CNC PLC, Macro and Skinning Programming Manual

Revision 20180820

Table of Contents

Introduction to the Manual

The PLC or Programmable Logic Controller is responsible for controlling outputs primarily based upon the state of inputs, but also for controlling outputs based upon time. The PLC Program is what is written to describe how the PLC reacts to the Inputs and when to cause Outputs to turn on or off.

This manual is for anyone trying to change or write a PLC program on a Centroid CNC11 or CNC12 system. A CNC11 system is one that is based on MPU11 hardware, while a CNC12 system is one that is based on MPU12 hardware. The Centroid CNC11 software works with a CNC11 system. The Centroid CNC12 software works with both a CNC11 and a CNC12 system. This manual assumes basic familiarity with PLC programming. Not every detail is explained. There are some definitions in Appendix H, but it is intended to clarify where there may be confusion between CNC11/12 PLC program meaning and a more general meaning. If you are unsure of what you are doing, please contact Tech. Support at support@centroidcnc.com and ask questions.

When modifying PLC programs, it is considered good practice to regularly make backups. Always make a report and store it in at least one location in case the program change needs to be reverted.

The manual explains the components that can be used to make a PLC Program and goes over the standard parts of PLC programs. There are several appendices including sample programs, detailed key mappings, differences in CNC10 and CNC11/12 PLC programs, and errors associated with the compiler. If you are experienced with CNC10 PLC programs be sure to read Appendix H about the differences between CNC10 and CNC11/12.

Conventions Used in this Manual

There are several text conventions used in this manual. The following list explains the most common ones.

- Code from PLC programs including System Variables and the various data types are in `Consolas font at 10 pt size`. An example is `SV_PC_VIRTUAL_JOGPANEL_ACTIVE`.
- Keyboard Keys are in **Arial font at 12 pt. size and bold**. An example is **ALT-Q**.
- Commands entered in the command line of a prompt window are in *`italicized Consolas font at 11 point size`*. An example of this is *`mpucomp ProgramName.src mpu.plc`*.
- On and `SET` are interchangeable, as are Off and `RST`.
- System Variable may be written as SV, which is interchangeable and means the same thing.
- PLC Program and program are used interchangeably and mean the same thing.
- Data Type and type are used interchangeably and mean the same thing.

- When a specific name for a Data Type is helpful it is used, but typically the direct name is used to remove confusion about what type is being used in a given example.
- Names of Data Types such as Memory Bits and Outputs are capitalized.
- Binary data is written in this document from Most Significant Bit (Msb) to Least Significant Bit (Lsb) on a Left to Right order when explaining how bits are moving around. This follows the typical convention in programming. Note that the PLC program reads and writes pure Binary data from Left to Right as well, but goes from Lsb to Msb. See BTW and WTB for more information.
- From a general PLC programming viewpoint, the term CNC11 software is the same as CNC12 software. The term “CNC software” denotes either CNC11 or CNC12. The same applies for MPU11 and MPU12, with the term “MPU hardware” denoting either MPU11 or MPU12 based hardware.

Compiling a PLC Program

The source code which you write or change must be converted to the format that CNC software uses internally. This is accomplished by compiling the program. The compiler is called mpucomp which is short for MPU compiler. The syntax for compiling a program is *`mpucomp.exe ProgramName.src mpu.plc`*. If the filename contains spaces, it must be surrounded with quotes. The first file is the source code (a text file) and the second file is the name of the output file (a text file). CNC software looks for a file named mpu.plc to load and execute as the PLC program, so the output file must be named exactly *`'mpu.plc,`* including the fact that it is is all lower case letters. Once the program is compiled successfully, the system should be powered off completely and powered back on again for the changes to take full effect. CNC software will display a warning when it detects the loading of a changed PLC program. Whether or not you can proceed without side effects depends upon what has been changed in the PLC program and the current state of the PLC system. When in doubt, power off.

PLC Program Statistics

The PLC program is constrained by certain factors and limits. The following list enumerates some of the more important ones. Pressing **ALT-I** on the main menu causes the Live PLC Diagnostic screen to appear. It shows the state of Inputs, Outputs, Stages, Memory Bits and Words as well as the Time that is being taken by execution of the PLC program.

- Presently there is a limit of about 80 Inputs and Outputs using a GPIO4D and four PLC1616ADD boards. DC3IOB systems can achieve a bit more than this. There is a hard system maximum of 768 Inputs and 768 Outputs in CNC software and MPU hardware. This limit does not apply to IO related to the Jog Panel.
- There are many powerful features that take a long time to execute and thus should be used sparingly. Usage of them is detailed below and recommendations against using them are included in the Operators section.

Language

Programming Conventions

It is helpful when debugging and reading a program to know what type a variable is without having to constantly search through the multiple uses of a name to get to the definition at the top of the program. Following is a table of basic suggestions of ways the types can be named to reduce confusion. Inputs and Outputs are typically named to indicate the purpose rather than applying an extra label to them. The basic idea is to put something like the code specific type name at the end of the declaration. Whether you put an underscore between the name and type, add the letters in all Caps, or capitalize the first letter is up to you.

Type	**Example Name**	**Comments**
Constants	AXIS_FLT_CLR_MSG	All Caps, end with MSG
Input	EstopOk	
Output	LubeOut	
Memory Bit	SpinFault_M	Alternatively use M or Mem
Word	Axis3FiberOk_W	Alternatively use W or Word
Double Word	BigCounter_DW	
Floating-point Word	SpindleRangeMultiplier_FW	
Double-Floating-point Word	PreciseNumber_DFW	
Timer	Fault_Clear_T	Alternatively use Timer
One-Shot	SlowFast_PD	PD is Positive Differential, meaning rising edge
Stage	InitialStage	Alternatively use STG
Fast Stage	CountSomething_FSTG	
PLC to CNC11 System Variable	DoToolCheck	SV_PLC_* System variables are not named, so the function is prefixed with an action word 'Do' or 'Select' and named for the function it tells CNC11 to do.
Keyboard Keys	Kb_a	Kb is short for Keyboard
M-Codes	M6	Start with Capital M

While technically no Stages are required to be explicitly used for a program to compile, they should be used. The benefits include allowing debug of small sections of code by turning off other stages, reduced Program running time by running only certain things all the time, and

compartmentalizing code for easier comprehension.

The compiler is not case sensitive so EstopOK, ESTOPOK, EsToPoK, and estopok are all references to the same Input. Convention states that capitalization be kept uniform for variable references, but it is not required by the compiler.

All lines should end after 79 columns to prevent printing issues and ensure readability on smaller monitors.

Stage names in the program are surrounded on lines above and below with a full width line of equal signs to designate the start of a stage. CamelCase is used to name Stages and the same naming should be used in the program. Type the Stage name 20 columns from the left.

Section titles within a stage have a full width line of minus signs, and also include the name of the section in that line or not as preference dictates. The name should start after 5 minus signs either way and optionally fill out the rest of the line with minus signs.

```
;-----INPUT DEFINITIONS-------------------------------------------------------
```

Section explanations longer than about 5 lines should have a full width line of minus signs before and after the section to show it is a block of comments. Also every line between the two bracketing lines should have a semi-colon to show it is part of the explanation.

Section explanations shorter than about 5 lines should just have a semi-colon in front of each line with no spaces.

```
;INP769 - INP784 encompass the MPU11 onboard input connections
;which are generally used for MPG and probing functions.
```

System Variables with names like `SV_STOP` should not be redefined to a Memory Bit in order to avoid obfuscation. There are some System Variables where the reading or writing of them should only occur once in a PLC program to prevent race conditions. In these cases, you will need a similarly named PLC variable, and a convention is to include the word temp or shadow in the name. In some debugging scenarios, a similarly named variable should include an ‘Echo’ prefix or suffix.

In the definition section the `IS` should line up with all others above and below in the section, not with the entirety of the defines. This increases readability while maximizing comment space. There should be two spaces between the longest variable name in the section and the `IS` keyword. Also, if possible, line up the comments after the definition.

Use only spaces in the PLC program, never use tabs unless you can figure out how to replace tabs with spaces in your editor. It is helpful to show non-printing characters as well to make sure trailing spaces are not causing extra wrapping of text.

Use only mono-spaced fonts such as Courier, Courier New, Consolas and MS Mincho.

The beginning of the actual program and end of the defines should be visibly demarcated with 5 blank lines before and after a title indicating the start of the program.

```
;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
;-----Program Start
;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
```

Defining Variables

The names of variables used in the PLC program are defined at the top of the PLC program only. Any definition after the first `IF...THEN` statement will cause an error at compile time. Any label may be used to refer to any of the data types, but there are conventions to make it easier to determine the type of data assigned to a variable. Example syntax for defining a variable is:

```
EstopOk    IS INP11
Lube       IS OUT2
```

In this example, `EstopOk` is the name given to the `INP` data type. Similarly, `Lube` is the name given to the `OUT` data type. In both cases, the keyword `IS` is part of the syntax for defining a name.

Data Types

The kind of information that a variable can hold is defined at compile time by the data type when it was declared. All of the types below can be used in a PLC program. Typically Words and Floating-point Words have enough precision to achieve the desired results versus Double Words and Double Floating-point Words.

Important Note

It is critical to understand when specific Data Types are updated during the execution of a PLC program. The execution of a PLC program occurs approximately 1000 times a second, but only PLC program code inside a fast stage (`FSTG` type) or outside of any stage is actually executed 1000 times a second. PLC program code inside a regular stage (`STG` type) is executed 50 times a second. The execution of a PLC program from top to bottom as it has been written is referred to as a PLC “pass”.

A question that quickly emerges when writing PLC programs is, “If I change the value of a variable in the PLC program, when does it actually take effect?” The answer to this question is that it depends upon the data type being changed. Timers, Inputs and Outputs are all buffered at the beginning of the program. This means that a snapshot is taken of the state of them and that image ***does not change during the pass of the PLC program***. In other words, Timers have the same value at the start of the pass as they do at the end. The same is true for Inputs and Outputs as far as the real-world physical state is concerned. The snapshot of the Inputs never change, but the image of the Outputs can be changed on any line and that brings us to the other category of when things update. Memory Bits, all Words, One-Shots, both kinds of Stages, System Variables, and the *image* of the Outputs are changed immediately and on the next line of the program will be at the value they were assigned to on the previous line.

Another important thing to note is that the Live PLC Diagnostic screen (**ALT-I)** does not show every transition of every variable. What you see is a snapshot of PLC data that is updated about 20 times a second.

Constant Definitions

It is often easier to remember a name for an error rather than the number associated with it. Defining a Constant allows for this ease of use. By convention Constants are put before Variable Definitions, but they can be put anywhere before the first `IF` statement. Typically, Constants are used for PLC message numbers, but they can be used for anything. Math is allowed when defining constants to help avoid mistyping a number. Parentheses are allowed to force correct math, but only for Integer values. Floating-point numbers cannot have math done on them or be used to create them. See the Syntax Errors section.

```
PI          IS 3.14159265358979323846264338327795
ASYNC       IS 2
SYNC        IS 1
MULTIPLIER IS 256
FAULT_MSG   IS (SYNC+5*MULTIPLIER)
```

Input – INP

Inputs are physical switches or buttons that can be either on or off. When defining an Input, it is written as `Limit_Switch IS INP2`. Inputs can be logically combined with Outputs, Memory Bits, Timers, One-Shots, Stages, Fast-Stages and System Variables that are bits using Logical Operators, but not Relational Operators. Limit switches, Jog Panel keys and Inverter signals such as fault or at speed are all examples of inputs. Analog voltage comes into the system, but it must be read into any of the Word type variables.

Output – OUT

Outputs are physical, real-world outputs such as relay contacts, relay driver signals, or analog signals that can be on or off. Analog voltages are converted from Bits in the PLC program to analog voltage at the header. They can be used to control other relays, lube pumps, spindle enable, inverter analog speed control, coolant, Jog Panel LEDs, etc. The standard way of specifying an output is `Lube IS OUT2` or `AutoCoolantLED IS JPO21`. Outputs can be logically combined the same way as Inputs.

Memory Bit – MEM

Memory Bits can be on or off and cannot directly affect anything outside the PLC program. These Bits are often used to make logic easier to read by combining many repetitive checks or keystrokes into one variable. MEM Bits are also used for debugging to store whether transient signals have occurred, whether to invert inputs based on parameter 178, etc. Memory Bits can be logically combined to other Bit type Variables with Logical Operators, but not Relational Operators.

Word – 32-bit – W

Words are Integer numbers that can be used to store error codes, parameter values, and System Variables that are Word sized themselves. Words are defined like `Error_Code IS W10`. Only the 12 words are visible on the Live PLC Diagnostic screen at one time. They can be compared with other Word types, System variables that are Integer numbers rather than one bit, and Timers. Relational Operators are allowed on any Word type variable, but Logical Operators are not. Doing a Relational comparison produces a result that can be used in a Logical Operator, however. An example of this is `IF (W1 > W2) || MEM1 THEN SET OUT6`. This statement tests whether `W1` is greater than `W2` first. If that is true or `MEM1` is true then the output is turned on. Words can hold values from -2147483648 to 2147483647.

Double Word – 64-bit – DW

Double Words are Integer numbers just like Words, but can hold values from -9223372036854775808 to 9223372036854775807. A Double Word is defined like `BigNumberDW IS DW3`. For most PLC programs, Double Word types are not necessary. Double Words are otherwise exactly the same in usage as Words.

Floating-point Word – 32-bit – FW

Floating-point Words are real numbers that can store fractional values from 2^-149 to 2^129. It is typically precise enough for any operation in a PLC program. It is defined like `SpindleDACFW IS FW1`. Precision problems can occur if comparing very very large and very very small numbers, but typically this is not a concern. Floating-point Words have the same comparison ability as Words.

Double-Floating-point Word – 64-bit – DFW

Double-Floating-point Words are real numbers that can store fractional values from 2^-1074 to 2^1022. They are more precise and can compare bigger numbers with smaller compared to Floating-point Words. In general is is not advised to use this type at all due to significant time required to do any calculations. The definition of a Double-Floating-point Word is `PreciseNumberDFW IS DFW1`. `DFW`s are compared exactly like Words.

Timer – 32-bit – T

Timers are counters with the special ability to be compared with both Relational and Logical Operators. This means that you can check `IF T1 THEN SET OUT1` to see if the Timer has reached its set point or `IF T1 > 1000 THEN SET OUT2` to see if the Timer has counted past 1000 milliseconds (one second). Timers are initialized with a 32-bit positive Integer number that is interpreted as the number of milliseconds to count before evaluating to true when checked with Logical Operators.

The value is typically stored in the Timer during the `InitalStage`. To start a Timer counting use `SET T1`. To reset the Timer so that it is waiting to count again use `RST T1`. Note that you do not need to store a value into a Timer each time it is `RST` unless you want to change the value to

which it counts.

Timers evaluate to true after reaching the stored time until they are reset. The actual count of the Timer continues to climb until it gets to about 10 seconds before the max value that Timers can hold. This means that Timers can be checked to see if they are over a certain value with Relational Operators.

Timers can count up to 2,147,473,646 milliseconds (24 days, 20 hours, 31 minutes, and 13.646 seconds). Be aware that if you are trying to time something longer than that you will need to come up with a scheme for tracking the rollover. A simple option for counting days is to set the Timer to 1 day worth of ms (24 hrs. * 60 min. * 60 s = 86400000 ms). When the Timer expires, increment a Word day counter and then RST and SET the Timer to begin counting again. See Application Examples for how to setup a Timer for counting days. The following example sets the Timer to one second then SETs the Timer if MEM1 is closed. When T1 counts to 1000, an Output is turned on and the Timer is RST. If the Memory Bit is set next time through the PLC program the Timer will be SET again. The first time the Timer value has counted past 10 a Memory Bit will be turned on.

```
T1 = 1000                    ;Set the value that the timer counts up to 1 second

IF MEM1 THEN SET T1          ;start the Timer counting

IF T1 THEN RST OUT2, RST T1 ;if 1 second has elapsed turn off the output and reset the
                            ;timer so it can be started again.

IF T1 > 10 THEN SET MEM50    ;if the timer has counted past 10 ms set MEM50
```

One-Shot – PD

One-Shots or Positive Differentials are used to detect the first rising edge of an event occurring. A One-Shot can only be turned on and off using a Coil. Because of this a One-Shot should only ever be SET/RST on one line of the PLC Program. It can be checked anywhere, but not SET/RST. Once the PD has been SET by the IF test, the conditional section must evaluate to false and thus RST the PD before it can be used again. This means that you cannot hold a button down and cause the One-Shot to trigger more than once. Using this in combination with Debounce allows safer detection of key presses to prevent multiple actions when only one is intended. One-Shots are often used on Jog Panel keys and M-Codes. One-Shots may not be desired in certain circumstances such as the Override +/- buttons for Spindle Speed. Typically one wants to hold down the button and have the Override value change as long as the button is held down. An example definition and usage of a One-Shot follows.

```
KeyPressPD IS PD1              ;define the One-Shot

IF JPI1 THEN (KeyPressPD)      ;if Jog Panel Input 1 is pushed, set the One-Shot

If KeyPressPD THEN SET MEM300 ;if the One-Shot is set, turn on a memory bit
```

Stage – STG

Stages are useful in many ways. First, they are used to break up different sections of the program to allow easier debugging and testing. Conventional programming dictates that there is at least an InitialStage and a MainStage in any PLC program, but Stages are not required to

be explicitly used at all. Turning a Stage on and off is just like the process for Outputs.

```
IF 1==1 THEN SET STG1 ;Turn on a Stage all the time

IF M6 THEN SET ATC_Main_Stage ; turn a Stage on when a tool change is called
```

When a Stage is RST the PLC executor does nothing with the logic inside the Stage. This means that whatever Variables are SET or RST when the Stage itself is RST maintain that state unless they are modified somewhere else in the program.

Warning: No checks are made against resetting any stages. It is possible to RST every stage and have nothing executed in the PLC program. This requires a system power off and reboot.

The InitialStage is often used to setup any timers used and the state of tools for ATC machines. At the end of the InitialStage a JMP should be called to the MainStage. The InitialStage should never be SET again. To troubleshoot a problem, Stages could be turned off to narrow down the problem section. Also, Stages are usually used to break up the steps of a tool change sequence and various M-Codes so that they are not executed or checked every time through the PLC program and for debugging. Stages are executed at the standard rate of PLC program execution which is 50 times per second.

Warning: Typically Coils should not be used in Stages that are turned on and off, especially if you come from a CNC10 PLC programming background. This is because CNC10 PLC typical coil behavior will not occur if a Stage is off. This means that One-Shots should also not be used in Stages that may be turned off. Odd effects can occur with One-Shots in Stages if you do not know what to expect. If, for example, a One-Shot is used to RST a Stage, the One-Shot will ***not*** be reset the next time through the PLC program, but will instead wait for the next time the Stage is SET and the Coiled One-Shot is scanned over. This will, instead of setting the One-Shot as might normally be expected, cause the One-Shot to be RST finally and then require the One-Shot to be triggered again in the ***next*** PLC Program pass, if the Stage is still SET.

Timer counting is not affected by a Stage being RST. This means that if you SET a Timer in a Stage and then RST the Stage the Timer still counts and can be checked outside of the Stage and the expiration of the Timer will be accurate.

If you want to use a One-Shot to exit a stage, say from a button push, then you should put the One-Shot in coils again to make sure it is turned off on the line that resets the Stage. The following example illustrates the concept. With this example, if the Jog Panel key is pressed when the Stage is executed, the Stage will always be reset on the first pass through.

```
;============
STG1
;============
IF JPI1 THEN (PD1)              ;push a Jog Panel button to trigger a One-Shot
IF PD1 THEN (PD1), RST STG1     ;accessing the PD again causes it to get RST, Stage is RST
```

Fast Stage – FSTG

Fast Stages behave exactly like Stages except that they are executed at up to 1000 times per

second. These should be used only when extremely precise timing is necessary.

Warning: Fast Stages do not interrupt the operation of normal speed stages, so if the normal PLC program takes longer than 1 ms to execute completely, then the Fast Stages will not execute at the stated rate. Make sure that the standard timings are below 1 ms to get the fastest execution time. The average time can be found in the PLC Diagnostic screen.

CNC to MPU System Variable – SV_*

These are System Variables that are `SET` or `RST` by CNC software. There are both Word and Bit type SVs. Both CNC software and the PLC program can read these System Variables. Convention states that they should be typed in all caps. For a complete list of System Variables and their uses see Appendix D.

PLC to CNC System Variable – SV_*

These are System Variables that are `SET` or `RST` by the PLC Program. There are both Word and Bit type SVs. Both CNC software and the PLC program can read these System Variables. Convention states that they should be typed in all caps. For a complete list of System Variables and explanation of usage see Appendix D.

Keywords

Keywords are reserved words in the PLC programming language that cannot be used except for the defined purpose. Often attempting to use the keyword in an undefined way will cause the program to fail compilation. In example code they will always be capitalized by convention and should not be confused with constant defined variables or System Variables.

Defining variables – IS

This keyword is used to setup labels for all the data types. It is only used in the definition section at the top of the program. It is not used in the actual PLC program that gets executed. When the program is compiled all the labels are replaced with what they refer to from the definition section of the PLC program. For example the E-stop Input is defined like:

```
EStopOK IS INP11.
```

`IS` is used on every data type to define a name for that variable. There is also a built in functionality for defining constant data to have a name. Math can be done in the definition and previous definitions of constants can be used as long as the entire assignment is in parentheses. An example is:

```
DEFINED_CONSTANT IS (1+2+5*7)
SECOND_CONST     IS (DEFINED_CONSTANT*10)
```

Conditional Statement – IF/THEN

The Conditional Statement is used for every line of the PLC Program that can be executed

and is synonymous with a rung in ladder logic. The first IF/THEN in a Program defines the start of the PLC program and the end of the Variable Definitions. The part of the line between the IF and the THEN must evaluate to a boolean value (true or false). If it cannot be resolved to a boolean value then an error is thrown at compile time. This means that Words must be checked with Relational Operators and a Word itself cannot be the test for truth. IF W1 > 10 THEN (OUT2) is a valid test whereas IF W1 THEN (OUT2) is not. There can be as many conditions on the truth of a Conditional Test as you need. They are all separated by Logical Operators such as OR(||) and AND(&&).

There is no ELSE Keyword, but the functionality can be achieved by copying the previous test and putting parentheses around it with a NOT symbol in front of it.

The following table defines what can be done in the test part of the Conditional Statement. While the statements are legal it does not mean that they are typically used. For example, typically Stages are not checked in IF/THEN statements.

Data Type	Usage
Input	IF INP50 THEN (OUT50)
Output	IF OUT50 THEN (MEM50)
Memory Bit	IF MEM40 THEN SET STG2
Stage	IF STG2 THEN (OUT4)
Fast Stage	IF FSTG1 THEN (OUT1)
Word	IF W1 > 156 THEN (OUT30)
Double Word	IF DW2 > 4000000000000 THEN (MEM300)
Floating-point Word	IF FW1 > 5.432 THEN SET OUT3
Double-Floating-point Word	IF DFW1 > 8900.983201293 THEN (OUT80)
SV bit	IF SV_ENABLE_AXIS_1 THEN (MEM70)
SV Word	IF SV_PC_CYCLONE_STATUS_2 == 4 THEN (MEM65)
One-Shot	IF PD2 THEN SET OUT7
Timer	IF T1 THEN SET OUT 5 IF T1 > 4000 THEN SET OUT 6

Print Message – MSG

Printing a message is useful when the user needs to know about status changes. It is often used in ATC PLC programs to indicate what part of the tool change process is being executed. Error reporting is the other key usage of the PLC messaging functionality. There are two types of messages that can be used with the PLC Message functionality, Synchronous and Asynchronous. Printing synchronous messages is done in its own Stage that is SET only when a message needs to be printed. This Stage should be the last Stage in the PLC program. Asynchronous messages are usually printed right inline with the rest of the code. Synchronous messages are displayed only when the SV_STOP System Variable is SET and Asynchronous messages are printed immediately. There is not a practical limit to the number of messages that can be defined for usage.

The command to send a message is `IF 1==1 THEN MSG W1` where `W1` stores a correctly setup message value. A Word variable must be used with the `MSG` command.

Only one message can be displayed per pass of the PLC program. This explicitly does not mean that one each Async and Sync message can be displayed per pass. There is no queue of messages that get buffered and sent out eventually. If a new message is sent, it overwrites the previous one. Effectively, the last MSG command in the program is what is displayed on screen, if it has changed.

Worth noting is the fact that once a particular Synchronous or Asynchronous message has been sent, a different message number must be sent of the same type (Sync or Async) before that original message can be sent again. For Synchronous messages, in addition to sending a different message number, it must be a different nonzero message number. For example, if you send a Lube Fault Synchronous message, when that clears you must send another different (and nonzero) Synchronous message before the Lube Fault message can be displayed again. This means that if the Lube Fault message is displayed and then the fault is cleared, but no new message is sent, if Lube Fault occurs again, the PLC program will be in the fault state, but there will be no message displayed. It is very important to avoid this case as it will look as though there is no error, but CNC11 will not start a job due to `SV_STOP` being `SET`. It can be debugged by putting the Word variable in the first twelve words so that they are displayed on the first PLC Diagnostic screen.

CNC11 takes the value sent via the `MSG` command and looks in the plcmsg.txt file for the selected message and prints it to the screen. The method of formatting this value is to start with a 1 or 2 for Synchronous or Asynchronous respectively and then add the message number times 256. An example table is shown below. The Word Value column in the following table is what should be stored in the Word to be sent out with the `MSG` command.

Message Number	Type	Word Value	Notes
1	Synchronous	257	1 + 1*256
2	Asynchronous	514	2 + 2*256
25	Synchronous	6401	1 + 25*256
50	Asynchronous	12802	2 + 50*256

It is easiest and less error prone to write the message number once by defining a Constant to store to a word before messaging it out. Below are the defines used in both Asynchronous and Synchronous messages for the example program.

```
INP13_GREEN_MSG IS (2 + 1*256)   ;258
INP13_RED_MSG   IS (2 + 2*256)   ;514
INP14_GREEN_MSG IS (1 + 3*256)   ;769
NO_SYNC_MSG     IS (1 + 99*256)  ;25345
NO_ASYNC_MSG    IS (2 + 100*256) ;25602


EStopOk         IS INP11
```

```
Async_I         IS INP13
Sync_I          IS INP14

Async_O         IS OUT13
Sync_O          IS OUT14

Sync_Cleared_M  IS MEM1
Stop            IS MEM2

Sync_W          IS W1
Async_W         IS W2

InitialStage    IS STG1
MainStage       IS STG2
SetError        IS STG3

;===============================================================================
                    InitialStage
;===============================================================================
;setup default Word values
IF 1==1 THEN Sync_W = NO_SYNC_MSG, Async_W = NO_ASYNC_MSG,
             RST InitialStage, SET MainStage

;===============================================================================
                    MainStage
;===============================================================================
IF !EStopOk THEN SET SV_STOP
IF SV_STOP THEN (Stop)

;prevent sync messages from showing up by resetting SV_STOP
IF !EStopOk && Sync_Cleared_M THEN RST Sync_Cleared_M, RST Sync_O, Sync_W = NO_SYNC_MSG
IF EStopOk && !Sync_Cleared_M THEN RST SV_STOP

;sync
;--if the Input is green, set the Sync message and override the Async messages
IF Sync_I THEN Sync_W = INP14_GREEN_MSG, SET SV_STOP, SET SetError, SET Sync_O
```

```
IF !Sync_I && Sync_O THEN SET Sync_Cleared_M


;async
;--show a message if the Input changes
IF Async_I  THEN Async_W = INP13_GREEN_MSG, MSG Async_W, SET Async_O
IF !Async_I THEN Async_W = INP13_RED_MSG  , MSG Async_W, RST Async_O


;=============================================================================
                     SetErrorStage
;=============================================================================
IF 1==1 THEN MSG Sync_W
;if the message has been cleared, reset this stage to allow Async messages
IF Sync_W == NO_SYNC_MSG && Sync_Cleared_M THEN RST SetError
```

The sample plcmsg.txt file use for the above example is:

```
1   9001 Input 13 Green
2   9002 Input 13 Red
3   9003 Input 14 Green
99  9099 No SYNC Message
100 9010 No ASYNC Message
```

plcmsg.txt

The plcmsg.txt file contains a list of all the messages that the PLC can send to CNC11. This facility is used to notify the user of status changes and fault conditions. The typical messages should not be overwritten by new custom messages, rather new numbers should be added. The format for each line of the plcmsg.txt file is as follows.

```
MessageNumber MessageLogNumber Message
```

There are three fields separated by one space each that must be setup for a line to be valid and usable. If the line is not formed correctly, you will not know it until the message is trying to display. The MessageNumber field is exactly the same number as the Message Number in the above table. The MessageLogNumber causes the printed message to be put in the msglog.txt file so that problems can be diagnosed by Tech. Support. The 9xxx series messages are reserved for PLC program usage. Make sure the Log Level is set to 4 in parameter 140 to ensure the messages are logged. Message is the useful text that will be printed along with the MessageLogNumber. It should contain a pithy message that informs the user about the change that occurred. All text to the end of the line is printed so no comments are allowed in this file. The standard plcmsg.txt file is listed below.

```
1   9001 PLC Execution Fault
5   9005 Axis 1 Communication In Fault
6   9006 Axis 2 Communication In Fault
7   9007 Axis 3 Communication In Fault
```

```
8   9008 Axis 4 Communication In Fault
9   9009 Axis 5 Communication In Fault
10  9010 Axis 6 Communication In Fault
11  9011 Axis 7 Communication In Fault
12  9012 Axis 8 Communication In Fault
13  9013 Axis 1 Communication Out Fault
14  9014 Axis 2 Communication Out Fault
15  9015 Axis 3 Communication Out Fault
16  9016 Axis 4 Communication Out Fault
17  9017 Axis 5 Communication Out Fault
18  9018 Axis 6 Communication Out Fault
19  9019 Axis 7 Communication Out Fault
20  9020 Axis 8 Communication Out Fault
21  9021 Axis Faults Cleared
22  9022 PLC Communication In Fault  (Fiber 3)
23  9023 PLC Communication Out Fault (Fiber 1)
24  9024 PLC Faults Cleared
34  9034 FAULT! REMOVE PROBE FROM SPINDLE!!!
35  9035 KEYBOARD JOGGING DISABLED
36  9036 LUBE FAULT
37  9037 PROBE TRIPPED WHILE JOGGING
38  9038 SPECIFIED SPIN SPEED < MIN SPIN SPEED
39  9039 Software Ready Fault
50  9050 Auto Coolant Mode
51  9051 Manual Coolant Mode
99  9099 Message Cleared
100 9100 BAD MESSAGE VALUE
```

System Variables – SV_...

System Variable names cannot be used verbatim as constant value labels, variable names, or as the beginning of either. Removing the underscores from the name, for example, gets around this problem. Attempting to do so will cause the program to fail at compile time.

Data Type Name

None of the names of Data Types used in the PLC program are allowed to be used as constant value labels or variable names. This means that `INP1` is invalid as is just `INP`. Attempting to do so will cause the program to fail at compile time.

Indexes – Data Type[Data Type or Constant]

Using the Index ability is useful when writing generic programs that allow users to set an Input or Output that they want a function to occur at in parameters. It is very important to check for Index Out of Range errors if this functionality is used. The default value for the PLC Program parameters is 0.0 and all Data types start at 1, so an Index Out of Range error would occur right away. This cannot be used with System Variables. In the following example Parameter 171 is checked to make sure it is within a certain range, then an Output number is set based on the information it finds.

```
p171_W       IS W1
```

```
LubeOut_W    IS W2
;read the parameter
IF 1==1 THEN p171_W = SV_MACHINE_PARAMETER_171
;make sure the parameter is in range, if not default value is used
IF ( p171_W <= 0) || ( p171_W >= 10) THEN LubeOut_W = 2
IF !(( p171_W <= 0) || ( p171_W >= 10)) THEN LubeOut_W = p171_W
;set output based on parameter
IF SV_PROGRAM_RUNNING || SV_MDI_MODE THEN (OUT[LubeOut_W])
```

Range Selection – '..'

This functionality allows setting or resetting of Bit type Variables including System Variables. Two dots are placed a space apart from the beginning variable and ending variable. All variables between and including the two book-end variables are either `SET` or `RST`, regardless of whether they are being used for anything else or if they have a label. This can be useful in error conditions if you want to turn off all Outputs or Stages. Usage is as follows.

```
IF JPI1 THEN (PD1)
IF PD1 THEN RST OUT1 .. Out80


IF JPI2 THEN (PD2)
IF PD2 THEN SET OUT80 .. OUT1
```

DUMP

The DUMP command causes the PLC program to dump the values stored in all of the first 64 Words, Double Words, Floating-point Words, and Double-Floating-point Words to debug_dump0.txt in the cncm or cnct directory. DUMP should only be used sparingly and then only when debugging the initial implementation of a PLC program because of the cost of writing data to a file on the hard disk. This is useful for making sure Floating-point variables are correct. The PLC Detective utility and the Live PLC IO display in CNC software have the ability to monitor and display The first 44 W and FW types and the first 11 DW and DFW types. In CNC12 v4.14, the first 88 W types can be displayed.

An example of using this is as follows.

```
IF JPI1 THEN (PD1)
IF PD1 THEN DUMP
```

Operators

The operators in a PLC program are used to compare data, to set one piece of data equal to another, invert the data, etc. There are both unary and binary operators. The unary operators only require one variable or piece of data to operate on. An example of this is inverting a memory bit like this `!MEM1` or turning a Memory Bit on like `SET MEM1`. A binary operator requires

two pieces of data. Examples of using a binary operator are `W1 >= W2` and `W3=W4`. There are no Bitwise Operators at present. This means that bit masking cannot be done by ANDing or ORing constant values with a Word. It can be done by setting and resetting bits and shifting as well. The following operators are all valid in a CNC11 PLC program.

Assignment – =

The equals sign is used to set a Word or Timer on the left of the equals sign to a Word, Timer or numerical value on the right of the equals sign. Some examples are listed below.

```
IF 1==1 THEN W1 = 10   ;Word1 is set to an integer value of 10
IF 1==1 THEN T1 = W1   ;Timer1 is also set to an integer value of 10 representing 10 ms
IF 1==1 THEN FW1 = 2.5 ;Floating-point Word1 is set to 2.5
```

Set – SET

SET turns on any of the Bit variables. The bit value is set to 1 and evaluates to true when checked. That is to say that any of the Outputs, Memory Bits, Timers, Stages, Fast-Stages and System Variables that are bits can have this keyword used on them. One-Shots cannot be `SET`. An example of using this is `IF 1==1 then SET MEM1`.

Data Types that can be used with SET	Example of using SET
Memory Bits	`IF 1==1 THEN SET MEM2`
Outputs	`IF 1==1 THEN SET OUT2`
Inputs	`IF 1==1 THEN SET INP2`
Timers	`IF 1==1 THEN SET T2`
Stages	`IF 1==1 THEN SET STG2`
Fast Stages	`IF 1==1 THEN SET FSTG2`
One-Shots	`IF 1==1 THEN SET PD2`

Reset – RST

Reset turns off any of the Bit variables. The bit value is set to 0 and evaluates to false when checked. That is to say that any of the Outputs, Memory Bits, Timers, Stages, Fast-Stages and System Variables that are bits can have this keyword used on them. One-Shots cannot be `RST`. An example of using this is `IF 1==1 then RST MEM2`.

Data Types that can be used with RST	Example of using RST
Memory Bits	IF 1==1 THEN RST MEM2
Outputs	IF 1==1 THEN RST OUT2
Inputs	IF 1==1 THEN RST INP2
Timers	IF 1==1 THEN RST T2
Stages	IF 1==1 THEN RST STG2
Fast Stages	IF 1==1 THEN RST FSTG2
One-Shots	IF 1==1 THEN RST PD2

Output Coil – ()

Output Coils are used to SET or RST any bit output based on the conditions before the THEN. If the test in the IF is true the output is SET, whereas if the test is false then the output is RST. The use of parenthesis does not constitute Output Coils unless they are used to the right of THEN on any program line. Output Coils cannot be used on Words. Be careful when using Coils because they can cause logical problems in your program. Do not put a variable in Coils on one line and then try to SET or RST it somewhere else in the program as well because it will change while moving through the PLC program and may have surprising results. If you are going to use a variable in Coils, all of the logic to turn it on or off must be on the same line to avoid trouble. Coils cannot be used on Timer Data types to start them counting because they are generally guaranteed to be turned off again on the very next pass of the PLC program. Some examples are shown below to illustrate some Coil concepts.

```
;basic Coil usage
IF 1==1 THEN (OUT1) ;always turn on OUT1
IF MEM1 THEN (OUT2); SET OUT2 if MEM1 is SET and RST OUT2 if MEM1 is RST

;potentially problematic Coil usage
IF MEM1 || INP5 THEN (OUT4) ;OUT4 is guaranteed to be SET or RST by this line
IF MEM3 THEN SET OUT4       ;OUT4 may be SET by this line if MEM3 is SET
IF INP10 THEN RST OUT4      ;OUT4 may be RST by this line if INP10 is SET

;better usage of Coil
IF (MEM1 || INP5) || MEM3 && !INP10 THEN (OUT4) ;all logic combined
```

Jump – JMP

Jump RSTs the current Stage and SETs another one. See Stages for the effects of turning on or off a stage. Execution does not jump around in the PLC program as in Assembler, but typically the stages are written one after the other so in essence it will move to that one. A sample usage of Jump is IF 1==1 THEN JMP STG3. If a JMP is called such that the current Stage is the stage being jumped to, then the Stage will still be set next time through the PLC

program. Jump can only be used on Stage variables. It is not advised to `JMP` out of the `MainStage`, whereas it should be done out of the `InitialStage`. If you jump out of the `MainStage` and no other stages are set, you must exit CNC11 and reboot the MPU11.

```
;typical usage

InitialStage

IF 1==1 THEN JMP MainStage ;InitialStage is RST and MainStage is SET.
```

Basic Math Operators – *, \, +, -, %

These are the basic four math functions, multiplication, division, addition and subtraction plus the Modulus operator. They are all binary operators. These operators are allowed to be used on Word types only. The modulus operator, %, is used to find the remainder of the division of two numbers.

Note that the calculation is done and then changed to the appropriate type when the value is assigned to a variable. Assignment of floating-point values to Integer Words results in the decimal point value being truncated. This means that `IF 1==1 THEN W1 = 2.5*1` will result in W1 being set to 2. The following examples illustrate usage of some of the operators.

```
;multiplication

IF 1==1 THEN W1 = 15*2; W1 is set to 30

IF 1==1 THEN FW1 = 0.5*; FW1 = half of current encoder counts for axis 1

;division

IF 1==1 THEN W2 = 128 / 2; W1 is set to 64

;Modulus

IF 1==1 THEN W3 = 15 % 2; W3 is set to 1 because 15/2 = 7R1

IF 1==1 THEN W4 = 128 % 15; W4 is set to 8 because 128/15 = 8R8
```

Relational Operators – <, >, <=, >=, !=, ==

Relational Operators are only valid when comparing Words, Word type System Variables and Timers. Bit type variables cannot be compared with these operators. A relational operator compares two variables of the same type and results in a 1 or 0 as the output.

Be aware that checking Timers with Relational Operators does not indicate anything about whether the Timer has expired or not. It merely compares against the current count of ms since the Timer was Set.

Less Than or < checks to see if the variable on the left is less than the value on the right to evaluate to true.

Less Than Or Equal or <= checks to see if the variable to the left is less than or equal to the variable on the right to evaluate to true.

Greater Than or > checks to see of the value on the left is greater than the value on the right to evaluate to true.

Greater Than Or Equal checks to see if the value on the left is greater than or equal to the value on the right to evaluate to true.

Not Equal or != checks to see if two variables are not exactly equal to evaluate to true. Note that Floating-point != checks will most likely fail on calculated values due to rounding error. >= or <= should be used instead to check Floating-point numbers.

Is Exactly Equal To or == checks to see if two variables are exactly equal to each other to evaluate to true. The same warning applies to == as != about Floating-point numbers.

The following examples show how a sample PLC program line looks with a Relational Operator on it.

```
IF 1==1 THEN W1 = 10, W2 = 10

IF W1 > W2 THEN (OUT1)    ;Out1 is RST because W1 is not greater than W2

IF W1 == W2 THEN (OUT2)   ;Out2 is SET because W1 is equal to W2
```

Logical Operators – !,&&, ||, XOR or ^

Logical Operators are used to compare or change the state of Bit type variables including System Variables.

&& is the operator for AND. This means that both sides of the operator must be true to have the statement evaluate to true. The following truth table shows all the options for evaluating a binary expression.

Left Side	Right Side	Result
0	0	0
0	1	0
1	0	0
1	1	1

|| is the operator for OR. This means that only one side of the operator needs to be true for the statement to evaluate to true.

Left Side	Right Side	Result
0	0	0
0	1	1
1	0	1
1	1	1

XOR or ^ are the operators for Exclusive OR. This means that one of the sides of the operator must be true and the other false to have the statement evaluate to true.

Left Side	Right Side	Result
0	0	0
0	1	1
1	0	1
1	1	0

The ! is a unary operator for NOT that inverts the state of a bit whether it is true or false.

Bit Value	Result
0	1
1	0

Logical Operators cannot be used on Word Type variables. The results of multiple Relational checks can be combined with Logical checks for more complex statements, however. All of the following lines are valid PLC code. Often it increases readability to use parentheses around conditions to ensure correct interpretations.

```
IF MEM1 && INP2 THEN (OUT1)
IF (W1 > W2) || !MEM4 THEN (OUT5)
IF !MEM1 && INP2 || STG1 || FSTG1 && OUT3 XOR PD1 && T3 XOR SV_PC_POWER_AXIS_1 THEN (OUT6)
```

Convert to Word – BTW

BTW takes a range of Inputs, Outputs or Memory Bits and does a binary to decimal conversion, storing the result in a Word. The default number of bits is 8, but a value of 1 to 32 is allowed. The first Bit is treated as the Least Significant Bit and the next bit is the next highest number of Input, Output, Memory Bit. For example `MEM1` to `MEM8` are set, from lowest to highest, as 0110 0100 where 0 is red and 1 is green. When `BTW` is executed the number is converted as 2+4+32 = 38 decimal. Note that the number of bits specified is a count number to use rather than a specific Bit number to convert up to and including. This means, starting at Bit 0, how many of the bits should be converted. The following example will setup the bits as the example above in code.

```
;setup the binary information
IF 1==1 THEN RST MEM1, SET MEM2, SET MEM3, RST MEM4
           , RST MEM5, SET MEM6, RST MEM7, RST MEM8
;evaluate the Memory Bits as a Binary number and convert it to Decimal, storing in W1
IF 1==1 THEN BTW W1 MEM1
```

Convert to Binary – WTB

WTB converts a Word type variable to Binary and writes to Outputs or Memory Bits. By default the lowest 8-bits of Binary are written. The lowest bit goes into the first bit and so on up to the highest Bit. An Integer number from 1-32 can be specified to print from 1 to 32 of the

Binary bits to either Memory or Output locations.

Note that the number of bits specified is a count number to use rather than a specific Bit number to convert up to and including. This means, starting at Bit 0, how many of the bits should be converted.

Note that the Binary bits will write to the next consecutive bits regardless of any other meaning in the PLC program. If you do a default WTB in a Memory section that only has 5 free bits, three of them will change something you likely do not want to change.

For example the Word W1 has a value of 12345 in decimal notation and 11000000111001 in Binary notation. A standard `WTB W1 MEM1` will print 00111001 with the Least Significant Bit written to `MEM1` and the Most Significant Bit written to `MEM8` as seen on PLC Diagnostics. In the first example the default `WTB` is used and 8 bits will be written out. In the second example 12 bits will be written to `OUT17` to `OUT28`.

Data Types that can be used with WTB	Example using WTB
Memory Bits	`IF 1==1 THEN WTB W5 MEM10`
Outputs	`IF 1==1 THEN WTB W5 OUT17 12`

Convert to Binary Coded Decimal – BCD

BCD is used primarily for interfacing with external devices that need information in Binary Coded Decimal format. BCD is different than Binary in that each number in a decimal number is converted to a 4-bit binary number.

This means that to represent the decimal number 125 in BCD the number must be broken up into the ones, tens and hundreds positions and each of those numbers is converted to a Binary number. The Binary numbers for 1, 2, and 5 are 0001 0010 0101 which is the BCD version of 125.

The following table shows the conversion for the Decimal and BCD values. To use this feature on outputs, four outputs must be used for each place in decimal that the external device will be looking for. In the case of an ATC with 24 tools two sets of four outputs must be used to represent all possible numbers. An example of using the BCD command is: `IF 1==1 THEN BCD W2`. The Word can then be sent to the correct Outputs with the `WTB` command.

Note that the largest Decimal value that can be converted is 8 digits because only 32-bit Integer Words can be used for the BCD command. This gives 32-bits divided by 4 bits gives 8 decimal places. This means that the BCD of a number greater than 99,999,999 is invalid and undefined. For reference, the bits higher than 8 places are truncated and cannot be recovered.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Convert from Binary Coded Decimal – BIN

BIN converts a Word value that is in BCD format to standard Binary format. When the Word is displayed on the PLC Diagnostic Screen it is shown as a Decimal number. The `WTB` command can be used to then print the Binary version of the number to Memory Bits or Outputs if desired. A simple use case of the `BIN` command is `IF 1==1 THEN BIN W1`.

Note that there is no checking on the BCD number to determine if it is a valid BCD number. There are many invalid BCD numbers in each group of four digits, which could be checked in a PLC program. In every group any BCD number greater than 1001 or 9 is invalid because in decimal notation each place can only hold 0-9. For example a BCD number 57004 is completely invalid because in Binary it is 1101 1110 1010 1100 all of which are above 1001.

One way to test the BCD number is outlined below.

```
;read the 4 bits in from inputs for one BCD number
IF 1==1 THEN BTW W1 INP1 4
;create a copy of the word
IF 1==1 THEN W2 = W1
;convert the BCD number to Binary
IF 1==1 THEN BIN W2
;convert the now Binary number back to BCD
IF 1==1 THEN BCD W2
;check to see if they are the same BCD again
IF W2 != W1 THEN SET BCD_Fault_MEM, AsyncW = 258, MSG AsyncW
```

Set or Reset a Bit in a Word – BITSET / BITRST

When manipulating binary data it is often desirable to `SET` or `RST` only certain bits in the data. BITSET and BITRST can by used on Words or Word Type System Variables. Depending on the command used, the specified bit from 0-31 will be changed in the specified Word. Some

System Variables that are Integer Word values are actually 16 to 32 different Binary Bits that can be checked, for example SV_PC_CYCLONE_STATUS_1. Note that the Bit number range is different than the one used in WTB and BTW. The Bit number is the actual Bit that is checked. Because Bit numbering starts at zero, when you reference a Bit by number you are checking the “Bit number + 1” Bit.

```
IF 1==1 THEN BITSET SV_PLC_DEBOUNCE_1 5 ;Turn on Bit 6 in the first Debounce SV
IF 1==1 THEN BITRST W2 20               ;Turn off Bit 21 in Word 2
```

Check if a Bit is Set in a Word – BITTST

BITTST checks the specified bit in the specified Word or System Variable and SETs a Memory Bit if true and RSTs the Memory Bit if it is false. The range of bits that can be tested is 0-31. Note that the Bit number range is different than the one used in WTB and BTW. The Bit number is the actual Bit that is checked. Because Bit numbering starts at zero, when you reference a Bit by number you are checking the “Bit number + 1” Bit. In the example below Word number one, Bit 17 is checked and the state of it is copied to Memory Bit one.

```
IF 1==1 THEN BITTST W1 16 MEM1
```

Left / Right Shift Bits in a Word – LSHIFT/RSHIFT

Left and Right Shift are used on Words to move bits to the left or right. Right Shift is a Logical Shift rather than an Arithmetic Shift. This means that, as data is shifted, upper bit positions will always be filled with zero.

A pictorial example of shifting is in the following table with the same commands shown in code afterwards. The left shifting adds zeroes to the right-hand side to bump the number to the left first. The decimal value is 3735928559 originally and then after the left shift it becomes 4009754624. Finally, the right shift adds zeroes to the left-hand side to bump the number to the right and generate a decimal 61184. Notice that this acts as a mask and removes all except the second 8-bits.

Operation	**Binary 32-bit Word value**
Original Value	1101 1110 1010 1101 1011 1110 1110 1111
Left Shift 24 bits	1110 1111 0000 0000 0000 0000 0000 0000
Right Shift 16 bits	0000 0000 0000 0000 1110 1111 0000 0000

```
IF 1==1 THEN LSHIFT W1 24
IF 1==1 THEN RSHIFT W1 16
```

Trigonometric Functions – SIN, ASIN, COS, ACOS, TAN, ATAN2

Trig. Functions Sine, ArcSine, Cosine, ArcCosine, Tangent and ArcTangent2 are available in the PLC program. That being said, they should not be used except when it cannot in any way be done in G-Code because of the significant time requirement to calculate any of the Trig. Functions.

The values sent into the Trigonometric Functions must be in Radians, not Degrees. If you do SIN(30) the answer you will get is approximately -0.98803162, which is obviously not the SIN(30 degrees) = 0.5. To convert a degree value to Radians, multiply by PI and divide by 180 degrees. It is best to setup a constant in the InitialStage to avoid excessive computation and significantly reduce typographical errors. All of the valid Trig. Functions are used in the following program.

```
;/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
;Program: trig.src
;Purpose: Show that the Trig functions work correctly
;Date: 23-APR-2010
;/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

PI            IS 3.14159265358979323846264338327950
DEG_TO_RAD    IS (PI/180)
RAD_TO_DEG    IS (180/PI)

DoneOut       IS OUT1; SET when done calculating

rad_deg1_FW   IS FW1 ;sin angle in radians
rad_deg2_FW   IS FW2 ;cos angle in radians
rad_deg3_FW   IS FW3 ;tan angle in radians

calc_sin_FW   IS FW4 ;calculated sin
calc_cos_FW   IS FW5 ;calculated cos
calc_tan_FW   IS FW6 ;calculated tan

calc_asin_FW  IS FW7 ;calculated arcsin in degrees
calc_acos_FW  IS FW8 ;calculated arccos in degrees
calc_atan2_FW IS FW9 ;calculated atan2 in degrees

InitialStage  IS STG1
MainStage     IS STG2

;======================================================================
                     InitialStage
;======================================================================
IF 1==1 THEN rad_deg1_FW = 30 * DEG_TO_RAD ;sample degree values must -
```

```
IF 1==1 THEN rad_deg2_FW = 60 * DEG_TO_RAD ;get converted to radians -
IF 1==1 THEN rad_deg3_FW = 45 * DEG_TO_RAD ;for Trig Functions

IF 1==1 THEN calc_sin_FW = SIN (rad_deg1_FW) ;calc sin, cos, tan -
IF 1==1 THEN calc_cos_FW = COS (rad_deg2_FW) ;of 30 deg
IF 1==1 THEN calc_tan_FW = TAN (rad_deg3_FW) ;

;calc arcsin
IF 1==1 THEN calc_asin_FW  = ASIN  (calc_sin_FW)
;calc arccos
IF 1==1 THEN calc_acos_FW  = ACOS  (calc_cos_FW)
;calc atan2
IF 1==1 THEN calc_atan2_FW = ATAN2 (calc_tan_FW, calc_tan_FW)

IF 1==1 THEN calc_asin_FW  = calc_asin_FW  * RAD_TO_DEG ;
IF 1==1 THEN calc_acos_FW  = calc_acos_FW  * RAD_TO_DEG ;convert to deg.
IF 1==1 THEN calc_atan2_FW = calc_atan2_FW * RAD_TO_DEG ;

IF 1==1 THEN JMP MainStage

;=====================================================================
                    MainStage
;=====================================================================
IF JPI1 THEN (PD1)
IF PD1 THEN DUMP, RST MainStage
```

The following list shows what the results are from the above calculations. You can follow along on your PC with the calculator as long as you enter the first three values in Radians.

PLC Dump Start

FW1: 0.52359879	FW2: 1.04719758
FW3: 0.78539819	FW4: 0.50000000
FW5: 0.49999997	FW6: 1.00000012
FW7:30.00000000	FW8:60.00000000
FW9:45.00000000	

Square Root – SQRT

The square root of a number is a different number that, when multiplied by itself results in the given number. Numbers that have the square root taken of them should be stored into a Floating-point Word or Double-Floating-point Word because they will most likely be non-Integer numbers. SQRT results can be stored into a Word, but the non-Integer portion will be truncated. This operation should rarely be used due to the time intensive nature of the operation.

An example of using the SQRT function follows.

```
IntSqrt_W IS W1
Sqrt_FW   IS FW1
;take the sqare root of 2 and store to a Floating-point Word
IF 1==1 THEN Sqrt_FW = SQRT (2)   ;returns 1.41421354
;take the sqare root of 17 and store to a Word
IF 1==1 THEN IntSqrt_W = SQRT(17) ;returns 4.12310562, but 4 stored into the Word
```

Raise Number to a Power – POW

Raising a number to a power means multiplying the number by itself some number of times. The `POW` command takes two arguments, the first being the number to raise and the second being the number of times to multiply the first by itself. Both numbers can be floating-point values. If the second number is relatively small, it is usually faster to just multiply the number out, for example 2*2*2 instead of 2^3. The following example shows how to use the `POW` command.

```
pow_FW      IS FW1
pow2_FW     IS FW2

;raise 2 to the 3rd power
IF 1==1 THEN pow_FW = POW(2, 3)        ;returns 8.00000000
;raise 2 to the 3.5th power
IF 1==1 THEN pow2_FW = POW(2, 3.5)     ;returns 11.31370831
;raise 300 to the 1/8th power
IF 1==1 THEN pow3_FW = POW(300, 0.125) ;returns 2.04004693
```

Absolute value – ABS

Absolute value.

```
If 1==1 THEN W1 = -42,
             W2 = ABS(W1)   ; W2 = 42
```

Standard PLC Program Layout

A PLC program is laid out with all definitions at the top of the program until the first `IF` statement is used. From then on no more definitions are allowed and will generate an error at compile time. When the PLC program is executed it runs from top to bottom and executes every line except under certain circumstances, specifically Stages. Top to bottom execution can lead to logic errors where something is `SET` in one part of the program and then reset somewhere else. Wherever possible all conditions that could change an output should be gathered together.

The basic functionality described in this chapter is based on the basic DC3IOB PLC program.

The PLC program needs to have certain sections to achieve a minimum of real-world functionality as outlined in this chapter. The PLC program should always check all of the connections between the MPU, the PLC board including miniPLC boards and the drive board. In order for a Jog Panel or Keyboard Jogging to work they must be written into the PLC program.

Definitions are used to make reading the program easier and as such, the names used for the definitions should reflect the function and polarity when tripped, on or Green in PLC Diagnostics. This means that if a variable is Normally Closed it is named something like `XminusLimitOk`. The logic to check it becomes `IF !XMinusLimitOk THEN SET MEM9`. If the variable is Normally Open, the naming should be inverted. For a Normally Open Lube Fault input the logic to check it becomes `IF LubeFault THEN SET MEM90`.

At a minimum a `WatchDogStage`, `InitialStage` and `MainStage` should be used in every PLC program. The basic PLC program has many more stages for things like spindle control, Lube timers, Checking System Variables, etc. The `InitialStage` is used to setup any timers and default states. The `MainStage` is set in the `InitialStage` which then resets itself and is never `SET` again while CNC11 is running. This means that the `InitialStage` will only run once for each time the CNC software starts. Exiting CNC11 and restarting it causes the `InitialStage` to run again.

Defining Variables

Any variable is defined by having the label set to a valid data type using the keyword `IS`. Further examples can be seen for each data type under Data Types.

Initial-Condition Setup

The on/off state of various modes of operation such as Auto/Manual Coolant, Fast/Slow Jogging, etc. must be set by the PLC program because everything defaults to off. It is desirable to do this without having to push the buttons to set the states the first time because being off means something as much as being on. For example, if the Fast/Slow Jog mode is not actively set, it defaults to Fast Jog. The convention is to use a Memory Bit called `OnAtPowerUp`. It is `SET` in the `InitialStage` and `RST` at the end of the `MainStage`. This means that

all setup should be done inside the MainStage or before the MainStage executes. This influences the location of the MainStage in a program if setup is done in other Stages with this Memory Bit because if you RST it at the end of the MainStage and reference it after the MainStage, it will be false and not everything will be setup as desired. Typical things that should be setup with this variable are the Spindle Override percentage, Auto/Manual Spindle Mode, Spindle Direction, Auto/Manual Coolant Mode, Incremental/Continuous Jogging, Incremental Jog Distance Multiplier and Fast/Slow Jog Mode. An example of setting the Spindle Mode follows.

```
;--Set spindle to auto mode on startup
;--explanation
;IF (Jog-Panel-Key AND Not-In-Auto-Spindle-Mode) OR First-time-through-the-Program
;   Then turn on Auto-Spindle-Mode


;--actual code
IF (SpinAutoManPD && !SpinAutoModeLED) || OnAtPowerUp
  THEN SET SpinAutoModeLED
```

Internal PLC Fault and Software Running Checking

Just before the InitialStage there should be a Stage that checks for catastrophic PLC problems and also for CNC software to be running. Problems currently consist of dividing by zero, invalid op-code and index out of bounds. If any of these errors occurs the PLC executor immediately stops executing the program and starts over again at the beginning. This is why it is important to check these errors in the first Stage before any division or Index usage occurs.

PLC Fault Status

The procedure is to see if any of the three errors are set and if they are, read the related System Variables to Words and use the PLC messaging functionality to tell the user. Note that to generate the invalid op-code error the compiled PLC program must be changed at the hexadecimal level, so it is a very hard error to generate. No sample is given for this case as it involves manipulating the compiled program. A sample program follows that only checks for the three errors. To get either of the divide by zero or index out of bounds errors comment in either of the sections in the MainStage. Note that only the first error to occur will be generated due to the nature of these errors.

Software Ready

The SV_PC_SOFTWARE_READY Bit is checked to see if CNC software is running. This is mostly used if CNC software crashes to prevent motion, too changes, etc. SV_STOP is set which will cause E-Stop be activated. When the software starts again, E-Stop must be cycled and then motion can be commanded again. SV_STOP is SET and the InitialStage is run whenever the software starts up again.

Checking PLC Fault and Software Ready

```
;===============================================================================
                          WatchDogStage
;===============================================================================

; Handle PLC executor faults. The only way to reset a PLC executor fault
; is to reboot the MPU11.
IF SV_PLC_FAULT_STATUS != 0
  THEN PLC_Fault_W    = SV_PLC_FAULT_STATUS,
       PLCFaultAddr_W = SV_PLC_FAULT_ADDRESS,
       ErrorCode_W    = PLC_EXECUTOR_FLT_MSG, MSG ErrorCode_W,
       SET PLCExecutorFault_M, RST SetErrorStage, SET SV_STOP

; Handle software exit.
IF !SV_PC_SOFTWARE_READY && (SV_PLC_FAULT_STATUS == 0)
  THEN SET SoftwareNotReady_M,
       SET SV_STOP,
       ErrorCode_W = SOFTWARE_EXIT_MSG

if SV_PC_SOFTWARE_READY && (SV_PLC_FAULT_STATUS == 0) THEN (SoftwareReadyPD)
IF SoftwareReadyPD && !SoftwareNotReady_M || !True THEN SET InitialStage

IF SoftwareReadyPD && SoftwareNotReady_M THEN RST SoftwareNotReady_M
;===============================================================================
                           MainStage
;===============================================================================
;This is just sample code to generate the errors, they should not actually be done in
a ;real PLC program. To cause either of the errors uncomment the lines starting with IF.
;divide by zero
;If 1==1 THEN Word2 = 10/0

;index out of bounds
;IF 1==1 THEN Word3 = 2500, Word4 = 2500
;IF 1==1 THEN Word5 = Word3 + Word4
;IF 1==1 THEN SET OUT[Word5]
```

Jog Panel and Keyboard Jogging

This is a significant part of the PLC program and as such is not transcribed from the basic program.

Nearly all Jog Panel functions that can be caused by key presses require Keyboard Jogging to be SET in Parameter 170 bit 0 and RST in Parameter 148 bit 1. The exceptions are Escape and Rapid Override.

There are some keys that require the Keyboard Jog Panel to be on screen as well as have Keyboard Jogging enabled. These functions are primarily limited to Jogging.

Axis Enable

The axes are checked for motion related fault conditions including a drive fault, stall error and fiber connections.

Fiber/Wire Connection Checking

There are two kinds of communication used on MPU11 systems. The DriveBus is used for communication related to motor control and can be either fibers 4 and 5 or the Drive Communication wiring for expanding the number axes that can be controlled. The PLCBus is on fibers 1 and 3 and is used for controlling I/O. If the Buses are being used, the connections must be checked to ensure communication problems are not clouding troubleshooting.

Checking the connections only needs to be done a few times a second to be effective. This means having a stage that is set by a Timer to read the information. Both of the following sections use the same logic to check for errors, but the first checks both connection methods and the second only checks the PLCBus. The third section needs to be added if an expansion board is added to the PLC.

Drive and PLCBus Checking

Presently the Optic4 and DC3IOB have DriveBus Fiber connections. The Optic4, DC3IOB and ALLIN1DC have Drive Communication In and/or Out Wire connections. The ALLIN1DC has an MPU11 on-board which means there are no visible Fiber connections, but the Drive and PLCBuses are still used and should be checked. If a system using 3rd party drives is used with more than four axes, the Optic4 must be used in conjunction with the GPIO4D and the DriveBus Fiber Checking must be done as well.

```
;===============================================================================
  CheckCycloneStatusStage
;===============================================================================
; Due to amount of time it takes to retrieve data from the cyclone, this stage
; is only called few times per second to help reduce scan time of the main PLC
; program.
```

```
; The logic below is the equivalent to the following:
; IF true THEN BITTST SV_PC_CYCLONE_STATUS_2 0 Axis1FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 1 Axis2FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 2 Axis3FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 3 Axis4FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 4 Axis5FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 5 Axis6FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 6 Axis7FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 7 Axis8FiberOk_M
IF true THEN WTB SV_PC_CYCLONE_STATUS_2 Axis1FiberOk_M


; Generate some messages for fiber or wire to MPU11 having issues
IF SV_AXIS_VALID_1 && !SV_DRIVE_ONLINE_1 THEN ErrorCode_W = AXIS1_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_2 && !SV_DRIVE_ONLINE_2 THEN ErrorCode_W = AXIS2_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_3 && !SV_DRIVE_ONLINE_3 THEN ErrorCode_W = AXIS3_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_4 && !SV_DRIVE_ONLINE_4 THEN ErrorCode_W = AXIS4_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_5 && !SV_DRIVE_ONLINE_5 THEN ErrorCode_W = AXIS5_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_6 && !SV_DRIVE_ONLINE_6 THEN ErrorCode_W = AXIS6_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_7 && !SV_DRIVE_ONLINE_7 THEN ErrorCode_W = AXIS7_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_8 && !SV_DRIVE_ONLINE_8 THEN ErrorCode_W = AXIS8_INFLT, SET
DriveComFltIn_M


; Generate some messages for fiber or wire to drive having issues
IF SV_AXIS_VALID_1 && SV_DRIVE_ONLINE_1 && SV_MASTER_ENABLE && !Axis1FiberOk_M
  THEN ErrorCode_W = AXIS1_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_2 && SV_DRIVE_ONLINE_2 && SV_MASTER_ENABLE && !Axis2FiberOk_M
  THEN ErrorCode_W = AXIS2_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_3 && SV_DRIVE_ONLINE_3 && SV_MASTER_ENABLE && !Axis3FiberOk_M
  THEN ErrorCode_W = AXIS3_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_4 && SV_DRIVE_ONLINE_4 && SV_MASTER_ENABLE && !Axis4FiberOk_M
  THEN ErrorCode_W = AXIS4_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_5 && SV_DRIVE_ONLINE_5 && SV_MASTER_ENABLE && !Axis5FiberOk_M
```

```
  THEN ErrorCode_W = AXIS5_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_6 && SV_DRIVE_ONLINE_6 && SV_MASTER_ENABLE && !Axis6FiberOk_M
  THEN ErrorCode_W = AXIS6_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_7 && SV_DRIVE_ONLINE_7 && SV_MASTER_ENABLE && !Axis7FiberOk_M
  THEN ErrorCode_W = AXIS7_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_8 && SV_DRIVE_ONLINE_8 && SV_MASTER_ENABLE && !Axis8FiberOk_M
  THEN ErrorCode_W = AXIS8_OUTFLT, set DriveComFltOut_M

If !EstopOk THEN rst DriveComFltIn_M, rst DriveComFltOut_M
If DriveComFltOut_M || DriveComFltIn_M THEN set AxisFault_M

;check PLC status bit
IF TRUE THEN BitTst SV_PC_CYCLONE_STATUS_1 21 PLCBusExtDevEn_M

;check input fiber
IF !SV_PLC_BUS_ONLINE THEN ErrorCode_W = PLC_INFLT,
                           rst PLCBus_Oe_M, set PLCFault_M

;check output fiber
IF SV_PLC_BUS_ONLINE && PLCBus_Oe_M && !PLCBusExtDevEn_M
  THEN ErrorCode_W = PLC_OUTFLT, SET PLCFault_M

;clear PLC errors
IF PLCFault_M && SV_PLC_BUS_ONLINE && PLCBusExtDevEn_M && !EstopOk
  THEN RST PLCFault_M, ErrorCode_W = PLC_FLT_CLR, SET PLCBus_Oe_M

IF True THEN RST CheckCycloneStatusStage
```

PLCBus Checking only

The GPIO4D communicates to the MPU11 only over the PLCBus. If a system only has a GPIO4D then only the PLCBus should be checked. This section details the PLC Fiber checking only. The errors are checked for and set in the AxesEnableStage.

```
;===============================================================================
  CheckCycloneStatusStage
;===============================================================================
; Due to amount of time it takes to retrieve data from the cyclone, this stage
```

```
; is only called few times per second to help reduce scan time of the main PLC
; program.

;check PLC status bit
IF True THEN BITTST SV_PC_CYCLONE_STATUS_1 21 PLCBusExtDevEn_M

;check input fiber
IF !SV_PLC_BUS_ONLINE THEN ErrorCode_W = PLC_INFLT,
                           RST PLCBus_Oe_M, SET PLCFault_M

;check output fiber
IF SV_PLC_BUS_ONLINE && PLCBus_Oe_M && !PLCBusExtDevEn_M
  THEN ErrorCode_W = PLC_OUTFLT, SET PLCFault_M

;clear PLC errors
IF PLCFault_M && SV_PLC_BUS_ONLINE && PLCBusExtDevEn_M && !EstopOk
  THEN RST PLCFault_M, ErrorCode_W = PLC_FLT_CLR, SET PLCBus_Oe_M

IF True THEN RST CheckCycloneStatusStage
```

MiniPLCBus Checking

This section should be added to any PLC program that uses the PLCADD1616 or ADD4AD4DA expansion boards. Specifically the checking should be added to the `CheckCycloneStatusStage`. Only the MiniPLCBus Online bits are checked in this section, but they should be added to the original Stage. Only the Expansion headers that are used should be checked. There is no equivalent to the `SV_Axis_Valid_1 - _7` System Variables. Looking in the mpu_info.txt file will show what expansion boards are plugged in to the headers.

```
;add to Constant Defines
MINI_PLC_1_FLT_MSG IS (1 + 256*60); 15361
;add to CheckCycloneStatusStage
;check the first Expansion board
IF 1==1 THEN BITTST SV_PC_MINI_PLC_ONLINE 0 ADD1616ok1_M
IF !ADD1616ok1_M && PLCBus_Oe_M THEN
  ErrorCode_W = MINI_PLC_1_FLT_MSG, SET SetErrorStage, SET PLCFault_M
```

LubeTimers

There are two options for setting up automatic lubing in the basic PLC program. The first type

is used for Lube pumps with no timer internally. This means that the PLC program must turn the output on and off to power the Lube pump correctly. The second type is for Lube pumps with internal timers. If Parameter 179 is set to a zero, then the pump timers will be used, otherwise the PLC program will keep track of the time that a job is running to determine when the next lube cycle should start.

Lube Pump Internal Timer

```
;===============================================================================
                        LubeUsePumpTimersStage
;===============================================================================

; METHOD 1 (SS == 0) For lube pumps with internal timers.
;
; When using this method, P179 should be set such that MMM is a
; value that is greater than the cycle time set on the internal timers and
; SS should be set to zero. How much greater MMM needs to be depends on the
; accuracy of the lube pump timers, but it is better to be on the long side
; to ensure proper operation.
;
; Example 1. The internal lube cycle interval is set to 60 minutes.
;            Set P179 = 7500. In this example, as long as the accuracy
;            of the lube timer interval causes the lube to turn on
;            within 75 minutes, it will work. Note that the amount of time
;            that lube is output is usually set with another timer control
;            on the lube pump and it does not factor into the setting of P179.
;
; It should be noted that lube pumps with internal timers may differ on how
; they operate.
;
; (a) For pumps that lube immediately when power is applied and then start timing
; until the next cycle, it is possible to run out of lube quickly on short job
; runs if, after the program has been run, lube power is removed.
;
; (b) For pumps that do not lube until it has been turned on for the interval time,
; it is possible that lube never gets applied if, after the short program has been run,
; lube power is removed.
;
```

```
; A short program or job run is defined as a job that finishes before
; the interval setting (60 minutes in the above example).
;
; For the above mentioned reasons, we want the power to be applied for at least
; the amount of time set by the inteval timer, noting that if the user decides
; to engage the E-stop to remove power after short jobs, then they risk the
; above mentioned problems accoding to the type of pump.
;
; On the start of SV_PROGRAM_RUNNING, the lube pump turns on.
; The lube pump is turned off when a program has NOT been
; running continuously for MMM minutes or E-stop is engaged.
; The reason the lube pump is turned off after a program has NOT been
; running for MMM minutes is to prevent lubing when the user leaves for the
; weekend, leaving the machine on and E-stop disengaged.

IF (SV_PROGRAM_RUNNING || SV_MDI_MODE) THEN SET Lube, RST LubeM_T
IF !(SV_PROGRAM_RUNNING || SV_MDI_MODE) THEN LubeM_T = LubeM_W, SET LubeM_T
IF LubeM_T || !EStopOk THEN RST Lube
```

Lube Pump External Timer

```
;===============================================================================
                        LubeUsePLCTimersStage
;===============================================================================
;
; METHOD 2 (SS != 0) For lube pumps that do not have internal timers.
;
;  When using this method P179 should be set so the lube turns on
;  every MMM minutes for SS seconds.
;
;  Example 1.
;    To set the lube pump power to come on for 5 seconds
;    every 10 minutes, set P179 =  1005.
;                                  MMMSS
;  Example 2.
;    To set the lube pump power to come on for 30 seconds
;    every 2 hours, set P179 = 12030
```

```
;                                MMMSS
;
; This method will accumulate time while a program is running until
; it reaches MMM minutes, at which time it will apply power
; for SS seconds (unless E-stop is engaged) and then start over. It is
; possible with frequent use of E-stop that a lube cycle is cut short.
;


IF (SV_PROGRAM_RUNNING  || SV_MDI_MODE)  THEN LubeM_T = LubeM_W, SET LubeM_T
IF !(SV_PROGRAM_RUNNING || SV_MDI_MODE) THEN (StopRunningPD)
IF StopRunningPD THEN LubeAccumTime_W = LubeAccumTime_W + LubeM_T, RST LubeM_T
IF LubeM_T || (LubeAccumTime_W + LubeM_T > LubeM_W)
  THEN SET Lube, LubeS_T = LubeS_W, SET LubeS_T, RST LubeM_T, LubeAccumTime_W = 0
IF LubeS_T || !EStopOk THEN RST Lube, RST LubeS_T
```

Feedrate Override

Feedrate Override allows changing the master feedrate or commanded velocity if it is enabled. The basic feedrate override starts by reading the Feedrate potentiometer then scaling it to 0% to 200% from 0 to 256. The program then determines whether the Feedrate knob or the keyboard feedrate keys should be applied to CNC software. The maximum value is limited by parameter 39. Parameter 78 allows the feedrate value to be adjusted down if the spindle speed does not keep up with the command so CNC software gets to see if it wants to modify the Feedrate Override. Finally the PLC program has one last chance to change the new value coming back from the CNC software, though typically it should not.

There is now a feature that allows using keyboard and Knob override at the same time. By default the Knob is used, but if the Feedrate Override keyboard keys are used, then the Feedrate override displayed is based on the Keyboard value and not the Knob value. When the Knob is turned again more than 3%, however that value is displayed.

Spindle Functionality

Spindle DAC Output

Precise spindle speed is controlled through an inverter by sending an analog signal from the PLC to tell the inverter what speed to go at. The inverter must be calibrated to the analog signal with a tachometer or spindle encoder. This section only details setting up the PLC side of inverter control. Spindle Digital to Analog output or DAC is set by the PLC Program based on the commanded Spindle Speed from the 'S' command in G-Code, the Spindle Override value and the Spindle Gear Ratio. The DAC Spindle Speed command won't be seen by MPU11 until the DoSpindleStart (`SV_PLC_FUNCTION_37`) goes from off to on or DoSpindleStop

(SV_PLC_FUNCTION_38) goes from on to off. This means that the Spindle Start Key cannot be held down at boot and have the spindle start as the software starts. This is a good thing. Both or either one of the Start or Stop can be used to control turning the Spindle on and off. Keep in mind that every time the Start or Stop changes state in the program, the DAC output will be updated, either turning on or off. This means that in most cases you should choose one of the two variables to use, rather than both, to avoid confusion.

The Spindle Analog Output bits are mapped to different places on different PLCs. The following table outlines the MPU11 board and the outputs for the Spindle Analog.

PLC	Spindle DAC Outputs
ALLIN1DC	241-252
DC3IOB	17-28
GPIO4D	305-316

Spindle Gear Ranges

The basic PLC programs have two ranges defined. Low Range and High Range. The High Range multiplier is taken from Parameter 33 and Low Range is taken from Parameter 65.

The example code is for a DC3IOB PLC program. The Inputs and Outputs for each of the standard PLC programs is detailed in Appendix E.

MPG Operation

The MPG or Manual Pulse Generator can be used to supplement Jog buttons. This section is used to enable the MPG mode control in CNC software for the standard Centroid CNC MPG. This includes provisions for up to 4 axis in the basic PLC program with three step amounts. Note that when the MPG is active, with the way this is programmed, jogging will not work. It can be written into the PLC program that when a Jog Key is pushed to turn off MPG mode temporarily. Windup mode is used to make sure that the motor moves all of the commanded steps that the MPG sent. This is not desirable in x100 mode because turning the MPG fast enough will result in motion well after the MPG has stopped turning.

```
;=============================================================================
                            MPGStage
;=============================================================================
;                          MPG Functions
;       Turn on/off Jog Panel MPG LED & on the MPG
IF MPGKey THEN (MpgPD)
IF MpgPD && MPGLED THEN SET MPGManOffFlag_M
IF !SV_MPG_1_ENABLED || (MpgPD && !MPGLED) THEN RST MPGManOffFlag_M


IF (MpgPD && !MPGLED) || (SV_MPG_1_ENABLED && !MPGManOffFlag_M) &&
```

```
   !SV_PROGRAM_RUNNING THEN SET MPG_LED_OUT, SET MPGLED

IF (!SV_MPG_1_ENABLED || (MpgPD && MPGLED))
   || SV_PROGRAM_RUNNING THEN RST MPG_LED_OUT, RST MPGLED

;x1, x10, x100 functions
;--X1
IF x1JogKey THEN (x1JogPD)
IF x1JogPD || OnAtPowerUp_M || X1_M || (MPG_Inc_X_1 && MPGLED)
  THEN SET x1JogLED, RST x10JogLED, RST x100JogLED

;--X10
IF x10JogKey THEN (x10JogPD)
IF x10JogPD || X10_M || (MPG_Inc_X_10 && MPGLED)
  THEN RST x1JogLED, SET x10JogLED, RST x100JogLED

;--X100
IF x100JogKey THEN (x100JogPD)
IF x100JogPD || X100_M || (MPG_Inc_X_100 && MPGLED)
  THEN RST x1JogLED, RST x10JogLED, SET x100JogLED

IF !KbIncreaseJogInc_M && !KbDecreaseJogInc_M THEN RST X1_M, RST X10_M,
  RST X100_M

;--MPG 1 Enable
IF MPG_AXIS_1 || MPG_AXIS_2 || MPG_AXIS_3 || MPG_AXIS_4 ||
  MPG_AXIS_5 || MPG_AXIS_6 || MPG_AXIS_7 || MPG_AXIS_8
  THEN (SV_MPG_1_ENABLED)

;                     Select axis to move
IF MPG_AXIS_1 THEN SV_MPG_1_AXIS_SELECT = 1
IF MPG_AXIS_2 THEN SV_MPG_1_AXIS_SELECT = 2
IF MPG_AXIS_3 THEN SV_MPG_1_AXIS_SELECT = 3
IF MPG_AXIS_4 THEN SV_MPG_1_AXIS_SELECT = 4
IF MPG_AXIS_5 THEN SV_MPG_1_AXIS_SELECT = 5
```

```
;                   Select MPG 1 Multiplier
IF (MPG_Inc_X_100) THEN SV_MPG_1_MULTIPLIER = 100
IF (MPG_Inc_X_10) THEN  SV_MPG_1_MULTIPLIER = 10
IF (MPG_Inc_X_1) THEN   SV_MPG_1_MULTIPLIER = 1

;            Disable "Windup" mode IF x100 selected
IF (!MPG_Inc_X_100) THEN (SV_MPG_1_WINDUP_MODE)
```

Coolant Control

Coolant refers to the Flood and Mist control. Automatic coolant does not allow turning on the outputs unless the M7/M8 macros are used. Pushing the key to turn off Automatic coolant will then allow manual only control where a button must be pushed to turn on/off the coolant.

```
;--Coolant Functions

;--Toggle auto coolant mode
IF CoolAutoManKey || KbTogCoolAutoMan_M THEN (CoolantAutoManualPD)

IF (!CoolAutoManLED && CoolantAutoManualPD) || OnAtPowerUp_M
  THEN SET CoolAutoManLED

IF (CoolAutoManLED && CoolantAutoManualPD)
  THEN RST CoolAutoManLED

;--Report coolant mode to CNC11
IF CoolAutoManLED THEN (SelectCoolAutoMan)

;--Display coolant mode message
;changing to auto coolant mode ;9050 Auto Coolant Selected 2 + 50*256
IF (!CoolAutoManLED && CoolantAutoManualPD)
  THEN AsyncMsg_W = 12802, MSG AsyncMsg_W

;changing to manual coolant mode ;9051 Manual Coolant Selected 2 + 51*256
IF (CoolAutoManLED && CoolantAutoManualPD)
  THEN  AsyncMsg_W = 13058, MSG AsyncMsg_W

;--Flood coolant on/off
```

```
IF ((CoolFloodKey || KbFloodOnOff_M) && !CoolAutoManLED) ||
  (M8 && CoolAutoManLED) || (DoCycleStart && M8 && CoolAutoManLED)
  THEN (CoolantFloodPD)

IF CoolantFloodPD && !CoolFloodLED THEN SET CoolFloodLED, SET Flood

IF SV_STOP || (CoolantFloodPD && CoolFloodLED) || (!M8 && CoolAutoManLED) ||
    (M8 && !CoolAutoManLED) || DoToolCheck THEN RST Flood, RST CoolFloodLED

IF CoolFloodLED THEN (SelectCoolantFlood)

;--Mist coolant on/off
IF ((CoolMistKey || KbMistOnOff_M)&& !CoolAutoManLED) || (M7 && CoolAutoManLED)
    || (DoCycleStart && M7 && CoolAutoManLED) THEN (CoolantMistPD)

IF (CoolantMistPD && !CoolMistLED) THEN SET Mist, SET CoolMistLED

IF SV_STOP || (CoolantMistPD && CoolMistLED) || (!M7 && CoolAutoManLED) ||
    (M7 && !CoolAutoManLED) || DoToolCheck THEN RST Mist, RST CoolMistLED

IF CoolMistLED THEN (SelectCoolantMist)
```

Probe Protection

There is some minimal protection built into the default PLC program to try and protect against crashing a probe. If the Mechanical probe trips while jogging, a probe fault is triggered. Jogging is not allowed in the direction that was being commanded when the probe tripped until the probe is cleared.

```
;-------------------------------------------------------------
;           Probe protection while jogging
;-------------------------------------------------------------
IF MechanicalProbe && !JogProbeFault_M && (DoAx1PlusJog || DoAx1MinusJog ||
   DoAx2PlusJog || DoAx2MinusJog || DoAx3PlusJog || DoAx3MinusJog ||
   DoAx4PlusJog || DoAx4MinusJog || DoAx5PlusJog || DoAx5MinusJog)
  THEN (JogProbeFaultPD)

IF MechanicalProbe && !JogProbeFault_M && FastSlowLED THEN SET LastProbeMode_M
```

```
IF MechanicalProbe && !JogProbeFault_M && !FastSlowLED THEN RST LastProbeMode_M

IF JogProbeFaultPD && !JogProbeFault_M THEN SET JogProbeFault_M, SET DoCycleCancel

IF JogProbeFault_M THEN ErrorCode_W = (PROBE_JOG_FAULT_MSG + 1),
                        SET FastSlowLED

IF !MechanicalProbe && JogProbeFault_M && !LastProbeMode_M THEN RST FastSlowLED

IF !MechanicalProbe THEN RST JogProbeFault_M,
                         RST Ax1PlusJogDisabled_M,
                         RST Ax1MinusJogDisabled_M,
                         RST Ax2PlusJogDisabled_M,
                         RST Ax2MinusJogDisabled_M
```

PLC Optional Sections

This chapter covers optional sections of the PLC will go into detail about setting up the Debounce registers and inverting Inputs. There are several new features in CNC PLC programs, namely Input debounce and Input inversion. There is a standard value of 1.5 ms of debounce applied to all applicable PLC inputs and 18 ms debounce for the jog panel. No inputs are inverted by default. The PLC program can have debounce applied in excess of the standard value for a particularly noisy input.

Debounce or Invert Inputs

When a physical Input goes from on to off or vice-versa the PLC may see multiple transitions as the input turns on and off very quickly. Debounce is used to make sure that the Input is really on or really off. Once the Input is seen as on or off for the specified time, it is considered to be in that state and reported to the PLC program.

The PLCbus Inputs are Debounced at 1.5 ms by default, which is typically a decent value. Sometimes a switch is extra noisy and require more time to settle into a state. When that happens the debounce System Variables need to be adjusted. Debounce can be set for the first 240 PLCbus physical Inputs, 112 Jog Panel Inputs and 32 MPU11 Local I/O. They are all done with different System Variables, but the procedure is the same. The new Debounce values must be set every time the MPU11 boots, so a good place to do this is in the `InitialStage`.

The Debounce procedure scans at a certain rate, which determines the speed that an Input can be Debounced. The following chart lays out the scan rate of each of the Input types with Debounce. Keep in mind that the number of consecutive states of an Input required is determined by dividing the desired Debounce time by the scan rate. For example to get 1.5 ms of Debounce on a standard Input requires 0.0015/0.0000625 = 24 scans. For reference 1 ms (millisecond) = 0.001 s and 1 µs (microsecond) = 0.000001 s.

Note that there are two different updates rates for the PLC Inputs depending on where the Input is located. The Inputs directly on-board the DC3IOB and GPIO4D, for example are updated at 16kHz, whereas the Inputs on the PLCADD1616 are updated at 4kHz. However, the scan time for debounce (62.5µs) is the same for all PLC and and PLC expansion devices. This means that even though the expansion devices update 4000 times per second they are checked at 16000 times per second. Therefore 16000/4000 = 4 times the number of scans must be done on an expansion board. Instead of doing 24 scans for 1.5 ms of debounce 96 scans must be done.

The minimum Debounce Time is 0 which equates to the base Scan Rate and the maximum Debounce Time of 32768 equates to 2.048 s for PLC and MPU11 Local Inputs and 24.576 s.

Always use Debounce Time 2 or greater when making a custom time because Debounce Time 1 is used by most Inputs by default and Debounce Time 0 is always 0 and cannot be modified.

System Variable Name	Scan Rate (µs)	Applies to
SV_PLC_DEBOUNCE_1 to _64	62.5	PLC Inputs
SV_JOG_LINK_DEBOUNCE_1 to _32	750	Jog Panel Inputs
SV_LOCAL_DEBOUNCE_1 to _13	62.5	MPU11 On-board Inputs

The default values for all of the Debounce Inputs is listed below.

	Default Debounce Setting Value	Default Debounce Time Value	Default Debounce Time (ms)
PLC Inputs	1	24	1.5
Jog Panel Inputs	1	24	18
MPU11 Local I/O	1*	24	1.5

*Input 770 for the DSP Probe Defaults to 0. Do not change this value.

Each group of System Variables is broken up into two sections. The majority of the System Variables are used for Debounce Setting to apply to the Inputs and the last few are for setting up the Debounce Time. This means that there is not a separately named SV for the time and selecting what Input is used, just a different number. The following table illustrates which System Variables are for Inputs and which are for setting up the time.

System Variable Range	Function
SV_PLC_DEBOUNCE_1 to _60	Select Debounce time to use for each Input
SV_PLC_DEBOUNCE_61 to _64	Choose number of scans to Debounce
SV_JOG_LINK_DEBOUNCE_1 to _28	Select Debounce time to use for each Input
SV_JOG_LINK_DEBOUNCE_29 to _32	Choose number of scans to Debounce
SV_LOCAL_DEBOUNCE_1 to _9	Select Debounce time to use for each Input
SV_LOCAL_DEBOUNCE_10 to _13	Choose number of scans to Debounce

For each of the three types of Debounced Input there are seven different times that can be used in total. This means that any Input can have one of seven different times applied to it for Debounce.

Each of the System Variables to setup Inputs are broken up into four 8 bit sections devoted to one Input each. The layout for SV_PLC_DEBOUNCE_1 is illustrated in the following table, but all of the System Variables for setting up Inputs are divided the same way.

SV_PLC_DEBOUNCE_1 Bits	Function
0-7	Input 1 Setting
8-15	Input 2 Setting
16-24	Input 3 Setting
25-31	Input 4 Setting

Each of the 8-bit Input Setting Bytes is laid out according to the following table.

Input Setting Bit Number	Function
0	Debounce Time Select 1-7 in binary 1 = Time 1 ... 111 = Time 7
1	
2	
3	reserved
4	reserved
5	reserved
6	Invert Input
7	Force Input On

In order to figure out which Debounce System Variable to use, divide the Input number by four and add one. PLC Input 211 is on 211 / 4 = 52.75 plus one gives 53.75 for SV_PLC_DEBOUNCE_53. The fractional component tells which of the four input bytes in the System Variable should be set to get the correct Input. In the example above 0.75 is ¾ which points to Byte 3 out of 4 or bits 16-24.

Setting up the Debounce Time System Variables is more strait forward because there are only four System Variables that can be setup for each Input type. Each of the System Variables is split into two 16-bit values with the lowest one in each category being unused. That is why only Debounce times from 1 to 7 are allowed to be customized. Debounce Time 0 is always set to 0 and cannot be modified. The following table illustrates the seven available Debounce Times by System Variable.

Debounce Time Sys. Vars.	Debounce Time
SV_PLC_DEBOUNCE_61	0 – 15 always set to 0 16 – 31 Debounce Time 1
SV_PLC_DEBOUNCE_62	0 – 15 Debounce Time 2 16 – 31 Debounce Time 3
SV_PLC_DEBOUNCE_63	0 – 15 Debounce Time 4 16 – 31 Debounce Time 5
SV_PLC_DEBOUNCE_64	0 – 15 Debounce Time 6 16 – 31 Debounce Time 7

There are several ways to setup the Debounce values. One option is to calculate out the bit values in decimal for all 32-bits and add them all up. Another option is to create each 8-bits for the Debounce Time or Debounce Setting in a temp Word and Left Shift the value up to the

correct place in the Word then add it to the Debounce System Variable. As an example the following program reads sets up INP6 with a debounce time of 13 ms and inverts the input. The values are recorded here in binary to show the bits changing. Note that these bits are written left to right and Msb to Lsb whereas WTB writes bits Lsb to Msb left to right.

Example Input Debounce Setup Program

```
;Orignal INP6 Debounce Setting Word Value
;0000 0001 0000 0001 0000 0001 0000 0001
;Desired Debounce Setting bits for INP6
;0000 0000 0000 0000 0000 0101 0000 0000
;modified Debounce Setting Word with new bits added in
;0000 0001 0000 0001 0000 0101 0000 0001
;Original Debounce Time Word Value
;0000 0000 0001 1000 0000 0000 0001 1000
;Debounce Time after removing the 16 bits that will be replaced
;0000 0000 0000 0000 0000 0000 0001 1000
;Temp Debounce Time Word with new value shifted into place
;0000 0000 1101 0000 0000 0000 0000 0000
;Final Debounce Time Word with new time value
;0000 0000 1101 0000 0000 0000 0001 1000

;/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
;Program: debounce.src
;Purpose: Example Debounce Setup for INP6
;         Set Time 5 for 13 ms., Invert Input
;Date: 20-APR-2010
;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

;--Variable Defines
Debounce_2_Bits_M     IS MEM1  ;display the bits from the Word
Debounce_63_Bits_M    IS MEM35 ;display the new bits for the Word

Bit0_M                IS MEM73 ; Deb Time Select Bit 0
Bit1_M                IS MEM74 ; Deb Time Select Bit 1
Bit2_M                IS MEM75 ; Deb Time Select Bit 2
Bit3_M                IS MEM76 ; reserved
```

```
Bit4_M                IS MEM77 ; reserved
Bit5_M                IS MEM78 ; reserved
Bit6_M                IS MEM79 ; Invert Input
Bit7_M                IS MEM80 ; Force Input On

Original_Deb_W        IS W1    ;Value read from Deb SV
Temp_Deb_W            IS W2    ;Calculated Word value
Final_Deb_W           IS W3    ;final combined Word value
Orig_Deb_Time_W       IS W4    ;Value read from Deb SV
Shift_Deb_Time_W      IS W5    ;value with high 16 bits removed
Temp_Deb_Time_W       IS W6    ;Calculated Word value
Final_Deb_Time_W      IS W7    ;final combined Word value

InitialStage          IS STG1
MainStage             IS STG2

;reading INP6 means reading from SV_PLC_DEBOUNCE_?
; ? = 6/4 + 1 = 2.5
;choosing which Byte to set requires looking at the remainder
;the remainder, 0.5 = 2/4 or byte two of four must be set

;use Time 5 to setup the Debounce Time
;5 decimal = 101 binary
;looking at the table gives SV_PLC_DEBOUNCE_63 high 16 bits should be used

;============
InitialStage
;============
IF 1==1 THEN Original_Deb_W = SV_PLC_DEBOUNCE_2   ;read the Settings SV
IF 1==1 THEN Orig_Deb_Time_W = SV_PLC_DEBOUNCE_63 ;read the Time SV
IF 1==1 THEN Final_Deb_W = Original_Deb_W

;zero out the bits that are going to be replaced
IF 1==1 THEN BITRST Final_Deb_W 8
IF 1==1 THEN BITRST Final_Deb_W 9
IF 1==1 THEN BITRST Final_Deb_W 10
```

```
IF 1==1 THEN BITRST Final_Deb_W 11
IF 1==1 THEN BITRST Final_Deb_W 12
IF 1==1 THEN BITRST Final_Deb_W 13
IF 1==1 THEN BITRST Final_Deb_W 14
IF 1==1 THEN BITRST Final_Deb_W 15

;Setup the 8 bits for the Setting Byte
IF 1==1 THEN SET Bit0_M ;
            ,RST Bit1_M ;binary 5 for Time 5
            ,SET Bit2_M ;
            ,RST Bit3_M ;unused
            ,RST Bit4_M ;unused
            ,RST Bit5_M ;unused
            ,RST Bit6_M ;do not Invert the Input
            ,RST Bit7_M ;do not Force On

;move the 8 bits to a Word
IF 1==1 THEN BTW Temp_Deb_W Bit0_M 8
;shift the values to the 2nd byte as calculated above
IF 1==1 THEN LSHIFT Temp_Deb_W 8

;copy the new 8 bits to the final word
IF 1==1 THEN Final_Deb_W = Final_Deb_W + Temp_Deb_W

;calculate the Debounce scans required for 13 ms
;onboard PLC INP so 0.0130/0.0000625 = 208 scans
IF 1==1 THEN Temp_Deb_Time_W = 208

;shift the bits up to the high 16 bits for the word
IF 1==1 THEN LSHIFT Temp_Deb_Time_W 16

;shift off the high bits from the original Debounce
;registers
IF 1==1 THEN Shift_Deb_Time_W = Orig_Deb_Time_W
IF 1==1 THEN LSHIFT Shift_Deb_Time_W 16
IF 1==1 THEN RSHIFT Shift_Deb_Time_W 16
```

```
;combine the low 16 bits and high 16 bits
IF 1==1 THEN Final_Deb_Time_W = Temp_Deb_Time_W + Shift_Deb_Time_W


;save the values back to the SV
IF 1==1 THEN SV_PLC_DEBOUNCE_2 = Final_Deb_W
IF 1==1 THEN SV_PLC_DEBOUNCE_63 = Final_Deb_Time_W


IF 1==1 THEN RST InitialStage, SET MainStage


;============
MainStage
;============
;write out bits of final Word settings for verification
IF 1==1 THEN WTB Final_Deb_W Debounce_2_Bits_M 32
           , WTB Final_Deb_Time_W Debounce_63_Bits_M 32
```

Setting Inputs High or Low for Testing

The Debounce registers that store the settings for the Inputs can also be used to invert, at the hardware level, the state of the input. This means that a Normally Closed Input can be turned into a Normally Open Input.

Note that on the DC3IOB Inputs 1-6 and 11 are acted on at the hardware level before the Force and Invert bits are looked at. This means that Normally Open Limits and E-Stop still cannot be used.

There is also the ability to Force an Input to be `SET`. This should really never be used outside of debugging on the bench due to the possibility of masking an input from indicating an error has occurred. Inputs can be Forced off or `RST` by setting both the Invert and Force on bits.

Looking at the Table above that explains the layout of the Setting Bytes shows that Bit 7 and 6 are used for Forcing the Input and Inverting the Input respectively. Time 1 should always be set if no custom Debounce time is going to be used.

In the following example the same input as above is going to just be Forced On and Inverted, effectively forcing the input off all the time. This can be verified in PLC Diagnostics by toggling INP6 and seeing that it does not change.

```
;Orignal INP6 Debounce Setting Word Value
;0000 0001 0000 0001 0000 0001 0000 0001
;Desired Debounce Setting bits for INP6
```

```
;0000 0000 0000 0000 0000 0101 0000 0000
;modified Debounce Time Word with new bits added in
;0000 0001 0000 0001 1100 0001 0000 0001

;/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
;Program: invert.src
;Purpose: Example invert/force Setup for INP6
;\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
;--Variable Defines
Orig_Deb_Bits_O IS OUT1  ;
Debounce_Bits_M IS MEM1  ;display the bits from the Word
Bit0_M          IS MEM73 ; Deb Time Select Bit 0
Bit1_M          IS MEM74 ; Deb Time Select Bit 1
Bit2_M          IS MEM75 ; Deb Time Select Bit 2
Bit3_M          IS MEM76 ; reserved
Bit4_M          IS MEM77 ; reserved
Bit5_M          IS MEM78 ; reserved
Bit6_M          IS MEM79 ; Invert Input
Bit7_M          IS MEM80 ; Force Input On

Original_Deb_W  IS W1    ;Value read from Deb SV
Temp_Deb_W      IS W2    ;Calculated Word value
Final_Deb_W     IS W3    ;final combined Word value

InitialStage    IS STG1
MainStage       IS STG2

;begin program

;============
InitialStage
;============
IF 1==1 THEN Original_Deb_W = SV_PLC_DEBOUNCE_2   ;read the Settings SV
IF 1==1 THEN Final_Deb_W = Original_Deb_W

;zero out the bits that are going to be replaced
```

```
IF 1==1 THEN BITRST Final_Deb_W 8
IF 1==1 THEN BITRST Final_Deb_W 9
IF 1==1 THEN BITRST Final_Deb_W 10
IF 1==1 THEN BITRST Final_Deb_W 11
IF 1==1 THEN BITRST Final_Deb_W 12
IF 1==1 THEN BITRST Final_Deb_W 13
IF 1==1 THEN BITRST Final_Deb_W 14
IF 1==1 THEN BITRST Final_Deb_W 15

;Setup the 8 bits for the Setting Byte
IF 1==1 THEN SET Bit0_M ;
            ,RST Bit1_M ;binary 1 for Time 1
            ,RST Bit2_M ;
            ,RST Bit3_M ;unused
            ,RST Bit4_M ;unused
            ,RST Bit5_M ;unused
            ,SET Bit6_M ;Invert the Input
            ,SET Bit7_M ;Force On the Input

;move the 8 bits to a Word
IF 1==1 THEN BTW Temp_Deb_W Bit0_M 8
;shift the values to the 2nd byte as calculated above
IF 1==1 THEN LSHIFT Temp_Deb_W 8

;copy the new 8 bits to the final word
IF 1==1 THEN Final_Deb_W = Final_Deb_W + Temp_Deb_W

;save the values back to the SV
IF 1==1 THEN SV_PLC_DEBOUNCE_2 = Final_Deb_W

;write out bits of final Word settings for verification
IF 1==1 THEN WTB Final_Deb_W Debounce_Bits_M 32
IF 1==1 THEN WTB Original_Deb_W Orig_Deb_Bits_O 32

IF 1==1 THEN JMP MainStage
;no MainStage shown because there is nothing to do there for this example
```

Compiler Errors

There are many ways that a program can be changed so that it does not function as intended. Always start at the top of the list of errors because fixing earlier problems may fix repeat errors later on. For example if you fail to define a variable that is used throughout the program, every reference to that variable will generate an error.

If the compiler hangs or doesn't generate any output after ten seconds it is stuck on something and will not compile the program most likely. Press Ctrl.+c to cancel compilation and undo the last changes if the problem just started occurring. If you think the changes are valid double check the usage and then send the file before and after to Tech. Support.

There is a limit of about 1000 characters per line that the compiler can digest so do not go above that limit. Typically you should stick to 80 characters so that the document can be printed and not cause the printout to look different than on your monitor.

The easiest to fix are compiler related syntax or expression errors. The first to sections details what potential errors mean.

Warnings

Already Defined

This message does not cause compilation to fail, but you should address this problem because using two different variables to do two different things will cause very hard to detect logical errors.

Direct PLC Reference

This Warning occurs when you test or use a specific data type directly by name. An example is using `MEM1` instead of defining `Test_M IS MEM1` and using `Test_M` in the program.

General Errors

These errors specify problems with the program that are more serious than an undefined variable. They are not related to the code in the Program, but the way that the compiler was called.

Malformed Command Line

This error means the the attempt to compile the program failed because no source file was specified.

Unrecognized Command Line Option

An invalid switch was used on the command line. Presently there are two options for switches. Neither of them should be used except for testing and debugging the compiler itself. The output is not usable as a PLC Program. The options are -i for outputting a CSV version of the program to be used as an include file and -d is for verbose debugging of what the compiler does with each token in the program. This may be used when there is a suspected problem with the compiler.

Error Opening File

The specified source file was not found in the directory that mpucomp was called from. There are two ways to compile the program. The easiest is to open a command prompt and change directories to the folder where the source file is located. Then you can issue the command *mpucomp(.exe) source.src mpu.plc*. On Windows you must have mpucomp.exe in the same folder as the source file unless you add c:\cncm\bin to your PATH. In Linux this has been done for you so mpucomp does not need to be in the same folder as the source file. The other way of compiling a program is to call mpucomp and give the absolute path to the source file and/or destination file. For example, the source file is in /cncroot/c and the command line indicates the current folder is /cncroot. To compile the source file and get the PLC program to the correct folder *mpucomp /cncroot/c/source.src /cncroot/c/cncm/mpu.plc* must be executed.

Syntax Errors

These are errors related to specific lines in the PLC program. Usually the offending line and column number is listed along with the specific error. Sometimes the row is one farther down the program than it should be so look around the area specified.

Compilation Failed

This message occurs when there are problems in the source file. It is displayed as the last message after other error messages. This message will not appear if there are too many errors.

Too Many Errors

This message is displayed when more than 25 errors occur at compile time. This can easily happen if you forget to define a Label and use it often in the program.

Undefined Label

This message occurs when a valid variable name is used without it being defined in the definition section of the PLC program.

[THEN, Word Type, Stage, Output, Parenthesis] expected

This message is generated when something is expected, but is not found. A Parenthesis is expected on the right if you put on one the left of a variable or expression.

```
IF INP1) THEN SET OUT1
```

End Of File Expected

This message can be generated many different ways, but the meaning is that a line was not written correctly. It can be caused because the `IF` at the start of a line was not used after the first `IF` statement in the program.

[Data Type] Out of Bounds

This message can occur on any Data Type when using the Indexing ability in the compiler. The error is generated when you make a constant reference to a Data Type number that cannot be accessed. For example if you try `IF 1==1 THEN SET OUT[500000000]` you will get this error. It is possible to fool the compiler and index out of range. See Internal PLC Fault Checking for how to detect this occurring and why it is so catastrophic.

Invalid Action

This error occurs when doing something that is not allowed. This includes trying to store a value into a Bit data type or a Constant define and failing to put a start location for a Range.

```
CONST1 IS 1              ;constant value
IF 1==1 THEN CONST1 = 10 ;cannot re-assign constant values
IF 1==1 THEN MEM1 = 10   ;cannot store Word values into Bits
IF 1==1 THEN .. OUT20    ;must include starting point for Range
IF 1==1 THEN SET W1      ;SET cannot be applied to Words
```

Rung Expected IF

The Keyword `IF` was not found after a valid `STG` reference.

```
STG2
1==1 THEN (OUT1)
```

Rung Expected THEN

The Keyword `THEN` is missing from the line specified.

```
IF 1==1 SET OUT1
```

JMP Expected STG or FSTG

If you call the `JMP` command and do not specify what Stage to Jump to, this message is

displayed.

System Variable is Read-Only

Attempting to modify a Read-only System Variable will result in this message displayed. The Read-only variables are ones that CNC11 has write control of here.

MSG Expected Word Reference

This error occurs when you try to use the `MSG` command with no Word variable or with an invalid data type. If no data type is specified, then the error will actually show up on the next line of the program and not show the MSG reference at all in most cases. This occurs because the compiler does not assume that a line feed is the end of a command. For example the first `IF` statement spanning two lines in the following code will compile whereas the second on only one line will not.

```
IF 1==1 THEN MSG
W1
IF 1==1 THEN MSG INP1
```

SMSG Expected String Reference

The `SMGS` functionality expects a string value and anything else causes this error. A string is defined as text between double quotes.

```
IF 1==1 THEN SMSG W1   ;not a string
```

BCD/BIN Expected Word Reference

If you fail to put a Word type variable right after the `BCD` and `BIN` command this error occurs.

```
IF 1==1 THEN BIN MEM1 32
```

BCD/BIN Cannot Use Bracketed Reference

Indexes are not allowed on Words when doing the `BCD` or `BIN` commands.

```
IF 1==1 THEN BCD W[10]
```

WTB Expected Word Reference

If you fail to put a Word type variable right after the WTB command this error occurs.

```
IF 1==1 THEN WTB MEM1 32
```

WTB Expected OUT/MEM Reference

If the Memory or Output location is not specified, this error occurs.

```
IF 1==1 THEN WTB W1 33
```

WTB Number of Bits Must be 1-32

When using the WTB command you can specify a number of bits to write. That value must be from 1 to 32. The incorrect value is shown after the error.

```
IF 1==1 THEN WTB W1 MEM1 33
```

BTW Expected Word Reference

If you fail to put a Word type variable right after the BTW command this error occurs.

```
IF 1==1 THEN BTW MEM1 32
```

BTW Expected INP/OUT/MEM Reference

If the Input, Memory or Output location is not specified, this error occurs.

```
IF 1==1 THEN WTB W1 33
```

BTW Number of Bits Must be 1-32

When using the BTW command you can specify a number of bits to write. That value must be from 1 to 32. The incorrect value is shown after the error.

```
IF 1==1 THEN BTW W1 MEM1 33
```

BITSET/RST Bit Must be 0-31

When using the BITSET or BITRST commands you must specify the bit number to `SET` or `RST`. That value must be from 0 to 31. The incorrect value is shown after the error.

```
IF 1==1 THEN BITSET W1 43
```

BITSET/RST Expected an Integer Value

Bits can only be changed by positive Constant Integer numbers so using a floating-point number, negative or Word value to try and change one will fail.

```
IF 1==1 THEN BITRST W1 1.5
IF 1==1 THEN BITSET W1 W2
IF 1==1 THEN BITSET W1 -10
```

BITSET/RST Word Indexing Not Allowed

Using the Index ability is not allowed when doing a BITSET/RST command.

```
IF 1==1 THEN BITSET W[W2] 2
```

BITSET/RST Expected Word Reference

If you fail to put a Word type variable right after the BITSET or BITRST command this error

occurs.

```
IF 1==1 THEN BITSET MEM1 4
```

BITTST MEM Indexing Not Allowed

Using the Index ability is not allowed when doing a BITTST command.

```
IF 1==1 THEN BITTST W2 2 MEM[5+1]
```

BITTST Expected MEM

The last of three arguments to a BITTST must point to a Memory Bit and it was not found or the wrong type was specified.

```
if 1==1 then BITTST w1 2 OUT1
```

BITTST Expected Word Reference

If you fail to put a Word type variable right after the BITTST command this error occurs.

```
IF 1==1 THEN BITTST INP1 5 MEM1
```

BITTST Bit Must be 0-31

When using BITTST you must specify the bit number to check in the Word. That value must be from 0 to 31. The incorrect value is shown after the error.

```
IF 1==1 THEN BITTST W1 43 MEM1
```

BITTST Expected Integer Value

Bits can only be changed by positive Constant Integer numbers so using a floating-point number, negative or Word value to try and change one will fail.

```
IF 1==1 THEN BITTST W1 1.5 MEM1
```

```
IF 1==1 THEN BITTST W1 W2 MEM1
```

```
IF 1==1 THEN BITTST W1 -10 MEM1
```

BITTST Word Indexing Not Allowed

Using the Index ability is not allowed when doing a BITTST command.

```
IF 1==1 THEN BITTST W[2+5] 2
```

LSHIFT/RSHIFT Bit Must be 0-31

When using the LSHIFT/RSHIFT commands you must specify the number of bits to shift. That value must be from 0 to 31. The incorrect value is shown after the error.

```
IF 1==1 THEN LSHIFT W1 99
```

LSHIFT/RSHIFT Expected Integer Value

Bits can only be changed by positive Constant Integer numbers so using a floating-point number, negative or Word value to try and change one will fail.

```
IF 1==1 THEN LSHIFT W1 1.5
IF 1==1 THEN LSHIFT W1 W2
IF 1==1 THEN LSHIFT W1 -10
```

LSHIFT/RSHIFT Word Indexing Not Allowed

Using the Index ability is not allowed when doing a LSHIFT/RSHIFT command.

```
IF 1==1 THEN LSHIFT W[10+2] 2
```

LSHIFT/RSHIFT Expected Word Reference

If you fail to put a Word type variable right after the LSHIFT/RSHIFT command this error occurs.

```
IF 1==1 THEN LSHIFT MEM1 4
```

Constant Integer Expression Label Not Found

If you reference a constant when defining other constants, but the referenced constant does not exist yet you will get this message. The following example will generate this error.

```
CONST2 IS 2*CONST3
```

Constant Integer Expression Expected Right Parenthesis

This error is generated when a left parenthesis is used to declare a constant, but the right parenthesis is left off.

```
CONST1 IS (10+2
```

Constant Integer Factor Expected

Constant math can only be done on Integer values. Putting a Floating-point value in a constant define that does math will give this error.

```
CONST1 IS (10+2.2)
```

Constant Integer Expression Label Does Not Reference an Integer

If you create a Constant that stores a floating-point value and then use it in the declaration of an Integer Constant you will get this message.

```
CONST1 IS 11.2
CONST2 IS (CONST1/2)
```

Constant Integer Expression Label not Found

This error only occurs when a previous Constant expression definition is invalid and it is referenced by another Constant expression definition.

```
CONST1 IS (11.2)    ;invalid because of parentheses, only constant numbers allowed
CONST2 IS (CONST1/2);invalid CONST1 causes another error.
```

Relational Operator Expected

A comparison of data types that need a Relational Operator used on them to evaluate to true or false is needed rather than the used operator.

```
IF W1 = W2 THEN SET OUT1 ;assignment not allowed before THEN
IF !W2 THEN SET OUT2     ;Logical Operator not allowed on Words
```

System Variable Bits Cannot be Used With '..'

The Range selector cannot be used on any System Variables.

```
IF 1==1 THEN SET SV_ENABLE_AXIS_1 .. SV_ENABLE_AXIS_8
```

Range Extension Expected OUT/MEM/STG/FSTG/T

When specifying a Range, the end of the Range was not specified. This error will be shown on the next line after the error actually occurs. One of the listed types must be used to finish the Range. It must be the same type that is on the left of the Range select.

```
IF 1==1 THEN SET OUT1 ..
```

Range Error. End is Before Start

The Range selectors were specified out of order. They must be used from lower to high bits.

```
IF 1==1 THEN SET OUT10 .. OUT1
```

SET/RST Expected OUT/MEM/STG/FSTG/T/Modifiable SV

This message is generated when you try to `SET` or `RST` something that the action cannot be applied to or nothing is supplied. Inputs, One-Shots, Words, and Word type System Variables are all examples of things that cannot be `SET` or `RST`.

```
IF 1==1 THEN SET
IF 1==1 THEN SET INP1
IF 1==1 THEN SET PD1
```

Coil Expected PD/OUT/MEM/Modifiable SV

This error occurs when trying to use Coils on Data Types that cannot be `SET`/`RST`. This includes Word types and Timers. There is no point turning on a Timer to check that only lasts one pass

of the PLC program. Words can have bits in them SET and RST using BITSET and BITRST.

```
IF INP1 THEN (T2)
```

Expected Right Bracket

This message occurs when a left bracket is used and a right one is not found.

if 1==1 then set out[10+3

if 1==1 then set out[10+3)]

Expected Left Bracket

This error occurs because a variable that can use Indexing had a space after the variable name and no bracket was found. The compiler then expects to see a Left Bracket. You may not be doing an Index reference at all, but merely had a space where there should not be one.

```
IF 1==1 THE SET OUT 10]
```

```
IF 1==1 THEN BTW W 3] 10
```

Bad Definition

The attempt at defining a variable is not written correctly. Brackets and negative numbers are not allowed. Constant defines are used for messaging mostly.

```
Const is [10] ; brackets cannot be used in the definition section of a PLC program
```

Number Expected Right Parenthesis

A right Parenthesis was expected in a math calculation in the Program.

```
IF 1==1 THEN W1 = (10 + 50
```

Numerical Factor Expected Right Parenthesis

Any command such as ABS, SIN, etc. is listed and then the calculation to be done is in parentheses afterwards. If the parenthesis on the right is missing, this error occurs.

```
if 1==1 then W1 = abs (10*5
```

Numerical Factor Expected Left Parenthesis

Any command such as ABS, SIN, etc. is listed and then the calculation to be done is in parentheses afterwards. If the parenthesis on the left is missing, this error occurs.

```
if 1==1 then FW1 = sin (3.14159265/180
```

ATAN2/POW Expected Right Parenthesis

In either an ATAN2 or POW operation, the right parenthesis was left off.

```
IF 1==1 THEN FW1 = ATAN2 (3.141592/180, 3.141592
```

ATAN2/POW Expected Comma

In either an ATAN2 or POW operation, the right parenthesis was left off.

```
IF 1==1 THEN FW1 = POW (1 1)
```

ATAN2/POW Expected Left Parenthesis

In either an ATAN2 or POW operation, the right parenthesis was left off.

```
IF 1==1 THEN FW1 = ATAN2 3.141592/180, 3.141592)
```

Expected Right Parenthesis

This error occurs when using a Coil on an Output or Memory Bit when there is a left parenthesis and not a right.

```
IF 1==1 THEN (OUT1
```

Boolean Factor Expected Right Parenthesis

In a Boolean check using parentheses, the right parenthesis was left off.

```
IF (INP1 THEN SET OUT1
```

Assignment Error

This error occurs when a value is getting stored to a variable and it cannot be completed. One example is to Index to a floating-point variable.

```
IF INP1 THEN W[10.4] = 1
```

Application Examples

These examples strive to give examples of using some of the various PLC functionality that isn't used in a standard PLC program. The examples are not meant to be useful in what is accomplished, but rather in how it is accomplished.

Toggle an Output Every Second

Pseudo-code

```
In InitialStage Second_On_T = 1000, Second_Off_T = 1000, SET  Second_Off_T
IF Second_On_T THEN RST  Second_On_T, SET  Second_Off_T, RST OUT1
IF Second_Off_T THEN RST  Second_Off_T, SET  Second_On_T, SET OUT1
```

Aux Key Jogging

Pseudo-code

```
IF Aux2 THEN Y+_Jog
IF Aux8 THEN Y-_Jog
IF Aux4 THEN X-_Jog
IF Aux6 THEN X+_Jog
IF Aux3 THEN Z+_Jog
IF Aux9 THEN Z-_Jog
IF Aux1 THEN W+_Jog
IF Aux7 THEN W-_Jog
```

Aux Key Override of M-Code

Pseudo-code

```
IF M-Code && (CNC_PROG_RUNNING || AUX14) THEN SET M-Code_Stage
```

Wait One Second Before Jogging on Key Press

Pseudo-code

```
IF !(all jog keys) THEN RST Jog_T
IF any jog key THEN SET Jog_T
IF Jog_T THEN DO Ax#_Jog
```

Interpret Enter Key as Cycle Start in MDI*

*Note that Centroid does not recommend doing this. It is merely an example of what is

possible as far as customizing the User experience. Often, after typing a command in MDI it is natural to hit Enter to try to run the line, but that merely causes a prompt to push Cycle Start to appear. With this change that will not happen. This functionality should only be allowed when in MDI to prevent unexpected motion.

Pseudo-code

```
IF MDI_MODE && AllowKBJog && Kb_Enter_Key THEN Cycle_Start
```

Count Machine On Time

Use a Timer to count one day and then increment a word. Update P171 every 30 min. to keep track of the value. Read the value of P170 in the Initial stage and increment values to it.

Custom M-Codes

Using M94/M95 Bits

Pick an unused M94/M95 bit and assign it the name of the custom M-Code. Create an mfuncXXX.mac file that has the same number as the assigned M-Code. Put whatever you want in the M-Code and if the PLC program needs to do something, put code in to check for MXXX being SET or RST.

Using One M94/M95 Bit and a Parameter

In the custom M-Code SET the bit with M94 and G10 a parameter in the 171-177 range to a different binary value for each M-Code. Use 1, 2, 4, 8, 16, 32, etc. so that you can use BITTST. Alternatively, you can just use decimal numbers and check Parameter == 1, ==2, ==3, etc.

Customizing Standard M-Codes

In the basic MPU11 PLC programs there are four “customized” M-Codes by default. They are for Spindle directions and Coolant. The reason that they are considered custom is that rather than using the built in CNC software functionality, files were created to extend the original functionality. Now there is a message printed on screen that halts execution if an auto spindle or coolant function is called and prompts the user to put the control in auto mode. This can be done for many of the M-Codes to expand the usability of or modify the given feature. The following example shows how the M3 and M4 M-Codes were changed.

Automatic Spindle On/Off – M3/M4

When a default M3 is commanded the Spindle should start spinning in the clockwise direction. If the control is in manual spindle mode though, the spindle will not turn on and program execution will continue like everything is fine. This is bad, so a warning should be given to the user and execution paused. The same holds true for Automatic Coolant which is why the custom M7 and M8 are needed.

```
;-------
;M3 macro
; Displays message to select auto spindle mode if it is not SET
;-------
;if searching or backplotting a program, skip the macro
IF #4202 || #4201 THEN GOTO 200
M95 /2 ; turn off CCW spindle
M94 /1 ;turn on CW spindle

;if AutoSpindle i.e. OUT1058 (a.k.a. JPO2) is set, exit the Macro
IF #61058 THEN GOTO 200
G4 P.1
#140 = 1.5
N100
```

```
; If not in AutoSpindle Mode display a message and wait
IF !#61058 THEN M225 #140 "Please Select Auto Spindle To Continue!"
G4 P.5
IF !#61058 THEN GOTO 100
N200
```

Troubleshooting and Changing PLC Programs

This chapter contains some suggestions for debugging problems in your PLC program.

Write Down and Think Through Changes to the Program

Before making any changes to a PLC Program you should always take a report. Write out as specifically as possible the exact functionality that is to be implemented. This potentially includes timeouts, fault conditions, interactions with other Inputs and Outputs, messages that will need to be shown to the User, etc.

PLC Diagnostic Screen

When at the main screen, push Alt.-i to cause the PLC Diagnostic screen to appear. It appears over running jobs so it should not be left on all the time, especially if there are user prompts in the program. The red and green dots represent Inputs, Outputs, Memory and Stage Bits. Red means the Bit is off or open and Green means the Bit is closed or on. This is true in all cases for the Bits. Keep in mind that setting the Invert or Force Debounce bits will change the way the Inputs are reported to the PLC program and thus the PLC Diagnostic screen.

The screen shows the state of Bits 80 at a time for the entire range available for that Bit type and 12 Words per page. All 1312 available Inputs and Outputs are shown as are all 1024 Memory Bits and 256 Stages. Presently the 44 Words that are sent up to CNC11 are shown as well. This means that all the Jog Panel buttons and LEDs as well as Spindle DAC outputs can be seen without mirroring them to the 1 to 80 range. This is true as of CNC11 3.0 rev84 beta release.

Future improvements include adding FW, DW, DFW, FSTG, and Timer display to the screen.

PLC Bit-State Dump

When the PLC Diagnostic screen is showing, pushing the Ctrl.- i key combination will cause the state of all the Bit variables to be stored to a file called plcstate.txt. This file is overwritten every time the operation is performed. This feature will be implemented in the future.

DUMP

Use the DUMP PLC command to print all of the values of the first 64 Word type variables to disk. This can be called at any time to check status words or ATC bin position in a certain stage. Make sure to call this infrequently because it does take quite a long time to write to disk compared to the time to execute the PLC program once.

Echo to a Memory Bit

Transient signals that are on and then quickly off may not show up reliably on the PLC Diagnostic screen long enough to see because the display only updates 30 times/second. In this case, to verify that the bit is changing at all you can SET a Memory Bit if the Bit in question ever gets SET or RST. It is a good idea to incorporate the ability to RST these diagnostic Bits into an unused Aux Key so that you do not have to power off the system to RST them.

Use Stages

Group new features into a Stage so that it can be turned on and off to check for problems. Use more stages for more complex features to narrow down where you need to troubleshoot.

Communication In/Out Faults

DriveBus

These faults occur when the communication on the DriveBus is disrupted. The DriveBus goes out on fibers 4 and 5 from the MPU11 and across the Drive Communication In/Out wire connections on the ALLIN1DC, Optic4 and DC3IOB. If any of these connections exist and are checked in the PLC program correctly, they will help diagnose connection problems.

PLCBus

The PLCBus also should be checked in the PLC program to make sure that communication is in a good state. There should be checks for the Fibers 1 and 3 and miniPLCBus connections if expansion boards are used.

Appendix A: Example PLC program

ALLIN1DC DC system example

This is the basic PLC program using an ALLIN1DC drive and PLC.

```
;-------------------------------------------------------------------------------
; File:        allin1dc-basic.src
; Programmer:  Scott Pratt, Keith Dennison, Marc Leonard
; Date:        7 April 2010
; Purpose:     PLC for MPU11 and allin1dc + keyboard jog
; Requires:    Requires CNC11 R83+
;
; $Id: allin1dc-basic.src 510 2011-03-02 11:05:28Z marc $
;
; Changes: 101123 KD - Added Aux 10, 11, 12 assignments to the gray Aux keys.
;
;NOTE Changes to keyboard jogging bmp needed.  See MEM400+
;     "Invalid key" messages will need to be suppressed

; Mpu11 based systems have the ability to invert, force and/or select a custom
; debounce time on PLC inputs 1-240 using SV_PLC_DEBOUNCE_1-SV_PLC_DEBOUNCE_64.
; Jog Panel inputs are modified in the same manner using SV_JOG_LINK_DEBOUNCE_1
; -SV_JOG_LINK_DEBOUNCE_64. See system variable section for more information.

; The Mpu11 board includes connections for several types of auxiliary I/O.
; 4 digital "high speed" inputs (INP769-772) typically used for probe/TT1
; related functions, 3 auxiliary digital inputs (INP784-786), 11 Digital inputs
; used for MPG increment and axis selection and 2 auxiliary digital outputs
; (Out770-771).

; ALLIN1DC Physical I/O: While each ALLIN1DC that is installed reserves (maps)
; 16 inputs and 16 Outputs, only 16 inputs and 9 outputs are accessible through
; hardware.

; Digital Inputs: The ALLIN1DC provides 16 inputs, 10 of which are available for
; general purpose use.  The first 6 inputs (1-6) are dedicated for limit switch
; use and must be either wired to a NC limit switch or defeated.  All 16 inputs
; can be configured (in banks of 4) for 5, 12 or 24VDC operation in either a
; sourcing or sinking configuration.

; Analog input: The ALLIN1DC provides a single 12 bit analog input which is
; mapped to inputs 241-252.  LSB = 241.  This input can be configured for the
; following input:

;  1. 0 - 5VDC
;  2. 0 - 10VDC
;  3. -5 - +5VDC
;  4. -10 - +10VDC

; Please see the ALLIN1DC manual for configuration information
```

```
; Outputs: The ALLIN1DC has 9 relay contact closure outputs.  Outputs 1-7 are
; are SPST type relays while Outputs 8 & 9 are SPDT type relays.

; Analog outputs: The 12 bit 0-10VDC analog output on the ALLIN1DC
; is mapped to outputs 241-252.  NOTE: The spindle speed command that comes down
; from the PC (SV_PC_DAC_SPINDLE_SPEED) is a 16 bit integer value from 0-65535
; that must be converted to a 12 bit value from 0-4095 by the PLC.  The PLC
; handles gear ranges by looking at the state of inputs and reading parameters
; (or hard coded values) to determine the ratio needed for adjusting the spindle
; speed display system variable

;-----------------------------------------------------------------
;                        CONSTANT DEFINITIONS
;-----------------------------------------------------------------

PLC_EXECUTOR_FLT_MSG IS 257; (1+256)

AXIS1_INFLT          IS 1282;(2+256*5) Fiber to MPU11 has a problem
AXIS2_INFLT          IS 1538;(2+256*6)
AXIS3_INFLT          IS 1794;(2+256*7)
AXIS4_INFLT          IS 2050;(2+256*8)
AXIS5_INFLT          IS 2306;(2+256*9)
AXIS6_INFLT          IS 2562;(2+256*10)
AXIS7_INFLT          IS 2818;(2+256*11)
AXIS8_INFLT          IS 3074;(2+256*12)

AXIS1_OUTFLT         IS 3330;(2+256*13) Fiber to axis drive has a problem
AXIS2_OUTFLT         IS 3586;(2+256*14)
AXIS3_OUTFLT         IS 3842;(2+256*15)
AXIS4_OUTFLT         IS 4098;(2+256*16)
AXIS5_OUTFLT         IS 4354;(2+256*17)
AXIS6_OUTFLT         IS 4610;(2+256*18)
AXIS7_OUTFLT         IS 4866;(2+256*19)
AXIS8_OUTFLT         IS 5122;(2+256*20)

AXIS_FLT_CLR         IS 5378;(2+256*21)

PLC_INFLT            IS 5634;(2+256*22)
PLC_OUTFLT           IS 5890;(2+256*23)
PLC_FLT_CLR          IS 6146;(2+256*24)

SPINDLE_FAULT_MSG    IS 7681;(1+256*30)
PROBE_FAULT_MSG      IS 8705;(1+256*34)

KB_JOG_MSG           IS 8962;(2+256*35)

LUBE_FAULT_MSG       IS 9217;(1+256*36)
LUBE_WARNING_MSG     IS 9218;(2+256*36)
PROBE_JOG_FAULT_MSG  IS 9473;(1+256*37)

MIN_SPEED_MSG        IS 9730;(2+256*38)
SOFTWARE_EXIT_MSG    IS 9985;(1+256*39)
```

```
MSG_CLEARED_MSG      IS 25345;(1+256*99)

;---------------------------------------------------------------
;                        INPUT DEFINITIONS
;               Closed = 1 (green)  Open = 0 (red)
;---------------------------------------------------------------
Ax1_MinusLimitOk      IS INP1
Ax1_PlusLimitOk       IS INP2
Ax2_MinusLimitOk      IS INP3
Ax2_PlusLimitOk       IS INP4
Ax3_MinusLimitOk      IS INP5
Ax3_PlusLimitOk       IS INP6
;spare                IS INP7
;spare                IS INP8
LubeOk                IS INP9  ;Lube is "ok" when input is closed      (*)
SpindleInverterOk     IS INP10 ;Inverter is "ok" when input is closed  (*)
EStopOk               IS INP11
SpinLowRange          IS INP12
;spare                IS INP13
;spare                IS INP14
;spare                IS INP15
;spare                IS INP16

;If a PLC expansion board (PLCADD1616) is used, the additional inputs will
;begin at input 17.

; (*) If SpindleInverterOk or LubeOk is moved to a different input, then
;     the P178 inversion logic at the start of MainStage will also need
;     to be updated, to invert the correct input.

;---------------------------------------------------------------
; INP769 - INP784 encompass the MPU11 onboard input connections
; which are generally used for MPG and probing functions.
;---------------------------------------------------------------
MechanicalProbe       IS INP769
DSPProbe              IS INP770
ProbeDetect           IS INP771
ProbeAux              IS INP772
MPG_Inc_X_1           IS INP773
MPG_Inc_X_10          IS INP774
MPG_Inc_X_100         IS INP775
MPG_AXIS_1            IS INP776
MPG_AXIS_2            IS INP777
MPG_AXIS_3            IS INP778
MPG_AXIS_4            IS INP779
MPG_AXIS_5            IS INP780
MPG_AXIS_6            IS INP781
MPG_AXIS_7            IS INP782
MPG_AXIS_8            IS INP783

;---------------------------------------------------------------
;                  ALLIN1DC PLC Output Definitions
;       Logic 1 = OUTPUT ON (Green), 0 = OUTPUT OFF (Red)
;---------------------------------------------------------------
```

```
NoFaultOut            IS OUT1  ;SPST Type
Lube                  IS OUT2  ;SPST Type
Flood                 IS OUT3  ;SPST Type
Mist                  IS OUT4  ;SPST Type
InverterResetOut      IS OUT5  ;SPST Type
Clamp                 IS OUT6  ;SPST Type - M10 On, M11 Off & Aux7
SpindleEnableOut      IS OUT7  ;SPST Type
SpindleDirectionOut   IS OUT8  ;SPDT Type
;                     IS OUT9  ;SPDT Type

;Outputs 10-16 are unavailable


; These bits control the actual analog hardware output on the ALLIN1DC.
; Output = 12bit (0-4095) 0-10VDC.
SpinAnalogOutBit0     IS OUT241
SpinAnalogOutBit1     IS OUT242
SpinAnalogOutBit2     IS OUT243
SpinAnalogOutBit3     IS OUT244
SpinAnalogOutBit4     IS OUT245
SpinAnalogOutBit5     IS OUT246
SpinAnalogOutBit6     IS OUT247
SpinAnalogOutBit7     IS OUT248
SpinAnalogOutBit8     IS OUT249
SpinAnalogOutBit9     IS OUT250
SpinAnalogOutBit10    IS OUT251
SpinAnalogOutBit11    IS OUT252

MPG_LED_OUT           IS OUT769

;-----------------------------------------------------------------
;                        Memory Bit Definitions
;-----------------------------------------------------------------
PLCExecutorFault_M      IS MEM1
SoftwareNotReady_M      IS MEM2   ; 0 = okay, 1 = CNC11 not running/ready
MPGManOffFlag_M         IS MEM3

Ax1PlusJogDisabled_M    IS MEM11
Ax1MinusJogDisabled_M   IS MEM12
Ax2PlusJogDisabled_M    IS MEM13
Ax2MinusJogDisabled_M   IS MEM14

MasterEnable_M          IS MEM40  ; 1 = enabled (echo of SV_MASTER_ENABLE)
PLCBus_Oe_M             IS MEM41  ; 1 = okay, 0 = incoming PLC fiber problem
PLCBusExtDevEn_M        IS MEM42  ; 1 = okay, 0 = PLC reports bad output fiber
ADD1616Ok1_M            IS MEM43

Stop_M                  IS MEM47  ; 0 = okay, 1 = fault (echo of SV_STOP)
Stall_M                 IS MEM48  ; 0 = okay, 1 = stall (echo of SV_STALL_ERROR)
LubeFault_M             IS MEM49  ; 0 = okay, 1 = lube fault
PLCFault_M              IS MEM50  ; 0 = okay, 1 = PLC fault
AxisFault_M             IS MEM51  ; 0 = okay, 1 = drive or drive fiber problem
DriveComFltIn_M         IS MEM52  ; 0 = okay, 1 = incoming drive fiber problem
DriveComFltOut_M        IS MEM53  ; 0 = okay, 1 = outgoing drive fiber problem
```

```
ProbeFault_M            IS MEM54  ; 0 = okay, 1 = tried to start spindle w/probe
JogProbeFault_M         IS MEM55  ; 0 = okay, 1 = tripped probe while jogging
SpindleFault_M          IS MEM56  ; 0 = okay, 1 = spindle drive fault
OtherFault_M            IS MEM57

KbJpActive_M            IS MEM60 ; aka SV_PC_VIRTUAL_JOGPANEL_ACTIVE

Axis1FiberOk_M          IS MEM70
Axis2FiberOk_M          IS MEM71
Axis3FiberOk_M          IS MEM72
Axis4FiberOk_M          IS MEM73
Axis5FiberOk_M          IS MEM74
Axis6FiberOk_M          IS MEM75
Axis7FiberOk_M          IS MEM76
Axis8FiberOk_M          IS MEM77
ProbeMsgSent_M          IS MEM78
true                    IS MEM81
SpinLowRange_M          IS MEM82
SpinHighRange_M         IS MEM85
SpindlePause_M          IS MEM86

DisableKbInput_M        IS MEM102 ;If 1, disable kb jogging
AllowKbInput_M          IS MEM103 ;If 1, allow kb jogging
JogOverOnly_M           IS MEM105
KbOverOnly_M            IS MEM106
UsingFeedrateKnob_M     IS MEM117
WaitingForSleepTimer_M  IS MEM118
X1_M                    IS MEM119
X10_M                   IS MEM120
X100_M                  IS MEM121

OnAtPowerUp_M           IS MEM200
LimitTripped            IS MEM208
LastProbeMode_M         IS MEM210


InvLubeOk_M             IS MEM300  ; P178 Bit 0  (1) ; add 1 to parm 178 if
                                   ; lubeOk is NO
InvSpinInverterOk_M     IS MEM301  ; P178 Bit 1  (2) ; Add 2 to parm 178 if
                                   ; InvSpinInverterOk_M is NO

KbCycleStart_M          IS MEM400 ; "alt" + "s"
KbCycleCancel_M         IS MEM401 ; escape
KbToolCheck_M           IS MEM402 ; "Ctrl" + "t"
KbTogSingleBlock_M      IS MEM403 ; "Ctrl" + "b"
KbIncreaseJogInc_M      IS MEM404 ; "Insert"
KbDecreaseJogInc_M      IS MEM405 ; "Delete"
KbIncFeedOver_M         IS MEM406 ; "ctrl" + "keyboard "+" ("=")
KbDecFeedOver_M         IS MEM407 ; "ctrl" + "keyboard "-"
KbFeedOver100_M         IS MEM450 ; "ctrl" + "\"
KbTogIncContJog_M       IS MEM408 ; "ctrl" + "i"
KbTogFastSlowJog_M      IS MEM409 ; "ctrl" + "f"
KbJogAx1Plus_M          IS MEM411 ; right arrow + KbJpActive_M
KbJogAx1Minus_M         IS MEM412 ; left arrow + KbJpActive_M
```

```
KbJogAx2Plus_M          IS MEM413 ; up arrow + KbJpActive_M
KbJogAx2Minus_M         IS MEM414 ; down arrow + KbJpActive_M
KbJogAx3Plus_M          IS MEM415 ; page up + KbJpActive_M
KbJogAx3Minus_M         IS MEM416 ; page down + KbJpActive_M
KbJogAx4Plus_M          IS MEM417 ; "home"+ KbJpActive_M
KbJogAx4Minus_M         IS MEM418 ; "end" + KbJpActive_M
KbAux1Key_M             IS MEM419 ; "ctrl" + "F1"
KbAux2Key_M             IS MEM420 ; "ctrl" + "F2"
KbAux3Key_M             IS MEM421 ; "ctrl" + "F3"
KbAux4Key_M             IS MEM422 ; "ctrl" + "F4"
KbAux5Key_M             IS MEM423 ; "ctrl" + "F5"
KbAux6Key_M             IS MEM424 ; "ctrl" + "F6"
KbAux7Key_M             IS MEM425 ; "ctrl" + "F7"
KbAux8Key_M             IS MEM426 ; "ctrl" + "F8"
KbAux9Key_M             IS MEM427 ; "ctrl" + "F9"
KbAux10Key_M            IS MEM428 ; "ctrl" + "F10"
KbAux11Key_M            IS MEM429 ; "ctrl" + "F11"
KbAux12Key_M            IS MEM430 ; "ctrl" + "F12"
KbTogRapidOver_M        IS MEM431 ; "ctrl" + "r"
KbTogSpinAutoMan_M      IS MEM432 ; "ctrl" + "a"
KbSpinCW_M              IS MEM433 ; "ctrl" + "c"
KbSpinCCW_M             IS MEM434 ; "ctrl" + "w"
KbSpinStart_M           IS MEM435 ; "ctrl" + "s"
KbSpinStop_M            IS MEM436 ; "ctrl" + "q"
KbFloodOnOff_M          IS MEM437 ; "ctrl" + "n"
KbMistOnOff_M           IS MEM451 ; "ctrl" + "k"
KbTogCoolAutoMan_M      IS MEM438 ; "ctrl" + "m"
KbFeedHold_M            IS MEM439 ; space bar
KbIncSpinOver_M         IS MEM440 ; "ctrl" + ">" (.)
KbDecSpinOver_M         IS MEM441 ; "ctrl" + "<" (,)
KbSpinOver100_M         IS MEM442 ; "ctrl" + "?" (/)

;-----------------------------------------------------------------
; Jog panel keys are referenced as JPI1 through JPI256. Alternatively,
; jog panel inputs can also be referenced as INP1057-INP1312.
;-----------------------------------------------------------------
; Definitions follow JOGBOARD layout top to bottom, left to right

SpinOverPlusKey       IS JPI1  ; Row  1 Column 1
SpinAutoManKey        IS JPI2  ; Row  1 Column 2
Aux1Key               IS JPI3  ; Row  1 Column 3
Aux2Key               IS JPI4  ; Row  1 Column 4
Aux3Key               IS JPI5  ; Row  1 Column 5

SpinOver100Key        IS JPI6  ; Row  2 Column 1
SpinCWKey             IS JPI7  ; Row  2 Column 2
Aux4Key               IS JPI8  ; Row  2 Column 3
Aux5Key               IS JPI9  ; Row  2 Column 4
Aux6Key               IS JPI10 ; Row  2 Column 5

SpinOverMinusKey      IS JPI11 ; Row  3 Column 1
SpinCCWKey            IS JPI12 ; Row  3 Column 2
Aux7Key               IS JPI13 ; Row  3 Column 3
Aux8Key               IS JPI14 ; Row  3 Column 4
```

```
Aux9Key               IS JPI15 ; Row  3 Column 5

SpinStopKey           IS JPI16 ; Row  4 Column 1
SpinStartKey          IS JPI17 ; Row  4 Column 2
Aux10Key              IS JPI18 ; Row  4 Column 3
Aux11Key              IS JPI19 ; Row  4 Column 4
Aux12Key              IS JPI20 ; Row  4 Column 5

CoolAutoManKey        IS JPI21 ; Row  5 Column 1
CoolFloodKey          IS JPI22 ; Row  5 Column 2
CoolMistKey           IS JPI23 ; Row  5 Column 3
Aux13Key              IS JPI24 ; Row  5 Column 4
Aux14Key              IS JPI25 ; Row  5 Column 5

IncrContKey           IS JPI26  ; Row  6 Column 1
x1JogKey              IS JPI27  ; Row  6 Column 2
x10JogKey             IS JPI28  ; Row  6 Column 3
x100JogKey            IS JPI29  ; Row  6 Column 4
MPGKey                IS JPI30  ; Row  6 Column 5

Ax4PlusJogKey         IS JPI31  ; Row  7 Column 1
UnusedR7C2Key         IS JPI32  ; Row  7 Column 2
Ax2PlusJogKey         IS JPI33  ; Row  7 Column 3
UnusedR7C4Key         IS JPI34  ; Row  7 Column 4
Ax3PlusJogKey         IS JPI35  ; Row  7 Column 5

UnusedR8C1Key         IS JPI36  ; Row  8 Column 1
Ax1MinusJogKey        IS JPI37  ; Row  8 Column 2
FastSlowKey           IS JPI38  ; Row  8 Column 3
Ax1PlusJogKey         IS JPI39  ; Row  8 Column 4
UnusedR8C5Key         IS JPI40  ; Row  8 Column 5

Ax4MinusJogKey        IS JPI41  ; Row  9 Column 1
UnusedR9C2Key         IS JPI42  ; Row  9 Column 2
Ax2MinusJogKey        IS JPI43  ; Row  9 Column 3
UnusedR9C4Key         IS JPI44  ; Row  9 Column 4
Ax3MinusJogKey        IS JPI45  ; Row  9 Column 5

CycleCancelKey        IS JPI46  ; Row 10 Column 1
SingleBlockKey        IS JPI47  ; Row 10 Column 2
ToolCheckKey          IS JPI48  ; Row 10 Column 3
FeedHoldKey           IS JPI49  ; Row 10 Column 4
CycleStartKey         IS JPI50  ; Row 10 Column 5

;-------------------------------------------------------------
;                    Feedrate Override Knob
;-------------------------------------------------------------
JpFeedOrKnobBit0      IS JPI193
JpFeedOrKnobBit1      IS JPI194
JpFeedOrKnobBit2      IS JPI195
JpFeedOrKnobBit3      IS JPI196
JpFeedOrKnobBit4      IS JPI197
JpFeedOrKnobBit5      IS JPI198
JpFeedOrKnobBit6      IS JPI199
```

```
JpFeedOrKnobBit7      IS JPI200
JpFeedOrKnobBit8      IS JPI201 ; Current jog panels send first 8 bits
JpFeedOrKnobBit9      IS JPI202 ; unused
JpFeedOrKnobBit10     IS JPI203 ; unused
JpFeedOrKnobBit11     IS JPI204 ; unused
JpFeedOrKnobBit12     IS JPI205 ; unused
JpFeedOrKnobBit13     IS JPI206 ; unused
JpFeedOrKnobBit14     IS JPI207 ; unused
JpFeedOrKnobBit15     IS JPI208 ; unused

;---------------------------------------------------------------
;              Jog Panel Output (LED) Definitions
;       Jog Panel LED's can be addressed as JPO1 - JPO256
;                             OR
;                       OUT1057 - OUT1312
;---------------------------------------------------------------
; Definitions follow JOGBOARD layout top to bottom, left to right
;
SpinOverPlusLED       IS JPO1  ; Row  1 Column 1
SpinAutoModeLED       IS JPO2  ; Row  1 Column 2
Aux1LED               IS JPO3  ; Row  1 Column 3
Aux2LED               IS JPO4  ; Row  1 Column 4
Aux3LED               IS JPO5  ; Row  1 Column 5

SpinOver100LED        IS JPO6  ; Row  2 Column 1
SpindleCWLED          IS JPO7  ; Row  2 Column 2
Aux4LED               IS JPO8  ; Row  2 Column 3
Aux5LED               IS JPO9  ; Row  2 Column 4
Aux6LED               IS JPO10 ; Row  2 Column 5

SpinOverMinusLED      IS JPO11 ; Row  3 Column 1
SpindleCCWLED         IS JPO12 ; Row  3 Column 2
Aux7LED               IS JPO13 ; Row  3 Column 3
Aux8LED               IS JPO14 ; Row  3 Column 4
Aux9LED               IS JPO15 ; Row  3 Column 5

SpinStopLED           IS JPO16 ; Row  4 Column 1
SpinStartLED          IS JPO17 ; Row  4 Column 2
Aux10LED              IS JPO18 ; Row  4 Column 3
Aux11LED              IS JPO19 ; Row  4 Column 4
Aux12LED              IS JPO20 ; Row  4 Column 5

CoolAutoManLED        IS JPO21 ; Row  5 Column 1
CoolFloodLED          IS JPO22 ; Row  5 Column 2
CoolMistLED           IS JPO23 ; Row  5 Column 3
Aux13LED              IS JPO24 ; Row  5 Column 4
Aux14LED              IS JPO25 ; Row  5 Column 5

IncrContLED           IS JPO26  ; Row  6 Column 1
x1JogLED              IS JPO27  ; Row  6 Column 2
x10JogLED             IS JPO28  ; Row  6 Column 3
x100JogLED            IS JPO29  ; Row  6 Column 4
MPGLED                IS JPO30  ; Row  6 Column 5
```

```
Ax4PlusJogLED         IS JPO31  ; Row  7 Column 1
UnusedR7C2LED         IS JPO32  ; Row  7 Column 2
Ax2PlusJogLED         IS JPO33  ; Row  7 Column 3
UnusedR7C4LED         IS JPO34  ; Row  7 Column 4
Ax3PlusJogLED         IS JPO35  ; Row  7 Column 5

UnusedR8C1LED         IS JPO36  ; Row  8 Column 1
Ax1MinusJogLED        IS JPO37  ; Row  8 Column 2
FastSlowLED           IS JPO38  ; Row  8 Column 3
Ax1PlusJogLED         IS JPO39  ; Row  8 Column 4
UnusedR8C5LED         IS JPO40  ; Row  8 Column 5

Ax4MinusJogLED        IS JPO41  ; Row  9 Column 1
UnusedR9C2LED         IS JPO42  ; Row  9 Column 2
Ax2MinusJogLED        IS JPO43  ; Row  9 Column 3
UnusedR9C4LED         IS JPO44  ; Row  9 Column 4
Ax3MinusJogLED        IS JPO45  ; Row  9 Column 5

CycleCancelLED        IS JPO46  ; Row 10 Column 1
SingleBlockLED        IS JPO47  ; Row 10 Column 2
; FOR JOGBRD REV??????, the LED outputs do not match Key inputs
; The PLC program should activate all three of these when
; it wants to turn on FeedHoldLED so that future hardware changes
; to put them in the same order as their corresponding inputs will work.
ToolCheckLED          IS JPO50  ; Row 10 Column 3
FeedHoldLED           IS JPO48  ; Row 10 Column 4
CycleStartLED         IS JPO49  ; Row 10 Column 5

;-------------------------------------------------------------------------------
;                     ---------SYSTEM VARIABLES--------
;
; For a complete list of System Variables and their functions, please see the
; MPU11 PLC manual.
;-------------------------------------------------------------------------------

; MPU11 based systems provide the PLC with the ability to read/write to a
; limited number of "System Variables".  While the use of System Variables
; greatly expands PLC functionality, it comes with additional reponsibility on
; the part of the PLC programmer.  Functionality that was once implemented as
; default behavior such as jogging, spindle speed, feedrate override, spindle
; gear ranges etc... is now implemented through System Variables in the PLC
; program.  It is now the sole responsibilty of the PLC program to provide a
; method to jog an axis, override the spindle speed or feedrates or even map a
; jog panel keypress to a specific function.  Pressing a jog key or Aux key
; won't DO anything unless the PLC assigns an action to the keypress.  All jog
; panel functions MUST be explicitly implemented in the PLC program.
;                                 ----IMPORTANT----
; Menu navigation in the CNC software requires that the escape key or Cycle
; Cancel key is used to back out of menus and screens.  You must use the PLC
; program to map a jog panel key and/or a keyboard key to the Cycle Cancel
; System Variable (SV_PLC_FUNCTION_1 has been declared as "DoCycleCancel")
; in order to use the control.  For example:
; The following lines map the escape key and Jog Panel Cycle Cancel key to
; produce a Cycle Cancel event:
```

```
; 1. Map escape keypress event to identifier to describe what key was pressed.
;    Kb_Escape    IS SV_PC_Keyboard_Key_1

; 2. Map MEM bit to identifier that describes what the keypress is used for.
;    KbCycleCancel_M  IS MEM401

; 3. Logic to "set" KbCycleCancel_M anytime the escape key is pressed.
;    if Kb_Escape THEN(KbCycleCancel_M)

; 4. Logic to cancel job if the escape key or cycle cancle key is pressed.
;    IF (CycleCancelKey || KbCycleCancel_M) && SV_PROGRAM_RUNNING
;      THEN (DoCycleCancel)

;   Some of the information made available to the PLC through System Variables:
; 1. Encoder positions: SV_MPU11_ABS_POS_1 - SV_MPU11_ABS_POS_7
; 2. Parameter values: SV_MACHINE_PARAMETER_1 - SV_MACHINE_PARAMETER_999
; 3. Spindle Speed command from PC: SV_PC_DAC_SPINDLE_SPEED
; 4. PC Keyboard Keypress: SV_PC_FUNCTION_1 - SV_PC_FUNCTION_127
; 5. ...

;   Some of the functionality controlled by the PLC through System Variables:
; 1. Axis jogging: SV_PLC_FUNCTION_12 - SV_PLC_FUNCTION_23
; 2. "Final" Spindle speed reported to PC: SV_PLC_SPINDLE_SPEED -provides nearly
;     unlimited gear ranges
; 3. Feedrate (through override knob): SV_PLC_FeedrateKnob_W
; 4. Custom debounce, invert/force inputs: SV_PLC_DEBOUNCE_1-SV_PLC_DEBOUNCE_64
; 5. ...

;-------------------------------------------------------------------------------
;        PLC Input manipulation - SV_PLC_DEBOUNCE_1 - SV_PLC_DEBOUNCE_64
; The System Variables in this section are used to modify the characteristics
; of PLC inputs 1-240.  Each input can be inverted, forced or assigned a custom
; debounce time.

;-----------------------------Debounce Times-----------------------------------
; SV_PLC_DEBOUNCE_61 - SV_PLC_DEBOUNCE_64 are used to define up to seven custom
; debounce times which can be selected for each input.

; The 32 bit integer System Variables SV_PLC_DEBOUNCE_61 - SV_PLC_DEBOUNCE_64,
; are broken up into 8, 16 bit words, only 7 of which are used. The first word,
; the 16 MSB of SV_PLC_DEBOUNCE_61 is unused.  Each 16 bit word can be used to
; store a debounce time of between 0-32767 (the MSB of each word is unused).
; Debounce times are in increments of 62.5 usecs which provides debounce times
; of up to ~2 secs.

;                          SV_PLC_DEBOUNCE_61
;                     Unused:Bits 0-15 (Selection 0)
;              15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

;                     Debounce Time Selection #1
;         MSB 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16

;                          SV_PLC_DEBOUNCE_62
```

```
;                     Debounce Time Selection #2
;              15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

;                     Debounce Time Selection #3
;         MSB 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16

;-----------------------Configuring Input Behavior--------------------------
; Each System Variable from SV_PLC_DEBOUNCE_1 - SV_PLC_DEBOUNCE_60 is a 32 bit
; integer word broken up into 4 bit words to control the behavior of 4 inputs.
; Inputs 1-4 are configured using SV_PLC_DEBOUNCE_1, inputs 5-8 are handled
; using SV_PLC_DEBOUNCE_2 and so on to SV_PLC_DEBOUNCE_60 which controls inputs
; 237-240

; As mentioned above, each 32 bit word defines the charactersitics for 4 inputs.
; SV_PLC_DEBOUNCE_1 defines the characteristics of INP1, INP2, INP3 & INP4 and
; so on through SV_PLC_DEBOUNCE_60 which handles INP237, INP238, INP239&INP240.
; The behavior of an input is set as follows:

; Five new operators have been introduced to simplify bit operations:
; BitSet, BitRst, BitTst, LShift & Rshift.  Below we will use bitset to
; invert an input.  This is convenient to use when a device is normally
; open and the logic is written for a normally closed device.  Inverting
; the input allows to reuse the existing logic rather than rewrite it.

; bitset and bitrst can not operate directly on SV_PLC_DEBOUNCE_# system
; variables, they can only operate on W32 variables.  In order to use bitset and
; bitrst to manipulate the debounce variables you'll have to perform all
; operations on a w32 first:

; Declare a W32:
; Inputs_9_12_W IS W1
; use BITSET or BITRST
; if 1 == 1 THEN bitset Inputs_9_12_W 14  ;invert INP10 (bit14)

; Set Debounce system variable = to W32 variable
; if 1 == 1 THEN SV_PLC_DEBOUNCE_3 = Inputs_9_12_W

;---------------------System Variable = SV_PLC_DEBOUNCE_1---------------------
;                     Inp4 = bits 31-24        Inp3 = bits 23-16
;                MSB 31 30 29 28 27 26 25 24| 23 22 21 20 19 18 17 16

;                       Inp2 = bits 15-8       Inp1 = bits 7-0
;                     15 14 13 12 11 10 9 8| 7 6 5 4 3 2 1 0LSB

;                               Each 8 bit word from above
;      MSB       7      6      5      4      3      2      1      0     LSB
;              Force  Invert Spare  Spare  Spare  Debounce Select (7)
;                                                        selects 1 of 7
;                                                        debounce times
;                                                       (zero is invalid)
;     Force (bit 7): Set this bit to force the input to a 1* (closed)
;     Invert(bit 6): Set this to invert an input
;     Spare(bit5-3): Not used
; Debounce(bit 0-2): Selects one of the 7 preset debounce times defined in
```

```
;                    SV_PLC_DEBOUNCE_61 - SV_PLC_DEBOUNCE_64
;
;  *If you wish to force an input to 0, set the both invert AND force bits
;   for the input.

;-------------------------------------------------------------------------------
; PLC Jog Panel input manipulation - The System Variables in this section are
; used to modify the characteristics of the Jog Panel keys.  The jog panel keys
; can be configured in the same manner as the PLC inputs and use debounce times
; as selected/set in SV_PLC_DEBOUNCE_61 - SV_PLC_DEBOUNCE_64.
;-------------------------------------------------------------------------------


;---------------------------------------------------------------
;  System variables:  Jog Panel Functions
;---------------------------------------------------------------
; Jog panel functions
;Invalid              IS SV_PLC_FUNCTION_0
DoCycleCancel         IS SV_PLC_FUNCTION_1
DoCycleStart          IS SV_PLC_FUNCTION_2
DoToolCheck           IS SV_PLC_FUNCTION_3
SelectSingleBlock     IS SV_PLC_FUNCTION_4
SelectX1JogInc        IS SV_PLC_FUNCTION_5
SelectX10JogInc       IS SV_PLC_FUNCTION_6
SelectX100JogInc      IS SV_PLC_FUNCTION_7
SelectUserJogInc      IS SV_PLC_FUNCTION_8
SelectIncContJog      IS SV_PLC_FUNCTION_9
SelectFastSlowJog     IS SV_PLC_FUNCTION_10
SelectMpgMode         IS SV_PLC_FUNCTION_11
DoAx1PlusJog          IS SV_PLC_FUNCTION_12
DoAx1MinusJog         IS SV_PLC_FUNCTION_13
DoAx2PlusJog          IS SV_PLC_FUNCTION_14
DoAx2MinusJog         IS SV_PLC_FUNCTION_15
DoAx3PlusJog          IS SV_PLC_FUNCTION_16
DoAx3MinusJog         IS SV_PLC_FUNCTION_17
DoAx4PlusJog          IS SV_PLC_FUNCTION_18
DoAx4MinusJog         IS SV_PLC_FUNCTION_19
DoAx5PlusJog          IS SV_PLC_FUNCTION_20
DoAx5MinusJog         IS SV_PLC_FUNCTION_21
DoAx6PlusJog          IS SV_PLC_FUNCTION_22
DoAx6MinusJog         IS SV_PLC_FUNCTION_23
DoAux1Key             IS SV_PLC_FUNCTION_24
DoAux2Key             IS SV_PLC_FUNCTION_25
DoAux3Key             IS SV_PLC_FUNCTION_26
DoAux4Key             IS SV_PLC_FUNCTION_27
DoAux5Key             IS SV_PLC_FUNCTION_28
DoAux6Key             IS SV_PLC_FUNCTION_29
DoAux7Key             IS SV_PLC_FUNCTION_30
DoAux8Key             IS SV_PLC_FUNCTION_31
DoAux9Key             IS SV_PLC_FUNCTION_32
DoAux10Key            IS SV_PLC_FUNCTION_33
SelectRapidOverride   IS SV_PLC_FUNCTION_34
SelectManAutoSpindle  IS SV_PLC_FUNCTION_35
DoSpindleStart        IS SV_PLC_FUNCTION_37
```

```
DoSpindleStop         IS SV_PLC_FUNCTION_38
DoAux11Key            IS SV_PLC_FUNCTION_39
DoAux12Key            IS SV_PLC_FUNCTION_40
;SelectCoolantMan     IS SV_PLC_FUNCTION_41 ;deprecated
;SelectCoolantAuto    IS SV_PLC_FUNCTION_42 ;deprecated
SelectCoolantFlood    IS SV_PLC_FUNCTION_43
SelectCoolantMist     IS SV_PLC_FUNCTION_44
DoFeedHold            IS SV_PLC_FUNCTION_45
SelectSpindleCCW      IS SV_PLC_FUNCTION_98
SelectSpindleCW       IS SV_PLC_FUNCTION_99
SelectCoolAutoMan     is SV_PLC_FUNCTION_104
DoIncreaseSpindleOr   IS SV_PLC_FUNCTION_106
DoDecreaseSpindleOr   IS SV_PLC_FUNCTION_107
SelectSpinOr100       IS SV_PLC_FUNCTION_108

;---------------------------------------------------------------
;  System variables:  Keyboard jogging functions
;---------------------------------------------------------------
;-------------------------------------------------------------------------------
; Keyboard Jogging Keys - The System Variables in this section inform the PLC
; that a PC keyboard keypress has occured.  Keep in mind that some key presses
; only come down while the keyboard jogging screen is enabled (alt-j) and that
; NONE of these keys not perform ANY default actions unless programmed to do so.
; The assignments provided below are for reference only.  For an example of
; mapping a keyboard key press to an MPU11 action, see the logic assigned to
; KbCycleStart_M or KbCycleCancel_M.
;
;Note:
; Keypresses are sent down as individual keys.  It is the responsibility of
; the PLC programmer to insure that a keypress is only acted on at the
; appropriate times.
; The "SV_PC_VIRTUAL_JOGPANEL_ACTIVE" system variable can be used to prevent
; a keypress form being acted on unless the keyboard jog screen is being
; displayed.  NOTE The above,29 character sys variable is mapped to
; KbJpActive_M (MEM80) to make it a "little" shorter......
;-------------------------------------------------------------------------------
Kb_a              IS SV_PC_KEYBOARD_KEY_60
Kb_b              IS SV_PC_KEYBOARD_KEY_79
Kb_c              IS SV_PC_KEYBOARD_KEY_77
Kb_d              IS SV_PC_KEYBOARD_KEY_62
Kb_e              IS SV_PC_KEYBOARD_KEY_41
Kb_f              IS SV_PC_KEYBOARD_KEY_63
Kb_g              IS SV_PC_KEYBOARD_KEY_64
Kb_h              IS SV_PC_KEYBOARD_KEY_65
Kb_i              IS SV_PC_KEYBOARD_KEY_46
Kb_j              IS SV_PC_KEYBOARD_KEY_66
Kb_k              IS SV_PC_KEYBOARD_KEY_67
Kb_l              IS SV_PC_KEYBOARD_KEY_68
Kb_m              IS SV_PC_KEYBOARD_KEY_81
Kb_n              IS SV_PC_KEYBOARD_KEY_80
Kb_o              IS SV_PC_KEYBOARD_KEY_47
Kb_p              IS SV_PC_KEYBOARD_KEY_48
Kb_q              IS SV_PC_KEYBOARD_KEY_39
Kb_r              IS SV_PC_KEYBOARD_KEY_42
```

```
Kb_s              IS SV_PC_KEYBOARD_KEY_61
Kb_t              IS SV_PC_KEYBOARD_KEY_43
Kb_u              IS SV_PC_KEYBOARD_KEY_45
Kb_v              IS SV_PC_KEYBOARD_KEY_78
Kb_w              IS SV_PC_KEYBOARD_KEY_40
Kb_x              IS SV_PC_KEYBOARD_KEY_76
Kb_y              IS SV_PC_KEYBOARD_KEY_44
Kb_z              IS SV_PC_KEYBOARD_KEY_75
Kb_spacebar       IS SV_PC_KEYBOARD_KEY_95
Kb_L_Shift        IS SV_PC_KEYBOARD_KEY_74
Kb_R_Shift        IS SV_PC_KEYBOARD_KEY_85
Kb_L_Alt          IS SV_PC_KEYBOARD_KEY_94
Kb_R_Alt          IS SV_PC_KEYBOARD_KEY_96
Kb_L_Ctrl         IS SV_PC_KEYBOARD_KEY_92
Kb_R_Ctrl         IS SV_PC_KEYBOARD_KEY_99
Kb_Ins            IS SV_PC_KEYBOARD_KEY_31
Kb_Home           IS SV_PC_KEYBOARD_KEY_32
Kb_End            IS SV_PC_KEYBOARD_KEY_53
Kb_PgDown         IS SV_PC_KEYBOARD_KEY_54
Kb_PgUp           IS SV_PC_KEYBOARD_KEY_33
Kb_Del            IS SV_PC_KEYBOARD_KEY_52
Kb_Back           IS SV_PC_KEYBOARD_KEY_30
Kb_Tab            IS SV_PC_KEYBOARD_KEY_38
Kb_Up             IS SV_PC_KEYBOARD_KEY_87
Kb_Down           IS SV_PC_KEYBOARD_KEY_101
Kb_Left           IS SV_PC_KEYBOARD_KEY_100
Kb_Right          IS SV_PC_KEYBOARD_KEY_102
Kb_Escape         IS SV_PC_KEYBOARD_KEY_1 ;Performs Cycle Cancel
Kb_F1             IS SV_PC_KEYBOARD_KEY_2
Kb_F2             IS SV_PC_KEYBOARD_KEY_3
Kb_F3             IS SV_PC_KEYBOARD_KEY_4
Kb_F4             IS SV_PC_KEYBOARD_KEY_5
Kb_F5             IS SV_PC_KEYBOARD_KEY_6
Kb_F6             IS SV_PC_KEYBOARD_KEY_7
Kb_F7             IS SV_PC_KEYBOARD_KEY_8
Kb_F8             IS SV_PC_KEYBOARD_KEY_9
Kb_F9             IS SV_PC_KEYBOARD_KEY_10
Kb_F10            IS SV_PC_KEYBOARD_KEY_11
Kb_F11            IS SV_PC_KEYBOARD_KEY_12
Kb_F12            IS SV_PC_KEYBOARD_KEY_13
Kb_Prt_Scrn       IS SV_PC_KEYBOARD_KEY_14
Kb_Scrl_Lck       IS SV_PC_KEYBOARD_KEY_15
Kb_Break          IS SV_PC_KEYBOARD_KEY_16
Kb_Num_Lock       IS SV_PC_KEYBOARD_KEY_34
Kb_1              IS SV_PC_KEYBOARD_KEY_18
Kb_2              IS SV_PC_KEYBOARD_KEY_19
Kb_3              IS SV_PC_KEYBOARD_KEY_20
Kb_4              IS SV_PC_KEYBOARD_KEY_21
Kb_5              IS SV_PC_KEYBOARD_KEY_22
Kb_6              IS SV_PC_KEYBOARD_KEY_23
Kb_7              IS SV_PC_KEYBOARD_KEY_24
Kb_8              IS SV_PC_KEYBOARD_KEY_25
Kb_9              IS SV_PC_KEYBOARD_KEY_26
Kb_0              IS SV_PC_KEYBOARD_KEY_27
```

```
Kb_10_Key_Div     IS SV_PC_KEYBOARD_KEY_35
Kb_10_Key_Mlt     IS SV_PC_KEYBOARD_KEY_36
Kb_10_Key_Sub     IS SV_PC_KEYBOARD_KEY_37
Kb_10_Key_0       IS SV_PC_KEYBOARD_KEY_103
Kb_10_Key_1       IS SV_PC_KEYBOARD_KEY_88
Kb_10_Key_2       IS SV_PC_KEYBOARD_KEY_89
Kb_10_Key_3       IS SV_PC_KEYBOARD_KEY_90
Kb_10_Key_4       IS SV_PC_KEYBOARD_KEY_71
Kb_10_Key_5       IS SV_PC_KEYBOARD_KEY_72
Kb_10_Key_6       IS SV_PC_KEYBOARD_KEY_73
Kb_10_Key_7       IS SV_PC_KEYBOARD_KEY_55
Kb_10_Key_8       IS SV_PC_KEYBOARD_KEY_56
Kb_10_Key_9       IS SV_PC_KEYBOARD_KEY_57
Kb_10_Key_Dec_Pt  IS SV_PC_KEYBOARD_KEY_104
Kb_10_Key_Plus    IS SV_PC_KEYBOARD_KEY_58
Kb_Num_Enter      IS SV_PC_KEYBOARD_KEY_91
Kb_L_Sq_Bracket   IS SV_PC_KEYBOARD_KEY_49
Kb_R_Sq_Bracket   IS SV_PC_KEYBOARD_KEY_50
Kb_Hyphen         IS SV_PC_KEYBOARD_KEY_28
Kb_Equals         IS SV_PC_KEYBOARD_KEY_29
Kb_Comma          IS SV_PC_KEYBOARD_KEY_82
Kb_Period         IS SV_PC_KEYBOARD_KEY_83
Kb_Slash          IS SV_PC_KEYBOARD_KEY_84
Kb_Backslash      IS SV_PC_KEYBOARD_KEY_86

;-------------------------------------------------------------------
; M functions - The System Variables in this section inform the
; PLC that an M function has been requested.
;-------------------------------------------------------------------
M3       IS SV_M94_M95_1 ;(Spindle CW)
M4       IS SV_M94_M95_2 ;(Spindle CCW)
M8       IS SV_M94_M95_3 ;(Flood On)
M10      IS SV_M94_M95_4 ; Clamp
M7       IS SV_M94_M95_5 ;(Mist)
;        IS SV_M94_M95_6 ;
;        IS SV_M94_M95_7 ;
;        IS SV_M94_M95_8 ;
;        IS SV_M94_M95_9 ;
;        IS SV_M94_M95_10;
;        IS SV_M94_M95_11;
;        IS SV_M94_M95_12;
;        IS SV_M94_M95_13;
;        IS SV_M94_M95_14;
;        IS SV_M94_M95_15;
;        IS SV_M94_M95_16;

;-------------------------------------------------------------------
;                     Word Definitions (int32)
;-------------------------------------------------------------------
ErrorCode_W             IS W1
TwelveBitSpeed_W        IS W2
LubeAccumTime_W         IS W3
KbOverride_W            IS W4
FeedrateKnob_W          IS W5
```

```
CycloneStatus_W         IS W6
FinalFeedOverride_W     IS W7
PLC_Fault_W             IS W8
PLCFaultAddr_W          IS W9
Last_FeedrateKnob_W     IS W10
AsyncMsg_W              IS W11
P148Value_W             IS W12
Lube_W                  IS W21
LubeM_W                 IS W22
LubeS_W                 IS W23
Inputs_9_12_W           IS W28
P170Value_W             IS W30
P178Value_W             IS W31


;---------------------------------------
;           Word Definitions cont. (f32)
;---------------------------------------
SpinRangeAdjust         IS FW1
RPMPerBit_FW            IS FW2
CfgMinSpeed_FW          IS FW3
CfgMaxSpeed_FW          IS FW4
TwelveBitSpeed_FW       IS FW5
SpinSpeedCommand_FW     IS FW6


;-----------------------------------
;            One Shot Definitions
;-----------------------------------
IncrContPD              IS PD1
SlowFastPD              IS PD2
MpgPD                   IS PD3
SingleBlockPD           IS PD4
FeedHoldPD              IS PD5
SpinAutoManPD           IS PD6
SpindlePlusPD           IS PD7
SpinOverMinusPD         IS PD8
SpinOver100PD           IS PD9
SpinStartPD             IS PD10
SpinStopPD              IS PD11
SpinCWPD                IS PD12
SpinCCWPD               IS PD13
F9PD                    IS PD14
x1JogPD                 IS PD15
x10JogPD                IS PD16
x100JogPD               IS PD17
Aux11KeyPD              IS PD18
RapidOverPD             IS PD19
CoolantAutoManualPD     IS PD21
CoolantFloodPD          IS PD22
CoolantMistPD           IS PD23
ToolCheckPD             IS PD24
JogProbeFaultPD         IS PD25
RigidTapPD              IS PD26
```

```
PCSpindleStartStopPD    IS PD30
PCSpindleManualPD       IS PD31
PCSpindleCWPD           IS PD32
PCSpindleCCWPD          IS PD33
StopRunningPD           IS PD35
SoftwareReadyPD         IS PD36
Aux1PD                  IS PD41
Aux2PD                  IS PD42
Aux3PD                  IS PD43
Aux4PD                  IS PD44
Aux5PD                  IS PD45
Aux6PD                  IS PD46
Aux7PD                  IS PD47
Aux8PD                  IS PD48
Aux9PD                  IS PD49

;-------------------------------------------------------------
;                    Timer Definitions
;-------------------------------------------------------------
; 1000 = 1 second for all timers.
;
MsgClear_T               IS T1
SleepTimer               IS T2
CycloneStatus_T          IS T3
Initialize_T             IS T4
LubeM_T                  IS T13
LubeS_T                  IS T14

;-------------------------------------------------------------
;                      Stage Definitions
;-------------------------------------------------------------
WatchDogStage           IS STG1
InitialStage            IS STG2
JogPanelStage           IS STG3
MainStage               IS STG4
AxesEnableStage         IS STG5
SpindleStage            IS STG6
MPGStage                IS STG7
CheckCycloneStatusStage IS STG8
LoadParametersStage     IS STG9
KeyboardEventsStage     IS STG10
LubeUsePumpTimersStage  IS STG13
LubeUsePLCTimersStage   IS STG14

SetErrorStage           IS STG50
BadErrorStage           IS STG51

;#############################################################################
;                         Program Start
;#############################################################################

;=============================================================================
                          WatchDogStage
;=============================================================================
```

```
; Handle PLC executor faults. The only way to reset a PLC executor fault
; is to reboot the MPU11.
IF SV_PLC_FAULT_STATUS != 0
  THEN PLC_Fault_W    = SV_PLC_FAULT_STATUS,
       PLCFaultAddr_W = SV_PLC_FAULT_ADDRESS,
       ErrorCode_W    = PLC_EXECUTOR_FLT_MSG, MSG ErrorCode_W,
       SET PLCExecutorFault_M, RST SetErrorStage, SET SV_STOP

; Handle software exit.
IF !SV_PC_SOFTWARE_READY && (SV_PLC_FAULT_STATUS == 0)
  THEN SET SoftwareNotReady_M,
       SET SV_STOP,
       ErrorCode_W = SOFTWARE_EXIT_MSG

if SV_PC_SOFTWARE_READY && (SV_PLC_FAULT_STATUS == 0) THEN (SoftwareReadyPD)
IF SoftwareReadyPD && !SoftwareNotReady_M || !True THEN SET InitialStage

IF SoftwareReadyPD && SoftwareNotReady_M THEN RST SoftwareNotReady_M

;=============================================================================
                          InitialStage
;=============================================================================
IF 1==1 THEN SET true,
             SET OnAtPowerUp_M,
             SET AxesEnableStage,
             SET MainStage,
             SET JogPanelStage,
             SET LoadParametersStage,
             SET MPGStage,
             SET PLCBus_Oe_M,
             RST DriveComFltIn_M,
             RST DriveComFltOut_M,
             RST PLCFault_M,
             CycloneStatus_T = 300,
             ErrorCode_W = MSG_CLEARED_MSG,
             RST BadErrorStage,
             SET SetErrorStage,
             Initialize_T = 1000, SET Initialize_T,
             RST InitialStage


;=============================================================================
                       LoadParametersStage
;=============================================================================

; There are two methods of control for the lube pump and they are set by CNC11
; Machine Parameter 179, where the value is between 0 - 65535 and is formatted
; as MMMSS where MMM is a time in minutes and SS is a time in seconds.
;
; METHOD 1 (SS == 0) For lube pumps with internal timers.
; METHOD 2 (SS != 0) For lube pumps with no timers (controlled soley by PLC).
;
; Load lube pump times from P179 and convert to milliseconds.
```

```
IF true THEN Lube_W = SV_MACHINE_PARAMETER_179,
             LubeM_W = (Lube_W / 100) * 60000,
             LubeS_W = (Lube_W % 100) * 1000

; Set the appropriate stage according to method of control
IF LubeS_W == 0 THEN SET LubeUsePumpTimersStage, RST LubeUsePLCTimersStage
IF LubeS_W != 0 THEN SET LubeUsePLCTimersStage, RST LubeUsePumpTimersStage

IF True THEN P148Value_W = SV_MACHINE_PARAMETER_148,  ; Misc Jogging Options
             P170Value_W = SV_MACHINE_PARAMETER_170,  ; Enable Keyboard Jogging
             P178Value_W = SV_MACHINE_PARAMETER_178   ; PLC IO NO / NC Settings

IF True THEN BITTST P148Value_W 1 DisableKbInput_M,
             BITTST P170Value_W 0 AllowKbInput_M
IF DisableKbInput_M THEN RST AllowKbInput_M
If true THEN BitTst P170Value_W 1 JogOverOnly_M
If true THEN BitTst P170Value_W 2 KbOverOnly_M
if JogOverOnly_M && KbOverOnly_M THEN rst KbOverOnly_M

If true THEN BitTst P178Value_W 0 InvLubeOk_M
If true THEN BitTst P178Value_W 1 InvSpinInverterOk_M

;===============================================================================
                          LubeUsePumpTimersStage
;===============================================================================

; METHOD 1 (SS == 0) For lube pumps with internal timers.
;
; When using this method, P179 should be set such that MMM is a
; value that is greater than the cycle time set on the internal timers and
; SS should be set to zero. How much greater MMM needs to be depends on the
; accuracy of the lube pump timers, but it is better to be on the long side
; to ensure proper operation.
;
; Example 1. The internal lube cycle interval is set to 60 minutes.
;            Set P179 = 7500. In this example, as long as the accuracy
;            of the lube timer interval causes the lube to turn on
;            within 75 minutes, it will work. Note that the amount of time
;            that lube is output is usually set with another timer control
;            on the lube pump and it does not factor into the setting of P179.
;
; It should be noted that lube pumps with internal timers may differ on how
; they operate.
;
; (a) For pumps that lube immediately when power is applied and then start timing
; until the next cycle, it is possible to run out of lube quickly on short job
; runs if, after the program has been run, lube power is removed.
;
; (b) For pumps that do not lube until it has been turned on for the interval time,
; it is possible that lube never gets applied if, after the short program has been run,
; lube power is removed.
;
; A short program or job run is defined as a job that finishes before
; the interval setting (60 minutes in the above example).
```

```
;
; For the above mentioned reasons, we want the power to be applied for at least
; the amount of time set by the inteval timer, noting that if the user decides
; to engage the E-stop to remove power after short jobs, then they risk the
; above mentioned problems accoding to the type of pump.
;
; On the start of SV_PROGRAM_RUNNING, the lube pump turns on.
; The lube pump is turned off when a program has NOT been
; running continuously for MMM minutes or E-stop is engaged.
; The reason the lube pump is turned off after a program has NOT been
; running for MMM minutes is to prevent lubing when the user leaves for the
; weekend, leaving the machine on and E-stop disengaged.

IF (SV_PROGRAM_RUNNING || SV_MDI_MODE) THEN SET Lube, RST LubeM_T
IF !(SV_PROGRAM_RUNNING || SV_MDI_MODE) THEN LubeM_T = LubeM_W, SET LubeM_T
IF LubeM_T || !EStopOk THEN RST Lube

;=============================================================================
                         LubeUsePLCTimersStage
;=============================================================================
;
; METHOD 2 (SS != 0) For lube pumps that do not have internal timers.
;
;  When using this method P179 should be set so the lube turns on
;  every MMM minutes for SS seconds.
;
;  Example 1.
;    To set the lube pump power to come on for 5 seconds
;    every 10 minutes, set P179 =  1005.
;                                  MMMSS
;  Example 2.
;    To set the lube pump power to come on for 30 seconds
;    every 2 hours, set P179 = 12030
;                                MMMSS
;
; This method will accumulate time while a program is running until
; it reaches MMM minutes, at which time it will apply power
; for SS seconds (unless E-stop is engaged) and then start over. It is
; possible with frequent use of E-stop that a lube cycle is cut short.
;

IF (SV_PROGRAM_RUNNING || SV_MDI_MODE)  THEN LubeM_T = LubeM_W, SET LubeM_T
IF !(SV_PROGRAM_RUNNING || SV_MDI_MODE) THEN (StopRunningPD)
IF StopRunningPD THEN LubeAccumTime_W = LubeAccumTime_W + LubeM_T, RST LubeM_T
IF LubeM_T || (LubeAccumTime_W + LubeM_T > LubeM_W)
  THEN SET Lube, LubeS_T = LubeS_W, SET LubeS_T, RST LubeM_T, LubeAccumTime_W = 0
IF LubeS_T || !EStopOk THEN RST Lube, RST LubeS_T

;=============================================================================
                         KeyboardEventsStage
;=============================================================================
; This stage handles functions that are required for menu navigation
; by CNC11, require multiple keypresses and/or need to be interlocked
; with SV_PC_VIRTUAL_JOGPANEL_ACTIVE and/or AllowKbInput_M.  Regarding
```

```
; "AllowKbInput_M": This PLC program reads a bit from a system parameter,
; in this case bit 0 of SV_MACHINE_PARAMETER_170, and sets "AllowKbInput_M"
; if the bit is a "0".  If the operator wishes to allow keyboard input
; to trigger PLC events, they must set parameter 170 to a "1"
; (or any odd number for that matter).  It should be mentioned that
; the programmer will not want to interlock all keyboard keys with
; SV_PC_VIRTUAL_JOGPANEL_ACTIVE and/or AllowKbInput_M.  For example:
; The "escape" key must be echoed by the PLC to CNC11 to aid in menu
; navigation.  NOTE: For backward comaptibility with CNC10, setting bit 1
; of SV_MACHINE_PARAMETER_148 OR clearing bit 0 of SV_MACHINE_PARAMETER_170
; will disable keyboard jogging.

;-------------------------Not interlocked-----------------------
; The code for cycle cancel has been moved to the main stage.
; It is commented out below but remains for reference
;Cycle Cancel
;if Kb_Escape THEN (KbCycleCancel_M)

;Rapidoverride: Ctrl-r
if Kb_r && (Kb_L_Ctrl || Kb_R_Ctrl) THEN (KbTogRapidOver_M)

;----------------Interlocked with AllowKbInput_M-------------------
;KbCycle Start: alt-s
if  Kb_s && (Kb_R_Alt || Kb_L_Alt) && AllowKbInput_M then (KbCycleStart_M)

;KbToolCheck_M: Ctrl-t
if Kb_t && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbToolCheck_M)

;KbTogSingleBlock_M: ctrl-b
if Kb_b && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbTogSingleBlock_M)

;KbTogSpinAutoMan_M: ctrl-a
if Kb_a && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbTogSpinAutoMan_M)

;KbSpinCW_M: ctrl-c
IF Kb_c && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN SET KbSpinCW_M,
    RST KbSpinCCW_M

;KbSpinCCW_M: ctrl-w
IF Kb_w && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN SET KbSpinCCW_M,
    RST KbSpinCW_M

;KbSpinStart_M:  ctrl-s
if Kb_s && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbSpinStart_M)

;KbSpindle stop: Ctrl-q
if Kb_q && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbSpinStop_M)

;KbIncSpinOver_M: ctrl (">")
if Kb_Period && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M
  then (KbIncSpinOver_M)

;KbDecSpinOver_M: ctrl ("<")
if Kb_Comma && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M
```

```
  then (KbDecSpinOver_M)

;KbSpinOver100_M: ctrl + /
IF Kb_Slash && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M
  THEN (KbSpinOver100_M)

;KbTogCoolAutoMan_M: Ctrl-m
if Kb_m && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbTogCoolAutoMan_M)

;KbFloodOnOff_M: Ctrl-n
if Kb_n && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbFloodOnOff_M)

;KbMistOnOff_M: Ctrl-k
if Kb_k && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbMistOnOff_M)

;KbTogIncContJog_M: "ctrl" +  "i"
if Kb_i &&  (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbTogIncContJog_M)

;KbTogFastSlowJog_M: "ctrl" + "f"
IF Kb_f &&  (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M
  THEN (KbTogFastSlowJog_M)

;KbAux1Key_M: "ctrl" + "F1"
if Kb_F1 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M then (KbAux1Key_M)

;KbAux2Key_M: "ctrl" + "F2"
if Kb_F2 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux2Key_M)

;KbAux3Key_M: "ctrl" + "F3"
if Kb_F3 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux3Key_M)

;KbAux4Key_M: "ctrl" + "F4"
if Kb_F4 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux4Key_M)

;KbAux5Key_M: "ctrl" + "F5"
if Kb_F5 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux5Key_M)

;KbAux6Key_M: "ctrl" + "F6"
if Kb_F6 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux6Key_M)

;KbAux7Key_M: "ctrl" + "F7"
if Kb_F7 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux7Key_M)

;KbAux8Key_M: "ctrl" + "F8"
if Kb_F8 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux8Key_M)

;KbAux9Key_M: "ctrl" + "F9"
if Kb_F9 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux9Key_M)

;KbAux10Key_M: "ctrl" + "F10"
if Kb_F10 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux10Key_M)

;KbAux11Key_M: "ctrl" + "F11"
if Kb_F11 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux11Key_M)
```

```
;KbAux12Key_M: "ctrl" + "F12"
if Kb_F12 && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M THEN (KbAux12Key_M)

;KbIncFeedOver_M: "ctrl" + "keyboard +" (actually "=")
IF Kb_Equals && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M
  THEN (KbIncFeedOver_M)

;KbDecFeedOver_M: "ctrl" + "keyboard -"
IF Kb_Hyphen && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M
  THEN (KbDecFeedOver_M)

;KbFeedOver100_M: "ctrl" + "keyboard \"
IF Kb_Backslash && (Kb_L_Ctrl || Kb_R_Ctrl) && AllowKbInput_M
  THEN (KbFeedOver100_M)


;--------Interlocked with AllowKbInput_M && KbJpActive_M-----------

;KbIncreaseJogInc_M: "insert"
if Kb_Ins && AllowKbInput_M && KbJpActive_M
  then (KbIncreaseJogInc_M)
if KbIncreaseJogInc_M && x1JogLED && !X1_M && !X10_M && !X100_M
  then set X10_M
if KbIncreaseJogInc_M && x10JogLED && !X1_M && !X10_M && !X100_M
  then set X100_M

;KbDecreaseJogInc_M: "delete"
if Kb_Del && AllowKbInput_M && KbJpActive_M
  then (KbDecreaseJogInc_M)
if KbDecreaseJogInc_M && x10JogLED && !X1_M && !X10_M && !X100_M
  then set X1_M
if KbDecreaseJogInc_M && x100JogLED && !X1_M && !X10_M && !X100_M
  then set X10_M


;KbJogAx1Plus_M: Right arrow
if Kb_Left && AllowKbInput_M && KbJpActive_M THEN (KbJogAx1Plus_M)

;KbJogAx1Minus_M: Right arrow
if Kb_Right && AllowKbInput_M && KbJpActive_M THEN (KbJogAx1Minus_M)

;KbJogAx2Plus_M: Up arrow
if Kb_Up && AllowKbInput_M && KbJpActive_M THEN (KbJogAx2Plus_M)

;KbJogAx1Minus_M: Down arrow
if Kb_Down && AllowKbInput_M && KbJpActive_M THEN (KbJogAx2Minus_M)

;KbJogAx3Plus_M: Page up
if Kb_PgUp && AllowKbInput_M && KbJpActive_M THEN (KbJogAx3Plus_M)

;KbJogAx3Minus_M: Page Down
if Kb_PgDown && AllowKbInput_M && KbJpActive_M THEN (KbJogAx3Minus_M)
```

```
;KbAx4PlusJog: "home"
if Kb_Home && AllowKbInput_M && KbJpActive_M
  then (KbJogAx4Plus_M)

;KbAx4MinusJog: "end"
if Kb_End && AllowKbInput_M && KbJpActive_M
  then (KbJogAx4Minus_M)

IF True THEN RST KeyboardEventsStage

;=============================================================================
                            MPGStage
;=============================================================================
;                         MPG Functions
;       Turn on/off Jog Panel MPG LED & on the MPG
IF MPGKey then (MpgPD)
IF MpgPD && MPGLED then set MPGManOffFlag_M
IF !SV_MPG_1_ENABLED || (MpgPD && !MPGLED) then RST MPGManOffFlag_M

IF (MpgPD && !MPGLED) || (SV_MPG_1_ENABLED && !MPGManOffFlag_M) &&
  !SV_PROGRAM_RUNNING THEN SET MPG_LED_OUT, SET MPGLED

IF (!SV_MPG_1_ENABLED || (MpgPD && MPGLED))
  || SV_PROGRAM_RUNNING THEN RST MPG_LED_OUT, RST MPGLED

;x1, x10, x100 functions
;--X1
IF x1JogKey THEN (x1JogPD)
IF x1JogPD || OnAtPowerUp_M || X1_M || (MPG_Inc_X_1 && MPGLED)
  THEN SET x1JogLED, RST x10JogLED, RST x100JogLED

;--X10
IF x10JogKey THEN (x10JogPD)
IF x10JogPD || X10_M || (MPG_Inc_X_10 && MPGLED)
  THEN RST x1JogLED, SET x10JogLED, RST x100JogLED

;--X100
IF x100JogKey THEN (x100JogPD)
IF x100JogPD || X100_M || (MPG_Inc_X_100 && MPGLED)
  THEN RST x1JogLED, RST x10JogLED, SET x100JogLED

if !KbIncreaseJogInc_M && !KbDecreaseJogInc_M then rst X1_M, rst X10_M,
  rst X100_M

;--MPG 1 Enable
IF MPG_AXIS_1 || MPG_AXIS_2 || MPG_AXIS_3 || MPG_AXIS_4 ||
  MPG_AXIS_5 || MPG_AXIS_6 || MPG_AXIS_7 || MPG_AXIS_8
  THEN (SV_MPG_1_ENABLED)

;                     Select axis to move
IF MPG_AXIS_1 THEN SV_MPG_1_AXIS_SELECT = 1
IF MPG_AXIS_2 THEN SV_MPG_1_AXIS_SELECT = 2
IF MPG_AXIS_3 THEN SV_MPG_1_AXIS_SELECT = 3
IF MPG_AXIS_4 THEN SV_MPG_1_AXIS_SELECT = 4
```

```
IF MPG_AXIS_5 THEN SV_MPG_1_AXIS_SELECT = 5

;                   Select MPG 1 Multiplier
IF (MPG_Inc_X_100) THEN SV_MPG_1_MULTIPLIER = 100
IF (MPG_Inc_X_10) THEN  SV_MPG_1_MULTIPLIER = 10
IF (MPG_Inc_X_1) THEN   SV_MPG_1_MULTIPLIER = 1

;            Disable "Windup" mode IF x100 selected
IF (!MPG_Inc_X_100) THEN (SV_MPG_1_WINDUP_MODE)


;=============================================================================
                           JogPanelStage
;=============================================================================
;--Select Incremental or Continuous Jog Mode
IF IncrContKey || KbTogIncContJog_M THEN (IncrContPD)
IF (IncrContPD && !IncrContLED) || OnAtPowerUp_M THEN SET IncrContLED
IF (IncrContPD && IncrContLED) THEN RST IncrContLED

;--Select Fast or Slow Jog Mode
IF FastSlowKey || KbTogFastSlowJog_M THEN (SlowFastPD)
IF (SlowFastPD && !FastSlowLED) || OnAtPowerUp_M
  THEN SET FastSlowLED
IF (SlowFastPD && FastSlowLED) THEN RST FastSlowLED

;--Single Block Mode

IF SingleBlockKey || KbTogSingleBlock_M THEN (SingleBlockPD)
IF SingleBlockPD && !SingleBlockLED && !SV_PROGRAM_RUNNING
  THEN SET SingleBlockLED
IF SingleBlockPD && SingleBlockLED THEN RST SingleBlockLED
IF SingleBlockLED THEN (SelectSingleBlock)

;--Toolcheck

IF (ToolCheckKey || KbToolCheck_M) && EstopOk THEN (ToolCheckPD)
IF ToolCheckPD THEN (DoToolCheck)

;--Feed Hold Mode
IF (FeedHoldKey || KbFeedHold_M) && SV_PROGRAM_RUNNING THEN (FeedHoldPD)
IF FeedHoldPD && !FeedHoldLED THEN SET FeedHoldLED
IF FeedHoldPD && FeedHoldLED && !SV_PROGRAM_RUNNING && !SV_MDI_MODE
  THEN RST FeedHoldLED
IF FeedHoldLED && (DoCycleStart || DoCycleCancel || ToolCheckPD)
  THEN RST FeedHoldLED
; (FeedHoldLED will be used later to signal MPU11 to do Feed Hold)

;--Feedrate Override Section
;-----------------------------------------------------------------------
; Feedrate override works as follows:
;
; 1. The PLC reads the 8 bit value of the FeedrateKnob_W directly (0-255)
; 2. The PLC scales this value to a 0-200 value (0-200%) by dividing by
;    the knob value by 127.5 and then multiplying the result by 100
```

```
; 3. If keyboard joggin is not disabled (it is enabled by default), the PLC
;    determines whether the operator is using the keyboard override or
;    the FeedrateKnob_W to override the feedrate by watching which was changed
;    most recently.  The most recently changed value is saved as
;    "FinalFeedOverride_W"
; 4. Parameter 39 in (From the "params" screen in CNC11 software) stores
;    a value which allows which the PLC program can use to limit the amount
;    of override applied to the programmed feedrate. This value is specified
;    as a percentage.
; 5. The PLC limits the override percentage by reading parameter 39 and, if
;    the feedrate override percentage as read from the knob is greater than
;    parameter 39, it sets the FinalFeedOverride_W value to the value of
;    parameter 39.
; 6. Once the override percentage has been determined and limited (if needed)
;    The PLC send this value up to the CNC11 software by setting
;    SV_PLC_FeedrateKnob_W = FinalFeedOverride_W
; 7. CNC11 reads SV_PLC_FEEDRATE_KNOB, factors in it's on own override based
;    on parameter 78 (see operators manual for more info on parm 78) and then
;    returns an override value to the PLC in the system variable
;    SV_PC_FEEDRATE_PERCENTAGE
; 8. The PLC reads SV_PC_FEEDRATE_PERCENTAGE and (typically) echoes the system
;    variable to SV_PLC_FEEDRATE_OVERRIDE which the MPU11 uses as the final
;    determination of the feedrate override percentage.
;-------------------------------------------------------------------------
; 1. The PLC reads the 8 bit value of the FeedrateKnob_W directly (0-255)
; NOTE: BTW = Bit To Word
; BTW reads the specified number of bits (if none is specified it defaults to 8)
; starting from a bit location and writes them to a word with the starting bit
; location being written to the LSB of the word used.  Below, BTW reads the bit
; values from JpFeedOrKnobBit0 to JpFeedOrKnobBit7 and writes them into to the
; word "FeedrateKnob_W" which sets FeedrateKnob_W to a value of 0-255
;-------------------------------------------------------------------------
IF true THEN FeedrateKnob_W = 0
if true THEN BTW FeedrateKnob_W JpFeedOrKnobBit0 8

;-------------------------------------------------------------------------
; 2. Scale this value to a 0-200 value (0-200%)
;-------------------------------------------------------------------------
IF true THEN FeedrateKnob_W = (FeedrateKnob_W/127.5)*100

;-------------------------------------------------------------------------
; 3.  Determine whether to use FeedrateKnob_W or KbOverride_W
;-------------------------------------------------------------------------
; This section determines when to use the feedrate override value sent down
; by the jogpanel (FeedrateKnob_W) or the feedrate override as determined
; by the PLC monitoring the keyboard override keys (KbOverride_W).

;-----------------------------------------------------------------------------
;     At powerup, default feedrate override is jog panel (FeedrateKnob_W)
;       To use both keyboard or jogpanel overrides set p170 to 0 (default)
;       To use jogpanel override only set p170 to 2
;       To use keyboard only set p170 to 4
;-----------------------------------------------------------------------------
IF OnAtPowerUp_M && KbOverOnly_M || KbFeedOver100_M THEN KbOverride_W = 100
```

```
IF OnAtPowerUp_M && !KbOverOnly_M THEN set UsingFeedrateKnob_M,
                                        KbOverride_W = FeedrateKnob_W,
                                        Last_FeedrateKnob_W = FeedrateKnob_W

;----------------Calculate keyboard feedrate override-------------------------
; SleepTimer is used to limit the KbOverride_W update rate to 20% per sec
;-------------------------------------------------------------------------------
if AllowKbInput_M && KbIncFeedOver_M && !WaitingForSleepTimer_M
  THEN KbOverride_W = KbOverride_W + 1,
       RST UsingFeedrateKnob_M,
       SET WaitingForSleepTimer_M,
       SleepTimer = 50, SET SleepTimer

if AllowKbInput_M && KbDecFeedOver_M && !WaitingForSleepTimer_M
  THEN KbOverride_W = KbOverride_W - 1, rst UsingFeedrateKnob_M,
  set WaitingForSleepTimer_M, SleepTimer = 50, set SleepTimer

if SleepTimer THEN rst WaitingForSleepTimer_M, rst SleepTimer

;------------Switch to FeedrateKnob_W if it changes more than 3%--------------
; Once it has changed by more than 3%, it will update as normal (1% increments)
; until it sees another KbOverride_W command at which point it will take
; another 3% change to re-activate the FeedrateKnob_W

if (abs(Last_FeedrateKnob_W - FeedrateKnob_W) >= 3) || UsingFeedrateKnob_M
  THEN FinalFeedOverride_W = FeedrateKnob_W, KbOverride_W = FeedrateKnob_W,
  Last_FeedrateKnob_W = FeedrateKnob_W, set UsingFeedrateKnob_M

;Limit keyboard override to parm 39.  Allowing the FeedrateKnob_W to go past
;parm 39, but keeping the KbOverride_W limited keeps the "dead space"
;down and allows the PLC to respond to changes in the FeedrateKnob_W even if
;above 120. Overall override is still limited later but this gives better
;response in changing between KbOverride_W & the FeedrateKnob_W
if KbOverride_W > SV_MACHINE_PARAMETER_39
  THEN KbOverride_W = SV_MACHINE_PARAMETER_39

if !UsingFeedrateKnob_M && !JogOverOnly_M
  THEN FinalFeedOverride_W = KbOverride_W

;-------------------------------------------------------------------------
; 4 & 5. Limit override percentage to value set in Parameter 39
;-------------------------------------------------------------------------
;------------------Limit final override percentage to parm 39-------------------
if FinalFeedOverride_W > SV_MACHINE_PARAMETER_39
  THEN FinalFeedOverride_W = SV_MACHINE_PARAMETER_39

if FinalFeedOverride_W < 0 THEN FinalFeedOverride_W = 0

;-------------------------------------------
;  Override Controls
;  It is important that the plc program only writes to SV_PLC_Feedrate_Knob once per pass
;-------------------------------------------
; Override control bit for the feedrate override
; 1 == feedrate override knob will effect feedrate
```

```
; 0 == override knob has NO effect on feedrate
IF  !SV_PC_OVERRIDE_CONTROL_FEEDRATE_OVERRIDE THEN FinalFeedOverride_W = 100

;---------------------------------------------------------------------
; 6. Send override percentage to CNC11
;---------------------------------------------------------------------
;---------------Send override to PC for modification if needed--------------
if true THEN SV_PLC_Feedrate_Knob = FinalFeedOverride_W

;---------------------------------------------------------------------
; 7. Copy the feedrate override sent from the PC to the MPU11.
;---------------------------------------------------------------------
;-------------------------------------------------------------------------
; Normally a number from 0.0-2.0, no limitations although V will not exceed
; Vmax. A negative number in here would be extremely bad.
;-------------------------------------------------------------------------
IF true THEN SV_PLC_FEEDRATE_OVERRIDE = SV_PC_FEEDRATE_PERCENTAGE/100.0

;--MPU11 Jog Panel Key Functions
IF KB_F9 then (F9PD)
IF KbTogRapidOver_M || (F9PD && (SV_PROGRAM_RUNNING || SV_MDI_MODE))
  THEN (RapidOverPD)
IF RapidOverPD^ SelectRapidOverride THEN (SelectRapidOverride)
IF OnAtPowerUp_M THEN SET SelectRapidOverride
IF (CycleCancelKey || KbCycleCancel_M) && (SV_PROGRAM_RUNNING || SV_MDI_MODE)
  THEN (DoCycleCancel)
IF (CycleStartKey  || KbCycleStart_M) THEN (DoCycleStart)

IF (Ax1PlusJogKey  || KbJogAx1Plus_M)  && !Ax1PlusJogDisabled_M &&
   !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx1PlusJog)
IF (Ax1MinusJogKey || KbJogAx1Minus_M) && !Ax1MinusJogDisabled_M &&
  !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx1MinusJog)
IF (Ax2PlusJogKey  || KbJogAx2Plus_M)  && !Ax2PlusJogDisabled_M  &&
  !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx2PlusJog)
IF (Ax2MinusJogKey || KbJogAx2Minus_M) && !Ax2MinusJogDisabled_M &&
  !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx2MinusJog)

IF (Ax3PlusJogKey  || KbJogAx3Plus_M) &&
   !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx3PlusJog)
IF (Ax3MinusJogKey || KbJogAx3Minus_M) &&
  !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx3MinusJog)
IF (Ax4PlusJogKey  || KbJogAx4Plus_M) &&
  !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx4PlusJog)
IF (Ax4MinusJogKey || KbJogAx4Minus_M) &&
  !(IncrContLED && FinalFeedOverride_W == 0) THEN (DoAx4MinusJog)

IF (Aux1Key        || KbAux1Key_M) THEN (DoAux1Key)
IF (Aux2Key        || KbAux2Key_M) THEN (DoAux2Key)
IF (Aux3Key        || KbAux3Key_M) THEN (DoAux3Key)
IF (Aux4Key        || KbAux4Key_M) THEN (DoAux4Key)
IF (Aux5Key        || KbAux5Key_M) THEN (DoAux5Key)
IF (Aux6Key        || KbAux6Key_M) THEN (DoAux6Key)
IF (Aux7Key        || KbAux7Key_M) THEN (DoAux7Key)
IF (Aux8Key        || KbAux8Key_M) THEN (DoAux8Key)
```

```
IF (Aux9Key        || KbAux9Key_M) THEN (DoAux9Key)
IF (Aux10Key       || KbAux10Key_M) THEN (DoAux10Key)
IF (Aux11Key       || KbAux11Key_M) THEN (DoAux11Key)
IF (Aux12Key       || KbAux12Key_M) THEN (DoAux12Key)
IF x1JogLED                 THEN (SelectX1JogInc)
IF x10JogLED                THEN (SelectX10JogInc)
IF x100JogLED               THEN (SelectX100JogInc)
IF IncrContLED              THEN (SelectIncContJog)
IF FastSlowLED              THEN (SelectFastSlowJog)
IF MPGLED                   THEN (SelectMpgMode)
IF FeedHoldLED              THEN (DoFeedHold)

;--Coolant Functions

;--Toggle auto coolant mode
IF CoolAutoManKey || KbTogCoolAutoMan_M THEN (CoolantAutoManualPD)

IF (!CoolAutoManLED && CoolantAutoManualPD) || OnAtPowerUp_M
  THEN SET CoolAutoManLED

IF (CoolAutoManLED && CoolantAutoManualPD)
  THEN RST CoolAutoManLED

;--Report coolant mode to CNC11
IF CoolAutoManLED THEN (SelectCoolAutoMan)

;--Display coolant mode message
;changing to auto coolant mode ;9050 Auto Coolant Selected 2 + 50*256
IF (!CoolAutoManLED && CoolantAutoManualPD)
  THEN AsyncMsg_W = 12802, MSG AsyncMsg_W

;changing to manual coolant mode ;9051 Manual Coolant Selected 2 + 51*256
IF (CoolAutoManLED && CoolantAutoManualPD)
  THEN  AsyncMsg_W = 13058, MSG AsyncMsg_W

;--Flood coolant on/off
IF ((CoolFloodKey || KbFloodOnOff_M) && !CoolAutoManLED) ||
  (M8 && CoolAutoManLED) || (DoCycleStart && M8 && CoolAutoManLED)
  THEN (CoolantFloodPD)

IF CoolantFloodPD && !CoolFloodLED Then SET CoolFloodLED, Set Flood

IF SV_STOP || (CoolantFloodPD && CoolFloodLED) || (!M8 && CoolAutoManLED) ||
  (M8 && !CoolAutoManLED) || DoToolCheck THEN Rst Flood, Rst CoolFloodLED

IF CoolFloodLED THEN (SelectCoolantFlood)

;--Mist coolant on/off
IF ((CoolMistKey || KbMistOnOff_M)&& !CoolAutoManLED) || (M7 && CoolAutoManLED)
  || (DoCycleStart && M7 && CoolAutoManLED) THEN (CoolantMistPD)

IF (CoolantMistPD && !CoolMistLED) THEN SET Mist, SET CoolMistLED

IF SV_STOP || (CoolantMistPD && CoolMistLED) || (!M7 && CoolAutoManLED) ||
```

```
  (M7 && !CoolAutoManLED) || DoToolCheck THEN Rst Mist, Rst CoolMistLED

IF CoolMistLED THEN (SelectCoolantMist)

;--Spindle Control
;-------------------------------------------------------------
; JOGBOARD SPINDLE CONTROL
; Spindle Auto Mode / Manual mode toggles via Auto/Man jog panel key
; CW/CCW jog keys determine spindle direction in manual mode
; M3/M4 system variables determine spindle direction in Auto mode
; Spindle can be stopped and restarted in auto mode using
; spin stop/start jog keys
;-------------------------------------------------------------
;--Select Auto or Manual Spindle Operation Mode
;Triggers to Toggle Auto/Manual Spindle Mode
IF SpinAutoManKey || KbTogSpinAutoMan_M THEN (SpinAutoManPD)

;--Set spindle to auto mode on startup
IF (SpinAutoManPD && !SpinAutoModeLED) || OnAtPowerUp_M
  THEN SET SpinAutoModeLED

;--Set spindle to manual mode
if SpinAutoManPD && SpinAutoModeLED THEN rst SpinAutoModeLED

;--Report the Spindle mode to CNC11
IF SpinAutoModeLED THEN (SelectManAutoSpindle)

;--Set triggers to start and stop the spindle
; NOTE: SpindlePause_M allows the operator to start and stop the
; spindle with the spin start and stop keys while in a job.  In
; this case, pressing the spindle start key will only restart
; the spindle if an M3 or M4 had previously been issued and is
; still active.

IF ((SpinStartKey || KbSpinStart_M) && !SpinAutoModeLED) ||
  (SpinAutoModeLED && (M3 || M4) && !SpindlePause_M) ||
  ((SpinStartKey || KbSpinStart_M) && ((M3 || M4) && SpinAutoModeLED))
  THEN (SpinStartPD), Rst SpindlePause_M

If (SpinAutoModeLED && (M3 || M4) && (SpinStopKey || KbSpinStop_M))
  THEN set SpindlePause_M

If (SpinStopKey || KbSpinStop_M) || (SpinAutoModeLED && !M3 && !M4) ||
  (SpinAutoManPD && SpindleEnableOut) || (SV_PC_RIGID_TAP_SPINDLE_OFF &&
  SpinAutoModeLED) THEN (SpinStopPD)

;--Adjust spindle override when entering manual or auto spin mode
;Set the override value to 100% when spin auto mode is first selected
IF SpinAutoManPD && !SpinAutoModeLED
  THEN SV_PLC_SPINDLE_KNOB = 100,
  SET SpinAutoModeLED

;Set the override value to 10% whenever manual mode is entered
IF SpinAutoManPD && SpinAutoModeLED
```

```
  THEN SV_PLC_SPINDLE_KNOB = 10, Rst SpinAutoModeLED

;--Set spindle direction
;------------------Set Clockwise direction
IF ((KbSpinCW_M || SpinCWKey) && !SpinAutoModeLED) || (M3 && SpinAutoModeLED)
  || (M3 && DoCycleStart) then (SpinCWPD)
IF SpinCWPD then rst SpindleDirectionOut
IF !SpindleDirectionOut then (SpindleCWLED), (SelectSpindleCW)

;---------------Set Counterclockwise direction
IF ((KbSpinCCW_M || SpinCCWKey) && !SpinAutoModeLED) || (M4 && SpinAutoModeLED)
  || (M4 && DoCycleStart) then (SpinCCWPD)
IF SpinCCWPD then set SpindleDirectionOut
IF SpindleDirectionOut then (SpindleCCWLED), (SelectSpindleCCW)

;-------------------------------------------------------------------------------
;                                Turn spindle on/off
;-------------------------------------------------------------------------------
IF ProbeDetect && SpinStartPD THEN set ProbeFault_M

IF (SpindleEnableOut || SpinStartPD ) &&
   !(SpinStopPD || SV_STOP || ProbeDetect)
  THEN (SpindleEnableOut)

IF !SpindleEnableOut  THEN (DoSpindleStop)

;-----------------------------------------------------
;          SPINDLE OVERRIDE CONTROL
;       Jogboard (-, +, and 100% keys),
; Keyboard "ctrl" + "<", "ctrl" + ">", "ctrl" + "/"
;-----------------------------------------------------
IF SpinOverPlusKey || KbIncSpinOver_M
  THEN SV_PLC_SPINDLE_KNOB = SV_PLC_SPINDLE_KNOB + 1
IF SpinOverMinusKey || KbDecSpinOver_M
  THEN SV_PLC_SPINDLE_KNOB = SV_PLC_SPINDLE_KNOB - 1
IF SpinOver100Key || KbSpinOver100_M || OnAtPowerUp_M
  THEN SV_PLC_SPINDLE_KNOB = 100

IF SV_PLC_SPINDLE_KNOB < 1  THEN SV_PLC_SPINDLE_KNOB = 1
IF SV_PLC_SPINDLE_KNOB > 200 THEN SV_PLC_SPINDLE_KNOB = 200

IF SV_PLC_SPINDLE_KNOB == 100 THEN
  (SpinOver100LED),
  (SelectSpinOr100)

IF SV_PLC_SPINDLE_KNOB < 100 THEN
  (SpinOverMinusLED),
  (DoDecreaseSpindleOr )

IF SV_PLC_SPINDLE_KNOB > 100 THEN
  (SpinOverPlusLED),
  (DoIncreaseSpindleOr)

;--Output 12-bit DAC value for spindle control
```

```
;-------------------------------------------------------------------------
;             Read spindle range inputs and/or range M codes
;
; NOTE: SV_SPINDLE_LOW_RANGE & SV_SPINDLE_MID_RANGE M are used to report the
; selected spindle range to CNC11
;-------------------------------------------------------------------------
;    hi      med-high  med-low  low
;    0          1        1      0 SV_SPINDLE_MID_RANGE M
;    0          0        1      1 SV_SPINDLE_LOW_RANGE

If SpinLowRange then set SpinLowRange_M
If !SpinLowRange then rst SpinLowRange_M

IF !SpinLowRange THEN (SpinHighRange_M),
                      RST SV_SPINDLE_LOW_RANGE,
                      rst SV_SPINDLE_MID_RANGE

if SpinLowRange_M && (SV_MACHINE_PARAMETER_65 > .01)
  THEN SpinRangeAdjust = SV_MACHINE_PARAMETER_65,
       SET SV_SPINDLE_LOW_RANGE,
       rst SV_SPINDLE_MID_RANGE

if SpinHighRange_M then SpinRangeAdjust = 1
if SpinRangeAdjust == 0 then SpinRangeAdjust = 1

;-----------------------------------------------------------------
;              Read commanded spindle speed, max & min
;
; ***NOTE***  SV_PC_COMMANDED_SPINDLE_SPEED already has override
;             factored in.
;-----------------------------------------------------------------
IF True THEN SpinSpeedCommand_FW =  SV_PC_COMMANDED_SPINDLE_SPEED,
             CfgMinSpeed_FW = SV_PC_CONFIG_MIN_SPINDLE_SPEED,
             CfgMaxSpeed_FW = SV_PC_CONFIG_MAX_SPINDLE_SPEED

;-----------------------------------------------------------------
; If commanded spindle speed is < Min Spin Speed * SpinRangeAdjust
; & commanded spindle speed > 0, force to commanded spindle speed
; = min spin speed value * SpinRangeAdjust.
;-----------------------------------------------------------------
IF (SpinSpeedCommand_FW > 0.0) &&
   (SpinSpeedCommand_FW < (CfgMinSpeed_FW * SpinRangeAdjust))
  THEN SpinSpeedCommand_FW = (CfgMinSpeed_FW * SpinRangeAdjust),
  ErrorCode_W = MIN_SPEED_MSG

;-------------------------------------------------------------------------
; If SpinSpeedCommand_FW > Max Spin Speed * SpinRangeAdjust, force
; SpinSpeedCommand_FW = max spin speed value * SpinRangeAdjust.
;-------------------------------------------------------------------------
IF SpinSpeedCommand_FW > (CfgMaxSpeed_FW * SpinRangeAdjust)
  THEN SpinSpeedCommand_FW = (CfgMaxSpeed_FW * SpinRangeAdjust)

;-----------------------------------------------------------------
; Convert Spindle "S" command to 12 bit value for output to DAC
```

```
;---------------------------------------------------------------
; Commanded Spindle speed (includes override factor) is sent down from CNC11
; in SV_PC_COMMANDED_SPINDLE_SPEED.  This value needs to be converted to a
; 12 bit value (0-4095) where full scale = SV_PC_CONFIG_MAX_SPINDLE_SPEED.

; Calculate #RPM's per bit of resolution
IF CfgMaxSpeed_FW > 0.0 THEN RPMPerBit_FW = CfgMaxSpeed_FW/4095.0
IF CfgMaxSpeed_FW <= 0.0 THEN RPMPerBit_FW = 1.0

;Convert RPM to 12 bit value
IF True THEN TwelveBitSpeed_FW = SpinSpeedCommand_FW/RPMPerBit_FW

; Factor in gear range
IF true THEN TwelveBitSpeed_FW = (TwelveBitSpeed_FW/SpinRangeAdjust)

;Convert to integer word for DAC & I/O display
if true then TwelveBitSpeed_W = TwelveBitSpeed_FW

; Bound min to 0, max to 4095
IF TwelveBitSpeed_W < 0 THEN TwelveBitSpeed_W = 0
IF TwelveBitSpeed_W > 4095 THEN TwelveBitSpeed_W = 4095

; Output to DAC
If true then WTB TwelveBitSpeed_W SpinAnalogOutBit0 12

;Display calculated RPM value on PC
IF True THEN SV_PLC_SPINDLE_SPEED = SpinSpeedCommand_FW

;=============================================================================
  CheckCycloneStatusStage
;=============================================================================
; Due to amount of time it takes to retrieve data from the cyclone, this stage
; is only called few times per second to help reduce scan time of the main PLC
; program.

; The logic below is the equivalent to the following:
; IF true THEN BITTST SV_PC_CYCLONE_STATUS_2 0 Axis1FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 1 Axis2FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 2 Axis3FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 3 Axis4FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 4 Axis5FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 5 Axis6FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 6 Axis7FiberOk_M,
;              BITTST SV_PC_CYCLONE_STATUS_2 7 Axis8FiberOk_M
IF true THEN WTB SV_PC_CYCLONE_STATUS_2 Axis1FiberOk_M

; Generate some messages for fiber or wire to MPU11 having issues
IF SV_AXIS_VALID_1 && !SV_DRIVE_ONLINE_1 THEN ErrorCode_W = AXIS1_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_2 && !SV_DRIVE_ONLINE_2 THEN ErrorCode_W = AXIS2_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_3 && !SV_DRIVE_ONLINE_3 THEN ErrorCode_W = AXIS3_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_4 && !SV_DRIVE_ONLINE_4 THEN ErrorCode_W = AXIS4_INFLT, SET
```

```
DriveComFltIn_M
IF SV_AXIS_VALID_5 && !SV_DRIVE_ONLINE_5 THEN ErrorCode_W = AXIS5_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_6 && !SV_DRIVE_ONLINE_6 THEN ErrorCode_W = AXIS6_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_7 && !SV_DRIVE_ONLINE_7 THEN ErrorCode_W = AXIS7_INFLT, SET
DriveComFltIn_M
IF SV_AXIS_VALID_8 && !SV_DRIVE_ONLINE_8 THEN ErrorCode_W = AXIS8_INFLT, SET
DriveComFltIn_M

; Generate some messages for fiber or wire to drive having issues
IF SV_AXIS_VALID_1 && SV_DRIVE_ONLINE_1 && SV_MASTER_ENABLE && !Axis1FiberOk_M
  THEN ErrorCode_W = AXIS1_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_2 && SV_DRIVE_ONLINE_2 && SV_MASTER_ENABLE && !Axis2FiberOk_M
  THEN ErrorCode_W = AXIS2_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_3 && SV_DRIVE_ONLINE_3 && SV_MASTER_ENABLE && !Axis3FiberOk_M
  THEN ErrorCode_W = AXIS3_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_4 && SV_DRIVE_ONLINE_4 && SV_MASTER_ENABLE && !Axis4FiberOk_M
  THEN ErrorCode_W = AXIS4_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_5 && SV_DRIVE_ONLINE_5 && SV_MASTER_ENABLE && !Axis5FiberOk_M
  THEN ErrorCode_W = AXIS5_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_6 && SV_DRIVE_ONLINE_6 && SV_MASTER_ENABLE && !Axis6FiberOk_M
  THEN ErrorCode_W = AXIS6_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_7 && SV_DRIVE_ONLINE_7 && SV_MASTER_ENABLE && !Axis7FiberOk_M
  THEN ErrorCode_W = AXIS7_OUTFLT, set DriveComFltOut_M
IF SV_AXIS_VALID_8 && SV_DRIVE_ONLINE_8 && SV_MASTER_ENABLE && !Axis8FiberOk_M
  THEN ErrorCode_W = AXIS8_OUTFLT, set DriveComFltOut_M

If !EstopOk THEN rst DriveComFltIn_M, rst DriveComFltOut_M
If DriveComFltOut_M || DriveComFltIn_M THEN set AxisFault_M

;check PLC status bit
IF TRUE THEN BitTst SV_PC_CYCLONE_STATUS_1 21 PLCBusExtDevEn_M

;check input fiber
IF !SV_PLC_BUS_ONLINE THEN ErrorCode_W = PLC_INFLT,
                           rst PLCBus_Oe_M, set PLCFault_M

;check output fiber
IF SV_PLC_BUS_ONLINE && PLCBus_Oe_M && !PLCBusExtDevEn_M
  THEN ErrorCode_W = PLC_OUTFLT, SET PLCFault_M

;clear PLC errors
IF PLCFault_M && SV_PLC_BUS_ONLINE && PLCBusExtDevEn_M && !EstopOk
  THEN RST PLCFault_M, ErrorCode_W = PLC_FLT_CLR, SET PLCBus_Oe_M

IF True THEN RST CheckCycloneStatusStage

;=============================================================================
                      AxesEnableStage
;=============================================================================
; Since CNC11 v3.03r24, the MPU11 has managed axis enables
; directly.  The PLC program has no further responsibility
; for SV_ENABLE_AXIS_n
```

```
;clear axis fault errors
IF AxisFault_M && !(DriveComFltOut_M || DriveComFltIn_M) && !EstopOk
  THEN ErrorCode_W = AXIS_FLT_CLR, RST AxisFault_M



;read the status bits
IF True THEN SET CycloneStatus_T
IF CycloneStatus_T THEN SET CheckCycloneStatusStage, RST CycloneStatus_T

;turn on drives if no drive or drive fiber errors
IF !DriveComFltIn_M && !DriveComFltOut_M THEN SET SV_MASTER_ENABLE
; (SV_MASTER_ENABLE will be turned off by stalls and other
;  errors in the Fault-handling section of MainStage, below.)

;(If there is any drive or drive fiber error, then AxisFault_M will have
; been set previously, which will cause SV_STOP to be set, and SV_MASTER_ENABLE
; to be reset, later in MainStage)

;=============================================================================
                           MainStage
;=============================================================================
;Do gather if commanded (uncomment and recompile for debugging purposes)
;IF Aux11Key THEN (Aux11KeyPD)
;If Aux11KeyPD THEN (SV_TRIGGER_PLOT_DUMP)

;read parameter 178 and check the Lube NO/NC state

; Get the previous value of the debounce configuration word
IF True THEN Inputs_9_12_W = SV_PLC_DEBOUNCE_3

; Invert input 9
IF  InvLubeOk_M THEN BITSET Inputs_9_12_W 6
IF !InvLubeOk_M THEN BITRST Inputs_9_12_W 6

 ; Invert input 10
IF  InvSpinInverterOk_M THEN BITSET Inputs_9_12_W 14
IF !InvSpinInverterOk_M THEN BITRST Inputs_9_12_W 14

; Write back to the debounce configuration word
IF true THEN SV_PLC_DEBOUNCE_3 = Inputs_9_12_W

;-----------------------------------------------------------------
;           Probe protection while jogging
;-----------------------------------------------------------------
If MechanicalProbe && !JogProbeFault_M && (DoAx1PlusJog || DoAx1MinusJog ||
  DoAx2PlusJog || DoAx2MinusJog || DoAx3PlusJog || DoAx3MinusJog ||
  DoAx4PlusJog || DoAx4MinusJog || DoAx5PlusJog || DoAx5MinusJog)
  THEN (JogProbeFaultPD)

IF MechanicalProbe && !JogProbeFault_M && FastSlowLED THEN SET LastProbeMode_M
IF MechanicalProbe && !JogProbeFault_M && !FastSlowLED THEN RST LastProbeMode_M
```

```
IF JogProbeFaultPD && !JogProbeFault_M THEN SET JogProbeFault_M, SET DoCycleCancel

IF JogProbeFault_M THEN ErrorCode_W = (PROBE_JOG_FAULT_MSG + 1),
                        SET FastSlowLED

IF !MechanicalProbe && JogProbeFault_M && !LastProbeMode_M THEN RST FastSlowLED

IF !MechanicalProbe THEN RST JogProbeFault_M,
                         RST Ax1PlusJogDisabled_M,
                         RST Ax1MinusJogDisabled_M,
                         RST Ax2PlusJogDisabled_M,
                         RST Ax2MinusJogDisabled_M

;--Clamp
IF M10 THEN (Clamp)  ; cleared by M11 or by program not running

;--Process important Keyboard keys all the time
;--Cycle Cancel (ESC)
if Kb_Escape THEN (KbCycleCancel_M)
;KbFeedHold_M (spacebar)
if Kb_spacebar && AllowKbInput_M && SV_PROGRAM_RUNNING then (KbFeedHold_M)

if SV_PC_VIRTUAL_JOGPANEL_ACTIVE THEN (KbJpActive_M)

;Call KeyboardEvents stage if needed
IF Kb_L_Ctrl || Kb_R_Ctrl ||
   Kb_L_Shift || Kb_R_Shift ||
   Kb_R_Alt || Kb_L_Alt || KbJpActive_M
  THEN SET KeyboardEventsStage

if (Kb_L_Ctrl || Kb_R_Ctrl || Kb_L_Shift || Kb_R_Shift || Kb_R_Alt || Kb_L_Alt)
  && (Kb_j || Kb_f || kb_a || kb_s) && !AllowKbInput_M
  THEN ErrorCode_W = KB_JOG_MSG

;--Handle Faults
if !EStopOk || PLCFault_M || SV_STALL_ERROR || SpindleFault_M ||
  LubeFault_M || AxisFault_M ||  ProbeFault_M || OtherFault_M THEN set SV_STOP

IF SV_Stop THEN RST SV_MASTER_ENABLE

IF !EstopOk THEN RST SV_STALL_ERROR,
                 RST LubeFault_M,
                 RST SpindleFault_M,
                 RST OtherFault_M,
                 RST ProbeFault_M,
                 RST ProbeMsgSent_M

IF Initialize_T && !LubeOk && !SV_PROGRAM_RUNNING
  THEN SET LubeFault_M, ErrorCode_W = LUBE_FAULT_MSG

IF !LubeOk && SV_PROGRAM_RUNNING THEN ErrorCode_W = LUBE_WARNING_MSG

IF Initialize_T && !SpindleInverterOk
  THEN ErrorCode_W = SPINDLE_FAULT_MSG, SET SpindleFault_M
```

```
IF !EstopOK && !SpindleInverterOk THEN (InverterResetOut)

; Echo some system variables to memory bits, for troubleshooting only
IF SV_MASTER_ENABLE THEN (MasterEnable_M)
IF SV_STALL_ERROR THEN (Stall_M)
IF SV_STOP THEN (Stop_M)

IF !SV_STOP THEN (NoFaultOut)

IF EStopOk &&
   !(PLCFault_M || SV_STALL_ERROR || SpindleFault_M || LubeFault_M ||
     AxisFault_M || OtherFault_M || SoftwareNotReady_M || PLCExecutorFault_M)
  THEN RST SV_STOP

IF ProbeFault_M && !ProbeMsgSent_M
  THEN ErrorCode_W = PROBE_FAULT_MSG, SET ProbeMsgSent_M

;--M-Codes
;    Reset these M-codes if not in CNC Program Running mode
IF !(SV_PROGRAM_RUNNING || SV_MDI_MODE) THEN RST M3, RST M4, RST M8, RST M7, RST M10

;--turn off default setup variable
IF true THEN RST OnAtPowerUp_M

;=================================================================
   SetErrorStage
;=================================================================
IF !((ErrorCode_W % 256 == 1) || (ErrorCode_W % 256 == 2)) THEN JMP BadErrorStage
IF true THEN MSG ErrorCode_W
If ErrorCode_W != MSG_CLEARED_MSG Then MsgClear_T = 1000, set MsgClear_T

IF (!EstopOk && !SoftwareNotReady_M) ||
   ((ErrorCode_W != MSG_CLEARED_MSG) && (ErrorCode_W % 2 == 0) && MsgClear_T)
  THEN ErrorCode_W = MSG_CLEARED_MSG, RST MsgClear_T


;=============================================================================
   BadErrorStage
;=============================================================================
IF true THEN AsyncMsg_W = 2+256*100, MSG AsyncMsg_W, AsyncMsg_W = 0
IF True THEN RST BadErrorStage
```

Appendix B: Jog Panel Mapping

JogPanel Inputs and Outputs

The shorthand reference for JogPanel Inputs and Outputs JPI1-JPI256 and JPO1-JP0256 are remapping of Inputs and Outputs 1057-1312. Note that JPI stands for **J**og **P**anel **I**nput and JPO stands for **J**og **P**anel **O**utput so the letters after JP are the letters 'eye' and 'oh' capitalized.

Spindle + Key JPI**1**/INP**1057** LED JPO**1**/OUT**1057**	Spindle Auto/Man Key JPI**2**/INP**1058** LED JPO**2**/OUT**1058**	Aux1 Key JPI**3**/INP**1059** LED JPO**3**/OUT**1059**	Aux2 Key JPI**4**/INP**1060** LED JPO**4**/OUT**1060**	Aux3 Key JPI**5**/INP**1061** LED JPO**5**/OUT**1061**
Spindle 100% Key JPI**6**/INP**1062** LED JPO**6**/OUT**1062**	Spindle CW Key JPI**7**/INP**1063** LED JPO**7**/OUT**1063**	Aux4 Key JPI**8**/INP**1064** LED JPO**8**/OUT**1064**	Aux5 Key JPI**9**/INP**1065** LED JPO**9**/OUT**1065**	Aux6 Key JPI**10**/INP**1066** LED JPO**10**/OUT**1066**
Spindle - Key JPI**11**/INP**1067** LED JPO**11**/OUT**1067**	Spindle CCW Key JPI**12**/INP**1068** LED JPO**12**/OUT**1068**	Aux7 Key JPI**13**/INP**1069** LED JPO**13**/OUT**1069**	Aux8 Key JPI**14**/INP**1070** LED JPO**14**/OUT**1070**	Aux9 Key JPI**15**/INP**1071** LED JPO**15**/OUT**1071**
Spindle Stop Key JPI**16**/INP**1072** LED JPO**16**/OUT**1072**	Spindle Start Key JPI**17**/INP**1073** LED JPO**17**/OUT**1073**	Aux10- Key JPI**18**/INP**1074** LED JPO**18**/OUT**1074**	Aux11- Key JPI**19**/INP**1075** LED JPO**19**/OUT**1075**	Aux12- Key JPI**20**/INP**1076** LED JPO**20**/OUT**1076**
Coolant Auto/Man Key JPI**21**/INP**1077** LED JPO**21**/OUT**1077**	Flood Key JPI**22**/INP**1078** LED JPO**22**/OUT**1078**	Mist Key JPI**23**/INP**1079** LED JPO**23**/OUT**1079**	Aux13- Key JPI**24**/INP**1080** LED JPO**24**/OUT**1080**	Aux14- Key JPI**25**/INP**1081** LED JPO**25**/OUT**1081**
INCR/CONT Key JPI**26**/INP**1082** LED JPO**26**/OUT**1082**	X1 Key JPI**27**/INP**1083** LED JPO**27**/OUT**1083**	X10 Key JPI**28**/INP**1084** LED JPO**28**/OUT**1084**	X100 Key JPI**29**/INP**1085** LED JPO**29**/OUT**1085**	MPG Key JPI**30**/INP**1086** LED JPO**30**/OUT**1086**
Axis4+ Key JPI**31**/INP**1087** LED JPO**31**/OUT**1087**	Blank- Key JPI**32**/INP**1088** LED JPO**32**/OUT**1088**	Axis2+ Key JPI**33**/INP**1089** LED JPO**33**/OUT**1089**	Blank- Key JPI**34**/INP**1090** LED JPO**34**/OUT**1090**	Axis3+ Key JPI**35**/INP**1091** LED JPO**35**/OUT**1091**
Blank- Key JPI**36**/INP**1092** LED JPO**36**/OUT**1092**	Axis1- Key JPI**37**/INP**1093** LED JPO**37**/OUT**1093**	SLOW/FAST Key JPI**38**/INP**1094** LED JPO**38**/OUT**1094**	Axis1+ Key JPI**39**/INP**1095** LED JPO**39**/OUT**1095**	Blank- Key JPI**40**/INP**1096** LED JPO**40**/OUT**1096**

Axis4- Key JPI**41**/INP**1097** LED JPO**41**/OUT**1097**	Blank- Key JPI**42**/INP**1098** LED JPO**42**/OUT**1098**	Axis2- Key JPI**43**/INP**1099** LED JPO**43**/OUT**1099**	Blank- Key JPI**44**/INP**1100** LED JPO**44**/OUT**1100**	Axis3- Key JPI**45**/INP**1101** LED JPO**45**/OUT**1101**
CYCLE CANCEL Key JPI**46**/INP**1102** LED JPO**46**/OUT**1102**	SINGLE BLOCK Key JPI**47**/INP**1103** LED JPO**47**/OUT**1103**	TOOL CHECK* Key JPI**48**/INP**1104** LED JPO**50**/OUT**1106**	FEED HOLD* Key JPI**49**/INP**1105** LED JPO**48**/OUT**1104**	CYCLE START* Key JPI**50**/INP**1106** LED JPO**49**/OUT**1105**

Appendix C: Keyboard Jog Mapping

Notes on Keyboard Jogging

The following table lists the SV_PC_KEYBOARD_KEY_1 to _104 system variables and the keys that they represent on a 101-key or 104-key QWERTY US English traditional keyboard. These keys have not been tested with any international/multilingual keyboard.

There is a System Variable called SV_PC_VIRTUAL_JOGPANEL_ACTIVE that is SET when the Keyboard Jogging screen (**Alt.-J**) is visible from the main menu of CNC11. Keyboard Jogging should not allow axis jogging unless this system variable is set, otherwise the motors may move while cursoring around in the Load menu. The other standard Jog Panel functions can be allowed all the time or restricted based on safety requirements for the particular installation. By default, standard PLC programs with Keyboard Jogging allow everything except axis jogging when the Keyboard Jogging screen is not visible. This mirrors the functionality of the Jog Panel itself.

Keys left blank should not be used. The second character of the two-character keys is the **Shift**ed key. This means that SV_PC_KEYBOARD_KEY_60 indicating the **a** or **A** has been pushed will look the same in a PLC program. It is the responsibility of the programmer to look for the **Shift** key separately. On many keyboards there is at least one **Windows, System**, **Print Screen**, **Scroll Lock** and **Pause/Break** key and sometimes multiple, but these keys are not accessible in the PLC program. There are usually two **Alt.**, **Shift** and **Control** keys as well on each keyboard and the PLC sees either one as the same key-press. The right hand key is reported as the left key being pressed so do not look for the right key. The number pad may not be present on every keyboard so it is not advised to write a Jogging PLC program looking for these keys to be used except as a custom program. The number pad keys are preceded with NP_ in the table below.

Keyboard Key Numbering Table

Index	Key	Index	Key	Index	Key	Index	Key
1	ESC	27	0)	53	END	79	bB
2	F1	28	-_	54	PAGEDOWN	80	nN
3	F2	29	=+	55	NP_7	81	mM
4	F3	30	BACKSPACE	56	NP_8	82	,<
5	F4	31	INSERT	57	NP_9	83	.>
6	F5	32	HOME	58	NP_PLUS	84	/?
7	F6	33	PAGEUP	59	CAPSLOCK	85	Right SHIFT(74)*
8	F7	34	NUMLOCK	60	aA	86	|
9	F8	35	NP_DIVIDE	61	sS	87	UP Arrow
0	F9	36	NP_MULT	62	dD	88	NP_1
11	F10	37	NP_MINUS	63	fF	89	NP_2
12	F11	38	TAB	64	gG	90	NP_3
13	F12	39	qQ	65	hH	91	NP_ENTER
14		40	wW	66	jJ	92	Left CTRL
15		41	eE	67	kK	93	
16		42	rR	68	lL	94	Left ALT
17	`~	43	tT	69	;:	95	SPACEBAR
18	1!	44	yY	70	'"	96	Right ALT(94)*
19	2@	45	uU	71	NP_4	97	
20	3/#	46	iI	72	NP_5	98	
21	4$	47	oO	73	NP_6	99	Right CTRL(92)*
22	5%	48	pP	74	Left SHIFT	100	LEFT Arrow
23	6^	49	[{	75	zZ	101	DOWN Arrow
24	7&	50	]}	76	xX	102	RIGHT Arrow
25	8*	51	ENTER	77	cC	103	NP_0
26	9(	52	DELETE	78	vV	104	NP_DP

* key event returns same index as the Left side key (shown in parenthesis)

Appendix D: System Variables

System Variable Types

The following table gives the shorthand version of the data types listed below in the Type column.

Type	Description
M	Memory bit
I32	32-bit signed Integer
I64	64-bit signed Integer
F32	32-bit Floating-point (float)
F64	64-bit Floating-point (double)

Notes on Certain System Variables

There are some System Variables that are written to or read from outside of the scope of execution of the PLC program. These System Variables should be changed in only one place in the PLC program to avoid odd effects and problems troubleshooting.

Externally Read System Variables

The table below lists the system variables that are read external to the PLC program execution and that could have different values if the PLC program changed them at several places. In general, these variables should only be updated once in a single pass. Rather than rely on stage activity or other complicated logic to ensure they are updated once per pass, it is best to write to a temporary variable and then use the temporary variable to set the system variable only once.

Bad Example:

```
IF EStop   THEN SET SV_STOP
IF !EStop  THEN RST SV_STOP
IF LowLube THEN SET SV_STOP
```

Good Example:

```
IF EStop   THEN SET TEMP_SV_STOP
IF !EStop  THEN RST TEMP_SV_STOP
IF LowLube THEN SET TEMP_SV_STOP
IF TEMP_SV_STOP  THEN SET SV_STOP
IF !TEMP_SV_STOP THEN RST SV_STOP
```

System Variable	Notes
SV_DRIVE_CONTROL_x	Used to SET the auxiliary output of an OPTIC4 axis.
SV_STOP*	
SV_SPINDLE_RPM_MODE*	Used for "C" axis.
SV_PLC_FEEDRATE_KNOB*	
SV_PLC_FEEDRATE_OVERRIDE*	
SV_PLC_FUNCTION_34*	Rapid Override. This usage should be deprecated in favor of SV_PC_TOGGLE_RAPID_OVERRIDE
SV_PLC_OP_IN_PROGRESS	This SV should not be used. It is listed here for completeness.
SV_MPG_x_AXIS_SELECT	The change will not take effect unless MPG movement is stopped.
SV_MPG_x_MULTIPLIER	
SV_MPG_x_WINDUP_MODE	
SV_MPG_x_ENABLED	The change will not take effect unless MPG movement is stopped.
SV_MPG_x_OFFSET_MODE	The change will not take effect unless MPG movement is stopped.
SV_MPG_x_PLC_MPG_MODE	The change will not take effect unless MPG movement is stopped.

Most likely to have unwanted side effects.

Externally Written System Variables

The table below lists the system variables that are written external to PLC program execution and therefore a PLC program could read at multiple times in a single pass and get different values. Thus, these variables should only be read once in a single pass. Rather than rely on stage activity or other complicated logic to ensure they are read once per pass, it is best to store the value into a temporary variable at the start of execution and then refer to that temporary variable throughout the program.

Bad Example:

```
IF SV_STALL_ERROR  THEN (MEM1)
IF !SV_STALL_ERROR THEN (MEM2)
```

Good Example:

```
IF TRUE THEN TEMP_SV_STALL_ERROR = SV_STALL_ERROR
IF TEMP_SV_STALL_ERROR  THEN (MEM1)
IF !TEMP_SV_STALL_ERROR THEN (MEM2)
```

Note that in the bad example, there is no guarantee that MEM1 != MEM2.

System Variable	Notes
SV_DRIVE_STATUS_x*	Typically used with OPTIC4 that has no external drive fault input visible to the PLC program (GPIO4D has INP17-20).
SV_STALL_ERROR*	
SV_PC_FEEDRATE_PERCENTAGE*	
SV_FSIOx	
SV_M_FUNCTION	Not used in any program.
SV_PC_POWER_AXIS_x	
SV_MPU11_LASH_OFFSET_x	
SV_MPU11_ABS_POS_x	
SV_MPU11_EXPECTED_POS_x	
SV_PC_RIGID_TAP_SPINDLE_OFF	

Note that all system variables are readable by either the PLC or CNC11. The following tables indicate who, either the PLC or CNC11 software, can change the state of the Bits or Words. In some cases there are System Variables that CNC11 and the PLC program can write to and it is so noted.

CNC11 Software Write-Controlled System Variables

Name	Type	Description
1-bit Boolean System Variables (CNC11 ---> PLC)		
SV_M94_M95_1-128	M	Used for M-Codes that require PLC interaction like M3, M4, M6, M7, M8, M10, M11 and custom M-Codes. They are set and reset from M & G-code programs using M94 and M95. These bits can also be controlled by the PLC program even though they are in the section labeled as Read Only for the PLC. Be aware that CNC11 has built-in default actions for some M-codes that control the first 16 of these variables. An example is to turn off auto coolant and spindle when not running a program. IF !SV_PROGRAM_RUNNING THEN RST M3, RST M4, RST M7, RST M8.
SV_PROGRAM_RUNNING	M	1 = MDI mode or a job is in progress
SV_MDI_MODE	M	1 = MDI mode active
SV_PC_OVERRIDE_CONTROL_FEEDRATE_OVERRIDE	M	1 = FeedRate Override Knob is allowed to change the feedrate on axis motion

SV_PC_OVERRIDE_CONTROL_SPINDLE_OVERRIDE	M	1 = Spindle Override keys or Knob change the spindle speed commanded
SV_PC_OVERRIDE_CONTROL_FEEDHOLD	M	1 = Feedhold pauses the G-Code program
SV_PC_POWER_AXIS_1-_8	M	1 = indicates if the axis is powered to hold position
SV_DRIVE_ONLINE_1-_8	M	1 = The drive for the axis is detected.
SV_LEGACY_JOG_PANEL_ONLINE	M	1 = Legacy Jog Panel detected. Ex: Uniconsole-2
SV_PLC_BUS_ONLINE	M	1 = Valid MPU11 PLC detected. Checked as part of Fiber Checking PLC Program section.
SV_JOG_LINK_ONLINE	M	1 = Valid Jog Panel detected as Jogboard.
SV_PLC_IO2_ONLINE	M	1 = IO2 Legacy PLC detected.
SV_AXIS_VALID_1-_8	M	1 = Motor Parameters screen axis label allows motion. Allowed labels are X, Y, Z, A, B, C, U, V, W.
SV_PC_RIGID_TAP_SPINDLE_OFF	M	CNC software SETs this bit to signal that the spindle should be turned off when the rigid tap depth has been reached if Parameter 36, bit 4 is SET, otherwise it is not needed. This is usually not needed. It is NOT needed just because you have a custom mfunc5.mac file. CNC software ignores custom M5s during a rigid tap. It decides which bit to turn turn off (M3 or M4) based on the spindle direction. The only time that this system variable is needed is as follows: If turning off the spindle requires doing more than clearing M3 or M4. Note that CNC software will only clear one of M3 or M4 during a rigid tap, not both.
SV_PC_KEYBOARD_KEY_1-104	M	1 = Keyboard key pressed, 0 = Keyboard key not pressed. See Appendix C.
SV_PC_VIRTUAL_JOGPANEL_ACTIVE	M	Indicates whether keyboard jogging is enabled i.e. the user has activated the keyboard (virtual) jog panel using ALT-J. Most things besides jogging are allowed without the screen being up by default.
SV_PC_HOME_SET	M	1 = home set for all valid axes, check this to force slow jog before homing unless Parameter 148 bit 0 is set.
SV_PC_SOFTWARE_READY	M	1 = CNC software is initialized and communicating with MPU normally 0 = CNC software exited normally or communication fault occurred. Intended to be used in a PLC program to disable certain functions when the CNC software is not running or there is another fault preventing normal communications. For example, on ATC machines, the carousel should not be rotated if the software is not running since the CNC software monitors the reported carousel position and saves changes to the job file.
SV_PC_TOGGLE_RAPID_OVERRIDE	M	CNC software SETs this bit when the state of Rapid Override needs to change. The PLC should RSTs this bit after toggling Rapid Override.
SV_DOING_AUTO_TOOL_MEASURE	M	This bit is set by the CNC software when performing an automatic tool measure with the TT1.

SV_JOG_PANEL_REQUIRED	M	Indicates the setting of “Jog Panel Required” in the Control Configuration Menu of CNC software.
SV_LIMIT_TRIPPED	M	1 = Any configured limit switch is tripped 0 = No configured limit switches are tripped Note: The PLC program is not required to perform any actions when this bit is set.
SV_JOB_IN_PROGRESS	M	Set when CNC software is running a job or running an MDI command, but not while at the MDI prompt waiting for input.
SV_RPM_MODE_ZERO_SPEED_1-8	M	1 = AC1 drive axis is at zero speed in RPM mode
SV_RPM_MODE_ACTIVE_1-8	M	1 = AC1 drive axis is in RPM mode
SV_?_AXIS_VALID	M	These nine system variables are mapped to the SV_AXIS_VALID_x bits according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_?_AXIS_DRIVE_ONLINE	M	These nine system variables are mapped to the SV_DRIVE_ONLINE_x bits according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_?_AXIS_FIBER_OK	M	These nine system variables are mapped to the bits in SV_PC_CYCLONE_STATUS_2 according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_?_AXIS_POWERED	M	These nine system variables are mapped to the SV_PC_POWER_AXIS_x bits according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_SKIN_EVENT_1-128	M	These 128 system variable bits are used to provide generic events that can be SET/RST by skinning applications via the CNC Skinning API. The Centroid Virtual Control Panel (VCP) is a skinning application that uses them to communicate the status of virtual jog panel button states, among other things. As a convention, SV_SKIN_EVENT_1 – 50 should be mapped to the same fifty keys found on a real hardware jog panel, i.e., SV_SKIN_EVENT_1 corresponding to Spindle+ key, and then continuing left to right, top to bottom with SV_SKIN_EVENT_50 corresponding to the CYCLE_START key.
32-bit Integer System Variables (CNC11 ---> PLC)		
SV_PC_DAC_SPINDLE_SPEED	I32	The DAC spindle speed as requested by the CNC software (value range is 0-65535)
SV_M_FUNCTION	I32	Set as part of M-code execution.
SV_TOOL_NUMBER	I32	Set as part of a tool change (M107), to indicate the requested tool number. In the case of “enhanced ATC” operation this is actually a request for a carousel bin

<table>
<tr><td></td><td></td><td>location. (The LDT command of XPLCCOMP is deprecated and not used.)</td></tr>
<tr><td>SV_ATC_CAROUSEL_POSITION</td><td>I32</td><td>Sent by CNC software when it first starts up or as part of an “enhanced ATC” reset feature. (The LCP command of XPLCCOMP is deprecated and not used.)</td></tr>
<tr><td>SV_ATC_TOOL_IN_SPINDLE</td><td>I32</td><td>Sent by CNC software when it first starts up and the .job file is parsed or as part of an “enhanced ATC” reset feature. (The LTS command of XPLCCOMP is deprecated and not used.)</td></tr>
<tr><td>SV_PC_FEEDRATE_PERCENTAGE</td><td>I32</td><td>0-200% adjustment for axis motion control. The value is sent by CNC software for machine parameter 78 bit 1 checking and on-screen-display. This value is not needed if SV_PC_OVERRIDE_CONTROL_FEEDRATE_OVERRIDE is not SET.</td></tr>
<tr><td>SV_PC_SPINDLE_OVERRIDE</td><td>I32</td><td>Not currently used.</td></tr>
<tr><td>SV_PC_CYCLONE_STATUS_1</td><td>I32</td><td>PLC and Drive Status Bits
<table>
<tr><th>Bit</th><th>Function</th></tr>
<tr><td>0-20</td><td>Reserved</td></tr>
<tr><td>21</td><td>Out Fiber for PLC OK</td></tr>
<tr><td>22-31</td><td>Reserved</td></tr>
</table></td></tr>
</table>

<table>
<tr>
<td>SV_PC_CYCLONE_STATUS_2</td>
<td>I32</td>
<td>The PLC program should check drive enables from this word to determine if fiber4 has been broken and cause a drive fault
<table>
<tr><th>Bit</th><th>Function</th></tr>
<tr><td>0</td><td>Drive Axis 1</td></tr>
<tr><td>1</td><td>Drive Axis 2</td></tr>
<tr><td>2</td><td>Drive Axis 3</td></tr>
<tr><td>3</td><td>Drive Axis 4</td></tr>
<tr><td>4</td><td>Drive Axis 5</td></tr>
<tr><td>5</td><td>Drive Axis 6</td></tr>
<tr><td>6</td><td>Drive Axis 7</td></tr>
<tr><td>7</td><td>Drive Axis 8</td></tr>
<tr><td>9-31</td><td>Reserved</td></tr>
</table></td>
</tr>
<tr>
<td>SV_PC_MINI_PLC_ONLINE</td>
<td>I32</td>
<td>Online bits for PLCADD1616 and other expansion PLC modules.
<table>
<tr><td>Bit</td><td>Function</td></tr>
<tr><td>0</td><td>1 = miniPLC1 is online</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>15</td><td>1 = miniPLC16 online</td></tr>
<tr><td>16-31</td><td>Reserved</td></tr>
</table></td>
</tr>
</table>

SV_DRIVE_STATUS_1-8

I32

High Speed Drive Status word, separated by Drive type.

ALLIN1DC/DC3IOB	
Bit	**Description**
0	Current Setting Low Bit
1	Current Setting High Bit
2	High power enable - FETs are installed to handle 15A current setting
3	Drive master/slave – 1 = communicating on fibers, 0 = comm. on wires
4	Aux 1 jumper state – 1 = jumper block removed, 0 = jumper block in place
5	Aux 2 jumper state – 1 = jumper block removed, 0 = jumper block in place
6	Reserved
7	Reserved
8-15	Reserved

DC1	
Bit	**Description**
0	Current Setting Low Bit
1	Current Setting High Bit
2	High power enable - FETs are installed to handle 15A current setting
3	Reserved
4	Reserved
5	Spare jumper state – 1 = jumper block removed, 0 = jumper block in place
6	Plus Limit State
7	Minus Limit State
8-15	Reserved

Optic4	
Bit	**Description**
0-10	Reserved
11	Quadrature Error: Incorrect

<table>
<tr><td></td><td></td><td><table>
<tr><td></td><td>encoder state transition</td></tr>
<tr><td>12</td><td>Direction Bit</td></tr>
<tr><td>13</td><td>Index Pulse</td></tr>
<tr><td>14</td><td>Differential error on encoder A or B channel</td></tr>
<tr><td>15</td><td>Drive fault from 3rd party drive (1 = Fault, or no voltage at input)</td></tr>
</table></td></tr>
<tr><td>SV_MPU11_LASH_OFFSET_0-
SV_MPU11_LASH_OFFSET_7</td><td>I32</td><td>The current lash offset</td></tr>
<tr><td>SV_FSIO_1-SV_FSIO32</td><td>I32</td><td>Fast Synchronous IO (see M300 commands in the Operator Manual)</td></tr>
<tr><td>SV_STALL_REASON</td><td></td><td>Set whenever SV_STALL_ERROR is set.
No Error = 0
position error = 1
full power without motion = 2
encoder differential error = 3
spindle slave position error = 4
OpticDirect C8 Error = 6
scale encoder differential error=15
encoder quadrature error = 16
scale encoder quadrature error = 17
standoff_error = 18
scale position error = 19
master enable turned off = 99</td></tr>
<tr><td>SV_STALL_AXIS</td><td>I32</td><td>Set whenever SV_STALL_ERROR is set
It will be set to the appropriate axis of the stall error. If the axis is not applicable then it will be set to 255.</td></tr>
<tr><td>SV_HSC_DRIVE_x_STATUS_y</td><td>I32</td><td>These variables contain every bit of the status packets sent back by AC1 drives. There are a total of 64 variables, eight for each of the eight possible AC1 drives. Only SV_HSC_DRIVE_x_STATUS_4 is documented here as it contains PLC related information of greatest importance.
SV_HSC_DRIVE_x_STATUS_4 bits:

bit00 FatalError
Indicates a serious error. The PLC program should monitor this bit on all AC1 drives and treat it as an emergency stop. This bit is cleared by the AC1 drive on the rising edge of SV_MASTER_ENABLE, but if the condition persists, the PLC program will not see a change. Therefore, if the FatalError bit is still on about a second after SV_MASTER_ENABLE has been turned back on (after a release of E-stop), then the fault should be thrown again. Other bits in this variable can be examined to determine the specific reason for the FatalError, but a PLC program does not need to handle every single error with a custom message as the state of all these bits can be viewed in the Log Options menu of CNC11, using the <F5> HSC function</td></tr>
</table>

		key. Also, this information is logged in the file hsc_status.txt and is included in a report. bit01 Warning Inidicates a warning condition. Other bits in this variable can be examined to determine the specific reason for the warning. bit02 ErrorUVWInvalid bit03 ErrorUVWBadTransition bit04 ErrorUVWBadSize bit05 EncoderOk bit06 QuadratureError Note that EncoderOk and QuadratureError bits are already monitored by MPU11 and if set, will generate the appropriate encoder differential or quadrature error messages. bit07 EncoderMismatch bit08 LineVoltageOn bit09 OvercurrrentHighSide bit10 OvercurrentLowSide bit11 OvervoltageMotor bit12 OvervoltageLine bit13 BrakeResistorMissing bit14 BrakeIGBTOpen bit15 MotorTemperatureSwitch bit16 HeatsinkTemperatureSwitch bit17 PlusLimit bit18 MinusLimit bit19 DriveShutdown bit20 BrakeOnTooMuch bit21 OvercurrentSensor bit22 WarningDriveHot bit23 ErrorDriveTooHot bit24 WarningMotorHot bit25 AccelTooGreat bit26 ADCOffsetOk bit27 ErrorMotorTooHot bit28 MoveSyncRunning bit29 StepRunning bit30 TuneRunning bit31 ErrorParameterChange
SV_SD_DRIVE_x_STATUS	I32	There are five of these variables that correspond to the status of legacy SD drives. Indicates a serious error. The PLC program should monitor this bit for all drives and treat it as an emergency stop condition. Other bits in this variable can be examined to determine the specific cause, but it is recommended that the PLC just echo this variable to W1-W44 so it can be viewed using line PLC diagnostics in CNC11.

		bit00 FatalErrorDetected bit01 ADMC300Online1 (should always be 1) bit02 ADMC300Online0 (should always be 0) bit03 IndexPulseDetected bit04 OvercurrentWarning bit05 PowerOn bit06 MoveRunning bit07 OvervoltageWarning bits08-15 unused bit16 ServoDriveErrorDetected bit17 OvervoltageDetected bit18 UndervoltageDetected bit19 CommutationEncoderError bit20 OvertemperatureDetected bit21 OvercurrentDetected bit22 DataTrasmitInError bit23 InvalidIndexPulse bit24 unused bit25 Lag (should not be reported) bit26 PositionError (should not be reported as MPU11 monitors and reports position errors) bit27 HighPowerNoMotion (should no be reported) bit28 EncoderBad (MPU11 monitors) bits29-31 unused
SV_DRIVE_TYPE_1-8	I32	The tyoe of drive connected to the axis: 0 = None 1 = Legacy DC 2 = DC3IOB 3 = SD 4 = AC1 5 = OPTIC4 6 = GPIO4D 7 = DC1 8 = ALLINONEDC 9 = Optic Direct 10 = RTK4 11 = ENCEXP 12 = OAK 13 = ACORN 14 = DRIVECOMM
SV_DRIVE_VERSION_1-8	I32	The drive firmware version
SV_?_AXIS_DRIVE_STATUS	I32	These nine system variables are mapped to the SV_DRIVE_STATUS_1-8 system variables according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_?_AXIS_DRIVE_STATUS	I32	These nine system variables are mapped to the SV_DRIVE_TYPE_1-8 system variables according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.

SV_AXIS_LABEL_1-8	I32	The uppercase ASCII character value assigned to the axis. Common values are: A = 65 B = 66 C = 67 N = 78 U = 85 V = 86 W = 87 X = 88 Y = 89 Z = 90
SV_SKINNING_DATA_W_1-12	I32	Generic 32-bit integer data that can be set by CNC Skinning API functions and read by the PLC,
SV_?_AXIS_DRIVE_NUMBER	I32	These nine system variables are mapped to machine parameter 300 (Axis 1 (?) Drive Number) through machine parameter 307 (Axis 8 (?) Drive Number). They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_PC_CURRENT_WCS	I32	The current work coordinate system (1-18) in effect by the CNC software.
SV_USB_MPG_AXIS_SELECT	I32	Value indicating the wireless USB MPG axis select switch 0 = Off 1-6 selected axis (X, Y, Z, 4th, 5th, 6th)
SV_USB_MPG_SCALE_SELECT	I32	Value indicating the wireless USB MPG scale selector knob 1 = x1 10 = x100 100 = x100 1000 = SPIN 10000 = FEED
SV_USB_MPG_BUTTON_STATE	I32	Conatins bits for wireless MPG USB button states bit 0 = Reset (Cycle Cancel) bit 1 = Feed Hold bit 2 = Cycle Start bit 3 = Jog Plus bit 4 = Jog Minus bit 5 = SPIN Auto/Man bit 6 = SPIN On/Off bit 7 = Macro 1 bit 8 = Macro 2 bit 9 = Macro 3 bit 10 = Macro 4 bit 11 = Tool Check bit 12 = Set Zero (=0)
SV_USB_MPG_ENCODER_WHEEL	I32	The MPG wheel position. This system variable counts up and down and does not roll over.
64-bit Integer System Variables		
SV_MPU11_ABS_POS_0 - SV_MPU11_ABS_POS_7	I64	The Absolute Position of the axis in encoder counts. This is the same value that is reported to CNC software and is viewed in PID screen as AbsPos.

SV_MPU11_EXPECTED_POS_0 - SV_MPU11_EXPECTED_POS_7	I64	The Expected Position of the axis in encoder counts. This is the current commanded position.
SV_STARTUP_TIME	I64	A 12 decimal digit integer representing the date and time the CNC software was started, The format is YYMMDDHHmmSS,.
32-bit Floating-point System Variables		
SV_MACHINE_PARAMETER_0-999	F32	Machine Parameter values as entered in CNC software. Note that these are 32-bit floating point values that have been converted from the 64-bit floating point values that are actually maintained by the CNC software, so there may be some loss of precision.
SV_PC_COMMANDED_SPINDLE_SPEED	F32	The commanded “S” value with Spindle Override factored in. Note that Parameters 65-67 for spindle range must be controlled in the PLC program.
SV_PC_CONFIG_MIN_SPINDLE_SPEED	F32	The minimum spindle speed from the control configuration.
SV_PC_CONFIG_MAX_SPINDLE_SPEED	F32	The maximum spindle speed from the control configuration.
SV_MEASURED_SPINDLE_SPEED	F32	The measured spindle speed in RPM, taking into account the SV_SPINDLE_MID_RANGE and SV_SPINDLE_LOW_RANGE system variable settings.
SV_SKINNING_DATA_FW_1-11	F32	Generic 32-bit floating point data that can be set by CNC Skinning API functions and read by the PLC,
SV_PC_MAXIMUM_CSS_SPEED	F32	For Lathe, the maximum constant surface speed set by a G50 command.
64-bit floating-point System Variables		
SV_SKINNING_DATA_DFW_1-11	F64	Generic 64-bit floating point data that can be set by CNC Skinning API functions and read by the PLC,

PLC Write-Controlled System Variables

Name	Type	Description
1-bit boolean system variables (MPU11 ---> PC)		
SV_SPINDLE_MID_RANGE	M	This must be set for Rigid Tapping and display of spindle speed to function correctly. In combination with SV_SPINDLE_LOW_RANGE below, up to four ranges are supported.
SV_SPINDLE_LOW_RANGE	M	This must be set for Rigid Tapping and display of spindle speed to function correctly. In combination with SV_SPINDLE_MID_RANGE above, up to four ranges are supported.
SV_PLC_FAULT	M	Obsolete. Do not use.

Name	Type	Description
SV_LUBRICANT_LOW	M	Obsolete. Do not use.
SV_DRIVE_FAULT	M	Obsolete. Do not use.
SV_SPINDLE_FAULT	M	Obsolete. Do not use.
SV_STOP	M	Bit that is SET bu the PLC program on a critical error or when E-Stop is pushed to singal to the CNC software and MPU to prevent axis motion, spindle commands and ATC changes. The bit should be RST when E-Stop is released and there are no other errors.
SV_PLC_OP_IN_PROGRESS	M	Obsolete. Do not use.
SV_PLC_FUNCTION_0	M	Invalid (Do not use)
SV_PLC_FUNCTION_1	M	Cycle Cancel
SV_PLC_FUNCTION_2	M	Cycle Start
SV_PLC_FUNCTION_3	M	Tool Check
SV_PLC_FUNCTION_4	M	Select Single Block
SV_PLC_FUNCTION_5	M	Select X1 Jog Mode
SV_PLC_FUNCTION_6	M	Select X10 Jog Mode
SV_PLC_FUNCTION_7	M	Select X100 Jog Mode
SV_PLC_FUNCTION_8	M	Not Used (formally User Jog Inc Mode)
SV_PLC_FUNCTION_9	M	Select Inc/Cont Jog Mode
SV_PLC_FUNCTION_10	M	Select Fast/Slow Jog Mode
SV_PLC_FUNCTION_11	M	Select Mpg Mode
SV_PLC_FUNCTION_12	M	Axis 1 + Jog
SV_PLC_FUNCTION_13	M	Axis 1 - Jog
SV_PLC_FUNCTION_14	M	Axis 2 + Jog
SV_PLC_FUNCTION_15	M	Axis 2 - Jog
SV_PLC_FUNCTION_16	M	Axis 3 + Jog
SV_PLC_FUNCTION_17	M	Axis 3 - Jog
SV_PLC_FUNCTION_18	M	Axis 4 + Jog
SV_PLC_FUNCTION_19	M	Axis 4 - Jog
SV_PLC_FUNCTION_20	M	Axis 5 + Jog
SV_PLC_FUNCTION_21	M	Axis 5 - Jog
SV_PLC_FUNCTION_22	M	Axis 6 + Jog
SV_PLC_FUNCTION_23	M	Axis 6 - Jog
SV_PLC_FUNCTION_24	M	Aux1
SV_PLC_FUNCTION_25	M	Aux2

Name	Type	Description
SV_PLC_FUNCTION_26	M	Aux3
SV_PLC_FUNCTION_27	M	Aux4
SV_PLC_FUNCTION_28	M	Aux5
SV_PLC_FUNCTION_29	M	Aux6
SV_PLC_FUNCTION_30	M	Aux7
SV_PLC_FUNCTION_31	M	Aux8
SV_PLC_FUNCTION_32	M	Aux9
SV_PLC_FUNCTION_33	M	Aux10
SV_PLC_FUNCTION_34	M	Select Rapid Override
SV_PLC_FUNCTION_35	M	Select Man or Auto Spindle Mode
SV_PLC_FUNCTION_36	M	Do not use
SV_PLC_FUNCTION_37	M	Spindle Start
SV_PLC_FUNCTION_38	M	Spindle Stop
SV_PLC_FUNCTION_39	M	Aux11
SV_PLC_FUNCTION_40	M	Aux12
SV_PLC_FUNCTION_41	M	Deprecated. Do not use
SV_PLC_FUNCTION_42	M	Deprecated. Do not use
SV_PLC_FUNCTION_43	M	Select Coolant Flood
SV_PLC_FUNCTION_44	M	Select Coolant Mist
SV_PLC_FUNCTION_45	M	Feed Hold
SV_PLC_FUNCTION_46 - SV_PLC_FUNCTION_97	M	Do not use.
SV_PLC_FUNCTION_98	M	Select Spindle CCW
SV_PLC_FUNCTION_99	M	Select Spindle CW
SV_PLC_FUNCTION_100 - SV_PLC_FUNCTION_103	M	Do not use.
SV_PLC_FUNCTION_104	M	Coolant Auto / Manual Mode
SV_PLC_FUNCTION_105	M	Do not use
SV_PLC_FUNCTION_106	M	Spindle Override +
SV_PLC_FUNCTION_107	M	Spindle Override -
SV_PLC_FUNCTION_108	M	Select Spindle Override / 100 %
SV_PLC_FUNCTION_109	M	Escape Key (sent to the PC)
SV_PLC_FUNCTION_110	M	Axis 7 Jog +
SV_PLC_FUNCTION_111	M	Axis 7 Jog -
SV_PLC_FUNCTION_112	M	Axis 8 Jog +
SV_PLC_FUNCTION_113	M	Axis 8 Jog -
SV_PLC_FUNCTION_114-	M	Unused

Name	Type	Description
SV_PLC_FUNCTION_127		
SV_PLC_OVERRIDE_CONTROL_FEEDRATE_OVERRIDE	M	1 = Feedrate can be changed from commanded value by Feedrate Override
SV_PLC_OVERRIDE_CONTROL_SPINDLE_OVERRIDE	M	1 = Spindle Speed can be changed from commanded value by Spindle Override
SV_PLC_OVERRIDE_CONTROL_FEEDHOLD	M	1 = Feedhold is allowed
SV_ENABLE_AXIS_1-SV_ENABLE_AXIS_8	M	Obsolete. Do not use.
SV_MASTER_ENABLE	M	PLC sets this bit to turn on the Master Enable to hardware devices (drives and PLCs)
SV_STALL_ERROR	M	1 = MPU11 detected an error that the PLC program should handle. See SV_STALL_REASON and SV_STALL_AXIS for further information on the specific cause. **PLC program should turn off all enables including the Master Enable when this error occurs.** The PLC Program should RST this bit if the error occurred and E-Stop is pushed in. (ie clear on E-Stop)
SV_MPG_1_ENABLED	M	This MPG is enabled. This bit switches between MPG mode and Vector controlled mode for all axes in this group. When this is enabled the MPU11 will not process motion vectors from the PC or allow Jogging.
SV_MPG_1_WINDUP_MODE	M	The MPG is in windup mode. This means the MPG will move the total distance commanded by the MPG encoder input. This mode is typically enabled for x1 and x10 mode. This mode should be disabled for x100 mode. When windup mode is disabled the MPG will try to keep up, but will go off position if the MPG encoder counts too fast. When this happens CNC11 will display the message "MPG moving too fast".
SV_MPG_1_OFFSET_MODE	M	The MPG is in Offset Mode. The MPG movement will be added to the current Expected Position instead of setting the Expected Position. In this mode the MPG is independent and is allowed to command motion while vectors are being processed.
SV_MPG_1_PLC_MPG_MODE	M	The MPG is in PLC Controlled Mode. The MPG encoder input will be read from SV_MPG_1_OFFSET instead of the actual encoder input. The plc program can change SV_MPG_1_OFFSET which will cause the mpg axis to move.
SV_MPG_2_ENABLED	M	See SV_MPG_1_ENABLED above.
SV_MPG_2_WINDUP_MODE	M	See SV_MPG_1_WINDUP_MODE above.
SV_MPG_2_OFFSET_MODE	M	See SV_MPG_1_OFFSET_MODE above.
SV_MPG_2_PLC_MPG_MODE	M	See SV_MPG_1_PLC_MPG_MODE above.

Name	Type	Description
SV_MPG_3_ENABLED	M	See SV_MPG_1_ENABLED above.
SV_MPG_3_WINDUP_MODE	M	See SV_MPG_1_WINDUP_MODE above.
SV_MPG_3_OFFSET_MODE	M	See SV_MPG_1_OFFSET_MODE above.
SV_MPG_3_PLC_MPG_MODE	M	See SV_MPG_1_PLC_MPG_MODE above.
SV_TRIGGER_PLOT_DUMP	M	This bit is intended for internal debugging purposes. When set, it will start a debug dump to be sent to CNC software. After CNC software receives all the debug data, it will launch the plot.exe program to display it. Without custom built CNC software, the debug dump has no useful data.
SV_RESET_PLC_STATS_MIN_MAX	M	Resets the current Minimum/Maximum PLC executor statistics which are displayed on the PLC diagnostic screen (ALT-I)
SV_RESET_PLC_STATS_AVG	M	Resets the current Average PLC executor statistics which are displayed on the PLC diagnostic screen (ALT-I)
SV_SPINDLE_RPM_MODE	M	Used for C Axis Lathe. When this bit is set, the mpu11 will send the current value of SV_SPINDLE_DAC as the PID output to the drive for the last axis configured as a “C axis” When this mode is active, full power without motion and position errors are disabled for the axis.
SV_SCALE_INHIBIT_AXIS_1-SV_SCALE_INHIBIT_AXIS_8	M	When set, scale compensation for the axis is disabled until the bit is reset. When the scale compensation is disabled in this manner it will undo any previous corrections. The DRO will show the absolute position of the motor encoder when the scale is disabled.
SV_ENABLE_IO_OVERRIDE	M	When set by the PLC prgram, CNC software will indirectly allow the inversion and forcing of PLC bits through the live PLC display (ALT-I on the main screen) by manipulating machine parameters 911-939, which the PLC program can use directly to set system variables that actually perform the state forcing and inversion.
SV_RPM_MODE_DIRECTION_1-8	M	Used with AC1 drives to set the direction for RPM mode.
SV_RPM_MODE_ENABLE_1-8	M	Used with AC1 drives to enable RPM mode.
SV_RPM_MODE_AXIS_ENABLE_1-8	M	Used with AC1 drives to enable the axis in RPM mode.
SV_DISABLE_STALL_DETECTION	M	When SET, will disable the detection of some stall errors, such as position errors and full power without motion errors.
SV_PLC_SET_AXIS_n_PART_ZERO	M	When SET by a PLC program and the control is not running a job and at the main screen, will request CNC software to set the Part Zero for the axis (n = 1-8).
32-bit signed integer system variables (MPU11 ---> PC)		
SV_PLC_FAULT_STATUS	I32	Bitwise parameter of certain fault conditions: 0x00000001 DIV_BY_ZERO 0x00000002 OUT_OF_BOUNDS 0x00000004 INVALID_OPCODE See Internal PLC Fault Checking.

Name	Type	Description
SV_PLC_FAULT_ADDRESS	I32	Exact address in the PLC program where the above fault occurred. Should be monitored as above. See Internal PLC Fault Checking.
SV_PLC_SPINDLE_SPEED		If parameter 78 is not set to display the actual spindle speed, this value is the current spindle speed that is displayed on the screen.
SV_STOP_REASON	I32	When motion stops a reason is given and that reason is stored here. Do not use at this time.
SV_PLC_CAROUSEL_POSITION	I32	Bin position that the carousel is at. It is critical that the Carousel not be allowed to turn unless CNC11 is running. Check the SV for Software running.
SV_PLC_FEEDRATE_KNOB	I32	The feedrate knob as the PLC would have CNC11 see.
SV_PLC_SPINDLE_KNOB	I32	Spindle Speed override percentage sent to the PLC
SV_SMSG_D_ARG_1 - SV_SMSG_D_ARG_9		Reserved for future use. Do not use.
SV_PLC_DEBOUNCE_1 to _64	I32	First 240 PLC inputs debounce configuration words
SV_JOG_LINK_DEBOUNCE_1 to _32	I32	Jog Panel Input debounce configuration words
SV_LOCAL_DEBOUNCE_1 to _13	I32	MPU11 Onboard/Local Inputs Debounce configuration
SV_MPG_1_AXIS_SELECT	I32	The current selected axis for the first mpg (1-8)
SV_MPG_1_MULTIPLIER	I32	The mpg multiplier value normally (1, 10, 100)
SV_MPG_1_PLC_OFFSET	I32	MPG Offset, for PLC controlled MPG Inputs. This allows the PLC to control the MPG through an Analog to Digital Input rather than an encoder. This mode is enabled when the PLCMPGmode bit is set.
SV_MPG_2_AXIS_SELECT	I32	The current selected axis for the second mpg (1-8)
SV_MPG_2_MULTIPLIER	I32	The mpg multiplier value normally (1, 10, 100)
SV_MPG_2_PLC_OFFSET	I32	MPG Offset, for PLC controlled MPG Inputs. This allows the PLC to control the MPG through an Analog to Digital Input rather than an encoder. This mode is enabled when the PLCMPGmode bit is set.
SV_MPG_3_AXIS_SELECT	I32	The current selected axis for the third mpg (1-8)
SV_MPG_3_MULTIPLIER	I32	The mpg multiplier value normally (1, 10, 100)
SV_MPG_3_PLC_OFFSET	I32	MPG Offset, for PLC controlled MPG Inputs. This allows the PLC to control the MPG through an Analog to Digital Input rather than an encoder. This mode is enabled when the PLCMPGmode bit is set.
SV_DRIVE_CONTROL_1 to _8	I32	High Speed Drive Control word (Lower 16 Bits) see Drive documentation **DC3IOB / ALLIN1DC Control**

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>0-14</td><td>Reserved</td></tr>
<tr><td>15*</td><td>Axis Enable</td></tr>
</table>
<table>
<tr><th colspan="2">DC1 Control</th></tr>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>0-14</td><td>Reserved</td></tr>
<tr><td>15*</td><td>Axis Enable</td></tr>
</table>
<table>
<tr><th colspan="2">OPTIC4 Control</th></tr>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>0-12</td><td>Reserved</td></tr>
<tr><td>13**</td><td>Tach. Direction Inversion</td></tr>
<tr><td>14</td><td>Auxiliary Output</td></tr>
<tr><td>15*</td><td>Axis Enable</td></tr>
</table>
Notes:
*MPU11 set this bit and will overwrite any attempt by the PLC to set it.
**Set parameters 200-207 to negative values rather than trying to change bit 13
</td></tr>
<tr><td>SV_SPINDLE_DAC</td><td>I32</td><td>Used for C Axis Lathe. When this bit is set, the MPU will send the current value of SV_SPINDLE_DAC as the PID output to the drive for the last axis configured as a “C axis” When this mode is active, full power without motion and position errors are disabled for the axis.</td></tr>
<tr><td>SV_NV_W1-
SV_NV_W10</td><td>I32</td><td>Nonvolatile memory.
These system variables can be used in the same manner as other I32 variables in the PLC program. Their values are retained even when power is off. A PLC programmer can expect any changes to these system variables to be written to non-volatile memory within 2ms.</td></tr>
<tr><td>SV_SYS_COMMAND</td><td>I32</td><td>Setting this value to a non-zero, positive number, will cause the CNC software to launch a process and try to execute the Windows batch file named plc_system_command_n.bat, where n is the number the SV_SYS_COMMAND is set to.</td></tr>
<tr><td>SV_INVERT_INP1_16_BITS
SV_INVERT_INP17_32_BITS
SV_INVERT_INP33_48_BITS
SV_INVERT_INP49_64_BITS
SV_INVERT_INP65_80_BITS</td><td>I32</td><td>The lower 16 bits of each of these system variables is used to invert the indicated inputs, with the least significant bit mapped to the lower INP bit and the most significant bit mapped to the higher numbered INP.</td></tr>
<tr><td>SV_FORCE_INP1_16_BITS
SV_FORCE_INP17_32_BITS</td><td>I32</td><td>The lower 16 bits of each of these system variables is used to force the state of the indicated inputs, with the least</td></tr>
</table>

Name	Type	Description
SV_FORCE_INP33_48_BITS SV_FORCE_INP49_64_BITS SV_FORCE_INP65_80_BITS		significant bit mapped to the lower INP bit and the most significant bit mapped to the higher numbered INP. The input is forced on if the equivalent bit in the SV_INVERT_INP* system variable is clear and forced off if the equivalent bit in SV_INVERT_INP* system variable is set.
SV_FORCE_ON_OUT1_16_BITS SV_FORCE_ON_OUT17_32_BITS SV_FORCE_ON_OUT33_48_BITS SV_FORCE_ON_OUT49_64_BITS SV_FORCE_ON_OUT65_80_BITS	I32	The lower 16 bits of each of these system variables is used to force ON the indicated outputs, with the least significant bit mapped to the lower OUT bit and the most significant bit mapped to the higher numbered OUT bit. Note that the equivalent bit in the SV_FORCE_OFF* (see below) system variable must be clear.
SV_FORCE_OFF_OUT1_16_BITS SV_FORCE_OFF_OUT17_32_BITS SV_FORCE_OFF_OUT33_48_BITS SV_FORCE_OFF_OUT49_64_BITS SV_FORCE_OFF_OUT65_80_BITS	I32	The lower 16 bits of each of these system variables is used to force OFF the indicated outputs, with the least significant bit mapped to the lower OUT bit and the most significant bit mapped to the higher numbered OUT bit. Note that the equivalent bit in the SV_FORCE_ON* (see above) system variable must be clear.
SV_?_AXIS_DRIVE_CONTROL	I32	These nine system variables are mapped to the SV_DRIVE_CONTROL_1-8 system variables according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_FORCE_ON_MEM1_16_BITS SV_FORCE_ON_MEM17_32_BITS SV_FORCE_ON_MEM33_48_BITS SV_FORCE_ON_MEM33_48_BITS SV_FORCE_ON_MEM65_80_BITS	I32	The lower 16 bits of each of these system variables is used to force ON the indicated memory bits, with the least significant bit mapped to the lower MEM bit and the most significant bit mapped to the higher numbered MEM bit. Note that the equivalent bit in the SV_FORCE_OFF_MEM* (see below) system variable must be clear. Note that the forced state is implemented between PLC program passes, i.e., if the PLC program changes a forced memory bit during execution of a program, it will update the state of the memory bit.
SV_FORCE_OFF_MEM1_16_BITS SV_FORCE_OFF_MEM17_32_BITS SV_FORCE_OFF_MEM33_48_BITS SV_FORCE_OFF_MEM49_64_BITS SV_FORCE_OFF_MEM65_80_BITS	I32	The lower 16 bits of each of these system variables is used to force OFF the indicated memory bit, with the least significant bit mapped to the lower MEM bit and the most significant bit mapped to the higher numbered MEM bit. Note that the equivalent bit in the SV_FORCE_ON_MEM* (see above) system variable must be clear. Note that the forced state is implemented between PLC program passes, i.e., if the PLC program changes a forced memory bit during execution of a program, it will update the state of the memory bit.
SV_SYS_MACRO	I32	Setting this system variable to a non-zero value while CNC software is at the main menu, will cause the CNC software to try to load and run the G-code program named plcmacroN.mac located in the system directory. For example, if on a Mill system, setting SV_SYS_MACRO = 3 will attempt to run the G-code file named

Name	Type	Description
		\cncm\system\plcmacro3.mac. This can be set to a negative value.
32-bit floating point system variables (MPU11 ---> PC)		
SV_PLC_FEEDRATE_OVERRIDE	F32	The Feedrate factor (for MPU11 motion control). Values can range from 0 to 2.0. A value of 1.0 results in no change to the G-code programmed or Jog Rate value. Note that the MPU11 will cap the Feedrate to the maximum set in Machine Setup. Care must be taken to never apply a negative value.
SV_SMSG_F_ARG_1- SV_SMSG_F_ARG_9	F32	Reserved for future use. Do not use.
SV_NV_FW1- SV_NV_FW10	F32	Nonvolatile memory. These system variables can be used in the same manner as other F32 variables in the PLC program. Their values are retained even when power is off. A PLC programmer can expect any changes to these system variables to be written to non-volatile memory within 2ms.
SV_METER_1 - 16	F32	These system variables are set by the PLC program to values between -100.0 and 100.0 for use of the CNC software to display as a meter in the DRO. Only the first eight are used, the meters 9-16 are reserved for future implementation.
SV_RPM_MODE_SPEED_REQUEST_1-8	F32	For AC1 drives, the requested RPM speed
SV_?_AXIS_METER	F32	These nine system variables are mapped to the SV_METER_1-8 system variables according to the axis labels (?) XYZABCUVW. They are intended to be a convenience when writing PLC programs so that they can handle axis changes more easily.
SV_SPINDLE_METER	F32	This value is mapped to one of the SV_METER_1 - 16 system variables, in particular the one that corresponds to the spindle axis.

Appendix E: PLC I/O Location

This chapter lays out the kinds of I/O available in general and specifically on the different PLC boards available.

Input Types

The Input tables in each of the manuals for these drives contain several headings that may need clarification as to their exact purpose. The first is Input and this directly corresponds to the value used in the PLC Program. Next, a heading called Type corresponds to the type of Input that can be wired in. Within this column there are several labels which need to be explained. Configurable indicates the voltage required for the Input can be customized. Typical Centroid 5, 12 and 24VDC SIP resistors can be used. 12VDC Optoisolated means that it must be 12VDC powering the Input and that it has signal isolation to prevent electrical noise getting onto the circuit board. 5VDC only means that 5V is the maximum signal voltage that can be wired to this input. It also does not provide any electrical isolation. Sourcing means that the input must be connected to the Input common to turn on. Lastly, ADC is an Analog-to-Digital Conversion. The ALLIN1DC currently is the only standard PLC that has this functionality available for use in the PLC program, though there is also an ADD4AD4DA expansion board that has four DAC outputs and four ADC inputs.

Output Types

The Output tables in the manuals also have similar columns including Type which again refers to the type of Output that can be wired. The first types are Relay Outputs that may or may not be fused depending on the specific Output. There are Single Throw and Double Throw relays. The Single Throw has two pins associated with the output and there is continuity between them only when the Output is SET. The Double Throw has three pins which are a so-called Output Common, Normally Open and Normally closed. There is continuity between the Normally Closed pin and the common when the Output is RST and continuity between the Normally Open and common when the Output is SET. The next type is a DAC Output. This is a Digital-to-Analog Converting output. The Bit number specified defines the range of discreet values that can be commanded. Open Collector is an output, usually limited to 5VDC that must be wired to an external relay for anything other than low current signals. It is much safer to use a relay and another power supply than assume the output will not draw too much current and damage the drive.

Warning: Always review the specific board manual that came with your system before changing any wiring.

ALLIN1DC

The ALLIN1DC has 16 inputs and 9 outputs that may be used in the PLC program. The MPG header has 14 Inputs and 3 outputs that can be used as General I/O in a customized program. Typically it is better to get a PLC1616ADD card rather than using the MPG header,

however. The first six inputs are dedicated to Limit switch wiring and cannot be used for general purpose Inputs. Moreover the first two inputs are tied to the first axis, the 3rd and 4th inputs to the second axis and likewise for the third. The firmware on the drive shuts down motion in the direction of the Input that is tripped, regardless of CNC11 software Limit Switch setup. There is also an analog Input and Output with 12-bit resolution. For the schematic and I/O map, reference the ALLIN1DC manual on the Dealer Support site or with the literature you received in your kit.

DC3IOB

The DC3IOB has 30 Inputs and 31 Outputs usable in the PLC program. There is one analog Output used for Spindle control. The first six inputs are dedicated to Limit switch wiring and cannot be used for general purpose Inputs. Moreover the first two inputs are tied to the first axis, the 3rd and 4th inputs to the second axis and likewise for the third. The firmware on the drive shuts down motion in the direction of the Input that is tripped, regardless of CNC11 software Limit Switch setup. For the schematic and I/O map, reference the DC3IOB manual on the Dealer Support site or with the literature you received in your kit.

GPIO4D

The DC3IOB has 30 Inputs and 31 Outputs usable in the PLC program. There is one analog Output used for Spindle control and four analog Outputs that are not PLC accessible that are used for 3rd party drive control. The first six Inputs are typically dedicated to Limit switch wiring, but can be used for general purpose Inputs. Inputs and Outputs 17-20 are dedicated to the axis enable and fault inputs respectively. For the schematic and I/O map, reference the DC3IOB manual on the Dealer Support site or with the literature you received in your kit.

PLC Expansion

All of the PLCs in this chapter can expand the Inputs and Outputs with miniPLC expansion boards. There are differing numbers of available ports so be sure to check the board manual to determine the total I/O range. There are 16 Debouncable Input and Output slots each capable of controlling 16 individual I/O for each board type. Each new board starts at a multiple of 16. There are 32 non-Debouncable Input and Output slots with 16 controllable bits which may be grouped as individual I/O or combined to produce one DAC. The slot numbering starts at 0 and goes to 47.

Once the boards are plugged into the PLC and powered as directed by the main PLC board's manual, power on the system and look in the mpu_info.txt file in the cncm or cnct directory. This gives a breakdown of the Inputs and Outputs and what board is associated with them.

The order that the I/O appears is directly related to which ADD port it is plugged into. ADD1 is first and ADD4 is last. Depending on which is the master PLC the ADD boards come in on the next available slot.

See the individual board manuals for more in-depth documentation of PLC Expansion with examples. MiniPLC devices must be plugged in when power is off on the drive to which they

are connecting.

This table shows what slot maps to what Input Range. To figure out where the I/O should start for a certain slot that is not shown here, multiply the slot number by 16 and add 1 to get the starting Input or Output number. Add 15 to that number to get the last number in the slot.

Slot Number	Input/Output Numbers
0	1-16
1	17-32
2	33-48
4	65-80
14	225-240
15	241-256
20	321-336
47	753-768

ALLIN1DC

The expansion for the ALLIN1DC starts at Slot 1 and 16 because the onboard I/O take up the first slot of Debounced and non-Debounced I/O.

DC3IOB

The expansion for the DC3IOB starts at Slots 2 and 15 for the Inputs and Slots 4 and 15 for the Outputs.

GPIO4D

The expansion for the GPIO4D starts at Slots 2 and 15 for the Inputs and Slots 2 and 20 for the Outputs.

Legacy IO2 (PLCIO2, RTK3) and Legacy RTK2 (RTK2, PLC5/15, PLC3/3)

Since CNC11 v3.10, there is support for programming legacy PLCs. To program a legacy PLC with an MPU11, the MPU11 must be fitted with the LEGACYADD daughter card, and the Control Configuration PLC type set to “IO2” or “RTK2”.

For PLCIO2 and RTK3, INP59-INP62 have been moved to INP34-INP37 and OUT59-OUT62 have been moved to OUT49-OUT52.

Appendix F: G/M-Code User/System Variable

This table is copied as a reference from Chapter 11 of the CNC11 User's Manual. In cases of discrepancy the User's Manual version shall be the final word. 50000 and on refer to the same things as are used in the PLC program, which is why this table is included.

Index	Description	Returns	R/W
1-3	Macro arguments A-C	The floating point value if defined by a G65 call, 0.0 otherwise. These can be used as private, local variables in any program or subprogram except in custom macro M functions, where they are passed by reference. (See examples.)	R/W
4-6	Macro arguments I-K (1st set)		R/W
7-9	Macro arguments D-F or 2nd set of I-K		R/W
10	3rd I (G is invalid)		R/W
11	Macro argument H or 3rd J		R/W
12	3rd K (L is invalid)		R/W
13	Macro argument M or 4th I		R/W
14	4th J (N is invalid)		R/W
15	4th K (O is invalid)		R/W
16	5th I (P is invalid)		R/W
17-18	Macro argument Q-R or 5th J-K		R/W
19-21	Macro arguments R-T or 6th set of I-K		R/W
22-24	Macro arguments U-W or 7th set of I-K		R/W
25-27	Macro arguments X-Z or 8th set of I-K		R/W
28-30	9th set of I-K		R/W
31-33	10th set of I-K		R/W
100 - 149	User variables	Floating-point value. Initialized to 0.0 at start of job processing	R/W
150 - 159	Nonvolatile user variables	Floating-point value saved in cncm.job file.	R/W
300-399	User string variables. These variables retain their values until the CNC software is exited	String Literal	R/W
2400, 2401-2418	Active WCS, WCS #1-18 CSR angles	Floating point value	R/W
2500, 2501-2518	Active WCS, WCS #1-18 Axis 1 values		R/W
2600, 2601-2618	Active WCS, WCS #1-18 Axis 2 values		R/W
2700, 2701-2718	Active WCS, WCS #1-18 Axis 3 values		R/W
2800, 2801-2818	Active WCS, WCS #1-18 Axis 4 values		R/W
2900, 2901-2918	Active WCS, WCS #1-18 Axis 5 values		R/W
3000, 3001-3018	Active WCS, WCS #1-18 Axis 6 values		R/W
3100, 3101-3118	Active WCS, WCS #1-18 Axis 7 values		R/W
3200, 3201-3218	Active WCS, WCS #1-18 Axis 8 values		R/W
3901	Parts Cut (Part #)		R/W

3902	Parts Required (Part Cnt)		R/W
4001	Move mode	0.0 (rapid) or 1.0 (feed)	R
4003	Positioning mode	90.0 (abs) or 91.0 (inc)	R
4006	Units of measure	20.0 (inches) or 21.0 (metric)	R
4014	WCS	54.0-71.0 (WCS#1-18)	R
4109	Feedrate (F)	Floating point value	R
4119	Spindle Speed (S)		R
4120	Tool Number (T)		R
4121	Current height offset number (H)		R
4122	Current diameter offset number (D, mill only)		R
4201	Job processing state	0 = normal, 1 = graph	R
4202	Search mode (0 = search mode off)	0 = search mode off	R
5021-5028	Machine Position (X=5021, Y=5022, etc.)	Floating point value	R
5041-5048	Current Position (X=5041, Y=5042, etc.)		R
9000-9399	Parameter values 0 - 399	See Chapter 14	R/W
9900-9999	Parameter values 900-999		
10000	Mill: Height offset amount, active H	Floating point value	R/W
10001-10200	Mill: Height offset amount, H001 - H200	Floating point value	R/W
11000	Mill: Diameter offset amount, active D	Floating point value	R/W
11001-11200	Mill: Diameter offset amount, D001 - D200	Floating point value	R/W
12000	Mill: Tool H number, active tool (T)	0 - 200	R/W
12001-12200	Mill: Tool H number, tools 1 - 200	0 - 200	R/W
13000	Mill: Tool D number, active tool (T)	0 - 200	R/W
13001-13200	Mill: Tool D number, tools 1 - 200	0 - 200	R/W
14000	Mill: Tool coolant, active tool (T)	7, 8, 9	R/W
14001-14200	Mill: Tool coolant, tools 1 - 200	7, 8, 9	R/W
15000	Mill: Tool spindle direction, active tool (T)	3, 4, 5	R/W
15001-15200	Mill: Tool spindle direction, tools 1 - 200	3, 4, 5	R/W
16000	Mill: Tool spindle speed, active tool (T)	Floating point value	R/W
16001-16200	Mill: Tool spindle speed, tools 1 - 200	Floating point value	R/W
17000	Mill: Tool bin number, active tool (T)	Floating point value	R/W
17001-17200	Mill: Tool bin number, tools 1 - 200	Floating point value	R/W
18000	Mill: Tool putback, active tool (T)	Floating point value	R/W
18001-18200	Mill: Tool putback, tools 1 - 200	Floating point value	R/W
20001-20008	max_rate for axes 1-8		R
20101-20108	label for axes 1-8		R
20201-20208	slow_jog for axes 1-8		R
20301-20308	fast_jog for axes 1-8		R
20401-20408	screw_pitch for axes 1-8		R/W
20501-20508	lash_comp for axes 1-8		R
20601-20608	counts_per_unit for axes 1-8		R
20701-20708	accel_time for axes 1-8		R

20801-20808	deadstart_velocity for axes 1-8		R
20901-20908	delta_vmax for axes 1-8		R
21001-21008	counts_per_turn for axes 1-8		R
21101-21108	minus_limit for axes 1-8		R
21201-21208	plus_limit for axes 1-8		R
21301-21308	minus_home for axes 1-8		R
21401-21408	plus_home for axes 1-8		R
21501-21508	reversed for axes 1-8		R
21601-21608	laser_comp for axes 1-8		R
21701-21708	proportional for axes 1-8		R
21801-21808	integration_limit for axes 1-8		R
21901-21908	kg for axes 1-8		R
22001-22008	integral for axes 1-8		R
22101-22108	kv1 for axes 1-8		R
22201-22208	derivative for axes 1-8		R
22301-22308	ka for axes 1-8		R
22401-22408	num_motor_poles for axes 1-8		R
22501-22508	drive_current for axes 1-8		R
22601-22608	drive_offset_angle for axes 1-8		R
22701-22708	pwm_kp for axes 1-8		R
22801-22808	pwm_ki for axes 1-8		R
22901-22908	pwm_kd for axes 1-8		R
23001-23008	abrupt_kp for axes 1-8		R
23101-23108	feed_forward_kp for axes 1-8		R
23201-23208	max_error (PID) for axes 1-8		R
23301-23308	min_error (PID) for axes 1-8		R
23401-23408	at_index_pulse for axes 1-8		R
23501-23508	travel_minus for axes 1-8		R/W
23601-23608	travel_plus for axes 1-8		R/W
23701-23708	axis_home_set for axes 1-8		R
23801-23808	abs_position (in encoder counts) for axes 1-8		R
23901-23908	PID_out for axes 1-8		R
24001-24008	reference set for axes 1-8		R
24101-24108	Axis reference value for axes 1-8		R
24201-24208	tilt table level offsets for axes 1-8		R
24301-24308	dsp positions for axes 1-8		R
24401-24408	abs_position (same as 23801-23808)		
24501-24508	dsp position in local coordinates		
24601-24608	Local probing +limit position		
24701-24708	Local probing -limit position		
24801-24808	Probe stylus compensation amount		
24901-24908	Servo controlled axis indicator		
25000	DRO_display_units		R
25001	default_units_of_measure		R
25002	PLC_type		R
25003	console_type		R
25004	jog_panel_optional		R
25005	min_spin_high		R
25006	max_spin_high		R
25007	home_at_powerup		R
25008	screen_blank_time		R

25009	Displayed / Calculated spindle speed. If parameter 178 =1 and spindle encoder is mounted.		R
25010	current spindle position (in counts)		R
25011	dsp_time (in seconds)		R
25012	time (in seconds)		R
25013	clear max/min PID errors		R
25014	software type (Mill/Lathe)		R
25015	feedrate override		R
25016	spindle override		R
25017	OS	Windows/LINUX = 2; other OS = 1.0	R
25018	CNC series number (11 for CNC11)		
25019	Software version number		
25020	Software Beta revision number		
25021	Digitizing boundary hit	1 = hit, 0 = no hit	
25022	Last M115/116/125/126 probe trip	1 = tripped	
26001-26008	Dsp mechanical machine positions		
26101-26108	Dsp mecanical local positions		
26201-26208	Local +travel limit positions		
26301-26308	Local -travel limit positions		
26401-26404	Return Point 1 – 4 Axis 1 Values		
26501-26504	Return Point 1 – 4 Axis 2 Values		
26601-26604	Return Point 1 – 4 Axis 3 Values		
26701-26704	Return Point 1 – 4 Axis 4 Values		
26801-26804	Return Point 1 – 4 Axis 5 Values		
26901-26904	Return Point 1 – 4 Axis 6 Values		
27001-27004	Return Point 1 – 4 Axis 7 Values		
27101-27104	Return Point 1 – 4 Axis 8 Values		
29000-31999	User variables. These variables retain their values until the CNC software is exited.	Floating point value	R/W
50001-51312	PLC Inputs 1-1312	Jog Panel is on INP1057-1312	R
60001-61312	PLC Outputs 1-1312	Jog Panel is on OUT1057-1312	R
70001-71024	PLC Memory Bits 1-1024		R
80001-89999	Reserved		R
90001-90064	Timer 1-64 status bits		R
91001-91064	Reserved		R
92001-92064	Reserved		R
93001-93256	Stage 1-256 status bits		R
94001-94256	Fast Stage 1-256 status bits		R
95001-95256	Reserved		R
96001-96044	W1-W44 (32-bit signed integers)		R
97001-97022*	DW1-DW22 (64-bit signed integers)		R
98001-98044	FW1-FW44 (32-bit floats)		R
99001-99022	DFW1-DFW22 (64-bit floats)		R

* Since user or system variables are turned into (double) floating point values when referenced in an M- or G-code program, the 64-bit integer values lose precision when they exceed 2^{53} (9,007,199,254,740,992).

Appendix G: What's New in CNC11

This chapter covers major differences between CNC10 PLC programs and CNC11 PLC programs.

There is Only One PLC Program

In CNC11 there is only one PLC program source file and there is a new compiler for that program. The new compiler is called mpucomp. The program should be compiled in the cncm or cnct directory which is how CNC10 PLC programs are complied as well. Because everything is taken care of in one file, there is no longer the confusion of what happens in what file, however the file does tend to be much larger than in CNC10 due to the requirement handed to the PLC program for taking care of many basic operations. Essentially, if the PLC program does not tell CNC11 about something I/O related, it does not know about it, period.

The PLC Program has the Final Word

CNC11 was designed to allow the PLC program to have the final say in all things that affect motion and I/O. The exception is that the PLC program cannot enable the motors, only alarm if there is a fault. Feedrate Override is a good example of this paradigm. The PLC program reads the Feedrate knob and tells CNC11 what it found, then CNC11 passes down what it thinks the feedrate override value should be for Parameter 78 compliance. Again the PLC program can change the value before sending it to the Motion Controller, though typically this is not done because it can break intended functionality.

Spindle Speed DAC Command

In CNC10 the Spindle Gear range was taken care of before the analog voltage was sent out by software. In CNC11 the S command spindle speed is sent down exactly as entered in G-Code and the PLC program must deal with the DAC output and tell CNC11 what it sent out.

Direct Control of and Responsibility for Jogging

In CNC11 the PLC program sees every Keyboard Key and Jog Panel Key press and must tell CNC11 that it has occurred. For example, this allows the PLC program to do such things as wait for a jog key to be pushed for some time before allowing motion or not reporting that the key has not been pushed until some other condition is met.

Compiler/Language Differences

The following sections represent the differences between CNC10 and CNC11 PLC Programming paradigms and keywords.

PLC Inputs and Outputs

In CNC11 all inputs and outputs are only physical inputs and outputs. Inputs can only be read or inverted by the PLC program. The M94/M95 input bits, Spindle Range input bits, and status outputs from CNC10, for example, are now System Variables.

Green == 1 == closed == SET

Any Bit Variable in CNC11 is Green on the PLC Diagnostic screen if it is SET or closed and Red otherwise. This is true for Normally Closed or Normally Open Variables.

Keywords Removed

In CNC10 there were several keywords (LDT, LTS, LMT, LCP) which have been removed in CNC11. They are all replaced by System Variables.

Timers

Timer units in CNC10 were 10 ms increments, but now in CNC11 the units are 1 ms This means that counting to 1000 in CNC10 takes 10 seconds whereas in CNC11 it would take 1 second.

Stages

In CNC11 when a Stage is reset all the variables inside retain their value, whereas in CNC10 all the variables are reset. Keep this in mind so that when a Jump is called or the Stage is reset manually, variables that you want off should be turned off explicitly before leaving the Stage. Be aware especially of One-Shots because in CNC10 when Jumping from one Stage to another all Coiled Variables are turned off whereas in CNC11 they are not. Read more on Stages in CNC11 here.

PLC Program Should Detect All Faults

In CNC10 PLC, Drive. Lube and Spindle Faults were detected by looking at what has become SV_STOP in CNC11 and printing the correct message based on several conditions. This is not allowed by convention in CNC11. The PLC programmer should take care of these errors with the PLC Messaging functionality.

Appendix H: Definitions of Unobvious Words

Bit

A Binary piece of information that can be on or off. For example, Inputs are Bits.

Integer Number

A positive or negative whole number. Examples include 15, -10, 0 and 1309921.

Floating-point Number

A positive or negative number with fractional components expressed to the right of a decimal point. Examples include 2.1453, -15.2, 0.0, and 104.999999.

Range

The span of numbers that can be represented by a variable.

Precision

A measure of the number of bits a variable can express. This directly relates to how big a number an integer variable can represent and how precise a number a floating-point variable can represent. A more precise (higher bit count) floating-point number is less prone to round-off error when comparing very large numbers with very small (ex: 1454394783237.35 – 0.00000001). More bits means longer processing time, so unless extreme accuracy is needed the 32-bit version of variables should be used.

Data Type

A classification for variables that defines what kind of information they can hold and the range of values they can represent. Further explanation for all the available data types can be found here.

Define/Declare

Create an easier to understand identifier for any data type. These are conveniences for the programmer to allow easier understanding of logic in a PLC program. Data types can be set or checked directly, but a warning is issued by the compiler because it is considered bad practice and can cause confusion. Examples are:

```
Estop     IS INP11                ;input 11 on DC systems is typically E-Stop
Shift_Key IS SV_PC_KEYBOARD_KEY_74 ;pushing the shift key results in this System
                                    ;Variable being set high.
```

Variable

Any of the data types or definitions of data types used in the PLC program that can change its value while CNC11 and the PLC program is running. Examples include inputs, outputs, stages, renamed inputs by definition, etc.

Constant

Any literal number or definition of a number. Typically definitions of constants are written in all caps to differentiate from variables. Examples include:

```
10
```

```
2.5
```

ARM_FAULT_CODE IS 2561