

**Subject:** MP2000 Best Practices Guideline

**Product:** MP2000 Controllers

---

**Status:** Rev 1.1

**Summary:** This document is a guide for use to implement Best Practices for programming design and architecture for an MP2000 controller system. The main topics discussed in this document are an overview of the design process, followed by three detailed topics of memory allocation, drawing use, and developing code for motion control.

## Format of Information:

The information provided in the topics discussed here are intentionally concise. The appendixes for the related topics provide information and examples for the user to reference if more detail is preferred. This document discusses the MP2000 Best Practices in the sequence of normal application development. The main sections are:

- Machine Information Gathering
- Selecting a Programming Method
- Program Architecture
- Memory Allocation
- Symbol and Commenting Naming Conventions
- Code Development

**While it is possible to control up to 256 axes, this document is based on up to 64 axes being utilized. For applications using more than 64 axes contact Yaskawa for technical support.**

<b>DOCUMENT REFERENCES:</b> .....	<b>3</b>
<b>MP MACHINE CONTROLLER PROGRAMMING BEST PRACTICES OVERVIEW</b> .....	<b>4</b>
WHY USE A YASKAWA MACHINE CONTROLLER? .....	4
WHAT'S IN MP PROGRAMMING BEST PRACTICES GUIDELINE?.....	4
BENEFITS OF USING MP BEST PRACTICES: .....	5
<b>GETTING STARTED – USING THE MP BEST PRACTICES GUIDELINE</b> .....	<b>7</b>
<b>MACHINE INFORMATION GATHERING</b> .....	<b>8</b>
<b>SELECTING PROGRAMMING METHOD</b> .....	<b>9</b>
OVERVIEW OF MOTION FROM MP CONTROLLER CODE TO AMPLIFIER.....	12
REGISTER INTERFACE.....	14
<b>PROGRAM ARCHITECTURE</b> .....	<b>15</b>
DRAWINGS DEFINITION .....	15
DRAWING FAMILY EXECUTION .....	15
DRAWING FAMILY HIERARCHY AND PROGRAM FLOW .....	17
BEST PRACTICE DRAWING USAGE.....	18
MP2000 BEST PRACTICE DRAWING ARCHITECTURE .....	19
<i>Drawing Architecture for A, H, and L</i> .....	20
<b>MEMORY ALLOCATION</b> .....	<b>22</b>
WHAT IS MEMORY MAPPING?.....	22
WHY IS IT IMPORTANT? .....	22
MEMORY MAPPING WITH FUNCTION BLOCKS .....	22
D REGISTERS FOR WORKING MEMORY .....	23
AUTOMATIC ADDRESS ALLOCATION.....	24
<i>Advantages</i> .....	24
<i>Disadvantages</i> .....	25
SYMBOL NAMING & COMMENTING CONVENTIONS .....	26
<i>Purpose of naming convention</i> .....	26
<i>Recommendations</i> .....	26
<b>DEVELOPING CODE</b> .....	<b>27</b>
LADDER TECHNIQUES .....	27
<i>Low Scan Interlocks</i> .....	27
<i>Purpose of Interlocking HMI</i> .....	33
<i>Machine Interlocks</i> .....	34
<i>Gearing</i> .....	37
<i>Waterfall Technique</i> .....	40
<i>Modulus Technique</i> .....	42
<i>Delta Scan</i> .....	44
<i>Handling Rollover</i> .....	45
<i>Sequencing Techniques</i> .....	46
RULES FOR MOTION PROGRAMMING .....	48
<i>Initiating Motion Program</i> .....	49
<i>Interlocking During Execution</i> .....	51

**DOCUMENT REFERENCES:**

Following is a list of documents used or referenced to in MP Best Practices

- eng.MCD.05.055 MP2000 Best Practices Guideline (.pdf)
- eng.MCD.05.096 MP2000 Best Practices Information Gathering (.pdf)
- eng.MCD.05.097 MP2000 Best Practices Global Memory Registration Map (.xls)
- eng.MCD.05.098 MP2000 Best Practices Motion Language Sample Code (.mal)
- eng.MCD.05.099 MP2000 Best Practices SVB Ladder Template Code (.mal) (*in development*)
- eng.MCD.05.031 Basic Design (.ppt)

## **MP MACHINE CONTROLLER PROGRAMMING BEST PRACTICES** **OVERVIEW**

### **Why Use a Yaskawa Machine Controller?**

General automation machine controllers are beneficial in that they offer a unique flexibility for programming the operation of functions and sequences within a machine. Whether the application warrants high performance point-to-point indexing, discrete motion operations, continuous motion operations, or complex scan based motion profiling with on-the-fly phase shifting – all benefit from tightly coupled motion and machine sequence logic control. As factory floor production requirements increase, machine cells must also increase their own production capability, and precise control of complex motion becomes vitally important to insure production rates are met and continue high quality product yield.

Many independent PLCs and Motion Controllers reach a limitation when programmers strive to achieve these new and improved line production performance benchmarks, and unfortunately result in a bottleneck when trying to increase the performance of a system. This bottleneck can often be caused by control systems latencies in data handshaking or lack of synchronization, not only from cell controller to cell controller, but also within a single controllers functional module interfacing. A good machine control platform tightly couples high performance motion control capabilities with machine sequence logic, and provides the foundation and scalability necessary to achieve performance in a changing environment.

True machine control tightly integrates the following: high and low performance features, priority setting and adjustment, determinism and predictability, synchronization of tasks / processors / networks, high execution rates, scalability without sacrificing performance, integration of peripheral components, and a focus on precise motion control. The result is smooth motion, less machine jerk, less mechanical vibration, less heating, smaller and more efficient motor sizes, higher machine reliability and longer life cycles – all with the advantage of built in future expansion.

To take advantage of the power of total machine synchronization and extend this capability across a wide range of performance requirements - from low performance, low cost machines to high performance, higher priced machines, a scalable machine control platform is offered by Yaskawa: the MP Series Machine Controller. To harness the power of flexibility and performance while maintaining an overall ease of use, Yaskawa also introduces the MP Machine Controller MP Programming Best Practices Guideline.

### **What's in MP Programming Best Practices Guideline?**

This document is a guideline to lead the controls engineer down a successful path that will not only increase programming productivity for any given machine, but also an entire line of complementary machines which can be offered for different user requirements. It outlines and encourages the use of best practices by some of the top motion and machine control engineers of the MP product line, including its predecessors. This document helps the programmer to realize the benefits of following best practices:

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

## Benefits of Using MP Best Practices:

Benefits can be realized at all levels of machine development from design to implementation to field support. In summary, all benefits work towards lowering the total cost of ownership in the machine's product lifecycle.

### - Design Level Benefits

- *Programming Method Selection:* A flexible machine control allows several ways to program motion control. This guideline helps the programmer select the best programming method for a given application type.
- *Standardized Model:* This guideline helps the programmer develop a robust framework and controls architecture, and secures a solid platform for code development. In addition, within the guideline are several examples of standardized programming methodologies that leverage template examples.
- *Step-By-Step Basic Design:* There's a higher probability of getting it right the first time if a step-by-step approach is followed, reducing the possibility of missing important steps along the way. This guideline provides a step-by-step flowchart of machine development features.
- *Risk Reduction:* A higher percentage of completing development projects on time, and on budget will result through the use of MP Programming Best Practices.

### - Implementation Level

- *Pre-Defined Code & Scalability:* This guideline provides pre-defined code and canned functionality recommendations, which in turn reduce development and debugging time. More efficient code development can be realized, which affords more time to be spent on process related issues.
- *Optimized Performance & Robustness:* Code efficiency is dependant upon good implementation of well thought out architecture. If the code is written right, performance of the control system can be optimized resulting in lower scan times, increased motion performance, and increased performance of device interfacing. This guideline encourages optimized performance in implementation resulting in robust code that works.
- *Organization:* This guideline makes programming easier because features are compartmentalized for streamlined implementation. An organization layout is provided for memory mapping, variable/tag usage, alarm interlocking, program flow, axis control management, I/O usage, etc.

- Support Level

- *Comprehensive Training:* Using Best Practices supplements Yaskawa's product training, allowing the skills learned to become better utilized.
- *Transferable Skills:* Using best practices across multiple machine designs, reduces risk by taking advantage of architecture knowledge acquired on previous designs.
- *Common Look and Feel:* Using commonly defined, standardized methods will make the system easier to support when the machine is in production. Tech support engineers can leverage the same best practices. Maintenance and troubleshooting become easier on a global basis. MP Best Practices is not meant to replace open standards, but rather to compliment standards such as OMAC PLC OPEN "function blocks for motion control".

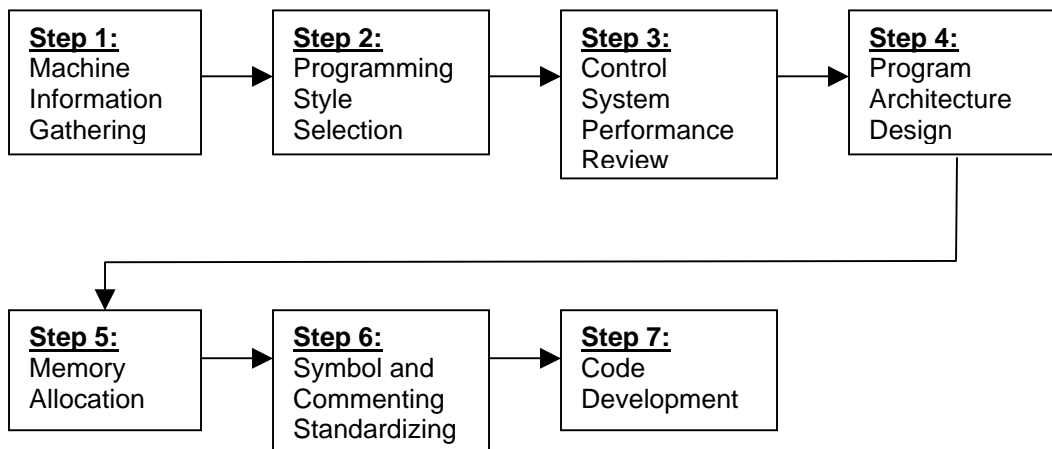
## **GETTING STARTED – USING THE MP BEST PRACTICES GUIDELINE**

This Best Practices Guideline is designed to help the controls engineer better utilize the strengths of Yaskawa technology, allowing them to leverage the unique flexibility of functionality that the MP2000 platform offers, and best align the implementation with the specifics of the application requirements. Understanding the control systems benefits and constraints early in the design process results in a higher probability of successful machine design when merging the controls programming with the application process, lowering risk and overall rework.

This guideline will serve as a machine controller design and development methodology, presenting program architecture and memory mapping recommendations, programming recommendations, as well as actual code examples and pre-defined templates to get jump started. It also provides experienced controls engineers with a benchmark to compare against, allowing them to make more informed decisions between known practical solutions, and new, unique ideas of a better method.

First time users will find this MP Best Practices guideline as a useful tool and guide them through a recommended step-by-step procedure for controls development on for an automated machine with tightly integrated motion control. More experienced users will find this MP Best Practices guideline as a useful reference tool for specific techniques and code modules that can be readily used. This guideline is formatted to provide the specific techniques, with a Step-by-Step framework.

The following diagram illustrates the recommended step-by-step procedure when developing a performance machine control system.

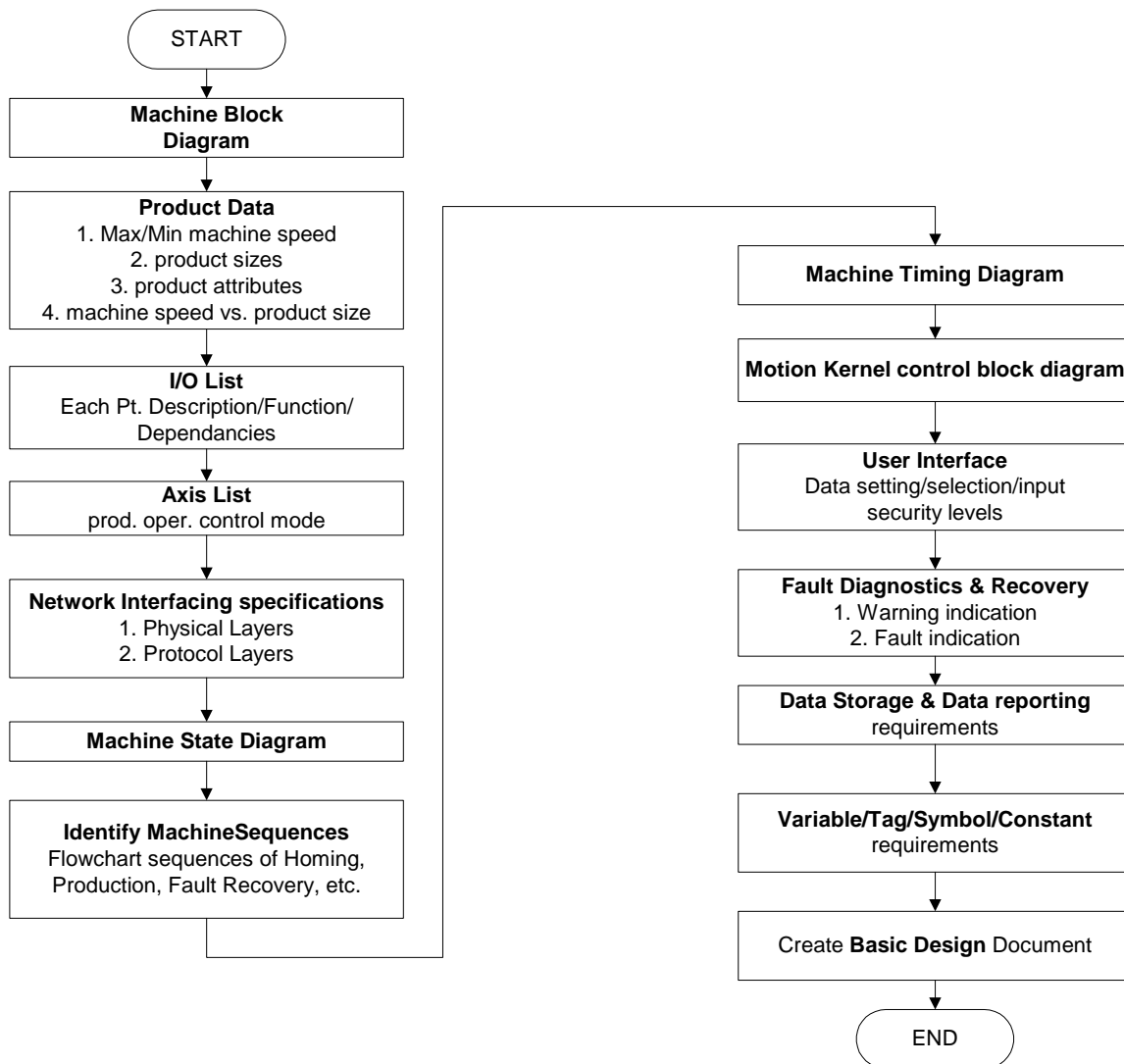


Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

**MACHINE INFORMATION GATHERING**

In order to implement a structured, reliable and efficient machine control system, requirements of the machine should be gathered thoroughly in a formal manner to insure details of the system design or functionality are not overlooked. The sequences of steps for machine information gathering are shown in the diagram below. Note the sequence of the diagram is in logical development of the application. Brief descriptions of the information gathering steps and references to examples of the steps are located in eng.MCD.05.096, MP2000 Best Practices Information Gathering. Basic Design template example is also referenced in eng.MCD.05.031 Basic Design Template.

**Basic Design Steps**





After the requirements and functionality of the machine are documented, the controls programmer is then able to evaluate the optimum methods of implementing machine and motion control. The machine block diagram, sequences, timing diagram, and motion kernel control block diagram will be key in evaluating the programming method to use in the application.

## SELECTING PROGRAMMING METHOD

Use the table below to assist in deciding the proper motion control programming implementation based on the type of machine being controlled and the motion required.

**Machine Complexities versus Programming Methods**

MACHINE COMPLEXITIES						
		Motion	Point to Point	All co-ordinated	scan based profile generation	Interpolation
P M o t i o n P r o g r a m	Function Blocks		O	O ##	X	X
	Motion Programs		O	O	**	O
	Ladder		O	O	O	O*

### KEY

O Applicable Method

X Not Viable

It is possible to mix motion programming methods in a controller, but it is recommended to run a axis by only one method.

## Axis count dependent

\* Consider method with caution

\*\* Any interpolation instructions generates scan based profile

After evaluating programming methods in relation to machine complexity, the user should evaluate the individual programming methods based on overall system constraints such as quantity of axes, programmer expertise, and overall system performance criteria. These parameters are summarized in the table, Programming Language Grid For High Level Topics on the following page.

**Programming Language Grid For High Level Topics**

	Ladder Based Motion	Text Based Motion Programs	Function Block Motion
<b>High Level Topics</b>			
Required Programmer Expertise Level	most flexible	easy	easy
Performance	<b>highest</b>	medium	high
Maximum axes	<b>Only Limited by Hardware</b>	(16 / group)	16
Scan time	<b>most efficient</b>	Motion programs synchronize with H Ladder	additional scan time over head
Troubleshooting	<b>Register based</b>	Register based	A function block handles this
Memory allocation	<b>Register based</b>	Register based	reserved data area required

Note: This table is a broad overview perspective.

Finally, consider the specific motion control functionality required. Individual, built in function detail is evaluated for the three programming methods listed in the table, "Programming Language Grid For Built In Functions" on the following page.

## Programming Language Grid For Built In Functions

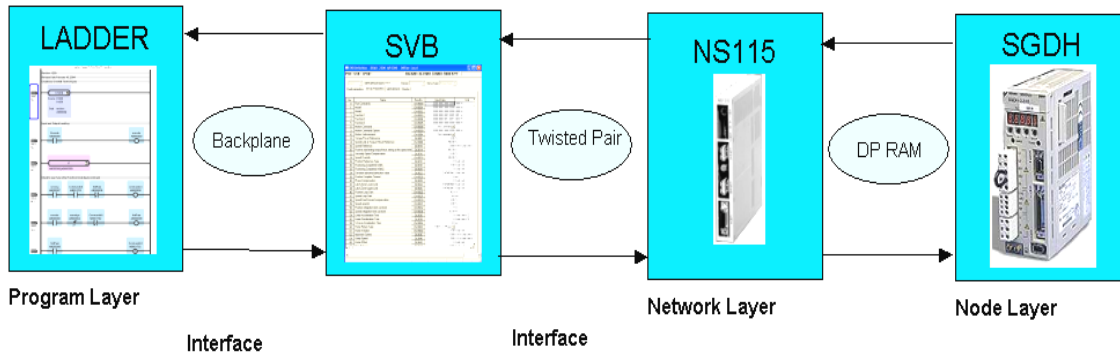
	Ladder Based Motion	Text Based Motion Programs	Function Block Motion
<b>Built in Function Detail</b>			
Jogging	possible	possible*	<b>possible</b>
Absolute or Relative Indexing	<b>possible</b>	possible	possible
Change final position on the fly	<b>possible</b>	not possible	possible
Linear interpolation	possible	<b>possible</b>	not possible
Circular/Helical interpolation	possible	<b>possible</b>	not possible
Homing	<b>possible</b>	possible	possible
Basic Camming	possible	not possible	<b>possible</b>
Advanced Camming (including master/slave shift or offset on the fly)	possible	not possible	<b>possible</b>
Basic Gearing	possible	not possible	<b>possible</b>
Advanced Gearing with slave shift	possible	not possible	<b>possible</b>
Customized scan-based profiling	<b>possible</b>	not possible	possible
Change Acc/Dec on the fly	<b>possible</b>	not possible	possible
Change Velocity on the fly	<b>possible</b>	possible**	possible
Position Latching	<b>possible</b>	not possible	possible
Latch Target	<b>possible</b>	possible	possible
Hold/Resume Motion	possible	possible*	possible
Servo Enable	<b>possible</b>	possible*	possible
Fault recovery	<b>possible</b>	not possible	possible
E-Stop Recovery	<b>possible</b>	possible*	possible

\* ladder coding req.

\*\* with ladder code

To further understand and evaluate motion programming methods, consider the table below. This diagram illustrates how logic in the controller is translated to motion via the SVB components of a MP Controller system.

### Overview of Motion from MP Controller Code to Amplifier



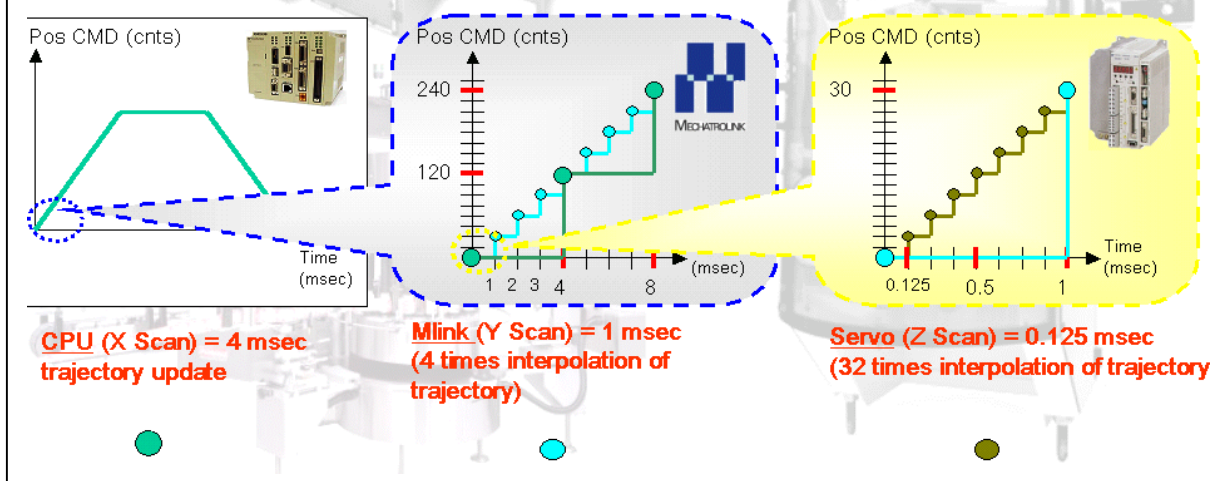
-The **Program Layer** can be broken down into High and Low scan priorities. Code efficiency determines the ultimate scan rate. The interface between the Program Layer and Network Layer is the SVB IF module. As the application program grows, performance of the segregated network layer module is not affected. This is a key point for performance!

-The **Network Layer** can be broken down into scalable scan rates and scalable data packet size. Scan rates are determined by the quantity of axes and quantity of network interface cards. Above eight axes, the system operates at a 2msec update rate, even for hundreds of axes. Data sent down to the network card is further interpolated at the servo update rate. The interface between Network Layer and Node layer is MechatrolinkII IF.

-The **Node Layer** can be broken down into Servopack, I/O, and Inverter/ VFD nodes. For servo motion control, the speed of the position loop update and local interpolation between commands promote high performance, and ability to control any of the following: Rotary, Direct Drive, and Linear Motors. The load, coupling, and transmission mechanics further determine overall performance.

**All Layers in a System**

- Interpolation Level 1: CPU application code to SVB M-Link (@ rate of M-Link network)
- Interpolation Level 2: M-Link to Servopack (@ rate of Servo Loop 125 microsec)
- Here is an example: 4msec prog scan, 1msec Mlink network scan, 125 microsec servo position loop scan



**Example: CPU X Scan = 4 msec (MP2300)**  
**Network Y Scan = 1 msec (8 axes of servo)**

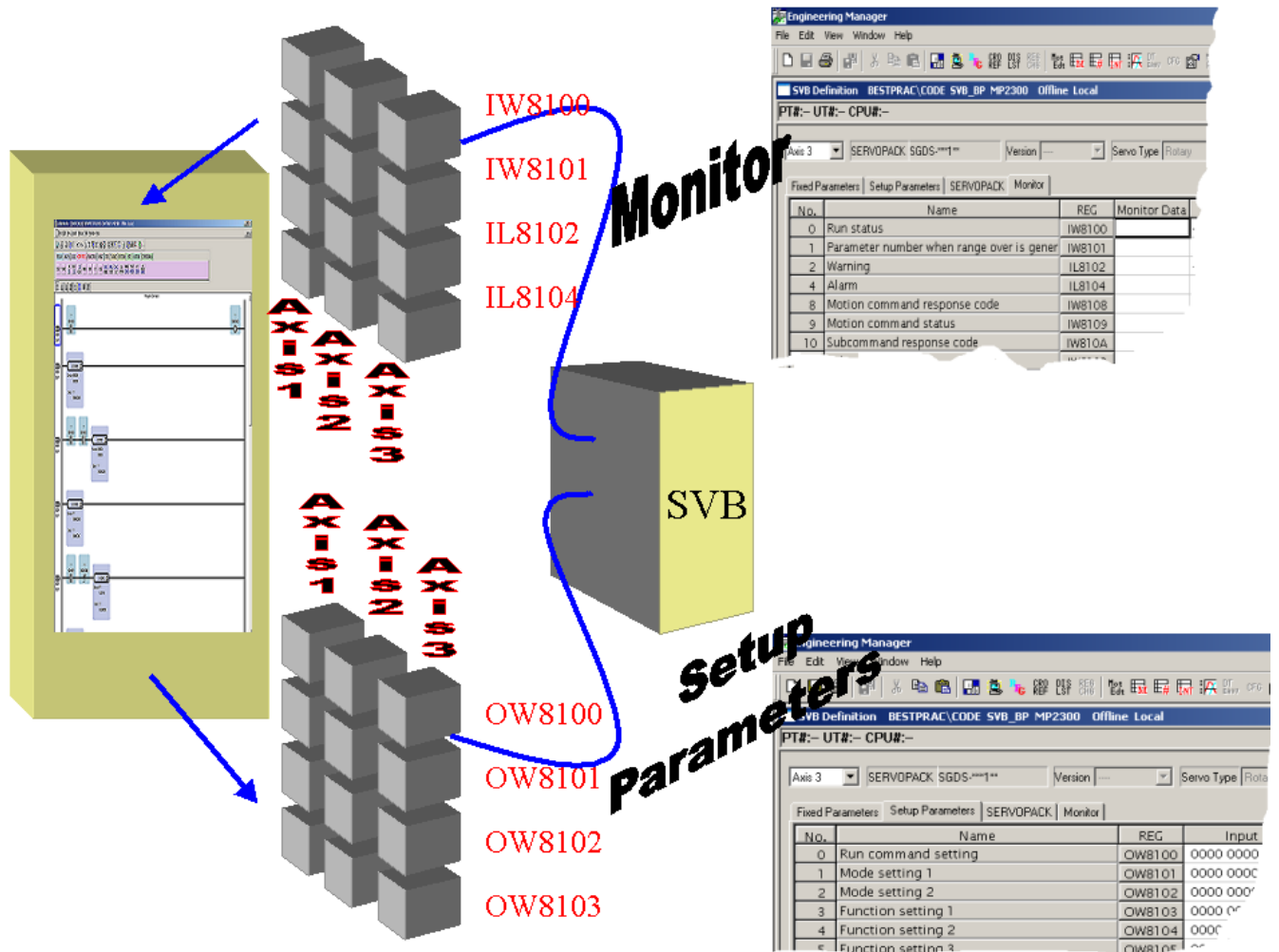
Final performance note for all layers in a system:

The data being transferred to the network devices is based on the 'motion and I/O registers.' All registers are updated at the end of the Program Layer scan at the same time, and the interface between each layer synchronizes to bring about system determinism. Hence, the performance is predictable. Network delay can be fed forward at a certain resolution to reduce steady state error of high-speed applications. This is key point for maximum performance!

At the programming level, the user interfaces to the motion system via Monitoring Parameters for feedback and Setup Parameters for commanding motion. Each axis has registers associated with it. The user application manipulates these registers either directly in Ladder Based Motion or indirectly using Motion Programming or Function Blocks.

## Register Interface

The application program interfaces to the motion network via register interface. These registers are updated each high scan of the processor.



Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

## **PROGRAM ARCHITECTURE**

After evaluating programming methods based on the system requirements, the user is ready to design the program architecture. The program architecture is designed by locating logic in the most appropriate drawings in either the high or low scan, based on the application requirements. Before specifying specific drawings based on function, it is key to understand drawings, their execution, and hierarchy in the MP Controller.

### **Drawings Definition**

Application programs for MP2000 controllers are created using modular sections of ladder logic code called “drawings.” There are four families of drawings: A, I, H and L, organized in a generational hierarchy and denoted in the following chart.

	Family A	Family I	Family H	Family L
Parent	A	I	H	L
Child	Axx	Ixx	Hxx	Lxx
Grand child	Axx.xx	Ixx.xx	Hxx.xx	Lxx.xx

Valid drawing names must adhere to the above format with xx = 01-99. The maximum combined number of Child/Grandchild drawings differs for each ladder family as follows:

A = 62  
I = 62  
H = 198  
L = 498

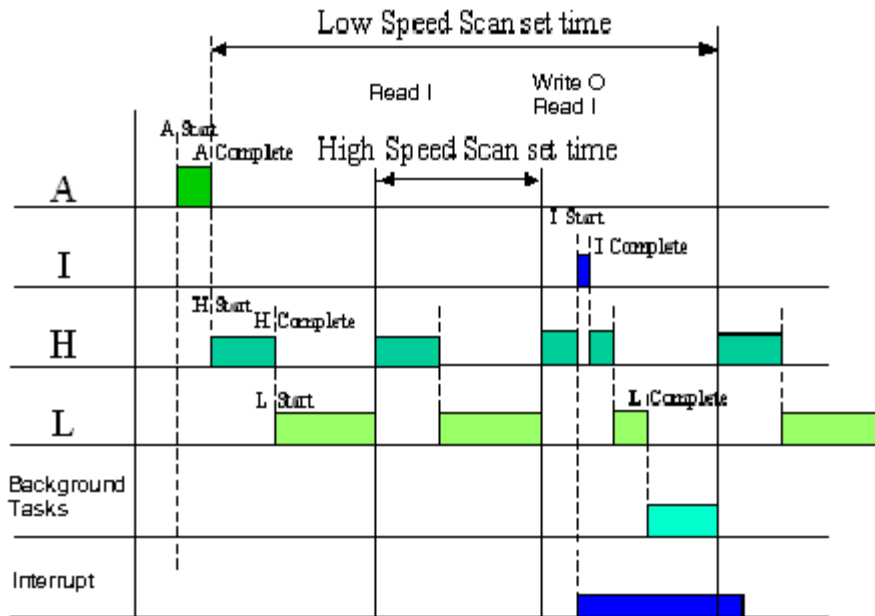
### **Drawing Family Execution**

Each drawing family executes at a unique point in the overall program scan, offering the user the opportunity to optimize system performance. The following charts demonstrate the priority given to each drawing type and also show an example of actual program execution. The MP controllers allow the user to set the High and Low Speed Scan intervals.

Drawing Family	Function
A (Power-up)	Executed only once upon power up.
I (Priority 1)	Executed once at the rising edge of an interrupt input signal, 1 <sup>st</sup> input on a bank of LIO or CPU I/O input 1
H (Priority 2)	Executed once every High speed scan interval
L (Priority 3)	Executed once every Low speed scan interval.

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

**Drawing Execution Chart**



The above drawing execution chart graphically illustrates the Drawing Family and Function table from the previous page.

- At power up, the A drawing executes AND completes before other drawings execute. The A drawings execute once only at power up.
- The high scan drawings always complete within the high speed scan set time as illustrated above.
- In the remaining time of the high speed scan, the low scan drawings execute in time slices until they complete execution within the low speed scan set time.
- At the completion of the low scan drawings, background system tasks are executed.
- If an interrupt condition occurs, the I drawing executes immediately, then normal scanning resumes.
- Setting Parameters and Monitor Parameters are updated each high speed scan while low scan I/O updates occur every low speed scan set time.

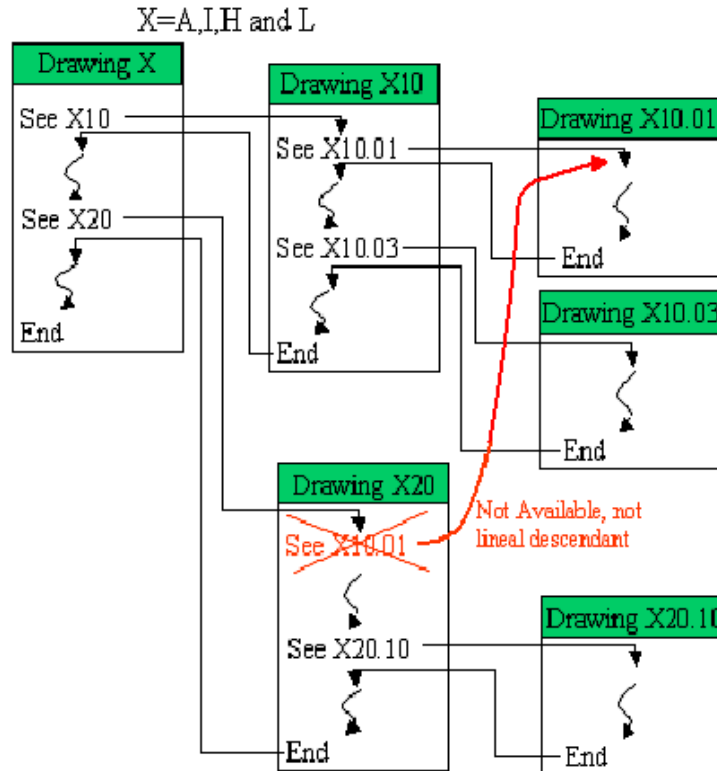
The update of I/O and parameters as well as understanding drawing execution is critical to code development so that the proper interlocks exist to obtain expected logical results.

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280



## Drawing family hierarchy and program flow

The parent drawings begin execution automatically upon their appropriate internally generated timing signal. Child and Grandchild drawings are programmatically called using the SEE instruction. Calls to a descendent drawing must be contained with the generational hierarchy, so only a linear descendent can be called. The figure below shows the execution flow of a drawing family. "X" represents a drawing family, which can be replaced with A, I, H or L.



Using a conditional "SEE" instruction is not recommended. This could result in problems such as leaving an output in an undesirable state or interfere with one-shot actions.

After reviewing the drawing types, execution and hierarchy, the user is now ready to examine the use of drawings for the machine control application.

## Best Practice drawing usage

### A-Drawings

This drawing family is best used for initial system setup since it is executed only once upon power up. **WARNING:** A-Drawings do not re-run after a CPU RUN/START from a CPU stop state.

### Clear Memory Registers

MP controllers retain the state of M registers with a battery. All memory register states are retained through power loss. Safety reasons dictate that machine states such as Auto/Manual and Run/Stop should be initialized to a desired value to avoid dangerous situations should a power outage occur.

### Initialization Function Blocks

RDAINIT1 and RDAINIT2 functions blocks are placed in this drawing family to setup Function Block programming environment.

It is recommended to set the registers to zero in the H-drawing using s system bit that comes on only for the first scan. Best practice is to set constant values into registers using a Low Speed L drawing (L40 in particular).

### I-Drawings

This drawing family is only used for time critical operations. Thanks to recent improvements in CPU processing speed, the High Speed Scan Interval is usually fast enough to handle most applications. You can generally avoid using this family.

### H-Drawings

The recommended main drawings are:

- H10: Common machine high-speed processes.
- H20: Machine axes control.
- H30: Machine high-speed auxiliary devices.

Additional drawings:

- H15: Axes trajectory calculation for master-slave processes.
- H25: Motion programs control.

This drawing family is best used for the time critical processing of the application. All code directly pertaining to motion control should be placed here.

Examples include:

Direct motion register control  
Motion critical Function Blocks  
Text-based Motion Programs calls.

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

Master encoder position handling for GEAR and CAM Interpolated  
motion scan-based trajectory calculation  
High speed I/O processing (example: PLS)

## L-drawings

The recommended main drawings are:

- L10: Common machine low speed processes.
- L20: Machine axes sequencing and interlocks.
- L30: Machine low speed auxiliary devices.
- L40: Machine constants setting.

Additional drawings:

- L01: HMI communication.
- L25: Motion programs control.

This drawing family is used for Non-time critical processing of the application.  
Most general machine control and logic sequencing should be placed here.

Examples include:

- HMI data handling
- Machine operation mode (Auto/Manual)
- Axis operation sequencing (Cycle 1 / Cycle 2)
- Pneumatic valve operation
- Conveyor operation (relatively constant speed)
- Constant value setting
- Auxiliary equipment control
- Low speed I/O processing

## MP2000 Best Practice Drawing Architecture

A key advantage of Best Practices drawing architecture is organization. Programming and debug is simplified by compartmentalizing functionality. Code efficiency is improved by well-designed architecture allowing for lower scan times and increased performance. Modularized code enables scalability and re-use of code segments. Most of the code is contained in the grandchild drawings.

### General Architecture Outline

- Servo axis high scan drawing executes only the motion code, equivalent low scan axis drawing specifies mode.
- H11-H19 can be used for high speed functions or calculations with H15 normally being reserved for trajectory calculations.
- Make an "Always Off" bit in the first rung of the A, H and L parent drawings. This is useful if an "Always Off" parameter input to a function is required
- Pushbutton and HMI logic sequencing should be in low scan.
- H25 can optionally be organized by Motion Program number as opposed to Group number.
- The maximum number of axes to follow the suggested architecture layout is 64.

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

*Drawing Architecture for A, H, and L*

### **Initialization Drawings (A)**

- A** Parent for initialization drawings, Memory Clear, Memory Clear and other settings
  - A01**
- A20** Axis Initialization if using Function Blocks
  - A20.01** Main initialization drawing Axis 1- RDAINIT if using FB
  - A20.02** Main initialization drawing Axis 2- RDAINIT if using FB

### **High Speed Scan Drawings (H)**

- H** High Scan Parent, calls child drawings
  - H10** Machine common high-speed processes (cam master, PLS master)
    - H10.01** Machine common high-speed process 1 (ex. cam master)
    - H10.02** Machine common high-speed process 2 (ex. PLS master)
  - H15** Reference output calculations (Axis trajectory)
    - H15.01** Axis #1 Cam target position
    - H15.02** Axis #2 Gear target position
  - H20** Individual axis motion controls for MCC or FB methods – call grandchild drawings for individual axis
    - H20.01** Axis 1 High Speed motion control commands or function blocks
    - H20.02** Axis 2 High Speed motion control commands or function blocks
  - H25** Group motion controls for Motion Program Language method – call grandchild drawings for each motion program
    - H25.01** Motion program control for Motion Group#1 (or optionally MPM001)
    - H25.02** Motion program control for Motion Group#2 (or optionally MPM002)
  - H30** High-speed Auxiliary machine devices (ex. PLS outputs, diverter gates)
    - H30.01** High speed Aux. device 1 (Glue Head #1)
    - H30.02** High speed Aux. device 2 (Diverter Gate #1)

### Low Speed Scan Drawings (L)

- L** Low Scan Parent, calls child drawings
  - L01** HMI interface – put communication code here if necessary
    - L01.01** HMI to MP Controller Conversions
    - L01.02** MP Controller to HMI Conversions
  
  - L10** Machine common ladder sequencing (machine control states)
    - L10.01** Machine Common Ladder sequence (Control Mode [Auto vs Manual], All Axes Normal, All Stopped, All Axes Enabled, Home, etc)
    - L10.02** Sequence A Motion Logic
    - L10.03** Sequence B Motion Logic
  
  - L20** Individual axis sequencing for MCC or FB methods (servo axis, drive roll)
    - L20.01** Axis 1 individual sequencing (Axis Normal, Servo On, Jog+, Jog-, Low Speed motion control, etc.)
    - L20.02** Axis 2 individual sequencing (Axis Normal, Servo On, Jog+, Jog-, Low Speed motion control, etc.)
  
  - L25** Motion Program sequencing and interlocking for MPL method (combined groups of axes)
    - L25.01** Group #1 motion sequencing and interlock (Low speed Ladder sequence to control the group or Motion Program)
    - L25.02** Group #2 motion sequencing and interlock (Low speed Ladder sequence to control the group or Motion Program)
  
  - L30** Low speed auxiliary machine devices (on/off devices, fan, pump, air cylinder)
    - L30.01** Aux. device 1 (pump#1)
    - L30.02** Aux. device 2 (fan #1)
  
  - L40** Machine Constants
    - L40.01** Axis 1 Constants
    - L40.02** Axis 2 Constants

Now the systems functional specifications are documented, motion-programming methods are selected in addition to the program architecture. With the control application outlined, it is now viable to accurately allocate memory for coding the application as discussed in the next section.

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

## **MEMORY ALLOCATION**

### **What is Memory Mapping?**

Memory mapping is the recommended layout of MP2000 memory. Memory mapping defines the layout for both global and local registers and is intended to keep the data as organized as possible. A useful tool is available to define memory, refer to document eng.MCD.05.097 (Best Practices Global Memory Registration Map).

### **Why is it important?**

Memory mapping is important because it provides a consistent layout that increases program flexibility, and reduces development and troubleshooting time

Memory mapping increases program flexibility for many reasons. Grouping information together can be beneficial for many reasons, such as:

- 1) Ease and speed of transfer (HMI)
- 2) Cam tables
- 3) Recipe tables

Troubleshooting time is reduced when a consistent memory map is followed; errors are easily diagnosed since the range of registers to be monitored is localized. Troubleshooting time is also reduced because as the user programs additional applications using a consistent memory map, the user gains familiarity with the layout and is able to navigate the MP2000 memory faster.

Lastly, having an organized and easy-to-navigate memory map also helps to reduce development time. Using a consistent memory map eliminates the setup time required to layout the memory, as well as providing a proven and robust memory layout to eliminate confusion and reduce programming errors.

### **Memory mapping with function blocks**

When using functions blocks, the function block RDA (reserved data area) is MW30000-MW65535, meaning that these registers are reserved for function blocks and cannot serve the user in any other capacity. The allowable range for user addresses when using function blocks is MW00000-MW29999.

## D registers for working memory

Global memory should be reserved only for data that is used by multiple drawings (or additional components such as an HMI). Data only used by one drawing should be kept in the D registers of the specific drawing to minimize global memory usage.

The MP controller has seven types of memory. Proper allocation of this memory will greatly simplify programming, debugging, maintenance, and transferability of a given project. Memory can be allocated automatically or manually. A brief explanation of memory types and their usage will assist the user in determining the proper allocation method to use. Below are the types of memory available; followed by the abbreviation used in parenthesis, physical size, and a brief description of their usage.

- 1) Global memory registers
  - a. Data (M) 64k  
"M" registers are shared by all drawings. Used as interfaces between drawings. Register number nnnnn is expressed as a decimal number.
  - b. System (S) 8k  
"S" registers provided by the system. Register number nnnnn is expressed as a decimal number. When the system is started, SW00000 to SW00049 are cleared to 0
  - c. Inputs (I) (Physical/Motion) 64k  
"I" registers used for input data. Register number hhhh is expressed as a hexadecimal number.
  - d. Outputs (O) (Physical/Motion) 64k  
"O" registers used for output data. Register number hhhh is expressed as a hexadecimal number.
  - e. Constants (C) 16k  
"C" registers can be read only in the program. Register number nnnnn is expressed as a decimal number.
- 2) Local (Drawing Specific)
  - a. General purpose (D) up to 16k  
D registers are unique to each drawing and can be used only in the corresponding drawing. The user of the MPE720 specifies the actual range used. Register number nnnnn is expressed as a decimal number.
  - b. Constants (#) up to 16k  
"#" registers can be read only in the corresponding drawing. The user of the MPE720 specifies the actual range used. Register number nnnnn is expressed as a decimal number.

## Automatic Address Allocation

### What is it?

Automatic Address Allocation is a feature in MotionWorks that automatically assigns a register location to a user-defined symbol based upon the kind, data type, and tag information entered by the user.

“Symbol” – variable name assigned to a specific register

“Tag” – a user-defined label that can be used to group variables or registers

### How does it work?

MotionWorks maintains a register map of the symbol and data type assigned to each register. As new symbols are entered into the register list and assigned tags, MotionWorks places the new symbol in the lowest available address for that tag range. Users enter tag names and define the register range and scope (global or local to a specific drawing) for each tag name in the automatic address allocation section of Symbol Manager.

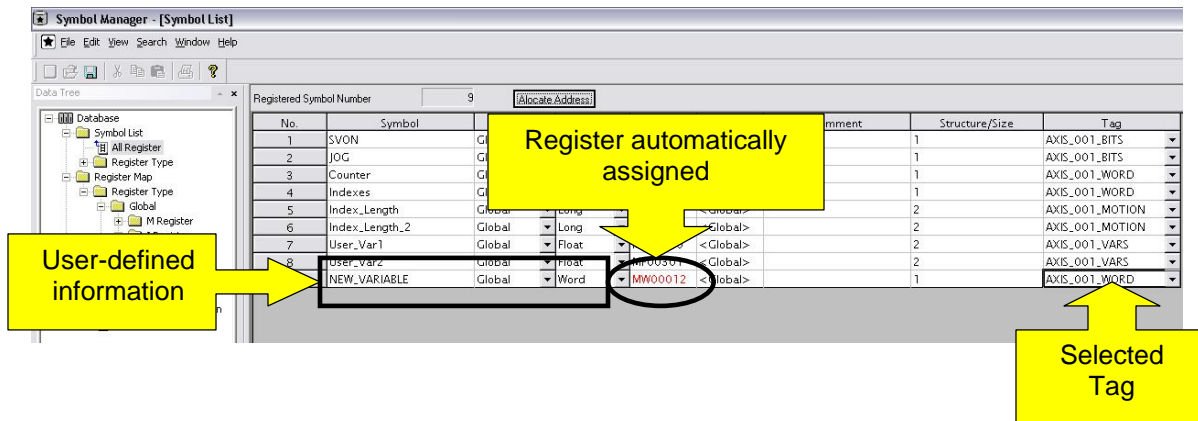


Fig 2. Using Automatic Address Allocation

## Advantages

Using Automatic Address Allocation allows the user to program in terms of symbol names rather than by register addresses. By focusing on symbol naming conventions, users can properly define their symbols in a way that allows them to most effectively organize their variables. Using an organized and well thought-out group of symbol names helps to greatly minimize errant data entering.

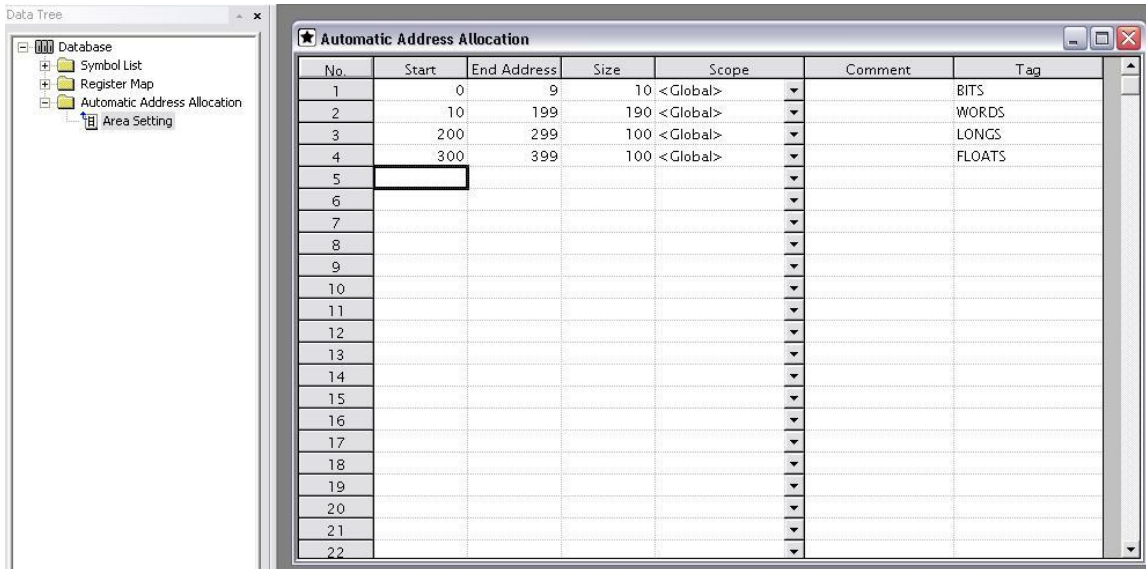
Using the Automatic Address Allocation also eliminates the chance of accidentally writing over previously used registers. By assigning symbols to registers, the Automatic Address Allocation register map recognizes how each register is being used, preventing symbols being assigned to register locations that are already in use.



**Disadvantages**

Automatic Address Allocation will not always start registers to be reserved for long and float data types on even numbered addresses (as recommended by the MP best practices). A solution is available to this problem as blocks of memory can be assigned within the auto-allocation function to certain data types. A block of addresses (as shown below), starting at MW0200, is to be reserved only for long data types. Since each long requires two registers, each long will start on an even number address.

- i.e. MW0000 thru MW0009 – bit data types
- MW0010 thru MW0199 – word data types
- MW0200 thru MW0299 – long data types
- MW0300 thru MW0400 – float data types



No.	Start	End Address	Size	Scope	Comment	Tag
1	0	9	10	<Global>		BITS
2	10	199	190	<Global>		WORDS
3	200	299	100	<Global>		LONGS
4	300	399	100	<Global>		FLOATS
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						

## Symbol naming & commenting conventions

### *Purpose of naming convention*

The purpose of using the symbol naming convention is to increase overall ease-of-use. Establishing a symbol convention that is easily understood and works well with the MotionWorks Symbol Manager makes sorting and a viewing of symbols easy-to-do and maintains the robustness of the program.

e.g.  
Symbol                      Comment  
Axis###X\_Velocity      [counts/sec]

**X** would be replaced by the first letter of the register type for that symbol:

B – bit type  
W – word type  
L – long type  
F – float type  
A – address pointer type

### *Recommendations*

Symbol names should be written so their function is clearly understood and easy to sort in MotionWorks. Below are some examples:

<b>Symbol</b>	<b>Comment</b>
Axis001L_Position	[counts/sec]
HMI_IndexCount	[number of indexes]
Machine_BeltLength	[inches]
Supervisory_Estop	[normally closed]

Tag naming convention for ease of searching and programming.

- Use real world names for easy to follow programs.
- By putting units in comments, this displays in ladder elements, again for ease of understanding program.

## **DEVELOPING CODE**

With the memory allocated and symbol naming conventions completed, code is ready to be developed as decided by the user's evaluation of the programming methods. The application is ready to be translated to executable code using Ladder, Motion Programs or Function Blocks. When translating the application to executable code, there are many key techniques used in developing a reliable, efficient application program. Key techniques for the three programming methods are examined in this section.

### **Ladder Techniques**

When programming in ladder, the user must consider the scan based I/O and Parameter update. This is a key difference to understand for programmers familiar with structured text based languages where the application directly controls the flow of instruction execution. In addition, high and low speed scan drawings require special consideration when developing logic to assure intended results.

#### *Low Scan Interlocks*

The MP Controllers have two user settable scan rates: high and low. As discussed in the program architecture section, this allows the application to be optimized. Low speed drawings typically contain non-speed critical machine sequences, such as I/O interfacing sequences and HMI interface logic.

#### **Axis Specific Low Scan Interlocks**

Enabling an axis is typically performed in a low scan drawing as described in the drawing architecture section. The goal here in Axis Interlocking is to check status of critical axis and system alarms, and to create an 'axis normal' signal. The axis normal signal will be used to allow the 'servo enable' signal to turn on, and remain on. If an error occurs during any mode of operation (such as manual mode or automatic production mode) then the servo on signal will turn off, and the axis will stop motion.

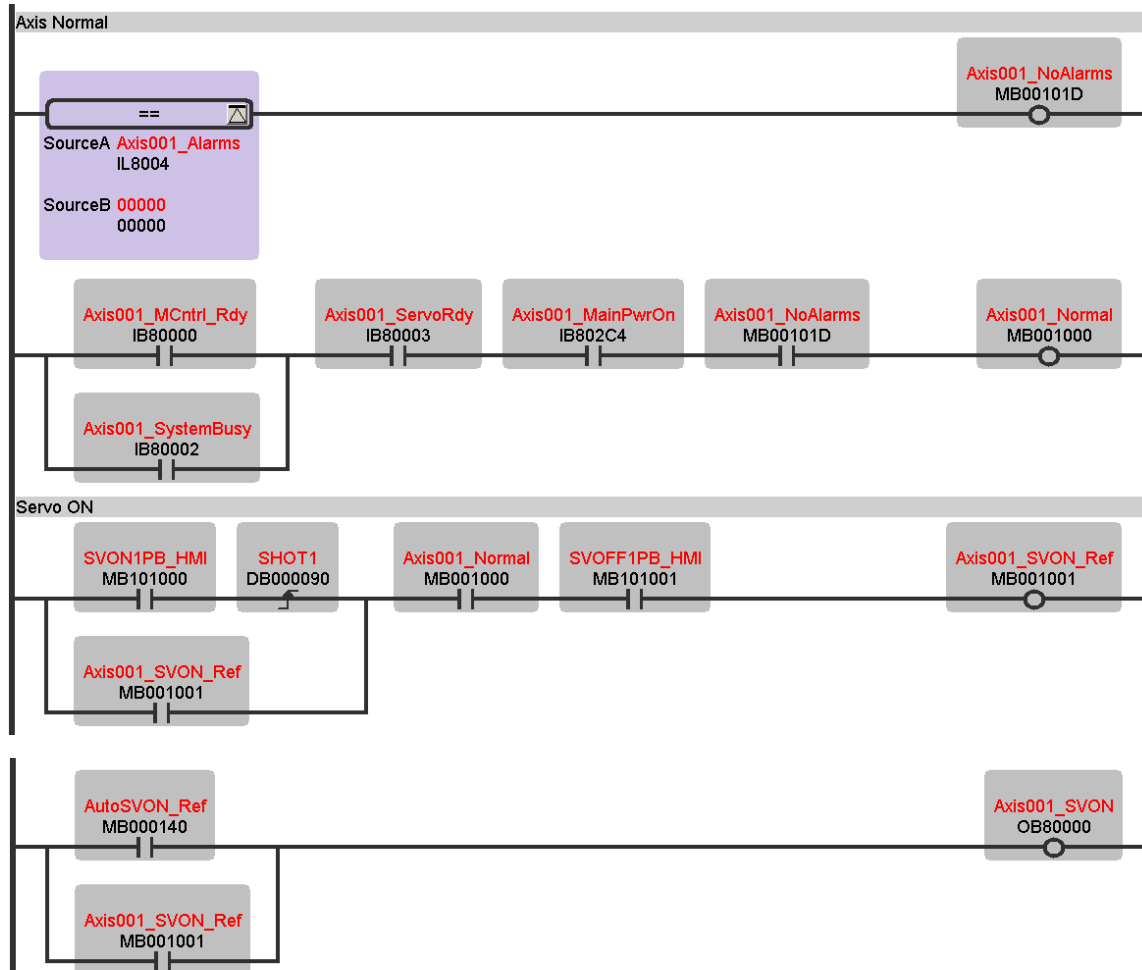
In DWG L20.01, the first rung of code checks the entire status register IL8004 loads its status into the Axis001\_NoAlarms catch all bit for axis #1. Even though the IL8004 register catches all alarms, it is helpful to summarize all the alarms with one bit to aid in visually troubleshooting if an alarm occurs (see user manual for bit breakout). IL8004 catches alarms such as

*IL8004 (Alarm Status for Axis #1)*

- Servo Driver Errors
- POT/NOT (positive and negative over travels)
- Positioning Timeout, Excessive Error, excessive speed
- Servopack Parameter setting error, comm. error, encoder disconnect, etc

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

**DWG L20.01**

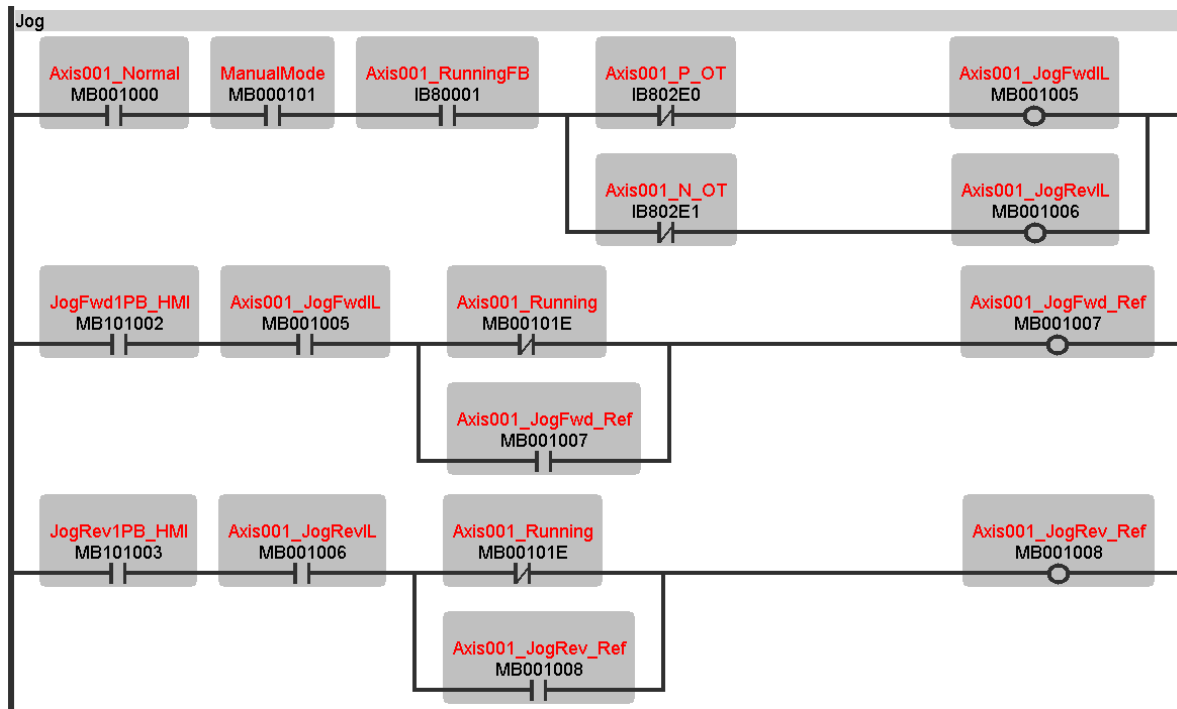


Its important to verify the axis status by confirming that the controller is ready, IB80000 (Axis 1) and the ServoPack is ready, IB80003 (Axis 1). MB00101D (Axis 1.) Breakout the main power on bit. IB802C4 (Axis 1). In the above code, "Axis Normal" is one of the conditions required to enable the servo. An alarm, loss of ServoPack power, or communication error will disable the axis. Also note that the user input is in series with a rising edge one shot to prevent rapid toggling of the enable signal.

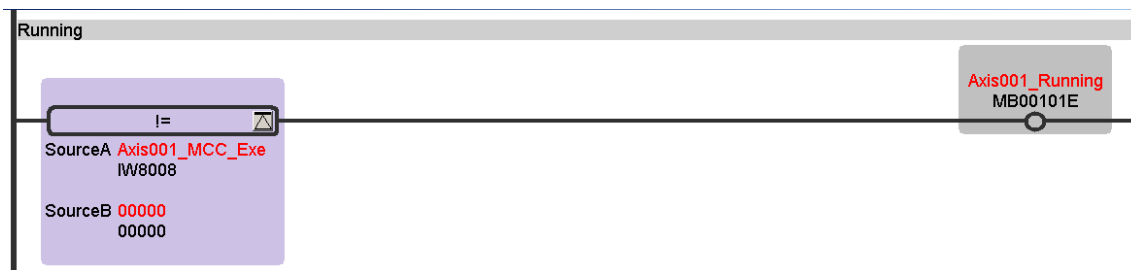
A normally open contact for breaking the latch of the servo on reference MB001001 (Axis 1) is common practice for machine control from using hardwired normally closed stop push buttons.

Jogging the axis individually, which is typically a manual machine mode function, is also interlocked in the low scan drawings. Typical logic includes an interlock to jog in each direction. The interlocks typically check the axis status for *normal*, that the axis is enabled, the machine is in manual mode and the over travel for the selected direction. Then if the jog interlocks are OK, the user input is allowed to reference the axis to jog in the corresponding direction. Bits set in the low scan drawing to reference the axis to jog then are used as contacts in the corresponding high scan drawing to set the motion command code, feed speed reference and direction.

**DWG L20.01**

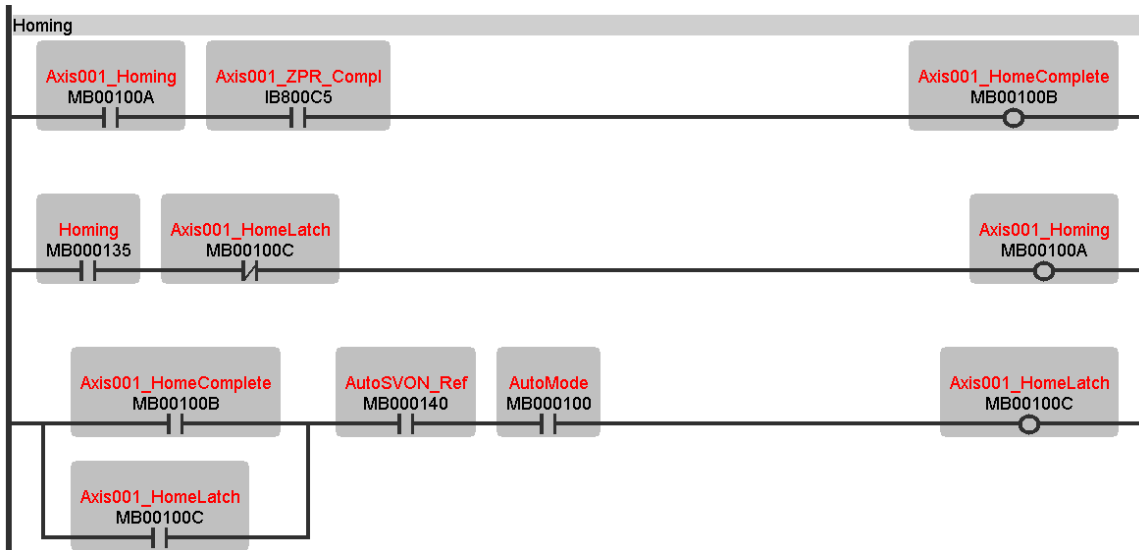


It is recommended to verify the axis is not already executing another command by monitoring the servo command type response IW8008 (Axis 1) at the bottom of drawing L20.01 for interlocking the axis commands as above for the axis jog references.



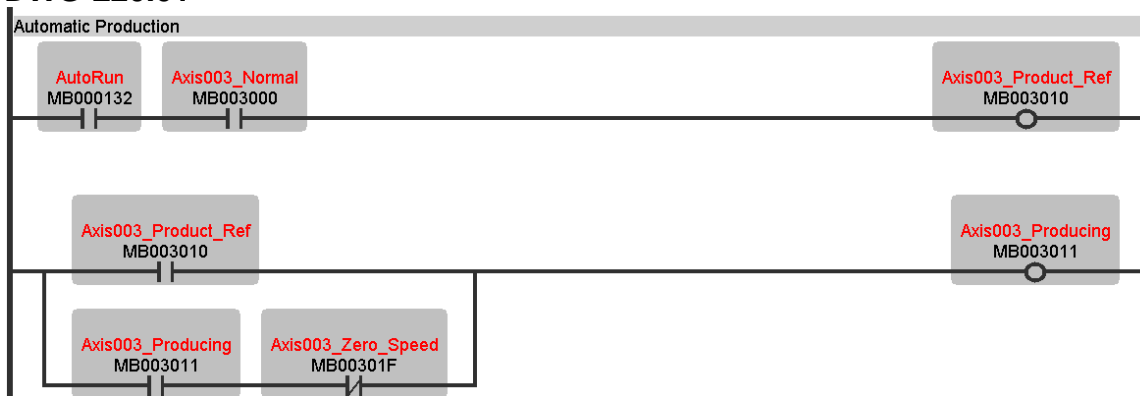
Homing an individual axis is another typical low scan interlock as shown below. First, monitor the axis to complete it's home or zero point return, with the next rung used to actually initiate the homing logic by setting the motion command code to "zero point return." The third rung is the "home completed" latch. Conditions that break the latch would be the axis is on and in automatic mode. These conditions depend on machine operation and change to suit the application.

**DWG L20.01**



Interlocking for an individual axis to be commanded for automatic production is also typically placed in the low scan drawing. Set the command bit for the axis to operate in production mode in a common machine sequence drawing such as L10. Interlock the production reference with individual axis conditions such as "axis normal" before actually commanding the "production" bit to run.

**DWG L20.01**



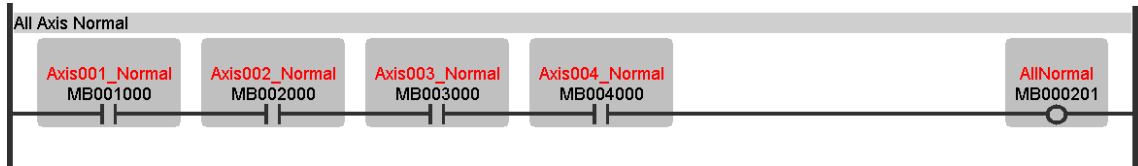
**Non-Axis Specific Low Scan Interlocks**

Other low scan interlocks include common machine sequences and control in the L10 drawings. Typical low scan interlocks include machine mode and mode switching logic such as auto / manual and homing.

L10 and its subsequent grandchild drawings (L10.xx) include common machine sequences, where the status of dependant axes are summarized in for interlocking purposes. Logic in the individual axis specific L20 drawings will be used to interlock in L10.

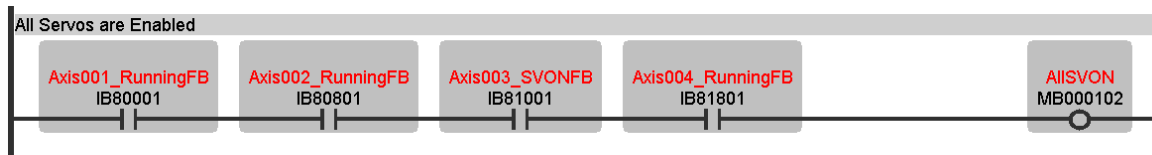
All “axes normal” summary for preventing any common action if an associated axis is abnormal.

**DWG L10.01**



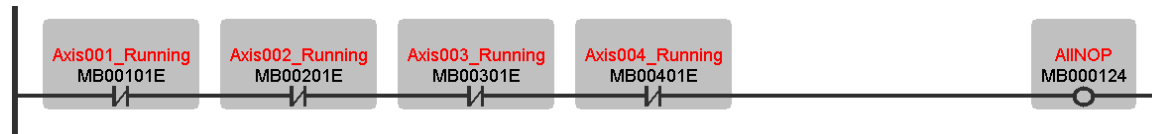
All axes “Servo On” summary for preventing any common action if an associated axis is off.

**DWG L10.01**



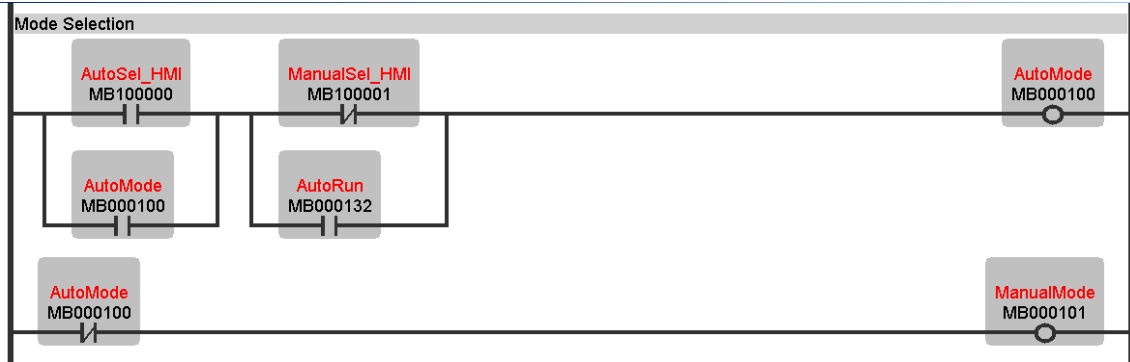
All axes are idle prevents issuing a command that would expect axis response, such as cycle start.

**DWG L10.01**



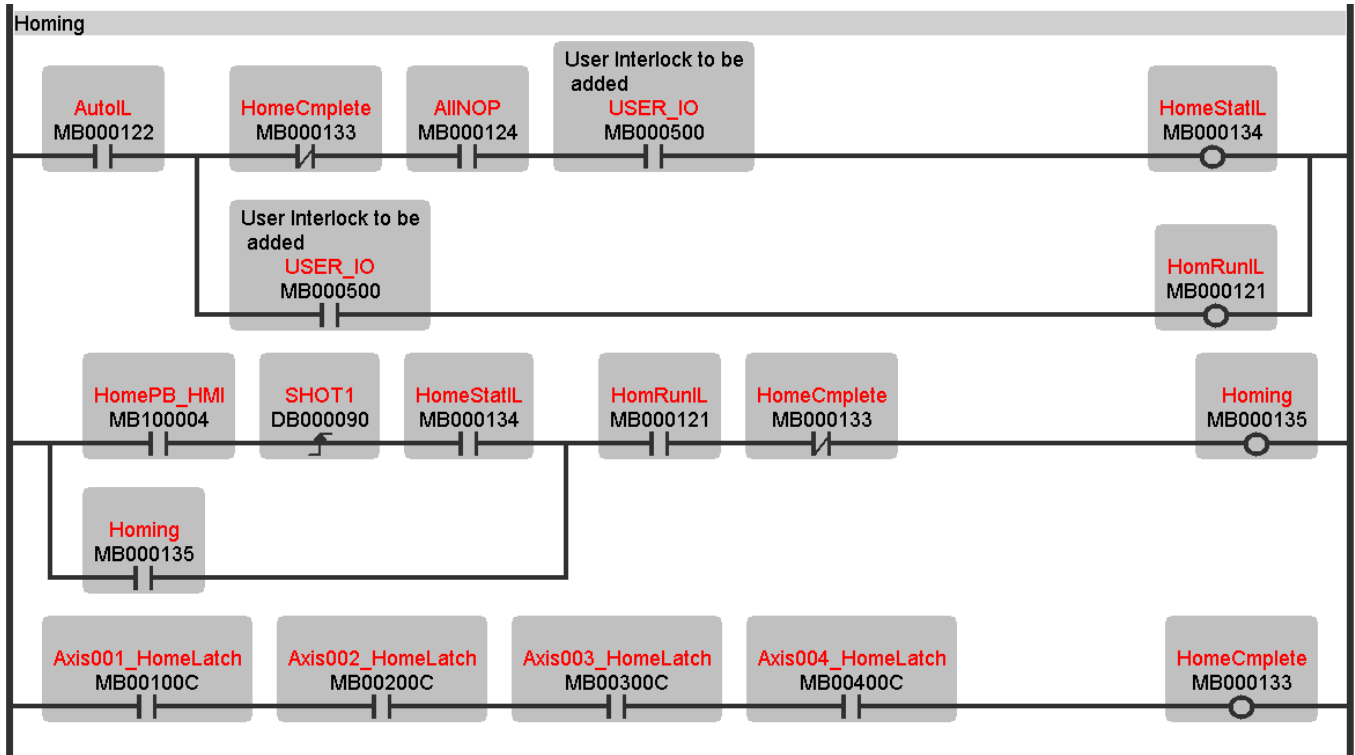
Auto / Manual Mode Selection

**DWG L10.01**



System Homing Interlock and Control

**DWG L10.01**

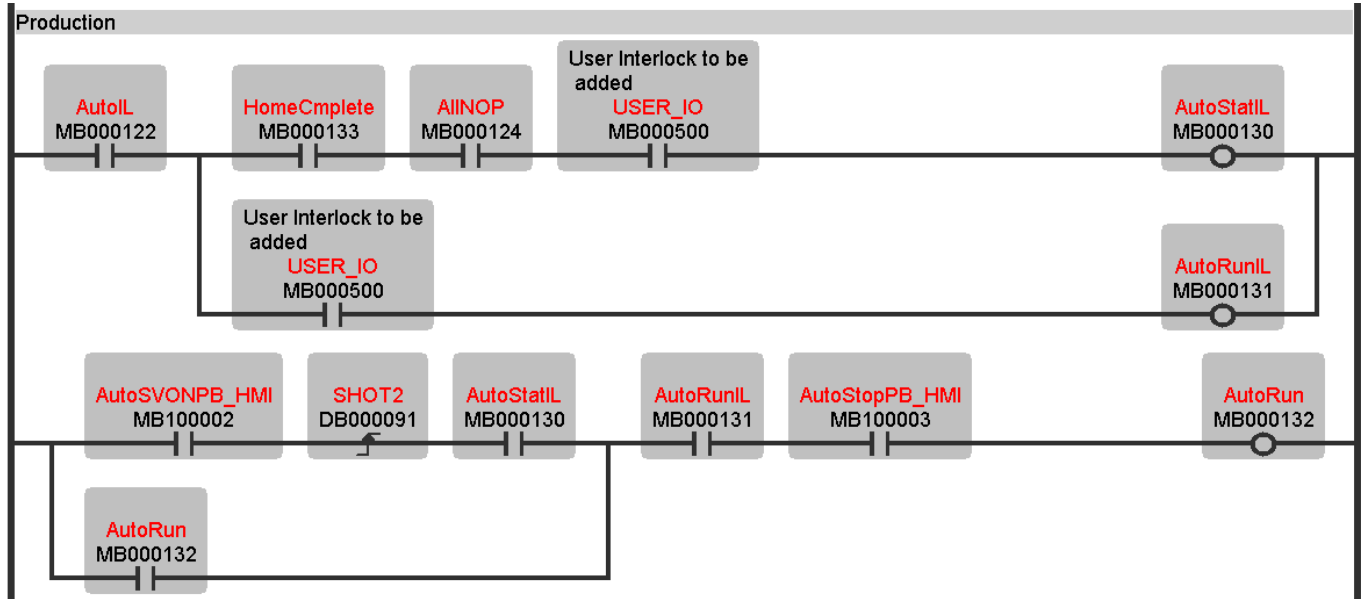


Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280



Auto Production Mode Interlock and starting

**DWG L10.01**



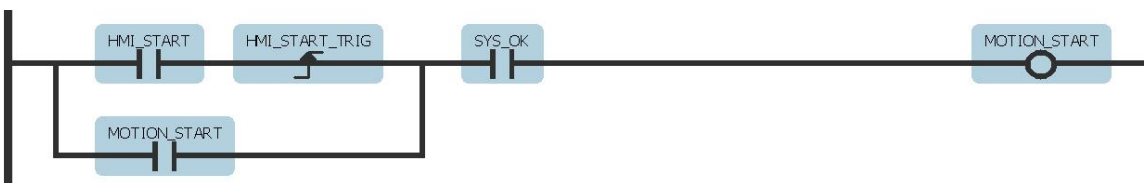
**HMI interlock (limiting, setting, resetting, handshaking, interlocking)**

Definition of information to be transferred initially (varies per application)

*Purpose of Interlocking HMI*

All command bits from an HMI to the controller should be programmed as latched momentary signals. This provides added safety for any type of E-stop or power-off conditions that may occur.

In the example below, the HMI\_START is a command signal from the HMI that is read only on a rising edge. The SYS\_OK must be closed indicating that the system is operating properly before motion can start. When the HMI\_START bit has a rising edge, the MOTION\_START coil is activated and latched on until the SYS\_OK bit goes low.



It is recommended to limit values from HMI so that errors are prevented (i.e. putting to small/large of value into a register)

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

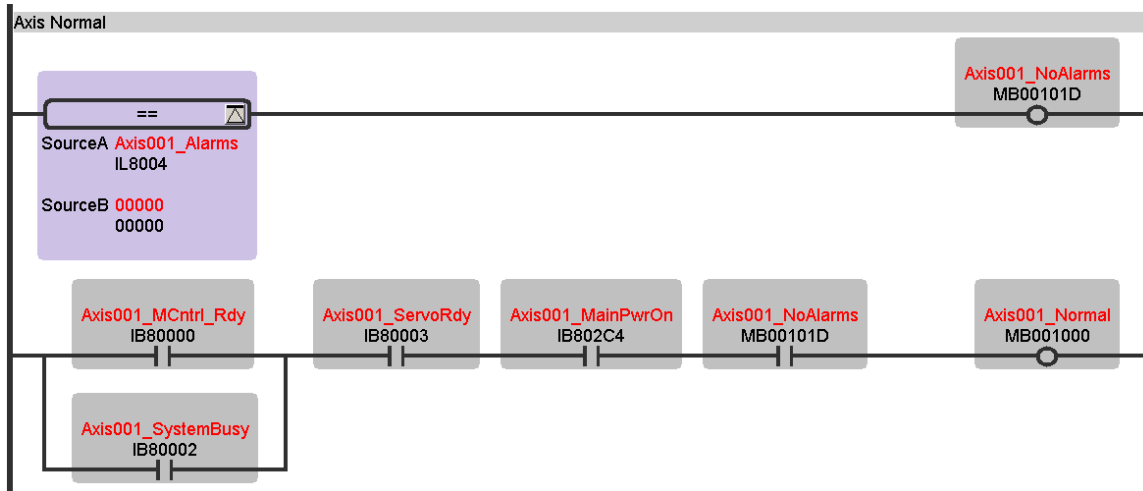
### Machine Interlocks

Machine interlocks are critical to allow the controller code to operate within the mechanical limitations of the machine and assure safe, reliable, high performance operation. Machine interlocks include conditions for sequencing operations where components may interfere with each other, peripheral components interlocking such as labelers or heater controls, upstream and downstream systems interlocks as well as safety system interlocks.

#### Axis Enable Interlocks

To prevent alarms or warnings, the machine safety circuit and amplifier power should be verified before attempting to enable an axis. This is done in the individual axis low scan drawings with the axis normal condition to enable an axis. By verifying the controller an amplifier is ready the power will be on the amplifier and the communications will be synchronized, therefore the controller will be able to enable the axis successfully.

#### DWG L20.01



#### Automatic Mode Interlocks

Before a machine can enable, the state all the axes must be normal and enabled, the safety circuit must be intact and no faults can exist.

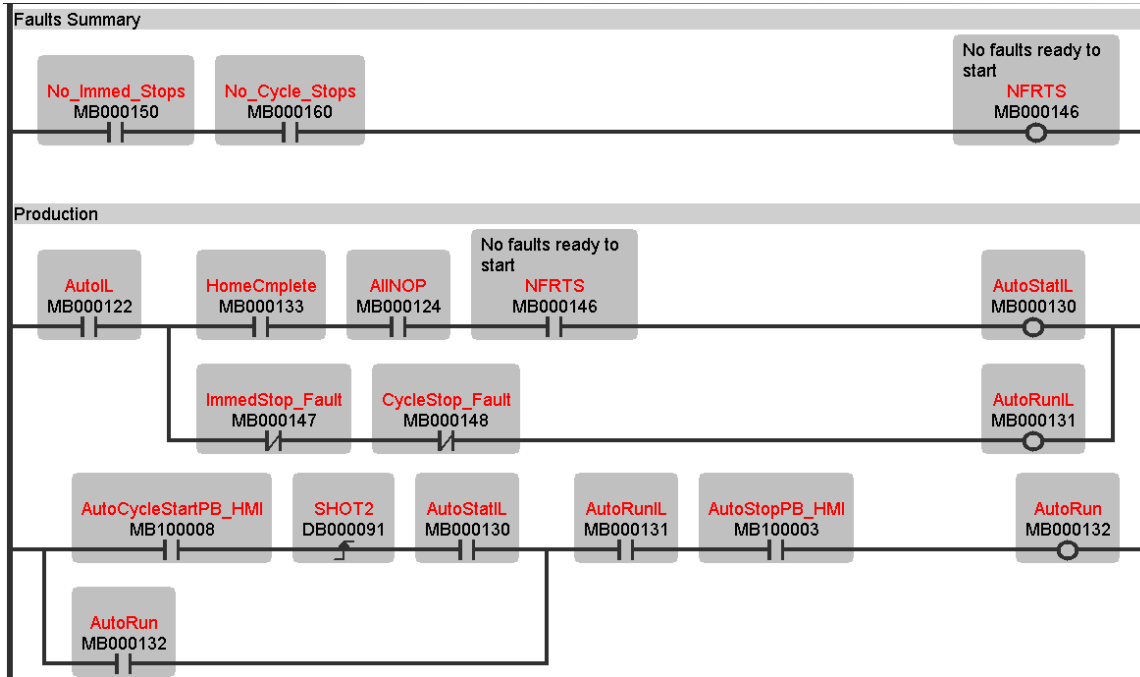
#### DWG L10.01



Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

Before the machine starts cycling in automatic production mode there must exist no faults on the system and the machine must be in the ready to start state.

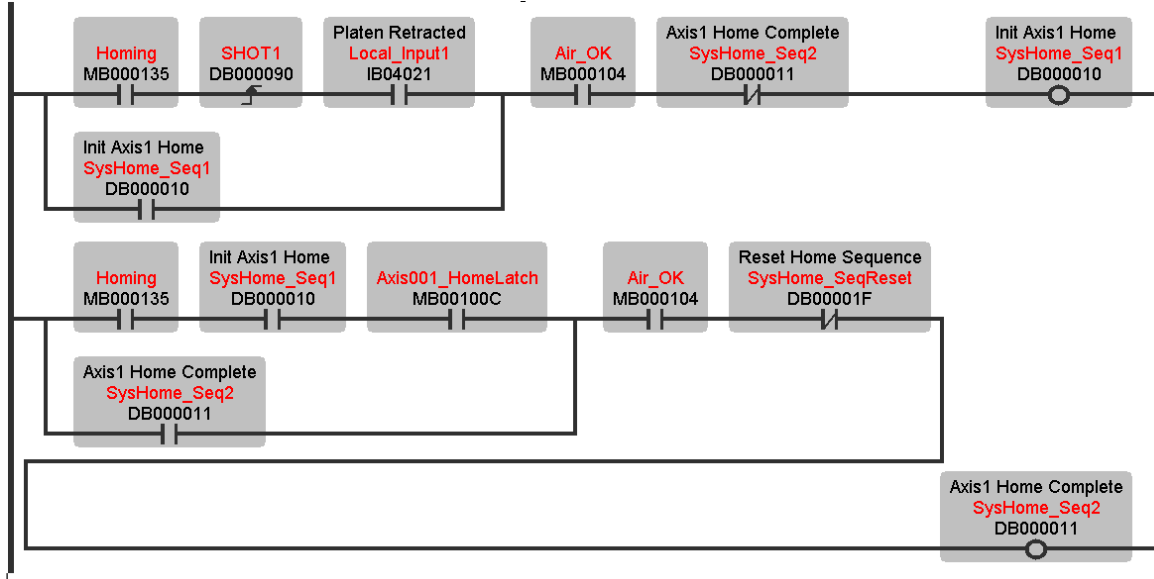
**DWG L10.01**



Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
 (800) YASKAWA - Fax (847) 887-7280

A system homing sequence may have to account for physical interference and require other components in a certain state for the sequence to advance. It is critical to include interlocks in any machine sequencing to prevent damage. In this example, air pressure is required as well as sensor to detect components are clear.

**DWG L10.02 1**



## Gearing

Creating a gear application mainly consists of scaling the incoming master pulses with a B/A ratio and sending the result to the slave's commanded position register. In the MP its possible to perform these calculations in ladder on a scan by scan basis. In this mode, the calculation is performed every scan, by taking the differential pulses of the master encoder (called Scan Difference), applying the formula below, and generating a scaled slave position output. The power of the MP platform allows adjustments to be made at each step of the calculation at the rate of the program scan, thereby allowing the user to change the ratio on the fly. The resulting scaled segments are accumulated in a register that is fed to the slave commanded position.

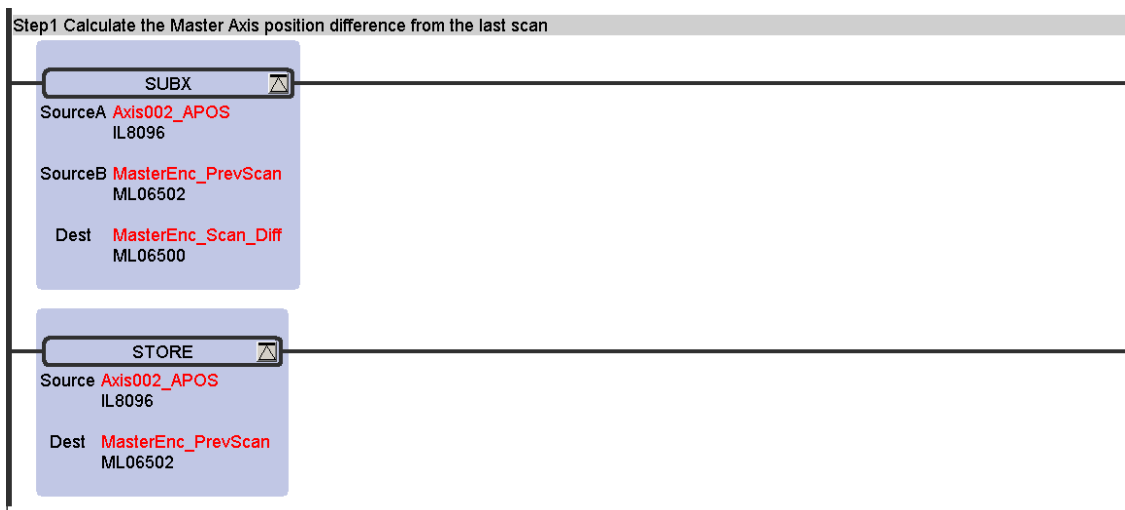
When scaling the master position segments, often the result is not an integer number; to avoid losing synchronization, the remainder must be stored and added on the next scan.

Therefore, the calculation done each scan should be:

$$\text{Scaled Slave Position Segment} = \frac{\text{Scan Difference} * B + \text{Modulus of the Last Scan}}{A}$$

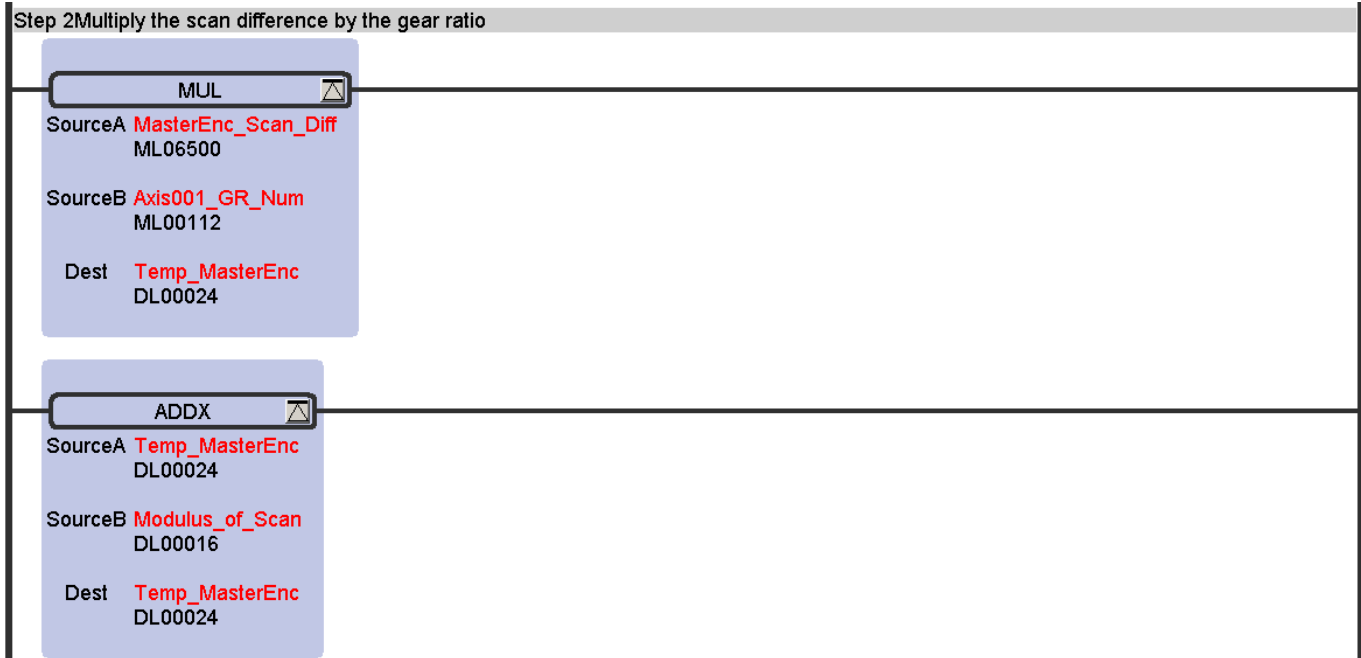
The Process in steps:

1. Calculate the master position difference from the last scan and store the current position.



Notice the use of SUBX and ADDX for proper calculation through the rollover point of a 32 bit register. These instructions perform "two's complement math," which results in the correct answer even as one value crosses the rollover point. No math errors are generated.

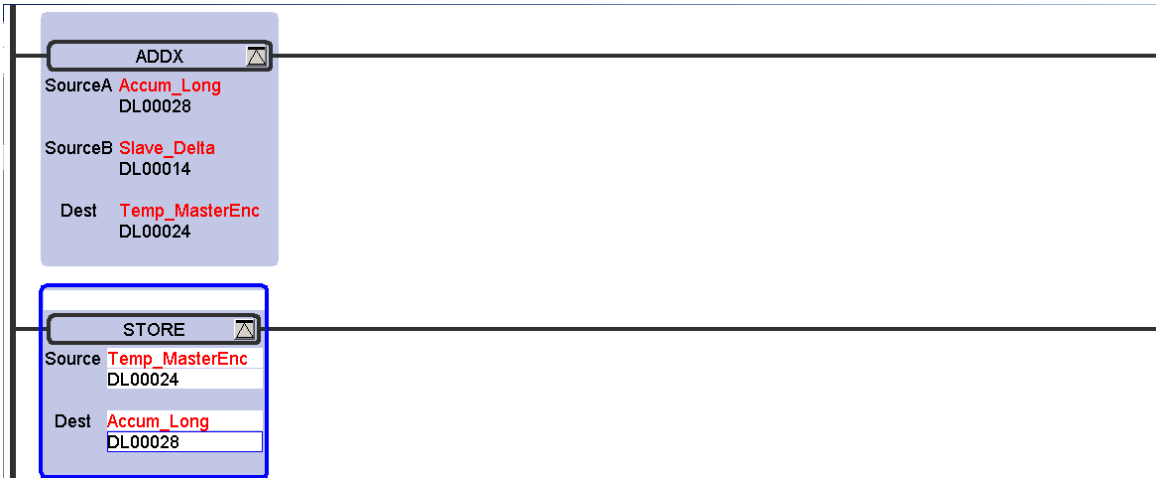
2. Multiply the result by B and add the remainder from the last scan.



3. Divide the result of step 2 by A and save the modulus of the division so it can be added in the next scan.



4. Accumulate the calculated slave position in an accumulator register.



5. Map the accumulated result into the slave position command register.



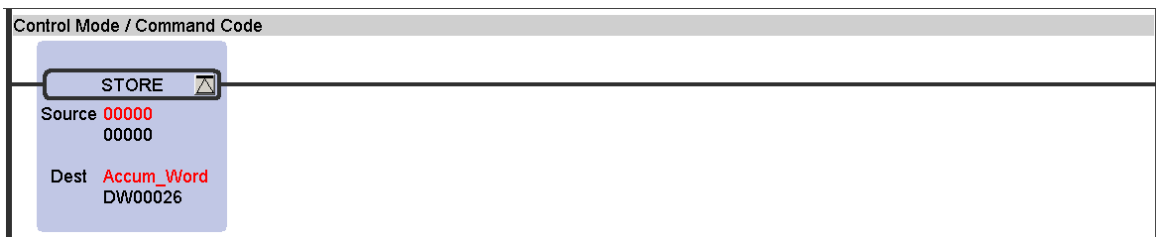
### Waterfall Technique

Because it is possible to write directly to the motion registers in a Yaskawa MP controller, it is good programming practice to write values to the register in only one place to simplify monitoring or debugging. This practice is accomplished using local registers as accumulators to interact with logic and to finally store a value to the actual motion register in one place. This method is referred to as the waterfall technique. Another benefit to this method is the ability to load a default value if no conditions explicitly call for another value.

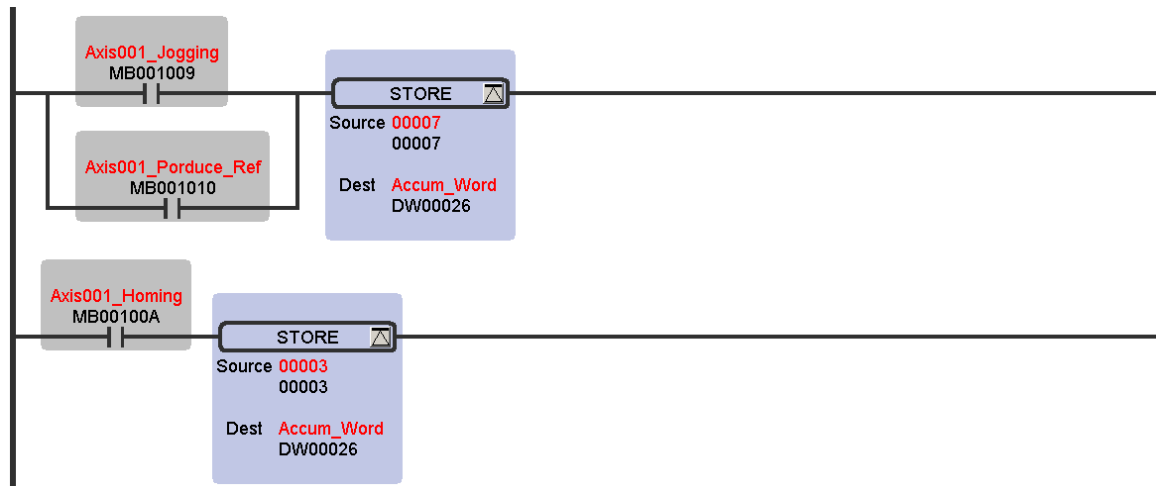
This coding technique is accomplished in three sections:

1. A default, initialization, or last scan value is stored to the accumulator register.
2. A conditional value is loaded to the accumulator based on interlocks with other logic values. This action overwrites the default value.
3. The final store occurs when the accumulator is copied to the actual motion register.

In the following example, the motion command code is set for axis 1. The default value is first stored to zero, or NOP as a safety so that no action will be taken if a fault happens in logic below. Note that DW00026 register is used as an accumulator in the code below, this will be described in more detail later.



If jogging, homing or production is required, the corresponding motion command code is loaded to the accumulator word DW00026 in the conditional section.



Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280



Finally, the value of the accumulator word is unconditionally stored to the Motion Command register for axis 1.



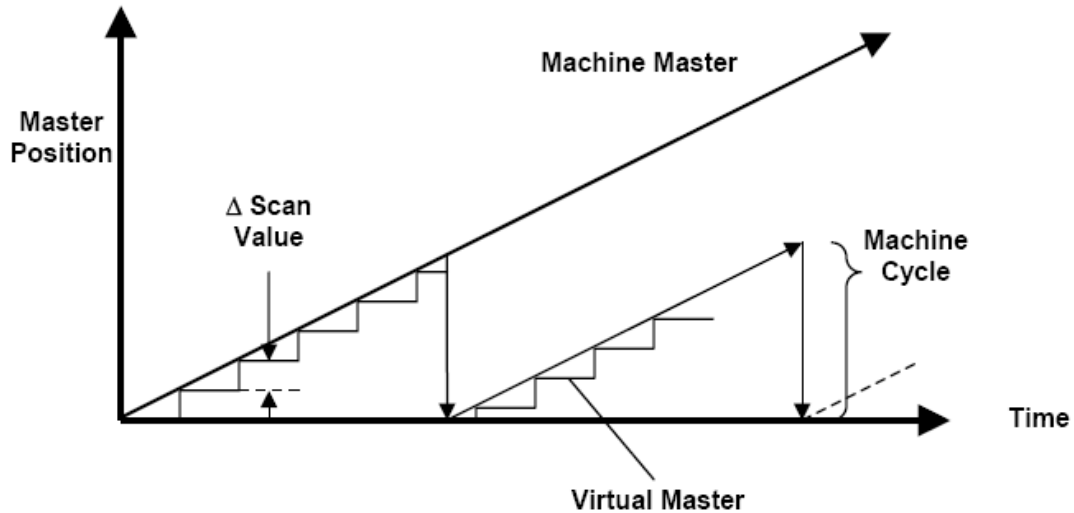
The Waterfall technique illustrates the suggested use of local registers, in particular the accumulators. Best Practices recommends the use of the following specific local registers when implementing the waterfall technique:

*Accumulator Register Address Recommendation:*

DW00026	16 bit integer accumulator	(ex. motion command code)
DW00027	16 bit logic (hexadecimal) accumulator	(ex. parameter number write)
DL00028	32 bit long accumulator	(ex. position reference)
DF00030	32 bit floating point accumulator	(ex. floating point references)

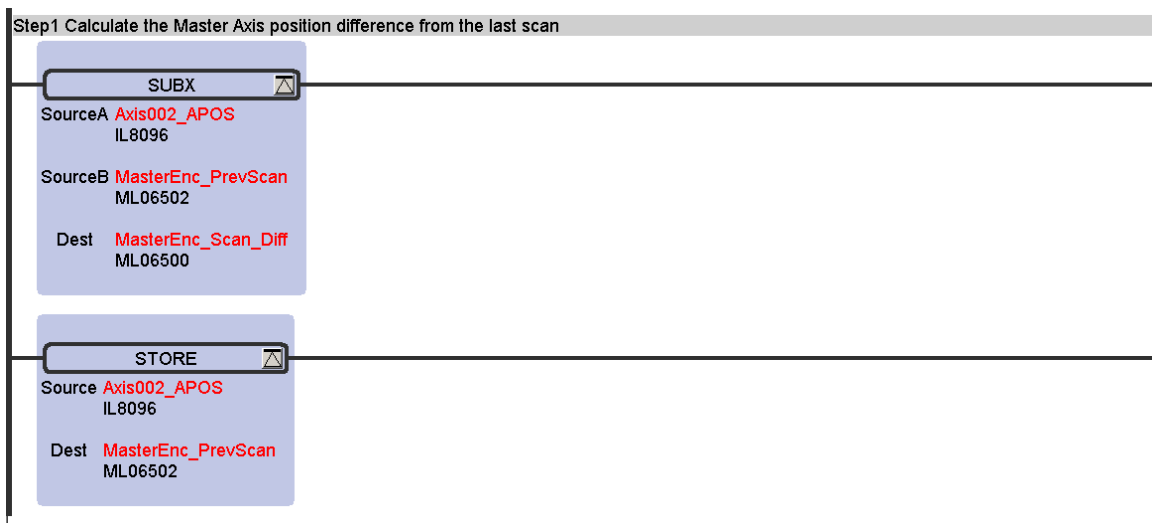
### Modulus Technique

When an axis repeats an operation indefinitely in one direction, it may be important to modulate it to create a repeating saw tooth value for a machine cycle or cam profile. A start location and condition may need to be determined also. Typically, most applications start at zero in the cam table, but it can vary as it is application dependant.



In the following example of a modulus technique, the start location is the beginning of the cam table and the condition is the ending of the machine cycle.

1. Calculate the master position difference from the last scan

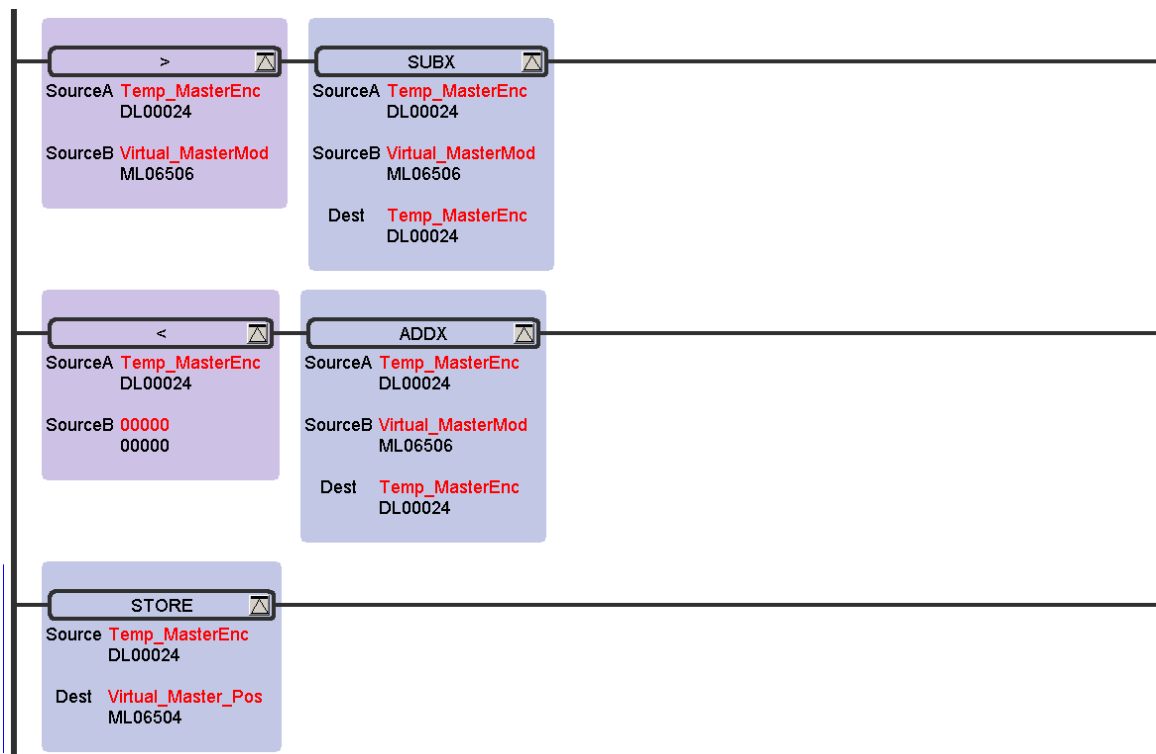


Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

2. Increment the virtual master by the scan differential amount



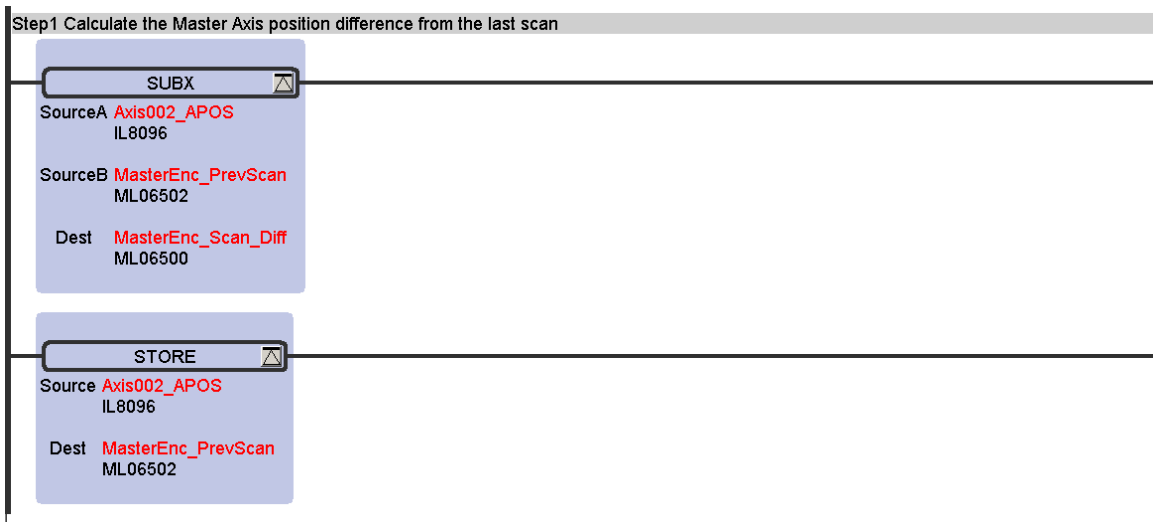
3. Test the virtual master to see if the cycle modulus has been exceeded. If so, reset the value of the virtual master to be within the machine cycle. When the modulus is exceeded, set the virtual master to the accumulated scan counts that have exceeded the machine cycle range.



### Delta Scan

The delta scan technique is a convenient method to calculate the change in position or value at the scan rate. This is useful in creating a modulus function or saw tooth position change for a virtual master that repeats cyclically. By using this method a saw tooth waveform is generated to represent the change of position versus time. This technique is also useful in calculating speed compensation that may be used to account for time delay due to network updates and the scan of the drawings. The delta scan method is the same as for the modulus and gearing techniques, repeated below for clarity.

1. Calculate the master position difference from the last scan



2. Increment the virtual master by the scan differential amount



The results can be multiplied by a given number of Mechatrolink network cycles for speed compensation. This is possible because Mechatrolink network updates are deterministic.

### Handling Rollover

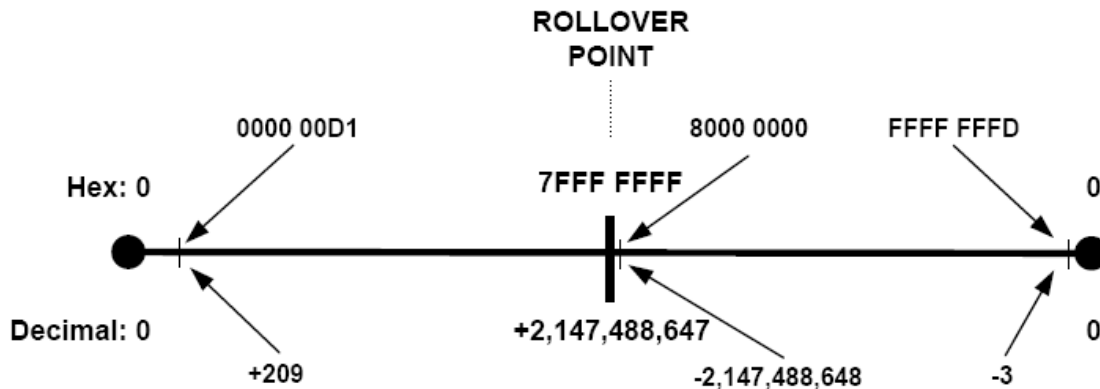
Use the math functions ADDX or SUBX to prevent miscalculations due to long registers, ML or DL rollover. With 32 bit registers, the maximum and minimum decimal values are +2,147,488,647 and -2,147,488,648, which correspond to 7FFF FFFF and 8000 0000 in Hex respectively. For example, if IL8016 has rolled over from the positive or the negative side, the IL8016 and ML00100 math would be as follows:

Positive Rollover:  $-2,147,488,000 - (+2,147,488,000) = +1296$

Negative Rollover:  $(+2,147,488,000 - (-2,147,488,000)) = -1296$

Using ADDX and SUBX always guarantees that the result of the math operation is a valid answer, and no error will be generated.

Graphical layout:



## Sequencing Techniques

Machinery is required to repeat processes accurately and consistently with robust control, without intervention other than user clearing mechanical jams and user interface to prompt their actions. When developing sequences using a machine controller, it is important to program for consistent execution of tasks, with built in recovery functionality to handle sequence exceptions.

### State Machine programming vs Step Sequencing

Step Sequencing is defined as motion or logic sequences programmed in steps using individual bits of words to initiate and monitor the steps of the sequence. Another common practice is state programming where states are defined by actions that take place. Transition conditions determine the sequence, and the state is determined by an integer that is updated when a new state is entered. Both methods are useful, below is a more detailed description of Step Sequencing.

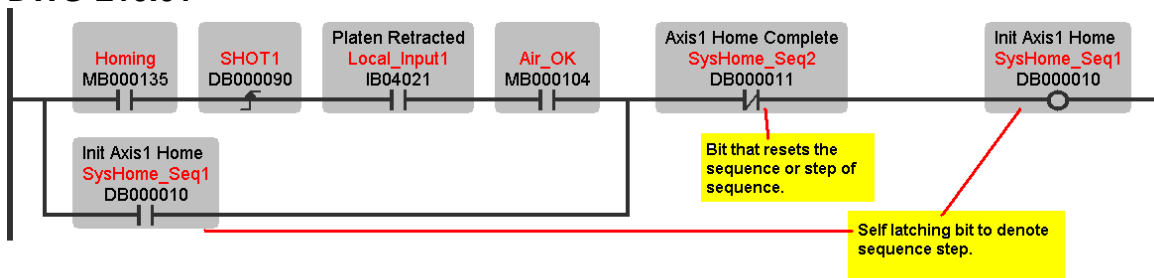
### Step Sequencing Technique:

Most controller programmers use two different types of sequencing by bits.

- 1) Retained Step Sequencing: each step of the sequence is set true, and remains true until the end of the sequence, or the sequence is aborted when all of the bits are reset.
- 2) Toggle Step Sequencing: In each step of the sequence, a bit is set. Then there is a corresponding bit representing that the step is completed, which toggles or resets the initial step bit. When the sequence completes, then all the “step completed” bits are reset. Some favor this method because steps that are completed should have no interaction with active steps.

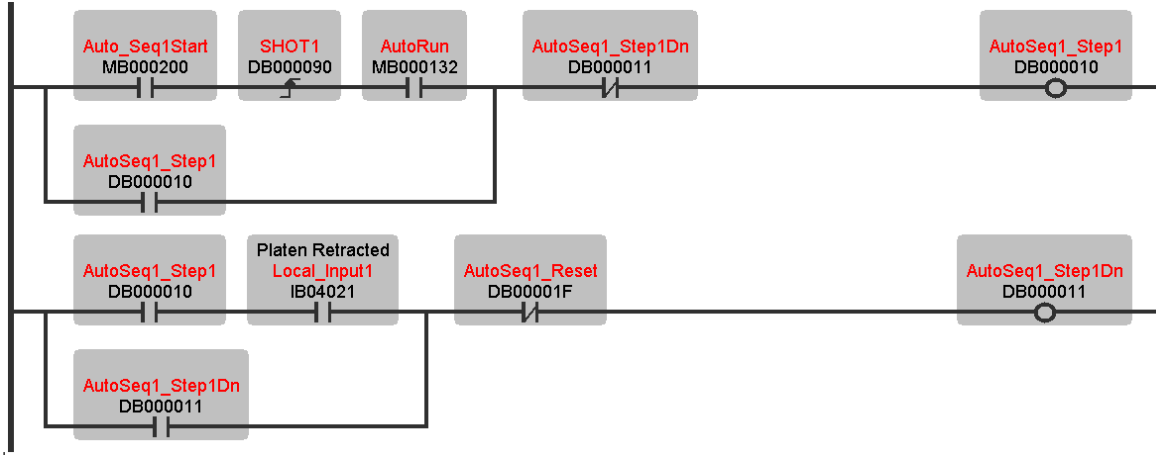
Regardless of the method selected, its recommended to use self-latching bits with the unlatching bits between the branch and coil, not in the branch, as shown below.

### DWG L10.01

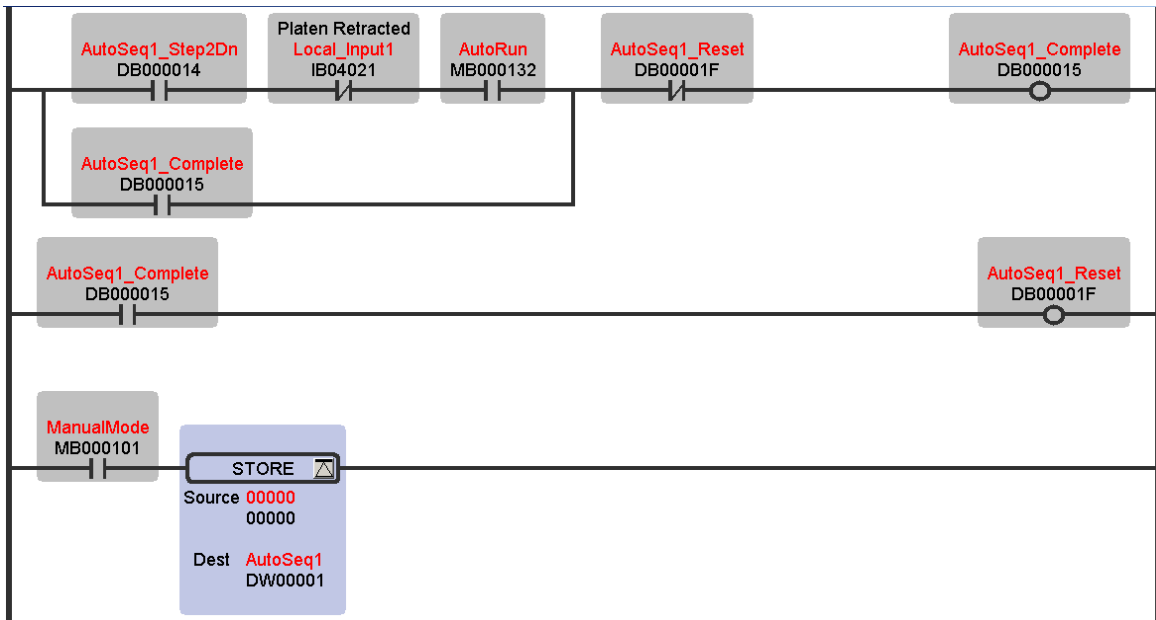


The Toggle Step Sequence technique uses a step complete bit (DB000011) to reset the sequence bit that first initiated the step (DB000010). Note in the second rung the entire sequence complete resets the done bit DB00001F.

**DWG L10.02**



The completion of the last sequence step should latch a sequence complete bit. Then the step complete bits are reset by the “sequence reset” bit, or if the system changes auto mode the sequence is aborted.



Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

## Rules For Motion Programming

The MP2000 series of motion controllers offer a powerful, yet flexible text-based motion programming language that lends itself well to many applications, especially those that involve linear interpolation or complex motion sequences. As always, the user is free to program as desired, but experience has shown that following a few simple guidelines, as illustrated in this section, will help the programmer produce a motion program and related ladder code that is robust, easy to troubleshoot, and easy to modify.

These guidelines are explained in detail and illustrated with examples in the pages that follow.

- Starting a Motion Program
- Active Interlocks During Motion Program Execution
- Stopping a Motion Program
- Bit Handshaking between Motion Program and Ladder Code
- Using WHILE loops in Motion Programs
- Using PFORK, JOINTO, and PJOINT in Motion Programs
- Using Subroutines in Motion Programs

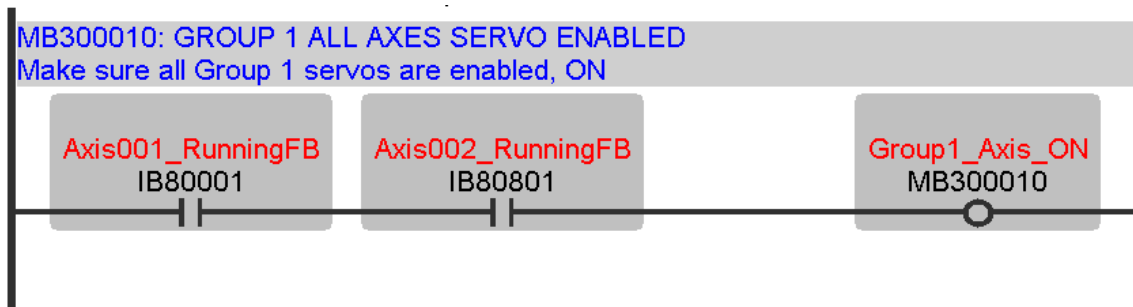


**Best Practice Rules For Using Motion Programs:**

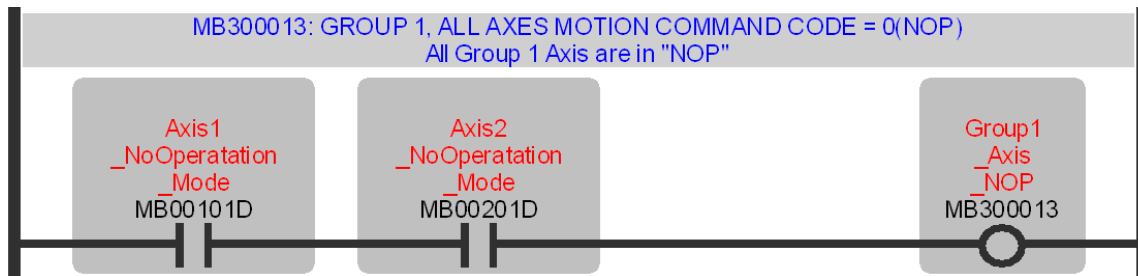
*Initiating Motion Program*

Before starting a motion program, ladder code should include interlocks that ensure the following conditions are true:

- All group axis servos must be ON (MB300010 in H25)

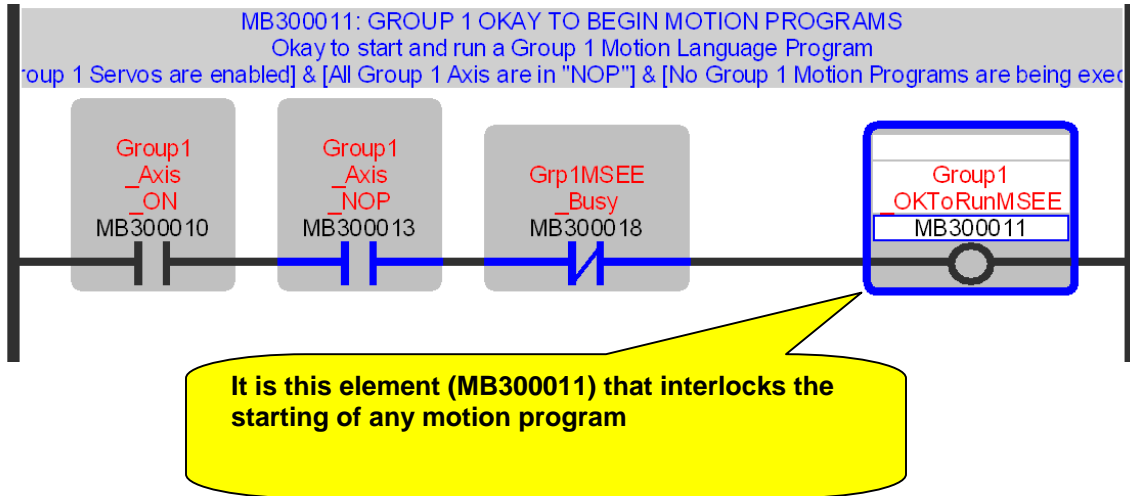


- All group axes must be in "NOP" (no operation) mode (MB300013 in H25)

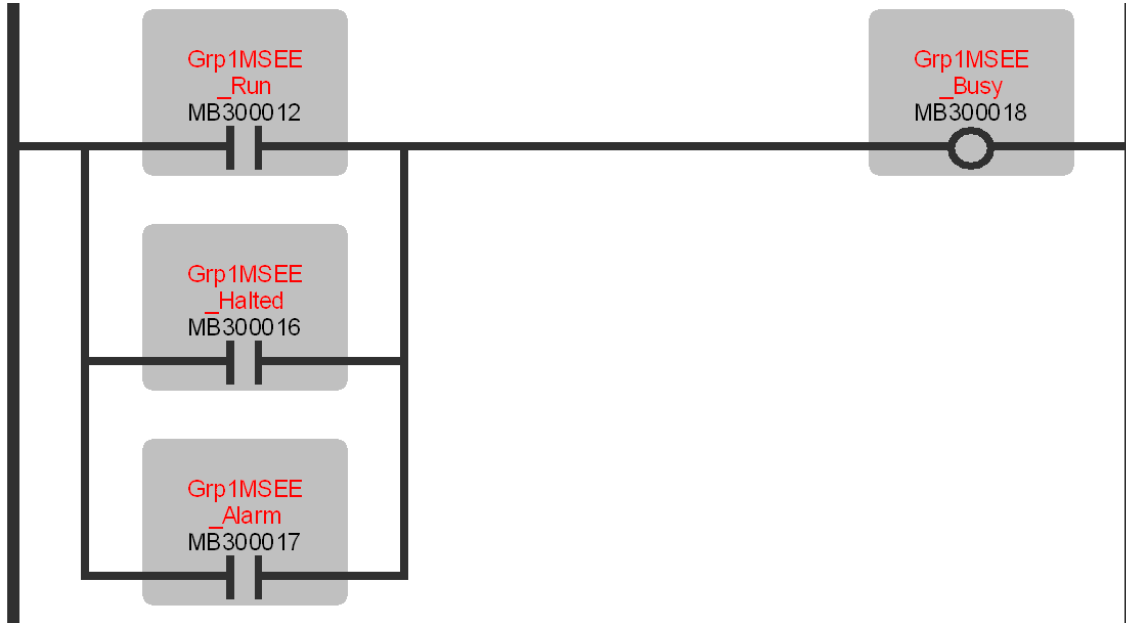


- No motion program is running, halted or in alarm for the specified group. (MB300018).

**In H25:**



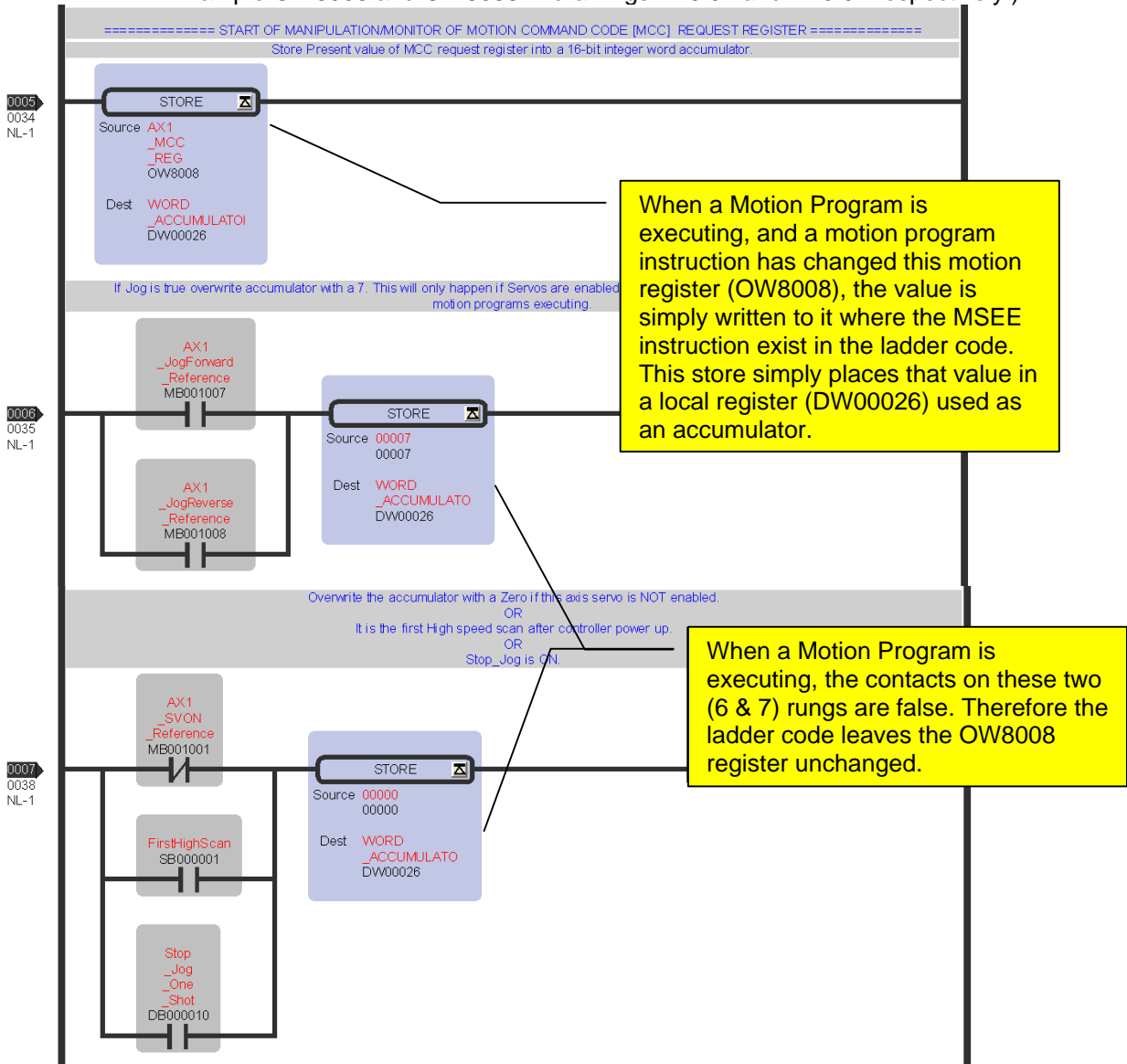
**In H25.01:**



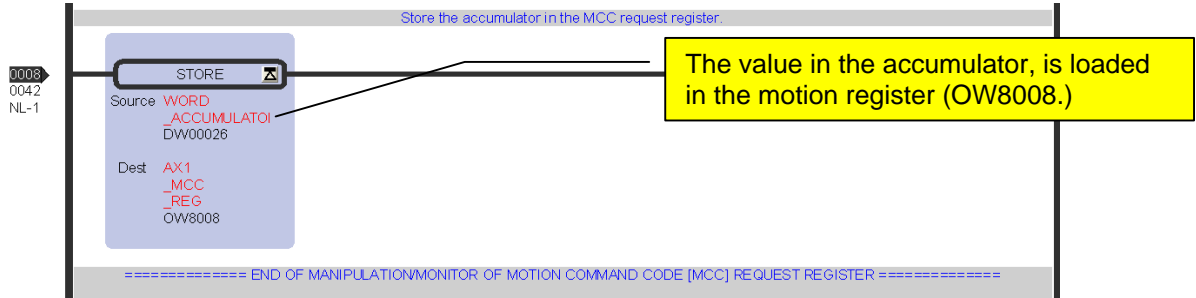
*Interlocking During Execution*

During the execution of the motion program:

- Interlocks should exist in the ladder code to keep it from changing global registers that the motion program writes to. These registers include “M”, “O”, and any group axis output registers. (Use the ladder’s “waterfall technique” – hold the register value. Example OW8008 and OW8088 in drawings H20.01 and H20.02 respectively.)



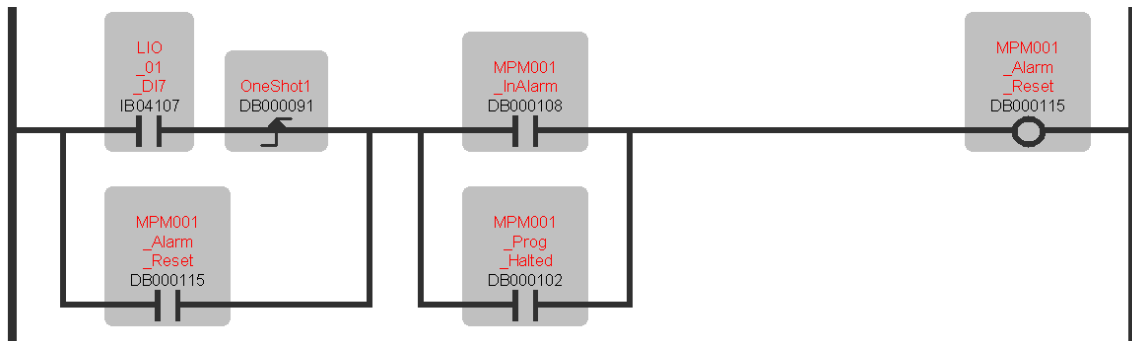
Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280



- Reading any global registers (M, I, O, S, C) are of no issue or concern.

Stopping a program that is executing:

- Expect to issue an alarm clear after halting (stopping) a motion program that was running. Expect an Alarm if the motion program running continuously was stopped. This means the program was either stopped, paused, or in single block mode when interrupted.



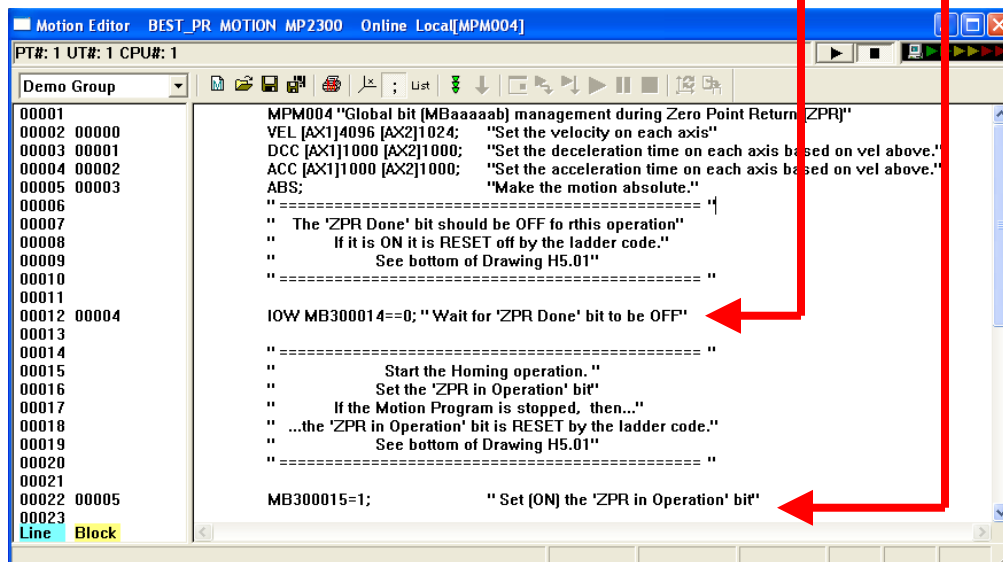
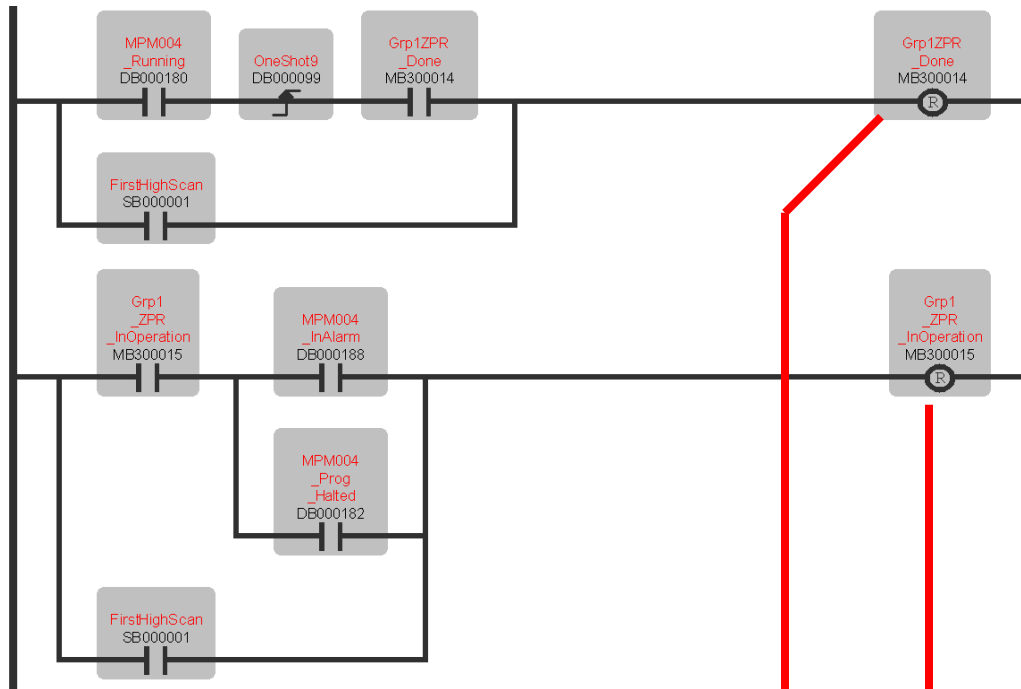
- Include ladder logic to flag the operator (or upper controller or other MP code) that the motion program was halted or alarmed.

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

Rules for bit handshaking or setting any global bit with Motion programs:

Understand that the instruction to turn a bit ON or OFF in a motion program is equivalent to a SET COIL and RESET COIL respectively in the ladder code.

- Ladder logic must exist to RESET any bit that was SET in the Motion Program, in case that program is stopped or begins to execute again. (MB300014 and MB300015 in MPM004, and H25.01)



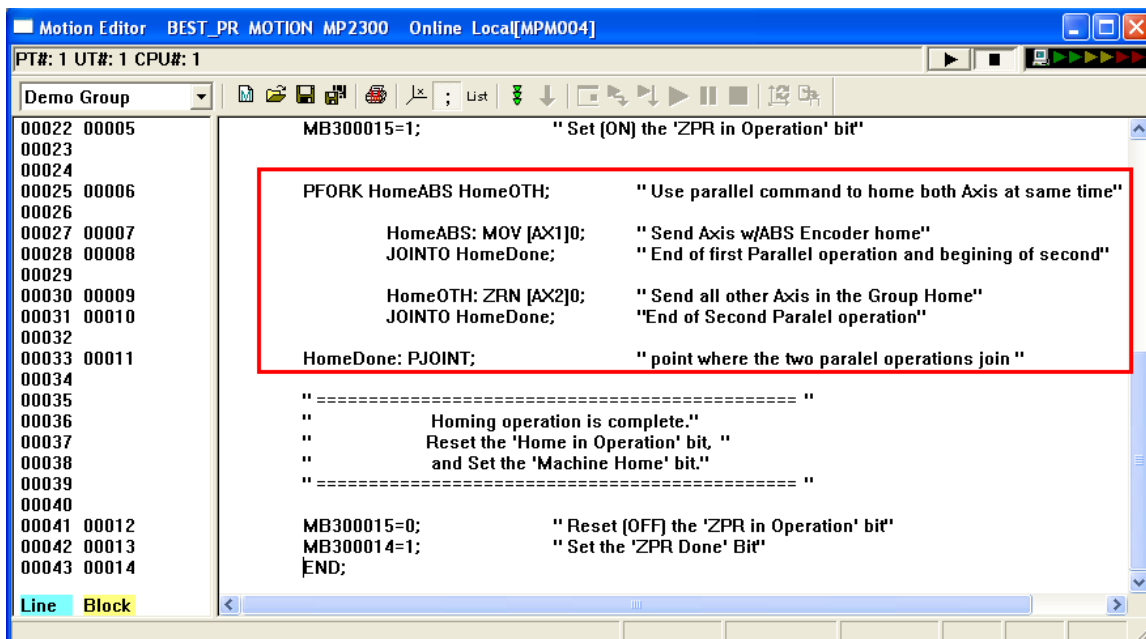
Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280

Rules for using a WHILE – WEND Loop in Motion programs.

- Example: An incremental move that repeats until the loop condition is satisfied.
- Code that takes more than one scan to execute may cause a watchdog alarm. Avoid this alarm by including the EOX instruction in the loop.

Rules for using parallel processing “PFORK –JOINTO - PJOINT”.

- Inside the parallel process only motion and global (M, O, I, S, C) register handshaking (MByyyyyb = 1 , IOW MByyyyyb = =1, MWzzzzz == 1234) instructions should be used. (Example: Home an axis with absolute encoder [using MOV] and an axis with an incremental encoder at the same time.)



```

Motion Editor BEST_PR MOTION MP2300 Online Local[MPM004]
PT#: 1 UT#: 1 CPU#: 1
Demo Group
00022 00005 MB300015=1; " Set [ON] the 'ZPR in Operation' bit"
00023
00024
00025 00006 PFORK HomeABS Home0TH; " Use parallel command to home both Axis at same time"
00026 HomeABS: MOV [AX1]0; " Send Axis w/ABS Encoder home"
00027 00007 JOINTO HomeDone; " End of first Parallel operation and beginning of second"
00028 00008 Home0TH: ZRN [AX2]0; " Send all other Axis in the Group Home"
00029 JOINTO HomeDone; "End of Second Paralel operation"
00030 00009 HomeDone: PJOINT; " point where the two paralel operations join "
00031 00010
00032
00033 00011 " ----- "
00034 " Homing operation is complete."
00035 " Reset the 'Home in Operation' bit, "
00036 " and Set the 'Machine Home' bit."
00037 " ----- "
00038
00039
00040
00041 00012 MB300015=0; " Reset [OFF] the 'ZPR in Operation' bit"
00042 00013 MB300014=1; " Set the 'ZPR Done' Bit"
00043 00014 END;
Line Block

```

Rules using motion program subroutines:

- Like any programming environment's subroutine, motion program subroutines should be used to recall a repeatable process. (Example: writing to Servopack Parameters.)

The screenshot displays the Motion Editor interface for a program named 'BEST\_PR MOTION MP2300'. The main program code is shown in a list with line numbers (00025 to 00072) and comments. Key comments include:
 

- "Target Position = Forcer Physical L ML04004 = ML04002 + OL801E + M"
- "REDUCE THE POSITIVE AND NEGATIVE TORQUE LIMITS"
- "Pn402 and Pn403 are positive and negative torque limits"
- "Set the Register in order to prepare for the write to Servopack parameter for positive torque limit"
- "Servo parameter Pn402 (positive torque limit) read or write"
- "Set the number of words in the data area of the parameter to read/write"
- "Positive torque limit 20%"
- "Write the Parameter"
- "Set the Register in order to prepare for the write to Servopack parameter for negative torque limit"
- "Servo parameter Pn403 (negative torque limit) read or write"
- "Set the number of words in the data area of the parameter to read/write"
- "Negative torque limit 20%"
- "Write the Parameter"
- "MOVE THE AXIS AND FORCE THE PARTS TOGETHER"
- "positioning to force parts together"
- "RESTORE THE POSITIVE AND NEGATIVE TORQUE LIMITS"
- "Pn402 and Pn403 are positive and negative torque limits"
- "Set the Register in order to prepare for the write to Servopack parameter for positive torque limit"
- "Servo parameter Pn402 (positive torque limit) read or write"
- "Set the number of words in the data area of the parameter to read/write"
- "Positive torque limit 300%"
- "Write the Parameter"
- "Set the Register in order to prepare for the write to Servopack parameter for negative torque limit"
- "Servo parameter Pn403 (negative torque limit) read or write"
- "Set the number of words in the data area of the parameter to read/write"
- "Negative torque limit 300%"
- "Write the Parameter"
- "Dwell one scan"

 The subroutine window, titled 'MPS006 "Parameter Write Subroutine"', contains the following code:
 

- "Force and check that axis is in NO-OP"
- OW8008=0; "Force Axis to NO-OP by setting MCC register to 0"
- IOW IW8008==0; "Wait for confirmation of MCC feedback to 0 [NO-OP]"
- "Execute the Parameter write and wait for it to Complete"
- OW8008=18; "Servo Parameter Write Command Code = 18 [PRM\_WRT]"
- IOW IW8008==18; "Wait for confirmation of MCC feedback to 18 [PRM\_WRT]"
- IOW IB8009==0; "Command NOT busy?"
- IOW IB8009==1; "Complete"
- "Force and check that axis is in NO-OP"
- OW8008=0; "Force Axis to NO-OP by setting MCC register to 0"
- IOW IW8008==0; "Wait for confirmation of MCC feedback to 0 [NO-OP]"
- RET;

 Red arrows indicate the flow of execution: from line 00033 in the main program to the start of the subroutine, and from the end of the subroutine back to line 00072 in the main program. Yellow callout boxes provide context:
 

- Box 1: "Before the subroutine is called the main code prepares the operation. The parameter number and value to write." (points to lines 00033-00038)
- Box 2: "The subroutine is called." (points to the call instruction at line 00039)
- Box 3: "The subroutine does the parameter write." (points to the subroutine code)

Register Ranges to use and Why:

When using motion programs the programmer should not use Function blocks. Therefore, the function block's Reserved Data Area (RDA) is available for Bit Handshaking and data sharing between Motion Programs and ladder code.

MP900: MW00100~MW03999  
MP2000: MW30000~MW65535

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085  
(800) YASKAWA - Fax (847) 887-7280