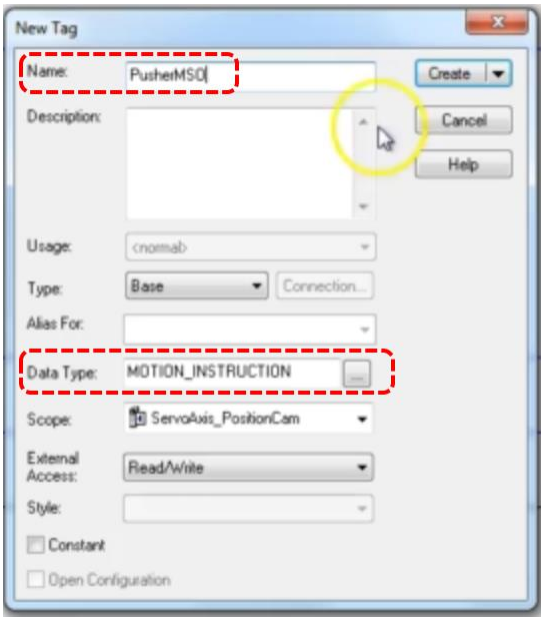

Now open the Main Routine so we can create some ladder logic.  Go to the Motion State tab to select the motion function blocks.

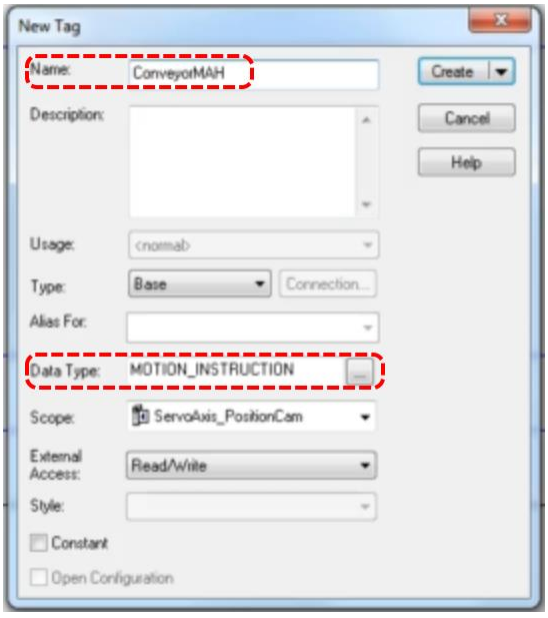


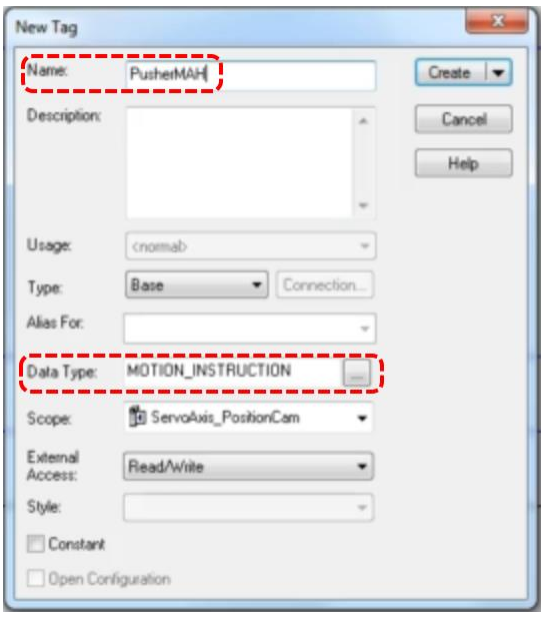

We will want to create Home and an On rungs.



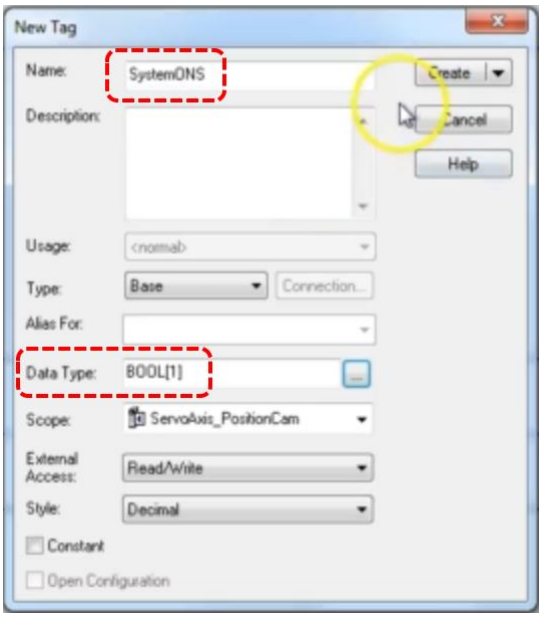

Where …

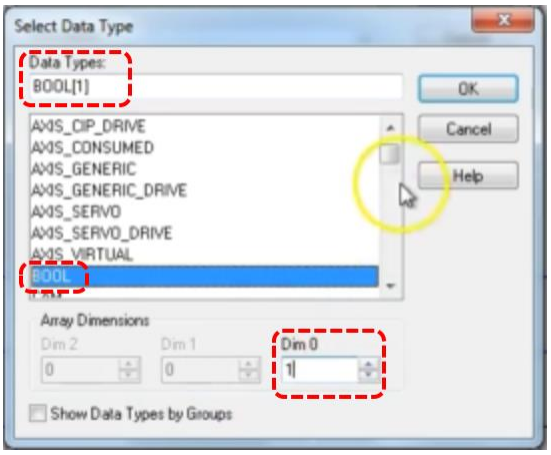


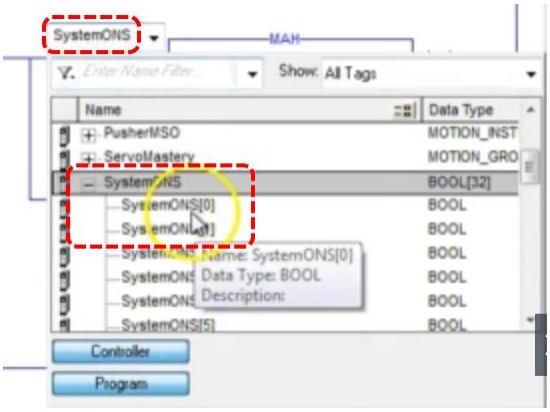



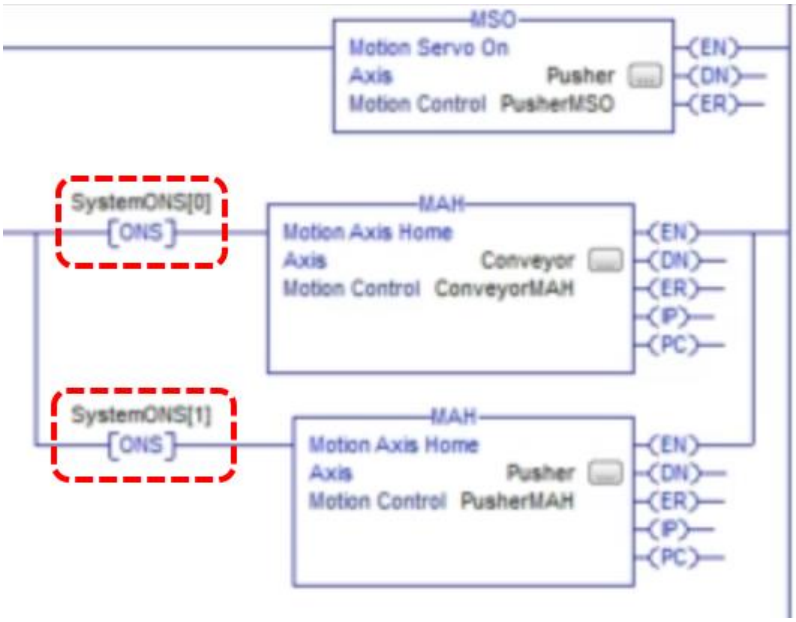




We want to add some NC rungs in front of our homing function blocks (MAH) so we do not re-home an already homed axis. There are several tags that can do this, a good one might be Conveyor.HomedStatus (same for pusher). But the simplest way is to use a one shot. Note that our one shot is declared as SystemONS with data type BOOL and is a control array of 1 dimension (next page).



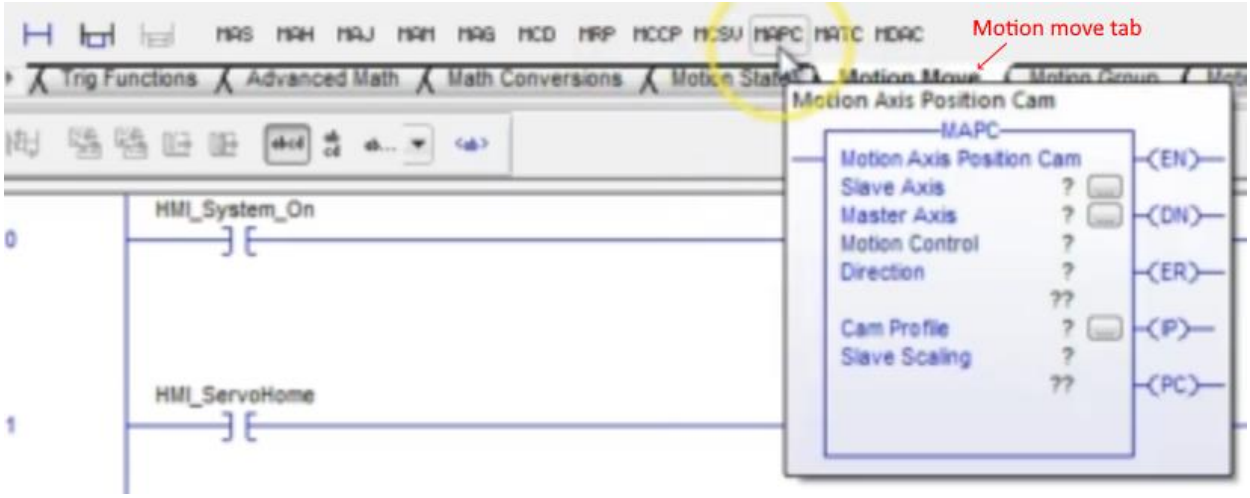



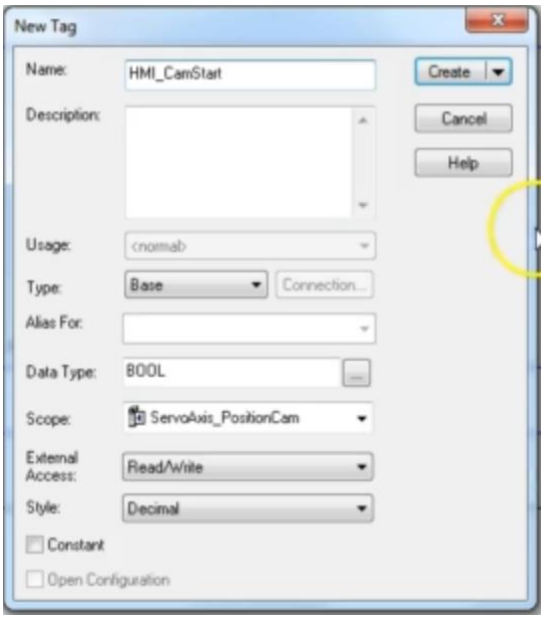



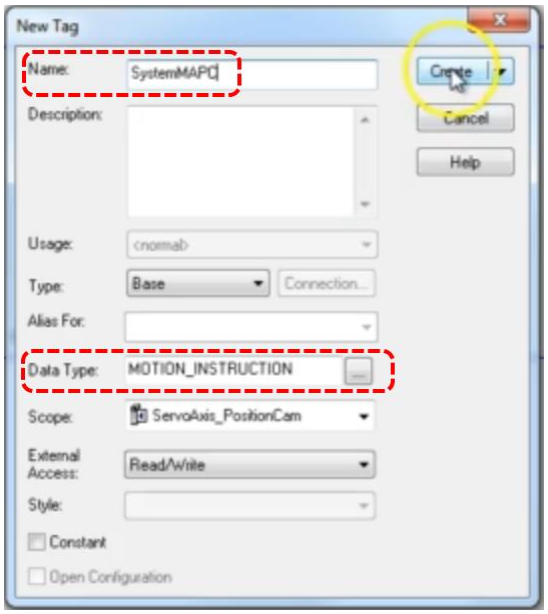

Now we write a rung to test the .PC bit (.ProcessComplete) to make sure both have come on and have completed homing. Then we want to check the .ServoActionStatus bit to be sure they report active. If all are true we can start the cam.


Where HMI_CamStart is a BOOL. Now add a one shot and a position cam function block, MAPC.

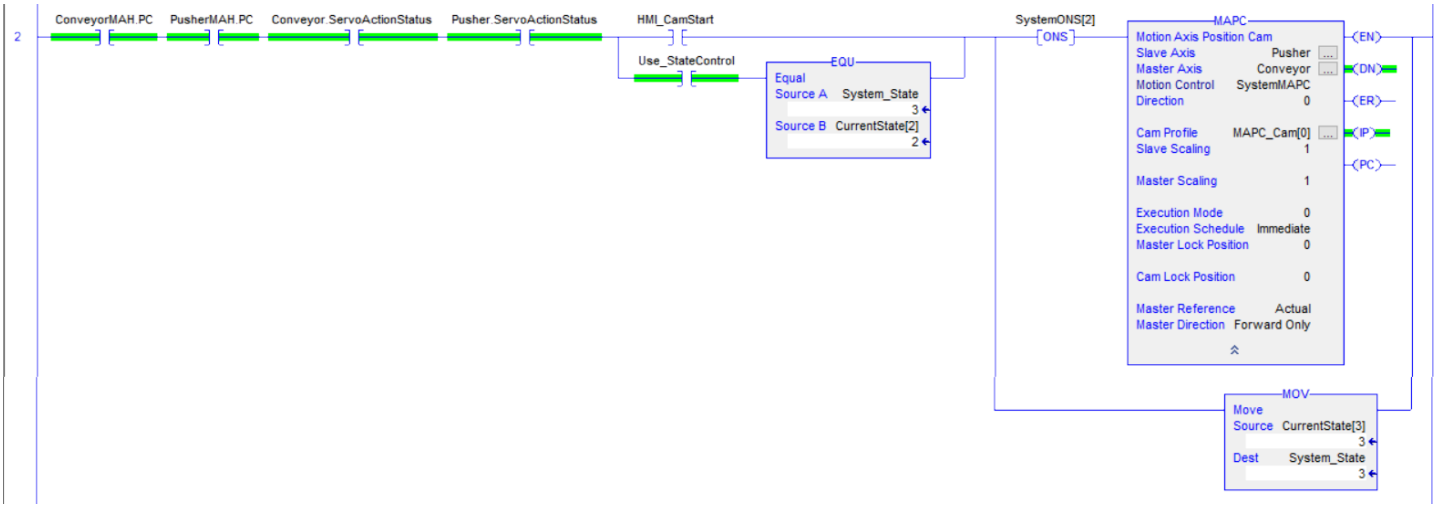


Create these other parameters…




Now we have …




At this point we can review the help file for some details…
The function block is filled out below.

## Motion Axis Position Cam (MAPC)

The Motion Axis Position Cam (MAPC) instruction provides electronic camming between any two axes according to the specified Cam Profile.

When executed, the specified Slave Axis is synchronized to the designated Master Axis using a position Cam Profile established by the RSLogix 5000 Cam Profile Editor, or by a previously executed Motion Calculate Cam Profile (MCCP) instruction. The direction of Slave Axis motion relative to the Master Axis is defined by a flexible Direction input parameter. The camming Direction, as applied to the slave, may be explicitly set as the Same or Opposite or set relative to the current camming direction as Reverse or Unchanged.

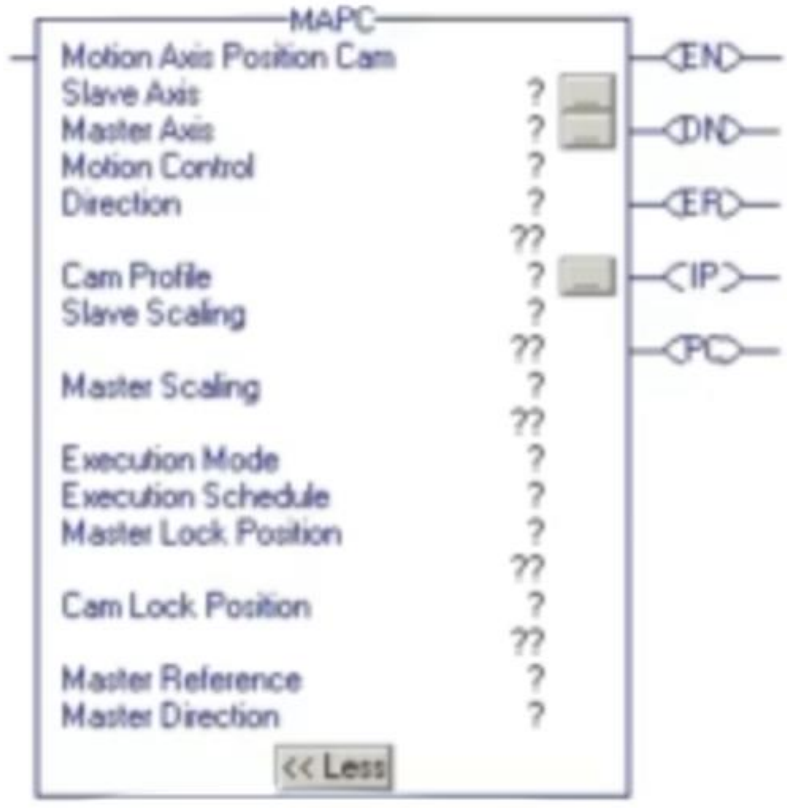
To accurately synchronize the slave axis position to master axis position, an Execution Schedule setting and an associated Master Lock Position can be specified for the master axis. When the master axis travels past the Master Lock Position in the direction specified by the Execution Schedule parameter, the slave axis is locked to the master axis position according to the specified Cam Profile beginning at the Cam Lock Position.

The cam profile can also be configured via the Execution Schedule parameter to execute Immediately or Pending completion of a currently executing position cam profile. The cam profile can also be executed Once or Continuously by specifying the desired Execution Mode. The Master Reference selection allows camming input from the master to be derived from either the Actual or Command position of the Master Axis.

To support applications which require unidirectional motion, a "slip clutch" feature is available, which prevents the slave from "backing-up" when the master axis reverses direction. This feature is controlled by the Master Direction parameter. Master and Slave Scaling functionality can be used to scale slave motion based on a standard cam profile without having to create a new cam table and calculate a new cam profile.

### Operands

### Ladder Diagram



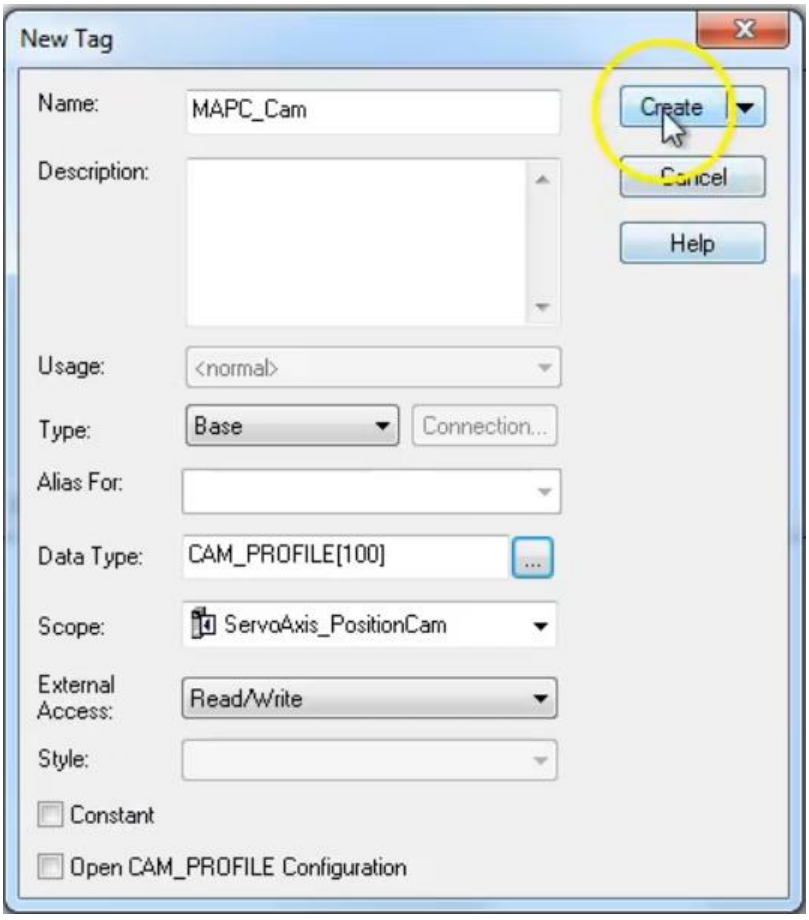
| Operand | Type | Format | Description |
|---|---|---|---|
| Slave Axis | AXIS_CIP_DRIVE<br>AXIS_VIRTUAL<br>AXIS_GENERIC<br>AXIS_SERVO<br>AXIS_SERVO_DRIVE | Tag | The name of the axis that the cam profile is applied to. Ellipsis launches Axis Properties dialog. |
| Master Axis | AXIS_CIP_DRIVE<br>AXIS_FEEDBACK<br>AXIS_CONSUMED<br>AXIS_VIRTUAL<br>AXIS_GENERIC<br>AXIS_SERVO<br>AXIS_SERVO_DRIVE | Tag | The axis that the slave axis follows according to the cam profile. Ellipsis launches Axis Properties dialog. If Pending is selected as the Execution Schedule, then Master Axis is ignored. |
| Motion Control | MOTION_INSTRUCTION | Tag | Structure used to access block status parameters. |
| Direction | UINT32 | Immediate or Tag | Relative direction of the slave axis to the master axis:<br>• Same – the slave axis position values are in the same sense as the master's.<br>• Opposite – the slave axis position values are in the opposite sense of the master's.<br>Or relative to the current or previous camming direction:<br>• Reverse – the current or previous direction of the position cam is reversed on execution. When executed for the first time with Reverse selected, the control defaults the direction to Opposite.<br>• Unchanged – this allows other cam parameters to be changed without altering the current or previous camming direction. When executed for the first time with Unchanged selected, the control defaults the direction to Same. |
| Cam Profile | CAM_PROFILE | Array | Tag name of the calculated cam profile array used to establish the master/slave position relationship. Only the zero array element ([0]) is allowed for the Cam Profile array. Ellipsis launches Cam Profile Editor. |
| Slave Scaling | REAL | Immediate or Tag | Scales the total distance covered by the slave axis through the cam profile. |
| Master Scaling | REAL | Immediate or Tag | Scales the total distance covered by the master axis through the cam profile. |
| Execution Mode | UINT | Immediate | Determines if the cam profile is executed only one time or repeatedly:<br>0 = Once – cam motion of slave axis starts only when the master axis moves into the range defined by the start and end points of the cam profile. When the master axis moves beyond the defined range cam motion on the slave axis stops and the Process Complete bit is set. Slave motion does not resume if the master axis moves back into the cam profile range.<br>1 = Continuous – Once started the cam profile is executed indefinitely. This |

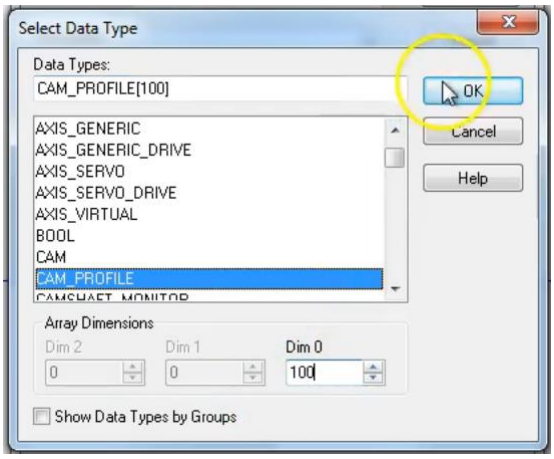
| | | | |
|---|---|---|---|
| | | | feature is useful in rotary applications where it is necessary that the cam position run continuously in a rotary or reciprocating fashion.<br>2 = Persistent - When the Master Axis moves beyond the defined range, cam motion on the Slave Axis stops and the PositionCamLockStatus bit is cleared. Slave motion resumes in the opposite direction when the Master Axis reverses and moves back into the cam profile range, at which time the PositionCamLockStatus bit is set. |
| Execution Schedule | UINT32 | Immediate | Selects the method used to execute the cam profile. Options are:<br>0 = Immediate – The slave axis is immediately locked to the master axis and the position camming process begins. |
| Execution Schedule | UINT32 | Immediate | Selects the method used to execute the cam profile. Options are:<br>0 = Immediate – The slave axis is immediately locked to the master axis and the position camming process begins.<br>1 = Pending – lets you blend a new position cam execution after an in process position cam is finished. When Pending is selected the following parameters are ignored: Master Axis, Master Lock Position, and Master Reference.<br>2 = Forward only – the cam profile starts when the master position crosses the Master Lock Position in the forward direction.<br>3 = Reverse only – the cam profile starts when the master position crosses the Master Lock Position in the reverse direction.<br>4 = Bi-directional – the cam profile starts when the master position crosses the Master Lock Position in either direction. |
| Master Lock Position | REAL | Immediate or Tag | When the Master Offset = 0.0, the Master Lock Position is the Master axis absolute position where the slave axis locks to the master axis. If the Master Offset is X, then the Slave axis will lock to the Master axis at the absolute master position value of Master Lock Position –X.<br>For example: Assume a Master Lock Position = 50 and a Master Offset Move = 10. Also assume that the Master axis move (MAM) and Master offset move (MOM) start at the same time. Then the Slave will lock to the Master at an absolute Master axis position of 40. This in effect shifts the Cam profile 10 units to the left.<br>If Pending is selected as the Execution Schedule value, then Master Lock Position is ignored. |
| Cam Lock Position | REAL | Immediate or Tag | This determines the starting location in the cam profile. |
| Master Reference | UINT32 | Immediate | Sets the master position reference to either Command position or Actual position. If Pending is selected for the Execution Schedule value, then Master Reference is ignored.<br>0 = Actual – slave axis motion is generated from the current position of the master axis as measured by its encoder or other feedback device.<br>1 = Command – slave axis motion is generated from the desired or commanded position of the master axis. |
| Master Direction | UINT32 | Immediate | This determines the direction of the master axis that generates slave motion according to the cam profile.<br>Options are:<br>0 = Bi-directional – slave axis can track the master axis in either direction. |

So now we need to put in a cam profile. The cam profile needs to have depth, we have to make space for the number of instances we plan to use. In this case we'll use 100.

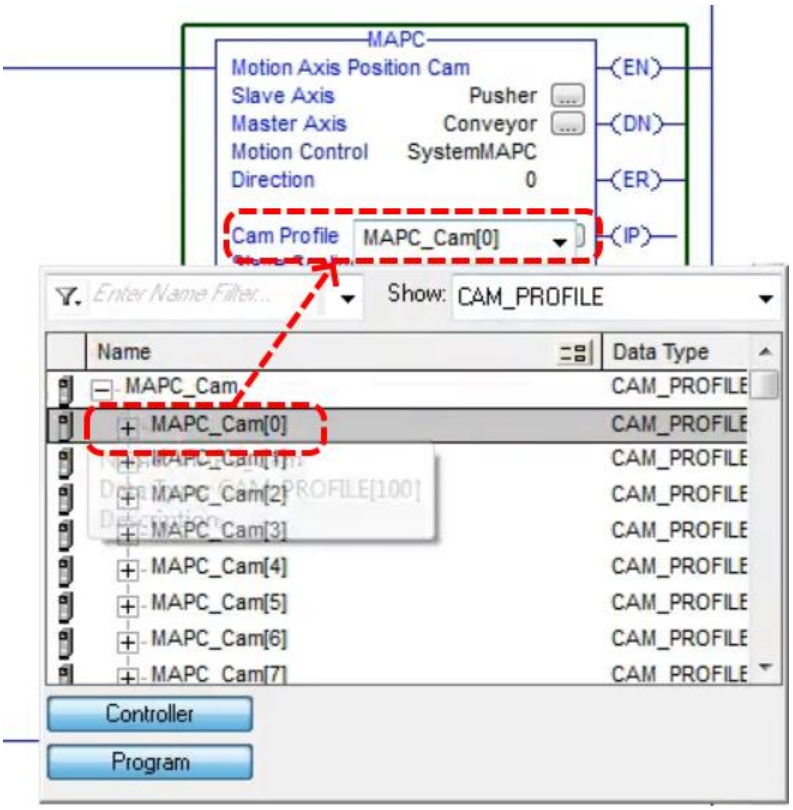
Data Types:

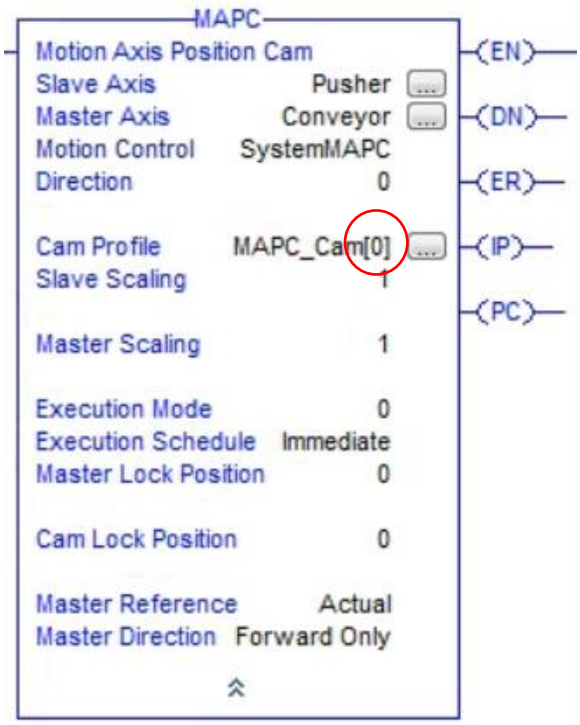
CAM_PROFILE[100]




Select the initial element in the MAPC_Cam Profile entry.

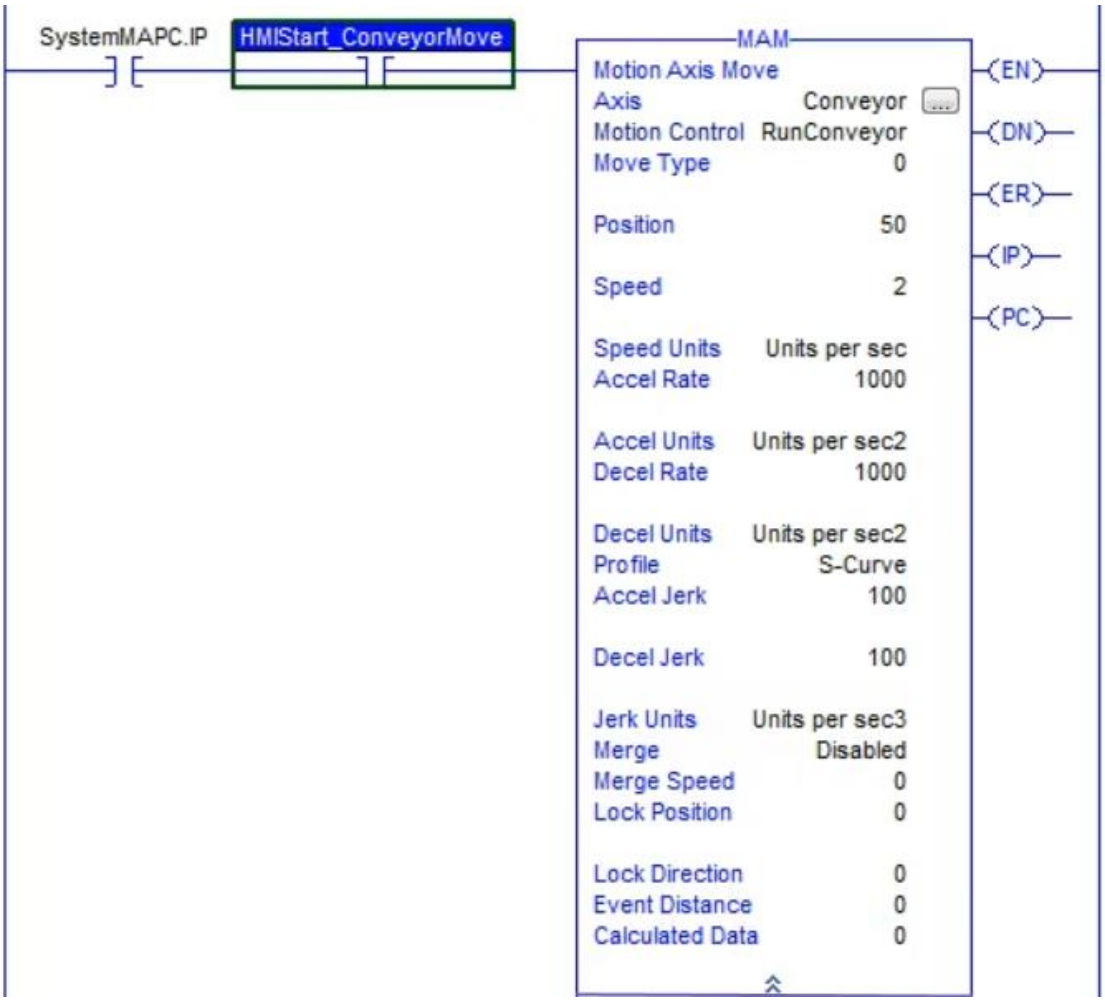


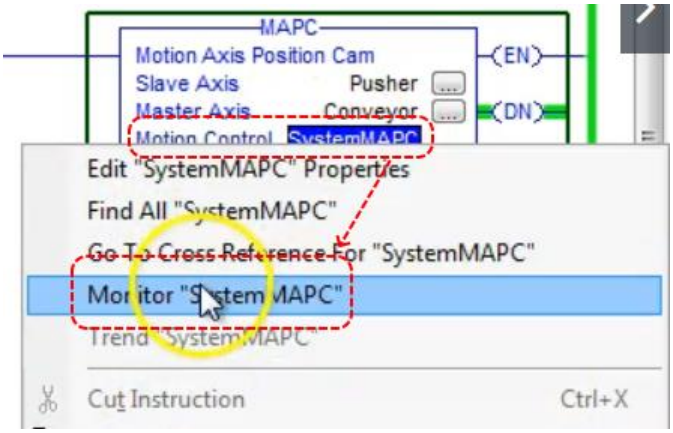



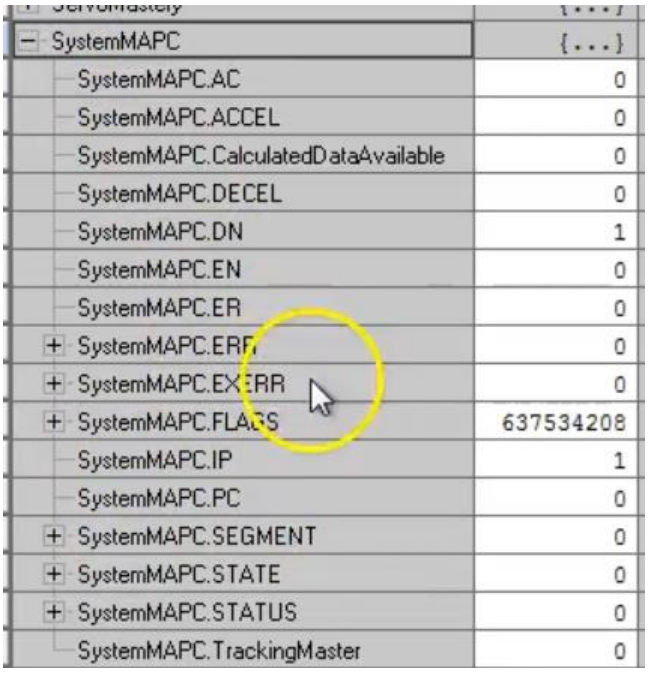

We could open the cam editor and type the points in manually, but it can also be done through Excel using the cell drag tool. You can paste into a blank cam table. Start by using the cubic cam. Next create a rung to run the cam. In this case we will use Motion Axis Move (MAM). We are going to run the master cam which is the conveyor. Notice MAPC Cam Profile, the array has 100 members but you always reference the first position (zero) in this position (naming field).

| To | Use This Move Type | And Enter |
|---|---|---|
| Move an axis to an absolute position | Absolute | 0 |
| Move an axis a specified distance from where it is now | Incremental | 1 |
| Move a Rotary axis to an absolute position in the **shortest** direction regardless of its current position | Rotary Shortest Path | 2 |
| Move a Rotary axis to an absolute position in the **positive** direction regardless of its current position | Rotary Positive | 3 |
| Move a Rotary axis to an absolute position in the **negative** direction regardless of its current position | Rotary Negative | 4 |
| Offset the master value of a position cam to an absolute position | Absolute Master Offset | 5 |
| Offset the master value of a position cam by an incremental distance | Incremental Master Offset | 6 |
| See Choose a Move Type for a Rotary Axis below for more information about rotary moves. | | |

The move type help file can be useful.

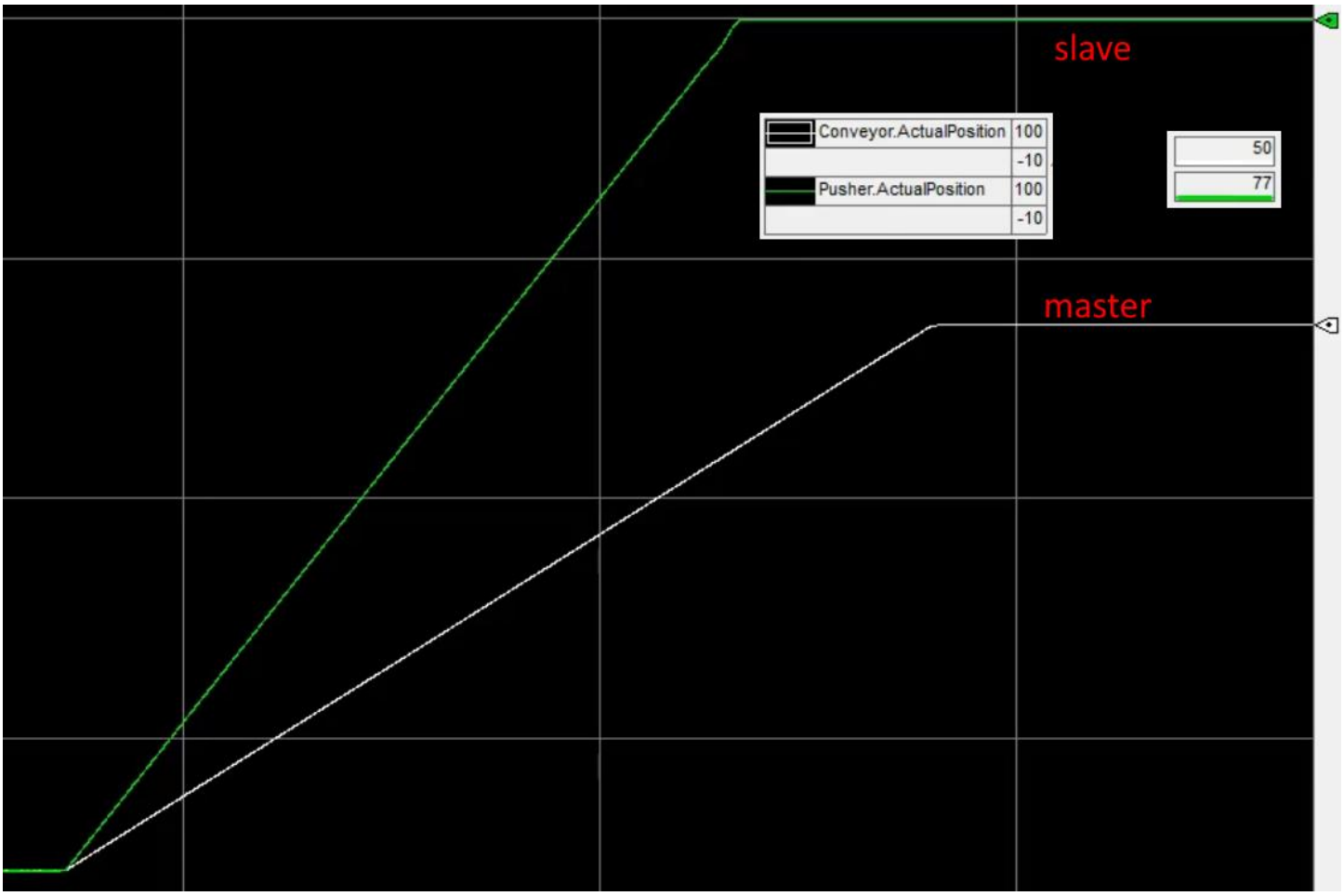


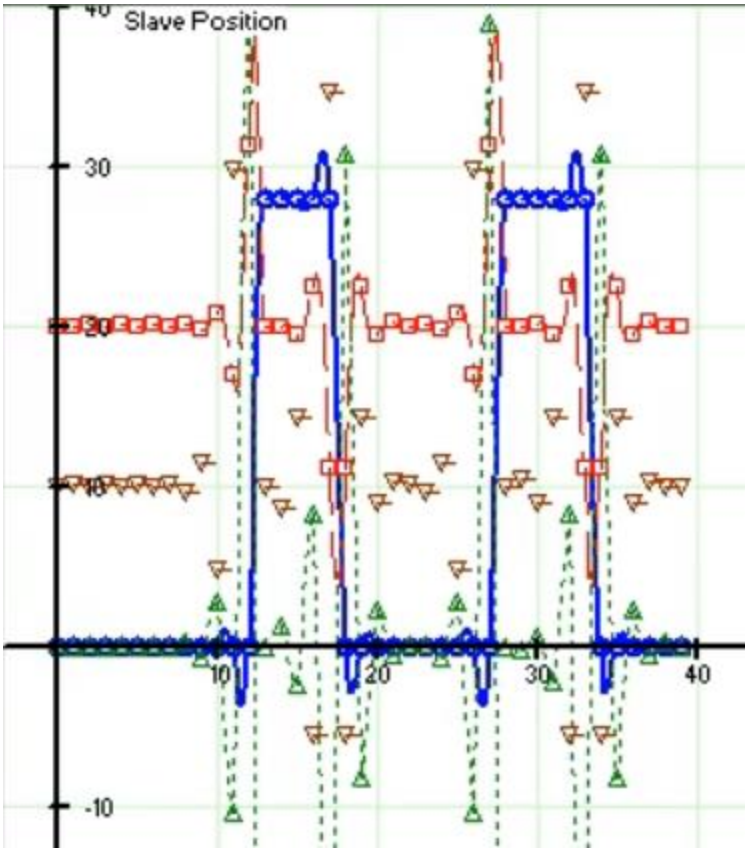


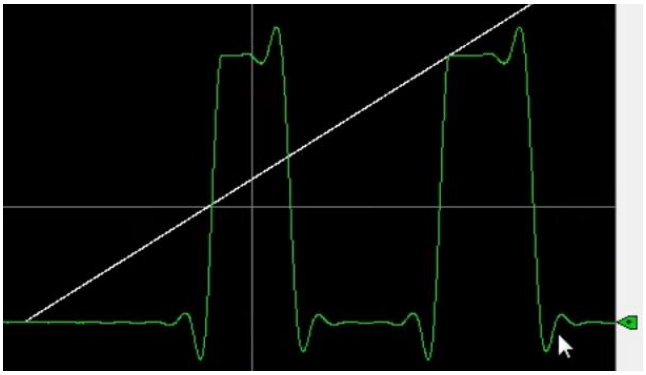

What to do if you have an error or fault? Right click on the Motion Control Entry in MAPC and click on Monitor "SystemMAPC" which will take you to the tag values. Look up your error code in the help file.

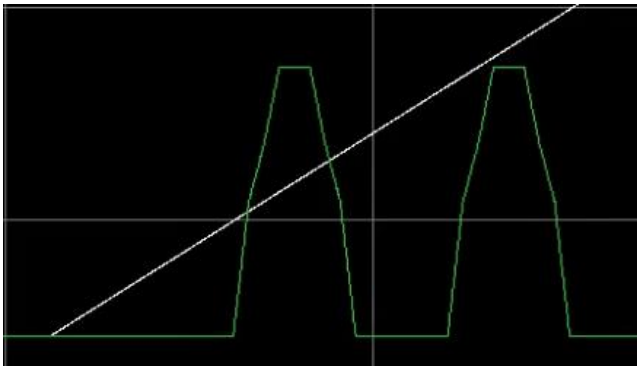
This is a test run of our first cam.



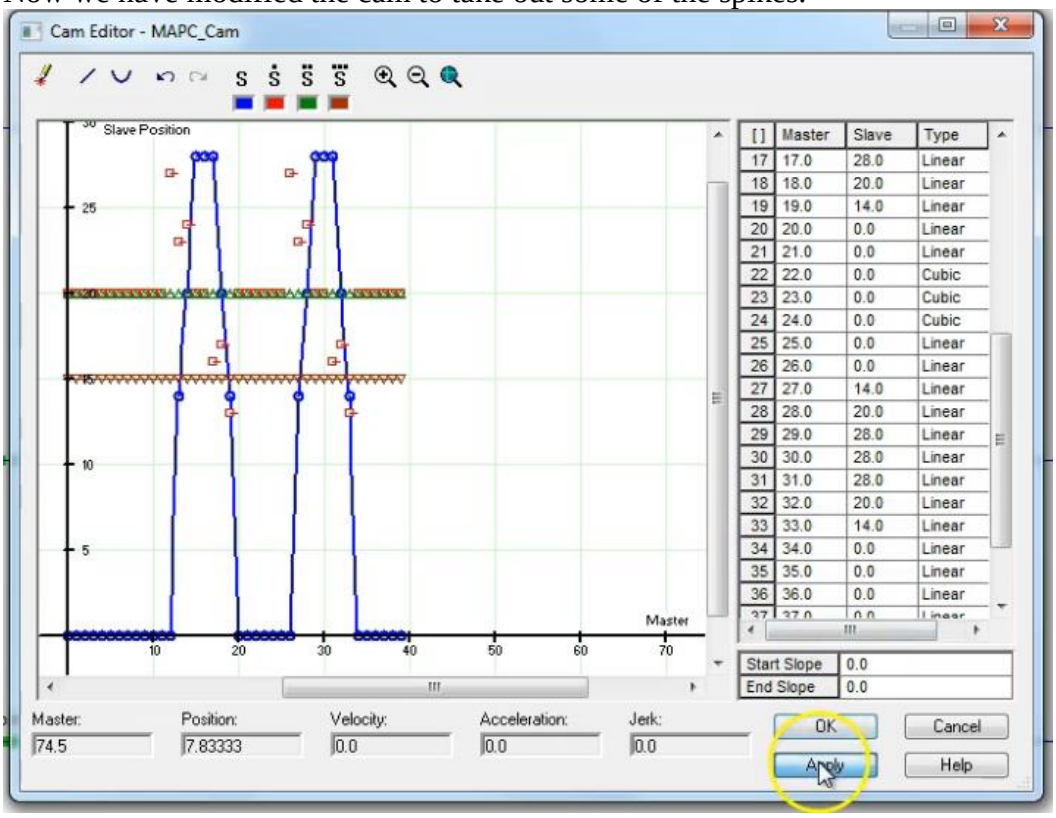



Now we have edited the cam so it more closely resembles a pusher.

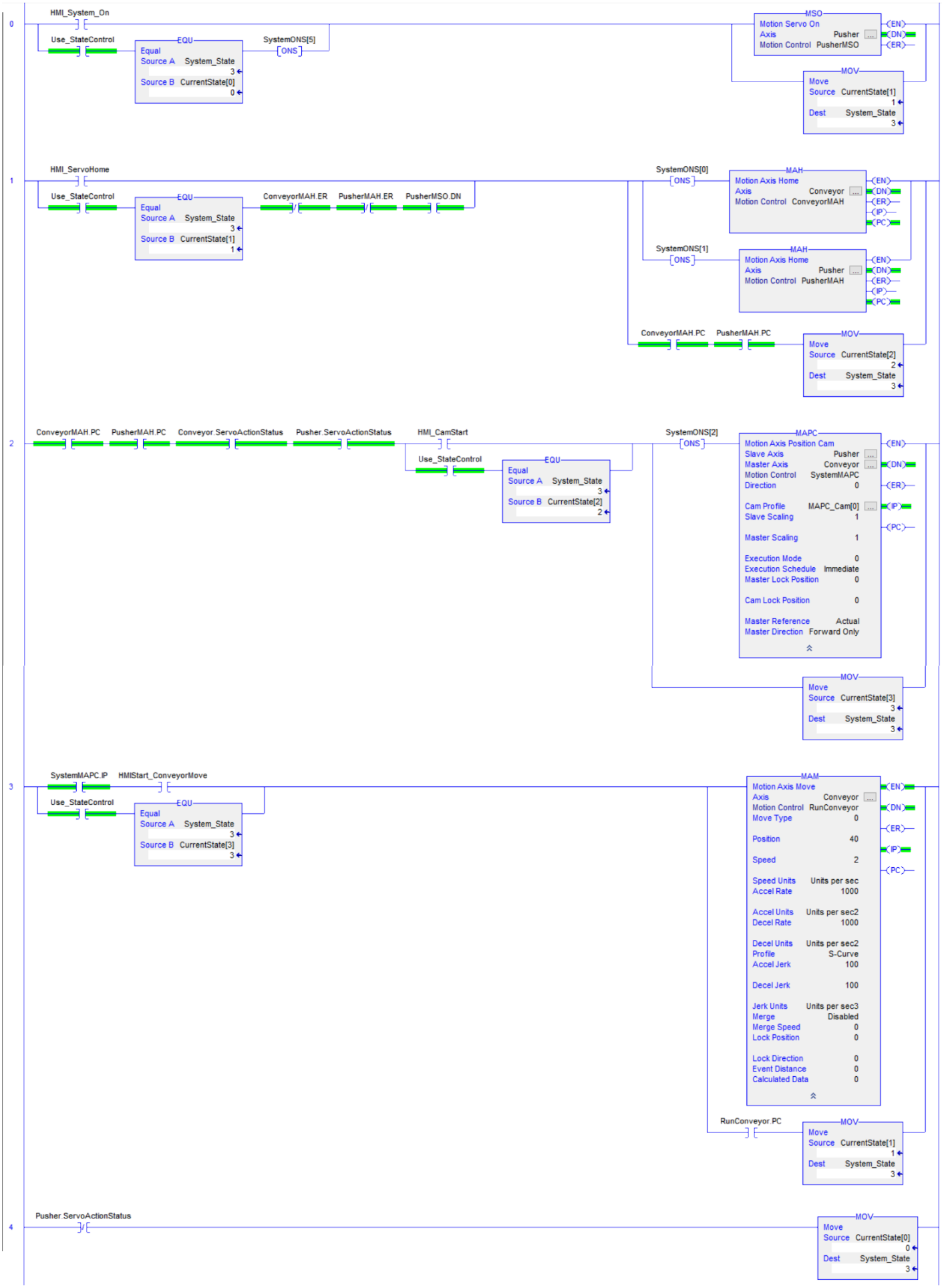




Now we have modified the cam to take out some of the spikes.

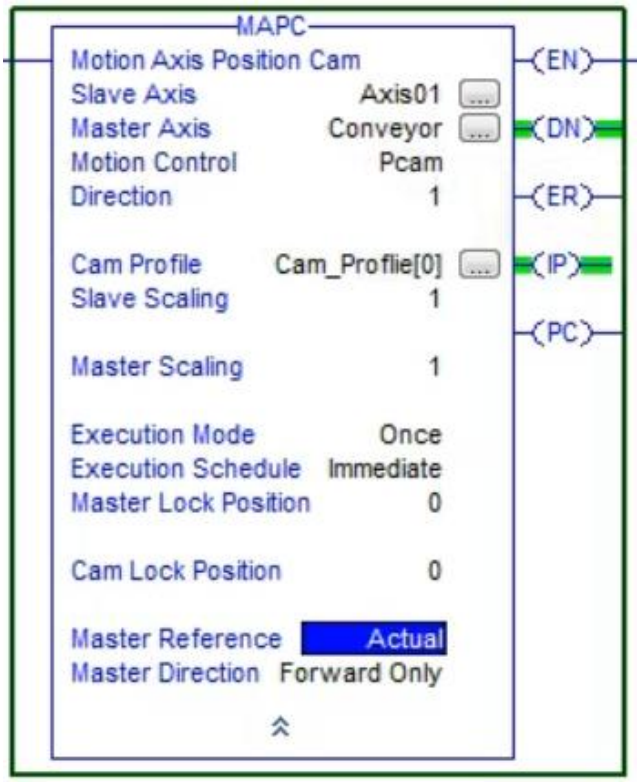



The video shows a version of the code where the master axis is reset once it hits 40 so the process repeats.


0
HMI_System_On
Use_StateControl
EQU
Equal
Source A System_State
3
Source B CurrentState[0]
0
SystemONS[5]
ONS
MSO
Motion Servo On
Axis Pusher
Motion Control PusherMSO
EN
DN
ER
MOV
Move
Source CurrentState[1]
1
Dest System_State
3
1
HMI_ServoHome
Use_StateControl
EQU
Equal
Source A System_State
3
Source B CurrentState[1]
1
ConveyorMAH.ER
PusherMAH.ER
PusherMSO.DN
SystemONS[0]
ONS
MAH
Motion Axis Home
Axis Conveyor
Motion Control ConveyorMAH
EN
DN
ER
IP
PC
SystemONS[1]
ONS
MAH
Motion Axis Home
Axis Pusher
Motion Control PusherMAH
EN
DN
ER
IP
PC
ConveyorMAH.PC
PusherMAH.PC
MOV
Move
Source CurrentState[2]
2
Dest System_State
3
2
ConveyorMAH.PC
PusherMAH.PC
Conveyor.ServoActionStatus
Pusher.ServoActionStatus
HMI_CamStart
Use_StateControl
EQU
Equal
Source A System_State
3
Source B CurrentState[2]
2
SystemONS[2]
ONS
MAPC
Motion Axis Position Cam
Slave Axis Pusher
Master Axis Conveyor
Motion Control SystemMAPC
Direction 0
Cam Profile MAPC_Cam[0]
Slave Scaling 1
Master Scaling 1
Execution Mode 0
Execution Schedule Immediate
Master Lock Position 0
Cam Lock Position 0
Master Reference Actual
Master Direction Forward Only
EN
DN
ER
IP
PC
MOV
Move
Source CurrentState[3]
3
Dest System_State
3
3
SystemMAPC.IP
HMIStart_ConveyorMove
Use_StateControl
EQU
Equal
Source A System_State
3
Source B CurrentState[3]
3
MAM
Motion Axis Move
Axis Conveyor
Motion Control RunConveyor
Move Type 0
Position 40
Speed 2
Speed Units Units per sec
Accel Rate 1000
Accel Units Units per sec2
Decel Rate 1000
Decel Units Units per sec2
Profile S-Curve
Accel Jerk 100
Decel Jerk 100
Jerk Units Units per sec3
Merge Disabled
Merge Speed 0
Lock Position 0
Lock Direction 0
Event Distance 0
Calculated Data 0
EN
DN
ER
IP
PC
RunConveyor.PC
MOV
Move
Source CurrentState[1]
1
Dest System_State
3
4
Pusher.ServoActionStatus
MOV
Move
Source CurrentState[0]
0
Dest System_State
3

## 32. MAPC Common Issue With Master Reference

In this lecture we will be discussing position cams. **There is a known problem with position cams**. This problem does occur in systems where the master cam is virtual. Say you are running a master cam and there is a physical motor connected. Using a Master Reference = Actual gives the system a more refined movement as opposed to Virtual.




Say you are in a situation where the motor is trying to hold itself in position and the cam profile senses that movement. So the IP bit of the Motion Axis Position Cam will trigger from IP (In Progress) to PC (Process Complete). It does this because it senses the movement.

If you have that problem it is a simple matter to change from Master Reference = Actual to Command and you will not have the problem. My guess is he means that in command mode the system waits for a command as opposed to monitoring physical movement and acting on that.

Also, any movement in the negative direction will cause the MAPC to transition from In Progress to Process Complete. This is a bug that RW knows about.

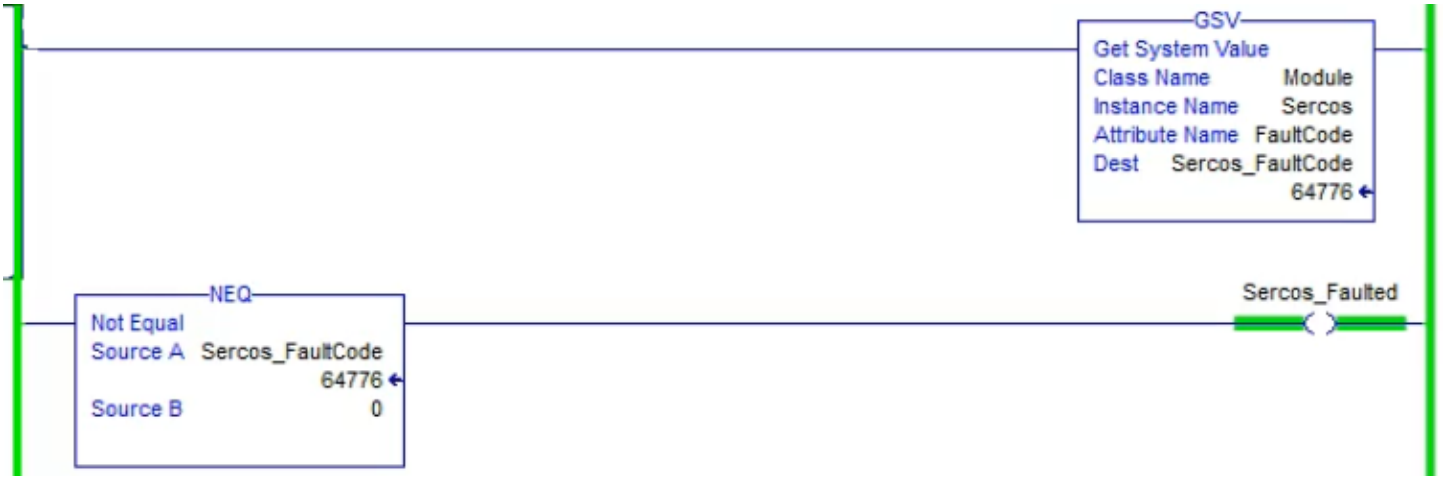
## 33. MAPC PC Bit explained

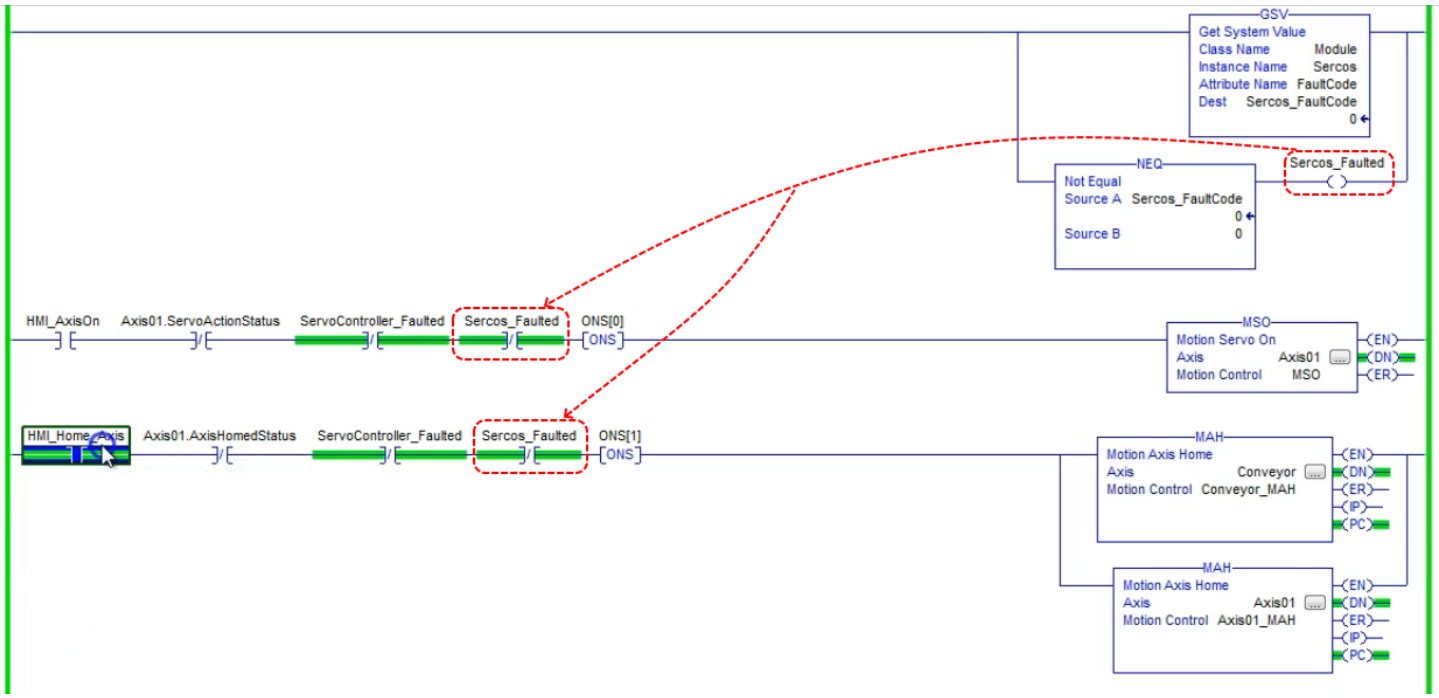
Here we are going to look further into the MAPC transition from IP to PC. "… as soon as the master axis goes out of the cam we see the transition from IP to PC." What he means is if we try to access a table point outside of our points. If we have a table with 10 rows and the program tries to access point 11, outside the table, it will transition in this way, from IP to PC. (which it should it would seem?)

## 34. MAPC Adding Common Health Checks

We have a servo and we want to turn it on, but we do not want to turn it on if it is already running. And we also want to make sure it is healthy before we start it. We can use the **GSV** function block with Class Name: Module, Instance Name: Kinetic_6000, Attribute Name: FaultCode, and Destination Tag: ServoController_FaultCode. Check the destination tag and if it is not equal to zero we know we have a fault.


GSV
Get System Value
Class Name Module
Instance Name Kinetix_6000
Attribute Name FaultCode
Dest ServoController_FaultCode
0
NEQ
Not Equal
Source A ServoController_FaultCode
0
Source B 0
ServoController_Faulted

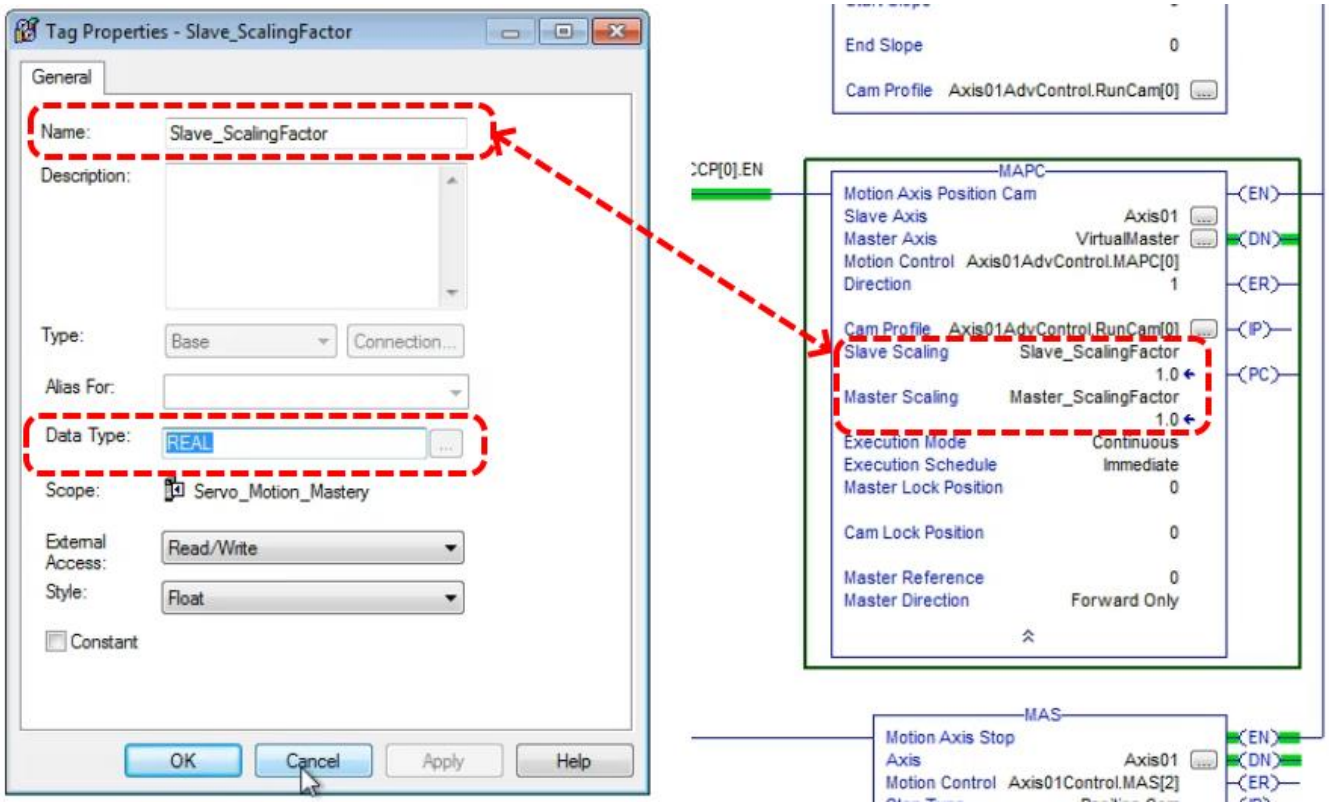

GSV
Get System Value
Class Name Module
Instance Name Sercos
Attribute Name FaultCode
Dest Sercos_FaultCode
64776
NEQ
Not Equal
Source A Sercos_FaultCode
64776
Source B 0
Sercos_Faulted



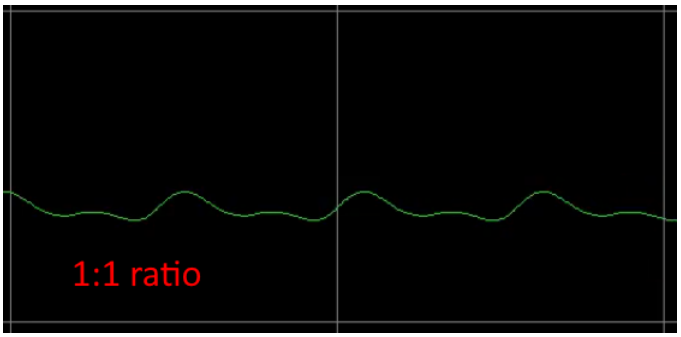

So we can now check for the conditions and take proper action. [IF NOT FAULTED PROCEED]

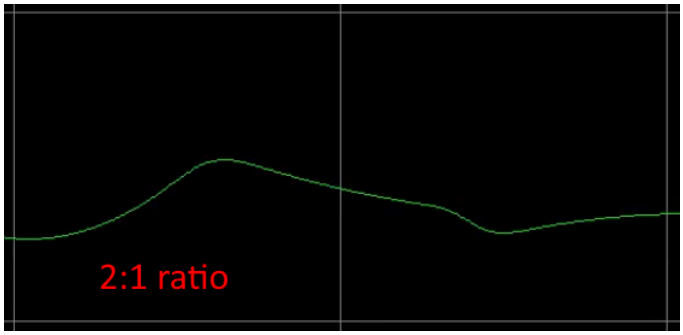
## 35. Use scaling in an MAPC instruction

The MAPC has master and slave scaling features. We want to look at how these values effect the servos motion.




As we might expect, changing the scale factor has a corresponding effect on the amplitude of the movement.




The way this works:

- making adjustments to the **cycle time** is done by changing the master scaling factor.
- Making adjustments to the **amplitude** is done by scaling the slave.

## 36. MAPC Cam Lock Position Function Talk - Not program and Show

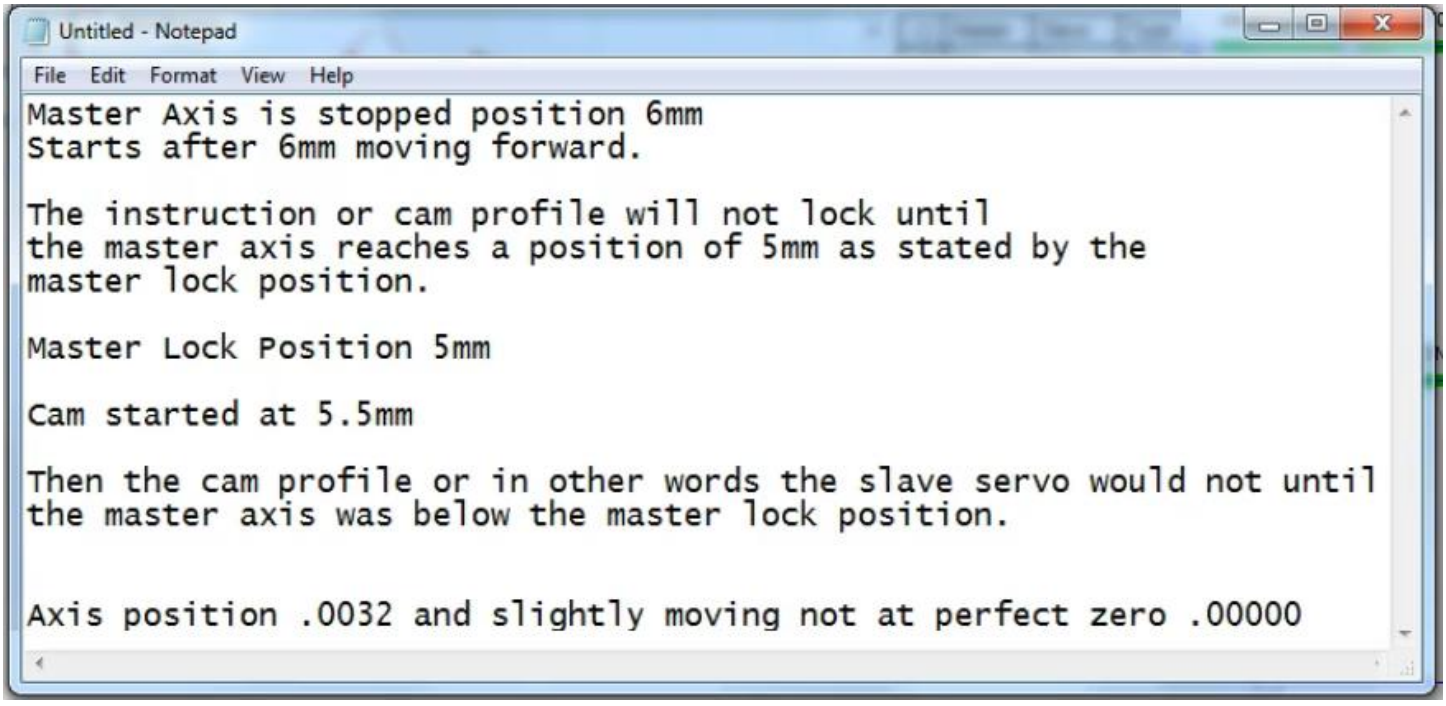
Not much, kind of just babbling.

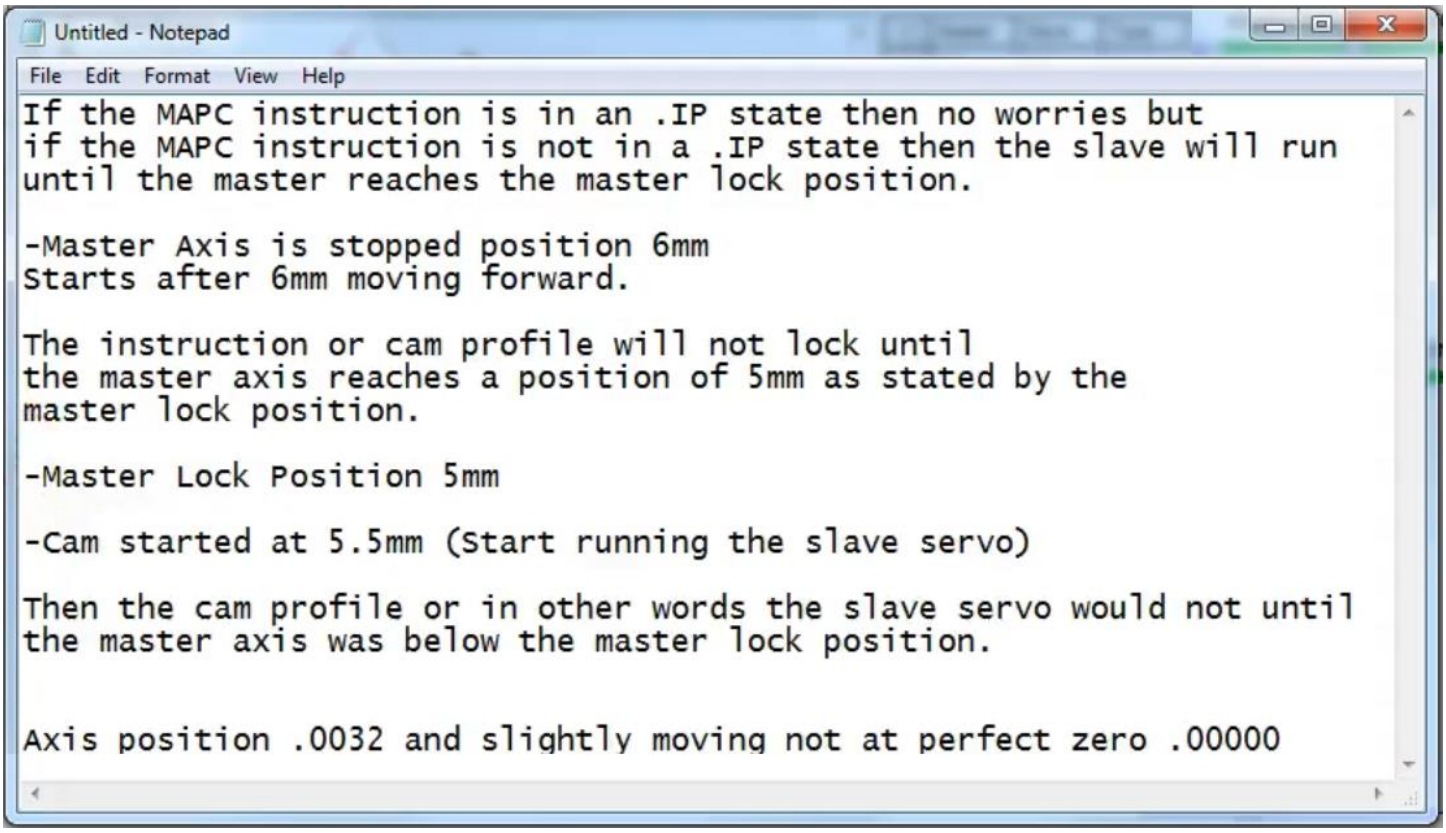
## 37. MAPC Master Lock Position Function - Not program and Show

Two points that are often times missed include **Master Lock Position** and **Cam Lock Position**. If the master axis is greater than the master lock position then it is not going to 'pick-up' until the next time the master axis is actually below that mark.

*From Shane: Say the master axis is stopped at position 6 mm and the Master Lock Position is set at 5 mm and the cam is started at 5.5 mm. Then the cam profile (also known as the slave) would not (start?) until the master axis was below the master lock position.*

Another example:






## 38. Difference In A MAPC and a MATC

Here we will discuss the differences between a time cam and a position cam.

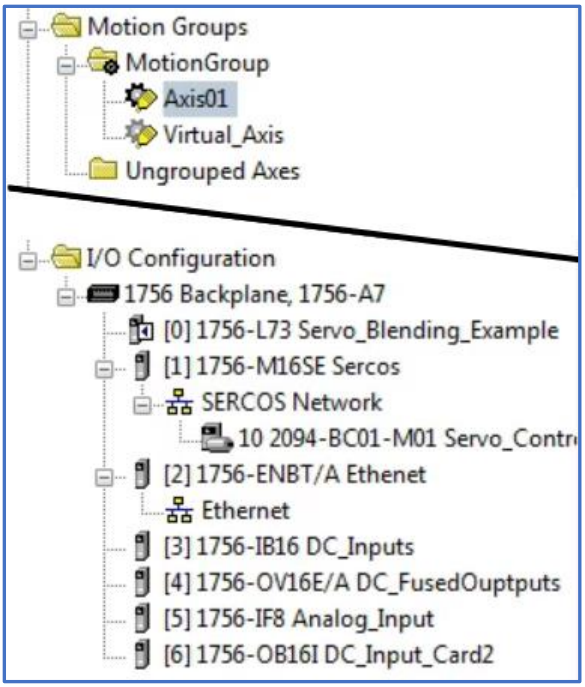
First the **position cam**. We've spoken of how the IP and PC bits work, they are position based on the master, master and slave coincide based on position. It boils down to understanding the **master reference**, whether it is set for **actual** or **command**. Meaning, is it the **actual** value of the encoder coming back or is it the **commanded** value that the master is being told to do? Using the position of the master to control the slave, operating based on another servo (the master). (seems like actual would be the way to go)

Now the **time cam**. In this case it is the time which it takes to do something. If you look at the distance the servo has to travel and the amount of time it has to travel that distance, these two constraints are used to determine the acceleration and decelleration as well as the velocity of the servo. You are doing something in a set time frame.

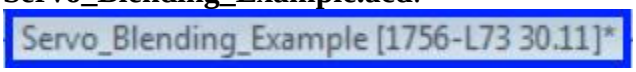
If you are the only controller controlling the servos (**master**) then you need to **enable time sychronization**.

**If your controller is a slave to another controller you do not need to enable time sychronization, you need your controller to be time sychronous with the other controller.**

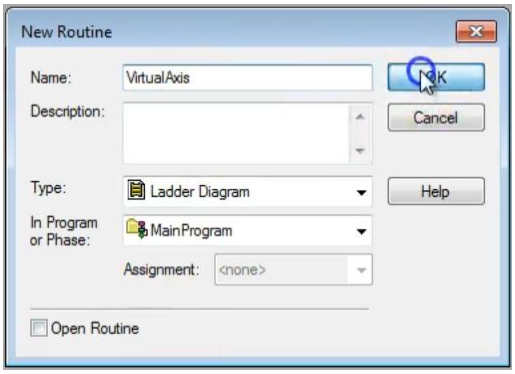
# 39. Starting from Scratch



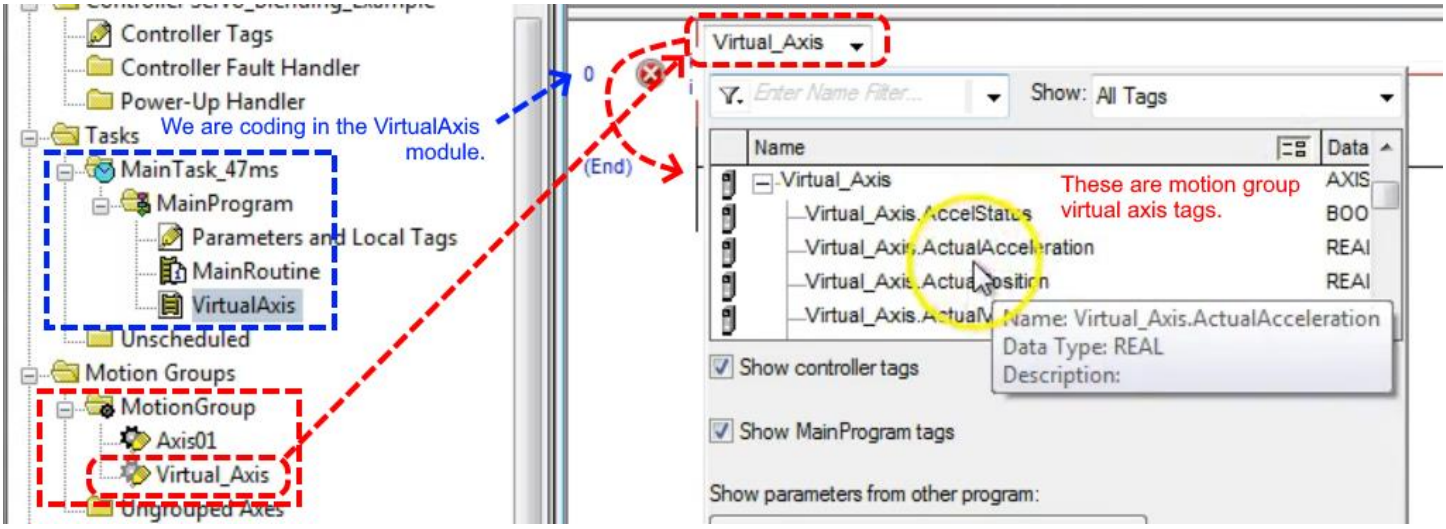

Here we begin a series of lectures on MAPC caming and how to **blend cams together**. We will work with a physical axis (Axis01) and a virtual axis. The name of this program is **Servo_Blending_Example.acd**.



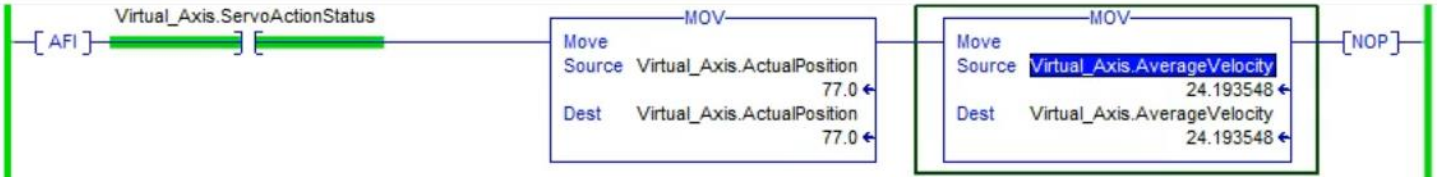

To begin we will add a routine which will represent the virtual axis.




Notice below that we want to add a contact from the virtual axis so we use the drop down to access the virtual axis, do not confuse with the code module VirtualAxis. Also, when you create a virtual axis a lot of tags are generated.




This rung will never execute due to the AFI statement. The purpose of the rung is to display the virtual axis average velocity and actual position so the programmer can see it. The .ServoActionStatus tag will only be true when the virtual axis is running. Note, it is usually better to monitor average velocity.

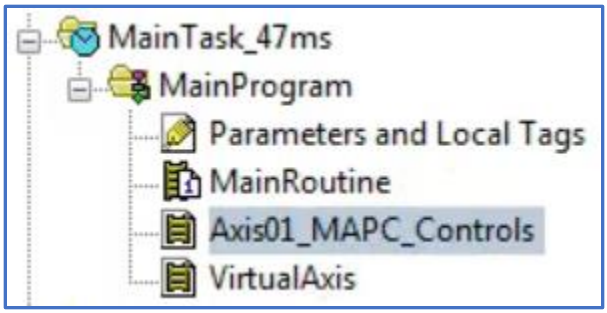


Next we configure a rung for jogging.


Now a command to stop the axis when we are not jogging.

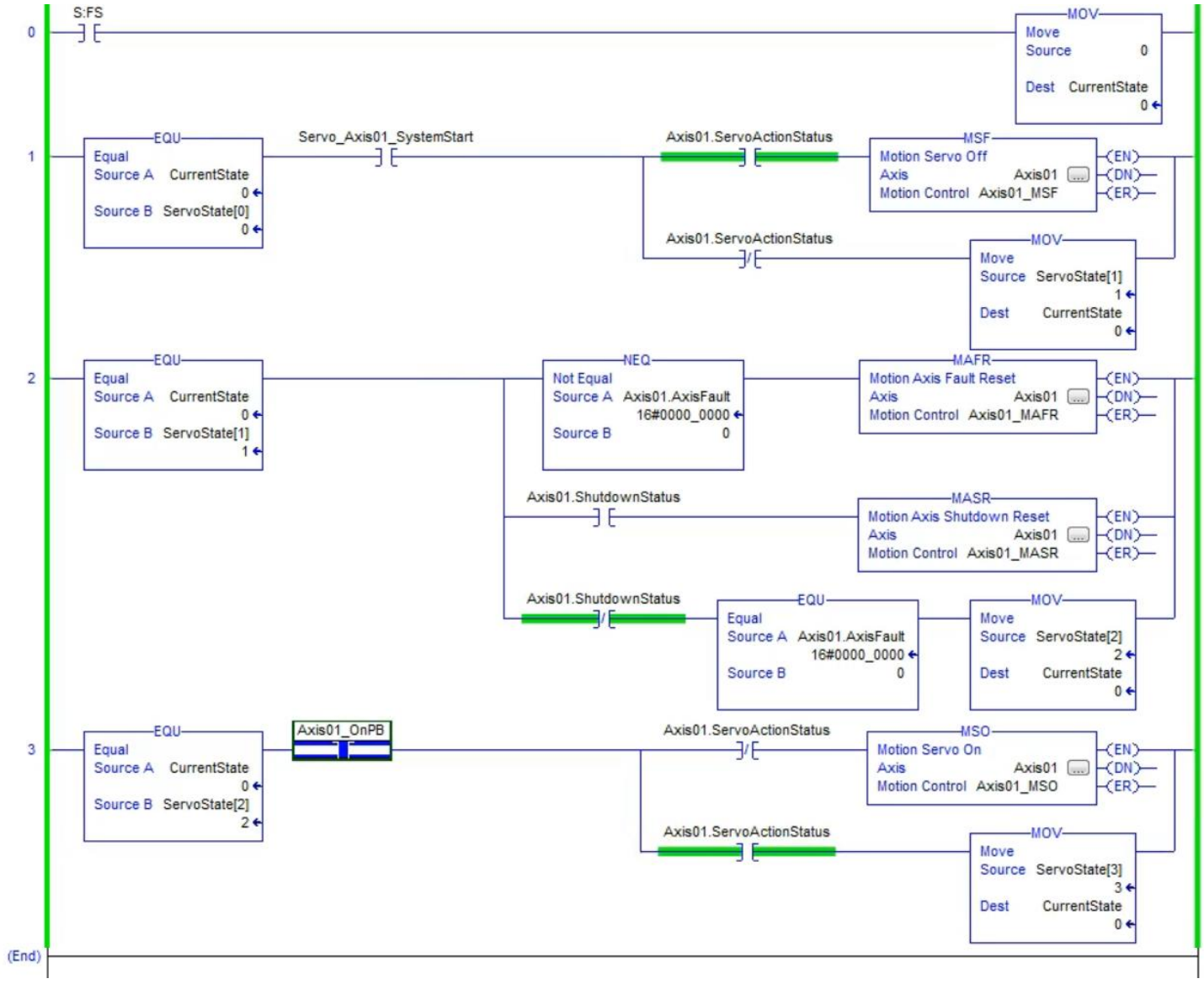
# 40. Setting up the Physical Axis

Now we are ready to start the programming for Axis01. First we add a new code module Axis01_MAPC_Controls.



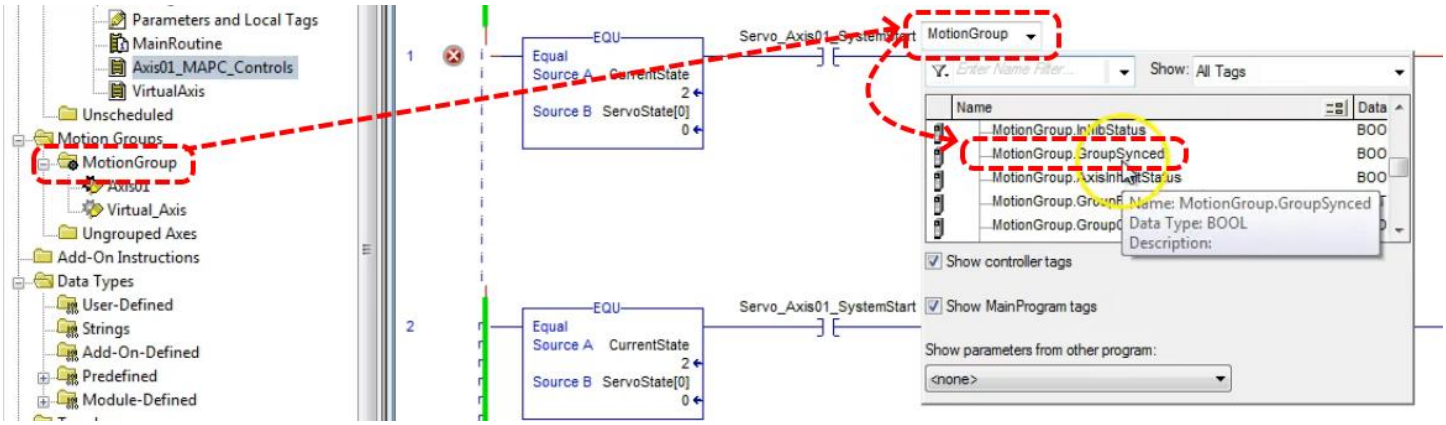

| ServoState | {...} |
|---|---|
| ServoState[0] | 0 |
| ServoState[1] | 1 |
| ServoState[2] | 2 |
| ServoState[3] | 3 |
| ServoState[4] | 4 |
| ServoState[5] | 5 |
| ServoState[6] | 6 |
| ServoState[7] | 7 |
| ServoState[8] | 8 |
| ServoState[9] | 9 |

Rough out of the program so far: make sure axis is off, clear any faults, perform a shutdown reset, turn servo on.

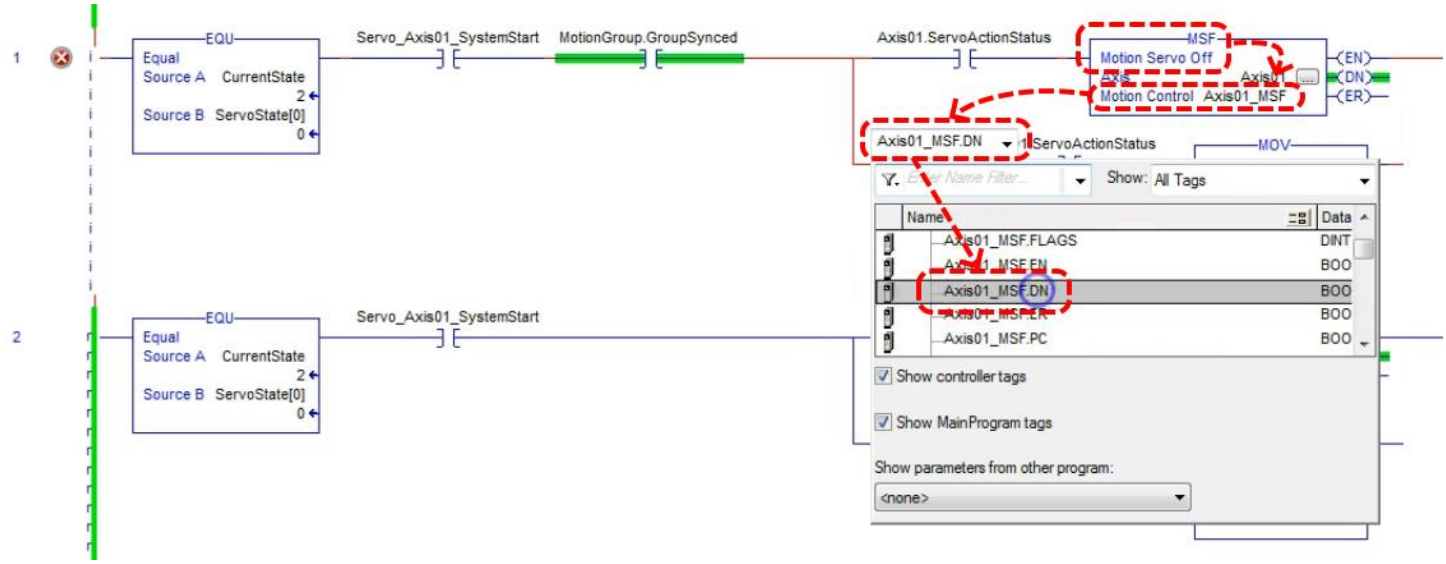


# 41. Adding elements for the state controls

Now we are going to add some controls to the state machine. To start we want to monitor our motion group. We will add a tag to the code that describes the motion group. **This particular tag lets us check that the group is synchronized**. This lets us know that the group is ready to operate and running the functions that follow will not cause a fault.

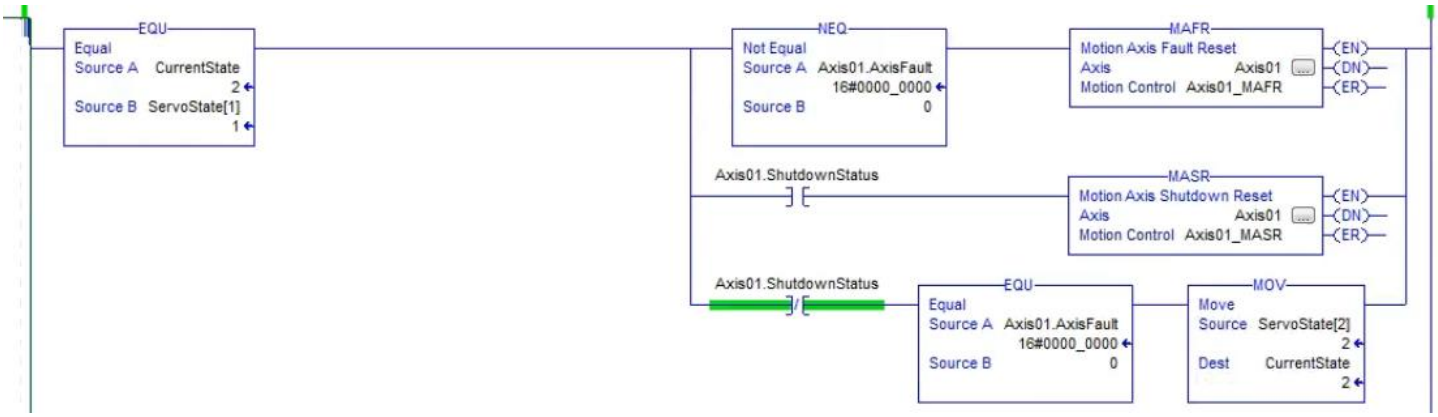



This next tag lets us check that the instruction actually did happen. To do this we check the Done bit.

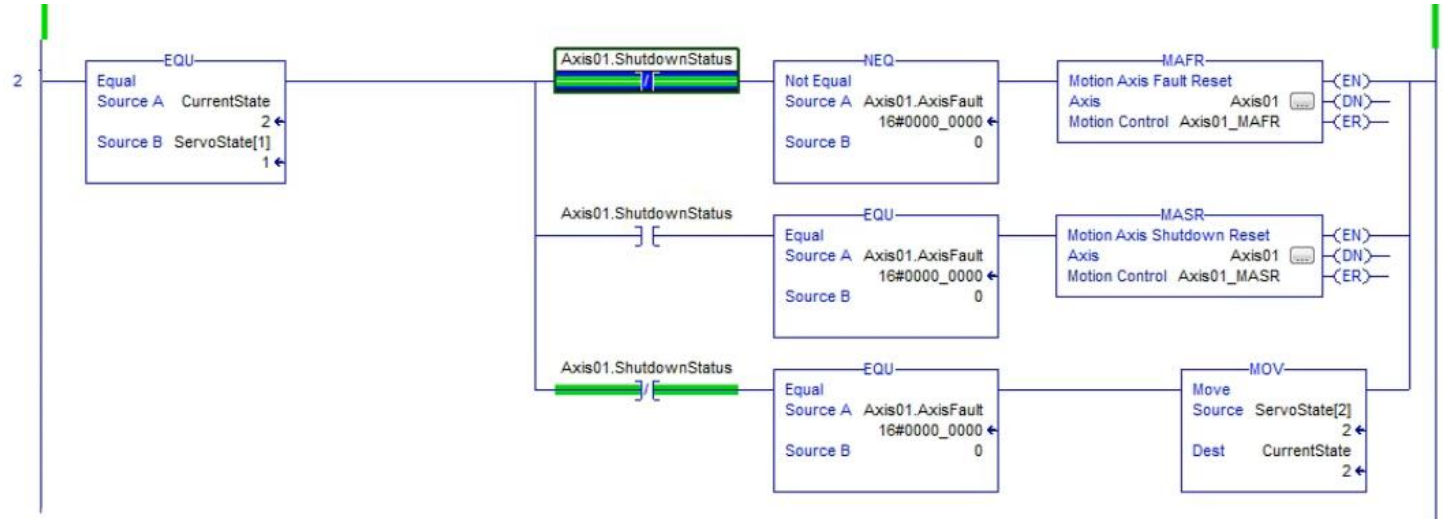



Now we would like a more robost method of reseting our state machine to zero, we do not want to depend only on the first scan rung so we are going to augment that rung. We will add a CLR (clear) function block. So now there are two possible ways of resetting the state machine, the first is the first scan function, the second is if a shutdown happens or if a fault happens. Under these conditions we would want to stop control and stop motion. So our first rung now looks like this.

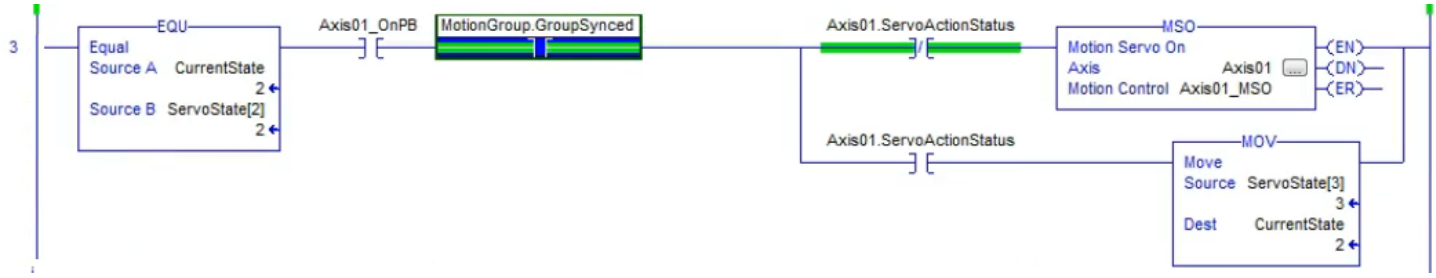

Next, in the below group, we want to make sure that we are only resetting one at a time. The group currently looks like this…



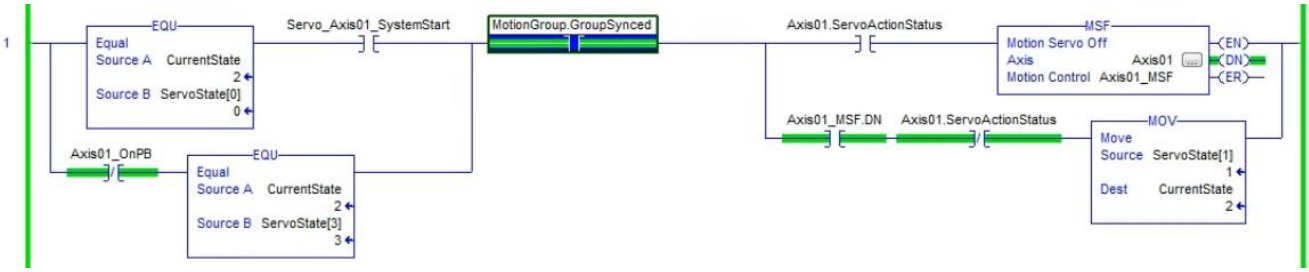

Now we are going to say, if it is faulted we will reset the fault only and if it is shut down, we will only activate the Motion Axis Shutdown function. Notice that if it is a shutdown we only want to activate the Motion Axis Shutdown Reset if we are not faulted. So we add the EQU statement to check for a zero in the fault register. It would actually be rare that a fault and shutdown happen at the same time. Our code block now looks like this…



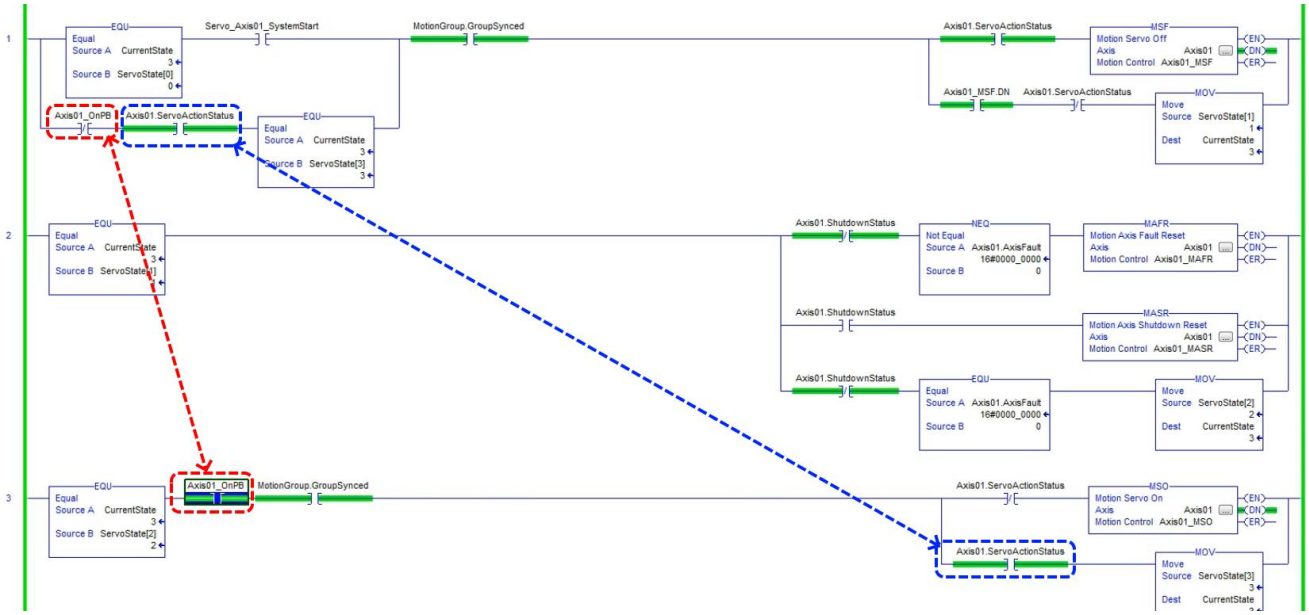

We are also going to add a tag on rung 3 which ensures that the motion group is synced before we turn on the servo and advance to the next group.



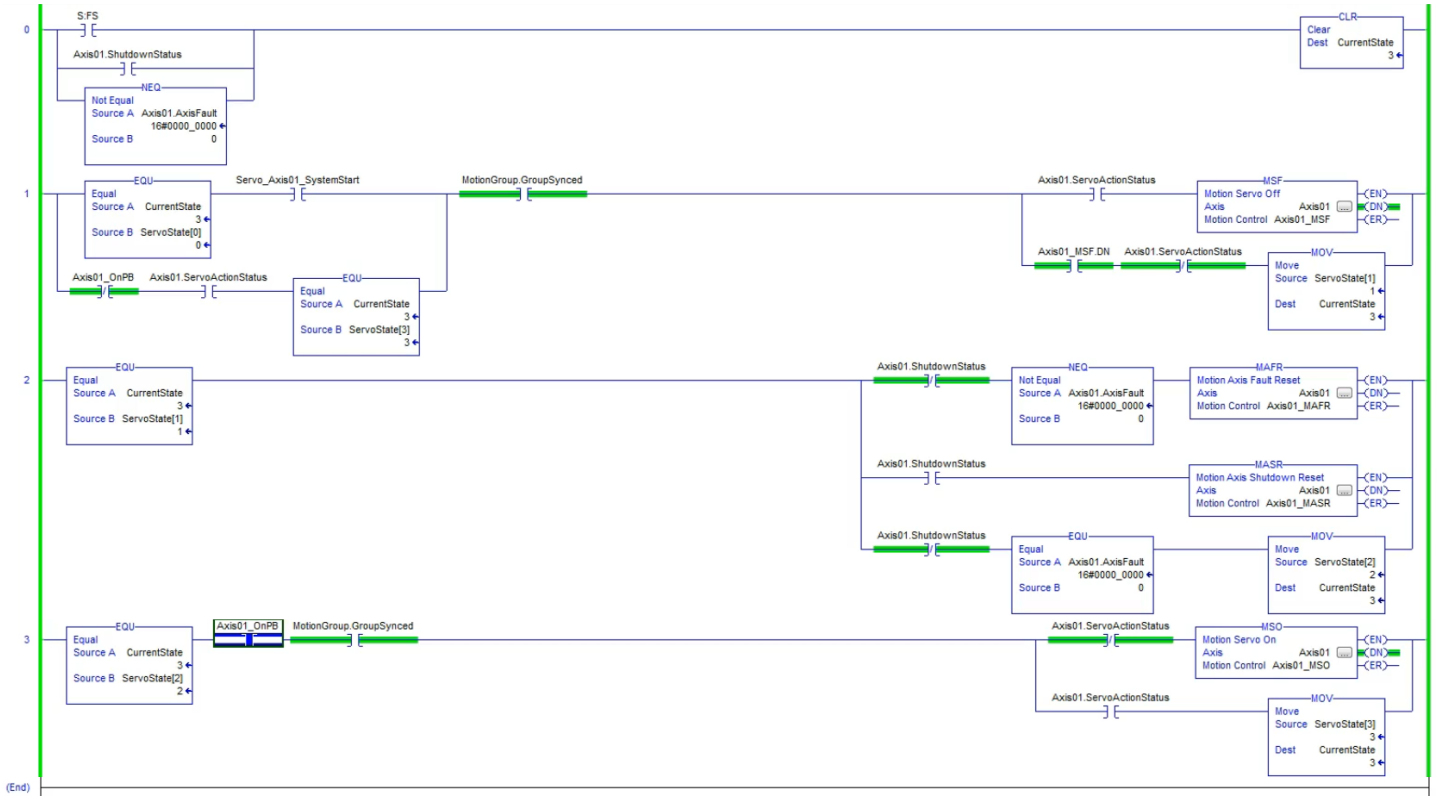

We have also added a ladder to rung 1 which forces the ServoState[3] to 2 when the Axis01_OnPB is not being pressed.




Another tag is added to ensure that if the Axis01_.ServoActionStatus is on we can cut it back off.

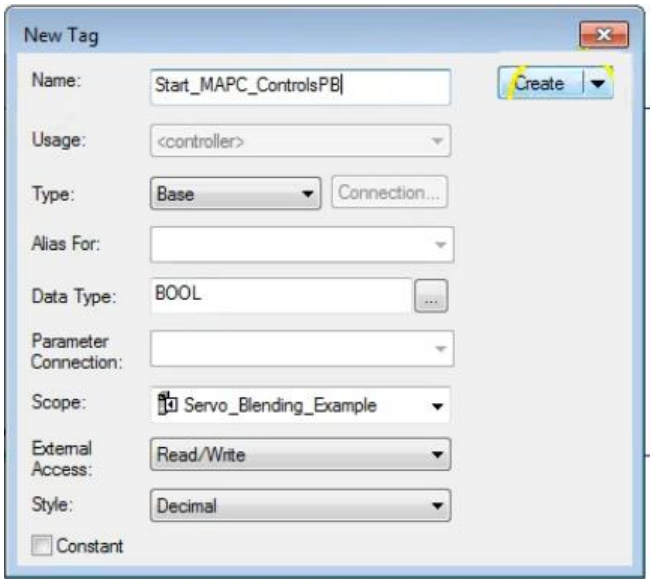



Application to this point.

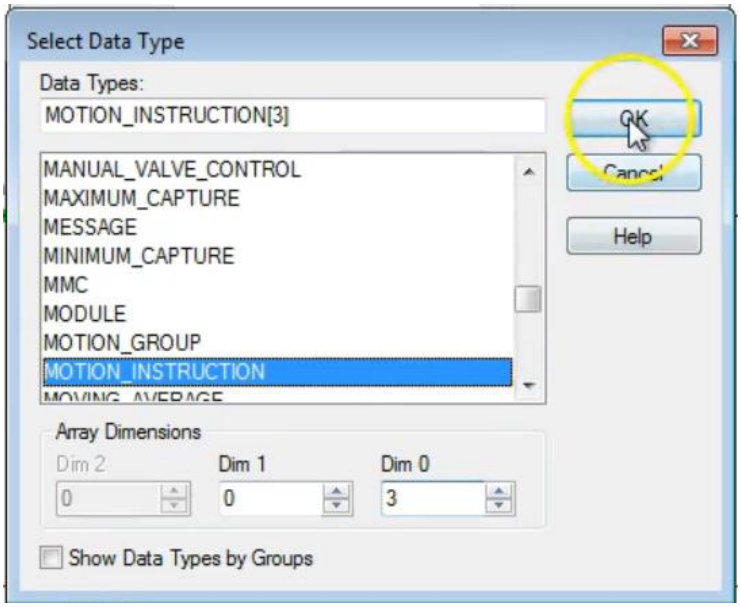


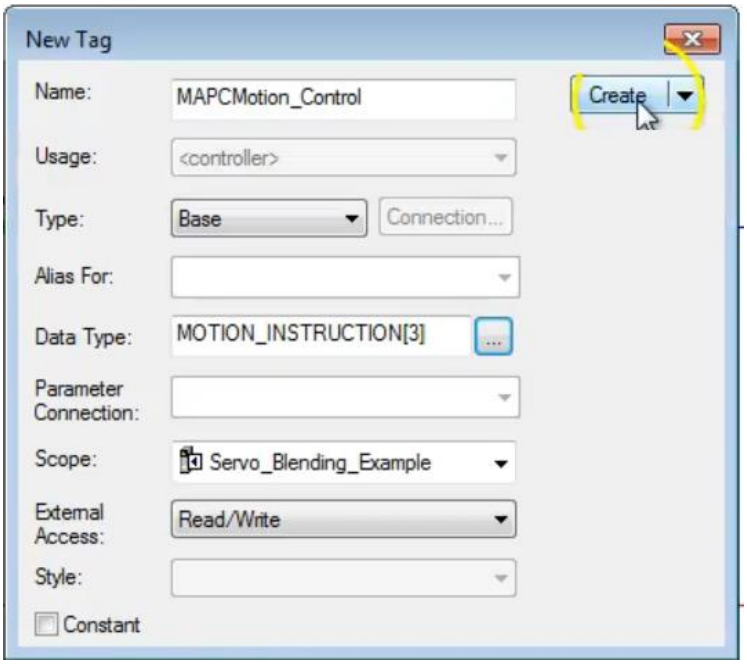

## 42. Adding our first MAPC

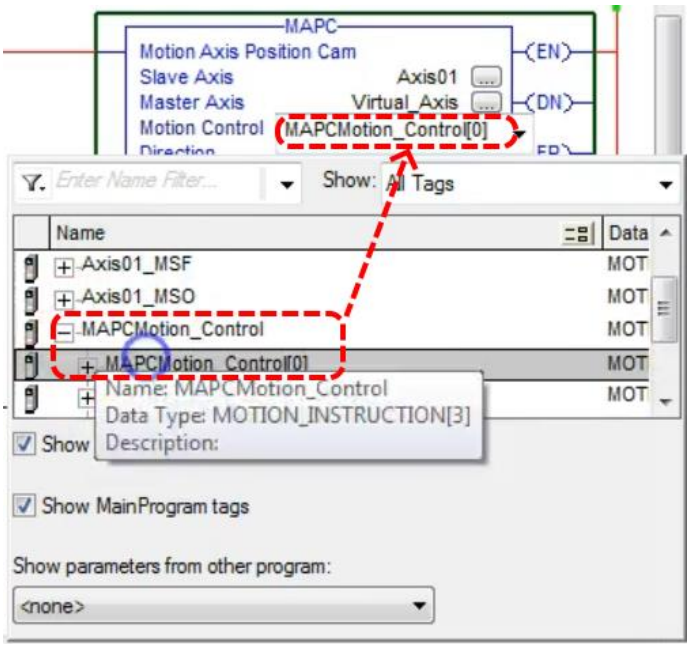
We add a new state at the bottom and add a new tag.




We begin with MAPC, the slave axis is Axis01, the master axis is Virtual_Axis, we will call it MAPCMotion_Control and give it a dimension of 3 as we will need several.

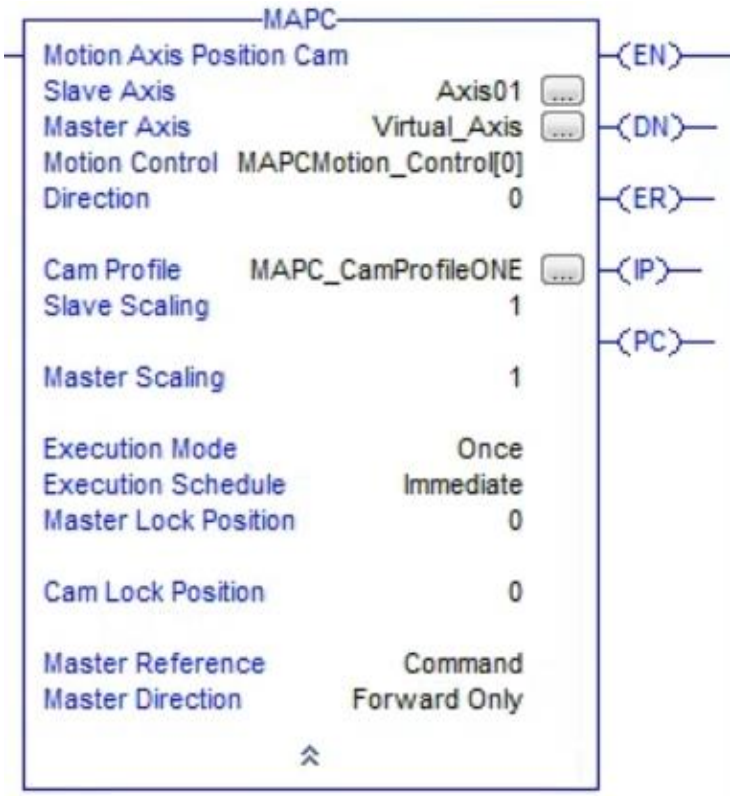




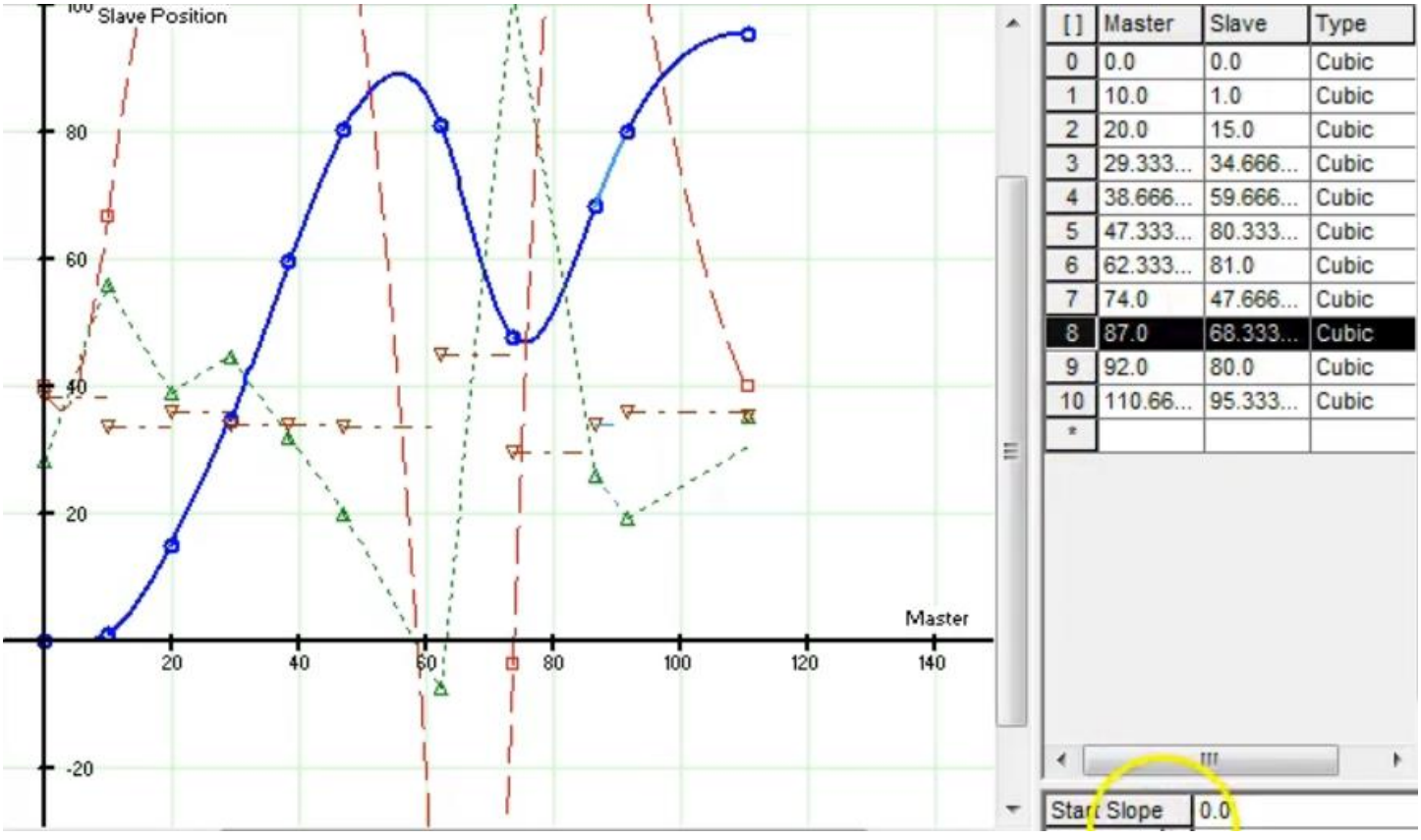




Below the help file for Direction is listed. We will use Same. Next we will have to create a cam profile, we will declare the tag MAPC_CamProfileONE. The remaining entries are shown in the graphic below.

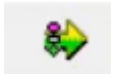
| | | | | |
|---|---|---|---|---|
| Direction | UINT32 | UINT32 | Immediate or Tag | Relative direction of the slave axis to the master axis:<br>• Same – the slave axis position values are in the same sense as the master's.<br>• Opposite – the slave axis position values are in the opposite sense of the master's.<br>Or relative to the current or previous camming direction:<br>• Reverse – the current or previous direction of the position cam is reversed on execution. When executed for the first time with Reverse selected, the control defaults the direction to Opposite.<br>• Unchanged – this allows other cam parameters to be changed without altering the current or previous camming direction. When executed for the first time with Unchanged selected, the control defaults the direction to Same. |

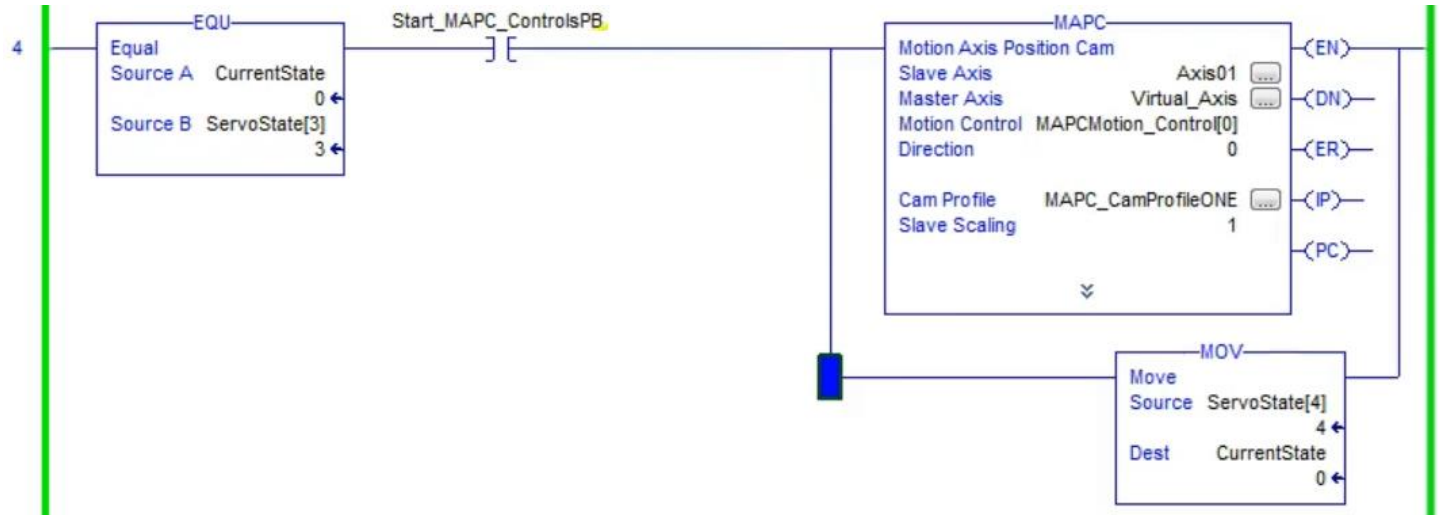




Next we will build up the cam profile. Below is our first draft. Note the velocity (red line) is kind of extreme.



We have also added a ladder to get us to the next state. The rung we are adding executes the MACP block but then immediately goes to state 4. To run, remove Axis01.ServoActionStatus at rung 1 and toggle

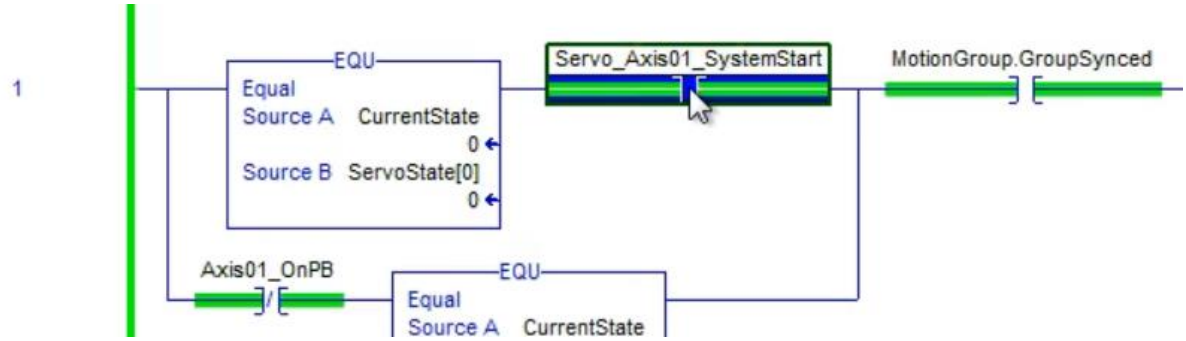
Servo_Axis01_SystemStart also on rung 1. Click  to compile.




Now go up to rung 1 and toggle Servo_Axis01_SystemStart to run the cam.

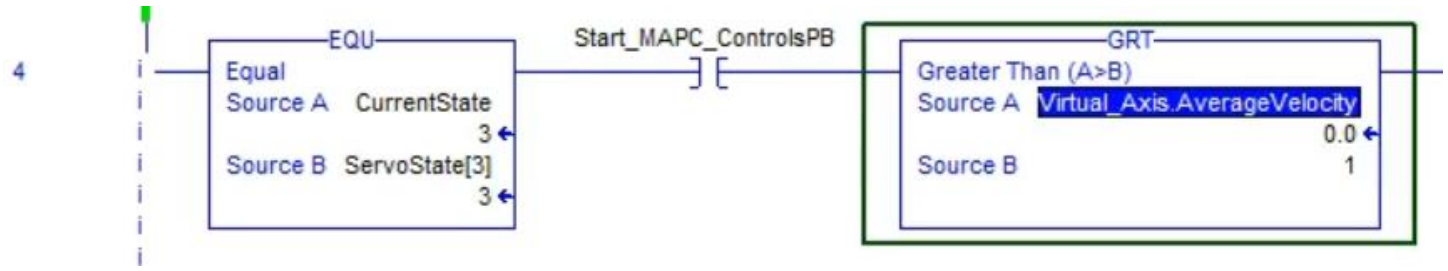
We will want to monitor some tags like average velocity, we want average virtual velocity greater than 0 (use "1" in GRT block) before we execute the Motion Axis Position Cam (MAPC) instruction. This code modification is shown on the next page. Again, this greater than instruction is ensuring the virtual axis is on and moving before we execute the MAPC instruction.

In this case the virtual axis is the master which Axis01 will follow according to the entries in the slave column.

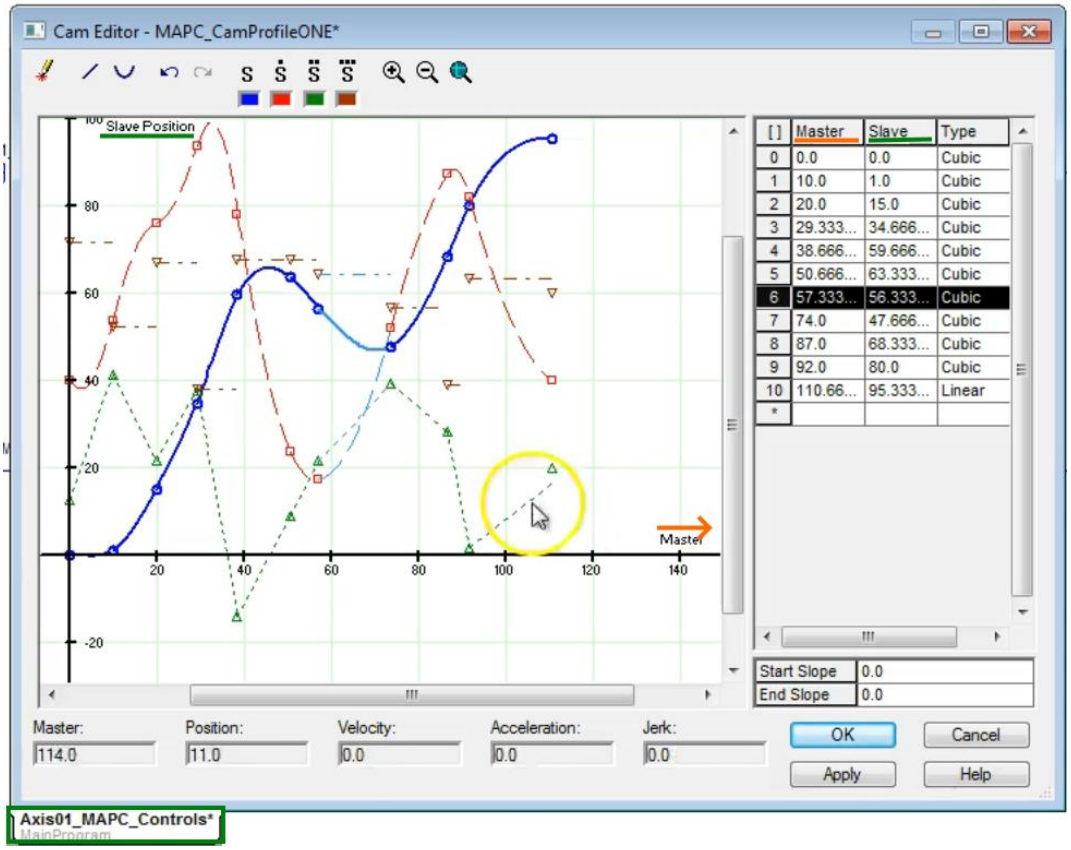



Note that we need to verify that the virtual axis is actually on. To do this we can monitor the velocity of the virtual axis. We can view the virtual axis code in the VirtualAxis module.

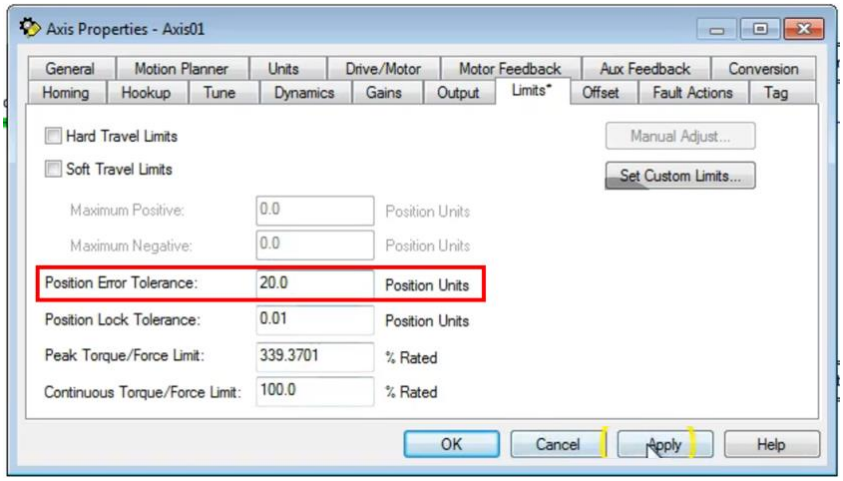

Before we turn on the MAPC we want to check that the Virtual_Axis.AverageVelocity is greater than zero.



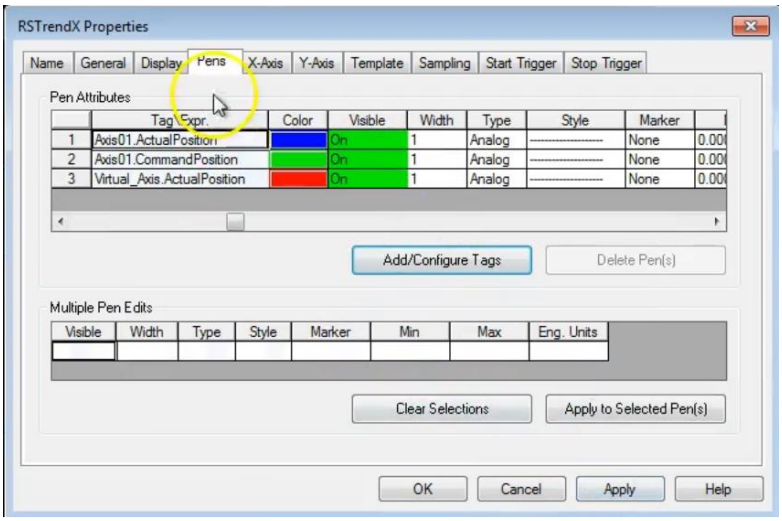
Note that in this case the master axis is virtual. Now run the virtual axis from the VirtualAxis_Jog contact in the VirtualAxis code module. So when we ran the virtual axis is triggered a fault, we will modify the curve to soften the movement.

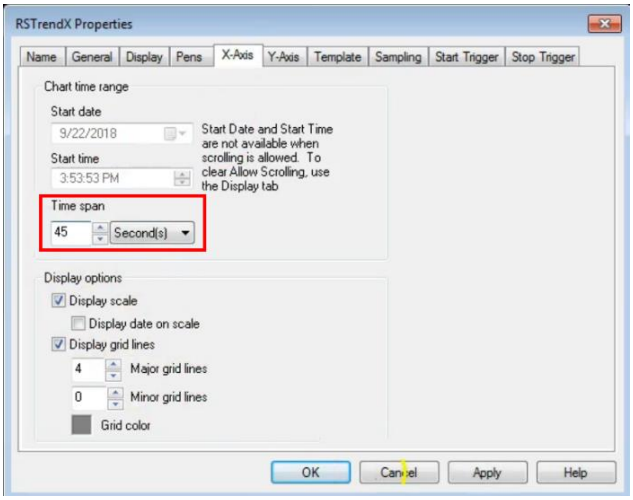


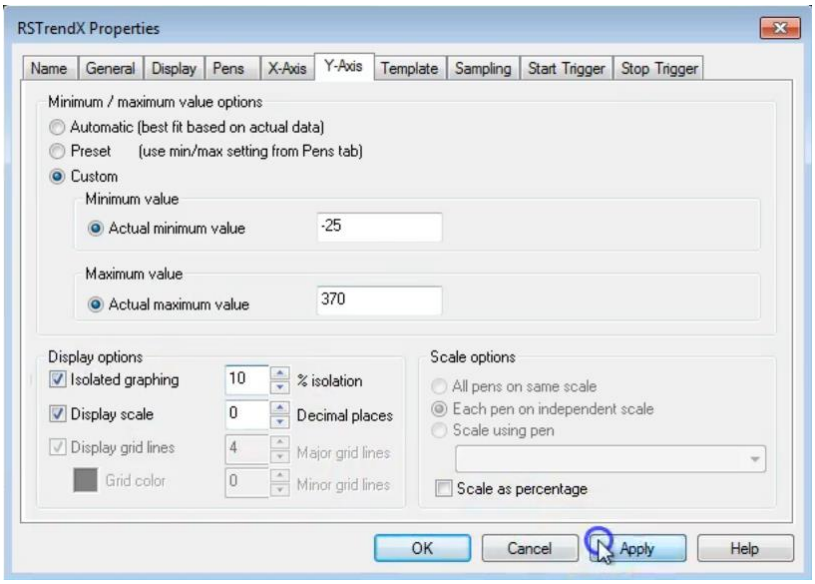

We try running again and find that we are faulting on a position error. Open the Axis Properties for Axis01 and increase the position error to 20.0.

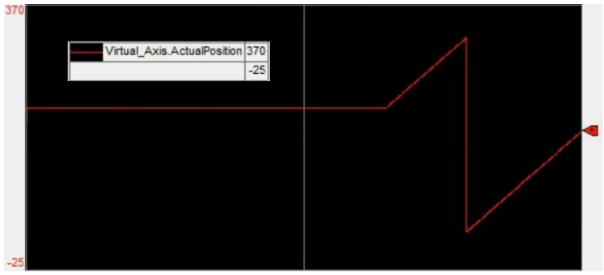



Now set up a trend. Add the actual and command position tags. Set the time scale to 45 seconds.


RSTrendX Properties
Name
General
Display
Pens
X-Axis
Y-Axis
Template
Sampling
Start Trigger
Stop Trigger
Pen Attributes
Tag, Expr.
Color
Visible
Width
Type
Style
Marker
1
Axis01.ActualPosition
On
1
Analog
None
2
Axis01.CommandPosition
On
1
Analog
None
3
Virtual_Axis.ActualPosition
On
1
Analog
None
Add/Configure Tags
Delete Pen(s)
Multiple Pen Edits
Visible
Width
Type
Style
Marker
Min
Max
Eng. Units
Clear Selections
Apply to Selected Pen(s)
OK
Cancel
Apply
Help

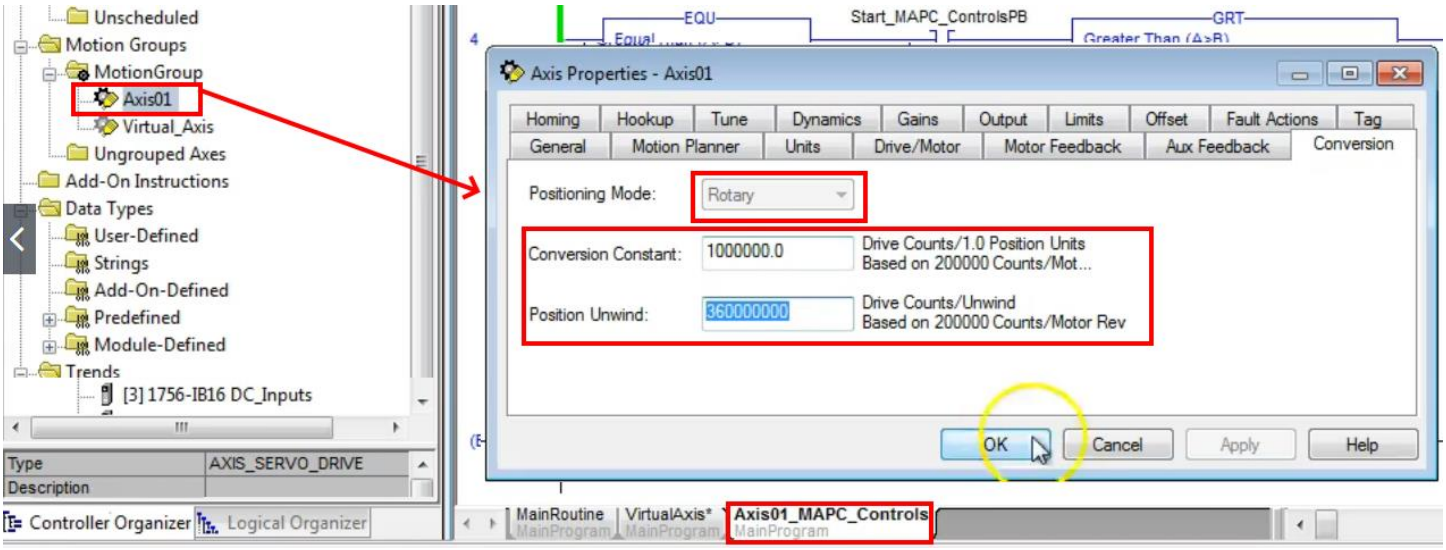

RSTrendX Properties
Name
General
Display
Pens
X-Axis
Y-Axis
Template
Sampling
Start Trigger
Stop Trigger
Chart time range
Start date
9/22/2018
Start time
3:53:53 PM
Start Date and Start Time are not available when scrolling is allowed. To clear Allow Scrolling, use the Display tab
Time span
45
Second(s)
Display options
Display scale
Display date on scale
Display grid lines
4
Major grid lines
0
Minor grid lines
Grid color
OK
Cancel
Apply
Help


RSTrendX Properties
Name
General
Display
Pens
X-Axis
Y-Axis
Template
Sampling
Start Trigger
Stop Trigger
Minimum / maximum value options
Automatic (best fit based on actual data)
Preset (use min/max setting from Pens tab)
Custom
Minimum value
Actual minimum value
-25
Maximum value
Actual maximum value
370
Display options
Isolated graphing
10
% isolation
Display scale
0
Decimal places
Display grid lines
4
Major grid lines
Grid color
0
Minor grid lines
Scale options
All pens on same scale
Each pen on independent scale
Scale using pen
Scale as percentage
OK
Cancel
Apply
Help

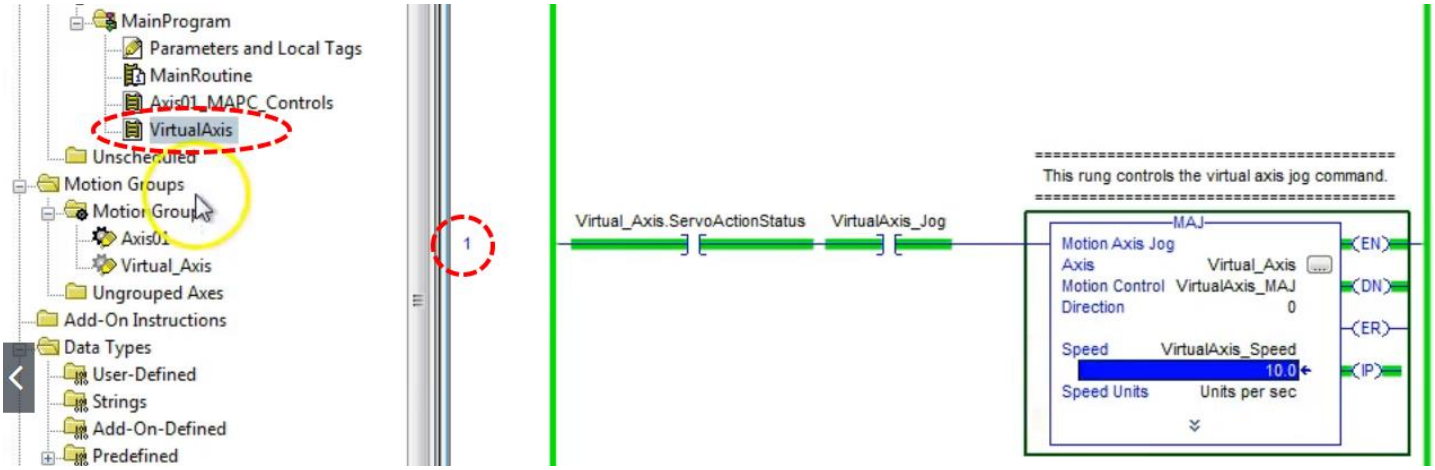

370
Virtual_Axis.ActualPosition
370
-25
-25

## 43. Adding the Second MAPC for Cam Bending

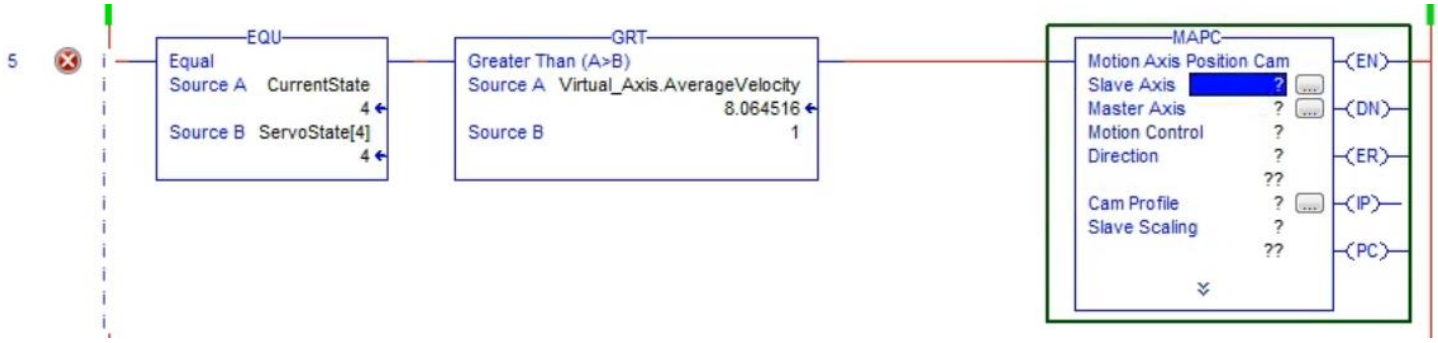
At this point there have been changes to the program in order to add the next MAPC. These changes are outlined in the next paragraphs.




Changing to a rotary axis and Setting the axis to roll over (position unwind) to 360 degrees saves us from having to home the axis.

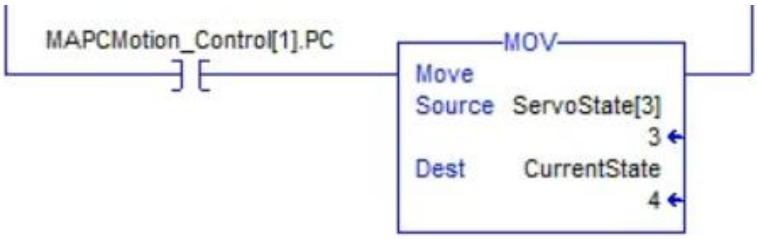



In the above graphic we have changed the virtual axis speed to 10 (decreased).




Above is the rung we are adding. Note below that we are using the next cam profile, [2]. MAPC_CamProfileTWO
Start with 1-to-1 scaling and execution mode of once and schedule of pending. **Master Lock Position** and **Cam Lock Position** refer to the position value to be used as the "lock" point. In this case our highest master value is 110.66 so we choose 111.0 for our lock position. It is a point outside the range of movement but also where we want the axis to stop movement. (lock up) We are using the same 111.0 for **Cam**.

We are going to add some logic to our rung. First, we want to wait for the MAPC to complete (.PC) and then we will move a 3 into the CurrentState variable so we can repeat/loop.



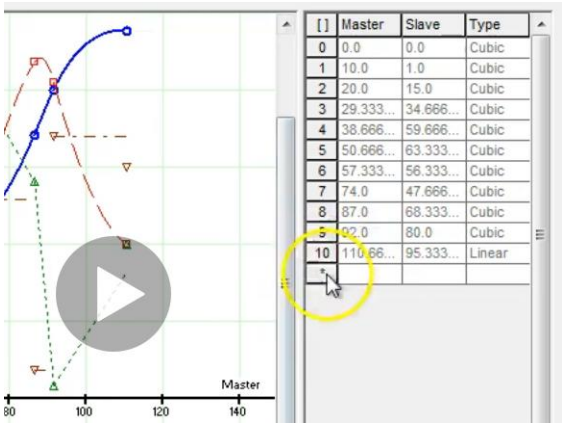
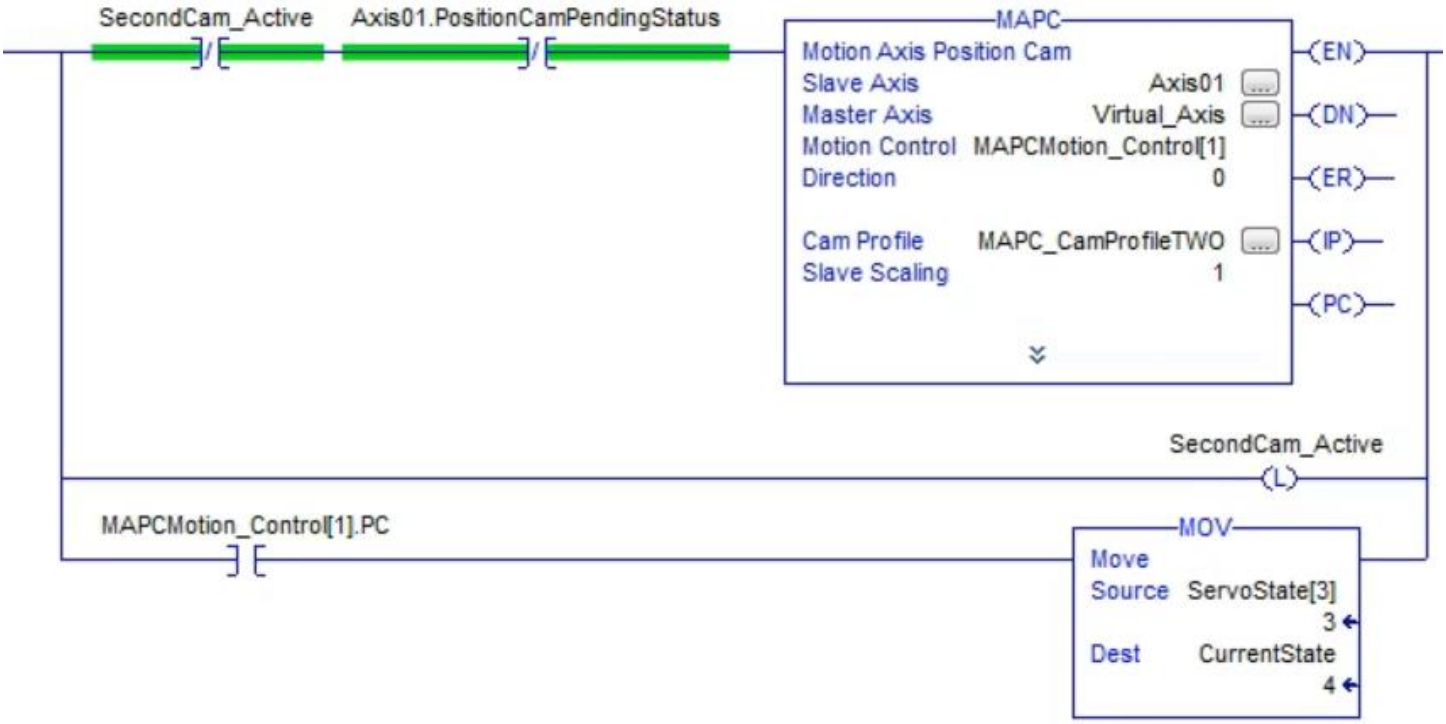
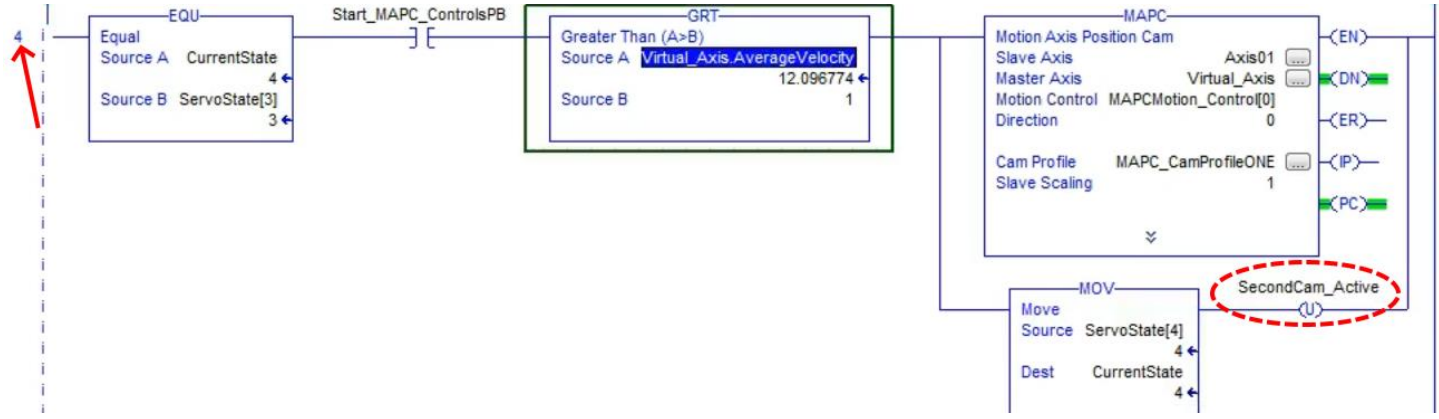




*Figure 5 selecting master and cam lock position.*

We will also add a pending status, we want to know if the axis is in action. We can tell this via tag **Axis01.PositionCamPendingStatus**. Next we add a latch for second cam active.

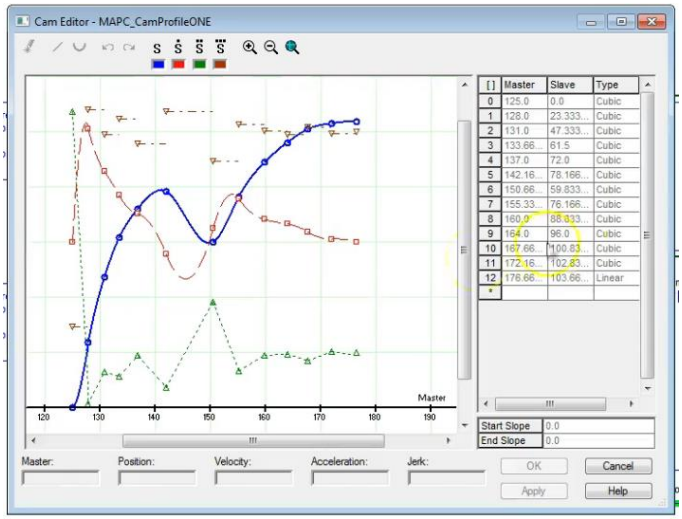


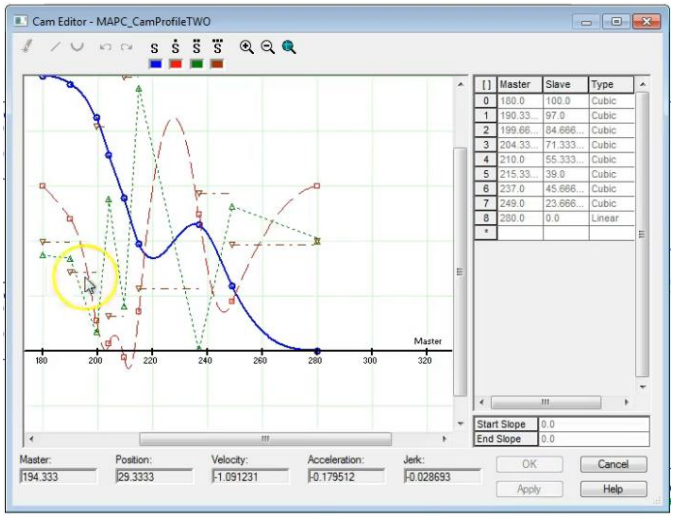

We will also have to add the SecondCam_Active latch to rung 4.

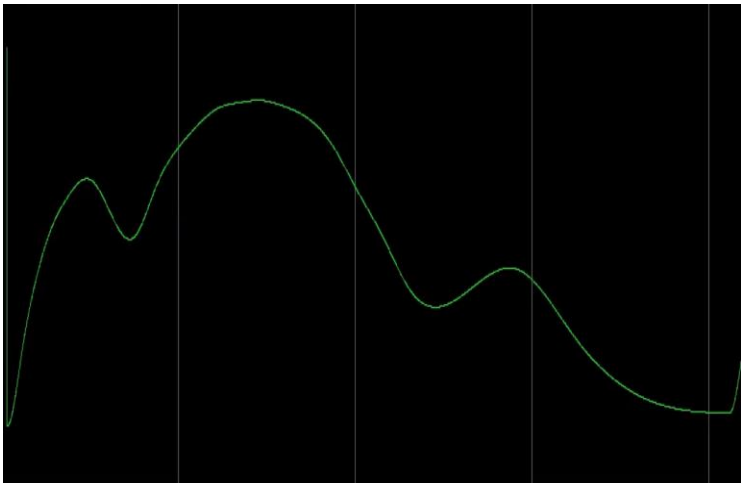


## 44. Showing a More In-Depth View of the Cams Blended with Added Features

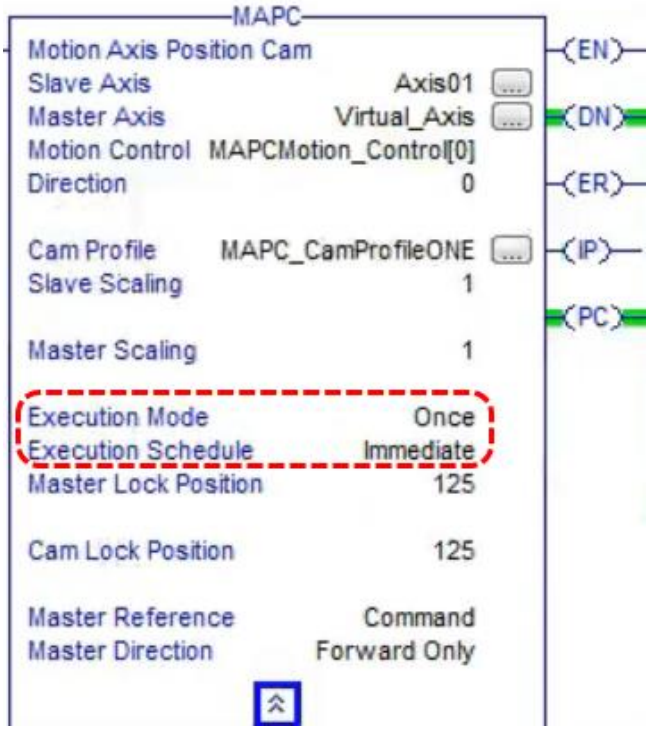
"Blending" seems to refer to moving back and forth between two states each with its own MAPC function block. The key is to wait until one is done before starting the other, this is done with status bits, see above. So you can have two (or more?) axis running back to back presumably against the same virtual master. The transition between the two has to be seamless. Now we will run the below two cams and inspect the trend. Blending time cams works the same way.

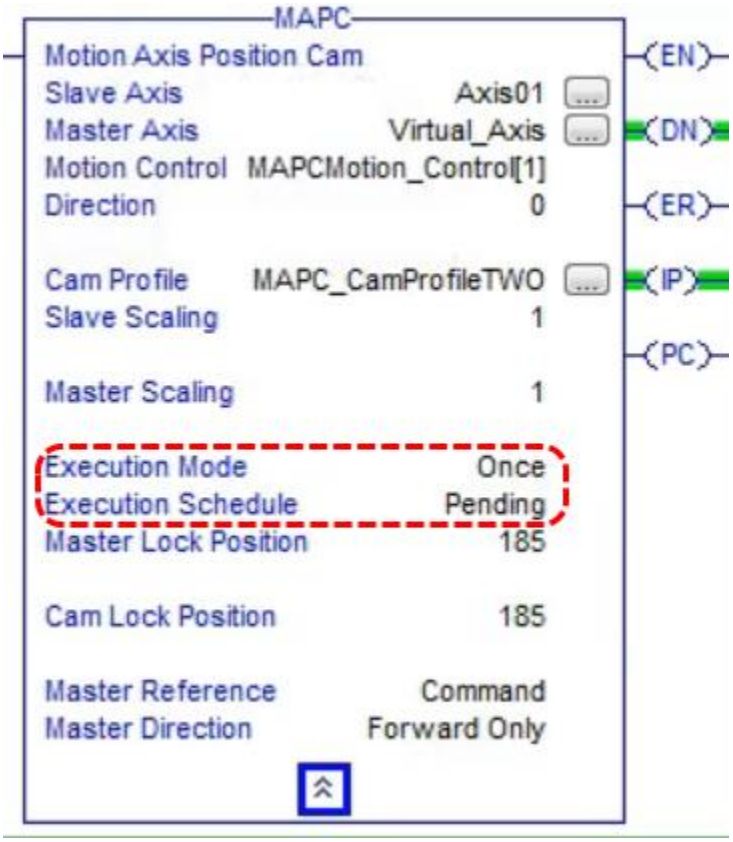


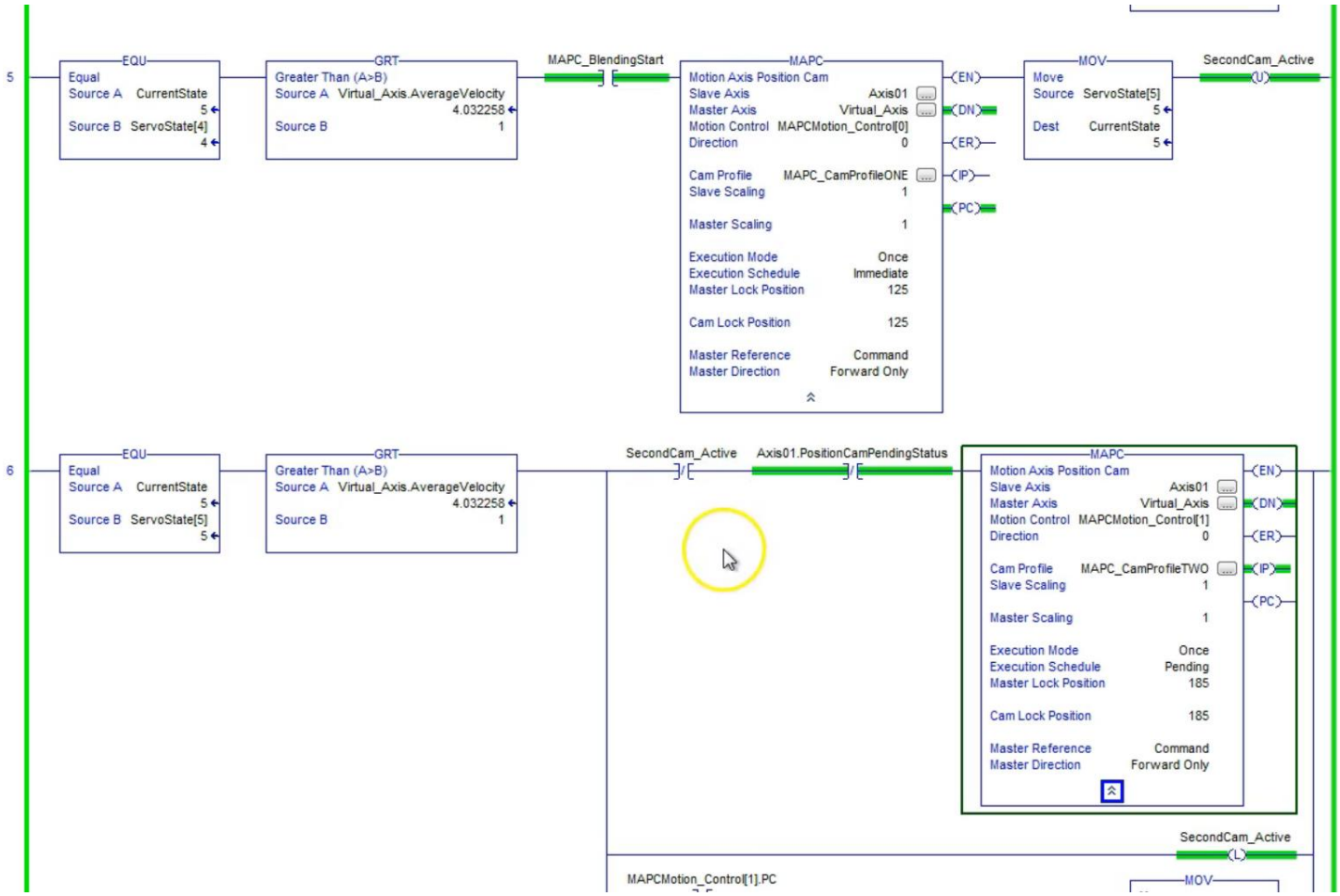
Looks good.
Question is, how does this happen? Look at the first MAPC function block, it is set for
Execution Mode: Once and
Execution Schedule: Immediate.





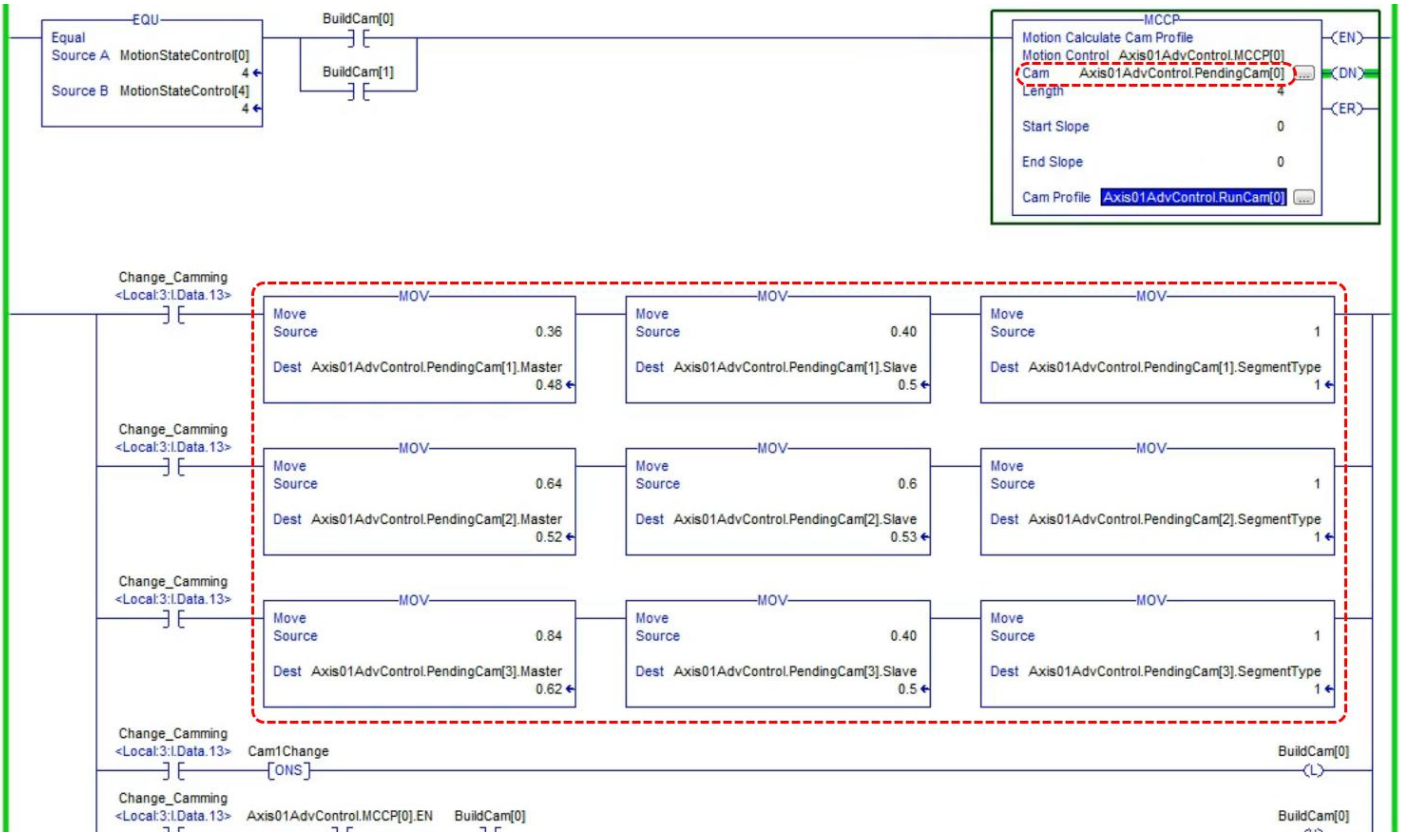

Now the second MAPC instruction is set up for…
Execution Mode: Once and
Execution Schedule: Pending.

Important note: this only works because of the SecondCam_Active discrete bit "blocking" MAPC 2 until MAPC 1 is compete. Otherwise MAPC 2 would run continuously even with execution schedule set to pending.

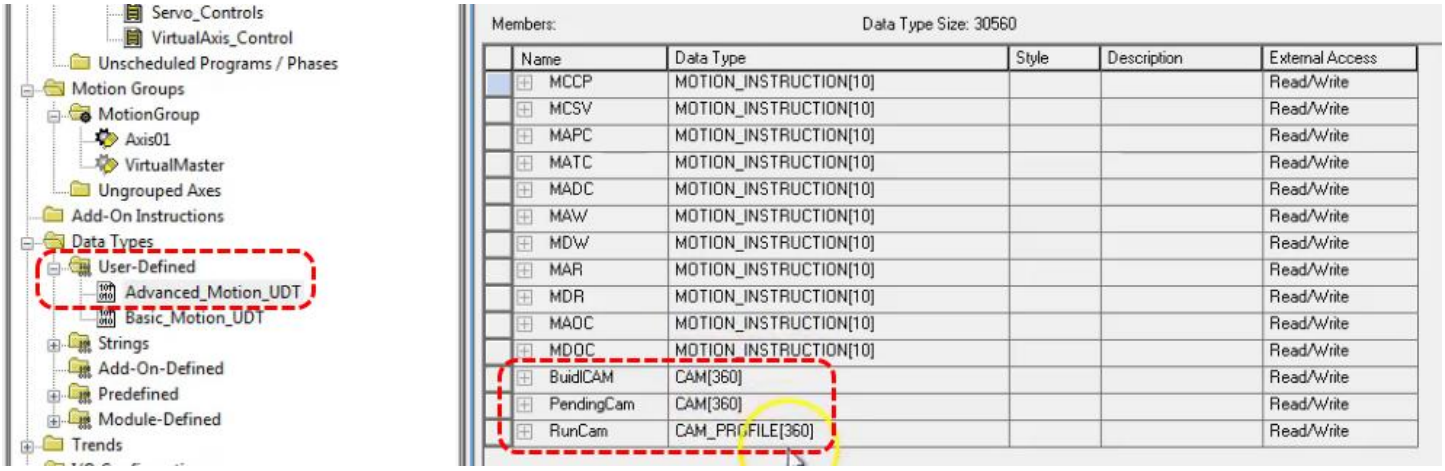


## 45. Using the MCCP Instruction

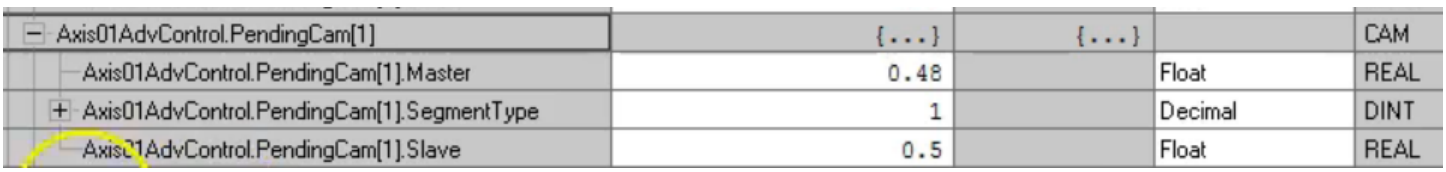
Note that a CAM and a CAM Profile are not the same thing. The CAM Profile collects the data from the CAM and runs it. You can make changes to the CAM array but they will not be loaded into the CAM Profile until you fire the MCCP instruction. Also be careful of the Length field of the MCCP instruction, you can build a deeper cam. This section talks about loading new values into the CAM Profile during run time using the MOV command.




We've added to the UDT.



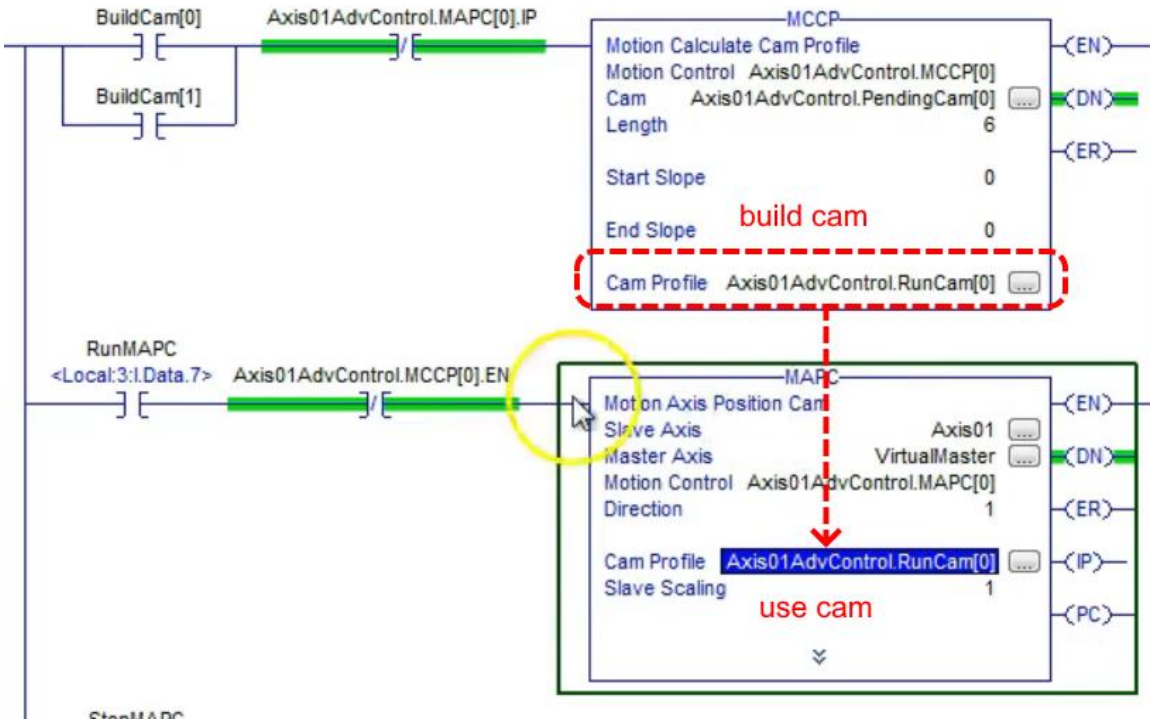

Now, what are the elements which make up the cam? We are not talking about the Cam Profile because we do not build the cam profile, we build the cam and load the cam profile.



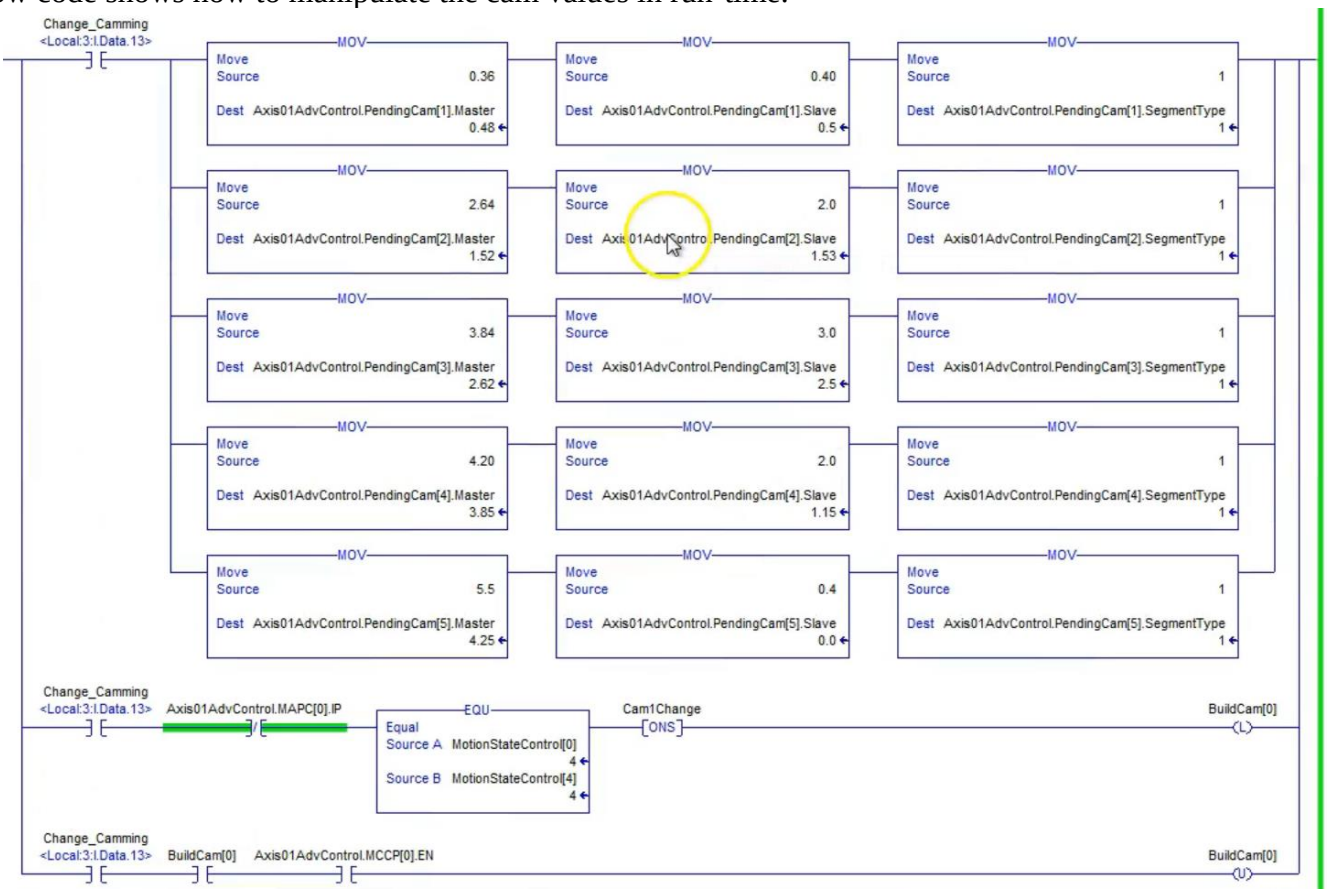

Segment Type is the Linear or Cubic entry. Master and Slave are position entries.

# 47. Using a MCCP and then Loading it into an MAPC Instruction For Use

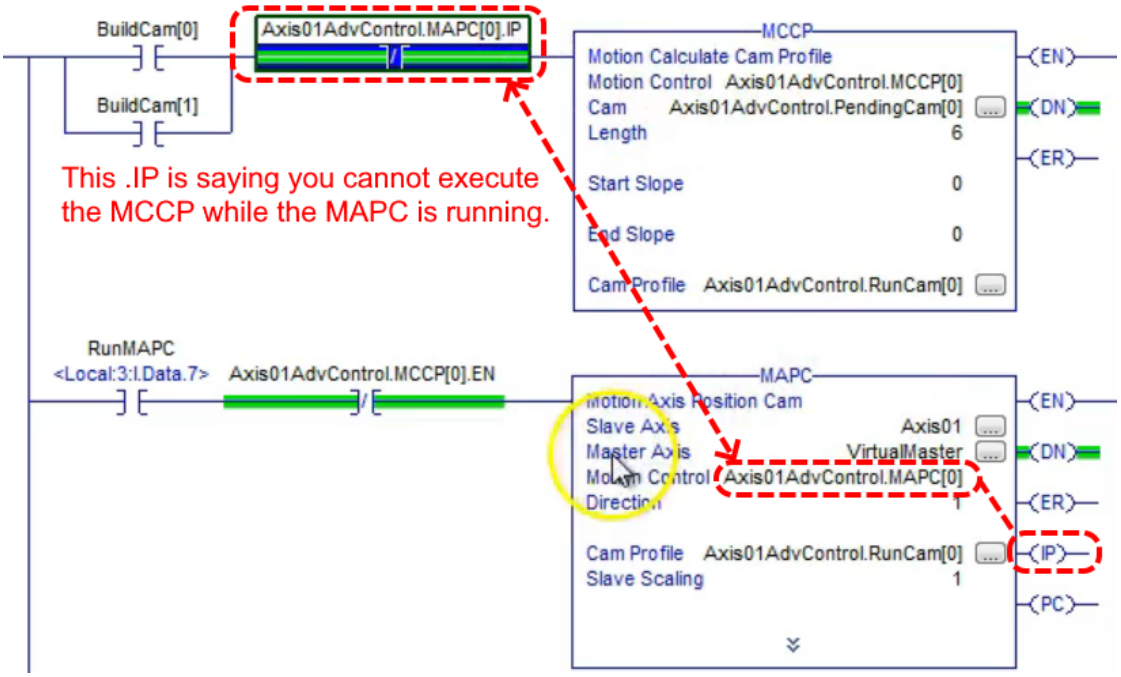
The idea of creating a CAM is to load it into a MATC or a MAPC. So how do we do that? We built a cam with MCCP, now how do we use that cam with the MCCP Cam Profile? The final CAM Profile we build we want to use in either a MATC or MAPC. Note that the axis cannot be running when you change the cam since you would be changing the dynamics of the servo.

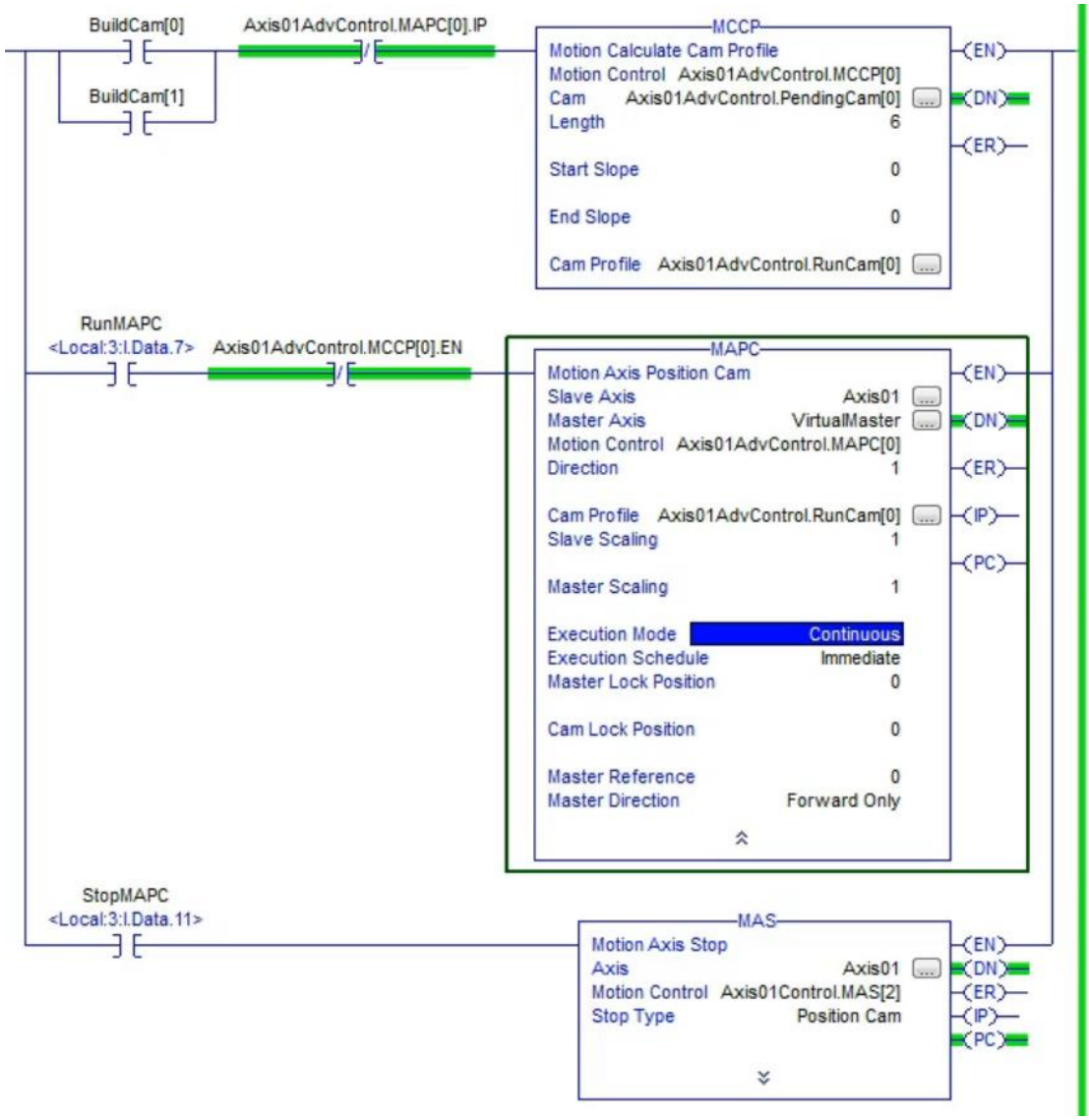



Below code shows how to manipulate the cam values in run-time.



The above cam value manipulation code can be repeated for further configurations.

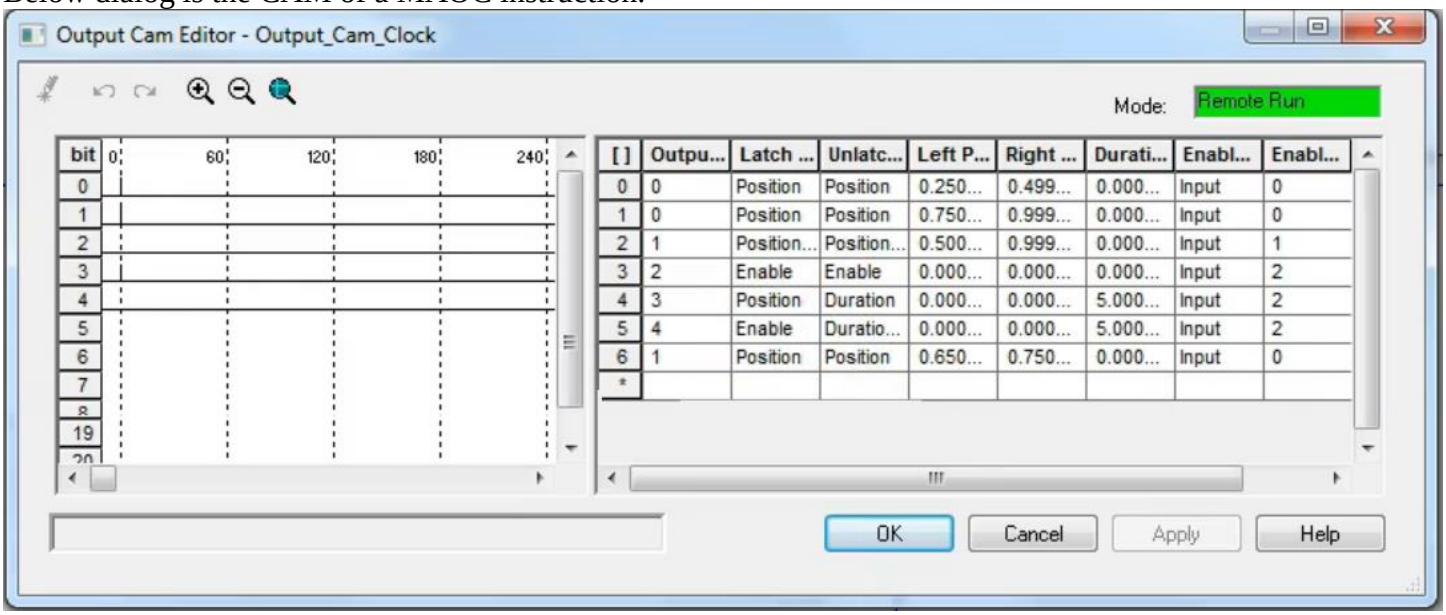


# 48. MAOC Introduction

MAOC is Motion Arm Output Cam. It is used in conjunction with motion to control output such as bit or IO you want to fire when the servo reaches a certain position.

Below dialog is the CAM of a MAOC instruction.

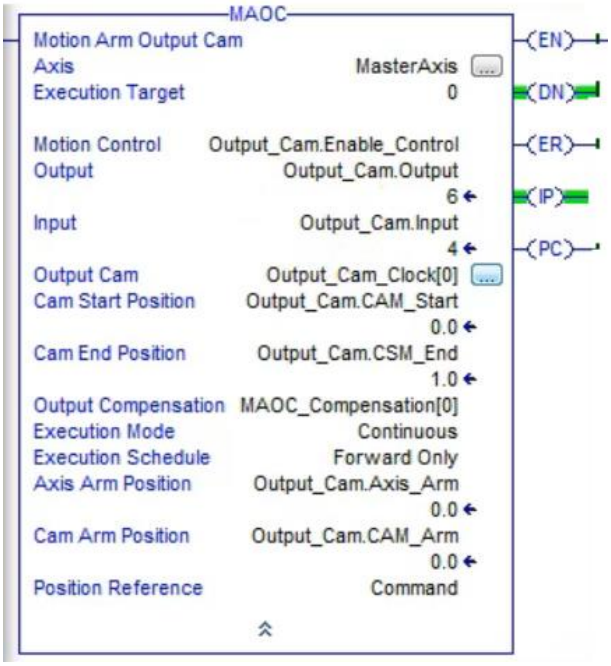


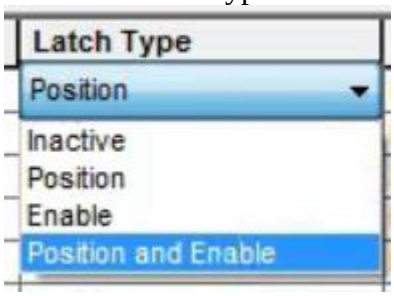
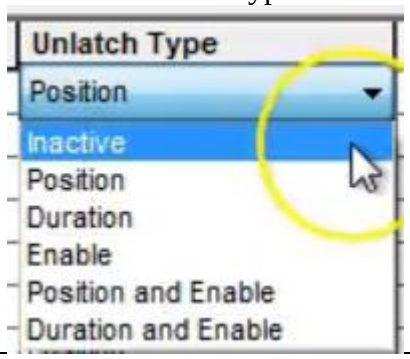

| [ ] | Outpu... | Latch ... | Unlatc... | Left P... | Right ... | Durati... | Enabl... | Enabl... |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Position | Position | 0.250... | 0.499... | 0.000... | Input | 0 |
| 1 | 0 | Position | Position | 0.750... | 0.999... | 0.000... | Input | 0 |
| 2 | 1 | Position... | Position... | 0.500... | 0.999... | 0.000... | Input | 1 |
| 3 | 2 | Enable | Enable | 0.000... | 0.000... | 0.000... | Input | 2 |
| 4 | 3 | Position | Duration | 0.000... | 0.000... | 5.000... | Input | 2 |
| 5 | 4 | Enable | Duratio... | 0.000... | 0.000... | 5.000... | Input | 2 |
| 6 | 1 | Position | Position | 0.650... | 0.750... | 0.000... | Input | 0 |
| * | | | | | | | | |

Below are the field headings.

| [ ] | Output Bit | Latch Type | Unlatch Type | Left Position | Right Position | Duration | Enable Type | Enable Bit |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Position | Position | 0.250000 | 0.499999 | 0.000000 | Input | 0 |
| 1 | 0 | Position | Position | 0.750000 | 0.999999 | 0.000000 | Input | 0 |
| 2 | 1 | Position and Enable | Position and Enable | 0.500000 | 0.999999 | 0.000000 | Input | 1 |
| 3 | 2 | Enable | Enable | 0.000000 | 0.000000 | 0.000000 | Input | 2 |
| 4 | 3 | Position | Duration | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 5 | 4 | Enable | Duration and Enable | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 6 | 1 | Position | Position | 0.650000 | 0.750000 | 0.000000 | Input | 0 |
| * | | | | | | | | |

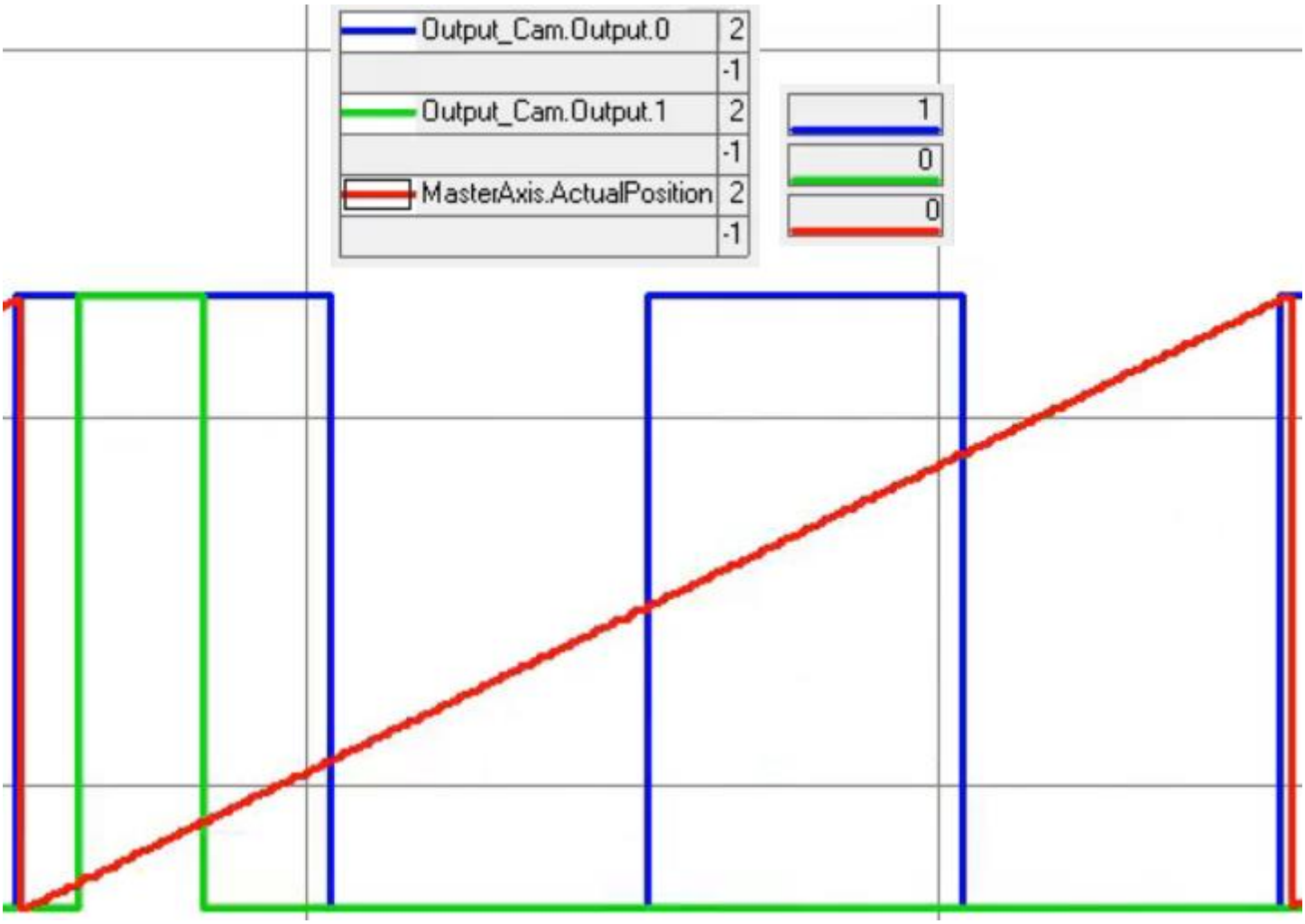
First column, **Output Bit**. Outputs associated with the instruction, note the Output, Input, and Output Cam fields.
**Latch Type**: Position & Enable, Inactive, Position, and Position & Enable.
**Latch Type Position** is associated with **Left Position**.
**Unlatch Type Position** is associated with **Right Position**.
If **LatchType** is set to Inactive the output bit is left unchanged.
Note output bits are reset when the enable bit becomes inactive.
Position and Enable means the output is reset when the axis position leaves the compensated range (cam range) and the enable bit becomes inactive (?).
For **Unlatch Type Duration and Enable** the output bit is reset when the duration expires and the enable bit becomes inactive.
Duration means you want to keep the bit on for a set time.
Duration and Enable means both are active. It sounds like "enabled" means if the other output bit in the cam is "on" the action of the bit you are configuring will take place.
More on this below.



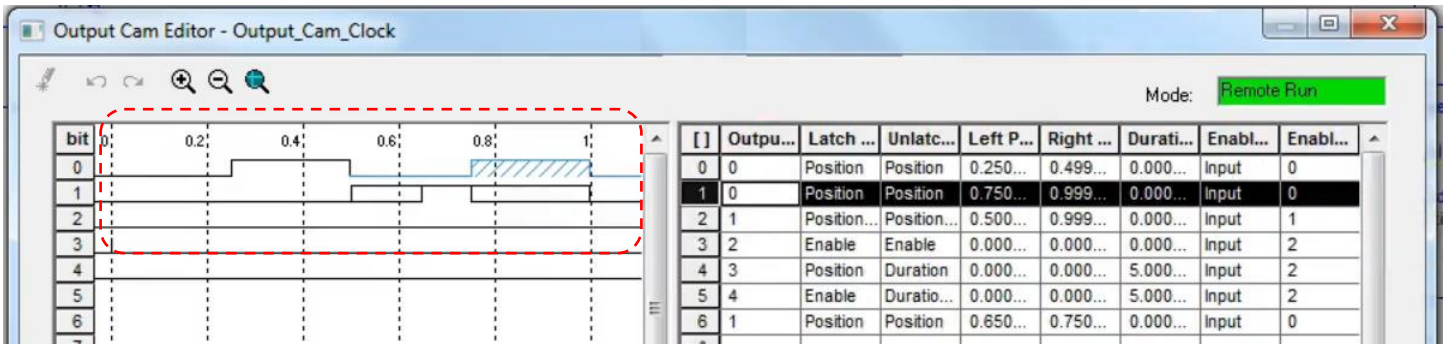
| Latch Type | Unlatch Type | | |
|---|---|---|---|
|  |  | | |







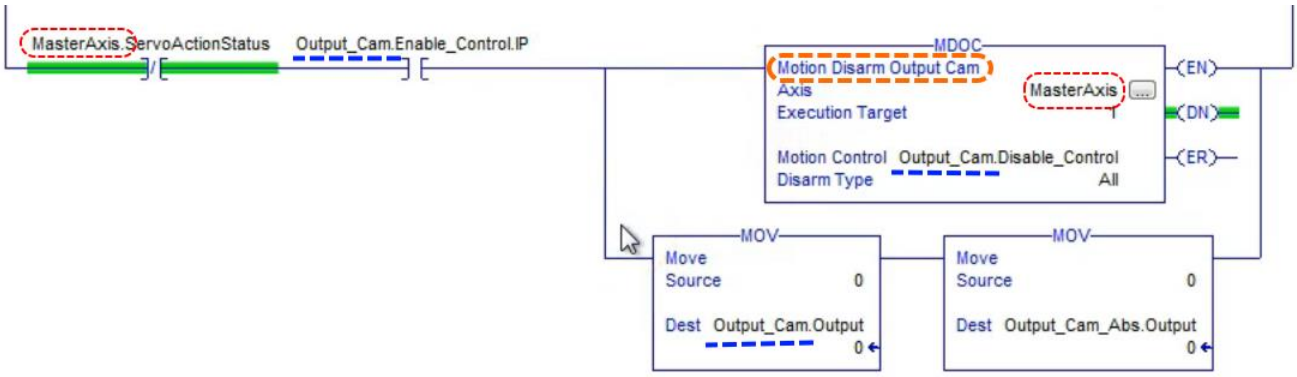

Here red is the master cam which the output cam follows. Remember, on the trend the vertical axis is position of the master cam. Blue is the output bit, from the table we see it should be coming on at .25, off at .5, on again at .75, and off at 1. It does seem to be following that pattern. Now output bit 1; it should come on at .5 and off at 1, and it has another line that says come on at .65 and off at .75. Not seeing this, may have to do with enable bits?

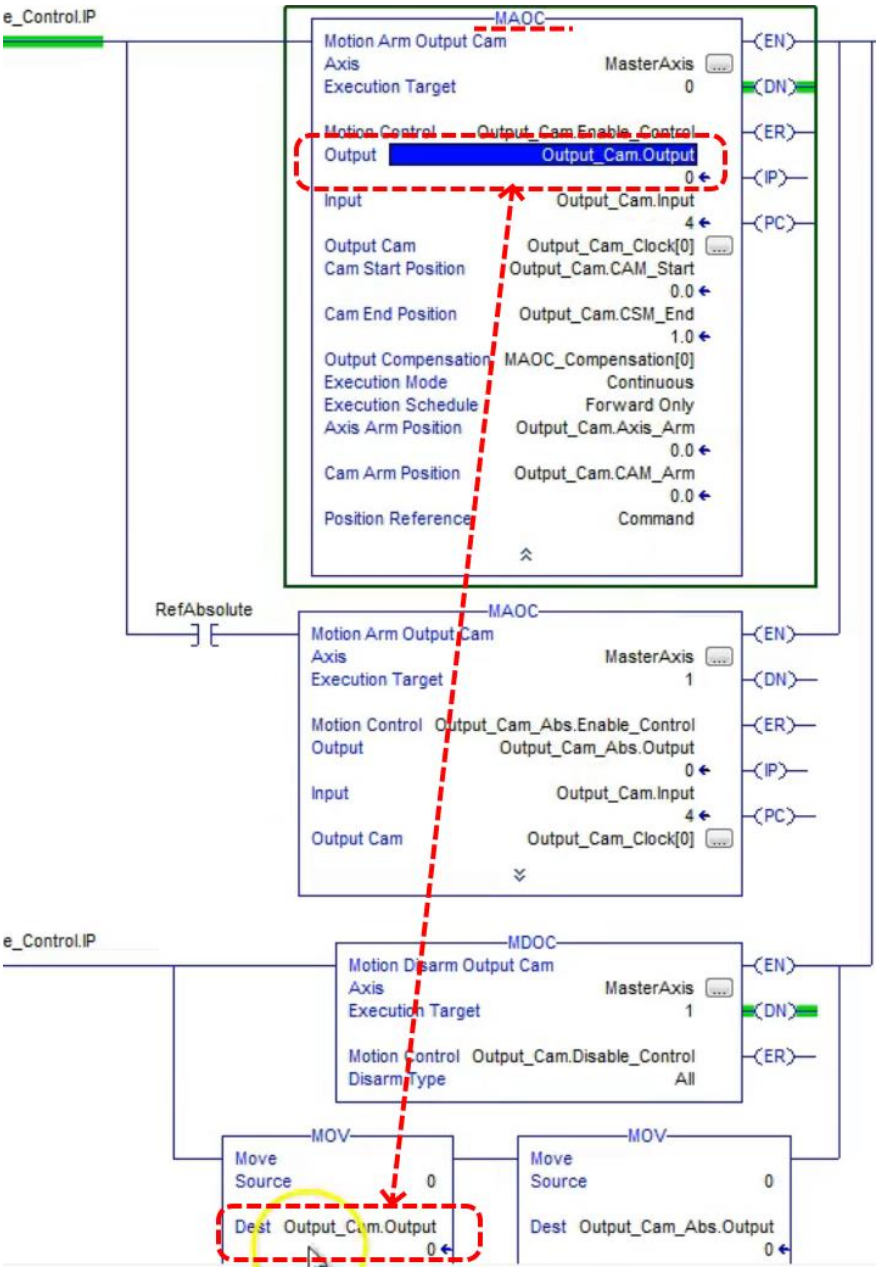



Looking at the traces on the Output Cam we see everything seems correct.

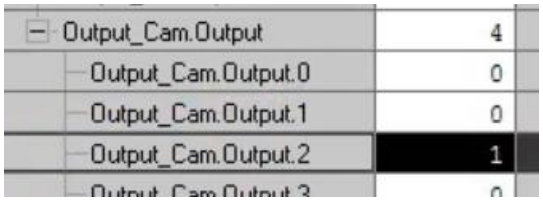
## 50. MAOC Cam Output Control

Regarding outputs, if they are aliased to an output card and are active, meaning being driven by the cam, and the program is stopped (axis is stopped) suddenly (abnormally) the output remains high (or whatever the driven state is). Note that during this time the.IP bit will be on. So it is necessary to write code to put the outputs in the desired safe state.

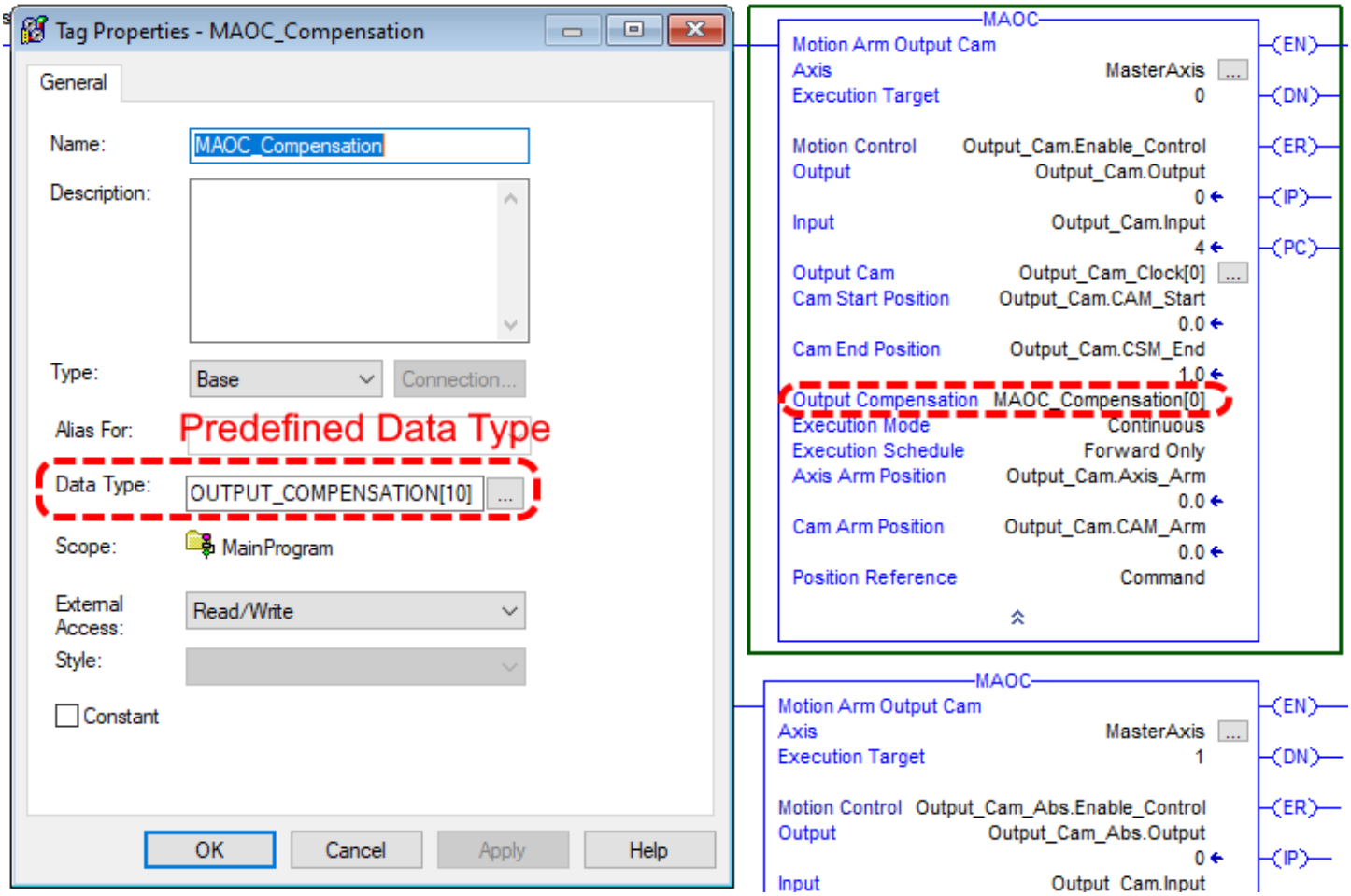


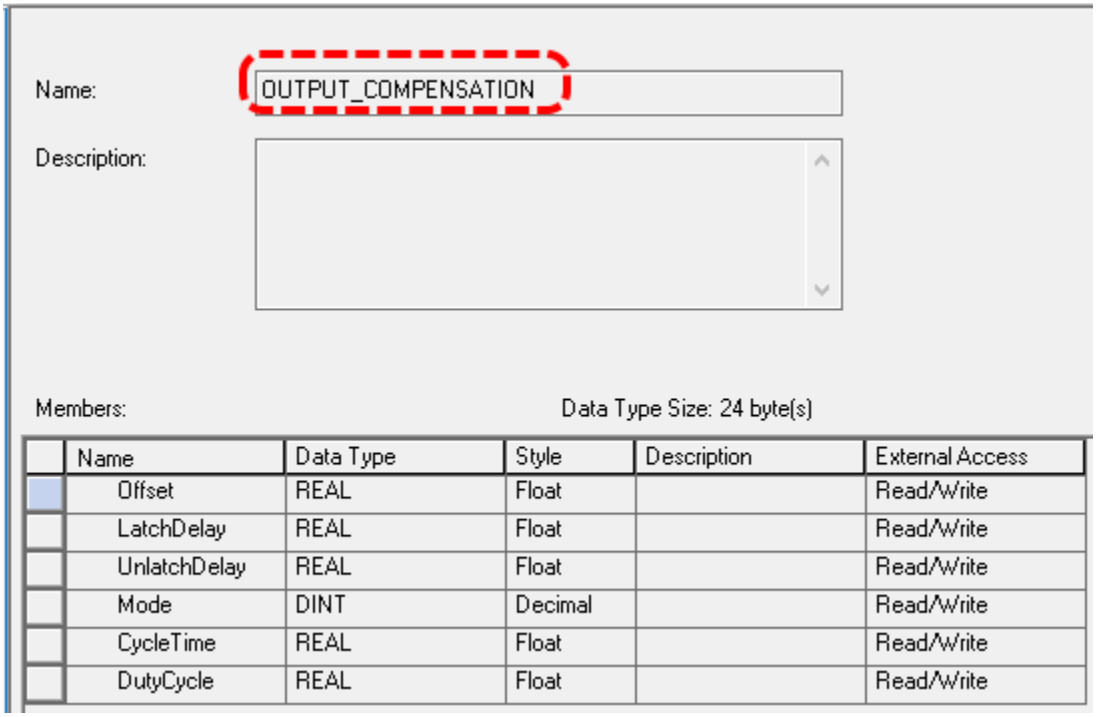

Writing the zero to the MAOC Output parameter sets the output bits to zero.



# 51. MAOC Compensation Usage Explained




When you create the compensation tag, keep in mind the type is in the predefined group of UDTs. Compensation data type can be declared in the program tag scope.

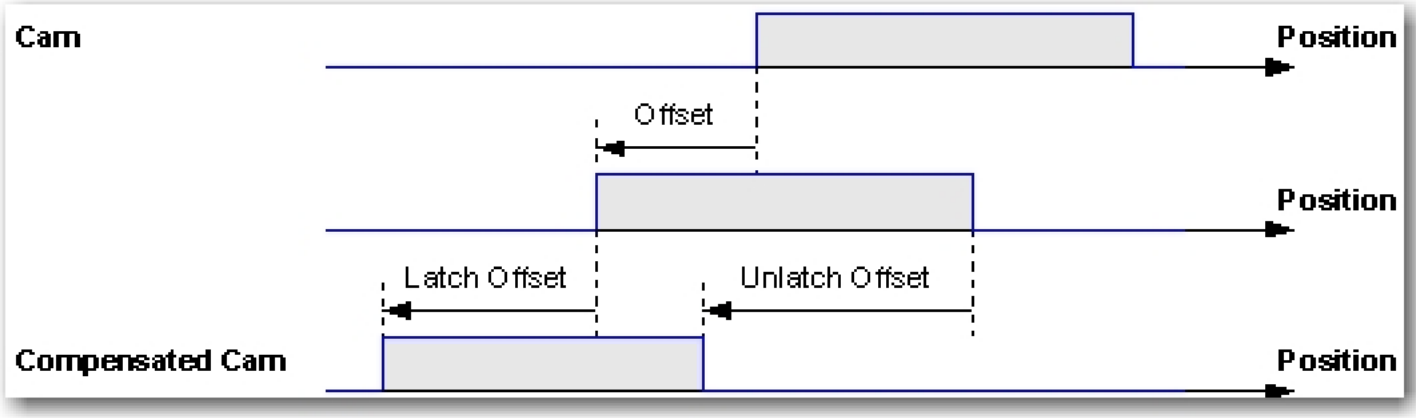



| Name | Data Type | Style | Description | External Access |
|---|---|---|---|---|
| Offset | REAL | Float | | Read/Write |
| LatchDelay | REAL | Float | | Read/Write |
| UnlatchDelay | REAL | Float | | Read/Write |
| Mode | DINT | Decimal | | Read/Write |
| CycleTime | REAL | Float | | Read/Write |
| DutyCycle | REAL | Float | | Read/Write |

| | | | | |
|---|---|---|---|---|
| MAOC_Compensation | {...} | {...} | | OUTPUT_COMP... |
| MAOC_Compensation[0] | {...} | {...} | | OUTPUT_COMP... |
| MAOC_Compensation[0].Offset | 0.0 | | Float | REAL |
| MAOC_Compensation[0].LatchDelay | 0.5 | | Float | REAL |
| MAOC_Compensation[0].UnlatchDelay | 0.5 | | Float | REAL |
| MAOC_Compensation[0].Mode | 0 | | Decimal | DINT |
| MAOC_Compensation[0].CycleTime | 0.0 | | Float | REAL |
| MAOC_Compensation[0].DutyCycle | 0.0 | | Float | REAL |

Output Compensation: array indices correspond to output bit numbers, the minimum size of an array is determined by the highest compensated output bit. Each bit has the 6 settings below:

| | |
|---|---|
| CycleTime: | Refers to pulse rising edge in Inverted & Pulsed compensation (see diagram) |
| DutyTime: | = (On-Duty Time)/(Cycle Time), on-duty time + pulse width |
| LatchDelay: | provides time delay compensation for the latch operation |
| Mode: | normal, inverted , pulsed, inverted & pulsed |
| Offset: | provides position compensation |
| UnLatchDelay: | provides time delay compensation for the unlatch operation (inoperative) |



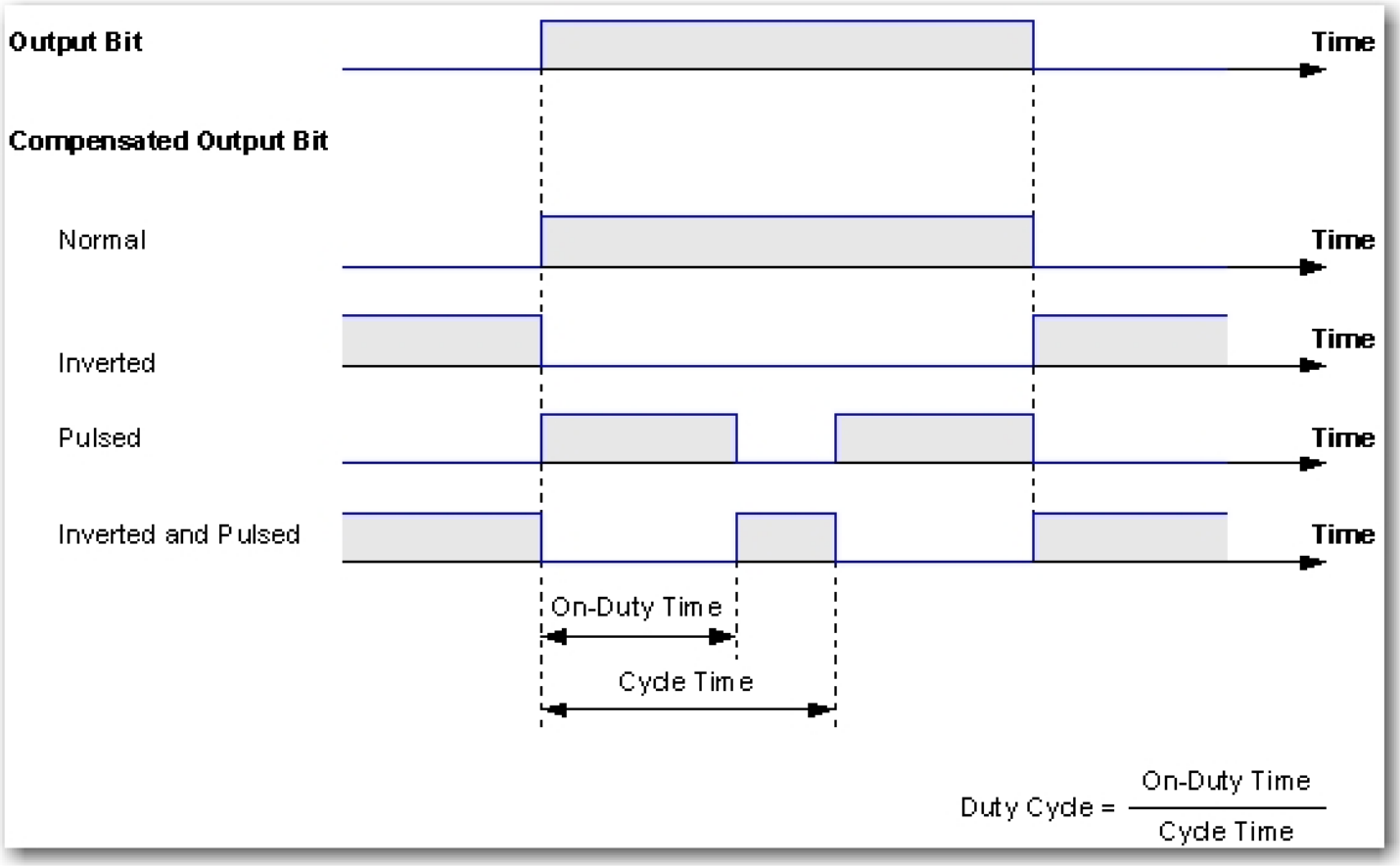

The cam range is defined by the left and right cam positions of the Output Cam element. The compensated cam range is defined by the cam range, offset, and latch and unlatch offsets. The latch and unlatch offsets are defined by the current speed v.

**Latch Offset = v * Latch Delay** **Unlatch Offset = v * Unlatch Delay**

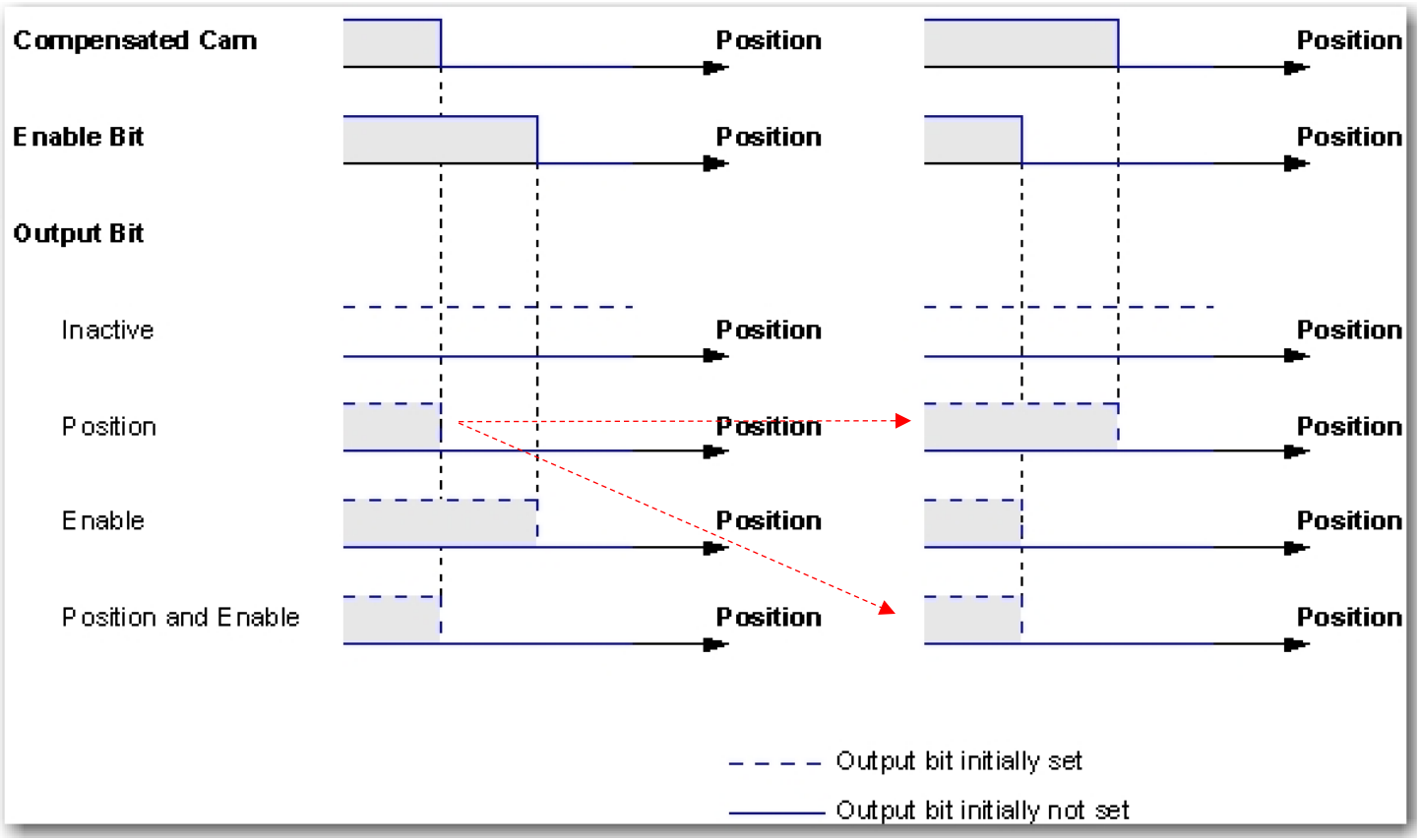
The resulting compensation offset can actually be larger than the difference between cam start and cam end position. The following equation illustrates the effect of the compensation values on the duration of an Output Cam element: **Compensated Duration = Duration + Latch Delay - Unlatch Delay**

| | |
|---|---|
| **Normal** | The output bit is set, when the output of the latch and unlatch operation becomes active. The output bit is reset, when the output of the latch and unlatch operation becomes inactive. |
| **Inverted** | The output bit is set, when the output of the latch and unlatch operation becomes inactive. The output bit is reset, when the output of the latch and unlatch operation becomes active. |
| **Pulsed** | The output bit is pulsed, when the output of the latch and unlatch operation is active. The on-duty state of the pulse corresponds to the active state of the output bit. The output bit is reset, when the output of the latch and unlatch operation becomes inactive. |
| **Inverted & Pulsed** | The output bit is pulsed, when the output of the latch and unlatch operation is active. The on-duty state of the pulse corresponds to the inactive state of the output bit. The output bit is set, when the output of the latch and unlatch operation becomes inactive. |

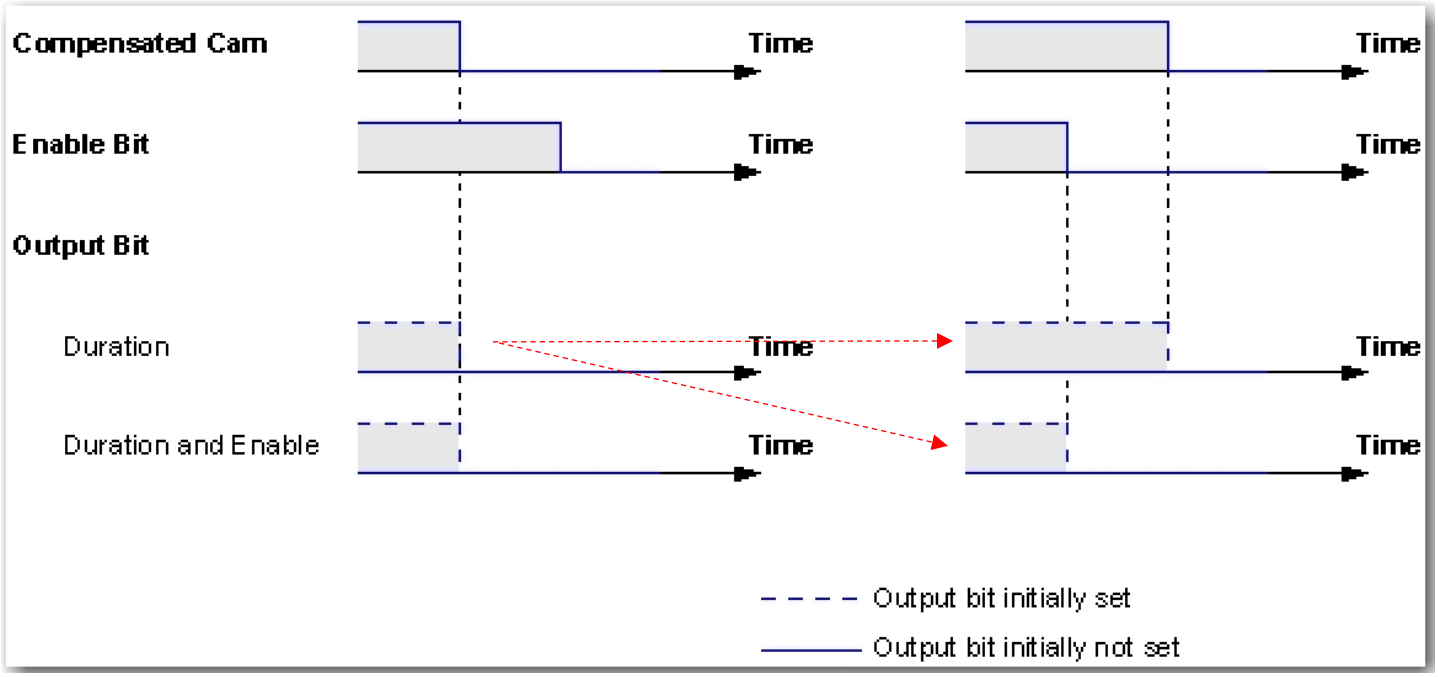
Below diagram illustrates the four modes described above.



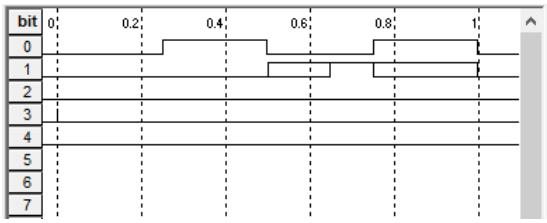

The following diagram shows the effect of the selected unlatch type on the output bit for different compensated cam and enable bit combinations as function of **position**.



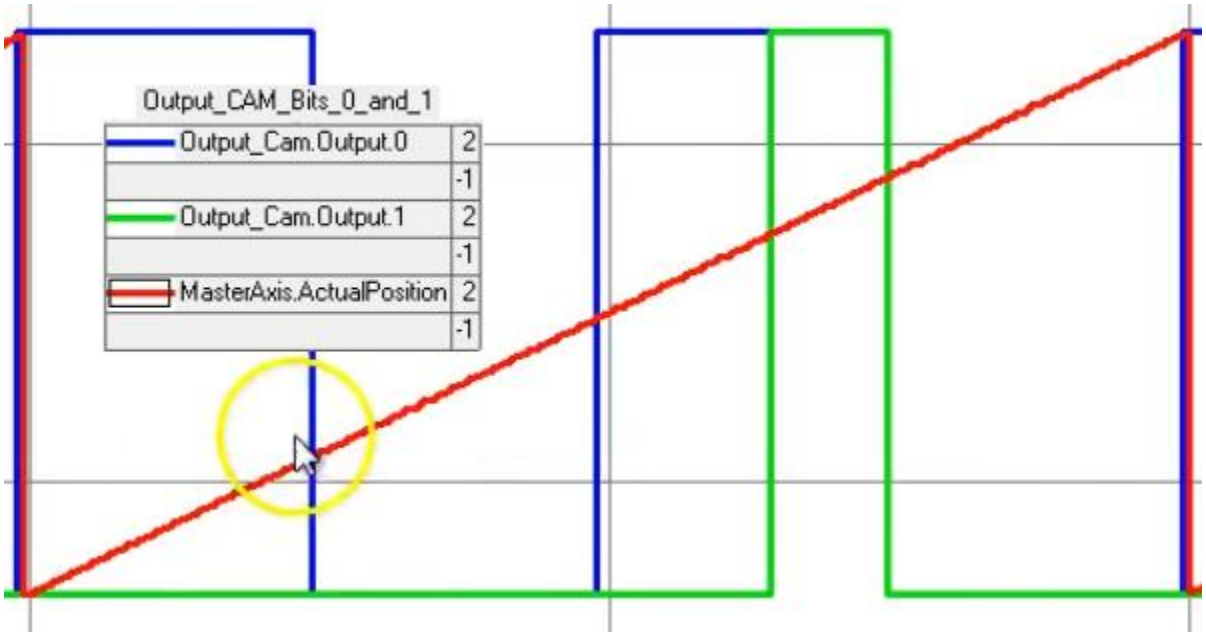
The following diagram shows the effect of the selected unlatch type on the output bit for different compensated cam and enable bit combinations as function of **time**.




The cam table from the program (sample below) will not necessarily show the effect of the above settings.

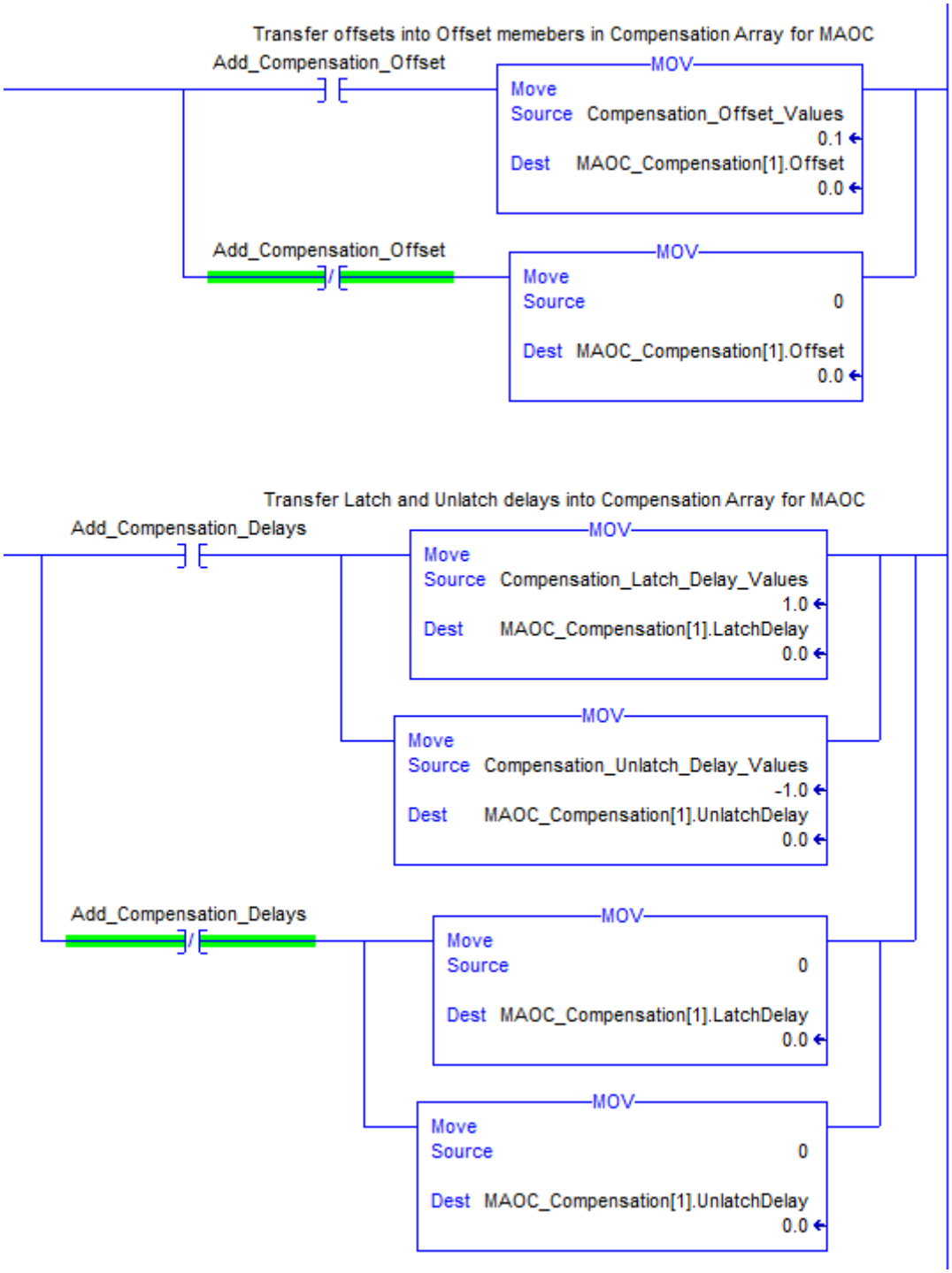


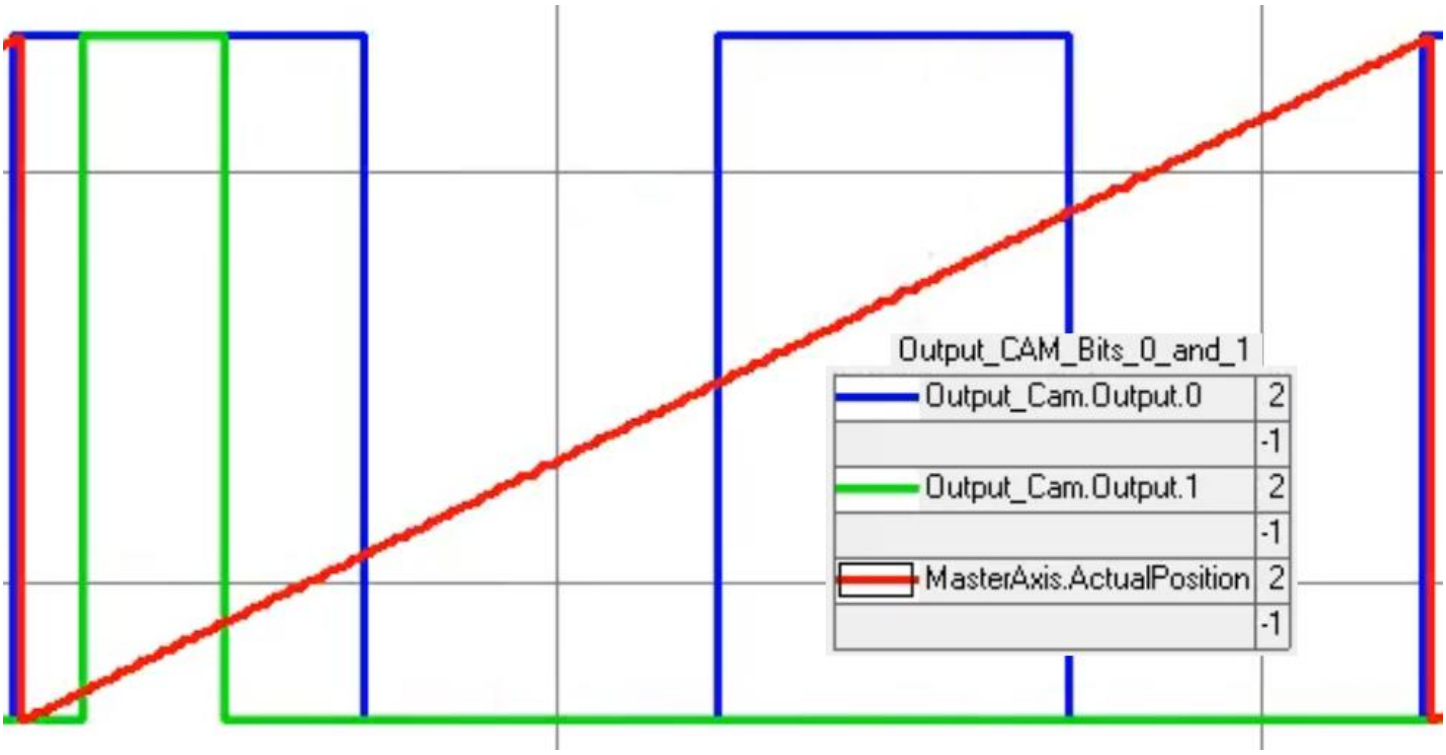

| [ ] | Outpu... | Latch Type | Unlatch Type | Left Position | Right Position | Duration | Enable Type | Enable Bit |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Position | Position | 0.250000 | 0.499999 | 0.000000 | Input | 0 |
| 1 | 0 | Position | Position | 0.750000 | 0.999999 | 0.000000 | Input | 0 |
| 2 | 1 | Position and E... | Position and En... | 0.500000 | 0.999999 | 0.000000 | Input | 1 |
| 3 | 2 | Enable | Enable | 0.000000 | 0.000000 | 0.000000 | Input | 2 |
| 4 | 3 | Position | Duration | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 5 | 4 | Enable | Duration and E... | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 6 | 1 | Position | Position | 0.650000 | 0.750000 | 0.000000 | Input | 0 |
| * | | | | | | | | |




Bit firing for above cam with no compensation. With compensation on bit 1 would be firing very close to the Master Axis initial position (position 1 is from .5 to almost the rollover).

Now we will add compensation and delays via the code shown below (MAOC Slot0.ADC).

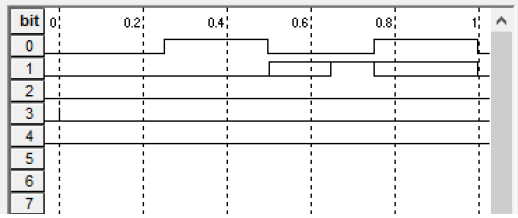

Transfer offsets into Offset memebers in Compensation Array for MAOC
Add_Compensation_Offset
MOV
Move
Source Compensation_Offset_Values
0.1
Dest MAOC_Compensation[1].Offset
0.0
Add_Compensation_Offset
MOV
Move
Source 0
Dest MAOC_Compensation[1].Offset
0.0
Transfer Latch and Unlatch delays into Compensation Array for MAOC
Add_Compensation_Delays
MOV
Move
Source Compensation_Latch_Delay_Values
1.0
Dest MAOC_Compensation[1].LatchDelay
0.0
MOV
Move
Source Compensation_Unlatch_Delay_Values
-1.0
Dest MAOC_Compensation[1].UnlatchDelay
0.0
Add_Compensation_Delays
MOV
Move
Source 0
Dest MAOC_Compensation[1].LatchDelay
0.0
MOV
Move
Source 0
Dest MAOC_Compensation[1].UnlatchDelay
0.0




Now, with compensation enabled, output 1 is firing very close to the beginning of cam 1. Here we have modified the compensation and delay values of the cam.

## 52. MAOC Input Enable Feature (Input Cam)

In this example our input cam is a parameter called Output_Cam.Input. Say we set our top three output cam rows to Position & Enable.




| [ ] | Output Bit | Latch Type | Unlatch Type | Left Position | Right Position | Duration | Enable Type | Enable Bit |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Position | Position and Enable | 0.250000 | 0.499999 | 0.000000 | Input | 0 |
| 1 | 0 | Position | Position and Enable | 0.750000 | 0.999999 | 0.000000 | Input | 0 |
| 2 | 1 | Position and ... | Position and Enable | 0.500000 | 0.999999 | 0.000000 | Input | 1 |
| 3 | 2 | Enable | Enable | 0.000000 | 0.000000 | 0.000000 | Input | 2 |
| 4 | 3 | Position | Duration | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 5 | 4 | Enable | Duration and Enable | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 6 | 1 | Position | Position | 0.650000 | 0.750000 | 0.000000 | Input | 0 |
| * | | | | | | | | |

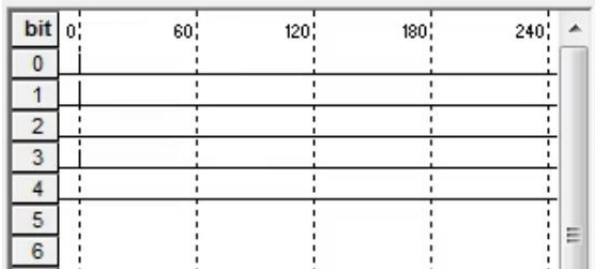
Now we set the first three Output_Cam.Input values to 1, this will enable those bits.

| | |
|---|---|
| Output_Cam.Input | 31 |
| Output_Cam.Input.0 | 1 |
| Output_Cam.Input.1 | 1 |
| Output_Cam.Input.2 | 1 |
| Output_Cam.Input.3 | 1 |
| Output_Cam.Input.4 | 1 |

Now we run and check the Output_Cam.Output bits…

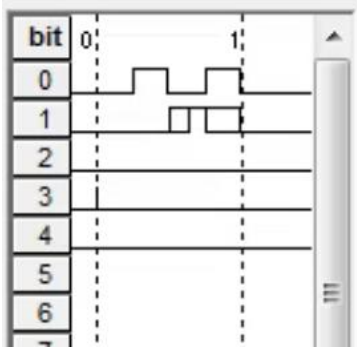
| | |
|---|---|
| Output_Cam.Output | 13 |
| Output_Cam.Output.0 | 1 |
| Output_Cam.Output.1 | 0 |
| Output_Cam.Output.2 | 1 |
| Output_Cam.Output.3 | 1 |
| Output_Cam.Output.4 | 0 |

We find that bits 0, 1 and 2 are cycling. We can try to discern the pattern. Does it look like the cam profile? It appears that only bits 0 and 1 have a cam defined. Now we will change all Enable bits to "6".




| [ ] | Outpu... | Latch ... | Unlatc... | Left P... | Right ... | Durati... | Enabl... | Enabl... |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Position... | Position... | 0.250... | 0.499... | 0.000... | Input | 6 |
| 1 | 0 | Position... | Position... | 0.750... | 0.999... | 0.000... | Input | 6 |
| 2 | 1 | Position... | Position... | 0.500... | 0.999... | 0.000... | Input | 6 |
| 3 | 2 | Enable | Enable | 0.000... | 0.000... | 0.000... | Input | 6 |
| 4 | 3 | Position | Duration | 0.000... | 0.000... | 5.000... | Input | 6 |
| 5 | 4 | Enable | Duratio... | 0.000... | 0.000... | 5.000... | Input | 6 |
| 6 | 1 | Position | Position | 0.650... | 0.750... | 0.000... | Input | 6 |

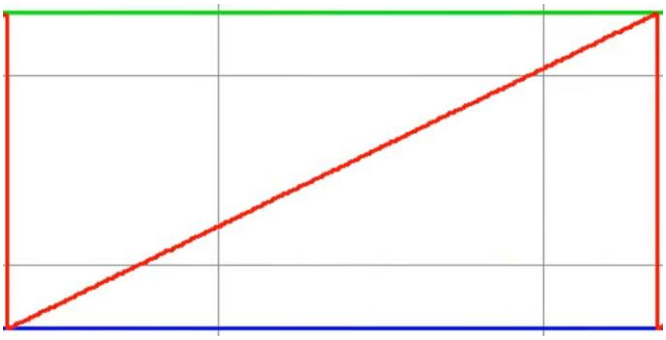
Now restart the servo but you must also restart/recycle the cam profile. We are interested in the bits set to Position and Enable for Latch/Unlatch.




| [ ] | Outpu... | Latch Type | Unlatch Type |
|---|---|---|---|
| 0 | 0 | Position and Enable | Position and Enable |
| 1 | 0 | Position and Enable | Position and Enable |
| 2 | 1 | Position and Enable | Position and Enable |
| 3 | 2 | Enable | Enable |
| 4 | 3 | Position | Duration |
| 5 | 4 | Enable | Duration and Enable |
| 6 | 1 | Position and Enable | Position and Enable |

The way we have the system configured we have no enables on (all are at 6) so we should not see any outputs responding. The positions set to 1 in the enable column will respond. We can run and actually see this in the code, the Output_Cam.Output value will change as the bit numbers which are responding to the cam change.

When all enable values are set to 6 no bits respond and Output_Cam.Output displays 10. Consider the Input_Cam.Input array, it has position 6 and beyond set to zero.

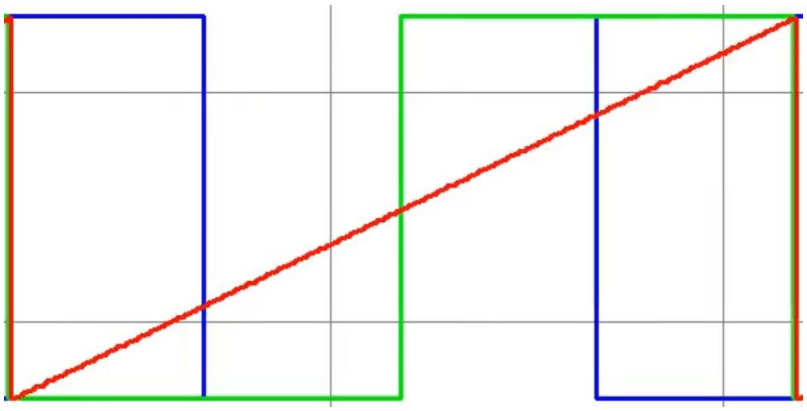


No outputs responding.

Now if we turn on Output_Cam.Input 6, set to 1, we will see different response.

| Output_Cam.Input | 95 |
|---|---|
| Output_Cam.Input.0 | 1 |
| Output_Cam.Input.1 | 1 |
| Output_Cam.Input.2 | 1 |
| Output_Cam.Input.3 | 1 |
| Output_Cam.Input.4 | 1 |
| Output_Cam.Input.5 | 0 |
| Output_Cam.Input.6 | 1 |

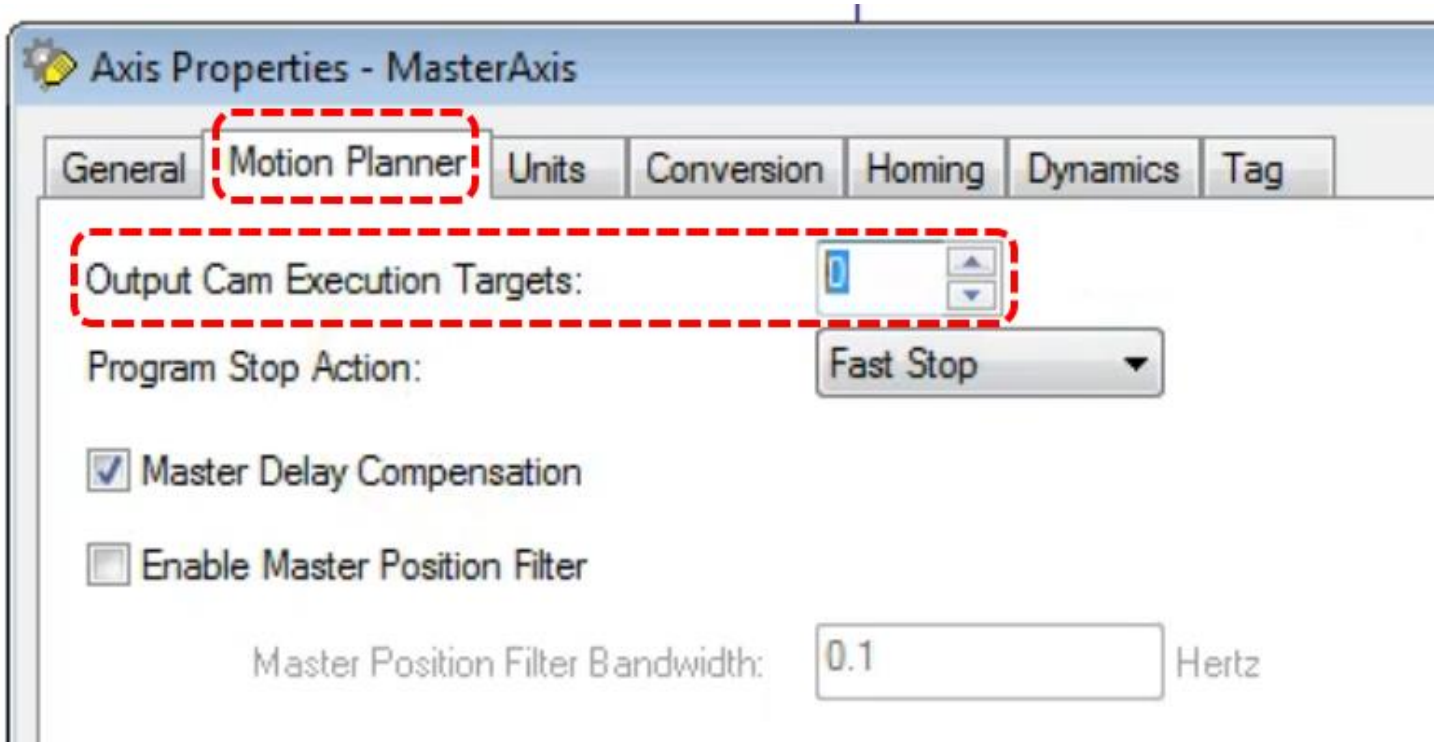
We see cam outputs firing. So the bit number in the Enable column tells the system which Output_Cam.Input bit to look and and use it as the enable signal (1 being enable). That's pretty much our Output_Cam.Input array.



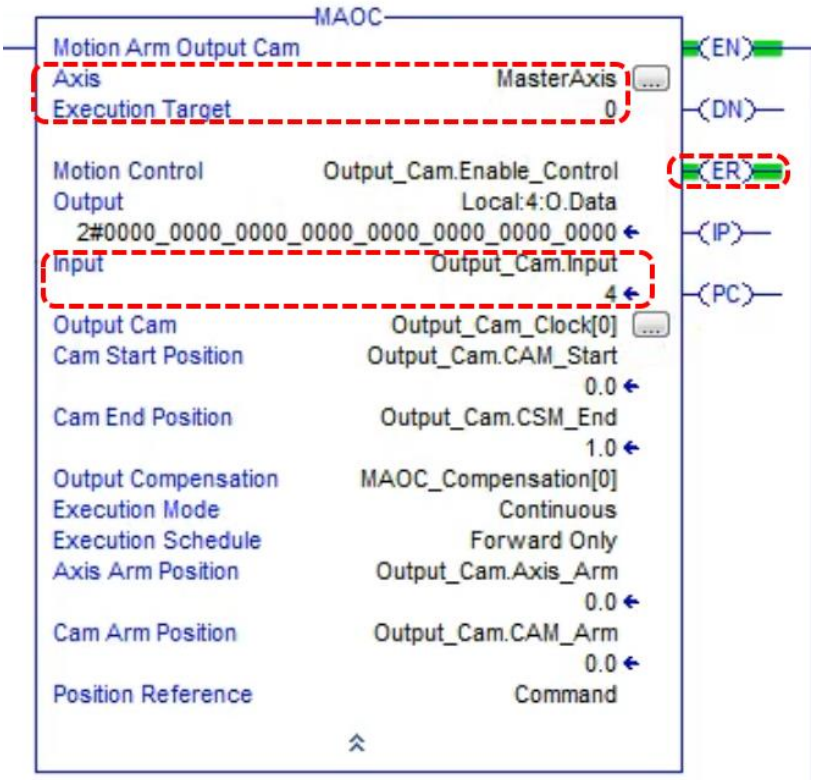
Now we ask, what if our Enable Type bit is set to Inverted Input? This means invert whatever the value is at Output_Cam.Input[#] and use that value as the enable signal. I believe these values can be changed at run time, in some cases the block enable has to be cycled but maybe not this case.

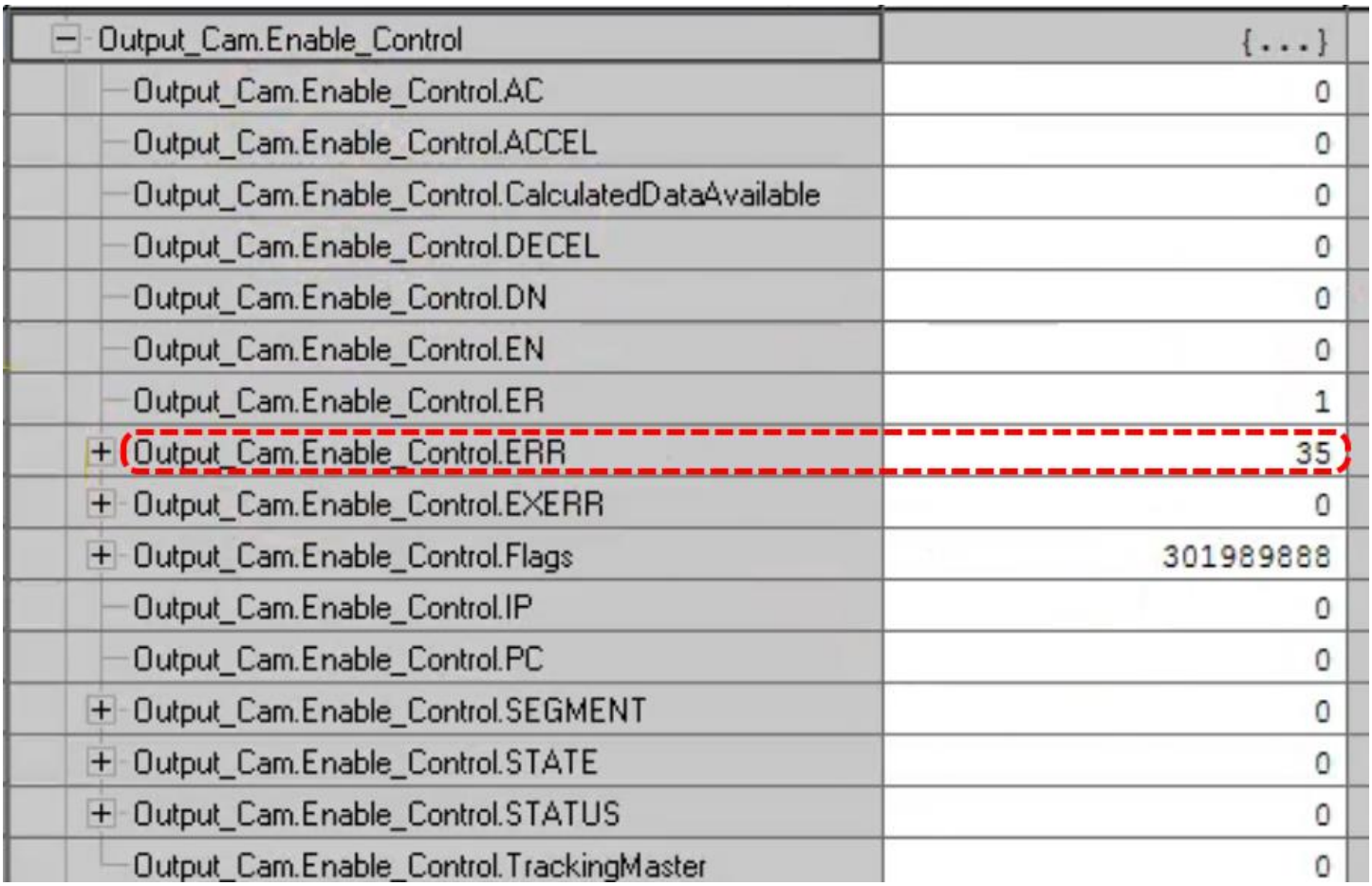
## 53. MAOC Axis Setup - Avoid an ERR

Consider the motion planner Output Cam Execution Targets setting in the virtual axis properties dialog for the "Axis" being used with the MAOC command.




If we leave the 0 here we will fail on an error when we run because the MAOC instruction is telling the Axis (in this case virtual) that it has output cams to execute and time must be set aside to do so. You must have a value here which is large enough to accommodate the output cam. This applies to real axis as well.





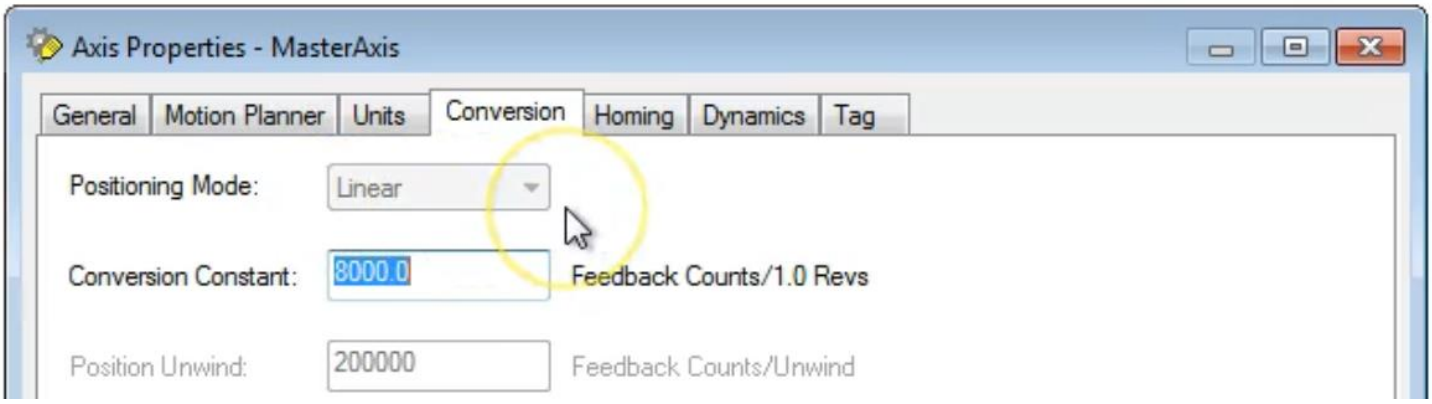
| Output_Cam.Enable_Control | {...} |
|---|---|
| Output_Cam.Enable_Control.AC | 0 |
| Output_Cam.Enable_Control.ACCEL | 0 |
| Output_Cam.Enable_Control.CalculatedDataAvailable | 0 |
| Output_Cam.Enable_Control.DECEL | 0 |
| Output_Cam.Enable_Control.DN | 0 |
| Output_Cam.Enable_Control.EN | 0 |
| Output_Cam.Enable_Control.ER | 1 |
| Output_Cam.Enable_Control.ERR | 35 |
| Output_Cam.Enable_Control.EXERR | 0 |
| Output_Cam.Enable_Control.Flags | 301989888 |
| Output_Cam.Enable_Control.IP | 0 |
| Output_Cam.Enable_Control.PC | 0 |
| Output_Cam.Enable_Control.SEGMENT | 0 |
| Output_Cam.Enable_Control.STATE | 0 |
| Output_Cam.Enable_Control.STATUS | 0 |
| Output_Cam.Enable_Control.TrackingMaster | 0 |

| | | |
|---|---|---|
| 35 | The specified execution target exceeds the number of Output Cam targets configured for the axis. | Illegal Execution Target |

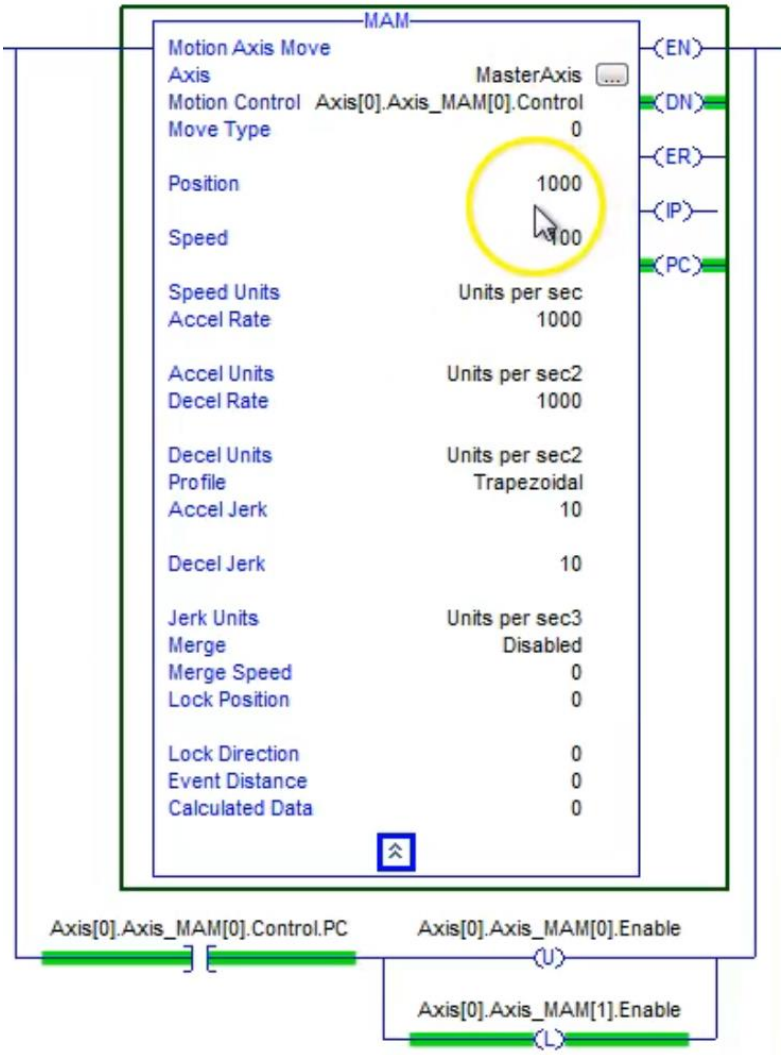
To fix this error, the motion planner execution target value should meet or exceed the value of Output_Cam.Input.

## 54. MAOC Cam Used In a Linear

What is the best application for a MAOC? Usually a rotary axis is used, this is because of the rollover (unwind value). Have a look at our emulated application…




So our position unwind is 200000 with a conversion constant of 8000 gives us 25 revolutions per reset/rollover.




Now we have configured this linear axis to run to position 1000 and then the .PC bit will fire which will clear Axis[0].Axis_MAM[0].Enable and set Axis[0].Axis_MAM[1].Enable.

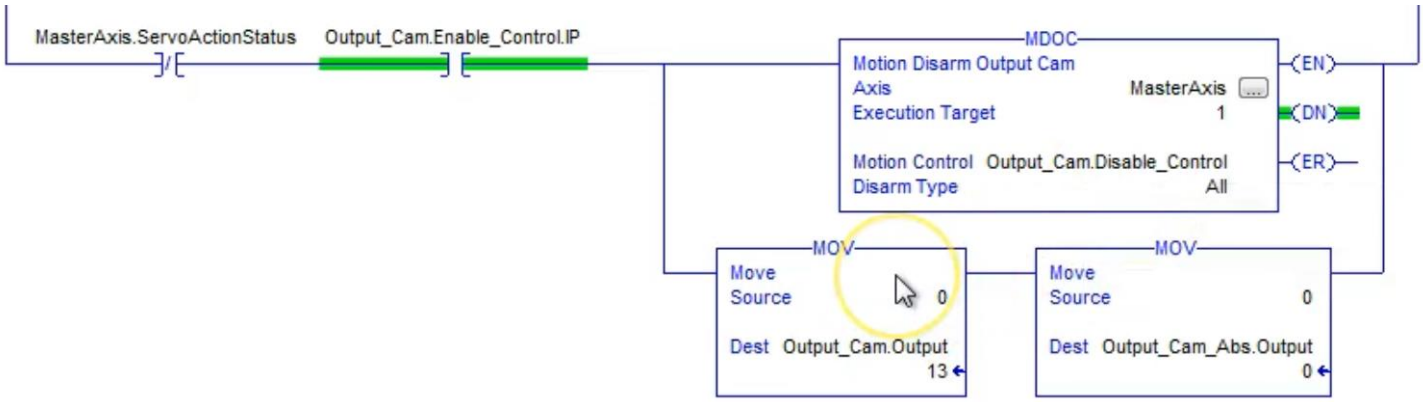
What Axis[0].Axis_MAM[1].Enable does is run back to zero. After which it will disable itself and enable the original axis so it can run to 1000 again. Process continues.

While this is happening a MAOC is firing off the positions. You can use the MAOC to do this but is it the best application for this instruction? Best way to implement this function? Maybe not.

Note that during the above scenario the MAOC Ececution Schedule is set to Bi-Directional. This tells MAOC to execute the cam in both directions.

# 55. MAOC Code Explained

Explanation of the MAOC code. Be aware of setting Position Reference, can be Command of Actual.
Actual = use the encoder of the motor, this is more accurate
Command = use the position commanded, most system use this command.



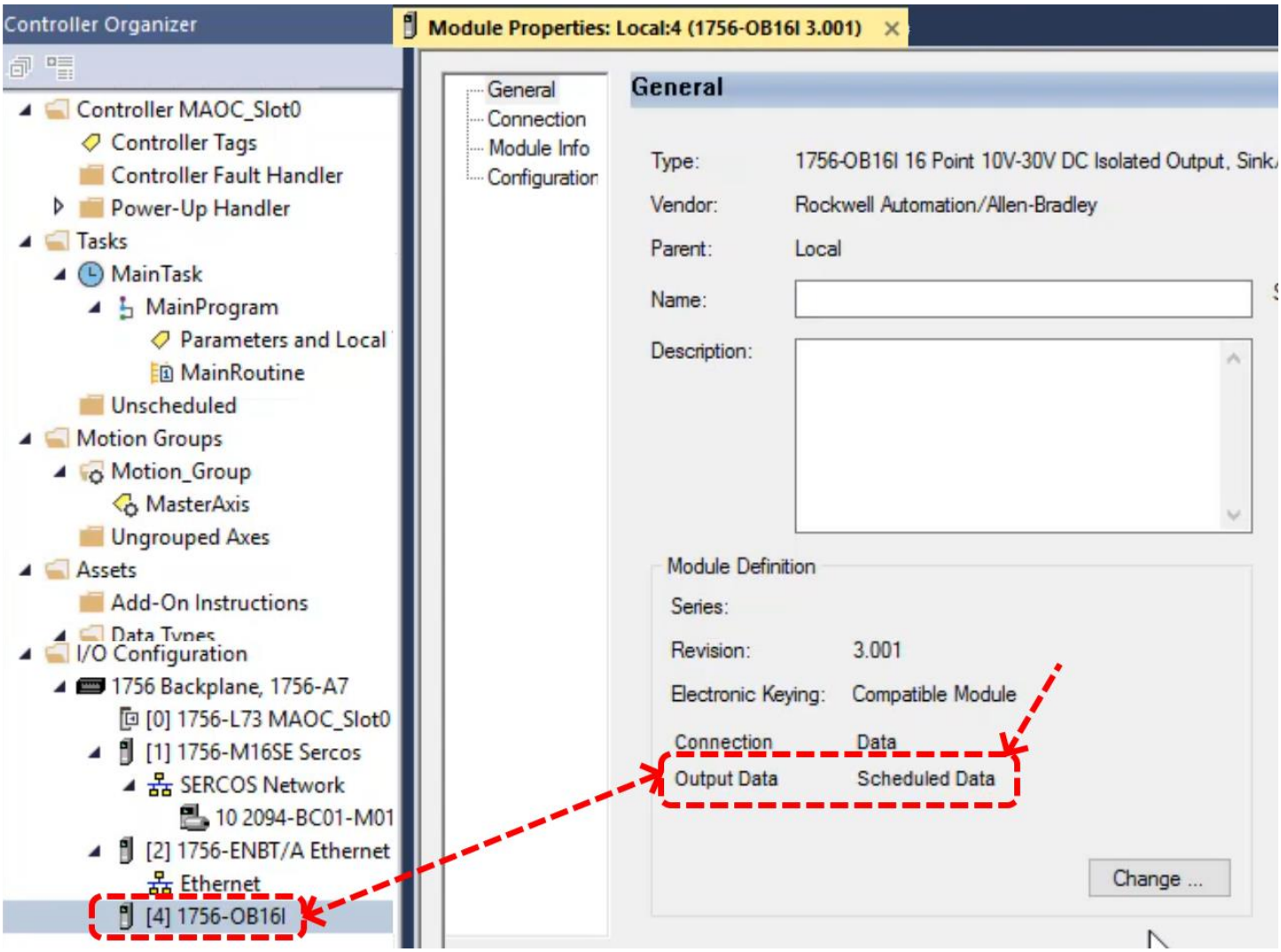

Be aware of **Motion Disable Output Cam**, here we are invoking it when ActionStatus is off and the MAOC declaration tag is done (.ip = true). It is very important to issue this command in order to disable cam outputs, this is you putting the machine in a safe condition. Otherwise the outputs remain active after shutdown.

Moving a zero into Output_Cam.Output will shutdown the cam action (all disabled).
Output_Cam_Abs.Output is the same thing on the other axis.

## 56. MAOC Update

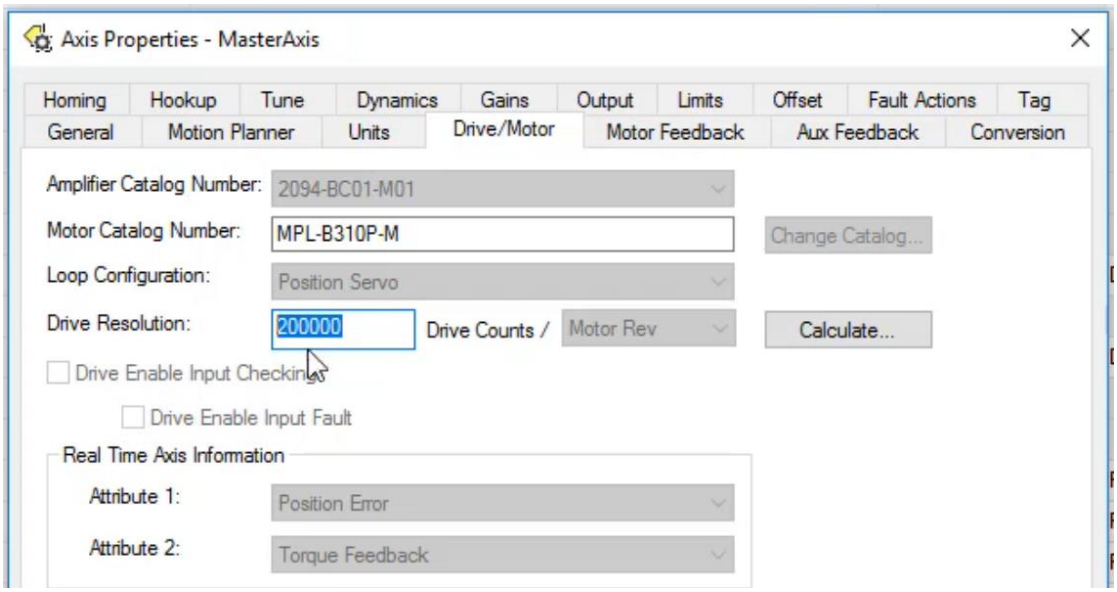
We will create a motion output cam using a MAOC instruction following the position of a servo to actually control an actual output card. In this exercise we are using a hardware / real servo axis. This is a position based system.

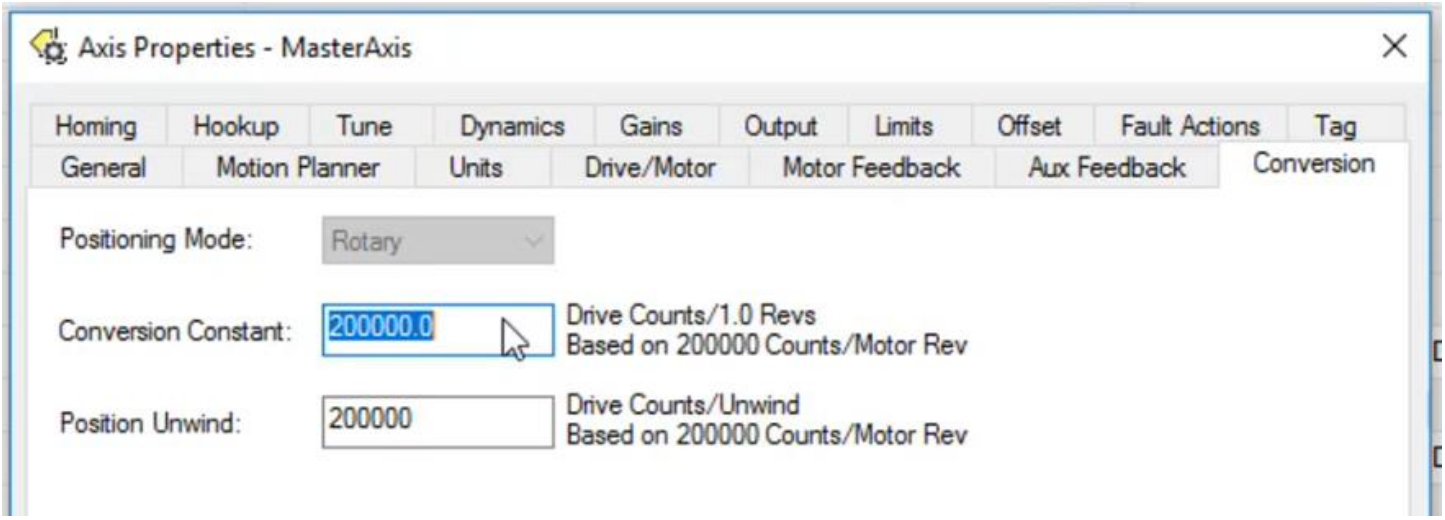
**Item 1: make sure your output card is set up for Scheduled Data. Found in the cards properties dialog.**



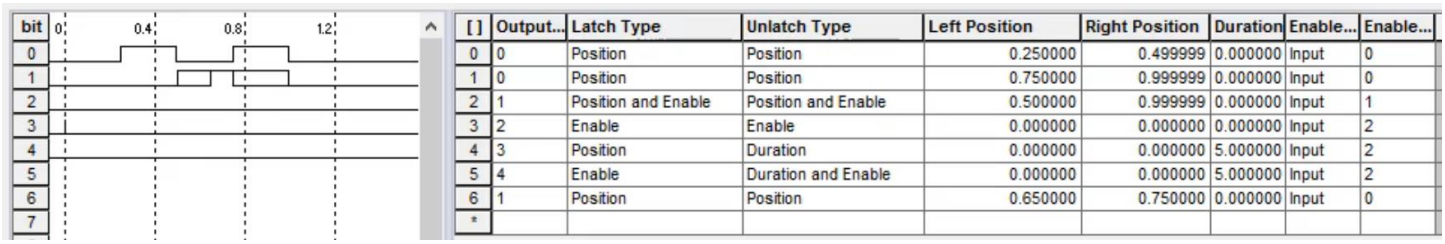

Schedule Data allows the program to go in and control the outputs through the MAOC command.

Note our Drive Resolution: 2000 Drive counts per Motor Rev is equal to the Conversion Constant so that we have one revolution per actual cycle.



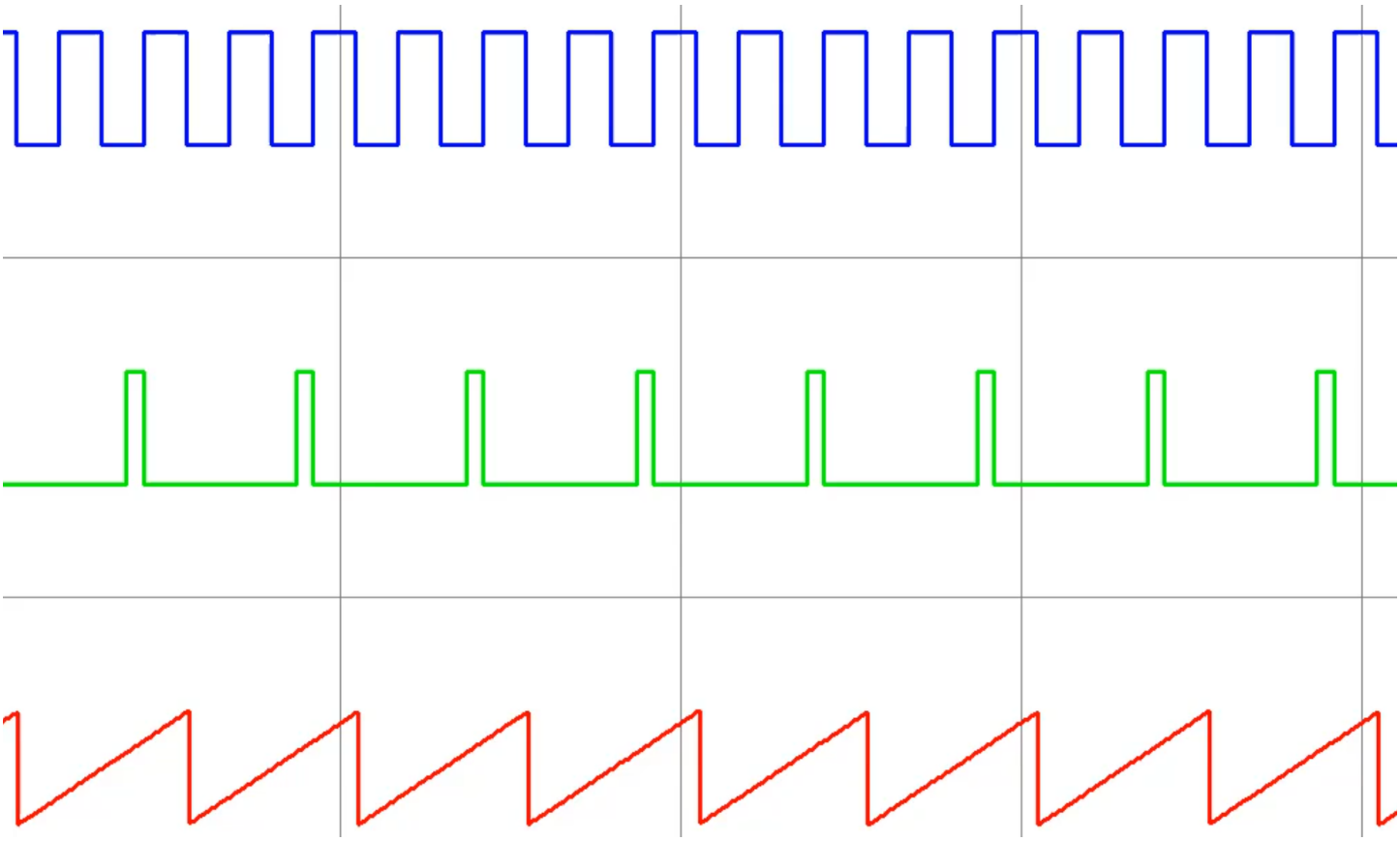

This is the cam profile we will run.



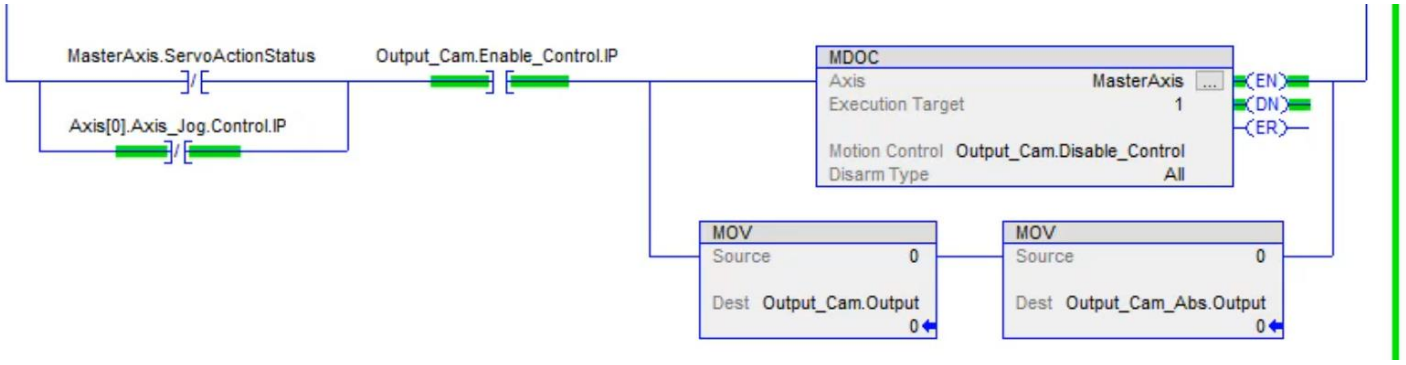
| bit | 0 | 0.4 | 0.8 | 1.2 |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |

| [ ] | Output... | Latch Type | Unlatch Type | Left Position | Right Position | Duration | Enable... | Enable... |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Position | Position | 0.250000 | 0.499999 | 0.000000 | Input | 0 |
| 1 | 0 | Position | Position | 0.750000 | 0.999999 | 0.000000 | Input | 0 |
| 2 | 1 | Position and Enable | Position and Enable | 0.500000 | 0.999999 | 0.000000 | Input | 1 |
| 3 | 2 | Enable | Enable | 0.000000 | 0.000000 | 0.000000 | Input | 2 |
| 4 | 3 | Position | Duration | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 5 | 4 | Enable | Duration and Enable | 0.000000 | 0.000000 | 5.000000 | Input | 2 |
| 6 | 1 | Position | Position | 0.650000 | 0.750000 | 0.000000 | Input | 0 |
| * | | | | | | | | |

Note Latch Type = Position and Unlatch Type = Duration, so you will trigger on position and unlatch after a certain duration.



Sawtooth is the master position rollover, green and blue are outputs.



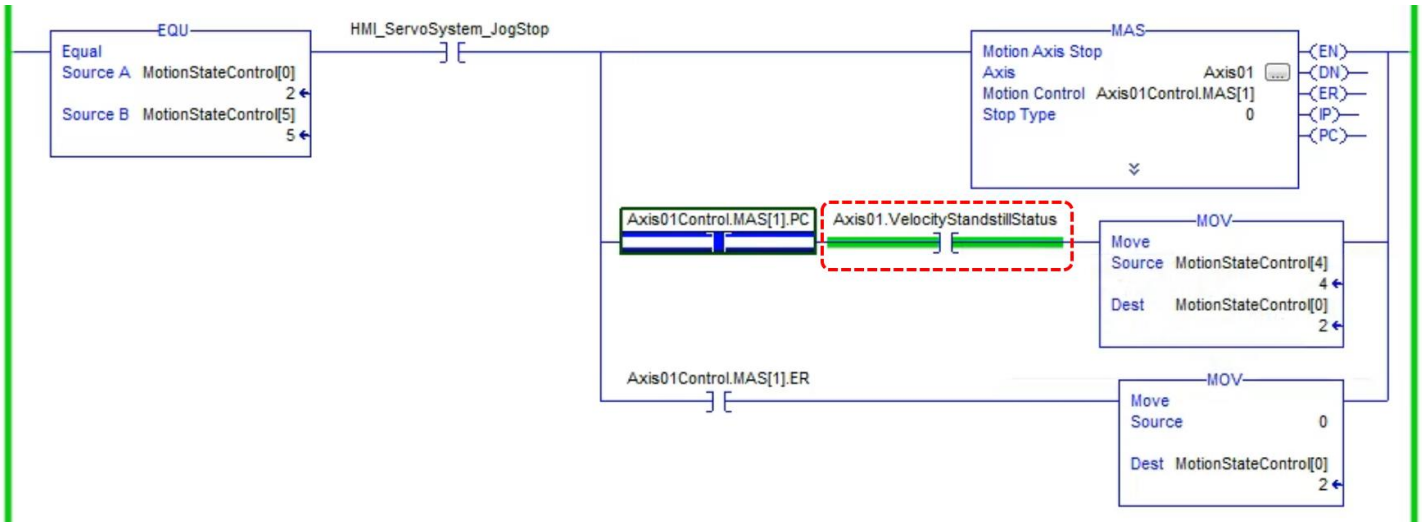

Above we have added a bit in parallel for moving zeros into the Output_Cam.Output and Output_Cam_Abs.Output bits which shutdown the physical outputs (safe state), see previous chapter. The new bit is just saying anytime Axis 0 Jog is not in progress and Output_Cam.Enable_Control.IP is in progress we can shutdown the output bits and issue the MDOC instruction (motion disarm output cam).

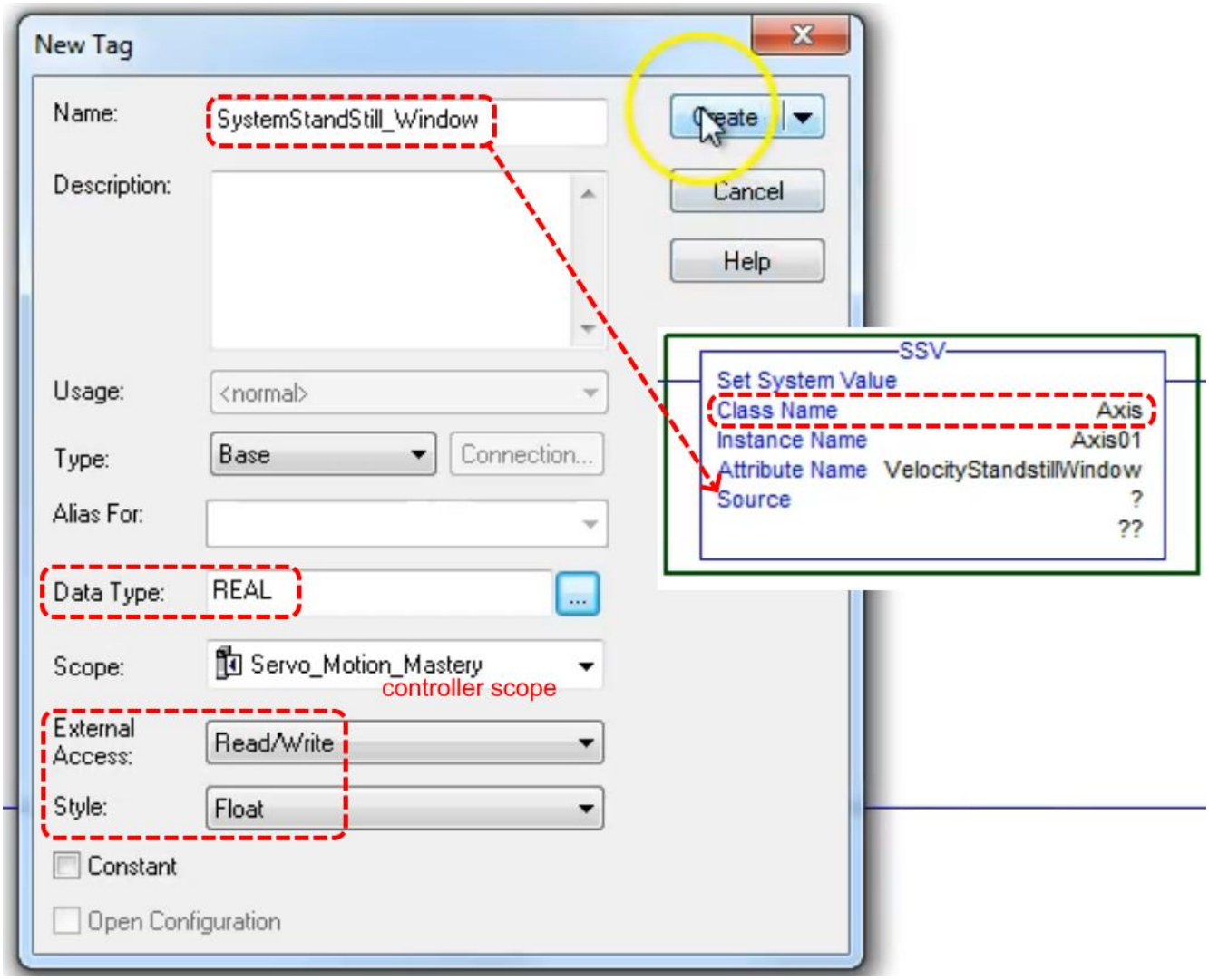
## Key Points on MAOC:

- **Be aware of how your axis is set up.**
- **When not running ISSUE MDOC TO SHUTOFF YOUR MAOC!**

# 57. Using a Axis StandStill Bit

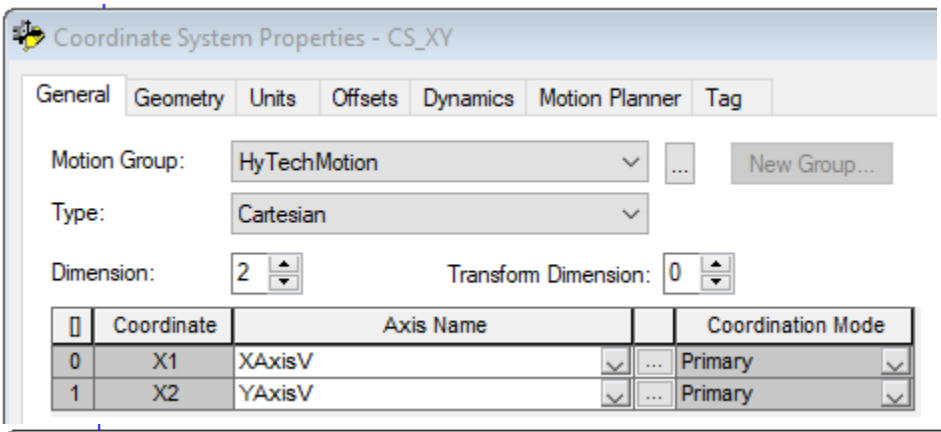
Say you want to make sure the servo is absolutely stopped. To achieve this by using **VelocityStandStillStatus**. This status bit is telling you the servo has absolutely no velocity. But there will be times when the axis is moving very slowly and this will hang up your process. We can adjust the standstill window using SSV. You can see this effect when a servo is not tuned properly and is hunting for the setpoint.

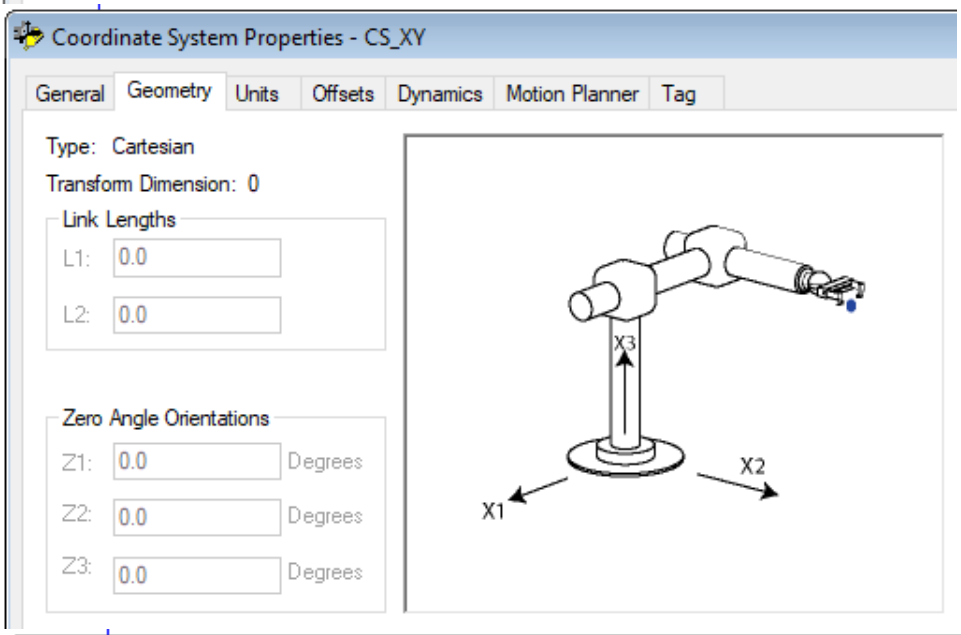


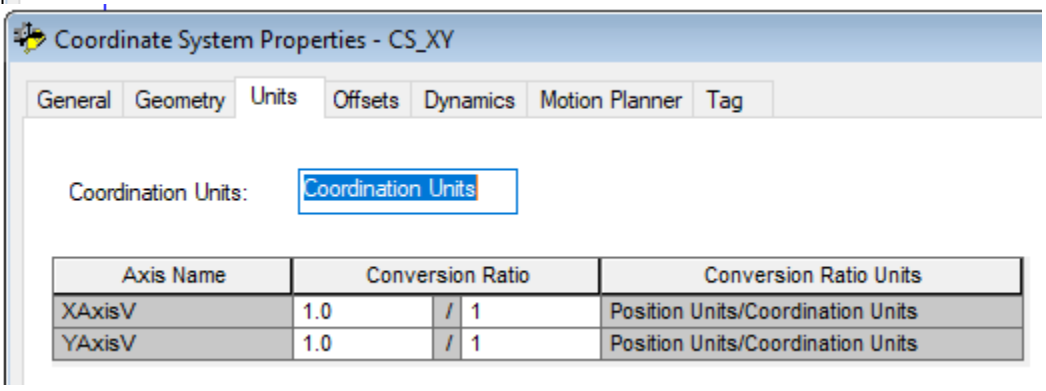




## 58. Intro to a Coordinated Motion

From servo CS_XY we can see how our activity is set up.



Coordinate System Properties - CS_XY

General | Geometry | Units | Offsets | Dynamics | Motion Planner | Tag

Type: Cartesian

Transform Dimension: 0

End Effector Offsets

X1e: 0.0

X2e: 0.0

X3e: 0.0

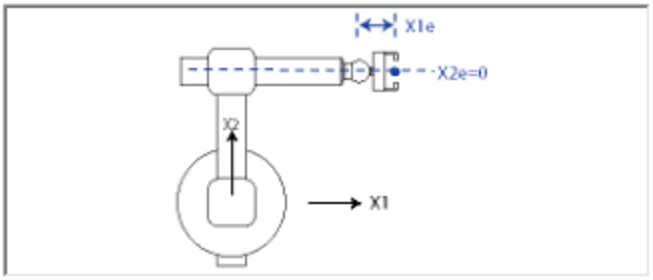
Base Offsets

X1b: 0.0

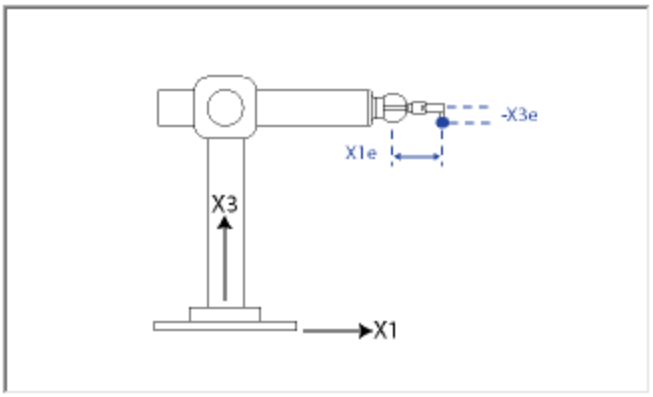
X2b: 0.0

X3b: 0.0

Top View:




Side View:




Coordinate System Properties - CS_XY

General | Geometry | Units | Offsets | Dynamics | Motion Planner | Tag

Manual Adjust...

Vector

| | | | | |
|---|---|---|---|---|
| Maximum Speed: | 3.0 | Coordination Units/s | | |
| Maximum Acceleration: | 20.0 | Coordination Units/s^2 | | |
| Maximum Deceleration: | 20.0 | Coordination Units/s^2 | | |
| Maximum Accel Jerk: | 133.33333 | Coordination Units/s^3 | = 100% of Max Accel Time | Calculate... |
| Maximum Decel Jerk: | 133.33333 | Coordination Units/s^3 | = 100% of Max Decel Time | Calculate... |

Position Tolerance

| | | |
|---|---|---|
| Actual: | 0.005 | Coordination Units |
| Command: | 0.005 | Coordination Units |

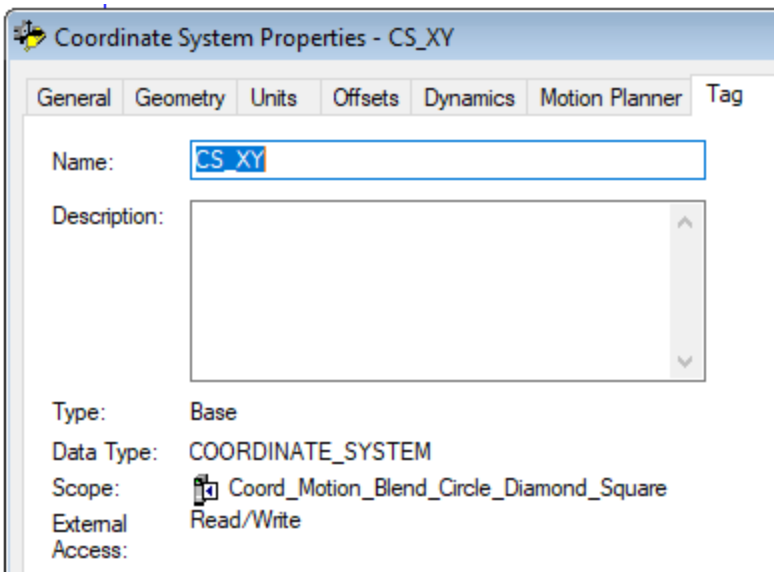
Coordinate System Properties - CS_XY

General | Geometry | Units | Offsets | Dynamics | Motion Planner | Tag

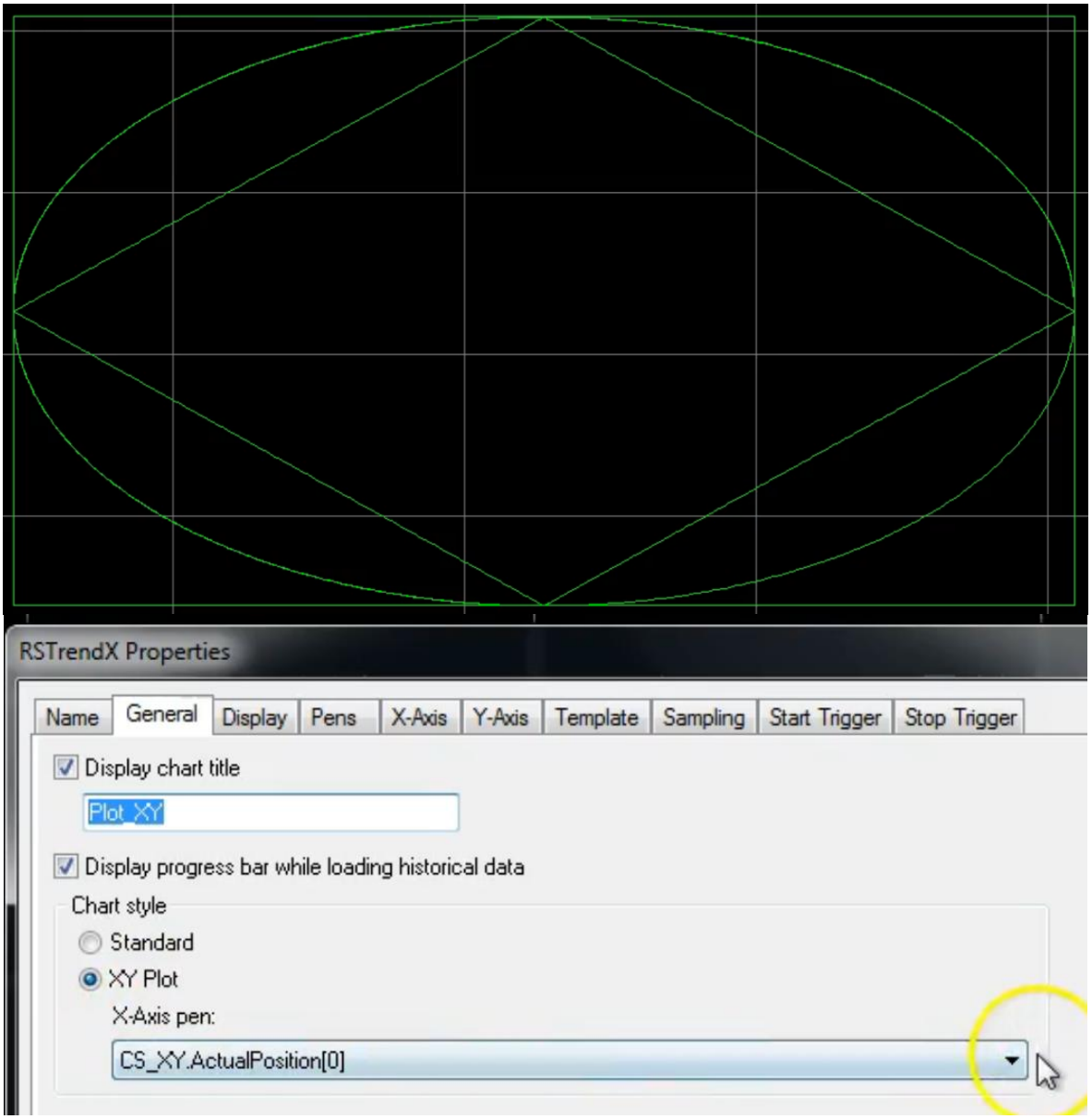
☑ Master Delay Compensation

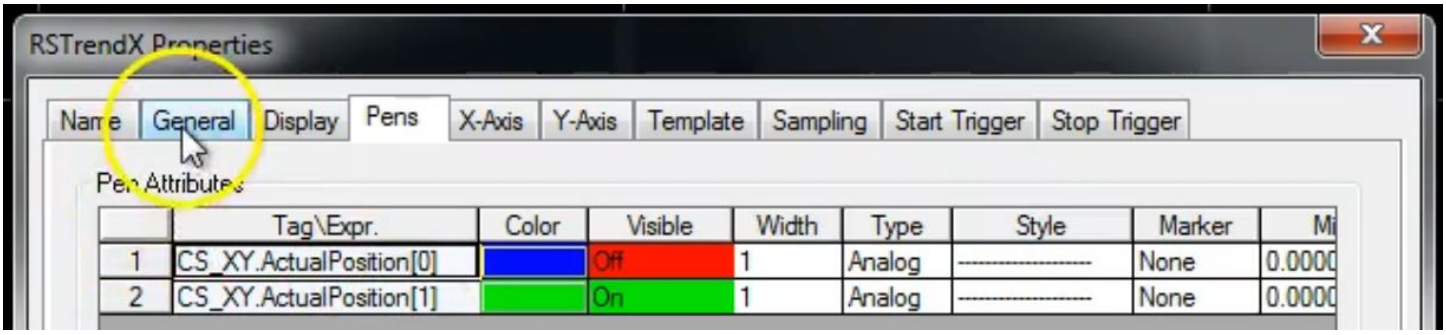
☐ Enable Master Position Filter

Master Position Filter Bandwidth: 0.1 Hertz

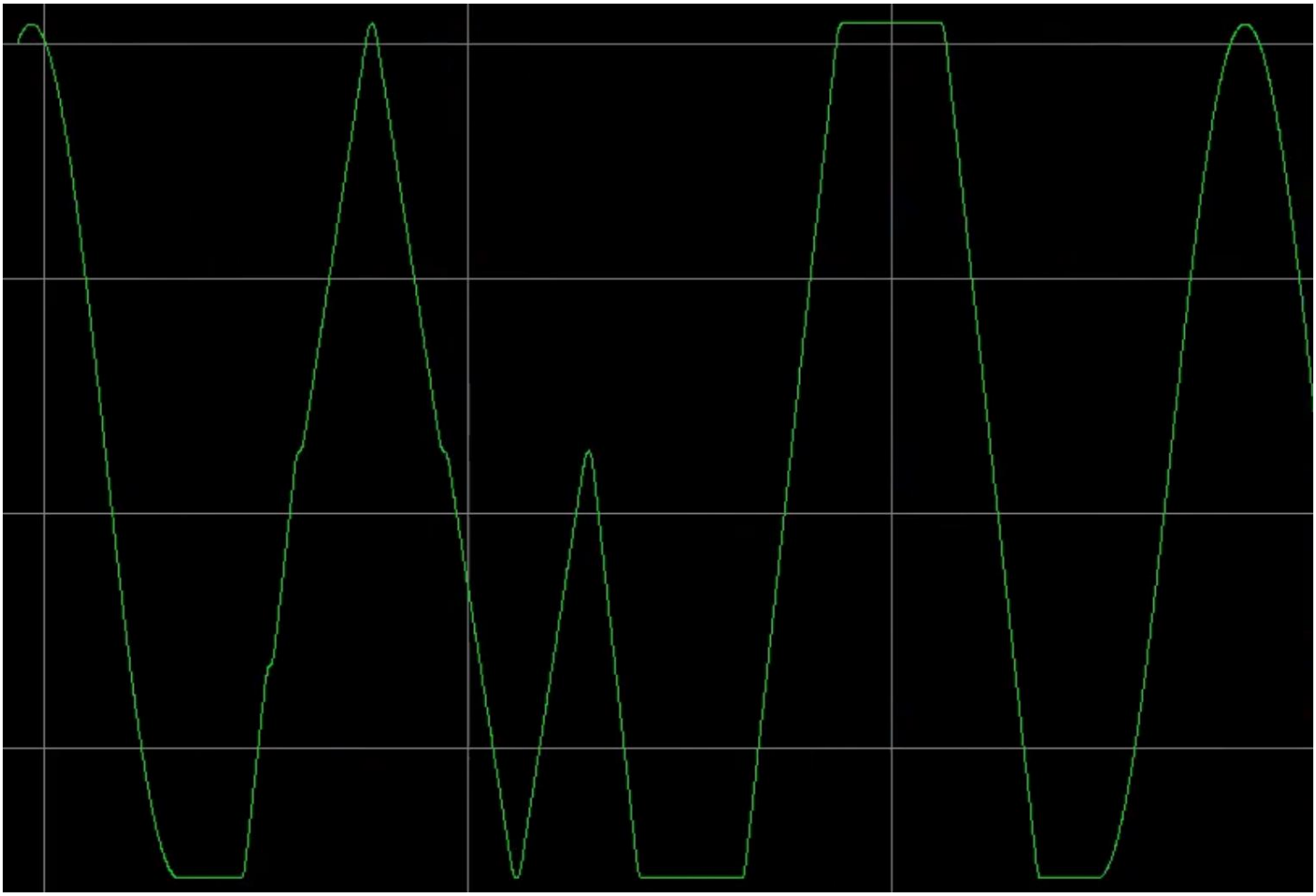

Coordinate System Properties - CS_XY
General
Geometry
Units
Offsets
Dynamics
Motion Planner
Tag
Name:
CS_XY
Description:
Type:
Base
Data Type:
COORDINATE_SYSTEM
Scope:
Coord_Motion_Blend_Circle_Diamond_Square
External Access:
Read/Write

And our plot…

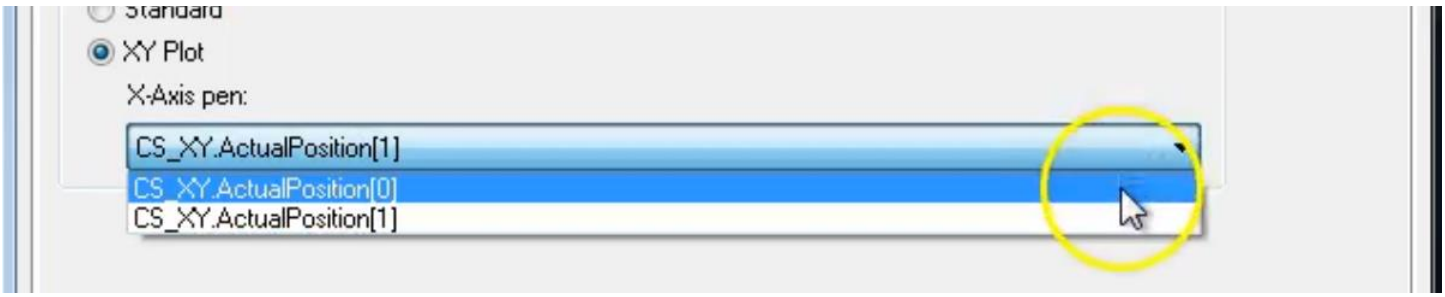

RSTrendX Properties
Name
General
Display
Pens
X-Axis
Y-Axis
Template
Sampling
Start Trigger
Stop Trigger
Display chart title
Plot_XY
Display progress bar while loading historical data
Chart style
Standard
XY Plot
X-Axis pen:
CS_XY.ActualPosition[0]



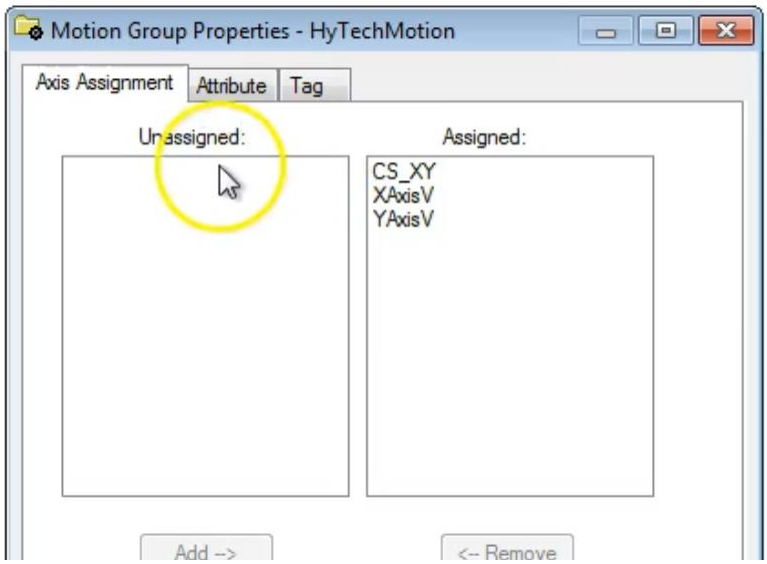

This is what the same pen would look like as a time based trend…



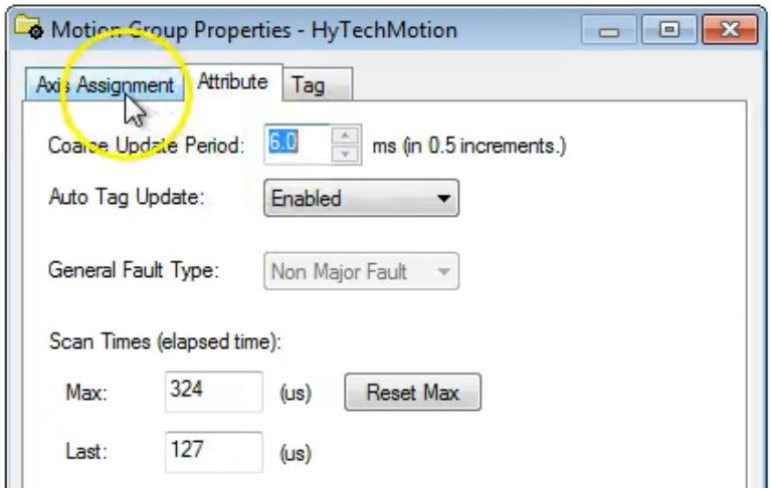
Note that there are two pens, even in XY Plot mode they give different traces.



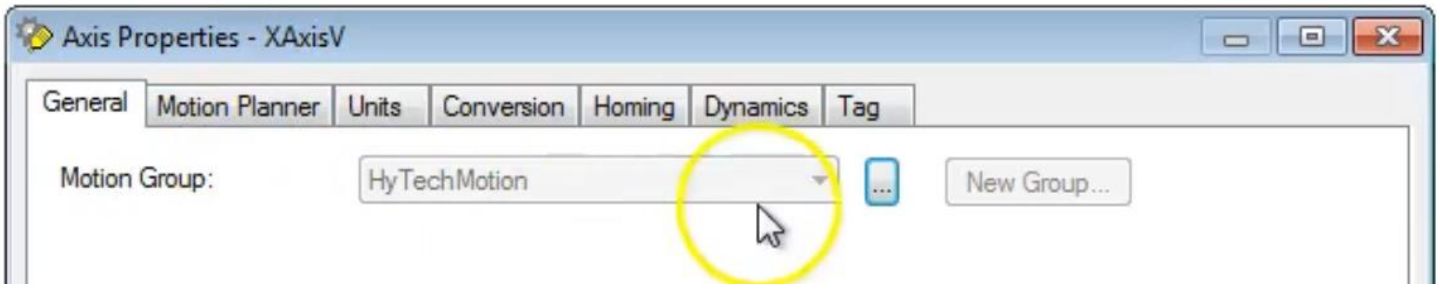
# 59. Coordinated Motion Section



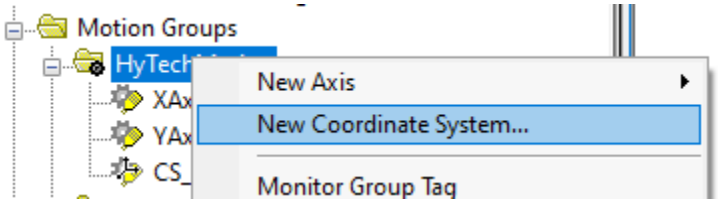

Standard motion group.

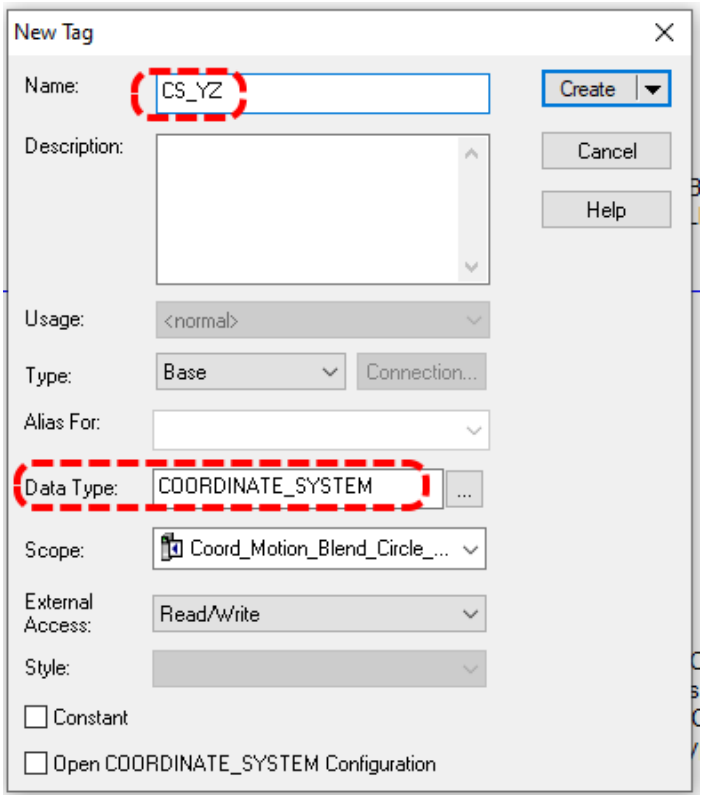


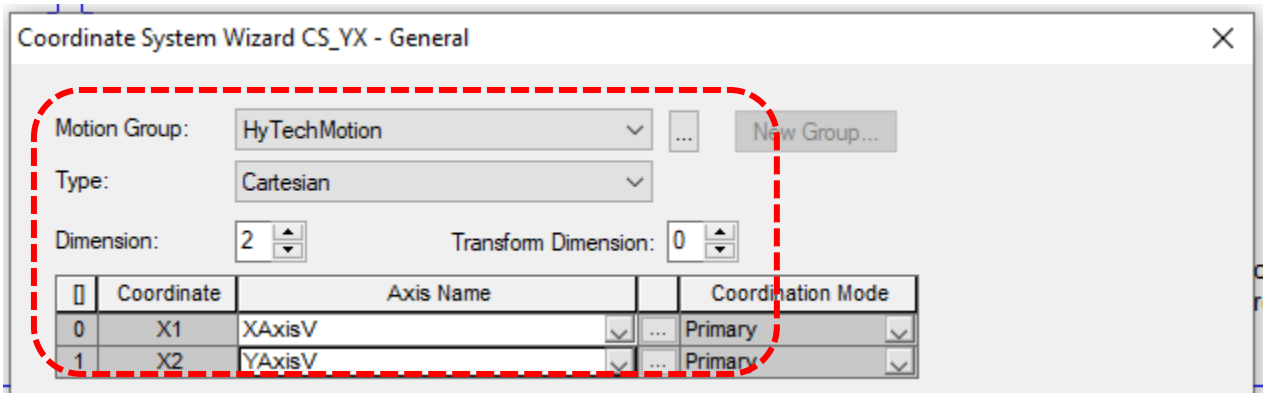

Now configure the axis… The X and Y axis are virtual axis.



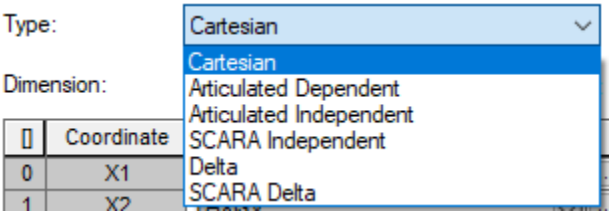
To create a cartesian axis system… Create a new tag of type COORDINATE_SYSTEM.
Coordinate System is really referring to Coordinat**ed** System.



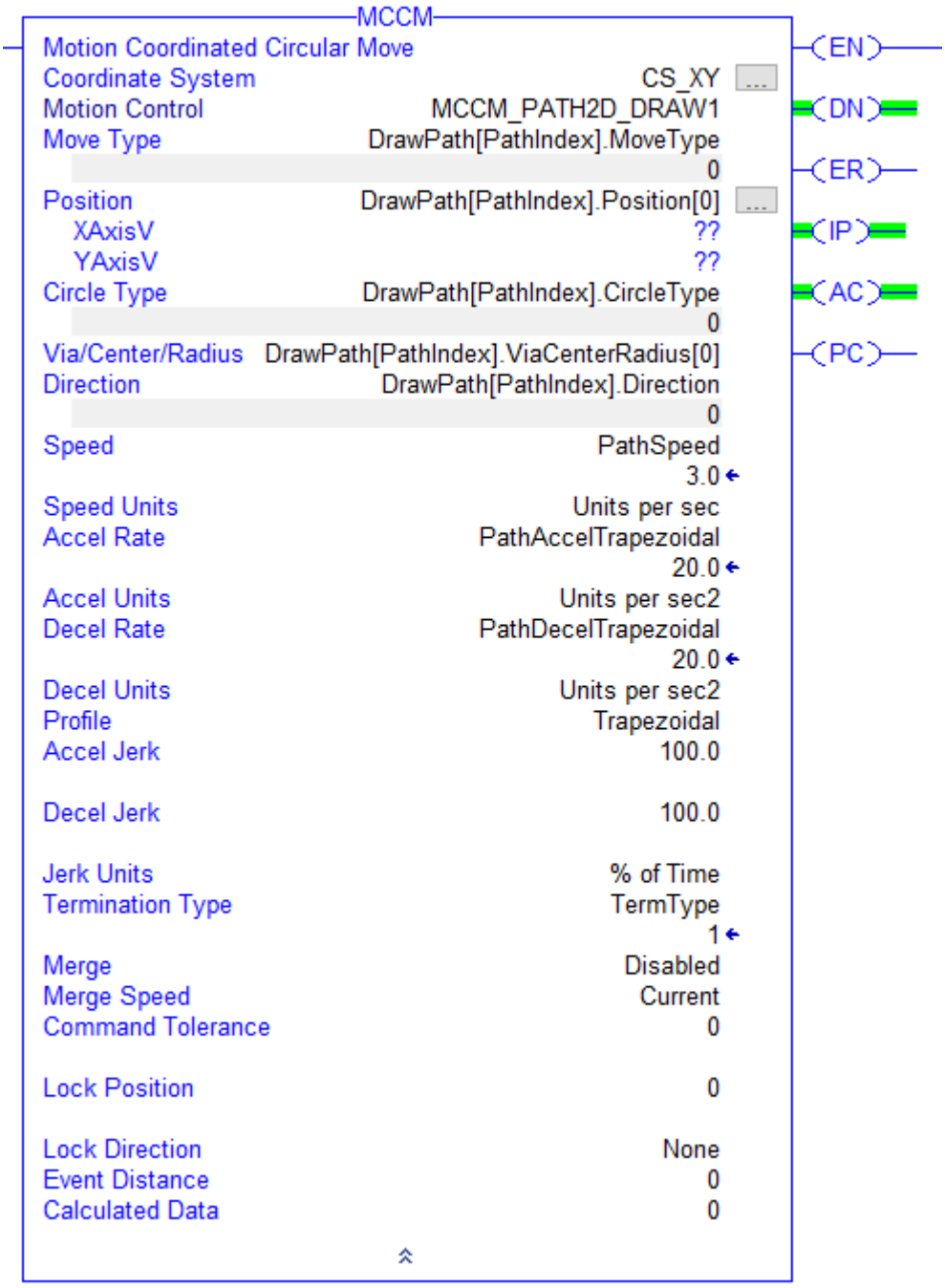




Types of coordinated systems include …



The command for a circular movement is **Motion Coordinated Circular Move (MCCM)**.




Our motion instruction has a new output bit, .AC. This stands for:

When you have a coordinated move instruction queued, the Active bit lets you know which instruction is controlling the motion. It sets when the coordinated move becomes active. It is reset when the Process Complete bit is set or when the instruction is stopped.

# 60. Coordinated Motion Section

**Motion Coordinated Linear Move (MCLM)**

Use the MCLM instruction to start a single or multi-dimensional linear coordinated move for the specified axes within a Cartesian coordinate system. You can define the new position as either absolute or incremental.

The Motion Coordinated Linear Move (MCLM) instruction performs a linear move using up to three (3) axes statically coupled as primary axes in a Cartesian coordinate system. You specify whether to use an absolute or incremental target position, the desired speed, maximum acceleration, maximum deceleration, acceleration jerk, deceleration jerk, and the units of each. The actual speed is a function of the programmed units of the speed (Units per sec, or % of Maximum, as configured for the coordinate system), and the combination of primary axes that are commanded to move. Each axis is commanded to move at a speed that allows all axes to reach the programmed endpoint (target position) at the same time.

**Dwells**

You have the option to program a dwell using Time Based Programming in either Time Driven Mode or MDSC Mode when a zero length move (see Zero Length Move below) is programmed. The acceleration, deceleration, and jerk parameters are ignored when a zero length move is programmed. Therefore, when in time driven mode, the duration of the dwell is in seconds. When in MDSC mode, the duration of the dwell is programmed in units of Master Distance.

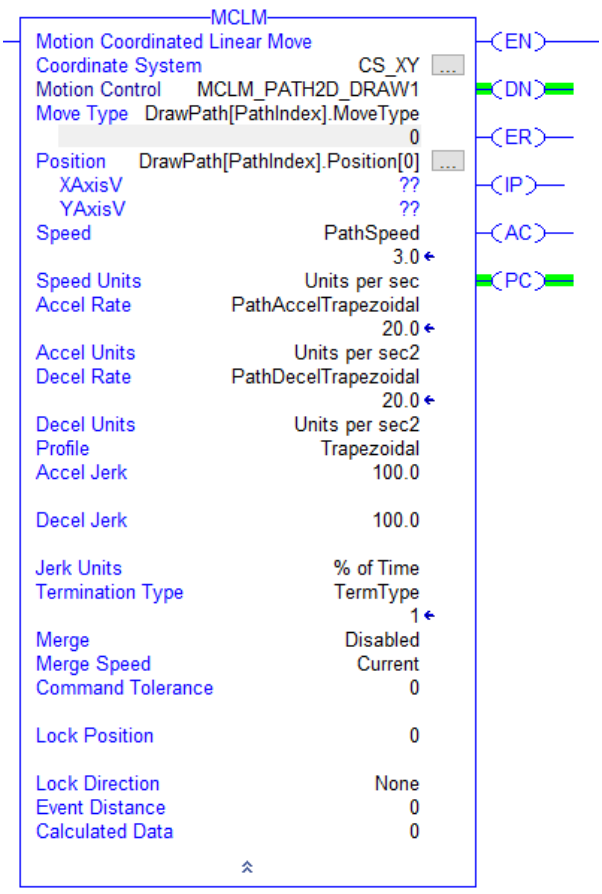
In MDSC mode, the dwell starts either at the Master Lock Position or immediately, depending on the programmed Lock Direction parameter, and continues for a duration as specified in the Speed parameter.

**Zero Length Move**

In Master Driven Mode and Time Driven Mode, you have the option of configuring a move with a Slave distance increment of zero (or a move with the same target and current position). If speed is specified in Master Units, the move remains active until the specified Master distance has been traversed. Use this type of move to generate a dwell in a multi-segment path move.

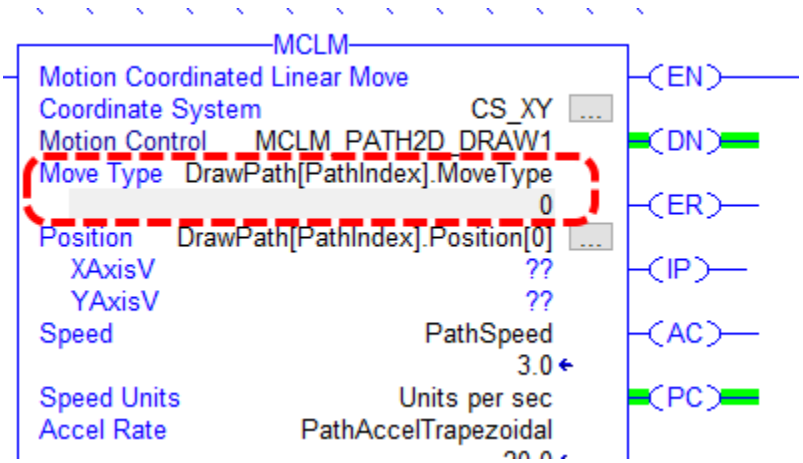
Similarly, when you program the move time directly in seconds in Time Driven Mode, a move of the duration of X seconds with a zero departure results in a programmed delay of the specified time.

Instructions with zero length cause velocity of the multi-axis instruction preceding the one with zero length to decelerate to zero at its endpoint. To avoid this behavior, it is suggested that you eliminated moves with zero length from your program.



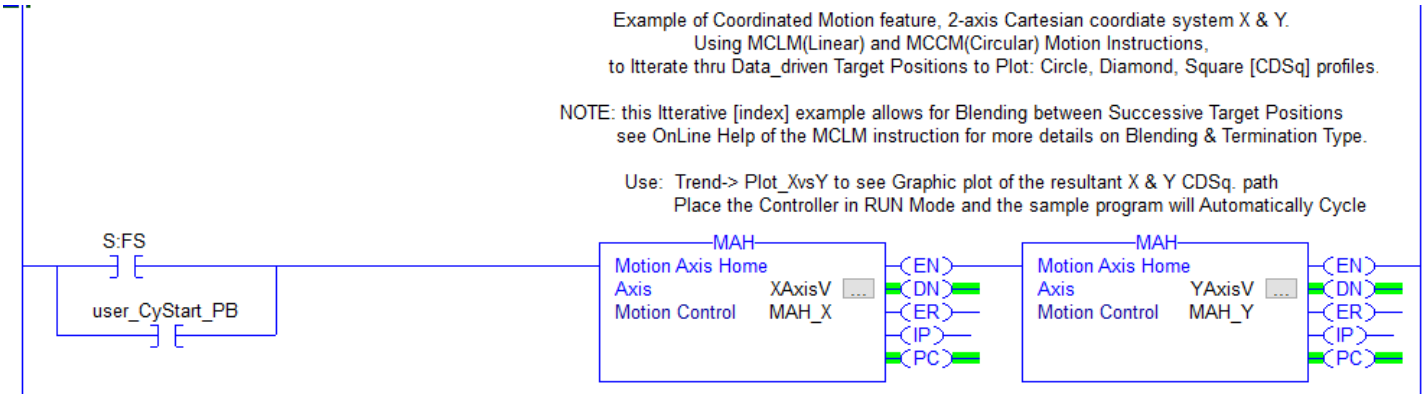

| Operand | Type | Format | Description |
|---|---|---|---|
| Coordinate System | COORDINATE_SYSTEM | Tag | Coordinated group of axes. |
| Motion Control | MOTION_INSTRUCTION | Tag | Structure used to access instruction status parameters. |
| Move Type | SINT, INT, or DINT | Immediate or Tag | Select the Move Type:<br>0 = Absolute<br>1 = Incremental |
| Position | REAL | Array tag [ ] | [coordination units] |
| Speed | SINT, INT, DINT, or REAL | Immediate or Tag | [coordination units] |
| Speed Units | SINT, INT, or DINT | Immediate | 0 = Units per Sec<br>1 = % of Maximum<br>4 = Units per MasterUnit<br>7 = Master Units |
| Accel Rate | SINT, INT, DINT, or REAL | Immediate or Tag | [coordination units] |
| Accel Units | SINT, INT, or DINT | Immediate | 0 = Units per Sec$^2$<br>1 = % of Maximum<br>4 = Units per MasterUnit$^2$<br>7 = Master Units |
| Decel Rate | SINT, INT, DINT, or REAL | Immediate or Tag | [coordination units] |
| Decel Units | SINT, INT, or DINT | Immediate | 0 = Units per Sec$^2$<br>1 = % of Maximum<br>4 = Units per MasterUnit$^2$<br>7 = Master Units |
| Profile | SINT, INT, or DINT | Immediate | 0 = Trapezoidal<br>1 = S-curve |
| Accel Jerk | SINT, INT, DINT, or REAL | Immediate or Tag | You **must** always enter values for the Accel and Decel Jerk operands. This instruction only uses the values if the Profile operand is configured as S-curve.<br>Enter the jerk rates in these Jerk Units.<br>0 = Units per sec$^3$<br>1 = % of Maximum<br>2 = % of Time<br>4 = Units per MasterUnit$^3$<br>6 = % of Time-Master Driven<br>7 = Master Units |
| Decel Jerk | SINT, INT, DINT, or REAL | Immediate or Tag | |
| Jerk Units | SINT, INT, or DINT | Immediate or Tag | Use these values to get started.<br>Accel Jerk = 100 (% of Time)<br>Decel Jerk = 100 (% of Time)<br>Jerk Units = 2 |

In our example the MCLM instruction uses indirect addressing for the Move Type tag. What they are doing is changing between type 0 (absolute) and type 1 (incremental) depending on which line segment is being calculated and drawn.

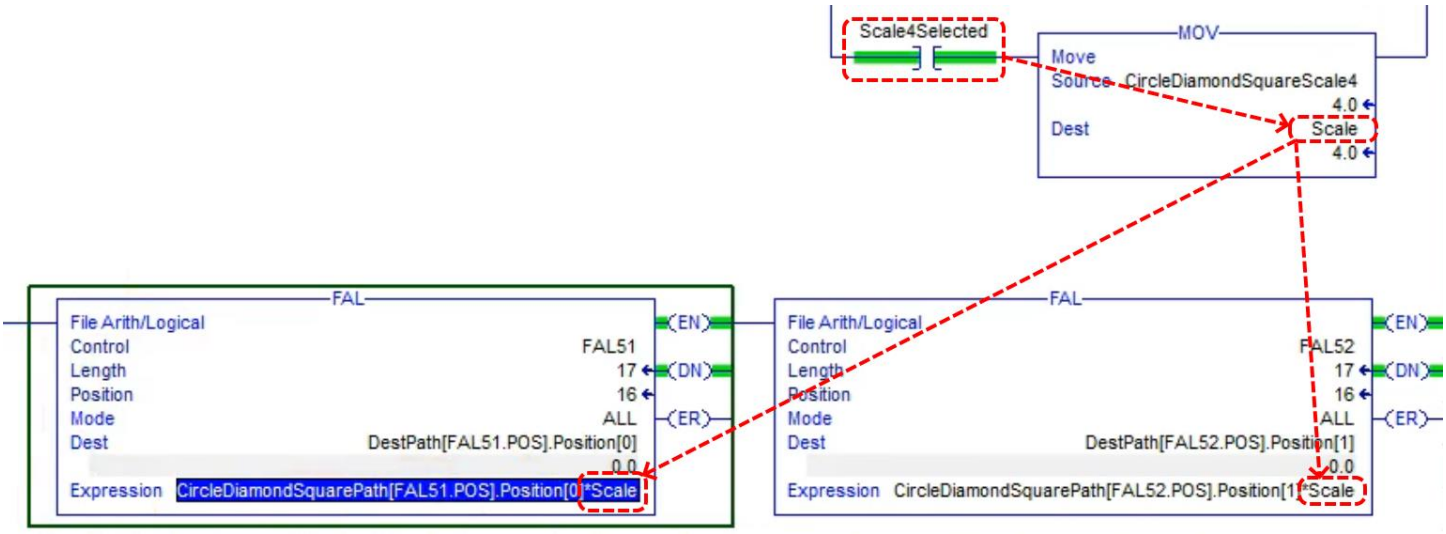


# 61. Coordinated Motion Section

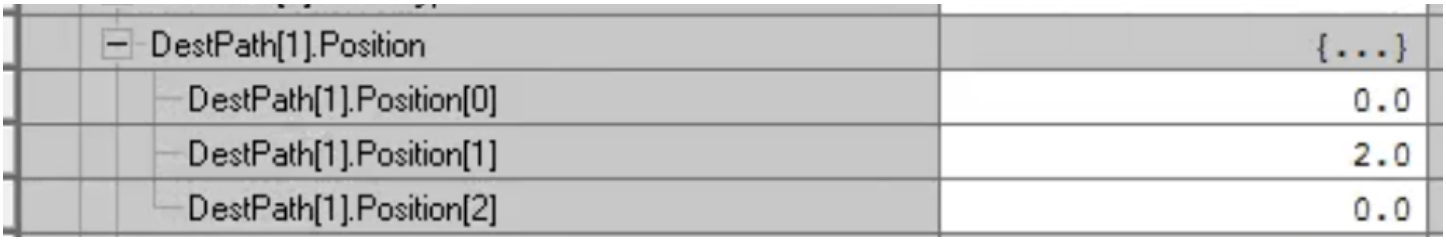
We begin the Run_Example_CirDSqr module by homing the two axis. Cycle Stop PB will do the same.



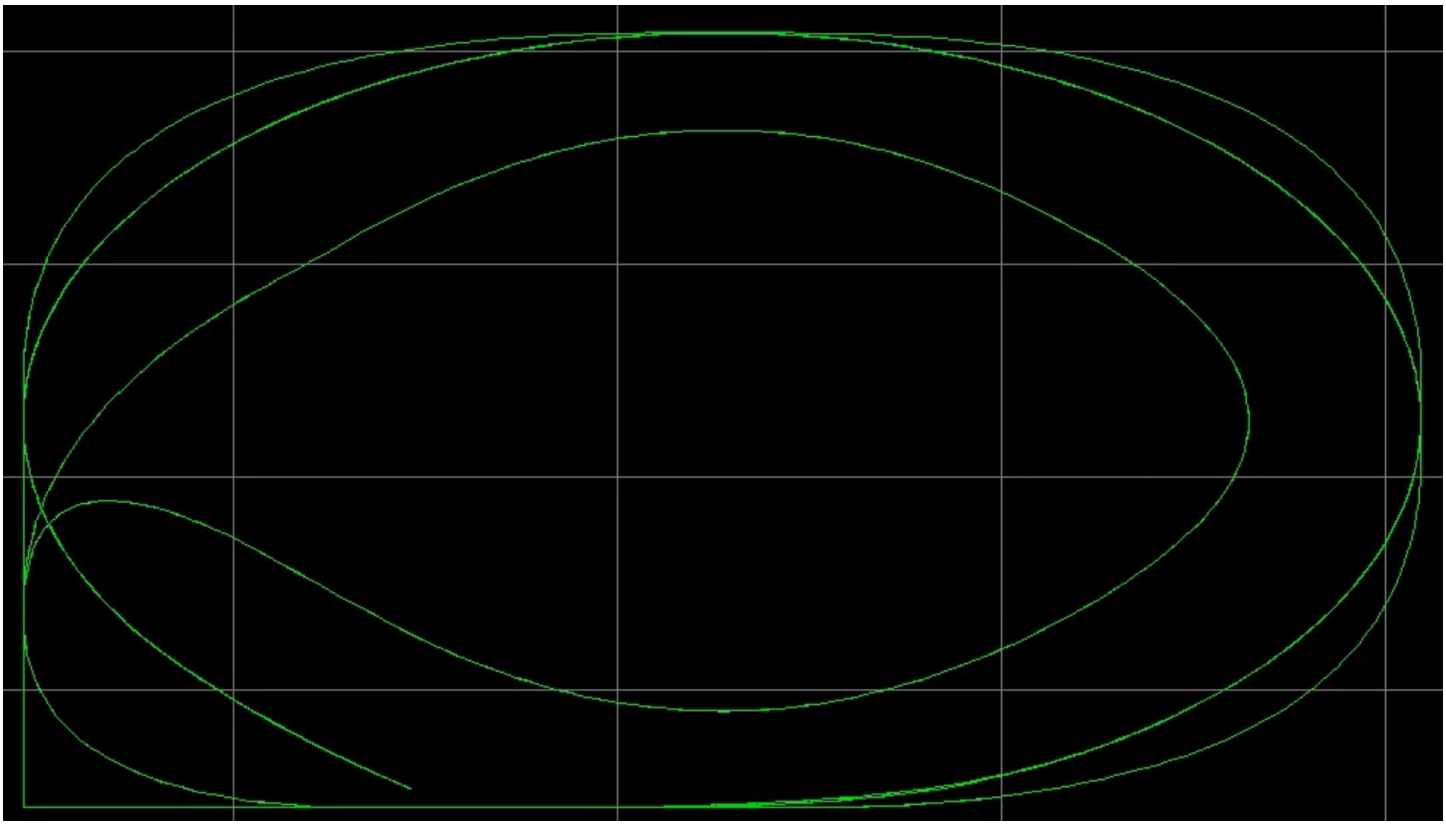

Once homing is complete we transition to MotionCircleDiamondSquare.



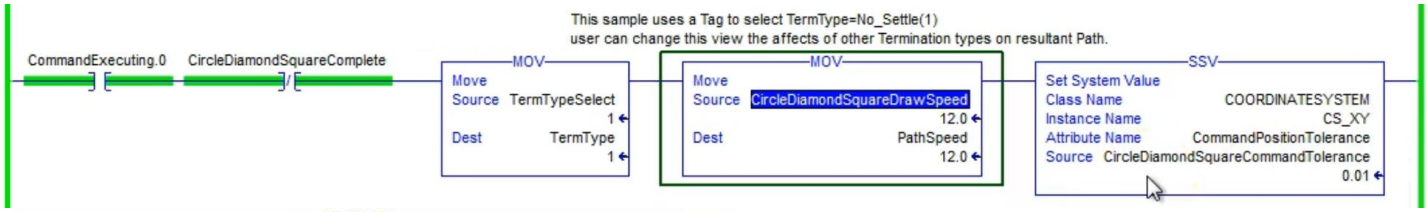

The FAL instructions make use of parameters FAL51.POS as an index into the path array. But what is it? Note FAL51 is the name of the control tag.

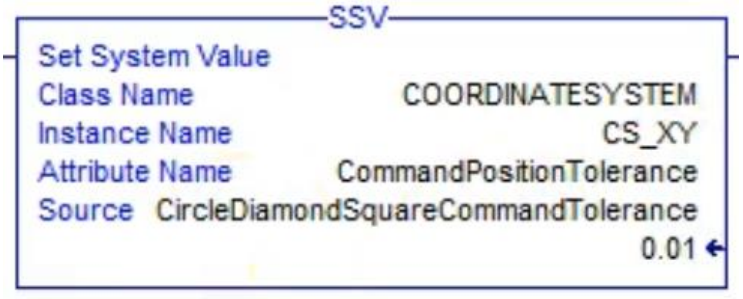



If you follow the DestPath[#].Position[#] parameter in the program tags area you'll see that the path entries change as you move through the DestPath[] index.



What happened here? The course rate update and the speed settings we have entered are not capable of creating the path trace we want. Note that Termination Type directly effects the course rate update (?). Basically, the x and y axis have to move so fast to make the speed they cannot create the trace we want.

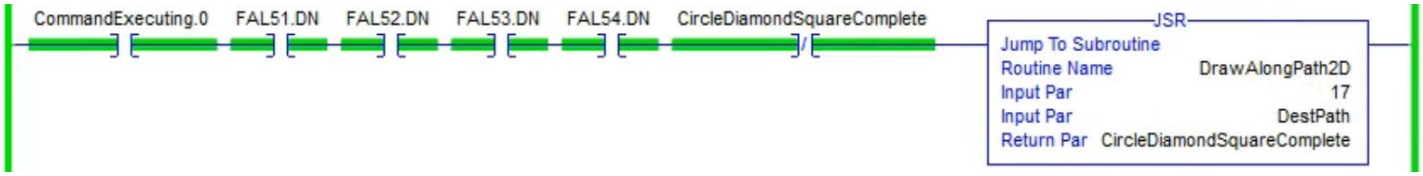


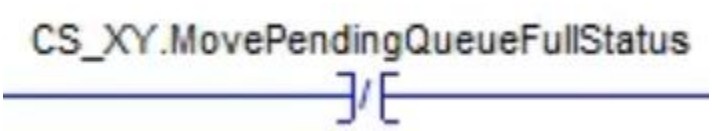




Here we use SSV to adjust the value of CommandPositionTolerance for our coordinated axis.

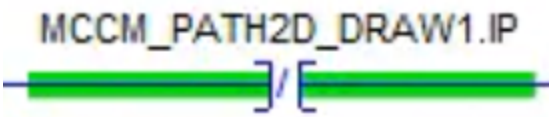
The commands to here give us our path. Now we will jump to DrawAlongPath2D passing some parameters.

Note that the blocks will not execute if PathError=T.

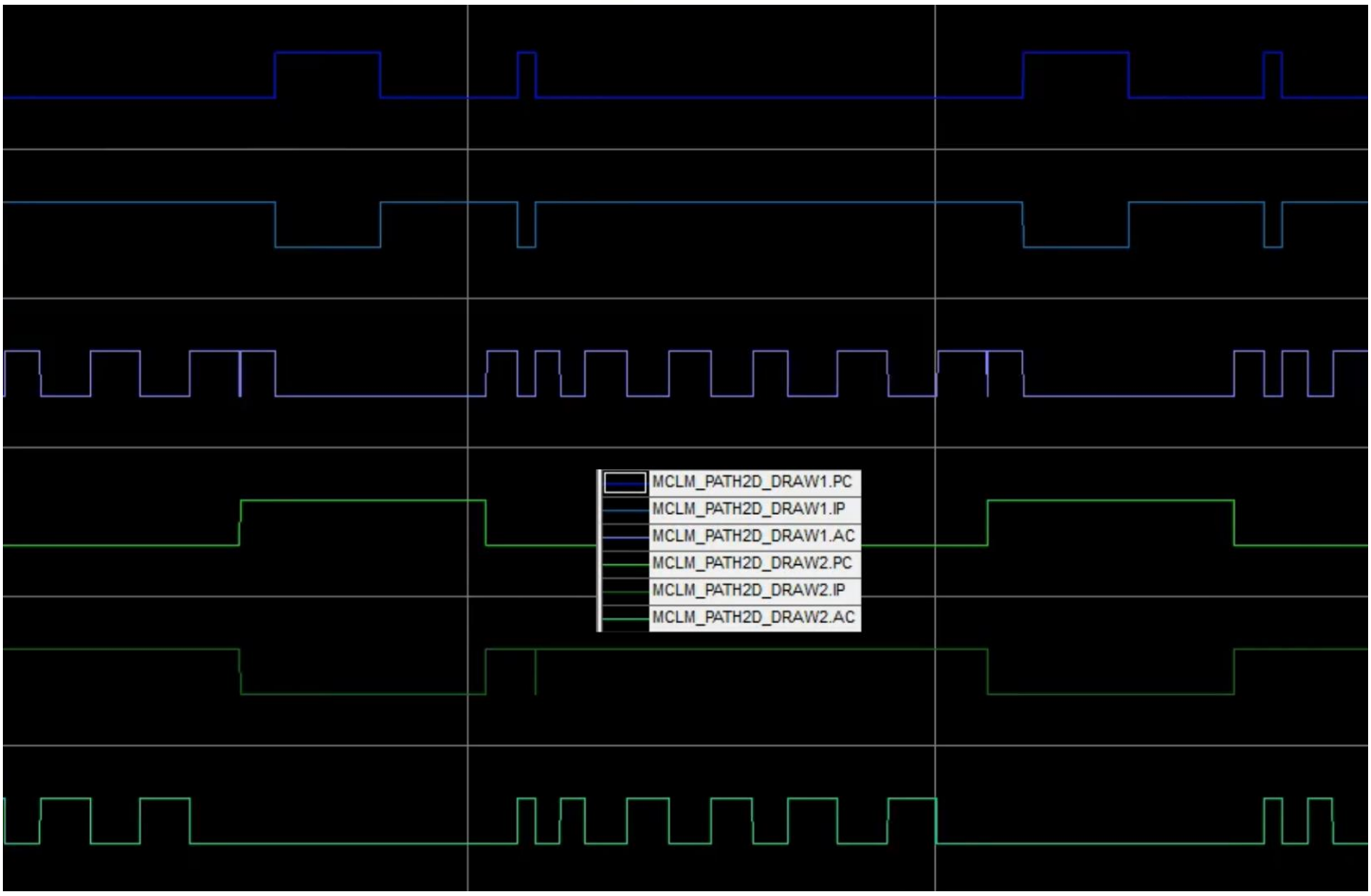



Prevents a pending move from being queued up if a move is already taking place.




The way we tell if a move is in process is the IP bit.


| | |
|---|---|
| DrawPath | {...} |
| DrawPath[0] | {...} |
| DrawPath[0].CircleType | 0 |
| DrawPath[0].Direction | 0 |
| DrawPath[0].InstructionType | 0 |
| DrawPath[0].MoveType | 1 |
| DrawPath[0].Position | {...} |
| DrawPath[0].ViaCenterRadius | {...} |

# 62. Kinetix 6000 Inputs

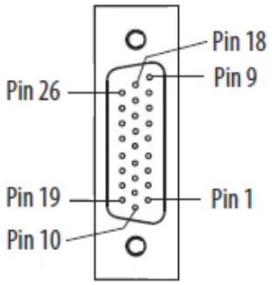

## I/O Connector Pinout

**Table 27 - IAM/AM I/O 26-pin (IOD) Connector**

| IOD Pin | Description | Signal |
|---|---|---|
| 1 | Hardware enable 24V DC power supply | +24V_PWR |
| 2 | Hardware enable input | ENABLE |
| 3 | Common | +24V_COM |
| 4 | Home switch 24V DC power supply | +24V_PWR |
| 5 | Home switch input | HOME |
| 6 | Common | +24V_COM |
| 7 | Positive overtravel 24V DC power supply | +24V_PWR |
| 8 | Positive overtravel limit switch input | OT+ |
| 9 | Common | +24V_COM |
| 10 | Negative overtravel 24V DC power supply | +24V_PWR |
| 11 | Negative overtravel limit switch input | OT- |
| 12 | Common | +24V_COM |
| 13 | 24V registration power | REG_24V |

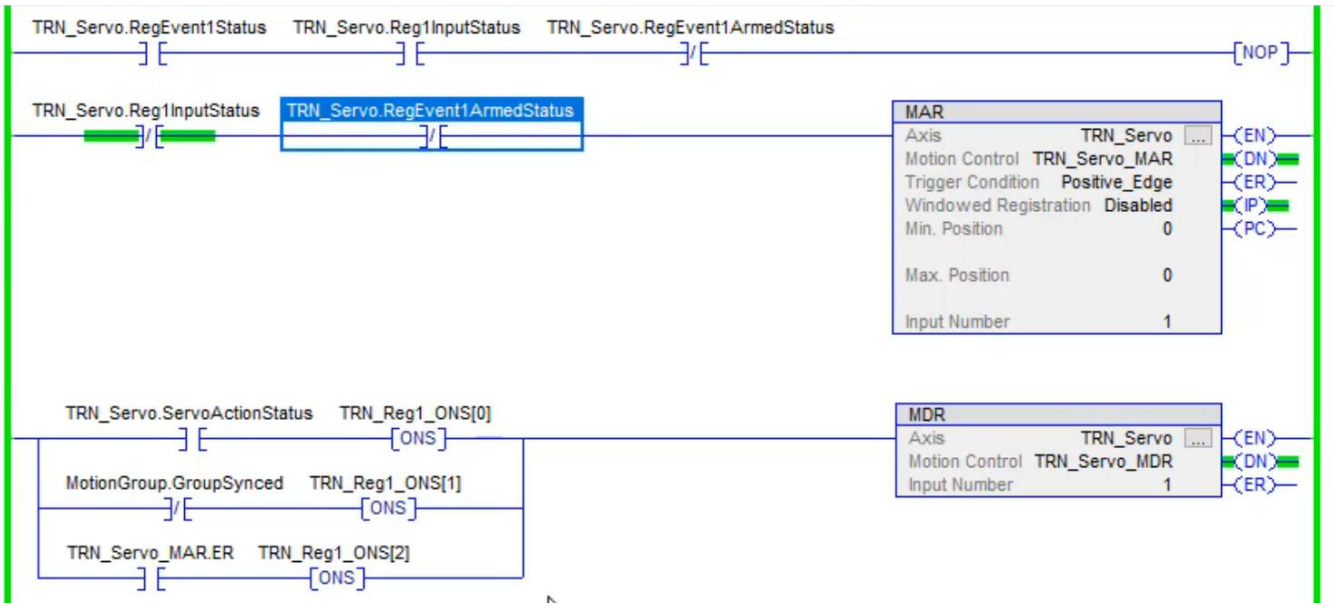
| IOD Pin | Description | Signal |
|---|---|---|
| 14 | High speed registration 1 input | REG1 |
| 15 | Common for registration | REG_COM |
| 16 | 24V registration power | REG_24V |
| 17 | High speed registration 2 input | REG2 |
| 18 | Common for registration | REG_COM |
| 19 | Reserved | – |
| 20 | Reserved | – |
| 21 | Reserved | – |
| 22 | Reserved | – |
| 23 | Analog output 0 | DAC0 |
| 24 | Analog output common | DAC_COM |
| 25 | Analog output 1 | DAC1 |
| 26 | Analog output common | DAC_COM |

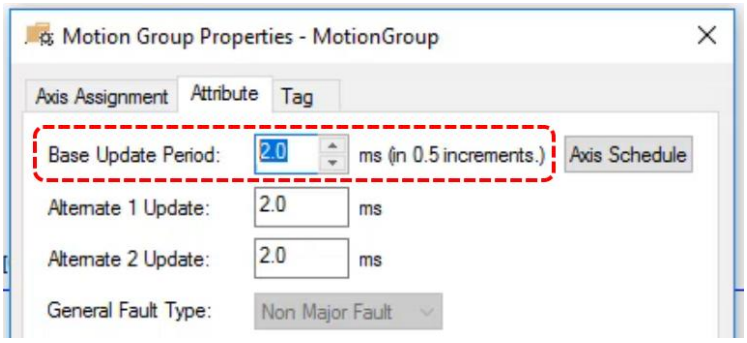
**Figure 26 - Pin Orientation for 26-pin I/O (IOD) Connector**



# 63. Understanding Servo Registration

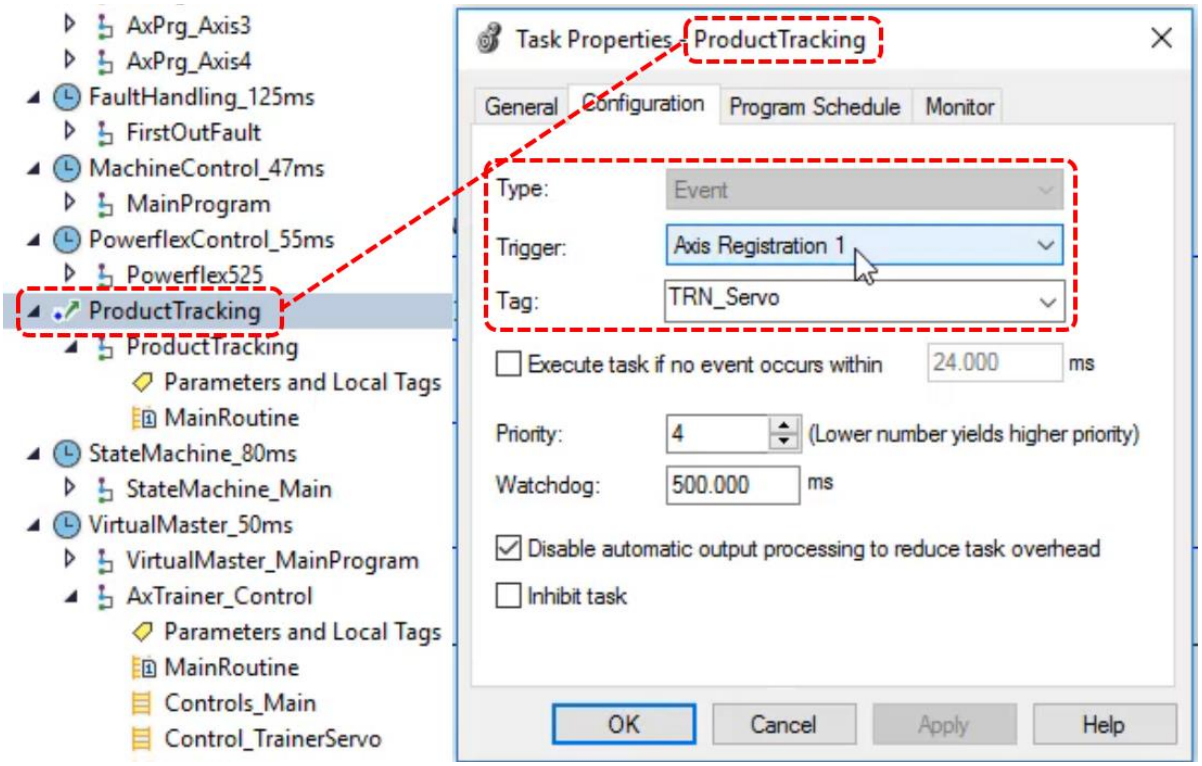
Here we will look at the Motion Arm Registration (MAR) and Motion Disarm Registration (MDR) instructions.




We can configure the routine to run on the event trigger "Axis Registration 1". TRN_Servo is the tag associated with the physical axis.

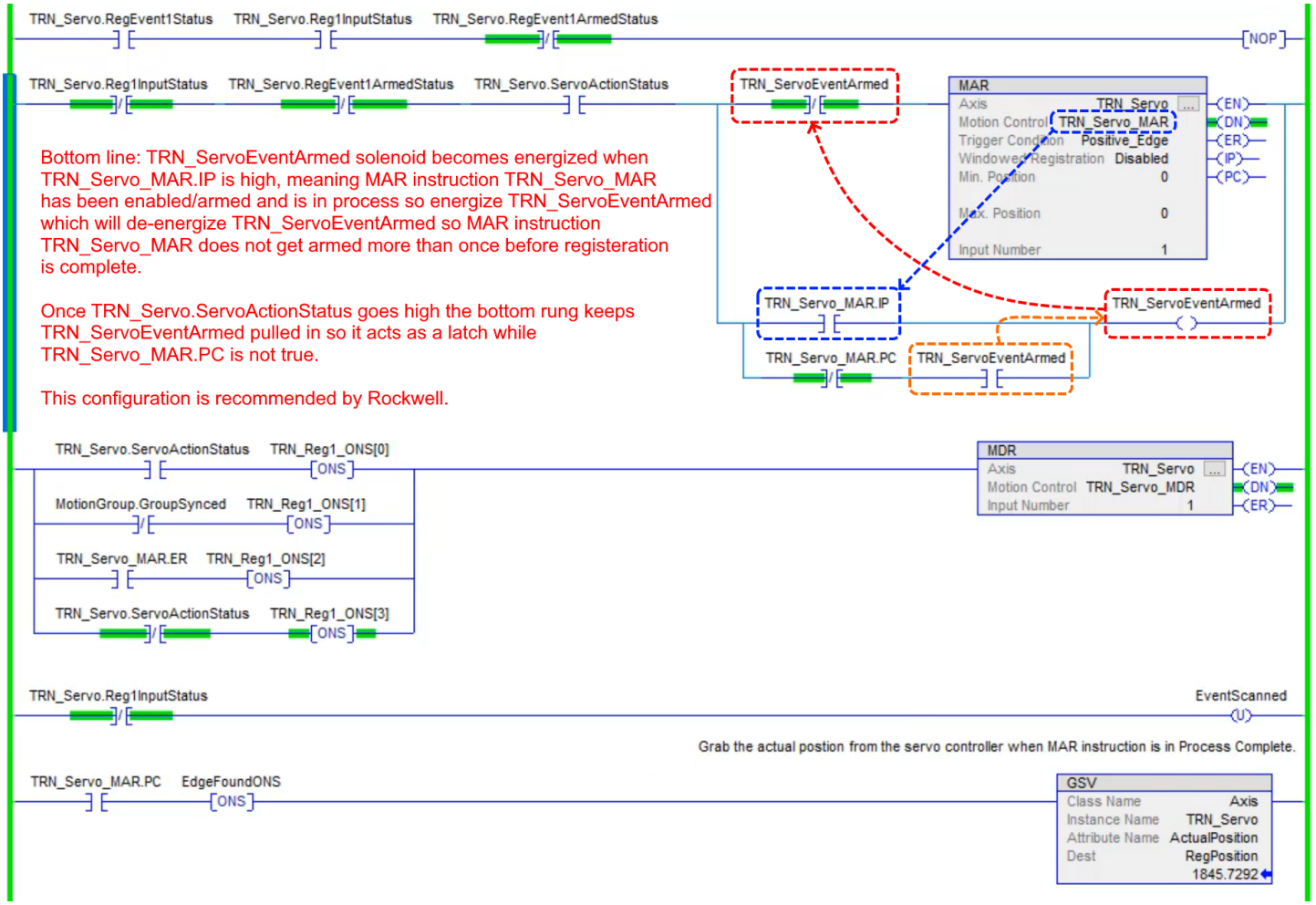
Again, **our routine triggers on the registration event.**



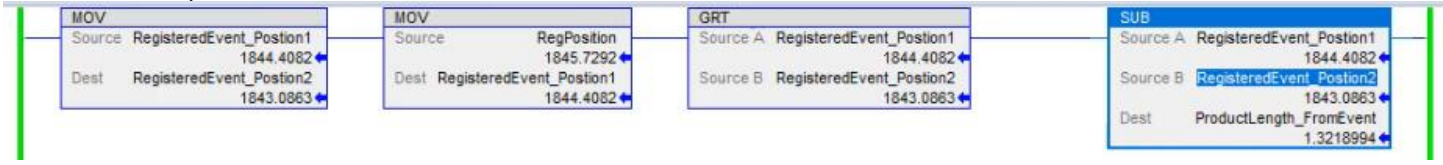
Key point is that the MAR instruction has to be enabled/re-armed each time you have to use it. This arming action tells the drive to monitor the registration bit more frequently, assigns it to a task running at a higher frequency. So trying to use the registration input when not armed is dicy at best, could miss the signal. **Arm the MAR instruction each time you need to use it.**

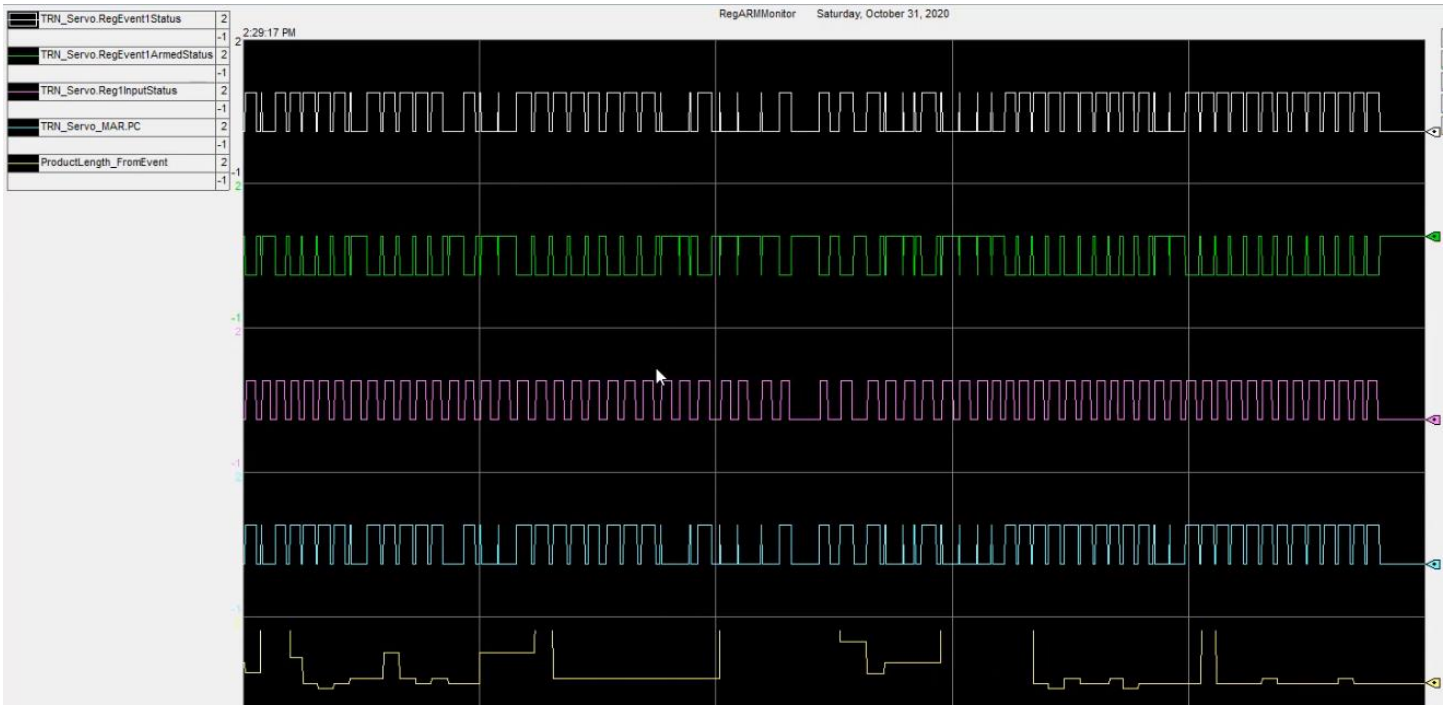
The MAR IP bit has to register each time you use the MAR instruction or the next time you try to use it the function will not preform properly. Register IP, register DN, and then re-arm for next event.

## 64. Length Detection Example (A deeper look into Registration)

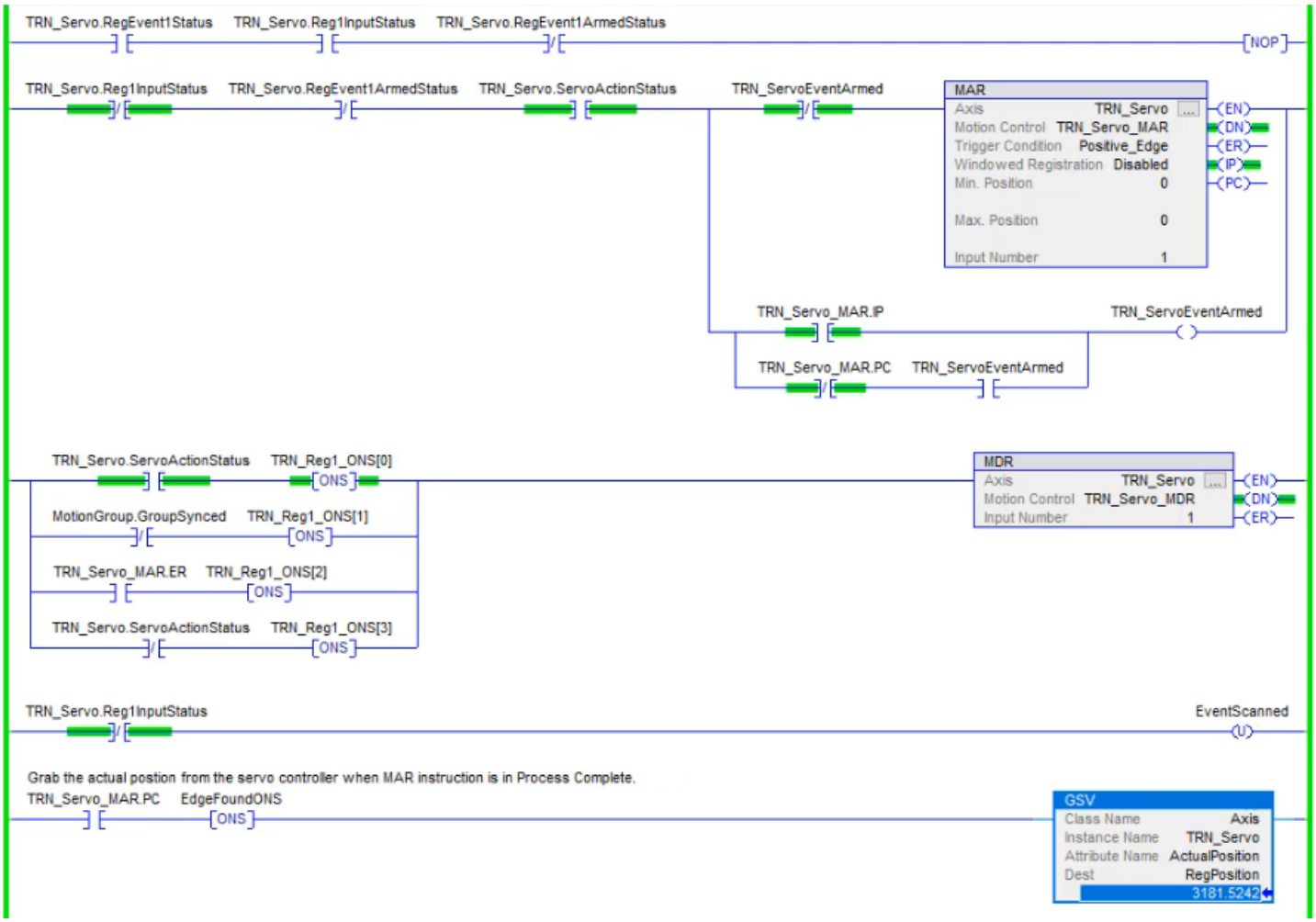



Bottom rung: we get the servo position, which at this point is the registration position, when TRN_Servo_MAR activity is complete. We will use this value in the product tracking routine, this is an event driven task on the registration input of the servo we are running. The idea here is we using this value in a calculation to determine the product length (this is an oversimplified example, an actual length measurement application would be much more involved).

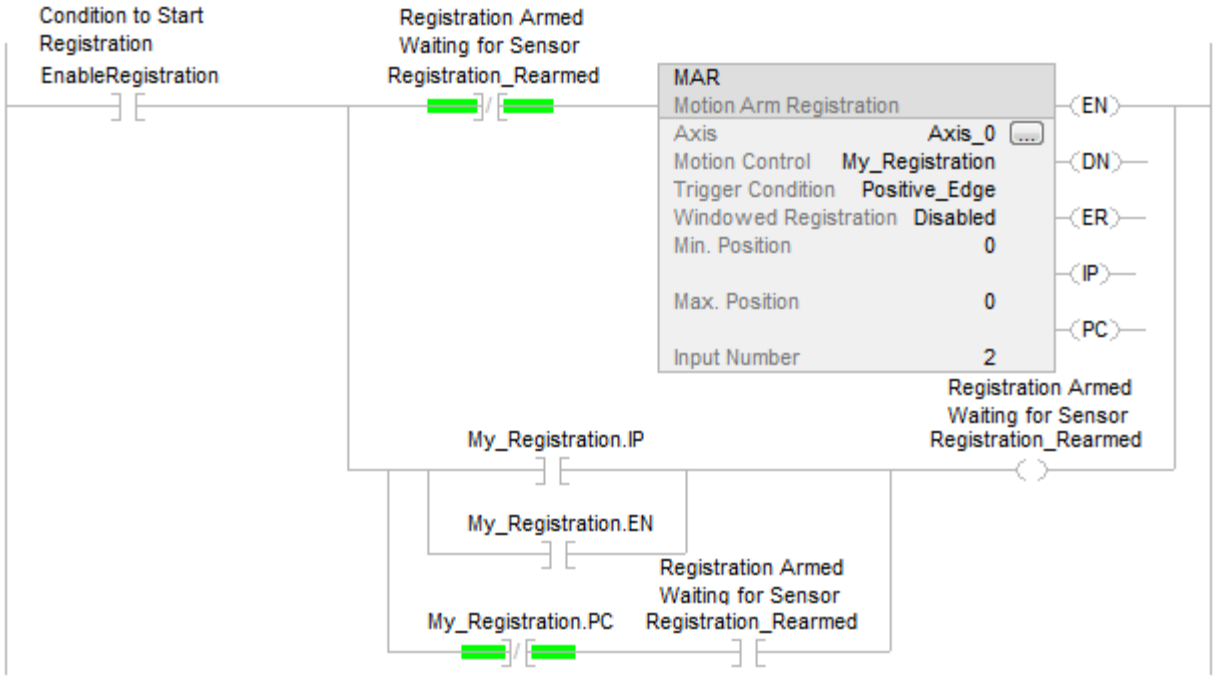




Trend of the length measurement application.



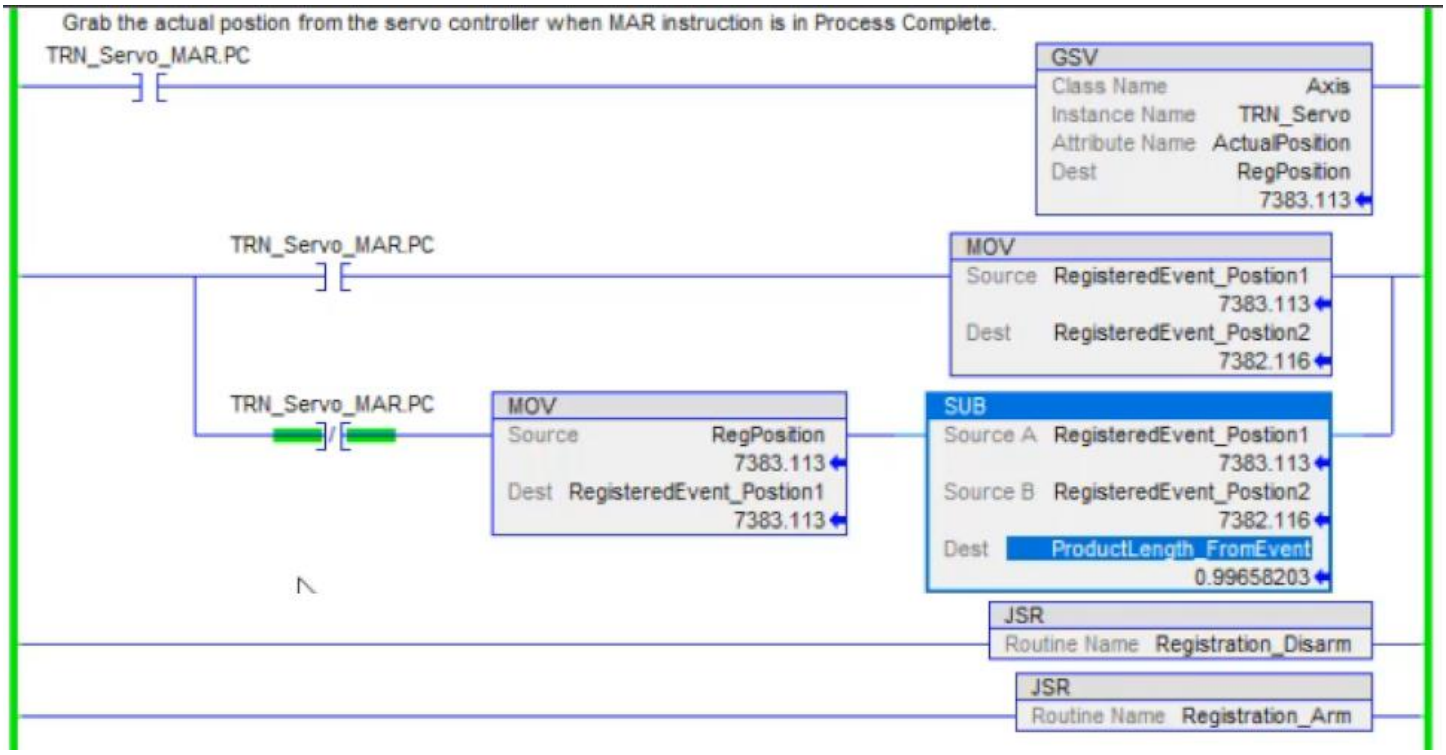
Length measurement code.  This is not the entire application.  Takes position each time .PC comes on.




RW: If your application requires rapid and continuous detection of a registration sensor, **we recommend using this logic**.

## 65 Servo Registration Using a Photo-Eye (Real-World Design)

The modified code below more closely suits a real-world application. The registration is now part of the mechanism, so we get consistent length even when we sow r speed the drive.




Improved application more closely resembling a real-world application.

Remember, event must be armed each time the input is going to fire.  And every registration application will be unique and reflect the real-world process.

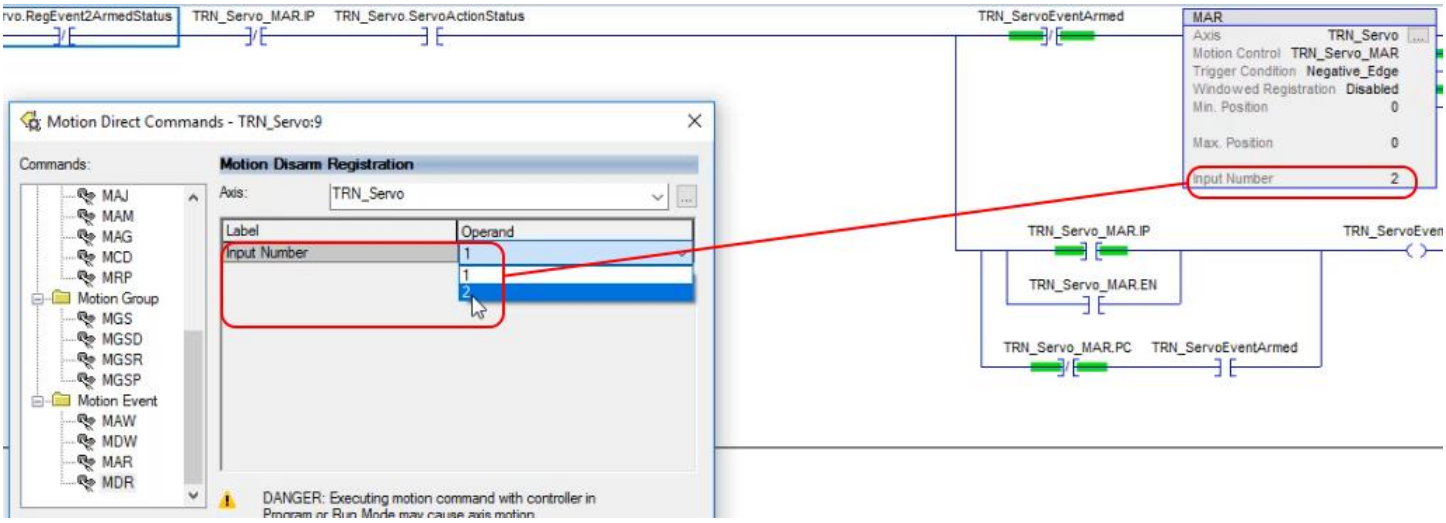
## 66. Servo Home Failed Because a Servo Event Armed

What happens if you try to home your servo and there is an armed registration?  You will get a fault.  You cannot home an axis which is in registration.

```
Motion Direct Commands - TRN_Servo:7, MAH, (16#0000) No Error
Motion Direct Commands - TRN_Servo:8, Execution Error.MAH, (16#0022) You are trying to start a Motion Axis Home (MAH) command while running a registration.
```

Here is a workaround using direct motion toolbox.

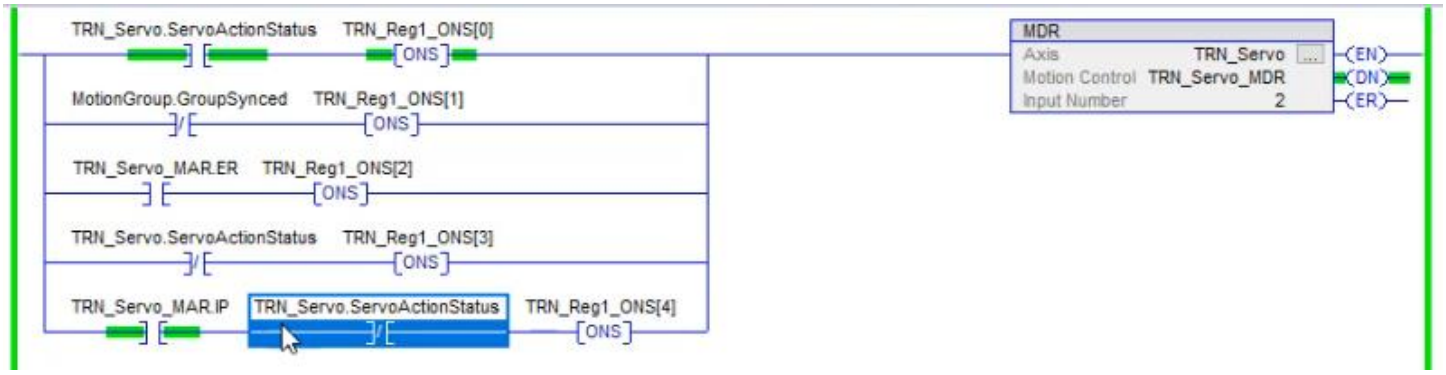
Right click on axis, select Motion Direct Commands.
Select Motion Disarm Registration (MDR) from the left hand pane tree.
Select the Input Number to correspond to the number being used in the MAR instruction.



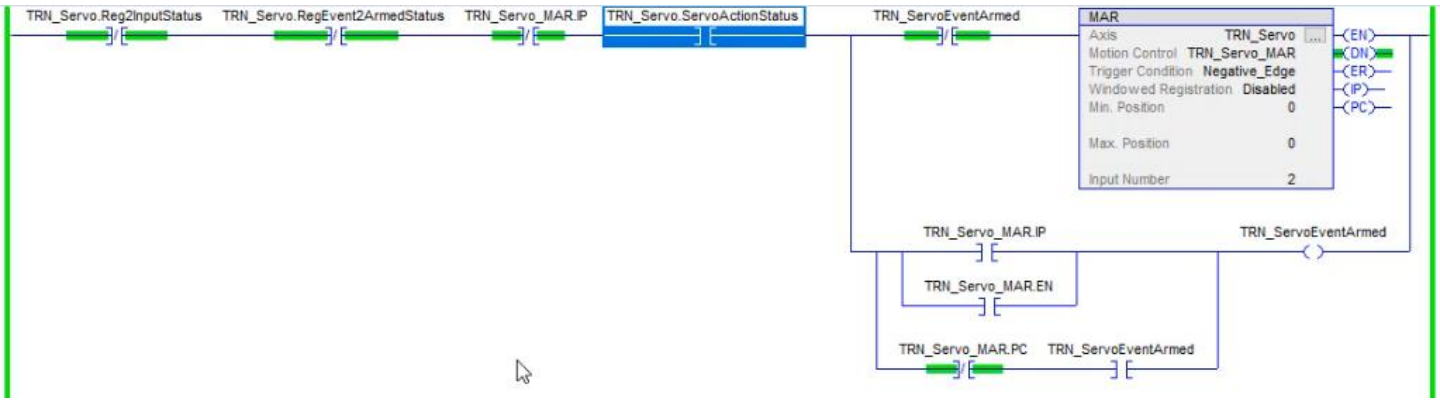

Click the Register button on the Motion Direct Command dialog.
Now select the MAH command in Motion Direct Command dialog and click on Execute.

Now, what is your registration position when you capture it?


It is essentially zero.




This code modification, and the one on the following page, fixes the problem with homing described above.




## 67. Closing Comments

He is discussing Advanced Servo Motion Mastery 2, a course which never came to fruition.